

HP OpenVMS

HP C ランタイム・ライブラリ・ リファレンス・マニュアル(下巻)

注文番号: 5991-6626

2006 年 10 月

本書は、リファレンス・セクションで OpenVMS システム用の HP C ランタイム・ライブラリの関数について説明します。

改訂 / 更新情報:

本書は『HP C Run-Time Library Reference Manual for OpenVMS Systems』の改訂版です。

ソフトウェア・バージョン:

HP OpenVMS I64 V8.2
HP OpenVMS Alpha V8.2
OpenVMS VAX V7.3

日本ヒューレット・パッカード株式会社

© Copyright 2006 Hewlett-Packard Development Company, L.P.

本書の著作権は Hewlett-Packard Development Company, L.P. が保有しており、本書中の解説および図、表は弊社からの文書による許可なしに、その全体または一部を、いかなる場合にも再版あるいは複製することを禁じます。

また、本書に記載されている事項は、予告なく変更されることがありますので、あらかじめご承知おきください。万一、本書の記述に誤りがあった場合でも、弊社は一切その責任を負いかねます。

本書で解説するソフトウェア (対象ソフトウェア) は、所定のライセンス契約が締結された場合に限り、その使用あるいは複製が許可されます。

弊社は、弊社または弊社の指定する会社から納入された機器以外の機器で対象ソフトウェアを使用した場合、その性能あるいは信頼性について一切責任を負いかねます。

UNIX®は、The Open Group の登録商標です。

X/Open®は、英国およびその他の国における X/Open Company Ltd. の登録商標です。

Intel®および Itanium®は、米国およびその他の国における Intel Corporation またはその子会社の商標または登録商標です。

原典： HP C Run-Time Library Reference Manual for OpenVMS Systems
© 2006 Hewlett-Packard Development Company, L.P.

本書は、日本語 VAX DOCUMENT V 2.1を用いて作成しています。

HP Cランタイム・ライブラリの一部は、カリフォルニア大学バークレイ校およびその協力者 (contributors) が著作権を保有するソースを使用して実装されています。

Copyright (c) 1981 Regents of the University of California.

All rights reserved.

次の条件が満たされる場合、変更されているかどうかにかかわらず、ソースおよびバイナリ形式の再配布と使用が認められます。

1. ソース・コードを再配布する際は、上記の著作権に関する通告、再配布と使用に関するこの条件一覧、以下の免責事項を添付する必要があります。
2. バイナリ形式での再配布の際は、添付するドキュメントや他のマテリアルとともに、上記の著作権に関する通告、再配布と使用に関するこの条件一覧、以下の免責事項を添付する必要があります。
3. 本ソフトウェアの機能や本ソフトウェアを使用していることを記載した宣伝広告資料すべてに、次の情報を記載する必要があります。「本製品には、カリフォルニア大学バークレイ校およびその協力者が開発したソフトウェアが含まれています。」
4. 事前に書面による承認を受けない限り、本ソフトウェアを利用して開発された製品の宣伝や販売促進で、大学の名前や協力者の名前を使用することは認められません。

本ソフトウェアは、開発者および協力者が提供するものを「そのまま」提供するものであり、商品性や特定の目的への適合性の暗黙の保証も含めて（これらに限定されません）、いかなる表明や暗黙の保証も適用されません。いかなる場合も、開発者および協力者は、直接的または間接的損害、事故による損害、特別損害など（たとえば、代替商品やサービスの調達の必要性、製品が使用できなくなる障害、データや収益の消失、ビジネスの中断など）の発生の責任を負いません。さらに、本ソフトウェアの使用によって発生する契約上の責任、無過失責任、不法行為（過失によるもの他）に関しても、そのような損害の可能性があらかじめ助言されていた場合でも、開発者および協力者は一切の責任を負いません。

目次

まえがき	xix
------------	-----

リファレンス・セクション

a64l (<i>Alpha, I64</i>)	REF-3
abort	REF-5
abs	REF-6
access	REF-7
acos	REF-9
acosh (<i>Alpha, I64</i>)	REF-10
[w]addch	REF-11
[w]addstr	REF-12
alarm	REF-13
asctime, asctime_r	REF-15
asin	REF-17
asinh (<i>Alpha, I64</i>)	REF-18
assert	REF-19
atan	REF-21
atan2	REF-22
atanh (<i>Alpha, I64</i>)	REF-24
atexit	REF-25
atof	REF-27
atoi, atol	REF-28
atoq, atoll (<i>Alpha, I64</i>)	REF-29
basename	REF-30
bcmp	REF-32
bcopy	REF-33
box	REF-34
brk	REF-35
bsearch	REF-37
btowc	REF-40
bzero	REF-41
cabs	REF-42
cacos (<i>Alpha, I64</i>)	REF-43
cacosh (<i>Alpha, I64</i>)	REF-44
calloc	REF-45
carg (<i>Alpha, I64</i>)	REF-46
casin (<i>Alpha, I64</i>)	REF-47
casinh (<i>Alpha, I64</i>)	REF-48

<code>catan</code> (<i>Alpha, I64</i>)	REF-49
<code>catanh</code> (<i>Alpha, I64</i>)	REF-50
<code>catclose</code>	REF-51
<code>catgets</code>	REF-52
<code>catopen</code>	REF-55
<code>cbrt</code> (<i>Alpha, I64</i>)	REF-58
<code>ccos</code> (<i>Alpha, I64</i>)	REF-59
<code>ccosh</code> (<i>Alpha, I64</i>)	REF-60
<code>ceil</code>	REF-61
<code>cexp</code> (<i>Alpha, I64</i>)	REF-62
<code>cfree</code>	REF-63
<code>chdir</code>	REF-64
<code>chmod</code>	REF-66
<code>chown</code>	REF-68
<code>cimag</code> (<i>Alpha, I64</i>)	REF-69
<code>[w]clear</code>	REF-70
<code>clearerr</code>	REF-71
<code>clearerr_unlocked</code> (<i>Alpha, I64</i>)	REF-72
<code>clearok</code>	REF-73
<code>clock</code>	REF-74
<code>clock_getres</code> (<i>Alpha, I64</i>)	REF-75
<code>clock_gettime</code> (<i>Alpha, I64</i>)	REF-77
<code>clock_settime</code> (<i>Alpha, I64</i>)	REF-78
<code>clog</code> (<i>Alpha, I64</i>)	REF-80
<code>close</code>	REF-81
<code>closedir</code>	REF-83
<code>[w]clrattr</code>	REF-86
<code>[w]clrtoobot</code>	REF-87
<code>[w]clrtoeol</code>	REF-88
<code>confstr</code>	REF-89
<code>conj</code> (<i>Alpha, I64</i>)	REF-91
<code>copysign</code> (<i>Alpha, I64</i>)	REF-92
<code>cos</code>	REF-93
<code>cosh</code>	REF-94
<code>cot</code>	REF-95
<code>cpow</code> (<i>Alpha, I64</i>)	REF-96
<code>cproj</code> (<i>Alpha, I64</i>)	REF-97
<code>creal</code> (<i>Alpha, I64</i>)	REF-98
<code>creat</code>	REF-99
<code>[no]crmode</code>	REF-106
<code>crypt</code>	REF-108
<code>csin</code> (<i>Alpha, I64</i>)	REF-110
<code>csinh</code> (<i>Alpha, I64</i>)	REF-111
<code>csqrt</code> (<i>Alpha, I64</i>)	REF-112
<code>ctan</code> (<i>Alpha, I64</i>)	REF-113
<code>ctanh</code> (<i>Alpha, I64</i>)	REF-114
<code>ctermid</code>	REF-115

ctime, ctime_r	REF-116
cuserid	REF-118
DECC\$CRTL_INIT	REF-119
decc\$feature_get	REF-120
decc\$feature_get_index	REF-122
decc\$feature_get_name	REF-123
decc\$feature_get_value	REF-124
decc\$feature_set	REF-126
decc\$feature_set_value	REF-128
decc\$feature_show	REF-130
decc\$feature_show_all	REF-131
decc\$fix_time	REF-132
decc\$from_vms	REF-134
decc\$match_wild	REF-136
decc\$record_read	REF-138
decc\$record_write	REF-139
decc\$set_child_default_dir (<i>Alpha, I64</i>)	REF-140
decc\$set_child_standard_streams	REF-142
decc\$set_reentrancy	REF-147
decc\$to_vms	REF-149
decc\$translate_vms	REF-152
decc\$validate_wchar	REF-154
decc\$write_eof_to_mbx	REF-156
[w]delch	REF-159
delete	REF-160
[w]deleteln	REF-162
delwin	REF-163
difftime	REF-164
dirname	REF-165
div	REF-167
dlclose	REF-168
dlopen	REF-169
dlsym	REF-170
drand48	REF-172
dup, dup2	REF-173
[no]echo	REF-175
ecvt	REF-176
encrypt	REF-177
endgrent (<i>Alpha, I64</i>)	REF-179
endpwent	REF-181
endwin	REF-182
erand48	REF-183
[w]erase	REF-184
erf	REF-186
execl	REF-187
execle	REF-188
	REF-190

execlp	REF-192
execv	REF-193
execve	REF-194
execvp	REF-196
exit, _exit	REF-197
exp	REF-199
exp2 (<i>Alpha, I64</i>)	REF-201
fabs	REF-203
fchmod	REF-204
fchown	REF-205
fclose	REF-207
fcntl	REF-209
fcvt	REF-216
fdim (<i>Alpha, I64</i>)	REF-218
fdopen	REF-219
feof	REF-220
feof_unlocked (<i>Alpha, I64</i>)	REF-221
ferror	REF-222
ferror_unlocked (<i>Alpha, I64</i>)	REF-223
fflush	REF-224
ffs	REF-225
fgetc	REF-226
fgetc_unlocked (<i>Alpha, I64</i>)	REF-227
fgetname	REF-228
fgetpos	REF-230
fgets	REF-232
fgetwc	REF-234
fgetws	REF-235
fileno	REF-237
finite (<i>Alpha, I64</i>)	REF-238
flockfile (<i>Alpha, I64</i>)	REF-239
floor	REF-240
fma (<i>Alpha, I64</i>)	REF-241
fmax (<i>Alpha, I64</i>)	REF-242
fmin (<i>Alpha, I64</i>)	REF-243
fmod	REF-244
fopen	REF-245
fp_class (<i>Alpha, I64</i>)	REF-247
fpathconf	REF-249
fprintf	REF-251
fputc	REF-253
fputc_unlocked (<i>Alpha, I64</i>)	REF-254
fputs	REF-255
fputwc	REF-256
fputws	REF-258
fread	REF-259
free	REF-261

freopen	REF-262
frexp	REF-264
fscanf	REF-266
fseek	REF-268
fseeko	REF-270
fsetpos	REF-271
fstat	REF-272
fstatvfs (<i>Alpha, I64</i>)	REF-276
fsync	REF-278
ftell	REF-279
ftello	REF-280
ftime	REF-281
ftruncate	REF-283
ftrylockfile (<i>Alpha, I64</i>)	REF-284
ftw	REF-285
funlockfile (<i>Alpha, I64</i>)	REF-288
fwait	REF-289
fwide	REF-290
fwprintf	REF-292
fwrite	REF-295
fwscanf	REF-297
gcvt	REF-299
getc	REF-301
getc_unlocked (<i>Alpha, I64</i>)	REF-302
[w]getch	REF-303
getchar	REF-304
getchar_unlocked (<i>Alpha, I64</i>)	REF-305
getclock	REF-306
getcwd	REF-308
getdtablesize	REF-310
getegid	REF-311
getenv	REF-313
geteuid	REF-315
getgid	REF-317
getgrent (<i>Alpha, I64</i>)	REF-319
getgrgid (<i>Alpha, I64</i>)	REF-321
getgrgid_r (<i>Alpha, I64</i>)	REF-323
getgrnam (<i>Alpha, I64</i>)	REF-325
getgrnam_r (<i>Alpha, I64</i>)	REF-327
getgroups	REF-329
getitimer	REF-331
getlogin	REF-333
getname	REF-334
getopt	REF-336
getpagesize	REF-340
getpgid (<i>Alpha, I64</i>)	REF-341
getpgrp (<i>Alpha, I64</i>)	REF-342

getpid	REF-343
getppid	REF-344
getpwent	REF-345
getpwnam, getpwnam_r	REF-347
getpwuid, getpwuid_r (<i>Alpha, I64</i>)	REF-350
gets	REF-353
getsid (<i>Alpha, I64</i>)	REF-354
[w]getstr	REF-355
gettimeofday	REF-356
getuid	REF-357
getw	REF-359
getwc	REF-360
getwchar	REF-361
getyx	REF-362
glob (<i>Alpha, I64</i>)	REF-363
globfree	REF-368
gmtime, gmtime_r	REF-369
gsignal	REF-371
hypot	REF-373
iconv	REF-374
iconv_close	REF-376
iconv_open	REF-377
ilogb (<i>Alpha, I64</i>)	REF-380
[w]inch	REF-381
index	REF-382
initscr	REF-383
initstate	REF-384
[w]insch	REF-386
[w]insertln	REF-387
[w]insstr	REF-388
isalnum	REF-389
isalpha	REF-390
isapipe	REF-391
isascii	REF-392
isatty	REF-393
iscntrl	REF-394
isdigit	REF-395
isgraph	REF-396
islower	REF-397
isnan (<i>Alpha, I64</i>)	REF-398
isprint	REF-399
ispunct	REF-400
isspace	REF-401
isupper	REF-402
iswalnum	REF-403
iswalpha	REF-404
iswcntrl	REF-405

iswctype	REF-406
iswdigit	REF-408
iswgraph	REF-409
iswlower	REF-410
iswprint	REF-411
iswpunct	REF-412
iswspace	REF-413
iswupper	REF-414
iswxdigit	REF-415
isxdigit	REF-416
j0, j1, jn (<i>Alpha, I64</i>)	REF-417
rand48	REF-419
kill	REF-421
l64a (<i>Alpha, I64</i>)	REF-423
labs	REF-425
lchown	REF-426
lcong48	REF-427
ldexp	REF-428
ldiv	REF-429
leaveok	REF-430
lgamma (<i>Alpha, I64</i>)	REF-431
link	REF-432
localeconv	REF-434
localtime, localtime_r	REF-438
log, log2, log10	REF-441
log1p (<i>Alpha, I64</i>)	REF-443
logb (<i>Alpha, I64</i>)	REF-444
longjmp	REF-445
longname	REF-447
rand48	REF-448
lrint (<i>Alpha, I64</i>)	REF-450
lround (<i>Alpha, I64</i>)	REF-451
lseek	REF-452
lstat (<i>Alpha, I64</i>)	REF-454
lwait	REF-455
malloc	REF-456
mblen	REF-458
mbrlen	REF-459
mbrtowc	REF-461
mbstowcs	REF-463
mbtowc	REF-465
mbsinit	REF-467
mbsrtowcs	REF-468
memccpy	REF-470
memchr	REF-472
memcmp	REF-474
memcpy	REF-475

memmove	REF-477
memset	REF-479
mkdir	REF-481
mkstemp	REF-485
mktemp	REF-486
mktime	REF-487
mmap	REF-489
modf	REF-495
[w]move	REF-496
mprotect	REF-497
rand48	REF-499
msync	REF-501
munmap	REF-503
mv[w]addch	REF-505
mv[w]addstr	REF-507
mvcur	REF-509
mv[w]delch	REF-510
mv[w]getch	REF-511
mv[w]getstr	REF-512
mv[w]inch	REF-513
mv[w]insch	REF-514
mv[w]insstr	REF-515
mvwin	REF-516
nanosleep (<i>Alpha, I64</i>)	REF-517
newwin	REF-519
nextafter (<i>Alpha, I64</i>)	REF-520
nexttoward (<i>Alpha, I64</i>)	REF-521
nice	REF-522
nint (<i>Alpha, I64</i>)	REF-524
[no]nl	REF-525
nl_langinfo	REF-526
rand48	REF-530
open	REF-532
opendir	REF-535
overlay	REF-537
overwrite	REF-538
pathconf	REF-539
pause	REF-541
pclose	REF-542
perror	REF-543
pipe	REF-545
poll (<i>Alpha, I64</i>)	REF-550
popen	REF-554
pow	REF-556
pread (<i>Alpha, I64</i>)	REF-558
printf	REF-560
[w]printw	REF-561

putc	REF-563
putc_unlocked (<i>Alpha, I64</i>)	REF-565
putchar	REF-567
putchar_unlocked (<i>Alpha, I64</i>)	REF-568
putenv	REF-569
puts	REF-571
putw	REF-572
putwc	REF-573
putwchar	REF-574
pwrite (<i>Alpha, I64</i>)	REF-575
qabs, labs (<i>Alpha, I64</i>)	REF-577
qdiv, lldiv (<i>Alpha, I64</i>)	REF-578
qsort	REF-579
raise	REF-581
rand, rand_r	REF-583
random	REF-584
[no]raw	REF-585
read	REF-587
readdir, readdir_r	REF-589
readlink (<i>Alpha, I64</i>)	REF-591
readv (<i>Alpha, I64</i>)	REF-593
realloc	REF-596
realpath	REF-598
[w]refresh	REF-600
remainder (<i>Alpha, I64</i>)	REF-601
remquo (<i>Alpha, I64</i>)	REF-602
remove	REF-603
rename	REF-605
rewind	REF-607
rewinddir	REF-608
rindex	REF-609
rint (<i>Alpha, I64</i>)	REF-610
rmdir	REF-611
sbrk	REF-612
scalb (<i>Alpha, I64</i>)	REF-614
scanf	REF-615
[w]scanw	REF-616
scroll	REF-618
scrollok	REF-619
seed48	REF-620
seekdir	REF-622
[w]setattr	REF-623
setbuf	REF-624
setenv	REF-626
seteuid (<i>Alpha, I64</i>)	REF-627
setgid	REF-628
setgrent (<i>Alpha, I64</i>)	REF-630

setitimer	REF-631
setjmp	REF-633
setkey	REF-635
setlocale	REF-636
setpgid (<i>Alpha, I64</i>)	REF-640
setpgrp (<i>Alpha, I64</i>)	REF-642
setpwent	REF-643
setregid (<i>Alpha, I64</i>)	REF-644
setreuid (<i>Alpha, I64</i>)	REF-646
setsid (<i>Alpha, I64</i>)	REF-648
setstate	REF-649
setuid	REF-651
setvbuf	REF-653
shm_open (<i>Alpha, I64</i>)	REF-656
shm_unlink (<i>Alpha, I64</i>)	REF-659
sigaction	REF-661
sigaddset	REF-665
sigblock	REF-667
sigdelset	REF-668
sigemptyset	REF-669
sigfillset	REF-671
sighold (<i>Alpha, I64</i>)	REF-672
sigignore (<i>Alpha, I64</i>)	REF-674
sigismember	REF-676
siglongjmp	REF-677
sigmask	REF-678
signal	REF-679
sigpause	REF-681
sigpending	REF-682
sigprocmask	REF-683
sigrelse (<i>Alpha, I64</i>)	REF-685
sigsetjmp	REF-687
sigsetmask	REF-689
sigstack (<i>VAX only</i>)	REF-690
sigsuspend	REF-692
sigtimedwait (<i>Alpha, I64</i>)	REF-694
sigvec	REF-696
sigwait (<i>Alpha, I64</i>)	REF-698
sigwaitinfo (<i>Alpha, I64</i>)	REF-700
sin	REF-702
sinh	REF-703
sleep	REF-704
snprintf	REF-705
sprintf	REF-707
sqrt	REF-709
srand	REF-710
srand48	REF-711

srandom	REF-712
sscanf	REF-713
ssignal	REF-715
[w]standend	REF-717
[w]standout	REF-718
stat	REF-719
statvfs (<i>Alpha, I64</i>)	REF-725
strcasecmp	REF-728
strcat	REF-729
strchr	REF-731
strcmp	REF-733
strcoll	REF-734
strcpy	REF-735
strcspn	REF-736
strdup	REF-737
strerror	REF-738
strfmon	REF-740
strftime	REF-744
strlen	REF-750
strncasecmp	REF-751
strncat	REF-752
strncmp	REF-753
strncpy	REF-755
strnlen	REF-757
strpbrk	REF-758
strptime	REF-759
strrchr	REF-764
strsep	REF-766
strspn	REF-768
strstr	REF-769
strtod	REF-771
strtok, strtok_r	REF-773
strtol	REF-777
strtoq, strtoll (<i>Alpha, I64</i>)	REF-779
strtoul	REF-781
strtouq, strtoull (<i>Alpha, I64</i>)	REF-783
strxfrm	REF-785
subwin	REF-789
swab	REF-791
swprintf	REF-792
swscanf	REF-794
symlink (<i>Alpha, I64</i>)	REF-796
sysconf	REF-798
system	REF-803
tan	REF-805
tanh	REF-806
telldir	REF-807

tempnam	REF-808
tgamma (<i>Alpha, I64</i>)	REF-811
time	REF-812
times	REF-813
tmpfile	REF-815
tmpnam	REF-816
toascii	REF-817
tolower	REF-818
_tolower	REF-819
touchwin	REF-821
toupper	REF-822
_toupper	REF-823
towctrans	REF-825
towlower	REF-826
towupper	REF-827
trunc (<i>Alpha, I64</i>)	REF-828
truncate	REF-829
ttyname, ttyname_r	REF-831
tzset	REF-833
ualarm	REF-838
umask	REF-840
uname	REF-842
ungetc	REF-843
ungetwc	REF-844
unlink (<i>Alpha, I64</i>)	REF-846
unordered (<i>Alpha, I64</i>)	REF-847
unsetenv	REF-848
usleep	REF-849
utime	REF-850
utimes	REF-853
VAXC\$CRTL_INIT	REF-856
VAXC\$ESTABLISH	REF-858
va_arg	REF-860
va_count	REF-861
va_end	REF-862
va_start, va_start_1	REF-863
vfork	REF-866
vfprintf	REF-868
vfscanf	REF-869
vfwprintf	REF-871
vfwscanf	REF-874
vprintf	REF-876
vscanf	REF-877
vsprintf (<i>Alpha, I64</i>)	REF-878
vsprintf	REF-880
vsscanf	REF-881
vswprintf	REF-883

vswscanf	REF-885
vwprintf	REF-887
vwscanf	REF-888
wait	REF-889
wait3	REF-890
wait4	REF-893
waitpid	REF-896
wcrtomb	REF-900
wcscat	REF-902
wcschr	REF-905
wcscmp	REF-907
wcscoll	REF-909
wcscpy	REF-910
wcscspn	REF-911
wcsftime	REF-913
wcslen	REF-919
wcsncat	REF-920
wcsncmp	REF-923
wcsncpy	REF-925
wcspbrk	REF-927
wcsrchr	REF-929
wcsrtombs	REF-931
wcsspn	REF-933
wcsstr	REF-935
wcstod	REF-936
wcstok	REF-938
wcstol	REF-941
wcstombs	REF-943
wcstoul	REF-945
wcswcs	REF-948
wcswidth	REF-950
wcsxfrm	REF-951
wctob	REF-955
wctomb	REF-956
wctrans	REF-957
wctype	REF-958
wcwidth	REF-962
wmemchr	REF-963
wmemcmp	REF-964
wmemcpy	REF-965
wmemmove	REF-966
wmemset	REF-968
wprintf	REF-969
wrapok	REF-971
write	REF-972
writev	REF-974
wscanf	REF-976

y0, y1, yn (*Alpha*, *I64*)REF-977

索引

図

REF-1 パイプの読み込みと書き込みREF-549

表

REF-1 mode 引数の解釈 REF-7
REF-2 ファイル保護の値とその意味REF-66
REF-3 RMS の有効なキーワードと値REF-101
REF-4 tm 構造体REF-439
REF-5 strfmon の変換指定に含まれるオプションの文字REF-741
REF-6 strfmon の変換指定子REF-741
REF-7 strftime 変換指定のオプション要素REF-745
REF-8 strftime の変換指定子REF-745
REF-9 strptime の変換指定REF-761
REF-10 sysconf 引数と戻り値REF-798
REF-11 タイム・ゾーン初期化規則REF-834
REF-12 vfork 関数と fork 関数REF-866
REF-13 wcsftime 変換指定のオプション要素REF-914
REF-14 wcsftime の変換指定子REF-914

まえがき

本書では、HP C ランタイム・ライブラリ (HP C Run-Time Library (RTL)) の関数について説明します。

HP C ランタイム・ライブラリの次のことについては『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』を参照してください。

- 入出力 (I/O) 操作、文字および文字列操作、算術演算、エラー検出、サブプロセスの生成、システム・アクセス、画面管理を行う RTL 関数およびマクロに関する参照情報
- オペレーティング・システム間の移植性に関する問題点と、可能な場合は、TCP/IP Services for OpenVMS製品や、他の TCP/IP プロトコルの実装用のインターネット・アプリケーション・プログラムを作成するために使用されるHP C for OpenVMSソケット・ルーチンについて

HP C RTL には、XPG4 準拠の国際化サポートが含まれており、異なる言語やカルチャーで動作可能なソフトウェアを開発するのに役立つ関数が提供されます。

本書の対象読者

本書は、HP C RTL で提供される関数とマクロに関する参照情報が必要な、経験の豊富なプログラマーおよび初心者プログラマーを対象にしています。

本書の構成

本書は、リファレンス・セクションで HP C RTL のすべての関数についてアルファベット順に提示し、説明しています。

関連ドキュメント

OpenVMS システム向けのプログラムをHP Cで作成する場合、次のドキュメントが役立ちます。

- 『HP C User's Guide for OpenVMS Systems』 — HP C for OpenVMSシステムの使用法に関する情報が必要な C プログラマーを対象にしています。
- 『HP C Language Reference Manual』 — HP システムでのHP Cの言語リファレンス情報を示します。

- 『VAX C to HP C Migration Guide』 — OpenVMS VAX アプリケーション・プログラマがVAX CからHP Cに移行するのに役立ちます。
- 『HP C Installation Guide for OpenVMS VAX Systems』 — VAX システムにHP Cソフトウェアをインストールする OpenVMS システム・プログラマを対象にしています。
- 『HP C Installation Guide for OpenVMS Alpha Systems』 — AlphaシステムにHP Cソフトウェアをインストールする OpenVMS システム・プログラマを対象にしています。
- 『OpenVMS Master Index』 — VAX および Alpha マシン・アーキテクチャや OpenVMS システム・サービスを使用する必要があるプログラマを対象にしています。このインデックスには、OpenVMS オペレーティング・システムへのアクセスに関する個別のトピックを説明したドキュメントの一覧が示されています。
- 『HP TCP/IP Services for OpenVMS Sockets API and System Services Programming』 — 『HP TCP/IP Services for OpenVMS』製品または他のTCP/IP プロトコル実装用の、インターネット・アプリケーション・プログラムを作成するためのソケット・ルーチンについての情報を示します。
- 『HP TCP/IP Services for OpenVMS Guide to IPv6』 — HP TCP/IP Services for OpenVMS のIPv6 機能や、システム上でのIPv6 のインストールや構成方法、ソケット・アプリケーション・プログラミング・インタフェース (API) の変更、IPv6 環境で動作させるためにアプリケーションを移植する方法についての情報を示します。
- 『X/Open Portability Guide, Issue 3』 — 一般に XPG3 と呼んでいる仕様について解説しています。
- 『X/Open CAE Specification System Interfaces and Headers, Issue 4』 — 一般に XPG4 と呼んでいる仕様について解説しています。
- 『X/Open CAE Specification, System Interfaces and Headers, Issue 4, Version 2』 — 一般に XPG4 V2 と呼んでいる仕様に関して解説しています。
- 『X/Open CAE Specification, System Interfaces and Headers, Issue 5』 — 一般に XPG5 と呼んでいる仕様に関して解説しています。
- 『Technical Standard. System Interfaces, Issue 6』 — Open Group の技術標準と IEEE 標準を組み合わせたものです。XPG6 と呼ぶ IEEE Std 1003.1-2001 仕様に関して解説しています。
- 『Standard for Information Technology - Portable Operating System Interface (POSIX) - Part 1: System Application Program Interface (API)—Amendment 2: Threads Extension [C Language]』 — 一般に POSIX 1003.1c-1995 と呼んでいる仕様に関して解説しています。
- 『ISO/IEC 9945-2:1993 - Information Technology - Portable Operating System Interface (POSIX) - Part 2: Shell and Utilities』 — 一般に ISO POSIX-2と呼んでいる仕様に関して解説しています。

- 『ISO/IEC 9945-1:1990 - Information Technology - Portable Operating System Interface (POSIX) - Part 1: System Application Programming Interface (API) (C Language)』 — 一般に ISO POSIX-1と呼んでいる仕様について解説しています。
- 『ANSI/ISO/IEC 9899:1999 - Programming Languages - C』 — 1999 年 12 月に ISO によって公開され、2000 年 4 月に ANSI 標準として採用された C99 標準について解説しています。
- 『ISO/IEC 9899:1990-1994 - Programming Languages - C, Amendment 1: Integrity』 — 一般にISO C, Amendment 1と呼んでいる仕様について解説しています。
- 『ISO/IEC 9899:1990[1992] - Programming Languages - C』 — 一般にISO Cと呼んでいる仕様について解説しています。標準的な部分 (normative part) はX3.159-1989, American National Standard for Information Systems - Programming Language C (ANSI Cとも呼ぶ) と同じです。

HP OpenVMS 製品およびサービスについての詳細は、弊社の Web サイトを参照してください。アドレスは次のとおりです。

<http://www.hp.com/go/openvms>

<http://www.hp.com/jp/openvms>

本書で使用する表記法

表記法	意味
<code>[Return]</code>	<code>[Return]</code> は端末上の Return キーを 1 回押すことを示します。
<code>Ctrl/X</code>	<code>Ctrl/X</code> (英字の X は端末の制御文字を表す) は、Ctrl キーを押したまま指定の端末文字キー (X) を押すことを示します。
<code>switch</code> 文 <code>int</code> データ型 <code>fprintf</code> 関数 <code><stdio.h></code> ヘッダ・ファイル	モノスペース文字は言語キーワードおよびHP C関数とヘッダ・ファイルの名前を示します。また、例で使用している特定の変数名を参照するときも使用します。
<i>arg1</i>	斜体は引数やパラメータ名を示すプレースホルダとして使用し、新出用語を強調するときも使用します。
<code>\$ RUN CPROG [Return]</code>	ユーザと対話する例では、ユーザ入力は大文字で示します。
<code>float x;</code> . . .	垂直方向の省略記号は、プログラムやプログラムからの出力の一部が省略されていることを示します。例では関連する部分だけが示されています。
<code>x = 5;</code> <code>option, ...</code>	水平方向の省略記号は、パラメータ、オプション、値を追加入力できることを示します。省略記号の前のコンマは、後続の項目の区切り文字を表します。

表記法	意味
[output-source, ...]	関数構文やその他の場所で使用している角括弧は、その構文要素が省略可能であることを示します。しかし、OpenVMS ファイル指定でディレクトリ名を区切るために使用する角括弧や、HP Cソース・コードで多次元配列の次元を区切るために使用する角括弧は省略できません。
sc-specifier::= auto static extern register	構文定義で、別々の行に示されている項目は組み合わせて指定できないことを示します。
[a b]	2 つ以上の項目が縦線()で区切られ、角括弧で囲まれている場合は、2 つの構文要素のいずれかを選択しなければならないことを示します。
Δ	デルタ記号は、1 文字の ASCII スペース文字を表します。

プラットフォーム・ラベル

プラットフォームは、異なる環境を提供するオペレーティング・システムとハードウェアの組み合わせです。本書では、VAX、Alpha、Itanium プロセッサで動作する OpenVMS オペレーティング・システムに適用される情報を示します。

次のように特に指定した場合を除き、本書の情報はすべてのプロセッサに適用されます。

ラベル	説明
(Alpha only)	Alpha プロセッサ固有。
(I64 only)	OpenVMS オペレーティング・システムが動作している Intel Itanium プロセッサ固有。このプラットフォームでは、オペレーティング・システムの製品名は OpenVMS Industry Standard 64 (またはその短縮形 OpenVMS I64 あるいは I64) です。
(VAX only)	VAX プロセッサ固有。
(Alpha, I64)	I64 プロセッサおよび Alpha プロセッサ固有。

新機能と変更された機能 - OpenVMS Version 8.2

OpenVMS Version 8.2 で、C ランタイム・ライブラリの以下の機能が拡張されました。これらの拡張により、UNIX 移植性、標準への準拠、追加のユーザ制御機能を選択する際の柔軟性が改善されます。また、新しい C RTL 関数も含まれています。

- I64 のサポート
HP C RTL は、HP OpenVMS Industry Standard 64 for Integrity Servers と、Alpha システムおよび VAX システムでサポートされるようになりました。
- ファイル・ロック関数

UNIX ポータビリティを高めるために、以下の X/Open ファイル・ポインタ・ロック関数が追加されました。これらの関数を使用することで、ファイル・ポインタをロックし、スレッド化されたプログラム全体でアクセスの同期が取れるようになります。

```
flockfile
ftrylockfile
funlockfile
clearerr_unlocked
getc_unlocked
getchar_unlocked
feof_unlocked
ferror_unlocked
fgetc_unlocked
fputc_unlocked
putc_unlocked
putchar_unlocked
```

- 標準準拠のstat構造体

UNIX ポータビリティを高めるために、stat構造体の新しい標準準拠の定義および関連する定義が、<stat.h>に追加されました。これらの新しい定義を使用するためには、次の新しい機能テスト・マクロを定義してアプリケーションをコンパイルする必要があります。

```
_USE_STD_STAT
```

- ファイル・システム統計情報のサポート

UNIX ポータビリティを高めるために、以下の X/Open 関数が追加されました。

```
statvfs
fstatvfs
```

- fcntlのファイル状態フラグ

fcntl関数にコマンド・オプション F_SETFL および F_GETFL が追加され、ファイル状態フラグの設定と取得ができるようになりました。

- globとglobfreeの 64 ビット・サポート

関数globとglobfreeに、64 ビット・サポートが追加されました。その結果、以下の追加の関数エントリ・ポイントが、32 ビットと 64 ビットのポインタ・サイズで利用できるようになりました。

```
_glob32      _glob64
_globfree32  _globfree64
```

- ソケット・ルーチンsocketpair

ソケット・ルーチンsocketpairが追加されました。このルーチンは、接続済みのソケット・ペアを作成するために使用します。下位の TCP/IP 製品で TCPIP\$SOCKETPAIR 機能が利用できる必要があります。

リファレンス・セクション

このリファレンス・セクションでは、HP C ランタイム・ライブラリ (RTL) に含まれている関数について、アルファベット順に説明します。

a64l (*Alpha, I64*)

文字列をlong整数に変換します。

Format

```
#include <stdlib.h>
long a64l (const char *s);
```

引数

s
long整数に変換される文字列へのポインタ。

Description

a64l関数とl64a関数は、次のように、base-64 ASCII 文字として格納された数値を操作するために使用されます。

- a64lは、文字列をlong整数に変換します。
- l64aは、long整数を文字列に変換します。

long整数を格納するための各文字は、0 ~ 63 の数値を表しています。long整数を表すために、最大 6 文字まで使用できます。

文字は、次のように変換されます。

- ピリオド(.)は、0 を表します。
- スラッシュ(/)は、1 を表します。
- 0 ~ 9 の数字は、2 ~ 11 を表します。
- 大文字 A ~ Z は、12 ~ 37 を表します。
- 小文字 a ~ z は、38 ~ 63 を表します。

a64l関数は、base-64 表現へのポインタ (最初の桁が最下位) を受け取り、対応するlong値を返します。sパラメータが指す文字列が 6 文字を超えている場合、a64lは最初の 6 文字だけを使用します。

文字列の最初の 6 文字の中にヌル終了文字が含まれている場合、a64lはヌル終了文字より前の文字だけを使用します。

a64l関数は、文字列を左から右へと変換します。このとき、最下位の数値が左端で、各文字を 6 ビットの base-64 数としてデコードします。

sが NULL ポインタの場合、またはsが指す文字列が 164aを以前に呼び出して生成された文字列でない場合、a64lの動作は規定されていません。

164aも参照してください。

Return value

n	正常終了した場合、入力文字列を変換して得られたlong値です。
0L	sが指している文字列が空文字列であることを示します。

abort

プログラムの実行を終了するシグナル SIGABRT を送信します。

Format

```
#include <stdlib.h>
void abort (void);
```

abs

整数の絶対値を返します。

Format

```
#include <stdlib.h>
int abs (int x);
```

引数

x
整数。

戻り値

x	入力引数の絶対値。引数が LONG_MIN の場合、-LONG_MIN は int 変数に格納できないため、abs は LONG_MIN を返します。
---	---

access

指定されたアクセス・モードがファイルで許可されているかどうかを確認します。この関数は UIC 保護のほか、OpenVMS ACL (Access Control List) を確認します。

注意

access関数は、引数としてネットワーク・ファイルを受け付けません。

Format

```
#include <unistd.h>

int access (const char *file_spec, int mode);
```

Argument

file_spec

OpenVMS 形式または UNIX 形式のファイル指定を示す文字列。ファイル指定には通常のデフォルトおよび論理名変換が適用されます。

mode

表 REF-1 に示すように解釈されます。

表 REF-1 mode 引数の解釈

mode 引数	アクセス・モード
F_OK	ファイルが存在するかどうかの判定
X_OK	実行
W_OK	書き込み (暗黙に削除アクセスも示す)
R_OK	読み込み

アクセス・モードの組み合わせは値の論理和 (OR) によって指定します。たとえば、ファイルで RWED アクセス・モードが許可されているかどうかを確認するには、次のように指定して access を呼び出します。

```
access (file_spec, R_OK | W_OK | X_OK);
```

Return value

0	アクセスが許可されていることを示します。
-1	アクセスが許可されていないことを示します。

例

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>

main()
{
    if (access("sys$login:login.com", F_OK)) {
        perror("ACCESS - FAILED");
        exit(2);
    }
}
```

acos

引数の逆余弦を返します。

Format

```
#include <math.h>
double acos (double x);
float acosf (float x); (Alpha, I64)
long double acosl (long double x); (Alpha, I64)
double acosd (double x); (Alpha, I64)
float acosdf (float x); (Alpha, I64)
long double acosdl (long double x); (Alpha, I64)
```

引数

x
領域 $[-1,1]$ の実数値として表したラジアン値。

Description

acos関数は、領域 $[-1,1]$ のxに対して、 $[0,\pi]$ ラジアンの範囲でxの逆余弦の主値を計算します。

acosd関数は、領域 $[-1,1]$ のxに対して、 $[0,180]$ 度の範囲でxの逆余弦の主値を計算します。

$\text{abs}(x) > 1$ の場合、 $\text{acos}(x)$ の値は0であり、 errno はEDOMに設定されます。

acosh (Alpha, I64)

引数の双曲線逆余弦を返します。

Format

```
#include <math.h>

double acosh (double x);
float acoshf (float x);
long double acoshl (long double x);
```

引数

x
領域[1, +Infinity]の実数値として表したラジアン値。

Description

acosh関数は、領域[1, + 無限大]のxに対して、xの双曲線逆余弦を返します。ただし、 $\text{acosh}(x) = \ln(x + \sqrt{x^2 - 1})$ です。

acosh関数はcosh関数の逆関数であり、 $\text{acosh}(\cosh(x)) = |x|$ です。

$x < 1$ は不正な引数です。

[w]addch

ウィンドウの現在のカーソルの位置に 1 文字を追加します。

Format

```
#include <curses.h>
int addch (char ch);
int waddch (WINDOW *win, char ch);
```

Argument

win

ウィンドウを指すポインタ。

ch

追加する文字。改行文字 (\n) は、行を行末まで消去し、カーソルを次の行の同じ x 座標に移動します。リターン文字 (\r) は、カーソルをウィンドウの行の先頭に移動します。タブ文字 (\t) は、カーソルをウィンドウ内の次のタブストップに移動します。

Description

waddch関数をサブウィンドウで使用すると、その親ウィンドウにも文字が書き込まれます。

addchルーチンはwaddchと同じ機能を実行しますが、stdscrウィンドウに対して動作する点が異なります。

文字を画面に書き込んだ後、カーソルは移動します。

Return value

OK

正常終了を示します。

ERR

文字を書き込んだ結果、画面が不正にスクロールされることを示します。詳細については、scrolllok関数を参照してください。

[w]addstr

strによって示される文字列をウィンドウの現在のカーソルの位置に追加します。

Format

```
#include <curses.h>
int addstr (char *str);
int waddstr (WINDOW *win, char *str);
```

Argument

win
ウィンドウを指すポインタ。

str
文字列を指すポインタ。

Description

waddstr関数をサブウィンドウで使用すると、文字列は親ウィンドウにも書き込まれます。

addstrルーチンはwaddstrと同じ機能を実行しますが、stdscrウィンドウに対して動作する点が異なります。

このルーチンを呼び出した結果、カーソルの位置は変化します。

Return value

OK	正常終了を示します。
ERR	関数を実行した結果、画面が不正にスクロールされることを示します。ただし、文字列は可能な範囲でウィンドウに書き込まれます。詳細については、scrollok関数を参照してください。

alarm

引数によって示される秒数が経過した後、シグナル SIGALRM (<signal.h>ヘッダ・ファイルに定義) を呼び出しプロセスに送信します。

Format

```
#include <unistd.h>

unsigned int alarm (unsigned int seconds); (ISO POSIX-1)
int alarm (unsigned int seconds); (Compatability)
```

引数

seconds
最大値は LONG_MAX 秒です。

Description

引数を 0 に設定してalarm関数を呼び出すと、保留中のアラームはすべて取り消されます。

alarmによって生成されたシグナルは、途中でキャッチされるか無視されない限り、プロセスを終了します。alarmを連続して呼び出すと、アラーム・クロックが再初期化されます。アラームはスタックされません。

クロックの分解能力は 1 秒であるため、シグナルは最大 1 秒、早く発生することがあります。SIGALRMシグナルがキャッチされると、スケジューリングの遅れのために実行の再開が遅れることがあります。

SIGALRMシグナルが生成されると、プロセスがハイバネート状態であるかどうかにかかわらず、SYSSWAKEの呼び出しが生成されます。保留中のウェイク処理が発生すると、現在のpause()はただちに戻ります (SIGALRM をキャッチした関数が終了した後)。

alarm

戻り値

n

前のアラーム要求から残っている秒数。

asctime, asctime_r

tm構造体に格納されている年月日時分秒形式の時刻を次の形式の 26 文字の文字列に変換します。

```
Sun Sep 16 01:03:52 1984\n\0
```

すべてのフィールドの幅は一定です。

Format

```
#include <time.h>

char *asctime (const struct tm *timeptr);
char *asctime_r (const struct tm *timeptr, char *buffer); (ISO POSIX-1)
```

Argument

timeptr

年月日時分秒形式の時刻を格納したtm型の構造体を指すポインタ。

tm構造体は<time.h>ヘッダ・ファイルに定義されており、localtimeの説明の表 REF-4 にも示されています。

buffer

26 バイト以上の長さの文字配列を指すポインタ。この配列は、生成された日付/時刻文字列を格納するために使用されます。

Description

asctime関数とasctime_r関数は、tmの内容を26 文字の文字列に変換し、その文字列を指すポインタを返します。

asctime_rとasctimeの相違点は、asctime_rが結果をユーザ指定バッファに格納するのに対し、asctimeはHP C RTL によって割り当てられたスレッド固有の静的メモリに結果を格納する点です。このスレッド固有の静的メモリは、ctimeまたはasctimeを再び呼び出すと、上書きされる可能性があります。保存する必要がある場合は、コピーを作成してください。

正常終了すると、asctimeは文字列を指すポインタを返します。asctime_rは2 番目の引数を返します。異常終了した場合は、これらの関数は NULL ポインタを返します。

tmのメンバの一覧については、localtime関数を参照してください。

注意

一般に、UTC ベースの時刻関数は、プロセス単位のデータであるメモリ内のタイム・ゾーン情報に影響を与える可能性があります。しかし、アプリケーションの実行中、システム・タイム・ゾーンが変化せず(これは一般的なケースです)、タイム・ゾーン・ファイルのキャッシュが有効に設定されている場合(これはデフォルトです)、時刻関数asctime_r, ctime_r, gmtime_r, localtime_rの_rバリエーションはスレッド・セーフで、かつASTリentrantです。

しかし、アプリケーションの実行中にシステム・タイム・ゾーンが変化する可能性がある場合や、タイム・ゾーン・ファイルのキャッシュが有効に設定されていない場合は、UTC ベースの時刻関数のバリエーションはいずれも第3の関数クラスに属し、これはスレッド・セーフでもASTリentrantでもありません。

Return value

x	正常終了した場合は、文字列を指すポインタ。
NULL	異常終了を示します。

asin

引数の逆正弦を返します。

Format

```
#include <math.h>

double asin (double x);
float asinf (float x); (Alpha, I64)
long double asinl (long double x); (Alpha, I64)
double asind (double x); (Alpha, I64)
float asindf (float x); (Alpha, I64)
long double asindl (long double x); (Alpha, I64)
```

引数

x
領域 $[-1,1]$ の実数値として表したラジアン値。

Description

asin関数は、領域 $[-1,1]$ のxに対して、 $[-\pi/2, \pi/2]$ ラジアンの範囲でxの逆正弦の主値を計算します。

asind関数は、領域 $[-1,1]$ のxに対して、 $[-90,90]$ 度の範囲でxの逆正弦の主値を計算します。

$\text{abs}(x)$ が 1.0 より大きい場合、 $\text{asin}(x)$ の値は 0 であり、`errno`は EDOM に設定されます。

asinh (Alpha, I64)

引数の双曲線逆正弦を返します。

Format

```
#include <math.h>

double asinh (double x);
float asinhf (float x);
long double asinhf (long double x);
```

引数

x
領域[−無限大,+ 無限大]の実数値で表したラジアン値。

Description

asinh関数は、領域[−無限大,+ 無限大]のxに対して、xの双曲線逆正弦を返します。ただし、 $\text{asinh}(x) = \ln(x + \sqrt{x^2 + 1})$ です。

asinh関数はsinhの逆関数であり、 $\text{asinh}(\sinh(x)) = x$ です。

assert

プログラムで実行時診断機能を実相するために使用します。

Format

```
#include <assert.h>

void assert (int expression);
```

引数

expression
int型の式。

Description

assertを実行したときに、*expression*が False である (つまり 0 に評価される) 場合は、assertは異常終了した特定の呼び出しに関する情報 (引数のテキスト、ソース・ファイルの名前、ソース行番号などで、最後の 2 つはそれぞれ前処理マクロ `__FILE__` および `__LINE__` の値) を、実装で定義されている形式で標準エラー・ファイルに書き込みます。その後、abort関数を呼び出します。

assert関数は次の形式でメッセージを書き込みます。

```
Assertion failed: expression, file aaa, line nnn
```

*expression*が True である (つまり 0 以外の値に評価される) 場合や、シグナル SIGABRT が無視されている場合は、assertは値を返しません。

注意

送出される式の一部にヌル文字 ('\0') が含まれている場合は、ヌル文字までのテキスト (ヌル文字を含む) だけが印刷されます。これは、ヌル文字が文字列の出力を終了するからです。

#include assert文より前にプリプロセッサ・ディレクティブ#define NDEBUG を指定するか、または CC コマンド修飾子/DEFINE=NDEBUG を指定してコンパイルすると、assert関数は無効になります。

例

```
#include <stdio.h>
#include <assert.h>

main()
{
    printf("Only this and the assert\n");
    assert(1 == 2);    /* expression is FALSE */

    /* abort should be called so the printf will not happen. */
    printf("FAIL abort did not execute");
}
```

atan

Format

```
#include <math.h>

double atan (double x);
float atanf (float x); (Alpha, I64)
long double atanl (long double x); (Alpha, I64)
double atand (double x); (Alpha, I64)
float atandf (float x); (Alpha, I64)
long double atandl (long double x); (Alpha, I64)
```

引数

x
実数値で表したラジアン値。

Description

atan関数は、 $[-\pi/2, \pi/2]$ ラジアンの範囲でxの逆正接の主値を計算します。

atand関数は、 $[-90, 90]$ 度の範囲でxの逆正接の主値を計算します。

atan2

Format

```
#include <math.h>

double atan2 (double y, double x);
float atan2f (float y, float x); (Alpha, I64)
long double atan2l (long double y, long double x); (Alpha, I64)
double atand2 (double y, double x); (Alpha, I64)
float atand2f (float y, float x); (Alpha, I64)
long double atand2l (long double y, long double x); (Alpha, I64)
```

Argument

y
実数値で表したラジアン値。

x
実数値で表したラジアン値。

Description

atan2関数は、 $[-\pi, \pi]$ ラジアンの範囲でy/xの逆正接の主値を計算します。atan2とatan2fの符号は、yの符号によって決定されます。atan2(y, x)の値は次のように計算されます。ただし、fはデータ型に関連付けられている小数ビットの数です。

入力引数の値	返される角度
$x = 0$ または $y/x > 2^{**}(f+1)$	$\pi/2 * (\text{sign } y)$
$x > 0$ および $y/x \leq 2^{**}(f+1)$	$\text{atan}(y/x)$
$x < 0$ および $y/x \leq 2^{**}(f+1)$	$\pi * (\text{sign } y) + \text{atan}(y/x)$

atand2関数は、 $[-180, 180]$ 度の範囲でy/xの逆正接の主値を計算します。atand2とatand2fの符号は、yの符号によって決定されます。

次の引数は, atan2関数およびatand2関数にとって不正な引数です。

関数	例外引数
atan2, atan2f, atan2l	$x = y = 0$
atan2, atan2f, atan2l	$ x = y = \text{無限大}$
atand2, atand2f, atand2l	$x = y = 0$
atand2, atand2f, atand2l	$ x = y = \text{無限大}$

atanh (Alpha, I64)

引数の双曲線逆正接を返します。

Format

```
#include <math.h>

double atanh (double x);
float atanhf (float x);
long double atanhl (long double x);
```

引数

x
領域 $[-1,1]$ の実数値として表したラジアン値。

Description

atanh関数はxの双曲線逆正接を返します。atanh関数はtanh関数の逆関数であり、 $\text{atanh}(\tanh(x)) = x$ です。

$|x| > 1$ は不正な引数です。

atexit

引数を指定せずに呼び出された関数をプログラムの終了時に登録します。

Format

```
#include <stdlib.h>
int atexit (void (*func) (void));
```

引数

func
登録する関数を指すポインタ。

Return value

0	登録が正常終了したことを示します。
0 以外の値	異常終了を示します。

制限事項

longjmp関数をハンドラの内部から実行することはできません。これは、longjmpの宛先アドレスがもはや存在しないからです。

例

```
#include <stdlib.h>
#include <stdio.h>

static void hw(void);

main()
{
    atexit(hw);
}

static void hw()
{
    puts("Hello, world\n");
}
```

この例では、次の出力が生成されます。

```
Hello, world
```

atof

ASCII 文字列を倍精度数値に変換します。

Format

```
#include <stdlib.h>

double atof (const char *nptr);
```

引数

nptr
倍精度数値に変換する文字列を指すポインタ。文字列は、浮動小数点定数を解釈するために使用される規則と同じ規則で解釈されます。

Description

変換される文字列の形式は次のとおりです。

[white-spaces][+|-]digits[radix-character][digits][e|E[+|-]integer]

ただし、radix-characterは現在のロケールで定義されます。

認識されない最初の文字が検出されると、変換は終了します。

この関数はstrtod(nptr, (char**) NULL)と同じです。

Return value

x	変換後の値。
0	アンダフローまたは変換を実行できないことを示します。この関数はerrnoを ERANGE または EINVAL に設定します。
±HUGE_VAL	オーバーフローが発生しました。errnoは ERANGE に設定されます。

atoi, atol

ASCII 文字列を適切な数値に変換します。

Format

```
#include <stdlib.h>

int atoi (const char *nptr);

long int atol (const char *nptr);
```

引数

nptr
数値に変換する文字列を指すポインタ。

Description

atoi関数とatol関数は、文字列の最初の部分をそれぞれ10進数のint値およびlong int値に変換します。atoi関数とatol関数は、変換で発生するオーバーフローを考慮しません。変換される文字列は次の形式です。

```
[white-spaces][+ | -]digits
```

関数呼び出し `atoi (str)` は `strtol (str, (char**)NULL, 10)` と同じであり、関数呼び出し `atoi (str)` は `(int) strtol (str, (char**)NULL, 10)` と同じです。ただし、どちらの場合もエラー発生時の動作が異なります。

戻り値

n	変換後の値。
1	1
2	1
3	2
4	2
5	3
6	3
7	4
8	4
9	5
10	5
11	6
12	6
13	7
14	7
15	8
16	8
17	9
18	9
19	10
20	10
21	11
22	11
23	12
24	12
25	13
26	13
27	14
28	14
29	15
30	15
31	16
32	16
33	17
34	17
35	18
36	18
37	19
38	19
39	20
40	20
41	21
42	21
43	22
44	22
45	23
46	23
47	24
48	24
49	25
50	25
51	26
52	26
53	27
54	27
55	28
56	28
57	29
58	29
59	30
60	30
61	31
62	31
63	32
64	32
65	33
66	33
67	34
68	34
69	35
70	35
71	36
72	36
73	37
74	37
75	38
76	38
77	39
78	39
79	40
80	40
81	41
82	41
83	42
84	42
85	43
86	43
87	44
88	44
89	45
90	45
91	46
92	46
93	47
94	47
95	48
96	48
97	49
98	49
99	50
100	50

Format

引数

Description

関数呼び出し `atof (str)` は `strtod (str, (char**)NULL, 10)` と同じですが、エラー発生時の動作が異なります。

戻り値

REF-29

basename

パス名の最後のコンポーネントを返します。

Format

```
#include <libgen.h>

char *basename (char *path);
```

関数バリエーション

basename関数には、_basename32および_basename64という名前のバリエーションがあり、それぞれ 32 ビット・ポインタ・サイズおよび 64 ビット・ポインタ・サイズで使用されます。ポインタ・サイズ固有の関数の使い方については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル(上巻)』第 1.10 節を参照してください。

引数

path
ベース・パス名を抽出する UNIX 形式のパス名。

Description

basename関数は、pathによって示される UNIX 形式のパス名を引数として受け付け、後続のスラッシュ(/)文字を削除して、パス名の最後のコンポーネントを指すポインタを返します。

pathがスラッシュ(/)文字だけで構成されている場合は、この関数は文字列 "/" を指すポインタを返します。

pathが NULL ポインタである場合や、空文字列を指す場合は、文字列 "." を指すポインタを返します。

basename関数はpathによって示される文字列を変更する可能性があります。

Return value

<code>x</code>	pathの最後のコンポーネントを指すポインタ。
<code>"/"</code>	pathが <code>'/'</code> 文字だけで構成されている場合。
<code>""</code>	pathが NULL ポインタであるか、または空文字列を指す場合。

bcmp

バイト列を比較します。

Format

```
#include <strings.h>

void bcmp (const void *string1, const void *string2, size_t length);
```

Argument

string1, string2
比較するバイト列。

length
バイト列の長さ (バイト数)。

Description

bcmp関数は、string1のバイト列をstring2のバイト列と比較します。

文字列関数と異なり、ヌル・バイトの確認は行われません。長さが0のバイト列は常に同一であると解釈されます。

bcmpは、ANSI C標準で定義されているmemcmpに相当します。したがって、移植性の高いプログラムを実現するには、memcmpを使用することをお勧めします。

Return value

0	バイト列は同一です。
0 以外の値	バイト列は同一ではありません。

bcopy

バイト列をコピーします。

Format

```
#include <strings.h>

void bcopy (const void *source, void *destination, size_t length);
```

Argument

source
コピー元バイト列を指すポインタ。

destination
コピー先バイト列を指すポインタ。

length
バイト列の長さ (バイト数)。

Description

bcopy関数は、可変長バイト列に対して機能します。length引数の値 (バイト数) によって指定される長さのバイト列をsource引数のバイト列からdestination引数のバイト列にコピーします。

文字列関数と異なり、ヌル・バイトの確認は行われません。length引数が0 (ゼロ) の場合、バイト列はコピーされません。

bcopyは、ANSI C標準で定義されているmemcpyに相当します。したがって、移植性の高いプログラムを実現するには、memcpyを使用することをお勧めします。

box

ウィンドウの周囲に四角形を描きます。四角形の縦線を描く文字として *vert* 文字を使用し、四角形の横線を描く文字として *hor* 文字を使用します。

Format

```
#include <curses.h>

int box (WINDOW *win, char vert, char hor);
```

Argument

win
ウィンドウのアドレス。

vert
ウィンドウの縦線を描く文字。

hor
ウィンドウの横線を描く文字。

Description

*box*関数は、サブウィンドウに描いた四角形を親ウィンドウにコピーします。サブウィンドウに四角形が描かれた状態で、*overlay*や*overwrite*などの関数を使用する場合は注意が必要です。このような関数は、四角形を親ウィンドウにコピーします。

Return value

OK	正常終了を示します。
ERR	エラーを示します。

brk

プログラムで使用されていない最小の仮想アドレスを判断します。

Format

```
#include <stdlib.h>

void *brk (unsigned long int addr);
```

引数

addr

最小アドレス。このアドレスがページ・サイズの次の倍数に切り上げられます。この切り上げられたアドレスを区切りアドレスと呼びます。

Description

区切りアドレス値に等しいかそれより大きく、スタック・ポインタより小さいアドレスは、プログラムのアドレス空間の外部にあると解釈されます。このようなアドレスを参照しようとする、アクセス違反が発生します。

プログラムを実行する場合、区切りアドレスは、プログラム記憶域およびデータ記憶域によって定義される最大記憶位置に設定されます。したがって、brk関数は、データ領域を拡大するプログラムでのみ必要です。

Return value

n

(void *)(-1)

新しい区切りアドレス。

プログラムが要求しているメモリ容量が多すぎることを示します。errnoとvaxc\$errnoが更新されます。

制限事項

他の C ライブラリの実装と異なり、HP C RTL のメモリ割り当て関数 (malloc など) では、プログラム・ヒープ空間の管理で brk や sbrk に依存していません。したがって、OpenVMS システムで brk または sbrk を呼び出すと、メモリ割り当てルーチンを妨害する可能性があります。brk 関数と sbrk 関数が提供されるのは、互換性を維持するためだけです。

bsearch

バイナリ検索を実行します。ソートされたオブジェクトの配列から指定のオブジェクトを検索します。

Format

```
#include <stdlib.h>

void *bsearch (const void *key, const void *base, size_t nmemb, size_t size, int (*compar)
               (const void *, const void *));
```

関数バリエーション

bsearch関数には、_bsearch32および_bsearch64という名前のバリエーションがあり、それぞれ 32 ビット・ポインタ・サイズおよび 64 ビット・ポインタ・サイズで使われます。ポインタ・サイズ固有の関数の使い方については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

Argument

key

配列内で検索するオブジェクトを指すポインタ。このポインタは、pointer-to-object 型で、pointer-to-void 型にキャストされなければなりません。

base

配列の最初のメンバを指すポインタ。このポインタは pointer-to-object 型で、pointer-to-void 型にキャストされなければなりません。

nmemb

配列内のオブジェクトの数。

size

オブジェクトのサイズ (バイト数)。

compar

比較関数を指すポインタ。

Description

配列はまず, *compar*によって示される比較関数に従って, 昇順にソートする必要があります。

*compar*によって示される比較関数に 2 つの引数が渡されます。2 つの引数は比較するオブジェクトを示します。最初の引数が 2 番目の引数より小さいか, 等しいか, 大きいかに応じて, 比較関数は 0 より小さい整数, 0, 0 より大きい引数を返さなければなりません。

比較関数 (*compar*) が配列内のすべてのバイトを比較する必要はありません。したがって, 配列内のオブジェクトには, 比較するデータの他に任意のデータを格納することができます。

返される値は *pointer-to-void* 型として宣言されるため, *pointer-to-object* 型にキャストされるかまたは割り当てられなければなりません。

Return value

<i>x</i>	配列内で一致するメンバを指すポインタ。一致するメンバが見つからない場合はヌル・ポインタ。
NULL	キーが配列から見つからなかったことを示します。

例

```
#include <stdio.h>
#include <stdlib.h>

#define SSIZE 30

extern int compare(); /* prototype for comparison function */

int array[SSIZE] = {30, 1, 29, 2, 28, 3, 27, 4, 26, 5,
                   24, 6, 23, 7, 22, 8, 21, 9, 20, 10,
                   19, 11, 18, 12, 17, 13, 16, 14, 15, 25};

/* This program takes an unsorted array, sorts it using qsort, */
/* and then calls bsearch for each element in the array,      */
/* making sure that bsearch returns the correct element.      */
main()
{
    int i;
    int failure = FALSE;
    int *rkey;

    qsort(array, SSIZE, sizeof (array[0]), &compare);
```

```

/* search for each element */
for (i = 0; i < SSIZE - 1; i++) {
    /* search array element i */
    rkey = bsearch((array + i), array, SSIZE,
                  sizeof(array[0]), &compare);
    /* check for successful search */
    if (&array[i] != rkey) {
        printf("Not in array, array element %d\n", i);
        failure = TRUE;
        break;
    }
}
if (!failure)
    printf("All elements successfully found!\n");
}

/* Simple comparison routine. */
/*                               */
/* Returns:  = 0 if a == b      */
/*           < 0 if a < b       */
/*           > 0 if a > b       */
/*                               */
int compare(int *a, int *b)
{
    return (*a - *b);
}

```

このサンプル・プログラムでは、次の出力が生成されます。

```
All elements successfully found!
```

btowc

初期シフト状態で 1 バイトのマルチバイト文字をワイド文字に変換します。

Format

```
#include <wchar.h>
wint_t btowc (int c);
```

引数

c
ワイド文字表現に変換する文字。

Description

btowc関数は、(unsigned char)cが初期シフト状態で有効な 1 バイトのマルチバイト文字であるかどうかを確認し、有効である場合は、その文字のワイド文字表現を返します。

Return value

x	unsigned char cのワイド文字表現。
WEOF	エラーを示します。c引数の値が EOF であるか、または初期シフト状態で有効な 1 バイトのマルチバイト文字ではありません。

bzero

ヌル文字をバイト列にコピーします。

Format

```
#include <strings.h>
void bzero (void *string, size_t length);
```

Argument

string

ヌル文字のコピー先のバイト列を指定します。

length

文字列の長さ (バイト数) を指定します。

Description

bzero関数は、ヌル文字 ('\0') をlengthバイトだけ、stringによって示されるバイト列にコピーします。lengthが0の場合は、バイトはコピーされません。

cabs

複素数の絶対値を返します。

Format

```
#include <math.h>

double cabs (cabs_t z);
float cabsf (cabsf_t z); (Alpha, I64)
long double cabsl (cabsl_t z); (Alpha, I64)
```

引数

z
cabs_t型, cabsf_t型, またはcabsl_t型の構造体。これらの型は次に示すように, <math.h>ヘッダ・ファイルに定義されています。

```
typedef struct {double x,y;} cabs_t;
typedef struct { float x, y; } cabsf_t; (Alpha, I64)
typedef struct { long double x, y; } cabsl_t; (Alpha, I64)
```

Description

cabs関数は, 2点間のユークリッド距離をそれぞれの2乗の平方根として計算することにより, 複素数の絶対値を返します。

$\sqrt{x^2 + y^2}$

オーバーフローが発生した場合, 戻り値は未定義です。

cabs関数, cabsf関数, cabsl関数はそれぞれ, hypot関数, hypotf関数, hypotl関数と同じです。

cacos (Alpha, I64)

引数 (複素数) の逆余弦を返します。

Format

```
#include <complex.h>
double complex cacos (double complex z);
float complex cacosf (float complex z);
long double complex cacosl (long double complex z);
```

引数

z
複素数値。

Description

cacos関数は、実軸上の区間 $[-1, +1]$ の外側に分岐線法を適用して、複素数 z の逆余弦を計算します。

Return value

n
複素数の逆余弦値。この値は、虚軸上では数学的に無限の範囲、また実軸上では区間 $[0, \pi]$ の範囲からなる帯状の領域にあります。

cacosh (*Alpha, I64*)

引数 (複素数) の双曲線逆余弦を返します。

Format

```
#include <complex.h>

double complex cacosh (double complex z);
float complex cacoshf (float complex z);
long double complex cacoshl (long double complex z);
```

引数

z
複素数値。

Description

cacosh関数は、実軸上の 1 より小さい値に分岐線法を適用して、複素数*z*の双曲線逆余弦を計算します。

Return value

n 複素数の双曲線逆余弦値。この値は、実軸上では値が負でない範囲、また虚軸上では区間 $[-i\pi, +i\pi]$ の範囲からなる帯状の領域にあります。

calloc

0 に初期化されたメモリ領域を割り当てます。この関数は AST リエントラントです。

Format

```
#include <stdlib.h>

void *calloc (size_t number, size_t size);
```

関数バリエーション

calloc 関数には、_calloc32 および _calloc64 という名前のバリエーションがあり、それぞれ 32 ビット・ポインタ・サイズ および 64 ビット・ポインタ・サイズ で使用されます。ポインタ・サイズ 固有の関数の使い方の詳細については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

Argument

number
割り当てる項目の数。

size
各項目のサイズ。

Description

calloc 関数は項目を 0 に初期化します。

malloc と realloc も参照してください。

Return value

x	クォードワード境界 (<i>Alpha only</i>) または オクタワード境界 (<i>I64 only</i>) にそろえられた最初のバイトのアドレス。
NULL	領域を割り当てることができないことを示します。

carg (Alpha, I64)

引数 (複素数) の位相角を返します。

Format

```
#include <complex.h>
double carg (double complex z);
float cargf (float complex z);
long double cargl (long double complex z);
```

引数

z
複素数値。

Description

carg関数は、実軸上の負の領域に分岐線法を適用して、zの位相角 (偏角ともいいます) を計算します。

Return value

n zの位相角値。この値は区間 $[-\pi, +\pi]$ の範囲にあります。

casin (Alpha, I64)

引数 (複素数) の逆正弦を返します。

Format

```
#include <complex.h>
double complex casin (double complex z);
float complex casinf (float complex z);
long double complex casinl (long double complex z);
```

引数

z
複素数値。

Description

casin関数は、実軸上の区間 $[-1, +1]$ の外側に分岐線法を適用して、複素数 z の逆正弦を計算します。

Return value

n
複素数の逆正弦値。この値は、虚軸では数学的に無限の範囲、また実軸上では区間 $[-\pi/2, +\pi/2]$ の範囲からなる帯状の領域にあります。

casinh (Alpha, I64)

引数 (複素数) の双曲線逆正弦を返します。

Format

```
#include <complex.h>

double complex casinh (double complex z);
float complex casinhf (float complex z);
long double complex casinhl (long double complex z);
```

引数

z
複素数値。

Description

casinh関数は、虚軸上の区間 $[-i, +i]$ の外側に分岐線法を適用して、複素数 z の双曲線逆正弦を計算します。

Return value

n
複素数の双曲線逆正弦値。この値は、実軸上では数学的に無限の範囲、また虚軸上では区間 $[-i\pi/2, +i\pi/2]$ の範囲からなる帯状の領域にあります。

catan (Alpha, I64)

引数 (複素数) の逆正接を返します。

Format

```
#include <complex.h>

double complex catan (double complex z);
float complex catanf (float complex z);
long double complex catanl (long double complex z);
```

引数

z
複素数値。

Description

catan関数は、虚軸上の区間 $[-i, +i]$ の外側に分岐線法を適用して、複素数zの逆正接を計算します。

Return value

n
複素数の逆正接値。この値は、虚軸上では数学的に無限の範囲、また実軸上では区間 $[-\pi/2, +\pi/2]$ の範囲からなる帯状の領域にあります。

catanh (Alpha, I64)

引数 (複素数) の双曲線逆正接を返します。

Format

```
#include <complex.h>

double complex catanh (double complex z);
float complex catanhf (float complex z);
long double complex catanhl (long double complex z);
```

引数

z
複素数値。

Description

catanh関数は、虚軸上の区間 $[-1, +1]$ の外側に分岐線法を適用して、複素数 z の双曲線逆正接を計算します。

Return value

n
複素数の双曲線逆正接値。この値は、実軸上では数学的に無限の範囲、また虚軸上では区間 $[-i\pi/2, +i\pi/2]$ の範囲からなる帯状の領域にあります。

catclose

メッセージ・カタログをクローズします。

Format

```
#include <nl_types.h>
int catclose (nl_catd catd);
```

引数

catd

メッセージ・カタログ記述子。この記述子は、catopenの呼び出しが正常終了したときに返されます。

Description

catclose関数は、catdによって参照されるメッセージ・カタログをクローズし、カタログ・ファイル記述子の割り当てを解除します。

Return value

- | | |
|----|--|
| 0 | カタログが正しくクローズされたことを示します。 |
| -1 | エラーが発生したことを示します。errnoは次の値に設定されます。 <ul style="list-style-type: none">• EBADF— カタログ記述子が不正です。 |

catgets

メッセージ・カタログからメッセージを取得します。

Format

```
#include <nl_types.h>

char *catgets (nl_catd catd, int set_id, int msg_id, const char *s);
```

関数バリエーション

catgets関数には、_catgets32および_catgets64という名前のバリエーションがあり、それぞれ 32 ビット・ポインタ・サイズおよび 64 ビット・ポインタ・サイズで使用されます。ポインタ・サイズ固有の関数の使い方については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

Argument

catd

メッセージ・カタログ記述子。この記述子は、catopenの呼び出しが正常終了したときに返されます。

set_id

整数のセット識別子。

msg_id

整数のメッセージ識別子。

s

メッセージを取得できないときに関数から返されるデフォルトのメッセージ文字列を指すポインタ。

Description

catgets関数は、メッセージ・カタログcatdから、set_idとmsg_idによって示されるメッセージを取得します。メッセージはnl_catd構造体のメッセージ・バッファに格納されます。catgetsを再び呼び出すと、格納されているメッセージは上書きされます。メッセージ文字列を保存する必要がある場合は、プログラムで別の場所にコピーしてください。

Return value

x 取得したメッセージを指すポインタ。

s デフォルト・メッセージ文字列を指すポインタ。関数が要求されたメッセージをカタログから取得できなかったことを示します。指定された引数の組み合わせ (set_d, msg_id) がオープンされているカタログで既存のメッセージを表さない場合は、この条件が発生することがあります。また、エラーが発生したことを示す場合もあります。エラーが発生した場合は、この関数はerrnoを次のいずれかの値に設定します。

- EBADF— カタログ記述子が不正です。
- EVMSRR—OpenVMS I/O 読み込みエラー。
OpenVMS エラー・コードはvaxc\$errnoに格納されています。

例

```
#include <nl_types.h>
#include <locale.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <unixio.h>

/* This test makes use of all the message catalog routines. catopen */
/* opens the catalog ready for reading, then each of the three */
/* messages in the catalog are extracted in turn using catgets and */
/* printed out. catclose closes the catalog after use. */
/* The catalog source file used to create the catalog is as follows: */
/* $ This is a message file
* $
* $quote "
* $ another comment line
* $set 1
* 1 "First set, first message"
* 2 "second message - This long message uses a backslash \
* for continuation."
* $set 2
* 1 "Second set, first message" */

char *default_msg = "this is the first message.";

main()
{
    nl_catd catalog;
    int msg1,
        msg2,
        retval;

    char *cat = "sys$disk:[]catgets_example.cat"; /*Force local catalog*/
```

```

char *msgtxt;
char string[128];
/* Create the message test catalog */
system("gencat catgets_example.msgx catgets_example.cat" );
if ((catalog = catopen(cat, 0)) == (nl_catd) - 1) {
    perror("catopen");
    exit(EXIT_FAILURE);
}

msgtxt = catgets(catalog, 1, 1, default_msg);
printf("%s\n", msgtxt);

msgtxt = catgets(catalog, 1, 2, default_msg);
printf("%s\n", msgtxt);

msgtxt = catgets(catalog, 2, 1, default_msg);
printf("%s\n", msgtxt);

if ((retval = catclose(catalog)) == -1) {
    perror("catclose");
    exit(EXIT_FAILURE);
}

delete("catgets_example.cat;") ; /* Remove the test catalog */
}

```

このサンプル・プログラムを実行すると、次の結果が生成されます。

```

First set, first message
second message - This long message uses a backslash for continuation.
Second set, first message

```

catopen

メッセージ・カタログをオープンします。

Format

```
#include <nl_types.h>

nl_catd catopen (const char *name, int oflag);
```

Argument

name

オープンするメッセージ・カタログの名前。

oflag

int型のオブジェクトであり、カタログ・ファイルの検索で、現在のプログラムのローケルのLC_MESSAGESカテゴリのローケル・セットを使用するのか、論理名LANGを使用するのかを指定します。

Description

catopen関数は、nameによって示されるメッセージ・カタログをオープンします。

nameにコロン(:)、左角括弧([)、左山括弧(<)のいずれかが含まれている場合や、nameが論理名として定義されている場合は、nameはカタログの完全なファイル指定であると解釈されます。

これらの文字が含まれていない場合、catopenは、nameが既存のカタログ・ファイルを指す論理名であると解釈します。nameが論理名でない場合は、論理名NLSPATHを使用して、メッセージ・カタログのファイル指定を定義します。NLSPATHはユーザのプロセスで定義されます。NLSPATH 論理名が定義されていない場合や、NLSPATHによって指定されるどのコンポーネントでもメッセージ・カタログをオープンできない場合は、SYS\$NLSPATH 論理名を使用してメッセージ・カタログ・ファイルを検索します。

NLSPATH と SYS\$NLSPATH はどちらもコンマ区切りのテンプレート・リストです。catopen関数は各テンプレートを使用してファイル指定を作成します。たとえば、NLSPATH は次のように定義することができます。

```
DEFINE NLSPATH SYS$SYSROOT:[SYS$I18N.MSG]%N.CAT,SYS$COMMON:[SYSMSG]%N.CAT
```

この例では、catopenはまず、ディレクトリ SYSSYSROOT:[SYSSI18N.MSG]から指定されたカタログを検索します。指定されたカタログが見つからない場合は、ディレクトリ SYSSCOMMON:[SYSMSG]を検索します。カタログ名は、%N をcatopenに渡された名前に置き換え、接尾語.cat を追加することにより作成されます。%N は置換フィールドと呼ばれます。次の置換フィールドは有効です。

フィールド	意味
%N	catopenに渡されたnameを置き換える。
%L ¹	ロケール名を置き換える。 ロケール名で使用されているピリオド(.)とアットマーク記号(@)は、下線(_)文字に置き換えられる。 たとえば、ロケール名が"zh_CN.dechanzi@radical"の場合は、ZH_CN_DECHANZI_RADICALに置換される。
%l ¹	ロケール名のlanguageの部分を置き換える。たとえば、en_GB.ISO8859-1というロケール名で language の部分は en である。
%t ¹	ロケール名のterritoryの部分を置き換える。たとえば、en_GB.ISO8859-1というロケール名で territory の部分は GB である。
%c ¹	ロケール名のcodeset名を置き換える。たとえば、en_GB.ISO8859-1というロケール名で codeset 名はISO8859-1である。

¹この置換では、ロケール名がlanguage_territory.codeset @modeという形式であるものと解釈される。

oflag引数が NL_CAT_LOCALE に設定されている場合は、LC_MESSAGES カテゴリに対して定義されている現在のロケールを使用して、%L、%l、%t、%c 置換フィールドの置換を判断します。oflag引数が 0 に設定されている場合は、LANG 環境変数の値をロケール名として使用して、これらのフィールドの置換を判断します。NL_CAT_LOCALE の使用は XPG4 仕様に準拠していますが、XPG3 との互換性を維持するために 0 という値が存在することに注意してください。また、catopenでは LANG 環境変数の値を使用しますが、この値を使用してプログラムのロケールを設定できるかどうかの確認は行いません。つまり、catopenはsetlocale呼び出しで、この値が有効なロケール引数として機能するかどうかを確認しません。

置換値が定義されていない場合は、空文字列に置換されます。

先頭のコンマまたは 2 つの隣接するコンマ (,,) は %N の指定に相当します。次の例を参照してください。

```
DEFINE NLSPATH ",%N.CAT,SYS$COMMON:[SYSMSG.%L]%N.CAT"
```

この例では、catopenは次の場所をここに示した順に検索します。

1. name (カレント・ディレクトリ)
2. name.cat (カレント・ディレクトリ)
3. SYS\$COMMON:[SYSMSG.locale_name]name.cat

Return value

x	メッセージ・カタログ・ファイル記述子。呼び出しが正常終了したことを示します。この記述子はcatgetsおよびcatcloseの呼び出しで使用されます。
(nl_catd) - 1	<p>エラーが発生したことを示します。errnoは次のいずれかの値に設定されます。</p> <ul style="list-style-type: none">• EACCES— 特権が不十分であるか、ファイル保護違反が発生したか、ファイルが現在別のユーザによってロックされています。• EMFILE— プロセス・チャンネル・カウントを超過しました。• ENAMETOOLONG— メッセージ・カタログの完全なファイル指定が長すぎます。• ENOENT— 要求されたメッセージ・カタログを見つけることができませんでした。• ENOMEM— 空きメモリが不足しています。• ENOTDIR— name引数の一部が有効なディレクトリではありません。• EVMSERR— errnoのどの値にも一致しないエラーが発生しました。vaxc\$errnoの値を確認してください。

cbrt (*Alpha, I64*)

cbrt (*Alpha, I64*)

yの丸められた立方根を返します。

Format

```
#include <math.h>
double cbrt (double y);
float cbrtf (float y);
long double cbrtl (long double y);
```

引数

y
実数値。

```
#include <complex.h>
double complex ccos (double complex z);
float complex ccosf (float complex z);
long double complex ccosl (long double complex z);
```

z
複素数值。

ccos関数は、複素数 z の余弦を返します。

x 複素数の余弦値。

ccosh (Alpha, I64)

引数 (複素数) の双曲線余弦を返します。

Format

```
#include <complex.h>
double complex ccosh (double complex z);
float complex ccoshf (float complex z);
long double complex ccoshl (long double complex z);
```

引数

z
複素数値。

Description

ccosh関数は、複素数zの双曲線余弦を返します。

Return value

x
複素数の双曲線余弦値。

Format

引数

戻り値

REF-61

`cexp` (*Alpha, I64*)

`cexp` (*Alpha, I64*)

引数 (複素数) の指数関数値を返します。

Format

```
#include <complex.h>
double complex cexp (double complex z);
float complex cexpf (float complex z);
long double complex cexpl (long double complex z);
```

引数

`z`
複素数値。

Description

`cexp`関数は、複素数`z`の指数関数値 ($e^{**}z$, つまり自然対数の底 e を`z`乗した値) を計算します。

Return value

`x` 引数 (複素数) の指数関数値。

cfree

前に呼び出した`calloc` , `malloc` , `realloc`によって割り当てられた領域を再割り当て可能な状態にします。この関数はASTリエントラントです。

Format

```
#include <stdlib.h>

void cfree (void *ptr);
```

引数

`ptr`
前に呼び出した`malloc` , `calloc` , `realloc`から返されたアドレス。

Description

割り当てが解除される領域の内容は変更されません。

HP C for OpenVMSシステムでは、`free`関数と`cfree`関数は同じです。他の一部のCの実装では、`malloc`または`realloc`に対しては`free`を使用し、`calloc`に対しては`cfree`を使用します。しかし、ANSI C標準には`cfree`が含まれていないため、`free`を使用する方が適切です。

`free`も参照してください。

chdir

デフォルト・ディレクトリを変更します。

Format

```
#include <unistd.h>

int chdir (const char *dir_spec); (ISO POSIX-1)

int chdir (const char *dir_spec, ...); (HP C Extension)
```

Argument

dir_spec

OpenVMS 形式または UNIX 形式の指定でディレクトリ名を指定するヌル区切り文字列。

...

この引数は、標準関連の機能テスト・マクロ (『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.5 節を参照) が定義されておらず、厳密な ANSI C モード (/STANDARD=ANSI89) でコンパイルしていないときに使用できる HP C の拡張機能です。引数は int 型の省略可能なフラグであり、USER モードから chdir を呼び出す場合にのみ有効です。

フラグの値が 1 の場合は、新しいディレクトリは複数のイメージ間で有効です。値が 1 でない場合は、イメージが終了するときに元のデフォルト・ディレクトリが復元されます。

Description

chdir関数はデフォルト・ディレクトリを変更します。永久的な変更または一時的な変更が可能です。永久的な変更とは、イメージが終了した後も、新しいディレクトリがデフォルト・ディレクトリとして有効になることを示します。一時的な変更とは、イメージが終了したときに、デフォルト・ディレクトリがイメージの実行前のディレクトリに戻されることを示します。

永久的な変更は次の 2 種類の方法で行うことができます。

- 2 番目の引数を 1 に設定して、USER モードから chdir を呼び出す方法。
- 2 番目の引数の値にかかわらず、SUPERVISOR モードまたは EXECUTIVE モードから chdir を呼び出す方法。

この 2 種類の方法以外の場合は、変更は一時的になります。

Return value

0	指定された名前にディレクトリを変更する処理が正常終了したことを示します。
-1	変更が失敗したことを示します。

chmod

ファイルの保護を変更します。

Format

```
#include <stat.h>

int chmod (const char *file_spec, mode_t mode);
```

Argument

file_spec

OpenVMS 形式または UNIX 形式のファイル指定の名前。

mode

ファイル保護。モードは、表 REF-2 に示した値のビット単位の論理和 (OR) を求めることにより作成されます。

表 REF-2 ファイル保護の値とその意味

値	特権
0400	OWNER:READ
0200	OWNER:WRITE
0100	OWNER:EXECUTE
0040	GROUP:READ
0020	GROUP:WRITE
0010	GROUP:EXECUTE
0004	WORLD:READ
0002	WORLD:WRITE
0001	WORLD:EXECUTE

modeの値が0の場合は、chmod関数はファイルにユーザのデフォルト・ファイル保護を割り当てます。

システムにはオーナーと同じ特権が与えられます。WRITE 特権を割り当てると、DELETE 特権も暗黙に割り当てられます。

Description

ファイルのモードを変更するには、そのファイルに対して WRITE 特権が必要です。

C RTL では、S_ISVTX ビットをサポートしていません。したがって、S_ISVTX モードを設定しても効果はありません。

Return value

0	モードが正しく変更されたことを示します。
-1	変更が失敗したことを示します。

chown

指定したファイルのユーザ ID とグループ ID を変更します。

Format

```
#include <unistd.h>

int chown (const char *file_spec, uid_t owner, gid_t group);
```

Argument

file_spec
ASCII ファイル名のアドレス。

owner
ファイルの新しいユーザ ID。

group
ファイルの新しいグループ ID。

Return value

0	正常終了を示します。
-1	異常終了を示します。

cimag (*Alpha, I64*)

引数 (複素数) の虚数部を返します。

Format

```
#include <complex.h>
double cimag (double complex z);
float cimagf (float complex z);
long double cimagl (long double complex z);
```

引数

Z
複素数值。

Description

cimag関数は、 z の虚数部を実数として返します。

Return value

x 虚数部の値。

[w]clear

[w]clear

指定されたウィンドウの内容を消去し、カーソルを座標 (0,0) にリセットします。clear関数はstdscrウィンドウに対して動作します。

Format

```
#include <curses.h>

int clear();

int wclear (WINDOW *win);
```

引数

win
ウィンドウを指すポインタ。

Return value

OK	正常終了を示します。
ERR	エラーを示します。

clearerr

ファイルのエラー指示子および EOF 指示子をリセットします (したがって, `ferror` と `feof` は 0 以外の値を返しません)。

Format

```
#include <stdio.h>

void clearerr (FILE *file_ptr);
```

引数

`file_ptr`
ファイル・ポインタ。

clearerr_unlocked (Alpha, I64)

clearerr関数と同様ですが、flockfileとfunlockfileで保護された範囲内だけで使用します。

Format

```
#include <stdio.h>

void clearerr_unlocked (FILE *file_ptr);
```

引数

file_ptr
ファイル・ポインタ。

Description

リエントラント版であるclearerr関数は、複数スレッドからの同時呼び出しに対してロックされます。その結果、ストリームの一貫性を保証するためのオーバーヘッドが生じます。アンロック版であるclearerr_unlockedを使用すると、このオーバーヘッドを避けることができます。clearerr_unlockedマクロは、clearerrマクロと機能的に同じですが、スレッド・セーフな方法で実装する必要がない点が異なります。clearerr_unlocked関数は、flockfile関数とfunlockfile関数を対で使用して保護された範囲内だけで、安全に使用することができます。呼び出し元は、clearerr_unlockedを使用する前に、ストリームを確実にロックする必要があります。

flockfile、ftrylockfile、およびfunlockfileも参照してください。

clearok

ウィンドウのクリア・フラグを設定します。

Format

```
#include <curses.h>
clearok (WINDOW *win, bool boolf);
```

Argument

win

端末画面全体のサイズ。clearok関数はウィンドウstdscrおよびcurscrに対して使用できます。

boolf

論理値 TRUE または FALSE。引数が TRUE の場合は、refreshを次回呼び出したときに、クリアスクリーンが出力されます。boolfが FALSE の場合は、画面のクリアは停止されます。

bool型は、次に示すように<curses.h>ヘッダ・ファイルに定義されています。

```
#define bool int
```

Description

clear関数と異なり、clearok関数はウィンドウの内容を変更しません。win引数がcurscrの場合は、refreshを次に呼び出したときに、refreshに渡されたウィンドウが端末画面全体のサイズのウィンドウでない場合でも、クリアスクリーンが実行されます。

clock

プロセスの開始時から使用された CPU 時間 (10 ミリ秒単位) を返します。報告される時間は呼び出しプロセスのユーザ時間とシステム時間の合計，および呼び出しプロセスがwaitまたはsystemを実行した，終了済み子プロセスの時間の合計です。

Format

```
#include <time.h>

clock_t clock (void);
```

Description

秒単位の時間を取得するには，clock関数から返された値を，標準ヘッダ・ファイル<time.h>に定義されている CLK_TCK の値で除算する必要があります。

clock_t型は，次に示すように<time.h>ヘッダ・ファイルに定義されています。

```
typedef long int clock_t;
```

C メイン・プログラムや，VAXC\$CRTL_INIT または DECC\$CRTL_INIT を呼び出すプログラムを実行する子プロセスの累積時間だけが含まれます。

clock関数は通常，プログラムが初期設定を終了した後呼び出され，時間を測定する対象となるコードをプログラムが実行した後，再び呼び出されます。その後，2つの値の差を求めることにより，経過 CPU 時間が算出されます。

Return value

n	使用されたプロセッサ時間。
-1	使用されたプロセッサ時間を取得できなかったことを示します。

clock_getres (Alpha, I64)

指定されたクロックの精度を取得します。

Format

```
#include <time.h>

int clock_getres (clockid_t clock_id, struct timespec *res);
```

Argument

clock_id

精度を取得するクロックのタイプ。CLOCK_REALTIME クロックがサポートされています。このクロックは、システムの TIME-OF-DAY クロックです。

res

クロックの精度の値を受け取る、timespecデータ構造体へのポインタです。

Description

clock_getres関数は、指定されたクロックの精度の値を取得します。クロックの精度は、実装によって決まり、プロセスが設定することはできません。

res引数が NULL でない場合、指定されたクロックの精度は、resが指す位置に格納されます。

resが NULL の場合、クロックの精度は格納されません。

clock_gettimeの時刻引数 (tp) がresの倍数でない場合、この値はresの倍数になるように切り捨てられます。

成功すると、この関数は 0 を返します。

失敗すると、この関数は-1 を返し、エラーを示す値をerrnoに設定します。

clock_gettime, clock_gettime, time, ctimeも参照してください。

Return value

- | | |
|----|--|
| 0 | 成功を示します。 |
| -1 | 失敗を示します。errnoには、次の値が設定されます。 <ul style="list-style-type: none">• EINVAL – clock_id引数が、既知のクロックを指定していません。 |

clock_gettime (Alpha, I64)

指定されたクロックの現在の時刻 (秒およびナノ秒) を返します。

Format

```
#include <time.h>

int clock_gettime (clockid_t clock_id, struct timespec *tp);
```

Argument

clock_id
設定されているクロックの時刻を取得するために使用されるクロック・タイプ。
CLOCK_REALTIME クロックがサポートされています。このクロックは、システムの TIME-OF-DAY クロックです。

tp
timespec データ構造体へのポインタ。

Description

clock_gettime関数は、指定されたクロックclock_idの現在のtp値を返します。

成功すると、この関数は0を返します。

失敗すると、この関数は-1を返し、エラーを示す値をerrnoに設定します。

clock_getres, clock_gettime, time, ctimeも参照してください。

Return value

- | | |
|----|--|
| 0 | 成功を示します。 |
| -1 | 失敗を示します。errnoには、次の値が設定されます。 <ul style="list-style-type: none">• EINVAL – clock_id引数が、既知のクロックを指定していません。または、tp引数が0より小さい、または10億ナノ秒以上を指定しています。 |

clock_settime (Alpha, I64)

指定されたクロックへの設定を行います。

Format

```
#include <time.h>

int clock_settime (clockid_t clock_id, const struct timespec *tp);
```

Argument

clock_id

設定対象のクロックのクロック・タイプ。CLOCK_REALTIME クロックがサポートされています。このクロックは、システムの TIME-OF-DAY クロックです。

tp

timespec データ構造体へのポインタ。

Description

clock_settime関数は、指定されたクロックclock_idに、tpで指定された値を設定します。指定されたクロックの精度の倍数でない時間値は、精度の倍数になるように切り捨てられます。

クロックは、システム単位 (つまり、すべてのプロセスから参照できる) のものと、プロセス単位 (計測時間はそのプロセスでのみ意味がある) のもののどちらでも構いません。

<time.h>に定義されている CLOCK_REALTIME クロックは、システムのリアルタイム・クロックです。このクロックでは、clock_settimeで指定する値とclock_gettimeで返される値は、エポックからの経過時間 (秒およびナノ秒) を示します。エポックは、グリニッジ標準時 (GMT) 1970 年 1 月 1 日 00:00:00:00 と定義されています。

clock_settime関数を使用するには、OPER、LOG_IO、およびSYSPRV 特権が必要です。

成功すると、この関数は 0 を返します。

失敗すると、この関数は-1 を返し、エラーを示す値をerrnoに設定します。

clock_getres, clock_gettime, time, ctimeも参照してください。

Return value

- | | |
|----|--|
| 0 | 成功を示します。 |
| -1 | 失敗を示します。errnoには、次の値が設定されます。 <ul style="list-style-type: none">• EINVAL – clock_id引数が、既知のクロックを指定していません。または、tp引数が、指定されたclock_idの範囲外か、0より小さいまたは10億ナノ秒以上を指定しています。• EPERM – 要求元プロセスには、指定されたクロックに設定を行うための適切な特権がありません。 |

clog (*Alpha*, *I64*)

clog (*Alpha*, *I64*)

引数 (複素数) の自然対数 (つまり底を e とした対数) を返します。

Format

```
#include <complex.h>

double complex clog (double complex z);
float complex clogf (float complex z);
long double complex clogl (long double complex z);
```

引数

z
複素数値。

Description

clog関数は、実軸上の負の領域に分岐線法を適用して、複素数 z の自然対数 (つまり底を e とした対数) を返します。

Return value

x	複素数の自然対数値。この値は、実軸上では数学的に無限の範囲、また虚軸上では区間 $[-i\pi, +i\pi]$ の範囲からなる帯状の領域にあります。
-----	---

close

ファイル記述子に関連付けられているファイルをクローズします。

Format

```
#include <unistd.h>

int close (int file_desc);
```

引数

file_desc
ファイル記述子。

Description

close関数は、暗黙にfflushを呼び出すことにより、バッファに格納されているデータを書き込みます。書き込みが失敗した場合は(たとえば、ディスクが満杯であったり、ユーザのクォータが超過したため)、closeは実行を続行します。この関数はOpenVMS チャネルをクローズし、バッファの割り当てを解除し、ファイル記述子(またはFILE ポインタ)に関連付けられているメモリを解放します。バッファに格納されているデータは消失し、ファイル記述子(またはFILE ポインタ)はファイルを参照しなくなります。

バッファに格納されているデータを書き込むときに発生したエラーから回復する必要がある場合は、closeを呼び出す前に、fsync (またはfflush) を明示的に呼び出す必要があります。

Return value

0	ファイルが正しくクローズされたことを示します。
-1	ファイル記述子が未定義であるか、ファイルのクローズでエラーが発生したことを示します(たとえば、バッファに格納されているデータを書き込むことができないなど)。

close

例

```
#include <unistd.h>
int fd;
.
.
.
fd = open ("student.dat", 1);
.
.
.
close(fd);
```

closedir

ディレクトリをクローズします。

Format

```
#include <dirent.h>
int closedir (DIR *dir_pointer);
```

引数

`dir_pointer`
オープンされているディレクトリの`dir`構造体を指すポインタ。

Description

`closedir`関数は、ディレクトリ・ストリームをクローズし、`dir_pointer`引数に関連付けられている構造体を解放します。関数から戻るときに、`dir_pointer`の値は必ずしも`DIR`型のアクセス可能なオブジェクトを示すとは限りません。

`DIR`型は、`<dirent.h>`ヘッダ・ファイルに定義されており、特定のディレクトリ内のすべてのディレクトリ・エントリを順に並べたシーケンスであるディレクトリ・ストリームを表します。ディレクトリ・エントリはファイルを表します。`readdir`関数の操作とは非同期的に、ファイルをディレクトリから削除したり、ディレクトリに追加することができます。

注意

後でディレクトリを正しくオープンすることができるように、オープンされているディレクトリは必ず`closedir`関数を使用してクローズする必要があります。

例

次の例では、`opendir`関数、`readdir`関数、`closedir`関数を使用して、ディレクトリからエントリ名を検索する方法を示しています。

```
#include <dirent.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define FOUND 1
#define NOT_FOUND 0

static int dir_example(const char *name, unsigned int unix_style)
{
    DIR *dir_pointer;
    struct dirent *dp;

    if ( unix_style )
        dir_pointer = opendir(".");
    else
        dir_pointer = opendir(getenv("PATH"));

    if ( !dir_pointer ) {
        perror("opendir");
        return NOT_FOUND;
    }

    /* Note, that if opendir() was called with UNIX style file
    /* spec like ".", readdir() will return only a single
    /* version of each file in the directory. In this case the
    /* name returned in d name member of the dirent structure
    /* will contain only file name and file extension fields,
    /* both lowercased like "foo.bar".
    /*
    /* If opendir() was called with OpenVMS style file spec,
    /* readdir() will return every version of each file in the
    /* directory. In this case the name returned in d name
    /* member of the dirent structure will contain file name,
    /* file extension and file version fields. All in upper
    /* case, like "FOO.BAR;1".
    /*

    for ( dp = readdir(dir_pointer);
          dp && strcmp(dp->d_name, name);
          dp = readdir(dir_pointer) )
        ;

    closedir(dir_pointer);

    if ( dp != NULL )
        return FOUND;
    else
        return NOT_FOUND;
}

int main(void)
{
    char *filename = "foo.bar";
    FILE *fp;
```

```
remove(filename);
if ( !(fp = fopen(filename, "w")) ) {
    perror("fopen");
    return (EXIT_FAILURE);
}
if ( dir_example( "FOO.BAR;1", 0 ) == FOUND )
    puts("OpenVMS style: found");
else
    puts("OpenVMS style: not found");
if ( dir_example( "foo.bar", 1 ) == FOUND )
    puts("UNIX style: found");
else
    puts("UNIX style: not found");
fclose(fp);
remove(filename);
return( EXIT_SUCCESS );
}
```

Return value

0	正常終了を示します。
-1	エラーを示します。エラーの詳細はグローバルなerrnoに指定されます。

[w]clrattr

ウィンドウ内のビデオ表示属性attrを無効にします。clrattr関数はstdscrウィンドウに対して動作します。

Format

```
#include <curses.h>

int clrattr (int attr);

int wclrattr (WINDOW *win, int attr);
```

Argument

win

ウィンドウを指すポインタ。

attr

ビデオ表示属性 (点滅, 太字, 反転表示, 下線のいずれか)。これらの属性は定義済み定数 `_BLINK`, `_BOLD`, `_REVERSE`, `_UNDERLINE` によって表されます。複数の属性をクリアするには, 次に示すように, ビット単位の OR 演算子 (`|`) で区切ります。

```
clrattr(_BLINK | _UNDERLINE);
```

Description

これらの関数はHP C for OpenVMSシステム固有であり, 移植できません。

Return value

OK	正常終了を示します。
ERR	エラーを示します。

[w]clrtoobot

ウィンドウの内容を現在のカーソルの位置からウィンドウの最下部まで消去します。clrtoobot関数はstdscrウィンドウに対して動作します。

Format

```
#include <curses.h>
int clrtoobot();
int wclrtoobot (WINDOW *win);
```

引数

win
ウィンドウを指すポインタ。

Return value

OK	正常終了を示します。
ERR	エラーを示します。

[w]clrtoeol

ウィンドウの内容を指定されたウィンドウの現在のカーソルの位置から行末まで消去します。clrtoeol関数はstdscrウィンドウに対して動作します。

Format

```
#include <curses.h>

int clrtoeol();

int wclrtoeol (WINDOW *win);
```

引数

win
ウィンドウを指すポインタ。

Return value

OK	正常終了を示します。
ERR	エラーを示します。

confstr

文字列値によって定義されるシステム変数の現在の値を返します。

Format

```
#include <unistd.h>

size_t confstr (int name, char *buf, size_t len);
```

Argument

name

システム変数の設定。name引数に対して指定できる値は、<unistd.h>ヘッダ・ファイルに定義されている_CS_X 名です。

buf

confstr関数がnameの値をコピーする先のバッファを指すポインタ。

len

nameの値が格納されるバッファのサイズ。

Description

confstr関数を使用すると、アプリケーションは文字列値によって定義される特定のシステム・パラメータ、リミット、オプションの現在の設定を判断することができます。この関数は主に、アプリケーションでPATH環境変数のシステム・デフォルト値を判断するために使用されます。

次の条件が満たされる場合は、confstr関数は値をbufによって示されるlenバイトのバッファにコピーします。

- len引数が0でない。
- name引数にシステムで定義されている値が指定されている。
- buf引数がNULLポインタでない。

返された文字列が末尾のヌルも含めてlenバイトより長い場合は、confstr関数は文字列をlen - 1バイトに切り捨て、結果の末尾にヌルを追加します。アプリケーションは、confstr関数から返された値を、len引数の値と比較することにより、文字列が切り捨てられたことを検出できます。

<limits.h>ヘッダ・ファイルには、システムで定義されているリミットが格納されています。<unistd.h>ヘッダ・ファイルには、システム定義環境変数が格納されています。

例

nameの文字列値を格納するのに必要なバッファのサイズを判断するには、次のように入力します。

```
confstr(_CS_PATH, NULL, (size_t) 0)
```

confstr関数は必要なバッファのサイズを返します。

Return value

0	<p>エラーを示します。指定されたnameの値に応じて、次のいずれかになります。</p> <ul style="list-style-type: none"> 不正な場合、errnoはEINVALに設定されます。 システム定義値がない場合、errnoは設定されません。
n	<p>値を格納するのに必要なバッファのサイズ。</p> <ul style="list-style-type: none"> name引数の値がシステム定義値の場合は、confstrは値全体を格納するのに必要なバッファのサイズを返します。この戻り値がlenの値より大きい場合は、bufの値として返された文字列は切り捨てられます。 len引数の値が0に設定されている場合や、bufの値がNULLの場合は、confstrはシステム定義値全体を格納するのに必要なバッファのサイズを返します。文字列の値はコピーされません。

conj (*Alpha, I64*)

引数 (複素数) の共役値を返します。

Format

```
#include <complex.h>

double complex conj (double complex z);
float complex conjf (float complex z);
long double complex conjl (long double complex z);
```

引数

z
複素数值。

Description

conj関数は、複素数 z の共役値、つまり虚数部の符号を反転させた複素数を返します。

Return value

x 共役複素数値。

copysign (Alpha, I64)

yと同じ符号でxを返します。

Format

```
#include <math.h>

double copysign (double x, double y);
float copysignf (float x, float y); (Alpha, I64)
long double copysignl (long double x, long double y); (Alpha, I64)
```

Argument

x
実数値。

y
実数値。

Description

copysign関数はyと同じ符号を付けてxを返します。IEEE 754 では、copysign(x,NaN) , copysignf(x,NaN) , copysignl(x,NaN) は +xまたは-xを返すように要求されています。

戻り値

x	yと同じ符号を付けたxの値。
---	----------------

COS

ラジアン単位の引数の余弦を返します。

Format

```
#include <math.h>
double cos (double x);
float cosf (float x); (Alpha, I64)
long double cosl (long double x); (Alpha, I64)
double cosd (double x); (Alpha, I64)
float cosdf (float x); (Alpha, I64)
long double cosdl (long double x); (Alpha, I64)
```

引数

x
実数値として表したラジアン値。

Description

cos関数は、引数の余弦 (ラジアン単位) を返します。

cosd関数は、引数の余弦 (度単位) を返します。

$|x| = \infty$ は不正な引数です。

Return value

x	引数の余弦。
HUGE_VAL	引数が大きすぎることを示します。errnoは ERANGE に設定されます。

cosh

ラジアン単位の引数の双曲線余弦を返します。

Format

```
#include <math.h>

double cosh (double x);
float coshf (float x); (Alpha, I64)
long double coshl (long double x); (Alpha, I64)
```

引数

x
実数値で表したラジアン値。

Description

cosh関数はxの双曲線余弦を返します。これらの関数は $(e^{**x} + e^{*(-x)})/2$ として定義されます。

Return value

x	引数の双曲線余弦。
HUGE_VAL	引数が大きすぎることを示します。errnoは ERANGE に設定されます。

cot

ラジアン単位の引数の余接を返します。

Format

```
#include <math.h>

double cot (double x);
float cotf (float x); (Alpha, I64)
long double cotl (long double x); (Alpha, I64)
double cotd (double x); (Alpha, I64)
float cotdf (float x); (Alpha, I64)
long double cotdl (long double x); (Alpha, I64)
```

引数

x
実数値で表したラジアン値。

Description

cot関数は、引数の余接 (ラジアン単位) を返します。

cotd関数は、引数の余接 (度単位) を返します。

$x = 0$ は不正な引数です。

Return value

x	引数の余接。
HUGE_VAL	引数が0であることを示します。errnoはERANGEに設定されます。

cpow (Alpha, I64)

複素数のべき乗関数値 $x**y$ を返します。

Format

```
#include <complex.h>

double complex cpow (double complex x, double complex y);
float complex cpowf (float complex x, float complex y);
long double complex cpowl (long double complex x, long double complex y);
```

引数

x
複素数値。

y
複素数値。

Description

cpow関数は、第 1 パラメータに対して実軸上の負の領域に分岐線法を適用して、複素数のべき乗関数値 $x**y$ を返します。

Return value

x
複素数のべき乗関数値。

REF-97

`creal` (*Alpha, I64*)

`creal` (*Alpha, I64*)

引数 (複素数) の実数部を返します。

Format

```
#include <complex.h>
double creal (double complex z);
float crealf (float complex z);
long double creall (long double complex z);
```

引数

`z`
複素数値。

Description

`creal`関数は、`z`の実数部を返します。

Return value

`x` 実数部の値。

creat

新しいファイルを作成します。

Format

```
#include <fcntl.h>

int creat (const char *file_spec, mode_t mode); (ISO POSIX-1)

int creat (const char *file_spec, mode_t mode, ... ); (HP C Extension)
```

Argument

file_spec

有効なファイル指定を格納したヌル区切り文字列。

mode

ファイル保護モードを指定する符号なし値。コンパイラはモードおよび現在の保護モードの補数に対してビット単位の AND 操作を実行します。

モードを作成するには、ビット単位の OR 演算子(|)を使用してモードの組み合わせを作成します。モードは次のとおりです。

0400	OWNER:READ
0200	OWNER:WRITE
0100	OWNER:EXECUTE
0040	GROUP:READ
0020	GROUP:WRITE
0010	GROUP:EXECUTE
0004	WORLD:READ
0002	WORLD:WRITE
0001	WORLD:EXECUTE

システムにはオーナーと同じの特権が与えられます。WRITE 特権がある場合は、DELETE 特権もあるものと解釈されます。

 注意

UNIX システム呼び出し関数 `umask`, `mkdir`, `creat`, `open` を使用して、OpenVMS RMS のデフォルト保護でファイルを作成するには、`umask` を絶対に呼び出すことがないプログラムでファイル保護モード引数を 0777 に設定して、`mkdir`, `creat`, `open` を呼び出します。これらのデフォルト保護には、ACL やファイルの前のバージョンなどをもとに正しく設定された保護が含まれます。

`vfork/exec` を呼び出すプログラムでは、新しいプロセス・イメージは、`umask` が呼び出しプロセス・イメージから呼び出されたかどうかを継承します。`umask` の設定と、`umask` 関数がこれまで呼び出されたかどうかは、どちらも継承される属性です。

...

次の形式の文字列で構成される省略可能な引数リスト。

"keyword = value", ..., "keyword = value"

"acc" または "err" の場合は、次の形式になります。

"keyword"

ここで、keyword はファイル・アクセス・ブロック (FAB) またはレコード・アクセス・ブロック (RAB) の RMS フィールドです。value はそのフィールドに代入できる有効な値です。一部のフィールドには、複数の値を指定できます。このような場合、値はコンマで区切ります。

RMS コールバック・キーワード "acc" と "err" だけは値を受け付けないキーワードです。これらのキーワードの後には、使用するコールバック・ルーチンを指すポインタを指定し、さらにコールバック・ルーチンの最初の引数として使用されるユーザ指定値を指すポインタを指定します。たとえば、`acc_callback` というアクセス・コールバック・ルーチンを設定し、そのルーチンの最初の引数が `open` の呼び出しの整数変数 `first_arg` を指すポインタの場合、次の文を使用できます。

```
open("file.dat", 0_RDONLY, 0, "acc", acc_callback, &first_arg)
```

コールバック・ルーチンに対する 2 番目の引数と 3 番目の引数はそれぞれ、FAB と RAB を指すポインタでなければならず、ルーチンの戻り値の型は `int` でなければなりません。コールバックが 0 より小さい値を返す場合は、`open`, `creat`, `fopen` は異常終了します。エラー・コールバックはエラー条件を修正し、0 以上の状態を返すことにより、`creat` 呼び出しを続行できます。上記の `open` 文を例にとってみると、`acc_callback` の関数プロトタイプは次の文のようになります。

```
#include <rms.h>
```

```
int acc_callback(int *first_arg, struct FAB *fab, struct RAB *rab);
```

FAB と RAB は<rms.h>ヘッダ・ファイルに定義されており，ルーチンに渡される実際のポインタは，ファイル file.dat をオープンするために使用される RAB と FAB を指すポインタです。

アクセス・コールバック・ルーチンが確立された後，このルーチンは，RMS 関数 sys\$create または sys\$open を呼び出す直前に，オープン・タイプのルーチンで呼び出されます。エラー・コールバック・ルーチンが確立され，sys\$create または sys\$open 関数からエラー状態が返された場合は，状態が確認され，エラー値が検出された直後に，コールバック・ルーチンが呼び出されます。

注意

コールバック関数で RAB または FAB を操作すると，その後の HP C RTL I/O 関数の呼び出しで重大な問題が発生する可能性があります。

表 REF-3 は RMS キーワードと値を示しています。

表 REF-3 RMS の有効なキーワードと値

キーワード	値	説明
"acc"	コールバック	アクセス・コールバック・ルーチン。
"alq = n"	10 進数	割り当て数量。
"bls = n"	10 進数	ブロック・サイズ。
"ctx = bin"	文字列	'\n' を変換せずに端末に出力する。バイナリ・データをファイルに書き込むときはこのキーワードを使用する。
"ctx = cvt"	文字列	"ctx=nocvt" の前の設定を否定する。これはデフォルトである。
"ctx = nocvt"	文字列	Fortran のキャリッジ制御バイトを変換しない。
"ctx = rec"	文字列	レコード・モード・アクセスを強制する。
"ctx = stm"	文字列	ストリーム・モード・アクセスを強制する。
"ctx = xplt"	文字列	fflush, close, fclose の呼び出しによって明示的に指定された場合にだけ，レコードを書き込む。
"deq = n"	10 進数	デフォルトの拡張数量。
"dna = filespec"	文字列	デフォルトのファイル名文字列。
"err"	コールバック	エラー・コールバック・ルーチン。
"fop = val, val, . . ."		ファイル処理オプション:

(次ページに続く)

表 REF-3 (続き) RMS の有効なキーワードと値

キーワード	値	説明
	ctg	連続。
	cbt	連続最適トライ。
	dfw	遅延書き込み。共用アクセスのためにオープンされたファイルにだけ適用される。
	dlt	クローズ時にファイルを削除する。
	tef	EOF で切り捨てる。
	cif	存在しない場合は作成する。
	sup	現在は使用されていない。
	scf	クローズ時にコマンド・ファイルとしてサブミットする。
	spl	クローズ時にシステム・プリンタにスプールする。
	tmd	一時的に削除する。
	tmp	一時的 (ファイル・ディレクトリでない)。
	nef	EOF でない。
	rck	読み込みチェック比較操作。
	wck	書き込みチェック比較操作。
	mxv	バージョン番号を最大にする。
	rwo	オープン時にファイルを巻き戻す。
	pos	現在の位置。
	rwc	クローズ時にファイルを巻き戻す。
	sqa	ファイルは順次処理だけが可能である。
"fsz = n"	10 進数	固定ヘッダ・サイズ。
"gbc = n"	10 進数	ファイルに対して要求されたグローバル・バッファの数。
"mbc = n"	10 進数	マルチブロック・カウント。
"mbf = n"	10 進数	マルチバッファ・カウント。
"mrs = n"	10 進数	最大レコード・サイズ。
"pmt=usr-prmpt"	文字列	端末入力を促すプロンプト。このオプションと"rop=pmt"が指定されている場合は、端末デバイスからの RMS 入力の前に"usr-prmpt"が付加される。
"rat = val, val . . . "		レコード属性:
	cr	キャリッジ・リターン制御。
	blk	レコードがブロック境界にまたがることを禁止する。
	ftn	Fortran プリント制御。
	none	明示的にキャリッジ制御なしを強制する。
	prn	ファイル・フォーマットをプリントする。
"rfm = val"		レコード・フォーマット:
	fix	固定長レコード・フォーマット。
	stm	RMS ストリーム・レコード・フォーマット。
	stm1f	改行区切り文字付きストリーム・フォーマット。
	stmcr	キャリッジ・リターン区切り文字付きストリーム・フォーマット。
	var	可変長レコード・フォーマット。
	vfc	固定長制御部付き可変長レコード。
	udf	未定義。
"rop = val, val . . . "		レコード処理操作:
	asy	非同期 I/O。

(次ページに続く)

表 REF-3 (続き) RMS の有効なキーワードと値

キーワード	値	説明
	cco	Ctrl/O をキャンセルする (端末 I/O で使用)。
	cvt	端末からの読み込み時に文字を大文字に変換する。
	eof	接続操作の場合のみ、レコード・ストリームを EOF に設定する。
	nlk	レコードをロックしない。
	pmt	端末からの入力時に“pmt=usr-prmpt”によって指定されるプロンプトの使用を有効にする。
	pta	端末からの読み込み時に先読みバッファの情報を消去する。
	rea	このプロセスに対して読み込み操作のためにレコードをロックするが、他のプロセスがレコードを読み込むことは許可する。
	rlk	書き込みのためにレコードをロックする。
	rne	キーボードからの入力時に、入力データの画面表示を行わない。
	rnf	端末入力で Ctrl/U, Ctrl/R, DELETE が制御コマンドとして解釈されないが、アプリケーション・プログラムに渡されることを示す。
	rrl	ロックとは無関係に読み込む。
	syncsts	要求されたサービスがタスクをただちに完了したときに、RMS\$_SYNCH という正常終了状態を返す。
	tmo	I/O 時間切れ。
	tpt	順次レコード・アクセス・モードを使用する put /write サービスがファイルの任意のポイントで発生することを許可し、そのポイントでファイルを切り捨てる。
	ulk	RMS が自動的にレコードのロックを解除することを禁止する。
	wat	レコードが現在別のストリームによってロックされている場合、そのレコードが使用可能になるまで待つ。
	rah	先読みする。
	wbh	後書きする。
“rtv=n”	10 進数	RMS がメモリ内で管理しなければならない検索ポイントの数 (0 ~ 127,255)。
“shr = val, val, . . . ”		ファイル共用オプション: del ユーザによる削除を許可する。 get ユーザによる読み込みを許可する。 mse マルチストリーム接続を許可する。 nil ファイルの共用を禁止する。 put ユーザによる書き込みを許可する。 upd ユーザによる更新を許可する。 upi 1 人以上のユーザによる書き込みを許可する。 nql クエリ・ロックを行わない (ファイル・レベル)。
“tmo = n”	10 進数	I/O 時間切れの値。

これらのオプションの他に、キー値を受け付けるオプション (“fop”や“rat”など) は、値の前に“no”を付けることにより否定することができます。たとえ

ば, “fop=notmp”と指定すると, “fop”フィールドの“tmp”ビットがクリアされます。

注意

- これらのオプションは柔軟性と機能性を向上しますが, 多くのオプションは, 正しく使用しないと, 重大な問題を引き起こす危険性があります。
 - デフォルトのHP C for OpenVMSストリーム・ファイル I/O を共用することはできません。ファイルを共用する場合は, “ctx=rec”を指定して, 強制的にレコード・アクセス・モードに設定する必要があります。また, 目的のアクセス・タイプに応じて, 適切な“shr”オプションも指定しなければなりません。
 - 追加のためにオープンされているファイルを共用する場合は, 適切な共用オプションとレコード・ロック・オプションを指定して, 同じファイルにアクセスする他のプロセスがレコードを読み込むことができるようにしなければなりません。この処理が必要なのは, EOF に到達するまでレコードをループで読み込むことにより, ファイルが EOF の位置に設定されるからです。
-

これらのオプションの詳細については, 『『OpenVMS Record Management Services Reference Manual』』を参照してください。

Description

HP C RTL は読み込みと書き込みのために新しいファイルをオープンし, 対応するファイル記述子を返します。

ファイルが存在する場合:

- 既存のバージョンより 1 だけ大きいバージョン番号が新規作成されたファイルに割り当てられます。
- デフォルト設定により, 新しいファイルは既存のファイル・バージョンから特定の属性を継承します。ただし, creat呼び出しに指定された属性は継承しません。次の属性が継承されます。
 - レコード・フォーマット (FAB\$B_RFM)
 - 最大レコード・サイズ (FAB\$W_MRS)
 - キャリッジ制御 (FAB\$B_RAT)
 - ファイル保護

- 新しいバージョンのファイルを作成するときに、そのファイルと同じ名前のシンボリック・リンクがすでに存在していると、そのシンボリック・リンクが参照しているファイルが作成されます。

ファイルが存在しない場合:

- mode引数および現在の保護マスクの補数に対してビット単位の AND を実行することにより作成されるファイル保護が与えられます。
- デフォルト設定により、ライン・フィード・レコード・セパレータと暗黙のキャリッジ・リターン属性が割り当てられたストリーム・フォーマットが設定されます。

このセクションのopen, close, read, write, lseekも参照してください。

Return value

n	ファイル記述子。
-1	保護違反, 未定義ディレクトリ, ファイル属性の競合などのエラーを示します。

[no]crmode

UNIX システム環境では、`crmode`関数と`nocrmode`関数は端末を `cbreak` モードに設定するか、または設定を解除します。`cbreak` モードでは、Return を処理せずに 1 文字の入力文字を処理することができます。このシングル文字入力モードは Curses 入力ルーチン`getch`でのみサポートされます。

Format

```
#include <curses.h>

crmode()

nocrmode()
```

例

```
/* Program to demonstrate the use of crmod() and curses */
#include <curses.h>
main()
{
    WINDOW *win1;
    char vert = '.',
        hor = '.',
        str[80];

    /* Initialize standard screen, turn echo off. */
    initscr();
    noecho();

    /* Define a user window. */
    win1 = newwin(22, 78, 1, 1);

    /* Turn on reverse video and draw a box on border. */
    setattr(_REVERSE);
    box(stdscr, vert, hor);
    mvwaddstr(win1, 2, 2, "Test cbreak input");
    refresh();
    wrefresh(win1);

    /* Set cbreak, do some input, and output it. */
    crmode();
    getch();
    nocrmode(); /* Turn off cbreak. */
    mvwaddstr(win1, 5, 5, str);
    mvwaddstr(win1, 7, 7, "Type something to clear the screen");
    wrefresh(win1);
}
```

```
/* Get another character, then delete the window. */  
getch();  
wclear(win1);  
touchwin(stdscr);  
endwin();  
}
```

この例では、`getch`の最初の呼び出しは、1文字が入力されるとただちに帰ります。これは、`getch`が呼び出される前に、`crmode`が呼び出されているからです。`getch`の2回目の呼び出しでは、入力された文字を処理する前に、Return キーが押されるのを待ちます。これは、`getch`の2回目の呼び出しの前に、`nocrmode`が呼び出されているからです。

crypt

パスワードを暗号化します。

Format

```
#include <unistd.h>
#include <stdlib.h>
char *crypt (const char *key, const char *salt;)
```

関数バリエント

crypt関数には、2つのバリエントがあります。1つは、ポインタのサイズが32ビットの_crypt32で、もう1つは、ポインタのサイズが64ビットの_crypt64です。ポインタのサイズに合わせた専用関数の詳細な使用方法については、第1.10節を参照してください。

引数

key
ユーザの入力したパスワード。

salt
2文字からなる文字列。

Description

crypt関数は、エンコード (つまり暗号化) されたパスワードを生成します。暗号化の方法はNBS Data Encryption Standardをベースにしていますが、DESをハードウェア化してもキーを見つけるのが困難になるように強化されています。

特別なことがない限り、cryptの第1引数はユーザが入力したパスワードです。第2引数 (salt) は、[a-zA-Z0-9./]から選択された2文字の文字列です。最初に、saltの文字列を使用してDESのアルゴリズムが攪乱されます (攪乱には4096通りの方法があって、その中の1つを選択するためにsaltが使用されます)。次に、パスワードをキーにして定数文字列が繰り返し暗号化されます。戻り値はポインタになっていて、暗号化されたパスワードを指しています。暗号化されたパスワードはsaltと同じようにアルファベットで表されており、最初の2文字がsaltと同じ文字列になっています。

cryptから返される値は静的なデータ領域を指していますが、その内容は、この関数を呼び出すたびに上書きされます。

encryptとsetkeyも参照してください。

戻り値

pointer

暗号化されたパスワードを指すポインタ。

csin (Alpha, I64)

引数 (複素数) の正弦を返します。

Format

```
#include <complex.h>
double complex csin (double complex z);
float complex csinf (float complex z);
long double complex csinl (long double complex z);
```

引数

z
複素数値。

Description

csin関数は、複素数zの正弦値を計算します。

Return value

x
複素数の正弦値。

csinh (Alpha, I64)

引数 (複素数) の双曲線正弦を返します。

Format

```
#include <complex.h>
double complex csinh (double complex z);
float complex csinhf (float complex z);
long double complex csinhl (long double complex z);
```

引数

z
複素数値。

Description

csinh関数は、複素数zの双曲線正弦を計算します。

Return value

x 複素数の双曲線正弦値。

csqrt (Alpha, I64)

引数 (複素数) の平方根を返します。

Format

```
#include <complex.h>
double complex csqrt (double complex z);
float complex csqrtf (float complex z);
long double complex csqrtl (long double complex z);
```

引数

z
複素数値。

Description

csqrt関数は、実軸上の負の領域に分岐線法を適用して、複素数zの平方根を計算します。

Return value

x
複素数の平方根値。この値は、右側の半平面 (虚軸を含む範囲) にあります。

REF-113

ctanh (*Alpha, I64*)

引数 (複素数) の双曲線正接を返します。

Format

```
#include <complex.h>

double complex ctanh  (double complex z);
float complex ctanhf  (float complex z);
long double complex ctanhl (long double complex z);
```

引数

Z
複素数值。

Description

ctanh関数は、複素数 z の双曲線正接値を返します。

Return value

X 複素数の双曲線正接値。

ctermid

SYSS\$COMMAND の等価文字列を与える文字列を返します。これは制御端末の名前です。

Format

```
#include <stdio.h>

char *ctermid (char *str);
```

関数バリエーション

ctermid関数には、_ctermid32および_ctermid64という名前のバリエーションがあり、それぞれ 32 ビット・ポインタ・サイズおよび 64 ビット・ポインタ・サイズで使用されます。ポインタ・サイズ固有の関数の使い方については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

引数

str
文字配列を指すポインタ。この引数が NULL の場合は、ファイル名は内部的に格納され、次にctermidを呼び出したときに上書きされます。NULL 以外の場合は、ファイル名は引数によって示される位置から始まる場所に格納されます。引数はL_ctermid (<stdio.h>ヘッダ・ファイルで定義) の長さの記憶域を示さなければなりません。

戻り値

ポインタ

文字列を指すポインタ。

ctime, ctime_r

1970 年 1 月 1 日 00:00:00 からの経過時間 (秒数) を, asctime関数で生成される形式の ASCII 文字列に変換します。

Format

```
#include <time.h>

char *ctime (const time_t *bintim);

char *ctime_r (const time_t *bintim, char *buffer); (ISO POSIX-1)
```

関数バリエーション

_DECC_V4_SOURCE および _VMS_V6_SOURCE 機能テスト・マクロを定義してコンパイルすると, OpenVMS Version 7.0 より前の動作に相当するこの関数のローカル・タイム・ベースのエントリ・ポイントが有効になります。

Argument

bintim

変換する時間値 (秒数) を指定する変数を指すポインタ。

buffer

長さが 26 バイト以上の文字列の配列を指すポインタ。この配列は, 生成された日付/時刻文字列を格納するために使用されます。

Description

ctime関数とctime_r関数は, bintimによって示される時刻を 26 文字の文字列に変換し, この文字列を指すポインタを返します。

ctime_r関数とctime関数の相違点は, ctime_r関数が結果をユーザ指定バッファに格納するのに対し, ctime関数はHP C RTL によって割り当てられたスレッド固有の静的メモリに結果を格納する点です。ctimeまたはasctimeを再び呼び出すと, スレッド固有の静的メモリに格納されている結果は上書きされる可能性があります。結果を保存する必要がある場合は, コピーを作成しなければなりません。

正常終了すると, ctimeは文字列を指すポインタを返します。ctime_rは 2 番目の引数を返します。異常終了すると, これらの関数は NULL ポインタを返します。

time_t型は、次に示すように<time.h>ヘッダ・ファイルに定義されています。

```
typedef long int time_t
```

ctime関数は、tzsetを呼び出したかのように動作します。

注意

一般に、UTC ベースの時刻関数は、プロセス単位のリソースであるメモリ内のタイム・ゾーン情報に影響を与える可能性があります。しかし、アプリケーションの実行中、システム・タイム・ゾーンが変化せず(これは一般的なケースです)、タイム・ゾーン・ファイルのキャッシュが有効に設定されている場合(これはデフォルトです)、時刻関数asctime_r, ctime_r, gmtime_r, localtime_rの_rバリエーションはスレッド・セーフで、かつ AST リエントラントです。

しかし、アプリケーションの実行中にシステム・タイム・ゾーンが変化する可能性がある場合や、タイム・ゾーン・ファイルのキャッシュが有効に設定されていない場合は、UTC ベースの時刻関数のバリエーションはいずれも第 3 の関数クラスに属し、これはスレッド・セーフでも AST リエントラントでもありません。

Return value

x	正常終了した場合は、26 文字の ASCII 文字列を指すポインタ。
NULL	異常終了を示します。

cuserid

現在のプロセスを開始したユーザの名前を格納した文字列を指すポインタを返します。

Format

```
#include <unistd.h> (X/Open, POSIX-1)
#include <stdio.h> (X/Open)
char *cuserid (char *str);
```

関数バリエーション

cuserid関数には、_cuserid32および_cuserid64という名前のバリエーションがあり、それぞれ 32 ビット・ポインタ・サイズおよび 64 ビット・ポインタ・サイズで使用されます。ポインタ・サイズ固有の関数の使い方については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

引数

str

この引数が NULL の場合は、ユーザ名は内部的に格納されます。引数が NULL でない場合は、長さが L_cuserid (<stdio.h> ヘッダ・ファイルで定義) の記憶域を指すポインタであり、名前はその記憶域に格納されます。ユーザ名がヌル文字列の場合は、NULL を返します。

Return value

ポインタ

文字列を指すポインタ。

NULL

ユーザ名がヌル文字列の場合。

DECC\$CRTL_INIT

この関数を使用すると、他の言語からHP C RTL を呼び出したり、メイン関数が C で作成されていないときにHP C RTL を使用することができます。この関数は実行時環境を初期化し、終了ハンドラと条件ハンドラの両方を設定します。VAXC\$CRTL_INITはDECC\$CRTL_INITの同意語です。どちらの名前も同じルーチンを起動します。

Format

```
#include <signal.h>

void DECC$CRTL_INIT(void);
```

Description

次の例は、DECC\$CRTL_INIT関数を使用してHP C RTL を呼び出す Pascal プログラムを示しています。

```
$ PASCAL EXAMPLE1
$ LINK EXAMPLE1
$ TY EXAMPLE1.PAS
PROGRAM TESTC(input, output);
PROCEDURE DECC$CRTL_INIT; extern;
BEGIN
    DECC$CRTL_INIT;
END
```

共用可能イメージでこの関数を呼び出す必要があるのは、そのイメージにシグナル処理、環境変数、I/O、終了処理、デフォルト・ファイル保護マスクのためのHP C関数が含まれている場合や、コンテキストを継承する子プロセスである場合だけです。

初期化処理の多くは1回だけ実行されますが、DECC\$CRTL_INITは複数回呼び出しても安全です。OpenVMS VAX システムでは、DECC\$CRTL_INITは、DECC\$CRTL_INITが呼び出されるたびにDECC\$CRTL_INITを呼び出すルーチンのフレーム内で、HP C RTL の内部 OpenVMS 例外ハンドラを設定します。

現在の呼び出しスタック内の少なくとも1つのフレームで、UNIX シグナルにマッピングされる OpenVMS 例外に対してハンドラが設定されている必要があります。

decc\$feature_get

decc\$feature_get_valueを、インデックスではなく文字列の機能名で呼び出します。

Format

```
#include <unixlib.h>

int decc$feature_get (const char *name, int mode);
```

引数

name

機能名 (文字列) を指すポインタ。このポインタを介して渡される名前は、サポート機能のリストにあるものでなければなりません。

mode

返される機能値を指定するための整数値。指定できるmodeの値は、次のとおりです。

__FEATURE_MODE_DEFVAL	デフォルト値
__FEATURE_MODE_CURVAL	現在の値
__FEATURE_MODE_MINVAL	最小値
__FEATURE_MODE_MAXVAL	最大値
__FEATURE_MODE_INIT_STATE	初期化状態

Description

decc\$feature_get関数を使用すれば、C RTL の内部テーブルへのインデックスではなく、文字列で表した機能名でdecc\$feature_get_value関数を呼び出すことができます。

エラーが発生すると、そのエラーを示す値がerrnoに設定されて、-1 が返されます。

decc\$feature_get_value, decc\$feature_get_index, decc\$feature_get_name, decc\$feature_set, decc\$feature_show_all および decc\$feature_show_all も参照してください。

Return value

n	指定したname引数とmode引数に対応する整数。
-1	エラーを示します。errnoが設定されます。

decc\$feature_get_index

機能値にアクセスするためのインデックスを返します。

Format

```
#include <unixlib.h>

int decc$feature_get_index (char *name);
```

引数

name
サポートされる機能の一覧で、名前として渡される文字列を指すポインタ。

Description

decc\$feature_get_index関数は、サポートされる機能の一覧から、nameとして渡された文字列を検索します。名前が見つかると、decc\$feature_get_indexは(負でない)インデックスを返します。このインデックスを使用して、機能の値を設定したり、取得することができます。nameの比較では、大文字と小文字は区別されません。

エラーが発生すると、-1 が返され、errnoはエラーを示すように設定されます。

decc\$feature_get, decc\$feature_get_value, decc\$feature_get_name, decc\$feature_set, decc\$feature. およびdecc\$feature_show_allも参照してください。

Return value

n	負でないインデックス。このインデックスを使用して、機能に対して指定された値を設定するか、または値を取得することができます。
-1	エラーを示します。errnoが設定されます。

decc\$feature_get_name

機能名を返します。

Format

```
#include <unixlib.h>
char *decc$feature_get_name (int index);
```

引数

index
0 から、割り当てられている最大の機能までの整数値。

Description

decc\$feature_get_name関数は、indexで指定されるエントリの機能名を格納したヌル区切り文字列を指すポインタを返します。indexの値は、0 から、割り当てられている最大の機能までの範囲です。indexの値に対応する機能がない場合は、NULL ポインタが返されます。

エラーが発生すると、NULL が返され、errnoはエラーを示すように設定されます。

decc\$feature_get、decc\$feature_get_index、decc\$feature_get_value、decc\$feature_set、decc\$fおよびdecc\$feature_show_allも参照してください。

Return value

x	indexで指定されるエントリの機能の名前を格納したヌル区切り文字列を指すポインタ。
NULL	エラーを示します。errnoが設定されます。

decc\$feature_get_value

引数indexとmodeで指定した機能の機能値を返します。

Format

```
#include <unixlib.h>

int decc$feature_get_value (int index, int mode);
```

Argument

index

0 から、割り当てられている最大の機能までの整数値。

mode

返す機能値を示す整数。 mode に対して指定できる値は次のとおりです。

__FEATURE_MODE_DEFVAL	デフォルト値
__FEATURE_MODE_CURVAL	現在の値
__FEATURE_MODE_MINVAL	最小値
__FEATURE_MODE_MAXVAL	最大値
__FEATURE_MODE_INIT_STATE	初期化状態

Description

decc\$feature_get_value関数は、indexによって指定される機能の値を取得します。modeは、どの値を返すかを指定します。

デフォルト値とは、論理名によって設定されていない場合や、decc\$feature_set_valueの呼び出しで変更されていない場合に使用される値です。

mode = 4 の場合は、初期化状態が返されます。初期化状態の値は次のとおりです。

- 0 初期化されていない
- 1 論理名によって設定
- 2 decc\$feature_set_valueによって設定
- 1— デフォルト値に初期化

エラーが発生すると、-1 が返され、errnoはエラーを示すように設定されます。

decc\$feature_get , decc\$feature_get_index , decc\$feature_get_name , decc\$feature_set , decc\$feature_show_all
およびdecc\$feature_show_allも参照してください。

Return value

- n 指定されたindex引数とmode引数に対応する整数。
- 1 エラーを示します。errnoが設定されます。

decc\$feature_set

decc\$feature_set_valueを、インデックスではなく文字列の機能名で呼び出します。

Format

```
#include <unixlib.h>

int decc$feature_set (const char *name, int mode, int value);
```

引数

name

機能名 (文字列) を指すポインタ。このポインタを介して渡される名前は、サポート機能のリストにあるものでなければなりません。

mode

返される機能値を指定するための整数値。指定できるmodeの値は、次のとおりです。

__FEATURE_MODE_DEFVAL	デフォルト値
__FEATURE_MODE_CURVAL	現在の値
__FEATURE_MODE_MINVAL	最小値
__FEATURE_MODE_MAXVAL	最大値
__FEATURE_MODE_INIT_STATE	初期化状態

value

設定する機能値。

Description

decc\$feature_set関数を使用すれば、C RTL の内部テーブルへのインデックスではなく、文字列で表した機能名でdecc\$feature_set_value関数を呼び出すことができます。

成功すると、この関数は、設定されていた以前の値を返します。

エラーが発生すると、そのエラーを示す値がerrnoに設定されて、-1 が返されます。

decc\$feature_set_value, decc\$feature_get, decc\$feature_get_index, decc\$feature_get_name, decc\$f
およびdecc\$feature_show_allも参照してください。

Return value

n	設定されていた以前の機能値。
-1	エラーを示します。errnoが設定されます。

decc\$feature_set_value

indexで指定される機能のデフォルト値または現在の値を設定します。

Format

```
#include <unixlib.h>

int decc$feature_set_value (int index, int mode, int value);
```

Argument

index

0 から、割り当てられている最大の機能までの整数値。

mode

デフォルト値を設定するのか、現在の機能値を設定するのかを示す整数値。mode に対して指定できる値は次のとおりです。

0 デフォルト値

1 現在の値

value

設定する機能値。

Description

decc\$feature_set_value関数は、indexによって指定される機能のデフォルト値または現在の値 (mode引数で指定) を設定します。

この関数が正常終了すると、前の値が返されます。

エラーが発生すると、-1 が返され、errnoはエラーを示すように設定されます。

decc\$feature_get, decc\$feature_get_index, decc\$feature_get_name, decc\$feature_get_value, decc\$feature_set_value およびdecc\$feature_show_allも参照してください。

Return value

n	前の機能値。
-1	エラーを示します。errnoが設定されます。

decc\$feature_show

指定した機能名に対応する機能値を、すべて表示します。

Format

```
#include <unixlib.h>

int decc$feature_show (const char *name);
```

引数

name

機能名 (文字列) を指すポインタ。このポインタを介して渡される名前は、サポート機能のリストにあるものでなければなりません。

Description

decc\$feature_show関数は、nameで指定した機能名の値を stdoutにすべて表示します。次に、その例を示します。

```
----- C RTL Feature Name -----   Cur  Def  Min  Max  Ini
DECC$V62_RECORD_GENERATION           0    0    0    1   -1
```

エラーが発生すると、そのエラーを示す値がerrnoに設定されて、-1 が返されます。

decc\$feature_get , decc\$feature_get_index , decc\$feature_get_name , decc\$feature_get_value , decc\$feature_show_allも参照してください。

Return value

0	成功したことを示します。
-1	エラーを示します。errnoが設定されます。

decc\$feature_show_all

すべての機能名について、その機能値をすべて表示します。

Format

```
#include <unixlib.h>
int decc$feature_show_all (void);
```

Description

`decc$feature_show_all`関数は、すべての機能名について、その機能値をすべて `stdout` に表示します。

エラーが発生すると、そのエラーを示す値が`errno`に設定されて、`-1`が返されます。

`decc$feature_get`, `decc$feature_get_index`, `decc$feature_get_name`, `decc$feature_get_value`, `decc$feature_set`,
および`decc$feature show`も参照してください。

Return value

0	成功したことを示します。
-1	エラーを示します。errnoが設定されます。

decc\$fix_time

OpenVMS バイナリ・システム時刻を UNIX バイナリ時刻に変換します。

Format

```
#include <unixlib.h>

unsigned int decc$fix_time (void *vms_time);
```

引数

vms_time
OpenVMS バイナリ時刻を格納したクォドワードのアドレス。

```
unsigned int quadword[2];
unsigned int *vms_time = quadword;
```

Description

decc\$fix_time ルーチンは、OpenVMS バイナリ・システム時刻 (1858 年 11 月 17 日 00:00 からの経過時間 (100 ナノ秒単位) を格納した 64 ビット・クォドワード) を UNIX バイナリ時刻 (1970 年 1 月 1 日 00:00 からの経過時間 (秒数) を格納したロングワード) に変換します。このルーチンは、OpenVMS システム・サービスおよび RMS サービスから返されたバイナリ時刻を、ctime や localtime などの HP C RTL ルーチンで使用される形式に変換するのに役立ちます。

Return value

x	1970 年 1 月 1 日 00:00 からの経過時間 (秒数) を格納したロングワード。
(unsigned int)(-1)	エラーを示します。戻り値 (unsigned int)(-1) は、2106 年 2 月 7 日 日曜日 06:28:15 という有効な日付も表すことに注意してください。

例

```
#include <unixlib.h>
#include <stdio.h>
#include <starlet.h> /* OpenVMS specific SYS$ routines) */

main()
{
    unsigned int current_vms_time[2]; /*quadword for OpenVMS time*/
    unsigned int number_of_seconds; /* number of seconds */

    /* first get the current system time */
    sys$gettim(&current_vms_time[0]);

    /* fix the time */
    number_of_seconds = decc$fix_time(&current_vms_time[0]);

    printf("Number of seconds since 00:00 January 1, 1970 = %d",
           number_of_seconds);
}
```

この例では、HP Cでdecc\$fix_timeルーチンを使用する方法を示しています。また、SYS\$GETTIM システム・サービスの使い方も示しています。

decc\$from_vms

OpenVMS ファイル指定を UNIX 形式のファイル指定に変換します。

Format

```
#include <unixlib.h>

int decc$from_vms (const char *vms_filespec, int action_routine, int wild_flag);
```

Argument

vms_filespec

OpenVMS ファイル指定形式の名前を格納したヌル区切り文字列のアドレス。

action_routine

指定された OpenVMS ファイル名から有効な UNIX 形式のファイル名への変換を格納したヌル区切り文字列を唯一の引数として受け付けるルーチンのアドレス。

action_routine が 0 以外の値 (TRUE) を返した場合は、ファイル変換は続行されます。0 (FALSE) を返した場合は、ファイル変換はそれ以上続行されません。

wild_flag

0 または 1 を値によって渡します。0 を指定した場合は、vms_filespec で検出されたワイルドカードは展開されません。0 以外の値を指定した場合は、ワイルドカードが展開され、それぞれの展開結果が action_routine に渡されます。展開されたファイル名のうち、既存の UNIX 形式のファイルに対応するファイル名だけが渡されます。

Description

decc\$from_vms ルーチンは、指定された OpenVMS ファイル指定に対応する UNIX 形式のファイル指定に変換します。OpenVMS ワイルドカードを指定することができます。ワイルドカードは UNIX 形式のファイル指定で、対応する既存のファイルのリストに変換されます。

戻り値

x

指定された OpenVMS ファイル指定から作成されたファイル名の数。

例

```

/* This example must be run as a foreign command      */
/* and be supplied with an OpenVMS file specification. */

#include <unixlib.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
    int number_found;          /* number of files found */
    int print_name();          /* name printer          */

    printf("Translating: %s\n", argv[1]);
    number_found = decc$from_vms(argv[1], print_name, 1);
    printf("\n%d files found", number_found);
}

/* print the name on each line */
print_name(char *name)
{
    printf("\n%s", name);
    /* will continue as long as success status is returned */
    return (1);
}

```

この例では、HP Cでdecc\$from_vmsルーチンを使用する方法を示しています。コマンド・ラインに指定された OpenVMS ファイル指定と一致する既存のファイルをリストしたlsコマンドの単純形式が作成されます。対応するファイルはUNIX形式のファイル指定で表示されます。

decc\$match_wild

文字列をパターンと照合します。

Format

```
#include <unixlib.h>

int decc$match_wild (char *test_string, char *string_pattern);
```

Argument

test_string

ヌル区切り文字列のアドレス。

string_pattern

照合するパターンを格納した文字列のアドレス。このパターンには正規表現 (範囲[a-z]など) だけでなく、ワイルドカード (アスタリスク(*), 疑問符(?), パーセント記号(%)) なども含むことができます。

Description

decc\$match_wildルーチンは、指定されたテスト文字列が、パターンによって指定される文字列セットのメンバであるかどうかを判断します。

Return value

1 (TRUE)

文字列はパターンと一致します。

0 (FALSE)

文字列はパターンと一致しません。

例

```
/* Define as a foreign command and then provide */
/* two arguments: test_string, string_pattern. */

#include <unixlib.h>
#include <stdio.h>
int main(int argc, char *argv[])
{
    if (decc$match_wild(argv[1], argv[2]))
        printf("\n%s matches %s", argv[1], argv[2]);
    else
        printf("\n%s does not match %s", argv[1], argv[2]);
}
```

decc\$record_read

レコードをファイルから読み込みます。

Format

```
#include <stdio.h>

int decc$record_read (FILE *fp, void *buffer, int nbytes);
```

Argument

fp
ファイル・ポインタ。ファイル・ポインタは、読み込みのために現在オープンされているファイルを参照しなければなりません。

buffer
入力データが格納される連続した記憶域のアドレス。

nbytes
読み込み操作で読み込まれる最大バイト数。

Description

decc\$record_read関数は OpenVMS システム固有であり、移植可能なアプリケーションを作成する場合は使用すべきではありません。

この関数はread関数と同じですが、最初の引数がファイル記述子ではなく、ファイル・ポインタである点が異なります。

Return value

x	読み込んだ文字数。
-1	物理的な入力エラー、不正なバッファ・アドレス、保護違反、未定義ファイル記述子などの読み込みエラーを示します。

decc\$record_write

レコードをファイルに書き込みます。

Format

```
#include <stdio.h>

int decc$record_write (FILE *fp, void *buffer, int nbytes);
```

Argument

fp
ファイル・ポインタ。ファイル・ポインタは、書き込みまたは更新のために現在オープンされているファイルを参照しなければなりません。

buffer
出力データが格納されている連続する記憶域のアドレス。

nbytes
書き込み操作で書き込まれる最大バイト数。

Description

decc\$record_write関数は OpenVMS システム固有であり、移植可能なアプリケーションを作成する場合は使用すべきではありません。

この関数はwrite関数と同じですが、最初の引数がファイル記述子ではなく、ファイル・ポインタである点が異なります。

Return value

x	書き込んだバイト数。
-1	未定義ファイル記述子、不正なバッファ・アドレス、物理 I/O エラーなどのエラーを示します。

decc\$set_child_default_dir (Alpha, I64)

execファミリの関数で生成される子プロセス用のデフォルト・ディレクトリを設定します。

Format

```
#include <unixlib.h>

int decc$set_child_default_dir (const char *default_dir);
```

Argument

default_dir
子プロセス用のデフォルト・ディレクトリの指定，または NULL。

Description

デフォルトでは，execファミリの関数で生成される子プロセスは，親プロセスのデフォルト (作業) ディレクトリを継承します。

decc\$set_child_default_dir関数を使用すると，子プロセス用のデフォルト・ディレクトリを設定できます。decc\$set_child_default_dirを呼び出した後，新しく生成される子プロセスは，実行開始時に，デフォルト・ディレクトリとしてdefault_dirが設定されます。default_dir引数は，正しいディレクトリ指定でなければなりません。正しくないと，この呼び出しの結果は予期できないものとなります (以後子プロセスを呼び出すと，何の通知もなく失敗することがあります)。この関数呼び出しでは，OpenVMS と UNIX の両方の形式のファイル指定がサポートされています。

default_dirに NULL を指定すると，デフォルトの動作に戻すことができます。以降は，新しく生成される子プロセスは，親プロセスの作業ディレクトリを継承します。

戻り値

0
成功を示します。新しいデフォルト・ディレクトリが継承されるようになります。

-1

失敗を示します。子プロセス用に新しいデフォルト・ディレクトリを設定することができませんでした。この関数は、次のいずれかの値をerrnoに設定します。

- ENOMEM – メモリ不足です。
- ENAMETOOLONG – 必要な SET DEFAULT コマンドを実行するには、default_dirが長すぎます。

decc\$set_child_standard_streams

exec関数ファミリの関数によって生成された子プロセスに対して、指定されたファイル記述子を子の標準ストリームstdin, stdout, stderrに関連付けます。

Format

```
#include <unixlib.h>

int decc$set_child_standard_streams (int fd1, int fd2, int fd3);
```

Argument

fd1

親プロセスでこのファイル記述子に関連付けられているファイルは、子プロセスでファイル記述子番号 0 (stdin) に関連付けられます。-1 を指定した場合、親のファイル記述子番号 0 に関連付けられているファイルが使用されます (デフォルト)。

fd2

親プロセスでこのファイル記述子に関連付けられているファイルは、子プロセスでファイル記述子番号 1 (stdout) に関連付けられます。-1 を指定した場合、親のファイル記述子番号 1 に関連付けられているファイルが使用されます (デフォルト)。

fd3

親プロセスでこのファイル記述子に関連付けられているファイルは、子プロセスでファイル記述子番号 2 (stderr) に関連付けられます。-1 を指定した場合、親のファイル記述子番号 2 に関連付けられているファイルが使用されます (デフォルト)。

Description

decc\$set_child_standard_streams関数を使用すると、指定されたファイル記述子と子のstdin/stdout/stderrストリームとの対応付けが可能になり、それによってOpenVMSシステムに実際のfork関数が欠如しているという問題をある程度補うことができます。

UNIX システムでは、forkとexecの間のコードは子プロセスのコンテキストで実行されます。

```

parent:
  create pipes p1, p2 and p3
  fork
child:
  map stdin to p1 like dup2(p1, stdin);
  map stdout to p2 like dup2(p2, stdout);
  map stderr to p3 like dup2(p3, stderr);

  exec (child reads from stdin and writes to stdout and stderr)
  exit
parent:
  communicates with the child using pipes

```

OpenVMS システムでは、同じタスクを次のように実行することができます。

```

parent:
  create pipes p1, p2 and p3
  decc$set_child_standard_streams(p1, p2, p3);
  vfork
  exec (child reads from stdin and writes to stdout and stderr)
parent:
  communicates with the child using pipes

```

decc\$set_child_standard_streamsの呼び出しで子の標準ストリームのマッピングを確立した後、このマッピングは、次のいずれかの呼び出しで明示的に無効にするまで有効です。

```
decc$set_child_standard_streams(-1, -1, -1);
```

または

```
decc$set_child_standard_streams(0, 1, 2);
```

通常、子プロセスは親のオープンされているすべてのファイル記述子を継承します。しかし、decc\$set_child_standard_streamsの呼び出しにファイル記述子番号nが指定されている場合は、親の記述子番号はファイル記述子番号nとして子プロセスに継承されず、子の標準ストリームの記述子番号になります。

注意

- 標準ストリームは、パイプにのみダイレクトすることができます。
- 親プロセスが DCL の DEFINE コマンドを再定義した場合、この再定義はユーザ定義チャネルを含むサブプロセスでは有効ではありません。サブプロセスは常に標準の DCL の DEFINE コマンドを使用します。
- 子プロセスが stdout と stderr に書き込んだすべての出力を使用するのは、親プロセスの責任です。サブプロセスが stdout と stderr に書き込む方法 (待機モードまたは待機なしモード) に応じて、サブプロセスは LEF 状態になり、読み込み側がデータを読み込むのを待つことがあります。たとえば、DCL は待機モードで SYS\$OUTPUT と SYS\$ERROR に書き込むの

で、DCL コマンド・プロシージャを実行する子プロセスは、すべての出力が親プロセスから読み込まれるまで待機状態になります。

推奨手順: EOF メッセージが受信されるまで、子プロセスのstdoutとstderrに関連付けられているパイプをループで読み込むか、またはこれらのメールボックスで書き込みアテンション AST を宣言してください。

- SYS\$OUTPUT に書き込まれるデータの量は、プロセスの確認状態 (SET VERIFY/NOVERIFY コマンド) に応じて異なります。サブプロセスは親プロセスの確認状態を継承します。サブプロセスが SYS\$OUTPUT に書き込むと考えられるデータの量に対応するように親プロセスの確認状態を設定するのは、呼び出し元の責任です。
- DTM など、一部のアプリケーションは、SYS\$ERROR を SYS\$OUTPUT として定義します。stderrが呼び出し元で再定義されていない場合は、サブプロセスで親の SYS\$ERROR として設定され、その場合、親の SYS\$OUTPUT に変換されます。

呼び出し元がstdoutをパイプに再定義し、stderrを再定義しなかった場合は、stderrに送信された出力は、stdoutに関連付けられているパイプに送られ、このメールボックスに書き込まれるデータの量は予想より多くなる可能性があります。標準チャンネルのサブセットの再定義はサポートされませんが、このような状況を回避するために、すべての標準チャンネル (少なくともstdoutとstderr) を明示的に再定義するのが常に安全です。

- DCL コマンド・プロシージャを実行する子プロセスの場合、SYS\$COMMAND は子のstdinに対して指定されたパイプに設定されるので、親プロセスはパイプを通じて SYS\$COMMAND から子が要求しているデータを送ることができます。DCL コマンド・プロシージャの場合、子の SYS\$INPUT を使用して親から子にデータを渡すことができます。これは、コマンド・プロシージャの場合、DCL は SYS\$INPUT をコマンド・ファイル自体として定義するからです。

Return value

x	子に対して設定されているファイル記述子の数。この数には、呼び出しに-1として指定されたファイル記述子は含まれません。
-1	不正なファイル記述子が指定されたことを示します。errnoはEBADFに設定されます。

 例

```

parent.c
=====

#include <stdio.h>
#include <string.h>
#include <unistd.h>

int decc$set_child_standard_streams(int, int, int);

main()
{
    int fdin[2], fdout[2], fderr[2];
    char msg[] = "parent writing to child's stdin";
    char buf[80];
    int nbytes;

    pipe(fdin);
    pipe(fdout);
    pipe(fderr);

    if ( vfork() == 0 ) {
        decc$set_child_standard_streams(fdin[0], fdout[1], fderr[1]);
        execl( "child", "child" );
    }
    else {
        write(fdin[1], msg, sizeof(msg));
        nbytes = read(fdout[0], buf, sizeof(buf));
        buf[nbytes] = '\0';
        puts(buf);
        nbytes = read(fderr[0], buf, sizeof(buf));
        buf[nbytes] = '\0';
        puts(buf);
    }
}

child.c
=====

#include <stdio.h>
#include <unistd.h>

main()
{
    char msg[] = "child writing to stderr";
    char buf[80];
    int nbytes;

    nbytes = read(0, buf, sizeof(buf));
    write(1, buf, nbytes);
    write(2, msg, sizeof(msg));
}

child.com
=====

```

decc\$set_child_standard_streams

```
$ read sys$command s
$ write sys$output s
$ write sys$error "child writing to stderr"
```

このサンプル・プログラムでは、child.cとchild.comの両方に対して次の情報が返されます。

```
$ run parent
parent writing to child's stdin
child writing to stderr
```

child.comを起動するには、parent.cプログラムで明示的にexecl("child.com", ...)を指定する必要があります。

decc\$set_reentrancy

リエントラントなHP C RTL ルーチンが示すリエントラントのタイプを制御します。

Format

```
#include <reentrancy.h>

int decc$set_reentrancy (int type);
```

引数

type

目的のリエントラントのタイプ。次のいずれかの値を使用します。

- `CSC_MULTITHREAD` — DECthreads 製品と組み合わせて使用するよう設計されています。DECthreads ロックを実行し、AST を絶対に無効にしません。この形式のリエントラントを使用するには、システムで DECthreads が使用可能でなければなりません。
- `CSC_AST` — `_BBSSI` (*VAX only*) または `_TESTBITSSI` (*Alpha, I64*) 組み込み関数を使用して、RTL コードのクリティカル・セクションの周囲で単純なロックを実行し、ロックされたコード領域で非同期システム・トラップ (AST) を無効にすることもあります。AST コードにHP C RTL I/O ルーチンの呼び出しが含まれている場合や、ユーザ・アプリケーションで AST を無効にする場合は、このタイプのロックを使用する必要があります。
- `CSC_TOLERANT` — `_BBSSI` (*VAX only*) または `_TESTBITSSI` (*Alpha, I64*) 組み込み関数を使用して、RTL コードのクリティカル・セクションの周囲で単純なロックを実行しますが、AST は無効になりません。AST が使用され、ただちに配布しなければならない場合は、このタイプのロックを使用する必要があります。TOLERANT はデフォルトのリエントラント・タイプです。
- `CSC_NONE` — HP C RTL で最適な性能を実現しますが、RTL コードのクリティカル・セクションの周囲で絶対にロックを行いません。HP C RTL を呼び出す AST によって実行スレッドが割り込まれる可能性がない場合に、単一スレッド環境でのみ使用するようにしなければなりません。

リエントラント・タイプは上げることができますが、下げることはできません。リエントラント・タイプを低いものから高いものへ順に並べると、`CSC_NONE`、`CSC_TOLERANT`、`CSC_AST`、`CSC_MULTITHREAD` の順になります。たとえば、アプリケーションをマルチスレッドに設定した後、リエントラントを AST に設定する呼び出しは無視されます。decc\$set_reentrancy の呼び出しでリエントラント・タイプを下げようとする、`-1` という値が返されます。

Description

リエントラント・ルーチンによって示されるリエントラントのタイプを変更するには、`decc$set_reentrancy`関数を使用します。

`decc$set_reentrancy`は非 AST レベルで明示的に呼び出す必要があります。

DECthreads を使用するアプリケーションでは、DECthreads が自動的にリエントラントをマルチスレッドに設定します。

戻り値

タイプ	この呼び出しの前に使用されていたリエントラントのタイプ。
-1	リエントラント・タイプを下げようとしたことを示します。

decc\$to_vms

UNIX 形式のファイル指定を OpenVMS ファイル指定に変換します。

Format

```
#include <unixlib.h>

int decc$to_vms (const char *unix_style_filespec, int (*action_routine)(char *OpenVMS_style_filespec,
int type_of_file), int allow_wild, int no_directory);
```

Argument

unix_style_filespec

UNIX のファイル指定形式で名前を格納したヌル区切り文字列のアドレス。

action_routine

次の引数を受け付ける decc\$to_vms ルーチンのアドレス。

- OpenVMS 形式への変換の結果となるヌル区切り文字列を指すポインタ。
- 次のいずれかの整数値。

値	変換
0 (DECC\$K_FOREIGN)	OpenVMS または VAXELN オペレーティング・システムを実行していないリモート・システムのファイル。
2 (DECC\$K_DIRECTORY)	UNIX 形式のファイル名から OpenVMS のディレクトリに変換することを示す。
1 (DECC\$K_FILE)	変換の対象がファイルであることを示す。

これらの値は、シンボル DECC\$K_FOREIGN、DECC\$K_DIRECTORY、DECC\$K_FILE によってシンボルとして定義することができます。詳細については、例を参照してください。

action_routine が 0 以外の値 (TRUE) を返した場合は、ファイル変換は続行されます。0 (FALSE) を返した場合は、それ以降のファイル変換は行われません。

allow_wild

0 または 1 を値によって渡します。0 を指定した場合は、unix_style_filespec で検出されたワイルドカードは展開されません。それ以外の場合は、ワイルドカードは展開され、それぞれの展開結果が action_routine に渡されます。展開されたファイル名のうち、既存の OpenVMS ファイルに対応するファイルだけが渡されます。

no_directory

次のいずれかの値の整数。

値	変換
0	ディレクトリは許可される。
1	文字列をディレクトリ名として展開することを禁止する。
2	強制的にディレクトリ名に変換する。

Description

decc\$to_vms関数は、指定された UNIX 形式のファイル指定を対応する OpenVMS ファイル指定 (すべて大文字) に変換します。UNIX 形式のワイルドカードを指定することができ、その場合は対応する OpenVMS ファイルのリストに変換されます。

decc\$to_vmsの動作に影響する次の機能論理名については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル(上巻)』第 1.6 節を参照してください。

DECC\$DISABLE_TO_VMS_LOGNAME_TRANSLATION
DECC\$NO ROOTED SEARCH LISTS

戻り値

x	指定された UNIX 形式のファイル指定から変換されたファイル名の数。
---	-------------------------------------

例

```

/* Translate "UNIX" wildcard file names to OpenVMS names */
/* Define as a foreign command and provide the name as */
/* an argument. */

#include <unixlib.h>
#include <stdio.h>
int print_name(char *, int);
int main(int argc, char *argv[])
{
    int number_found;          /* number of files found */
    printf("Translating: %s\n", argv[1]);
    number_found = decc$to_vms(argv[1], print_name, 1, 0);
    printf("%d files found\n", number_found);
}

/* action routine that prints name and type on each line */

```

```
int print_name(char *name, int type)
{
    if (type == DECC$K_DIRECTORY)
        printf("directory: %s\n", name);
    else if (type == DECC$K_FOREIGN)
        printf("remote non-VMS: %s\n", name);
    else
        printf("file:      %s\n", name);
    /* Translation continues as long as success status is returned */
    return (1);
}
```

この例では、HP Cでdecc\$to_vmsルーチンを使用する方法を示しています。引数としてUNIX形式のファイル指定を受け付け、OpenVMS ファイル指定形式で対応する既存の各ファイルの名前を表示します。

decc\$translate_vms

OpenVMS ファイル指定を UNIX 形式のファイル指定に変換します。

Format

```
#include <unixlib.h>

char *decc$translate_vms (const char *vms_filespec);
```

引数

vms_filespec

OpenVMS ファイル指定形式で名前を格納したヌル区切り文字列のアドレス。

Description

decc\$translate_vms関数は、ファイルが存在するかどうかにかかわらず、指定された OpenVMS ファイル指定を対応する UNIX 形式のファイル指定に変換します。変換された名前文字列はスレッド固有のメモリに格納され、同じスレッドから decc\$translate_vms を呼び出すたびに、このメモリの内容は上書きされます。

decc\$from_vms関数は既存のファイルだけを変換しますが、この関数はファイルが存在するかどうかにかかわらず変換します。

Return value

x	UNIX のファイル指定形式で名前を格納したヌル区切り文字列のアドレス。
0	ファイル名がヌルであるか、または構文が不正であることを示します。
-1	ファイル指定に反復記号 (たとえば[...]a.dat) が含まれているものの、その他の部分は正しいことを示します。OpenVMS の反復記号構文を正しい UNIX 形式のファイル指定に変換することはできません。

例

```
/* Demonstrate translation of a "UNIX" name to OpenVMS */
/* form, define a foreign command, and pass the name as */
/* the argument.                                         */

#include <unixlib.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
    char *ptr;                      /* translation result */
    ptr = decc$translate_vms( argv[1] );
    if ((int) ptr == 0 || (int) ptr == -1)
        printf( "could not translate %s\n", argv[1]);
    else
        printf( "%s is translated to %s\n", argv[1], ptr );
}
```

decc\$validate_wchar

現在のプログラムのロケールで、引数が有効なワイド文字であるかどうかを確認します。

Format

```
#include <unistd.h>

int decc$validate_wchar (wchar_t wc);
```

引数

wc
確認するワイド文字。

Description

decc\$validate_wchar関数は、指定されたwchar_t型の引数が現在のプログラムのロケールで有効なワイド文字であるかどうかを確認するための便利な方法として使用できます。

decc\$validate_wcharを呼び出す1つの理由として、isw*ワイド文字分類関数およびマクロが、文字プロパティを記述するclassmask配列の逆参照を行う前に、引数の有効性を確認できないという理由があげられます。現在のプログラムのロケールのワイド文字の最大値を超える値をisw*関数に渡すと、割り当てられているclassmask配列を超えるメモリにアクセスする可能性があります。

ワイド文字の有効性を確認するための標準的な方法は、wctomb関数を呼び出す方法ですが、この方法では、十分なサイズのマルチバイト文字配列を宣言し、その配列をwctombに渡す必要があるので、使用するのに不便です。

Return value

1	指定されたワイド文字が現在のプログラムのロケールで有効なワイド文字であることを示します。
0	指定されたワイド文字が現在のプログラムのロケールで有効なワイド文字でないことを示します。errnoは設定されません。

decc\$write_eof_to_mbx

ファイルの終端 (EOF) メッセージをメールボックスに書き込みます。

Format

```
#include <unistd.h>

int decc$write_eof_to_mbx (int fd);
```

引数

fd
メールボックスに関連付けられているファイル記述子。

Description

decc\$write_eof_to_mbx関数は、EOF (ファイルの終端) メッセージをメールボックスに書き込みます。

パイプでないメールボックスの場合、nbytes引数の値を0に設定してwrite関数を呼び出すと、EOFメッセージがメールボックスに送信されます。しかし、パイプの場合は、メールボックスにEOFメッセージを書き込むには、パイプをクローズするしかありません。

decc\$set_child_standard_streams関数の呼び出しによって子の標準入力パイプにリダイレクトされる場合は、親プロセスはこのパイプに対してdecc\$write_eof_to_mbxを呼び出すことで、子にEOFメッセージを送信することができます。子がデータを端末から読み込み、Ctrl/Zが押された場合と同じ結果になります。

decc\$write_eof_to_mbxを呼び出した後、たとえば他の子との通信のために、パイプを再利用することができます。これがdecc\$write_eof_to_mbxの目的です。つまり、EOFメッセージを送信するためだけにパイプをクローズするのではなく、パイプを再利用することを許可します。

Return value

0	正常終了を示します。
-1	異常終了を示します。errnoとvaxc\$errnoは、SYSSQIOW から返された異常終了状態に従って設定されます。

例

```

/*      decc$write_eof_to_mbx_example.c      */
#include <errno.h>
#include <stdio.h>
#include <string.h>

#include <fcntl.h>
#include <unistd.h>
#include <unixio.h>

#include <descrip.h>
#include <ssdef.h>
#include <starlet.h>

int decc$write_eof_to_mbx( int );

main()
{
    int status, nbytes, failed = 0;
    int fd, fd2[2];
    short int channel;
    $DESCRIPTOR(mbxname_dsc, "TEST_MBX");
    char c;

    /* first try a mailbox created by SYS$CREMBX      */
    status = sys$crembx(0, &channel, 0, 0, 0, 0, &mbxname_dsc, 0, 0);
    if ( status != SS$_NORMAL ) {
        printf("sys$crembx failed: %s\n",strerror(EVMSERR, status));
        failed = 1;
    }

    if ( (fd = open(mbxname_dsc.dsc$a_pointer, O_RDWR, 0)) == -1 ) {
        perror("? open mailbox");
        failed = 1;
    }

    if ( decc$write_eof_to_mbx(fd) == -1 ) {
        perror("? decc$write_eof_to_mbx to mailbox");
        failed = 1;
    }
}

```

```

if ( (nbytes = read(fd, &c, 1)) != 0 || errno != 0 ) {
    perror("? read mailbox");
    printf("? nbytes = %d\n", nbytes);
    failed = 1;
}

if ( close(fd) == -1 ) {
    perror("? close mailbox");
    failed = 1;
}

/* Now do the same thing with a pipe */
errno = 0;          /* Clear errno for consistency */
if ( pipe(fd2) == -1 ) {
    perror("? opening pipe");
    failed = 1;
}

if ( decc$write_eof_to_mbx(fd2[1]) == -1 ) {
    perror("? decc$write_eof_to_mbx to pipe");
    failed = 1;
}

if ( (nbytes = read(fd2[0], &c, 1)) != 0 || errno != 0 ) {
    perror("? read pipe");
    printf("? nbytes = %d\n", nbytes);
    failed = 1;
}

/* Close both file descriptors involved with the pipe */
if ( close(fd2[0]) == -1 ) {
    perror("close(fd2[0])");
    failed = 1;
}

if ( close(fd2[1]) == -1 ) {
    perror("close(fd2[1])");
    failed = 1;
}

if ( failed )
    puts("?Example program failed");
else
    puts("Example ran to completion");
}

```

このサンプル・プログラムを実行すると、次の結果が生成されます。

Example ran to completion

[w]delch

指定されたウィンドウで現在のカーソルの位置にある文字を削除します。delch関数はstdscrウィンドウに対して動作します。

Format

```
#include <curses.h>

int delch();

int wdelch (WINDOW *win);
```

引数

win
ウィンドウを指すポインタ。

Description

カーソルと同じ行で、カーソルの右側にある文字はすべて左に移動し、行末までブランク文字が追加されます。

Return value

OK	正常終了を示します。
ERR	エラーを示します。

delete

ファイルを削除します。

Format

```
#include <unixio.h>

int delete (const char *file_spec);
```

引数

file_spec

OpenVMS または UNIX 形式のファイル指定である文字列を指すポインタ。ファイル指定のバージョン番号にはワイルドカードを指定できますが、他の部分には指定できません。たとえば、filename.txt;* という形式のファイルは削除することができます。

Description

ファイル名にディレクトリを指定し、それがエラーを含む検索リストである場合は、HP C for OpenVMS システムはそのディレクトリをファイル・エラーとして解釈します。

シンボリック・リンクを指定して delete を実行した場合は、そのリンクから参照されているファイルではなく、そのリンク自体が削除されます。

HP C RTL では、remove 関数と delete 関数は機能的に同じです。

remove も参照してください。

注意

C++ プログラマは delete ルーチンを使用できません。これは、C++ の予約語である delete と競合するからです。C++ プログラマは代わりに ANSI/ISO C 標準関数である remove を使用してください。

Return value

0	正常終了を示します。
0 以外の値	操作が異常終了したことを示します。

[w]deleteln

現在のカーソルの位置にある行を削除します。deletelnはstdscrウィンドウに対して動作します。

Format

```
#include <curses.h>

int deleteln();

int wdeleteln (WINDOW *win);
```

引数

win
ウィンドウを指すポインタ。

Description

削除した行の下にある行はすべて上に移動し、一番下の行はブランク行になります。カーソルの現在の座標 (y,x) は変更されません。

Return value

OK	正常終了を示します。
ERR	エラーを示します。

delwin

指定されたウィンドウをメモリから削除します。

Format

```
#include <curses.h>
int delwin (WINDOW *win);
```

引数

win
ウィンドウを指すポインタ。

Description

削除するウィンドウにサブウィンドウが含まれている場合は、サブウィンドウは無効になります。親を削除する前にサブウィンドウを削除してください。delwin関数は、削除したウィンドウが覆っていたすべてのウィンドウの表示を更新します。

Return value

OK	正常終了を示します。
ERR	エラーを示します。

time1引数とtime2引数によって指定される 2 つの時刻の差を秒数として計算します。

Format

Argument

Description

戻り値

REF-164

dirname

ファイル・パス名の親ディレクトリ名を報告します。

Format

```
#include <libgen.h>

char *dirname (char *path);
```

関数バリエーション

dirname関数には、_dirname32および_dirname64という名前のバリエーションがあり、それぞれ 32 ビット・ポインタ・サイズおよび 64 ビット・ポインタ・サイズで使用されます。ポインタ・サイズ固有の関数の使い方については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

引数

path
ファイル・パス名。

Description

dirname関数は、UNIX パス名を格納した文字列を指すポインタを受け付け、そのファイルの親ディレクトリのパス名を格納した文字列を指すポインタを返します。パスの後続のスラッシュ(/)文字はパスの一部として解釈されません。

path引数が次のいずれかである場合、この関数は文字列 "." (ドット) を指すポインタを返します。

- スラッシュ(/)が含まれていない。
- NULL ポインタである。
- 空文字列を指す。

dirname関数は、path引数によって示される文字列を変更することがあります。

dirname関数とbasename関数によって完全なパス名が作成されます。dirname(path) という式は、basename(path) が見つかったディレクトリのパス名を取得します。

basenameも参照してください。

Return value

x	path引数の親ディレクトリである文字列を指すポインタ。
"."	path引数が次のいずれかである場合。 <ul style="list-style-type: none">• スラッシュ(/)が含まれていない。• NULL ポインタである。• 空文字列を指す。

例

dirname関数を使用すると、次の例はパス名を読み込み、現在のワーキング・ディレクトリを親ディレクトリに変更し、ファイルをオープンします。

```
char path [MAXPATHLEN], *pathcopy;
int fd;

fgets(path, MAXPATHLEN, stdin);
pathcopy = strdup(path);
chdir(dirname(pathcopy));
fd = open(basename(path), O_RDONLY);
```

div

引数を除算した後，商と余りを返します。

Format

```
#include <stdlib.h>
div_t div (int numer, int denom);
```

Argument

numer
int型の分子。

denom
int型の分母。

Description

div_t型は，次に示すように<stdlib.h>標準ヘッダ・ファイルに定義されています。

```
typedef struct
{
    int    quot, rem;
} div_t;
```

dlclose

共用ライブラリのアドレス空間の割り当てを解除します。

Format

```
#include <dlfcn.h>

void dlclos (void *handle);
```

引数

handle
共用ライブラリを指すポインタ。

Description

dlclose関数は、ハンドルのためにHP C RTL によって割り当てられたアドレス空間の割り当てを解除します。

OpenVMS システムでは、LIB\$FIND_IMAGE_SYMBOL ルーチンによって動的にロードされた共用可能イメージをアンロードする手段はありません。LIB\$FIND_IMAGE_SYMBOL ルーチンはdlsym関数によって呼び出されます。つまり、OpenVMS システムでは、dlsymによってメモリに読み込まれた共用可能イメージが使用しているアドレス空間を解放する手段はありません。

dlerror

dlopen , dlclose , dlsymの呼び出しで発生した最後のエラーを記述する文字列を返します。

Format

```
#include <dlfcn.h>

char *dlerror (void);
```

戻り値

x	dlopen , dlclose , dlsymの呼び出しで発生した最後のエラーを記述する文字列。
---	---

dlopen

実行時に共用可能イメージのロードと呼び出しを可能にするために、動的ライブラリ・ローダに対するインタフェースを提供します。

Format

```
#include <dlfcn.h>

void *dlopen (char *pathname, int mode);
```

Argument

pathname

共用可能イメージの名前。この名前は、この後、`dlsym`関数で使用するために保存されます。

mode

この引数は OpenVMS システムでは無視されます。

Description

`dlopen`関数は、実行時に共用可能イメージのロードと呼び出しを可能にするために、動的ライブラリ・ローダに対するインタフェースを提供します。

この関数は共用可能イメージをロードするわけではなく、`dlsym`関数でこの後使用するためにpathname引数を保存します。`LIB$FIND_IMAGE_SYMBOL` を呼び出すことにより、共用可能イメージを実際にロードする関数は`dlsym`です。

`dlopen`関数のpathname引数は、共用可能イメージの名前でなければなりません。この名前はfilename引数として、`dlsym`関数から `LIB$FIND_IMAGE_SYMBOL` ルーチンにそのまま渡されます。`LIB$FIND_IMAGE_SYMBOL` の呼び出しにimage-name引数を指定しないと、デフォルトのファイル指定である `SYSS$SHARE:.EXE` がイメージ名に適用されます。

`dlopen`関数は、`dlsym`または`dlclose`呼び出しで使用されるハンドルを返します。エラーが発生した場合は、`NULL` ポインタが返されます。

Return value

x

dlsymまたはdlclose呼び出しで使用されるハンドル。

NULL

エラーを示します。

dlsym

共用可能イメージから検索したシンボル名のアドレスを返します。

Format

```
#include <dlfcn.h>

void *dlsym (void *handle, char *name);
```

Argument

handle
共用可能イメージを指すポインタ。

name
シンボル名を指すポインタ。

Description

dlsym関数は、共用可能イメージからhandleに対応するシンボル名を検索して、そのアドレスを返します。シンボルが見つからない場合は、NULLポインタを返します。

OpenVMS Version 7.3-2 以降では、小文字が含まれるライブラリ・シンボルをdlsym関数を使用してロードできるようになりました。全体的に、ライブラリを動的にロードする関数(dlopen, dlsym, dlclose, dlerror)は、以下の機能を提供するように拡張されました。

- 大文字と小文字が混じったシンボル名を持つライブラリのサポート
- 完全ファイルパスをdlopenに渡すことが可能
- 指定されたライブラリ名の検証

Return value

x	検索されたシンボル名のアドレス。
NULL	シンボルが見つからなかったことを示します。

drand48

均一に分布した擬似乱数シーケンスを生成します。48 ビットの負でない倍精度浮動小数点数値を返します。

Format

```
#include <stdlib.h>

double drand48 (void);
```

Description

drand48関数は、線形合同アルゴリズム (linear congruential algorithm) と 48 ビット整数算術演算を使用して、擬似乱数を生成します。

$0.0 \leq y < 1.0$ の範囲で均一に分布した、負でない倍精度浮動小数点数値を返します。

drand48を呼び出す前に、srand48, seed48, lcong48のいずれかを使用して、乱数ジェネレータを初期化します。drand48関数を起動する前に初期化が必要なのは、最後に生成された 48 ビット X_i が内部バッファに格納されるからです (推奨できる方法ではありませんが、最初に初期化関数を呼び出さずに、drand48, lrand48, mrand48関数のいずれかを呼び出した場合、一定のデフォルト初期化値が自動的に与えられます)。

drand48関数は、線形合同公式に従って、48 ビット整数値 X_i のシーケンスを生成することにより動作します。

$$X_{n+1} = (aX_n + c) \bmod m \quad n \geq 0$$

引数 m は 2^{48} に等しいため、48 ビットの整数演算が実行されます。lcong48関数を呼び出した場合を除き、乗数値 a と加数値 c は次のようになります。

```
a = 5DEECE66D16 = 2736731631558
c = B16 = 138
```

drand48から返される値は、シーケンス内の次の 48 ビット X_i を最初に生成することにより計算されます。その後、返されるデータ項目の型に従って、適切なビットが X_i の上位 (最上位) ビットからコピーされ、戻り値に変換されます。

srand48, seed48, lcong48, lrand48, mrand48も参照してください。

drand48

戻り値

n

負でない倍精度浮動小数点数値。

dup, dup2

`open` , `creat` , `pipe`から返されたファイル記述子によって指定されるファイルを参照する新しい記述子を割り当てます。

Format

```
#include <unistd.h>

int dup  (int file_desc1);

int dup2 (int file_desc1, int file_desc2);
```

Argument

`file_desc1`

複製するファイル記述子。

`file_desc2`

`file_desc1`によって指定されるファイルに割り当てられる新しいファイル記述子。

Description

`dup`関数を呼び出すと、前に割り当てが解除された記述子が引数を参照するようになります。一方、`dup2`関数を使用すると、2番目の引数が最初の引数と同じファイルを参照するようになります。

引数`file_desc1`がオープンされているファイルを記述しない場合は、この引数は不正です。新しいファイル記述子を割り当てることができない場合は、`file_desc2`は不正です。`file_desc2`がオープンされているファイルに接続されている場合は、そのファイルはクローズされます。

Return value

<code>n</code>	新しいファイル記述子。
<code>-1</code>	不正な引数が関数に渡されたことを示します。

[no]echo

文字が端末画面に表示されるのか，表示されないのかを設定します。このシングル文字入力モードは，Curses でのみサポートされます。

Format

```
#include <curses.h>

void echo  (void);
void noecho (void);
```

Description

noecho関数は，wgetchおよびwgetstrで端末画面から入力を受け付けるときに役立ちます。この関数は，入力された文字が画面に表示されないようにします。

ecvt

引数をヌル区切りの ASCII 数字列に変換し、数字列のアドレスを返します。この数字列は、HP C RTL で作成されたスレッド固有のメモリ記憶位置に格納されます。

Format

```
#include <stdlib.h>

char *ecvt (double value, int ndigits, int *decpt, int *sign);
```

Argument

value

ヌルで区切られた ASCII 数字列に変換される double 型のオブジェクト。

ndigits

変換後の数字列で使用する ASCII 数字の桁数。

decpt

返される数字列の 1 文字目を基準にした小数点の位置。int の値が負の場合は、小数点の位置は、返された数字の左側に decpt 個のスペースを付加した位置になります (スペースには 0 が埋められます)。値が 0 の場合は、小数点の位置は返された数字列の最初の桁のすぐ左に設定されます。

sign

value 引数が正の値であるか、負の値であるかを示す整数値。value が負の場合は、sign によって指定されるアドレスに 0 以外の値が代入されます。この引数が負でない場合は、sign によって指定されるアドレスに 0 が代入されます。

Description

ecvt 関数は、value を長さ ndigits のヌル区切り数字列に変換し、数字列を指すポインタを返します。C の E 形式で ndigits 桁の数字を出力するために、下位桁が適切な桁に丸められます。decpt 引数は、数字列の最初の数字を基準にして小数点の位置を表します。

ecvt 関数を繰り返し呼び出すと、既存の数字列は上書きされます。

ecvt、fcvt、gcvt 関数は、浮動小数点算術演算に関して IEEE 標準で指定されている次の特殊な値を表します。

値	表現
クワイエット NaN	NaNQ
シグナリング NaN	NaNS
+ 無限大	無限大
−無限大	−無限大

これらの各値に割り当てられる符号は，sign引数に格納されます。IEEE 浮動小数点表現では，0 (ゼロ) という値は正の場合も負の場合もあり，どちらであるかはsign引数によって設定されます。

gcvtとfcvtも参照してください。

戻り値

x 変換後の文字列の値。

encrypt

setkey関数で生成したキーを使用して、文字列を暗号化します。

Format

```
#include <unistd.h>
#include <stdlib.h>
void encrypt (char *block[64], int edflag;)
```

引数

block

0 と 1 を含む、長さ 64 の文字配列。

edflag

整数値。edflag として 0 を指定すると、引数が暗号化されます。edflag として 0 以外の値を指定すると、引数が復号化されます。

Description

encrypt関数は、setkey関数で生成したキーを使用して、文字列を暗号化します。

encryptの 1 番目の引数には、0 と 1 を含む、長さ 64 の文字配列を指定します。この配列が、その位置で新しい配列に変換されます。新しい配列は、元の配列の内容に対して、setkeyで設定したキーを使用して DES アルゴリズムを適用したものになります。

2 番目の引数edflagによって、1 番目の引数を暗号化するのか、または復号化するのかを指定します。edflagとして 0 を指定すると、1 番目の引数は暗号化されます。0 以外の値を指定すると、復号化されます。

戻り値はありません。

cryptとsetkeyも参照してください。

encrypt

戻り値

ポインタ

暗号化されたパスワードへのポインタ。

endgrent (*Alpha, I64*)

処理完了時に、グループ・データベースをクローズします。

Format

```
#include <grp.h>
void endgrent (void);
```

Description

endgrent関数は、グループ・データベースをクローズします。

この関数は、必ず成功します。戻り値はなく、errnoは設定されません。

endpwent

getpwentが使用したユーザ・データベースとプライベート・ストリームをクローズします。

Format

```
#include <pwd.h>

void endpwent (void);
```

Description

endpwent関数は、getpwentが使用したユーザ・データベースとプライベート・ストリームをクローズします。

戻り値はありません。入出力エラーが発生した場合、この関数はerrnoにEIOを設定します。

getpwent, getpwuid, getpwnam, setpwentも参照してください。

endwin

端末画面をクリアし、Curses 構造体に割り当てられている仮想メモリを解放します。

Format

```
#include <curses.h>

void endwin (void);
```

Description

Curses 関数を呼び出すプログラムは、終了する前にendwin関数を呼び出して、端末画面の元の環境を復元する必要があります。

erand48

均一に分布した擬似乱数シーケンスを生成します。48 ビットの負でない倍精度浮動小数点数値を返します。

Format

```
#include <stdlib.h>

double erand48 (unsigned short int xsubi[3]);
```

引数

xsubi
3 つのshort intの配列。結合すると 48 ビット整数が作成されます。

Description

erand48関数は、線形合同アルゴリズム (linear congruential algorithm) および 48 ビット整数演算を使用して、擬似乱数を生成します。

$0.0 \leq y < 1.0$ の範囲で均一に分布した負でない倍精度浮動小数点数値を返します。

erand48関数は、線形合同公式に従って、48 ビットの整数値 X_i のシーケンスを生成することにより動作します。

$$X_{n+1} = (aX_n + c) \bmod m \quad n \geq 0$$

引数 m は 2^{48} に等しいため、48 ビットの整数演算が実行されます。lcong48関数を呼び出した場合を除き、乗数値 a と加数値 c は次のようになります。

$$\begin{aligned} a &= 5DEECE66D_{16} = 2736731631558 \\ c &= B_{16} = 138 \end{aligned}$$

erand48関数を使用する場合、呼び出し元のプログラムはxsubi引数として配列を渡さなければなりません。最初の呼び出しでは、配列を擬似乱数シーケンスの値に初期化しなければなりません。drand48関数と異なり、最初の呼び出しの前に初期化関数を呼び出す必要はありません。

erand48関数では、異なる引数を使用することにより、大きなプログラムの個別のモジュールが複数の独立した擬似乱数シーケンスを生成することができます。たとえ

ば、1つのモジュールが生成する乱数シーケンスは、他のモジュールから関数が呼び出される回数に依存しません。

戻り値

n 負でない倍精度浮動小数点数値。

[w]erase

ブランクを書き込むことにより，ウィンドウを消去します。erase関数はstdscrウィンドウに対して動作します。

Format

```
#include <curses.h>

int erase();

int werase (WINDOW *win);
```

引数

win
ウィンドウを指すポインタ。

Description

erase関数とwerase関数はどちらも，終了した後，カーソルを端末画面の現在の位置に保持します。カーソルはホーム座標 (0,0) に戻りません。

Return value

OK	正常終了を示します。
ERR	エラーを示します。

erf

引数のエラー関数を返します。

Format

```
#include <math.h>

double erf (double x);
float erff (float x); (Alpha, I64)
long double erfl (long double x); (Alpha, I64)
double erfc (double x); (Alpha, I64)
float erfcf (float x); (Alpha, I64)
long double erfcl (long double x); (Alpha, I64)
```

引数

x
実数値で表したラジアン値。

Description

erf関数は x のエラー関数を返します。ただし, erf(x), erff(x), erfl(x) は, 0 ~ xの範囲で曲線 $e^{-(t^2)}$ の下の領域の $2/\sqrt{\pi}$ 倍に等しい値です。

erfc関数は $(1.0 - \text{erf}(x))$ を返します。erfc関数を呼び出すと, xが大きくなるにつれてアンダフローが発生する可能性があります。

Return value

x	エラー関数 (erf) または補数エラー関数 (erfc) の値。
NaN	xが NaN です。errnoは EDOM に設定されます。
0	アンダフローが発生しました。errnoは ERANGE に設定されます。

execl

子プロセスで起動されるイメージの名前を渡します。この関数は非リエントラントです。

Format

```
#include <unistd.h>

int execl (const char *file_spec, const char *arg0, ... , (char *)0); (ISO POSIX-1)
int execl (char *file_spec, ... ); (Compatability)
```

Argument

file_spec

子プロセスで起動される新しいイメージの完全なファイル指定。

arg0, ...

ヌル区切り文字列を指すポインタのシーケンス。

POSIX-1 形式を使用する場合は、少なくとも 1 つの引数を指定する必要があり、その引数は新しいプロセス・ファイル名 (またはその最後のコンポーネント) と同一である文字列を指さなければなりません (このポインタは NULL ポインタでも構いませんが、その場合は、execl は何も実行しません)。最後のポインタは NULL ポインタでなければなりません。互換形式を使用するときも、この規則が適用されます。

Description

exec 関数の動作方法を理解するには、次の構文に示すように、OpenVMS システムが HP C プログラムを呼び出す方法を考慮する必要があります。

```
int main (int argc, char *argv[], char *envp[]);
```

識別子 argc は引数の数です。argv は引数文字列の配列です。配列の最初のメンバ (argv[0]) にはイメージの名前が格納されます。引数は配列の 2 番目以降の要素に格納されます。配列の最後の要素は常に NULL ポインタです。

exec 関数は、実行時システムが他の HP C プログラムを呼び出すのと同じ方法で子プロセスを呼び出します。exec 関数は、子で起動されるイメージの名前を渡します。この値は argv[0] に格納されます。しかし、これらの関数は、引数および環境情報を子に渡す方法が異なります。

- 引数は個別の文字列で渡すことができ (execl , execle , execlp) , 文字列配列で渡すこともできます (execv , execve , execvp)。
- 環境は配列で明示的に渡すことができ (execleとexecve) , 親の環境から取得することもできます (execl , execv , execlp , execvp)。

exec関数を起動する前にvforkが呼び出された場合は、exec関数が終了したときに、制御はvfork呼び出しの時点の親プロセスに返されます。vforkが呼び出されていない場合は、exec関数は、子が実行を終了するまで待ち、実行が終了すると、親プロセスを終了します。詳細については、vforkおよび『HP C ランタイム・ライブラリ・リファレンス・マニュアル(上巻)』第5章を参照してください。

戻り値

-1

異常終了を示します。

execle

子プロセスで起動されるイメージの名前を渡します。この関数は非リエントラントです。

Format

```
#include <unistd.h>

int execle (char *file_spec, char *arg0, . . . , (char *)0, char *envp[]); (ISO POSIX-1)
int execle (char *file_spec, . . . ); (Compatability)
```

Argument

file_spec

子プロセスで起動される新しいイメージの完全なファイル指定。

arg0, ...

ヌル区切り文字列を指すポインタのシーケンス。

POSIX-1 形式を使用する場合は、少なくとも 1 つの引数を指定する必要があり、その引数は新しいプロセス・ファイル名 (またはその最後のコンポーネント) と同一である文字列を指さなければなりません (このポインタは NULL ポインタでも構いませんが、その場合は、execleは何も実行しません)。最後のポインタは NULL ポインタでなければなりません。互換形式を使用するときも、この規則が適用されます。

envp

プログラムの環境を指定する文字列配列。envpの各文字列は次の形式です。

name = value

name は次の名前のいずれかとして指定でき、value は name に関連付けられるヌル区切り文字列です。

- HOME — ログイン・ディレクトリ
- TERM — 使用している端末の種類
- PATH — デフォルトのデバイスとディレクトリ
- USER — プロセスを開始したユーザの名前

envpの最後の要素は NULL ポインタでなければなりません。

オペレーティング・システムがプログラムを実行する場合、現在の環境ベクタ (envp) のコピーを外部変数 environ に格納します。

Description

exec関数の動作方法については、execlを参照してください。

戻り値

-1

異常終了を示します。

execlp

子プロセスで起動されるイメージの名前を渡します。この関数は非リエントラントです。

Format

```
#include <unistd.h>

int execlp (const char *file_name, const char *arg0, . . . , (char *)0); (ISO POSIX-1)
int execlp (char *file_name, . . . ); (Compatability)
```

Argument

file_name

子プロセスで起動される新しいイメージのファイル名。ファイルのデバイスおよびディレクトリ指定は、VAXC\$PATH 環境名を検索することにより取得されます。

argn

ヌル区切り文字列を指すポインタのシーケンス。規則では、少なくとも 1 つの引数を指定しなければならず、その引数は新しいプロセス・ファイル名 (またはその最後のコンポーネント) と同一である文字列を指すポインタでなければなりません。

. . .

文字列を指すポインタのシーケンス。リストを終了するために少なくとも 1 つのポインタが必要です。このポインタは NULL でなければなりません。

Description

exec 関数の動作方法については、execl を参照してください。

戻り値

-1

異常終了を示します。

execv

子プロセスで起動されるイメージの名前を渡します。この関数は非リエントラントです。

Format

```
#include <unistd.h>

int execv (char *file_spec, char *argv[]);
```

Argument

file_spec

子プロセスで起動される新しいイメージの完全なファイル指定。

argv

ヌル区切り文字列を指すポインタの配列。これらの文字列は、新しいプロセスで使用できる引数リストを構成します。規則では、argv[0]は、新しいプロセス・ファイル名(またはその最後のコンポーネント)と同一である文字列を指すポインタでなければなりません。argvは NULL ポインタで終了します。

Description

exec関数の動作方法については、execlを参照してください。

戻り値

−1

異常終了を示します。

execve

子プロセスで起動されるイメージの名前を渡します。この関数は非リエントラントです。

Format

```
#include <unistd.h>

int execve (const char *file_spec, char *argv[], char *envp[]);
```

Argument

file_spec

子プロセスで起動される新しいイメージの完全なファイル指定。

argv

ヌル区切り文字列を指すポインタの配列。これらの文字列は新しいプロセスで使用できる引数リストを構成します。規則では、argv[0]は、新しいプロセス・ファイル名(またはその最後のコンポーネント)と同一である文字列を指すポインタでなければなりません。argvは NULL ポインタで終了します。

envp

プログラムの環境を指定する文字列配列。envpの各文字列は次の形式です。

name = value

name は次の名前のいずれかとして指定でき、value は name に関連付けられるヌル区切り文字列です。

- HOME — ログイン・ディレクトリ
- TERM — 使用している端末の種類
- PATH — デフォルトのデバイスとディレクトリ
- USER — プロセスを開始したユーザの名前

envpの最後の要素は NULL ポインタでなければなりません。

オペレーティング・システムがプログラムを実行する場合、現在の環境ベクタ (envp) のコピーを外部変数 environ に格納します。

Description

exec関数の動作方法については、execlを参照してください。

戻り値

-1

異常終了を示します。

execvp

子プロセスで起動されるイメージの名前を渡します。この関数は非リエントラントです。

Format

```
#include <unistd.h>

int execvp (const char *file_name, char *argv[]);
```

Argument

file_name

子プロセスで起動される新しいイメージのファイル名。ファイルのデバイスおよびディレクトリ指定は環境名 VAXC\$PATH を検索することにより取得されます。

argv

ヌル区切り文字列を指すポインタの配列。これらの文字列は新しいプロセスで使用できる引数リストを構成します。規則では、argv[0]は、新しいプロセス・ファイル名（またはその最後のコンポーネント）と同一である文字列を指すポインタでなければなりません。argvは NULL ポインタで終了します。

Description

exec関数の動作方法については、execlを参照してください。

戻り値

−1

異常終了を示します。

exit, _exit

関数を呼び出したプログラムの実行を終了します。これらの関数は非リエントラントです。

Format

```
#include <stdlib.h>

void exit (int status);

#include <unistd.h>

void _exit (int status);
```

引数

status

状態値 EXIT_SUCCESS (1) または EXIT_FAILURE (2) , または 3 ~ 255 の数値。

- 状態値 1 または EXIT_SUCCESS は , OpenVMS の SSS_NORMAL 状態コードに変換され , OpenVMS の正常終了値を返します。
- 状態値 2 または EXIT_FAILURE はエラー・レベル終了状態に変換されます。状態値は親プロセスに渡されます。
- 他の状態値はそのまま変更されません。

これらの状態値を使用するには , <unistd.h>を取り込み , _POSIX_EXIT 機能テスト・マクロを設定してコンパイルします (ファイルを取り込む前に , ファイルの先頭に#define _POSIX_EXITを指定するか , または/DEFINE=_POSIX_EXIT を指定します)。この動作が可能なのは , OpenVMS Version 7.0 およびそれ以降のシステムの場合だけです。

Description

プロセスが DCL で起動された場合 , 状態は DCL によって解釈され , メッセージが表示されます。

プロセスがvforkまたはexec関数を使用して生成された子プロセスの場合は , 子プロセスは終了し , 制御は親に返されます。2つの関数は同じです。_exit関数が残されているのは , VAX Cとの互換性を維持するためです。

exit関数と_exit関数では、SEXIT システム・サービスを使用します。ハイバネーションおよびスケジューリングされたウェイクアップ修飾子を使用して RUN コマンドによってプロセスが起動される場合は、exitまたは_exit呼び出しが実行されたときに、プロセスはハイバネーション状態に正しく戻らないことがあります。

C コンパイラでは、コマンド行修飾子/[NO]MAIN=POSIX_EXIT を使用することで、mainから戻るときに、exitではなく__posix_exitを呼び出させるようにできます。デフォルトは、/NOMAIN です。

注意

EXIT_SUCCESS と EXIT_FAILURE はどのANSI Cコンパイラ間でも移植可能であり、それぞれ正常終了と異常終了を示します。OpenVMS システムでは、OpenVMS の条件コードに変換され、重大度がそれぞれ正常終了または異常終了に設定されます。子プロセスでは、親プロセスに渡すデータの量を少なくするために、3 ~ 255 の範囲の値を使用できます。親はwait, wait3, wait4, waitpid関数を使用してこのデータを取得します。

exp

底 e に対する引数のべき乗を返します。

Format

```
#include <math.h>
double exp (double x);
float expf (float x); (Alpha, I64)
long double expl (long double x); (Alpha, I64)
double expm1 (double x); (Alpha, I64)
float expm1f (float x); (Alpha, I64)
long double expm1l (long double x); (Alpha, I64)
```

引数

x
実数値。

Description

`exp`関数は、 e^{**x} として定義される指数関数の値を計算します。ただし、 e は自然対数の底として使用される定数です。

`expm1`関数は、 x の値が小さい場合でも、 $\exp(x) - 1$ を正確に計算します。

オーバーフローが発生した場合、`exp`関数は可能な最大の浮動小数点数値を返し、`errno`を `ERANGE` に設定します。可能な最大の浮動小数点数値を表すために、定数 `HUGE_VAL` が `<math.h>` ヘッダ・ファイルに定義されています。

Return value

x	引数の指数値。
HUGE_VAL	オーバーフローが発生したことを示します。errnoはERANGEに設定されます。
0	アンダフローが発生したことを示します。errnoはERANGEに設定されます。
NaN	xがNaNであることを示します。errnoはEDOMに設定されます。

exp2 (Alpha, I64)

2 のべき乗値，つまり引数で指定した回数だけ 2 をべき乗した値を返します。

Format

```
#include <math.h>

double exp2 (double x);
float exp2f (float x);
long double exp2l (long double x);
```

引数

x
実数値。

Description

exp2関数は，2 を底とするべき乗値，つまり 2 をx回だけべき乗した値を計算します。

オーバーフローが発生すると，exp関数は errnoに値 ERANGE を設定するとともに，表現可能な浮動小数点数の最大値を返します。この浮動小数点数の最大値を表すための定数 HUGE_VAL が，<math.h>ヘッダ・ファイルに定義されています。

Return value

n	$2^{**}x$ 。
HUGE_VAL	オーバーフローが発生したことを示します。errnoには ERANGE が設定されます。
1	xが +0 か-0 であったことを示します。errnoには ERANGE が設定されます。
0	xが -Inf (負の無限大) であったか，アンダフローが発生したことを示します。errnoには ERANGE が設定されます。
x	xが +Inf (正の無限大) であったことを示します。errnoには ERANGE が設定されます。

`exp2` (*Alpha*, *I64*)

NaN

xがNaNであったことを示します。errnoにはEDOMが設定されます。

引数の絶対値を返します。

Format

```
#include <math.h>
double fabs (double x);
float fabsf (float x); (Alpha, I64)
long double fabsl (long double x); (Alpha, I64)
```

引数

X
実数値。

戻り値

x	引数の絶対値。
----------	---------

fchmod

ファイルのアクセス許可を変更します。

Format

```
#include <stat.h>
int fchmod (int fildes, mode_t mode);
```

Argument

fildes

オープンされているファイルのファイル記述子。

mode

アクセス許可を指定するためのビット・パターン。

Description

fchmod関数は、次の点を除いて、chmod関数と同じです。つまり、fchmod関数では、対象となるファイルを、ファイル名ではなくファイル記述子 (fildes) で指定します。

Return value

0	モードが変更できたことを示します。
-1	変更に失敗したことを示します。

fchown

ファイルのオーナーとグループを変更します。

Format

```
#include <unistd.h>

int fchown (int fildes, uid_t owner, gid_t group);
```

Argument

fildes

オープンされているファイルの記述子。

owner

ファイルの新しいオーナーに対応するユーザ ID。

group

ファイルのグループに対応するグループ ID。

Description

fchown関数はchownと同じ機能を実行しますが、オーナーとグループを変更するファイルをファイル記述子fildesで指定する点異なります。

Return value

0

正常終了を示します。

-1

異常終了を示します。errnoが次のいずれかの値に設定されます。

次の場合、fchown関数は異常終了します。

- EBADF – fildes引数がオープンされているファイルの記述子ではありません。
- EPERM – 有効なユーザ ID がファイルのオーナーに対応しないか、またはプロセスに適切な特権がありません。
- EROFS – fildesで参照されるファイルは読み込み専用ファイル・システムに存在します。

次の場合、fchown関数は異常終了することがあります。

- EINVAL – オーナ ID またはグループ ID が実装でサポートされない値です。
- EIO— 物理的な I/O エラーが発生しました。
- EINTR – fchown関数が検出されたシグナルによって割り込まれました。

fclose

ファイル制御ブロックに関連付けられているバッファの内容を出力し、ファイル・ポインタに関連付けられているファイル制御ブロックおよびバッファを解放することにより、ファイルをクローズします。

Format

```
#include <stdio.h>

int fclose (FILE *file_ptr);
```

引数

`file_ptr`
クローズするファイルを指すポインタ。

Description

プログラムが正常終了すると、オープンされているすべてのファイルに対してfclose関数が自動的に呼び出されます。

fclose関数は、fflushを暗黙に呼び出すことにより、バッファに格納されているデータを書き込もうとします。

書き込みが失敗した場合 (たとえばディスクが満杯の場合や、ユーザのクォータを超えた場合)、fcloseは実行を続行します。OpenVMS チャネルをクローズし、バッファの割り当てを解除し、ファイル記述子 (または FILE ポインタ) に関連付けられているメモリを解放します。バッファに格納されているデータは消失し、ファイル記述子 (または FILE ポインタ) はファイルを参照しなくなります。

バッファに格納されているデータを書き込むときに発生したエラーから回復する必要がある場合は、プログラムでfcloseを呼び出す前に、fsync (またはfflush) を明示的に呼び出さなければなりません。

fclose

Return value

0	正常終了を示します。
EOF	ファイル制御ブロックがオープンされているファイルに関連付けられていないことを示します。

fcntl

オープンされているファイルに対して制御操作を実行します。

Format

```
#include <sys/types.h>
#include <unistd.h>
#include <fcntl.h>

int fcntl (int file_desc, int request [, int arg]);
int fcntl (int file_desc, int request [, struct flock *arg]);
```

Argument

file_desc

正常終了したopen, fcntl, pipe関数から返されたオープン・ファイル記述子。

request

実行する操作。

arg

request引数の値に応じて異なる変数。

requestが F_DUPFD, F_SETFD, または F_SETFL の場合は, argとしてintを指定します。

requestが F_GETFD と F_GETFL の場合は, argを指定しません。

requestが F_GETLK, F_SETLK, または F_SETLKW の場合は, argとしてflock構造体へのポインタを指定します。

Description

fcntl関数は, file_desc引数によって指定される, オープンされているファイルに対して制御操作を実行します。

request引数の値は, ヘッダ・ファイル<fcntl.h>に定義されています。次の値が定義されています。

F_DUPFD	<p>int型の整数として受け付けられる 3 番目の引数 (arg) に等しいか、またはそれ以上で、使用可能な (つまりまだオープンされていない) ファイル記述子のうち、最小の新しいファイル記述子を返します。</p> <p>新しいファイル記述子は元のファイル記述子 (file_desc) と同じファイルを参照します。新しいファイル記述子に関連付けられるFD_CLOEXECフラグは、exec関数のいずれかの呼び出しでファイルをオープンしたままにしておくためにクリアされます。</p> <p>次の 2 つの呼び出しは同じです。</p> <pre>fid = dup(file_desc); fid = fcntl(file_desc, F_DUPFD, 0);</pre> <p>次の呼び出しがあるとします。</p> <pre>fid = dup2(file_desc, arg);</pre> <p>これは、次の呼び出しに類似しています (ただし、同じではありません)。</p> <pre>close(arg); fid = fcntl(file_desc, F_DUPFD, arg);</pre>
F_GETFD	<p>ファイル記述子file_descに関連付けられている close-on-exec フラグの値を取得します。ファイル記述子フラグは、1 つのファイル記述子に関連付けられ、同じファイルを参照する他のファイル記述子には影響を与えません。arg引数は指定しないでください。</p>
F_SETFD	<p>file_descに関連付けられている close-on-exec フラグを 3 番目の引数の値に設定します。この引数はint型です。</p> <p>3 番目の引数が 0 の場合は、ファイルはexec関数の呼び出しでオープンされたままになります。つまり、exec関数から生成された子プロセスは、親からこのファイル記述子を継承します。</p> <p>3 番目の引数がFD_CLOEXECの場合は、次のexec関数の実行が正常終了したときに、ファイルはクローズされます。つまり、exec関数で生成された子プロセスは、親からこのファイル記述子を継承しません。</p>
F_GETFL	<p>file_descに関連付けられたファイル記述に対し、<fcntl.h>で定義されているファイル状態フラグとファイル・アクセス・モードを取得します。ファイル・アクセス・モードは、<fcntl.h>で定義されているマスク O_ACCMODE を使用して、戻り値から取り出すことができます。ファイル状態フラグおよびファイル・アクセス・モードは、ファイル記述に関連付けられており、異なるオープン・ファイル記述で同じファイルを指す、ほかのファイル記述子には影響しません。</p>
F_SETFL	<p>file_descに関連付けられたファイル記述に対し、int型の第 3 引数argの対応するビットから、<fcntl.h>で定義されているファイル状態フラグを設定します。argで設定されているビットのうち、<fcntl.h>で定義されているファイル・アクセス・モードやファイル作成フラグに対応するビットは、無視されます。ここで説明した以外のargのビットをアプリケーションが変更すると、結果は不定となります。</p>

注意:認識される状態ビットは `O_APPEND` だけです。 `O_APPEND` のサポートは標準に準拠していません。 `X/Open` 標準では、「ファイル状態フラグとファイル・アクセス・モードは、ファイル記述に関連付けられており、異なるオープン・ファイル記述で同じファイルを指す、ほかのファイル記述子には影響しない」となっています。しかし、追加ビットは FCB に格納されるため、同じ FCB を使用するすべてのファイル記述子は、同じ追加フラグを使用することになり、`fcntl(F_SETFL)` を使用してこれを設定すると、FCB を共用しているすべてのファイル、すなわち、同じファイル記述子から複製されたすべてのファイルに影響があります。

レコード・ロックの要求

<code>F_GETLK</code>	argパラメータ (struct flock型を指すポインタ) で指定したロック記述構造体に対して、それをブロックしている最初のロックを取得します。そのようなロックがあった場合は、その情報によって、fcntl関数に渡されたflock構造体の中の情報が上書きされます。このロックの生成を妨げているロックがなかった場合は、ロック・タイプの設定が <code>F_UNLCK</code> に変更されることを除いて、その構造体がそのまま返されます。
<code>F_SETLK</code>	argパラメータ (struct flock型を指すポインタ) で指定したロック記述構造体の内容に従って、ファイル・セグメントのロックを設定または削除します。 <code>F_SETLK</code> は、共用ロックを設定する場合 (<code>F_RDLCK</code>)、排他ロックを設定する場合 (<code>F_WRLCK</code>)、または、それらのいずれかを削除する場合 (<code>F_UNLCK</code>) に使用します。共用 (読み取り) ロックまたは排他 (書き込み) ロックが設定できないと、fcntl関数はすぐに終了して、値 <code>-1</code> を返します。 flock構造体の <code>l_len</code> がゼロになっていなくて、しかも要求セグメントの最後のバイトのオフセットが <code>off_t</code> 型オブジェクトの最大値になっているようなアンロック要求 (<code>F_UNLCK</code>) は、プロセスが、 <code>l_len = 0</code> でしかも要求セグメントの最後のバイトを含むようなロックをすでに行っていると、その要求セグメントを先頭からアンロックする要求として扱われます。また、これ以外の条件でアンロックを要求すると (<code>F_UNLCK</code>)、要求したファイル全体を対象にして、アンロックが試みられます。
<code>F_SETLKW</code>	共用ロックまたは排他ロックが他のロックによってブロックされていればそのロックがアンブロックされるまで待つということ以外は、 <code>F_SETLK</code> と同じです。fcntlの処理で、ある領域がアンロックされるのを待っているときにシグナルを受信すると、関数が中断され、 <code>-1</code> が返されるとともに、 <code>errno</code> に <code>EINTR</code> が設定されます。

ファイルのロック

`C RTL` では、fcntl関数の `F_GETLK` コマンド、`F_SETLK` コマンド、および `F_SETLKW` コマンドで、`X/Open` で定義されている仕様と同じように、バイト単位のファイル・ロックをサポートしています。このバイト単位のファイル・ロックは、OpenVMS クラスタ間でもサポートされています。ただし、オフセットとして使用できる値は、32 ビットの符号なし整数に収まる範囲だけです。

ファイルのセグメントに設定されているロックが共用ロックである場合は、クラスタ上の他のプロセスから、そのセグメントやその一部に対して、共用ロックを設定することができます。しかし、たとえその範囲が一部であっても、共用ロックが設定されている保護領域に他のプロセスから排他ロックを設定することはいっさいできません。また、共用ロックの要求は、ファイル記述子が読み取り用にオープンされていないと失敗します。

設定されているロックのタイプが排他ロックである場合は、たとえその範囲が一部であっても、クラスタ上の他のプロセスからその保護領域に共用ロックまたは排他ロックを設定することはできません。また、排他ロックの要求は、ファイル記述子が書き込み用にオープンされていないと失敗します。

`flock`構造体には、対象となるファイル・セグメントの、型 (`l_type`)、開始オフセット (`l_whence`)、相対オフセット (`l_start`)、サイズ (`l_len`)、およびプロセス ID (`l_pid`) を設定します。

`l_whence`の値として設定できる値は、`SEEK_SET` (相対オフセット `l_start`のバイトをファイルの先頭から計算する)、`SEEK_CUR` (現在の位置から計算する)、および `SEEK_END` (ファイルの末尾から計算する) です。`l_len`の値は、ロック対象領域のバイト数です (このロック対象領域は連続している必要があります)。`l_len`の値には、負の値を設定することもできます (ただし、`off_t`の定義でそれが許されている場合だけ)。`l_pid`フィールドは、そのロックをブロックしているプロセスの ID を返してもらうために、`F_GETLK` でだけ使用されます。要求が `F_GETLK` の場合は、その処理に成功すると、`l_whence`に `SEEK_SET` が設定されます。

`l_len`に正の値を設定すると、`l_start`から始まって `l_start + l_len - 1` で終わる領域が対象になります。`l_len`に負の値を設定すると、`l_start + l_len`から始まって `l_start - 1` で終わる領域が対象になります。ロックの対象となる領域の開始位置と終了位置は、現在のファイルの末尾より後ろにあってもかまいません。つまり、現在のファイルの末尾より後ろにある領域もロックできます。しかし、対象領域の開始位置または終了位置をファイルの先頭に対して負の方向に設定することはできません。`l_len` に 0 を設定すると、ロックの対象領域つまり終了位置は、そのファイルのオフセットして可能な最大値まで常に拡張されます。またこれに加えて、`l_start`にも 0 を設定して、さらに `l_whence`に `SEEK_SET` を設定すると、ファイル全体がロックの対象になります。

ロックされているセグメントの中間部分をアンロックしたり、そのロック・タイプを変更したりすると、両端にそれより小さいセグメントが残ってしまいます。また呼び出しプロセスがすでにロックしているセグメントを再度ロックすると、設定されていたロックのタイプが削除されて、新しいロックのタイプが有効になります。

あるファイルを特定のプロセスがロックしていたときに、そのプロセスがそのファイルのファイル記述子をクローズしたり、そのファイル記述子を保持しているプロセスが終了したりすると、それらのプロセスに関連したそのファイルのロックがすべて削除されます。ロックが、子プロセスに継承されることはありません。

`request`引数に `F_SETLKW` を設定した場合は、そのロック要求が他のプロセスの掛けたロックでブロックされると、そのロックの解除を待つてスリープすることになり、デッドロックが発生して、アプリケーションがハングすることがあります。

Return value

n

正常終了時に返される値は、次に示すように、request 引数の値に応じて異なります。

- `F_DUPFD`— 新しいファイル記述子を返します。
- `F_GETFD`— `FD_CLOEXEC` または 0 を返します。
- `F_SETFD`, `F_GETLK`, `F_SETLK`, `F_UNLCK`— 1 以外の値を返します。

エラーが発生したことを示します。errnoは以下のいずれかの値に設定されます。

- EACCES – (1) request 引数が F_SETLK であったか、(2) ロックのタイプ (l_type) が共用 (F_RDLCK) または排他 (F_WRLCK) で、対象となるファイルのセグメントが他のプロセスからの排他ロックですでにロックされているか、(3) ロックのタイプが排他 (F_WRLCK) で、対象となるファイルのセグメントが他のプロセスからの共用ロックまたは排他ロックですでにロックされています。

- EBADF—file_desc引数が有効なオープンされているファイル記述子ではなく、arg引数が負の値であるか、またはプロセス単位のリミットに等しいか、それ以上の値です。

requestパラメータの値が F_SETLK または F_SETLKW で、しかもロックのタイプ (l_type) が共用ロック (F_RDLCK) であったのにもかかわらず、file_descで指定されたファイル記述子が読み取り用にオープンされていないか、有効ではありませんでした。

ロックのタイプ (l_type) が排他ロック (F_WRLCK) であったのにもかかわらず、file_descで指定されたファイル記述子が書き込み用にオープンされていないか、有効ではありませんでした。

- EFAULT—arg引数が不正なアドレスです。
- EINVAL—request引数が F_DUPFD であり、arg引数が負の値であるか、または OPEN_MAX に等しいか、それより大きな値です。

OPEN_MAX の値またはプロセス単位のソフト記述子リミットが確認されます。

不正な値がrequest引数に対して指定されました。

引数requestの値が F_GETLK、F_SETLK、または F_SETLKW であるにもかかわらず、argの指しているデータが無効か、file_descが、ロックをサポートしていないファイルを参照していました。

- EMFILE—request引数が F_DUPFD であり、呼び出しプロセスで現在多すぎる個数または OPEN_MAX 個のファイル記述子がオープンされています。または、arg引数以上のファイル記述子を使用することはできません。

OPEN_MAX の値またはプロセス単位のソフト記述子リミットが確認されます。

- EOVERFLOW — 返すべき値の中に、正しく表現できないものがありました。
引数requestの値は F_GETLK, F_SETLK, または F_SETLKW ですが、対象となるセグメントにあるバイトの最小オフセットまたは最大オフセット (l_lenがゼロでない場合) が, off_t型のオブジェクトで正しく表現できませんでした。
- EINTR — 引数requestの値は F_SETLKW ですが、シグナルによって関数が中断されました。
- ENOLCK — 引数requestの値は F_SETLK または F_SETLKW ですが、ロックのまたはアンロックの要求を満たそうとすると、構成可能なシステム・リミットの値 NLOCK_RECORD を超えてしまいます。
- ENOMEM — システムは要求されたファイル記述子に対してメモリを割り当てることができませんでした。

fcvt

引数をヌル区切りの ASCII 数字列に変換し、数字列のアドレスを返します。数字列は HP C RTL で作成されたスレッド固有の記憶位置に格納されます。

Format

```
#include <stdlib.h>

char *fcvt (double value, int ndigits, int *decpt, int *sign);
```

Argument

value

ヌル区切りの ASCII 数字列に変換される double 型のオブジェクト。

ndigits

変換後の数字列で使用する小数点以下の ASCII 桁数。

decpt

返される数字列の 1 文字目を基準にした小数点の位置。返される数字列には実際の小数点は含まれません。int の値が負の値の場合は、小数点の位置は、返された数字列の左側に decpt 個のスペースを付加した位置になります (スペースには 0 が埋められます)。値が 0 の場合は、小数点の位置は返された数字列の最初の桁のすぐ左に設定されます。

sign

value 引数が正の値であるか、負の値であることを示す整数値。value が負の場合は、fcvt 関数は sign によって指定されるアドレスに 0 以外の値を格納します。負でない場合は、この関数は sign によって指定されるアドレスに 0 を代入します。

Description

fcvt 関数は、value をヌル区切り数字列に変換し、その数字列を指すポインタを返します。返される下位桁は、C の F 形式で ndigits 桁を出力するために、適切な桁に丸められます。decpt 引数には、数字列の最初の数字を基準にした小数点の位置が代入されます。

C の F 形式では、ndigits は小数点以下の桁数です。非常に大きい数値の場合、小数点の左側の数字列が非常に長くなり、小数点以下の桁数は ndigit になります。大きい数値の場合、E 形式が使用されるように、gcvt または ecvt 関数を使用する方が適切です。

fcvt関数を繰り返し呼び出すと、既存の数字列は上書きされます。

ecvt, fcvt, gcvt関数は、浮動小数点演算に関して IEEE 標準で指定されている次の特殊な値を表します。

値	表現
クワイエット NaN	NaNQ
シグナリング NaN	NaNS
+ 無限大	無限大
−無限大	−無限大

これらの各値に割り当てられる符号は、sign引数に格納されます。IEEE 浮動小数点表現では、0 (ゼロ) という値は正の場合も負の場合もあり、どちらであるかはsign引数によって設定されます。

gcvtとecvtも参照してください。

戻り値

X 変換後の数字列を指すポインタ。

fdim (*Alpha, I64*)

引数の正の差分を調べます。

Format

```
#include <math.h>

double fdim (double x, double y);
float fdimf (float x, float y);
long double fdiml (long double x, long double y);
```

引数

x
実数値。

y
実数値。

Description

fdim関数は、指定した引数について、その正の差分を調べます。xがyより大きい場合は、 $x - y$ を返します。xがy以下である場合は、 $+0$ を返します。

Return value

n	正の差分がとれたことを示します。nは、その正の差分値です。
HUGE_VAL	$x - y$ の値は正ですが、オーバーフローしました。errnoには ERANGE が設定されます。
0	$x - y$ の値は正ですが、アンダフローしました。errnoには ERANGE が設定されます。
NaN	xまたはyが NaN です。errnoには EDOM が設定されません。

fdopen

ファイル・ポインタを, `open`, `creat`, `dup`, `dup2`, `pipe`関数から返されたファイル記述子に関連付けます。

Format

```
#include <stdio.h>

FILE *fdopen (int file_desc, char *a_mode);
```

Argument

`file_desc`

`open`, `creat`, `dup`, `dup2`, `pipe`から返されたファイル記述子。

`a_mode`

アクセス・モード指示子。詳細については, `fopen`関数を参照してください。指定するアクセス・モードは, ファイルをオープンするために使用したモードと一致しなければなりません。バイナリ/テキスト・アクセス・モード (`fdopen`の "b"モードおよび `creat`や `open`の "ctx=bin"オプション) も含まれます。

Description

`fdopen`関数を使用すると, UNIX I/O 関数のいずれかでオープンされているファイルに, 標準 I/O 関数を使用してアクセスすることができます。通常, ファイルにアクセスするには, そのファイルをオープンした方法に応じて, ファイル記述子またはファイル・ポインタを使用しますが, 両方を使用してアクセスすることはできません。詳細については, 『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1 章 および第 2 章を参照してください。

Return value

ポインタ

操作が正常終了したことを示します。

NULL

エラーが発生したことを示します。

feof

ファイルがファイルの終端 (EOF) に到達しているかどうかを判定します。

Format

```
#include <stdio.h>
int feof (FILE *file_ptr);
```

引数

file_ptr
ファイル・ポインタ。

Return value

0 以外の整数

ファイルの終端 (EOF) に到達したことを示します。

0

ファイルの終端 (EOF) に到達していないことを示します。

feof_unlocked (Alpha, I64)

feof関数と同様ですが、flockfileとfunlockfileで保護された範囲だけで使用します。

Format

```
#include <stdio.h>

int feof_unlocked (FILE *file_ptr);
```

引数

file_ptr
ファイル・ポインタ。

Description

リエントラント版であるfeof関数は、複数スレッドからの同時呼び出しに対してロックされます。その結果、ストリームの一貫性を保証するためのオーバーヘッドが生じます。アンロック版であるfeof_unlockedを使用すると、このオーバーヘッドを避けることができます。feof_unlocked関数は、feof関数と機能的に同じですが、スレッド・セーフな方法で実装する必要がない点が異なります。feof_unlocked関数は、flockfile関数とfunlockfile関数を対で使用して保護された範囲内でだけ、安全に使用することができます。呼び出し元は、feof_unlockedを使用する前に、ストリームを確実にロックする必要があります。

flockfile、ftrylockfile、およびfunlockfileも参照してください。

Return value

0 以外の整数	ファイルの終端に到達したことを示します。
0	ファイルの終端に到達していないことを示します。

ferror

ファイルの読み込みまたは書き込みでエラーが発生した場合，0 以外の整数を返します。

Format

```
#include <stdio.h>

int ferror (FILE *file_ptr);
```

引数

file_ptr
ファイル・ポインタ。

Description

ferrorを呼び出すと，ファイルがクローズされるまで，またはclearerrが呼び出されるまで，0 以外の整数が返されます。

Return value

0	正常終了を示します。
0 以外の整数	エラーが発生したことを示します。

ferror_unlocked (Alpha, I64)

ferror関数と同様ですが、flockfileとfunlockfileで保護された範囲内だけで使用します。

Format

```
#include <stdio.h>

int ferror_unlocked (FILE *file_ptr);
```

引数

file_ptr
ファイル・ポインタ。

Description

リエントラント版であるferror関数は、複数スレッドからの同時呼び出しに対してロックされます。その結果、ストリームの一貫性を保証するためのオーバーヘッドが生じます。アンロック版であるferror_unlockedを使用すると、このオーバーヘッドを避けることができます。ferror_unlocked関数は、ferror関数と機能的に同じですが、スレッド・セーフな方法で実装する必要がない点が異なります。ferror_unlocked関数は、flockfile関数とfunlockfile関数を対で使用して保護された範囲内でだけ、安全に使用することができます。呼び出し元は、ferror_unlockedを使用する前に、ストリームを確実にロックする必要があります。

flockfile、ftrylockfile、およびfunlockfileも参照してください。

Return value

0	成功を示します。
0 以外の整数	エラーが発生したことを示します。

fflush

バッファに格納されている情報を指定されたファイルに書き込みます。

Format

```
#include <stdio.h>

int fflush (FILE *file_ptr);
```

引数

file_ptr

ファイル・ポインタ。この引数が NULL ポインタの場合は、現在オープンされているすべてのファイルに関連付けられているすべてのバッファの内容が書き込まれます。

Description

出力ファイルは通常、端末に出力される場合にだけバッファに格納されます。ただし、stderrの場合は例外で、デフォルトではバッファに格納されません。

fflush関数は、HP C RTL バッファの内容を書き込みます。しかし、RMS には独自のバッファがあります。fflush関数を呼び出しても、ファイルがディスクに書き込まれるという保証はありません (バッファをディスクに書き込む方法については、fsyncの説明を参照してください)。

file_ptrによって示されるファイルがレコード・モードでオープンされていて、バッファにまだ書き込まれていないデータがある場合は、fflushは常にレコードを生成します。

Return value

0

操作が正常終了したことを示します。

EOF

バッファに格納されているデータをファイルに書き込むことができないか、ファイル制御ブロックが出力ファイルに関連付けられていないことを示します。

ffs

文字列から、セットされている最初のビットのインデックスを検索します。

Format

```
#include <strings.h>
int ffs (int iteger);
```

引数

integer
最初のビットがセットされているかどうか確認する整数。

Description

ffs関数は、セットされている最初のビットを検索し (最下位ビットから順に確認します)、そのビットのインデックスを返します。ビットには1 (最下位ビット) から順に番号が付けられます。

Return value

x	セットされている最初のビットのインデックス。
0	indexが 0 であることを示します。

fgetc

指定されたファイルから次の文字を返します。

Format

```
#include <stdio.h>
int fgetc (FILE *file_ptr);
```

引数

`file_ptr`
アクセスするファイルを指すポインタ。

Description

`fgetc`関数は、指定されたファイルから次の文字を取り出して返します。

`__UNIX_PUTC` マクロを定義してコンパイルすれば、この関数の高速インライン版を使用して、最適化できます。

`fgetc_unlocked`関数と`getc`マクロも参照してください。

Return value

<code>x</code>	返された文字。
<code>EOF</code>	ファイルの終端 (EOF) またはエラーを示します。

fgetc_unlocked (Alpha, I64)

fgetc関数と同様ですが、flockfileとfunlockfileで保護された範囲内だけで使用します。

Format

```
#include <stdio.h>

int fgetc_unlocked (FILE *file_ptr);
```

引数

file_ptr
ファイル・ポインタ。

Description

リエントラント版であるfgetc関数は、複数スレッドからの同時呼び出しに対してロックされます。その結果、ストリームの一貫性を保証するためのオーバーヘッドが生じます。アンロック版であるfgetc_unlockedを使用すると、このオーバーヘッドを避けることができます。fgetc_unlocked関数はfgetc関数と機能的に同じですが、fgetc_unlockedはflockfile関数とfunlockfile関数を対で使用して保護された範囲内でだけ安全に使用することができる点で異なります。呼び出し元は、fgetc_unlockedを使用する前に、ストリームを確実にロックする必要があります。

__UNIX_PUTC マクロを定義してコンパイルすれば、この関数の高速インライン版を使用して、最適化できます。

getc_unlocked, flockfile, ftrylockfile, および funlockfileも参照してください。

Return value

n	返された文字。
EOF	ファイルの終端 (EOF) またはエラーを示します。

fgetname

ファイル・ポインタに関連付けられているファイル指定を返します。

Format

```
#include <stdio.h>

char *fgetname (FILE *file_ptr, char *buffer, ... );
```

関数バリエーション

fgetname関数には、_fgetname32および_fgetname64という名前のバリエーションがあり、それぞれ 32 ビット・ポインタ・サイズおよび 64 ビット・ポインタ・サイズで使用されます。ポインタ・サイズ固有の関数の使い方の詳細については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

Argument

file_ptr

ファイル・ポインタ。

buffer

ファイル指定を格納できる十分な大きさの文字列を指すポインタ。

...

省略可能な追加引数であり、1 または 0 に指定できます。1 を指定すると、fgetname関数は OpenVMS 形式でファイル指定を返します。0 を指定すると、UNIX 形式でファイル指定を返します。この引数を指定しないと、現在のコマンド言語インタプリタに従ってファイル名を返します。UNIX 形式のファイル指定の詳細については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.4.3 項を参照してください。

Description

fgetname関数は、バッファ内の指定されたアドレスにファイル指定を格納します。バッファは完全に修飾されたファイル指定を格納できるだけの十分な大きさの配列でなければなりません (最大長は 256 文字)。

Return value

n	バッファのアドレス。
0	エラーを示します。

制限事項

fgetname関数はHP C RTL 固有の関数であり、移植できません。

fgetpos

指定されたファイルの現在のファイルの位置を格納します。

Format

```
#include <stdio.h>
int fgetpos (FILE *stream, fpos_t *pos);
```

Argument

stream

ファイル・ポインタ。

pos

実装で定義されている構造体を指すポインタ。fgetpos関数はこの構造体に、この後のfsetposの呼び出しで利用できる情報を格納します。

Description

fgetpos関数は、streamによって示されるストリームのファイル位置指示子の現在の値を、posによって示されるオブジェクトに格納します。

Return value

0	正常終了を示します。
-1	エラーが発生したことを示します。

例

```
#include <stdio.h>
#include <stdlib.h>

main()
{
    FILE *fp;
    int stat,
        i;
    int character;
    char ch,
        c_ptr[130],
        d_ptr[130];
    fpos_t posit;

    /* Open a file for writing. */
    if ((fp = fopen("file.dat", "w+")) == NULL) {
        perror("open");
        exit(1);
    }

    /* Get the beginning position in the file. */
    if (fgetpos(fp, &posit) != 0)
        perror("fgetpos");

    /* Write some data to the file. */
    if (fprintf(fp, "this is a test\n") == 0) {
        perror("fprintf");
        exit(1);
    }

    /* Set the file position back to the beginning. */
    if (fsetpos(fp, &posit) != 0)
        perror("fsetpos");

    fgets(c_ptr, 130, fp);
    puts(c_ptr);          /* Should be "this is a test." */

    /* Close the file. */
    if (fclose(fp) != 0) {
        perror("close");
        exit(1);
    }
}
```

fgets

指定されたファイルから 1 行を読み込みます。指定された最大文字数、または改行文字まで (どちらか最初に検出された方) を読み込みます。文字列はstrに格納されます。

Format

```
#include <stdio.h>

char *fgets (char *str, int maxchar, FILE *file_ptr);
```

関数バリエーション

fgets関数には、_fgets32および_fgets64という名前のバリエーションがあり、それぞれ 32 ビット・ポインタ・サイズおよび 64 ビット・ポインタ・サイズで使用されます。ポインタ・サイズ固有の関数の使い方の詳細については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

Argument

str
ファイルから読み込まれた情報を格納できるだけの十分な大きさの文字列を指すポインタ。

maxchar
読み込む最大文字数。

file_ptr
ファイル・ポインタ。

Description

fgets関数は、行をヌル文字(\\0)で区切ります。gets関数と異なり、fgets関数は、読み込んだ文字数がmaxcharに指定された文字数より少ない場合、入力行を区切るために改行文字をユーザ・バッファに追加します。

file_ptrによって示されるファイルがレコード・モードでオープンされている場合は、fgetsはレコードの終端を改行文字と同じ方法で取り扱います。したがって、改行文字までを読み込むか、またはレコードの終端までを読み込みます。

Return value

x	strを指すポインタ。
NULL	ファイルの終端 (EOF) またはエラーを示します。読み込みエラーが発生した場合は、strの内容は未定義です。

例

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

main()
{
    FILE *fp;
    char c_ptr[130];

    /* Create a dummy data file */
    if ((fp = fopen("file.dat", "w+")) == NULL) {
        perror("open");
        exit(1);
    }

    fprintf(fp, "this is a test\n") ;
    fclose(fp) ;

    /* Open a file with some data -"this is a test" */
    if ((fp = fopen("file.dat", "r+")) == NULL) {
        perror("open error") ;
        exit(1);
    }

    fgets(c_ptr, 130, fp);
    puts(c_ptr);          /* Display what fgets got. */
    fclose(fp);

    delete("file.dat") ;
}
```

fgetwc

指定されたファイルから次の文字を読み込み、その文字をワイド文字コードに変換します。

Format

```
#include <wchar.h>

wint_t fgetwc (FILE *file_ptr);
```

引数

`file_ptr`
アクセスするファイルを指すポインタ。

Description

正常終了すると、`fgetwc`関数は、`file_ptr`によって示されるファイルから読み込んだ後、`wint_t`型に変換したワイド文字コードを返します。ファイルがファイルの終端 (EOF) にある場合は、EOF 指示子が設定され、WEOF が返されます。I/O 読み込みエラーが発生した場合は、エラー指示子が設定され、WEOF が返されます。

アプリケーションで`ferror`または`feof`を使用すると、エラー条件と EOF 条件を区別できます。

Return value

<code>x</code>	読み込んだ文字のワイド文字コード。
<code>WEOF</code>	ファイルの終端 (EOF) またはエラーを示します。読み込みエラーが発生した場合は、 <code>errno</code> は次のいずれかの値に設定されます。 <ul style="list-style-type: none">• EALREADY— 同じファイルに対して操作がすでに実行されています。• EBADF— ファイル記述子が不正です。• EILSEQ— 不正な文字が検出されました。

fgetws

指定されたファイルから 1 行のワイド文字を読み込みます。

Format

```
#include <wchar.h>

wchar_t *fgetws (wchar_t *wstr, int maxchar, FILE *file_ptr);
```

関数バリエーション

fgetws関数には、_fgetws32および_fgetws64という名前のバリエーションがあり、それぞれ 32 ビット・ポインタ・サイズおよび 64 ビット・ポインタ・サイズで使用されます。ポインタ・サイズ固有の関数の使い方の詳細については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

Argument

wstr

ファイルから読み込んだ情報を格納できるだけの十分な大きさのワイド文字の文字列を指すポインタ。

maxchar

読み込むワイド文字の最大文字数。

file_ptr

ファイル・ポインタ。

Description

fgetws関数は、指定されたファイルからワイド文字を読み込み、wstrによって示される配列に格納します。この関数は、maxchar-1文字まで読み込むか、改行文字が読み込まれ、変換され、wstrに転送されるまで読み込むか、またはファイルの終端 (EOF) 条件が検出されるまで読み込みます。行末はヌル・ワイド文字で区切られます。読み込んだ文字数がmaxchar文字より少ない場合、fgetwsは入力行の最後に改行文字を付けてユーザ・バッファに格納します。

Return value

x	wstrを指すポインタ。
NULL	ファイルの終端 (EOF) またはエラーを示します。読み込みエラーが発生した場合、wstrの内容は未定義です。読み込みエラーが発生した場合、errnoが設定されます。設定される可能性のあるerrnoの値の一覧については、fgetwcを参照してください。

例

```
#include <stdlib.h>
#include <stdio.h>
#include <locale.h>
#include <wchar.h>

main()
{
    wchar_t wstr[80],
            *ret;
    FILE *fp;

    /* Create a dummy data file */
    if ((fp = fopen("file.dat", "w+")) == NULL) {
        perror("open");
        exit(1);
    }

    fprintf(fp, "this is a test\n") ;
    fclose(fp) ;

    /* Open a test file containing : "this is a test" */
    if ((fp = fopen("file.dat", "r")) == (FILE *) NULL) {
        perror("File open error");
        exit(EXIT_FAILURE);
    }

    ret = fgetws(wstr, 80, fp);
    if (ret == (wchar_t *) NULL) {
        perror("fgetws failure");
        exit(EXIT_FAILURE);
    }

    fputws(wstr, stdout);
    fclose(fp);
    delete("file.dat");
}
```

fileno

指定されたファイル・ポインタに関連付けられているファイル記述子を返します。

Format

```
#include <stdio.h>
int fileno (FILE *file_ptr);
```

引数

file_ptr
ファイル・ポインタ。

Description

V5.2 またはそれ以前のバージョンの C コンパイラを使用する場合は、filenoマクロの定義を解除します。

```
#if defined(fileno)
#undef fileno
#endif
```

Return value

x	整数のファイル記述子。
−1	エラーを示します。

`finite` (*Alpha, I64*)

`finite` (*Alpha, I64*)

引数が有限の数値の場合は整数値 1 (TRUE) を返し、それ以外の場合は 0 (FALSE) を返します。

Format

```
#include <math.h>

int finite (double x);
int finitf (float x);
int double finitel (long double x);
```

引数

`x`
実数値。

Description

−無限大 < `x` < +無限大の場合は、`finite`関数は 1 を返します。 `| x |` =無限大または`x`が NaN の場合は、0 を返します。

flockfile (Alpha, I64)

stdioストリームをロックします。

Format

```
#include <stdio.h>

void flockfile (FILE *file_ptr);
```

引数

file_ptr
ファイル・ポインタ。

Description

flockfile関数は、stdioストリームをロックし、複数の入出力操作にわたって、スレッドがそのストリームを排他的に使用できるようになります。たとえばあるスレッドのprintfの出力と、printfを使おうとしているほかのスレッドの出力が混じらないようにしたい場合は、flockfile関数を使用します。

引数のファイル・ポインタは、正しいことが前提です。flockfileは、ファイル・ポインタが無効でもロックを実行します。また、funlockfile関数は、呼び出し元スレッドが引数のファイル・ポインタのロックを所有していない場合でも、エラーになりません。

対応するflockfileとfunlockfileの呼び出しは、ネストさせることができます。ストリームを再帰的にロックすると、対応する最後のfunlockfileを呼び出すまでは、ストリームはロックされたままになります。

すべてのC RTL ファイル・ポインタ入出力関数は、flockfileおよびfunlockfileを呼び出したかのように、ファイル・ポインタをロックします。

ftrylockfileおよびfunlockfileも参照してください。

floor

引数に等しいか、それより小さい最大の整数を返します。

Format

```
#include <math.h>

double floor (double x);
float floorf (float x); (Alpha, I64)
long double floorl (long double x); (Alpha, I64)
```

引数

x
実数値。

戻り値

n
引数に等しいか、それより小さい最大の整数。

fma (Alpha, I64)

$(x * y) + z$ を 1 つの 3 項演算として計算した後、その結果を丸めます。

Format

```
#include <math.h>

double fma (double x, double y, double z);
float fmaf (float x, float y, float z);
long double fmal (long double x, long double y, long double z);
```

引数

x
実数値。

y
実数値。

z
実数値。

Description

fma関数は、 $(x * y) + z$ を 1 つの 3 項演算として計算した後、その結果を丸めます。計算は値の精度が無限大であるかのようにして行い、その結果を、FLT_ROUNDS の値で指定されている丸めモードに従って 1 度で丸めます。

Return value

n	処理に成功したことを示します。n は、 $(x * y) + z$ の結果を 1 つの 3 項演算として丸めた値です。
NaN	xまたはyが NaN です。errnoには EDOM が設定されます。

fmax (Alpha, I64)

引数の中から大きい方を最大値として返します。

Format

```
#include <math.h>

double fmax (double x, double y);
float fmaxf (float x, float y);
long double fmaxl (long double x, long double y);
```

引数

x
実数値。

y
実数値。

Description

fmax関数は、どちらの引数が高いかを調べます。引数として NaN を指定すると、データがないものとして扱われます。一方の引数が NaN で、もう一方が数値であると、数値側の値が返されます。

Return value

n	処理に成功したことを示します。n は、大きかった方の数値です。一方の引数が NaN であった場合は、もう一方の引数が返されます。
NaN	xとyがどちらも NaN であったことを示します。

fmin (Alpha, I64)

引数の中から小さい方を最小値として返します。

Format

```
#include <math.h>

double fmin (double x, double y);
float fminf (float x, float y);
long double fminl (long double x, long double y);
```

引数

x
実数値。

y
実数値。

Description

fmin関数は、どちらの引数が小さいかを調べます。引数として NaN を指定すると、データがないものとして扱われます。一方の引数が NaN で、もう一方が数値であると、数値側の値が返されます。

Return value

n	処理に成功したことを示します。n は、小さかった方の数値です。一方の引数が NaN であった場合は、もう一方の引数が返されます。
NaN	xとyがどちらも NaN であったことを示します。

fmod

余り (浮動小数点数値) を計算します。

Format

```
#include <math.h>

double fmod (double x, double y);
float fmodf (float x, float y); (Alpha, I64)
long double fmodl (long double x, long double y); (Alpha, I64)
```

Argument

x
実数値。

y
実数値。

Description

fmod関数は、最初の引数を 2 番目の引数で除算した余りを浮動小数点数値で返します。2 番目の引数が 0 の場合は、0 が返されます。

Return value

x	一部の整数 <i>i</i> に対して、 $x == i * y + f$ が成り立つ値 <i>f</i> 。符号は <i>x</i> と同じです。ただし、 <i>f</i> の大きさは <i>y</i> の大きさより小さい値です。
0	<i>y</i> が 0 であることを示します。

fopen

FILE 構造体のアドレスを返すことにより、ファイルをオープンします。

Format

```
#include <stdio.h>

FILE *fopen (const char *file_spec, const char *a_mode); (ANSI C)
FILE *fopen (const char *file_spec, const char *a_mode, ... ); (HP C Extension)
```

Argument

file_spec

有効なファイル指定を格納した文字列。

a_mode

アクセス・モード指示子。次のいずれかの文字列を使用します。

"r" , "w" , "a" , "r+" , "w+" , "rb" , "r+b" , "rb+" , "wb" , "w+b" ,
"wb+" , "ab" , "a+b" , "ab+" , "a+"。

これらのアクセス・モードを使用すると、次の結果になります。

- "r"は、読み込みのために既存のファイルをオープンします。
- "w"は、必要に応じて新しいファイルを作成し、書き込みのためにファイルをオープンします。ファイルがすでに存在する場合は、同じ名前でバージョン番号が1だけ大きい新しいファイルを作成します。
- "a"は、追加アクセスのためにファイルをオープンします。ファイルの位置は既存のファイルの終端 (EOF) に設定され、そこにデータが書き込まれます。ファイルが存在しない場合は、HP C RTL はファイルを作成します。

更新アクセス・モードでは、読み込みと書き込みの両方のためにファイルをオープンすることができます。既存のファイルに対してこのモードを使用する場合、"r+"と"a+"は、ファイル内での最初の位置設定のみが異なります。各モードは次のとおりです。

- "r+"は読み込み更新アクセスのために既存のファイルをオープンします。ファイルは読み込みのためにオープンされ、最初はファイルの先頭に位置設定されますが、書き込みも許可されます。
- "w+"は書き込み更新アクセスのために新しいファイルをオープンします。

- "a+"は追加更新アクセスのためにファイルをオープンします。ファイルはまず、ファイルの終端 (EOF) に (書き込みのために) 位置設定されます。ファイルが存在しない場合は、HP C RTL はファイルを作成します。
- "b"は、バイナリ・アクセス・モードを示します。この場合、キャリッジ制御情報の変換は行われません。

...

省略可能なファイル属性引数。ファイル属性引数は、creat関数で使用する引数と同じです。詳細については、creat関数を参照してください。

Description

ファイルがすでに存在する場合は、fopenで作成される新しいファイルは、既存のファイルから特定の属性を継承します。ただし、fopenの呼び出しに指定されている属性は継承しません。次の属性が継承されます。

レコード・フォーマット
最大レコード・サイズ
キャリッジ制御
ファイル保護

ファイル名にディレクトリを指定し、そのディレクトリがエラーを含む検索リストである場合は、HP C for OpenVMSシステムはファイル・オープン・エラーとして解釈します。

ファイル制御ブロックは、fclose関数を使用して解放するか、またはプログラムが正常終了するときにデフォルトで解放することができます。

Return value

x	ファイル・ポインタ。
NULL	<p>エラーを示します。定数 NULL は、<stdio.h>ヘッダ・ファイルに NULL ポインタの値として定義されています。この関数は、次のエラーを示すために NULL を返します。</p> <ul style="list-style-type: none"> • ファイル保護違反。 • 読み込みアクセスのために存在しないファイルをオープンしようとした。 • 指定されたファイルをオープンすることができません。

fp_class (Alpha, I64)

IEEE 浮動小数点数値のクラスを判断します。

Format

```
#include <math.h>
int fp_class (double x);
int fp_classf (float x);
int fp_classl (long double x);
```

引数

x
IEEE 浮動小数点数値。

Description

fp_class関数は、指定された IEEE 浮動小数点数値のクラスを判断し、<fp_class.h>ヘッダ・ファイルから定数を返します。シグナリング NaN (Not-a-Number) の場合でも、例外は発生しません。これらの関数は、バイナリ浮動小数点演算に関してIEEE 754-1985標準の付録で勧告されているclass(x)関数を実相します。<fp_class.h>内の定数は次の値クラスを参照します。

FP_SNAN	シグナリング NaN (Not-a-Number)
FP_QNAN	クワイエット NaN
FP_POS_INF	+ 無限大
FP_NEG_INF	-無限大
FP_POS_NORM	正の正規化
FP_NEG_NORM	負の正規化
FP_POS_DENORM	正の非正規化
FP_NEG_DENORM	負の非正規化
FP_POS_ZERO	+0.0 (正のゼロ)
FP_NEG_ZERO	-0.0 (負のゼロ)

戻り値

x <fp_class.h>ヘッダ・ファイルに定義されている定数。

fpathconf

ファイル実装属性を取得します。

Format

```
#include <unistd.h>

long int fpathconf (int filedес, int name);
```

Argument

filedes

オープンされているファイル記述子。

name

取得する構成属性。この属性をfiledes引数によって指定されるファイルに適用できない場合は、fpathconfはエラーを返します。

Description

fpathconf関数を使用すると、アプリケーションはfiledes引数によって指定されるファイルの基礎になるファイル・システムでサポートされる操作の属性を取得することができます。指定されたファイルの読み込みアクセス許可、書き込みアクセス許可、実行アクセス許可は必要ありませんが、ファイルまでのパス内のすべてのディレクトリを検索できなければなりません。

name引数のシンボル値は、次に示すように<unistd.h>ヘッダ・ファイルに定義されています。

_PC_LINK_MAX	ファイルへのリンクの最大数。filedes引数がディレクトリを参照する場合は、返される値はディレクトリ自体に適用されます。
_PC_MAX_CANON	標準的な入力行の最大バイト数。これは端末デバイスにのみ適用されます。
_PC_MAX_INPUT	入力キューで認められるタイプの数。これは端末デバイスにのみ適用されます。
_PC_NAME_MAX	ファイル名の最大バイト数 (区切り文字のヌルは含みません)。値の範囲は 13 ~ 255 です。これはディレクトリ・ファイルにのみ適用されます。返される値はディレクトリ内のファイル名に適用されます。

<code>_PC_PATH_MAX</code>	パス名の最大バイト数 (区切り文字のヌルは含みません)。値は 65,535 以下でなければなりません。これはディレクトリ・ファイルにのみ適用されます。指定したディレクトリがワーキング・ディレクトリの場合は、返される値は相対パス名の最大長です。
<code>_PC_PIPE_BUF</code>	自動的に書き込まれることが保証される最大バイト数。これは FIFO にのみ適用されます。返される値は参照されたオブジェクトに適用されます。path 引数がディレクトリを参照する場合は、返される値は、既存の FIFO、またはディレクトリ内で作成可能な FIFO に適用されます。
<code>_PC_CHOWN_RESTRICTED</code>	返される値は、既存のファイル (ディレクトリ以外) に適用されるか、またはディレクトリ内で作成可能なファイルに適用されます。これはディレクトリ・ファイルにのみ適用されます。
<code>_PC_NO_TRUNC</code>	NAME_MAX で指定される長さより長いコンポーネント名を指定することにより、エラーが発生する場合は、1 を返します。長いコンポーネント名が切り捨てられる場合は、0 (ゼロ) を返します。これはディレクトリ・ファイルにのみ適用されます。
<code>_PC_VDISABLE</code>	これは常に 0 (ゼロ) です。無効にする文字は定義されていません。これは端末デバイスにのみ適用されます。

Return value

<code>x</code>	name 引数に指定された構成属性の値。
<code>-1</code>	エラーを示します。errno は次のいずれかの値に設定されます。 <ul style="list-style-type: none"> • EINVAL— name 引数が不明の属性または適用できない属性を指定しています。 • EBADF— filesdes 引数が不正なファイル記述子です。

fprintf

指定したファイルに対して書式設定した出力を実行します。

Format

```
#include <stdio.h>

int fprintf (FILE *file_ptr, const char *format_spec, ... );
```

Argument

file_ptr

出力先のファイルを指すポインタ。

format_spec

書式指定を格納した文字列を指すポインタ。書式指定および変換文字の詳細については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 2 章を参照してください。

...

省略可能な式であり、これらの式の型は、書式指定に指定した変換指定に対応します。

変換指定を指定しない場合は、出力ソースを省略できます。変換指定を指定する場合は、関数呼び出しに変換指定と正確に同じ数の出力ソースを指定する必要があります。変換指定は出力ソースの型と一致しなければなりません。

変換指定は左から右への順に出力ソースに対応付けられます。出力ソースの数の方が多い場合は、超過するソースは無視されます。

例

次の例は変換指定を示しています。

```
#include <stdio.h>

main()
{
    int temp = 4, temp2 = 17;
    fprintf(stdout, "The answers are %d, and %d.", temp, temp2);
}
```

この例は、stdoutファイルに次の文字列を出力します。

```
The answers are 4, and 17.
```

書式指定と出力ソースの詳細については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 2 章を参照してください。

Return value

x

負の値

区切り文字のヌルを除き、書き込んだバイト数。

エラーを示します。errnoは次のいずれかに設定されます。

- EILSEQ— 不正な文字が検出されました。
- EINVAL— 引数の数が不足しています。
- ENOMEM— 変換のために使用できるメモリが不足しています。
- ERANGE— 浮動小数点演算オーバーフローが発生しました。
- EVMSERR— 変換不可能な OpenVMS エラー。vaxc\$errnoには OpenVMS エラー・コードが格納されます。これは、オーバーフローが発生したために、数値への変換が失敗したことを示します。

I/O サブシステムからエラーが返された場合、この関数はerrnoを次の値に設定します。

- EBADF— ファイル記述子が不正です。
- EIO— I/O エラー。
- ENOSPC— ファイルを格納しているデバイスに空き領域がありません。
- ENXIO— デバイスが存在しません。
- EPIPE— パイプが壊れています。
- ESPIPE— 追加のためにオープンされたファイルで不正なシークが行われました。
- EVMSERR— 変換不可能な OpenVMS エラー。vaxc\$errnoには OpenVMS エラー・コードが格納されます。これは、対応する C エラー・コードが定義されていない I/O エラーが発生したことを示します。

fputc

1 文字を指定のファイルに書き込みます。

Format

```
#include <stdio.h>
int fputc (int character, FILE *file_ptr);
```

Argument

character
int型のオブジェクト。

file_ptr
ファイル・ポインタ。

Description

fputc関数は 1 文字を指定されたファイルに書き込み、その文字を返します。

__UNIX_PUTC マクロを定義してコンパイルすれば、この関数の高速インライン版を使用して、最適化できます。

fputc_unlocked関数とputcマクロも参照してください。

Return value

x	ファイルに書き込んだ文字。正常終了を示します。
EOF	出力エラーを示します。

fputc_unlocked (Alpha, I64)

fputc関数と同様ですが、flockfileとfunlockfileで保護された範囲内だけで使用します。

Format

```
#include <stdio.h>

int fputc_unlocked (int character, FILE *file_ptr);
```

Argument

character
書き込む文字。int型のオブジェクト。

file_ptr
ファイル・ポインタ。

Description

putc_unlockedマクロを参照してください。

__UNIX_PUTC マクロを定義してコンパイルすれば、この関数の高速インライン版を使用して、最適化できます。

flockfile、ftrylockfile、およびfunlockfileも参照してください。

Return value

n	返された文字。
EOF	ファイルの終端 (EOR) またはエラーを示します。

fputs

文字列をファイルに書き込みます。ただし、文字列のヌル区切り文字 (\0) はコピーしません。

Format

```
#include <stdio.h>

int fputs (const char *str, FILE *file_ptr);
```

Argument

str
文字列を指すポインタ。

file_ptr
ファイル・ポインタ。

Description

putsと異なり、fputs関数は出力文字列の末尾に改行文字を追加しません。

putsも参照してください。

Return value

負でない値	正常終了を示します。
EOF	エラーを示します。

fputwc

1 文字のワイド文字を対応するマルチバイト値に変換し、結果を指定のファイルに書き込みます。

Format

```
#include <wchar.h>

wint_t fputwc (wint_t wc, FILE *file_ptr);
```

Argument

wc
wint_t 型のオブジェクト。

file_ptr
ファイル・ポインタ。

Description

fputwc 関数は、1 文字のワイド文字をファイルに書き込み、その文字を返します。

putwc も参照してください。

Return value

x	ファイルに書き込んだ文字。正常終了を示します。
---	-------------------------

WEOF

出力エラーを示します。errnoは次の値に設定されます。

- EILSEQ— 不正なワイド文字コードが検出されました。

I/O サブシステムからエラーが返された場合も、errnoが次の値に設定されます。

- EBADF— ファイル記述子が不正です。
- EIO— I/O エラー。
- ENOSPC— ファイルを格納しているデバイスに空き領域がありません。
- ENXIO— デバイスが存在しません。
- EPIPE— パイプが壊れています。
- ESPIPE— 追加のためにオープンされているファイルで不正なシークが行われました。
- EVMSError— 変換不可能な OpenVMS エラー。vaxc\$errnoには OpenVMS エラー・コードが格納されます。これは、対応する C エラー・コードが定義されていない I/O エラーが発生したことを示します。

fputws

ワイド文字の文字列をファイルに書き込みます。ヌル区切り文字はコピーしません。

Format

```
#include <wchar.h>

int fputws (const wchar_t *wstr, FILE *file_ptr);
```

Argument

wstr
ワイド文字の文字列を指すポインタ。

file_ptr
ファイル・ポインタ。

Description

`fputws`関数は、指定されたワイド文字の文字列をマルチバイト文字列に変換し、その文字列を指定のファイルに書き込みます。ヌル・ワイド文字に対応する区切り文字のヌル・バイトは出力文字列に追加されません。

Return value

負でない値	正常終了を示します。
-1	エラーを示します。この関数は <code>errno</code> を設定します。値の一覧については、 <code>fputwc</code> を参照してください。

fread

指定の数の項目をファイルから読み込みます。

Format

```
#include <stdio.h>

size_t fread (void *ptr, size_t size_of_item, size_t number_items, FILE *file_ptr);
```

Argument

ptr
メモリ内の記憶位置を指すポインタ。読み込んだ情報はその記憶位置に格納されます。ポインタが示すオブジェクトの型は、読み込む項目の型によって決定されます。

size_of_item
読み込む項目のサイズ (バイト数)。

number_items
読み込む項目の数。

file_ptr
項目を読み込むファイルを示すポインタ。

Description

`size_t`型は、次に示すように`<stdio.h>`ヘッダ・ファイルに定義されています。

```
typedef unsigned int size_t
```

読み込みはファイルの現在の位置から開始されます。読み込んだ項目は、最初の引数によって指定される記憶位置から始まる記憶域に格納されます。項目のサイズもバイト数で指定しなければなりません。

`file_ptr`によって示されるファイルがレコード・モードでオープンされている場合は、`fread`は`size_of_item`に`number_items`を乗算したバイト数をファイルから読み込みます。つまり、必ずしも`number_items`によって示される数のレコードが読み込まれるわけではありません。

fread

Return value

n	読み込んだバイト数をsize_of_itemで除算した値。
0	ファイルの終端 (EOF) またはエラーを示します。

free

前に実行した`calloc` , `malloc` , `realloc`呼び出しで割り当てられた領域の割り当てを解除して、再割り当て可能な状態にします。

Format

```
#include <stdlib.h>

void free (void *ptr);
```

引数

`ptr`
`malloc` , `calloc` , `realloc`の前の呼び出しから返されたアドレス。`ptr`が `NULL` ポインタの場合は、動作は何も実行されません。

Description

ANSI C 標準では、`free`は値を返さない関数として定義されています。したがって、`free`の関数プロトタイプは、戻り値の型が`void`で宣言されています。しかし、`free`が異常終了することもあり、HP C RTL の以前のバージョンでは、`free`は`int`を返すように宣言されていたため、`free`の実装では、正常終了時に `0` , 異常終了時に `-1` を返します。

freopen

ファイル・ポインタによって示されるオープンされているファイルを、ファイル指定によって示されるファイルに置き換えます。オープンされているファイルはクローズされます。

Format

```
#include <stdio.h>

FILE *freopen (const char *file_spec, const char *a_mode, FILE *file_ptr, ... );
```

Argument

file_spec

有効な OpenVMS または UNIX 形式のファイル指定を格納した文字列を指すポインタ。関数呼び出しの後、指定のファイル・ポインタがこのファイルに関連付けられます。

a_mode

アクセス・モード指示子。詳細については、fopen関数を参照してください。

file_ptr

ファイル・ポインタ。

...

省略可能なファイル属性引数。ファイル属性引数は、creat関数で使用する引数と同じです。

Description

freopen関数は通常、定義済みの名前stdin、stdout、stderrのいずれかをファイルに関連付けるために使用されます。これらの定義済みの名前の詳細については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第2章を参照してください。

Return value

`file_ptr`
NULL

freopenが正常終了した場合は、ファイル・ポインタ。
エラーを示します。

frexp

浮動小数点数値の小数部および指数部を計算します。

Format

```
#include <math.h>

double frexp (double value, int *eptr);
float frexpf (float value, int *eptr); (Alpha, I64)
long double frexpl (long double value, int *eptr); (Alpha, I64)
```

Argument

value
double , float , long double型の浮動小数点数値。

eptr
frexpが指数を格納するintを指すポインタ。

Description

frexp関数は浮動小数点数値 (value) を、次に示すように正規化された小数部と、2のべき乗に分割します。

$$\text{value} = \text{fraction} * (2^{\text{exp}})$$

小数部は戻り値として返されます。指数部は、eptrによって示される整数変数に格納されます。

例

```
#include <math.h>

main ()
{
    double val = 16.0, fraction;
    int exp;

    fraction = frexp(val, &exp);
    printf("fraction = %f\n", fraction);
    printf("exp = %d\n", exp);
}
```

この例では、frexpは16という値を $5 * 2^5$ に変換します。この例では次の出力が生成されます。

```
fraction = 0.500000
exp = 5
```

| value | =無限大またはNaNは、不正な引数です。

Return value

x	valueの小数部。
0	結果の小数部と整数部の両方が0です。
NaN	valueがNaNの場合は、NaNが返され、errnoはEDOMに設定され、*eptrの値は不定になります。
value	value =無限大の場合は、valueが返され、errnoはEDOMに設定され、*eptrの値は不定になります。

fscanf

指定されたファイルから書式設定された入力を実行し、その入力を書式指定に従って解釈します。

Format

```
#include <stdio.h>

int fscanf (FILE *file_ptr, const char *format_spec, ... );
```

Argument

file_ptr

入力テキストを提供するファイルを指すポインタ。

format_spec

書式指定を格納した文字列を指すポインタ。変換文字の詳細については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル(上巻)』第2章を参照してください。

...

省略可能な式であり、この式の結果は書式指定に指定した変換指定に対応します。

変換指定を指定しない場合は、入力ポインタを省略できます。変換指定を指定する場合は、関数呼び出しに変換指定と正確に同じ数の入力ポインタを指定する必要があります。変換指定は入力ポインタの型と一致しなければなりません。

変換指定は左から右への順に入力ソースに対応付けられます。入力ポインタの数の方が多い場合は、超過するポインタは無視されます。

Description

次の例は変換指定を示しています。

```
#include <stdio.h>

main ()
{
    int    temp, temp2;

    fscanf(stdin, "%d %d", &temp, &temp2);
    printf("The answers are %d, and %d.", temp, temp2);
}
```

stdinによって指定されるファイルに次の内容が格納されているとします。

4 17

この例の変換指定は次の結果を生成します。

The answers are 4, and 17.

書式指定と入力ポインタの詳細については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 2 章を参照してください。

Return value

x	正しく照合され、代入された入力項目の数。
EOF	<p>ファイルの終端 (EOF) が検出されたか、読み込みエラーが発生したことを示します。読み込みエラーが発生した場合は、errnoは次のいずれかに設定されます。</p> <ul style="list-style-type: none">• EILSEQ— 不正な文字が検出されました。• EVMSERR— 変換不可能な OpenVMS エラー。vaxc\$errnoには OpenVMS エラー・コードが格納されます。これはオーバーフローのために数値への変換に失敗したことを示します。 <p>I/O サブシステムからエラーが返された場合、この関数はerrnoを次の値に設定することがあります。</p> <ul style="list-style-type: none">• EBADF— ファイル記述子が不正です。• EIO— I/O エラー。• ENXIO— デバイスが存在しません。• EPIPE— パイプが壊れています。• EVMSERR— 変換不可能な OpenVMS エラー。vaxc\$errnoには OpenVMS エラー・コードが格納されます。これは、対応する C エラー・コードが定義されていない I/O エラーが発生したことを示します。

fseek

ファイルの位置をファイル内の指定されたバイト・オフセットに設定します。

Format

```
#include <stdio.h>

int fseek (FILE *file_ptr, long int offset, int direction);
```

Argument

file_ptr

ファイル・ポインタ。

offset

バイト数で指定したオフセット。

direction

新しい位置を計算するためにoffsetが加算される位置を示す整数。新しい位置は、directionがSEEK_SETの場合はファイルの先頭、directionがSEEK_CURの場合はファイル位置指示子の現在の値、directionがSEEK_ENDの場合はファイルの終端 (EOF) になります。

Description

fseek関数は、キャリッジ制御やストリーム・アクセス・ファイルを含まない固定長レコード・アクセス・ファイルの場合、ファイルの位置を任意のバイト・オフセットに設定できますが、他のすべてのファイルの場合はレコード境界にのみ設定できます。

提供される標準 I/O 関数は、可変長または VFC レコード・ファイルを最初のバイト、ファイルの終端 (EOF)、レコード境界のいずれかに設定します。したがって、fseekに指定する引数は次のいずれかを指定しなければなりません。

- ファイルの先頭または終端
- 現在の位置からオフセット 0 (任意のレコード境界)
- 前の有効なftell呼び出しから返された位置

これらのレコード・タイプのファイルで任意の位置までシークするための移植可能な方法については、`fgetpos`および`fsetpos`関数を参照してください。

警告

ストリーム・ファイルにアクセスするときに、ファイルの終端 (EOF) を超えてシークし、ファイルに書き込んだ場合、`fseek`関数はスキップしたバイトに 0 を埋め込むことにより、ホールを作成します。

一般にレコード・ファイルに対して`fseek`関数を使用する場合、`ftell`に対する前の有効な呼び出しによって返された絶対位置に設定するか、あるいはファイルの先頭または終端に設定するようにしなければなりません。`fseek`の呼び出しがこれらの条件を満たさない場合、結果は予測できません。

`open` , `creat` , `dup` , `dup2` , `lseek`も参照してください。

Return value

0	シークが正常終了したことを示します。
-1	不正なシークが指定されたことを示します。

fseeko

ファイルの位置をファイル内の指定されたバイト・オフセットに設定します。この関数はfseekと同じです。

Format

```
#include <stdio.h>

int fseeko (FILE *file_ptr, off_t offset, int direction);
```

Argument

file_ptr

ファイル・ポインタ。

offset

バイト数で指定したオフセット。off_tデータ型は32ビット整数または64ビット整数です。64ビット・インタフェースの場合、2 GB以上のファイル・サイズが認められます。このインタフェースは、次に示すように_LARGEFILE 機能テスト・マクロを定義することにより、コンパイル時に選択できます。

```
CC/DEFINE= _LARGEFILE
```

direction

新しい位置を計算するためにoffsetが加算される位置を示す整数。新しい位置は、directionがSEEK_SETの場合はファイルの先頭、directionがSEEK_CURの場合はファイル位置指示子の現在の値、directionがSEEK_ENDの場合はファイルの終端 (EOF) になります。

Description

fseeko関数はfseek関数と同じですが、offset引数のデータ型がlong intではなく、off_t型である点が異なります。

fsetpos

指定されたファイルのファイル位置指示子を設定します。

Format

```
#include <stdio.h>

int fsetpos (FILE *stream, const fpos_t *pos);
```

Argument

stream

ファイル・ポインタ。

pos

実装で定義されている構造体を指すポインタ。fgetpos関数は、この後のfsetposの呼び出しで利用できる情報をこの構造体に格納します。

Description

fgetpos関数を使用する前に、fsetpos関数を呼び出します。

Return value

0	正常終了を示します。
-1	エラーを示します。

fstat

ファイル記述子によって指定されるファイルに関する情報にアクセスします。

Format

```
#include <stat.h>

int fstat (int file_desc, struct stat *buffer);
```

関数バリエーション

`_DECC_V4_SOURCE` および `_VMS_V6_SOURCE` 機能テスト・マクロを定義してコンパイルすると、`fstat` 関数に対して、OpenVMS Version 7.0 より前の動作に相当するローカル時刻ベースのエントリ・ポイントが有効になります。

Argument

`file_desc`

ファイル記述子。

`buffer`

`stat_t` 型の構造体を指すポインタ。この構造体は `<stat.h>` ヘッダ・ファイルに定義されています。指定されたファイルに関する情報がこの引数に格納されます。`buffer` によって示される構造体のメンバは次のとおりです。

メンバ	型	定義
<code>st_dev</code>	<code>dev_t</code>	物理デバイス名を指すポインタ
<code>st_ino[3]</code>	<code>ino_t</code>	ファイル ID が格納される 3 ワード
<code>st_mode</code>	<code>mode_t</code>	ファイル“mode” (<code>prot</code> , <code>dir</code> , ...)
<code>st_nlink</code>	<code>nlink_t</code>	UNIX システムとの互換性を維持するためにのみ提供される
<code>st_uid</code>	<code>uid_t</code>	オーナーのユーザ ID
<code>st_gid</code>	<code>gid_t</code>	グループ・メンバ: <code>st_uid</code> から取得
<code>st_rdev</code>	<code>dev_t</code>	UNIX システムとの互換性-常に 0
<code>st_size</code>	<code>off_t</code>	バイト数で表したファイル・サイズ
<code>st_atime</code>	<code>time_t</code>	ファイル・アクセス時刻, 常に <code>st_mtime</code> と同じ
<code>st_mtime</code>	<code>time_t</code>	最終変更時刻
<code>st_ctime</code>	<code>time_t</code>	ファイル作成時刻
<code>st_fab_rfm</code>	<code>char</code>	レコード・フォーマット

メンバ	型	定義
st_fab_rat	char	レコード属性
st_fab_fsz	char	固定ヘッダ・サイズ
st_fab_mrs	unsigned	レコード・サイズ

型 `dev_t`, `ino_t`, `off_t`, `mode_t`, `nlink_t`, `uid_t`, `gid_t`, `time_t` は `<stat.h>` ヘッダ・ファイルに定義されています。しかし、互換性を維持するように設定してコンパイルした場合は (`/DEFINE=_DECC_V4_SOURCE`), `dev_t`, `ino_t`, `off_t` だけが定義されます。

`off_t` データ型は 32 ビット整数または 64 ビット整数です。64 ビットインタフェースの場合、2 GB 以上のファイル・サイズが認められます。このインタフェースは、次に示すように `_LARGEFILE` 機能テスト・マクロを定義することにより、コンパイル時に選択できます。

```
CC/DEFINE=_LARGEFILE
```

OpenVMS Version 7.0 では、時刻は Epoch (1970 年 1 月 1 日 GMT 標準時 00:00:00) からの時間 (秒数) で与えられます。

`st_mode` 構造体メンバは状態情報モードであり、`<stat.h>` ヘッダ・ファイルに定義されています。`st_mode` ビットは次のとおりです。

ビット	定数	定義
0170000	S_IFMT	ファイルのタイプ
0040000	S_IFDIR	ディレクトリ
0020000	S_IFCHR	文字スペシャル
0060000	S_IFBLK	ブロック・スペシャル
0100000	S_IFREG	一般
0030000	S_IFMPC	多重化された文字スペシャル
0070000	S_IFMPB	多重化されたブロック・スペシャル
0004000	S_ISUID	実行時にユーザ ID を設定
0002000	S_ISGID	実行時にグループ ID を設定
0001000	S_ISVTX	使用後もスワップされたテキストを保存
0000400	S_IRREAD	読み込みアクセス許可, オーナ
0000200	S_IWRITE	書き込みアクセス許可, オーナ
0000100	S_IEXEC	実行/検索アクセス許可, オーナ

Description

fstat関数はリモート・ネットワーク・ファイルに対しては機能しません。

注意 (Alpha, I64)

OpenVMS Alpha システムおよび I64 システムでは、stat, fstat, utime, utimes関数は、POSIX 準拠のファイル・タイムスタンプに対する新しいファイル・システム・サポートを活用できるように拡張されています。

このサポートは、OpenVMS Alpha Version 7.3 およびそれ以降の OpenVMS Alpha システムと I64 システムの ODS-5 デバイスでのみ使用できます。

このように変更される前は、stat関数とfstat関数はst_ctime, st_mtime, st_atime フィールドの値を、次のファイル属性をもとに設定していました。

st_ctime - ATR\$C_CREDATE (ファイル作成時刻)
 st_mtime - ATR\$C_REVDATE (ファイル改訂時刻)
 st_atime - ファイル・アクセス時刻がサポートされないため、常に st_mtime に設定。

また、ファイル変更時刻に関しては、utimeとutimesはATR\$C_REVDATE ファイル属性を変更し、file-access-time 引数を無視していました。

機能が変更された後、ODS-5 デバイスのファイルでは、stat関数とfstat関数は次の新しいファイル属性をもとに、st_ctime, st_mtime, st_atime フィールドの値を設定します。

st_ctime - ATR\$C_ATTDATE (最終属性変更時刻)
 st_mtime - ATR\$C_MODDATE (最終データ変更時刻)
 st_atime - ATR\$C_ACCDATE (最終アクセス時刻)

ODS-2 デバイスなど、ATR\$C_ACCDATE が 0 の場合、stat関数とfstat関数はst_atime を st_mtime に設定します。

ファイル変更時刻に関しては、utime関数とutimes関数はATR\$C_REVDATE ファイル属性と ATR\$C_MODDATE ファイル属性の両方を変更します。ファイル・アクセス時刻に関しては、これらの関数はATR\$C_ACCDATE ファイル属性を変更します。ATR\$C_MODDATE ファイル属性と ATR\$C_ACCDATE ファイル属性を ODS-2 デバイスで設定しても、効果はありません。

互換性を維持するために、デバイスの種類にかかわらず、stat, fstat, utime, utimesの以前の動作はデフォルトのまま残されています。

新しい動作は、アプリケーションを起動する前に、実行時に DECC\$EFS_FILE_TIMESTAMPS 論理名を "ENABLE"に定義することにより明示的に有効にする必要があります。この論理名を設定しても、ODS-2 デバイスのファイルの場合、stat, fstat, utime, utimesの動作には影響ありません。

Return value

0	正常終了を示します。
-1	保護違反以外のエラーを示します。
-2	保護違反を示します。

fstatvfs (Alpha, I64)

指定されたファイルが格納されているデバイスに関する情報を取得します。

Format

```
#include <statvfs.h>

int fstatvfs (int filedes, struct statvfs *buffer);
```

Argument

filedes

open関数またはfcntl関数の呼び出しで取得したファイル記述子。

buffer

返される情報を保持する，statvfs構造体へのポインタ。

Description

fstatvfs関数は、指定されたファイルが格納されているデバイスに関する情報を返します。指定されたファイルに対する読み込み，書き込み，実行の許可は必要ありません。情報は，statvfs構造体の形で返されます。この構造体はヘッダ・ファイル<statvfs.h>で定義されており，以下のメンバが含まれています。

unsigned long f_bsize - 優先ブロックサイズ。

unsigned long f_frsize - 基本ブロックサイズ。

fsblkcnt_t f_blocks - ブロック数の合計。単位はf_frsize

fsblkcnt_t f_bfree - 空きブロックの合計数。DFS ディスクに対する\$GETDVIで，無意味な空きブロック数が誤ってレポートされる場合，f_bfreeには最大ブロック数が設定されます。

fsblkcnt_t f_bavail - 利用可能な空きブロックの数。呼び出し元のディスク・クォータの未使用分が設定されます。

fsfilcnt_t f_files - ファイル (inode) の合計数。

fsfilcnt_t f_ffree - 空きファイル (inode) 数。OpenVMS システムでは、この値は「空きブロック数/クラスタサイズ」で計算されます。

fsfilcnt_t f_favail - 非特権空きファイル (inode) 数。f_ffreeが設定されます。

unsigned long f_fsid - ファイル・システム ID。この ID は、割り当てクラス・デバイス名に基づきます。デバイスがローカルにマウントされている限り、デバイスに基づいた一意の値が提供されます。

unsigned long f_flag - 以下の 1 つ以上のフラグを表すビット・マスク。

ST_RDONLY - ボリュームは読み込み専用。

ST_NOSUID - ボリュームで保護されたサブシステムが有効。

unsigned long f_namemax - ファイル名の最大長。

char f_basetype[64] - デバイス・タイプ名。

char f_fstr[64] - 論理ボリューム名。

char __reserved[64] - メディア・タイプ名。

正常に完了すると、fstatvfsは 0 (ゼロ) を返します。失敗すると-1 を返し、errnoにエラーを示す値を設定します。

statvfsも参照してください。

戻り値

0	正常終了。
-1	<p>エラーを示します。errnoには、以下のいずれかが設定されます。</p> <ul style="list-style-type: none"> • EBADF - ファイル記述子パラメータの値が不正です。 • EIO - デバイスの読み込み中に入出力エラーが発生しました。 • EINTR - 関数の実行中にシグナルを受信しました。 • EOVERFLOW - 返却する値の 1 つが、bufferが示す構造体で正しく表現できません。

fsync

すべてのデータをディスクに書き込みます。

Format

```
#include <unistd.h>

int fsync (int fd);
```

引数

fd
オープンされているファイルに対応するファイル記述子。

Description

fsync関数はfflush関数とほとんど同じ動作をします。2つの関数の主な違いは、fflushがRMSバッファに格納されているデータだけを書き込むのに対し、fsyncはすべてのデータをディスクに書き込む点です。また、fflushでは、すべてのバッファを一度に書き込むことができます。fsyncでは、このような操作はできません。

Return value

0	正常終了を示します。
-1	エラーを示します。

ftell

指定されたストリーム・ファイルの現在のバイト・オフセットを返します。

Format

```
#include <stdio.h>

long int ftell (FILE *file_ptr);
```

引数

file_ptr
ファイル・ポインタ。

Description

ftell関数は、ファイルの先頭からのバイト・オフセットを返します。

可変長ファイル、VFC ファイル、キャリッジ制御属性を含むファイルの場合、ファイルがレコード・モードでオープンされているときは、ftellは現在のバイト・オフセットではなく、現在のレコードの先頭の位置を返します。

レコード・ファイルを使用する場合、ftell関数は、ungetcまたはungetwcを使用して取得された文字を無視します。ストリーム・ファイルが使用されているときは、この動作は発生しません。

どのファイル・タイプでも正確なオフセットを判断するための移植可能な方法については、fgetpos関数を参照してください。

Return value

n	現在のオフセット。
EOF	エラーを示します。

ftello

指定されたストリーム・ファイルの現在のバイト・オフセットを返します。この関数はftellと同じです。

Format

```
#include <stdio.h>

off_t ftello (FILE *file_ptr);
```

引数

file_ptr
ファイル・ポインタ。

Description

ftello関数はftell関数と同じですが、戻り値がlong int型ではなく、off_t型である点が異なります。

off_tデータ型は64ビット整数または32ビット整数です。64ビット・インタフェースでは、2 GB以上のファイル・サイズが認められます。このインタフェースは、次に示すように_LARGEFILE 機能テスト・マクロを定義することにより、コンパイル時に選択できます。

```
CC/DEFINE=_LARGEFILE
```

ftime

timeptrによって示される構造体に、1970年1月1日00:00:00からの経過時間を返します。

Format

```
#include <timeb.h>

int ftime (struct timeb *timeptr);
```

関数バリエーション

_DECC_V4_SOURCE および_VMS_V6_SOURCE 機能テスト・マクロを定義してコンパイルすると、ftime関数に対して、OpenVMS Version 7.0 より前の動作に相当するローカル時刻ベースのエントリ・ポイントが有効になります。

引数

timeptr
構造体timeb_tを指すポインタ。

Description

typedef timeb_tは、<timeb.h>ヘッダ・ファイルに定義されている次の構造体を参照します。

```
typedef struct timeb
{
    time_t      time;
    unsigned short millitm;
    short       timezone;
    short       dstflag;
};
```

メンバtimeは秒数で時刻を示します。

メンバmillitmは小数点以下の時間をミリ秒単位で示します。

ftimeを呼び出した後、timeb構造体のtimezoneメンバとdstflagメンバには、それぞれグローバル変数timezoneおよびdstflagの値が格納されます。timezoneおよびdstflagグローバル変数については、tzset関数の説明を参照してください。

Return value

0	正常終了。timeb_t構造体に情報が格納されます。
-1	エラーを示します。システムの時差係数 (つまりシステム時刻と UTC 時刻の差) が正しく設定されていないことを示すことがあります。 SYSSTIMEZONE_DIFFERENTIAL 論理名の値が不正な場合は、この関数は異常終了し、errnoはEINVALに設定されます。

ftruncate

ファイルを指定の長さに切り捨てます。

Format

```
#include <unistd.h>

int ftruncate (int filedes, off_t length);
```

Argument

filedes

書き込みのためにオープンされているファイルの記述子。

length

ファイルの新しい長さ (バイト数)。off_tデータ型は 32 ビット整数または 64 ビット整数です。64 ビット・インタフェースでは、2 GB 以上のファイル・サイズが認められます。このインタフェースは、次に示すように_LARGEFILE 機能テスト・マクロを定義することにより、コンパイル時に選択できます。

```
CC/DEFINE= _LARGEFILE
```

Description

ftruncate関数は、ファイルを指定された位置で切り捨てます。レコード・ファイルの場合、位置はレコード境界でなければなりません。また、ファイルはローカル的一般ファイルでなければなりません。

切り捨てる前のファイルのサイズがlengthより大きい場合は、超過するデータは消失します。切り捨てる前のファイルのサイズがlengthより短い場合は、前の長さと新しい長さの間のバイトは 0 として読み込まれます。

Return value

0

正常終了を示します。

-1

エラーが発生しました。errnoはエラーを示すように設定されます。

ftrylockfile (Alpha, I64)

stdio (FILE*) オブジェクトの所有権を取得します。

Format

```
#include <stdio.h>
int ftrylockfile (FILE *file_ptr);
```

引数

file_ptr
ファイル・ポインタ。

Description

ftrylockfile関数はスレッドにより使用され、stdio (FILE*) オブジェクトが使用可能であれば、その所有権を取得します。ftrylockfile関数は、flockfileのノン・ブロッキング版です。

ftrylockfile関数は、成功するとゼロを返し、ロックを取得できないとゼロ以外を返します。

flockfileおよびfunlockfileも参照してください。

Return value

0	成功を示します。
0 以外	ロックを取得できなかったことを示します。

ftw

ファイル・ツリーを検索します。

Format

```
#include <ftw.h>

int ftw (const char *path, int(*function)(const char *, const struct stat *, int), int depth);
```

Argument

path

検索するディレクトリ階層構造。

function

ディレクトリ階層構造の各ファイルに対して起動する関数。

depth

ftwで使えるディレクトリ・ストリームまたはファイル記述子 (またはその両方) の最大数。この引数は 1 ~ OPEN_MAX の範囲でなければなりません。

Description

ftw関数は、path引数によって指定されるディレクトリから順にディレクトリ階層構造を再帰的に検索します。

階層構造内の各ファイルに対して、ftwはfunction引数によって指定される関数を呼び出し、ファイルの名前を格納したヌル区切り文字列、ファイルに関する情報を格納したstat構造体を指すポインタ、および整数を渡します。

整数はファイル・タイプを示します。使用できる値は<ftw.h>に定義されており、次のとおりです。

FTW_F	一般ファイル
FTW_D	ディレクトリ
FTW_DNR	読み込むことができないディレクトリ
FTW_NS	statを正しく実行できないファイル

整数がFTW_DNRの場合は、そのディレクトリに格納されているファイルとサブディレクトリは処理されません。

整数が FTW_NS の場合は、stat 構造体の内容は意味がありません。たとえば、読み込みアクセス許可が与えられ、実行 (検索) アクセス許可が与えられていないディレクトリ内のファイルの場合、function 引数は FTW_NS を渡します。

ftw 関数はファイルまたはサブディレクトリを処理する前に、ディレクトリの処理を終了します。

ftw 関数は次のいずれかが発生するまで、検索を続行します。

- path 引数によって指定されたディレクトリ階層構造が終了するまで。
- function 引数によって指定された関数の呼び出しが 0 以外の値を返すまで。
- ftw 関数の内部でエラー (I/O エラーなど) が検出されるまで。

ftw 関数は再帰的であるため、非常に深いファイル構造に適用すると、スタック・オーバーフローが発生して、メモリ・フォルトで終了する可能性があります。

ftw 関数では、操作時に動的記憶域を割り当てるために malloc 関数が使用されます。function 引数によって示される関数から longjmp を呼び出した場合など、ftw が強制終了された場合、ftw はその記憶域の割り当てを解除することができません。したがって、記憶域は割り当てられたままになります。

割り込みを安全に処理するには、割り込みが発生したという事実を格納し、function 引数によって指定される関数が次回呼び出されるときに 0 以外の値を返すようにします。

注意

- ftw 関数はリエントラントです。function 引数として指定される関数もリエントラントであることを確認してください。
 - C RTL は、stat 構造体の標準準拠の定義および関連する定義をサポートしています。これを使用するには、機能テスト・マクロ_USE_STD_STAT を使用してアプリケーションをコンパイルします。詳細は、お使いのシステムのヘッダ・ファイル <stat.h> を参照してください。
-

malloc, longjump, および stat も参照してください。

Return value

0 正常終了を示します。

x

function引数によって指定される関数が検索を停止したことを示し、関数から返された値を返します。

-1

エラーを示します。errnoは次のいずれかの値に設定されます。

- EACCES—path引数のコンポーネントに対して検索アクセス許可が拒否されたか、またはpath引数に対して読み込みアクセス許可が拒否されました。
- ENAMETOOLONG—パス文字列の長さが PATH_MAX より長い、または[_POSIX_NO_TRUNC]が有効なときに、パス名コンポーネントが NAME_MAX より長いことを示します。
- ENOENT—path引数が存在しないファイルの名前を指しているか、または空文字列を指しています。
- ENOMEM—この操作を実行するのに必要なメモリが不足しています。

また、function引数によって示される関数でエラーが検出されると、errnoが適切な値に設定されることがあります。

funlockfile (Alpha, I64)

stdioストリームをアンロックします。

Format

```
#include <stdio.h>

void funlockfile (FILE *file_ptr);
```

引数

file_ptr
ファイル・ポインタ。

Description

funlockfile関数は、stdioストリームをアンロックします。ロックを保持していたスレッドは、ストリームの排他使用を手放すことになります。

引数のファイル・ポインタは、正しいことが前提です。flockfileは、ファイル・ポインタが無効でもロックを実行します。また、funlockfile関数は、呼び出し元スレッドが引数のファイル・ポインタのロックを所有していない場合でも、エラーになりません。

対応するflockfileとfunlockfileの呼び出しは、ネストさせることができます。ストリームを再帰的にロックすると、対応する最後のfunlockfileを呼び出すまでは、ストリームはロックされたままになります。

すべてのC RTL ファイル・ポインタ入出力関数は、flockfileおよびfunlockfileを呼び出したかのように、ファイル・ポインタをロックします。

flockfileおよびftrylockfileも参照してください。

fwait

特定のファイルに対する I/O が完了するのを待ちます。

Format

```
#include <stdio.h>
int fwait (FILE *fp);
```

引数

fp
オープンされているファイルに対応するファイル・ポインタ。

Description

fwait関数は主に、保留状態の非同期 I/Oが完了するのを待つために使用されます。

Return value

0	正常終了を示します。
-1	エラーを示します。

fwide

ストリームの単位を判断し，設定します。

Format

```
#include <wchar.h>

int fwide (FILE *stream, int mode);
```

Argument

stream
ファイル・ポインタ。

mode
ストリームの単位を指定する値。

Description

fwide関数は，streamによって示されるストリームの単位を判断し，ストリームの単位が設定されていない場合は，次の方法でmode引数に従って設定します。

mode引数:	fwide 関数の動作:
0 より大きい場合	ストリームをワイド文字単位に設定する。
0 より小さい場合	ストリームをバイト単位に設定する。
0 の場合	ストリームの単位を変更しない。

ストリームの単位がすでに設定されている場合は，fwideは単位を変更しません。fwideに対してエラー状態が定義されていないため，fwideが0を返した場合は，呼び出し元のアプリケーションはerrnoを確認する必要があります。

Return value

> 0	呼び出しの後、ストリームはワイド文字単位になります。
< 0	呼び出しの後、ストリームはバイト単位になります。
0	呼び出しの後、ストリームは単位が設定されない状態になるか、またはストリーム引数が不正です。errnoが設定されます。

fwprintf

ワイド文字の書式文字列の制御のもとで出力をストリームに書き込みます。

Format

```
#include <wchar.h>

int fwprintf (FILE *stream, const wchar_t *format, ... );
```

Argument

stream

ファイル・ポインタ。

format

書式指定を格納したワイド文字の文字列を指すポインタ。書式指定および変換指定とそれに対応する引数の詳細については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 2 章を参照してください。

...

省略可能な式であり、式の型は書式指定に指定した変換指定に対応します。

変換指定を指定しない場合は、出力ソースを省略できます。変換指定を指定する場合は、関数呼び出しに変換指定と正確に同じ数の出力ソースを指定する必要があります。変換指定は出力ソースの型と一致しなければなりません。

変換指定は左から右への順に出力ソースに対応付けられます。出力ソースの数の方が多い場合は、超過するソースは無視されます。

Description

fwprintf関数は、formatによって示されるワイド文字の文字列の制御のもとで、streamによって示されるストリームに出力を書き込みます。書式指定は、後の引数を出力に変換する方法を指定します。書式指定に対して引数が不足している場合は、動作は未定義になります。引数が残っているのに、書式指定が不足する場合は、超過する引数は評価されますが、無視されます。書式指定文字列の末尾が検出されると、fwprintf関数は制御を呼び出し元に返します。

format 引数は、次のような 0 個以上のディレクティブで構成されます。

- 通常のワイド文字 (パーセント記号(%)を除く)
- 変換指定

Return value

n

負の値

書き込んだワイド文字の文字数。

エラーを示します。errno は次のいずれかの値に設定されます。

- EILSEQ— 不正な文字が検出されました。
- EINVAL— 引数が不足しています。
- ENOMEM— 変換のために使用できるメモリが不足しています。
- ERANGE— 浮動小数点演算オーバーフロー。
- EVMSERR— 変換不可能な OpenVMS エラー。vaxc\$errno には OpenVMS エラー・コードが格納されます。これは、オーバーフローが発生したために数値への変換が失敗したことを示します。

I/O サブシステムからエラーが返された場合、この関数は errno を次の値に設定することがあります。

- EBADF— ファイル記述子が不正です。
- EIO— I/O エラー。
- ENOSPC— ファイルを格納しているデバイスに空き領域がありません。
- ENXIO— デバイスが存在しません。
- EPIPE— パイプが壊れています。
- ESPIPE— 追加のためにオープンされているファイルで不正なシークが実行されました。
- EVMSERR— 変換不可能な OpenVMS エラー。vaxc\$errno には、OpenVMS エラー・コードが格納されます。これは、対応する C エラー・コードがない I/O エラーが発生したことを示します。

例

次の例では、日付と時刻を "Sunday, July 3, 10:02" という形式でプリントし、その後、 π を小数点以下 5 桁まで印刷する方法を示しています。

```
#include <math.h>
#include <stdio.h>
#include <wchar.h>
/* ... */
wchar_t *weekday, *month; /* pointers to wide-character strings */
int day, hours, min;
fwprintf(stdout, L"%ls, %ls %d, %.2d:%.2d\n",
    weekday, month, day, hour, min);
fwprintf(stdout, L"pi = %.5f\n", 4 * atan(1.0));
```

fwrite

指定された数の項目をファイルに書き込みます。

Format

```
#include <stdio.h>

size_t fwrite (const void *ptr, size_t size_of_item, size_t number_items, FILE *file_ptr);
```

Argument

ptr
情報の書き込み元のメモリ位置を指すポインタ。示されるオブジェクトの型は書き込まれる項目の型によって決定されます。

size_of_item
書き込む項目のサイズ (バイト数)。

number_items
書き込む項目の数。

file_ptr
項目の書き込み先のファイルを示すファイル・ポインタ。

Description

`size_t`型は、次に示すように`<stdio.h>`ヘッダ・ファイルに定義されています。

```
typedef unsigned int size_t
```

書き込みはファイルの現在の位置から開始されます。項目は、最初の引数によって示される位置から始まる記憶域から書き込まれます。項目のサイズ (バイト数) も指定する必要があります。

`file_ptr`によって示されるファイルがレコード・ファイルの場合は、`fwrite`関数は少なくとも`number_items`個のレコードを出力します。各レコードの長さは`size_of_item`です。

戻り値

x

書き込んだ項目の数。書き込まれるレコードの数は、ファイルの最大レコード・サイズに応じて異なります。

fwscanf

ワイド文字の書式指定文字列の制御のもとで、ストリームから入力を読み込みます。

Format

```
#include <wchar.h>

int fwscanf (FILE *stream, const wchar_t *format, ... );
```

Argument

stream

ファイル・ポインタ。

format

書式指定を格納したワイド文字の文字列を指すポインタ。書式指定と変換指定、および対応する引数の詳細については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 2 章を参照してください。

...

省略可能な式であり、式の結果は書式指定に指定した変換指定に対応します。書式指定と変換指定、およびそれに対応する引数の詳細については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 2 章を参照してください。

変換指定を指定しない場合は、入力ポインタを省略できます。変換指定を指定する場合は、関数呼び出しに変換指定と正確に同じ数の入力ポインタを指定する必要があります。変換指定は入力ポインタの型と一致しなければなりません。

変換指定は左から右への順に入力ソースに対応付けられます。入力ポインタの数の方が多い場合は、超過するポインタは無視されます。

Description

fwscanf関数は、formatによって示されるワイド文字の文字列の制御のもとで、streamによって示されるストリームから入力を読み込みます。書式指定に対して引数が不足している場合は、動作は未定義になります。引数が残っているのに、書式指定が不足する場合は、超過する引数は評価されますが、無視されます。

書式指定は、次のような 0 個以上のディレクティブで構成されます。

- 1 つ以上の空白ワイド文字

- 通常のワイド文字 (パーセント記号(%)) と空白ワイド文字を除く)
- 変換指定

各変換指定はワイド文字%から始まります。

stream 引数によって示されるストリームがバイト単位でもワイド文字単位でもない場合は、fwscanfはストリームをワイド文字単位に設定します。

Return value

n	代入した入力項目の数。提供された項目の数より少なくなることがあり、早い段階で照合エラーが発生した場合は、0 になることもあります。
EOF	エラーを示します。変換の前に入力エラーが発生しました。

gcv

引数をヌル区切りの ASCII 数字列に変換し、文字列のアドレスを返します。

Format

```
#include <stdlib.h>

char *gcv (double value, int ndigit, char *buffer);
```

関数バリエーション

gcv関数には、_gcv32および_gcv64という名前のバリエーションがあり、それぞれ32ビット・ポインタ・サイズおよび64ビット・ポインタ・サイズで使用されます。ポインタ・サイズ固有の関数の使い方については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

Argument

value

ヌル区切りの ASCII 数字列に変換されるdouble型のオブジェクト。

ndigit

変換後の数字列で使用する ASCII 数字の桁数。ndigitが6未満の場合は、値は6になります。

buffer

変換後の数字列を格納する記憶域の位置。

Description

gcv関数は、変換された文字列をバッファに格納し、バッファのアドレスを返します。可能な場合は、gcvはF形式で有効桁ndigit桁の数字を生成します。不可能な場合はE形式で生成します。後続の0は削除されます。

ecvt、fcvt、gcv関数は、浮動小数点演算に関して IEEE 標準で指定されている次の特殊な値を表します。

gcvt

値	表現
クワイエット NaN	NaNQ
シグナリング NaN	NaNS
+ 無限大	無限大
− 無限大	−無限大

これらの各値に割り当てられる符号は、sign引数に格納されます。IEEE 浮動小数点表現では、0 (ゼロ) という値は正の場合も負の場合もあり、どちらであるかはsign引数によって設定されます。

fcvtおよびecvtも参照してください。

戻り値

X

バッファのアドレス。

getc

指定されたファイルから次の文字を返します。

Format

```
#include <stdio.h>
int getc (FILE *file_ptr);
```

引数

file_ptr
アクセスするファイルを指すポインタ。

Description

getcマクロは、file_ptrパラメータで指定された入力ストリームから次のバイトを取り出して返し、ファイル・ポインタが定義されていれば、ファイル・ポインタを入力ストリーム中で1バイト進めます。

getcはマクロであるため、副作用のあるファイル・ポインタ引数 (たとえば、getc (*f++)) は正しく評価されないことがあります。このような場合は、代わりにfgetc関数を使用してください。fgetc関数を参照してください。

getc_unlockedも参照してください。

Return value

n	返された文字。
EOF	ファイルの終端 (EOF) またはエラーを示します。

getc_unlocked (Alpha, I64)

getcマクロと同様ですが、flockfileとfunlockfileで保護された範囲内だけで使用します。

Format

```
#include <stdio.h>

int getc_unlocked (FILE *file_ptr);
```

引数

file_ptr
ファイル・ポインタ。

Description

リエントラント版であるgetcマクロは、複数スレッドからの同時呼び出しに対してロックされます。その結果、ストリームの一貫性を保証するためのオーバーヘッドが生じます。アンロック版であるgetc_unlockedを使用すると、このオーバーヘッドを避けることができます。getc_unlockedマクロは、getcマクロと機能的に同じですが、スレッド・セーフな方法で実装する必要がない点が異なります。getc_unlockedマクロは、flockfile関数とfunlockfile関数を対で使用して保護された範囲内でだけ、安全に使用することができます。呼び出し元は、getc_unlockedを使用する前に、ストリームを確実にロックする必要があります。

getc_unlockedはマクロであるため、副作用のあるファイル・ポインタ引数は正しく評価されないことがあります。このような場合は、代わりにfgetc_unlocked関数を使用してください。

flockfile、ftrylockfile、およびfunlockfileも参照してください。

Return value

n	返された文字。
EOF	ファイルの終端 (EOF) またはエラーを示します。

[w]getch

端末画面から 1 文字を取得し、その文字を指定のウィンドウに表示します。getch関数は文字をstdscrウィンドウに表示します。

Format

```
#include <curses.h>
char getch();
char wgetch (WINDOW *win);
```

引数

win
ウィンドウを指すポインタ。

Description

getch関数とwgetch関数は、文字を読み込む前に、指定されたウィンドウの表示を更新します。詳細については、scrolllok関数を参照してください。

Return value

x	返された文字。
ERR	関数が画面を不正にスクロールすることを示します。

getchar

標準入力 (stdin) から 1 文字を読み込みます。

Format

```
#include <stdio.h>
int getchar (void);
```

Description

getchar関数はfgetc(stdin) と同じです。

getchar_unlockedも参照してください。

Return value

x	stdinから読み込まれ、intに変換された次の文字。
EOF	ファイルの終端 (EOF) またはエラーを示します。

getchar_unlocked (Alpha, I64)

getchar関数と同様ですが、flockfileとfunlockfileで保護された範囲内だけで使用します。

Format

```
#include <stdio.h>

int getchar_unlocked (void);
```

Description

リエントラント版であるgetchar関数は、複数スレッドからの同時呼び出しに対してロックされます。その結果、ストリームの一貫性を保証するためのオーバーヘッドが生じます。アンロック版であるgetchar_unlockedを使用すると、このオーバーヘッドを避けることができます。getchar_unlocked関数は、getchar関数と機能的に同じですが、スレッド・セーフな方法で実装する必要がない点が異なります。getchar_unlocked関数は、flockfile関数とfunlockfile関数を対で使用して保護された範囲内でだけ、安全に使用することができます。呼び出し元は、getchar_unlockedを使用する前に、ストリームを確実にロックする必要があります。

flockfile、ftrylockfile、およびfunlockfileも参照してください。

Return value

x	stdinから読み込まれ、intに変換された次の文字。
EOF	ファイルの終端 (EOF) またはエラーを示します。

getclock

システム単位のクロックの現在の値を取得します。

Format

```
#include <timers.h>

int getclock (int clktyp, struct timespec *tp);
```

Argument

clktyp

システム単位のクロックの種類。

tp

システム単位のクロックの現在の値が格納されているtimespec構造体を指すポインタ。

Description

getclock関数は、clktypによって指定されるクロックの現在の値を、tpによって示される記憶位置に格納します。

clktyp引数は、<timers.h>ヘッダ・ファイルに定義されているシンボル定数名として指定します。TIMEOFDAYシンボル定数だけがサポートされます。この定数は、アクセスするシステム単位の時刻として、通常のtime-of-dayクロックを指定します。

TIMEOFDAYによって指定されるクロックの場合、この関数から返される値は、Epochからの経過時間です。Epochは1970年1月1日UTC（協定世界時）00:00:00を参照します。

getclock関数はtimespec構造体を返します。この構造体は、次に示すように<timers.h>ヘッダ・ファイルに定義されています。

```
struct timespec {
    unsigned long tv_sec    /* Elapsed time in seconds since the Epoch*/
    long         tv_nsec   /* Elapsed time as a fraction of a second */
                          /* since the Epoch (in nanoseconds)      */
};
```

Return value

0	正常終了を示します。
-1	<p>エラーを示します。errnoは次のいずれかの値に設定されます。</p> <ul style="list-style-type: none">• EINVAL—clktyp引数が認識されるシステム単位のクロックを指定していません。 または <code>SYS\$TIMEZONE_DIFFERENTIAL</code> 論理名の値が不正です。• EIO—clktyp引数によって指定されるシステム単位のクロックにアクセスしたときにエラーが発生しました。

getcwd

現在のワーキング・ディレクトリのファイル指定を指すポインタを返します。

Format

```
#include <unistd.h>

char *getcwd (char *buffer, size_t size); (ISO POSIX-1)
char *getcwd (char *buffer, unsigned int size, . . . ); (HP C Extension)
```

関数バリエーション

getcwd関数には、_getcwd32および_getcwd64という名前のバリエーションがあり、それぞれ 32 ビット・ポインタ・サイズおよび 64 ビット・ポインタ・サイズで使用されます。ポインタ・サイズ固有の関数の使い方については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

Argument

buffer

ディレクトリ指定を格納できるだけの十分な大きさの文字列を指すポインタ。

bufferが NULL ポインタの場合は、getcwdはmallocを使用してsizeバイトの領域を取得します。この場合、getcwdから返されたポインタをこの後のfreeの呼び出しで引数として使用することができます。

size

返されるディレクトリ指定の長さ。

. . .

省略可能な引数であり、1 または 0 に設定できます。1 を指定した場合、OpenVMS 形式でディレクトリ指定が返されます。0 を指定した場合は、UNIX 形式でディレクトリ指定 (パス名) が返されます。この引数を指定しないと、getcwdは現在のコマンド言語インタプリタ (CLI) に従ってファイル名を返します。UNIX 形式のディレクトリ指定の詳細については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.4.3 項を参照してください。

Return value

x	ファイル指定を指すポインタ。
NULL	エラーを示します。

getdtablesize

プロセスが同時にオープンできるファイル記述子の総数を取得します。

Format

```
#include <unistd.h>
int getdtablesize (void);
```

Description

getdtablesize関数は、プロセスが同時にオープンできるファイル記述子の総数を返します。各プロセスでオープンできるファイル記述子の数は一定の数に制限されています。

プロセスがオープンできるファイル記述子の数は、次の値の中の最小値です。

- HP C RTL のオープン・ファイル・リミット – OpenVMS Alpha および I64 では 65535 , OpenVMS VAX では 2048。
- SYSGEN CHANNELCNT パラメータ – パーマネント I/O チャンネル・カウント。
- プロセスのオープン・ファイル・クォータ FILLM パラメータ – 一度にプロセスがオープンできるファイルの数。

Return value

x	プロセスが同時にオープンできるファイル記述子の数。
-1	エラーを示します。

getegid

POSIX ID が無効に設定されている場合は、この関数はgetgidと同じであり、ユーザ識別コード (UIC) からグループ番号を返します。

POSIX ID が有効に設定されている場合は、この関数は呼び出し元プロセスの実効グループ ID を返します。

Format

```
#include <unistd.h>

gid_t getegid (void);
```

Description

getegid関数は POSIX 形式の識別子 (ID) または UIC ベースの識別子に対して使用できます。

POSIX 形式の ID は、OpenVMS Version 7.3-2 およびそれ以降でサポートされています。

POSIX 形式の ID が無効に設定されている場合、getegid関数とgetgid関数は同じであり、現在の UIC からグループ番号を返します。たとえば、UIC が[313,031]の場合は、313 がグループ番号です。

POSIX 形式の ID が有効に設定されている場合、getegidは呼び出しプロセスの実効グループ ID を返し、getgidは呼び出しプロセスの実グループ ID を返します。実グループ ID はログイン時に指定されます。実効グループ ID はより一時的なものであり、set-group-IDプロセスの実行時に追加のアクセス許可を決定します。getgid関数が最も役立つのはこのようなプロセスです。

getegid関数は、必ず成功します。エラーを示すために予約されている戻り値はありません。

POSIX 形式の ID を有効または無効にする方法については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.7 節を参照してください。

geteuidおよびgetuidも参照してください。

戻り値

x

実効グループ ID (POSIX ID が有効に設定されている場合)、または UIC から取り出したグループ番号 (POSIX ID が無効に設定されている場合)。

getenv

現在のプロセスの環境配列を検索し、指定された環境名に関連付けられている値を返します。

Format

```
#include <stdlib.h>

char *getenv (const char *name);
```

引数

name

次の値のいずれか。

- HOME — ユーザのログイン・ディレクトリ
- TERM — 使用している端末の種類
- PATH — デフォルトのデバイスとディレクトリ
- USER — プロセスを開始したユーザの名前
- 論理名またはコマンド言語インタプリタ (CLI) シンボル名
- setenvまたはputenvで設定された環境変数

指定するnameの大文字と小文字の区別は重要です。

Description

特定の状況では、getenv関数はユーザ指定引数に対して論理名変換を実行しようとします。

1. getenvに対する引数が環境配列内の環境文字列のいずれとも一致しない場合は、getenvはファイル処理の場合と同様に、LNMS\$FILE_DEV 論理名によって示される論理名テーブルを検索することにより、引数を論理名として変換しようとします。

getenvはまず、大文字と小文字を区別した検索を実行します。その検索が失敗すると、大文字と小文字を区別しない検索を実行します。ほとんどの場合、論理名は大文字で定義されていますが、getenvは小文字を含む論理名も検索することができます。

getenvは繰り返しの論理名変換を実行しません。

2. 論理名が複数の等価名を含む検索リストの場合、返される値は最初の等価名を指します。次の例を参照してください。

```
$ DEFINE A B,C
ptr = getenv("A");
```

Aは"B"を指すポインタを返します。

3. 論理名が存在しない場合、getenvは引数文字列を CLI シンボルとして変換しようとします。この処理が正しく行われると、変換されたシンボル・テキストが返されます。この処理に失敗すると、戻り値は NULL になります。

getenvは繰り返しの CLI 変換を実行しません。

CLI が DEC/Shell の場合は、Shell 環境シンボルは DCL シンボルとして実相されているため、関数は論理名変換を実行しません。

注意

OpenVMS Version 7.1 では、OpenVMS 環境変数 (つまり論理名と DCL シンボル) のキャッシュがgetenv関数に追加され、論理名の変換や、DCL シンボルの値の取得のためにライブラリが呼び出しを繰り返すのを回避するようになりました。デフォルト設定では、キャッシュは無効に設定されています。アプリケーションで、実行中に発生する可能性のある OpenVMS 環境変数の変化を追跡する必要がない場合は、アプリケーションを起動する前に DECC\$ENABLE_GETENV_CACHE 論理名 (等価文字列) を設定することにより、キャッシュを有効にすることができます。

Return value

x	変換後のシンボルを格納した配列を指すポインタ。インデックス 0 に等価名が返されます。
NULL	変換が失敗したことを示します。

geteuid

POSIX ID が無効に設定されている場合、この関数はgetuidと同じであり、ユーザ識別コード (UIC) からメンバ番号 (OpenVMS の用語) を返します。

POSIX ID が有効に設定されている場合は、この関数は実効ユーザ ID を返します。

Format

```
#include <unistd.h>

uid_t geteuid (void);
```

Description

geteuid関数は POSIX 形式の識別子 (ID) または UIC ベースの識別子に対して使用できます。

POSIX 形式の ID は、OpenVMS Version 7.3-2 およびそれ以降でサポートされています。

POSIX 形式の ID が無効に設定されている場合 (デフォルト)、geteuid関数とgetuid関数は同じであり、次に示すように現在の UIC からメンバ番号を返します。

- `_VMS_V6_SOURCE` 機能テスト・マクロを設定してコンパイルされたプログラムや、`<unistd.h>` ヘッダ・ファイルを取り込まないプログラムの場合、getuid関数とgeteuid関数は OpenVMS UIC のメンバ番号を返します。たとえば、UIC が[313,31]の場合、メンバ番号 31 が返されます。
- `_VMS_V6_SOURCE` 機能テスト・マクロを設定せずにコンパイルされ、`<unistd.h>` ヘッダ・ファイルを取り込むプログラムの場合は、完全な UIC が返されます。たとえば、UIC が[313, 31]の場合は、 $20512799 (31 + 313 * 65536)$ が返されます。

POSIX 形式の ID が有効に設定されている場合、geteuidは呼び出しプロセスの実効ユーザ ID を返し、getuidは呼び出しプロセスの実ユーザ ID を返します。

POSIX 形式の ID を有効または無効にする方法については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.7 節を参照してください。

getegidおよびgetgidも参照してください。

戻り値

x

実効ユーザ ID (POSIX ID が有効に設定されている場合)、または現在の UIC から取り出したメンバ番号または完全な UIC (POSIX ID が無効に設定されている場合)。

getgid

POSIX ID が無効に設定されている場合、この関数はgetegidと同じであり、ユーザ識別コード (UIC) からグループ番号を返します。

POSIX ID が有効に設定されている場合は、この関数は実グループ ID を返します。

Format

```
#include <unistd.h>

gid_t getgid (void);
```

Description

getgid関数は POSIX 形式の識別子または UIC ベースの識別子に対して使用できます。

POSIX 形式の ID は、OpenVMS Version 7.3-2 およびそれ以降でサポートされています。

POSIX 形式の ID が無効に設定されている場合 (デフォルト)、getegid関数とgetgid関数は同じであり、現在の UIC からグループ番号を返します。たとえば、UIC が[313,031]の場合、313 がグループ番号です。

POSIX 形式の ID が有効に設定されている場合、getegidは呼び出しプロセスの実効グループ ID を返し、getgidは呼び出しプロセスの実グループ ID を返します。実グループ ID はログイン時に指定されます。実効グループ ID はより一時的なものであり、set-group-IDプロセスの実行時に追加のアクセス許可を決定します。getgid関数が最も役立つのはこのようなプロセスです。

POSIX 形式の ID を有効または無効にする方法については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.7 節を参照してください。

geteuidおよびgetuidも参照してください。

戻り値

x

実グループ ID (POSIX ID が有効に設定されている場合)、または現在の UIC から取り出したグループ番号 (POSIX ID が無効に設定されている場合)。

getgrent (*Alpha, I64*)

グループ・データベースのエントリを取得します。

Format

```
#include <grp.h>
struct group *getgrent (void);
```

Description

getgrent関数は、順次検索における次のグループを返します。グループ・データベース内のエントリから取り出したフィールドが格納された構造体へのポインタが返されます。

最初に呼び出されたとき、getgrentは、グループ・データベース内の1番目のエントリが格納されたgroup構造体へのポインタを返します。それ以後は、グループ・データベース内の次のgroup構造体へのポインタを返します。このため、連続して呼び出すことで、データベース全体を検索できます。

読み取り時に、ファイルの終端またはエラーが検出された場合、getgrentはNULLポインタを返し、errnoを設定します。

Return value

x	成功した場合、group構造体へのポインタです。
---	--------------------------

NULL

エラーが発生したことを示します。この関数は、`errno`に以下のいずれかの値を設定します。

- `EACCES` – ユーザ・プロセスが、ユーザ登録ファイルにアクセスするための適切な特権を持っていません。
- `EINTR` – 操作中に、シグナルをキャッチしました。
- `EIO` – 入出力エラーが発生したことを示します。
- `EMFILE` – 呼び出し元プロセス内で、`OPEN_MAX`個のファイル記述子が現在オープンされています。
- `ENFILE` – 許されている最大個数のファイルが、現在システム内でオープンされています。

getgrgid (Alpha, I64)

グループ ID に対応するグループ・データベース・エントリを取得します。

Format

```
#include <types.h>
#include <grp.h>
struct group *getgrgid (gid_t gid);
```

引数

gid
グループ・データベース・エントリを取り出すグループのグループ ID。

Description

getgrgid関数は、グループ・データベースでgidが一致するエントリを検索します。そして、一致するエントリが格納されたgroup構造体へのポインタを返します。

Return value

x 一致するエントリが格納された、有効なgroup構造体へのポインタです。

NULL

エラーが発生しました。

注意: 戻り値は、以後のgetgrent, getgrgid, またはgetgrnamの呼び出しで上書きされる静的領域を指しています。

エラーが発生すると、この関数は、errnoに以下のいずれかの値を設定します。

- EACCES – ユーザ・プロセスが、ユーザ登録ファイルにアクセスするための適切な特権を持っていません。
- EIO – 入出力エラーが発生しました。
- EINTR – getgrgidの実行中にシグナルをキャッチしました。
- EMFILE – 呼び出し元プロセス内で、OPEN_MAX個のファイル記述子が現在オープンされています。
- ENFILE – 許されている最大個数のファイルが、現在システム内でオープンされています。

エラー状態をチェックするアプリケーションは、getgrgidを呼び出す前に、errnoに0を設定する必要があります。戻り時にerrnoが設定されている場合、エラーが発生しています。

getgrgid_r (Alpha, I64)

グループ ID に対応するグループ・データベース・エントリを取得します。

Format

```
#include <types.h>
#include <grp.h>

int getgrgid_r (gid_t gid, struct group *grp, char *buffer, size_t bufsz, struct group **result);
```

Argument

gid
グループ・データベース・エントリを取り出すグループのグループ ID。

grp
取り出したgroup構造体を保持する記憶域。

buffer
データベース内の最長のグループ・エントリを保持できる作業バッファ。

bufsz
bufferの長さ (文字数)。

result
成功して戻った場合、resultは取り出したgroup構造体を指します。

失敗して戻った場合、resultには NULL が設定されます。

Description

getgrgid_r関数は、grpが指すgroup構造体をアップデートし、resultが指すメモリ位置に、この構造体へのポインタを格納します。この構造体には、gidが一致したグループ・データベースのエントリが格納されています。group構造体から指す記憶域は、buffer引数のメモリ (サイズはbufsz文字) から割り当てられます。このバッファに必要なサイズは、sysconf関数の_SC_GETGR_R_SIZE_MAX パラメータで調べることができます。エラーの場合、または要求されたエントリが見つからない場合は、resultが指すメモリ位置に、NULL ポインタが返されます。

Return value

0	成功を示します。
x	<p>エラーの場合、この関数は以下のいずれかの値を戻り値として設定します。</p> <ul style="list-style-type: none">• EACCES – ユーザ・プロセスが、ユーザ登録ファイルにアクセスするための適切な特権を持っていません。• EIO – 入出力エラーが発生しました。• EINTR – getgrgidの実行中にシグナルをキャッチしました。• EMFILE – 呼び出し元プロセス内で、OPEN_MAX個のファイル記述子が現在オープンされています。• ENFILE – 許されている最大個数のファイルが、現在システム内でオープンされています。• ERANGE – buffer引数とbufsize引数で指定された記憶域は、得られたgroup構造体から指すデータを格納するには不十分です。

getgrnam (Alpha, I64)

名前に対応するグループ・データベース・エントリを取得します。

Format

```
#include <types.h>
#include <grp.h>
struct group *getgrnam (const char *name);
```

引数

name
グループ・データベース・エントリを取り出すグループのグループ名。

Description

getgrnam関数は、グループ・データベースでnameが一致するエントリを検索します。そして、一致したエントリが格納されたgroup構造体へのポインタを返します。

Return value

x	一致したエントリが格納された、有効なgroup構造体へのポインタです。
---	-------------------------------------

NULL

エラーを示します。

注意: 戻り値は、以後のgetgrent, getgrgid, またはgetgrnamの呼び出しで上書きされる静的領域を指しています。

エラーが発生すると、この関数は、errnoに以下のいずれかの値を設定します。

- EACCES – ユーザ・プロセスが、ユーザ登録ファイルにアクセスするための適切な特権を持っていません。
- EIO – 入出力エラーが発生しました。
- EINTR – getgrnamの実行中にシグナルをキャッチしました。
- EMFILE – 呼び出し元プロセス内で、OPEN_MAX個のファイル記述子が現在オープンされています。
- ENFILE – 許されている最大個数のファイルが、現在システム内でオープンされています。

エラー状態をチェックするアプリケーションは、getgrnamを呼び出す前に、errnoに0を設定する必要があります。戻り時にerrnoが設定されている場合、エラーが発生しています。

getgrnam_r (Alpha, I64)

名前に対応するグループ・データベース・エントリを取得します。

Format

```
#include <types.h>
#include <grp.h>
int getgrnam_r (const char *name, struct group *grp, char *buffer, size_t bufsz, struct group **result);
```

Argument

name
グループ・データベース・エントリを取り出すグループのグループ名。

grp
取り出したgroup構造体を保持する記憶域。

buffer
データベース内の最長のグループ・エントリを保持できる作業バッファ。

bufsz
bufferの長さ (文字数)。

result
成功して戻った場合、resultは取り出したgroup構造体を指します。

失敗して戻った場合、resultには NULL が設定されます。

Description

getgrnam_r関数は、grpが指すgroup構造体をアップデートし、resultが指すメモリ位置に、この構造体へのポインタを格納します。この構造体には、nameが一致したグループ・データベースのエントリが格納されています。group構造体から指す記憶域は、buffer引数のメモリ (サイズはbufsz文字) から割り当てられます。このバッファに必要なサイズは、sysconf関数の_SC_GETGR_R_SIZE_MAX パラメータで調べることができます。エラーの場合、または要求されたエントリが見つからない場合は、resultが指すメモリ位置に、NULL ポインタが返されます。

Return value

0	成功を示します。
x	<p>エラーの場合、この関数は以下のいずれかの値を戻り値として設定します。</p> <ul style="list-style-type: none">• EACCES – ユーザ・プロセスが、ユーザ登録ファイルにアクセスするための適切な特権を持っていません。• EIO – 入出力エラーが発生しました。• EINTR – getgrnamの実行中にシグナルをキャッチしました。• EMFILE – 呼び出し元プロセス内で、OPEN_MAX個のファイル記述子が現在オープンされています。• ENFILE – 許されている最大個数のファイルが、現在システム内でオープンされています。• ERANGE – buffer引数とbufsize引数で指定された記憶域は、得られたgroup構造体から指すデータを格納するには不十分です。

getgroups

呼び出しプロセスの現在の補助グループ ID リストを取得します。

Format

```
#include <unistd.h>

int getgroups (int gidsetsize, gid_t grouplist[]);
```

Argument

gidsetsize

grouplistパラメータの指す配列に格納可能な、最大エントリ数。

grouplist

取得した補助グループ ID の格納先となる配列。getgroups関数から呼び出しプロセスの実効グループ ID が返されるのは、その実効グループ ID が呼び出しプロセスの補助グループ ID にもなっている場合だけです。

Description

getgroups関数は、呼び出しプロセスの現在の補助グループ ID リストを取得します。このリストは、grouplistパラメータが指している配列に格納して返されます。gidsetsizeパラメータは、この配列に格納できるエントリの数を示します。

getgroups関数が返す ID の数は、sysconfパラメータ_SC_NGROUPS_MAX で指定されている値以下です。

getgidとsetgidも参照してください。

戻り値

n

成功したことを示します。n は、grouplist パラメータが指す配列に格納して返されたエレメントの数です。

−1

失敗したことを示します。errnoには、次のいずれかの値が設定されます。

- EFAULT — gidsetsizeパラメータと grouplistパラメータで指定した配列の一部または全体が、そのプロセスに割り当てられているアドレス空間に収まっていませんでした。
- EINVAL —gidsetsizeパラメータの値はゼロではありませんでしたが、補助グループ ID の数より小さい値でした。

getitimer

インターバル・タイマの値を返します。

Format

```
#include <time.h>

int getitimer (int which, struct itimerval *value);
```

Argument

- which**
インターバル・タイマの種類。HP C RTL では ITIMER_REAL だけがサポートされます。
- value**
itimerval構造体を指すポインタ。この構造体のメンバはタイマのインターバルおよびインターバルが終了するまでの時間を指定します。

Description

getitimer関数は、valueによって示される構造体のwhich引数によって指定されるタイマの現在の値を返します。

タイマ値はitimerval構造体によって定義されます。

```
struct itimerval {
    struct timeval it_interval;
    struct timeval it_value;
};
```

次の表はitimerval構造体のメンバの値を示しています。

itimerval メンバの値	意味
it_interval = 0	it_valueが0以外の値であるものとして、タイマが次に満了した後、タイマを無効にする。
it_interval = 0 以外の値	タイマの満了時にit_valueの再ロードで使用する値を指定する。
it_value = 0	タイマを無効にする。
it_value = 0 以外の値	タイマが次に満了するまでの時間を示す。

システム・クロックの分解能より小さい時間値は、この分解能になるように切り上げられます。

HP C RTL は各プロセスに 1 つのインターバル・タイマを提供します。このタイマは `<time.h>` ヘッダ・ファイルに `ITIMER_REAL` として定義されています。このタイマはリアルタイムで減分され、タイマの満了時に `SIGALRM` シグナルを配布します。

Return value

0	正常終了を示します。
-1	エラーを示します。errno は <code>EINVAL</code> に設定されます (value 引数に取り扱うことができない大きな時間値が指定されました)。

getlogin

ログイン名を取得します。

Format

```
#include <unistd.h>

char *getlogin (void);

int *getlogin_r (char *name, size_t namesize);
```

Description

getlogin関数は、現在のセッションに関連付けられているユーザのログイン名を返します。同じユーザ ID にログイン名が複数個あっても、getloginからポインタを通して返されるログイン名は、そのユーザがログインしたときに使用したログイン名です。ただし、このことがいえるのは、返されたポインタの値が NULL でない場合だけです。

getlogin_r関数は、getloginのリエントラント版です。処理に成功すると、getlogin_rは、ログイン動作によってその呼び出しプロセスの制御端末に関連づけられている名前をnameが指す文字配列に置いて、戻り値 0 を返します。この配列は長さがnamesize文字ですが、その長さには、名前と終了文字 (NULL) を格納できるだけの余裕が必要です。ログイン名の最長サイズは、LOGIN_NAME_MAX です。

同じユーザ ID にログイン名が複数個ある場合にgetlogin_rを実行して成功すると、nameの指す場所に、そのユーザがログインで使用した名前が格納されて返されます。

Return value

x	getloginが成功したことを示します。getloginは、静的バッファに null で終了する文字列、つまりログイン名を置いて、そのバッファを指すポインタを返します。
0	getlogin_rが成功したことを示します。
NULL	エラーが発生したことを示します。errnoが設定されます。

getname

ファイル記述子に関連付けられているファイル指定を返します。

Format

```
#include <unixio.h>

char *getname (int file_desc, char *buffer, ... );
```

関数バリエーション

getname関数には、_getname32および_getname64という名前のバリエーションがあり、それぞれ 32 ビット・ポインタ・サイズおよび 64 ビット・ポインタ・サイズで使用されます。ポインタ・サイズ固有の関数の使い方については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

Argument

file_desc

ファイル記述子。

buffer

ファイル指定を格納できるだけの十分な大きさの文字列を指すポインタ。

...

省略可能な引数であり、1 または 0 に設定できます。1 を指定した場合は、getname関数は OpenVMS 形式でファイル指定を返します。0 を指定した場合は、UNIX 形式でファイル指定を返します。この引数を省略すると、現在のコマンド言語インタプリタ (CLI) に従ってファイル名を返します。UNIX 形式のファイル指定の詳細については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.4.3 項を参照してください。

Description

getname関数は、bufferによって示される領域にファイル指定を格納し、そのアドレスを返します。bufferによって示される領域は、完全に修飾したファイル指定 (最大長は 256 文字) を格納できるだけの十分な大きさの配列でなければなりません。

Return value

x	buffer引数に渡されたアドレス。
0	エラーを示します。

getopt

UNIX コマンド・ラインの規則に従うアプリケーションで利用できるコマンド・ライン・パーサ。

Format

```
#include <unistd.h> (X/Open, POSIX-1)
#include <stdio.h> (X/Open, POSIX-2)
int getopt (int argc, char * const argv[], const char *optstring);
extern char *optarg;
extern int optind, opterr, optopt;
```

Argument

argc

mainに渡される引数の数。

argv

mainに渡される引数配列。

optstring

認識されるオプション文字で構成される文字列。文字の後にコロンが続く場合、そのオプションは引数を受け付けます。

Description

変数optindは、処理するargvベクタの次の要素のインデックスです。この変数はシステムで1に初期化され、argvの各要素の処理が終了したきに、getoptによって更新されます。argvの1つの要素に複数のオプション文字が含まれている場合は、どのオプションが処理済みかをgetoptがどのように判断するかは不定です。

getopt関数は、argvのオプション文字のうち、optstringの文字と一致する次のオプション文字(そのような文字が見つかった場合)を返します。オプションが引数を受け付ける場合は、getoptは変数optargを、次に示すようにオプション引数を指すポインタに設定します。

- オプションがargvの要素によって示される文字列内の最後の文字である場合は、optargにはargvの次の要素が格納され、optindに2が加算されます。optindの結果値がargcより小さくない場合は、getoptはエラーを返し、オプション引数が不足していることを示します。
- それ以外の場合、optargは、argvのその要素内のオプション文字の後の文字列を指し、optindに1を加算します。

次のいずれかの条件が満たされる場合、getoptはoptindを変更せずに、-1を返します。

argv[optind]がNULLポインタである
 *argv[optind]が文字-でない
 argv[optind]が文字列"- "を示している

argv[optind]が文字列"- "を指す場合、getoptは、optindを増分した後、-1を返します。

getoptがoptstringに含まれていないオプション文字を検出すると、疑問符(?)を返します。

getoptが不足している引数を検出した場合、optstringの1文字目がコロンのときは、コロン文字(:)を返します。それ以外の場合は、疑問符を返します。

上記の2つの場合、getoptは変数optoptを、エラーの原因となったオプション文字に設定します。アプリケーションで変数opterrを0に設定しておらず、optstringの1文字目がコロンでない場合は、getoptは診断メッセージをstderrにプリントします。

Return value

x	コマンド・ラインに指定した次のオプション文字。 getoptが不足している引数を検出し、optstringの1文字目がコロンの場合は、コロンを返します。 getoptがoptstringにないオプション文字を検出した場合や、不足している引数を検出し、optstringの1文字目がコロンでない場合は、疑問符を返します。
-1	すべてのコマンド・ライン・オプションが解析された場合。

例

次の例は、ユーティリティに対する引数の処理方法を示しています。このユーティリティは、同時に組み合わせて指定することができないオプションaとb、およびオプションfとoを受け付けることができ、この2つのオプションはいずれも引数を必要とします。

```
#include <unistd.h>

int main (int argc, char *argv[ ])
{
    int c;
    int bflg, aflg, errflg;
    char *ifile;
    char *ofile;
    extern char *optarg;
    extern int optind, optopt;
    .
    .
    .
    while ((c = getopt(argc, argv, ":abf:o:")) != -1) {
        switch (c) {
            case 'a':
                if (bflg)
                    errflg++;
                else
                    aflg++;
                break;
            case 'b':
                if (aflg)
                    errflg++;
                else {
                    bflg++;
                    bproc();
                }
        }
    }
}
```

```

        break;
    case 'f':
        ifile = optarg;
        break;
    case 'o':
        ofile = optarg;
        break;
    case ':': /* -f or -o without operand */
        fprintf (stderr,
            "Option -%c requires an operand\n" optopt);
        errflg++;
        break;
    case '?':
        fprintf (stderr,
            "Unrecognized option -%c\n" optopt);
        errflg++;
    }
}
if (errflg) {
    fprintf (stderr, "usage: ...");
    exit(2);
}
for ( ; optind < argc; optind++) {
    if (access(argv[optind], R_OK)) {
        .
        .
        .
    }
}

```

このサンプル・コードは、以下のいずれも等価なものとして受け付けます。

```

cmd -ao arg path path
cmd -a -o arg path path
cmd -o arg -a path path
cmd -a -o arg -- path path
cmd -a -oarg path path
cmd -aoarg path path

```

getpagesize

システム・ページ・サイズを取得します。

Format

```
#include <unistd.h>

int getpagesize (void);
```

Description

getpagesize関数は、ページ内のバイト数を返します。システム・ページ・サイズは、メモリ管理システム呼び出しに対する引数を指定するときに便利です。

ページ・サイズはシステム・ページ・サイズであり、必ずしもハードウェア・ページ・サイズと同一ではありません。

戻り値

x	常に正常終了を示します。ページ内のバイト数を返します。
---	-----------------------------

getpgid (Alpha, I64)

プロセスに対応するプロセス・グループ ID を取得します。

Format

```
#include <unistd.h>

pid_t getpgid (pid_t pid);
```

引数

pid
グループ ID を要求するプロセスのプロセス ID。

Description

getpgid関数は、pidで指定されたプロセスのプロセス・グループ ID を返します。pidが 0 の場合、getpgid関数は、呼び出し元プロセスのプロセス・グループ ID を返します。

Return value

x	指定されたプロセスのセッション・リーダのプロセス・グループ ID です。
(pid_t)-1	エラーを示します。この関数は、errnoに以下のいずれかの値を設定します。 <ul style="list-style-type: none">• EPERM – pidで指定されたプロセスが、呼び出し元プロセスと同じセッションに属していないため、この実装では、呼び出し元プロセスからこのプロセスのプロセス・グループ ID へのアクセスが許されていません。• ESRCH – プロセス ID がpidであるようなプロセスはありません。• EINVAL – pidの値が不正です。

getpgrp (Alpha, I64)

呼び出し元プロセスのプロセス・グループ ID を取得します。

Format

```
#include <unistd.h>
pid_t getpgrp (void);
```

Description

getpgrp関数は、呼び出し元プロセスのプロセス・グループ ID を返します。

getpgrp関数は必ず成功します。エラーを示すために予約されている戻り値はありません。

Return value

x	呼び出し元プロセスのプロセス・グループ ID です。
---	----------------------------

getpid

現在のプロセスのプロセス ID を返します。

Format

```
#include <unistd.h>
pid_t getpid (void);
```

戻り値

x	現在のプロセスのプロセス ID。
---	------------------

getppid

呼び出しプロセスの親プロセス ID を返します。

Format

```
#include <unistd.h>
pid_t getppid (void);
```

Return value

x	親プロセス ID。
0	呼び出しプロセスに親プロセスがないことを示します。

getpwent

ユーザ・データベース内のユーザ・エントリ情報にアクセスします。passwd構造体へのポインタが返されます。

Format

```
#include <pwd.h>

struct passwd *getpwent (void);
```

関数バリエント

getpwent関数には、__getpwent32および__getpwent64という名前のバリエントがあり、それぞれ 32 ビット・ポインタ・サイズおよび 64 ビット・ポインタ・サイズで使われます。ポインタ・サイズ固有の関数の使い方の詳細については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル(上巻)』第 1.10 節を参照してください。

Description

getpwent関数は、ユーザ・データベースのエントリから得た値がフィールドに格納された構造体へのポインタを返します。データベースのエントリは、getpwentによって順にアクセスされます。最初に呼び出されたとき、getpwentは、ユーザ・データベース内の 1 番目のエントリが格納されたpasswd構造体へのポインタを返します。その後の呼び出しでは、この関数は、ユーザ・データベース内の次のエントリが格納されたpasswd構造体へのポインタを返します。このため、連続して呼び出しを行うことで、ユーザ・データベース全体を検索できます。

passwd構造体は、<pwd.h>ヘッダ・ファイルに次のように定義されています。

pw_name	ユーザの名前
pw_uid	ユーザの ID
pw_gid	ユーザの基本グループのグループ ID
pw_dir	ユーザのホーム・ディレクトリ
pw_shell	ユーザの初期プログラム

読み取り時に、ファイルの終端またはエラーが検出された場合、getpwentは NULL ポインタを返します。

getpwentはユーザ登録ファイル (SYSUAF) を直接アクセスするため、プロセスには適切な特権が必要です。特権がないと、この関数は失敗します。

注意

getpwent関数で生成される情報はすべて、スレッド単位の静的領域に格納され、関数の次の呼び出しで上書きされます。

パスワード・ファイル・エントリが長すぎる場合は無視されます。

Return value

x	成功した場合、passwd構造体へのポインタです。
NULL	ファイルの終端またはエラーの発生を示します。この関数は、errnoに以下のいずれかの値を設定します。 <ul style="list-style-type: none">• EIO – 入出力エラーの発生、またはユーザ登録ファイル (SYSUAF) にアクセスするための適切な特権がユーザにないことを示します。• EMFILE – 呼び出し元プロセス内で、OPEN_MAX個のファイル記述子が現在オープンされています。• ENFILE – 許されている最大個数のファイルが、現在システム内でオープンされています。

getpwnam, getpwnam_r

getpwnam関数は、指定されたnameのユーザ・データベース・エントリについての情報を返します。

getpwnam_r関数は、getpwnamのリエントラント・バージョンです。

Format

```
#include <pwd.h>

struct passwd *getpwnam (const char *name); (ISO POSIX-1)
struct passwd *getpwnam (const char *name, ...); (HP C Extension)
int getpwnam_r (const char *name, struct passwd *pwd, char *buffer, size_t bufsz, struct passwd **result); (ISO POSIX-1), (Alpha, I64)
int getpwnam_r (const char *name, struct passwd *pwd, char *buffer, size_t bufsz, struct passwd **result, ...); (HP C Extension), (Alpha, I64)
```

関数バリエーション

getpwnam関数とgetpwnam_r関数には、__getpwnam32, getpwnam_r32, __getpwnam64, __getpwnam_r64という名前のバリエーションがあり、それぞれ32ビット・ポインタ・サイズおよび64ビット・ポインタ・サイズで使用されます。ポインタ・サイズ固有の関数の使い方の詳細については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

Argument

name

属性を読み込むユーザの名前。

pwd

この関数が結果を格納するpasswd構造体のアドレス。

buffer

passwd構造体の最長のエントリを保持できる、result引数用の作業バッファ。passwd構造体から指す記憶域は、buffer引数のメモリ (長さはbufsize文字) から割り当てられます。

bufsize

bufferが指す文字配列の長さ。

result

成功して戻った場合、pwdが設定されます。失敗して戻った場合、result には NULL が設定されます。

...

1 または 0 となるオプションの引数。1 を指定すると、ディレクトリ指定は、OpenVMS 形式で返されます。0 を指定すると、ディレクトリ指定 (パス名) は、UNIX 形式で返されます。この引数を省略すると、この関数は現在のコマンド言語インタプリタに従って、ディレクトリ指定を返します。UNIX 形式のディレクトリ指定についての詳細は、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.4.3 項を参照してください。

Description

getpwnam関数は、ユーザ・データベース内で指定されたnameのエントリを検索します。この関数は、データベース内のユーザ・エントリのうち、passwd構造体のpw_nameメンバがname引数と一致する最初のエントリを返します。

passwd構造体は、次に示すように<pwd.h>ヘッダ・ファイルに定義されています。

pw_name	ユーザのログイン名
pw_uid	数値ユーザ ID
pw_gid	数値グループ ID
pw_dir	ユーザのホーム・ディレクトリ
pw_shell	ユーザの初期プログラム

注意

getpwnam関数で生成された情報はすべて、スレッドごとの静的領域に格納され、この関数の次の呼び出しで上書きされます。

getpwnam_r関数は、getpwnamのリエントラント・バージョンです。getpwnam_r関数は、pwdが指すpasswd構造体をアップデートし、resultが指すメモリ位置にこの構造体へのポインタを格納します。この構造体には、指定されたnameと一致するユーザ・データベースのエントリが格納されます。この構造体から指す記憶域は、buffer引数のメモリ (長さはbufsize文字) から割り当てられます。このバッファに必要なサイズは、sysconf関数の_SC_GETPW_R_SIZE_MAXパラメータで調べることができます。エラーの場合、または要求されたエントリが見つからない場合は、resultが指すメモリ位置に、NULL ポインタが返されます。

エラー状態をチェックするアプリケーションは、getpwnamを呼び出す前に、errnoに 0 を設定する必要があります。getpwnamが NULL ポインタを返し、errnoがゼロでない場合、エラーが発生しています。

Return value

x	一致するエントリが見つかった場合、getpwnamは正しいpasswd構造体へのポインタを返します。
NULL	<p>エラーが発生した場合、または指定されたエントリが見つからなかった場合、getpwnamはNULLを返します。errnoには、エラーを示す値が設定されます。getpwnam関数は、以下の場合に失敗します。</p> <ul style="list-style-type: none"> • EIO – 入出力エラーが発生しました。 • EINTR – getpwnamの実行中にシグナルをキャッチしました。 • EMFILE – 呼び出し元プロセス内で、OPEN_MAX個のファイル記述子が現在オープンされています。 • ENFILE – 許されている最大個数のファイルが、現在システム内でオープンされています。
0	成功した場合、getpwnam_rは0を返し、resultが指すメモリ位置に、アップデートしたpasswd構造体へのポインタを格納します。
0	<p>失敗した場合 (エラー、または要求されたエントリが見つからない場合)、getpwnam_rは0を返し、resultが指すメモリ位置に、NULLポインタを格納します。getpwnam_r関数は、次の場合に失敗します。</p> <ul style="list-style-type: none"> • ERANGE – bufferとbufsizeで指定された記憶域は、得られたpasswd構造体から指すデータを格納するには不十分です。

getpwuid, getpwuid_r (Alpha, I64)

getpwuid関数は、指定されたuidのユーザ・データベース・エントリについての情報を返します。

getpwuid_r関数は、getpwuidのリエントラント・バージョンです。

Format

```
#include <pwd.h>

struct passwd *getpwuid (uid_t uid); (ISO POSIX-1)
struct passwd *getpwuid (uid_t uid, ... ); (HP C Extension)
int getpwuid_r (uid_t uid, struct passwd *pwd, char *buffer, size_t bufsize, struct passwd **result); (ISO
POSIX-1)
int getpwuid_r (uid_t uid, struct passwd *pwd, char *buffer, size_t bufsize, struct passwd **result, ... );
(HP C Extension)
```

関数バリエーション

getpwuid関数とgetpwuid_r関数には、__getpwuid32, __getpwuid_r32, __getpwuid64, __getpwuid_r64という名前のバリエーションがあり、それぞれ32ビット・ポインタ・サイズおよび64ビット・ポインタ・サイズで使用されます。ポインタ・サイズ固有の関数の使い方の詳細については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

Argument

uid

属性を読み込むユーザ ID (UID)。

pwd

取り出したpasswd構造体を格納するメモリ位置。

buffer

passwd構造体内のエントリを保持できる、result引数用の作業バッファ。passwd構造体から指す記憶域は、buffer引数のメモリ (長さはbufsize文字) から割り当てられます。

bufsize

bufferが指す文字配列の長さ。

result

成功して戻った場合、resultにはpwdが設定されます。失敗して戻った場合、resultには NULL が設定されます。

...

1 または 0 となるオプションの引数。1 を指定すると、ディレクトリ指定は、OpenVMS 形式で返されます。0 を指定すると、ディレクトリ指定 (パス名) は、UNIX 形式で返されます。この引数を省略すると、この関数は現在のコマンド言語インタプリタに従って、ディレクトリ指定を返します。UNIX 形式のディレクトリ指定についての詳細は、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.4.3 項を参照してください。

Description

getpwuid関数は、ユーザ・データベース内で、指定されたuidのエントリを検索します。この関数は、passwd構造体のpw_uidメンバとuid引数が一致する、データベース内の最初のユーザ・エントリを返します。

passwd構造体は、<pwd.h>ヘッダ・ファイルに次のように定義されています。

pw_name	ユーザのログイン名
pw_uid	数値ユーザ ID
pw_gid	数値グループ ID
pw_dir	ユーザのホーム・ディレクトリ
pw_shell	ユーザの初期プログラム

注意

getpwuid関数で生成されたすべての情報は、スレッド単位の静的領域に格納され、この関数の次の呼び出しで上書きされます。

getpwuid_r関数は、getpwuidのリエントラント・バージョンです。getpwuid_r関数は、pwdが指す passwd構造体をアップデートし、resultが指すメモリ位置にこの構造体へのポインタを格納します。この構造体には、uidが一致するユーザ・データベースのエントリが格納されます。この構造体から指す記憶域は、buffer引数のメモリ (サイズはbufsize文字) から割り当てられます。このバッファに必要なサイズは、sysconf関数の_SC_GETGR_R_SIZE_MAX パラメータで調べることができます。エラーの場合、または要求されたエントリが見つからない場合は、resultが指すメモリ位置に、NULL ポインタが返されます。

エラー状態をチェックするアプリケーションは、getpwuidを呼び出す前に、errnoに 0 を設定する必要があります。getpwuidが NULL ポインタを返し、errnoがゼロでない場合、エラーが発生しています。

Return value

x	一致するエントリが見つかった場合、getpwuidは正しいpasswd構造体へのポインタを返します。
NULL	<p>エラーが発生した場合、または指定されたエントリが見つからなかった場合、getpwuidはNULLを返します。errnoには、エラーを示す値が設定されます。getpwuid関数は、以下の場合に失敗します。</p> <ul style="list-style-type: none"> • EIO – 入出力エラーが発生しました。 • EINTR – getpwuidの実行中にシグナルをキャッチしました。 • EMFILE – 呼び出し元プロセス内で、OPEN_MAX個のファイル記述子が現在オープンされています。 • ENFILE – 許されている最大個数のファイルが、現在システム内でオープンされています。
0	成功した場合、getpwuid_rは0を返し、resultが指すメモリ位置に、アップデートしたpasswd構造体へのポインタを格納します。
0	<p>失敗した場合 (エラー、または要求されたエントリが見つからない場合)、getpwuid_rは0を返し、resultが指すメモリ位置に、NULLポインタを格納します。getpwuid_r関数は、次の場合に失敗します。</p> <ul style="list-style-type: none"> • ERANGE – bufferとbufsizeで指定された記憶域は、得られたpasswd構造体から指すデータを格納するには不十分です。

gets

標準入力 (stdin) から 1 行を読み込みます。

Format

```
#include <stdio.h>

char *gets (char *str);
```

関数バリエント

gets関数には、_gets32および_gets64という名前のバリエントがあり、それぞれ 32 ビット・ポインタ・サイズおよび 64 ビット・ポインタ・サイズで使用されます。ポインタ・サイズ固有の関数の使い方の詳細については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

引数

str
stdinから読み込んだ情報を格納できるだけの十分な大きさの文字列を指すポインタ。

Description

行末の改行文字(\n)は、この関数で ASCII のヌル文字(\0)に置換されます。

stdinがレコード・モードでオープンされている場合は、getsはレコードの終端を改行文字と同じように取り扱うので、読み込みは改行文字まで、またはレコードの終端まで行われます。

Return value

x	str引数を指すポインタ。
NULL	エラーが発生したか、または改行文字が検出される前にファイルの終端 (EOF) が検出されたことを示します。読み込みエラーが発生した場合、strの内容は未定義です。

getsid (Alpha, I64)

セッション・リーダーのプロセス・グループ ID を取得します。

Format

```
#include <unistd.h>

pid_t getsid (pid_t pid);
```

引数

pid
セッション・リーダーのプロセス・グループ ID を要求するプロセスのプロセス ID です。

Description

getsid関数は、pidで指定されたプロセスのセッション・リーダーであるプロセスのプロセス・グループ ID を取得します。pidが (pid_t)0 の場合は、呼び出し元プロセスが指定されます。

Return value

- | | |
|-----------|---|
| x | 指定されたプロセスのセッション・リーダーのプロセス・グループ ID です。 |
| (pid_t)-1 | エラーを示します。この関数は、errnoに以下のいずれかの値を設定します。 <ul style="list-style-type: none">• EPERM – pidで指定されたプロセスが、呼び出し元プロセスと同じセッションに属していないため、この実装では、呼び出し元プロセスからこのプロセスのセッション・リーダーのプロセス・グループ ID へのアクセスが許されていません。• ESRCH – プロセス ID がpidであるようなプロセスはありません。 |

[w]getstr

端末画面から文字列を取得し、その文字列をstr変数に格納し、指定のウィンドウに表示します。getstr関数はstdscrウィンドウに対して動作します。

Format

```
#include <curses.h>

int getstr (char *str);

int wgetstr (WINDOW *win, char *str);
```

Argument

win
ウィンドウを指すポインタ。

str
ウィンドウから読み込んだ文字列を格納できる十分な大きさでなければなりません。

Description

getstr関数とwgetstr関数は、文字列を読み込む前に指定されたウィンドウの表示を更新します。読み込んだ文字列から、末尾の改行文字は削除されます。詳細については、scrollok関数を参照してください。

Return value

OK	正常終了を示します。
ERR	関数が画面を不正にスクロールしたことを示します。

gettimeofday

日付と時刻を取得します。

Format

```
#include <time.h>

int gettimeofday (struct timeval *tp, void *tzp);
```

Argument

tp
timeval構造体を指すポインタ。この構造体は<time.h>ヘッダ・ファイルに定義されています。

tzp
NULL ポインタ。この引数が NULL ポインタでない場合は無視されます。

Description

gettimeofday関数は、1970年1月1日 UTC (協定世界時) 00::00 からの経過時間として、現在の時刻 (秒数とマイクロ秒数) を取得します。現在の時刻は、tp引数によって示されるtimeval構造体に格納されます。

tzp引数は、カーネルによって設定されたタイム・ゾーン情報を格納する目的で使用されます。しかし、OpenVMS カーネルはタイム・ゾーン情報を設定しないので、tzp引数は NULL でなければなりません。NULL でない場合は無視されます。この関数は BSD プログラムとの互換性を維持するためにサポートされます。

SYS\$TIMEZONE_DIFFERENTIAL 論理名の値が不正な場合は、この関数は異常終了し、errnoはEINVAL に設定されます。

Return value

0	正常終了を示します。
-1	エラーが発生したことを示します。errnoはエラーを示すように設定されます。

getuid

POSIX ID が無効に設定されている場合，この関数はgeteuidと同じであり，ユーザ識別コード (UIC) からメンバ番号 (OpenVMS の用語) を返します。

POSIX ID が有効に設定されている場合は，実ユーザ ID を返します。

Format

```
#include <unistd.h>

uid_t getuid (void);
```

Description

getuid関数は POSIX 形式の識別子または UIC ベースの識別子に対して使用できます。

POSIX 形式の ID は，OpenVMS Version 7.3-2 およびそれ以降でサポートされています。

POSIX 形式の ID が無効に設定されている場合 (デフォルト)，geteuid関数とgetuid関数は同じであり，次に示すように現在の UIC からメンバ番号を返します。

- `_VMS_V6_SOURCE` 機能テスト・マクロを設定してコンパイルされたプログラムや，`<unistd.h>` ヘッダ・ファイルを取り込まないプログラムの場合，getuid関数とgeteuid関数は OpenVMS UIC のメンバ番号を返します。たとえば，UIC が[313,31]の場合，メンバ番号 31 を返します。
- `_VMS_V6_SOURCE` 機能テスト・マクロを設定せずにコンパイルされたプログラムで，`<unistd.h>` ヘッダ・ファイルを取り込むプログラムの場合は，完全な UIC を返します。たとえば，UIC が[313, 31]の場合は，20512799 ($31 + 313 * 65536$) を返します。

POSIX 形式の ID が有効に設定されている場合，geteuidは呼び出しプロセスの実効ユーザ ID を返し，getuidは呼び出しプロセスの実ユーザ ID を返します。

POSIX 形式の ID を有効または無効にする方法については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.7 節を参照してください。

getegidおよびgetgidも参照してください。

getuid

戻り値

x

実ユーザ ID (POSIX ID が有効に設定されている場合) ,
または現在の UIC から取り出したメンバ番号または完全
な UIC (POSIX ID が無効に設定されている場合)。

getw

指定されたファイルから文字を返します。

Format

```
#include <stdio.h>
int getw (FILE *file_ptr);
```

引数

`file_ptr`
アクセスするファイルを指すポインタ。

Description

`getw`関数は、`int`として指定された入力ファイルから次の 4 文字を返します。

Return value

<code>x</code>	<code>int</code> 内の次の 4 文字。
<code>EOF</code>	4 文字のいずれかを取得しているときにファイルの終端 (EOF) が検出され、4 文字すべてが消失したことを示します。EOF は受け付け可能な整数であるため、関数が正常終了したかどうか確認するには、 <code>feof</code> と <code>ferror</code> を使用します。

getwc

指定されたファイルから次の文字を読み込み、ワイド文字コードに変換します。

Format

```
#include <wchar.h>

wint_t getwc (FILE *file_ptr);
```

引数

file_ptr
アクセスするファイルを指すポインタ。

Description

getwcはマクロとして実相されているため、副作用のあるファイル・ポインタ引数 (たとえばgetwc (*f++)) は正しく評価されないことがあります。このような場合は、代わりにfgetwc関数を使用してください。fgetwc関数を参照してください。

Return value

n	返された文字。
WEOF	ファイルの終端 (EOF) またはエラーを示します。エラーが発生した場合、この関数はerrnoを設定します。この関数が設定する値の一覧については、fgetwcを参照してください。

getwchar

1 文字のワイド文字を標準入力 (stdin) から読み込みます。

Format

```
#include <wchar.h>
wint_t getwchar (void);
```

Description

getwchar関数はfgetwc(stdin) と同じです。

Return value

x	stdinから読み込み，wint_tに変換した次の文字。
WEOF	ファイルの終端 (EOF) またはエラーを示します。エラーが発生した場合，この関数はerrnoを設定します。この関数が設定する値の一覧については，fgetwcを参照してください。

getyx

win上での現在のカーソルの位置の (y,x) 座標を変数yおよびxに格納します。

Format

```
#include <curses.h>
getyx (WINDOW *win, int y, int x);
```

Argument

win

ウィンドウを指すポインタ。

y

有効な lvalue でなければなりません。

x

有効な lvalue でなければなりません。

glob (Alpha, I64)

パターンに一致するパス名を生成します。

Format

```
#include <glob.h>

int glob (const char *pattern, int flags, int (*errfunc)(const char *epath, int eerrno), glob_t *pglob);
```

関数バリエーション

glob関数には、_glob32および_glob64というバリエーションがあり、それぞれ 32 ビットと 64 ビットのポインタ・サイズで使います。ポインタ・サイズ固有の関数の使用方法については、第 1.10 節を参照してください。

Argument

pattern

アクセス可能なパス名と比較するファイル名パターン。

flags

glob関数でカスタマイズ可能な動作を制御します。

errfunc

glob関数がエラー状態を検出した際に呼び出される関数 (オプション)。

pglob

glob_t構造体へのポインタ。この構造体は、呼び出し元が割り当てます。この構造体の配列には、glob関数がpattern引数と一致するファイル名を検索して格納します。最後のエントリは、NULL です。

glob_t構造体は、<glob.h>ヘッダ・ファイルで定義されており、少なくとも次のメンバを含んでいます。

```
size_t  gl_pathc    //Count of paths matched by pattern.
char ** gl_pathv    //Pointer to a list of matched pathnames.
size_t  gl_offs     //Slots to reserve at the beginning of gl_pathv.
```

epath

ディレクトリがオープンできないか読み込めないために失敗したパス名。

errno

epath引数で指定されるパス名に関する失敗で、opendir関数、readdir関数、またはstat関数が設定したerrno値。

Description

glob関数は、pattern引数に一致する、アクセス可能なファイルのリストを作成します。

glob関数は、UNIX モードと OpenVMS モードのどちらかのモードで動作します。

DECC\$GLOB_UNIX_STYLE 機能論理名を有効にすることで、UNIX モードを明示的に選択できます。このモードは、デフォルトでは無効になっています。

glob関数は、以下のいずれの状態でもなければ、デフォルトで OpenVMS モードを使用します。以下のいずれかの状態の場合、globは UNIX モードを使用します。

1. DECC\$GLOB_UNIX_STYLE が有効である。
2. DECC\$FILENAME_UNIX_ONLY 機能論理名が有効である。
3. glob関数が、指定されたパス名指示パターン (ディレクトリ区切り文字など) をチェックし、UNIX 形式のパス名であると判定した。

OpenVMS モード

このモードを使用すると、OpenVMS のプログラマは、OpenVMS 形式のパターンをglob関数に渡し、期待した OpenVMS 形式の出力を得ることができます。OpenVMS 形式のパターンは、DCL コマンドでユーザが受け取ったり、sys\$sparse システム・ルーチンおよび sys\$search システム・ルーチンへの入力として使用するパターンです。

このモードでは、アスタリスク(*)やパーセント(%)などの、任意の OpenVMS ワイルドカードを使用できます。つまり、OpenVMS システム・コールがサポートしているワイルドカードすべてを使用できます。

OpenVMS モードは、UNIX のワイルドカード、?や[]のパターン・マッチングをサポートしていません。OpenVMS ユーザは、[]を、ディレクトリ区切り文字として使用します。

他にも、OpenVMS モードと UNIX モードの動作には、相違点があります。

- OpenVMS モードでは、UNIX モードのような相対ファイル指定ではなく、完全なファイル指定が出力されます。

- OpenVMS モードでは、GLOB_MARK フラグは無視されます。これは、OpenVMS ではディレクトリにスラッシュ(/)を付加しても意味がないためです。

例:

Sample pattern input	Sample output
[.SUBDIR1]A.TXT	DEV: [DIR.SUBDIR1]A.TXT;1
[.SUB*]*.*	DEV: [DIR.SUBDIR1]A.TXT;1

UNIX モード

次のコマンドで、このモードを明示的に有効にできます。

```
$ DEFINE DECC$GLOB_UNIX_STYLE ENABLE
```

UNIX モードは、DECC\$FILENAME_UNIX_ONLY 機能論理名が設定されている場合や、指定されたパターンが UNIX 形式のパス名であるとglob関数が判断した場合にも、有効になります。

UNIX モードでは、glob関数は、以下の拡張ファイル・システム (EFS) 上の制限を満たす範囲で、可能なかぎり X/Open 仕様に従います。

- ファイル名に大文字と小文字が混在していても、表示上の意味しかありません。名前が "a" と "A" のファイルは、同じファイルです。ファイル名は、最初に作成されたときのとおりに大文字小文字を区別して表示されます。
- 次の文字を、ファイル名に含めることはできません。

? *

例:

Sample pattern input	Sample output
./a/b/c	./a/b/c
./?/b/*	./a/b/c
[a-c]	c

共通の説明

glob関数は、アクセス可能なすべてのパス名をパターンと比較し、一致するすべてのパス名のリストを作成します。パス名にアクセスするためには、glob関数は、最後以外の各パス名コンポーネントに対する検索パーミッションと、pattern引数の各ファイル名コンポーネントの各ディレクトリに対する読み込みパーミッションを必要とします。

glob関数は、一致したパス名の数と、パス名へのポインタのリストへのポインタをpglob引数に格納します。パス名は、現在のロケールの LC_COLLATE カテゴリの設定に基づいてソートされます。最後のパス名の後の最初のポインタは、NULL で

す。どのパス名ともパターンが一致しない場合、一致したパス名の数として 0 が返されます。

pglob 引数が指す構造体は、呼び出し元が作成しなければなりません。glob 関数は、必要に応じて他の領域も割り当てます。globfree 関数は、以前の glob 関数呼び出しの結果として pglob 引数に関連して割り当てられた領域を解放します。

flags 引数は、glob 関数の動作を制御するために使用されます。flags 値は、以下の定数の、ビット単位の論理和です。これらの定数は、<glob.h> ヘッダ・ファイルで定義されています。

GLOB_APPEND	以前に検索されたパス名に、この呼び出しで検索されたパス名を付加します。
GLOB_DOOFFS	gl_offs メンバを使用して、pglob 引数の gl_pathv コンポーネントの始めに追加する NULL ポインタの数を指定します。
GLOB_ERR	オープンや読み込みができないディレクトリを検出した場合に glob 関数が戻るようにします。GLOB_ERR フラグが設定されていない場合、glob 関数は、オープンや読み込みができないディレクトリを検出しても、検索を続けます。
GLOB_MARK	ディレクトリである各パス名にスラッシュ(/)を付加する必要があることを指定します。OpenVMS モードでは、GLOB_MARK は無視されます。これは、OpenVMS システムではディレクトリにスラッシュを付加しても意味がないためです。
GLOB_NOCHECK	pattern 引数がどのパス名とも一致しなかった場合、glob 関数は、pattern 引数だけからなるリストを返し、一致したパス名の数を 1 とします。
GLOB_NOESCAPE	GLOB_NOESCAPE フラグを設定した場合、バックスラッシュ(\)は、メタキャラクタをエスケープするためには使用できません。

GLOB_APPEND フラグを使用すると、glob 関数の以前の呼び出しで見つかったパス名に、新しいパス名のセットを追加できます。以下の規則は、同じ pglob 引数値で glob 関数への呼び出しを複数回行い、その間に globfree 関数を呼び出さない場合に適用されます。

- アプリケーションが glob 関数の 1 回目の呼び出しで GLOB_DOOFFS フラグを設定した場合、このフラグは 2 回目の呼び出しでも設定され、pglob 引数の gl_offs フィールド値は、これらの呼び出しの間で変更されません。
- アプリケーションが glob 関数の 1 回目の呼び出しで GLOB_DOOFFS フラグを設定しなかった場合、このフラグは 2 回目の呼び出しでも設定されません。
- 2 回目の呼び出しの後、pglob->gl_pathv は、次の情報を含むリストを指します。
 - GLOB_DOOFFS フラグと pglob->gl_offs で指定された、ゼロ個以上の NULL。
 - 呼び出し前に pglob->gl_pathv リストにあった、パス名へのポインタ。ポインタの順序は、glob 関数の 1 回目の呼び出し後と同じです。
 - 2 回目の呼び出しで作成された新しいパス名へのポインタ (指定された順序)。

- pglob->gl_offs 引数で返されるカウントは、2つの呼び出しで返されるパス名の合計数です。
- アプリケーションは、2つの呼び出しの間で、pglob->gl_pathc フィールドや pglob->gl_pathv フィールドを変更してはなりません。

成功して完了すると、glob 関数は値 0 (ゼロ) を返します。pglob->gl_pathc フィールドは、一致したパス名の数を返します。pglob->gl_pathv フィールドには、一致したパス名をソートした、NULL で終了するリストへのポインタが格納されています。pglob->gl_pathc 内の、一致したパス名の数が 0 (ゼロ) の場合、pglob->gl_pathv 引数のポインタの値は定義されていません。

glob 関数がエラーで終了した場合、この関数は、ゼロ以外の定数 GLOB_ABORTED、GLOB_NOMATCH、または GLOB_NOSPACE (<glob.h> ヘッダ・ファイルに定義されています) のいずれかを返します。この場合、pglob 引数の値は、やはり上記で定義されたとおりに設定されます。

検索中、オープンできないか読み込めないディレクトリを検出し、errfunc 引数値が NULL でなかった場合は、glob 関数は 2 つの引数 epath と errno を指定して errfunc を呼び出します。

epath— ディレクトリがオープンできないか読み込めないために失敗したパス名。
 errno— epath 引数で指定されるパス名に関する失敗で、opendir 関数、readdir 関数、または stat 関数が設定した errno 値。

errfunc を呼び出して、ゼロ以外が返されるか、flags に GLOB_ERR フラグが設定されている場合、glob 関数は、スキャン済みのパス名を反映するために pglob 引数への設定を行った後、スキャンを停止し、GLOB_ABORTED を返します。GLOB_ERR が設定されておらず、errfunc が NULL であるか、errfunc がゼロを返した場合、エラーは無視されます。

errno 値は返されません。

globfree、fnmatch、readdir、および stat も参照してください。

Return value

0	成功を示します。
GLOB_ABORTED	GLOB_ERROR が設定されているか、errfunc がゼロ以外の値を返したため、スキャンが停止しました。
GLOB_NOMATCH	パターンが既存のパス名と一致せず、flags に GLOB_NOCHECK が設定されていませんでした。
GLOB_NOSPACE	メモリを割り当てようとしたが、失敗しました。

globfree

glob関数への以前の呼び出しで pglob引数に関連付けられたスペースを解放します。

Format

```
#include <glob.h>

void globfree (glob_t *pglob);
```

関数バリエーション

globfree関数には、_globfree32および_globfree64というバリエーションがあり、それぞれ 32 ビットおよび 64 ビットのポインタ・サイズで使います。ポインタ・サイズ固有の関数の使用方法については、第 1.10 節を参照してください。

引数

pglob
以前に割り当てられたglob_t構造体へのポインタ。

Description

globfree関数は、glob関数への以前の呼び出しで pglob引数に関連付けられたスペースを解放します。globfree関数には、戻り値はありません。

gmtime, gmtime_r

秒数で表した時間を年月日時分秒形式の UTC 時刻に変換します。

Format

```
#include <time.h>

struct tm *gmtime (const time_t *timer);

struct tm *gmtime_r (const time_t *timer, struct tm *result); (ISO POSIX-1)
```

関数バリエーション

_DECC_V4_SOURCE および _VMS_V6_SOURCE 機能テスト・マクロを定義してコンパイルすると、gmtime_r関数に対して、OpenVMS Version 7.0 より前の動作に相当するローカル時刻ベースのエントリ・ポイントが有効になります。

Argument

timer

Epoch からの経過時間を秒数で指定する変数を指すポインタ。

result

結果が格納されるtm構造体を指すポインタ。

tm構造体は<time.h>ヘッダ・ファイルに定義されており、localtimeの説明の表 REF-4 にも示されています。

Description

gmtime関数とgmtime_r関数は、timerによって示される時刻 (Epoch からの経過時間 (秒数)) を、協定世界時 (UTC) として表した年月日時分秒形式の時刻に変換し、tm構造体に格納します。

gmtime_r関数とgmtime関数の違いは、gmtime_r関数は、ユーザ指定のtm構造体に結果を格納するのに対し、gmtime関数は、HP C RTL で割り当てられたスレッド固有の静的メモリに結果を格納する点です。このため、gmtimeを再び呼び出すと、結果が上書きされます。結果を保存する必要がある場合は、コピーを作成しなければなりません。

gmtimeは、正常終了すると、tm構造体を指すポインタを返します。gmtime_rは2番目の引数を返します。異常終了すると、これらの関数はNULLポインタを返します。

注意

一般に、UTCベースの時刻関数は、プロセス単位のリソースであるメモリ内のタイム・ゾーン情報に影響を与える可能性があります。しかし、アプリケーションの実行中、システム・タイム・ゾーンが変化せず(これは一般的なケースです)、タイム・ゾーン・ファイルのキャッシュが有効に設定されている場合(これはデフォルトです)、時刻関数asctime_r, ctime_r, gmtime_r, localtime_rの_rバリエーションはスレッド・セーフで、かつASTリentrantです。

しかし、アプリケーションの実行中にシステム・タイム・ゾーンが変化する可能性がある場合や、タイム・ゾーン・ファイルのキャッシュが有効に設定されていない場合は、UTCベースの時刻関数のバリエーションはいずれも3番目の関数クラスに属し、これはスレッド・セーフでもASTリentrantでもありません。

Return value

x	tm構造体を指すポインタ。
NULL	エラーを示します。errnoは次の値に設定されます。 <ul style="list-style-type: none"> EINVAL—timer引数がNULLです。

gsignal

指定されたソフトウェア・シグナルを生成します。このシグナルは、`signal`、`ssignal`、`sigvec`関数によって設定されたアクション・ルーチンを起動します。

Format

```
#include <signal.h>

int gsignal (int sig [, int sigcode]);
```

Argument

sig
生成するシグナル。

sigcode
省略可能なシグナル・コード。たとえば、シグナル SIGFPE (算術演算トラップ・シグナル) には 10 種類のコードがあり、各コードは異なる種類の算術演算トラップを表します。

シグナル・コードはニーモニックまたは数字で表現できます。算術演算トラップ・コードは 1 ~ 10 の数字で表現されますが、SIGILL コードは 0 ~ 2 の数字で表現されます。コード値は<signal.h>ヘッダ・ファイルに定義されています。シグナルのニーモニック、コード、対応する OpenVMS 例外の一覧については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』表 4-4 および表 4-5 を参照してください。

Description

gsignal関数を呼び出すと、次のいずれかの結果になります。

- gsignalが<signal.h>ヘッダ・ファイルに定義されている範囲外のsig引数を指定する場合は、gsignalは 0 を返し、errnoを EINVAL に設定します。
- signal、ssignal、sigvecがシグナルに対して SIG_DFL (デフォルト・アクション) を設定する場合は、gsignalは呼び出し元に戻りません。イメージは終了し、シグナルに対応するOpenVMS エラー・コードが設定されます。
- signal、ssignal、sigvecがシグナルのアクションとして SIG_IGN (シグナルの無視) を設定した場合は、gsignalはその引数sigを返します。

- シグナルに対してアクション・ルーチンを設定するには、`signal`、`ssignal`、`sigvec`のいずれかを使用しなければなりません。その関数が呼び出され、戻り値が`gsignal`から返されます。

詳細については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 4 章を参照してください。

`raise`、`signal`、`ssignal`、`sigvec`も参照してください。

Return value

0	sig 引数が <code><signal.h></code> ヘッダ・ファイルに定義されている範囲外であることを示します。errno が EINVAL に設定されます。
sig	SIG_IGN (シグナルの無視) がシグナルのアクションとして設定されていることを示します。
x	signal、ssignal、sigvec がシグナルに対してアクション関数を設定したことを示します。その関数が呼び出され、gsignal から戻り値が返されます。

hypot

直角三角形の斜辺の長さを返します。

Format

```
#include <math.h>

double hypot (double x, double y);
float hypotf (float x, float y); (Alpha, I64)
long double hypotl (long double x, long double y); (Alpha, I64)
```

Argument

x
実数値。

y
実数値。

Description

hypot関数は直角三角形の斜辺の長さを返します。ただし、xとyは三角形の垂直な辺を表します。長さは次の公式で計算されます。

$$\sqrt{x^2 + y^2}$$

オーバーフローが発生すると、戻り値は未定義になり、errnoはERANGEに設定されます。

Return value

x	斜辺の長さ。
HUGE_VAL	オーバーフローが発生したことを示します。errnoはERANGEに設定されます。
0	アンダフローが発生したことを示します。errnoはERANGEに設定されます。
NaN	xまたはyがNaNです。errnoはEDOMに設定されません。

iconv

あるコードセットでコーディングされている文字を別のコードセットでコーディングされた文字に変換します。

Format

```
#include <iconv.h>

size_t iconv (iconv_t cd, const char **inbuf, size_t *inbytesleft, char **outbuf, size_t *outbytesleft);
```

Argument

cd

変換記述子。iconv_openの呼び出しが正常終了すると、この値が返されます。

inbuf

入力バッファの1文字目を指す変数を指すポインタ。

inbytesleft

この引数の初期値は、入力バッファ (inbuf) の末尾までのバイト数を示す変数を指すポインタです。変換が完了すると、この変数はinbufで変換されなかったバイト数を示します。

outbuf

出力バッファの最初の使用可能なバイトを指す変数を指すポインタ。出力バッファには変換後の文字が格納されます。

outbytesleft

この引数の初期値は、出力バッファ (outbuf) の末尾までのバイト数を示す変数を指すポインタです。変換が完了すると、この変数はoutbufに残されているバイト数を示します。

Description

iconv関数は、inbufで示されるバッファ内の文字を別のコードセットの文字に変換します。変換後の文字はoutbufで示されるバッファに格納されます。変換のタイプは変換記述子cdによって指定します。iconv_openの呼び出しが正常終了すると、この記述子が返されます。

入力バッファから不正な文字が検出されると、有効な最後の文字まで変換した後、変換は停止します。inbytesleftによって示される変数は、変換されなかった入力バッファ内のバイト数を表すように更新されます。outbytesleftによって示される変数は、出力バッファに残されているバイト数を示すように更新されます。

Return value

x

実行された同一でない変換の数。正常終了した変換を示します。ほとんどの場合、0が返されます。

(size_t) -1

エラー条件を示します。errnoは次のいずれかに設定されます。

- EBADF—cd引数が有効な変換記述子ではありません。
- EILSEQ—不正な文字が検出されたため、変換が停止しました。
- E2BIG—出力バッファの領域が不足しているため、変換が停止しました。
- EINVAL—入力バッファの末尾に不完全な文字があるため、変換が停止しました。

iconv_close

指定された変換記述子と、その記述子に割り当てられているリソースの割り当てを解除します。

Format

```
#include <iconv.h>

int iconv_close (iconv_t cd);
```

引数

cd

割り当てを解除する変換記述子。iconv_openの呼び出しが正常終了すると、変換記述子が返されます。

Return value

- | | |
|----|--|
| 0 | 変換記述子の割り当てが正しく解除されたことを示します。 |
| -1 | エラーが発生したことを示します。errnoは次のいずれかに設定されます。 <ul style="list-style-type: none">• EBADF—cd引数が有効な変換記述子ではありません。• EVMSERR—変換不可能な OpenVMS エラーが発生しました。vaxc\$errnoには VMS エラー・コードが格納されます。 |

iconv_open

指定されたコードセット変換のために変換記述子を割り当てます。

Format

```
#include <iconv.h>

iconv_t iconv_open (const char *tocode, const char *fromcode);
```

Argument

tocode

変換後の文字のコードセットの名前。

fromcode

変換前のコードセットの名前。現在使用可能なコードセットの一覧と新しいコードセットの追加方法の詳細については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 10 章を参照してください。

Return value

x

変換記述子。呼び出しが正常終了したことを示します。この記述子はこの後の iconv の呼び出しで使用されます。

(iconv_t) -1

エラーが発生したことを示します。errno は次のいずれかに設定されます。

- EMFILE— プロセスにファイルをオープンするための十分な I/O チャンネルがありません。
- ENOMEM— 使用できる領域が不足しています。
- EINVAL— fromcode と tocode によって指定される変換はサポートされません。
- EVMSERR— 変換不可能な OpenVMS エラーが発生しました。vaxc\$errno には OpenVMS エラー・コードが格納されます。vaxc\$errno に SSS_BADCHKSUM という値が格納されている場合は、変換テーブル・ファイルは見つかったものの、テーブルの内容が壊れていることを示します。vaxc\$errno に SSS_IDMISMATCH という値が格納されている場合は、変換テーブル・ファイルのバージョンが C ランタイム・ライブラリのバージョンと一致しないことを示します。

例

```

#include <stdio.h>
#include <iconv.h>
#include <errno.h>

int main()
{
    /* Declare variables to be used */
    char fromcodeset[30];
    char tocodeset[30];
    int iconv_opened;
    iconv_t iconv_struct;      /* Iconv descriptor */
    /* Initialize variables */
    sprintf(fromcodeset, "DECHANYU");
    sprintf(tocodeset, "EUCTW");
    iconv_opened = FALSE;

    /* Attempt to create a conversion descriptor for the */
    /* codesets specified. If the return value from */
    /* iconv_open is -1 then an error has occurred. */
    /* Check the value of errno. */
    if ((iconv_struct = iconv_open(tocodeset, fromcodeset))
        == (iconv_t) - 1) {
        /* Check the value of errno */
        switch (errno) {
            case EMFILE:
            case ENFILE:
                printf("Too many iconv conversion files open\n");
                break;

            case ENOMEM:
                printf("Not enough memory\n");
                break;

            case EINVAL:
                printf("Unsupported conversion\n");
                break;

            default:
                printf("Unexpected error from iconv_open\n");
                break;
        }
    }
    else
        /* Successfully allocated a conversion descriptor */
        iconv_opened = TRUE;

    /* Was a conversion descriptor allocated */
    if (iconv_opened) {

```

```

/* Attempt to deallocate the conversion descriptor. */
/* If iconv_close returns -1 then an error has      */
/* occurred.                                         */
if (iconv_close(iconv_struct) == -1) {
    /* An error occurred. Check the value of errno */
    switch (errno) {
    case EBADF:
        printf("Conversion descriptor is invalid\n");
        break;
    default:
        printf("Unexpected error from iconv_close\n");
        break;
    }
}
return (EXIT_FAILURE);
}

```

ilogb (Alpha, I64)

引数の指数部を返します。

Format

```
#include <math.h>
int ilogb (double x);
int ilogbf (float x);
int ilogbl (long double x);
```

引数

x
実数値。

Description

ilogb関数は、引数xの指数部を返します。正式にいえば、 $\log_r |x|$ の整数部 (符号付き整数) が戻り値になります。ただし、xはゼロでない数値が対象となります。また r は、そのマシンにおける浮動小数点演算の基数で、<float.h>に FLT_RADIX 値として定義されているものが使用されます。

Return value

n	成功したことを示します。n は、xの指数部を示す符号付き整数値です。これらの関数では、対応するlogb関数を呼び出して、その戻り値をint型にキャスト (型変換) した場合と同じ結果が得られます。
---	--

[w]inch

ウィンドウを変更せずに、指定されたウィンドウの現在のカーソルの位置にある文字を返します。inch関数はstdscrウィンドウに対して動作します。

Format

```
#include <curses.h>

char inch();

char winch (WINDOW *win);
```

引数

win
ウィンドウを指すポインタ。

Return value

x	返された文字。
ERR	入力エラーを示します。

index

文字列から 1 文字を検索します。

Format

```
#include <strings.h>
char *index (const char *s, int c);
```

関数バリエント

index関数には、_index32および_index64という名前のバリエントがあり、それぞれ 32 ビット・ポインタ・サイズおよび 64 ビット・ポインタ・サイズで使用されます。ポインタ・サイズ固有の関数の使い方の詳細については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

Argument

s
文字を検索する文字列。

c
検索する目的の文字。

Description

index関数はstrchr関数と同じであり、一部の UNIX 実装との互換性を維持するために提供されます。

initscr

端末タイプのデータおよびすべての画面関数を初期化します。Curses 関数を使用する場合は、その前にinitscrを呼び出す必要があります。

Format

```
#include <curses.h>

void initscr (void);
```

Description

initscr関数の OpenVMS Curses バージョンは、初期化を行う前に画面をクリアします。BSD ベースの Curses バージョンは画面をクリアしません。

initstate

乱数ジェネレータを初期化します。

Format

```
#include <stdlib.h>

char *initstate (unsigned int seed, char *state, int size);
```

Argument

seed

初期シード値。

state

状態情報の配列を指すポインタ。

size

状態情報配列のサイズ。

Description

initstate関数は乱数ジェネレータを初期化します。この関数を使用すると、将来使用するために、引数として渡された状態配列を初期化できます。状態配列のサイズ(バイト数)は、使用する乱数ジェネレータがどの程度精巧であるかを判断するために、initstate関数で使用されます。状態配列が大きくなればなるほど、乱数はよりランダムになります。

状態情報のサイズは8, 32, 64, 128, 256 バイトのいずれかです。8 バイトより小さいサイズを指定すると、エラーになります。ここに示した値以外の場合は、これらの値の中で最も近い値になるように切り捨てられます。

seed引数は乱数シーケンスの開始ポイントを指定し、同じポイントで再起動するために提供されます。initstate関数は、前の状態情報配列を指すポインタを返します。

状態を初期化した後、setstate関数を使用して状態を迅速に切り換えることができます。state引数によって定義される配列は、initstate関数が呼び出されるか、setstate関数が再び呼び出されるまで、この後の乱数の生成で使用されません。setstate関数は前の状態配列を指すポインタを返します。

初期化した後、状態配列は次のいずれかの方法で異なるポイントから再開することができます。

- seed引数、state配列、配列のsizeを適切に設定して、initstate関数を使用する方法。
- 適切な状態を設定してsetstate関数を使用した後、適切なseedを指定してsrandom関数を使用する方法。2つの関数を使用すると、状態配列を初期化した後、状態配列のサイズを保存する必要がないという利点があります。

setstate、srandom、randomも参照してください。

Return value

x	前の状態配列情報を指すポインタ。
0	エラーを示します。8バイト未満の状態情報を指定して関数を呼び出しました。詳細なエラーはグローバルなerrnoに指定されます。

[w]insch

指定されたウィンドウの現在のカーソルの位置に 1 文字を挿入します。insch関数はstdscrウィンドウに対して動作します。

Format

```
#include <curses.h>

int insch (char ch);

int winsch (WINDOW *win, char ch);
```

Argument

win
ウィンドウを指すポインタ。

ch
挿入する文字。

Description

文字を挿入した後、その行の各文字は右に移動し、行末の文字は削除されます。詳細については、scrollok関数を参照してください。

Return value

OK	正常終了を示します。
ERR	関数が画面を不正にスクロールしたことを示します。

[w]insertln

現在カーソルが設定されている行の上に 1 行を挿入します。insertln関数はstdscrウィンドウに対して動作します。

Format

```
#include <curses.h>

int insertln();

int wininsertln (WINDOW *win);
```

引数

win
ウィンドウを指すポインタ。

Description

現在の行とその行の下各行は下に移動し、一番下の行は消去されます。挿入される行は空白行で、現在の座標 (y,x) は変化しません。詳細については、scrolllok関数を参照してください。

Return value

OK	正常終了を示します。
ERR	関数が画面を不正にスクロールしたことを示します。

[w]insstr

指定されたウィンドウの現在のカーソルの位置に文字列を挿入します。insstr関数はstdscrウィンドウに対して動作します。

Format

```
#include <curses.h>

int insstr (char *str);

int winsstr (WINDOW *win, char *str);
```

Argument

win
ウィンドウを指すポインタ。

str
挿入する文字列を指すポインタ。

Description

挿入した文字列の後の各文字は右に移動し、最後の文字は消去されます。これらの関数はHP C for OpenVMSシステム固有であり、移植できません。

Return value

OK	正常終了を示します。
ERR	関数が画面を不正にスクロールしたことを示します。詳細については、このセクションのscrolllok関数を参照してください。

isalnum

プログラムの現在のロケールで、文字が英字または数字として分類されるかどうかを示します。

Format

```
#include <ctype.h>

int isalnum (int character);
```

引数

character

int型のオブジェクト。characterの値はunsigned charとして表現できるか、マクロEOFの値に等しくなければなりません。他の値の場合は、動作は未定義です。

Return value

0 以外の値

英数字の場合。

0

英数字でない場合。

isalpha

プログラムの現在のロケールで、文字が英字として分類されるかどうかを示します。

Format

```
#include <ctype.h>
int isalpha (int character);
```

引数

character

int型のオブジェクト。characterの値はunsigned charとして表現できるか、またはマクロ EOF の値に等しくなければなりません。他の値の場合は、動作は未定義です。

Return value

0 以外の値	英字の場合。
0	英字でない場合。

isapipe

指定されたファイル記述子がパイプに関連付けられているかどうかを示します。

Format

```
#include <unixio.h>

int isapipe (int file_desc);
```

引数

file_desc
ファイル記述子。

Description

パイプの詳細については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 5 章を参照してください。

Return value

1	パイプに関連付けられていることを示します。
0	パイプに関連付けられていないことを示します。
-1	エラーを示します (たとえば、ファイル記述子がオープンされているファイルに関連付けられていない場合など)。

isascii

文字が ASCII 文字であるかどうかを示します。

Format

```
#include <ctype.h>
int isascii (int character);
```

引数

character
char型のオブジェクト。

Return value

0 以外の値	ASCII 文字の場合。
0	ASCII 文字でない場合。

isatty

指定されたファイル記述子が端末に関連付けられているかどうかを示します。

Format

```
#include <unistd.h>
int isatty (int file_desc);
```

引数

file_desc
ファイル記述子。

Return value

1	ファイル記述子が端末に関連付けられている場合。
0	ファイル記述子が端末に関連付けられていない場合。
-1	エラーを示します (たとえば、ファイル記述子がオープンされているファイルに関連付けられていない場合など)。

isctrnl

プログラムの現在のロケールで、文字が制御文字として分類されるかどうかを示します。

Format

```
#include <ctype.h>

int isctrnl (int character);
```

引数

character

int型のオブジェクト。characterの値はunsigned charとして表現できるか、またはマクロ EOF の値に等しくなければなりません。他の値の場合は、動作は未定義です。

Return value

0 以外の値	制御文字の場合。
0	制御文字でない場合。

isdigit

プログラムの現在のロケールで、文字が数字として分類されるかどうかを示します。

Format

```
#include <ctype.h>
int isdigit (int character);
```

引数

character

int型のオブジェクト。characterの値はunsigned charとして表現できるか、マクロEOF の値に等しくなければなりません。他の値の場合は、動作は未定義です。

Return value

0 以外の値

10 進数の場合。

0

10 進数でない場合。

isgraph

プログラムの現在のロケールで、文字がグラフィック文字として分類されるかどうかを示します。

Format

```
#include <ctype.h>

int isgraph (int character);
```

引数

character

int型のオブジェクト。characterの値はunsigned charとして表現できるか、マクロEOFの値に等しくなければなりません。他の値の場合は、動作は未定義です。

Return value

0 以外の値	グラフィック文字の場合。
0	グラフィック文字でない場合。

islower

プログラムの現在のロケールで、文字が小文字として分類されるかどうかを示します。

Format

```
#include <ctype.h>

int islower (int character);
```

引数

character

int型のオブジェクト。characterの値はunsigned charとして表現できるか、マクロEOFの値に等しくなければなりません。他の値の場合は、動作は未定義です。

Return value

0 以外の値

小文字の英字の場合。

0

小文字の英字でない場合。

isnan (Alpha, I64)

NaN かどうかを判定します。引数が NaN の場合は 1 を返し , NaN でない場合は 0 を返します。

Format

```
#include <math.h>

int isnan (double x);
int isnanf (float x);
int isnanl (long double x);
```

引数

x
実数値。

Description

isnan関数は , xが NaN (not-a-number 値として予約されている IEEE 浮動小数点数値) の場合は整数値 1 (TRUE) を返し , それ以外の場合は 0 (FALSE) を返します。

isprint

プログラムの現在のロケールで、文字がプリント文字として分類されるかどうかを示します。

Format

```
#include <ctype.h>

int isprint (int character);
```

引数

character

int型のオブジェクト。characterの値はunsigned charとして表現できるか、マクロEOFの値に等しくなければなりません。他の値の場合は、動作は未定義です。

Return value

0 以外の値	プリント文字の場合。
0	プリント文字でない場合。

ispunct

プログラムの現在のロケールで、文字が句読点文字として分類されるかどうかを示します。

Format

```
#include <ctype.h>

int ispunct (int character);
```

引数

character

int型のオブジェクト。characterの値はunsigned charとして表現できるか、マクロEOFの値に等しくなければなりません。他の値の場合は、動作は未定義です。

Return value

0 以外の値

句読点文字の場合。

0

句読点文字でない場合。

isspace

プログラムの現在のロケールで、文字が空白として分類されるかどうかを示します。つまり、ASCII のスペース、タブ (水平または垂直)、キャリッジ・リターン、フォーム・フィード、改行文字のいずれかであるかどうかを示します。

Format

```
#include <ctype.h>

int isspace (int character);
```

引数

character

int型のオブジェクト。characterの値はunsigned charとして表現できるか、マクロEOF の値に等しくなければなりません。他の値の場合は、動作は未定義です。

Return value

0 以外の値

空白文字の場合。

0

空白文字でない場合。

isupper

プログラムの現在のロケールで、文字が大文字として分類されるかどうかを示します。

Format

```
#include <ctype.h>

int isupper (int character);
```

引数

character

int型のオブジェクト。characterの値はunsigned charとして表現できるか、マクロEOFの値に等しくなければなりません。他の値の場合は、動作は未定義です。

Return value

0 以外の値

大文字の英字の場合。

0

大文字の英字でない場合。

iswalnum

プログラムの現在のロケールで、ワイド文字が英字または数字として分類されるかどうかを示します。

Format

```
#include <wctype.h> (ISO C)
#include <wchar.h> (XPG4)
int iswalnum (wint_t wc);
```

引数

wc
wint_t型のオブジェクト。wcの値は現在のロケールでwchar_tとして表現できるか、またはマクロ WEOF の値に等しくなければなりません。他の値の場合は、動作は未定義です。

Return value

0 以外の値	英数字の場合。
0	英数字でない場合。

iswalpha

プログラムの現在のロケールで、ワイド文字が英字として分類されるかどうかを示します。

Format

```
#include <wctype.h> (ISO C)
#include <wchar.h> (XPG4)
int iswalpha (wint_t wc);
```

引数

wc

wint_t型のオブジェクト。wcの値は現在のロケールでwchar_tとして表現できるか、またはマクロ WEOF の値に等しくなければなりません。他の値の場合は、動作は未定義です。

Return value

0 以外の値	英字の場合。
0	英字でない場合。

iswcntrl

プログラムの現在のロケールで、ワイド文字が制御文字として分類されるかどうかを示します。

Format

```
#include <wctype.h> (ISO C)
#include <wchar.h> (XPG4)
int iswcntrl (wint_t wc);
```

引数

wc

wint_t型のオブジェクト。wcの値は現在のロケールでwchar_tとして表現できるか、またはマクロ WEOF の値に等しくなければなりません。他の値の場合は、動作は未定義です。

Return value

0 以外の値	制御文字の場合。
0	制御文字でない場合。

iswctype

ワイド文字が指定されたプロパティを持つかどうかを示します。

Format

```
#include <wctype.h> (ISO C)
#include <wchar.h> (XPG4)

int iswctype (wint_t wc, wctype_t wc_prop);
```

Argument

wc

wint_t型のオブジェクト。wcの値は現在のロケールで有効なワイド文字コードとして表現できるか、またはマクロ WEOF の値に等しくなければなりません。他の値の場合は、動作は未定義です。

wc_prop

現在のロケールで有効なプロパティ名。この名前はwctype関数を呼び出すことにより設定されます。

Description

iswctype関数は、wcに文字クラス・プロパティwc_propが割り当てられているかどうかを判定します。wc_propは、wctype関数を呼び出すことにより設定します。

wctypeも参照してください。

Return value

0 以外の値

文字にプロパティwc_propが割り当てられている場合。

0

文字にプロパティwc_propが割り当てられていない場合。

例

```

#include <locale.h>
#include <wchar.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

/* This test will set up the "upper" character class using      */
/* wctype() and then verify whether the characters 'a' and 'A'   */
/* are members of this class                                     */
#include <stdlib.h>

main()
{
    wchar_t w_char1,
            w_char2;
    wctype_t ret_val;

    char *char1 = "a";
    char *char2 = "A";

    ret_val = wctype("upper");

    /* Convert char1 to wide-character format - w_char1 */
    if (mbtowc(&w_char1, char1, 1) == -1) {
        perror("mbtowc");
        exit(EXIT_FAILURE);
    }

    if (iswctype((wint_t) w_char1, ret_val))
        printf("[%C] is a member of the character class upper\n",
               w_char1);
    else
        printf("[%C] is not a member of the character class upper\n",
               w_char1);

    /* Convert char2 to wide-character format - w_char2 */
    if (mbtowc(&w_char2, char2, 1) == -1) {
        perror("mbtowc");
        exit(EXIT_FAILURE);
    }

    if (iswctype((wint_t) w_char2, ret_val))
        printf("[%C] is a member of the character class upper\n",
               w_char2);
    else
        printf("[%C] is not a member of the character class upper\n",
               w_char2);
}

```

このサンプル・プログラムを実行すると、次の結果が生成されます。

```

[a] is not a member of the character class upper
[A] is a member of the character class upper

```

iswdigit

プログラムの現在のロケールで、ワイド文字が数字として分類されるかどうかを示します。

Format

```
#include <wctype.h> (ISO C)
#include <wchar.h> (XPG4)
int iswdigit (wint_t wc);
```

引数

wc

wint_t型のオブジェクト。wcの値は現在のロケールでwchar_tとして表現できるか、またはマクロ WEOF の値に等しくなければなりません。他の値の場合は、動作は未定義です。

Return value

0 以外の値	10 進数の場合。
0	10 進数でない場合。

iswgraph

プログラムの現在のロケールで、ワイド文字がグラフィック文字として分類されるかどうかを示します。

Format

```
#include <wctype.h> (ISO C)
#include <wchar.h> (XPG4)
int iswgraph (wint_t wc);
```

引数

wc
wint_t型のオブジェクト。wcの値は現在のロケールでwchar_tとして表現できるか、またはマクロ WEOF の値に等しくなければなりません。他の値の場合は、動作は未定義です。

Return value

0 以外の値	グラフィック文字の場合。
0	グラフィック文字でない場合。

iswlower

プログラムの現在のロケールで、ワイド文字が小文字として分類されるかどうかを示します。

Format

```
#include <wctype.h> (ISO C)
#include <wchar.h> (XPG4)
int iswlower (wint_t wc);
```

引数

wc

wint_t型のオブジェクト。wcの値は現在のロケールでwchar_tとして表現できるか、またはマクロ WEOF の値に等しくなければなりません。他の値の場合は、動作は未定義です。

Return value

0 以外の値	小文字の場合。
0	小文字でない場合。

iswprint

プログラムの現在のロケールで、ワイド文字がプリント文字として分類されるかどうかを示します。

Format

```
#include <wctype.h> (ISO C)
#include <wchar.h> (XPG4)
int iswprint (wint_t wc);
```

引数

wc

wint_t型のオブジェクト。wcの値は現在のロケールでwchar_tとして表現できるか、またはマクロ WEOF の値に等しくなければなりません。他の値の場合は、動作は未定義です。

Return value

0 以外の値

プリント文字の場合。

0

プリント文字でない場合。

iswpunct

プログラムの現在のロケールで、ワイド文字が句読点文字として分類されるかどうかを示します。

Format

```
#include <wctype.h> (ISO C)
#include <wchar.h> (XPG4)
int iswpunct (wint_t wc);
```

引数

wc

wint_t型のオブジェクト。wcの値は現在のロケールでwchar_tとして表現できるか、またはマクロ WEOF の値に等しくなければなりません。他の値の場合は、動作は未定義です。

Return value

0 以外の値	句読点文字の場合。
0	句読点文字でない場合。

iswspace

プログラムの現在のロケールで、ワイド文字が空白文字として分類されるかどうかを示します。

Format

```
#include <wctype.h> (ISO C)
#include <wchar.h> (XPG4)
int iswspace (wint_t wc);
```

引数

wc
wint_t型のオブジェクト。wcの値は現在のロケールでwchar_tとして表現できるか、またはマクロ WEOF の値に等しくなければなりません。他の値の場合は、動作は未定義です。

Return value

0 以外の値	空白文字の場合。
0	空白文字でない場合。

iswupper

プログラムの現在のロケールで、ワイド文字が大文字として分類されるかどうかを示します。

Format

```
#include <wctype.h> (ISO C)
#include <wchar.h> (XPG4)
int iswupper (wint_t wc);
```

引数

wc

wint_t型のオブジェクト。wcの値は現在のロケールでwchar_tとして表現できるか、またはマクロ WEOF の値に等しくなければなりません。他の値の場合は、動作は未定義です。

Return value

0 以外の値	大文字の場合。
0	大文字でない場合。

iswxdigit

プログラムの現在のロケールで、ワイド文字が 16 進数 (0 ~ 9 , A ~ F , a ~ f) であるかどうかを示します。

Format

```
#include <wctype.h> (ISO C)
#include <wchar.h> (XPG4)
int iswxdigit (wint_t wc);
```

引数

wc

wint_t 型のオブジェクト。wc の値は現在のロケールで wchar_t として表現できるか、またはマクロ WEOF の値に等しくなければなりません。他の値の場合は、動作は未定義です。

Return value

0 以外の値	16 進数の場合。
0	16 進数でない場合。

isxdigit

プログラムの現在のロケールで、文字が 16 進数 (0 ~ 9 , A ~ F , a ~ f) であるかどうかを示します。

Format

```
#include <ctype.h>

int isxdigit (int character);
```

引数

character

int 型のオブジェクト。character の値は unsigned char として表現できるか、マクロ EOF の値に等しくなければなりません。他の値の場合は、動作は未定義です。

Return value

0 以外の値	16 進数の場合。
0	16 進数でない場合。

j0, j1, jn (Alpha, I64)

第 1 種ベッセル関数を計算します。

Format

```
#include <math.h>
double j0 (double x);
float j0f (float x);
long double j0l (long double x);
double j1 (double x);
float j1f (float x);
long double j1l (long double x);
double jn (int n, double x);
float jnf (int n, float x);
long double jnl (int n, long double x);
```

Argument

x
実数。

n
整数。

Description

j0関数は、0 次の第 1 種ベッセル関数の値を返します。

j1関数は、1 次の第 1 種ベッセル関数の値を返します。

jn関数は、n 次の第 1 種ベッセル関数の値を返します。

j1関数とjn関数は、xの値が小さいと、アンダフローを引き起こすことがあります。
これが起こるxの値の最大値は、nの関数です。

Return value

x	xの第 1 種ベッセル関数の値。
0	x引数の値が大きすぎるか、アンダフローが発生しました。errnoは ERANGE に設定されます。
NaN	xは NaN です。errnoは EDOM に設定されます。

jrand48

一様分布の擬似乱数列を生成します。48 ビットの符号付き long 整数を返します。

Format

```
#include <stdlib.h>

long int jrand48 (unsigned short int xsubi[3]);
```

Argument

xsubi
連結されたときに 48 ビット整数となる、3 つの short int の配列。

Description

jrand48関数は、線形合同法アルゴリズムと 48 ビット整数算術演算を使用して、擬似乱数を生成します。

この関数は、 $-2^{31} \leq y < 2^{31}$ の範囲内で一様に分布する符号付き long 整数を返します。

この関数は、次の線形合同式に従って、48 ビットの整数値 X_i のシーケンスを生成します。

$$X_{n+1} = (aX_n + c) \bmod m \quad n \geq 0$$

引数 m は 2^{48} に等しいので、48 ビット整数算術演算が実行されます。lcong48関数を呼び出さなかった場合、乗数値 a と加算される値 c は次のようになります。

```
a = 5DEECE66D16 = 2736731631558
c = B16 = 138
```

jrand48関数では、呼び出し元プログラムが、xsubi引数として、最初の呼び出しの時点で擬似乱数列の初期値に初期化された配列を渡す必要があります。drand48関数とは異なり、最初の呼び出しの前に初期化関数を呼び出す必要はありません。

jrand48では、異なる引数を使用することで、大きなプログラムの個々のモジュールが、複数の互いに独立した擬似乱数列を生成することができます。たとえば、1 つのモジュールが生成する乱数列は、関数が他のモジュールから呼び出された回数には依存しません。

Return value

n

$-2^{31} \leq y < 2^{31}$ の範囲で一様分布する符号付き long 整数。

kill

プロセス ID によって指定されたプロセスにシグナルを送信します。

Format

```
#include <signal.h>
int kill (int pid, int sig);
```

Argument

pid
プロセス ID。

sig
シグナル・コード。

Description

kill関数は、main関数を含んでいる C および C++ プログラムでのみ使用できます。

kill関数は、プロセスがraiseを呼び出したときと同じように、プロセスに対してシグナルを送信します。シグナルがターゲット・プログラムによってトラップまたは無視されなかった場合、そのプログラムは実行を終了します。

OpenVMS VAX および Alpha は、シグナルの送信先として指定できるプロセスに関して、異なる規則を実装しています。プログラムは、vfork/execによって起動された子プロセスには、つねにシグナルを送信する権限を持っています。その他のプロセスでの結果は、システムの OpenVMS セキュリティ・モデルによって決定されます。

OpenVMS の制約のために、kill関数は、特権付きでインストールされたイメージを実行するターゲット・プロセスにシグナルを配信することはできません。

システム特権がない限り、送信側と受信側のプロセスは、同じ利用者識別コード (UIC) を持っている必要があります。

V7.0 より前の OpenVMS システムでは、killはシグナル値 0 を、SIGKILL が指定された場合と同じように扱います。

OpenVMS Version 7.0 およびそれ以降のシステムでは、`<stdlib.h>`をインクルードし、`_POSIX_EXIT` 機能テスト・マクロを設定してコンパイルすると、次のようになります。

- シグナル値が 0 ならば、kill はプロセス ID を確認するが、シグナルを送信しない。
- プロセス ID が有効でなければ、kill は -1 を返し、`errno`を `ESRCH` に設定する。

Return value

0	kill をキューに入れることに成功したことを示します。
-1	エラーを示します。受信側のプロセスが異なる UIC を持っており、ユーザがシステム・ユーザでないか、受信側のプロセスが存在しません。

l64a (Alpha, I64)

long 整数を, 文字列に変換します。

Format

```
#include <stdlib.h>
char *l64a (long l);
```

引数

l
文字列に変換する long 整数。

Description

a64l関数とl64a関数は, base-64 ASCII 文字として格納された数値を操作するために使用します。

- a64lは, 文字列を long 整数に変換します。
- l64aは, long 整数を文字列に変換します。

long 整数を格納するための各文字は, 0 ~ 63 の数値を表しています。long 整数を表すために, 最大 6 文字を使用できます。

文字は, 次のように変換されます。

- ピリオド(.)は, 0 を表します。
- スラッシュ(/)は, 1 を表します。
- 0 ~ 9 の数字は, 2 ~ 11 を表します。
- 大文字 A ~ Z は, 12 ~ 37 を表します。
- 小文字 a ~ z は, 38 ~ 63 を表します。

l64a関数は, long 整数を受け取り, 下位 32 ビットに対応する base-64 記法文字列へのポインタを返します。

l64aが返す値は, スレッド固有のバッファへのポインタです。そのバッファの内容は, 同じスレッドからの以降の呼び出しで上書きされます。

a64lも参照してください。

戻り値

x

成功した場合、対応する base-64 ASCII 文字列へのポインタです。lパラメータが 0 の場合、l64aは空文字列へのポインタを返します。

labs

整数の絶対値をlong intとして返します。

Format

```
#include <stdlib.h>
long int labs (long int j);
```

Argument

j
long int型の値。

lchown

指定したファイルについて、そのユーザとグループの所有権を変更します。

Format

```
#include <unistd.h>

int lchown (const char *file_path, uid_t file_owner, gid_t file_group);
```

Argument

file_path
オーナーとグループの ID を変更するファイルの名前。

file_owner
ファイルの新しいユーザ ID。

file_group
ファイルの新しいグループ ID。

Description

lchown関数は、指定したファイル (file_path) のオーナーやグループを変更します。ファイルがシンボリック・リンクの場合は、そのシンボリック・リンクのオーナーが変更されます (これとは対照的に、chownでは、そのシンボリック・リンクから参照されているファイルのオーナーが変更されます)。

symlink, unlink, readlink, realpath, およびlstatも参照してください。

Return value

0	成功したことを示します。
-1	エラーが発生したことを示します。errnoには、chownから返された errno値が設定されます。

lcong48

48 ビットの一様分布の擬似乱数列を初期化します。

Format

```
#include <stdlib.h>

void lcong48 (unsigned short int param[7]);
```

Argument

param
初期値 Xi , 乗数値a , および加算される値cを指定する配列。

Description

lcong48関数は、線形合同法アルゴリズムと 48 ビット整数算術演算を使用して、擬似乱数を生成します。lcong48は、以下の関数を呼び出す前に、乱数ジェネレータを初期化する目的に使用することができます。

drand48
lrand48
mrand48

lcong48関数は、初期値 Xi , 乗数値a , および加算される値cを指定します。param配列の各要素は、以下のものを指定します。

param[0-2]	Xi
param[3-5]	乗数値a
param[6]	16 ビットの加算される値c

lcong48が呼び出された後に、srand48またはseed48が呼び出されると、aとcは以前に指定された標準値に復元されます。

lcong48関数は値を返しません。

drand48 , lrand48 , mrand48 , srand48 , seed48も参照してください。

ldexp

第 2 引数を n としたときに , 第 1 引数に 2 の n 乗を掛けた値 , すなわち $x(2^n)$ を返します。

Format

```
#include <math.h>

double ldexp (double x, int n);
float ldexp (float x, int n); (Alpha, I64)
long double ldexp (long double x, int n); (Alpha, I64)
```

Argument

x
 2^n を掛けることになる , double , float , または long double 型の基本値。

n
2 を何乗するかを示す整数の指数値。

Return value

$x(2^n)$	第 2 引数を n としたときに , 第 1 引数に 2 の n 乗を掛けた値。
0	アンダフローが発生しました。errno は ERANGE に設定されます。
HUGE_VAL	オーバフローが発生しました。errno は ERANGE に設定されます。
NaN	x は NaN です。errno は EDOM に設定されます。

ldiv

引数の間で除算を行い，商と剰余を返します。

Format

```
#include <stdlib.h>

ldiv_t ldiv (long int numer, long int denom);
```

Argument

numer
long int型の分子。

denom
long int型の分母。

Description

ldiv_t型は，<stdlib.h>ヘッダ・ファイルに次のように定義されています。

```
typedef struct
{
    long    quot, rem;
} ldiv_t;
```

divも参照してください。

leaveok

Curses に対し，ウィンドウの更新後に，カーソルを現在の座標に配置するよう指示します。

Format

```
#include <curses.h>

leaveok (WINDOW *win, bool boolf);
```

Argument

win

ウィンドウへのポインタ。

boolf

論理型の TRUE または FALSE 値。boolf が TRUE ならば，カーソルは最後の更新後もその位置に留まり，win の座標設定はそれに従って変更されます。boolf が FALSE ならば，カーソルは win の現在指定されている (y,x) 座標に移動します。

Description

leaveok 関数は，デフォルトでは，win の現在座標にカーソルを移動します。bool 型は，<curses.h> ヘッダ・ファイルに次のように定義されています。

```
#define bool int
```

lgamma (Alpha, I64)

ガンマ関数の対数を計算します。

Format

```
#include <math.h>

double lgamma (double x);
float lgammaf (float x);
long double lgammal (long double x);
```

Argument

x
実数。xは0，負の整数，または無限大であってはなりません。

Description

lgamma関数は，xのガンマの絶対値の対数，すなわち G をガンマ関数として， $\ln(|G(x)|)$ を返します。

xのガンマの符号は，外部整変数signgamに返されます。x引数は0，負の整数，または無限大であってはなりません。

Return value

x	x引数のガンマの対数。
-HUGE_VAL	x引数は負の整数です。errnoは ERANGE に設定されます。
NaN	x引数は NaN です。errnoは EDOM に設定されます。
0	アンダフローが発生しました。errnoは ERANGE に設定されます。
HUGE_VAL	オーバーフローが発生しました。errnoは ERANGE に設定されます。

link

既存のファイルへの新しいリンク (ディレクトリ・エントリ) を作成します。この関数は、ハード・リンク・カウントが有効になっているボリューム上でのみサポートされています。

Format

```
#include <unistd.h>

link (const char *path1, const char *path2);
```

Argument

path1
既存のファイルを指定するパス名へのポインタ。

path2
作成する新しいディレクトリ・エントリを指定するパス名へのポインタ。

Description

link関数は、既存のファイルのための新しいリンクをアトミックに作成し、ファイルのリンク・カウントを1だけインクリメントします。

link関数は、ディレクトリ・ファイルに対して使用することができます。

linkが失敗した場合、リンクは作成されず、ファイルのリンク・カウントは変更されません。

Return value

0	実行に成功しました。
---	------------

-1

エラーを示します。この関数は、`errno`を以下のいずれかの値に設定します。

- `EEXIST` – `path2`で指定されたリンクが存在する。
- `EFTYPE` – `path1`または`path2`にワイルドカードが含まれている。
- `EINVAL` – 片方または両方の引数が、構文的に無効なパス名を指定している。
- `ENAMETOOLONG` – `path1`または`path2`の長さが `PATH_MAX` を超えているか、パス名コンポーネントが `NAME_MAX` を超えている。
- `EXDEV` – `path2`によって指定されたリンクと、`path1`によって指定されたファイルは、異なるデバイス上に存在している。

localeconv

struct lconv型の構造体のメンバを、現在のロケールの規則に従って数値の書式を設定するための値に設定します。

Format

```
#include <locale.h>

struct lconv *localeconv (void);
```

Description

localeconv関数は、<locale.h>ヘッダ・ファイルに定義されているlconv構造体へのポインタを返します。この構造体は、プログラムから変更してはなりません。これはlocaleconvの呼び出し、LC_NUMERIC、LC_MONETARY、またはLC_ALLカテゴリを変更するsetlocale関数の呼び出しによって上書きされます。

この構造体のメンバは以下のとおりです。

メンバ	説明
char *decimal_point	基数文字。
char *thousands_sep	桁のグループを区切るために使用される文字。
char *grouping	非金額値の桁をどのようにグループ化するかを定義する文字列。
char *int_curr_symbol	国際通貨シンボル。
char *currency_symbol	ローカル通貨シンボル。
char *mon_decimal_point	金額値の書式に使用される基数文字。
char *mon_thousands_sep	金額値の桁のグループを区切るために使用される文字。
char *mon_grouping	金額値の桁をどのようにグループ化するかを定義する文字列。
char *positive_sign	負でない金額値を示すために使用される文字列。
char *negative_sign	負の金額値を示すために使用される文字列。
char int_frac_digits	国際通貨シンボルでフォーマットされた金額値の中の基数文字の後に表示される桁数。
char frac_digits	金額値の中の基数文字の後に表示される桁数。
char p_cs_precedes	正の金額値で、ローカルまたは国際通貨シンボルが数字の前に置かれる場合には1に、シンボルが数字の後に置かれる場合には0に設定される。

メンバ	説明
char p_sep_by_space	正の金額値で、通貨シンボルと数字の間にスペースがない場合には 0 に設定される。スペースがある場合には 1 に、シンボルと符号文字列の間にスペースがある場合には 2 に設定される。
char n_cs_precedes	負の金額値で、ローカルまたは国際通貨シンボルが数字の前に置かれる場合には 1 に、シンボルが数字の後に置かれる場合には 0 に設定される。
char n_sep_by_space	負の金額値で、通貨シンボルと数字の間にスペースがない場合には 0 に設定される。スペースがある場合には 1 に、シンボルと符号文字列の間にスペースがある場合には 2 に設定される。
char p_sign_posn	負でない金額で、positive_sign 文字列をどこに置くかを指定するために使用される整数。
char n_sign_posn	負の金額で、negative_sign 文字列をどこに置くかを指定するために使用される整数。

char*型の構造体のメンバは文字列へのポインタであり、そのうちの (decimal_point を除く) 任意のものが "" をポイントすることができます。これは、対応する値が現在のロケールでは使用できないか、長さゼロであることを示します。char型の構造体のメンバは正の数値で、そのうちの任意のものが CHAR_MAX になることができます。これは、対応する値が現在のロケールでは使用できないことを示します。CHAR_MAXは<limits.h>ヘッダ・ファイルに定義されています。

<limits.h>ヘッダの中の CHAR_MAX マクロの値は、プログラムが/UNSIGNED_CHAR 修飾子を付けてコンパイルされたかどうかに依存することに注意してください。

- CHAR_MAX マクロを、現在のロケールで使用できない値を示すために使用するの、プログラムが/UNSIGNED_CHAR なしでコンパイルされた場合に限られる (デフォルトは/NOUNSIGNED_CHAR)。
- プログラムが/UNSIGNED_CHAR を指定してコンパイルされた場合には、CHAR_MAX マクロの代わりに SCHAR_MAX マクロを使用する。

/NOUNSIGNED_CHAR モードでは、CHAR_MAX と SCHAR_MAX の値は同じです。このため、SCHAR_MAX との比較では、どちらの/[NO]UNSIGNED_CHAR モードを使用しているかにかかわらず、正しい結果が得られます。

メンバ grouping と mon_grouping は、数値をフォーマットするときの、個々の桁のグループのサイズを定義する文字列をポイントします。個々のグループ・サイズはセミコロン (;) によって区切られます。たとえば、grouping が文字列5;3をポイントしており、thousands_sep 文字がコンマ (,) ならば、数値 123450000 は 1,234,50000 にフォーマットされます。

grouping と mon_grouping の要素は、次のように解釈されます。


```

        /* It failed to load the locale */
        printf("ERROR : The locale is unknown");
        exit(EXIT_FAILURE);
    }

    /* Get the lconv structure from the locale */
    lconv_ptr = (struct lconv *) localeconv();

    /* Compare the international currency symbol string with an */
    /* empty string. If they are equal, then the international */
    /* currency symbol is not defined in the locale. */
    if (strcmp(lconv_ptr->int_curr_symbol, "")) {
        printf("International Currency Symbol = %s\n",
            lconv_ptr->int_curr_symbol);
    }
    else {
        printf("International Currency Symbol =");
        printf("[Not available in this locale]\n");
    }

    /* Compare International Fractional Digits with CHAR_MAX. */
    /* If they are equal, then International Fractional Digits */
    /* are not defined in this locale. */
    if ((unsigned char) (lconv_ptr->int_frac_digits) != CHAR_MAX) {
        printf("International Fractional Digits = %d\n",
            lconv_ptr->int_frac_digits);
    }
    else {
        printf("International Fractional Digits =");
        printf("[Not available in this locale]\n");
    }
}

```

上の例のプログラムを実行すると、次の結果が出力されます。

```

International Currency Symbol = GBP
International Fractional Digits = 2

```

localtime, localtime_r

時刻値を、分割されたローカル時刻に変換します。

Format

```
#include <time.h>

struct tm *localtime (const time_t *timer);

struct tm *localtime_r (const time_t *timer, struct tm *result); (ISO POSIX-1)
```

関数バリエーション

`_DECC_V4_SOURCE` および `_VMS_V6_SOURCE` 機能テスト・マクロを定義してコンパイルすると、OpenVMS Version 7.0 より前の動作と等価な、`localtime_r`関数へのローカル時刻ベースのエントリ・ポイントが使用可能となります。

Argument

`timer`

Epoch 後の経過秒数へのポインタ。この時刻は、`time`関数を使って生成することも、独自に指定することもできます。

`result`

結果が格納される`tm`構造体へのポインタ。`tm`構造体は`<time.h>`ヘッダ・ファイルに定義されており、表 REF-4 にも示しています。

Description

`localtime`および`localtime_r`関数は、`timer`によって指定された時刻 (Epoch 後の経過秒数) を、ローカル時刻で表現される分割された時刻に変換し、`tm`構造体に格納します。

`localtime_r`関数と`localtime`関数の違いは、前者が結果をユーザ指定の`tm`構造体に格納することです。後者は結果をHP C RTL によって割り当てられたスレッド固有の静的メモリに格納します。これは、後の`localtime`の呼び出しによって上書きされるので、保存しておきたい場合にはコピーを作成する必要があります。

実行に成功すると、`localtime`は`tm`構造体へのポインタを返し、`localtime_r`はその第2引数を返します。実行に失敗すると、これらの関数はNULLポインタを返します。

tm構造体は<time.h>ヘッダ・ファイルに定義されており，表 REF-4 にも示しています。

表 REF-4 tm 構造体

int tm_sec;	秒 (0-60)
int tm_min;	分 (0-59)
int tm_hour;	時 (0-23)
int tm_mday;	日 (1-31)
int tm_mon;	月 (0-11)
int tm_year;	1900 年を基準とした年
int tm_wday;	日曜日を基準とした曜日 (0-6)
int tm_yday;	1 月 1 日からの日数 (0-365)
int tm_isdst;	サマータイム・フラグ
	<ul style="list-style-type: none">tm_isdst = 0 標準tm_isdst = 1 サマータイム
long tm_gmtoff; ¹	グリニッジ標準時からの東へのずれの秒数 (負の値はグリニッジ標準時からの西へのずれの秒数を示す)
char *tm_zone; ¹	タイム・ゾーン文字列。例: "GMT"

¹このフィールドは ANSI C 構造体に対する拡張です。このフィールドは，プログラムのコンパイルの際に/STANDARD=ANSI89 を指定するか，_DECC_V4_SOURCE を定義した場合には存在しません。

time_t型は，<time.h>ヘッダ・ファイルに，次のように定義されています。

typedef long int time_t

注意

一般に，UTC ベースの時刻関数は，全プロセス的なデータであるメモリ内のタイム・ゾーン情報に影響を与えることがあります。ただし，アプリケーションの実行中にシステム・タイム・ゾーンが変化せず (通常は変化しません)，タイム・ゾーン・ファイルのキャッシュが有効になっていれば (デフォルトの設定)，時刻関数の_rバリエント，asctime_r, ctime_r, gmtime_r, およびlocaltime_rは，スレッドセーフかつAST リエントラントとなります。

ただし，アプリケーションの実行中にシステム・タイム・ゾーンが変化する場合がある場合，またはタイム・ゾーン・ファイルのキャッシュが有効になっていない場合には，UTC ベースの時刻関数の両方のバリエントは，スレッドセーフでもAST リエントラントでもない第3クラスの関数に属します。

localtime, localtime_r

Return value

x	tm構造体へのポインタ。
NULL	失敗を示します。

log, log2, log10

引数の対数を返します。

Format

```
#include <math.h>
double log (double x);
float logf (float x); (Alpha, I64)
long double logl (long double x); (Alpha, I64)
double log2 (double x); (Alpha, I64)
float log2f (float x); (Alpha, I64)
long double log2l (long double x); (Alpha, I64)
double log10 (double x);
float log10f (float x); (Alpha, I64)
long double log10l (long double x); (Alpha, I64)
```

Argument

x
実数。

Description

log関数は、xの自然対数 (底 e) を計算します。

log2関数は、xの底 2 の対数を計算します。

log10関数は、xの常用対数 (底 10) を計算します。

Return value

x	引数の (それぞれの底の) 対数。
−HUGE_VAL	xは 0 (errnoは ERANGE に設定されます) , または負の値 (errnoは EDOM に設定されます) です。
NaN	xは NaN です。errnoは EDOM に設定されます。

log1p (*Alpha, I64*)

$\ln(1+y)$ を正確に計算します。

Format

```
#include <math.h>
double log1p (double y);
float log1pf (float y);
long double log1pl (long double y);
```

Argument

y
−1 よりも大きい実数。

Description

log1p関数は、yが小さい値である場合でも、 $\ln(1+y)$ を正確に計算します。

Return value

x	(1+y) の自然対数。
−HUGE_VAL	yは−1 よりも小さい (errnoは EDOM に設定されます) , またはy = −1 (errnoは ERANGE に設定されます) です。
NaN	yは NaN です。errnoは EDOM に設定されます。

logb (Alpha, I64)

引数の基底に依存しない指数を返します。

Format

```
#include <math.h>

double logb (double x);
float logbf (float x);
long double logbl (long double x);
```

Argument

x
ゼロでない実数。

Description

logb関数は、ゼロでないxに対し、xの指数部、すなわち $\log_2 |x|$ の整数部を符号付き浮動小数点値として返します。

Return value

x	xの指数。
−HUGE_VAL	x = 0.0; errnoは EDOM に設定されます。
+Infinity	xは +Infinity または −Infinity です。
NaN	yは NaN です。errnoは EDOM に設定されます。

longjmp

ネストした一連の関数呼び出しから、通常の方法を使わずに、つまり一連のreturn文を使用せずに、定義済みのポイントに制御を戻すための手段を提供します。longjmp関数は、環境バッファのコンテキストを復元します。

Format

```
#include <setjmp.h>

void longjmp (jmp_buf env, int value);
```

Argument

env

環境バッファ。呼び出し元の関数のレジスタ・コンテキストを保持するのに十分な長さを持つ整数の配列でなくてはなりません。jmp_buf型は<setjmp.h>ヘッダ・ファイルに定義されています。バッファには、プログラム・カウンタ (PC) を含む汎用レジスタの内容が格納されます。

value

longjmpからsetjmpに渡され、後のsetjmp呼び出しの戻り値になります。渡されたvalueが0だった場合には、1に変換されます。

Description

setjmpは、初めて呼び出されたときには値0を返します。その後、setjmpの呼び出しと同じ環境を指定してlongjmpを呼び出すと、制御は通常どおりに返ったかのように再びsetjmp呼び出しに戻されます。この2回目のリターンにおけるsetjmpの戻り値は、longjmp呼び出しで指定されたvalueです。setjmpの真の値を保存するためには、対応するlongjmpが呼び出されるまで、setjmpを呼び出す関数を再び呼び出さないようにする必要があります。

setjmp関数はハードウェア汎用レジスタを保存し、longjmp関数はそれらを復元します。longjmpの後には、volatileとしてマークされていないローカル自動変数を除くすべての変数が、longjmpの時点の値を持つようになります。volatileとしてマークされていないローカル自動変数の値は不定です。

setjmpおよびlongjmp関数は、OpenVMS 条件処理機能を使用して、シグナル・ハンドラによる非ローカル goto を実現します。longjmp関数は、HP C RTL 指定のシグナルを生成し、OpenVMS 条件処理機能がデスティネーションに戻るようにすること

で実装されています。HP C RTL は、任意のHP Cイメージのシグナル処理を制御できなくてはなりません。

HP Cがシグナル処理を制御できるようにするためには、すべての例外処理を (LIB\$ESTABLISHではなく) VAXC\$ESTABLISH関数の呼び出しを通して設定しなくてはなりません。詳細については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 4.2.5 項と、VAXC\$ESTABLISH関数を参照してください。

注意

Alpha システムと I64 システムの C RTL には、標準とは異なるdecc\$setjmp関数と decc\$fast_longjmp関数が用意されています。標準関数の代わりにこれらの非標準関数を使用するためには、プログラムを__FAST_SETJMP または __UNIX_SETJMP マクロを定義してコンパイルする必要があります。

標準のlongjmp関数とは異なり、decc\$fast_longjmp関数は第 2 引数を 0 から 1 に変換しません。decc\$fast_longjmpの呼び出しの後、対応するsetjmp関数は、decc\$fast_longjmp呼び出しで指定された第 2 引数とまったく同じ値で返ります。

制限事項

OpenVMS 条件ハンドラからlongjmp関数を呼び出すことはできません。ただし、以下のネスト制約の範囲内で、HP C RTL がサポートしている任意のシグナルに対して確立されたシグナル・ハンドラからlongjmpを呼び出すことができます。

- longjmp関数は、ネストしたシグナル・ハンドラから呼び出された場合には動作しない。他のシグナル・ハンドラ内で生成された例外の結果として実行されたシグナル・ハンドラから呼び出されたlongjmp関数の結果は未定義である。
- 対応するlongjmpを、シグナルの処理が完了する前に発行したい場合を除いて、シグナル・ハンドラからsetjmp関数を呼び出してはならない。
- 終了ハンドラ (atexitまたはSYSSDCLEXH で設定) の中からlongjmp関数を呼び出してはならない。終了ハンドラはイメージのティアダウンの後に呼び出されるので、longjmpのデスティネーション・アドレスは存在しなくなっている。
- シグナル・ハンドラの中から、メインの実行スレッドに戻るためにlongjmpを呼び出すと、プログラムの状態の一貫性が失われることがある。副作用として、I/O が実行できなくなったり、UNIX シグナルを受信できなくなったりする可能性がある。

longname

ターミナルのフル・ネームを返します。

Format

```
#include <curses.h>

void longname (char *termbuf, char *name);
```

関数バリエーション

longname関数は、それぞれ 32 ビットと 64 ビットのポインタ・サイズで使用するための `_longname32` と `_longname64` という名前のバリエーションを持っています。ポインタ・サイズ固有の関数の使用方法については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

Argument

termbuf
ターミナルの名前を含んでいる文字列。

name
64 文字以上の長さを持つ文字列バッファ。

Description

ターミナル名は可読形式なので、Curses がターミナルを正しく識別したかどうかをダブル・チェックすることができます。仮引数の `termbuf` は UNIX ソフトウェアとの互換性のために必要なもので、OpenVMS 環境では何の機能も持っていません。移植性が重要な場合は、UNIX システム環境でデータベース `termcap` が提供している機能を実行するダミー・ルーチンのセットを作成する必要があります。

lrand48

一様分布の擬似乱数列を生成します。48 ビットの符号付き long 整数を返します。

Format

```
#include <stdlib.h>

long int lrand48 (void);
```

Description

lrand48関数は、線形合同法アルゴリズムと48ビット整数算術演算を使用して、擬似乱数を生成します。

この関数は、 $0 \leq y < 2^{31}$ の範囲内で一様に分布する、負でない long 整数を返します。

lrand48関数を呼び出す前に、srand48、seed48、またはlcong48を使用して乱数ジェネレータを初期化してください。初期化はlrand48関数を呼び出す前に行わなくてはなりません。lrand48は、生成された最後の48ビットのXiを内部バッファに格納するからです(これは推奨はされませんが、drand48、lrand48、またはmrand48関数が、初期化関数を呼び出すことなく呼び出された場合には、定数のデフォルト・イニシアライザ値が自動的に提供されます)。

この関数は、次の線形合同式に従って、48ビットの整数値Xiのシーケンスを生成します。

$$X_{n+1} = (aX_n + c) \bmod m \quad n \geq 0$$

引数mは 2^{48} に等しいので、48ビット整数算術演算が実行されます。lcong48関数を呼び出さなかった場合、乗数値aと加算される値cは次のようになります。

$$\begin{aligned} a &= 5DEECE66D_{16} = 2736731631558 \\ c &= B_{16} = 138 \end{aligned}$$

lrand48関数から返される値は、まずシーケンス内の次の48ビットXiを生成することによって計算されます。その後、返されるデータ項目の型に応じて、適切なビットがXiの最上位ビットからコピーされ、戻り値に変換されます。

drand48、lcong48、mrand48、seed48、およびsrand48も参照してください。

Return value

n

$0 \leq y < 2^{31}$ の範囲で一様分布する, 符号付きの負でない
long 整数。

lrint (Alpha, I64)

現在の丸め方向で最も近い整数値に丸めます。

Format

```
#include <math.h>
long lrint (double x);
long lrintf (float x);
long lrintl (long double x);
```

引数

x
実数値。

Description

lrint関数は、xを現在の丸め方向で最も近い整数値に丸めた値を返します。

Return value

n
成功したことを示します。nは、丸めて得られた整数値です。

lround (*Alpha, I64*)

現在の丸め方向とは関係なく、最も近い整数値に丸めます。値が中央にある場合は、ゼロから離れる方向へ丸めます。

Format

```
#include <math.h>

long lround (double x);
long lroundf (float x);
long lroundl (long double x);
```

引数

x
実数値。

Description

lround関数は、xを現在の丸め方向とは関係なく最も近い整数値に丸めた値を返します。値が中央にある場合は、ゼロから離れる方向へ丸めます。

Return value

n
成功したことを示します。nは、丸めて得られた整数値です。

lseek

ファイルを任意のバイト位置に置き、新しい位置を返します。

Format

```
#include <unistd.h>

off_t lseek (int file_desc, off_t offset, int direction);
```

Argument

file_desc

open, creat, dup, またはdup2から返された整数。

offset

バイト単位で指定されるオフセット。off_tデータ型は、32 ビット整数または 64 ビット整数です。64 ビット・インタフェースでは、2 GB よりも大きなファイル・サイズを扱うことができます。これは、コンパイル時に_LARGEFILE 機能テスト・マクロを次のように定義することで選択できます。

```
CC/DEFINE= _LARGEFILE
```

direction

オフセットを、ファイルの先頭から順方向に計測するのか (direction=SEEK_SET), 現在位置から順方向に計測するのか (direction=SEEK_CUR), またはファイルの終端から逆方向に計測するのか (direction=SEEK_END) を示す整数。

Description

lseek関数は、キャリッジ・コントロールのない固定長レコード・アクセス・ファイルまたはストリーム・アクセス・ファイルを任意のバイト・オフセットに置くことができますが、その他のすべてのファイルはレコード境界に置くことしかできません。

使用可能な標準 I/O 関数は、レコード・ファイルを、その先頭バイト、ファイルの終端、またはレコード境界に置きます。このため、lseekに与えられる引数は、ファイルの先頭または終端、現在位置からの 0 オフセット (任意のレコード境界)、または以前の有効なlseek呼び出しから返された位置を指定していなくてはなりません。

この関数は、新しいファイル位置をoff_t型の整数として返します。これは、offset引数と同じく、_LARGEFILE が定義されている場合には 64 ビット整数で、定義されていない場合には 32 ビット整数です。

任意のタイプのファイルで任意のバイト位置を設定できる移植性の高い方法については、`fgetpos`および`fsetpos`関数を参照してください。

警告

ストリーム・ファイルにアクセスしているときに、ファイルの終端を越えてシークを行い、ファイルへの書き込みを行うと、`lseek`関数はスキップされたバイトにゼロを充填して穴を作成します。

一般にレコード・ファイルでは、`lseek`に対する命令は、以前の`lseek`の有効な呼び出しから返された絶対位置か、ファイルの先頭または終端への移動に限定すべきです。`lseek`の呼び出しがこれらの条件を満たしていなかった場合の結果は予測不可能です。

`open`, `creat`, `dup`, `dup2`, `fseek`も参照してください。

Return value

<code>x</code>	新しいファイル位置。
<code>-1</code>	ファイル記述子が未定義であるか、ファイルの先頭よりも前に対するシークが試みられたことを示します。

lstat (*Alpha, I64*)

指定したファイルの情報を取得します。

Format

```
#include <sys/stat.h>

int lstat (const char *restrict file_path, struct stat *restrict user_buffer);
```

Argument

file_path

対象ファイルのパス名を指すポインタ。

user_buffer

stat 構造体を指すポインタ。この構造体に、ファイルのステータス情報が格納されて返されます。

Description

lstat関数は、指定したファイル (file_path) の情報を取得します。指定したファイルがシンボリック・リンクである場合は、そのシンボリック・リンク自体の情報を返します (この関数とは対照的に、statでは、シンボリック・リンクから参照されているファイルの情報を返します)。

symlink, unlink, readlink, realpath, およびlchownも参照してください。

Return value

0

成功したことを示します。

-1

エラーが発生したことを示します。errnoには、statから返されたerrno値が設定されます。

lwait

特定のファイルに対する I/O が完了するのを待ちます。

Format

```
#include <stdio.h>
int lwait (int fd);
```

Argument

fd
オープン・ファイルに対応するファイル記述子。

Description

lwait関数は、主に保留中の非同期 I/Oが完了するのを待つために使用されます。

Return value

0	実行に成功したことを示します。
-1	エラーを示します。

malloc

メモリの領域を割り当てます。これらの関数はAST リエントラントです。

Format

```
#include <stdlib.h>

void *malloc (size_t size);
```

関数バリエント

malloc関数は、それぞれ 32 ビットと 64 ビットのポインタ・サイズで使用するための `_malloc32` と `_malloc64` という名前のバリエントを持っています。ポインタ・サイズ固有の関数の使用方法については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

Argument

size
割り当てる合計のバイト数。

Description

malloc関数は、引数として指定されたバイト数の、連続したメモリ領域を割り当てます。スペースの初期化は行われません。

注意

mallocルーチンはシステム・ルーチン `LIB$VM_MALLOC` を呼び出します。`LIB$VM_MALLOC` はメモリを割り当てるための汎用ルーチンとして設計されており、幅広いシナリオでブロックの割り当てと再割り当てを効率的に行うために呼び出されます。最も一般的な用途は、比較的小さなメモリ・ブロックの管理ですが、これらの状況でのメモリ割り当ての最も重要な側面は効率性です。

`LIB$VM_MALLOC` は、大きなブロックの分割と隣接するブロックのマージのためにヒープ・ストレージが使い尽くされた場合には、独自の空きスペースを使って要求の処理を行います。メモリはフラグメント化され、未使用ブロックが残ることがあります。ヒープ・ストレージが使い尽くされると、`LIB$VM_MALLOC` は要求を満たすために独自の空きスペースとマージ済みブロックを管理しますが、さまざまなサイズのメモリ割り当てが行われると、どうしても未使用のブロックが生じます。

LIB\$VM_MALLOC はすべての状況で最適な対応を行えるわけではないので、メモリ使用上の特殊なニーズがある場合には、プログラマは独自のメモリ管理を行うようにしてください。これにより、個々のアプリケーションでメモリを最適に使用することができます。

『OpenVMS Programming Concepts Manual』は、使用可能ないくつかのメモリ割り当てルーチンについて解説しています。これらは3つの階層レベルにグループ化されています。

1. 一番上のレベルには、RTL ヒープ管理ルーチンの LIB\$GET_VM と LIB\$FREE_VM があります。これらは、任意のサイズのメモリ・ブロックの割り当てと解放を行うメカニズムを提供しています。またこのレベルには、LIB\$CREATE_VM_ZONE などのように、ゾーン概念に基づくルーチンがあります。
 2. 次のレベルには、RTL ページ管理ルーチンの LIB\$GET_VM_PAGE と LIB\$FREE_VM_PAGE があります。これらは、指定された数の連続したページを割り当てます。
 3. 一番下のレベルには、\$CRETVA や\$EXPREG などのメモリ管理システム・サービスがあります。これらを使用すると、アドレス空間の割り当てを細かく制御することができます。ただしこのレベルでは、プログラマが割り当てを正確に管理する必要があります。
-

Return value

x	クォードワード境界 (<i>Alpha only</i>) または オクタワード境界 (<i>I64 only</i>) に配置された第 1 バイトのアドレス。
NULL	関数が十分なメモリを割り当てられなかったことを示します。errno は ENOMEM に設定されます。

mblen

マルチバイト文字を構成するバイトの数を決定します。

Format

```
#include <stdlib.h>

int mblen (const char *s, size_t n);
```

Argument

s
マルチバイト文字へのポインタ。

n
マルチバイト文字を構成するバイト数の最大値。

Description

文字がnバイト以下である場合、mblen関数はsがポイントするマルチバイト文字を構成するバイトの数を返します。文字がnバイトよりも大きい場合、関数はエラーを示す-1を返します。

この関数は、プログラムのカレント・ロケールの LC_CTYPE カテゴリの影響を受けます。

Return value

x	次のn個以下のバイトが有効な文字を構成している場合、マルチバイト文字を構成するバイトの数。
0	sが NULL または NULL 文字へのポインタであることを示します。
-1	エラーを示します。関数はerrnoを、無効な文字が検出されたことを示す EILSEQ に設定します。

mbrlen

マルチバイト文字を構成するバイトの数を決定します。

Format

```
#include <wchar.h>

size_t mbrlen (const char *s, size_t n, mbstate_t *ps);
```

Argument

s
マルチバイト文字へのポインタ。

n
マルチバイト文字を構成するバイト数の最大値。

ps
`mbstate_t`オブジェクトへのポインタ。NULL ポインタが指定された場合、関数は内部の`mbstate_t`オブジェクトを使用します。`mbstate_t`は、状態依存のコードセットのための変換状態を保持する目的に使われる不透明のデータ型です。

Description

`mbrlen`関数は、次の呼び出しと等価です。

```
mbrtowc(NULL, s, n, ps != NULL ? ps : &internal)
```

`internal`は、`mbrlen`関数の`mbstate_t`オブジェクトです。

`s`がポイントするマルチバイト文字が`n`バイト以下であれば、関数は文字を構成するバイトの数を返します (シフト・シーケンスを含みます)。

エンコーディング・エラーが発生するか、次の`n`バイトが不完全な、しかし有効なマルチバイト文字の一部でありうるバイト列である場合、関数はそれぞれ-1 または-2 を返します。

`mbrtowc`も参照してください。

Return value

x	マルチバイト文字を構成するバイトの数。
0	sが NULL ポインタまたは null バイトへのポインタであることを示します。
-1	エンコーディング・エラーが発生したことを示します。 次のn個以下のバイトは、完全で有効なマルチバイト文字を構成しません。errnoは EILSEQ に設定されます。変換状態は未定義です。
-2	不完全な、しかし有効なマルチバイト文字の一部でありうるバイト列であることを示します (n個すべてのバイトが処理されています)。

mbrtowc

マルチバイト文字を、そのワイド文字表現に変換します。

Format

```
#include <wchar.h>

size_t mbrtowc (wchar_t *pwc, const char *s, size_t n, mbstate_t *ps);
```

Argument

pwc

結果として得られるワイド文字コードへのポインタ。

s

マルチバイト文字へのポインタ。

n

マルチバイト文字を構成するバイト数の最大値。

ps

mbstate_t オブジェクトへのポインタ。NULL ポインタが指定された場合、関数は内部のmbstate_t オブジェクトを使用します。mbstate_t は、状態依存のコードセットのための変換状態を保持する目的に使われる不透明のデータ型です。

Description

s が NULL ポインタである場合、mbrtowc は次の呼び出しと等価です。

```
mbrtowc(NULL, "", 1, ps)
```

この場合、pwc と n の値は無視されます。

s が NULL ポインタでない場合、mbrtowc は s がポイントするバイトから始まる最高 n バイトの内容を検査し、次のマルチバイト文字を完成させるために必要なバイト数を決定します (シフト・シーケンスを含みます)。

次のマルチバイト文字が完成したと判断した場合、関数は対応するワイド文字の値を決定し、pwc が NULL ポインタでなければ、その値を pwc がポイントするオブジェクトに格納します。対応するワイド文字が null ワイド文字である場合、結果として得られる状態は初期変換状態です。

mbrtowcがカウント関数として呼び出された場合、つまりpwcがNULLポインタで、sがNULLポインタでもnullバイトへのポインタでもなかった場合、内部のmbstate_tオブジェクトの値は変更されません。

Return value

x	マルチバイト文字を構成するバイトの数。
0	次のn個以下のバイトは、nullワイド文字 (pwcがNULLポインタでない場合に格納される値) に対応するマルチバイト文字を完成させます。nullバイトに対応するワイド文字コードはゼロです。
-1	エンコーディング・エラーを示します。次のn個以下のバイトは、完全で有効なマルチバイト文字を構成しません。errnoはEILSEQに設定されます。変換状態は未定義です。
-2	不完全な、しかし有効なマルチバイト文字の一部でありうるバイト列であることを示します (n個すべてのバイトが処理されています)。

mbstowcs

マルチバイト文字のシーケンスを、対応するワイド文字コードのシーケンスに変換します。

Format

```
#include <stdlib.h>

size_t mbstowcs (wchar_t *pwcs, const char *s, size_t n);
```

Argument

pwcs
結果として得られるワイド文字コードのシーケンスが格納される配列へのポインタ。

s
マルチバイト文字の配列へのポインタ。

n
pwcsがポイントする配列に格納できるワイド文字コードの数の最大値。

Description

mbstowcs関数は、sがポイントする配列内のマルチバイト文字のシーケンスを、コードn個を上限としてワイド文字コードのシーケンスに変換し、pwcsがポイントする配列に格納します

この関数は、プログラムのカレント・ロケールの LC_CTYPE カテゴリの影響を受けます。コピーが互いにオーバーラップするオブジェクト間で行われた場合の動作は未定義です。

Return value

x	変更される，または必要とされる配列要素の数。末尾のゼロ・コードは含みません。返される値がnの場合には，配列の末尾にはゼロは格納されません。pwcsが NULL ポインタならば，mbstowcsはワイド文字配列に必要とされる要素の数を返します。
(size_t) -1	エラーが発生したことを示します。関数はerrnoを，無効な文字が検出されたことを示す EILSEQ に設定します。

mbtowc

マルチバイト文字を、それと等価なワイド文字に変換します。

Format

```
#include <stdlib.h>

int mbtowc (wchar_t *pwc, const char *s, size_t n);
```

Argument

pwc
結果として得られるワイド文字コードへのポインタ。

s
マルチバイト文字へのポインタ。

n
次のマルチバイト文字を構成するバイトの数の最大値。

Description

文字がnバイト以下である場合、mbtowc関数はsがポイントするマルチバイト文字を、それと等価なワイド文字に変換します。文字が無効であるか、nバイトよりも大きい場合、関数はエラーを示す-1を返します。

pwcが NULL ポインタで、sが null ポインタでない場合、関数は (nの値にかかわらず) sがポイントするマルチバイト文字を構成するバイト数を判定します。

この関数は、プログラムのカレント・ロケールの LC_CTYPE カテゴリの影響を受けます。

Return value

x	sがポイントする有効な文字を構成するバイト数。
0	sは NULL ポインタであるか, null バイトへのポインタです。
-1	エラーを示します。関数はerrnoを, 無効な文字が検出されたことを示す EILSEQ に設定します。

mbsinit

mbstate_tオブジェクトが初期変換状態を記述しているかどうかを判定します。

Format

```
#include <wchar.h>

int mbsinit (const mbstate_t *ps);
```

Argument

ps
mbstate_tオブジェクトへのポインタ。mbstate_tは、状態依存のコードセットのための変換状態を保持する目的に使われる不透明のデータ型です。

Description

psが NULL ポインタでない場合、mbsinit関数は、psがポイントするmbstate_tオブジェクトが初期変換状態を記述しているかどうかを判定します。ゼロのmbstate_tオブジェクトは、つねに初期変換状態を記述しています。

Return value

ゼロ以外	ps引数が NULL ポインタであるか、psがポイントするmbstate_tオブジェクトが初期変換状態を記述しています。
0	psがポイントするmbstate_tオブジェクトは初期変換状態を記述していません。

mbsrtowcs

マルチバイト文字のシーケンスを、対応するワイド文字コードのシーケンスに変換します。

Format

```
#include <wchar.h>

size_t mbsrtowcs (wchar_t *dst, const char **src, size_t len, mbstate_t *ps);
```

関数バリエーション

mbsrtowcs関数は、それぞれ 32 ビットと 64 ビットのポインタ・サイズで使用するための _mbsrtowcs32と _mbsrtowcs64という名前のバリエーションを持っています。ポインタ・サイズ固有の関数の使用方法については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

Argument

dst

結果として得られるワイド文字コードのシーケンスが格納されるデスティネーション配列へのポインタ。

src

変換するマルチバイト文字のシーケンスを含んだ配列へのポインタのアドレス。

len

dstがポイントする配列に格納できるワイド文字コードの数の最大値。

ps

mbstate_tオブジェクトへのポインタ。NULL ポインタが指定された場合、関数は内部のmbstate_tオブジェクトを使用します。mbstate_tは、状態依存のコードセットのための変換状態を保持する目的に使われる不透明のデータ型です。

Description

mbsrtowcs関数は、psがポイントするオブジェクトが記述している変換状態から開始して、srcが間接的にポイントしている配列に含まれているマルチバイト文字のシーケンスを、対応するワイド文字のシーケンスに変換します。

dstがNULLポインタでなければ、変換後の文字はdstがポイントする配列に格納されます。変換は末尾の null 文字まで行われ、この null 文字も格納されます。

以下のいずれかの理由が生じた場合、変換は途中で停止します。

- 有効なマルチバイト文字を構成しないバイトのシーケンスを検出した。
- dstがNULLポインタでない場合、dstがポイントする配列にlen個のコードが格納された。

dstがNULLポインタでない場合、srcがポイントするポインタ・オブジェクトには、NULLポインタが代入されるか(末尾の null ワイド文字に達したために変換が停止した場合)、または最後に変換されたマルチバイト文字(存在する場合)の直後のアドレスが代入されます。末尾の null ワイド文字に達したために変換が停止した場合、結果として得られる状態は初期変換状態です。

Return value

n	変換に成功したマルチバイト文字の数。末尾の null は(存在する場合でも) 含みません。
-1	エラーを示します。有効なマルチバイト文字を構成しないバイトのシーケンスが現れました。errnoは EILSEQ に設定されます。変換状態は未定義です。

memccpy

メモリ領域内の文字列の間で、文字を順次コピーします。

Format

```
#include <string.h>

void *memccpy (void *dest, void *source, int c, size_t n);
```

関数バリエーション

memccpy関数は、それぞれ 32 ビットと 64 ビットのポインタ・サイズで使用するための `_memccpy32` と `_memccpy64` という名前のバリエーションを持っています。ポインタ・サイズ固有の関数の使用方法については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

Argument

dest
デスティネーション文字列の位置へのポインタ。

source
ソース文字列の位置へのポインタ。

c
検索したい文字。

n
コピーしたい文字数。

Description

memccpy関数は、メモリ領域内の文字列に作用します。メモリ領域とは、カウントによって制限され、null 文字で終了しない、連続した文字のグループです。この関数は、コピー先のメモリ領域でのオーバーフローをチェックしません。memccpy関数は `<string.h>` ヘッド・ファイルに定義されています。

memccpy関数は、sourceがポイントする位置から、以下のいずれかが起こるまで、destがポイントする位置へと文字を順次コピーします。

- cで指定された文字 (unsigned charに変換) のコピーが終了した。
- nで指定された数の文字のコピーが終了した。

Return value

x	destがポイントする文字列の中の、cで指定される文字の後の文字へのポインタ。
NULL	エラーを示します。文字列の中でn個の文字をスキャンしても、文字cは発見されませんでした。

memchr

指定されたオブジェクトの最初のsizeバイト内で、指定されたバイトの最初のカレンスを発見します。

Format

```
#include <string.h>

void *memchr (const void *s1, int c, size_t size);
```

関数バリエーション

memchr関数は、それぞれ 32 ビットと 64 ビットのポインタ・サイズで使用するための _memchr32と _memchr64という名前のバリエーションを持っています。ポインタ・サイズ固有の関数の使用方法については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

Argument

s1
検索するオブジェクトへのポインタ。

c
検索するバイト値。

size
検索するオブジェクトの長さ。

sizeがゼロの場合、memchrは NULL を返します。

Description

strchrとは異なり、memchr関数は null 文字が現れても停止しません。

Return value

ポインタ

バイトの最初のオカレンスへのポインタ。

NULL

指定されたバイトがオブジェクト内に含まれていなかったことを示します。

memcmp

2つのオブジェクトをバイト単位で比較します。比較操作は、各オブジェクトの最初のバイトから開始されます。

Format

```
#include <string.h>

int memcmp (const void *s1, const void *s2, size_t size);
```

Argument

s1

第1のオブジェクトへのポインタ。

s2

第2のオブジェクトへのポインタ。

size

比較するオブジェクトの長さ。

sizeがゼロの場合、2つのオブジェクトは等しいものと見なされます。

Description

memcmp関数はネイティブなバイト比較を使用します。返される値の符号は、比較されるオブジェクトの中の、最初の異なるバイトの値を比較したときに得られる符号によって決定されます。strcmp関数とは異なり、memcmp関数はnull文字が現れても停止しません。

Return value

x

負の整数、0、または正の整数。それぞれ、第1のオブジェクトの文字値が第2のオブジェクトの文字値よりも小さい、等しい、または大きい場合に対応します。

memcpy

オブジェクト間で指定された数のバイトをコピーします。

Format

```
#include <string.h>

void *memcpy (void *dest, const void *source, size_t size);
```

関数バリエーション

memcpy関数は、それぞれ 32 ビットと 64 ビットのポインタ・サイズで使用するための _memcpy32 と _memcpy64 という名前のバリエーションを持っています。ポインタ・サイズ固有の関数の使用方法については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

Argument

dest
デスティネーション・オブジェクトへのポインタ。

source
ソース・オブジェクトへのポインタ。

size
コピーするオブジェクトの長さ。

Description

memcpy関数は、sourceがポイントするオブジェクトから、destがポイントするオブジェクトに、size個のバイトをコピーします。コピー先のメモリ領域 (dest) のオーバーフローのチェックは行いません。strcpy関数とは異なり、memcpy関数は null 文字が現れても停止しません。

memcpy

Return value

x

destの値。

Example

```
#include <string.h>
#include <stdio.h>

main()
{
    char pdest[14] = "hello  there";
    char *psource = "you are there";

    memmove(pdest, psource, 7);
    printf("%s\n", pdest);
}
```

この例は、次の出力を生成します。

```
you are there
```

memset

指定されたオブジェクト内の指定されたバイト数を、指定された値に設定します。

Format

```
#include <string.h>

void *memset (void *s, int value, size_t size);
```

関数バリエーション

memset関数は、それぞれ 32 ビットと 64 ビットのポインタ・サイズで使用するための `_memset32` と `_memset64` という名前のバリエーションを持っています。ポインタ・サイズ固有の関数の使用方法については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

Argument

s
配列ポインタ。

value
s に格納する値。

size
s に格納するバイト数。

Description

memset関数は、value (unsigned char に変換) を、s がポイントするオブジェクトの最初のsize個の文字のそれぞれにコピーします。

この関数はsを返します。s がポイントするコピー先のメモリ領域のオーバーフローのチェックは行いません。

memset

Return value

x

sの値。

mkdir

ディレクトリを作成します。

Format

```
#include <stat.h>

int mkdir (const char *dir_spec, mode_t mode); (ISO POSIX-1)

int mkdir (const char *dir_spec, mode_t mode, ... ); (HP C Extension)
```

Argument

dir_spec

有効な OpenVMS または UNIX スタイルのディレクトリ指定。デバイス名を含むことができます。次に例を示します。

```
DBA0:[BAY.WINDOWS]      /*  OpenVMS      */
/dba0/bay/windows        /*  UNIX style  */
```

この指定は、ノード名、ファイル名、ファイル・タイプ、ファイル・バージョン、またはワイルドカード文字を含むことはできません。同じ制約が、UNIX スタイルのディレクトリ指定にも適用されます。UNIX スタイルの指定の制約については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル(上巻)』第 1 章を参照してください。

mode

ファイル保護。具体的なファイル保護の情報については、このセクションの `chmod` 関数を参照してください。

新しいディレクトリのファイル保護は、`mode` 引数、プロセスのファイル保護マスク (`umask` 関数を参照)、および親ディレクトリのデフォルトの保護から決定されます。

OpenVMS におけるディレクトリ作成の動作と同様に、`mkdir` はディレクトリに対して決して削除アクセスを適用しません。削除アクセスを設定する必要があるアプリケーションは、`chmod` を明示的に呼び出して、書き込み許可を設定するようにしてください。

新しく作成されたディレクトリのファイル保護がどのように設定されるかについては、この関数の説明のセクションを参照してください。

...

以下のオプションの引数を表します。これらの引数は、引数リストの中での位置が固定されており、適当な位置に置くことはできません。

`unsigned int uic`

作成されるディレクトリのオーナーを識別する利用者識別コード (UIC)。この引数が 0 の場合、HP C RTL は、作成されたディレクトリに、親ディレクトリの UIC を与えます。この引数が指定されなかった場合、HP C RTL は作成されたディレクトリにユーザの UIC を与えます。このオプションの引数はHP C RTL に固有のものであり、移植性はありません。

`unsigned short max_versions`

作成されたディレクトリに保持されるファイル・バージョンの数の最大値。システムはディレクトリを自動的にページして、各ファイルを最高でmax_versions個しか保持しません。

この引数が 0 の場合、HP C RTL はファイル・バージョンの数に上限を設けません。

この引数が指定されなかった場合、HP C RTL は作成されたディレクトリに、親ディレクトリのデフォルトのバージョン制限を設定します。

このオプションの引数はHP C RTL に固有のものであり、移植性はありません。

`unsigned short r_v_number`

デバイスがボリューム・セットの一部である場合、作成されたディレクトリを配置するボリューム (デバイス)。この引数が指定されなかった場合、HP C RTL は作成されたディレクトリをボリューム・セット内に適当に配置します。このオプションの引数はHP C RTL に固有のものであり、移植性はありません。

Description

`dir_spec`が、存在しないディレクトリを含んだパスを指定していた場合には、中間のディレクトリも作成されます。UNIX システムでは、これらの中間ディレクトリは存在していなくてはならず、自動的に作成されることはありません。

オプションの引数を 1 つも指定しなかった場合、HP C RTL はディレクトリにユーザの UIC と親ディレクトリのデフォルトのバージョン制限を設定し、ボリューム・セット内で適当にディレクトリを配置します。何らかの引数を指定した場合、`uic`引数と`max_versions`引数のデフォルト値は設定されません。

注意

UNIX システム・コール関数 `umask`, `mkdir`, `creat`, および `open` を使っ
て, OpenVMS RMS デフォルト保護を指定してファイルを作成するに
は, `umask` を直接呼び出さないプログラムから, 0777 のファイル保護モード
引数を使用して `mkdir`, `creat`, および `open` を呼び出します。これらのデフォ
ルト保護では, ACL やファイルの前のバージョンなどに基づいて, 保護が正し
く設定されます。

`vfork/exec` 呼び出しを行うプログラムでは, 新しいプロセス・イメージ
は, `umask` が呼び出されたかどうかの状態を, 呼び出し元のプロセス・イメー
ジから継承します。 `umask` 設定と, `umask` 関数が呼び出されたかどうかの状態
は, どちらも属性として継承されます。

`mode` 引数で指定されるファイル保護は, 新しいディレクトリのファイル保護
が `mode` 引数とファイル保護マスクの補数のビット論理積に設定されるように, プロ
セスのファイル保護マスクによって変更されます。

新しいディレクトリには, ゼロの保護値ビットが親ディレクトリから継承されるよう
な形で, 親ディレクトリからデフォルトのファイル保護が継承されます。ただし, 親
ディレクトリのファイル保護の中の, 削除アクセスを示すビットは, 新しいディレク
トリのファイル保護の中の対応するビットには継承されません。

Return value

0	成功を示します。
-1	失敗を示します。

Examples

```
1.  umask (0002); /* turn world write access off */
    mkdir ("sys$disk:[.parentdir.childdir]", 0222); /* turn write
                                                    access on */

Parent directory file protection: System:RWD, Owner:RWD, Group:R,
                                World:R
```

モード引数と, `umask` によって設定されたファイル保護マスクの組み合わせから得
られるファイル保護は, $(0222) \& \sim(0002)$, すなわち 0220 です。この保護に親デ
ィレクトリのデフォルト値が適用されると, 新しいディレクトリの保護は次のよ
うになります。

```
File protection:  System:RWD, Owner:RWD, Group:RWD, World:R
```

```
2. umask (0000);  
   mkdir ("sys$disk:[.parentdir.childdir]", 0444); /* turn read  
                                                    access on */  
  
   Parent directory file protection: System:RWD, Owner:RWD,  
                                     Group:RWD, World:RWD
```

モード引数と、umaskによって設定されたファイル保護マスクの組み合わせから得られるファイル保護は、(0444) & ~(0000)、すなわち 0444 です。この保護に親ディレクトリのデフォルト値が適用されると、新しいディレクトリの保護は次のようになります。

File protection: System:RW, Owner:RW, Group:RW, World:RW

削除アクセスは継承されないことに注意してください。

mkstemp

一意のファイル名を作成します。

Format

```
#include <stdlib.h>

int mkstemp (char *template);
```

Argument

template

一意のファイル名に置き換えられる文字列へのポインタ。template引数の中の文字列は、末尾に 6 つの X を含んだファイル名でなくてはなりません。

Description

mkstemp関数は、templateがポイントする文字列の末尾の 6 つの X を一意の文字のセットに置き換え、読み書き用にオープンされたファイルのファイル記述子を返します。

templateがポイントする文字列は、末尾に 6 つの X を含んだファイル名の形になっていなくてはなりません。mkstemp関数は、既存のファイル名との重複が起こらないように、個々の X を移植性のあるファイル名文字セットの文字に置き換えます。

templateがポイントする文字列が、末尾に 6 つの X を含んでいない場合には、-1 が返されます。

Return value

x	オープンされたファイル記述子。
-1	エラーを示します (templateがポイントする文字列は、末尾に 6 つの X を含んでいません)。

mktemp

テンプレートから一意のファイル名を作成します。

Format

```
#include <stdlib.h>

char *mktemp (char *template);
```

関数バリエーション

mktemp関数は、それぞれ 32 ビットと 64 ビットのポインタ・サイズで使用するための `_mktemp32` と `_mktemp64` という名前のバリエーションを持っています。ポインタ・サイズ固有の関数の使用方法については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

Argument

template

ユーザ定義のテンプレートを含んでいるバッファへのポインタ。テンプレートは `namXXXXXX` の形式で指定します。末尾の 6 つの X は、一意の文字の並びに置き換えられます。ユーザは最初の 3 つの文字を指定することができます。template 引数は上書きされるので、文字列リテラル (const オブジェクト) は指定しないようにしてください。

Description

新規のアプリケーションでは、mktempを使用することは勧められません。推奨される方法については、`tmpnam` および `mkstemp` 関数を参照してください。

Return value

x

テンプレートへのポインタ。このテンプレートの内容は、作成されたファイル名に変更されます。この値が `null` 文字列へのポインタだった場合には、一意のファイル名が作成できなかったことを示します。

mktime

ローカル時刻構造体を， Epoch からの経過秒数に変換します。

Format

```
#include <time.h>
time_t mktime (struct tm *timeptr);
```

関数バリエーション

`_DECC_V4_SOURCE` および `_VMS_V6_SOURCE` 機能テスト・マクロを定義してコンパイルすると， OpenVMS Version 7.0 より前の動作と等価な，ローカル時刻ベースの `mktime` 関数へのエントリ・ポイントが使用可能になります。

Argument

`timeptr`
ローカル時刻構造体へのポインタ。

Description

`mktime` 関数は， `timeptr` がポイントするローカル時刻構造体 (`struct tm`) を， `time` 関数が返す値と同じように， Epoch 後の経過秒数 (`time_t` 変数) に変換します。

構造体の `tm_wday` 要素と `tm_yday` 要素の元の値は無視され，ほかの要素の元の値も， `<time.h>` で定義されている範囲には制限されません。正常に完了すると，構造体の `tm_wday` 要素と `tm_yday` 要素が適切に設定され，ほかの要素には指定された時刻を表すように，正常な範囲の値が設定されます。

ローカル時刻をエンコードできない場合， `mktime` は値 (`time_t`) (−1) を返します。

`time_t` 型は `<time.h>` ヘッド・ファイルに次のように定義されています。

```
typedef unsigned long int time_t;
```

ローカル・タイム・ゾーン情報は， `mktime` が `tzset` を呼び出したかのように設定されます。

timeptrがポイントするローカル時刻構造体の中のtm_isdstフィールドが正の値の場合、mktimeは、初期状態では指定された時刻でサマータイム (DST) が有効になっていると仮定します。

tm_isdstが0ならば、mktimeは初期状態ではDST が有効でないと仮定します。

tm_isdstが負の値ならば、mktimeは指定された時刻でDST が有効になっているかどうかを判定しようと試みます。

Return value

x	指定されたカレンダー時刻を、time_t型の値としてエンコードしたもの。
(time_t)(-1)	ローカル時刻をエンコードできませんでした。 戻り値 (time_t)(-1) は、Sun Feb 7 06:28:15 2106 という有効な日付も表現できることに注意してください。

mmap

ファイル・システム・オブジェクトを仮想メモリにマップします。この関数はリエントラントです。

Format

```
#include <types.h>
#include <mman.h>

void mmap (void *addr, size_t len, int prot, int flags, int filedес, off_t off); (X/Open, POSIX-1)
void mmap (void *addr, size_t len, int prot, int flags, int filedес, off_t off ...); (HP C Extension)
```

関数バリエント

mmap関数は、それぞれ 32 ビットと 64 ビットのポインタ・サイズで使用するための `_mmap32` と `_mmap64` という名前のバリエントを持っています。ポインタ・サイズ固有の関数の使用方法については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

Argument

addr

新しいリージョンの開始アドレス (切り捨てによってページ境界に設定されます)。

len

新しいリージョンのバイト長 (ページ境界に丸められます)。

prot

<mman.h>ヘッダ・ファイルに定義されているアクセス許可。PROT_NONE, PROT_READ, または PROT_WRITE を指定します。

flags

以下の任意の組み合わせに対するビット論理和演算の結果として得られる、マップされたリージョンの属性。

- MAP_FILE または MAP_ANONYMOUS
- MAP_VARIABLE または MAP_FIXED
- MAP_SHARED または MAP_PRIVATE

filedes

open関数から返された、新しいマップされたファイル・リージョンにマップしたいファイル。

off

バイト数で指定されるオフセット。off_tデータ型は、64ビット整数または32ビット整数です。64ビット・インタフェースでは、2 GB よりも大きなファイル・サイズを扱うことができます。これは、コンパイル時に_LARGEFILE 機能テスト・マクロを次のように定義することで選択できます。

```
CC/DEFINE= _LARGEFILE
```

...

MAP_SHARED の SYSSCRMPSC システム・サービスのための追加のフラグを指定するオプションの整数。mmap関数のこのオプションの引数 (HP C拡張) は、OpenVMS Version 7.2 で導入されました。

Description

mmap関数は、新しいマップされたファイル・リージョン、新しいプライベート・リージョン、または新しい共用メモリ・リージョンを作成します。

アプリケーションは、mmapを、read、write、標準入出力などの他のファイル・アクセス方式と組み合わせて使用するときには、同期が正しく行われるように注意する必要があります。

また、呼び出し元のアプリケーションは、mmapを呼び出す前に、範囲[off, off+len]の中のですべてのバイトがファイルに書き込まれていることを確認しなくてはなりません (たとえばfsync関数を使用します)。この条件が満たされていないと、mmapは実行に失敗し、errnoを ENXIO (そのようなデバイスまたはアドレスは存在しない) に設定します。

addrおよびlen引数は、要求された開始アドレスと長さを、新しいリージョンのバイト数として指定します。アドレスは、sysconf(_SC_PAGE_SIZE) から返されるページ・サイズの倍数です。

len引数がsysconf(_SC_PAGE_SIZE) から返されるページ・サイズの倍数でなかった場合、リージョンの終端と、リージョンの終端を含んでいるページの終端の間のアドレスに対するすべての参照の結果は未定義となります。

flags引数は、マップされたリージョンの属性を指定します。flagsの値は、<mman.h>ヘッダ・ファイルに定義されている次のシンボリック名のリストのフラグのビット論理和演算によって作成されます。

MAP_FILE	マップされたファイル・リージョンを作成する。
MAP_ANONYMOUS	名前なしのメモリ・リージョンを作成する。
MAP_VARIABLE	リージョンを計算されたアドレスに配置する。
MAP_FIXED	リージョンを固定されたアドレスに配置する。
MAP_SHARED	変更点を共有する。
MAP_PRIVATE	変更点はプライベートである。

MAP_FILE および MAP_ANONYMOUS フラグは、マップしようとしているリージョンがマップされたファイル・リージョンなのか、匿名の共用メモリ・リージョンなのかを制御します。いずれかのフラグが選択されている必要があります。

flags 引数の中で MAP_FILE が設定されている場合:

- 新しいマップされたファイル・リージョンが作成され、filedes 引数に関連付けられたファイルがマッピングされる。
- off 引数は、マッピングが開始されるファイル・バイト・オフセットを指定する。このオフセットは、sysconf(_SC_PAGE_SIZE) から返されるページ・サイズの倍数でなくてはならない。
- マップされたファイル・リージョンの終端がファイルの終端よりも後にある場合、ファイルの終端を越えたオフセットに対応するマップされたファイル・リージョンの中のアドレスへのすべての参照の結果は不定となる。

flags 引数の中で MAP_ANONYMOUS が設定されている場合:

- 新しいメモリ・リージョンが作成され、すべてゼロに初期化される。
- filedes 引数が -1 でなければ、mmap 関数は実行に失敗する。

要求されたアドレスが null でなく、リージョンをこのアドレスに置くことが可能であれば、新しいリージョンは要求されたアドレスに配置されます。要求されたアドレスが null であるか、要求されたアドレスにリージョンを置くことができない場合には、MAP_VARIABLE および MAP_FIXED フラグがリージョンの配置を制御します。いずれかのフラグが選択されている必要があります。

flags 引数の中で MAP_VARIABLE が設定されている場合:

- 要求されたアドレスが null であるか、システムがリージョンを要求されたアドレスに置くことができない場合、リージョンはシステムによって選択されたアドレスに配置される。

flags 引数の中で MAP_FIXED が設定されている場合:

- 要求されたアドレスが null でない場合、mmap 関数は、要求されたアドレスがすでに他のリージョンの一部であっても実行に成功する (アドレスが既存のリージョン内にある場合、そのリージョン内のページと、2つのリージョンがオーバーラップする領域内のページに対する効果は、それらがアンマップされた場合と同じにな

る。言い換えると、`addr`と`addr + len`の間でマップされているものはすべてアンマップされる)。

- 要求されたアドレスが `null` で、`MAP_FIXED` が指定されている場合の結果は未定義である。

`MAP_PRIVATE` および `MAP_SHARED` フラグは、マップされたファイルまたは共有メモリ・リージョンに対する変更の可視性を制御します。いずれかのフラグが選択されている必要があります。

`flags` 引数の中で `MAP_SHARED` が設定されている場合:

- リージョンがマップされたリージョンである場合、リージョンに対する変更は、`MAP_SHARED` を使って同じリージョンをマップしている他のプロセスから見える。
- リージョンがマップされたファイル・リージョンである場合、リージョンに対する変更はファイルに書き込まれる (バッファ・キャッシュの遅れのために、変更はただちにファイルに書き込まれるわけではないことに注意すること。つまり、ファイルへの書き込みは、バッファ・キャッシュを再利用する必要があるまでは行われぬ。変更をただちにファイルに書き込む必要がある場合には、`msync`関数を使用する)。

`flags` 引数の中で `MAP_PRIVATE` が設定されている場合:

- 呼び出し元プロセスがマップされたリージョンに加えた変更は、`MAP_PRIVATE` または `MAP_SHARED` を使って同じリージョンをマップしている他のプロセスからは見えない。
- 呼び出し元プロセスがマップされたリージョンに加えた変更は、ファイルには書き込まれない。

リージョンを `MAP_SHARED` を使ってマップしたプロセスによって加えられた変更が、同じリージョンを `MAP_PRIVATE` を使ってマップしている他のプロセスから見えるかどうかは定められていません。

`prot` 引数は、マップされたリージョンに対するアクセス許可を指定します。以下のいずれかを指定します。

<code>PROT_NONE</code>	アクセスなし
<code>PROT_READ</code>	読み込み専用
<code>PROT_WRITE</code>	読み書きアクセス

`mmap` 関数が実行に成功したら、`filedes` 引数は、マップされたリージョンやマップされたファイルの内容に影響を与えずにクローズすることができます。個々のマップされたリージョンは、オープン・ファイル記述子に似たファイル参照を作成し、ファイル・データが割り当て解除されるのを防ぎます。

注意

OpenVMS 固有のファイル参照には、以下の規則が適用されます。

- 追加のファイル参照のために、filedesがファイルの共用を指定してオープンされていなかった場合、mmapはファイル共用を有効にしてファイルを再オープンする。
 - マップされたリージョンに対してファイル参照が追加されている場合、マップされているファイルに対するそれ以降のopen, fopen, またはcreate呼び出しは、ファイル共用を指定しなくてはならない。
-

write関数を使って行われるファイルの変更は、マップされたリージョンから見ることであり、マップされたリージョンに対する変更は、read関数で見ることができません。

注意

OpenVMS Version 7.2 およびそれ以降のmmap関数は、MAP_SHARED 要求を処理する際に、SYS\$CRMPSC サービスのflags引数を、MAP_SHARED 要求を処理するために自分で設定したビットと、呼び出し元がオプションの引数で指定したビットのビット論理和として作成します。

デフォルトでは、mmap関数はMAP_SHARED のために一時的なグループ・グローバル・セクションを作成します。オプションのmmap引数は、呼び出し元に対し、SYS\$CRMPSC システム・サービスの機能への直接のアクセスを提供します。

呼び出し元は、オプションの引数を使用することで、システム・グローバル・セクション (SEC\$M_SYSGBL ビット) や永久的グローバル・セクション (SEC\$M_PERM ビット) などを作成することができます。たとえば、システム永久的グローバル・セクションを作成するには、呼び出し元はオプションの引数で (SEC\$M_SYSGBL | SEC\$M_PERM) を指定します。

mmap関数は、特権のチェックや設定は行いません。呼び出し元は、mmapをオプションの引数を付けて呼び出す前に、SEC\$M_SYSGBL には SYSGBL, SEC\$M_PERM には PRMGBL などの適切な特権を設定する責任を負います。

read, write, open, fopen, creat, およびsysconfも参照してください。

Return value

x

MAP_FAILED

マッピングが配置されるアドレス。

エラーを示します。errnoは以下のいずれかの値に設定されます。

- EACCES—filedesが参照するファイルが読み込みアクセス用にオープンされていないか、ファイルが書き込みアクセス用にオープンされておらず、MAP_SHARED マッピング操作で PROT_WRITE が設定されていた。
- EBADF—filedes引数は有効なファイル記述子でない。
- EINVAL—flagsあるいはprot引数が無効である、または、addrあるいはoff引数がsysconf(_SC_PAGE_SIZE)から返されるページ・サイズの倍数でない。または、flagsで MAP_ANONYMOUS が指定されており、filedesが-1でない。
- ENODEV— ファイル記述子filedesは、ターミナルなどのマップ不可能なオブジェクトを参照している。
- ENOMEM—lenバイトをマップするのに十分なアドレス空間がない。
- ENXIO— 範囲[off, off + len]で指定されたアドレスは、filedesでは無効である。
- EFAULT—addr引数は無効なアドレスである。

modf

浮動小数点数を分解します。

Format

```
#include <math.h>

double modf (double x, double *iptr);
float modff (float x, float *iptr); (Alpha, I64)
long double modfl (long double x, long double *iptr); (Alpha, I64)
```

Argument

x
double, float , またはlong double型のオブジェクト。

iptr
xの型に対応するdouble, float, またはlong double型のオブジェクトへのポインタ。

Description

modf関数は、第 1 引数のxを、それぞれxと同じ符号を持つ、正の小数部fと整数部iに分解します。

この関数はfを返し、第 2 引数 (iptr) がポイントするオブジェクトにiを代入します。

Return value

x	引数xの小数部。
NaN	xはNaN です。errnoは EDOM に設定され、*iptrはNaN に設定されます。
0	アンダフローが発生しました。errnoは ERANGE に設定されます。

[w]move

指定されたウィンドウ上の現在のカーソル位置を、座標 (y, x) に変更します。move関数はstdscrウィンドウに作用します。

Format

```
#include <curses.h>

int move (int y, int x);

int wmove (WINDOW *win, int y, int x);
```

Argument

win
ウィンドウへのポインタ。

y
ウィンドウ座標。

x
ウィンドウ座標。

Description

詳細については、scrollok関数を参照してください。

Return value

OK	成功を示します。
ERR	関数がスクリーンに不正なスクロールを引き起こすことを示します。

mprotect

メモリ・マッピングのアクセス保護を変更します。この関数はリエントラントです。

Format

```
#include <mman.h>

int mprotect (void *addr, size_t len, int prot);
```

Argument

addr

変更しようとしているリージョンのアドレス。

len

変更しようとしているリージョンのバイト長。

prot

<mman.h>ヘッダ・ファイルに定義されているアクセス許可。PROT_NONE, PROT_READ, または PROT_WRITE を指定します。

Description

mprotect関数は、マップされたファイルまたは共用メモリ・リージョンのアクセス保護を変更します。

addrおよびlen引数は、変更しようとしているリージョンのアドレスとバイト長を指定します。len引数は、sysconf(_SC_PAGE_SIZE)から返されるページ・サイズの倍数でなくてはなりません。lenがsysconf(_SC_PAGE_SIZE)から返されるページ・サイズの倍数でなかった場合、リージョンの長さは、ページ・サイズの次の倍数に切り上げられます。

prot引数は、マップされたリージョンのアクセス許可を指定します。以下のいずれかを指定します。

PROT_NONE	アクセスなし
PROT_READ	読み込み専用
PROT_WRITE	読み書きアクセス

mprotect関数は、指定されたリージョンの外に位置するリージョンのアクセス許可を変更しません。ただし、リージョンの終端と、リージョンの終端を含んでいるページの終端の間のアドレスに対する効果は定められていません。

mprotect関数が、EINVALで指定される条件以外の条件で実行に失敗した場合、範囲[addr, addr + len]にある一部のページのアクセス保護は変更される可能性があります。たとえば、addr2にある何らかのページでエラーが起こった場合、mprotectは範囲[addr, addr2]にあるすべてのページの保護を変更する可能性があります。

sysconfも参照してください。

Return value

0	成功を示します。
-1	エラーを示します。errnoは以下のいずれかの値に設定されます。 <ul style="list-style-type: none">• EACCESS—prot引数は、下位のファイルのアクセス許可と矛盾する保護を指定している。• EINVAL—prot引数が無効であるか、addr引数がsysconf(_SC_PAGE_SIZE)から返されるページ・サイズの倍数でない。• EFAULT—範囲[addr, addr + len]に無効なアドレスが含まれている。

mrnd48

一様分布の擬似乱数列を生成します。48 ビットの符号付き long 整数を返します。

Format

```
#include <stdlib.h>

long int mrnd48 (void);
```

Description

mrnd48関数は、線形合同法アルゴリズムと48ビット整数算術演算を使用して、擬似乱数を生成します。

この関数は、 $-2^{31} \leq y < 2^{31}$ の範囲内で一様に分布する、符号付き long 整数を返します。

mrnd48関数を呼び出す前に、srand48, seed48, またはlcong48を使用して乱数ジェネレータを初期化してください。初期化はmrnd48関数を呼び出す前に行わなくてはなりません。mrnd48は、生成された最後の48ビットのXiを内部バッファに格納するからです(これは推奨はされませんが、drand48, lrand48, またはmrnd48関数が、初期化関数を呼び出すことなく呼び出された場合には、定数のデフォルト・イニシライザ値が自動的に提供されます)。

この関数は、次の線形合同式に従って、48ビットの整数値Xiのシーケンスを生成します。

$$X_{n+1} = (aX_n + c) \bmod m \quad n \geq 0$$

引数mは 2^{48} に等しいので、48ビット整数算術演算が実行されます。lcong48関数を呼び出さなかった場合、乗数値aと加算される値cは次のようになります。

$$\begin{aligned} a &= 5DEECE66D_{16} = 2736731631558 \\ c &= B_{16} = 138 \end{aligned}$$

mrnd48関数から返される値は、まずシーケンス内の次の48ビットXiを生成することによって計算されます。その後、返されるデータ項目の型に応じて、適切なビットがXiの最上位ビットからコピーされ、戻り値へと変換されます。

drand48, lrand48, lcong48, seed48, およびsrand48も参照してください。

Return value

n	$-2^{31} \leq y < 2^{31}$ の範囲で一様分布する, 符号付き long 整数を返します。
---	--

msync

マップされたファイルを同期化します。

Format

```
#include <mman.h>

int msync (void *addr, size_t len, int flags);
```

Argument

addr

同期化するリージョンのアドレス。

len

同期化するリージョンのバイト長。

flags

<mman.h>ヘッダ・ファイルに定義されている以下のシンボリック定数のうちの 1 つ。

MS_SYNC	同期キャッシュのフラッシュ
MS_ASYNC	非同期キャッシュのフラッシュ
MS_INVALIDATE	キャッシングされたページの無効化

Description

msync関数は、マップされたファイル・リージョンのキャッシング操作を制御します。msyncを使用すると、以下の操作を行うことができます。

- リージョン内の変更されたページが、ファイルの下位のストレージ・デバイスに転送されるようにする。
- ファイル・システム操作の観点での変更点の可視性を制御する。

addrおよびlen引数は、同期化するリージョンを指定します。len引数は、sysconf(_SC_PAGE_SIZE)から返されるページ・サイズの倍数でなくてはなりません。そうでなければ、リージョンの長さは次のページ・サイズの倍数に切り上げられます。

flags引数の設定は、以下の効果を持ちます。

flags引数	msync 関数の動作
MS_SYNC	システムがすべての I/O 操作を完了するまで返らない。
MS_ASYNC	システムがすべての I/O 操作のスケジューリングを行った後に返る。
MS_INVALIDATE	ページのすべてのキャッシングされたコピーを無効化する。オペレーティング・システムは、次にアプリケーションがページを参照したときに、ページの新しいコピーをファイル・システムから取得しなくてはならない。

各 flags 引数を設定して、msync関数の呼び出しに成功した後の状況は、以下のようになります。

- MS_SYNC – MS_SYNC - マップされたリージョンのそれまでのすべての変更が、プロセスからread引数を使って見えるようになる。それ以前の、write関数を使って行われたファイルの変更は失われる。
- MS_INVALIDATE – write関数を使って行われた、ファイルに対する以前のすべての変更が、マップされたリージョンから見えるようになる。それ以前のマップされたリージョンに対する直接の変更は失われる。

read, write , およびsysconfも参照してください。

Return value

0	成功を示します。
-1	エラーを示します。errnoは以下のいずれかの値に設定されます。 <ul style="list-style-type: none">• EIO— ファイル・システムの読み書きの際に I/O エラーが発生した。• ENOMEM—[addr, addr + len]によって指定された範囲は、プロセスのアドレス空間内では無効である。または、この範囲は 1 つまたは複数のアンマップされたページを指定している。• EINVAL—addr引数は、sysconf(_SC_PAGE_SIZE) から返されるページ・サイズの倍数でない。• EFAULT— 範囲[addr, addr + len]は無効なアドレスを含んでいる。

munmap

マップされたリージョンをアンマップします。この関数はリエントラントです。

Format

```
#include <mman.h>

int munmap (void *addr, size_t len);
```

Argument

addr
アンマップしたいリージョンのアドレス。

len
アンマップしたいリージョンのバイト長。

Description

`munmap`関数は、マップされたファイルまたは共用メモリ・リージョンをアンマップします。

`addr`および`len`引数は、アンマップするリージョンのアドレスとバイト長をそれぞれ指定します。

`len`引数は、`sysconf(_SC_PAGE_SIZE)`から返されるページ・サイズの倍数でなくてはなりません。そうでない場合は、リージョンの長さは次のページ・サイズの倍数に切り上げられます。

アンマップされたリージョンに含まれており、それ以降のどのマップされたリージョンにも含まれていないアドレスを使用した場合の結果は未定義です。

`sysconf`も参照してください。

Return value

0	成功を示します。
-1	エラーを示します。errnoは以下のいずれかの値に設定されます。 <ul style="list-style-type: none">• ENIVAL—addr引数は、sysconf(_SC_PAGE_SIZE)から返されるページ・サイズの倍数でない。• EFAULT—範囲[addr, addr + len]は無効なアドレスを含んでいる。

mv[w]addch

カーソルを座標 (y,x) に移動し, 指定されたウィンドウに文字を追加します。

Format

```
#include <curses.h>

int mvaddch (int y, int x, char ch);
int mvwaddch (WINDOW *win, int y, int x, char ch);
```

Argument

win
ウィンドウへのポインタ。

y
ウィンドウ座標。

x
ウィンドウ座標。

ch
この引数が改行文字 (\n) である場合, mvaddchおよびmvwaddch関数は行を終端までクリアし, カーソルを次の行の同じx座標に移動します。キャリッジ・リターン (\r) は, カーソルを指定された行の先頭に移動します。タブ (\t) は, カーソルをウィンドウ内の次のタブストップに移動します。

Description

このルーチンは, mvwaddchと同じ機能を, stdscrウィンドウに対して実行します。

サブウィンドウに対して使用された場合, mvwaddchは下位のウィンドウに対しても文字を書き込みます。

mv[w]addch

Return value

OK	成功を示します。
ERR	文字の書き込みがスクリーンに不正なスクロールを引き起こすことを示します。詳細については、 <code>scrollok</code> 関数を参照してください。

mv[w]addstr

カーソルを座標 (y,x) に移動し, strがポイントする指定された文字列を, 指定されたウィンドウに追加します。

Format

```
#include <curses.h>

int mvaddstr (int y, int x, char *str);
int mvwaddstr (WINDOW *win, int y, int x, char *str);
```

Argument

win
ウィンドウへのポインタ。

y
ウィンドウ座標。

x
ウィンドウ座標。

str
文字列へのポインタ。

Description

このルーチンは, mvwaddstrと同じ機能を, stdscrウィンドウに対して実行します。

サブウィンドウに対して使用された場合, mvwaddstrは下位のウィンドウに対しても文字列を書き込みます。

Return value

OK	成功を示します。
ERR	関数がスクリーンに不正なスクロールを引き起こすことを示します。ただし、関数はウィンドウ上に可能な限り文字列を追加します。詳細については、 <code>scrollok</code> 関数を参照してください。

mvcur

ターミナルのカーソルを , (lasty,lastx) から (newy,newx) に移動します。

Format

```
#include <curses.h>
int mvcur (int lasty, int lastx, int newy, int newx);
```

Argument

lasty
カーソル位置。

lastx
カーソル位置。

newy
新しいカーソル位置。

newx
新しいカーソル位置。

Description

HP C for OpenVMSシステムでは , mvcurとmoveは同じ機能を実行します。

moveも参照してください。

Return value

OK	成功を示します。
ERR	ウィンドウの移動により , ウィンドウの一部または全体がターミナル・スクリーンから外れることを示します。ターミナル・スクリーンは変更されません。

mv[w]delch

カーソルを座標 (y,x) に移動し、指定されたウィンドウ上の文字を削除します。mvdelch関数はstdscrウィンドウに作用します。

Format

```
#include <curses.h>

int mvdelch (int y, int x);
int mvwdelch (WINDOW *win, int y, int x);
```

Argument

win
ウィンドウへのポインタ。

y
ウィンドウ座標。

x
ウィンドウ座標。

Description

同じ行のそれ以降の文字は左にシフトし、最後の文字は空白になります。

Return value

OK	成功を示します。
ERR	文字の削除がスクリーンに不正なスクロールを引き起こすことを示します。詳細については、scrolllok関数を参照してください。

mv[w]getch

カーソルを座標 (y,x) に移動し、ターミナル・スクリーンから文字を取得し、指定されたウィンドウにエコーします。mvgetch関数はstdscrウィンドウに作用します。

Format

```
#include <curses.h>

int mvgetch (int y, int x);
int mvwgetch (WINDOW *win, int y, int x);
```

Argument

win
ウィンドウへのポインタ。

y
ウィンドウ座標。

x
ウィンドウ座標。

Description

mvgetchおよびmvwgetch関数は、文字をフェッチする前に、指定されたウィンドウをリフレッシュします。

Return value

x	返される文字。
ERR	関数がスクリーンに不正なスクロールを引き起こすことを示します。詳細については、scrollok関数を参照してください。

mv[w]getstr

カーソルを座標 (y,x) に移動し、ターミナル・スクリーンから文字列を取得し、これを変数str (文字列を保持できるだけの大きさでなくてはなりません) に格納し、指定されたウィンドウにエコーします。mvgetstr関数はstdscrウィンドウに作用します。

Format

```
#include <curses.h>

int mvgetstr (int y, int x, char *str);
int mvwgetstr (WINDOW *win, int y, int x, char *str);
```

Argument

win
ウィンドウへのポインタ。

y
ウィンドウ座標。

x
ウィンドウ座標。

str
表示される文字列。

Description

mvgetstrおよびmvwgetstr関数は、文字列から改行文字 (\n) を除去します。

Return value

OK	成功を示します。
ERR	関数がスクリーンに不正なスクロールを引き起こすことを示します。

mv[w]inch

カーソルを座標 (y,x) に移動し、ウィンドウに変更を加えずに、指定されたウィンドウ上の文字を返します。mvinch関数はstdscrウィンドウに作用します。

Format

```
#include <curses.h>

int mvinch  (int y, int x);
int mvwinch (WINDOW *win, int y, int x);
```

Argument

win
ウィンドウへのポインタ。

y
ウィンドウ座標。

x
ウィンドウ座標。

Return value

x	返される文字。
ERR	入力エラーを示します。

mv[w]insch

カーソルを座標 (y,x) に移動し、文字chを指定されたウィンドウに挿入します。mvinsch関数はstdscrウィンドウに作用します。

Format

```
#include <curses.h>

int mvinsch (int y, int x, char ch);
int mvwinsch (WINDOW *win, int y, int x, char ch);
```

Argument

win
ウィンドウへのポインタ。

y
ウィンドウ座標。

x
ウィンドウ座標。

ch
ウィンドウの座標に挿入する文字。

Description

文字の挿入後、行の上のすべての文字は右にシフトし、行の最後の文字は削除されます。

Return value

OK	成功を示します。
ERR	関数がスクリーンに不正なスクロールを引き起こすことを示します。詳細については、scrollok関数を参照してください。

mv[w]insstr

カーソルを座標 (y,x) に移動し, 指定された文字列を指定されたウィンドウに挿入します。mvinsstr関数はstdscrウィンドウに作用します。

Format

```
#include <curses.h>

int mvinsstr (int y, int x, char *str);
int mvwinsstr (WINDOW *win, int y, int x, char *str);
```

Argument

win
ウィンドウへのポインタ。

y
ウィンドウ座標。

x
ウィンドウ座標。

str
表示する文字列。

Description

文字列の後のすべての文字は右にシフトし, 最後の文字は消えます。mvinsstrおよびmvwinsstr関数はHP C for OpenVMSシステムに固有のもので, 移植性はありません。

Return value

OK	成功を示します。
ERR	関数がスクリーンに不正なスクロールを引き起こすことを示します。詳細については, scrolllok関数を参照してください。

mvwin

ウィンドウの開始位置を、指定された (y,x) 座標に移動します。

Format

```
#include <curses.h>

mvwin (WINDOW *win, int y, int x);
```

Argument

win
ウィンドウへのポインタ。

y
ウィンドウ座標。

x
ウィンドウ座標。

Description

サブウィンドウを移動するとき、mvwin関数は、下位ウィンドウ上の新しい位置にサブウィンドウの内容を再表示しません。移動後にサブウィンドウへの書き込みを行った場合、関数は下位ウィンドウにも書き込みを行います。

Return value

OK	成功を示します。
ERR	ウィンドウの移動により、ウィンドウの一部または全体がターミナル・スクリーンから外れることを示します。ターミナル・スクリーンは変更されません。

nanosleep (Alpha, I64)

高精度のスリープです (リアルタイム)。指定された時間だけ、プロセス (プログラムがスレッド化されている場合はスレッド) の実行を中断します。

Format

```
#include <time.h>

int nanosleep (const struct timespec *rqtp, struct timespec *rmtp);
```

Argument

rqtp

呼び出し元のプロセスまたはスレッドを中断する時間を指定する `timespec` 構造体へのポインタ。

rmtp

以前に要求されていた時間の残り時間 (時間が完全に経過した場合は、ゼロ) を受け取る `timespec` データ構造体へのポインタ。

Description

`nanosleep` 関数は、次のいずれかの条件を満たすまで、プロセスまたはスレッドの実行を中断します。

- `rqtp` 引数で指定された時間が経過した。
- 呼び出し元プロセスにシグナルが渡され、その動作がシグナル・キャッチ関数の起動、またはプロセスの終了である。

引数の値がスリープの精度の整数倍に切り上げられたり、システムにより他の動作がスケジューリングされることがあるため、中断時間は、要求された時間よりも長くなることがあります。シグナルによって割り込まれた場合を除き、中断時間は、`rqtp` 引数で指定された時間より短くなることはありません (システム・クロック `CLOCK_REALTIME` で計測されます)。

`nanosleep` 関数を使用しても、シグナルの動作やブロック条件には影響しません。

要求された時間が経過した場合、呼び出しが成功し、`nanosleep` 関数はゼロを返します。

失敗すると、nanosleep関数は-1を返し、失敗を示す値をerrnoに設定します。この関数は、シグナルに割り込まれるか、指定されたrqtp引数が0より小さいか10億以上の場合に失敗します。

rmtp引数がNULLでない場合、この引数が指すtimespec構造体は、時間間隔の残り時間(要求された時間から実際にスリープした時間を引いたもの)を含むようにアップデートされます。

rmtp引数がNULLの場合、残り時間は返されません。

clock_getres, clock_gettime, clock_settime, およびsleepも参照してください。

Return value

- | | |
|----|---|
| 0 | 成功を示します。要求された時間が経過しました。 |
| -1 | 失敗を示します。関数呼び出しが成功しなかったか、シグナルに割り込まれました。次の値のいずれかが、errnoに設定されます。 <ul style="list-style-type: none">• EINTR – nanosleep関数が、シグナルに割り込まれました。• EINVAL – rqtp引数が、0より小さい、または10億以上のナノ秒値を指定していました。 |

newwin

ターミナル・スクリーン上の座標 (begin_y, begin_x) から , numlines行 , numcolsカラムの新しいウィンドウを作成します。

Format

```
#include <curses.h>

WINDOW *newwin (int numlines, int numcols, int begin_y, int begin_x);
```

Argument

numlines

これが 0 の場合 , newwin関数はサイズを LINES (begin_y) に設定します。 LINES x COLS のサイズの新しいウィンドウを作成するには , 次のようにします。

```
newwin (0, 0, 0, 0)
```

numcols

これが 0 の場合 , newwin関数はサイズを COLS (begin_x) に設定します。 LINES x COLS のサイズの新しいウィンドウを作成するには , 次のようにします。

```
newwin (0, 0, 0, 0)
```

begin_y

ウィンドウ座標。

begin_x

ウィンドウ座標。

Return value

x

割り当てられたウィンドウのアドレス。

ERR

エラーを示します。

nextafter (*Alpha, I64*)

xから見てyの向きにある，機械による表現が可能な次の数値を返します。

Format

```
#include <math.h>

double nextafter (double x, double y);
float nextafterf (float x, float y);
long double nextafterl (long double x, long double y);
```

Argument

x
実数。

y
実数。

Description

nextafter関数は，xから見てyの向きにある，機械による表現が可能な次の浮動小数点数を返します。yがxよりも小さい場合，nextafterは，機械による表現が可能なxよりも小さい最大の浮動小数点数を返します。

Return value

n	xから見てyの向きにある，機械による表現が可能な次の浮動小数点値。
HUGE_VAL	オーバーフロー。errnoは ERANGE に設定されます。
NaN	xまたはyが NaN です。errnoは EDOM に設定されます。

nexttoward (Alpha, I64)

「説明」に示す例外を除いて、nextafter関数と同じです。

Format

```
#include <math.h>

double nexttoward (double x, long double y);
float nexttowardf (float x, long double y);
long double nexttowardl (long double x, long double y);
```

Argument

x
実数値。

y
実数値。

Description

nexttoward関数は、次の違いを除いて、対応するnextafter関数と同じです。つまり、第2パラメータの型はlong doubleになっているので、xとyが等しい場合は、yを関数の型に変換して返します。

Return value

n	xから見てy側にあって、機械による表現が可能な次の浮動小数点値です。
y (ただし、型はxと同じ)	xとyが等しい場合の戻り値です。
HUGE_VAL	オーバーフローが発生したことを示します。errnoにはERANGEが設定されます。
NaN	xまたはyがNaNであったことを示します。errnoにはEDOMが設定されます。

nice

プロセスの現在の優先度を基準とし、引数で指定されたレベルだけ優先度を増減します。この関数は非リエントラントです。

Format

```
#include <unistd.h>

int nice (int increment);
```

Argument

increment

正の引数は、優先度を下げます。負の引数は、優先度を上げます。nice(0)を発行すると、基本の優先度に戻されます。結果として得られる優先度は、1よりも小さくはならず、プロセスの基本優先度よりも高くはなりません。この条件が満たされなければ、nice関数は何の処理も行いません。

Description

プロセスがvfork関数を呼び出した場合、結果として作成される子プロセスは親の優先度を継承します。

DECC\$ALLOW_UNPRIVILEGED_NICE 機能論理名が有効になっている場合、nice関数は従来の動作を行い、呼び出し元プロセスの権限をチェックしません(つまり、任意のユーザが、nice値を小さくして、プロセスの優先度を高くすることができます)。また、呼び出し元がMAX_PRIORITYを超える優先度を設定した場合、nice値には、基本の優先度が設定されます。

DECC\$ALLOW_UNPRIVILEGED_NICE が無効になっている場合、nice関数はX/Open 標準に従い、呼び出し元プロセスの権限をチェックします (ALTPRI 特権を持つユーザだけがnice値を小さくし、プロセスの優先度を高くすることができます)。また、呼び出し元がMAX_PRIORITYを超える優先度を設定した場合、nice値にはMAX_PRIORITY が設定されます。

vforkも参照してください。

Return value

0	成功を示します。
-1	失敗を示します。

nint (*Alpha, I64*)

nint (*Alpha, I64*)

引数に最も近い整数値を返します。

Format

```
#include <math.h>
double nint (double x);
float nintf (float x,);
long double nintl (long double x);
```

Argument

x
実数。

Description

nint関数は、xに最も近い整数値を返します。中間の値は、xよりも絶対値が大きい整数値に丸められます。この関数は Fortran の汎用組み込み関数nintに対応します。

Return value

n	xに最も近い整数値。
NaN	xは NaN です。errnoは EDOM に設定されます。

[no]nl

nlおよびnonl関数は、UNIXソフトウェアとの互換性のためにのみ用意されており、OpenVMS環境では機能を持ちません。

Format

```
#include <curses.h>
void nl (void);
void nonl (void);
```

nl_langinfo

プログラムのカレント・ロケールから取得された情報を含んでいる文字列へのポインタを返します。

Format

```
#include <langinfo.h>

char *nl_langinfo (nl_item item);
```

Argument

item
必要な情報を指定する定数の名前。これらの定数は<langinfo.h>に定義されています。

以下の定数が有効です。

定数	カテゴリ	説明
D_T_FMT	LC_TIME	日付と時刻をフォーマットするための文字列
D_FMT	LC_TIME	日付をフォーマットするための文字列
T_FMT	LC_TIME	時刻をフォーマットするための文字列
T_FMT_AMPM	LC_TIME	AM/PM 文字列を含む時刻フォーマット
AM_STR	LC_TIME	AM を 12 時間制で表現する文字列
PM_STR	LC_TIME	PM を 12 時間制で表現する文字列
DAY_1	LC_TIME	週の最初の曜日の名前
...		
DAY_7	LC_TIME	週の 7 番目の曜日の名前
ABDAY_1	LC_TIME	週の最初の曜日の短縮名
...		
ABDAY_7	LC_TIME	週の 7 番目の曜日の短縮名
MON_1	LC_TIME	年の最初の月の名前
...		
MON_12	LC_TIME	年の 12 番目の月の名前
ABMON_1	LC_TIME	年の最初の月の短縮名
...		
ABMON_12	LC_TIME	年の 12 番目の月の短縮名
ERA	LC_TIME	Epoch の記述文字列

定数	カテゴリ	説明
ERA_D_FMT	LC_TIME	Epoch の日付フォーマット文字列
ERA_T_FMT	LC_TIME	Epoch の時刻フォーマット
ERA_D_T_FMT	LC_TIME	Epoch の日付と時刻のフォーマット
ALT_DIGITS	LC_TIME	数字の代替シンボル
RADIXCHAR	LC_NUMERIC	基数文字
THOUSEP	LC_NUMERIC	非金額値で桁のグループを区切るために使われる文字
YESEXp	LC_MESSAGES	yes/no の質問に対する肯定的な応答の表現
NOEXP	LC_MESSAGES	yes/no の質問に対する否定的な応答の表現
CRNCYSTR	LC_MONETARY	通貨シンボル。次のいずれかが前に置かれる: <ul style="list-style-type: none"> シンボルを値の前に置く場合にはマイナス(-) シンボルを値の後に置く場合にはプラス(+) シンボルが基数文字を置き換える場合にはピリオド(.)
CODESET	LC_CTYPE	コードセット名

Description

現在のロケールに言語情報が定義されていない場合、関数は C ロケールから情報を返します。プログラムは、この関数から返された文字列を変更するべきではありません。この文字列は、後の `nl_langinfo` の呼び出しによって上書きされることがあります。

`nl_langinfo` の呼び出しの後に `setlocale` 関数が呼び出された場合、以前の `nl_langinfo` の呼び出しから返されたポインタは無効になります。この場合には、`nl_langinfo` 関数を再度呼び出すようにしてください。

Return value

x 要求された情報を含んでいる文字列へのポインタ。item が無効な場合、関数は空の文字列を返します。

Example

```

#include <stdio.h>
#include <locale.h>
#include <langinfo.h>

/* This test sets up the British English locale, and then      */
/* inquires on the data and time format, first day of the week, */
/* and abbreviated first day of the week.                      */

#include <stdlib.h>
#include <string.h>

int main()
{
    char *return_val;
    char *nl_ptr;

    /* set the locale, with user supplied locale name */
    return_val = setlocale(LC_ALL, "en_gb.iso8859-1");
    if (return_val == NULL) {
        printf("ERROR : The locale is unknown");
        exit(1);
    }
    printf("+-----+\n");

    /* Get the date and time format from the locale. */
    printf("D_T_FMT = ");

    /* Compare the returned string from nl_langinfo with */
    /* an empty string. */
    if (!strcmp((nl_ptr = (char *) nl_langinfo(D_T_FMT)), "")) {
        /* The string returned was empty this could mean that either */
        /* 1) The locale does not contain a value for this item      */
        /* 2) The value for this item is an empty string             */
        printf("nl_langinfo returned an empty string\n");
    }
    else {
        /* Display the date and time format */
        printf("%s\n", nl_ptr);
    }

    /* Get the full name for the first day of the week from locale */
    printf("DAY_1 = ");

    /* Compare the returned string from nl_langinfo with */
    /* an empty string. */
    if (!strcmp((nl_ptr = (char *) nl_langinfo(DAY_1)), "")) {
        /* The string returned was empty this could mean that either */
        /* 1) The locale does not contain a value for the first      */
        /*    day of the week                                         */
        /* 2) The value for the first day of the week is            */
        /*    an empty string                                         */

```



```

        printf("nl_langinfo returned an empty string\n");
    }
    else {
        /* Display the full name of the first day of the week */
        printf("%s\n", nl_ptr);
    }
    /* Get the abbreviated name for the first day of the week
                                   from locale */

    printf("ABDAY_1 = ");

    /* Compare the returned string from nl_langinfo with an empty */
    /* string. */
    if (!strcmp((nl_ptr = (char *) nl_langinfo(ABDAY_1)), "")) {
        /* The string returned was empty this could mean that either */
        /* 1) The locale does not contain a value for the first */
        /*    day of the week */
        /* 2) The value for the first day of the week is an */
        /*    empty string */
        printf("nl_langinfo returned an empty string\n");
    }
    else {
        /* Display the abbreviated name of the first day of the week */
        printf("%s\n", nl_ptr);
    }
}

```

上の例のプログラムを実行すると、次の結果が出力されます。

```

+-----+
D_T_FMT = %a %e %b %H:%M:%S %Y
DAY_1 = Sunday
ABDAY_1 = Sun

```

nrnd48

一様分布の擬似乱数列を生成します。48 ビットの符号付き long 整数を返します。

Format

```
#include <stdlib.h>

long int nrnd48 (unsigned short int xsubi[3]);
```

引数

xsubi
 連結されたときに 48 ビット整数となる，3 つの short int の配列。

Description

nrnd48関数は，線形合同法アルゴリズムと 48 ビット整数算術演算を使用して，擬似乱数を生成します。

nrnd48関数
 は， $0 \leq y < 2^{31}$ の範囲内で一様に分布する，負でない long 整数を返します。

この関数は，次の線形合同式に従って，48 ビットの整数値 X_i のシーケンスを生成します。

$$X_{n+1} = (aX_n + c) \bmod m \quad n \geq 0$$

引数 m は 2^{48} に等しいので，48 ビット整数算術演算が実行されます。lcong48関数を呼び出さなかった場合，乗数値 a と加算される値 c は次のようになります。

$$\begin{aligned} a &= 5DEECE66D_{16} = 2736731631558 \\ c &= B_{16} = 138 \end{aligned}$$

nrnd48関数では，呼び出し元プログラムが，xsubi引数として，最初の呼び出しの時点で擬似乱数列の初期値に初期化された配列を渡す必要があります。drand48関数とは異なり，最初の呼び出しの前に初期化関数を呼び出す必要はありません。

nrnd48では，異なる引数を使用することで，大きなプログラムの個々のモジュールが，複数の互いに独立した擬似乱数列を生成することができます。たとえば，1 つのモジュールが生成する乱数列は，関数が他のモジュールから呼び出された回数には依存しません。

Return value

n $0 \leq y < 2^{31}$ の範囲で一様分布する, 負でない long 整数を返します。

open

読み込み，書き込み，または編集のためにファイルをオープンします。ファイルの位置は先頭 (バイト 0) に設定されます。

Format

```
#include <fcntl.h>

int open (const char *file_spec, int flags, mode_t mode); (ANSI C)
int open (const char *file_spec, int flags, ... ); (HP C Extension)
```

Argument

file_spec

有効なファイル指定を含んでいる，null で終わる文字列。file_spec でディレクトリを指定し，それがエラーを含んだ検索リストだった場合，HP C はそれをファイル・オープン・エラーとして解釈します。

file_spec パラメータの指しているファイルがシンボリック・リンクである場合は，そのシンボリック・リンクで参照されているファイルをオープンします。

flags

<fcntl.h> ヘッダ・ファイルには以下の値が定義されています。

<code>O_RDONLY</code>	読み込み専用でオープンする
<code>O_WRONLY</code>	書き込み専用でオープンする
<code>O_RDWR</code>	読み書き用にオープンする
<code>O_NDELAY</code>	非同期入力用にオープンする
<code>O_APPEND</code>	各書き込みで追加する
<code>O_CREAT</code>	存在しない場合にはファイルを作成する
<code>O_TRUNC</code>	このファイルの新しいバージョンを作成する
<code>O_EXCL</code>	既存のファイルの作成を試みた場合にはエラーとする

これらのフラグは，ビット論理和演算子 (|) で指定されたフラグを区切ることで設定します。

`O_APPEND` を指定してファイルをオープンすると，ファイルに対する個々の書き込みはファイルの終端に付加されます (一方，VAX C RTL では，付加モードでファイルをオープンすると，EOF で書き込みが開始され，その後は現在のファイル位置に書き込みが行われていました)。

O_TRUNCが指定され、ファイルが存在していた場合、openはバージョン番号を 1 だけ増やして新しいファイルを作成し、古いバージョンはそのまま残します。

O_CREATが設定されており、指定されたファイルが存在しなかった場合、HP C RTL は 4 番目とそれ以降の引数 (...) で指定されたすべての属性を持つファイルを作成します。O_EXCLがO_CREATとともに設定されており、指定されたファイルが存在していた場合、オープンの試みはエラーを返します。

mode

ファイル保護モードを指定する符号なしの値。コンパイラは、モードと、現在の保護モードの補数に対してビット論理積演算を実行します。

モードは、ビット論理和演算子(|)で指定されたモードを区切ることによって作成します。以下のモードがあります。

0400	OWNER:READ
0200	OWNER:WRITE
0100	OWNER:EXECUTE
0040	GROUP:READ
0020	GROUP:WRITE
0010	GROUP:EXECUTE
0004	WORLD:READ
0002	WORLD:WRITE
0001	WORLD:EXECUTE

システムには、オーナーと同じアクセス特権が与えられます。WRITE 特権は DELETE 特権も暗黙のうちに含んでいます。

...

オプションのファイル属性引数。ファイル属性引数は、creat関数で 사용되는ものと同じです。詳細については、creat関数を参照してください。

Description

ファイルの何らかのバージョンが存在する場合、openで作成された新しいファイルは、open呼び出しで指定されていない限り、特定の属性を既存のファイルから継承します。継承される属性は、レコード形式、最大レコード・サイズ、キャリッジ・コントロール、およびファイル保護です。

注意

- ファイルに対するランダム書き込みを行う場合、ファイルはO_RDWRのflags値を指定することで、更新用にオープンしなくてはならない。

- UNIX システム・コール関数 `umask`, `mkdir`, `creat`, および `open` を使って, OpenVMS RMS デフォルト保護を指定してファイルを作成するには, `umask` を直接呼び出さないプログラムから, 0777 のファイル保護モード引数を使用して `mkdir`, `creat`, および `open` を呼び出す。これらのデフォルト保護では, ACL やファイルの前のバージョンなどに基づいて, 保護が正しく設定される。

`vfork/exec` 呼び出しを行うプログラムでは, 新しいプロセス・イメージは, `umask` が呼び出されたかどうかの状態を, 呼び出し元のプロセス・イメージから継承します。 `umask` 設定と, `umask` 関数が呼び出されたかどうかの状態は, どちらも属性として継承されます。

`creat`, `read`, `write`, `close`, `dup`, `dup2`, および `lseek` も参照してください。

Return value

<code>x</code>	負でないファイル記述子番号。
<code>-1</code>	ファイルが存在しない, 読み込みまたは書き込みから保護されている, またはその他の理由からオープンできないことを示します。

Example

```
#include <unixio.h>
#include <fcntl.h>
#include <stdlib.h>

main()
{
    int file,
        stat;
    int flags;

    flags = O_RDWR; /* Open for read and write,          */
                  /* with user default file protection, */
                  /* with max fixed record size of 2048, */
                  /* and a block size of 2048 bytes.     */

    file=open("file.dat", flags, 0, "rfm=fix", "mrs=2048", "bls=2048");
    if (file == -1)
        perror("OPEN error"), exit(1);

    close(file);
}
```

opendir

指定されたディレクトリをオープンします。

Format

```
#include <dirent.h>
DIR *opendir (const char *dir_name);
```

Argument

dir_name
オープンするディレクトリの名前。

Description

opendir関数は、dir_nameで指定されたディレクトリをオープンし、それにディレクトリ・ストリームを関連付けます。ディレクトリ・ストリームの位置は最初のエントリに設定されます。<dirent.h>ヘッダ・ファイルに定義されているDIR型は、ディレクトリ・ストリームを表します。ディレクトリ・ストリームとは、特定のディレクトリの中のすべてのディレクトリ・エントリの順序付きのシーケンスです。

また、opendir関数は、それ以降の操作でディレクトリ・ストリームを識別するためのポインタも返します。dir_nameで指定されたディレクトリにアクセスできない場合、またはストリーム全体を保持するのに十分なメモリがない場合には、NULLポインタが返されます。

注意

オープンされたディレクトリは、その次にディレクトリをオープンする試みが成功するように、必ずclosedir関数でクローズされなくてはなりません。opendir関数は、readdir, closedir, およびrewinddir関数と組み合わせて、ディレクトリの内容を確認するために使用します。

Example

closedirの項のプログラム例を参照してください。

Return value

x

DIR型のオブジェクトへのポインタ。

NULL

エラーを示します。errnoは以下のいずれかの値に設定されます。

- EACCES—dir_nameのいずれかのコンポーネントに対する検索許可が拒否されたか、dir_nameに対する読み込み許可が拒否された。
- ENAMETOOLONG—dir_name文字列の長さがPATH_MAXを超えたか、パス名コンポーネントがNAME_MAXよりも長かった。
- ENOENT—dir_name引数が、存在しないファイルの名前をポイントしているか、空の文字列である。

overlay

win1をwin2に対して非破壊的にスーパーインポーズします。この関数は、両方のウィンドウの開始座標を起点として、win1の内容をwin2に書き込みます。win1上の空白は、win2上の対応するスペースの内容を変更せずに残します。overlay関数は、ウィンドウのボックスを可能な限りコピーします。

Format

```
#include <curses.h>

int overlay (WINDOW *win1, WINDOW *win2);
```

Argument

win1
ウィンドウへのポインタ。

win2
ウィンドウへのポインタ。

Return value

OK	成功を示します。
ERR	失敗を示します。

overwrite

win1の内容を破壊的にwin2に書き込みます。

Format

```
#include <curses.h>

int overwrite (WINDOW *win1, WINDOW *win2);
```

Argument

win1
ウィンドウへのポインタ。

win2
ウィンドウへのポインタ。

Description

overwrite関数は、両方のウィンドウの開始座標を起点として、win1の内容をwin2に書き込みます。win1上の空白は、win2上では空白として書き込まれます。この関数は、ウィンドウのボックスを可能な限りコピーします。

Return value

OK	成功を示します。
ERR	エラーを示します。

pathconf

ファイル実装特性を取得します。

Format

```
#include <unistd.h>

long int pathconf (const char *path, int name);
```

Argument

path

ファイルまたはディレクトリのパス名。

name

問い合わせる構成属性。この属性が、path引数で指定されたファイルに適用不可能な場合、pathconf関数はエラーを返します。

Description

pathconf関数により、アプリケーションは、pathで指定されたファイルの下にあるファイル・システムがサポートしている操作の特性を判定することができます。指定されたファイルの読み込み、書き込み、または実行の許可は不要ですが、パス内の、そのファイルに至るまでのすべてのディレクトリを検索する必要があります。

name引数のためのシンボリック値は、<unistd.h>ヘッダ・ファイルに次のように定義されています。

_PC_LINK_MAX	ファイルへのリンク数の最大値。path引数がディレクトリを参照している場合、返される値はディレクトリそのものに適用される。
_PC_MAX_CANON	正規入力行に含まれるバイト数の最大値。これはターミナル・デバイスにのみ適用される。
_PC_MAX_INPUT	入力キューに入れることができる型の数。これはターミナル・デバイスにのみ適用される。
_PC_NAME_MAX	ファイル名に含まれるバイト数の最大値 (末尾の null を含まない)。バイト範囲の値は 13 ~ 255 である。これはディレクトリ・ファイルにのみ適用される。返される値は、ディレクトリ内のファイル名に適用される。

<code>_PC_PATH_MAX</code>	パス名に含まれるバイト数の最大値 (末尾の null を含まない)。この値はつねに 65,535 以下である。これはディレクトリ・ファイルにのみ適用される。返される値は、指定されたディレクトリがワーキング・ディレクトリである場合の相対パス名の長さの最大値である。
<code>_PC_PIPE_BUF</code>	アトミックに書き込まれることが保証されるバイト数の最大値。これは FIFO にのみ適用される。返される値は参照先オブジェクトに適用される。path 引数がディレクトリを参照している場合、返される値は、ディレクトリ内に存在する、または作成することができるすべての FIFO に適用される。
<code>_PC_CHOWN_RESTRICTED</code>	これはディレクトリ・ファイルにのみ適用される。返される値は、ディレクトリ内に存在する、または作成することができるすべての (ディレクトリ以外の) ファイルに適用される。
<code>_PC_NO_TRUNC</code>	NAME_MAX で許されるよりも長いコンポーネント名がエラーを引き起こす場合に、1 を返す。長いコンポーネント名が切り捨てられる場合には、0 (ゼロ) を返す。これはディレクトリ・ファイルにのみ適用される。
<code>_PC_VDISABLE</code>	これはつねに 0 (ゼロ) である。無効化文字は定義されない。これはターミナル・デバイスにのみ適用される。

Return value

<code>x</code>	name で指定された構成属性の、結果として得られた値。
<code>-1</code>	<p>エラーを示します。errno は、以下のいずれかの値に設定されます。</p> <ul style="list-style-type: none"> • EACCES— パス接頭辞のコンポーネントに対して、検索許可が拒否された。 • EINVAL— name 引数は、未知の、または適用不可能な特性を指定している。 • EFAULT— path 引数は無効なアドレスである。 • ENAMETOOLONG— path 文字列の長さが PATH_MAX を超えている、またはパス名コンポーネントが NAME_MAX よりも長い。 • ENOENT— 指定されたファイルが存在しない、または path 引数が空の文字列をポイントしている。 • ENOTDI— path 接頭辞のコンポーネントはディレクトリではない。

pause

シグナル・キャッチ関数の実行かプロセスの終了を引き起こすアクションを持つシグナルが送信されるまで、呼び出し元プロセスを一時停止します。

Format

```
#include <unistd.h>

int pause (void);
```

Description

pause関数は、シグナル・キャッチ関数の実行かプロセスの終了を引き起こすアクションを持つシグナルが送信されるまで、呼び出し元プロセスを一時停止します。

アクションがプロセスの終了だった場合、pauseは返りません。

アクションがシグナル・キャッチ関数の実行だった場合、pauseはそのシグナル・キャッチ関数が返った後に返ります。

Return value

−1

pause関数は、シグナルに割り込まれるまでプロセスの実行をいつまでも一時停止するため、成功を示す戻り値はありません。

pauseが返る場合、戻り値は−1で、errnoはEINTRに設定されます。

pclose

プロセスへのパイプをクローズします。

Format

```
#include <stdio.h>

int pclose (FILE *stream);
```

Argument

stream

以前のpopen関数の呼び出しから返された、オープン・パイプのFILE構造体へのポインタ。

Description

pclose関数は、呼び出し元プログラムと、実行されるシェル・コマンドの間のパイプをクローズします。pcloseは、popenでオープンした任意のストリームをクローズする目的に使用します。pclose関数は、対応するプロセスが終了するのを待ってから、コマンドの終了ステータスとともに返ります。終了ステータスの解釈については、waitpidの説明を参照してください。

OpenVMS Version 7.3-1 から、_VMS_WAITマクロを定義してコンパイルした場合、pclose関数は子プロセスの OpenVMS 完了コードを返すようになりました。

popenも参照してください。

Return value

x

子プロセスの終了ステータス。

−1

エラーを示します。stream引数はpopen関数に関連付けられていません。errnoは以下の値に設定されます。

- ECHILD— 子プロセスのステータスを取得できない。

perror

stderrに、errnoの現在の値を記述する短いエラー・メッセージを書き込みます。

Format

```
#include <stdio.h>

void perror (const char *str);
```

Argument

str
通常は、エラーを引き起こしたプログラムの名前。

Description

perror関数は、外部変数errnoのエラー番号を使用して、適切なロケール依存のエラー・メッセージを取得します。この関数が出力するメッセージは、str (ユーザ提供のエラー・メッセージに対する接頭辞)、コロンとスペース、メッセージそのもの、そして改行文字から構成されます。

発生する可能性のあるエラーのリストについては、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第4章のerrnoの説明を参照してください。

strerrorも参照してください。

Example

```
#include <stdio.h>
#include <stdlib.h>

main(argc, argv)
    int argc;
    char *argv[];
{
    FILE *fp;

    fp = fopen(argv[1], "r");    /* Open an input file. */
    if (fp == NULL) {
```

perror

```
/* If the fopen call failed, perror prints out a      */
/* diagnostic:                                         */
/*                                                     */
/* "open: <error message>"                             */
/* This error message provides a diagnostic explaining */
/* the cause of the failure.                           */
perror("open");
exit(EXIT_FAILURE);
}
else
    fclose(fd) ;
}
```

pipe

親プロセスと子プロセスの間でデータの読み書きに使用できる一時的なメールボックスを作成します。プロセスが通信に使用するチャンネルは、パイプと呼ばれます。

Format

```
#include <unistd.h>

int pipe (int array_fdscptr[2]); (ISO POSIX-1)
int pipe (int array_fdscptr[2], ... ); (HP C Extension)
```

Argument

array_fdscptr

ファイル記述子の配列。パイプは、メールボックスに関連付けられたファイル記述子の配列として実装されます。これらのメールボックス記述子は、isapipe関数に渡されたときに1を返すファイル記述子であるという点で特殊な存在です。

ファイル記述子は、次のように割り当てられます。

- 最初の利用可能なファイル記述子書き込みに割り当てられ、次の利用可能なファイル記述子が読み込みに割り当てられる。
- その後、ファイル記述子は配列に逆順に格納される。要素0には読み込みのためのファイル記述子が格納され、要素1には書き込みのためのファイル記述子が格納される。

...

3つのオプションの定位置引数flag, bufsize, およびbufquotaを表します。

flag

ビットマスクとして使用されるオプションの引数。

O_NDELAYまたはO_NONBLOCKビットが設定されていると、array_fdscptrファイル記述子を通してのメールボックスへのI/O操作は、他のプロセスを待つのではなく、ただちに終了します。

たとえば、O_NDELAYビットが設定されており、親プロセスがメールボックスにデータを入れる前に、子プロセスがメールボックスに対するread要求を発行すると、readは0ステータスでただちに終了します。O_NDELAYビットもO_NONBLOCKビットも設定されていない場合は、子プロセスは、親プロセスがメールボックスにデータを書き込むまで待ってから読み込みを行います。これは、flag引数が指定されなかった場合のデフォルトの動作です。

O_NDELAY と O_NONBLOCK の値は<fcntl.h>ヘッダ・ファイルに定義されています。flag引数の中のその他のビットはすべて無視されます。この引数は、第2のオプションの定位置引数bufsizeが指定されている場合には、必ず指定する必要があります。bufsize引数を指定するためのだけの目的にflag引数を指定する場合には、flagとして0を指定してください。

bufsize

メールボックスのサイズをバイト単位で指定するint型の引数 (オプション)。512 ~ 65535 の値を指定します。

この引数に0を指定するか、この引数を省略すると、オペレーティング・システムはデフォルトのサイズ (512 バイト) のメールボックスを作成します。

0 より小さい、または 65535 より大きい値を指定すると、予期できない結果となります。

この引数を指定する場合は、flag引数を前に指定してください。

DECC\$PIPE_BUFFER_SIZE 機能論理名を使用しても、メールボックスのサイズを指定できます。bufsizeを指定すると、DECC\$PIPE_BUFFER_SIZE の値よりも優先されます。指定しないと、DECC\$PIPE_BUFFER_SIZE の値が使用されます。

bufsizeと DECC\$PIPE_BUFFER_SIZE のどちらも指定しなかった場合は、デフォルトのバッファ・サイズ 512 が使用されます。

bufquota

パイプのメールボックスのバッファ・クォータを指定するint型の引数 (オプション)。512 ~ 2147483647 の値を指定します。

OpenVMS Version 7.3-2 で、この引数が追加されました。以前のバージョンの OpenVMS では、バッファ・クォータとバッファ・サイズは同じでした。

DECC\$PIPE_BUFFER_QUOTA 機能論理名を使用しても、バッファ・クォータを指定できます。pipe関数のbufquotaオプション引数を指定すると、DECC\$PIPE_BUFFER_QUOTA の値よりも優先されます。指定しないと、DECC\$PIPE_BUFFER_QUOTA の値が使用されます。

bufquotaと DECC\$PIPE_BUFFER_QUOTA のどちらも指定しなかった場合、バッファ・クォータは以前と同じように、デフォルトでバッファ・サイズとなります。

Description

パイプのために使われるメールボックスは、一時的なメールボックスです。メールボックスは、そのメールボックスに対するチャンネルをオープンしているすべてのプロセスがそれらのチャンネルをクローズするまでは削除されません。パイプを最後にクロー

ズするプロセスは、メールボックスに対して、ファイルの終端を示すメッセージを書き込みます。

メールボックスは、以下の特性を指定して、\$CREMBX システム・サービスを使って作成されます。

- 最大メッセージ長は 512 文字
- バッファ・クォータは 512 文字
- USER および GROUP にすべての特権を与え、SYSTEM または WORLD には何の特権も与えない保護マスク

512 文字のバッファ・クォータは、メールボックスの全体または一部の読み込みを行うまで、メールボックスに 512 文字以上の書き込みを行えないということを意味しています。メールボックス・レコードは、それが含んでいるメッセージのデータ部よりも若干大きいので、512 文字すべてをメッセージ・データに使用できるわけではありません。バッファのサイズは、pipe関数のオプションの第 3 引数によって代替サイズを指定することで増やすことができます。OpenVMS システムにおけるパイプは、キャリッジ・コントロール属性を持たないストリーム型のファイルです。HP C RTL では、これはデフォルトで完全にバッファリングされます。パイプとして使用されるメールボックスは、アプリケーションによって作成されるメールボックスとは異なります。アプリケーションによって作成されるメールボックスは、デフォルトでは、キャリッジ・リターンとキャリッジ・コントロールを含むレコード型のファイルになります。さらに、メールボックスに長さゼロのレコードを書き込んだとき、メールボックスがクローズされるたびに、EOF が書き込まれます。一方、パイプでは、パイプの最後のクローズのみが EOF を書き込みます。

パイプは、vforkとexec関数が呼び出される前に、親プロセスによって作成されます。子プロセスは、まずpipeを呼び出すことで、パイプのオープン・ファイル記述子を継承します。その後、getname関数を使って、必要ならばパイプに関連付けられているメールボックスの名前を返すことができます。getnameから返されるメールボックス名は、nnnnまたはnnnnnを一意的な数として、_MBAnnnn: (*Alpha only*)または_MBAnnnnn: (*I64 only*) の形式を持っています。

親と子の両方が、パイプに対してどのファイル記述子が割り当てられるのかを事前に知る必要があります。この情報は実行時に取得することはできません。このため、HP C for OpenVMSプログラムにおいてファイル記述子がどのように使われるかを理解しておくことが重要となります。ファイル記述子の詳細については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル(上巻)』第 2 章を参照してください。

ファイル記述子 0、1、および 2 は、HP C for OpenVMSプログラムでは、それぞれstdin (SYSS\$INPUT), stdout (SYSS\$OUTPUT), およびstderr (SYSS\$ERROR) に対してオープンされています。このため、pipeが呼び出されたときに他のファイルがオープンされていない場合、pipeはファイル記述子 3 を書き込み用に、ファイル記述子

4 を読み込み用に割り当てます。pipe から返される配列では、要素 0 に 4 が、要素 1 に 3 が格納されます。

他のファイルがオープンされている場合、pipe は最初の利用可能なファイル記述子を書き込み用に、次の利用可能なファイル記述子を読み込み用に割り当てます。この例では、パイプは必ずしも隣接するファイル記述子を使用しません。たとえば、2 つのファイルがオープンされ、ファイル記述子 3 と 4 が割り当てられた後に、前者のファイルがクローズされたとします。この時点で pipe が呼び出されると、ファイル記述子 3 が書き込み用に、ファイル記述子 5 が読み込み用に割り当てられます。配列の要素 0 には 5 が、要素 1 には 3 が格納されます。

大量の I/O を行う大きなアプリケーションでは、パイプにどのファイル記述子が割り当てられるかを予測するのがさらに困難になります。しかし、子プロセスは、どのファイル記述子が使われるかを知らないで、パイプの読み書きを正しく行うことができません。

正しいファイル記述子が使われるようにするには、次の手順を使用する方法があります。

1. 親と子の両方が知っている 2 つの記述子番号を選択する。これらの番号は、パイプが作成される前に行われる I/O を考慮に入れて、十分に大きな値でなくてはならない。
2. 親プロセスの中で、exec 関数を呼び出す前のいずれかの時点で pipe を呼び出す。
3. 親プロセスの中で、dup2 を使用して、pipe から返されたファイル記述子を、上で選んだファイル記述子に割り当てる。これにより、これらのファイル記述子がパイプ用に予約される。それ以降の I/O は、パイプと干渉することはない。

UNIX I/O 関数の read と write を使用し、適切なファイル記述子を指定することで、パイプを通しての読み書きを行うことができます。別の方法として、fdopen 呼び出しを発行して、これらのファイル記述子にファイル・ポインタを関連付ければ、標準 I/O 関数 (fread と fwrite) を使用できるようになります。

パイプの読み込みと書き込みには 2 つの異なるファイル記述子を使用されますが、使用されるメールボックスは 1 つだけなので、何らかの I/O の同期化が必要となります。たとえば、親プロセスがパイプにメッセージを書き込んだとします。親プロセスがパイプから読み込みを行う最初のプロセスだった場合には、図 REF-1 に示すように、自分のメッセージを読み戻すことになります。

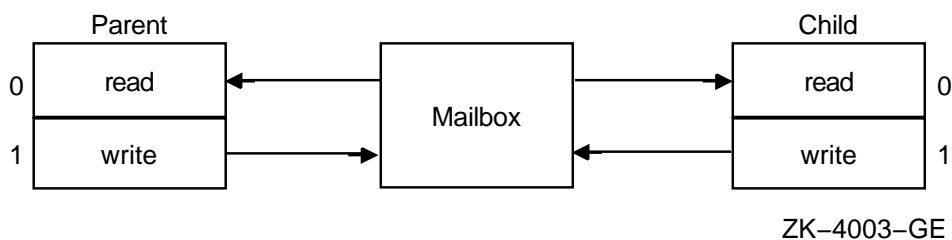
注意

UNIX との互換性を高めるため、以下の機能論理名を使用して、C RTL のパイプ処理の動作を制御することができます。

- DECC\$STREAM_PIPE 機能論理名に ENABLE を定義すると、pipe 関数がレコード入出力ではなくストリーム入出力を使用ようになります。

- DECC\$POpen_NO_CRLF_REC_ATTR 機能論理名に ENABLE を定義すると、popen関数でオープンしたパイプのパイプ・レコードに、CR/LFのキャリッジ制御が追加されなくなります。この機能を有効にすると、getsのようなキャリッジ・リターン文字に依存している関数で、望ましくない動作になる可能性がある点に注意してください。

図 REF-1 パイプの読み込みと書き込み



Return value

0	成功を示します。
-1	エラーを示します。

poll (Alpha, I64)

複数のファイル記述子で複数のオープン・ストリームを参照している場合に、それらのファイル記述子に入出力を多重化するためのメカニズムを提供します。

Format

```
#include <poll.h>

int poll (struct pollfd filedes [], nfds_t nfds, int timeout);
```

引数

filedes

pollfd構造体の配列を指すポインタ。構造体の一つ一つが、対象となるファイル記述子にそれぞれ対応しています。pollfd構造体は、次のメンバから構成されています。

int fd — ファイル記述子
int events — 要求する (つまり報告の対象となる) 条件状況 (イベント)
int revents — 報告される成立条件/状況 (イベント)

nfds

filedes配列を構成するpollfd構造体の数。

timeout

指定したイベントが少なくとも 1 つ発生するまでに待つことのできる最大時間 (単位はミリ秒)。

Description

poll関数は、複数のファイル記述子で複数のオープン・ストリームを参照している場合に、それらのファイル記述子へ入出力を多重化するためのメカニズムとして使用できます。pollは、filedesが指す配列の各メンバごと、つまりファイル記述子ごとに、eventsで指定したイベントが発生しているかどうかを調べます。具体的には、poll関数は、アプリケーションがメッセージを送信または受信できるストリームや、あるイベントが発生したストリームを調べます。

対象にするファイル記述子と、各ファイル記述子に対して調べたいイベントは、filedesパラメータで指定します。このパラメータはポインタになっていて、pollfd構造体の配列を指しています。対象となるオープン・ファイルは、このpollfd構造体の fdメンバとして、ファイル記述子で示します。poll関数は、eventsメンバを使用してそのファイル記述子の報告すべき条件/状況を調べ、その

1 つまたは複数が真になっていれば、reventsメンバに、対応する成立条件/状況を設定します。

pollfd構造体のeventsメンバと reventsメンバは、ビットマスクになっています。eventsの値は呼び出しプロセス側が、また、reventsの値はpoll側がそれぞれ設定します。これらのビットマスクは、条件/状況項目を論理和でまとめたものになっています。条件/状況項目には、次のものがあります。

POLLERR — そのファイル記述子でエラーが発生した。このオプションは、reventsビットマスクでだけ有効で、eventsメンバでは指定できません。

対象が STREAMS デバイスの場合は、そのファイル記述子でエラーが発生しているという状況と、デバイスが切断されているという状況が重なっていると、pollから、POLLHUPではなく POLLERR が返されます。つまり、POLLERR が POLLHUP より優先されます。

POLLHUP — デバイスが切断された。このイベントと POLLOUT は排他関係にあって、同時に設定されることはありません。ハングアップが発生すると、そのストリームへ書き込めなくなります。このイベントは、POLLIN、POLLRDNORM、POLLRDBAND、または POLLPRI と一緒に設定されることがあります。この条件/状況項目は、reventsビットマスクでだけ有効です。eventsメンバにこの条件/状況項目を指定しても、無視されます。

POLLIN — 優先順位の低いデータを、ブロックしないで読み取れる。この条件/状況項目は、メッセージの長さがゼロであっても、reventsに設定されません。reventsにこの条件/状況項目と POLLPRI が同時に設定されることはありません。

POLLNVAL — fdに指定されている値が無効である。この条件/状況項目は、reventsメンバでだけ有効です。eventsメンバにこの条件/状況項目を指定しても、無視されます。

POLLOUT — 通常 (優先順位帯域が 0) のデータを、ブロックしないで書き込める。

POLLPRI — 優先順位の低いデータを、ブロックしないで受信できる。この条件/状況項目は、メッセージの長さがゼロであっても、reventsに設定されません。reventsにこの条件/状況項目と POLLIN が同時に設定されることはありません。

POLLRDBAND — 優先順位帯域がゼロでないデータを、ブロックしないで読み取れる。この条件/状況項目は、メッセージの長さがゼロであっても、reventsに設定されます。

POLLRDNORM — 通常 (優先順位帯域が 0) のデータを, ブロックしないで読み取れる。この条件/状況項目は, メッセージの長さがゼロであっても, revents に設定されます。

POLLWRBAND — 優先データ (優先順位帯域が 0 より大きいデータ) を書き込める。この条件/状況項目でチェックされるのは, 書き込みが 1 度でも行われたことのある帯域だけです。

POLLWRNORM — POLLOUT と同じです。

poll 関数は, fd メンバの値がゼロ (0) より小さいと, その pollfd 構造体を無視します。また, すべての pollfd 構造体の fd メンバが 0 より小さいと, 0 を返すだけで, その他の処理は行いません。

条件/状況項目の POLLNORM と POLLOUT が真になるのは, 少なくとも 1 バイトのデータがブロックしないで読み取れるか書き込める場合だけです。ただし, 例外が 2 つあります。1 つは通常ファイルの場合で, POLLNORM と POLLOUT のポーリング結果は必ず真になります。もう 1 つはパイプで, ファイルの終わりを示すためにゼロを返すという処理規則になっている場合です。

条件/状況項目の POLLERR, POLLHUP, および POLLNVAL は, 指定されたファイル記述子でその条件が成立していれば, events にその条件が設定されているかどうかに関係なく, 必ず revents に設定されます。

poll 関数に対して報告される条件/状況項目には, 2 つのタイプがあります。1 つは, 必ず報告される条件/状況項目で, もう 1 つは, events で指定されたときに報告される条件/状況項目です。poll 関数を呼び出すと, これら 2 つのタイプが一緒に報告されます。poll 関数は, これらの条件/状況項目が成立しているファイル記述子があると, そのファイル記述子の revents にそれらの条件/状況項目が成立していることをすべて示して, 呼び出し側に制御を戻します。

poll 関数は, 報告すべき条件/状況項目があってもそれらがまだ 1 つも成立していないと, その条件/状況項目が成立するまで, 最大 timeout ミリ秒待ちます。そして, どのファイル記述子であるかに関係なくこの待ち時間内に 1 つでも条件/状況項目が成立すれば, そのファイル記述子の revents メンバにその条件/状況項目を設定して戻ります。しかし, 待ち時間が過ぎてもそれらの条件/状況項目が成立しないと, poll は, revents ビットマスクを設定しないで戻ります。

timeout パラメータの値として -1 を指定すると, poll 関数は, 指定したイベントが少なくとも 1 つ発生するまで戻りません。timeout パラメータの値として 0 を指定すると, poll 関数は, 指定したイベントが発生していなくても, そのイベントの発生を待たないで, すぐに戻ります。

poll関数の動作は、指定したファイル記述子に O_NONBLOCK オプションが設定されているかどうかとはいっさい関係ありません。

poll関数では、通常ファイル、ターミナル・デバイス、擬似ターミナル・デバイス、STREAM ベース・ファイル、FIFO、およびパイプをサポートしています。これらとはタイプの異なるファイルを指定、つまりタイプの異なるファイル記述子を指定してpollを行った場合の動作は、規定されていません。

ファイル記述子でソケットを指定して接続をリッスンしている場合は、接続が利用できるようになった時点で読み取れるようになります。また、ファイル記述子でソケットを指定して非同期型の接続を行う場合は、接続が確立した時点で書き込めるようになります。

Return value

n	成功したことを示します。n (負でない値) は、pollによって revents ビットマスクが設定されているファイル記述子の数です。
0	pollがタイムアウトしたことを示します。revents ビットマスクは設定されていません。
-1	エラーが発生したことを示します。errnoに、エラーを示す次のいずれかの値が設定されます。 <ul style="list-style-type: none"> • EAGAIN — 内部データ構造体を割り当てられませんでした。pollを再度呼び出せば、成功する可能性があります。 • EINTR — poll関数の処理中にシグナルが発生したが、そのシグナル・ハンドラでこの関数を再起動しないようになっていました。 • EINVAL — nfds/パラメータの値が OPEN_MAX より大きい、fdメンバのいずれかがストリームまたはマルチプレクサを参照していて、そのストリームまたはマルチプレクサが別のマルチプレクサの下流側に直接または間接的にリンクされています。

popen

プロセスに対するパイプを開始します。

Format

```
#include <stdio.h>

FILE *popen (const char *command, const char *type);
```

Argument

command

シェル・コマンド・ラインを含んでいる， null で終わる文字列へのポインタ。

type

I/O モードを含んでいる， null で終わる文字列へのポインタ。オープン・ファイルは共用されるので， type *r* コマンドを入力フィルタとして， type *w* コマンドを出力フィルタとして使用することができます。type 引数としては，以下のいずれかの値を使用します。

- *r*— 呼び出し元プログラムは，返されたファイル・ストリームから読み込みを行うことで，コマンドの標準出力から読み込みを行うことができる。
- *w*— 呼び出し元プログラムは，返されたファイル・ストリームに書き込みを行うことで，コマンドの標準入力に書き込みを行うことができる。

Description

popen関数は，呼び出し元プログラムと，実行を待っているシェル・コマンドの間にパイプを作成します。関数はストリームのためのFILE構造体へのポインタを返します。

popen関数は，DECC\$PIPE_BUFFER_SIZE 機能論理名の値を使用して，パイプ用に作成するメールボックスのバッファ・サイズを設定します。DECC\$PIPE_BUFFER_SIZE の値には，512 ~ 65024 バイトを指定できます。DECC\$PIPE_BUFFER_SIZE が指定されていない場合，デフォルトのバッファ・サイズ 512 が使用されます。

注意

- popen関数を使って出力フィルタを呼び出す場合には、出力データがプログラム・バッファに残るために生じるデッドロックの可能性に注意するようにしてください。これは、setvbuf関数を使って出力ストリームがバッファリングされないようにするか、fflush関数を使って、pclose関数を呼び出す前にすべてのバッファリングされたデータをフラッシュさせることによって避けることができます。
- UNIX との互換性を高めるため、以下の機能論理名を使用して、C RTL のパイプ処理の動作を制御することができます。
 - DECC\$STREAM_PIPE 機能論理名に ENABLE を定義すると、pipe関数がレコード入出力ではなくストリーム入出力を使用ようになります。
 - DECC\$POPEN_NO_CRLF_REC_ATTR 機能論理名に ENABLE を定義すると、popen関数でオープンしたパイプのパイプ・レコードに、CR/LF のキャリッジ制御が追加されなくなります。この機能を有効にすると、getsのようなキャリッジ・リターン文字に依存している関数で、望ましくない動作になる可能性がある点に注意してください。

fflush, pclose, およびsetvbufも参照してください。

Return value

x	オープンされたストリームのためのFILE構造体へのポインタ。
NULL	エラーを示します。ファイルまたはプロセスを作成することができませんでした。

pow

第 1 引数の , 第 2 引数のべき乗を返します。

Format

```
#include <math.h>

double pow (double x, double y);
float powf (float x, float y); (Alpha, I64)
long double powl (long double x, long double y); (Alpha, I64)
```

Argument

x
指数yでべき乗を計算するときの基数として使用される浮動小数点数。

y
基数xのべき乗を計算するために使用する指数。

Description

pow関数は、浮動小数点数の基数xと、浮動小数点数の指数yを使ってべき乗を計算します。pow(x,y) の値は、正のxでは $e^{y \ln(x)}$ として計算されます。

xが 0 で、yが負の値である場合は ±HUGE_VAL が返され、errnoに ERANGE または EDOM が設定されます。

Return value

x	第 1 引数の , 第 2 引数のべき乗を計算した結果。
1.0	基数が 0 で、指数が 0 だった場合。
HUGE_VAL	結果がオーバーフローしました。errnoは ERANGE に設定されます。
±HUGE_VAL	基数が 0 で、指数が負の値でした。errnoには ERANGE または EDOM が設定されます。

Example

```
#include <stdio.h>
#include <math.h>
#include <errno.h>

main()
{
    double x;

    errno = 0;

    x = pow(-3.0, 2.0);
    printf("%d, %f\n", errno, x);
}
```

この例のプログラムは、次の出力を生成します。

```
0, 9.000000
```

pread (*Alpha, I64*)

ファイル・ポインタを変更せずに、ファイル内の、指定された位置からバイトを読み込みます。

Format

```
#include <unistd.h>

ssize_t read (int file_desc, void *buffer, size_t nbytes, off_t offset);
```

Argument

file_desc
現在読み込み用にオープンされているファイルを指すファイル記述子。

buffer
入力データが格納される，連続領域のアドレス。

nbytes
読み込み操作を行う最大バイト数。

offset
ファイル内の読み込み位置を指定するオフセット。

Description

pread関数は、ファイル・ポインタを変更せずにファイル内の指定された位置から読み込みを行うことを除き、readと同じ動作を行います。preadの最初の3つの引数は、readと同じです。ファイル内の読み込み位置を指定するために、4番目の引数offsetが追加されています。シーク不能のファイルに対してpreadを行おうとすると、エラーとなります。

Return value

n	読み込むバイト数。
---	-----------

-1

失敗すると、ファイル・ポインタは変更されず、preadは以下のいずれかの値をerrnoに設定します。

- EINVAL – offset 引数が不正です。値が負です。
- EOVERFLOW – このファイルは通常のファイルで、ファイルに対応する最大オフセットの位置またはそれを越えた位置で、読み込みや書き込みが行われようとなりました。
- ENXIO – デバイスの能力外の要求でした。
- ESPIPE – ファイル記述子が、パイプまたは FIFO に対応しています。

printf

標準出力 (stdout) に書式設定された出力を行います。書式指定子については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 2 章を参照してください。

Format

```
#include <stdio.h>

int printf (const char *format_spec, ... );
```

Argument

`format_spec`

出力にそのまま書き込まれる，または ... 引数の指定に従って変換して書き込まれる文字。

...

書式指定で指定された変換指定に対応する型を持つ，オプションの式。

変換指定が与えられなかった場合には，出力ソースは省略することができます。それ以外の場合は，関数呼び出しは変換指定と同じ数だけの出力ソースを持たなくてはならず，変換指定は出力ソースの型と一致していなくてはなりません。

変換指定は，左から右の順序で出力ソースと対応づけられます。余分な出力ポインタが存在する場合には，無視されます。

Return value

`x`

書き込まれたバイト数。

負の値

出力エラーが発生したことを示します。関数は `errno` を設定します。この関数が設定する `errno` 値のリストについては，`fprintf` を参照してください。

[w]printw

カーソルの現在位置から開始して、指定されたウィンドウ内でprintfを実行します。printw関数はstdscrウィンドウに作用します。

Format

```
#include <curses.h>

printw (char *format_spec, ... );
int wprintw (WINDOW *win, char *format_spec, ... );
```

Argument

win

ウィンドウへのポインタ。

format_spec

書式指定文字列へのポインタ。

...

書式指定で与えられた変換指定に対応する型を持つオプションの式。

変換指定が与えられなかった場合には、出力ソースは省略することができます。それ以外の場合は、関数呼び出しは変換指定と同じ数だけの出力ソースを持たなくてはならず、変換指定は出力ソースの型と一致していなくてはなりません。

変換指定は、左から右の順序で出力ソースと対応づけられます。余分な出力ポインタが存在する場合には、無視されます。

Description

書式指定 (format_spec) とその他の引数は、printf関数で 사용되는ものと同じです。

printw関数とwprintw関数は、書式指定の結果として得られた文字列を、addstr関数を使ってウィンドウに入力します。詳細については、このセクションのprintfおよびscroll関数を参照してください。書式指定子の詳細については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第2章を参照してください。

Return value

OK	成功を示します。
ERR	関数がウィンドウに不正なスクロールを引き起こすことを示します。

putc

putcマクロは、指定されたファイルに 1 つの文字を書き込みます。

Format

```
#include <stdio.h>

int putc (int character, FILE *file_ptr);
```

Argument

character

書き込む文字。

file_ptr

出力ストリームへのファイル・ポインタ。

Description

putcマクロは、バイトcharacterを (unsigned char に変換後) file_ptrパラメータで指定された出力に書き込みます。バイトはファイル・ポインタが現在指している位置 (定義されている場合) に書き込まれ、指示子が適切に進められます。ファイルが位置設定要求をサポートしていない場合や、出力ストリームが追加モードでオープンされている場合は、バイトは出力ストリームに追加されます。

putcはマクロなので、副作用を持つファイル・ポインタ引数 (例: putc (ch, *f++)) は間違っって評価される可能性があります。このような場合には、代わりにfputc関数を使用してください。

__UNIX_PUTC マクロを定義してコンパイルすれば、この関数の高速インライン版を使用して、最適化できます。

putc_unlockedも参照してください。

Return value

x

ファイルに書き込まれた文字。成功を示します。

putc

EOF

出力エラーを示します。

putc_unlocked (Alpha, I64)

putcマクロと同様ですが、flockfileとfunlockfileで保護された範囲内だけで使用します。

Format

```
#include <stdio.h>

int putc_unlocked (int character, FILE *file_ptr);
```

引数

character
書き込む文字。

file_ptr
出力ストリームへのファイル・ポインタ。

Description

リエントラント版であるputcマクロは、複数スレッドからの同時呼び出しに対してロックされます。その結果、ストリームの一貫性を保証するためのオーバーヘッドが生じます。アンロック版であるputc_unlockedを使用すると、このオーバーヘッドを避けることができます。putc_unlockedマクロはputcマクロと機能的に同じですが、スレッド・セーフな方法で実装する必要がない点が異なります。putc_unlockedマクロは、flockfile関数とfunlockfile関数を対で使用して保護された範囲内でだけ、安全に使用することができます。呼び出し元は、putc_unlockedを使用する前に、ストリームを確実にロックする必要があります。

putc_unlockedはマクロであるため、副作用のあるファイル・ポインタ引数は正しく評価されないことがあります。このような場合は、代わりにfputc_unlocked関数を使用してください。

__UNIX_PUTC マクロを定義してコンパイルすれば、この関数の高速インライン版を使用して、最適化できます。

flockfile、ftrylockfile、およびfunlockfileも参照してください。

Return value

x	ファイルに書き込まれた文字。成功を示します。
EOF	ファイルの終端 (EOF) またはエラーを示します。

putchar

標準出力 (stdout) に 1 つの文字を書き込み，その文字を返します。

Format

```
#include <stdio.h>
int putchar (int character);
```

Argument

character
int型のオブジェクト。

Description

putchar関数は，fputc (character, stdout)と同じです。

__UNIX_PUTC マクロを定義してコンパイルすれば，この関数の高速インライン版を使用して，最適化できます。

Return value

character	成功を示します。
EOF	出力エラーを示します。

putchar_unlocked (Alpha, I64)

putchar関数と同様ですが、flockfileとfunlockfileで保護された範囲内だけで使用します。

Format

```
#include <stdio.h>

int putchar_unlocked (int character);
```

引数

character
int型のオブジェクト。

Description

リエントラント版であるputchar関数は、複数スレッドからの同時呼び出しに対してロックされます。その結果、出力ストリームの一貫性を保証するためのオーバーヘッドが生じます。アンロック版であるputchar_unlockedを使用すると、このオーバーヘッドを避けることができます。putchar_unlocked関数は、putchar関数と機能的に同じですが、スレッド・セーフな方法で実装する必要がない点が異なります。putchar_unlocked関数は、flockfile関数とfunlockfile関数を対で使用して保護された範囲でだけ、安全に使用することができます。呼び出し元は、putchar_unlockedを使用する前に、ストリームを確実にロックする必要があります。

__UNIX_PUTC マクロを定義してコンパイルすれば、この関数の高速インライン版を使用して、最適化できます。

flockfile、ftrylockfile、およびfunlockfileも参照してください。

Return value

x	ファイルに書き込まれた文字。成功を示します。
EOF	ファイルの終端 (EOF) またはエラーを示します。

putenv

環境変数を設定します。

Format

```
#include <stdlib.h>

int putenv (const char *string);
```

Argument

string
name=value文字列へのポインタ。

Description

putenv関数は、既存の変数を変更するか、新しい変数を作成することによって、環境変数の値を設定します。string引数はname=valueの形の文字列をポイントします。nameは環境変数、valueはその環境変数の新しい値を示します。

stringがポイントする文字列は環境の一部になるので、この文字列を変更すると環境が変更されます。新しい文字列定義名がputenvに渡されると、stringが使用していたスペースは使用されなくなります。

注意

putenv関数はenviron外部変数がポイントしている環境を操作するもので、getenvと組み合わせて使用することができます。ただし、main関数の第3の引数(環境ポインタ)は変更されません。

putenv関数は、malloc関数を使用して環境を拡大します。

putenvの呼び出しでは、引数として自動変数を使用し、stringがまだ環境の一部である間に呼び出し元関数を終了すると、エラーが発生することがあります。

Return value

0	成功を示します。
-1	エラーを示します。errnoは ENOMEM に設定されます。環境リストを拡大するためのメモリが十分にありません。

Description

putenv関数は 64 ビット・アドレスを取ることができません。『HP C ランタイム・ライブラリ・リファレンス・マニュアル(上巻)』第 1.10 節を参照してください。

puts

標準出力 (stdout) に文字列を書き込み，続けて改行文字を書き込みます。

Format

```
#include <stdio.h>
int puts (const char *str);
```

Argument

str
文字列へのポインタ。

Description

puts関数は，末尾の null 文字を出力ストリームにコピーしません。

Return value

負でない値	成功を示します。
EOF	出力エラーを示します。

putw

指定されたファイルに文字を書き込みます。

Format

```
#include <stdio.h>
int putw (int integer, FILE *file_ptr);
```

Argument

integer
intまたはlong型のオブジェクト。

file_ptr
ファイル・ポインタ。

Description

putw関数は、出力ファイルに4つの文字をintとして書き込みます。変換は行われません。

Return value

integer	成功を示します。
EOF	出力エラーを示します。

putwc

ワイド文字を対応するマルチバイト値に変換し、結果を指定されたファイルに書き込みます。

Format

```
#include <wchar.h>

wint_t putwc (wint_t wc, FILE *file_ptr);
```

Argument

wc
wint_t型のオブジェクト。

file_ptr
ファイル・ポインタ。

Description

putwcはマクロとして実装されることがあるため、副作用を持つファイル・ポインタ引数 (例: putwc (wc, *f++)) は間違っって評価される可能性があります。このような場合には、代わりにfputwc関数を使用してください。

fputwcも参照してください。

Return value

x	ファイルに書き込まれる文字。成功を示します。
WEOF	出力エラーを示します。関数はerrnoを設定します。この関数が設定するerrno値のリストについては、fputwcを参照してください。

putwchar

ワイド文字を標準出力 (stdout) に書き込み, その文字を返します。

Format

```
#include <wchar.h>
wint_t putwchar (wint_t wc);
```

Argument

wc
wint_t型のオブジェクト。

Description

putwchar関数はfputwc(wc, stdout) と同じです。

Return value

x	ファイルに書き込まれる文字。成功を示します。
WEOF	出力エラーを示します。関数はerrnoを設定します。この関数が設定するerrno値のリストについては, fputwcを参照してください。

– 1

失敗すると、ファイル・ポインタは変更されず、pwriteは以下のいずれかの値をerrnoに設定します。

- EINVAL – offset 引数が不正です。値が負です。
- ESPIPE – ファイル記述子が、パイプまたは FIFO に対応しています。

qabs, labs (*Alpha, I64*)

整数の絶対値を__int64として返します。llabsはqabsの同義語です。

Format

```
#include <stdlib.h>

__int64 qabs  (__int64 j);
__int64 labs  (__int64 j);
```

Argument

j
__int64型の値。

qdiv, lldiv (Alpha, I64)

引数の間で除算を行い、商と剰余を返します。lldivはqdivの同義語です。

Format

```
#include <stdlib.h>

qdiv_t qdiv  (__int64 numer, __int64 denom);
lldiv_t lldiv (__int64 numer, __int64 denom);
```

Argument

numer
__int64型の分子。

denom
__int64型の分母。

Description

qdiv_t型とlldiv_t型は、<stdlib.h>ヘッダ・ファイルに次のように定義されています。

```
typedef struct
{
    __int64 quot, rem;
} qdiv_t, lldiv_t;
```

qsort

オブジェクトの配列をその場でソートします。クイック・ソート・アルゴリズムを実装しています。

Format

```
#include <stdlib.h>

void qsort (void *base, size_t nmemb, size_t size, int (*compar) (const void *, const void *));
```

関数バリエーション

qsort関数は、それぞれ 32 ビットと 64 ビットのポインタ・サイズで使用するための `_qsort32` と `_qsort64` という名前のバリエーションを持っています。ポインタ・サイズ固有の関数の使用方法については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

Argument

base
配列の最初のメンバへのポインタ。このポインタは要素へのポインタ型で、文字へのポインタ型にキャストされていなくてはなりません。

nmemb
配列の中のオブジェクトの数。

size
オブジェクトのサイズ (バイト数)。

compar
比較関数へのポインタ。

Description

`compar` がポイントしている比較関数には、2 つの引数が渡されます。この 2 つの引数は、比較されるオブジェクトをポイントしています。第 1 引数が第 2 引数よりも小さい場合、等しい場合、および大きい場合に、比較関数はそれぞれ 0 よりも小さい整数、0、および 0 よりも大きい整数を返します。

比較関数`compar`は、すべてのバイトを比較する必要はないので、オブジェクトには比較する値に加えて任意のデータを格納することができます。

比較の結果、等しいと見なされる 2 つのオブジェクトの出力順序は予測不可能です。

raise

指定されたソフトウェア・シグナルを発生させます。シグナルを生成すると、`signal`、`ssignal`、または`sigvec`関数によって設定されたアクション・ルーチンが呼び出されます。

Format

```
#include <signal.h>

int raise (int sig); (ANSI C)

int raise (int sig[, int sigcode]); (HP C Extension)
```

Argument

`sig`

生成するシグナル。

`sigcode`

オプションのシグナル・コード。strict ANSI C モードでコンパイルしていない場合にのみ使用できます。たとえば、シグナル SIGFPE (算術トラップ・シグナル) は、それぞれ異なるタイプの算術トラップを表す 10 種類のコードを持っています。

シグナル・コードは、ニーモニックまたは番号で表すことができます。算術トラップ・コードは 1 ~ 10 の番号で表現され、SIGILL コードは番号 0 ~ 2 で表現されます。コード値は<signal.h>ヘッダ・ファイルに定義されています。シグナルのニーモニック、コード、および対応する OpenVMS 例外のリストについては、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』表 4-4 および表 4-5 を参照してください。

Description

`raise`関数を呼び出すと、以下のいずれかの結果が生じます。

- `raise`が<signal.h>ヘッダ・ファイルに定義されている範囲外の`sig`引数を指定していた場合、`raise`関数は 0 を返し、`errno`変数は EINVAL に設定される。
- `signal`、`ssignal`、または`sigvec`がシグナルの SIG_DFL(デフォルト・アクション)を設定している場合、関数は返らない。イメージは、シグナルに対応する OpenVMS エラー・コードで終了する。

raise

- `signal`, `ssignal`, または `sigvec` がシグナルのアクションとして `SIG_IGN` (シグナルを無視) を設定している場合, `raise` は引数 `sig` を返す。
- `signal`, `ssignal`, または `sigvec` は, シグナルのアクション関数を設定しなくてはならない。その関数が呼び出され, その戻り値が `raise` から返される。

シグナル処理の詳細については, 『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』 第 4 章を参照してください。

`gsignal`, `signal`, `ssignal`, および `sigvec` も参照してください。

Return value

0	成功。
ゼロ以外	失敗。

rand, rand_r

0 ~ $2^{31} - 1$ の範囲の擬似乱数を返します。

Format

```
#include <stdlib.h>
int rand (void);
int rand_r (unsigned int seed); (Alpha, I64)
```

Argument

seed
初期シードの値。

Description

rand関数は、少なくとも 2^{32} の周期で、0 ~ {RAND_MAX} の範囲の擬似乱数整数のシーケンスを算出します。

rand_r関数は、0 ~ {RAND_MAX} の範囲の擬似乱数整数のシーケンスを算出します。{RAND_MAX}マクロの値は、少なくとも 32767 です。

seedが指すオブジェクトの初期値が同じ状態でrand_rが呼び出され、rand_rから戻って次に呼び出すまでの間にそのオブジェクトを変更していない場合は、同じシーケンスが生成されます。

srandも参照してください。

他の乱数アルゴリズムについては、randomと、すべての*48関数を参照してください。

戻り値

n
擬似乱数です。

random

よりランダムなシーケンスの擬似乱数を生成します。

Format

```
#include <stdlib.h>

long int random (void);
```

Description

random関数は、rand関数と基本的に同じ呼び出しシーケンスと初期化プロパティを持つが、よりランダムなシーケンスを生成する乱数ジェネレータです。randが生成する下位 12 ビットはパターンとして循環します。しかし、randomが生成するビットはすべてが利用可能です。たとえば、random() &1 はランダムな 2 進値を生成します。

random関数は、デフォルトでは整数 31 個のサイズの状態配列を使用する、非線形の、加算的フィードバックの乱数ジェネレータを使用して、 $0 \sim 2^{31} - 1$ の範囲の連続した擬似乱数を返します。この乱数ジェネレータの周期は、約 $16 \cdot (2^{31} - 1)$ です。乱数ジェネレータの周期は、状態配列のサイズによって決定されます。状態配列のサイズを増やすと、周期も増えます。

256 バイトの状態情報では、乱数ジェネレータの周期は 2^{69} よりも大きくなり、ほとんどの用途に十分です。

rand関数と同様に、random関数はデフォルトでは、シードとして 1 の値を使ってsrandom関数を呼び出すことで複製できる数のシーケンスを生成します。srandom関数は、srand関数とは異なり、使用される状態情報の量が 1 ワードよりも多いため、古いシードを返しません。

rand, srand, srandom, setstate, およびinitstateも参照してください。

Return value

n 乱数です。

[no]raw

raw モードは、Curses 入力ルーチンの [w]getch と [w]getstr でしか動作しません。
raw モードは、UNIX I/O、ターミナル I/O、および標準 I/O の HP C RTL エミュレーションではサポートされていません。

Format

```
#include <curses.h>

raw()

noraw()
```

Description

raw モードの読み込みは、次の 2 つの条件のうちのどちらかで満たされます。ターミナルに一定の文字数 (5) が入力された後、またはターミナルから文字を受信してから一定の時間 (10 秒) が経った後です。

Example

```
/* Example of standard and raw input in Curses package. */
#include <curses.h>
main()
{
    WINDOW *win1;
    char vert = '.',
        hor = '.',
        str[80];

    /* Initialize standard screen, turn echo off. */
    initscr();
    noecho();

    /* Define a user window. */
    win1 = newwin(22, 78, 1, 1);
    leaveok(win1, TRUE);
    leaveok(stdscr, TRUE);
    box(stdscr, vert, hor);

    /* Reset the video, refresh(redraw) both windows. */
```

[no]raw

```
mvwaddstr(win1, 2, 2, "Test line terminated input");
wrefresh(win1);

/* Do some input and output it. */
nocrmode();
wgetstr(win1, str);

mvwaddstr(win1, 5, 5, str);
mvwaddstr(win1, 7, 7, "Type something to clear screen");
wrefresh(win1);

/* Get another character then delete the window. */

wgetch(win1);
wclear(win1);

mvwaddstr(win1, 2, 2, "Test raw input");
wrefresh(win1);

/* Do some raw input 5 chars or timeout - and output it. */
raw();
getstr(str);
noraw();
mvwaddstr(win1, 5, 5, str);
mvwaddstr(win1, 7, 7, "Raw input completed");
wrefresh(win1);

endwin();
}
```

read

ファイルからバイトを読み込み、それらをバッファに格納します。

Format

```
#include <unistd.h>

ssize_t read (int file_desc, void *buffer, size_t nbytes); (ISO POSIX-1)

int read (int file_desc, void *buffer, int nbytes); (Compatibility)
```

Argument

file_desc

ファイル記述子。指定されるファイル記述子は、現在読み込みのためにオープンされているファイルを参照していなくてはなりません。

buffer

入力データが格納される連続したストレージのアドレス。

nbytes

読み込み操作に関わるバイト数の上限。

Description

read関数は、読み込んだバイトの数を返します。戻り値は必ずしもnbytesに等しくはなりません。たとえば、入力がターミナルからである場合には、たかだか1行分の文字しか読み込まれません。

注意

read関数はレコード・ファイルの中のレコード境界を越えないため、たかだか1つのレコードしか読み込みません。各レコードに対し、個別に読み込みを行う必要があります。

Return value

n	読み込まれたバイト数。
-1	読み込みエラーを示します。物理的な入力エラー，不正なバッファ・アドレス，保護違反，未定義のファイル記述子などの理由が考えられます。

Example

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>

main()
{
    int fd,
        i;
    char buf[10];
    FILE *fp ;           /* Temporary STDIO file */

    /* Create a dummy data file */
    if ((fp = fopen("test.txt", "w+")) == NULL) {
        perror("open");
        exit(1);
    }
    fputs("XYZ\n",fp) ;
    fclose(fp) ;

    /* And now practice "read" */
    if ((fd = open("test.txt", O_RDWR, 0, "shr=upd")) <= 0) {
        perror("open");
        exit(0);
    }

    /* Read 2 characters into buf. */
    if ((i = read(fd, buf, 2)) < 0) {
        perror("read");
        exit(0);
    }

    /* Print out what was read. */
    if (i > 0)
        printf("buf='%c%c'\n", buf[0], buf[1]);

    close(fd);
}
```

readdir, readdir_r

ディレクトリ内のエントリを探します。

Format

```
#include <dirent.h>

struct dirent *readdir (DIR *dir_pointer);

int readdir_r (DIR *dir_pointer, struct dirent *entry, struct dirent **result);
```

Argument

`dir_pointer`

オープン・ディレクトリの`dir`構造体へのポインタ。

`entry`

指定されたストリームの現在位置にあるディレクトリ・エントリによって初期化される, `dirent`構造体へのポインタ。

`result`

実行に成功したときに, `entry`へのポインタが格納される位置。

Description

`readdir`関数は, `dir_pointer`によって指定されるディレクトリ・ストリーム内の現在位置にあるディレクトリ・エントリを表す構造体へのポインタを返し, ディレクトリ・ストリームの位置を次のエントリに設定します。ディレクトリ・ストリームの終端に達すると, `NULL` ポインタを返します。ディレクトリ・エントリは, `<dirent.h>`ヘッダ・ファイルに定義されている`dirent`構造体によって記述されます。

`<dirent.h>`ヘッダ・ファイルに定義されている`DIR`型は, ディレクトリ・ストリームを表します。ディレクトリ・ストリームとは, 特定のディレクトリ内のすべてのディレクトリ・エントリの順序付きのシーケンスです。ディレクトリ・エントリはファイルを表します。`readdir`関数の操作とは非同期的に, ディレクトリのファイルの削除や追加を行うことができます。

`readdir`関数から返されるポインタは, 同じディレクトリ・ストリームに対する別の`readdir`呼び出しによって上書きすることができるデータをポイントしています。このデータは, 他のディレクトリ・ストリームに対する別の`readdir`呼び出しによって上書きされることはありません。

直前の`opendir`または`rewinddir`関数の呼び出しの後に、ディレクトリのファイルが削除または追加された場合、その後の`readdir`関数の呼び出しでは、そのファイルのエントリが返されない可能性があります。

ディレクトリの終端に達したとき、または無効な`seekdir`操作を検出したとき、`readdir`関数は `null` 値を返します。

無効な位置をシークしようと試みると、`readdir`関数は、次に呼び出されたときに `null` 値を返します。その前の`tellldir`関数呼び出しが、その位置を返します。

`readdir_r`関数は、`readdir`のリエントラントなバージョンです。`dir_pointer`に加えて、指定されたストリームの現在のディレクトリ・エントリが返される`dirent`構造体へのポインタを指定する必要があります。

操作が成功した場合、`readdir_r`は `0` を返し、`result`に次の2つのポインタのうちのどちらかを格納します。

- エントリが発見された場合には、`entry`へのポインタ
- ディレクトリ・ストリームの終端に達した場合には、`NULL` ポインタ

エラーが発生した場合には、エラーの原因を示すエラー値が返されます。

`entry`がポイントするストレージは、少なくとも `NAME_MAX + 1` 個の要素を含んだ`char d_name`メンバの配列を含んでいる`dirent`に十分な大きさでなくてはなりません。

Example

例については、`closedir`の説明を参照してください。

Return value

<code>x</code>	<code>readdir</code> の実行が成功した場合には、 <code>struct dirent</code> 型のオブジェクトへのポインタ。
<code>0</code>	<code>readdir_r</code> の実行が成功しました。
<code>x</code>	エラーの場合には、エラー値 (<code>readdir_r</code> のみ。)
<code>NULL</code>	エラーが発生したか、ディレクトリ・ストリームの終端に達しました (<code>readdir_r</code> のみ)。エラーが発生した場合、 <code>errno</code> はその原因を示す値に設定されます。

readlink *(Alpha, I64)*

指定したシンボリック・リンクの内容を読み取って、ユーザの指定したバッファに格納します。

Format

```
#include <unistd.h>

ssize_t readlink (const char *restrict link_name, char *restrict user_buffer, size_t buffer_size);
```

Argument

link_name
シンボリック・リンク・ファイルの名前を示すテキスト文字列へのポインタ。

user_buffer
ユーザ・バッファへのポインタ。

buffer_size
ユーザ・バッファのサイズ。

Description

readlink関数は、指定したシンボリック・リンク (link_name) の内容を読み取って、ユーザの指定した、サイズがbuffer_sizeのバッファ (user_buffer) に格納します。

symlink, unlink, realpath, lchown, およびlstatも参照してください。

Return value

n	成功したことを示します。n は、user_bufferに格納されたバイト数です。
---	--

– 1

エラーが発生したことを示します。バッファは変更され
ておらず、errnoに、エラーを示す次のいずれかの値が設
定されます。

- EACCES —(1) 読み取り対象のシンボリック・リンク
があるディレクトリに、このユーザの読み取り許可
が設定されていないか、(2) link_nameのパス接頭辞
に、検索許可のないコンポーネントが存在していま
す。
- ENAMETOOLONG —(1) link_name引数の長さが
PATH_MAX を超えているか、(2) パス名に、長さが
NAME_MAX を超えるコンポーネントが存在してい
ます。
- close , open , またはreadから返されたerrnoの値。

readv (Alpha, I64)

ファイルから読み込みを行います。

Format

```
#include <sys/uio.h>

ssize_t readv (int file_desc, const struct iovec *iov, int iovcnt);
ssize_t _readv64 (int file_desc, struct __iovec64 *iov, int iovcnt);
```

関数バリエーション

readv関数は、それぞれ 32 ビットと 64 ビットのポインタ・サイズで使用するのための、_readv32と_readv64という名前のバリエーションを持っています。ポインタ・サイズ固有の関数の使用方法については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

Argument

file_desc

ファイル記述子。ファイル記述子は、現在読み込み用にオープンされているファイルを指していなければなりません。

iov

入力データが置かれる、iovec構造体の配列。

iovcnt

iov配列のメンバで指定するバッファの数。

Description

readv関数は、readと同等ですが、入力データを、iov配列のメンバ (iov[0], iov[1], ..., iov[iovcnt-1]) で指定されるiovcnt個のバッファに置きます。iovcnt引数は、0 より大きく、IOV_MAX 以下であれば有効です。

各iovecエントリは、データを置くメモリ領域のベース・アドレスと長さを指定します。readv関数は、次に進む前に必ず領域を完全に埋めます。

正常に終了すると、readvは、ファイルの st_atime フィールドをアップデートするよう、マークします。

同期入出力オプションがサポートされている場合は、次の処理が行われます。

O_DSYNC ビットと O_RSYNC ビットが設定されている場合、ファイル記述子に対する読み込み入出力操作は、同期 I/O データ一貫性の完了により定義されているとおりに完了します。

O_SYNC ビットと O_RSYNC ビットが設定されている場合、ファイル記述子に対する読み込み I/O 操作は、同期 I/O ファイル一貫性の完了により定義されているとおりに完了します。

共用メモリ・オブジェクト・オプションがサポートされている場合は、次の処理が行われます。

file_desc が共用メモリ・オブジェクトを指している場合、read 関数の結果は未定義です。

通常のファイルの場合、file_desc に対応する、オープン済みのファイル記述子に設定されている最大オフセットを越えるデータ転送は実行されません。

Return value

n	読み込んだバイト数。
---	------------

- 1

読み込みエラーを示します。この関数は、`errno` に以下のいずれかの値を設定します。

- `EAGAIN` – `O_NONBLOCK` フラグがファイル記述子に設定されていて、プロセスの遅延が発生します。
- `EBADF` – `file_desc` 引数が、読み取り用にオープンされている有効なファイル記述子ではありません。
- `EBADMSG` – `control-normal` モードが設定されている `STREAM` ファイルの場合に、読み込み待ちのメッセージに制御部が含まれていました。
- `EINTER` – シグナルを受信したために読み込み操作が終了し、データは転送されませんでした。
- `EINVAL` – `file_desc` で参照されている `STREAM` やマルチプレクサが、マルチプレクサからダウンストリームで直接的または間接的にリンクされていました。
または
`iov` 配列内の `iov_len` 値の合計が、`ssize_t` を超えていました。
- `EIO` – 物理 I/O エラーが発生しました。
または
制御端末から読み込みを行おうとしているバックグラウンド・プロセスのメンバとなっているプロセスで、プロセスが `SIGTTIN` シグナルを無視またはブロックしているか、プロセス・グループがオーファンドの状態でした。
- `EISDIR` – `file_desc` 引数がディレクトリを参照していますが、実装では、`read`、`pread`、または `readv` によるディレクトリの読み込みを許していません。代わりに、`readdir` 関数を使用してください。
- `EOVERFLOW` – ファイルが通常のファイルで、`nbyte` が 0 より大きい場合に、開始位置が、ファイルの終端よりも前で、`file_desc` に対応するオープン済みのファイル記述子で指定されている最大オフセット以上でした。

`readv` 関数は、次の場合に失敗することがあります。

- `EINVAL` – `iovcnt` 引数が 0 以下、または `IOV_MAX` より大きい値でした。

realloc

第 1 引数がポイントしている領域のサイズを、第 2 引数で指定されたバイト数に変更します。これらの関数は AST リエントラントです。

Format

```
#include <stdlib.h>

void *realloc (void *ptr, size_t size);
```

関数バリエーション

realloc関数は、それぞれ 32 ビットと 64 ビットのポインタ・サイズで使用するための _realloc32と _realloc64という名前のバリエーションを持っています。ポインタ・サイズ固有の関数の使用方法については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

Argument

ptr
割り当て済み領域をポイントするポインタか、あるいは NULL。

size
割り当て済み領域の新しいサイズ。

Description

ptrが NULL ポインタである場合、realloc関数の動作はmalloc関数と同じです。

領域の内容は、古いサイズと新しいサイズのうちの小さい方までは変更されません。ANSI C 標準は次のように述べています。「新しいサイズが古いサイズよりも大きい場合、新しく割り当てられるメモリ部分の値は不定である」。古い実装との互換性のために、HP Cは新しく割り当てられたメモリを 0 に初期化します。

効率を高めるために、以前の実際の割り当ては、要求されたサイズよりも大きいサイズで行われた可能性があります。割り当てがmallocで行われた場合、前に要求された割り当てと実際の割り当ての間のメモリ部分の値は不定です。割り当てがcallocで行われた場合、そのメモリは 0 に初期化されています。アプリケーション

が、reallocがメモリを0に初期化することに依存している場合には、最初の割り当てにmallocではなくcallocを使用するようにしてください。

free, cfree, calloc, およびmallocも参照してください。

Return value

x	領域のアドレス。ワード (<i>Alpha only</i>) またはオクタワード (<i>I64 only</i>) にアラインされています。アドレスが返されるのは、十分なスペースを再割り当てするために、エリアを新しいアドレスに移動しなくてはならないことがあるためです。エリアが移動される場合、以前に占有されていたスペースは解放されます。
NULL	スペースの再割り当てを行えないことを示します (十分なスペースがなかった場合など)。

realpath

POSIX ルートからの絶対パス名を返します。

Format

```
#include <stdlib.h>

char realpath (const char *restrict file_name, char *restrict resolved_name);
```

Argument

file_name

絶対パスを必要とするファイルについて、その名前を示すテキスト文字列へのポインタ。

resolved_name

生成した絶対パスへのポインタ (生成した絶対パスは、null で終了する文字列として格納)。

Description

realpath関数は、POSIX ルートからの絶対パス名を返します。生成したパス名はnull で終了する文字列として、resolved_nameが指すバッファ内に格納されます。格納されるパス名の最大長は、PATH_MAX バイトです。

realpath関数は、POSIX 準拠モードでだけ、つまり、DECC\$POSIX_COMPLIANT_PATHNAMES の値が、許されている値のいずれかになっている場合にだけ、サポートされます。

symlink, unlink, readlink, lchown, およびlstatも参照してください。

Return value

x	成功したことを示します。x は、resolved_nameへのポインタです。
---	--

NULL

エラーが発生したことを示します。ヌル・ポインタが返され、resolved_nameが指しているバッファの内容は意味がありません。errnoに、エラーを示す次のいずれかの値が設定されます。

- ENAMETOOLONG —file_name引数の長さが PATH_MAX を超えたか、パス名に長さが NAME_MAX を超えるコンポーネントがあります。
- ENOENT —file_nameに既存のファイルを指定していないコンポーネントがあるか、file_nameが空の文字列を指しています。
- chdirまたはstatから返された errnoの値。

[w]refresh

ターミナル・スクリーンに指定されたウィンドウを再ペイントします。refresh関数はstdscrウィンドウに作用します。

Format

```
#include <curses.h>

int refresh();

int wrefresh (WINDOW *win);
```

Argument

win
ウィンドウへのポインタ。

Description

このプロセスの結果として、ウィンドウの中の、サブウィンドウや他のウィンドウに占有されていない部分が、ターミナル・スクリーンに表示されます。占有されているウィンドウ全体をターミナル・スクリーンに表示するには、refreshまたはwrefresh関数の代わりにtouchwin関数を呼び出します。

touchwinも参照してください。

Return value

OK	成功を示します。
ERR	エラーを示します。

remainder *(Alpha, I64)*

浮動小数点の剰余 $r = x - n*y$ (ただし, y はゼロでない数) を返します。

Format

```
#include <math.h>

double remainder (double x, double y);
float remainderf (float x, float y);
long double remainderl (long double x, long double y);
```

引数

x
実数値。

y
実数値。

Description

これらの関数は, 浮動小数点の剰余 $r = x - n*y$ (ただし, y はゼロでない数) を返します。値 n は, 値 x/y に最も近い整数, つまり, $n = \text{rint}(x/y)$ です。

$|n - x/y| = 1/2$ の場合は, n の値として偶数が選択されます。

remainder関数の動作は, 丸めモードに依存していません。

remainder関数の機能は, remquo関数の機能と同じです。

Return value

r	成功したことを示します。 r は, 浮動小数点の剰余 $r = x - ny$ (y はゼロでない数) です。
NaN	x または y が NaN です。

remquo (Alpha, I64)

浮動小数点の剰余 $r = x - n*y$ (ただし, y はゼロでない数) を返します。

Format

```
#include <math.h>

double remquo (double x, double y, int * quo);
float remquof (float x, float y, int * quo);
long double remquol (long double x, long double y, int * quo);
```

引数

x
実数値。

y
実数値。

quo
商の格納先となる int 型変数へのポインタ。

Description

remquo(), remquof(), および remquol() 関数から得られる剰余は, それぞれが, remainder(), remainderf(), および remainderl() 関数から得られる剰余と同じです。また, quo が指す変数には, その符号が x/y と同じで, しかも, その絶対値が, $2n$ を法としたときに x/y の整数商 (の絶対値) と合同になるような値を格納します (ただし, n は実装で定義される, 3 以上の整数)。

remquo 関数の機能は, remainder 関数の機能と同じです。

Return value

r	成功したことを示します。n は, 浮動小数点の剰余 $r = x - ny$ (y はゼロでない数) です。
NaN	x または y が NaN です。

remove

ファイルを削除します。

Format

```
#include <stdio.h>

int remove (const char *file_spec);
```

Argument

file_spec

OpenVMS または UNIX スタイルのファイル指定である文字列へのポインタ。ファイル指定は、そのバージョン番号にワイルドカードを含むことができます。したがって、たとえばfilename.txt:*という形式のファイルを削除することができます。

Description

ファイル名内でディレクトリを指定し、それがエラーを含んでいる検索リストだった場合、HP C for OpenVMSシステムはこれをファイル・エラーとして解釈します。

注意

DECC\$ALLOW_REMOVE_OPEN_FILES 機能論理名は、オープン済みのファイルに対するremove関数の動作を制御します。通常は、この操作は失敗します。ただし、POSIXの準拠条件によれば、この操作は成功することになっています。

DECC\$ALLOW_REMOVE_OPEN_FILES が有効になっている場合、このPOSIX 準拠の動作が行われます。

シンボリック・リンクの削除にremoveを使用した場合は、リンク自体が削除されるだけで、リンクが参照しているファイルは削除されません。

remove関数とdelete関数は、HP C RTL では機能的に等価です。

deleteも参照してください。

remove

Return value

0	成功を示します。
ゼロ以外の値	失敗を示します。

rename

既存のファイルに新しい名前を与えます。

Format

```
#include <stdio.h>

int rename (const char *old_file_spec, const char *new_file_spec);
```

Argument

`old_file_spec`
名前を変更するファイルの既存の名前である文字列へのポインタ。

`new_file_spec`
ファイルの新しい名前となる文字列へのポインタ。

Description

現在オープンされているファイルの名前を変更しようと試みた場合の動作は未定義です。1つの物理デバイスから別の物理デバイスへとファイルの名前を変更することはできません。古いファイル指定と新しいファイル指定は、同じデバイス上に存在している必要があります。

`new_file_spec`がファイル・タイプを含んでいない場合には、`old_file_spec`のファイル・タイプが使用されます。名前を変更して、ファイル・タイプをなくすには、`new_file_spec`にピリオド(.)が含まれていなくてはなりません。たとえば、次のようにすると、`SYSS$DISK:[]FILE.DAT`が`SYSS$DISK:[]FILE1.DAT`に変更されます。

```
rename("file.dat", "file1");
```

ただし、次の呼び出しは、`SYSS$DISK:[]FILE.DAT`を`SYSS$DISK:[]FILE1`に変更します。

```
rename("file.dat", "file1.");
```

注意

rename関数はファイル・タイプの特珠な処理を行うので、ファイル名引数を受け付けるCランタイム・ライブラリ関数の呼び出しで、名前を変更したファイルの名前を指定するときには、呼び出し元は慎重になる必要があります。たとえば、次のrename関数の呼び出しの後、新しいファイルはfopen("bar.dat",...)としてオープンする必要があります。

```
rename("foo.dat", "bar");
```

rename関数は、DECC\$RENAME_NO_INHERIT 機能論理名と DECC\$RENAME_ALLOW_DIR 機能論理名の設定に影響されます。

- DECC\$RENAME_NO_INHERIT は、ファイルの新しい名前が古い名前から何か（たとえば、ファイル・タイプ）を継承するか、または完全に指定しなければならないかに影響します。
- DECC\$RENAME_ALLOW_DIR を使用すると、あいまいなファイル指定が2番目の引数の論理名として渡された場合にディレクトリ指定への変換を許すという以前の OpenVMS 動作にするか、ファイルからディレクトリへの名前変更を許さない UNIX 準拠の動作にするかを選択できます。

詳細は、『HP C ランタイム・ライブラリ・リファレンス・マニュアル(上巻)』第 1.6 節の DECC\$RENAME_NO_INHERIT と DECC\$RENAME_ALLOW_DIR の説明を参照してください。

Return value

0	成功を示します。
ゼロ以外の値	失敗を示します。

rewind

ファイルの位置をその先頭に設定します。

Format

```
#include <stdio.h>

void rewind (FILE *file_ptr); (ISO POSIX-1)

int rewind (FILE *file_ptr); (HP C Extension)
```

Argument

file_ptr
ファイル・ポインタ。

Description

rewind関数は、fseek (file_ptr, 0, SEEK_SET) と等価です。rewind関数は、レコード・ファイルとストリーム・ファイルのどちらでも使用できます。

rewindの呼び出しに成功すると、ファイルのエラー・インディケータはクリアされます。

ANSI C 標準は、rewindを値を返さない関数として定義しています。このため、rewindの関数プロトタイプは、voidの戻り型で宣言されています。しかし、rewindは実行に失敗する可能性があり、以前のバージョンのHP C RTLはrewindをintを返す関数として宣言していたため、rewindのコードは成功時には0を、失敗時には-1を返します。

fseekも参照してください。

rewinddir

指定されたディレクトリ・ストリームの位置を、ディレクトリの先頭に再設定します。

Format

```
#include <dirent.h>

void rewinddir (DIR *dir_pointer);
```

引数

`dir_pointer`
オープン・ディレクトリの`dir`構造体へのポインタ。

Description

`rewinddir`関数は、指定されたディレクトリ・ストリームの位置を、ディレクトリの先頭に再設定します。また、`opendir`関数を使用した場合と同じように、ディレクトリ・ストリームが対応するディレクトリの現在の状態を参照するようにします。`dir_pointer`引数がディレクトリ・ストリームを参照していない場合の効果は未定義です。

`<dirent.h>`ヘッダ・ファイルに定義されている`DIR`型は、ディレクトリ・ストリームを表します。ディレクトリ・ストリームとは、特定のディレクトリ内のすべてのディレクトリ・エントリの順序付きのシーケンスです。ディレクトリ・エントリはファイルを表します。

`opendir`も参照してください。

rindex

文字列の中の文字を検索します。

Format

```
#include <strings.h>
char *rindex (const char *s, int c);
```

関数バリエント

rindex関数は、それぞれ 32 ビットと 64 ビットのポインタ・サイズで使用するための `_rindex32` と `_rindex64` という名前のバリエントを持っています。ポインタ・サイズ固有の関数の使用方法については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

Argument

s
検索対象の文字列。

c
検索しようとしている文字。

Description

rindex関数はstrchr関数と同じものであり、一部の UNIX との互換性のために用意されています。

rint (Alpha, I64)

引数を, ユーザが指定した現在の IEEE 丸め向きに従って, 整数値に丸めます。

Format

```
#include <math.h>
double rint (double x);
float rintf (float x);
long double rintl (long double x);
```

Argument

x
実数

Description

rint関数は, /ROUNDING_MODE コマンド・ライン修飾子で指定された現在の IEEE 丸めモードの向きの, xに最も近い整数値を返します。

現在の丸めモードが負の無限大向きである場合, rintはfloorと同じです。現在の丸めモードが正の無限大向きである場合, rintはceilと同じです。

|x| =無限大の場合, rintはxを返します。

Return value

n	現在の IEEE 丸めモードの向きの, xに最も近い整数値。
NaN	xは NaN です。errnoは EDOM に設定されます。

rmdir

ディレクトリ・ファイルを削除します。

Format

```
#include <unistd.h>
int rmdir (const char *path);
```

引数

path
ディレクトリ・パス名。

Description

rmdir関数は、path引数で名前が指定されたディレクトリ・ファイルを削除します。
ディレクトリが削除されるのは、それが空である場合のみです。

指定したpathがシンボリック・リンクであると、rmdirが失敗して、errnoに
ENOTDIR が設定されます。

Description

OpenVMS 形式の名前を使用する場合、path引数はdirectory.dirの形式でなくてはなりません。

Return value

0	成功を示します。
-1	エラーが発生しました。errnoはエラーを示す値に設定されます。

sbrk

プログラムで使用されていない、値の最も小さい仮想アドレスを決定します。

Format

```
#include <unistd.h>

void *sbrk (long int incr);
```

Argument

incr
現在のブレーク・アドレスに加えるバイト数。

Description

sbrk関数は、引数によって指定されたバイト数を現在のブレーク・アドレスに加え、以前のブレーク・アドレスを返します。

プログラムが実行されると、ブレーク・アドレスはプログラムおよびデータ・ストレージ領域によって定義される最も高い位置に設定されます。したがって、sbrkは、データ領域が増大するプログラムでしか必要とされません。

sbrk(0)は現在のブレーク・アドレスを返します。

Return value

x	前のブレーク・アドレス。
(void *)(-1)	プログラムが要求しているメモリが多すぎることを示します。

制限事項

他の C ライブラリの実装とは異なり、HP C RTL メモリ割り当て関数 (malloc など) は、プログラム・ヒープ・スペースの管理に brk や sbrk を使用しません。このため、OpenVMS システムでは、brk または sbrk を呼び出すとメモリ割り当てルーチンと干渉することがあります。brk および sbrk 関数は互換性のためにのみ用意されています。

scalb (Alpha, I64)

浮動小数点数の指数を返します。

Format

```
#include <math.h>

double scalb (double x, double n);
float scalbf (float x, float n);
long double scalbl (long double x, long double n);
```

Argument

x
ゼロでない浮動小数点数。

n
整数。

Description

scalb関数は、整数nに対して $x \cdot (2^{**}n)$ を返します。

Return value

x	実行に成功すると、 $x \cdot (2^{**}n)$ が返されます。
$\pm\text{HUGE_VAL}$	オーバーフローでは、scalbは $\pm\text{HUGE_VAL}$ (xのサインにより) を返し、errnoは ERANGE に認定されます。
0	アンダフローが発生しました。errnoは ERANGE に設定されます。
x	xは $\pm\text{Infinity}$ です。
NaN	xまたはnはNaNです。errnoは EDOM に設定されます。

scanf

標準入力 (stdin) からの書式付き入力を実行し、書式指定に従って解釈します。書式指定子については『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 2 章を参照してください。

Format

```
#include <stdio.h>

int scanf (const char *format_spec, ...);
```

Argument

`format_spec`

書式指定を含んでいる文字列へのポインタ。書式指定は、入力からそのまま取得され、指定された入力ソースに変換されてメモリに格納される文字から構成されています。変換文字のリストについては、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 2 章を参照してください。

...

結果として得られる型が、書式指定で与えられた変換指定に対応するオブジェクトへのポインタであるオプションの式。

変換指定が与えられなかった場合、これらの入力ポインタは省略することができます。そうでない場合は、関数呼び出しは変換指定の数以上の入力ポインタを持っていてはならず、変換指定は入力ポインタの型と一致していなくてはなりません。

変換指定は、左から右の順序で入力ソースと照合されます。余分な入力ポインタがある場合には、無視されます。

Return value

`x`

照合に成功し、代入が行われた入力項目の数。

`EOF`

変換に成功する前に読み込みエラーが発生したことを示します。関数は `errno` を設定します。この関数が設定する `errno` 値のリストについては、`fscanf` を参照してください。

[w]scanw

ウィンドウ上でscanfを実行します。scanw関数はstdscrウィンドウに作用します。

Format

```
#include <curses.h>

int scanw (char *format_spec, ... );
int wscanw (WINDOW *win, char *format_spec, ... );
```

Argument

win

ウィンドウへのポインタ。

format_spec

書式指定文字列へのポインタ。

...

結果として得られる型が、書式指定で与えられた変換指定に対応するオブジェクトへのポインタであるオプションの式。変換指定が与えられなかった場合、これらの入力ポインタは省略することができます。

そうでない場合は、関数呼び出しは変換指定の数以上の入力ポインタを持っていないのではなく、変換指定は入力ポインタの型と一致していなくてはなりません。

変換指定は、左から右の順序で入力ソースと照合されます。余分な入力ポインタがある場合には、無視されます。

Description

書式指定 (format_spec) とその他の引数は、scanf関数で 사용되는ものと同じです。

scanwおよびwscanw関数は、ターミナル・スクリーンからテキスト行を受け付け、書式を設定し、返します。詳細については、scrolllokおよびscanf関数を参照してください。

Return value

OK	成功を示します。
ERR	関数がスクリーンに不正なスクロールを引き起こすか、 スキャンが失敗したことを示します。

scroll

ウィンドウ上のすべての行を 1 行上に移動します。一番上の行はウィンドウからスクロールして消え、一番下の行は空白になります。

Format

```
#include <curses.h>
int scroll (WINDOW *win);
```

Argument

win
ウィンドウへのポインタ。

Return value

OK	成功を示します。
ERR	エラーを示します。

scrolllok

指定されたウィンドウのスクロール・フラグを設定します。

Format

```
#include <curses.h>
scrolllok (WINDOW *win, bool boolf);
```

Argument

win

ウィンドウへのポインタ。

boolf

論理型の TRUE または FALSE 値。boolf が FALSE の場合、スクロールは許可されません。これはデフォルトの設定です。bool型は、<curses.h>ヘッダ・ファイルに次のように定義されています。

```
#define bool int
```

seed48

48 ビットの乱数ジェネレータを初期化します。

Format

```
#include <stdlib.h>

unsigned short *seed48 (unsigned short seed_16v[3]);
```

引数

seed_16v
48 ビットのシード値を構成する，3 つの unsigned short int の配列。

Description

seed48関数は乱数ジェネレータを初期化します。この関数は，プログラム内でdrand48，lrand48，またはmrand48関数を呼び出す前に使用することができます (これは推奨はされませんが，drand48，lrand48，またはmrand48関数が，初期化関数を呼び出すことなく呼び出された場合には，定数のデフォルト・イニシアライザ値が自動的に提供されます)。

seed48関数は，次の線形合同式に従って，48 ビットの整数値 X_i のシーケンスを生成します。

$$X_{n+1} = (aX_n + c) \bmod m \quad n > 0$$

引数 m は 2^{48} に等しいので，48 ビット整数算術演算が実行されます。lcong48関数を呼び出さなかった場合，乗数値 a と加算される値 c は次のようになります。

$$\begin{aligned} a &= 5DEECE66D_{16} = 2736731631558 \\ c &= B_{16} = 138 \end{aligned}$$

初期化関数seed48は，次の処理を行います。

- X_i の値を，seed_16vがポイントする配列で指定された 48 ビット値に設定する。
- seed48のみが使用する， X_i の前の値を含んでいる 48 ビットの内部バッファへのポインタを返す。

返されたポインタを使用して、任意の時点で擬似乱数シーケンスを再開することができます。このポインタを使って、前の Xi 値を一時的配列にコピーしてください。元のシーケンスが中断した場所から再開するには、この配列へのポインタを指定して seed48 を呼び出します。

drand48, lrand48, および mrand48 も参照してください。

Return value

x	48 ビットの内部バッファへのポインタ。
---	----------------------

seekdir

ディレクトリ・ストリームの位置を設定します。

Format

```
#include <dirent.h>

void seekdir (DIR *dir_pointer, long int location);
```

Argument

dir_pointer

オープン・ディレクトリのdir構造体へのポインタ。

location

ディレクトリの先頭を基準としたエントリの番号。

Description

seekdir関数は、dir_pointerによって指定されたディレクトリ・ストリーム上の次のreaddir操作の位置を、locationによって指定された位置に設定します。locationの値は、以前のtelldirの呼び出しから返されたものです。

locationの値が以前のtelldir関数の呼び出しから返されたものでない場合、またはこのディレクトリ・ストリームに対するrewinddir関数の呼び出しが途中で行われていた場合の効果は定められていません。

<dirent.h>ヘッダ・ファイルに定義されているDIR型は、ディレクトリ・ストリームを表します。ディレクトリ・ストリームとは、特定のディレクトリ内のすべてのディレクトリ・エントリの順序付きのシーケンスです。ディレクトリ・エントリはファイルを表します。readdir関数の操作とは非同期的に、ディレクトリのファイルの削除や追加を行うことができます。

readdir, rewinddir, およびtelldirを参照してください。

[w]setattr

ウィンドウ内でビデオ表示属性attrを有効にします。setattr関数はstdscrウィンドウに作用します。

Format

```
#include < curses.h>

int setattr (int attr);

int wsetattr (WINDOW *win, int attr);
```

Argument

win

ウィンドウへのポインタ。

attr

点滅, ボールド, 反転ビデオ, および下線のビデオ表示属性。それぞれ定義済み定数 `_BLINK`, `_BOLD`, `_REVERSE`, および `_UNDERLINE` によって表現されます。次のようにビット論理和演算子 (`|`) で区切ることで、複数の属性を設定することができます。

```
setattr(_BLINK | _UNDERLINE);
```

Description

setattrおよびwsetattr関数はHP C for OpenVMSシステムに固有のもので、移植性はありません。

Return value

OK

成功を示します。

ERR

エラーを示します。

setbuf

入力ファイルまたは出力ファイルに新しいバッファを関連付けます。バッファリングの動作を変更することもあります。

Format

```
#include <stdio.h>

void setbuf (FILE *file_ptr, char *buffer);
```

Argument

file_ptr
ファイル・ポインタ。

buffer
文字配列へのポインタ，または NULL ポインタ。

Description

setbuf関数は，指定されたファイルがオープンされた後，ただし I/O 操作が実行される前に使用することができます。

bufferが NULL ポインタである場合，この呼び出しは，同じfile_ptr，NULL のbufferポインタ，_IONBF のバッファリング・タイプ (バッファリングなし)，およびバッファ・サイズ 0 を指定するsetvbuf呼び出しと等価です。

bufferが NULL ポインタでない場合，この呼び出しは，同じfile_ptr，同じbufferポインタ，_IOFBF のバッファリング・タイプ，および値 BUFSIZ (<stdio.h>に定義) で指定されるバッファ・サイズを指定するsetvbuf呼び出しと等価です。したがって，setbufの呼び出しで使用するbuffer引数を割り当てるときには BUFSIZ を使用するようにします。次に例を示します。


```
#include <stdio.h>
.
.
.
char my_buf[BUFSIZ];
.
.
.
setbuf(stdout, my_buf);
.
.
.
```

ユーザ・プログラムは、ストリームに対して I/O が実行された後には、buffer の内容に依存してはなりません。HP C RTL は、どの I/O 操作についても、buffer を使用する場合と使用しない場合があります。

もともと setbuf 関数は、古いバージョンの UNIX のシステム・デフォルト・バッファの代わりに、プログラマが大きなバッファを割り当てられるようにすることを目的としていました。現在の C の実装では、デフォルト・バッファ・サイズが大きいため、この関数はほとんどのケースでは不要です。setbuf 関数は、ANSI C 標準では、古いプログラムとの互換性のために残されています。新規のプログラムでは、代わりに setvbuf を使用するようしてください。この関数では、プログラマはコンパイル時ではなく実行時にバッファ・サイズをバインドすることができますし、テスト可能な結果値が返されます。

setenv

現在の環境リストのnameで指定された環境変数を挿入または再設定します。

Format

```
#include <stdlib.h>

int setenv (const char *name, const char *value, int overwrite);
```

Argument

name

環境変数リストの中の変数名。

value

環境変数の値。

overwrite

環境変数が存在していた場合、それを再設定するかどうかを示す 0 または 1 の値。

Description

setenv関数は、現在の環境リストの環境変数nameを挿入または再設定します。リストに変数nameが存在しなければ、value引数を使って挿入されます。変数が存在していれば、overwrite引数がテストされます。overwrite引数の値に応じて、次の処理が行われます。

- 0 変数は再設定されない。
- 1 変数はvalueに再設定される。

Return value

0

成功を示します。

-1

エラーを示します。errnoは ENOMEM に設定されます。環境リストを拡張するのに十分なメモリがありません。

seteuid (Alpha, I64)

プロセスの実効ユーザ ID を設定します。

Format

```
#include <unistd.h>
int seteuid (uid_t euid);
```

引数

euid
実効ユーザ ID として設定する値。

Description

プロセスに IMPERSONATE 特権がある場合、seteuid関数は、プロセスの実効ユーザ ID を設定します。

特権のないプロセスは、euid引数がプロセスの実ユーザ ID、実効ユーザ ID、保存済みユーザ ID のいずれかと同じ場合にのみ、実効ユーザ ID を設定できます。

Return value

- | | |
|----|--|
| 0 | 成功を示します。 |
| -1 | エラーを示します。この関数は、次のいずれかの値をerrnoに設定します。 <ul style="list-style-type: none">• EINVAL – euid引数の値が不正、またはサポートされていません。• EPERM – プロセスに IMPERSONATE 特権がなく、euidが実ユーザ ID と保存済みセット・ユーザ ID のどちらとも一致しません。 |

setgid

POSIX ID が無効化されている場合には、プログラムの移植性のためにsetgidが実装されており、何の機能も持ちません。成功を示す 0 を返します。

POSIX ID が有効になっている場合には、setgidはグループ ID を設定します。

Format

```
#include <types.h>
#include <unistd.h>
int setgid (_gid_t gid); (_DECC_V4_SOURCE)
int setgid (gid_t gid); (not _DECC_V4_SOURCE)
```

Argument

gid
グループ ID に設定する値。

Description

setgid関数は、POSIX スタイル識別子が有効の場合でも無効の場合でも使用できます。

POSIX 形式の ID は、OpenVMS Version 7.3-2 およびそれ以降でサポートされています。

POSIX ID が無効化されている場合、setgid関数はプログラムの移植性のために実装されており、何の機能も持ちません。成功を示す 0 を返します。

POSIX ID が有効になっている場合には、次の処理が行われます。

- プロセスが IMPERSONATE 特権を持っている場合、setgid関数は実グループ ID、実効グループ ID、および保存済みセット・グループ ID をgidに設定する。
- プロセスが適切な特権を持っていないが、gidが実グループ ID または保存済みセット・グループ ID と等しければ、setgid関数は実効グループ ID をgidに設定する。実グループ ID と保存済みセット・グループ ID は変更されない。
- 呼び出し元プロセスの補助グループ ID はすべて変更されない。

POSIX 形式の ID を有効または無効にする方法については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル(上巻)』第 1.7 節を参照してください。

Return value

0	成功を示します。
-1	エラーを示します。関数は <code>errno</code> を以下のいずれかの値に設定します。 <ul style="list-style-type: none">• <code>EINVAL</code> – <code>gid</code> 引数の値が無効で、実装によってサポートされていない。• <code>EPERM</code> – プロセスは適切な特権を持っておらず、<code>gid</code> は実グループ ID または保存済みセット・グループ ID と一致しない。

setgrent (*Alpha, I64*)

グループ・データベースをリwindします。

Format

```
#include <grp.h>
void setgrent (void);
```

Description

setgrent関数は、グループ・データベースを実質的にリwindし、検索を繰り返し実行できるようにします。

この関数は、必ず成功します。戻り値はありません。また、errnoも設定されません。

setitimer

インターバル・タイマの値を設定します。

Format

```
#include <time.h>

int setitimer (int which, struct itimerval *value, struct itimerval *ovalue);
```

Argument

which

インターバル・タイマのタイプ。HP C RTL は ITIMER_REAL のみをサポートしています。

value

タイマ・インターバルとインターバルの終わりまでの残り時間をメンバとして含んでいるitimerval構造体へのポインタ。

ovalue

現在のタイマ・インターバルとインターバルの終わりまでの残り時間をメンバとして含んでいるitimerval構造体へのポインタ。

Description

setitimer関数は、whichで指定されたタイマをvalueで指定された値に設定し、ovalueがゼロ以外の値であれば、タイマの前の値を返します。

タイマ値はitimerval構造体によって定義されます。

```
struct itimerval {
    struct timeval it_interval;
    struct timeval it_value;
};
```

itimerval構造体のメンバの値は以下のとおりです。

itimerval メンバの値	意味
it_interval = 0	次にタイマが満了したときにタイマを無効化する (it_valueがゼロ以外の値である場合)。
it_interval = ゼロ以外	タイマが満了したときに、it_valueの再ロードに使用される値を指定する。
it_value = 0	タイマを無効化する。
it_value = ゼロ以外	次にタイマが満了するまでの残り時間を示す。

システム・クロックの解像度よりも小さいタイマ値は、この解像度にまで丸められます。

getitimer関数は、<time.h>ヘッダ・ファイルに ITIMER_REAL として定義されている 1 つのインターバル・タイマを提供しています。このタイマはリアル・タイムでデクリメントします。タイマが満了すると、SIGALARM シグナルが送信されます。

注意

setitimerと、alarm、sleep、またはusleepの間の相互作用は定められていません。

Return value

0	成功を示します。
-1	エラーが発生しました。errnoはエラーを示す値に設定されます。

setjmp

ネストした一連の関数呼び出しから，通常の方法を使わずに，つまり一連のreturn文を使用せずに，定義済みのポイントに制御を移すための手段を提供します。setjmp関数は，呼び出し元関数のコンテキストを環境バッファに保存します。

Format

```
#include <setjmp.h>

int setjmp (jmp_buf env);
```

Argument

env

環境バッファ。呼び出し元の関数のレジスタ・コンテキストを保持するのに十分な長さを持つ整数の配列でなくてはなりません。jmp_buf型は<setjmp.h>ヘッダ・ファイルに定義されています。このバッファには，プログラム・カウンタ (PC) を含む汎用レジスタの内容が格納されます。

Description

setjmpは，初めて呼び出されたときには値0を返します。その後，setjmpの呼び出しと同じ環境を指定してlongjmpを呼び出すと，制御は通常どおりに返ったかのように再びsetjmp呼び出しに戻されます。この2回目のリターンにおけるsetjmpの戻り値は，longjmp呼び出しで指定された値です。setjmpの真の値を保存するためには，対応するlongjmpが呼び出されるまで，setjmpを呼び出す関数を再び呼び出さないようにする必要があります。

setjmp関数はハードウェア汎用レジスタを保存し，longjmp関数はそれらを復元します。longjmpの後には，volatileとしてマークされていないローカル自動変数を除くすべての変数が，longjmpの時点の値を持つようになります。volatileとしてマークされていないローカル自動変数の値は不定です。

setjmpおよびlongjmp関数は，OpenVMS 条件処理機能を使用して，シグナル・ハンドラによる非ローカル goto を実現します。longjmp関数は，HP C RTL 指定のシグナルを生成し，OpenVMS 条件処理機能がデスティネーションに戻るようにすることで実装されています。

HP C RTL は、任意のHP C イメージのシグナル処理を制御できなくてはなりません。HP C がシグナル処理を制御できるようにするためには、すべての例外処理をVAXC\$ESTABLISH関数の呼び出しを通して設定しなくてはなりません。詳細については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 4.2.5 項と、VAXC\$ESTABLISH関数を参照してください。

注意

Alpha システムと I64 システムの C RTL には、標準とは異なるdecc\$setjmp関数と decc\$fast_longjmp関数が用意されています。標準関数の代わりにこれらの非標準関数を使用するためには、プログラムを__FAST_SETJMP または __UNIX_SETJMP マクロを定義してコンパイルする必要があります。

標準のlongjmp関数とは異なり、decc\$fast_longjmp関数は第 2 引数を 0 から 1 に変換しません。decc\$fast_longjmpの呼び出しの後、対応するsetjmp関数は、decc\$fast_longjmp呼び出しで指定された第 2 引数とまったく同じ値で返ります。

制限事項

OpenVMS 条件ハンドラからlongjmp関数を呼び出すことはできません。ただし、以下のネスト制約の範囲内で、HP C RTL がサポートしている任意のシグナルに対して確立されたシグナル・ハンドラからlongjmpを呼び出すことができます。

- longjmp関数は、ネストしたシグナル・ハンドラから呼び出された場合には動作しない。他のシグナル・ハンドラ内で生成された例外の結果として実行されたシグナル・ハンドラから呼び出されたlongjmp関数の結果は未定義である。
- 対応するlongjmpを、シグナルの処理が完了する前に発行したい場合を除いて、シグナル・ハンドラからsetjmp関数を呼び出してはならない。
- 終了ハンドラ (atexitまたはSYSSDCLEXH で設定) の中からlongjmp関数を呼び出してはならない。終了ハンドラはイメージのティアダウンの後に呼び出されるので、longjmpのデスティネーション・アドレスは存在しなくなっている。
- シグナル・ハンドラの中から、メインの実行スレッドに戻るためにlongjmpを呼び出すと、プログラムの状態の一貫性が失われることがある。副作用として、I/O が実行できなくなったり、UNIX シグナルを受信できなくなったりする可能性がある。代わりにsiglongjmpを使用すること。

Return value

説明のセクションを参照してください。

setkey

encrypt関数で使用するエンコード・キーを設定します。

Format

```
#include <unistd.h>
#include <stdlib.h>
void setkey (const char *key;)
```

引数

key
0, 1 を含む, 長さ 64 の文字配列。

Description

setkeyの引数には, 数値の 0 と 1 からなる文字 (正確にはバイト・データ) だけを含む, 長さが 64 の文字配列を指定します。マシンにキーとして設定されるのは, この文字列, つまりバイト列を 8 個のグループに分けて, その各グループの最下位ビットを無視した 56 ビットです。

戻り値はありません。

cryptとencryptも参照してください。

setlocale

categoryおよびlocale引数によって指定された、プログラムのロケールの該当部分を選択します。この関数を使用すると、1つのカテゴリ、またはプログラムのカレント・ロケール全体を変更または照会することができます。

Format

```
#include <locale.h>

char *setlocale (int category, const char *locale);
```

Argument

category

カテゴリの名前。ロケール全体を変更または照会するには、LC_ALLを指定します。その他の有効なカテゴリ名は以下のとおりです。

- LC_COLLATE
- LC_CTYPE
- LC_MESSAGES
- LC_MONETARY
- LC_NUMERIC
- LC_TIME

locale

ロケールを指定する文字列へのポインタ。

Description

setlocale関数は、categoryおよびlocale引数によって指定された、プログラムのロケールの該当部分を設定または照会します。category引数としてLC_ALLを指定すると、ロケール全体が指定されます。その他の値を指定すると、プログラムのロケールの一部のみが指定されます。

locale引数は、使用するロケールを識別する文字列をポイントします。この引数は以下のいずれかです。

- パブリック・ロケールの名前

次の形式でパブリック・ロケールを指定します。

```
language_country.codeset[@modifier]
```

関数は、論理名 `SYSS$I18N_LOCALE` によって定義された位置で、パブリック・ロケール・バイナリ・ファイルを探します。ファイル・タイプのデフォルトは、`LOCALE` です。名前の中のピリオド(.)とアットマーク(@)文字は、下線(_)に置き換えられます。

たとえば、指定された名前が `zh_CN.dechanzi@radical` だった場合、関数は `SYSS$I18N_LOCALE:ZH_CN_DECHANZI_RADICAL.LOCALE` バイナリ・ロケール・ファイルを探します。

- ファイル指定

バイナリ・ロケール・ファイルを指定します。任意の有効なファイル指定を使用することができます。デバイスまたはディレクトリが省略された場合、関数は現在の呼び出し元デバイスとディレクトリを、省略されたコンポーネントのデフォルト値として使用します。ファイルが見つからなかった場合、関数は `SYSS$I18N_LOCALE` 論理名によって定義されたデバイスとディレクトリをデフォルト値として適用します。ファイル・タイプのデフォルト値は `LOCALE` です。ワイルドカードは使用できません。バイナリ・ロケール・ファイルはリモート・ノード上には存在できません。

- "C"

C ロケールを指定します。プログラムが `setlocale` を呼び出さない場合には、C ロケールがデフォルトとなります。

- "POSIX"

C ロケールと同じです。

- ""

ロケールが国際環境論理名の設定をもとに初期化されることを指定します。関数は、定義されている論理名を発見するまで、次の順序で論理名をチェックします。

1. `LC_ALL`
2. カテゴリに対応する論理名。たとえば、カテゴリとして `LC_NUMERIC` が指定されている場合、`setlocale` がチェックする最初の論理名は `LC_NUMERIC` となる。
3. `LANG`
4. `SYSS$LC_ALL`
5. `SYSS$LC_*` 論理名によって定義される、カテゴリのシステム・デフォルト。たとえば、`LC_NUMERIC` カテゴリのデフォルトは、`SYSS$LC_NUMERIC` 論理名によって定義される。

6. SYSS\$LANG

どの論理名も定義されていなければ、デフォルトではC ロケールが使用されます。SYSS\$LC_*論理名は、システムのスタートアップ時にセットアップされます。

locale引数と同様に、国際環境論理名の等価名は、パブリック・ロケールの名前か、ファイル指定となります。setlocale関数はこの等価名を、locale引数で指定されたかのように扱います。

- NULL

setlocaleにカレント・ロケールを照会させます。関数は、プログラムのロケールのcategoryに関連付けられた部分を記述する文字列へのポインタを返します。LC_ALL カテゴリを指定すると、ロケール全体を記述する文字列が返されます。ロケールの変更は行われません。

- 前のsetlocaleの呼び出しから返された文字列

関数は、プログラムのロケールのcategoryに関連付けられた部分を復元します。文字列がロケール全体の記述を含んでいる場合には、文字列のcategoryに対応する部分を使用されます。文字列がプログラムのロケールのうちの1つのカテゴリの部分を記述している場合には、そのロケールが使用されます。つまり、たとえばLC_COLLATE カテゴリを指定したsetlocale呼び出しから返された文字列を使って、LC_MESSAGES カテゴリに同じロケールを設定することができます。

指定されたロケールが使用可能である場合、setlocaleは、プログラムのロケールのcategoryに関連付けられた部分を記述する文字列へのポインタを返します。LC_ALL カテゴリでは、返される文字列はプログラムのロケール全体を記述します。エラーが発生した場合には、NULL ポインタが返され、プログラムのロケールは変更されません。

setlocaleへのそれ以降の呼び出しは、返された文字列を上書きします。ロケールのその部分を復元する必要がある場合には、プログラム内で文字列を保存しておくようにしてください。呼び出し元のプログラムは、返される文字列の形式または長さに関する仮定を行うべきではありません。

Return value

x	ロケールを記述する文字列へのポインタ。
NULL	エラーが発生したことを示します。errnoが設定されません。

Example

```
#include <errno.h>
#include <stdio.h>
#include <locale.h>

/* This program calls setlocale() three times. The second call */
/* is for a nonexistent locale. The third call is for an      */
/* existing file that is not a locale file.                    */

main()
{
    char *ret_str;

    errno = 0;
    printf("setlocale (LC_ALL, \"POSIX\")");
    ret_str = (char *) setlocale(LC_ALL, "POSIX");

    if (ret_str == NULL)
        perror("setlocale error");
    else
        printf(" call was successful\n");

    errno = 0;
    printf("\n\nsetlocale (LC_ALL, \"junk.junk_codeset\")");
    ret_str = (char *) setlocale(LC_ALL, "junk.junk_codeset");

    if (ret_str == NULL)
        perror(" returned error");
    else
        printf(" call was successful\n");

    errno = 0;
    printf("\n\nsetlocale (LC_ALL, \"sys$login:login.com\")");
    ret_str = (char *) setlocale(LC_ALL, "sys$login:login.com");

    if (ret_str == NULL)
        perror(" returned error");
    else
        printf(" call was successful\n");
}
```

この例のプログラムを実行すると、次の結果が生成されます。

```
setlocale (LC_ALL, "POSIX") call was successful
setlocale (LC_ALL, "junk.junk_codeset")
returned error: no such file or directory

setlocale (LC_ALL, "sys$login:login.com")
returned error: nontranslatable vms error code: 0x35C07C
%c-f-localebad, not a locale file
```

REF-640

- 1

エラーを示します。この関数は、以下のいずれかの値をerrnoに設定します。

- EACCES – pid引数の値が、呼び出し元プロセスの子プロセスのプロセス ID と一致し、その子プロセスが execファミリ関数のいずれかを正常に実行していました。
- EINVAL – pgid引数の値が 0 より小さいか、実装でサポートされていない値でした。
- EPERM – pid引数で指定されたプロセスは、セッション・リーダーです。pid引数の値が、呼び出し元プロセスの子プロセスのプロセス ID と一致し、その子プロセスが、呼び出し元プロセスと同じセッションに属していません。pgid引数の値は有効ですが、pid引数で指定されたプロセスのプロセス ID と一致していず、呼び出し元プロセスと同じセッション内に、pgid引数の値と一致するプロセス・グループ ID を持つプロセスがありません。
- ESRCH – pid引数の値が、呼び出し元プロセスのプロセス ID、呼び出し元プロセスの子プロセスのプロセス ID のどちらとも一致しません。

setpgrp (*Alpha, I64*)

プロセス・グループ ID を設定します。

Format

```
#include <unistd.h>
pid_t setpgrp (void);
```

Description

呼び出し元プロセスがまだセッション・リーダーでなければ、setpgrpは、呼び出し元プロセスのプロセス・グループ ID として、呼び出し元プロセスのプロセス ID を設定します。setpgrpで新しいセッションが作成される場合、新しいセッションには制御端末はありません。

呼び出し元プロセスがセッション・リーダーの場合、setpgrp関数の効果はありません。

戻り値

x	呼び出し元プロセスのプロセス・グループ ID です。
---	----------------------------

setpwent

ユーザ・データベースをリwindします。

Format

```
#include <pwd.h>
void setpwent (void);
```

Description

setpwent関数は、ユーザ・データベースを実質的にリwindし、検索を繰り返し実行できるようにします。

戻り値はありませんが、I/O エラーが発生した場合は、errnoに EIO が設定されます。

getpwentも参照してください。

setregid (Alpha, I64)

実グループ ID と実効グループ ID を設定します。

Format

```
#include <unistd.h>

int setregid (gid_t rgid, gid_t egid);
```

Argument

rgid
実グループ ID として設定する値。

egid
実効グループ ID として設定する値。

Description

setregid関数は、呼び出し元プロセスの実グループ ID と実効グループ ID を設定するために使用されます。rgidが-1の場合、実グループ ID は変更されません。egidが-1の場合、実効グループ ID は変更されません。実グループ ID と実効グループ ID には、一度の呼び出しで異なる値を設定できます。

IMPERSONATE 特権を持つプロセスだけが、実グループ ID と実効グループ ID に任意の有効な値を設定できます。

特権のないプロセスは、実グループ ID としてexec関数による保存済みセット・グループ ID を設定するか、実効グループ ID として保存済みセット・グループ ID または実グループ ID を設定することができます。

呼び出し元プロセスの補助グループ ID は変更されません。

セット・グループ ID プロセスが、自身の実効グループ ID として自身の実グループ ID を設定した場合、実効グループ ID を保存済みセット・グループ ID に戻すことができます。

Return value

0	成功を示します。
-1	<p>エラーを示します。どのグループ ID も変更されず、errnoに以下のいずれかの値が設定されます。</p> <ul style="list-style-type: none">• EINVAL – rgid引数またはegid引数の値が、不正または範囲外です。• EPERM – プロセスに IMPERSONATE 特権がなく、実グループ ID として保存済みセット・グループ ID を設定したり、実効グループ ID として実グループ ID または保存済みグループ ID を設定する変更以外の変更が要求されました。

Format

Argument

Description

IMPERSONATE 特権のないプロセスが、実ユーザ ID を、プロセスの現在の実ユーザ ID、実効ユーザ ID、または保存済みユーザ ID と一致する値に変更できるかどうかは、規定されていません。

Return value

REF-646

-1

エラーを示します。この関数は、以下のいずれかの値をerrnoに設定します。

- EINVAL – ruid引数またはeuid引数の値が、不正または範囲外です。
- EPERM – 現在のプロセスに IMPERSONATE 特権がなく、実効ユーザ ID を実ユーザ ID や保存済みセット・ユーザ ID 以外の値に変更しようとしたか、実ユーザ ID を実装で許されていない値に変更しようとした。

setuid (Alpha, I64)

セッションを作成し、プロセス・グループ ID を設定します。

Format

```
#include <unistd.h>

pid_t setuid (void);
```

Description

setuid関数は、呼び出し元プロセスがプロセス・グループ・リーダーでなければ、新しいセッションを作成します。戻ったときには、呼び出し元プロセスは、この新しいセッションのセッション・リーダーとなっており、また新しいプロセス・グループのプロセス・グループ・リーダーとなっています。制御端末は持っていません。呼び出し元プロセスのプロセス・グループ ID には、呼び出し元プロセスのプロセス ID と同じ値が設定されます。呼び出し元プロセスは、新しいプロセス・グループ内で唯一のプロセスであり、また新しいセッション内で唯一のプロセスです。

setuid関数のプロトタイプが見えるようにするには、`__USE_LONG_GID_T` 機能マクロを有効にする必要があります。

Return value

x	呼び出し元プロセスのプロセス・グループ ID です。
(pid_t)-1	エラーを示します。この関数は、次の値をerrnoに設定します。
	<ul style="list-style-type: none">• EPERM – 呼び出し元プロセスがすでにプロセス・グループ・リーダーであるか、呼び出し元プロセス以外のプロセスのプロセス・グループ ID が、呼び出し元プロセスのプロセス ID と一致しています。

setstate

乱数ジェネレータの再開と変更を行います。

Format

```
char *setstate (char *state;)
```

Argument

state
状態情報の配列をポイントします。

Description

setstate関数は、乱数ジェネレータの再開と変更を行います。

状態の初期化を行った後、setstate関数を使用すると、状態配列の間で素早く切り替えを行うことができます。stateによって定義された状態は、initstate関数が呼び出されるか、setstate関数が再び呼び出されるまで、それ以降の乱数生成に使用されます。setstate関数は、以前の状態配列へのポインタを返します。

初期化の後には、次のように異なる2つの方法で、状態配列を再開することができます。

- 希望のseed、状態配列、および配列のサイズを指定して、initstate関数を使用する。
- 希望の状態を指定してsetstate関数を呼び出した後に、希望のseedを指定してsrandom関数を使用する。両方の関数を使用することの利点は、いったん初期化した状態配列のサイズを保存しておかなくてもすむ点にある。

initstate, srandom, およびrandomも参照してください。

Return value

x	以前の状態配列情報へのポインタ。
0	エラーを示します。状態情報は破壊されています。また、errnoに次の値が設定されます。 <ul style="list-style-type: none">EINVAL—state引数が無効。

setuid

POSIX ID が無効化されている場合には、プログラムの移植性のために実装されており、何の機能も持ちません。成功を示す 0 を返します。

POSIX ID が有効になっている場合には、ユーザ ID を設定します。

Format

```
#include <types.h>
#include <unistd.h>
int setuid (_uid_t uid); (_DECC_V4_SOURCE)
uid_t setuid (uid_t uid); (not _DECC_V4_SOURCE)
```

Argument

uid
ユーザ ID に設定する値。

Description

setuid関数は、POSIX スタイル識別子が有効の場合でも無効の場合でも使用できます。

POSIX 形式の ID は、OpenVMS Version 7.3-2 およびそれ以降でサポートされています。

POSIX ID が無効化されている場合 (デフォルト)、setuid関数はプログラムの移植性のために実装されており、何の機能も持ちません。成功を示す 0 を返します。

POSIX ID が有効になっている場合には、次の処理が行われます。

- プロセスが IMPERSONATE 特権を持っている場合、setuid関数は実ユーザ ID、実効ユーザ ID、および保存済みセット・ユーザ ID をuidに設定する。
- プロセスが適切な特権を持っていないが、uidが実ユーザ ID または保存済みセット・ユーザ ID と等しければ、setuid関数は実効ユーザ ID をuidに設定する。実ユーザ ID と保存済みセット・ユーザ ID は変更されない。

POSIX 形式の ID を有効または無効にする方法については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.7 節を参照してください。

Return value

0	成功を示します。
-1	<p>エラーを示します。関数はerrnoを以下のいずれかの値に設定します。</p> <ul style="list-style-type: none">• EINVAL—uid引数の値が無効で、実装によってサポートされていない。• EPERM—プロセスは適切な特権を持っておらず、uidは実ユーザ ID または保存済みセット・ユーザ ID と一致しない。

setvbuf

入力ファイルまたは出力ファイルに新しいバッファを関連付けます。バッファリングの動作を変更することもあります。

Format

```
#include <stdio.h>

int setvbuf (FILE *file_ptr, char *buffer, int type, size_t size);
```

Argument

file_ptr

ファイルへのポインタ。

buffer

文字配列へのポインタ，または NULL ポインタ。

type

バッファリング・タイプ。<stdio.h>に定義されている，_IOFBF または _IOLBF のいずれかの値を使用します。

size

HP C RTL がこのファイルのバッファリングに使用するbufferによって使用されるバイト数。バッファ・サイズは，8192 バイト以上，32767 バイト以下でなくてはなりません。

Description

setvbuf関数は，指定されたファイルがオープンされた後，ただし I/O 操作が実行される前に使用することができます。

C RTL では，ANSI に準拠したファイル・バッファリングとして，以下のタイプが用意されています。

行バッファード I/O では，文字はメモリ領域にバッファリングされ，改行文字が現れた時点で，適切な RMS ルーチンが呼び出されてバッファ全体が送信されます。行バッファリングはシステム・オーバーヘッドを軽減するのでアンバッファード I/O よりも効率的ですが，出力データをユーザまたはディスクが利用できるタイミングが遅れます。

フル・バッファード I/O では、文字はブレイク文字の有無にかかわらず、バッファがいっぱいになるまでメモリ領域内にバッファリングされます。フル・バッファリングは行バッファリングやアンバッファード I/O よりも効率的ですが、出力データが利用できるようになるタイミングは行バッファリングよりもさらに遅れます。

行バッファード I/O とフル・バッファード I/O を指定するための type 引数としては、`<stdio.h>` に定義されている値 `_IOLBF` と `_IOFBF` をそれぞれ使用します。

HP C RTL では、`file_ptr` がターミナル・デバイスを指定している場合にだけ行バッファード I/O を使用し、それ以外の場合はフル・バッファード I/O を使用します。

このマニュアルの以前の版では `_IONBF` もサポートの対象になっていましたが、この版ではサポートの対象から外されているので、注意してください。

HP C RTL は、個々の I/O ストリームに使用されるバッファを自動的に割り当てます。したがって、バッファ割り当てには以下に示す可能性があることになります。

- `buffer` が NULL ポインタでなく、`size` が自動的に割り当てられるバッファよりも小さい場合、`setvbuf` はファイル・バッファとして `buffer` を使用する。
- `buffer` が NULL ポインタであるか、`size` が自動的に割り当てられるバッファよりも小さい場合には、自動的に割り当てられるバッファがバッファ領域として使用される。
- `buffer` が NULL ポインタで、`size` が自動的に割り当てられるバッファよりも大きい場合、`setvbuf` は指定されたサイズの新しいバッファを割り当て、これをファイル・バッファとして使用する。

ユーザ・プログラムは、ストリームに対して I/O が実行された後には、`buffer` の内容に依存してはなりません。HP C RTL は、どの I/O 操作についても、`buffer` を使用する場合と使用しない場合があります。

一般に、`setvbuf` または `setbuf` を使って、HP C RTL が使用するバッファ・サイズを制御する必要はありません。自動的に割り当てられるバッファ・サイズは、実行される I/O 操作の種類とデバイス特性（ターミナル、ディスク、ソケットなど）に基づいて、効率性を念頭に置いて選択されます。

`setvbuf` および `setbuf` 関数は、バッファを導入して、`stdout` ストリームに大量のテキストを書き込むときの性能を改善したい場合に有用です。このストリームは、ターミナル・デバイスにバインドされているとき（通常のケース）はバッファリングされておらず、`setvbuf` または `setbuf` の呼び出しによって HP C RTL バッファリングが導入されない限り、多数の OpenVMS バッファード I/O 操作が発生します。

`setvbuf` 関数は、HP C RTL が使用するバッファリングを制御するためのみに使用され、下位の RMS I/O 操作が使用するバッファリングは制御しません。RMS のデフォルトのバッファリング動作を変更するには、ファイルを `creat`、`freopen`、また

はopen関数でオープンするときに、ctx、fop、rat、gbc、mbc、mbf、rfm、およびropのRMSキーワードに対して各種の値を指定します。

Return value

0	成功を示します。
ゼロ以外の値	typeまたはfile_ptrに無効な入力値が指定されたか、file_ptrが別のスレッドによって使用されていることを示します(『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.9.1 項を参照)。

shm_open (Alpha, I64)

共用メモリ・オブジェクトをオープンします。

Format

```
#include <sys/mman.h>

int shm_open (const char *name, int oflag, mode_t mode);
```

引数

name

共用メモリ・オブジェクトの名前 (文字列) を指すポインタ。

oflag

ファイルの状態とアクセス許可を定義するオプション。この引数には、<fcntl.h>ヘッダ・ファイルに定義されているオプション (0 個以上) を、ビット単位の論理和にまとめて指定します。

mode

共用メモリ・オブジェクトの許可ビット。この引数は、共用メモリ・オブジェクトを作成するときにだけ使用します。

Description

shm_open関数は、共用メモリ・オブジェクトにファイル記述子を結びつけます。この関数を呼び出すと、共用メモリ・オブジェクトを参照するオープン・ファイル記述と、そのオープン・ファイル記述を参照するオープン・ファイル記述子が作成されます。このファイル記述子は、他の関数がその共用メモリ・オブジェクトを参照するために使用します。name引数には、共用メモリ・オブジェクトの名前 (文字列) を指すポインタを指定します。名前は、パス名であってもかまいません。ただしその場合は、そのパス名を参照する他のプロセスが、同じ共用メモリ・オブジェクトを参照することになります。

作成された共用メモリ・オブジェクトの状態と関連データはすべて、その共用メモリがアンリンクされるまで保持されます。

shm_open関数からは共用メモリ・オブジェクトのファイル記述子が返されますが、そのファイル記述子には、そのプロセス用に現在オープンされていないファイル記述子から番号の最も小さいものが選択されて、割り当てられます。

オープン・ファイル記述にあるファイル状態フラグとファイル・アクセス・モードには、`oflag`で指定した次の値 (0 個以上指定可) が設定されます。

`O_RDONLY` — 読み取りアクセス専用でオープンします。

`O_RDWR` — 読み取りおよび書き込み用にオープンします。

`O_CREAT` — 指定した共用メモリ・オブジェクトが存在していなければ、そのメモリ・オブジェクトを作成します。共用メモリ・オブジェクトのユーザ ID とグループ ID は、呼び出しプロセスと同じものになります。また、共用メモリ・オブジェクトの許可ビットについては、そのプロセスのファイル・モード作成マスクに設定されていないビットだけが、`mode`の値に従って設定されます。

`O_EXCL` — `O_CREAT` を指定したにもかかわらず、その共用メモリ・オブジェクトがすでに存在していれば、その共用メモリ・オブジェクトをオープンしないようにします。このオプションは、必ず `O_CREAT` と組み合わせて使用します。

`O_TRUNC` — 共用メモリ・オブジェクトを読み書き両用 (`O_RDWR`) でオープンできたら、この共用メモリ・オブジェクトを長さ 0 に切り詰めます。

共用メモリ・オブジェクトの内容の初期状態は、バイナリでゼロになっています。

Return value

<code>n</code>	成功したことを示します。 <code>n</code> はファイル記述子を示す 0 または正の整数で、まだ使用されていないものから番号の最も小さいものが選択されて割り当てられます。このファイル記述子は、共用メモリ・オブジェクトを指しています。
----------------	---

エラーが発生したことを示します。errnoに、エラーを示す次のいずれかの値が設定されます。

- EACCES — (1) 共用メモリ・オブジェクトを作成するための許可が認められなかったか、(2) その共用メモリ・オブジェクトがすでに存在していて、oflagで指定した許可が認められなかったか、(3) O_TRUNCが指定されているにもかかわらず、書き込み許可が認められませんでした。
- EEXIST — O_CREAT と O_EXCL の指定がありましたが、指定された共用メモリ・オブジェクトがすでに存在していました。
- EINTR — shm_openの処理がシグナルで中断されました。
- EINVAL — 指定されたnameに対しては、shm_open操作がサポートされていません。
- EMFILE — このプロセスで現在使用中のファイル記述子が多すぎます。
- ENAMETOOLONG — name引数の長さが PATH_MAX を超えているか、パス名に長さが NAME_MAX を超えているコンポーネントが存在します。
- ENFILE — システムで現在オープンされている共用メモリ・オブジェクトの数が多すぎます。
- ENOENT — O_CREAT が設定されていないにもかかわらず、指定された共用メモリ・オブジェクトが存在していません。
- ENOSPC — 共用メモリ・オブジェクトを新しく作成するためのメモリ領域が十分にありません。

shm_unlink *(Alpha, I64)*

共用メモリ・オブジェクトを削除します。

Format

```
#include <sys/mman.h>

int shm_unlink (const char *name);
```

引数

name

削除する共用メモリ・オブジェクトの名前を示す文字列へのポインタ。

Description

shm_unlink関数は、name (の指す文字列) で指定した共用メモリ・オブジェクトについて、その名前を削除します。

ただしshm_unlinkの処理では、共用メモリ・オブジェクトをアンリンクするときにそのオブジェクトの参照元が1個以上存在していると、その名前を削除しただけで、呼び出し元へ戻ります。メモリ・オブジェクトの内容は、その時には削除されません。メモリ・オブジェクトの内容が削除されるのは、その共用メモリ・オブジェクトに対して、オープンされている参照とマッピングされている参照がすべて削除された後です。

共用メモリ・オブジェクトは、最後のshm_unlinkが実行された後も存在し続けることがあるかもしれません。しかし、そのような場合にshm_unlinkでその名前を指定して再使用しようとしても、その名前の共用メモリ・オブジェクトが存在していないかのように処理されるだけです。つまり、shm_openは、O_CREATを設定しないで呼び出すと失敗し、O_CREATを設定して呼び出すと、新しい共用メモリ・オブジェクトが作成されます。

Return value

0 成功したことを示します。

– 1

失敗したことを示します。指定した共用メモリ・オブジェクトは変更されていません。errnoに、エラーを示す次のいずれかの値が設定されます。

- EACCES — 指定した共用メモリ・オブジェクトをアンリンクするための許可が認められませんでした。
- ENAMETOOLONG — name引数の長さが PATH_MAX を超えているか、パス名に長さが NAME_MAX を超えているコンポーネントが存在しています。
- ENOENT — 指定した共用メモリ・オブジェクトが存在していません。

sigaction

シグナルが配信されたときに実行するアクションを指定します。

Format

```
#include <signal.h>

int sigaction (int sig, const struct sigaction *action, struct sigaction *o_action);
```

Argument

sig

アクションに対応するシグナル。

action

sig引数によって指定されたシグナルの受信時に実行されるアクションを記述するsigaction構造体へのポインタ。

o_action

sigaction構造体へのポインタ。sigaction関数が呼び出しから返ると、以前に指定されたシグナルにアタッチされていたアクションは、この構造体に格納されます。

Description

プロセスは、sigaction関数によって、指定されたシグナルが配信されたときに実行されるアクションの確認と指定の両方を行うことができます。引数は、sigaction関数の動作を次のように決定します。

- sig引数の指定は、影響を受けるシグナルを識別する。<signal.h>ヘッダ・ファイルに定義されている、SIGKILL 以外の任意のシグナル値を使用することができます。

sigが SIGCHLD で、SA_NOCLDSTOP フラグがsa_flagsに設定されていない場合には、子プロセスが停止するたびに、呼び出し元プロセスに対して SIGCHLD シグナルが生成される。sigが SIGCHLD で、SA_NOCLDSTOP フラグがsa_flagsに設定されている場合には、SIGCHLD シグナルはこのような形では生成されない。

- action引数の指定は、null でなければ、シグナルの受信時に実行されるアクションを定義するsigaction構造体をポインタする。action引数が null ならば、シグナル処理は変更されないで、この呼び出しを使ってシグナルの現在の処理に関する問い合わせを行うことができる。

- `o_action` 引数の指定は、`null` でなければ、指定されたシグナルに以前にアタッチされていたアクションを含んでいる `sigaction` 構造体をポイントする。

`sigaction` 構造体は以下のメンバから構成されています。

```
void      (*sa_handler)(int);
sigset_t  sa_mask;
int       sa_flags;
```

`sigaction` 構造体のメンバは、以下のように定義されています。

`sa_handler` このメンバは、以下の値を含むことができます。

- `SIG_DFL`— シグナルの配信時に実行されるデフォルト・アクションを指定する。
- `SIG_IGN`— シグナルが受信側プロセスに対して何の効果も持たないことを指定する。
- 関数ポインタ — シグナルをキャッチするよう要求する。シグナルは関数呼び出しを引き起こす。

`sa_mask` このメンバは、`sa_handler` メンバが指定するシグナル・ハンドラ関数の実行中に、プロセス・シグナル・マスクに含まれるシグナルに加えて、個々のシグナルの配信をブロックするように要求することができます。

`sa_flags` このメンバは、シグナルの配信時に実行されるアクションをさらに細かく制御するフラグを設定することができます。

`sigaction` 構造体の `sa_flags` メンバは、以下の値を持ちます。

<code>SA_ONSTACK</code>	このビットを設定すると、システムは、 <code>sigstack</code> 関数によって指定されたシグナル・スタック上でシグナル・キャッチ関数を実行します。このビットが設定されていない場合は、関数はシグナルの配信先のプロセスのスタック上で実行されます。
<code>SA_RESETHAND</code>	このビットを設定すると、シグナルは <code>SIG_DFL</code> に再設定されます。 <code>SIGILL</code> と <code>SIGTRAP</code> は自動的に再設定できないことに注意してください。
<code>SA_NODEFER</code>	このビットを設定すると、キャッチされたシグナルは自動的にブロックされません。
<code>SA_NOCLDSTOP</code>	このビットが設定されており、 <code>sig</code> 引数が <code>SIGCHLD</code> に等しいときに、呼び出し元プロセスの子プロセスが停止した場合、 <code>SIGCHLD</code> シグナルは、 <code>SIGCHLD</code> で <code>SA_NOCLDSTOP</code> が設定されていない場合にのみ呼び出し元プロセスに送信されます。

`sigaction` によってインストールされたシグナル・キャッチ関数によってシグナルがキャッチされると、新しいシグナル・マスクが計算され、そのシグナル・キャッチ関数の実行中は (または `sigprocmask` あるいは `sigsuspend` が呼び出されるまでは) そのシグナル・マスクがインストールされます。このマスクは、現在のシグナル・マスクと、`SA_NODEFER` または `SA_RESETHAND` が設定されていない場合は、配信されるシグナルの `sa_mask` の値のユニオンを計算し、さらに配信されるシグナルを加えることによって作成されます。ユーザのシグナル・ハンドラが正常に返った時点で、元のシグナル・マスクが復元されます。

いったん特定のシグナルに対するアクションがインストールされると、そのアクションは別のアクションが (sigactionの再度の呼び出しによって) 明示的に要求されるまで、SA_RESETHAND フラグがハンドラの再設定を引き起こすまで、またはいずれかのexec関数が呼び出されるまでインストールされたままとなります。

指定されたシグナルの以前のアクションがsignalによって設定されていた場合、sigactionのo_action引数がポイントする構造体に返されるフィールドの値は定められておらず、特に、o_action->sa_handlerは必ずしもsignalに渡された値と一致しません。ただし、同じ構造体へのポインタ、またはそのコピーが、sigactionのaction引数によってそれ以降のsigactionの呼び出しに渡された場合、シグナルは元のsignal呼び出しが繰り返されたかのように処理されます。

sigactionが実行に失敗した場合には、新しいシグナル・ハンドラはインストールされません。

キャッチすることも、無視することもできないシグナルに対するアクションを SIG_DFL に設定しようとしたときに、その試みが無視されるか、errnoが EINVAL に設定されてエラーが返されるかどうかは定められていません。

シグナル処理の詳細については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 4.2 節を参照してください。

注意

sigvecおよびsignal関数は、古いUNIX システムとの互換性のために用意されています。これらの関数の機能は、sigaction関数の機能のサブセットです。

sigstack, sigvec, signal, wait, read, およびwriteも参照してください。

Return value

0	成功を示します。
---	----------

- 1

エラーを示します。新しいシグナル・ハンドラはインストールされません。errnoは以下のいずれかの値に設定されます。

- EFAULT—actionまたはo_action引数は、プロセスの割り当てられたアドレス空間の外の位置をポイントしている。
- EINVAL—sig引数は有効なシグナル番号ではない。または、SIGKILL、SIGSTOP、および SIGCONT シグナルの無視、あるいはこれらのシグナルのハンドラの指定が試みられた。

sigaddset

指定された個々のシグナルを追加します。

Format

```
#include <signal.h>
int sigaddset (sigset_t *set, int sig_number);
```

Argument

set
シグナル・セット。

sig_number
個々のシグナル。

Description

sigaddset関数は、シグナルのセットを操作します。この関数は、アプリケーションからアドレス指定できるデータ・オブジェクトに作用するものであり、システムが把握している任意のシグナルのセットに適用できるわけではありません。たとえば、この関数は、プロセスへの配信がブロックされているセットや、プロセスに対して保留中となっているセットには適用できません。

sigaddset関数は、sig_numberで指定される個々のシグナルをsetによって指定されたシグナル・セットから追加します。

Example

次の例は、SIGINT シグナルのみの配信をブロックするシグナル・マスクを生成し、使用方法を示しています。

```
#include <signal.h>
int return_value;
sigset_t newset;
...
sigemptyset(&newset);
sigaddset(&newset, SIGINT);
return_value = sigprocmask (SIG_SETMASK, &newset, NULL);
```

Return value

0	成功を示します。
-1	エラーを示します。errnoには、次の値が設定されます。 <ul style="list-style-type: none">• EINVAL – sig_numberの値が有効なシグナル番号でない。

REF-667

sigdelset

指定された個々のシグナルを削除します。

Format

```
#include <signal.h>

int sigdelset (sigset_t *set, int sig_number;)
```

Argument

set
シグナル・セット。

sig_number
個々のシグナル。

Description

sigdelset関数は、setによって指定されたシグナル・セットから、sig_numberで指定される個々のシグナルを削除します。

この関数は、アプリケーションからアドレス指定できるデータ・オブジェクトに作用するものであり、システムが把握している任意のシグナルのセットに適用できるわけではありません。たとえば、この関数は、プロセスへの配信がブロックされているセットや、プロセスに対して保留中となっているセットには適用できません。

Return value

0	成功を示します。
-1	エラーを示します。errnoには、次の値が設定されます。 <ul style="list-style-type: none">EINVAL—sig_numberの値が有効なシグナル番号でない。

sigemptyset

シグナル・セットを初期化して、すべてのシグナルを除外します。

Format

```
#include <signal.h>
int sigemptyset (sigset_t *set);
```

Argument

set
シグナル・セット。

Description

sigemptyset関数は、setがポイントするシグナル・セットを初期化して、すべてのシグナルを除外します。sigset_t型のオブジェクトを他の用途に使用するためには、この型のオブジェクトに対してsigemptysetまたはsigfillset関数の呼び出しを少なくとも1回は行わなくてはなりません。

この関数は、アプリケーションからアドレス指定できるデータ・オブジェクトに作用するものであり、システムが把握している任意のシグナルのセットに適用できるわけではありません。たとえば、この関数は、プロセスへの配信がブロックされているセットや、プロセスに対して保留中となっているセットには適用できません。

sigfillsetも参照してください。

Example

次の例は、SIGINT シグナルのみの配信をブロックするシグナル・マスクを生成し、使用方法を示しています。

```
#include <signal.h>
int return_value;
sigset_t newset;
. . .
sigemptyset(&newset);
sigaddset(&newset, SIGINT);
return_value = sigprocmask (SIG_SETMASK, &newset, NULL);
```

Return value

0	成功を示します。
-1	エラーを示します。グローバルなerrnoが、エラーを示す値に設定されます。

sigfillset

シグナル・セットを初期化して、すべてのシグナルを含めます。

Format

```
#include <signal.h>
int sigfillset (sigset_t *set);
```

Argument

set
シグナル・セット。

Description

sigfillset関数は、setがポイントするシグナル・セットを初期化して、すべてのシグナルを含めます。sigset_t型のオブジェクトを他の用途に使用するためには、この型のオブジェクトに対してsigemptysetまたはsigfillsetの呼び出しを少なくとも1回は行わなくてはなりません。

この関数は、アプリケーションからアドレス指定できるデータ・オブジェクトに作用するものであり、システムが把握している任意のシグナルのセットに適用できるわけではありません。たとえば、この関数は、プロセスへの配信がブロックされているセットや、プロセスに対して保留中となっているセットには用できません。

sigemptysetも参照してください。

Return value

0	成功を示します。
-1	エラーを示します。errnoには、次の値が設定されます。 <ul style="list-style-type: none">EINVAL – sig_number引数の値が有効なシグナル番号でない。

sighold (*Alpha, I64*)

呼び出し元プロセスのシグナル・マスクに、指定されたシグナルを追加します。

Format

```
#include <signal.h>
int sighold (int signal);
```

引数

signal

シグナル・マスクに追加するシグナル。signal引数には、SIGKILLとSIGSTOPを除き、<signal.h>ヘッダ・ファイルで定義されている任意のシグナルを指定できます。

Description

sighold, sigrelse, および sigignore 関数では、簡易なシグナル管理を行うことができます。

- sighold 関数は、呼び出し元プロセスのシグナル・マスクに signal を追加します。
- sigrelse 関数は、呼び出し元プロセスのシグナル・マスクから、signal を削除します。
- sigignore 関数は、signal の処理方法として SIG_IGN を設定します。

sighold 関数を sigrelse および sigpause と合わせて使用すると、コードのクリティカル・セクションでシグナルの配信を一時的に保留させることができます。

成功すると、sighold 関数は値 0 を返します。失敗すると、値 -1 を返し、エラーを示す値を errno に設定します。

注意

これらのインタフェースは、互換性のためにのみ用意されています。新しいプログラムでは、sigaction と sigprocmask を使用してシグナルの処理方法を制御してください。

Return value

0	成功を示します。
-1	エラーを示します。errnoには、次の値が設定されます。 <ul style="list-style-type: none">• EINVAL – signal引数の値が不正なシグナル番号、または SIGKILL です。

sigignore (Alpha, I64)

指定されたシグナルの処理方法として、SIG_IGN を設定します。

Format

```
#include <signal.h>
int sigignore (int signal);
```

引数

signal
SIG_IGN を設定するシグナル。signal 引数には、SIGKILL と SIGSTOP を除き、<signal.h>ヘッダ・ファイルで定義されている任意のシグナルを指定できます。

Description

sighold、sigrelse、およびsigignore関数では、簡易なシグナル管理を行うことができます。

- sighold関数は、呼び出し元プロセスのシグナル・マスクにsignalを追加します。
- sigrelse関数は、呼び出し元プロセスのシグナル・マスクから、signalを削除します。
- sigignore関数は、signalの処理方法として SIG_IGN を設定します。

sighold関数をsigrelseおよびsigpauseと合わせて使用すると、コードのクリティカル・セクションでシグナルの配信を一時的に保留させることができます。

成功すると、sigignore関数は値 0 を返します。失敗すると、値-1 を返し、エラーを示す値をerrnoに設定します。

注意

これらのインタフェースは、互換性のためにのみ用意されています。新しいプログラムでは、sigactionとsigprocmaskを使用してシグナルの処理方法を制御してください。

Return value

0	成功を示します。
-1	エラーを示します。errnoには、次の値が設定されます。 <ul style="list-style-type: none">• EINVAL – signal引数の値が不正なシグナル番号、または SIGKILL です。または、キャッチできないシグナルをキャッチしようとしたか、無視できないシグナルを無視しようとした。

sigismember

指定されたシグナルがシグナル・セットのメンバであるかどうかをテストします。

Format

```
#include <signal.h>

int sigismember (const sigset_t *set, int sig_number);
```

Argument

set
シグナル・セット。

sig_number
個々のシグナル。

Description

sigismember関数は、sig_numberが、setがポイントするシグナル・セットのメンバであるかどうかをテストします。

この関数は、アプリケーションからアドレス指定できるデータ・オブジェクトに作用するものであり、システムが把握している任意のシグナルのセットに適用できるわけではありません。たとえば、この関数は、プロセスへの配信がブロックされているセットや、プロセスに対して保留中となっているセットには適用できません。

Return value

1	成功を示します。指定されたシグナルは、指定されたセットのメンバです。
0	エラーを示します。指定されたシグナルは、指定されたセットのメンバではありません。

siglongjmp

シグナル処理による非ローカル goto。

Format

```
#include <setjmp.h>

void siglongjmp (sigjmp_buf env, int value);
```

Argument

env
sigjmp_buf構造体のアドレス。

value
ゼロ以外の値。

Description

siglongjmp関数は、同じプロセス内で、対応するsigjmp_buf引数を使って行われた直前のsigsetjmp呼び出しによって保存された環境を復元します。

すべてのアクセス可能なオブジェクトは、siglongjmpの呼び出し時に値を持ちます。唯一の例外として、sigsetjmp呼び出しとsiglongjmp呼び出しの間に変更された自動記憶時間のオブジェクトの値は不定です。

siglongjmpは通常の間数呼び出しとリターン・メカニズムをバイパスするので、割り込み、シグナル、およびそれらの関連する関数の実行中も正しく動作します。ただし、siglongjmpをネストしたシグナル・ハンドラ (たとえば、別のシグナルの処理中に生成したシグナルの結果として呼び出された関数) から呼び出した場合の動作は未定義です。

siglongjmp関数は、ゼロ以外のsavemask引数でsigsetjmpを呼び出してenv引数を初期化した場合にのみ、保存されていたシグナル・マスクを復元します。

siglongjmpの完了後、プログラムは、対応するsigsetjmpの呼び出しがvalueによって指定された値で返ったかのように実行を続けます。siglongjmp関数は、sigsetjmpに 0 (ゼロ) を返させることはできません。valueが 0 ならば、sigsetjmpは 1 を返します。

sigsetjmpも参照してください。

sigmask

指定されたシグナル番号のためのマスクを作成します。

Format

```
#include <signal.h>
int sigmask (signum);
```

Argument

signum
マスクを作成するシグナル番号。

Description

sigmask関数は、指定されたsignumのためのマスクを作成するために使用されます。このマスクは、sigblock関数で使うことができます。

Return value

x signumのために作成されたマスク。

signal

シグナルsigの処理方法を指定することができます。シグナルのデフォルト処理を使用するか、シグナルを無視するか、または指定されたアドレスのシグナル・ハンドラを呼び出すことができます。

Format

```
#include <signal.h>

void (*signal (int sig, void (*func) (int))) (int);
```

Argument

sig

シグナルに関連付けられた番号またはニーモニック。この引数は、通常は<signal.h>ヘッダ・ファイルに定義されているニーモニックの1つです。

func

シグナルが生成されたときに実行するアクション、またはシグナルの処理に必要な関数のアドレス。

Description

funcが定数 SIG_DFL だった場合、指定されたシグナルに対するアクションはデフォルトのアクション、すなわち受信側プロセスの終了に再設定されます。引数が SIG_IGN だった場合には、シグナルは無視されます。すべてのシグナルを無視できるわけではありません。

funcが SIG_DFL でも SIG_IGN でもない場合、このfuncはシグナル処理関数のアドレスを指定します。シグナルが生成されると、アドレスが指定された関数が、sigを引数として指定して呼び出されます。アドレスが指定された関数が返ると、割り込まれたプロセスは、割り込みのポイントから実行を続けます (これはシグナルのキャッチと呼ばれます。シグナルは、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第4章に示した例外のケースを除き、キャッチ後は SIG_DFL に再設定されます)。

signal関数は、シグナルをキャッチしようとするたびに呼び出す必要があります。

シグナル処理の詳細については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第4.2節を参照してください。

OpenVMS 例外またはシグナルで UNIX スタイルのシグナルを生成させるためには、OpenVMS 条件ハンドラは、自分で処理する気がない例外を受信したときに SSS_RESIGNAL を返さなくてはなりません。SSS_CONTINUE を返すと、UNIX スタイル・シグナルは正しく生成されません。UNIX シグナルに対応する OpenVMS 例外のリストについては、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 4 章を参照してください。

Return value

x	以前にシグナルを処理するように設定された関数のアドレス。
SIG_ERR	sig 引数が範囲外であることを示します。

Format

Argument

Description

Return value

REF-681

sigpending

保留中のシグナルを確認します。

Format

```
#include <signal.h>
int sigpending (sigset_t *set);
```

Argument

set
sigset_t構造体へのポインタ。

Description

sigpending関数は、配信がブロックされ、呼び出し元プロセスに対して保留中になっているシグナルのセットを、set引数がポイントする位置に格納します。

sigset_t型のオブジェクトを他の用途に使用するためには、この型のオブジェクトに対してsigemptysetまたはsigfillsetの呼び出しを少なくとも1回は行わなくてはなりません。この方法でオブジェクトを初期化せずに、sigpending関数への引数を指定した場合の結果は未定義です。

sigemptysetとsigfillsetも参照してください。

Return value

0	成功を示します。
-1	エラーを示します。errnoは次の値に設定されます。 <ul style="list-style-type: none">• SIGSEGV— 不正なマスク引数。

sigprocmask

現在のシグナル・マスクを設定します。

Format

```
#include <signal.h>

int sigprocmask (int how, const sigset_t *set, sigset_t *o_set);
```

Argument

how

マスクされたシグナルのセットをどのように変更するかを示す整数値。以下のいずれかの値を使用します。

SIG_BLOCK	結果として得られるセットは、現在のセットと、set引数がポイントしているシグナル・セットのユニオンです。
SIG_UNBLOCK	結果として得られるセットは、現在のセットと、set引数がポイントしているシグナル・セットの補集合のインタセクションです。
SIG_SETMASK	結果として得られるセットは、set引数がポイントしているシグナル・セットです。

set

シグナル・セット。set引数の値に応じて、次の意味を持ちます。

- NULL でない — 現在ブロックされているセットを変更するために使用されるシグナルのセットをポイントする。
- NULL — how 引数の値は意味を持たず、プロセス・シグナル・マスクは変更されない。したがって、この呼び出しは現在ブロックされているシグナルを照会する目的に使用できる。

o_set

呼び出しの時点で有効だったシグナル・マスクが格納される位置への NULL でないポインタ。

Description

sigprocmask関数は、呼び出し元プロセスのシグナル・マスクの確認または変更に使
用されます。

一般に、コードのクリティカル・セクションの間はsigprocmaskで SIG_BLOCK 値を
使用してシグナルをブロックし、sigprocmaskで SIG_BLOCK 値から返された以前の
値にマスクを復元するときにはsigprocmaskで SIG_SETMASK 値を使用します。

sigprocmask関数の呼び出しの後に、保留中になっているブロックされていないシグ
ナルが存在する場合、これらのシグナルのうちの少なくとも1つが、sigprocmask関
数が返る前に配信されます。

sigprocmask関数では SIGKILL シグナルや SIGSTOP シグナルをブロックすることは
できません。プログラムがこれらのシグナルの1つをブロックしようと試みた場合で
も、sigprocmask関数はエラーを発生させません。

Example

次の例は、SIGINT シグナルのみの配信をブロックするシグナル・マスクを設定する
方法を示しています。

```
#include <signal.h>

int return_value;
sigset_t newset;
...
sigemptyset(&newset);
sigaddset(&newset, SIGINT);
return_value = sigprocmask (SIG_SETMASK, &newset, NULL);
```

Return value

- | | |
|----|--|
| 0 | 成功を示します。 |
| -1 | エラーを示します。プロセスのシグナル・マスクは変更
されず、errnoは以下のいずれかの値に設定されます。 <ul style="list-style-type: none">• EINVAL—how引数の値が、どの定義済みの値とも等
しくない。• EFAULT—setまたはo_set引数は、プロセスの割り当
てられたアドレス空間の外の位置をポイントしてい
る。 |

sigrelse (Alpha, I64)

呼び出し元プロセスのシグナル・マスクから、指定されたシグナルを削除します。

Format

```
#include <signal.h>
int sigrelse (int signal);
```

引数

signal

シグナル・マスクから削除するシグナル。signal引数には、SIGKILL と SIGSTOP を除き、<signal.h>ヘッダ・ファイルで定義されている任意のシグナルを指定できます。

Description

sighold、sigrelse、およびsigignore関数では、簡易なシグナル管理を行うことができます。

- sighold関数は、呼び出し元プロセスのシグナル・マスクにsignalを追加します。
- sigrelse関数は、呼び出し元プロセスのシグナル・マスクから、signalを削除します。
- sigignore関数は、signalの処理方法として SIG_IGN を設定します。

sighold関数をsigrelseおよびsigpauseと合わせて使用すると、コードのクリティカル・セクションでシグナルの配信を一時的に保留させることができます。

成功すると、sigrelse関数は値 0 を返します。失敗すると、値-1 を返し、エラーを示す値をerrnoに設定します。

注意

これらのインタフェースは、互換性のためにのみ用意されています。新しいプログラムでは、sigactionとsigprocmaskを使用してシグナルの処理方法を制御してください。

Return value

0	成功を示します。
-1	エラーを示します。errnoには、次の値が設定されます。 <ul style="list-style-type: none">EINVAL – signal引数の値が不正なシグナル番号、または SIGKILL です。

sigsetjmp

非ローカル goto のジャンプ・ポイントを設定します。

Format

```
#include <setjmp.h>

init sigsetjmp (sigjmp_buf env, int savemask);
```

Argument

env

sigjmp_buf 構造体のアドレス。

savemask

現在のシグナル・マスクを保存する必要があるかどうかを指定する整数値。

Description

sigsetjmp関数は、呼び出し元環境を、後にsiglongjmp関数でできるようにenv引数に保存します。

savemaskの値が 0 (ゼロ) でない場合、sigsetjmpはプロセスの現在のシグナル・マスクも、呼び出し元環境の一部として保存します。

siglongjmpも参照してください。

制限事項

OpenVMS 条件ハンドラからlongjmp関数を呼び出すことはできません。ただし、以下のネスト制約の範囲内で、HP C RTL がサポートしている任意のシグナルに対して確立されたシグナル・ハンドラからlongjmpを呼び出すことができます。

- longjmp関数は、ネストしたシグナル・ハンドラから呼び出された場合には動作しない。他のシグナル・ハンドラ内で生成された例外の結果として実行されたシグナル・ハンドラから呼び出されたlongjmp関数の結果は未定義である。
- 対応するlongjmpを、シグナルの処理が完了する前に発行したい場合を除いて、シグナル・ハンドラからsigsetjmp関数を呼び出してはならない。

- 終了ハンドラ (atexitまたは SYSSDCLEXH で設定) の中から longjmp 関数を呼び出してはならない。終了ハンドラはイメージのティアダウンの後に呼び出されるので、longjmp のデスティネーション・アドレスは存在しなくなっている。
- シグナル・ハンドラの中から、メインの実行スレッドに戻るために longjmp を呼び出すと、プログラムの状態の一貫性が失われることがある。副作用として、I/O が実行できなくなったり、UNIX シグナルを受信できなくなったりする可能性がある。代わりに siglongjmp を使用すること。

Return value

0	成功を示します。
ゼロ以外	siglongjmp 関数の呼び出しが行われます。

sigstack (VAX only)

シグナルの処理に使用される代替スタックを定義します。これにより、カレント・プロセスとは別の環境でシグナルを処理できるようになります。この関数は非リエントラントです。

Format

```
#include <signal.h>

int sigstack (struct sigstack *ss, struct sigstack *oss);
```

Argument

ss

ssが NULL でない場合には、シグナルを配信するシグナル・スタックとして使用される指定されたメモリ・セクションへのポインタを保持する構造体のアドレスを指定します。

oss

ossが NULL でない場合には、スタック・アドレスの古い値が返される構造体のアドレスを指定します。

Description

sigstack構造体は、標準ヘッダ・ファイル<signal.h>に定義されています。

```
struct sigstack
{
    char    *ss_sp;
    int     ss_onstack;
};
```

sigvec関数が、シグナル・ハンドラをシグナル・スタック上で実行するように指定した場合、システムはプロセスがすでにそのスタック上で実行されているかどうかをチェックします。プロセスがシグナル・スタック上で実行されていないければ、システムはシグナル・ハンドラが実行されている間はシグナル・スタックに切り替えるように調整を行います。oss引数が NULL でなければ、シグナル・スタックの現在の状態が返されます。

シグナル・スタックには適切な量のストレージが割り当てられなくてはなりません。ランタイム・スタックのように拡張されることはないからです。たとえば、シグナル・ハンドラがprintfや、これと同じほど複雑なHP C RTL ルーチンを呼び出す場合には、シグナル・スタック用に少なくとも 12,000 バイトのストレージを割り当てる必要があります。スタックがオーバーフローを起こすと、エラーが発生します。

ss_spは、割り当てられたメモリ領域の終端よりも少なくとも 4 バイト前の位置をポイントしていなくてはなりません (例を参照)。これはアーキテクチャに依存しており、他のマシン・アーキテクチャやオペレーティング・システムへの移植性はないと考えられます。

Return value

0	成功を示します。
-1	失敗を示します。

Example

```
#define ss_size 15000
static char mystack[ss_size];
struct sigstack ss = {&mystack + sizeof(mystack) - sizeof(void *), 1};
```

sigsuspend

ブロックされているシグナルのセットをアトミックに変更し、シグナルを待ちます。

Format

```
#include <signal.h>

int sigsuspend (const sigset_t *signal_mask);
```

Argument

signal_mask
シグナルのセットへのポインタ。

Description

sigsuspend関数は、プロセスのシグナル・マスクを、signal_mask引数がポイントするシグナルのセットに置き換えます。その後、アクションとしてシグナル・キャッチ関数を実行するか、プロセスを終了させるシグナルが配信されるまで、プロセスの実行を中断します。sigsuspend関数では、SIGKILL シグナルや SIGSTOP シグナルをブロックすることはできません。プログラムがこれらのシグナルのブロックを試みた場合でも、sigsuspendはエラーを発生させません。

シグナルの配信によってプロセスが終了する場合、sigsuspendは返りません。シグナルの配信によってシグナル・キャッチ関数が実行される場合、sigsuspendはシグナル・キャッチ関数が返った後に返り、シグナル・マスクはsigsuspendの呼び出しの前のセットに復元されます。

sigsuspend関数は、1つのアトミックな操作として、シグナル・マスクを設定し、ブロックされていないシグナルを待ちます。つまり、マスクの設定とシグナルを待つ操作の間にシグナルを発生させることはできません。プログラムがsigprocmask SIG_SETMASK とsigsuspendを別々に呼び出した場合、これらの関数の間に発生したシグナルは、通常はsigsuspendには意識されません。

通常の用途では、シグナルはクリティカル・セクションの先頭で、sigprocmask関数を使ってブロックされます。その後、プロセスは実行すべき作業があるかどうかを調べます。作業がなければ、プロセスは、前にsigprocmaskから返されたマスクを使ってsigsuspendを呼び出して、作業を待ちます。

シグナルが呼び出し元プロセスによってキャッチされ、制御がシグナル・ハンドラから返った場合、呼び出し元プロセスはsigsuspendの後から実行を再開します。sigsuspendはつねに値-1を返し、errnoをEINTRに設定します。

sigpauseとsigprocmaskも参照してください。

sigtimedwait (Alpha, I64)

呼び出し元スレッドを中断し、シグナル通知が到着するのを待ちます。

Format

```
#include <signal.h>

int sigtimedwait (const sigset_t set, siginfo_t *info, const struct timespec *timeout);
```

Argument

set

待機するシグナルのセット。

info

siginfo構造体へのポインタ。この構造体には、シグナルをポストする際に指定されたアプリケーション定義データなど、シグナルを示すデータが格納されます。

timeout

待機する際のタイムアウト時間。timeoutが NULL の場合、この引数は無視されます。

Description

sigtimedwait関数は、sigwaitinfo関数と同じ動作を行います。ただし、setで指定されたシグナルが保留されていない場合、sigtimedwaitはtimeoutが指すtimespec構造体で指定された時間だけ待機します。timeoutが指すtimespec構造体の値がゼロで、setで指定されたシグナルが保留されていない場合、sigtimedwaitはすぐにエラーで戻ります。

sigwaitとsigwaitinfoも参照してください。

シグナル処理についての詳細は、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 4.2 節を参照してください。

Return value

x	成功して終了すると、選択されたシグナル番号が返されます。
-1	<p>エラーの発生を示します。errnoには、以下のいずれかの値が設定されます。</p> <ul style="list-style-type: none">• EINVAL – timeout 引数のtv_nsec値が、0 より小さいか 10 億以上でした。• EINTR – 待機が、ブロックされていないシグナルのキャッチで割り込まれました。• EAGAIN – setで指定されたシグナルが、指定されたタイムアウト期間内に生成されませんでした。

sigvec

特定のシグナルのためのハンドラを永久的に割り当てます。

Format

```
#include <signal.h>

int sigvec (int sigint, struct sigvec *sv, struct sigvec *osv);
```

Argument

sigint

シグナル識別子。

sv

sigvec構造体へのポインタ (説明のセクションを参照)。

osv

osvが NULL でない場合には、シグナルの以前の処理情報が返されます。

Description

svが NULL でない場合には、ハンドラ・ルーチンへのポインタ、指定されたシグナルを配信するときに使用されるマスク、およびシグナルを代替スタック上で処理すべきかどうかを示すフラグを含んだ構造体のアドレスを指定します。sv->onstackの値が 1 である場合、システムはシグナルをsigstackで指定されたシグナル・スタック上のプロセスに配信します。

sigvec関数は、明示的に削除されるかイメージが終了するまで存在を続けるハンドラを設定します。

sigvec構造体は、<signal.h>ヘッダ・ファイルに定義されています。

```
struct sigvec
{
    int    (*handler) ();
    int    mask;
    int    onstack;
};
```

シグナル処理の詳細については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 4.2 節を参照してください。

Return value

0	呼び出しが成功したことを示します。
-1	エラーが発生したことを示します。

sigwait (Alpha, I64)

呼び出し元スレッドを中断し、シグナル通知が到着するのを待ちます。

Format

```
#include <signal.h>

int sigwait (const sigset_t set, int *sig);
```

Argument

set
待機するシグナルのセット。

sig
選択されたシグナルのシグナル番号を返します。

Description

sigwait関数は、set引数内のシグナルの少なくとも1つが、呼び出し元プロセスの保留シグナル・セットに含まれるようになるまで、呼び出し元スレッドを中断します。この状態が発生したら、いずれかのシグナルが自動的に選択され、保留シグナル・セットから削除されます。その後、シグナルを識別するシグナル番号が、sigが指すメモリ位置に返されます。

sigwait関数が呼び出されたときに、set引数内のシグナルにブロックされていないものがある場合の動作は、規定されていません。

set引数は、セット操作関数sigemptyset、sigfillset、sigaddset、およびsigdelsetを使用して作成されます。

sigwait関数が待機している間に、配信可能な（つまり、シグナル・マスクでブロックされていない）シグナルが発生すると、そのシグナルは非同期に処理され、待機状態が中断されます。

sigtimedwaitとsigwaitinfoも参照してください。

シグナル処理についての詳細は、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第4.2節を参照してください。

Return value

0	成功すると、sigwaitはsigが指すメモリ位置に、受信したシグナルのシグナル番号を格納して、0を返します。
0 以外	エラーの発生を示します。errnoに、次の値が設定されます。 <ul style="list-style-type: none">• EINVAL – set引数が、不正なシグナル番号、またはサポートされていないシグナル番号を含んでいます。

sigwaitinfo (Alpha, I64)

呼び出し元スレッドを中断し、シグナル通知が到着するのを待ちます。

Format

```
#include <signal.h>

int sigwaitinfo (const sigset_t set, siginfo_t *info);
```

Argument

set

待機するシグナルのセット。

info

siginfo構造体へのポインタ。この構造体には、シグナルをポストする際に指定されたアプリケーション定義データなど、シグナルを示すデータが格納されます。

Description

info引数が NULL の場合、sigwaitinfo関数は、sigwait関数と同じ動作を行います。

info引数が NULL でない場合、sigwaitinfo関数は、sigwait関数と同じ動作を行います。ただし、選択されたシグナル番号は、siginfo構造体のsi_signoメンバに格納され、またシグナルの原因が、si_codeメンバに格納されます。選択されたシグナルに、キューイングされている値がある場合、キューイングされている1番目の値がキューから外され、その値がinfoのsi_valueメンバに格納されます。シグナルのキューイングに使用されているシステム・リソースは解放され、他のシグナルのキューイングに利用できるようになります。キューイングされている値がない場合、si_valueメンバの内容は未定義です。選択されたシグナルに対して、他にキューイングされているシグナルがない場合は、そのシグナルの保留表示がリセットされます。

sigtimedwaitとsigwaitも参照してください。

シグナル処理についての詳細は、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 4.2 節を参照してください。

Return value

x	成功すると、選択されたシグナル番号が返されます。
-1	エラーの発生を示します。errnoには、以下のいずれかの値が設定されます。 <ul style="list-style-type: none">• EINVAL – set引数が、不正なシグナル番号、またはサポートされていないシグナル番号を含んでいます。• EINTR – 待機が、ブロックされていないシグナルのキャッチで割り込まれました。

sin

ラジアン単位の引数の正弦を返します。

Format

```
#include <math.h>

double sin  (double x);
float sinf  (float x); (Alpha, I64)
long double sinl (long double x); (Alpha, I64)
double sind  (double x); (Alpha, I64)
float sindf  (float x); (Alpha, I64)
long double sindl (long double x); (Alpha, I64)
```

Argument

x
浮動小数点数として表現されたラジアン。

Description

sin関数は、ラジアン単位のxの正弦を計算します。

sind関数は、度単位のxの正弦を計算します。

Return value

x	引数の正弦。
NaN	x = ±Infinity or NaN; errno is set to EDOM.
0	アンダフローが発生しました。errnoは ERANGE に設定されます。

sinh

引数の双曲線正弦を返します。

Format

```
#include <math.h>

double sinh (double x);
float sinhf (float x); (Alpha, I64)
long double sinhl (long double x); (Alpha, I64)
```

Argument

x
実数。

Return value

n	引数の双曲線正弦。
HUGE_VAL	オーバーフローが発生しました。errnoは ERANGE に設定されます。
0	アンダフローが発生しました。errnoは ERANGE に設定されます。
NaN	xは NaN です。errnoは EDOM に設定されます。

sleep

現在のプロセス (プログラムがスレッド化されている場合はスレッド) の実行を、少なくともその引数が指定した秒数だけ一時停止します。

Format

```
#include <unistd.h>

unsigned int sleep (unsigned seconds); (_DECC_V4_SOURCE)
int sleep (unsigned seconds); (not _DECC_V4_SOURCE)
```

Argument

seconds
秒数。

Description

sleep関数は、指定された秒数だけ、シグナルが受信されるまで、またはプロセス (プログラムがスレッド化されている場合はスレッド) が `SYSSWAKE` の呼び出しを実行するまでスリープします。

`SIGALRM` シグナルが生成されたが、ブロックまたは無視された場合、sleep関数は返ります。その他のシグナルでは、シグナルがブロックまたは無視されてもsleepは返りません。

Return value

x	プロセスがどれだけ早く実行を再開したかを示す秒数。
0	プロセスがsecondsで指定された秒数だけスリープしていた場合。

snprintf

メモリ内の文字列に対して書式付きの出力を行います。

Format

```
#include <stdio.h>

int snprintf (char *str, size_t n, const char *format_spec, ... );
```

Argument

str

書式付きの出力を受け取る文字列のアドレス。

n

strが指すバッファのサイズ。

format_spec

書式指定を含んでいる文字列へのポインタ。書式指定と変換文字の詳細については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル(上巻)』第2章を参照してください。

...

結果として得られる型が、書式指定で与えられた変換指定に対応する式(オプション)。

変換指定がない場合、出力ソースは省略することができます。そうでない場合は、関数呼び出しは変換指定の数以上の出力ソースを持っていなくてはならず、変換指定は出力ソースの型と一致していなくてはなりません。

変換指定は、左から右の順序で出力ソースと対応づけられます。余分な出力ソースがある場合には、無視されます。

Description

snprintf関数は、sprintf関数と同等ですが、strが指すバッファのサイズを指定するn引数が追加されています。

成功して終了するとsnprintfは、nが十分に大きい場合にstrに書き込まれるバイトの数(末尾のヌル・バイトは除く)を返します。

`n` が 0 の場合は、何も書き込まれませんが、`n` が十分に大きい場合に書き込まれるバイトの数 (末尾のヌル・バイトは除く) が返されます。この場合、`str` は NULL ポインタでも構いません。0 以外の場合、`n - 1` 番目より後の出力バイトは配列には書き込まれず、破棄されます。そして、実際に配列に書き込まれたバイト列の最後にヌル・バイトが書き込まれます。

出力エラーが検出された場合、負の値が返されます。

書式指定と出力ソースについての詳細は、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 2 章を参照してください。

Return value

x	<code>n</code> が十分に大きい場合に <code>str</code> に書き込まれるバイトの数 (末尾のヌル・バイトを除く) です。
負の値	出力エラーが発生したことを示します。この関数は、 <code>errno</code> を設定します。この関数が設定する <code>errno</code> 値のリストについては、 <code>fprintf</code> を参照してください。

sprintf

メモリ内の文字列に対して書式付きの出力を行います。

Format

```
#include <stdio.h>

int sprintf (char *str, const char *format_spec, ... );
```

Argument

str

書式付きの出力を受け取る文字列のアドレス。この文字列は出力を保持できるだけの大きさを持っているものと仮定されます。

format_spec

書式指定を含んでいる文字列へのポインタ。書式指定と変換文字の詳細については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル(上巻)』第2章を参照してください。

...

結果として得られる型が、書式指定で与えられた変換指定に対応する式(オプション)。

変換指定が与えられなかった場合、出力ソースは省略することができます。そうでない場合は、関数呼び出しは変換指定の数以上の出力ソースを持っていないとせず、変換指定は出力ソースの型と一致してはいけません。

変換指定は、左から右の順序で出力ソースと対応づけられます。余分な出力ポインタがある場合には、無視されます。

Description

sprintf関数は、出力の最後にヌル文字(\\0)を付加し、*strから始まる連続したバイト内に置きます。ユーザは、十分な領域を確保しなければなりません。

次の変換指定の例を考えてみましょう。

```
#include <stdio.h>

main()
{
    int  temp = 4, temp2 = 17;
    char s[80];

    sprintf(s, "The answers are %d, and %d.", temp, temp2);
}
```

この例では、文字列sは次の内容を持ちます。

The answers are 4, and 17.

書式指定と出力ソースについての詳細は、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第2章を参照してください。

Return value

x	出力文字列に格納される文字の数。最後の null 文字は含みません。
負の値	出力エラーが発生したことを示します。関数はerrnoを設定します。この関数が設定するerrno値のリストについては、fprintfを参照してください。

sqrt

引数の平方根を返します。

Format

```
#include <math.h>
double sqrt (double x);
float sqrtf (float x); (Alpha, I64)
long double sqrtl (long double x); (Alpha, I64)
```

Argument

x
実数。

Return value

val	xが負でなかった場合, xの平方根。
0	xは負です。errnoは EDOM に設定されます。
NaN	xは NaN です。errnoは EDOM に設定されます。

srand

擬似乱数ジェネレータrandを初期化します。

Format

```
#include <stdlib.h>

void srand (unsigned int seed);
```

Argument

seed
符号なしの引数。

Description

srand関数は、引数を、それ以降のrandの呼び出しで返される擬似乱数の新しいシーケンスのシードとして使用します。

後でsrandを同じシード値で呼び出した場合、擬似乱数のシーケンスが繰り返されます。

srandの呼び出しの前にrandを呼び出した場合、最初にsrandをシード値 1 で呼び出したときと同じ擬似乱数のシーケンスが生成されます。

srand48

48 ビットの乱数ジェネレータを初期化します。

Format

```
#include <stdlib.h>

void srand48 (long int seed_val);
```

Argument

seed_val

ランダム化の開始時に使用される初期化値。この値を変更すると、ランダム化のパターンが変化します。

Description

srand48関数は乱数ジェネレータを初期化します。この関数は、プログラムの中でdrand48, lrand48, またはmrand48関数を呼び出す前に使用することができます (これは推奨はされませんが, drand48, lrand48, またはmrand48関数が、初期化関数を呼び出すことなく呼び出された場合には、定数のデフォルト・イニシアライザ値が自動的に提供されます)。

この関数は、次の線形合同式に従って、48 ビットの整数値 X_i のシーケンスを生成します。

$$X_{n+1} = (aX_n + c) \bmod m \quad n \geq 0$$

引数 m は 2^{48} に等しいので、48 ビット整数算術演算が実行されます。lcong48関数を呼び出さなかった場合、乗数値 a と加算される値 c は次のようになります。

$$\begin{aligned} a &= 5DEECE66D_{16} = 2736731631558 \\ c &= B_{16} = 138 \end{aligned}$$

初期化関数srand48は、 X_i の上位 32 ビットを、その引数に含まれている下位 32 ビットに設定します。 X_i の下位 16 ビットは、恣意的な値 $330E_{16}$ に設定されます。

drand48, lrand48, およびmrand48も参照してください。

srandom

擬似乱数ジェネレータrandomを初期化します。

Format

```
#include <stdlib.h>

int srandom (unsigned seed);
```

Argument

seed
シードの初期値。

Description

srandom関数は、引数を、それ以降のrandomの呼び出しで返される擬似乱数の新しいシーケンスのシードとして使用します。この関数はsrand関数と実質的に同じ呼び出しシーケンスと初期化特性を持っていますが、よりランダムなシーケンスを生成します。

srandom関数は、現在の状態をシードの初期値で初期化します。srandom関数は、srand関数とは異なり、使用される状態情報の量が1ワードよりも多いため、以前のシードは返しません。

rand、srand、random、setstate、およびinitstateも参照してください。

Return value

0	成功を示します。状態のシードを初期化します。
-1	エラーを示します。エラーの具体的な内容はグローバルなerrnoで示されます。

sscanf

メモリ内の文字列から入力を読み込み、書式指定に従って解釈します。

Format

```
#include <stdio.h>

int sscanf (const char *str, const char *format_spec, ... );
```

Argument

str

sscanfへの入力テキストを提供する文字列のアドレス。

format_spec

書式指定を含んでいる文字列へのポインタ。書式指定と変換文字の詳細については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 2 章を参照してください。

...

結果として得られる型が、書式指定で与えられた変換指定に対応するオプションの式。

変換指定が与えられなかった場合、入力ポインタは省略することができます。そうでない場合は、関数呼び出しは変換指定の数以上の入力ポインタを持っていなくてはならず、変換指定は入力ポインタの型と一致していなくてはなりません。

変換指定は、左から右の順序で入力ソースと照合されます。余分な入力ポインタがある場合には、無視されます。

Description

次に変換指定の例を示します。

```
main ()
{
    char str[] = "4 17";
    int    temp,
           temp2;

    sscanf(str, "%d %d", &temp, &temp2);
    printf("The answers are %d and %d.", temp, temp2);
}
```

この例は、次の出力を生成します。

```
$ RUN EXAMPLE
```

```
The answers are 4 and 17.
```

書式指定と入力ポインタの詳細については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル(上巻)』第2章を参照してください。

Return value

x

照合と代入に成功した入力項目の数。

EOF

変換の前に読み込みエラーが発生したことを示します。
関数はerrnoを設定します。この関数が設定する値のリストについては、fscanfを参照してください。

ssignal

特定のシグナルが生成されたときに実行するアクションを指定することができます。

Format

```
#include <signal.h>

void (*ssignal (int sig, void (*func) (int, ... ))) (int, ... );
```

Argument

sig

シグナルに関連付けられた番号またはニーモニック。シグナル値のシンボリック定数は<signal.h>ヘッダ・ファイルに定義されています (『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 4 章を参照)。

func

シグナルが生成されたときに実行するアクション，またはシグナルが生成されたときに実行される関数のアドレス。

Description

ssignal関数は，エラー条件が発生したときの戻り値を除けば，signal関数と等価です。

signal関数は ANSI C 標準によって定義されており，ssignal関数は定義されていないため，移植性を高めるためにはsignalを使用するようにしてください。

シグナル処理の詳細については，『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 4.2 節を参照してください。

Return value

x	以前にシグナルのためのアクションとして設定された関数のアドレス。アドレスとしては値 SIG_DFL (0) または SIG_IGN (1) を使用することができます。
0	エラーを示します。このため、リターン・ステータス 0 が失敗を示すのか、前のアクションが SIG_DFL (0) だったのかを区別することはできません。

[w]standend

指定されたウィンドウのボールドフェイス属性を無効にします。standend関数はstdscrウィンドウに作用します。

Format

```
#include <curses.h>

int standend (void);

int wstandend (WINDOW *win);
```

Argument

win
ウィンドウへのポインタ。

Description

standendおよびwstandend関数は、属性_BOLDを指定して呼び出されたclrattrおよびwclrattrと等価です。

Return value

OK	成功を示します。
ERR	エラーを示します。

[w]standout

指定されたウィンドウのボールドフェイス属性を有効にします。standout関数はstdscrウィンドウに作用します。

Format

```
#include <curses.h>

int standout (void);

int wstandout (WINDOW *win);
```

Argument

win
ウィンドウへのポインタ。

Description

standoutおよびwstandout関数は、属性_BOLDを指定して呼び出されたsetattrおよびwsetattrと等価です。

Return value

OK	成功を示します。
ERR	エラーを示します。

stat

指定されたファイルに関する情報にアクセスします。

Format

```
#include <stat.h>

int stat (const char *file_spec, struct stat *buffer); (ISO POSIX-1)

int stat (const char *file_spec, struct stat *buffer, ...); (HP C Extension)
```

関数バリエーション

`_DECC_V4_SOURCE` および `_VMS_V6_SOURCE` 機能テスト・マクロを定義してコンパイルすると、OpenVMS Version 7.0 およびそれ以前の動作と等価な、`stat`関数へのローカル時刻ベースのエントリ・ポイントが使用可能となります。

機能テスト・マクロ `_USE_STD_STAT` を定義してコンパイルすると、X/Open 標準の `stat` 構造体の定義を使用する `stat` 関数のバリエーションが有効になります。機能テスト・マクロ `_USE_STD_STAT` は、`_DECC_V4_SOURCE` マクロおよび `_VMS_V6_SOURCE` マクロと互いに排他です。

Argument

`file_spec`

有効な OpenVMS または UNIX スタイルのファイル指定 (ワイルドカードは使用できません)。指定されたファイルの読み込み、書き込み、または実行の許可は不要ですが、ファイル指定の中のファイルに至るすべてのディレクトリに到達できる必要があります。UNIX スタイルのファイル指定の詳細については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1 章を参照してください。

`buffer`

`stat` 型の構造体へのポインタ。便宜のため、ヘッダ・ファイル `<stat.h>` の中で、`typedef stat_t` が `struct stat` として定義されています。

この引数は特定のファイルに関する情報を受け取ります。説明のセクションに、`buffer` がポイントする構造体のメンバを示します。

...

オプションのデフォルト・ファイル名文字列。

これはstat関数に対して指定できる、唯一のオプションのRMSキーワードです。オプションのRMSキーワードとその値の一覧については、creat関数の説明を参照してください。

Description

機能テスト・マクロ_USE_STD_STATを有効にしていない場合、従来のstat構造体が使用されます。_USE_STD_STATを有効にすると、X/Open標準に準拠したstat構造体が使用されます。

従来の stat 構造体

機能テスト・マクロ_USE_STD_STATにDISABLEを定義すると、以下に示す従来のstat構造体が使用されます。

メンバ	型	定義
st_dev	dev_t	物理デバイス名へのポインタ
st_ino[3]	ino_t	ファイルIDを受け取る3つのワード
st_mode	mode_t	ファイル・“モード”(prot, dirなど)
st_nlink	nlink_t	UNIXシステムとの互換性のため
st_uid	uid_t	オーナー・ユーザID
st_gid	gid_t	グループ・メンバ: st_uidより
st_rdev	dev_t	UNIXシステムとの互換性のため。-つねに0
st_size	off_t	ファイル・サイズ(バイト数)
st_atime	time_t	ファイルのアクセス時刻。常にst_mtimeと同じ
st_mtime	time_t	最終変更時刻
st_ctime	time_t	ファイル作成時刻
st_fab_rfm	char	レコード形式
st_fab_rat	char	レコード属性
st_fab_fsz	char	固定ヘッダ・サイズ
st_fab_mrs	unsigned	レコード・サイズ

型dev_t, ino_t, off_t, mode_t, nlink_t, uid_t, gid_t, およびtime_tは、<stat.h>ヘッダ・ファイルに定義されています。ただし、互換性を指定してコンパイルする場合(/DEFINE=_DECC_V4_SOURCE)には、dev_t, ino_t, およびoff_tのみが定義されます。

off_tデータ型は32ビット整数または64ビット整数です。64ビット・インタフェースでは2GB以上のファイル・サイズを扱うことができ、次のように、コンパイル時に_LARGEFILE機能テスト・マクロを定義することで選択することができます。

```
CC/DEFINE=_LARGEFILE
```

OpenVMS Version 7.0の段階では、時刻はEpoch (00:00:00 GMT, 1970年1月1日)後の経過秒数で指定されます。

st_mode構造体のメンバは、<stat.h>ヘッダ・ファイルに定義されているステータス情報モードです。以下にst_modeのビットを示します。

ビット	定数	定義
0170000	S_IFMT	ファイルのタイプ
0040000	S_IFDIR	ディレクトリ
0020000	S_IFCHR	キャラクタ・スペシャル
0060000	S_IFBLK	ブロック・スペシャル
0100000	S_IFREG	通常
0030000	S_IFMPC	多重化キャラクタ・スペシャル
0070000	S_IFMPB	多重化ブロック・スペシャル
0004000	S_ISUID	実行時にユーザ ID を設定
0002000	S_ISGID	実行時にグループ ID を設定
0001000	S_ISVTX	使用後もスワップされたテキストを保存
0000400	S_IREAD	読み込み許可，オーナ
0000200	S_IWRITE	書き込み許可，オーナ
0000100	S_IEXEC	実行/検索許可，オーナ

stat関数は、リモート・ネットワーク・ファイルには使用できません。

ファイルがレコード・ファイルである場合、st_sizeフィールドにはキャリッジ・コントロール情報が含まれます。このため、st_size値は、ファイルから読み込める文字数には対応しません。

また、st_sizeが正しい値を報告するように、C RTL と RMS の両方のバッファをフラッシュする必要があります。

標準準拠の stat 構造体

OpenVMS Version 8.2 では、UNIX との互換性を向上させるため、機能テスト・マクロ_USE_STD_STAT と標準準拠のstat構造体が導入されました。

_USE_STD_STAT に ENABLE を定義すると、以下の動作となります。

- 従来のstruct statの定義

struct statの古い定義は使用できなくなります。新しい機能を利用するには、アプリケーションの再コンパイルが必要となります。既存のアプリケーションは、新しい機能を使用するように再コンパイルしない限り、従来の定義および機能を引き続き使用します。

- 関数バリエーション

stat, fstat, lstat, およびftwの呼び出しでは、新しい型の構造体へのポインタが受け付けられます。これらの関数呼び出しは、それぞれ新しいライブラリ・エントリ__std_stat, __std_fstat, __std_lstat, および__std_ftwにマッピングされます。

- ほかの機能テスト・マクロとの互換性

`_DECC_V4_SOURCE` のソース・コード互換性はサポートされません。`_DECC_V4_SOURCE` と `_USE_STD_STAT` を同時に有効にしないでください。

`_VMS_V6_SOURCE` のバイナリ互換性はサポートされません。`_VMS_V6_SOURCE` と `_USE_STD_STAT` を同時に有効にしないでください。その結果、`time_t` フィールドでは、ローカル時刻ではなく UTC だけがサポートされます。

- 型の変更

以下の型の変更が有効になります。

- 32 ビットの `gid` 型 `gid_t` が使用されます。`_DECC_SHORT_GID_T` はサポートされません。
- `_LARGEFILE` オフセットが使用されます。`off_t` は 64 ビットになります。
- ファイル番号を示す型 `ino_t` は、`unsigned int` クォドワード (64 ビット) です。以前は、`unsigned short` でした。
- デバイス ID を示す型 `dev_t` は、`unsigned int` クォドワード (64 ビット) です。以前は、32 ビットの文字ポインタでした。新しい型は算術型であるため、標準に準拠しています。
- 型 `blksize_t` および `blkcnt_t` が追加されました。`unsigned int` クォドワード (64 ビット) として定義されています。

- 構造体メンバの変更

- `struct stat` に 2 つのメンバが追加されました。

```
blksize_t    st_blksize;
blkcnt_t     st_blocks;
```

X/Open 標準によれば、`st_blksize` は、指定したファイルに対するファイル・システム固有の優先入出力ブロック・サイズです。OpenVMS システムでは、`st_blksize` にはデバイス・バッファ・サイズにディスク・クラスタ・サイズを掛けた値が設定されます。`st_blocks` には、ファイルの割り当てサイズがブロック単位で設定されます。`st_blocks` を計算するために使用するブロック・サイズは、必ずしも `st_blksize` と同じである必要はなく、ほとんどの場合違った値になります。

- `struct stat` では、メンバ `st_ino` の型は `ino_t` です。以前のバージョンの C RTL では、型は `ino_t [3]` (`ino_t` 型の 3 つの配列) でした。`ino_t` がワードからクォドワードに変更されたため、このメンバのサイズが 1 ワード拡大されました。この変更の主なポイントは、ほとんどのオープン・ソース・アプリケーションと同じく、`st_ino` がスカラになる点です。
- `ino_t` の定義が新しくなったことで、ヘッダ・ファイル `<dirent.h>` をインクルードしているアプリケーションにも影響があります。`<stat.h>` では、`struct stat` の `st_ino` メンバと同様に、`struct dirent` の `d_ino` も変更となります。

- `ino_t`を構成する値へのアクセスを容易にするために、どの標準にもないマクロが<stat.h>にいくつか追加されました。

`S_INO_NUM(ino)`、`S_INO_SEQ(ino)`、`S_INO_RVN(ino)` は、それぞれ `ino` の FILES-11 ファイル番号、シーケンス番号、相対ボリューム番号を `unsigned short` として返します。

`S_INO_RVN_RVN(ino)` は、相対ボリューム番号を含む RVN フィールドのバイトを返します。

`S_INO_RVN_NMX(ino)` は、ファイル番号拡張を含む RVN フィールドのバイトを返します。

このようにして各要素を切り出すことが可能ですが、X/Open 標準の一部ではないため、ポータビリティを確保したいアプリケーションでは使用しないことをお勧めします。

• 意味の変更

`dev_t`型の値は、デバイスごとにクラスタ全体で一意となりました。デバイス名と割り当てクラスまたは SCSSYSTEMID (シングル・パスのデバイスの場合) に基づくアルゴリズムにより、X/Open 標準の要件である、このような特性を持ったデバイス ID が計算されます。一般に、ファイル番号とデバイス ID の組み合わせにより、クラスタ内のファイルが一意に識別されます。

この変更により、`stat`構造体のメンバ `st_dev` と `st_rdev` に影響があります。以前のリリースとの互換性のため、`st_rdev` には 0 または `st_dev` が設定されます。

注意 (Alpha, I64)

OpenVMS Alpha システムと I64 システムでは、`stat`、`fstat`、`utime`、および `utimes` 関数は、新しいファイル・システムの POSIX 準拠のファイル・タイムスタンプ・サポートに対応して拡張されています。

このサポートは、V7.3 およびそれ以降の OpenVMS Alpha システム上の ODS-5 デバイスでのみ使用できます。

この変更が行われる以前、`stat` および `fstat` 関数は、以下のファイル属性に基づいて `st_ctime`、`st_mtime`、および `st_atime` フィールドの値を設定していました。

```
st_ctime - ATR$C_CREDATE (ファイル作成時刻)
st_mtime - ATR$C_REVDATE (ファイル更新時刻)
st_atime - ファイル・アクセス時刻がサポートされていなかったため、常に st_mtime に設定されていた
```

また、ファイル変更時刻については、`utime` および `utimes` は `ATR$C_REVDATE` ファイル属性を変更し、ファイル・アクセス時刻引数は無視していました。

変更が行われて以降、ODS-5 デバイス上のファイルに関しては、`stat` および `fstat` 関数は、以下の新しいファイル属性に基づいて `st_ctime`、`st_mtime`、および `st_atime` フィールドの値を設定するようになりました。

stat

st_ctime - ATR\$_C_ATTDATE (最終属性変更時刻)
st_mtime - ATR\$_C_MODDATE (最終データ変更時刻)
st_atime - ATR\$_C_ACCDATE (最終アクセス時刻)

ODS-2 デバイスのように ATR\$_C_ACCDATE がゼロである場合、stat および fstat 関数は st_atime を st_mtime に設定します。

ファイル変更時刻については、utime および utimes 関数は、ATR\$_C_REVDATE と ATR\$_C_MODDATE の両方のファイル属性を変更します。ファイル・アクセス時刻については、これらの関数は ATR\$_C_ACCDATE ファイル属性を変更します。ODS-2 デバイス上で ATR\$_C_MODDATE および ATR\$_C_ACCDATE ファイル属性を設定しても効果はありません。

互換性を保つために、デバイスの種類にかかわらず、stat、fstat、utime、および utimes の以前の動作がデフォルトとなっています。

新しい動作を有効にするためには、アプリケーションを呼び出す前に、DECC\$EFS_FILE_TIMESTAMPS 論理名に明示的に ENABLE を定義する必要があります。この論理名を設定しても、ODS-2 デバイス上のファイルに対する stat、fstat、utime、および utimes の動作には影響はありません。

Return value

0	成功を示します。
-1	特権違反以外のエラーを示します。errno はエラーを示す値に設定されます。
-2	特権違反を示します。

statvfs (Alpha, I64)

指定されたファイルが格納されているデバイスに関する情報を取得します。

Format

```
#include <statvfs.h>

int statvfs (const char *restrict path, struct statvfs *restrict buffer);
```

Argument

path
マウントされているデバイス上の任意のファイル。

buffer
返される情報を保持する，statvfs構造体へのポインタ

Description

statvfs関数は，指定されたファイルが格納されているデバイスに関する情報を返します。指定されたファイルに対する読み込み，書き込み，実行の許可は必要ありません。情報は，statvfs構造体の形で返されます。この構造体はヘッダ・ファイル<statvfs.h>で定義されており，以下のメンバが含まれています。

unsigned long f_bsize - 優先ブロック・サイズ。

unsigned long f_frsize - 基本ブロック・サイズ。

fsblkcnt_t f_blocks - ブロック・サイズの合計。単位はf_frsize。

fsblkcnt_t f_bfree - 空きブロックサイズの合計。DFS ディスクに対する\$GETDVI で，無意味な空きブロック数が誤ってレポートされる場合，f_bfreeには最大ブロック数が設定されます。

fsblkcnt_t f_bavail - 利用可能な空きブロックの数。呼び出し元のディスク・クォータの未使用分が設定されます。

fsfilcnt_t f_files - ファイル・シリアル番号 (たとえば inode) の合計。

fsfilcnt_t f_ffree - 空きファイル・シリアル番号の合計。OpenVMS システムでは、この値は「空きブロック数/クラスタサイズ」で計算されます。

fsfilcnt_t f_favail - 非特権プロセスが利用できるファイル・シリアル番号の数 (OpenVMS システムでは 0)。

unsigned long f_fsid - ファイル・システム ID。この ID は、割り当てクラス・デバイス名に基づきます。デバイスがローカルにマウントされている限り、デバイスに基づいた一意の値が提供されます。

unsigned long f_flag - 以下の 1 つ以上のフラグを表すビット・マスク。

ST_RDONLY - ボリュームは読み込み専用。

ST_NOSUID - ボリュームで保護されたサブシステムが有効。

unsigned long f_namemax - ファイル名の最大長。

char f_basetype[64] - デバイス・タイプ名。

char f_fstr[64] - 論理ボリューム名。

char __reserved[64] - メディア・タイプ名。

正常に完了すると、statvfsは 0 (ゼロ) を返します。失敗すると-1 を返し、errnoにエラーを示す値を設定します。

fstatvfsも参照してください。

戻り値

0	正常終了。
---	-------

-1

エラーを示します。errnoには、以下のいずれかが設定されます。

- EACCES - パス接頭辞のコンポーネントに対する検索許可が拒否されました。
- EIO - デバイスの読み込み中に入出力エラーが発生しました。
- EINTR - 関数の実行中にシグナルを受信しました。
- EOVERFLOW - 返却する値の 1 つが、bufferが示す構造体で正しく表現できません。
- ENAMETOOLONG - パス・パラメータのコンポーネントの長さが NAME_MAX を超えたか、パス・パラメータの長さが PATH_MAX を超えました。
- ENOENT - pathのコンポーネントが、存在するファイルを示していないか、pathが空文字列です。
- ENOTDIR - pathパラメータのパス接頭辞のコンポーネントが、ディレクトリではありません。

strcasecmp

2 つの 7 ビット ASCII 文字列の , 大文字小文字を区別しない比較を行います。

Format

```
#include <strings.h>

int strcasecmp (const char *s1, const char *s2);
```

Argument

s1
比較する 2 つの文字列のうちの第 1 のもの

s2
比較する 2 つの文字列のうちの第 2 のもの

Description

strcasecmp関数は大文字小文字を区別しません。返される字句的な違いは , 小文字に変換されています。

strcasecmp関数は 7 ビット ASCII の比較のみを行います。国際化されたアプリケーションでは , この関数は使用しないようにしてください。

Return value

n
s1文字列がs2文字列よりも大きい , 等しい , または小さい場合に , それぞれ 0 よりも大きい , 等しい , または小さい整数値。

strcat

str_1の末尾に、終端の null 文字を含めてstr_2を連結します。

Format

```
#include <string.h>
char *strcat (char *str_1, const char *str_2);
```

関数バリエント

strcat関数は、それぞれ 32 ビットと 64 ビットのポインタ・サイズで使用するための_strcat32と_strcat64という名前のバリエントを持っています。ポインタ・サイズ固有の関数の使用方法については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

Argument

str_1, str_2
null で終了する文字列へのポインタ。

Description

strncatを参照してください。

Return value

x	第 1 引数str_1のアドレス。これは連結された結果を保持できるだけの大きさを持つと仮定されます。
---	--

Example

```
#include <string.h>
#include <stdio.h>

/* This program concatenates two strings using the strcat    */
/* function, and then manually compares the result of strcat */
/* to the expected result.                                   */

#define S1LENGTH 10
#define S2LENGTH 8

main()
{
    static char s1buf[S1LENGTH + S2LENGTH] = "abcmnxyz";
    static char s2buf[] = " orthis";
    static char test1[] = "abcmnxyz orthis";

    int i;
    char *status;

    /* Take static buffer s1buf, concatenate static buffer    */
    /* s2buf to it, and compare the answer in s1buf with the   */
    /* static answer in test1.                                   */

    status = strcat(s1buf, s2buf);
    for (i = 0; i <= S1LENGTH + S2LENGTH - 2; i++) {
        /* Check for correct returned string.                  */
        if (test1[i] != s1buf[i])
            printf("error in strcat");
    }
}
```

Format

関数バリエーション

Argument

Description

これとは対照的に、`strrchr`では、`null`で終了する文字列の中を先頭から調べて、指定した文字が最後に見つかったアドレスを返します。

Return value

REF-731

NULL

文字列の中に文字が含まれていないことを示します。

Example

```
#include <stdio.h>
#include <string.h>

main()
{
    static char s1buf[] = "abcdefghijkl lkjihgfedcba";
    int i;
    char *status;

    /* This program checks the strchr function by incrementally */
    /* going through a string that ascends to the middle and then */
    /* descends towards the end. */
    for (i = 0; s1buf[i] != '\0' && s1buf[i] != ' '; i++) {
        status = strchr(s1buf, s1buf[i]);

        /* Check for pointer to leftmost character - test 1. */
        if (status != &s1buf[i])
            printf("error in strchr");
    }
}
```

strcmp

2 つの ASCII 文字列を比較し，第 1 の文字列の個々の文字の ASCII 値が第 2 の文字列の値よりも小さいか，等しいか，大きいことを示す負の整数，0，または正の整数を返します。

Format

```
#include <string.h>

int strcmp (const char *str_1, const char *str_2);
```

Argument

str_1, str_2
文字列へのポインタ。

Description

文字列の比較は，null 文字が検出されるか，文字列に違いが発見されるまで行われます。

Return value

< 0	str_1がstr_2よりも小さいことを示します。
= 0	str_1がstr_2と等しいことを示します。
> 0	str_1がstr_2よりも大きいことを示します。

strcoll

2つの文字列を比較し、文字列が異なるかどうか、またどのように異なるかを示す整数を返します。この関数は、現在のロケールのLC_COLLATEカテゴリの照合情報を使用して、比較の実行方法を決定します。

Format

```
#include <string.h>

int strcoll (const char *s1, const char *s2);
```

Argument

s1, s2
文字列へのポインタ。

Description

strcoll関数は、strcmpとは異なり、2つの文字列をロケールに依存した方法で比較します。エラー条件のための値は予約されていないので、アプリケーションは関数呼び出しの前にerrnoを0に設定し、呼び出しの後にその値をテストすることでチェックを行う必要があります。

strxfrmも参照してください。

Return value

< 0	s1がs2よりも小さいことを示します。
= 0	文字列が等しいことを示します。
> 0	s1がs2よりも大きいことを示します。

Format

関数バリエーション

Argument

Description

destがポイントする領域がsourceがポイントする領域とオーバーラップしている場合、この関数の動作は未定義です。

Return value

REF-735

strcspn

指定された文字セットに含まれていない文字のみから構成される文字列の接頭辞の長さを返します。

Format

```
#include <string.h>

size_t strcspn (const char *str, const char *charset);
```

Argument

str

文字列へのポインタ。この文字列が null 文字列だった場合には、0 が返されます。

charset

文字のセットを含んだ文字列へのポインタ。

Description

strcspn関数は、文字列内の文字をスキャンし、charsetに含まれる文字を検出した時点で停止し、charsetに含まれない文字から構成される文字列の先頭のセグメントの長さを返します。

strとcharsetがポイントする文字列のどの文字も一致しない場合、strcspnは文字列の長さを返します。

Return value

x

セグメントの長さ。

strdup

指定された文字列を複製します。

Format

```
#include <string.h>
char *strdup (const char *s1);
```

関数バリエーション

strdup関数は、それぞれ 32 ビットと 64 ビットのポインタ・サイズで使用するための `_strdup32` と `_strdup64` という名前のバリエーションを持っています。ポインタ・サイズ固有の関数の使用方法については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

Argument

`s1`
複製する文字列。

Description

strdup関数は、`s1` がポイントする文字列の正確な複製である文字列へのポインタを返します。新しい文字列のためのスペースの割り当てには `malloc` 関数が使用されます。strdup関数は、既存のシステムとの互換性のために用意されています。

Return value

<code>x</code>	結果として得られる文字列へのポインタ。
<code>NULL</code>	エラーを示します。

strerror

error_codeの中のエラー番号を、ロケール依存のエラー・メッセージ文字列にマップします。

Format

```
#include <string.h>

char *strerror (int error_code); (ANSI C)

char *strerror (int error_code[, int vms_error_code]); (HP C Extension)
```

Argument

error_code
エラー・コード。

vms_error_code
OpenVMS エラー・コード。

Description

strerror関数は、error_codeの中のエラー番号を使用して、適切なロケール依存のエラー・メッセージを取得します。エラー・メッセージ文字列の内容は、プログラムの現在のロケールの LC_MESSAGES カテゴリによって決定されます。

プログラムが標準関連の機能テスト・マクロ (『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.5.1 項を参照) を指定せずにコンパイルされた場合、strerrorは以下のように使用される第 2 の引数 (vms_error_code) を取ります。

- error_codeが EVMSERR で、第 2 引数が存在する場合には、その第 2 引数がvaxc\$errno値として使用される。
- error_codeが EVMSERR で、第 2 引数が存在しない場合には、vaxc\$errnoを参照して OpenVMS エラー条件を取得する。

「例」の項を参照してください。

第 2 の引数は、strerrorのANSI Cの定義には含まれていないので、移植性はありません。

エラーを示す戻り値は予約されていないため、アプリケーションは`errno`の値を 0 に設定した後に`strerror`を呼び出し、`errno`の値をテストしなくてはなりません。ゼロ以外の値はエラー条件を示します。

戻り値

x

該当するエラー・メッセージを含んでいるバッファへのポインタ。プログラム内でこのバッファを変更しないでください。また、`strerror`関数の呼び出しにより、このバッファは新しいメッセージで上書きされることがあります。

Example

```
#include <stdio.h>
#include <errno.h>
#include <string.h>
#include <stdlib.h>
#include <ssdef.h>

main()
{
    puts(strerror(EVMSERR));
    errno = EVMSERR;
    vaxc$errno = SS$_LINKEXIT;
    puts(strerror(errno));
    puts(strerror(EVMSERR, SS$_ABORT));
    exit(1);
}
```

この例のプログラムを実行すると、次の出力が生成されます。

```
nontranslatable vms error code: <none>
network partner exited
abort
```

strfmon

金額値を文字列に変換します。変換は書式文字列によって制御されます。

Format

```
#include <monetary.h>

ssize_t strfmon (char *s, size_t maxsize, const char *format, ... );
```

Argument

s

結果として得られる文字列へのポインタ。

maxsize

結果として得られる文字列に格納される最大バイト数。

format

出力文字列の書式を制御する文字列へのポインタ。

...

出力文字列へと整形されるdouble型の金額値。formatがポイントする書式文字列の中の変換指定と同じ数の値がなくてはなりません。値が足りなかった場合、関数は実行に失敗します。余分な引数は無視されます。

Description

strfmon関数は、指定された金額値を使用して、sがポイントする文字列を作成します。最高maxsizeバイトがsにコピーされます。

formatがポイントする書式文字列は、通常の文字と変換指定から構成されています。すべての通常の文字は、変更なしに出力文字列にコピーされます。変換指定は、指定された金額値の1つが、出力文字列でどのように整形されるかを定義します。

変換指定は、パーセント文字(%),いくつかのオプションの文字(表 REF-5 を参照),および変換指定子(表 REF-6 を参照)から構成されます。

表 REF-5 に示しているオプション文字が変換指定に含まれる場合、それらの文字は表に示した順序で現れなくてはなりません。

表 REF-5 strfmon の変換指定に含まれるオプションの文字

文字	意味
=character	小数点以上の桁数を指定した場合に、characterを数値フィル文字として使用する。デフォルトの数値フィル文字はスペース文字である。フィル文字は、有効桁数と幅のカウントに対応できるように、1 バイトとして表現できなくてはならない。この変換指定子は、小数点以上の桁数が指定されなかった場合には無視される。また、幅のフィルはつねにスペース文字を使用するので、この変換指定子の影響を受けない。
^	数値の書式指定に区切り文字を使用しない。デフォルトでは、桁は現在のロケールの LC_MONETARY カテゴリのmon_groupingフィールドに従ってグループ化される。
+	現在のロケールのpositive_signまたはnegative_signフィールドで指定された文字列を追加する。p_sign_posnまたはn_sign_posnが0に設定されている場合、デフォルトでは負の値を示すために括弧が使用される。それ以外の場合には、符号文字列が値の符号を示すために使用される。同じ変換指定で+と(を同時に使用することはできない。
(負の値を括弧で囲む。デフォルトの設定には、現在のロケールのp_sign_posnまたはn_sign_posnフィールドが使用される。p_sign_posnまたはn_sign_posnが0に設定されている場合、デフォルトでは負の値を示すために括弧が使用される。それ以外の場合には、符号文字列が値の符号を示すために使用される。同じ変換指定で+と(を同時に使用することはできない。
!	通貨シンボルを抑止する。デフォルトでは、通貨シンボルが追加される。
-	フィールド内で値を左揃えにする。デフォルトでは、値は右揃えになる。
フィールド幅	変換の結果の位置を揃えるときに使用される最小のフィールド幅を指定する10進整数。デフォルトのフィールド幅は、結果を格納できる最も小さいフィールドである。
#left_precision	#の後の10進整数は、小数点以上の桁数を指定する。余った桁はフィル文字によって埋められる。デフォルトでは、引数に必要な最小限の桁数が使用される。^変換指定子でグループ化が抑止されておらず、現在のロケールでグループ化が定義されていた場合には、フィル文字が追加される前にグループ化の区切り文字が挿入される。グループ化の区切り文字は、フィル文字が数字として定義されている場合でも、フィル文字には適用されない。
.right_precision	ピリオド(.)の後の10進整数は、小数点以下の桁数を指定する。余分な桁はゼロで埋められる。値はこの小数点以下の桁数にまで丸められる。小数点以下の桁数がゼロである場合、出力には小数点は含められない。デフォルトでは、小数点以下の桁数は現在のロケールのfrac_digitsまたはint_frac_digitsフィールドによって定義される。

表 REF-6 strfmon の変換指定子

指定子	意味
i	通貨シンボルが抑止されていなければ、現在のロケールのint_currency_symbolフィールドによって定義されている国際通貨シンボルを使用する。
n	通貨シンボルが抑止されていなければ、現在のロケールのcurrency_symbolフィールドによって定義されているローカル通貨シンボルを使用する。
%	%文字を出力する。変換指定は%%でなくてはならない。この指定子では、どのオプション文字も使用できない。

Return value

x	sがポイントする文字列に書き込まれるバイト数。終端のnull文字は含みません。
-1	エラーを示します。関数はerrnoを以下のいずれかの値に設定します。 <ul style="list-style-type: none"> • EINVAL— 変換指定の構文が間違っている。 • E2BIG— 書式文字列全体を処理すると、出力がmaxsizeバイトを超える。

Example

```
#include <stdlib.h>
#include <stdio.h>
#include <locale.h>
#include <monetary.h>
#include <errno.h>

#define MAX_BUF_SIZE 124

main()
{
    size_t ret;
    char buffer[MAX_BUF_SIZE];
    double amount = 102593421;

    /* Display a monetary amount using the en_US.ISO8859-1 */
    /* locale and a range of different display formats.    */
    if (setlocale(LC_ALL, "en_US.ISO8859-1") == (char *) NULL) {
        perror("setlocale");
        exit(EXIT_FAILURE);
    }
    ret = strfmon(buffer, MAX_BUF_SIZE, "International: %i\n", amount);
    printf(buffer);

    ret = strfmon(buffer, MAX_BUF_SIZE, "National:      %n\n", amount);
    printf(buffer);

    ret = strfmon(buffer, MAX_BUF_SIZE, "National:      %=*#10n\n", amount);
    printf(buffer);

    ret = strfmon(buffer, MAX_BUF_SIZE, "National:      %(n\n", -1 * amount);
    printf(buffer);

    ret = strfmon(buffer, MAX_BUF_SIZE, "National:      %^!n\n", amount);
    printf(buffer);
}
```

この例のプログラムを実行すると、次の出力が生成されます。

```
International: USD 102,593,421.00
National:      $102,593,421.00
National:      $**102,593,421.00
National:      ($102,593,421.00)
National:      102593421.00
```

strftime

tm構造体に格納されている日付および時刻情報を使って出力文字列を作成します。出力文字列の書式は書式文字列によって制御されます。

Format

```
#include <time.h>

size_t strftime (char *s, size_t maxsize, const char *format, const struct tm *timeptr);
```

関数バリエーション

_DECC_V4_SOURCE および _VMS_V6_SOURCE 機能テスト・マクロを定義してコンパイルすると、OpenVMS Version 7.0 およびそれ以前の動作と等価な、strftime関数へのローカル時刻ベースのエントリ・ポイントが使用可能となります。

Argument

s
結果として得られる文字列へのポインタ。

maxsize
結果として得られる文字列に格納される最大バイト数 (終端の null を含みます)。

format
出力文字列の書式を制御する文字列へのポインタ。

timeptr
ローカル時刻 (tm) 構造体へのポインタ。tm構造体は<time.h>ヘッダ・ファイルに定義されています。

Description

strftime関数は、timeptrがポイントする構造体に含まれているデータを使用して、sがポイントする文字列を作成します。最高maxsizeバイトがsにコピーされます。

書式文字列は、0 個以上の変換指定と通常の文字から構成されています。すべての通常の文字は (終端の null 文字を含めて)、変更なしに出力文字列にコピーされます。変換指定は、tm構造体の中のデータが、出力文字列でどのように整形されるかを定義します。

変換指定は、パーセント文字 (%), 1 つまたは複数のオプションの文字 (表 REF-7 を参照), および変換指定子 (表 REF-8 を参照) から構成されます。表 REF-7 に示しているオプション文字が指定される場合、それらの文字は表に示した順序で現れなくてはなりません。

strptime関数は、tzsetを呼び出した場合と同じように動作します。

表 REF-7 strptime 変換指定のオプション要素

要素	意味
-	フィールド幅にオプションとして付け、そのフィールドが左揃えされ、スペースでパディングされることを示す。0 要素と同時に使用することはできない。
0	フィールド幅にオプションとして付け、そのフィールドが右揃えされ、ゼロでパディングされることを示す。-要素と同時に使用することはできない。
フィールド幅	最大フィールド幅を指定する 10 進整数。
.precision	フィールド内のデータの精度を指定する 10 進整数。 d, H, I, j, m, M, o, S, U, w, W, y,およびY変換指定子では、精度指定子は、フィールド内の桁数の最小値である。変換指定が精度によって指定された桁数よりも少ない場合には、先頭にゼロが追加される。 a, A, b, B, c, D, E, h, n, N, p, r, t, T, x, X, Z,および%変換指定子では、精度指定子は、フィールド内の文字数の最大値である。変換指定が精度によって指定された桁数よりも多くの文字を含んでいる場合には、右側の文字が切り捨てられる。 d, H, I, m, M, o, S, U, w, W, yおよびY変換指定子のデフォルトの精度は 2 である。j変換指定子のデフォルトの精度は 3 である。

表 REF-7 の変換指定のリストは、XPG4 仕様の拡張であることに注意してください。

表 REF-8 は変換指定子を示しています。strptime関数は、プログラムの現在のロケールのLC_TIMEカテゴリのフィールドから値を取得します。たとえば、%Bが指定されている場合、関数はLC_TIMEのmonフィールドにアクセスして、tm構造体で指定された月の完全な名前を取得します。無効な変換指定子を使用したときの結果は未定義です。

表 REF-8 strptime の変換指定子

指定子	置き換え
a	ロケールの短縮された曜日名。

(次ページに続く)

表 REF-8 (続き) strftime の変換指定子

指定子	置き換え
A	ロケールの完全な曜日名。
b	ロケールの短縮された月の名前。
B	ロケールの完全な月の名前。
c	ロケールの適切な日付および時刻表現。
C	10 進数 (00 ~ 99) として表現される世紀 (年を 100 で割り、整数に切り捨て)。
d	その月の 10 進数 (01 ~ 31) として表現される日付。
D	%m/%d/%yと同じ。
e	先頭がスペース文字でフィルされた 2 桁のフィールドに格納される、その月の 10 進数 (1 ~ 31) として表現される日付。
Ec	ロケールの代替日付および時刻表現。
EC	ロケールの代替表現における基本年 (期間) の名前。
Ex	ロケールの代替日付表現。
EX	ロケールの代替時刻表現。
Ey	ロケールの代替表現における基本年 (%EC) からのオフセット。
EY	ロケールの完全な代替年表現。
h	%bと同じ。
H	10 進数 (00 ~ 23) としての時刻 (24 時間制)。
I	10 進数 (01 ~ 12) としての時刻 (12 時間制)。
j	10 進数 (001 ~ 366) としての、その年の中での日。
m	10 進数 (01 ~ 12) としての月。
M	10 進数 (00 ~ 59) としての分。
n	改行文字。
Od	ロケールの代替数値シンボルを使用した、その月の中での日。
Oe	ロケールの代替数値シンボルを使用した、その月の中での日付。
OH	ロケールの代替数値シンボルを使用した時刻 (24 時間制)。
OI	ロケールの代替数値シンボルを使用した時刻 (12 時間制)。
Om	ロケールの代替数値シンボルを使用した月。
OM	ロケールの代替数値シンボルを使用した分。
OS	ロケールの代替数値シンボルを使用した秒。
Ou	ロケールの代替表現での曜日を数値で表したもの (月曜日=1)。
OU	ロケールの代替数値シンボルを使用した、その年の中での週 (週は日曜日から始まる)。
OV	ロケールの代替数値シンボルを使用した、10 進数 (01 ~ 53) としての、その年の中での週 (週は月曜日から始まる)。1 月 1 日を含んでいる週が、新年に 4 日以上ある場合には、その週が 1 番目の週と見なされる。そうでない場合は、前年の 53 番目の週と見なされ、次の週が 1 番目の週となる。
Ow	ロケールの代替数値シンボルを使用した、数値としての曜日 (日曜日=0)。
OW	ロケールの代替数値シンボルを使用した、その年の中での数値としての週 (週は月曜日から始まる)。

(次ページに続く)

表 REF-8 (続き) strftime の変換指定子

指定子	置き換え
Oy	ロケールの代替数値シンボルを使用した, 世紀を除いた年。
p	ロケールの 12 時間制における AM/PM 指定。
r	AM/PM 表記での時刻。
R	24 時間表記での時刻 (%H:%M)。
S	10 進数 (00 ~ 61) としての秒。
t	タブ文字。
T	時刻 (%H:%M:%S)。
u	1 ~ 7 の範囲の 10 進数としての曜日 (月曜日=1)。
U	10 進数 (00 ~ 53) としての, その年の中の週 (最初の日曜日が 1 番目の週の最初の日と見なされる)。
V	10 進数 (00 ~ 53) としての, その年の中の週 (週は月曜日から始まる)。1 月 1 日を含んでいる週が, 新年に 4 日以上ある場合には, その週が 1 番目の週と見なされる。そうでない場合は, 前年の 53 番目の週と見なされ, 次の週が 1 番目の週となる。
w	10 進数 (0 [日曜日] ~ 6) としての曜日。
W	10 進数 (00 ~ 53) としての, その年の中での週 (最初の月曜日が 1 番目の週の最初の日と見なされる)。
x	ロケールの適切な日付表現。
X	ロケールの適切な時刻表現。
Y	10 進数 (00 ~ 99) としての, 世紀を除いた年。
Y	10 進数としての, 世紀を含んだ年。
Z	タイム・ゾーン名またはその短縮形。タイム・ゾーン情報がない場合には, 文字は出力されない。
%	リテラルの%文字。

Return value

x	sがポイントする配列に格納された文字数。終端の null 文字は含まれません。
0	エラーが発生したことを示します。配列の内容は不定です。

Example

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <locale.h>
#include <errno.h>

#define NUM_OF_DATES 7
#define BUF_SIZE 256
```

strftime

```
/* This program formats a number of different dates, once */
/* using the C locale and then using the fr_FR.ISO8859-1 */
/* locale. Date and time formatting is done using strftime(). */

main()
{
    int count,
        i;
    char buffer[BUF_SIZE];
    struct tm *tm_ptr;
    time_t time_list[NUM_OF_DATES] =
        {500, 68200000, 694223999, 694224000,
         704900000, 705000000, 705900000};

    /* Display dates using the C locale */
    printf("\nUsing the C locale:\n\n");
    setlocale(LC_ALL, "C");
    for (i = 0; i < NUM_OF_DATES; i++) {
        /* Convert to a tm structure */
        tm_ptr = localtime(&time_list[i]);

        /* Format the date and time */
        count = strftime(buffer, BUF_SIZE,
                         "Date: %A %d %B %Y\nTime: %T%n\n", tm_ptr);
        if (count == 0) {
            perror("strftime");
            exit(EXIT_FAILURE);
        }

        /* Print the result */
        printf(buffer);
    }

    /* Display dates using the fr_FR.ISO8859-1 locale */
    printf("\nUsing the fr_FR.ISO8859-1 locale:\n\n");
    setlocale(LC_ALL, "fr_FR.ISO8859-1");
    for (i = 0; i < NUM_OF_DATES; i++) {
        /* Convert to a tm structure */
        tm_ptr = localtime(&time_list[i]);

        /* Format the date and time */
        count = strftime(buffer, BUF_SIZE,
                         "Date: %A %d %B %Y\nTime: %T%n\n", tm_ptr);
        if (count == 0) {
            perror("strftime");
            exit(EXIT_FAILURE);
        }

        /* Print the result */
        printf(buffer);
    }
}
```

この例のプログラムを実行すると、次の出力が生成されます。

Using the C locale:

Date: Thursday 01 January 1970
Time: 00:08:20

Date: Tuesday 29 February 1972
Time: 08:26:40

Date: Tuesday 31 December 1991
Time: 23:59:59

Date: Wednesday 01 January 1992
Time: 00:00:00

Date: Sunday 03 May 1992
Time: 13:33:20

Date: Monday 04 May 1992
Time: 17:20:00

Date: Friday 15 May 1992
Time: 03:20:00

Using the fr_FR.ISO8859-1 locale:

Date: jeudi 01 janvier 1970
Time: 00:08:20

Date: mardi 29 f rier 1972
Time: 08:26:40

Date: mardi 31 d embre 1991
Time: 23:59:59

Date: mercredi 01 janvier 1992
Time: 00:00:00

Date: dimanche 03 mai 1992
Time: 13:33:20

Date: lundi 04 mai 1992
Time: 17:20:00

Date: vendredi 15 mai 1992
Time: 03:20:00

strlen

ASCII 文字列の長さを返します。返される長さには、終端の null 文字 (\0) は含まれません。

Format

```
#include <string.h>

size_t strlen (const char *str);
```

Argument

str
文字列へのポインタ。

Return value

x	文字列の長さ。
---	---------

strncasecmp

2 つの 7 ビット ASCII 文字列の間で、大文字小文字を区別しない比較を行います。

Format

```
#include <strings.h>

int strncasecmp (const char *s1, const char *s2, size_t n);
```

Argument

s1
比較する 2 つの文字列のうちの第 1 のもの。

s2
比較する 2 つの文字列のうちの第 2 のもの。

n
比較する文字列に含まれる最大バイト数。

Description

strncasecmp関数は大文字小文字を区別しません。返される字句的な違いは、小文字に変換されています。strncasecmp関数はstrcasecmp関数に似ていますが、サイズの比較も行います。NULL よりも前にnによって指定されるサイズが読み込まれた場合には、比較は停止します。

strcasecmp関数は 7 ビット ASCII の比較のみを行います。国際化されたアプリケーションでは、この関数は使用しないようにしてください。

Return value

n	s1文字列がs2文字列よりも大きい、等しい、または小さい場合に、それぞれ 0 よりも大きい、等しい、または小さい整数値。
---	--

strncat

str_2からstr_1の末尾に、maxchar個以下の文字を付加します。

Format

```
#include <string.h>

char *strncat (char *str_1, const char *str_2, size_t maxchar);
```

関数バリエント

strncat関数は、それぞれ 32 ビットと 64 ビットのポインタ・サイズで使用するための `_strncat32` と `_strncat64` という名前のバリエントを持っています。ポインタ・サイズ固有の関数の使用方法については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

Argument

str_1, str_2

null で終了する文字列へのポインタ。

maxchar

str_2から連結する文字数。先にstr_2の中でstrncatが終端の null を検出した場合には、そこで終了します。maxcharが 0 の場合には、str_2から文字のコピーは行われません。

Description

strncat関数の結果には、つねに null 文字が付加されます。strncatは、指定された上限に達すると、str_1の中の次のバイトを null 文字に設定します。

Return value

x

第 1 引数str_1のアドレス。これは、連結された結果を保持できるだけの大きさを持つと仮定されます。

strncmp

2 つの ASCII 文字列の間でmaxchar個以下の文字を比較し、第 1 の文字列に含まれる個々の文字の ASCII 値が第 2 の文字列の値よりも小さい、等しい、または大きいことを示す負の整数、0、または正の整数を返します。

Format

```
#include <string.h>

int strncmp (const char *str_1, const char *str_2, size_t maxchar);
```

Argument

str_1, str_2
文字列へのポインタ。

maxchar
str_1とstr_2の両方で検索する文字数の上限 (最初の文字を含みます)。maxcharが 0 の場合には、比較は行われず、0 が返されます (文字列は等しいものと見なされます)。

Description

strncmp関数は、str_1がポイントする文字列のmaxchar個以下の文字を、str_2がポイントする文字列と比較します。文字列の比較は、null 文字が検出されるか、文字列に違いが発見されるか、maxcharに達するまで行われます。違いが発見された後の文字や null 文字の後の文字は比較されません。

Return value

< 0	str_1がstr_2よりも小さいことを示します。
= 0	str_1がstr_2と等しいことを示します。
> 0	str_1がstr_2よりも大きいことを示します。

Examples

```
1. #include <string.h>
   #include <stdio.h>

   main()
   {
       printf( "%d\n", strncmp("abcde", "abc", 3));
   }
```

この例をリンクして実行すると、2つの文字列の先頭の3文字が等しいため、0が返されます。

```
$ run tmp
0
```

```
2. #include <string.h>
   #include <stdio.h>

   main()
   {
       printf( "%d\n", strncmp("abcde", "abc", 4));
   }
```

この例をリンクして実行すると、2つの文字列の先頭の4文字は等しくないため、0よりも大きい値が返されます (第1の文字列の "d" が、第2の文字列の null 文字と等しくありません)。

```
$ run tmp
100
```

strncpy

sourceのmaxchar個以下の文字をdestにコピーします。

Format

```
#include <string.h>

char *strncpy (char *dest, const char *source, size_t maxchar);
```

関数バリエント

strncpy関数は、それぞれ 32 ビットと 64 ビットのポインタ・サイズで使用するための_strncpy32と_strncpy64という名前のバリエントを持っています。ポインタ・サイズ固有の関数の使用方法については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

Argument

dest
デスティネーション文字列へのポインタ。

source
ソース文字列へのポインタ。

maxchar
sourceからdestにコピーする文字数の上限。sourceの終端の null 文字は含みません。

Description

strncpy関数は、sourceのmaxchar個以下の文字をdestにコピーします。sourceの終端の null 文字は含みません。sourceが含んでいる文字がmaxchar個より少ない場合、destには null 文字がパディングされます。sourceが含んでいる文字がmaxchar個以上である場合には、可能な限り多くの文字がdestにコピーされます。strncpyを呼び出した後のdest引数は、終端に null 文字がない場合があることに注意してください。

strncpy

Return value

x

destのアドレス。

REF-757

strpbrk

文字列の中で、指定された文字のセットのうちのいずれかのオカレンスを検索します。

Format

```
#include <string.h>

char *strpbrk (const char *str, const char *charset);
```

関数バリエント

strpbrk関数は、それぞれ 32 ビットと 64 ビットのポインタ・サイズで使用するための _strpbrk32と _strpbrk64という名前のバリエントを持っています。ポインタ・サイズ固有の関数の使用方法については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

Argument

str
文字列へのポインタ。この文字列が null 文字列だった場合には、0 が返されます。

charset
この関数が探す文字のセットを含んでいる文字列へのポインタ。

Description

strpbrk関数は、文字列の中の文字をスキャンし、charsetに含まれている文字を検出した時点で停止し、文字列の中の文字セットに含まれている最初の文字のアドレスを返します。

Return value

x	文字列の中の、セットに含まれていた最初の文字のアドレス。
NULL	セットに含まれている文字がなかったことを示します。

strptime

文字列を、`tm`構造体に格納される日付および時刻値に変換します。変換は書式文字列によって制御されます。

Format

```
#include <time.h>

char *strptime (const char *buf, const char *format, struct tm *timeptr);
```

関数バリエント

`strptime`関数は、それぞれ 32 ビットと 64 ビットのポインタ・サイズで使用するための `_strptime32`と `_strptime64`という名前のバリエントを持っています。ポインタ・サイズ固有の関数の使用方法については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

Argument

`buf`

変換する文字列へのポインタ。

`format`

入力文字列の変換方法を定義する文字列へのポインタ。

`timeptr`

ローカル時刻構造体へのポインタ。`tm`構造体は`<time.h>`ヘッダ・ファイルに定義されています。

Description

`strptime`関数は、`buf`がポイントする文字列を、`timeptr`がポイントする構造体に格納される値に変換します。`format`がポイントする文字列は、変換の実行方法を定義します。

`strptime`関数は、`tm`構造体の中のフィールドのうち、対応する変換指定が書式に含まれているもののみを変更します。特に、`strptime`は`tm`構造体の`tm_isdst`メンバは決して設定しません。

書式文字列は 0 個以上のディレクティブから構成されます。ディレクティブは、以下のいずれかから構成されます。

- 1 つまたは複数の空白文字 (isspace関数の定義に従う)。このディレクティブが指定された場合、関数は空白文字でない最初の文字まで入力を読み込む。
- パーセント文字 (%) または空白文字以外の任意の文字。このディレクティブが指定された場合、関数は次の文字を読み込む。読み込まれた文字は、ディレクティブを構成する文字と同じでなくてはならない。文字が異なった場合、関数は実行に失敗する。
- 変換指定。変換指定は、入力文字列の中の文字が、tm構造体に格納される値としてどのように解釈されるかを定義する。変換指定は、パーセント文字 (%) の後に変換指定子を続けたものである。表 REF-9 は有効な変換指定を示している。

strptime関数は、プログラムの現在のロケールの LC_TIME カテゴリのフィールドから値を取得します。

注意

X/Open CAE Specification System Interfaces and Headers Issue 5 (通称 XPG5) に準拠するために、strptime関数は "%y"ディレクティブを、これまでのバージョンのHP C RTL とは異なる方法で処理します。

V6.4 およびそれ以降の C コンパイラでは、世紀内の 2 桁の年について、世紀が指定されていなかった場合、"%y"ディレクティブの値が次のように解釈されます。

- 69 ~ 99 は、20 世紀の年を表す (1969 年から 1999 年まで)
- 00 ~ 68 は、21 世紀の年を表す (2000 年から 2068 年まで)

これまでの (XPG4 準拠の) バージョンのHP C RTL では、strptimeは世紀が指定されない 2 桁の年を、20 世紀の年として解釈していました。

現在では、XPG-5 準拠のstrptimeが、HP C RTL のデフォルト・バージョンとなっています。

以前の XPG4 準拠のstrptime関数の動作を利用するには、以下のいずれかを指定します。

- DECC\$XPG4_STRPTIME 論理名を次のように定義する:

```
$ DEFINE DECC$XPG4_STRPTIME ENABLE
```

または

- 関数decc\$strptime_xpg4として、XPG4 のstrptimeを直接に呼び出す。

XPG5 バージョンのstrptimeに戻すには、DECC\$XPG4_STRPTIME 論理名に対して DEASSIGN を実行します:

```
$ DEASSIGN DECC$XPG4_STRPTIME
```

表 REF-9 strptime の変換指定

指定	置き換え
%a	曜日名。短縮名または完全な名前。
%A	%aと同じ。
%b	月の名前。短縮名または完全な名前。
%B	%bと同じ。
%c	ロケールの日付書式を使用した日付と時刻。
%Ec	ロケールの代替日付および時刻表現。
%C	10 進数 (00 ~ 99) としての世紀 (年を 100 で割り、整数に切り捨て)。先頭のゼロは許容される。
%EC	ロケールの代替表現での基本年 (期間) の名前。
%d	10 進数 (01 ~ 31) としての、月の中の日。先頭のゼロは許容される。
%Od	ロケールの代替数値シンボルを使用した、月の中の日。
%D	%m/%d/%yと同じ。
%e	%dと同じ。
%Oe	ロケールの代替数値シンボルを使用した、月の中の日付。
%h	%bと同じ。
%H	10 進数 (00 ~ 23) としての時間 (24 時間制)。先頭のゼロは許容される。
%OH	ロケールの代替数値シンボルを使用した時間 (24 時間制)。
%I	10 進数 (01 ~ 12) としての時間 (12 時間制)。先頭のゼロは許容される。
%OI	ロケールの代替数値シンボルを使用した時間 (12 時間制)。
%j	10 進数 (001 ~ 366) としての、年の中の日。
%m	10 進数 (01 ~ 12) としての月。先頭のゼロは許容される。
%Om	ロケールの代替数値シンボルを使用した月。
%M	10 進数 (00 ~ 59) としての分。先頭のゼロは許容される。
%OM	ロケールの代替数値シンボルを使用した分。
%n	任意の空白文字。
%p	ロケールの 12 時間制における AM/PM 指定。
%r	AM/PM 表記での時刻 (%I:%M:%S %p)。
%R	24 時間表記での時刻 (%H:%M)。
%S	10 進数 (00 ~ 61) としての秒。先頭のゼロは許容される。
%OS	ロケールの代替数値シンボルを使用した秒。
%t	任意の空白文字。
%T	時刻 (%H:%M:%S)。
%U	10 進数 (00 ~ 53) としての、その年の週 (最初の日曜日が 1 番目の週の最初の日と見なされる)。先頭のゼロは許容される。
%OU	ロケールの代替数値シンボルを使用した、その年の週 (週は日曜日から始まる)。
%w	10 進数としての曜日 (0 [日曜日] ~ 6)。先頭のゼロは許容される。
%Ow	ロケールの代替数値シンボルを使用した、数値としての曜日 (日曜日=0)。

(次ページに続く)

表 REF-9 (続き) strptime の変換指定

指定	置き換え
%W	10 進数 (00 ~ 53) としての、その年の週 (最初の月曜日が 1 番目の週の最初の日と見なされる)。
%OW	ロケールの代替数値シンボルを使用した、その年の週 (週は月曜日から始まる)。
%x	ロケールの適切な日付表現。
%Ex	ロケールの代替日付表現。
%EX	ロケールの代替時刻表現。
%X	ロケールの適切な時刻表現。
%y	10 進数 (00 ~ 99) としての、世紀を除いた年。
%Ey	ロケールの代替表現における基本年 (%EC) からのオフセット。
%Oy	ロケールの代替数値シンボルを使用した、世紀を除いた年。
%Y	10 進数としての、世紀を含む年。
%EY	ロケールの完全な代替年表現。
%%	リテラルの%文字。

Return value

x	最後に解析された文字の次の文字へのポインタ。
NULL	エラーが発生したことを示します。tm構造体の内容は未定義となります。

Example

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <locale.h>
#include <errno.h>

#define NUM_OF_DATES 7
#define BUF_SIZE 256

/* This program takes a number of date and time strings and
/* converts them into tm structs using strptime(). These tm
/* structs are then passed to strftime() which will reverse the
/* process. The resulting strings are then compared with the
/* originals and if a difference is found then an error is
/* displayed.*/
```

```

main()
{
    int count,
        i;
    char buffer[BUF_SIZE];
    char *ret_val;
    struct tm time_struct;
    char dates[NUM_OF_DATES][BUF_SIZE] =
    {
        "Thursday 01 January 1970 00:08:20",
        "Tuesday 29 February 1972 08:26:40",
        "Tuesday 31 December 1991 23:59:59",
        "Wednesday 01 January 1992 00:00:00",
        "Sunday 03 May 1992 13:33:20",
        "Monday 04 May 1992 17:20:00",
        "Friday 15 May 1992 03:20:00"};

    for (i = 0; i < NUM_OF_DATES; i++) {
        /* Convert to a tm structure */
        ret_val =.strptime(dates[i], "%A %d %B %Y %T", &time_struct);

        /* Check the return value */
        if (ret_val == (char *) NULL) {
            perror("strptime");
            exit(EXIT_FAILURE);
        }

        /* Convert the time structure back to a formatted string */
        count = strftime(buffer, BUF_SIZE, "%A %d %B %Y %T", &time_struct);

        /* Check the return value */
        if (count == 0) {
            perror("strftime");
            exit(EXIT_FAILURE);
        }

        /* Check the result */
        if (strcmp(buffer, dates[i]) != 0) {
            printf("Error: Converted string differs from the original\n");
        }
        else {
            printf("Successfully converted <%s>\n", dates[i]);
        }
    }
}

```

この例のプログラムを実行すると、次の出力が生成されます。

```

Successfully converted <Thursday 01 January 1970 00:08:20>
Successfully converted <Tuesday 29 February 1972 08:26:40>
Successfully converted <Tuesday 31 December 1991 23:59:59>
Successfully converted <Wednesday 01 January 1992 00:00:00>
Successfully converted <Sunday 03 May 1992 13:33:20>
Successfully converted <Monday 04 May 1992 17:20:00>
Successfully converted <Friday 15 May 1992 03:20:00>

```

strchr

`null` で終了する文字列の中の、特定の文字の最後のオカレンスのアドレスを返します。

Format

```
#include <string.h>

char *strchr (const char *str, int character);
```

関数バリエント

`strchr`関数は、それぞれ 32 ビットと 64 ビットのポインタ・サイズで使用するための `_strchr32`と `_strchr64`という名前のバリエントを持っています。ポインタ・サイズ固有の関数の使用方法については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

Argument

`str`
`null` で終了する文字列へのポインタ。

`character`
`int`型のオブジェクト。

Description

この関数は、`null` で終了する文字列の中を先頭から調べて、指定した文字が最後に見つかったアドレスを返します。終端の `null` 文字は、文字列の一部と見なされます。

これとは対照的に、`strchr`では、`null` で終了する文字列の中を先頭から調べて、指定した文字が最初に見つかったアドレスを返します。

Return value

x	指定された文字の最後のオカレンスのアドレス。
---	------------------------

NULL

文字が文字列に含まれていないことを示します。

strsep

文字列を分割します。

Format

```
#include <string.h>

char *strsep (char **stringp, char *delim);
```

関数バリエント

strsep関数は、それぞれ 32 ビットと 64 ビットのポインタ・サイズで使用するための `_strsep32` と `_strsep64` という名前のバリエントを持っています。ポインタ・サイズ固有の関数の使用方法については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

Argument

stringp

文字列へのポインタへのポインタ。

delim

区切り文字として使用する文字を含んでいる文字列へのポインタ。

Description

strsep関数は、stringpの中で、delimに含まれる任意の文字 (または終端の `'\0'` 文字) の最初のオカレンスを探し、それを `'\0'` に置き換えます。区切り文字 (または文字列の終端に達した場合は NULL) の次の文字の位置がstringp引数に格納されます。stringp引数の元の値が返されます。

区切り文字が 2 つ隣接していたときに生じる「空」のフィールドは、stringp引数に返されたポインタが参照している位置を `'\0'` と比較することで検出できます。

stringp引数が初期状態で NULL だった場合、strsepは NULL を返します。

Return value

x	stringpがポイントする文字列のアドレス。
NULL	stringpが NULLであることを示します。

Example

次の例は、strsepを使用して、空白で区切られたトークンを含んでいる文字列を解析し、引数のベクトルに格納します。

```
char **ap, **argv[10], *inputstring;
for (ap = argv; (*ap = strsep(&inputstring, " \t")) != NULL;)
    if (**ap != '\0')
        ++ap;
```

strspn

文字列の、文字のセットに含まれている文字のみから構成される接頭辞の長さを返します。

Format

```
#include <string.h>

size_t strspn (const char *str, const char *charset);
```

Argument

str

文字列へのポインタ。この文字列が null 文字列だった場合には、0 が返されます。

charset

関数が検索の対象とする文字を含んでいる文字列へのポインタ。

Description

strspn関数は、文字列の中の文字をスキャンし、charsetに含まれていない文字を検出した時点で停止し、charsetに含まれている文字から構成される文字列の最初のセグメントの長さを返します。

Return value

x

セグメントの長さ。

strstr

s1がポイントしている文字列の中での、s2がポイントしている文字列に含まれる文字のシーケンスの最初のオカレンスを探します。

Format

```
#include <string.h>

char *strstr (const char *s1, const char *s2);
```

関数バリエーション

strstr関数は、それぞれ 32 ビットと 64 ビットのポインタ・サイズで使用するための _strstr32 と _strstr64 という名前のバリエーションを持っています。ポインタ・サイズ固有の関数の使用方法については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

Argument

s1, s2
文字列へのポインタ。

Return value

ポインタ	発見された文字列へのポインタ。
NULL	文字列が発見されなかったことを示します。

Example

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

main()
{
    static char lookin[]="that this is a test was at the end";
```

```

        putchar('\n');
        printf("String: %s\n", &lookin[0] );
        putchar('\n');
        printf("Addr: %s\n", &lookin[0] );
        printf("this: %s\n", strstr( &lookin[0] , "this" ) );
        printf("that: %s\n", strstr( &lookin[0] , "that" ) );
        printf("NULL: %s\n", strstr( &lookin[0], "" ) );
        printf("was: %s\n", strstr( &lookin[0], "was" ) );
        printf("at: %s\n", strstr( &lookin[0], "at" ) );
        printf("the end: %s\n", strstr( &lookin[0], "the end" ) );
        putchar('\n');

        exit(0);
    }

```

この例は、次の結果を生成します。

```

$ RUN STRSTR_EXAMPLE
String: that this is a test was at the end
Addr: that this is a test was at the end
this: this is a test was at the end
that: that this is a test was at the end
NULL: that this is a test was at the end
was: was at the end
at: at this is a test was at the end
the end: the end
$

```

strtod

指定された文字列を倍精度の数値に変換します。

Format

```
#include <stdlib.h>

double strtod (const char *nptr, char **endptr);
```

関数バリエーション

strtod関数は、それぞれ 32 ビットと 64 ビットのポインタ・サイズで使用するための `_strtod32` と `_strtod64` という名前のバリエーションを持っています。ポインタ・サイズ固有の関数の使用方法については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

Argument

`nptr`

倍精度の数値に変換する文字列へのポインタ。

`endptr`

関数が、スキャンを終了させた最初の認識不可能な文字のアドレスを格納できるオブジェクトのアドレス。`endptr` が NULL ポインタである場合、最初の認識不可能な文字のアドレスは保存されません。

Description

strtod関数は、オプションとして空白文字 (`isspace` の定義に従う) のシーケンスを、さらにオプションのプラスまたはマイナス記号を、さらにオプションとして基数文字を含んだ数字のシーケンスを、さらにオプションの文字 (`e` または `E`) を、最後にオプションの符号付きの整数を認識します。最初の認識不可能な文字が現れた時点で、変換は終了します。

文字列は、浮動小数点定数を解釈するとき使用されるのと同じ規則によって解釈されます。

基数文字は、プログラムの現在のロケール (カテゴリ `LC_NUMERIC`) によって定義されます。

この関数は、変換後の値を返します。strtodでは、オーバーフローは次のように処理されます。

- 正しい値がオーバーフローを引き起こす場合には、(値の符号に従ってプラスまたはマイナス記号が付いた) HUGE_VAL が返され、errnoは ERANGE に設定される。
- 正しい値がアンダフローを引き起こす場合には、0 が返され、errnoは EINVAL に設定される。

文字列が認識不可能な文字から始まっている場合には、変換は実行されない。*entptrはnptrに設定され、0の値が返され、errnoはEINVALに設定される。

Return value

x	変換された文字列。
0	変換が実行できなかったことを示します。errnoは以下のいずれかに設定されます。 <ul style="list-style-type: none">• EINVAL - 変換は実行できなかった。• ERANGE - 値はアンダフローを引き起こす。• ENOMEM - 内部変換バッファ用のメモリが足りなかった。
±HUGE_VAL	オーバーフローが発生しました。errnoは ERANGE に設定されます。

strtok, strtok_r

文字列をトークンに分解します。

Format

```
#include <string.h>

char *strtok (char *s1, const char *s2);
char *strtok_r (char *s, const char *sep, char **lasts);
```

関数バリエント

strtok関数は、それぞれ 32 ビットと 64 ビットのポインタ・サイズで使用するための `_strtok32` と `_strtok64` という名前のバリエントを持っています。同様に、`strtok_r` 関数には、`_strtok_r32` と `_strtok_r64` というバリエントがあります。ポインタ・サイズ固有の関数の使用方法については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

Argument

s1

最初の呼び出しでは、ゼロ個以上のテキスト・トークンを含んでいる文字列へのポインタ。その文字列に対するそれ以降のすべての呼び出しでは、NULL ポインタ。

s2

1 つまたは複数の文字から構成される区切り文字列へのポインタ。区切り文字列は呼び出しごとに異なっても構いません。

s

区切り文字列sepの 1 個以上の文字で区切られた、ゼロ個以上のトークンのシーケンスである、null で終わる文字列。

sep

区切り文字の、null で終わる文字列。この区切り文字列は、呼び出しごとに異なっても構いません。

lasts

strtok_rが同じ文字列のスキャンを続けるために必要とする情報を格納するためのユーザ提供ポインタを指すポインタ。

Description

strtok関数は、指定された文字列内で、テキスト・トークンを見つけます。テキスト・トークンは、ユーザが指定した区切り文字列内の1個以上の文字で区切られます。この関数は、呼び出し間をまたいで文字列内でのトークンの位置を把握しています。そして、呼び出しが連続して行われると、以前の呼び出して識別されたトークンの次のテキスト・トークンを識別して、文字列全体を処理します。

s1の中のトークンは、区切り文字列s2に含まれていない最初の文字から始まり、文字列の終端か、区切り文字の手前で終了します。

strtok関数の最初の呼び出しでは、最初のトークンに含まれている最初の文字へのポインタが返され、s1の中の返されたトークンの直後に null 文字が書き込まれます。それ以降の(第1引数の値を NULL のままにした)個々の呼び出しでは、s1がもともポイントしていた文字列の中の、その次のトークンへのポインタが返されます。文字列にトークンが残っていない場合、strtok関数は NULL ポインタを返します(これは、文字列が空であるか、区切り文字のみを含んでいる場合には、strtokの最初の呼び出しでも起こります)。

strtokはトークンを区切るためにs1に null 文字を挿入するので、s1はconstオブジェクトであってはなりません。

strtok_r関数は、strtokのリエントラント・バージョンです。strtok_r関数は、nullで終了する文字列sを、区切り文字列sep内の1個以上の文字で区切られた、ゼロ個以上のテキスト・トークンのシーケンスと見なします。lasts引数は、同じ文字列のスキャンを続けるためにstrtok_rが必要とする情報を格納するためのユーザ提供ポインタを指します。

strtok_rの1回目の呼び出しでは、sはnullで終わる文字列を指し、sepは区切り文字の、nullで終わる文字列を指し、lastsが指す値は無視されます。strtok_r関数は、1番目のトークンの1番目の文字へのポインタを返し、s内の、返したトークンの直後の位置に null 文字を書き込み、lastsが指しているポインタをアップデートします。

以降の呼び出しでは、呼び出すたびに文字列s上を移動するように、sは NULL ポインタとし、lastsは以前の呼び出しのままとします。残りのトークンがなくなるまで、次々とトークンが返されます。区切り文字列sepは、呼び出しごとに異なっても構いません。s内にトークンが残っていない場合は、NULL ポインタが返されます。

Return value

x	文字列内の、検出されたトークンの1文字目へのポインタです。
---	-------------------------------

NULL

文字列内にトークンが残っていないことを示します。

Examples

```
1. #include <stdio.h>
   #include <string.h>

   main()
   {
       static char str[] = "...ab..cd,,ef.hi";

       printf("%s\n", strtok(str, "."));
       printf("%s\n", strtok(NULL, ","));
       printf("%s\n", strtok(NULL, "."));
       printf("%s\n", strtok(NULL, "."));
   }
```

この例のプログラムを実行すると、次の結果が出力されます。

```
$ RUN STRTOK_EXAMPLE1
|ab|
|.cd|
|ef|
|hi|
$
```

```
2. #include <stdio.h>
   #include <string.h>

   main()
   {
       char *ptr,
           string[30];

       /* The first character not in the string "-" is "A". The */
       /* token ends at "C."                                     */

       strcpy(string, "ABC");
       ptr = strtok(string, "-");
       printf("%s\n", ptr);

       /* Returns NULL because no characters not in separator */
       /* string "-" were found (i.e. only separator characters */
       /* were found)                                           */

       strcpy(string, "-");
       ptr = strtok(string, "-");
       if (ptr == NULL)
           printf("ptr is NULL\n");
   }
```

この例のプログラムを実行すると、次の結果が出力されます。

strtok, strtok_r

```
$ RUN STRTOK_EXAMPLE2
|abc|
ptr is NULL
$
```

strtol

ASCII 文字の文字列を、適切な数値に変換します。

Format

```
#include <stdlib.h>

long int strtol (const char *nptr, char **endptr, int base);
```

関数バリエーション

strtol関数は、それぞれ 32 ビットと 64 ビットのポインタ・サイズで使用するための `_strtol32` と `_strtol64` という名前のバリエーションを持っています。ポインタ・サイズ固有の関数の使用方法については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

Argument

`nptr`

long に変換する文字列へのポインタ。

`endptr`

関数が、変換プロセスの中で検出した最初の認識不可能な文字 (つまり、変換された文字列の中の最後の文字の直後の文字) へのポインタを格納できるオブジェクトのアドレス。endptr が NULL ポインタである場合、最初の認識不可能な文字のアドレスは保存されません。

`base`

変換の底として使用する 2 ~ 36 の値。

Description

strtol関数は、底の値に応じて、さまざまな形式の文字列を認識します。この関数は、指定された文字列の中の先頭の空白文字 (<ctype.h>の `isspace` の定義に従う) はすべて無視します。オプションのプラスまたはマイナス記号を認識した後に、底の値に応じて整数を表現する数字または文字のシーケンスを認識します。最初の認識不可能な文字が、変換を終了させます。

オプションの符号の後の先頭のゼロは無視され、底が 16 の場合には 0x と 0X も無視されます。

base が 0 の場合、文字のシーケンスは、整数を解釈するのに使用されるのと同じ規則によって解釈されます。オプションの符号の後の先頭の 0 は 8 進変換を、先頭の 0x または 0X は 16 進変換を、その他の組み合わせは 10 進変換を示します。

代入の後、または明示的なキャストには (算術例外が生じる場合でも) long から int への切り捨てが起こることがあります。関数呼び出し `atol(str)` は、`strtol(str, (char**)NULL, 10)` と等価です。

Return value

x	変換された値。
LONG_MAX または LONG_MIN	変換された値がオーバーフローを引き起こすことを示します。
0	文字列が認識不可能な文字から始まっている、または base の値が無効であることを示します。文字列が認識不可能な文字から始まっている場合、*endptr は nptr に設定されます。

strtoq, strtoll (Alpha, I64)

ASCII 文字の文字列を、適切な数値に変換します。strtollはstrtoqの同義語です。

Format

```
#include <stdlib.h>

__int64 strtoq (const char *nptr, char **endptr, int base);
__int64 strtoll (const char *nptr, char **endptr, int base);
```

関数バリエーション

これらの関数は、それぞれ 32 ビットと 64 ビットのポインタ・サイズで使用するための `_strtoq32`, `_strtoll32` と、`_strtoq64`, `_strtoll64` という名前のバリエーションを持っています。ポインタ・サイズ固有の関数の使用方法については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

Argument

`nptr`

`__int64` に変換する文字列へのポインタ。

`endptr`

関数が、変換プロセスの中で検出した最初の認識不可能な文字 (つまり、変換された文字列の中の最後の文字の直後の文字) へのポインタを格納できるオブジェクトのアドレス。`endptr` が NULL ポインタである場合、最初の認識不可能な文字のアドレスは保存されません。

`base`

変換の底として使用する 2 ~ 36 の値。

Description

`strtoq` 関数と `strtoll` 関数は、底の値に応じて、さまざまな形式の文字列を認識します。指定された文字列内の、先頭の空白文字 (`<ctype.h>` の `isspace` で定義されています) は、無視されます。これらの関数は、オプションのプラスまたはマイナス記号を認識した後に、底の値に従って整数を表現する数字または文字のシーケンスを認識します。最初の認識不可能な文字が、変換を終了させます。

オプションの符号の後の先頭のゼロは無視され、底が 16 の場合には 0x と 0X も無視されます。

base が 0 の場合、文字のシーケンスは、整数を解釈するのに使用されるのと同じ規則によって解釈されます。オプションの符号の後の先頭の 0 は 8 進変換を、先頭の 0x または 0X は 16 進変換を、その他の組み合わせは 10 進変換を示します。

関数呼び出し `atog (str)` は `strtog (str, (char**)NULL, 10)` と等価です。

Return value

<code>x</code>	変換された値。
<code>__INT64_MAX</code> また は <code>__INT64_MIN</code>	変換された値がオーバーフローを引き起こすことを示します。
<code>0</code>	文字列が認識不可能な文字から始まっている、または base の値が無効であることを示します。文字列が認識不可能な文字から始まっている場合、 <code>*endptr</code> は <code>nptr</code> に設定されます。

strtoul

nptrがポイントする文字列の最初の部分を、unsigned long整数に変換します。

Format

```
#include <stdlib.h>

unsigned long int strtoul (const char *nptr, char **endptr, int base);
```

関数バリエーション

strtoul関数は、それぞれ 32 ビットと 64 ビットのポインタ・サイズで使用するための _strtoul32 と _strtoul64 という名前のバリエーションを持っています。ポインタ・サイズ固有の関数の使用方法については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

Argument

nptr

unsigned long に変換する文字列へのポインタ。

endptr

関数が、変換プロセスの中で検出した最初の認識不可能な文字 (つまり、変換された文字列の中の最後の文字の直後の文字) へのポインタへのポインタを格納できるオブジェクトのアドレス。endptr が NULL ポインタである場合、最初の認識不可能な文字のアドレスは保存されません。

base

変換の底として使用する 2 ~ 36 の値。オプションの符号の後の先頭のゼロは無視され、底が 16 の場合には 0x と 0X も無視されます。

base が 0 の場合、文字のシーケンスは、整数を解釈するのに使用されるのと同じ規則によって解釈されます。オプションの符号の後の先頭の 0 は 8 進変換を、先頭の 0x または 0X は 16 進変換を、その他の組み合わせは 10 進変換を示します。

Return value

x	変換された値。
0	文字列が認識不可能な文字から始まっている、またはbaseの値が無効であることを示します。文字列が認識不可能な文字から始まっている場合、*endptrはnptrに設定されます。
ULONG_MAX	変換された値がオーバーフローを引き起こすことを示します。

strtouq, strtoull (Alpha, I64)

nptrがポイントする文字列の最初の部分を、unsigned __int64整数に変換します。strtoullはstrtouqの同義語です。

Format

```
#include <stdlib.h>

unsigned __int64 strtouq (const char *nptr, char **endptr, int base);
unsigned __int64 strtoull (const char *nptr, char **endptr, int base);
```

関数バリエーション

これらの関数は、それぞれ 32 ビットと 64 ビットのポインタ・サイズで使用するのための _strtouq32, _strtoull32 と、_strtouq64, _strtoull64 という名前のバリエーションを持っています。ポインタ・サイズ固有の関数の使用方法については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

Argument

nptr
unsigned __int64 に変換する文字列へのポインタ。

endptr
関数が、変換プロセスの中で検出した最初の認識不可能な文字 (つまり、変換された文字列の中の最後の文字の直後の文字) へのポインタへのポインタを格納できるオブジェクトのアドレス。endptr が NULL ポインタである場合、最初の認識不可能な文字のアドレスは保存されません。

base
変換の底として使用する 2 ~ 36 の値。オプションの符号の後の先頭のゼロは無視され、底が 16 の場合には 0x と 0X も無視されます。

base が 0 の場合、文字のシーケンスは、整数を解釈するのに使用されるのと同じ規則によって解釈されます。オプションの符号の後の先頭の 0 は 8 進変換を、先頭の 0x または 0X は 16 進変換を、その他の組み合わせは 10 進変換を示します。

Return value

x	変換された値。
0	文字列が認識不可能な文字から始まっている, またはbaseの値が無効であることを示します。文字列が認識不可能な文字から始まっている場合, *endptrはnptrに設定されます。
__UINT64_MAX	変換された値がオーバーフローを引き起こすことを示します。

strxfrm

変更後の文字列をstrcmp関数に渡したときに、未変更の文字列をstrcoll関数に渡したときと同じ結果が得られるように、文字列を変更します。

Format

```
#include <string.h>

size_t strxfrm (char *s1, const char *s2, size_t maxchar);
```

Argument

s1, s2
文字列へのポインタ。

maxchar
s1に格納するバイト数の最大値 (終端の null を含みます)。

Description

strxfrm関数は、s2がポイントしている文字列を変換し、結果として得られた文字列を、s1がポイントする配列に格納します。s1がポイントする配列には、終端の null を含めて、maxcharバイト以下の文字が格納されます。

maxcharの値が、変換後の文字列 (終端の null を含む) を格納するのに必要なサイズよりも小さかった場合、s1がポイントする配列の内容は不定となります。この場合、関数は変換後の文字列のサイズを返します。

maxcharが 0 の場合、s1は NULL ポインタであってよく、関数は変換を行う前に、s1配列の必要なサイズを返します。

文字列比較関数のstrcollとstrcmpは、比較する 2 つの文字列を与えられたときに、異なる結果を生成することがあります。これは、strcmpが文字列の中の文字のコード・ポイント値を単純に比較するのに対し、strcollが比較のためにロケール情報を使用することが原因です。ロケールによっては、strcollによるの比較はマルチパスの操作になり、strcmpよりも速度が遅くなります。

strxfrm関数の目的は、2 つの変換後の文字列をstrcmp関数に渡したときの結果が、2 つの元の文字列をstrcoll関数に渡したときと同じになるように、文字列を変換することです。strxfrm関数は、同じ文字列に対してstrcollを使って多数の比較を行わな

くてはならないアプリケーションで有用です。この場合には、(ロケールによっては) strxfrmを使って文字列の変換を行った後に、strcmpで比較を行った方が効率的なことがあります。

Return value

x	<p>s1がポイントする、結果として得られる文字列の長さ (終端の null 文字は含みません)。</p> <p>エラー条件のための戻り値は予約されていません。ただし、関数はerrnoを EINVAL に設定することがあります。s2がポイントする文字列は、照合シーケンスのドメイン外の文字を含んでいます。</p>
---	--

Example

```

/* This program verifies that two transformed strings when      */
/* passed through strxfrm and then compared, provide the same   */
/* result as if passed through strcoll without any              */
/* transformation.                                             */
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>

#define BUFF_SIZE 256

main()
{
    char string1[BUFF_SIZE];
    char string2[BUFF_SIZE];
    int errno;
    int coll_result;
    int strcmp_result;
    size_t strxfrm_result1;
    size_t strxfrm_result2;

    /* setlocale to French locale */
    if (setlocale(LC_ALL, "fr_FR.ISO8859-1") == NULL) {
        perror("setlocale");
        exit(EXIT_FAILURE);
    }

    /* collate string 1 and string 2 and store the result */
    errno = 0;
    coll_result = strcoll("<a'>bcd", "abcz");
    if (errno) {
        perror("strcoll");
        exit(EXIT_FAILURE);
    }
}

```

```

else {
    /* Transform the strings (using strxfrm) into string1 */
    /* and string2 */
    strxfrm_result1 = strxfrm(string1, "<a'>bcd", BUFF_SIZE);
    if (strxfrm_result1 == ((size_t) - 1)) {
        perror("strxfrm");
        exit(EXIT_FAILURE);
    }
    else if (strxfrm_result1 > BUFF_SIZE) {
        perror("\n** String is too long **\n");
        exit(EXIT_FAILURE);
    }
    else {
        strxfrm_result2 = strxfrm(string2, "abcz", BUFF_SIZE);
        if (strxfrm_result2 == ((size_t) - 1)) {
            perror("strxfrm");
            exit(EXIT_FAILURE);
        }
        else if (strxfrm_result2 > BUFF_SIZE) {
            perror("\n** String is too long **\n");
            exit(EXIT_FAILURE);
        }
        /* Compare the two transformed strings and verify */
        /* that the result is the same as the result from */
        /* strcoll on the original strings */
        else {
            strcmp_result = strcmp(string1, string2);
            if (strcmp_result == 0 && (coll_result == 0)) {
                printf("\nReturn value from strcoll() and "
                    "return value from strcmp() are both zero.");
                printf("\nThe program was successful\n\n");
            }
            else if ((strcmp_result < 0) && (coll_result < 0)) {
                printf("\nReturn value from strcoll() and "
                    "return value from strcmp() are less than zero.");
                printf("\nThe program successful\n\n");
            }
            else if ((strcmp_result > 0) && (coll_result > 0)) {
                printf("\nReturn value from strcoll() and "
                    "return value from strcmp() are greater than zero.");
                printf("\nThe program was successful\n\n");
            }
            else {
                printf("** Error **\n");
                printf("\nReturn values are not of the same type");
            }
        }
    }
}
}
}

```

この例のプログラムを実行すると、次の結果が出力されます。

```
Return value from strcoll() and return value
      from strcmp() are less than zero.
The program was successful
```

subwin

ターミナル・スクリーン上の座標 (begin_y,begin_x) に , numlines行 , numcolsカラムの新しいサブウィンドウを作成します。

Format

```
#include <curses.h>

WINDOW *subwin (WINDOW *win, int numlines, int numcols, int begin_y, int begin_x);
```

Argument

win

親ウィンドウへのポインタ。

numlines

サブウィンドウの中の行の数。numlinesが 0 ならば , 関数はその値を `LINES - begin_y` に設定します。 `LINES x COLS` の大きさのサブウィンドウを得るには , 次の形式を使用します。

```
subwin (win, 0, 0, 0, 0)
```

numcols

サブウィンドウの中のカラムの数。numcolsが 0 ならば , 関数はその値を `COLS - begin_x` に設定します。 `LINES x COLS` の大きさのサブウィンドウを得るには , 次の形式を使用します。

```
subwin (win, 0, 0, 0, 0)
```

begin_y

サブウィンドウを作成するウィンドウ座標。

begin_x

サブウィンドウを作成するウィンドウ座標。

Description

サブウィンドウを作成するとき , begin_yとbegin_xはターミナル・スクリーン全体を基準としています。 numlinesまたはnumcolsが 0 だった場合 , subwin関数はサイズをそれぞれ (`LINES - begin_y`) または (`COLS - begin_x`) に設定します。

winがポイントするウィンドウは、サブウィンドウの領域全体が収まるだけの大きさでなくてはなりません。どちらかのウィンドウ内で、サブウィンドウの座標内に加えられた変更は、両方のウィンドウに反映されます。

Return value

ウィンドウ・ポインタ	新しく作成されたサブウィンドウに対応するウィンドウ構造体のインスタンスへのポインタ。
ERR	エラーを示します。

swab

バイトをスワップします。

Format

```
#include <unistd.h>

void swab (const void *src, void *dest, ssize_t nbytes);
```

Argument

src

コピーする文字列の位置へのポインタ。

dest

結果のコピー先へのポインタ。

nbytes

コピーするバイト数。この引数は偶数値にしてください。奇数値だった場合、swab関数は代わりにnbytes - 1を使用します。

Description

swab関数は、nbytesによって指定されたバイト数を、srcがポイントする位置からdestがポイントする配列にコピーします。その後、関数は隣接するバイトを交換します。コピーが互いにオーバーラップするオブジェクトの間で行われた場合の結果は未定義です。

swprintf

ワイド文字書式文字列の制御下で、ワイド文字の配列に出力を書き込みます。

Format

```
#include <wchar.h>

int swprintf (wchar_t *s, size_t n, const wchar_t *format, ... );
```

Argument

s

結果として得られるワイド文字シーケンスへのポインタ。

n

sがポイントする配列に書き込むことができるワイド文字の数の最大値。終端の null ワイド文字を含みます。

format

書式指定を含んだワイド文字列へのポインタ。書式指定と変換指定、およびその対応する引数については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 2 章を参照してください。

...

結果として得られる型が、書式指定で与えられた変換指定に対応しているオプションの式。

変換指定が与えられなかった場合、出力ソースは省略することができます。そうでない場合は、関数呼び出しは変換指定と同じ数の出力ソースを持っていなくてはならず、変換指定は出力ソースの型と一致していなくてはなりません。

変換指定は、左から右の順序で出力ソースと照合されます。余分な出力ポインタがある場合には、無視されます。

Description

swprintf関数は、第 1 引数がストリームではなくワイド文字の配列を指定しているという点を除けば、fwprintf関数と等価です。

書き込まれるワイド文字の数は、終端の null ワイド文字を含めて、n個以下です。終端の null ワイド文字は (nが 0 でなければ) つねに追加されます。

fwprintfも参照してください。

Return value

x	書き込まれたワイド文字の数。終端の null ワイド文字は含みません。
負の値	エラーを示します。n個以上のワイド文字の書き込みが要求されたか、変換エラーが発生しました。後者の場合、errnoは EILSEQ に設定されます。

swscanf

ワイド文字書式文字列の制御下で、ワイド文字列からの入力を読み込みます。

Format

```
#include <wchar.h>

int swscanf (const wchar_t *s, const wchar_t *format, ... );
```

Argument

s

入力元のワイド文字列へのポインタ。

format

書式指定を含んだワイド文字列へのポインタ。書式指定と変換指定、およびその対応する引数については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第2章を参照してください。

...

結果として得られる型が、書式指定で与えられた変換指定に対応しているオプションの式。

変換指定が与えられなかった場合、入力ポインタは省略することができます。そうでない場合は、関数呼び出しは変換指定と同じ数の入力ポインタを持っていなくてはならず、変換指定は入力ポインタの型と一致していなくてはなりません。

変換指定は、左から右の順序で入力ソースと照合されます。余分な入力ポインタがある場合には、無視されます。

Description

swscanf関数は、第1引数がストリームではなくワイド文字の配列を指定しているという点を除けば、fwscanf関数と等価です。ワイド文字列の終端に達するのは、fwscanf関数でEOFを検出することに対応します。

fwscanfも参照してください。

Return value

x	代入が行われた入力項目の数。これは用意された項目の数よりも少なくなることがあり、照合が早い段階で失敗した場合には 0 になることもあります。
EOF	エラーを示します。変換が行われる前に入力エラーが発生しました。

symlink (Alpha, I64)

指定した内容を含むシンボリック・リンクを作成します。

Format

```
#include <unistd.h>

int symlink (const char *link_contents, const char *link_name);
```

Argument

link_contents

シンボリック・リンク・ファイルの内容。シンボリック・リンクが指すことになるパス名 (テキスト文字列) を、ポインタで指定します。

link_name

シンボリック・リンク・ファイルの名前 (テキスト文字列)。

Description

シンボリック・リンクは、他のファイルを指す特殊なファイルです。シンボリック・リンクはディレクトリ・エントリになっていて、一部のサービスがアクセスしたときに POSIX パス名として解釈されるテキスト文字列を、ファイル名と関連付けます。OpenVMS システムにおけるシンボリック・リンクは、編成が SPECIAL でタイプが SYMBOLIC_LINK のファイルとして実装されています。

symlink関数は、指定した内容 (link_contents) を含んだシンボリック・リンク (link_name) を作成します。リンクを作成するときは、シンボリック・リンクの内容はいつさい解釈されません。

readlink , unlink , realpath , lchown , および lstat も参照してください。

Return value

0 成功したことを示します。

-1

エラーが発生したことを示します。errnoに、エラーを示す次のいずれかの値が設定されます。

- EACCES — シンボリック・リンクを作成しようとしているディレクトリで書き込み許可が認められなかったか、link_nameのパス接頭辞部分に対して、検索許可が認められませんでした。
- EEXIST — link_name引数で指定した名前のファイルまたはシンボリック・リンクがすでに存在しています。
- ENAMETOOLONG — (1) link_name引数の長さが PATH_MAX を超えているか、(2) パス名に長さが NAME_MAX を超えるコンポーネントがあるか、(3) link_contents引数の長さが SYMLINK_MAX を超えています。
- ENOSPC — (1) ファイル・システムに十分な空き領域がなくて、新しいシンボリック・リンクの置き先となるディレクトリが拡張できなかったか、(2) ファイル・システムに十分な空き領域がなくて、新しいシンボリック・リンクを作成できなかったか、(3) ファイル・システムにファイルへ割り当てるリソースがありませんでした。
- creat, fsync, lstat, またはwriteから返された上記以外のerrno値。

sysconf

構成可能なシステム変数を取得します。

Format

```
#include <unistd.h>

long int sysconf (int name);
```

Argument

name
照会するシステム変数を指定します。

Description

sysconf関数は、構成可能なシステム制限の現在の値と、オプション機能がサポートされているかどうかを調べるための手段を提供します。

name引数にはシンボリック定数を指定します。sysconfは、それに対応するシステム変数の値を返します。

- シンボリック定数は<unistd.h>ヘッダ・ファイルに定義されている。
- システム変数は<limits.h>および<unistd.h>ヘッダ・ファイルに定義されている。

表 REF-10 は、sysconf関数から返されるシステム変数と、name値として指定できるシンボリック定数を示しています。

表 REF-10 sysconf 引数と戻り値

返されるシステム変数	nameのシンボリック定数	意味
ISO POSIX-1		
ARG_MAX	_SC_ARG_MAX	exec関数の 1 つに対して指定される引数の長さ (環境データを含む) の最大値 (バイト数)。
CHILD_MAX	_SC_CHILD_MAX	個々の実ユーザ ID の同時プロセスの数の最大値。

(次ページに続く)

表 REF-10 (続き) sysconf 引数と戻り値

返されるシステム変数	nameのシンボリック定数	意味
ISO POSIX-1		
CLK_TCK	_SC_CLK_TCK	1 秒当たりのクロック・ティックの数。CLK_TCK の値は変動する。CLK_TCK がコンパイル時定数であると仮定してはならない。
NGROUPS_MAX	_SC_NGROUPS_MAX	個々のプロセスの同時補助グループ ID の数の最大値。
OPEN_MAX	_SC_OPEN_MAX	1 つのプロセスが同時にオープンできるファイルの数の最大値。
STREAM_MAX	_SC_STREAM_MAX	1 つのプロセスが同時にオープンできるストリームの数。
TZNAME_MAX	_SC_TZNAME_MAX	タイム・ゾーンの名前としてサポートされているバイト数の最大値 (TZ環境変数の長さではない)。
_POSIX_JOB_CONTROL	_SC_JOB_CONTROL	この変数は、システムがジョブ・コントロールをサポートしている場合には値 1 を持ち、そうでない場合には-1が返される。
_POSIX_SAVED_IDS	_SC_SAVED_IDS	この変数は、個々のプロセスが保存済みセット・ユーザ ID と保存済みセット・グループ ID を持つ場合は値 1 を持ち、そうでない場合には-1が返される。
_POSIX_VERSION	_SC_VERSION	システムがサポートしている最新バージョンの POSIX-1 標準の承認日。日付は 6 桁の数値で、最初の 4 桁が年を、最後の 2 桁が月を表す。 POSIX_VERSION が定義されていない場合には、-1が返される。 IEEE Standards Board は、定期的に POSIX-1 標準の新しいバージョンを承認しており、承認日はこれらの異なるバージョンを区別するために使用される。
ISO POSIX-2		
BC_BASE_MAX	_SC_BC_BASE_MAX	bcコマンドで、obase変数に使用できる値の最大値。
BC_DIM_MAX	_SC_BC_DIM_MAX	bcコマンドで、配列が含むことができる要素の数の最大値。
BC_SCALE_MAX	_SC_BC_SCALE_MAX	bcコマンドで、スケール変数に使用できる値の最大値。
BC_STRING_MAX	_SC_BC_STRING_MAX	bcコマンドが受け付ける文字列定数の長さの最大値。
COLL_WEIGHTS_MAX	_SC_COLL_WEIGHTS_MAX	ロケール定義ファイルのLC_COLLATEロケール依存情報のエントリに割り当てることができる重みの最大値。
EXPR_NEST_MAX	_SC_EXPR_NEST_MAX	exprコマンドで、括弧に入れてネストすることができる式の数の最大値。

(次ページに続く)

表 REF-10 (続き) sysconf 引数と戻り値

返されるシステム変数	nameのシンボリック定数	意味
ISO POSIX-2		
LINE_MAX	_SC_LINE_MAX	テキスト・ファイル処理するユーティリティのコマンド入力行 (標準入力または他のファイル) の長さの最大値 (バイト数)。長さには、終端の改行文字が含まれる。
RE_DUP_MAX	_SC_RE_DUP_MAX	edコマンドのmおよびn引数のように、インターバル表記引数を使用するとき、正規表現を何度繰り返して使用できるかを示す最大値。
_POSIX2_CHAR_TERM	_SC_2_CHAR_TERM	この変数は、システムが少なくとも1つのターミナル・タイプをサポートしている場合には値1を持ち、そうでない場合には-1が返される。
_POSIX2_C_BIND	_SC_2_C_BIND	この変数は、システムがC言語バインディング・オプションをサポートしている場合には値1を持ち、そうでない場合には-1が返される。
_POSIX2_C_DEV	_SC_2_C_DEV	この変数は、システムがオプションのISO POSIX-2 標準のC Language Development Utilitiesをサポートしている場合には値1を持ち、そうでない場合には-1が返される。
_POSIX2_C_VERSION	_SC_2_C_VERSION	ISO POSIX-2 標準 (C言語バインディング) のバージョンを示す整数値。ISO POSIX-2 標準の新しいバージョンごとに変更される。
_POSIX2_VERSION	_SC_2_VERSION	ISO POSIX-2 標準 (コマンド) のバージョンを示す整数値。ISO POSIX-2 標準の新しいバージョンごとに変更される。
_POSIX2_FORT_DEV	_SC_2_FORT_DEV	この変数は、システムがISO POSIX-2 標準のFortran Development Utilities Optionをサポートしている場合には値1を持ち、そうでない場合には-1が返される。
_POSIX2_FORT_RUN	_SC_2_FORT_RUN	この変数は、システムがISO POSIX-2 標準のFortran Runtime Utilities Optionをサポートしている場合には値1を持ち、そうでない場合には-1が返される。
_POSIX2_LOCALEDEF	_SC_2_LOCALEDEF	この変数は、システムがlocaledefコマンドでの新しいロケールの作成をサポートしている場合には値1を持ち、そうでない場合には-1が返される。
_POSIX2_SW_DEV	_SC_2_SW_DEV	この変数は、システムがISO POSIX-2 標準のSoftware Development Utilities Optionをサポートしている場合には値1を持ち、そうでない場合には-1が返される。
_POSIX2_UPE	_SC_2_UPE	この変数は、システムがUser Portability Utilities Optionをサポートしている場合には値1を持ち、そうでない場合には-1が返される。

(次ページに続く)

表 REF-10 (続き) sysconf 引数と戻り値

返されるシステム変数	nameのシンボリック定数	意味
POSIX 1003.1c-1995		
<code>_POSIX_THREADS</code>	<code>_SC_THREADS</code>	この変数は、システムが POSIX スレッドをサポートしている場合には値 1 を持ち、そうでない場合には-1が返される。
<code>_POSIX_THREAD_ATTR_STACKSIZE</code>	<code>_SC_THREAD_ATTR_STACKSIZE</code>	この変数は、システムが POSIX スレッドのスタック・サイズ属性をサポートしている場合には値 1 を持ち、そうでない場合には-1が返される。
<code>_POSIX_THREAD_PRIORITY_SCHEDULING</code>	<code>_SC_THREAD_PRIORITY_SCHEDULING</code>	1003.1c 実装は、リアルタイム・スケジューリング関数をサポートしている。
<code>_POSIX_THREAD_SAFE_FUNCTIONS</code>	<code>_SC_THREAD_SAFE_FUNCTIONS</code>	実装が POSIX 1003.1c のスレッド・セーフな ANSI C 関数をサポートしている場合に TRUE。
<code>PTHREAD_DESTRUCTOR_ITERATIONS</code>	<code>_SC_THREAD_DESTRUCTOR_ITERATIONS</code>	スレッドが終了したとき、DECthreads はスレッド内のすべての NULL でないスレッド固有データ値を巡回し、それぞれの登録されたデストラクタ・ルーチン (存在する場合) を呼び出す。デストラクタ・ルーチンは、1 つまたは複数のスレッド固有データ・キーの新しい値を作成することができる。この場合、DECthreads はプロセス全体を再実行する。 実装は、ループを <code>_SC_THREAD_DESTRUCTOR_ITERATIONS</code> 回実行した後、たとえ NULL でない値が存在してもスレッドを終了させる。
<code>PTHREAD_KEYS_MAX</code>	<code>_SC_THREAD_KEYS_MAX</code>	アプリケーションが作成できるスレッド固有データ・キーの数の最大値。
<code>PTHREAD_STACK_MIN</code>	<code>_SC_THREAD_STACK_MIN</code>	新しいスレッドのスタックのサイズとして許容される最小値。"stacksize"スレッド属性に対してこれよりも小さな値が指定された場合には、このサイズにまで切り上げられる。
<code>UINT_MAX</code>	<code>_SC_THREAD_THREADS_MAX</code>	アプリケーションが作成できるスレッドの数の最大値。DECthreads は固定の上限を設けていないため、この値は-1である。
X/Open		
<code>_XOPEN_VERSION</code>	<code>_SC_XOPEN_VERSION</code>	システムがサポートしている X/Open 標準の最新バージョンを示す整数。
<code>PASS_MAX</code>	<code>_SC_PASS_MAX</code>	パスワードの中の有効なバイト数の最大値 (終端の null を含まない)。
<code>XOPEN_CRYPT</code>	<code>_SC_XOPEN_CRYPT</code>	この変数は、システムが X/Open Encryption Feature Group をサポートしている場合には値 1 を持ち、そうでない場合には-1が返される。

(次ページに続く)

表 REF-10 (続き) sysconf 引数と戻り値

返されるシステム変数	nameのシンボリック定数	意味
X/Open		
XOPEN_ENH_I18N	_SC_XOPEN_ENH_I18N	この変数は、システムが X/Open enhanced Internationalization Feature Group をサポートしている場合には値 1 を持ち、そうでない場合には-1が返される。
XOPEN_SHM	_SC_XOPEN_SHM	この変数は、システムが X/Open Shared Memory Feature Group をサポートしている場合には値 1 を持ち、そうでない場合には-1が返される。
X/Open Extended		
ATEXIT_MAX	_SC_ATEXIT_MAX	1 つのプロセスでatexitを使って登録できる関数の数の最大値。
PAGESIZE	_SC_PAGESIZE	ページのサイズ (バイト数)。
PAGE_SIZE	_SC_PAGE_SIZE	PAGESIZE と同じ。PAGESIZE と PAGE_SIZE のどちらが定義されている場合には、もう一方の定数も同じ値で定義される。
IOV_MAX	_SC_IOV_MAX	1 つのプロセスがreadvまたはwritevで使用できるiovec構造体の数の最大値。
XOPEN_UNIX	_SC_XOPEN_UNIX	この変数は、システムが X/Open CAE Specification, August 1994, System Interfaces and Headers, Issue 4, Version 2 (ISBN: 1-85912-037-7, C435) をサポートしている場合には値 1 を持ち、そうでない場合には-1が返される。

Return value

x	システム上の現在の変数の値。この値は、呼び出し元プロセスが存在している間は変化しません。
-1	エラーを示します。 name引数の値が無効だった場合、errnoはエラーを示す値に設定されます。 name引数の値が未定義だった場合、errnoは変更されません。

system

指定された文字列を、コマンド・プロセッサで実行させるためにホスト環境に渡します。この関数は非リエントラントです。

Format

```
#include <stdlib.h>

int system (const char *string);
```

Argument

string

実行する文字列へのポインタ。stringが NULL である場合には、ゼロ以外の値が返されます。string は DCL コマンドであり、イメージの名前ではありません。イメージを実行するには、いずれかのexecルーチンを使用してください。

Description

system関数はサブプロセスをスポンし、そのサブプロセスの中で、stringで指定されたコマンドを実行します。system関数は、サブプロセスが完了するのを待ってから、関数の戻り値としてサブプロセス・ステータスを返します。

サブプロセスは、system呼び出しの中で、vforkの呼び出しによってスポンされます。このため、systemの呼び出しは、vforkの呼び出しの後、それに対応するexec関数の呼び出しの前に行うべきではありません。

OpenVMS Version 7.0 およびそれ以降のシステムでは、<stdlib.h>をインクルードし、_POSIX_EXIT 機能テスト・マクロを設定してコンパイルを行うと、system関数が返すステータスは、子を待つためにwaitpidを呼び出したときと同じようになります。このため、WIFEXITED および WEXITSTATUS マクロを使用して、0 ~ 255 の範囲の終了ステータスを取得するようにしてください。

_POSIX_EXIT 機能テスト・マクロの設定は、/DEFINE_POSIX_EXIT を使用するか、ファイルの先頭で、他のファイルをインクルードする前に#define _POSIX_EXIT を指定することによって行います。

Return value

ゼロ以外の値

stringが NULL だった場合には、system関数がサポートされていることを示す値 1 が返されます。stringが NULL でなければ、値はサブプロセスの OpenVMS リターン・ステータスです。

Example

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>    /* write, close */
#include <fcntl.h>     /* Creat */

main()
{
    int status,
        fd;

    /* Creat a file we are sure is there      */
    fd = creat("system.test", 0);
    write(fd, "this is an example of using system", 34);
    close(fd);

    if (system(NULL)) {
        status = system("DIR/NOHEAD/NOTRAIL/SIZE SYSTEM.TEST");
        printf("system status = %d\n", status);
    }
    else
        printf("system() not supported.\n");
}
```

この例のプログラムを実行すると、次の結果が出力されます。

```
DISK3$: [JONES.CRTL.2059.SRC]SYSTEM.TEST;1
1
system status = 1
```

tan

ラジアン of 引数の正接であるdouble値を返します。

Format

```
#include <math.h>

double tan (double x);
float tanf (float x); (Alpha, I64)
long double tanl (long double x); (Alpha, I64)
double tand (double x); (Alpha, I64)
float tandf (float x); (Alpha, I64)
long double tandl (long double x); (Alpha, I64)
```

Argument

x
実数として表現されたラジアン。

Description

tan関数は、ラジアン単位のxの正接を計算します。

tand関数は、度単位のxの正接を計算します。

Return value

x	引数の正接。
HUGE_VAL	xは特異点です。 (... $-3\pi/2$, $-\pi/2$, $\pi/2$...)
NaN	xは NaN です。errnoは EDOM に設定されます。
0	xは±無限大です。errnoは EDOM に設定されます。
±HUGE_VAL	オーバーフローが発生しました。errnoは ERANGE に設定されます。
0	アンダフローが発生しました。errnoは ERANGE に設定されます。

tanh

引数の双曲線正接を返します。

Format

```
#include <math.h>

double tanh (double x);
float tanhf (float x); (Alpha, I64)
long double tanhl (long double x); (Alpha, I64)
```

Argument

x
実数。

Description

tanh関数は、 $(e^{**x} - e^{**(-x)}) / (e^{**x} + e^{**(-x)})$ として計算される、引数の双曲線正接を返します。

Return value

n	引数の双曲線正接。
HUGE_VAL	引数の値が大きすぎます。errnoは ERANGE に設定されます。
NaN	xは NaN です。errnoは EDOM に設定されます。
0	アンダフローが発生しました。errnoは ERANGE に設定されます。

telldir

指定されたディレクトリ・ストリームに関連付けられている現在の位置を返します。
ディレクトリに対する操作を実行します。

Format

```
#include <dirent.h>

long int telldir (DIR *dir_pointer);
```

Argument

`dir_pointer`
オープン・ディレクトリのDIR構造体へのポインタ。

Description

telldir関数は、指定されたディレクトリ・ストリームに関連付けられている現在の位置を返します。

Return value

x	現在の位置。
-1	エラーを示します。エラーの詳細内容は、グローバルなerrnoに設定されます。

tempnam

一時的ファイルの名前を作成します。

Format

```
#include <stdio.h>

char *tempnam (const char *directory, const char *prefix, ... );
```

Argument

directory

ファイルを作成するディレクトリのパス名へのポインタ。

prefix

ファイル名の先頭の文字シーケンスへのポインタ。prefix引数は null であってもよく、一時的ファイル名の先頭の文字として使用される 5 文字までの文字列をポイントすることもできます。

...

省略可能な引数であり、1 または 0 を指定できます。1 を指定した場合、tempnam は、OpenVMS 形式のファイル指定を返します。0 を指定した場合、tempnam は、UNIX 形式のファイル指定を返します。UNIX 形式のファイル指定の詳細については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.4.3 項を参照してください。

Description

tempnam 関数は、一時的ファイルのためのファイル名を生成します。この関数では、ファイルを作成するディレクトリを選択することができます。

directory 引数が null であるか、適切なディレクトリのパス名でない文字列をポイントしていた場合には、<stdio.h> ヘッド・ファイルに P_tmpdir として定義されているパス名が使用されます。

ユーザ環境で TMPDIR 環境変数を指定することで、パス名の選択をバイパスすることができます。TMPDIR 変数の値は、目的の一時的ファイル・ディレクトリのパス名です。

prefix 引数を使用して、一時的ファイル名の 5 文字までの接頭辞を指定することができます。

tempnam関数は、後にfree関数の呼び出しで利用できる、生成されたパス名へのポインタを返します。

freeも参照してください。

注意

tmpnamとは対照的に、tempnamは呼び出しのたびに異なるファイル名を生成するとは限りません。tempnamは、指定された名前のファイルが存在する場合にのみ、新しいファイル名を生成します。呼び出しのたびに一意のファイル名が必要な場合には、tempnamではなくtmpnamを使用してください。

Return value

x	後にfree関数の呼び出しで利用できる、生成されたパス名へのポインタ。
NULL	エラーが発生しました。errnoはエラーを示す値に設定されます。

tgamma (Alpha, I64)

引数に対するガンマ関数の値を返します。

Format

```
#include <math.h>
double tgamma (double x);
float tgammaf (float x);
long double tgamma1 (long double x);
```

引数

x
実数値。

Description

tgamma関数は、xに対するガンマ関数の値を計算します。

Return value

n	成功したことを示します。 n は、xに対するガンマ関数の値です。
-1	xの値が負です。errnoに、EDOM が設定されます。
±HUGE_VAL	オーバーフローが発生したか、xの値が±0 です。errnoに、ERANGE が設定されます。
NaN	xの値が NaN か -Inf です。errnoに、EDOM が設定されます。
x	xの値が +Inf です。

time

1970 年 1 月 1 日の 00:00:00 からの (協定世界時での) 経過秒数を返します。

Format

```
#include <time.h>
time_t time (time_t *time_location);
```

関数バリエーション

`_DECC_V4_SOURCE` および `_VMS_V6_SOURCE` 機能テスト・マクロを定義してコンパイルすると、OpenVMS Version 7.0 より前の動作と等価な、`time`関数へのローカル時刻ベースのエントリ・ポイントが使用可能となります。

Argument

`time_location`
NULL, または返された時刻が格納される場所へのポインタ。`time_t`型は、`<time.h>`ヘッダ・ファイルに次のように定義されています。

```
typedef unsigned long int time_t;
```

Return value

<code>x</code>	Epoch からの経過秒数
<code>(time_t)(-1)</code>	エラーを示します。SYS\$TIMEZONE_DIFFERENTIAL 論理名の値が間違っている場合、関数は実行に失敗し、 <code>errno</code> は EINVAL に設定されます。

times

現在のプロセスと、その終了した子プロセスの累積時間を返します。

Format

```
#include <times.h>

clock_t times (struct tms *buffer); (OpenVMS V7.0 and higher)

void times (tbuffer_t *buffer); (pre OpenVMS V7.0)
```

Argument

buffer
ターミナル・バッファへのポインタ。

Description

プロセスと子プロセスの両方の時刻について、構造体は時間をユーザ時間とシステム時間に分離します。OpenVMS システムはシステム時間とユーザ時間を区別しないため、すべてのシステム時間は 0 として返されます。累積 CPU 時間は、10 ミリ秒単位で返されます。

子プロセスの累積時間は、C のメイン・プログラム、または VAXC\$CRTL_INIT あるいは DECC\$CRTL_INIT を呼び出すプログラムでのみ含まれます。

OpenVMS Version 7.0 およびそれ以降のシステムでは、times 関数は、過去の任意の基準時刻 (たとえばシステムのスタートアップ時刻) 以降の経過時間をクロック・ティック単位で返します。この基準時刻は、times 関数の呼び出しの間で変化しません。戻り値は、clock_t 型の値の範囲を超えてオーバーフローすることがあります。times は実行に失敗すると値 -1 を返します。HP C RTL は基準時刻としてシステムのブート時刻を使用します。

times

Return value

<code>x</code>	システムのブート時刻以降の、クロック・ティック単位での経過時間。
<code>(clock_t)(-1)</code>	エラーを示します。

tmpfile

更新が可能なようにオープンされた一時的ファイルを作成します。

Format

```
#include <stdio.h>
FILE *tmpfile (void);
```

Description

ファイルは、プロセスが存在している間のみ、またはファイルがクローズされるまで存在し、`vfork`の呼び出しの前後で保持されます。

Return value

<code>x</code>	ファイル・ポインタのアドレス (<stdio.h>ヘッダ・ファイルに定義)。
<code>NULL</code>	エラーを示します。

tmpnam

一時的ファイルに安全に使用できるファイル名を生成します。

Format

```
#include <stdio.h>

char *tmpnam (char *name);
```

関数バリエント

tmpnam関数は、それぞれ 32 ビットと 64 ビットのポインタ・サイズで使用するための `_tmpnam32` と `_tmpnam64` という名前のバリエントを持っています。ポインタ・サイズ固有の関数の使用方法については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

Argument

name

関数またはマクロのファイル名引数の代わりとして使用する名前を含んでいる文字列。後に null 引数を指定して `tmpnam` を呼び出すと、関数は現在の名前を上書きします。

Return value

x

name 引数が NULL ポインタ値 NULL である場合、`tmpnam` は内部記憶領域のアドレスを返します。name が NULL でない場合には、長さ `L_tmpnam` (`<stdio.h>` ヘッダ・ファイルに定義) の領域のアドレスと見なされます。この場合、`tmpnam` は結果として name 引数を返します。

toascii

8 ビット ASCII 文字の引数を 7 ビット ASCII 文字に変換します。

Format

```
#include <ctype.h>
int toascii (char character);
```

Argument

character
char型のオブジェクト。

Return value

x 7 ビット ASCII 文字。

tolower

文字を小文字に変換します。

Format

```
#include <ctype.h>

int tolower (int character);
```

Argument

character

unsigned charとして表現可能なint型のオブジェクト，またはEOFの値。これ以外の値が指定された場合の動作は未定義です。

Description

引数が大文字を表しており，プログラム・ロケール・カテゴリ LC_CTYPE の文字型情報の定義に従って，それに対応する小文字が存在する場合には，対応する小文字が返されます。

引数が大文字でない場合には，その文字が変更なしに返されます。

Return value

x	引数に対応する小文字。引数が大文字でない場合には，その引数を変更なしに返されます。
---	---

_tolower

大文字を小文字に変換します。

Format

```
#include <ctype.h>

int _tolower (int character);
```

Argument

character
この引数は大文字でなくてはなりません。

Description

_tolowerマクロは、引数が大文字でなくてはならない (小文字や EOF であってはならない) ことを除けば、tolower関数と等価です。

OpenVMS Version 8.3 では、_tolowerマクロの仕様が C99 ANSI 標準と X/Open 仕様に準拠するように変えられて、特に指定しない限りはパラメータを複数回評価しないようになっています。その結果、_tolowerマクロはtolower 関数を単に呼び出すだけとなり、式を評価する回数をユーザが指定できる場合の副作用 (i++や関数呼び出しなど) を回避できるようになりました。

_tolowerマクロの動作を最適化された古い仕様のままで維持したい場合は、/DEFINE=_FAST_TOUPPER を指定してコンパイルします。そうすれば、_tolowerの呼び出しが以前のリリースと同じように最適化されるので、実行時の呼び出しオーバーヘッドを避けることができます。ただし、その場合はパラメータがチェックされて計算方法が決定されるので、予期しない副作用が発生することがあります。そのため、/DEFINE=_FAST_TOUPPER を指定してコンパイルした場合は、副作用を引き起こしそうな引数を_tolowerマクロで使用しないでください。たとえば、次の例のようにこのマクロを使用しても、期待した結果は得られません。

```
d = _tolower (C++);
```

`_tolower`

Return value

`x`

引数に対応する小文字。

touchwin

ターミナル・スクリーンに、指定されたウィンドウの最新の編集済みバージョンを表示します。

Format

```
#include <curses.h>

int touchwin (WINDOW *win);
```

Argument

win
ウィンドウへのポインタ。

Description

touchwin関数は、通常はオーバーラップするウィンドウを再表示する目的にのみ使用されます。

Return value

OK	成功を示します。
ERR	エラーを示します。

toupper

文字を大文字に変換します。

Format

```
#include <ctype.h>
int toupper (int character);
```

Argument

character

unsigned charとして表現可能なint型のオブジェクト，またはEOFの値。これ以外の値が指定された場合の動作は未定義です。

Description

引数が小文字を表しており，プログラム・ロケール・カテゴリ LC_CTYPE の文字型情報の定義に従って，それに対応する大文字が存在する場合には，対応する大文字が返されます。

引数が小文字でない場合には，その文字が変更なしに返されます。

Return value

x	引数に対応する大文字。引数が小文字でない場合には，その引数を変更なしに返されます。
---	---

_toupper

小文字を大文字に変換します。

Format

```
#include <ctype.h>
int _toupper (int character);
```

Argument

character
この引数は小文字でなくてはなりません。

Description

_toupperマクロは、引数が小文字でなくてはならない (大文字や EOF であってはならない) ことを除けば、toupper関数と等価です。

OpenVMS Version 8.3 では、_toupperマクロの仕様が C99 ANSI 標準と X/Open 仕様に準拠するように変えられて、特に指定しない限りはパラメータを複数回評価しないようになっています。その結果、_toupperマクロはtoupper 関数を単に呼び出すだけとなり、式を評価する回数をユーザが指定できる場合の副作用 (i++ や関数呼び出しなど) を回避できるようになりました。

_toupperマクロの動作を最適化された古い仕様のままで維持したい場合は、/DEFINE=_FAST_TOUPPER を指定してコンパイルします。そうすれば、_toupperの呼び出しが以前のリリースと同じように最適化されるので、実行時の呼び出しオーバーヘッドを避けることができます。ただし、その場合はパラメータがチェックされて計算方法が決定されるので、予期しない副作用が発生することがあります。そのため、/DEFINE=_FAST_TOUPPER を指定してコンパイルした場合は、副作用を引き起こしそうな引数を _toupperマクロで使用しないでください。たとえば、次の例のようにこのマクロを使用しても、期待した結果は得られません。

```
d = _toupper (c++);
```

_toupper

Return value

x

引数に対応する大文字。

towctrans

指定されたマッピング記述子に従って、1つのワイド文字を別のワイド文字にマップします。

Format

```
#include <wctype.h>

wint_t towctrans (wint_t wc, wctrans_t desc);
```

Argument

wc

マップするワイド文字。

desc

wctrans関数の呼び出しを通して取得されるマッピングの記述。

Description

towctrans関数は、descによって記述されたマッピングを使用して、wcで指定されたワイド文字をマップします。

LC_CTYPEカテゴリの現在の設定は、descの値を返したwctrans関数の呼び出しの際の設定と同じでなくてはなりません。

戻り値

x

ワイド文字wcがdescによって記述されたマッピング内に存在する場合には、そのマップされた値。存在しない場合には、wcの値が返されます。

tolower

ワイド文字コードの引数を小文字に変換します。引数が大文字でなければ、引数が変更なしに返されます。

Format

```
#include <wctype.h> (ISO C)
#include <wchar.h> (XPG4)
int tolower (wint_t wc);
```

Argument

wc

現在のロケールで有効なワイド文字として表現可能なwint_t型のオブジェクト、または WEOF の値。それ以外の値での動作は未定義です。

Description

引数が大文字のワイド文字だった場合には、それに対応する小文字のワイド文字 (ロケールの LC_CTYPE カテゴリの定義に従う) が返されます (存在する場合)。存在しない場合、関数は入力引数を変更なしに返します。

towupper

ワイド文字の引数を大文字に変換します。引数が小文字でなければ、引数が変更なしに返されます。

Format

```
#include <wctype.h> (ISO C)
#include <wchar.h> (XPG4)
int towupper (wint_t wc);
```

Argument

wc

現在のロケールで有効なワイド文字として表現可能なwint_t型のオブジェクト、または WEOF の値。それ以外の値での動作は未定義です。

Description

引数が小文字のワイド文字だった場合には、それに対応する大文字のワイド文字 (ロケールの LC_CTYPE カテゴリの定義に従う) が返されます (存在する場合)。存在しない場合、関数は入力引数を変更なしに返します。

`trunc` (*Alpha, I64*)

`trunc` (*Alpha, I64*)

引数を整数値に切り捨てます。

Format

```
#include <math.h>
double trunc (double x);
float truncf (float x,);
long double trunc (long double x);
```

Argument

`x`
浮動小数点数。

Return value

`n` 引数を切り捨てて得られた整数値。

truncate

ファイルの長さを、指定されたバイト数に変更します。

Format

```
#include <unistd.h>

int truncate (const char *path, off_t length);
```

Argument

path

切り捨てるファイルの名前。この引数は、呼び出し元プロセスが書き込み許可を持っている通常のファイルを指定するパス名をポイントしていません。

length

ファイルの新しい長さ (バイト数)。lengthのoff_t型は 64 ビット整数または 32 ビット整数です。64 ビット・インタフェースでは 2 GB よりも大きいファイル・サイズを扱うことができ、コンパイル時に次のように_LARGEFILE 機能テスト・マクロを定義することで選択できます。

```
CC/DEFINE=_LARGEFILE
```

Description

truncate関数は、ファイルの長さを、length引数で指定されたバイト数に変更します。

新しい長さが以前の長さよりも小さかった場合、関数は指定されたファイルからlengthバイトよりも後のデータをすべて削除します。新しいファイルの終端と以前のファイルの終端の間のすべてのファイル・データは破棄されます。

ストリーム・ファイルでは、新しい長さが以前の長さよりも大きかった場合、以前のファイルの終端と新しいファイルの終端の間に、すべてゼロから構成される新しいファイル・データが追加されます (レコード・ファイルでは、この方法でファイルを拡張することはできません)。

truncate

Return value

0	成功を示します。
-1	エラーが発生しました。errnoはエラーを示す値に設定されます。

ttyname, ttyname_r

ターミナルのパス名を見つけます。

Format

```
#include <unistd.h> (Compatibility)
char *ttyname (void); (Compatibility)
#include <unistd.h> (OpenVMS V7.3-2 and higher)
char *ttyname (int filedes); (OpenVMS V7.3-2 and higher)
int ttyname_r (int filedes, char *name, size_t namesize); (OpenVMS V7.3-2 and higher), (Alpha, I64)
```

Argument

filedes

オープン・ファイルの記述子。

name

ターミナル名が格納されるバッファへのポインタ。

namesize

name引数が指すバッファの長さ。

Description

引数のないttyname関数は、旧版との互換性のためにのみ用意されています。この旧式の実装では、デフォルトの入力デバイス (stdin) であるファイル記述子 0 に対応するターミナル・デバイスの、null で終了する名前へのポインタを返します。SYSS\$INPUT が TTY デバイスでない場合、値 0 が返されます。

ttyname_r関数と、filedes引数をとるttynameの実装は、UNIX 標準に準拠しています。また、OpenVMS Version 7.3-2 およびそれ以降でのみ利用できます。

標準に準拠しているttyname関数は、ファイル記述子filedesに対応するターミナルの、null で終了するパス名からなる文字列へのポインタを返します。戻り値は、呼び出しのたびに内容が上書きされる、静的データを指しています。ttynameインタフェースは、リエントラントとは限りません。

ttyname_r関数は、ファイル記述子fildesに対応するターミナルの、nullで終了するパス名を格納する領域へのポインタを、nameという名前の文字配列として受け取ります。この配列の長さは、namesize文字で、名前と末尾のnull文字用の領域が必要です。ターミナル名の最大長は、TTY_NAME_MAXです。

成功すると、ttynameは文字列へのポインタを返します。失敗すると、NULLポインタが返され、エラーを示す値がerrnoに設定されます。

成功すると、ttyname_rは、nameが指すバッファにnullで終了する文字列としてターミナル名を格納し、0を返します。失敗すると、エラーを示すエラー番号が返されます。

Return value

x	成功して終了すると、ttynameは、nullで終了する文字列へのポインタを返します。
NULL	失敗すると、ttynameはNULLポインタを返し、失敗を示す値をerrnoに設定します。 <ul style="list-style-type: none"> EBADF – fildes引数が、正しいファイル記述子ではありません。 ENOTTY – fildes引数が、ターミナル・デバイスを指していません。
0	成功して終了すると、ttyname_rは0を返します。
n	失敗すると、ttyname_rは失敗を示す値をerrnoに設定し、同じerrnoコードを返します。 <ul style="list-style-type: none"> EBADF – fildes引数が、正しいファイル記述子ではありません。 ENOTTY – fildes引数が、TTYデバイスを指していません。 ERANGE – namesizeの値が、返却する文字列(末尾のnull文字を含む)の長さよりも小さくなっています。
0	旧式のttynameで、SYSS\$INPUTがTTYデバイスではないことを示します。

tzset

タイム・ゾーン変換の設定とアクセスを行います。

Format

```
#include <time.h>

void tzset (void);

extern char *tzname[];
extern long int timezone;
extern int daylight;
```

Description

tzset関数は、ctime、localtime、mktime、strftime、およびwcsftime関数が使用する時刻変換情報を初期化します。

tzset関数は、以下の外部変数を設定します。

- tznameは、"std"を標準タイム・ゾーンの3バイトの名前、"dst"をサマータイム・ゾーンの3バイトの名前として、次のように設定される。

```
tzname[0] = "std"
tzname[1] = "dst"
```

- daylightは、タイム・ゾーンにサマータイムが決して設定されない場合には0に設定される。そうでなければ、daylightは1に設定される。
- timezoneは、UTCとローカル標準時の間の差に設定される。

環境変数TZは、tzsetが時刻変換情報をどのように初期化するかを指定します。

- TZが環境に存在しなかった場合には、次のように実装依存のタイム・ゾーン情報が使用される。

デフォルトのタイム・ゾーン規則を記述するtzfile形式のファイルをポイントするSYSS\$LOCALTIMEシステム論理名の定義に従って、ローカル・ウォール・クロック時刻に最も近いものが使用される。

このシステム論理名は、OpenVMS Version 7.0以降のバージョンにおいてインストール時に設定されるもので、SYSS\$COMMON:[SYSS\$ZONEINFO.SYSTEM]をルートとするディレクトリ階層下のタイム・ゾーン・ファイルを値とする。

 注意

HP C RTL は、パブリック・ドメインのタイム・ゾーン処理パッケージを使用しており、タイム・ゾーン変換規則は容易にアクセスあるは変更可能なファイルに格納されています。これらのファイルは、SYS\$COMMON:[SYS\$ZONEINFO.SYSTEM.SOURCES]ディレクトリに置かれています。

タイム・ゾーン・コンパイラ zic は、これらのファイルを、<tzfile.h>ヘッダ・ファイルによって記述されている特殊な形式に変換します。変換後のファイルは、SYS\$TZDIR システム論理名がポイントする SYS\$COMMON:[SYS\$ZONEINFO.SYSTEM]をルート・ディレクトリとして作成されます。この形式は、タイム・ゾーン情報を処理する C ライブラリ関数から読み込むことができます。たとえば、米国東部では、SYS\$LOCALTIME は SYS\$COMMON:[SYS\$ZONEINFO.SYSTEM.US]EASTERN に定義されます。

- TZが環境に存在するが、その値が null 文字列だった場合には、協定世界時 (UTC) が使用される (うるう秒の修正は行われない)。
- TZが環境に存在しており、その値が null 文字列でなければ、その値は表 REF-11 に示す 3 つの形式のいずれかになっている。

表 REF-11 タイム・ゾーン初期化規則

TZ の形式	意味
:	UTC が使用される。
:pathname	コロンの後の文字列は、時刻変換情報を読み込むtzfile形式のファイルのパス名を指定する。スラッシュ(/)で始まるパス名は絶対パス名を表す。それ以外のパス名は、SYS\$TZDIRによって指定されるシステム時刻変換情報ディレクトリからの相対パスである。このディレクトリはデフォルトではSYS\$COMMON:[SYS\$ZONEINFO.SYSTEM]である。
stdoffset[dst[offset][,rule]]	この値は、まず時刻変換情報を読み込むファイルのパス名として使用される (:pathname形式と同じ)。 そのファイルを読み込めなかった場合、この値は次のように時刻変換情報の直接指定として解釈される。

(次ページに続く)

表 REF-11 (続き) タイム・ゾーン初期化規則

TZ の形式	意味
	<p>stdとdst— タイム・ゾーンを指定する 3 つ以上の文字:</p> <ul style="list-style-type: none"> • std— 標準タイム・ゾーン。必須。 • dst— サマータイム・ゾーン。オプション。dstが省略された場合、サマータイムは適用されない。 <p>大文字と小文字は明示的に許容されている。以下のものを除く任意の文字が使用できる。</p> <ul style="list-style-type: none"> • 数字 • 先頭のコロロン(:) • コンマ(,) • マイナス(-) • プラス(+) • ASCII null 文字 <p>offset—UTC に合わせるためにローカル時刻に追加される値。オフセットは次の形式を持つ。</p> <p>hh[:mm[:ss]]</p> <p>この形式の各項目は、以下の意味を持つ。</p> <ul style="list-style-type: none"> • hh (時) は、0 ~ 24 の範囲の 1 つまたは 2 つの数字。 • mm (分) は、0 ~ 59 の範囲の値 (オプション)。 • ss (秒) は、0 ~ 59 の範囲の値 (オプション)。 <p>stdの後のオフセットは必須である。dstの後にオフセットがなかった場合には、標準時よりも 1 時間早いサマータイムが仮定される。1 つまたは複数の数字を使用できる。値はつねに 10 進数として解釈される。</p> <p>オフセットの前にマイナス記号(-)がある場合、そのタイム・ゾーンはグリニッジの東にある。そうでなければグリニッジの西にある。これはプラス記号(+)で示すこともできる。</p>

(次ページに続く)

表 REF-11 (続き) タイム・ゾーン初期化規則

TZ の形式	意味
	<p>rule— サマータイムにいつ切り替わり、いつ終了するのかを示す。規則は次の形式を持つ。</p> <p>start[/time], end[/time]</p> <p>各項目は以下の意味を持つ。</p> <ul style="list-style-type: none"> • startは、標準時からサマータイムに切り替わる日付を示す。 • endは、サマータイムから標準時へと戻る日付を示す。 <p>startとendが省略された場合のデフォルトは、米国のサマータイムの開始日と終了日である。startとendの形式は、次のいずれかでなくてはならない。</p> <ul style="list-style-type: none"> • Jn— ユリウス日n ($1 \leq n \leq 365$)。うるう日は算入されない。つまり、うるう年を含むすべての年において、2月28日は59日目、3月1日は60日目である。2月29日を明示的に参照することはできない。 • n— ゼロ・ベースのユリウス日 ($0 \leq n \leq 365$)。うるう日は算入され、2月29日を参照することができる。 • Mm.n.d— m月のn番目のd曜日。 <div style="margin-left: 40px;"> $\begin{array}{rcl} 0 & n & 5 \\ 0 & d & 6 \\ 1 & m & 12 \end{array}$ <p>nが5である場合には、m月の最後のd曜日を示す。0番目の曜日は日曜日である。</p> </div> <p>time— 現在のローカル時刻で、サマータイムへの切り替えまたは終了が起こる時刻。time引数は、先頭のマイナス(−)またはプラス(+)記号を使用できないという点を除けば、offsetと同じ形式を持つ。timeが指定されなかった場合のデフォルトは02:00:00である。</p> <p>TZ指定に rule が含まれていない場合、SYS\$POSIXRULES システム論理名に定義されているtzfile形式のファイルで指定された規則が使用される。その際、標準時とサマータイムの UTC からのオフセットは、TZで指定されたオフセット値に置き換えられる。</p> <p>TZがtzfile形式のファイルを指定しておらず、直接指定としても解釈できない場合には、UTC が使用される。</p>

注意

OpenVMS Version 7.0 で導入された UTC ベースの時刻関数は、非 UTC ベースの時刻関数と比べると性能が低くなっていました。

OpenVMS Version 7.1 では、性能を改善するために、タイム・ゾーン・ファイル用のキャッシュが追加されました。キャッシュのサイズは論理名 DECC\$TZ_CACHE_SIZE によって決定されます。ほとんどの国が時刻を年に2回変更することに対応して、デフォルトのキャッシュ・サイズはタイム・ゾーン・ファイルを2つ保持できるだけの大きさになっています。

ctime , localtime , mktime , strftime , および wcsftime も参照してください。

TZ 指定の例

```
EST5EDT4,M4.1.0,M10.5.0
```

この TZ 指定の例は、1987 年に定められた米国東部タイム・ゾーンの規則を記述したものです。

- EST (Eastern Standard Time) は標準時を示し、UTC から 5 時間遅れています。
- EDT (Eastern Daylight Time) はサマータイムを示し、UTC から 4 時間遅れています。EDT は 4 月の最初の日曜日に始まり、10 月の最後の日曜日に終わります。

どちらの場合も time は指定されていないため、変更はデフォルトの時刻である午前 2:00 に行われます。この例では、開始日および終了日はデフォルト値に等しいので省略することもできます。

ualarm

インターバル・タイマのタイムアウトを設定または変更します。

Format

```
#include <unistd.h>

useconds_t ualarm (useconds_t mseconds, useconds_t interval);
```

Argument

mseconds

実時間のマイクロ秒を指定します。

interval

タイマを繰り返すインターバルを指定します。

Description

ualarm関数は、usecondsで指定された実時間のマイクロ秒数が経過した後に、呼び出し元プロセスに対してSIGALRMシグナルを送信します。interval引数がゼロでなければ、intervalで指定されたマイクロ秒の間隔で、タイムアウト通知が繰り返し発生します。通知シグナルSIGALRMがキャッチされないか、無視された場合、呼び出し元プロセスは終了します。

ualarm関数とsetitimer関数を組み合わせて呼び出した場合、AST 状態が無効になっていれば、戻り値は無効となります。

ualarm関数とsetitimer関数を組み合わせて呼び出した場合、AST 状態が有効になっていれば、戻り値は有効となります。

これは、AST が無効化されていたり、AST レベルで呼び出されたハンドラから呼び出されていた場合、AST ハンドラを呼び出してタイマの以前の値をクリアすることが不可能であるためです。

注意

ualarmとalarmまたはsleepの間の相互作用については定められていません。

setitimerも参照してください。

Return value

n	以前のualarmまたはsetitimer呼び出しからの残りのマイクロ秒数。
0	予定されているタイムアウトがない, またはualarmが以前に呼び出されたことはありません。
-1	エラーを示します。

umask

新しいファイルが作成されるときに使用されるファイル保護マスクを作成し、以前のマスク値を返します。

Format

```
#include <stat.h>

mode_t umask (mode_t mode_complement);
```

Argument

`mode_complement`

新しいファイルが作成されるときに、どのビットをオフにするかを示します。各ビットの意味については、`chmod`の説明を参照してください。

Description

初期状態では、ファイル保護マスクはカレント・プロセスのデフォルト・ファイル保護を元に設定されます。これは、Cのメイン・プログラムがスタートアップするか、`DECC$CRTL_INIT` (または`VAXC$CRTL_INIT`) が呼び出されたときに行われます。この設定は、`umask`を呼び出すことで、プログラムが作成するすべてのファイルについて変更することもできますし、`chmod`を使用して個々のファイルのファイル保護を変更することもできます。`open`または`creat`によって作成されるファイルのファイル保護は、`open`と`creat`のモード引数と、前の呼び出しで`umask`に渡された値の補数の間のビット論理積です。

注意

UNIX システム・コール関数`umask`、`mkdir`、`creat`、および`open`で OpenVMS RMS のデフォルト保護を持つファイルを作成するには、`umask`を決して呼び出さないプログラムの中で、0777 のファイル保護モード引数を指定して`mkdir`、`creat`、および`open`を呼び出します。これらのデフォルト保護では、ACL やファイルの以前のバージョンなどに基づいて、保護が正しく設定されます。

`vfork/exec`呼び出しを行うプログラムでは、新しいプロセス・イメージは、`umask`が呼び出されたかどうかの状態を、呼び出し元のプロセス・イメージから継承します。`umask`設定と、`umask`関数が呼び出されたかどうかの状態は、どちらも属性として継承されます。

Return value

x

以前のマスク値。

uname

システム識別情報を取得します。

Format

```
#include <utsname.h>

int uname (struct utsname *name);
```

引数

name
現在のシステム識別子。

Description

uname関数は、現在のシステムを識別する情報の null で終了する文字列を、name引数が参照する構造体に格納します。

utsname構造体は<utsname.h>ヘッダ・ファイルに定義されており、以下のメンバを含んでいます。

sysname	オペレーティング・システムの名前
nodename	このマシンのネットワーク名
release	オペレーティング・システムのリリース・レベル
version	オペレーティング・システムのバージョン・レベル
machine	マシン・ハードウェア・プラットフォーム

Return value

0	成功を示します。
-1	エラーを示します。errnoまたはvaxc\$errnoが適切な値に設定されます。

ungetc

文字を入力ストリームに戻し、ストリームの位置をその文字の前に設定します。

Format

```
#include <stdio.h>

int ungetc (int character, FILE *file_ptr);
```

Argument

character
int型の値。

file_ptr
ファイル・ポインタ。

Description

ungetc関数を使用すると、文字がfile_ptrが指定するファイルに戻されます。

以前にそのファイルに対する操作が行われていなくても、1つの文字が戻されることが保証されます。fseek関数は、戻された文字に関するすべての記憶を消去します。戻された文字は、下位のファイルには書き込まれません。戻される文字がEOFだった場合、操作は失敗し、入力ストリームは変更されず、EOFが返されます。

fseekとgetcも参照してください。

Return value

x	戻される文字。
EOF	文字を戻せないことを示します。

ungetwc

ワイド文字を入力ストリームに戻します。

Format

```
#include <wchar.h>

wint_t ungetwc (wint_t wc, FILE *file_ptr);
```

Argument

wc
wint_t型の値。

file_ptr
ファイル・ポインタ。

Description

ungetwc関数を使用すると、ワイド文字がfile_ptrが指定するファイルに戻されます。

以前にそのファイルに対する操作が行われていなくても、1つの文字が戻されることが保証されます。戻された文字が読み込まれる前にファイル位置設定関数 (fseekなど) が呼び出されると、戻された文字を表すバイトは失われます。

戻される文字が WEOF だった場合、操作は失敗し、入力ストリームは変更されず、WEOF が返されます。

getwcも参照してください。

Return value

x

戻される文字。

WEOF

関数が文字を戻せなかったことを示します。errnoは以下のいずれかに設定されます。

- EBADF— ファイル記述子が有効でない。
- EALREADY— すでに同じファイルに対する操作が進行中である。
- EILSEQ— 無効なワイド文字コードが検出された。

unlink (Alpha, I64)

指定したシンボリック・リンクをシステムから削除します。

Format

```
#include <unistd.h>

int unlink (const char *link_name);
```

Argument

link_name
削除するシンボリック・リンクの名前。

Description

unlink関数は、指定したシンボリック・リンク (link_name) をシステムから削除します。シンボリック・リンクの内容は調べられません。また、その内容で指定されているファイルについても、何も行われません。シンボリック・リンク以外のファイルを指定して unlink関数を実行したときの動作は、C RTLのremove関数と同じです。

symlink, readlink, realpath, lchown, およびlstatも参照してください。

Return value

0	成功したことを示します。
-1	エラーが発生したことを示します。指定したファイル (link_name) は変更されていません。errnoに、removeから返された errnoの値が設定されます。

unordered (*Alpha, I64*)

引数の片方または両方が NaN だった場合に値 1 (TRUE) を返します。それ以外の場合は、値 0 (FALSE) を返します。

Format

```
#include <math.h>

double unordered (double x, double y);
float unorderedf (float x, float y);
long double unorderedl (long double x, long double y);
```

Argument

x
実数。

y
実数。

Return value

1	引数の片方または両方が NaN です。
0	どちらの引数も NaN ではありません。

unsetenv

環境リストから、環境変数名のすべてのインスタンスを削除します。

Format

```
#include <stdlib.h>

void unsetenv (const char *name);
```

Argument

name
環境リストから削除する環境変数。

Description

unsetenv関数は、name引数がポイントする変数名のすべてのインスタンスを環境リストから削除します。

usleep

指定されたインターバルだけ実行を一時停止します。

Format

```
#include <unistd.h>

int usleep (unsigned int mseconds);
```

Argument

mseconds
実行を一時停止するマイクロ秒数。

Description

usleep関数は、mseconds引数で指定されたマイクロ秒数だけ、カレント・プロセスの実行を一時停止します。この引数は1,000,000未満でなくてはなりません。値が0だった場合、呼び出しは何の効果も持ちません。

各プロセスに、1つのリアルタイムのインターバル・タイマが存在しています。usleep関数は、このタイマの前の設定には干渉しません。プロセスがusleepを呼び出す前にこのタイマを設定しており、msecondsで指定された時間がインターバル・タイマの以前の設定と等しいか、それよりも長かった場合、プロセスはタイマが満了する直前にウェイクアップします。

Return value

0	成功を示します。
-1	エラーが発生したことを示します。errnoはEINVALに設定されます。

utime

ファイルのアクセスおよび変更時刻を設定します。

Format

```
#include <utime.h>

int utime (const char *path, const struct utimbuf *times);
```

Argument

path

ファイルへのポインタ。

times

NULL ポインタ, またはutimbuf構造体へのポインタ。

Description

utime関数は, path引数で指定されたファイルのアクセスおよび変更時刻を設定します。

timesが NULL ポインタである場合には, ファイルのアクセスおよび変更時刻は現在の時刻に設定されます。utimeをこの形で使用するためには, プロセスの実効ユーザ ID がファイルのオーナーと一致する, またはプロセスがファイルへの書き込み許可を持っているか, 適切な特権を持っている必要があります。

timesが NULL ポインタでない場合には, utimbuf構造体へのポインタとして解釈され, アクセスおよび変更時刻は指定された構造体に含まれている値に設定されます。utimeをこの形で使用できるのは, ファイルのユーザ ID に等しい実効ユーザ ID を持っているプロセスか, または適切な特権を持つプロセスに限られます。

utimbuf構造体は<utime.h>ヘッダ・ファイルによって定義されています。utimbuf構造体の中の時刻は, Epoch 以降の経過秒数です。

実行に成功すると, utimeは最後のファイル状態の変更時刻st_ctimeを更新します。<stat.h>ヘッダ・ファイルを参照してください。

 注意 (Alpha, I64)

OpenVMS Alpha システムおよび I64 システムでは、stat、fstat、utime、およびutimes関数は、新しいファイル・システムの POSIX 準拠のファイル・タイムスタンプ・サポートに対応して拡張されています。

このサポートは、V7.3 およびそれ以降の OpenVMS Alpha システム上の ODS-5 デバイスでのみ使用できます。

この変更が行われる以前、statおよびfstat関数は、以下のファイル属性に基づいてst_ctime、st_mtime、およびst_atimeフィールドの値を設定していました。

```
st_ctime—ATR$C_CREDATE (ファイル作成時刻)
st_mtime—ATR$C_REVDATE (ファイル更新時刻)
st_atime—ファイル・アクセス時刻がサポートされていなかったため、常にst_mtimeに設定されていた
```

また、ファイル変更時刻については、utimeおよびutimesはATR\$C_REVDATE ファイル属性を変更し、ファイル・アクセス時刻引数は無視していました。

変更が行われて以降、ODS-5 デバイス上のファイルについては、statおよびfstat関数は、以下の新しいファイル属性に基づいてst_ctime、st_mtime、およびst_atimeフィールドの値を設定するようになりました。

```
st_ctime—ATR$C_ATTDATE (最終属性変更時刻)
st_mtime—ATR$C_MODDATE (最終データ変更時刻)
st_atime—ATR$C_ACCDATE (最終アクセス時刻)
```

ODS-2 デバイスのようにATR\$C_ACCDATE が0である場合、statおよびfstat関数はst_atimeをst_mtimeに設定します。

ファイル変更時刻については、utimeおよびutimes関数は、ATR\$C_REVDATE とATR\$C_MODDATE の両方のファイル属性を変更します。ファイル・アクセス時刻については、これらの関数はATR\$C_ACCDATE ファイル属性を変更します。ODS-2 デバイス上でATR\$C_MODDATE およびATR\$C_ACCDATE ファイル属性を設定しても効果はありません。

互換性を保つために、デバイスの種類にかかわらず、stat、fstat、utime、およびutimesの以前の動作はデフォルトのままとなっています。

新しい動作を有効にするためには、アプリケーションを呼び出す前に、DECC\$EFS_FILE_TIMESTAMPS 論理名を明示的に "ENABLE" に定義する必要があります。この論理名を設定しても、ODS-2 デバイス上のファイルに対するstat、fstat、utime、およびutimesの動作には影響はありません。

Return value

0

実行に成功しました。

-1

エラーを示します。関数はerrnoを以下のいずれかの値に設定します。

utime関数は、以下の場合に失敗します。

- EACCES – path接頭辞のコンポーネントで検索許可が拒否された。または、times引数が NULL ポインタで、プロセスの実効ユーザ ID がファイルのオーナーと一致せず、書き込みアクセスが拒否された。
- ELOOP – pathを解決する過程で検出したシンボリック・リンクの数が多すぎた。
- ENAMETOOLONG – path引数の長さが PATH_MAX を超えた、pathname コンポーネントが NAME_MAX よりも長かった、またはシンボリック・リンクのパス名解決の過程で、長さが PATH_MAX を超える中間結果が生成された。
- ENOENT – pathが既存のファイルを指定していない、またはpathが空の文字列である。
- ENOTDIR – path接頭辞のコンポーネントがディレクトリでない。
- EPERM – timesは NULL ポインタではなく、呼び出し元プロセスの実効ユーザ ID はファイルへの書き込みアクセスを持っているが、ファイルのオーナーと一致せず、呼び出し元プロセスは適切な特権を持っていない。
- EROFS – ファイルを含んでいるファイル・システムは読み込み専用である。

utimes

ファイルのアクセスおよび変更時刻を設定します。

Format

```
#include <time.h>

int utimes (const char *path, const struct timeval times[2]);
```

Argument

path
ファイルへのポインタ。

times
timeval構造体の配列。第 1 の配列メンバは最終アクセスの日付と時刻を、第 2 のメンバは最終変更の日付と時刻を表します。timeval構造体に含まれる時刻は Epoch 以来の経過秒数とマイクロ秒数を表しますが、最も近い秒への丸めが行われることもあります。

Description

utimes関数は、path引数がポイントするファイルのアクセスおよび変更時刻を、times 引数の値に設定します。utimes関数では、マイクロ秒までの精度で時刻を指定することができます。

times引数が NULL ポインタである場合には、ファイルのアクセスおよび変更時刻は現在の時刻に設定されます。関数を用いるためには、プロセスの実効ユーザ ID がファイルのオーナーと一致する、またはプロセスがファイルへの書き込み許可を持っているか、適切な特権を持っている必要があります。

実行に成功すると、utimesは最後のファイル状態の変更時刻st_ctimeを更新します。

 注意 (Alpha, I64)

OpenVMS Alpha システムおよび I64 システムでは、stat、fstat、utime、およびutimes関数は、新しいファイル・システムの POSIX 準拠のファイル・タイムスタンプ・サポートに対応して拡張されています。

このサポートは、V7.3 およびそれ以降の OpenVMS Alpha システム上の ODS-5 デバイスでのみ使用できます。

この変更が行われる以前、statおよびfstat関数は、以下のファイル属性に基づいてst_ctime、st_mtime、およびst_atimeフィールドの値を設定していました。

```
st_ctime—ATR$C_CREDATE (ファイル作成時刻)
st_mtime—ATR$C_REVDATE (ファイル更新時刻)
st_atime—ファイル・アクセス時刻がサポートされていなかったため、常にst_mtimeに設定されていた
```

また、ファイル変更時刻については、utimeおよびutimesはATR\$C_REVDATE ファイル属性を変更し、ファイル・アクセス時刻引数は無視していました。

変更が行われて以降、ODS-5 デバイス上のファイルについては、statおよびfstat関数は、以下の新しいファイル属性に基づいてst_ctime、st_mtime、およびst_atimeフィールドの値を設定するようになりました。

```
st_ctime—ATR$C_ATTDATE (最終属性変更時刻)
st_mtime—ATR$C_MODDATE (最終データ変更時刻)
st_atime—ATR$C_ACCDATE (最終アクセス時刻)
```

ODS-2 デバイスのようにATR\$C_ACCDATE が0である場合、statおよびfstat関数はst_atimeをst_mtimeに設定します。

ファイル変更時刻については、utimeおよびutimes関数は、ATR\$C_REVDATE とATR\$C_MODDATE の両方のファイル属性を変更します。ファイル・アクセス時刻については、これらの関数はATR\$C_ACCDATE ファイル属性を変更します。ODS-2 デバイス上でATR\$C_MODDATE およびATR\$C_ACCDATE ファイル属性を設定しても効果はありません。

互換性を保つために、デバイスの種類にかかわらず、stat、fstat、utime、およびutimesの以前の動作はデフォルトのままとなっています。

新しい動作を有効にするためには、アプリケーションを呼び出す前に、DECC\$EFS_FILE_TIMESTAMPS 論理名を明示的に "ENABLE" に定義する必要があります。この論理名を設定しても、ODS-2 デバイス上のファイルに対するstat、fstat、utime、およびutimesの動作には影響はありません。

Return value

0	実行に成功しました。
-1	<p>エラーを示します。ファイルの時刻は変更されず、関数は <code>errno</code> を以下のいずれかの値に設定します。</p> <p><code>utimes</code> 関数は、以下の場合に失敗します。</p> <ul style="list-style-type: none">• <code>EACCES</code>—<code>path</code> 接頭辞のコンポーネントで検索許可が拒否された。または、<code>times</code> 引数が <code>NULL</code> ポインタで、プロセスの実効ユーザ ID がファイルのオーナーと一致せず、書き込みアクセスが拒否された。• <code>ELOOP</code>—<code>path</code> を解決する過程で検出したシンボリック・リンク・リンクの数が多すぎた。• <code>ENAMETOOLONG</code>—<code>path</code> 引数の長さが <code>PATH_MAX</code> を超えた、<code>pathname</code> コンポーネントが <code>NAME_MAX</code> よりも長かった、またはシンボリック・リンクのパス名解決の過程で、長さが <code>PATH_MAX</code> を超える中間結果が生成された。• <code>ENOENT</code>—<code>path</code> が既存のファイルを指定していない、または <code>path</code> が空の文字列である。• <code>ENOTDIR</code>—<code>path</code> 接頭辞のコンポーネントがディレクトリでない。• <code>EPERM</code>—<code>times</code> 引数は <code>NULL</code> ポインタではなく、呼び出し元プロセスの実効ユーザ ID はファイルへの書き込みアクセスを持っているが、ファイルのオーナーと一致せず、呼び出し元プロセスは適切な特権を持っていない。• <code>EROFS</code>—ファイルを含んでいるファイル・システムは読み込み専用である。

VAXC\$CRTL_INIT

他の言語からHP C RTL を呼び出したり，メイン関数が C でない場合にHP C RTL を利用したりすることができます。この関数はランタイム環境を初期化し，終了および条件ハンドラの両方を設定します。VAXC\$CRTL_INITはDECC\$CRTL_INITの同義語です。どちらの名前も同じルーチンを呼び出します。

Format

```
#include <signal.h>

void VAXC$CRTL_INIT();
```

Description

次の例は，VAXC\$CRTL_INIT関数を使ってHP C RTL を呼び出す Pascal プログラムを示しています。

OpenVMS VAX システムの場合:

```
$ PASCAL EXAMPLE
$ LINK EXAMPLE,SYS$LIBRARY:DECCRTL/LIB
$ TY EXAMPLE.PAS
PROGRAM TESTC(input, output);
PROCEDURE VAXC$CRTL_INIT; extern;
BEGIN
    VAXC$CRTL_INIT;
END
$
```

OpenVMS Alpha システムの場合:

```
$ PASCAL EXAMPLE
$ LINK EXAMPLE,SYS$LIBRARY:VAXCRTL/LIB
$ TY EXAMPLE.PAS
PROGRAM TESTC(input, output);
PROCEDURE VAXC$CRTL_INIT; extern;
BEGIN
    VAXC$CRTL_INIT;
END
$
```

共用可能イメージがこの関数を呼び出す必要があるのは，そのイメージがシグナル処理，環境変数，I/O，終了処理，およびデフォルト・ファイル保護マスクのためのHP C関数を含んでいるか，コンテキストを継承しなくてはならない子プロセスである場合に限られます。

多くの初期化活動は 1 回しか実行されませんが、DECC\$CRTL_INITは何度呼び出しても安全です。OpenVMS VAX システムでは、DECC\$CRTL_INITは、DECC\$CRTL_INITが呼び出されるたびに、DECC\$CRTL_INITを呼び出したルーチンのフレーム内で、HP CRTL の内部 OpenVMS 例外ハンドラを設定します。

OpenVMS 例外が UNIX シグナルにマップされるためには、現在のコール・スタックの中の少なくとも 1 つのフレームがハンドラを設定している必要があります。

VAXC\$ESTABLISH

特定のルーチン用に OpenVMS 例外ハンドラを設定するために使用されます。この関数は、それを呼び出したルーチンの中で、特殊な HP C RTL 例外ハンドラを設定します。この特殊なハンドラは、後のルーチン内で発生するすべての RTL 関連の例外をキャッチし、それ以外のすべての例外をプログラマが用意したハンドラに渡します。

Format

```
#include <signal.h>

void VAXC$ESTABLISH (unsigned int (*exception_handler)(void *sigarr, void *mecharr));
```

Argument

exception_handler

OpenVMS 例外ハンドラとして設定する関数の名前。この関数へのポインタを、VAXC\$ESTABLISH へのパラメータとして渡すことになります。

sigarr

シグナル配列へのポインタ。

mecharr

メカニズム配列へのポインタ。

Description

プログラムが HP C RTL ルーチンの `setjmp` または `longjmp` を使用するときには、`LIB$ESTABLISH` の代わりに `VAXC$ESTABLISH` を使用する必要があります。`setjmp` と `longjmp`、または `sigsetjmp` と `siglongjmp` を参照してください。

`VAXC$ESTABLISH` 関数は HP C for OpenVMS 関数からしか呼び出せません。この関数は、HP C コンパイラがランタイム・スタック上に割り当てるデータ・スペースに依存しているからです。HP C 関数から直接に OpenVMS システム・ライブラリ・ルーチン `LIB$ESTABLISH` を呼び出すと、`setjmp` および `longjmp` 関数で未定義の動作が発生します。

OpenVMS 例外に UNIX スタイルのシグナルを生成させるためには、ユーザ例外ハンドラは、自分が処理したくない例外を受信したときに `SS$_RESIGNAL` を返さなくてはなりません。`SS$_NORMAL` を返すと、UNIX スタイルのシグナルの生成は行われません。UNIX シグナルは、メインの C プログラムのスタック・フレーム内の例外ハンドラによって生成されたときと同じように扱われます。すべての OpenVMS 例外が UNIX

シグナルに対応しているわけではありません。OpenVMS 例外と UNIX スタイルのシグナルの関係については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 4 章を参照してください。

引数 NULL を指定して VAXC\$ESTABLISH を呼び出すと、そのルーチン内の既存のハンドラが取り消されます。

注意

- OpenVMS Alpha システムでは、VAXC\$ESTABLISH は、HP C RTL 関数ではなく、コンパイラの組み込み関数として実装されている。 (*Alpha only*)
 - OpenVMS VAX システムでは、/NAMES=AS_IS を指定してコンパイルされたプログラムは、そのプログラムが/PREFIX_LIBRARY_ENTRIES スイッチを指定してコンパイルされていたかどうかにかかわらず、名前 VAXC\$ESTABLISH を解決するために SYS\$LIBRARY:DECCRTL.OLB をリンクする必要がある。これは実装内における制約である。 (*VAX only*)
-

va_arg

引数リストの次の項目を返します。

Format

```
#include <stdarg.h> (ANSI C)
#include <varargs.h> (HP C Extension)
type va_arg (va_list ap, type);
```

Argument

ap

取得しようとしている次の引数を含んでいる可変リスト。

type

リスト内の次の項目のサイズを決定するために使用されるデータ型。引数リストはさまざまなサイズの項目を含むことができますが、これを実行時に判定することはできないため、呼び出し元ルーチンは期待される引数の型を指定する必要があります。

Description

va_arg関数は、type に基づくリスト・インクリメントによって指定されたアドレスにあるオブジェクトを解釈します。対応する引数が存在しない場合の動作は未定義です。

va_argを使って移植性のあるアプリケーションを作成したい場合には、<varargs.h>ヘッダ・ファイルではなく (ANSI C 標準で定義されている) <stdarg.h>ヘッダ・ファイルをインクルードし、va_argは<stdarg.h>に定義されている他の関数およびマクロのみと組み合わせて使用するようにします。

<stdarg.h>の関数と定義を使った引数リストの処理の例については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』例 3-6 を参照してください。

va_count

引数リストの中のロングワード (*VAX only*) またはクォドワード (*Alpha only*) の数を返します。

Format

```
#include <stdarg.h> (ANSI C)
#include <varargs.h> (HP C Extension)
void va_count (int count);
```

Argument

count

ロングワード (*VAX only*) またはクォドワード (*Alpha only*) の数を取得しようとしている整変数の名前。

Description

va_count マクロは、引数リストの中のロングワード (*VAX only*) またはクォドワード (*Alpha only*) の数を count に格納します。count に返される値は、count フィールド自体を除いた、関数引数ブロックの中のロングワード (*VAX only*) またはクォドワード (*Alpha only*) の数です。

引数リストが、その格納にロングワード (*VAX only*) またはクォドワード (*Alpha only*) 以下のメモリしか必要としない項目を含んでいる場合、count 引数に返される値は引数の数でもあります。しかし、引数リストがロングワード (*VAX only*) またはクォドワード (*Alpha only*) よりも大きい項目を含んでいる場合には、引数の数を得るためには count の解釈を行う必要があります。double は 8 バイトなので、OpenVMS VAX システムでは引数リスト位置を 2 つ分、OpenVMS Alpha システムと I64 システムでは 1 つ分占有します。

va_count マクロは HP C for OpenVMS システムに固有のものであり、移植性はありません。

va_end

<varargs.h>または<stdarg.h>セッションを終了します。

Format

```
#include <stdarg.h> (ANSI C)
#include <varargs.h> (HP C Extension)
void va_end (va_list ap);
```

Argument

ap

引数リストの長さをトラバースするために使用されるオブジェクト。引数apは、形式のセクションに示している方法で宣言し、使用する必要があります。

Description

それぞれva_start ... va_endで区切って、引数リストの複数のトラバースルを実行することができます。va_end関数は、apを NULL に設定します。

この関数を使って移植性のあるアプリケーションを作成したい場合には、<varargs.h>ヘッダ・ファイルではなく (ANSI C 標準で定義されている) <stdarg.h>ヘッダ・ファイルをインクルードし、va_endは<stdarg.h>に定義されている他のルーチンのみと組み合わせて使用するようになります。

<stdarg.h>の関数と定義を使った引数リストの処理の例については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』例 3-6 を参照してください。

va_start, va_start_1

変数を引数リストの先頭に初期化するために使用されます。

Format

```
#include <varargs.h> (HP C Extension)

void va_start (va_list ap);

void va_start_1 (va_list ap, int offset);
```

Argument

ap

オブジェクト・ポインタ。引数apは、形式のセクションに示している方法で宣言し、使用する必要があります。

offset

リスト内の (引数リストの先頭ではなく) それ以降の引数をポイントするためにapをインクリメントするバイト数。ゼロ以外のオフセットを使用することで、apを、複数の固定引数の後にある最初のオプション引数のアドレスに初期化することができます。

Description

va_startマクロは、変数apを引数リストの先頭に初期化します。

va_start_1マクロは、既知の数の定義済み引数の後にある引数のアドレスにapを初期化します。printf関数は、引数リスト全体の先頭から一定のオフセットに可変長の引数リストを含んでいるHP C RTL 関数の一例です。可変長引数リストは、書式文字列のアドレスだけずれています。

va_start_1で使用するオフセット引数の値を決定する際には、OpenVMS 呼び出し規則の意味を考慮に入れる必要があります。

OpenVMS VAX では、ほとんどの引数項目はロングワードです。たとえば、OpenVMS VAX のchar型とshort型の引数は、引数リストに含まれているときにはロングワードのメモリを使用します。一方、OpenVMS VAX のfloat型の引数は、double型に変換されるため、2つのロングワードを使用します。

OpenVMS Alpha システムおよび I64 システムでは、すべての引数項目がクォドワードです。

注意

特に他のプログラミング言語で書かれたプログラムから (C で書かれた) サブルーチンに渡された引数リストにアクセスするときには、OpenVMS 呼び出し規則の意味を考慮に入れる必要があります。OpenVMS 呼び出し規則の詳細については、『HP C User's Guide for OpenVMS Systems』または『OpenVMS Calling Standard』を参照してください。

上に示したバージョンの `va_start` と `va_start_1` は HP C RTL に固有のものであり、移植性はありません。

次の構文は、ANSI C 標準に定義されている、`<stdarg.h>` ヘッダ・ファイルの `va_start` マクロに関するものです。

Format

```
#include <stdarg.h> (ANSI C)

void va_start (va_list ap, parmN);
```

Argument

`ap`

オブジェクト・ポインタ。引数 `ap` は、形式のセクションに示している方法で宣言し、使用する必要があります。

`parmN`

最後の既知の固定引数の名前。

Description

ポインタ `ap` は、引数リストの中の `parmN` の後にある最初のオプション引数をポイントするように初期化されます。

このバージョンの `va_start` は、必ず関数プロトタイプを使って宣言され、定義されている関数と組み合わせて使用するようにしてください。また、移植性のあるプログラムを作成したい場合には、このバージョンの `va_start` を使用してください。

<stdarg.h>の関数と定義を使った引数リストの処理の例については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル(上巻)』例 3-6 を参照してください。

vfork

独立した子プロセスを作成します。この関数は非リエントラントです。

Format

```
#include <unistd.h>

int vfork (void); (_DECC_V4_SOURCE)

pid_t vfork (void); (not _DECC_V4_SOURCE)
```

Description

HP C for OpenVMSシステムの用意しているvfork関数は、他のCの実装のfork関数とは異なります。表 REF-12 に、両者の間の主な違い 2 つを示します。

表 REF-12 vfork 関数と fork 関数

vfork 関数	fork 関数
exec関数とともに使用される。 親プロセスの一部の特性を共用する独立した子プロセスを作成する。	非同期処理で、exec関数なしで使用できる。 vforkが呼び出されたポイントで分岐する、親プロセスの正確な複製を作成する。親と子は、異なる実行段階にある同じプロセスであるかのように振る舞う。

vfork関数は、それ以降のexec関数の呼び出しのために必要なセットアップを行います。vforkはプロセスは作成しませんが、以下のステップを実行します。

- 後にexec関数の呼び出しのリターン・アドレスとして使用されるリターン・アドレス (vfork呼び出しのアドレス) を保存する。
- 現在のコンテキストを保存する。
- 初めて呼び出されたとき (exec関数の呼び出しの前) には整数 0 を返す。対応するexec関数呼び出しが行われた後は、exec関数はvfork呼び出しのポイントで親プロセスに制御を返し、戻り値として子プロセスのプロセス ID を返す。exec関数が実行に失敗しない限り、呼び出しがvfork関数に対して 1 回、exec関数に対して 1 回行われたにもかかわらず、制御はvforkから 2 回返ったように見える。

vfork関数の動作は、setjmp関数の動作に似ています。vforkとsetjmpは、どちらも後に使用されるリターン・アドレスを設定し、このアドレスをセットアップするために初めて呼び出されたときには整数 0 を返し、第 2 の戻り値を、対応するexecまたはlongjmp関数呼び出しからではなく、自分で返したような形で返します。

しかし、`setjmp`とは違って`vfork`では、`volatile`で修飾されたものも含むすべてのローカルな自動変数が、`vfork`の呼び出しとそれに対応する`exec`ルーチンの呼び出しの間で変更された場合には、不定の値を持つ可能性があります。

Return value

0	コンテキストの作成に成功したことを示します。
ゼロ以外	子プロセスのプロセス ID (PID) を示します。
-1	エラーを示します。 - 子プロセスの作成に失敗しました。

vfprintf

引数リストに基づいて、書式付きの出力をプリントします。

Format

```
#include <stdio.h>

int vfprintf (FILE *file_ptr, const char *format, va_list ap);
```

Argument

file_ptr

出力の送信先のファイルへのポインタ。

format

書式指定を含んだ文字列へのポインタ。書式指定および変換指定とそれぞれに対応する引数の詳細については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 2 章を参照してください。

ap

結果として得られる型が、書式指定で与えられた変換指定に対応する式のリスト。

Description

vprintfとvsprintfも参照してください。

Return value

x

負の値

書き込まれたバイト数。

出力エラーを示します。関数はerrnoを設定します。設定される可能性のあるerrno値のリストについては、fprintfを参照してください。

vfscanf

引数リストに基づいて、書式付き入力を読み込みます。

Format

```
#include <stdio.h>

int vfscanf (FILE *file_ptr, const char *format, va_list ap);
```

Argument

file_ptr

入力テキストを提供するファイルへのポインタ。

format

書式指定を含んだ文字列へのポインタ。

ap

結果として得られる型が、書式指定で与えられた変換指定に対応する式のリスト。

Description

vfscanf関数はfscanf関数に似ていますが、可変個の引数を指定して呼び出される代わりに、va_start (また、おそらくはそれ以降のva_arg呼び出し) によって初期化された引数リストを指定して呼び出される点が異なります。

変換指定が与えられなかった場合、入力ポインタは省略することができます。そうでなければ、関数呼び出しは変換指定と同じ数の入力ポインタを持っていないとならず、変換指定は入力ポインタの型と一致していません。

変換指定は、左から右の順序で入力ソースと照合されます。余分な入力ポインタが存在する場合には、無視されます。

書式指定および変換指定とそれぞれに対応する引数の詳細については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第2章を参照してください。

この関数は、照合に成功し、代入が行われた入力項目の数を返します。

vscanfとvsscanfも参照してください。

Return value

n

照合に成功し、代入が行われた入力項目の数。

EOF

ファイルの終端に達したか、読み込みエラーが発生したことを示します。読み込みエラーが発生した場合、関数は `errno` を以下のいずれかに設定します。

- `EILSEQ`— 無効な文字を検出した。
- `EINVAL`— 引数が足りなかった。
- `ENOMEM`— 変換のために利用できるメモリが足りなかった。
- `ERANGE`— 浮動小数点計算のオーバーフロー。
- `EVMSError`— 変換不可能な OpenVMS エラー。 `vaxc$errno` に OpenVMS エラー・コードが含まれている。オーバーフローのために数値への変換が失敗した可能性がある。

また、関数は I/O サブシステムから返されたエラーの結果として、`errno` を以下の値に設定することもあります。

- `EBADF`— ファイル記述子が有効でない。
- `EIO`— I/O エラー。
- `ENXIO`— デバイスが存在しない。
- `EPIPE`— パイプが破壊されている。
- `EVMSError`— 変換不可能な OpenVMS エラー。 `vaxc$errno` に OpenVMS エラー・コードが含まれている。等価な C エラー・コードがない I/O エラーが発生した可能性がある。

vfwprintf

ワイド文字書式文字列の制御下で、ストリームに出力を書き込みます。

Format

```
#include <wchar.h>

int vfwprintf (FILE *stream, const wchar_t *format, va_list ap);
```

Argument

stream

ファイル・ポインタ。

format

書式指定を含んだワイド文字列へのポインタ。書式指定および変換指定とそれぞれに対応する引数の詳細については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 2 章を参照してください。

ap

出力に必要な項目の可変リスト。

Description

vfwprintf関数は、fwprintf関数の可変引数リストをap引数に置き換えたものです。apの初期化は、<stdarg.h>のva_startマクロ (また、おそらくはその後のva_arg呼び出し) を使って行います。

streamがポイントするストリームが向きを持っていなかった場合、vfwprintfはストリームをワイド向きにします。

fwprintfも参照してください。

Return value

n

負の値

書き込まれたワイド文字の数。

エラーを示します。関数はerrnoを以下のいずれかに設定します。

- EILSEQ— 無効な文字を検出した。
- EINVAL— 引数が足りなかった。
- ENOMEM— 変換のために利用できるメモリが足りなかった。
- ERANGE— 浮動小数点計算のオーバーフロー。
- EVMSERR— 変換不可能な OpenVMS エラー。vaxc\$errnoに OpenVMS エラー・コードが含まれている。オーバーフローのために数値への変換が失敗した可能性がある。

また、関数は I/O サブシステムから返されたエラーの結果として、errnoを以下の値に設定することもあります。

- EBADF— ファイル記述子が有効でない。
- EIO— I/O エラー。
- ENOSPC— ファイルを含んでいるデバイス上に空きスペースがない。
- ENXIO— デバイスが存在しない。
- EPIPE— パイプが破壊されている。
- ESPIPE— 追加用にオープンされたファイル内での不正なシーク。
- EVMSERR— 変換不可能な OpenVMS エラー。vaxc\$errnoに OpenVMS エラー・コードが含まれている。等価な C エラー・コードがない I/O エラーが発生した可能性がある。

Examples

次の例は、一般的なエラー報告ルーチンにおけるvfwprintf関数の使用例を示しています。

```
#include <stdarg.h>
#include <stdio.h>
#include <wchar.h>

void error(char *function_name, wchar_t *format, ... );
{
    va_list args;
    va_start(args, format);
    /* print out name of function causing error */
    fprintf(stderr, L"ERROR in %s: ", function_name);
    /* print out remainder of message */
    vfwprintf(stderr, format, args);
    va_end(args);
}
```

vfwscanf

ワイド文字書式文字列の制御下で、ストリームから入力を読み込みます。

Format

```
#include <wchar.h>

int vfwscanf (FILE *stream, const wchar_t *format, va_list ap);
```

Argument

stream

ファイル・ポインタ。

format

書式指定を含んだワイド文字列へのポインタ。

ap

結果として得られる型が、書式指定で与えられた変換指定に対応する式のリスト。

Description

vfwscanf関数はfwscanf関数に似ていますが、可変個の引数を指定して呼び出される代わりに、va_start (また、おそらくはそれ以降のva_arg呼び出し) によって初期化された引数リスト (ap) を指定して呼び出される点が異なります。

streamがポイントするストリームが向きを持っていなかった場合、vfwscanfはストリームをワイド向きにします。

書式指定および変換指定とそれぞれに対応する引数の詳細については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第2章を参照してください。

Return value

n	照合に成功し、代入が行われたワイド文字入力項目の数。
EOF	変換が行われる前に読み込みエラーが発生したことを示します。関数はerrnoを設定します。この関数が設定する値のリストについては、vfwscanfを参照してください。

vprintf

引数リストに基づいて、書式付きの出力をプリントします。

この関数はprintf関数に似ていますが、可変個の引数を指定して呼び出される代わりに、<stdarg.h>のva_startマクロ (また、おそらくはそれ以降のva_arg呼び出し) によって初期化された引数リスト<stdarg.h>を指定して呼び出される点が異なります。

Format

```
#include <stdio.h>

int vprintf (const char *format, va_list ap);
```

Argument

format

書式指定を含んだ文字列へのポインタ。書式指定および変換指定とそれぞれに対応する引数の詳細については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 2 章を参照してください。

ap

出力に必要な項目の可変リスト。

Description

vfprintfおよびvsprintf関数を参照してください。

Return value

x

負の値

書き込まれたバイト数。

出力エラーを示します。関数はerrnoを設定します。設定される可能性があるerrno値のリストについては、fprintfを参照してください。

vscanf

引数リストに基づいて、書式付きの入力を読み込みます。

Format

```
#include <stdio.h>

int vscanf (const char *format, va_list ap);
```

Argument

format

書式指定を含んだ文字列へのポインタ。

ap

結果として得られる型が、書式指定で与えられた変換指定に対応する式のリスト。

Description

vscanf関数はscanf関数に似ていますが、可変個の引数を指定して呼び出される代わりに、va_startマクロ (また、おそらくはそれ以降のva_arg呼び出し) によって初期化された引数リスト (ap) を指定して呼び出される点が異なります。

書式指定および変換指定とそれぞれに対応する引数の詳細については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第2章を参照してください。

scanf、vfscanf、およびvsscanfも参照してください。

Return value

n

照合に成功し、代入が行われた入力項目の数。

EOF

変換が行われる前に読み込みエラーが発生したことを示します。関数はerrnoを設定します。この関数が設定する値のリストについては、vfscanfを参照してください。

vsnprintf (Alpha, I64)

引数リストに基づいて、書式付きの出力をプリントします。

Format

```
#include <stdio.h>

int vsnprintf (char *str, size_t n, const char *format, va_list ap);
```

Argument

str

書式付きの出力を受け取る文字列へのポインタ。

format

書式指定を含んでいる文字列へのポインタ。書式指定および変換指定とそれぞれに対応する引数の詳細については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 2 章を参照してください。

ap

結果として得られる型が、書式指定で与えられた変換指定に対応する式のリスト。

Description

vsnprintf関数は、snprintf関数と同じです。ただし、可変個の引数で呼び出されるのではなく、va_start (および、それ以降のva_arg呼び出し) で初期化された引数リストで呼び出されます。

この関数は、va_endマクロを呼び出しません。これは、この関数はva_argマクロを呼び出すため、呼び出し後のapの値が規定されていないからです。

vsnprintfを使用するアプリケーションは、後でva_end(ap) を呼び出して、クリーン・アップを行う必要があります。

Return value

x	nが十分に大きい場合に、strに書き込まれるバイトの数 (末尾の null バイトを除く) です。
負の値	出力エラーが発生したことを示します。この関数は、errnoを設定します。設定される可能性のあるerrno値のリストについては、fprintfを参照してください。

vsprintf

引数リストに基づいて、書式付きの出力をプリントします。

この関数はsprintf関数に似ていますが、可変個の引数を指定して呼び出される代わりに、va_start (また、おそらくはそれ以降のva_arg呼び出し) によって初期化された引数リストを指定して呼び出される点が異なります。

Format

```
#include <stdio.h>

int vsprintf (char *str, const char *format, va_list ap);
```

Argument

str

書式付き出力を受け取る文字列へのポインタ。この文字列は、出力を保持できるだけの大きさであると仮定されます。

format

書式指定を含んだ文字列へのポインタ。書式指定および変換指定とそれぞれに対応する引数の詳細については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第2章を参照してください。

ap

結果として得られる型が、書式指定で与えられた変換指定に対応する式のリスト。

Return value

x

負の値

書き込まれたバイト数。

出力エラーが発生したことを示します。関数はerrnoを設定します。設定される可能性のあるerrno値のリストについては、fprintfを参照してください。

vsscanf

引数リストに基づいて、書式付きの入力を読み込みます。

Format

```
#include <stdio.h>

int vsscanf (char *str, const char *format, va_list ap);
```

Argument

str

sscanfへの入力テキストを提供する文字列のアドレス。

format

書式指定を含んだ文字列へのポインタ。

ap

結果として得られる型が、書式指定で与えられた変換指定に対応する式のリスト。

Description

vsscanf関数はsscanf関数に似ていますが、可変個の引数を指定して呼び出される代わりに、va_start (また、おそらくはそれ以降のva_arg呼び出し) によって初期化された引数リストを指定して呼び出される点が異なります。

また、vsscanf関数は、第1引数がストリームではなくワイド文字列を指定するという点を除けば、vfscanf関数と等価です。ワイド文字列の終端に達するのは、vfscanf関数でEOFを検出することに対応します。

書式指定および変換指定とそれぞれに対応する引数の詳細については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル(上巻)』第2章を参照してください。

vsscanfおよびsscanfも参照してください。

Return value

n	照合に成功し、代入が行われた入力項目の数。
EOF	変換が行われる前に読み込みエラーが発生したことを示します。関数はerrnoを設定します。この関数が設定する値のリストについては、vfscanfを参照してください。

vswprintf

ワイド文字書式文字列の制御下で、ストリームに出力を書き込みます。

Format

```
#include <wchar.h>

int vswprintf (wchar_t *s, size_t n, const wchar_t *format, va_list ap);
```

Argument

s

マルチバイト文字シーケンスへのポインタ。

n

マルチバイト文字を構成するバイト数の最大値。

format

書式指定を含んだワイド文字列へのポインタ。書式指定および変換指定とそれぞれに対応する引数の詳細については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 2 章を参照してください。

ap

出力に必要な項目の可変リスト。

Description

vswprintf関数は、可変引数リストがap引数に置き換えられていることを除けば、swprintf関数と等価です。apの初期化はva_startマクロを使って、またおそらくはそれ以降のva_arg呼び出しを使って行います。

swprintfも参照してください。

Return value

n

負の値

書き込まれたワイド文字の数。

エラーを示します。関数はerrnoを以下のいずれかに設定します。

- EILSEQ— 無効な文字を検出した。
- EINVAL— 引数が足りなかった。
- ENOMEM— 変換のために利用できるメモリが足りなかった。
- ERANGE— 浮動小数点計算のオーバーフロー。
- EVMSERR— 変換不可能な OpenVMS エラー。vaxc\$errnoに OpenVMS エラー・コードが含まれている。オーバーフローのために数値への変換が失敗した可能性がある。

また、関数は I/O サブシステムから返されたエラーの結果として、errnoを以下の値に設定することもあります。

- EBADF— ファイル記述子が有効でない。
- EIO— I/O エラー。
- ENOSPC— ファイルを含んでいるデバイス上に空きスペースがない。
- ENXIO— デバイスが存在しない。
- EPIPE— パイプが破壊されている。
- ESPIPE— 追加用にオープンされたファイル内での不正なシーク。
- EVMSERR— 変換不可能な OpenVMS エラー。vaxc\$errnoに OpenVMS エラー・コードが含まれている。等価な C エラー・コードがない I/O エラーが発生した可能性がある。

vswscanf

ワイド文字書式文字列の制御下で、ストリームから入力を読み込みます。

Format

```
#include <wchar.h>

int vswscanf (wchar_t *s, const wchar_t *format, va_list ap);
```

Argument

s

入力を取得するワイド文字列へのポインタ。

format

書式指定を含んだワイド文字列へのポインタ。

ap

結果として得られる型が、書式指定で与えられた変換指定に対応する式のリスト。

Description

vswscanf関数はswscanf関数に似ていますが、可変個の引数を指定して呼び出される代わりに、va_start (また、おそらくはそれ以降のva_arg呼び出し) によって初期化された引数リスト (ap) を指定して呼び出される点が異なります。

また、vswscanf関数は、第1引数がストリームではなくワイド文字列を指定するという点を除けば、vfwscanf関数と等価です。ワイド文字列の終端に達するのは、vfwscanf関数でEOFを検出することに対応します。

書式指定および変換指定とそれぞれに対応する引数の詳細については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第2章を参照してください。

vfwscanfおよびswscanfも参照してください。

Return value

n	読み込んだワイド文字の数。
EOF	変換が行われる前に読み込みエラーが発生したことを示します。関数はerrnoを設定します。この関数が設定する値のリストについては、 vfscanf を参照してください。

vwprintf

ワイド文字書式文字列の制御下で、ワイド文字の配列に出力を書き込みます。

Format

```
#include <wchar.h>

int vwprintf (const wchar_t *format, va_list ap);
```

Argument

format

書式指定を含んだワイド文字列へのポインタ。書式指定および変換指定とそれぞれに対応する引数の詳細については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 2 章を参照してください。

ap

出力に必要な項目の可変リスト。

Description

vwprintf関数は、可変引数リストがap引数に置き換えられていることを除けば、wprintf関数と等価です。apの初期化はva_startマクロを使って、またおそらくはそれ以降のva_arg呼び出しを使って行います。vwprintf関数はva_endマクロを呼び出しません。

wprintfも参照してください。

Return value

x

書き込まれたワイド文字の数。終端の null ワイド文字は含みません。

負の値

エラーを示します。n個以上のワイド文字の書き込みが要求されたか、変換エラーが発生しました。後者の場合、errnoは EILSEQ に設定されます。

vwscanf

ワイド文字書式文字列の制御下で、ワイド文字の配列から入力を読み込みます。

Format

```
#include <wchar.h>

int vwscanf (const wchar_t *format, va_list ap);
```

Argument

format

書式指定を含んだワイド文字列へのポインタ。

ap

結果として得られる型が、書式指定で与えられた変換指定に対応する式のリスト。

Description

vwscanf関数はwscanf関数に似ていますが、可変個の引数を指定して呼び出される代わりに、va_start (また、おそらくはそれ以降のva_arg呼び出し) によって初期化された引数リスト (ap) を指定して呼び出される点が異なります。

書式指定および変換指定とそれぞれに対応する引数の詳細については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第2章を参照してください。

wscanfも参照してください。

Return value

n

読み込まれたワイド文字の数。

EOF

変換が行われる前に読み込みエラーが発生したことを示します。関数はerrnoを設定します。この関数が設定する値のリストについては、vfscanfを参照してください。

wait

終了する前に、子プロセスの状態をチェックします。子プロセスは、親プロセスが終了するときに終了します。

Format

```
#include <wait.h>

pid_t wait (int *status);
```

Argument

status

終了した子プロセスの最終的な状態が格納される位置のアドレス。子プロセスはexit関数を使って状態を設定することができ、親プロセスはstatusを指定することでこの値を取得することができます。

Description

wait関数は、終了した子プロセスの最終的な状態が子から返されるまで、親プロセスを一時停止します。

OpenVMS Version 7.0 およびそれ以降のシステムでは、<wait.h>をインクルードし、_POSIX_EXIT機能テスト・マクロを指定してコンパイルした場合には(/DEFINE=_POSIX_EXIT を指定するか、ファイルの先頭で他のファイルをインクルードする前に#define _POSIX_EXITを使用する)、wait関数はwaitpid(0, status, 0)と等価です。

Return value

x	終了した子プロセスのプロセス ID (PID)。複数の子プロセスが作成されていた場合、waitは終了した子のうち最後に作成された子の PID を返します。それ以降の呼び出しでは、終了した子のうちその前に作成された子の PID を返します。
-1	子プロセスはスポンされていないでした。

wait3

子プロセスが停止または終了するのを待ちます。

Format

```
#include <wait.h>

pid_t wait3 (int *status_location, int options, struct rusage *resource_usage);
```

Argument

status_location

<wait.h>ヘッダ・ファイルに定義されている，子プロセスの終了状態を含んでいる位置へのポインタ。

OpenVMS Version 7.2 およびそれ以降，_VMS_WAIT マクロを定義してコンパイルした場合，wait3関数はstatus_location引数で指定されたアドレスに，子プロセスのOpenVMS 完了コードを格納するようになりました。

options

関数の動作を変更するフラグ。これらのフラグの定義は，説明のセクションに示しています。

resource_usage

終了した子プロセスのリソース利用情報を含んでいる構造体の位置。

Description

wait3関数は，要求が完了するまで呼び出し元プロセスを一時停止し，呼び出し元スレッドのみが一時停止するように再定義を行います。

options引数は関数の動作を変更します。options引数のフラグは，ビット論理和を指定することで組み合わせることができます。以下にフラグを示します。

WNOWAIT

status_locationに状態が返されるプロセスを，待機可能状態に置くよう指定する。このプロセスを再び待っても，同じ結果が得られる。

WNOHANG	呼び出し元プロセスが一時停止しないようにする。停止または終了した子プロセスが複数存在する場合には、WNOHANGフラグが指定されなかった場合と同様に、そのうちの1つが選ばれ、waitpid関数はその子プロセスのプロセスIDを返す。終了したプロセスが存在しない(つまり、waitpidが呼び出し元プロセスをWNOHANGフラグなしで一時停止した)場合には、0(ゼロ)が返される。プロセス0を待つことはできないので、この戻り値から混乱が生じることはない。
WUNTRACED	子プロセスがSIGTTIN、SIGTTOU、SIGSTOP、またはSIGTSTOPシグナルを受信したために、カレント・プロセスの子プロセスが停止したときに、呼び出しに追加情報を返させるように指定する。

wait3関数が、子プロセスの状態が入手可能であるために返った場合には、子プロセスのプロセスIDが返されます。情報は、status_locationがnullでなければ、このポインタがポイントしている位置に格納されます。

status_locationがポイントする位置に格納される値が0(ゼロ)になるのは、状態が以下のいずれかを行った終了した子プロセスから返された場合に限られます。

- main関数から0を返した。
- _exitまたはexit関数のstatus引数として0を渡した。

status_locationの値にかかわらず、この情報は、<wait.h>ヘッダ・ファイルに定義されている、整数式に評価されるマクロを使って定義することができます。以下のマクロの説明では、status_value引数はstatus_location引数がポイントしている整数値と等しい値です。

WIFEXITED(status_value)	正常に終了した子プロセスの状態が返された場合に、ゼロ以外の値に評価される。
WEXITSTATUS(status_value)	WIFEXITED(status_value)の値がゼロ以外である場合、このマクロは、子プロセスが_exitまたはexit関数に渡したstatus引数の下位8ビットか、子プロセスがmain関数から返した値に評価される。
WIFSIGNALED(status_value)	キャッチされなかったシグナルが受信されたために終了した子プロセスの状態が返された場合に、ゼロ以外の値に評価される。
WTERMSIG(status_value)	WIFSIGNALED(status_value)の値がゼロ以外である場合、このマクロは、子プロセスを終了させたシグナルの番号に評価される。
WIFSTOPPED(status_value)	現在停止している子プロセスの状態が返された場合に、ゼロ以外の値に評価される。
WSTOPSIG(status_value)	WIFSTOPPED(status_value)の値がゼロ以外である場合、このマクロは、子プロセスを停止させたシグナルの番号に評価される。
WIFCONTINUED(status_value)	実行を再開した子プロセスの状態が返された場合に、ゼロ以外の値に評価される。

status_locationがポイントする位置に格納された情報が、WUNTRACEDフラグを指定したwait3呼び出しによって格納されていた場合には、以下のいずれかのマクロがゼロ以外の値に評価されます。

- WIFEXITED(*status_value)

- WIFSIGNALED(*status_value)
- WIFSTOPPED(*status_value)
- WIFCONTINUED(*status_value)

status_locationがポイントする位置に格納された情報が、WUNTRACEDフラグを指定しないwait3呼び出しによって格納されていた場合には、以下のいずれかのマクロがゼロ以外の値に評価されます。

- WIFEXITED(*status_value)
- WIFSIGNALED(*status_value)

wait3関数はBSDシステムとの互換性を備えています。resource_usage引数は、<resource.h>ヘッダ・ファイルに定義されている子プロセス用のリソース使用情報を含んだ位置をポイントします。

親プロセスが、すべての子プロセスが終了するのを待たずに終了した場合、残った子プロセスには、initプロセスのプロセスIDに等しい親プロセスIDが割り当てられます。

exit, _exit, およびinitも参照してください。

Return value

0	成功を示します。停止または終了した子プロセスは存在せず、WNOHANGオプションが指定されています。
x	子プロセスのprocess_id。子プロセスの状態が取得可能です。
-1	エラーを示します。errnoは以下のいずれかの値に設定されます。 <ul style="list-style-type: none"> • ECHILD— 待機すべき子プロセスが存在しない。 • EINTR— 呼び出し元プロセスがキャッチしたシグナルの受信によって終了した。 • EFAULT— status_locationまたはresource_usage引数は、プロセスのアドレス空間の外の位置をポイントしている。 • EINVAL— options引数の値が無効である。

wait4

子プロセスが停止または終了するのを待ちます。

Format

```
#include <wait.h>

pid_t wait4 (pid_t process_id, union wait *status_location, int options, struct rusage *resource_usage);
```

Argument

status_location

<wait.h>ヘッダ・ファイルに定義されている，子プロセスの終了状態を含んでいる位置へのポインタ。

OpenVMS Version 7.2 およびそれ以降，_VMS_WAIT マクロを定義してコンパイルした場合，wait4関数はstatus_location引数で指定されたアドレスに，子プロセスのOpenVMS 完了コードを格納するようになりました。

process_id

子プロセスまたは子プロセスのセット。

options

関数の動作を変更するフラグ。これらのフラグの定義は，説明のセクションに示しています。

resource_usage

終了した子プロセスのリソース利用情報を含んでいる構造体の位置。

Description

wait4関数は，要求が完了するまで呼び出し元プロセスを一時停止します。

process_id引数により，呼び出し元プロセスは，次の規則に従って特定の子プロセスのセットの状態を収集することができます。

process_idの値	要求される状態
-1 に等しい	任意の子プロセス。この点で，waitpid関数はwait関数と等価である。
0 よりも大きい	1 子プロセスについて，プロセス ID を指定する。

wait4関数は，このセットの子プロセスの状態のみを返します。

options引数はwait4関数の動作を変更します。options引数のフラグは、ビット論理和を指定することで組み合わせることができます。以下にフラグを示します。

WNOWAIT	status_locationに状態が返されるプロセスを、待機可能状態に置くよう指定する。このプロセスを再び待っても、同じ結果が得られる。
WNOHANG	呼び出し元プロセスが一時停止しないようにする。停止または終了した子プロセスが複数存在する場合には、WNOHANGフラグが指定されなかった場合と同様に、そのうちの1つが選ばれ、waitpid関数はその子プロセスのプロセスIDを返す。終了したプロセスが存在しない(つまり、waitpidが呼び出し元プロセスをWNOHANGフラグなしで一時停止した)場合には、0(ゼロ)が返される。プロセス0を待つことはできないので、この戻り値から混乱が生じることはない。
WUNTRACED	子プロセスがSIGTTIN、SIGTTOU、SIGSTOP、またはSIGTSTOPSIGNALを受信したために、カレント・プロセスの子プロセスが停止したときに、呼び出しに追加情報を返させるように指定する。

wait4関数が、子プロセスの状態が入手可能であるために返った場合には、子プロセスのプロセスIDが返されます。情報は、status_locationがnullでなければ、このポインタがポイントしている位置に格納されます。

status_locationがポイントする位置に格納される値が0(ゼロ)になるのは、状態が以下のいずれかを行った終了した子プロセスから返された場合に限られます。

- main関数から0を返した。
- _exitまたはexit関数のstatus引数として0を渡した。

status_locationの値にかかわらず、この情報は、<wait.h>ヘッダ・ファイルに定義されている、整数式に評価されるマクロを使って定義することができます。以下のマクロの説明では、status_value引数はstatus_location引数がポイントしている整数値と等しい値です。

WIFEXITED(status_value)	正常に終了した子プロセスの状態が返された場合に、ゼロ以外の値に評価される。
WEXITSTATUS(status_value)	WIFEXITED(status_value)の値がゼロ以外である場合、このマクロは、子プロセスがexitまたは_exit関数に渡したstatus引数の下位8ビットが、子プロセスがmain関数から返した値に評価される。
WIFSIGNALED(status_value)	キャッチされなかったシグナルが受信されたために終了した子プロセスの状態が返された場合に、ゼロ以外の値に評価される。
WTERMSIG(status_value)	WIFSIGNALED(status_value)の値がゼロ以外である場合、このマクロは、子プロセスを終了させたシグナルの番号に評価される。
WIFSTOPPED(status_value)	現在停止している子プロセスの状態が返された場合に、ゼロ以外の値に評価される。
WSTOPSIG(status_value)	WIFSTOPPED(status_value)の値がゼロ以外である場合、このマクロは、子プロセスを停止させたシグナルの番号に評価される。
WIFCONTINUED(status_value)	実行を再開した子プロセスの状態が返された場合に、ゼロ以外の値に評価される。

status_locationがポイントする位置に格納された情報が、WUNTRACEDフラグを指定したwait4呼び出しによって格納されていた場合には、以下のいずれかのマクロがゼロ以外の値に評価されます。

- WIFEXITED(*status_value)
- WIFSIGNALED(*status_value)
- WIFSTOPPED(*status_value)
- WIFCONTINUED(*status_value)

status_locationがポイントする位置に格納された情報が、WUNTRACEDフラグを指定しないwait4呼び出しによって格納されていた場合には、以下のいずれかのマクロがゼロ以外の値に評価されます。

- WIFEXITED(*status_value)
- WIFSIGNALED(*status_value)

wait4関数はwait3関数に似ています。ただし、wait4関数はprocess_id引数によって指定された特定の子プロセスを待ちます。resource_usage引数は、<resource.h>ヘッダ・ファイルに定義されている子プロセスのリソース使用情報を含んだ位置をポイントします。

exitと_exitも参照してください。

Return value

0	成功を示します。停止または終了した子プロセスは存在せず、WNOHANGオプションが指定されています。
x	子プロセスのprocess_id。子プロセスの状態が取得可能です。
-1	エラーを示します。errnoは以下のいずれかの値に設定されます。 <ul style="list-style-type: none"> • ECHILD— 待機すべき子プロセスが存在しない。 • EINTR— 呼び出し元プロセスがキャッチしたシグナルの受信によって終了した。 • EFAULT— status_locationまたはresource_usage引数は、プロセスのアドレス空間の外の位置をポイントしている。 • EINVAL— options引数の値が無効である。

waitpid

子プロセスが終了または停止するのを待ちます。

Format

```
#include <wait.h>

pid_t waitpid (pid_t process_id, int *status_location, int options);
```

Argument

`process_id`

子プロセスまたは子プロセスのセット。

`status_location`

<wait.h>ヘッダ・ファイルに定義されている，子プロセスの終了状態を含んでいる位置へのポインタ。

OpenVMS Version 7.2 およびそれ以降，_VMS_WAIT マクロを定義してコンパイルした場合，waitpid関数はstatus_location引数で指定されたアドレスに，子プロセスのOpenVMS 完了コードを格納するようになりました。

`options`

関数の動作を変更するフラグ。これらのフラグの定義は，説明のセクションに示しています。

Description

waitpid関数は，要求が完了するまで呼び出し元プロセスを一時停止します。呼び出し元スレッドのみが一時停止するように再定義を行います。

process_id引数が-1で，options引数が0の場合，waitpid関数はwait関数と同じように動作します。これらの引数が他の値を持つ場合，waitpid関数はこれらの値の指定に従って変更されます。

process_id引数により，呼び出し元プロセスは，次の規則に従って特定の子プロセスのセットの状態を収集することができます。

process_idの値	要求される状態
-1 に等しい	任意の子プロセス。この点で、waitpid関数はwait関数と等価である。
0 よりも大きい	1 子プロセスについて、プロセス ID を指定する。

waitpid関数は、このセットの子プロセスの状態のみを返します。

options引数はwaitpid関数の動作を変更します。options引数のフラグは、ビット論理和を指定することで組み合わせることができます。以下にフラグを示します。

WCONTINUED	process_id引数で指定された子プロセスのうち、実行が再開されてから状態が報告されていないすべての子プロセスの状態を呼び出し元プロセスに返すよう指定する。
WNOWAIT	status_locationに状態が返されるプロセスを、待機可能状態に置くよう指定する。このプロセスを再び待っても、同じ結果が得られる。
WNOHANG	呼び出し元プロセスが一時停止しないようにする。停止または終了した子プロセスが複数存在する場合には、WNOHANGフラグが指定されなかった場合と同様に、そのうちの1つが選ばれ、waitpid関数はその子プロセスのPIDを返す。終了したプロセスが存在しない(つまり、waitpidが呼び出し元プロセスをWNOHANGフラグなしで一時停止した)場合には、0 (ゼロ) が返される。プロセス0を待つことはできないので、この戻り値から混乱が生じることはない。
WUNTRACED	子プロセスがSIGTTIN、SIGTTOU、SIGSTOP、またはSIGTSTOPシグナルを受信したために、カレント・プロセスの子プロセスが停止したときに、呼び出しに追加情報を返させるように指定する。

waitpid関数が、子プロセスの状態が入手可能であるために返った場合には、子プロセスのプロセスIDが返されます。情報は、status_locationがnullでなければ、このポインタがポイントしている位置に格納されます。status_locationがポイントする位置に格納される値が0になるのは、状態が以下のいずれかを行った終了した子プロセスから返された場合に限られます。

- main関数から0を返した。
- _exitまたはexit関数のstatus引数として0を渡した。

status_locationの値にかかわらず、この情報は、<wait.h>ヘッダ・ファイルに定義されている、整数式に評価されるマクロを使って定義することができます。以下のマクロの説明では、status_value引数はstatus_location引数がポイントしている整数値と等しい値です。

WIFEXITED(status_value)	正常に終了した子プロセスの状態が返された場合に、ゼロ以外の値に評価される。
WEXITSTATUS(status_value)	WIFEXITED(status_value)の値がゼロ以外である場合、このマクロは、子プロセスが_exitまたはexit関数に渡したstatus引数の下位8ビットか、子プロセスがmain関数から返した値に評価される。

WIFSIGNALED(status_value)	キャッチされなかったシグナルが受信されたために終了した子プロセスの状態が返された場合に、ゼロ以外の値に評価される。
WTERMSIG(status_value)	WIFSIGNALED(status_value) の値がゼロ以外である場合、このマクロは、子プロセスを終了させたシグナルの番号に評価される。
WIFSTOPPED(status_value)	現在停止している子プロセスの状態が返された場合に、ゼロ以外の値に評価される。
WSTOPSIG(status_value)	WIFSTOPPED(status_value) の値がゼロ以外である場合、このマクロは、子プロセスを停止させたシグナルの番号に評価される。
WIFCONTINUED(status_value)	実行を再開した子プロセスの状態が返された場合に、ゼロ以外の値に評価される。

status_locationがポイントする位置に格納された情報が、WUNTRACEDフラグを指定したwaitpid呼び出しによって格納されていた場合には、以下のいずれかのマクロがゼロ以外の値に評価されます。

- WIFEXITED(*status_value)
- WIFSIGNALED(*status_value)
- WIFSTOPPED(*status_value)
- WIFCONTINUED(*status_value)

status_locationがポイントするバッファに格納された情報が、WUNTRACEDフラグを指定しないwaitpid呼び出しによって格納されていた場合には、以下のいずれかのマクロがゼロ以外の値に評価されます。

- WIFEXITED(*status_value)
- WIFSIGNALED(*status_value)

親プロセスが、すべての子プロセスが終了するのを待たずに終了した場合、残った子プロセスには、init プロセスのプロセス ID に等しい親プロセス ID が割り当てられます。

exit, _exit, およびwaitも参照してください。

Return value

0	成功を示します。WNOHANGオプションが指定されており、停止または終了した子プロセスがなかった場合にも、waitpid関数は値 0 を返します。
-1	<p>エラーを示します。errnoは以下のいずれかの値に設定されます。</p> <ul style="list-style-type: none">• ECHILD— 呼び出し元プロセスは、既存の待たれていない子プロセスを持っていない。process_id 引数によって指定されたプロセスまたはプロセス・グループ ID が存在しない、または呼び出し元プロセスの子プロセスでない。• EINTR— 関数は受信したシグナルによって終了させられた。 waitpid関数が、子プロセスの状態が取得可能であったために返った場合には、子プロセスのプロセス ID が呼び出し元プロセスに返される。呼び出し元プロセスがシグナルをキャッチしたために返った場合には、-1が返される。• EFAULT— status_location 引数は、プロセスのアドレス空間の外の位置をポイントしている。• EINVAL— options 引数の値が無効である。

wrtomb

ワイド文字をそのマルチバイト文字表現に変換します。

Format

```
#include <wchar.h>

size_t wrtomb (char *s, wchar_t wc, mbstate_t *ps);
```

Argument

s

結果として得られるマルチバイト文字へのポインタ。

wc

ワイド文字。

ps

mbstate_tオブジェクトへのポインタ。NULLポインタが指定された場合、関数は内部のmbstate_tオブジェクトを使用します。mbstate_tは、状態依存のコードセットの変換状態を保持することを目的とする不透明のデータ型です。

Description

sがNULLポインタである場合、wrtomb関数は次の呼び出しと等価です。

```
wrtomb (buf, L'\0', ps)
```

bufは内部バッファです。

sがNULLポインタでない場合、wrtomb関数は、wcが指定するワイド文字(シフト・シーケンスを含む)に対応するマルチバイト文字を表現するために必要なバイト数を決定し、結果として得られたバイトを、第1要素がsによってポイントされている配列に格納します。格納されるのは最高MB_CUR_MAXバイトです。

wcがnullワイド文字である場合には、初期シフト状態を復元するために必要なシフト・シーケンスの後にnullバイトが格納されます。結果として記述される状態は、初期変換状態です。

Return value

n	マルチバイト文字を表現するためのシフト・シーケンスを含む、結果の配列に格納されたバイト数。
-1	エンコーディング・エラーを示します。wc引数は有効なワイド文字ではありません。グローバルなerrnoはEILSEQに設定されます。変換状態は未定義です。

wcscat

2 つのワイド文字列を連結します。

Format

```
#include <wchar.h>

wchar_t *wcscat (wchar_t *wstr_1, const wchar_t *wstr_2);
```

関数バリエント

wcscat関数は、それぞれ 32 ビットと 64 ビットのポインタ・サイズで使用するための_wcscat32と_wcscat64という名前のバリエントを持っています。ポインタ・サイズ固有の関数の使用方法については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

Argument

wstr_1, wstr_2
null で終了するワイド文字列へのポインタ。

Description

wcscat関数は、終端の null 文字を含むワイド文字列wstr_2を、wstr_1の末尾に追加します。

wcsncatも参照してください。

Return value

x	連結された結果を保持できるだけの大きさを持つと仮定される第 1 引数wstr_1。
---	---

Example

```

#include <stdlib.h>
#include <stdio.h>
#include <wchar.h>
#include <string.h>

/* This program concatenates two wide-character strings using */
/* the wcscat function, and then manually compares the result */
/* to the expected result */

#define S1LENGTH 10
#define S2LENGTH 8

main()
{
    int i;
    wchar_t s1buf[S1LENGTH + S2LENGTH];
    wchar_t s2buf[S2LENGTH];
    wchar_t test1[S1LENGTH + S2LENGTH];

    /* Initialize the three wide-character strings */

    if (mbstowcs(s1buf, "abcmnxyz", S1LENGTH) == (size_t)-1) {
        perror("mbstowcs");
        exit(EXIT_FAILURE);
    }

    if (mbstowcs(s2buf, " orthis", S2LENGTH) == (size_t)-1) {
        perror("mbstowcs");
        exit(EXIT_FAILURE);
    }

    if (mbstowcs(test1, "abcmnxyz orthis", S1LENGTH + S2LENGTH)
        == (size_t)-1) {
        perror("mbstowcs");
        exit(EXIT_FAILURE);
    }

    /* Concatenate s1buf with s2buf, placing the result */
    /* into * s1buf. Then compare s1buf with the expected */
    /* result in test1. */

    wcscat(s1buf, s2buf);

    for (i = 0; i < S1LENGTH + S2LENGTH - 2; i++) {
        /* Check that each character is correct */
        if (test1[i] != s1buf[i]) {
            printf("Error in wcscat\n");
            exit(EXIT_FAILURE);
        }
    }

    printf("Concatenated string: <%S>\n", s1buf);
}

```

この例のプログラムを実行すると、次の結果が生成されます。

```
Concatenated string: <abcmnxyz orthis>
```

wcschr

指定されたワイド文字列の中でワイド文字をスキャンします。

Format

```
#include <wchar.h>

wchar_t *wcschr (const wchar_t *wstr, wchar_t wc);
```

関数バリエント

wcschr関数は、それぞれ 32 ビットと 64 ビットのポインタ・サイズで使用するための_wcschr32と_wcschr64という名前のバリエントを持っています。ポインタ・サイズ固有の関数の使用方法については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

Argument

wstr
null で終了するワイド文字列へのポインタ。

wc
wchar_t型の文字。

Description

wcschr関数は、null で終了するワイド文字列の中の、指定されたワイド文字の最初のオカレンスのアドレスを返します。終端の null 文字は文字列の一部と見なされます。

wcsrchrも参照してください。

Return value

x	指定されたワイド文字の最初のオカレンスのアドレス。
NULL	ワイド文字が文字列中に存在しないことを示します。

Example

```
#include <stdlib.h>
#include <stdio.h>
#include <wchar.h>
#include <string.h>

#define BUFF_SIZE 50

main()
{
    int i;
    wchar_t slbuf[BUFF_SIZE];
    wchar_t *status;

    /* Initialize the buffer */
    if (mbstowcs(slbuf, "abcdefghijkl lkjihgfedcba", BUFF_SIZE)
        == (size_t)-1) {
        perror("mbstowcs");
        exit(EXIT_FAILURE);
    }

    /* This program checks the wcschr function by incrementally */
    /* going through a string that ascends to the middle and */
    /* then descends towards the end. */
    for (i = 0; (slbuf[i] != '\0') && (slbuf[i] != ' '); i++) {
        status = wcschr(slbuf, slbuf[i]);
        /* Check for pointer to leftmost character - test 1. */
        if (status != &slbuf[i]) {
            printf("Error in wcschr\n");
            exit(EXIT_FAILURE);
        }
    }

    printf("Program completed successfully\n");
}
```

この例のプログラムを実行すると、次の結果が生成されます。

```
Program completed successfully
```

wcscmp

2つのワイド文字列を比較します。文字列が異なるかどうか、どのように異なるかを示す整数を返します。

Format

```
#include <wchar.h>

int wcscmp (const wchar_t *wstr_1, const wchar_t *wstr_2);
```

Argument

wstr_1, wstr_2
null で終了するワイド文字列へのポインタ。

Description

wcscmp関数は、wstr_1の中のワイド文字をwstr_2の中のワイド文字と比較します。文字列が異なる場合、関数は次の値を返します。

- wstr_1の中の最初の異なる文字のコード・ポイントが、wstr_2の中の対応する文字のコード・ポイントよりも小さい場合には、0よりも小さい整数。
- wstr_1の中の最初の異なる文字のコード・ポイントが、wstr_2の中の対応する文字のコード・ポイントよりも大きい場合には、0よりも大きい整数。

ワイド文字列が等しい場合、関数は0を返します。

wscoll関数とは異なり、wcscmp関数は個々のワイド文字の2進値に基づいて文字列を比較します。

wcsncmpも参照してください。

Return value

< 0	wstr_1がwstr_2よりも小さいことを示します。
= 0	wstr_1がwstr_2と等しいことを示します。
> 0	wstr_1がwstr_2よりも大きいことを示します。

wcscoll

2つのワイド文字列を比較し、文字列が異なるかどうか、どのように異なるかを示す整数を返します。この関数は、カレント・ロケールの LC_COLLATE カテゴリの照合情報を使用して、比較の方法を決定します。

Format

```
#include <wchar.h>

int wcscoll (const wchar_t *ws1, const wchar_t *ws2);
```

Argument

ws1, ws2
ワイド文字列へのポインタ。

Description

wcscoll関数は、wcscmp関数とは異なり、2つの文字列をロケールに依存する形で比較します。エラー条件のための値は予約されていないので、アプリケーションは関数呼び出しの前にerrnoを0に設定し、呼び出しの後にこれをテストすることで、エラーをチェックしなくてはなりません。

wcsxfrmも参照してください。

Return value

< 0	ws1がws2よりも小さいことを示します。
0	文字列が等しいことを示します。
> 0	ws1がws2よりも大きいことを示します。

wcscpy

ワイド文字列sourceを、終端の null 文字も含めてdestにコピーします。

Format

```
#include <wchar.h>

wchar_t *wcscpy (wchar_t *dest, const wchar_t *source);
```

関数バリエーション

wcscpy関数は、それぞれ 32 ビットと 64 ビットのポインタ・サイズで使用するための_wcscpy32と_wcscpy64という名前のバリエーションを持っています。ポインタ・サイズ固有の関数の使用方法については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

Argument

dest
null で終了するワイド文字デスティネーション文字列へのポインタ。

source
null で終了するワイド文字ソース文字列へのポインタ。

Description

wcscpy関数はsourceをdestにコピーし、sourceの null 文字をコピーした後に停止します。コピーが 2 つのオーバーラップする文字列の間で行われた場合の動作は未定義です。

wcsncpyも参照してください。

Return value

x sourceのアドレス。

wcscspn

ワイド文字列の中の文字を，ワイド文字のセットと比較します。関数は，ワイド文字のセットに含まれていない文字だけからなる先頭の部分文字列の長さを返します。

Format

```
#include <wchar.h>

size_t wcscspn (const wchar_t *wstr1, const wchar_t *wstr2);
```

Argument

wstr1

null で終了するワイド文字列へのポインタ。これが null 文字列だった場合には 0 が返されます。

wstr2

関数が探すワイド文字のセットを含んでいる，null で終了するワイド文字列へのポインタ。

Description

wcscspn関数は，wstr1がポイントする文字列の中のワイド文字を，wstr2に含まれている文字を検出するまでスキャンします。関数は，wstr2に含まれていない文字のみから構成される，wstr1の先頭のセグメントの長さを返します。

Return value

x

セグメントの長さ。

Example

```

#include <stdlib.h>
#include <stdio.h>
#include <wchar.h>
#include <string.h>

/* This test sets up 2 strings, buffer and w_string, and */
/* then uses wscspn() to calculate the maximum segment */
/* of w_string, which consists entirely of characters */
/* NOT from buffer. */

#define BUFF_SIZE 20
#define STRING_SIZE 50

main()
{
    wchar_t buffer[BUFF_SIZE];
    wchar_t w_string[STRING_SIZE];
    size_t result;

    /* Initialize the buffer */

    if (mbstowcs(buffer, "abcdefg", BUFF_SIZE) == (size_t)-1) {
        perror("mbstowcs");
        exit(EXIT_FAILURE);
    }

    /* Initialize the string */
    if (mbstowcs(w_string, "jklmabcjklabcdehijklmno", STRING_SIZE)
        == (size_t)-1) {
        perror("mbstowcs");
        exit(EXIT_FAILURE);
    }

    /* Using wscspn - work out the largest string in w_string */
    /* which consists entirely of characters NOT from buffer */
    result = wscspn(w_string, buffer);
    printf("Longest segment NOT found in w_string is: %d", result);
}

```

この例のプログラムを実行すると、次の結果が生成されます。

Longest segment NOT found in w_string is: 4

wcsftime

tm構造体に格納されている日付および時刻情報を使用して、ワイド文字出力文字列を作成します。出力文字列の形式は書式文字列によって制御されます。

Format

```
#include <wchar.h>

size_t wcsftime (wchar_t *wcs, size_t maxsize, const char *format, const struct tm *timeptr); (XPG4)
size_t wcsftime (wchar_t *wcs, size_t maxsize, const wchar_t *format, const struct tm *timeptr);
               (ISO C)
```

関数バリエーション

_DECC_V4_SOURCE および_VMS_V6_SOURCE 機能テスト・マクロを定義してコンパイルすると、OpenVMS Version 7.0 より前の動作と等価な、wcsftime関数へのローカル時刻ベースのエントリ・ポイントが使用可能となります。

Argument

wcs

結果として得られるワイド文字列へのポインタ。

maxsize

結果として得られる文字列に格納されるワイド文字の数の最大値。

format

出力文字列の形式を制御する文字列へのポインタ。XPG4 インタフェースでは、この引数は定数文字列へのポインタです。ISO C インターフェースでは、定数ワイド文字列へのポインタです。

timeptr

ローカル時刻構造体へのポインタ。tm構造体は<time.h>ヘッダ・ファイルに定義されています。

Description

wcsftime関数は、timeptrがポイントする構造体の中のデータを使用して、wcsがポイントするワイド文字列を作成します。wcsには最高でmaxsize個のワイド文字がコピーされます。

書式文字列はゼロ個以上の変換指定と通常の文字から構成されます。すべての通常の文字は(終端の null 文字を含めて) 変換なしに出力文字列にコピーされます。変換指定は、tm構造体の中のデータが出力文字列にどのようにフォーマットされるかを定義します。

変換指定はパーセント(%) 文字、1つまたは複数のオプションの文字(表 REF-13 を参照)、および変換指定子(表 REF-14 を参照) から構成されます。表 REF-13 に示しているオプション文字が変換指定に含まれる場合、それらの文字は表に示した順序で現れなくてはなりません。

表 REF-13 wcsftime 変換指定のオプション要素

要素	意味
-	フィールド幅にオプションとして付け、そのフィールドが左揃えされ、スペースでパディングされることを示す。0 要素と同時に使用することはできない。
0	フィールド幅にオプションとして付け、そのフィールドが右揃えされ、ゼロでパディングされることを示す。-要素と同時に使用することはできない。
フィールド幅	最大フィールド幅を指定する 10 進整数。
. 精度	<p>フィールド内のデータの精度を指定する 10 進整数。</p> <p>d, H, I, j, m, M, o, S, U, w, W, y, および Y 変換指定子では、精度指定子は、フィールド内の桁数の最小値である。変換指定が精度によって指定された桁数よりも少ない場合には、先頭にゼロが追加される。</p> <p>a, A, b, B, c, D, E, h, n, N, p, r, t, T, x, X, Z, および % 変換指定子では、精度指定子は、フィールド内のワイド文字数の最大値である。変換指定が精度によって指定された桁数よりも多く文字を含んでいる場合には、右側の文字が切り捨てられる。</p> <p>d, H, I, m, M, o, S, U, w, W, y, および Y 変換指定子のデフォルトの精度は 2 である。j 変換指定子のデフォルトの精度は 3 である。</p>

表 REF-13 の変換指定のオプション要素のリストは、XPG4 仕様の弊社による拡張であることに注意してください。

表 REF-14 は変換指定子を示しています。wcsftime関数は、プログラムの現在のロケールの LC_TIME カテゴリのフィールドから値を取得します。たとえば、%B が指定されている場合、関数は LC_TIME の mon フィールドにアクセスして、tm構造体で指定された月の完全な名前を取得します。無効な変換指定子を使用したときの結果は未定義です。

表 REF-14 wcsftime の変換指定子

指定子	置き換え
a	ロケールの短縮された曜日名。
A	ロケールの完全な曜日名。
b	ロケールの短縮された月の名前。

(次ページに続く)

表 REF-14 (続き) wcsftime の変換指定子

指定子	置き換え
B	ロケールの完全な月の名前。
c	ロケールの適切な日付および時刻表現。
C	10 進数 (00 ~ 99) として表現される世紀 (年を 100 で割り、整数に切り捨て)。
d	その月の 10 進数 (00 ~ 31) として表現される日付。
D	%m/%d/%yと同じ。
e	先頭がスペース文字でフィルされた 2 桁のフィールドに格納される、その月の 10 進数 (1 ~ 31) として表現される日付。
Ec	ロケールの代替日付および時刻表現。
EC	ロケールの代替表現における基本年 (期間) の名前。
Ex	ロケールの代替日付表現。
Ey	ロケールの代替表現における基本年 (%EC) からのオフセット。
EY	ロケールの完全な代替年表現。
h	%bと同じ。
H	10 進数 (00 ~ 23) としての時刻 (24 時間制)。
I	10 進数 (01 ~ 12) としての時刻 (12 時間制)。
j	10 進数 (001 ~ 366) としての、その年の中での日。
m	10 進数 (01 ~ 12) としての月。
M	10 進数 (00 ~ 59) としての分。
n	改行文字。
Od	ロケールの代替数値シンボルを使用した、その月の中での日。
Oe	ロケールの代替数値シンボルを使用した、その月の中での日付。
OH	ロケールの代替数値シンボルを使用した時刻 (24 時間制)。
OI	ロケールの代替数値シンボルを使用した時刻 (12 時間制)。
Om	ロケールの代替数値シンボルを使用した月。
OM	ロケールの代替数値シンボルを使用した分。
OS	ロケールの代替数値シンボルを使用した秒。
Ou	ロケールの代替表現での曜日を数値で表したもの (月曜日=1)。
OU	ロケールの代替数値シンボルを使用した、その年の中での週 (週は日曜日から始まる)。
OV	ロケールの代替数値シンボルを使用した、10 進数 (01 ~ 53) としての、その年の中での週 (週は月曜日から始まる)。1 月 1 日を含んでいる週が、新年に 4 日以上ある場合には、その週が 1 番目の週と見なされる。そうでなければ、前年の 53 番目の週と見なされ、次の週が 1 番目の週となる。
Ow	ロケールの代替数値シンボルを使用した、数値としての曜日 (日曜日=0)。
OW	ロケールの代替数値シンボルを使用した、その年の中での数値としての週 (週は月曜日から始まる)。
Oy	ロケールの代替数値シンボルを使用した、世紀を除いた年。
p	ロケールの 12 時間制における AM/PM 指定。

(次ページに続く)

表 REF-14 (続き) wcsftime の変換指定子

指定子	置き換え
r	AM/PM 表記での時刻。
R	24 時間表記での時刻 (%H:%M)。
S	10 進数 (00 ~ 61) としての秒。
t	タブ文字。
T	時刻 (%H:%M:%S)。
u	1 ~ 7 の範囲の 10 進数としての曜日 (月曜日=1)。
U	10 進数 (00 ~ 53) としての、その年の中での週 (最初の日曜日が 1 番目の週の最初の日と見なされる)。
V	10 進数 (00 ~ 53) としての、その年の中で週 (週は月曜日から始まる)。1 月 1 日を含んでいる週が、新年に 4 日以上ある場合には、その週が 1 番目の週と見なされる。そうでなければ、前年の 53 番目の週と見なされ、次の週が 1 番目の週となる。
w	10 進数 (0 [日曜日] ~ 6) としての曜日。
W	10 進数 (00 ~ 53) としての、その年の中での週 (最初の月曜日が 1 番目の週の最初の日と見なされる)。
x	ロケールの適切な日付表現。
X	ロケールの適切な時刻表現。
Y	10 進数 (00 ~ 99) としての、世紀を除いた年。
Y	10 進数としての、世紀を含んだ年。
Z	タイム・ゾーン名またはその短縮形。タイム・ゾーン情報がない場合には、文字は出力されない。
%	リテラルの%文字。

Return value

x	wcsがポイントする配列に格納されたワイド文字数。終端の null 文字は含まれません。
0	エラーが発生したことを示します。配列の内容は不定です。

Example

```

/* Exercise the wcsftime formatting routine.          */
/* NOTE: the format string is an "L" (or wide character) */
/*      string indicating that this call is NOT in      */
/*      the XPG4 format, but rather in ISO C format.    */

```

```

#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <wchar.h>
#include <locale.h>
#include <errno.h>

#define NUM_OF_DATES 7
#define BUF_SIZE 256

/* This program formats a number of different dates, once using the */
/* C locale and then using the fr_FR.ISO8859-1 locale. Date and time */
/* formatting is done using wcsftime(). */

main()
{
    int count,
        i;
    wchar_t buffer[BUF_SIZE];
    struct tm *tm_ptr;
    time_t time_list[NUM_OF_DATES] =
        {500, 68200000, 694223999,
         694224000, 704900000, 705000000,
         705900000};

    /* Display dates using the C locale */
    printf("\nUsing the C locale:\n\n");
    setlocale(LC_ALL, "C");
    for (i = 0; i < NUM_OF_DATES; i++) {
        /* Convert to a tm structure */
        tm_ptr = localtime(&time_list[i]);

        /* Format the date and time */
        count = wcsftime(buffer, BUF_SIZE, L"Date: %A %d %B %Y\nTime: %T%n\n",
                        tm_ptr);
        if (count == 0) {
            perror("wcsftime");
            exit(EXIT_FAILURE);
        }

        /* Print the result */
        printf("%S", buffer);
    }

    /* Display dates using the fr_FR.ISO8859-1 locale */
    printf("\nUsing the fr_FR.ISO8859-1 locale:\n\n");
    setlocale(LC_ALL, "fr_FR.ISO8859-1");
    for (i = 0; i < NUM_OF_DATES; i++) {
        /* Convert to a tm structure */
        tm_ptr = localtime(&time_list[i]);
    }
}

```

```

        /* Format the date and time */
        count = wcsftime(buffer, BUF_SIZE, L"Date: %A %d %B %Y\nTime: %T\n\n",
                        tm_ptr);
        if (count == 0) {
            perror("wcsftime");
            exit(EXIT_FAILURE);
        }

        /* Print the result */
        printf("%S", buffer);
    }
}

```

この例のプログラムを実行すると、次の結果が生成されます。

Using the C locale:

```

Date: Thursday 01 January 1970
Time: 00:08:20

Date: Tuesday 29 February 1972
Time: 08:26:40

Date: Tuesday 31 December 1991
Time: 23:59:59

Date: Wednesday 01 January 1992
Time: 00:00:00

Date: Sunday 03 May 1992
Time: 13:33:20

Date: Monday 04 May 1992
Time: 17:20:00

Date: Friday 15 May 1992
Time: 03:20:00

```

Using the fr_FR.ISO8859-1 locale:

```

Date: jeudi 01 janvier 1970
Time: 00:08:20

Date: mardi 29 f rier 1972
Time: 08:26:40

Date: mardi 31 d embre 1991
Time: 23:59:59

Date: mercredi 01 janvier 1992
Time: 00:00:00

Date: dimanche 03 mai 1992
Time: 13:33:20

Date: lundi 04 mai 1992
Time: 17:20:00

Date: vendredi 15 mai 1992
Time: 03:20:00

```

wcslen

ワイド文字列の中のワイド文字の数を返します。返される長さには、終端の null 文字は含まれません。

Format

```
#include <wchar.h>

size_t wcslen (const wchar_t *wstr);
```

Argument

wstr
null で終了するワイド文字列へのポインタ。

Return value

x	終端の null ワイド文字を除いたワイド文字列の長さ。
---	------------------------------

wcsncat

1つの文字列の一定数のワイド文字を別の文字列に連結します。

Format

```
#include <wchar.h>

wchar_t *wcsncat (wchar_t *wstr_1, const wchar_t *wstr_2, size_t maxchar);
```

関数バリエント

wcsncat関数は、それぞれ 32 ビットと 64 ビットのポインタ・サイズで使用するための `_wcsncat32` と `_wcsncat64` という名前のバリエントを持っています。ポインタ・サイズ固有の関数の使用方法については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

Argument

`wstr_1`, `wstr_2`

`null` で終了するワイド文字列へのポインタ。

`maxchar`

`wstr_2` から `wstr_1` にコピーされるワイド文字の数の最大値。`maxchar` が 0 の場合、`wstr_2` から文字はコピーされません。

Description

wcsncat関数は、ワイド文字列 `wstr_2` から `wstr_1` の末尾に、最高 `maxchar` 個のワイド文字を追加します。wcsncat関数の結果には、終端の `null` ワイド文字がつけに追加されます。このため、`wstr_1` に格納されるワイド文字の数の最大値は `wcslen(wstr_1) + maxchar + 1` です。

wcscatも参照してください。

Return value

x 連結された結果を保持できるだけの大きさを持つと仮定される第 1 引数 `wstr_1`。

Example

```
#include <stdlib.h>
#include <stdio.h>
#include <wchar.h>
#include <string.h>

/* This program concatenates two wide-character strings using */
/* the wcsncat function, and then manually compares the result */
/* to the expected result */

#define S1LENGTH 10
#define S2LENGTH 8
#define SIZE      3

main()
{
    int i;
    wchar_t s1buf[S1LENGTH + S2LENGTH];
    wchar_t s2buf[S2LENGTH];
    wchar_t test1[S1LENGTH + S2LENGTH];

    /* Initialize the three wide-character strings */

    if (mbstowcs(s1buf, "abcmnxyz", S1LENGTH) == (size_t)-1) {
        perror("mbstowcs");
        exit(EXIT_FAILURE);
    }

    if (mbstowcs(s2buf, " orthis", S2LENGTH) == (size_t)-1) {
        perror("mbstowcs");
        exit(EXIT_FAILURE);
    }

    if (mbstowcs(test1, "abcmnxyz orthis", S1LENGTH + SIZE)
        == (size_t)-1) {
        perror("mbstowcs");
        exit(EXIT_FAILURE);
    }

    /* Concatenate s1buf with SIZE characters from s2buf, */
    /* placing the result into s1buf. Then compare s1buf */
    /* with the expected result in test1. */
    wcsncat(s1buf, s2buf, SIZE);
```

```
for (i = 0; i <= S1LENGTH + SIZE - 2; i++) {  
    /* Check that each character is correct */  
    if (test1[i] != slbuf[i]) {  
        printf("Error in wcnscat\n");  
        exit(EXIT_FAILURE);  
    }  
}  
printf("Concatenated string: <%S>\n", slbuf);  
}
```

この例のプログラムを実行すると、次の結果が生成されます。

```
Concatenated string: <abcmnxyz or>
```

wcsncmp

2つのワイド文字列の中の文字を、最高maxchar個比較します。文字列が異なるかどうか、またどのように異なるかを示す整数を返します。

Format

```
#include <wchar.h>

int wcsncmp (const wchar_t *wstr_1, const wchar_t *wstr_2, size_t maxchar);
```

Argument

wstr_1, wstr_2

nullで終了するワイド文字列へのポインタ。

maxchar

wstr_1とwstr_2の両方で探す文字の数の最大値。maxcharが0の場合、比較は実行されず、0が返されます(文字列は等しいと見なされます)。

Description

文字列はnull文字が検出されるか、文字列に違いが検出されるか、またはmaxcharに達するまで比較されます。文字が異なる場合、wcsncmpは次の値を返します。

- wstr_1の中の最初の異なる文字のコード・ポイントが、wstr_2の中の対応する文字のコード・ポイントよりも小さい場合には、0よりも小さい整数。
- wstr_1の中の最初の異なる文字のコード・ポイントが、wstr_2の中の対応する文字のコード・ポイントよりも大きい場合には、0よりも大きい整数。

maxchar個の文字を比較しても違いが発見されなかった場合、関数は0を返します。

wcscmpも参照してください。

Return value

< 0	wstr_1がwstr_2よりも小さいことを示します。
0	wstr_1がwstr_2と等しいことを示します。
> 0	wstr_1がwstr_2よりも大きいことを示します。

wcsncpy

sourceからdestにワイド文字をコピーします。関数は最高maxchar個の文字をコピーします。

Format

```
#include <wchar.h>

wchar_t *wcsncpy (wchar_t *dest, const wchar_t *source, size_t maxchar);
```

関数バリエント

wcsncpy関数は、それぞれ 32 ビットと 64 ビットのポインタ・サイズで使用するための_wcsncpy32と_wcsncpy64という名前のバリエントを持っています。ポインタ・サイズ固有の関数の使用方法については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

Argument

dest
null で終了するワイド文字デスティネーション文字列へのポインタ。

source
null で終了するワイド文字ソース文字列へのポインタ。

maxchar
sourceからdestにコピーするワイド文字の数の最大値。

Description

wcsncpy関数は、sourceからdestに最高maxchar個の文字をコピーします。sourceに含まれている文字がmaxchar個未満である場合には、maxchar個の文字が書き込まれるまで、destには null 文字が追加されます。

sourceに含まれている文字がmaxchar個以上である場合には、可能な限り多くの文字がdestにコピーされます。sourceの終端の null 文字はdestにはコピーされません。

wscpyも参照してください。

wcsncpy

Return value

x

destのアドレス。

wcspbrk

ワイド文字列の中で、指定されたワイド文字のセットの1つの最初のおカレンスを検索します。

Format

```
#include <wchar.h>

wchar_t *wcspbrk (const wchar_t *wstr, const wchar_t *charset);
```

関数バリエーション

wcspbrk関数は、それぞれ 32 ビットと 64 ビットのポインタ・サイズで使用するための `_wcspbrk32` と `_wcspbrk64` という名前のバリエーションを持っています。ポインタ・サイズ固有の関数の使用方法については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

Argument

wstr
ワイド文字列へのポインタ。これが null 文字列だった場合には、NULL が返されます。

charset
関数が探すワイド文字のセットを含んでいるワイド文字列へのポインタ。

Description

wcspbrk関数は文字列の中のワイド文字をスキャンし、charsetに含まれているワイド文字を検出した時点で停止し、文字列の中の、文字セットに含まれる最初の文字のアドレスを返します。

Return value

x	文字列の中の , セットに含まれる最初のワイド文字のアドレス。
NULL	どの文字もcharsetに含まれていないことを示します。

wcsrchr

指定された文字列の中のワイド文字の最後のオカレンスをスキャンします。

Format

```
#include <wchar.h>

wchar_t *wcsrchr (const wchar_t *wstr, wchar_t wc);
```

関数バリエント

wcsrchr関数は、それぞれ 32 ビットと 64 ビットのポインタ・サイズで使用するための_wcsrchr32と_wcsrchr64という名前のバリエントを持っています。ポインタ・サイズ固有の関数の使用方法については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

Argument

wstr
null で終了するワイド文字列へのポインタ。

wc
wchar_t型の文字。

Description

wcsrchr関数は、null で終了するワイド文字列の中の、指定されたワイド文字の最後のオカレンスのアドレスを返します。終端の null 文字は文字列の一部と見なされません。

wcschrも参照してください。

Return value

x	指定されたワイド文字の最後のオカレンスのアドレス。
NULL	ワイド文字が文字列に含まれていないことを示します。

Example

```

#include <stdlib.h>
#include <stdio.h>
#include <wchar.h>
#include <string.h>

#define BUFF_SIZE 50
#define STRING_SIZE 6

main()
{
    int i;
    wchar_t slbuf[BUFF_SIZE],
            w_string[STRING_SIZE];
    wchar_t *status;
    wchar_t *pbuf = slbuf;

    /* Initialize the buffer */
    if (mbstowcs(slbuf, "hijklabcdefg ytuijklfedcba", BUFF_SIZE)
        == (size_t)-1) {
        perror("mbstowcs");
        exit(EXIT_FAILURE);
    }

    /* Initialize the string to be searched for */

    if (mbstowcs(w_string, "hijkl", STRING_SIZE) == (size_t)-1) {
        perror("mbstowcs");
        exit(EXIT_FAILURE);
    }

    /* This program checks the wcsrchr function by searching for */
    /* the last occurrence of a string in the buffer slbuf and */
    /* prints out the contents of slbuff from the location of */
    /* the string found. */

    status = wcsrchr(slbuf, w_string[0]);
    /* Check for pointer to start of rightmost character string. */
    if (status == pbuf) {
        printf("Error in wcsrchr\n");
        exit(EXIT_FAILURE);
    }

    printf("Program completed successfully\n");
    printf("String found : [%S]\n", status);
}

```

この例のプログラムを実行すると、次の結果が生成されます。

```

Program completed successfully
String found : [hijklfedcba]

```

wcsrtombs

ワイド文字のシーケンスを、対応するマルチバイト文字のシーケンスに変換します。

Format

```
#include <wchar.h>

size_t wcsrtombs (char *dst, const wchar_t **src, size_t len, mbstate_t *ps);
```

関数バリエーション

wcsrtombs関数は、それぞれ 32 ビットと 64 ビットのポインタ・サイズで使用するための_wcsrtombs32と_wcsrtombs64という名前のバリエーションを持っています。ポインタ・サイズ固有の関数の使用方法については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

Argument

dst

変換されたマルチバイト文字シーケンスのためのデスティネーション配列へのポインタ。

src

変換するワイド文字のシーケンスを含んだ配列へのポインタのアドレス。

len

dstがポイントする配列に格納できるバイト数の最大値。

ps

mbstate_tオブジェクトへのポインタ。NULL ポインタが指定された場合、関数は内部のmbstate_tオブジェクトを使用します。mbstate_tは、状態依存のコードセットの変換状態を保持することを目的とする不透明のデータ型です。

Description

wcsrtombs関数は、srcが間接的にポイントする配列の中のワイド文字のシーケンスを、psがポイントするオブジェクトによって記述される変換状態を出発点として、対応するマルチバイト文字のシーケンスに変換します。

dstが NULL ポインタでない場合、変換された文字はdstがポイントする配列に格納されます。変換は終端の null ワイド文字まで続けられ、その null ワイド文字も格納されます。

変換は次の 2 つの場合に停止します。

- 有効なマルチバイト文字に対応しないコードに達した
- dstが NULL ポインタでなく、次のマルチバイト文字を格納したときに、dstがポイントする配列に格納できる合計のlenバイトの上限を超える場合

個々の変換は、wrtomb関数を呼び出したかのように実行されます。

dstが NULL ポインタでない場合、srcがポイントするポインタ・オブジェクトには、NULL ポインタ (終端の null ワイド文字に達したために変換が停止した場合) か、変換された最後のワイド文字の直後のアドレス (存在する場合) が代入されます。終端の null ワイド文字に達したために変換が終了した場合、記述される結果の状態は初期変換状態です。

dstに NULL ポインタを指定して、wcsrtombs関数をカウント関数として呼び出した場合、内部のmbstate_tオブジェクトの値は変更されません。

wrtombも参照してください。

Return value

x	結果として得られる配列に格納されたバイト数。終端の null は (存在していても) 含みません。
-1	エンコーディング・エラーを示します。有効なマルチバイト文字に対応しない文字が検出されました。errnoはEILSEQに設定されます。変換状態は未定義です。

wcsspnp

ワイド文字列の中の文字を、ワイド文字のセットと比較します。関数は、ワイド文字のセットに含まれる文字のみから構成される最初の部分文字列の長さを返します。

Format

```
#include <wchar.h>

size_t wcsspnp (const wchar_t *wstr1, const wchar_t *wstr2);
```

Argument

wstr1

null で終了するワイド文字列へのポインタ。この文字列が null 文字列だった場合には、0 が返されます。

wstr2

関数が探すワイド文字のセットを含んでいる、null で終了するワイド文字列へのポインタ。

Description

wcsspnp関数は、wstr2に含まれない文字を検出するまで、wstr1がポイントするワイド文字列の中でワイド文字をスキャンします。関数は、wstr2に含まれる文字から構成される、wstr1の最初のセグメントの長さを返します。

Return value

x

セグメントの長さ。

Example

```
#include <stdlib.h>
#include <stdio.h>
#include <wchar.h>
#include <string.h>

/* This test sets up 2 strings, buffer and w_string. It */
/* then uses wcssp() to calculate the maximum segment */
/* of w_string that consists entirely of characters */
/* from buffer. */

#define BUFF_SIZE 20
#define STRING_SIZE 50

main()
{
    wchar_t buffer[BUFF_SIZE];
    wchar_t w_string[STRING_SIZE];
    size_t result;

    /* Initialize the buffer */

    if (mbstowcs(buffer, "abcdefg", BUFF_SIZE) == (size_t)-1) {
        perror("mbstowcs");
        exit(EXIT_FAILURE);
    }

    /* Initialize the string */

    if (mbstowcs(w_string, "abcdjklmabcjklabcdehijkl", STRING_SIZE)
        == (size_t)-1) {
        perror("mbstowcs");
        exit(EXIT_FAILURE);
    }

    /* Using wcssp - work out the largest string in w_string */
    /* that consists entirely of characters from buffer */

    result = wcssp(w_string, buffer);
    printf("Longest segment found in w_string is: %d", result);
}
```

この例のプログラムを実行すると、次の結果が生成されます。

```
Longest segment found in w_string is: 5
```

wcsstr

s1がポイントする文字列の中で、s2がポイントする文字列に含まれるワイド文字のシーケンスの最初のオカレンスを探します。

Format

```
#include <wchar.h>

wchar_t *wcsstr (const wchar_t *s1, const wchar_t *s2);
```

関数バリエーション

wcsstr関数は、それぞれ 32 ビットと 64 ビットのポインタ・サイズで使用するための_wcsstr32と_wcsstr64という名前のバリエーションを持っています。ポインタ・サイズ固有の関数の使用方法については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

Argument

s1, s2
null で終了するワイド文字列へのポインタ。

Description

s2が長さ 0 のワイド文字列をポイントしている場合、wcsstr関数はs1を返します。

Return value

x	発見された文字列へのポインタ。
NULL	エラーを示します。文字列は発見されませんでした。

wcstod

指定されたワイド文字列を倍精度の数値に変換します。

Format

```
#include <wchar.h>

double wcstod (const wchar_t *nptr, wchar_t **endptr);
```

Argument

nptr

倍精度の数値に変換するワイド文字列へのポインタ。

endptr

関数が、スキャンを終了させた最初の認識不可能なワイド文字のアドレスを格納できるオブジェクトのアドレス。endptrが NULL ポインタである場合、最初の認識不可能なワイド文字のアドレスは保存されません。

Description

wcstod関数は、オプションとして空白文字 (iswspaceの定義に従う) のシーケンスを、さらにオプションのプラスまたはマイナス記号を、さらにオプションとして基数文字を含んだ数字のシーケンスを、さらにオプションの文字 (e または E) を、最後にオプションの符号付きの整数を認識します。最初の認識不可能な文字が現れた時点で、変換は終了します。

文字列は、浮動小数点定数を解釈するときに使用されるのと同じ規則によって解釈されます。

基数文字は、プログラムの現在のロケール (カテゴリ LC_NUMERIC) によって定義されています。

この関数は、変換後の値を返します。wcstodでは、オーバーフローは次のように処理されます。

- 正しい値がオーバーフローを引き起こす場合には、(値の符号に従ってプラスまたはマイナス記号が付いた) HUGE_VAL が返され、errnoは ERANGE に設定される。

- 正しい値がアンダフローを引き起こす場合には、0 が返され、errnoは EINVAL に設定される。

文字列が認識不可能なワイド文字で始まる場合、*endptrはnptrに設定され、値 0 が返されます。

Return value

x	変換された文字列。
0	変換が実行できなかったことを示します。errnoは以下のいずれかに設定されます。 <ul style="list-style-type: none">• EINVAL— 変換は実行できなかった。• ERANGE— 値はアンダフローを引き起こす。• ENOMEM— 内部変換バッファ用のメモリが足りなかった。
±HUGE_VAL	オーバーフローが発生しました。errnoは ERANGE に設定されます。

wcstok

指定されたワイド文字列の中でテキスト・トークンを探します。

Format

```
#include <wchar.h>

wchar_t *wcstok (wchar_t *ws1, const wchar_t *ws2); (XPG4)

wchar_t *wcstok (wchar_t *ws1, const wchar_t *ws2, wchar_t **ptr); (ISO C)
```

関数バリエント

wcstok関数は、それぞれ 32 ビットと 64 ビットのポインタ・サイズで使用するための _wcstok32と _wcstok64という名前のバリエントを持っています。ポインタ・サイズ固有の関数の使用方法については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

Argument

ws1

ゼロ個以上のテキスト・トークンを含んだワイド文字列へのポインタ。

ws2

1 つ以上のワイド文字から構成される区切り文字列へのポインタ。区切り文字列は呼び出しごとに異なっても構いません。

ptr

ISO C 標準のみ。ws1が NULL の場合にのみ使用されます。ptrは、wcstokが同じワイド文字列のスキャンを続行するために必要な情報を格納する、呼び出し元が提供するwchar_tポインタです。

Description

wcstokの一連の呼び出しにより、ws1がポイントするワイド文字列を、ws2がポイントするワイド文字列に含まれているワイド文字で区切られたトークンのシーケンスに分割することができます。

wcstok関数は、次の呼び出しがあるまでのワイド文字列中の自分の位置を保持しており、その後の呼び出しがあったときに、前の呼び出しで識別されたテキスト・トークンの次のテキスト・トークンを識別してワイド文字列の中を移動します。

ws1の中のトークンは、wcstokがws1に挿入する null 文字によって区切られます。このため、ws1はconstオブジェクトであってはなりません。

以下のセクションでは、wcstokの XPG4 標準およびISO C標準インタフェースの違いを説明します。

XPG4 標準の動作

wcstok関数の最初の呼び出しでは、ワイド文字列の中で、ws2がポイントする区切り文字列に含まれていない最初の文字が検索されます。最初の呼び出しは、最初のトークンの最初のワイド文字へのポインタを返し、ws1の中で、返されたトークンの直後に null ワイド文字を書き込みます。

wcstokのそれ以降の呼び出しでは、ws2がポイントする区切り文字列に含まれているワイド文字が検索されます。個々の呼び出しで (第 1 引数として値 NULL を指定)、最初にws1がポイントしていた文字列の中の次のトークンへのポインタが返されます。文字列にトークンが残っていなかった場合、wcstokは NULL ポインタを返します。

ISO C 標準の動作

シーケンスの中の最初の呼び出しでは、ws1はワイド文字列をポイントしています。同じ文字列に対するそれ以降の呼び出しでは、ws1は NULL です。ws1が NULL の場合、ptrがポイントする値は、同じワイド文字列に対するそれ以前の呼び出しによって格納された値と一致します。それ以外の場合、ptrがポイントする値は無視されます。

シーケンスの中の最初の呼び出しは、ws1がポイントするワイド文字列の中で、ws2がポイントしている現在の区切りワイド文字列に含まれていない最初のワイド文字を検索します。そのようなワイド文字が見つからなかった場合には、ws1がポイントするワイド文字列にはそのようなトークンが含まれていないということになり、wcstokは NULL ポインタを返します。

その後、wcstok関数は、現在の区切りワイド文字列に含まれているワイド文字を探します。そのようなワイド文字が見つからなかった場合には、現在のトークンはws1がポイントするワイド文字列の終わりまで続いていることになり、同じワイド文字列に対するそれ以降のトークンの検索は NULL ポインタを返します。そのようなワイド文字が見つかった場合には、null ワイド文字で上書きされ、現在のトークンは終了します。

どのケースでも、wcstokは、ptrがポイントするポインタに必要な情報を格納し、ws1に対して NULL ポインタ、ptrに対して変更なしのポインタ値を指定したそれ以降の呼び出しが、null ワイド文字 (存在する場合) で上書きされた要素の直後から検索を開始するようにします。

Return value

<code>x</code>	トークンの最初の文字へのポインタ。
<code>NULL</code>	トークンが見つからなかったことを示します。

値

1. `/* XPG4 version of wcstok call */`

```
#include <wchar.h>
#include <string.h>
#include <stdio.h>

main()
{
    wchar_t str[] = L"...ab..cd,,ef.hi";

    printf("%S\n", wcstok(str, L"."));
    printf("%S\n", wcstok(NULL, L","));
    printf("%S\n", wcstok(NULL, L",."));
    printf("%S\n", wcstok(NULL, L",."));
}
```
2. `/* ISO C version of wcstok call */`

```
#include <wchar.h>
#include <string.h>
#include <stdio.h>

main()
{
    wchar_t str[] = L"...ab..cd,,ef.hi";
    wchar_t *savgptr = NULL;

    printf("%S\n", wcstok(str, L".", &savgptr));
    printf("%S\n", wcstok(NULL, L",", &savgptr));
    printf("%S\n", wcstok(NULL, L",.", &savgptr));
    printf("%S\n", wcstok(NULL, L",.", &savgptr));
}
```

この例のプログラムを実行すると、次の結果が生成されます。

```
$ $ RUN WCSTOK_EXAMPLE
|ab|
|.cd|
|ef|
|hi|
$
```

wcstol

指定された底のワイド文字列を long の整数値に変換します。

Format

```
#include <wchar.h>

long int wcstol (const wchar_t *nptr, wchar_t **endptr, int base);
```

関数バリエント

wcstol関数は、それぞれ 32 ビットと 64 ビットのポインタ・サイズで使用するための _wcstol32と _wcstol64という名前のバリエントを持っています。ポインタ・サイズ固有の関数の使用方法については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

Argument

nptr

long整数に変換するワイド文字列へのポインタ。

endptr

関数が変換プロセスの中で検出した最初の認識不可能な文字へのポインタを格納できるオブジェクトのアドレス (変換する文字列の中の、最後に処理された文字の直後の文字)。endptrが NULL ポインタである場合、最初の認識不可能な文字のアドレスは保存されません。

base

変換の底として使用する 2 ~ 36 の値。

baseが 16 の場合、オプションの符号の後の先頭のゼロは無視され、0x と 0X も無視されます。

baseが 0 の場合、文字のシーケンスは、整数を解釈するのに使用されるのと同じ規則で解釈されます。オプションの符号の後には、以下の文字を指定できます。

- 先頭の 0 は 8 進変換を示す。
- 先頭の 0x または 0X は 16 進変換を示す。
- その他の先頭の文字の組み合わせは 10 進変換を示す。

Description

wcstol関数は、底の値に応じて、さまざまな形式の文字列を認識します。この関数は、指定された文字列に含まれる先頭の空白文字 (isspace関数の定義に従う) を無視します。オプションのプラスまたはマイナス記号と、それに続く、底の値に応じた整数を表現できる数字または英字のシーケンスを認識します。最初の認識不可能な文字が検出された時点で、変換は終了します。

Return value

x	変換された値。
0	文字列が認識不可能なワイド文字で始まるか、baseの値が無効であることを示します。文字列が認識不可能なワイド文字で始まる場合、*endptrはnptrに設定されます。関数はerrnoを EINVAL に設定します。
LONG_MAX または LONG_MIN	変換された値が、それぞれ正または負のオーバーフローを引き起こすことを示します。関数はerrnoを ERANGE に設定します。

wcstombs

ワイド文字コードのシーケンスをマルチバイト文字のシーケンスに変換します。

Format

```
#include <stdlib.h>

size_t wcstombs (char *s, const wchar_t *pwcs, size_t n);
```

Argument

s
結果として得られるマルチバイト文字が格納される配列へのポインタ。

pwcs
ワイド文字コードのシーケンスを含んでいる配列へのポインタ。

n
sがポイントする配列に格納されるバイト数の最大値。

Description

wcstombs関数は、pwcsがポイントする配列の中のマルチバイト文字に対応するコードのシーケンスをマルチバイト文字のシーケンスに変換し、最高nバイトまで、sがポイントする配列に格納します。返される値は、変換された文字数か、エラーが発生した場合には-1です。

この関数は、プログラムの現在のロケールの LC_CTYPE カテゴリの影響を受けません。

sが NULL の場合、この関数呼び出しはカウント操作になり、nは無視されます。

wctombも参照してください。

Return value

<code>x</code>	sに格納されたバイト数。終端の <code>null</code> バイトは含まれません。sが <code>NULL</code> の場合、 <code>wcstombs</code> はマルチバイト文字配列に必要なバイト数を返します。
<code>(size_t) -1</code>	エラーが発生したことを示します。関数は <code>errno</code> を、無効な文字シーケンス、すなわちワイド文字コードが有効な文字に対応していないことを示す <code>EILSEQ</code> に設定します。

wcstoul

nptrがポイントするワイド文字列の先頭の部分をunsigned long整数に変換します。

Format

```
#include <wchar.h>

unsigned long int wcstoul (const wchar_t *nptr, wchar_t **endptr, int base);
```

関数バリエーション

wcstoul関数は、それぞれ 32 ビットと 64 ビットのポインタ・サイズで使用するための_wcstoul32と_wcstoul64という名前のバリエーションを持っています。ポインタ・サイズ固有の関数の使用方法については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

Argument

nptr
unsigned longに変換するワイド文字列へのポインタ。

endptr
関数が変換プロセスの中で検出した最初の認識不可能な文字のアドレスを格納できるオブジェクトのアドレス (変換する文字列の中の、最後に処理された文字の直後の文字)。endptrが NULL ポインタである場合、最初の認識不可能な文字のアドレスは保存されません。

base
変換の底として使用する 2 ~ 36 の値。

baseが 16 の場合、オプションの符号の後の先頭のゼロは無視され、0x と 0X も無視されます。

baseが 0 の場合、文字のシーケンスは、整数を解釈するのに使用されるのと同じ規則で解釈されます。オプションの符号の後では、先頭の 0 は 8 進変換を示し、先頭の 0x または 0X は 16 進変換を示し、その他の先頭の文字の組み合わせは 10 進変換を示します。

Description

wcstoul関数は、底の値に応じて、さまざまな形式の文字列を認識します。この関数は、指定された文字列に含まれる先頭の空白文字 (isspace関数の定義に従う) を無視します。オプションのプラスまたはマイナス記号と、それに続く、底の値に応じた整数を表現できる数字または英字のシーケンスを認識します。最初の認識不可能な文字が検出された時点で、変換は終了します。

Return value

x	変換された値。
0	文字列が認識不可能なワイド文字で始まるか、baseの値が無効であることを示します。文字列が認識不可能なワイド文字で始まる場合、*endptrはnptrに設定されます。関数はerrnoを EINVAL に設定します。
ULONG_MAX	変換された値がオーバーフローを引き起こすことを示します。関数はerrnoを ERANGE に設定します。

Example

```
#include <stdlib.h>
#include <stdio.h>
#include <wchar.h>
#include <errno.h>
#include <limits.h>

/* This test calls wcstoul() to convert a string to an
 * unsigned long integer. wcstoul outputs the resulting
 * integer and any characters that could not be converted. */

#define MAX_STRING 128

main()
{
    int base = 10,
        errno;
    char *input_string = "1234.56";
    wchar_t string_array[MAX_STRING],
        *ptr;
    size_t size;
    unsigned long int val;
    printf("base = [%d]\n", base);
    printf("String to convert = %s\n", input_string);
    if ((size = mbstowcs(string_array, input_string, MAX_STRING)) ==
        (size_t)-1) {
```

```

        perror("mbstowcs");
        exit(EXIT_FAILURE);
    }
    printf("wchar_t string is = [%S]\n", string_array);

    errno = 0;
    val = wcstoul(string_array, &ptr, base);
    if (errno == 0) {
        printf("returned unsigned long int from wcstoul = [%u]\n", val);
        printf("wide char terminating scan(ptr) = [%S]\n\n", ptr);
    }
    if (errno == ERANGE) {
        perror("error value is :");
        printf("ULONG_MAX = [%u]\n", ULONG_MAX);
        printf("wcstoul failed, val = [%d]\n\n", val);
    }
}

```

この例のプログラムを実行すると、次の結果が生成されます。

```

base = [10]
String to convert = 1234.56
wchar_t string is = [1234.56]
returned unsigned long int from wcstoul = [1234]
wide char terminating scan(ptr) = [.56]

```

WCSWCS

wstr1がポイントする文字列の中で、wstr2がポイントする文字列に含まれるワイド文字のシーケンスの最初のカレンスを探します。

Format

```
#include <wchar.h>

wchar_t *wcswcs (const wchar_t *wstr1, const wchar_t *wstr2);
```

関数バリエーション

WCSWCS関数は、それぞれ 32 ビットと 64 ビットのポインタ・サイズで使用するための_wcswcs32と_wcswcs64という名前のバリエーションを持っています。ポインタ・サイズ固有の関数の使用方法については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

Argument

wstr1, wstr2
null で終了するワイド文字列へのポインタ。

Return value

ポインタ	発見されたワイド文字列へのポインタ。
NULL	ワイド文字列が発見されなかったことを示します。

Example

```
#include <stdlib.h>
#include <stdio.h>
#include <wchar.h>

/* This test uses wcs wcs() to find the occurrence of each */
/* subwide-character string, string1 and string2, within */
/* the main wide-character string, lookin. */

#define BUF_SIZE 50
```

```

main()
{
    static char lookin[] = "that this is a test was at the end";
    char string1[] = "this",
        string2[] = "the end";
    wchar_t buffer[BUF_SIZE],
        input_buffer[BUF_SIZE];

    /* Convert lookin to wide-character format.          */
    /* Buffer and print it out.                          */

    if (mbstowcs(buffer, lookin, BUF_SIZE) == (size_t)-1) {
        perror("mbstowcs");
        exit(EXIT_FAILURE);
    }

    printf("Buffer to look in: %S\n", buffer);

    /* Convert string1 to wide-character format and use */
    /* wcswcs() to locate it within buffer              */

    if (mbstowcs(input_buffer, string1, BUF_SIZE) == (size_t)-1) {
        perror("mbstowcs");
        exit(EXIT_FAILURE);
    }

    printf("this: %S\n", wcswcs(buffer, input_buffer));

    /* Convert string2 to wide-character format and use */
    /* wcswcs() to locate it within buffer              */

    if (mbstowcs(input_buffer, string2, BUF_SIZE) == (size_t)-1) {
        perror("mbstowcs");
        exit(EXIT_FAILURE);
    }

    printf("the end: %S\n", wcswcs(buffer, input_buffer));
    exit(1);
}

```

この例のプログラムを実行すると、次の結果が生成されます。

```

Buffer to look in: that this is a test was at the end
this: this is a test was at the end
the end: the end

```

wcswidth

ディスプレイ・デバイス上の、ワイド文字列の表示に必要なプリント位置の数を決定します。

Format

```
#include <wchar.h>

int wcswidth (const wchar_t *pwcs, size_t n);
```

Argument

pwcs
ワイド文字列へのポインタ。

n
文字列の中の文字数の最大値。

Description

wcswidth関数は、pwcsがポイントする文字列の最初のn文字を表示するために必要なプリント位置の数を返します。文字列に含まれるワイド文字がn個未満の場合、関数は文字列全体に必要な位置の数を返します。

Return value

x	必要なプリント位置の数。
0	pwcsは null 文字です。
-1	pwcsがポイントする文字列の中の 1 つ (またはそれ以上) のワイド文字が、プリント可能な文字でないことを示します。

wcsxfrm

ワイド文字列を変更して、変更後の文字列を `wcscmp` 関数に渡したときに、変更なしの文字列を `wscoll` 関数に渡したときと同じ結果が得られるようにします。

Format

```
#include <wchar.h>

size_t wcsxfrm (wchar_t *ws1, const wchar_t *ws2, size_t maxchar);
```

Argument

`ws1`, `ws2`
ワイド文字列へのポインタ。

`maxchar`
`s1`に格納できるワイド文字の数の最大値。終端の `null` ワイド文字を含みます。

Description

`wcsxfrm`関数は、`ws2`がポイントする文字列を変換し、その結果の文字列を`ws1`がポイントする配列に格納します。`ws1`がポイントする配列に格納されるワイド文字の数は、終端の `null` ワイド文字を含めて、`maxchar`個以下です。

`maxchar`の値が、(終端の `null` を含めて) 変換後の文字列を格納するのに必要なサイズよりも小さかった場合、`ws1`がポイントする配列の内容は不定となります。この場合、関数は変換後の文字列のサイズを返します。

`maxchar`が 0 の場合、`ws1`は `NULL` ポインタであってよく、関数は変換を行う前に、`ws1`配列の必要なサイズを返します。

ワイド文字列比較関数の `wscoll` と `wcscmp` は、同じ 2 つのワイド文字列を与えられたときに、異なる結果を生成することがあります。これは、`wcscmp` が文字列中の文字のコード・ポイント値の直接の比較を行うのに対し、`wscoll` が比較の実行にロケール情報を使用するためです。ロケールによっては、`wscoll` の比較はマルチパスの操作になることがあり、`wcscmp` よりも遅くなります。

`wcsxfrm`関数は、`wcscmp`関数に 2 つの変換後の文字列を渡したときに、元の 2 つの文字列を `wscoll` 関数に渡したときと同じ結果が得られるような形でワイド文字列を変換します。`wcsxfrm`関数は、`wscoll`を使って同じワイド文字列に対する多数の比較を

行う必要があるアプリケーションで便利です。この場合には、(ロケールによっては) `wcsxfrm`関数を使って文字列を変換した後に、`wscmp`関数を使って比較を行う方が効率的である場合があります。

Return value

<code>x</code>	結果として得られる、 <code>ws1</code> がポイントする文字列の長さ。終端の <code>null</code> 文字は含みません。
<code>(size_t) -1</code>	エラーが発生したことを示します。関数は <code>errno</code> を <code>EINVAL</code> に設定します。 <code>ws2</code> がポイントする文字列は、照合シーケンスの領域の外にある文字を含みます。

Example

```
#include <wchar.h>
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>

/* This program verifies that two transformed strings, */
/* when passed through wcsxfrm and then compared, provide */
/* the same result as if passed through wscoll without */
/* any transformation. */

#define BUFF_SIZE 20

main()
{
    wchar_t w_string1[BUFF_SIZE];
    wchar_t w_string2[BUFF_SIZE];
    wchar_t w_string3[BUFF_SIZE];
    wchar_t w_string4[BUFF_SIZE];
    int errno;
    int coll_result;
    int wscmp_result;
    size_t wcsxfrm_result1;
    size_t wcsxfrm_result2;

    /* setlocale to French locale */
    if (setlocale(LC_ALL, "fr_FR.ISO8859-1") == NULL) {
        perror("setlocale");
        exit(EXIT_FAILURE);
    }

    /* Convert each of the strings into wide-character format. */

    if (mbstowcs(w_string1, "<a>bcd", BUFF_SIZE) == (size_t)-1) {
        perror("mbstowcs");
        exit(EXIT_FAILURE);
    }
}
```



```

if (mbstowcs(w_string2, "abcz", BUFF_SIZE) == (size_t)-1) {
    perror("mbstowcs");
    exit(EXIT_FAILURE);
}

/* Collate string 1 and string 2 and store the result. */
errno = 0;
coll_result = wcscoll(w_string1, w_string2);
if (errno) {
    perror("wcscoll");
    exit(EXIT_FAILURE);
}
else {
    /* Transform the strings (using wcsxfrm) into */
    /* w_string3 and w_string4. */
    wcsxfrm_result1 = wcsxfrm(w_string3, w_string1, BUFF_SIZE);
    if (wcsxfrm_result1 == ((size_t) - 1))
        perror("wcsxfrm");
    else if (wcsxfrm_result1 > BUFF_SIZE) {
        perror("\n** String is too long **\n");
        exit(EXIT_FAILURE);
    }
    else {
        wcsxfrm_result2 = wcsxfrm(w_string4, w_string2, BUFF_SIZE);
        if (wcsxfrm_result2 == ((size_t) - 1)) {
            perror("wcsxfrm");
            exit(EXIT_FAILURE);
        }
        else if (wcsxfrm_result2 > BUFF_SIZE) {
            perror("\n** String is too long **\n");
            exit(EXIT_FAILURE);
        }
    }

    /* Compare the two transformed strings and verify that */
    /* the result is the same as the result from wcscoll on */
    /* the original strings. */
}

```

```

else {
    wcscmp_result = wcscmp(w_string3, w_string4);
    if (wcscmp_result == 0 && (coll_result == 0)) {
        printf("\nReturn value from wcscoll() and return value"
               " from wcscmp() are both zero.");
        printf("\nThe program was successful\n\n");
    }
    else if ((wcscmp_result < 0) && (coll_result < 0)) {
        printf("\nReturn value from wcscoll() and return value"
               " from wcscmp() are less than zero.");
        printf("\nThe program was successful\n\n");
    }
    else if ((wcscmp_result > 0) && (coll_result > 0)) {
        printf("\nReturn value from wcscoll() and return value"
               " from wcscmp() are greater than zero.");
        printf("\nThe program was successful\n\n");
    }
    else {
        printf("** Error **\n");
        printf("\nReturn values are not of the same type");
    }
}
}
}
}
}

```

この例のプログラムを実行すると、次の結果が生成されます。

```

Return value from wcscoll() and return value
    from wcscmp() are less than zero.
The program was successful

```

wctob

ワイド文字がシングルバイトのマルチバイト文字に対応するかどうかを判定し、そのマルチバイト文字表現を返します。

Format

```
#include <stdio.h>
#include <wchar.h>
int wctob (wint_t c);
```

引数

c
シングルバイトのマルチバイト文字に変換するワイド文字。

Description

wctob関数は、指定されたワイド文字が初期シフト状態にあるときにシングルバイトのマルチバイト文字に対応するかどうかを判定し、そうであればそのマルチバイト文字表現を返します。

Return value

x	指定されたワイド文字のシングルバイト表現。
EOF	エラーを示します。指定されたワイド文字は、シングルバイトのマルチバイト文字に対応していません。

wctomb

ワイド文字をマルチバイト文字表現に変換します。

Format

```
#include <stdlib.h>

int wctomb (char *s, wchar_t wchar);
```

Argument

s
結果として得られるマルチバイト文字へのポインタ。

wchar
ワイド文字のコード。

Description

wctomb関数は、wcharによって指定されたワイド文字を、そのマルチバイト文字表現に変換します。sが NULL の場合には、0 が返されます。それ以外の場合には、マルチバイト文字を構成するバイト数が返されます。sがポイントする配列オブジェクトには、最高で MB_CUR_MAX バイトが格納されます。

この関数は、プログラムの現在のロケールの LC_CTYPE カテゴリの影響を受けません。

Return value

x	wcharに対応するマルチバイト文字を構成するバイト数。
0	sは NULL です。
-1	wcharは有効な文字ではありません。

wctrans

後にtowctransの呼び出しに使用できる、指定されたプロパティに対応するマッピングの記述を返します。

Format

```
#include <wctype.h>

wctrans_t wctrans (const char *property);
```

引数

property

マッピングの名前。以下のプロパティ名は、すべてのロケールに対して定義されています。

- "toupper"
- "tolower"

現在のロケールの LC_CTYPE カテゴリに、その他のプロパティ名が定義されていることもあります。

Description

wctrans関数は、property引数によって識別されたワイド文字の間のマッピングを記述する、wctrans_t型の値を作成します。

towctransも参照してください。

Return value

ゼロ以外

現在のプログラム・ロケールの LC_CTYPE カテゴリに従い、property 引数として指定された文字列は、既存の文字マッピングの名前です。返された値は、towctrans関数の呼び出しに使用することができます。

0

エラーを示します。property 引数は、現在のプログラムのロケールの文字マッピングを識別していません。

wctype

文字クラスの定義に使用されます。この関数から返される値は、`iswctype`関数の呼び出しで使用されます。

Format

```
#include <wctype.h> (ISO C)
#include <wchar.h> (XPG4)
wctype_t wctype (const char *char_class);
```

引数

`char_class`
有効な文字クラス名へのポインタ。

Description

`wctype`関数は、現在のロケールに対して定義されている有効な文字クラスを、`wctype_t`型のオブジェクトに変換します。以下の文字クラスは、すべてのロケールに対して定義されています。

<code>alnum</code>	<code>cntrl</code>	<code>lower</code>	<code>space</code>
<code>alpha</code>	<code>digit</code>	<code>print</code>	<code>upper</code>
<code>blank</code>	<code>graph</code>	<code>punct</code>	<code>xdigit</code>

現在のロケールの `LC_CTYPE` カテゴリに、その他の文字クラスが定義されていることもあります。

`iswctype`も参照してください。

Return value

<code>x</code>	<code>iswctype</code> 関数の呼び出しに使用できる <code>wctype_t</code> 型のオブジェクト。
<code>0</code>	文字クラス名が現在のロケールに対して有効ではありません。

Example

```

#include <locale.h>
#include <wchar.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>

/* This test will set up a number of character class using wctype() */
/* and then verify whether calls to iswctype() using these classes */
/* produce the same results as calls to the is**** routines.      */

main()
{
    wchar_t w_char;
    wctype_t ret_val;
    char *character = "A";

    /* Convert character to wide character format - w_char */
    if (mbtowc(&w_char, character, 1) == -1) {
        perror("mbtowc");
        exit(EXIT_FAILURE);
    }

    /* Check if results from iswalnum() matches check on */
    /* alnum character class                               */
    if ((iswalnum((wint_t) w_char) &&
         (iswctype((wint_t) w_char, wctype("alnum"))))
        printf("[%C] is a member of the character class alnum\n", w_char);
    else
        printf("[%C] is not a member of the character class alnum\n", w_char);

    /* Check if results from iswalpha() matches check on */
    /* alpha character class                               */
    if ((iswalpha((wint_t) w_char) &&
         (iswctype((wint_t) w_char, wctype("alpha"))))
        printf("[%C] is a member of the character class alpha\n", w_char);
    else
        printf("[%C] is not a member of the character class alpha\n", w_char);

    /* Check if results from iswcntrl() matches check on */
    /* cntrl character class                               */
    if ((iswcntrl((wint_t) w_char) &&
         (iswctype((wint_t) w_char, wctype("cntrl"))))
        printf("[%C] is a member of the character class cntrl\n", w_char);
    else
        printf("[%C] is not a member of the character class cntrl\n", w_char);

    /* Check if results from iswdigit() matches check on */
    /* digit character class                               */

```

```

if ((iswdigit((wint_t) w_char) &&
    (iswctype((wint_t) w_char, wctype("digit"))))
    printf("[%C] is a member of the character class digit\n", w_char);
else
    printf("[%C] is not a member of the character class digit\n", w_char);
/* Check if results from iswgraph() matches check on */
/* graph character class */
if ((iswgraph((wint_t) w_char) &&
    (iswctype((wint_t) w_char, wctype("graph"))))
    printf("[%C] is a member of the character class graph\n", w_char);
else
    printf("[%C] is not a member of the character class graph\n", w_char);
/* Check if results from iswlower() matches check on */
/* lower character class */
if ((iswlower((wint_t) w_char) &&
    (iswctype((wint_t) w_char, wctype("lower"))))
    printf("[%C] is a member of the character class lower\n", w_char);
else
    printf("[%C] is not a member of the character class lower\n", w_char);
/* Check if results from iswprint() matches check on */
/* print character class */
if ((iswprint((wint_t) w_char) &&
    (iswctype((wint_t) w_char, wctype("print"))))
    printf("[%C] is a member of the character class print\n", w_char);
else
    printf("[%C] is not a member of the character class print\n", w_char);
/* Check if results from iswpunct() matches check on */
/* punct character class */
if ((iswpunct((wint_t) w_char) &&
    (iswctype((wint_t) w_char, wctype("punct"))))
    printf("[%C] is a member of the character class punct\n", w_char);
else
    printf("[%C] is not a member of the character class punct\n", w_char);
/* Check if results from iswspace() matches check on */
/* space character class */
if ((iswspace((wint_t) w_char) &&
    (iswctype((wint_t) w_char, wctype("space"))))
    printf("[%C] is a member of the character class space\n", w_char);
else
    printf("[%C] is not a member of the character class space\n", w_char);
/* Check if results from iswupper() matches check on */
/* upper character class */
if ((iswupper((wint_t) w_char) &&
    (iswctype((wint_t) w_char, wctype("upper"))))
    printf("[%C] is a member of the character class upper\n", w_char);
else
    printf("[%C] is not a member of the character class upper\n", w_char);

```



```

/* Check if results from iswxdigit() matches check on */
/* xdigit character class */
if ((iswxdigit((wint_t) w_char)) &&
    (iswctype((wint_t) w_char, wctype("xdigit"))))
    printf("[%C] is a member of the character class xdigit\n", w_char);
else
    printf("[%C] is not a member of the character class xdigit\n", w_char);
}

```

この例のプログラムを実行すると、次の結果が生成されます。

```

[A] is a member of the character class alnum
[A] is a member of the character class alpha
[A] is not a member of the character class cntrl
[A] is not a member of the character class digit
[A] is a member of the character class graph
[A] is not a member of the character class lower
[A] is a member of the character class print
[A] is not a member of the character class punct
[A] is not a member of the character class space
[A] is a member of the character class upper
[A] is a member of the character class xdigit

```

wcwidth

ディスプレイ・デバイス上の、指定されたワイド文字の表示に必要なプリント位置の数を決定します。

Format

```
#include <wchar.h>

int wcwidth (wchar_t wc);
```

引数

wc
ワイド文字。

Description

`wcwidth`関数は、指定されたワイド文字`wc`に必要なカラム位置の数を決定します。`wc`の値は、現在のロケールにおける有効なワイド文字でなくてはなりません。

Return value

<code>x</code>	<code>wc</code> に必要なプリント位置の数。
<code>0</code>	<code>wc</code> は <code>null</code> 文字です。
<code>-1</code>	<code>wc</code> が有効でプリント可能なワイド文字を表していないことを示します。

wmemchr

ワイド文字の配列の中で、指定されたワイド文字の最初のオカレンスを探します。

Format

```
#include <wchar.h>

wchar_t wmemchr (const wchar_t *s, wchar_t c, size_t n);
```

関数バリエーション

wmemchr関数は、それぞれ 32 ビットと 64 ビットのポインタ・サイズで使用するための_wmemchr32と_wmemchr64という名前のバリエーションを持っています。ポインタ・サイズ固有の関数の使用方法については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

Argument

s
検索の対象とするワイド文字の配列へのポインタ。

c
検索するワイド文字値。

n
検索の対象とする配列内のワイド文字の数の最大値。

Description

wmemchr関数は、sがポイントする配列の最初のn個のワイド文字の中で、指定されたワイド文字の最初のオカレンスを探します。

Return value

x	配列中のワイド文字の最初のオカレンスへのポインタ。
NULL	指定されたワイド文字は配列中に存在しません。

wmemcmp

2つのワイド文字の配列を比較します。

Format

```
#include <wchar.h>

int wmemcmp (const wchar_t *s1, const wchar_t *s2, size_t n);
```

Argument

s1, s2
ワイド文字配列へのポインタ。

n
比較するワイド文字の数の最大値。

Description

wmemcmp関数は、s1がポイントする配列の最初のn個のワイド文字を、s2がポイントする配列の最初のn個のワイド文字と比較します。ワイド文字の比較は、ロケール依存の照合規則に従って行われるのではなく、wchar_t型の整数オブジェクトとして行われます。

Return value

0	配列が等しいことを示します。
正の値	第1の配列が第2の配列よりも大きいことを示します。
負の値	第1の配列が第2の配列よりも小さいことを示します。

REF-965

wmemmove

指定された数のワイド文字を、1つのワイド文字配列から別のワイド文字配列にコピーします。

Format

```
#include <wchar.h>

wchar_t wmemmove (wchar_t *dest, const wchar_t *source, size_t n);
```

関数バリエーション

wmemmove関数は、それぞれ 32 ビットと 64 ビットのポインタ・サイズで使用するための `_wmemmove32` と `_wmemmove64` という名前のバリエーションを持っています。ポインタ・サイズ固有の関数の使用方法については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

Argument

dest
デスティネーション配列へのポインタ。

source
ソース配列へのポインタ。

n
移動するワイド文字の数。

Description

wmemmove関数は、sourceがポイントする位置のn個のワイド文字を、destがポイントする位置にコピーします。

wmemmoveルーチンとwmemcpyルーチンは同じ機能を実行しますが、wmemmoveは、2つの配列がオーバーラップする場合でも、ソース配列の内容を確実にデスティネーション配列にコピーします。このようなオーバーラップが生じる可能性がある場合、移植性を必要とするプログラムはwmemcpyではなくwmemmoveを使用するようにしてください。

Return value

x

destの値。

wmemset

ワイド文字の配列の中の指定された数のワイド文字に、指定された値を設定します。

Format

```
#include <wchar.h>

wchar_t wmemset (wchar_t *s, wchar_t c, size_t n);
```

関数バリエーション

wmemset関数は、それぞれ 32 ビットと 64 ビットのポインタ・サイズで使用するための_wmemset32と_wmemset64という名前のバリエーションを持っています。ポインタ・サイズ固有の関数の使用方法については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

Argument

S
ワイド文字の配列へのポインタ。

C
配列の先頭のn個のワイド文字に格納する値。

n
指定された値cに設定するワイド文字の数。

Description

wmemset関数は、sがポイントする配列の先頭のn個のワイド文字のそれぞれに、cの値をコピーします。

Return value

x sの値。

wprintf

標準出力 (stdout) に対して書式付きの出力を行います。書式指定子については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 2 章を参照してください。

Format

```
#include <wchar.h>

int wprintf (const wchar_t *format, ...);
```

Argument

format

書式指定を含んだワイド文字列へのポインタ。書式指定および変換指定とそれぞれに対応する引数の詳細については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 2 章を参照してください。

...

結果として得られる型が、書式指定で与えられた変換指定に対応するオプションの式。

変換指定が与えられなかった場合、出力ソースは省略することができます。そうでなければ、関数呼び出しは変換指定と同じ数の出力ソースを持っていなくてはならず、変換指定は出力ソースの型と一致していなくてはなりません。

変換指定は、左から右の順序で出力ソースと照合されます。余分な出力ポインタが存在する場合には、無視されます。

Description

wprintf関数は、その引数の前にstdout引数を指定したfwprintf関数と等価です。

Return value

n

負の値

書き込まれたワイド文字の数。

エラーを示します。関数はerrnoを以下のいずれかに設定します。

- EILSEQ— 無効な文字を検出した。
- EINVAL— 引数が足りなかった。
- ENOMEM— 変換のために利用できるメモリが足りなかった。
- ERANGE— 浮動小数点計算のオーバーフロー。
- EVMSERR— 変換不可能な OpenVMS エラー。vaxc\$errnoに OpenVMS エラー・コードが含まれている。オーバーフローのために数値への変換が失敗した可能性がある。

また、関数は I/O サブシステムから返されたエラーの結果として、errnoを以下の値に設定することもあります。

- EBADF— ファイル記述子が有効でない。
- EIO— I/O エラー。
- ENOSPC— ファイルを含んでいるデバイス上に空きスペースがない。
- ENXIO— デバイスが存在しない。
- EPIPE— パイプが破壊されている。
- ESPIPE— 追加用にオープンされたファイル内での不正なシーク。
- EVMSERR— 変換不可能な OpenVMS エラー。vaxc\$errnoに OpenVMS エラー・コードが含まれている。等価な C エラー・コードがない I/O エラーが発生した可能性がある。

wrapok

UNIX システム環境で、ウィンドウの右端のワードを次の行の先頭にラップできるようにします。このルーチンは UNIX ソフトウェアとの互換性のためにのみ用意されているもので、OpenVMS 環境では何の機能も持ちません。

Format

```
#include <curses.h>
wrapok (WINDOW *win, bool boolf);
```

Argument

win

ウィンドウへのポインタ。

boolf

論理型の TRUE または FALSE の値。boolf が FALSE の場合、スクロールは許容されません。これはデフォルトの設定です。bool型は、<curses.h>ヘッダ・ファイルに次のように定義されています。

```
#define bool int
```

write

バッファ内の指定された数のバイトをファイルに書き込みます。

Format

```
#include <unistd.h>

ssize_t write (int file_desc, void *buffer, size_t nbytes); (ISO POSIX-1)

int write (int file_desc, void *buffer, int nbytes); (Compatibility)
```

Argument

file_desc

現在、書き込みまたは更新用にオープンされているファイルを参照するファイル記述子。

buffer

出力データの取得元となる、連続した記憶域のアドレス。

nbytes

書き込みを行うバイト数の最大値。

Description

writeがRMS レコード・ファイルに対するもので、バッファに埋め込みの改行文字が含まれている場合には、複数のレコードがファイルに書き込まれます。埋め込みの改行文字がない場合でも、nbytesがファイルの最大レコード・サイズよりも大きければ、ファイルには複数のレコードが書き込まれます。write関数は、必ず少なくとも1つのレコードを生成します。

writeがメールボックスに対するもので、第3引数のnbytesが長さ0を指定している場合には、メールボックスにファイルの終端メッセージが書き込まれます。これはアプリケーションがSY\$CREMBXを使って作成したメールボックスの場合であり、POSIX パイプを実装するために作成されたメールボックスは例外となります。詳細については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル(上巻)』第5章を参照してください。

Return value

x	書き込まれたバイト数。
-1	未定義のファイル記述子，不正なバッファ・アドレス，物理 I/O エラーなどのエラーを示します。

writev

ファイルに書き込みを行います。

Format

```
#include <uio.h>

ssize_t writev (int file_desc, const struct iovec *iov, int iovcnt);

ssize_t __writev64 (int file_desc, const struct __iovec64 *iov, int iovcnt); (Alpha, I64)
```

関数バリエーション

writev関数は、それぞれ 32 ビットと 64 ビットのポインタ・サイズで使用するための `_writev32` と `__writev64` という名前のバリエーションを持っています。ポインタ・サイズ固有の関数の使用方法については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 1.10 節を参照してください。

Argument

file_desc

現在、書き込みまたは更新用にオープンされているファイルを参照するファイル記述子。

iov

出力データの取得元となる `iovec` 構造体の配列。

iovcnt

iov 配列のメンバによって指定されるバッファの数。

Description

writev関数はwriteと同じ機能を持ちますが、出力データをiov配列のメンバ (iov[0], iov[1], ..., iov[iovcnt-1]) で指定されるiovcnt個のバッファから取得します。iovcnt引数は、0 よりも大きく、<limits.h>に定義されている{IOV_MAX}以下である場合に有効となります。

個々のiovecエントリは、データの取得元となるメモリ内の領域のベース・アドレスと長さを指定します。writev関数は、その領域全体を書き込んだ後に、次の領域に進みます。

filedesが通常のファイルを参照しており、iovがポイントする配列内のすべてのiov_lenメンバが0だった場合、writevは0を返し、何の効果も持ちません。

その他のファイル・タイプでの動作は定められていません。

iov_len値の合計がSSIZE_MAXよりも大きい場合、操作は失敗し、データは転送されません。

実行に成功すると、writevは実際に書き込まれたバイト数を返します。それ以外の場合は値-1を返し、ファイル・ポインタは変更されず、errnoはエラーを示す値に設定されます。

Return value	
x	書き込まれたバイト数。
-1	エラーを示します。ファイルの時刻は変更されず、関数はerrnoを以下のいずれかの値に設定します。 <ul style="list-style-type: none">• EBADF—file_desc引数が、書き込み用にオープンされた有効なファイル記述子でない。• EINTR—書き込み操作はシグナルを受信したために終了し、データは転送されなかった。• EINVAL—iov配列の中のiov_len値の合計がssize_tのオーバーフローを引き起こす、またはiocvcnt引数が0以下、あるいは{IOV_MAX}よりも大きかった。• EIO—物理 I/O エラーが発生した。• ENOSPC—ファイルを含んでいるデバイス上に空きスペースがない。• EPIPE—どのプロセスの読み込み用にもオープンされていない、または片側だけがオープンされているパイプまたはFIFOに書き込もうとした。スレッドにはSIGPIPEシグナルも送信される。

wscanf

ワイド文字書式文字列の制御下で、標準入力 (stdin) から入力を読み込みます。

Format

```
#include <wchar.h>

int wscanf (const wchar_t *format, ... );
```

Argument

format

書式指定を含んだワイド文字列へのポインタ。書式指定および変換指定とそれぞれに対応する引数の詳細については、『HP C ランタイム・ライブラリ・リファレンス・マニュアル (上巻)』第 2 章を参照してください。

...

結果として得られる型が、書式指定で与えられた変換指定に対応するオプションの式。

変換指定が与えられなかった場合、入力ポインタは省略することができます。そうでなければ、関数呼び出しは変換指定と同じ数の入力ポインタを持っていないとならず、変換指定は入力ポインタの型と一致していません。

変換指定は、左から右の順序で入力ソースと照合されます。余分な入力ポインタが存在する場合には、無視されます。

Description

wscanf関数は、その引数の前にstdin引数を指定したfwscanf関数と等価です。

Return value

n	代入が行われた入力項目の数。この数は、早い段階で照合に失敗した場合には、指定された数よりも少なかったり、ときには0になることがあります。
EOF	エラーを示します。変換が行われる前に入力エラーが発生しました。

y0, y1, yn (*Alpha, I64*)

第 2 種ベッセル関数を計算します。

Format

```
#include <math.h>
double y0 (double x);
float y0f (float x);
long double y0l (long double x);
double y1 (double x);
float y1f (float x);
long double y1l (long double x);
double yn (int n, double x);
float ynf (int n, float x);
long double ynl (int n, long double x);
```

Argument

x
正の実数値。

n
整数。

Description

y0関数は、0 次の第 2 種ベッセル関数の値を返します。

y1関数は、1 次の第 2 種ベッセル関数の値を返します。

yn関数は、n 次の第 2 種ベッセル関数の値を返します。

Return value

x	xの第 2 種ベッセル関数の値。
−HUGE_VAL	x引数が 0.0 です。errnoは ERANGE に設定されます。
NaN	x引数が負の値または NaN です。errnoは EDOM に設定されます。
0	アンダフローが発生しました。errnoは ERANGE に設定されます。
HUGE_VAL	オーバフローが発生しました。errnoは ERANGE に設定されます。

A

a64l関数	REF-3
abort関数	REF-5
abs関数	REF-6
access関数	REF-7
acosh関数	REF-10
acos関数	REF-9
addch関数	REF-11
addstr関数	REF-12
alarm関数	REF-13
asctime_r関数	REF-15
asctime関数	REF-15
asinh関数	REF-18
asin関数	REF-17
assert関数	REF-19
AST リエントラント	REF-147
atan2関数	REF-22
atanh関数	REF-24
atan関数	REF-21
atexit関数	REF-25
atof関数	REF-27
atoi関数	REF-28
atoll関数	REF-29
atol関数	REF-28
atof関数	REF-29

B

basename関数	REF-30
bcmp関数	REF-32
bcopy関数	REF-33
box関数	REF-34
brk関数	REF-35
bsearch関数	REF-37
btowc関数	REF-40
bzero関数	REF-41

C

cabs関数	REF-42
cacosh関数	REF-44
acos関数	REF-43
calloc関数	REF-45, REF-63
carg関数	REF-46
casinh関数	REF-48
casin関数	REF-47

catanh関数	REF-50
catan関数	REF-49
catclose関数	REF-51
catgets関数	REF-52
catopen関数	REF-55
cbirt関数	REF-58
ccosh関数	REF-60
ccos関数	REF-59
ceil関数	REF-61
cexp関数	REF-62
cfree関数	REF-63
chdir関数	REF-64
chmod関数	REF-66
chown関数	REF-68
cimag関数	REF-69
clearerr_unlocked関数	REF-72
clearerr関数	REF-71
clearok関数	REF-73
clear関数	REF-70, REF-73
clock_getres関数	REF-75
clock_gettime関数	REF-77
clock_settime関数	REF-78
clock関数	REF-74
clog関数	REF-80
closedir関数	REF-83
close関数	REF-81
clrattr関数	REF-86
clrtoebot関数	REF-87
clrtoeol関数	REF-88
confstr関数	REF-89
conj関数	REF-91
copysign関数	REF-92
cosh関数	REF-94
cos関数	REF-93
cot関数	REF-95
cpow関数	REF-96
cproj関数	REF-97
creal関数	REF-98
creat関数	REF-99, REF-175, REF-209, REF-219
crmode関数	REF-106
C RTL 新機能	xxii
crypt関数	REF-108
csinh関数	REF-111
csin関数	REF-110
csqrt関数	REF-112

ctanh関数	REF-114
ctan関数	REF-113
ctermid関数	REF-115
ctime_r関数	REF-116
ctime関数	REF-15, REF-116
tzset関数との組み合わせ	REF-833
Curses 関数	
box	REF-34
clearok	REF-73
delwin	REF-163
endwin	REF-183
getyx	REF-362
initscr	REF-383
leaveok	REF-430
longname	REF-447
mv[w]addch	REF-505
mv[w]addstr	REF-507
mv[w]delch	REF-510
mv[w]getch	REF-511
mv[w]getstr	REF-512
mv[w]inch	REF-513
mv[w]insch	REF-514
mv[w]insstr	REF-515
mvcur	REF-509
mvwin	REF-516
newwin	REF-519
[no]crmode	REF-106
[no]echo	REF-176
[no]nl	REF-525
[no]raw	REF-585
overlay	REF-537
overwrite	REF-538
scroll	REF-618
scrollok	REF-619
subwin	REF-789
touchwin	REF-821
[w]addch	REF-11
[w]addstr	REF-12
[w]clear	REF-70
[w]clrattr	REF-86
[w]clrtobot	REF-87
[w]clrtoeol	REF-88
[w]delch	REF-159
[w]deleteln	REF-162
[w]erase	REF-186
[w]getch	REF-303
[w]getstr	REF-355
[w]inch	REF-381
[w]insch	REF-386
[w]insertln	REF-387
[w]insstr	REF-388
[w]move	REF-496
[w]printw	REF-561
[w]refresh	REF-600
[w]scanw	REF-616
[w]setattr	REF-623
[w]standend	REF-717
[w]standout	REF-718

Curses 関数 (続き)

wrapok	REF-971
cuserid関数	REF-118

D

DECC\$ALLOW_UNPRIVILEGED_NICE 機能論理名	
名	REF-522
DECC\$CRTL_INIT関数	REF-119, REF-840
decc\$feature_get_index機能設定ルーチン	REF-122
decc\$feature_get_name機能設定ルーチン	REF-123
decc\$feature_get_value機能設定ルーチン	REF-124
decc\$feature_get機能設定ルーチン	REF-120
decc\$feature_set_value機能設定ルーチン	REF-128
decc\$feature_set機能設定ルーチン	REF-126
decc\$feature_show_all機能設定ルーチン	REF-131
decc\$feature_show機能設定ルーチン	REF-130
decc\$fix_timeファイル指定変換ルーチン	REF-132
decc\$from_vmsファイル指定変換ルーチン	REF-134
decc\$match_wildファイル指定変換ルーチン	REF-136
DECC\$PIPE_BUFFER_QUOTA 機能論理名	
名	REF-546
DECC\$PIPE_BUFFER_SIZE 機能論理名	
名	REF-546
DECC\$POPEN_NO_CRLF_REC_ATTR 機能論理名	
名	REF-548
decc\$record_read関数	REF-138
decc\$record_write関数	REF-139
decc\$set_child_default_dir関数	REF-140
decc\$set_child_standard_streams関数	REF-142
decc\$set_reentrancy関数	REF-147
DECC\$STREAM_PIPE 機能論理名	
名	REF-548
decc\$validate_wchar関数	REF-154
decc\$write_eof_to_mbx関数	REF-156
decc\$to_vmsファイル指定変換ルーチン	REF-149
decc\$translate_vmsファイル指定変換ルーチン	REF-152
delch関数	REF-159
deleteln関数	REF-162
delete関数	REF-160, REF-603
delwin関数	REF-163
difftime関数	REF-164
dirname関数	REF-165
div関数	REF-167
dlclose関数	REF-168
dlopen関数	REF-169

dlopen関数 REF-170
 dlsym関数 REF-172
 drand48関数 REF-173
 lcong48関数との組み合わせ REF-427
 seed48関数との組み合わせ REF-620
 srand48関数との組み合わせ REF-711
 dup2関数 REF-175, REF-219, REF-548
 dup関数 REF-175, REF-219

E

echo関数 REF-176
 ecvt関数 REF-177
 encrypt関数 REF-179
 endgrent関数 REF-181
 endpwent関数 REF-182
 endwin関数 REF-183
 erand48関数 REF-184
 erase関数 REF-186
 erf関数 REF-187
 execl関数 REF-190
 execlp関数 REF-192
 execl関数 REF-188
 execve関数 REF-194
 execvp関数 REF-196
 execv関数 REF-193
 exec関数 REF-547
 exit関数 REF-197
 _exit関数 REF-197
 _exp2関数 REF-201
 exp関数 REF-199

F

fabs関数 REF-203
 fchmod関数 REF-204
 fchown関数 REF-205
 fclose関数 REF-207, REF-246
 fcntl関数 REF-209
 fcvt関数 REF-216
 fdim関数 REF-218
 fdopen関数 REF-219, REF-548
 feof_unlocked関数 REF-221
 feof関数 REF-220
 ferror_unlocked関数 REF-223
 ferror関数 REF-222
 fflush関数 REF-224
 popen関数との組み合わせ REF-554
 ffs関数 REF-225
 fgetc_unlocked関数 REF-227
 fgetc関数 REF-226
 fgetname関数 REF-228
 fgetpos関数 REF-230
 fgets関数 REF-232
 fgetwc関数 REF-234
 fgetws関数 REF-235
 fileno関数 REF-237

finite関数 REF-238
 flockfile関数 REF-239
 floor関数 REF-240
 fmax関数 REF-242
 fma関数 REF-241
 fmin関数 REF-243
 fmod関数 REF-244
 fopen関数 REF-245
 fork関数 REF-866
 fp_classf関数 REF-247
 fp_classl関数 REF-247
 fp_class関数 REF-247
 fpathconf関数 REF-249
 fprintf関数 REF-251
 fputc_unlocked関数 REF-254
 fputc関数 REF-253
 fputs関数 REF-255
 fputwc関数 REF-256
 fputws関数 REF-258
 fread関数 REF-259
 free関数 REF-63, REF-261, REF-308
 tempnam関数との組み合わせ REF-810
 freopen関数 REF-262
 frexp関数 REF-264
 fscanf関数 REF-266
 fseeko関数 REF-270
 fseek関数 REF-268, REF-843, REF-844
 fsetpos関数 REF-271
 fstatvfs関数 REF-276
 fstat関数 REF-272
 fsync関数 REF-278
 ftello関数 REF-280
 ftell関数 REF-279
 ftime関数 REF-281
 ftruncate関数 REF-283
 ftrylockfile関数 REF-284
 ftw関数 REF-285
 funlockfile関数 REF-288
 fwait関数 REF-289
 fwide関数 REF-290
 fwprintf関数 REF-292
 fwrite関数 REF-295
 fwscanf関数 REF-297

G

gcvt関数 REF-299
 getc_unlocked関数 REF-302
 getchar_unlocked関数 REF-305
 getchar関数 REF-304
 getch関数 REF-303
 getclock関数 REF-306
 getcwd関数 REF-308
 getc関数 REF-301
 getdtablesize関数 REF-310
 getegid関数 REF-311
 getenv関数 REF-313
 putenv関数との組み合わせ REF-569

geteuid関数 REF-315
 getgid関数 REF-317
 getgrent関数 REF-319
 getgrgid r関数 REF-323
 getgrgid関数 REF-321
 getgrnam r関数 REF-327
 getgrnam関数 REF-325
 getgroup関数 REF-329
 getitimer関数 REF-331
 getlogin関数 REF-333
 getname関数 REF-334, REF-547
 getopt関数 REF-336
 getpagesize関数 REF-340
 getpgid関数 REF-341
 getpgrp関数 REF-342
 getpid関数 REF-343
 getppid関数 REF-344
 getpwent関数 REF-345
 getpwnam r関数 REF-347
 getpwnam関数 REF-347
 getpwuid r関数 REF-350
 getpwuid関数 REF-350
 getsid関数 REF-354
 getstr関数 REF-355
 gets関数 REF-232, REF-353
 gettimeofday関数 REF-356
 getuid関数 REF-357
 getwchar関数 REF-361
 getwc関数 REF-360
 getw関数 REF-359
 getyx関数 REF-362
 globfree関数 REF-368
 glob関数 REF-363
 gmtime r関数 REF-369
 gmtime関数 REF-369
 gsignal関数 REF-371

H

hypot関数 REF-373

I

iconv_close関数 REF-376
 iconv_open関数 REF-377
 iconv関数 REF-374
 ilogb関数 REF-380
 inch関数 REF-381
 index関数 REF-382
 initscr関数 REF-383
 initstate関数 REF-384
 setstate関数 REF-649
 insch関数 REF-386
 insertln関数 REF-387
 insstr関数 REF-388
 isalnum関数 REF-389
 isalpha関数 REF-390

isapipe関数 REF-391
 isascii関数 REF-392
 isatty関数 REF-393
 iscntrl関数 REF-394
 isdigit関数 REF-395
 isgraph関数 REF-396
 islower関数 REF-397
 isnan関数 REF-398
 isprint関数 REF-399
 ispunct関数 REF-400
 isspace関数 REF-401
 isupper関数 REF-402
 iswalnum関数 REF-403
 iswalpha関数 REF-404
 iswcntrl関数 REF-405
 iswctype関数 REF-406
 iswdigit関数 REF-408
 iswgraph関数 REF-409
 iswlower関数 REF-410
 iswprint関数 REF-411
 iswpunct関数 REF-412
 iswspace関数 REF-413
 iswupper関数 REF-414
 iswxdigit関数 REF-415
 isxdigit関数 REF-416
 itimerval構造体 REF-331, REF-631

J

j0関数 REF-417
 j1関数 REF-417
 jn関数 REF-417
 jrand48関数 REF-419

K

kill関数 REF-421

L

l64a関数 REF-423
 labs関数 REF-425
 lchown関数 REF-426
 lcong48関数 REF-427
 drand48関数との組み合わせ REF-173
 lrand48関数との組み合わせ REF-448
 mrnd48関数との組み合わせ REF-499
 ldexp関数 REF-428
 ldiv関数 REF-429
 leaveok関数 REF-430
 lgamma関数 REF-431
 LIB\$ESTABLISH関数 REF-858
 link関数 REF-432
 llabs関数 REF-577
 lldiv関数 REF-578
 localeconv関数 REF-434
 localtime_r関数 REF-438

localtime関数 REF-438
 tzset関数との組み合わせ REF-833
 log10関数 REF-441
 loglp関数 REF-443
 logb関数 REF-444
 log関数 REF-441
 longjmp関数 REF-445, REF-858, REF-866
 longjmpメンバ
 ftw関数との組み合わせ REF-286
 longname関数 REF-447
 lrand48関数 REF-448
 lcong48関数との組み合わせ REF-427
 seed48関数との組み合わせ REF-620
 srand48関数との組み合わせ REF-711
 lrint関数 REF-450
 lround関数 REF-451
 lseek関数 REF-452
 lstat関数 REF-454
 lwait関数 REF-455

M

main関数
 wait3関数との組み合わせ REF-891
 wait4関数との組み合わせ REF-894
 waitpid関数との組み合わせ REF-897
 malloc関数 REF-63, REF-456
 ftw関数との組み合わせ REF-286
 putenv関数との組み合わせ REF-569
 mblen関数 REF-458
 mbrlen関数 REF-459
 mbrtowc関数 REF-461
 mbsinit関数 REF-467
 mbsrtowcs関数 REF-468
 mbstate_t REF-459, REF-461, REF-467,
 REF-468, REF-900, REF-931
 mbstowcs関数 REF-463
 mbtowc関数 REF-465
 memcpy関数 REF-470
 memchr関数 REF-472
 memcmp関数 REF-474
 memcpy関数 REF-475
 memmove関数 REF-477
 memset関数 REF-479
 mkdir関数 REF-481
 mkstemp関数 REF-485
 mktemp関数 REF-486
 mktime関数 REF-487
 tzset関数との組み合わせ REF-833
 mmap関数 REF-489
 modf関数 REF-495
 move関数 REF-496
 mprotect関数 REF-497
 mrand48関数 REF-499
 lcong48関数との組み合わせ REF-427
 seed48関数との組み合わせ REF-620
 srand48関数との組み合わせ REF-711

msync関数 REF-501
 MULTITHREAD リエントラント REF-147
 munmap関数 REF-503
 mvaddch関数 REF-505
 mvaddstr関数 REF-507
 mvcur関数 REF-509
 mvdelch関数 REF-510
 mvgetch関数 REF-511
 mvgetstr関数 REF-512
 mvinch関数 REF-513
 mvinsch関数 REF-514
 mvinsstr関数 REF-515
 mvwaddch関数 REF-505
 mvwaddstr関数 REF-507
 mvwdelch関数 REF-510
 mvwgetch関数 REF-511
 mvwgetstr関数 REF-512
 mvwinch関数 REF-513
 mvwinsch関数 REF-514
 mvwinsstr関数 REF-515
 mvwin関数 REF-516

N

nanosleep関数 REF-517
 newwin関数 REF-519
 nextafter関数 REF-520
 nexttoward関数 REF-521
 nice関数 REF-522
 nint関数 REF-524
 nl_langinfo関数 REF-526
 nl関数 REF-525
 nocrmode関数 REF-106
 noecho関数 REF-176
 NONE リエントラント REF-147
 nonl関数 REF-525
 noraw関数 REF-585
 nrand48関数 REF-530

O

opendir関数 REF-535
 readdir関数との組み合わせ REF-590
 rewinddir関数との組み合わせ REF-608
 open関数 REF-175, REF-209, REF-219,
 REF-532
 overlay関数 REF-34, REF-537
 overwrite関数 REF-34, REF-538

P

passwd構造体 REF-345, REF-348, REF-351
 pathconf関数 REF-539
 pause関数 REF-541
 pclose関数 REF-542
 popen関数との組み合わせ REF-554
 perror関数 REF-543

pipe関数 REF-175, REF-209, REF-219,
REF-545
poll関数 REF-550
popen関数 REF-554
pow関数 REF-556
pread関数 REF-558
printf関数 REF-560
printw関数 REF-561
putc_unlocked関数 REF-565
putchar_unlocked関数 REF-568
putchar関数 REF-567
putc関数 REF-563
putenv関数 REF-569
puts関数 REF-571
putwchar関数 REF-574
putwc関数 REF-573
putw関数 REF-572
pwrite関数 REF-575

Q

qabs関数 REF-577
qdiv関数 REF-578
qsort関数 REF-579

R

raise関数 REF-421, REF-581
random関数 REF-584
rand関数 REF-583
raw関数 REF-585
readdir_r関数 REF-589
readdir関数 REF-589
closedir関数との組み合わせ REF-83
readlink関数 REF-591
readv関数 REF-593
read関数 REF-587
realloc関数 REF-596
realpath関数 REF-598
refresh関数 REF-73, REF-600
remainder関数 REF-601, REF-602
remove関数 REF-160, REF-603
remquo関数 REF-601, REF-602
rename関数 REF-605
rewinddir関数
readdir関数との組み合わせ REF-590
rewind関数 REF-607
rindex関数 REF-609
rint関数 REF-610
rmdir関数 REF-611

S

sbrk関数 REF-612
scalb関数 REF-614
scanf関数 REF-615
scanw関数 REF-616

scrollok関数 REF-619
scroll関数 REF-618
seed48関数 REF-620
drand48関数との組み合わせ REF-173
lcong48関数との組み合わせ REF-427
lrand48関数との組み合わせ REF-448
mrand48関数との組み合わせ REF-499
seekdir関数 REF-622
setattr関数 REF-623
setbuf関数 REF-624
setenv関数 REF-626
seteuid関数 REF-627
setgid関数 REF-628
setgrent関数 REF-630
setitimer関数 REF-631
ualarm関数との組み合わせ REF-838
setjmp関数 REF-445, REF-633, REF-858,
REF-866
setkey関数 REF-635
setlocale関数 REF-636
setpgid関数 REF-640
setpgrp関数 REF-642
setpwent関数 REF-643
setregid関数 REF-644
setreuid関数 REF-646
setsid関数 REF-648
setstate関数 REF-649
initstate関数との組み合わせ REF-384
setuid関数 REF-651
setvbuf関数 REF-653
popen関数との組み合わせ REF-554
shm_open関数 REF-656
shm_unlink関数 REF-659
sigaction関数 REF-661
sigaction構造体 REF-662
sigaddset関数 REF-665
sigblock関数 REF-667, REF-681
sigdelset関数 REF-668
sigemptyset関数 REF-669
sigfillset関数 REF-671
sighold関数 REF-672
sigignore関数 REF-674
sigismember関数 REF-676
siglongjmp関数 REF-677
sigmask関数 REF-678
signal関数 REF-371, REF-581, REF-679,
REF-715
sigpause関数 REF-681
sigpending関数 REF-682
sigprocmask関数 REF-683
sigrelse関数 REF-685
sigsetjmp関数 REF-687
sigsetmask関数 REF-689
sigstack関数 REF-690
sigsuspend関数 REF-692
sigtimedwait関数 REF-694
sigvec関数 REF-371, REF-581, REF-696

sigwaitinfo関数 REF-700
 sigwait関数 REF-698
 sinh関数 REF-703
 sin関数 REF-702
 sleep関数 REF-704
 snprintf関数 REF-705
 sprintf関数 REF-707
 sqrt関数 REF-709
 srand48関数 REF-711
 drand48関数との組み合わせ REF-173
 lcong48関数との組み合わせ REF-427
 lrand48関数との組み合わせ REF-448
 mrand48関数との組み合わせ REF-499
 srand関数 REF-712
 random関数との組み合わせ REF-584
 srand関数 REF-710
 sscanf関数 REF-713
 signal関数 REF-371, REF-581, REF-715
 standend関数 REF-717
 standout関数 REF-718
 statvfs関数 REF-725
 stat関数 REF-719
 ftw関数との組み合わせ REF-285
 stat構造体
 ftw関数での使用 REF-285
 stderr REF-224, REF-262, REF-543,
 REF-547
 stdin REF-262, REF-547, REF-615
 stdout REF-262, REF-547, REF-560,
 REF-567, REF-571, REF-574
 strcasecmp関数 REF-728
 strcat関数 REF-729
 strchr関数 REF-472, REF-731
 strcmp関数 REF-474, REF-733
 strcoll関数 REF-734
 strcpy関数 REF-475, REF-735
 strcspn関数 REF-736
 strdup関数 REF-737
 strerror関数 REF-738
 strfmon関数 REF-740
 strftime関数 REF-744
 tzset関数との組み合わせ REF-833
 strlen関数 REF-750
 strncasecmp関数 REF-751
 strncat関数 REF-752
 strncmp関数 REF-753
 strncpy関数 REF-755
 strnlen関数 REF-757
 strpbrk関数 REF-758
 strptime関数 REF-759
 strrchr関数 REF-764
 strsep関数 REF-766
 strspn関数 REF-768
 strstr関数 REF-769
 strtod関数 REF-27, REF-771
 strtok_r関数 REF-773
 strtok関数 REF-773

strtoll関数 REF-779
 strtol関数 REF-28, REF-29, REF-777
 strtog関数 REF-779
 strtoull関数 REF-783
 strtoul関数 REF-781
 strtoug関数 REF-783
 strxfrm関数 REF-785
 subwin関数 REF-789
 swab関数 REF-791
 swprintf関数 REF-792
 swscanf関数 REF-794
 symlink関数 REF-796
 SYSSWAKE REF-13
 sysconf関数 REF-798
 system関数 REF-803

T

tanh関数 REF-806
 tan関数 REF-805
 telldir関数 REF-807
 tempnam関数 REF-808
 tgamma関数 REF-811
 timespec構造体 REF-306
 times関数 REF-813
 time関数 REF-812
 tmpfile関数 REF-815
 tmpnam関数 REF-816
 toascii関数 REF-817
 TOLERANT リエントラント REF-147
 tolower関数 REF-819
 Tolower関数 REF-818
 touchwin関数 REF-821
 toupper関数 REF-823
 Toupper関数 REF-822
 towctrans関数 REF-825
 tolower関数 REF-826
 toupper関数 REF-827
 truncate関数 REF-829
 trunc関数 REF-828
 ttyname_r関数 REF-831
 ttyname関数 REF-831
 tzset関数 REF-833

U

ualarm関数 REF-838
 umask関数 REF-840
 uname関数 REF-842
 ungetc関数 REF-843
 ungetwc関数 REF-844
 UNIX I/O 関数
 close REF-81
 creat REF-99
 dup REF-175
 dup2 REF-175
 fcntl REF-209
 fileno REF-237

UNIX I/O 関数 (続き)

fstat	REF-272
getname	REF-334
getopt	REF-336
isapipe	REF-391
isatty	REF-393
lseek	REF-452
open	REF-532
pread	REF-558
pwrite	REF-575
read	REF-587
readv	REF-593
stat	REF-719
ttyname	REF-831
ttyname_r	REF-831
write	REF-972
unlink関数	REF-846
unordered関数	REF-847
unsetenv関数	REF-848
usleep関数	REF-849
utimes関数	REF-853
utime関数	REF-850

V

va_arg関数	REF-860
va_countマクロ	REF-861
va_end関数	REF-862
va_start_1マクロ	REF-863
va_startマクロ	REF-863
VAX\$CRTL INIT関数	REF-840, REF-856
VAX\$ESTABLISH関数	REF-446, REF-634, REF-858
vfork関数	REF-547, REF-866
vfprintf関数	REF-868
vfscanf関数	REF-869
vfwprintf関数	REF-871
vfwscanf関数	REF-874
vprintf関数	REF-876
vscanf関数	REF-877
vsnprintf関数	REF-878
vsprintf関数	REF-880
vsscanf関数	REF-881
vswprintf関数	REF-883
vswscanf関数	REF-885
vwprintf関数	REF-887
vwsscanf関数	REF-888

W

waddch関数	REF-11
waddstr関数	REF-12
wait3関数	REF-890
wait4関数	REF-893
waitpid関数	REF-896
wait関数	REF-889
waitpid関数との組み合わせ	REF-896

wclear関数	REF-70
wclrattr関数	REF-86
wclrtoebot関数	REF-87
wclrtoeol関数	REF-88
wcrtomb関数	REF-900
wscat関数	REF-902
wcschr関数	REF-905
wscmp関数	REF-907
wscoll関数	REF-909
wscpy関数	REF-910
wscspn関数	REF-911
wcsftime関数	REF-913
wcslen関数	REF-919
wcsncat関数	REF-920
wcsncmp関数	REF-923
wcsncpy関数	REF-925
wcspbrk関数	REF-927
wcsrchr関数	REF-929
wcsrtombs関数	REF-931
wcsspn関数	REF-933
wcsstr関数	REF-935
wcstod関数	REF-936
wcstok関数	REF-938
wcstol関数	REF-941
wcstombs関数	REF-943
wcstoul関数	REF-945
wcswcs関数	REF-948
wcswidth関数	REF-950
wcsxfrm関数	REF-951
wctob関数	REF-955
wctomb関数	REF-956
wctrans関数	REF-957
wctype関数	REF-958
wcwidth関数	REF-962
wdelch関数	REF-159
wdelete関数	REF-162
werase関数	REF-186
wgetch関数	REF-176, REF-303
wgetstr関数	REF-176, REF-355
winch関数	REF-381
winsch関数	REF-386
winsertln関数	REF-387
winsstr関数	REF-388
wmemchr関数	REF-963
wmemcmp関数	REF-964
wmemcpy関数	REF-965
wmemmove関数	REF-966
wmemset関数	REF-968
wmove関数	REF-496
wprintf関数	REF-969
wprintw関数	REF-561
wrapok関数	REF-971
wrefresh関数	REF-600
writev関数	REF-974
write関数	REF-972
wscanf関数	REF-976
wscanw関数	REF-616

wsetattr関数	REF-623
wstandend関数	REF-717
wstandout関数	REF-718

Y

y0関数	REF-977
y1関数	REF-977
yn関数	REF-977

イ

移植性の問題

_exit関数	REF-197
gsignal関数	REF-372
longname関数	REF-447
mkdirへの引数	REF-482
mv[w]insstr関数	REF-515
[no]nl関数	REF-525
raise関数	REF-582
ssignal関数	REF-715
ttyname_r関数	REF-831
ttyname関数	REF-831
va_startマクロ	REF-863
vforkとfork関数	REF-866
[w]clrattr関数	REF-86
[w]insstr関数	REF-388
[w]setattr関数	REF-623
メモリ割り当ての解除	REF-63

エ

エラー処理関数

abort	REF-5
exit	REF-197
_exit	REF-197
perror	REF-543
strerror	REF-738

キ

機能設定ルーチン

decc\$feature_get	REF-120
decc\$feature_get_index	REF-122
decc\$feature_get_name	REF-123
decc\$feature_get_value	REF-124
decc\$feature_set	REF-126
decc\$feature_set_value	REF-128
decc\$feature_show	REF-130
decc\$feature_show_all	REF-131

金額値の書式設定関数

strfmon	REF-740
---------------	---------

ク

グループ・データベース関数

endgrent	REF-181
getgrent	REF-319
getgrgid_r	REF-323
getgrnam	REF-325
getgrnam_r	REF-327
setgrent	REF-630

グローバル・データベース関数

getgrgid	REF-321
----------------	---------

コ

子プロセス

pipeによるデータの共用	REF-545
vforkによる作成	REF-866

サ

サブプロセス

pipeによるデータの共用	REF-545
---------------------	---------

サブプロセス関数

decc\$set_child_default_dir	REF-140
decc\$set_child_standard_streams	REF-142
decc\$validate_wchar	REF-154
decc\$write_eof_to_mbx	REF-156
execl	REF-188
execle	REF-190
execlp	REF-192
execv	REF-193
execve	REF-194
execvp	REF-196
pipe	REF-545
vfork	REF-866
wait	REF-889

算術関数

abs	REF-6
acos	REF-9
acosh	REF-10
asin	REF-17
asinh	REF-18
atan	REF-21
atan2	REF-22
atanh	REF-24
cabs	REF-42
cacos	REF-43
cacosh	REF-44
carg	REF-46
casin	REF-47
casinh	REF-48
catan	REF-49
catanh	REF-50
cbrt	REF-58
ccos	REF-59
ccosh	REF-60

算術関数 (続き)

ceil	REF-61
cexp	REF-62
cimag	REF-69
clog	REF-80
conj	REF-91
copysign	REF-92
cos	REF-93
cosh	REF-94
cot	REF-95
cpow	REF-96
cproj	REF-97
creal	REF-98
csin	REF-110
csinh	REF-111
csqrt	REF-112
ctan	REF-113
ctanh	REF-114
div	REF-167
erf	REF-187
exp	REF-199
exp2	REF-201
fabs	REF-203
fdim	REF-218
finite	REF-238
floor	REF-240
fma	REF-241
fmax	REF-242
fmin	REF-243
fp_class	REF-247
fp_classf	REF-247
fp_classl	REF-247
frexp	REF-264
hypot	REF-373
ilogb	REF-380
isnan	REF-398
j0	REF-417
j1	REF-417
jn	REF-417
labs	REF-425
ldexp	REF-428
ldiv	REF-429
lgamma	REF-431
llabs	REF-577
lldiv	REF-578
log	REF-441
log10	REF-441
log1p	REF-443
logb	REF-444
lrint	REF-450
lround	REF-451
modf	REF-495
nextafter	REF-520
nexttoward	REF-521
nint	REF-524
pow	REF-556
qabs	REF-577
qdiv	REF-578

算術関数 (続き)

rand	REF-583
remainder	REF-601
remquo	REF-602
rint	REF-610
scalb	REF-614
shm_open	REF-656
shm_unlink	REF-659
sin	REF-702
sinh	REF-703
sqrt	REF-709
srand	REF-710
tan	REF-805
tanh	REF-806
tgamma	REF-811
trunc	REF-828
unordered	REF-847
y0	REF-977
y1	REF-977
yn	REF-977

シ

シグナル処理関数

alarm	REF-13
gsignal	REF-371
kill	REF-421
longjmp	REF-445
pause	REF-541
raise	REF-581
setjmp	REF-633
sigaction	REF-661
sigaddset	REF-665
sigblock	REF-667
sigdelset	REF-668
sigemptyset	REF-669
sigfillset	REF-671
sighold	REF-672
sigignore	REF-674
sigismember	REF-676
siglongjmp	REF-677
sigmask	REF-678
signal	REF-679
sigpause	REF-681
sigpending	REF-682
sigprocmask	REF-683
sigrelse	REF-685
sigsetjmp	REF-687
sigsetmask	REF-689
sigstack	REF-690
sigsuspend	REF-692
sigtimedwait	REF-694
sigvec	REF-696
sigwait	REF-698
sigwaitinfo	REF-700
sleep	REF-704
ssignal	REF-715
VAXC\$ESTABLISH	REF-858

時刻関連関数

asctime	REF-15
asctime_r	REF-15
clock	REF-74
clock_getres	REF-75
clock_gettime	REF-77
clock_settime	REF-78
ctime	REF-116
ctime_r	REF-116
decc\$fix_time	REF-132
difftime	REF-164
ftime	REF-281
getclock	REF-306
getitimer	REF-331
gettimeofday	REF-356
gmtime	REF-369
gmtime_r	REF-369
localtime_r	REF-438
mktime	REF-487
nanosleep	REF-517
setitimer	REF-631
strftime	REF-744
strptime	REF-759
time	REF-812
times	REF-813
tzset	REF-833
ualarm	REF-838
usleep	REF-849
utime	REF-850
utimes	REF-853
wcsftime	REF-913

システム関数

asctime	REF-15
asctime_r	REF-15
assert	REF-19
atexit	REF-25
bsearch	REF-37
chdir	REF-64
chmod	REF-66
chown	REF-68
clock	REF-74
ctermid	REF-115
ctime	REF-116
ctime_r	REF-116
cuserid	REF-118
difftime	REF-164
fchmod	REF-204
fchown	REF-205
fmod	REF-244
ftime	REF-281
getcwd	REF-308
getenv	REF-313
getpid	REF-343
getppid	REF-344
gmtime	REF-369
gmtime_r	REF-369
localtime	REF-438
localtime_r	REF-438

システム関数 (続き)

memset	REF-479
mkdir	REF-481
nice	REF-522
qsort	REF-579
remainder	REF-601
remove	REF-160, REF-603
remquo	REF-602
rename	REF-605
setbuf	REF-624
setvbuf	REF-653
strtod	REF-771
strtok	REF-773
strtok_r	REF-773
system	REF-803
time	REF-812
times	REF-813
umask	REF-840
utime	REF-850
utimes	REF-853
vfprintf	REF-868
vfprintf	REF-869
vprintf	REF-876
vscanf	REF-877
vsnprintf	REF-878
vsprintf	REF-880
vsscanf	REF-881
wcstod	REF-936
wcstok	REF-938
wrtv	REF-974
新機能	xxii

シンボリック・リンク関数

lchown	REF-426
lstat	REF-454
readlink	REF-591
realpath	REF-598
symlink	REF-796
unlink	REF-846

セ

セキュリティ/インパーソネーション関数

getegid	REF-311
geteuid	REF-315
getgid	REF-317
getgroup	REF-329
getpgid	REF-341
getpgrp	REF-342
getsid	REF-354
getuid	REF-357
seteuid	REF-627
setgid	REF-628
setpgid	REF-640
setpgrp	REF-642
setregid	REF-644
setreuid	REF-646
setsid	REF-648
setuid	REF-651

タ

タイム・ゾーン・キャッシュ REF-836

ターミナル I/O 関数

getchar REF-304
gets REF-353
getwchar REF-361
printf REF-560
putchar REF-567
puts REF-571
putwchar REF-574
scanf REF-615

ハ

パスワード暗号化関数

encrypt REF-179
setkey REF-635

パスワードの暗号化関数

crypt REF-108

ヒ

標準 I/O 関数

clearerr REF-71
clearerr_unlocked REF-72
delete REF-160, REF-603
dlclose REF-168
dlerror REF-169
dlopen REF-170
dlsym REF-172
fclose REF-207
fdopen REF-219
feof REF-220
feof_unlocked REF-221
ferror REF-222
ferror_unlocked REF-223
fflush REF-224
fgetc REF-226
fgetc_unlocked REF-227
fgetname REF-228
fgets REF-232
fgetwc REF-234
fgetws REF-235
flockfile REF-239
fopen REF-245
fprintf REF-251
fputc REF-253
fputc_unlocked REF-254
fputs REF-255
fputwc REF-256
fputws REF-258
fread REF-259
freopen REF-262
fscanf REF-266
fseek REF-268
fseeko REF-270

標準 I/O 関数 (続き)

ftell REF-279
ftello REF-280
ftrylockfile REF-284
funlockfile REF-288
fwrite REF-295
getc REF-301
getc_unlocked REF-302
getchar_unlocked REF-305
getw REF-359
getwc REF-360
mktemp REF-486
putc REF-563
putc_unlocked REF-565
putchar_unlocked REF-568
putw REF-572
putwc REF-573
rewind REF-607
setbuf REF-624
setvbuf REF-653
snprintf REF-705
sprintf REF-707
sscanf REF-713
tmpfile REF-815
tmpnam REF-816
ungetc REF-843
ungetwc REF-844

フ

ファイル指定変換ルーチン

decc\$fix_time REF-132
decc\$from_vms REF-134
decc\$match_wild REF-136
decc\$to_vms REF-149
decc\$translate_vms REF-152

ファイル保護 REF-66, REF-840

マ

マルチバイト文字サポート

btowc REF-40
mblen REF-458
mbrlen REF-459
mbrtowc REF-461
mbsinit REF-467
mbtowc REF-465
wctomb REF-900
wctob REF-955
wctomb REF-956

マルチバイト文字列サポート

mbsrtowcs REF-468
mbstowcs REF-463
wcsrtombs REF-931
wcstombs REF-943

メ

メッセージング関数

catclose	REF-51
catgets	REF-52
catopen	REF-55

メモリの再割り当て

REF-261

メモリの割り当て

calloc関数	REF-45
malloc関数	REF-456
realloc関数	REF-596

メモリ割り当て関数

brk	REF-35
calloc	REF-45
cfree	REF-63
free	REF-261
malloc	REF-456
realloc	REF-596
sbrk	REF-612

モ

文字セット変換関数

iconv	REF-374
iconv_close	REF-376
iconv_open	REF-377

文字分類関数

isalnum	REF-389
isalpha	REF-390
isascii	REF-392
iscntrl	REF-394
isdigit	REF-395
isgraph	REF-396
islower	REF-397
isprint	REF-399
ispunct	REF-400
isspace	REF-401
isupper	REF-402
iswalnum	REF-403
iswalpha	REF-404
iswcntrl	REF-405
iswctype	REF-406
iswdigit	REF-408
iswgraph	REF-409
iswlower	REF-410
iswprint	REF-411
iswpunct	REF-412
iswspace	REF-413
iswupper	REF-414
iswxdigit	REF-415
isxdigit	REF-416
wctype	REF-958

文字変換関数

ecvt	REF-177
fcvt	REF-216
gcvt	REF-299
toascii	REF-817

文字変換関数 (続き)

tolower	REF-818
tolower	REF-819
toupper	REF-822
toupper	REF-823
tolower	REF-826
toupper	REF-827
wcswidth	REF-950
wcwidth	REF-962

文字列エンコード関数

crypt	REF-108
encrypt	REF-179
setkey	REF-635

文字列処理関数

atof	REF-27
atoi	REF-28
atol	REF-28
atoll	REF-29
atof	REF-29
basename	REF-30
bcmp	REF-32
bcopy	REF-33
bzero	REF-41
dirname	REF-165
ffs	REF-225
index	REF-382
memchr	REF-472
memcmp	REF-474
memcpy	REF-475
memmove	REF-477
memset	REF-479
rindex	REF-609
strcascmp	REF-728
strcat	REF-729
strchr	REF-731
strcmp	REF-733
strcoll	REF-734
strcpy	REF-735
strcspn	REF-736
strdup	REF-737
strlen	REF-750
strncasecmp	REF-751
strncat	REF-752
strncmp	REF-753
strncpy	REF-755
strnlen	REF-757
strpbrk	REF-758
strrchr	REF-764
strsep	REF-766
strspn	REF-768
strtok	REF-773
strtok_r	REF-773
strtol	REF-777
strtoll	REF-779
strtoq	REF-779
strtoul	REF-781
strtoull	REF-783
strtouq	REF-783

文字列処理関数 (続き)

strxfrm	REF-785
swab	REF-791
wscat	REF-902
wcschr	REF-905
wscmp	REF-907
wscoll	REF-909
wscpy	REF-910
wscspn	REF-911
wcslen	REF-919
wcsncat	REF-920
wcsncmp	REF-923
wcsncpy	REF-925
wcspbrk	REF-927
wcsrchr	REF-929
wcsspn	REF-933
wcstok	REF-938
wcstol	REF-941
wcstoul	REF-945
wcswcs	REF-948
wcsxfrm	REF-951
文字列比較関数	
wscoll	REF-909

ユ

ユーザ・データベース関数

endpwent	REF-182
getpwuid	REF-350
getpwuid_r	REF-350
setpwent	REF-643

リ

リエントラント	REF-147
AST	REF-147
MULTITHREAD	REF-147
NONE	REF-147
TOLERANT	REF-147
リエントラント関数	
decc\$set reentrancy	REF-147
リスト処理関数	
va_arg	REF-860
va_countマクロ	REF-861
va_end	REF-862
va_start	REF-863
va_start_1	REF-863

ロ

ロケール・サポート関数

localeconv	REF-434
nl_langinfo	REF-526
setlocale	REF-636

ワ

ワイド文字関数

btowc	REF-40
fgetwc	REF-234
fgetws	REF-235
fputwc	REF-256
fputws	REF-258
fwide	REF-290
fwprintf	REF-292
fwscanf	REF-297
getwc	REF-360
getwchar	REF-361
iswalnum	REF-403
iswalpha	REF-404
iswcntrl	REF-405
iswctype	REF-406
iswdigit	REF-408
iswgraph	REF-409
iswlower	REF-410
iswprint	REF-411
iswpunct	REF-412
iswspace	REF-413
iswupper	REF-414
iswxdigit	REF-415
mbrlen	REF-459
mbrtowc	REF-461
mbsinit	REF-467
mbsrtowcs	REF-468
putwc	REF-573
putwchar	REF-574
swprintf	REF-792
swscanf	REF-794
towctrans	REF-825
tolower	REF-826
toupper	REF-827
ungetwc	REF-844
vfwprintf	REF-871
vfwscanf	REF-874
vswprintf	REF-883
vswscanf	REF-885
vwprintf	REF-887
vwscanf	REF-888
wcrtomb	REF-900
wscat	REF-902
wcschr	REF-905
wscmp	REF-907
wscoll	REF-909
wscpy	REF-910
wscspn	REF-911
wcsftime	REF-913
wcslen	REF-919
wcsncat	REF-920
wcsncmp	REF-923
wcsncpy	REF-925
wcspbrk	REF-927
wcsrchr	REF-929

ワイド文字関数 (続き)

wcsrtombs	REF-931	wctob	REF-955
wcsspn	REF-933	wctrans	REF-957
wcsstr	REF-935	wctype	REF-958
wcstod	REF-936	wcwidth	REF-962
wcstok	REF-938	wmemchr	REF-963
wcstol	REF-941	wmemcmp	REF-964
wcstoul	REF-945	wmemcpy	REF-965
wcswcs	REF-948	wmemmove	REF-966
wcswidth	REF-950	wmemset	REF-968
wcsxfrm	REF-951	wprintf	REF-969
		wscanf	REF-976

OpenVMS
HP C ランタイム・ライブラリ・リファレンス・マニュアル (下巻)

2006 年 10 月 発行

日本ヒューレット・パカード株式会社

〒140-8641 東京都品川区東品川 2-2-24 天王洲セントラルタワー

電話 (03)5463-6600 (大代表)
