HP-UX 参考手册

第 3 节：库函数 (A-M)

HP-UX 11i v3

第 6 卷 （共 10 卷）

生产部件号：B2355-91042
E0207

法律声明

本文档中的信息如有更改，恕不另行通知。

担保

随 HP 产品及服务提供的明示性担保声明中列出了适用于此 HP 产品及服务的专用担保条款。本文中的任何内容均不构成额外的担保。HP 对本文中的技术或编辑错误以及缺漏不负任何责任。

美国政府许可

机密计算机软件。必须有 HP 授予的有效许可证，方可拥有、使用或复制本软件。根据供应商的标准商业许可证的规定，美国政府应遵守 FAR 12.211 和 12.212 中有关“商业计算机软件”、“计算机软件文档”与“商业货物技术数据”条款的规定。

附加版权声明

本文档及其所涉及的软件可能同时受到下述一项或多项版权的保护。某些单独的联机帮助页将对这些附加版权加以认可。

商标声明

Intel 和 Itanium 均为 Intel Corporation 在美国和其他国家（地区）的注册商标，使用时已受到许可。

Java 是 Sun Microsystems, Inc. 在美国的商标。

MS-DOS 和 Microsoft 是 Microsoft Corporation 在美国的注册商标。

OSF/Motif 是 The Open Group 在美国和其他国家 （地区）的商标。

UNIX 是 Open Group 的注册商标。

X Window System 是 X/Open Group 的商标。

前言

HP-UX 是 Hewlett-Packard Company 开发的一种 UNIX® 操作系统，可与各种行业标准兼容。该操作系统基于 System V Release 4 操作系统，并包括 Fourth Berkeley Software Distribution 中的重要功能。

本手册共十卷，由系统参考文档资料组成，其中包括若干个称为**联机帮助页**的条目，这些联机帮助页以 `man` 命令来命名，通过该命令，可以在系统上显示这些联机帮助页条目（请参阅 *man* (1)）。这些条目也称为联机帮助页和参考页。

简介

有关 HP-UX 以及联机帮助页的结构和格式的简介信息，请参阅第 9 卷中的 *introduction* (9) 联机帮助页。

小节简介

联机帮助页分为若干节，各节也包含 introduction 或 intro 联机帮助页，这两个联机帮助页对相应的节中的具体内容进行了说明。这些节有：

intro (1)	**第 1 节：用户命令** （第 1 卷包含 A-M；第 2 卷包含 N-Z）
intro (1M)	**第 1M 节：系统管理命令** （第 3 卷包含 A-M；第 4 卷包含 N-Z）
intro (2)	**第 2 节：系统调用** （第 5 卷）
intro (3C)	**第 3 节：库函数** （第 6 卷包含 A-M；第 7 卷包含 N-Z）
intro (4)	**第 4 节：文件格式** （第 8 卷）
intro (5)	**第 5 节：其他主题** （第 9 卷）
intro (7)	**第 7 节：设备专用文件** （第 10 卷）
intro (9)	**第 9 节：常规信息** （第 10 卷）
索引	**所有卷中的索引** （第 10 卷）

印刷字体约定

audit (5)	表示 HP-UX 联机帮助页。“audit”是联机帮助页名称，“5”是该帮助页在《HP-UX 参考手册》中的小节号。在网站和 Instant Information DVD 上，可能是指向该联机帮助页的热链接。在 HP-UX 命令行输入“`man audit`”或“`man 5 audit`”可以查看该联机帮助页。详见 *man* (1)。
《书名》	表示文档中引用的书籍、手册的名称，以宋体表示。
“术语”	表示文档中引用的专用术语，以宋体表示。
键盘操作	键盘键名称。注意， Return 和 Enter 指的是同一个键。
强调内容	第一次定义的名词和强调的内容用**黑体**表示。
`系统字体`	表示计算机显示的文本和系统项。
`可替换变量`	命令、功能中可以替换的变量名以仿宋表示。
`[ ]`	格式和命令说明中的可选内容。如果内容用 “|” 分隔，就必须选择其中之一。
`{ }`	格式和命令说明中的必需内容。如果内容用 “|” 分隔，就必须选择其中之一。
....	前面的元素可以重复任意多次。
|	分隔选项列表中的项目。

命令语法

文本	可按原样输入的词或字符。
可替换变量	用适当的值来替换的词或字符。
intro1	分为一组的一个或多个命令选项 -ikx。 chars 通常是由一系列字符组成的字符串，每个字符表示一个具体选项。例如， -ikx 等效于单独列出的选项 -i、 -k 和 -x。有时使用加号 (+) 作为选项前缀。
intro1M	一个命令选项，例如， -help。 word 是一个关键字。与 -chars 的区别通常是显而易见的，在 “选项”说明中对其进行了 解释。有时使用加号 (+) 和双连字符 (--) 作为选项前缀。
[ ]	格式和命令描述中可选的内容。
{ }	格式和命令描述中必需的内容。
|	竖线用来分隔一系列选项中的各个项，通常用在方括号或花括号中。
...	标记后面的省略号 (abc...)、右方括号后面的省略号 ([ ]...) 或右花括号后面的省略号 ({ }...) 表示前面的元素及其前面的空白字符 （如果有）可重复若干次。
...	省略号有时表示一个范围内省略的项。

函数概要和语法

HP-UX 函数采用定义格式，而不是用法格式。定义格式包含类型信息，在程序中插入此函数调用时会省略该信息。

函数的语法元素与命令的语法元素相同，但选项除外；请参阅第 VII 页上的“命令语法”。

函数常规定义

常规定义格式为：

```
类型  函数（类型  参数［，类型  参数］...  ）；
```

例如：

```
int setuname ( const char *name , size_t namelen );
```

函数用法

用法格式为：

```
函数（参数［，参数］...）；
```

例如：

```
setuname ( name [ , namelen ]... );
```

版本说明

部件号	发行版；日期；文档格式；发布形式
B2355-91037~46	HP-UX 11i v3；2007 年 2 月；PDF （10 卷）；docs.hp.com 和印刷版本。中文版。
B2355-90931~40	HP-UX 11i v2；2004 年 9 月；PDF （10 卷）；docs.hp.com 。中文版。

第 6 卷
目录
第 3 节

第 6 卷
目录

第 3 节

目录
第 6 、7 卷

第 3 节：库函数

目录
第 6 、7 卷

目录
第 6 、7 卷

目录
第 6 、7 卷

目录
第 6 、7 卷

目录
第 6 、7 卷

目录
第 6 、7 卷

目录
第 6 、7 卷

目录
第 6 、7 卷

目录
第 6 、7 卷

目录
第 6 、7 卷

目录
第 6 、7 卷

目录
第 6 、7 卷

目录
第 6 、7 卷

目录
第 6 、7 卷

目录
第 6 、7 卷

目录
第 6 、7 卷

目录
第 6 、7 卷

目录
第 6 、7 卷

目录
第 6 、7 卷

目录
第 6 、7 卷

目录
第 6 、7 卷

目录
第 6 、7 卷

目录
第 6 、7 卷

目录
第 6 、7 卷

条目名称(节编号)：名称 **说明**

目录
第 6 、7 卷

目录
第 6 、7 卷

目录
第 6 、7 卷

目录
第 6 、7 卷

条目名称(节编号)：名称 **说明**

目录
第 6 、7 卷

目录
第 6 、7 卷

目录
第 6 、7 卷

条目名称(节编号)：名称 **说明**

目录
第 6 、7 卷

目录
第 6 、7 卷

目录
第 6 、7 卷

目录
第 6 、7 卷

目录
第 6 、7 卷

目录
第 6 、7 卷

目录
第 6 、7 卷

目录
第 6 、7 卷

目录
第 6 、7 卷

目录
第 6 、7 卷

目录
第 6 、7 卷

目录
第 6 、7 卷

目录
第 6 、7 卷

目录
第 6 、7 卷

目录
第 6 、7 卷

目录
第 6 、7 卷

目录
第 6 、7 卷

目录
第 6 、7 卷

名称

intro - 子例行程序和库简介

说明

本节介绍各个库中出现的函数，而不是本卷第 (2) 节所介绍的，可直接调用 HP-UX 系统原语的那些函数。对于某些主要集合，将在小节标识符 (3) 后面加一个字母进行标识：

(3C) 这些函数，连同操作系统调用和那些标有 (3S) 的函数，构成了 C 编译程序 *cc*(1) 可自动加载的标准 C 库 **libc** 。可通过相应条目中指定的 **#include** 文件获取其中某些函数的声明。

(3E) 这些函数构成了 ELF 访问库 (**libelf**) ，程序可通过该库操作 ELF (*Executable and Linking Format*) 对象文件、归档文件和归档成员。如果指定了 **-lelf** 选项，则链接编辑器可搜索此库。头文件 **<libelf.h>** 可提供所有库服务（已在 *elf*(3E) 中介绍）的类型和函数声明。

(3G) 这些函数构成了图形库，在单独的手册中记录了这些函数。

(3I) 这些函数构成工具支持（设备 I/O）库。

(3M) 这些函数构成数学库 **libm** 。如果指定了 **-lm** 选项，则链接编辑器可搜索此库。头文件 **<math.h>** 和 **<fenv.h>** 中提供了这些函数的声明。 **<math.h>** （请参阅 *math*(5) ）中还定义了多个常用的数学常量。

(3N) 这些函数适用于 Internet 网络，并且是标准 C 库 **libc** 的一部分。

(3S) 这些函数构成“标准 I/O 包”（请参阅 *stdio*(3S) ）。这些函数在库 **libc** 中已进行了说明。可通过 **#include** 文件 **<stdio.h>** 获取这些函数的声明。

(3T) 这些函数构成 Pthread 库。如果指定了 **-lpthread** 选项，则链接编辑器 **ld** （请参阅 *ld*(1) ）可搜索此库。有关线程的详细信息，请参阅 *pthread*(3T) 。

(3X) 多种专用库。将在相应的条目中指定在其中找到这些库的文件。

诊断信息

如果未定义给定参数的函数，或者值无法表示，则 C 库和数学库（即 (3C) 和 (3M)）中的函数将返回常规值 **0** 或 **+|-HUGE_VAL** 。 **HUGE_VAL** 在 **<math.h>** 头文件中定义为 **+INFINITY** 。数学库中的函数还可返回 **+|-INFINITY** 或 **NaN** 。在这些情况下，还可将外部变量 **errno** （请参阅 *errno*(2) ）设置为值 [EDOM] 或 [ERANGE]。

文件

/usr/lib/libc.a （适用于 PA-RISC 系统）。标准 I/O、操作系统调用和通用例行程序归档库。

/usr/lib/libc.sl （适用于 PA-RISC 系统）。标准 I/O、操作系统调用和通用例行程序共享库。

/usr/lib/hpuxXX/libc.so （适用于基于 Itanium(R) 的系统）。标准 I/O、操作系统调用、通用例行程序 *32* 位和 *64* 位共享库。

/usr/lib/libcurses.sl	（适用于 PA-RISC 系统）。处理共享库的 CRT 屏幕。
/usr/lib/hpuxXX/libcurses.so	（适用于基于 Itanium 的系统）。处理 *32* 位和 *64* 位共享库的 CRT 屏幕。
/usr/lib/libelf.a	（适用于 PA-RISC 系统）。 ELF 归档库。
/usr/lib/libelf.sl	（适用于 PA-RISC 系统）。 ELF 共享库。
/usr/lib/hpuxXX/libelf.so	（适用于基于 Itanium 的系统）。 ELF *32*-位和 *64*-位共享库。
/usr/lib/libm.a	（适用于 PA-RISC 系统）。符合 SVID3、XPG4.2 和 ANSI C 的数学归档库。
/usr/lib/libm.sl	（适用于 PA-RISC 系统）。符合 SVID3、XPG4.2 和 ANSI C 的数学归档库。
/usr/lib/hpuxXX/libm.so	（适用于基于 Itanium 的系统）。符合 SVID3、XPG4.2 和 ANSI C 的 *32*-位和 *64*-位数学共享库。

另请参阅

ar(1)、 elf(3E)、 ld(1)、 nm(1)、 intro(2)、 stdio(3S)、 hier(5)、 math(5)、 thread_safety(5)、 introduction(9)。

可以通过 Web 访问 HP-UX 文档，网址为： **http://docs.hp.com** 。

名称

_UNW_createContextForSelf()、 _UNW_createContext()、 _UNW_destroyContext() - 分配和取消分配 unwind 库数据结构

概要

```
#include <unwind.h>

_Unwind_Context * _UNW_createContextForSelf(void);

_Unwind_Context * _UNW_createContext(
        _UNW_ReadTargetMem read_tgt_mem,
        _UNW_LoadMapFromIP load_map_from_ip,
        uint32_t ident);

void _UNW_destroyContext(_Unwind_Context* p);
```

说明

_UNW_createContextForSelf 和 **_UNW_createContext** 都可以初始化称为 **_Unwind_Context** 的数据结构，该数据结构由堆栈 unwind 库管理，并作为 **libunwind.so** 提供。当某个进程计划辗转开解自身的堆栈时，通常会使用 **_UNW_createContextForSelf** 。当某个进程计划辗转开解其他进程的堆栈，或者辗转开解核心文件中保留的“无用”进程的堆栈时，通常会使用 **_UNW_createContext** 。

使用 **_UNW_createContext** 辗转开解进程（称作“目标进程”）时，客户端进程必须提供以下三个参数：

1. 参数 *read_tgt_mem* 是一个函数，unwind 库将调用此函数从目标进程内存中读取值，包括辗转开解标头、辗转开解表和辗转开解信息块、过程调用堆栈和寄存器堆栈引擎后备存储区。由于 Integrity 系统的内核中断辗转开解标头是从内核网关内存页读取的，因此必须能够从 64 位地址空间（甚至是 32 位应用程序）的任何位置读取此参数。

   read_tgt_mem 包含类型定义 **_UNW_ReadTargetMem** ，它在 **unwind.h** 中定义为

   ```
   typedef void * (* _UNW_ReadTargetMem) (void *dst,
       uint64_t src,
       size_t length,
       uint32_t ident);
   ```

   这些参数与 *memcpy*(3C) 的参数大致相同。参数 *dst* 是目标地址，将从参数 *src* 向其复制参数 *length* 字节。请注意， *src* 的类型为 **uint64_t** ，这是为了能够表示读取内核网关页时所需要的 64 位地址。通过 64 位应用程序执行辗转开解操作时，辗转开解期间使用的所有 *src* 和 *dst* 地址都是 64 位指针。通过 32 位应用程序代码执行辗转开解操作时，辗转开解期间使用的大多数地址为 32 位指针（尽管 unwind 库在调用目标内存读取器之前，会将这些地址 调整为完全限定的 64 位地址）。有关内存模型及术语“调整”的详细解释，请参阅《Itanium Processor Family Runtime Architecture Supplement : 32-Bit Runtime Architecture for HP-UX》，“第 1 节：Memory Model”。

   需要 64 位指针的一系列地址（而从 32 位地址空间无法到达这些地址），只是那些与内核信号处理程序包装函数 **__user_sendsig** 的辗转开解标头和辗转开解信息相关联的地址。客户端内存读取回调函数必须

能够检测及读取此地址系列。程序启动时微加载程序和动态加载程序将在 **/usr/include/crt0.h** 所定义的结构 **load_info_t** 中定义此系列（或者在完全绑定的可执行文件的 **crt0.o** 中定义）。请参阅《Itanium Processor Family Runtime Architecture Supplement: Signal Handling on HP-UX》和《Itanium Processor Family Runtime Architecture Supplement: Program Startup on HP-UX》。

read_tgt_mem 的第三个参数 *ident* 对 unwind 库自身来说是透明的，它是针对辗转开解程序客户端程序的使用而提供的。客户端程序为 unwind 库提供 *ident* 的值（通过调用 **_UNW_createContext** ），接着 unwind 库又将此值传递给 *read_tgt_mem* 回调。例如，调试程序可以使用 *ident* 来标识目标进程内的各个线程。

2. 参数 *load_map_from_ip* 是一个函数，当 unwind 库需要获取目标进程内存中给定 IP 地址的 **load_module_desc** 时，可以调用这个函数，而不必调用 **dlmodinfo** （请参阅 *dlmodinfo*(3C) ）。调用 *load_map_from_ip* 后，它必须使用任何有效的用户进程指令地址（包括与 **__user_sendsig** 关联的指令地址）的 **linkage_ptr** 、 **unwind_base** 和 **text_base** 字段的准确值填充 **load_module_desc** 结构。**Unwind_base** （unwind 标头的位置）和 *text_base* （加载 ELF **.text** 段的位置）必须都是完全 64 位地址（与 32 位未调整指针相对）。 *Linkage_ptr* 是过程的 *GP* 寄存器的（完全 64 位）值。请参阅《Itanium Processor Family Software Conventions and Runtime Architecture》：第 8 章，“Procedure Linkage”。 *__user_sendsig* 的辗转开解信息将按上文 *read_tgt_mem* 介绍所述，在内核网关页中进行通信。

   load_map_from_ip 包含类型定义 **_UNW_LoadMapFromIP** ，它在 **unwind.h** 中定义为

```
typedef void (* _UNW_LoadMapFromIP) (
        struct load_module_desc * new_load_map,
        uint64_t ip,
        uint32_t ident);
```

   参数 *new_load_map* 是指向 *load_map_from_ip* 所要填充的预分配 *struct load_module_desc* 的指针。

   参数 *ip* 是与请求了信息的加载模块关联的任何指令指针。

   参数 *ident* 对 unwind 库自身来说是透明的，它是针对辗转开解程序客户端程序的使用而提供的。客户端程序为 unwind 库提供 *ident* 的值（通过调用 **_UNW_createContext** 提供），接着 unwind 库又将此值传递给 *load_map_from_ip* 回调。例如，调试程序可以使用 *ident* 来标识目标进程内的各个线程。

3. 上面的 *read_tgt_mem* 和 *load_map_from_ip* 段落介绍了 **_UNW_createContext** 的第三个参数 *ident* 的语义，该参数是针对使用辗转开解程序客户端程序而提供的。

尽管 **_UNW_createContext** 的作用是实现辗转开解除自身以外的进程的要求，进程仍然可以使用 **_Unwind_Context** 创建的 **_UNW_createContext** 来辗转开解自身。执行此操作的原因包括需要辗转开解动态生成代码或者运行时提供的代码。在这些情况下，客户端可能需要注册 **_UNW_LoadMapFromIP** 和 **_UNW_ReadTargetMem** 回调。

_UNW_destroyContext 可以释放 **_UNW_createContextForSelf** 或 **_UNW_createContext** 分配的内存。不再使用 **_Unwind_Context** 时，分配 **_Unwind_Context** 对象的应用程序应调用 **_UNW_destroyContext** ，以避免内存泄漏。

实际应用信息

_UNW_createContextForSelf、 **_UNW_createContext** 和 **_UNW_destroyContext** 是线程安全的。

返回值

指向结构 **_Unwind_Context** 的指针。

错误

出现以下情况时， **_UNW_createContextForSelf** 和 **_UNW_createContext** 可能无法创建辗转开解环境数据结构：

- 低内存的情况。在低内存的情况下，以下 unwind 库的行为可以得到保证：

  _Unwind_Context 构建失败可以为客户端程序创建足够的 *_Unwind_Context* 对象，以支持调用堆栈 unwind 库入口点 **_UNW_getAlertCode()** ，出现构建失败或 **_Unwind_Context** 指针为 NULL 时，它会返回 **_UNW_MEMORY_ALLOCATION_ERROR** （同时返回失败构建的指示符）。如果构建成功，**_UNW_getAlertCode()** 将返回 **_UNW_OK** 。

  在极低内存的情况下出现构建失败时，将通过返回 NULL 进行通信，同时将 **_Unwind_Context** 指针设置为 NULL 。如果 **_Unwind_Context** 指针为 NULL ，后续调用 **_UNW_ReturnCode** 返回类型的接口函数将返回 **_UNW_MEMORY_ALLOCATION_ERROR** 。

举例

示例 **1**

为辗转开解当前运行的进程分配和初始化 **_Unwind_Context** ：

```
#include <unwind.h>
#include <stdio.h>
_Unwind_Context *uc;
_UNW_ReturnCode retcode;
uc = _UNW_createContextForSelf();
if (uc == NULL) {
  /* Code to notify user of low memory problem */
}
if (_UNW_getAlertCode(uc) == _UNW_MEMORY_ALLOCATION_ERROR) {
  /* Code to notify user of low memory problem */
}
/* Example use of the _Unwind_Context * follows: */
if ((retcode = _UNW_currentContext(uc)) != _UNW_OK) {
   /* Notify user: Couldn't determine current context.  */
}
```

示例 **2**

为辗转开解除当前运行的进程以外的其他进程分配和初始化 **_Unwind_Context** ：

```
#include <unwind.h>
```

```
#include <stdio.h>
_Unwind_Context *uc;
_UNW_ReturnCode retcode;
extern _UNW_ReadTargetMem my_mem_reader;
extern _UNW_LoadMapFromIP my_dlmodule_info;
uint64_t target_ip;   /*  Target process instruction pointer */
uc = _UNW_createContext(my_mem_reader, my_dlmodule_info,1);
if (_UNW_getAlertCode(uc) == _UNW_MEMORY_ALLOCATION_ERROR) {
  /* Code to notify user of low memory problem */
}
/* Example use of the _Unwind_Context * follows: */
if ((retcode = _UNW_setIP(uc,target_ip)) != _UNW_OK) {
    /* Notify user: Initialization problem  */
}
```

作者

_UNW_createContextForSelf 、 **_UNW_createContext** 和 **_UNW_destroyContext** 由 HP 开发。

另请参阅

U_STACK_TRACE(3X)、 _UNW_currentContext(3X)、 _UNW_getGR(3X)、 unwind(5)。

名称

_UNW_currentContext() 、 _UNW_clear() 、 _UNW_jmpbufContext() 、 _UNW_setAR() 、 _UNW_setBR() 、 _UNW_setCFM()、 _UNW_setFR()、 _UNW_setGR()、 _UNW_setGR_NaT()、 _UNW_setIP()、 _UNW_setPR()、 _UNW_setPreds() 、 _UNW_step() 、 _UNW_FR_PhysicalNumber() 、 _UNW_GR_PhysicalNumber() 、 _UNW_PR_PhysicalNumber() - 操作 unwind 库数据结构中的值

概要

```
#include <unwind.h>

_UNW_ReturnCode _UNW_currentContext(_Unwind_Context* p);

_UNW_ReturnCode _UNW_clear(_Unwind_Context* p);

_UNW_ReturnCode _UNW_jmpbufContext(_Unwind_Context* p,
        jmp_buf env);

_UNW_ReturnCode _UNW_setGR(_Unwind_Context* p,
        uint32_t num,
        uint64_t value);

_UNW_ReturnCode _UNW_setGR_NaT(_Unwind_Context* p,
        uint32_t num,
        uint64_t value,
        _UNW_Boolean NaTval);

_UNW_ReturnCode _UNW_setFR(_Unwind_Context* p,
        uint32_t num,
        uint64_t first_container,
        uint64_t second_container);

_UNW_ReturnCode _UNW_setBR(_Unwind_Context* p,
        uint32_t num,
        uint64_t value);

_UNW_ReturnCode _UNW_setAR(_Unwind_Context* p,
        _UNW_AppReg num,
        uint64_t value);

_UNW_ReturnCode _UNW_setPR(_Unwind_Context* p,
        uint32_t num,
        _UNW_Boolean value);

_UNW_ReturnCode _UNW_setPreds(_Unwind_Context* p,
        uint64_t value);

_UNW_ReturnCode _UNW_setIP(_Unwind_Context* p,
```

```
        uint64_t value);

_UNW_ReturnCode _UNW_setCFM(_Unwind_Context* p,
        uint64_t value);

uint32_t _UNW_GR_PhysicalNumber(_Unwind_Context* p,
        uint32_t logical_num);

uint32_t _UNW_FR_PhysicalNumber(_Unwind_Context* p,
        uint32_t logical_num);

uint32_t _UNW_PR_PhysicalNumber(_Unwind_Context* p,
        uint32_t logical_num);

_UNW_ReturnCode _UNW_step(_Unwind_Context* p);
```

说明

_UNW_currentContext() 可初始化 **_Unwind_Context** 对象，以描述调用 **_UNW_currentContext** 的过程的处理器状态 — 换句话说，处理器的“self”状态。

_UNW_jmpbufContext() 可初始化 **_Unwind_Context** 对象，以描述调用 **setjmp()** （请参阅 *setjmp*(3C) ）后在 **jmp_buf** 中捕获的处理器状态。

_UNW_clear() 可以将 **_Unwind_Context** 对象重置为“just constructed”状态。 **_Unwind_Context** 中的所有寄存器值均会失效。将 unwind 库放入 *Init* 状态表示客户端现在可以使用 *_UNW_set* 例行程序来初始化 **_Unwind_Context** 中的值。 *unwind*(5) 中进一步介绍了 *Init* 状态。

_UNW_setGR() 可以将已编号的常规寄存器的值 *num* 初始化为参数 *p* 指向的 **_Unwind_Context** 中的 *value* 。寄存器的 NAT 位设置为 **_UNW_FALSE** 。

_UNW_setGR_NaT() 可以将已编号的常规寄存器的值 *num* 初始化为参数 *p* 指向的 **_Unwind_Context** 中的 *value*。寄存器的 NAT 位设置为 **NaTval** 。

_UNW_setFR() 可以将 **_Unwind_Context** 中参数 *p* 指向的已编号浮点寄存器的值 *num* ，初始化为 *first_container* 和 *second_container* 中表示的值映像。这两个容器表示浮点寄存器 Spill/Fill 内存格式的两个连续双字。请参阅《Intel IA-64 Architecture Software Developer's Manual》，“第 1 卷：IA-64 Application Architecture”，“第 5.3 节：Floating Point Instructions”。

_UNW_setBR() 可以将已编号的分支寄存器的值 *num* 初始化为参数 *p* 指向的 **_Unwind_Context** 中的 *value* 。

_UNW_setAR() 可以将枚举的应用程序寄存器的值 *num* 初始化为 *p* 指向的 **_Unwind_Context** 中的 *value* 。枚举类型 **_UNW_AppReg** 用于访问应用程序寄存器。

_UNW_setPR() 可以将参数 *p* 指向的 **_Unwind_Context** 中的已编号谓词寄存器值 *num* ，初始化为 *value* ，即 **_UNW_TRUE** 和 **_UNW_FALSE** 中的一个。

_UNW_setPreds() 可以将参数 *p* 指向的 **_Unwind_Context** 中的所有谓词寄存器值，初始化为 *value* 中传递的表示形式。 *value* 包含与谓词寄存器编号对应的位中每个谓词寄存器的内容。例如，位 63 包含谓词寄存器 63 的内容

（1 或 0）。

_UNW_setIP() 可以将指令指针的值初始化为参数 *p* 指向的 **_Unwind_Context** 中的 *value* 。 *value* 必须为完整的 64 位地址。也就是说，将来自 32 位地址空间的任何 32 位指针值传递到 **_UNW_setIP()** 之前，必须对其进行调整。请参阅《Itanium Processor Family Runtime Architecture Supplement: 32-Bit Runtime Architecture for HP-UX》，“第 1 节：Memory Model”。

_UNW_setCFM() 将 “当前帧标记”值 (CFM) 初始化为参数 *p* 指向的 **_Unwind_Context** 中的 *value* 。请参阅《Intel IA-64 Architecture Software Developer's Manual》，“第 1 卷：IA-64 Application Architecture”，“第 3.1.7 节：Current Frame Marker”。当前帧标记包含堆栈帧各个部分的大小。此外还指定三个寄存器重命名基值（在寄存器循环中使用）。当前帧标记不是体系结构可视值。当初始化一个需要辗转开解的 **_Unwind_Context** ，以便提供包含寄存器堆栈引擎 (RSE) 图片的初始 **_Unwind_Context** （特别说明从 *GR32* 到 *GR127* 的范围内，有多少堆栈的常规寄存器在当前帧中），以及初始化任何有效的寄存器基循环时，unwind 库会使用当前帧标记。客户端可以通过多种方法构建当前帧标记值:

- 需要辗转开解自身的过程称为“当前状态”集合过程，其目的是填充描述自身状态的 **_Unwind_Context** 结构。该过程（如果是用汇编语言编写的）可以使用自身的寄存器用途知识来制定当前帧标记值。
- 希望辗转开解被其中断（例如，通过设置分隔点）的除自身以外的过程的工具（如调试程序），可以拼凑由于中断事件导致的中断环境中的当前帧标记值。请参阅《Intel IA-64 Architecture Software Developer's Manual》，“第 2 卷：IA-64 System Architecture”，“第 5 章：IA-64 Interruptions”。

参数 *p* 指向的 **_Unwind_Context** 的 **_UNW_GR_PhysicalNumber()** 、 **_UNW_FR_PhysicalNumber()** 和 **_UNW_PR_PhysicalNumber()** 可返回与逻辑寄存器编号对应的物理寄存器编号 *logical_num* 。以下“初始化程序函数之间的交互”一节解释了物理寄存器编号与逻辑寄存器编号之间的差别。

初始化程序函数之间的交互

堆栈 unwind 库要求只有在 **Init** 状态（以及处理符合 ANSI C++ 标准例外情况时的 **Pre-install** 状态）时，才调用这些 setter 函数。 *unwind*(5) 中讨论了状态限制。

调用 **_UNW_clear()** 、 **_UNW_createContextForSelf()** 或 **_UNW_createContext()** 时，会切换到 Init 状态。在 *Init* 状态下，允许 **_UNW_set...** 调用初始化常规寄存器 1-31、临时寄存器和保留的浮点寄存器、临时寄存器和保留的谓词寄存器、分支寄存器 0-7、 *IP* 、 *CFM* 以及 *RSC* 、 *BSP* 、 *BSPSTORE* 、 *RNAT* 、 *CCV* 、 *UNAT* 、 *FPSR* 、 *ITC* 、 *PFS* 、 *LC* 集中应用程序寄存器中 **_Unwind_Context** 的值。写入特定寄存器可以使 **_Unwind_Context** 中的该值有效。调用 **_UNW_setCFM** 可以使 GR32 至 GR127 范围内 **_Unwind_Context** 的常规寄存器值失效，及使 **_Unwind_Context** 的当前帧标记 (CFM) 值有效。一旦 CFM 值有效，就允许在 *GR32* 至 *GR32 + CFM.sof* 的范围内对常规寄存器进行写入操作（为了初始化当前过程 RSE 帧中的 RSE 堆栈常规寄存器）。

为使 **_UNW_step()** 调用成功，必须使 *unwind*(5) “初始化”一节中所列的特定寄存器集有效。

不允许初始化上一段落中特别列出的寄存器值。例如，不允许初始化常量寄存器（ GR0 、 *FR0* 、 *FR1* 和 *PR0* ）。如果违反这条规则， **_Unwind_Context** 警报代码将设置为 **_UNW_INITIALIZATION_RANGE_ERROR**，同时 **_UNW_set...** 函数将返回 **_UNW_INITIALIZATION_RANGE_ERROR** 。

在不允许初始化值的状态下初始化 **_Unwind_Context** 对象，将不会更改 **_Unwind_Context** 中的值。而是导致返回警报代码 **_UNW_SET_NOT_ALLOWED_IN_STATE** 。另请参阅 *unwind*(5) 。

unwind 库值初始化程序函数通过与映射（逻辑）编号（与 *CFM.sor* 、 *CFM.rrb.gr* 、 *CFM.rrb.pr* 、 *CFM.rrb.fr* 中指示的循环关联）相对的物理编号来表示寄存器。如果相应 *CFM.rrb* 字段的值为 0，堆栈 unwind 库会将物理编号定义为值所驻留的寄存器编号。

映射函数 **_UNW_GR_PhysicalNumber** 、 **_UNW_FR_PhysicalNumber** 和 **_UNW_PR_PhysicalNumber** 可以返回物理寄存器编号，该编号将被传递到 **_UNW_get...** 初始化程序函数，以访问给定的逻辑寄存器。如果为映射函数传递一个超出映射逻辑编号范围的逻辑编号，将返回 0，并设置 **_UNW_QUERY_RANGE_ERROR** 警报状态。例如，当 *CFM.sor* 为 *0* 时，调用 **_UNW_GR_PhysicalNumber(uc,42)** 将会设置警报状态。使用映射函数之前应初始化 **_Unwind_Context** 对象的 CFM 内容。

_UNW_step 可以修改参数 *p* 指向的 **_Unwind_Context** ，表示 先前的处理器状态。

返回值

如果 **_Unwind_Context** 处于不允许初始化值的状态， **_UNW_set...** 将返回 **_UNW_SET_NOT_ALLOWED_IN_STATE** 。允许在其中执行初始化的状态有 Init 和 Pre_Install。请参阅联机帮助页 *unwind*(5) 上的“Error conditions and recovery”。

如果禁止初始化某个寄存器，则尝试初始化该寄存器时， **_UNW_set...** 将返回 **_UNW_INITIALIZATION_RANGE_ERROR** 。请参阅以上小节“初始化程序函数之间的交互”。否则，如果成功完成， **_UNW_set...** 将返回 **_UNW_OK** 。

如果成功完成， **_UNW_clear** 将返回 **_UNW_OK** 。否则，当 **_Unwind_Context** 不为下列其中一种状态时， **_UNW_clear** 将返回 **_UNW_CLEAR_NOT_ALLOWED_IN_STATE** ： *Init* 、 *User_Interrupted_Frame* 、 *User_Sendsig_Frame* 、 *Kernel_Bottom_Frame* 、 *Frame* 。请参阅 *unwind*(5) 。

_UNW_GR_PhysicalNumber 、 **_UNW_FR_PhysicalNumber** 和 **_UNW_PR_PhysicalNumber** 中的每个都会返回一个寄存器编号。如果传递了超出寄存器类映射逻辑编号范围的逻辑编号，这些函数会返回 0（同时附带地将 **_Unwind_Context** 警报代码设置为 **_UNW_QUERY_RANGE_ERROR** ）。

_UNW_step 可以生成以下返回代码:

_UNW_STEP_KERNEL_SAVE_STATE

表示 **_Unwind_Context** ，描述一种范围，超出这个范围时堆栈 unwind 库将无法步进。除与支持用户签名处理的 **_user_sendsig()** （请参阅 *signal*(5) ）相关联的 HP_UX 内核中断帧外，其他任何内核中断帧都会导致此返回值。

_UNW_STEP_BOTTOM

表示 **_Unwind_Context** ，描述一种范围，超出这个范围时堆栈 unwind 库将无法步进。当调用 **_UNW_step** 以实现描述一个其帧标记了堆栈底部约定 — 已保存返回链接 0 （请参阅《Itanium Processor Family Software Conventions and Runtime Architecture》，“第 11.1 章: Unwinding the stack”）的过程的任何 **_Unwind_Context** 时，将生成此返回代码。

_UNW_OK

全部正常。

_UNW_STEP_ERROR

步进过程中发生了一些常规问题。此返回代码表示 **_Unwind_Context** 处于 Bad 状态。请参阅 *unwind*(5) “状态”一节。

_UNW_INTERNAL_ERROR

步进过程中发生了一些逻辑问题。请与 HP 支持联系。

_UNW_STEP_INVALID_IP

_Unwind_Context 中的指令指针值标记为无效。可能性原因是对其初始化失败。此返回代码表示 **_Unwind_Context** 处于 *Bad* 状态。请参阅 *unwind*(5) 中“状态”一节。

_UNW_STEP_INVALID_SP

_Unwind_Context 中的堆栈指针值标记为无效。可能性原因是对其初始化失败。此返回代码表示 **_Unwind_Context** 处于 *Bad* 状态。请参阅 *unwind*(5) 中“状态”一节。

_UNW_STEP_INVALID_GR

步进过程中 **_Unwind_Context** 遇到标记为无效的常规寄存器值。可能性原因是对其初始化失败。此返回代码表示 **_Unwind_Context** 处于 *Bad* 状态。请参阅 *unwind*(5) 中“状态”一节。

_UNW_STEP_INVALID_PFS

_Unwind_Context 中的 AR.PRS 值标记为无效。可能性原因是对其初始化失败。此返回代码表示 **_Unwind_Context** 处于 *Bad* 状态。请参阅 *unwind*(5) 中“状态”一节。

_UNW_STEP_INVALID_RSC

_Unwind_Context 中的 AR.RSC 值标记为无效。可能性原因是对其初始化失败。此返回代码表示 **_Unwind_Context** 处于 *Bad* 状态。请参阅 *unwind*(5) 中“状态”一节。

_UNW_STEP_INVALID_BSP

_Unwind_Context 中的 AR.BSP 值标记为无效。可能性原因是对其初始化失败。此返回代码表示 **_Unwind_Context** 处于 *Bad* 状态。请参阅 *unwind*(5) 中“状态”一节。

_UNW_STEP_INVALID_BSPSTORE

_Unwind_Context 中的 AR.BSPSTORE 值标记为无效。可能性原因是对其初始化失败。此返回代码表示 **_Unwind_Context** 处于 *Bad* 状态。请参阅 *unwind*(5) 中“状态”一节。

_UNW_STEP_INVALID_CFM

_Unwind_Context 中的 AR.BFM 值标记为无效。可能性原因是对其初始化失败。此返回代码表示 **_Unwind_Context** 处于 *Bad* 状态。请参阅 *unwind*(5) 中“状态”一节。

_UNW_STEP_INVALID_BR

在步进过程中， **_Unwind_Context** 中遇到标记为无效的分支寄存器值。可能性原因是对其初始化失败。此返回代码表示 **_Unwind_Context** 处于 *Bad* 状态。请参阅 *unwind*(5) 中“状态”一节。

_UNW_STEP_BAD_BSP_ALIGNMENT

_UNW_currentContext 的值 *ar.BSP* 未对齐。此返回代码表示 **_Unwind_Context** 处于 *Bad* 状态。请参阅 *unwind*(5) 中“状态”一节。

_UNW_STEP_INVALID_RNAT

_Unwind_Context 中的 AR.RNAT 值标记为无效。可能性原因是对其初始化失败。此返回代码表示 **_Unwind_Context** 处于 *Bad* 状态。请参阅 *unwind*(5) 中“状态”一节。

_UNW_STEP_NO_DESCRIPTOR_FOR_NON_LEAF

堆栈 unwind 库无法为过程找到辗转开解 (unwind) 描述符，堆栈 unwind 库可以证明此过程不是叶过程。要获得辗转开解描述符，需要所有的非叶过程。发生此情况的部分而不是全部的可能性原因包括

- 不符合《Itanium Processor Family Software Conventions and Runtime Architecture》，“第 11.3 章：Coding conventions for reliable unwinding”中指定的编码约定的代码
- **_UNW_LoadMapFromIP** 回调函数（请参阅 *unwind*(5)）识别辗转开解标头或文本段基的位置失败
- 系统内存损坏
- 使用了不正确的值初始化 **_Unwind_Context**

_UNW_STEP_CORRUPT_DESCRIPTOR

帧的辗转开解描述符格式不当。此返回代码表示 **_Unwind_Context** 处于 *Bad* 状态。请参阅 *unwind*(5) 中“状态”一节。

_UNW_STEP_RSE_NOT_FLUSHED

寄存器堆栈引擎未刷新。请参阅 *unwind*(5) 。此返回代码表示 **_Unwind_Context** 处于 *Bad* 状态。请参阅 *unwind*(5) 中“状态”一节。

_UNW_currentContext 包含以下返回代码：

_UNW_OK

全部正常。

_UNW_CURRENT_CONTEXT_FAILED

调用 **_UNW_currentContext** 时发生了常规问题。

_UNW_STEP_CORRUPT_DESCRIPTOR

帧的辗转开解描述符格式不当。此返回代码表示 **_Unwind_Context** 处于 *Bad* 状态。请参阅 *unwind*(5) 中“状态”一节。

_UNW_MEMORY_ALLOCATION_ERROR

堆栈 unwind 库无法分配足够的内存来执行所请求的功能。 **_Unwind_Context** 处于 *Bad* 状态。请参阅 *unwind*(5) 。

_UNW_INTERNAL_ERROR

步进过程中发生了一些逻辑问题。请与 HP 支持联系。

错误

如果未提供足够的内存来执行初始化，或者有库交互问题（例如，调用服务管理器或 **dlmodinfo** ）阻止 **unwindlib** 执行初始化， **_UNW_currentContext** 将正常失败（即返回相应的错误）。

举例

初始化常规寄存器 5 至 400 的值:

```
#include <unwind.h>
main() {
   _Unwind_Context *uc;
   _UNW_ReturnCode retcode;
   uc = _UNW_createContextForSelf();
   if ((retcode = _UNW_setGR(uc,5,400)) != _UNW_OK) {
      /* Notify client: Initialization problem  */
   }
}
```

作者

_UNW_currentContext() 、 **_UNW_clear()** 、 **_UNW_jmpbufContext()** 、 **_UNW_setAR()** 、 **_UNW_setBR()** 、 **_UNW_setCFM()** 、 **_UNW_setFR()** 、 **_UNW_setGR()** 、 **_UNW_setGR_NaT()** 、 **_UNW_setIP()** 、 **_UNW_setPR()** 、 **_UNW_setPreds()** 、 **_UNW_step()** 、 **_UNW_FR_PhysicalNumber()** 、 **_UNW_GR_PhysicalNumber()** 和 **_UNW_PR_PhysicalNumber()** 由 HP 开发。

另请参阅

U_STACK_TRACE(3X)、 _UNW_createContextForSelf(3X)、 _UNW_getGR(3X)、 unwind(5)。

名称

_UNW_getGR() 、 _UNW_getAR() 、 _UNW_getAlertCode() 、 _UNW_getBR() 、 _UNW_getCFM() 、 _UNW_getFR()、 _UNW_getGR_NaT()、 _UNW_getIP()、 _UNW_getKernelSavedContext()、 _UNW_getPR()、 _UNW_getPreds()、 _UNW_clearAlertCode() - unwind 库数据结构中的查询值

概要

#include <unwind.h>

uint64_t _UNW_getGR(_Unwind_Context* p, uint32_t num);

_UNW_GR_Value _UNW_getGR_NaT(_Unwind_Context* p, uint32_t num);

_UNW_FR_Value _UNW_getFR(_Unwind_Context* p, uint32_t num);

uint64_t _UNW_getBR(_Unwind_Context* p, uint32_t num);

_UNW_getAR(_Unwind_Context* p, _UNW_AppReg num);

_UNW_Boolean _UNW_getPR(_Unwind_Context* p, uint32_t num);

uint64_t _UNW_getPreds(_Unwind_Context* p);

uint64_t _UNW_getIP(_Unwind_Context* p);

uint64_t _UNW_getCFM(_Unwind_Context* p);

_UNW_ReturnCode _UNW_getAlertCode(const _Unwind_Context *p);

void _UNW_clearAlertCode(_Unwind_Context *p);

说明

_UNW_getGR() 可返回参数 *p* 指向的 **_Unwind_Context** 对象中的已编号常规寄存器 *num* 的值。

_UNW_getGR_NaT() 可返回参数 *p* 指向的 **_Unwind_Context** 对象中的已编号常规寄存器 *num* 的值和 NaT 位。这些值是通过 **_UNW_GR_Value** 类型的结构返回的。该结构的 *value* 字段包括寄存器值，同时它的 *NaT* 字段包括寄存器 NaT 位。

_UNW_getFR() 可返回参数 *p* 指向的 **_Unwind_Context** 对象中的已编号浮点寄存器 *num* 的值。这些值是通过 **_UNW_FR_Value** 类型的结构返回的。该结构的 *first_container* 和 *second_container* 字段包含浮点寄存器 Spill/Fill 内存格式的两个连续双字。请参阅《Intel IA-64 Architecture Software Developer's Manual》“第 1 卷：IA-64 Application Architecture”“第 5.3 节：Floating Point Instructions”。

_UNW_getBR() 可返回参数 *p* 指向的 **_Unwind_Context** 对象中的已编号分支寄存器 *num* 的值。

_UNW_getAR() 可返回参数 *p* 指向的 **_Unwind_Context** 对象中的枚举应用程序寄存器 *num* 的值。枚举类型 **_UNW_AppReg** 用于访问应用程序寄存器。

_UNW_getPR() 可返回参数 *p* 指向的 **_Unwind_Context** 对象中的已编号谓词寄存器 *num* 的值。

_UNW_getPreds() 可返回参数 *p* 指向的 **_Unwind_Context** 对象中的所有谓词寄存器的值。返回值包含位中对应

于谓词寄存器编号的每个谓词寄存器的值 — 例如，位 63 包含谓词寄存器 63 的值。

getIP() 可返回参数 *p* 指向的 **_Unwind_Context** 对象中的指令指针的内容。

getCFM() 可返回参数 *p* 指向的 **_Unwind_Context** 对象中的当前帧标记的内容。

_UNW_getAlertCode() 可返回最近遇到的“客户需要知道的”返回代码。如果没有遇到“客户需要知道的”的情况，**_UNW_getAlertCode()** 将返回 **_UNW_OK** 。“客户需要知道的”返回代码是枚举 **_UNW_ReturnCode** 中除 **_UNW_OK** 以外的所有枚举符。

_UNW_clearAlertCode() 可清除参数 *p* 指向的 **_Unwind_Context** 对象中的任何“客户需要知道的”返回代码。

_UNW_getKernelSavedContext() 提供获取内核调试程序等工具的方法，以获取所需的与内核中断有关的某些信息。这些值是通过 **_UNW_KernelSavedContext** 类型的结构返回的，客户端可通过该结构获取两个字段，即格式 P10 辗转开解 (unwind) 描述符的 **p10_abi_value** 和 **p10_context_value** 。有关详细信息，请参考 *unwind*(5) 和《Itanium Processor Family Software Conventions and Runtime Architecture》。

返回值

如果成功完成，除 **_UNW_getAlertCode** 以外的其他寄存器值查询函数（带有前缀 **_UNW_get...** 的函数）将返回所请求的值，否则返回相当于 0 的值。

_UNW_getAlertCode 返回来自枚举 **_UNW_ReturnCode** 的值。请参阅 *unwind*(5) 上的“Error conditions and recovery”。

如果成功完成，**_UNW_clear** 将返回 **_UNW_OK** 。否则，当 **_Unwind_Context** 不为下列其中一种状态时，**_UNW_clear** 将返回 **_UNW_CLEAR_NOT_ALLOWED_IN_STATE** ： *Init* 、 *User_Interrupted_Frame* 、 *User_Sendsig_Frame* 、 *Kernel_Bottom_Frame* 、 *Frame* 。

举例

读取此过程帧的环境中的常规寄存器 5 的值；检查范围错误、查询错误等；最后清除警报代码：

```
#include <unwind.h>
_Unwind_Context *uc;
_UNW_ReturnCode retcode;
_UNW_GR_Value result;
uc = _UNW_createContextForSelf();
if (_UNW_OK != _UNW_currentContext(uc)) {
  /* Notify client: Initialization problem  */
}
else {
  result=_UNW_getGR_NaT(uc,5);
  switch (_UNW_getAlertCode(uc)) {
      case  _UNW_OK:
          break;
      case  _UNW_QUERY_RANGE_ERROR:
```

```
            /* Notify user of range error */
            break;
        default:
            /* Notify user of other error */
            break;
    }
}
_UNW_clearAlertCode(uc);
```

警告

请参阅 *unwind*(5) 上“客户端/库交互”一节中的“查询”小节中的表格，其中列出了何时保证查询函数返回的值可反映给定过程的处理器状态的实际寄存器值。例如，临时寄存器值只有在跨越与中断关联的堆栈帧后才有效。请参阅《Itanium Processor Family Software Conventions and Runtime Architecture》“第 5 章：Register Usage”。

作者

_UNW_getGR() 、 **_UNW_getAR()** 、 **_UNW_getAlertCode()** 、 **_UNW_getBR()** 、 **_UNW_getCFM()** 、 **_UNW_getFR()** 、 **_UNW_getGR_NaT()** 、 **_UNW_getIP()** 、 **_UNW_getKernelSavedContext()** 、 **_UNW_getPR()**、 **_UNW_getPreds()** 和 **_UNW_clearAlertCode()** 由 HP 开发。

另请参阅

U_STACK_TRACE(3X)、 _UNW_createContextForSelf(3X)、 _UNW_currentContext(3X)、 unwind(5)。

名称

a64l( )、l64a( ) - 在 long 整型和 Base-64 ASCII 字符串间进行转换

概要

#include <stdlib.h>

long int a64l(const char *s);

char *l64a(long int l);

过时的接口

int l64a_r(long int l, char *buffer, int buflen);

说明

这些函数用于维护存储在 *base-64* ASCII 字符中的数字。这是一种表示法，通过它 long 整型可以用最多 6 个字符来表示；每个字符表示 Radix-64 表示法中的一“位”。

用来表示“位”的字符中，用 **.** 表示 0， **/** 表示 1， **0** 到 **9** 表示 2-11， **A** 到 **Z** 表示 12-37，而 **a** 到 **z** 表示 38-63。

最左边的字符为最低有效位。例如，

$$a0 = (38 \times 64^{0}) + (2 \times 64^{1}) = 166$$

a64l() 接受一个指向以 null 结尾的 Base-64 表示形式的指针，并返回一个对应的 **long** 值。如果 *s* 所指向的字符串包含的字符多于 6 个，则 **a64l()** 使用头 6 个。

l64a() 接受一个 **long** 参数，并返回一个指向对应的 Base-64 表示形式的指针。如果参数为 0， **l64a()** 返回一个指向空字符串的指针。

过时的接口

l64a_r() 在 long 整型和 Base-64 ASCII 字符串间进行转换。

警告

l64a() 的返回值是一个指向缓冲区的指针，其内容被同一线程进行的后续调用覆盖。

l64a_r() 是过时接口，只是为了与现有 DCE 应用程序兼容才支持它。新的多线程应用程序应当使用 **l64a()** 。

另请参阅

thread_safety(5)。

符合的标准

a64l()：SVID2、SVID3

l64a()：SVID2、SVID3

名称

abort( ) - 生成软件异常中止错误

概要

#include <stdlib.h>

void abort(void);

说明

abort() 将首先关闭所有打开的文件、流、目录流及消息清单描述符（如果可能），然后将信号 **SIGABRT** 发送给调用进程。这样可能会导致生成核心转储（请参考 *signal*(2) ）。

如果捕获到信号 **SIGABRT** ，则执行相应的处理函数。如果处理函数有返回值，则随即将 **SIGABRT** 的操作重置为 **SIG_DFL** ，并再次向进程发送信号 **SIGABRT** ，以确保进程终止。

返回值

abort() 无返回值。

错误

未定义任何错误。

实际应用信息

SIGABRT 并非一定会被捕获。

诊断信息

如果既未捕获也未忽略 **SIGABRT** ，并且当前目录为可写目录，则将生成一个核心转储，并使用 Shell 写入消息 **abort - core dumped** 。

另请参阅

adb(1)、 exit(2)、 kill(2)、 signal(2)、 signal(5)、 thread_safety(5)。

符合的标准

abort()： AES、SVID2、SVID3、XPG2、XPG3、XPG4、FIPS 151-2、POSIX.1、ANSI C

名称

abs( )、labs( )、llabs( )、imaxabs( ) - 返回整数绝对值

概要

#include <stdlib.h>

int abs(int i);

long int labs(long int i);

long long llabs(long long i);

#include <inttypes.h>

intmax_t imaxabs(intmax_t i);

说明

abs() 计算其整数操作数的绝对值。

labs() 计算其 long 整型操作数的绝对值。

llabs() 计算其 long long 整型操作数的绝对值。

imaxabs() 计算其 intmax_t 整型操作数的绝对值。

如果无法显示结果值，则说明未定义这些函数的行为。

返回值

abs() 返回其整数操作数的绝对值。

labs() 返回其 long 整型操作数的绝对值。

llabs() 返回其 long long 整型操作数的绝对值。

imaxabs() 返回其 intmax_t 整数操作数的绝对值。

最大负整数将返回它本身。

警告

在这两个函数的补充表示形式中，无法显示最大负整数的绝对值。

另请参阅

imaxdiv(3C)、 floor(3M) 和 thread_safety(5)。

符合的标准

abs()：AES、SVID2、SVID3、XPG2、XPG3、XPG4、FIPS 151-2、POSIX.1 和 ANSI C

labs()：AES、SVID3、XPG4 和 ANSI C

llabs()：ISO/IEC 9899:1999 (C99), UNIX 03

imaxabs()：ISO/IEC 9899:1999 (C99), UNIX 03

名称

aclsort( ) - 对访问控制列表（仅适用于 JFS 文件系统）进行排序

概要

#include <sys/types.h>
#include <sys/aclv.h>

int aclsort(int nentries, int calclass, struct acl *aclbufp);

说明

aclsort() 例行程序可将 JFS 访问控制列表 (ACL) 条目排列成能使 *acl*(2) 系统调用接受的适当顺序。

aclbufp 指向包含 ACL 条目的缓冲区；如果 **calclass** 为非零值，那么它表示应重新计算 **CLASS_OBJ** 权限； **nentries** 指定缓冲区中的 ACL 条目数。

aclsort() 按以下方式排序 ACL 缓冲区的内容：

条目的顺序为 **USER_OBJ** 、 **USER** 、 **GROUP_OBJ** 、 **GROUP** 、 **CLASS_OBJ** 、 **OTHER_OBJ** 、 **DEF_USER_OBJ** 、 **DEF_USER** 、 **DEF_GROUP_OBJ** 、 **DEF_GROUP** 、 **DEF_CLASS_OBJ,** 和 **DEF_OTHER_OBJ** 。

将按数字 ID 的升序排列 **USER** 、 **GROUP** 、 **DEF_USER** 和 **DEF_GROUP** 类型的条目。

如果以下所有条件都成立， **aclsort()** 调用将会成功：

USER_OBJ 、 **GROUP_OBJ** 、 **CLASS_OBJ** 和 **OTHER_OBJ** 类型中的每一种正好都有一个条目。

DEF_USER_OBJ 、 **DEF_GROUP_OBJ** 、 **DEF_CLASS_OBJ** 和 **DEF_OTHER_OBJ** 类型中的每一种最多有一个条目。

USER 、 **GROUP** 、 **DEF_USER** 或 **DEF_GROUP** 类型的条目不可以包含重复的条目。重复的条目是指包含了相同数字 ID 的相同类型的条目，与其权限位无关。

如果 **calclass** 参数为零，并且不存在 **USER** 和 **GROUP** 类型的条目，则 **GROUP_OBJ** 条目和 **CLASS_OBJ** 条目的权限必须相同。

如果不存在 **DEF_USER** 和 **DEF_GROUP** 类型的条目，并且指定了 **DEF_GROUP_OBJ** 条目，那么还必须指定 **DEF_CLASS_OBJ** 条目，同时 **DEF_GROUP_OBJ** 条目和 **DEF_CLASS_OBJ** 条目的权限必须相同。

返回值

如果成功完成，则返回值为 0。如果存在重复的条目，则返回值为第一个重复条目的位置。如果 **USER_OBJ** 、 **GROUP_OBJ** 、 **CLASS_OBJ** 、 **OTHER_OBJ** 、 **DEF_USER_OBJ** 、 **DEF_GROUP_OBJ** 、 **DEF_CLASS_OBJ** 或 **DEF_OTHER_OBJ** 类型存在多个条目，那么会将这些条目视为重复的条目，并且返回值为第一个重复条目的位置。对于其他所有错误，返回值为 -1。

注释

在检查任何失败之前，需要按类型和 ID 对缓冲区排序。因此，缓冲区始终是已排序的，即使出现失败，也是如此。

出现失败时返回的重复条目的位置不是该重复条目与其基址的字节偏移量；该位置其实是指已排序的缓冲区中重复条目的条目编号。

将按条目类型的顺序执行检查。如果出现了多次失败，则将返回遇到的第一个失败，例如，如果 ACL 缓冲区包含一个重复的 **USER** 条目，并且不包含 **OTHER_OBJ** 条目，那么返回值将是第一个重复的 **USER** 条目。

将 ACL 传递给 *acl*(2) 之前，无须使用 **aclsort()** 对其排序。

相关内容

只有标准的 HP-UX 操作系统上的 JFS 文件系统才支持 **aclsort()** 。

作者

aclsort() 由 AT&T 开发。

另请参阅

acl(2)、 aclv(5)。

名称

acltostr() - 将访问控制列表 (ACL) 结构转换为字符串格式（仅适用于 HFS 文件系统）

概要

#include <acllib.h>

char *acltostr(int nentries, const struct acl_entry acl[], int form);

过时的接口

```
int acltostr_r(
  int nentries,
  const struct acl_entry acl[],
  int form,
  char *strbuf,
  int length);
```

备注

为确保持续符合新的行业标准，将来的发行版可能会更改此手册条目中描述的功能。

说明

acltostr() 可以将访问控制列表由结构格式转换为字符串表示形式。 **acltostr()** 使用指向 ACL 条目 (**acl**) 数组的第一个元素的指针，这些条目包含有效条目的指定数目 (*nentries*) （零个或多个），以及所需的输出格式 （**FORM_SHORT** 或 **FORM_LONG** ）。它可以返回一个指向静态字符串的指针（下一次调用会将其覆盖的指针），该字符串为 ACL 的符号表示形式，以空字符结尾。 *acl*(5) 中介绍了输出格式。在长格式中，返回的字符串包含换行符。

ACL_NSUSER 的用户 ID 和 **ACL_NSGROUP** 的组 ID 均由 **%** 表示。在使用 **ls** 命令（请参阅 *ls*(1) ）时，如果条目包含 **/etc/passwd** 或 **/etc/group** 中未列出的任何其他用户 ID 或组 ID 值， **acltostr()** 将返回与 ID 编号等效的代用字符串。

正如管理 **/etc/passwd** 文件的例行程序一样， **acltostr()** 将用户名和组名截断为八个字符。

注释： **acltostr()** 在功能上是对 **strtoacl()** 的补充。

过时的接口

acltostr_r() 可将访问控制列表 (ACL) 结构转换为字符串格式。

返回值

如果 **acltostr()** 成功，则返回指向以空字符结尾的字符串的指针。如果 *nentries* 小于或等于零，则字符串的长度为零。如果 *nentries* 大于 **NACLENTRIES** （在 **<sys/acl.h>** 中定义），或者 *form* 为无效值，则调用返回 (char *) NULL。

如果 **acltostr_r()** 成功，则返回 0。如果失败，则返回 -1，并设置 **errno** 。如果 *nentries* 小于或等于零，则字符串的长度为零。

错误

[EINVAL] *strbuf* 等于 NULL，或者 *nentries* 大于 **NACLENTRIES** （在 **<sys/acl.h>** 中定义），或者 *form* 不是有效格式之一，或者 *strbuf* 的 *length* 太短。

[ERANGE] *length* 小于或等于零。

举例

以下代码段读取文件 **/users/ggd/test** 上的 ACL 后，输出其短格式表示形式。

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

int nentries;
struct acl_entry acl [NACLENTRIES];

if ((nentries = getacl ("/users/ggd/test", NACLENTRIES, acl)) < 0)
    error (...);

fputs (acltostr (nentries, acl, FORM_SHORT), stdout);
```

警告

acltostr() 返回的值是指向缓冲区的指针，同一线程对 **acltostr()** 执行后续调用时，将覆盖此缓冲区的内容。

acltostr_r() 是已过时接口，支持它仅为实现与现有 DCE 应用程序的兼容。新的多线程应用程序应该使用 **acltostr()** 。

相关内容

只有标准的 HP-UX 操作系统上的 HFS 文件系统才支持 **acltostr()** 。

作者

acltostr() 由 HP 开发。

文件

/etc/passwd
/etc/group

另请参阅

getacl(2)、 setacl(2)、 cpacl(3C)、 chownacl(3C)、 setaclentry(3C)、 strtoacl(3C)、 acl(5)、 thread_safety(5)。

名称

acos( )、acosf( )、acosl( )、acosw( )、acosq( ) - 反余弦函数

概要

```
#include <math.h>

double acos(double x);

float acosf(float x);
```

仅适用于 **HP Integrity** 服务器

```
long double acosl(long double x);

extended acosw(extended x);

quad acosq(quad x);
```

说明

acos() 返回 *x* 的反余弦值，其范围为从 0 到 π。

acosf() 是 **acos()** 的 **float** 版本；它采用 **float** 参数，并返回 **float** 结果。

仅适用于 **Integrity** 服务器

acosl() 是 **acos()** 的 **long double** 版本；它采用 **long double** 参数，并返回 **long double** 结果。

acosw() 是 **acos()** 的 **extended** 版本；它采用 **extended** 参数，并返回 **extended** 结果。

在 HP-UX 系统中， **acosq()** 等效于 **acosl()** 。

用法

要使用这些函数，请使用缺省的 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。

（对于 Integrity 服务器）要使用 **acosw()** 或 **acosq()** ，也可以使用 **-fpwidetypes** 选项进行编译。

要使用上面的任何函数，请确保程序包含 **<math.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

有关详细信息，请参阅《HP-UX floating-point guide for HP Integrity servers》（位于以下站点：**http://www.hp.com/go/fp** ）。

仅适用于 **PA-RISC** 系统

可以使用 Millicode 版的 **acos()** 和 **acosf()** 函数。 Millicode 版的数学库函数通常比标准库中对应的函数运行速度快。要使用这些版本，请使用 **+Olibcalls** 或 **+Oaggressive** 优化选项编译程序。

特殊情况下，Millicode 版本返回在“返回值”一节中描述的值，但不设置 **errno** 。

返回值

acos() 返回 +0。

如果 *x* 的值大于 1，则 **acos()** 将返回 NaN 并引起无效异常。

如果 *x* 为 NaN，则 **acos()** 将返回 NaN。

当未引起其他异常时， **acos()** 是否会引起不精确的异常是不确定的。

错误

如果 *x* 的值大于 1， **acos()** 会将 **errno** 设置为 [EDOM]。

仅适用于 **Integrity** 服务器

缺省情况下，Integrity 服务器上的 HP-UX **libm** 函数不设置 **errno** 。对于 **errno** 设置，可以使用 **+Olibmerrno** 选项进行编译。

另请参阅

acosd(3M)、 asin(3M)、 atan(3M)、 atan2(3M)、 cacos(3M)、 cos(3M)、 sin(3M)、 tan(3M)、 math(5)。

符合的标准

acos() ： SVID3、XPG4.2、ANSI C 和 ISO/IEC C99（包括附件 F“IEC 60559 floating-point arithmetic”）

acosf() 和 **acosl()** ： ISO/IEC C99（包括附件 F“IEC 60559 floating-point arithmetic”）

名称

acosd( )、acosdf( )、acosdl( )、acosdw( )、acosdq( ) - 以度为单位的反余弦函数

概要

#include <math.h>

double acosd(double x);

float acosdf(float x);

仅适用于 **HP Integrity** 服务器

long double acosdl(long double x);

extended acosdw(extended x);

quad acosdq(quad x);

说明

acosd() 返回以度为单位的 *x* 的反余弦值，范围从 0 到 180。

acosdf() 是 **float** 版的 **acosd()** ；它采用 **float** 参数，并返回 **float** 结果。

仅适用于 **Integrity** 服务器

acosdl() 是 **long double** 版的 **acosd()** ；它采用 **long double** 参数，并返回 **long double** 结果。

acosdw() 是 **extended** 版的 **acosd()** ；它采用 **extended** 参数，并返回 **extended** 结果。

acosdq() 等效于 HP-UX 系统上的 **acosdl()** 。

用法

要使用这些函数，请使用缺省的 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。

（对于 Integrity 服务器）要使用 **acosdw()** 或 **acosdq()** ，也可以使用 **-fpwidetypes** 选项进行编译。

要使用上面的任何函数，请确保程序包含 **<math.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

有关详细信息，请参阅《HP-UX floating-point guide for HP Integrity servers》（位于以下站点：**http://www.hp.com/go/fp** ）。

返回值

acosd() 返回 +0。

如果 *x* 的量值大于 1，则 **acosd()** 将返回 NaN 并引发无效异常。

如果 *x* 为 NaN，则 **acosd()** 将返回 NaN。

错误

如果 *x* 的量值大于 1，则 **acosd()** 会将 **errno** 设置为 [EDOM]。

仅适用于 **Integrity** 服务器

缺省情况下，Integrity 服务器上的 HP-UX libm 函数不设置 **errno** 。对于 **errno** 设置，可以使用 **+Olibmerrno** 选项进行编译。

另请参阅

acos(3M)、 asind(3M)、 atand(3M)、 atan2d(3M)、 cosd(3M)、 sind(3M)、 tand(3M)、 atan2d(3M)、 math(5)。

符合的标准

任何标准均未指定这些函数。

名称

acosh( )、acoshf( )、acoshl( )、acoshw( )、acoshq( ) - 反双曲余弦函数

概要

#include <math.h>

double acosh(double x);

仅适用于 **HP Integrity** 服务器

float acoshf(float x);

long double acoshl(long double x);

extended acoshw(extended x);

quad acoshq(quad x);

说明

acosh() 返回 *x* 的反双曲余弦值，范围从 +0 到正无穷大。

仅适用于 **Integrity** 服务器

acoshf() 是 **float** 版的 **acosh()** ；它采用 **float** 参数，并返回 **float** 结果。

acoshl() 是 **long double** 版的 **acosh()** ；它采用 **long double** 参数，并返回 **long double** 结果。

acoshw() 是 **extended** 版的 **acosh()** ；它采用 **extended** 参数，并返回 **extended** 结果。

acoshq() 等效于 HP-UX 系统上的 **acoshl()** 。

用法

（对于 Integrity 服务器）要使用 **acoshf()** 、 **acoshl()** 、 **acoshw()** 或 **acoshq()** ，请使用缺省的 **-Ae** 选项或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。

（对于 Integrity 服务器）要使用 **acoshw()** 或 **acoshq()** ，也可以使用 **-fpwidetypes** 选项进行编译。

要使用上面的任何函数，请确保程序包含 **<math.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

有关详细信息，请参阅《HP-UX floating-point guide for HP Integrity servers》（位于以下站点：**http://www.hp.com/go/fp** ）。

返回值

acosh() 返回 +0。

如果 $x < 1.0$， **acosh()** 将返回 NaN 并引发无效异常。

如果 *x* 为正无穷大， **acosh()** 将返回正无穷大。

如果 *x* 为 NaN， **acosh()** 将返回 NaN。

如果未引发其他异常，则说明没有指定 **acosh()** 是否引发不精确异常的这种情况。

错误

如果 $x < 1.0$， **acosh()** 就会将 **errno** 设置为 [EDOM]。

仅适用于 **Integrity** 服务器

缺省情况下，Integrity 服务器上的 HP-UX **libm** 函数不设置 **errno** 。对于 **errno** 设置，可以使用 **+Olibmerrno** 选项进行编译。

另请参阅

asinh(3M)、 atanh(3M)、 cacosh(3M)、 cosh(3M)、 math(5)。

符合的标准

acosh() 、 **acoshf()** 、 **acoshl()** ： ISO/IEC C99（包括附件 F“IEC 60559 floating-point arithmetic”）

名称

ACPS - 访问控制策略交换

概要

#include <acps.h>
#include <acps_api.h>
#include <acps_spi.h>

cc [*flag*]... *file*... **-lacps** [*library*]...

说明

通过解释某种格式的预配置策略，访问控制策略交换 (ACPS) 可在必须作出授权决策的应用程序与提供决策响应的基础模块之间提供分隔层。该交换提供三个接口：

ACPS API 应用程序编程接口，支持访问控制的应用程序可以调用该接口请求授权决策。有关详细信息，请参阅 *acps_api*(3) 。

ACPS SPI 服务提供程序接口 (SPI)，该接口允许自定义模块提供访问控制决策。模块可以写入 SPI 以插入调用该 API 的应用程序。有关详细信息，请参阅 *acps_spi*(3) 。

acps.conf 配置接口（文件配置），管理员在部署时使用该接口选择引用的模块（或模块组）以作出授权决策。有关详细信息，请参阅 *acps.conf*(4) 。

上述的每个接口都在其各自的联机帮助页中进行了更为详细的描述。

访问请求基础

ACPS 框架可识别访问控制请求的以下三个基础组件：

主体 尝试访问资源的实体。在操作系统环境中，主体通常是用户或与用户关联的进程。

操作 对资源执行的操作。操作可以直接与应用程序或命令相对应。对于 HP-UX RBAC，操作是以点分隔的具有层次结构的字符串，例如 **hpux.user.add** 。

对象 操作的目标。这通常与最终资源相同。

除了这些核心组件外，有时还包括能影响访问控制决策的主体、操作、对象或环境的其他属性。 ACPS 框架的设计目的是使应用程序能够适当地指定其他属性，并且将此信息传递给配置为回复请求的模块。

通过 ACPS 传递的最后一条信息是一种凭证，证明主体实际上是用户要求成为的角色。这种称为凭据的凭证可以有几种不同格式，例如口令或 Kerberos 凭证。根据决策提供程序及其与应用程序确立的信任关系，凭据可能是可选的。

参数类型

ACPS 框架的设计目的是使应用程序能够将各种属性传递给决策提供程序。为便于普遍了解属性和组件的可能类型，ACPS 框架要求大部分参数带有一个可能隐式或显式指定的关联类型。

这些类型通用于应用程序接口 (API) 和决策提供程序接口 (SPI)，其定义如下：

主体类型

ACPS_ID_NAME	与主体关联的 HP-UX 用户名。
ACPS_ID_RFC822	主体的 RFC822 标识符（例如， **user@hp.com** ）。
ACPS_ID_UID	编码为 **char *** 的与主体关联的 HP-UX UID。
ACPS_ID_X500	主体的 X.500 标识符（即， **LDAP DN** ）。

凭据类型

ACPS_CRED_KERBTICKET	以 ASN.1 DER 编码的 Kerberos 凭证。
ACPS_CRED_PASSWORD	明文口令。
ACPS_CRED_PIN	表示为字符串的明文 Pin（个人识别号）。
ACPS_CRED_SAML	SAML 凭据断言。
ACPS_CRED_X509CERT	Base64 ASCII 编码的证书。

主体属性

ACPS_SUBATTR_ASSIGNEDROLES	与主体关联的（活动）角色列表，以逗号分隔。
ACPS_SUBATTR_GECOS	与主体关联的 GECOS 信息，如 passwd 文件或其他名称服务交换 **'passwd'** 储备库中所定义。
ACPS_SUBATTR_GROUPS	**ACPS_SUBATTR_ASSIGNEDROLES** 与主体关联的组列表，以逗号分隔。
ACPS_SUBATTR_SHELL	与主体关联的 Shell，如 passwd 文件或其他 NSS **'passwd'** 储备库中所定义。

操作类型

ACPS_OP_BASIC	文字编码的操作字符串。
ACPS_OP_DOTHEIRARCHICAL	使用“.”作为分隔符的按层次结构编码的操作字符串。

对象类型

ACPS_OBJ_CIM28	以 XML 编码的 CIM 版本 2.8 对象表示。
ACPS_OBJ_FILEPATH	编码为路径的文件对象。
ACPS_OBJ_GENERIC	编码为通用字符串的对象，无其他解释。
ACPS_OBJ_URI	以 uri 语法编码的对象。

环境属性

CPS_ENVATTR_COMPARTMENT	与访问控制请求关联的隔离专区标记。对“关联”的解释由应用程序来完成。

返回值

ACPS_ALLOW	已允许所请求的访问。此返回代码仅作为 **checkauth()** 请求的结果返回。
ACPS_CONFIG_ERROR	ACPS 的配置不正确。这可能是由于 **acps.conf** 文件中存在句法错误。
ACPS_DENY	已拒绝所请求的访问。此返回代码仅作为 **checkauth()** 请求的结果返回。
ACPS_GEN_ERROR	所请求的操作失败。当更具体的错误代码不适用时，此值是用于所有操作的缺省错误代码。当此值作为 **checkauth()** 调用的结果返回时，应用程序应该不允许访问指定资源。
ACPS_MEMORY_ERROR	由于分配或取消分配内存时出错，因此所请求的操作失败。
ACPS_NEED_AUTHENTICATION	由于缺少必需凭据，因此拒绝所请求的访问。提供所请求的凭据后，应用程序可以再次调用 **checkauth()** ，这样便可能允许访问。如果应用程序无法提供必需的凭据，则应当将此结果视为等效于 **ACPS_DENY** 。
ACPS_NOINFO	储备库没有关于用户的访问控制信息。此返回代码仅作为 **acpm_checkauth()** 请求的结果返回，并且永远不会返回到应用程序。
ACPS_SUCCESS	所请求的操作成功。此返回代码永远不会作为 **checkauth()** 请求的结果返回。

另请参阅

acps_api(3)、 acps_spi(3)、 acps.conf(4)、 rbac(5)。

名称

acps_api： acps_addenvattr()、 acps_addobjattr()、 acps_addopattr()、 acps_addsubattr()、 acps_addsubcred()、 acps_checkauth()、 acps_end()、 acps_setobj()、 acps_setop()、 acps_setsubid()、 acps_start() - ACPS 应用程序编程接口

概要

#include <acps.h>

#include <acps_api.h>

cc [*flag*]... *file*... **-lacps** [*library*]...

int acps_addenvattr(acp_handle_t *h***, char ****type***, char ****attr***);**

int acps_addobjattr(acp_handle_t *h***, char ****type***, char ****attr***);**

int acps_addopattr(acp_handle_t *h***, char ****type***, char ****attr***);**

int acps_addsubattr(acp_handle_t *h***, char ****type***, char ****attr***);**

int acps_addsubcred(acp_handle_t *h***, char ****type***, char ****attr***);**

int acps_checkauth(acp_handle_t *h***);**

int acps_end(acp_handle_t *h***);**

int acps_setobj(acp_handle_t *h***, char ****type***, char ****object***);**

int acps_setop(acp_handle_t *h***, char ****type***, char ****operation***);**

int acps_setsubid(acp_handle_t *h***, char ****type***, char ****ident***);**

int acps_start(acp_handle_t *h***);**

说明

ACPS 应用程序编程接口 (API) 定义了一组函数，应用程序编程人员可以使用这些函数执行访问控制检查。这种检查通常用于确定某个特定的用户对于某个资源是否拥有访问权限。

ACPS API 提供了一个简化的接口，用于请求访问权限检查；同时还提供了一个较为灵活的接口，该接口允许应用程序指定其他的访问控制属性。对于这两个接口，基本的访问控制请求都采用下面的形式：

Is subject X allowed to perform operation Y on object Z?

此请求是使用不同的函数构成的，具体取决于应用程序的要求。

简化的接口

在简化的接口中，此基本问题的表现形式如下：

int acps_simplecheckauth(char *ident, char *operation, char *object);

对于一个标准的请求，此查询将根据配置的模块中提供的策略返回 **ACPS_ALLOW** 或 **ACPS_DENY** 。对于很多应用程序来说，这种查询方式已经足够了，但是该查询会对用于用户身份、操作和对象的表示类型进行若干简单

的假定。尤其对于此请求，身份被假定为用户名 (**ACPS_ID_NAME**)，操作被假定为典型的 HP-UX 操作字符串 (**ACPS_OP_DOTHEIRARCHICAL**)，对象则被假定为通用字符串 (**ACPS_OBJ_GENERIC**)。有关这些类型的详细信息，请参阅 *acps*(3)。

详细的接口

对于更为复杂的请求，该查询会采用“概要”中所述的形式： **acps_addenvattr()** 、 **acps_addobjattr()** 、 **acps_addopattr()** 、 **acps_addsubattr()** 、 **acps_addsubcred()** 、 **acps_checkauth()** 、 **acps_end()** 、 **acps_setobj()**、 **acps_setop()** 、 **acps_setsubid()** 、 **acps_start()** 。

该组接口包含几个主要组件。

acps_start() 和 **acps_end()** 函数用于分配和取消分配内存，以便在调用之间维护状态。传递到每个调用的 **acp_handle_t_t** 是一个指向不透明数据结构的指针。此结构的内存作为对 **acps_start()** 调用的一部分进行分配，然后再在 **acps_end()** 中释放。

对 **acps_setX()** 和 **acps_addX()** 的后续调用实质上会将信息编码到句柄中，以准备调用 **acps_checkauth()** 。这五部分编码信息如下：

Identity	主题标识（例如， 用户名、 *uid* 或 X.500 DN），以及关联的属性（例如，角色、组）。
Credentials	主题凭据（例如，口令和（或）Kerberos 标记）。
Operation	操作以及关联的属性。
Object	对象以及关联的属性。
Environment	环境属性（例如，隔离专区 ID、处理器使用情况等）。

对于此信息， **acps_setX()** 接口会将一个单独的值编码到句柄中。对同一接口的后续调用将覆盖以前写入的值。这与 **acps_addX()** 接口正好相反，后者会将该值附加到值链。 **acps_setX()** 接口用于唯一地标识访问控制请求的值。 **acps_addX()** 接口用于凭据和属性，可能会显示任意数量的凭据和属性。

对 **acps_checkauth()** 的调用会根据 ACPS 配置，将授权请求封送到后端模块。后端模块可以检索存储在 *acps_spi*(3) 指定的句柄中的信息。

还有一个可选接口没有在上述代码段中显示，当某个应用程序收到来自 **acps_checkauth()** 调用的 **ACPS_NEED_AUTHENTICATION** 时，该应用程序可以选择调用该接口：

char *acps_getreqcred(acp_handle_t h);

此函数返回一个字符串，指明策略所需的凭据类型。应用程序可以使用此接口向用户请求其他凭据（如果需要）。然后该应用程序可以使用 **acps_addsubcred()** 将这些凭据添加到句柄，然后再次调用 **acps_checkauth** 。

ACPS API 只接受以 C 语言环境编码的字符串。

返回值

ACPS API（以及 ACPS SPI）返回的值在 *acps*(3) 中进行了定义。

举例

下面的示例说明了如何在示例应用程序中使用简化的接口：

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

int
main(int argc, char **argv)
{
  int ret;

  if(argc != 4) {
      fprintf(stderr, "Usage:  chkauth username operation object\n");
      exit(1);
  }

  ret = acps_simplecheckauth(argv[1], argv[2], argv[3]);

  if(ret == ACPS_ALLOW) {
      printf("Access Allowed\n");
  }
  else if(ret == ACPS_DENY) {
      printf("Access Denied\n");
  }
  else {
      printf("Error returned: %d\n", ret);
  }

  return ret;
}
```

另请参阅

acps(3)、 acps_spi(3)。

名称

acps_spi : acpm_getenvattrs() 、 acpm_getobj() 、 acpm_getobjattrs() 、 acpm_getop() 、 acpm_getopattrs() 、 acpm_getsubattrs()、 acpm_getsubcreds()、 acpm_getsubid() - ACPS 服务提供程序接口

概要

#include <acps.h>
#include <acps_spi.h>

cc [*flag*]... *file*... **-lacps** [*library*]...

int acpm_getenvattrs(acp_handle_t *h***, acp_attr_node ****head***);**

int acpm_getobj(acp_handle_t *h***, char *****type***, char ** object);**

int acpm_getobjattrs(acp_handle_t *h***, acp_attr_node *****head***);**

int acpm_getop(acp_handle_t *h***, char *****type***, char *****operation***);**

int acpm_getopattrs(acp_handle_t *h***, acp_attr_node *****head***);**

int acpm_getsubattrs(acp_handle_t *h***, acp_attr_node *****head***);**

int acpm_getsubcreds(acp_handle_t *h***, acp_attr_node *****head***);**

int acpm_getsubid(acp_handle_t *h***, char *****type***, char *****id***);**

说明

使用 ACPS 服务提供程序接口，可以定义对访问控制请求进行响应的自定义模块。通常写入此模块以强制执行自定义策略，或与定义此类策略的其他系统连接。通过创建此模块并将适当的条目插入到 **acps.conf** 文件，由支持访问控制策略切换 (Access Control Policy Switch) 的应用程序发出的所有访问控制请求将自动路由到该模块，而无需修改应用程序。

服务提供程序（模块）接口主要由明确定义的 **acpm_checkauth()** 例行程序和一组帮助例行程序组成，前者是每个模块必须提供的，后者是切换提供的，用于将信息编码或解码到不透明的句柄中。

由每个模块提供的单个接口定义如下：

int acpm_checkauth(acp_handle_t h, int argc, const char **argv);

句柄包含所有请求信息。 *argv* 参数包含 **char *** 元素的数组，每个元素都表示在 ACPS 配置文件条目中为模块指定的参数（请参阅下文）。模块返回值与 API 返回值精确匹配，其中附加值选项指明给定请求 (**ACPS_NOINFO**) 不存在任何可用的访问信息。

为使模块从 句柄中检索有关访问控制请求的信息，切换提供下列一组 **acpm*** 帮助程序例行程序来提取所需信息，如“概要”中所述： **acpm_getenvattrs()** 、 **acpm_getobj()** 、 **acpm_getobjattrs()** 、 **acpm_getop()** 、 **acpm_getopattrs()** 、 **acpm_getsubattrs()** 、 **acpm_getsubcreds()** 、 **acpm_getsubid()** 。

请注意，这些例行程序与 API 例行程序非常相似，不同的是多值属性返回为链接的列表，而不是需要单个的 **acpm_getX()** 例行程序。这种情况的实际原因是：在根据模块的需要处理属性数据时，所允许的灵活性更大。

与应用程序可以检索模块所请求的凭据的方式相同（请参阅 *acps_api*(3)），模块采用下列接口将该信息编码到句柄中：

int acpm_setreqcred(acp_handle_t h, char *cred);

当应用程序不存在所需凭据时，模块通常会使用此代码。应用程序可能会添加必需的信息并重复对 **acps_checkauth()** 的调用。

返回值

由 ACPS SPI（以及 ACPS API）返回的值在 *acps*(3) 中进行定义。

举例

下例说明强制执行该策略的策略模块示例：

"users Ron, Ren, and Bill may read or write the password object"

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

extern int acpm_checkauth(acp_handle_t h, int argc, const char **argv)
{
   char *user;
   char *operation;
   char *object;
   char *buf;
   int retval;

   // get the user
   if((retval = acpm_getsubid(h, buf, &user)) != ACPS_SUCCESS)
      return retval;
   if(strcmp(buf, ACPS_ID_NAME) != 0) // type validation
      return ACPS_GEN_ERROR;

   // get the operation
   if((retval=acpm_getop(h, buf, &operation)) != ACPS_SUCCESS)
      return retval;
   if(strcmp(buf, ACPS_OP_DOTHEIRARCHICAL) != 0)
      return ACPS_GEN_ERROR;

   // get the object
   if((retval = acpm_getobj(h, buf, &object)) != ACPS_SUCCESS)
      return retval;
   if(strcmp(buf, ACPS_OBJ_GENERIC) != 0)
```

```
        return ACPS_GEN_ERROR;

    // evaluate primitives against policy
    if( (   (strcmp(user, "Ron") == 0)
        || (strcmp(user, "Ren") == 0)
        || (strcmp(user, "Bill") == 0))
      &&(   (strcmp(operation, "read") == 0)
        || (strcmp(operation, "write") == 0))
      &&(strcmp(object, "password"))) {
          return ACPS_ALLOW;
    }
    else {
        return ACPS_DENY;
    }
}
```

另请参阅

acps(3)、 acps_api(3)。

名称

addch、mvaddch、mvwaddch 和 waddch — 向窗口添加单字节字符和呈现方式并向前移动光标

概要

#include <curses.h>

int addch(const chtype *ch***);**

int mvaddch(int *y***, int** *x***, const chtype** *ch***);**

int mvwaddch(WINDOW ****win*, int** *y***, int** *x***, const chtype** *ch***);**

int waddch(WINDOW ****win*, const chtype** *ch***);**

说明

addch() 、 **mvaddch()** 、 **mvwaddch()** 和 **waddch()** 函数将 *ch* 放入当前窗口或指定窗口的当前或指定位置，然后向前移动窗口光标位置。这些函数执行换行。这些函数执行特殊字符处理。

返回值

成功完成后，这些函数返回 OK。否则，返回 ERR。

错误

没有定义任何错误。

实际应用信息

这些函数只保证对这样的字符集进行可靠操作：其中的每个字符都属于单字节，只能用带有 A_ 前缀的常量来表示其属性。

另请参阅

add_wch(3X)、 attroff(3X)、 doupdate(3X)、 curses_intro(3X)、参阅输出到窗口中的字符的呈现方式一节；<curses.h>。

历史变更记录

在 X/Open Curses 第 2 期中首次发布。

X/Open Curses 第 **4** 期

为清楚起见，重新编写了该条目。同时，参数 *ch* 的类型已从 **chtype** 更改为 **const chtype**。

名称

addchnstr、mvaddchnstr、mvwaddchnstr、 waddchnstr — 向窗口中添加有限长度的单字节字符串和显示形式

概要

#include <curses.h>

int addchnstr(const chtype ****chstr*, int** *n***);**

int mvaddchnstr(int *y***, int** *x***, const chtype ****chstr***, int** *n***);**

int mvwaddchnstr(WINDOW ****win*, int** *y***, int** *x***, const chtype ****chstr***,**
int *n***);**

int waddchnstr(WINDOW ****win*, const chtype ****chstr***, int** *n***);**

说明

这些函数将用 *chstr* 所指向的数组的内容覆盖当前窗口或指定窗口的内容，起止位置如下：从当前位置或指定的位置开始，直到在 *chstr* 所指向的数组中遇到空的 **chtype** 为止。

这些函数不会更改光标位置。这些函数不执行特殊字符处理操作。这些函数不执行换行操作。

这些函数最多可复制 *n* 项，但不能超过行上所能容纳的项数。如果 *n* 为 –1，则复制整个字符串，但复制的最大字符数不超过行上能容纳的字符数。

返回值

成功完成后，这些函数将返回 **OK** 。否则，它们将返回 **ERR** 。

错误

未定义任何错误。

实际应用信息

这些函数只确保对特定的字符集进行可靠的操作，在这些字符集中，每个字符都是单字节字符，其属性只能使用前缀为 A_ 的常量来表示。

另请参阅

addch(3X)、 add_wch(3X)、 add_wchstr(3X)、 <curses.h>。

历史变更记录

在 X/Open Curses 第 4 期中首次发布。

名称

addchstr、mvaddchstr、mvwaddchstr、waddchstr — 向窗口添加单字节字符串和呈现方式

概要

#include <curses.h>

int addchstr(const chtype ****chstr*);**

int mvaddchstr(int *y***, int** *x***, const chtype ****chstr***);**

int mvwaddchstr(WINDOW ****win*, int** *y***, int** *x***, const chtype ****chstr***);**

int waddchstr(WINDOW ****win*, const chtype ****chstr***);**

说明

这些函数使用 *chstr* 指向的数组的内容覆盖当前窗口或指定窗口的内容，从当前位置或指定的位置开始，直到在 *chstr* 指向的数组中遇到空 **chtype** 为止。

这些函数不会更改光标位置。这些函数不执行特殊字符处理。这些函数不执行换行操作。

返回值

成功完成后，这些函数返回 **OK** 。否则，返回 **ERR** 。

错误

没有定义任何错误。

实际应用信息

这些函数只保证对这样的字符集进行可靠操作：其中的每个字符都属于单字节，只能用带有 A_ 前缀的常量来表示其属性。

另请参阅

addch(3X)、 addchnstr(3X)、 add_wch(3X)、 add_wchnstr(3X)、 <curses.h>。

历史变更记录

在 X/Open Curses 第 4 期中首次发布。

名称

addnstr、 addstr、 mvaddnstr、 mvaddstr、 mvwaddnstr、 mvwaddstr、 waddnstr 和 waddstr - 向窗口添加多字节字符串（不带呈现方式）并向前移动光标

概要

#include <curses.h>

int addnstr(const char ****str*, int** *n***);**

int addstr(const char ****str*);**

int mvaddnstr(int *y***, int** *x***, const char ****str***, int** *n***);**

int mvaddstr(int *y***, int** *x***, const char ****str***);**

int mvwaddnstr(WINDOW ****win*, int** *y***, int** *x***, const char ****str***, int** *n***);**

int mvwaddstr(WINDOW ****win*, int** *y***, int** *x***, const char ****str***);**

int waddnstr(WINDOW ****win*, const char ****str***, int** *n***);**

int waddstr(WINDOW ****win*, const char ****str***);**

说明

这些函数使用后台呈现方式，从当前或指定位置开始将字符串 *str* 写在当前或指定窗口。

这些函数会使光标位置向前移动。这些函数执行特殊字符处理操作。这些函数执行换行。

addstr() 、 **mvaddstr()** 、 **mvwaddstr()** 和 **waddstr()** 函数类似于对 *str* 调用 **mbstowcs()** ，然后再分别调用 **addwstr()** 、 **mvaddwstr()** 、 **mvwaddwstr()** 和 **waddwstr()** 。

addnstr() 、 **mvaddnstr()** 、 **mvwaddnstr()** 和 **waddnstr()** 函数使用从 *str* 开始的至多 *n* 个字节。当 *n* 为 –1 时，这些函数添加整个字符串。这些函数类似于对 *str* 的前 *n* 个字节调用 **mbstowcs()** ，然后再分别调用 **addwstr()** 、 **mvaddwstr()** 、 **mvwaddwstr()** 和 **waddwstr()** 。

返回值

成功完成后，这些函数返回 **OK** 。否则，返回 **ERR** 。

错误

没有定义任何错误。

另请参阅

addnwstr(3X) 、 mbstowcs() （位于《X/Open System Interfaces and Headers, Issue 4, Version 2》规范中）、<curses.h>。

历史变更记录

在 X/Open Curses 第 4 期中首次发布。

在 X/Open Curses 第 3 期中， **addstr()** 、 **mvaddstr()** 、 **mvwaddstr()** 和 **waddstr()** 函数在 **addstr()** 条目中进行了描述。在 X/Open Curses 第 4 期中，对这些函数所定义的 *str* 参数的类型由 **char *** 更改为 **char *const** ，并且更改

了说明部分，指明这些函数将正确处理多字节序列。

名称

addnwstr、addwstr、mvaddnwstr、mvaddwstr、mvwaddnwstr、mvwaddwstr、 waddnwstr、waddwstr — 向窗口中添加宽字符字符串并向前移动光标

概要

#include <curses.h>

int addnwstr(const wchar_t *_wstr_, int _n_);

int addwstr(const wchar_t *_wstr_);

int mvaddnwstr(int _y_, int _x_, const wchar_t *_wstr_, int _n_);

int mvaddwstr(int _y_, int _x_, const wchar_t *_wstr_);

int mvwaddnwstr(WINDOW *_win_, int _y_, int _x_, const wchar_t *_wstr_, int _n_);

int mvwaddwstr(WINDOW *_win_, int _y_, int _x_, const wchar_t *_wstr_);

int waddnwstr(WINDOW *_win_, const wchar_t *_wstr_, int _n_);

int waddwstr(WINDOW *_win_, const wchar_t *_wstr_);

说明

这些函数将在当前窗口或指定窗口的当前光标位置或指定的光标位置处写入宽字符字符串 *wstr* 的字符。

这些函数将使光标位置向前移动。这些函数执行特殊字符处理操作。这些函数可执行换行操作。

其作用类似于根据 **wchar_t** 和背景显示形式来构建 **cchar_t**，以及为字符串中的每个 **wchar_t** 字符调用一次 **wadd_wch()**。由 *mv* 函数所指定的光标移动将仅在操作开始时执行一次。

addnwstr()、 **mvaddnwstr()**、 **mvwaddnwstr()** 和 **waddnwstr()** 函数最多可写入 *n* 个宽字符。如果 *n* 为 –1，则将添加整个字符串。

返回值

成功完成后，这些函数将返回 **OK**。否则，它们将返回 **ERR**。

错误

未定义任何错误。

另请参阅

add_wch(3X)、 <curses.h>。

历史变更记录

在 X/Open Curses 第 4 期中首次发布。

名称

addsev() - 为格式化例行程序定义附加的严重性

概要

#include <pfmt.h>

int addsev(int sev, const char *sev_string);

说明

通过 **addsev()** 例行程序，用户可以标准的消息格式定义供格式化例行程序使用的附加严重性（请参阅 *pfmt*(3C)）。 *sev* 为严重性级别。它必须介于 5 和 255 之间（包括 5 和 255）。 *sev_string* 是一个与此严重性级别关联的字符串。如果格式化例行程序的 *flags* 参数的严重性组成部分与 *sev* 相匹配，则 *sev_string* 将输出为严重性字符串。

可多次调用 **addsev()** 例行程序，以建立关联的列表。如果已设置 *sev* ，则 *sev_string* 将覆盖前一字符串。如果 *sev_string* 为 NULL，则从列表中删除此关联。

addsev() 假定已使用当前语言环境将 *sev_string* 转换为语言环境特定的字符串。

返回值

如果成功， **addsev()** 将返回 0。否则，它将返回 **–1** 。

举例

```
#define MM_USER 10
addsev(MM_USER, "MY_NOTE");
pfmt(stdout, MM_USER|MM_GET,"my_appl_cat:1:The file is writable");
```

本例中将把以下消息写入标准输出：

MY_NOTE: The file is writable

另请参阅

pfmt(3C)、 thread_safety(5)。

符合的标准

addsev()： SVID3

名称

add_wch、mvadd_wch、mvwadd_wch、wadd_wch — 向窗口中添加复合字符和呈现方式

概要

#include <curses.h>

int add_wch(const cchar_t *_wch_);

int wadd_wch(WINDOW *_win_, const cchar_t *_wch_);

int mvadd_wch(int _y_, int _x_, const cchar_t *_wch_);

int mvwadd_wch(WINDOW *_win_, int _y_, int _x_, const cchar_t *_wch_);

说明

这些函数将在当前位置或指定的位置向当前窗口或指定的窗口中添加信息，然后向前移动光标。这些函数执行环绕。这些函数执行特殊字符处理操作。

- 如果 *wch* 是指占位字符，则位于此位置前的一个字符将被删除，并随即在此位置之前添加由 *wch* 指定的新字符，其呈现方式也由 *wch* 指定；随后，光标将前移至屏幕上的下一个占位字符。
- 如果 *wch* 是指非占位字符，则位于此位置前的所有字符都将予以保留，而 *wch* 的非占位字符将被添加到占位复合字符的前面，且忽略 *wch* 所指定的呈现方式。

返回值

成功完成后，这些函数将返回 **OK** 。否则，它们将返回 **ERR** 。

错误

未定义错误。

另请参阅

addch(3X)、 curses_intro(3X)、 Rendition of Characters Placed into a Window 一节、<curses.h>。

历史变更记录

第一版发行在 X/Open Curses 第 4 期中。

名称

add_wchnstr、add_wchstr、mvadd_wchnstr、mvadd_wchstr、 mvwadd_wchnstr、mvwadd_wchstr、wadd_wchnstr、wadd_wchstr — 向窗口中添加由复合字符及呈现方式组成的数组

概要

#include <curses.h>

int add_wchnstr(const cchar_t ****wchstr*, int** *n***);**

int add_wchstr(const cchar_t ****wchstr*);**

int wadd_wchnstr(WINDOW ****win*, const cchar_t ****wchstr***, int** *n***);**

int wadd_wchstr(WINDOW ****win*, const cchar_t ****wchstr***);**

int mvadd_wchnstr(int *y***, int** *x***, const cchar_t ****wchstr***, int** *n***);**

int mvadd_wchstr(int *y***, int** *x***, const cchar_t ****wchstr***);**

int mvwadd_wchnstr(WINDOW ****win*, int** *y***, int** *x***, const cchar_t ****wchstr***,**
int *n***);**

int mvwadd_wchstr(WINDOW ****win*, int** *y***, int** *x***, const cchar_t ****wchstr***);**

说明

这些函数会从当前或指定的光标位置开始，将由 *wchstr* 指定的 **cchar_t** 数组写入当前窗口或指定的窗口中。

这些函数不会使光标位置向前移动。如果 *wchstr* 中包含任何特殊字符，则结果将不可预测。

遇到空的 **cchar_t** 时，函数将成功结束。填充好当前行时，函数也将成功结束。如果当前行的行尾无法与某个字符完全匹配，这些列中就会填充背景字符及呈现方式。

写入 *n* 个 **cchar_t** （如果 *n* 为 –1，则为整个 **cchar_t** 数组）后， **add_wchnstr()** 、 **mvadd_wchnstr()** 、 **mvwadd_wchnstr()** 及 **wadd_wchnstr()** 函数将成功结束。

返回值

成功完成后，这些函数将返回 **OK** 。否则，它们将返回 **ERR** 。

错误

未定义错误。

另请参阅

<curses.h>。

历史变更记录

第一版发行在 X/Open Curses 第 4 期中。

名称

annuity( )、annuityf( )、annuityl( )、annuityw( )、annuityq( ) - 养老金的现值因子

概要

#include <math.h>

double annuity(double rate, double periods);

float annuityf(float rate, float periods);

long double annuityl(long double rate, long double periods);

extended annuityw(extended rate, extended periods);

quad annuityq(quad rate, quad periods);

说明

函数 **annuity()** 计算养老金的现值因子:

(1 - (1 + *rate*) ** (-*periods*)) / *rate*

annuityf() 是 **float** 版的 **annuity()** ; 它采用 **float** 参数，并返回 **float** 结果。

annuityl() 是 **long double** 版的 **annuity()** ; 它采用 **long double** 参数，并返回 **long double** 结果。

annuityw() 是 **extended** 版的 **annuity()** ; 它采用 **extended** 参数，并返回 **extended** 结果。

在 HP-UX 系统中， **annuityq()** 等效于 **annuityl()** 。

用法

要使用这些函数，请使用缺省的 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。要使用 **annuityw()** 或 **annuityq()** ，请使用 **-fpwidetypes** 选项进行编译。确保程序包含 **<math.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

返回值

如果 *periods* 等于零，则 **annuity()** 返回 0.0。

如果 *rate* 等于零，则 **annuity()** 返回 *periods* 。

如果任一参数为 NaN，则 **annuity()** 返回 NaN。

如果 *rate* < -1，则 **annuity()** 返回 NaN 并引起无效异常。

如果 *rate* = -1 并且 *periods* = +INFINITY，则 **annuity()** 返回 +INFINITY。

如果 *rate* 等于 -1 并且 *periods* > 0，则 **annuity()** 等效于 **1.0 / 0.0** 。

如果 *rate* 等于 -1，则 **annuity()** 返回 -1。

如果 *rate* 等于 +INFINITY 并且 *periods* >= 0，则 **annuity()** 返回 +0.0。

如果 *rate* 等于 +INFINITY 并且 -1 < *periods* < 0，则 **annuity()** 返回 -0.0。

如果 *rate* 等于 +INFINITY 并且 *periods* = -1，则 **annuity()** 返回 -1。

如果 *rate* 等于 +INFINITY，则 **annuity()** 返回 -INFINITY。

如果 *periods* 为无穷大且 (*rate* * *periods*) >0 ，则 **annuity()** 等效于 **1.0 /** *rate* 。

如果 *periods* 为无穷大，则 **annuity()** 返回 *periods* 。

annuity() 返回一个适当的带符号无穷大值以代替非常大的值，并引起溢出和不精确异常。

当结果很小（实际上不正常或为 0）而使精度降低时， **annuity()** 会引起下溢和不精确的异常，并可能引起结果只是很小时的那些异常。

错误

没有定义任何错误。

另请参阅

compound(3M)、 exp(3M)、 expm1(3M)、 pow(3M)、 math(5)。

符合的标准

任何标准均未指定这些函数。

名称

asin( )、asinf( )、asinl( )、asinw( )、asinq( ) - 反正弦函数

概要

#include <math.h>

double asin(double x);

float asinf(float x);

仅适用于 **HP Integrity** 服务器

long double asinl(long double x);

extended asinw(extended x);

quad asinq(quad x);

说明

asin() 返回 *x* 的反正弦值，范围从 $-\pi/2$ 到 $\pi/2$. 。

asinf() 是 **float** 形式的 **asin()** ；它采用 **float** 参数，并返回 **float** 结果。

仅适用于 **Integrity** 服务器

asinl() 是 **long double** 形式的 **asin()** ；它采用 **long double** 参数，并返回 **long double** 结果。

asinw() 是 **extended** 形式的 **asin()** ；它采用 **extended** 参数，并返回 **extended** 结果。

asinq() 等效于 HP-UX 系统上的 **asinl()** 。

用法

要使用这些函数，请使用缺省的 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。

（对于 Integrity 服务器）要使用 **asinw()** 或 **asinq()** ，也可以使用 **-fpwidetypes** 选项进行编译。

要使用上面的任何函数，请确保程序包含 **<math.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

有关详细信息，请参阅《HP-UX floating-point guide for HP Integrity servers》（位于以下站点：**http://www.hp.com/go/fp** ）。

仅适用于 **PA-RISC** 系统

可以使用 Millicode 版本的 **asin()** 和 **asinf()** 函数。 Millicode 版本的数学库函数通常比标准库中对应的函数运行速度快。要使用这些形式，请使用 **+Olibcalls** 或 **+Oaggressive** 优化选项编译程序。

特殊情况下，Millicode 版本返回在“返回值”一节中描述的值，但不设置 **errno** 。

返回值

asin(±0**)** 将返回 ±0。

如果 *x* 的量值大于 1，则 **asin()** 将返回 NaN 并引发无效异常。

如果 *x* 为 NaN，则 **asin()** 将返回 NaN。

如果未引发其他异常，则说明未指定 **asin()** 是否引发不精确异常的情况。

错误

如果 *x* 的量值大于 1，则 **asin()** 会将 **errno** 设置为 [EDOM]。

Integrity 服务器

缺省情况下，Integrity 服务器上的 HP-UX **libm** 函数不设置 **errno** 。对于 **errno** 设置，可以使用 **+Olibmerrno** 选项进行编译。

另请参阅

acos(3M)、 asind(3M)、 atan(3M)、 atan2(3M)、 casin(3M)、 cos(3M)、 sin(3M)、 tan(3M)、 math(5)。

符合的标准

asin() ： SVID3、XPG4.2、ANSI C、 ISO/IEC C99（包括附件 F“IEC 60559 floating-point arithmetic”）

asinf() 、 **asinl()** ： ISO/IEC C99（包括附件 F“IEC 60559 floating-point arithmetic”）

名称

asind( )、asindf( )、asindl( )、asindw( )、asindq( ) - 以度为单位的反正弦函数

概要

#include <math.h>

double asind(double x);

float asindf(float x);

仅适用于 **HP Integrity** 服务器

long double asindl(long double x);

extended asindw(extended x);

quad asindq(quad x);

说明

asind() 返回 *x* 的反正弦度数值，其范围是 −90 到 90。

asindf() 是 **asind()** 的 **float** 版本；它采用 **float** 参数，并返回 **float** 结果。

仅适用于 **Integrity** 服务器

asindl() 是 **asind()** 的 **long double** 版本；它采用 **long double** 参数，并返回 **long double** 结果。

asindw() 是 **asind()** 的 **extended** 版本；它采用 **extended** 参数，并返回 **extended** 结果。

在 HP-UX 系统中， **asindq()** 等效于 **asindl()** 。

用法

要使用这些函数，请使用缺省的 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。

（对于 Integrity 服务器）要使用 **asindw()** 或 **asindq()** ，也可以使用 **-fpwidetypes** 选项进行编译。

要使用上面的任何函数，请确保程序包含 **<math.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

有关详细信息，请参阅《HP-UX floating-point guide for HP Integrity servers》（位于以下站点：**http://www.hp.com/go/fp** ）。

返回值

asind(±0**)** 返回 ±0。

如果 *x* 的值大于 1，则 **asind()** 将返回 NaN 并引起无效异常。

如果 *x* 为 NaN，则 **asind()** 将返回 NaN。

错误

如果 *x* 的值大于 1， **asind()** 会将 **errno** 设置为 [EDOM]。

Integrity 服务器

缺省情况下，Integrity 服务器上的 HP-UX **libm** 函数不设置 **errno** 。对于 **errno** 设置，可以使用 **+Olibmerrno** 选项进行编译。

另请参阅

acosd(3M)、 asin(3M)、 atand(3M)、 atan2d(3M)、 cosd(3M)、 sind(3M)、 tand(3M)、 atan2d(3M)、 math(5)。

符合的标准

任何标准均未指定这些函数。

名称

asinh( )、asinhf( )、asinhl( )、asinhw( )、asinhq( ) -反双曲正弦函数

概要

#include <math.h>

double asinh(double x);

仅适用于 **HP Integrity** 服务器

float asinhf(float x);

long double asinhl(long double x);

extended asinhw(extended x);

quad asinhq(quad x);

说明

asinh() 返回 *x* 的反双曲正弦值。

仅适用于 **Integrity** 服务器

asinhf() 是 **asinh()** 的 **float** 版本；它采用 **float** 参数，并返回 **float** 结果。

asinhl() 是 **asinh()** 的 **long double** 版本；它采用 **long double** 参数，并返回 **long double** 结果。

asinhw() 是 **asinh()** 的 **extended** 版本；它采用 **extended** 参数，并返回 **extended** 结果。

在 HP-UX 系统中， **asinhq()** 等效于 **asinhl()** 。

用法

（对于 Integrity 服务器）要使用 **asinhf()** 、 **asinhl()** 、 **asinhw()** 或 **asinhq()** ，请使用缺省的 **-Ae** 选项或者 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。

（对于 Integrity 服务器）要使用 **asinhw()** 或 **asinhq()** ，也可以使用 **-fpwidetypes** 选项进行编译。

要使用上面的任何函数，请确保程序包含 **<math.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

有关详细信息，请参阅《HP-UX floating-point guide for HP Integrity servers》（位于以下站点：**http://www.hp.com/go/fp** ）。

返回值

asinh(±0) 返回 ±0。

如果 *x* 为 ±INFINITY，则 **asinh()** 将分别返回 ±INFINITY。

如果 *x* 为 NaN，则 **asinh()** 将返回 NaN。

未指定 **asinh()** 是否会引起不精确异常。

错误

没有定义任何错误。

另请参阅

acosh(3M)、 atanh(3M)、 casinh(3M)、 sinh(3M)、 math(5)。

符合的标准

asinh() 、 **asinhf()** 和 **asinhl()** ： ISO/IEC C99（包括附件 F“IEC 60559 floating-point arithmetic”）

名称

assert( ) - 验证程序断言

概要

#include <assert.h>

void assert(int expression);

说明

该宏可用于将诊断信息加入程序。执行它时，如果 *expression* 为 false（零），则 **assert()** 在标准错误输出中输出：

Assertion failed: *expression*, **file** *xyz*, **line** *nnn*

且异常中止。在错误消息中， *xyz* 为源文件的名称， *nnn* 为 **assert()** 语句的源行号。

通过使用预处理器选项 **-DNDEBUG** 进行编译（请参阅 *cpp*(1)），或使用 **#include <assert.h>** 语句前面的预处理器控制语句 **#define NDEBUG** 进行编译，可阻止将断言编译到程序中。

警告

assert() 在兼容模式下使用的表达式参数不能包含无转义符的字符串或双引号。

另请参阅

cpp(1)、abort(3C)、 thread_safety(5)。

符合的标准

assert()：AES、SVID2、SVID3、XPG2、XPG3、XPG4、FIPS 151-2、POSIX.1 和 ANSI C

名称

atan2( )、atan2f( )、atan2l( )、atan2w( )、atan2q( ) - 反正切象限函数

概要

#include <math.h>

double atan2(double y, double x);

float atan2f(float y, float x);

仅适用于 **HP Integrity** 服务器

long double atan2l(long double y, long double x);

extended atan2w(extended y, extended x);

quad atan2q(quad y, quad x);

说明

atan2() 返回 y/x 的反正切值，范围从 $-\pi$ 到 π。并通过两个参数的符号确定返回值的象限。

atan2f() 是 **atan2()** 的 **float** 版本；它采用 **float** 参数，并返回 **float** 结果。

仅适用于 **Integrity** 服务器

atan2l() 是 **atan2()** 的 **long double** 版本；它采用 **long double** 参数，并返回 **long double** 结果。

atan2w() 是 **atan2()** 的 **extended** 版本；它采用 **extended** 参数，并返回 **extended** 结果。

在 HP-UX 系统中， **atan2q()** 等效于 **atan2l()** 。

用法

要使用这些函数，请使用缺省的 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。

（对于 Integrity 服务器）要使用 **atan2w()** 或 **atan2q()** ，也可以使用 **-fpwidetypes** 选项进行编译。

要使用上面的任何函数，请确保程序包含 **<math.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

有关详细信息，请参阅《HP-UX floating-point guide for HP Integrity servers》（位于以下站点：**http://www.hp.com/go/fp** ）。

仅适用于 **PA-RISC** 系统

可以使用 Millicode 版的 **atan2()** 函数。 Millicode 版的数学库函数通常比标准库中对应的函数运行速度快。要使用这些版本，请使用 **+Olibcalls** 或 **+Oaggressive** 优化选项编译程序。

特殊情况下，Millicode 版本的返回值与标准库中对应函数返回的值相同（请参阅“返回值”一节）。

返回值

如果 y 为 ±0 并且 x 为 –0，则 **atan2()** 分别返回 $\pm\pi$。

如果 *y* 为 ±0 并且 *x* 为 +0，则 **atan2()** 分别返回 ±0。

如果 *y* 为 ±0 ，并且 *x* 小于零，则 **atan2()** 分别返回 returns ±π。

如果 *y* 为 ±0 并且 *x* 大于零，则 **atan2()** 分别返回 ±0。

如果 *y* 小于零，并且 *x* 为零，则 **atan2()** 将返回 −π/2。

如果 *y* 大于零，并且 *x* 为零，则 **atan2()** 将返回 π/2。

如果 *y* 大于零，并且 *x* 为 –INFINITY，则 **atan2()** 返回 Pi。

如果 *y* 小于零，并且 *x* 为 –INFINITY，则 **atan2()** 将返回 −π。

如果 *y* 大于零，并且 *x* 为 INFINITY，则 **atan2()** 返回 +0。

如果 *y* 小于零，并且 *x* 为 INFINITY，则 **atan2()** 返回 –0。

如果 *y* 为 ±INFINITY 并且 *x* 为有限值，则 **atan2()** 将分别返回 ±π/2。

如果 *y* 为 ±INFINITY 并且 *x* 为 –INFINITY，则 **atan2()** 将分别返回 ±3π/4。

如果 *y* 为 ±INFINITY 并且 *x* 为 +INFINITY，则 **atan2()** 将分别返回 ±π/4。

如果 *x* 或 *y* 为 NaN，则 **atan2()** 返回 NaN。

未指定 **atan2()** 是否会引起不精确的异常。

错误

没有定义任何错误。

另请参阅

acos(3M)、 asin(3M)、 atan(3M)、 atan2d(3M)、 carg(3M)、 cos(3M)、 sin(3M)、 tan(3M)、 math(5)。

符合的标准

atan2()： SVID3、XPG4.2、ANSI C 和 ISO/IEC C99（包括附件 F“IEC 60559 floating-point arithmetic”）

atan2f() 和 **atan2l()**： ISO/IEC C99（包括附件 F“IEC 60559 floating-point arithmetic”）

名称

atan2d( )、atan2df( )、atan2dl( )、atan2dw( )、atan2dq( ) - 以度为单位的反正切象限函数

概要

#include <math.h>

double atan2d(double y, double x);

float atan2df(float y, float x);

仅适用于 **HP Integrity** 服务器

long double atan2dl(long double y, long double x);

extended atan2dw(extended y, extended x);

quad atan2dq(quad y, quad x);

说明

atan2d() 是 **atan2()** 函数的求度数值的版本。它返回 *y*/*x* 的反正切值，其范围是 –180 到 180 度，并通过两个参数的符号确定返回值的象限。

atan2df() 是 **atan2d()** 的 **float** 版本；它采用 **float** 参数，并返回 **float** 结果。

仅适用于 **Integrity** 服务器

atan2dl() 是 **atan2d()** 的 **long double** 版本；它采用 **long double** 参数，并返回 **long double** 结果。

atan2dw() 是 **atan2d()** 的 **extended** 版本；它采用 **extended** 参数，并返回 **extended** 结果。

在 HP-UX 系统中， **atan2dq()** 等效于 **atan2dl()** 。

用法

要使用这些函数，请使用缺省的 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。

（对于 Integrity 服务器）要使用 **atan2dw()** 或 **atan2dq()** ，也可以使用 **-fpwidetypes** 选项进行编译。

要使用上面的任何函数，请确保程序包含 **<math.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

有关详细信息，请参阅《HP-UX floating-point guide for HP Integrity servers》（位于以下站点：**http://www.hp.com/go/fp** ）。

返回值

如果 *y* 为 ±0 ，并且 *x* 为 –0，则 **atan2d()** 分别返回 ±180。

如果 *y* 为 ±0 ，并且 *x* 为 +0，则 **atan2d()** 分别返回 ±0。

如果 *y* 为 ±0 ，并且 *x* 小于 0，则 **atan2d()** 分别返回 ±180。

如果 *y* 为 ±0 ，并且 *x* 大于 0，则 **atan2d()** 分别返回 ±0。

如果 *y* 小于零，并且 *x* 为零，则 **atan2d()** 返回 –90。

如果 *y* 大于零，并且 *x* 为零，则 **atan2d()** 返回 90。

如果 *y* 大于零，并且 *x* 为 –INFINITY，则 **atan2d()** 返回 180。

如果 *y* 小于零，并且 *x* 为 –INFINITY，则 **atan2d()** 返回 –180。

如果 *y* 大于零，并且 *x* 为 INFINITY，则 **atan2d()** 返回 +0。

如果 *y* 小于零，并且 *x* 为 INFINITY，则 **atan2d()** 返回 –0。

如果 *y* 为 ±INFINITY ，并且 *x* 为无穷大，则 **atan2d()** 分别返回 ±90。

如果 *y* 为 ±INFINITY ，并且 *x* 为 –INFINITY，则 **atan2d()** 分别返回 ±135。

如果 *y* 为 ±INFINITY ，并且 *x* 为 +INFINITY，则 **atan2d()** 分别返回 ±45。

如果 *x* 或 *y* 为 NaN，则 **atan2d()** 返回 NaN。

错误

没有定义任何错误。

另请参阅

acosd(3M)、 asind(3M)、 atand(3M)、 atan2(3M)、 cosd(3M)、 sind(3M)、 tand(3M)、 math(5)。

符合的标准

任何标准均未指定这些函数。

名称

atan( )、atanf( )、atanl( )、atanw( )、atanq( ) - 反正切函数

概要

#include <math.h>

double atan(double x);

float atanf(float x);

仅适用于 **HP Integrity** 服务器

long double atanl(long double x);

extended atanw(extended x);

quad atanq(quad x);

说明

atan() 返回 *x* 的反正切值，范围是从 $-\pi/2$ 到 $\pi/2$. 。

atanf() 是 **float** 版的 **atan()** ；它采用 **float** 参数，并返回 **float** 结果。

仅适用于 **Integrity** 服务器

atanl() 是 **long double** 版的 **atan()** ；它采用 **long double** 参数，并返回 **long double** 结果。

atanw() 是 **extended** 版的 **atan()** ；它采用 **extended** 参数，并返回 **extended** 结果。

atanq() 等效于 HP-UX 系统上的 **atanl()** 。

用法

要使用这些函数，请使用缺省的 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。

（对于 Integrity 服务器）要使用 **atanw()** 或 **atanq()** ，也可以使用 **-fpwidetypes** 进行编译。

要使用上面的任何函数，请确保程序包含 **<math.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。有关详细信息，请参阅《HP-UX floating-point guide for HP Integrity servers》（位于以下站点: **http://www.hp.com/go/fp** ）。

仅适用于 **PA-RISC** 系统

可以使用 Millicode 版的 **atan()** 和 **atanf()** 函数。 Millicode 版的数学库函数通常比标准库中对应的函数运行速度快。要使用这些版本，请编译程序的 **+Olibcalls** 或 **+Oaggressive** 优化选项。

特殊情况下，Millicode 版本的返回值与标准库中对应函数返回的值相同（请参阅“返回值”一节）。

返回值

atan(±0) 将返回 ±0。

如果 *x* 为 ±INFINITY， **atan()** 将分别返回 $\pm\pi/2$。

如果 *x* 为 NaN， **atan()** 将返回 NaN。

未指定 **atan()** 是否引发不精确异常的情况。

错误

没有定义任何错误。

另请参阅

acos(3M)、 asin(3M)、 atand(3M)、 atan2(3M)、 catan(3M)、 cos(3M)、 sin(3M)、 tan(3M)、 math(5)。

符合的标准

atan()： SVID3、XPG4.2、ANSI C、 ISO/IEC C99（包括附件 F“IEC 60559 floating-point arithmetic”）

atanf() 、 **atanl()**： ISO/IEC C99（包括附件 F“IEC 60559 floating-point arithmetic”）

名称

atand( )、atandf( )、atandl( )、atandw( )、atandq( ) - 以度为单位的反正切函数

概要

#include <math.h>

double atand(double x);

float atandf(float x);

仅适用于 **HP Integrity** 服务器

long double atandl(long double x);

extended atandw(extended x);

quad atandq(quad x);

说明

atand() 返回 *x* 的反正切度数值，范围从 –90 到 90。

atandf() 是 **atand()** 的 **float** 版本；它采用 **float** 参数，并返回 **float** 结果。

仅适用于 **Integrity** 服务器

atandl() 是 **atand()** 的 **long double** 版本；它采用 **long double** 参数，并返回 **long double** 结果。

atandw() 是 **atand()** 的 **extended** 版本；它采用 **extended** 参数，并返回 **extended** 结果。

在 HP-UX 系统中， **atandq()** 等效于 **atandl()** 。

用法

要使用这些函数，请使用缺省的 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。

（对于 Integrity 服务器）要使用 **atandw()** 或 **atandq()** ，也可以使用 **-fpwidetypes** 选项进行编译。

要使用上面的任何函数，请确保程序包含 **<math.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

有关详细信息，请参阅《HP-UX floating-point guide for HP Integrity servers》（位于以下站点：**http://www.hp.com/go/fp** ）。

返回值

atand(±0**)** 返回 ±0。

如果 *x* 为 ±INFINITY，则 **atand()** 分别返回 ±90。

如果 *x* 为 NaN，则 **atand()** 返回 NaN。

错误

没有定义任何错误。

另请参阅

acosd(3M)、 asind(3M)、 atan(3M)、 atan2d(3M)、 cosd(3M)、 sind(3M)、 tand(3M)、 atan2d(3M)、 math(5)。

符合的标准

任何标准均未指定这些函数。

名称

atanh( )、atanhf( )、atanhl( )、atanhw( )、atanhq( ) - 反双曲正切函数

概要

#include <math.h>

double atanh(double x);

仅适用于 **HP Integrity** 服务器

float atanhf(float x);

long double atanhl(long double x);

extended atanhw(extended x);

quad atanhq(quad x);

说明

atanh() 返回 *x* 的反双曲正切值。

仅适用于 **Integrity** 服务器

atanhf() 是 **float** 形式的 **atanh()**；它采用 **float** 参数，并返回 **float** 结果。

atanhl() 是 **long double** 形式的 **atanh()**；它采用 **long double** 参数，并返回 **long double** 结果。

atanhw() 是 **extended** 形式的 **atanh()**；它采用 **extended** 参数，并返回 **extended** 结果。

atanhq() 等效于 HP-UX 系统上的 **atanhl()**。

用法

（对于 Integrity 服务器）要使用 **atanhf()**、**atanhl()**、**atanhw()** 或 **atanhq()**，请使用缺省的 **-Ae** 选项或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。

（对于 Integrity 服务器）要使用 **atanhw()** 或 **atanhq()**，也可以使用 **-fpwidetypes** 选项进行编译。

要使用上面的任何函数，请确保程序包含 **<math.h>**，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

有关详细信息，请参阅《HP-UX floating-point guide for HP Integrity servers》（位于以下站点：**http://www.hp.com/go/fp**）。

返回值

atanh(±0) 将返回 ±0。

atanh(±1) 返回 ±Inf 并引发除数为零的浮点异常。

如果 $|x| > 1.0$，则 **atanh()** 将返回 NaN 并引发无效异常。

如果 *x* 为 NaN，则 **atanh()** 将返回 NaN。

如果未引发其他异常，则说明未指定 **atanh()** 是否引发不精确异常的情况。

错误

如果 $|x| > 1.0$，则 **atanh()** 会将 **errno** 设置为 [EDOM]。

仅适用于 **Integrity** 服务器

缺省情况下，Integrity 服务器上的 HP-UX **libm** 函数不设置 **errno** 。对于 **errno** 设置，可以使用 **+Olibmerrno** 选项进行编译。

另请参阅

acosh(3M)、 asinh(3M)、 catanh(3M)、 tanh(3M)、 math(5)。

符合的标准

atanh() 、 **atanhf()** 、 **atanhl()** ： ISO/IEC C99（包括附件 F“IEC 60559 floating-point arithmetic”）

名称

atexit - 注册在程序终止时调用的函数

概要

#include <stdlib.h>

int atexit(void (*func)(void));

说明

atexit() 注册在程序正常终止时调用的函数 *func* ，后者不带任何参数。 **atexit()** 注册的函数按照注册的相反顺序调用。

退出处理过程中的 **atexit()** 调用始终失败。

所注册函数的数目不应超过 **<limits.h>** 中指定的 **ATEXIT_MAX** 。

在必要时， **crt0()** 或 **dld** （请参阅 *crt0*(3) 和用于 PA-RISC 系统的 *dld.sl*(5) 或用于基于 Itanium 的系统的 *dld.so*(5)）使用 **atexit()** 注册一个或多个函数，以允许在程序正常终止时进行一些处理。应该首先进行这种注册。

返回值

如果注册成功， **atexit()** 返回零；如果失败，则返回非零值。

另请参阅

适用于 PA-RISC 系统的 exit(2)、crt0(3) 或 dld.sl(5)。
适用于基于 Itanium 的系统的 dld.so(5)。

符合的标准

atexit()：AES、SVID3、XPG4 和 ANSI C

名称

attr_get、attr_off、attr_on、attr_set、color_set、 wattr_get、wattr_off、wattr_on、wattr_set、wcolor_set — 窗口属性控制函数

概要

#include <curses.h>

int attr_get(attr_t ****attrs*, short ****color***, void ****opts***);**

int attr_off(attr_t *attrs***, void ****opts***);**

int attr_on(attr_t *attrs***, void ****opts***);**

int attr_set(attr_t *attrs***, short** *color***, void ****opts***);**

int color_set(short *color***, void ****opts***);**

int wattr_get(WINDOW ****win*, attr_t ****attrs***, short ****color***, void ****opts***);**

int wattr_off(WINDOW ****win*, attr_t** *attrs***, void ****opts***);**

int wattr_on(WINDOW ****win*, attr_t** *attrs***, void ****opts***);**

int wattr_set(WINDOW ****win*, attr_t** *attrs***, short** *color***, void ****opts***);**

int wcolor_set(WINDOW ****win*, short** *color***, void ****opts***);**

说明

这些函数控制着当前窗口或所指定窗口的窗口呈现方式的属性和颜色。

attr_get() 和 **wattr_get()** 函数将获取窗口的当前呈现方式。

attr_off() 和 **wattr_off()** 函数将关闭当前窗口或所指定窗口的 *attrs*，而不会影响其他任何参数。

attr_on() 和 **wattr_on()** 函数将打开当前窗口或所指定窗口的 *attrs*，而不会影响其他任何参数。

attr_set() 和 **wattr_set()** 函数将把当前窗口或所指定窗口的窗口呈现方式设置为 *attrs* 和 *color*。

color_set() 和 **wcolor_set()** 函数将把当前窗口或所指定窗口的窗口颜色设置为彩色。

opts 参数将保留在此文档未来的版本中进行定义。目前，应用程序必须为 *opts* 提供一个空指针。

返回值

这些函数始终返回 OK。

错误

未定义错误。

另请参阅

attroff(3X)、 <curses.h>。

历史变更记录

第一版发行在 X/Open Curses 第 4 期中。

名称

attroff、attron、attrset、 wattroff、wattron、wattrset — 受限的窗口属性控制函数

概要

#include <curses.h>

int attroff(int *attrs***);**

int attron(int *attrs***);**

int attrset(int *attrs***);**

int wattroff(WINDOW ****win*, int** *attrs***);**

int wattron(WINDOW ****win*, int** *attrs***);**

int wattrset(WINDOW ****win*, int** *attrs***);**

说明

这些函数可控制当前窗口或指定窗口的窗口属性。

attroff() 和 **wattroff()** 函数将关闭当前窗口或指定窗口的 *attrs*，而不会影响其他任何参数。

attron() 和 **wattron()** 函数将打开当前窗口或指定窗口的 *attrs*，而不会影响其他任何参数。

attrset() 和 **wattrset()** 函数将把当前窗口或指定窗口的背景属性设置为 *attrs*。

但未指定这些函数能否用于控制除 **A_BLINK** 、 **A_BOLD** 、 **A_DIM** 、 **A_REVERSE** 、 **A_STANDOUT** 和 **A_UNDERLINE** 以外的其他属性。

返回值

这些函数始终返回 **OK** 或 **1** 。

错误

未定义任何错误。

另请参阅

attr_get(3X)、 standend(3X)、 <curses.h>。

历史变更记录

在 X/Open Curses 第 2 期中首次发布。

X/Open Curses 第 4 期

为了清楚说明，此条目已经过重新编写。更新后的说明一节指明：对于这些函数能否用于控制除 X/Open Curses 第 3 期中所定义的那些属性以外的其他属性，并未予以定义。 **standend()** 、 **standout()** 、 **wstandend()** 和 **wstandout()** 函数已移至条目 **standend()** 。

名称

basename( )、dirname( ) - 提取路径名的各个部分

概要

#include <libgen.h>

char *basename(char *path);

char *dirname(char *path);

说明

basename() 使用 *path* 指向的路径名，并返回指向文件名最后一个部分的指针，以删除任何结尾“/”字符。如果字符串完全由“/”字符组成，则 **basename()** 将返回指向字符串“/”的指针。如果 *path* 是一个空指针或者指向空字符串，则 **basename()** 将返回指向字符串“.”的指针。

dirname() 使用 *path* 指向的路径名，并返回指向字符串的指针，该字符串是文件父目录的路径名。如果 *path* 是一个空指针、指向空字符串或者不包含“/”字符，则 **dirname()** 返回指向字符串“.”的指针。

返回值

basename() 返回指向 *path* 的最后一个部分的指针。

dirname() 返回指向字符串的指针，该字符串是 *path* 的父目录。

举例

以下代码段调用 **basename()** 和 **dirname()** 。

```
#include <libgen.h>

const char *path="/usr/local/bin/foo";
char *base, *dir;
char *dup0, *dup1;
char *dup2, *dup3;

        /*Use strdup in case literals
        * are in text, or basename()
        * and dirname() modify the
        * input string.
        */

        dup0=strdup(path);
        dup1=strdup(path);

        base=basename(dup0);
        dir=dirname(dup1);

        /* Use strdup before modifying
        * return string in case a
```

```
* pointer to a literal is
* returned.
*/

dup2=strdup(base);
dup3=strdup(dir);
```

警告

basename() 和 **dirname()** 可能会覆盖 *path* 。

非线程应用程序中的 **basename()** 和 **dirname()** 当前使用每一个函数中的静态缓冲区保存结果字符串。对 **basename()** 和 **dirname()** 的任何后续调用都可能会覆盖静态缓冲区。在 HP-UX 11i v1 之后的某些版本中，HP 可能会使用 **malloc()** 分配缓冲区。分配后，这些缓冲区将被重新使用，并且它们的地址不会更改；但是，如果 **malloc()** 失败， **basename()** 和 **dirname()** 将返回“.”，而且 **errno** 将被设置为 [ENOMEM] 。

作者

basename() 和 **dirname()** 由 HP 开发。

另请参阅

basename(1)、 thread_safety(5)。

符合的标准

basename()： XPG4.2

dirname()： XPG4.2

名称

baudrate — 获取终端的波特率

概要

#include <curses.h>

int baudrate(void);

说明

baudrate() 函数提取终端的输出速度（以每秒位数为单位）。

返回值

baudrate() 函数返回终端的输出速度。

错误

没有定义任何错误。

另请参阅

tcgetattr() （《X/Open System Interfaces and Headers, Issue 4, Version 2》规范）、 <curses.h>。

历史变更记录

在 X/Open Curses 第 2 期中首次发布。

X/Open Curses 第 **4** 期

参数列表已显式声明为 **void**。

名称

beep — 可听信号

概要

```
#include <curses.h>

int beep(void);
```

说明

beep() 函数用于向用户提示报警。它可在终端上发出可听警报，如果不能发出警报，它可使屏幕闪烁（可视警报）。如果没有出现任何信号，则什么都没有发生。

返回值

beep() 函数始终返回 OK。

错误

没有定义任何错误。

实际应用信息

几乎所有终端都有可听警报，但只有某些终端能够使屏幕闪烁。

另请参阅

flash(3X)、 <curses.h>。

历史变更记录

在 X/Open Curses 第 2 期中首次发布。

X/Open Curses 第 4 期

参数列表已显式声明为 **void**。已将返回值部分更改为表明函数始终返回 OK。 **flash()** 函数已移到它自己的条目中。

名称

bgets() - 读取流，直到下一定界符

概要

#include <libgen.h>

char *bgets(
char * *buffer*,
size_t * *count*,
FILE * *stream*,
const char * *breakstring*
);

说明

在 *count* 耗尽或在流中遇到 *breakstring* 中的某个字符前， **bgets** 将 *stream* 中的字符读取到 *buffer* 。返回空字节 (**\0**) 和指向尾随空字符的指针时，数据读取操作即告终止。如果遇到 *breakstring* 字符，则上一个非空字符就会成为终止扫描用的定界符。

请注意，除返回值指向读取字符串的 **end** 而非起始位置之外，调用

bgets(buffer, sizeof buffer, stream, \n);

与以下调用等同：

fgets(buffer, sizeof buffer, stream);

缓冲区中会始终为尾随空字符保留足够的空间。

如果 *breakstring* 是一个空指针，则使用上一调用的 *breakstring* 值。如果 *breakstring* 在第一次调用时为空字符，则没有用于界定字符串的字符。

要使用此接口，请通过指定 **--lgen** 来链接到 libgen 库。例如：

cc foo.c -lgen

返回值

遇到错误或到达文件末尾时，将返回 NULL。如果读取字符后未返回值，则情况报告将被延迟到下一调用。

举例

```
#include <libgen.h>

char buffer[8];
/* read in first user name from /etc/passwd */
fp = fopen("/etc/passwd","r");
bgets(buffer, 8, fp, ":");
```

另请参阅

gets(3S)、 thread_safety(5)。

名称

bigcrypt() - 对大字符串实现散列加密

概要

#include <hpsecurity.h>

#include <prot.h>

char *bigcrypt(char *key, char *salt);

说明

bigcrypt() 与 *crypt*(3C) 类似，但它可处理更大的字符串。 **bigcrypt()** 使用明文段，并分别对这些段进行加密，首先使用传入的 salt，然后将以前加密的段的前两个字符作为下一个段的 salt。这在口令字符重复时可避免重复的密文组块，因此一个段的加密涉及到所有以前的段的加密。

每一个密文段连接在一起（以 salt 开头）构成整个加密字符串。

警告

HP-UX 11i v3 是支持受信任系统功能的最后一个版本。

作者

bigcrypt 由 HP 开发。

另请参阅

crypt(3C)。

名称

bkgd、bkgdset、getbkgd、wbkgd、wbkgdset - 使用单字节字符来设置或获取背景字符及呈现方式

概要

#include <curses.h>

int bkgd(chtype *ch***);**

void bkgdset(chtype *ch***);**

chtype getbkgd(WINDOW ****win*);**

int wbkgd(WINDOW ****win*, chtype** *ch***);**

void wbkgdset(WINDOW ****win*, chtype** *ch***);**

说明

bkgdset() 和 **wbkgdset()** 函数将基于 *ch* 中的信息设置当前窗口或所指定窗口的背景属性。如果 *ch* 是指多列字符，则结果将不可预测。

bkgd() 和 **wbkgd()** 函数将设置当前窗口或所指定窗口的背景属性，然后将此设置应用于此窗口中的每个字符位置:

- 屏幕上每个字符的呈现方式都更改为新的背景呈现方式。
- 以前的背景字符无论出现在何处，都会更改为新的背景字符。

getbkgd() 函数将提取所指定窗口的背景字符及呈现方式。

返回值

成功完成后， **bkgd()** 和 **wbkgd()** 将返回 OK。否则，它们将返回 ERR。

bkgdset() 和 **wbkgdset()** 函数不返回值。

成功完成后， **getbkgd()** 将返回所指定窗口的背景字符及呈现方式。否则，它将返回 (**chtype**)ERR。

错误

未定义错误。

实际应用信息

只能保证这些函数在特定的字符集中可靠地运行，即在这些字符集中，每个字符都与一个单字节字符匹配，其属性只能通过带有 A_ prefix 的常量来表示。

另请参阅

curses_intro(3X)、 Rendition 小节、<curses.h>。

历史变更记录

第一版发行在 X/Open Curses 第 4 期中。

名称

bkgrnd、bkgrndset、getbkgrnd、wbkgrnd、wbkgrndset、wgetbkgrnd — 使用复合字符来设置或获取背景字符及呈现方式

概要

#include <curses.h>

int bkgrnd(const cchar_t *_wch_);

void bkgrndset(const cchar_t *_wch_);

int getbkgrnd(cchar_t *_wch_);

int wbkgrnd(WINDOW *_win_, const cchar_t *_wch_);

void wbkgrndset(WINDOW *_win_, const cchar_t *_wch_);

int wgetbkgrnd(WINDOW *_win_, cchar_t *_wch_);

说明

bkgrndset() 和 **wbkgrndset()** 函数将基于 *wch* 中的信息设置当前窗口或所指定窗口的背景属性。

bkgrnd() 和 **wbkgrnd()** 函数将设置当前窗口或所指定窗口的背景属性，然后将此设置应用于此窗口中的每个字符位置:

- 屏幕上每个字符的呈现方式都更改为新的背景呈现方式。
- 以前的背景字符无论出现在何处，都会更改为新的背景字符。

如果 *wch* 是指 **bkgrnd()** 、 **bkgrndset()** 、 **wbkgrnd()** 和 **wbkgrndset()** 的非占位复合字符，就会将 *wch* 添加到作为背景字符的现有占位复合字符上。如果 *wch* 是指多列字符，则结果将不可预测。

getbkgrnd() 和 **wgetbkgrnd()** 函数将把窗口的背景字符及呈现方式的值存储到 *wch* 所指向的区域。

返回值

bkgrndset() 和 **wbkgrndset()** 函数不返回值。

成功完成后，其他函数将返回 OK。否则，它们将返回 ERR。

错误

未定义错误。

另请参阅

curses_intro(3X)、 Rendition 小节、<curses.h>。

历史变更记录

第一版发行在 X/Open Curses 第 4 期中。

名称

border、 wborder — 利用单字节字符及显示形式绘制边框

概要

#include <curses.h>

int border(chtype *ls*, **chtype** *rs*, **chtype** *ts*, **chtype** *bs*, **chtype** *tl*,
chtype *tr*, **chtype** *bl*, **chtype** *br*);

int wborder(WINDOW ****win*, **chtype *ls*, **chtype** *rs*, **chtype** *ts*, **chtype** *bs*,
chtype *tl*, **chtype** *tr*, **chtype** *bl*, **chtype** *br*);

说明

border() 和 **wborder()** 函数将围绕当前窗口或所指定窗口的边缘绘制边框。这些函数不会使光标位置向前移动。这些函数不执行特殊字符处理操作。这些函数不执行换行操作。

位于下表左侧列中的参数包含单字节字符及显示形式，在绘制边框时具有下列用法：

参数名称	用法	缺省值
ls	起始列一侧	ACS_VLINE
rs	结束列一侧	ACS_VLINE
ts	第一行一侧	ACS_HLINE
bs	最后一行一侧	ACS_HLINE
tl	第一行与起始列的交角	ACS_ULCORNER
tr	第一行与结束列的交角	ACS_URCORNER
bl	最后一行与起始列的交角	ACS_BLCORNER
br	最后一行与结束列的交角	ACS_BRCORNER

如果左侧列中任意参数的值为 0，则使用右侧列中的缺省值。如果左侧列中任意参数的值为多列字符，则结果将无法确定。

返回值

成功完成后，这些函数将返回 OK。否则，它们将返回 ERR。

错误

未定义任何错误。

实际应用信息

这些函数只能确保对特定的字符集进行可靠的操作，在这些字符集中，每个字符都是单字节字符，其属性只能使用前缀为 A_ 的常量来表示。

另请参阅

border_set(3X)、 box(3X)、 hline(3X)、 <curses.h>。

历史变更记录

在 X/Open Curses 第 4 期中首次发布。

名称

border_set 和 wborder_set — 通过组合字符和呈现方式绘制边框

概要

```
#include <curses.h>

int border_set(const cchar_t *ls, const cchar_t *rs, const cchar_t *ts,
        const cchar_t *bs, const cchar_t *tl, const cchar_t *tr,
        const cchar_t *bl, const cchar_t *br);

int wborder_set(WINDOW *win, const cchar_t *ls, const cchar_t *rs,
        const cchar_t *ts, const cchar_t *bs,
        const cchar_t *tl, const cchar_t *tr,
        const cchar_t *bl, const cchar_t *br);
```

说明

border_set() 和 **wborder_set()** 函数沿着当前窗口或指定窗口的边缘绘制边框。这些函数不会使光标位置向前移动。这些函数不执行特殊字符处理。这些函数不执行换行操作。

下表左侧列中的参数包含带呈现方式的占位组合字符，这些参数在绘制边框时用法如下：

参数名称	用法	缺省值
ls	起始列	WACS_VLINE
rs	结束列	WACS_VLINE
ts	第一行	WACS_HLINE
bs	最后一行	WACS_HLINE
tl	第一行和起始列之间的角	WACS_ULCORNER
tr	第一行和结束列之间的角	WACS_URCORNER
bl	最后一行和起始列之间的角	WACS_BLCORNER
br	最后一行和结束列之间的角	WACS_BRCORNER

如果左侧列中某一参数的值是空指针，则使用右侧列中的缺省值。如果左侧列中某一参数的值是多列字符，则结果是不确定的。

返回值

成功完成后，这些函数返回 OK。否则，将返回 ERR。

错误

没有定义任何错误。

另请参阅

box_set(3X)、 hline_set(3X)、 <curses.h>。

历史变更记录

在 X/Open Curses 第 4 期中首次发布。

名称

box — 通过单字节字符和呈现方式绘制边框

概要

#include <curses.h>

int box(WINDOW ****win*, chtype** *verch***, chtype** *horch***);**

说明

box() 函数沿着指定窗口的边缘绘制边框。该函数不向前移动光标位置。该函数不执行特殊字符处理。该函数也不执行换行。

函数 **box(***win***,** *verch***,** *horch***)** 与下面的函数具有相同的作用：

wborder(*win***,** *verch***,** *verch***,** *horch***,** *horch***, 0, 0, 0, 0);**

返回值

成功完成后， **box()** 返回 OK。否则，返回 ERR。

错误

没有定义任何错误。

实际应用信息

这些函数只保证对这样的字符集进行可靠操作：其中的每个字符都属于单字节，只能用带有 A_ 前缀的常量来表示其属性。

另请参阅

border(3X)、 box_set(3X)、 hline(3X)、 <curses.h>。

历史变更记录

在 X/Open Curses 第 2 期中首次发布。

X/Open Curses 第 **4** 期

说明已更改为根据对 **wborder()** 函数的调用情况来描述该函数。

名称

box_set — 通过组合字符和呈现方式绘制边框

概要

#include <curses.h>

int box_set(WINDOW *_win_, const cchar_t *_verch_, const cchar_t *_horch_);

说明

box_set() 函数沿着指定窗口的边缘绘制边框。该函数不向前移动光标位置。该函数不执行特殊字符处理。该函数也不执行换行。

函数 **box_set(_win_, _verch_, _horch_)** 与下面的函数具有相同的作用：

wborder_set(*win*, *verch*, *verch*, *horch*, *horch*, NULL, NULL, NULL, NULL);

返回值

成功完成后，该函数返回 OK。否则，返回 ERR。

错误

没有定义任何错误。

另请参阅

border_set(3X)、 hline_set(3X)、 <curses.h>。

历史变更记录

在 X/Open Curses 第 4 期中首次发布。

名称

bsdproc：killpg()、getpgrp()、setpgrp()、signal()、sigvec() - 与 4.2 BSD 兼容的进程控制工具

概要

```
#include <signal.h>

int killpg(int pgrp, int sig);

int getpgrp(int pid);

int setpgrp(int pid, int pgrp);

void (*signal(int sig, void (*func)(int)))(int);
```

过时的接口

```
int sigvec(
    int sig,
    struct sigvec *vec,
    struct sigvec *ovec
);
```

说明

这些调用模拟 4.2 Berkeley Software Distribution 中的同名函数（提供这些调用是为了与同名函数向后兼容）。

此版本的 **setpgrp()** 等效于系统调用 **setpgid(***pid***,** *pgrp***)** （请参阅 *setpgid*(2) ）。

此版本的 **getpgrp()** 等效于系统调用 **getpgrp2(***pid***)** （请参阅 *getpid*(2) ）。

killpg() 等效于系统调用 **kill(-***pgrp***,** *sig***)** （请参阅 *kill*(2) ）。

不再支持函数 **sigvec()** 。请参阅 *sigaction*(2) 的联机帮助页。

此版本的 **signal()** 与 **sigvec(***sig***,** *vec***,** *ovec***)** 具有相同的作用，其中 *vec−>sv_handler* 等于 *func* ， *vec−>sv_mask* 等于 0， *vec−>sv_flags* 等于 0。如果等效的 **sigvec()** 调用成功， **signal()** 返回的值将存储在 *ovec−>sv_handler* 。否则， **signal()** 将返回 −1，并且 **errno** 设置为指示原因，这与等效的 **sigvec()** 调用对其的设置方式相同。

对于使用 BSD 信号的应用程序，建议它们使用 *sigaction(2)* 。系统支持具有与 Berkeley Software Distribution 类似的行为的其他信号，API 为 **bsd_signal** ，但是不推荐使用。

警告

虽然 4.3 BSD 版本定义了此处描述的某些接口的扩展，但是该程序包只模拟 4.2 BSD 接口。

bsdproc() 不应该与 *sigset*(3C) 中描述的工具一起使用。

过时的接口

不再支持函数 **sigvec()** 。请参阅 *sigaction*(2) 的联机帮助页。

实际应用信息

线程注意事项

进程中的所有线程共享 **sigvec()** 和 **signal()** 建立的信号配置（如 **catch/ignore/default** ）。

有关信号和线程的详细信息，请参考 *signal*(5) 。

作者

bsdproc() 由 HP 和加州大学伯克利分校联合开发。

另请参阅

ld(1)、 kill(2)、 getpid(2)、 msgop(2)、 msgsnd(2)、 msgrcv(2)、 read(2)、 semop(2)、 setpgid(2)、 setsid(2)、 wait(2)、 write(2)、 sigaction(2)、 sigset(3C)、 sigstack(2)、 bsd_signal(3C)、 signal(5)。

名称

bsearch( ) - 二进制搜索排序表

概要

```
#include <stdlib.h>

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

说明

bsearch() 是从 Knuth (6.2.1) 算法 B 归纳而来的二进制搜索例行程序。此例行程序可返回指向表的指针，以指示在哪里可以找到某个数据。必须事先根据所提供的比较函数对该表进行升序排列。 *key* 指向要在表中查找的数据实例。 *base* 指向表基址的元素。 *nel* 是表中的元素数。 *size* 是表中每个元素的大小。 *compar* 是比较函数的名称，将使用指向要比较的元素的两个参数调用该函数。此函数必须返回一个小于、等于或大于零的整数，表示第一个参数（关键字）被视为小于、等于或大于第二个参数（数组元素）。

注释

指向表基址的关键字和元素的指针应该为 pointer-to-element 类型，并且应转换为 pointer-to-void 类型。

比较函数不必要比较每个字节，因此，元素中除了包含要比较的值外，还可能包含任意数据。

尽管返回值声明为 pointer-to-void 类型，但还是应将其转换为 pointer-to-element 类型。

返回值

如果无法在表中找到关键字，则返回 NULL 指针。

举例

以下示例将搜索一个表，该表包含指向由字符串及其长度组成的节点的指针。该表在每个条目指向的节点中的字符串上按字母顺序排序。

该代码段将读入字符串，然后要么查找对应的节点，并输出字符串及其长度，要么输出错误消息。

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

#define TABSIZE  1000

struct node {                   /* these are stored in the table */
   char *string;
```

```
    int length;
};
struct node table[TABSIZE];     /* table to be searched */
    .
    .
    .
{
    struct node *node_ptr, node;
    /* routine to compare 2 nodes */
    int node_compare(const void *, const void *);
    char str_space[20];         /* space to read string into */
    .
    .
    .
    node.string = str_space;
    while (scanf("%s", node.string) != EOF) {
        node_ptr = (struct node *)bsearch((void *)(&node),
              (void *)table, TABSIZE,
              sizeof(struct node), node_compare);
        if (node_ptr != NULL) {
            (void)printf("string = %20s, length = %d\n",
                node_ptr->string, node_ptr->length);
        } else {
            (void)printf("not found:%s\n", node.string);
        }
    }
}

/*      This routine compares two nodes based on an
        alphabetical ordering of the string field.
*/
int
node_compare(const void *node1, const void *node2)
struct node *node1, *node2;
{
    return strcoll(((const struct node *)node1)->string,
                ((const struct node *)node2)->string);
}
```

警告

如果搜索的表包含与选择条件匹配的两个或更多条目， **bsearch()** 将按照搜索算法的指定返回一个随机条目。

另请参阅

hsearch(3C)、lsearch(3C)、qsort(3C)、tsearch(3C)、 thread_safety(5)。

符合的标准

bsearch()：AES、SVID2、SVID3、XPG2、XPG3、XPG4、FIPS 151-2、POSIX.1、ANSI C

名称

btowc(), wctob() - 单字节字符与宽字符之间的转换

概要

#include <stdio.h>
#include <wchar.h>

wint_t btowc(int *c***);**

int wctob(wint_t *c***);**

说明

btowc() **btowc()** 函数将确定 *c* 在初始转换状态下是否构成有效的（单字节）字符。

wctob() **wctob()** 函数将确定 *c* 是否与扩展字符集的一个成员相对应（在初始切换状态下该成员的字符表示为单字节）。

实际应用信息

如果应用程序满足下列条件，则可以使用这些函数的原型：

a. 符合 **c99** 的要求。

b. 使用值 >=500 的 **-D_XOPEN_SOURCE** 宏进行编译。

c. 使用值 >= 200112 的 **-D_POSIX_C_SOURCE** 宏进行编译。

外部语言环境影响

环境变量

这些函数的行为受到 **LC_CTYPE** 类别的影响。

返回值

如果 *c* 的值为 **EOF** 或作为无符号型字符的 *c* 在初始转换状态下未构成有效（单字节）字符， **btowc()** 函数就会返回 **WEOF** 。否则，它将返回此字符的宽字符形式。

如果 *c* 在初始转换状态下不对应长度为一个字符的字符， **wctob()** 函数就会返回 **EOF** 。否则，它将返回字符的单字节形式。

错误

没有定义任何错误。

作者

btowc() 由 HP 和 Mitsubishi Electric Corp. 联合开发。

另请参阅

thread_safety(5)。

名称

bufsplit() - 将缓冲区分为若干字段

概要

#include <libgen.h>

size_t bufsplit(char *_buf_, size_t _n_, char **_a_);

说明

bufsplit 将对缓冲区 *buf* 进行检查，然后为指针数组 *a* 赋值，以便指针可以指向由制表符或换行符所分隔的 *buf* 中的前 *n* 个字段。

要更改用于分隔字段的字符，请调用 **bufsplit** ，同时使 *buf* 指向相应的字符串，并将 *n* 和 *a* 设置为 0。例如，要将“**:**”、“**.**”及“**,**”与制表符和换行符一起用作分隔符：

```
bufsplit(":.,\t\n", 0, (char**)0);
```

要使用此接口，请通过指定 **-lgen** 来链接到 libgen 库。例如：

```
cc foo.c -lgen
```

返回值

数组 *a* 中所分配的字段数。如果 *buf* 为 0，返回值也将为 0，同时数组保持不变。否则，值至少为 1。对于数组中元素的其余部分，则为其分配缓冲区末尾的空字节的地址。

举例

```
/*
 * set a[0] = "This", a[1] = "is", a[2] = "a",
 * a[3] = "test"
 */
bufsplit("This\tis\ta\ttest\n", 4, a);
```

警告

bufsplit 将把定界符更改为 *buf* 中的空字节。

另请参阅

thread_safety(5)。

名称

bwtmps: bwtmpname()、 updatebwdb()、 getbwent()、 setbwent()、 endbwent() -访问并更新 wtmps 和 btmps 数据库的例行程序

概要

#include <utmps.h>

void bwtmpname(char *file);

int updatebwdb(struct utmps *utmps, size_t size);

struct utmps * getbwent(size_t size);

void setbwent(void);

void endbwent(void);

说明

getbwent() 返回指向 **utmps** 结构的指针。 **utmps** 结构的关键成员包括:

char ut_user[]	用户登录名
char ut_id[]	用于区分某个条目的唯一 ID
char ut_line[]	设备名
pid_t ut_pid	进程 Id
short ut_type	条目类型
struct ut_exit	进程的退出状态
struct timeval ut_tv	生成了时间条目
char ut_host[]	主机名（如果是远程操作）
uint8_t ut_addr[]	主机的 Internet 地址（如果是远程操作）
short ut_addr_type	用于标识地址类型的标志in ut_addr

例行程序

提供了以下例行程序:

bwtmpname() 允许用户设置针对 **WTMPS_FILE** 或 **BTMPS_FILE** 而检查的数据库。 **bwtmpname()** 不会打开数据库 -- 它只是关闭当前打开的旧文件，然后保存为新文件名。 **WTMPS_FILE** 和 **BTMPS_FILE** 是 **<utmps.h>** 头文件中定义的宏，分别引用 **/var/adm/wtmps** 和 **/var/adm/btmps** 文件。

getbwent() 根据事先对 **bwtmpname()** 的调用，可能从 **WTMPS_FILE** ，也可能从 **BTMPS_FILE** 读入下一个条目。如果数据库尚未打开， **getbwent()** 可将其打开。如果到达文件的结尾， **getbwent()** 将会失败。

updatebwdb() 根据事先调用 **bwtmpname()** 后的设置，将 **utmps** 指向的 **utmps** 结构追加到 **BTMPS_FILE** 的后面或 **WTMPS_FILE** 的后面。

setbwent() 根据事先调用 **bwtmpname()** 后的指定，将 **getbwent()** 请求重置为从数据库的起始位置开始。

endbwent() 关闭当前打开的数据库的句柄。

返回值

如果成功， **getbwent()** 将返回指向 **struct utmps** 类型的结构的指针。如果 *size* 不是受支持的 **utmps** 结构大小之一， **getbwent()** 将返回 NULL，同时将 errno 设置为 [EINVAL] 。如果到达数据库的结尾， **getbwent** 将返回 NULL。

如果对数据库追加成功， **updatebwdb()** 将返回 0。如果追加失败， **updatebwdb()** 将返回 -1。

错误

[EINVAL] 传递给 *bwtmps*(3C) 函数的大小参数不匹配任何受支持的 **utmps** 结构大小。

警告

应用程序不应直接访问 **wtmps** 和 **btmps** 数据库，而应使用这些函数，原因是写入到这些数据库的结构是 **utmps** 结构的超集。 **updatebwdb()** 不更新 **/var/adm/wtmp** 文件和 **/var/adm/btmp** 文件。

bwtmps(3C) 接口可以加载共享库 **libuseracct.so.1/libuseracct.1** 。

归档应用程序的编译/链接命令（仅适用于 **PA-RISC**）

如果使用了 *bwtmps*(3C) 接口，那么在编译/链接应用程序归档文件时，请注意 *bwtmps*(3C) 接口和 **libdld.sl** 存在相关性，这需要更改编译/链接命令：

编译：

cc -Wl,-a,archive -Wl,-E -Wl,+n -l:libdld.sl -o *outfile source*

或使用 **CCOPTS** 和 **LDOPTS:** 编译

export CCOPTS="-Wl,-a,archive options -Wl,-E -l:libdld.sl"

export LDOPTS="options -E +n -l:libdld.sl"

cc -o *outfile source*

选项 **-Wl,-a,archive** 与位置相关，应在编译行的起始处应用。为获得将来发行版的最佳兼容性，应避免与其他共享库（为满足以上情况而使用的 **libdld.sl** 除外）一起使用归档文件 libc。

作者

bwtmps 例行程序由 HP 开发。

文件

/var/adm/wtmps
/var/adm/btmps

另请参阅

getuts(3C)、thread_safety(5)。

名称

htonl( )、htons( )、ntohl( )、ntohs( ) - 在主机和网络字节顺序之间进行值的转换

概要

#include <netinet/in.h>

_XOPEN_SOURCE_EXTENDED only
#include <arpa/inet.h>

unsigned long htonl(unsigned long hostlong);

unsigned short htons(unsigned short hostshort);

unsigned long ntohl(unsigned long netlong);

unsigned short ntohs(unsigned short netshort);

说明

这些例行程序将在网络字节顺序与主机字节顺序之间对 16 位和 32 位的量进行转换。在 HP-UX 系统上，网络字节顺序和主机字节顺序是相同的，因此这些例行程序在包含文件 **<netinet/in.h>** 中被定义为空的宏。如果定义了 **_XOPEN_SOURCE_EXTENDED** ，则在包含文件 **<arpa/inet.h>** 中也随即定义了这些例行程序。

这些例行程序通常与 **gethostent()** 和 **getservent()** 所返回的 Internet 地址及端口配合使用（请参阅 *gethostent*(3N) 和 *getservent*(3N) ）。使用这些例行程序可以编写可移植的程序。

作者

byteorder() 由加州大学伯克利分校开发。

另请参阅

gethostent(3N)、 getservent(3N)。

符合的标准

byteorder()： XPG4

名称

cabs( )、cabsf( )、cabsl( )、cabsw( )、cabsq( ) - 复数绝对值（也称为范数、模数或量值）函数

概要

#include <complex.h>

double cabs(double complex z);

float cabsf(float complex z);

long double cabsl(long double complex z);

extended cabsw(extended complex z);

quad cabsq(quad complex z);

说明

这些函数仅适用于 Integrity 服务器。

cabs() 返回 *z* 的复数绝对值。

cabsf() 是 **float complex** 版的 **cabs()** ；它采用 **float complex** 参数，并返回 **float** 结果。

cabsl() 是 **long double complex** 版的 **cabs()** ；它采用 **long double complex** 参数，并返回 **long double** 结果。

cabsw() 是 **extended complex** 版的 **cabs()** ；它采用 **extended complex** 参数，并返回 **extended** 结果。

cabsq() 等效于 HP-UX 系统上的 **cabsl()** 。

用法

要使用这些函数，请使用缺省的 **-Ae** 选项进行编译。要使用 **cabsw()** 或 **cabsq()** ，请使用 **-fpwidetypes** 选项进行编译。确保程序包含 **<complex.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

返回值

cabs(z) 返回相同的值并引发与 **hypot(creal(z),cimag(z))** 相同的浮点异常。

如果 **+Ocxlimitedrange** 编译程序选项有效，则可以使用不匹配常规规范的更快的方法来计算 **cabs()** ，以获取有限值以及超大扩展值和象限值。 HP C/C++ 缺省为 **+Onocxlimitedrange** 。 **+Ofltacc=relaxed** 选项隐含 **+Ocxlimitedrange** 。

错误

没有定义任何错误。

另请参阅

fabs(3M)、 carg(3M)、 complex(5)、 hypot(3M)。

符合的标准

cabs() 、 **cabsf()** 、 **cabsl()** ： ISO/IEC C99（包括附件 G“IEC 60559-compatible complex arithmetic”）

名称

cacos( )、cacosf( )、cacosl( )、cacosw( )、cacosq( ) - 复变反余弦函数

概要

#include <complex.h>

double complex cacos(double complex z);

float complex cacosf(float complex z);

long double complex cacosl(long double complex z);

extended complex cacosw(extended complex z);

quad complex cacosq(quad complex z);

说明

这些函数仅适用于 Integrity 服务器。

cacos() 返回 *z* 的复变反余弦值，其范围是沿虚轴的数学上无界的带状区域和沿实轴的 [0,Pi] 区间内。

分支切割位于沿实轴的 [-1,+1] 区间之外。

cacosf() 是 **cacos()** 的 **float** 版本；它采用 **float complex** 参数，并返回 **float complex** 结果。

cacosl() 是 **long double complex cacos()** ；它采用 **long double complex** 参数，并返回 **long double complex** 结果。

cacosw() 是 **cacos()** 的 **extended** 版本；它采用 **extended complex** 参数，并返回 **extended complex** 结果。

在 HP-UX 系统中， **cacosq()** 等效于 **cacosl()** 。

用法

要使用这些函数，请使用缺省的 **-Ae** 选项进行编译。要使用 **cacosw()** 或 **cacosq()** ，请使用 **-fpwidetypes** 选项进行编译。确保程序包含 **<complex.h>** ，并通过在编译程序命令行或链接程序命令行上指定 **-lm** 链接到数学库。

返回值

cacos(conj(z)) = conj(**cacos**(z))

cacos(±0+i0) 返回 Pi/2-i0。

cacos(±0+iNaN) 返回 Pi/2+iNaN。

对于有限 x， **cacos**(x+iInf) 返回 Pi/2-iInf。

对于非零有限 x， **cacos**(x+iNaN) 返回 NaN+iNaN，并引起无效浮点运算异常（可选）。

对于有限正号 y， **cacos**(-Inf+iy) 返回 Pi-iInf。

对于有限正号 y， **cacos**(+Inf+iy) 返回 +0-iInf。

cacos(-Inf+iInf) 返回 3Pi/4-iInf。

cacos(+Inf+iInf) 返回 Pi/4-iInf。

cacos(±Inf+iNaN**)** 返回 NaN±iInf （其中未指定结果中的虚数部分的符号）。

对于有限 y， **cacos(**NaN+iy**)** 返回 NaN+iNaN，并引起无效浮点运算异常（可选）。

cacos(NaN+iInf**)** 返回 NaN-iInf。

cacos(NaN+iNaN**)** 返回 NaN+iNaN。

错误

没有定义任何错误。

另请参阅

acos(3M)、 ccos(3M)、 complex(5)。

符合的标准

cacos() 、 **cacosf()** 、 **cacosl()** ： ISO/IEC C99（包括附件 G“IEC 60559-compatible complex arithmetic”）

名称

cacosh( )、cacoshf( )、cacoshl( )、cacoshw( )、cacoshq( ) - 复变反双曲余弦函数

概要

#include <complex.h>

double complex cacosh(double complex z);

float complex cacoshf(float complex z);

long double complex cacoshl(long double complex z);

extended complex cacoshw(extended complex z);

quad complex cacoshq(quad complex z);

说明

这些函数仅适用于 Integrity 服务器。

cacosh() 返回 *z* 的复变反双曲余弦值，取值范围如下：沿实轴方向为非负数值的半个条带区域，而沿虚轴方向则位于区间 [-iPi,iPi] 内。沿着实轴，在值小于 1 的地方有一个分支切割。

cacoshf() 是 **float complex** 版的 **cacosh()** ；它采用 **float complex** 参数，并返回 **float complex** 结果。

cacoshl() 是 **long double complex** 版的 **cacosh()** ；它采用 **long double complex** 参数，并返回 **long double complex** 结果。

cacoshw() 是 **extended complex** 版的 **cacosh()** ；它采用 **extended complex** 参数，并返回 **extended complex** 结果。

cacoshq() 等效于 HP-UX 系统上的 **cacoshl()** 。

用法

要使用这些函数，请使用缺省的 **-Ae** 选项进行编译。要使用 **cacoshw()** 或 **cacoshq()** ，请使用 **-fpwidetypes** 选项进行编译。确保程序包含 **<complex.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

返回值

cacosh(conj(z)) = conj(**cacosh**(z))

cacosh(±0+i0) 将返回 +0+iPi/2。

对于有限值 x， **cacosh**(x+iInf) 将返回 +Inf+iPi/2。

对于有限值 x， **cacosh**(x+iNaN) 将返回 NaN+iNaN 并可引发无效的浮点异常。

对于正有限值 y， **cacosh**(-Inf+iy) 将返回 +Inf+iPi。

对于正有限值 y， **cacosh**(+Inf+iy) 将返回 +Inf+i0。

cacosh(-Inf+iInf) 将返回 +Inf+i3Pi/4。

cacosh(+Inf+iInf) 将返回 +Inf+iPi/4。

cacosh(±Inf+iNaN**)** 将返回 +Inf+iNaN。

对于有限值 y， **cacosh(**NaN+iy**)** 将返回 NaN+iNaN 并可引发无效的浮点异常。

cacosh(NaN+iInf**)** 将返回 +Inf+iNaN。

cacosh(NaN+iNaN**)** 将返回 NaN+iNaN。

错误

没有定义任何错误。

另请参阅

acosh(3M)、 ccosh(3M)、 complex(5)。

符合的标准

cacosh() 、 **cacoshf()** 、 **cacoshl()** ： ISO/IEC C99（包括附件 G“IEC 60559-compatible complex arithmetic”）

名称

can_change_color、color_content、has_colors、init_color、init_pair、 start_color、pair_content — 颜色操作函数

概要

#include <curses.h>

bool can_change_color(void);

int color_content(short *color*, **short ***red*, **short ***green*, **short ***blue*);

int COLOR_PAIR(int *n*);

bool has_colors(void);

int init_color(short *color*, **short** *red*, **short** *green*, **short** *blue*);

int init_pair(short *pair*, **short** *f*, **short** *b*);

int pair_content(short *pair*, **short ***f*, **short ***b*);

int PAIR_NUMBER(int *value*);

int start_color(void);

extern int COLOR_PAIRS;

extern int COLORS;

说明

这些函数用于操作那些支持颜色处理的终端上的颜色。

查询功能

has_colors() 函数指示终端是否为彩色终端。 **can_change_color()** 函数指示终端是否为可以重新定义其上面的颜色的彩色终端。

初始化

在调用任何颜色操作函数之前，必须调用 **start_color()** 函数，以启用颜色。该函数可初始化八种基色（黑色、蓝色、绿色、青色、红色、品红色、黄色和白色）， **<curses.h>** 中定义的颜色宏（例如 **COLOR_BLACK** ）可指定这些基色（请参阅 **<curses.h>** 中的 Colour-related Macros ）。未指定这八种颜色的最初外观。

该函数还可初始化两种全局外部变量：

- **COLORS** 定义终端支持的颜色数（请参阅下面的颜色识别）。如果 **COLORS** 为 0，则表示终端不支持重新定义颜色（同时 **can_change_colour()** 将返回 FALSE）。
- **COLOR_PAIRS** 定义终端支持的颜色对的最大数目（请参阅下面的用户定义的颜色对）。

start_color() 函数还可以将终端上的颜色恢复到终端专用的初始值。所有终端上的初始背景色均假定为黑色。

颜色识别

init_color() 函数重新定义可支持颜色重新定义的终端上的颜色编号 *color*，使 *red*、*green* 和 *blue* 可分别指定红色、绿色和蓝色深度组件。还可以通过调用 **init_color()** 将屏幕上出现的所有指定颜色更改为新的定义。

color_content() 函数可识别颜色编号 *color* 的深度组件。它可以将该颜色的红色、绿色和蓝色深度组件分别存储在 *red*、*green* 和 *blue* 指向的地址中。

对于这两个函数， *color* 参数必须在 0 至 **COLORS** – 1 （包括此限值）的范围内。有效深度值的范围为 0（无深度组件）至 1000（包括此限值，即该组件的最大深度）。

用户定义的颜色对

调用 **init_pair()** 可定义或重新定义颜色对编号 *pair*，以获取前景色 *f* 和背景色 *b*。调用 **init_pair()** 可将颜色对以前的定义中显示的任何字符更改为新的定义，并可刷新屏幕。

定义颜色对后，宏 **COLOR_PAIR**(*n*) 将返回颜色对 *n* 的值。该值为颜色属性，它是从 **chtype** 中提取的。相反，宏 **PAIR_NUMBER**(*value*) 返回与颜色属性 *value* 关联的颜色对编号。

pair_content() 函数检索某个颜色对编号 *pair* 的组件颜色。它将前景色和背景色编号分别存储在 *f* 和 *b* 指向的变量中。

对于 **init_pair()** 和 **pair_content()** ， *pair* 的值必须在 0 至 **COLOR_PAIRS** – 1 （包括此限值）的范围内（*pair* 的有效值可能存在特定实现的下限，但这些限值都至少必须为 63）。 *f* 和 *b* 的有效值在 0 至 **COLORS** – 1 （包括此限值）的范围内。

返回值

如果终端能够操作颜色，则 **has_colors()** 函数返回 TRUE；否则将返回 FALSE。

如果终端支持颜色，并且可以更改这些颜色的定义，则 **can_change_color()** 函数返回 TRUE；否则将返回 FALSE。

如果成功完成，其他函数将返回 OK；否则将返回 ERR。

错误

未定义错误。

实际应用信息

要使用这些函数，通常必须在调用 **initscr()** 后紧接着调用 **start_color()** 。

can_change_color() 和 **has_colors()** 函数有助于写入与终端无关的程序。例如，程序员可使用这些函数判断是否要使用颜色或其他某个视频属性。

在彩色终端上， **COLORS** 的典型值为 8， **COLOR_BLACK** 等宏可返回 0 至 7（包括此限值）范围内的值。但是，应用程序不能依赖于此结果为 True。

另请参阅

attroff(3X)、 delscreen(3X)、 <curses.h>。

历史变更记录

在 X/Open Curses 第 4 期中首次发布。

名称

carg( )、cargf( )、cargl( )、cargw( )、cargq( ) - 复变自变量（也称为相角）函数

概要

#include <complex.h>

double carg(double complex z);

float cargf(float complex z);

long double cargl(long double complex z);

extended cargw(extended complex z);

quad cargq(quad complex z);

说明

这些函数仅适用于 Integrity 服务器。

carg() 返回 *z* 的复变自变量，它位于区间 [-pi,+pi]。

cargf() 是 **carg()** 的 **float complex** 版本；它采用 **float complex** 参数，并返回 **float** 结果。

cargl() 是 **carg()** 的 **long double complex** 版本；它采用 **long double complex** 参数，并返回 **long double** 结果。

cargw() 是 **carg()** 的 **extended complex** 版本；它采用 **extended complex** 参数，并返回 **extended** 结果。

在 HP-UX 系统中， **cargq()** 等效于 **cargl()** 。

用法

要使用这些函数，请使用缺省的 **-Ae** 选项进行编译。要使用 **cargw()** 或 **cargq()** ，请使用 **-fpwidetypes** 选项进行编译。确保程序包含 **<complex.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

返回值

carg(*z***)** 与 **atan2(cimag(***z***),creal(***z***))** 返回的值相同，并引起相同的异常。

错误

没有定义任何错误。

另请参阅

atan2(3M)、 cabs(3M)、 complex(5)。

符合的标准

carg() 、 **cargf()** 和 **cargl()** ： ISO/IEC C99（包括附件 G“IEC 60559-compatible complex arithmetic”）

名称

casin( )、casinf( )、casinl( )、casinw( )、casinq( ) - 复变反正弦函数

概要

#include <complex.h>

double complex casin(double complex z);

float complex casinf(float complex z);

long double complex casinl(long double complex z);

extended complex casinw(extended complex z);

quad complex casinq(quad complex z);

说明

这些函数仅适用于 Integrity 服务器。

casin() 返回 *z* 的复变反正弦值，取值范围如下：沿虚轴方向为在数学角度上没有界限的一个条带区域，而沿实轴方向则位于区间 [-Pi/2,+Pi/2] 内。沿着实轴，在区间 [-1,+1] 之外是分支切割。

casinf() 是 **float complex** 版的 **casin()** ；它采用 **float complex** 参数，并返回 **float complex** 结果。

casinl() 是 **long double complex** 版的 **casin()** ；它采用 **long double complex** 参数，并返回 **long double complex** 结果。

casinw() 是 **extended complex** 版的 **casin()** ；它采用 **extended complex** 参数，并返回 **extended complex** 结果。

casinq() 等效于 HP-UX 系统上的 **casinl()** 。

用法

要使用这些函数，请使用缺省的 **-Ae** 选项进行编译。要使用 **casinw()** 或 **casinq()** ，请使用 **-fpwidetypes** 选项进行编译。确保程序包含 **<complex.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

返回值

casin(conj(z)) = conj(**casin**(z)) 和 **casinh** 属于奇函数。

casin(+0+i0) 将返回 0+i0。

对于正有限值 y， **casin**(Inf+iy) 将返回 Pi/2+iInf。

对于有限值 y， **casin**(NaN+iy) 将返回 NaN+iNaN 并可引发无效异常。

对于正有限值 x， **casin**(x+iInf) 将返回 +0+iInf。

casin(+Inf+iInf) 将返回 Pi/4+iInf。

casin(NaN+iInf) 将返回 NaN+iInf。

casin(+0+iNaN) 将返回 +0+iNaN。

对于非零的有限值 x， **casin**(x+iNaN) 将返回 NaN+iNaN 并可引发无效异常。

casin(Inf+iNaN) 将返回 NaN±iInf （这里未指定结果的虚数部分的符号）。

casin(NaN+iNaN) 将返回 NaN+iNaN。

错误

没有定义任何错误。

另请参阅

asin(3M)、 asinh(3M)、 csin(3M)、 complex(5)。

符合的标准

casin()、 **casinf**()、 **casinl**()： ISO/IEC C99（包括附件 G“IEC 60559-compatible complex arithmetic”）

名称

casinh( )、casinhf( )、casinhl( )、casinhw( )、casinhq( ) - 复变反双曲正弦函数

概要

#include <complex.h>

double complex casinh(double complex z);

float complex casinhf(float complex z);

long double complex casinhl(long double complex z);

extended complex casinhw(extended complex z);

quad complex casinhq(quad complex z);

说明

这些函数仅适用于 Integrity 服务器。

casinh() 返回 *z* 的复变反双曲正弦值，取值范围如下：沿实轴方向为在数学角度上没有界限的一个条带区域，而沿虚轴方向则位于区间 [-i*Pi/2,+i*Pi/2] 内。沿着虚轴，在区间 [-i,+i] 之外是分支切割。

casinhf() 是 **float complex** 版的 **casinh()** ；它采用 **float complex** 参数，并返回 **float complex** 结果。

casinhl() 是 **long double complex** 版的 **casinh()** ；它采用 **long double complex** 参数，并返回 **long double complex** 结果。

casinhw() 是 **extended complex** 版的 **casinh()** ；它采用 **extended complex** 参数，并返回 **extended complex** 结果。

casinhq() 等效于 HP-UX 系统上的 **casinhl()** 。

用法

要使用这些函数，请使用缺省的 **-Ae** 选项进行编译。要使用 **casinhw()** 或 **casinhq()** ，请使用 **-fpwidetypes** 选项进行编译。确保程序包含 **<complex.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

返回值

casinh(conj(z)) = conj(**casinh**(z)) 和 **casinh** 属于奇函数。

casinh(+0+i0) 将返回 0+i0。

对于正有限值 x， **casinh**(x+iInf) 将返回 +Inf+iPi/2。

对于有限值 x， **casinh**(x+iNaN) 将返回 NaN+iNaN 并可引发无效异常。

对于正有限值 y， **casinh**(+Inf+iy) 将返回 +Inf+i0。

casinh(+Inf+iInf) 将返回 +Inf+iPi/4。

casinh(+Inf+iNaN) 将返回 +Inf+iNaN。

casinh(NaN+i0) 将返回 NaN+i0。

对于非零的有限值 y， **casinh(**NaN+iy**)** 将返回 NaN+iNaN 并可引发无效异常。

casinh(NaN+iInf**)** 将返回 ±Inf+iNaN （这里未指定结果的实数部分的符号）。

casinh(NaN+iNaN**)** 将返回 NaN+iNaN。

错误

没有定义任何错误。

另请参阅

csinh(3M)、 complex(5)。

符合的标准

casinh() 、 **casinhf()** 、 **casinhl()** ： ISO/IEC C99（包括附件 G“IEC 60559-compatible complex arithmetic”）

名称

catan( )、catanf( )、catanl( )、catanw( )、catanq( ) - 复变反正切函数

概要

#include <complex.h>

double complex catan(double complex z);

float complex catanf(float complex z);

long double complex catanl(long double complex z);

extended complex catanw(extended complex z);

quad complex catanq(quad complex z);

说明

这些函数仅适用于 Integrity 服务器。

catan() 返回 *z* 的复变反正切值，取值范围如下：沿虚轴方向为在数学角度上没有界限的一个条带区域，而沿实轴方向则位于区间 [-Pi/2,+Pi/2] 内。沿着虚轴，在区间 [-1,+1] 之外是分支切割。

catanf() 是 **float complex** 版的 **catan()** ；它采用 **float complex** 参数，并返回 **float complex** 结果。

catanl() 是 **long double complex** 版的 **catan()** ；它采用 **long double complex** 参数，并返回 **long double complex** 结果。

catanw() 是 **extended complex** 版的 **catan()** ；它采用 **extended complex** 参数，并返回 **extended complex** 结果。

catanq() 等效于 HP-UX 系统上的 **catanl()** 。

用法

要使用这些函数，请使用缺省的 **-Ae** 选项进行编译。要使用 **catanw()** 或 **catanq()** ，请使用 **-fpwidetypes** 选项进行编译。确保程序包含 **<complex.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

返回值

catan(conj(z)) = conj(**catan**(z)) 和 **catan** 属于奇函数。

catan(+0+i0) 将返回 +0+i0。

catan(+NaN+i0) 将返回 +NaN+i0。

catan(+0+i1) 将返回 +0+iInf 并可引发除数为零的浮点异常。

对于正有限值 y， **catan**(Inf+iy) 将返回 Pi/2+i0。

对于非零的有限值 y， **catan**(NaN+iy) 将返回 NaN+iNaN 并可引发无效的浮点异常。

对于正有限值 x， **catan**(+x+iInf) 将返回 Pi/2+i0。

catan(+Inf+iInf) 将返回 Pi/2+i0。

catan(+NaN+iInf**)** 将返回 +NaN+i0。

对于有限值 x， **catan(**x+iNaN**)** 将返回 NaN+iNaN，并可引发无效的浮点异常。

catan(Inf+iNaN**)** 返回 Pi/2±i0 （这里未指定结果的虚数部分的符号）。

catan(NaN+iNaN**)** 将返回 NaN+iNaN。

错误

没有定义任何错误。

另请参阅

atan(3M)、 ctan(3M)、 complex(5)。

符合的标准

catan() 、 **catanf()** 、 **catanl()** ： ISO/IEC C99（包括附件 G“IEC 60559-compatible complex arithmetic”）

名称

catanh( )、catanhf( )、catanhl( )、catanhw( )、catanhq( ) - 复变反双曲正切函数

概要

#include <complex.h>

double complex catanh(double complex z);

float complex catanhf(float complex z);

long double complex catanhl(long double complex z);

extended complex catanhw(extended complex z);

quad complex catanhq(quad complex z);

说明

这些函数仅适用于 Integrity 服务器。

catanh() 返回 *z* 的复变反双曲正切函数，从数学角度来讲，其范围是沿实轴无界的带状区域，和沿虚轴的 [-i*π*/2,i*π*/2] 区间内。分支切割位于实轴的 [-1,+1] 区间之外。

catanhf() 是 **float complex** 版的 **catanh()** ；它采用 **float complex** 参数，并返回 **float complex** 结果。

catanhl() 是 **long double complex** 版的 **catanh()** ；它采用 **long double complex** 参数，并返回 **long double complex** 结果。

catanhw() 是 **extended complex** 版的 **catanh()** ；它采用 **extended complex** 参数，并返回 **extended complex** 结果。

在 HP-UX 系统中， **catanhq()** 等效于 **catanhl()** 。

用法

要使用这些函数，请使用缺省的 **-Ae** 选项进行编译。要使用 **catanhw()** 或 **catanhq()** ，请使用 **-fpwidetypes** 选项进行编译。确保程序包含 **<complex.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

返回值

catanh(conj(*z***)) = conj(catanh(***z***))** 和 **catanh** 为奇数值。

catanh(+0+i0) 返回 +0+i0。

catanh(+0+iNaN) 返回 +0+iNaN。

catanh(+1+i0) 返回 +Inf+i0，并引起除数为零的浮点运算异常。

catanh(*x*+iInf) 返回 +0+i*π*/2，而对于有限正 *x* ，返回。

对于非零有限 *x* ， **catanh**(*x*+iNaN) 返回 NaN+iNaN，并可能引起无效浮点运算异常。

catanh(+Inf+i*y*) 返回 +0+i*π*/2，对于有限正 *y* ，返回。

catanh(+Inf+iInf) 返回 +0+i*π*/2、。

catanh(+Inf+iNaN) 返回 +0+iNaN。

对于有限 *y* ， **catanh**(NaN+i*y*) 返回 NaN+iNaN，并可能引起无效浮点运算异常。

catanh(NaN+iInf) 返回 ±0+iπ/2 （此处未指定结果的实数部分的符号）。

catanh(NaN+iNaN) 返回 NaN+iNaN。

错误

没有定义任何错误。

另请参阅

atanh(3M)、 ctanh(3M)、 complex(5)。

符合的标准

catanh() 、 **catanhf()** 、 **catanh()** ： ISO/IEC C99（包括附件 G“IEC 60559-compatible complex arithmetic”）

名称

catgets( ) - 获取程序消息

概要

```
#include <nl_types.h>

char *catgets(
  nl_catd catd,
  int set_num,
  int msg_num,
  const char *def_str
);
```

说明

catgets() 函数将从 **catd** （一个从上一次 **catopen()** 调用中返回的清单描述符）所标识的消息清单的集合 **set_num** 中读取消息 **msg_num** （请参阅 *catopen*(3C) ）。如果调用失败， **def_str** 就会指向 **catgets()** 返回的缺省消息字符串。

长度超过 **NL_TEXTMAX** 个字节的消息将被截断。返回的消息字符串始终以 null （空）字节结尾。**NL_TEXTMAX** 是在 **<limits.h>** 中定义的。

外部语言环境影响

国际代码集支持

支持单字节字符代码集和多字节字符代码集。

返回值

如果调用成功， **catgets()** 就会返回一个指针，指向由 null（空）结尾的消息字符串所在的内部缓冲区。如果调用不成功， **catgets()** 就会返回一个指向 **def_str**的指针。

错误

在以下任意一种情况下， **catgets()** 将失败并设置 **errno** ：

[EBADF] **catd** 不是有效的清单描述符。

[EINTR] 在 *read*(2) 系统调用中捕获到信号。

[EINVAL] 由 **catd** 标识的消息清单已损坏。

[ENOMSG] 由 **set_num** 或 **msg_num** 标识的消息不在消息清单内。

[ERANGE] 长度超过 **NL_TEXTMAX** 个字节的消息被截断。

警告

catgets() 函数将返回一个指针，指向每个调用上被覆盖的静态区域。

作者

catgets() 由 HP 开发。

文件

/usr/include/nl_types.h

另请参阅

dumpmsg(1)、findmsg(1)、gencat(1)、catclose(3C)、catopen(3C)、 thread_safety(5)。

符合的标准

catgets()：AES、SVID3、XPG2、XPG3、XPG4

名称

catopen( )、catclose( ) - 打开和关闭要读取的消息清单

概要

#include <nl_types.h>

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

int catclose(nl_catd catd);

说明

catopen() 函数可打开消息清单，并返回一个清单描述符。 **name** 指定打开的消息清单的名称。包含斜线 (*/*) 的 **name** 指定消息清单的路径名。否则，将使用环境变量 **NLSPATH** （请参阅 *environ*(5) ）。如果 **NLSPATH** 指定了多个路径， **catopen()** 将返回该函数可以在其上成功打开指定的消息清单的第一个路径的清单描述符。如果环境中不存在 **NLSPATH** ，或者无法打开任何 **NLSPATH**指定的路径的消息清单， **catopen()** 将使用系统级的缺省路径。如果 **oflag** 的值为 **NL_CAT_LOCALE** ， **LC_MESSAGES** 将影响缺省值。如果 **oflag** 的值为零，则环境变量 **LANG** 将影响缺省值。有关详细信息，请参阅 *environ*(5) 。

如果 **catopen()** 是通过带有所有者超级用户权限的 **setuid** 或 **setgid** 程序调用的，则不应直接使用环境变量 **NLSPATH** 来查找消息清单，而是应考虑使用配置文件 **/etc/default/nlspath** 和环境变量 **NLSPATH** 两者提供的路径来查找消息清单。有关详细信息，请参阅 *nlspath*(4) 。

例如：如果环境变量设置为：

NLSPATH=/usr/lib/nls/msg/%L/%N.cat:/tmp/%L/%N.cat

并且 **/etc/default/nlspath** 配置文件包含条目：

NLSPATH=/usr/lib/nls/msg/%L/%N.cat

则只能考虑使用路径 **/usr/lib/nls/msg/%L/%N.cat** 来查找清单文件。提供此功能只是为了实现依赖于 **NLSPATH** 环境变量的那些 **setuid** 或 **setgid** 根程序的向后兼容。所有新的 **setgid** 或 **setuid** 根程序均不应依赖于 **NLSPATH** 环境变量，并且只应该使用绝对路径名。

只有当某个进程关闭了某个消息清单描述符，或者当成功调用了 **exec()** 函数之一后，该消息清单描述符才会保持有效。更改 **LC_MESSAGES** 类别的设置可能会使现有的已打开清单失效。

文件描述符用于实现消息清单描述符，并为消息清单描述符设置了 close-on-exec 标记（请参阅 *fcntl*(2) ）。

如果 **oflag** 为零，将使用 **LANG** 环境变量来查找清单。如果 **oflag** 为 **NL_CAT_LOCALE** ，则只有在调用 **catopen()** 之前已成功调用 **setlocale()** 的情况下，才可使用 **LC_MESSAGES** 类别来查找消息清单。将 **oflag** 设置为其他任何值，其结果是不确定的。

catclose() 函数可关闭消息清单 **catd** ，也就是以前成功调用 **catopen()** 后返回的消息清单描述符。

返回值

如果成功， **catopen()** 将返回一个消息清单描述符。否则， **catopen()** 将返回值 (*nl_catd*) − 1 ，并设置 **errno** 以指明错误。

如果成功， **catclose()** 将返回零。否则， **catclose()** 将返回 −1，并设置 **errno** 以指明错误。

错误

出现下列任一情况时， **catopen()** 将会失败，并且不会打开消息清单，同时针对最后一个尝试的路径设置 **errno**：

[EACCES]	路径前缀的某部分拒绝了搜索权限，或者拒绝了对指定文件的读取权限。
[EMFILE]	当前打开了允许的最大数目的文件描述符。
[ENAMETOOLONG]	指定的路径名的长度超过 **PATH_MAX** 个字节；或者，当 **_POSIX_NO_TRUNC** 有效时，路径名的某个组件的长度超过 **NAME_MAX** 个字节。
[ENFILE]	系统文件表已满。
[ENOENT]	指定的清单不存在，或者路径为空。
[ENOTDIR]	路径前缀的某部分不是目录。

如果在 **catopen()** （请参阅 *catgets*(3C) ）失败后接着调用 **catgets()** ，则可以使用此函数来提供缺省消息。如果向 **catgets()** 传递了一个无效的清单描述符，该函数将返回其 **def_str** 参数。

如果存在以下情况， **catclose()** 将会失败：

[EBADF]	**catd** 不是有效的已打开消息清单描述符。

警告

使用 **NLSPATH** 时， **catopen()** 不会为 **LANG** 提供缺省值。

作者

catopen() 和 **catclose()** 由 HP 开发。

文件

/usr/include/nl_types.h	包含本地语言类型的包含文件。
/usr/lib/nls/msg	只针对于 HP-UX 核心产品的消息清单缺省路径。

另请参阅

fcntl(2)、 catgets(3C)、 setlocale(3C)、 nlspath(4)、 environ(5)、 thread_safety(5)。

符合的标准

catopen()：AES、SVID3、XPG2、XPG3、XPG4

catclose()：AES、SVID3、XPG2、XPG3、XPG4

名称

cbreak、nocbreak、noraw、raw - 输入模式控制函数

概要

#include <curses.h>

int cbreak(void);

int nocbreak(void);

int noraw(void);

int raw(void);

说明

cbreak() 函数可将当前终端的输入模式设置为 *cbreak* 模式并覆盖对 **raw()** 的调用。

nocbreak() 函数可将当前终端的输入模式设置为精细处理模式 (Cooked Mode)，但不更改 ISIG 和 IXON 的状态。

noraw() 函数可将当前终端的输入模式设置为精细处理模式 (Cooked Mode)，并设置 ISIG 和 IXON 标志。

raw() 函数可将当前终端的输入模式设置为原始模式。

返回值

成功完成后，这些函数将返回 OK。否则，它们将返回 ERR。

错误

未定义错误。

实际应用信息

如果在应用程序调用 **initscr()** 时无法确定进程所处的输入模式，应使用这些函数指定所需的输入模式。

另请参阅

curses_intro(3X)、 Input Mode 小节、<curses.h>。

第 2 版，第 4 期 X/Open System Interface Definitions 规范第 9 章： General Terminal Interface 。

历史变更记录

第一版发行在 X/Open Curses 第 2 期中。

X/Open Curses 第 4 期

raw() 和 **noraw()** 函数已与此条目合并。在以前各期中，这些函数出现在各自的条目中。

此条目已被重新编写，用于进行明确说明。所有这些函数的参数列表已明确声明为 **void**。

名称

cbrt( )、cbrtf( )、cbrtl( )、cbrtw( )、cbrtq( ) - 立方根函数

概要

#include <math.h>

double cbrt(double x);

float cbrtf(float x);

仅适用于 **HP Integrity** 服务器

long double cbrtl(long double x);

extended cbrtw(extended x);

quad cbrtq(quad x);

说明

cbrt() 计算 *x* 的立方根。

cbrtf() 是 **float** 版的 **cbrt()**；它采用 **float** 参数，并返回 **float** 结果。

仅适用于 **Integrity** 服务器

cbrtl() 是 **long double** 版的 **cbrt()**；它采用 **long double** 参数，并返回 **long double** 结果。

cbrtw() 是 **extended** 版的 **cbrt()**；它采用 **extended** 参数，并返回 **extended** 结果。

在 HP-UX 系统中，**cbrtq()** 等效于 **cbrtl()**。

用法

要使用这些函数，请使用缺省的 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。

（对于 Integrity 服务器）要使用 **cbrtw()** 或 **cbrtq()**，也可以使用 **-fpwidetypes** 选项进行编译。

要使用上面的任何函数，请确保程序包含 **<math.h>**，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

有关详细信息，请参阅《HP-UX floating-point guide for HP Integrity servers》（位于以下站点：**http://www.hp.com/go/fp**）。

返回值

如果 *x* 为 ±INFINITY 或 ±0，则 **cbrt()** 返回 *x*。

如果 *x* 为 NaN，则 **cbrt()** 返回 NaN。

只要四舍五入的结果不等于精确结果，**cbrt()** 就引起不精确异常。

错误

没有定义任何错误。

另请参阅

exp(3M)、 log(3M)、 pow(3M)、 sqrt(3M)、 math(5)。

符合的标准

cbrt()： SVID3、XPG4.2 和 ISO/IEC C99（包括附件 F“IEC 60559 floating-point arithmetic”）

cbrtf() 和 **cbrtl()**： ISO/IEC C99（包括附件 F“IEC 60559 floating-point arithmetic”）

名称

ccos( )、ccosf( )、ccosl( )、ccosw( )、ccosq( ) - 复变余弦函数

概要

#include <complex.h>

double complex ccos(double complex z);

float complex ccosf(float complex z);

long double complex ccosl(long double complex z);

extended complex ccosw(extended complex z);

quad complex ccosq(quad complex z);

说明

这些函数仅适用于 Integrity 服务器。

ccos() 返回 *z* 的复变余弦值。

ccosf() 是 **ccos()** 的 **float complex** 版本；它采用 **float complex** 参数，并返回 **float complex** 结果。

ccosl() 是 **ccos()** 的 **long double complex** 版本；它采用 **long double complex** 参数，并返回 **long double complex** 结果。

ccosw() 是 **ccos()** 的 **extended complex** 版本；它采用 **extended complex** 参数，并返回 **extended complex** 结果。

在 HP-UX 系统中， **ccosq()** 等效于 **ccosl()** 。

用法

要使用这些函数，请使用缺省的 **-Ae** 选项进行编译。要使用 **ccosw()** 或 **ccosq()** ，请使用 **-fpwidetypes** 选项进行编译。确保程序包含 **<complex.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

返回值

ccos(conj(*z***)) = conj(ccos(***z***))** ，并且 **ccos** 为偶数。

ccos(+0+i0**)** 返回 1-i0。

ccos(+0+iInf**)** 返回 NaN±i0 （此处未指定结果的虚数部分的符号），并引起无效浮点运算异常。

ccos(+0+iNaN**)** 返回 NaN±i0 （此处未指定结果的虚数部分的符号）。

对于非零有限 *x* ， **ccos(***x***+**iInf**)** 返回 NaN+iNaN，并引起无效浮点运算异常。

对于非零有限 *x* ， **ccos(***x***+**iNaN**)** 返回 NaN+iNaN，并可能引起无效浮点运算异常。

ccos(+Inf+i0**)** 返回 +Inf-i0。

对于非零有限 *y* ， **ccos(**+Inf+i*y***)** 返回 +Inf(cos(y)-isin(y))。

ccos(+Inf+iInf**)** 返回 ±Inf+iNaN （此处未指定结果的实数部分的符号），并引起无效浮点运算异常。

ccos(+Inf+iNaN**)** 返回 +Inf+iNaN。

ccos(NaN+i0**)** 返回 NaN±i0 （此处未指定结果的虚数部分的符号）。

对于所有非零 *y* ， **ccos(**NaN+i*y*) 返回 NaN+iNaN，并可能引起无效浮点运算异常。

ccos(NaN+iNaN**)** 返回 NaN+iNaN。

错误

没有定义任何错误。

另请参阅

cos(3M)、 cacos(3M)、 complex(5)。

符合的标准

ccos() 、 **ccosf()** 和 **ccosl()** ： ISO/IEC C99（包括附件 G“IEC 60559-compatible complex arithmetic”）

名称

ccosh( )、ccoshf( )、ccoshl( )、ccoshw( )、ccoshq( ) - 复数双曲余弦函数

概要

#include <complex.h>

double complex ccosh(double complex z);

float complex ccoshf(float complex z);

long double complex ccoshl(long double complex z);

extended complex ccoshw(extended complex z);

quad complex ccoshq(quad complex z);

说明

这些函数仅适用于 Integrity 服务器。

ccosh() 返回 *z* 的复数双曲余弦值。

ccoshf() 是 **float complex** 版的 **ccosh()** ；它采用 **float complex** 参数，并返回 **float complex** 结果。

ccoshl() 是 **long double complex** 版的 **ccosh()** ；它采用 **long double complex** 参数，并返回 **long double complex** 结果。

ccoshw() 是 **extended complex** 版的 **ccosh()** ；它采用 **extended complex** 参数，并返回 **extended complex** 结果。

ccoshq() 等效于 HP-UX 系统上的 **ccoshl()** 。

用法

要使用这些函数，请使用缺省的 **-Ae** 选项进行编译。要使用 **ccoshw()** 或 **ccoshq()** ，请使用 **-fpwidetypes** 选项进行编译。确保程序包含 **<complex.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

返回值

ccosh(conj(z)) = **conj(ccosh**(z)) 和 **ccosh** 属于偶函数。

ccosh(+0+i0) 将返回 1+i0。

ccosh(+0+iInf) 将返回 NaN±i0 （这里未指定结果的虚数部分的符号），并可引发无效的浮点异常。

ccosh(+0+iNaN) 将返回 NaN±i0 （这里未指定结果的虚数部分的符号）。

对于非零的有限值 *x* ， **ccosh**(*x*+iInf) 将返回 NaN+iNaN，并可引发无效的浮点异常。

对于非零的有限值 *x* ， **ccosh**(*x*+iNaN) 将返回 NaN+iNaN，并可引发无效的浮点异常。

ccosh(+Inf+i0) 将返回 +Inf+i0。

对于非零的有限值 *y* ， **ccosh**(+Inf+i*y*) 将返回 +Inf(**cos**(*y*)**+isin**(*y*)) 。

ccosh(+Inf+iInf) 将返回 ±Inf+iNaN （这里未指定结果的实数部分的符号），并可引发无效的浮点异常。

ccosh(+Inf+iNaN**)** 将返回 +Inf+iNaN。

ccosh(NaN+i0**)** 将返回 NaN±i0 （这里未指定结果的虚数部分的符号）。

对于所有非零的数字值 *y*， **ccosh(**NaN+i*y*) 将返回 NaN+iNaN，并可引发无效的浮点异常。

ccosh(NaN+iNaN**)** 将返回 NaN+iNaN。

错误

没有定义任何错误。

另请参阅

cosh(3M)、 cacosh(3M)、 complex(5)。

符合的标准

ccosh() 、 **ccoshf()** 、 **ccoshf()** ： ISO/IEC C99（包括附件 G“IEC 60559-compatible complex arithmetic”）

名称

ceil( )、ceilf( )、ceill( )、ceilw( )、ceilq( ) - 上限函数

概要

#include <math.h>

double ceil(double x);

仅适用于 **HP Integrity** 服务器

float ceilf(float x);

long double ceill(long double x);

extended ceilw(extended x);

quad ceilq(quad x);

说明

ceil() 将返回不小于 *x* 的最小整数值（表示为双精度数）。

仅适用于 **Integrity** 服务器

ceilf() 是 **float** 形式的 **ceil()**；它采用 **float** 参数，并返回 **float** 结果。

ceill() 是 **long double** 形式的 **ceil()**；它采用 **long double** 参数，并返回 **long double** 结果。

ceilw() 是 **extended** 形式的 **ceil()**；它采用 **extended** 参数，并返回 **extended** 结果。

ceilq() 等效于 HP-UX 系统上的 **ceill()**。

用法

（对于 Integrity 服务器）要使用 **ceilf()**，请使用缺省的 **-Ae** 选项或 **-Aa** 选项编译。

（对于 Integrity 服务器）要使用 **ceill()**、**ceilw()** 或 **ceilq()**，请使用缺省的 **-Ae** 选项或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。

（对于 Integrity 服务器）要使用 **ceilw()** 或 **ceilq()**，也可以使用 **-fpwidetypes** 选项进行编译。

要使用这些函数，请确保程序包含 **<math.h>**，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

有关详细信息，请参阅《HP-UX floating-point guide for HP Integrity servers》（位于以下站点：**http://www.hp.com/go/fp**）。

返回值

如果 *x* 为 ±INFINITY 或 ±0，则 **ceil()** 将返回 *x*。

如果 *x* 为 NaN，则 **ceil()** 将返回 NaN。

如果 *x* 不是整数或无穷大，则 **ceil()** 将可能会引发不精确异常。

错误

没有定义任何错误。

另请参阅

fabs(3M)、 floor(3M)、 fmod(3M)、 rint(3M)、 math(5)。

符合的标准

ceil() ： SVID3、XPG4.2、ANSI C、ISO/IEC C99 （包括附件 F“IEC 60559 floating-point arithmetic”）

ceilf() 、 **ceill()** ： ISO/IEC C99（包括附件 F“IEC 60559 floating-point arithmetic”）

名称

cexp( )、cexpf( )、cexpl( )、cexpw( )、cexpq( ) - 复变指数函数

概要

#include <complex.h>

double complex cexp(double complex z);

float complex cexpf(float complex z);

long double complex cexpl(long double complex z);

extended complex cexpw(extended complex z);

quad complex cexpq(quad complex z);

说明

这些函数仅适用于 Integrity 服务器。

cexp() 返回 *z* 的基数为 e 的复变指数值。

cexpf() 是 **cexp()** 的 **float complex** 版本；它采用 **float complex** 参数，并返回 **float complex** 结果。

cexpl() 是 **cexp()** 的 **long double complex** 版本；它采用 **long double complex** 参数，并返回 **long double complex** 结果。

cexpw() 是 **cexp()** 的 **extended complex** 版本；它采用 **extended complex** 参数，并返回 **extended complex** 结果。

在 HP-UX 系统中， **cexpq()** 等效于 **cexpl()** 。

用法

要使用这些函数，请使用缺省的 **-Ae** 选项进行编译。要使用 **cexpw()** 或 **cexpq()** ，请使用 **-fpwidetypes** 选项进行编译。确保程序包含 **<complex.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

返回值

cexp(conj(*z***)) = conj(cexp(***z***))** 。

cexp(±0+i0**)** 返回 1+i0。

对于有限 *x* ， **cexp(***x***+**iInf**)** 返回 NaN+iNaN，并引起无效浮点运算异常。

对于有限 *x* ， **cexp(***x***+**iNaN**)** 返回 NaN+iNaN，并可能引起无效浮点运算异常。

cexp(+Inf+i0**)** 返回 +Inf+i0。

对于有限 *y* ， **cexp(**-Inf+i*y***)** 返回 +0**(cos(***y***)+isin(***y***))** 。

对于非零有限 *y* ， **cexp(**+Inf+i*y***)** 返回 +Inf**(cos(***y***)+isin(***y***))** 。

cexp(-Inf+iInf**)** 返回 ±0±i0 （此处未指定结果的实数部分和虚数部分的符号）。

cexp(+Inf+iInf**)** 返回 ±Inf+iNaN 并引起无效浮点运算异常（此处未指定结果的实数部分的符号）。

cexp(-Inf+iNaN) 返回 ±0±i0 （此处未指定结果的实数部分和虚数部分的符号）。

cexp(+Inf+iNaN) 返回 ±Inf+iNaN （此处未指定结果的实数部分的符号）。

cexp(NaN+i0) 返回 NaN+i0。

对于所有非零 *y* ， **cexp**(NaN+iy) 返回 NaN+iNaN，并可能引起无效浮点运算异常。

cexp(NaN+iNaN) 返回 NaN+iNaN。

错误

没有定义任何错误。

另请参阅

exp(3M)、 cis(3M)、 clog(3M)、 complex(5)。

符合的标准

cexp() 、 **cexpf**() 和 **cexpl**() ： ISO/IEC C99（包括附件 G“IEC 60559-compatible complex arithmetic”）

名称

cfgetospeed( )、cfsetospeed( )、cfgetispeed( )、cfsetispeed( ) - tty 波特率函数

概要

#include <termios.h>

speed_t cfgetospeed(const struct termios *termios_p);

int cfsetospeed(struct termios *termios_p, speed_t speed);

speed_t cfgetispeed(const struct termios *termios_p);

int cfsetispeed(struct termios *termios_p, speed_t speed);

说明

这些函数设置和获得由 *termios_p* 引用的 *termios* 结构中的输入和输出速率代码。 *termios* 结构包含这些表示输入和输出波特率的速率代码，以及其他与终端相关的参数。在终端文件中设置的参数只有在成功调用 **tcsetattr()** 之后才会生效。

cfgetospeed() 返回来自 *termios_p* 引用的 *termios* 结构的输出速率代码。

cfsetospeed() 将由 *termios_p* 引用的 *termios* 结构中的输出速率代码设置为 *speed* 。代表零波特率的速率代码 **B0** 用来终止连接。如果指定了 **B0** ，则不再声明调制解调器控制线路，正常情况下这会断开该线路。

cfgetispeed() 返回来自 *termios_p* 引用的 *termios* 结构中的输入速率代码。

cfsetispeed() 将由 *termios_p* 引用的 *termios* 结构中的输入速率代码设置为 *speed* 。

返回值

cfgetospeed() 返回来自 *termios_p* 引用的 *termios* 结构中的输出速率代码。

cfgetispeed() 返回来自 *termios_p* 引用的 *termios* 结构中的输入速率代码。

如果成功完成， **cfsetispeed()** 和 **cfsetospeed()** 返回零。否则，它们返回 -1，并将 **errno** 设置为指示出错。

错误

当遇到以下条件时， **cfsetispeed()** 和 **cfsetospeed()** 失败：

[EINVAL] *speed* 的值超出了在 **<termios.h>** 中指定的可能速率代码范围。

警告

cfsetispeed() 和 **cfsetospeed()** 可用来设置终端硬件不支持的 *termios* 结构中的速率代码。

另请参阅

tcattribute(3C)、thread_safety(5)、termio(7)。

符合的标准

cfgetispeed()：AES、SVID3、XPG3、XPG4、FIPS 151-2 和 POSIX.1

cfgetospeed()：AES、SVID3、XPG3、XPG4、FIPS 151-2 和 POSIX.1

cfsetispeed()：AES、SVID3、XPG3、XPG4、FIPS 151-2 和 POSIX.1

cfsetospeed()：AES、SVID3、XPG3、XPG4、FIPS 151-2 和 POSIX.1

名称

chgat、mvchgat、mvwchgat、wchgat — 更改窗口中字符的显示形式

概要

```
#include <curses.h>

int chgat(int n, attr_t attr, short color, const void *opts);

int mvchgat(int y, int x, int n, attr_t attr, short color,
        const void *opts);

int mvwchgat(WINDOW *win, int y, int x, int n, attr_t attr,
         short color, const void *opts);

int wchgat(WINDOW *win, int n, attr_t attr, short color,
        const void *opts);
```

说明

这些函数可更改当前窗口或指定窗口中后 *n* 个字符的显示形式（如果 *n* 为 −1，则更改当前行的剩余字符的显示形式），而起始位置将从当前光标位置或指定的光标位置开始。属性和颜色由 **setcchar()** 的 *attr* 和 *color* 指定。

这些函数不会对光标进行更新。这些函数不执行换行操作。

如果 *n* 的值大于一行上的剩余字符数，则其不属于错误。

opts 参数将保留用于在此文档未来的版本中进行定义。目前，应用程序必须为 *opts* 提供一个空指针。

返回值

成功完成后，这些函数将返回 OK。否则，它们将返回 ERR。

错误

未定义任何错误。

另请参阅

setcchar(3X)、 <curses.h>。

历史变更记录

在 X/Open Curses 第 4 期中首次发布。

名称

chownacl( ) - 更改文件访问控制列表 (ACL) 中指示的拥有者和（或）组（仅适用于 HFS 文件系统）

概要

```
#include <acllib.h>

void chownacl(
  int nentries,
  const struct acl_entry *acl,
  uid_t olduid,
  gid_t oldgid,
  uid_t newuid,
  gid_t newgid
);
```

备注：

为确保持续符合新的行业标准，将来的发行版可能会更改此手册条目中描述的功能。

说明

将旧文件连同访问控制列表 (ACL) 复制到新文件后，此例行程序可以更改 ACL ，以反映文件拥有者或组 ID 发生的更改。 **chownacl()** 可通过类似于 **chown()** （请参阅 *chown*(2) ）的方式转移所有权（也就是说，修改基础 ACL 条目）。下文及 *acl*(5) 中介绍了算法。

nentries 参数为 **acl[]** 数组中的 ACL 的当前条目数（零个或多个；负值被视为零）。 *olduid* 和 *oldgid* 值为原始文件的拥有者的用户 ID 和组 ID ，通常为 **stat()** （请参阅 *stat*(2) ）的 **st_uid** 和 **st_gid** 值。 *newuid* 和 *newgid* 值为新文件的拥有者的用户 ID 和组 ID ，通常为 **geteuid()** 和 **getegid()** （请参阅 *getuid*(2) 中的 *geteuid*(2) 和 *getegid*(2) ）的返回值。

如果 **acl[]** 中的 ACL 条目包含 *olduid* 的 *uid* ，以及 **ACL_NSGROUP** 的 *gid* （也就是拥有者基础 ACL 条目），**chownacl()** 会将 *uid* 更改为 *newuid* （有例行情况 – 请参阅下文）。如果某个条目包含 **ACL_NSUSER** 的 *uid* ，以及 *oldgid* 的 *gid* （也就是组基础 ACL 条目）， **chownacl()** 会将 *gid* 更改为 *newgid* 。无论哪种情况，都只更改最后一个匹配的 ACL 条目；有效的 ACL 只能具有其中一个类型。

对于 *chown*(2) ，如果新用户或新组已包含一个 ACL 条目（也就是说， *newuid* 的 *uid* 及 **ACL_NSGROUP** 的 *gid*；或者 **ACL_NSUSER** 的 *uid* 及 *newgid* 的 *gid* ） ， **chownacl()** 将不更改旧用户或组基础 ACL 条目；将同时保留新旧 ACL 条目。

作为一种特殊情况，如果 *olduid* (*oldgid*) 等于 *newuid* (*newgid*) ， **chownacl()** 将不会在 **acl[]** 中搜索要更改的旧用户（组）基础 ACL 条目。如果在 *olduid* 等于 *newuid* ， *oldgid* 等于 *newgid* 的情况下调用 **chownacl()** ，该例行程序将不起作用。

建议使用

该例行程序可用于创建其原始件由（或可能由）不同的用户或组拥有的文件新副本或替换副本的程序，以及将旧文件的 ACL 复制到新文件的程序。复制另一个用户的文件和（或）另一个组的文件等效于先获取原始文件的拥有

者和（或）组副本，然后使用 **chown()** 将文件转移到新的拥有者和（或）组。如果只是更改文件的所有权，则不必使用此例行程序；在这种情况下， **chown()** 可适当地修改 ACL 。

如果程序还要将文件的其他模式位从旧文件复制到新文件，则必须使用 **chmod()** （请参阅 *chmod*(2) ）。但由于 **chmod()** 将删除可选的 ACL 条目，因此必须在调用 **setacl()** （请参阅 *setacl*(2) ）之前调用此例行程序。此外，为避免新文件暂时失去保护， **chmod()** 调用只应设置文件的其他模式位，而将所有访问权限模式位设置为零（也就是说，使用 07000 屏蔽模式）。 **cpacl()** 库调用可封装此操作，同时还相应地处理远程文件。

举例

以下代码段将从 **oldfile** 获取 **stat()** 信息和 ACL ，同时将 **newfile** 的所有权转移给调用方，并将 ACL 修改为 **newfile** 。

```
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/acl.h>

int nentries;
struct acl_entry acl [NACLENTRIES];
struct stat statbuf;

if (stat ("oldfile", & statbuf) < 0)
   error (...);

if ((nentries = getacl ("oldfile", NACLENTRIES, acl)) < 0)
   error (...);

chownacl (nentries, acl, statbuf.st_uid, statbuf.st_gid,
   geteuid(), getegid());

if (setacl ("newfile", nentries, acl))
   error (...);
```

相关内容

只有标准的 HP-UX 操作系统上的 HFS 文件系统才支持 **chownacl()** 。

作者

chownacl() 由 HP 开发。

另请参阅

chown(2) 、 getacl(2) 、 getegid(2) 、 geteuid(2) 、 getuid(2) 、 setacl(2) 、 stat(2) 、 acltostr(3C) 、 cpacl(3C) 、 setaclentry(3C)、strtoacl(3C)、 acl(5)、thread_safety(5)。

名称

cimag()、cimagf()、cimagl()、cimagw()、cimagq() - 复数值的虚数部分

概要

#include <complex.h>

double cimag(double complex z);

float cimagf(float complex z);

long double cimagl(long double complex z);

extended cimagw(extended complex z);

quad cimagq(quad complex z);

说明

这些函数仅适用于 Integrity 服务器。

cimag() 返回 *z* 的虚数部分（作为实数）。

cimagf() 是 **cimag()** 的 **float complex** 版本；它采用 **float complex** 参数，并返回 **float** 结果。

cimagl() 是 **cimag()** 的 **long double complex** 版本；它采用 **long double complex** 参数，并返回 **long double** 结果。

cimagw() 是 **cimag()** 的 **extended complex** 版本；它采用 **extended complex** 参数，并返回 **extended** 结果。

在 HP-UX 系统中， **cimagq()** 等效于 **cimagl()** 。

这些函数不引起浮点运算异常。

用法

要使用这些函数，请使用缺省的 **-Ae** 选项进行编译。要使用 **cimagw()** 或 **cimagq()** ，请使用 **-fpwidetypes** 选项进行编译。确保程序包含 **<complex.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

错误

没有定义任何错误。

另请参阅

creal(3M)、 complex(5)。

符合的标准

cimag() 、 **cimagf()** 和 **cimagl()** ： ISO/IEC C99（包括附件 G“IEC 60559-compatible complex arithmetic”）

名称

cis( )、cisf( )、cisl( )、cisw( )、cisq( ) - 带有单位量值和给定角度（以弧度指定）的复数值

概要

#include <complex.h>

double complex cis(double x);

float complex cisf(float x);

long double complex cisl(long double x);

extended complex cisw(extended x);

quad complex cisq(quad x);

说明

这些函数仅适用于 Integrity 服务器。

cis() 返回实数部分和虚数部分分别为 *x* 的余弦值和正弦值的复数值。 **cis(***x***)** 等效于 **cexp(***ix***)** 。

cisf() 是 **float complex** 版的 **cis()** ； 它采用 **float** 参数，并返回 **float complex** 结果。

cisl() 是 **long double complex** 版的 **cis()** ； 它采用 **long double** 参数，并返回 **long double complex** 结果。

cisw() 是 **extended complex** 版的 **cis()** ； 它采用 **extended** 参数，并返回 **extended complex** 结果。

cisq() 等效于 HP-UX 系统上的 **cisl()** 。

用法

要使用这些函数，请使用缺省的 **-Ae** 选项进行编译。要使用 **cisw()** 或 **cisq()** ，请使用 **-fpwidetypes** 选项进行编译。确保程序包含 **<complex.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

返回值

cis(-*x***) = conj(cis(***x***))** 。

cis(±0**)** 将返回 1±i0。

cis(+Inf**)** 将返回 NaN+iNaN。

cis(NaN**)** 将返回 NaN+iNaN。

错误

没有定义任何错误。

另请参阅

cexp(3M)、 sincos(3M)、 sincosd(3M)、 complex(5)。

符合的标准

任何标准均未指定这些函数。

名称

clear、erase、wclear 和 werase — 清除窗口

概要

#include <curses.h>

int clear(void);

int erase(void);

int wclear(WINDOW *_win_);

int werase(WINDOW *_win_);

说明

clear() 、 **erase()** 、 **wclear()** 和 **werase()** 函数清除当前窗口或指定窗口中的每一个位置。

clear() 和 **wclear()** 函数还可以达到与调用 **clearok()** 相同的效果，因此，为窗口下一次调用 **wrefresh()** 时，将完全清除窗口并完整地刷新窗口。

返回值

成功完成后，这些函数返回 OK。否则，将返回 ERR。

错误

没有定义任何错误。

另请参阅

clearok(3X)、 doupdate(3X)、 <curses.h>。

历史变更记录

在 X/Open Curses 第 2 期中首次发布。

X/Open Curses 第 4 期

erase() 和 **werase()** 函数在该条目中合并。在先前的几期中，它们都出现在自己的条目中。

为清楚起见，重新编写了该条目。 **clear()** 和 **erase()** 函数的参数列表已显式声明为 **void**。

名称

clearenv - 清除进程环境

概要

#include <stdlib.h>

int clearenv(void);

说明

clearenv() 清除进程环境。在调用 **clearenv()** 后没有立即定义环境变量。

clearenv() 修改指针 *environ* 的值。这意味着，在调用 **clearenv()** 后该指针的副本是无效的。

返回值

成功完成后， **clearenv()** 将返回零；否则，将返回 –1，并将 **errno** 设置为指示出错。

错误

如果遇到以下情况， **clearenv()** 将失败：

[ENOMEM] 为进程环境释放或重新分配内存失败。

另请参阅

environ(5)、getenv(3C)、putenv(3C)、 thread_safety(5)、 <stdlib.h>。

符合的标准

clearenv()： AES

名称

clearok()、 idlok()、 leaveok()、 scrollok()、 setscrreg()、 wsetscrreg() - 终端输出控制函数

概要

#include <curses.h>

int clearok(WINDOW ****win*, bool** *bf***);**

int idlok(WINDOW ****win*, bool** *bf***);**

int leaveok(WINDOW ****win*, bool** *bf***);**

int scrollok(WINDOW ****win*, bool** *bf***);**

int setscrreg(int *top***, int** *bot***);**

int wsetscrreg(WINDOW ****win*, int** *top***, int** *bot***);**

说明

这些函数可设置 Curses 内处理输出的那些选项。

clearok() 函数将 *bf* 的值分配到指定窗口内控制刷新过程中清除屏幕的内部标记。如果在指定的窗口上执行刷新的过程中， *curscr* 中的标记或指定窗口中的标记为 TRUE，那么实现将清除屏幕，然后完全刷新屏幕，并将 *curscr* 中的标记和指定窗口中的标记设置为 FALSE。初始状态是不确定的。

idlok() 函数指定实现是否可以使用所配备的设备的硬件插入行、删除行和滚动功能。如果 *bf* 为 TRUE，则启用这些功能。如果 *bf* 为 FALSE，则禁用这些功能，而是按需要刷新这些行。初始状态为 FALSE。

leaveok() 函数控制刷新操作之后的光标位置。如果 *bf* 为 TRUE，则指定窗口上的刷新操作将终端的光标放置在任意位置。如果 *bf* 为 FALSE，那么在任何刷新操作结束时，终端的光标将定位在指定窗口内保留的光标位置上。初始状态为 FALSE。

scrollok() 函数控制滚动的使用。如果 *bf* 为 TRUE，则为指定的窗口启用滚动，启用结果如 *curses_intro*(3X) 中“截断、包装和滚动”所述。如果 *bf* 为 FALSE，则为指定的窗口禁用滚动。初始状态为 FALSE。

setscrreg() 和 **wsetscrreg()** 函数定义当前窗口或指定窗口内的软件滚动区域。 *top* 和 *bot* 参数是定义滚动区域的第一行和最后一行的行号 (行 0 是窗口的顶行）。如果启用了此选项和 **scrollok()** ，那么在尝试移走边缘上的最后一行时，滚动区域内的所有行都向第一行滚动一行。窗口内只有字符才能滚动。如果设置了软件滚动区域，并且不启用 **scrollok()** ，那么在尝试移走边缘上的最后一行时，不重新定位滚动区域内的任何行。

实际应用信息

启用 **idlok()** 功能的唯一原因是，使用滚动实现局部窗口（例如，屏幕编辑器的局部窗口）运动的可视效果。对于其他情况，这种功能可能会造成视觉干扰。

leaveok() 选项可为不使用光标的应用程序提供更高的效率。

返回值

如果成功完成， **setscrreg()** 和 **wsetscrreg()** 将返回 OK。否则返回 ERR。

其他函数始终返回 OK。

错误

没有定义任何错误。

另请参阅

clear(3X)、 delscreen(3X)、 doupdate(3X)、 scrl(3X)、 <curses.h>。

名称

clock( ) - 报告占用的 CPU 时间

概要

#include <time.h>

clock_t clock(void);

说明

clock() 返回自第一次调用 **clock()** 以来的总 CPU 时间（以微秒计）。所报告的时间是调用进程及其最终的子进程因执行 **wait()** 、 **system()** 或 **pclose()** 而占用的用户时间及系统时间的总和（请参阅 *wait*(2) 、 *system*(3S) 及 *popen*(3S) ）。要确定此时间（以秒计），应利用 **clock()** 的返回值除以宏 **CLOCKS_PER_SEC** 的值。

时钟的精度会有所不同，这取决于硬件和软件配置。

如果无法获得所占用的处理器时间或无法表示它的值，函数就会返回值 (**clock_t**) – 1 。

警告

clock() 的返回值是以微秒为单位定义的，目的是与精度较高的 CPU 时钟实现兼容。因此，在累积仅 4295 秒的 CPU 时间（约 72 分钟）后，返回值就会回绕。

相关内容

缺省的时钟精度为 10 毫秒。

另请参阅

times(2)、wait(2)、system(3S)、 thread_safety(5)。

符合的标准

clock()：AES、SVID2、SVID3、XPG2、XPG3、XPG4、ANSI C

名称

clog( )、clogf( )、clogl( )、clogw( )、clogq( ) - 复变自然对数（底数为 e）函数

概要

#include <complex.h>

double complex clog(double complex z);

float complex clogf(float complex z);

long double complex clogl(long double complex z);

extended complex clogw(extended complex z);

quad complex clogq(quad complex z);

说明

这些函数仅适用于 Integrity 服务器。

clog() 返回 *z* 的复变自然对数值，其范围是沿实轴的数学上无界的带状区域和沿虚轴的 [-iPi,+iPi] 区间内。分支切割位于负实轴。

clogf() 是 **clog()** 的 **float complex** 版本；它采用 **float complex** 参数，并返回 **float complex** 结果。

clogl() 是 **clog()** 的 **long double complex** 版本；它采用 **long double complex** 参数，并返回 **long double complex** 结果。

clogw() 是 **clog()** 的 **extended complex** 版本；它采用 **extended complex** 参数，并返回 **extended complex** 结果。

在 HP-UX 系统中， **clogq()** 等效于 **clogl()** 。

用法

要使用这些函数，请使用缺省的 **-Ae** 选项进行编译。要使用 **clogw()** 或 **clogq()** ，请使用 **-fpwidetypes** 选项进行编译。确保程序包含 **<complex.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

返回值

clog(conj(*z***)) = conj(clog(***z***))** 。

clog(-0+i0) 返回 -Inf+iPi，并引起除数为零的浮点运算异常。

clog(+0+i0) 返回 -Inf+i0，并引起除数为零的浮点运算异常。

对于有限 *x* ， **clog**(*x*+iInf) 返回 +Inf+iPi/2。

对于有限 *x* ， **clog**(*x*+iNaN) 返回 NaN+iNaN，并可能引起无效浮点运算异常。

对于有限正 *y* ， **clog**(-Inf+i*y*) 返回 +Inf+iPi。

对于有限正 *y* ， **clog**(+Inf+i*y*) 返回 +Inf+i0。

clog(-Inf+iInf) 返回 +Inf+i3Pi/4。

clog(+Inf+iInf**)** 返回 +Inf+ipi/4。

clog(±Inf+iNaN**)** 返回 +Inf+iNaN。

对于有限 *y* ， **clog(**NaN+i*y*) 返回 NaN+iNaN，并可能引起无效浮点运算异常。

clog(NaN+iInf**)** 返回 +Inf+iNaN。

clog(NaN+iNaN**)** 返回 NaN+iNaN。

错误

没有定义任何错误。

另请参阅

log(3M)、 cexp(3M)、 complex(5)。

符合的标准

clog() 、 **clogf()** 和 **clogl()** ： ISO/IEC C99（包括附件 G“IEC 60559-compatible complex arithmetic”）

名称

clrtobot 和 wclrtobot — 清除从光标位置到窗口末尾的内容

概要

#include <curses.h>

int clrtobot(void);

int wclrtobot(WINDOW *_win_);

说明

clrtobot() 和 **wclrtobot()** 函数清除当前窗口或指定窗口中光标位置后面的所有行，并清除从光标位置到行尾的当前行（含边界）。

返回值

成功完成后，这些函数返回 OK。否则，将返回 ERR。

错误

没有定义任何错误。

另请参阅

doupdate(3X)、 <curses.h>。

历史变更记录

在 X/Open Curses 第 2 期中首次发布。

X/Open Curses 第 **4** 期

为清楚起见，重新编写了该条目。 **clrtobot()** 函数的参数列表已显式声明为 **void**。

名称

clrtoeol 和 wclrtoeol — 清除从光标位置到行尾的内容

概要

#include <curses.h>

int clrtoeol(void);

int wclrtoeol(WINDOW *_win_);

说明

clrtoeol() 和 **wclrtoeol()** 函数清除当前窗口或指定窗口中从光标位置到行尾的当前行（含边界）。

返回值

成功完成后，这些函数返回 OK。否则，将返回 ERR。

错误

没有定义任何错误。

另请参阅

doupdate(3X)、 <curses.h>。

历史变更记录

在 X/Open Curses 第 2 期中首次发布。

X/Open Curses 第 **4** 期

为清楚起见，重新编写了该条目。 **clrtoeol()** 函数的参数列表已显式声明为 **void**。

名称

cmpt_change()、cmpt_get() - 设置和获取进程的隔离专区

概要

#include <sys/cmpt.h>

int cmpt_change(cmpt_t *cid***);**

cmpt_t cmpt_get(pid_t *pid***);**

参数

cid 隔离专区 ID。

pid 目标进程的 ID，或者调用进程使用的 0 。

说明

cmpt_change() 和 **cmpt_get()** 函数可查询和处理进程的隔离专区。

cmpt_get() 函数返回给定进程的隔离专区 ID。如果将 0 作为 *pid* 参数进行传递，则返回调用进程的隔离专区 ID。

cmpt_change() 函数可将调用进程的隔离专区 ID 更改为 *cid* 的值。

安全性限制

要使 **cmpt_change()** 函数有效，调用进程必须拥有 **CHANGECMPT** 权限。有关权限的详细信息，请参阅 *privileges*(5) 。

返回值

cmpt_get() 可返回下列值:

***n*>0** 成功完成。该函数返回一个有效的隔离专区 ID。
-1 函数失败。设置 **errno** 以指明错误。

cmpt_change() 可返回下列值:

0 成功完成。
-1 函数失败。设置 **errno** 以指明错误。

错误

如果出现下列任意情况，函数将失败，并设置 **errno** 。

[EINVAL] 指定的隔离专区 ID 无效。

[ENOMEM] 函数未能为其操作分配足够的内存。

[EPERM] 进程没有 **CHANGECMPT** 权限。

[ESRCH] *pid* 无效。

举例

```
#include <errno.h>
#include <sys/cmpt.h>


main()
{
    cmpt_t c = cmpt_getbyname("init");
    if (cmpt_change(c) == -1 )
    {
        perror("could not enter init compartment");
        exit(1);
    }

    printf("The process is now running in compartment %d\n",
        cmpt_get(0));
}
```

相关内容

这些函数是 **libsec** 库的一部分。

另请参阅

cmpt_getbynum(3)、 cmpt_getbyname(3)、 compartments(4)、 compartments(5)、 privileges(5)。

名称

cmpt_getbynum()、 cmpt_endent()、 cmpt_getbyname()、 cmpt_getent()、 cmpt_setent() - 将隔离专区名称映射到编号或将编号映射到名称

概要

#include <sys/cmpt.h>

void cmpt_endent(struct cmpt_state *****state*);**

cmpt_t cmpt_getbyname(const char ****cmpt_name*);**

char *cmpt_getbynum(cmpt_t *cmpt_num***);**

int cmpt_setent(struct cmpt_state *****state*);**

const struct cmpt_pair *cmpt_getent(struct cmpt_state *****state*);**

参数

cmpt_name	指向包含隔离专区名称的字符串的指针。
cmpt_num	隔离专区编号。
state	指向内部状态的指针。

结构成员

cmpt_state 结构在 **<sys/cmpt.h>** 中定义。它至少包含以下字段:

```
const char   *cmpt_name;  /* compartment name */
int          cmpt_num;   /* compartment number */
```

说明

隔离专区由 **/etc/cmpt** 下配置文件中的字符串引用，但在内部作以编号形式进行维护。函数 **cmpt_getbyname()** 、 **cmpt_getbynum()** 、 **cmpt_setent()** 、 **cmpt_getent()** 和 **cmpt_setent()** 查询并迭代此数据库。

cmpt_getbyname()	返回与 *cmpt_name* 指定的字符串对应的隔离专区编号。
cmpt_getbynum()	返回与 *cmpt_num* 对应的隔离专区名称。如果返回值不是 NULL，则调用方需要使用 **free()** 释放返回值。请参阅 *free*(3C) 。
cmpt_setent()	打开数据库连接以允许迭代搜索。
cmpt_getent()	返回指向数据库中下一个隔离专区名称/编号对的指针。用户不能调用 **free()** 以释放返回值指向的内存。
cmpt_endent()	关闭数据库连接。

注释

对 **cmpt_setent()** 的每个调用必须与对 **cmpt_endent()** 的对应调用匹配。

返回值

cmpt_getbyname() 可返回下列值：

n>**0** 成功完成。函数返回有效的隔离专区编号。
-1 函数失败。设置 **errno** 以指明错误。

cmpt_getbynum() 可返回下列值：

指针 成功完成。返回指向包含隔离专区名称的字符串的非空指针。
空指针 函数失败。返回空指针，并设置 **errno** 以指明错误。

需要由调用方来释放结果。

cmpt_setent() 可返回下列值：

0 成功完成。
-1 函数失败。设置 **errno** 以指明错误。

cmpt_getent() 可返回下列值：

指针 成功完成。返回指向 **cmpt_pair** 结构的非空指针。
空指针 函数失败或已到达最后一个条目。如果出现错误，则设置 **errno** 以指明错误。

当已到达最后一个条目时， **cmpt_getent()** 将 **errno** 设置为 **0** 并返回空指针。

错误

如果出现下列任意情况，函数将失败，并设置 **errno** 。

[ENOENT] 隔离专区名称或编号不存在。

[ENOSYS] 此系统上未启用隔离专区。

[ENOMEM] 函数未能为其操作分配足够的内存。

举例

示例 **1**

获取与名称关联的隔离专区编号。

```
#include <sys/cmpt.h>

void main(int argc, char **argv) {
   cmpt_t cmptid;

   cmptid = cmpt_getbyname("INIT");
   if (cmptid != -1) {
      printf("\nCompartment id : %d\n", cmptid);
   } else {
      printf("\nError getting compartment number\n");
```

```
    }
  }
```

示例 **2**

获取与编号关联的隔离专区名称。

```
#include <sys/cmpt.h>

void main(int argc, char **argv) {
   char *name;

   name = cmpt_getbynum((cmpt_t)2);
   if (name) {
      printf("\nCompartment name : %s\n", name);
      free(name);
   } else {
      printf("\nError getting compartment name\n");
   }
}
```

示例 **3**

对隔离专区映射条目进行迭代。

```
#include <errno.h>
#include <sys/cmpt.h>

void main(int argc, char **argv) {
   struct cmpt_pair *pair;
   struct cmpt_state *state

   if (cmpt_setent(&state) != 0) {
     perror("cmpt_setent failed");
     exit(1);
   }

   /* get entries until a NULL returned */
   while ( pair = cmpt_getent(&state) )
   {
      printf("\nCompartment Nnumber : %d  Name : %s\n",
            pair->cmpt_num, pair->cmpt_name);
   }
```

```
    if (errno == 0)
    {
       printf("\nAll Entries printed\n");
    } else {
       printf("\nError getting compartment entry\n");
    }

    cmpt_endent();
}
```

相关内容

这些函数是 **libsec** 库的一部分，可以使用 **-lsec** 选项将该库与可执行文件关联。

另请参阅

compartments(4)、 compartments(5)。

名称

cmpt_get_ifcid()、cmpt_get_addrcid() - 获取与网络接口关联的隔离专区 ID

概要

#include <sys/cmpt.h>

cmpt_t cmpt_get_ifcid(const char ****ifname*);**

cmpt_t cmpt_get_addrcid(int *family***, void ****addr***, uint32_t** *mask***);**

参数

ifname 网络接口名称字符串的指针。

family 指定的 IP 地址的地址系列。当前支持 AF_INET 和 AF_INET6 地址系列。

addr 如果 family 参数为 AF_INET，addr 指向保存 IPv4 地址的缓冲区；如果 family 参数为 AF_INET6，则指向保存 IPv6 地址的缓冲区。地址在网络字节顺序中应当以数字格式表示。数字格式是将要加入套接字地址结构的二进制值。

二进制值是 IPv4/IPv6 地址的十六进制表示形式。对于 IPv4 地址，此二进制值驻留在 **in_addr** 结构中，而对于 IPv6 地址，则驻留在 **in6_addr** 结构中（有关详细信息，请参阅 *inet*(3N)）。

mask 以 CIDR 形式表示的网络掩码。

说明

cmpt_get_ifcid() 函数返回与网络接口关联的隔离专区 ID。

cmpt_get_addrcid() 函数返回 **setrules** 命令以前调用所设置的 (*addr*,*mask*,*family*) 元组的隔离专区 ID。

返回值

函数 **cmpt_get_ifcid()** 和 **cmpt_get_addrcid()** 返回下列值:

***n*>0** 成功完成。 *n* 是接口或 (*addr*,*mask*,*family*) 元组的隔离专区。

-1 函数失败。可以设置 **errno** 以指明错误。

错误

如果出现下列任意情况， **cmpt_get_ifcid()** 和 **cmpt_get_addrcid()** 将失败，并设置 **errno** 。

[ENOSYS] 不支持设置隔离专区功能。

[EPERM] 此进程没有相应的权限。请参阅 *privileges*(5) 。

[ENOENT] 指定的接口或地址元组没有隔离专区映射。

[EINVAL] 指定的 mask 或 family 无效。

相关内容

这些函数是 **libsec** 库的一部分。

另请参阅

compartments(4)、 compartments(5)、 privileges(5)。

名称

cmpt_get_peer_cid()、 cmpt_get_endpoint_cid() - 获取 INET 域通信的隔离专区 ID。

概要

#include <sys/ioctl.h>

#include <sys/cmpt.h>

cmpt_t cmpt_get_endpoint_cid(int *s*);

cmpt_t cmpt_get_peer_cid(int *s*);

参数

s INET 域套接字或流。

说明

cmpt_get_endpoint_cid() 和 **cmpt_get_peer_cid()** 函数返回与 INET 域传输端点（套接字或流）关联的隔离专区。

cmpt_get_endpoint_cid() 函数返回与传输端点 *s* 关联的隔离专区 ID (cid)。传输端点的 cid 在创建时已设置为进程的 cid，即使在进程之间进行传输，或者创建进程更改了该传输端点的隔离专区，该 cid 也不会发生更改。

cmpt_get_peer_cid() 函数返回与传输端点 *s* 通信的对等端的隔离专区 ID。对于环回通信，该 cid 与对等端传输端点关联。对于通过网络接口进行的通信，对等端 cid 与接收通信的网络接口关联。对于 TCP 通信，建立连接后，对等端 cid 将持续存在。对于其他协议，对等端 cid 可能针对收到的每个数据包发生更改。

如果未建立连接（对于 TCP），或者未收到任何通信（对于其他所有协议），函数将返回错误。

返回值

cmpt_get_endpoint_cid() 和 **cmpt_get_peer_cid()** 可返回下列值:

***n*>0** 成功完成。该函数返回一个有效的隔离专区 ID。

-1 函数失败。将设置 **errno** 以指示来自基础 **ioctl()** 系统调用的错误。

错误

有关 **errno** 指示的错误，请参阅 *ioctl*(2) 。

相关内容

这些函数是 **libsec** 库的一部分。

另请参阅

compartments(4)、 compartments(5)。

名称

COLS — 终端屏幕中的列数

概要

#include <curses.h>

extern int COLS;

说明

外部变量 *COLS* 指明终端屏幕中的列数。

另请参阅

initscr(3X)、<curses.h>。

历史变更记录

在 X/Open Curses 第 4 期中首次发布。

名称

compound( )、compoundf( )、compoundl( )、compoundw( )、compoundq( ) - 复利因子

概要

#include <math.h>

double compound(double rate, double periods);

float compoundf(float rate, float periods);

long double compoundl(long double rate, long double periods);

extended compoundw(extended rate, extended periods);

quad compoundq(quad rate, quad periods);

说明

compound() 函数将计算复利因子:

(1 + *rate*) ** *periods*

compoundf() 是 **float** 版的 **compound()** ; 它采用 **float** 参数，并返回 **float** 结果。

compoundl() 是 **long double** 版的 **compound()** ; 它采用 **long double** 参数，并返回 **long double** 结果。

compoundw() 是 **extended** 版的 **compound()** ; 它采用 **extended** 参数，并返回 **extended** 结果。

compoundq() 等效于 HP-UX 系统上的 **compoundl()** 。

用法

要使用这些函数，请使用缺省的 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。

要使用 **compoundw()** 或 **compoundq()** ，也可以使用 **-fpwidetypes** 选项进行编译。确保程序包含 **<math.h>** 。

通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

返回值

如果 *periods* 等于零， **compound()** 将返回 1.0。

如果参数为 NaN， **compound()** 就会返回 NaN。

如果 *rate* 小于 -1， **compound()** 将返回 NaN 并引发无效异常。

如果 *rate* 等于 -1， **compound()** 将等效于 **pow**(0, *periods*) 。

如果 *rate* 等于正无穷大， **compound()** 将等效于 **pow**(*rate*, *periods*) 。

如果 *periods* 为无限大且 *rate* 等于零， **compound()** 将返回 1.0。

如果 *periods* 为无限大且 *rate* 不等于零， **compound()** 将等效于 **exp**(*rate* * *periods*) 。

compound() 将返回带相应符号的无穷大（代替量值太大的值），并引发溢出及不精确异常。

当结果太小（本质上异常或为零），因此会丧失准确性时， **compound()** 将引发下溢和不精确异常；如果结果只是太小，即可能引发这些异常。

错误

没有定义任何错误。

另请参阅

annuity(3M)、 exp(3M)、 expm1(3M)、 pow(3M)、 math(5)。

符合的标准

任何标准均未指定这些函数。

名称

confstr( ) - 获取值为字符串的配置值

概要

#include <unistd.h>

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

说明

confstr() 为应用程序提供获取配置定义的字符串值的方法。其用途和目的类似于 **sysconf()** （请参阅 *sysconf*(2)），不同之处在于，它是在返回了字符串值，而不是数值的条件下使用的。

name 参数包括在 **<unistd.h>** 中定义的以下 *name* 值。

_CS_PATH **PATH** 环境变量的缺省值，可用来查找《HP-UX 参考手册》第一节所述的命令，及 POSIX.2 标准中定义的实用程序，当前 HP-UX 操作系统中实现了这些实用程序。

_CS_HW_CPU_SUPP_BITS

硬件上支持哪种内核。当前的返回值包括“32”、“32/64”或“64”。

_CS_KERNEL_BITS

内核是 32 位还是 64 位。当前的返回值包括“32”或“64”。

_CS_MACHINE_MODEL

硬件型号字符串。

_CS_MACHINE_IDENT

每台物理计算机的标识符。作为可输出 ASCII 字符的不透明字符串返回。对于物理计算机的所有分区，此字符串的值相同。对于随 HP-UX 11i 或更高版本首次发行的硬件类，此 ID 在所有硬件类中是唯一的。对于早期的硬件类，此 ID 号码只有在本身的硬件类中才是唯一的。如果没有可用的 ID 号码，将返回空字符串；只有原型机，或者制造时没有正确配置的其他系统，才应该会发生这种情况。必须使用字符串比较函数来比较此值，请参阅 *string*(3C) 。

_CS_PARTITION_IDENT

计算机上每个分区的标识符。作为可输出 ASCII 字符的不透明字符串返回。对于不支持分区的任何计算机，此值与 **_CS_MACHINE_IDENT** 相同。必须使用字符串比较函数来比较此值，请参阅 *string*(3C) 。

_CS_MACHINE_SERIAL

计算机序列号可以在外部机箱的标签上找到。此值应该是可输出的 ASCII 字符串。此字符串并非在所有计算机类上可用；如果不可用，此字符串为空。由于不同类的计算机可能使用相同的序列号，因此该字符串不是计算机的唯一标识符。

如果需要唯一标识符，请使用 **_CS_MACHINE_IDENT** 或 **_CS_PARTITION_IDENT**。

_CS_XBS5_ILP32_OFF32_CFLAGS

初始选项集将指定给 **cc** 和 **c89** 实用程序，以使用带有 32 位 int、 long、pointer 和 *off_t* 类型的编程模型来构建应用程序。

_CS_XBS5_ILP32_OFF32_LDFLAGS

最终选项集将指定给 **cc** 和 **c89** 实用程序，以使用带有 32 位 int、 long、pointer 和 *off_t* 类型的编程模型来构建应用程序。

_CS_XBS5_ILP32_OFF32_LIBS

库集将指定给 **cc** 和 **c89** 实用程序，以使用带有 32 位 int、 long、pointer 和 *off_t* 类型的编程模型来构建应用程序。

_CS_XBS5_ILP32_OFF32_LINTFLAGS

选项集将指定给 **lint** 实用程序，以使用带有 32 位 int、 long、pointer 和 *off_t* 类型的编程模型来检查应用程序源。

_CS_XBS5_ILP32_OFFBIG_CFLAGS

初始选项集将指定给 **cc** 和 **c89** 实用程序，以使用带有 32 位 int、 long、pointer 类型，及至少使用 64 位的 *off_t* 类型的编程模型来构建应用程序。

_CS_XBS5_ILP32_OFFBIG_LDFLAGS

最终选项集将指定给 **cc** 和 **c89** 实用程序，以使用带有 32 位 int、 long 和 pointer 类型，及至少使用 64 位的 *off_t* 类型的编程模型来构建应用程序。

_CS_XBS5_ILP32_OFFBIG_LIBS

库集将指定给 **cc** 和 **c89** 实用程序，以使用带有 32 位 int、 long、pointer 类型，及至少使用 64 位的 *off_t* 类型的编程模型来构建应用程序。

_CS_XBS5_ILP32_OFFBIG_LINTFLAGS

选项集将指定给 **lint** 实用程序，以使用带有 32 位 int、 long 和 pointer 类型，及至少使用 64 位的 *off_t* 类型的编程模型来检查应用程序源。

_CS_XBS5_LP64_OFF64_CFLAGS

初始选项集将指定给 **cc** 和 **c89** 实用程序，以使用带有 32 位 int、 64 位 long、pointer 和 *off_t* 类型的编程模型来构建应用程序。

_CS_XBS5_LP64_OFF64_LDFLAGS

最终选项集将指定给 **cc** 和 **c89**
实用程序，以使用带有 32 位 int、 64 位 long、pointer 和 *off_t* 类型的编程模型来构建应用程序。

_CS_XBS5_LP64_OFF64_LIBS

库集将指定给 **cc** 和 **c89** 实用程序，以使用带有 32 位 int、 64 位 long、pointer 和 *off_t* 类型的编程模型来构建应用程序。

_CS_XBS5_LP64_OFF64_LINTFLAGS

选项集将指定给 **lint** 实用程序，以使用带有 32 位 int、 64 位 long、pointer 和 *off_t* 类型的编程模型来检查应用程序源。

_CS_XBS5_LPBIG_OFFBIG_CFLAGS

初始选项集将指定给 **cc** 和 **c89** 实用程序，以通过带有使用 32 位的 int 类型，及至少使用 64 位的 long、pointer 和 *off_t* 类型的编程模型来构建应用程序。

_CS_XBS5_LPBIG_OFFBIG_LDFLAGS

库集将指定给 **cc** 和 **c89** 实用程序，以通过带有使用 32 位 的 int 类型，及至少使用 64 位的 long、pointer 和 *off_t* 类型的编程模型来构建应用程序。

_CS_XBS5_LPBIG_OFFBIG_LIBS

库集将指定给 **cc** 和 **c89** 实用程序，以通过带有使用 32 位 的 int 类型，及至少使用 64 位的 long、pointer 和 *off_t* 类型的编程模型来构建应用程序。

_CS_XBS5_LPBIG_OFFBIG_LINTFLAGS

选项集将指定给 **lint** 实用程序，以通过带有使用 32 位的 int 类型，及至少使用 64 位的 long、pointer 和 *off_t* 类型的编程模型来检查应用程序源。

_CS_POSIX_V6_ILP32_OFF32_CFLAGS

初始选项集将指定给 **c99** 实用程序，以使用带有 32 位 int、long、pointer 和 *off_t* 类型的编程模型来构建应用程序。

_CS_POSIX_V6_ILP32_OFF32_LDFLAGS

最终选项集将指定给 **c99** 实用程序，以使用带有 32 位 int、long、pointer 和 *off_t* 类型的编程模型来构建应用程序。

_CS_POSIX_V6_ILP32_OFF32_LIBS

库集将指定给 **c99** 实用程序，以使用带有 32 位 int、long、pointer 和 *off_t* 类型的编程模型来构建应用程序。

_CS_POSIX_V6_ILP32_OFFBIG_CFLAGS

初始选项集将指定给 **c99** 实用程序，以使用带有 32 位 int、long、pointer 类型，及至少使用 64 位的 *off_t* 类型的编程模型来构建应用程序。

_CS_POSIX_V6_ILP32_OFFBIG_LDFLAGS

最终选项集将指定给 **c99** 实用程序，以使用带有 32 位 int、long、pointer 类型，及至少使用 64 位的 *off_t* 类型的编程模型来检查应用程序源。

_CS_POSIX_V6_ILP32_OFFBIG_LIBS

库集将指定给 **c99** 实用程序，以使用带有 32 位 int、long、pointer 类型，及至少使用 64 位的 *off_t* 类型的编程模型来构建应用程序。

_CS_POSIX_V6_LP64_OFF64_CFLAGS

初始选项集将指定给 **c99** 实用程序，以使用带有 32 位 int 和 64 位 long、pointer 和 *off_t* 类型的编程模型来构建应用程序。

_CS_POSIX_V6_LP64_OFF64_LDFLAGS

最终选项集将指定给 **c99** 实用程序，以使用带有 32 位 int 和 64 位 long、pointer 和 *off_t* 类型的编程模型来构建应用程序。

_CS_POSIX_V6_LP64_OFF64_LIBS

库集将指定给 **c99** 实用程序，以使用至少带有 32 位 int 和 64 位 long、pointer，及 *off_t* 类型的编程模型来构建应用程序。

_CS_POSIX_V6_LPBIG_OFFBIG_CFLAGS

初始选项集将指定给 **c99** 实用程序，以通过带有至少使用 32 位的 long、pointer，及至少使用 64 位的 *off_t* 类型的编程模型来构建应用程序。

_CS_POSIX_V6_LPBIG_OFFBIG_LDFLAGS

最终选项集将指定给 **c99** 实用程序，以通过带有至少使用 32 位的 int 类型和至少使用 64 位的 long、pointer，及 *off_t* 类型的编程模型来构建应用程序。

_CS_POSIX_V6_LPBIG_OFFBIG_LIBS

将指定给 **c99** 实用程序的库集，以通过带有至少使用 32 位的 int 类型，及至少使用 64 位的 long、pointer 和 *off_t* 类型的编程模型来构建应用程序。

_CS_POSIX_V6_WIDTH_RESTRICTED_ENVS

该值是一个实现支持的、以 <换行符> 分隔的编程环境名称列表，其中 blksize_t、cc_t、mode_t、nfds_t、pid_t、ptrdiff_t、size_t、speed_t、ssize_t、suseconds_t、tcflag_t、useconds_t、wchar_t 和 wint_t 类型的宽度不能超过 long 类型的宽度。

如果 *len* 不是零， *name* 已知且包含配置定义的值，则 **confstr()** 会将该值复制到 *buf* 指向的 *len* 字节缓冲区。如果要返回的字符串的长度包括结尾空字符在内大于 *len* 字节，则 **confstr()** 将字符串截为 *len* - 1 字节，并使用 null 作为结果的结尾。应用程序可以通过将 **confstr()** 返回的值与 *len* 进行比较，检测字符串是否被截断。

如果 *len* 为零且 *buf* 为 NULL， **confstr()** 将返回定义如下的整数值，但是不返回字符串。如果 *len* 为零，但是 *buf* 不是 NULL，结果将是不确定的。

返回值

如果 *name* 无效，则 **confstr()** 返回零，且将 **errno** 设置为 [EINVAL] 。

如果 *name* 不包含配置定义的值，则 **confstr()** 返回 0（零），且不更改 **errno** 。

如果 *name* 包含配置定义的值，则 **confstr()** 返回保存整个配置定义的值所需的缓冲区大小。如果此返回值小于 *len* ，则表示已截断 *buf* 中返回的字符串。

举例

以下代码段将调用 **confstr()** 来确定 **_CS_PATH** 的正确缓冲区大小，为此缓冲区分配空间，然后获取 **_CS_PATH** 的配置值。

```
#include <unistd.h>
#include <stddef.h>

size_t bufsize;
char *buffer;

bufsize=confstr(_CS_PATH,NULL,(size_t)0);
buffer=(char *)malloc(bufsize);
confstr(_CS_PATH,buffer,bufsize);
```

作者

confstr() 由 HP 开发。

文件

/usr/include/unistd.h

用于支持 /usr/group 标准的符号常量和结构

另请参阅

getconf(1)、 errno(2)、 fpathconf(2)、 pathconf(2)、 sysconf(2)、 malloc(3C)、 thread_safety(5)。

符合的标准

confstr()：XPG4、POSIX.2

名称

conj( )、conjf( )、conjl( )、conjw( )、conjq( ) - 复变共轭函数

概要

#include <complex.h>

double complex conj(double complex z);

float complex conjf(float complex z);

long double complex conjl(long double complex z);

extended complex conjw(extended complex z);

quad complex conjq(quad complex z);

说明

这些函数仅适用于 Integrity 服务器。

conj() 返回 *z* 的共轭复数，它是通过改变虚数部分的符号计算的。

conjf() 是 **conj()** 的 **float complex** 版本；它采用 **float complex** 参数，并返回 **float complex** 结果。

conjl() 是 **conj()** 的 **long double complex** 版本；它采用 **long double complex** 参数，并返回 **long double complex** 结果。

conjw() 是 **conj()** 的 **extended complex** 版本；它采用 **extended complex** 参数，并返回 **extended complex** 结果。

在 HP-UX 系统中， **conjq()** 等效于 **conjl()** 。

这些函数不会引发异常。

用法

要使用这些函数，请使用缺省的 **-Ae** 选项进行编译。要使用 **conjw()** 或 **conjq()** ，请使用 **-fpwidetypes** 选项进行编译。确保程序包含 **<complex.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

错误

没有定义任何错误。

另请参阅

creal(3M)、 cimag(3M)、 complex(5)。

符合的标准

conj() 、 **conjf()** 和 **conjl()** ： ISO/IEC C99（包括附件 G“IEC 60559-compatible complex arithmetic”）

名称

toupper( )、tolower( )、_toupper( )、_tolower( )、toascii( ) - 转换字符

概要

#include <ctype.h>

int toupper(int c);

int tolower(int c);

int _toupper(int c);

int _tolower(int c);

int toascii(int c);

说明

toupper() 与 **tolower()** 具有和 *getc*(3S) 相同的域范围：从 –1 到 255 的整数。如果 **toupper()** 的参数表示一个小写字母，那么结果为对应的大写字母。如果 **tolower()** 的参数表示一个大写字母，那么结果为对应的小写字母。该域中所有其他参数返回时都不变。该域之外的参数会引起未定义的结果。

宏 **_toupper()** 和 **_tolower()** 分别与 **toupper()** 和 **tolower()** 相同。

toascii() 生成其参数，使不属于标准 7 位 ASCII 字符的所有位被清除，以便与其他系统兼容。

警告

toascii() 既作为库函数提供，也作为 **<ctype.h>** 头文件中定义的宏提供。通常使用宏版本。要获取库函数，可以使用 **#undef** 删除宏定义，或者，如果是以 ANSI C 模式编译的，请用圆括号将函数名括起来或采用其地址。下面的示例使用 **toascii()** 的库函数：

```
#include <ctype.h>
#undef toascii
    ...
main()
{
    ...
  c1 = toascii(c);
    ...
}
```

或者

```
#include <ctype.h>
    ...
main()
{
  int (*conv_func)();
```

```
    ...
  c1 = (toascii)(c);
    ...
  conv_func = toascii;
    ...
}
```

下面的示例使用 **toupper()** 的库函数：

```
#include <ctype.h>
#undef toupper
    ...
main()
{
    ...
char *c;
*c=toupper ((unsigned char*) c);
    ...
}
```

外部语言环境影响

语言环境

LC_CTYPE 类别确定要进行的转换。

国际代码集支持

支持单字节字符代码集。

作者

conv() 由 IBM、OSF 和 HP 联合开发。

另请参阅

ctype(3C)、getc(3S)、setlocale(3C)、lang(5)、thread_safety(5)。

符合的标准

_tolower()：AES、SVID2、SVID3、XPG2、XPG3 和 XPG4

_toupper()：AES、SVID2、SVID3、XPG2、XPG3 和 XPG4

toascii()：AES、SVID2、SVID3、XPG2、XPG3 和 XPG4

tolower()：AES、SVID2、SVID3、XPG2、XPG3、XPG4、FIPS 151-2、POSIX.1 和 ANSI C

toupper()：AES、SVID2、SVID3、XPG2、XPG3、XPG4、FIPS 151-2、POSIX.1 和 ANSI C

名称

copylist() - 将文件复制到内存

概要

#include <libgen.h>

char *copylist(const char * *filenm*, **off_t *** *szptr*);

说明

copylist 将文件中的一组项目复制到新分配的内存中，并使用空字符替换换行符。它需要两个参数：一个指向待复制文件的名称的指针 *filenm* ，一个指向将存储文件大小的变量的指针 *szptr* 。

成功完成后， **copylist** 返回一个指向已分配内存的指针。如果在查找文件、调用 **malloc** 或打开文件时遇到问题，则返回 NULL 。

要使用此接口，请通过指定 **-lgen** 来链接到 libgen 库。例如：

```
cc foo.c -lgen
```

举例

```
/* read "file" into buf */
off_t size;
char *buf;
buf = copylist("file", &size) ;"
for (i = 0; i < size; i++)
  if(buf[i])
      putchar(buf[i]);
  else
      putchar('\n');
```

另请参阅

malloc(3C)、 thread_safety(5)。

名称

copysign( )、copysignf( )、copysignl( )、copysignw( )、copysignq( ) - copysign 函数

概要

#include <math.h>

double copysign(double x, double y);

float copysignf(float x, float y);

仅适用于 **HP Integrity** 服务器

long double copysignl(long double x, long double y);

extended copysignw(extended x, extended y);

quad copysignq(quad x, quad y);

说明

copysign() 函数返回 *x* ，并且其符号已更改为 *y* 的符号。

copysignf() 是 **copysign()** 的 **float** 版本；它采用 **float** 参数，并返回 **float** 结果。

仅适用于 **Integrity** 服务器

copysignl() 是 **copysign()** 的 **long double** 版本；它采用 **long double** 参数，并返回 **long double** 结果。

copysignw() 是 **copysign()** 的 **extended** 版本；它采用 **extended** 参数，并返回 **extended** 结果。

在 HP-UX 系统中， **copysignq()** 等效于 **copysignl()** 。

用法

要使用这些函数，请使用缺省的 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。（对于 Integrity 服务器）要使用 **copysignw()** 或 **copysignq()** ，请使用 **-fpwidetypes** 选项进行编译。确保程序包含 **<math.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

返回值

copysign() 函数返回具有 *x* 的值和 *y* 的符号的值。

这些函数不会引发异常。

错误

没有定义任何错误。

另请参阅

fabs(3M)、 fpclassify(3M)、 remainder(3M)、 scalbn(3M)、 signbit(3M)、 math(5)。

符合的标准

copysign() 、 **copysignf()** 和 **copysignl()** ： ISO/IEC C99（包括附件 F“IEC 60559 floating-point arithmetic”）

名称

copywin — 复制窗口的区域

概要

#include <curses.h>

int copywin(const WINDOW ****srcwin*, WINDOW ****dstwin***, int** *sminrow***,**
int *smincol***, int** *dminrow***, int** *dmincol***, int** *dmaxrow***,**
int *dmaxcol***, int** *overlay***);**

说明

copywin() 函数比 **overlay()** 和 **overwrite()** 函数提供更细粒度的控制。与在 **prefresh()** 函数中一样，在目标窗口中指定矩形 (*dminrow*, *dmincol*) 和 (*dmaxrow*, *dmaxcol*)，以及源窗口的左上角坐标 (*sminrow*, *smincol*)。如果 *overlay* 为 TRUE，则与在 **overlay()** 中一样，复制没有破坏性。如果 *overlay* 为 FALSE，则与在 **overwrite()** 中一样，复制具有破坏性。

返回值

成功完成后， **copywin()** 返回 OK。否则，返回 ERR。

错误

没有定义任何错误。

另请参阅

newpad(3X)、 overlay(3X)、 <curses.h>。

历史变更记录

在 X/Open Curses 第 4 期中首次发布。

名称

cos( )、cosf( )、cosl( )、cosw( )、cosq( ) - 余弦函数

概要

#include <math.h>

double cos(double x);

float cosf(float x);

仅适用于 **HP Integrity** 服务器

long double cosl(long double x);

extended cosw(extended x);

quad cosq(quad x);

说明

cos() 返回 *x* （以弧度形式指定的 x ）的余弦值。

在 PA-RISC 系统中，当 *x* 和零差很多时， **cos()** 可能变得不准确。

cosf() 是 **cos()** 的 **float** 版本；它采用 **float** 参数，并返回 **float** 结果。

仅适用于 **Integrity** 服务器

cosl() 是 **cos()** 的 **long double** 版本；它采用 **long double** 参数，并返回 **long double** 结果。

cosw() 是 **cos()** 的 **extended** 版本；它采用 **extended** 参数，并返回 **extended** 结果。

在 HP-UX 系统中， **cosq()** 等效于 **cosl()** 。

用法

要使用这些函数，请使用缺省的 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。

（对于 Integrity 服务器）要使用 **cosw()** 或 **cosq()** ，也可以使用 **-fpwidetypes** 选项进行编译。

要使用上面的任何函数，请确保程序包含 **<math.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

有关详细信息，请参阅《HP-UX floating-point guide for HP Integrity servers》（位于以下站点：**http://www.hp.com/go/fp** ）。

仅适用于 **PA-RISC** 系统

可以使用 Millicode 版的 **cos()** 和 **cosf()** 函数。 Millicode 版的数学库函数通常比标准库中对应的函数运行速度快。要使用这些版本，请使用 **+Olibcalls** 或 **+Oaggressive** 优化选项编译程序。

特殊情况下，Millicode 版本的返回值与标准库中对应函数返回的值相同（请参阅“返回值”一节）。

返回值

cos(±0**)** 返回 1。

如果 *x* 为 ±INFINITY，则 **cos()** 返回 NaN，并引起无效异常。

如果 *x* 为 NaN，则 **cos()** 返回 NaN。

当未引起其他异常时， **cos()** 是否会引起不精确的异常是不确定的。

错误

没有定义任何错误。

另请参阅

acos(3M)、 asin(3M)、 atan(3M)、 atan2(3M)、 ccos(3M)、 cosd(3M)、 sin(3M)、 sincos(3M)、 tan(3M)、 math(5)。

符合的标准

cos()： SVID3、XPG4.2、ANSI C 和 ISO/IEC C99 （包括附件 F“IEC 60559 floating-point arithmetic”）

cosf() 和 **cosl()**： ISO/IEC C99（包括附件 F“IEC 60559 floating-point arithmetic”）

名称

cosd( )、cosdf( )、cosdl( )、cosdw( )、cosdq( ) - 以度为单位所指定的参数的余弦函数

概要

#include <math.h>

double cosd(double x);

float cosdf(float x);

仅适用于 **HP Integrity** 服务器

long double cosdl(long double x);

extended cosdw(extended x);

quad cosdq(quad x);

说明

cosd() 返回 *x* （ x 以度为单位指定）的余弦值。

在 PA-RISC 系统上，如果 *x* 距离原点较远，则 **cosd()** 可能会丧失准确性。

cosdf() 是 **float** 形式的 **cosd()** ； 它采用 **float** 参数，并返回 **float** 结果。

仅适用于 **Integrity** 服务器

cosdl() 是 **long double** 形式的 **cosd()** ； 它采用 **long double** 参数，并返回 **long double** 结果。

cosdw() 是 **extended** 形式的 **cosd()** ； 它采用 **extended** 参数，并返回 **extended** 结果。

cosdq() 等效于 HP-UX 系统上的 **cosdl()** 。

用法

要使用这些函数，请使用缺省的 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。

（对于 Integrity 服务器）要使用 **cosdw()** 或 **cosdq()** ，也可以使用 **-fpwidetypes** 选项进行编译。

要使用上面的任何函数，请确保程序包含 **<math.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

有关详细信息，请参阅《HP-UX floating-point guide for HP Integrity servers》（位于以下站点：**http://www.hp.com/go/fp** ）。

返回值

cosd(±0) 将返回 1。

如果 *x* 为 ±INFINITY，则 **cosd()** 将返回 NaN 并引发无效异常。

如果 *x* 为 NaN， **cosd()** 将返回 NaN。

错误

没有定义任何错误。

另请参阅

acosd(3M)、 asind(3M)、 atand(3M)、 atan2d(3M)、 cos(3M)、 sincosd(3M)、 sind(3M)、 tand(3M)、 math(5)。

符合的标准

任何标准均未指定这些函数。

名称

cosh( )、coshf( )、coshl( )、coshw( )、coshq( ) - 双曲余弦函数

概要

#include <math.h>

double cosh(double x);

float coshf(float x);

仅适用于 **HP Integrity** 服务器

long double coshl(long double x);

extended coshw(extended x);

quad coshq(quad x);

说明

cosh() 返回 *x* 的双曲余弦值。

coshf() 是 **float** 形式的 **cosh()** ；它采用 **float** 参数，并返回 **float** 结果。

仅适用于 **Integrity** 服务器

coshl() 是 **long double** 形式的 **cosh()** ；它采用 **long double** 参数，并返回 **long double** 结果。

coshw() 是 **extended** 形式的 **cosh()** ；它采用 **extended** 参数，并返回 **extended** 结果。

coshq() 等效于 HP-UX 系统上的 **coshl()** 。

用法

要使用这些函数，请使用缺省的 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。

（对于 Integrity 服务器）要使用 **coshw()** 或 **coshq()** ，也可以使用 **-fpwidetypes** 选项进行编译。

要使用上面的任何函数，请确保程序包含 **<math.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

有关详细信息，请参阅《HP-UX floating-point guide for HP Integrity servers》（位于以下站点：**http://www.hp.com/go/fp** ）。

返回值

cosh(±0) 将返回 1。

如果 *x* 为 ±INFINITY，则 **cosh()** 将返回 +INFINITY。

如果 *x* 为 NaN， **cosh()** 将返回 NaN。

cosh() 将返回无穷大（等于 **HUGE_VAL** ），以代替量值太大的值，并引发溢出及不精确异常。

如果未引发其他异常，则说明未指定 **cosh()** 是否引发不精确异常的情况。

错误

如果正确的值会溢出，则 **cosh()** 会将 **errno** 设置为 [ERANGE]。

仅适用于 **Integrity** 服务器

缺省情况下，Integrity 服务器上的 HP-UX **libm** 函数不设置 **errno** 。对于 **errno** 设置，可以使用 **+Olibmerrno** 选项进行编译。

另请参阅

acosh(3M)、 ccosh(3M)、 sinh(3M)、 tanh(3M)、 math(5)。

符合的标准

cosh() ： SVID3、XPG4.2、ANSI C、ISO/IEC C99 （包括附件 F“IEC 60559 floating-point arithmetic”）

coshf() 、 **coshl()** ： ISO/IEC C99（包括附件 F“IEC 60559 floating-point arithmetic”）

名称

cot( )、cotf( )、cotl( )、cotw( )、cotq( ) - 余切函数

概要

#include <math.h>

double cot(double x);

float cotf(float x);

long double cotl(long double x);

extended cotw(extended x);

quad cotq(quad x);

说明

这些函数仅适用于 Integrity 服务器。

cot() 返回 *x* （以弧度形式指定的 x ）的余切值。

cotf() 是 **cot()** 的 **float** 版本；它采用 **float** 参数，并返回 **float** 结果。

cotl() 是 **cot()** 的 **long double** 版本；它采用 **long double** 参数，并返回 **long double** 结果。

cotw() 是 **cot()** 的 **extended** 版本；它采用 **extended** 参数，并返回 **extended** 结果。

在 HP-UX 系统中， **cotq()** 等效于 **cotl()** 。

用法

要使用这些函数，请使用缺省的 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。

要使用 **cotw()** 或 **cotq()** ，也可以使用 **-fpwidetypes** 选项进行编译。

要使用上面的任何函数，请确保程序包含 **<math.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

返回值

cot(±0) 返回 ±INFINITY。

如果 *x* 为 ±INFINITY，则 **cot()** 返回 NaN，并引起无效异常。

如果 *x* 为 NaN，则 **cot()** 返回 NaN。

当未引起其他异常时， **cot()** 是否会引起不精确的异常是不确定的。

错误

没有定义任何错误。

另请参阅

atan(3M)、 cos(3M)、 ctan(3M)、 sin(3M)、 tan(3M)、 math(5)。

符合的标准

任何标准均未指定这些函数。

名称

cotd( )、cotdf( )、cotdl( )、cotdw( )、cotdq( ) - 以度数形式指定的参数的余切函数

概要

#include <math.h>

double cotd(double x);

float cotdf(float x);

long double cotdl(long double x);

extended cotdw(extended x);

quad cotdq(quad x);

说明

这些函数仅适用于 Integrity 服务器。

cotd() 返回 *x* （以度数形式指定的 x ）的余切值。

cotdf() 是 **cotd()** 的 **float** 版本；它采用 **float** 参数，并返回 **float** 结果。

cotdl() 是 **cotd()** 的 **long double** 版本；它采用 **long double** 参数，并返回 **long double** 结果。

cotdw() 是 **cotd()** 的 **extended** 版本；它采用 **extended** 参数，并返回 **extended** 结果。

在 HP-UX 系统中， **cotdq()** 等效于 **cotdl()** 。

用法

要使用这些函数，请使用缺省的 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。

要使用 **cotdw()** 或 **cotdq()** ，也可以使用 **-fpwidetypes** 选项进行编译。

要使用上面的任何函数，请确保程序包含 **<math.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

返回值

cotd(±0**)** 返回 ±INFINITY。

如果 *x* 为 ±INFINITY，则 **cotd()** 返回 NaN，并引起无效异常。

如果 *x* 为 NaN，则 **cotd()** 返回 NaN。

错误

没有定义任何错误。

另请参阅

atand(3M)、 cosd(3M)、 sincosd(3M)、 sind(3M)、 tand(3M)、 math(5)。

符合的标准

任何标准均未指定这些函数。

名称

cpacl()、fcpacl() - 将访问控制列表 (ACL) 和模式位从一个文件复制到另一个文件（仅适用于 HFS 和 JFS 文件系统）

概要

```
#include <acllib.h>

int cpacl(
  const char *fromfile,
  const char *tofile,
  mode_t frommode,
  uid_t fromuid,
  gid_t fromgid,
  uid_t touid,
  gid_t togid
);

int fcpacl(
  int fromfd,
  int tofd,
  mode_t frommode,
  uid_t fromuid,
  gid_t fromgid,
  uid_t touid,
  gid_t togid
);
```

备注

为确保持续符合新的行业标准，将来的发行版可能会更改此手册条目中描述的功能。

说明

cpacl() （适用于 HFS 和 JFS 文件系统）和 **fcpacl()** （仅适用于 HFS 文件系统）都可以将访问控制列表和模式位（即文件访问权限位和其他模式位；请参阅 *chmod*(2) ）从一个文件复制到另一个文件并且像 *chown*(2) 一样转移所有权。

cpacl() 只能将 HFS ACL 复制到其他 HFS 文件，或者将 JFS ACL 复制到其他 JFS 文件；而不能将 HFS ACL 转换为 JFS ACL，反之亦然。 **cpacl()** 和 **fcpacl()** 使用以下参数：

- 路径名（ fromfile 和 *tofile* ）或打开文件描述符（ fromfd 和 *tofd* ）。
- 一个模式值（ frommode ，通常是 **stat()** 返回的 **st_mode** 值 – 请参阅 *stat*(2) ），包含始终要复制的文件其他模式位，以及当两个文件为远程文件时，代替访问控制列表复制的文件访问权限位。

- 要转移所有权的文件的用户 ID 和组 ID （ fromuid 、 *touid* 和 *fromgid* 、 *togid* ）（通常， *fromuid* 和 *fromgid* 是 **stat()** 返回的 **st_uid** 值和 **st_gid** 值， *touid* 和 *togid* 是 **geteuid()** 和 **getegid()** 的返回值 – 请参阅 *getuid*(2) 中的 *geteuid*(2) 和 *getegid*(2) 。

如果两个文件都是本地的， **cpacl()** 例行程序将复制访问控制列表，必要时调用 **chownacl()** （仅适用于 HFS；请参阅 *chownacl*(3C) ），将所有权从 *fromfile* 转移至 *tofile* 。

cpacl() (**fcpacl()**) 发现 **acl()** 、 **getacl()** (**fgetacl()**) 或 **setacl()** (**fsetacl()**) （请参阅 *acl*(2) and *setacl*(2) ）失败后，可处理远程复制（通过 NFS），以及从 HFS 到 JFS 的复制或方向与此相反的复制。将模式从 *fromfile* (*fromfd*) 复制到 *tofile* (*tofd*) 时， **cpacl()** 使用 **chmod()** (**fchmod()**) 将整个 *frommode* （即文件其他模式位和文件访问权限位）复制到 *tofile* (*tofd*) 。可以关闭某些其他模式位；请参阅 *chmod*(2) 。

cpacl() (**fcpacl()**) 可以将访问控制列表从 *fromfile* (*fromfd*) 复制到 *tofile* (*tofd*) 且不转移所有权，但仍可确保执行错误检查和远程文件处理。参数传递时，使 *touid* 等于 *fromuid* ， *togid* 等于 *fromgid* ，即四个参数都是零，则可实现该操作。对于远程文件，将忽略 *fromuid* 、 *touid* 、 *fromgid* 和 *togid* 。

返回值

如果成功， **cpacl()** 和 **fcpacl()** 将返回零。如果出错，这些例行程序将设置 **errno** ，以指示失败的原因，同时返回以下负值:

–1 无法在本地 *fromfile* (*fromfd*) 上执行 **acl()** 或 **getacl()** (**fgetacl()**) 。

–2 无法在 *tofile*
.RI (tofd) 上执行 **chmod()** (**fchmod()**) 以设置其文件其他模式位。只要 *fromfile* (*fromfd*) 是本地的，则不论文件是本地的还是远程的， **cpacl()** (**fcpacl()**) 都会尝试此操作。

–3 无法在本地 *tofile* (*tofd*) 上执行 **acl()** 或 **setacl()** (**fsetacl()**) 。因此，将删除文件的可选 ACL 条目（仅适用于 HFS），将其文件访问权限位归零，并可能更改它的其他模式位。

–4 无法在 *tofile* (*tofd*) 上执行 **chmod()** (**fchmod()**) 以设置其模式。因此，如果 *fromfile* (*fromfd*) 是本地的，而不论文件是本地的还是远程的，都将删除 *tofile* 的（ tofd 的）可选 ACL 条目（仅适用于 HFS），将其访问权限位归零，并可能更改它的其文件其他模式位。

举例

以下代码段将获取有关 **oldfile** 的 *stat* 信息，并将其文件其他位和访问控制列表复制到调用方拥有的 **newfile** 。如果任何一个文件是远程的，则只复制 **oldfile** 上的 **st_mode** 。

```
#include <sys/types.h>
#include <sys/stat.h>

struct stat statbuf;

if (stat ("oldfile", & statbuf) < 0)
    error (...);
```

```
if (cpacl ("oldfile", newfile , statbuf.st_mode,
    statbuf.st_uid, statbuf.st_gid, geteuid()、 getegid()) < 0)
{
    error (...);
}
```

相关内容

只有标准的 HP-UX 操作系统上的 HFS 和 JFS 文件系统才支持 **cpacl()** 和 **fcpacl()** 。

作者

cpacl() 和 **fcpacl()** 由 HP 开发。

另请参阅

acl(2)、 chown(2)、 getacl(2)、 getegid(2)、 geteuid(2)、 getuid(2)、 setacl(2)、 stat(2)、 acltostr(3C)、 chownacl(3C)、 strtoacl(3C)、 acl(5)、 aclv(5) 和 thread_safety(5)。

名称

cpow( )、cpowf( )、cpowl( )、cpoww( )、cpowq( ) - 复变幂函数

概要

#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);

extended complex cpoww(extended complex x, extended complex y);

quad complex cpowq(quad complex x, quad complex y);

说明

这些函数仅适用于 Integrity 服务器。

cpow() 返回 x^y。沿负实轴方向，第一个参数有一个分支切割。

cpowf() 是 **float complex** 形式的 **cpow()** ；它采用 **float complex** 参数，并返回 **float complex** 结果。

cpowl() 是 **long double complex** 形式的 **cpow()** ；它采用 **long double complex** 参数，并返回 **long double complex** 结果。

cpoww() 是 **extended complex** 形式的 **cpow()** ；它采用 **extended complex** 参数，并返回 **extended complex** 结果。

cpowq() 等效于 HP-UX 系统上的 **cpowl()** 。

用法

要使用这些函数，请使用缺省的 **-Ae** 选项进行编译。要使用 **cpoww()** 或 **cpowq()** ，请使用 **-fpwidetypes** 选项进行编译。确保程序包含 **<complex.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

返回值

对于非零的 *y* ， **cpow(*x*,*y*)** 将返回 **cexp(*y**clog(*x*))** 的计算值，并从该计算中引发异常（中间计算结果的精度和范围可能会更大）。

cpow(*x*,±0±i0) 将返回 1+i0。

错误

没有定义任何错误。

另请参阅

cexp(3M)、 clog(3M)、 csqrt(3M)、 pow(3M)、 complex(5)。

符合的标准

cpow() 、 **cpowf()** 、 **cpowl()** ： ISO/IEC C99（包括附件 G“IEC 60559-compatible complex arithmetic”）

名称

cproj( )、cprojf( )、cprojl( )、cprojw( )、cprojq( ) - 将所有无穷大值映射到正实轴值的函数

概要

#include <complex.h>

double complex cproj(double complex z);

float complex cprojf(float complex z);

long double complex cprojl(long double complex z);

extended complex cprojw(extended complex z);

quad complex cprojq(quad complex z);

说明

这些函数仅适用于 Integrity 服务器。

cproj() 将返回其参数 *z* ，但如果此参数为复数无穷大（即使它是有一个无穷大部分和一个 NaN 部分的复数无穷大），则 **cproj()** 将返回实轴上的正无穷大，其虚数部分为 **copysign(0.0,cimag(** *z* **))** 。此函数可用于帮助建立 Riemann 球体模型。

cprojf() 是 **float complex** 形式的 **cproj()** ； 它采用 **float complex** 参数，并返回 **float complex** 结果。

cprojl() 是 **long double complex** 形式的 **cproj()** ； 它采用 **long double complex** 参数，并返回 **long double complex** 结果。

cprojw() 是 **extended complex** 形式的 **cproj()** ； 它采用 **extended complex** 参数，并返回 **extended complex** 结果。

cprojq() 等效于 HP-UX 系统上的 **cprojl()** 。

这些函数不会引发异常。

用法

要使用这些函数，请使用缺省的 **-Ae** 选项进行编译。要使用 **cprojw()** 或 **cprojq()** ，请使用 **-fpwidetypes** 选项进行编译。确保程序包含 **<complex.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

错误

没有定义任何错误。

另请参阅

creal(3M)、 cimag(3M)、 complex(5)。

符合的标准

cproj() 、 **cprojf()** 、 **cprojl()** ： ISO/IEC C99（包括附件 G“IEC 60559-compatible complex arithmetic”）

名称

cr_close - 关闭崩溃转储描述符

概要

#include <libcrash.h>

int cr_close(CRASH *crash_cb);

说明

cr_close() 函数关闭 *crash_cb* 指向的崩溃转储描述符结构。 **crash_cb** 结构是 *cr_open*(3) 调用填充的崩溃转储描述符。将释放所有相关资源（内存和文件）。

返回值

返回零表示成功。 *libcrash*(5) 中描述了其他的可能返回值。

作者

cr_close() 由 HP 开发。

另请参阅

cr_open(3)、libcrash(5)。

名称

creal( )、crealf( )、creall( )、crealw( )、crealq( ) - 复数值的实数部分

概要

#include <complex.h>

double creal(double complex z);

float crealf(float complex z);

long double creall(long double complex z);

extended crealw(extended complex z);

quad crealq(quad complex z);

说明

这些函数仅适用于 Integrity 服务器。

creal() 返回 *z* 的实数部分。

crealf() 是 **creal()** 的 **float complex** 版本；它采用 **float complex** 参数，并返回 **float** 结果。

creall() 是 **creal()** 的 **long double complex** 版本；它采用 **long double complex** 参数，并返回 **long double** 结果。

crealw() 是 **creal()** 的 **extended complex** 版本；它采用 **extended complex** 参数，并返回 **extended** 结果。

在 HP-UX 系统中， **crealq()** 等效于 **creall()** 。

这些函数不会引发异常。

用法

要使用这些函数，请使用缺省的 **-Ae** 选项进行编译。要使用 **crealw()** 或 **crealq()** ，请使用 **-fpwidetypes** 选项进行编译。确保程序包含 **<complex.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

错误

没有定义任何错误。

另请参阅

cimag(3M)、 complex(5)。

符合的标准

creal() 、 **crealf()** 和 **creall()** ： ISO/IEC C99（包括附件 G“IEC 60559-compatible complex arithmetic”）

名称

cr_info - 检索崩溃转储信息

概要

#include <libcrash.h>

cr_info_t *cr_info(CRASH *crash_cb);

说明

cr_info() 函数返回指向 **cr_info_t** 结构的指针，该结构带有与打开的崩溃转储有关的信息。

cr_info_t

cr_info_t 结构包含以下字段。请注意，此列表中的放置位置和结构中的顺序不存在必然的相关性，并且结构可以包含其他保留的字段。以后，可能会更改此结构的大小，并且不应复制此结构，以免与各发行版发生兼容性的问题。

char	****cri_hostname***	崩溃系统的名称
char	****cri_model***	系统的型号字符串
char	****cri_panic***	异常消息
char	****cri_release***	系统的发行版字符串
char	****cri_dumptime***	崩溃转储的时间
char	****cri_savetime***	保存崩溃转储的时间
uint64_t	***cri_chunksize***	未压缩的映像文件的大小。
uint64_t	***cri_memsize***	内存的物理大小（字节）
uint64_t	***cri_num_err***	错误消息的数目
char	*****cri_errmsg***	来自 ***INDEX*** 文件的错误消息
uint64_t	***cri_num_warn***	警告消息的数目
char	*****cri_warnmsg***	来自 ***INDEX*** 文件的警告消息
uint64_t	***cri_num_mod***	模块结构的数目
cri_modinfo_t	****cri_modinfo***	模块结构的数组
uint32_t	***cri_num_nodes***	节点数
uint32_t	***cri_monarch_node***	***monarch*** 物理节点数

字段 *cri_hostname* 、 *cri_model* 、 *cri_panic* 、 *cri_release* 、 *cri_dumptime* 和 *cri_savetime* 都是以 null 结尾的字符串；它们分别代表已转储系统的名称、已转储系统的型号、异常消息、内核的发行版字符串、转储系统的时间和保存转储的时间。 *cri_memsize* 中定义了系统上的物理内存量（单位为字节）。 *cri_chunksize* 中定义了单个未压缩映像文件的目标字节大小。 *cri_errmsg* 和 *cri_warnmsg* 是带有已保存消息列表的以 null 结尾的字符串的数组。 *cri_num_err* 和 *cri_num_warn* 是每个列表中的字符串数。 *cri_modinfo* 是定义已加载内核模块的 **cri_modinfo_t** 结构的数组；数组中有 *cri_num_mod* 个元素。 *cri_num_nodes* 是计算机包含的节点数， *cri_monarch_node* 是 monarch 的物理节点数。

cri_modinfo_t

cri_modinfo_t 结构包含系统崩溃时载入内存的特定内核模块的信息。特定的字段包括：

char	****mdi_loadpath***	载入内存时的文件路径
char	****mdi_savepath***	保存到崩溃目录时的文件路径
uint64_t	***mdi_size***	字节大小
uint64_t	***mdi_checksum***	***cksum(1)*** 中的文件校验和

同样，不指定结构中这些字段的顺序，并且结构中可能存在其他保留字段。

调用方不得修改 **cr_info()** 返回的任何结构的任何信息。

返回值

cr_info() 返回指向上述 **cr_info_t** 结构的指针。如果 **crash_cb** 不是使用 **cr_open()** 打开的崩溃转储的有效描述符，那么返回值是不确定的。

举例

假定某个进程打开了一个崩溃转储，对 *cr_info*(3) 执行以下调用将检索与打开的崩溃转储有关的信息。

```
#include <libcrash.h>

CRASH *crshdes;
cr_info_t *cri;

cri = cr_info(crshdes);
```

作者

cr_info() 由 HP 开发。

另请参阅

cr_open(3)、libcrash(5)。

名称

cr_isaddr - 验证是否已转储物理页码

概要

#include <libcrash.h>

int cr_isaddr(CRASH *crash_cb, uint64_t pagenum, int *avail);

说明

cr_isaddr() 检查指定的页码 *pagenum* 是否出现在由 *crash_cb* 表示的打开的崩溃转储中。它将布尔值设置为 *avail* 指向的值，以表示该页是存在 (1) 还是不存在 (0)。

返回值

返回零表示成功。 *libcrash*(5) 中描述了其他的可能返回值。

举例

假设某个进程已打开崩溃转储，对 *cr_isaddr*(3) 进行的以下调用将测试 256 页是否可用：

```
#include <libcrash.h>

CRASH cr_cb;
int avail;

int retval = cr_isaddr(&cr_cb, (uint64_t)256, &avail);

if (retval >= 0 && avail > 0) {
    /* Read and process page 256 */
} /* Else ignore pages that are not available */
```

作者

cr_isaddr() 由 HP 开发。

另请参阅

cr_open(3)、cr_read(3)、libcrash(5)。

名称

cr_open() - 打开崩溃转储，进行读取

概要

#include <libcrash.h>

int cr_open(const char *path, CRASH **crash_cb, int flags);

说明

cr_open() 库调用将打开崩溃转储，并回传崩溃转储描述符。

path 参数指向用于命名崩溃转储目录或文件的路径名，且长度不得超过 **PATH_MAX** 字节。

crash_cb 所指向的 **CRASH *** 被设置为崩溃转储描述符，它可以回传给其他 **cr_*()** 函数，用于访问崩溃转储。

flags 是零的位码或以下的标志值，它将影响以后为崩溃转储调用 **libcrash** 例行程序时的操作，但 *cr_verify*(3) 除外，因为它有自己的 *flags* 参数。

CR_NOCHECKSUM 设置此标志后，库不会再在故障转储中验证文件的校验和。

CR_DELAYMSGS 如果设置此标志，在较为耗时的操作（解压缩及校验和）过程中，库会将消息写入 stderr。

返回值

返回 0，表示成功。有关其他可能的返回值的说明，请参考 *libcrash*(5)。

举例

以下对 **cr_open()** 的调用将打开目录 **/var/adm/crash/core.0** 中所含的崩溃转储，并返回崩溃转储描述符 **crash_cb**。有关读取崩溃转储 **/var/adm/crash/core.0** 的示例，请参阅 *cr_read*(3) 手册条目。

```
#include <libcrash.h>

CRASH *crash_cb;
int    ret;

ret = cr_open ("/var/adm/crash/core.0", &crash_cb, CR_DELAYMSGS);
```

作者

cr_open() 由 HP 开发。

另请参阅

cr_close(3)、 cr_perror(3)、 libcrash(5)。

名称

cr_perror - 输出 libcrash 错误或警告消息

概要

#include <libcrash.h>

void cr_perror(CRASH *crash_cb, int error);

说明

cr_perror() 将 *error* 所对应的错误或警告消息输出到标准错误中，它应该是一系列 **libcrash** 调用中紧邻的前一个调用的返回值。该消息描述出现的问题，在需要时说明它的含义，并提供适当的纠正操作。

如果使用值为零的 *error* 进行调用，则表示成功， **cr_perror()** 不输出任何消息。因此，在每个 **libcrash** 调用后调用它不会造成损害（如果在 **cr_verify()** 后使用已设置的 **CR_ERRORMSGS** 标志进行调用，则可能会产生重复的错误消息）。

crash_cb 应该是崩溃转储描述符，它与传递到返回 *error* 的调用的崩溃转储描述符相同。如果调用是 **cr_open()** ，则传递该调用返回的崩溃转储描述符，即使它为空也是如此。

如果 *error* 为 **CRERR_ERRNO** ，则表示系统变量 **errno** 包含错误信息， **cr_perror()** 将引用 **errno** 以便输出相应的消息。这种情况下，在引起冲突的 **libcrash** 调用返回后立即调用 **cr_perror()** 是至关重要的。

作者

cr_perror () 由 HP 开发。

另请参阅

cr_open(3)、cr_verify(3)、libcrash(5)。

名称

cr_read - 读取崩溃转储

概要

#include <libcrash.h>

int cr_read(CRASH *crash_cb, void *buf,
uint64_t mem_page, int *num_pages);

说明

cr_read() 函数尝试将 *mem_page* 和 *num_pages* 定义的内存区域读入使用 *crash_cb* 打开的崩溃转储中的 *buf* 指向的缓冲区。

cr_read() 从与 *mem_page* 给出的物理内存偏移量相关的崩溃转储中的位置开始。如果物理内存页 *mem_page* 不在崩溃转储中，则 **cr_read()** 将 **num_pages* 设置为 0 并返回 0。

超出不在崩溃转储中内存页的部分，不会发生数据传输。如果开始位置 *mem_page* 加上读取长度 **num_pages* 超出了不在崩溃转储中的内存区域，则 **cr_read()** 将 **num_pages* 设置为实际读取的连续页数（从 *mem_page* 开始）。

返回值

返回零表示成功。 *libcrash*(5) 中描述了其他的可能返回值。

举例

假设某个进程已打开崩溃转储，对 *cr_read*(3) 进行的以下调用会将崩溃转储中前 **CRSIZE** 页读入 *mybuf* 指向的缓冲区：

```
#include <libcrash.h>

CRASH in_cb;
char mybuf[CRSIZE*NBPG];
int rstat;
int size = CRSIZE;

rstat = cr_read (in_cb, mybuf, 0, &size);
```

警告

由于实现方法的具体细节不同， **cr_read()** 返回的页数可能比请求的页数少。请始终检查返回的页数。如果返回的页数少于请求的页数，则发出新的请求，请求从没有返回的第一页开始。仅当新的请求读取零页（或返回错误）时，才能确定该页没有转储。

作者

cr_read() 由 HP 开发。

另请参阅

cr_open(3)、libcrash(5)。

名称

cr_set_node -设置节点号

概要

#include <libcrash.h>

int cr_set_node(CRASH *crash_cb, int node_num, int *old_node_num);

说明

cr_set_node() 函数需要在 **node_num** 中传递的物理节点号。 **cr_read()** 和 **cr_isaddr()** 将使用该节点号访问特定节点包含的节点专用内存。该函数仅对 **ccnumadir** （第 4 版）转储有效。如果 **old_node_num** 参数为非空，则 **old_node_num** 引用的地址将被设置为前一个节点号。如果 **old_node_num** 为空，则不返回前一个节点号。

返回值

返回零表示成功。 *libcrash*(5) 中描述了其他的可能返回值。

举例

假设某个进程已打开崩溃转储，对 **cr_set_node()** 进行的以下调用会将物理节点号设置为零。

```
#include <libcrash.h>

CRASH *cr_cb;
int   old_node;
int   retval;

retval = cr_set_node(cr_cb, 0, &old_node);
```

作者

cr_set_node() 由 HP 开发。

另请参阅

cr_open(3)、libcrash(5)。

名称

crt0： crt0.o、gcrt0.o、mcrt0.o - 执行启动例行程序

概要

备注

对于 Integrity 系统，请参阅 *crt0_ia*(3) 。

对于 PA-RISC 系统，请参阅 *crt0_pa*(3) 。

使用 **uname** 命令确定您的系统类型。在 Integrity 系统上， **uname -m** 返回 **ia64** 。其他所有值均表示 PA-RISC 系统。

另请参阅

crt0_ia(3)、 crt0_pa(3)、 uname(1)。

名称

crt0_ia： crt0.o - 执行启动例行程序，适用于 Integrity 系统

概要

备注

此联机帮助页介绍了 Integrity 系统上的 **crt0.o** 。有关 PA-RISC 系统上的 **crt0.o** ，请参阅 *crt0_pa*(3) 。

说明

C、aC++ 和 FORTRAN 编译程序在静态绑定程序的对象文件 **crt0.o** 中链接，以便提供程序执行的启动功能和环境。它包含必须使用 **ld** 才能链接到每一个静态绑定程序的启动代码。在动态链接程序（缺省方法）中，不使用 **crt0.o** 对象文件，并且改为由动态加载程序 *dld.so*(5) 完成通常与其相关的所有操作。

crt0.o 处理 *Initializer* 和 *Terminator* 。 *Initializer* 是在程序入口点之前调用的例行程序，而 *Terminator* 是程序通过 **exit** 例行程序终止时调用的例行程序。 *Initializer* 按照链接行的相反顺序调用，这样，可以先初始化相关库，然后再初始化依赖这些相关库的其他库。而另一方面，对于 *Terminator* ，程序将以正向顺序调用它。

crt0.o 不定义任何变量。但它会设置下列全局变量:

__argc	*long* 类型的变量，包含参数的数目。
__argv	指向参数自身的字符指针的数组。
_environ, __envp	指向程序运行环境的字符指针的数组。该数组以空指针结束。
_SYSTEM_ID	*long* 类型的变量，包含可执行程序的系统 ID 值。
__tls_size	*long* 类型的变量，它包含所的请求线程本地存储大小。将使用内核中的数据初始化该变量。
__load_info	*void* * 类型的变量，它包含从内核传递的加载信息。
__gp	*void* * 类型的变量，其值等于全局指针。 在 **<machine/sys/uregs.h>** 中还声明了一个名为 **__gp** 的符号作为枚举常量，并且与 **<crt0.h>** 中的声明不兼容。要同时包含两个头文件，必须在包含 **<machine/sys/uregs.h>** 之前定义宏 **__UREGS_SKIP__GP** 。这将强制 **<machine/sys/uregs.h>** 头文件忽略 **__gp** 的冲突定义。在此环境中，枚举常量 **__r1** 等于 **__gp** ，并可以替换使用。请参阅 *ttrace*(2) 。

作者

本条目中描述的功能来自 AT&T UNIX System III。

文件

crt0.h

另请参阅

系统工具

aCC(1)	调用 HP-UX aC++ 编译程序
cc(1)	调用 HP-UX C 编译程序
dld.so(5)	动态加载程序
exec(2)	执行文件
f90(1)	调用 HP-UX FORTRAN 编译程序
ld(1)	调用链接编辑器

其他信息

end(3C)	程序中最后位置的符号

名称

crt0_pa： crt0.o、gcrt0.o、mcrt0.o - 执行启动例行程序，适用于 PA-RISC 系统

概要

备注

PA-RISC 64 位 ELF 只使用 **crt0.o** 。

此联机帮助页介绍了 PA-RISC 系统上的 **crt0.o** 。有关 Integrity 系统上的 **crt0.o** ，请参阅 *crt0_ia*(3) 。

说明

PA-RISC 32 位 SOM

C 语言、Pascal 和 FORTRAN 编译程序可链接对象文件 **crt0.o** 、 **gcrt0.o** 或 **mcrt0.o** ，以便为程序执行提供启动功能和环境。所有这些对象文件是相似的，不同之处在于， **gcrt0.o** 和 **mcrt0.o** 可分别为 *gprof*(1) 和 *prof*(1) 的配置处理支持提供附加的功能。

这些对象文件中定义了以下符号：

_environ	字符指针的数组，这些指针指向要在其中运行程序的环境。该数组以空指针结束。
_FPU_MODEL	*short* 类型的变量，包含 FP 状态指令返回的 FPU 型号。将使用内核中的数据初始化该变量。
_FPU_REVISION	*short* 类型的变量，包含 FP 状态指令返回的 FPU 版本号。将使用内核中的数据初始化该变量。
_CPU_KEYBITS_1	*int* 类型的变量，包含特定 CPU 的信息。将使用内核中的数据初始化该变量。
_CPU_REVISION	*int* 类型的变量，包含计算机的 CPU 版本。将使用内核中的数据初始化该变量。
_SYSTEM_ID	*int* 类型的变量，包含可执行程序的系统 ID 值。
$START$	执行启动地址。
_start	C 语言程序的辅助启动例行程序，将通过 **$START$** 调用该例行程序，而该例行程序又将调用 **main** 。该例行程序包含在 C 语言库，而不是 **crt0.o** 文件中。对于 Pascal 和 FORTRAN 程序，此符号可标记外部块（主程序）的开头，并且它是由编译程序生成的。
$global$	程序的数据指针的初始地址。启动代码可将此地址载入常规寄存器 27。
$UNWIND_START	堆栈辗转开解 (unwind) 表的开头。
$UNWIND_END	堆栈辗转开解表的结尾。
$RECOVER_START	尝试/恢复表的开头。
$RECOVER_END	尝试/恢复表的结尾。

crt0.o 文件可为 **_mcount** 定义一个空过程，因此，不使用配置处理便可以链接通过配置处理编译的程序。

链接程序定义了以下两个符号:

__text_start	程序的文本区的起始地址。
__data_start	程序的数据区的起始地址。

PA-RISC 64 位 ELF

C 语言、Pascal 和 FORTRAN 编译程序可链接对象文件 **crt0.o** ，以便为程序执行提供启动功能和环境。该对象文件包含启动代码，必须使用 **ld** 将该启动代码链接到每个 **-noshared** PA-RISC 64 位程序。在 **-dynamic** 程序中，不使用 **crt0.o** 对象文件，而是通过动态加载程序 *dld.sl*(5) 执行与该对象文件正常关联的所有操作。 **crt0.o** 不再处理为 *prof*(1) 和 *gprof*(1) 配置处理支持提供的附加功能。

在 PA-RISC 64 位环境中， **crt0.o** 处理 *Initializer* 和 *Terminator* 。而 *Initializer* 是在程序入口点之前调用的例行程序， *Terminator* 是通过 **exit** 例行程序终止程序时调用的例行程序。 *Initializer* 按照链接行的相反顺序调用，这样，可以先初始化相关库，然后再初始化依赖这些相关库的其他库。而另一方面，对于 *Terminator* ，程序将以正向顺序调用它。

与 SOM 版本的 **crt0.o** 不同，PA-RISC 64 位 ELF 的 **crt0.o** 不定义任何变量。但是，该对象文件可设置以下全局变量:

__argc	*long* 类型的变量，包含参数的数目。
__argv	指向参数自身的字符指针的数组。
_environ, __envp	字符指针的数组，这些指针指向要在其中运行程序的环境。该数组以空指针结尾。
_CPU_KEYBITS_1	*int* 类型的变量，包含特定 CPU 的信息。将使用内核中的数据初始化该变量。
_FPU_MODEL	*long* 类型的变量，包含 FP 状态指令返回的 FPU 型号。将使用内核中的数据初始化该变量。
_FPU_REVISION	*long* 类型的变量，包含 FP 状态指令返回的 FPU 版本号。将使用内核中的数据初始化该变量。
_CPU_REVISION	*long* 类型的变量，包含计算机的 CPU 版本。将使用内核中的数据初始化该变量。
_SYSTEM_ID	*long* 类型的变量，包含可执行程序的系统 ID 值。
__tls_size	*long* 类型的变量，它包含所的请求线程本地存储大小。将使用内核中的数据初始化该变量。
__load_info	*void* * 类型的变量，它包含从内核传递的加载信息。

作者

本条目中描述的功能来自 AT&T UNIX System III。

文件

crt0.h

另请参阅

配置处理和调试工具

gprof(1)	显示调用图形配置文件数据
monitor(3C)	准备执行配置文件
prof(1)	显示配置文件数据
profil(2)	执行时间配置文件

系统工具

cc(1)	调用 HP-UX C 编译程序
exec(2)	执行文件
f77(1)	调用 HP-UX FORTRAN 编译程序
ld(1)	调用链接编辑器
dld.sl(5)	PA-RISC 动态加载程序
pc(1)	调用 HP-UX Pascal 编译程序

其他信息

end(3C)	程序中最后位置的符号

名称

cr_uncompress - 解压缩崩溃转储中的文件

概要

#include <libcrash.h>

int cr_uncompress(CRASH *crash_cb, const char *pathname,
uint64_t size, uint64_t checksum);

说明

cr_uncompress() 可确保对作为 *crash_cb* 所描述的崩溃转储组成部分的文件进行解压缩，并使之与预期的大小和校验和（利用 *cksum*(1) 计算而来）相匹配。此调用通常用于确保作为崩溃转储组成部分的模块文件的完整性；请参阅 *cr_info*(3) 。

pathname 是要解压缩的文件的名称。支持的压缩方法包括 *gzip*(1) （在文件名上附加 **.gz** ）和 *compress*(1) （附加 **.Z** ）。 *size* 和 *checksum* 分别是解压缩的文件的预期大小及校验和。通过为对应的参数指定 0，可禁止任一有效性检查。

返回值

返回 0，表示成功。有关其他可能的返回值的说明，请参考 *libcrash*(5) 。

举例

以下对 *cr_uncompress*(3) 的调用可确保对内核文件 **vmunix** 进行解压缩和验证。

```
#include <libcrash.h>

result = cr_uncompress (crash, "vmunix", vmunix_size, vmunix_cksum);
```

作者

cr_uncompress() 由 HP 开发。

另请参阅

gunzip(1)、uncompress(1)、cr_info(3)、cr_open(3)、libcrash(5)。

名称

cr_verify - 验证崩溃转储的完整性

概要

#include <libcrash.h>

int cr_verify(CRASH *crash_cb, int flags);

说明

cr_verify() 将解压缩并验证 *crash_cb* 所标识的崩溃转储中每个文件的大小及校验和。

flags 是零的位码或以下的标志值：

CR_NOCHECKSUM 设置此标志后， **cr_verify()** 不会再在故障转储中验证文件的校验和。此时将仅对大小进行验证。

CR_DELAYMSGS 如果设置此标志，在较为耗时的操作（解压缩及校验和）过程中， **cr_verify()** 会将消息写入 stderr。

CR_ERRORMSGS **cr_verify()** 将把描述所遇到的任何验证问题的消息写入 stderr 。如果设置此标志，则在 **cr_verify()** 返回时不应调用 **cr_perror()** ；这样会生成重复的错误消息。

返回值

返回 0，表示成功。有关其他可能的返回值的说明，请参考 *libcrash*(5) 。

举例

以下对 *cr_verify*(3) 的调用将验证转储的完整性。

```
#include <libcrash.h>

CRASH *crash_cb;
int ret;

ret = cr_verify(crash_cb, CR_DELAYMSGS | CR_ERRORMSGS);
```

警告

由于 **cr_verify()** 将对转储中的所有文件进行解压缩并验证其校验和，因此此过程会非常耗时。如果未指定 **CR_DELAYMSGS** ，则在调用 **cr_verify()** 前，调用应用程序应通知其用户：此调用可能会有较长时间的延迟。

作者

cr_verify() 由 HP 开发。

另请参阅

cr_open(3)、cr_perror(3)、libcrash(5)。

名称

crypt - 生成散列加密

概要

```
#include <crypt.h>
#include <unistd.h>

char *crypt(const char *key, const char *salt);
```

过时的接口

```
char *crypt_r(const char *key, const char *salt, CRYPTD *cd);

void setkey_r(const char *key, CRYPTD *cd);

void encrypt_r(char block[64], int edflag, CRYPTD *cd);

void setkey(const char *key);

void encrypt(char block[64], int edflag);
```

说明

crypt( ):

crypt() 是一种口令加密函数。它基于单向散列加密算法，并有多种变体，用于阻止使用密钥搜索的硬件实现。

key 是用户键入的口令。 *salt* 是从 **[a-zA-Z0-9./]** 集合中选出的双字符字符串；此字符串用于以 4096 种方式之一来阻止执行散列算法，之后将使用口令作为密钥，反复加密常量字符串。返回值指向加密的口令。前两个字符为 salt 自身。

过时的接口

crypt_r() 、 **setkey_r()** 、 **encrypt_r()** 、 **setkey()** 和 **encrypt()** 生成散列加密。

警告

crypt() 的返回值指向其内容被每次调用所覆盖的数据。

crypt_r() 、 **setkey_r()** 和 **encrypt_r()** 是仅支持与现有 DCE 应用程序兼容的过时接口。新的多线程应用程序应使用 **crypt()** 。

另请参阅

crypt(1)、login(1)、passwd(1)、getpass(3C)、passwd(4) 和 thread_safety(5)。

符合的标准

crypt()：SVID2、SVID3、XPG2、XPG3、XPG4

名称

csin( )、csinf( )、csinl( )、csinw( )、csinq( ) - 复变正弦函数

概要

#include <complex.h>

double complex csin(double complex z);

float complex csinf(float complex z);

long double complex csinl(long double complex z);

extended complex csinw(extended complex z);

quad complex csinq(quad complex z);

说明

这些函数仅适用于 Integrity 服务器。

csin() 返回 *z* 的复变正弦值。

csinf() 是 **csin()** 的 **float complex** 版本；它采用 **float complex** 参数，并返回 **float complex** 结果。

csinl() 是 **csin()** 的 **long double complex** 版本；它采用 **long double complex** 参数，并返回 **long double complex** 结果。

csinw() 是 **csin()** 的 **extended complex** 版本；它采用 **extended complex** 参数，并返回 **extended complex** 结果。

在 HP-UX 系统中， **csinq()** 等效于 **csinl()** 。

用法

要使用这些函数，请使用缺省的 **-Ae** 选项进行编译。要使用 **csinw()** 或 **csinq()** ，请使用 **-fpwidetypes** 选项进行编译。确保程序包含 **<complex.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

返回值

csin(conj(*z***)) = conj(csin(***z***))** ，并且 **csin** 为奇数。

csin(+0+i0**)** 返回 +0+i0。

csin(+Inf+i0**)** 返回 NaN±i0 （此处未指定结果的虚数部分的符号），并引起无效浮点运算异常。

csin(+NaN+i0**)** 返回 NaN±i0 （此处未指定结果的虚数部分的符号）。

对于有限正 *y* ， **csin(**Inf+i*y***)** 返回 NaN+iNaN，并引起无效浮点运算异常。

对于非零有限 *y* ， **csin(**NaN+i*y***)** 返回 NaN+iNaN，并可能引起无效浮点运算异常。

csin(+0+iInf**)** 返回 +0+iInf。

对于有限正 *x* ， **csin(**+*x*+iInf**)** 返回 +Inf**(sin(***y***)+icos(***y***))** 。

csin(+Inf+iInf**)** 返回 NaN±iInf （此处未指定结果的虚数部分的符号），并引起无效浮点运算异常。

csin(+NaN+iInf**)** 返回 +NaN±iInf （此处未指定结果的虚数部分的符号）。

csin(0+iNaN**)** 返回 0+iNaN。

对于所有非零 *x* ， **csin(***x*+iNaN**)** 返回 NaN+iNaN，并可能引起无效浮点运算异常。

csin(NaN+iNaN**)** 返回 NaN+iNaN。

错误

没有定义任何错误。

另请参阅

sin(3M)、 casin(3M)、 complex(5)。

符合的标准

csin() 、 **csinf()** 和 **csinl()** ： ISO/IEC C99（包括附件 G“IEC 60559-compatible complex arithmetic”）

名称

csinh( )、csinhf( )、csinhl( )、csinhw( )、csinhq( ) - 复变双曲正弦函数

概要

#include <complex.h>

double complex csinh(double complex z);

float complex csinhf(float complex z);

long double complex csinhl(long double complex z);

extended complex csinhw(extended complex z);

quad complex csinhq(quad complex z);

说明

这些函数仅适用于 Integrity 服务器。

csinh() 返回 *z* 的复变双曲正弦值。

csinhf() 是 **csinh()** 的 **float complex** 版本；它采用 **float complex** 参数，并返回 **float complex** 结果。

csinhl() 是 **csinh()** 的 **long double complex** 版本；它采用 **long double complex** 参数，并返回 **long double complex** 结果。

csinhw() 是 **csinh()** 的 **extended complex** 版本；它采用 **extended complex** 参数，并返回 **extended complex** 结果。

在 HP-UX 系统中， **csinhq()** 等效于 **csinhl()** 。

用法

要使用这些函数，请使用缺省的 **-Ae** 选项进行编译。要使用 **csinhw()** 或 **csinhq()** ，请使用 **-fpwidetypes** 选项进行编译。确保程序包含 **<complex.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

返回值

csinh(conj(*z***)) = conj(csinh(***z***))** ，并且 **csinh** 为奇数。

csinh(+0+i0) 返回 +0+i0。

csinh(+0+iInf) 返回 ±0+iNaN （此处未指定结果的实数部分的符号），并引起无效浮点运算异常。

csinh(+0+iNaN) 返回 ±0+iNaN （此处未指定结果的实数部分的符号）。

对于有限正 *x* ， **csinh**(*x*+iInf) 返回 NaN+iNaN，并引起无效浮点运算异常。

对于非零有限 *x* ， **csinh**(*x*+iNaN) 返回 NaN+iNaN，并可能引起无效浮点运算异常。

csinh(+Inf+i0) 返回 +Inf+i0。

对于有限正 *y* ， **csinh**(+Inf+i*y*) 返回 +Inf(**cos**(*y*)+**isin**(*y*)) 。

csinh(+Inf+iInf) 返回 ±Inf+iNaN （此处未指定结果的实数部分的符号），并引起无效浮点运算异常。

csinh(+Inf+iNaN) 返回 ±Inf+iNaN （此处未指定结果的实数部分的符号）。

csinh(NaN+i0) 返回 NaN+i0。

对于所有非零 *y* ， **csinh**(NaN+i*y*) 返回 NaN+iNaN，并可能引起无效浮点运算异常。

csinh(NaN+iNaN) 返回 NaN+iNaN。

错误

没有定义任何错误。

另请参阅

sinh(3M)、 casinh(3M)、 complex(5)。

符合的标准

csinh()、 **csinhf**() 和 **csinhl**()： ISO/IEC C99（包括附件 G“IEC 60559-compatible complex arithmetic”）

名称

csqrt( )、csqrtf( )、csqrtl( )、csqrtw( )、csqrtq( ) - 复数平方根函数

概要

#include <complex.h>

double complex csqrt(double complex z);

float complex csqrtf(float complex z);

long double complex csqrtl(long double complex z);

extended complex csqrtw(extended complex z);

quad complex csqrtq(quad complex z);

说明

这些函数仅适用于 Integrity 服务器。

csqrt() 返回 *z* 的复数平方根，范围为右半平面（包括虚轴）。沿负实轴有一个分支切割。

csqrtf() 是 **float complex** 形式的 **csqrt()** ；它采用 **float complex** 参数，并返回 **float complex** 结果。

csqrtl() 是 **long double complex** 形式的 **csqrt()** ；它采用 **long double complex** 参数，并返回 **long double complex** 结果。

csqrtw() 是 **extended complex** 形式的 **csqrt()** ；它采用 **extended complex** 参数，并返回 **extended complex** 结果。

csqrtq() 等效于 HP-UX 系统上的 **csqrtl()** 。

用法

要使用这些函数，请使用缺省的 **-Ae** 选项进行编译。要使用 **csqrtw()** 或 **csqrtq()** ，请使用 **-fpwidetypes** 选项进行编译。确保程序包含 **<complex.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

返回值

csqrt(conj(*z***)) = conj(csqrt(***z***))**。

csqrt(±0+i0**)** 将返回 +0+i0。

对于所有 *x* （包括 NaN） ， **csqrt(***x***+iInf)** 将返回 +Inf+iInf。

对于有限值 *x* ， **csqrt(***x***+iNaN)** 将返回 NaN+iNaN，并可选择引发无效的浮点异常。

对于正有限值 *y* ， **csqrt(**-Inf+i*y***)** 将返回 +0+iInf。

对于正有限值 *y* ， **csqrt(**+Inf+i*y***)** 将返回 +Inf+i0。

csqrt(-Inf+iNaN**)** 将返回 NaN±iInf （这里未指定结果的虚数部分的符号）。

csqrt(+Inf+iNaN**)** 将返回 +Inf+iNaN。

对于有限值 *y* ， **csqrt(**NaN+i*y***)** 将返回 NaN+iNaN，并可选择引发无效的浮点异常。

csqrt(NaN+iNaN) 将返回 NaN+iNaN。

错误

没有定义任何错误。

另请参阅

sqrt(3M)、 cpow(3M)、 complex(5)。

符合的标准

csqrt() 、 **csqrtf**() 、 **csqrtl**() ： ISO/IEC C99（包括附件 G“IEC 60559-compatible complex arithmetic”）

名称

ctan( )、ctanf( )、ctanl( )、ctanw( )、ctanq( ) - 复变正切函数

概要

#include <complex.h>

double complex ctan(double complex z);

float complex ctanf(float complex z);

long double complex ctanl(long double complex z);

extended complex ctanw(extended complex z);

quad complex ctanq(quad complex z);

说明

这些函数仅适用于 Integrity 服务器。

ctan() 返回 *z* 的复变正切值。

ctanf() 是 **float complex** 形式的 **ctan()** ；它采用 **float complex** 参数，并返回 **float complex** 结果。

ctanl() 是 **long double complex** 形式的 **ctan()** ；它采用 **long double complex** 参数，并返回 **long double complex** 结果。

ctanw() 是 **extended complex** 形式的 **ctan()** ；它采用 **extended complex** 参数，并返回 **extended complex** 结果。

ctanq() 等效于 HP-UX 系统上的 **ctanl()** 。

用法

要使用这些函数，请使用缺省的 **-Ae** 选项进行编译。要使用 **ctanw()** 或 **ctanq()** ，请使用 **-fpwidetypes** 选项进行编译。确保程序包含 **<complex.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

返回值

ctan(conj(*z***)) = conj(ctan(***z***))** 和 **ctan** 属于奇函数。

ctan(+0+i0**)** 将返回 +0+i0。

对于有限值 *y* ， **ctan(**Inf+i*y***)** 将返回 NaN+iNaN，并引发无效的浮点异常。

对于有限值 *y* ， **ctan(**NaN+i*y***)** 将返回 NaN+iNaN，并可选择引发无效的浮点异常。

对于正有限值 *x* ， **ctan(**+*x*+iInf**)** 将返回 **0sin(**2*x***)**+i1 。

ctan(+Inf+iInf**)** 将返回 ±0+i1 （这里未指定结果的实数部分的符号）。

ctan(+NaN+iInf**)** 将返回 ±01+i1 （这里未指定结果的实数部分的符号）。

ctan(0+iNaN**)** 将返回 0+iNaN。

对于所有非零的数字值 *x* ， **ctan(***x***+**iNaN**)** 将返回 NaN+iNaN，并可选择引发无效的浮点异常。

ctan(NaN+iNaN) 将返回 NaN+iNaN。

错误

没有定义任何错误。

另请参阅

tan(3M)、 catan(3M)、 complex(5)。

符合的标准

ctan() 、 **ctanf()** 、 **ctanl()** ： ISO/IEC C99（包括附件 G“IEC 60559-compatible complex arithmetic”）

名称

ctanh( )、ctanhf( )、ctanhl( )、ctanhw( )、ctanhq( ) - 复变双曲正切函数

概要

#include <complex.h>

double complex ctanh(double complex z);

float complex ctanhf(float complex z);

long double complex ctanhl(long double complex z);

extended complex ctanhw(extended complex z);

quad complex ctanhq(quad complex z);

说明

这些函数仅适用于 Integrity 服务器。

ctanh() 返回 *z* 的复变双曲正切值。

ctanhf() 是 **float complex** 形式的 **ctanh()**；它采用 **float complex** 参数，并返回 **float complex** 结果。

ctanhl() 是 **long double complex** 形式的 **ctanh()**；它采用 **long double complex** 参数，并返回 **long double complex** 结果。

ctanhw() 是 **extended complex** 形式的 **ctanh()**；它采用 **extended complex** 参数，并返回 **extended complex** 结果。

ctanhq() 等效于 HP-UX 系统上的 **ctanhl()** 。

用法

要使用这些函数，请使用缺省的 **-Ae** 选项进行编译。要使用 **ctanhw()** 或 **ctanhq()**，请使用 **-fpwidetypes** 选项进行编译。确保程序包含 **<complex.h>**，通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

返回值

ctanh(conj(*z***)) = conj(ctanh(***z***))** 和 **ctanh** 属于奇函数。

ctanh(+0+i0**)** 将返回 +0+i0。

对于有限值 *x*， **ctanh(***x***+**iInf**)** 将返回 NaN+iNaN，并引发无效的浮点异常。

对于有限值 *x*， **ctanh(***x***+**iNaN**)** 将返回 NaN+iNaN，并可选择引发无效的浮点异常。

对于正有限值 *y*， **ctanh(**+Inf+i*y***)** 将返回 1+i0**sin(**2*y***)** 。

ctanh(+Inf+iInf**)** 将返回 1±i0 （这里未指定结果的虚数部分的符号）。

ctanh(+Inf+iNaN**)** 将返回 1±i0 （这里未指定结果的虚数部分的符号）。

ctanh(NaN+i0**)** 将返回 NaN+i0。

对于所有非零的数字值 *y*， **ctanh(**NaN+i*y***)** 将返回 NaN+iNaN，并可选择引发无效的浮点异常。

ctanh(NaN+iNaN**)** 将返回 NaN+iNaN。

错误

没有定义任何错误。

另请参阅

tanh(3M)、 catanh(3M)、 complex(5)。

符合的标准

ctanh() 、 **ctanhf()** 、 **ctanhl()** ： ISO/IEC C99（包括附件 G“IEC 60559-compatible complex arithmetic”）

名称

ctermid( ) - 为终端生成文件名

概要

#include <stdio.h>

char *ctermid(char *s);

说明

ctermid() 生成的字符串在用作路径名时，将引用当前进程的控制终端。

如果 *s* 是一个空指针，则字符串存储在内部静态区域，在对 **ctermid()** 进行下一次调用时将覆盖静态区域的内容并返回静态区域的地址。否则，将假定 *s* 指向至少包含 **L_ctermid** 元素的字符数组；路径名位于该数组中，并返回 *s* 的值。常量 **L_ctermid** 是在 **<stdio.h>** 头文件中进行定义的。

如果进程没有控制终端，则无法确定控制终端的路径名，或者出现其他错误，那么 **ctermid()** 返回空字符串。

对于多线程应用程序，如果 *s* 是一个空指针，则不能执行操作，并且返回空指针。

注释

ctermid() 与 **ttyname()** 之间的区别在于，必须向 **ttyname()** 传递文件描述符，它会返回与该文件描述符相关的终端的实际名称，而 **ctermid()** 返回引用终端的字符串 **(/dev/tty)**（如果该字符串用作文件名）。（请参阅 *ttyname*(3C) ）。因此，仅当进程已至少向终端打开一个文件时， **ttyname()** 才有用。

另请参阅

ttyname(3C)、 thread_safety(5)。

符合的标准

ctermid()：AES、SVID2、SVID3、XPG2、XPG3、XPG4、FIPS 151-2 和 POSIX.1

名称

ctime()、 ctime_r()、 asctime()、 asctime_r()、 daylight()、 difftime()、 gmtime()、 gmtime_r()、 localtime()、 localtime_r()、 mktime()、 timezone()、 tzname()、 tzset() - 将日期和时间转换为字符串

概要

```
#include <time.h>

char *asctime(const struct tm *timeptr);

char *asctime_r(const struct tm *timeptr, char *buffer);

char *ctime(const time_t *timer);

char *ctime_r(const time_t *timer, char *buffer);

double difftime(time_t time1, time_t time0);

struct tm *gmtime(const time_t *timer);

struct tm *gmtime_r(const time_t *timer, struct tm *result);

struct tm *localtime(const time_t *timer);

struct tm *localtime_r(const time_t *timer, struct tm *result);

time_t mktime(struct tm *timeptr);

extern long timezone;

extern int daylight;

extern char *tzname[2];

void tzset(void);
```

说明

asctime() 对包含在 *timeptr* 所指向的结构中包含的分解时间进行转换，并按以下格式返回指向 26 个字符的字符串指针：

Sun Sep 16 01:03:52 1973\n\0

所有字段的宽度都是固定的。

如果 **timeptr** 中的 *tm_year* 小于 0 或大于 8099，则 **asctime()** 返回 NULL，并将 errno 设置为 [ERANGE] 。在 32 位和 64 位 HP-UX 上， **asctime()** 支持的最小日期为 January 1 00:00:00 1900， **asctime()** 支持的最大日期为 December 31 23:59:59 9999。

asctime_r() 与 **asctime()** 类似，不同之处在于，如果成功，它会将结果放入用户提供的 **buffer** ，并返回指向 **buffer** 的指针。需要长度至少为 26 的缓冲区。

ctime() 转换 *timer* 指向的日历时间，表示自起始参考时间以来的时间（以秒为单位），并以字符串形式返回一个指向本地时间的指针。等价于：

asctime(localtime(timer))

在 32 位和 64 位 HP-UX 上， **ctime()** 支持的最小日期为 Friday December 13 20:45:52 UTC 1901。在 32 位和 64 位 HP-UX 上， **ctime()** 支持的最大日期分别为 Tuesday January 19 03:14:07 UTC 2038 和 Friday December 31 23:59:59 UTC 9999。

在 64 位 HP-UX 上，如果计时器小于最小支持日期（也就是 **limits.h** 中定义的 **INT_MIN**）对应的秒数，或者超过最大支持日期对应的秒数，则 **ctime()** 返回 NULL，并将 errno 设置为 [ERANGE] 。

ctime_r() 与 **ctime()** 类似，不同之处在于，如果成功，它会将结果放入用户提供的 **buffer** ，并返回指向 **buffer** 的指针。需要长度至少为 26 的缓冲区。

difftime() 返回两个日历时间的秒差： *time1* - *time0* 。

gmtime() 直接转换为国际标准时间 (UTC)，即 HP-UX 操作系统使用的时间标准。 **gmtime()** 将返回指向下述 **tm** 结构的指针。

在 32 位和 64 位 HP-UX 上， **gmtime()** 支持的最小日期为 Friday December 13 20:45:52 UTC 1901。在 32 位和 64 位 HP-UX 上， **gmtime()** 支持的最大日期分别为 Tuesday January 19 03:14:07 UTC 2038 和 Friday December 31 23:59:59 UTC 9999。

在 64 位 HP-UX 上，如果计时器小于最小支持日期（也就是 **limits.h** 中定义的 **INT_MIN**）对应的秒数，或者超过最大支持日期对应的秒数，则 **gmtime()** 返回 NULL，并将 errno 设置为 ERANGE。

gmtime_r() 与 **gmtime()** 类似，不同之处在于，如果成功， **gmtime_r()** 将结果存储在 **result** 指向的 **tm struct** 中，并返回 **result** 。

localtime() 根据 **TZ** 环境变量（请参阅下文“环境变量”）的内容，针对时区和任何夏时制时区调整（例如美国夏时制）进行更正。 **localtime()** 可以返回指向下述 **tm** 结构的指针。

在 32 位和 64 位 HP-UX 上， **localtime()** 支持的最小日期为 Friday December 13 20:45:52 UTC 1901。在 32 位和 64 位 HP-UX 上， **localtime()** 支持的最大日期分别为 Tuesday January 19 03:14:07 UTC 2038 和 Friday December 31 23:59:59 UTC 9999。

在 64 位 HP-UX 上，如果计时器小于与最小支持日期（也就是 **limits.h** 中定义的 **INT_MIN**）对应的秒数，或者超过与最大支持日期对应的秒数，则 **localtime()** 返回 NULL，并将 errno 设置为 [ERANGE] 。

localtime_r() 与 **localtime()** 类似，不同之处在于，如果成功， **localtime_r()** 将结果存储在 **result** 指向的 **tm struct** 中，并返回 **result** 。

mktime() 将 *timeptr* 指向的结构中的分解时间（表示为本地时间）转换为一个日历时间值，该值与 **time**() 返回值具有相同的编码。将忽略该结构的 **tm_wday** 和 **tm_yday** 部分的原始值，并且不会将其他部分的原始值限制为以下指定的范围。

如果 **tm_isdst** 为正值或 0 值，按最初设置， **mktime()** 将分别假定夏时制在指定的时间有效或无效。如果 **tm_isdst** 为负值， **mktime()** 将尝试确定夏时制在指定的时间是否有效。

如果成功完成，将设置所有组件，以表示指定的日历时间，但是其值将强制为以下指定的范围。在确定 **tm_mon** 和 **tm_year** 之前，将不设置 **tm_mday** 的最终值。 **mktime()** 可以返回编码为类型 **time_t** 的值的指定日历时间。

如果无法表示日历时间，函数将返回值 (**time_t**) — 1 ，并将 **errno** 设置为 [ERANGE] 。请注意，值 (**time_t**) — 1 还与 1969 年 12 月 31 日 23:59:59 时间对应（加上或减去时区和夏日制调整）。因此，有必要同时检查返回值和 **errno** ，以可靠地检测错误情况。

tzset()

根据 **TZ** 环境变量的内容，设置外部变量 *timezone* 、 *daylight* 和 *tzname* 的值（与任何时间值无关）。函数 **localtime()** 、 **mktime()** 、 **ctime()** 、 **asctime()** 和 **strftime()** （请参阅 *strftime*(3C) ）可以调用 **tzset()** ，并将下述外部变量中返回的值用于其运算。用户也可以直接调用 **tzset()** 。

如果未设置环境变量 **TZ** ， **tzset()** 将会检查时区值的缺省文件 **/etc/default/tz** ，并根据检查结果设置时区值。文件 **/etc/default/tz** 包含未设置环境变量 **TZ** 时， **tzset()** 使用的时区值。此文件的格式与不带前缀 **TZ=** 的 **TZ** 格式相同。有关 **TZ** 的格式，请查阅 *environ*(5) 。

如果使用环境变量 **TZ** 或文件 **/etc/default/tz** 无法确定 *timezone* 的值，则将其设置为缺省值 *EST5EDT* 。如果 *timezone* 设置为缺省值 *EST5EDT* ，且时区调整文件不可用，则将 *timezone* 设置为缺省值 *UTC* （国际标准时间）。

可以修改 **/etc/default/tz** ，使之成为时区的相应缺省值。

<time.h> 头文件包含所有相关函数和外部变量的声明。此外还包含 **tm** 结构，该结构包括以下成员：

```
int tm_sec;     /* seconds after the minute - [0,61] */
int tm_min;     /* minutes after the hour - [0,59] */
int tm_hour;    /* hours - [0,23] */
int tm_mday;    /* day of month - [1,31] */
int tm_mon;     /* month of year - [0,11] */
int tm_year;    /* years since 1900 */
int tm_wday;    /* days since Sunday - [0,6] */
int tm_yday;    /* days since January 1 - [0,365] */
int tm_isdst;   /* daylight savings time flag */
```

如果夏时制时区调整（例如夏时制）有效，则 **tm_isdst** 的值是正数；如果夏时制时区调整无效，则该值为 0；如果未提供信息，则该值为负数。

外部变量 **timezone** 包含 UTC 与本地标准时间之间的秒差（例如，在美国东部时区 (EST)， **timezone** 为 5*60*60）只有当 **TZ** 环境变量中指定了夏时制时区调整时，外部变量 **daylight** 才不是非零值。外部变量 **tzname[2]** 包含 **TZ** 环境变量指定的本地标准和本地夏时制时区的缩略形式。

外部语言环境影响

语言环境

LC_CTYPE 类别可确定将 *format* 内的字节解释为单字节字符和（或）多字节字符。

环境变量

tzset() 函数使用 **TZ** 的内容来设置外部变量 **timezone** 、 **daylight** 和 **tzname** 。 **TZ** 还可确定针对 **%Z** 和 **%z** 指令需要替换的时区名称，以及 **localtime()** 、 **mktime()** 和 **ctime()** 执行的时区调整。 *environ*(5) 中介绍了在 **TZ** 内指定时区的两种方法。

国际代码集支持

支持单字节字符代码集和多字节字符代码集。

返回值

对于 **asctime_r()** 和 **ctime_r()** ，如果缓冲区的长度不足，将返回 NULL，并将 errno 设置为 [EINVAL] 。

对于 **asctime_r()** 、 **ctime_r()** 、 **gmtime_r()** 和 **localtime_r()** ，如果以参数的形式传入了空指针，则返回 NULL，并将 errno 设置为 [EINVAL] 。

如果对以下例行程序的输入不在支持的范围内，则返回 NULL，并将 errno 设置为 [ERANGE] ： **asctime()** 、 **asctime_r()** 、 **ctime()** 、 **ctime_r()** 、 **gmtime()** 、 **gmtime_r()** 、 **localtime()** 、 **localtime_r()** 。

实际应用信息

asctime() 、 **ctime()** 、 **gmtime()** 和 **localtime()** 的返回值指向每次调用后其内容将被覆盖的静态数据。

警告

asctime_r() 、 **ctime_r()** 、 **gmtime_r()** 和 **localtime_r()** 的用户还应注意，现在这些函数符合 POSIX.1c。为了与现有的 DCE 应用程序兼容，还支持这些函数的早期原型。

tm_sec([0,61]) 的范围可以扩展到 61，以支持不经常出现的一次或两次闰秒。但是，由 **time()** 返回的并作为 *timer* 参数传递的“seconds since the Epoch”值不包括累积的闰秒。 **localtime()** 和 **gmtime()** 生成的 **tm** 结构永远不会反映任何闰秒。如果成功完成， **mktime()** 将 **tm_sec** 组件的值强制设置在范围 [0,59] 内。

作者

ctime() 由 AT&T 和 HP 联合开发。

另请参阅

time(2) 、 getdate(3C) 、 setlocale(3C) 、 strftime(3C) 、 tztab(4) 、 environ(5) 、 lang(5) 、 langinfo(5) 、 thread_safety(5)。

符合的标准

ctime()：AES、SVID2、SVID3、XPG2、XPG3、XPG4、FIPS 151-2、POSIX.1、ANSI C

asctime()：AES、SVID2、SVID3、XPG2、XPG3、XPG4、FIPS 151-2、POSIX.1、ANSI C

daylight：AES、SVID2、SVID3、XPG2、XPG3、XPG4

difftime()：AES、SVID3、XPG4、ANSI C

gmtime(): AES、SVID2、SVID3、XPG2、XPG3、XPG4、FIPS 151-2、POSIX.1、ANSI C

localtime(): AES、SVID2、SVID3、XPG2、XPG3、XPG4、FIPS 151-2、POSIX.1、ANSI C

mktime(): AES、SVID3、XPG3、XPG4、FIPS 151-2、POSIX.1、ANSI C

timezone: AES、SVID3、XPG2、XPG3、XPG4

tzname: AES、XPG2、XPG3、XPG4、FIPS 151-2、POSIX.1

tzset(): AES、XPG2、XPG3、XPG4、FIPS 151-2、POSIX.1

asctime_r()、 **ctime_r()**、 **localtime_r()**、 **gmtime_r()**: POSIX.1c

名称

ctype : isalnum()、 isalpha()、 isascii()、 isblank()、 iscntrl()、 isdigit()、 isgraph()、 islower()、 isprint()、 ispunct()、 isspace()、 isupper()、 isxdigit() - 根据类型对字符进行分类

概要

#include <ctype.h>

int isalnum(int *c***);**
int isalpha(int *c***);**
int isascii(int *c***);**
int isblank(int *c***);**
int iscntrl(int *c***);**
int isdigit(int *c***);**
int isgraph(int *c***);**
int islower(int *c***);**
int isprint(int *c***);**
int ispunct(int *c***);**
int isspace(int *c***);**
int isupper(int *c***);**
int isxdigit(int *c***);**

说明

根据上次成功调用 **setlocale()** （请参阅 *setlocale*(3C) ）后确定的编码字符集的规则，这些函数将对已编码的字符整数值进行分类。其中每个函数都是一个谓词，结果为 true 时返回非零值，结果为 false 时返回零。

如果调用 **setlocale()** 失败，将根据缺省的 ASCII 7 位编码字符集（请参阅 *setlocale*(3C) ）规则对字符进行分类。

isascii() 的定义适用于所有整数值；其他函数的定义适用于范围 **-1** (EOF) 到 **255** 。

返回值

出现以下情况时，这些函数将返回非零值；否则返回零：

isalnum(c)	*c* 是一个字母数字字符（字母或数字）。
isalpha(c)	*c* 是一个字母。
isascii(c)	*c* 是介于 0 至 0177 之间（包括边界值）的任何 ASCII 字符。
isblank(c)	*c* 是一个空白字符；即空格或制表符。
iscntrl(c)	*c* 是一个控制字符（ASCII 形式：小于 040 的字符代码及删除字符 (0177)）。
isdigit(c)	*c* 是一个十进制数字（ASCII 形式：字符 [0-9]）。
isgraph(c)	*c* 是一个可见字符（ASCII 形式：除空格字符 (040) 以外的可输出字符）。

islower(c)	*c* 是一个小写字母。
isprint(c)	*c* 是一个输出字符。
ispunct(c)	*c* 是一个标点字符（ASCII 形式：除空格字符 (040)、数字、字母以外的任何可输出字符）。
isspace(c)	*c* 字符可在显示文本中创建空白符（ASCII 形式：空格、制表符、回车符、换行符、纵向制表符和换页符）。
isupper(c)	*c* 为大写字母。
isxdigit(c)	*c* 是一个十六进制数字（ASCII 形式：字符 [0-9]、[A-F] 或 [a-f]）。

如果其中任何函数的参数超出该函数的域，则结果是不确定的。

外部语言环境影响

环境变量

LC_CTYPE 类别确定字符类型的分类。

国际代码集支持

支持单字节字符代码集。

警告

这些函数既可以作为库函数提供，也可以作为 **<ctype.h>** 头文件中定义的宏提供。通常使用宏版本。要获取库函数，可以使用 **#undef** 删除宏定义，如果是在 ANSI -C 模式下编译的，也可以使用圆括号括起函数名，或者使用函数的地址。以下示例将使用 **isalpha()** 、 **isdigit()** 和 **isspace()** 的库函数：

```
#include <ctype.h>
#undef isalpha
    ...
main()
{
  int (*ctype_func)();
    ...
  if ( isalpha(c) )
    ...
  if ( (isdigit)(c) )
    ...
  ctype_func = isspace;
    ...
}
```

作者

ctype() 由 IBM、OSF 和 HP 联合开发。

另请参阅

setlocale(3C)、 ascii(5)、 thread_safety(5)。

符合的标准

isalnum(): AES、SVID2、SVID3、XPG2、XPG3、XPG4、FIPS 151-2、POSIX.1、ANSI C

isalpha(): AES、SVID2、SVID3、XPG2、XPG3、XPG4、FIPS 151-2、POSIX.1、ANSI C

isascii(): AES、SVID2、SVID3、XPG2、XPG3、XPG4

isblank(): ISO/IEC 9899:1999 (C99), UNIX 03

iscntrl(): AES、SVID2、SVID3、XPG2、XPG3、XPG4、FIPS 151-2、POSIX.1、ANSI C

isdigit(): AES、SVID2、SVID3、XPG2、XPG3、XPG4、FIPS 151-2、POSIX.1、ANSI C

isgraph(): AES、SVID2、SVID3、XPG2、XPG3、XPG4、FIPS 151-2、POSIX.1、ANSI C

islower(): AES、SVID2、SVID3、XPG2、XPG3、XPG4、FIPS 151-2、POSIX.1、ANSI C

isprint(): AES、SVID2、SVID3、XPG2、XPG3、XPG4、FIPS 151-2、POSIX.1、ANSI C

ispunct(): AES、SVID2、SVID3、XPG2、XPG3、XPG4、FIPS 151-2、POSIX.1、ANSI C

isspace(): AES、SVID2、SVID3、XPG2、XPG3、XPG4、FIPS 151-2、POSIX.1、ANSI C

isupper(): AES、SVID2、SVID3、XPG2、XPG3、XPG4、FIPS 151-2、POSIX.1、ANSI C

isxdigit(): AES、SVID2、SVID3、XPG2、XPG3、XPG4、FIPS 151-2、POSIX.1、ANSI C

名称

curscr — 当前窗口

概要

#include <curses.h>

extern WINDOW *curscr;

说明

外部变量 *curscr* 指向内部数据结构。可以将其指定为某些函数的参数，例如此规范中允许的 **clearok()** 。

另请参阅

clearok(3X)、 <curses.h>。

历史变更记录

在 X/Open Curses 第 4 期中首次发布。

名称

curses_intro - 终端和打印机处理和优化程序包

说明

接口的使用和实现

这些例行程序提供了一种方法，用于以独立于终端的方式更新屏幕，并使之合理优化。下面每个语句都适用，除非在随后的详细说明中另行明确指定。如果函数的参数具有无效值（例如，值位于函数域之外，或指针位于程序地址空间之外，或为空指针），则行为是未定义的。在头文件中声明的任何函数也可以作为该头文件中定义的宏来实现，因此，如果包含库函数的头文件，则不应显式声明该库函数。通过将函数名称括在括号中，可在本地禁止使用函数的任何宏定义，因为这样该名称就不会后跟表示扩展宏函数名称的左括号。出于同样的语义原因，允许采用库函数的地址，即使它也定义为宏也是如此。使用 C 语言 **#undef** 结构来删除任何这样的宏定义，还将确保引用实际函数。作为宏实现的库函数的任何调用将扩展为仅一次评估其每个参数的代码，如有必要，则通过括号进行完全保护，因此将任意表达式用作参数通常是安全的。同样地，如果可在表达式的任何位置调用具有兼容返回类型的函数，那么，则可以调用下面几节描述的类似于函数的宏。

假定不用引用头文件中定义的任何类型即可声明库函数，则也将允许显式或隐式地声明函数，并允许在不包括其相关头文件的情况下使用该函数。如果未声明接受可变数量参数的函数（以显式方式或者通过包括其相关头文件），则行为是未定义的。

由于该版本的《Curses 规范》中引入了一些变更，应用程序编写人员只需包括最少数量的头文件即可。 XSI 兼容系统的实现将使所有必要的符号均可见，如本文档“头文件”一节所述。

C 语言定义

作为本文档中概要和代码示例的基础的 C 语言是 **ISO C** ，它在所引用的 ISO C 标准中定义。 **C** 语言常见用法指的是标准化之前的 C 语言，它是该规范以前版本的基础。

编译环境

应用程序应确保在包括任何头文件之前定义功能测试宏 **_XOPEN_SOURCE** 。该操作对于启用本文档中描述的功能是必需的，对于启用“常见应用程序环境”中其他位置定义的功能可能也需要。

_XOPEN_SOURCE 宏可以由编译过程自动定义，但是为了确保最大的可移植性，应用程序应确保在任何 **#include** 指令之前，使用编译程序选项或源文件中的 **#define** 指令定义 **_XOPEN_SOURCE** 。本文档中的标识符仅可使用 **#undef** 指令设置为未定义。这些 **#undef** 指令必须位于任何头文件的所有 **#include** 指令之后。

极其严格地遵循 POSIX 和 ISO C 标准的应用程序，将在符合该规范的系统上编译。但是，由于此处所述目的之外的任何目的，而使用任何标记为 POSIX 和 ISO C 扩展的项目的应用程序则不能编译。这种情况下，可能必须更改这些应用程序才能使用备用标识符。

因为本文档遵守 ISO C 标准，并且通过设置为 2 的 **_POSIX_C_SOURCE**

启用的所有功能都应由 **_XOPEN_SOURCE** 启用，所以，如果定义了 **_XOPEN_SOURCE** ，则应该无须定义 **_POSIX_SOURCE** 或 **_POSIX_C_SOURCE** 。因此，如果定义了 **_XOPEN_SOURCE** ，并定义了 **_POSIX_SOURCE** ，或者将 **_POSIX_C_SOURCE** 设置为 1 或 2，则行为与只定义 **_XOPEN_SOURCE** 相同。

但是，如果将 **_POSIX_C_SOURCE** 设置为大于 2 的值，则行为是未定义的。

c89 和 **cc** 实用程序可识别标准库附加的 **-l** 操作数：

-l curses 此操作数可使该规范中引用的所有库函数可见（但是那些标记为“增强 CURSES”的库函数以及带有 EC 旁注的部分除外）。

如果实现定义了 **_XOPEN_CURSES**，并且应用程序定义了 **_XOPEN_SOURCE_EXTENDED** 功能测试宏，则 **-l curses** 也会使该规范中引用的所有库函数以及标记为“增强 CURSES”的库函数可见。

使用任何指定为“增强 CURSES”的 API 的应用程序必须在每个源文件中定义 **_XOPEN_SOURCE_EXTENDED = 1**，或者将其定义为其编译环境的一部分。如果在源文件中定义 **_XOPEN_SOURCE_EXTENDED = 1**，则必须在包括任何头文件之前进行该操作。

如果实现支持“Curses 规范”中标记为 开发的实用程序，则 **lint** 实用程序可识别标准库附加的 **-l curses** 操作数：

-l curses 命名库 **llib-lcurses.ln**，该库将包含本文档中指定的函数。

未指定库 **llib-lcurses.ln** 是否作为常规文件存在。

X/Open 命名空间（增强 CURSES）

本节中的要求仅对声明符合增强 Curses 的实现有效。

本文档中的所有标识符至少在 **<curses.h>**、**<term.h>** 和 **<unctrl.h>** 头文件之一中定义。如果定义了 **_XOPEN_SOURCE**，则每个头文件定义或声明一些标识符，这些标识符可能会与应用程序使用的标识符冲突。应用程序可见的标识符集合刚好由以下标识符组成：所包括头文件的头文件页中的标识符，以及为实现保留的附加标识符。另外，某些头文件还可以使其他头文件中的标识符可见，如相关头文件页中所指定。

保留供实现使用的标识符如下所述。

1. 如果包括了头文件，则头文件部分中描述的每个带有外部链接的标识符将保留作为带有外部链接的标识符使用。
2. 如果包括了头文件，则头文件部分中描述的每个宏名称将保留用于任何用途。
3. 如果包括了头文件，则头文件部分中描述的每个带有文件范围的标识符将保留作为同一命名空间中带有文件范围的标识符使用。
4. 所有标识符刚好由两个大写字母组成。

如果包括下表中的任何头文件，则带有所显示前缀、后缀或完整名称的标识符将保留供实现使用。

头文件	前缀	后缀	完整名称
<curses.h>	**add**、 **attr**、 **get**、 **in**、 **mousr**、 **mv**、 **scr**、 **slk**、 **un**、 **wadd**、 **wattr**、 **wbkg**、 **win**		
任何头文件		**_t**	

如果包括了下表中的任何头文件，则可以定义带有所显示前缀的宏。在上次包括了给定头文件之后，应用程序可以根据其自己的使用情况，使用带有对应前缀的标识符，前提是要以对应宏的 **#undef** 开头。

头文件	前缀
<curses.h>	**A_**、 **ACS_**、 **ALL_**、 **BUTTON**、 **COLOR_**、 **KEY_**、 **MOUSE**、 **REPORT_**、 **WA_**、 **WACS_**
<term.h>	**ext_**

无论是否包括头文件，都将保留下列标识符：

1 所有以下划线和大写字母或其他下划线开头的标识符，始终保留供实现使用。

2 所有以下划线开头的标识符，始终保留作为普通标识符和标记命名空间中带有文件范围的标识符使用。

3 《X/Open System Interfaces and Headers, Issue 4, Version 2》规范中列为保留的所有标识符，将保留作为带有外部链接的标识符使用。

本文档中定义的带有外部链接的所有标识符，始终保留作为带有外部链接的标识符使用。

其他任何标识符均不保留。

应用程序不得声明或定义与同一环境中保留的标识符同名的标识符。因为无论范围和命名空间如何，只要发现宏名称就会将其替换，所以，如果包括了任何相关头文件，则不得定义与任何保留标识符名称相匹配的宏名称。

可以按任意顺序包括头文件，并且在给定范围内可以多次包括每个头文件，其效果与只包括一次并没有什么区别。

如果使用头文件，则该头文件必须包括在任何外部声明或定义之外，并且必须在第一次引用它所定义的任何类型或宏之前，或者它所声明的任何函数或对象之前，先包括该头文件。但是，如果在多个头文件中声明或定义了标识符，则可以在初始引用该标识符之后包括第二个和后续的相关头文件。在包括头文件之前，程序不得定义其名称在词汇上与该头文件定义的符号完全相同的任何宏。

作为宏实现的接口（增强 **CURSES**）

本节中的要求仅对声明符合增强 Curses 的实现有效。

必须将下列带参数的接口作为宏来实现。与应用程序编程人员相关的内容是：不能在这些参数之前使用 **&** 字符。

宏	联机帮助页
COLOR_PAIR()、**PAIR_NUMBER()**	*can_change_color*(3X)
getbegyx()、**getmaxyx()**、**getparyx()**、**getyx()**	*getbegyx*(3X)

<curses.h> 、 **<term.h>** 和 **<unctrl.h>** 中的说明列出了其他宏，如 **COLOR_BLACK** ，这些宏不采用参数。

与《**X/Open System Interfaces and Headers, Issue 4, Version 2**》规范的关系

错误编号

大多数函数在 **errno** 中提供错误编号，它是在 **<errno.h>** 中定义的变量或宏；该宏将扩展为一个类型为 **int** 的可修改的 **lvalue** 。

errno 的有效值列表，以及为应用程序编程人员提供的有关使用 **errno** 的建议，位于《X/Open System Interfaces and Headers, Issue 4, Version 2》规范中。

数据类型

Curses 函数使用的所有数据类型均由实现定义。下表描述了这些类型：

attr_t 整数类型，至少可包含一个 **unsigned short** 。类型 **attr_t** 用于包含 **<curses.h>** 中定义的以前缀 **WA_** 开头的属性进行了 OR 运算之后得到的属性集。

bool 布尔数据类型

chtype 整数类型，至少可包含一个 **unsigned**char 和多个属性。类型为 **chtype** 的值由 **unsigned**-char 值以及 **<curses.h>** 中定义的带有 **A_** 前缀的零个或多个基本属性标记进行 OR 运算之后所形成。为此，应用程序可以使用 **<curses.h>** 中定义的基本掩码来提取 **chtype** 值的这些组成部分。

chtype 数据类型还包含一个颜色对。类型为 **chtype** 的值由 **unsigned**char 值、一个颜色对，以及 **<curses.h>** 中定义的以 **A_** 前缀开头的零个或多个属性进行 OR 运算之后所形成。为此，应用程序可以使用 **<curses.h>** 中定义的掩码来提取 **chtype** 值的这些组成部分。

SCREEN 不透明终端表示法。

wchar_t 如 **<stddef.h>** 中所述。

cchar_t 该类型可以引用宽字符字符串（最长可达与实现相关的长度）、一个颜色对，以及本文档中定义的所有属性集中的零个或多个属性。空 **cchar_t** 对象是引用了空宽字符字符串的对象。**cchar_t** 对象的数组以空的 **cchar_t** 对象终止。

WINDOW 不透明窗口表示法。

接口概述

组件

Curses 初始化函数（通常为 **initscr()** ）通过引用参数或环境变量来确定正在使用的终端型号。如果 **terminfo** 中定义了该型号，则同一个 **terminfo** 条目会告知 Curses 具体如何操作该终端。

在这种情况下，通过使用综合的 API，应用程序可以执行终端操作。 Curses 运行时系统接收每个终端请求，并向该终端发送相应的命令以便获得所需的结果。

与《**X/Open System Interface Definitions, Issue 4, Version 2**》规范的关系

使用 Curses 的应用程序不应使用《X/Open System Interface Definitions, Issue 4, Version 2》规范第 9 章“General Terminal Interface”中定义的通用终端接口的功能来控制终端。

在 Curses 环境之外执行 Curses 模式时是否有范例并没有要求（请参阅 *def_prog_mode*(3X) 页）。

与信号的关系

如果在调用 **initscr()** 时（请参阅 *initscr*(3X) 页），SIGINT、SIGQUIT 和 SIGTSTP 信号的配置为 SIGDFL，则 Curses 实现可以提供这些信号的特殊处理。

在进程生命周期内或者在进程更改信号配置之前，对这些信号的任何特殊处理可以一直保持有效。

对于信号来说，不要求 Curses 函数是安全的（请参阅《X/Open System Interfaces and Headers, Issue 4, Version 2》规范中的 *sigaction*(2) ）。

未指定《X/Open System Interfaces and Headers, Issue 4, Version 2》规范中未定义与信号有关的 Curses 行为。

屏幕、窗口和终端

屏幕

屏幕是终端的物理输出设备。在 Curses 中， **SCREEN** 数据类型是与终端相关联的不透明数据类型。每个窗口（请参阅下文）都与 **SCREEN** 相关联。

窗口

Curses 函数允许处理 **window** 对象，这些对象可以被看作字符及其呈现方式的二维数组。提供的缺省窗口名为 **stdscr** ，其大小为终端屏幕的大小。其他窗口可以使用 **newwin()** 进行创建。

声明为 **WINDOW *** 的变量指的是窗口（以及子窗口、派生窗口、面板，如下所示）。这些数据结构是使用 terminfo(4) 中的参考联机帮助页所述的函数处理的。其中最基本的函数是 **move()** 和 **addch()** 。将包括这些函数的更通用形式，进程可以通过其指定窗口。

使用函数处理窗口之后，将调用 **refresh()** ，以便告知 Curses 使 CRT 屏幕与 **stdscr** 类似。

线描字符可指定为输出。在输入时，Curses 还能够将传输转义序列的箭头键和功能键转换为单个值。线描字符和输入值使用 **<curses.h>** 中定义的名称。

每个窗口都有一个标记，表示该窗口中的信息可能与终端设备上显示的信息不同。要对窗口中的内容进行任何更改、移动或修改窗口，或者设置窗口的光标位置，需要设置该标记（对窗口进行 **touches** 处理）。

子窗口

子窗口是在另一个窗口（称为 父窗口）中创建的窗口，并相对于父窗口进行定位。可以通过调用 **derwin()** 、**newpad()** 或 **subwin()** 创建子窗口。对子窗口所做的更改不会影响其父窗口。

可以通过调用 **subwin()** 在父窗口中创建子窗口。子窗口在屏幕上的位置和大小必须与父窗口相同，或者完全包含在父窗口中。对父窗口或子窗口所做的更改对两者都有影响。窗口裁剪不是子窗口的属性。

祖先

ancestor 一词指的是窗口的父窗口，或父窗口的父窗口，依此类推。

派生窗口

派生窗口是子窗口，其位置通过参照父窗口定义，而不是以绝对屏幕坐标定义。派生窗口在其他方面与子窗口完全相同。

面板

面板是子窗口的一个特殊情况，它不一定与屏幕的可视部分相关联。处理面板的函数全部在 **newpad()** 中进行了讨论。

终端

终端是逻辑输入和输出设备，基于字符的应用程序通过其与用户进行交互。 **TERMINAL** 是与终端相关联的一种不透明数据类型。 **TERMINAL** 数据结构主要包含有关终端功能的信息，这些信息由 **terminfo** 定义。 **TERMINAL** 还包含有关终端模式以及输入和输出操作的当前状态的信息。每个屏幕（请参阅上文）都与 **TERMINAL** 相关联。

字符

字符存储大小

一直以来，屏幕上的某个位置都与单个存储的字节相对应。由于下面几个原因，这种对应关系不再存在:

- 某些字符在屏幕上显示时可能会占用若干列（请参阅“多列字符”一节）。
- 某些字符可能是不带空格的字符，仅与带空格字符关联时才定义（请参阅“不带空格字符（增强 CURSES）”一节）。
- 保存扩展字符集中字符所用的字节数取决于 **LC_CTYPE** 语言环境类别。

字符和呈现方式的内部存储格式均未指定。在类型为 **chtype** 和 **cchar_t** 的对象中，字符和呈现方式的内部存储格式和外部表示法之间没有隐含的对应关系。

多列字符

某些字符集定义 多列字符，这些字符在屏幕上显示时会占用多列位置。

写入其宽度大于目标窗口宽度的字符是一种错误。

属性

每个字符显示时可以带有 属性，如在支持这些显示增强功能的终端上显示下划线、反白显示或显示彩色。窗口的当前属性将应用于使用 **waddch()** 、 **wadd_wch()** 、 **waddstr()** 、 **waddchstr()** 、 **waddwstr()** 、 **waddwchstr()** 和 **wprintw()** 写入该窗口的所有字符。属性可以组合使用。

使用 **<curses.h>** 中指定的带有 **A_** 前缀的常量可以指定属性。 **A_** 常量可处理类型为 **chtype** 的对象中的属性。使用带有 **WA_** 前缀的常量，可以指定其他属性。 **WA_** 常量可处理类型为 **attr_t** 的对象中的属性。

以 **A_** 和 **WA_** 开头的两种常量以及表示相同终端功能的常量，指的是 **terminfo** 数据库中和窗口数据结构中的相同属性。窗口中的效果不会随应用程序是指定了 **A_** 常量还是 **WA_** 常量而有所区别。例如，当应用程序使用支持 **A_** 值的接口更新窗口属性时，使用返回 **WA_** 值的函数对窗口属性进行查询会反映该更新。当它使用支持 **WA_** 值（存在相对应的 **A_** 值）的接口更新窗口属性时，使用返回 **A_** 值的函数对窗口属性进行查询会反映该更新。

显示形式

屏幕上所显示的字符的 显示形式表示其属性和颜色对。

输出到屏幕的字符的显示形式将成为该字符的属性，并通过任何滚动和插入（或删除）行（或字符）操作与字符一起移动。在特定终端的可能范围内，字符的显示形式与输出到屏幕的该字符的图形显示形式相对应。

如果特定终端不支持应用程序要尝试使用的呈现方式，则 Curses 可能会将其替换为不同的呈现方式。

颜色总是成对使用的（称为颜色对）。颜色对由一种前景颜色（用于字符）和一种背景颜色（用于显示字符的字段）组成。

不带空格字符

本节中的要求仅对声明符合增强 Curses 的实现有效。

某些字符集可能包含 不带空格字符（不带空格字符是 **wcwidth()** 返回的宽度为零的字符）。应用程序可以将不带空格字符写入窗口中。窗口中的每个不带空格字符都与带空格字符相关联，并且会修改带空格字符。不能单独处理窗口中的不带空格字符。每当 Curses 操作影响与不带空格字符相关联的带空格字符时，就将隐式地处理该不带空格字符。

不带空格字符不支持属性。对于使用宽字符和属性的接口，如果宽字符是不带空格字符，则会忽略属性。对于所有列，多列字符仅有一个属性集。通过使用宽字符接口，应用程序可以控制不带空格字符与带空格字符之间的关联。宽字符字符串函数提供了与代码集相关的关联。

与名为 *c* 的带空格字符关联的不带空格字符具有如下两个典型作用:

- 不带空格字符可以修改 *c* 的外观（例如，可能会存在向字符添加区别标记的不带空格字符。不过，可能还会有一些具有内置区别标记的带空格字符）。
- 不带空格字符可以将 *c* 与 *c* 后面的字符连接起来（这种用法的示例包括连字符构成，以及字符向复合显示形式、词或表意文字的转换）。

实现可能会限制可与带空格字符相关联的不带空格字符的数量，前提是任何限制值至少为 **5**。

复合字符

复合字符是相关字符的集合，其中可能包括带空格字符，也可能包括与之相关联的任何不带空格字符。 带空格复合字符是后接与之相关联的不带空格字符的带空格字符。即，带空格复合字符是包括一个带空格字符的复合字符。具有复合字符的代码集示例为 ISO/IEC 10646-1:1993。

复合字符可以输出到屏幕；如果复合字符不包括带空格字符，则所有不带空格字符将与位于指定屏幕位置的带空格复合字符相关联。当应用程序从屏幕读回信息时，它将获取带空格复合字符。

cchar_t 数据类型表示复合字符及其呈现方式。当 **cchar_t** 表示不带空格的复合字符时（即，复合字符中没有空格字符），则不使用其呈现方式；当将该字符输出到屏幕时，它将使用已显示的带空格字符指定的呈现方式。

类型为 **cchar_t** 的对象可以使用 **setcchar()** 进行初始化，其内容可以通过使用 **getcchar()** 进行提取。如果应用程序提供的 **cchar_t** 值不是通过这种方式初始化的，或者是从带有 **cchar_t** 输出参数的 Curses 函数获取的，则采用 **cchar_t** 输入参数的函数的行为未定义。

窗口属性

下面列出了与每个窗口关联的属性，这些属性影响将字符输出到窗口的方式（请参阅“输出到窗口中的字符的呈现方式”）。

窗口呈现方式

每个窗口都有呈现方式，它与如下所述的窗口背景属性的呈现方式组成部分是不同的。

窗口背景

每个窗口都有背景属性。背景属性指定：

- 从屏幕删除可见信息的各种情况下将使用带空格复合字符（背景字符）。
- 在上述情况下以及“输出到窗口中的字符的呈现方式”中指定的其他情况下，显示背景字符时要使用的呈现方式。

概念操作

屏幕寻址

很多 Curses 函数都使用坐标对。在“说明”一节中，坐标位置表示为 (*y*, *x*)，因为 *y* 参数在函数调用中总是位于 *x* 参数的前面。这些坐标表示行（或列）的位置，而不是字符位置。

坐标 *y* 总是表示（窗口的）行，*x* 总是表示列。第一行和第一列为数字 0，而不是 1。位置 (0, 0) 是窗口的 原点。

例如，对于显示 ISO 8859-1 字符集（写入方向为从左到右）的终端，(0, 0) 表示屏幕的左上角。

以 **mv** 开头的函数采用指定 (*y*, *x*) 位置的参数，并在执行所请求的操作之前移动光标（就好像调用了 **move()** 一样）。作为所请求操作的一部分，将产生更多在各自的参考联机帮助页上指定的光标移动。

基本字符操作

添加（覆盖）

包含 add 一词的 Curses 函数（如 **addch()**），实际上了指定一个或多个字符，以替换（覆盖）窗口中已有的字符。如果这些函数仅指定不带空格字符，则将它们附加到窗口中已有的带空格字符的后面；另请参阅“不带空格字符（增强 CURSES）”一节。

使用需要更少列数的字符替换多列字符时，会从指定或暗示的列位置开始添加新字符。前一多列字符占用的而新字符不需要的所有列是 孤列，这些列将使用背景字符和呈现方式填写。

使用需要更多列的字符替换某个字符时，同时会替换该行中一个或多个后续字符。该过程也会产生孤列。

截断、换行和滚动

如果应用程序指定了字符或字符串，则将它们写入窗口将超出行尾（例如，如果应用程序尝试在一行的最后一列放入任何多列字符），具体行为取决于函数是否支持换行：

- 如果函数不支持换行，则上述操作失败。
- 如果函数支持换行，则它会从无法在原始行完全放下的第一个字符开始，在窗口中将一个或多个字符放在下一行的开始处。

  如果该行的最后一个字符是一个无法在该行完全放下的多列字符，则整个字符将换到下一行，而位于原始行末尾的列可能会成为孤列。

  如果原始行是窗口中的最后一行，则换行可能会导致进行滚动：

  — 如果启用了滚动，则进行滚动。窗口中第一行的内容将丢失。窗口中剩余每行的内容将移动到上一行。窗口的最后一行将使用换行的任何字符填充。最后一行中剩余的任何空间将使用背景字符和呈现方式填充。

  — 如果禁用了滚动，则超出最后一行的最后一列的任何字符都将被截断。

scrollok() 函数可启用或禁用滚动。

某些 **add** 函数会将光标移动到添加的最后一个字符的后面。如果该位置超出了行尾，则在上述第二个项目符号所指定的情况下会导致换行和滚动。

插入

插入函数（如 **insch()** ）可将字符紧接在指定或暗示的光标位置前面插入。

插入操作会将光标行中原来位于光标位置或光标位置之后的所有字符向该行的行尾方向移动。可能因此超出行尾的字符的处理取决于函数是否支持换行：

- 如果函数不支持换行，则将从窗口中删除这些字符。这样可能会产生孤列。
- 如果函数支持换行，则其结果与上面 截断部分所述的结果相同（但是以破折号为项目符号的最后一条中讨论的覆盖操作是插入）。

如果显示多列字符，则某些光标位置位于一个多列字符中，而不是某个字符的开头。任何在不是多列字符开头插入数据的请求都将进行调整，以便实际光标位置位于请求位置所在的多列字符的开头。

对于光标重定位没有任何警示。应用程序不应维护光标位置的映像，因为这样会导致在应用程序中放入终端特定信息，从而失去了使用 Curses 的意义。

可移植应用程序无法假定插入函数中指定的光标位置是实际光标位置的可重用标记。

删除

删除函数（如 **delch()** ）可删除指定或暗示的光标位置处的简单或复合字符，无论它是字符的哪一列。使用背景字符和呈现方式替换所有列位置，并且不对光标进行重定位。如果字符删除操作会导致以前的换行操作撤消，则结果未指定。

窗口操作

重叠窗口（即将一个窗口放在另一个窗口的上面）以及覆盖窗口（即将一个窗口的内容复制到另一个窗口中）按照沿多列字形边界覆盖多列字形的操作进行。所有孤列都按照字符操作中的方式进行处理。

跨越子窗口边界的字符

可以定义一个子窗口，使多列字符跨越子窗口边界。字符操作按照下列方式处理这些跨越字符：

- 使用 **in_wch()** 等函数读取子窗口，将同时读取整个跨越字符。
- 在子窗口中进行添加、插入或删除会在请求的操作开始之前删除整个跨越字符，并且不会重定位光标。
- 在子窗口中滚动行具有下列结果：
  - — 位于行首的跨越字符在滚动操作开始之前将被完全删除。
  - — 位于行尾的跨越字符将沿着滚动方向移动，并继续跨越子窗口边界。子窗口之外位于跨越字符以前位置的列位置将成为孤列，除非另一个跨越字符滚动到这些位置。

如果应用程序调用 **border()** 等函数，则不会发生上述情况，因为在子窗口中写入边界会删除所有跨越字符。

如果上述情况涉及多列字符，则限于子窗口的操作可以修改屏幕上位于该子窗口之外的部分。因此，保存子窗口，在子窗口中执行操作，然后恢复该子窗口可能会影响屏幕的外观。为了消除这些影响（例如，对于弹出窗口），应用程序应该刷新整个屏幕。

特殊字符

某些函数按照下列指定的方式处理特殊字符。

在无法基于窗口中放置的信息移动光标的函数中，这些特殊字符仅在字符串中使用，以便影响后续字符的放置；下面指定的光标移动在操作结束后，将不会停留在光标可见的位置。在移动光标的函数中，这些特殊字符可用于影响后续字符的放置，并且移动可见的光标。

退格符	除非光标已位于第 0 列中，否则退格符会将光标朝着当前行的行首方向移动一列，并从此处开始添加或插入退格符之后的任何字符。
回车符	除非光标已位于第 0 列中，否则回车符会将光标移动到当前行的行首，并从此处开始添加或插入回车符之后的任何字符。
换行符	在添加操作中，Curses 会将背景字符添加到连续列中，直至到达行尾为止。滚动的发生方式如“截断、换行和滚动”所述。换行符后面的任何字符将添加在新行的行首处。

在插入操作中，换行符会将光标移动到新行的行首（从而导致如“截断、换行和滚动”所述的滚动）。换行符后面的任何字符都将放置在新行的行首处。

filter() 函数可以禁止这种处理。

制表符　　文本中的 Tab 字符会将后面的字符移动到下一个水平制表位处。缺省情况下，制表位位于第 0 列、第 8 列、第 16 列，依此类推。

在插入或添加操作中，Curses 会分别将背景字符插入或添加到连续列中，直至到达下一个制表位为止。如果当前行中没有其他制表位，则将如“截断、换行和滚动”中所述，产生换行和滚动。

控制字符

从概念上来说，执行特殊字符处理的 Curses 函数会将控制字符转换为插入字符 (^)，后面跟着第二个字符（如果它是字母，则为大写字母），然后将该字符串写入窗口来代替控制字符。从窗口检索文本的函数不会检索原始的控制字符。

输出到窗口中的字符的呈现方式

当应用程序向窗口中添加或插入字符时，其结果如下所示：

如果字符不是空格字符，则窗口会收到：

- 应用程序指定的字符
- 应用程序指定的颜色；或窗口颜色（如果应用程序没有指定颜色）
- 指定的属性，已与窗口属性进行了 OR 运算。

如果字符是空格字符，则窗口会收到：

- 背景字符
- 应用程序指定的颜色；或背景颜色（如果应用程序没有指定颜色）
- 指定的属性，已与背景属性进行了 OR 运算。

输入处理

Curses 输入模型提供了从键盘获取输入的多种方法。

键盘处理

应用程序可以通过调用 **keypad()** 来启用或禁用 **keypad translation** 。如果启用了转换，则 Curses 会尝试将表示按功能键的终端输入序列转换为一个键代码。如果禁用了转换，则 Curses 会将终端输入传递到应用程序而不进行这种转换，表示按键盘键的输入的任何解释都必须由应用程序完成。

Curses 可以处理的键盘键的完整键代码集合由 **<curses.h>** 中定义的常量指定，其名称以 **KEY_** 开头。 **terminfo** 数据库中所述的每个终端类型可以支持部分或全部这些键代码。 **terminfo** 数据库指定终端类型中与每个键代码相对应的输入字符序列（请参阅 *terminfo*(4) 中的 **Keypad** ）。

对于按键不会传输唯一序列的终端， Curses 实现无法转换该终端的键盘键。

当启用了转换并且收到的字符可能是某个功能键（如 escape）的开头时，Curses 会记录时间，并开始累积字符。如果从收到第一个字符的时间开始的一个未指定间隔内，Curses 收到了表示按键盘键的附加字符，则 Curses 会将该输入转换为键代码，以便呈现到应用程序中。如果在此间隔内没有收到这样的字符，则不会转换该输入，各个字符将分别呈现到应用程序中。（因为 Curses 等待此间隔来累积键代码，所以很多终端会在用户按 escape 键的时间与 escape 返回到应用程序的时间之间经历延迟）。

另外，无超时模式规定，在 Curses 收到部分功能键序列的任何情况下，它会无限期等待完整的键序列。上一段中所述的“未指定间隔”在无超时模式下将成为无限期。无超时模式允许在较慢的通信线路中使用功能键。通过无超时模式，用户可以键入功能键序列的单个字符，但是当用户键入的字符（不是功能键）开始功能键序列时，也会延迟应用程序响应。由于这个原因，在无超时模式中，很多终端在用户按 escape 键的时间与按另一键的时间之间表现为挂起状态。通过调用 **notimeout()** 可以切换无超时模式。

如果任何特殊字符（请参阅“特殊字符”一节）被定义或重新定义为功能键序列的成员，则 Curses 将无法识别和转换这些功能键。

下面讨论的几种模式描述了输入可用性方面的内容。如果启用了键盘转换，那么一旦 Curses 开始收键盘序列，则在完全接收该序列或者间隔已过之前，将无法使用输入。

输入模式

《X/Open System Interface Definitions, Issue 4, Version 2》规范（"“特殊字符”"）定义了流控制字符、中断字符、清除字符和终止字符。四种互相排斥的 Curses 模式，使得应用程序可以控制这些输入字符的效果：

输入模式	效果
精细处理模式	该模式会对在应用程序之外处理的所有特殊字符进行普通的“每次一行”处理。这将获得与《X/Open System Interface Definitions, Issue 4, Version 2》规范中指定的标准模式输入处理相同的效果。通过调用 **cbreak()** 进入该模式时， ISIG 和 IXON 标记的状态不会更改，通过调用 **noraw()** 进入该模式时，将设置这两个标记。 实现支持任何受支持语言环境中的清除字符和终止字符，无论该字符的宽度为多少。
cbreak 模式	用户键入的字符可立即用于应用程序，并且 Curses 不会对清除字符或终止字符执行特殊处理。应用程序可以选择 *cbreak* 模式编辑其自己的行，但是不允许使用中止字符来中止任务。该模式获得的效果与《X/Open System Interface Definitions, Issue 4, Version 2》 规范中指定的非标准模式 - 条件 B 输入处理（MIN 设置为 1 并清除了 ICRNL）的效果相同。进入该模式时，ISIG 和 IXON 标记的状态不会更改。
半延迟模式	效果与 *cbreak* 相同，只是输入函数将等待，直到字符可用或者应用程序定义的间隔已过为止，具体看两个时间哪个先到。该模式获得的效果与《X/Open System Interface Definitions, Issue 4, Version 2》 规范中指定的非标准模式 - 条件 C 输入处理（TIME 设置为应用程序指定的值）的效果相

同。进入该模式时，ISIG 和 IXON 标记的状态不会更改。

原始模式　　原始模式为应用程序提供了对终端输入的最大程度的控制。应用程序可以看到键入的每个字符。该模式获得的效果与《X/Open System Interface Definitions, Issue 4, Version 2》规范中指定的非标准模式 - 条件 D 输入处理效果相同。进入该模式时，会清除 ISIG 和 IXON 标记。

当进程调用 **initscr()** 或 **newterm()** 来初始化 Curses 时，会记录终端接口设置，当调用 **endwin()** 时，则还原这些设置。除非实现支持符合增强 Curses（这种情况下初始输入模式为 *cbreak* 模式），否则将不指定 Curses 操作的初始输入模式。

BREAK 键的行为取决于显示驱动程序中不是由 Curses 设置的其他位。

延迟模式

两种互相排斥的延迟模式指定：调用某些 Curses 函数时，若不等待任何终端输入的情况下，这些函数返回到应用程序的速度：

No Delay　　函数失败。

Delay　　应用程序将等待，直到实现将文本传递到应用程序中为止。如果设置了 *cbreak* 模式或原始模式，则该延迟在一个字符之后产生。否则，该延迟在第一换行符、行结束标志字符或文件结束标志字符之后产生。

未指定无延迟模式对功能键处理的影响。

回显处理

回显模式确定 Curses 是否将键入的字符回显到屏幕上。回显模式的效果与 **termios** 结构（与窗口相连的终端设备相关联）的本地模式字段中 ECHO 标记的效果相似。但是，Curses 始终在其运行时清除 ECHO 标记，以阻止操作系统执行回显。这种回显字符的方法与操作系统回显字符的方法不同，因为 Curses 会执行附加的终端输入处理。

如果处于回显模式中，则 Curses 会执行其自己的回显：任何可见的输入字符都通过应用程序调用的输入函数存储在当前或指定的窗口中，位置为该窗口的光标位置，与调用 **addch()** 类似，并且具有随之发生的所有效果，如光标移动和换行。

如果未处于回显模式中，则任何输入回显都必须由应用程序执行。应用程序通常在屏幕的受控区域内执行其自己的回显操作，或者根本不执行回显，因此它们将禁用回显模式。

Curses 函数的设置

通过 Curses 函数，可以执行：整个屏幕、窗口和面板的处理；到窗口和面板的输出；读取终端输入；控制终端和 Curses 输入和输出选项；环境查询函数；颜色处理；使用标签功能键；访问终端功能的 **terminfo** 数据库；以及访问低级函数。

函数命名约定

参考联机帮助页中介绍了多个 Curses 函数系列。大多数函数系列都具有不同的函数，为编程人员提供了下列选择：

- 带有基本名称的函数在窗口 **stdscr** 上进行操作。具有同样名称以及 **w** 前缀的函数在 *win* 参数指定的窗口上运行。

  当某个函数系列的参考联机帮助页引用 “当前或指定窗口”时，对于基本函数它表示 **stdscr** ，而对于任何 **w** 函数则表示 *win* 指定的窗口。

  名称具有 **p** 前缀的函数需要一个表示面板而不是窗口的参数。

- 带有基本名称的函数基于（当前或指定窗口，如上所述）当前光标位置运行。具有相同名称以及 **mv** 前缀的函数会在执行指定的操作之前，将光标移动到 *y* 和 *x* 参数指定的位置。

  当某个函数系列的参考联机帮助页引用 “当前或指定位置”时，对于基本函数它表示光标位置，而对于任何 **mv** 函数则表示 (*y*, *x*) 位置。

  mvw 前缀存在并且将此处讨论的 **mv** 语义与上面讨论的 **w** 语义进行组合。窗口参数总是在坐标之前指定。

- 出于历史兼容性目的，通常提供带有基本名称的函数，这种函数仅针对单字节字符运行。带有相同名称以及 **w** 中缀的函数针对宽（多字节）字符运行。带有相同名称以及 **_w** 中缀的函数针对复合字符及其呈现方式运行。

- 当带有基本名称的函数针对单个字符运行时，有时会有具有相同名称以及 **n** 中缀的函数针对多个字符运行。 **n** 参数指定要处理的字符数。各自的联机帮助页都指定了 **n** 值不正确时的结果。

提供的函数系列

函数名称	说明	**s**	**w**	**c**	请参阅
	添加（覆盖）				
[mv][w]addch()	添加字符	Y	Y	Y	*addch*(3X)
[mv][w]addch[n]str()	添加字符串	N	N	N	*addchstr*(3X)
[mv][w]add[n]str()	添加字符串	Y	Y	Y	*addnstr*(3X)
[mv][w]add[n]wstr()	添加宽字符字符串	Y	Y	Y	*addnwstr*(3X)
[mv][w]add_wch()	添加宽字符和呈现方式	Y	Y	Y	*add_wch*(3X)
[mv][w]add_wch[n]str()	添加宽字符和呈现方式的数组	?	N	N	*add_wchnstr*(3X)
	更改呈现方式				
[mv][w]chgat()	更改字符在窗口中的呈现方式	-	N	N	*chgat*(3X)
	删除				
[mv][w]delch()	删除字符	-	-	N	*delch*(3X)
	获取（从键盘输入到窗口）				
[mv][w]getch()	获取字符	Y	Y	Y	*getch*(3X)
[mv][w]get[n]str()	获取字符串	Y	Y	Y	*getnstr*(3X)
[mv][w]get_wch()	获取宽字符	Y	Y	Y	*get_wch*(3X)
[mv][w]get[n]_wstr()	获取宽字符和键代码的数组	Y	Y	Y	*get_wstr*(3X)
	显示移动光标				
[w]move()	移动光标	-	-	-	*move*(3X)
	输入（退出窗口以进行读取）				
[mv][w]inch()	输入字符	-	-	-	*inch*(3X)
[mv][w]inch[n]str()	输入宽字符和呈现方式的数组	-	-	-	*inchnstr*(3X)
[mv][w]in[n]str()	输入字符串	-	-	-	*innstr*(3X)
[mv][w]in[n]wstr()	输入宽字符字符串	-	-	-	*innwstr*(3X)
[mv][w]in_wch()	输入宽字符和呈现方式	-	-	-	*in_wch*(3X)
[mv][w]in_wch[n]str()	输入字符和属性的数组	-	-	-	*in_wchnstr*(3X)
	插入				
[mv][w]insch()	插入字符	Y	N	N	*insch*(3X)
[mv][w]ins[n]str()	插入字符串	Y	N	N	*insnstr*(3X)
[mv][w]ins_[n]wstr()	插入宽字符字符串	Y	N	N	*ins_nwstr*(3X)
[mv][w]ins_wch()	插入宽字符	Y	N	N	*ins_wch*(3X)
	打印和扫描				
[mv][w]printw()	输出已设置格式的文本	-	-	-	*mvprintw*(3X)
[mv][w]scanw()	转换设置了格式的输出内容	-	-	-	*mvscanw*(3X)

图例

下面的注释说明了将字符移动到屏幕时的效果（对于 Get 函数，这仅适用于启用回显的情况下）。

s Y 表示这些函数执行特殊字符处理（请参阅“特殊字符”一节）。N 表示不执行。？表示对特殊字符应用这些函数时结果未指定。

w Y 表示这些函数执行换行（请参阅“截断、换行和滚动”一节）。N 表示不执行。

c Y 表示这些函数向前移动光标（请参阅“截断、换行和滚动”一节）。N 表示不执行。

- 该列指定的属性不适用于这些函数。

作为宏实现的接口

必须将下列带参数的接口作为宏来实现。与应用程序编程人员相关的内容是：不能在这些参数之前使用“ **&** ”字符。

宏	联机帮助页
COLOR_PAIR()	*can_change_color*(3X)
getbegyx()、**getmaxyx()**、**getparyx()**、**getyx()**	*getbegyx*(3X)

头文件参考联机帮助页列出了不采用参数的其他宏，如 **COLOR_BLACK** 。

初始化的 **Curses** 环境

执行使用 Curses 的应用程序之前，必须按照下列方式准备终端：

- 如果终端具有硬件制表位，则应该设置这些制表位。
- 为终端定义的任何初始化字符串都必须输出到该终端。

生成的终端状态必须与 Curses 拥有的终端型号兼容，该型号在 **terminfo** 数据库的终端条目中反映（请参阅 *terminfo*(4) ）。

要初始化 Curses，应用程序必须在调用处理窗口和屏幕的任何其他函数之前调用 **initscr()** 或 **newterm()** ，并且必须在退出之前调用 **endwin()** 。要在不执行回显的情况下获得“每次一个字符”输入（大多数面向屏幕的交互式程序都希望如此），则应该使用下列序列：

```
initscr();
cbreak();
noecho();
```

大多数程序还要使用下列序列：

```
nonl();
intrflush( stdscr , FALSE);
keypad( stdscr , TRUE);
```

同步终端和联网的异步终端

本节说明了应用程序编程人员在驱动同步终端、联网异步终端 (NWA) 或非标准直接相连异步终端时要记住的一些注意事项。

此类终端通常在大型机环境中使用，并以块模式与主机通信。就是说，用户在终端键入字符，然后按一个特殊键

启动到主机的字符传输。

通常情况下，尽管可以向主机发送任意大小的块，但是不能或者不应该仅通过一次键击才传输一个字符。

这样会导致希望使用单字符输入的应用程序出现严重问题；请参阅“输入处理”一节。

输出

可以采用普通方式将 Curses 接口用于与终端输出相关的所有操作，但是在某些终端上可能会出现这样的例外：**refresh()** 例行程序可能必须重绘整个屏幕内容才能执行任何更新。

如果还必须在每次这样的操作之前清除屏幕，则可能会出现不想要的结果。

输入

因为同步（块模式）终端和 NWA 终端的操作特征，所以可能无法支持所有或任一 Curses 输入函数。具体来说，应该注意以下几点：

- 可能无法进行单字符输入。可能必须按一个特殊键才能使在终端上键入的所有字符传输到主机。
- 有时可能无法禁用回显。字符回显可能直接由终端执行。在以这种方式操作的终端上，任何执行输入的 Curses 应用程序都应注意：键入的任何字符都将显示在屏幕上光标所在的任何位置。该位置不一定与窗口中的光标位置相对应。

另请参阅

sigaction(2)、 add_wch(3X)、 add_wchnstr(3X)、 addch(3X)、 addchstr(3X)、 addnstr(3X)、 addnwstr(3X)、 border(3X)、 can_change_color(3X)、 cbreak(3X)、 chgat(3X)、 def_prog_mode(3X)、 delch(3X)、 derwin(3X)、 endwin(3X)、 filter(3X)、 get_wch(3X)、 getbegyx(3X)、 getcchar(3X)、 getch(3X)、 getnstr(3X)、 in_wch(3X)、 in_wchnstr(3X)、 inch(3X)、 inchnstr(3X)、 initscr(3X)、 innstr(3X)、 innwstr(3X)、 ins_nwstr(3X)、 ins_wch(3X)、 insch(3X)、 insnstr(3X)、 keypad(3X)、 move(3X)、 mvprintw(3X)、 mvscanw(3X)、 newpad(3X)、 newwin(3X)、 notimeout(3X)、 setcchar(3X)、 terminfo(4)、 curses(5)。

<curses.h>、 <errno.h>、 <stddef.h>、 <term.h>、 <unctrl.h>。

《Curses 规范》

《X/Open System Interface Definitions, Issue 4, Version 2》

《X/Open System Interfaces and Headers, Issue 4, Version 2》

名称

curs_set — 设置光标模式

概要

#include <curses.h>

int curs_set(int *visibility***);**

说明

curs_set() 函数根据 *visibility* 的值设置光标的外观：

visibility 的值	光标的外观
0	不可见
1	终端专用的普通模式
2	终端专用的高可见性模式

终端不一定支持上述所有值。

返回值

如果终端支持 *visibility* 指定的光标模式，则 **curs_set()** 返回前一个光标状态。否则，函数将返回 ERR。

错误

没有定义任何错误。

另请参阅

<curses.h>。

历史变更记录

在 X/Open Curses 第 4 期中首次发布。

名称

cur_term — 当前终端的信息

概要

#include <term.h>

extern TERMINAL *cur_term;

说明

外部变量 *cur_term* 确定与当前使用的终端相关联的 terminfo 数据库中的记录。

另请参阅

del_curterm(3X)、 tigetflag(3X)、 <term.h>。

历史变更记录

在 X/Open Curses 第 4 期中首次发布。

名称

cuserid( ) - 获取用户的字符登录名

概要

#include <stdio.h>

char *cuserid(char *s);

备注：

由于该函数在 HP-UX 的先前版本及其他系统上表现不同，因此不推荐使用。之所以提供该函数，只是为了符合当前的行业标准，在 HP-UX 的未来版本中有可能取消该函数。

为了实现可移植性和安全性，应用程序编写人员和维护人员应该搜索现有代码，并使用等价调用 **getpwuid (getuid())** 、 **getpwuid (geteuid())** 或 **getlogin()** 替换对 **cuserid()** 的引用，具体取决于需要的用户名。

说明

cuserid() 生成进程的有效用户 ID 所对应的用户名的字符串表示形式。如果 *s* 是一个空指针，则在内部静态区域生成该表示形式，并返回静态区域的地址。否则，假定 *s* 指向至少包含 **L_cuserid** 字符的数组；该表示形式保留在此数组中。常量 **L_cuserid** 是在 **<stdio.h>** 头文件中定义的。

对于多线程应用程序，如果 *s* 是一个空指针，则不能执行操作，并且返回空指针。

诊断信息

如果找不到登录名， **cuserid()** 返回一个空指针；如果 *s* 不是空指针，则空字符 **(\0)** 将放置在 *s*[0] 。

警告

过时的接口

csuserid() 会在将来某一日过时。

另请参阅

geteuid(2)、 getuid(2)、 getlogin(3C)、 getpwent(3C)、 getpwuid(3C)、 thread_safety(5)。

符合的标准

cuserid()：AES、SVID3、XPG2、XPG3、XPG4、FIPS 151-2 和 POSIX.1

名称

datalock( ) - 分配数据空间和堆栈空间后在内存锁定进程

概要

#include <sys/lock.h>

int datalock(size_t datsiz, size_t stsiz);

说明

datalock() 至少分配 *datsiz* 字节的数据空间和 *stsiz* 字节的堆栈空间，然后在内存中锁定程序。数据空间是由 **malloc()** 分配的（请参阅 *malloc*(3C) ）。锁定程序后， **free()** 将释放该数据空间，使其可用（请参阅 *malloc*(3C) ）。这样，调用程序就可以动态使用更多的空间，而无须接收 **SIGSEGV** 信号。

调用进程的有效用户 ID 必须是超级用户，或者有使用该调用的 PRIV_MLOCK 权限的组的成员，或者具有该组的有效组 ID 的用户（请参阅 *getprivgrp*(2) 中的 *setprivgrp*(2) ）。

举例

对 **datalock()** 的以下调用将分配 4096 字节的数据空间和 2048 字节的堆栈空间，然后在内存中锁定进程：

```
datalock (4096, 2048);
```

返回值

如果 **malloc()** 无法分配足够的内存或 **plock()** 返回错误，则 **datalock()** 返回 -1（请参阅 *plock*(2) ）。

警告

多个 datalock 不能与一个大的 datalock 相同。

计算所需大小的方法仍未很好地开发。

作者

datalock() 由 HP 开发。

另请参阅

getprivgrp(2)、 plock(2)、 thread_safety(5)。

名称

dbminit、fetch、store、delete、firstkey、nextkey、dbmclose - 数据库子例程

概要

#include <dbm.h>

int dbminit(const char *file);

datum fetch(datum key);

int store(datum key, datum content);

int delete(datum key);

datum firstkey(void);

datum nextkey(datum key);

void dbmclose(void);

说明

这些函数用于维护数据库中的关键字/内容对。它们可以处理极大型（十亿个块（一个块 = 1024 字节））数据库，并可以查找一个或两个文件系统访问中的带关键字的项目。

datum 描述 *key* 参数和 *content* 参数。 **datum** 指定 *dptr* 指向的 *dsize* 字节的字符串。允许任意的二进制数据和普通的 ASCII 字符串。数据库存储在两个文件中。其中一个文件为包含关键字位图的目录，其后缀为 **.dir** 。第二文件包含所有数据，其后缀为 **.pag** 。

只有 **dbminit** 打开了数据库后，才能访问该数据库。执行此调用时，文件 *file***.dir** 和 *file***.pag** 必须存在。（通过创建长度为零的 **.dir** 文件和 **.pag** 文件可以创建空数据库）。

打开数据库后， *fetch* 将访问存储在关键字下面的数据，同时 *store* 将数据置于此关键字的下面。如果将数据存储在现有的关键字上面，则会替换现有数据。 **delete** 可删除某个关键字（及其关联的内容）。可以使用 **firstkey** 和 **nextkey** 按（表面）随机顺序建立一个贯穿数据库中所有关键字的线性传递。 **firstkey** 将返回数据库中的第一个关键字。 **nextkey** 可通过任何关键字返回数据库中的下一个关键字。可以使用以下代码遍历数据库：

for (key = firstkey(); key.dptr != NULL; key = nextkey(key))

可通过调用 **dbmclose** 关闭数据库。要打开新的数据库，必须先关闭当前打开的数据库。

诊断信息

返回 **int** 的所有函数将通过负值指示错误，通过 0 指示成功。返回 **datum** 的函数通过空的 *dptr* 指示错误。

警告

无论怎样，都不应将此库中提供的 *dbm* 函数与那些通用数据库管理系统（如 ALLBASE/HP-UX SQL）相混淆。这些函数 不能为每个条目提供多个搜索关键字， 不能防止多用户访问（也就是说，不能锁定记录或文件），也不能提供更强大的数据库管理系统中存在的其他许多有用的数据库功能。通过这些函数创建和更新数据库相对较慢，原因是数据复制是在哈希冲突的基础上完成的。对于需要快速查找相对静态的信息（需要由单个关键字建立索引）的应用程序来说，这些函数 十分有用。

.pag 文件包含漏洞，因此其外在大小约为实际内容的四倍。处理这些漏洞时，某些早期的 UNIX 系统可为其创建实际的文件块。在没有扩展的情况下，无法通过正常的方式（如 *cp*(1) 、 *cat*(1) 、 *tar*(1) 或 *ar*(1) ）复制这些文件。

这些子例程返回的 *dptr* 指针将指向后续调用更改的静态存储。

关键字/内容对的总大小不得超过内部块大小（当前为 1024 字节）。此外，混列在一起的所有关键字/内容对必须适合放在单个块上。如果磁盘块填充了不可分割的数据， **store** 将返回错误。

delete 不可以物理回收文件空间，尽管它确实能使之可重复使用。

firstkey 和 **nextkey** 提供的关键字顺序取决于哈希函数，而不是任何相关的项目。

通过 **firstkey** 和 **nextkey** 传递关键字时，执行 **store** 或 **delete** 将产生意外的结果。

作者

dbm(3C) 由加州大学伯克利分校开发。

另请参阅

ndbm(3X)。

名称

def_prog_mode()、 def_shell_mode()、 reset_prog_mode()、 reset_shell_mode() - 保存或恢复程序终端模式或 Shell 终端模式

概要

#include <curses.h>

int def_prog_mode(void);

int def_shell_mode(void);

int reset_prog_mode(void);

int reset_shell_mode(void);

说明

def_prog_mode() 函数将当前终端模式保存为“程序”（在 Curses 中）状态，供 **reset_prog_mode()** 使用。

def_shell_mode() 函数将当前终端模式保存为“Shell”（不在 Curses 中）状态，供 **reset_shell_mode()** 使用。

reset_prog_mode() 函数将终端恢复至“程序”（在 Curses 中）状态。

reset_shell_mode() 函数将终端恢复至“Shell”（不在 Curses 中）状态。

这些函数影响与当前屏幕关联的终端的模式。

返回值

成功完成后，这些函数将返回 OK。否则，将返回 ERR。

错误

没有定义任何错误。

实际应用信息

initscr() 函数可以实现调用 **def_shell_mode()** 的效果，以保存先前的终端设置，这样在调用 **endwin()** 的过程中可以恢复这些设置；它还可以实现调用 **def_prog_mode()** 的效果，以指定程序终端模式的初始定义。

通常，应用程序不需要引用 Shell 终端模式。应用程序可能发现它对于保存和恢复程序终端模式很有用。

另请参阅

doupdate(3X)、 endwin(3X)、 initscr(3X)、 **<curses.h>** 。

名称

delay_output — 延迟输出

概要

#include <curses.h>

int delay_output(int *ms***);**

说明

在支持填充字符的终端上，使用 **delay_output()** 至少可暂停输出 *ms* 毫秒。否则，表示不指定延迟的时间长度。

返回值

成功完成后， **delay_output()** 返回 OK。否则，返回 ERR。

错误

没有定义任何错误。

实际应用信息

无论终端是否支持填充字符， **delay_output()** 函数都不是精确的计时方法。

另请参阅

terminfo(4) 中的已定义的功能、 napms()、 <curses.h>。

历史变更记录

在 X/Open Curses 第 2 期中首次发布。

X/Open Curses 第 **4** 期

为清楚起见，重新编写了该条目。

名称

delch、mvdelch、mvwdelch 和 wdelch — 从窗口中删除字符

概要

```
#include <curses.h>

int delch(void);

int mvdelch(int y, int x);

int mvwdelch(WINDOW *win, int y, int x);

int wdelch(WINDOW *win);
```

说明

这些函数在当前窗口或指定窗口的当前或指定位置删除字符。该函数不更改光标位置。

返回值

成功完成后，这些函数返回 OK。否则，将返回 ERR。

错误

没有定义任何错误。

另请参阅

<curses.h>。

历史变更记录

在 X/Open Curses 第 2 期中首次发布。

X/Open Curses 第 4 期

为清楚起见，重新编写了该条目。 **delch()** 函数的参数列表已显式声明为 **void**。

名称

del_curterm、restartterm、set_curterm、setupterm — **terminfo** 数据库的接口

概要

#include <term.h>

int del_curterm(TERMINAL *_oterm_);

int restartterm(char *_term_, int _fildes_, int *_errret_);

TERMINAL *set_curterm(TERMINAL *_nterm_);

int setupterm(char *_term_, int _fildes_, int *_errret_);

extern TERMINAL *cur_term;

说明

这些函数可从 **terminfo** 数据库检索信息。

要获取对 **terminfo** 数据库的访问权限，必须首先调用 **setupterm()** 。而该函数是由 **initscr()** 和 **newterm()** 自动调用的。 **setupterm()** 函数可初始化其他函数，以使用指定终端的 **terminfo** 记录（这取决于是否调用了 **use_env()**）。该函数将 *cur_term* 外部变量设置为一个 **TERMINAL** 结构，该结构包含指定终端的 **terminfo** 数据库中的记录。

终端类型为字符型字符串 *term*；如果 *term* 为空指针，则使用环境变量 TERM。如果未设置 TERM，或者其值为空字符串，则将 **"unknown"** 用作终端类型。调用 **setupterm()** 之前，应用程序必须将 *fildes* 设置为文件描述符，并且应打开此文件描述符，以便能输出到终端设备。如果 *errret* 非空，则其指向的整数将设置为以下值之一，以报告函数的执行结果：

–1	找不到 **terminfo** 数据库（函数失败）。
0	**terminfo** 中找不到终端的条目（函数失败）。
1	成功。

如果 **setupterm()** 检测到错误，并且 *errret* 是空指针，那么 **setupterm()** 将写入一个诊断消息，然后退出。

对使用所有缺省值，并向 *stdout* 发送输出的 **setupterm()** 执行简单调用的方式为：

setupterm((char *)0, fileno(stdout), (int *)0);

set_curterm() 函数将变量 *cur_term* 设置为 *nterm*，并使所有 **terminfo** 布尔变量、数字变量和字符串变量都使用 *nterm* 中的值。

del_curterm() 函数可释放 *oterm* 指向的空间，并使之可供以后使用。如果 *oterm* 与 *cur_term* 相同，那么在再次调用 **setupterm()** 之前，对任何 **terminfo** 布尔变量、数字变量和字符串变量的引用，都可能会引用无效的内存位置。

restartterm() 函数假定已事先调用 **setupterm()** （可能通过 **initscr()** 或 **newterm()** 调用）。该函数允许应用程序

在 *term* 中指定不同的终端类型，并根据 *fildes* 更新 **baudrate()** 返回的信息，但是不损坏 **initscr()** 、 **newterm()** 或 **setupterm()** 创建的其他信息。

返回值

如果成功完成， **set_curterm()** 将返回 *cur_term* 的前一个值。否则，将返回一个空指针。

如果成功完成，其他函数将返回 OK。否则，这些函数将返回 ERR。

错误

未定义错误。

实际应用信息

如果某个应用程序需要访问 **terminfo** 数据库，但除此之外，不必要使用 Curses，那么该应用程序需要调用 **setupterm()** 。

另请参阅

baudrate(3X) 、 erasechar(3X) 、 has_ic(3X) 、 longname(3X) 、 putc(3S) 、 termattrs(3X) 、 termname(3X) 、 tgetent(3X)、 tigetflag(3X)、 use_env(3X)、 terminfo(4)、选择终端一节、<term.h>。

历史变更记录

在 X/Open Curses 第 4 期中首次发布。

名称

deleteln 和 wdeleteln — 从窗口中删除行

概要

#include <curses.h>

int deleteln(void);

int wdeleteln(WINDOW *_win_);

说明

deleteln() 和 **wdeleteln()** 函数从当前窗口或指定窗口中删除光标所在的行，并将当前行后面的所有行向光标方向移动一行。窗口的最后一行将被清除。光标位置不会更改。

返回值

成功完成后，这些函数返回 OK。否则，将返回 ERR。

错误

没有定义任何错误。

另请参阅

insdelln(3X)、 <curses.h>。

历史变更记录

在 X/Open Curses 第 2 期中首次发布。

X/Open Curses 第 **4** 期

为清楚起见，重新编写了该条目。 **deleteln()** 函数的参数列表已显式声明为 **void**。

名称

delscreen — 释放与屏幕相关的存储空间

概要

#include <curses.h>

void delscreen(SCREEN ****sp*);**

说明

delscreen() 函数释放与 *sp* 指向的 **SCREEN** 相关的存储空间。

返回值

delscreen() 函数不返回值。

错误

没有定义任何错误。

另请参阅

endwin(3X)、 initscr(3X)、 <curses.h>。

历史变更记录

在 X/Open Curses 第 4 期中首次发布。

名称

delwin — 删除窗口

概要

#include <curses.h>

int delwin(WINDOW *_win_);

说明

delwin() 函数删除 *win*，从而释放与其相关的所有内存。应用程序在删除主窗口之前必须删除子窗口。

返回值

成功完成后， **delwin()** 返回 OK。否则，返回 ERR。

错误

没有定义任何错误。

另请参阅

derwin(3X)、 dupwin(3X)、 <curses.h>。

历史变更记录

在 X/Open Curses 第 2 期中首次发布。

X/Open Curses 第 **4** 期

为清楚起见，重新编写了该条目。

名称

derwin -创建相对窗口的函数

概要

#include <curses.h>

WINDOW *derwin(WINDOW *_orig_, int _nlines_, int _ncols_, int _begin_y_, int _begin_x_);

说明

derwin() 函数创建具有 *nlines* 行和 *ncols* 列的新窗口，并使新窗口的原点位于与窗口 *orig* 的原点相对的位置 (*begin_y*, *begin_x*)。如果新窗口的任何部分位于 *orig* 之外，则函数失败，且不会创建窗口。

返回值

成功完成后，这些函数返回指向新窗口的指针。否则，它们返回空指针。

错误

没有定义任何错误。

实际应用信息

在执行子窗口的第一次刷新之前，可移植应用程序应该对父窗口调用 **touchwin()** 或 **touchline()** 。

每一个窗口都要维护屏幕图像和状态的内部描述。屏幕图像在窗口层次结构中的所有窗口间共享。刷新操作依赖于窗口中已更改的信息，该信息对于每个窗口来说是专用的。对一个窗口进行更新时，对另一个窗口进行的刷新可能不能执行所需的更新，因为窗口不共享该信息。

另请参阅

delwin(3X)、 newwin(3X)、 is_linetouched(3X)、 doupdate(3X)、 <curses.h>。

历史变更记录

在 X/Open Curses 第 4 期中首次发布。

名称

devnm - 将设备 ID 映射到文件路径

概要

```
#include <devnm.h>

int devnm (
  mode_t devtype,
  dev_t devid,
  char *path,
  size_t pathlen,
  int cache
);
```

说明

如果给定了设备类型、设备 ID，以及要在其中返回结果的字符串，那么 **devnm()** 就可以通过搜索 **/dev** 将该类型和 ID 映射到一个块，或者映射到字符专用的文件（设备文件）名称。该函数将在 *path* 中返回遇到的第一个带有匹配的设备类型和 ID 的专用文件的路径名。它可以按不确定的顺序并以递归方式搜索 **/dev** 及其所有子目录。

参数包括:

devtype *stat*(5) 中记录的文件类型值 **S_IFBLK** 或 **S_IFCHR** 中的一个。除了 **S_IFMT** 集合中的那些位，其他的位将被忽略。因此，举例来说，该值可能是 **stat()** （请参阅 *stat*(2) ）返回的 **st_mode** 值。

devid 一个设备 ID （由主编号和次编号组成），例如 **stat()** 在 **st_dev** 字段或 **st_rdev** 字段中返回的 ID。

path 指向在其中返回路径名结果的缓冲区的指针。

pathlen 通知 *path* 字符串包括 NUL 结尾字符在内的可用长度。如果因 *path* 太短而无法保存完整的路径名，则只在以 null 结尾的字符串中返回第一个 *pathlen*−1 字符，并且将更改返回值（请参阅下文）。

cache 一个标志，指示 **devnm()** 是否要在 **malloc()** 分配的内存中保存文件信息，以及以后是否要使用该保存的信息而不必再次搜索 **/dev** 。当 *cache* 为非零值时，执行后续调用可能会快得多，特别是当 **/dev** 为大树时。但是，当 *cache* 为 True 时，执行第一次调用可能会比较慢，这是因为 **devnm()** 要创建缓存，必须先读取一遍所有的 **/dev** 树，而不是在找到匹配的文件后立即返回。当 *cache* 设置为零时执行的任何调用都将忽略缓存（如果有），并读取目录。

为了配合将来可能要实现的增强，应将 *cache* 限制为值 0 和 1。

无法通知 **devnm()** 释放其缓存的内存。

devnm() 将忽略不可读目录和文件，而对于这些目录和文件，执行 **stat()** 失败。

返回值

devnm() 可返回下列值之一:

0 成功。结果在 *path* 中。

–1 **ftw()** 失败。 **errno** 包含 **ftw()** 返回的值。如果设置了 *cache* ，则可能会更改 *path* 。如果 *cache* 是第一次设置的，则 **devnm()** 释放了当前调用分配的任何内存。

–2 未找到匹配的专用文件。 **errno** 不确定。不更改 *path* 。

–3 找到了匹配的专用文件，但是为了适合 *path* 而截断了其名称。 **errno** 不确定。

如果 **malloc()** 失败， **devnm()** 将以无提示方式放弃在 *cache* 为 True 的情况下，尝试在当前调用或任何后续调用中执行缓存，同时，将释放当前调用分配的任何内存。

作者

devnm() 由 HP 创建。

另请参阅

devnm(1M)、 stat(2)、 ftw(3C)、 malloc(3C)、 ttyname(3C)、 stat(5)、 thread_safety(5)。

名称

dial()、undial() - 建立一个拨出终端线路连接

概要

#include <dial.h>

int dial(CALL call);

void undial(int fd);

说明

dial() 函数可返回为执行读取/写入操作而打开的终端线路的文件描述符。 **dial()** 的参数为 **CALL** 结构（在 **<dial.h>** 头文件中定义）。

处理完终端线路后，调用程序必须调用 **undial()** 来释放在分配终端设备过程中设置的信号量。

<dial.h> 头文件中的 **CALL** 的定义为：

```
typedef struct {
  struct termio     *attr;     /* pointer to termio attribute struct */
  int               baud;      /* transmission data rate */
  int               speed;     /* 212A modem: low=300, high=1200 */
  char              *line;     /* device name for out-going line */
  char              *telno;    /* pointer to tel-no digits string */
  int               modem;     /* specify modem control for direct lines */
  char              *device;   /* Will hold the name of the device used to make a connection */
  int               dev_len;   /* The length of the device used to make connection */
} CALL;
```

CALL 元素如下：

speed 只适用于拨出通话，在这种情况下，其值要么是用于标识 113A 调制解调器的 300 或 1200，要么是针对 212A 调制解调器使用的高速或低速设置。请注意，113A 调制解调器或采用低速设置的 212A 调制解调器能以每秒 0 至 300 位范围内的任何速率传输数据。但是，采用高速设置的 212A 调制解调器只能以每秒 1200 位的速率传输和接收数据。

baud 需要的传输波特率。例如，用户可能将 *baud* 设置为 110，将 *speed* 设置为 300（或 1200）。但如果将 *speed* 设置为 1200，则必须将 *baud* 设置为更高值 (1200)。

line 如果所需的终端线路为直线，则指向其设备名的字符串指针应置于 **CALL** 结构的 *line* 元素内。 **Devices** 文件中保留了这些终端设备名的合法值。在这种情况下则不必要指定 *baud* 元素，因为可以通过 **Devices** 文件确定此元素。

telno 一个指针，指向表示要拨打的电话号码的字符型字符串。这些号码可能只包括下面所述的符号。结尾符号是由 **dial()** 函数提供的，不应加入到已传递给 **CALL** 结构中的 **dial()** 的 *telno* 字符串。

允许的代码

0-**9**	拨打 **0**-**9**
***** 或 **:**	拨打 *****
# 或 **;**	拨打 **#**
-	4 秒延迟后响第二次拨号音
e 或 **<**	号码结尾
w 或 **=**	等待第二次拨号音
f	摘机 (off-hook) 指示灯闪烁 1 秒

modem 针对直线指定调制解调器控制。如果需要调制解调器控制，请将其设置为非零值。

attr 指向 **<termio.h>** 头文件中定义的 **termio** 结构的指针。可以将此指针元素的 NULL 值传递给 **dial()** 函数，但如果包含这样一个结构，那么应该先针对传出终端线路在该结构中设置所指定的元素，然后再建立连接。通常，对于某些属性（例如奇偶校验和波特率）来说，这是十分重要的。

device 保存建立连接的设备名称。

dev_len 已复制到阵列设备的设备名的长度。

返回值

如果失败，将返回一个负值，指明失败的原因。 **<dial.h>** 头文件中定义了此处所列的负值索引的助记符。

```
INTRPT   -1     /* interrupt occurred */
D_HUNG   -2     /* dialer hung (no return from write) */
NO_ANS   -3     /* no answer within 10 seconds */
ILL_BD   -4     /* illegal baud-rate */
A_PROB   -5     /* automatic call unit (acu) problem (open() failure) */
L_PROB   -6     /* line problem (open() failure) */
NO_Ldv   -7     /* can't open LDEVS file */
DV_NT_A  -8     /* requested device not available */
DV_NT_K  -9     /* requested device not known */
NO_BD_A  -10    /* no device available at requested baud */
NO_BD_K  -11    /* no device known at requested baud */
```

警告

包含 **<dial.h>** 头文件将自动包含 **<termio.h>** 头文件。

以上例行程序需要使用 **<stdio.h>** ，这将导致程序大小意外增加，否则程序无法使用标准 I/O。

dial() 函数将修改 **CALL** 结构的某些字段的值，因此，如果重新调用了 **dial()** ，将重新初始化 **CALL** 结构的值。

文件

/etc/uucp/Devices

另请参阅

uucp(1)、 alarm(2)、 read(2)、 write(2)、 thread_safety(5)、 termio(7)。

《Remote Access User's Guide》中的 *UUCP* 教程。

名称

directory： closedir()、 opendir()、 readdir()、 readdir_r()、 rewinddir()、 seekdir()、 telldir() - 目录操作

概要

```
#include <dirent.h>

DIR *opendir(const char *dirname);

struct dirent *readdir(DIR *dirp);

int readdir_r(DIR *dirp, struct dirent *entry, struct dirent **result);

long int telldir(DIR *dirp);

void seekdir(DIR *dirp, long int loc);

void rewinddir(DIR *dirp);

int closedir(DIR *dirp);
```

说明

此库程序包提供的函数使程序不必要了解与文件系统关联的实际目录格式就可以读取目录条目。因为通过这些函数，可以在带有不同目录格式的文件系统上灵便地使用程序，所以建议以此方法读取目录条目。

opendir() 打开目录 *dirname* ，并为其关联一个目录流。 **opendir()** 将返回一个指针，用来标识后续操作中的目录流。 **opendir()** 使用 **malloc()** 来分配内存。

readdir() 返回指向下一个目录条目的指针。如果到达目录的结尾，或者检测到无效的 **seekdir()** 操作，将返回 NULL 指针。有关目录条目中可用字段的说明，请参阅 *dirent*(5) 。

readdir_r() 初始化 **entry** 引用的 **dirent** 结构，以表示 **dirp** 引用的目录流中的当前位置；在 **result** 引用的位置上存储指向该结构的指针。

telldir() 返回与 *dirp* 引用的目录流相关联的当前位置（已编码）。

seekdir() 在 *dirp* 引用的目录流上设置下一个 **readdir()** 操作的位置。 *loc* 参数是从 **telldir()** 获取的目录流中的一个位置。目录流的位置将恢复到 **telldir()** 返回 *loc* 值时它所处的位置。只有当从其中派生了 **telldir()** 返回值的 **DIR** 指针处于打开状态时，这些值才有效。如果将目录流关闭后再重新打开， **telldir()** 值可能无效。

rewinddir() 将 *dirp* 引用的目录流的位置重置到目录的开头。它还能使目录流引用对应目录的当前状态，如同调用 **opendir()** 一样。

closedir() 关闭指定的目录流，然后释放与 **DIR** 指针关联的结构。

实际应用信息

readdir_r() 的用户应注意，现在 **readdir_r()** 符合 POSIX.1c 线程标准。为了与现有的 DCE 应用程序兼容，还支持 **readdir_r()** 的早期原型。

返回值

opendir() 如果成功完成，将返回一个指向引用已打开目录流的 **DIR** 类型对象的指针。否则，将返回 NULL 指针，并设置全局变量 **errno** 以指明错误。

readdir() 如果成功完成，将返回一个指向描述目录条目的 **struct dirent** 类型对象的指针。如果到达目录的结尾， **readdir()** 将返回 NULL 指针，并且不更改 **errno** 的值。否则，将返回 NULL 指针，并设置 **errno** 以指明错误。

readdir_r() 如果成功完成，将返回 0。如果成功返回，那么在 **result** 位置返回的指针与参数 **entry** 的值相同。如果到达目录流的结尾， **result** 的值为 NULL 。出现错误时将返回错误编号。

telldir() 如果成功完成，将返回一个 long 值，指示目录中的当前位置。否则，将返回 **-1** ，并设置 **errno** 以指明错误。

seekdir() 不返回任何值，但如果遇到错误，将设置 **errno** 以指明错误。

closedir() 如果成功完成，将返回值 **0** 。否则，将返回值 **-1** ，并设置 **errno** 以指明错误。

错误

如果出现下列任意情况， **opendir()** 将失败：

[EACCES] *dirname* 组件的搜索权限被拒绝，或者 *dirname* 的读取权限被拒绝。

[EFAULT] *dirname* 指向进程的分配地址空间以外的区域。该错误的检测是否可信与实现有关。

[ELOOP] 转换路径名时，遇到过多的符号链接。

[EMFILE] 当前，为调用进程打开了过多的已打开文件描述符。

[ENAMETOOLONG] *dirname* 的某个组件超过 **PATH_MAX** 字节，或者，当 **_POSIX_NO_TRUNC** 有效时，*dirname* 的整个长度超过 **PATH_MAX** – 1 字节。

[ENFILE] 当前，系统上打开了过多的已打开文件描述符。

[ENOENT] *dirname* 的某个组件不存在，或 *dirname* 指向空字符串。

[ENOMEM] **malloc()** 无法提供足够的内存来处理目录。

[ENOTDIR] *dirname* 的某个组件不是目录。

如果出现下列任意情况， **readdir()** 或 **readdir_r()** 可能会失败：

[EBADF] *dirp* 不引用打开的目录流。

[EFAULT] *dirp* 指向进程的分配地址空间以外的区域。

[EIO] 发生了 I/O 错误。

[ENOENT] *dirp* 引用的目录流不在有效的目录条目上。

[ENXIO] 目录可能已损坏。

如果出现下列任意情况， **telldir()** 可能失败：

[EBADF] *dirp* 不引用打开的目录流。

[ENOENT] *dirp* 指定了不正确的文件系统块大小。

如果遇到以下情况， **seekdir()** 可能失败：

[ENOENT] *dirp* 指定了不正确的文件系统块大小。

如果遇到以下任何情况， **closedir()** 可能失败：

[EBADF] *dirp* 不引用打开的目录流。

[EFAULT] *dirp* 指向进程的分配地址空间以外的区域。

如果出现下列任意情况， **rewinddir()** 可能失败：

[EBADF] *dirp* 不引用打开的目录流。

[EFAULT] *dirp* 指向进程的分配地址空间以外的区域。

举例

以下代码将在当前目录中搜索条目 *name* ：

```
DIR *dirp;
struct dirent *dp;
dirp = opendir(".");
while ((dp = readdir(dirp)) != NULL) {
    if (strcmp(dp->d_name, name) == 0) {
        (void) closedir(dirp);
        return FOUND;
    }
}
(void) closedir(dirp);
return NOT_FOUND;
```

警告

readdir() 和 **getdirentries()** （请参阅 *getdirentries*(2) ）是访问远程 NFS 目录的唯一方法。尝试使用 **read()** 以通过 NFS 读取远程目录，将返回 **-1** ，并将 **errno** 设置为 [EISDIR] （请参阅 *read*(2) ）。

如果在调用 **opendir()** 或 **rewinddir()** 之后，紧接着在目录中删除或添加文件，则后续调用 **readdir()** 或 **readdir_r()** 之后，是否会返回该文件的条目将是无法预测的。

对于 32 位应用程序，如果文件系统使用 64 位值，则 **dirent** 结构的 *d_ino* 字段可能溢出。在这种情况下，最突出的字节将被截断，且不生成错误，而 *d_ino* 值可能不再是唯一的。

作者

directory 由 AT&T、HP 和加州大学伯克利分校联合开发。

另请参阅

close(2)、 getdirentries(2)、 lseek(2)、 open(2)、 read(2)、 dir(4)、 dirent(5)、 thread_safety(5)。

符合的标准

closedir()：AES、SVID2、SVID3、XPG2、XPG3、XPG4、FIPS 151-2、POSIX.1

opendir()：AES、SVID2、SVID3、XPG2、XPG3、XPG4、FIPS 151-2、POSIX.1

readdir()：AES、SVID2、SVID3、XPG2、XPG3、XPG4、FIPS 151-2、POSIX.1

readdir_r()：POSIX.1c

rewinddir()：AES、SVID2、SVID3、XPG2、XPG3、XPG4、FIPS 151-2、POSIX.1

seekdir()：AES、SVID3、XPG2、XPG3、XPG4

telldir()：AES、XPG2、XPG3、XPG4

名称

div( )、ldiv( )、lldiv( )、imaxdiv( ) - 整数除法和余数

概要

#include <stdlib.h>

div_t div(int numer, int denom);

ldiv_t ldiv(long int numer, long int denom);

lldiv_t lldiv(long long numer, long long denom);

#include <inttypes.h>

imaxdiv_t imaxdiv(intmax_t numer, intmax_t denom);

说明

div() 计算分子 *numer* 被分母 *denom* 除后的商及余数。如果除法不精确，则最终商的符号就是代数商的符号，而最终商的量值则是小于代数商的量值的最大整数值。如果结果可以表示，则将以 **div_t** 类型的结构体（在 **<stdlib.h>** 中定义）返回该结果，结构体成员 *quot* 和 *rem* 分别对应商和余数。这两个成员的类型均为 **int** ，值为 *quot* × *denom* **+** *rem* **=** *numer* 。如果结果无法表示，则过程将不确定。

ldiv() 与 **div()** 类似，不同之处在于，每个参数的类型均为 **long**int ，而结果返回的结构体类型为 **ldiv_t**（在 **<stdlib.h>** 中定义），其 **long**int 类型的成员 *quot* 和 *rem* 分别对应商和余数。

lldiv() 与 **div()** 类似，不同之处在于，每个参数的类型均为 **long**long ，而结果返回的结构体类型为 **lldiv_t** （在 **<stdlib.h>** 中定义），其 **long**long 类型的成员 *quot* 和 *rem* 分别对应商和余数。

imaxdiv() 与 **div()** 类似，不同之处在于，每个参数的类型均为 **intmax_t** ，而结果返回的结构体类型为 **imaxdiv_t** （在 **<inttypes.h>** 中定义），其 **intmax_t** 类型的成员 *quot* 和 *rem* 分别对应商和余数。

警告

如果 *denom* 为 0，则结果将不确定。

另请参阅

floor(3M)、 thread_safety(5)。

符合的标准

div()：AES、SVID3、XPG4、ANSI C

ldiv()：AES、SVID3、XPG4、ANSI C

lldiv()：ISO/IEC 9899:1999 (C99), UNIX 03

imaxdiv()：ISO/IEC 9899:1999 (C99), UNIX 03

名称

dladdr() - 获取某个地址的符号信息

概要

cc [*flag*]... *cfile* ... **-ldl** [*library*]...

#include <dlfcn.h>
int dladdr(void *address, Dl_info *dlip);

多线程应用信息

该例行程序是线程安全的。

说明

dladdr() 是使用户能够直接访问动态链接设备（在编译程序或 **ld** 命令行上使用 **-ldl** 选项）的一系列例行程序之一。进程可通过 **dladdr()** 获取有关最近定义给定 *address* 的符号的信息。 **dladdr()** 可确定指定的 *address* 是否位于构成进程的进址空间的其中一个加载模块（可执行库或共享库）内。如果某个地址位于在其上面映射加载模块的基址和为该加载模块映射的最高虚拟地址之间（包括两端），则认为该地址在加载模块的范围内。如果某个加载模块符合这个条件，则会搜索其动态符号表，以查找与指定的 *address* 最接近的符号。最接近的符号是指其值等于，或最为接近但小于指定的 *address* 的符号。

dlip 是指向 **Dl_info** 结构的指针。该结构必须由用户分配。如果指定的 *address* 在其中一个加载模块的范围内，则结构成员由 **dladdr()** 设置。 **Dl_info** 结构包含下列成员：

```
struct {
    const char *dli_fname;
    void *dli_fbase;
    const char *dli_sname;
    void *dli_saddr;
    size_t dli_size;    /* ELF only */
    int dli_bind;       /* ELF only */
    int dli_type;
};
```

Dl_info 结构包含以下字段：

dli_fname 一个指针，指向包含 *address* 的加载模块的文件名。每次调用 **dladdr()** 后，该内存位置的内容都可能发生更改。

dli_fbase 加载模块的句柄。该句柄可用作 **dlsym()** 的第一个参数。

dli_sname 一个指针，指向与指定的 *address* 最接近的符号的名称。该符号要么带有相同的地址，要么是带有低位地址的最接近符号。两次调用 **dladdr()** 后，该内存位置的内容可能发生更改。

dli_saddr 最接近符号的实际地址。对于代码符号，它包含最接近代码符号的 OPD（正式 Plabel 描述符）的地址。

dli_size （仅限 ELF 进程）动态符号表中定义的最接近符号的大小。

dli_bind （仅限 ELF 进程）动态符号表中定义的最接近符号的绑定属性。其值用于 ELF 符号表（请参阅 **<elf.h>** ）中的符号的绑定。

dli_type 最接近符号的类型。对于 ELF 进程，这与动态符号表中的类型的值相同。其值用于 ELF 符号表（请参阅 **<elf.h>** ）中的符号的类型。对于 SOM 进程，这可能包括 **<dl.h>** 中定义的值 **TYPE_DATA** 或 **TYPE_PROCEDURE** 。

返回值

如果指定的 *address* 不在其中一个加载模块的范围内，则返回 **0** ；且不修改 **Dl_info** 结构的内容。否则，将返回一个非零值，同时设置 **Dl_info** 结构的字段。

诊断信息

如果在包含 *address* 的加载模块内，找不到其值小于或等于 *address* 的符号，则 *dli_sname* 、 *dli_saddr* 和 *dli_size* 字段将设置为 **0** ； *dli_bind* 字段设置为 **STB_LOCAL** ， *dli_type* 字段设置为 **STT_NOTYPE** 。

对于 **a.out** ，通常只导出一部分可见符号：尤其是链接了 **a.out** 的加载模块引用的那些符号。可以使用链接程序（请参阅 *ld*(1) ）来控制任何共享库或 **a.out** 的输出符号的确切集合。

错误

如果 **dladdr()** 失败，则随后对 **dlerrno()** 的调用返回下列值之一：

[RTLD_ERR_BAD_DLL]	加载模块中的符号地址无效。
[RTLD_ERR_CANT_APPLY_RELOC]	不能在库中进行重定位。
[RTLD_ERR_DLADDR_NOTFOUND]	在所有加载模块中都找不到该地址。
[RTLD_ERR_NO_MEMORY]	内存不足。
[RTLD_ERR_SETCANCELSTATE_FAILED]	进入或退出 **dladdr()** 时 **__thread_setcancelstate** 失败。
[RTLD_ERR_SIGENABLE_FAILED]	退出 **dladdr()** 时 **sigenable** 失败。
[RTLD_ERR_SIGINHIBIT_FAILED]	进入 **dladdr()** 时 **siginhibit** 失败。

另请参阅

cc(1)、 ld(1)、 sh(1)、 exec(2)、 dlclose(3C)、 dlerrno(3C)、 dlerror(3C)、 dlsym(3C)。

文本和教程

《HP-UX Linker and Libraries Online User Guide》
（请参阅 **+help** 选项）

《HP-UX Linker and Libraries User's Guide》
（有关订购信息，请参阅 *manuals*(5) ）

名称

dlclose() - 关闭共享库

概要

cc [*flag* ...] *file* ... **-ldl** [*library*]...

#include <dlfcn.h>
int dlclose(void *handle);

多线程应用信息

该例行程序是线程安全的。

注释：动态加载程序使用递归 pthread 互斥锁将多线程应用程序中共享库的加载和卸载串行化。有关详细信息，请参阅《HP-UX Linker and Libraries User's Guide》。

说明

dlclose() 是一系列例行程序中的一个例行程序，这些例行程序使用户可以直接访问动态链接工具（使用编译程序上的 **-ldl** 选项或 **ld** 命令行）。 **dlclose()** 取消以前由 **dlopen()** 打开的一个共享对象与当前进程之前的关联。一旦使用 **dlclose()** 关闭了一个对象， **dlsym()** 将不再能使用其符号。由于对被引用对象调用 **dlopen()** 而自动加载的所有对象也将被关闭（请参阅 *dlopen*(3C) ）。 *handle* 是之前调用 **dlopen()** 所返回的值。

返回值

如果成功关闭了被引用对象， **dlclose()** 返回 **0** 。如果未能关闭该对象，或者 handle 没有引用打开的对象，则 **dlclose()** 返回一个非零值。可通过 **dlerror()** 获得更详细的诊断信息。

错误

如果 **dlclose()** 失败，则随后对 **dlerrno()** 进行调用将返回下列值之一：

[RTLD_ERR_CANT_APPLY_RELOC]	不能在库中进行重定位。
[RTLD_ERR_DLCLOSE_REMAINING_DEP]	存在剩余的相关性，无法关闭库。
[RTLD_ERR_INV_HANDLE]	句柄无效。
[RTLD_ERR_NO_MEMORY]	内存不足。
[RTLD_ERR_SETCANCELSTATE_FAILED]	进入或退出 **dlclose()** 时 **__thread_setcancelstate** 失败。
[RTLD_ERR_SIGENABLE_FAILED]	退出 **dlclose()** 时 **sigenable** 失败。
[RTLD_ERR_SIGINHIBIT_FAILED]	进入 **dlclose()** 时 **siginhibit** 失败。
[RTLD_ERR_UNKNOWN_HANDLE]	未知句柄。

警告

成功调用 **dlclose()** 并不保证已经从进程的地址空间中实际删除了与 句柄相关的对象。由某一次调用 **dlopen()** 所加载的对象，也可能会由另一次调用 **dlopen()** 加载。同一对象也可能会被多次打开。只有在关闭了通过显式

dlopen() 调用打开的对象的所有引用，并且还关闭了隐式引用该对象的所有其他对象时，才能将该对象从地址空间中删除。

一旦使用 **dlclose()** 关闭了一个对象，则该对象中包含的引用符号可能会导致未定义的行为。

另请参阅

dlerrno(3C)、 dlerror(3C)、 dlopen(3C)、 dlsym(3C)。

文本和教程

《HP-UX Linker and Libraries Online User Guide》
（请参阅 **+help** 选项）

《HP-UX Linker and Libraries User's Guide》
（有关订购信息，请参阅 *manuals*(5) ）

名称

dlerrno() - 获取动态链接调用的错误代码

概要

cc [*flag*...] *file*... **-ldl** [*library*]...

#include <dlfcn.h>
int *dlerrno(void);

说明

dlerrno() 是允许用户直接访问动态链接设备（使用编译程序上的 **-ldl** 选项或 **ld** 命令行）的其中一个例行程序系列。 **dlerrno()** 返回一个与动态链接处理期间发生的最后一个错误对应的数字型错误代码。如果自上次调用 **dlerrno** 以来未发生动态链接错误， **dlerrno** 将返回 [RTLD_ERR_NO_ERR] (**-1**) 。因此，如果在调用 **dlerrno()** 后立即再次调用它，就会导致返回 [RTLD_ERR_NO_ERR] (**-1**) 。

下面是助记错误代码的完整列表。可以在 **<dlfcn.h>** 中找到数字值。

[RTLD_ERR_UNKNOWN_ERR]	未知错误。
[RTLD_ERR_NO_ERR]	无错误。
[RTLD_ERR_OPEN]	无法打开加载模块。
[RTLD_ERR_IO]	映射加载模块时发生 I/O 错误。
[RTLD_ERR_BAD_DLL]	加载模块无效。
[RTLD_ERR_BAD_ELF_VER]	库中 ELF 版本未知。
[RTLD_ERR_LIB_OPEN]	无法找到库。
[RTLD_ERR_NO_MEMORY]	无法分配动态内存。
[RTLD_ERR_BAD_RELOC]	未知的重定位类型。
[RTLD_ERR_INTERNAL_ERROR]	遇到内部错误。
[RTLD_ERR_DLOPEN_BAD_FLAGS]	**dlopen()** 调用的标记无效。
[RTLD_ERR_CANT_APPLY_RELOC]	未能应用重定位。
[RTLD_ERR_INV_NEXT_HANDLE]	**RTLD_NEXT** 参数无效。
[RTLD_ERR_UNKNOWN_SYMBOL]	未知符号。
[RTLD_ERR_INV_LIB_INDEX]	加载模块索引无效。
[RTLD_ERR_INV_HANDLE]	加载模块句柄无效。
[RTLD_ERR_DLCLOSE_REMAINING_DEP]	存在剩余的附属件，无法关闭加载模块。

[RTLD_ERR_TPREL_NON_TLS_SYM] 针对非线程特定的符号进行 **TPREL** 重定位。

[RTLD_ERR_NON_TLS_RELOC_TO_TLS_SYM]
对线程特定的符号进行非线程特定的重定位引用。

[RTLD_ERR_MMAP_FAILED] **mmap()** 失败。

[RTLD_ERR_DLOPEN_TLS_LIB] 无法使用线程特定的数据针对加载模块调用 **dlopen()** 。

[RTLD_ERR_UNLOAD_HANDLE] 用于卸载的加载模块句柄无效。

[RTLD_ERR_UNLOAD_DEP] 存在剩余的附属件，无法卸载加载模块。

[RTLD_ERR_CODE_UNSAT] 加载模块中不符合要求的代码符号。

[RTLD_ERR_DATA_UNSAT] 加载模块中不符合要求的数据符号。

[RTLD_ERR_BAD_SYMNAME] **shl_findsym()** 的符号名称无效。

[RTLD_ERR_BAD_SYMTYPE] 符号类型无效。

[RTLD_ERR_SHL_LOAD_BAD_FLAGS]
shl_load() 的标记无效。

[RTLD_ERR_BAD_TYPE_DEFINESYM]
shl_definesym() 的符号类型无效。

[RTLD_ERR_UNKNOWN_HANDLE] 未知句柄。

[RTLD_ERR_INV_BUFFER_ARGUMENT]
描述符参数无效。

[RTLD_ERR_INV_ADDRESS] **dlmodinfo()** 的地址无效。

[RTLD_ERR_INV_DESC_VERSION] 描述符版本无效。

[RTLD_ERR_FB_CANNOT_WRITE] 无法写入 **fastbind** 数据。

[RTLD_ERR_FB_CANNOT_OPEN_AOUT]
无法打开用于读取 **fastbind** 数据的程序文件。

[RTLD_ERR_FB_CANNOT_READ] 无法读取 **fastbind** 数据。

[RTLD_ERR_FB_DATA_OUT_OF_DATE]
Fastbind 数据过时。

[RTLD_ERR_INV_OPTION] **dld** 选项无效。

[RTLD_ERR_FB_NO_FASTBIND_DATA]
没有 **fastbind** 数据。

[RTLD_ERR_SIGINHIBIT_FAILED] 进入时 **siginhibit()** 调用失败。

[RTLD_ERR_SIGENABLE_FAILED] 退出时 **sigenable()** 调用失败。

[RTLD_ERR_READ_TGT_MEM_FAILED]
read_tgt_mem 未能读取目标内存。

[RTLD_ERR_BAD_ABI1] 因为 64 位程序找到 32 位共享库，所以 ABI 不匹配。

[RTLD_ERR_BAD_ABI2] 因为 32 位程序找到 64 位共享库，所以 ABI 不匹配。

[RTLD_ERR_INV_DLHOOK_FLAG] **dlhook** 标记无效。

[RTLD_ERR_BAD_DLL_MAGIC_NUM]
由于幻数错误，库无效。

[RTLD_ERR_BAD_DLL_ALIGNMENT]
由于对齐错误，库无效。

[RTLD_ERR_BAD_DLL_NO_SYMTAB]
由于缺少符号表，库无效。

[RTLD_ERR_BAD_DLL_BAD_PHDR] 由于缺少程序头，库无效。

[RTLD_ERR_BAD_DLL_BAD_MACHINE]
由于计算机类型错误，库无效。

[RTLD_ERR_BAD_DLL_BAD_OBJFILE]
由于对象文件类型错误，库无效。

[RTLD_ERR_DLDD_COMM_FAILURE]
无法从 **dldd** 中获取共享的固定地址。

[RTLD_ERR_BAD_DLL_SEGMENT_COUNT]
存在多个数据段，无法使用固定地址。

[RTLD_ERR_SHL_LOAD_NULL_FILE_NAME]
NULL 库名无效。

[RTLD_ERR_BAD_SHL_GETSYMBOLS_FLAG]
shl_getsymbols() 的标记或类型错误。

[RTLD_ERR_INV_DLMODADD_ARGUMENT]
dlmodadd() 的参数无效。

[RTLD_ERR_FILTER_TLS] 无法打开过滤的库：TLS 大小超出过滤器中记录的大小。

[RTLD_ERR_INST_OPEN] Caliper 规范失败：无法打开文件。

[RTLD_ERR_INST_NOTE_SEG] Caliper 规范失败：无法读取可加载的注释部分。

[RTLD_ERR_INST_PIPE] Caliper 规范失败：无法打开用于通信的管道。

[RTLD_ERR_INST_FORK] Caliper 规范失败： **fork()** 失败。

[RTLD_ERR_INST_EXEC] Caliper 规范失败： **execve()** 失败。

[RTLD_ERR_INST_CALIPER_ACK] Caliper 规范失败：Caliper 返回失败。

[RTLD_ERR_INST_INTERFERENCE] Caliper 规范失败：在管道上检索到未知的消息。

[RTLD_ERR_INST_STAT] Caliper 规范失败：无法对文件调用 stat。

[RTLD_ERR_DYN_FILTER_STLS_REF]
库包含对动态加载库中定义的符号的静态 TLS 引用。

[RTLD_ERR_SETCANCELSTATE_FAILED]
调用 **__thread_setcancelstate()** 失败。

[RTLD_ERR_DLADDR_NOTFOUND] 在所有加载模块中都找不到该地址。

[RTLD_ERR_MPROTECT_FAILED] 针对库调用 **mprotect()** 失败。

[ERR_INV_DLSETLIBPATH_ARGUMENT]
dlsetlibpath() 的参数无效。

[ERR_DLGETFILEINFO_MTEXT] 库具有多个文本段。

[ERR_DLGETFILEINFO_MDATA] 库具有多个数据段。

[ERR_INV_DLGETFILEINFO_ARGUMENT]
dlgetfileinfo() 的参数无效。

[ERR_DLGETFILEINFO_IO] 使用 **dlgetfileinfo()** 访问库时发生 I/O 错误。

[ERR_DLOPENE_BAD_ADDR] **dlopene()** 的加载地址无效。

[ERR_PREALLOC_ADDR_NOT_USE] 由于使用了固定地址，无法使用预配置地址来映射库。

[RTLD_ERR_DLOPENE_NO_EXEC_PERM]
共享库缺少执行权限。

[RTLD_ERR_NOMMAP_FAILED] 加载库时遇到错误。

[RTLD_ERR_ARCH_EXT_NOT_SUPPORTED]
硬件中不存在所需的扩展，无法加载库。

多线程应用信息

该例行程序是线程安全的。

另请参阅

dlclose(3C)、 dlerror(3C)、 dlopen(3C)、 dlsym(3C)。

文本和教程:

《HP-UX Linker and Libraries Online User Guide》
（请参阅 **+help** 选项）

《HP-UX Linker and Libraries User's Guide》
（有关订购信息，请参阅 *manuals*(5) ）

名称

dlerror - 获取诊断信息

概要

cc [*flag* ...] *file* ... **-ldl** [*library*] ...

#include <dlfcn.h>
char *dlerror(void);

说明

dlopen 是允许用户直接访问动态链接设备（编译程序或 **ld** 命令行使用 **-ldl** 选项可实现）的一组例行程序。 **dlerror** 将返回以 null（空）结尾的字符串（没有尾随换行符），用于说明在动态链接处理过程中发生的最后一个错误。如果自上次调用 **dlerror**以来未发生动态链接错误， **dlerror** 就会返回 NULL。如果在前一次调用后紧跟着再次调用 **dlerror** ，就会返回 NULL。

多线程应用信息

该例行程序是线程安全的。

警告

dlerror 返回的消息可能会驻留在每次调用 **dlerror**时都将予以覆盖的静态缓冲区中。不应将应用程序代码写入此缓冲区内。对于希望保留错误消息的程序，应生成一份自己的此消息的副本。

另请参阅

dlclose(3C)、dlopen(3C)、dlsym(3C)。

文本和教程:

《HP-UX Linker and Libraries Online User Guide》
（请参阅 **+help** 选项）
《HP-UX Linker and Libraries User's Guide》
（有关订购信息，请参阅 *manuals*(5) ）

名称

dlget() - 检索有关已加载模块（程序或共享库）的信息

概要

cc [*flag*]... *file*... **-ldl** [*library*]...

```
#include <dlfcn.h>
void *dlget(int index,
        struct load_module_desc *desc,
        size_t desc_size);
```

多线程应用信息

该例行程序是线程安全的。

说明

dlget() 是一系列例行程序中的一个例行程序，这些例行程序使用户可以直接访问动态链接工具。 **dlget()** 为进程返回有关已加载模块的信息。它从 *index* 中检索加载模块的相关信息，该 *index* 指定加载模块在动态加载程序搜索列表中的位置。值为 **-1** 的 *index* 请求动态加载程序的相关信息。值为 **-2** 的 *index* 请求关于程序文件本身的信息。

如果成功， **dlget()** 将返回 **dlopen()** 的返回值定义的共享库句柄。

desc 必须由用户预先分配。结构成员是由动态加载程序使用所请求共享库的相关信息填充的。

load_module_desc 结构具有下列成员：

```
struct load_module_desc {
  unsigned long text_base;
  unsigned long text_size;
  unsigned long data_base;
  unsigned long data_size;
  unsigned long unwind_base;
  unsigned long linkage_ptr;
  unsigned long phdr_base;
  unsigned long tls_size;
  unsigned long tls_start_addr;
  }
```

desc_size 指定用户发送的 *load_module_desc* 结构的大小（以字节为单位）。

如果调用 **dlget()** 失败，则返回空指针， *desc* 保持不变。

错误

如果 **dlget()** 失败，则随后对 **dlerrno()** 的调用返回下列值之一：

[RTLD_ERR_CANT_APPLY_RELOC] 不能在库中进行重定位。

[RTLD_ERR_INV_BUFFER_ARGUMENT]	描述符参数无效。
[RTLD_ERR_INV_DESC_VERSION]	描述符版本无效。
[RTLD_ERR_INV_LIB_INDEX]	加载模块索引无效。
[RTLD_ERR_NO_MEMORY]	内存不足。
[RTLD_ERR_SETCANCELSTATE_FAILED]	进入或退出 **dlget()** 时 **__thread_setcancelstate** 失败。
[RTLD_ERR_SIGENABLE_FAILED]	退出 **dlget()** 时 **sigenable** 失败。
[RTLD_ERR_SIGINHIBIT_FAILED]	进入 **dlget()** 时 **siginhibit** 失败。

作者

dlget() 由 HP 开发。

另请参阅

系统工具

exec(2)	系统加载程序。
ld(1)	调用链接编辑器。

其他信息

a.out(4)	汇编程序、编译程序和链接程序输出。
dlclose(3C)	卸载以前由 **dlopen()** 加载的共享库。
dlerror(3C)	输出 **dld** 记录的最新错误消息。
dlerrno(3C)	返回 **dld** 记录的最新错误的错误代码。
dlgetname(3C)	返回包含加载模块的存储空间的名称。
dlmodinfo(3C)	返回有关已加载模块的信息。
dlopen(3C)	加载共享库。
dlsym(3C)	获取共享库中符号的地址。

文本和教程

《HP-UX Linker and Libraries Online User Guide》
（请参阅 **+help** 选项）

《HP-UX Linker and Libraries User's Guide》
（有关订购信息，请参阅 *manuals*(5) ）

名称

dlgetfileinfo() - 在加载库之前返回库的文件信息

概要

cc [*flag*...] *file*... **-ldl** [*library*]...

```
#include  <dlfcn.h>
uint64_t dlgetfileinfo(const char *file,
                size_t info_size,
                struct dlfileinfo *info);
```

多线程应用信息

该例行程序是线程安全的。

说明

dlgetfileinfo() 是允许用户直接访问动态链接设备（使用编译程序或 **ld** 命令行上的 **-ldl** 选项）的例行程序系列之一。 **dlgetfileinfo()** 在加载库之前返回库的文件信息。此信息可用于在调用 **dlopene()** 之前分配加载段。

file 用于构建指向库的路径名。动态加载程序使用 **dlopen()** 和 **dlopene()** 所使用的标准搜索规则搜索库。如果找到库并且该库是有效的共享库，则 **dlgetfileinfo** 将通过 *info* 参数返回有关该库的信息。

info_size 是 *info* 缓冲区的大小（以字节为单位）。

info 是指向由用户程序分配的缓冲区的指针。动态加载程序使用文件信息填充此缓冲区。

dlfileinfo 结构具有下列成员：

```
struct dlfileinfo {
  size_t text_size;
  size_t data_size;
  char *filename;
}
```

text_size 是共享库的文本段的大小（以字节为单位）。

data_size 是共享库的数据段的大小（以字节为单位）。

filename 是指向共享库的路径。可以将此路径传递到随后的 **dlopen()** 或 **dlopene()** 调用以避免再次搜索库。 **dlgetfileinfo()** 的调用方必须复制 *filename* 的值以确保该值未损坏。

举例

下面的示例说明使用 **dlgetfileinfo()** 分配内存以映射共享库的数据段。为简化示例，省略了错误检查。

```
#include <dlfcn.h>
#include <string.h>


/* allocate_data is a user-supplied routine that allocates a
```

```
 * a memory buffer of at least "data_size" bytes and returns
 * a pointer to the buffer.
 */
extern char *allocate_data(size_t data_size);

int main() {
    void   *handle;
    int    status;
    char   *pathname;
    struct dlfileinfo  info;
    struct dlopen_opts opts;

    /* Locate library and get file information */
    status = dlgetfileinfo("mylib.so", sizeof(info), &info);

    if (status == 0) {

        /* Make a copy of the library pathname returned by
         * dlgetfileinfo().
         */
        pathname = strdup(info.filename);

        /* Allocate data segment */
        opts.data_addr = allocate_data(info.data_size);

        /* Not preallocating text segment */
        opts.text_addr = 0;

        /* Set dlopene() flags to indicate the data segment
         * has been preallocated.
         */
        opts.flags = RTLD_EXT_DATA_ADDR;

        /* Call dlopene() to load the library using the path
         * where dlgetfileinfo found the library and the
         * preallocated memory buffer for mapping the library's
         * data segment.
         */
        handle = dlopene(pathname, RTLD_LAZY, &opts);
```

```
        /* Remove copy of library pathname */
        free(pathname);

        /* Insert code here to use library */

        /* Close library */
        status = dlclose(handle);

        /* Insert code here to free storage allocated by
         * allocate_data().
         */
      }
    }
```

返回值

如果成功， **dlgetfileinfo()** 返回 **0** ，否则返回非零值。可通过 **dlerror()** 或 **dlerrno()** 获得更详细的诊断信息。

错误

如果 **dlgetfileinfo()** 失败，则随后对 **dlerrno()** 的调用返回下列值之一：

[RTLD_ERR_BAD_ELF_VER]
库中 ELF 版本未知。

[RTLD_ERR_LIB_OPEN]
无法找到库。

[RTLD_ERR_NO_MEMORY]
无法分配动态内存。

[RTLD_ERR_INTERNAL_ERROR]
dlgetfileinfo() 中遇到内部错误。

[RTLD_ERR_CANT_APPLY_RELOC]
在解析对 **dlgetfileinfo()** 的调用时，未能应用重定位。

[RTLD_ERR_SIGINHIBIT_FAILED]
进入 **dlgetfileinfo()** 时 **siginhibit** 调用失败。

[RTLD_ERR_SIGENABLE_FAILED]
退出 **dlgetfileinfo()** 时 **sigenable** 调用失败。

[RTLD_ERR_BAD_ABI1]
因为 64 位程序找到 32 位共享库，所以 ABI 不匹配。

[RTLD_ERR_BAD_ABI2]
因为 32 位程序找到 64 位共享库，所以 ABI 不匹配。

[RTLD_ERR_BAD_DLL_MAGIC_NUM]
由于幻数错误，库无效。

[RTLD_ERR_BAD_DLL_BAD_MACHINE]
由于计算机类型错误，库无效。

[RTLD_ERR_BAD_DLL_BAD_OBJFILE]
由于对象文件类型错误，库无效。

[RTLD_ERR_SETCANCELSTATE_FAILED]
__thread_setcancelstate 在进入或退回 **dlgetfileinfo()** 时失败。

[RTLD_ERR_DLGETFILEINFO_MTEXT]
因为库具有多个文本段，所以无法返回文件信息。

[RTLD_ERR_DLGETFILEINFO_MDATA]
因为库具有多个数据段，所以无法返回文件信息。

[RTLD_ERR_INV_DLGETFILEINFO_ARGUMENT]
对 **dlgetfileinfo()** 调用中的参数无效。

[RTLD_ERR_DLGETFILEINFO_IO]
读取文件信息时出现 IO 错误。

另请参阅

dlerrno(3C)、 dlerror(3C)、 dlopen(3C)、 dlopene(3C)、 dlsetlibpath(3C)。

文本和教程：

《HP-UX Linker and Libraries Online User Guide》
（请参阅 **+help** 选项）

《HP-UX Linker and Libraries User's Guide》
（有关订购信息，请参阅 *manuals*(5) ）

名称

dlgetmodinfo() - 检索有关已加载模块（程序或共享库）的信息

概要

cc [*flag*...] *file* ... **-ldl** [*library*] ...

```
#include <dlfcn.h>
uint64_t dlgetmodinfo(int index,
                struct load_module_desc *desc,
                size_t desc_size,
                void *(*read_tgt_mem)(void* buffer,
                                uint64_t ptr,
                                size_t bufsiz,
                                int ident),
                int ident_parm,
                uint64_t load_map_parm);
```

多线程应用信息

在 **libdl.sl** 中是线程安全的，但在 **libxpdl.sl** 中不是线程安全的。

说明

dlgetmodinfo() 是用于向用户授予对动态链接工具的直接访问权限的一系列例行程序之一。 **dlgetmodinfo()** 将根据 *index* （用于指定加载模块在动态加载程序的搜索列表中所在的位置）检索有关加载模块的信息。与 **dlget()** 不同， **dlgetmodinfo()** 可以检索有关另一进程中的加载模块的信息。 *index* 为 **-1** 时将请求有关动态加载程序的信息。 *index* 为 **-2** 时则请求有关程序文件自身的信息。 **dlgetmodinfo()** 例行程序将用匹配的加载模块中的信息填写 *load_module_desc* 。

desc 、 *desc_size* 、 *read_tgt_mem* 、 *ident_parm* 和 *load_map_parm* 参数与 **dlmodinfo()** 中对应的参数相同。请参阅 *dlmodinfo*(3C) 。

返回值

如果成功， **dlgetmodinfo** 将返回由 **dlopen()** 的返回值所定义的共享库的句柄。否则将返回 NULL。返回值的类型将转换为 **uint64_t** 。

错误

如果 **dlgetmodinfo()** 失败，则随后对 **dlerrno()** 的调用返回下列值之一:

[RTLD_ERR_CANT_APPLY_RELOC]	不能在库中进行重定位。
[RTLD_ERR_INV_ADDRESS]	地址无效。
[RTLD_ERR_INV_BUFFER_ARGUMENT]	描述符参数无效。
[RTLD_ERR_INV_DESC_VERSION]	描述符版本无效。

[RTLD_ERR_INV_LIB_INDEX]	加载模块索引无效。
[RTLD_ERR_NO_MEMORY]	内存不足。
[RTLD_ERR_READ_TGT_MEM_FAILED]	**read_tgt_mem** 未能读取目标内存。
[RTLD_ERR_SETCANCELSTATE_FAILED]	**__thread_setcancelstate** 在进入或退回 **dlgetmodinfo()** 时失败。
[RTLD_ERR_SIGENABLE_FAILED]	退出 **dlgetmodinfo()** 时 **sigenable** 失败。
[RTLD_ERR_SIGINHIBIT_FAILED]	进入 **dlgetmodinfo()** 时 **siginhibit** 失败。

作者

dlgetmodinfo() 由 HP 开发。

另请参阅

系统工具

exec(2)	系统加载程序。
ld(1)	调用链接编辑器。

其他信息

a.out(4)	汇编程序、编译程序和链接程序输出。
dlclose(3C)	卸载以前由 **dlopen()** 加载的共享库。
dlerror(3C)	返回 **dld** 所记录的最新错误消息。
dlerrno(3C)	返回 **dld** 所记录的最新错误的错误代码。
dlget(3C)	返回有关已加载模块的信息。
dlgetname(3C)	返回包含加载模块的存储空间的名称。
dlopen(3C)	加载共享库。
dlsym(3C)	获取共享库中符号的地址。
dlmodinfo(3C)	检索有关已加载模块（程序或共享库）的信息。

文本和教程

《HP-UX Linker and Libraries Online User Guide》
（请参阅 **+help** 选项）

《HP-UX Linker and Libraries User's Guide》
（有关订购信息，请参阅 *manuals*(5) ）

名称

dlgetname() - 在给定加载模块描述符的情况下检索加载模块的名称

概要

cc [*flag*]... *file*... **-ldl** [*library*]...

```
#include <dlfcn.h>
char *dlgetname(struct load_module_desc *desc,
        size_t desc_size,
        void *(*read_tgt_mem)(void* buffer,
                unsigned long long ptr,
                size_t bufsiz,
                int ident),
        int ident_parm,
        unsigned long long load_map_parm);
```

多线程应用信息

该例行程序是线程安全的。

说明

dlgetname() 是用于向用户授予对动态链接工具的直接访问权限的一系列例行程序之一。 **dlgetname()** 将返回 *desc* 表示的加载模块的路径名。 *read_tgt_mem* 、 *ident_parm* 和 *load_map_parm* 参数与 **dlmodinfo()** 中对应的参数相同。请参阅 *dlmodinfo*(3C) 。

dlgetname() 的调用者必须复制返回值，以确保不会损坏此返回值。

返回值

如果 *desc* 描述的不是已加载模块，则返回 NULL。可以通过 **dlerror()** 获取更详细的诊断信息。

错误

如果 **dlgetname()** 失败，则随后对 **dlerrno** 的调用返回下列值之一:

[RTLD_ERR_CANT_APPLY_RELOC]	不能在库中进行重定位。
[RTLD_ERR_NO_MEMORY]	内存不足。
[RTLD_ERR_READ_TGT_MEM_FAILED]	**read_tgt_mem** 未能读取目标内存。
[RTLD_ERR_SETCANCELSTATE_FAILED]	进入或退出 **dlgetname()** 时 **__thread_setcancelstate** 失败。
[RTLD_ERR_SIGENABLE_FAILED]	退出 **dlgetname()** 时 **sigenable** 失败。
[RTLD_ERR_SIGINHIBIT_FAILED]	进入 **dlgetname()** 时 **siginhibit** 失败。

作者

dlgetname() 由 HP 开发。

另请参阅

系统工具

exec(2)	系统加载程序。
ld(1)	调用链接编辑器。

其他信息

a.out(4)	汇编程序、编译程序和链接程序输出。
dlclose(3C)	卸载以前由 **dlopen()** 加载的共享库。
dlerror(3C)	返回 **dld** 记录的最新错误消息。
dlerrno(3C)	返回 **dld** 记录的最新错误的错误代码。
dlget(3C)	返回有关已加载模块的信息。
dlmodinfo(3C)	返回有关已加载模块的信息。
dlopen(3C)	加载共享库。
dlsym(3C)	获取共享库中符号的地址。

文本和教程

《HP-UX Linker and Libraries Online User Guide》
（请参阅 **+help** 选项）

《HP-UX Linker and Libraries User's Guide》
（有关订购信息，请参阅 *manuals*(5) ）

名称

dlmodadd() - 注册有关动态生成函数的信息

概要

cc [*flag*]... *file*... **-ldl** [*library*]...

```
#include <dlfcn.h>
void* dlmodadd(void* associate_handle,
        void *func_start,
        size_t func_size,
        void *linkage_ptr,
        void *unwind_info);
```

说明

dlmodadd() 用于注册有关动态生成函数的信息，通过 **dlmodinfo()** 可检索这些信息。可以使用 **dlmodremove()** 接口删除已注册的信息。

associate_handle 是新函数与之关联的现有模块的模块句柄。如果动态加载程序是通过新函数调用的，那么该动态加载程序的行为如同通过关联的模块调用的一样。该句柄必须是由 **dlopen()** 或 **dlget()** 返回的句柄。

func_start 是函数的生成机器代码的起始地址。而不是函数描述符的地址。

func_size 是生成机器代码的字节大小。

linkage_ptr 是函数要使用的 gp（全局指针）值。它可能是关联模块的 gp。即使生成的代码实际上不使用 gp，也必须将其设置为有效值，因为这是跟踪展开信息中的个性例行程序指针时最起码需要具备的条件。

unwind_info 是指向函数的展开信息块开头的指针。展开信息块包含标头单词、展开描述符、个性例行程序指针和特定语言的数据，如基于 Itanium 的系统的文档

"Software Conventions and Runtime Architecture" 中所述。

调用 **dlmodadd()** 注册有关动态生成函数的信息后，动态加载程序将为该函数创建一个展开标头和单条目展开表。展开标头与动态生成函数占用的地址范围相关联。 **dlmodadd()** 例行程序返回一个句柄，作为新添加函数的标识。 **dlmodadd()** 返回的句柄将与 **dlopen()** 和 **dlget()** 返回的句柄共享同一个命名空间，但是在调用 **dlclose()** 或 **dlsym()** 时不可以使用这些句柄。如果 **dlmodinfo()** 是使用 *ip_value* 调用的，而后者属于通过 **dlmodadd()** 注册的函数，那么它将返回 **dlmodadd()** 调用期间指定的 *associate_handle* 。如果 **dlclose()** 卸载了某个库，则将同时删除与卸载的库关联的所有动态生成函数的展开信息。

多线程应用信息

该例行程序是线程安全的。

返回值

如果成功， **dlmodadd()** 会返回一个句柄，作为新添加函数的标识。否则，返回 NULL。

错误

如果 **dlmodadd()** 失败，那么后续对 **dlerrno()** 的调用将会返回下列值之一:

[RTLD_ERR_NO_MEMORY]	内存不足。
[RTLD_ERR_CANT_APPLY_RELOC]	不能在库中进行重定位。
[RTLD_ERR_SIGINHIBIT_FAILED]	进入 **dlmodadd()** 时 **siginhibit** 失败。
[RTLD_ERR_SIGENABLE_FAILED]	退出 **dlmodadd()** 时 **sigenable** 失败。
[RTLD_ERR_INV_DLMODADD_ARGUMENT]	**dlmodadd()** 参数无效。
[RTLD_ERR_SETCANCELSTATE_FAILED]	进入或退出 **dlmodadd()** 时 **__thread_setcancelstate** 失败。

作者

dlmodadd() 由 HP 开发。

另请参阅

ld(1)、dlclose(3C)、dlerrno(3C)、dlerror(3C)、dlget(3C)、dlmodremove(3C)、 dlopen(3C)、dlsym(3C)、a.out(4)、dld.so(5)。

文本和教程

《HP-UX Linker and Libraries Online User Guide》
（请参阅 **+help** 选项）

《HP-UX Linker and Libraries User's Guide》
（有关订购信息，请参阅 *manuals*(5) ）

名称

dlmodinfo() - 检索有关已加载模块（程序或共享库）的信息

概要

cc [*flag*]... *file*... **-ldl** [*library*]...

```
#include <dlfcn.h>
uint64_t dlmodinfo(uint64_t ip_value,
        struct load_module_desc *desc,
        size_t desc_size,
        void *(*read_tgt_mem)(void* buffer,
                uint64_t ptr,
                size_t bufsiz,
                int ident),
        int ident_parm,
        uint64_t load_map_parm);
```

多线程应用信息

在 **libdl.sl** 中是线程安全的，但在 **libxpdl.sl** 中不是线程安全的。

说明

dlmodinfo() 是使用户能够直接访问动态链接设备的一系列例行程序之一。 **dlmodinfo()** 可以通过给定的地址值检索有关某个已加载模块的信息。 **dlmodinfo()** 可搜索当前已加载的所有加载模块，以查找其地址范围（所有已加载段的地址范围）保存了给定地址值的加载模块。 **dlmodinfo()** 例行程序使用匹配的加载模块中的信息填充 *load_module_desc* 。

ip_value 是所请求的库的指令指针值。如果值为 NULL，则 *desc* 包含 **dld** 本身的模块信息。 *desc* 是用户程序分配的内存的缓冲区。动态加载程序将使用模块信息对其进行填充。 *desc_size* 是 *desc* 缓冲区的字节大小。 *read_tgt_mem* 是指向 **dlmodinfo()** 用来检索所需信息的函数的指针。如果该值为 NULL，则动态加载程序将使用自己的内部数据结构来查找适当的加载模块，同时忽略以下两个参数。

ident_parm 只用于将第四个参数传递给 *read_tgt_mem* 。

load_map_parm 仅当通过 *read_tgt_mem* 调用时才使用。它包含加载映射的起始地址。

否则，在搜索过程中，将使用函数指针读取使用以下这些参数的内存：

buffer **dlmodinfo()** 提供的要读入的缓冲区

ptr 要读取的虚拟内存地址

bufsiz 缓冲区的字节大小

ident **dlmodinfo()** 的 *ident_parm* 参数值

如果成功， *read_tgt_mem* 将返回其缓冲区参数的值，否则，将返回 NULL。 *read_tgt_mem* 允许 **dlmodinfo** 在代表某个进程的另一个进程中查找加载模块。为了从 **dlmodinfo()** 所驻留的进程地址空间读取其他进程地址空间中

的内存，调用进程将通过 *read_tgt_mem* 传递回调。 *read_tgt_mem* 中的 *ip_value* 、 *load_map_parm* 和 *ptr* 可能是指向另一进程中的对象的指针。例如，如果某个 32 位程序需要查询某个 64 位程序， *ip_value* 和 *load_map_parm* 应该是 64 位值。任何 32 位指针传入 *ip_value* 或 *load_map_parm* 参数后，用户程序应将其类型转换为 64 位。

如果调用进程使用通过 *read_tgt_mem* 注册的回调调用 **dlmodinfo()** ，则它必须在 **dlmodinfo()** 的 *load_map_parm* 参数中提供目标进程加载映射的起始地址。可以使用目标程序文件中的 **DT_HP_LOAD_MAP** 动态表条目检索该地址。

可通过 **dlmodinfo** 执行跨进程的加载模块操作，例如，向 **ttrace()** 发出一个调用请求。

返回值

如果成功， **dlmodinfo()** 将为 **dlopen()** 的返回值定义的共享库返回一个句柄。否则，将返回 NULL。返回值的类型将转换为 **uint64_t** 。

错误

如果 **dlmodinfo()** 失败，则随后对 **dlerrno()** 进行调用将返回下列值之一:

[RTLD_ERR_CANT_APPLY_RELOC]	不能在库中进行重定位。
[RTLD_ERR_INV_ADDRESS]	地址无效。
[RTLD_ERR_INV_BUFFER_ARGUMENT]	描述符参数无效。
[RTLD_ERR_INV_DESC_VERSION]	描述符版本无效。
[RTLD_ERR_NO_MEMORY]	内存不足。
[RTLD_ERR_READ_TGT_MEM_FAILED]	**read_tgt_mem** 未能读取目标内存。
[RTLD_ERR_SETCANCELSTATE_FAILED]	进入或退出 **dlmodinfo()** 时 **__thread_setcancelstate** 失败。
[RTLD_ERR_SIGENABLE_FAILED]	退出 **dlmodinfo()** 时 **sigenable** 失败。
[RTLD_ERR_SIGINHIBIT_FAILED]	进入 **dlmodinfo()** 时 **siginhibit** 失败。

作者

dlmodinfo 由 HP 开发。

另请参阅

系统工具

exec(2)	系统加载程序。
ld(1)	调用链接编辑器。

其他信息

a.out(4)	汇编程序、编译程序和链接程序输出。
dlclose(3C)	卸载以前由 **dlopen()** 加载的共享库。
dlerror(3C)	返回 **dld** 记录的最新错误消息。

dlerrno(3C)	返回 **dld** 记录的最新错误的错误代码。
dlget(3C)	返回有关已加载模块的信息。
dlgetname(3C)	返回包含加载模块的存储空间的名称。
dlopen(3C)	加载共享库。
dlsym(3C)	获取共享库中符号的地址。

文本和教程

《HP-UX Linker and Libraries Online User Guide》
（请参阅 **+help** 选项）

《HP-UX Linker and Libraries User's Guide》
（有关订购信息，请参阅 *manuals*(5) ）

名称

dlmodremove() - 删除使用 dlmodadd 注册的信息

概要

cc [*flag*]... *file*... **-ldl** [*library*]...

#include <dlfcn.h>
int dlmodremove(void* handle);

多线程应用信息

该例行程序是线程安全的。

说明

dlmodremove() 删除单个动态生成的函数的注册信息。

handle 必须是 **dlmodadd()** 返回的句柄。

dlmodadd() 删除指示函数的注册，并释放 **dlmodadd()** 创建的展开标题和展开表所用的空间。它不会释放展开信息块或生成的代码使用的空间，这些空间已由用户分配且必须由用户释放。

返回值

dlmodremove() 例行程序在成功时返回 **0** 。如果 *handle* 不是有效句柄，则 **dlmodremove()** 返回非零值。

错误

如果 **dlmodremove()** 失败，则随后对 **dlerrno()** 的调用返回下列值之一：

[RTLD_ERR_CANT_APPLY_RELOC]	不能在库中进行重定位。
[RTLD_ERR_INV_HANDLE]	句柄无效。
[RTLD_ERR_NO_MEMORY]	内存不足。
[RTLD_ERR_SETCANCELSTATE_FAILED]	进入或退出 **dlmodremove()** 时 **__thread_setcancelstate** 失败。
[RTLD_ERR_SIGENABLE_FAILED]	退出 **dlmodremove()** 时 **sigenable** 失败。
[RTLD_ERR_SIGINHIBIT_FAILED]	进入 **dlmodremove()** 时 **siginhibit** 失败。

作者

dlmodadd() 由 HP 开发。

另请参阅

ld(1)、dlerrno(3C)、dlerror(3C)、dlmodadd(3C)、a.out(4)、dld.so(5)。

文本和教程

《HP-UX Linker and Libraries Online User Guide》
（请参阅 **+help** 选项）

《HP-UX Linker and Libraries User's Guide》
（有关订购信息，请参阅 *manuals*(5) ）

名称

dlopen() - 打开共享库

概要

备注

对于 Integrity 系统，请参阅 *dlopen_ia*(3C) 。

对于 PA-RISC 系统，请参阅 *dlopen_pa*(3C) 。

使用 **uname** 命令确定您的系统类型。在 Integrity 系统上， **uname -m** 返回 **ia64** 。其他所有值均表示 PA-RISC 系统。

另请参阅

dlopen_ia(3C)、 dlopen_pa(3C)、 uname(1)。

名称

dlopen_ia： dlopen()、 dlopene() - 在 Integrity 系统上打开共享库

概要

命令： **cc** [*flag*]... *cfile*... **-ldl** [*library*]...

#include <dlfcn.h>

void *dlopen(const char *file, int mode);

void *dlopene(const char *file, int mode, struct dlopen_opts *opts);

备注

此联机帮助页介绍了 Integrity 系统上的 **dlopen()** 。有关 HP 9000 系统上的 **dlopen()** ，请参阅 *dlopen_pa*(3C) 。

多线程应用信息

这些例行程序是线程安全的。

注释：动态加载程序 **dld.so** 可使用递归 pthread 互斥锁序列化多线程应用程序中共享库的加载和卸载。有关详细信息，请参阅《HP-UX Linker and Libraries Online User Guide》。

说明

dlopen() 和 **dlopene()** 是允许用户直接访问动态链接设备（使用编译程序上的 **-ldl** 选项或 **ld** 命令行）的系列例行程序的成员。

正在运行的进程可通过 **dlopen()** 使用 *file* 指定的共享对象。而共享对象可以指定要保证正确执行“所需的”其他对象。这些相关对象是由原始对象的 **.dynamic** 段中的 **DT_NEEDED** 条目指定的。而每个所需的对象又可以指定其他所需的对象。调用 **dlopen()** 后，将连同原始对象一起加载所有这些对象。

dlopene() 是 **dlopen()** 的一个扩展，它允许调用方显式指定共享库的文本和数据段的位置（在动态加载库的情况下）。

成功执行 **dlopen()** 或 **dlopene()** 调用后，将为进程返回后续调用 **dlsym()** 和 **dlclose()** 时可能需要使用的 *handle* 。进程不应以任何方式解释此值。

file 用于构建指向对象文件的路径名。如果 *file* 包含斜线字符，则 *file* 参数自身将用作路径名。否则， **dlopen()** 将按以下顺序，在一系列目录中搜索 *file* ：

- 通过调用 **dlsetlibpath()** 设置的动态路径所指定的任何目录。
- 环境变量 **LD_LIBRARY_PATH** 指定的任何目录。
- **SHLIB_PATH** 指定的任何目录。
- 原始程序对象 **.dynamic** 段中的 **DT_RPATH** 条目指定的任何目录
- 32 位模式下的目录 **/usr/lib/hpux32** ，及 64 位模式下的目录 **/usr/lib/hpux64** 。

- 当前工作目录。

如果 *file* 的值为 **0** ， **dlopen()** 将在“全局符号对象”上提供 *handle* 。可以通过此对象访问由原始 **a.out** 、程序启动时随 **a.out** 一起加载的所有对象、通过 **dlopen()** 操作并随 **RTLD_GLOBAL** 标记一起加载的所有对象组成的排序对象集中的符号。如同后面的对象集在执行过程中可以变化一样， *handle* 标识的对象集也可以动态变化。

即使为引用某个对象文件而数次调用 **dlopen()** ，或者即使使用了不同的路径名来引用该文件，最终只能将对象文件的一个副本送入地址空间。

如果将共享对象带入进程的地址空间，该共享对象可包含对符号的引用，而这些符号的地址是在加载对象后才知道的。在可以访问符号之前，必须重定位这些引用。 *mode* 参数可以控制何时开始这些重定位操作，此参数包括以下值：

RTLD_TEXT_PRIVATE
在此 *mode* 下，加载的共享库将其文本段映射为专用。这对于调试是非常有用的。

RTLD_LAZY 在此 *mode* 下，加载对象时只会重定位对数据符号的引用。在首次调用给定的函数后，才会重定位对函数的引用。由于进程不可以引用任何给定共享对象中的所有函数，此 *mode* 可以带来更佳的性能。

RTLD_NOW 在此 *mode* 下，首先加载对象时，将执行所有必要的重定位操作。如果为从来没有引用过的函数执行了重定位，可能在一定程度上使努力白费。但是，对于那些需要在加载对象后，尽快知道执行过程中将要引用的所有符号是否可用的应用程序来说，该操作是十分有用的。

通过需要针对全局符号重定位的 **dlopen()** 加载对象，可以引用原始 **a.out** 、程序启动时加载的任何对象中的符号，也可以从对象自身、相同 **dlopen()** 调用中包含的任何其他对象、指定 **RTLD_GLOBAL** 标记的任何 **dlopen()** 调用中加载的任何对象引用符号。要确定通过 **dlopen()** 调用加载的符号的可见性范围，应使用下列其中一个值对 *mode* 参数执行按位逻辑 OR 运算：

RTLD_GLOBAL 使对象符号可用于其他任何对象的重定位处理。此外，使用 **dlopen(0,***mode***)** 和关联 **dlsym()** 执行符号查找时，可以搜索通过 **RTLD_GLOBAL** 加载的对象。

RTLD_LOCAL 使对象符号只可用于相同的 **dlopen()** 调用中加载的对象的重定位处理。

如果既没有指定 **RTLD_GLOBAL** ，也没有指定 **RTLD_LOCAL** ，则缺省值为 **RTLD_LOCAL** 。

如果在多个 **dlopen()** 调用中指定了 *file* ，则在每个调用上解释 *mode* 。但是请注意，一旦指定 **RTLD_NOW** ，将完成所有重定位，并呈现其他任何冗余的 **RTLD_NOW** 操作，及其他任何不相关的 **RTLD_LAZY** 操作。

同时请注意，一旦指定 **RTLD_GLOBAL** ，只要在地址空间中保留对象（请参阅 *dlclose*(3C) ），无论以前或以后如何指定 **RTLD_LOCAL** ，对象都将保持 **RTLD_GLOBAL** 状态。

要确定可用于 **dlopen()** 调用中加载对象的重定位处理的符号范围，应使用下列其中一个值对 *mode* 参数执行按位逻辑 OR 运算：

RTLD_GROUP 在此 *mode* 下，指定的对象及其相关对象的行为就如同它们是使用 **-B group** （请参阅 *ld*(1)）构建的一样。只有相同的 **dlopen()** 调用中加载的对象的符号才可用于重定位。这可以确保使用相同 **dlopen()** 调用中的符号定义来完成所有重定位。

RTLD_WORLD 在此 *mode* 下，只有全局对象的符号及对象自身的符号才可用于重定位处理。它不使用那些作为 **dlopen()** 调用的一部分加载的其他对象的符号定义。此标记不影响使用 **-B group** （请参阅 *ld*(1) ）构建的对象。

RTLD_PARENT 在此 *mode* 下，调用了 **dlopen()** 的对象的符号也可用于重定位。

dlopen() 的缺省模式为 **RTLD_WORLD|RTLD_GROUP** 。当通过不同的模式加载同一对象时，将对这些标记一起进行逻辑 OR 运算。

下列标记不影响重定位处理，但提供其他功能：

RTLD_NODELETE 在此 *mode* 下，指定的对象及其相关对象的行为就如同它们是使用 **-B nodelete** （请参阅 *ld*(1) ）构建的一样。使用 **dlclose()** 或 **shl_load()** 执行显式卸载将以无提示方式返回成功，且不会将共享库与进程分离。以后，共享库句柄只对 **shl_findsym()** 有效。对于 **dlsym()** 、**dlclose()** 和 **shl_unload()** ，在下一次使用 **shl_load()** 或 **dlclose()** 执行显式加载之前，该句柄将一直处于无效状态。

RTLD_NOLOAD 在此 *mode* 下，不会将指定的对象载入进程的地址空间，但如果进程的地址空间中已存在该对象时，将返回有效句柄。如果指定对象不存在，则返回错误。可以使用 **RTLD_NOLOAD** 查询对象是否存在，也可以使用它来覆盖现有对象的模式。

在重定位活动中，可以使用调用 **dlopen()** 后引入程序的符号。以这种方法引入的符号能够复制由程序或以前的 **dlopen()** 操作所定义的符号。要解析这种情况下可能出现的多义性，应根据符号解析顺序来解析对符号定义的符号引用。

定义了与此相关的两种解析顺序： *load* 和 *dependency* 顺序。

加载顺序使用加载包含了定义的对象时所遵循的临时顺序，在符号定义之间建立排序规则，使得首先加载的定义的优先级高于其后添加的定义。重定位处理使用加载顺序。

相关性顺序使用“宽度优先”顺序，从给定对象开始，然后是其所有相关对象，再后是各相关对象的所有相关对象，将不断重复此顺序，直到符合所有相关性条件为止。除了使用 **0** 值在 *file* 上通过 **dlopen()** 操作而获取的全局符号对象外， **dlsym()** 函数可以对其他对象使用相关性顺序。全局符号对象的 **dlsym()** 操作中可以使用加载顺序。

第一次使某个对象可以通过 **dlopen()** 进行访问时，将以相关性顺序添加此对象及其相关对象。一旦添加了所有对象，即可使用加载顺序执行重定位操作。请注意，如果通过以前的 **dlopen()** 调用，或者通过启动加载了某个对象及其相关对象，则加载顺序和相关性顺序可能会产生不同的结果。

由 **dlopen()** 操作引入的且可以通过 **dlsym()** 使用的符号，是由对象作为全局范围符号“导出”的。对于共享对象来说，这些符号通常是（例如）C 源代码中指定为包含 *extern* 链接的符号。对于 **a.out** ，通常只导出一部分可见符号：尤其是链接了 **a.out** 的共享对象引用的那些符号。

可以使用链接程序（请参阅 *ld*(1)）来控制任何共享对象或 **a.out** 的输出符号的精确集合。

dlopen_opts 结构包括以下成员：

```
struct dlopen_opts {
        long flags;
        char* text_addr;
        char* data_addr;
        };
```

flags 包含由下列值的逻辑 OR 运算定义的加载选项：

RTLD_EXT_TEXT_ADDR
表示提供了共享库文本段的显式基址。

RTLD_EXT_DATA_ADDR
表示提供了共享库专用数据段的显式基址。

RTLD_EXT_DATA_NO_ZERO_FILL
如果设置了此标记， **dlopene()** 将不用零填充数据段的 bss 部分。对于具有很大 bss 部分的库，这可以改善加载时间。此标记仅对 **RTLD_EXT_DATA_ADDR** 有效。

动态加载程序仅访问由 *flags* 字段指定的地址字段。

text_addr 包含共享库文本段的显式基址。

data_addr 包含共享库数据段的显式基址。

text_addr 和 *data_addr* 都必须在 16 字节边界对齐。

调用方可以调用 **dlgetfileinfo()** 来获得为加载段分配内存所需的信息。

dlopene() 的调用方负责分配内存，使之具有相应的权限：

- 对于 *text_addr* ，具有读取、写入和执行 (RWX) 权限。
- 对于 *data_addr* ，具有读取和写入 (RW) 权限。

返回值

如果无法找到、打开或读取 *file* ，或者它不是共享对象，或者加载 *file* 或重定位其符号引用的过程中出现错误，**dlopen()** 将返回 NULL。可以通过 **dlerror()** 或 **dlerrno()** 获取更详细的诊断信息。

错误

如果 **dlopen()** 或 **dlopene()** 失败，则随后对 **dlerrno()** 的调用返回下列值之一：

[RTLD_ERR_ARCH_EXT_NOT_SUPPORTED]
硬件中不存在所需的扩展，无法加载库。

[RTLD_ERR_BAD_ABI1] 64 位程序找到 32 位共享库。

[RTLD_ERR_BAD_ABI2] 32 位程序找到 64 位共享库。

[RTLD_ERR_BAD_DLL] 库无效。

[RTLD_ERR_BAD_DLL_ALIGNMENT]
库无效：对齐错误。

[RTLD_ERR_BAD_DLL_BAD_MACHINE]
库无效：计算机类型错误。

[RTLD_ERR_BAD_DLL_BAD_OBJFILE]
库无效：目标文件类型错误。

[RTLD_ERR_BAD_DLL_BAD_PHDR]
库无效：缺少程序头。

[RTLD_ERR_BAD_DLL_MAGIC_NUM]
库无效：幻数错误。

[RTLD_ERR_BAD_DLL_NO_SYMTAB]
库无效：缺少符号表。

[RTLD_ERR_BAD_DLL_SEGMENT_COUNT]
库需要共享的固定地址，但具有多个数据段。

[RTLD_ERR_BAD_ELF_VER] 库中未知的 ELF 版本。

[RTLD_ERR_BAD_RELOC] 未知的重定位类型。

[RTLD_ERR_CANT_APPLY_RELOC]
不能在库中进行重定位。

[RTLD_ERR_CODE_UNSAT] 库中不符合要求的代码符号。

[RTLD_ERR_DATA_UNSAT] 库中不符合要求的数据符号。

[RTLD_ERR_DLDD_COMM_FAILURE]
库需要共享的固定地址，但无法从 **dldd** 获得它。

[RTLD_ERR_DLOPEN_BAD_FLAGS]
dlopen() 的标记无效。

[RTLD_ERR_DLOPEN_TLS_LIB] 无法 **dlopen()** 库，因为它包含 TLS 数据。

[RTLD_ERR_DLOPENE_BAD_ADDR]
段的加载地址无效（仅限 **dlopene()** ）。

[RTLD_ERR_DLOPENE_NO_EXEC_PERM]
共享库缺少执行权限（仅限 **dlopene()** ）。

[RTLD_ERR_DYN_FILTER_STLS_REF]
库包含对动态加载库中定义的符号的静态 TLS 引用。

[RTLD_ERR_FILTER_TLS] 无法打开过滤的库：TLS 大小超出过滤器中记录的大小。

[RTLD_ERR_INTERNAL_ERROR]
dld 中遇到内部错误。

[RTLD_ERR_IO] 映射库时发生 I/O 错误。

[RTLD_ERR_LIB_OPEN] 无法找到库。

[RTLD_ERR_MMAP_FAILED] 针对库调用 **mmap()** 失败。

[RTLD_ERR_MPROTECT_FAILED]
针对库调用 **mprotect()** 失败。

[RTLD_ERR_NO_MEMORY] 内存不足。

[RTLD_ERR_NOMMAP_FAILED]
加载库时遇到错误（仅限 **dlopene()** ）。

[RTLD_ERR_NON_TLS_RELOC_TO_TLS_SYM]
针对 TLS 符号进行非线程特定的重定位引用。

[RTLD_ERR_OPEN] 无法打开库。

[RTLD_ERR_PREALLOC_ADDR_NOT_USE]
需要使用固定地址，不能使用预配置地址来映射库（仅限 **dlopene()** ）。

[RTLD_ERR_SETCANCELSTATE_FAILED]
进入或退出 dld API 时 **__thread_setcancelstate** 失败。

[RTLD_ERR_SIGENABLE_FAILED]
退出 dld API 时 **sigenable** 失败。

[RTLD_ERR_SIGINHIBIT_FAILED]
进入 dld API 时 **siginhibit** 失败。

[RTLD_ERR_TPREL_NON_TLS_SYM]
针对非 TLS 符号进行 TPREL 重定位。

举例

以下示例说明如何使用 **dlopene()** 加载具有显式数据段地址的共享库。为简化示例，省略了错误检查。

```
#include <dlfcn.h>
#include <sys/mman.h>
```

```
int main() {
   struct dlfileinfo info;
   void *handle;
   struct dlopen_opts opts;
   int status;

   memset(&info, 0, sizeof(info));
   memset(&opts, 0, sizeof(opts));

   /* Get file info */
   status = dlgetfileinfo("libfoo.so", sizeof(info), &info);

   opts.flags = RTLD_EXT_DATA_ADDR;
   /* allocate memory for the data segment */
   opts.data_addr = (char*) mmap(0, info.data_size,
                        PROT_READ|PROT_WRITE,
                        MAP_SHARED|MAP_ANONYMOUS,
                        -1, 0);

   /* call dlopene */
   handle = dlopene("libfoo.so", RTLD_NOW|RTLD_GLOBAL, &opts);

   /* Insert user code to use library */

   /* close library */
   status = dlclose(handle);

   /* free memory */
   munmap(opts.data_addr, info.data_size);
}
```

警告

环境变量 **LD_LIBRARY_PATH** 和 **SHLIB_PATH** 应包含目录的逗号分隔列表，其格式与 **PATH** 变量（请参阅 *sh*(1) ）相同。如果进程的实际用户 ID 不同于其有效的用户 ID，或者其实际组 ID 不同于其有效的组 ID（请参阅 *exec*(2) ），将忽略 **LD_LIBRARY_PATH** 和 **SHLIB_PATH** 。

如果指定 **+compat** 选项，搜索相关库时，将忽略 **LD_LIBRARY_PATH** 和嵌入了 **+b** 的路径。

使用外部库附属件构建共享库时，请保持谨慎。不应该将包含线程本地存储 (TLS) 并使用静态 TLS 模型的任何库用作附属件。有关详细信息，请参阅 *dld.so*(5) 中的“线程本地存储”一节。

如果相关库包含 TLS，并且是用静态 TLS 模型构建的，同时，在程序启动过程中未加载该相关库（也就是说，没有针对可执行文件完成链接），则动态加载程序将无法执行操作。为避免错误，可以使用 **+tls=dynamic** 编译程序选项来重新编译库。

另请参阅

cc(1)、 ld(1)、 sh(1)、 exec(2)、 dlclose(3C)、 dlerrno(3C)、 dlerror(3C)、 dlgetfileinfo(3C)、 dlsetlibpath(3C)、 dlsym(3C)。

文本和教程

《HP-UX Linker and Libraries Online User Guide》
（请参阅 **+help** 选项）

《HP-UX Linker and Libraries User's Guide》

名称

dlopen_pa： dlopen()、 dlopene() - 打开 HP 9000 共享库；打开具有显式加载地址的 HP 9000 64 位共享库

概要

命令： **cc** [*flag*]... *cfile*... **-ldl** [*library*]...

#include <dlfcn.h>

void *dlopen(const char *file, int mode);

void *dlopene(const char *file, int mode, struct dlopen_opts *opts);

备注

此联机帮助页介绍了 HP 9000 系统上的 **dlopen()** 。有关 Integrity 系统上的 **dlopen()** ，请参阅 *dlopen_ia*(3C) 。

多线程应用信息

这些例行程序是线程安全的。

注释：动态加载程序 **dld.sl** 可使用递归 pthread 互斥锁来序列化多线程应用程序中共享库的加载和卸载。有关详细信息，请参阅《HP-UX Linker and Libraries Online User Guide》。

说明

dlopen() 和 **dlopene()** 是允许用户直接访问动态链接设备（使用编译程序上的 **-ldl** 选项或 **ld** 命令行）的系列例行程序的成员。

正在运行的进程可通过 **dlopen()** 使用 *file* 指定的共享对象。而共享对象可以指定要保证正确执行“所需的”其他对象。这些相关对象是由原始对象的 **.dynamic** 段中的 **DT_NEEDED** 条目指定的。而每个所需的对象又可以指定其他所需的对象。调用 **dlopen()** 后，将连同原始对象一起加载所有这些对象。

dlopene() 是 **dlopen()** 的一个扩展，它允许调用方显式指定共享库的文本和数据段的位置（在动态加载库的情况下）。

成功执行 **dlopen()** 或 **dlopene()** 调用后，将为进程返回后续调用 **dlsym()** 和 **dlclose()** 时可能需要使用的 *handle* 。进程不应以任何方式解释此值。

file 用于构建指向对象文件的路径名。如果 *file* 包含斜线字符，则 *file* 参数自身将用作路径名。否则， **dlopen()** 将按以下顺序，在一系列目录中搜索 *file* ：

- 通过调用 **dlsetlibpath()** 设置的动态路径所指定的任何目录。
- 环境变量 **LD_LIBRARY_PATH** 指定的任何目录。
- **SHLIB_PATH** 指定的任何目录。
- 对于 ELF 应用程序，表示原始程序对象的 **.dynamic** 段中的 **DT_RPATH** 条目指定的任何目录

- 32 位模式下的目录 **/usr/lib** 和 **usr/ccs/lib** ，及 64 位模式下的目录 **/usr/lib/pa20_64** 和 **/usr/ccs/lib/pa20_64**。
- 当前工作目录。

SHLIB_PATH 的使用是静态的。这表示当程序启动时，动态加载程序 **dld** 将使用环境变量 **SHLIB_PATH** 中的值。程序执行过程中对 **SHLIB_PATH** 所做的任何更改均无效。 **dld_getenv()** 函数可以删除这些限制。可以通过 **dld_getenv()** 更改 **SHLIB_PATH** 值（使用 **putenv** ），及调用 **dld_getenv()** ，以便 **dld** 读取当前的 **SHLIB_PATH** 值。后续库搜索时可以使用这个新的 **SHLIB_PATH** 值。

通过 **shl_load()** 或 **dlopen()** 加载共享库时可以使用 **dld_getenv()** 。如前文所述，应通过 **chatr +s enable** 选项启用 **SHLIB_PATH** 搜索功能。如果是操作 **shl_load()** ，则应使用 **DYNAMIC_PATH** 标记。

以下示例说明了 **dld_getenv()** 函数：

```
#include <stdio.h>
#include <dl.h>
main(int argc, char **argv) {
char * str;
shl_t shl;
void (*fptr)(void);
char *strpath[267];
sprintf(strpath,  %s%s'', SHLIB_PATH= ,argv[1]);
putenv(strpath);
dld_getenv();
shl = shl_load( a.sl'', BIND_IMMEDIATE | DYNAMIC_PATH, 0);
if(!shl) {
printf("shl_load failed\n");
exit(-1);
} }
```

如果 *file* 的值为 **0** ， **dlopen()** 将在“全局符号对象”上提供 *handle* 。可以通过此对象访问由原始 **a.out** 、程序启动时随 **a.out** 一起加载的所有对象、通过 **dlopen()** 操作并随 **RTLD_GLOBAL** 标记一起加载的所有对象组成的排序对象集中的符号。如同后面的对象集在执行过程中可以变化一样， *handle* 标识的对象集也可以动态变化。

即使为引用某个对象文件而数次调用 **dlopen()** ，或者即使使用了不同的路径名来引用该文件，最终只能将对象文件的一个副本送入地址空间。

如果将共享对象带入进程的地址空间，该共享对象可包含对符号的引用，而这些符号的地址是在加载对象后才知道的。在可以访问符号之前，必须重定位这些引用。 *mode* 参数可以控制何时开始这些重定位操作，此参数包括以下值：

RTLD_TEXT_PRIVATE

在此 *mode* 下，加载的共享库将其文本段映射为专用。这对于调试是非常有用的。

RTLD_LAZY 在此 *mode* 下，加载对象时只会重定位对数据符号的引用。在首次调用给定的函数后，才会重定位对函数的引用。由于进程不可以引用任何给定共享对象中的所有函数，此 *mode* 可以带来更佳的性能。

RTLD_NOW 在此 *mode* 下，首次加载对象时，将执行所有必要的重定位操作。如果为从来没有引用过的函数执行了重定位，可能在一定程度上使努力白费。但是，对于那些需要在加载对象后，尽快知道执行过程中将要引用的所有符号是否可用的应用程序来说，该操作是十分有用的。

通过需要针对全局符号重定位的 **dlopen()** 加载对象，可以引用原始 **a.out** 、程序启动时加载的任何对象中的符号，也可以从对象自身、相同 **dlopen()** 调用中包含的任何其他对象、指定 **RTLD_GLOBAL** 标记的任何 **dlopen()** 调用中加载的任何对象引用符号。要确定通过 **dlopen()** 调用加载的符号的可见性范围，应使用下列其中一个值对 *mode* 参数执行按位逻辑 OR 运算：

RTLD_GLOBAL 使对象符号可用于其他任何对象的重定位处理。此外，使用 **dlopen(0,***mode***)** 和关联 **dlsym()** 执行符号查找时，可以搜索通过 **RTLD_GLOBAL** 加载的对象。

RTLD_LOCAL 使对象符号只可用于相同的 **dlopen()** 调用中加载的对象的重定位处理。

如果既没有指定 **RTLD_GLOBAL** ，也没有指定 **RTLD_LOCAL** ，则缺省值为 **RTLD_LOCAL** 。

如果在多个 **dlopen()** 调用中指定了 *file* ，则在每个调用上解释 *mode* 。但是请注意，一旦指定 **RTLD_NOW** ，将完成所有重定位，并呈现其他任何冗余的 **RTLD_NOW** 操作，及其他任何不相关的 **RTLD_LAZY** 操作。同时请注意，一旦指定 **RTLD_GLOBAL** ，只要在地址空间中保留对象（请参阅 *dlclose*(3C) ），无论以前或以后如何指定 **RTLD_LOCAL** ，对象都将保持 **RTLD_GLOBAL** 状态。

要确定可用于 **dlopen()** 调用中加载对象的重定位处理的符号范围，应使用下列其中一个值对 *mode* 参数执行按位逻辑 OR 运算：

RTLD_GROUP 在此 *mode* 下，指定的对象及其相关对象的行为就如同它们是使用 **-B group** （请参阅 *ld*(1)）构建的一样。只有相同的 **dlopen()** 调用中加载的对象的符号才可用于重定位。这可以确保使用相同 **dlopen()** 调用中的符号定义来完成所有重定位。

RTLD_WORLD 在此 *mode* 下，只有全局对象的符号及对象自身的符号才可用于重定位处理。它不使用那些作为 **dlopen()** 调用的一部分加载的其他对象的符号定义。此标记不影响使用 **-B group** （请参阅 *ld*(1) ）构建的对象。

RTLD_PARENT 在此 *mode* 下，调用了 **dlopen()** 的对象的符号也可用于重定位。

当前只有 64 位应用程序支持 **RTLD_GROUP** 、 **RTLD_WORLD** 和 **RTLD_PARENT** 模式。

dlopen() 的缺省模式为 **RTLD_WORLD|RTLD_GROUP** 。当通过不同的模式加载同一对象时，将对这些标记一起进行逻辑 OR 运算。

下列标记不影响重定位处理，但提供其他功能：

RTLD_NODELETE 在此 *mode* 下，指定的对象及其相关对象的行为就如同它们是使用 **-B nodelete** （请参阅 *ld*(1) ）构建的一样。使用 **dlclose()** 或 **shl_load()** 执行显式卸载将以无提示方式返回成功，且不会将共享库与进程分离。以后，共享库句柄只对 **shl_findsym()** 有效。对于 **dlsym()** 、**dlclose()** 和 **shl_unload()** ，在下一次使用 **shl_load()** 或 **dlclose()** 执行显式加载之前，该句柄将一直处于无效状态。

RTLD_NOLOAD 在此 *mode* 下，不会将指定的对象载入进程的地址空间，但如果进程的地址空间中已存在该对象时，将返回有效句柄。如果指定对象不存在，则返回错误。可以使用 **RTLD_NOLOAD** 查询对象是否存在，也可以使用它来覆盖现有对象的模式。

在重定位活动中，可以使用调用 **dlopen()** 后引入程序的符号。以这种方法引入的符号能够复制由程序或以前的 **dlopen()** 操作所定义的符号。要解析这种情况下可能出现的多义性，应根据符号解析顺序来解析对符号定义的符号引用。定义了与此相关的两种解析顺序： *load* 和 *dependency* 顺序。加载顺序使用加载包含了定义的对象时所遵循的临时顺序，在符号定义之间建立排序规则，使得首先加载的定义的优先级高于其后添加的定义。重定位处理使用加载顺序。相关性顺序使用“宽度优先”顺序，从给定对象开始，然后是其所有相关对象，再后是各相关对象的所有相关对象，将不断重复此顺序，直到符合所有相关性条件为止。除了使用 **0** 值在 *file* 上通过 **dlopen()** 操作而获取的全局符号对象外， **dlsym()** 函数可以对其他对象使用相关性顺序。全局符号对象的 **dlsym()** 操作中可以使用加载顺序。

第一次使某个对象可以通过 **dlopen()** 进行访问时，将以相关性顺序添加此对象及其相关对象。一旦添加了所有对象，即可使用加载顺序执行重定位操作。请注意，如果通过以前的 **dlopen()** 调用，或者通过启动加载了某个对象及其相关对象，则加载顺序和相关性顺序可能会产生不同的结果。

由 **dlopen()** 操作引入的且可以通过 **dlsym()** 使用的符号，是被对象作为全局范围符号“导出的”那些符号。对于共享对象来说，这些符号通常是（例如）C 语言源代码中指定为包含 *extern* 链接的符号。对于 **a.out** ，通常只导出一部分可见符号：尤其是链接了 **a.out** 的共享对象引用的那些符号。可以使用链接程序（请参阅 *ld*(1) ）来控制任何共享对象或 **a.out** 的输出符号的确切集合。

HP 9000 64 位 dlopene()

dlopen_opts 结构包括以下成员：

```
struct dlopen_opts {
        long flags;
        char* text_addr;
        char* data_addr;
        };
```

flags 包含由下列值的逻辑 OR 运算定义的加载选项：

RTLD_EXT_TEXT_ADDR

表示提供了共享库文本段的显式基址。

RTLD_EXT_DATA_ADDR

表示提供了共享库专用数据段的显式基址。

RTLD_EXT_DATA_NO_ZERO_FILL

如果设置了此标记， **dlopene()** 将不用零填充数据段的 bss 部分。对于具有很大 bss 部分的库，这可以改善加载时间。此标记仅对 **RTLD_EXT_DATA_ADDR** 有效。

动态加载程序仅访问由 *flags* 字段指定的地址字段。

text_addr 包含共享库文本段的显式基址。

data_addr 包含共享库数据段的显式基址。

text_addr 和 *data_addr* 都必须在 16 字节边界对齐。

调用方可以调用 **dlgetfileinfo()** 来获得为加载段分配内存所需的信息。

dlopene() 的调用方负责分配内存，使之具有相应的权限：

- 对于 *text_addr* ，具有读取、写入和执行 (RWX) 权限。
- 对于 *data_addr* ，具有读取和写入 (RW) 权限。

返回值

如果无法找到、打开或读取 *file* ，或者它不是共享对象，或者加载 *file* 或重定位其符号引用的过程中出现错误，**dlopen()** 将返回 NULL。可以通过 **dlerror()** 或 **dlerrno()** （仅限 64 位）获取更详细的诊断信息。

错误

如果 **dlopen()** 或 **dlopene()** 失败，则随后对 **dlerrno()** 的调用返回下列值之一：

[RTLD_ERR_BAD_ABI1] 64 位程序找到 32 位共享库。

[RTLD_ERR_BAD_ABI2] 32 位程序找到 64 位共享库。

[RTLD_ERR_BAD_DLL] 库无效。

[RTLD_ERR_BAD_DLL_ALIGNMENT]
库无效：对齐错误。

[RTLD_ERR_BAD_DLL_BAD_MACHINE]
库无效：计算机类型错误。

[RTLD_ERR_BAD_DLL_BAD_PHDR]
库无效：缺少程序头。

[RTLD_ERR_BAD_DLL_BAD_OBJFILE]
库无效：目标文件类型错误。

[RTLD_ERR_BAD_DLL_MAGIC_NUM]
库无效：幻数错误。

[RTLD_ERR_BAD_DLL_NO_SYMTAB]
库无效：缺少符号表。

[RTLD_ERR_BAD_DLL_SEGMENT_COUNT]
库需要共享的固定地址，但具有多个数据段。

[RTLD_ERR_BAD_ELF_VER] 库中未知的 ELF 版本。

[RTLD_ERR_BAD_RELOC] 未知的重定位类型。

[RTLD_ERR_CANT_APPLY_RELOC]
不能在库中进行重定位。

[RTLD_ERR_CODE_UNSAT] 库中不符合要求的代码符号。

[RTLD_ERR_DATA_UNSAT] 库中不符合要求的数据符号。

[RTLD_ERR_DLDD_COMM_FAILURE]
库需要共享的固定地址，但无法从 **dldd** 获得它。

[RTLD_ERR_DLOPEN_BAD_FLAGS]
dlopen() 的标记无效。

[RTLD_ERR_DLOPEN_TLS_LIB] 无法 **dlopen()** 库，因为它包含 TLS 数据。

[RTLD_ERR_DLOPENE_BAD_ADDR]
段的加载地址无效（仅限 **dlopene()** ）。

[RTLD_ERR_DLOPENE_NO_EXEC_PERM]
共享库缺少执行权限（仅限 **dlopene()** ）。

[RTLD_ERR_DYN_FILTER_STLS_REF]
库包含对动态加载库中定义的符号的静态 TLS 引用。

[RTLD_ERR_FILTER_TLS] 无法打开过滤的库：TLS 大小超出过滤器中记录的大小。

[RTLD_ERR_INTERNAL_ERROR]
dld 中遇到内部错误。

[RTLD_ERR_IO] 映射库时发生 I/O 错误。

[RTLD_ERR_LIB_OPEN] 无法找到库。

[RTLD_ERR_MMAP_FAILED] 针对库调用 **mmap()** 失败。

[RTLD_ERR_MPROTECT_FAILED]
针对库调用 **mprotect()** 失败。

[RTLD_ERR_NO_MEMORY] 内存不足。

[RTLD_ERR_NOMMAP_FAILED]
加载库时遇到错误（仅限 **dlopene()** ）。

[RTLD_ERR_NON_TLS_RELOC_TO_TLS_SYM]
针对 TLS 符号进行非线程特定的重定位引用。

[RTLD_ERR_OPEN] 无法打开库。

[RTLD_ERR_PREALLOC_ADDR_NOT_USE]
需要使用固定地址，不能使用预配置地址来映射库（仅限 **dlopene()** ）。

[RTLD_ERR_SETCANCELSTATE_FAILED]
进入或退出 dld API 时 **__thread_setcancelstate** 失败。

[RTLD_ERR_SIGENABLE_FAILED]
退出 dld API 时 **sigenable** 失败。

[RTLD_ERR_SIGINHIBIT_FAILED]
进入 dld API 时 **siginhibit** 失败。

[RTLD_ERR_TPREL_NON_TLS_SYM]
针对非 TLS 符号进行 TPREL 重定位。

举例

以下示例说明如何使用 **dlopene()** 加载具有显式数据段地址的共享库。为简化示例，省略了错误检查。

```
#include <dlfcn.h>
#include <sys/mman.h>

int main() {
   struct dlfileinfo info;
   void *handle;
   struct dlopen_opts opts;
   int status;

   memset(&info, 0, sizeof(info));
   memset(&opts, 0, sizeof(opts));

   /* Get file info */
   status = dlgetfileinfo("libfoo.so", sizeof(info), &info);
```

```
    opts.flags = RTLD_EXT_DATA_ADDR;
    /* allocate memory for the data segment */
    opts.data_addr = (char*) mmap(0, info.data_size,
                        PROT_READ|PROT_WRITE,
                        MAP_SHARED|MAP_ANONYMOUS,
                        -1, 0);

    /* call dlopene */
    handle = dlopene("libfoo.so", RTLD_NOW|RTLD_GLOBAL, &opts);

    /* Insert user code to use library */

    /* close library */
    status = dlclose(handle);

    /* free memory */
    munmap(opts.data_addr, info.data_size);
}
```

警告

在 64 位模式下，环境变量 **LD_LIBRARY_PATH** 和 **SHLIB_PATH** 应包含目录的逗号分隔列表，其格式与 **PATH** 变量（请参阅 *sh*(1) ）相同。如果进程的实际用户 ID 不同于其有效的用户 ID，或者其实际组 ID 不同于其有效的组 ID（请参阅 *exec*(2) ），将忽略 **LD_LIBRARY_PATH** 和 **SHLIB_PATH** 。

在 64 位模式下，如果指定了 **+compat** 选项，在搜索相关库时将忽略 **LD_LIBRARY_PATH** 和嵌入了 **+b** 的路径。

使用外部库附属件构建共享库时请保持谨慎。不应该将任何包含线程本地存储 (TLS) 的库用作附属件。如果相关库包含 TLS，且在程序启动过程中未加载该库（也就是说，没有与可执行文件建立链接），动态加载程序将无法执行操作。

另请参阅

cc(1)、 ld(1)、 sh(1)、 exec(2)、 dlclose(3C)、 dlerrno(3C)、 dlerror(3C)、 dlsym(3C)、 dlgetfileinfo(3C)、 dlsetlibpath(3C)。

文本和教程

《HP-UX Linker and Libraries Online User Guide》
（请参阅 **+help** 选项）

《HP-UX Linker and Libraries User's Guide》

名称

dlsetlibpath() - 设置用于定位共享库的动态搜索路径

概要

cc [*flag*]... *file*... **-ldl** [*library*]...

#include <dlfcn.h>

int dlsetlibpath(const char *libpath, int flags)

多线程应用信息

该例行程序是线程安全的。对 **dlsetlibpath()** 的调用会影响随后在任何线程上对 **dlopen()** 、 **dlopene()** 和 **dlgetfileinfo()** 所有调用。

说明

dlsetlibpath() 是允许用户直接访问动态链接设备（使用编译程序或 **ld** 命令行上的 **-ldl** 选项）的一组例行程序。**dlsetlibpath()** 设置 **dlopen()** 、 **dlopene()** 和 **dlgetfileinfo()** 用来定位共享库的动态搜索路径。

libpath 是动态搜索路径。它是用冒号 (**:**) 分隔的一个或多个路径名的列表。

当搜索库时，动态加载程序按以下缺省顺序使用搜索路径：

1. 对 **dlsetlibpath()** 的调用中指定的动态搜索路径。
2. **LD_LIBRARY_PATH** 环境变量。
3. **SHLIB_PATH** 环境变量。
4. 对 **dlopen()** 、 **dlopene()** 或 **dlgetfileinfo()** 的调用中指定库的调用模块（可执行程序或共享库）的嵌入式路径。对于相关的库，使用库（该库将其指定为相关库）的嵌入式路径。
5. 标准库路径。
6. 当前工作目录（仅适用于在对 **dlopen()** 、 **dlopene()** 和 **dlgetfileinfo()** 的调用中指定的不是其相关库的库）。

（有关搜索路径和可更改上述顺序的选项的其他信息，请参阅 *dld.so*(5) ）。

如果将下列标记值进行 OR 运算，则将 *flags* 设置这些标记值中一个或多个，就可以禁用这些路径的任意组合。如果设置了以下值，则动态加载程序不搜索指定的位置：

RTLD_FLAG_DISABLE_DYNAMIC_PATH
动态搜索路径中指定的目录。

RTLD_FLAG_DISABLE_LD_LIBRARY_PATH
LD_LIBRARY_PATH 环境变量中指定的目录。

RTLD_FLAG_DISABLE_SHLIB_PATH
SHLIB_PATH 环境变量中指定的目录。

RTLD_FLAG_DISABLE_EMBEDDED_PATH
嵌入式路径中指定的目录。

RTLD_FLAG_DISABLE_STD_PATH 标准库目录。

RTLD_FLAG_DISABLE_CWD_PATH 当前工作目录。

可以使用 OR 运算符连接单个标记来禁用多个搜索路径：

```
flags = RTLD_FLAG_DISABLE_STD_PATH | RTLD_FLAG_DISABLE_CWD_PATH
```

可以通过将 *flags* 设置为禁用搜索路径的标记值的补集来启用单个搜索路径：

```
flags = ~RTLD_FLAG_DISABLE_DYNAMIC_PATH
```

举例

下面的示例说明了使用 **dlsetlibpath()** 设置动态搜索路径并禁用其他搜索路径。为简化示例，省略了错误检查。

```
#include <dlfcn.h>

int main() {
   void *handle;
   int   status;
   int   flags;

   /* Set dynamic search path and disable the embedded
    * path and the standard library directory.
    */
   flags = RTLD_FLAG_DISABLE_EMBEDDED_PATH |
         RTLD_FLAG_DISABLE_STD_PATH;
   status = dlsetlibpath("/opt/lib:/opt/usr/lib", flags);

   /* Call dlopen to load a library using the dynamic
    * search path.
    */
   handle = dlopen("mylib.so", RTLD_LAZY);

   /* Remove the dynamic search path and reenable all
    * disabled search paths.
    */
   status = dlsetlibpath(NULL, 0);
```

```
}
```

返回值

如果成功， **dlsetlibpath()** 返回 **0** ，否则返回非零值。可通过 **dlerror()** 或 **dlerrno()** 获得更详细的诊断信息。

错误

如果 **dlsetlibpath()** 失败，则随后对 **dlerrno()** 进行调用将返回下列值之一：

[RTLD_ERR_NO_MEMORY] 无法分配动态内存。

[RTLD_ERR_CANT_APPLY_RELOC]
在解析对 **dlsetlibpath()** 的调用时，未能应用重定位。

[RTLD_ERR_SIGINHIBIT_FAILED]
进入 **dlsetlibpath()** 时 **siginhibit** 调用失败。

[RTLD_ERR_SIGENABLE_FAILED]
退出 **dlsetlibpath()** 时 **sigenable** 调用失败。

[RTLD_ERR_SETCANCELSTATE_FAILED]
进入或退出 **dlsetlibpath()** 时 **__thread_setcancelstate** 失败。

[RTLD_ERR_INV_DLSETLIBPATH_ARGUMENT]
对 **dlsetlibpath()** 调用中的参数无效。

另请参阅

dlopen(3C)、dlopene(3C)、dlgetfileinfo(3C)、dlerrno(3C)、dlerror(3C)、 dld.so(5)。

文本和教程：

《HP-UX Linker and Libraries Online User Guide》
（请参阅 **+help** 选项）

《HP-UX Linker and Libraries User's Guide》
（有关订购信息，请参阅 *manuals*(5) ）

名称

dlsym() - 获取共享库中某个符号的地址

概要

cc [*flag*]... *file*... **-ldl** [*library*]...

#include <dlfcn.h>
void *dlsym(void *handle, const char *name);

多线程应用信息

该例行程序是线程安全的。

说明

dlsym() 是使用户能够直接访问动态链接设备（在编译程序或 **ld** 命令行上使用选项 **-ldl** ）的一系列例行程序之一。进程可通过 **dlsym()** 获取 **dlopen()** 以前打开的共享库中定义的某个符号的地址。 *handle* 要么是调用 **dlopen()** 后返回的值，要么是专用标记 **RTLD_NEXT** 、 **RTLD_SELF** 和 **RTLD_DEFAULT** 中的一个。对于前一种情况，不得使用 **dlclose()** 关闭对应的共享对象。 *name* 是符号的字符型字符串形式的名称。

dlsym() 可以在加载 *handle* （请参阅 *dlopen*(3C) ）引用的对象后自动加载的所有共享对象中，搜索指定的符号。

如果 *handle* 为 **RTLD_NEXT** ，搜索将从调用 **dlsym()** 所在的那个对象后面的“下一个”对象开始。将使用 *load* 顺序符号解析算法（请参阅 *dlopen*(3C) ）搜索对象。“下一个”对象，及所有其他搜索的对象，不是在全局范围内（这是因为它们是在启动时加载的，或者是使用 **RTLD_GLOBAL** 标记执行的 **dlopen()** 操作的一部分加载的），就是与加载 **dlsym()** 调用方相同的 **dlopen()** 操作加载的对象。

如果 *handle* 为 **RTLD_SELF** ，搜索将从调用 **dlsym()** 所在的那个对象开始。将使用 *load* 顺序符号解析算法搜索对象。

如果 *handle* 为 **RTLD_DEFAULT** ，那么将在调用 **dlsym()** 的对象的范围内完成符号搜索。例如，如果调用方对象是使用 **RTLD_GROUP** 执行 **dlopen()** （请参阅 *dlopen*(3C) ）后加载的，则不搜索不是与该调用方对象相同的 **dlopen** 调用中加载的对象中的符号。

返回值

如果 *handle* 不引用 **dlopen()** 打开的有效对象，或者在与 *handle* 关联的任何对象中找不到指定的符号，那么 **dlsym()** 将返回 NULL。可以通过 **dlerror()** 获取更详细的诊断信息。

错误

如果 **dlsym()** 失败，则随后对 **dlerrno()** 的调用返回下列值之一:

[RTLD_ERR_CANT_APPLY_RELOC]	不能在库中进行重定位。
[RTLD_ERR_INTERNAL_ERROR]	**dlsym()** 中遇到内部错误。
[RTLD_ERR_INV_HANDLE]	句柄无效。
[RTLD_ERR_INV_NEXT_HANDLE]	*liblist* 结尾的 **RTLD_NEXT** 参数无效。

[RTLD_ERR_NO_MEMORY]	内存不足。
[RTLD_ERR_SETCANCELSTATE_FAILED]	进入或退出 **dlsym()** 时 **__thread_setcancelstate** 失败。
[RTLD_ERR_SIGENABLE_FAILED]	退出 **dlsym()** 时 **sigenable** 失败。
[RTLD_ERR_SIGINHIBIT_FAILED]	进入 **dlsym()** 时 **siginhibit** 失败。
[RTLD_ERR_UNKNOWN_SYMBOL]	未知符号。

实际应用信息

可以使用 **RTLD_NEXT** 导航为通过 *interposition* 所创建的多个定义的符号，有针对性地创建的层次结构。例如，某个程序需要为嵌入了某些统计信息（收集了有关内存分配的信息）的 **malloc()** 的创建实现，那么，这样的实现可以定义自己的 **malloc()** ，以收集必要的信息，然后同时使用 **dlsym()** 和 **RTLD_NEXT** 查找“实际的” **malloc()**，以执行实际内存分配。当然，这个“实际的” **malloc()** 可能是另一个用户定义的接口，该接口先添加自己的值，然后使用 **RTLD_NEXT** 查找系统 **malloc()** 。

举例

以下示例说明如何使用 **dlopen()**

和 **dlsym()** 访问函数对象或数据对象。为简化示例，省略了错误检查。

```
void *handle;
int i, *iptr;
int (*fptr)(int);

/* open the needed object */
handle = dlopen("/usr/mydir/mylib.sl", RTLD_LAZY);

/* find address of function and data objects */
fptr = (int (*)(int))dlsym(handle, "some_function");

iptr = (int *)dlsym(handle, "int_object");

/* invoke function, passing value of integer as a parameter */

i = (*fptr)(*iptr);
```

以下示例说明如何同时使用 **RTLD_NEXT** 和 **dlsym()** 将功能添加到某个现有接口。同样，省略了错误检查。

```
extern void record_malloc(void *, size_t);

void *
malloc(size_t sz)
```

```
{
    void *ptr;
    void *(*real_malloc)(size_t);

    real_malloc = (void * (*) (size_t))
            dlsym(RTLD_NEXT, "malloc");
    ptr = (*real_malloc)(sz);
    record_malloc(ptr, sz);
    return ptr;
}
```

另请参阅

dlclose(3C)、dlerrno(3C)、dlerrno(3C)、dlerror(3C)、dlopen(3C)。

文本和教程

《HP-UX Linker and Libraries Online User Guide》
（请参阅 **+help** 选项）

《HP-UX Linker and Libraries User's Guide》
（有关订购信息，请参阅 *manuals*(5) ）

名称

doupdate、refresh、 wnoutrefresh 和 wrefresh — 刷新窗口和行

概要

#include <curses.h>

int doupdate(void);

int refresh(void);

int wnoutrefresh(WINDOW *_win_);

int wrefresh(WINDOW *_win_);

说明

refresh() 和 **wrefresh()** 函数可刷新当前窗口或指定窗口。函数可在窗口的光标位置定位终端光标，但是，如果启用了 **leaveok()** 模式，则可将光标置于任意位置。

wnoutrefresh() 函数确定终端的哪些部分可能需要更新。 **doupdate()** 函数向终端发送命令，以执行任何所需的更改。

返回值

成功完成后，这些函数将返回 OK。否则，返回 ERR。

错误

没有定义任何错误。

实际应用信息

通常，刷新整个窗口比分别刷新多个子窗口效率更高。高效的调用顺序是这样的：在每个已更改的子窗口上调用 **wnoutrefresh()** ，接着调用更新终端的 **doupdate()** 。

必须调用 **refresh()** 或 **wrefresh()** 函数（或者先调用 **wnoutrefresh()** ，然后调用 **doupdate()** ）以将输出发送到终端，这是因为其他 Curses 函数仅处理数据结构。

另请参阅

clearok(3X)、 redrawwin(3X)、 <curses.h>。

历史变更记录

在 X/Open Curses 第 4 期中首次发布。

该条目合并了 X/Open Curses 第 3 期中包含的 **refresh()** 和 **wnoutrefresh()** 条目。为清楚起见，重新编写了说明，并且 **doupdate()** 和 **refresh()** 函数的参数列表已显式声明为 **void**。否则，功能等同于 X/Open Curses 第 3 期中定义的功能。

名称

drand48()、erand48()、lrand48()、nrand48()、mrand48()、jrand48()、srand48()、seed48()、lcong48() - 生成平均分布的伪随机数

概要

```
#include <stdlib.h>

double drand48(void);

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

long int lrand48(void);

long int nrand48(unsigned short int xsubi[3]);

long int mrand48(void);

long int jrand48(unsigned short int xsubi[3]);

void srand48(long int seedval);

unsigned short int *seed48(unsigned short int seed16v[3]);

void lcong48(unsigned short int param[7]);
```

过时的接口

```
int drand48_r(struct drand48_data *dp, double *randval);

int erand48_r(
  unsigned short int xsubi[3],
  struct drand48_data *dp,
  double *randval);

int lrand48_r(struct drand48_data *dp, long int *randval);

int nrand48_r(
  unsigned short int xsubi[3],
  struct drand48_data *dp,
  long int *randval);

int mrand48_r(struct drand48_data *dp, long int *randval);

int jrand48_r(
  unsigned short int xsubi[3],
  struct drand48_data *dp,
  long int *randval);

int srand48_r(long int seedval, struct drand48_data *dp);
```

int seed48_r(unsigned short int seed16v[3], struct drand48_data *dp);

int lcong48_r(unsigned short int param[7], struct drand48_data *dp);

说明

此函数系列可使用常见的线性同余算法和 48 位整数算术生成伪随机数。

在下面的介绍中，正式数学表示法 [*low*,*high*) 表示包括 *low* 但不包括 *high* 的区间。

drand48() 和 **erand48()** 返回在 [0.0,1.0) 区间内平均分布的非负双精度浮点值。

lrand48() 和 **nrand48()** 返回在 [0,2^31) 区间内平均分布的非负 long 整型数。

mrand48() 和 **jrand48()** 返回在 [–2^31,2^31) 区间内平均分布的带符号 long 整型数。

srand48() 、 **seed48()** 和 **lcong48()** 为初始化入口点，只有在调用其中一项之后，才可以调用 **drand48()** 、 **lrand48()** 或 **mrand48()**

（虽然这不是建议的做法，但如果在调用 **drand48()** 、 **lrand48()** 或 **mrand48()** 之前，没有事先调用初始化入口点，就应该自动提供常量缺省初始化程序值）。 **erand48()** 、 **nrand48()** 和 **jrand48()** 不需要首先调用初始化入口点。

所有例行程序的工作方式是，根据以下线性同余公式，生成 48 位整数值的序列 *X*[*i*]

$$X[n+1] = (a*X[n] + c) \text{ modulo } m \qquad n>=0$$

参数 *m* = 2^48 ； 由此执行 48 位整数算术。

除非已调用 **lcong48()** ，否则按以下计算方法得出缺省乘数值 *a* 和缺省加数值 *c*

a = 0x5DEECE66D（十六进制数）= 0273673163155（八进制数）
c = 0xB（十六进制数）= 013（八进制数）

计算 **drand48()** 、 **erand48()** 、 **lrand48()** 、 **nrand48()** 、 **mrand48()** 或 **jrand48()** 函数中任何一个返回的值时，首先生成序列中的下一个 48 位 *X*[*i*] 。然后根据要返回的数据项目的类型，从 *X*[*i*] 的高次（最左边）位复制相应数目的位，并将这些位转换为返回值。

函数 **drand48()** 、 **lrand48()** 和 **mrand48()** 可存储内部缓冲区中生成的最后一个 48 位 *X*[*i*] ；这就是为什么在调用这些函数之前需要对其初始化的原因。调用函数 **erand48()** 、 **nrand48()** 和 **jrand48()** 时，这些函数需要调用程序为指定为参数的数组中的连续 *X*[*i*] 值提供存储空间。这就是为什么不必要初始化这些例行程序的原因；调用程序只要将所需的 *X*[*i*] 的初始值置于数组内，并将其作为参数传递。通过使用不同的参数， **erand48()** 、 **nrand48()** 和 **jrand48()** 允许大程序的单个模块生成伪随机数的若干 *independent* 流；例如，每个流中的编号序列根据调用例行程序以生成其他流的编号的次数，执行 *not* 。

初始化程序函数 **srand48()** 可将 *X*[*i*] 的高次 32 位设置为其参数包含的 32 位。 *X*[*i*] 的低次 16 位设置为任意值 0x330E（十六进制数）。

初始化程序函数 **seed48()** 可将 *X*[*i*] 的值设置为参数数组中指定的 48 位值。此外， *X*[*i*] 的上一个值将复制到 48 位内部缓冲区，并且只由 **seed48()** 使用，指向此缓冲区的指针为 **seed48()** 返回的值。不需要该返回指针时可以将

其忽略，如果某个程序在将来某个时候需要从指定的点重新启动，使用这个返回指针是十分有帮助的；使用该指针可到达和存储最后一个 *X*[*i*] 值，程序重新启动时可使用此值通过 **seed48()** 执行重新初始化。

用户可通过初始化函数 **lcong48()** 指定初始 *X*[*i*] 、乘数值 *a* 和加数值 *c* 。参数数组元素 *param*[0-2] 指定 *X*[*i*] ，*param*[3-5] 指定乘数 *a* ， *param*[6] 指定 16 位加数 *c* 。调用 **lcong48()** 后，后续调用 **srand48()** 或 **seed48()** 将恢复上面指定的 *a* 和 *c* 的缺省乘数和缺省加数值。

过时的接口

drand48_r() 、 **erand48_r()** 、 **lrand48_r()** 、 **nrand48_r()** 、 **mrand48_r()** 、 **jrand48_r()** 、 **srand48_r()** 、 **seed48_r()** 、 **lcong48_r()** 生成平均分布的伪随机数。

警告

drand48_r() 、 **erand48_r()** 、 **lrand48_r()** 、 **nrand48_r()** 、 **mrand48_r()** 、 **jrand48_r()** 、 **srand48_r()** 、 **seed48_r()** 和 **lcong48_r()** 是为了与现有 DCE 应用程序兼容而受到支持的已过时接口。新的多线程应用程序应该使用 **drand48()** 、 **erand48()** 、 **lrand48()** 、 **nrand48()** 、 **mrand48()** 、 **jrand48()** 、 **srand48()** 、 **seed48()** 和 **lcong48()** 。

另请参阅

rand(3C)、 random(3M)、 thread_safety(5)、 random(7)。

符合的标准

drand48()：AES、SVID2、SVID3、XPG2、XPG3、XPG4

erand48()：AES、SVID2、SVID3、XPG2、XPG3、XPG4

jrand48()：AES、SVID2、SVID3、XPG2、XPG3、XPG4

lcong48()：AES、SVID2、SVID3、XPG2、XPG3、XPG4

lrand48()：AES、SVID2、SVID3、XPG2、XPG3、XPG4

mrand48()：AES、SVID2、SVID3、XPG2、XPG3、XPG4

nrand48()：AES、SVID2、SVID3、XPG2、XPG3、XPG4

seed48()：AES、SVID2、SVID3、XPG2、XPG3、XPG4

srand48()：AES、SVID2、SVID3、XPG2、XPG3、XPG4

名称

dupwin — 复制窗口

概要

#include <curses.h>

WINDOW *dupwin(WINDOW *_win_);

说明

dupwin() 函数创建窗口 *win* 的副本。

返回值

成功完成后， **dupwin()** 返回指向新窗口的指针。否则返回空指针。

错误

没有定义任何错误。

另请参阅

derwin(3X)、 doupdate(3X)、 <curses.h>。

历史变更记录

首次发布于 X/Open Curses，第 4 期。

名称

echo 和 noecho — 启用/禁用终端回显

概要

#include <curses.h>

int echo(void);

int noecho(void);

说明

echo() 函数为当前屏幕启用回显模式。 **noecho()** 函数为当前屏幕禁用回显模式。最初，启用的是 Curses 软件回显模式，禁用的是 tty 驱动程序的硬件回显模式。

返回值

成功完成后，这些函数将返回 OK。否则，将返回 ERR。

错误

没有定义任何错误。

另请参阅

getch(3X)、 curses_intro(3X)，输入处理小节、<curses.h>、《X/Open System Interface Definitions, Issue 4, Version 2》规范，第 9.2 节 Parameters That Can Be Set 。

历史变更记录

在第 2 期中首次发布。

X/Open Curses 第 4 期

为清楚起见，重新编写了该条目。 **echo()** 和 **noecho()** 函数的参数列表已显式声明为 **void**。

名称

echochar 和 wechochar — 将单字节字符和呈现方式回显到窗口并进行刷新

概要

#include <curses.h>

int echochar(const chtype *ch***);**

int wechochar(WINDOW ****win*, const chtype** *ch***);**

说明

echochar() 函数等同于先调用 **addch()** ，然后调用 **refresh()** 。

wechochar() 函数等同于先调用 **waddch()** ，然后调用 **wrefresh()** 。

返回值

成功完成后，这些函数返回 OK。否则，它们返回 ERR。

错误

没有定义任何错误。

实际应用信息

这些函数只保证对这样的字符集进行可靠操作：其中的每个字符都属于单字节，只能用带有 A_ 前缀的常量来表示其属性。

另请参阅

addch(3X)、 doupdate(3X)、 echo_wchar(3X)、 <curses.h>。

历史变更记录

在 X/Open Curses 第 4 期中首次发布。

名称

echo_wchar 和 wecho_wchar — 写入组合字符并立即刷新窗口

概要

#include <curses.h>

int echo_wchar(const cchar_t *_wch_);

int wecho_wchar(WINDOW *_win_, const cchar_t *_wch_);

说明

echo_wchar() 函数等同于先调用 **add_wch()** ，然后调用 **refresh()** 。

wecho_wchar() 函数等同于先调用 **wadd_wch()** ，然后调用 **wrefresh()** 。

返回值

成功完成后，这些函数将返回 OK。否则，返回 ERR。

错误

没有定义任何错误。

另请参阅

addch(3X)、 add_wch(3X)、 doupdate(3X)、 <curses.h>。

历史变更记录

在 X/Open Curses 第 4 期中首次发布。

名称

ecvt( )、fcvt( )、gcvt( ) - 将浮点数转换为字符串

概要

```
#include <stdlib.h>

char *ecvt(double value, int ndigit, int *__restrict decpt, int *__restrict sign);

char *fcvt(double value, int ndigit, int *__restrict decpt, int *__restrict sign);

char *gcvt(double value, int ndigit, char *buf);
```

过时的接口

```
int ecvt_r(
  double value,
  int ndigit,
  int *decpt,
  int *sign,
  char *buffer,
  int buflen);

int fcvt_r(
  double value,
  int ndigit,
  int *decpt,
  int *sign,
  char *buffer,
  int buflen);
```

说明

ecvt() 将 *value* 转换为 *ndigit* 位数的以 null 结尾的字符串，同时返回指向该字符串的指针。除非值为零，否则高位数字为非零值。将舍入低位数字。通过 *decpt* 间接存储与字符串开头相对的小数分隔符的位置（负值表示位于返回数字的左边）。返回的字符串中不包括小数分隔符。如果结果的符号为负值，则 *sign* 指向的单词为非零值，否则为零。

如果转换的值超出范围，则可能返回三个非数字字符型字符串中的一个。如果该值大于指数能够包含的值，当该值分别为负值和正值时，将分别返回 **--** 和 **++** 。如果数字非法，例如零除数，则返回第三个字符串。结果值为非数值 (NAN) ，同时返回一个 **?** 字符。

fcvt() 与 **ecvt()** 类似，不同之处在于，已根据 *ndigit* 指定位数的 printf **%f** （FORTRAN F 格式）输出对正确的数字进行舍入。

gcvt() 在 *buf* 指向的数组中，将 *value* 转换为以 null 结尾的字符串，同时返回 *buf* 。如果可能，它可生成 FORTRAN F 格式的 *ndigit* 有效数字，否则生成 E 格式的有效数字。将在返回的字符串中加入一个负号（如果必要）和一个小数分隔符。禁止尾随零。小数分隔符由当前加载的 NLS 环境（请参阅

setlocale(3C) ）确定。如果未成功调用 **setlocale()** ，则使用缺省的 NLS 环境“C”（请参阅 *lang*(5)）。缺省环境将句点 (**.**) 指定为小数分隔符。

过时的接口

ecvt_r() 和 **fcvt_r()** 可将浮点数转换为字符串。

外部语言环境影响

语言环境

LC_NUMERIC 类别可确定当前的 NLS 环境中的小数分隔符的值。

警告

同一线程对这些接口执行后续调用后，将覆盖 **ecvt()** 和 **fcvt()** 返回的值所指向的数组的内容。

ecvt_r() 和 **fcvt_r()** 是已过时接口，只是为了与现有 DCE 应用程序兼容而受到支持。新的多线程应用程序应该使用 **ecvt()** 和 **fcvt()** 。

作者

ecvt() 和 **fcvt()** 由 AT&T 开发。 **gcvt()** 由 AT&T 和 HP 开发。

另请参阅

setlocale(3C)、 printf(3S)、lang(5)、 thread_safety(5)、glossary(9)。

符合的标准

ecvt()： XPG2

fcvt()： XPG2

gcvt()： XPG2

名称

elf - 对象文件访问库

概要

cc [*flag*...] *file*... **-lelf** [*library*] ...

#include <libelf.h>

说明

程序可以使用 ELF 访问库中的函数来操作 ELF（可执行文件和链接格式）对象文件、归档文件和归档成员。头文件可提供所有库服务的类型和函数声明。

程序可以使用 “**ELF** 描述符”与多个高级例行程序通信。也就是说，当程序开始处理文件时， **elf_begin** 会创建一个 ELF 描述符，而程序将通过此描述符操作文件中的结构和信息。可以使用这些 ELF 描述符读取和写入文件。建立文件的 ELF 描述符后，程序可能获取 *section descriptors* 以操作文件的各个段（请参阅 *elf_getscn*(3E)）。这些段保存了对象文件的大量实际信息，例如文本、数据、符号表等等。如同某个章节属于某个文件一样，段描述符“属于”某个特定的 ELF 描述符。最后，可以通过段描述符获取 *data descriptors* ，使程序能够操作与某个段关联的信息。数据描述符“属于”段描述符。

描述符为文件及其片段提供专用句柄。换句话说，数据描述符与一个段描述符关联，此段描述符又与一个 ELF 描述符关联，而此 ELF 描述符又与一个文件关联。虽然描述符是专用的，但是它们可以授予可能共享的数据以访问权限。想象一下可以结合输入文件，且使用输入数据创建或更新另一个文件的程序。这样的程序可能会获取输入和输出段的数据描述符。然后更新输出描述符，以重新使用输入描述符的数据。也就是说，虽然描述符是各不相同的，但它们能够共享关联的数据字节。这种共享方式可以避免由于使用重复缓冲区而发生的空间开销，以及进行不必要的数据复制而产生的性能开销。

文件类

ELF 提供了一个框架，可以在其中定义一系列支持多个处理器和体系结构的对象文件。对象文件之间存在的重要差异在于 *class* ，或者文件的容量。在 32 位类支持的体系结构中， 32 位对象可以表示地址、文件大小等等，如下文所述。

名称	用途
Elf32_Addr	无符号地址
Elf32_Half	无符号的中整数
Elf32_Off	无符号文件偏移量
Elf32_Sword	带符号的大整数
Elf32_Word	无符号的大整数
无符号字符	无符号小整数

可以视需要定义其他类以支持更大（或更小）的计算机。有些库服务只能处理特定类的数据对象，而另有一些却与类无关。为了更明确地体现这个差异，可以通过库函数的名称反映各自的状态，如下文所述。

数据表示形式

从概念上讲，两组并行的对象能够支持交叉的编译环境。其中一组对应于文件内容，而另一组对应于操作文件的程序的本地内存映像。头文件提供的类型定义将在本地计算机上工作，而本地计算机与目标计算机的数据编码（大小、字节顺序等等）可能不同。虽然本地内存对象与文件对象的大小至少应该相同（为避免信息丢失），但只要对主机更有利，就可以增加本地内存对象的大小。

提供转换设施，用于相互转换文件和内存表示形式。有些库例行程序能够自动转换数据，而另有一些将转换工作交给程序来负责。无论是哪种方式，创建对象文件的程序必须将文件类型的对象写入那些对象文件；读取对象文件的程序必须使用相同的视图。有关详细信息，请参阅 *elf_xlate*(3E) 和 *elf_fsize*(3E) 。

程序可以显式转换数据，这样可以完全控制对象文件的布局和语义。如果程序不希望拥有和执行完全控制权限，库还提供了一个高级接口，可用来隐藏对象文件的许多详细信息。 **elf_begin** 及其相关函数允许程序处理本地内存类型，当读取或写入某个对象文件时，它们可以自动地将内存对象及其等效文件相互进行转换。

ELF 版本

对象文件版本使 ELF 能够适应新的要求。如何协调三个独立的版本，对程序来说十分重要。第一，由于应用程序是根据特定的头文件编译的，因而它了解特定的版本。第二，访问库同样也是使用头文件编译的，而此头文件控制了其识别的版本。第三， ELF 对象文件拥有可识别其版本的值，此值由文件生成器已知的 ELF 版本确定。最佳状态是这三个版本相同，但事实它们可能存在差异。

如果程序版本高于访问库版本，则程序使用的信息对库来说可能是未知的。这样，转换例行程序可能无法正常工作，从而导致不确定的行为。在这种情况下，就应该安装新库。

库的版本可能高于程序和文件的版本。库能够识别旧版本，因而可以避免在这种情况下出现的兼容性问题。

最后，文件的版本可能高于程序或库这两者识别的版本。程序能否正确地处理文件，取决于文件是否带有额外的信息，以及能否安全地忽略该信息。同样，安全的可选做法是安装一个能识别文件版本的新库。

为适应这些差异，程序必须使用 **elf_version** 向库传递其版本，从而为进程建立 “工作版本”。通过这种方式，库与程序之间能够相互接收和发送以适当形式表示的数据。当库读取对象文件时，需要使用每个文件的版本来解释数据。当库写入文件，或者将内存类型转换为等效文件时，需要使用程序的文件数据的工作版本。

系统服务

如上所述， **elf_begin** 及其相关例行程序为 ELF 文件提供了一个高级接口，代表应用程序执行输入和输出操作。这些例行程序假定程序能够在内存中保存全部文件，而不必显式使用临时文件。库例行程序读取文件时，会将数据送入内存，同时对内存副本执行后续操作。以这种模式读取或写入大对象文件的程序，必须在具有大进程虚拟地址空间的计算机上运行。如果基础操作系统限制了打开文件的数目，程序可以使用 **elf_cntl** 从文件中检索所有必要的数据，以便关闭文件描述符，然后重新使用。

虽然对许多程序来说， **elf_begin** 接口方便且有效，但对某些程序来说，这些接口是不适用的。在这些情况下，应用程序可以直接调用 **elf_xlate** 数据转换例行程序。这些例行程序不执行任何输入或输出操作，而将此工作交给应用程序负责。应用程序需要控制其输入和输出模式，因而承担了大部分的作业量。

库名称

与库关联的名称存在多种格式。

elf_name 这些与类无关的名称将为程序执行某种 *name* 服务。

elf32_name 带有嵌入类（此处为 **32** ）的服务名称，表示它们只适用于指定类的文件。

elf64_name 带有嵌入类（此处为 **64** ）的服务名称，表示它们只适用于指定类的文件。

Elf_Type 数据类型也可以与类无关，而使用 *Type* 加以区分。

Elf32_Type 与类相关的数据类型包括嵌入类名（此处为 **32** ）。

Elf64_Type 与类相关的数据类型包括嵌入类名（此处为 **64** ）。

ELF_C_CMD 某些函数使用可以控制其操作的命令。这些值是 **Elf_Cmd** 枚举的成员；其范围介于 0 至 **ELF_C_NUM-1** 之间。

ELF_F_FLAG 某些函数使用可以控制库状态和（或）操作的标记。标记是可以组合的位。

ELF32_FSZ_TYPE 这些常量可指定32 位类文件的基本 ELF 类型的以字节为单位的文件大小。有关详细信息，请参阅 **elf_fsize** 。

ELF64_FSZ_TYPE 这些常量可指定64 位类文件的基本 ELF 类型的以字节为单位的文件大小。有关详细信息，请参阅 **elf_fsize** 。

ELF_K_KIND 函数 **elf_kind** 可以识别与 ELF 描述符关联的文件的 *KIND* 。这些值是 **Elf_Kind** 枚举的成员；其范围介于 0 至 **ELF_K_NUM-1** 之间。

ELF_T_TYPE 当 **elf_xlate** 等服务函数处理多个类型时，这种格式的名称将指定所需的 *TYPE* 。因此，例如， **ELF_T_EHDR** 将直接与 **Elf32_Ehdr** 关联。这些值是 **Elf_Type** 枚举的成员；其范围介于 0 至 **ELF_T_NUM-1** 之间。

注释

ELF 头文件中的信息将划分为普通部分和处理器特定的部分。程序可以通过加入以下适当的头文件，使处理器的信息可用： **sys/elf_NAME.h** ，其中 *NAME* 与 ELF 文件标头中使用的处理器名称匹配。

符号	处理器
parisc	**PA RISC**

必要时可以在表中添加其他处理器。例如，程序可以使用以下代码来“查看”WE 32100 的处理器特定信息。

```
#include <libelf.h>

#include <sys/elf_M32.h>
```

如果没有 **sys/elf_M32.h** 定义，则只能看到普通的 ELF 信息。

另请参阅

a.out(4) 、 ar(4) 、 elf_begin(3E) 、 elf_cntl(3E) 、 elf_end(3E) 、 elf_error(3E) 、 elf_fill(3E) 、 elf_flag(3E) 、 elf_fsize(3E) 、 elf_getarhdr(3E) 、 elf_getarsym(3E) 、 elf_getbase(3E) 、 elf_getdata(3E) 、 elf_getehdr(3E) 、 elf_getident(3E)、 elf_getphdr(3E)、 elf_getscn(3E)、elf_getshdr(3E)、 elf_hash(3E)、elf_kind(3E)、elf_next(3E)、 elf_rand(3E)、elf_rawfile(3E)、elf_strptr(3E)、 elf_update(3E)、elf_version(3E)、elf_xlate(3E)。

名称

elf_begin() - 为 ELF 文件生成文件描述符

概要

命令： **cc** [*flag*]... *file*... **-lelf** [*library*]...

#include <libelf.h>

Elf *elf_begin(
int *fildes*,
Elf_Cmd *cmd*,
**Elf ***ref*
);

说明

elf_begin() 、 **elf_next()** 、 **elf_rand()** 和 **elf_end()** 一起使用，可以对 ELF 对象文件进行单独的处理，或者将其当作归档的成员进行处理。从 **elf_begin()** 获取 ELF 描述符后，程序可能读取、更新现有的文件，或创建新文件。*fildes* 是一个打开文件的描述符， **elf_begin()** 使用其执行读取或写入操作。初始文件偏移量（请参阅 *lseek*(2) ）不受限制，所得文件偏移量不确定。 *cmd* 可能包括下列值。

ELF_C_NULL 程序将 *cmd* 设置为此值后， **elf_begin()** 将返回一个空指针，且不打开新描述符。此命令将忽略 *ref* 。有关详细信息，请参阅 *elf_next*(3E) 和下面的示例。

ELF_C_READ 当程序需要检查现有文件的内容时，应该将 *cmd* 设置为此值。此命令将检查归档成员或全部文件，这取决于 *ref* 的值。可能会出现三种情况。

首先，如果 *ref* 是空指针， **elf_begin()** 将分配一个新的 ELF 描述符，并准备处理整个文件。如果读取的文件是归档文件， **elf_begin()** 同样会准备所得描述符，以便在下一次调用 **elf_begin()** 时检查初始归档成员，其操作方式如同程序使用了 **elf_next()** 或 **elf_rand()** 来“移动”到初始成员。

其次，如果 *ref* 是与归档文件关联的非空描述符， **elf_begin()** 将允许程序获取与单个成员关联的单独 ELF 描述符。程序应该使用 **elf_next()** 或 **elf_rand()** 来正确定位 *ref* （ **elf_begin()** 准备的初始成员除外；请参阅下面的示例）。在这种情况下， *fildes* 应该与父级归档使用的文件描述符相同。

最后，如果 *ref* 是非空 ELF 描述符，而此 ELF 描述符不是归档， **elf_begin()** 将增加描述符的激活次数，并返回 *ref* ，且不分配新的描述符和更改描述符的读/写权限。为了终止 *ref* 的描述符，程序必须在每次激活时调用一次 **elf_end()** 。有关详细信息，请参阅 *elf_next*(3E) 和下面的示例。

ELF_C_RDWR 此命令将复制 **ELF_C_READ** 的操作，此外还允许程序更新文件映像（请参阅 *elf_update*(3E) ）。也就是说，使用 **ELF_C_READ** 将提供文件的只读视图，而使用 **ELF_C_RDWR** 允许程序读取和写入文件。 **ELF_C_RDWR** 对归档成员无效。必须使用 **ELF_C_RDWR** 命令创建非空的 *ref* 。

ELF_C_WRITE 如果程序为了创建新文件而忽略以前的文件内容，则应该将 *cmd* 设置为此值。此命令将忽略 *ref* 。

只要 **elf_begin()** 能够为其内部结构分配内存，以及从文件读取任何必要的信息，它就可以在所有文件（包括零字节文件）上“工作”。这样，读取对象文件的程序可以调用 **elf_kind()** 或 **elf_getehdr()** 以确定文件类型（只有对象文件才具有 ELF 标头）。如果文件是没有其他待处理成员的归档文件，或者发生了错误， **elf_begin()** 将返回一个空指针。否则，返回值为非空 ELF 描述符。首次调用 **elf_begin()** 之前，程序必须调用 **elf_version()** 来调整版本。

系统服务

处理文件时，将由库决定何时读取或写入文件，这取决于程序的请求。通常，库假定文件描述符对 ELF 描述符的整个生命周期都可用。但是，如果程序必须同时处理多个文件，而基础操作系统限制了打开文件的数目，程序可以通过 **elf_cntl()** 来重新使用文件描述符。使用适当的参数调用 **elf_cntl()** 后，程序可以在不干扰库的条件下关闭文件描述符。

在 **elf_end()** 终止描述符的最后一次激活之前，与 ELF 描述符关联的所有数据都处于已分配状态。终止描述符后，将释放存储；尝试引用这些数据将导致不确定的行为。因此，处理多个输入（或输出）文件的程序在完成处理之前，必须使 ELF 描述符处于活动状态。

注释

当程序调用 COFF 文件上的 **elf_begin()** 时，库将 COFF 结构转换为其 ELF 等效结构，以便程序将 COFF 文件当作 ELF 文件来读取（但不可写入）。此转换操作只适用于内存映像，而不适用于文件自身。

举例

下面展示了一个读取文件的示例。如果文件是简单对象文件，则程序只执行循环一次，第二次迭代将收到空描述符。在这种情况下， **elf** 和 **arf** 都将得到相同的值，激活计数将为 2，并且程序需要两次调用 **elf_end()** 来终止描述符。如果文件是归档文件，循环将依次处理每个归档成员，并忽略那些不是对象文件的文件。

```
if (elf_version(EV_CURRENT) == EV_NONE)
    {
    /* library out of date */
    /* recover from error */
    }
cmd = ELF_C_READ;
arf = elf_begin(fildes, cmd, (Elf *)0);
while ((elf = elf_begin(fildes, cmd, arf)) != 0)
    {
    if ((ehdr = elf32_getehdr(elf)) != 0)
        {
        /* process the file .. . */
        }
    cmd = elf_next(elf);
    elf_end(elf);
```

```
        }
    elf_end(arf);
```

另举一例说明随机归档的处理。将文件识别为归档文件后，程序将反复处理相关的归档成员。为了更清楚地说明问题，本示例省略了错误检查，并忽略了简单对象文件。此外，此片段保留所有归档成员的 ELF 描述符，这是因为它不调用 **elf_end()** 以终止这些 ELF 描述符。

```
elf_version(EV_CURRENT);
arf = elf_begin(fildes, ELF_C_READ, (Elf *)0);
if (elf_kind(arf) != ELF_K_AR)
    {
    /* not an archive */
    }
/* initial processing */
/* set offset = .. . for desired member header */
while (elf_rand(arf, offset) == offset)
    {
    if ((elf = elf_begin(fildes, ELF_C_READ, arf)) == 0)
        break;
    if ((ehdr = elf32_getehdr(elf)) != 0)
        {
        /* process archive member .. . */
        }
    /* set offset = .. . for desired member header */
    }
```

以下简短示例显示如何创建新的 ELF 文件。为了显示整个流程而简化了此示例。

```
elf_version(EV_CURRENT);
fildes = open("path/name", O_RDWR|O_TRUNC|O_CREAT, 0666);
if ((elf = elf_begin(fildes, ELF_C_WRITE, (Elf *)0)) == 0)
    return;
ehdr = elf32_newehdr(elf);
phdr = elf32_newphdr(elf, count);
scn = elf_newscn(elf);
shdr = elf32_getshdr(scn);
data = elf_newdata(scn);
elf_update(elf, ELF_C_WRITE);
elf_end(elf);
```

最后，以下简短示例显示如何更新现有的 ELF 文件。同样，为了显示整个流程而简化了此示例。

```
elf_version(EV_CURRENT);
```

```
fildes = open("path/name", O_RDWR);
elf = elf_begin(fildes, ELF_C_RDWR, (Elf *)0);

/* add new or delete old information . . . */

close(creat("path/name", 0666));
elf_update(elf, ELF_C_WRITE);
elf_end(elf);
```

在以上示例中，调用 **creat** 后将截断文件，以确保所得文件具有“正确的”大小。如果不截断，则即使删除了信息，已更新文件的大小可能与原来一样。如果可能，库将使用 **ftruncate** （请参阅 *truncate*(2) ）截断文件。但是，某些系统不支持 **ftruncate** ，并且调用 **creat** 时还会防止截断文件。请注意，两个创建文件的示例使用读取和写入权限打开文件。如果系统支持 **mmap** ，则库可以使用其增强性能。此外， **mmap** 需要可读的文件描述符。虽然库可以使用只写的文件描述符，但是应用程序却无法获取 **mmap** 的性能优势。

另请参阅

creat(2)、 lseek(2)、 mmap(2)、 open(2)、 truncate(2)、 elf(3E)、 elf_cntl(3E)、 elf_end(3E)、 elf_getarhdr(3E)、 elf_getbase(3E) 、 elf_getdata(3E) 、 elf_getehdr(3E) 、 elf_getphdr(3E) 、 elf_getscn(3E) 、 elf_kind(3E) 、 elf_next(3E)、 elf_rand(3E)、 elf_rawfile(3E)、 elf_update(3E)、 elf_version(3E)、 ar(4)。

名称

elf_cntl - 控制一个文件描述符

概要

cc [*flag*...] *file*... **-lelf** [*library*] ...

#include <libelf.h>

int elf_cntl(Elf *elf, Elf_Cmd cmd);

说明

elf_cntl 指示库根据 ELF 描述符 *elf* 更改其行为。如 *elf_begin*(3E) 所述，ELF 描述符可以有多种激活方式，多个 ELF 描述符可以共享一个文件描述符。一般情况下， **elf_cntl** 命令适用于所有的 *elf* 激活方式。而且，如果 ELF 描述符与一个归档文件关联，该归档中的成员描述符也将受到影响，如下所述。除非另行说明，否则对归档成员的操作不会影响包含归档的描述符。

cmd 参数指示要采取的操作，并可能具有下列值。

ELF_C_FDDONE 该值指示库不要使用与 *elf* 关联的描述符。当程序请求了要使用的所有信息，并且希望免除读取文件其余部分的开销时，应当使用该命令。所有已完成操作的内存保持有效，但如果数据已不在内存中，则后面的文件操作（例如对某一部分执行最初的 **elf_getdata** ）将会失败。

ELF_C_FDREAD 该命令类似于 **ELF_C_FDDONE** ，唯一的不同之处在于它强制库读取文件的其余部分。当程序必须关闭文件描述符但还没有读取它所需要的全部内容时，应当使用该命令。在 **elf_cntl** 完成 **ELF_C_FDREAD** 命令后，将来的操作（例如 **elf_getdata** ）将使用文件的内存版本而不必使用文件描述符。

如果 **elf_cntl** 成功，则返回零。否则， *elf* 为空或发生错误，该函数返回 -1。

注释

在利用 **ELF_C_FDDONE** 或 **ELF_C_FDREAD** 禁用文件描述符后，如果程序希望使用“原始”操作（请参阅 *elf_getdata*(3E) 和 *elf_rawfile*(3E) 中描述的 **elf_rawdata** ），则它必须预先明确执行原始操作。否则，原始文件操作将会失败。调用 **elf_rawfile** 可使整个映像可用，从而支持随后的 **elf_rawdata** 调用。

另请参阅

elf(3E)、 elf_begin(3E)、 elf_getdata(3E)、 elf_rawfile(3E)。

名称

elf_end - 结束使用对象文件

概要

cc [*flag*...] *file*... **-lelf** [*library*] ...

#include <libelf.h>

int elf_end(Elf *elf);

说明

程序使用 **elf_end** 来终止 ELF 描述符 *elf* 并取消分配与描述符关联的数据。在程序终止描述符之前，仍将继续分配数据。 *elf* 应为 **elf_begin** 以前返回的值；可以将空指针用作参数，以简化错误的处理过程。如果程序要将与 ELF 描述符关联的数据写入文件中，则必须在调用 **elf_end** 前使用 **elf_update** 。

如 *elf_begin*(3E) 所述，一个描述符可以进行多次激活。调用 **elf_end** 将删除一个激活点并返回剩余的激活点数。在激活点数达到零之前，库不会终止描述符。因此，返回值为零时表示 ELF 描述符已不再有效。

另请参阅

elf(3E)、 elf_begin(3E)、 elf_update(3E)。

名称

elf_errmsg 和 elf_errno - 错误处理

概要

cc [*flag*...] *file*... **-lelf** [*library*] ...

#include <libelf.h>
const char *elf_errmsg(int err);
int elf_errno(void);

说明

如果 ELF 库函数失败，则程序可能会调用 **elf_errno** 来检索库的内部错误编号。其副作用是，该函数将内部错误编号重置为零，它表示没有错误。

elf_errmsg 使用错误编号 *err* ，并返回以空字符结尾的错误消息（没有结尾换行符）来描述问题。值为零的 *err* 可以检索消息中的最新错误。如果没有出现错误，则返回值为空指针（而不是指向空字符串的指针）。使用值为 -1 的 *err* 也可以检索最新错误，但是会确保返回非空值，即使没有出现错误也是如此。如果没有提供给定编号的消息，则 **elf_errmsg** 返回指向相应消息的指针。该函数不会产生清除内部错误编号的副作用。

举例

以下代码段清除内部错误编号，并在稍后检查其错误。除非第一次调用 **elf_errno** 后出现错误，否则下一次调用将返回零。

```
(void)elf_errno();
while (more_to_do)
{
    /* processing ... */
    if ((err = elf_errno()) != 0)
    {
        msg = elf_errmsg(err);
        /* print msg */
    }
}
```

另请参阅

elf(3E)、 elf_version(3E)。

名称

elf_fill - 设置填充字节

概要

cc [*flag*...] *file*... **-lelf** [*library*] ...

#include <libelf.h>

void elf_fill(int fill);

说明

ELF 文件的对齐限制有时需要存在“空隙”。例如，如果某一部分的数据需要以八字节边界开始，但是前面的部分太“短”，则库必须填充介于其间的字节。这些字节被设置为 *fill* 字符。除非应用程序提供值，否则库使用零字节。有关这些空隙的详细信息，请参阅 *elf_getdata*(3E) 。

注释

应用程序可通过设置 **ELF_F_LAYOUT** 位，来取得对象文件组织的控制权（请参阅 *elf_flag*(3E) ）。该操作完成后，库 不填充空隙。

另请参阅

elf(3E)、 elf_getdata(3E)、 elf_flag(3E)、 elf_update(3E)。

名称

elf_flagdata、 elf_flagehdr、 elf_flagelf、 elf_flagphdr、 elf_flagscn、 elf_flagshdr - 操作标记

概要

cc [*flag*...] *file*... **-lelf** [*library*] ...

#include <libelf.h>

unsigned elf_flagdata(Elf_Data *data, Elf_Cmd cmd, unsigned flags);
unsigned elf_flagehdr(Elf *elf, Elf_Cmd cmd, unsigned flags);
unsigned elf_flagelf(Elf *elf, Elf_Cmd cmd, unsigned flags);
unsigned elf_flagphdr(Elf *elf, Elf_Cmd cmd, unsigned flags);
unsigned elf_flagscn(Elf_Scn *scn, Elf_Cmd cmd, unsigned flags);
unsigned elf_flagshdr(Elf_Scn *scn, Elf_Cmd cmd, unsigned flags);

说明

这些函数可操作与某个 ELF 文件的各个结构关联的标记。如果给定了一个 ELF 描述符 *elf* 、一个数据描述符 *data* 或一个段描述符 *scn* ，这些函数就可以设置或清除关联的状态位，同时返回更新的位。为简化错误处理，不允许空描述符；对于此简并案例，所有函数都返回零。

cmd 可能包括以下值:

ELF_C_CLR 函数可清除 *flags* 中断言的位。只清除 *flags* 中的非零位；零位不更改描述符的状态。

ELF_C_SET 函数可设置 *flags* 中断言的位。只设置 *flags* 中的非零位；零位不更改描述符的状态。

下面显示了定义的标记位的描述。

ELF_F_DIRTY 当程序要写入某个 ELF 文件时，该标记可断言需要写入该文件的关联信息。因此，例如，某个程序需要更新现有文件的 ELF 标头，那么可以在 *flags* 中设置了该位，并且 *cmd* 等于 **ELF_C_SET** 的条件下，调用 **elf_flagehdr** 。以后调用 **elf_update** 会将带有标记的标头写入文件。

ELF_F_LAYOUT 通常，库可确定如何安排一个输出文件。也就是说，它可以自行确定在哪里放置段，如何在文件中对齐这些段，等等。如果针对某个 ELF 描述符设置了该位，则程序需要负责确定所有的文件定位。该位只对 **elf_flagelf** 有意义，并且可适用于与描述符关联的整个文件。

如果针对某个项目设置了标记位，则此标记位同样会影响所有子项目。例如，程序使用 **elf_flagelf** 设置了 **ELF_F_DIRTY** 位，则整个逻辑文件是“脏的”。

举例

以下代码段说明如何为要写入输出文件的 ELF 标头作标记。

```
ehdr = elf32_getehdr(elf);
/* dirty ehdr . . . */
elf_flagehdr(elf, ELF_C_SET, ELF_F_DIRTY);
```

另请参阅

elf(3E)、 elf_end(3E)、 elf_getdata(3E)、 elf_getehdr(3E)、 elf_update(3E)。

名称

elf32_fsize 和 elf64_fsize - 分别为 elf32 文件和 elf64 文件返回对象文件类型的大小。

概要

cc [*flag*...] *file*... **-lelf** [*library*] ...

#include <libelf.h>
size_t elf32_fsize(Elf_Type type, size_t count, unsigned ver);
size_t elf64_fsize(Elf_Type type, size_t count, unsigned ver);

说明

elf32_fsize 为具有给定 *type* 的 *count* 数据对象提供以 32 位文件表示时的大小（以字节为单位）。库使用版本 *ver* 计算大小（请参阅 *elf*(3E) 和 *elf_version*(3E) ）。

常量值可用于表示基本类型的大小。

Elf_Type	文件大小	内存大小
ELF_T_ADDR	**ELF32_FSZ_ADDR**	**sizeof(Elf32_Addr)**
ELF_T_BYTE	**1**	**sizeof**（无符号的 **char**）
ELF_T_HALF	**ELF32_FSZ_HALF**	**sizeof(Elf32_Half)**
ELT_T_OFF	**ELF32_FSZ_OFF**	**sizeof(Elf32_Off)**
ELF_T_SWORD	**ELF32_FSZ_SWORD**	**sizeof(Elf32_Sword)**
ELF_T_WORD	**ELF32_FSZ_WORD**	**sizeof(Elf32_Word)**

如果不知道 *type* 或者 *ver* 的值，则 **elf32_fsize** 将返回零。有关 *type* 值列表的详细信息，请参阅 *elf_xlate*(3E) 。

elf64_fsize 为具有给定 *type* 的 *count* 数据对象提供以 64 位文件表示时的大小（以字节为单位）。库使用版本 *ver* 计算大小（请参阅 *elf*(3E) 和 *elf_version*(3E) ）。

常量值可用于表示基本类型的大小。

Elf_Type	文件大小	内存大小
ELF_T_ADDR	**ELF64_FSZ_ADDR**	**sizeof(Elf64_Addr)**
ELF_T_BYTE	**1**	**sizeof**（无符号的 **char**）
ELF_T_HALF	**ELF64_FSZ_HALF**	**sizeof(Elf64_Half)**
ELT_T_OFF	**ELF64_FSZ_OFF**	**sizeof(Elf64_Off)**
ELF_T_SWORD	**ELF64_FSZ_SWORD**	**sizeof(Elf64_Sword)**
ELF_T_WORD	**ELF64_FSZ_WORD**	**sizeof(Elf64_Word)**

如果不知道 *type* 或者 *ver* 的值，则 **elf64_fsize** 将返回零。有关 *type* 值列表的详细信息，请参阅 *elf_xlate*(3E) 。

另请参阅

elf(3E)、 elf_version(3E)、 elf_xlate(3E)。

名称

elf_getarhdr - 检索归档成员标头

概要

cc [*flag*...] *file*... **-lelf** [*library*] ...

#include <libelf.h>

Elf_Arhdr *elf_getarhdr(Elf *elf);

说明

如果有用于 ELF 描述符 *elf* 的归档成员标头，则 **elf_getarhdr** 返回指向该归档成员标头的指针。在其他情况下，即不存在归档成员标头、发生错误或者 *elf* 为空，那么 **elf_getarhdr** 返回一个空值。标头包括下列成员。

```
char          *ar_name;
time_t         ar_date;
long           ar_uid;
long           ar_gid;
unsigned long  ar_mode;
off_t          ar_size;
char          *ar_rawname;
```

归档成员名称（可通过 **ar_name** 提供）是一个以空字符结尾的字符串，并且删除了 **ar** 格式控制字符。 **ar_rawname** 成员拥有一个以空字符结尾的、表示文件中原名称字节的字符串，包括如归档格式中所指定的终止斜线与尾随空格。

除了“常规”归档成员之外，归档格式中还定义了一些特殊成员。所有特殊成员名称以一个斜线 (**/**) 开始，将其与常规成员区分开（常规成员的名称中不能包括斜线）。这些特殊成员拥有如下定义的名称 (**ar_name**) 。

/ 这是一个归档符号表。如果存在，它将是第一个归档成员。程序可以通过 **elf_getarsym** 访问归档符号表。符号表中的信息对于随机归档处理很有用（请参阅 *elf_rand*(3E) ）。

// 如果存在，这一成员拥有一用于长归档成员名称的字符串表。一个归档成员的标头包含用于名称的 16 字节区域，在一些文件系统中可能会超出此区域。该库将 **ar_name** 设置为一个适当的值，自动从该字符串表中检索长成员名称。

在某些错误情况下，一个成员的名称可能不可用。尽管这会导致该库将 **ar_name** 设置为一个空指针，但仍将照常设置 **ar_rawname** 成员。

另请参阅

ar(4)、 elf(3E)、 elf_begin(3E)、 elf_getarsym(3E)、 elf_rand(3E)。

名称

elf_getarsym - 检索归档符号表

概要

cc [*flag*...] *file*... **-lelf** [*library*] ...

#include <libelf.h>

Elf_Arsym *elf_getarsym(Elf *elf, size_t *ptr);

说明

如果有用于 ELF 描述符 **elf** 的归档符号表，则 **elf_getarsym** 返回指向该归档符号表的指针。在其他情况下，即归档没有符号表、发生错误或者 **elf** 为空，那么 **elf_getarsym** 返回空值。符号表是包含下列成员的结构数组。

```
char        *as_name;
size_t      as_off;
unsigned long  as_hash;
```

这些成员可以具有以下语义。

as_name 此处保存指向以空字符结尾的符号名称的指针。

as_off 归档开头处与成员标头的字节偏移量。位于给定偏移位置处的归档成员将定义相关符号。可以将 **as_off** 中的值作为参数传递给 **elf_rand** ，以便访问所需的归档成员。

as_hash 由 **elf_hash** 计算的名称的哈希值。

如果 *ptr* 为非空，则库在 *ptr* 指向的位置存储表条目的个数。当返回值为空时，将该值设置为零。表的最后一个条目包含在计数中，它拥有空 **as_name** 、 **as_off** 的零值，以及 **as_hash** 的 **~0UL** 。

另请参阅

elf(3E)、 elf_getarhdr(3E)、 elf_hash(3E)、 elf_rand(3E)、 ar(4)。

名称

elf_getbase - 获取对象文件的基础偏移量

概要

cc [*flag*...] *file*... **-lelf** [*library*] ...

#include <libelf.h>

off_t elf_getbase(Elf *elf);

说明

如果文件地址偏移量是可知或可获取的，则 **elf_getbase** 返回文件第一个字节的文件地址偏移量或与 *elf* 相关的归档成员；否则，返回 -1。可以通过空 *elf* 简化错误处理；此种情况下将返回值 -1。归档成员的基础偏移量是成员信息的开头，而不是归档成员标头的开头。

另请参阅

ar(4)、 elf(3E)、 elf_begin(3E)。

名称

elf_getdata、 elf_newdata、 elf_rawdata - 获取段数据

概要

cc [*flag*...] *file*... **-lelf** [*library*] ...

#include <libelf.h>
Elf_Data *elf_getdata(Elf_Scn *scn, Elf_Data *data);
Elf_Data *elf_newdata(Elf_Scn *scn);
Elf_Data *elf_rawdata(Elf_Scn *scn, Elf_Data *data);

说明

这些函数可以访问和操作与段描述符 *scn* 关联的数据。读取现有的文件时，将为段提供一个与之关联的单一数据缓冲区。程序可以按片段来构建新段，但是，需要通过多个数据缓冲区组合新数据。出于此原因，查看段的“这些”数据时应将其当作缓冲区列表，而其中的每个缓冲区可以通过数据描述符获得。

elf_getdata 允许程序逐步执行段的数据列表。如果输入数据描述符 *data* 为空，函数将返回与段关联的第一个缓冲区。否则， *data* 应该是与 *scn* 关联的数据描述符，函数将为程序授予段的下一个数据元素的访问权限。如果 *scn* 为空，或发生错误， **elf_getdata** 将返回空指针。

elf_getdata 将数据由文件表示形式转换为内存表示形式（请参阅 *elf_xlate*(3E) ），同时根据文件的 *class* （请参阅 *elf*(3E) ）为程序提供带有内存数据类型的对象。工作库版本（请参阅 *elf_version*(3E) ）指定程序希望 **elf_getdata** 提供的内存结构版本。

elf_newdata 为段创建新的数据描述符，并将其追加到与段关联的任何数据元素。如下文所述，新数据描述符显示为空，表示元素中没有数据。为方便起见，将描述符类型（下面所述的 d_type ）设置为 **ELF_T_BYTE** ，将版本（下面所述的 d_version ）设置为工作版本。程序负责根据需要设置（或更改）描述符成员。此函数隐式设置段的数据（请参阅 *elf_flag*(3E) ）的 **ELF_F_DIRTY** 位。如果 *scn* 为空，或发生错误， **elf_newdata** 将返回空指针。

elf_rawdata 与 **elf_getdata** 的不同之处在于，它只返回未解释的字节，而不论段类型如何。通常，只有需要从读取的文件中检索段映像，以及程序必须避免以下所述的自动数据转换时，才应该使用此函数。此外，在执行初始操作之前，程序不可以关闭或禁用（请参阅 *elf_cntl*(3E) ）与 *elf* 关联的文件描述符，原因是 **elf_rawdata** 可以从文件读取数据，确保不影响 **elf_getdata** 。有关适用于整个文件的相关工具，请参阅 *elf_rawfile*(3E) 。如果 **elf_getdata** 能够提供正确的转换，则建议用它取代 **elf_rawdata** 。如果 *scn* 为空，或发生错误， **elf_rawdata** 将返回空指针。

Elf_Data 结构包括以下成员。

```
Elf_Void        *d_buf;
Elf_Type        d_type;
Elf64_Xword     d_size;
Elf64_Off       d_off;
Elf64_Xword     d_align;
unsigned        d_version;
```

程序可以直接操作这些成员。下文提供了说明。

d_buf 指向数据缓冲区的指针驻留在此处。不包含数据的数据元素具有空指针。

d_type 此成员的值指定 **d_buf** 指向的数据的类型。段类型确定如何解释段的内容，下文提供了概述。

d_size 此成员保留数据占用的内存总大小(以字节为单位)。此大小可能与文件中表示的大小不同。如果不存在数据，此大小为零（有关详细信息，请参阅下文 **SHT_NOBITS** 的讨论）。

d_off 此成员在缓冲区所驻留的段中指定偏移量。此偏移量相对于文件的段，而不是内存对象的段。

d_align 此成员保留缓冲区自段开始位置的必要对齐。也就是说， **d_off** 是此成员值的倍数。例如，如果此成员的值为四，则缓冲区的起始位置将在段中进行四字节对齐。此外，整个段应根据其最大范围的组成结构进行对齐，以确保为段和文件中的缓冲区提供正确的对齐。

d_version 此成员保留缓冲区中对象的版本号。当库最初从对象文件读取数据时，会使用工作版本来控制内存对象转换。

数据对齐

如上所述，段中的数据缓冲区具有明显的对齐限制。因此，相邻的缓冲区有时无法邻接，而导致在段中产生“漏洞”。创建输出文件的程序具有两种方法处理这些漏洞。

第一种方法，程序使用 **elf_fill** 通知库应如何设置介入其中的字节。如果库必须在文件中生成间隔，则需要使用填充字节来初始化此位置的数据。库最初的填充值为 0，应用程序可以通过 **elf_fill** 更改此值。

第二种方法，应用程序可以生成用来占用这些间隔的自身数据缓冲区，并使用适用于所创建的段的数据来填充间隔。程序甚至可以对不同的段使用不同的填充值。例如，它可以将文本段的字节设置为无操作指令，同时使用零来填充数据段的漏洞。如果使用这种方法，库将找不到要填充的漏洞，因为应用程序已将其消除。

段和内存类型

elf_getdata 根据区段头（可通过 *elf_getshdr*(3E) 获取）中注明的段类型解释段的数据。下表显示了段类型，以及库如何使用 32 位文件类的内存数据类型表示这些段类型。其他类应具有类似的表。内存数据类型可以通过 *elf_xlate*(3E) 隐式控制转换。

段类型	**Elf_Type**	**32** 位类型
SHT_DYNAMIC	***ELF_T_DYN***	***Elf32_Dyn***
SHT_DYNSYM	***ELF_T_SYM***	***Elf32_Sym***
SHT_HASH	***ELF_T_WORD***	***Elf32_Word***
SHT_NOBITS	***ELF_T_BYTE***	无符号字符
SHT_NOTE	***ELF_T_BYTE***	无符号字符
SHT_NULL	无	无
SHT_PROGBITS	***ELF_T_BYTE***	无符号字符
SHT_REL	***ELF_T_REL***	***Elf32_Rel***
SHT_RELA	***ELF_T_RELA***	***Elf32_Rela***
SHT_STRTAB	***ELF_T_BYTE***	无符号字符
SHT_SYMTAB	***ELF_T_SYM***	***Elf32_Sym***
其他	***ELF_T_BYTE***	无符号字符

elf_rawdata 可使用类型 **ELF_T_BYTE** 创建缓冲区。

如上所述，程序的工作版本将控制库应该为应用程序创建哪些结构。同样，库根据版本转换段类型。如果段类型“属于”的版本高于应用程序的工作版本，则库不转换段数据。因为在这种情况下，应用程序无法知道数据格式，库只提供 **ELF_T_BYTE** 类型的未转换缓冲区，如同为未识别段类型提供的缓冲区一样。

即使区段头指示了非零大小，带特殊类型 **SHT_NOBITS** 的段也不会在对象文件中占用任何空间。 **elf_getdata** 和 **elf_rawdata** 将在这样的段上“工作”，并设置 *data* 结构，使之包含空缓冲区指针，及上文所述的类型。虽然不提供数据，但可以将 **d_size** 值设置为区段头的大小。当程序创建 **SHT_NOBITS** 类型的新段时，应使用 **elf_newdata** 将数据缓冲区添加到此段。应该将这些“空”数据缓冲区的 **d_size** 成员设置为所需的大小，并将其 **d_buf** 成员设置为 null。

举例

以下片段将获取保留段名称的字符串表（忽略错误检查）。有关字符串表的处理差异，请参阅 *elf_strptr*(3E) 。

```
ehdr = elf32_getehdr(elf);
scn = elf_getscn(elf, (size_t)ehdr->e_shstrndx);
shdr = elf32_getshdr(scn);
if (shdr->sh_type != SHT_STRTAB)
{
      /* not a string table */
}
data = 0;
if ((data = elf_getdata(scn, data)) == 0
   || data->d_size == 0)
{
      /* error or no data */
}
```

ELF 头中的 **e_shstrndx** 成员保留字符串表的段表索引。程序将获取该段的段描述符，再验证它是否为字符串表，然后检索数据。此段结束时， **data->d_buf** 将指向字符串表的第一个字节，同时 **data->d_size** 将保留以字节为单位的字符串表的大小。

另请参阅

elf(3E)、 elf_cntl(3E)、 elf_fill(3E)、 elf_flag(3E)、 elf_getehdr(3E)、 elf_getscn(3E)、 elf_getshdr(3E)、 elf_rawfile(3E)、 elf_strptr(3E)、 elf_version(3E)、 elf_xlate(3E)。

名称

elf_getehdr： elf32_getehdr()、 elf32_newehdr()、 elf64_getehdr()、 elf64_newehdr() - 为 elf32 或 elf64 文件检索与类相关的对象文件标题

概要

命令： **cc** [*flag*]... *file*... **-lelf** [*library*]...

#include <libelf.h>

Elf32_Ehdr *elf32_getehdr(Elf *_elf_);

Elf32_Ehdr *elf32_newehdr(Elf *_elf_);

Elf64_Ehdr *elf64_getehdr(Elf *_elf_);

Elf64_Ehdr *elf64_newehdr(Elf *_elf_);

说明

32 位类文件

对于 32 位的类文件， **elf32_getehdr()** 将返回指向 ELF 标题的指针（如果有可用于 ELF 描述符的标题 *elf* ）。如果没有用于此描述符的标题， **elf32_newehdr()** 就会分配一个“干净”的 ELF 标题，但其操作与 **elf32_getehdr()** 的相同。如果已存在标题，则它不会分配新标题。如果标题不存在（对于 **elf_getehdr()**），且无法予以创建（对于 **elf_newehdr()**），则说明发生系统错误，或文件不是 32 位类文件，或者 *elf* 为空，因此两个函数返回空指针。标题包括下列组成部分。

```
unsigned char     e_ident[EI_NIDENT];
Elf32_Half        e_type;
Elf32_Half        e_machine;
Elf32_Word        e_version;
Elf32_Addr        e_entry;
Elf32_Off         e_phoff;
Elf32_Off         e_shoff;
Elf32_Word        e_flags;
Elf32_Half        e_ehsize;
Elf32_Half        e_phentsize;
Elf32_Half        e_phnum;
Elf32_Half        e_shentsize;
Elf32_Half        e_shnum;
Elf32_Half        e_shstrndx;
```

elf32_newehdr() 将自动设置 **ELF_F_DIRTY** 位（请参阅 *elf_flag*(3E)）。程序可以利用 **elf_getident** 检查文件中的标识字节。

64 位类文件

对于 64 位的类文件， **elf64_getehdr()** 将返回指向 ELF 标题的指针（如果有可用于 ELF 描述符的标题 *elf* ）。如果用于此描述符的标题不存在， **elf64_newehdr()** 就会分配一个“干净”的 ELF 标题，但其操作与 **elf64_getehdr()**的相同。如果已存在标题，则它不会分配新标题。如果标题不存在（对于 **elf_getehdr()** ），且无法予以创建（对于 **elf_newehdr()** ），则说明发生系统错误，或文件不是 64 位类文件，或者 *elf* 为空，因此两个函数返回空指针。标题包括下列组成部分。

```
unsigned char   e_ident[EI_NIDENT];
Elf64_Half      e_type;
Elf64_Half      e_machine;
Elf64_Word      e_version;
Elf64_Addr      e_entry;
Elf64_Off       e_phoff;
Elf64_Off       e_shoff;
Elf64_Word      e_flags;
Elf64_Half      e_ehsize;
Elf64_Half      e_phentsize;
Elf64_Half      e_phnum;
Elf64_Half      e_shentsize;
Elf64_Half      e_shnum;
Elf64_Half      e_shstrndx;
```

elf64_newehdr() 将自动设置 **ELF_F_DIRTY** 位（请参阅 *elf_flag*(3E)）。程序可以利用 **elf_getident** 检查文件中的标识字节。

另请参阅

elf(3E)、 elf_begin(3E)、 elf_flag(3E)、 elf_getident(3E)。

名称

elf_getident - 检索文件标识数据

概要

cc [*flag*...] *file*... **-lelf** [*library*] ...

#include <libelf.h>

char *elf_getident(Elf *elf, size_t *ptr);

说明

如 *elf*(3E) 所说明，ELF 为各类文件提供一种框架，其中基本对象可以有 32 位、64 位等等。为了适应这些不同，而不在位数较小的机器上强制使用位数较大的对象，那么 ELF 文件中的最初字节保存了所有类型的文件都具有的标识信息。每个 ELF 标头的 **e_ident** 都有 **EI_NIDENT** 个字节，其解释如下。

e_ident 索引	值	用途
EI_MAG0	**ELFMAG0**	文件标识
EI_MAG1	**ELFMAG1**	
EI_MAG2	**ELFMAG2**	
EI_MAG3	**ELFMAG3**	
EI_CLASS	**ELFCLASSNONE**	文件类
	ELFCLASS32	
	ELFCLASS64	
EI_DATA	**ELFDATANONE**	数据编码
	ELFDATA2LSB	
	ELFDATA2MSB	
EI_VERSION	**EV_CURRENT**	文件版本
EI_OSABI	**ELFOSABI_NONE**	**OS/ABI** 标识
	ELFOSABI_HPUX	
EI_ABIVERSION	**0**	**ABI** 版本
	1	
EI_PAD		开始填充字节
10-15	**0**	尚未使用，设置为零

其他种类的文件（请参阅 *elf_kind*(3E) ）也可以有标识数据，尽管它们与 **e_ident** 不一致。

elf_getident 返回一个指向文件的“初始字节”的指针。如果库识别出文件，可能会将文件映像转换为内存映像。在任何情况下，尽管未修改区域的大小取决于文件类型，但要确保不修改标识字节。如果 *ptr* 不为空，则库将标识字节的数量存储在 *ptr* 所指向的位置。如果没有数据，则 *elf* 为空，或出现一个错误，返回值为一个空指针，并通过 *ptr* 存储零值（可选）。

另请参阅

elf(3E)、 elf_begin(3E)、 elf_getehdr(3E)、 elf_kind(3E)、 elf_rawfile(3E)。

名称

elf32_getphdr、 elf32_newphdr、 elf64_getphdr 和 elf64_newphdr - 分别检索与类相关的 elf32 和 elf64 文件的程序标题表

概要

cc [*flag*...] *file*... **-lelf** [*library*] ...

#include <libelf.h>

Elf32_Phdr *elf32_getphdr(Elf *elf);
Elf32_Phdr *elf32_newphdr(Elf *elf, size_t count);
Elf64_Phdr *elf64_getphdr(Elf *elf);
Elf64_Phdr *elf64_newphdr(Elf *elf, size_t count);

说明

对于 32 位类文件，如果有一个文件对于 ELF 描述符 *elf* 可用，则 **elf32_getphdr** 返回一个指向程序执行标题表的指针。

elf32_newphdr 分配一个包含 *count* 条目的新表（而不管以前是否存在一个），并设置表的 **ELF_F_DIRTY** 位（请参阅 *elf_flag*(3E) ）。指定 *count* 为零会删除现有表。请注意，此行为不同于 **elf32_newehdr** 的行为（请参阅 *elf_getehdr*(3E) ），允许程序替换或删除程序标题表、更改其大小（如有必要）。

如果没有程序标题表，则文件不是一个 32 位类文件，并会发生错误，或者 *elf* 为空，两个函数都返回一个空指针。另外，如果 *count* 为零，则 **elf32_newphdr** 返回一个空指针。

此表是一组 **Elf32_Phdr** 结构，每种结构都包含下列成员。

```
Elf32_Word      p_type;
Elf32_Off       p_offset;
Elf32_Addr      p_vaddr;
Elf32_Addr      p_paddr;
Elf32_Word      p_filesz;
Elf32_Word      p_memsz;
Elf32_Word      p_flags;
Elf32_Word      p_align;
```

ELF 标题的 **e_phnum** 成员指示程序标题表有多少条目（请参阅 *elf_getehdr*(3E) ）。程序可以检查此值来确定现有表的大小； **elf32_newphdr** 自动将此成员的值设置为 *count* 。如果程序正在构建一个新文件，则它负责在创建程序标题表之前创建此文件的 ELF 标题。

对于 64 位类文件，如果有一个文件对于 ELF 描述符 *elf* 可用，则 **elf64_getphdr** 返回一个指向程序执行标题表的指针。

elf64_newphdr 分配一个包含 *count* 条目的新表（而不管以前是否有一个），并设置此表的 **ELF_F_DIRTY** 位（请参阅 **elf_flag**(3E) ）。指定 *count* 为零会删除现有表。请注意，此行为不同于 **elf64_newehdr** 的行为（请参阅

elf_getehdr(3E) ），允许程序替换或删除程序标题表、更改其大小（如有必要）。

如果没有程序标题表，则文件不是一个 64 位类文件，发生错误，或者 *elf* 为空，两个函数都返回一个空指针。另外，如果 *count* 为零，则 **elf64_newphdr** 将返回一个空指针。

此表是一组 **Elf64_Phdr** 结构，每种结构都包含下列成员。

```
Elf64_Word      p_type;
Elf64_Off       p_offset;
Elf64_Addr      p_vaddr;
Elf64_Addr      p_paddr;
Elf64_Xword     p_filesz;
Elf64_Xword     p_memsz;
Elf64_Word      p_flags;
Elf64_Xword     p_align;
```

ELF 标题的 e_phnum 成员指示程序标题表有多少条目（请参阅 *elf_getehdr*(3E) ）。程序可以检查此值来确定现有表的大小； **elf64_newphdr** 自动将此成员的值设置为 *count* 。如果程序正在构建一个新文件，则它负责在创建程序标题表之前创建此文件的 ELF 标题。

另请参阅

elf(3E)、elf_begin(3E)、elf_flag(3E)、elf_getehdr(3E)。

名称

elf_getscn()、 elf_ndxscn()、 elf_newscn()、 elf_nextscn() - 获取 ELF 文件信息

概要

命令： **cc** [*flag*]... *file*... **-lelf** [*library*]...

#include <libelf.h>

Elf_Scn *elf_getscn(
**Elf ***elf*,
size_t *index*
);

size_t elf_ndxscn(
**Elf_Scn ***scn*
);

Elf_Scn *elf_newscn(
**Elf ***elf*
);

Elf_Scn *elf_nextscn(
Elf * *elf*,
**Elf_Scn ***scn*
);

说明

通过这些函数，能以索引和有序的方式访问与 ELF 描述符 *elf* 关联的段。如果程序需要构建一个新文件，那么在创建段之前，它应负责创建该文件的 ELF 标头；请参阅 *elf_getehdr*(3E) 。

如果为文件的段标头表提供了一个 索引， **elf_getscn()** 将返回一个段描述符。请注意，第一个“实际”段包含索引 1。虽然程序可以为 索引等于 0 的段（不确定的段 **SHN_UNDEF** ）获取段描述符，但是该段不包含数据，并且段标头是“空的”（尽管存在）。如果指定的段不存在，将会出现错误；如果 *elf* 为空， **elf_getscn()** 将返回空指针。

elf_newscn() 可创建一个新段，并将其追加到 *elf* 的列表中。因为 **SHN_UNDEF** 段是必需的，并且与应用程序“无关”，库将自动创建该段。为不存在段的 ELF 描述符首次调用 **elf_newscn()** ，将返回段 1 的描述符。如果出错，或者 *elf* 为空， **elf_newscn()** 将返回空指针。

创建一个新的段描述符后，程序可以使用 **elf_getshdr()** 检索新建的“干净”段标头。新的段描述不包含关联的数据（请参阅 *elf_getdata*(3E) ）。如果以这种方式创建新段，库将更新 ELF 标头的 **e_shnum** 成员，同时设置该段的 **ELF_F_DIRTY** 位（请参阅 *elf_flag*(3E) ）。如果程序需要构建一个新文件，那么在创建新段之前，它应负责创建该文件的 ELF 标头（请参阅 *elf_getehdr*(3E) ）。

elf_nextscn() 使用现有的段描述符 *scn* ，并返回下一个更高位段的段描述符。可以使用空的 *scn* 获取索引为 1 的段的段描述符（跳过索引为 **SHN_UNDEF** 的段）。如果不存在其他段，或者出错， **elf_nextscn()** 将返回空指

针。

elf_ndxscn() 使用现有的段描述符 *scn* ，并返回其段表索引。如果 *scn* 为空，或者出错， **elf_ndxscn()** 将返回 **SHN_UNDEF** 。

举例

下面显示有序访问的示例。每完成一次循环后，将处理文件中的下一个段；处理完所有段后，循环将会终止。

```
scn = 0;
while ((scn = elf_nextscn(elf, scn)) != 0)
{
    /* process section */
}
```

另请参阅

elf(3E)、 elf_begin(3E)、 elf_flag(3E)、 elf_getdata(3E)、 elf_getehdr(3E)、 elf_getshdr(3E)。

名称

elf32_getshdr 和 elf64_getshdr - 分别检索 elf32 和 elf64 文件的与类相关的区段 (section) 头

概要

cc [*flag*...] *file*... **-lelf** [*library*] ...

#include <libelf.h>

Elf32_Shdr *elf32_getshdr(Elf_Scn *scn);
Elf64_Shdr *elf64_getshdr(Elf_Scn *scn);

说明

对于 32 位类文件， **elf32_getshdr** 返回指向区段描述符 *scn* 的区段头的指针。否则，文件并非 32 位类文件， *scn* 为空，或者出现错误；因此 **elf32_getshdr** 返回 NULL。区段头包括下列成员。

```
Elf32_Word      sh_name;
Elf32_Word      sh_type;
Elf32_Word      sh_flags;
Elf32_Addr      sh_addr;
Elf32_Off       sh_offset;
Elf32_Word      sh_size;
Elf32_Word      sh_link;
Elf32_Word      sh_info;
Elf32_Word      sh_addralign;
Elf32_Word      sh_entsize;
```

对于 64 位类文件， **elf64_getshdr** 返回指向区段描述符 *scn* 的区段头的指针。否则，文件并非 64 位类文件， *scn* 为空，或者出现错误；因此 **elf64_getshdr** 返回 NULL。区段头包括下列成员。

```
Elf64_Word      sh_name;
Elf64_Word      sh_type;
Elf64_Xword     sh_flags;
Elf64_Addr      sh_addr;
Elf64_Off       sh_offset;
Elf64_Xword     sh_size;
Elf64_Word      sh_link;
Elf64_Word      sh_info;
Elf64_Xword     sh_addralign;
Elf64_Xword     sh_entsize;
```

如果程序要构建新文件，则它负责在创建区段之前创建文件的 ELF 头。

另请参阅

elf(3E)、elf_flag(3E)、elf_getscn(3E)、elf_strptr(3E)。

名称

elf_hash - 计算散列值

概要

cc [*flag*...] *file*... **-lelf** [*library*] ...

#include <libelf.h>

unsigned long elf_hash(const char *name);

说明

在给定空的终止字符串 *name* 的情况下， **elf_hash** 可计算散列值。返回的散列值 *h* 可用作容器索引（通常在计算 *h* mod *x* 之后），以确保正确绑定。

由于 **elf_hash** 使用了无符号算术运算来避免因不同计算机的符号运算而可能导致的差别，因此可以在一台计算机上构建散列表，然后在另一台计算机上使用。尽管 *name* 显示为上面的 **char*** ，但 **elf_hash** 却将其视为 **unsigned char*** ，以免导致符号扩展差异。使用 **char*** 可以消除与 **elf_hash** （“名称”）这类表达式的冲突。

ELF 文件的符号散列表就是用此函数进行计算的（请参阅 *elf_getdata*(3E) 和 *elf_xlate*(3E) ）。返回的散列值保证不是全为 1 的位模式 **(~0UL)** 。

另请参阅

elf(3E)、elf_getdata(3E)、elf_xlate(3E)。

名称

elf_kind - 确定文件类型

概要

cc [*flag*...] *file*... **-lelf** [*library*] ...

#include <libelf.h>

Elf_Kind elf_kind(Elf *elf);

说明

此函数返回标识了与 ELF 描述符 *elf* 相关联的文件的类型的值。当前定义的值如下所示。

ELF_K_AR 此文件是归档文件（请参阅 *ar*(4) ）。ELF 描述符也可与归档 *member* （而非归档自身）相关联，并由 **elf_kind** 标识成员的类型。

ELF_K_ELF 此文件是 ELF 文件。程序可使用 **elf_getident** 来确定类。其他函数（如 **elf_getehdr** ）则可用于检索其他文件信息。

ELF_K_NONE 它表示库未知的某种文件。

其他值将予以保留，并根据需要分配给新的文件类型。 *elf* 应为某个以前由 **elf_begin** 返回的值。可以使用 null 指针以简化错误处理过程，并使 **elf_kind** 返回 **ELF_K_NONE** 。

另请参阅

ar(4)、elf(3E)、elf_begin(3E)、elf_getehdr(3E)、 elf_getident(3E)。

名称

elf_next - 顺序归档成员访问

概要

cc [*flag*...] *file*... **-lelf** [*library*] ...

#include <libelf.h>

Elf_Cmd elf_next(Elf *elf);

说明

elf_next 、 **elf_rand** 和 **elf_begin** 将处理简单对象文件和归档。 *elf* 是一个以前从 **elf_begin** 返回的 ELF 描述符。

elf_next 将提供对下一归档成员的顺序访问。也就是说，如果使用与归档成员相关联的 ELF 描述符 *elf* ，则 **elf_next** 将准备包含的归档在程序调用 **elf_begin** 时访问后面的成员。成功定位下一成员的归档后， **elf_next** 将返回值 **ELF_C_READ** 。否则就说明打开的文件不是归档文件、 *elf* 为空或发生错误，此时返回值为 **ELF_C_NULL**。在任一情况下，返回值都可以作为参数传递给 **elf_begin** ，以指定相应的操作。

另请参阅

ar(4)、elf(3E)、elf_begin(3E)、 elf_getarsym(3E)、elf_rand(3E)。

名称

elf_rand - 随机归档成员访问

概要

cc [*flag*...] *file*... **-lelf** [*library*] ...

#include <libelf.h>

size_t elf_rand(Elf *elf, size_t offset);

说明

elf_rand 、 **elf_next** 和 **elf_begin** 将处理简单对象文件和归档。 *elf* 是一个以前从 **elf_begin** 返回的 ELF 描述符。

elf_rand 将提供随机归档处理过程，并准备 *elf* 以访问任意归档成员。 *elf* 必须是归档自身的描述符，而非归档中的成员。 *offset* 提供了从归档开始位置到所需成员的归档标头的字节偏移量。有关归档成员偏移量的信息，请参阅 *elf_getarsym*(3E) 。当 **elf_rand** 生效时，它将返回 *offset* 。否则它将返回 0，原因是出现了错误， *elf* 为空值或此文件不属于归档（任何归档成员的偏移量都不能是零）。程序可以混用随机归档和连续归档过程。

举例

归档以带有 **SARMAG** 字节的“magic 字符串”开头；初始归档成员紧随其后。这样，应用程序可提供以下函数对归档进行回卷（出错时函数返回 -1，其他情况则返回 0）。

```
#include <ar.h>
#include <libelf.h>

int
rewindelf(Elf *elf)
{
    if (elf_rand(elf, (size_t)SARMAG) == SARMAG)
        return 0;
    return -1;
}
```

另请参阅

ar(4)、elf(3E)、elf_begin(3E)、 elf_getarsym(3E)、elf_next(3E)。

名称

elf_rawfile - 检索未解释的文件内容

概要

cc [*flag*...] *file*... **-lelf** [*library*] ...

#include <libelf.h>

char *elf_rawfile(Elf *elf, size_t *ptr);

说明

elf_rawfile 可返回指向文件的未解释字节映像的指针。只能使用此函数检索读取的文件。例如，程序可使用 **elf_rawfile** 检索某个归档成员的字节。

在初次调用 **elf_rawfile** 之前，程序不可以关闭或禁用（请参阅 *elf_cntl*(3E) ）与 *elf* 关联的文件描述符，这是因为，如果 **elf_rawfile** 的内存中没有初始字节，它可能需要从文件读取数据。通常，针对未知的文件类型使用此函数比针对对象文件使用此函数更有效。库可以隐式转换内存中的对象文件，但不修改未知的文件。因此，请求某个对象文件的未解释映像可以在内存中创建一个复本。

elf_rawdata （请参阅 *elf_getdata*(3E) ）是一个相关的函数，可提供对文件中的段的访问权限。

如果 *ptr* 非空，库还可以在 *ptr* 指向的位置存储文件的字节大小。如果不存在任何数据， *elf* 为空，或者出错，那么返回值将是一个空指针，同时有选择性地通过 *ptr* 存储零。

注释

使用 **elf_rawfile** ，并且也解释与对象文件相同的文件的程序，可能拥有两份内存字节。如果这样的程序首先请求原始映像，那么在该程序请求转换的信息（通过 **elf_getehdr** 、 **elf_getdata** 等函数）之前，库将“冻结”原始映像的初始内存副本。然后程序将此冻结的副本用作创建已转换对象的源，而不必再次读取文件。因此，除非应用程序需要更改其拥有的、以后转换的数据视图，否则应将 **elf_rawfile** 返回的原始文件映像视为只读的缓冲区。对于任何一种情况，应用程序都可以更改已转换的对象，且不必更改原始映像中的可见字节。

使用相同的 ELF 描述符多次调用 **elf_rawfile** 将返回相同的值；库不创建文件的复本。

另请参阅

elf(3E)、elf_begin(3E)、elf_cntl(3E)、 elf_getdata(3E)、 elf_getehdr(3E)、elf_getident(3E)、elf_kind(3E)。

名称

elf_strptr - 生成字符串指针

概要

cc [*flag*...] *file*... **-lelf** [*library*] ...

#include <libelf.h>

char *elf_strptr(Elf *elf, size_t section, size_t offset);

说明

此函数把字符串区段的 *offset* 转换为字符串指针。 *elf* 标识了该字符串区段所在的文件，而 *section* 则提供此字符串区段的表索引。 **elf_strptr** 通常返回指向字符串的指针，但是，如果 *elf* 为 null（空），或者 *section* 无效或不是 **SHT_STRTAB** 类型的区段，或者无法获取到此区段的数据，或者 *offset* 无效，或者发生错误，在这些情况下将返回一个空指针。

举例

下面显示了一个用于检索区段名称的原型。文件头指定了 **e_shstrndx** 成员中区段名称的字符串表。以下代码将在区段之间循环并输出其名称。

```
if ((ehdr = elf32_getehdr(elf)) == 0)
{
    /* handle the error */
    return;
}
ndx = ehdr->e_shstrndx;
scn = 0;
while ((scn = elf_nextscn(elf, scn)) != 0)
{
    char    *name = 0;
    if ((shdr = elf32_getshdr(scn)) != 0)
       name = elf_strptr(elf, ndx,
                  (size_t)shdr->sh_name);
    printf("'%s'\n", name? name: "(null)");
}
```

注释

程序可以调用 **elf_getdata** 以检索整个字符串表区段。对于某些应用程序而言，这样要比使用 **elf_strptr** 更为高效，更为便捷。

另请参阅

elf(3E)、elf_getdata(3E)、elf_getshdr(3E)、 elf_xlate(3E)。

名称

elf_update -更新 ELF 描述符

概要

cc [*flag*...] *file*... **-lelf** [*library*] ...

#include <libelf.h>

off_t elf_update(Elf *elf, Elf_Cmd cmd);

说明

elf_update 能使库检查与某个 ELF 描述符 *elf* 关联的信息，及重新计算生成文件映像所需的结构数据。

cmd 包括以下值。

ELF_C_NULL 此值通知 **elf_update** 重新计算各个值，同时只更新 ELF 描述符的内存结构。任何已修改的结构都带有 **ELF_F_DIRTY** 位标记。这样，程序就可以更新结构信息，然后对其重新检查，且不更改与 ELF 描述符关联的文件。由于这不会更改文件，因此 ELF 描述符允许读取、写入或读写（请参阅 *elf_begin*(3E)）。

ELF_C_WRITE 如果 *cmd* 包含此值， **elf_update** 将重复其 **ELF_C_NULL** 操作，同时写入与文件的 ELF 描述符关联的任何“脏”信息。也就是说，当程序使用 **elf_getdata** 或 **elf_flag** 设备为某个 ELF 描述符提供新信息（或更新现有信息）时，将检查、调整这些数据，必要时将转换这些数据（请参阅 *elf_xlate*(3E)），同时将数据写入文件。对文件的某些部分执行写入后，将重置任何 **ELF_F_DIRTY** 位，指示不再需要将这些项目写入文件（请参阅 *elf_flag*(3E)）。按段的段标头条目的顺序写入段的数据，将段标头表写入文件的末尾。

如果使用 **elf_begin** 创建了 ELF 描述符，则该 ELF 描述符必须允许写入文件。也就是说，**elf_begin** 命令必须为 **ELF_C_RDWR** 或 **ELF_C_WRITE** 。

如果 **elf_update** 成功，将返回文件映像（不是内存映像）的以字节为单位的总大小。否则，将出现错误，同时函数返回 -1。

更新内部结构时， **elf_update** 将自行设置某些成员。下列成员由应用程序负责，这些成员保留了程序指定的值。

	成员	注释
ELF 标头	**e_ident[EI_DATA]**	库控制其他 **e_ident** 值
	e_type	
	e_machine	
	e_version	
	e_entry	
	e_phoff	仅当已断言 **ELF_F_LAYOUT** 时

	e_shoff	仅当已断言 **ELF_F_LAYOUT** 时
	e_flags	
	e_shstrndx	

	成员	注释
	p_type	应用程序控制所有
	p_offset	程序标头条目
	p_vaddr	
	p_paddr	
程序标头	**p_filesz**	
	p_memsz	
	p_flags	
	p_align	

	成员	注释
	sh_name	
	sh_type	
	sh_flags	
	sh_addr	
	sh_offset	仅当已断言 **ELF_F_LAYOUT** 时
段标头	**sh_size**	仅当已断言 **ELF_F_LAYOUT** 时
	sh_link	
	sh_info	
	sh_addralign	仅当已断言 **ELF_F_LAYOUT** 时
	sh_entsize	

	成员	注释
	d_buf	
	d_type	
	d_size	
数据描述符	**d_off**	仅当已断言 **ELF_F_LAYOUT** 时
	d_align	
	d_version	

请注意，程序负责 ELF 标头中两个特别重要的成员（除其他成员外）。 **e_version** 成员控制已写入文件的数据结构的版本。如果版本为 **EV_NONE** ，则库使用自己的内部版本。 **e_ident[EI_DATA]** 条目控制文件中使用的数据编码。作为一种特殊情况，值可能为 **ELFDATANONE** ，以请求主机的本地数据编码。在这种情况下，如果本地编码与库已知的文件编码不匹配，则会出现错误。

此外请注意，程序负责 **sh_entsize** 段标头成员。虽然它是库使用已知的类型为段设置的，但它不能确认所有段的值是否正确。因此，库需要依赖程序来提供未知段类型的值。如果条目大小未知或不适用，则应将值设置为零。

确定如何构建输出文件时， **elf_update** 将遵循各个数据缓冲区的对齐要求来创建输出段。由段的已经过最严格对齐的数据缓冲区控制该段的对齐。必要时，库还会在缓冲区之间插入填充字符，以确保适当地对齐每个缓冲区。

注释

如上所述， **ELF_C_WRITE** 命令将数据写入文件之前，会根据需要对其进行转换。对应用程序来说，这种转换不 *not* 一定是透明的。如果程序获取了指向与某个文件（有关示例，请参阅 *elf_getehdr*(3E) 和 *elf_getdata*(3E)）关联的数据的指针，那么程序在调用 **elf_update** 之后应重新建立指针。

如 *elf_begin*(3E) 所述，程序可以“更新”COFF 文件，使映像与 ELF 一致（在某些计算机体系结构上（如 Intel），COFF 是加在 ELF 前面的对象文件格式。当程序调用 COFF 文件上的 elf_begin 时，库将 COFF 结构转换为其 ELF 等效结构，以便程序将 COFF 文件当作 ELF 文件来读取（但不可写入）。此转换操作只适用于内存映像，而不适用于文件自身）。 **ELF_C_NULL** 命令只能更新内存映像；也可以使用 **ELF_C_WRITE** 命令修改文件。绝对可执行文件（ **a.out** 文件）需要进行特殊的对齐，在 COFF 和 ELF 之间通常无法做到这一点。因此，用户不可以使用 **ELF_C_WRITE** 命令（虽然允许 **ELF_C_NULL** ）更新 COFF 可执行文件。

另请参阅

elf(3E) 、 elf_begin(3E) 、 elf_flag(3E) 、 elf_fsize(3E) 、 elf_getdata(3E) 、 elf_getehdr(3E) 、 elf_getshdr(3E) 、 elf_xlate(3E)。

名称

elf_version() - 协调 ELF 库和应用程序版本

概要

命令： **cc** [*flag*]... *file*... **-lelf** [*library*]...

#include <libelf.h>

unsigned elf_version(unsigned *ver***);**

说明

如 *elf*(3E) 所说明，程序、库和对象文件都有独立的“最新”ELF 版本的表示形式。 **elf_version()** 允许程序确定 ELF 库的 内部版本。它还允许程序将自己的 工作版本 *ver* 提供给库，以指定所使用的内存类型。使用 ELF 库的每个程序都必须按如下所述协调版本。

头文件 **<libelf.h>** 使用宏 **EV_CURRENT**

给程序提供版本。如果库的内部版本（对于库来说，是已知的最高版本）低于程序本身的已知版本，则此库可能缺少程序所假定的语义知识。相应地， **elf_version** 将不接受库未知的工作版本。

传送的 *ver* 等于 **EV_NONE** 时，会导致 **elf_version()** 返回库的内部版本，但不更改工作版本。如果 *ver* 是库已知的版本，则 **elf_version()** 返回以前的（最初的）工作版本号。否则，工作版本保留不变，而 **elf_version()** 返回 **EV_NONE** 。

举例

下面从应用程序摘录的代码段保护程序自身，使之免于使用旧库。

```
if (elf_version(EV_CURRENT) == EV_NONE)
{
  /* library out of date */
  /* recover from error */
}
```

警告

对于特定 ELF 描述符的所有操作，工作版本都应该是相同的。更改描述符操作之间的版本可能不会提供预期的结果。

另请参阅

elf(3E)、 elf_begin(3E)、 elf_xlate(3E)。

名称

elf32_xlatetof、 elf32_xlatetom、 elf64_xlatetof、 elf64_xlatetom - 分别对 elf32 文件和 elf64 文件进行与类相关的数据转换

概要

cc [*flag*...] *file*... **-lelf** [*library*] ...

#include <libelf.h>

Elf_Data *elf32_xlatetof(Elf_Data *dst, const Elf_Data *src,
unsigned encode);
Elf_Data *elf32_xlatetom(Elf_Data *dst, const Elf_Data *src,
unsigned encode);
Elf_Data *elf64_xlatetof(Elf_Data *dst, const Elf_Data *src,
unsigned encode);
Elf_Data *elf64_xlatetom(Elf_Data *dst, const Elf_Data *src,
unsigned encode);

说明

elf32_xlatetom 可将不同的数据结构由 32 位类文件表示形式转换为内存表示形式； **elf64_xlatetom** 可将不同的数据结构由 64 位类文件表示形式转换为内存表示形式； **elf32_xlatetof** 和 **elf64_xlatetof** 提供相反的转换。对于跨开发环境来说，这种转换特别重要。 *src* 是指向保存初始数据的源缓冲区的指针； *dst* 是指向保存已转换副本的目标缓冲区的指针。 *encode* 提供表示（或要表示）文件对象的字节编码形式，它必须具有为 ELF 标头的 **e_ident[EI_DATA]** 条目（请参阅 *elf_getident*(3E) ）定义的编码值之一。如果可以转换数据，则函数返回 *dst* 。否则，函数将由于出错（例如类型不兼容、目标缓冲区溢出等等）而返回 NULL。

elf_getdata(3E) 描述转换例行程序按以下方式使用的 Elf_Data 描述符。

d_buf 源和目标都必须具有有效的缓冲区指针。

d_type 此成员的值指定 **d_buf** 指向的数据的类型，以及要在目标中创建的数据的类型。程序可提供源中的 **d_type** 值；库可将目标的 **d_type** 设置为相同的值。下面汇总了这些值。

d_size 此成员保存源数据占用的内存的总字节大小，以及为目标数据分配的大小。如果目标缓冲区的大小不足，则例行程序不更改其初始内容。转换发生后，转换例行程序将目标的 **d_size** 成员重置为所需的实际大小。源大小和目标大小可能不同。

d_version 此成员保存缓冲区中（所需的）对象的版本号。源版本和目标版本是彼此独立的。

转换例行程序允许源缓冲区和目标缓冲区一致。也就是说， **dst->d_buf** 可以等于 **src->d_buf** 。对于其他情况，如果源缓冲区和目标缓冲区重叠，则行为是不确定的。

对于 elf32 文件：

Elf_Type	**32** 位内存类型
ELF_T_ADDR	**Elf32_Addr**
ELF_T_BYTE	带符号字符
ELF_T_DYN	**Elf32_Dyn**
ELF_T_EHDR	**Elf32_Ehdr**
ELF_T_HALF	**Elf32_Half**
ELT_T_OFF	**Elf32_Off**
ELF_T_PHDR	**Elf32_Phdr**
ELF_T_REL	**Elf32_Rel**
ELF_T_RELA	**Elf32_Rela**
ELF_T_SHDR	**Elf32_Shdr**
ELF_T_SWORD	**Elf32_Sword**
ELF_T_SYM	**Elf32_Sym**
ELF_T_WORD	**Elf32_Word**

对于 elf64 文件：

Elf_Type	**64** 位内存类型
ELF_T_ADDR	**Elf64_Addr**
ELF_T_BYTE	带符号字符
ELF_T_DYN	**Elf64_Dyn**
ELF_T_EHDR	**Elf64_Ehdr**
ELF_T_HALF	**Elf64_Half**
ELT_T_OFF	**Elf64_Off**
ELF_T_PHDR	**Elf64_Phdr**
ELF_T_REL	**Elf64_Rel**
ELF_T_RELA	**Elf64_Rela**
ELF_T_SHDR	**Elf64_Shdr**
ELF_T_SWORD	**Elf64_Sword**
ELF_T_SYM	**Elf64_Sym**
ELF_T_WORD	**Elf64_Word**

ELF_T_BYTE 类型的“转换”缓冲区不更改字节顺序。

另请参阅

elf(3E)、 elf_fsize(3E)、 elf_getdata(3E)、 elf_getident(3E)。

名称

end、 edata、 etext、 __data_start、 __text_start - 程序中的最后一个位置

概要

extern void *_end, *end, *_etext, *etext, *_edata, *edata, *__data_start, *__text_start;

说明

这些名称既不引用例行程序，也不引用包含相关内容的位置。符号 **_etext** 和 **etext** 的地址是程序文本上方的第一个地址， **_edata** 和 **edata** 的地址是已初始化数据区域上方的第一个地址， **_end** 和 **end** 的地址是未初始化数据区域上方的第一个地址。

符号 **__data_start** 的地址是程序数据区域的开始地址， **__text_start** 是程序文本区域的开始地址。

如果程序引用了这些符号但未定义，则链接程序以适当的数值定义它们。如果用户试图定义 **_edata** 、 **_etext** 、 **_end** 、 **__data_start** 、或 **__text_start** ，则链接程序会发出错误。

当开始执行时，程序中断（超出数据的第一个位置）与 **_end** 一致，但程序中断可以由 *brk*(2)、 *malloc*(3C) 的例行程序、标准输入／输出 (*stdio*(3S)) 、 *cc*(1) 的配置 (**-p**) 选项等重设。因此，程序中断的当前值应当由 **sbrk(0)** （请参阅 *brk*(2)）确定。

警告

在 C 中，这些名字必须看起来像地址。因此，使用 **&end** 代替 **end** 来访问 *end* 的当前值。

另请参阅

系统工具

cc(1)	调用 HP-UX C 编译程序
ld(1)	调用链接编辑器

其他信息

brk(2)	更改数据段空间分配
crt0(3)	执行启动例行程序
malloc(3C)	主内存分配器
stdio(3S)	标准的缓冲输入/输出流文件程序包

符合的标准

end()：XPG2

edata()：XPG2

etext()：XPG2

名称

endwin — 挂起 Curses 会话

概要

#include <curses.h>

int endwin(void);

说明

endwin() 函数将通过至少恢复所保存的 Shell 终端模式、刷新终端的任何输出以及将光标移动到屏幕最后一行的第一列，在 Curses 操作后恢复终端。刷新窗口将恢复程序模式。退出前，应用程序必须针对所用的每个终端调用 **endwin()** 。如果为同一终端多次调用了 **newterm()** ，则所创建的第一个屏幕必须为调用 **endwin()** 的最后一个屏幕。

返回值

成功完成后， **endwin()** 将返回 OK。否则，它将返回 ERR。

错误

未定义任何错误。

实际应用信息

endwin() 函数不会释放与屏幕有关的存储空间。因此，如果已不再需要某个特定的屏幕，则应在 **endwin()** 之后调用 **delscreen()** 。

要临时离开 Curses 模式，可移植的应用程序应调用 **endwin()** 。以后要想返回 Curses 模式，应调用 **doupdate()** 、**refresh()** 或 **wrefresh()** 。

另请参阅

delscreen(3X)、 doupdate(3X)、 initscr(3X)、 isendwin(3X)、 <curses.h>。

历史变更记录

在 X/Open Curses 第 2 期中首次发布。

X/Open Curses 第 **4** 期

为了清楚说明，此条目已经过重新编写。参数列表已明确声明为 **void**。

名称

erasechar 和 killchar — 单字节终端环境查询函数

概要

#include <curses.h>

char erasechar(void);

char killchar(void);

说明

erasechar() 函数返回当前的清除字符。 **erasewchar()** 在 *ch* 指向的对象中存储当前的清除字符。如果没有定义清除字符，则此函数将失败，并且 *ch* 指向的对象将不会更改。

killchar() 函数返回当前抹行字符。

返回值

erasechar() 函数返回清除字符， **killchar()** 返回抹行字符。如果这些字符是多字节字符，则返回值是未指定的。

错误

没有定义任何错误。

实际应用信息

erasechar() 和 **killchar()** 函数只保证对这样的字符集进行可靠操作：其中的每个字符都属于单字节，只能用带有 A_ 前缀的常量来表示其属性。此外，它们没有可靠地指示没有分别定义清除字符和抹行字符的情况。 **erasewchar()** 和 **killwchar()** 函数克服了这些限制。

另请参阅

clearok(3X)、 delscreen(3X)、 erasewchar(3X)、 curses_intro(3X)，请参阅 Attributes 、<curses.h>、 tcattribute(3C)（位于《X/Open System Interfaces and Headers, Issue 4, Version 2》规范中）。

历史变更记录

在 X/Open Curses 第 2 期中首次发布。

X/Open Curses 第 4 期

killchar() 函数合并到这一条目中。在前几期中，它出现在自己的条目中。

为清楚起见，重新编写了此条目。 **erasechar()** 和 **killchar()** 函数的参数列表显式声明为 **void**。

名称

erasewchar、killwchar — 终端环境查询函数

概要

#include <curses.h>

int erasewchar(wchar_t *_ch_);

int killwchar(wchar_t *_ch_);

说明

erasewchar() 函数将存储 *ch* 所指向的对象中的当前清除字符。如果未定义清除字符，函数将会失败，且不会更改 *ch* 所指向的对象。

killwchar() 函数将存储 *ch* 所指向的对象中的当前行抹行字符。如果未定义行抹行字符，则函数将会失败，且不会更改 *ch* 所指向的对象。

返回值

成功完成后， **erasewchar()** 和 **killwchar()** 将返回 OK。否则，它们将返回 ERR。

错误

未定义任何错误。

另请参阅

curses_intro(3X)、 clearok(3X)、 delscreen(3X)、 <curses.h>、 tcattribute(3C) （位于《X/Open System Interfaces and Headers, Issue 4, Version 2》规范中）中的 Attributes 。

历史变更记录

在 X/Open Curses 第 4 期中首次发布。

名称

erf( )、erff( )、erfl( )、erfw( )、erfq( )、erfc( )、erfcf( )、erfcl( )、erfcw( )、erfcq( ) - 误差函数与补余误差函数

概要

```
#include <math.h>

double erf(double x);

double erfc(double x);
```

仅适用于 **HP Integrity** 服务器

```
float erff(float x);

long double erfl(long double x);

extended erfw(extended x);

quad erfq(quad x);

float erfcf(float x);

long double erfcl(long double x);

extended erfcw(extended x);

quad erfcq(quad x);
```

说明

erf() 返回 *x* 的误差函数，定义为：

(2/**sqrt**(**pi**)) *
（对 *t* 从 0 至 *x* 的 (**exp**(**pow**(-*t*,2))) 积分）

erfc() 返回互补值 1.0 – **erf**(*x*) 。如果对于大 *x* 调用 **erf**(*x*) 并从 1.0 中减去此结果，则其将防止相对精度的极值损失（例如，对于 *x* = 5，损失 12 个小数位）。

仅适用于 **Integrity** 服务器

erff() 与 **erfcf()** 分别是 **float** 版的 **erf()** 和 **erfc()** ；它们使用 **float** 参数，并返回 **float** 结果。

erfl() 和 **erfcl()** 分别是 **long double** 版的 **erf()** 和 **erfc()** ；它们使用 **long double** 参数，并返回 **long double** 结果。

erfw() 和 **erfcw()** 分别是 **long double** 版的 **erf()** 和 **erfc()** ；它们使用 **extended** 参数，并返回 **extended** 结果。

在 HP-UX 系统中， **erfq()** 和 **erfcq()** 分别等效于 **erfl()** 和 **erfcl()** 。

用法

要使用这些函数，请使用缺省的 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。

（对于 Integrity 服务器）要使用 **erfw()** 、 **erfcw()** 、 **erfq()** 或 **erfcq()** ，也可以使用 **-fpwidetypes** 选项编译。

要使用上面的任何函数，请确保程序包含 **<math.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

有关详细信息，请参阅《HP-UX floating-point guide for HP Integrity servers》（位于以下站点：**http://www.hp.com/go/fp**）。

返回值

erf(±0) 返回 ±0。

如果 *x* 为 +INFINITY，**erf()** 返回 1.0。

如果 *x* 为 –INFINITY，**erf()** 返回 –1.0。

如果 *x* 为 +INFINITY，**erfc()** 返回 0。

如果 *x* 为 –INFINITY，**erfc()** 返回 2.0。

未指定 **erf()** 与 **erfc()** 是否会产生不精确异常。

如果 *x* 为 NaN，**erf()** 与 **erfc()** 返回 NaN。

错误

没有定义任何错误。

另请参阅

exp(3M)、pow(3M)、sqrt(3M)、math(5)。

符合的标准

erf() 和 **erfc()**：SVID3、XPG4.2、ISO/IEC C99（包括附件 F“IEC 60559 floating-point arithmetic”）

erff()、**erfl()**、**erfcf()** 和 **erfcl()**：ISO/IEC C99（包括附件 F“IEC 60559 floating-point arithmetic”）

名称

EvmConnCheck()、 EvmConnWait()、 EvmConnDispatch()、 EvmConnFlush() - 与 EVM 守护程序保持连接

概要

```
#include <sys/time.h>
#include <evm/evm.h>

EvmStatus_t EvmConnCheck (
    EvmConnection_t  connection
    EvmBoolean_t *IOWaiting  );

EvmStatus_t EvmConnWait (

    EvmConnection_t  connection,
    const struct timeval *timeout  );

EvmStatus_t EvmConnDispatch (
    EvmConnection_t  connection  );

EvmStatus_t EvmConnFlush (
   EvmConnection_t  connection,
   EvmInt32_t *count  );
```

库

EVM 支持库 (**libevm.so**) 。

参数

connection 要监视的 EVM 连接。请参阅 *EvmConnCreate*(3) 联机帮助页。

IOWaiting 这是返回操作数。下面是该操作数的可能值:

EvmFALSE 连接上没有未完成的 I/O 活动。

EvmTRUE 连接上存在未完成的 I/O 活动。客户端需要调用 **EvmConnDispatch()** 来处理未完成的活动。

timeout 如果 *timeout* 的值为 NULL，则指定 **EvmConnWait()** 一直等到连接上出现活动为止。如果 *timeout* 的值不为 NULL，则指定在连接上出现活动之前 **EvmConnWait()** 要等待的时间。

count 接收由于要发送到 EVM 守护程序而排队的消息的计数。

说明

EvmConnCheck() 例行程序可以检查连接上是否存在任何未完成的 I/O 活动。如果存在未完成活动，则将 *IOWaiting* 设置为 **EvmTRUE** 。然后，程序调用 **EvmConnDispatch()** 来处理该活动。如果不存在未完成的 I/O 活动，则将 *IOWaiting* 设置为 **EvmFALSE** 。来自该函数的 **EvmTRUE** 响应不会保证事件已到达； **EvmTRUE** 响应只表示连接上存在需要处理的消息。

在连接上检测到活动之前， **EvmConnWait()** 例行程序将一直阻塞。如果超时的值不为 NULL，则指定在返回状

态 **EvmERROR_TIMEOUT** 之前，该函数等待连接上出现活动的时间。

EvmConnDispatch() 例行程序将根据需要调用客户端的回调函数，处理连接上某一个传入的 I/O 消息。如果不存在等待处理的消息，函数将立即返回。调用此函数不保证会调用连接的回调函数，同时，即使调用了回调函数，也不表示事件一定已经到达。

如果由于以前在标记为非阻塞的连接上调用了 **EvmEventPost()** 而使消息排队以进行输出， **EvmConnFlush()** 例行程序会尝试向 EVM 守护进程发送所有这些消息。该例行程序将持续发送队列中的消息，直到发送完所有的消息，或者发送缓冲区已满为止。返回时，如果 *count* 输出参数是一个大于零的数值，那么该数值就是仍在排队等待发送的消息的数量。 **EvmConnFlush** 返回 **EvmERROR_QUEUED** 。有关详细信息，请参阅 *EvmConnControl*(3) 。

返回值

EvmERROR_NONE 操作已完成且未发生错误。

EvmERROR_INVALID_ARGUMENT
函数的某个参数无效。

EvmERROR_INVALID_VALUE
结构成员中的某个值无效。

EvmERROR_NO_MEMORY 尝试获取堆内存失败，导致操作失败。

EvmERROR_READ 从 EVM 守护程序连接中读取时发生读取错误。

EvmERROR_WRITE 写入 EVM 守护程序连接时发生写入错误。

EvmERROR_SELECT EVM 连接上发生错误。调用 **EvmConnDestroy()** 以破坏连接。

EvmERROR_TIMEOUT 超过了超时时间。

EvmERROR_SIGNAL 由于收到了信号，当前操作被中断。

EvmERROR_QUEUED 一个或多个消息正在排队，以等待发送到 EVM 守护程序。

错误

不设置 **errno** 的值。

文件

/var/evm/sockets/evmd 域套接字的缺省路径名。

另请参阅

函数

connect(2)、select(2)、socket(2)。

例行程序

EvmConnControl(3)、EvmConnCreate(3)、EvmConnDestroy(3)、EvmEventPost(3)。

事件管理
EVM(5)。

事件回调
EvmCallback(5)。

事件连接
EvmConnection(5)。

EVM 事件
EvmEvent(5)。

名称

EvmConnControl() - 控制 EVM 连接的信息

概要

#include <evm/evm.h>

EvmStatus_t EvmConnControl(
EvmConnection_t *connection*,
EvmInt32_t *request*,
void ***arg);**

库

EVM 支持库 (**libevm.so**) 。

参数

connection 要控制的 EVM 连接。请参阅 *EvmConnCreate*(3) 联机帮助页。

request 请求的控制操作。有关可用请求的信息，请参阅“说明”一节。

arg *arg* 的值和用法取决于 *request* 。有关可用含义和与 *request* 的关联，请参阅“说明”一节。

说明

EvmConnControl() 例行程序可设置或获取连接的控制信息。

下面的列表显示 *request* 参数的可用值以及 *arg* 参数的关联值。

request 参数 *arg* 参数值

EvmCONN_RCV_SZ_GET
获取连接的接收缓冲区的大小。 *arg* 参数是指向接收大小值的 **EvmInt32_t** 字段的指针。

EvmCONN_SND_SZ_GET
获取连接的发送缓冲区的大小。 *arg* 参数是指向接收大小值的 **EvmInt32_t** 字段的指针。

注释: HP-UX 当前不支持设置接收和发送缓冲区大小的 **EvmCONN_RCV_SZ_SET** 和 **EvmCONN_SND_SZ_SET** 。而应使用 **EvmConnFdGet()** 获取连接的套接字描述符，使用 **setsockopt()** 更改套接字的缓冲区大小。

EvmCONN_CHECKSUM_ENABLE
为发布的事件启用校验和生成。这是建立连接时的缺省状态。必须将 *arg* 参数设置为 NULL。

EvmCONN_CHECKSUM_DISABLE
为发布的事件禁用校验和生成。此选项可供使用本地 EVM 守护程序频繁发布事件的高性能应用程序使用。必须将 *arg* 参数设置为 NULL。

EvmCONN_CHECKSUM_IS_ENABLED

查询连接以了解是否启用了校验和生成。 *arg* 参数是指向接收布尔型值的 **EvmBoolean_t** 字段的指针。

EvmCONN_POST_ITEM_MASK_SET

修改插入到特定事件中的环境数据项的集合，这些事件发布到所提供的连接。 *arg* 参数是一个 **EvmItemMask_t** 值，包含要插入项目值的位图。使用 **EvmItemMask_()** 宏可将项目枚举值转换为位图格式。例如，

EvmItemMask_(EvmITEM_TIMESTAMP) | EvmItemMask_(EvmITEM_USER_NAME)

EvmCONN_POST_ITEM_MASK_RESET

恢复自动插入到特定事件中的环境数据项的缺省集合，这些事件发布到所提供的连接。必须将 *arg* 参数设置为 NULL 。此请求将清除以前任何 **EvmCONN_POST_ITEM_MASK_SET** 请求的影响。

EvmCONN_POST_NONBLOCK_SET

在发布事件的尝试可能溢出连接发送缓冲区的情况下，控制调用进程的行为。如果将此选项设置为 **EvmFALSE** （建立连接时的缺省状态），则进程将根据需要进行阻塞，以便将消息同步传送到 EVM 守护程序。

如果将其设置为 **EvmTRUE** ，则在不阻塞就无法完全发送事件时， **EvmEventPost()** 将返回 **EvmERROR_QUEUED** ，并且稍后该进程必须调用 **EvmConnFlush()** ，以避免在内存中堆积未发送的消息。对 **EvmEventPost()** 的进一步调用还将刷新该队列。有关详细信息，请参考 *EvmConnFlush*(3) 联机帮助页。

不能将此选项用于除具有 **EvmRESPONSE_IGNORE** 响应模式的发布连接之外的任何连接类型。此选项是为必须在极短的时间内发布大量事件的关键高性能进程提供的。该选项不适合一般的用途。

arg 参数是一个 **EvmBoolean_t** 值。 **EvmFALSE** 值允许进程根据需要进行阻塞，而 **EvmTRUE** 值则防止阻塞并启用消息排队。

返回值

EvmERROR_NONE	操作已完成且未发生错误。
EvmERROR_INVALID_ARGUMENT	函数的某个参数无效。
EvmERROR_INVALID_VALUE	结构成员中的某个值无效。
EvmERROR_NO_MEMORY	尝试获取堆内存失败，导致操作失败。
EvmERROR_RESOURCE	尝试获取或控制系统资源时出现意外错误。
EvmERROR_SIGNAL	由于收到了信号，当前操作被中断。

错误

不设置 **errno** 的值。

另请参阅

函数

connect(2)、select(2)。

例行程序

EvmConnCheck(3)、EvmConnCreate(3)、EvmConnSubscribe(3)、EvmConnWait(3)。

事件管理

EVM(5)。

事件连接

EvmConnection(5)。

名称

EvmConnCreate()、 EvmConnCreatePoster()、 EvmConnCreateSubscriber()、 EvmConnDestroy()、 EvmConnFdGet() - 建立或破坏与 EVM 守护程序的连接

概要

```
#include <evm/evm.h>

EvmStatus_t EvmConnCreate(
    EvmConnectionType_t type,
    EvmResponseMode_t responseMode,
    const EvmTransport_t  transport,
    EvmCallback_t callback,
    EvmCallbackArg_t callbackArg,
    EvmConnection_t *connection );

EvmStatus_t EvmConnCreatePoster(
    EvmConnection_t *connection );

EvmStatus_t EvmConnCreateSubscriber(
    EvmCallback_t callback,
    EvmCallbackArg_t callbackArg,
    EvmConnection_t *connection );

EvmStatus_t EvmConnDestroy(
    EvmConnection_t connection );

EvmStatus_t EvmConnFdGet(
    EvmConnection_t connection,
    EvmFd_t * fd );
```

库

EVM 支持库 (**libevm.so**) 。

参数

type 建立连接功能。可能使用下列值:

EvmCONNECTION_POST
创建由发布客户端使用的连接。

EvmCONNECTION_LISTEN
创建由订阅客户端使用的连接。必须将 *responseMode* 操作数设置为 **EvmRESPONSE_CALLBACK** 。

responseMode 在已向 EVM 守护程序发出请求后，建立面向连接的特定 API 函数的行为。如果函数调用导致了这种请求，该函数将根据此操作数的值，按以下方式返回响应:

EvmRESPONSE_IGNORE

函数将会返回，而不等待守护程序的响应。在此模式下，除了 **EvmERROR_NONE** 之外的任何返回状态均指明在初步验证时检测到了问题。 **EvmERROR_NONE** 响应仅表示已向守护程序发送了请求。 **EvmERROR_NONE** 响应不能确定是收到了请求，还是接受了请求。

EvmRESPONSE_WAIT

函数将执行初步验证，并返回一个状态，以反映所发现的任何错误。如果该函数未发现错误，则将请求传递给 EVM 守护程序，然后将一直阻塞，直到收到守护程序的响应为止。该函数返回的状态可以由初步验证产生，或由守护程序执行的验证产生。如果函数未返回任何错误，则表示守护程序已接受请求。

EvmRESPONSE_CALLBACK

函数将会返回，而不等待响应。除 **EvmERROR_NONE** 之外的任何返回状态均指明在初步验证时检测到问题。

如果未发现任何错误，则函数将请求传递给 EVM 守护程序，然后立即返回，并指示状态 **EvmERROR_NONE** 。调用进程必须使用 **EvmConnWait()** 或相关的函数来监视连接上的 I/O 活动，然后调用 **EvmConnDispatch()** 处理该活动。多数情况下，将调用连接的回调函数来处理传入的消息。

必须使用此响应模式监听连接。

transport 指定与守护程序连接的类型。因为 EVM 目前仅支持本地连接守护程序，所以必须始终将此参数设置为 NULL。如果此参数的值不为 NULL，便会出现错误。

callback 一个指针，指向由 **EvmConnDispatch()** 调用以处理传入消息的例行程序。必须按 *EvmCallback*(5) 联机帮助页中的说明声明该例行程序。如果响应模式不是 **EvmRESPONSE_CALLBACK** ，则此操作数的值为 NULL。

callbackArg **EvmConnDispatch()** 每次调用回调例行程序时，便将提供此参数。请参阅 *EvmCallback*(5) 。调用进程使用此参数将其自身用于连接的环境数据传递给回调函数。如果不需要任何环境数据，则此操作数的值为 NULL。

connection 对于 **EvmConnCreate()** ， *connection* 是新连接的返回操作数。对于本联机帮助页描述的其他例行程序， *connection* 是要处理的 EVM 连接。

fd 与提供的连接关联的文件号的返回操作数。

说明

EvmConnCreate() 例行程序可以在客户端进程和 EVM 守护程序之间建立连接。如果连接成功，则在连接参数中返回指向新连接环境的指针。

EvmConnCreatePoster() 例行程序是一个可以与本地守护程序建立发布连接的宏。使用此宏等效于在将 *type* 设置为 **EvmCONNECTION_POST** 、 *responseMode* 设置为 **EvmRESPONSE_WAIT** 以及 *transport callback* 和 *callbackArg* 均设置为 NULL 的情况下调用 **EvmConnCreate()** 。

EvmConnCreateSubscriber() 例行程序是一个可以与本地守护程序建立订阅连接的宏。使用此宏等效于在 *type* 设置为 **EvmCONNECTION_LISTEN** *responseMode* 设置为 **EvmRESPONSE_CALLBACK** 以及 *transport* 设置为 NULL 的情况下调用 **EvmConnCreate()** 。

EvmConnFdGet() 例行程序返回与连接关联的文件号。此文件号可由处理多个 I/O 源的客户端应用程序使用，调用 **select()** 以确定连接上何时会出现活动时，将使用这些 I/O 源。程序必须调用 **EvmConnDispatch()** 来处理活动。程序不得关闭文件描述符，也不得针对文件描述符执行任何直接 I/O。

EvmConnDestroy() 例行程序可破坏连接，以及释放与连接关联的任何资源。必须调用该例行程序破坏连接。

返回值

EvmERROR_NONE	操作已完成且未发生错误。
EvmERROR_INVALID_ARGUMENT	函数的某个参数无效。
EvmERROR_INVALID_VALUE	结构成员中的某个值无效。
EvmERROR_NO_MEMORY	尝试获取堆内存失败，导致操作失败。
EvmERROR_READ	从 EVM 守护程序连接中读取时发生读取错误。
EvmERROR_WRITE	写入 EVM 守护程序连接时发生写入错误。
EvmERROR_CONNECT	尝试连接到 EVM 守护程序时发生错误。
EvmERROR_SELECT	EVM 连接上发生错误。调用 **EvmConnDestroy()** 以破坏连接。
EvmERROR_SIGNAL	由于收到了信号，当前操作被中断。

错误

不设置 **errno** 的值。

文件

/var/evm/sockets/evmd	域套接字的缺省路径名。

另请参阅

命令

evmget(1)、evmpost(1)、evmwatch(1)。

函数

connect(2)、select(2)、socket(2)。

例行程序

EvmConnControl(3)、EvmConnDispatch(3)、EvmConnWait(3)。

事件管理

EVM(5)。

事件回调

EvmCallback(5)。

事件连接

EvmConnection(5)。

EVM 事件

EvmEvent(5)。

《EVM Programmer's Guide》。

名称

EvmConnSubscribe()、 EvmConnRegistrationGet()、 EvmConnTemplateScan() - 建立事件通知的订阅

概要

#include <evm/evm.h>

EvmStatus_t EvmConnRegistrationGet(
EvmConnection_t *connection*,
const EvmString_t *event_class*,
EvmPointer_t *user_arg* **);**

EvmStatus_t EvmConnSubscribe(
EvmConnection_t *connection*,
**void ***unused*,
EvmString_t *filter_string* **);**

EvmStatus_t EvmConnTemplateScan(
EvmConnection_t *connection*,
const EvmString_t *filter_string*,
EvmPointer_t *user_arg* **);**

库

EVM 支持库 (**libevm.so**) 。

参数

connection 为监听事件建立的 EVM 连接。请参阅 *EvmConnCreate*(3) 联机帮助页。

unused 此参数必须指定为 NULL。

filter_string 指定有意义的事件的字符串。有关 *filter_string* 语法的正式说明，请参阅 *EvmFilter*(5) 联机帮助页。

event_class 至少包含事件名称的一部分的事件名称字符串。事件类不能包含通配符。例如，在 EVM 事件名称 **sys.unix.colad** 中， **sys**、 **unix** 和 **colad** 都是事件名称的组成部分。

user_arg 此参数中提供的值在回调函数的扩展回调结构中返回。该值可供调用代码使用，以便匹配发出的请求的回调。

说明

要在事件发生时收到通知，程序必须使用 **EvmConnCreate()** 例行程序建立到 EVM 守护程序的监听连接。然后该程序必须调用 **EvmConnSubscribe()** ，将过滤器与该连接关联。只要守护程序接收到匹配过滤器的事件，就会将该事件发送到订户。订户必须使用 *EvmConnWait*(3) 联机帮助页中说明的 API 例行程序来处理传入事件。

EvmConnSubscribe() 例行程序将提供的 *filter_string* 传递到 EVM 守护程序，以选择预订的事件集。如果先期验证失败，例行程序将立即返回；否则会将请求传递到守护程序。而守护程序的响应将通过连接的回调函数返回，并返回原因代码 **EvmREASON_SUBSCRIBE_COMPLETE** 。调用方必须使用 **EvmConnWait()** 或相关的函数监

视响应，然后调用 **EvmConnDispatch()** 进行处理。

EvmConnRegistrationGet() 例行程序请求守护程序返回与提供的 *event_class* 对应的事件模板（如果该事件已注册）。守护程序通过连接的回调函数，以 EVM 事件的形式返回事件模板来进行响应，并返回原因代码 **EvmREASON_TEMPLATE_INFO_DELIVERED** 。

如果该事件类已注册，则扩展回调数据结构中的事件字段包含一个与模板对应的事件。调用方必须按照传入事件来处理此事件。

如果未注册该事件类，则将回调状态代码设置为 **EvmERROR_NOT_PRESENT** ，并将结构中的事件句柄设置为 NULL。

EvmConnTemplateScan() 例行程序请求守护程序返回与提供的 *filter_string* 匹配的一系列事件模板。守护程序通过连接的回调函数，以 EVM 事件的形式返回事件模板来进行响应，并返回原因代码 **EvmREASON_TEMPLATE_INFO_DELIVERED** 。

首次调用函数时将返回单个事件模板（如果有一个可用的事件模板）。在 *filter_string* 参数设置为 NULL 的后续调用中，将返回下一个匹配的模板。直到匹配原始 *filter_string* 的所有模板已返回，才继续返回下一个匹配的模板。

如果需要将各回调响应与其原始请求匹配，则调用方必须为循环中的每个请求提供不同的 *user_arg* 。调用方还必须验证每个传入模板是否与预期的请求相关联。

注释

守护程序不返回用户没有访问权限的模板。

要避免丢失传入模板，不要在进行任何回调前发送 **EvmConnRegistrationGet()** 或 **EvmConnTemplateScan()** 请求数据流。请在请求/回调/请求循环中进行。

返回值

EvmERROR_NONE	操作已完成且未发生错误。
EvmERROR_INVALID_ARGUMENT	函数的某个参数无效。
EvmERROR_INVALID_VALUE	结构成员中的某个值无效。
EvmERROR_NO_MEMORY	尝试获取堆内存失败，导致操作失败。
EvmERROR_NOT_PRESENT	对于 **EvmConnRegistrationGet()** 例行程序，请求的事件模板未注册。 对于 **EvmConnTemplateScan()** 例行程序，没有与指定的 *filter_string* 匹配的模板或模板列表已耗尽。
EvmERROR_READ	从 EVM 守护程序连接中读取时发生读取错误。
EvmERROR_WRITE	写入 EVM 守护程序连接时发生写入错误。

错误

不设置 **errno** 的值。

另请参阅

例行程序

EvmConnCreate(3)、EvmConnDestroy(3)、EvmConnWait(3)、EvmStatusTextGet(3)。

文件

evm.auth(4)。

事件管理

EVM(5)。

事件回调

EvmCallback(5)。

事件连接

EvmConnection(5)。

事件过滤器

EvmFilter(5)。

名称

EvmEventCreate()、 EvmEventCreateVa()、 EvmEventDestroy() 与 EvmEventDup() - 创建事件和终结事件

概要

```
#include <evm/evm.h>

EvmStatus_t EvmEventCreate(
    EvmEvent_t *event);

EvmStatus_t EvmEventCreateVa(
    EvmEvent_t *event);
    [ EvmItemId_t item_id,
    const EvmItemValue_t item_value...],
EvmITEM_NONE );

EvmStatus_t EvmEventDestroy(
    EvmEvent_t event );

EvmStatus_t EvmEventDup(
    EvmEvent_t *dupev,
    EvmEvent_t event );
```

注释

方括号 [ ] 不是语法的一部分；它们表示可选参数。

库

EVM 支持库 (**libevm.so**) 。

参数

event	要创建、复制或终结的事件。
item_id	要设置的事件数据项的标识符。有关此操作数的可能的值以及此数据项包含的数据，请参阅 *EvmItemSet*(3) 联机帮助页。
item_value	要存储在 *item_id* 指示的关联项中的值。
dupev	存储重复（新）事件的返回位置。

说明

EvmEventCreate() 例行程序创建空事件。在事件引用参数中返回新创建的事件。

EvmEventCreateVa() 例行程序创建事件并在单个调用中添加提供的项目。项目列表由设置为 **EvmITEM_NONE** 的 *item_id* 操作数的实例终止。在事件引用参数中返回新创建的事件。

EvmEventDestroy() 例行程序终结以前创建的事件，同时释放其内存。如果需要终结事件以确保释放所有关联的内存，必须使用此函数。

EvmEventDup() 例行程序创建所提供事件的副本并在 *dupev* 引用参数中返回新事件。

注释

为避免内存泄漏，必须使用适当的 EVM API 释放例行程序释放从 API 例行程序中提供给调用方的所有结构。

返回值

EvmERROR_NONE	操作已完成且未发生错误。
EvmERROR_INVALID_ARGUMENT	函数的某个参数无效。
EvmERROR_INVALID_VALUE	结构成员中的某个值无效。
EvmERROR_NO_MEMORY	尝试获取堆内存失败，导致操作失败。

错误

不设置 **errno** 的值。

另请参阅

命令

evmpost(1)。

例行程序

EvmConnCreate(3)、EvmEventPost(3)、EvmItemSet(3)、EvmEventWrite(3)。

事件管理

EVM(5)。

EVM 事件

EvmEvent(5)。

名称

EvmEventDump() - 按可显示的格式转储事件

概要

```
#include <evm/evm.h>
#include <stdio.h>

EvmStatus_t EvmEventDump(
    EvmConstEvent_t event,
    FILE *fd );
```

库

EVM 支持库 (**libevm.so**) 。

参数

event	请求格式设置转储的事件。显示该事件中包含的所有数据项和变量。
fd	指向设置了格式的事件将转储到的 FILE 结构。其通常为标准输出或标准错误。

说明

EvmEventDump() 例行程序按显示格式转储事件。显示该事件中包含的所有数据项和变量。

注释

版本间的输出格式不能保证一致。不要依赖于任何标准处理的输出。

返回值

EvmERROR_NONE	操作已完成且未发生错误。
EvmERROR_INVALID_ARGUMENT	函数的某个参数无效。
EvmERROR_INVALID_VALUE	结构成员中的某个值无效。
EvmERROR_NO_MEMORY	尝试获取堆内存失败，导致操作失败。

错误

不设置 **errno** 的值。

另请参阅

命令

evmshow(1)。

例行程序

EvmEventFormat(3)。

事件管理

EVM(5)。

EVM 事件

EvmEvent(5)。

名称

EvmEventFormat()、 EvmEventFormatFromTemplate()、 EvmVarFormat() - 设置事件的格式以便于显示

概要

```
#include <evm/evm.h>

EvmStatus_t EvmEventFormat(
    char *buffer,
    size_t length,
    EvmEvent_t event );

EvmStatus_t EvmEventFormatFromTemplate(
    char *buffer,
    size_t length",
    EvmString_t show_template,
    EvmEvent_t event );

EvmStatus_t EvmVarFormat(
    EvmEvent_t event ,
    const char *var_name,
    char *buffer,
    size_t length",
    EvmBoolean_t translate );
```

库

EVM 支持库 (**libevm.so**) 。

参数

buffer 指向接收已设置格式的事件数据的字符串的指针。即使存储的字符串被截断，存储的字符串也将以空字符结尾，但长度操作数为零 (0) 时例外。

length 输出字符串的最大长度（字符数）。如果设置了格式的事件大于 *length* ，则输出将被截断。

event 要设置格式的事件。

show_template 用于设置输出格式的主模板。

var_name 要设置格式的事件的变量数据项的名称。

translate 指明是否要对变量数据项 (*var_name*) 进行 I18N（国际化）翻译的布尔型值。如果 *translate* 操作数为 **EvmTRUE** ，则尝试翻译操作。此外，必须满足下列条件：

- 变量为字符串类型，并包含 I18N（国际化）消息标识符
- 事件包含 I18N（国际化）目录信息。

说明

必须将事件中包含的信息设置为可读的格式。发布事件的软件（事件发布程序）通过事件的格式数据项提供设置了格式的信息。此处所述的例行程序说明了该事件的格式设置功能。

EvmEventFormat() 例行程序将所提供的事件格式数据项与事件中可用的事件格式项内指定的所有数据项或变量相结合。 **EvmEventFormat()** 在 *buffer* 中返回该事件的一个已设置格式的版本。

如果该事件包含 I18N 消息清单信息，且该清单在用户语言环境中可用，则在可能的情况下，将从消息清单中检索格式字符串。如果无法进行转换，则将使用来自该事件的格式字符串。

如果该事件不包含格式信息， **EvmEventFormat()** 则生成事件名称和任何变量的常规显示。

EvmEventFormatFromTemplate() 例行程序还可以设置所提供的事件的格式。 **EvmEventFormatFromTemplate()** 使用所提供的 *show_template* 字符串作为其主格式设置模板。 *show_template* 中的文字字符毫无变化地传输到缓冲区。

如果存在提供的事件，则可以使用其对应数据项的值代替格式 **@***item_name***[%***width***]** 或 **@{***item_name***[%***width***]}** 的任何标记。 *item_name* 是存储在事件中的项的名称。如果提供了 *width* ，该值将占用最小量的 *width* 字符位置。 *width* 是一个整数数值，用于指定 *item_name* 中存储的值的大小。 *item_name* 是一个标准事件数据项的名称。特殊的字符串 **@@** 会替换为将 **EvmEventFormat()** 应用到事件所产生的结果。

有关设置事件格式以便于显示的详细信息，请参考 *evmshow*(1) 和 *EvmEvent*(5) 。

EvmVarFormat() 从所提供的事件中检索一个名为 *var_name* 的变量，并在 *buffer* 中返回其已设置格式的版本的值。

如果 *translate* 为 **EvmTRUE** ，则可以根据需要应用 I18N 翻译。

注释

使用 **printf() %.8g** 格式说明符对浮点值进行格式设置。

返回值

EvmERROR_NONE	操作已完成且未发生错误。
EvmERROR_INVALID_ARGUMENT	函数的某个参数无效。
EvmERROR_INVALID_VALUE	结构成员中的某个值无效。
EvmERROR_NO_MEMORY	尝试获取堆内存失败，导致操作失败。

错误

不设置 **errno** 的值。

另请参阅

命令

evmshow(1)、 locale(1)。

例行程序

EvmEventDump(3)、printf(3S)、setlocale(3C)。

事件管理

EVM(5)。

EVM 事件

EvmEvent(5)。

《EVM Programmer's Guide》。

名称

EvmEventNameMatch()、 EvmEventNameMatchStr() - 匹配 EVM 事件名称

概要

#include <evm/evm.h>

EvmStatus_t EvmEventNameMatch(
const char ****pattern*,**
EvmEvent_t *event***,**
EvmBoolean_t ****match* **);

EvmStatus_t EvmEventNameMatchStr(
const char ****pattern*,**
const char ****candidate*,**
EvmBoolean_t ****match* **);

库

EVM 支持库 (**libevm.so**) 。

参数

pattern 搜索的事件名称模式。模式可以是任何有效的事件名称字符串。模式可以包含通配符，用于代替任何组成部分。

event 包含要与模式进行比较的事件名称的事件。

match 匹配的结果。如果名称与模式匹配，则该操作数将设置为 **EvmTRUE** ；否则，将设置为 **EvmFALSE** 。

candidate 要与模式匹配的字符串

说明

由于在确定候选事件名称是否与已知名称匹配时可以应用特殊的匹配规则，因此，应使用 EVM 名称匹配函数进行匹配，而不应使用 C 字符串比较函数（ **(memcpm()** 、 **strcmp()** ）。EVM 函数会将事件名称与提供的模式进行匹配，忽略在候选名称的末尾追加的组成部分，并正确地匹配通配符。

EvmEventNameMatch 函数将事件和事件名称模式用作输入，并返回一个结果，用于指示事件是否包含与匹配输出参数中的模式相匹配的名称。模式可以是任何有效的事件名称字符串，也可以包含通配符，用于代替任何组成部分。模式中的通配符 ***** 可以与零个或多个名称组成部分匹配。 **?** 只能与一个组成部分匹配。如果事件名称与模式指定的所有组成部分匹配，则会发生匹配，即使该名称具有其他尾随的元素，也是如此。

EvmEventNameMatchStr 函数与 **EvmEventNameMatch** 执行相同的检查，但是前者将字符串用作候选事件名称，而不是从所提供的事件中提取事件名称。

如果名称与模式匹配，这两个函数都会将匹配输出参数设置为 **EvmTRUE** ，否则将其设置为 **EvmFALSE** 。

返回值

EvmERROR_NONE	操作已完成且未发生错误。比较成功。匹配操作数的值指出名称是否与模式匹配。
EvmERROR_INVALID_ARGUMENT	函数的某个参数无效。提供的模式包含无效字符。
EvmERROR_NOT_PRESENT	提供的事件不包含名称。

错误

不设置 **errno** 的值。

另请参阅

例行程序

memccpy(3C)、strcat(3C)。

事件管理

EVM(5)。

EVM 事件

EvmEvent(5)。

《EVM Programmer's Guide》。

名称

EvmEventPost() 和 EvmEventPostVa() - 发布 EVM 事件

概要

#include <evm/evm.h>

EvmStatus_t EvmEventPost(
EvmConnection_t *connection*,
const EvmEvent_t *event*);

EvmStatus_t EvmEventPostVa(
EvmConnection_t *connection*,
const EvmEvent_t *event*);
[EvmItemId_t *item_id"*,
const EvmItemValue_t *item_id"*]...,
EvmITEM_NONE);

库

EVM 支持库 (**libevm.so**) 。

参数

connection 通过此连接发布事件。请参阅 *EvmConnCreate*(3) 。

event 要发布到指定连接的事件。请参阅 *EvmEventCreate*(3) 。

item_id 要设置的事件数据项的标识符。有关该操作数的可能值和数据项中包含的数据的信息，请参阅 *EvmItemSet*(3) 。

item_value 要存储在 item_id 指示的关联项中的值。

说明

EVM 客户端程序使用该页所述的例行程序将事件传递到 EVM 守护程序，以分发给用户。

EvmEventPost() 发布已使用 **EvmEventCreate()** 或 **EvmEventCreateVa()** 创建的事件。

EvmEventPostVa() 可以在单个调用中创建、发布和终结事件。它首先创建一个新事件，同时向该事件添加调用中提供的任何数据项。如果事件的操作数不为 NULL，那么，所提供的事件的内容将合并到新事件中，并发布新事件。新事件在例行程序返回前被终结，但所提供的事件（如果有的话）却保持不变。有关 *item_ids* 和 *item_values*（可以作为操作数传递给该例行程序）的详细信息，请参阅 *EvmItemSetVa*(3) 。

如果将 NULL 作为连接操作数传递给其中任意一个例行程序，该事件将通过临时连接发布到本地守护程序。这一临时的连接将在例行程序返回前被终结。由于创建和终结 EVM 连接的相关开销，应仅在程序极少发布事件时传递 NULL。

如果任一例行程序未能将事件传递到守护程序，则立即通过返回代码报告错误。如果将该事件传递到守护程序，则响应的类型取决于创建连接时指定的响应模式。

注释

1. 这些例行程序不会终结作为事件操作数传递的事件。一旦不再需要该事件，则必须显式调用 **EvmEventDestroy()** 来终结它并释放其内存。
2. 避免在短时间内连续发布大量事件，因为这样可能会填充连接的发送缓冲区，并导致发布进程阻塞，直到 EVM 守护程序释放部分或全部数据。虽然这是进程同步的正常部分，但是某些高性能进程可能会尽可能地减少阻塞。 **EvmConnControl()** 可用于增加发送缓冲区的大小。有关详细信息，请参阅 *EvmConnControl*(3) 。
3. 在使用扩展的 *item_ids* 来提供 **EvmEventPostVa()** 变量项的名称和值时，请确保为它们提供了正确数量的操作数。所有这些项目都需要一个变量名称和值，而某些变量类型还需要其他参数。有关详细信息，请参阅 *EvmItemSetVa*(3) 。

返回值

EvmERROR_NONE	操作已完成且未发生错误。
EvmERROR_EVENT_REJECTED	EVM 守护程序拒绝了该事件。最可能的原因是没有为该事件注册模板或未授权调用方发布该事件。有关详细信息，请参阅 *evmtemplate*(4) 和 *evm.auth*(4) 。
EvmERROR_INVALID_ARGUMENT	函数的某个参数无效。
EvmERROR_INVALID_VALUE	结构成员中的某个值无效。
EvmERROR_NO_MEMORY	尝试获取堆内存失败，导致操作失败。
EvmERROR_QUEUED	连接的输出缓冲区已满，该事件已在堆空间中列队。当对 **EvmEventPost()** 进行下一次调用或调用方调用 **EvmConnFlush()** 时，将再次尝试刷新队列。只有当已调用 **EvmConnControl()** 且将请求参数设置为 **EvmCONN_POST_NONBLOCK_SET** 时，才会为其连接返回该错误代码。
EvmERROR_READ	从 EVM 守护程序连接中读取时发生读取错误。该连接不再可用。调用方需要调用 **EvmConnDestroy()** 来终结它。
EvmERROR_WRITE	写入 EVM 守护程序连接时发生写入错误。该连接不再可用。调用方需要调用 **EvmConnDestroy()** 来终结它。

错误

不设置 **errno** 的值。

另请参阅

例行程序

EvmConnControl(3) 、 EvmConnCreate(3) 、 EvmConnDestroy(3) 、 EvmConnRegistrationGet(3) 、 EvmEventCreate(3)、EvmEventDestroy(3)、 EvmItemSetVa(3)。

事件管理
EVM(5)。

事件连接
EvmConnection(5)。

EVM 事件
EvmEvent(5)。

名称

EvmEventRead() 和 EvmEventWrite() - 执行 EVM 事件传入和传出文件的 I/O

概要

```
#include <evm/evm.h>

EvmStatus_t EvmEventRead(
    EvmFd_t fd,
    EvmEvent_t *event );

EvmStatus_t EvmEventWrite(
    EvmFd_t fd,
    const EvmEvent_t event );
```

库

EVM 支持库 (**libevm.so**) 。

参数

event 对于 **EvmEventRead()** ，是指为事件读取而创建的返回操作数。对于 **EvmEventWrite()** ，是指要被写入的事件。

fd 写入或读取事件的文件描述符。

说明

EvmEventRead() 例行程序创建一个新的事件结构并用从提供的文件描述符 (**fd**) 中读取的事件填充它。

当不再需要该新事件时，必须使用 **EvmEventDestroy()** 例行程序终结它。

EvmEventWrite() 例行程序将提供的事件写入指定的文件描述符 (**fd**) 。它不终结该事件。如果不再需要该事件，可调用 **EvmEventDestroy()** 来终结它。

返回值

EvmERROR_NONE	操作已完成且未发生错误。
EvmERROR_EOF	读取时遇到文件结束条件。
EvmERROR_INVALID_ARGUMENT	函数的某个参数无效。
EvmERROR_NO_MEMORY	尝试获取堆内存失败，导致操作失败。
EvmERROR_READ	读取事件时出错。
EvmERROR_WRITE	写入事件时出错。

错误

不设置 **errno** 的值。

另请参阅

例行程序

EvmEventDestroy(3)。

事件管理

EVM(5)。

EVM 事件

EvmEvent(5)。

名称

EvmEventValidate() - 对事件执行数据完整性检查

概要

#include <evm/evm.h>

EvmStatus_t EvmEventValidate(
EvmEvent_t *event* **);**

库

EVM 支持库 (**libevm.so**) 。

参数

event 将被验证的事件。

说明

EvmEventValidate() 例行程序对事件执行数据完整性检查。数据完整性检查旨在对通过连接接收的事件或者从存储中检索的事件进行验证。请不要对经过调用方修改的事件执行数据完整性检查。

返回值

EvmERROR_NONE	操作已完成且未发生错误。
EvmERROR_INVALID_ARGUMENT	函数的某个参数无效。
EvmERROR_INVALID_VALUE	结构成员中的某个值无效。
EvmERROR_NO_MEMORY	尝试获取堆内存失败，导致操作失败。

错误

不设置 **errno** 的值。

另请参阅

事件管理

EVM(5)。

EVM 事件

EvmEvent(5)。

名称

EvmFilterCreate()、 EvmFilterDestroy()、 EvmFilterIsFile()、 EvmFilterReadFile()、 EvmFilterSet()、 EvmFilterTest() - 事件过滤器求值例行程序

概要

```
#include <evm/evm.h>

EvmStatus_t EvmFilterCreate(
    EvmFilter_t * filter_evaluator );

EvmStatus_t EvmFilterDestroy(:
    EvmFilter_t filter_evaluator );

EvmBoolean_t EvmFilterIsFile(
    const char * filter_string );

char * EvmFilterReadFile(
    const char * filter_string );

EvmStatus_t EvmFilterSet(
    EvmFilter_t filter_evaluator,
    const EvmString_t filter_string );

EvmStatus_t EvmFilterTest(
    EvmFilter_t filter_evaluator,
    EvmEvent_t event,
    EvmBoolean_t *matchFlag );
```

库

EVM 支持库 (**libevm.so**)。

参数

filter_evaluator 对于 **EvmFilterCreate()**， *filter_evaluator* 是已建立的过滤器求值程序的返回句柄。

对于 **EvmFilterSet()**， *filter_evaluator* 用于标识将在后续匹配中使用 *filter_string* 的过滤器求值程序。

对于 **EvmFilterTest()**， *filter_evaluator* 用于标识将对所提供的事件与过滤器字符串进行比较的过滤器求值程序。

对于 **EvmFilterDestroy()**， *filter_evaluator* 用于标识要终结的过滤器求值程序。

filter_string 过滤器求值程序在后续匹配中要使用的过滤器字符串。有关语法的信息，请参阅 *EvmFilter*(5) 联机帮助页。

event 要与当前关联于过滤器求值程序的过滤器字符串比较的事件。

matchFlag 当比较提供的事件和当前关联于过滤器求值程序的过滤器字符串时，所得到的结果。可能的返回值如下：

EvmTRUE 事件与 *filter_string* 匹配。

EvmFALSE 事件与 *filter_string* 不匹配。

说明

过滤器求值例行程序检查事件是否与指定事件过滤器匹配。这些函数对于要将复杂的过滤器传递到 EVM 守护程序的客户端而言，非常有用。然后客户端可以根据过滤器的子部件对输入事件进行测试，以确定适当的操作。

EvmFilterCreate() 例行程序建立了过滤器求值程序的实例，在 *filter_evaluator* 中返回一个句柄。

EvmFilterSet() 例行程序将 *filter_string* 传递到 *filter_evaluator* ，以便在后续匹配中使用。

EvmFilterTest() 例行程序将提供的事件与当前和 *filter_evaluator* 关联的 *filter_string* 进行比较。结果在 *matchFlag* 中返回。可能的返回值如下：

EvmTRUE 事件与 *filter_string* 匹配。

EvmFALSE 事件与 filter_string 不匹配。

EvmFilterIsFile() 和 **EvmFilterReadFile()** 例行程序支持间接过滤器语法 *@filename* 。支持间接过滤器的程序可以使用 **EvmFilterIsFile()** 来确定用户提供的过滤器字符串是否为间接的。如果是，则可以使用 **EvmFilterReadFile()** 将文件扩展为常规过滤器字符串。

如果提供的 *filter_string* 使用间接过滤器表示法， **EvmFilterIsFile()** 将返回 **EvmTRUE** ，否则将返回 **EvmFALSE**。

EvmFilterReadFile() 将解释提供的 *filter_string* ，并将试图打开和读取引用的文件。 **EvmFilterReadFile()** 返回指向包含扩展过滤器字符串的堆空间内存的指针。调用方负责在不再需要内存时释放内存。如果读取该文件时发生错误，将返回 NULL。请注意， **EvmFilterReadFile()** 不会试图验证该文件中包含的过滤器字符串。

EvmFilterDestroy() 例行程序将终结 *filter_evaluator* ，释放所有关联的资源。

返回值

EvmERROR_NONE	操作已完成且未发生错误。
EvmERROR_FILTER_NOT_VALID	传递到过滤器求值程序的过滤器字符串在语法上无效。
EvmERROR_INVALID_ARGUMENT	函数的某个参数无效。
EvmERROR_INVALID_VALUE	结构成员中的某个值无效。
EvmERROR_NO_MEMORY	尝试获取堆内存失败，导致操作失败。
EvmERROR_NOT_INITIALIZED	试图在没有先调用可提供初始过滤器字符串的 **EvmFilterSet()** 情况下，使用过滤器求值程序。

NULL 试图读取过滤器文件失败。

错误

不设置 **errno** 的值。

另请参阅

事件管理

EVM(5)。

EVM 事件

EvmEvent(5)。

事件过滤器

EvmFilter(5)。

名称

EvmItemGet()、 EvmItemListFree()、 EvmItemListGet()、 EvmItemRelease()、 EvmItemSet()、 EvmItemSetVa() - 创建和处理事件项

概要

```
#include <evm/evm.h>

EvmStatus_t EvmItemGet(
    EvmEvent_t event,
    EvmItemId_t item_id,
    EvmItemValue_t *item_value );

EvmStatus_t EvmItemListGet(
    EvmEvent_t event,
    EvmCount_t *item_count,
    EvmItemList_t *itemList );

EvmStatus_t EvmItemRelease(
    EvmItemId_t item_id,
    EvmItemValue_t item_value );

EvmStatus_t EvmItemSet(
    EvmEvent_t event,
    EvmItemId_t item_id,
    EvmItemValue_t item_value );

EvmStatus_t EvmItemSetVa(
    EvmEvent_t event,
    [EvmItemId_t item_id,
    EvmItemValue_t item_value]...,
    EvmITEM_NONE );

void EvmItemListFree(
    EvmItemList_t itemList );
```

库

EVM 支持库 (**libevm.so**) 。

参数

event 包含要处理项目的事件。

item_id 调用的例行程序要设置、获取或释放的事件数据项的标识符。此操作数的可能值以及数据项中包含的数据在下表中标识。有关事件数据项的说明，请参阅 *EvmEvent*(5) 联机帮助页。

标识符	数据项
EvmITEM_NONE	无（控制使用）
EvmITEM_NAME	事件名称
EvmITEM_PRIORITY	优先级
EvmITEM_PID	进程标识符
EvmITEM_PPID	父进程标识符
EvmITEM_EVENT_ID	事件标识符
EvmITEM_TIMESTAMP	发布时间
EvmITEM_REPEAT_COUNT	重复计数
EvmITEM_LAST_TIMESTAMP	上次时间戳
EvmITEM_I18N_MSG_ID	I18N 消息标识符
EvmITEM_I18N_SET_ID	I18N 消息集合标识符
EvmITEM_I18N_CATALOG	I18N 清单
EvmITEM_HOST_IP	主机 IP 地址
EvmITEM_HOST_NAME	主机名
EvmITEM_USER_NAME	用户名
EvmITEM_FORMAT	事件格式
EvmITEM_REF	引用
EvmITEM_VAR	变量
EvmITEM_KERNEL_ONLY	仅内核分发

还可能为每个支持的变量类型提供单独的 *item_id* 。在这些情况下，除 *item_value* 外，还必须提供一个或两个（取决于类型）额外的操作数。有关详细信息，请参阅下面的说明。支持下面的扩展变量项 ID：

标识符	变量类型	参数
EvmITEM_VAR_CHAR	**EvmTYPE_CHAR**	名称、值
EvmITEM_VAR_INT16	**EvmTYPE_INT16**	名称、值
EvmITEM_VAR_INT32	**EvmTYPE_INT32**	名称、值
EvmITEM_VAR_INT64	**EvmTYPE_INT64**	名称、值
EvmITEM_VAR_UINT8	**EvmTYPE_UINT8**	名称、值
EvmITEM_VAR_UINT16	**EvmTYPE_UINT16**	名称、值
EvmITEM_VAR_UINT32	**EvmTYPE_UINT32**	名称、值
EvmITEM_VAR_UINT64	**EvmTYPE_UINT64**	名称、值
EvmITEM_VAR_FLOAT	**EvmTYPE_FLOAT**	名称、值
EvmITEM_VAR_DOUBLE	**EvmTYPE_DOUBLE**	名称、值
EvmITEM_VAR_STRING	**EvmTYPE_STRING**	名称、值
EvmITEM_VAR_STRING_I18N	**EvmTYPE_STRING**	名称、值、I18N 消息 ID
EvmITEM_VAR_OPAQUE	**EvmTYPE_OPAQUE**	名称、值、大小

item_value 存储在项中的值由关联的 *item_id* 指定。对于 **EvmItemGet()** ，它是从事件中提取的值的存储位置。

item_count 返回的操作数。它是 *item_list* 中的条目数，不包括终止 **EvmItem_NONE** 条目。

说明

这些例行程序处理事件中标准项内的数据。

EvmItemSet() 例行程序将 *item_id* 指定的事件数据项的值设置为调用方提供的 *item_value* 。

EvmItemSetVa() 例行程序设置事件中的项目值。 **EvmITEM_NONE** 的 *item_id* 实例终止 *item_id* 和 *item_value* 操作数列表。

包含扩展变量项 ID 简化了编程，因为它允许向事件添加变量项，而不需要先设置 **EvmVarStruct_t** 结构，或在之后进行释放。要使用这些扩展项 ID，则每个 ID 应后接变量名称，再后接适当类型的值。此外，**EvmITEM_VAR_STRING_I18N** 值必须后接一个 I18N 消息 ID， **EvmITEM_VAR_OPAQUE** 值必须后接大小。

EvmItemGet() 例行程序在 *item_value* 引用参数中返回由 *item_id* 指定的事件数据项的值。调用方必须使用 **EvmItemRelease()** 释放为此项分配的所有空间。

EvmItemRelease() 例行程序将释放使用 **EvmItemGet()** 从事件检索 *item_id* 命名的数据项时分配的所有存储空间。

EvmItemListGet() 例行程序在 *itemList* 引用参数中返回所有项的列表，这些项当前为事件定义。列表中的最后一项具有 **EvmITEM_NONE** 的 *item_id* 值。列表中的项的数量（不包括终止空标识符），在 *item_count* 中返回。调用方负责通过调用 **EvmItemListFree()** 释放列表使用的内存。

EvmItemListFree() 例行程序释放项列表使用的内存。 *itemList* 操作数是 **EvmItemListGet()** 生成的事件中的项的列表。

返回值

EvmERROR_NONE	操作已完成且未发生错误。
EvmERROR_INVALID_ARGUMENT	函数的某个参数无效。
EvmERROR_INVALID_VALUE	结构成员中的某个值无效。
EvmERROR_NO_MEMORY	尝试获取堆内存失败，导致操作失败。
EvmERROR_NOT_PRESENT	请求在 *itemList* 或 *varList* 中指定了不属于正在处理的事件的项或变量名称组件。

错误

不设置 **errno** 的值。

另请参阅

命令

evmshow(1)。

例行程序

EvmEventCreate(3)、EvmEventDestroy(3)、EvmEventFormat(3)、EvmEventPost(3)。

事件管理

EVM(5)。

EVM 事件

EvmEvent(5)。

《EVM Programmer's Guide》。

名称

EvmSrvStart()、 EvmSrvMessageGet() - 事件服务函数

概要

#include <evm/evm.h>

EvmStatus_t EvmSrvStart(
char ***service_name*,**
char ***arg_string*,**
EvmTransport_t ***transport*,**
EvmConnection_t ***connection*);**

EvmStatus_t EvmSrvMessageGet(
EvmConnection_t *connection* ,
EvmEvent_t ***event*,**
char ****msg*,**
char ****errmsg*,**
EvmInt32_t ***info*);**

备注

目前不支持将 **EvmSrvMessageGet** 用于一般用途。

库

EVM 支持库 (**libevm.so**) 。

参数

service_name 该服务的名称。该名称必须为 EVM 守护程序配置文件 **/etc/evmdaemon.conf**中定义的服务。

arg_string 如果该操作数不为 NULL，启动服务时，该操作数将被写入连接。

transport 指定与守护程序的连接的类型。因为 EVM 目前仅支持本地连接该守护程序，所以必须始终将 *transport* 设置为 NULL。如果此参数不为 NULL，则会出现错误。

EVM 不支持远程连接。

connection 对于 **EvmSrvStart()** ，此为返回操作数。当调用例行程序由连接来完成时，该程序应将该值传递给 **EvmConnDestroy()** 。

对于 **EvmSrvMessageGet()** ，此为消息将要读取的服务连接。

event 如果该操作数在返回时不为 NULL，则它包含一个由服务返回的 EVM 事件。

msg 如果该操作数在返回时不为 NULL，则它包含一个由服务返回的字符串。

errmsg 如果该操作数在返回时不为 NULL，则它包含一个由服务返回的错误消息字符串。

info 如果该操作数在返回时不为 NULL，则它包含一个由服务返回的数字信息。

说明

EvmSrvStart() 例行程序创建一个到 EVM 守护程序的服务连接，请求启动标识为 *service_name* 的服务。如果该服务对于守护程序是已知的，则该服务将作为该守护程序的子进程启动，并将连接被传输到服务进程。如果 *arg_string* 不为 NULL，则函数将对其追加一个换行符并将其写入连接。

EvmSrvMessageGet() 例行程序读取来自于连接标识的服务连接的单个 EVM 服务协议消息，并将其返回给调用方。该消息可能为过程协议消息，指示服务已启动或终止，或可能携带数据。程序性消息由 API 函数处理。数据消息可能为下列各项之一：

- *event* 中返回到调用方的 EVM 事件
- *msg* 中返回到调用方的字符串
- *errmsg* 中返回到调用方的错误字符串

在各种情况下，当且仅当作为相应参数传递给函数的地址不为 NULL 时，才会将数据返回到调用方。

该函数最多在其中一个返回参数中返回一个非空值，而将其他参数设置为 NULL。如果传入的消息不包含数据（例如，它是一条指示服务正在终止的协议消息），则所有的三个参数都将被设置为 NULL。

消息可能携带一个信息项，如果 *info* 不为 NULL，则将返回该信息项。该值在服务终止协议消息中使用，用于指示服务进程的退出代码。

注释

目前不支持将 **EvmSrvMessageGet** 用于一般用途。

返回值

EvmERROR_NONE 如果将非空指针作为相应参数提供，则收到的数据消息在三个数据返回参数之一中可用。

EvmERROR_SERVICE_TERMINATED
服务指示其正在终止。如果调用方传递一个非空数据参数，则将在 *info* 操作数中返回该服务进程的退出代码。

EvmERROR_EOF 服务连接在接收到服务终止消息之前被服务关闭。如果调用方在该服务被终止后继续尝试调用，则返回该值。

错误

不设置 **errno** 的值。

文件

/etc/evmdaemon.conf evmd 配置文件

另请参阅

命令

evmget(1)。

例行程序
EvmConnDestroy(3)。

文件
evmdaemon.conf(4)。

事件管理
EVM(5)。

事件连接
EvmConnection(5)。

EVM 事件
EvmEvent(5)。

名称

EvmStatusTextGet() - 设置 EVM 状态代码的文本版本格式

概要

#include <evm/evm.h>

EvmStatus_t EvmStatusTextGet(
EvmStatus_t *evm_status*,
EvmString_t *buffer*,
EvmSize_t *nBytes* **);**

库

EVM 支持库 (**libevm.so**) 。

参数

evm_status 要为其准备文本说明的状态代码。

buffer 指向 **EvmStatusTextGet()** 存储已设置格式的事件数据所在的字符串。除非 *nBytes* 操作数为零 (0)，否则即使存储的字符串被截断，存储的字符串也将以空字符结尾。

nBytes 输出字符串的最大长度（字符数）。如果状态的格式设置为长于 *nBytes* ，则将截断输出。

说明

EvmStatusTextGet() 例行程序将 EVM API 状态代码用作输入，并返回一个描述该状态的字符串。该字符串可用于显示错误或调试用途。缓冲区中返回的字符串以空字符结尾。

返回值

EvmERROR_NONE 操作已完成且未发生错误。

EvmERROR_INVALID_ARGUMENT 函数的某个参数无效。

错误

不设置 **errno** 的值。

另请参阅

事件管理

EVM(5)。

名称

EvmVarGet() 、 EvmVarGetOpaque() 、 EvmVarGetString() 、 EvmVarGetType() 、 EvmVarGetXxx() 、 EvmVarListFree() 、 EvmVarListGet() 、 EvmVarRelease() 、 EvmVarSet() 、 EvmVarSetOpaque() 、 EvmVarSetStringI18N()、 EvmVarSetXxx() - 处理事件变量

概要

```
#include <evm/evm.h>

EvmStatus_t EvmVarGet(
    EvmEvent_t event,
    EvmVarName_t v_name,
    EvmVarStruct_t *var );

EvmStatus_t EvmVarGetOpaque(
    EvmEvent_t event,
    EvmVarName_t v_name,
    EvmOpaque_t *item_value,
    EvmSize_t *size );

EvmStatus_t EvmVarGetString(
    EvmEvent_t event,
    EvmVarName_t v_name,
    EvmString_t *item_value,
    EvmI18NMsgId_t *msg_id );

EvmStatus_t EvmVarGetType(
    EvmEvent_t event,
    EvmVarName_t v_name,
    EvmVarType_t *type );

EvmStatus_t EvmVarGetXxx(
    EvmEvent_t event,
    EvmVarName_t v_name,
    evm_type *item_value );

EvmStatus_t EvmVarListFree(
    EvmVarList_t varlist );

EvmStatus_t EvmVarListGet(
    EvmEvent_t event,
    EvmCount_t *var_count,
    EvmVarList_t *varList );

EvmStatus_t EvmVarRelease(
```

EvmVarStruct_t *_var_);

EvmStatus_t EvmVarSet(
EvmEvent_t *event*,
EvmVarName_t *v_name*,
EvmVarType_t *type*,
EvmVarValue_t *item_value*,
EvmI18NMsgId_t *msg_id*,
EvmSize_t *size*);

EvmStatus_t EvmVarSetOpaque(
EvmEvent_t *event*,
EvmVarName_t *v_name*,
const EvmOpaque_t *item_value*,
EvmSize_t *size*);

EvmStatus_t EvmVarSetStringI18N(
EvmEvent_t *event*,
EvmVarName_t *v_name*,
const EvmString_t *item_value*,
EvmI18NMsgId_t *msg_id*);

EvmStatus_t EvmVarSetXxx(
EvmEvent_t *event*,
EvmVarName_t *v_name*,
const evm_type *item_value*);

库

EVM 支持库 (**libevm.so**) 。

参数

event 要处理的事件。

v_name 要设置或获取其数据的事件中的某个变量的名称。

对于 **EvmVarSet()** ，如果存在具有该名称的变量，则修改关联的数据。如果不存在该变量，则会创建该变量并将其添加到事件。要存储的值将传入 *item_value* 。

type 要在 *v_name* 指定的变量中存储的 *item_value* 的类型的枚举常量。该字段的可能值包括：

EvmTYPE_BOOLEAN
EvmTYPE_CHAR
EvmTYPE_INT16
EvmTYPE_INT32
EvmTYPE_INT64

EvmTYPE_UINT8
EvmTYPE_UINT16
EvmTYPE_UINT32
EvmTYPE_UINT64
EvmTYPE_FLOAT
EvmTYPE_DOUBLE
EvmTYPE_STRING
EvmTYPE_OPAQUE

对于 **EvmVarGetType()** ， *type* 操作数是一个返回值。

item_value 要在 *v_name* 指定的变量中存储的值。

msg_id 对于在国际化事件中设置的字符串变量， *msg_id* 可能包含对应于变量值的 I18N 消息标识符。该标识符必须指代与事件关联的消息清单和消息集合（请参阅 *EvmItemSet*(3) 联机帮助页）。

对于其类型不为 **EvmTYPE_STRING** 的变量，将忽略 *msg_id* 操作数。

size 关联的 *item_value* 的大小（字节）。调用方在设置类型为 **EvmTYPE_OPAQUE** 的变量时，必须指定正确的大小。对于其他所有变量类型，将忽略大小。

var 对于 **EvmVarGet()** 例行程序，这是指向输出数据存储到的位置的指针。

对于 **EvmVarRelease()** 例行程序，这是指向要释放的存储空间的指针。

var_count 这是 **EvmVarListGet()** 获取的 *var_list* 中的名称的数目，不包括结尾的空指针。

varlist 从 **EvmVarListGet()** 例行程序返回后，这是一个指针列表，这些指针指向与事件关联的变量的名称。列表中的最后一项为空指针。

对于 **EvmVarListFree()** 例行程序，这是要释放其内存的变量的名称的列表。该列表必须以空指针结尾。

说明

EvmVarSet() 例行程序可设置事件中由 *v_name* 指定的变量数据项的值。该例行程序既可用于添加变量，也可用于更改该变量的值。如果事件中并不存在 *v_name* 变量，则添加该变量。

在设置变量方面， **EvmVarSetXxx()** 、 **EvmVarSetStringI18N()** 和 **EvmVarSetOpaque()** 例行程序与普通 **EvmVarSet()** 例行程序相比，提供了一种更简单的方法。使用这些例行程序，一次调用便可以设置任何类型的变量。要使用这些例行程序，请将例行程序名称中的 *Xxx*() 替换为所需的类型名称，并提供相应类型的值，作为 *item_value* 操作数。支持下列类型：

Boolean、Char、Int16、Int32、Int64、Uint8、Uint16、Uint32、Uint64、Float、 Double、String。

可以使用 **EvmVarSetStringI18N()** 设置具有非零 I18N 消息 ID 的字符串变量。请使用 **EvmVarSetOpaque()** 设置不透明的变量。其中每个例行程序所需的操作数比该组中其他例行程序所需的操作数多一个。

EvmVarGet() 例行程序可以返回 *var* 变量中的 *v_name* 指定的事件变量。调用方必须通过调用 **EvmVarRelease()**

来释放变量使用的任何空间。

EvmVarGetType() 例行程序返回指定的变量的类型。

在检索变量值方面， **EvmVarGetXxx()** 、 **EvmVarGetString()** 和 **EvmVarGetOpaque()** 例行程序与普通 **EvmVarGet()** 例行程序相比，提供了一种更简单的方法。这些例行程序通过一次调用便可以检索特定类型的变量。要使用这些例行程序，请将例行程序名称中的 *Xxx*() 替换为所需的类型名称，并提供一个指向相应类型位置的指针，作为 *item_value* 指针。有关支持的类型的详细信息，请参考上面的 **EvmVarSetXxx()** 说明。

使用 **EvmVarGetString()** 检索字符串变量，如果不希望接收 I18N 消息 ID，请为 *msg_id* 操作数指定 NULL。使用 **EvmVarGetOpaque()** 检索不透明的变量。其中每个例行程序所需的操作数比该组中其他例行程序所需的操作数多一个。

请注意， **EvmVarGetString()** 和 **EvmVarGetOpaque()** 在内存堆中返回其变量值。如果不再需要该空间，则可以使用 **free()** 将其释放。

EvmVarRelease() 例行程序可以释放使用 **EvmVarGet()** 从事件中检索指定的 *var* 时分配的任何存储空间。重要的一点是，如果不再需要该变量，应调用该例行程序，以确保释放与该变量关联的所有内存。不要使用该例行程序来释放使用任何 **EvmVarGetXxx()** 例行程序检索的变量。

EvmVarListGet() 例行程序在 *varList* 引用参数中返回一个指向指针数组的指针，该指针数组指向事件中包含的所有变量的名称，其中的最后一个条目为空指针。 *var_count* 参数将接收列表中名称的数目，该数目未计入结尾空值。调用方必须通过调用 **EvmVarListFree()** 来释放列表使用的内存。

EvmVarListFree() 例行程序将释放变量列表 (*varList*) 使用的内存。

注释

调用方向事件添加不透明的变量时，必须指定大小。

为避免内存泄漏，必须使用相应的例行程序释放从 API 例行程序提供给调用方的内存。

返回值

EvmERROR_NONE	操作已完成且未发生错误。
EvmERROR_INVALID_ARGUMENT	函数的某个参数无效。
EvmERROR_INVALID_VALUE	结构成员中的某个值无效。
EvmERROR_NO_MEMORY	尝试获取堆内存失败，导致操作失败。
EvmERROR_NOT_PRESENT	请求在 **itemList** 或 **varList** 中指定了不属于正在处理的事件的项或变量名称组件。

错误

不设置 **errno** 的值。

另请参阅

事件管理

EVM(5)。

EVM 事件

EvmEvent(5)

《EVM Programmer's Guide》。

名称

exp10()、exp10f()、exp10l()、exp10w()、exp10q() - 以 10 为底数的指数函数

概要

#include <math.h>

仅适用于 **HP Integrity** 服务器

double exp10(double x);

float exp10f(float x);

long double exp10l(long double x);

extended exp10w(extended x);

quad exp10q(quad x);

说明

仅适用于 **Integrity** 服务器

exp10() 返回 10^x。

exp10f() 是 **float** 形式的 **exp10()**；它采用 **float** 参数，并返回 **float** 结果。

exp10l() 是 **long double** 形式的 **exp10()**；它采用 **long double** 参数，并返回 **long double** 结果。

exp10w() 是 **extended** 形式的 **exp10()**；它采用 **extended** 参数，并返回 **extended** 结果。

exp10q() 等效于 HP-UX 系统上的 **exp10l()**。

用法

要使用这些函数，请使用缺省的 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。

要使用 **exp10w()** 或 **exp10q()**，也可以使用 **-fpwidetypes** 选项进行编译。

要使用上面的任何函数，请确保程序包含 **<math.h>**，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

有关详细信息，请参阅《HP-UX floating-point guide for HP Integrity servers》（位于以下站点：**http://www.hp.com/go/fp**）。

返回值

exp10(±0) 将返回 1。

如果 *x* 为正无穷大，**exp10()** 将返回正无穷大。

如果 *x* 为负无穷大，则 **exp10()** 将返回 0。

如果 *x* 为 NaN，则 **exp10()** 将返回 NaN。

exp10() 将返回无穷大以代替量值太大的值，并引发溢出及不精确异常。

只要结果很小（实际异常或为零），**exp10()** 就会引发下溢和不精确异常，并因此丧失数据准确性；如果结果只

是很小，则可能引发这些异常。

只要舍入后的结果不等于算术结果， **exp10()** 就会引发不精确异常。

错误

没有定义任何错误。

另请参阅

cbrt(3M)、 exp(3M)、 exp2(3M)、 expm1(3M)、 log10(3M)、 pow(3M)、 sqrt(3M)、 math(5)。

符合的标准

任何标准均未指定这些函数。

名称

exp2( )、exp2f( )、exp2l( )、exp2w( )、exp2q( ) - 底数为 2 的指数函数

概要

#include <math.h>

double exp2(double x);

仅适用于 **HP Integrity** 服务器

float exp2f(float x);

long double exp2l(long double x);

extended exp2w(extended x);

quad exp2q(quad x);

说明

exp2() 返回 2^x。

仅适用于 **Integrity** 服务器

exp2f() 是 **exp2()** 的 **float** 版本；它使用 **float** 参数，并返回 **float** 结果。

exp2l() 是 **exp2()** 的 **long double** 版本；它使用 **long double** 参数，并返回 **long double** 结果。

exp2w() 是 **exp2()** 的 **extended** 版本；它使用 **extended** 参数，并返回 **extended** 结果。

在 HP-UX 系统中， **exp2q()** 等效于 **exp2l()** 。

用法

要使用这些函数，请使用缺省的 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。

（对于 Integrity 服务器）要使用 **exp2w()** 或 **exp2q()** ，也可以使用 **-fpwidetypes** 选项进行编译。

要使用这些函数，请确保程序包含 **<math.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

有关详细信息，请参阅《HP-UX floating-point guide for HP Integrity servers》（位于以下站点：**http://www.hp.com/go/fp** ）。

返回值

exp2(±0**)** 返回 1。

如果 *x* 为 +INFINITY，则 **exp2()** 返回 +INFINITY。

如果 *x* 为 –INFINITY，则 **exp2()** 返回零。

如果 *x* 为 NaN，则 **exp2()** 返回 NaN。

exp2() 返回无穷大（等于 **HUGE_VAL** ）来代替太大的值，并引发溢出和不精确异常。

只要结果很小（实质上是非正常表示的或为零）， **exp2()** 就引发下溢和不精确异常，因而会有精度损失，并且可

能引发这些异常（如果结果很小很小）。

只要舍入的结果不等于精确结果， **exp2()** 就引发不精确异常。

错误

仅适用于 **Integrity** 服务器

如果正确的值导致溢出，则 **exp2()** 将 **errno** 设置为 [ERANGE]。

缺省情况下，Integrity 服务器上的 HP-UX **libm** 函数不设置 **errno** 。对于 **errno** 设置，可以使用 **+Olibmerrno** 选项进行编译。

另请参阅

cbrt(3M)、 exp(3M)、 exp10(3M)、 expm1(3M)、 log2(3M)、 pow(3M)、 sqrt(3M)、 math(5)。

符合的标准

exp2() 、 **exp2f()** 和 **exp2l()** ： ISO/IEC C99（包括附件 F“IEC 60559 floating-point arithmetic”）

名称

exp( )、expf( )、expl( )、expw( )、expq( ) - 指数函数

概要

```
#include <math.h>

double exp(double x);

float expf(float x);
```

仅适用于 **HP Integrity** 服务器

```
long double expl(long double x);

extended expw(extended x);

quad expq(quad x);
```

说明

exp() 返回 e^x。

expf() 是 **float** 版的 **exp()**；它采用 **float** 参数，并返回 **float** 结果。

仅适用于 **Integrity** 服务器

expl() 是 **long double** 版的 **exp()**；它采用 **long double** 参数，并返回 **long double** 结果。

expw() 是 **extended** 版的 **exp()**；它采用 **extended** 参数，并返回 **extended** 结果。

expq() 等效于 HP-UX 系统上的 **expl()**。

用法

要使用这些函数，请使用缺省的 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。

（对于 Integrity 服务器）要使用 **expw()** 或 **expq()**，也可以使用 **-fpwidetypes** 选项进行编译。

要使用上面的任何函数，请确保程序包含 **<math.h>**，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

有关详细信息，请参阅《HP-UX floating-point guide for HP Integrity servers》（位于以下站点：**http://www.hp.com/go/fp**）。

仅适用于 **PA-RISC** 系统

可以使用 Millicode 版的 **exp()** 函数。数学库函数的 Millicode 版通常比标准库中其对应的函数运行速度快。要使用这些版本，请使用 **+Olibcalls** 或 **+Oaggressive** 优化选项来编译程序。

特殊情况下，Millicode 版本的返回值与标准库中对应函数返回的值相同（请参阅“返回值”一节），但不设置 **errno**。

返回值

exp(±0**)** 将返回 1。

如果 *x* 为正无穷大， **exp()** 将返回正无穷大。

如果 *x* 为负无穷大， **exp()** 将返回 +0。

如果 *x* 为 NaN， **exp()** 将返回 NaN。

exp() 将返回无穷大（等于 **HUGE_VAL** ），以代替量值太大的值，并引发溢出及不精确异常。

当结果太小（本质上异常或为零），因此会丧失准确性时， **exp()** 将引发下溢和不精确异常；如果结果只是太小，即可能引发这些异常。

如果未引发其他异常，则说明未指定 **exp()** 是否引发不精确异常的情况。

错误

如果正确的值会溢出， **exp()** 将 **errno** 设置为 [ERANGE]。

仅适用于 **Integrity** 服务器

缺省情况下，Integrity 服务器上的 HP-UX **libm** 函数不设置 **errno** 。对于 **errno** 设置，可以使用 **+Olibmerrno** 选项进行编译。

另请参阅

cbrt(3M)、 cexp(3M)、 exp10(3M)、 exp2(3M)、 expm1(3M)、 log(3M)、 pow(3M)、 sqrt(3M)、 math(5)。

符合的标准

exp() ： SVID3、XPG4.2、ANSI C、ISO/IEC C99 （包括附件 F“IEC 60559 floating-point arithmetic”）

expf() 、 **expl()** ： ISO/IEC C99（包括附件 F“IEC 60559 floating-point arithmetic”）

名称

expm1()、expm1f()、expm1l()、expm1w()、expm1q() - 指数减 1 的函数

概要

```
#include <math.h>

double expm1(double x);
```

仅适用于 **HP Integrity** 服务器

```
float expm1f(float x);

long double expm1l(long double x);

extended expm1w(extended x);

quad expm1q(quad x);
```

说明

expm1() 函数等效于 **exp(***x***)** - 1，但对于非常小的 *x* 值可能会更为精确。

在 *x* 值非常小的情况下， **expm1()** 和 **log1p()** 函数可用于确保 (((1+*x*)**n)-1)/*x* 的财务运算准确无误，即：

expm1(n * **log1p**(*x*))/*x*

对于小规模的日利率计算，上例可能较为适用。

另请参阅 **annuity()** 。

仅适用于 **Integrity** 服务器

expm1f() 是 **float** 版的 **expm1()** ； 它采用 **float** 参数，并返回 **float** 结果。

expm1l() 是 **long double** 版的 **expm1()** ； 它采用 **long double** 参数，并返回 **long double** 结果。

expm1w() 是 **extended** 版的 **expm1()** ； 它采用 **extended** 参数，并返回 **extended** 结果。

expm1q() 等效于 HP-UX 系统上的 **expm1l()** 。

用法

要使用这些函数，请使用缺省的 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。

（对于 Integrity 服务器）要使用 **expm1w()** 或 **expm1q()** ，也可以使用 **-fpwidetypes** 选项来编译。

确保程序包含 **<math.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

有关详细信息，请参阅《HP-UX floating-point guide for HP Integrity servers》（位于以下站点：**http://www.hp.com/go/fp** ）。

返回值

expm1(±0**)** 将返回 ±0。

如果 *x* 为正无穷大， **expm1()** 将返回正无穷大。

如果 *x* 为负无穷大， **expm1()** 将返回 1.0。

如果 *x* 为 NaN， **expm1()** 将返回 NaN。

expm1() 将返回无穷大（等于 **HUGE_VAL** ），以代替量值太大的值，并引发溢出及不精确异常。

如果未引发其他异常，则说明未指定 **expm1()** 是否引发不精确异常的情况。

错误

如果正确的值会溢出， **expm1()** 将 **errno** 设置为 [ERANGE]。

仅适用于 **Integrity** 服务器

缺省情况下，Integrity 服务器上的 HP-UX **libm** 函数不设置 **errno** 。对于 **errno** 设置，可以使用 **+Olibmerrno** 选项进行编译。

另请参阅

annuity(3M)、 compound(3M)、 exp(3M)、 log1p(3M)、 math(5)。

符合的标准

expm1() ： XPG4.2、ISO/IEC C99 （包括附件 F“IEC 60559 floating-point arithmetic”）

expm1f() 、 **expm1l()** ： ISO/IEC C99（包括附件 F“IEC 60559 floating-point arithmetic”）

名称

fabs( )、fabsf( )、fabsl( )、fabsw( )、fabsq( ) - 绝对值函数

概要

#include <math.h>

double fabs(double x);

float fabsf(float x);

仅适用于 **HP Integrity** 服务器

long double fabsl(long double x);

extended fabsw(extended x);

quad fabsq(quad x);

说明

fabs() 函数返回 *x* 的绝对值 |*x*| 。

fabsf() 是 **float** 形式的 **fabs()** ；它采用 **float** 参数，并返回 **float** 结果。

仅适用于 **Integrity** 服务器

fabsl() 是 **long double** 形式的 **fabs()** ；它采用 **long double** 参数，并返回 **long double** 结果。

fabsw() 是 **extended** 形式的 **fabs()** ；它采用 **extended** 参数，并返回 **extended** 结果。

fabsq() 等效于 HP-UX 系统上的 **fabsl()** 。

用法

要使用 **fabsf()** ，请使用缺省的 **-Ae** 选项或 **-Aa** 选项进行编译。对于 PA-RISC （仅适用于此系统），**-D_HPUX_SOURCE** 选项必须与 **-Aa** 选项一起使用。

（对于 Integrity 服务器）要使用 **fabsl()** 、 **fabsw()** 或 **fabsq()** ，请使用缺省的 **-Ae** 选项或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。要使用 **fabsw()** 或 **fabsq()** ，请使用 **-fpwidetypes** 选项进行编译。

要使用这些函数，请确保程序包含 **<math.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

有关详细信息，请参阅《HP-UX floating-point guide for HP Integrity servers》（位于以下站点：**http://www.hp.com/go/fp** ）。

返回值

如果 *x* 为 ±INFINITY，则 **fabs()** 将返回正无穷大。

如果 *x* 为 NaN，则 **fabs()** 将返回 NaN。

这些函数不会引发异常。

错误

没有定义任何错误。

另请参阅

abs(3C)、 cabs(3M)、 ceil(3M)、 floor(3M)、 fmod(3M)、 rint(3M)、 math(5)。

符合的标准

fabs() ： SVID3、XPG4.2、ANSI C、ISO/IEC C99（包括附件 F“IEC 60559 floating-point arithmetic”）

fabsf() 、 **fabsl()** ： ISO/IEC C99（包括附件 F“IEC 60559 floating-point arithmetic”）

名称

fattach( ) - 将 STREAMS 文件描述符附加到文件系统命名空间中的对象

概要

#include <stropts.h>

int fattach(int fd, const char *path);

说明

fattach() 函数可将 *fd* 文件描述符附加到 *path* 指定的文件系统命名空间中的某个对象。 *fd* 可将打开文件描述符指定给某个 STREAMS 设备或基于 STREAMS 的管道。 *path* 指定文件系统中某个现有的对象的路径名。可将 STREAMS 设备或管道附加到文件系统命名空间中的多个节点。也就是说，允许 STREAMS 设备或管道具有多个关联的名称。 STREAMS 设备或管道从节点分离（使用 *fdetach*(3C) 或 *fdetach*(1M) ）之前，针对 *path* 执行的所有操作都将作用于 STREAMS 设备或管道，而不是文件系统对象 *path* 。

根据以下方案设置已执行了 **fattach** 的流的属性（请参阅 *stat*(2) 参考页）：

- 组 ID、用户 ID、时间和权限都设置为 *path* 的对应项。
- 大小及设备编号都设置为 *fd* 参数指定的 STREAMS 设备或管道的对应项。请注意，虽然执行了 **fattach** 的 STREAMS 设备或管道的属性可能发生更改（请参阅 *chmod*(2) 参考页），但是不更改基础文件系统对象 *path* 的属性。
- 链接数设置为 1。

返回值

如果成功完成， **fattach()** 函数将返回值 0（零）。否则，将返回值 -1，同时设置 **errno** 以指示错误。

错误

如果出现以下任一情况， **fattach()** 函数会将 **errno** 设置为与该情况对应的值。

[EACCES]	虽然用户是 *path* 的所有者，但是用户对其没有写入权限。
[EBADF]	*fd* 参数是无效的文件描述符。
[EBUSY]	已挂接了 *path* 参数指定的现有对象，或者有一个文件描述符附加到了该对象。
[EFAULT]	*path* 参数指向进程的已分配地址空间以外的位置。
[EINVAL]	*fd* 参数不引用 STREAMS 设备或基于 STREAMS 的管道。
[ELOOP]	转换了 *path* 后，找到了过多的符号链接。
[ENOENT]	*path* 不存在。
[ENOTDIR]	*path* 参数的目录部分不存在。
[ENAMETOOLONG]	当 **_POSIX_NO_TRUNC** 有效时，路径名组成部分的大小超过 **NAME_MAX** ；或者路径名的长度超过 **PATH_MAX** 。

[EPERM] 当前有效的用户 ID 不是 *path* 参数指定的现有对象的所有者。

另请参阅

fdetach(3C)、 isastream(3C)、 chmod(2)、 stat(2)、 fdetach(1M)、 streamio(7)。

符合的标准

fattach()：SVID3

名称

fclose()、 fflush()、 fclose_unlocked() 和 fflush_unlocked() - 关闭或刷新流

概要

#include <stdio.h>

int fclose(FILE *stream);

int fflush(FILE *stream);

过时的接口

int fclose_unlocked(FILE *stream);

int fflush_unlocked(FILE *stream);

说明

fclose() 导致指定的 *stream* 的任何缓冲数据写出，而 *stream* 关闭。由标准的输入（或输出）系统分配的缓冲可能被释放出来。

调用 *exit*(2) 时，针对所有打开的文件自动执行 **fclose()** 。

如果 *stream* 指向的最近一个操作是输出的输出流或更新流， **fflush()** 将导致 *stream* 的任一缓冲数据被写入此文件；否则将忽略任何缓冲数据。 *stream* 保持打开状态。

如果 *stream* 是一个空指针， **fflush()** 将对当前打开的所有流执行这一刷新操作。

过时的接口

fclose_unlocked() 和 **fflush_unlocked()** 关闭或刷新流。

返回值

成功完成后， **fclose()** 和 **fflush()** 返回 0。否则，返回 EOF，并设置 **errno** 以指示出错。

错误

如果 **fclose()** 、 **fflush()** 、 **fclose_unlocked()** 或 **fflush_unlocked()** 失败， **errno** 将设置为如下状态之一:

[EAGAIN] 为文件描述符的基础 *stream* 设置了 **O_NONBLOCK** 标记，并且将在写入操作中延迟进程。

[EBADF] 文件描述符基础 *stream* 无效。

[EFBIG] 尝试对超出进程文件大小限制或最大文件大小（请参阅 *ulimit*(2) ）的文件进行写入。

[EINTR] **fclose()** 或 **fflush()** 被信号中断。

[EIO] 该进程位于后台进程组内，正尝试写入到其控制终端，设置了 **TOSTOP** ，此进程不会忽略或阻塞 **SIGTTOU** 信号，但进程的进程组是孤立的。

[ENOSPC] 包含文件的设备上没有任何可用空间。

[EPIPE] 尝试写入因未打开而无法供任何进程读取的管道。此外还向进程发送了 **SIGPIPE** 信号。

其他的 **errno** 值可能由基础 **write()** 、 **lseek()** 和 **close()** 函数设置（请参阅 *write*(2) 、 *lseek*(2) 和 *close*(2) ）。

警告

fclose_unlocked() 和 **fflush_unlocked()** 是过时的接口，支持它们的目的是为了实现与现有 DCE 应用程序的兼容。新的多线程应用程序应使用 **fclose()** 和 **fflush()** 。

另请参阅

close(2)、 exit(2)、 lseek(2)、 write(2)、 flockfile(3S)、 fopen(3S)、 setbuf(3S)、 thread_safety(5)。

符合的标准

fclose()：AES、SVID2、SVID3、XPG2、XPG3、XPG4、FIPS 151-2、POSIX.1 和 ANSI C

fflush()：AES、SVID2、SVID3、XPG2、XPG3、XPG4、FIPS 151-2、POSIX.1 和 ANSI C

名称

fdetach( ) - 从 STREAMS 文件描述符分离名称

概要

#include <stropts.h>

int fdetach(const char *path);

说明

fdetach() 函数从 *path* 指定的文件系统中的名称中分离出文件描述符。 *path* 指定之前附加的文件系统命名空间中现有对象的路径名（请参阅 *fattach*(3C) ）。 **fdetach()** 操作的结果是，节点的状态和权限返回到将此文件附加到此节点之前的状态。对 *path* 进行的任何后续操作都将只影响文件系统节点，而不会影响所附加的文件。

安全性限制

fdetach() 函数仅限超级用户，拥有 **OWNER** 权限的用户，或那些拥有 *path* 和写入权限的用户使用。除了拥有 **OWNER** 权限外，还可能需要 **DACREAD** 或 **DACWRITE** 权限。有关授权访问支持精细划分权限的系统的详细信息，请参阅 *privileges*(5) 。

返回值

成功完成之后， **fdetach()** 函数返回值零。否则，它返回一个被返回的值 -1，并设置 **errno** 以指明错误。

错误

如果出现下列任意情况， **fdetach()** 函数将 **errno** 设置为与此条件相对应的值。

[EFAULT]	*path* 参数指向分配给此过程的地址空间外部。
[EACCES]	用户没有访问此文件的正确权限。请参阅“安全性限制”一节。
[EINVAL]	*path* 参数所指向的对象没有被附加到 STREAMS 设备或管道。
[ELOOP]	在转换 *path* 时，发现过多的符号链接。
[ENOENT]	*path* 参数指向一个并不存在的路径名。
[ENOTDIR]	*path* 参数的目录部分不存在。
[ENAMETOOLONG]	当 **_POSIX_NO_TRUNC** 有效时，路径名组成部分的大小大于 **NAME_MAX** ，或者路径名长度大于 **PATH_MAX** 。
[EPERM]	当前有效的用户 ID 不是由 *path* 参数所指定现有对象的所有者，或者当前有效用户 ID 未指定拥有正确权限的用户。

另请参阅

fdetach(1M)、 umount(2)、 fattach(3C)、 isastream(3C)、 privileges(5)、 streamio(7)。

符合的标准

fdetach()：SVID3

名称

fdim( )、fdimf( )、fdiml( )、fdimw( )、fdimq( ) - 正差函数

概要

#include <math.h>

double fdim(double x, double y);

仅适用于 **HP Integrity** 服务器

float fdimf(float x, float y);

long double fdiml(long double x, long double y);

extended fdimw(extended x, extended y);

quad fdimq(quad x, quad y);

说明

fdim() 函数可确定其参数间的正差数。

仅适用于 **Integrity** 服务器

fdimf() 是 **float** 版的 **fdim()**；它采用 **float** 参数，并返回 **float** 结果。

fdiml() 是 **long double** 版的 **fdim()**；它采用 **long double** 参数，并返回 **long double** 结果。

fdimw() 是 **extended** 版的 **fdim()**；它采用 **extended** 参数，并返回 **extended** 结果。

fdimq() 等效于 HP-UX 系统上的 **fdiml()**。

用法

要使用此函数，请使用缺省的 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。（对于 Integrity 服务器）要使用 **fdimw()** 或 **fdimq()**，请使用 **-fpwidetypes** 选项进行编译。确保程序包含 **<math.h>**，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

返回值

fdim() 函数将返回 *x* 与 *y* 之间的正差数。

如果 *x* **>** *y*， **fdim()** 就会返回 *x* **-** *y*（并引发因减法而导致的某些异常）。

如果 *x* **<=** *y*， **fdim()** 将返回 +0。

如果 *x* 或 *y* 为 NaN， **fdim()** 将返回 NaN。

如果两个参数都为 NaN， **fdim()** 就会返回 NaN。

fdim() 将返回正无穷大（代替量值太大的值），并引发溢出及不精确异常。

错误

没有定义任何错误。

另请参阅

fdim(3M)、 fmin(3M)、 math(5)。

符合的标准

fdim() 、 **fdimf()** 、 **fdiml()** ： ISO/IEC C99（包括附件 F“IEC 60559 floating-point arithmetic”）

名称

feclearexcept( ) - 清除浮点异常标记

概要

#include <fenv.h>

仅适用于 **HP Integrity** 服务器

int feclearexcept(int excepts);

仅适用于 **PA-RISC** 系统

void feclearexcept(int excepts);

说明

feclearexcept() 函数清除其参数表示的异常标记。此参数可以构造为异常宏的的按位 OR： **FE_INEXACT** 、 **FE_DIVBYZERO** 、 **FE_UNDERFLOW** 、 **FE_OVERFLOW** 和 **FE_INVALID** 。 **FE_ALL_EXCEPT** 表示所有的浮点异常。

用法

要使用此函数，请使用缺省的 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。确保程序包含 **<fenv.h>**。

对于 Integrity 服务器，请在编译程序命令行上指定 **+Ofenvaccess** ，或者将对此函数的调用放到确定的 **FENV_ACCESS** pragma 的有效范围之内：

#pragma STDC FENV_ACCESS ON

如果 **FENV_ACCESS** pragma 放在文件中的任何顶级声明之外，则此 pragma 将应用于其后的编译中的所有声明，直到遇到另一个 **FENV_ACCESS** pragma 或达到文件的末尾为止。

如果 **FENV_ACCESS** pragma 放在块（复合语句）的开头，则此 pragma 将应用，直到遇到另一个 **FENV_ACCESS** pragma 或到达块的末尾为止。

对于 PA-RISC，可能需要使用 +Onomoveflops 编译程序选项，以阻止可能终结此函数的指定行为的优化。

通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

有关详细信息，请参阅《HP-UX Floating-Point Guide》。

返回值

仅适用于 **Integrity** 服务器

此函数始终返回 0，表明所有指定的异常已清除。

仅适用于 **PA-RISC** 系统

无。

错误

没有定义任何错误。

举例

清除下溢和不精确的浮点异常标记:

```
#include <fenv.h>
/*...*/
  feclearexcept(FE_UNDERFLOW | FE_INEXACT);
```

清除所有浮点异常标记:

```
#include <fenv.h>
/*...*/
  feclearexcept(FE_ALL_EXCEPT);
```

另请参阅

fegetexceptflag(3M)、 fegettrapenable(3M)、 feraiseexcept(3M)、 fesetexceptflag(3M)、 fesettrapenable(3M)、 fetestexcept(3M)、 fenv(5)。

符合的标准

feclearexcept()： ISO/IEC C99（包括附件 F“IEC 60559 floating-point arithmetic”）

名称

fegetenv( ) -获取浮点环境

概要

#include <fenv.h>

仅适用于 **HP Integrity** 服务器

int fegetenv(fenv_t *envp);

仅适用于 **PA-RISC** 系统

void fegetenv(fenv_t *envp);

说明

fegetenv() 函数在参数 *envp* 指向的对象中存储当前的浮点环境。

要使用此函数，请使用缺省 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项编译。

确保程序包含 **<fenv.h>** 。

对于 Integrity 服务器，请在编译程序命令行上指定 **+Ofenvaccess** ，或者将对此函数的调用放到确定的 **FENV_ACCESS** pragma 的有效范围之内：

#pragma STDC FENV_ACCESS ON

如果 **FENV_ACCESS** pragma 放在文件中的任何顶级声明之外，则此 pragma 将应用于其后的编译中的所有声明，直到遇到另一个 **FENV_ACCESS** pragma 或达到文件的末尾为止。

如果 **FENV_ACCESS** pragma 放在块（复合语句）的开头，则此 pragma 将应用，直到遇到另一个 **FENV_ACCESS** pragma 或到达块的末尾为止。

对于 PA-RISC，可能需要使用 +Onomoveflops 编译程序选项，以阻止可能破坏此函数的指定行为的优化。

通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

有关详细信息，请参阅《HP-UX floating-point guide for HP Integrity servers》（位于以下站点：**http://www.hp.com/go/fp** ）。

返回值

仅适用于 **Integrity** 服务器

此函数始终返回 0，这表示当前环境的表示形式已成功存储。

仅适用于 **PA-RISC** 系统

无。

错误

没有定义任何错误。

举例

存储当前的浮点环境：

```
#include <fenv.h>
/*...*/
  fenv_t env;
  /*...*/
  fegetenv(&env);
```

另请参阅

feholdexcept(3M)、 fesetenv(3M)、 feupdateenv(3M)、 fenv(5)。

符合的标准

fegetenv()： ISO/IEC C99（包括附件 F “IEC 60559 floating-point arithmetic”）

名称

fegetexceptflag( ) - 获取浮点异常标记

概要

#include <fenv.h>

仅适用于 **HP Integrity** 服务器

int fegetexceptflag(fexcept_t *flagp, int excepts);

仅适用于 **PA-RISC** 系统

void fegetexceptflag(fexcept_t *flagp, int excepts);

说明

对于由参数 *flagp* 所指向的对象中的参数 *excepts* 所指示的浮点异常标记， **fegetexceptflag()** 函数将存储其状态表示。 *excepts* 参数可以通过浮点异常宏的按位或运算而构成： **FE_INEXACT** 、 **FE_DIVBYZERO** 、 **FE_UNDERFLOW** 、 **FE_OVERFLOW** 及 **FE_INVALID** 。 **FE_ALL_EXCEPT** 代表所有浮点异常。可以将存储浮点异常标记的对象传给 **fesetexceptflag** ，用于恢复系统异常标记，但对象中所存储的浮点异常标记的表示形式却不可预测。

使用 **fetestexcept()** （而非 **fegetexceptflag()** ）可以确定所设置的浮点异常标记。

用法

要使用此函数，请使用缺省的 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。确保程序包含 **<fenv.h>**。

对于 Integrity 服务器，请在编译程序命令行上指定 **+Ofenvaccess** ，或者将对此函数的调用放到确定的 **FENV_ACCESS** pragma 的有效范围之内：

#pragma STDC FENV_ACCESS ON

如果 **FENV_ACCESS** pragma 放在文件中的任何顶级声明之外，则此 pragma 将应用于其后的编译中的所有声明，直到遇到另一个 **FENV_ACCESS** pragma 或达到文件的末尾为止。

如果 **FENV_ACCESS** pragma 放在块（复合语句）的开头，则此 pragma 将应用，直到遇到另一个 **FENV_ACCESS** pragma 或到达块的末尾为止。

对于 PA-RISC，可能需要使用 **+Onomoveflops** 编译程序选项，以便防止会破坏该函数指定行为的优化操作。

通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

有关详细信息，请参阅《HP-UX floating-point guide for HP Integrity servers》（位于以下站点： **http://www.hp.com/go/fp** ）。

返回值

仅适用于 **Integrity** 服务器

此函数将始终返回 0，指示已成功存储一种表示形式。

仅适用于 **PA-RISC** 系统

无。

错误

没有定义任何错误。

举例

存储溢出异常标记及无效的浮点异常标记：

```
#include <fenv.h>
/*...*/
  fexcept_t flags;
  /*...*/
  fegetexceptflag(&flags, FE_OVERFLOW | FE_INVALID);
```

另请参阅

feclearexcept(3M)、 fegettrapenable(3M)、 feraiseexcept(3M)、 fesetexceptflag(3M)、 fesettrapenable(3M)、 fetestexcept(3M)、 fenv(5)。

符合的标准

fegetexceptflag()： ISO/IEC C99（包括附件 F“IEC 60559 floating-point arithmetic”）

名称

fegetflushtozero( ) - 获取浮点下溢模式

概要

#include <fenv.h>

int fegetflushtozero(void);

说明

fegetflushtozero() 函数检索表示当前下溢模式的值，当前下溢模式要么是符合 IEEE-754 的（渐进的）下溢模式，要么是“下溢到零”模式。

缺省的下溢模式符合 IEEE-754。

“下溢到零”模式也称为快速下溢模式，大多数 PA1.1 系统以及所有的 PA2.0 和基于 Itnaium(R) 的系统都支持此模式。在符合 IEEE-754 的模式下，可以由内核来承担处理下溢情况的责任，其中，通过软件模拟完成 IEEE 要求的将结果转换为非规范值或零的工作。在 PA-RISC 系统中，“下溢到零”模式允许替换非规范操作数和操作结果的零，而无需由内核来承担。在 HP Integrity 服务器中，“下溢到零”模式使得可以替换非规范结果（而非操作数）的零，而无需由内核来承担。“下溢到零”模式可以为一些应用程序提供重要的性能改进。

用法

要使用此函数，请使用缺省的 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。确保程序包含 **<fenv.h>**。

对于 Integrity 服务器，请在编译程序命令行上指定 **+Ofenvaccess** ，或者将对此函数的调用放到确定的 **FENV_ACCESS** pragma 的有效范围之内：

#pragma STDC FENV_ACCESS ON

如果 **FENV_ACCESS** pragma 放在文件中的任何顶级声明之外，则此 pragma 将应用于其后的编译中的所有声明，直到遇到另一个 **FENV_ACCESS** pragma 或达到文件的末尾为止。

如果 **FENV_ACCESS** pragma 放在块（复合语句）的开头，则此 pragma 将应用，直到遇到另一个 **FENV_ACCESS** pragma 或到达块的末尾为止。

对于 PA-RISC，可能需要使用 **+Onomoveflops** 编译程序选项，以阻止可能破坏此函数的指定行为的优化。

通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

有关详细信息，请参阅《HP-UX floating-point guide for HP Integrity servers》（位于以下站点：**http://www.hp.com/go/fp** ）。

返回值

如果当前下溢模式为符合 IEEE-754 的模式，则 **fegetflushtozero()** 函数返回零。如果当前的下溢模式为“下溢到零”模式，则此函数返回 1。

在不支持“下溢到零”模式的系统中，此函数返回一个未定义的值。

错误

没有定义任何错误。

举例

保存当前下溢模式，设置“下溢到零”模式，并恢复以前的模式。

```
#include <fenv.h>
/*...*/
  int fm_saved;

  fm_saved = fegetflushtozero();
  fesetflushtozero(1);
  /*...*/
  fesetflushtozero(fm_saved);
```

作者

fegetflushtozero() 由 HP 开发，目前的任何标准都未做要求。

另请参阅

fesetflushtozero(3M)、 fenv(5)。

名称

fegetround( ) - 获取浮点舍入方向模式

概要

#include <fenv.h>

int fegetround(void);

说明

fegetround() 函数将获取当前的舍入方向（由 IEEE 754 (IEC 60559) 浮点标准指定）。

缺省舍入方向模式被舍入到最接近的 (**FE_TONEAREST**)。

用法

要使用此函数，请使用缺省的 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项来编译。确保程序包含 **<fenv.h>** 。

对于 HP Integrity 服务器，请在编译程序命令行上指定 **+Ofenvaccess** ，或者将对此函数的调用放到确定的 **FENV_ACCESS** pragma 的有效范围之内：

#pragma STDC FENV_ACCESS ON

如果 **FENV_ACCESS** pragma 放在文件中的任何顶级声明之外，则此 pragma 将应用于其后的编译中的所有声明，直到遇到另一个 **FENV_ACCESS** pragma 或达到文件的末尾为止。

如果 **FENV_ACCESS** pragma 放在块（复合语句）的开头，则此 pragma 将应用，直到遇到另一个 **FENV_ACCESS** pragma 或到达块的末尾为止。

对于 PA-RISC，可能需要使用 **+Onomoveflops** 编译程序选项，以防止会破坏该函数指定行为的优化操作。

通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

有关详细信息，请参阅《HP-UX floating-point guide for HP Integrity servers》（位于以下站点：**http://www.hp.com/go/fp** ）。

返回值

fegetround() 函数将返回表示当前舍入方向的舍入方向宏的值。返回值将与下列的宏之一匹配（这些宏是在 **<fenv.h>** 中定义的）：

FE_TONEAREST 向最接近的值舍入。如果两个最接近的表示值与原值的接近程度相等，则取有效的零位最少的那个值。

FE_UPWARD 向上舍入（向正无穷大舍入）。

FE_DOWNWARD 向下舍入（向负无穷大舍入）。

FE_TOWARDZERO 向零舍入。

错误

没有定义任何错误。

举例

保存、设置和恢复舍入方向。

```
#include <fenv.h>
/*...*/
{
  int save_round;

  save_round = fegetround();
  fesetround(FE_UPWARD);
  /*...*/
  fesetround(save_round);
  /*...*/
}
```

另请参阅

fesetround(3M)、 fenv(5)。

符合的标准

fegetround() ： ISO/IEC C99（包括附件 F“IEC 60559 floating-point arithmetic”）

名称

fegettrapenable( ) - 启用浮点异常陷阱

概要

#include <fenv.h>

int fegettrapenable(void);

说明

fegettrapenable() 函数确定当前启用了哪些浮点异常陷阱。

用法

要使用此函数，请使用缺省的 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项编译。确保程序包含 **<fenv.h>** 。

对于 HP Integrity 服务器，请在编译程序命令行上指定 **+Ofenvaccess** ，或者将对此函数的调用放到确定的 **FENV_ACCESS** pragma 的有效范围之内：

#pragma STDC FENV_ACCESS ON

如果 **FENV_ACCESS** pragma 放在文件中的任何顶级声明之外，则此 pragma 将应用于其后的编译中的所有声明，直到遇到另一个 **FENV_ACCESS** pragma 或达到文件的末尾为止。

如果 **FENV_ACCESS** pragma 放在块（复合语句）的开头，则此 pragma 将应用，直到遇到另一个 **FENV_ACCESS** pragma 或到达块的末尾为止。

对于 PA-RISC，可能需要使用 **+Onomoveflops** 编译程序选项，以阻止可能破坏此函数的指定行为的优化。

通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

有关详细信息，请参阅《HP-UX floating-point guide for HP Integrity servers》（位于以下站点：**http://www.hp.com/go/fp** ）。

返回值

fegettrapenable() 函数返回对应于当前启用的异常陷阱的浮点异常宏的按位 OR。这些宏为 **FE_INEXACT** 、 **FE_DIVBYZERO** 、 **FE_UNDERFLOW** 、 **FE_OVERFLOW** 和 **FE_INVALID** 。 **FE_ALL_EXCEPT** 表示所有浮点异常。

错误

没有定义任何错误。

举例

检索当前的陷阱设置并确定是否启用除数为零异常的陷阱。

```
#include <fenv.h>
/*...*/
  if (fegettrapenable() & FE_DIVBYZERO)
    printf("divide by zero trap set\\n");
```

作者

fegettrapenable() 由 HP 开发，目前的任何标准都未做要求。

另请参阅

feclearexcept(3M)、 fegetexceptflag(3M)、 feraiseexcept(3M)、 fesetexceptflag(3M)、 fesettrapenable(3M)、 fenv(5)。

名称

feholdexcept( ) - 保存浮点环境

概要

#include <fenv.h>

int feholdexcept(fenv_t *envp);

说明

feholdexcept() 函数将保存参数 *envp* 所指向的对象中的当前浮点环境。此函数还将清除浮点异常标记并禁用所有陷阱。

将 **feholdexcept()** 与 **feupdateenv()** 配合使用可以隐藏假的浮点异常。将它与 **fesetenv()** 配合使用则可隐藏所有浮点异常。

用法

要使用此函数，请使用缺省的 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。确保程序包含 **<fenv.h>**。

对于 HP Integrity 服务器，请在编译程序命令行上指定 **+Ofenvaccess** ，或者将对此函数的调用放到确定的 **FENV_ACCESS** pragma 的有效范围之内：

#pragma STDC FENV_ACCESS ON

如果 **FENV_ACCESS** pragma 放在文件中的任何顶级声明之外，则此 pragma 将应用于其后的编译中的所有声明，直到遇到另一个 **FENV_ACCESS** pragma 或达到文件的末尾为止。

如果 **FENV_ACCESS** pragma 放在块（复合语句）的开头，则此 pragma 将应用，直到遇到另一个 **FENV_ACCESS** pragma 或到达块的末尾为止。

对于 PA-RISC，可能需要使用 **+Onomoveflops** 编译程序选项，以阻止可能破坏此函数的指定行为的优化。

通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

有关详细信息，请参阅《HP-UX floating-point guide for HP Integrity servers》（位于以下站点：**http://www.hp.com/go/fp** ）。

返回值

feholdexcept() 函数将返回零值，指示已成功禁用任何陷阱。

错误

没有定义任何错误。

举例

在 **holdenv** 中存储当前浮点环境，隐藏假的下溢异常，继续任何在遇到对 **feupdateenv()** 的调用前所发生的浮点异常。

```
#include <fenv.h>
/*...*/
  fenv_t holdenv;

  feholdexcept(&holdenv);
  /* perform operations */
  if (/* test for spurious underflow */)
    feclearexcept(FE_UNDERFLOW);
  feupdateenv(&holdenv); /* raise accumulated exceptions */
```

另请参阅

fegetenv(3M)、 fesetenv(3M)、 feupdateenv(3M)、 fenv(5)。

符合的标准

feholdexcept()： ISO/IEC C99（包括附件 F“IEC 60559 floating-point arithmetic”）

名称

feraiseexcept( ) - 引发浮点异常

概要

#include <fenv.h>

仅适用于 **HP Integrity** 服务器

int feraiseexcept(int excepts);

仅适用于 **PA-RISC** 系统

void feraiseexcept(int excepts);

说明

feraiseexcept() 函数引发其参数表示的浮点异常。此参数可以构造为异常宏的按位 OR，这些异常宏包括：**FE_INEXACT** 、 **FE_DIVBYZERO** 、 **FE_UNDERFLOW** 、 **FE_OVERFLOW** 和 **FE_INVALID** 。**FE_ALL_EXCEPT** 表示所有异常。

将采用为指定异常启用的任何陷阱。

用法

要使用此函数，请使用缺省 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。确保程序包含 **<fenv.h>** 。

对于 Integrity 服务器，请在编译程序命令行上指定 **+Ofenvaccess** ，或者将对此函数的调用放到确定的 **FENV_ACCESS** pragma 的有效范围之内：

#pragma STDC FENV_ACCESS ON

如果 **FENV_ACCESS** pragma 放在文件中的任何顶级声明之外，则此 pragma 将应用于其后的编译中的所有声明，直到遇到另一个 **FENV_ACCESS** pragma 或达到文件的末尾为止。

如果 **FENV_ACCESS** pragma 放在块（复合语句）的开头，则此 pragma 将应用，直到遇到另一个 **FENV_ACCESS** pragma 或到达块的末尾为止。

对于 PA-RISC，可能需要使用 **+Onomoveflops** 编译程序选项，以阻止可能破坏此函数的指定行为的优化。

通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

有关详细信息，请参阅《HP-UX floating-point guide for HP Integrity servers》（位于以下站点：**http://www.hp.com/go/fp** ）。

返回值

仅适用于 **Integrity** 服务器

此函数始终返回 0，这表示已成功引发所有指定的异常。

仅适用于 **PA-RISC** 系统

无。

错误

没有定义任何错误。

举例

引发下溢和不精确异常:

```
#include <fenv.h>
/*...*/
  feraiseexcept(FE_UNDERFLOW | FE_INEXACT);
```

引发所有异常:

```
#include <fenv.h>
/*...*/
  feraiseexcept(FE_ALL_EXCEPT);
```

另请参阅

feclearexcept(3M)、 fegetexceptflag(3M)、 fegettrapenable(3M)、 fesetexceptflag(3M)、 fesettrapenable(3M)、 fetestexcept(3M)、 fenv(5)。

符合的标准

feraiseexcept()： ISO/IEC C99（包括附件 F“IEC 60559 floating-point arithmetic”）

名称

ferror()、 feof()、 clearerr()、 ferror_unlocked()、 feof_unlocked()、 clearerr_unlocked() - 流状态查询

概要

#include <stdio.h>

int ferror(FILE *stream);

int feof(FILE *stream);

void clearerr(FILE *stream);

过时的接口

int ferror_unlocked(FILE *stream);

int feof_unlocked(FILE *stream);

void clearerr_unlocked(FILE *stream);

说明

ferror() 当以前从给定的 *stream* 读取或向其写入而出现 I/O 错误时，返回非零值，否则返回零。如果没有由 **clearerr()** 清除或者特定的 *stdio* 例行程序没有这样指示，则此错误指示持续到流关闭为止。

feof() 当以前读取给定的输入 *stream* 而检测到 EOF 时，返回非零值，否则返回零。

clearerr() 将给定 *stream* 上的错误指示符和 EOF 指示符重置为零。

过时的接口

ferror_unlocked() 、 **feof_unlocked()** 和 **clearerr_unlocked()** 流状态查询。

警告

所有这些例行程序将作为库函数和宏实现。缺省使用的宏版本在 **<stdio.h>** 中定义。要获取此库函数，要么使用 **#undef** 删除宏定义，要么在 ANSI-C 模式下编译时，将函数名放入圆括号内或使用函数地址。下面的示例演示了每一种方法：

```
#include <stdio.h>
#undef ferror
  ...
main()
{
   int (*find_error()) ();
      ...
   return_val=ferror(fd);
      ...
   return_val=(feof)(fd1);
      ...
```

```
        find_error = feof;
    };
```

重入接口

如果 **_REENTRANT** 是在包括 **<stdio.h>** 之前定义的，则缺省使用 **ferror()**、" feof() 和 **clearerr()** 的库函数的锁定版本。

ferror_unlocked() 、 **feof_unlocked()** 和 **clearerr_unlocked()** 都是过时的接口，它们都是为了实现与现有的 DCE 应用程序兼容而受支持。新的多线程应用程序应该使用 **ferror()** 、 **feof()** 和 **clearerr()** 。

另请参阅

open(2)、 flockfile(3S)、 fopen(3S)、 thread_safety(5)。

符合的标准

ferror()：AES、SVID2、SVID3、XPG2、XPG3、XPG4、FIPS 151-2、POSIX.1、ANSI C

clearerr()：AES、SVID2、SVID3、XPG2、XPG3、XPG4、FIPS 151-2、POSIX.1、ANSI C

feof()：AES、SVID2、SVID3、XPG2、XPG3、XPG4、FIPS 151-2、POSIX.1、ANSI C

名称

fesetenv( ) - 设置浮点环境

概要

#include <fenv.h>

仅适用于 **HP Integrity** 服务器

int fesetenv(const fenv_t *envp);

仅适用于 **PA-RISC** 系统

void fesetenv(const fenv_t *envp);

说明

fesetenv() 函数将建立由 *envp* 所指向的对象表示的浮点环境。参数 *envp* 必须指向由 **fegetenv()** 或 **feholdexcept()** 调用所设置的对象，或者等于宏 **FE_DFL_ENV** 。

请注意， **fesetenv()** 只是安装浮点异常标记的状态（通过它的参数来表示），而不会引发这些浮点异常（因此也就不会采用陷阱）。

用法

要使用此函数，请使用缺省的 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。确保程序包含 **<fenv.h>**。

对于 Integrity 服务器，请在编译程序命令行上指定 **+Ofenvaccess** ，或者将对此函数的调用放到确定的 **FENV_ACCESS** pragma 的有效范围之内：

```
#pragma STDC FENV_ACCESS ON
```

如果 **FENV_ACCESS** pragma 放在文件中的任何顶级声明之外，则此 pragma 将应用于其后的编译中的所有声明，直到遇到另一个 **FENV_ACCESS** pragma 或达到文件的末尾为止。

如果 **FENV_ACCESS** pragma 放在块（复合语句）的开头，则此 pragma 将应用，直到遇到另一个 **FENV_ACCESS** pragma 或到达块的末尾为止。

对于 PA-RISC，可能需要使用 **+Onomoveflops** 编译程序选项，以阻止可能破坏此函数的指定行为的优化。

通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

有关详细信息，请参阅《HP-UX floating-point guide for HP Integrity servers》（位于以下站点：**http://www.hp.com/go/fp** ）。

返回值

仅适用于 **Integrity** 服务器

此函数将始终返回 0，指示已成功建立环境。

仅适用于 **PA-RISC** 系统

无。

错误

没有定义任何错误。

举例

存储当前浮点环境，即使遇到异常也会继续，然后恢复以前环境而不引发累积的浮点异常。

```
#include <fenv.h>
/*...*/
  fenv_t holdenv;

  feholdexcept(&holdenv);
   /* perform operations */
  fesetenv(&holdenv);
```

恢复缺省环境：

```
#include <fenv.h>
/*...*/
  fesetenv(FE_DFL_ENV);
```

另请参阅

fegetenv(3M)、 feholdexcept(3M)、 feupdateenv(3M)、 fenv(5)。

符合的标准

fesetenv()： ISO/IEC C99（包括附件 F“IEC 60559 floating-point arithmetic”）

名称

fesetexceptflag( ) - 设置浮点异常标记

概要

#include <fenv.h>

仅适用于 **HP Integrity** 服务器

int fesetexceptflag(const fexcept_t *flagp, int excepts);

仅适用于 **PA-RISC** 系统

void fesetexceptflag(const fexcept_t *flagp, int excepts);

说明

fesetexceptflag() 函数将参数 *excepts* 指示的浮点异常标记的状态设置为存储在 *flagp* 指向的对象中存储的状态。**flagp* 的值必须由上一次对 **fegetexceptflag()** 的调用设置，此函数的第二个参数至少表示参数 *excepts* 表示的浮点异常；否则就未定义对浮点异常标记的影响。此函数并不引发异常，而只是设置标记的状态（因而没有采用陷阱）。*excepts* 参数构造为浮点异常宏的按位 OR：**FE_INEXACT** 、 **FE_DIVBYZERO** 、 **FE_UNDERFLOW**、**FE_OVERFLOW** 和 **FE_INVALID** 。 **FE_ALL_EXCEPT** 表示所有的浮点异常。

用法

要使用此函数，请使用缺省的 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。确保程序包含 **<fenv.h>**。

对于 Integrity 服务器，请在编译程序命令行上指定 **+Ofenvaccess** ，或者将对此函数的调用放到确定的 **FENV_ACCESS** pragma 的有效范围之内：

```
#pragma STDC FENV_ACCESS ON
```

如果 **FENV_ACCESS** pragma 放在文件中的任何顶级声明之外，则此 pragma 将应用于其后的编译中的所有声明，直到遇到另一个 **FENV_ACCESS** pragma 或达到文件的末尾为止。

如果 **FENV_ACCESS** pragma 放在块（复合语句）的开头，则此 pragma 将应用，直到遇到另一个 **FENV_ACCESS** pragma 或到达块的末尾为止。

对于 PA-RISC，可能需要使用 **+Onomoveflops** 编译程序选项，以阻止可能破坏此函数的指定行为的优化。

通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

有关详细信息，请参阅《HP-UX floating-point guide for HP Integrity servers》（位于以下站点：**http://www.hp.com/go/fp** ）。

返回值

仅适用于 **Integrity** 服务器

此函数始终返回 0，指示所有指定的标记都成功设置为适当的状态。

仅适用于 **PA-RISC** 系统

无。

错误

没有定义任何错误。

举例

使用 **fegetexceptflag()** 保存两个浮点异常标记的当前状态。以后，可以使用 **fesetexceptflag()** 来恢复保存的状态。

```
#include <fenv.h>
/*...*/
  fexcept_t saved_flags;
  /*...*/
  fegetexceptflag(&saved_flags, FE_DIVBYZERO | FE_INEXACT);
  /*...*/
  fesetexceptflag(&saved_flags, FE_DIVBYZERO | FE_INEXACT);
```

另请参阅

feclearexcept(3M)、 fegetexceptflag(3M)、 fegettrapenable(3M)、 feraiseexcept(3M)、 fesettrapenable(3M)、 fetestexcept(3M)、 fenv(5)。

符合的标准

fesetexceptflag()： ISO/IEC C99（包括附件 F“IEC 60559 floating-point arithmetic”）

名称

fesetflushtozero( ) - 设置浮点下溢模式

概要

#include <fenv.h>

void fesetflushtozero(int);

说明

fesetflushtozero() 函数将设置当前的下溢模式。如果参数为 1，则下溢模式将设置为“下溢到零”模式。如果参数为 0，则下溢模式将设置为符合 IEEE-754 的（渐进）下溢模式。对于除 1 或 0 以外的参数，其作用未定义。

缺省的下溢模式符合 IEEE-754。

大多数 PA1.1 系统和所有 PA2.0 及 HP Integrity 服务器都支持“下溢到零”模式（也称为快速下溢模式）。在符合 IEEE-754 的下溢模式下，可能导致下溢的情况可通过陷入内核的方式进行处理，其中通过 IEEE 授权的转换将结果转换为反规范化值或零的过程是由软件仿真实现的。

在 PA-RISC 系统上，“下溢到零”模式允许用零替换异常操作数和操作结果，而不会陷入内核。

在 Integrity 服务器上，“下溢到零”模式将用零替换异常结果（但不是操作数），而不会陷入内核。

“下溢到零”模式可以显著地改善某些应用程序的性能。

用法

要使用此函数，请使用缺省的 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。确保程序包含 **<fenv.h>**。

对于 Integrity 服务器，请在编译程序命令行上指定 **+Ofenvaccess** ，或者将对此函数的调用放到确定的 **FENV_ACCESS** pragma 的有效范围之内：

```
#pragma STDC FENV_ACCESS ON
```

如果 **FENV_ACCESS** pragma 放在文件中的任何顶级声明之外，则此 pragma 将应用于其后的编译中的所有声明，直到遇到另一个 **FENV_ACCESS** pragma 或达到文件的末尾为止。

如果 **FENV_ACCESS** pragma 放在块（复合语句）的开头，则此 pragma 将应用，直到遇到另一个 **FENV_ACCESS** pragma 或到达块的末尾为止。

对于 PA-RISC，可能需要使用 **+Onomoveflops** 编译程序选项，以阻止可能破坏此函数的指定行为的优化。

通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

有关详细信息，请参阅《HP-UX floating-point guide for HP Integrity servers》（位于以下站点：**http://www.hp.com/go/fp** ）。

返回值

无。

错误

没有定义任何错误。

举例

保存当前下溢模式，设置“下溢到零”模式，并恢复以前的模式。

```
#include <fenv.h>
/*...*/
  int fm_saved;

  fm_saved = fegetflushtozero();
  fesetflushtozero(1);
  /*...*/
  fesetflushtozero(fm_saved);
```

作者

fesetflushtozero() 由 HP 开发，目前的任何标准都未做要求。

另请参阅

fegetflushtozero(3M)、 fenv(5)。

名称

fesetround( ) - 设置浮点舍入方向模式

概要

#include <fenv.h>

int fesetround(int round);

说明

fesetround() 函数建立由其参数 *round* 表示的舍入方向。 *round* 参数必须等效于下面的某个宏： **FE_TONEAREST** 、 **FE_UPWARD** 、 **FE_DOWNWARD** 和 **FE_TOWARDZERO** 。如果参数不匹配舍入方向宏，则不会更改舍入方向。

缺省舍入方向模式被舍入到最接近的 (**FE_TONEAREST**) 。

用法

要使用此函数，请使用缺省 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。确保程序包含 **<fenv.h>** 。

对于 HP Integrity 服务器，请在编译程序命令行上指定 **+Ofenvaccess** ，或者将对此函数的调用放到确定的 **FENV_ACCESS** pragma 的有效范围之内：

#pragma STDC FENV_ACCESS ON

如果 **FENV_ACCESS** pragma 放在文件中的任何顶级声明之外，则此 pragma 将应用于其后的编译中的所有声明，直到遇到另一个 **FENV_ACCESS** pragma 或达到文件的末尾为止。

如果 **FENV_ACCESS** pragma 放在块（复合语句）的开头，则此 pragma 将应用，直到遇到另一个 **FENV_ACCESS** pragma 或到达块的末尾为止。

对于 PA-RISC，可能需要使用 **+Onomoveflops** 编译程序选项，以阻止可能破坏此函数的指定行为的优化。

通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

有关详细信息，请参阅《HP-UX floating-point guide for HP Integrity servers》（位于以下站点：**http://www.hp.com/go/fp** ）。

返回值

当且仅当参数等效于舍入方向宏，则 **fesetround()** 函数将返回零值。

错误

没有定义任何错误。

举例

保存、设置和恢复舍入方向。

```
#include <fenv.h>
/*...*/
{
```

```
    int save_round;

    save_round = fegetround();
    fesetround(FE_UPWARD);
    /*...*/
    fesetround(save_round);
    /*...*/
}
```

另请参阅

fegetround(3M)、 fenv(5)。

符合的标准

fesetround() ： ISO/IEC C99（包括附件 F，“IEC 60559 floating-point arithmetic”）

名称

fesettrapenable( ) - 启用异常陷阱

概要

#include <fenv.h>

void fesettrapenable(int excepts);

说明

fesettrapenable() 函数启用由参数 *excepts* 表示的异常陷阱，并禁用不是此参数表示的异常陷阱。此参数可以构造为异常宏的按位 OR： **FE_INEXACT** 、 **FE_DIVBYZERO** 、 **FE_UNDERFLOW** 、 **FE_OVERFLOW** 和 **FE_INVALID** 。 **FE_ALL_EXCEPT** 表示所有异常。

用法

要使用此函数，请使用缺省 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。确保程序包含 **<fenv.h>** 。

对于 HP Integrity 服务器，请在编译程序命令行上指定 **+Ofenvaccess** ，或者将对此函数的调用放到确定的 **FENV_ACCESS** pragma 的有效范围之内：

#pragma STDC FENV_ACCESS ON

如果 **FENV_ACCESS** pragma 放在文件中的任何顶级声明之外，则此 pragma 将应用于其后的编译中的所有声明，直到遇到另一个 **FENV_ACCESS** pragma 或达到文件的末尾为止。

如果 **FENV_ACCESS** pragma 放在块（复合语句）的开头，则此 pragma 将应用，直到遇到另一个 **FENV_ACCESS** pragma 或到达块的末尾为止。

对于 PA-RISC，可能需要使用 **+Onomoveflops** 编译程序选项，以阻止可能破坏此函数的指定行为的优化。

通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

有关详细信息，请参阅《HP-UX floating-point guide for HP Integrity servers》（位于以下站点：**http://www.hp.com/go/fp** ）。

返回值

无。

错误

没有定义任何错误。

举例

启用溢出和除数为零陷阱并禁用其他形式

```
#include <fenv.h>
/*...*/
  fesettrapenable(FE_OVERFLOW | FE_DIVBYZERO);
```

作者

fesettrapenable() 由 HP 开发，目前的任何标准都未做要求。

另请参阅

feclearexcept(3M)、 fegetexceptflag(3M)、 fegettrapenable(3M)、 feraiseexcept(3M)、 fesetexceptflag(3M)、 fetestexcept(3M)、 fenv(5)。

名称

fetestexcept( ) - 测试浮点异常

概要

#include <fenv.h>

int fetestexcept(int excepts);

说明

fetestexcept() 函数确定当前设置了指定子集的哪些浮点异常标记。 *excepts* 参数指定要查询的浮点标记。此参数可以构造为异常宏的按位 OR，这些异常宏包括： **FE_INEXACT** 、 **FE_DIVBYZERO** 、 **FE_UNDERFLOW** 、 **FE_OVERFLOW** 和 **FE_INVALID** 。 **FE_ALL_EXCEPT** 表示所有浮点异常。

用法

要使用此函数，请使用缺省 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。确保程序包含 **<fenv.h>** 。

对于 HP Integrity 服务器，请在编译程序命令行上指定 **+Ofenvaccess** ，或者将对此函数的调用放到确定的 **FENV_ACCESS** pragma 的有效范围之内：

#pragma STDC FENV_ACCESS ON

如果 **FENV_ACCESS** pragma 放在文件中的任何顶级声明之外，则此 pragma 将应用于其后的编译中的所有声明，直到遇到另一个 **FENV_ACCESS** pragma 或达到文件的末尾为止。

如果 **FENV_ACCESS** pragma 放在块（复合语句）的开头，则此 pragma 将应用，直到遇到另一个 **FENV_ACCESS** pragma 或到达块的末尾为止。

对于 PA-RISC，可能需要使用 **+Onomoveflops** 编译程序选项，以阻止可能破坏此函数的指定行为的优化。

通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

有关详细信息，请参阅《 HP-UX floating-point guide for HP Integrity servers 》（位于以下站点： **http://www.hp.com/go/fp** ）。

返回值

fetestexcept() 函数返回异常宏的按位 OR，它对应于包含在 *excepts* 中的当前设置的浮点异常。

错误

没有定义任何错误。

举例

如果设置为无效，则调用 **f()** ；如果设置为溢出，则调用 **g()** ：

```
#include <fenv.h>
/*...*/
  int set_excepts;
  /* operations that may raise exceptions */
  set_excepts = fetestexcept(FE_INVALID | FE_OVERFLOW);
```

```
if (set_excepts & FE_INVALID) f();
if (set_excepts & FE_OVERFLOW) g();
```

另请参阅

feclearexcept(3M)、 fegetexceptflag(3M)、 fegettrapenable(3M)、 feraiseexcept(3M)、 fesetexceptflag(3M)、 fesettrapenable(3M)、 fenv(5)。

符合的标准

fetestexcept()： ISO/IEC C99（包括附件 F“IEC 60559 floating-point arithmetic”）

名称

feupdateenv( ) - 更新浮点环境

概要

#include <fenv.h>

仅适用于 **HP Integrity** 服务器

int feupdateenv(const fenv_t *envp);

仅适用于 **PA-RISC** 系统

void feupdateenv(const fenv_t *envp);

说明

feupdateenv() 函数将在其自动存储空间中保存当前引发的浮点异常，安装由 *envp* 指向的对象表示的浮点环境，然后引发所保存的浮点异常。将 **feholdexcept()** 与 **feupdateenv()** 配合使用可以隐藏假的浮点异常。

参数 *envp* 必须指向由对 **fegetenv()** 或 **feholdexcept()** 的调用所设置的对象，或者等于宏 **FE_DFL_ENV** 。

用法

要使用此函数，请使用缺省的 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。确保程序包含 **<fenv.h>**。

对于 Integrity 服务器，请在编译程序命令行上指定 **+Ofenvaccess** ，或者将对此函数的调用放到确定的 **FENV_ACCESS** pragma 的有效范围之内：

```
#pragma STDC FENV_ACCESS ON
```

如果 **FENV_ACCESS** pragma 放在文件中的任何顶级声明之外，则此 pragma 将应用于其后的编译中的所有声明，直到遇到另一个 **FENV_ACCESS** pragma 或达到文件的末尾为止。

如果 **FENV_ACCESS** pragma 放在块（复合语句）的开头，则此 pragma 将应用，直到遇到另一个 **FENV_ACCESS** pragma 或到达块的末尾为止。

对于 PA-RISC，可能需要使用 **+Onomoveflops** 编译程序选项，以阻止可能破坏此函数的指定行为的优化。

通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

有关详细信息，请参阅《HP-UX floating-point guide for HP Integrity servers》（位于以下站点：**http://www.hp.com/go/fp** ）。

返回值

仅适用于 **Integrity** 服务器

此函数始终返回 0，指示已成功执行完所有的操作。

仅适用于 **PA-RISC** 系统

无。

错误

没有定义任何错误。

举例

在 **holdenv** 中存储当前浮点环境，隐藏假的下溢异常，继续任何在遇到对 **feupdateenv()** 的调用前所发生的浮点异常。

```
#include <fenv.h>
/*...*/
  fenv_t holdenv;

  feholdexcept(&holdenv);
  /* perform operations */
  if (/* test for spurious underflow */)
    feclearexcept(FE_UNDERFLOW);
  feupdateenv(&holdenv); /* raise accumulated exceptions */
```

另请参阅

fegetenv(3M)、 feholdexcept(3M)、 fesetenv(3M)、 fenv(5)。

符合的标准

feupdateenv()： ISO/IEC C99（包括附件 F“IEC 60559 floating-point arithmetic”）

名称

fgetpos( )、fsetpos( ) - 保存和恢复流的文件位置指示符

概要

#include <stdio.h>

int fgetpos(FILE *__restrict stream, fpos_t *__restrict pos);

int fsetpos(FILE *stream, const fpos_t *pos);

说明

fgetpos() 将 *stream* 所指向的流的文件位置指示符的当前值存储在 *pos* 所指向的对象中。存储值包含 **fsetpos()** 可用来在调用 **fgetpos()** 时将流重新定位到其位置的信息。

fsetpos() 根据 *pos* 所指向的对象的值来设置 *stream* 所指向流的文件位置指示符，此值必须是以前在同一个流上调用 **fgetpos()** 时设置的值。

成功调用 **fsetpos()** 可以清除流的文件末尾指示符并取消 *ungetc*(3S) 对同一个流的任何影响。调用 **fsetpos()** 之后，更新流的下一项操作可以是输入或输出。

返回值

如果成功，这些函数返回零；否则返回非零值。

错误

如果 **fgetpos()** 失败，则将 **errno** 设置为下列值之一：

[EINVAL] 文件位置的当前值不能在此环境中大小为 **fpos_t** 的对象内正确表示。

基础 **ftell()** 函数可能将 **errno** 设置为其他值（请参阅 *fseek*(3S)）。

警告

如果将这些函数用于不是通过 **fopen()** 打开的文件，则会失败。尤其是，它们不能用于通过 *popen*(3S) 打开的终端或文件。

fsetpos() 对可以追加的流没有影响（请参阅 *fopen*(3S)）。

另请参阅

fgetpos64(3S)、 fseek(3S)、 fopen(3S)、 popen(3S)、 ungetc(3S)、 thread_safety(5)、 glossary(9)。

符合的标准

fgetpos()：AES、SVID3、XPG4、ANSI C

fsetpos()：AES、SVID3、XPG4、ANSI C

名称

fgetpos64( )、 fopen64( )、 freopen64( )、 fseeko64( )、 fsetpos64( )、 fstatvfsdev64( )、 ftello64( )、 ftw64( )、 nftw64( )、statvfsdev64( )、tmpfile64( ) - 支持大型文件的非 POSIX 标准 API 接口。

概要

```
#include <stdio.h>

int fgetpos64(FILE *stream, fpos64_t *pos);

#include <stdio.h>

FILE *fopen64(const char *pathname, const char *type);

#include <stdio.h>

FILE *freopen64(const char *pathname, const char *type, FILE *stream);

#include <stdio.h>

int fseeko64(FILE *stream, off64_t *offset, int whence);

#include <stdio.h>

int fsetpos64(FILE *stream, const fpos64_t *pos);

#include <sys/statvfs.h>

int fstatvfsdev64(int filedes, struct statvfs64 *buf);

#include <stdio.h>

off64_t ftello64(FILE *stream);

#include <ftw.h>

int ftw64(const char *path,
      int (*fn)(const char *obj_path,
            const struct stat64 *obj_stat,
            int obj_flags),
      int depth);

#include <ftw.h>

int nftw64(const char *path,
       int (*fn)(const char *obj_path,
             const struct stat64 *obj_stat,
             int obj_flags,
             struct FTW obj_FTW),
       int depth,
       int flags);
```

```
#include <sys/statvfs.h>

int statvfsdev64(const char *path, struct statvfs64 *buf);

#include <stdio.h>

FILE *tmpfile64(void);
```

说明

支持大型文件的 API 接口。这些 API 接口不是 POSIX 标准的一部分，将来可能被删除。

fgetpos64() 除了 **fgetpos64()** 返回在 **fpos64_t** 中（而不是 **fpos_t** ）的位置外，函数 **fgetpos64()** 和 **fgetpos()** 完全相同。所有其他函数的行为、返回值和错误都是一样的。

fopen64() 在 64 位编译环境中，函数 **fopen64()** 和 **fopen()** 完全相同。函数 **fopen64()** 返回一个指向 FILE 的指针，如果需要，利用此 FILE 可以将文件增大到 2 GB 以上。所有其他函数的行为、返回值和错误与 **fopen()** 的都是一样的。

freopen64() 在 64 位编译环境中，函数 **freopen64()** 和 **freopen()** 完全相同。函数 **freopen64()** 返回一个指向 FILE 的指针，如果需要，利用此 FILE 可以将文件增大到 2 GB 以上。所有其他函数的行为、返回值和错误与 **freopen()** 的都是一样的。

fseeko64() 除了函数 **fseeko64()** 接受 size 参数的一个 **off64_t** （而不是 **off_t** ）外，函数 **fseeko64()** 和 **fseeko()** 完全相同。所有其他函数的行为、返回值和错误都是一样的。

fsetpos64() 除了函数 **fsetpos64()** 接受 pos 参数的一个 **fpos64_t** （而不是 **fpos_t** ）外，函数 **fsetpos64()** 和 **fsetpos()** 完全相同。所有其他函数的行为、返回值和错误都是一样的。

fstatvfsdev64() 除了函数 **fstatvfsdev64()** 接受第二个参数的 **struct statvfs64** （而不是 **struct statvfs** ）外，函数 **fstatvfsdev64()** 和 **fstatvfsdev()** 完全相同。所有其他函数的行为、返回值和错误都是一样的。

ftello64() 除了函数 **ftello64()** 返回在 **off64_t** 中（而不是 **off_t** ）的文件位置外，函数 **ftello64()** 和 **ftello()** 完全相同。所有其他函数的行为、返回值和错误都是一样的。

ftw64() 除了函数 **ftw64()** 使用一个 **struct stat64** 作为函数的第二个参数（它的指针被传递给 **ftw64()** ）外，它和 **ftw()** 完全相同。所有其他函数的行为、返回值和错误都是一样的。

nftw64() 除了函数 **nftw64()** 使用一个 **struct stat64** 作为函数的第二个参数（它的指针被传递给 **nftw64()** ）外，它和 **nftw()** 完全相同。所有其他函数的行为、返回值和错误都是一样的。

statvfsdev64() 除了函数 **statvfsdev64()** 接受第二个参数的 **struct statvfs64** （而不是 **struct statvfs** ）外，函数 **statvfsdev64()** 和 **statvfsdev()** 完全相同。所有其他函数的行为、返回值和错误都是一样的。

tmpfile64() 在 64 位编译环境中，函数 **tmpfile64()** 和 **tmpfile()** 完全相同。两种接口都可以创建一个临时文件，如果需要，此文件能够增大到 2GB 以上。所有其他函数的行为、返回值和错

误都是一样的。

另请参阅

thread_safety(5)。

名称

fgetws()、 fgetws_unlocked() - 可从流文件中获取宽字符字符串

概要

#include <stdio.h>

#include <wchar.h>

wchar_t *fgetws(wchar_t *__restrict ws, int n, FILE *__restrict stream);

过时的接口

wchar_t *fgetws_unlocked(wchar_t *ws, int n, FILE *stream);

备注

fgetws() 与 XPG4 Worldwide Portability Interface （全球可移植性接口）宽字符 I/O 函数兼容。它相当于 *gets*(3C) 中定义的 8 位字符 I/O 函数。

说明

fgetws() 读取 *stream* 中的字符，然后将这些字符转换为对应的宽字符，再将这些字符置于 *ws* 指向的数组中，直到读取了 *n* – 1 个字符，及读取了一个换行符，并将该换行符转换为 *ws* ，或者直到遇到文件结束条件。然后宽字符串以空宽字符结尾。

<wchar.h> 头文件中提供了该函数和类型 **wchar_t** 的定义。

过时的接口

fgetws_unlocked() 可从流文件中获取宽字符字符串。

实际应用信息

将 **fgetws()** 应用到流后，该流将面向宽字符（请参阅 *orientation*(5) ）。

外部语言环境影响

环境变量

LC_CTYPE 确定如何执行宽字符转换。

国际代码集支持

支持单字节字符代码集和多字节字符代码集。

返回值

如果成功完成， **fgetws()** 和 **fgetws_unlocked()** 返回 *ws* 。如果流位于文件结束标记处，则设置流的文件结束标记指示符并返回空指针。

到达文件结束标记后，当与打开的流相对应的文件得到扩展时，对这些函数的所有后续调用都将获得成功，而且文件结束标记指示符将保持设定状态。但是在 UNIX2003 标准环境中（请参阅 *standards*(5) ），这些函数将失败，并且它们将返回空指针；文件结束标记指示符保持设定状态。

如果发生读取错误，将设置流的错误指示符，并设置 **errno** 以指明错误，同时返回一个空指针。

可以使用 **ferror()** 或 **feof()** 来区分错误情况和文件结束的情况。

错误

如果需要将数据读入 *stream* 的缓冲区， **fgetws()** 或 **fgetws_unlocked()** 将会失败，并出现另外一个错误，同时将 **errno** 设置为下列项目之一：

[EAGAIN] 为文件描述符的基础 *stream* 设置了 **O_NONBLOCK** 标记，并且将在读取操作中延迟进程。

[EBADF] 文件描述符的基础 *stream* 不是为读取而打开的有效文件描述符。

[EINTR] 由于接收到信号，或未传输数据，抑或实现未报告此文件的部分传输情况，因此终止了读取操作。

[EILSEQ] 从输入流获取的数据没有构成有效的宽字符字符串。

[EIO] 此进程是后台进程的成员，正尝试从其控制终端读取数据，进程可能忽略或阻塞 **SIGTTIN** 信号；或者进程的进程组被孤立。

可以通过基础 **read()** 函数（请参阅 *read*(2) ）设置其他的 **errno** 值。

警告

fgetws_unlocked() 是已过时接口，只是为了与现有 DCE 应用程序兼容而受到支持。新的多线程应用程序应该使用 **fgetws()** 。

作者

fgetws() 由 OSF 和 HP 联合开发。

另请参阅

ferror(3S)、 flockfile(3S)、 fopen(3S)、 fread(3S)、 getwc(3C)、 putws(3C)、 scanf(3S)、 orientation(5)、 standards(5)、 thread_safety(5)、 glossary(9)。

符合的标准

fgetws()： XPG4

名称

fileno( ) -将流指针映射到文件描述符

概要

#include <stdio.h>

int fileno(FILE *stream);

过时的接口

int fileno_unlocked(FILE *stream);

说明

fileno() 返回与给定 *stream* 相关联的整数文件描述符；请参阅 *open*(2) 。

当程序启动时， **<unistd.h>** 中的下列符号值定义与 **stdin** 、 **stdout** 和 **stderr** 相关联的文件描述符：

STDIN_FILENO	值为零表示标准输入， **stdin** 。
STDOUT_FILENO	值为 1 表示标准输出， **stdout** 。
STDERR_FILENO	值为 2 表示标准错误， **stderr** 。

过时的接口

fileno_unlocked() 将流指针映射到文件描述符

实际应用信息

fileno_unlocked() 是一个过时的接口，它是为了实现与现有的 DCE 应用程序兼容而受支持。新的多线程应用程序应该使用 **fileno()** 。

返回值

fileno() 和 **fileno_unlocked()** 在发生错误时返回 -1。

另请参阅

open(2)、 flockfile(3S)、 fopen(3S)、 thread_safety(5)。

符合的标准

fileno()： AES、SVID2、SVID3、XPG2、XPG3、XPG4、FIPS 151-2 和 POSIX.1

名称

filter — 禁用某些终端功能

概要

#include <curses.h>

void filter(void);

说明

filter() 函数更改初始化终端功能（其假定终端有多行）的算法。对 **initscr()** 或 **newterm()** 的后续调用执行下列附加操作：

- 禁用 **clear** 、 **cud** 、 **cud1** 、 **cup** 、 **cuu1** 和 **vpa** 。
- 将 **home** 字符串的值设置为 **cr** 字符串的值
- 将 **lines** 设置为 1。

对 **filter()** 的任何调用必须发生在 **initscr()** 或 **newterm()** 之前。

返回值

filter() 函数不返回值。

错误

没有定义任何错误。

另请参阅

initscr(3X)、 terminfo(4)，请参阅已定义功能、<curses.h>。

历史变更记录

在 X/Open Curses 第 4 期中首次发布。

名称

flash — 使屏幕闪烁

概要

#include <curses.h>

int flash(void);

说明

flash() 函数用于向用户提示报警。系统使屏幕闪烁，如果不能使屏幕闪烁，则在终端发出可听警报。如果没有出现任何信号，则什么都没有发生。

返回值

flash() 函数始终返回 OK。

错误

没有定义任何错误。

实际应用信息

几乎所有终端都有可听警报，但只有某些终端能够使屏幕闪烁。

另请参阅

beep(3X)、 <curses.h>。

历史变更记录

在 X/Open Curses 第 4 期中首次发布。

在先前的几期中，此函数包含在 **beep()** 条目中。在 X/Open Curses 第 4 期中它移到了自己的条目，参数列表已显式声明为 **void**，并且返回值部分已更改为表明函数始终返回 OK。

名称

flockfile( )、ftrylockfile( )、funlockfile( ) - 多线程应用程序中的显式流锁定

概要

#include <stdio.h>

void flockfile(FILE *stream);

int ftrylockfile(FILE *stream);

void funlockfile(FILE *stream);

说明

flockfile() 、 **ftrylockfile()** 和 **funlockfile()** 函数提供应用程序级显式流锁定功能。线程可以使用这些函数来描述作为一个执行单元的一系列 I/O 语句。

线程使用 **flockfile()** 函数来获取 **(FILE *)** 对象的所有权。

线程使用 **ftrylockfile()** 函数来获取 **(FILE *)** 对象的所有权（如果此对象可用）； **ftrylockfile()** 是 **flockfile()** 的无阻塞版本。

使用 **funlockfile()** 函数来放弃授予线程的所有权。如果不同于当前所有者的线程调用 **funlockfile()** 函数，那么此行为是未定义的。

在逻辑上，每个流都与一个计数相关联。当创建流时，此计数隐式地初始化为零。当此计数为零时，则流是未锁定的。当计数为正数时，单个线程拥有此流。当调用 **flockfile()** 函数时，如果计数为零，或者计数为正数但调用者拥有流，则此计数递增。否则，挂起调用线程，等待计数返回零。每调用一次 **funlockfile()** ，此计数递减。这使得可以嵌套对 **flockfile()** （或对 **ftrylockfile()** 的成功调用）和 **funlockfile()** 的匹配调用。

所有引用 **(FILE *)** 对象的 POSIX.1 与 C 标准函数的行为，就好像它们内部使用 **flockfile()** 和 **funlockfile()** 来获得这些 **(FILE *)** 对象的所有权。

返回值

flockfile() 和 **funlockfile()** 没有返回值。成功时，函数 **ftrylockfile()** 返回零，返回非零表示不能获取锁。

名称

floor( )、floorf( )、floorl( )、floorw( )、floorq( ) - floor 函数

概要

#include <math.h>

double floor(double x);

仅适用于 **HP Integrity** 服务器

float floorf(float x);

long double floorl(long double x);

extended floorw(extended x);

quad floorq(quad x);

说明

floor() 返回不大于 *x* 的最大整数（以双精度数表示）。

仅适用于 **Integrity** 服务器

floorf() 是 **floor()** 的 **float** 版本；它使用 **float** 参数，并返回 **float** 结果。

floorl() 是 **floor()** 的 **long double** 版本；它使用 **long double** 参数，并返回 **long double** 结果。

floorw() 是 **floor()** 的 **extended** 版本；它使用 **extended** 参数，并返回 **extended** 结果。

在 HP-UX 系统中，**floorq()** 等效于 **floorl()**。

用法

（对于 Integrity 服务器）要使用 **floorf()**，请使用缺省 **-Ae** 选项或 **-Aa** 选项编译。

（对于 Integrity 服务器）要使用 **floorl()**、**floorw()** 或 **floorq()**，请使用缺省 **-Ae** 选项或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。

（对于 Integrity 服务器）要使用 **floorw()** 或 **floorq()**，也可以使用 **-fpwidetypes** 选项进行编译。

要使用这些函数，请确保程序包含 **<math.h>**，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

有关详细信息，请参阅《HP-UX floating-point guide for HP Integrity servers》（位于以下站点：**http://www.hp.com/go/fp**）。

返回值

如果 *x* 为 ±INFINITY 或 ±0，则 **floor()** 返回 *x*。

如果 *x* 为 NaN，则 **floor()** 返回 NaN。

如果 *x* 不是整数和有限数，则 **floor()** 可能引发不精确异常。

错误

没有定义任何错误。

另请参阅

ceil(3M)、 fabs(3M)、 fmod(3M)、 rint(3M)、 math(5)。

符合的标准

floor() ： SVID3、XPG4.2、ANSI C 和 ISO/IEC C99 （包括附件 F“IEC 60559 floating-point arithmetic”）

floorf() 和 **floorl()** ： ISO/IEC C99（包括附件 F“IEC 60559 floating-point arithmetic”）

名称

flushinp — 忽略输入

概要

#include <curses.h>

int flushinp(void);

说明

flushinp() 函数忽略（清除）与当前屏幕相关联的输入缓冲区中的任何字符。

返回值

flushinp() 函数始终返回 OK。

错误

没有定义任何错误。

另请参阅

<curses.h>。

历史变更记录

在 X/Open Curses 第 2 期中首次发布。

X/Open Curses 第 **4** 期

为清楚起见，重新编写了该条目。 **flushinp()** 函数的参数列表已显式声明为 **void**。

名称

fma( )、fmaf( )、fmaw( )、fmal( )、fmaq( ) - 浮点乘-加函数

概要

#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);

extended fmaw(extended x, extended y, extended z);

quad fmaq(quad x, quad y, quad z);

说明

fma() 返回 $(x*y) + z$ ，作为一个三元运算进行舍入：它将此值（近似值）计算为无限精度，并根据当前的舍入模式一次舍入到结果格式。

fmaf() 是 **fma()** 的 **float** 版本；它使用 **float** 参数，并返回 **float** 结果。

fmal() 是 **fma()** 的 **long double** 版本；它使用 **long double** 参数，并返回 **long double** 结果。

fmaw() 是 **fma()** 的 **extended** 版本；它使用 **extended** 参数，并返回 **extended** 结果。

在 HP-UX 系统中， **fmaq()** 等效于 **fmal()** 。

FP_FAST_FMA 、 **FP_FAST_FMAF** 和 **FP_FAST_FMAW** 宏定义在 **<math.h>** 中，指示 **fma()** 、 **fmaf()** 和 **fmaw()** 都与乘和加一样快。

用法

这些函数仅适用于 Integrity 服务器。

要使用这些函数，请使用缺省的 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。

要使用 **fmaw()** 或 **fmaq()** ，也可以使用 **-fpwidetypes** 选项进行编译。

要使用这些函数，请确保程序包含 **<math.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

返回值

如果 *x* 和 *y* 中的一个为无穷大，则另一个为零，而 *z* 为 NaN， **fma()** 返回 NaN，并且（可选）引发非法异常。

如果 *x* 和 *y* 中的一个为无穷大，则另一个为零，而 *z* 为 NaN， **fma()** 返回 NaN，并且（可选）引发无效异常。

如果 *x* 乘 *y* 正好为无穷大，而 *z* 也为无穷大但符号相反，则 **fma()** 返回 NaN，并引发无效异常。

fma() 返回带有正确的符号的无穷大来代替太大的值，并引发溢出和不精确异常。

只要结果很小（实质上是非正常表示的或为零）， **fma()** 就引发下溢和不精确异常，因而会有精度损失，并且可能引发这些异常（如果结果很小）。

只要舍入的结果不等于精确结果， **fma()** 就引发不精确异常。

错误

没有定义任何错误。

另请参阅

math(5)。

符合的标准

fma() 、 **fmaf()** 和 **fmal()** ： ISO/IEC C99（包括附件 F“IEC 60559 floating-point arithmetic”）

名称

fmax( )、fmaxf( )、fmaxl( )、fmaxw( )、fmaxq( ) - 最大值函数

概要

#include <math.h>

double fmax(double x, double y);

仅适用于 **HP Integrity** 服务器

float fmaxf(float x, float y);

long double fmaxl(long double x, long double y);

extended fmaxw(extended x, extended y);

quad fmaxq(quad x, quad y);

说明

fmax() 函数确定其参数的最大数值。

仅适用于 **Integrity** 服务器

fmaxf() 是 **fmax()** 的 **float** 版本；它使用 **float** 参数，并返回 **float** 结果。

fmaxl() 是 **fmax()** 的 **long double** 版本；它使用 **long double** 参数，并返回 **long double** 结果。

fmaxw() 是 **fmax()** 的 **extended** 版本；它使用 **extended** 参数，并返回 **extended** 结果。

在 HP-UX 系统中， **fmaxq()** 等效于 **fmaxl()** 。

用法

要使用此函数，请使用缺省的 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。（对于 Integrity 服务器）要使用 **fmaxw()** 或 **fmaxq()** ，请使用 **-fpwidetypes** 选项进行编译。确保程序包含 **<math.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

返回值

fmax() 函数返回其参数的最大数值。

如果一个参数为 NaN，而另一个参数为数值，则 **fmax()** 返回数值参数。

如果两个参数都为 NaN，则 **fmax()** 返回 NaN。

错误

没有定义任何错误。

另请参阅

fdim(3M)、 fmin(3M)、 math(5)。

符合的标准

fmax()、 **fmaxf**() 和 **fmaxl**()： ISO/IEC C99（包括附件 F“IEC 60559 floating-point arithmetic”）

名称

fmin( )、fminf( )、fminl( )、fminw( )、fminq( ) - 最小值函数

概要

#include <math.h>

double fmin(double x, double y);

仅适用于 **HP Integrity** 服务器

float fminf(float x, float y);

long double fminl(long double x, long double y);

extended fminw(extended x, extended y);

quad fminq(quad x, quad y);

说明

fmin() 函数确定其参数的最小数值。

仅适用于 **Integrity** 服务器

fminf() 是 **fmax()** 的 **float** 版本；它使用 **float** 参数，并返回 **float** 结果。

fminl() 是 **fmin()** 的 **long double** 版本；它使用 **long double** 参数，并返回 **long double** 结果。

fminw() 是 **fmin()** 的 **extended** 版本；它使用 **extended** 参数，并返回 **extended** 结果。

在 HP-UX 系统中， **fminq()** 等效于 **fminl()** 。

用法

要使用此函数，请使用缺省的 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。（对于 Integrity 服务器）要使用 **fminw()** 或 **fminq()** ，请使用 **-fpwidetypes** 选项进行编译。确保程序包含 **<math.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

返回值

fmin() 函数返回其参数的最小数值。

如果一个参数为 NaN，而另一个参数为数值，则 **fmin()** 返回数值参数。

如果两个参数都为 NaN，则 **fmin()** 返回 NaN。

错误

没有定义任何错误。

另请参阅

fdim(3M)、 fmax(3M)、 math(5)。

符合的标准

fmin()、 **fminf()** 和 **fminl()**： ISO/IEC C99（包括附件 F“IEC 60559 floating-point arithmetic”）

名称

fmod( )、fmodf( )、fmodl( )、fmodw( )、fmodq( ) - 余数函数

概要

#include <math.h>

double fmod(double x, double y);

float fmodf(float x, float y);

仅适用于 **HP Integrity** 服务器

long double fmodl(long double x, long double y);

extended fmodw(extended x, extended y);

quad fmodq(quad x, quad y);

说明

fmod() 函数返回 *x* 被 *y* 除后的浮点余数 (*f*)，其中 *f* 与 *x* 具有相同的符号，$x = iy + f$（对于某个整数 *i* 而言），且 $|f| < |y|$。

fmodf() 是 **float** 版的 **fmod()**；它采用 **float** 参数，并返回 **float** 结果。

仅适用于 **Integrity** 服务器

fmodl() 是 **long double** 版的 **fmod()**；它采用 **long double** 参数，并返回 **long double** 结果。

fmodw() 是 **extended** 版的 **fmod()**；它采用 **extended** 参数，并返回 **extended** 结果。

fmodq() 等效于 HP-UX 系统上的 **fmodl()**。

用法

要使用这些函数，请使用缺省的 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。

（对于 Integrity 服务器）要使用 **fmodw()** 或 **fmodq()**，也可以使用 **-fpwidetypes** 选项进行编译。

要使用上面的任何函数，请确保程序包含 **<math.h>**，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

返回值

如果 *y* 为 ±无穷大，*x* 不是 ±无穷大，**fmod()** 会返回 *x*。

如果 *x* 为 ±0，但 *y* 为非零值，则 **fmod()** 会返回 *x*。

如果 *x* 或 *y* 为 NaN，则 **fmod()** 会返回 NaN。

如果 *x* 为 ±无穷大，或 *y* 为 0，则 **fmod()** 会返回 NaN 并引发无效异常。

错误

如果 *y* 为 0 或 *x* 为无限值，**fmod()** 会将 **errno** 设置为 [EDOM]。

Integrity 服务器

缺省情况下，Integrity 服务器上的 HP-UX **libm** 函数不设置 **errno** 。对于 **errno** 设置，可以使用 **+Olibmerrno** 选项进行编译。

另请参阅

ceil(3M)、 fabs(3M)、 floor(3M)、 remainder(3M)、 remquo(3M)、 rint(3M)、 math(5)。

符合的标准

fmod() ： SVID3、XPG4.2、ANSI C、ISO/IEC C99 （包括附件 F“IEC 60559 floating-point arithmetic”）

fmodf() 、 **fmodl()** ： ISO/IEC C99（包括附件 F“IEC 60559 floating-point arithmetic”）

名称

fmtmsg() - 在标准错误和控制台上显示格式化消息

概要

```
#include <fmtmsg.h>

int fmtmsg(
    long class,
    const char *label,
    int severity,
    const char *text,
    const char *action,
    const char *tag
);
```

说明

fmtmsg() 例行程序可用作与语言无关的错误消息服务。消息可在系统控制台、标准错误或同时在两者上显示，具体取决于 *class* 参数的设置。用户可以控制在标准错误上显示的消息的格式，及指定要显示的消息数目。但是，用户不能控制在系统控制台上显示的消息格式。

所有消息都将指定 *class* ，用于指示错误情况的原因和状态，以及消息应该定向的位置。通过指定 **MM_NULLMC** ，可以从功能上排除 *class* 。如果使用此设置，将不生成任何消息。类的类型包括：

显示	显示类允许的值包括将消息定向到标准错误的 **MM_PRINT** 、将消息定向到系统控制台的 **MM_CONSOLE** ，或将消息同时定向到标准错误和系统控制台的 **MM_PRINT | MM_CONSOLE** 。
“错误类型”	错误类型类可能是下列其中之一：表示硬件错误的 **MM_HARD** 、表示软件错误的 **MM_SOFT** ，或者表示固件错误的 **MM_FIRM** 。
“错误原因”	错误原因类可能是下列其中之一：表示应用程序的 **MM_APPL** 、表示操作系统实用程序的 **MM_UTIL** ，或者表示系统错误的 **MM_OPSYS** 。
“恢复状态”	恢复状态类可能是下列其中之一： **MM_RECOVER** （表示可以从错误状态下恢复）或 **RMM_NRECOV** （表示不可以恢复）。

剩下的参数在功能上是可选的，因为生成的消息可能会排除其中的任意一个或所有项目。将按参数的顺序讨论其中的每项。

label 组件说明错误产生的位置。其格式为 *major:minor* ，其中的 *major* 为 10 个字符的字段，用于指定生成消息的主系统； *minor* 为 14 个字符的字段，用于指定错误的子系统。例如，

mail:send

通过将其设置为 **MM_NULLLBL** 或 **NULL** ，可排除该组件。

severity 组件说明失败的重要性程度。预定义的严重性等级包括：

MM_NOSEV	未指定任何严重性并且未生成任何输出字符串。
MM_INFO	该消息只是提供参考信息，将生成 **INFO** 字符串。
MM_WARNING	可能出现了错误，应执行检查。将生成 **WARNING** 字符串。
MM_ERROR	检测到错误。将生成 **ERROR** 字符串。
MM_HALT	检测到不可恢复的错误。将生成 **HALT** 字符串。

text 组件以最具体的方式描述失败的性质。要排除与错误原因有关的所有疑点，该组件应提供足够详细的错误说明。通过将 *text* 设置为 **MM_NULLTXT** 或 **MM_NULL** ，可排除该组件。

action 组件描述可用于从错误中恢复的选项。输出时，以字符串 **TO FIX:** 开头。通过将 **action** 设置为 **MM_NULLACT** 或 **NULL** ，可禁用该组件。

tag 组件用于将用户指定到相应的文档，以更正或避免错误。通过将 *tag* 设置为 **MM_NULLTXT** 或 **RMM_NULL**，可禁用该组件。

外部语言环境影响

用户通过使用以下两个环境变量，可以控制在标准错误上生成的消息的显示形式： **MSGVERB** 和 **SEV_LEVEL**。这些环境变量对控制台消息不起作用。

MSGVERB 环境变量说明用户希望看到的组件。如果指定了多个组件， **MSGVERB** 的值将是一个或多个以逗号分隔的组件列表。调用 **fmtmsg()** 时，将显示 **MSGVERB** 中列出的值不为 NULL 的所有组件；其他所有组件都将被排除。只有组件名 *label* 、 *severity* 、 *text* 、 *action* 和 *tag* 才是有效的列表元素。其他任何元素（或者空列表）均视为 **MSGVERB** 格式的错误。如果 **MSGVERB** 是错误的，或者未对其进行设置，则 **fmtmsg()** 将生成完整消息。

如果不是上面说明的严重性，则用户可以通过 **SEV_LEVEL** 环境变量，控制针对严重性组件显示的文字字符串。无法覆盖预定义的严重性等级（从 MM_NOSEV 到 **MM_HALT** ）。

SEV_LEVEL 的值是一个或多个以逗号分隔的等级说明符列表。级别说明符是由三个以逗号分隔的项目组成的，采用格式 *identifier* ， *level* ， *message* 的列表。 *identifier* 不能由 **fmtmsg()** 使用，并且只在要实现兼容的情况下才使用它。但该项目不是可选项，因为 **fmtmsg()** 在检查其他两个部分的等级说明符时需要使用它。 *level* 是大于 4 的数字，用于指示定义的等级。 *message* 是要在 **severity** 字段中显示的字符串，其显示方式类似于针对 **MM_HALT** 和 **MM_INFO** 严重性显示的 **HALT** 和 **INFO** 。包含大于或小于三个项目的等级说明符无效，会将其视为空等级说明符。值无效或值为空的 **SEV_LEVEL** 列表不会影响 **fmtmsg()** 的行为。

返回值

fmtmsg() 例行程序在退出时，将返回下列值之一：

MM_OK	成功。
MM_NOTOK	失败。
MM_NOMSG	对标准错误失败，对系统控制台成功。

MM_NOCON 对标准错误成功，对系统控制台失败。

作者

fmtmsg() 由 OSF 和 HP 联合开发。

另请参阅

printf(3S)、 thread_safety(5)。

符合的标准

fmtmsg：SVID3

名称

fnmatch( ) - 与文件名模式匹配

概要

#include <fnmatch.h>

int fnmatch(const char *pattern, const char *string, int flags);

说明

fnmatch() 执行模式匹配，如模式匹配表示法下的 *regexp*(5) 中所述。缺省情况下，并不应用对于文件名扩展的规则限制；即，将句点（点）和斜线作为普通字符进行匹配。可以使用以下所述的标志修改这一缺省行为。

flag 参数修改 *pattern* 和 *string* 的解释。如果在 **<fnmatch.h>** 中定义的 **FNM_PATHNAME** 在 *flag* 中被设置，*string* 中的斜线字符必须与 *pattern* 中的斜线字符明确匹配；它既不能由星号或问号特殊字符匹配，也不能由一个括号表达式匹配。

如果 **FNM_PERIOD** 在 *flag* 中被设置，则必须明确匹配一个前导句点 (**.**) 。它不能由一个括号表达式、问号或星号匹配。缺省情况下，如果句点是 *string* 中的第一个字符，则它为前导字符。如果 **FNM_PATHNAME** 在 *flag* 被设置，如果句点是 *string* 的第一个字符或者紧随在一个斜线之后，则句点为前导字符。

如果 **FNM_NOESCAPE** 未在 *flag* 中设置，则 *pattern* 中后面跟有其他任意字符的反斜线符号 (****) 与 *string* 中的第二个字符匹配。具体而言， **** 与 *string* 中的反斜线匹配。如果设置了 **FNM_NOESCAPE** ，则将反斜线字符视为普通字符。

如果 *flag* 为零，则将斜线字符和句点视为正常字符。如果 *flag* 拥有任意其它取值，结果为未定义。

返回值

如果 *string* 与 *pattern* 所指定的模式匹配，则 **fnmatch()** 返回零。否则， **fnmatch()** 返回非零值。

举例

以下程序片断使用 **fnmatch()** 检查目录中每个违反模式 ***.c** 的文件：

```
pattern = "*.c";

while(dp = readdir(dirp)){
   if((fnmatch(pattern, dp->d_name,0)) == 0){
      /* do processing for match */
           ...
   }
}
```

作者

fnmatch() 由 OSF 和 HP 联合开发。

另请参阅

sh(1)、glob(3C)、 thread_safety(5)。

符合的标准

fnmatch()：XPG4、POSIX.2

名称

fopen()、 freopen()、 fdopen() - 打开（或重新打开）流文件或将文件转换为流

概要

#include <stdio.h>

FILE *fopen(const char *__restrict pathname, const char *__restrict type);

FILE *freopen(const char *__restrict pathname, const char *__restrict type, FILE *__restrict stream);

FILE *fdopen(int fildes, const char *type);

说明

fopen() 打开 *pathname* 命名的文件，并将 *stream* 与其关联。 **fopen()** 返回指向与 *stream* 关联的 **FILE** 结构的指针。

freopen() 使用指定的文件替换打开的 *stream* 。将关闭原始的 *stream* ，无论最终是否成功打开。 **freopen()** 将返回指向与 *stream* 关联的 **FILE** 结构的指针，并隐式调用 **clearerr()** （请参阅 *ferror*(3S) ）。成功调用 **freopen()** 后，将清除 *stream* 的方向（请参阅 *orientation*(5) ）。

freopen() 通常用于将与 **stdin** 、 **stdout** 和 **stderr** 关联的预先打开的 *streams* 附加到其他文件中。

fdopen() 将流与文件描述符关联。文件描述符可从 **open()** 、 **dup()** 、 **creat()** 或 **pipe()** 中获取（请参阅 *open*(2) 、 *dup*(2) 、 *creat*(2) 和 *pipe*(2) ），这些描述符可打开文件，但不会指向 **FILE** 结构流的指针。对于许多段 (3S) 库例行程序来说，流是必要的输入内容。流的 *type* 必须与打开文件的模式一致。有关 **fdopen()** 调用中使用的 *type* 的含义，除了 **w** 、 **w+** 、 **wb** 和 **wb+** 不会导致截断文件外，在其他方面与上文的说明完全相同。

pathname 指向包含要打开的文件的名称的字符型字符串。

type 包含下列其中一个值的字符型字符串。以下值中的 **b** 不起作用。它用来区分二进制文件和文本文件；但是，在 UNIX 系统上，这些类型的文件没有差别（要符合 ISO C 标准，此项目是必需的）。

r 或 **rb**	打开文件进行读取
w 或 **wb**	截断到零长度，或者创建文件进行写入
a 或 **ab**	追加；在文件结尾打开文件进行写入，或者创建文件进行写入
r+、**rb+** 或 **r+b**	打开文件进行更新（读取和写入）
w+、**wb+** 或 **w+b**	将文件截断到零长度，或者创建文件进行更新
a+、**ab+** 或 **a+b**	追加；在文件结尾打开或创建文件进行更新

打开某个文件进行更新时，可以在生成的 *stream* 上同时执行输入和输出。但是，如果没有附加调用 **fflush()** 或文件定位函数 （**fseek()** 、 **fsetpos()** 或 **rewind()** ），则不能直接在输出后紧跟输入。同时，如果没有附加调用文件

定位函数，则不能直接在输入后紧跟输出，除非输入操作遇到文件结束标记。

打开文件进行追加时（即， *type* 为 **a** 、 **a+** 、 **ab+** 或 **a+b** 时），不能覆盖文件中已有的信息。所有输出都是在文件结尾写入的，无论是否干预对 **fseek()** 的调用都是如此。如果两个单独的进程打开了同一文件进行追加，则每个进程都可以任意写入文件，而不必担心会损坏对方写入的输出。来自这两个进程的输出，将按其写入的顺序在文件中相互混合。

注释

HP-UX 二进制文件的 *types* 等效于与其对应的非二进制类型。例如，类型 **r** 和 **rb** 是等效的。

返回值

成功完成后， **fopen()** 、 **fdopen()** 和 **freopen()** 将返回指向流的 **FILE *** 指针。否则，将返回空指针，并设置 **errno** 以指明错误。

错误

如果出现任意情况，则 **fopen()** 、 **fdopen()** 和 **freopen()** 将会失败：

[EBADF] *fildes* 参数不是有效的文件描述符。

[EINVAL] *type* 参数不是有效的模式。

[ENOMEM] 用来分配缓冲区的空间不足。

如果出现以下情况，则 **fopen()** 和 **freopen()** 将会失败：

[EACCES] 拒绝对路径前缀的组成部分的搜索权限；或者文件存在，同时拒绝 *type* 指定的权限；或者文件不存在，同时拒绝要创建的文件的父目录的写入权限。

[EINTR] 调用 **fopen()** 或 **freopen()** 函数期间捕获到一个信号。

[EISDIR] 指定的文件是一个目录， *type* 需要具有写入权限。

[EMFILE] 调用进程尝试超出打开文件的限制。

[ENAMETOOLONG]
pathname 字符串的长度超过 **PATH_MAX** ，或者当 **POSIX_NO_TRUNC** 有效时，路径名组成部分的长度超过 **NAME_MAX** 。

[ENFILE] 系统文件表已满。

[ENOENT] 指定的文件不存在，或者 *pathname* 参数指向空字符串。

[ENOSPC] 无法扩展应包含新文件的目录或文件系统，文件不存在，将创建该文件。

[ENOTDIR] 路径前缀的某部分不是目录。

[ENXIO] 指定的文件为字符专用或块专用的文件，并且与此专用文件关联的设备不存在。

[EOVERFLOW] 指定的文件为常规文件，在这种环境下，无法在大小为 **off_t** 的对象中正确表示该文件的大小。

[EROFS] 指定的文件驻留在只读的文件系统上，*type* 需要写入访问权限。

可以通过 **fopen()** 和 **freopen()** 函数（请参阅 *open*(2) ）执行基本的 **open()** 调用，来设置 **errno** 的其他值。

另请参阅

creat(2)、 dup(2)、 open(2)、 pipe(2)、 fclose(3S)、 fgetpos64(3S)、 fseek(3S)、 popen(3S)、 setbuf(3S)、 orientation(5)、 thread_safety(5)、 glossary(9)。

符合的标准

fopen()：AES、SVID2、SVID3、XPG2、XPG3、XPG4、FIPS 151-2、POSIX.1、ANSI C

fdopen()：AES、SVID2、SVID3、XPG2、XPG3、XPG4、FIPS 151-2、POSIX.1

freopen()：AES、SVID2、SVID3、XPG2、XPG3、XPG4、FIPS 151-2、POSIX.1、ANSI C

名称

fpclassify( ) - 浮点值分类宏

概要

#include <math.h>

int fpclassify(*floating-type* **x);**

说明

fpclassify() 宏将自己的参数值分为以下类别：NaN、无限、标准化、反规范化或零。参数必须是浮点类型，并且分类是基于参数类型的。对于 HP Integrity 服务器，参数可以是任何浮点类型。对于 PA-RISC，参数必须为 **double** 或 **float** 中的任意一个。

在与 **signbit()** 宏一起使用的情况下， **fpclassify()** 宏可以取代 **fpclassify()** 和 **fpclassifyf()** 函数。这些函数已过时，将不再受支持。

用法

要使用 **fpclassify()** 宏，请使用缺省的 **-Ae** 选项或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。确保程序包含 **<math.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

返回值

fpclassify() 宏将返回与其参数类型和参数值相对应的数字分类宏的值。

返回值为下列的宏之一（这些宏在 **<math.h>** 中定义）：

FP_NORMAL 标准化
FP_ZERO 零
FP_INFINITE 无穷大
FP_SUBNORMAL 反规范化
FP_NAN NaN

每个可能的参数值都属于这些类别之一，因此该宏决不会生成错误。

该宏不会引发任何异常。

错误

没有定义任何错误。

举例

如果 **x** 为反规范化的 **float** 值或为零，则应采取一定的操作：

```
#include <math.h>
/*...*/
  int class;
  float x;
  /*...*/
  class = fpclassify(x);
```

```
if ( (class == FP_SUBNORMAL) || (class == FP_ZERO) ) {
  /*...*/
}
```

另请参阅

isfinite(3M)、 isinf(3M)、 isnan(3M)、 isnormal(3M)、 signbit(3M)、 math(5)。

符合的标准

fpclassify()： ISO/IEC C99

名称

fread()、 fwrite() - 为流文件缓冲的二进制输入（或输出）

概要

```
#include <stdio.h>

size_t fread(void *__restrict ptr, size_t size, size_t nitems, FILE *__restrict stream);

size_t fwrite(const void *__restrict ptr, size_t size, size_t nitems, FILE *__restrict stream);
```

过时的接口

```
size_t fread_unlocked(
  void *ptr, size_t size, size_t nitems, FILE *stream);

size_t  fwrite_unlocked(
  const void *ptr, size_t size, size_t nitems, FILE *stream);
```

说明

fread() 复制到 *ptr* 所指向的数组中，最多可以从指定的输入 *stream* 中复制 *nitems* 个项目，其中数据项目是长度为 *size* 的字节序列。（不必以空字节结尾)。在读取 *stream* 时，如果遇到文件末尾或错误条件，或者如果已经读取了 *nitems* 个项目， **fread()** 就停止添加字节。如果已定义，则 **fread()** 将文件指针留在 *stream* 中，并指向最后一个读取的字节后面的字节（如果有）。 **fread()** 不更改 *stream* 的内容。

fwrite() 最多可以从 *ptr* 所指向的数组向指定的输出 *stream* 添加 *nitems* 个数据项目。当 **fwrite()** 向 *stream* 添加了 *nitems* 个数据项目或遇到错误条件时，它就停止添加。 **fwrite()** 不更改 *ptr* 指向的数组的内容。

参数 *size* 通常为 **sizeof(*** *ptr*) ，其中伪函数 **sizeof** 指定 *ptr* 所指向的项目的长度。

过时的接口

fread_unlocked() 和 **fwrite_unlocked()** 为流文件缓冲二进制输入（或输出）。

实际应用信息

在将 **fread()** 或 **fwrite()** 应用于流后，此流就变成面向字节的（请参阅 *orientation*(5) ）。

返回值

fread() 、 **fwrite()** 、 **fread_unlocked()** 和 **fwrite_unlocked()** 返回读写的项目数量。如果 *size* 或 *nitems* 为 0，那么就没有读写字符，并且返回 0。

仅当出现读错误或文件结束标记时，返回的值才会小于 *nitems* 。

到达文件结束标记后，当与打开的流相对应的文件得到扩展时，对 **fread()** 的所有后续调用都将获得成功，而且文件结束标记指示符将保持设定状态。但是在 UNIX2003 标准环境中（请参阅 *standards*(5) ），该函数将返回零，并且文件结束标记指示符保持设定状态。

必须使用 **ferror()** 或 **feof()** 函数来区分错误条件和文件结束条件。

错误

有关 **fread()** 返回的错误的说明，请参考 *getc*(3S) 。

有关 **fwrite()** 返回的错误的说明，请参考 *putc*(3S) 。

警告

fread_unlocked() 和 **fwrite_unlocked()** 为过时的接口，它是为了实现与现有的 DCE 应用程序兼容而受支持。新的多线程应用程序应当使用 **fread()** 和 **fwrite()** 。

另请参阅

read(2) 、 write(2) 、 fopen(3S) 、 flockfile(3S) 、 getc(3S) 、 gets(3S) 、 printf(3S) 、 putc(3S) 、 puts(3S) 、 scanf(3S)、 orientation(5)、 standards(5)、 thread_safety(5)、 glossary(9)。

符合的标准

fread():AES、SVID2、SVID3、XPG2、XPG3、XPG4、FIPS 151-2、POSIX.1 和 ANSI C

fwrite():AES、SVID2、SVID3、XPG2、XPG3、XPG4、FIPS 151-2、POSIX.1 和 ANSI C

名称

frexp( )、frexpf( )、frexpl( )、frexpw( )、frexpq( ) - 从浮点数提取尾数和指数

概要

#include <math.h>

double frexp(double value, int *exp);

仅适用于 **HP Integrity** 服务器

float frexpf(float value, int *exp);

long double frexpl(long double value, int *exp);

extended frexpw(extended value, int *exp);

quad frexpq(quad value, int *exp);

说明

frexp() 函数将浮点数分成规范化部分和二次整数幂。它在 *exp* 指向的 **int** 对象中存储整数指数。

仅适用于 **Integrity** 服务器

frexpf() 是 **frexp()** 的 **float** 版本；它采用 **float** 参数作为第一个参数，并返回 **float** 结果。

frexpl() 是 **frexp()** 的 **long double** 版本；它采用 **long double** 参数作为第一个参数，并返回 **long double** 结果。

frexpw() 是 **frexp()** 的 **extended** 版本；它采用 **extended** 参数作为第一个参数，并返回 **extended** 结果。在 HP-UX 系统中， **frexpq()** 等效于 **frexpl()** 。

用法

（对于 Integrity 服务器）要使用 **frexpf()** ，请使用缺省的 **-Ae** 选项或 **-Aa** 选项编译。

（对于 Integrity 服务器）要使用 **frexpl()** 、 **frexpw()** 或 **frexpq()** ，请使用缺省的 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。

（对于 Integrity 服务器）要使用 **frexpw()** 或 **frexpq()** ，也可以使用 **-fpwidetypes** 选项进行编译。

要使用上面的任何函数，请确保程序包含 **<math.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。有关详细信息，请参阅《HP-UX floating-point guide for HP Integrity servers》（位于以下站点：**http://www.hp.com/go/fp** ）。

返回值

frexp() 函数返回值 *x* ，这样 *x* 就是位于区间 [0.5, 1] 的 **double** 值或者为零，而 *value* 等于 *x* 乘以 **exp* 的二次方。

如果 *value* 为零，则返回值并将零存储在 *exp* 指向的对象中。

如果 *value* 是 NaN，则 **frexp()** 返回 NaN，并且 **exp* 的值是未定义的。

如果 *value* 为 ±INFINITY，则 **frexp()** 返回 *value* ，并且 **exp* 的值是未定义的。

这些函数不会引发异常。

错误

没有定义任何错误。

另请参阅

ilogb(3M)、 ldexp(3M)、 logb(3M)、 modf(3M)、 scalb(3M)、 scalbln(3M)、 scalbn(3M)、 math(5)。

符合的标准

frexp()： SVID3、XPG4.2、ANSI C 和 ISO/IEC C99 （包括附件 F“IEC 60559 floating-point arithmetic”）

frexpf()、 **frexpl()**： ISO/IEC C99（包括附件 F“IEC 60559 floating-point arithmetic”）

名称

fseek( )、fseeko( )、rewind( )、ftell( )、ftello( ) - 重新定位流中的文件指针

概要

#include <stdio.h>

int fseek(FILE *stream, long int offset, int whence);

int fseeko(FILE *stream, off_t offset, int whence);

void rewind(FILE *stream);

long int ftell(FILE *stream);

off_t ftello(FILE *stream);

过时的接口

int fseek_unlocked(FILE *stream, long int offset, int whence);

void rewind_unlocked(FILE *stream);

long int ftell_unlocked(FILE *stream);

说明

fseek() 可设置 *stream* 的文件位置指示符。通过将 *offset* 添加到 *whence* 指定的位置，可以获取从文件开头开始以字节计量的新位置。指定的位置包括文件开头 **SEEK_SET** 、当前位置 **SEEK_CUR** 和文件结束 **SEEK_END** 。

fseeko() 是 **_LARGEFILE_SOURCE** 编译选项提供的非 POSIX 标准 API。它与 **fseek()** 相同，区别在于 *offset* 参数是 **off_t** 而不是 **long int** 。其他所有功能性行为、返回值和错误都与 POSIX **fseek()** 相同。

如果 *stream* 上的最新操作（除 **ftell()** 外）为 **fflush()** ，则将调整基础的打开文件描述中的文件偏移量，以反映 **fseek()** 指定的位置。

rewind(*stream***)** 与 **fseek (stream, 0L, SEEK_SET)** 等效，但它不返回任何值。

fseek() 和 **rewind()** 可以消除 *ungetc*(3S) 的任何影响。

执行 **fseek()** 或 **rewind()** 后，对打开要进行更新的文件所要执行的下一操作可能是输入，也可能是输出。 **fseek()** 可清除 *stream* 的 EOF 指示符。 **rewind()** 可隐式执行 **clearerr()** 调用（请参阅 *ferror*(3S) ）。

ftell() 可返回相对于文件（与指定的 *stream* 关联）开头的当前字节的偏移量。

ftello() 是 **_LARGEFILE_SOURCE** 编译选项提供的非 POSIX 标准 API。它与 **ftell()** 相同，区别在于它返回的是 **off_t** ，而不是 **long int** 。其他所有的行为、返回值和错误都与 POSIX **ftell()** 相同。

过时的接口

fseek_unlocked() 、 **rewind_unlocked()** 和 **ftell_unlocked()** 可重新定位流中的文件指针。

返回值

fseek() 和 **fseek_unlocked()** 成功时返回零。否则，将返回 −1，并设置 **errno** 以指明错误。

ftell() 和 **ftell_unlocked()** 可返回流的文件位置指示符的当前值，此位置从文件开头开始以字节计量。否则，**ftell()** 和 **ftell_unlocked()** 将返回 −1，并设置 **errno** 以指明错误。

rewind() 和 **rewind_unlocked()** 不返回任何值。因此，需要检测错误的任何应用程序都应该在调用 **rewind()** 或 **rewind_unlocked()** 之前清除 **errno** 。这样，在执行完成后，如果 **errno** 为非零值，则应假定出现了错误。

错误

如果 *stream* 未进入缓冲区或者缓冲的数据需要刷新，或者满足以下任一情况时， **fseek()** 、 **fseeko()** 、 **fseek_unlocked()** 、 **ftell()** 、 **ftello()** 、 **ftell_unlocked()** 、 **rewind()** 和 **rewind_unlocked()** 都将会失败：

[EAGAIN] 为文件描述符设置了 **O_NONBLOCK** 标记，并且将在写入操作中延迟进程。

[EBADF] *stream* 为 NULL。

[EBADF] 未打开基础文件进行写入。

[EFBIG] 尝试对一个超出进程文件大小限制或最大文件大小（请参阅 *ulimit*(2) ）的文件进行写入。

[EINVAL] 在此环境下，无法在 **long** 类型或 **off_t** 大小的对象中正确表示文件 *offset* 。

[EINTR] 执行写入操作期间捕获到一个信号。

[EIO] 该进程位于后台进程组内，正尝试写入到其控制终端，设置了 **TOSTOP** ，此进程不会忽略或阻塞 **SIGTTOU** 信号，但进程的进程组是孤立的。

[ENOSPC] 包含文件的设备上没有任何可用空间。

[EPIPE] 尝试写入因未打开而无法供任何进程读取的管道。此外还向进程发送了 **SIGPIPE** 信号。

[ESPIPE] 尝试执行搜索操作，位于 *stream* 之下的文件描述符与管道关联。

如果出现任意情况，则 **fseek()** 和 **fseek_unlocked()** 将失败：

[EINVAL] *whence* 参数无效，或者文件位置指示符可能将设置为负值。

可以通过基础的 **write()** 和 **lseek()** 函数（请参阅 *write*(2) 和 *lseek*(2) ）设置 **errno** 的其他值。

警告

在 HP-UX 系统上， **ftell()**或ftell_unlocked() 返回的偏移量是以字节计量的，允许搜索相对于该偏移量的位置。但是，移植到非HP-UX 系统后，应直接使用 **fseek()** ，而不必依赖于从 **ftell()** 中获取的任何偏移量，这是因为，如果这个偏移量不是在特定的操作系统上以字节计量的，则对其执行运算是没有意义的。

fseek() 、 **fseek_unlocked()** 、 **rewind()** 和 **rewind_unlocked()** 对在追加模式（请参阅 *fopen*(3S) ）下打开的流不起作用。

fseek_unlocked() 、 **ftell_unlocked()** 和 **rewind_unlocked()** 是为了与现有 DCE 应用程序兼容而支持的已过时接口。新的多线程应用程序应该使用 **fseek()** 、 **ftell()** 和 **rewind()** 。

另请参阅

lseek(2) 、 write(2) 、 ferror(3S) 、 flockfile(3S) 、 fopen(3S) 、 fgetpos(3S) 、 fgetpos64(3S) 、 ungetc(3S) 、 thread_safety(5)。

符合的标准

fseek()：AES、SVID2、SVID3、XPG2、XPG3、XPG4、FIPS 151-2、POSIX.1、ANSI C

ftell()：AES、SVID2、SVID3、XPG2、XPG3、XPG4、FIPS 151-2、POSIX.1、ANSI C

rewind()：AES、SVID2、SVID3、XPG2、XPG3、XPG4、FIPS 151-2、POSIX.1、ANSI C

名称

ftok( ) - 创建进程间通信标识符

概要

#include <sys/ipc.h>

key_t ftok(const char *path, int id);

说明

所有的进程间通信工具都要求用户提供键值， **msgget()** 、 **semget()** 和 **shmget()** 系统调用将使用此键值来获取进程间通信标识符（请参阅 *msgget*(2) 、 *semget*(2) 和 *shmget*(2) ） 。

ftok() 根据 *path* 和 *id* 返回键值，此键值可用于随后的 **msgget()** 、 **semget()** 和 **shmget()** 系统调用。

ftok() 函数的参数如下:

path	必须是可访问进程的现有文件的路径名。
id	是唯一标识项目的字符。这意味着只有 ID 的低八位是重要的。请注意，当以相同的 *id* 调用时， **ftok()** 返回链接的文件的相同键值；当以相同的文件名称而不同的 *id* 调用时，它返回不同的键值。

返回值

如果 *path* 不存在或不可以访问进程，则 **ftok()** 返回 (**key_t)-1** 。

举例

下面对 **ftok**() 的调用返回与文件 *myfile* 和 id **A** 相关的键值:

```
key_t mykey;
    mykey = ftok ("myfile", 'A');
```

警告

如果其 *path* 传送到 **ftok()** 的文件在键值仍然引用此文件时被删除，则将来以相同的 *path* 和 *id* 对 **ftok()** 的调用将返回错误。如果重新创建相同的文件，则 **ftok()** 有可能返回与原来调用的不同的键值。

另请参阅

intro(2)、 msgget(2)、 semget(2)、 shmget(2)、 thread_safety(5)。

名称

ftw()、nftw() - 执行函数遍历文件树

概要

```
#include <ftw.h>

int ftw (const char *path,
        int (*fn)(const char *obj_path,
                const struct stat *obj_stat,
                int obj_flags),
        int depth);

int nftw (const char *path,
         int (*fn)(const char *obj_path,
                 const struct stat *obj_stat,
                 int obj_flags,
                 struct FTW obj_FTW),
         int depth,
         int flags);
```

UNIX95

```
int nftw (const char *path,
        int (*fn)(const char *obj_path,
                const struct stat *obj_stat,
                int obj_flags,
                struct *FTW obj_FTW),
        int depth,
        int flags);
```

过时的接口

```
int nftw2(const char *path,
        int (*fn)(const char *obj_path,
                const struct stat *obj_stat,
                int obj_flags,
                struct FTW obj_FTW),
        int depth,
        int flags);
```

说明

ftw() 函数以递归方式遍历位于 *path* 中的目录分层结构。对于分层结构中的每个对象， **ftw()** 将调用 *fn* ，并为其传递一个指向包含对象名称的 null 结尾字符型字符串的指针、指向包含对象有关信息（请参阅 *stat*(2) 中的 **lstat()**）的 **stat** 结构的指针，以及一个整数 *obj_flags* 。 **<ftw.h>** 头文件中定义了 *obj_flags* 的可能值，包括：

FTW_F 该对象是一个文件。

FTW_D 该对象是一个目录。

FTW_SL 该对象是一个符号链接。

FTW_DNR 该对象是没有读取权限的目录。不可以针对它的任何派生对象调用 *fn* 。

FTW_NS 在该对象上执行 **lstat()** 失败。 **stat** 结构的内容不可预测。如果失败是由于缺少权限导致的，遍历将会继续。对于其他所有错误， **ftw()** 将终止遍历，并返回 **-1** 。

到达树的尽头、调用 *fn* 后返回非零值、在 **ftw()** 中检测到错误（例如 I/O 错误）时，树遍历才会终止。如果到达树的尽头， **ftw()** 将返回零。如果 *fn* 返回非零值， **ftw()** 将停止树遍历，并返回一个值，该值 *fn* 返回的值相同。如果 **ftw()** 检测到错误，将返回 **-1** ，同时在 **errno** （请参阅 *errno*(2) ）中设置错误类型。

ftw() 报告目录后再报告它的任何派生对象。

ftw() 和 **nftw()** 针对树中的每个层次使用一个文件描述符。 *depth* 参数限制可以使用的文件描述符数目。如果 *depth* 为 0 或负值，则效果与等于 1 时相同。 *depth* 不得大于当前可用的文件描述符数目。为获得最佳性能，*depth* 的大小至少应该等于树中的层数。

nftw() 与 **ftw()** 类似，不同之处在于它可以使用附加参数 *flags* ，并且不报告或进入遍历期间已经访问过的目录。*flags* 字段是下列值的逻辑或运算（不包含符号）得到的， **<ftw.h>** 头文件中定义了这些值：

FTW_PHYS 如果设置该值， **nftw()** 将执行物理遍历。此时，它的后面不接符号链接，但可以接硬链接。

如果清除该值， **nftw()** 后面可以同时接符号链接和硬链接。此外，如果定义了 **UNIX95** 环境，就不会向 *fn* 多次报告任何对象。而标准的 HP **nftw()** 行为是只要引用一个文件，就可以报告该文件。

FTW_MOUNT 如果设置该值， **nftw()** 只报告与 *path* 相同的文件系统中的文件（有关环回文件系统 (LOFS) 的一种例外情况，请参阅 **WARNINGS** ）。

如果清除该值， **nftw()** 将报告遍历时遇到的所有对象。

FTW_DEPTH 如果设置该值， **nftw()** 将执行深度优先的搜索。报告目录的内容之后，再向 *fn* 报告目录。

如果清除该值，将在报告目录之后，再报告其派生对象。

FTW_CHDIR 如果设置该值，遍历将对每个目录执行 **chdir()** （请参阅 *chdir*(2) ），然后再读取其内容。向 *fn* 报告目录时，当前工作目录将成为所报告目录的父级。目录必须同时具有读取和执行权限，否则将目录当作 **FTW_DNR** 对象进行报告；此外，不执行且不输入 **chdir()** 。

如果清除该值， **nftw()** 将不更改进程当前的工作目录，同时目录只需要读取权限，以便将其当作 **FTW_D** 或 **FTW_DP** 对象进行报告。

FTW_SERR 如果设置该值，在发生 **lstat()** 或 **stat()** 失败时，将使用 **FTW_NS** 对象标志调用 *fn* ，同时遍历将会继续。

如果清除该值，在由于缺少权限而导致 **lstat()** 或 **stat()** 失败时，将使用 **FTW_NS** 对象类型调用 *fn* 。如果定义了 **UNIX95** 环境，遍历将会继续。而 HP 标准行为将会终止遍历，并返回 **-1** 。如果不是出现权限错误，或者不是由于其他系统调用（如 *opendir*(3C) 或 *readdir*(3C) ）而导致的错误， **nftw()** 将不调用 *fn* ，并终止遍历和返回 **-1** 。

对于每个报告的对象， **nftw()** 将使用四个参数调用 *fn* 。第一个参数为文件、目录或符号链接的路径名。第二个参数为指向包含与对象有关信息的 **stat** 结构（请参阅 *lstat*(2) ）的指针。第三个参数为提供以下附加信息的整数：

FTW_F 该对象是一个文件。

FTW_D 该对象是一个目录。

FTW_DP 该对象是一个目录，已访问其子目录。只有在指定了 **FTW_DEPTH** 时，才可以将它传递给 *fn* 。

FTW_SL 该对象是一个符号链接。只有在指定了 **FTW_PHYS** 时，才可以将它传递给 *fn* 。

FTW_SLN 该对象为不指向现有对象的符号链接。没有指定 **FTW_PHYS** 时，才可以将它传递给 *fn*。

FTW_DNR 该对象是不可读取的目录，或者在指定了 **FTW_CHDIR** 的情况下，该对象没有执行权限。不会针对该目录的任何派生对象调用函数 *fn* 。

FTW_NS 在该对象上执行 **lstat()** 或 **stat()** 失败。传递给 *fn* 的 **stat** 结构的内容不可预测。如果失败是由于缺少权限而导致的， **errno** 将设置为 [EACCES] 。如果定义了 **UNIX95** 环境，在出现权限错误后，遍历将一直继续。对于标准的 HP 行为，要使遍历继续，必须设置 **FTW_SERR** 标志。

请注意，此行为与 **ftw()** 有所不同。

第四个参数在缺省环境中与在 **UNIX95** 环境中是不相同的。在缺省环境中，它是结构 **FTW** 。在 **UNIX95** 环境中，它是指向结构 **FTW** （即 ***FTW** ）的指针。 **FTW** 包含以下成员：

```
int base;
int level;
```

base 的值是从路径名第一个字符到对象基名起始位置的偏移量；会将此路径名当作第一个参数传递给 *fn* 。 *level* 的值指示对应于遍历起始位置的深度，此处的起始层次的值为零。

实际应用信息

能以单独的线程并行执行 **ftw()** 。如果 *path* 参数是相对的（也就是说，不以“/”开头），或者设置了 **FTW_CHDIR** 标志，将序列化 **nftw()** 和 **nftw2()** 。为获得最佳的并行性，请使用绝对起始路径调用 **nftw()** ，并且不要设置 **FTW_CHDIR** 。

要使用 **UNIX95** 原型，必须定义 **UNIX95** 环境。要实现此操作，请定义 **UNIX95** 环境变量，将 **_XOPEN_SOURCE_EXTENDED** 标志当作编译程序的选项来传递，同时在路径上添加 **/usr/xpg4/bin** 。可以按如下方式实现此操作：

1. **export UNIX95=**
2. **PATH=/usr/xpg4/bin:$PATH**
3. **cc foo.c -D_XOPEN_SOURCE_EXTENDED**

nftw2() 今后将会过期。

错误

如果 **ftw()** 或 **nftw()** 失败，会将 **errno** （请参阅 *errno*(2) ）设置为下列其中一个值：

[EACCES]	如果 *path* 前缀的某个部分拒绝搜索权限，或者拒绝了 *path* 的读取权限， *fn* 将返回 **-1** ，且不重置 **errno** 。
[EINVAL]	*depth* 参数的值无效。
[ENAMETOOLONG]	指定的路径名的长度超过 **PATH_MAX** 字节；或者，当 **_POSIX_NO_TRUNC** 有效时，路径名的某个部分的长度超过 **NAME_MAX** 字节。
[ENOENT]	*path* 指向一个不存在的文件的名称，或者指向空字符串。
[ENOTDIR]	*path* 的某个组件不是目录。
[EOVERFLOW]	**struct stat** （**st_size** 或 **st_blocks** ）中的一个或多个值过大，以致于无法在要传递给 *fn* 所指向函数的结构中存储。对于 32 位应用程序，请改用 **ftw64()** 或 **nftw64()** 。

此外，如果 *fn* 指向的函数遇到系统错误，可以相应地设置 **errno** 。

警告

对于 32 位应用程序，如果文件系统使用 64 位值，则将 **st_ino** 截断至最不重要的 32 位。请注意，由于 **st_ino** 定义为无符号 long 字符， **ftw64()** 调用也会发生这种情况。

对于访问大型文件系统的 32 位应用程序，请改用 **ftw64()** 或 **nftw64()** ，以防 **stat** 结构溢出。

nftw() 包含某种递归，当应用到包含大量目录或大量文件（清除了 FTW_PHYS，并定义了 UNIX95 环境时）的文件树后，可以使用内存故障错误进行终止。

在操作过程中， **nftw()** 将使用 **malloc()** 来分配动态存储（请参阅 *malloc*(3C) ）。如果被强制终止（例如 *fn* 或中断例行程序执行了 **longjmp()** ），调用函数将没有机会释放该存储，因而在进程终止之前它一直处于已分配状态。处理中断的一种安全方法就是存储中断已发生的事实，并且在下次调用 *fn* 时尽量使它返回一个非零值。

如果起始 *path* 位于环回文件系统 (LOFS) 中，并且设置了 **FTW_MOUNT** （保存在 LOFS 中），但是清除了 **FTW_PHYS** （后接符号链接和硬链接）， **nftw()** 将无法确定引用相同 _device_ 上的文件的链接是否真正在 LOFS 内。可以使用这个特定的系数组合将某些文件报告给 *fn* ，而事实上不应该是向其报告。

过时的接口

nftw2() 今后将会过期。

作者

ftw() 、 **nftw()** 和 **nftw2()** 由 AT&T 和 HP 开发。

另请参阅

stat(2)、fgetpos64(3S)、malloc(3C)、 thread_safety(5)。

符合的标准

ftw()：AES、SVID2、SVID3、XPG2、XPG3、XPG4

名称

fwide( ) - 设置流的方向

概要

#include <stdio.h>

#include <wchar.h>

int fwide(FILE *stream, int mode);

说明

fwide() 函数确定 *stream* 指向的流的方向。如果 *mode* 大于零，则函数首次尝试使流面向宽度。如果 *mode* 小于零，则函数首次尝试使流面向字节。否则， *mode* 为零，函数不能改变流的方向。

如果流的方向已经确定，则 **fwide()** 不能更改它。由于没有返回值用来指示错误，因此希望检查错误情况的应用程序应将 **errno** 设置为 0，调用 **fwide()** ，然后检查 **errno** ，如果它为非零值，则认为已出现错误。

实际应用信息

对将 *mode* 设为 0 的 **fwide()** 的调用可以用来确定当前流的方向。

如果应用程序满足下列条件，则可以使用该函数的原型：

a. 符合 **c99** 的要求。

b. 使用值 >=500 的 **-D_XOPEN_SOURCE** 宏进行编译。

c. 使用值 >= 200112 的 **-D_POSIX_C_SOURCE** 宏进行编译。

返回值

调用以后，如果流是面向宽度的，则 **fwide()** 函数返回大于零的值；如果流是面向字节的，则此函数返回小于零的值；但是，如果流没有方向，则函数返回零。

错误

在下列情况下， **fwide()** 函数可能会失败：

[EBADF] *stream* 参数不是有效的流。

作者

fwide() 由 HP 和 Mitsubishi Electric Corporation 联合开发。

另请参阅

orientation(5)、 thread_safety(5)。

名称

fwprintf()、wprintf()、swprintf() - 输出已设置格式的宽字符输出

概要

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

int fwprintf(
        FILE *__restrict stream,
        const wchar_t *__restrict format,
        ...);

int wprintf(
        const wchar_t *__restrict format,
        ...);

int swprintf(
wchar_t *__restrict s,
size_t n,
const wchar_t *__restrict format,
...);
```

说明

fwprintf 函数可以将输出放置在指定的输出 *stream* 上。

wprintf() 函数可以将输出放置在标准的输出流 *stdout* 上。

swprintf() 函数可以将后接空宽字符的输出，放置在以 ***s;*** 开始的连续宽字符中；写入的宽字符不可超过 *n* 个，这包括始终会添加的终止空宽字符（除非 *n* 为零）。

其中每个函数都可以在格式宽字符字符串的控制下，转换、格式设置和输出其参数。 *format* 由零个或多个指令组成：普通宽字符，只能复制到输出流和转换规范，并且都可以提取零个或多个参数。

转换可以应用到参数列表中 *format* 后面的第 *n* 个参数，而不是下一个未使用的参数。在这种情况下，转换宽字符 **%** （请参阅下文）将由序列 **%***n***$** 替换，此处的 *n* 是 1 至 **{NL_ARGMAX}** 范围内的十进制整数，用来指定参数在参数列表中的位置。此功能可提供格式宽字符字符串的定义，这些字符串将根据与特定语言相适应的顺序选择参数（请参阅“举例”一节）。

在包含 **%***n***$** 形式转换规范的宽字符字符串中，可以通过格式宽字符字符串，以需要的次数引用参数列表中的已编号参数。

在包含 **%** 形式转换规范的格式宽字符字符串中，参数列表中的每个参数只能使用一次。

所有形式的 **fwprintf()** 函数都允许在以宽字符值输出的输出字符串中，插入语言相关的小数分隔符。小数分隔符在程序的语言环境（类别 **LC_NUMERIC** ）中定义。

在 POSIX 语言环境中，或者其中未定义小数分隔符的语言环境中，小数分隔符的缺省值为句点 **(.)** 。

% 宽字符或宽字符序列 **%***n***$** ，将引入每个转换规范随后出现以下序列：

1. 可修改转换规范含义的零个或多个 标记（任意顺序）。
2. 一个可选的最小 字段宽度。如果转换值的宽字符数小于 字段宽度，将按缺省在左侧用空格填充。
3. 一个可选 精度，指定针对 **d** 、 **i** 、 **o** 、 **u** 、 **x** 和 **X** 转换显示的最小位数。
4. 下列零个或多个可选字符规范：
   - 可选的 **h** ，指定后面的 **n** 转换宽字符可应用于指向 short int 类型参数的指针
   - 可选的 **h** ，指定后面的 **d** 、 **i** 、 **o** 、 **u** 、 **x** 或 **X** 转换宽字符可应用于 short int 类型或无符号 short int 类型参数（将按照整数升级的要求升级此参数，输出前其值将转换为 short int 类型或无符号 short int 类型）
   - 可选 **hh** ，指定后面的 **d** 、 **i** 、 **o** 、 **u** 、 **x** 或 **X** 转换说明符应用于带符号的 char 或无符号 char 参数（将按照整数升级的要求升级此参数，但输出前其值将转换为 short 或无符号 short 类型）
   - 可选的 **hh** ，指定后面的 **n** 转换说明符可应用于指向带符号 char 参数的指针
   - 可选的 **hL** ，指定后面的 **e** 、 **f** 或 **g** 转换宽字符可应用于类型 **extended** ，该类型在 Itanium(® 体系结构中为 80 位 IEEE-754 双精度扩展类型
   - 可选的 **j** ，指定后面的 **d** 、 **i** 、 **o** 、 **u** 、 **x** 或 **X** 转换说明符可应用于 intmax_t 或 uintmax_t 参数
   - 可选的 **j** ，指定后面的 **n** 转换说明符可应用于指向 intmax_t 参数的指针
   - 可选的 **l** （字母 L），指定后面的 **c** 转换宽字符可应用于 wint_t 参数
   - 可选的 **l** （字母 L），指定后面的 **s** 转换宽字符可应用于 wchar_t 参数
   - 可选的 **ll** （字母 LL），指定后面的 **d** 、 **i** 、 **o** 、 **u** 、 **x** 或 **X** 转换说明符可应用于 long long 或无符号 long long 参数
   - 可选的 **ll** （字母 LL），指定后面的 **n** 转换说明符可应用于指向 long long 参数的指针
   - 可选的 **t** ，指定后面的 **d** 、 **i** 、 **o** 、 **u** 、 **x** 或 **X** 转换说明符可应用于 ptrdiff_t 或相应的无符号类型参数
   - 可选的 **t** ，指定后面的 **n** 转换说明符可应用于指向 ptrdiff_t 参数的指针
   - 可选的 **z** ，指定后面的 **d** 、 **i** 、 **o** 、 **u** 、 **x** 或 **X** 转换说明符可应用于 size_t 参数或对应的有符号整型参数
   - 可选的 **z** ，指定后面的 **n** 转换说明符可应用于指向 size_t 参数的指针
5. 转换宽字符，表示要应用的转换的类型。

可以使用星号 (*) 表示字段宽度或精度，或者这两项。在这种情况下，int 类型的参数可提供 字段宽度或 精度。

指定 字段宽度、 精度或这两者的参数，必须在要转换的参数（如果有）之前以该顺序显示。负值 字段宽度可看作是 **-** 标记后接正值 字段宽度。负值 精度可看作是省略了其精度。在包含 **%***n***$** 形式转换规范的格式宽字符字符串中，可以使用序列 **%***n***$** 来表示 字段宽度或 精度，其中 *m* 是 **[1, {NL_ARGMAX}]** 范围内的十进制整数，用于指定包含 字段宽度或 精度的整数参数在参数列表中的位置（在格式参数之后），例如：

wprintf(L"%1$d:%2$. *3$d:%4$, *3$d\n", hour, min, precision, sec);

format 可能包含已编号的参数规范（即 **%***n***$** 和 ***m$** ），也可能包含未编号的参数规范（即 **%** 和 ***** ），但一般不会同时包含这两种。仅有一种例外就是将 **%%** 和 **%***n***$** 形式相混合。在格式宽字符字符串中混合已编号和未编号的参数规范，其结果是不确定的。使用已编号的参数规范时，如果要指定第 N 个参数，需要在格式宽字符字符串中指定所有排在其前面的参数，也就是从第一至第 (N-1) 个参数。

标记宽字符及其含义如下：

' 将使用千位分组宽字符来设置十进制转换（%i、%d、%u、 %g 或 %G）结果的整数部分的格式。对于其他转换，该行为是不确定的。将使用非货币分组宽字符。

- 转换结果将在字段内左对齐。如果未指定此标记，此转换将会右对齐。

+ 带符号转换的结果始终以符号（+ 或 -）开头。如果未指定此标记，只有当转换了负值时，转换才以符号开头。

space 如果带符号转换的第一个宽字符不是符号，或者带符号转换没有产生宽字符，将在结果的前面添加空格前缀。这意味着如果空格和 **+** 标记同时出现，则忽略空格标记。

此标记指定此值将转换为备用形式。对于 **o** 转换，它会提高精度（如果必要），以将结果的第一位数强制设置为 0。对于 **x** 或 **X** 转换，非零结果之前将添加 0x（或 0X）前缀。对于 **e**、 **E** 、 **f** 、 **F** 、 **g** 或 **G** 转换，即使小数分隔符后没有数字，结果仍始终会包含小数分隔符。如果没有此标记，则仅当小数分隔符后面有数字时，这些转换的结果中才会显示该小数分隔符。对于 **g** 和 **G** 转换，将不会像正常情况那样从结果中删除尾随的零。对于其他转换，该行为是不确定的。

0 对于 **d** 、 **i** 、 **o** 、 **u** 、 **x** 、 **X** 、 **e** 、 **E** 、 **f** 、 **F** 、 **g** 和 **G** 转换，将使用前导零（前面为任何符号或基数指示）填充字段宽度；但不会使用空格填充。如果同时出现 **0** 和 **-** 标记，则 **0** 标记将被忽略。对于 **d** 、 **i** 、 **o** 、 **u** 、 **x** 和 **X** 转换，如果指定了精度，将忽略 **0** 标记。如果同时出现 **0** 和 **i** 标记，则分组宽字符将在零填充的前面插入。对于其他转换，该行为是不确定的。

转换宽字符及其含义如下：

d、**i** int 参数转换为 [-]*dddd* 形式的带符号十进制。精度指定要显示的最小位数；如果要转换的值可以用更少的位数表示，将使用前导零对其扩展。缺省精度是 1。如果转换显式精度为 0 的 0 值，其结果是无宽字符。

o 无符号 int 参数转换为 *dddd* 形式的无符号八进制格式。精度指定要显示的最小位数；如果要转换的值可以用更少的位数表示，将使用前导零对其扩展。缺省精度是 1。如果转换显式精度

为 0 的 0 值，其结果是无宽字符。

u 无符号 int 参数转换为 *dddd* 形式的无符号十进制格式。精度指定要显示的最小位数；如果要转换的值可以用更少的位数表示，将使用前导零对其扩展。缺省精度是 1。如果转换显式精度为 0 的 0 值，其结果是无宽字符。

x 无符号 int 参数转换为 *dddd* 形式的无符号十六进制格式；将使用字母 **abcdef** 。精度指定要显示的数字的最小位数；如果要转换的值可以用更少的位数表示，则使用前导零对其进行扩展。缺省精度为 1。如果转换显式精度为 0 的 0 值，其结果是无宽字符。

X 除了将字母 **abcdef** 改为使用 **ABCDEF** 以外，其他行为与 **x** 转换宽字符相同。

f、**F** 双精度参数转换为 [-]*ddd.ddd* 形式的十进制表示法，其中小数分隔符后的位数等于精度规定值。当缺少精度时，将取值 6；如果精度明确为 0，则不显示 **#** 标记和小数分隔符。如果显示小数分隔符，则其前面至少会显示一位数。该值将舍入为合适的位数。 **F** 转换说明符生成“INF”、“INFINITY”或“NAN”，分别替换“inf”、“infinity”或“nan”。 **fwprintf()** 系列函数可以为 infinity 和 NaN 提供可用的宽字符字符串表示形式。

e、**E** 双精度参数转换为 [-]*d.ddde_dd* 的形式，其中小数分隔符之前有一位数（如果参数不是零，则此数不会是零），小数分隔符之后的位数等于精度；当缺少精度时，将显示为 6；如果精度为 0，则不显示 **#** 标记和小数分隔符。该值将舍入为合适的位数。 **E** 转换宽字符可以使用 **E**（而不是引入指数的 **e** ）来生成一个数字。该指数总是至少包含两位数。如果值为 0，则指数为 0。 **fwprintf()** 系列函数可以为无穷和 NaN 提供可用的宽字符串表示形式。

g、**G** 双精度参数以 **f**、**F** 或 **e** 形式转换（如果是 **G** 转换宽字符，则形式为 **E** ），并包含可以指定有效位数的精度。如果显式精度为 0，则显示为 1。所使用的形式取决于所转换的值；仅当转换所得的指数小于 -4，或者大于或等于精度时，才使用 **e** （或 **E** ）形式。将从结果的小数部分删除尾随零；仅当小数分隔符后面有数字时，才会显示小数分隔符。 **fwprintf()** 系列函数可以为 infinity 和 NaN 提供可用的宽字符字符串表示形式。

a、**A** 仅适用于基于 Itanium 系统。双精度 *arg* 转换为 **[-]0x***hrhhh***p±***d* 的形式，其中 *r* 为小数分隔符。小数分隔符之前有一位数，小数分隔符之后的位数等于精度；当缺少精度时，将为 **double** 生成 13 位，为 **extended** 生成 15 位，为 **long double** 生成 28 位，足够准确表示该值；如果精度为 0，则不显示小数分隔符。 **a** 转换使用字符 **abcdef** ， **A** 转换使用字符 **ABCDEF** 。 **A** 转换说明符可以生成一个带 **X** 和 **P** （而非 **x** 和 **p** ）的数字。指数始终至少包含一位数，以及用来表示十进制指数 2 的符合需要的更多位数。如果值为 0，则指数为 0。

c 如果不存在 **l** 限定符，int 参数将转换为宽字符（如同调用了 **btowc()** 函数一样），同时会输出生成的宽字符。否则，wint_t 参数将转换为 wchar_t，并输出生成的内容。

s 如果不存在 **l** 限定符，则参数必须是指向包含以初始切换状态开始的字符序列的字符数组的指针。如同重复调用 **mbrtowc()** 函数一样转换该数组中的字符，在转换第一个字符之前，会将 mbstate_t 对象描述的转换状态初始化为零，并且输出终止空宽字符（但不包括此项）之前的

内容。如果指定了精度，则最多只能输出所指定的宽字符。如果未指定精度，或者精度大于数组大小，则数组必须包含空宽字符。如果存在 **l** 限定符，则参数必须是指向 wchar_t 类型数组的指针。可以输出数组中终止空宽字符（但不包括此项）之前的宽字符。如果未指定精度，或者精度大于数组大小，则数组必须包含空宽字符。如果指定了精度，则最多只能输出所指定的宽字符。

p 参数必须是指向 void 的指针。指针值将以与实现相关的方式转换为可输出宽字符的序列。

n 参数必须是指向某个整数的指针，通过此次调用其中一个 **fwprintf()** 函数，到目前为止已写入输出的宽字符数目将被写入该整数。不转换任何参数。

C 与 **lc** 相同。

S 与 **ls** 相同。

% 输出 **%** 字符；不转换参数。整体转换规范必须为 **%%** 。

实际应用信息

fwprintf() 或 **wprintf()** 应用到流后，该流将面向宽字符（请参阅 *orientation*(5)）。

如果应用程序满足下列条件，则可以使用这些函数的原型：

a. 符合 **c99** 的要求。

b. 使用值 >=500 的 **-D_XOPEN_SOURCE** 宏进行编译。

c. 使用值 >= 200112 的 **-D_POSIX_C_SOURCE** 宏进行编译。

返回值

如果成功完成，这些函数会返回已传输的宽字符数，但不包括遇到输出错误时，出现 **swprintf()** 或负值而发生的终止空宽字符。

如果请求输出 **n** 个或更多宽字符， **swprintf()** 将返回负值。

错误

有关在哪些情况下 **fwprintf()** 和 **wprintf()** 将失败的信息，请参考 **fputwc()** ；并请参阅 *putwc*(3C) 。此外，在以下任何一种情况时，所有形式的 **fwprintf()** 都可能失败：

[EILSEQ] 检测到与有效字符不对应的宽字符代码。

[EINVAL] 参数不足。

[ENOMEM] 未提供足够的存储空间。

此外，在以下情况时， **wprintf()** 可能失败：

[EILSEQ] **stdout** 面向字节。

在以下情况时， **fwprintf()** 可能失败：

[EILSEQ] *stream* 指向的流面向字节。

举例

要输出语言无关的日期和时间格式，可以使用以下语句：

wprintf(format, weekday, month, day, hour, min);

在美国使用时，格式可能是指向以下宽字符字符串的指针：

L"%s, %s %d %d:%.2d"

生成消息：

Sunday, July 3, 10:02

而在德国使用时，格式可能是指向以下宽字符字符串的指针：

L"%1$s, %3$d. %2$s, %4$d:%5$.2d"

生成消息：

Sonntag, 3. Juli, 10:02

作者

fwprintf() 、 **wprintf()** 和 **swprintf()** 由 HP 和 Mitsubishi Electric Corporation 联合开发。

另请参阅

btowc(3C) 、 fputwc(3C) 、 fwscanf(3C) 、 mbrtowc(3C) 、 putwc(3C) 、 setlocale(3C) 、 orientation(5) 、 thread_safety(5)、 glossary(9)。

名称

fwscanf( )、wscanf( )、swscanf( ) - 转换已设置格式的宽字符输入

概要

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

int fwscanf(FILE *__restrict stream, const wchar_t
        *__restrict format, ... );

int wscanf(const wchar_t *__restrict format, ... );

int swscanf(const wchar_t *__restrict s, const wchar_t
        *__restrict format, ... );
```

说明

fwscanf() 函数可以从指定的输入流读取数据。

wscanf() 函数可以从标准的输入流 **stdin** 中读取数据。 **swscanf()** 函数可以从宽字符字符串 *s* 中读取数据。

每个函数都可读取宽字符，根据 *format* 解释宽字符，以及将结果存储在其参数中。每个参数都需要提供下文所述的控制宽字符字符串格式，以及一组用来指示已转换输入存储位置的指针参数。如果没有为格式提供足够的参数，结果将是不确定的。如果格式已用尽，但尚有余下的参数，则多余的参数会被计算，否则会被忽略。

转换可以应用到参数列表中格式后面的第 *n* 个参数，而不是下一个未使用的参数。在这种情况下，转换宽字符 **%**（请参阅下文）将由序列 **%***n***$** 替换，此处的 *n* 是 **[1,{NL_ARGMAX}]** 范围内的的十进制整数。此功能可提供格式宽字符字符串的定义，这些字符串将根据与指定语言相适应的顺序选择参数。在包含 **%***n***$** 形式转换规范的格式宽字符字符串中，未指明是否可以通过格式宽字符字符串，多数引用参数列表中的编号参数。

格式可能包含转换规范两种形式中的一种，即 **%** 或 **%***n***$** ，但通常不可以在单个格式宽字符字符串中混合这两种形式。仅有一种例外就是使用 **%***n***$** 混合 **%%** 或 **%*** 。

以各种形式存在的 **fwscanf()** 函数允许在编码为宽字符值的输入字符串中检测语言相关小数分隔符。小数分隔符是在程序的语言环境（类别 **LC_NUMERIC** ）中定义的。在 POSIX 语言环境中，或者在其中未定义小数分隔符的语言环境中，小数分隔符的缺省值为句点 **(.)** 。

格式是由零个或多个指令组成的宽字符字符串。每个指令由下列其中一个项组成：

- 一个或多个空白宽字符（空格、制表符、换行符、纵向制表符或换页符）；
- 一个普通宽字符（不是 **%** 和空白字符）；或者
- 一个转换规范。

% 或序列 **%***n***$** 将引入每个转换规范，然后依次显示以下内容：

- 一个可选的禁止赋值字符 ***** 。
- 一个可选的非零十进制整数，指定 *field width* 最大值。

- 一个可选的大小修饰符 **h** 、 **hh** 、 **j** 、 **l** （字母 L）、 **L** 、 **ll** （字母 LL）、 **t** 或 **z** 指示接收对象的大小。

  如果对应的参数是指向 **wchar_t** 的指针，而不是指向某个字符类型的指针，则转换宽字符 **c** 、 **s** 和 **[** 必须以 **l** （字母 L）开头。

  如果对应的参数是指向 short int（而不是 int）的指针，则转换宽字符 **d** 、 **i** 和 **n** 必须以 **h** 开头；如果是指向有符号 char 的指针，则上述转换宽字符必须以 **hh** 开头；如果是指向 intmax_t 的指针，则必须以 **j** 开头；如果是指向 long int 的指针，则必须以 **l** （字母 L）开头；如果是指向 long long 的指针，则必须以 **ll** （字母 LL）开头；如果是指向 ptrdiff_t 的指针，则必须以 **t** 开头；如果是指向 ssize_t 的指针，则必须以 **z** 开头。

  同样，如果对应的参数是指向无符号 short int（而不是无符号的 int）的指针，则转换宽字符 **o** 、 **u** 和 **x** 必须以 **h** 开头；如果对应的参数是指向无符号 char 的指针，则转换宽字符必须以 **hh** 开头；或者如果是指向 uintmax_t 的指针，则必须以 **j** 开头；如果是指向无符号 long int 的指针，则必须以 **l** （字母 L）开头；如果是指向无符号 long long 的指针，则必须以 **ll** （字母 LL）开头；如果是指向无符号 ptrdiff_t 的指针，则必须以 **t** 开头；如果是指向 size_t 的指针，则必须以 **z** 开头。

  如果对应的参数是指向 double（而不是 float）的指针，则转换宽字符 **e** 、 **f** 和 **g** 必须以 **l** （字母 L）开头；如果是指向 long double 的指针，则必须以 **L** 开头。如果 **h** 、 **hh** 、 **j** 、 **l** （字母 L）、 **ll** （字母 LL）、 **L** 、 **t** 或 **z** 与其他任何转换宽字符一同显示，则该行为是不确定的。

- 一个转换宽字符，指定要应用的转换的类型。下文介绍了有效的转换宽字符。

fwscanf() 函数可以依次执行格式的每个指令。如果某个指令失败，如下文详述，该函数将会返回。失败可描述为输入失败（由于输入字节不可用），或匹配失败（由于输入不当）。

通过读取输入，直到没有可供读取的有效输入，或者一直读取到不是空白宽字符且尚未读取的第一个宽字符，来执行由一个或多个空白宽字符组成的指令。

作为普通宽字符的指令将按以下方式执行。下一个宽字符将从输入中读取，并且需要与构成指令的宽字符进行比较；如果比较结果显示两者不相同，则指令失败，同时不读取有差异的后续宽字符。

作为转换规范的指令将按下文所述，为每个转换宽字符定义一组匹配的输入序列。转换规范按以下步骤执行：

跳过输入的空白宽字符（由 **iswspace()** 指定），除非转换规范包含 **[** 、 **c** 或 *n* 转换字符。

从输入读取项目，除非转换规范包含 **n** 转换宽字符。输入项目定义为输入宽字符的最长序列，不超过指定的 *field width* ，该值是某个匹配序列的初始部分。不读取输入项目后的第一个宽字符（如果有）。如果输入项目的长度为 0，转换规范的执行将会失败，这种情况属于匹配失败。但如果有文件结束标记、编码错误或读取错误阻止通过 *stream* 输入，则这种情况属于输入失败。如果不是 **%** 转换宽字符的情况（或者 **%***n* 转换规范、输入宽字符计数的情况），输入项目将转换为与转换宽字符适应的类型。如果输入项目不是匹配序列，转换规范的执行将会失败；这种情况属于匹配失败。除非 ***** 指示了禁止赋值，否则转换结果将放置在第一个参数指向的对象中，该参数的前面是一个格式参数，如果转换规范是由 **%** 引入的，或者位于第 n 个参数（由宽字符序列 **%***n***$** 引入时），则该格式参数尚未接收到转换结果。如果此对象不包含适当的类型，或者无法在提供的空间内表示转换结果，则该行为是不确定的。

以下转换宽字符有效：

d 与有选择性地加符号的十进制整数匹配，其格式与 **wcstol()** 主题序列要求的格式相同，并且基参数为值 10。没有大小修饰符时，对应的参数必须是指向 int 的指针。

i 与有选择性地加符号的十进制整数匹配，其格式与 **wcstol()** 主题序列要求的格式相同，并且基参数为值 0。没有大小修饰符时，对应的参数必须是指向 int 的指针。

o 与有选择性地加符号的八进制整数匹配，其格式与 **wcstoul()** 主题序列要求的格式相同，并且基参数为值 8。没有大小修饰符时，对应的参数必须是指向无符号 int 的指针。

u 与有选择性地加符号的十进制整数匹配，其格式与 **wcstoul()** 主题序列要求的格式相同，并且基参数为值 10。没有大小修饰符时，对应的参数必须是指向无符号 int 的指针。

x 与有选择性地加符号的十六进制整数匹配，其格式与 **wcstoul()** 主题序列要求的格式相同，并且基参数为值 16。没有大小修饰符时，对应的参数必须是指向无符号 int 的指针。

a,**e**,**f**,**g** 与有选择性地加符号的浮点数，其格式与 **wcstod()** 主题序列要求的格式相同。没有大小修饰符时，对应的参数必须是指向 float 的指针。

如果 **fwprintf()** 系列函数为无穷和 NaN（以浮点格式编码的 7858 符号实体）生成了字符型字符串表示形式，以支持 ANSI/IEEE Std 754:1985 标准， **fwscanf()** 系列函数将把它们识别为输入。

s 与非空白宽字符序列匹配。如果不存在 **l** （字母 L）限定符，则会按重复调用 **wcrtomb()** 函数一样的方式转换输入字段中的字符，在转换第一个宽字符之前，会将 **mbstate_t** 对象描述的转换状态初始化为零。对应的参数必须是指向字符数组的指针，而此字符数组的大小足以接受序列和自动添加的结尾空字符。

否则，对应的参数必须为指向 **wchar_t** 数组的指针，而此数组的大小足以接受序列和自动添加的结尾空宽字符。

[与一组要求的宽字符（扫描集）中的宽字符非空序列匹配。如果不存在 **l** （字母 L）限定符，则会按重复调用 **wcrtomb()** 函数一样的方式转换输入字段中的宽字符，在转换第一个宽字符之前，会将 **mbstate_t** 对象描述的转换状态初始化为零。对应的参数必须是指向字符数组的指针，而此字符数组的大小足以接受序列和自动添加的结尾空字符。

如果存在 **l** （字母 L）限定符，对应的参数必须是指向 **wchar_t** 数组的指针，而此数组的大小足以接受序列和自动添加的结尾空宽字符。

转换规范包括格式字符串中的所有后续宽字符，而格式字符串包括的范围直至匹配的右方括号 (**]**)。方括号之间的宽字符（扫描列表）由扫描集组成，除非左方括号后面的宽字符是抑扬重音符号 (^)，在这种情况下，扫描集将包含不会在抑扬重音符号和右方括号之间的扫描列表中显示的所有宽字符。如果转换规范以 **[]** 或 **[^]** 开头，则扫描列表将包含右方括号，下一个右方括号成为结束转换规范的匹配右方括号；否则，第一个右方括号将结束转换规范。如果 **-** 位于扫描列表中，并且不是第一个、第二个和最后一个宽字符，（此时第一个宽字符为 ^ ），则该行为与实现相关。

c 与 *field width* 指定的编号（如果转换规范中不存在字段宽度，则此编号为 1）的宽字符序列匹配。如果不存在 **l**（字母 L）限定符，则会按重复调用 **wcrtomb()** 函数一样的方式转换输入字段中的宽字符，在转换第一个宽字符之前，会将 **mbstate_t** 对象描述的转换状态初始化为零。对应的参数必须是指向字符数组的指针，而此字符数组的大小足以接受序列。未添加空字符。否则，对应的参数必须是指向 **wchar_t** 数组的指针，而此数组的大小足以接受序列。不添加空宽字符。

p 与实现相关的序列集匹配，必须与对应 **fwprintf()** 函数的 **%p** 转换生成的序列集相同。对应参数必须是指向 void 的指针。如果输入项目是在同一程序执行过程中较早转换的值，生成的指针将与该值进行同等性比较；否则， **%p** 转换行为不可预测。

n 不消耗任何输入。对应的参数必须是指向整数的指针，通过此次调用 **fwscanf()** 函数，到目前为止已经从输入读取的宽字符数目将被写入该整数。执行 ***%n*** 转换规范不会递增函数执行完成时返回的赋值计数。

C 与 **lc** 相同。

S 与 **ls** 相同。

% 与单个 **%** 匹配；不发生转换或赋值。完整的转换规范必须是 **%%** 。

如果转换规范无效，则该行为是不确定的。

转换字符 **A** 、 **E** 、 **F** 、 **G** 和 **X** 也是有效的，其行为分别与 **a** 、 **e** 、 **f** 、 **g** 和 **x** 相同。

如果输入期间遇到文件结束标记，转换将会终止。如果在读取与当前转换规范（ ***%n*** 除外）匹配的任何宽字符（前导空白字符除外，因为这是允许的）之前，发生了文件结束错误，执行当前转换规范将以输入失败的理由终止。另外，如果执行当前转换规范不是以匹配失败的理由终止，执行以下转换规范（如果有）将以输入失败的理由终止。

到达 **swscanf()** 中的字符串结尾就等同于遇到 **fwscanf()** 的文件结束标记。

如果转换由于某个冲突输入而终止，将不读取输入中的此违规输入。除非转换规范已将其匹配，否则不读取任何尾随空白字符（包括换行符）。文字匹配和禁止赋值是否成功只能直接通过 ***%n*** 转换规范确定。

fwscanf() 和 **wscanf()** 函数可以为与流关联的文件的 **st_atime** 字段做更新标记。通过可返回以前调用 **ungetc()** 后未提供的数据的流，第一次成功执行 **fgetc()** 、 **fgetwc()** 、 **fgets()** 、 **fgetws()** 、 **fread()** 、 **getc()** 、 **getwc()** 、 **getchar()** 、 **getwchar()** 、 **gets()** 、 **fscanf()** 或 **fwscanf()** 后，将为 **st_atime** 字段做更新标记。

实际应用信息

fwscanf() 或 **wscanf()** 应用到流后，该流将面向宽字符（请参阅 *orientation*(5)）。

在包含 **%** 形式转换规范的格式字符串中，参数列表中的每个参数只能确切地使用一次。

如果应用程序满足下列条件，则可以使用这些函数的原型：

a. 符合 **c99** 的要求。

b. 使用值 >=500 的 **-D_XOPEN_SOURCE** 宏进行编译。

c. 使用值 >= 200112 的 **-D_POSIX_C_SOURCE** 宏进行编译。

返回值

成功完成后，这些函数将返回已成功匹配和赋值的输入项目数；如果过早出现匹配失败，这个数可能为 0。如果输入在第一次匹配失败或转换之前结束，将返回 **EOF** 。如果发生读取错误，将设置流的错误指示符，并返回 **EOF** ，同时设置 **errno** 以指明错误。

错误

有关在哪些情况下 **fwscanf()** 函数将会（或可能）失败的信息，请参考 **fgetwc()** 。此外，在以下情况时， **fwscanf()** 可能失败：

[EILSEQ] 输入字节序列不形成有效的字符。

[EINVAL] 参数不足。

[ENOMEM] 未提供足够的存储空间。

此外，在以下情况时， **fwscanf()** 可能失败：

[EILSEQ] *stream* 指向的流面向字节。

此外，在以下情况时， **wscanf()** 可能失败：

[EILSEQ] **stdin** 面向字节。

举例

调用：

```
int i, n; float x; char name[50];
n = wscanf(L"%d%f%s", &i, &x, name);
```

带输入行：

```
25 54.32E-1 Hamster
```

将为 *n* 赋值 3，为 *i* 赋值 25，为 *x* 赋值 5.432，名称将包含字符串 **Hamster** 。

调用：

```
int i; float x; char name[50];
(void)wscanf(L"%2d%f%*d %[0123456789]", &i, &x, name);
```

带输入：

```
56789 0123 56a72
```

将为 *i* 赋值 56，为 *x* 赋值 789.0，跳过 0123，并将字符串 56 放入 *name* 。下一次调用 **getchar()** 将返回字符 **a** 。

作者

fwscanf() 、 **wscanf()** 和 **swscanf()** 由 HP 和 Mitsubishi Electric Corporation 联合开发。

另请参阅

fwprintf(3C)、 getwc(3C)、 setlocale(3C)、 wcstod(3C)、 wcstol(3C)、 wcrtomb(3C)、 wcstoul(3C)、 langinfo(5)、 orientation(5)、 thread_safety(5)、 glossary(9)。

名称

getaddrinfo()、 getnameinfo()、 freeaddrinfo()、 gai_strerror() - 获取主机名和地址条目

概要

```
#include <sys/socket.h>
#include <netdb.h>

int getaddrinfo(const char *hostname, const char *servname,
        const struct addrinfo *hints, struct addrinfo **res);

int getnameinfo(const struct sockaddr *sa, size_t salen,
        char *host, size_t hostlen,
        char *serv, size_t servlen, int flags);

 _XOPEN_SOURCE_EXTENDED

int getnameinfo(const struct sockaddr *sa, socklen_t salen,
        char *host, socklen_t hostlen,
        char *serv, socklen_t servlen, int flags);

void freeaddrinfo(struct addrinfo *ai);

char *gai_strerror(int encode);

_XOPEN_SOURCE_EXTENDED

const char *gai_strerror(int encode);
```

说明

getaddrinfo()

主机名-地址转换是使用 **getaddrinfo()** 函数，通过与协议无关的方式完成的。

getaddrinfo() 函数将返回整数。输入参数 *hostname* 和 *servname* 是指向以 null 结尾的字符串或 NULL 的指针。这两个参数之一或全部必须是非空指针。

在正常的客户端方案中，将同时指定 *hostname* 和 *servname* 。非 NULL *hostname* 字符串可以是主机名，也可以是数字形式的主机地址字符串（点分十进制 IPv4 地址或十六进制 IPv6 地址）。非 NULL *servname* 字符串可以是服务名，也可以是十进制端口号。但是，在服务器方案中，只指定 *servname* 。

第三个输入参数是指向在 **<netdb.h>** 中，按如下方式定义的 **addrinfo** 类型结构的指针：

```
struct addrinfo {
int    ai_flags;
int    ai_family;
int    ai_socktype;
int    ai_protocol;
size_t  ai_addrlen;
```

```
char   *ai_canonname;
struct sockaddr  *ai_addr;
struct addrinfo  *ai_next;
};
```

该结构成员包括:

ai_flags 用于设置 IPv4 地址或 IPv6 地址的套接字地址结构的标记。

如果 **AI_PASSIVE** 位为 **set** ，则 **getaddrinfo()** 函数返回的套接字地址结构将用作 **bind()** 函数调用的参数。如果设置了此位，并且 **getaddrinfo()** 函数的 *hostname* 参数为 NULL 指针，则对于 IPv4 地址和 IPv6 地址，套接字地址结构的 IP 地址部分将分别设置为 **INADDR_ANY** 和 **IN6ADDR_ANY_INIT** 。

如果不设置 **AI_PASSIVE** ，则返回的套接字地址结构将用作 **connect()** 、 **sendto()** 或 **sendmsg()** 函数的参数。在这种情况下，如果 *hostname* 参数为 NULL 指针，则套接字地址的 IP 地址部分将设置为 **loopback** 地址。

如果已设置 **AI_CANONNAME** 位，返回成功时， **getaddrinfo()** 将返回第一个 **addrinfo** 结构（包含指定主机规范名称的 NULL 结尾字符串）的 **ai_canonname** 成员。

如果使用 **AF_INET6** 的 **ai_family** 值指定 **AI_V4MAPPED** 标记，在没有找到任何匹配的 IPv6 地址的情况下， **getaddrinfo()** 将返回由 IPv4 映射的 IPv6 地址。如果 **ai_family** 不等于 **AF_INET6** ， **getaddrinfo()** 将忽略 **AI_V4MAPPED** 标记。

如果 **AI_ALL** 标记与 **AI_V4MAPPED** 标记一同使用， **getaddrinfo()** 将返回所有匹配的 IPv6 和 IPv4 地址。 **getaddrinfo()** 将忽略不包含 **AI_V4MAPPED** 标记的 **AI_ALL** 标记。

如果不指定 **AI_ADDRCONFIG** 标记，则只有在本地系统中配置了 IPv4 地址时，才返回 IPv4 地址；同时只有在本地系统中配置了 IPv6 地址时，才返回 IPv6 地址。在这种情况下，不会将环回地址视为有效的配置地址。

ai_family 调用方接受的协议系列。如果此成员设置为 **PF_UNSPEC** ，则调用方将接受任何协议系列。如果调用方只处理 IPv4 堆栈，而不处理 IPv6 堆栈，则必须将 **ai_family** 设置为 **PF_INET** 。

ai_protocol 调用方支持的协议。如果 **ai_protocol** 设置为 **0** ，则调用方将接受任何协议。

ai_socktype 调用方支持的套接字类型。如果 **ai_socktype** 设置为 **0** ，则调用方将接受任何套接字类型。但是，如果调用方只处理 TCP 而不处理 UDP，则必须将 **ai_socktype** 设置为 **SOCK_STREAM** 。

ai_addrlen IPv4 或 IPv6 地址的长度，单位为字节。

ai_canonname 主机的规范名称。

ai_addr 主机的二进制地址。

ai_next 链接列表中的下一个 **addrinfo** 结构。

以上参数是可选的。如果调用方希望提供其支持的诸如套接字类型、协议系列的相关信息，则可以使用 **addrinfo** 结构来指定。此信息传递到 **getaddrinfo()** 后，除 **ai_flags** 、 **ai_family** 、 **ai_socktype** 和 **ai_protocol** 以外的所有字段必须设置为 零，或 NULL 指针。

成功返回后， ****res** 将保存指向一个或多个 **addrinfo** 结构的链接列表的指针。调用方可以遵循 **ai_next** 指针，处理此列表中的每个 **addrinfo** 结构，直到遇到空指针。在每个返回的 **addrinfo** 结构中，将 **ai_family** 、 **ai_socktype** 和 **ai_protocol** 三个成员用作 **socket()** 函数调用的参数。 **ai_addr** 成员将指向一个套接字地址结构，其长度由 **ai_addrlen** 成员指定。

如果结果为 **success** ，则 **getaddrinfo()** 函数的返回值为 **0** ，否则为 非零错误代码。以下为 **getaddrinfo()** 指定的非零错误代码，这些代码在 **<netdb.h>** 中定义：

EAI_ADDRFAMILY 不支持 *hostname* 的地址系列。

EAI_AGAIN 名称解析中出现暂时性失败。

EAI_BADFLAGS **ai_flags** 的值无效。

EAI_FAIL 名称解析中出现不可恢复的失败。

EAI_OVERFLOW 参数缓冲区溢出。

EAI_FAMILY 不支持 **ai_family** 。

EAI_MEMORY 内存分配失败。

EAI_NODATA 没有与 *hostname* 关联的地址。

EAI_NONAME 未提供 *hostname* 和 *servname* ，或者未知。

EAI_SERVICE *servname* 不支持 **ai_socktype** 。

EAI_SOCKTYPE 不支持 **ai_socktype** 。

EAI_SYSTEM errno 中返回的系统错误。

freeaddrinfo()

getaddrinfo() 返回的以下所有信息都是动态分配的： **addrinfo** 结构、套接字地址结构和 **addrinfo** 结构指向的规范的主机名字符串。要将这些信息返回系统，需要调用函数 **freeaddrinfo()** ：

```
#include <sys/socket.h>
#include <netdb.h>

void freeaddrinfo(struct addrinfo *ai);
```

ai 参数指向的 **addrinfo** 结构将与此结构指向的任何动态存储一起释放。该操作在遇到 NULL **ai_next** 指针之前将不断地重复执行。

gai_strerror()

为帮助应用程序根据 **getaddrinfo()** 返回的 **EAI_***xxx* 代码输出错误消息，可定义 **gai_strerror** 函数。

```
#include <sys/socket.h>
#include <netdb.h>

const char *gai_strerror(int ecode);
```

该参数是以前定义的 **EAI_***xxx* 值之一，返回值指向描述错误的字符串。如果该参数不是 **EAI_***xxx* 值之一，函数仍然会返回指向某个字符串的指针，该字符串的内容指示未知错误。

getnameinfo()

getnameinfo() 函数用于查找指定了二进制地址和端口的主机名和服务名。该函数按以下方式定义：

```
#include <sys/socket.h>
#include <netdb.h>

int getnameinfo(const struct sockaddr *sa, socklen_t salen,
        char *host, socklen_t hostlen, char *serv,
        socklen_t servlen, int flags);
```

该函数在 DNS 和系统特定的数据库中查找调用方提供的 IP 地址和端口号，并在调用方提供的缓冲区中返回这两者的文本字符串。

函数返回零值表示成功完成；返回非零值则表示失败。

如果地址是未指定的 IPv6 地址 (::)，则可能需要采取下列操作：

getnameinfo() 返回 **EAI_NONAME** （如果已设置 **NI_NAMEREQD** 标记）。

getnameinfo() 返回成功（如果已设置 **NI_NAMEREQD** 标记）。

host 参数包含数字形式的 IPv6 地址，且 **getnameinfo()** 不会执行对 IPv6 地址的查找。

第一个参数 **sa** ，将指向保存 IP 地址和端口号的 **sockaddr_in** 结构（适用于 IPv4）或 **sockaddr_in6** 结构（适用于 IPv6）。 **salen** 参数可指定 **sockaddr_in** 或 **sockaddr_in6** 结构的长度。

该函数在 **host** 参数指向的缓冲区中返回与 IP 地址关联的主机名。调用方可通过 **hostlen** 参数提供该缓冲区的大小。将在 **serv** 指向的缓冲区中返回与端口号关联的服务名， **servlen** 参数指定了该缓冲区的长度。调用方可以通过为 **hostlen** 或 **servlen** 参数提供零值，指定不返回两个字符串中的一个。否则，调用方必须提供其大小足以保存主机名和服务名（包括终止空字符）的缓冲区。

遗憾的是，多数系统没有提供可以指定完全限定域名或服务名最大大小的常量。因此，为帮助应用程序为这两个返回字符串分配缓冲区，将在 **<netdb.h>** 中定义以下常量：

```
#define NI_MAXHOST  1025
#define NI_MAXSERV   32
```

事实上，在 DNS/BIND 的最新版本中，第一个值 **NI_MAXHOST** 定义为头文件 **<arpa/nameser.h>** 中的常量 **MAXDNAME** （BIND 的早期版本将此常量定义为 256）。

getnameinfo() 函数的最后一个参数是可以更改此函数缺省操作的标记。缺省情况下，将在 DNS 和返回值中查找主机的完全限定域名 (FQDN)。如果标记位 **NI_NOFQDN** 为 **set** ，则只返回本地主机 FQDN 的主机名部分。

如果标记位 **NI_NUMERICHOST** 为 **set** ，或者无法在 DNS 中找到主机名，则返回数字形式的主机地址，而不是其名称（通过调用 **inet_ntop()** ，而不是 **gethostbyaddr()** ）。

如果标记位 **NI_NAMEREQD** 为 **set** ，并且无法在 DNS 中找到主机名，则返回一个错误。

如果标记位 **NI_NUMERICSERV** 为 **set** ，则返回数字形式的服务地址（端口号），而不是其名称。要支持很多命令提供的 **-n** 标记，需要具有两个 **NI_NUMERIC***xxx* 标记。

第五个标记位 **NI_DGRAM** 指定服务是数据报服务，并且使得用第二个参数“udp”（而不是缺省的“tcp”）来调用 **getservbyport()** 。对于可以为 UDP 和 TCP 提供不同服务的少数几个端口 (512-514) 来说，这是必要的。

将在 **<netdb.h>** 中定义这些 **NI_***xxx* 标记，同时还定义已经为 **getaddrinfo()** 定义的 **AI_***xxx* 标记。

基于名称服务交换的操作

getnameinfo() 和 **getaddrinfo()** 库例行程序将内部调用名称服务交换，以访问在 **/etc/nsswitch.conf** 文件（请参阅 *nsswitch.conf*(4) ）中为名称（或地址）解析配置的 **ipnodes** 数据库查找策略，以及服务（或端口）解析的 **services** 数据库查找策略。查找策略定义可用于解析的支持名称服务的顺序和条件。如果与所有 **ipnodes** 指令通信后没有收集地址，并且调用方将 **ai_family** 设置为 **AF_INET** ，或者使用 **AF_INET6** 的 **ai_family** 将 **ai_flags** 设置为 **AI_V4MAPPED** ， **getaddrinfo()/getnameinfo()** 将使用 **/etc/nsswitch.conf** 文件中的 **hosts** 指令来解析主机名（或地址）。在这种情况下，如果 **hosts** 指令主机名（或地址）解析失败，所返回的错误编号就是 **hosts** 指令查找返回的错误编号。下面介绍域名服务器和非服务器模式这两种名称服务的操作:

域名服务器操作

如果本地系统配置为使用用于名称（或地址）解析的 BIND 名称服务器 **named** （请参阅 *named*(1M) 和 *resolver*(4) ），则函数 **getnameinfo()/getaddrinfo()** 可以从名称服务器中检索主机信息。匹配主机名时将忽略字母大小写。例如，域名 **berkeley.edu** 、 **Berkeley.EDU** 和 **BERKELEY.EDU** 与名称服务器数据库中的同一条目 **berkeley.edu** 相匹配。如果将 **hosts** 指令用于主机名（或地址）解析，可能会由于使用为 **hosts** 指令指定的源执行其他查找，而监测到延迟。

非服务器操作

在名称/地址解析期间，如果为文件解析配置了数据库， **getnameinfo()** 和 **getaddrinfo()** 将把 **/etc/hosts** 文件用于解析。同样地，如果为文件解析配置了 **services** 数据库，将把 **/etc/services** 文件用于解析。下面列出了将 **/etc/hosts** 文件用于名称/地址解析时，函数 **getnameinfo()** 和 **getaddrinfo()** 搜索地址所使用的方法:

getnameinfo() 按顺序搜索 **/etc/hosts** 文件，直到找到与 *src* 参数匹配的地址，或者遇到文件结束标记。

getaddrinfo()	按顺序搜索 **/etc/hosts** 文件，直到找到与 *name* 参数匹配的主机名（正式名称或别名），或者遇到文件结束标记。匹配主机名时将忽略字母大小写。

警告

过时的接口

为支持现有的应用程序而提供了以下接口，将来的发行版可能会删除这些接口。

```
struct hostent *getipnodebyname(const char *name,
                    int af,
                    int flags,
                    int *error_num);

int getipnodebyaddr(const void *src,
                size_t len,
                int af,
                int *error_num);
```

为了实现名称/地址解析，新应用程序必须使用 API **getaddrinfo()** 和 **getnameinfo()** 。

过时的指令

ipnodes 指令供 **getaddrinfo()** 和 **getnameinfo()** API使用，将来的 HP-UX 发行版可能不再支持该指令。为了最大程度地减小对应用程序的影响，建议 **hosts** 与 **ipnodes** 指令使用相同的配置。

作者

getaddrinfo() 由 HP 开发。

文件

/etc/hosts

/etc/services

另请参阅

gethostent(3N)、 resolver(3N)、 nsswitch.conf(4)。

名称

getauduser - 检索当前进程的负责用户

概要

#include <sys/audit.h>

int getauduser(char *user, char *stime, size_t usize, size_t
tsize);

说明

getauduser() 可检索当前进程的负责用户，并将此信息保存到 *user* 指向的缓冲区中。如果有身份验证的时间信息，它还会对其进行检索，并将其保存到 *stime* 指向的缓冲区中。调用方必须将 *usize* 和 *tsize* 设置为 *user* 和 *stime* 缓冲区的大小。请注意，这两个缓冲区始终以空字符结尾，即使这会导致某些数据丢失，也是如此。

使用此例行程序的程序必须使用 **-lsec** 进行编译。

安全性限制

此调用要求用户是超级用户，或拥有 **SELFAUDIT** 权限。有关详细信息，请参阅 *privileges*(5) 。

返回值

getauduser() 可返回下列值:

n 成功完成。 *n* 是检索到的登录名的长度（不包括空字节）。如果 *n* 小于 *usize* ，则表示检索到的登录名未被截断。否则，表示 *usize* -1 个字节的登录名被复制到 *user* ，并且 *user* 以空字符结尾。调用方需要使用更大的缓冲区重试 **getauduser()** 。

-1 失败。设置 **errno** 以指明错误。

错误

如果 **getauduser** 失败，则将 **errno** 设置为下列值之一:

[EPERM] 调用方不是超级用户或有权限的进程。

[EINVAL] *stime* 缓冲区大小小于 **MAX_TIME_LEN+1** 。请参阅 **<sys/audit.h>** 。

[EILSEQ] 无法识别的 **audit tag** 。

举例

```
char user[256], time[MAX_TIME_LEN+1];
int n;

if ((n=getauduser(user, time, sizeof(user), sizeof(time))) == -1) {
   non_overflow_errors();
} else if (n >= sizeof(user)) {
   overflow_error();
}
```

作者

getauduser() 由 HP 开发。

针对此系统调用而选择返回值和空字符填充语义的原因在于 C 语言程序中缓冲区溢出的可能性最小。如果出现在 OpenBSD 中，则这些语义将由 C99 的 **snprintf()** ，以及 **strlcpy()** 和 **strlcat()** 调用。

另请参阅

getaudid(2)、 setaudid(2)、 setauduser(3)、 audit(5)、 privileges(5)。

名称

getbegyx、getmaxyx、getparyx — 获取其他光标和窗口坐标

概要

#include <curses.h>

void getbegyx(WINDOW *_win_, int _y_, int _x_);

void getmaxyx(WINDOW *_win_, int _y_, int _x_);

void getparyx(WINDOW *_win_, int _y_, int _x_);

说明

getbegyx() 宏将在 *y* 和 *x* 中存储所指定窗口的原点的绝对屏幕坐标。

getmaxyx() 宏将在 *y* 中存储所指定窗口的行数，在 *x* 中存储窗口的列数。

如果所指定的窗口为子窗口， **getparyx()** 宏将在 *y* 和 *x* 中存储窗口原点相对于其父窗口位置的坐标。否则，*y* 和 *x* 中将存储 −1。

返回值

未定义返回值。

错误

未定义错误。

实际应用信息

这些接口为宏命令，且在 *y* 和 *x* 参数前不能使用“**&**”。

另请参阅

getyx(3X)、 <curses.h>。

历史变更记录

在 X/Open Curses 第 4 期中首次发布。

名称

getbootpent()、putbootpent()、setbootpent()、endbootpent()、 parse_bp_htype()、parse_bp_haddr()、parse_bp_iaddr() - 获取或放置 bootptab 条目

概要

```
#include <bootpent.h>

int getbootpent (struct bootpent **bootpent);

int setbootpent (const char *path);

int endbootpent (void);

int putbootpent (
    struct bootpent *bootpent,
    int numfields,
    FILE * bootpfile
);

int parse_bp_htype (const char *source);

int parse_bp_haddr (
    char **source,
    int htype,
    unsigned char *result,
    unsigned int *bytes
);

int parse_bp_iaddr (
    char **source,
    unsigned long *result
);
```

备注

这些函数驻留在 libdc.a 中，可以使用 **-ldc** 选项将它们与 **ld** 或 **cc** 命令链接。

说明

这些函数帮助程序以一次一个条目的方式读取或修改 bootptab （**bootpd** 控制）文件。 **getbootpent()** 可以在 **/etc/bootptab** 文件或指定给 **setbootpent()** 的备选文件中定位条目，并返回指向 **struct bootpent** 类型对象数组的指针，该类型可以将条目分割成带有前置行、嵌入行或注释（文本）行的单独数据字段。

bootpent 结构是在 **<bootpent.h>** 中定义的，它包括以下成员：

```
int   bp_type;   /* BP_DATA, BP_COMMENT, BP_BLANK  */
char  *bp_text;  /* one field or one comment line  */
```

文件还定义以下数据类型和常量：

```
typedef struct bootpent *bpp_t;

#define BP_NULLP  ((bpp_t) 0)
#define BP_SIZE   (sizeof (struct bootpent))

#define MAXHADDRLEN             6

#define HTYPE_UNKNOWN           0   /* 0 bytes  */
#define HTYPE_ETHERNET          1   /* 6 bytes  */
#define HTYPE_EXP_ETHERNET      2   /* 1 byte   */
#define HTYPE_AX25              3   /* 0 bytes  */
#define HTYPE_PRONET            4   /* 1 byte   */
#define HTYPE_CHAOS             5   /* 0 bytes  */
#define HTYPE_IEEE802           6   /* 6 bytes  */
#define HTYPE_ARCNET            7   /* 0 bytes  */

#define MAXHTYPES               7
```

以下字段定义一节描述了这些字段。每个函数的用途如下。

getbootpent()

首次调用 **getbootpent()** 后，它将返回指向 **bootpent** 结构数组的指针，并在该结构中返回元素数。该数组在包含前导行、嵌入行或注释行的 **/etc/bootptab** 文件（或通过调用 **setbootpent()** 指定的备选文件）中占用第一个条目。以后每次调用都将返回指向文件中下一个条目的指针，因此可以通过连续调用来搜索整个文件。

如果当前内存中没有任何文件， **getbootpent()** 将在读取 **/etc/bootptab** 文件后再执行工作。

返回的数组保存在静态空间（malloc'd 内存）中，而下一次调用后将覆盖此静态空间（因此以前返回的指针无效）。但是，每个数组元素的 **bp_text** 指针都指向文件的内存中副本内的文本。下一次调用（或更改文件本身）时不会更改此文本。因此，可以按以下举例中说明的方式，通过复制条目数组将其保存。数据在下一次调用 **setbootpent()** 或 **endbootpent()** 之前一直有效。

如果在最后一个条目之后提供了注释，会单独将它们当作无数据部分的条目返回。

setbootpent()

打开指定的文件以供 **getbootpent()** 读取，将副本读入内存，及关闭文件（如同负面影响解除了文件的锁定；请参阅 *lockf*(2) ）。如果给定的 *path* 是一个空指针或空字符串， **setbootpent()** 将打开和读取 **/etc/bootptab** 。

如果 **setbootpent()** 打开（或者由 **getbootpent()** 隐式打开）的最后一个文件为 **/etc/bootptab** ，那么在针对相同文件后续调用 **setbootpent()** 时，会将文件恢复到最初的情况，以便能看到该文件最近发生的任何更改，而不必首先调用 **endbootpent()**。

endbootpent() 释放 **setbootpent()** 或 **getbootpent()** 打开的最后一个文件的内存中副本。

putbootpent() （在 *bootpfile* 指定流的当前位置）写入包含一个文件条目及其前导、嵌入或注释行的 **bootpent** 结构指定数组（包括 *numfields* 数组的全部元素）的 ASCII 等效项。以规范的格式写入条目，也就是说，条目名和每个数据字段都列在单独的行上，每个数据字段都以一个制表符开头，除最后一行以外的其他每行都以“:\”结尾。如果 *numfields* 小于或等于零，将不写入任何内容。

parse_bp_htype() 以与 **bootpd** 相同的方式，将主机地址类型由字符串格式转换为数字格式 (**HTYPE_***)。

parse_bp_haddr() 以与 **bootpd** 相同的方式（指定主地址类型 (**HTYPE_***) ），将主机（硬件、链接级别）地址由字符串格式转换为二进制格式。将修改调用程序的 *result* （必须是至少包含 **MAXHADDRLEN** 个元素的数组），以保存主机地址的二进制值；同时修改 **bytes** ，以指定得到的地址的字节长度。可以通过这种方式比较两个主机地址，而无论字符串表示形式如何。将修改 *source* ，使之指向已分析地址后面的第一个字符。

parse_bp_iaddr() 以与 **bootpd** 相同的方式，将 Internet 地址由字符串格式转换为二进制格式。可以通过这种方式比较两个 Internet 地址，与字符串表示形式无关。将修改调用程序的 *result* ，以保存 Internet 地址的二进制值。将修改 *source* ，使之指向已分析地址后面的第一个字符。

字段定义

如果 **bootpent.bp_type** 是 **BP_DATA** ，则关联的文本是当前条目中的一个字段，不是名称字段，就是其中一个标记字段。将忽略且不返回空标记字段（一行两个冒号）。

如果 **bootpent.bp_type** 是 **BP_COMMENT** 或 **BP_BLANK** ，则关联的文本是文件中的一个注释行或者空行，该行要么在当前条目的开头，要么嵌在当前条目中，其前面为后接反斜杠的数据行。该文本与文件中显示的文本完全相同，包括空行上显示的空白字符，并且没有任何尾随的换行符。

返回的数组元素的顺序与文件中数据字段及注释行显示的顺序相同。

条目字段字符串的格式如下：

tag [**@**] [**="***value***"**]

其中删除了两边的任何空白字符（有关完整说明，请参阅 *bootpd*(1M) ）。双引号和反斜杠可以在字段字符串的任何位置出现。

（由使用 **tc** 字段的其他条目引用的）模板条目并不是特殊条目。可以像其他条目一样对其进行管理。调用程序负责正确地管理字段顺序、 **tc** 字段，以及用来覆盖原先字段值的“@”字段。

返回值

如果成功完成， **getbootpent()** 将返回有效数组元素数（一个或多个）。到达输入文件结尾时返回零。如果无法打开或关闭文件，将返回 -1。如果遇到内存分配、映射或读取错误，将返回 -2。

setbootpent() 如果成功地打开和读取指定的或缺省的文件，将返回零。如果无法打开或关闭文件，将返回 -1。如

果遇到内存分配、映射或读取错误，将返回 -2。

endbootpent() 如果成功释放当前打开文件的内存，将返回零。如果当前没有文件，将返回 -1。如果无法释放当前文件的内存，将返回 -2。

如果成功地将条目写入指定的文件， **putbootpent()** 将返回零，并清除 **ferror()** 指示（请参阅 *ferror*(3S) ）。否则，将返回非零值，同时设置 **ferror()** 。

对于上述所有情况，如果失败是由于系统调用失败导致的，系统返回的 **errno** 值对已调用函数的返回值来说是有效的。

如果硬件类型字符串不可识别， **parse_bp_htype()** 将返回 **HTYPE_UNKNOWN** 。

成功时， **parse_bp_haddr()** 将返回零；否则，在出现分析错误、无效的 *htype* 或地址长度未知的主机地址类型时，将返回非零值；将修改 *source* ，使之指向第一个非法字符（字符串过短时为 NUL ）。未修改调用方的 *bytes* 值，但可能已更改 *result* 。

成功时， **parse_bp_iaddr()** 将返回零；否则返回非零值，同时修改 *source* ，使之指向第一个非法字符（字符串为空时为 NUL ）。

举例

以下代码段将 **/etc/bootptab** 中的所有数据和注释复制到文件的临时副本。它还附带地将数据条目转换为规范格式，并以标准输出格式输出每个复制条目的第一个字段（假如条目不是以连续行开头，则应该为字段名）。

```
#include <bootpent.h>

FILE *newfilep;   /* to write temp file */
bpp_t bp;         /* read from file */
int field;        /* current field number */
int fields;       /* total in array for one entry */

if ((newfilep = fopen ("/tmp/bootptab", "w")) == (FILE *) NULL)
{
   (handle error)
}

while ((fields = getbootpent (&bp)) > 0)
{
   for (field = 0; field < fields; ++field)
   {
      if ((bp[field].bp_type) == BP_DATA)
      {
         (void) puts (bp[field].bp_text);
         break;
```

```
        }
    }

    if (putbootpent (bp, fields, newfilep))
    {
        (handle error)
    }
}

if (fields < 0)    /* error reading file */
{
    (handle error)
}

if (endbootpent())
{
    (handle error)
}

if (fclose (newfilep))
{
    (handle error)
}
```

以下代码段将保存 **getbootpent()** 返回的 bootptab 条目副本。

```
#include <malloc.h>
#include <string.h>
#include <bootpent.h>

bpp_t bpnew;
unsigned size;

size = fields *BP_SIZE;

if ((bpnew = (bpp_t) malloc (size)) == BP_NULLP)
{
    (handle error)
}
```

```
(void) memcpy ((void *)bpnew, (void *)bp, size);
```

警告

在多线程应用程序中，这些函数是不安全的。

调用 **setbootpent()** 将解除所打开文件的任何锁定。

作者

这些函数由 HP 开发。

文件

/etc/bootptab **bootpd** 的控制文件

另请参阅

bootpd(1M)、 errno(2)、 lockf(2)、 ferror(3S)、 fopen(3S)、 malloc(3C)。

名称

getc()、 getc_unlocked()、 getchar()、 getchar_unlocked()、 fgetc()、 getw()、 getw_unlocked() - 从流文件获取字符或单词

概要

```
#include <stdio.h>

int getc(FILE *stream);

int getc_unlocked(FILE *stream);

int getchar(void);

int getchar_unlocked(void);

int fgetc(FILE *stream);

int getw(FILE *stream);
```

过时的接口

```
int getw_unlocked(FILE *stream);
```

说明

getc() 返回指定的输入 *stream* 中的下一个字符（即字节），作为转换为整数的无符号字符。如果已定义，它还可以将 *stream* 中的文件指针向前移动一个字符。 **getchar()** 定义为 **getc(stdin)** 。 **getc()** 和 **getchar()** 既可以定义为宏，也可以定义为函数。

fgetc() 与 **getc()** 相似，但它是一个函数，而不是宏。 **fgetc()** 的运行速度比 **getc()** 慢，但每次调用所占用的空间较少，并且可以将其名称作为参数传递给函数。

getw() 返回指定的输入 *stream* 中的下一个单词（即 C 语言中的 **int** ）。如果已定义， **getw()** 可以递增关联的文件指针，使之指向下一个单词。单词的大小是整数的大小，该值随计算机的不同而异。 **getw()** 假定文件中不存在特殊的调整。

getc_unlocked() 和 **getchar_unlocked()** 分别与 **getc()** 和 **getchar()** 类似，不同之处在于，前者不能针对多线程应用程序执行 *stream* 的任何内部锁定。

过时的接口

getw_unlocked() 从流文件获取字符或单词。

实际应用信息

getc_unlocked() 和 **getchar_unlocked()** 接口应该由多线程应用程序使用，并且这些应用程序已使用 **flockfile()** 来获取 *stream* （请参阅 *flockfile*(3S) ）的互斥锁。

getc() 、 **getc_unlocked()** 、 **getchar()** 、 **getchar_unlocked()** 、 **fgetc()** 或 **getw()** 应用到流后，该流将面向字节（请参阅 *orientation*(5) ）。

返回值

如果成功完成， **getc()** 、 **getc_unlocked()** 、 **getchar()** 、 **getchar_unlocked()** 和 **fgetc()** 将返回 *stream* （如果是 **getchar()** 和 **getchar_unlocked()** ，则为 **stdin** ）指向的输入流中的下一个字节。如果流位于文件结束的位置，则设置流的文件结束指示符，并返回 EOF。如果发生读取错误，将设置流的错误指示符，并设置 **errno** 以指明错误，同时返回 EOF。

到达文件结束标记后，当与打开的流相对应的文件得到扩展时，对这些函数的所有后续调用都将获得成功，而且文件结束标记指示符将保持设定状态。但是，在 UNIX2003 标准环境中（请参阅 *standards*(5) ），这些函数将返回 EOF，并且文件结束标记指示符保持设定状态。

如果成功完成， **getw()** 和 **getw_unlocked()** 将返回 *stream* 指向的输入流中的下一个单词。如果流位于文件结束的位置，则设置流的文件结束指示符，同时 **getw()** 和 **getw_unlocked()** 将返回 EOF。如果发生读取错误，将设置流的错误指示符， **getw()** 和 **getw_unlocked()** 将返回 EOF，同时设置 **errno** 以指明错误。

可以使用 **ferror()** 和 **feof()** 来区分错误条件和文件结束条件。

错误

如果需要将数据读入 *stream* 的缓冲区， **getc()** 、 **getc_unlocked()** 、 **getchar()** 、 **getchar_unlocked()** 、 **getw()** 、 **getw_unlocked()** 和 **fgetc()** 将会失败，此外：

[EAGAIN] 为文件描述符的基础 *stream* 设置了 **O_NONBLOCK** 标记，并且将在读取操作中延迟进程。

[EBADF] 文件描述符的基础 *stream* 不是为读取而打开的有效文件描述符。

[EINTR] 由于接收到信号，或未传输数据，抑或实现未报告此文件的部分传输情况，因此终止了读取操作。

[EIO] 发生了物理 I/O 错误，或者该进程是后台进程的成员，并尝试从其控制终端读取数据，该进程忽略或阻塞 **SIGTTIN** 信号，或者进程的进程组被孤立。

可以通过基础 **read()** 函数（请参阅 *read*(2) ）设置其他的 **errno** 值。

警告

getc() 和 **getchar()** 既可以实现为库函数，也可以实现为宏。 **<stdio.h>** 中定义了宏版本，缺省情况下使用这些版本。要获取库函数，可以使用 **#undef** 删除宏定义，如果是在 ANSI-C 模式下进行编译的，还可以使用圆括号括起函数名，或者使用函数地址。以下示例说明了每种方法：

```
#include <stdio.h>
#undef getc
   ...
main()
{
   int (*get_char()) ();
   ...
   return_val=getc(c,fd);
```

```
    ...
    return_val=(getc)(c,fd1);
    ...
    get_char = getchar;
};
```

如果将 **getc()** 、 **getc_unlocked()** 、 **getchar()** 、 **getchar_unlocked()** 或 **fgetc()** 返回的整数值存储到字符变量后，再根据整数变量 **EOF** 执行比较，比较将永远不能成功，这是因为扩展到整数的字符的符号扩展部分与计算机有关。

宏版本的 **getc()** 将通过负面影响不正确地处理 *stream* 参数。特别是， **getc(*f++)** 不能敏感地工作。因此，应改用函数版本的 **getc()** 或 **fgetc()** 。

由于单词长度和字节排序上可能存在差异，使用 **putw()** 编写的文件将与计算机有关，并且不同的处理器上的 **getw()** 可能无法读取。

重入接口

如果加入 **<stdio.h>** 之前定义了 **_REENTRANT** ，将按缺省情况使用 **getc()** 和 **getchar()** 的锁定版本的库函数。

getw_unlocked() 是为了与现有 DCE 应用程序兼容而受到支持的已过时接口。新的多线程应用程序应该使用 **getw()** 。

另请参阅

read(2)、 fclose(3S)、 ferror(3S)、 flockfile(3S)、 fopen(3S)、 fread(3S)、 gets(3S)、 putc(3S)、 scanf(3S)、 orientation(5)、 standards(5)、 thread_safety(5)。

符合的标准

getc()：AES、SVID2、SVID3、XPG2、XPG3、XPG4、FIPS 151-2、POSIX.1、ANSI C
getc_unlocked()：POSIX.1C
fgetc()：AES、SVID2、SVID3、XPG2、XPG3、XPG4、XPG4.2、FIPS 151-2、POSIX.1、ANSI C
getchar()：AES、SVID2、SVID3、XPG2、XPG3、XPG4、FIPS 151-2、POSIX.1、ANSI C
getchar_unlocked()：POSIX.1C
getw()：AES、SVID2、SVID3、XPG2、XPG3、XPG4

名称

getcchar — 从 **cchar_t** 获取宽字符串和呈现

概要

#include <curses.h>

int getcchar(const cchar_t *_wcval_, wchar_t *_wch_, attr_t *_attrs_,
short *_color_pair_, void *_opts_);

说明

当 *wch* 不是一个空指针时， **getcchar()** 函数从 *wcval* 定义的 **cchar_t** 提取信息，将字符属性存储在 *attrs* 指向的对象中，将颜色对存储在 *color_pair* 指向的对象中，而将 *wcval* 引用的宽字符字符串存储在 *wch* 指向的数组中。

当 *wch* 是一个空指针时， **getcchar()** 获取 *wcval* 指向的对象中的宽字符的数目，而没有更改 *attrs* 或 *color_pair* 指向的对象。

opts 参数是为本文档将来的版本中的定义而保留的。目前，应用程序必须提供空指针作为 *opts*。

返回值

当 *wch* 是一个空指针时， **getcchar()** 返回 *wcval* 引用的字符的数目，其中包括空终止符。

当 *wch* 不是一个空指针时， **getcchar()** 在成功完成后返回 OK，否则返回 ERR。

错误

没有定义任何错误。

实际应用信息

wcval 参数可以是通过调用 **setcchar()** 生成的值，也可以由具有 **cchar_t** 输出参数的函数生成的值。如果 *wcval* 是通过任何其他方式构造的，则效果是未定义的。

另请参阅

attroff(3X)、 can_change_color(3X)、 setcchar(3X)、 <curses.h>。

历史变更记录

在 X/Open Curses 第 4 期中首次发布。

名称

getch、wgetch、mvgetch 和 mvwgetch — 从终端获取单字节字符

概要

#include <curses.h>

int getch(void);

int mvgetch(int *y*, **int** *x*);

int mvwgetch(WINDOW ****win*, **int *y*, **int** *x*);

**int wgetch(WINDOW ****win*);

说明

这些函数从与当前或指定的窗口相关的终端读取单字节字符。如果输入不是单字节字符，则结果是未指定的。如果启用了 **keypad()** ，则这些函数通过返回 **<curses.h>** 中定义的相应 KEY_ 值来响应按功能键的操作。

终端输入的处理遵循 *curses_intro*(3X) 中的输入处理定义的一般规则。

如果启用了回响，则会回响字符，好像它是作为 **insch()** 的输入参数提供的一样，下列字符除外：

<退格>、<左箭头> 和当前清除字符：
: 根据 **specialchars** 中的指定解释输入，然后删除光标所在的最终位置处的字符，就好像调用 **delch()** 一样，不同之处在于，如果光标原来就在此行的第一列，则会向用户发出警报，就好像调用了 **beep()** 一样。

功能键
: 更改用户，就好像调用了 **beep()** 一样。功能键的信息不返回到调用方。

如果当前的窗口或指定的窗口不是一个板，并且已经由于最后的刷新操作而移动或修改，则在读取另一个字符之前它将会刷新。

返回值

一旦成功完成，这些函数就会返回单字节字符、KEY_ 键或 ERR。

如果在无延迟的模式下或没有数据可用，则返回 ERR。

错误

没有定义任何错误。

实际应用信息

应用程序不应该自己将 escape 键定义为单字符函数。

在使用这些函数时，则不应该同时 nocbreak 模式 (**nocbreak()**) 和回响模式 (**echo()**) 。根据输入字符时终端的状态，程序可能产生不需要的结果。

另请参阅

cbreak(3X)、 doupdate(3X)、 insch(3X)、 curses_intro(3X)，请参阅输入处理、<curses.h>。

历史变更记录

在 X/Open Curses 第 2 期中首次发布。

X/Open Curses 第 **4** 期

为清楚起见，重新编写了此条目。 **getch()** 的参数列表函数显式声明为 **void**。

名称

getclock - 获取系统级时钟的当前值

概要

#include <sys/timers.h>

int getclock(int clock_type, struct timespec *tp);

说明

getclock() 函数获取指定系统级时钟 **clock_type** 的当前值 **tp** 。

getclock() 支持定义在 **<sys/timers.h>** 中的 **TIMEOFDAY** 的 **clock_type** ，它表示系统一天中的时间。对于此时钟， **getclock()** 返回的值表示自 Epoch 的时间。

返回值

成功完成后， **getclock()** 返回零；否则，它返回 −1，并将 **errno** 设置为指示出错。

错误

如果遇到下列情况， **getclock()** 将失败：

[EINVAL] **clock_type** 不指定已知系统级时钟。

[EIO] 访问时钟设备时所出现的错误。

文件

/usr/include/sys/timers.h

另请参阅

clocks(2)、clock_gettime(2)、gettimer(3C)、setclock(3C)、 thread_safety(5)。

符合的标准

getclock()：AES

名称

getcwd( ) - 获取当前工作目录的路径名

概要

#include <unistd.h>

char *getcwd(char *buf, size_t size);

说明

getcwd() 函数将当前工作目录的绝对路径名置于由 *buf* 所指定的数组中，并返回 *buf* 。 *size* 的数值至少比所返回路径名的长度大 1。

如果 *buf* 是一个空指针， **getcwd()** 利用 **malloc()** 获取空间的 *size* 字节（请参阅 *malloc*(3C) ）。在此情况下，由 **getcwd()** 返回的指针可以用作随后调用 **free()** 的参数（请参阅 *malloc*(3C) ）。并不推荐以 *buf* 作为空指针调用 **getcwd()** ，因为在 HP-UX 操作系统的未来版本中可能会删除这一功能。

返回值

在成功完成后， **getcwd()** 返回指向当前目录路径名的指针。否则，如果 *size* 不是足够大，或者如果在一个低级函数中发生错误，则其返回带有 NULL ， 并设置 **errno** 。

错误

如果遇到以下任何一种情况，则 **getcwd()** 失败：

[EINVAL] *size* 参数为零。

[ERANGE] *size* 参数大于零，但小于路径名的长度。

[ENAMETOOLONG] 指定路径名的长度超出 **PATH_MAX+1** 字节，或者路径名的一个组成部分的长度超出 **NAME_MAX** 字节，同时 **_POSIX_NO_TRUNC** 有效。

如果遇到以下任何一种情况，则 **getcwd()** 可能失败：

[EACCES] 拒绝了对路径名的某部分的读取或搜索权限。

[EFAULT] *buf* 指向为此过程所分配地址空间的外部。 **getcwd()** 可能并不总会检测这一错误。

[ENOMEM] **malloc()** 未能提供内存的 *size* 字节。

示例

```
#include <unistd.h>
#include <limits.h>

char *cwd;
char buf[PATH_MAX + 1 ];
                . . .
if ((cwd = getcwd(buf, PATH_MAX+1)) == NULL) {
```

```
        perror("pwd");
        exit(1);
    }
    puts(cwd);
```

作者

getcwd() 由 AT&T 开发。

另请参阅

pwd(1)、malloc(3C)、 thread_safety(5)。

符合的标准

getcwd()：AES、SVID2、SVID3、XPG2、XPG3、XPG4、FIPS 151-2 和 POSIX.1

名称

getdate() - 转换用户格式的日期和时间

概要

#include <time.h>

struct tm *getdate(const char *string);

过时的接口

int getdate_r(const char *string, struct tm *result, int *errnum);

说明

getdate() 函数可以将 *string* 指向的用户可定义的日期和（或）时间规范转换为 **struct tm** 。结构声明位于 **<time.h>** 头文件（请参阅 *ctime*(3C) ）中。

用户提供的模板用于分析和解释输入字符串。模板是用户创建的文本文件，可通过环境变量 **DATEMSK** 进行标识。应设置 **DATEMSK** 以指示模板文件的完整路径名。可以使用模板中与输入规范匹配的第一行来解释和转换为内部时间格式。如果成功完成， **getdate()** 将返回指向 **struct tm** 的指针；否则返回 NULL，并设置符号 **getdate_err** 以指明错误。

支持以下字符描述符：

%%	与 **%** 相同
%a	缩写的一周中的某一天的名称
%A	完整的一周中的某一天的名称
%b	缩写的月份名称
%B	完整的月份名称
%c	语言环境的适当日期和时间表示形式
%C	世纪编号（00 到 99；允许但不要求使用前导零）
%d	一个月中的某一天（01 至 31；可以选择使用前导 0）
%e	与 **%d** 相同
%D	**%m/%d/%y** 形式的日期
%h	缩写的月份名称
%H	小时（00 至 23）
%I	小时（01 至 12）
%m	月份（01 至 12）
%M	分钟（00 至 59）
%n	与 **\n** 相同
%p	语言环境中与 AM 或 PM 等效的表示形式
%r	**%I:%M:%S %p** 形式的时间
%R	**%H:%M** 形式的时间
%S	秒（00 至 61）

%t	插入一个制表符
%T	**%H:%M:%S** 形式的时间
%w	一周中某一天的编号（星期日 = 0，以此类推，星期六 = 6）
%x	语言环境的适当日期表示形式
%X	语言环境的适当时间表示形式
%y	不带世纪的年份（00 至 99）。如果输入内容为 69-99，则假定为二十世纪；如果输入内容为 00-68，则假定为二十一世纪。
%Y	*ccyy* 形式的年份（如 1986）
%Z	时区名称，如果不存在时区则没有字符。如果 **%Z** 提供的时区不同于 **getdate()** 所需的时区，将返回无效规范的错误。 **getdate()** 可通过 **TZ** 环境变量来计算所需的时区。

月份名称和一周中的某一天的名称可以包括大小写字母的任意组合。用户可通过设置 **LC_TIME** 类别（请参阅 *setlocale*(3C) ）来请求输入日期或时间规范使用特定的语言。

对于允许使用前导零的描述符，这些前导零是可选的。但是，这些描述符占用的位数包括前导零在内，不得超过两位。无论是模板文件中还是 *string* 中存在的额外空白字符都将被忽略。

包含不受支持的字段描述符的 **%c** 、 **%x** 和 **%X** 字段描述符同样也不受支持。

以下示例说明了模板的可能的内容：

```
%m
%A %B %d, %Y, %H:%M:%S
%A
%B
%m/%d/%y %I %p
%d,%m,%Y %H:%M
at %A the %dst of %B in %Y
run job at %I %p, %B %dnd
%A den %d. %B %Y %H.%M Uhr
```

以下为上述模板的有效输入规范的示例：

```
getdate("10/1/87 4 PM");
getdate("Friday");
getdate("Friday September 18, 1987, 10:30:30");
getdate("24,9,1986 10:30");
getdate("at monday the 1st of december in 1986");
getdate("run job at 3 PM, december 2nd");
```

如果将 **LC_TIME** 类别设置为德国的语言环境，并且该语言环境将 **freitag** 用作一周中的某一天的名称，将 **oktober** 用作月份名称，那么以下规范有效：

getdate("freitag den 10. oktober 1986 10.30 Uhr");

此示例说明如何在模板中定义本地日期和时间规范：

调用	模板中的行
getdate("11/27/86")	***%m/%d/%y***
getdate("27.11.86")	***%d.%m.%y***
getdate("86-11-27")	***%y-%m-%d***
getdate("Friday 12:00:00")	***%A %H:%M:%S***

将输入规范转换为内部格式时，可应用以下规则：

- 在只是给定了一周中的某一天的情况下，如果给定日等于当日，则假定为当日；如果给定日小于当日，则假定为下一周。
- 在只是给定了月份的情况下，如果给定月等于当月，则假定为当月；如果给定月小于当月，并且没有给定年份，则假定为下一年（如果没有给定日，则假定为当月的第一天）。
- 如果没有给定小时、分钟和秒，将假定为当前小时、分钟和秒。
- 在没有给定日期的情况下，如果给定小时大于当前小时，则假定为当日；如果给定小时小于当前小时，则假定为明日。

以下示例可帮助说明上述规则，此示例假设当前日期为 **Mon Sep 22 12:19:47 EDT 1986** ， **LC_TIME** 类别设置为缺省的 **C** 语言环境。

输入	模板中的行	日期
Mon	***%a***	Mon Sep 22 12:19:47 EDT 1986
Sun	***%a***	Sun Sep 28 12:19:47 EDT 1986
Fri	***%a***	Fri Sep 26 12:19:47 EDT 1986
September	***%B***	Mon Sep 1 12:19:47 EDT 1986
January	***%B***	Thu Jan 1 12:19:47 EST 1987
December	***%B***	Mon Dec 1 12:19:47 EST 1986
Sep Mon	***%b %a***	Mon Sep 1 12:19:47 EDT 1986
Jan Fri	***%b %a***	Fri Jan 2 12:19:47 EST 1987
Dec Mon	***%b %a***	Mon Dec 1 12:19:47 EST 1986
Jan Wed 1989	***%b %a %Y***	Wed Jan 4 12:19:47 EST 1989
Fri 9	***%a %H***	Fri Sep 26 09:00:00 EDT 1986
Feb 10:30	***%b %H:%S***	Sun Feb 1 10:30:00 EST 1987
10:30	***%H:%M***	Tue Sep 23 10:30:00 EDT 1986
13:30	***%H:%M***	Mon Sep 22 13:30:00 EDT 1986

过时的接口

getdate_r() 可转换用户格式的日期和时间。另请参阅“警告”一节。

错误

如果失败， **getdate()** 将返回 NULL，并设置符号 **getdate_err** 以指明错误。

以下为 **getdate_err** 的设置及其解释的完整列表：

1 **DATEMSK** 环境变量为 null 或不可预测，

2 无法打开要读取的模板文件，

3 无法获取文件状态信息，

4 模板文件不是常规文件，

5 读取模板文件时遇到错误，

6 内存分配失败（没有足够的可用内存），

7 模板内没有与输入匹配的行，

8 输入规范无效。例如，输入了 February 31；或者 32 位 HP-UX 中 **time_t** 数据类型无法表示指定的时间（表示为 Tuesday January 19 03:14:07 UTC, 2038），或者超过 64 位 HP-UX 支持的最大日期（为 Friday December 31 23:59:59 UTC, 9999）。

警告

getdate() 的返回值指向其内容将被同一线程每次执行调用后覆盖的数据。

getdate_r() 是为了与现有 DCE 应用程序兼容而受到支持的已过时接口。新的多线程应用程序应该使用 **getdate()**。

另请参阅

ctime(3C)、ctype(3C)、setlocale(3C)、strftime(3C)、 thread_safety(5)。

名称

getdiskbyname( ) - 按名称获取磁盘说明

概要

```
#include <disktab.h>

struct disktab *getdiskbyname(const char *name);
```

过时的接口

```
int getdiskbyname_r(
    const char *name,
    struct disktab *result,
    char *buffer,
    int buflen);
```

说明

getdiskbyname() 将获取磁盘名称（例如 hp7959B）并返回一个指向描述磁盘几何信息及标准磁盘分区表的结构的指针。所有信息都是从 disktab 数据库文件中获取的（请参阅 *disktab*(4)）。

disktab 结构的内容中包括以下成员。请注意，此列表中的位置与结构中的顺序之间不一定要有相关性。

```
char    *d_name;            /* drive name */
char    *d_type;            /* drive type */
int     d_secsize;          /* sector size in bytes */
int     d_ntracks;          /* # tracks/cylinder */
int     d_nsectors;         /* # sectors/track */
int     d_ncylinders;       /* # cylinders */
int     d_rpm;              /* revolutions/minute */
struct  partition {

    int     p_size;         /* #sectors in partition */
    short   p_bsize;        /* block size in bytes */
    short   p_fsize;        /* frag size in bytes */

} d_partitions[NSECTIONS];
```

常量 **NSECTIONS** 是在 **<disktab.h>**中定义的。

过时的接口

getdiskbyname_r() 将按名称获取磁盘说明。

诊断信息

出现错误或在 disktab 数据库文件中未找到 *name* 的情况下，将返回 NULL （空）指针。

警告

getdiskbyname() 的返回值将指向其内容被每次调用所覆盖的数据。 **getdiskbyname_r()** 是一个过时的接口。之所以还提供对该接口的支持，只是为了实现与现有 DCE 应用程序的兼容。新的多线程应用程序应使用 **getdiskbyname()** 。

作者

getdiskbyname() 由 HP 和加州大学伯克利分校联合开发。

另请参阅

disktab(4)、 thread_safety(5)。

名称

getdvagent()、getdvagnam()、setdvagent()、enddvagent()、putdvagnam()、copydvagent() - 操作受信任系统的设备分配数据库条目

概要

```
#include <sys/types.h>
#include <hpsecurity.h>
#include <prot.h>

struct dev_asg *getdvagent( );

struct dev_asg *getdvagnam(const char *name);

void setdvagent( );

void enddvagent( );

int putdvagnam(const char *name, struct dev_asg *dv);

struct dev_asg *copydvagent(struct dev_asg *dv);
```

说明

getdvagent 、 **getdvagnam** 和 **copydvagent** 中的每一个都可以返回指向带有以下结构的对象的指针，该结构包含设备分配数据库中某个条目的组成字段。每个数据库条目的返回形式为 **<prot.h>** 头文件中声明的 **dev_asg** 结构：

```
struct dev_field {
    char     *fd_name;          /* external name */
    char     **fd_devs;         /* device list */
    mask_t   fd_type[1];        /* tape, printer, terminal */
    char     **fd_users;        /* authorized user list */
};

/* Device Assignment Database entry */

#define AUTH_DEV_TYPE           "device type"
#define AUTH_DEV_PRINTER        0
#define AUTH_DEV_TERMINAL       1
#define AUTH_DEV_TAPE           2
#define AUTH_DEV_REMOTE         3
#define AUTH_MAX_DEV_TYPE       3
#define AUTH_DEV_TYPE_SIZE      (WORD_OF_BIT (AUTH_MAX_DEV_TYPE) + 1)

/* this structure tells which of the corresponding fields
 * in dev_field are valid (filled).
 */
struct dev_flag {
```

```
    unsigned short
        fg_name  : 1,
        fg_devs  : 1,
        fg_type  : 1,
        fg_users : 1,
    ;
};

struct dev_asg {
    struct dev_field ufld;
    struct dev_flag  uflg;
    struct dev_field sfld;
    struct dev_flag  sflg;
};
```

设备分配数据库可存储与用户验证和同义有关的设备特性。在支持网络连接的系统上，设备分配数据库可存储关于 hosts 启动连接的信息。

每个条目包含与终端控制数据库交叉引用的 *name* ，以及 *devices* 列表，列表中的每个项都是对应于该设备的路径名。通过此列表，受信任系统的设备分配软件在重新赋值设备时，能使对设备的所有引用失效。该列表是由字符型字符串指针组成的表格，其中的最后一个条目为 NULL 指针。

fd_users 是一个指针，它指向引用了用户允许的访问权限的字符型字符串指针的以 null 结尾的表格。

对于支持网络连接的受信任系统版本，设备名可以是 12 个字符的主机名，其中前 8 个字符为设备的 ASCII 十六进制地址，最后 4 个字符为 ASCII 零。例如，对于 Internet 地址为 129.75.0.3 的主机，其设备名为 814b00030000。尾随的四个零是为了与终端集线器上的端口兼容而提供的。 SAM API 支持将主机名转换为设备名。因此，可以在主机以及直接连接的终端上强制执行敏感度级别范围和用户授权列表。

首次调用 **getdvagent** 后，将返回指向第一个设备分配条目的指针。此后，返回指向下一个条目的指针，这样便可以通过连续调用来搜索数据库。 **getdvagnam** 从数据库的起始位置开始搜索，直到找到包含与 *name* 匹配的设备名的条目，然后返回指向该条目的指针。如果读取时遇到文件结束标记或错误，这些函数将返回 NULL 指针。 **copydvagent** 将所引用的设备分配结构和字段复制到新赋值的数据区。由于 **getdvagent** 、 **getdvagnam** 和 **putdvagent** 在访问数据库时可重复使用静态结构，如果要再次使用这些例行程序，则必须保存任何条目的值。可以使用 *free* （请参阅 *malloc*(3C) ）释放 **copydvagent** 返回的 *dev_asg* 结构。

调用 **setdvagent** 能够将设备分配数据库设置回到第一个条目，以便反复搜索数据库。 **enddvagent** 可释放所有内存，及关闭用于支持这些例行程序的所有文件。

putdvagnam 向数据库重写或添加条目。如果某个条目的 *fd_name* 字段与 *name* 参数匹配，则使用 *dv* 结构的内容替换该条目。否则，会将该条目添加到数据库。

注释

必须使用 **-lsec** 编译使用此例行程序的程序。

实际应用信息

在多线程的应用程序中，仅从一个专用线程调用这些例行程序是安全的。这些例行程序不是 POSIX.1c 异步取消安全的，也不是异步信号安全的。

返回值

getdvagent 和 **getdvagnam** 在成功时返回指向静态结构的指针，在失败时返回 NULL 指针。此静态结构将被 **getdvagent** 、 **getdvagnam** 和 **putdvagnam** 覆盖。

putdvagnam 在成功时返回 1，在失败时返回 0。

copydvagent 在成功时返回指向新赋值的结构的指针，如果出现内存分配错误，则返回 NULL 指针。

警告

此例行程序返回的结构包含指向字符型字符串和列表的 指针，而不是独立的实体。要保存返回的结构，必须使用 **copydvagent** 而不是结构赋值。

getdvagent 和 **getdvagnam** 返回的值将引用调用这些例行程序后所覆盖的结构。要检索某个条目，应先对其修改，再在数据库中替换，再使用 **copydvagent** 复制此条目，然后将修改的缓冲区提供给 **putdvagent** 。

HP-UX 11i v3 是支持受信任系统功能的最后一个版本。

文件

/tcb/files/devassign	设备分配数据库

另请参阅

authcap(4)。

名称

getenv( ) - 返回环境名的值

概要

#include <stdlib.h>

char *getenv(const char *name);

说明

getenv() 将搜索环境列表（请参阅 *environ*(5) ）以查找 *name*=*value* 形式的字符串，并返回指向当前环境中 *value* 的指针（如果存在此类字符串），否则就会返回一个 NULL （空）指针。 *name* 可以是所需的名称、以 null 结尾的字符串或是 *name*=*value* 形式的值（这种情况下，说明 **getenv()** 使用了 **=** 左侧的部分作为搜索键值）。

警告

getenv() 将返回一个指向可被后续调用覆盖的静态数据的指针。

外部语言环境影响

语言环境

LC_CTYPE 类别用于确定将 *name* 中的字符解释为单字节字符和（或）多字节字符。

国际代码集支持

支持单字节字符代码集和多字节字符代码集。

另请参阅

exec(2)、putenv(3C)、environ(5)、 thread_safety(5)。

符合的标准

getenv()：AES、SVID2、SVID3、XPG2、XPG3、XPG4、FIPS 151-2、POSIX.1、POSIX.2、ANSI C

名称

get_expiration_time() - 向当前的绝对系统时间添加特定的时间间隔

概要

```
#include <time.h>

int get_expiration_time(
        struct timespec *delta,
        struct timespec *abstime

);
```

说明

get_expiration_time() 函数向当前的绝对系统时间添加特定的时间间隔，并返回新的绝对时间。这一新的绝对时间在调用 *pthread_cond_timedwait*(3T) 时用作到期时间。

delta 参数表示添加到当前系统时间的秒和纳秒数。从此函数返回时， *abstime* 参数包含调用 *pthread_cond_timedwait*(3T) 时使用的绝对系统时间。

参数

delta 添加到当前系统时间的秒和纳秒数。

abstime 向当前的绝对系统时间添加 *delta* 后，输出绝对系统时间的参数。

返回值

成功完成后， **get_expiration_time()** 返回零。否则，返回错误编号，以表明错误（未设置 **errno** 变量）。

错误

如果出现下面任何一种情况， **get_expiration_time()** 函数都会返回相应的错误编号:

[EINVAL] *delta* 或者 *abstime* 指定的值是无效的。

作者

get_expiration_time() 由 X/Open 开发。

另请参阅

pthread_cond_timedwait(3T)。

符合的标准

get_expiration_time() : X/Open。

名称

getfsent( )、getfsspec( )、getfsfile( )、getfstype( )、setfsent( )、endfsent( ) - 获取文件系统描述符文件条目

概要

```
#include <checklist.h>

struct checklist *getfsent(void);

struct checklist *getfsspec(const char *spec);

struct checklist *getfsfile(const char *file);

struct checklist *getfstype(const char *type);

int setfsent(void);

int endfsent(void);
```

备注:

仅针对与 4.2 BSD 的兼容性才包括这些例行程序。为获得最大的可移植性或改进的功能，新应用程序应使用 *getmntent*(3X) 库例行程序。

说明

getfsent() 、 **getfsspec()** 、 **getfsfile()** 和 **getfstype()** 都返回一个指向对象的指针，此对象具有以下结构，即在 **/etc/fstab** 文件中的一个行中包含断开的字段。此结构是在 **<checklist.h>** 标题文件中声明的:

```
struct checklist {
   char   *fs_spec;    /* special file name */
   char   *fs_bspec;   /* block special file name */
   char   *fs_dir;     /* file sys directory name */
   char   *fs_type;    /* type: ro, rw, sw, xx */
   int    fs_passno;   /* fsck pass number */
   int    fs_freq;     /* backup frequency */
};
```

此字段具有 *fstab*(4) 中描述的意义。如果阻止特定的文件名，则文件系统目录名和类型并不都在 **/etc/fstab** 的关联行中定义，这些例行程序返回的指针指向 **fs_bspec** 、 **fs_dir** 和 **fs_type** 字段中的 NULL 。如果在此行中未提供传递数值或备份频率字段，则这些例行程序将在相应的结构成员中返回 −1 。 **fs_freq** 被保留以备将来之用。

getfsent() 读取文件的下一行，必要时打开文件。

setfsent() 打开并回卷文件。

endfsent() 关闭文件。

getfsspec() 从文件的开头按顺序搜索，直至找到匹配的特殊文件名，或者直至遇到 EOF 为止。

getfsfile() 从文件的开头按顺序搜索，直至找到匹配的文件系统文件名，或者直至遇到 EOF 为止。

getfstype() 从文件的开头按顺序搜索，直至找到匹配的文件系统类型字段，或者直至遇到

EOF 为止。

诊断信息

在遇到 EOF 、无效项或错误时将返回空指针。

警告

由于所有信息都包含在静态区域中，必须复制它才能保存。

过时的接口

getfsent()、 **getfsspec()**、 **getfsfile()**、 **getfstype()**、 **setfsent()** 和 **endfsent()** 是在将来日期将会过时的接口。

作者

getfsent() 由 HP 和加州大学伯克利分校联合开发。

文件

/etc/fstab

另请参阅

fstab(4)。

名称

getgrent()、 getgrgid()、 getgrgid_r()、 getgrnam()、 getgrnam_r()、 setgrent()、 endgrent()、 fgetgrent() - 获取组文件条目

概要

```
#include <grp.h>

struct group *getgrent(void);

struct group *getgrgid(gid_t gid);

int  getgrgid_r(gid_t gid, struct group *grp, char *buffer,
        size_t buflen, struct group ** result);

struct group *getgrnam(const char *name);

int  getgrnam_r(const char *name, struct group *grp, char *buffer,
        size_t buflen, struct group ** result);

void setgrent(void);

void endgrent(void);

struct group *fgetgrent(FILE *stream);
```

过时的接口

```
#include <grp.h>

int  getgrent_r(struct group *result, char *buffer, int buflen,
        FILE **grfp);

void setgrent_r(FILE **grfp);

void endgrent_r(FILE **grfp);

int  fgetgrent_r(FILE *stream, struct group *result, char *buffer,
         int buflen);
```

说明

getgrent() 、 **getgrgid()** 和 **getgrnam()** 可用于获取组条目，并返回指向 **group** 结构的某个对象的指针。条目可能来自 **/etc/nsswitch.conf** 文件中指定的 **group** 的任何源。请参阅 *nsswitch.conf*(4) 。

group 结构是在 **<grp.h>** 中定义的，它包括以下成员：

```
char  *gr_name;      /* the name of the group */
char  *gr_passwd;    /* the encrypted group password */
gid_t gr_gid;        /* the numerical group ID */
char  **gr_mem;      /* null-terminated array of pointers to member names */
```

getgrent() 首次调用 **getgrent()** 后，它将返回指向组数据库中第一个 **group** 结构的指针；然后，返回指向数据库中下一个 **group** 结构的指针。这样，便可以通过后续调用搜索整个数据库；

setgrent() 能够回卷组数据库，以便反复执行搜索；

endgrent() 可以调用它来指示已完成组数据库的处理；

getgrgid() 从组数据库的起始位置开始搜索，直到找到与 *gid* 匹配的数字组 ID ，然后返回指向在其中找到了该 ID 的特定结构的指针；

getgrnam() 从组数据库的起始位置开始搜索，直到找到与 *name* 匹配的组名，然后返回指向在其中找到该组名的特定结构的指针。

fgetgrent() 返回指向标准 I/O 流 *stream* 中的下一个 **group** 结构的指针，该流应处于打开状态，以便进行读取，并且其内容应与 **/etc/group** 的格式匹配。

过时的接口

getgrent_r() 、 **setgrent_r()** 、 **endgrent_r()** 、 **fgetgrent_r()** 可获取组文件条目。

重入接口

getgrgid_r() 和 **getgrnam_r()** 都可以更新 **grp** 指向的 **struct** 组，以及将指向该结构的指针存储在 **result** 指向的位置。该结构应包含来自组数据库的带有匹配的 **gid** 或 **name** 的条目。应在 **buffer** 参数提供的内存（大小为 **buflen**）中分配 **grp** 指向的组结构所引用的存储。 **_SC_GETGR_R_SIZE_MAX sysconf()** 参数可确定该缓冲区所需的最大大小。如果出现错误，或者未找到请求的条目，将在 **result** 指向的位置返回空指针。

返回值

如果在读取时遇到文件结束标记或错误， **getgrent()** 、 **getgrgid()** 、 **getgrnam()** 和 **fgetgrent()** 将返回空指针。否则，返回值将指向包含有效的 **group** 结构的内部静态区。

getgrgid_r() 和 **getgrnam_r()** 成功时返回零。否则，返回一个错误编号以指明错误。

错误

如果出现任意任何情况， **getgrent()** 、 **getgrgid()** 、 **getgrnam()** 和 **fgetgrent()** 将会失败：

[EIO] 发生了 I/O 错误。

[EMFILE] 当前调用进程中打开了 *OPEN_MAX* 文件描述符。

[ENFILE] 当前系统中打开了所允许的最大数目的文件。

如果满足下列条件， **getgrgid_r()** 函数和 **getgrnam_r()** 函数将失败：

[ERANGE] 通过 *buffer* 和 *bufsize* 提供的存储空间不足以容纳得到的组结构所引用的数据。

警告

getgrent() 、 **getgrgid()** 、 **getgrnam()** 和 **fgetgrent()** 返回的值指向每次调用这其中任何一个函数后，都将被覆盖的区域。必须通过复制来保存该值。

getgrgid_r() 和 **getgrnam_r()** 的用户应注意，现在这些接口符合 POSIX.1c 标准。 **getgrent_r()** 、 **setgrent_r()** 、

endgrent_r() 和 **fgetgrent_r()** 是过时的接口。为了与现有的 DCE 应用程序兼容，还支持这些接口以及 **getgrgid_r()** 和 **getgrnam_r()** 的早期原型。

接口 **setgrent()** 、 **getgrent()** 、 **endgrent()** 、 **getgrgid()** 、 **getgrnam()** 、 **getgrgid_r()** 和 **getgrgnam_r()** 使用动态名称服务交换（请参阅 *nsswitch.conf*(4) ）。无法对使用这些接口的应用程序进行完全归档绑定。

相关内容

NFS

文件

/var/yp/*domainname***/group.byname**

/var/yp/*domainname***/group.bygid**

另请参阅

ypcat(1) 。

文件

/etc/group

另请参阅

ypcat(1)、 getgroups(2)、 getpwent(3C)、 stdio(3S)、 group(4)、 thread_safety(5)。

符合的标准

getgrent()：SVID2、SVID3、XPG2

endgrent()：SVID2、SVID3、XPG2

fgetgrent()：SVID2、SVID3、XPG2

getgrgid()：AES、SVID2、SVID3、XPG2、XPG3、XPG4、FIPS 151-2、POSIX.1

getgrnam()：AES、SVID2、SVID3、XPG2、XPG3、XPG4、FIPS 151-2、POSIX.1

setgrent()：SVID2、SVID3、XPG2

getgrnam_r()、**getgrgid_r()**：POSIX.1c

名称

gethostent()、 gethostbyaddr()、 gethostbyname()、 sethostent()、 endhostent() - 获取、设置或结束网络主机条目

概要

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

struct hostent *gethostent(void);

struct hostent *gethostbyname(const char *name);

struct hostent *gethostbyaddr(const char *addr,
                int len,
                int type);

_XOPEN_SOURCE_EXTENDED only
struct hostent *gethostbyaddr(const void *addr,
                size_t len,
                int type);

int sethostent(int stayopen);

int endhostent(void);

_XOPEN_SOURCE_EXTENDED only
void sethostent(int stayopen);
void endhostent(void);
```

说明

gethostent() 、 **gethostbyname()** 和 **gethostbyaddr()** 函数中的每一个都可以返回指向在 **<netdb.h>** 中按如下方式定义的 **hostent** 类型结构的指针：

```
struct hostent {
    char    *h_name;
    char    **h_aliases;
    int     h_addrtype;
    int     h_length;
    char    **h_addr_list;
};

#define h_addr  h_addr_list[0]
```

该结构成员包括：

h_name 主机的正式名称。

h_aliases	主机备选名称的以 null 结尾数组。
h_addrtype	返回的地址类型；始终为 **AF_INET** 。
h_length	地址长度（以字节为单位）。
h_addr_list	主机网络地址的以 null 结尾的数组。
h_addr	**h_addr_list** 中的第一个地址；该成员是为了与以前的 HP-UX 实现兼容而提供的，其中 **struct hostent** 对于每个主机只包含一个网络地址。

基于名称服务交换的操作

这些主机条目库例行程序可以内部调用名称服务交换，以访问 **/etc/nsswitch.conf** 文件（请参阅 *nsswitch.conf*(4) ）中配置的“hosts”数据库查找策略。查找策略定义用于解析主机名和 Internet 地址的支持名称服务的顺序和条件。下面列出了以下这些名称服务的操作：域名服务器、 NIS 和非服务器模式（如文件）。

域名服务器操作

如果本地系统的配置将使用用于名称/地址解析的 **named** 名称服务器（请参阅 *named*(1M) 和 *resolver*(4) ），则函数：

gethostent()	始终返回空指针。
sethostent()	如果 *stayopen* 标记为非零值，则请求使用连接流套接字查询名称服务器。每次调用 **gethostbyname()** 或 **gethostbyaddr()** 后将保持连接。
endhostent()	关闭流套接字连接。
gethostbyname() **gethostbyaddr()**	每一个都可以从名称服务器检索主机信息。匹配名称时忽略大小写。例如，**berkeley.edu** 、 **Berkeley.EDU** 和 **BERKELEY.EDU** 都与 **berkeley.edu** 条目匹配。

NIS 服务器操作

如果将网络信息服务（请参阅 *ypserv*(1M) ）的服务器 **ypserv** 用于名称或地址解析，则函数：

gethostent()	在 NIS 数据库中返回下一个条目。
sethostent()	初始化 NIS 数据库的内部密钥。如果 *stayopen* 标记为非零值，则调用 **endhostent()** 后不清除内部密钥。
endhostent()	清除 NIS 数据库内部密钥。
gethostbyname() **gethostbyaddr()**	每一个都可以从 NIS 数据库检索主机信息。匹配名称时忽略大小写。例如，**berkeley.edu** 、 **Berkeley.EDU** 和 **BERKELEY.EDU** 都与 **berkeley.edu** 条目匹配。

非服务器操作

如果将 **/etc/hosts** 文件用于名称或地址解析，则函数：

gethostent()	读取 **/etc/hosts** 的下一行，必要时打开文件。
sethostent()	打开并回卷文件。如果 *stayopen* 标记为非零值，则每次调用 **gethostent()** （无论是直接调用还是通过其他一次 **gethost** 调用间接调用）后不关闭主机数据库。
endhostent()	关闭文件。
gethostbyname()	按顺序从文件的开头开始搜索，直到找到与其 *name* 参数匹配的主机名（无论是正式名还是别名），或者遇到 EOF。匹配名称时将忽略大小写，这与上述名称服务器的情况一样。
gethostbyaddr()	按顺序从文件的开头开始搜索，直到找到与其 *addr* 参数匹配的 Internet 地址，或者遇到 EOF。

后续调用函数 **gethost*()** 、 **getaddrinfo()** 和 **getnameinfo()** 之前，必须保存返回值 **struct hostent** 。

在多线程应用程序中， **gethostent()** 、 **gethostbyaddr()** 和 **gethostbyname()** 使用每次调用可重用的线程特定存储。返回值 **struct hostent** 对每个线程应该是唯一的，线程执行下一次 **gethost*()** 调用之前，应该按需要保存该返回值。后续调用函数 **getaddrinfo()** 或 **getnameinfo()** 之前，必须保存该返回值，原因是这些函数可能内部调用 **gethost*()** 函数，从而覆盖它们的返回值。

对于多线程应用程序中的枚举，枚举中的位置是所有线程共享的进程间属性。可以在多线程应用程序中使用 **sethostent()** ，但它会重置所有线程的枚举位置。如果多个线程交错调用 **gethostent()** ，这些线程将枚举主机数据库的脱节子集。

参数

目前只识别 Internet 地址格式。在 **gethostbyaddr()** 调用中，参数 *addr* 必须是指向 *in_addr* 结构（遵循网络顺序的 Internet 地址，请参阅 *byteorder*(3N) 和头文件 **<netinet/in.h>** ）的指针。参数 *len* 必须是 Internet 地址中的字节数；也就是 **sizeof (struct in_addr)** 。参数 *type* 必须是常量 **AF_INET** 。

返回值

如果成功， **gethostbyname()** 、 **gethostbyaddr()** 和 **gethostent()** 将返回指向所请求的 **hostent** 结构的指针。

对于 **gethostbyname()** 和 **gethostbyaddr()** ，如果无法在数据库中分别找到其 *host* 和 *addr* 参数，它们将返回 NULL。如果 **/etc/hosts** 被占用，并且无法打开 **/etc/hosts** ，将同样返回 NULL。

如果 **gethostbyaddr()** 的 *addr* 或 *len* 参数无效，将同样返回 NULL。

如果名称服务器被占用， **gethostent()** 始终返回 NULL。

错误

如果名称服务器被占用，并且 **gethostbyname()** 或 **gethostbyaddr()** 返回了空指针，外部整数 **h_errno** 将包含下列其中一个值：

HOST_NOT_FOUND	没有这样的主机。
TRY_AGAIN	这通常是暂时性错误。本地服务器没有收到授权服务器的响应。稍后再试可能会成功。
NO_RECOVERY	这是一个不可恢复的错误。
NO_ADDRESS	请求的名称有效，但是不包含 IP 地址；这不是暂时性错误。这意味着向名称服务器发送另一类型的请求将得到响应。

如果名称服务器未被占用，则 **h_errno** 的值没有意义。

举例

以下代码片断将统计主机条目数：

```
int count = 0;

(void) sethostent(0);
while (gethostent() != NULL)
    count++;
(void) endhostent();
```

以下示例程序将输出给定的以“.”分隔的 IP 地址的规范名称、别名和以“.”分隔的 Internet 地址。

```
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <netinet/in.h>

main(int argc, const  char **argv)
{
    u_int addr;
    struct hostent *hp;
    char **p;

    if (argc != 2) {
        (void) printf("usage: %s IP-address\n",argv[0]);
        exit (1);
    }
    if ((int) (addr = inet_addr (argv[1])) == -1) {
        (void) printf("IP-address must be of the form a.b.c.d\n");
        exit (2);
    }
```

```
        hp=gethostbyaddr((char *) &addr, sizeof (addr), AF_INET);
        if (hp == NULL) {
            (void) printf("host information for %s no found \n", argv[1]);
            exit (3);
        }

        for (p = hp->h_addr_list; *p!=0;p++){
            struct in_addr in;
            char **q;

            (void)memcpy(&in.s_addr, *p, sizeof(in.s_addr));
            (void)printf("%s\t%s",inet_ntoa(in), hp->h_name);
            for (q=hp->h_aliases;*q != 0; q++)
                (void) printf("%s", *q);
            (void)putchar('\n');
        }
        exit (0);
    }
```

警告

由于这些函数的实现在运行时采用共享对象的动态加载和链接，因此无法静态链接使用了本联机帮助页所述接口的程序。

h_errno 引用为单线程应用程序的 **extern int** ，并定义为 **/usr/include/netdb.h** 文件中多线程应用程序的函数调用宏。引用 **h_errno** 的应用程序需要包括 **/usr/include/netdb.h** 。

过时的接口

```
int gethostent_r(struct hostent *result,
        struct hostent_data *buffer);

int gethostbyname_r(const char *name,
        struct hostent *result,
        struct hostent_data *buffer);

int gethostbyaddr_r(const char *addr,
        int len,
        int type,
        struct hostent *result,
        struct hostent_data *buffer);

int sethostent_r(int stayopen, struct hostent_data *buffer);

int endhostent_r(struct hostent_data *buffer);
```

已将上述重入接口从 **libc** 移动到 **libd4r** 。为支持现有的应用程序而提供了这些接口，将来的发行版可能会删除这

些接口。新的多线程应用程序应该使用常规 API（不带 **-r** 前缀）。

重入接口的工作方式与不带 **-r** 前缀的常规接口相同。但是，需要为 **gethostent_r()**、 **gethostbyname_r()** 和 **gethostbyaddr_r()** 传递 **struct hostent** 的地址，同时它们将在提供的参数上存储结果的地址。传入重入接口的 **struct hostent_data** 地址不能是空指针。

如果操作成功，或者在执行 **gethostent_r()** 的情况下，到达了主机列表的末端，重入例行程序将返回 **-1** 。否则返回 **0** 。

作者

gethostent() 由 Sun Microsystems Inc. 开发。

文件

/etc/hosts

另请参阅

named(1M)、 ypserv(1M)、 resolver(3N)、 ypclnt(3C)、 hosts(4)、 nsswitch.conf(4)、 ypfiles(4)、 thread_safety(5)。

符合的标准

gethostent()： XPG4

名称

gethrtime( ) - 获取高精度的时间

概要

#include <time.h>

hrtime_t gethrtime(void);

说明

gethrtime() 函数将返回当前的高精度实时时间。时间表示为自过去某个时刻开始算起的纳秒（十亿分之一秒）数。此 API 利用快速轻便的系统调用获取自某个时刻以来的纳秒数。它与当日时间毫无关系。此 API 可用于性能评测任务，并可用于廉价而精确的间隔计时。

hrtime_t 是一个带符号的 64 位数字。

返回值

成功完成后， **gethrtime()** 将返回纳秒数。否则，它将返回 -1。

警告

此 API 仅在应用程序以 -Ae（扩展的 ANSI）模式进行编译的情况下可用，原因是 64 位整数在 -Aa (ANSI) 模式下不可用。请参考 *cc*(1) 。

举例

以下代码段可测定 **getgid()** 的平均资源消耗量：

```
hrtime_t begin, end;

int i, count = 1000;

begin = gethrtime();

for (i = 0; i < count; i++)

    getgid();

end = gethrtime();

printf("Avg getgid() time = %lld nsec", (end - begin)/count );
```

名称

getlogin()、getlogin_r() - 获取登录该终端的用户的名称

概要

#include <unistd.h>

char *getlogin(void);

int getlogin_r(char *buf, size_t buflen);

说明

getlogin() 函数可检索当前登录与调用进程关联的终端的用户的名称，在 *utmpd*(1M) 维护的用户帐户数据库中可以找到该名称。

标准输入、标准输出或标准错误中，至少必须有一个是终端。对于其中出现的第一个终端，用户必须已登录该终端，并且该终端必须是调用进程会话的会话发起者进程的控制终端。

当多个登录名共享同一个用户 ID 时，可以将 **getlogin()** 函数与 **getpwnam()** 一起使用，以查找正确的口令文件条目。

要获取与调用进程的实际用户 ID 关联的用户名，建议的过程是调用 **getlogin()** ，如果调用失败，再调用 **getpwuid(getuid())** 。

要获取与有效用户 ID 关联的用户名，可调用 **getpwuid(geteuid())** 。

getlogin_r() 与 **getlogin()** 执行的操作相同，但前者在 *buf* 指向的缓冲区（该缓冲区的字节大小应传入 *buflen* ）中返回登录名。 *buf* 应该为名称和结尾空字符提供空间。使用 **sysconf()** API，并将 **_SC_LOGIN_NAME_MAX** 用作参数，可获得最大的登录名。

实际应用信息

getlogin() 的返回值指向每次调用后其内容将被覆盖的静态数据。

返回值

如果成功找到并验证了登录终端的用户的用户名， **getlogin()** 将返回指向该名称的指针。否则，将返回空指针，同时设置 **errno** 以指明错误。

如果成功找到和验证了登录终端的用户的用户名，并且将该用户名复制到了缓冲区， **getlogin_r()** 成功时返回 0，失败时返回一个错误编号。

错误

如果出现下列任意情况， **getlogin()** 和 **getlogin_r()** 将会失败：

[EACCES]	获取终端设备文件的状态所需的访问权限被拒绝。
[EMFILE]	该进程占用了过多的文件描述符。
[ENFILE]	系统占用了过多的文件描述符。

[ENOENT] 无法找到终端设备文件。

[ENOTTY] 标准输入、标准输出和标准错误中没有一个是终端，或者对于其中的第一个终端，当前没有登录注册到该终端，或者调用进程的会话发起者进程没有控制终端。

[EPERM] 标准输入、标准输出和标准错误中有一个是终端，并且在该终端上找到了当前的登录，但是该终端与作为调用进程会话的控制终端不同。

[ESRCH] 调用进程的会话发起者进程不再运行。

与 [EPERM] 关联的错误情况，阻止了对其他用户的终端具有访问权限的进程确认它们是与其他用户的登录会话相关的。

如果出现以下情况， **getlogin_r()** 也会失败：

[ERANGE] 要返回的名称的长度包括结尾空字节在内，超过了 *buflen* 。

警告

getlogin_r() 的用户应注意，现在 **getlogin_r()** 符合 POSIX.1c 线程标准。为了与现有的 DCE 应用程序兼容，还支持 **getlogin_r()** 的早期原型。

另请参阅

utmpd(1M)、 getuid(2)、 sysconf(2)、 getgrent(3C)、 getpwent(3C)、 thread_safety(5)。

符合的标准

getlogin()：AES、SVID2、SVID3、XPG2、XPG3、XPG4、FIPS 151-2、POSIX.1

getlogin_r()：POSIX.1c

名称

getmntent()、 getmntent_r()、 setmntent()、 addmntent()、 delmntent()、 endmntent()、 hasmntopt() - 获取文件系统描述符文件条目

概要

```
#include <mntent.h>

FILE *setmntent(const char *path, char *type);

struct mntent *getmntent(FILE *stream);

int getmntent_r(
    FILE *stream,
    struct mntent *result,
    char *buffer,
    int buflen);

int addmntent(FILE *stream, struct mntent *mnt);

int delmntent(FILE *stream, struct mntent *mnt);

char *hasmntopt(struct mntent *mnt, const char *opt);

int endmntent(FILE *stream);
```

说明

这些例行程序可以替换过时的用于访问文件系统描述文件 **/etc/fstab** 的 **getfsent()** 例行程序（请参阅 *getfsent*(3X)）。还可用于访问挂接的文件系统描述文件 **/etc/mnttab** 。

setmntent() 打开一个文件系统说明文件，并返回一个文件指针，该指针可用于 **getmntent()** 、 **addmntent()** 、 **delmntent()** 或 **endmntent()** 。 *type* 参数与 **fopen()** 中使用的参数相同。

getmntent() 从 *stream* 读取下一行，返回指向以下结构的指针，该结构包含文件系统描述文件 **<mntent.h>** 中某个行的组成字段。这些字段的含义如 *fstab*(4) 中所述。

```
struct mntent {
  char   *mnt_fsname;       /* file system name */
  char   *mnt_dir;          /* file system path prefix */
  char   *mnt_type;         /* hfs, nfs, swap, or xx */
  char   *mnt_opts;         /* ro, suid, etc. */
  int    mnt_freq;          /* dump frequency, in days */
  int    mnt_passno;        /* pass number on parallel fsck */
  long   mnt_time;          /* When file system was mounted;*/
                            /* see mnttab(4). */
                            /* (0 for NFS) */
};
```

getmntent_r() 使用三个额外的参数来提供与 **getmntent()** 生成的结果等效的结果。这些额外参数为：

1. 将在其中存储结果的 **struct mntent** 的地址。
2. 用于存储 **struct mntent** 中字段所指向的字符串的缓冲区。
3. 用户提供的缓冲区的长度。建议使用的缓冲区长度为 1025。

addmntent() 将 **mntent** 结构 *mnt* 添加到打开文件 *stream* 的结尾。请注意，必须打开 *stream* ，以便能够读取。调用 **addmntent** 并返回结果后， *stream* 的文件位置指示符将指向 EOF。

delmntent() 删除使用 **setmntent** 打开的 *stream* 文件内，与 **mntent** 结构 *mnt* 中的 *mnt_fsname* 和 *mnt_dir* 匹配的所有条目。如果 *mnt_fsname* 是空指针，将删除与 *mnt_dir* 匹配的所有条目。如果 *mnt_dir* 是空指针，将删除与 *mnt_fsname* 匹配的所有条目。如果 *mnt_fsname* 和 *mnt_dir* 都是空指针，则这是一个错误情况。请注意，必须通过 **setmntent** 打开 *stream* ，以便能够读取和写入（r+ 或 a+）。调用 **delmntent** 并返回结果后， *stream* 的文件位置指示符将指向 EOF。

hasmntopt() 扫描 **mntent** 结构 *mnt* 的 **mnt_opts** 字段，以查找与 *opt* 匹配的子字符串。如果找到匹配项，将返回该子字符串的地址。否则，返回值为下列 *default behaviors* 中的一个：

选项字符串	对应的缺省行为
MNTOPT_RW	如果 **mnt_opts** 没有 **MNTOPT_RO** 选项，将返回 **MNTOPT_RW**。否则，返回 NULL。
MNTOPT_SUID	如果 **mnt_opts** 没有 **MNTOPT_NOSUID** 选项，将返回 **MNTOPT_SUID** 。否则，返回 NULL。
MNTOPT_NOQUOTA	如果 **mnt_opts** 没有 **MNTOPT_USRQUOTA** 或 **MNTOPT_QUOTA** 选项，将返回 **MNTOPT_NOQUTOA** 。否则，返回 NULL。
MNTOPT_FG	如果 **mnt_opts** 没有 **MNTOPT_BG** 选项，并且 **mnt_type** 为 **MNTTYPE_NFS** ，将返回 **MNTOPT_FG** 。否则，返回 NULL。
MNTOPT_HARD	如果 **mnt_opts** 没有 **MNTOPT_SOFT** 选项，并且 **mnt_type** 为 **MNTTYPE_NFS** ，将返回 **MNTOPT_HARD** 。否则，返回 NULL。
MNTOPT_INTR	如果 **mnt_opts** 没有 **MNTOPT_NOINTR** 选项，并且 **mnt_type** 为 **MNTTYPE_NFS** ，将返回 **MNTOPT_INTR** 。否则，返回 NULL。
MNTOPT_DEVS	如果 **mnt_opts** 没有 **MNTOPT_NODEVS** 选项，并且 **mnt_type** 为 **MNTTYPE_NFS** ，将返回 **MNTOPT_DEVS** 。否则，返回 NULL。

如果没有发生任何 *default behaviors* ，则返回 NULL。

注释：如果返回值是其中一个 *default behaviors* 的结果，则它是伪选项字符串，而不是 **mnt_ops** 字段中的指针。

endmntent() 关闭文件。

<mntent.h> 中提供了以下定义：

```
#define MNT_CHECKLIST  " /etc/fstab"
#define MNT_MNTTAB     "    /etc/mnttab"

#define MNTMAXSTR      128                  /* Max size string in mntent */

#define MNTTYPE_HFS    "hfs"                /* HFS file system */
#define MNTTYPE_CDFS   "cdfs"               /* CD-ROM file system */
#define MNTTYPE_NFS    "nfs"                /* Network file system */
#define MNTTYPE_SWAP   "swap"               /* Swap device */
#define MNTTYPE_SWAPFS "swapfs"             /* File system swap */
#define MNTTYPE_IGNORE "ignore"             /* Ignore this entry */

#define MNTOPT_DEFAULTS "defaults"          /* Use all default options */
#define MNTOPT_RO      "ro"                 /* Read only */
#define MNTOPT_RW      "rw"                 /* Read/write */
#define MNTOPT_SUID    "suid"               /* Set uid allowed */
#define MNTOPT_NOSUID  "nosuid"             /* No set uid allowed */
#define MNTOPT_QUOTA   "quota"              /* Enable user quotas */
#define MNTOPT_USRQUOTA "usrquota"          /* Enable user quotas */
#define MNTOPT_GRPQUOTA "grpquota"          /* Enable group quotas */
#define MNTOPT_NOQUOTA "noquota"            /* Disable user quotas */
#define MNTOPT_DEV     "dev"                /* Device ID of the filesystem mounted */
```

<mntent.h> 中提供了以下用于设备交换的定义：

```
#define MNTOPT_END   "end"                  /* swap after end of file system,
                                               Series 300/400/700 only */
```

<mntent.h> 中提供了以下用于文件系统交换的定义：

```
#define MNTOPT_MIN   "min"          /* minimum file system swap */
#define MNTOPT_LIM   "lim"          /* maximum file system swap */
#define MNTOPT_RES   "res"          /* reserve space for file system */
#define MNTOPT_PRI   "pri"          /* file system swap priority */
```

网络功能

NFS

<mntent.h> 中提供了以下定义：

```
#define MNTOPT_BG      "bg"          /* Retry mount in background */
#define MNTOPT_FG      "fg"          /* Retry mount in foreground */
#define MNTOPT_RETRY   "retry"       /* Number of retries allowed */
#define MNTOPT_RSIZE   "rsize"       /* Read buffer size in bytes */
#define MNTOPT_WSIZE   "wsize"       /* Write buffer size in bytes*/
#define MNTOPT_TIMEO   "timeo"       /* Timeout in 1/10 seconds */
#define MNTOPT_RETRANS "retrans"     /* Number of retransmissions */
#define MNTOPT_PORT    "port"        /* Server's IP NFS port */
#define MNTOPT_SOFT    "soft"        /* Soft mount */
#define MNTOPT_HARD    "hard"        /* Hard mount */
#define MNTOPT_INTR    "intr"        /* Interruptible hard mounts */
#define MNTOPT_NOINTR  "nointr"      /* Uninterruptible hard mounts*/
#define MNTOPT_DEVS    "devs"        /* Device file access allowed */
#define MNTOPT_NODEVS  "nodevs"      /* No device file access allowed*/
```

返回值

setmntent() 出现错误时返回空指针。 **setmntent()** 尝试在其打开的文件上建立一个独占写入锁，即：如果将下列类型之一传递给 **setmntent()** ，以打开要写入（或更新）的文件：“w”、“a”、“r+”、“w+”或“a+”。如果 **setmntent()** 无法获取锁，将返回空指针，并将 errno 设置为 [EACCES] 或 [EAGAIN] 。有关 **setmntent()** 的详细信息，请参阅下面的“实际应用信息”。

getmntent() 如果遇到错误或 EOF，将返回空指针。否则， **getmntent()** 返回指向 **mntent** 结构的指针。在 **/etc/fstab** 和 **/etc/mnttab** 中，构成 **mntent** 结构的某些字段是可选的。在提供的结构中，这些缺少的字符指针字段设置为 NULL， **mnt_freq** 和 **mnt_passno** 所缺少的整数字段设置为 –1。如果缺少 **mnt_time** 的整数字段，该字段将设置为 **0** 。

getmntent_r() 遇到错误或 EOF，或者提供的缓冲区长度不足时，将返回 **-1** 。如果操作成功，将返回 **0**。

addmntent() 错误时返回 **1** 。

delmntent() 错误时返回 **-1** 。如果 *stream* 或 *mnt* 是空指针，或者 **mntent** 结构 *mnt* 中的 *mnt_fsname* 和 *mnt_dir* 都是空指针，会将 errno 设置为 [EINVAL] 。如果已打开 *stream* ，以便能够读取 (r)、追加 (a) 或写入 (w)，会将 errno 设置为 [EBADF] 。如果操作成功，将返回从文件删除的条目数。如果没有匹配的条目， **delmntent** 将返回 **0** ，且不设置 errno。

endmntent() 如果 **setmntent()** 锁定了文件，则返回 **1** ，并解锁文件。

举例

以下代码将删除一个条目：

```
struct mntent mnt_entry;
FILE *fp;
int retval = NOT_DELETED;

mnt_entry.mnt_fsname = "/dev/vg00/lvol7";
mnt_entry.mnt_dir = "/disk7";

if ((fp = setmntent(MNT_MNTTAB, "r+")) != NULL) {
      if (delmntent(fp, &mnt_entry) > 0)
            retval = DELETED;
      (void)endmntent(fp);
}
return(retval);
```

实际应用信息

读取 **mntent** 数据时将无法保证数据的完整性，原因是使用只读权限打开文件时， **setmntent()** 无法锁定此文件。作为解决这个问题的一种方法，程序可以不断地循环，直到读取文件之前和之后的最后一个修改时间及文件大小相同。

以下代码将实现数据完整性。

```
struct stat statbuf;
FILE *fp;
time_t orig_mtime;
off_t orig_size;
int read_status = FALSE;
int retry;

for (retry = 0; retry < NO_OF_RETRIES; retry++){

  /*
   * If file is empty, do not bother reading it.
   * Sleep and then retry the stat().
   */

  if (stat(MNT_MNTTAB, &statbuf) != 0){
     return (STAT_FAILED);
  }
```

```
        if (statbuf.st_size == 0){
            sleep(1);
            if (stat(MNT_MNTTAB, &statbuf) != 0){
                return(STAT_FAILED);
            }
            else{
                if (statbuf.st_size == 0){
                    continue;
                }
            }
        }

        if ((fp = setmntent(MNT_MNTTAB, "r")) == NULL){
            return (SETMNTENT_FAILED);
        }

        /*
         * operation on MNT_MNTTAB goes here...
         */

        orig_mtime = statbuf.st_mtime;
        orig_size = statbuf.st_size;

        if (stat(MNT_MNTTAB, &statbuf) != 0){
            return (STAT_FAILED);
        }
        if ((statbuf.st_mtime == orig_mtime) && (statbuf.st_size == orig_size)){
            read_status = TRUE;
            break;
        }
    }
```

程序应该预料到另一进程可能对这些 API 访问的文件执行写入锁定，原因是打开此文件以便写入/更新时， **setmntent()** 将尝试建立一个排它写入锁。

不支持使用文本编辑器来操作这些 API 访问的文件。 **setmntent()** 和 **endmntent()** 对于每进程锁定来说是安全的。由 **fork()** 之后且 **exec()** 之前的子进程调用 **setmntent()** 、 **getmntent()** 、 **addmntent()** 、 **delmntent()** 、 **hasmntent()** 和 **endmntent()** 是不安全的。

作者

addmntent() 、 **endmntent()** 、 **getmntent()** 、 **hasmntopt()** 和 **setmntent()** 由加州大学伯克利分校、Sun Microsystems, Inc. 和 HP 联合开发。 **delmntent()** 由 HP 开发。

文件

/etc/fstab
/etc/mnttab

另请参阅

getfsent(3X)、 fstab(4)、 mnttab(4)、 thread_safety(5)。

名称

getnetconfig()、setnetconfig()、endnetconfig()、getnetconfigent()、 freenetconfigent()、nc_perror()、nc_sperror() - 获取网络配置数据库条目

概要

#include <netconfig.h>

struct netconfig *getnetconfig(void *_handlep_);

void *setnetconfig(void);

int endnetconfig(void *_handlep_);

struct netconfig *getnetconfigent(const char *_netid_);

void freenetconfigent(struct netconfig *_netconfigp_);

void nc_perror(const char *_msg_);

char *nc_sperror(void);

说明

本页介绍的库例行程序是网络选择组件的一部分。应用程序可以通过这些例行程序访问系统网络配置数据库 **/etc/netconfig** 。除用于访问 **netconfig** 数据库的例行程序外，网络选择还包括环境变量 **NETPATH** （请参阅 *environ*(5) ），以及 *getnetpath*(3N) 中介绍的 **NETPATH** 访问例行程序。

getnetconfig() 返回指向 **netconfig** 数据库中格式设置为 **struct netconfig** 的当前条目的指针。连续调用该例行程序将在 **netconfig** 数据库中返回连续的 **netconfig** 条目。可以使用 **getnetconfig()** 搜索整个 **netconfig** 文件。到达文件的结尾时， **getnetconfig()** 返回 NULL。 *handlep* 是通过 **setnetconfig()** 获取的句柄。

调用 **setnetconfig()** 可以“绑定到”或者“回卷” **netconfig** 数据库。首次调用 **getnetconfig()** 之前，必须先调用 **setnetconfig()** ，也可以在其他任何时候调用后者。调用 **getnetconfigent()** 之前，不必要先调用 **setnetconfig()** 。 **setnetconfig()** 将返回 **getnetconfig()** 使用的唯一句柄。

当处理已完成，可以释放重新使用的资源时，应调用 **endnetconfig()** 。 *handlep* 是通过 **setnetconfig()** 获取的句柄。但是，程序员应注意，最后一次调用 **endnetconfig()** 将释放 **getnetconfig()** 为 **struct netconfig** 数据结构分配的所有内存。不可以在调用 **endnetconfig()** 之后调用 **setnetconfig()** 。

getnetconfigent() 返回指向对应于 *netid* 的 **struct netconfig** 结构的指针。如果 *netid* 无效（也就是说，没有在 *netconfig* 数据库中指定条目），该例行程序将返回 NULL。

freenetconfigent() 可释放 *netconfigp* （事先由 **getnetconfigent()** 返回）指向的 **netconfig** 结构。

nc_perror() 可将一条消息输出到标准错误，指明上述任何例行程序失败的原因。在该消息的前面添加字符串 *msg* 和一个冒号。在消息的结尾追加一个 NEWLINE。

nc_sperror() 与 **nc_perror()** 相似，但它不将消息发送到标准错误，而是返回一个指向包含错误消息的字符串的指针。

nc_perror() 和 **nc_sperror()** 还可以与 *getnetpath*(3N) 中定义的 **NETPATH** 访问例行程序一起使用。

多线程应用信息

线程安全: 是
取消安全: 是
派生安全: 否
异步取消安全: 否
异步信号安全: 否

在多线程环境中可以安全地调用这些函数。它们可能是取消点，原因是它们调用作为取消点的函数。

在多线程环境中，在 **fork()** 之后和 **exec()** 之前由子进程调用这些函数并不安全。支持异步取消或异步信号的多线程应用程序不应调用这些函数。

返回值

setnetconfig() 返回 **getnetconfig()** 要使用的唯一句柄。如果出错， **setnetconfig()** 将返回 NULL，同时可以使用 **nc_perror()** 或 **nc_sperror()** 输出失败原因。

getnetconfig() 返回指向 **netconfig()** 数据库中格式设置为 **struct netconfig** 的当前条目的指针。到达文件的结尾时， **getnetconfig()** 返回 NULL。

如果成功， **endnetconfig()** 返回 **0** ，如果失败（例如，未事先调用 **setnetconfig()** 时），则返回 **-1** 。

如果成功， **getnetconfigent()** 返回指向对应于 *netid* 的 **struct netconfig** 结构的指针；否则，返回 NULL。

nc_sperror() 返回指向包含了错误消息字符串的缓冲区的指针。每次调用后都会覆盖此缓冲区。在多线程应用程序中，此缓冲区将实现为特定线程的数据。

另请参阅

getnetpath(3N)、 netconfig(4)、 environ(5)。

名称

getnetent()、 getnetbyaddr()、 getnetbyname()、 setnetent()、 endnetent() - 获取、设置或结束网络条目

概要

```
#include <sys/socket.h>
#include <netdb.h>

struct netent *getnetent(void);

struct netent *getnetbyname(const char *name);

struct netent *getnetbyaddr(unsigned int net, int type);

_XOPEN_SOURCE_EXTENDED only
struct netent *getnetbyaddr(in_addr_t net, int type);

int setnetent(int stayopen);

int endnetent(void);

_XOPEN_SOURCE_EXTENDED only
void setnetent(int stayopen);
void endnetent(void);
```

说明

getnetent() 、 **getnetbyname()** 和 **getnetbyaddr()** 中的每一个都可以返回指向 *netent* 类型的结构的指针，该结构包含网络数据库 **/etc/networks** 中某个行的组成字段。

该结构成员包括:

n_name	网络的正式名称。
n_aliases	网络备选名称的以 null 结尾的列表。
n_addrtype	返回的网络号的类型；始终为 **AF_INET** 。
n_net	网络号。

函数的行为如下:

getnetent()	读取文件的下一行，必要时打开文件。
setnetent()	打开并回卷文件。如果 *stayopen* 标记为非零值，则每次调用 **getnetent()** （无论是直接调用还是通过其他 **getnet*** 调用间接调用）后不关闭网络数据库。
endnetent()	关闭文件。
getnetbyname()	按顺序从文件的开头开始搜索，直到找到与其参数 *name* 匹配的网络名（无论是正式名称还是别名），或者遇到 EOF。

getnetbyaddr() 按顺序从文件的开头开始搜索，直到找到与其参数 *net* 匹配的网络号，或者遇到 EOF。参数 *net* 必须按网络顺序排列。参数 *type* 必须是常量 **AF_INET** 。网络号按主机顺序提供（请参阅 *byteorder*(3N) ）。

如果系统运行的是网络信息服务 (NFS)，那么 **getnetbyname()** 和 **getnetbyaddr()** 将从 NIS 服务器（请参阅 *ypserv*(1M) 和 *ypfiles*(4) ）获取各自的网络信息。

在多线程应用程序中， **getnetent()** 、 **getentbyaddr()** 和 **getentbyname()** 使用每次调用可重复使用的特定线程的存储。返回值 **struct netent** 应该对每个线程是唯一的，线程执行下一次 **getnet*()** 调用之前，应该按需要保存该返回值。

对于多线程应用程序中的枚举，枚举中的位置是所有线程共享的进程间属性。可以在多线程应用程序中使用 **setnetent()** ，但它会重置所有线程的枚举位置。如果多个线程交错调用 **getnetent()** ，这些线程将枚举网络数据库的脱节子集。

基于名称服务交换的操作

库例行程序 **getnetbyname()** 、 **getnetbyaddr()** 和 **getnetent()** 可以内部调用名称服务交换，以访问 **/etc/nsswitch.conf** 文件（请参阅 *nsswitch.conf*(4) ）中配置的“networks”数据库查找策略。查找策略定义用于解析网络名和地址的受支持名称服务的顺序和条件。

返回值

如果 **getnetent()** 、 **getnetbyname()** 和 **getnetbyaddr()** 无法打开 **/etc/networks** ，或者遇到 EOF 时，那么它们将返回空指针 (0)。如果 **getnetbyaddr()** 的 *type* 参数无效，它也会返回空指针。

举例

以下代码片断将统计网络条目数：

```
int count = 0;

(void) setnetent(0);
while (netbuf=getnetent() != NULL)
    count++;
(void) endnetent();
```

过时的接口

int getnetent_r(struct netent *result, struct netent_data *buffer);

```
int getnetbyname_r(
    const char *name,
    struct netent *result,
    struct netent_data *buffer);

int getnetbyaddr_r(
    int net,
    int type,
```

struct netent *result,
struct netent_data *buffer);

int setnetent_r(int stayopen, struct netent_data *buffer);

int endnetent_r(struct netent_data *buffer);

上述重入接口已从 **libc** 移至 **libd4r** 。为支持现有的应用程序而包括了这些接口，将来的发行版可能会删除这些接口。新的多线程应用程序不应使用这些 API。

重入接口与常规接口（不带 **-r** 后缀）的工作方式相同。但是，需要为 **getnetent_r()** 、 **getnetbyname_r()** 和 **getnetbyaddr_r()** 传递 *struct netent* 的地址，同时它们将在提供的参数上存储结果的地址。文件 **<netdb.h>** 中定义的一个附加参数，即 *struct netent_data* 的地址，不能是空指针。

如果操作不成功， **getnetent_r()** 、 **getnetbyname_r()** 、 **getnetbyaddr_r()** 、 **setnetent_r()** 和 **endnetent_t()** 将返回 –1。否则，将返回 0。

警告

由于这些函数的实现在运行时采用共享对象的动态加载和链接，因此无法静态链接使用了本联机帮助页所述接口的程序。

作者

getnetent() 由 Sun Microsystems Inc. 开发。

文件

/etc/networks

另请参阅

ypserv(1M)、 networks(4)、 ypfiles(4)、 nsswitch.conf(4)、 thread_safety(5)。

符合的标准

getnetent()： XPG4

名称

getnetgrent( )、 setnetgrent( )、 endnetgrent( )、 innetgr( )、 - 获取网络组条目

概要

```
int innetgr(
    char *netgroup,
    char *machine,
    char *user,
    char *domain
);

int setnetgrent(char *netgroup);

int endnetgrent();

int getnetgrent(
    char **machinep,
    char **userp,
    char **domainp
);
```

说明

这些函数用于测试系统数据库中定义的“netgroup”网络组中的成员关系，及枚举这些网络组的成员。 Netgroup 是 (machine,user,domain) 三元组（请参阅 *netgroup*(4) ）的集合。

这些函数可查询 **/etc/nsswitch.conf** 文件（请参阅 *nsswitch.conf*(4) ）中为 **netgroup** 指定的源。

如果某个 netgroup *netgroup* 包含作为成员的指定三元组 *machine* 、 *user* 和 *domain* ，那么函数 **innetgr()** 将返回 **1** ；否则，该函数返回 **0** 。所提供的指针 *machine* 、 *user* 和 *domain* 中的任何一个都可能是 **NULL** ，表示三元组的该位置中与所有值匹配的“通配符”。

在多线程应用程序中使用 **innetgr()** 函数是安全的。

函数 **setnetgrent()** 、 **getnetgrent()** 和 **endnetgrent()** 用于枚举给定的网络组成员。

函数 **setnetgrent()** 可建立参数 *netgroup* 中指定的网络组，并将此网络组作为需要枚举其成员的当前组。

连续调用函数 **getnetgrent()** 将枚举通过调用 **setnetgrent()** 建立的组的成员；如果每次调用成功获取了网络组的另一个成员，则该调用将返回 1；如果组中没有其他的成员，则返回 0。

调用 **getnetgrent()** 时，会将这三个字符指针的地址用作参数；例如：

char ****mp*, ****up***, ****dp***;**

getnetgrent(&*mp***, &***up***, &***dp***);**

如果从 **getnetgrent()** 成功返回，指针 *mp* 将指向其中包含成员三元组的计算机名部分的特定线程存储区， *up* 指向其中包含用户名的特定线程存储区， *dp* 指向其中包含域名的特定线程存储区。如果为 *mp* 、 *up* 或 *dp* 返回的

指针为 **NULL** ，则表示 netgroup 的元素包含三元组的该位置中的通配说明符。

调用 **endnetgrent()** 后，将释放 **setnetgrent()** 分配的存储，但不应由调用方释放。

函数 **endnetgrent()** 可释放以前的 **setnetgrent()** 调用所分配的空间。只要对新的网络组执行了 **setnetgrent()** 调用，便会隐式执行 **endnetgrent()** 的等效调用。

请注意，尽管在多线程应用程序中使用 **setnetgrent()** 和 **endnetgrent()** 是安全的，但两者的影响是进程范围内的。调用 **setnetgrent()** 将重置所有线程的枚举位置。如果多个线程交错调用 **getnetgrent_r()** ，则每个线程都将枚举 netgroup 的脱节子集。因此，要在多线程应用程序中有效地使用这些函数，需要调用方进行协调。

多线程应用信息

线程安全:	是
取消安全:	是
派生安全:	否
异步取消安全:	否
异步信号安全:	否

在多线程环境中可以安全地调用这些函数。它们可能是取消点，原因是它们调用作为取消点的函数。

在多线程环境中，在 **fork()** 之后和 **exec()** 之前由子进程调用这些函数并不安全。支持异步取消或异步信号的多线程应用程序不应调用这些函数。

警告

由于这些函数的实现在运行时采用共享对象的动态加载和链接，因此无法静态链接使用了本联机帮助页所述接口的程序。

文件

/etc/netgroup

/etc/nsswitch.conf

另请参阅

netgroup(4)、nsswitch.conf(4)。

名称

getnetpath()、setnetpath()、endnetpath() - 获取对应于 NETPATH 组件的 /etc/netconfig 条目

概要

#include <netconfig.h>

struct netconfig *getnetpath(void *handlep);

void *setnetpath(void);

int endnetpath(void *handlep);

说明

本页介绍的例行程序是 Network Selection 组件的一部分。它们为应用程序提供了访问系统网络配置数据库 **/etc/netconfig** 提供应用程序权限（当 **NETPATH** 环境变量过滤数据库时（请参阅 *environ*(5)）。对于同时直接访问网络配置数据库的其他例行程序，请参阅 *getnetconfig*(3N)。 **NETPATH** 变量是一个以冒号分隔的网络标识符列表。

getnetpath() 返回一个指向 **netconfig** 数据库条目的指针，此条目对应于第一个有效的 **NETPATH** 组件。 **netconfig** 条目的格式被编排为 **struct netconfig** 。针对每个后续调用， **getnetpath()** 都返回一个指向 **netconfig** 条目的指针，此条目对应于下一个有效的 **NETPATH** 组件。因此， **getnetpath()** 可用于在 **netconfig** 数据库中搜索 **NETPATH** 变量中包括的所有网络。当用尽 **NETPATH** 时， **getnetpath()** 将返回 NULL。

调用 **setnetpath()** 将会“绑定到”或“回卷” **NETPATH** 。 **setnetpath()** 必须在首次调用 **getnetpath()** 之前调用，同时也可以在其他任何时候调用。该函数返回 **getnetpath()** 使用的句柄。

getnetpath() 以无人提示方式忽略无效的 **NETPATH** 组件。如果 **netconfig** 数据库中没有相应的条目，则 **NETPATH** 组件将无效。

如果未设置 **NETPATH** 变量，则 **getnetpath()** 将表现为 **NETPATH** 好像按照列举顺序被设置为 **netconfig** 数据库中的“缺省”或“可视”网络序列。

在进程完成时，可以调用 **endnetpath()** 从 **NETPATH** “取消绑定”，以便释放资源供重新使用。但是，程序员应注意，对于 **struct netconfig** 数据结构， **endnetpath()** 将释放由 **getnetpath()** 分配的所有内存。

多线程应用信息

线程安全:	是
取消安全:	是
派生安全:	否
异步取消安全:	否
异步信号安全:	否

在多线程环境中可以安全地调用这些函数。它们可能是取消点，原因是它们调用作为取消点的函数。

在多线程环境中，在 **fork()** 之后和 **exec()** 之前由子进程调用这些函数并不安全。支持异步取消或异步信号的多线程应用程序不应调用这些函数。

返回值

setnetpath() 返回一个由 **getnetpath()** 使用的句柄。在出现错误时， **setnetpath()** 返回 NULL。 **nc_perror()** 或 **nc_sperror()** 可用于输出失败的原因。请参阅 *getnetconfig*(3N) 。

当第一次调用时， **getnetpath()** 返回一个指向 **netconfig** 数据库条目的指针，此条目对应于第一个有效的 **NETPATH** 组件。当用尽 **NETPATH** 时， **getnetpath()** 将返回 NULL。

如果成功， **endnetpath()** 返回 **0** ，如果失败（例如，未事先调用 **setnetpath()** 时），则返回 **-1** 。

另请参阅

getnetconfig(3N)、 netconfig(4)、 environ(5)。

名称

getnstr、mvgetnstr、mvwgetnstr、wgetnstr、 — 从终端获取多字节字符长度有限的字符串

概要

#include <curses.h>

int getnstr(char ****str*, int** *n***);**

int mvgetnstr(int *y***, int** *x***, char ****str***, int** *n***);**

int mvwgetnstr(WINDOW ****win*, int** *y***, int** *x***, char ****str***, int** *n***);**

int wgetnstr(WINDOW ****win*, char ****str***, int** *n***);**

说明

getnstr() 和 **wgetnstr()** 的作用就好像是在接收换行符或回车符之前，对 **getch()** 进行一系列调用。得到的值放在 *str* 所指向的区域中。 **getnstr()** 和 **wgetnstr()** 函数最多读取 *n* 个字节，从而防止输入缓冲区发生溢出。对用户的清除字符和抹行字符以及所有特殊键（如功能键、Home 键和清除键等）都进行了解释。

mvgetnstr() 函数除了对 **move()** 进行调用，然后对 **getch()** 进行一系列调用之外，与 **getnstr()** 完全相同。**mvwgetnstr()** 函数除了对 **wmove()** 进行调用，然后对 **wgetch()** 进行一系列调用之外，与 **getnstr()** 相同。

getnstr() 、 **wgetnstr()** 、 **mvgetnstr()** 和 **mvwgetnstr()** 函数都只返回与字符相关联的整个多字节序列。如果数组至少能包含一个字符，那么这些函数将使用完整字符填充数组。如果数组无法包含任何完整的字符，那么函数将失败。

返回值

成功完成后，这些函数将返回 OK。否则，将返回 ERR。

错误

没有定义任何错误。

实际应用信息

传统实现通常限制返回的字节数为 256。

另请参阅

beep(3X)、 getch(3X)、 curses_intro(3X)、请参阅“输入进程”，<curses.h>。

历史变更记录

首次发布于 X/Open Curses，第 4 期。

名称

getn_wstr()、 get_wstr()、 mvget_wstr()、 mvgetn_wstr()、 mvwget_wstr()、 mvwgetn_wstr()、 wget_wstr()、 wgetn_wstr() - 从终端获取宽字符数组和功能键代码

概要

#include <curses.h>

int getn_wstr(wchar_t ****wstr*, int** *n***);**

int get_wstr(wchar_t ****wstr*);**

int mvgetn_wstr(int *y***, int** *x***, wchar_t ****wstr***, int** *n***);**

int mvget_wstr(int *y***, int** *x***, wchar_t ****wstr***);**

int mvwgetn_wstr(WINDOW ****win*, int** *y***, int** *x***, wchar_t ****wstr***, int** *n***);**

int mvwget_wstr(WINDOW ****win*, int** *y***, int** *x***, wchar_t ****wstr***);**

int wgetn_wstr(WINDOW ****win*, wchar_t ****wstr***, int** *n***);**

int wget_wstr(WINDOW ****win*, wchar_t ****wstr***);**

说明

get_wstr() 的作用好像调用了一系列 **get_wch()** ，直到处理了一个新行字符、行末尾字符或文件末尾字符。根据 **<wchar.h>** 中的定义，使用 WEOF 表示文件末尾字符。新行或行末尾表示为它的 **wchar_t** 值。在所有的实例中，字符串以空 **wchar_t** 结尾。得到的数值放置在 *wstr* 所指向的区域中。

解释用户的清除和终止字符，并影响返回字符的顺序。

wget_wstr() 的作用好像调用了一系列 **wget_wch()** 。

mvget_wstr() 的作用好像调用了 **move()** ，然后再调用一系列 **get_wch()** 。 **mvwget_wstr()** 的作用好像调用了 **wmove()** ，然后再调用一系列 **wget_wch()** 。 **mvget_nwstr()** 的作用好像调用了 **move()** ，然后再调用一系列 **get_wch()** 。 **mvwget_nwstr()** 的作用好像调用了 **wmove()** ，然后再调用一系列 **wget_wch()** 。

getn_wstr() 、 **mvgetn_wstr()** 、 **mvwgetn_wstr()** 和 **wgetn_wstr()** 函数最多读取 *n* 个字符，可以防止应用程序溢出输入缓存。

返回值

成功完成后，这些函数返回 OK。否则，返回 ERR。

错误

没有定义任何错误。

实际应用信息

使用 **get_wstr()** 、 **mvget_wstr()** 、 **mvwget_wstr()** 或 **wget_wstr()** 读取溢出 *wstr* 所指向的数组会引发未定义的结果。建议分别使用 **getn_wstr()** 、 **mvgetn_wstr()** 、 **mvwgetn_wstr()** 或 **wgetn_wstr()** 。

这些函数不能返回 KEY_ 值，因为无法区分 KEY_ 值和有效的 **wchar_t** 值。

另请参阅

get_wch(3X)、 getstr(3X)、 <curses.h>、 <wchar.h>。

《X/Open System Interfaces and Headers, Issue 4, Version 2》

《X/Open System Interface Definitions, Issue 4, Version 2》，第 9 章“General Terminal Interface”。

名称

getopt()、 optarg、 opterr、 optind、 optopt - 从参数向量获取选项字母

概要

```
#include <unistd.h>

int getopt(
        int argc,
        char * const argv[],
        const char *optstring
        );

extern char *optarg;

extern int optind, opterr, optopt;
```

说明

getopt() 返回 *argv* 中的下一个选项字母（从 *argv***[1]** 开始），该字母与 *optstring* 中的某个字母匹配。 *argc* 和 *argv* 是传递给 **main()** 的参数计数和参数数组。 *optstring* 是已识别的选项字符的字符串；如果某个字符后接冒号，则选项需要带有一个参数，并且可以或不可以使用空白字符将选项与参数分隔。

optind 是要处理的 *argv***[]** 向量的下一个元素的索引。系统会将其初始化为 1， **getopt()** 处理完 *argv***[]** 的每个元素后，会将其更新。

getopt() 返回 *argv* 中与 *optstring* 中的某个字符匹配的下一个选项字符（如果有匹配项的话）。如果选项带有参数， **getopt()** 会按以下方式将变量 **optarg** 设置为指向选项参数：

- 如果选项为 *argv* 的某个元素指向的字符串中的最后一个字符，则 **optarg** 包含 *argv* 的下一个元素，同时将 **optind** 加 2。如果所得的 **optind** 值大于或等于 *argc* ，则表示缺失了某个选项参数， **getopt()** 将返回错误指示。
- 否则， **optarg** 将指向接在 *argv* 的该元素中的选项字符后面的字符串，同时将 **optind** 加 1。

如果调用了 **getopt()** ，或者 *argv***[optind]** 为 NULL，或者， *argv***[optind]** 指向的字符串要么不是以字符 **-** 开头，要么只包括字符 **-** ，那么 **getopt()** 将返回 –1，且不更改 **optind** 。如果 *argv***[optind]** 指向字符串 **--** ， **getopt()** 在递增 **optind** 后将返回 –1。

如果 **getopt()** 遇到了 *optstring* 中未包含的选项字符，将返回问号 (**?**) 字符。在检测到缺失的选项参数的情况下，如果 *optstring* 的第一个字符为冒号，将返回冒号字符 (**:**)，否则返回问号字符。无论是哪种情况， **getopt()** 都会将变量 **optopt** 设置为导致错误的选项字符。如果应用程序没有将变量 **opterr** 设置为零，并且 *optstring* 的第一个字符不是冒号， **getopt()** 还会将诊断消息输出到标准错误。

可以使用特殊选项 **--** 来定界选项的结尾；将返回 –1，并跳过 **--** 。

返回值

getopt() 返回命令行上指定的下一个选项字符。如果 **getopt()** 检测到有缺失的参数，并且 *optstring* 的第一个字符为冒号 (**:**) ，则返回冒号 (**:**)。

如果 **getopt()** 遇到 *optstring* 中未包含的选项字符，或者检测到有缺失的参数，并且 *optstring* 的第一个字符不是冒号 (**:**) ，则返回问号 (**?**)。

否则，在分析了所有命令行选项后， **getopt()** 将返回 −1。

外部语言环境影响

语言环境

LC_CTYPE 类别可确定将选项字母解释为单字节字符和（或）多字节字符。

国际代码集支持

支持单字节字符代码集和多字节字符代码集。

错误

如果出现以下情况， **getopt()** 将会失败：

[EILSEQ] 处理选项的过程中遇到无效的多字节字符序列。

举例

以下代码段说明如何处理一个命令的参数，该命令可以带有互斥选项 **a** 和 **b** ，以及选项 **f** 和 **o** ，这两组选项都需要具有参数：

```
#include <stdio.h>
#include <unistd.h>
main (int argc, char *argv[])
{
   int c;
   int bflg, aflg, errflg;
   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 arguments */
            fprintf(stderr, "Option -%c requires an argument\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], 4)) {
      .
      .
      .
   }
```

警告

选项可以是除冒号 (**:**) 、问号 (**?**) 或 NULL (**\0**) 以外的任何 ASCII 字符。

另请参阅

getopt(1)、 thread_safety(5)。

符合的标准

getopt()：AES、SVID2、SVID3、XPG2、XPG3、XPG4、POSIX.2

optarg: AES、SVID2、SVID3、XPG2、XPG3、XPG4、POSIX.2

opterr: AES、SVID2、SVID3、XPG2、XPG3、XPG4、POSIX.2

optind: AES、SVID2、SVID3、XPG2、XPG3、XPG4、POSIX.2

optopt: AES、SVID3、XPG4、POSIX.2

名称

getpass( ) - 读取口令

概要

#include <unistd.h>

char *getpass(const char *prompt);

说明

在利用以 null（空）结尾的字符串 *prompt* 提示标准错误输出且禁用回显后， **getpass()** 可从文件 **/dev/tty** 中进行读取，直到遇到换行符或 EOF。返回的指针指向一个最多 8 个字符且以 null（空）结尾的字符串。如果无法打开 **/dev/tty** ，则返回 NULL （空）指针。返回值之前如果有中断，就会终止输入并向调用程序发送中断信号。

警告

返回值指向其内容被每次调用所覆盖的静态数据。

过时的接口

getpass() 很快就会过时。

文件

/dev/tty

另请参阅

crypt(3C)、 thread_safety(5)。

符合的标准

getpass()：AES、SVID2、SVID3、XPG2、XPG3、XPG4

名称

getprdfent()、getprdfnam()、setprdfent()、endprdfent()、putprdfnam() - 操作受信任系统的系统缺省数据库条目

概要

```
#include <sys/types.h>
#include <hpsecurity.h>
#include <prot.h>

struct pr_default *getprdfent(void);

struct pr_default *getprdfnam(const char *name);

void setprdfent(void);

void endprdfent(void);

int putprdfnam(const char *name, struct pr_default *pr);
```

说明

getprdfent 和 **getprdfnam** 都可以返回指向带有以下结构的对象的指针，该结构包含系统缺省数据库中某个行的组成字段。数据库中的每条记录都包含在 **<prot.h>** 头文件中声明的 *pr_default* 结构：

```
struct  system_default_fields  {
    time_t  fd_inactivity_timeout ;
    char    fd_boot_authenticate ;

} ;

struct  system_default_flags {
    unsigned short
        fg_inactivity_timeout:1,
        fg_boot_authenticate:1,

} ;
struct  pr_default  {
    char                    dd_name[20] ;
    char                    dg_name ;
    struct pr_field           prd ;
    struct pr_flag            prg ;
    struct t_field            tcd ;
    struct t_flag             tcg ;
    struct dev_field            devd ;
    struct dev_flag             devg ;
    struct system_default_fields    sfld ;
```

```
    struct system_default_flags    sflg ;
} ;
```

当前，名称 **default** 只引用了系统缺省数据库中的一个条目。

系统缺省数据库包含受保护口令数据库、终端控制数据库和设备分配数据库中所有参数的缺省值，以及可配置的系统级参数。相应的手册条目中介绍了其他数据库的字段。 *fd_inactivity_timeout* 是受信任系统上的会话终止之前所经过的秒数。

fd_boot_authenticate 是一个布尔型标记，指示在系统开始操作之前，是否对授权用户进行验证。

首次调用 **getprdfent** 后，它将返回指向数据库中第一个 *pr_default* 结构的指针。以后，将返回指向数据库中下一个 *pr_default* 结构的指针，从而可以通过连续调用来搜索数据库（只支持一个条目）。

getprdfnam 从文件的起始位置开始搜索，直到找到与 *name* 匹配的缺省条目，然后返回指向特定结构的指针（该条目就是在此结构中找到的）。如果在读取时遇到文件结束标记或错误，这些函数将返回 NULL 指针。当前，所有程序都是通过调用 **getprdfnam** （条目名为 **default** ）来访问缺省数据库。

调用 **setprdfent** 可以回卷缺省值控制文件，以便能够执行重复搜索。处理完成后，可以调用 **endprdfent** 关闭数据库。

putprdfnam 可将一个新的或替换的缺省值控制条目 *pr* （带关键字 *name* ）置于数据库中。如果 *prg.fd_name* 字段为 0，则通过系统缺省数据库删除请求的条目。在执行所有更新操作时， **putprdfnam** 将锁定数据库，更新结束或尝试失败后，还会执行 **endprdfent** 。

注释

getprdfent 和 **getprdfnam** 返回的值将引用调用这些例行程序后所覆盖的结构。要检索某个条目，应先对其修改，再在数据库中替换，再使用结构赋值复制此条目，然后将修改的缓冲区提供给 **putprdfnam** 。

必须使用 **-lsec** 编译使用这些例行程序的程序。

实际应用信息

在多线程的应用程序中，仅从一个专用线程调用这些例行程序是安全的。这些例行程序不是 POSIX.1c 异步取消安全的，也不是异步信号安全的。

返回值

遇到 **EOF** 或错误时， **getprdfent** 和 **getprdfnam** 将返回 NULL 指针。如果 **putprdfnam** 无法添加或更新条目，将返回 0。

警告

不要删除系统缺省值条目。

HP-UX 11i v3 是支持受信任系统功能的最后一个版本。

文件

/tcb/files/auth/system/default 系统缺省数据库

另请参阅

authcap(4)、default(4)、getprpwent(3)、getprtcent(3)、getdvagent(3)。

名称

getprotoent()、 getprotobynumber()、 getprotobyname()、 setprotoent()、 endprotoent() - 获取、设置或结束协议条目

概要

```
#include <netdb.h>

struct protoent *getprotoent(void);

struct protoent *getprotobyname(const char *name);

struct protoent *getprotobynumber(int proto);

int setprotoent(int stayopen);

int endprotoent(void);
```

仅适用于 **_XOPEN_SOURCE_EXTENDED**

```
void setprotoent(int stayopen);

void endprotoent(void);
```

说明

getprotoent() 、 **getprotobyname()** 和 **getprotobynumber()** 函数中的每一个都可以返回指向 *protoent* 类型的结构的指针，该结构包含网络协议数据库 **/etc/protocols** 中某个行的组成字段。

该结构成员包括:

p_name 协议的正式名称。

p_aliases 协议备选名称的以 null 结尾的列表。

p_proto 协议号。

函数的行为如下:

getprotoent() 读取文件的下一行，必要时打开文件。

setprotoent() 打开并回卷文件。如果 *stayopen* 标记为非零值，则每次调用 **getprotoent()** （无论是直接调用还是通过其他某种 **getproto*** 调用间接调用）后不关闭协议数据库。

endprotoent() 关闭文件。

getprotobyname(), **getprotobynumber()**
其中每个都会按顺序从文件的开头开始搜索，直到找到匹配的协议名（无论是正式名还是别名）或协议号，或者直到遇到 EOF。

在多线程应用程序中， **getprotoent()** 、 **getprotobyaddr()** 和 **getprotobyname()** 使用每次调用可复用的特定线程存储。返回值 **struct**protoent 应该对每个线程是唯一的，线程执行下一次 **getproto*()** 调用之前，应该按需要保存该返回值。

对于多线程应用程序中的枚举，枚举中的位置是所有线程共享的进程间属性。可以在多线程应用程序中使用

setprotoent() ，但它会重置所有线程的枚举位置。 **getprotoent()** 可枚举协议条目：连续调用 **getprotoent()** 不是返回连续的协议条目，就是返回 NULL。如果多个线程交错调用 **getprotoent()** ，这些线程将枚举协议数据库的脱节子集。

如果系统运行的是网络信息服务 (NIS)，那么 **getprotobyname()** 和 **getprotobynumber()** 将分别从 NIS 服务器（请参阅 *ypserv*(1M) 和 *ypfiles*(4) ）获取协议信息。

基于名称服务交换的操作

库例行程序 **getprotobyname()** 、 **getprotobynumber()** 、 **getprotoent()** 及其重入对等例行程序可以内部调用名称服务交换，以访问 **/etc/nsswitch.conf** 文件（请参阅 *nsswitch.conf*(4) ）中配置的“protocols”数据库查找策略。查找策略定义用于解析协议名称和编号的受支持名称服务的顺序和条件。

返回值

当 **getprotoent()** 、 **getprotobyname()** 和 **getprotobynumber()** 遇到 EOF，或者无法打开 **/etc/protocols** 时，将返回空指针 (0)。

过时的接口

```
int getprotoent_r(struct protoent *result,
                  struct protoent_data *buffer);

int getprotobyname_r(const char *name,
                     struct protoent *result,
                     struct protoent_data *buffer);

int getprotobynumber_r(int proto,
                       struct protoent *result,
                       struct protoent_data *buffer);

int setprotoent_r(int stayopen, struct protoent_data *buffer);

int endprotoent_r(struct protoent_data *buffer);
```

上述重入接口已从 **libc** 移至 **libd4r** 。为支持现有的应用程序而包括了这些接口，将来的发行版可能会删除这些接口。新的多线程应用程序应该使用不带 **_r** 的常规 API。

重入接口与其常规对等接口（不带 **_r** 后缀）执行的操作相同。但是，需要为 **getprotoent_r()** 、 **getprotobyname_r()** 和 **getprotobyport_r()** 传递 **struct protoent** 参数的地址，同时它们将在此提供的参数上存储结果的地址。文件 **<netdb.h>** 中定义的一个附加参数，即 *struct protoent_data* 的地址，不能是空指针。

如果操作不成功，或者在执行 **getprotoent_r()** 时，到达了服务列表的结束位置，重入例行程序将返回 **-1** 。否则返回 **0** 。

举例

以下代码片断将统计协议条目数：

```
int count = 0;
```

```
(void) setprotoent(0);
while (getprotoent() != NULL)
    count++;
(void) endprotoent();
```

警告

由于这些函数的实现在运行时采用共享对象的动态加载和链接，因此无法静态链接使用了本联机帮助页所述接口的程序。

作者

getprotoent() 由 Sun Microsystems Inc. 开发。

文件

/etc/protocols

另请参阅

ypserv(1M)、 nsswitch.conf(4)、 protocols(4)、 ypfiles(4)、 thread_safety(5)。

符合的标准

getprotoent()： XPG4

名称

getprpwent()、getprpwuid()、getprpwnam()、getprpwaid()、setprpwent()、endprpwent()、putprpwnam() - 操作受保护口令数据库条目（仅适用于受信任系统）

概要

```
#include <sys/types.h>
#include <hpsecurity.h>
#include <prot.h>

struct pr_passwd *getprpwent(void);

struct pr_passwd *getprpwuid(uid_t uid);

struct pr_passwd *getprpwnam(const char *name);

struct pr_passwd *getprpwaid(aid_t aid)

void setprpwent(void);

void endprpwent(void);

int putprpwnam(const char *name, struct pr_passwd *pr);
```

说明

getprpwent 、 **getprpwuid** 、 **getprpwaid** 和 **getprpwnam** 中的每一个都可以返回指向 *pr_passwd* 结构的指针，该结构包含受保护口令数据库中某个行的组成字段。数据库中的每行都包含在 **<prot.h>** 头文件中声明的 *pr_passwd* 结构：

```
struct pr_field  {
    /* Identity: */
    char    fd_name[9];          /* uses 8 character maximum(and NULL) from utmp */
    uid_t   fd_uid;              /* uid associated with name above */
    char    fd_encrypt[xxx];     /* encrypted password */
    char    fd_owner[9];         /* if a pseudo-user, the user accountable */
    char    fd_boot_auth;        /* boot authorization */
    mask_t  fd_auditcntl;        /* reserved */
    mask_t  audit_reserve1;      /* reserved */
    mask_t  fd_auditdisp;        /* reserved */
    mask_t  audit_reserve2;      /* reserved */
    aid_t   fd_pw_audid;         /* audit ID */
    int     fd_pw_audflg;        /* audit flag */

    /* Password maintenance parameters: */
    time_t  fd_min;              /* minimum time between password changes */
    int     fd_maxlen;           /* maximum length of password */
    time_t  fd_expire;           /* expiration time duration in secs */
```

```
    time_t  fd_lifetime;            /* account death duration in seconds */
    time_t  fd_schange;             /* last successful change in secs past 1/1/70 */
    time_t  fd_uchange;             /* last unsuccessful change */
    time_t  fd_acct_expire;         /* absolute account lifetime in seconds */
    time_t  fd_max_llogin;          /* max time allowed between logins */
    time_t  fd_pw_expire_warning; /* password expiration warning */
    uid_t   fd_pswduser;            /* who can change this user's password */
    char    fd_pick_pwd;            /* can user pick his own passwords? */
    char    fd_gen_pwd;             /* can user get passwords generated for him? */
    char    fd_restrict;            /* should generated passwords be restricted? */
    char    fd_nullpw;              /* is user allowed to have a NULL password? */
    uid_t   fd_pwchanger;           /* who last changed user's password */
    long    fd_pw_admin_num;    /* password generation verifier */
    char    fd_gen_chars;           /* can have password of random ASCII? */
    char    fd_gen_letters;         /* can have password of random letters? */
    char    fd_tod[AUTH_TOD_SIZE];    /* times when user may login */

    /* Login parameters: */
    time_t  fd_slogin;              /* last successful login */
    time_t  fd_ulogin;              /* last unsuccessful login */
    char    fd_suctty[14];          /* tty of last successful login */
    int     fd_nlogins;             /* consecutive unsuccessful logins */
    char    fd_unsuctty[14];        /* tty of last unsuccessful login */
    int     fd_max_tries;           /* maximum unsuc login tries allowed */
    char    fd_lock;                /* Unconditionally lock account? */
};

struct pr_flag  {
    unsigned short
        /* Identity: */
        fg_name:1,                  /* Is fd_name set? */
        fg_uid:1,                   /* Is fd_uid set? */
        fg_encrypt:1,               /* Is fd_encrypt set? */
        fg_owner:1,                 /* Is fd_owner set? */
        fg_boot_auth:1,             /* Is fd_boot_auth set? */
        fg_pw_audid:1,              /* Is fd_auditcntl set? */
        fg_pw_audflg:1,             /* Is fd_auditdisp set? */

        /* Password maintenance parameters: */
        fg_min:1,                   /* Is fd_min set? */
        fg_maxlen:1,                /* Is fd_maxlen set? */
```

```
        fg_expire:1,                /* Is fd_expire set? */
        fg_lifetime:1,              /* Is fd_lifetime set? */
        fg_schange:1,               /* Is fd_schange set? */
        fg_uchange:1,               /* Is fd_fchange set? */
        fg_acct_expire:1,           /* Is fd_acct_expire set? */
        fg_max_llogin:1,            /* Is fd_max_llogin set? */
        fg_pw_expire_warning:1,     /* Is fd_pw_expire_warning set? */
        fg_pswduser:1,              /* Is fd_pswduser set? */
        fg_pick_pwd:1,              /* Is fd_pick_pwd set? */
        fg_gen_pwd:1,               /* Is fd_gen_pwd set? */
        fg_restrict:1,              /* Is fd_restrict set? */
        fg_nullpw:1,                /* Is fd_nullpw set? */
        fg_pwchanger:1,             /* Is fd_pwchanger set? */
        fg_pw_admin_num:1,          /* Is fd_pw_admin_num set? */
        fg_gen_chars:1,             /* Is fd_gen_chars set? */
        fg_gen_letters:1,           /* Is fd_gen_letters set? */
        fg_tod:1,                   /* Is fd_tod set? */

        /* Login parameters: */
        fg_slogin:1,                /* Is fd_slogin set? */
        fg_suctty: 1,               /* is fd_suctty set ? */
        fg_unsuctty: 1,             /* is fd_unsuctty set ? */
        fg_ulogin:1,                /* Is fd_ulogin set? */
        fg_nlogins:1,               /* Is fd_nlogins set? */
        fg_max_tries:1,             /* Is fd_max_tries set? */
        fg_lock:1;                  /* Is fd_lock set? */

};

struct pr_passwd  {
    struct pr_field ufld;           /* user specific fields */
    struct pr_flag uflg;            /* user specific flags */
    struct pr_field sfld;           /* system wide fields */
    struct pr_flag sflg;            /* system wide flags */
};
```

受保护口令数据库可存储用户的验证配置文件。用户特定条目中的 *pr_passwd* 结构引用特定于用户的参数。系统缺省数据库中的 *pr_passwd* 结构可以设置不存在针对特定用户的替换项时所使用的参数。

用户特定条目是通过 *fd_name* 字段键入的，它是对用户的 **/etc/passwd** 条目的交叉引用。 *fd_uid* 字段必须与该文件中的 UID 匹配。 *fd_encrypt* 字段是加密的口令。该口令以八个字符的段进行加密，因此该字段的大小为加密段

中字符数的倍数（**AUTH_CIPHERTEXT_SIZE** 宏）。

fd_owner 是构成帐户的用户名。当系统缺省文件指定需要引导授权时，将使用 *fd_boot_auth* 字段。 **init** 命令提示输入用户名和口令。如果验证成功，该字段中的某个值将允许用户继续系统引导的过程。

fd_min 是以秒为单位的时间，经过这个时间后用户才可以更改口令。 *fd_maxlen* 是用户的最大口令长度（以字符数表示）。 *fd_expire* 是用户口令过期之前的时间，以秒为单位。 *fd_lifetime* 是口令失效之前所经过的秒数。如果口令失效，帐户被视为已锁定。

fd_schange 和 *fd_uchange* 可记录上一次口令更改成功和失败的时间。

fd_acct_expire 字段指定帐户可以使用的绝对时间段（以秒为单位）。可以指定一个绝对到期日期，该日期随后可转换为该字段中存储的秒。这与 *fd_expire* 有所不同， *fd_acct_expire* 可指定一个绝对到期日期，而 *fd_expire* 可通过更改每个口令完成重置。

fd_max_llogin 指定自上次登录以后至帐户锁定之前，所允许的最大时间（以秒为单位）。 *fd_pw_expire_warning* 为 *fd_expire* 结束之前的时间（以秒为单位），到此时间系统将警告用户，口令即将过期。 *fd_pswduser* 存储允许更改帐户口令的用户的 ID。通常，该用户为帐户拥有者。

接下来的标记字段可控制口令生成。如果设置了 *fd_pick_pwd* ，则允许用户选择自己的口令。如果设置了 *fd_nullpw* ，则允许在使用帐户时不必输入口令。 *fd_gen_pwd* 允许使用该帐户口令的随机可发音口令生成器。 *fd_gen_chars* 和 *fd_gen_letters* 允许口令生成器生成由随机可输出字符和随机字母组成的口令，这两个组成部分都不容易记住。口令更改软件允许用户从其帐户可用的任何选项中作选择。必须设置这三个字段（ fd_gen_pwd 、 *fd_gen_chars* 或 *fd_gen_letters* ）中的一个。

fd_pwchanger 是上一次更改用户帐户口令的用户的 ID ，但这个用户不是帐户拥有者。如果设置了 *fd_restrict* ，将在选择帐户口令后执行细节检查，以避免使用词典中显示的回文、用户名、计算机名轮排和单词。

fd_tod 说明符是格式类似于 UUCP 系统文件的字符串，用于指定用户可以登录的时间间隔。

接下来的字段可用于防止登录欺骗行为，及防止列出上次登录的时间和位置。 *fd_slogin* 和 *fd_ulogin* 是上次成功和失败登录尝试的时间戳。 *fd_suctty* 和 *fd_unsuctty* 是终端设备，或发生上次登录尝试的终端或主机的主机名（如果支持）。

fd_nlogins 是自上一次成功登录以来，失败登录尝试的次数。每次成功登录后会将其重置为零。 *fd_max_tries* 是帐户被视为锁定之前的失败尝试次数。

fd_lock 表示是否在帐户上设置了管理锁。请注意，帐户被视为禁用（锁定）可能是由于 *fd_lock* 没有指明的原因。出现以下一种或多种活动时，帐户被视为禁用（锁定）：

1. 如果口令失效，
2. 如果超过失败尝试的最大次数，
3. 如果设置了管理锁，
4. 如果达到帐户的过期时间，或者
5. 如果超过两次登录之间的时限要求。

如果首先调用 **getprpwent** ，将返回指向数据库中第一个用户 *pr_passwd* 结构的指针；然后，返回指向数据库中下

一个 *pr_passwd* 结构的指针，以便可以使用连续调用来搜索数据库。请注意，将会跳过在 **/etc/passwd** 中没有对应条目的条目。将按照条目在 **/etc/passwd** 中显示的顺序扫描这些条目。

getprpwuid 从数据库的起始位置开始搜索，直到找到与 *uid* 匹配的数字形式的用户 ID ，然后返回指向特定结构的指针（该指针就是在此结构中找到的）。只有使用审核 ID，而不是 UID 时， **getprpwaid** 的功能才类似于 **getprpwuid** 。

getprpwnam 从数据库的起始位置开始搜索，直到找到与 *name* 匹配的登录名，然后返回指向特定结构的指针（该指针就是在此结构中找到的）。如果在读取时遇到文件结束标记或错误，这些函数将返回 NULL 指针。

调用 **setprpwent** 可以回卷受保护口令数据库，以便能够执行重复搜索。处理完成后，如需关闭受保护口令数据库，可以调用 **endprpwent** 。

putprpwnam 可以在数据库中放入一个新的受保护口令条目 *pr* ，或者使用密钥 *name* 替换该条目。如果 *uflg.fg_name* 字段为 0，则通过受保护口令数据库删除请求的条目。在执行所有更新操作时， **putprpwnam** 将锁定数据库，更新结束或尝试失败后，还会执行 **endprpwent** 。

注释

getprpwent 和 **getprpwnam** 返回的值将引用调用这些例行程序后所覆盖的结构。要检索某个条目，应先对其修改，再在数据库中替换，再使用结构赋值复制此条目，然后将修改的缓冲区提供给 **putprpwnam** 。

在支持网络连接的系统上， *fd_suctty* 和 *fd_unsuctty* 字段可以是上一次通过此处远程登录帐户成功或失败的主机的网络地址 ASCII 表示形式。使用 **getdvagnam** （请参阅 *getdvagent*(3) ）可以调查设备的类型，以确定上次成功或失败登录是否使用了某个主机或终端。

必须使用 **-lsec** 编译使用这些例行程序的程序。

如果使用归档文件版本 **libsec** (**libsec.a**) 链接应用程序，则这些例行程序在工作时将独立于名称服务交换。受保护口令数据库只能存在于本地系统上的 **/tcb** 目录中。

getprpwent 假定每个 UID 只有一个名称，同时每个名称只有一个 UID。连续扫描将在多个 UID 的最初两个实例之间循环。

getprpwent 使用 *getpwent*(3C) 例行程序来连续扫描数据库。使用此处所述的任何例行程序（即 * **prp** * 例行程序）后，用户程序对使用 *getpwent*(3C) 例行程序获取的口令条目的引用将会无效。

所有这些例行程序都取决于名称服务交换文件 **/etc/nsswitch.conf** 的配置。这些例行程序使用 **passwd** 数据库的交换。

实际应用信息

在多线程的应用程序中，仅从一个专用线程调用这些例行程序是安全的。这些例行程序不是 POSIX.1c 异步取消安全的，也不是异步信号安全的。用于受保护数据库 API 的名称服务交换数据库为 **passwd** 。例如， **/etc/nsswitch.conf** 中的条目将包含 **passwd: files** 。

返回值

出现 EOF 或错误时， **getprpwent** 、 **getprpwuid** 、 **getprpwaid** 和 **getprpwnam** 将返回 NULL 指针。如果 **putprpwnam** 无法添加或更新条目，则返回 0。

警告

HP-UX 11i v3 是支持受信任系统功能的最后一个版本。

文件

/etc/passwd	系统口令文件
/tcb/files/auth/*/*	受保护口令数据库
/tcb/files/auth/system/default	系统缺省数据库

另请参阅

authcap(4)、getpwent(3C)、getprdfent(3)、prpwd(4)。

名称

getprtcent、getprtcnam、setprtcent、endprtcent、putprtcnam - 操作受信任系统的终端控制数据库条目

概要

```
#include <sys/types.h>
#include <hpsecurity.h>
#include <prot.h>

struct pr_term *getprtcent(void);

struct pr_term *getprtcnam(const char *name);

void setprtcent(void);

void endprtcent(void);

int putprtcnam(const char *name, struct pr_term *pr);
```

说明

getprtcent 和 **getprtcnam** 都可以返回指向带有以下结构的对象的指针，该结构包含终端控制数据库中某个条目的组成字段。数据库中的每个条目都包含在 **<prot.h>** 头文件中声明的 *pr_term* 结构：

```
struct   t_field  {
  char    fd_devname[14];       /* Terminal (or host) name */
  uid_t   fd_uid;               /* uid of last successful login */
  time_t  fd_slogin;            /* time stamp of successful login */
  uid_t   fd_uuid;              /* uid of last unsuccessful login */
  time_t  fd_ulogin;            /* time stamp of unsuccessful login */
  int     fd_nlogins;           /* consecutive failed attempts */
  int     fd_max_tries;         /* maximum unsuc login tries allowed */
  time_t  fd_logdelay;          /* delay between login tries */
  char    fd_lock;              /* terminal locked? */
  int     fd_login_timeout;     /* login timeout in seconds */
};
```

```
struct  t_flag  {
  unsigned short
    fg_devname:1,          /* Is fd_devname set? */
    fg_uid:1,              /* Is fd_uid set? */
    fg_slogin:1,           /* Is fd_stime set? */
    fg_uuid:1,             /* Is fd_uuid set? */
    fg_ulogin:1,           /* Is fd_ftime set? */
    fg_nlogins:1,          /* Is fd_nlogins set? */
    fg_max_tries:1,        /* Is fd_max_tries set? */
    fg_logdelay:1,         /* Is fd_logdelay set? */
    fg_lock:1,             /* Is fd_lock set? */
    fg_login_timeout:1     /* is fd_login_timeout valid? */
    ;
};

struct  pr_term  {
  struct t_field ufld;
  struct t_flag uflg;
  struct t_field sfld;
  struct t_flag sflg;
};
```

系统将最后一次成功登录的用户 ID 和时间（*fd_uid* 和 *fd_slogin*），以及未成功登录的用户 ID 和时间（*fd_uuid* 和 *fd_ulogin*），存储在相应的终端控制数据库条目内。对于每次未成功的登录，系统将递增 *fd_nlogins*，对于每次成功的登录，则会将该字段重置为 0。*fd_max_tries* 字段可限制锁定帐户之前允许的未成功登录次数。也可以应用由非零的 *fd_lock* 字段表示的管理锁。*fd_logdelay* 可存储两次未成功登录尝试之间系统等待的时间（秒），*fd_login_timeout* 可存储从开始验证尝试到结束登录尝试之间的秒数。

请注意，*ufld* 和 *uflg* 引用特定用户的条目，而 *sfld* 和 *sflg* 引用系统缺省值（请参阅 *authcap*(4)）。

getprtcent 或 **getprtcnam** 返回的值表示调用这些例行程序所覆盖的结构。要检索某个条目，应先对其修改，然后在数据库中替换，再使用结构赋值复制此条目，并且将已修改的缓冲区提供给 **putprtcnam**。

首次调用 **getprtcent** 后，它将返回指向数据库中第一个终端 *pr_term* 结构的指针。随后返回指向数据库中下一个 *pr_term* 结构的指针，这样便可以通过连续调用来搜索数据库。**getprtcnam** 从数据库的起始位置开始搜索，直到找到与 *name* 匹配的终端名为止，然后返回指向在其当中找到该终端名的特定结构的指针。如果在读取时遇到文件结束标记或错误，则这些函数将返回 NULL 指针。

调用 **setprtcent** 可以回卷终端控制数据库，以便能够重复搜索。处理完成后，可以调用 **endprtcent** 将终端控制数据库关闭。

putprtcnam 可将一个新的或替换的终端控制条目 *pr*（关键字为 *name*）置于数据库中。如果 *fg_devname* 字段为 0，则通过终端控制数据库删除请求的条目。在执行所有更新操作时，**putprtcnam** 将锁定数据库，更新结束或尝

试失败后，还会执行 **endprtcent** 。

实际应用信息

在多线程的应用程序中，仅从一个专用线程调用这些例行程序是安全的。这些例行程序不是 POSIX.1c 异步取消安全的，也不是异步信号安全的。

返回值

遇到 **EOF** 或错误时， **getprtcent** 和 **getprtcnam** 将返回 NULL 指针。如果 **putprtcnam** 无法添加或更新条目，则将返回 0。

注释

系统支持连接上的 *fd_devname* 字段可以引用某个主机名的 ASCII 表示形式。通过 **getdvagnam** （请参阅 *getdvagent*(3) ）向设备分配数据库查询设备的类型，同时以参数的形式传入终端控制结构的 *fd_devname* 字段，可以确定此引用。这允许计算机，而不是生成会话的设备（通常为伪 tty）执行封锁。

必须使用 **-lsec** 编译使用这些例行程序的程序。

通过系统缺省数据库中的对应字段填写 *sfld* 结构和 *sflg* 结构。这样，该程序就可以轻易地为每个数据库字段（请参阅 *getprpwent* 和 *getdvagent* ）提取特定用户的参数或系统级参数。

警告

HP-UX 11i v3 是支持受信任系统功能的最后一个版本。

文件

/tcb/files/ttys	终端控制数据库
/tcb/files/auth/system/default	系统缺省数据库

另请参阅

getprdfent(3)、authcap(4)、ttys(4)。

名称

getpublickey()、getsecretkey()、publickey() - 检索公用密钥或私用密钥

概要

```
#include <rpc/rpc.h>
#include <rpc/key_prot.h>

int getpublickey(
        const char netname[MAXNETNAMELEN],
        char publickey[HEXKEYBYTES+1]);

int getsecretkey(
        const char netname[MAXNETNAMELEN],
        char secretkey[HEXKEYBYTES+1],
        const char *passwd);
```

说明

getpublickey() 和 **getsecretkey()** 将获取 *netname* 的公用密钥和私用密钥。密钥可来自以下来源之一： **/etc/publickey** 文件（请参阅 *publickey*(4) ）、NIS 映射 **publickey.byname** 或 LDAP 目录中的 **user/host** 条目。在 **/etc/nsswitch.conf** 文件（请参阅 *nsswitch.conf*(4) ）中指定了来源及其查找顺序。

getsecretkey() 中有一个额外参数 *passwd* ，用于对存储在数据库中的加密的私用密钥进行解密。

多线程应用信息

线程安全:	是
取消安全:	是
派生安全:	否
异步取消安全:	否
异步信号安全:	否

在多线程环境中可以安全地调用这些函数。它们可能是取消点，原因是它们调用作为取消点的函数。

在多线程环境中，在 **fork()** 之后和 **exec()** 之前由子进程调用这些函数并不安全。支持异步取消或异步信号的多线程应用程序不应调用这些函数。

返回值

如果上述例行程序成功地查找到密钥，它们将返回 **1** ，否则就会返回 **0** （零）。返回的密钥以 NULL 结尾，为十六进制字符串。如果向 **getsecretkey()** 提供的口令未能解密私用密钥，例行程序将会返回 **1** ，但 *secretkey*[0] 将会设置为 NULL。

警告

HP-UX 11i v2 是支持 NIS+ 的最后一个 HP-UX 版本。建议使用 LDAP 替代 NIS+。 HP 完全支持基于 LDAP 的行业标准命名服务。

另请参阅

secure_rpc(3N)、 nsswitch.conf(4)、 publickey(4)。

《LDAP-UX Client Services Administrator's Guide》
《LDAP-UX Client Services Release Notes》

名称

getpw( ) - 从 UID 获取名称

概要

#include <pwd.h>

int getpw(uid_t uid, char *buf);

说明

getpw() 将搜索口令文件以查找等于 *uid* 的用户 ID 号，将从中找到 *uid* 的口令文件行复制到 *buf* 所指向的数组中，然后返回 0。如果找不到 *uid* ， **getpw()** 就会返回非零值。行以 null（空）结尾。

提供此例行程序的目的只是为了实现与以前系统的兼容性，请勿使用；有关相应的替代例行程序，请参阅 *getpwent*(3C) 。

网络功能

NFS

此例行程序是通过使用 **getpwuid()** （请参阅 *getpwent*(3C) ）实现的，因此会用到 *passwd*(4) 中所述的网络信息服务网络数据库。

返回值

出现错误时， **getpw()** 将返回非零值。

警告

上面的例行程序使用了 **<stdio.h>** ：对于未使用标准 I/O 的程序而言，这会增加程序的大小（可能超出预期值）。

过时的接口

getpw() 很快就会过时。

作者

getpw() 由 AT&T 和 HP 联合开发。

文件

/etc/passwd

另请参阅

getpwent(3C)、 passwd(4)、 thread_safety(5)。

符合的标准

getpw()：XPG2

名称

getpwent()、 getpwuid()、 getpwuid_r()、 getpwnam()、 getpwnam_r()、 setpwent()、 endpwent()、 fgetpwent() - 获取口令文件条目

概要

```
#include <pwd.h>

struct passwd *getpwent(void);

struct passwd *getpwuid(uid_t uid);

int  getpwuid_r(uid_t uid, struct passwd *pwd, char *buffer,
    size_t buflen, struct passwd **result);

struct passwd *getpwnam(const char *name);

int  getpwnam_r(char *name, struct passwd *pwd, char *buffer,
    size_t buflen, struct passwd **result);

void setpwent(void);

void endpwent(void);

struct passwd *fgetpwent(FILE *stream);
```

过时的接口

```
#include <pwd.h>

int  getpwent_r(struct passwd *result, char *buffer, int buflen,
    FILE **pwfp);

void setpwent_r(FILE **pwfp);

void endpwent_r(FILE **pwfp);

int fgetpwent_r(FILE *f, struct passwd *result, char *buffer,
    int buflen);
```

说明

getpwent() 、 **getpwuid()** 和 **getpwnam()** 可用于获取口令条目，及返回指向 **passwd** 结构的某个对象的指针。条目可能来自 **/etc/nsswitch.conf** 文件中指定的 **passwd** 的任何源。请参阅 *nsswitch.conf*(4) 。

passwd 结构是在 **<pwd.h>** 中定义的，它包括以下成员：

```
char   *pw_name;        /* user name */
char   *pw_passwd;      /* encrypted password */
uid_t   pw_uid;         /* user id */
gid_t   pw_gid;         /* group id */
char   *pw_age;         /* password aging */
```

```
char   *pw_comment;        /* unused */
char   *pw_gecos;          /* user fullname, office, extension, homephone*/
char   *pw_dir;            /* initial directory */
char   *pw_shell;          /* initial shell */
aid_t  pw_audid;           /* numerical audit id */
int    pw_audflg;          /* numerical audit flag */
```

不使用 *pw_comment* 字段。有关其他字段的详细信息，请参阅 *passwd*(4) 联机帮助页。

getpwent() 首次调用 **getpwent()** 后，它将返回指向口令数据库中第一个 **passwd** 结构的指针。随后将返回指向数据库中下一个 **passwd** 结构的指针。

setpwent() 能够展开口令数据库，以便反复执行搜索。

endpwent() 可以调用它来指示已完成口令数据库的处理。

getpwuid() 从口令数据库的起始位置开始搜索，直到找到与 *uid* 匹配的数字用户 ID，然后返回指向在其中找到了该 ID 的特定结构的指针。

getpwnam() 从口令数据库的起始位置开始搜索，直到找到与 *name* 匹配的登录名，然后返回指向在其中找到该登录名的特定结构的指针。

fgetpwent() 与上述其他函数不同，它既不使用 **nsswitch.conf** ，也不访问 NIS。该函数返回指向标准 I/O 流 *stream* 中的下一个 **passwd** 结构的指针，该流应处于打开状态，以便进行读取，并且其内容应与 **/etc/passwd** 的格式匹配。

过时的接口

getpwent_r() 、 **setpwent_r()** 、 **endpwent_r()** 、 **fgetpwent_r()** 可获取口令文件条目。

重入接口

getpwuid_r() 和 **getpwnam_r()** 都可以更新 **pwd** 指向的 **struct passwd** ，以及将指向该结构的指针存储在 **result** 指向的位置。该结构必须包含来自用户数据库的带有匹配的 **uid** 或 **name** 的条目。在 **buffer** 参数提供的内存（大小定义为 **buflen** ）中分配 **pwd** 指向的 **passwd** 结构所引用的存储。 **_SC_GETPW_R_SIZE_MAX sysconf()** 参数可确定该缓冲区所需的最大大小。如果出现错误，或者未找到请求的条目，将在 **result** 指向的位置返回 NULL 指针。

注释

如果将储备库设置为 **/etc/nsswitch.conf** 文件中的 **files** ，则 **getpwent()** 、 **getpwuid()** 、 **getpwuid_r()** 、 **getpwnam()** 和 **getpwnam_r()** 调用将返回与在 **/etc/passwd** 文件中出现的完全一样的 **passwd** 结构。在映像标准模式下，这些调用通常在 *pw_passwd* 字段中返回“X”（而不是加密的口令和时限信息）。如果将参数流设置为 **/etc/passwd** ，则这一点同样也适用于 **fgetpwent()** 调用。

安全功能

如果将系统转换为受信任的系统，则不会返回口令、审核 ID 和审核标记。口令将是 **/etc/passwd** 中的缺省 ***** ，审核 ID 和审核标记将设置为 **–1** 。在受信任的系统上，如果没有必要从常规口令文件获取信息，则 **/etc/passwd** 用

户应使用 **getprpwent()** 访问受保护的口令数据库。请参阅 *getprpwent*(3) 和 *getspwent*(3X) 。

putpwent() 只影响 **/etc/passwd** ，同时将忽略口令结构中的审核 ID 和审核标记。必须使用 **putprpwnam()** 修改受保护的口令数据库条目。请参阅 *getprpwent*(3) 。

请参阅关于受信任系统支持的“警告”一节。

返回值

如果在读取过程中遇到文件结束标记或错误， **getpwent()** 、 **getpwuid()** 、 **getpwnam()** 和 **fgetpwent()** 将返回 NULL 指针。如果遇到无效的条目， **fgetpwent()** 将返回 NULL 指针。无效的条目是指没有接在 **/etc/passwd** 结构后面的条目。否则，返回值将指向包含有效的 **passwd** 结构的内部静态区。

getpwuid_r() 和 **getpwnam_r()** 成功时返回零。否则，返回一个错误编号以指明错误。

错误

如果出现下列任意情况， **getpwent()** 、 **getpwuid()** 、 **getpwnam()** 和 **fgetpwent()** 将会失败：

[EIO] 发生了 I/O 错误。

[EMFILE] 当前调用进程中打开了 *OPEN_MAX* 描述符。

[ENFILE] 当前系统中打开了所允许的最大数目的文件。

在下列情况下， **getpwnam_r()** 和 **getpwuid_r()** 函数将会失败：

[ERANGE] 通过 *buffer* 和 *bufsize* 提供的存储大小不足以包含由所得到的 **passwd** 结构引用的数据

警告

getpwent() 、 **getpwuid()** 、 **getpwnam()** 和 **fgetpwent()** 返回的值指向每次调用其中的任何函数都会被覆盖的单一静态区，因此必须通过复制来保存此值。

以下字段存在规定的数字限制：

- 用户 ID 是介于 -2 和 **UID_MAX** （包括此限值）之间的整数值。
- 组 ID 是介于 0 和 **UID_MAX** （包括此限值）之间的整数值。

getpwuid_r() 和 **getpwnam_r()** 的用户应注意，现在这些接口符合 POSIX.1c 标准。 **getpwent_r()** 、 **setpwent_r()**、 **endpwent_r()** 和 **fgetpwent_r()** 是过时的接口。为了与现有的 DCE 应用程序兼容，还支持这些接口以及 **getpwuid_r()** 和 **getpwnam_r()** 的早期原型。

接口 **getpwuid()** 、 **getpwnam()** 、 **getpwent()** 、 **setpwent()** 、 **endpwent()** 、 **fgetpwent()** 、 **getpwuid_r()** 和 **getpwnam_r()** 使用动态名称服务交换。请参阅 *nsswitch.conf*(4) 。无法对使用这些接口的应用程序进行完全归档绑定。

HP-UX 11i v3 是支持受信任系统功能的最后一个版本。

举例

以下代码片断将输出在此终端上登录的用户的名称和 uid：

```
struct passwd pwd;
struct passwd *result;
char logBuffer [1024];
char pwdBuffer [1024];

if (getlogin_r (loginBuffer, 1024) == 0)
 if (getpwnam_r (logBuffer, &pwd, pwdBuffer, 1024, &result) == 0)
  printf ("Name = %s; uid = %d\n", pwd.pw_name, pwd.pw_uid);
```

相关内容

NFS

文件

/var/yp/*domainname***/passwd.byname**

/var/yp/*domainname***/passwd.byuid**

/var/nis/*hostname***/passwd.org_dir**

作者

getpwent() 、 **getpwuid()** 、 **getpwnam()** 、 **setpwent()** 、 **endpwent()** 和 **fgetpwent()** 由 Sun 和 HP 联合开发。

文件

/etc/passwd 系统口令文件

另请参阅

ypcat(1)、 cuserid(3S)、 getgrent(3C)、 getlogin(3C)、 getprpwent(3)、 getspwent(3X)、 stdio(3S)、 putpwent(3C)、 nsswitch.conf(4)、 passwd(4)、 limits(5)、 thread_safety(5)。

符合的标准

getpwent()：SVID2、SVID3、XPG2

endpwent()：SVID2、SVID3、XPG2

fgetpwent()：SVID2、SVID3、XPG2

getpwnam()：AES、SVID2、SVID3、XPG2、XPG3、XPG4、FIPS 151-2、POSIX.1

getpwuid()：AES、SVID2、SVID3、XPG2、XPG3、XPG4、FIPS 151-2、POSIX.1

setpwent()：SVID2、SVID3、XPG2

名称

getresuid、getresgid - 获取实际、有效且已保存的用户 ID 或组 ID

概要

#include <unistd.h>

int getresuid (uid_t *ruid, uid_t *euid, uid_t *suid);
int getresgid (gid_t *rgid, gid_t *egid, gid_t *sgid);

说明

getresuid 和 **getresgid** 将返回当前进程实际、有效且已保存的用户 ID 或组 ID。

返回值

出现错误时，返回值为 -1，并相应设置 **errno** 。成功时，返回值为 0。

错误

[EFAULT] 某个参数所指定的地址位于调用程序的地址空间以外。

另请参阅

getuid(2)。

名称

getrpcent()、 getrpcbyname()、 getrpcbynumber()、 setrpcent()、 endrpcent() - 获取 RPC 条目

概要

cc [*flag*]... *file* ... **-lnsl** [*library*]...

#include <netdb.h>

struct rpcent *getrpcbyname(char *_name_);

struct rpcent *getrpcbynumber(int _number_);

struct rpcent *getrpcent(void);

void setrpcent(int _stayopen_);

void endrpcent();

说明

这些函数用于获取 RPC （远程过程调用）服务的条目。条目可能来自 **/etc/nsswitch.conf** 文件（请参阅 *nsswitch.conf*(4) ）中指定的 **rpc** 的任何源。

getrpcbyname() 可搜索带有 *name* 参数指定的 RPC 服务名的条目。

getrpcbynumber() 可搜索带有 RPC 程序编号 *number* 的条目。

函数 **setrpcent()** 、 **getrpcent()** 和 **endrpcent()** 用于枚举数据库中的 RPC 条目。

setrpcent() 可将枚举设置（或重置）到 RPC 条目集的开头。首次调用 **getrpcent()** 之前，应调用该函数。调用 **getrpcbyname()** 和 **getrpcbynumber()** 将使枚举位置处于不确定的状态。如果 *stayopen* 标记为非零值，则在后续调用 **endrpcent()** 之前，系统可以保留已分配的资源，例如打开文件描述符。

连续调用 **getrpcent()** 将返回连续的条目，或返回表示枚举末尾的 NULL。

可以调用 **endrpcent()** ，以表示调用方不再继续执行 RPC 条目检索操作；然后系统可取消分配所使用的资源。调用 **endrpcent()** 后，仍然允许进程调用其他的 RPC 条目检索函数，但这种操作可能效率较低。

多线程应用信息

线程安全:	是
取消安全:	是
派生安全:	否
异步取消安全:	否
异步信号安全:	否

在多线程环境中可以安全地调用这些函数。它们可能是取消点，原因是它们调用作为取消点的函数。

在多线程环境中，在 **fork()** 之后和 **exec()** 之前由子进程调用这些函数并不安全。支持异步取消或异步信号的多线程应用程序不应调用这些函数。

getrpcbyname() 、 **getrpcbynumber()** 和 **getrpcent()** 使用每次调用可再次使用的特定线程存储。应该对于每个线

程，返回值 **struct rpcent** 应是唯一的，并且在线程执行下一次 **getrpc*()** 调用之前，如果需要应该保存该返回值。

对于多线程应用程序中的枚举，枚举中的位置是所有线程共享的进程级属性。 **setrpcent()** 可用于多线程应用程序，但它会重置所有线程的枚举位置。如果多个线程交错调用 **getrpcent()** ，则这些线程将枚举 RPC 条目数据库的脱节子集。

返回值

RPC 条目是由 **<netdb.h>** 中定义的 **struct rpcent** 结构表示的：

```
struct rpcent {
  char *r_name;      /* name of this rpc service */
  char **r_aliases;  /* zero-terminated list of alternate names */
  int r_number;      /* rpc program number */
};
```

如果函数 **getrpcbyname()** 和 **getrpcbynumber()** 成功查找到所请求的条目，则将各自返回指向 **struct rpcent** 的指针；否则，将返回 NULL。

如果函数 **getrpcent()** 成功枚举了一个条目，则返回指向 **struct rpcent** 的指针；否则，将返回表示枚举的末尾的 NULL。

警告

由于这些函数的实现在运行时采用共享对象的动态加载和链接，因此无法静态链接使用了本联机帮助页所述接口的程序。

作者

getrpcent() 由 Sun Microsystems, Inc. 开发。

文件

/etc/rpc
/etc/nsswitch.conf

另请参阅

rpcinfo(1M)、rpc(4)、nsswitch.conf(4)。

名称

getrpcport( ) - 获取 RPC 端口号

概要

u_short getrpcport(char ****host*,**
rpcprog_t *prognum***,**
rpcvers_t *versnum***,**
rpcprot_t *proto***);**

说明

getrpcport() 返回 RPC 程序 *prognum* 的版本 *versnum* 的端口号，该 RPC 程序在 *host* 上运行，并使用协议 *proto*。如果 RPC 系统未能与远程 portmap 服务通信、未注册与 *prognum* 关联的程序或者程序与端口之间不存在映射，则该函数将返回 **0** 。如果 *prognum* 已注册，但不具有版本 *versnum* ，则它返回最后的注册 (*prognum*, *proto*) 对。

该例行程序只是为了实现向后兼容而存在。增强功能由 **rpcb_getaddr()** 提供（请参阅 **rpcbind(3N)** ）。

警告

调用该例行程序的用户应用程序必须与 **/usr/lib/librpcsvc.a** 建立链接。例如，

cc my_source.c -lrpcsvc

作者

getrpcport() 由 Sun Microsystems, Inc. 开发。

另请参阅

portmap(1M)。

名称

gets()、 fgets() - 从流获取字符串

概要

#include <stdio.h>

char *gets(char *s);

char *fgets(char *__restrict s, int n, FILE *__restrict stream);

过时的接口

char *fgets_unlocked(char *s, int n, FILE *stream);

说明

gets() 将字符从标准输入流 **stdin** 读入 *s* 所指向的数组，直到遇到换行符或文件结束标记为止。换行符被忽略，字符串以空字符结束。

fgets() 将字符从 *stream* 读入 *s* 所指向的数组，直到读取 *n*–1 字符、换行符（传输到 *s* ）或遇到文件结束标记为止。字符串以空字符结束。

过时的接口

fgets_unlocked() 从流获取字符串。

实际应用信息

gets() 或 **fgets()** 应用于流之后，此流就成为面向字节的（请参阅 *orientation*(5) ）。

返回值

成功完成后， **fgets()** 、 **fgets_unlocked()** 和 **gets()** 都返回 *s* 。如果流位于文件末尾，则设置此流的文件结束指示器，并返回一个空指针。

当对应于打开流的文件在到达文件结尾后扩展时，对这些函数的任何后续调用都将成功，并且文件结束指示符将保持为已设置状态。但是，在 UNIX2003 标准环境中（请参阅 *standards*(5) ），这些函数将返回空指针，并且文件结束指示符仍将保持为已设置状态。

如果发生读取错误，将设置流的错误指示符，并设置 **errno** 以指明错误，同时返回一个空指针。

可以使用 **ferror()** 和 **feof()** 来区分错误条件和文件结束条件。

错误

如果需要将数据读入 *stream* 的缓冲区，则 **fgets()** 、 **fgets_unlocked()** 和 **gets()** 将失败，并且：

[EAGAIN] 为文件描述符的基础 *stream* 设置了 **O_NONBLOCK** 标记，并且将在读取操作中延迟进程。

[EBADF] 文件描述符的基础 *stream* 不是为读取而打开的有效文件描述符。

[EINTR] 由于接收到信号，或未传输数据，抑或实现未报告此文件的部分传输情况，因此终止了读取操作。

[EIO] 此进程是后台进程的成员，正尝试从其控制终端读取数据，进程可能忽略或阻塞 **SIGTTIN** 信号；或者进程的进程组被孤立。

可以通过基础 **read()** 函数（请参阅 *read*(2) ）设置其他的 **errno** 值。

警告

fgets_unlocked() 是仅支持与现有的 DCE 应用程序兼容的过时接口。新的多线程应用程序应该使用 **fgets()** 。

另请参阅

ferror(3S) 、 flockfile(3S) 、 fopen(3S) 、 fread(3S) 、 getc(3S) 、 puts(3S) 、 scanf(3S) 、 orientation(5) 、 standards(5)、 thread_safety(5)、 glossary(9)。

符合的标准

gets()：AES、SVID2、SVID3、XPG2、XPG3、XPG4、FIPS 151-2、POSIX.1 和 ANSI C

fgets()：AES、SVID2、SVID3、XPG2、XPG3、XPG4、FIPS 151-2、POSIX.1 和 ANSI C

名称

getservent()、 getservbyport()、 getservbyname()、 setservent()、 endservent() - 获取、设置或结束服务条目

概要

```
#include <netdb.h>

struct servent *getservent(void);

struct servent *getservbyname(const char *name,
                const char *proto);

struct servent *getservbyport(int port, const char *proto);

int setservent(int stayopen);

int endservent(void);

_XOPEN_SOURCE_EXTENDED only
void setservent(int stayopen);
void endservent(void);
```

说明

getservent() 、 **getservbyname()** 和 **getservbyport()** 函数中的每一个都可以返回指向 *servent* 类型的结构的指针，该结构包含网络服务数据库 **/etc/services** 中某个行的组成字段。

该结构成员包括：

s_name 服务的正式名称。

s_aliases 服务备用名称的以 null 结尾的列表。

s_port 服务所在的端口号。

s_proto 与服务通信时使用的协议的名称。

函数的行为如下：

getservent() 读取文件的下一行，必要时打开文件。

setservent() 打开并回卷文件。如果 *stayopen* 标记为非零值，则每次调用 **getservent()**（无论是直接调用还是通过其他某个 **getserv*** 调用间接调用）后不关闭服务数据库。

endservent() 关闭文件。

getservbyname(), **getservbyport()**
其中每个都会按顺序从文件的开头开始搜索，直到找到匹配的服务名（无论是正式名还是别名）或端口号为止，或者直到遇到 EOF 为止。如果同时提供了非空协议名（例如 **tcp** 或 **udp** ），则搜索操作还必须匹配协议。

如果系统运行的是网络信息服务 (NIS)，那么 **getservbyname()** 将从 NIS 服务器（请参阅 *ypserv*(1M) 和

ypfiles(4)）获取服务信息。

在多线程应用程序中， **getservent()** 、 **getservbyaddr()** 和 **getservbyname()** 使用每次调用可再使用的特定线程存储。对于每个线程，返回值 **struct servent** 应该是唯一的，并且在线程执行下一次 **getserv*()** 调用之前，应该按需要保存该返回值。

对于多线程应用程序中的枚举，枚举中的位置是所有线程共享的进程级属性。 **setservent()** 可用于多线程应用程序中，但它会重置所有线程的枚举位置。如果多个线程交错调用 **getservent()** ，则这些线程将枚举服务数据库的脱节子集。

基于名称服务交换的操作

库例行程序 **getservbyname()** 、 **getservbyport()** 、 **getservent()** 及其重入例行程序可以内部调用名称服务交换，以访问 **/etc/nsswitch.conf** 文件（请参阅 *nsswitch.conf*(4) ）中配置的“services”数据库查找策略。查找策略定义用于解析服务名和端口的受支持名称服务的顺序和条件。

过时的接口

```
int getservent_r(struct servent *result,
                 struct servent_data *buffer);

int getservbyname_r(const char *name,
                    const char *proto,
                    struct servent *result,
                    struct servent_data *buffer);

int getservbyport_r(int port,
                    const char *proto,
                    struct servent *result,
                    struct servent_data *buffer);

int setservent_r(int stayopen, struct servent_data *buffer);

int endservent_r(struct servent_data *buffer);
```

上述重入接口已从 **libc** 移至 **libd4r** 。包括这些接口是为支持现有的应用程序，将来的版本中可能会删除这些接口。新的多线程应用程序应该使用不带 **_r** 后缀的常规 API。

重入接口执行与其常规对等接口（不带 **_r** 后缀）相同的操作。但是，应向 **getservent_r()** 、 **getservbyname_r()** 和 **getservbyport_r()** 传递 **struct servent** 的地址，同时它们将在所提供的参数中存储结果的地址。文件 **<netdb.h>** 中定义的一个附加参数，即 *struct servent_data* 的地址，不能是空指针。

如果操作不成功，或者在执行 **getservent_r()** 时，到达了服务列表的结尾，则重入例行程序将返回 **–1** 。否则返回 **0** 。

返回值

当 **getservent()** 、 **getservbyname()** 和 **getservbyport()** 遇到 EOF，或者无法打开 **/etc/services** 时，将返回空指针 (0)。

举例

以下代码片断将统计服务条目数：

```
int count = 0;

(void) setservent(0);
while (getservent() != NULL)
    count++;
(void) endservent();
```

警告

由于这些函数的实现在运行时采用共享对象的动态加载和链接，因此无法静态链接使用了本联机帮助页所述接口的程序。

作者

getservent() 由 Sun Microsystems Inc. 开发。

文件

/etc/services

另请参阅

ypserv(1M)、 services(4)、 ypfiles(4)、 nsswitch.conf(4)、 thread_safety(5)。

符合的标准

getservent()：XPG4

名称

getspnam()、 getspnam_r()、 getspent()、 setspent()、 endspent()、 fgetspent() - 访问影子口令条目

概要

#include <shadow.h>

struct spwd *getspnam (const char *name);

struct spwd *getspnam_r (const char *name, struct spwd *result, char *buffer, size_t bufsiz);

struct spwd *getspent (void);

void setspent (void);

void endspent (void);

struct spwd *fgetspent (FILE *stream);

说明

例行程序 **getspnam()** 、 **getspnam_r()** 、 **getspent()** 和 **fgetspent()** 返回指向影子口令条目的指针。每个影子口令条目都是 **<shadow.h>** 头文件中声明的 **spwd** 结构，该结构包括以下成员：

```
char  *sp_namp;     /* the user's login name */
char  *sp_pwdp;     /* the encrypted password for the user */
long  sp_lstchg;    /* # of days from 1/1/70 when passwd was last modified */
long  sp_min;       /* min # of days allowed between password changes */
long  sp_max;       /* max # of days allowed between password changes */
long  sp_warn;      /* # of days before password expires and warning issued*/
long  sp_inact;     /* # of days between account inactive and disabled */
long  sp_expire;    /* # of days from 1/1/70 when account is locked */
unsigned long  sp_flag;  /* currently unused */
```

getspnam() 例行程序返回指向包含来自影子口令数据库（带有匹配的 **name** ）条目的结构的指针。

getspnam_r() 例行程序与 **getspnam()** 相似，不同之处在于，它不能在已转换为受信任模式的系统上工作，并且它包含三个附加参数。 **getspnam_r()** 可更新 **result** 指向的 **spwd** 结构，并返回指向该结构的指针。在 **buffer** 参数提供的内存（大小为 **buflen** ）中分配 **result** 指向的 **spwd** 结构所引用的存储。建议的缓冲区长度为 2048。

初次调用 **getspent()** 将返回指向第一个 **spwd** 结构的指针。后续调用将返回指向后续的 **spwd** 结构的指针。可以通过反复调用 **getspent()** 来搜索口令数据库中的所有条目。 **getspnam()** 例行程序将从头到尾地搜索口令条目，直到找到与 *name* 匹配的登录名，然后返回指向该条目的指针。

可以通过 **setspent()** 例行程序重置对影子口令条目的访问权限。调用 **setspent()** 之后，后续调用 **getspent()** 将返回第一个影子口令条目。使用此机制便能够反复搜索影子口令条目。 **endspent()** 例行程序用于指示已完成口令条目的处理。

fgetspent() 与上述其他函数不同，它既不使用 **/etc/nsswitch.conf** ，也不访问 NIS。它返回一个指向标准 I/O 流中

的下一个 **spwd** 结构的指针。I/O 流应处于打开状态，以便进行读取，并且其内容应与 **/etc/shadow** 的格式匹配。

注释

影子口令条目通常驻留在 **/etc/shadow** 中。但是，有两种例外情况。在不包含 **/etc/shadow** 文件的标准系统上，口令和时限信息将从 **/etc/passwd** 获取，并将转换为 **spwd** 结构。如果系统已转换为受信任的系统，那么将从受保护口令数据库 (**/tcb/files/auth/*/***) 获取口令和时限信息，并且这些信息将转换为 **spwd** 结构。

如果条目中未指定与 **sp_min** 、 **sp_max** 、 **sp_lstchg** 、 **sp_warn** 、 **sp_inact** 、 **sp_expire** 或 **sp_flag** 对应的字段，那么这些字段将缺省为 **-1** 。如果 **sp_min** 、 **sp_max** 或 **sp_warn** 的返回值为 **-1** ，那么与该字段关联的功能被视为禁用。

例行程序 **getspent()** 、 **getspnam()** 和 **getspnam_r()** 取决于 **/etc/nsswitch.conf** 文件的配置。请参阅 *nsswitch.conf*(4) 。条目可以驻留在 **/etc/nsswitch.conf** 中指定的任何储备库中。这些例行程序使用 **passwd** 数据库的交换；例如， **/etc/nsswitch.conf** 中的条目将包含 **"passwd: nis files"** 。

必须使用 **-lsec** 编译使用这些例行程序的程序。

实际应用信息

在标准系统上的多线程应用程序中， **getspnam** 、 **getspnam_r()** 、 **getspent()** 、 **setspent()** 、 **endspent()** 和 **fgetspent()** 是线程安全的，但不是异步取消安全的。当某个线程执行其中的任何接口时，将出现一个取消点。对于已转换为受信任模式的系统，只有 **fgetspent()** 是线程安全的。

返回值

如果在读取时遇到 EOF 或错误， **getspnam()** 、 **getspnam_r()** 、 **getspent()** 和 **fgetspent()** 将返回 **NULL** 指针。否则，返回值将指向有效的 **spwd** 结构。如果是操作 **getspnam()** 、 **getspent()** 和 **fgetspent()** ，那么 **spwd** 结构将驻留在内部区。如果是操作 **getspnam_r()** ，那么 **spwd** 结构将驻留在 **result** 参数指向的 **spwd** 结构内。

警告

HP-UX 11i v3 是支持受信任系统功能的最后一个版本。

文件

/etc/passwd	系统口令文件。
/etc/shadow	影子口令文件。
/tcb/files/auth/*/*	受信任系统的受保护口令数据库。

另请参阅

getpwent(3C)、 getprpwent(3)、 nsswitch.conf(4)、 passwd(4)、 shadow(4)。

符合的标准

getspent ： SVID3

名称

getspwent() 、 getspwuid() 、 getspwaid() 、 getspwnam() 、 setspwent() 、 endspwent() 、 fgetspwent() 、 getspwent_r()、 getspwuid_r()、 getspwaid_r()、 getspwnam_r()、 setspwent_r()、 endspwent_r()、 fgetspwent_r() - 获取受信任系统上的安全口令文件条目

概要

```
#include <pwd.h>

struct s_passwd *getspwent(void);

struct s_passwd *getspwuid(uid_t uid);

struct s_passwd *getspwaid(aid_t aid);

struct s_passwd *getspwnam(const char *name);

void setspwent(void);

void endspwent(void);

struct s_passwd *fgetspwent(FILE *stream);
```

过时的接口

下列重入接口将要过时:

getspwent_r() 、 getspwuid_r() 、 getspwaid_r() 、 getspwnam_r() 、 setspwent_r() 、 endspwent_r() 、 fgetspwent_r() 。

```
#include <pwd.h>

int getspwent_r(struct s_passwd *result, char *buffer, int buflen,
    FILE **pwfp);

int getspwuid_r(uid_t uid, struct s_passwd *result,
    char *buffer, int buflen);

int getspwaid_r(aid_t aid, struct s_passwd *result,
    char *buffer, int buflen);

int getspwnam_r(char *name, struct s_passwd *result,
    char *buffer, int buflen);

void setspwent_r(FILE **pwfp);

void endspwent_r(FILE **pwfp);

int fgetspwent_r(FILE *f, struct s_passwd *result,
    char *buffer, int buflen);
```

说明

这些有权限例行程序可提供对受保护口令数据库的访问权限，其工作方式类似于 *getpwent*(3C) 例行程序处理常规口令文件 **/etc/passwd** 。

如果没有必要获取常规口令文件中的信息，则使用这些例行程序是特别有用的。在受信任系统上，可以使用 *getspwent*(3X) 返回口令、审核 ID 和审核标记信息。使用这些例行程序的程序必须与安全库 **libsec** 链接。

请注意，不再支持 **getspwent()** 例行程序。但为了实现向后兼容，这些例行程序暂时可用。访问受信任系统上的受保护口令数据库的新应用程序应该使用 **getprpwent()** 例行程序。请参阅 *getprpwent*(3) 。

getspwent() 、 **getspwuid()** 、 **getspwaid()** 和 **getspwnam()** 中的每一个都可返回指向 *s_passwd* 结构的某个对象的指针。 *s_passwd* 结构是为了与现有软件兼容而保留的，它包括下列五个字段：

```
struct s_passwd {
   char  *pw_name;          /* login name */
   char  *pw_passwd;        /* encrypted password */
   char  *pw_age;           /* password age */
   int   pw_audid;          /* audit ID   */
   int   pw_audflg;         /* audit flag 1=on, 0=off */
};
```

由于 **<pwd.h>** 头文件中已声明了 *s_passwd* 结构，没有必要再次声明。

要访问受保护的口令数据库中未加入到 *s_passwd* 结构的其他字段，请使用 **getprpwent()** 。有关详细信息，请参阅 *getprpwent*(3) 。

getspwent() 首次调用 **getspwent()** 后，它将返回指向依次从每个用户的受保护口令数据库中获取的每个 *s_passwd* 结构的指针。可通过后续调用搜索整个数据库。

getspwuid() 搜索与指定的 *uid* 匹配的条目。然后返回指向在其中找到了 *uid* 的特定结构的指针。

getspwaid() 同样，搜索与 *aid* 匹配的数字审核 ID，并返回指向在其中找到了 *aid* 的特定结构的指针（有关此字段的详细信息，请参阅 *passwd*(4) ）。

getspwnam() 搜索与指定的 *name* 匹配的条目。并返回指向在其中找到了 *name* 的特定结构的指针。

setspwent() 将受保护的口令数据库指针重置到文件的起始位置，以便执行反复搜索。

endspwent() 处理完成后，应调用此例行程序来关闭受保护的口令数据库文件。

fgetspwent() 不再受支持。它是为不使用 **/.secure/etc/passwd** 的那些应用程序提供的。

重入接口

需要为 **getspwuid_r()** 、 **getspwaid_r()** 、 **getspwnam_r()** 和 **fgetspwent_r()** 传递三个额外的参数：

1. 将在其中存储结果的 *s_passwd* 结构的地址；
2. 一个缓冲区，用于存储 *s_passwd* 结构中的字段所指向的字符型字符串（例如口令）；

3. 用户提供的缓冲区的长度。建议的缓冲区长度为 1024。

除上述三个参数外， **getspwent_r()** 还需要一个指向 (**FILE ***) 变量的指针。 **setspwent_r()** 和 **endspwent_r()** 只能与 **getspwent_r()** 一同使用，并且将指向 (**FILE ***) 变量的同一指针用作参数。可以使用 **setspwent_r()** 来回卷或打开受保护的口令数据库。完成操作后，应调用 **endspwent_r()** 来关闭文件。

请注意，在第一次将 (**FILE ***) 变量传递给 **getspwent_r()** 或 **setspwent_r()** 之前，必须将其初始化为 NULL。并且以后不能以任何方式对其进行修改。

fgetspwent_r() 和 **setspwent_r()** 今后将会过时。

实际应用信息

在多线程的应用程序中，仅从一个专用线程调用这些例行程序是安全的。这些例行程序不是 POSIX.1c 异步取消安全的，也不是异步信号安全的。

返回值

在搜索过程中，如果 **getspwent()** 的任何例行程序遇到文件结束标记或错误，或者调用进程的有效用户 ID 不是零，那么将返回 NULL 指针。

如果 **getspwent_r()** 的任何例行程序遇到文件结束标记或错误，或者提供的缓冲区长度不足，那么将返回 −1。如果操作成功，将返回 0。

警告

以上例行程序需要使用 **<stdio.h>** ，因此它们可能会以超出意外的幅度增加程序的大小。

由于 **getspwent()** 、 **getspwuid()** 、 **getspwaid()** 、 **getspwnam()** 、 **setspwent()** 、 **endspwent()** 和 **fgetspwent()** 的所有信息都保留在静态区，因此通过复制对其进行保存。

受信任系统不支持网络信息服务。

不再支持本联机帮助页所述的例行程序。为了实现向后兼容，它们暂时可用，但是将会过时。

过时的接口

下列接口将要过时： **getspwent_r()** 、 **getspwuid_r()** 、 **getspwaid_r()** 、 **getspwnam_r()** 、 **setspwent_r()** 、 **endspwent_r()** 和 **fgetspwent_r()** 。

HP-UX 11i v3 是支持受信任系统功能的最后一个版本。

举例

以下代码片断将统计受保护的口令数据库中的条目数：

```
int count = 0;
struct s_passwd *pwbuf;

setspwent();
while (pwbuf=getspwent())
   count++;
endspwent();
```

作者

getspwent() 由 HP 开发。

文件

/tcb/files/auth/*/* 受保护口令数据库

另请参阅

ypcat(1)、 getgrent(3C)、 getlogin(3C)、 getpwent(3C)、 getprpwent(3)、 passwd(4)。

名称

getstr、mvgetstr、mvwgetstr、wgetstr — 从终端获取多字节字符串

概要

#include <curses.h>

int getstr(char *_str_);

int mvgetstr(int _y_, int _x_, char *_str_);

int mvwgetstr(WINDOW *_win_, int _y_, int _x_, char *_str_);

int wgetstr(WINDOW *_win_, char *_str_);

说明

getstr() 的作用类似于对 **getch()** 进行一系列调用，直到收到换行符或回车符这样的返回值。最终生成的值将被放入 *str* 所指向的区域。系统将对用户的清除字符和抹行字符以及特殊键（如功能键、home 键、clear 键等）进行解释。

mvgetstr() 函数与 **getstr()** 基本相同，不同之处在于其作用类似于先对 **move()** 进行调用，然后对 **getch()** 进行一系列调用。 **mvwgetstr()** 函数与 **getstr()** 基本相同，不同之处在于其作用类似于先对 **wmove()** 进行调用，然后对 **wgetch()** 进行一系列调用。

返回值

成功完成后，这些函数将返回 OK。否则，它们将返回 ERR。

错误

未定义错误。

实际应用信息

对于 *str* 所指向的数组而言，当用 **getstr()** 、 **mvgetstr()** 、 **mvwgetstr()** 或 **wgetstr()** 读取导致其溢出的行时，可产生不可预测的结果。

传统的实现通常会将返回的字节数限制为 256。

另请参阅

beep(3X)、 getch(3X)、 getnstr(3X)、 curses_intro(3X)，请参阅输入处理、<curses.h>。

历史变更记录

在 X/Open Curses 第 2 期中首次发布。

第 **3** 期

在 X/Open Curses 第 3 期中，有关 **getstr()** 、 **mvgetstr()** 、 **mvwgetstr()** 和 **wgetstr()** 函数的说明，请见 **addstr()** 条目。在 X/Open Curses 第 4 期中，这些函数的说明已经过重新编写（为了清楚说明）和更新（用于指示它们可正确处理多字节序列）。

名称

getsubopt( ) - 分析字符串中的子选项。

概要

#include <stdlib.h>

int getsubopt(char **optionp, char * const *tokens, char **valuep);

说明

getsubopt() 可分析某个标志参数中最初由 **getopt()** （请参阅 *getopt*(3C) ）分析的子选项。这些子选项由逗号分隔，并且可以包含单个标记，或由等号分隔的标记-值对。因为逗号可分隔选项字符串中的子选项，所以不允许将其用作该子选项的一部分，或者某个子选项的值。同样，由于等号是用来分隔某个标记及其值的，因此，标记中不得包含等号。使用此语法的一个示例命令为 **mount** 。通过 **mount** ，可以使用 **-** 交换来指定参数，如下所示：

mount -o rw,hard,bg,wsize=1024 speed:/usr /usr

此示例存在四个子选项： **rw** 、 **hard** 、 **bg** 和 **wsize** ，其中最后一个子选项的关联值为 1024。

getsubopt() 使用指向选项字符串的指针的地址、可能的标记的向量，以及某个值字符串指针的地址。如果没有匹配项，则它将返回与输入字符串中的子选项匹配的标记的索引，或者返回 –1。如果 ***optionp* 位置上的选项字符串只包含一个子选项，则 **getsubopt()** 将更新 ***optionp* ，使之指向字符串结尾的 NULL；否则，它将通过使用 NULL 替换逗号分隔符来隔离子选项，同时更新 ***optionp* ，使之指向下一个子选项的开头。如果子选项具有一个关联值，则 **getsubopt()** 将更新 ***valuep* ，使之指向第一个字符的值。否则，它将 ***valuep* 设置为 NULL。

标记向量组成为一系列指向以 NULL 结尾的字符串的指针。标记向量的结尾以 NULL 标识。

当 **getsubopt()** 返回时，如果 ***valuep* 不为空，则已处理的子选项将包括一个值。调用程序可使用此信息来确定，存在或缺少该子选项的值是否为一个错误。

此外，如果 **getsubopt()** 无法将子选项与 *tokens* 数组中的标记相匹配，则调用程序应确定这是否为一个错误，或者是否应将无法识别的选项传递给另一个程序。

外部语言环境影响

语言环境

LC_CTYPE 类别可确定将选项字母解释为单字节字符和（或）多字节字符。

国际代码集支持

支持单字节字符代码集和多字节字符代码集，但多字节字符文件名除外。

举例

以下代码段显示如何通过 **getsubopt()** ，并根据 **mount** 命令处理选项。

```
char *myopts[] = {
#define READONLY        0
                "ro",
#define READWRITE       1
                "rw",
```

```
#define WRITESIZE       2
                    "wsize",
#define READSIZE        3
                    "rsize",
                    NULL};

main (int argc, char **argv)
{
   int sc, c, errflag;
   char *options, *value;
   extern char *optarg;
   extern int optind;
   .
   .
   .
   while ((c = getopt(argc, argv, "abf:o:"))!= EOF)
      switch (c) {
      case 'a':/* process 'a' option */
         break;
      case 'b':/* process 'b' option */
         break;
      case 'f':
         ofile = optarg;
         break;
      case '?':
         errflag++;
         break;
      case 'o':
         options = optarg;
         while (*options != '\0') {
            switch(getsubopt(&options, myopts, &value)) {
            case READONLY:/* process ro option */
               break;
            case READWRITE:/* process rw option */
               break;
            case WRITESIZE:/* process wsize option */
               if (value == NULL) {
                  error_no_arg();
                  errflag++;
               }
```

```
                    else
                        write_size = atoi(value);
                    break;
                case READSIZE:/* process rsize option */
                    if (value == NULL) {
                        error_no_arg();
                        errflag++;
                    }
                    else
                        read_size = atoi(value);
                    break;
                default:
                    /* process unknown token */
                    error_bad_token(value);
                    errflag++;
                    break;
                }
            }
            break;
        }
    }
    if (errflg) {
        fprintf(stderr, "usage:. . . ");
        exit (2);
    }
    for ( ; optind < argc; optind++) {
        /* process remaining arguments */
        .
        .
        .
}
```

另请参阅

getopt(3C)、 thread_safety(5)。

符合的标准

getsubopt()： SVID3

名称

gettimer - 获取每进程计时器的值

概要

#include <sys/timers.h>

int gettimer(timer_t timerid, struct itimerspec *value);

说明

gettimer() 函数向 **value** 参数返回 **itimerspec** 结构值。对于在 **timerid** 中指定的计时器来说，此结构的 **it_value** 成员表示计时器到期之前的当前间隔中的时间，或者，如果禁用计时器，则表示零。 **it_interval** 成员具有 **reltimer()** 上一次设置的值（请参阅 *reltimer*(3C) ）。 **value** 的成员易受计时器精度的影响（请参阅 *mktimer*(3C) ）。

如果 **value** 为 NULL，则此函数的行为是不确定的。

返回值

成功完成后， **gettimer()** 返回零；否则，它返回 −1，并将 **errno** 设置为指示出错。

错误

如果遇到下列情况之一， **gettimer()** 将失败：

[EINVAL] **timerid** 不对应于 **mktimer()** 返回的 ID。

[EIO] 访问时钟设备时所出现的错误。

文件

/usr/include/sys/timers.h

另请参阅

timers(2)、 timer_gettime(2)、 mktimer(3C)、 reltimer(3C)、 thread_safety(5)。

符合的标准

gettimer()： AES

名称

gettxt() - 从消息文件中读取文本字符串

概要

#include <unistd.h>

char *gettxt(char *msg_id, char *def_str);

说明

gettxt() 例行程序从当前语言环境的消息文件中检索文本字符串。

msg_id 使用的语法如下：

*msgfilename***:***msgnumber*

其中， *msgfilename* 是由 *mkmsgs*(1) 生成的消息文件的名称。如果 *msgfilename* 为 NULL，则 **gettxt()** 使用在最后一次调用 *setcat*(3C) 中所指定的消息文件。 *msgnumber* 是此消息文件中的文本字符串的序列号（序列从 1 开始）。

gettxt() 返回消息

Message not found!!

在下列任何情况下：

- *msgfilename* 是无效的消息清单名。
- 没有在 *msg_id* 中或通过 *setcat*(3C) 指定清单。
- *msgnumber* 为非正数。
- 没有检索任何消息，并且 *def_str* 为 NULL。

外部语言环境影响

环境变量

gettxt() 使用环境变量 **LC_MESSAGES** 确定语言环境，以搜索 *msgfilename* 消息文件。如果未设置 **LC_MESSAGES** ，则使用环境变量 **LANG** 。如果未设置 **LANG** ，则使用“C”语言环境。用户还可以通过 *setlocale*(3C) 例行程序更改语言环境。

如果在指定的语言环境中没有找到 *msgfilename* 消息文件，或者如果 *msgnumber* 是越限的，则 **gettxt()** 试图从“C”语言环境中检索文本字符串。如果即使从“C”语言环境中不能检索到文本字符串， *def_str* 为返回的字符串。

举例

以下代码段是等效的：

```
gettxt("mytest:1", "my default message");

setcat("mytest");
gettxt(":1", "my default message");
```

警告

过时的接口

gettxt() 在将来某一天会过时。

另请参阅

mkmsgs(1)、setcat(3C)、setlocale(3C)、environ(5)、 thread_safety(5)。

符合的标准

gettxt()：SVID3

名称

getusershell( )、setusershell( )、endusershell( ) - 获取合法用户 Shell

概要

#include <unistd.h>

char *getusershell(void);

void setusershell(void);

void endusershell(void);

过时的接口

char *getusershell_r(char **shell_datap);

void setusershell_r(char **shell_datap);

void endusershell_r(char **shell_datap);

说明

getusershell() 返回指向在文件 **/etc/shells** 中所定义的（请参阅 *shells*(4) ）第一个合法用户 Shell 的指针。如果 **/etc/shells** 不存在或不可读，则 **getusershell()** 返回下列标准系统 Shell：

/sbin/sh
/usr/bin/sh
/usr/bin/rsh
/usr/bin/ksh
/usr/bin/rksh
/usr/bin/csh
/usr/bin/keysh

就像它们包含在 **/etc/shells** 中一样。文件保持打开，以便下一个调用返回下一个 Shell。当出现 EOF 或错误时返回空指针 (0)。

setusershell() 回卷文件。

endusershell() 关闭文件。

过时的接口

getusershell_r() 、 **setusershell_r()** 和 **endusershell_r()** 获取合法用户 Shell。

警告

getusershell_r() 、 **setusershell_r()** 和 **endusershell_r()** 都是过时的接口，只是为了与现有 DCE 应用程序兼容而受支持。新的多线程应用程序应该使用 **getusershell()** 、 **setusershell()** 和 **endusershell()** 。

作者

getusershell() 由 HP 和加州大学伯克利分校联合开发。

文件

/etc/shells

另请参阅

shells(4)、 thread_safety(5)。

名称

getut : getutent()、 getutid()、 getutline()、 pututline()、 _pututline()、 setutent()、 endutent()、 utmpname() - 访问 **utmp** 文件条目

概要

```
#include <utmp.h>

struct utmp *getutent(void);

struct utmp *getutid(const struct utmp *id);

struct utmp *getutline(const struct utmp *line);

struct utmp *_pututline(const struct utmp *utmp);

void pututline(const struct utmp *utmp);

void setutent(void);

void endutent(void);

int utmpname(const char *file);
```

过时的接口

```
int getutent_r(struct utmp **utmp, struct utmp_data *ud);

int getutid_r(
    struct utmp *id,
    struct utmp **utmp,
    struct utmp_data *ud);

int getutline_r(
    struct utmp *line,
    struct utmp **utmp,
    struct utmp_data *ud);

int pututline_r(const struct utmp *utmp, struct utmp_data *ud);

void setutent_r(struct utmp_data *ud);

void endutent_r(struct utmp_data *ud);

int utmpname_r(const char *file);
```

说明

getutent() 、 **getutid()** 和 **getutline()** 都可返回指向以下类型的结构的指针 :

```
struct utmp {
    char ut_user[8];            /* User login name */
    char ut_id[4];              /* /etc/inittab id (usually line #) */
```

```
    char ut_line[12];          /* device name (console, lnxx) */
    pid_t ut_pid;              /* process id */
    short ut_type;             /* type of entry */
    struct exit_status {
      short e_termination;     /* Process termination status */
      short e_exit;            /* Process exit status */
      } ut_exit;               /* The exit status of a process marked as DEAD_PROCESS.*/
    unsigned short ut_reserved1; /* Reserved for future use */
    time_t ut_time;            /* time entry was made */
    char ut_host[16];          /* host name, if remote;NOTSUPPORTED*/
    unsigned long ut_addr;     /* Internet addr of host, if remote */
};
```

getutent() 从类似于 **utmp** 的文件读入下一个条目。如果该文件尚未打开， **getutent()** 可将其打开。如果到达文件的结尾， **getutent()** 将会失败。

getutid() 如果指定的类型为 **RUN_LVL** 、 **BOOT_TIME** 、 **OLD_TIME** 或 **NEW_TIME** ，将从 **utmp** 文件中的当前点开始向前搜索，直到找到与 *id*-*ut_type* 匹配的带有 **ut_type** 的条目。如果 *id* 中指定的类型为 **INIT_PROCESS** 、 **LOGIN_PROCESS** 、 **USER_PROCESS** 或 **DEAD_PROCESS** ， **getutid()** 将返回指向其类型为这四个类型之一，并且其 **ut_id** 字段与 *id*-*ut_id* 匹配的第一个条目的指针。如果到达文件的结尾时还是没有找到匹配项， **getutid()** 将会失败。

getutline() 从 **utmp** 文件中的当前点开始向前搜索，直到找到类型为 **LOGIN_PROCESS** 或 **USER_PROCESS** ，并且带有与 *line*-*ut_line* 字符串匹配的 **ut_line** 字符串的条目。如果到达文件的结尾时还是没有找到匹配项， **getutline()** 将会失败。

pututline() 将提供的 **utmp** 结构写入 **utmp** 文件，然后将提供的 **utmp** 结构转换为 **utmpx** 结构，再将其写入 **utmpx** 文件。如果 **pututline()** 不在合适的位置，它将使用 **getutid()** 来向前搜索合适的位置。通常，应用程序需要使用 **getut()** 例行程序之一搜索合适的条目后再调用 **pututline()** 。如果已经执行了搜索，则 **pututline()** 不再重复此操作。如果 **pututline()** 没有为新条目找到匹配的插槽，它将在文件的结尾添加一个新条目。

_pututline() 与 **pututline()** 执行的操作相同，不同之处在于，它可以返回对错误检查有用的值。

setutent() 将输入流重置到文件的开头。如果希望检查整个文件，则应该在每次执行新条目搜索之前完成此操作。

endutent() 关闭当前打开的文件。

utmpname() 允许用户将检查的文件的名称由 **/etc/utmp** 和 **/etc/utmpx** 更改为任何其他文件。在这种情况下，为 **utmpname** 提供的名称将用于 **utmp** 函数。将在该名称后面追加一个 **x** ， *getutx*(3C) 函数将使用这个标记。 **pututl()** 和 **pututx()** 函数是这种情况的例外，这些函数可访问这两种文件，以尝试使 **utmp** 和 **utmpx** 文件同步。其他文件通常是指 **/var/adm/wtmp**

和 **/var/adm/wtmpx** 。如果文件不存在，那么在第一次后续尝试引用该文件之前，将无法发现该文件的缺失。 **utmpname()** 不会打开文件 — 如果当前打开了旧文件，它只是将其关闭，然后保存新文件名。

最新的条目保存在静态结构中。如果要进行多次访问，那么只有在复制了该结构后才能执行下一次访问。每次调用 **getutid()** 或 **getutline()** 时，将在检查静态结构后执行其他 I/O 操作。如果静态结构的内容与例行程序所搜索的内容匹配，则不会执行其他搜索。因此，如果正在使用 **getutline()** 来搜索多个出现的项目，那么在每次成功后，需要将静态结构清零；否则， **getutline()** 只会一次又一次地返回相同的指针。执行新的读取操作之前，在与删除结构有关的规则方面存在一种例外：如果用户刚刚修改了 **getutent()** 、 **getutid()** 或 **getutline()** 返回的静态结构的内容，并且将指针传回给了 **pututline()** ，那么 **pututline()** 执行的隐式读取（如果它发现自己不在文件中的适当位置）不会更改这些内容。

过时的接口

getutent_r() 、 **getutid_r()** 、 **getutline_r()** 、 **pututline_r()** 、 **setutent_r()** 、 **endutent_r()** 、 **utmpname_r()** 可访问 utmp 文件条目。

返回值

如果出现读取失败（无论是出于权限的原因还是到达了文件的结尾），或者出现写入失败，这些函数将返回 NULL 指针。如果文件的大小不是 **sizeof(struct utmp)** 的整数倍数，它们也会返回 NULL 指针。

_pututline() 的行为与 **pututline()** 相似，不同之处在于，如果 **_pututline()** 调用成功，它将返回指向包含了最新的 utmp 条目的静态位置的指针。如果成功，该结构的内容与提供的 **utmp** 结构的内容相同。如果 **_pututline()** 在写入 **utmp** 时失败，将返回 NULL 指针。如果 **_putuline()** 在写入 **utmp** 文件时成功，但在写入 **utmpx** 文件时失败，那么 **_pututline()** 的行为就如同它成功执行了这些操作。请注意，由于上述行为， **utmp** 文件和 **utmpx** 文件可能不同步。如果成功完成操作， **pututline()** 和 **_pututline()** 只能保证写入了 **utmp** 文件。

重入接口

如果成功完成操作， **getutent_r()** 、 **getutid_r()** 、 **getutline_r()** 和 **pututline_r()** 将返回 **0** 。否则，全部返回 **-1** ，并设置 **errno** 。

错误

重入接口

[EINVAL] *utmp* 或 *ud* 参数等于 NULL。

警告

getutent_r() 、 **getutid_r()** 、 **getutline_r()** 、 **pututline_r()** 、 **setutent_r()** 、 **endutent_r()** 和 **utmpname_r()** 是过时接口，之所以还提供对该接口的支持，只是为了实现与现有 DCE 应用程序兼容。新的多线程应用程序应使用具有等同功能的 *getutx*(3C) 函数。

如果存在 **utmp** 文件，但它不是 **sizeof(struct utmp)** 的整数倍数，则某些供应商的 **getutent()** 版本会清除该文件。假如用户为 *utmpname* 提供名称时出现错误（例如为 *who*(1) 指定了不正确的参数），HP-UX 不是执行清除操作，而是返回一个错误指示。

为实现可移植性， *getutx*(3C) 是所有这些函数中的首选项。

文件

/etc/utmp
/etc/utmpx
/var/adm/wtmp

另请参阅

utmpd(1M)、 getuts(3C)、 getutx(3C)、 pututxline(3C)、 ttyslot(3C)、 utmp(4)、 thread_safety(5)。

符合的标准

endutent()：SVID2、SVID3、XPG2

getutent()：SVID2、SVID3、XPG2

getutid()：SVID2、SVID3、XPG2

getutline()：SVID2、SVID3、XPG2

pututline()：SVID2、SVID3、XPG2

setutent()：SVID2、SVID3、XPG2

utmpname()：SVID2、SVID3、XPG2

名称

getuts： endutsent()、 getutsent()、 getutsid()、 getutsline()、 pututsline()、 setutsent()、 ENDUTSENT()、 GETUTSENT()、 GETUTSID()、 GETUTSLINE()、 PUTUTSLINE()、 SETUTSENT()、 - **utmpd** 维护的用户帐户数据库的访问和更新例行程序

概要

#include <utmps.h>

struct utmps * getutsent(size_t utmps_size):

struct utmps * getutsid(const struct utmps *id, size_t utmps_size):

struct utmps * getutsline(const struct utmps *line, size_t utmps_size):

struct utmps * pututsline(const struct utmps *utmps, size_t utmps_size):

struct utmps * getutspid(pid_t pid, size_t utmps_size):

void setutsent(void):

void endutsent(void):

备注

需要将 **utmps** 结构的大小（例如， **sizeof(struct utmps)** ）作为 *utmps_size* 参数传入上述调用。

上述每个函数调用都存在等效的宏。这些宏可通过将 *utmps_size* 参数隐式传递给对应的 *getuts*(3C) 函数，为调用 *getuts*(3C) 函数提供捷径。

struct utmps * GETUTSENT():

struct utmps * GETUTSID(const struct utmps *id):

struct utmps * GETUTSLINE(const struct utmps *line):

struct utmps * PUTUTSLINE(const struct utmps * utmps):

struct utmps * GETUTSPID(pid_t pid):

void SETUTSENT(void):

void ENDUTSENT(void):

说明

getutsent() 、 **getutsid()** 和 **getutsline()** 中的每一个都可以返回指向 **utmps** 结构的指针， **utmps** 结构的关键元素包括

char ut_user[]	用户登录名
char ut_id[]	用于区分某个条目的唯一 ID
char ut_line[]	设备名称
pid_t ut_pid	进程 ID

short ut_type	条目类型
struct ut_exit	进程的退出状态
struct timeval ut_tv	已设置时间条目
char ut_host[]	如果使用的是远程主机，则为该主机名
uint8_t ut_addr[]	如果使用的是远程主机£则为该主机的 Internet 地址
short ut_addr_type	用于标识in ut_addr 中的地址类型的标记

例行程序

getutsent() 该调用将返回 *utmpd*(1M) 维护的内存中用户帐户数据库中的下一个记录。如果到达 **utmpd** 的数据库的结尾， **getutsent()** 将会失败，并返回 NULL。宏 **GETUTSENT()** 是围绕 **getutsent()** 的包装，它可以隐式传递 *utmps_size* 参数。

getutsid() 对于与 **INIT_PROCESS** 、 **LOGIN_PROCESS** 、 **USER_PROCESS** 和 **DEAD_PROCESS** 匹配的 *ut_type* 的条目， **getutsid()** 将搜索 **utmpd** 的数据库，以找到与 *id->ut_id* 字段匹配的条目。提取记录时将忽略内部排序中的当前位置，并且记录不会更改 **getutsent()** API 的内部排序中的当前位置。

如果指定的 *ut_type* 为 **RUN_LVL** 、 **BOOT_TIME** 、 **OLD_TIME** 或 **NEW_TIME** ，那么 **getutsid()** 将提取其 *ut_type* 与 *id->ut_type* 匹配的条目。如果没有找到匹配的条目， **getutsid()** 将会失败，并返回 NULL。宏 **GETUTSID()** 是围绕 **getutsid()** 的包装，它可以将 *utmps_size* 参数隐式传递给 **getutsid()** 。

getutsline() 与 **getutsid()** 类似。它可以搜索 *utmpd*(1M) 数据库，以找到类型为 **LOGIN_PROCESS** 或 **USER_PROCESS** ，并且其 *ut_line* 字段与 *line->ut_line* 字符串匹配的条目。如果在 **utmpd()** 的数据库中未找到该条目， **getutsline()** 将失败，并返回 NULL。提取记录时将忽略内部排序中的当前位置，并且记录不会更改 **getutsent()** API 的内部排序。宏 **GETUTSLINE()** 是围绕 **getutsline()** 的包装，它可以将 *utmps_size* 参数隐式传递给 **getutsline()** 。

pututsline() 将提供的 **utmps** 结构写入 *utmpd*(1M) 维护的内存中用户帐户数据库。对于匹配 **INIT_PROCESS** 、 **LOGIN_PROCESS** 、 **USER_PROCESS** 和 **DEAD_PROCESS** 的 *ut_type matching* 条目， **pututsline()** 将搜索 *utmpd*(1M) 的数据库，以查找与 *utmps->ut_id* 字段匹配的条目。如果 *utmpd*(1M) 的数据库中存在该条目，该 API 将更新已经存在的条目。否则， **pututsline()** 将向 *utmpd*(1M) 的数据库添加一个新条目。

如果指定的 *ut_type* 为 **RUN_LVL** 、 **BOOT_TIME** 、 **OLD_TIME** 或 **NEW_TIME** ，那么 **pututsline()** 将更新其 *ut_type* 与 *utmps->ut_type* 匹配的条目。宏 **PUTUTSLINE()** 是围绕 **pututsline()** 的包装，它可以将 *utmps_size* 参数隐式传递给 **pututsline()** 。

getutspid() 该 API 可搜索 *utmpd*(1M) 的数据库，以找到其 *ut_pid* 与 *pid* 参数匹配的条目。由于 *pid* 只对当前活动的进程有意义，因此匹配只是针对类型为 **INIT_PROCESS** 、

LOGIN_PROCESS 或 **USER_PROCESS** 的条目执行的。如果找到匹配的条目，则 **getutspid()** 返回对应的条目；否则， **getutspid()** 将会失败，并返回 NULL。宏 **GETUTSPID()** 是围绕 **getutspid()** 的包装，它可以将 *utmps_size* 参数隐式传递给 **getutspid()** 。

setutsent() 将 **getutsent()** 请求重置为从 *utmpd*(1M) 的数据库的起始位置开始。它还可以清除静态结构。宏 **SETUTSENT()** 是围绕 **setutsent()** 的包装。

endutsent() 可关闭任何打开的文件描述符，及清除静态结构。宏 **ENDUTSENT()** 是围绕 **endutsent()** 的包装。

为了将来能够扩展接口和 **utmps** 结构，同时做到与使用 *getuts*(3C) 接口编写的程序的向后兼容，需要使用特定的调用约定来传递所需的数据结构大小。

这些接口将在 *utmps_size* 参数中编码 **utmps** 结构的版本信息。如果将来的发行版更改了 **utmps** 结构， *utmpd*(1M) 和 libc 接口将根据 *utmps_size* 参数来检测编译应用程序时使用的 **utmps** 结构的版本，同时返回相应的 **utmps** 结构。

仅当没有运行 *utmpd*(1M) 时， *getuts*(3C) 接口才使用 **/etc/utmps** 文件。将来的发行版可能会停用此功能。

utmps 结构中的 **ut_addr** 字段可保存 16 个字节的 IPv6 地址。如果应用程序需要将四个字节的 IPv4 地址写入该字段，则必须将 **ut_addr_type** 字段初始化为 **IPV4_ADDRESS** ，再使用 **ut_addr** 字段的最后四个字节。需要将 IPv6 地址写入 **ut_addr** 字段的应用程序必须将 **ut_addr_type** 初始化为 **IPV6_ADDRESS** 。 **IPV4_ADDRESS** 和 **IPV6_ADDRESS** 是在 **utmps.h** 头文件中定义的宏。

返回值

如果成功， **getutsent()** 、 **getutsid()** 、 **getutsline()** 、 **pututsline()** 和 **getutspid()** 将返回指向静态 **utmps** 结构的指针。如果失败，则返回 NULL。

错误

[EINVAL] 传递给 **getuts** 函数的大小参数不匹配服务器支持的任何结构大小。

警告

如果将来的发行版更改了 **utmps** 结构的某个成员的大小，那么对于使用 **utmps** 结构的早期版本编译的应用程序，将截断其扩展的结构字段的信息。要获取完整的信息，必须重新编译应用程序。

getutsent() 、 **getutsid()** 、 **getutsline()** 、 **pututsline()** 和 **getutspid()** 返回的值指向每次调用其中的任何函数都会被覆盖的单一静态区，因此必须通过复制来保存此值。

作者

getuts() 例行程序由 HP 开发。

文件

/etc/utmps

另请参阅

utmpd(1M)、 bwtmps(3C)、 thread_safety(5)。 @@@**getuts(3C)**

维护的用户帐户数据库的访问和更新例行程序@@@**getuts(3C)**

名称

getutx： endutxent()、 getutxent()、 getutxid()、 getutxline()、 pututxline()、 setutxent() - 访问 utmpx 文件条目

概要

```
#include <utmpx.h>

struct utmpx *getutxent(void);

struct utmpx *getutxid(const struct utmpx *id);

struct utmpx *getutxline(const struct utmpx *line);

struct utmpx *pututxline(const struct utmpx *utmpx);

void setutxent(void);

void endutxent(void);
```

备注：

不建议使用 **utmpx** 结构中的 **ut_addr** 字段，这些字段在将来的版本中将过时且会被删除。如果需要此字段，请使用 **getuts()** 函数（请参阅 *getuts*(3C) ）。

说明

getutxent() 、 **getutxid()** 和 **getutxline()** 各自向以下类型结构返回一个指针：

```
struct utmpx {
   char ut_user[24];            /* User login name */
   char ut_id[4];               /* /etc/inittab id (usually line #) */
   char ut_line[12];            /* device name (console, lnxx) */
   pid_t ut_pid;                /* process id */
   short ut_type;               /* type of entry */
   struct __exit_status {
      short __e_termination;    /* Process termination status */
      short __e_exit;           /* Process exit status */
      } ut_exit;                /* The exit status of a process marked as DEAD_PROCESS.*/
   unsigned short ut_reserved1; /* Reserved for future use */
   struct timeval {
      time_t  tv_sec;           /* seconds */
      long    tv_usec;          /* and microseconds */
      } ut_tv;                  /* time entry was made */
   char ut_host[64];            /* host name, if remote; NOT SUPPORTED */
   unsigned long ut_addr;       /* Internet addr of host, if remote; TO BE OBSOLETED */
   char ut_reserved2[12] ;      /* Reserved for future use */
};
```

getutxent() 从 **utmpx**-like 文件读入下一个条目。如果此文件尚未打开，则使用 **getutxent()** 打开它。如果已到达文件的最后， **getutxent()** 将失败。

getutxid() 从 *utmpx* 文件中的当前位置向前搜索，如果指定的类型是 **RUN_LVL** 、 **BOOT_TIME** 、 **OLD_TIME**、或 **NEW_TIME** ，则直到它找到一个与 *id->ut_type* 匹配的 **ut_type** 的条目为止。如果在 *id* 中指定的类型是 **INIT_PROCESS** 、 **LOGIN_PROCESS** 、 **USER_PROCESS**、或 **DEAD_PROCESS** ，则 **getutxid()** 向第一个条目返回一个指针，第一个条目的类型应该是这四种类型中的一个，并且其 **ut_id** 字段应该与 *id->ut_id* 相匹配。如果搜索到文件最后也没有找到匹配项，则 **getutxid()** 将失败。

getutxline() 从 *utmpx* 文件中的当前位置向前搜索，直到它找到一个特殊类型的条目为止。此条目的类型应该是 **LOGIN_PROCESS** 或 **USER_PROCESS** ，同时具有一个与 *line->ut_line* 字符串匹配的 **ut_line** 字符串。如果搜索到文件最后也没有找到匹配项，则 **getutxline()** 将失败。

pututxline() 将提供的 **utmpx** 结构写入 *utmpx* 文件，将提供的 **utmpx** 结构转换为 **utmp** 结构并将其写入 *utmp* 文件。如果此文件不在此处，则 **pututxline()** 使用 **getutxid()** 向前搜索适当的位置。一般情况下，在调用 **pututxline()**之前，应用程序已经通过使用 **getutx()** 某个例行程序搜索到了适当的条目。如果已经进行了搜索，则 **pututxline()** 不再重复执行。如果 **pututxline()** 没有为新条目找到匹配的插槽，则它将一个新条目添加到此文件的末端。

setutxent() 将输入流重置到文件的开头。如果需要检查整个文件，则在每次搜索新条目之前应当完成此操作。

endutxent() 关闭当前处于打开状态的文件。

最新条目保存在静态结构中。多次访问要求在执行进一步访问之前复制此结构。在每次调用 **getutxid()** 或 **getutxline()** 的过程中，在执行更多 I/O 检查之前先检查静态结构。如果静态结构的内容与例行程序搜索的内容匹配，则不执行额外的搜索。因此，如果使用 **getutxline()** 执行多次搜索，则必须清除每次成功搜索后的静态结构；否则，**getutxline()** 将多次返回同一指针。对于关于进行新的读取操作前删除结构的规则，有一个例外：如果用户已修改 **getutxent()** 、 **getutxid()** 或 **getutxline()** 返回的静态结构的内容，并将指针传回 **pututxline()**，则 **pututxline()** 所完成的隐式读取（如果它发现它已经不在文件中的正确位置）不会使这些内容发生改变。

返回值

当未能读取（不论是因为权限，还是因为已到达文件末尾）或未能写入时，这些函数返回 NULL 指针。如果文件大小不是 **sizeof(struct utmpx)** 的整数倍，则它们也将返回 NULL 指针。

如果 **pututxline()** 成功，则其将指针返回到包含最新 **utmpx** 条目的静态位置。如果成功，则此结构的内容与所提供的 **utmpx** 结构的内容是相同的。如果 **pututline()** 写入 **utmpx** 时失败，则其返回 NULL 指针。如果 **putuline()** 写入 **utmpx** 文件成功，但是写入 **utmp** 文件失败，那么 **pututline()** 将表现为成功。请注意， **utmpx** 文件和 **utmp** 文件可能因上述行为而不同步。 **pututxline()** 只保证成功完成对 **utmpx** 文件的写操作。

警告

不建议使用 **utmpx** 结构中的 **ut_addr** 字段，这些字段在将来的版本中将过时且会被删除。如果需要此字段，请使用 **getuts()** 函数（请参阅 *getuts*(3C) ）。

文件

/etc/utmp
/etc/utmpx
/var/adm/wtmp

另请参阅

getuts(3C)、 getutent(3C)、 getuts(3C)、 ttyslot(3C)、 utmp(4)、 thread_safety(5)。

符合的标准

endutxent()： XPG4.2

getutxent()： XPG4.2

getutxid()： XPG4.2

getutxline()： XPG4.2

pututxline()： XPG4.2

setutxent()： XPG4.2

名称

getwc()、 getwchar()、 fgetwc() - 从流文件获取宽字符

概要

#include <wchar.h>

wint_t getwc(FILE *stream);

wint_t getwchar(void);

wint_t fgetwc(FILE *stream);

过时的接口

wint_t getwc_unlocked(FILE *stream);

wint_t getwchar_unlocked(void);

wint_t fgetwc_unlocked(FILE *stream);

备注

这些函数遵循 XPG4 Worldwide Portability Interface 宽字符 I/O 函数。它们与定义在 *getc*(3S) 中的 8 位字符 I/O 函数相似。

说明

getwc() 返回指定输入 *stream* 中的下一个字符，将其转换为对应的宽字符，并在 *stream* 中将文件指针向前移动一个字符。 **getwchar()** 定义为 **getwc(stdin)** 。 **getwc()** 和 **getwchar()** 都定义为宏和函数。

fgetwc() 行为和 **getwc()** 相同，但是一个函数而不是一个宏。

这些函数的定义、类型 **wint_t** 、 **wchar_t** 和值 **WEOF** 都在头文件 **<wchar.h>** 中提供。

过时的接口

getwc_unlocked() 、 **getwchar_unlocked()** 、 **fgetwc_unlocked()** 从流文件中获取宽字符。

实际应用信息

将 **getwc()** 、 **getwchar()** 或 **fgetwc()** 应用于流后，此流就变成面向宽字符的（请参阅 *orientation*(5) ）。

返回值

成功完成后， **getwc()** 、 **getwc_unlocked()** 、 **getwchar()** 、 **getwchar_unlocked()** 、 **fgetwc()** 和 **fgetwc_unlocked()** 就返回从 *stream* （用于 getwchar() 的 **stdin** ）读取并转换为类型 **wint_t** 的下一个宽字符。如果此流位于文件末尾，则设置此流的文件末尾指示符并返回 **WEOF** 。

当对应于打开流的文件在到达文件结尾后被扩展时，任何对这些函数的后续调用都将成功，并且文件结束指示符将保持为已设置状态。但是，在 UNIX2003 标准环境中（请参阅 *standards*(5) ），这些函数将返回 **WEOF** 并且文件结束指示符仍将保持为已设置状态。

如果发生读取错误，将设置了流的错误指示符，并将设置 **errno** 以指明错误，同时返回 **WEOF** 。

可以使用 **ferror()** 和 **feof()** 来区分错误条件和文件结束条件。

错误

如果需要将数据读入 *stream* 的缓存中，则 **getwc()** 、 **getwc_unlocked()** 、 **getwchar()** 、 **getwchar_unlocked()** 、 **fgetwc()** 和 **fgetwc_unlocked()** 将失败，并且：

[EAGAIN] 为文件描述符的基础 *stream* 设置了 **O_NONBLOCK** 标记，并且将在读取操作中延迟进程。

[EBADF] 文件描述符的基础 *stream* 不是为读取而打开的有效文件描述符。

[EINTR] 由于接收到信号，或未传输数据，抑或实现未报告此文件的部分传输情况，因此终止了读取操作。

[EIO] 发生了物理 I/O 错误，或者该进程是后台进程的成员，并尝试从其控制终端读取数据，该进程忽略或阻塞 **SIGTTIN** 信号，或者进程的进程组被孤立。

[EILSEQ] 从输入流获取的数据没有构成有效的宽字符。

可以通过基础 **read()** 函数（请参阅 *read*(2) ）设置其他的 **errno** 值。

外部语言环境影响

环境变量

LC_CTYPE 类别决定如何转换宽字符。

国际代码集支持

支持单字节字符代码集和多字节字符代码集。

警告

如果由 **getwc()** 、 **getwchar()** 、 **fgetwc()** 或 **fgetwc_unlocked()** 返回的值存储在类型为 **wchar_t** 的变量中，然后与常量 **WEOF** 比较，则这种比较可能永远不会成功，因为将 **wchar_t** 扩展到 **wint_t** 与计算机有关。

getwc_unlocked() 、 **getwchar_unlocked()** 和 **fgetwc_unlocked()** 是过时的接口，只是为了兼容现有的 DCE 应用程序而进行支持。新的多线程应用程序应当使用 **getwc()** 、 **getwchar()** 和 **fgetwc()** 。

作者

getwc() 由 OSF 和 HP 联合开发。

另请参阅

fclose(3S) 、 ferror(3S) 、 flockfile(3S) 、 fopen(3S) 、 fread(3S) 、 fgetws(3C) 、 orientation(5) 、 putwc(3C) 、 read(2)、 scanf(3S)、 orientation(5)、 standards(5)、 thread_safety(5)。

符合的标准

getwc()：XPG4

fgetwc()：XPG4

getwchar()：XPG4

名称

get_wch、mvget_wch、mvwget_wch、wget_wch — 从终端获取宽字符

概要

#include <curses.h>

int get_wch(wint_t *_ch_);

int mvget_wch(int _y_, int _x_, wint_t *_ch_);

int mvwget_wch(WINDOW *_win_, int _y_, int _x_, wint_t *_ch_);

int wget_wch(WINDOW *_win_, wint_t *_ch_);

说明

这些函数从与当前或指定窗口相关的终端中读取一个字符。如果启用 **keypad()** ，则这些函数对按下功能键做出反应：将 *ch* 指向的对象设置为 **<curses.h>** 中所定义的 **KEY_** 值并返回 **KEY_CODE_YES** 。

终端输入的处理遵循在 *curses_intro*(3X) 中输入处理中所描述的一般规则。

如果启用回显，则此字符将被回显，如同其被作为输入参数提供给 **ins_wch()** ，以下字符除外：

<退格>、<左箭头> 及当前清除字符：
: 首先根据 **specialchars** 中的规定解释输入，然后删除最终光标位置处的字符，如同调用了 **delch()** ，唯一不同之处在于，如果光标原来位于此行第一列，则将向用户报警，如同调用了 **beep()** 。

功能键
: 对用户进行报警，如同调用了 **beep()** 。未将有关功能键的信息返回给调用程序。

如果当前窗口或指定窗口不是一块，且在上一次刷新之后经过移动或修改，则在读取其他字符之前将刷新此窗口。

返回值

当这些函数成功报告按下功能键时，它们返回 **KEY_CODE_YES** 。当它们成功报告一个宽字符时，它们返回 OK。否则，返回 ERR。

错误

没有定义任何错误。

实际应用信息

应用程序自身不应定义 escape 键作为单字符函数。

在使用这些函数时，不可同时使用 **nocbreak()** 模式和 **echo()** 模式。根据按下每个字符时终端的状态，应用程序可能会产生不符合需要的结果。

另请参阅

beep(3X)、 cbreak(3X)、 ins_wch(3X)、 keypad(3X)、 move(3X)、 curses_intro(3X)，请参阅输入处理、<curses.h>、 <wchar.h> （位于《X/Open System Interfaces and Headers, Issue 4, Version 2》规范中）。

历史变更记录

在 X/Open Curses 第 4 期中首次发布。

名称

getwd( ) - 获取当前工作目录的路径名

概要

#include <unistd.h>

char *getwd(char *buf);

说明

getwd() 将当前工作目录的绝对路径名放在 *buf* 指向的数组中，并返回 *buf* 。

如果当前工作目录的路径名的长度大于 **PATH_MAX+1** 字节，则 **getwd()** 失败并返回一个空指针。

返回值

成功完成后， **getwd()** 返回指向当前目录路径名的指针。否则，将返回设置了 **errno** 的 NULL。

错误

如果遇到下列情况，则 **getwd()** 将失败：

[ENAMETOOLONG] 指定路径名的长度超过 **PATH_MAX+1** 字节，或路径名某个部分的长度超过 **NAME_MAX** 字节，但 **_POSIX_NO_TRUNC** 仍然有效。

如果遇到下面任何情况之一，则 **getwd()** 可能失败：

[EACCES] 拒绝对路径名某个部分的读取或搜索权限。

[EFAULT] *buf* 指向进程的已分配地址空间之外的空间。 **getwd()** 可能始终无法检测这一错误。

举例

```
#include <stdio.h>
#include <unistd.h>
char *cwd;
char buf[PATH_MAX + 1 ];
                . . .
if ((cwd = getwd(buf)) == NULL) {
   perror("getwd");
   exit(1);
}
puts(cwd);
```

警告

出于可移植性考虑， **getcwd()** 的优先级高于此函数。

作者

getwd() 由 HP 和加州大学伯克利分校联合开发。

另请参阅

getcwd(3C)、 thread_safety(5)。

符合的标准

getwd()：XPG4.2

名称

getwin、 putwin - 将窗口转储至文件以及从文件中重新加载窗口

概要

#include <curses.h>

WINDOW *getwin(FILE *_filep_);

int putwin(WINDOW *_win_, FILE *_filep_);

说明

getwin() 函数将通过 **putwin()** 读取文件中存储的、与窗口有关的数据。随后，该函数将利用这些数据创建和启动新窗口。

putwin() 函数会利用不确定的格式将所有与 *win* 关联的数据写入 *filep* 所指向的 *stdio* 流。以后可使用 **getwin()** 检索此信息。

返回值

成功完成后， **getwin()** 将返回一个指向所创建的窗口的指针。否则，它将返回空指针。

成功完成后， **putwin()** 将返回 OK。否则，它将返回 ERR。

错误

未定义错误。

另请参阅

scr_dump(3X)、 <curses.h>。

历史变更记录

在 X/Open Curses 第 4 期中首次发布。

名称

getyx — 获取光标和窗口坐标

概要

#include <curses.h>

void getyx(WINDOW ****win*, int** *y***, int** *x***);**

说明

getyx() 宏将指定窗口的光标位置存储在 *y* 和 *x* 中。

返回值

未定义返回值。

错误

没有定义任何错误。

实际应用信息

这些接口为宏，并且在 *y* 和 *x* 参数之前不能使用“**&**”。

另请参阅

getbegyx(3X)、 <curses.h>。

历史变更记录

在 X/Open Curses 第 2 期中首次发布。

X/Open Curses 第 **4** 期

为清楚起见，重新编写了此条目。

名称

glob()、globfree() - 文件名生成函数

概要

```
#include <glob.h>

int glob(
    const char *__restrict pattern,
    int flags,
    int (*errfunc)(const char *, int),
    glob_t *__restrict pglob
);

void globfree(glob_t *pglob);
```

说明

glob() 是路径名生成器。 *pattern* 是指向要扩展的路径名模式的指针。如果 *pattern* 包含特殊字符 ***** 、 **?** 或 **[** 中的任何一项，那么将针对所有可访问的路径名匹配 *pattern* 。 **glob()** 要访问路径名，需要具有：

- 对除最后一个路径以外的其他路径的每个组件的搜索权限。
- 对 *pattern* （包含上述任何特殊字符）的任何文件名组成部分的每个目录的读取权限。

glob() 可存储 *pglob* –> *gl_pathc* 中匹配的路径名数目，并可存储指向 *pglob*–>*gl_pathv* 中已排序的路径名列表的指针。最后一个路径名之后的第一个指针为空指针。

调用方负责为 *pglob* 指向的结构分配空间。 **glob()** 根据需要分配其他空间，包括 *gl_pathv* 指向的内存。 **globfree()** 可通过事先调用 **glob()** ，释放与 *pglob* 关联的任何空间。

flags 参数用于控制 **glob()** 的行为。 *flags* 的值为 **<glob.h>** 中定义的以下常量的按位包含逻辑 OR：

GLOB_NOESCAPE 禁用反斜杠转义。

GLOB_ERR 使 **glob()** 在第一次遇到无法打开或读取的目录时便返回。正常情况下， **glob()** 将继续查找匹配项。

GLOB_MARK 在与 *pattern* 匹配的并且是目录的每个路径名后面追加 **/** 。

GLOB_NOSORT 正常情况下， **glob()** 根据 **LC_COLLATE** 类别定义的当前处于活动状态的 排序序列对匹配的路径名排序。如果使用此标记，返回的路径名顺序将是不可预测的。

GLOB_NOCHECK 如果 *pattern* 不匹配任何路径名， **glob()** 将返回一个只包括 *pattern* 的列表，同时匹配的路径名数目为 1。

GLOB_DOOFFS 利用 *pglob*–> *gl_offs* 。如果设置了此标记，将使用 *pglob*–> *gl_offs* 指定要向 *pglob*–> *gl_pathv* 的开头添加多少个 NULL 指针。换句话说， *pglob*–> *gl_pathv* 将指向 *pglob*–> *gl_offs* 空指针，其后接 *pglob*–> *gl_pathc* 路径名指针，再后接一个空指针。

GLOB_APPEND 将生成的路径名追加到上一次调用 **glob()** 后生成的路径名后面。

GLOB_APPEND 标记可用于将一组新的路径名追加到上一次调用 **glob()** 后找到的路径名后面。如果使用 *pglob* 的相同值对 **glob()** 执行了两次或更多次调用，并且没有对 **globfree()** 执行附加调用，则可应用以下规则：

- 第一次调用时不得设置 **GLOB_APPEND** 。而所有后续调用必须对其进行设置。
- 所有调用都必须要么设置，要么不设置 **GLOB_DOOFFS** 。
- 第二次调用之后， *pglob –> gl_pathv* 指向包含以下项的列表：
  - **GOB_DOOFFS** 和 *pglob –> gl_offs* 指定的零个或多个空指针。
  - 指向执行调用之前在 *pglob –> gl_pathv* 列表中存在的，按与以前相同的顺序排列的路径名的指针。
  - 指向由第二次调用生成的，按指定顺序排列的新路径名的指针。
- *pglob –> gl_pathc* 中返回的计数是两次调用后生成的路径名总数。
- 调用 **glob()** 后，应用程序可以更改任何字段。如果应用程序要执行更改，则必须在执行后续调用之前，使用相同的 *pglob* 值将这些字段重置为原始值，或者使用 **GLOB_APPEND** 标记将字段重置为 **globfree()** 或 **glob()** 。

如果在搜索期间遇到了无法打开或读取的目录，并且 *errfunc* 不是 NULL， **glob()** 将使用两个参数调用 (**errfunc*errfunc*)( ) ：

- 指向失败的路径的指针。
- 从失败返回的 **errno** 值。

如果调用 *errfunc* 后返回了非零值，或者在 *flags* 中设置了 **GLOB_ERR** 标记，那么 **glob()** 将停止扫描，并且在 *pglob* 中设置 **gl_pathc** 和 **gl_pathv** 以反映已扫描的路径，然后返回 **GLOB_ABORTED** 。如果未设置 **GLOB_ERR** ，同时，无论 *errfunc* 为 NULL，还是 (**errfunc*errfunc*)( ) 返回零，都将忽略错误。

模式匹配表示法

模式的格式是符合文件名扩展要求（请参阅 *regexp*(5) ）的模式匹配表示法，它包括以下例外：

- 不执行波浪符 (~) 扩展。
- 不执行变量扩展。

外部语言环境影响

语言环境

LC_COLLATE 类别可确定编译和执行正则表达式时使用的排序序列，此外，如果未选择 **GLOB_NOSORT** ，该类别还可确定返回的路径的顺序。

LC_CTYPE 类别可确定将文本解释为单字节和（或）多字节字符，及确定正则表达式中的字符类表达式所匹配

的字符。

国际代码集支持

支持单字节字符代码集和多字节字符代码集。

返回值

如果 **glob()** 由于错误而终止，将返回下列其中一个常量（ **<glob.h>** 中已定义）；否则，将返回零。

GLOB_NOSPACE 尝试分配内存失败。

GLOB_ABORTED 由于设置了 **GLOB_ERR** ，或者 (**errfunc*)( ) 返回了零，因此扫描已停止。

GLOB_NOMATCH *pattern* 不匹配任何现有的路径名，并且 *flags* 中未设置 **GLOB_NOCHECK** 。

无论是哪种情况，参数 *pglob–> gl_pathc* 都将返回匹配的路径名数目，同时参数 *pglob–> gl_pathv* 将包含指向匹配的且已排序的路径名的以 null 结尾列表的指针。

但如果 *pglob–> gl_pathc* 为零，则 *pglob–> gl_pathv* 的内容是不确定的。

如果已传递给 **glob()** 的 *pattern* 参数的构建错误， **glob()** 将返回零，同时将 **gl_pathc** 设置为零；但是，在设置了 **GLOB_NOCHECK** 的情况下，将返回 *pattern* ，同时将 **gl_pathc** 设置为 1。

警告

初次调用 **glob()** 时不得设置 **GLOB_APPEND** 。

作者

glob() 和 **globfree()** 由 OSF 和 HP 开发。

另请参阅

sh(1)、 fnmatch(3C)、 regexp(5)、 thread_safety(5)。

符合的标准

glob()： XPG4、 POSIX.2

globfree()： XPG4、 POSIX.2

名称

grantpt - 授予访问 STREAMS 从属 pty 的权限。

概要

int grantpt (int fildes);

说明

已传递的参数 *fildes* 是一个文件描述符，它从一个成功打开的 STREAMS 主 pty（伪终端）设备返回。 **grantpt()** 函数修改与其主 pty 对应设备关联的从属 pty 设备特殊文件的所有权和模式。

采用如下方式派生 **setuid()** 根程序来更改 pty 从属设备文件文件的所有权和模式：将组 ID 设置为名为“tty”的保留组。将从属用户 ID 设置为调用进程的有效所有者。设置从属设备的权限以便使所有者具有读写访问权，并使组具有写入访问权。

返回值

成功完成后， **grantpt()** 函数返回 0（零）。否则，返回 -1。

在下列情况下可能会发生失败：

- 由 *fildes* 参数指定的文件描述符不是打开的文件描述符。
- 由 *fildes* 参数指定的文件描述符与 STREAMS 主 pty 设备没有关联。
- 不能访问相应的从属 pty 设备。

警告

如果应用程序已经安装了信号处理程序， **grantpt()** 函数还可能无法捕获 SIGCHLD（子进程结束）的信号。

举例

以下示例显示了通常使用 **grantpt()** 的方法。

```
int fd_master, fd_slave;
char *slave;
    ...
fd_master = open("/dev/ptmx", O_RDWR);
grantpt(fd_master);
unlockpt(fd_master);
slave = ptsname(fd_master);
fd_slave = open(slave, O_RDWR);
ioctl(fd_slave, I_PUSH, "ptem");
ioctl(fd_slave, I_PUSH, "ldterm");
```

作者

grantpt() 由 HP 和 OSF 联合开发。

另请参阅

open(2)、 unlockpt(3C)、 ptsname(3C)、 ptm(7)、 pts(7)、 ptem(7)、 ldterm(7)。

名称

gss_accept_sec_context( ) - 在应用程序和环境接受器之间建立安全环境

概要

#include <gssapi.h>

OM_uint32 gss_accept_sec_context (
OM_uint32 *minor_status,
gss_ctx_id_t *context_handle,
const gss_cred_id_t acceptor_cred_handle,
const gss_buffer_t input_token_buffer,
const gss_channel_bindings_t input_chan_bindings,
const gss_name_t *src_name,
gss_OID *actual_mech_type,
gss_buffer_t output_token,
int *ret_flags,
OM_uint32 *time_rec
gss_cred_id_t *delegated_cred_handle)

说明

调用 **gss_accept_sec_context()** 例行程序是在环境启动器和环境接受器之间建立安全环境的第二步。在第一步时环境启动器调用了 **gss_init_sec_context()** 例行程序。 **gss_init_sec_context()** 例行程序可为安全环境生成一个标记，并将其传递给环境启动器。环境启动器再将此标记发送给环境接受器。

在第二步，环境接受器将接受环境启动器的调用，然后调用 **gss_accept_sec_context()** 例行程序。需要为 **gss_accept_sec_context()** 例行程序提供 *input_token* 参数的值。 *input_token* 参数的值是由 **gss_init_sec_context()** 例行程序生成的，该值需要由启动器传递给接受器。

gss_accept_sec_context() 例行程序还可以返回 *output_token* 参数的值。环境启动器将为 **gss_init_sec_context()** 例行程序提供标记。如果接受器不必要将标记发送给启动器，那么 **gss_accept_sec_context()** 会将 *output_token* 参数的长度字段设置为 0（零）。

要完成环境的建立，环境启动器可能需要环境接受器的一个或多个答复标记。如果应用程序需要答复标记，**gss_accept_sec_context()** 例行程序将返回包含 **GSS_S_CONTINUE_NEEDED** 的状态值。从环境接受器收到答复标记后，应用程序将再次调用例行程序。应用程序可通过 *output_token* 参数将标记传递给 **gss_accept_sec_context()** 例行程序。

除非例行程序返回了状态 **GSS_S_COMPLETE** ，否则使用 *src_name* 、 *ret_flags* 、 *time_rec* 和 *delegated_cred_handle* 参数返回的值是不可预测的。

输入参数

acceptor_cred_handle 指定环境接受器声明的凭据句柄（身份）。这是可选的信息。凭据只能是 ACCEPT 类型凭据和 BOTH 类型凭据中的一种。指定 **GSS_C_NO_CREDENTIAL** 可以接受将环境用作主体

input_token_buffer 指定从环境接受器收到的标记。

input_chan_bindings 指定环境启动器提供的绑定。使环境启动器能够将通道标识信息安全地绑定到安全环境。如果没有使用通道绑定，请指定 **GSS_C_NO_CHANNEL_BINDINGS** 。

输入/输出参数

context_handle 为新环境指定环境句柄。环境启动器第一次使用该例行程序时，请指定 **GSS_C_NO_CONTEXT** 来设置特定的环境。在后续调用中，将使用该参数返回的值。

输出参数

src_name 返回环境启动器的已认证名称。该信息是可选的。如果不需要已认证的名称，请指定 NULL。要取消分配已认证的名称，请将其传递给 **gss_release_name()** 例行程序。

actual_mech_type 实际使用的机制。如果不需要请指定 NULL。

output_token 返回一个要传递给环境接受器的标记。如果没有要传递给环境启动器的标记，该例行程序会将返回的标记缓冲区的长度字段设置为 0（零）。

ret_flags 返回包含了六个独立标志的位掩码，其中每一个都要求环境支持某个服务选项。提供的下列符号名称将对应于每个标志。应使用 *ret_flags* 的值对这些符号名称进行逻辑 AND 运算，以测试环境是否支持服务选项。

GSS_C_DELEG_FLAG 。True/False 值为：

True 可从 *delegated_cred_handle* 参数获得委派的凭据。

False 没有委派凭据。

GSS_C_MUTUAL_FLAG 。True/False 值为：

True 环境接受器要求相互认证。

False 环境接受器不要求相互认证。

GSS_C_REPLAY_FLAG 。True/False 值为：

True 检测重置的已签名消息或密封消息。

False 不检测重置的消息。

GSS_C_SEQUENCE_FLAG 。True/False 值为：

True 检测无序的已签名消息或密封消息。

False 不检测无序的已签名消息或密封消息。

GSS_C_CONF_FLAG 。True/False 值为：

True 可通过调用 **gss_seal()** 例行程序获得保密服务。

False 不提供保密服务。但是，应用程序可调用 **gss_seal()** 例行程序来提供消息封装、数据源认证和完整性服务。

GSS_C_INTEG_FLAG 。True/False 值为：

True 可通过调用 **gss_sign()** 或 **gss_seal()** 例行程序来调用完整性服务。

False 不提供单个消息的完整性服务。

GSS_C_ANON_FLAG 。True/False 值为：

True 尚未显示启动器的身份，如果已将任何发出的标记传递给了接受器，则不会显示启动器的身份。

False 已按正常方式，或者将按正常方式认证启动器的身份。

GSS_C_PROT_READY_FLAG 。True/False 值为：

True 如果随附的主要状态返回值为 **GSS_S_COMPLETE** 或 **GSS_S_CONTINUE_NEEDED** ，则保护服务（由 **GSS_C_CONF_FLAG** 和 **GSS_C_INTEG_FLAG** 的状态指定）可用。

False 仅当随附的主要状态返回值为 **GSS_S_COMPLETE** 时，保护服务（由 **GSS_C_CONF_FLAG** 和 **GSS_C_INTEG_FLAG** 的状态指定）才可用。

GSS_C_TRANS_FLAG 。True/False 值为：

True 可通过调用 **gss_export_sec_context()** 将得到的安全环境传输给其他进程。

False 不可传输安全环境。

time_rec 返回环境保持有效的秒数。这是可选的信息。如果不需要时间，请指定 NULL。

delegated_cred_handle 返回从环境启动器收到的凭据的凭据句柄。仅当委派的凭据可用时，凭据句柄才有效。如果 *ret_flags* 参数为 True，将设置标志 **GSS_C_DELEG_FLAG** ，以指示委派的凭据可用。

minor_status 返回来自安全机制的状态代码。

状态代码

可返回下列状态代码：

GSS_S_COMPLETE 已成功完成例行程序。

GSS_S_BAD_BINDINGS *input_token* 参数包含的某些通道绑定不同于使用 *input_chan_bindings* 参数指定的通道绑定。

GSS_S_BAD_SIG *input_token* 参数包含无效的签名。

GSS_S_CONTINUE_NEEDED

要完成建立环境，必须使用环境接受器提供的必要的标记再次调用 **gss_accept_sec_context()** 例行程序。

GSS_S_CREDENTIALS_EXPIRED
引用的凭据已过期。

GSS_S_DEFECTIVE_CREDENTIAL
针对凭据执行的一致性检查失败。

GSS_S_DEFECTIVE_TOKEN
针对 *input_token* 参数执行的一致性检查失败。

GSS_S_DUPLICATE_TOKEN
已处理 *input_token* 参数。这是在建立环境的过程中出现的致命错误。

GSS_S_FAILURE 例行程序失败。有关详细信息，请参阅 *minor_status* 参数的返回值。

GSS_S_NO_CONTEXT 提供的环境句柄不引用有效的环境。

GSS_S_NO_CRED 指示要么提供的凭据对环境接受无效，要么凭据句柄不引用任何凭据。

GSS_S_OLD_TOKEN *input_token* 参数太旧。这是在建立环境的过程中出现的致命错误。

GSS_S_BAD_MECH 收到的标记指定了不受支持的机制

作者

gss_accept_sec_context() 由 Sun Microsystems, Inc. 开发。

另请参阅

gss_acquire_cred(3)、 gss_delete_sec_context(3)、 gss_init_sec_context(3)。

DCE-CoreTools 产品提供了 DCE-GSSAPI 的联机帮助页。要参阅这些联机帮助页，请将 **/opt/dce/share/man** 添加到 **MANPATH** 。

名称

gss_acquire_cred( ) - 允许应用程序获取现有命名凭证的句柄

概要

```
#include <gssapi.h>

OM_uint32  gss_acquire_cred (
        OM_uint32 *minor_status,
        const gss_name_t desired_name,
        OM_uint32 time_req,
        const gss_OID_set desired_mechs,
        int cred_usage,
        gss_cred_id_t *output_cred_handle,
        gss_OID_set *actual_mechs,
        OM_int32 *time_rec );
```

说明

gss_acquire_cred() 例行程序允许应用程序获取先前存在的命名凭证的句柄。此凭证可以是 ACCEPT、INITIATE 或 BOTH。接着，应用程序将凭证句柄传递到 **gss_init_sec_context()** 或 **gss_accept_sec_context()** 例行程序。如果 *desired_name* 是 **GSS_C_NO_NAME** ，则将调用解释为调用缺省行为的凭证句柄的请求。

输入参数

desired_name 指定用于凭证的主体名称。

time_req 凭证一直保持有效的秒数。指定 **GSS_C_INDEFINITE** 来请求凭证具有最大允许的生命周期。

desired_mechs 指定用于安全机制的 OID 集合以与凭证一起使用。为了帮助确保应用程序的可移植性，通过指定 **GSS_C_NULL_OID_SET** 来请求缺省的安全机制。

cred_usage 指定下列情况之一：

GSS_C_BOTH 指定环境发起方用来启动或接受安全环境的凭证。

GSS_C_INITIATE 指定环境发起方只用来启动安全环境的凭证。

GSS_C_ACCEPT 指定环境发起方只用来接受安全环境的凭证。

输出参数

output_cred_handle 返回返回凭证的句柄。

actual_mechs 返回一组凭证对其有效的机制。此信息是可选的。如果不需要一组返回的机制，则指定为 NULL。

time_rec 返回返回凭证一直保持有效的实际秒数。此信息是可选的。如果不需要实际秒数，则指定为 NULL。

minor_status 返回安全机制的状态代码。

状态代码

以下列表说明了可返回的 GSS 状态代码：

GSS_S_COMPLETE 例行程序已成功完成。

GSS_S_BAD_MECH 所请求的安全机制不受支持或不可用。

GSS_S_BAD_NAMETYPE 不支持由 *desired_name* 参数传递的名称。

GSS_S_BAD_NAME *desired_name* 参数传递了无效名称。

GSS_S_CREDENTIALS_EXPIRED
无法获取凭证，因为它们已过期。

GSS_S_NO_CRED 没有找到指定名称的凭证。

GSS_S_FAILURE 例行程序失败。有关详细信息，请参阅 *minor_status* 参数返回值。

作者

gss_acquire_cred() 由 Sun Microsystems, Inc. 开发。

另请参阅

gss_init_sec_context(3)。

DCE-GSSAPI 联机帮助页包含在 DCE-CoreTools 产品中。要查看这些联机帮助页，请将 **/opt/dce/share/man** 添加到 **MANPATH** 。

名称

gss_add_cred( ) - 例行程序可将凭据元素添加到凭据。

概要

```
#include <gssapi.h>

OM_uint32  gss_add_cred (
        OM_uint32 *minor_status,
        const gss_cred_id_t input_cred_handle,
        const gss_name_t desired_name,
        const gss_OID desired_mech,
        gss_cred_usage_t  cred_usage,
        OM_uint32 initiator_time_req,
        OM_uint32 acceptor_time_req,
        gss_cred_id_t *output_cred_handle,
        gss_OID_set   *actual_mechs,
        OM_uint32 *initiator_time_rec,
        OM_uint32 *acceptor_time_rec);
```

说明

gss_add_cred() 例行程序可将凭据元素添加到凭据。凭据元素是由其引用的主体的名称标识的。

如果 desired_name 为 **GSS_C_NO_NAME** ，则会将调用解释为请求添加凭据元素，将其传递给 **gss_init_sec_context()** 或 **gss_accept_sec_context()** 时，将调用缺省行为。该例行程序可用于编写包含了除新近获取的凭据元素外，所有原始凭据元素的新凭据，或向现有凭据添加新凭据元素。如果为 *output_cred_handle* 参数指定了 NULL，则将新的凭据元素添加到 *input_cred_handle* 标识的凭据；如果为 *output_cred_handle* 参数指定了有效的指针，则将创建新的凭据句柄。

如果将 **GSS_C_NO_CREDENTIAL** 指定为 *input_cred_handle* ，则 **gss_add_cred()** 将基于缺省行为编写凭据。

输入参数

input_cred_handle 指定将向其添加凭据元素的凭据结构的句柄。如果指定了 **GSS_C_NO_CREDENTIAL**，则该例行程序将基于缺省行为编写新凭据。

desired_name 指定应获取其凭据的主体名。

desired_mechs 指定可对其使用新凭据的安全机制的 OID 集合。

initiator_time_req 指定凭据为启动安全环境而保持有效的秒数。如果编写的凭据的类型为 **GSS_C_ACCEPT** ，则忽略该参数。指定 **GSS_C_INDEFINITE** 请求凭据具有允许的最大启动器生命周期。

acceptor_time_req 指定凭据为接收安全环境而保持有效的秒数。如果编写的凭据的类型为 **GSS_C_INITIATE** ，则忽略该参数。指定 **GSS_C_INDEFINITE** 请求凭据具有允许的最大接收器生命周期。

cred_usage 指定下列其中一项：

GSS_C_BOTH 指定环境启动器可以使用的凭据启动或接收安全环境。

GSS_C_INITIATE 指定环境启动器可以使用的凭据仅启动安全环境。

GSS_C_ACCEPT 指定环境启动器可以使用的凭据仅接收安全环境。

输出参数

output_cred_handle 返回的凭据句柄，包含新的凭据元素及 input_cred_handle 中的所有凭据元素。如果将该参数指定为 NULL，则向 input_cred_handle 标识的凭据添加新近获取的凭据元素。

actual_mechs 返回一组凭据有效的机制。该信息是可选的。如果不需要返回一组机制，请指定 NULL。

initiator_time_rec 指定凭据为启动使用指定机制的安全环境而保持有效的实际秒数。如果实现或机制不支持凭据过期，则将返回值 **GSS_C_INDEFINITE** 。

acceptor_time_rec 指定凭据为接收使用指定机制的安全环境而保持有效的实际秒数。如果实现或机制不支持凭据过期，则将返回值 **GSS_C_INDEFINITE** 。

minor_status 返回安全机制中的状态代码。

状态代码

以下列表说明可以返回的 GSS 状态代码：

GSS_S_COMPLETE 已成功完成例行程序。

GSS_S_BAD_MECH 请求的安全机制不受支持或不可用。

GSS_S_BAD_NAMETYPE *desired_name* 参数传递的名称不受支持。

GSS_S_BAD_NAME *desired_name* 参数传递了无效的名称。

GSS_S_DUPLICATE_ELEMENT
凭据已包含请求机制的具有重叠用途和有效期限的元素。

GSS_S_CREDENTIALS_EXPIRED
所需的凭据已过期，因此无法添加这些凭据。

GSS_S_NO_CRED 未找到指定名称的凭据。

作者

gss_add_cred() 由 Sun Microsystems, Inc. 开发。

另请参阅

gss_init_sec_context(3)。

DCE-CoreTools 产品提供了 DCE-GSSAPI 的联机帮助页。要参阅这些联机帮助页，请将 **/opt/dce/share/man** 添加到 **MANPATH** 。

名称

gss_add_oid_set_member( ) - 将对象标识符 (OID) 添加到 OID 集合

概要

```
#include <gssapi.h>

OM_uint32  gss_add_oid_set_member (
        OM_uint32* minor_status,
        gss_OID* member_OID,
        gss_OID_set* OID_set)
```

说明

gss_add_oid_set_member() 例行程序将新的对象标识符添加到对象标识符设置。如果 OID 集合不存在，则可以使用 **gss_create_empty_oid_set()** 例行程序创建新的、空的 OID 集合。

输入参数

member_OID 指定要添加到 OID 集合的 OID。

OID_set 指定 OID 集合。

输出参数

minor_status 返回安全机制的状态代码。

状态代码

可返回下列状态代码:

GSS_S_COMPLETE 已成功完成例行程序。

GSS_S_FAILURE 例行程序失败。有关详细信息，请检查 *minor_status* 参数。

作者

gss_add_oid_set_member() 由 Sun Microsystems, Inc. 开发。

另请参阅

gss_create_empty_oid_set(3)、 gss_acquire_cred(3)。

DCE-GSSAPI 的联机帮助页随 DCE-CoreTools 产品附送。要查看这些联机帮助页，请将 **/opt/dce/share/man** 添加到 **MANPATH** 。

名称

gss_canonicalize_name( ) - 将内部名称转换为内部机制名称 (MN) 不透明的内部名称的表示

概要

#include <gssapi.h>

OM_uint32 gss_canonicalize_name (
OM_uint32 *minor_status,
const gss_name_t input_name,
const gss_OID *mech_type,
gss_name_t *output_name)

说明

gss_canonicalize_name() 例行程序生成任意内部名称的规范机制名称 (MN)。此机制名称是成功验证环境后返回到环境接受器的名称，在这种机制中，发起方使用 *input_name* 成功调用 **gss_acquire_cred()** ，指定对象标识符 (OID) 集包含 *mech_type* 作为其唯一的成员，接着调用 **gss_init_sec_context()** 来指定 *mech_type* 作为验证机制。

输入参数

input_name 指定规范格式所需的名称。

mech_type 指定名称的规范格式所需的验证机制。必须显式指定所需的机制：没有提供缺省机制。

输出参数

output_name 得到的规范名称。

minor_status 返回安全机制的状态代码。

状态代码

可返回以下状态代码：

GSS_S_COMPLETE 例行程序已成功完成。

GSS_S_BAD_MECH 不支持标识机制。

GSS_S_BAD_NAMETYPE 所提供的内部名称不包括指定机制处理的元素。

GSS_S_BAD_NAME 所提供的内部名称不合理。

GSS_S_FAILURE 例行程序失败。有关详细信息，请检查 *minor_status* 参数。

作者

gss_canonicalize_name() 由 Sun Microsystems, Inc. 开发。

另请参阅

gss_compare_name(3)、 gss_import_name(3)、 gss_release_name(3)。

DCE-GSSAPI 的联机帮助页包含在 DCE-CoreTools 产品中。要查看这些联机帮助页，请将 **/opt/dce/share/man** 添加到 **MANPATH** 。

名称

gss_compare_name( ) - 允许应用程序比较两个内部名称，以确定它们是否相同

概要

```
#include <gssapi.h>

OM_uint32  gss_compare_name (
        OM_uint32 *minor_status,
        const gss_name_t name1,
        const gss_name_t name2,
        int *name_equal)
```

说明

gss_compare_name() 例行程序允许应用程序比较两个内部名称，以确定它们是否相同。该例行程序不会通过对名称进行解析的方式来判定它们是否指向同一对象。它只是比较输入名称，以此来确定它们是否相同。如果提供给 **gss_compare_name()** 的名称之一代表匿名主体，例行程序应告知这两个名称所指向的并非同一个对象标识。

输入参数

name1 指定第一个内部名称。

name2 指定第二个内部名称。

输出参数

name_equal 返回下列值之一:

True 名称相同。

False 名称不同。

minor_status 通过安全机制返回状态代码。

状态代码

可以返回下列状态代码:

GSS_S_COMPLETE 成功完成例行程序。

GSS_S_BAD_NAMETYPE 不支持 *name1* 或 *name2* 参数所传递的名称。

GSS_S_BAD_NAME *name1* 或 *name2* 参数所传递的名称无效。

GSS_S_FAILURE 该例行程序失败。有关详细信息，请检查 *minor_status* 参数。

作者

gss_compare_name() 由 Sun Microsystems, Inc. 开发。

另请参阅

gss_import_name(3)、 gss_release_name(3)。

DCE-GSSAPI 的联机帮助页是随 DCE-CoreTools 产品一起提供的。要查看这些联机帮助页，请将

/opt/dce/share/man 添加至 **MANPATH** 。

名称

gss_context_time( ) - 检查环境将保持有效的秒数

概要

#include <gssapi.h>

OM_uint32 gss_context_time (
OM_uint32 *minor_status,
const gss_ctx_id_t context_handle,
OM_int32 *time_rec)

说明

gss_context_time() 例行程序检查环境将保持有效的秒数。

输入参数

context_handle 指定要检查的环境。

输出参数

time_rec 返回环境将保持有效的秒数。如果环境已经过期，则返回 0（零）。

minor_status 返回安全机制的状态代码。

状态代码

可返回下列状态代码:

GSS_S_COMPLETE 已成功完成例行程序。

GSS_S_CONTEXT_EXPIRED
环境已过期。

GSS_S_NO_CONTEXT 在 *context_handle* 参数中标识的环境是无效的。

GSS_S_FAILURE 例行程序失败。有关详细信息，请参阅 *minor_status* 参数返回值。

作者

gss_context_time() 由 Sun Microsystems, Inc. 开发。

另请参阅

DCE-GSSAPI 的联机帮助页包含在 DCE-CoreTools 产品中。要查看那些联机帮助页，请将 **/opt/dce/share/man** 添加到 **MANPATH** 。

名称

gss_create_empty_oid_set( ) - 创建可向其添加成员的新的、空的 OID 集合

概要

#include <gssapi.h>

OM_uint32 gss_create_empty_oid_set (
OM_uint32 *minor_status,
gss_OID_set *OID_set);

说明

gss_create_empty_oid_set( ) 例行程序创建环境发起方可向其添加成员的新的、空的 OID 集合。使用 **gss_add_oid_set_member( )** 例行程序向 OID 集合添加成员。这些例行程序旨在用于构建一些机制对象标识符集，以供 **gss_acquire_cred( )** 例行程序的输入使用。

输入参数

OID_set 指定要创建的 OID 集合。

输出参数

minor_status 返回安全机制的状态代码。

状态代码

可返回下列状态代码：

GSS_S_COMPLETE 已成功完成例行程序。

GSS_S_FAILURE 例行程序失败。检查 *minor_status* 参数以获得详细信息。

作者

gss_create_empty_oid_set() 由 Sun Microsystems, Inc. 开发。

另请参阅

gss_add_oid_set_member(3)、 gss_acquire_cred(3)。

DCE-GSSAPI 的联机帮助页随 DCE-CoreTools 产品附送。要查看这些联机帮助页，可以将 **/opt/dce/share/man** 添加到 **MANPATH** 。

名称

gss_delete_sec_context( ) - 删除安全环境

概要

#include<gssapi.h>

OM_uint32 gss_delete_sec_context (
OM_uint32 *minor_status,
gss_ctx_id_t *context_handle,
gss_buffer_t output_token_buffer)

说明

gss_delete_sec_context( ) 例行程序删除安全环境。它还删除与此安全环境关联的本地数据结构。当删除了此安全环境后，该例行程序就可生成令牌。应用程序将该令牌传递到环境接受方。该环境接受方接着将此令牌传递到 **gss_process_context_token( )** 例行程序，指示它删除此环境及所有关联的本地数据结构。

当删除此环境后，应用程序就无法使用其他安全服务的 *context_handle* 参数。

输入参数

context_handle 指定要删除的环境的环境句柄。

输出参数

minor_status 返回安全机制的状态代码。

output_token_buffer 返回令牌以传递到环境接受方。

状态代码

可返回下列状态代码：

GSS_S_COMPLETE 已成功完成例行程序。

GSS_S_FAILURE 例行程序失败。有关详细信息，请参阅 *minor_status* 参数返回值。

GSS_S_NO_CONTEXT 所提供的环境句柄没有引用有效的环境。

作者

gss_delete_sec_context() 由 Sun Microsystems, Inc. 开发。

另请参阅

gss_accept_sec_context(3)、 gss_init_sec_context(3)、 gss_process_context_token(3)。

DCE-GSSAPI 的联机帮助页随 DCE-CoreTools 产品附送。要查看这些联机帮助页，可以将 **/opt/dce/share/man** 添加到 **MANPATH** 。

名称

gss_display_name( ) - 向应用程序提供不透明的内部名称的文本形式

概要

#include <gssapi.h>

OM_uint32 gss_display_name (
OM_uint32 *minor_status,
const gss_name_t input_name,
gss_buffer_t output_name_buffer,
gss_OID *output_name_type)

说明

gss_display_name( ) 例行程序向应用程序提供不透明的内部名称的文本格式。应用程序可以使用文本来显示此名称，但不能输出它。

输入参数

input_name 指定要转换成文本的名称。

输出参数

output_name_buffer 返回作为字符串的名称。

output_name_type 返回要显示为指向静态存储的指针的名称类型。应用程序应该将其视为只读。

minor_status 返回安全机制的状态代码。

状态代码

可返回下列状态代码:

GSS_S_COMPLETE 已成功完成例行程序。

GSS_S_BAD_NAMETYPE 不识别 *input_name* 参数传递的名称。

GSS_S_BAD_NAME *input_name* 参数传递了无效名称。

GSS_S_FAILURE 例行程序失败。检查 *minor_status* 参数以获得详细信息。

作者

gss_display_name() 由 Sun Microsystems, Inc. 开发。

另请参阅

gss_compare_name(3)、 gss_import_name(3)、 gss_release_name(3)。

DCE-GSSAPI 的联机帮助页随 DCE-CoreTools 产品附送。要查看这些联机帮助页，可以将 **/opt/dce/share/man** 添加到 **MANPATH** 。

名称

gss_display_status( ) - 提供具有可以对用户显示或进行日志记录的 GSSAPI 状态代码的文本形式的应用程序

概要

```
#include <gssapi.h>

OM_uint32  gss_display_status (
                OM_uint32 *minor_status,
                int status_value,
                int status_type,
                const gss_OID mech_type,
                int *message_context,
                gss_buffer_t status_string)
```

说明

gss_display_status( ) 例行程序提供具有状态代码的文本形式环境启动程序，以便该应用程序对用户显示消息或记录消息。因为有些状态值可以指示一种以上的错误，所以该例行程序启用调用应用程序来处理具有多个消息的状态代码。

message_context 参数指示应用程序应该从 status_value 参数中提取哪个错误消息。应用程序第一次调用例行程序时，它应该将 message_context 参数初始化为 0 并返回第一个消息。如果存在要读取的其他消息，则 **gss_display_status( )** 例行程序返回非零值。应用程序可以反复调用 **gss_display_status( )** 为每个调用生成单个文本字符串。

输入参数

status_value 指定要转换的状态值。

status_type 指定以下状态类型之一：

GSS_C_GSS_CODE 主要状态 - GSS 状态代码

GSS_C_MECH_CODE 次要状态 - 机制状态代码，例如，Kerberos

mech_type 指定基础安全机制。提供 **GSS_C_NULL_OID** 获取系统缺省值。

输入（或输出）

message_context 指示状态代码是否具有多个要读取的消息。应用程序第一次调用例行程序时，需要将参数初始化为 0。例行程序返回第一个消息。如果存在多个消息，则例行程序将参数设置为非零值。应用程序重复调用例行程序以获取下一消息，直到 *message_context* 参数再次为零为止。

输出

status_string 作为文本消息返回状态值。

minor_status 返回安全机制的状态代码。

状态代码

可以返回下列状态代码:

GSS_S_COMPLETE	已成功完成例行程序。
GSS_S_BAD_MECH	转换需要一种不支持或不可用的机制。
GSS_S_BAD_STATUS	指示未识别状态值或状态类型不是 **GSS_C_GSS_CODE** 或 **GSS_C_MECH_CODE**。
GSS_S_FAILURE	例行程序失败。检查 *minor_status* 以获取详细信息。

作者

gss_display_status() 由 Sun Microsystems, Inc. 开发。

另请参阅

DCE-GSSAPI 的联机帮助页随 DCE-CoreTools 产品附送。要查看这些联机帮助页，可以将 **/opt/dce/share/man** 添加到 **MANPATH** 。

名称

gss_duplicate_name( ) - 允许应用程序创建现有的内部名称的原样副本

概要

```
#include <gssapi.h>

OM_uint32  gss_duplicate_name (
        OM_uint32 *minor_status,
        const gss_name_t src_name,
        gss_name_t *dest_name)
```

说明

gss_duplicate_name( ) 例行程序创建现有的内部名称 *src_name* 的原样副本。新的 *dest_name* 将与 *src_name* 无关。

输入参数

src_name 要复制的内部名称。

输出参数

dest_name 得到的 *src_name* 的副本。

minor_status 返回安全机制的状态代码。

状态代码

可返回下列状态代码:

GSS_S_COMPLETE 已成功完成例行程序。

GSS_S_BAD_NAME *src_name* 参数不合理。

作者

gss_duplicate_name() 由 Sun Microsystems, Inc. 开发。

另请参阅

gss_display_name(3)、 gss_import_name(3)、 gss_release_name(3)。

DCE-GSSAPI 的联机帮助页随 DCE-CoreTools 产品附送。要查看这些联机帮助页，可以将 **/opt/dce/share/man** 添加到 **MANPATH** 。

名称

gss_export_name() - 将机制名称 (MN) 转换为适于直接比较的格式

概要

#include <gssapi.h>

OM_uint32 gss_export_name (
OM_uint32 *minor_status,
const gss_name_t input_name,
gss_buffer_t exported_name)

说明

gss_export_name() 将机制名称 (MN) 转换为导出格式。

输入参数

input_name 指定要导出的机制名称 (MN)。

输出参数

exported_name *input_name* 的规范连续的字符串格式。

minor_status 返回安全机制的状态代码。

状态代码

可返回下列状态代码:

GSS_S_COMPLETE 已成功完成例行程序。

GSS_S_BAD_NAMETYPE 不识别 *input_name* 参数传递的名称。

GSS_S_BAD_NAME 该例行程序不能将 *input_name* 参数解释为指定类型的名称。

GSS_S_NAME_NOT_MN 所提供的内部名称不是机制名称。

作者

gss_export_name() 由 Sun Microsystems, Inc. 开发。

另请参阅

gss_compare_name(3)、 gss_display_name(3)、 gss_release_name(3)。

DCE-GSSAPI 的联机帮助页随 DCE-CoreTools 产品附送。要查看这些联机帮助页，可以将 **/opt/dce/share/man** 添加到 **MANPATH** 。

名称

gss_export_sec_context() - 将安全环境传输到一台计算机上的另一进程

概要

#include <gssapi.h>

OM_uint32 gss_export_sec_context (
OM_uint32 *minor_status,
gss_ctx_id_t *context_handle,
gss_buffer_t interprocess_token)

说明

gss_export_sec_context( ) 将停用调用进程的安全环境并创建一个进程间标记。当该标记传给另一进程中的 **gss_import_sec_context( )** 时，它将在第二个进程中重新激活此环境。

对于某个给定的环境而言，在任意时刻只能有它的一个实例处于活动状态。环境导出程序再试图访问导出的安全环境时将会失败。所创建的 **gss_ctx_id_t**

在功能上与原来的环境相同。

如果进程间标记创建成功，实现就会取消所有与安全环境关联的进程范围内的资源分配，并将 *context_handle* 设置为 **GSS_C_NO_CONTEXT** 。创建环境时，将通过 **gss_init_sec_context()** 或 **gss_accept_sec_context()** 在其 *ret_flags* 参数中设置 **GSS_C_TRANS_FLAG** 位来指示能传输安全环境的功能。

输入参数

context_handle 指定用于标识所要传输的环境的环境句柄。

interprocess_token 指定要传输到目标进程的标识。

输出参数

minor_status 通过安全机制返回状态代码。

状态代码

可以返回下列状态代码:

GSS_S_COMPLETE 成功完成例行程序。

GSS_S_CONTEXT_EXPIRED
环境已过期

GSS_S_NO_CONTEXT 提供的环境句柄未引用有效的环境。

GSS_S_UNAVAILABLE 不支持该操作。

作者

gss_export_sec_context() 由 Sun Microsystems, Inc. 开发。

另请参阅

gss_import_sec_context(3)。

DCE-GSSAPI 的联机帮助页是随 DCE-CoreTools 产品一起提供的。要查看这些联机帮助页，请将 **/opt/dce/share/man** 添加至 **MANPATH** 。

名称

gss_get_mic( ) - 计算消息的加密消息完整性代码 (MIC) 并在令牌中返回

概要

```
#include <gssapi.h>

OM_uint32  gss_get_mic (
        OM_uint32 *minor_status,
        const gss_ctx_id_t context_handle,
        gss_qop_t qop_req,
        const gss_buffer_t message_buffer,
        gss_buffer_t msg_token)
```

说明

gss_get_mic( ) 例行程序为所提供的消息生成加密消息完整性代码 (MIC)，并将其放在用于传输到对等端应用程序的令牌中。如果所选机制支持 *qop_req* 参数，则该参数允许在几个加密算法之间进行选择。

输入参数

context_handle	指定发送消息的环境。
qop_req	指定加密算法或保护质量。要接受所选机制提供的缺省保护质量，可以指定 **GSS_C_QOP_DEFAULT** 。
message_buffer	指定要保护的消息。

输出参数

msg_token	接收密封消息的缓冲区。
minor_status	返回安全机制的状态代码。

状态代码

可返回下列状态代码:

GSS_S_COMPLETE	已成功完成例行程序。
GSS_S_CONTEXT_EXPIRED	环境已过期。
GSS_S_CREDENTIALS_EXPIRED	识别了环境，但相关凭据已过期。
GSS_S_NO_CONTEXT	在 *context_handle* 参数中标识的环境是无效的。
GSS_S_BAD_QOP	该机制不支持指定的 QOP。

作者

gss_get_mic() 由 Sun Microsystems, Inc. 开发。

另请参阅

gss_verify_mic(3)。

DCE-GSSAPI 的联机帮助页随 DCE-CoreTools 产品附送。要查看这些联机帮助页，可以将 **/opt/dce/share/man** 添加到 **MANPATH** 。

名称

gss_import_name( ) - 将可输出名称转换为内部格式

概要

#include <gssapi.h>

OM_uint32 gss_import_name (
OM_uint32 *minor_status,
const gss_buffer_t input_buffer_name,
const gss_OID input_name_type,
gss_name_t *output_name)

说明

gss_import_name( ) 例行程序将可输出名称转换为内部格式。

输入参数

input_name_buffer 指定包含要转换的可输出名称的缓冲区。

input_name_type 指定可输出名称的类型的对象标识符。应用程序可以指定 **GSS_C_NULL_OID** 使用本地系统特定的可输出语法，或指定 GSSAPI 实现注册的 OID 命名特定的命名空间。

输出参数

output_name 以内部格式返回名称。

minor_status 返回安全机制的状态代码。

状态代码

可返回下列状态代码:

GSS_S_COMPLETE 已成功完成例行程序。

GSS_S_BAD_NAMETYPE 不识别 *input_name* 参数传递的名称。

GSS_S_BAD_NAME 例行程序不能将 *input_name* 参数解释为指定类型的名称。

GSS_S_BAD_MECH 输入的名称类型为 **GSS_C_NT_EXPORT_NAME** ，但不支持包含在 *input_name* 中的机制。

GSS_S_FAILURE 检查 *minor_status* 参数以获得详细信息。

作者

gss_import_name( ) 由 Sun Microsystems, Inc. 开发。

另请参阅

gss_compare_name(3)、 gss_display_name(3)、 gss_export_name(3)、 gss_release_name(3)。

DCE-GSSAPI 的联机帮助页随 DCE-CoreTools 产品附送。要查看这些联机帮助页，可以将 **/opt/dce/share/man** 添加到 **MANPATH** 。

名称

gss_import_sec_context( ) - 在单个计算机上将安全环境传输到另一个进程

概要

#include <gssapi.h>

OM_uint32 gss_import_sec_context (
OM_uint32 *minor_status,
const gss_buffer_t interprocess_token)
gss_ctx_id_t *context_handle)

说明

gss_import_sec_context( ) 例行程序允许某一进程导入由另一个进程建立的安全环境。对给定的进程间令牌可能仅导入一次。

输入参数

interprocess_token 指定要传输到目标进程的令牌。

输出参数

context_handle 指定新激活的环境的环境句柄。

minor_status 返回安全机制的状态代码。

状态代码

可返回下列状态代码:

GSS_S_COMPLETE 已成功完成例行程序。

GSS_S_NO_CONTEXT 所提供的环境句柄没有引用有效的环境。

GSS_S_DEFECTIVE_TOKEN
令牌无效。

GSS_S_UNAVAILABLE 不支持此操作。

GSS_S_UNAUTHORIZED 本地策略通过当前进程防止导入此环境。

作者

gss_import_sec_context( ) 由 Sun Microsystems, Inc. 开发。

另请参阅

gss_export_sec_context(3)。

DCE-GSSAPI 的联机帮助页随 DCE-CoreTools 产品附送。要查看这些联机帮助页，可以将 **/opt/dce/share/man** 添加到 **MANPATH** 。

名称

gss_indicate_mechs() - 允许应用程序确定哪一个基础安全机制可用

概要

```
#include <gssapi.h>

OM_uint32  gss_indicate_mechs (
        OM_uint32 *minor_status,
        gss_OID_set *mech_set)
```

说明

gss_indicate_mechs( ) 例行程序允许应用程序确定哪一个基础安全机制可用。

输出参数

mech_set 返回支持的安全机制集。 *gss_OID_set* 的值是一个指向静态存储器的指针，环境发起方将其视为只读。

minor_status 返回安全机制的状态代码。

状态代码

可返回下列状态代码:

GSS_S_COMPLETE 已成功完成例行程序。

GSS_S_FAILURE 例行程序失败。检查 *minor_status* 参数以获得详细信息。

作者

gss_indicate_mechs( ) 由 Sun Microsystems, Inc. 开发。

另请参阅

DCE-GSSAPI 的联机帮助页随 DCE-CoreTools 产品附送。要查看这些联机帮助页，可以将 **/opt/dce/share/man** 添加到 **MANPATH** 。

名称

gss_init_sec_context() - 在环境启动器和环境接受器之间建立安全环境

概要

```
#include <gssapi.h>

OM_uint32  gss_init_sec_context (
                OM_uint32 *minor_status,
                const gss_cred_id_t claimant_cred_handle,
                gss_ctx_id_t *context_handle,
                gss_name_t target_name,
                const gss_OID mech_type,
                int req_flags,
                int time_req,
                const gss_channel_bindings_t input_channel_bindings,
                const gss_buffer_t input_token,
                gss_OID *actual_mech_types,
                gss_buffer_t output_token,
                int *ret_flags,
                OM_int32 *time_rec)
```

说明

调用 **gss_init_sec_context()** 例行程序是在环境启动器和环境接受器之间建立安全环境的第一步。为确保应用程序的可移植性，请通过将 **GSS_C_NO_CREDENTIAL** 提供给 *claimant_cred_handle* 参数来使用应用程序的缺省凭据。当应用程序需要其他凭据时，可指定一个显式凭据；例如，使用委派。

当应用程序第一次调用 **gss_init_sec_context()** 例行程序时，请将 *input_token* 参数指定为 **GSS_NO_BUFFER** 。调用此例行程序可返回要传输给环境接受器的 *output_token* 。环境启动器将为 **gss_accept_sec_context()** 例行程序提供标记。

如果环境启动器不需要标记， **gss_init_sec_context()** 会将 *output_token* 参数的长度字段设置为 0（零）。

要完成环境的建立，调用应用程序可能需要环境接受器的一个或多个答复标记。如果应用程序需要答复标记，**gss_init_sec_context()** 例行程序将返回 **GSS_S_CONTINUE_NEEDED** 的状态值。从环境接受器收到答复标记后，应用程序将再次调用例行程序，然后通过 *input_token* 参数将标记传递给 **gss_init_sec_context()** 例行程序。

除非例行程序返回了状态 **GSS_S_COMPLETE** ，否则 *ret_flags* 和 *time_rec* 参数返回的值是不可预测的。

如果初次调用 **gss_init_sec_context()** 失败，则调用不创建环境对象，并且使 *context_handle* 参数的值设置为 **GSS_C_NO_CONTEXT** ，以指示失败。

输入参数

claimant_cred_handle 指定凭据的可选句柄。要使用缺省的凭据，请提供 **GSS_C_NO_CREDENTIAL** 。创建的凭据句柄将引用 DCE 缺省的登录环境。如果未定义缺省的启动器，函数将返回

GSS_S_NO_CRED 。

target_name 指定环境接受器的名称。

mech_type 指定安全机制。提供 **GSS_C_NO_OID** 以获取特定实现的缺省值。

req_flags 指定独立的标志，其中每一个都要求环境支持某个服务选项。提供的下列符号名称将对应于每个标志。应对这些符号名称进行逻辑 OR 运算，以形成位掩码值。

GSS_C_DELEG_FLAG 。True/False 值为：

True 已将凭据委派给环境接受器。

False 没有委派凭据。

GSS_C_MUTUAL_FLAG 。True/False 值为：

True 要求环境接受器认证自身。

False 未要求环境启动器认证自身。

GSS_C_REPLAY_FLAG 。True/False 值为：

True 检测重置的已签名消息或密封消息。

False 不检测重置的消息。

GSS_C_SEQUENCE_FLAG 。True/False 值为：

True 检测无序的已签名消息或密封消息。

False 不检测无序的已签名消息或密封消息。

GSS_C_CONF_FLAG 。True/False 值为：

True 要求提供保密服务

False 不需要每消息保密服务。

GSS_C_INTEG_FLAG 。True/False 值为：

True 要求提供完整性服务

False 不需要每消息完整性服务。

GSS_C_ANON_FLAG 。True/False 值为：

True 不向接受器显示启动器的身份。

False 正常认证。

time_req 指定环境保持有效的所需秒数。要指定缺省的有效期限，请使用 0（零）。

input_chan_bindings 指定环境启动器设置的绑定。使环境启动器能够将通道标识信息安全地绑定到安全环境。如果没有使用通道绑定，请指定 **GSS_C_NO_CHANNEL_BINDINGS** 。

input_token 指定从环境接受器收到的标记。

当应用程序第一次调用该例行程序时，请指定 **GSS_NO_BUFFER** 。后续调用需要有环境接受器的标记。

输入/输出参数

context_handle 为新环境指定环境句柄。

当应用程序第一次调用该例行程序时，请指定 **GSS_C_NO_CONTEXT** 。后续调用将使用第一次调用返回的值。

输出参数

actual_mech_type 返回实际使用的机制的 OID。如果不需要请指定 NULL。

output_token 返回要传递给环境接受器的标记。如果返回的缓冲区的长度字段为 0（零），则不发送标记。

ret_flags 返回六个独立的标志，其中每一个都指示环境可支持某个服务选项。如果不需要请指定 NULL。提供的下列符号名称将对应于每个标志：

GSS_C_DELEG_FLAG 。True/False 值为：

True 已将凭据委派给环境接受器。

False 没有委派凭据。

GSS_C_MUTUAL_FLAG 。True/False 值为：

True 要求环境接受器认证自身。

False 不要求环境接受器认证自身。

GSS_C_REPLAY_FLAG 。True/False 值为：

True 检测重置的已签名消息或密封消息。

False 不检测重置的消息。

GSS_C_SEQUENCE_FLAG 。True/False 值为：

True 检测无序的已签名消息或密封消息。

False 不检测无序的已签名消息或密封消息。

GSS_C_CONF_FLAG 。True/False 值为：

True 可通过调用 **gss_seal()** 例行程序来调用保密服务。

False 不提供保密服务（可以使用 **gss_seal()** 例行程序来提供保密，这只会提供消息封装、数据源认证和完整性服务）。

GSS_C_INTEG_FLAG 。True/False 值为:

True 可通过调用 **gss_get_mic()** 或 **gss_wrap()** 例行程序来调用完整性服务。

False 不提供单个消息的完整性服务。

GSS_C_ANON_FLAG 。True/False 值为:

True 不向接受器显示启动器的身份。

False 正常认证。

GSS_C_PROT_READY_FLAG 。True/False 值为:

True 如果随附的主要状态为 **GSS_S_COMPLETE** 或 **GSS_S_CONTINUE_NEEDED** ，则保护服务（由 **GSS_C_CONF_FLAG** 和 **GSS_C_INTEG_FLAG** 的状态指定）可用。

False 如果随附的主要状态为 **GSS_S_COMPLETE** ，则保护服务（由 **GSS_C_CONF_FLAG** 和 **GSS_C_INTEG_FLAG** 的状态指定）可用。

GSS_C_TRANS_FLAG 。True/False 值为:

True 可以将得到的安全环境传输给其他进程

False 不可传输安全环境

time_rec 返回环境有效的秒数。如果机制不支持凭据过期，该例行程序将返回值 **GSS_C_INDEFINITE** 。如果不需要凭据过期时间，请指定 NULL。

minor_status 返回来自安全机制的状态代码。

状态代码

可返回下列状态代码:

GSS_S_COMPLETE 已成功完成例行程序。

GSS_S_BAD_BINDINGS *input_token* 参数包含的某些通道绑定不同于使用 *input_chan_bindings* 参数指定的通道绑定。

GSS_S_BAD_NAMETYPE *target_name* 参数包含无效的或不支持的名称类型。

GSS_S_BAD_NAME *target_name* 参数的格式不当。

GSS_S_BAD_SIG 指示要么 *input_token* 参数包含无效的签名，要么 *input_token* 参数包含无法验证的签名。

GSS_S_CONTINUE_NEEDED

要完成建立环境，必须使用环境接受器提供的必要的标记再次调用 **gss_init_sec_context()** 例行程序。

GSS_S_CREDENTIALS_EXPIRED
引用的凭据已过期。

GSS_S_DEFECTIVE_CREDENTIAL
针对凭据执行的一致性检查失败。

GSS_S_DEFECTIVE_TOKEN
针对 *input_token* 参数执行的一致性检查失败。

GSS_S_DUPLICATE_TOKEN
已处理 *input_token* 参数。这是在建立环境的过程中出现的致命错误。

GSS_S_FAILURE 例行程序失败。有关详细信息，请参阅 *minor_status* 参数的返回值。

GSS_S_NO_CONTEXT 提供的环境句柄不引用有效的环境。

GSS_S_OLD_TOKEN *input_token* 参数太旧。这是在建立环境的过程中出现的致命错误。

GSS_S_NO_CRED 提供的凭据对环境启动无效，或者凭据句柄不引用任何凭据。

GSS_S_BAD_MECH 提供的凭据不支持指定的机制

作者

gss_init_sec_context() 由 Sun Microsystems, Inc. 开发。

另请参阅

gss_accept_sec_context(3)、 gss_delete_sec_context(3)。

DCE-CoreTools 产品提供了 DCE-GSSAPI 的联机帮助页。要参阅这些联机帮助页，请将 **/opt/dce/share/man** 添加到 **MANPATH** 。

名称

gss_inquire_context() - 获得安全环境的信息

概要

```
#include <gssapi.h>

OM_uint32  gss_inquire_context (
        OM_uint32 *minor_status,
        const gss_ctx_id_t context_handle,
        gss_name_t *src_name,
        gss_name_t *target_name,
        OM_uint32 *lifetime_rec,
        gss_OID *mech_type)
        OM_uint32 *ctx_flags,
        int *locally_initiated,
        int *open)
```

说明

gss_inquire_context() 例行程序为调用应用程序提供安全环境信息。该调用应用程序首先必须为凭证调用了用于句柄的 **gss_acquire_cred()** 例行程序。

输入参数

context_handle 引用安全环境的句柄。

输出参数

src_name 环境发起方的名称。如果使用匿名身份验证建立环境，并且应用程序调用 **gss_inquire_context()** 是环境接受方，则将返回匿名名称。如果没有要求，则指定为 NULL。

context_handle 环境接受方的名称。如果环境接受方对自己不进行身份验证，并且发起方没有在其对 **gss_init_sec_context()** 的调用中指定目标名称，则将返回值 **GSS_C_NO_NAME** 。如果没有要求，则指定为 NULL。

lifetime_rec 返回凭证一直保持有效的秒数。如果凭证过期，则参数返回 0（零）。如果凭证没有过期，则参数返回值 **GSS_C_INDEFINITE** 。如果对过期时间没有要求，则指定为 NULL。

targ_name 环境接受方的名称。调用 **gss_release_name()** 之后，应用程序必须释放与该名称关联的存储空间。

mech_type 提供该环境的安全机制。如果没有要求，则指定为 NULL

ctx_flags 指定独立的标志，每个标志请求该环境支持服务选项。提供以下对应于每个标志的符号名称。应该对这些符号名称进行逻辑或运算，以形成位掩码值。

GSS_C_DELEG_FLAG 。True/False 值为:

True 将凭证委托给环境接受方。

False 没有委托任何凭证。

GSS_C_MUTUAL_FLAG 。True/False 值为:

True 要求环境接受方对自己进行身份验证。

False 不要求环境接受方对自己进行身份验证。

GSS_C_REPLAY_FLAG 。True/False 值为:

True 将检测重播的有符号或密封的消息。

False 不检测重播的消息。

GSS_C_SEQUENCE_FLAG 。True/False 值为:

True 将检测无序有符号或密封的消息。

False 不检测无序有符号或密封的消息。

GSS_C_CONF_FLAG 。True/False 值为:

True 请求保密服务可用。

False 对每条消息保密服务没有要求。

GSS_C_INTEG_FLAG 。True/False 值为:

True 请求完整性服务有用。

False 对每条消息的完整性服务没有要求。

GSS_C_ANON_FLAG 。True/False 值为:

True 没有向接受方显示发起方的标识。

False 通常进行身份验证。

cred_usage 返回下列描述应用程序可如何使用凭证的值之一:

GSS_C_INITIATE
GSS_C_ACCEPT
GSS_C_BOTH

如果对信息的使用没有要求，则指定为 NULL。

mechs 返回一组凭证所支持的安全机制。如果没有要求，则指定为 NULL。

minor_status 返回安全机制的状态代码。

状态代码

可返回以下状态代码：

GSS_S_COMPLETE 已成功完成例行程序。

GSS_S_NO_CONTEXT 无法访问引用的环境。

GSS_S_DEFECTIVE_CREDENTIAL
凭证是无效的。

GSS_S_FAILURE 例行程序失败。检查 *minor_status* 参数以获得详细信息。

GSS_S_NO_CRED 例行程序无法访问这些凭证。

作者

gss_inquire_context() 由 Sun Microsystems, Inc. 开发。

另请参阅

gss_acquire_cred(3)。

DCE-GSSAPI 的联机帮助页包含在 DCE-CoreTools 产品中。要查看这些联机帮助页，可以将 **/opt/dce/share/man** 添加到 **MANPATH** 。

名称

gss_inquire_cred() - 提供有关凭据的调用应用程序信息

概要

```
#include <gssapi.h>

OM_uint32  gss_inquire_cred (
        OM_uint32 *minor_status,
        const gss_cred_id_t cred_handle,
        gss_name_t *name,
        OM_uint32 *lifetime,
        int *cred_usage,
        gss_OID_set *mechs)
```

说明

gss_inquire_cred() 例行程序提供有关凭据的调用应用程序信息。该调用应用程序首先必须为凭据调用了用于句柄的 **gss_acquire_cred()** 例行程序。

输入参数

cred_handle 指定目标凭据的句柄。要获取有关缺省凭据的信息，可以指定 **GSS_C_NO_CREDENTIAL** 。

输出参数

name 返回凭据所断言的主体名称。如果对主体名称没有要求，则指定 NULL。

lifetime 返回凭据一直保持有效的秒数。

如果凭据过期，则参数返回 0 （零）。如果凭据没有过期，则参数返回值 **GSS_C_INDEFINITE** 。如果对过期时间没有要求，则指定 NULL。

cred_usage 返回下列描述应用程序可如何使用凭据的值之一:

GSS_C_INITIATE
GSS_C_ACCEPT
GSS_C_BOTH

如果对信息的使用没有要求，则指定 NULL。

mechs 返回一组凭据所支持的安全机制。

minor_status 返回安全机制的状态代码。

状态代码

可返回下列状态代码:

GSS_S_COMPLETE 已成功完成例行程序。

GSS_S_CREDENTIALS_EXPIRED
凭据过期。如果将生命周期作参数为 NULL 传递，则将该参数设置为 0（零）。

GSS_S_DEFECTIVE_CREDENTIAL
凭据无效。

GSS_S_FAILURE 例行程序失败。检查 *minor_status* 参数以获得详细信息。

GSS_S_NO_CRED 例行程序无法访问凭据。

作者

gss_inquire_cred() 由 Sun Microsystems, Inc. 开发。

另请参阅

gss_acquire_cred(3)。

DCE-GSSAPI 的联机帮助页随 DCE-CoreTools 产品附送。要查看这些联机帮助页，可以将 **/opt/dce/share/man** 添加到 **MANPATH** 。

名称

gss_inquire_cred_by_mech() - 提供调用应用程序有关凭据的每种机制信息

概要

#include <gssapi.h>.B

OM_uint32 gss_inquire_cred_by_mech (
OM_uint32 *minor_status ,
const gss_cred_id_t cred_handle ,
const gss_OID *mech_type)
gss_name_t *name ,
OM_uint32 *initiator_lifetime ,
OM_uint32 *acceptor_lifetime,
int *cred_usage)

说明

gss_inquire_cred_by_mech() 例行程序可提供有关凭据的每种机制信息。

输入参数

cred_handle 为目标凭据指定一个句柄。要获取有关缺省凭据的信息，请指定 **GSS_C_NO_CREDENTIAL** 。

mech_type 应为其返回信息的安全机制的设置。

输出参数

name 返回由凭据声明的主体名称。如果不需要主体名称，请指定 NULL。

initiator_lifetime 返回在指定的机制下能够启动安全环境的有效秒数。如果无法再使用此凭据来启动环境，或者用于此机制的凭据为 **GSS_C_ACCEPT** ，此参数就会被设置为零。

如果实现不支持让初始凭据过期，此参数就会返回值 **GSS_C_INDEFINITE** 。如果不需要过期时间，请指定 NULL。

acceptor_lifetime 返回在指定的机制下能够接受安全环境的有效秒数。

如果实现不支持让初始凭据过期，此参数就会返回值 **GSS_C_INDEFINITE** 。如果不需要过期时间，请指定 NULL。

cred_usage 返回下列值之一，描述应用程序可以如何使用凭据：

GSS_C_INITIATE
GSS_C_ACCEPT
GSS_C_BOTH

如果不需要使用信息，请指定 NULL。

minor_status 通过安全机制返回状态代码。

状态代码

可以返回下列状态代码:

GSS_S_COMPLETE 成功完成例行程序。

GSS_S_CREDENTIALS_EXPIRED
凭据已过期。如果传递的 *lifetime* 参数值为 NULL，就会将其设置为 0（零）。

GSS_S_DEFECTIVE_CREDENTIAL
凭据无效。

GSS_S_FAILURE 该例行程序失败。有关详细信息，请检查 *minor_status* 参数。

GSS_S_NO_CRED 该例行程序无法访问凭据。

作者

gss_inquire_cred_by_mech() 由 Sun Microsystems, Inc. 开发。

另请参阅

gss_acquire_cred(3)。

DCE-GSSAPI 的联机帮助页是随 DCE-CoreTools 产品一起提供的。要查看这些联机帮助页，请将 **/opt/dce/share/man** 添加至 **MANPATH** 。

名称

gss_inquire_mechs_for_name() - 列出支持指定名称类型的机制

概要

```
#include <gssapi.h>

OM_uint32  gss_inquire_mechs_for_name (
        OM_uint32 *minor_status,
        const gss_name_t input_name,
        gss_OID_set *mech_types)
```

说明

gss_inquire_mechs_for_name() 例行程序返回可能可以处理指定名称的 GSS-API 实现所支持的机制集。

输入参数

input_name 与查询相关的名称。

输出参数

mech_types 可能支持指定名称的机制集。 **gss_OID_set** 的值是一个指向静态存储器的指针，环境发起方将其视为只读。

minor_status 返回安全机制的状态代码。

状态代码

可返回下列状态代码:

GSS_S_COMPLETE 已成功完成例行程序。

GSS_S_BAD_NAME *input_name* 参数不合理。

GSS_S_BAD_NAMETYPE *input_name* 参数包含无效的或不受支持的名称类型。

作者

gss_inquire_mechs_for_name() 由 Sun Microsystems, Inc. 开发。

另请参阅

DCE-GSSAPI 的联机帮助页包含在 DCE-CoreTools 产品中。要查看这些联机帮助页，请将 **/opt/dce/share/man** 添加到 **MANPATH** 。

名称

gss_inquire_names_for_mech() - 列出指定机制所支持的名称类型

概要

```
#include <gssapi.h>

OM_uint32  gss_inquire_names_for_mech (
        OM_uint32 *minor_status,
        const gss_OID *mechanism,
        gss_OID_set *name_types)
```

说明

gss_inquire_names_for_mech() 例行程序返回指定机制所支持的名称类型集。

输入参数

mechanism 要询问的机制。

输出参数

name_types 指定机制所支持的名称类型集。 **gss_OID_set** 的值是一个指向静态存储器的指针，环境发起方将其视为只读。

minor_status 返回安全机制的状态代码。

状态代码

可返回下列状态代码:

GSS_S_COMPLETE 已成功完成例行程序。

GSS_S_FAILURE 例行程序失败。检查 *minor_status* 参数以获得详细信息。

作者

gss_inquire_names_for_mech() 由 Sun Microsystems, Inc. 开发。

另请参阅

DCE-GSSAPI 的联机帮助页包含在 DCE-CoreTools 产品中。要查看这些联机帮助页，请将 **/opt/dce/share/man** 添加到 **MANPATH** 。

名称

gss_process_context_token() - 将环境传递给安全服务

概要

#include <gssapi.h>

OM_uint32 gss_process_context_token (
OM_uint32 *minor_status,
const gss_ctx_id_t *context_handle,
const gss_buffer_t input_token_buffer)

说明

gss_process_context_token() 例行程序可将 **gss_delete_security_context()** 例行程序所生成的标记传递给安全服务。

标记通常与环境的建立或与每条消息的安全服务关联。如果标记与环境的建立关联，就会被传递给 **gss_init_sec_context()** 或 **gss_accept_sec_context()** 例行程序。如果标记与每条消息的安全服务关联，就会被传递给 **gss_verify()** 或 **gss_unseal()** 例行程序。

gss_process_context_token() 例行程序将把 **gss_delete_security_context()** 例行程序所生成的标记传给安全服务进行处理。

输入参数

context_handle 指定安全服务藉此处理标记用的环境句柄。

input_token_buffer 指定一个不透明指针，指向所要处理的标记的第一个字节。

输出参数

minor_status 通过安全机制返回状态代码。

状态代码

可以返回下列状态代码:

GSS_S_COMPLETE 成功完成例行程序。

GSS_S_DEFECTIVE_TOKEN
对 input_token 参数执行的一致性检查失败。

GSS_S_FAILURE 该例行程序失败。有关详细信息，请参阅 minor_status 参数的返回值。

GSS_S_NO_CONTEXT 提供的环境句柄未引用有效的环境。

作者

gss_process_context_token() 由 Sun Microsystems, Inc. 开发。

另请参阅

gss_accept_sec_context(3) 、 gss_delete_sec_context(3) 、 gss_init_sec_context(3) 、 gss_verify_mic(3) 、 gss_unseal(3)。

DCE-GSSAPI 的联机帮助页是随 DCE-CoreTools 产品一起提供的。要查看这些联机帮助页，请将 **/opt/dce/share/man** 添加至 **MANPATH** 。

名称

gss_release_buffer() - 释放与缓冲区关联的存储空间

概要

```
#include <gssapi.h>

OM_uint32  gss_release_buffer (
        OM_uint32 *minor_status,
        gss_buffer_t buffer)
```

说明

gss_release_buffer() 例行程序通过释放与缓冲区关联的存储空间来删除该缓冲区。

输入参数

buffer	要删除的缓冲区。

输出参数

minor_status	返回安全机制的状态代码。

状态代码

可返回下列状态代码:

GSS_S_COMPLETE	已成功完成例行程序。
GSS_S_FAILURE	例行程序失败。查看 *minor_status* 参数以获得详细信息。

作者

gss_release_buffer() 由 Sun Microsystems, Inc. 开发。

另请参阅

DCE-GSSAPI 的联机帮助页包含在 DCE-CoreTools 产品中。要查看这些联机帮助页，请将 **/opt/dce/share/man** 添加到 **MANPATH** 。

名称

gss_release_cred() - 标记删除凭据

概要

#include <gssapi.h>

OM_uint32 gss_release_cred (
OM_uint32 *minor_status,
gss_cred_id_t *cred_handle)

说明

gss_release_cred() 例行程序通知 GSSAPI 不再需要凭据，并将其标记为删除。

输入参数

cred_handle 指定包含要释放的不透明凭据句柄的缓冲区。要释放缺省凭据，可以指定 **GSS_C_NO_CREDENTIAL** 。

输出参数

minor_status 返回安全机制的状态代码。

状态代码

可返回下列状态代码:

GSS_S_COMPLETE 已成功完成例行程序。

GSS_S_NO_CRED 无法访问凭据。

作者

gss_release_cred() 由 Sun Microsystems, Inc. 开发。

另请参阅

DCE-GSSAPI 的联机帮助页包含在 DCE-CoreTools 产品中。要查看这些联机帮助页，请将 **/opt/dce/share/man** 添加到 **MANPATH** 。

名称

gss_release_name() - 释放与 GSSAPI 例行程序分配的内部名称关联的存储空间

概要

```
#include <gssapi.h>

OM_uint32  gss_release_name (
        OM_uint32 *minor_status,
        gss_name_t *name)
```

说明

gss_release_name() 例行程序通过释放与内部名称关联的存储空间来删除该内部名称。

输入参数

name 要删除的名称。

输出参数

minor_status 返回安全机制的状态代码。

状态代码

可返回下列状态代码:

GSS_S_COMPLETE 已成功完成例行程序。

GSS_S_BAD_NAME *name* 参数未包含有效的名称。

GSS_S_FAILURE 例行程序失败。检查 *minor_status* 参数以获得详细信息。

作者

gss_release_name() 由 Sun Microsystems, Inc. 开发。

另请参阅

gss_compare_name(3)、 gss_display_name(3)、 gss_import_name(3)。

DCE-GSSAPI 的联机帮助页随 DCE-CoreTools 产品附送。要查看这些联机帮助页，可以将 **/opt/dce/share/man** 添加到 **MANPATH** 。

名称

gss_release_oid_set() - 释放与 gss_OID_set 对象关联的存储空间

概要

```
#include <gssapi.h>

OM_uint32 gss_release_oid_set (
        OM_uint32 *minor_status,
        gss_OID_set *set)
```

说明

gss_release_oid_set() 例行程序释放与 *gss_OID_set* 参数关联，并由 GSSAPI 例行程序分配的的存储空间。

输入参数

set 将删除与 *gss_OID_set* 关联的存储空间。

输出参数

minor_status 返回安全机制的状态代码。

状态代码

可返回下列状态代码:

GSS_S_COMPLETE 已成功完成例行程序。

GSS_S_FAILURE 例行程序失败。检查 *minor_status* 参数以获得详细信息。

作者

gss_release_oid_set() 由 Sun Microsystems, Inc. 开发。

另请参阅

DCE-GSSAPI 的联机帮助页包含在 DCE-CoreTools 产品中。要查看这些联机帮助页，请将 **/opt/dce/share/man** 添加到 **MANPATH** 。

名称

gss_test_oid_set_member() - 检查 OID 集合，查找所指定的 OID

概要

```
#include <gssapi.h>

OM_uint32  gss_test_oid_set_member (
        OM_uint32 *minor_status,
        const gss_OID member_OID,
        const gss_OID_set set,
        int* is_present /* 1 = present, 0 = absent */ );
```

说明

gss_test_oid_set_member() 例行程序可检查 OID 集合，从而确定所指定的 OID 是否为该 OID 集合的成员。要为 OID 集添加成员，请使用 **gss_add_oid_set_member()** 例行程序。

gss_test_oid_set_member() 例行程序将使用 **gss_acquire_cred()** 例行程序的 *actual_mechs* 输出参数值获取 OID 的列表。它将对此列表进行检查，以确定是否有 OID 是该 OID 集合的成员。此例行程序将与 **gss_indicate_mechs()**、 **gss_acquire_cred()** 及 **gss_inquire_cred()** 返回的 OID 集合配合使用。

输入参数

member_OID 指定要在 OID 集合内搜索的 OID。

set 指定所要检查的 OID 集合。

输出参数

is_present 返回以下值之一，用于指示此 OID 是否为该 OID 集合的成员：

1 如果此 OID 是该 OID 集合的成员。

0 如果此 OID 不存在，因此不是该 OID 集合的成员。

minor_status 通过安全机制返回状态代码。

状态代码

可以返回下列状态代码：

GSS_S_COMPLETE 成功完成例行程序。

GSS_S_FAILURE 该例行程序失败。有关详细信息，请检查 *minor_status* 参数。

作者

gss_test_oid_set_member() 由 Sun Microsystems, Inc. 开发。

另请参阅

gss_add_oid_set_member(3)、 gss_acquire_cred(3)、 gss_indicate_mechs(3)。

DCE-GSSAPI 的联机帮助页是随 DCE-CoreTools 产品一起提供的。要查看这些联机帮助页，请将

/opt/dce/share/man 添加至 **MANPATH** 。

名称

gss_unwrap - 如有必要，可使用附加消息完整性代码 (MIC) 验证消息，并解密消息内容

概要

```
#include <gssapi.h>

OM_uint32  gss_unwrap (
        OM_uint32 *minor_status,
        gss_ctx_id_t context_handle,
        gss_buffer_t input_message_buffer,
        gss_buffer_t output_message_buffer,
        int *conf_state,
        gss_qop_t *qop_state)
```

说明

gss_unwrap() 例行程序将保护消息转换为有用的格式，并验证嵌入的消息完整性代码 (MIC)。 *conf_state* 参数指示是否对消息进行了加密。 *qop_state* 参数指示用于提供机密性和完整性服务的保护质量。

输入参数

context_handle 指定消息到达的环境。

input_message_buffer 指定保护消息。

output_message_buffer 指定接收未包装的消息的缓存区。

输出参数

conf_state 返回机密性和完整性服务的请求级别，如下所示：

Non-zero 使用了机密性和完整性服务。

zero 仅使用完整性服务。

qop_state 返回加密算法或保护质量。

minor_status 返回安全机制的状态代码。

状态代码

可返回下列状态代码：

GSS_S_COMPLETE 已成功完成例行程序。

GSS_S_BAD_SIG 签名不正确。

GSS_S_CONTEXT_EXPIRED
环境已过期。

GSS_S_CREDENTIALS_EXPIRED
识别了环境，但相关凭据已过期。

GSS_S_DEFECTIVE_TOKEN
令牌无法进行一致性检查。

GSS_S_DUPLICATE_TOKEN
令牌是有效的，并包含正确的签名，但已经被处理。

GSS_S_FAILURE 例行程序失败。在 *context_handle* 参数中指定的环境是无效的。

GSS_S_NO_CONTEXT 在 *context_handle* 参数中标识的环境是无效的。

GSS_S_OLD_TOKEN 令牌是有效的，并包含正确的签名，但是它太旧了。

GSS_S_UNSEQ_TOKEN 令牌是有效的，并包含消息的正确 MIC，但是经验证它是无序的：已接收了后面的令牌。

GSS_S_GAP_TOKEN 令牌是有效的，并包含消息的正确 MIC，但是经验证它是无序的：还未接收前面需要的令牌。

作者

gss_unwrap() 由 Sun Microsystems, Inc. 开发。

另请参阅

gss_get_mic(3)、gss_wrap(3)。

名称

gss_verify_mic() - 依据消息检查加密的消息完整性代码 (MIC)，以验证其完整性

概要

```
#include <gssapi.h>

OM_uint32  gss_verify_mic(
        OM_uint32 *minor_status,
        const gss_ctx_id_t context_handle,
        const gss_buffer_t message_buffer,
        const gss_buffer_t token_buffer)
        gss_qop_t *qop_state)
```

说明

gss_verify_mic() 例行程序可验证加密的 MIC （包含在 *token_buffer* 参数中）是否适合所提供的消息。接收消息的应用程序可使用 *qop_state* 参数检查保护的力度。

输入参数

context_handle 指定收到该消息的环境。

message_buffer 指定要验证的消息。

token_buffer 指定与此消息关联的标记。

输出参数

qop_state 返回通过消息完整性代码 (MIC) 所了解到的保护力度。如果不需要，请指定 NULL。

minor_status 通过安全机制返回状态代码。

状态代码

可以返回下列状态代码:

GSS_S_COMPLETE 成功完成例行程序。

GSS_S_DEFECTIVE_TOKEN 对标记进行的一致性检查失败。

GSS_S_BAD_SIG MIC 不正确。

GSS_S_DUPLICATE_TOKEN 标记有效，且包含对于消息而言正确的 MIC，但它已被处理过。

GSS_S_OLD_TOKEN 标记有效，且包含对于消息而言正确的 MIC，但它太旧。

GSS_S_UNSEQ_TOKEN 标记有效，且包含对于消息而言正确的 MIC ，但是经过验证，它是失序的。已收到更新的标记。

GSS_S_GAP_TOKEN 标记有效，且包含对于消息而言正确的 MIC ，但是经过验证，它是失序的。有个标记本来预计早就能收到，但尚未接收到。

GSS_S_CONTEXT_EXPIRED
环境已过期。

GSS_S_FAILURE 该例行程序失败。有关详细信息，请检查 *minor_status* 参数。

GSS_S_NO_CONTEXT *context_handle* 参数中所标识的环境无效。

作者

gss_verify_mic() 由 Sun Microsystems, Inc. 开发。

另请参阅

gss_get_mic(3)、gss_wrap(3)。

DCE-GSSAPI 的联机帮助页是随 DCE-CoreTools 产品一起提供的。要查看这些联机帮助页，请将 **/opt/dce/share/man** 添加至 **MANPATH** 。

名称

gss_wrap() - 向消息上附加消息完整性代码 (MIC) 并可加密消息内容

概要

#include <gssapi.h>

OM_uint32 gss_wrap (
OM_uint32 *minor_status,
const gss_ctx_id_t context_handle,
int conf_req_flag,
gss_qop_t qop_req,
const gss_buffer_t input_message_buffer,
int *conf_state,
gss_buffer_t output_message_buffer)

说明

gss_wrap() 例行程序可附加加密的消息完整性代码 (MIC)，并可对 *input_message* 进行加密。 *output_message* 中包含了 MIC 和消息。

尽管 *qop_req* 参数允许在多种保护力度中进行选择，但如果指定一种不支持的保护力度， **gss_wrap()** 例行程序就会返回 **GSS_S_FAILURE** 状态。

输入参数

context_handle 指定发送消息的环境。

conf_req_flag 指定所请求的保密及完整性服务的级别，如下所示：

Non-zero 同时请求了保密及完整性服务。

Zero 仅请求了完整性服务。

qop_req 指定加密算法或保护力度。通过将 *qop_req* 设置为 **GSS_C_QOP_DEFAULT** ，有时可以请求面向特定机制的缺省值。

input_message_buffer 指定要保护的消息。

输出参数

conf_state 返回所请求的保密及完整性服务的级别，如下所示：

Non-zero 已应用保密、数据来源、验证及完整性服务。

Zero 仅应用了完整性及数据来源服务。

output_message_buffer 返回用于接收受保护的消息的缓冲区。

minor_status 通过安全机制返回状态代码。

状态代码

可以返回下列状态代码:

GSS_S_COMPLETE 成功完成例行程序。

GSS_S_CONTEXT_EXPIRED
环境已过期。

GSS_S_CREDENTIALS_EXPIRED
可以识别环境，但关联的凭据已过期。

GSS_S_FAILURE 该例行程序失败。有关详细信息，请检查 *minor_status* 参数。

GSS_S_NO_CONTEXT *context_handle* 参数中所标识的环境无效。

GSS_S_BAD_QOP 指定的 QOP 不受机制支持。

作者

gss_wrap() 由 Sun Microsystems, Inc. 开发。

另请参阅

gss_unwrap(3)、 gss_wrap_size_limit(3)。

DCE-GSSAPI 的联机帮助页是随 DCE-CoreTools 产品一起提供的。要查看这些联机帮助页，请将 **/opt/dce/share/man** 添加至 **MANPATH** 。

名称

gss_wrap_size_limit() - 确定环境上 gss_wrap 的标记大小限制

概要

#include <gssapi.h>

OM_uint32 gss_wrap_size_limit (
OM_uint32 *minor_status,
const gss_ctx_id_t context_handle,
int conf_req_flag,
gss_qop_t qop_req,
OM_uint32 req_output_size,
OM_uint32 max_input_size)

说明

gss_wrap_size_limit() 例行程序允许应用程序确定消息的最大大小：如果向 **gss_wrap()** 提供相同的 *conf_req_flag* 和 *qop_req* 参数，将使输出标记所含的字节数不超过 *req_output_size* 。

此调用设计用于通过对消息最大大小有限制的协议进行通信的应用程序。它将使应用程序在应用保护前对消息进行分段。

输入参数

context_handle 指定发送该消息的环境。

conf_req_flag 指定所请求的保密及完整性服务的级别，如下所示：

Non-zero 同时请求了保密及完整性服务。

Zero 仅请求了完整性服务。

qop_req 指定加密算法或保护力度。通过将 *qop_req* 设置为 **GSS_C_QOP_DEFAULT** ，有时可以请求面向特定机制的缺省值。

req_output_size 被 **gss_wrap()** 省略的标记所需的最大大小。

输出参数

max_input_size 可提供给 **gss_wrap()** 以确保省略的标记不会大于 *req_output_size* 字节的最大输入消息大小。

minor_status 通过安全机制返回状态代码。

状态代码

可以返回下列状态代码：

GSS_S_COMPLETE 成功完成例行程序。

GSS_S_NO_CONTEXT *context_handle* 参数中所标识的环境无效。

GSS_S_CONTEXT_EXPIRED
环境已过期。

GSS_S_BAD_QOP 指定的 QOP 不受机制支持。

作者

gss_wrap_size_limit() 由 Sun Microsystems, Inc. 开发。

另请参阅

gss_wrap(3)。

DCE-GSSAPI 的联机帮助页是随 DCE-CoreTools 产品一起提供的。要查看这些联机帮助页，请将 **/opt/dce/share/man** 添加至 **MANPATH** 。

名称

halfdelay — 控制输入字符延迟模式

概要

#include <curses.h>

int halfdelay(int *tenths***);**

说明

halfdelay() 函数将当前窗口的输入模式设置为半延迟模式，并将 *tenths* 指定为十分之一秒，作为半延迟间隔。*tenths* 参数的范围为 1 到 255（包括 255）。

返回值

成功完成后， **halfdelay()** 返回 OK。否则返回 ERR。

错误

没有定义任何错误。

实际应用信息

应用程序可以调用 **nocbreak()** 来保持半延迟模式。

另请参阅

cbreak(3X)、 curses_intro(3X)，参阅输入处理、<curses.h>、《X/Open System Interface Definitions, Issue 4, Version 2》规范的第 9 章 General Terminal Interface 。

历史变更记录

在 X/Open Curses 第 4 期中首次发布。

名称

has_ic、has_il - 查询具有终端插入和删除功能的函数

概要

#include <curses.h>

bool has_ic(void);

bool has_il(void);

说明

has_ic() 函数可指示终端是否具有插入字符和删除字符的功能。

has_il() 函数可指示终端是否具有插入行和删除行的功能，或者是否能用滚动区域模拟这些功能。

返回值

如果终端具有插入字符和删除字符的功能， **has_ic()** 函数就会返回 TRUE。否则，它将返回 FALSE。

如果终端具有插入行和删除行的功能， **has_il()** 函数就会返回 TRUE。否则，它将返回 FALSE。

错误

未定义错误。

实际应用信息

has_il() 函数可用于确定是否适合用 **scrollok()**打开物理滚动功能。

另请参阅

<curses.h>。

历史变更记录

第一版发行在 X/Open Curses 第 2 期中。

X/Open Curses 第 4 期

has_il() 函数已与此条目合并。在以前各期中，它出现在自己的条目中。

此条目已被重新编写，用于进行明确说明。 **has_ic()** 和 **has_il()** 函数的参数列表已明确声明为 **void**。

名称

hg: hg_busywait()、 hg_context_switch_involuntary()、 hg_context_switch_tries()、 hg_context_switch_voluntary()、 hg_gethrcycles() 、 hg_gethrtime() 、 hg_getspu() 、 hg_nano_to_cycle_ratio() 、 hg_public_init() 、 hg_public_is_onRunQ() 、 hg_public_is_reporting() 、 hg_public_is_running() 、 hg_public_nMailboxes() 、 hg_public_nMailboxesInUse()、 hg_public_remove()、 hg_setcrit() - 用于在用户和内核空间之间以轻量级方式传输数据的 Mercury 库接口

概要

#include<sys/mercury.h>

cc *flags filename* **-lhg**

Mercury 公共接口

uint64_t hg_public_init(void);

uint64_t hg_public_remove(void);

uint64_t hg_public_is_reporting(uint64_t hg_public_handle);

uint64_t hg_public_is_running(uint64_t hg_public_handle);

uint64_t hg_public_is_onRunQ(uint64_t hg_public_handle);

uint64_t hg_public_nMailboxes(void);

uint64_t hg_public_nMailboxesInUse(void);

Mercury 专用接口

uint64_t hg_gethrcycles(void);

uint64_t hg_gethrtime(void);

double hg_nano_to_cycle_ratio(void);

uint64_t hg_busywait(double seconds);

spu_t hg_getspu(void);

uint64_t hg_context_switch_tries(void);

uint64_t hg_context_switch_involuntary(void);

uint64_t hg_context_switch_voluntary(void);

uint64_t hg_setcrit(unsigned long on_or_off, int willing_to_block);

说明

“**Mercury** 库接口” (HG) 可以在用户程序和内核之间提供高性能接口，这样就能高速来回传输关键的信息片段。

由于执行系统调用时存在系统开销，用户空间与内核之间的通信相对较慢。 Mercury API 有助于避免大部分或全

部系统开销。

交换的信息可划分为两大类：

- 公共数据：

  与特定内核线程有关的信息，很可能与其他线程相关。这些数据是通过 “**Mercury** 公共接口”函数提供的。接收到的这些信息是所查询的线程的当前运行状态。

- 专用数据：

  可能与单独的同一线程相关的信息。通过 “**Mercury** 专用接口”，内核可以通知线程通常只对该线程有用或相关的事件数。

只有在线程注册后，其他任何线程才可以查看该注册线程的公共信息。要查看其他线程的信息，无需进行注册。

如果线程注册其自身，则会为其提供一个可以传送到其他线程的 *handle* 。 Mercury 库可以提供 API，用于查询基础线程的运行状态的 *handle* 。因此，线程可以由分配到该线程的 *handle* 识别。

任何线程的运行状态信息均存储在公用的用户映射 *mailbox* 中。只要可能，内核就会在 *mailbox* 中持续发布和更新调用内核线程的运行状态。请注意，来自提供线程运行状态信息的任何 API 的所有信息都将被视为有关状态的提示，且不隐含任何保证，尽管系统已尽力保持这些信息的准确性。

如果使用 MxN 线程，则不建议使用 Mercury 库公共接口。这是因为该接口显示的状态始终是基础内核线程的状态。因此，范围为从内核线程移至内核线程的 **PTHREAD_FORCE_SCOPE_PROCESS** 的线程，将包含报告的混淆的信息。如果使用 Mercury 公共接口，请始终对所有线程使用 **PTHREAD_FORCE_SCOPE_SYSTEM** 。在可以使用 Mercury 的系统上，线程的缺省范围为 **PTHREAD_FORCE_SCOPE_SYSTEM** 。因此，用户可以在这些系统上使用 Mercury，而不必更改范围。有关详细信息，请参阅 *pthread_attr_setscope*(3T) 。

“环境切换”是指出于某种原因，线程停止在 CPU 上运行，然后又重新开始运行的行为；这一组操作对代表一次环境切换（“切换出”，然后“切换入”）。“自发”环境切换是由线程执行的某个操作导致的，该操作要求该线程停止运行。例如，线程调用 **sched_yield()** 或线程针对页错误进行阻塞。“非自发”环境切换是指操作系统出于策略方面的原因，对线程强加的环境切换。例如，操作系统切换出一个线程，以运行优先级更高的线程。

libhg 公共接口

下面是 **libhg** 提供的公共接口：

hg_public_init(void);

hg_public_init() 可以分配和初始化 *mailbox* ，其中，内核可以更新调用内核线程的运行状态。更新可立即开始。 *mailbox* 将映射到系统中每个线程的地址空间，这样，指定了该 *mailbox* 的地址的任何线程将可以随时检查线程是否正在运行。

hg_public_remove(void);

hg_public_remove() 可以终止更新运行状态信息，以及取消分配 *mailbox* 。删除的邮箱将保持已映射状态，以便继续检查这些邮箱的用户程序不会出错，但是，邮箱内容将标记为 **HG_PUBLIC_INVALID** （内核重用 *mailbox* 之前）。合理编写的应用程序在删除 *mailbox* 后，将会通知它自己的线程。

hg_public_is_running(uint64_t hg_public_handle);

hg_public_is_running() 可检查提供的 *mailbox* ，并返回一个布尔指示符，指示线程是否正在运行。如果线程已被切换出、被终止或者使用 **hg_public_remove()** 删除了自身，则它可能不会运行。

hg_public_is_reporting (uint64_t hg_public_handle);

hg_public_is_reporting() 可检查提供的 *mailbox* ，并返回一个布尔指示符，指示线程是否正在向 *mailbox* 报告。

hg_public_is_onRunQ (uint64_t hg_public_handle);

hg_public_is_onRunQ() 可检查提供的 *mailbox* ，并返回一个布尔指示符，指示线程当前是否已准备就绪，但尚未运行。

hg_public_nMailboxes (void);

hg_public_nMailboxes() 可返回内核支持的最大附件数。

hg_public_nMailboxesInUse (void);

hg_public_nMailboxesInUse() 可返回在系统级服务的当前附件数。

libhg 专用接口

下面是 **libhg** 提供的专用接口：

hg_gethrcycles (void);

hg_gethrcycles() 可返回自系统引导后的计算机工作周期。该次数不受 **settimeofday()** 和 **adjtime()** 调用（及其类似调用）的影响，并且，无论此接口从哪个处理器调用，都可以使用。

如果从相同的处理器调用，或者由相同的线程（即使该线程在处理器之间移动）调用，则该调用将始终以单调递增顺序返回次数。如 **hg_gethrtime()** 中的详细介绍，在某些情况下，通信线程有时会发现其次数稍有失序。

hg_gethrtime (void);

hg_gethrtime() 等同于 **hg_gethrcycles()** ，但是前者将返回自引导以来的时间（以纳秒为单位），而不返回计算机工作周期。

通常，该调用也可以按单调递增顺序返回次数，但存在一种例外情况。每个处理器运行时使用的准确时钟速率，会围绕中央额定值前后有少量的偏移。如果两个线程企图通过极高速渠道（类似于共享内存）前后抵消时间值，那么，在时间戳失序时，将可以观察到此偏移。有时会发生数百个循环的顺序倒置。但是需要高速交换和高速时间接口。 **gethrtime()** 之类的慢速接口难以观察到这种时钟抖动。 **gettimeofday()** 则根本不存在这种问题，原因是调用处理器始终将请求与处理器零同步。这是 **gettimeofday()** 比 **gethrtime()** 和 **hg_gethrcycles()** 要慢得多的原因之一。

然而，坚持以更普通的方式使用时间（例如测量间隔和生成间隙更宽的时间戳）的线程应该不会存在任何不便。

hg_nano_to_cycle_ratio (void);

hg_nano_to_cycle_ratio() 返回该特定处理器的纳秒数（或计算机工作周期）。

hg_busywait (double seconds);

hg_busywait() 将在用户模式下旋转请求的秒数。实际经过的时间将大于或等于指定的秒数。

hg_getspu (void);

hg_getspu() 返回其中运行了线程的 spu 的数目。请记住，在进行调用与使用信息的时间范围内，线程可能会按意料中的方式切换到另一个处理器。

hg_context_switch_tries (void);

hg_context_switch_tries() 返回内核线程切换出（或至少是尝试切换出）的总次数。这包括自发和非自发切换出操作。

hg_context_switch_involuntary (void);

hg_context_switch_involuntary() 返回内核线程以非自发方式切换出的次数（即该线程被另一线程抢占的次数）。

hg_context_switch_voluntary (void);

hg_context_switch_voluntary() 返回内核线程以自发方式切换出（例如获取页面错误）的次数。自发切换出的次数是根据总切换出次数与非自发切换次数之差计算得出的。

hg_setcrit (unsigned long on_or_off, int willing_to_block);

hg_setcrit() 通知内核线程何时进入或离开关键代码区域。进入关键区域时， *on_or_off* 应该为非零；离开关键区域时，应该为 0。

尽管任何非零值都可以配合 *on_or_off* ，但是，传入锁地址或其他某些关键信息片段（例如调用例行程序的地址）将十分有用。

内核使用这些信息帮助调度线程 - 如果设置了此关键区域，那么，只要有可能，内核就会避免将线程切换到不可运行状态。但是，在做了关键标记的区域中花费了太多时间的线程最终将被切换出，原因是内核必须完全平衡系统中不同线程的需求。

willing_to_block 参数表明如必要设置此关键区域，则线程将进行阻塞，以设置此关键区域。在进入关键区域之前希望进行阻塞的线程不太可能在关键区域中以非自发方式阻塞。提供的指导是在进入最远的关键区域时，将 *willing_to_block* 设置为 1（或其他某个非零值），在内部的关键区域中将它设置为零。

hg_setcrit() 不存在任何嵌套表示形式，因此，有关在嵌套的关键区域中调用 **hg_setcrit()** 的决策由调用线程负责。

返回值

如果调用成功， **hg_public_init()** 将返回 *mailbox* 的地址，否则将返回 NULL。

如果成功， **hg_public_remove()** 将返回值 1。

如果未报告， **hg_public_is_reporting()** 将返回零；如果报告，则返回非零。

如果未运行， **hg_public_is_running()** 将返回零；如果运行，则返回非零。

如果不在运行队列中， **hg_public_is_onRunQ()** 将返回 0；如果在运行队列中，则返回非零。

hg_public_nMailboxes() 返回附件可用的邮箱总数。

hg_public_nMailboxesInUse() 返回调用时附件可用的邮箱数。

hg_gethrcycles() 返回自计算机引导后的计算机工作周期。

hg_gethrtime() 返回自计算机引导后的纳秒数。

hg_nano_to_cycle_ratio() 返回该特定处理器的纳秒数（或计算机工作周期）。

成功完成时， **hg_busywait()** 将返回 0。

hg_getspu() 返回线程在其上运行的处理器的数目。

hg_context_switch_tries() 返回内核线程切换出（或至少是尝试切换出）的总次数。

hg_context_switch_involuntary() 返回内核线程以非自发方式切换出的次数。

hg_context_switch_voluntary() 返回内核线程以自发方式切换出的次数。

hg_setcrit() 返回线程放弃处理器的次数。

错误

libhg API 返回的唯一错误为 **HG_ERROR** 。如果下列任一调用失败，则会返回此错误：

hg_public_init()、 **hg_public_nMailboxes()**、 **hg_public_nMailboxesInUse()** 和 **hg_busywait()** 。

举例

下面是应用程序如何使用 Mercury API 来实现性能的实例。

由于线程可以在任何时间点删除邮箱，因此，可以使用 **hg_public_is_reporting()** 来帮助查找邮箱。如果 *mailbox* 仍在使用中，则可以获取与其关联的线程的运行状态。有时这种功能非常重要，原因是如果某个线程分离了 *mailbox* ，则侦测线程可能获得与被侦测线程有关的错误消息，这是因为 *mailbox* 仍保持已映射状态。另一种情况可能是，即使 *mailbox* 在使用中，也可能会被前面的线程放弃，并重新分配到全新的线程。

内核在关键区域中运行时抢先于线程的可能性比不在关键区域中运行时要低。因此，保存锁的代码区域是可以标记为关键区域的理想位置，原因是如果保存关键资源的线程停止运行，而其他线程正在等待获取这些资源，则应用程序吞吐量可能会大幅下降。

由于 **hg_setcrit()** 不进入内核，因此，它的速度很快，并且不会显著增加关键路径的长度。如果将 *willing_to_block* 设置为 1（而不是 0），则必要时 **hg_setcrit()** 将放弃处理器，然后开始实现关键区域，以减少在关键区域中被抢先的可能性。

下面是一种建议的序列：

hg_setcrit(*lock_address***, WILLING_TO_BLOCK);**
acquire lock
critical region
release lock

hg_setcrit(CRIT_OFF, UNWILLING_TO_BLOCK);

或者:

hg_setcrit(*lock_address***, UNWILLING_TO_BLOCK);**
acquire lock
critical region
release lock
hg_setcrit(CRIT_OFF, WILLING_TO_BLOCK);

或 ...

hg_setcrit(*lock_address***, WILLING_TO_BLOCK);**
acquire lock
critical region
release lock
hg_setcrit(CRIT_OFF, WILLING_TO_BLOCK);

永远没有设置 **WILLING_TO_BLOCK** 的绝对必要，但是，进行此设置可以减少在实现关键区域期间被切换出的可能性。请记住， **WILLING_TO_BLOCK** 表示您希望在 **hg_setcrit()** 调用期间（不是在实现关键区域期间）进行阻塞。

注释

主动退回的所有信息应视为 提示。根据这些信息的本质，以及调度用户进程和线程的性质，从理论上讲，其中许多信息在传递到用户后可能会立即过时，或者在传递后的任何时间过时。用户代码必须在其设计中考虑到此因素。

libpthread 库将与 Mercury 库链接，以针对在任何时间点进行的线程调度做出决策。因此，同时使用 **libpthread** 和 **libhg** 的应用程序需要注意 **libpthread** 已使用 **libhg** 进行调度决策。但是，如果应用程序觉得有必要，仍然可以使用 Mercury API。

警告

请记住，内核发布这些线程信息后，这些信息可能会在任何时间过时。请注意您对它的依赖程度。编写代码时，完全由用户代码负责考虑这个问题。

作者

Mercury API 由 HP 开发。

另请参阅

pthread(3T)、pthread_attr_setscope(3T)。

名称

hline、mvhline、mvvline、mvwhline、mvwvline、vline、whline、wvline — 利用单字节字符及呈现方式绘制直线

概要

#include <curses.h>

int hline(chtype *ch*, **int** *n*);

int mvhline(int *y*, **int** *x*, **chtype** *ch*, **int** *n*);

int mvvline(int *y*, **int** *x*, **chtype** *ch*, **int** *n*);

int mvwhline(WINDOW ***_win_, **int *y*, **int** *x*, **chtype** *ch*, **int** *n*);

int mvwvline(WINDOW ***_win_, **int *y*, **int** *x*, **chtype** *ch*, **int** *n*);

int vline(chtype *ch*, **int** *n*);

int whline(WINDOW ***_win_, **chtype *ch*, **int** *n*);

int wvline(WINDOW ***_win_, **chtype *ch*, **int** *n*);

说明

这些函数会从当前或指定的位置开始，利用 *ch* 在当前窗口或指定的窗口中绘制直线。直线最多有 *n* 个位置长，或者符合窗口的大小。

这些函数不会使光标位置向前移动。这些函数不执行特殊字符处理操作。这些函数不执行换行操作。

hline() 、 **mvhline()** 、 **mvwhline()** 和 **whline()** 函数将向同一行的最后一列的方向绘制直线。

vline() 、 **mvvline()** 、 **mvwvline()** 和 **wvline()** 函数将向窗口的最后一行的方向绘制直线。

返回值

成功完成后，这些函数将返回 OK。否则，它们将返回 ERR。

错误

未定义错误。

实际应用信息

只能保证这些函数在特定的字符集中可靠地运行，即在这些字符集中，每个字符都与一个单字节字符匹配，其属性只能通过带有 A_ prefix 的常量来表示。

另请参阅

border(3X)、 box(3X)、 hline_set(3X)、 <curses.h>。

历史变更记录

在 X/Open Curses 第 4 期中首次发布。

名称

hline_set、 mvhline_set、 mvvline_set、 mvwhline_set、 mvwvline_set、 vline_set、 whline_set 和 wvline_set — 利用组合字符及呈现方式绘制直线

概要

#include <curses.h>

int hline_set(const cchar_t ****wch*, int** *n***);**

int mvhline_set(int *y***, int** *x***, const cchar_t ****wch***, int** *n***);**

int mvvline_set(int *y***, int** *x***, const cchar_t ****wch***, int** *n***);**

int mvwhline_set(WINDOW ****win*, int** *y***, int** *x***, const cchar_t ****wch***, int** *n***);**

int mvwvline_set(WINDOW ****win*, int** *y***, int** *x***, const cchar_t ****wch***, int** *n***);**

int vline_set(const cchar_t ****wch*, int** *n***);**

int whline_set(WINDOW ****win*, const cchar_t ****wch***, int** *n***);**

int wvline_set(WINDOW ****win*, const cchar_t ****wch***, int** *n***);**

说明

这些函数使用 *ch* 在当前或指定窗口（从当前或指定位置开始）中绘制直线。所绘制的直线最多有 *n* 个位置长，或者刚好适合窗口。

这些函数不会使光标位置向前移动。这些函数不执行特殊字符处理操作。这些函数也不执行换行操作。

hline_set() 、 **mvhline_set()** 、 **mvwhline_set()** 和 **whline_set()** 函数绘制从同一行的第一列到最后一列的直线。

vline_set() 、 **mvvline_set()** 、 **mvwvline_set()** 和 **wvline_set()** 函数绘制从窗口的第一行到最后一行的直线。

返回值

成功完成后，这些函数返回 OK。否则，返回 ERR。

错误

没有定义任何错误。

另请参阅

border_set(3X)、 <curses.h>。

历史变更记录

在 X/Open Curses 第 4 期中首次发布。

名称

hosts_access()、hosts_ctl()、request_init()、request_set() - 访问控制库

概要

```
#include <tcpd.h>

extern int allow_severity;
extern int deny_severity;
extern int rfc931_timeout;

struct request_info *request_init(request, key, value, ..., 0)
struct request_info *request;

struct request_info *request_set(request, key, value, ..., 0)
struct request_info *request;

int hosts_access(request)
struct request_info *request;

int hosts_ctl(daemon, client_name, client_addr, client_user)
char *daemon;
char *client_name;
char *client_addr;
char *client_user;
```

说明

此处介绍的例行程序是 **libwrap.a** 库的一部分。这些例行程序可以使用规则生效时执行的可选 shell 命令，来实现基于规则的访问控制语言。

request_init() 根据有关客户端请求的信息初始化结构。 **request_set()** 更新已初始化的请求结构。这两个函数都使用关键字-值对的可变长度列表，并返回各自的第一个参数。参数列表以零关键字值结尾。将复制所有值为字符串的参数。所需的关键字（及对应的值类型）为：

RQ_FILE (int)
与请求关联的文件描述符。

RQ_CLIENT_NAME (char *)
客户端主机名。

RQ_CLIENT_ADDR (char *)
客户端网络地址的可输出表示形式。

RQ_CLIENT_SIN (struct sockaddr_in *)
客户端网络地址和端口的内部表示形式。不复制结构的内容。

RQ_SERVER_NAME (char *)
与服务器端点地址关联的主机名。

RQ_SERVER_ADDR (char *)
服务器端点地址的可输出表示形式。

RQ_SERVER_SIN (struct sockaddr_in *)
服务器端点地址和端口的内部表示形式。不复制结构的内容。

RQ_DAEMON (char *)
在服务器主机上运行的守护程序进程的名称。

RQ_USER (char *)
客户端代表其发出请求的用户的名称。

hosts_access() 可查询 *hosts_access*(5) 手册页中介绍的访问控制表。如果内部端点信息可用，则将请求结构用作缓存，按需要查找主机名和客户端用户名。如果应拒绝访问，则 **hosts_access()** 将返回零。

hosts_ctl() 是围绕 **request_init()** 和 **hosts_access()** 例行程序的包装，可能带有更方便的接口（尽管它不传递足够的信息来支持自动客户端用户名查找）。客户端主机地址、客户端主机名和用户名参数应包含有效的数据或 **STRING_UNKNOWN** 。如果应拒绝访问，则 **hosts_ctl()** 将返回零。

allow_severity 和 *deny_severity* 变量确定如何记录接受和拒绝的请求。这些变量必须由调用方提供，并可以根据访问控制表中的规则进行修改。

rfc931_timeout 变量确定客户端的用户名查找的超时值。该变量必须设置为正值，如果值为 0，则将禁止用户名查找。

返回值

request_init() 和 **request_set()** 函数返回指向 **request_info** 结构的指针。使用分别传递给函数 **request_init()** 和 **request_set()** 的值，来初始化和更新已返回结构的成员。

如果授予了对所请求服务的访问权限，则 **hosts_access()** 和 **hosts_ctl()** 函数将返回 **1** ；如果拒绝了对所请求服务的访问权限，则返回 **0** 。

诊断信息

syslog 守护程序 **syslogd** 将以 **info**、notice、 **warning** 和 **err** 级别报告问题。

警告

hosts_access() 使用 **strtok()** 库函数。这可能会影响其他依赖于 **strtok()** 的代码。

作者

这些例行程序的开发方为

Wietse Venema (wietse@wzv.win.tue.nl)
Department of Mathematics and Computing Science
Eindhoven University of Technology

Den Dolech 2, P.O. Box 513,
5600 MB Eindhoven, The Netherlands

文件

/etc/hosts.allow 访问控制表。

/etc/hosts.deny 访问控制表。

另请参阅

hosts_access(5) 访问控制表的格式。

hosts_options(5) 基础语言的可选扩展。

名称

hppac : HPPACADDD() 、 HPPACCMPD() 、 HPPACCVAD() 、 HPPACCVBD() 、 HPPACCVDA() 、 HPPACCVDB()、 HPPACDIVD()、 HPPACLONGDIVD()、 HPPACMPYD()、 HPPACNSLD()、 HPPACSLD()、 HPPACSRD()、 HPPACSUBD() - HP 3000 模式压缩十进制库

概要

```
#include <hppac.h>

void HPPACADDD(
    unsigned char *operand2,
    int op2digs,
    unsigned char *operand1,
    int op1digs,
    enum HPPAC_CC *compcode,
    int *pacstatus
);

void HPPACCMPD(
    unsigned char *operand1,
    int op1digs,
    unsigned char *operand2,
    int op2digs,
    enum HPPAC_CC *compcode,
    int *pacstatus
);

void HPPACCVAD(
    unsigned char *target,
    int targetdigs,
    unsigned char *source,
    int sourcedigs,
    enum HPPAC_CC *compcode,
    int *pacstatus
);

void HPPACCVBD(
    unsigned char *target,
    int targetdigs,
    unsigned short *source,
    int sourcewords,
    enum HPPAC_CC *compcode,
    int *pacstatus
```

```
);

void HPPACCVDA(
    unsigned char *target,
    int targetdigs,
    unsigned char *source,
    int sign_control,
    enum HPPAC_CC *compcode,
    int *pacstatus
);

void HPPACCVDB(
    unsigned short *target,
    unsigned char *source,
    int sourcedigs,
    enum HPPAC_CC *compcode,
    int *pacstatus
);

void HPPACDIVD(
    unsigned char *operand2,
    int op2digs,
    unsigned char *operand1,
    int op1digs,
    enum HPPAC_CC *compcode,
    int *pacstatus
);

void HPPACLONGDIVD(
    unsigned char *operand2,
    int op2digs,
    unsigned char *operand1,
    int op1digs,
    enum HPPAC_CC *compcode,
    int *pacstatus
);

void HPPACMPYD(
    unsigned char *operand2,
    int op2digs,
    unsigned char *operand1,
    int op1digs,
```

```
    enum HPPAC_CC *compcode,
    int *pacstatus
);

void HPPACNSLD(
    unsigned char *operand2,
    int op2digs,
    unsigned char *operand1,
    int op1digs,
    int *shift_amt,
    enum HPPAC_CC *compcode,
    int *pacstatus,
    int *carry
);

void HPPACSLD(
    unsigned char *operand2,
    int op2digs,
    unsigned char *operand1,
    int op1digs,
    int shift_amt,
    enum HPPAC_CC *compcode,
    int *pacstatus,
    int *carry
);

void HPPACSRD(
    unsigned char *operand2,
    int op2digs,
    unsigned char *operand1,
    int op1digs,
    int shift_amt,
    enum HPPAC_CC *compcode,
    int *pacstatus
);

void HPPACSUBD(
    unsigned char *operand2,
    int op2digs,
    unsigned char *operand1,
    int op1digs,
```

```
    enum HPPAC_CC *compcode,
    int *pacstatus
);
```

说明

这组调用调用库函数来模拟 3000 模式 (MPE V/E) 压缩十进制运算。这些函数位于库 **libcl** 中，当选项 **-lcl** 与 **cc** 或 *ld*(1) 一起使用时，将搜索该库。

HPPACADDD()	执行压缩十进制加法运算。
HPPACCMPD()	比较两个压缩十进制数。
HPPACCVAD()	将 ASCII 表示转换为压缩十进制。
HPPACCVBD()	将二进制表示转换为压缩十进制。
HPPACCVDA()	将压缩十进制数转换为 ASCII。
HPPACCVDB()	将压缩十进制数转换为二进制。
HPPACDIVD()	执行压缩十进制除法运算。
HPPACLONGDIVD()	执行压缩十进制除法运算（备用例行程序）。
HPPACMPYD()	执行压缩十进制乘法运算。
HPPACNSLD()	执行压缩十进制标准化左移位运算。
HPPACSLD()	执行压缩十进制左移位运算。
HPPACSRD()	执行压缩十进制右移位运算。
HPPACSUBD()	执行压缩十进制减法运算。

对于所有运算而言， *compcode* 参数所指向的变量中的返回值是下列 **enum HPPAC_CC** 类型的值之一：

HPPAC_CCG	结果 > 0 或 operand1 > operand2
HPPAC_CCL	结果 < 0 或 operand1 < operand2
HPPAC_CCE	结果 == 0 或 operand1 == operand2

对于所有运算而言， *pacstatus* 参数所指向的变量中的返回值是下列 **enum HPPAC_STATUS** 类型的值之一：它们的含义应该很明显：

HPPAC_NO_ERROR

HPPAC_DECIMAL_OVERFLOW

HPPAC_INVALID_ASCII_DIGIT

HPPAC_INVALID_PACKED_DECIMAL_DIGIT

HPPAC_INVALID_SOURCE_WORD_COUNT

HPPAC_INVALID_DECIMAL_OPERAND_LENGTH

HPPAC_DECIMAL_DIVIDE_BY_ZERO

作者

HPPAC 库由 HP 开发。

另请参阅

《Compiler Library/XL Reference Manual》

名称

hsearch( )、hcreate( )、hdestroy( ) - 管理哈希搜索表

概要

#include <search.h>

ENTRY *hsearch(ENTRY item, ACTION action);

int hcreate(size_t nel);

void hdestroy(void);

说明

hsearch() 是从 Knuth (6.4) 算法 D 归纳而来的哈希表搜索例行程序。此例行程序可返回指向哈希表的指针，以指示可以找到条目的位置。只复制指针，因此调用例行程序必须存储数据（“关键字”的值必须是唯一的）。 *item* 是 **ENTRY** 类型（在 **<search.h>** 头文件中定义）的结构，该结构包含两个指针： **item.key** 指向比较关键字，**item.data** 指向要与该关键字关联的其他任何数据（指向类型而非字符的指针应转换为指针字符）。 *action* 是枚举类型 **ACTION** 的成员，指示如果无法在表中找到条目，应如何处理该条目。 **ENTER** 指示应该将项目插入表中相应的点。 **FIND** 指示不应创建任何条目。返回 NULL 指针表示解析不成功。

hcreate() 可为表分配足够的空间，使用 **hsearch()** 之前，必须先调用该函数。 *nel* 是表可包含的预计最大条目数。为了得到某些数学上有利的条件，可通过算法向上调整此数目。

hdestroy() 可损坏搜索表，随后可以对 **hcreate()** 执行另一次调用。

举例

以下示例将读入后接两个数字的字符串，然后将这些字符串存储在哈希表中，同时忽略重复项。该示例再读入字符串，并在哈希表中查找匹配的条目，然后输出该条目。

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

struct info {          /* this is the info stored in the table */
    int age, room;     /* other than the key.*/
};
#define NUM_EMPL    5000    /* # of elements in search table */

main( )
{
    /* space to store strings */
    char string_space[NUM_EMPL*20];

    /* space to store employee info */
    struct info info_space[NUM_EMPL];
```

```
/* next avail space in string_space */
char *str_ptr = string_space;

/* next avail space in info_space */
struct info *info_ptr = info_space;
ENTRY item, *found_item, *hsearch( );
/* name to look for in table */

char name_to_find[30];
int i = 0;

/* create table */
(void) hcreate(NUM_EMPL);
while (scanf("%s%d%d", str_ptr, &info_ptr->age,
      &info_ptr->room) != EOF && i++ < NUM_EMPL) {

    /* put info in structure, and structure in item */
    item.key = str_ptr;
    item.data = (char *)info_ptr;
    str_ptr += strlen(str_ptr) + 1;
    info_ptr++;

    /* put item into table */
    (void) hsearch(item, ENTER);
}

/* access table */
item.key = name_to_find;
while (scanf("%s", item.key) != EOF) {
   if ((found_item = hsearch(item, FIND)) != NULL) {

       /* if item is in the table */
       (void)printf("found %s, age = %d, room = %d\n",
           found_item->key,
           ((struct info *)found_item->data)->age,
           ((struct info *)found_item->data)->room);
       } else {
          (void)printf("no such employee %s\n",
              name_to_find);
```

```
                }
            }
        }
```

返回值

如果操作为 **FIND** ，并且无法找到项目，或操作为 **ENTER** ，且表已满，则 **hsearch()** 将返回 NULL 指针。

如果 **hcreate()** 无法为表分配足够的空间，则它将返回零。

警告

hsearch() 和 **hcreate()** 使用 **malloc()** 来分配空间（请参阅 *malloc*(3C) ）。

在任何给定的时间只能有一个哈希搜索表处于活动状态。

另请参阅

bsearch(3C)、lsearch(3C)、malloc(3C)、string(3C)、tsearch(3C)、 thread_safety(5)。

符合的标准

hsearch()：AES、SVID2、SVID3、XPG2、XPG3、XPG4

hcreate()：AES、SVID2、SVID3、XPG2、XPG3、XPG4

hdestroy()：AES、SVID2、SVID3、XPG2、XPG3、XPG4

名称

hypotf( )、hypotl( )、hypotw( )、hypotq( ) - 欧几里德距离函数

概要

#include <math.h>

double hypot(double x, double y);

仅适用于 **HP Integrity** 服务器

float hypotf(float x, float y);

long double hypotl(long double x, long double y);

extended hypotw(extended x, extended y);

quad hypotq(quad x, quad y);

说明

hypot() 将计算 *x* 和 *y* 的平方和的平方根，它没有不合理的上溢或下溢。

仅适用于 **Integrity** 服务器

hypotf() 是 **float** 版的 **hypot()** ；它采用 **float** 参数，并返回 **float** 结果。

hypotl() 是 **long double** 版的 **hypot()** ；它采用 **long double** 参数，并返回 **long double** 结果。

hypotw() 是 **extended** 版的 **hypot()** ；它采用 **extended** 参数，并返回 **extended** 结果。

hypotq() 等效于 HP-UX 系统上的 **hypotl()** 。

用法

要使用上面的任何函数，请使用缺省的 **-Ae** 选项或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。

（对于 Integrity 服务器）要使用 **hypotw()** 或 **hypotq()** ，也可以使用 **-fpwidetypes** 选项进行编译。

确保程序包含 **<math.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

返回值

hypot(*x*,*y***)** 、 **hypot(***y*,*x***)** 和 **hypot(***x*,-*y***)** 是等效的。

hypot(*x*,±0**)** 等效于 **fabs(***x***)** 。

即使 *y* 为 NaN， **hypot(**±Inf,*y***)** 也将返回 +Inf。

如果两个参数都为 NaN， **hypot()** 就会返回 NaN。

hypot() 将返回无穷大（用于代替量值太大的值）并引发溢出及不精确异常。

当结果太小（本质上异常或为零），因此会丧失准确性时， **hypot()** 将引发下溢和不精确异常；如果结果只是太小，即可能引发这些异常。

当舍入后的结果不等于算术结果时， **hypot()** 将引发不精确异常。

错误

没有定义任何错误。

另请参阅

cabs(3M)、 fabs(3M)、 sqrt(3M)、 math(5)。

符合的标准

hypot() ： SVID3、XPG4.2、ISO/IEC C99 （包括附件 F“IEC 60559 floating-point arithmetic”）

hypotf() 、 **hypotl()** ： ISO/IEC C99（包括附件 F“IEC 60559 floating-point arithmetic”）

名称

iconv()、iconv_open()、iconv_close() - 代码集转换例行程序

概要

```
#include <iconv.h>

iconv_t iconv_open(const char *tocode, const char *fromcode);

size_t iconv(
    iconv_t         cd,
    const char      **inbuf,
    size_t          *inbytesleft,
    char            **outbuf,
    size_t          *outbytesleft
);

int iconv_close(iconv_t cd);
```

备注

这些接口符合 XPG4 标准，应使用它们来取代 9.0 **iconv** 接口，例如 **iconvopen()** 、 **iconvclose()** 、 **iconvsize()** 、 **iconvlock()** 、 **ICONV()** 、 **ICONV1()** 和 **ICONV2()** 。

要了解转换过程，请参考标题为《HP-UX 11.0 - 11i 国际化特性白皮书》的白皮书。此白皮书介绍了 **iconv** 如何使用表和方法来执行转换。此外还说明了如何自定义自己的转换。此白皮书位于 **http://docs.hp.com/zh_cn/index.html** 。

说明

iconv_open() 例行程序使用两个配置文件，即位于 **/usr/lib/nls/iconv** 目录中的 **system.config.iconv** 和 **config.iconv**。 **system.config.iconv** 和 **config.iconv** 文件中的条目是 **iconv()** 支持的转换的集合（代码集名称）。最前面两列对应于 *fromcode* 和 *tocode* 名称。可以直接使用这些名称，或者使用其对应的别名，作为 **iconv_open()** 的参数。

iconv_open() 返回一个转换描述符，描述将 *fromcode* 参数指向的字符串所指定的代码集转换为 *tocode* 参数指定的代码集。

在某个过程中，转换描述符将保持有效，直到该进程将其关闭。

配置文件 **/usr/lib/nls/iconv/system.config.iconv** 或 **/usr/lib/nls/iconv/config.iconv** 中，必须具有 *fromcode* 和 *tocode* 参数的对应条目。

系统文件 **/usr/lib/nls/iconv/system.config.iconv** 不能修改，且包含操作系统支持的代码集名称。

用户可以自定义修改 **/usr/lib/nls/iconv/config.iconv** 。

iconv_open() 会依次在 **/usr/lib/nls/iconv/system.config.iconv** 和 **/usr/lib/nls/iconv/config.iconv** 中搜索代码集名称，以检查是否支持请求的转换。如果是，则 **iconv_open()** 会确

定将哪个表和（或）方法用于转换。

iconv() 将 *inbuf* 指定的数组包含的代码集中的字符序列，转换为 *outbuf* 指定的数组包含的另一代码集中对应的字符序列。这里的代码集是返回转换描述符 *cd* 的 **iconv_open()** 调用中指定的代码集。 *inbuf* 参数指向一个指向输入缓冲区中第一个字符的变量， *inbytesleft* 表示所转换的缓冲区中的剩余字节数。 *outbuf* 参数指向一个指向输出缓冲区中第一个可用字节的变量， *outbytesleft* 表示该缓冲区中剩余的可用字节数。

如果输入字节序列不能构成指定代码集中的有效字符，那么转换将在上一个成功转换的字符后面停止。如果输入缓冲区以不完整的字符或转换序列（请参阅“特殊应用信息”一节）结束，那么转换将在上一个成功转换的字符后面停止。如果输出缓冲区的大小不足以保存整个转换的输出，那么转换刚好在会导致输出缓冲区溢出的字符之前停止。 *inbuf* 指向的变量更新为指向跟在转换中成功使用的最后一个字节后面的字节。减小 *inbyesleft* 指向的值，以反映输入缓冲区中尚未转换的字节数。将 *outbuf* 指向的变量更新为指向跟在转换的输出数据最后一个字节后面的字节。减小 *outbytesleft* 指向的值，以反映输出缓冲区中仍然可用的字节数。

如果 **iconv()** 在输入缓冲区中遇到一个字符，而该字符是合法的，但是目标代码集中不存在与它相同的字符， **iconv()** 会将该字符映射为生成表时定义的，称为“活版字符”的预定义字符。（请参阅 *genxlt*(1) ）。

iconv_close() 取消分配转换描述符 *cd* ，以及 **iconv_open()** 分配的所有其他相关资源。

实际应用信息

可移植应用程序必须假定调用任何 **exec** 函数后，转换描述符将会无效。

特殊应用信息

在状态相关编码中，将根据输入“状态”解释字符。输入中出现特定的字节序列时，将发生状态切换。这些序列会改变后续字符的解释方式（也就是说，最初字符可能是单字节字符，发生状态切换后，可能将后续字符解释为双字节字符）。对于状态相关的编码， **iconv_open()** 后面的转换描述符处于代码集相关的初始切换状态，可以直接与 **iconv()** 一起使用。

对于状态相关的编码，通过调用其 *inbuf* 为空指针，或者其 *inbuf* 指向空指针的 **iconv()** ，可以将转换描述符 *cd* 置于其初始切换状态。如果以这种方法调用了 **iconv()** ，并且 *outbuf* 不是空指针，也不是指向空指针的指针，同时 *outbytesleft* 指向正值，则 **iconv()** 会将更改输出缓冲区的字节序列置于其初始切换状态。如果输出缓冲区的大小不足以保存整个重置序列， **iconv()** 将会失败，并将 **errno** 设置为 [E2BIG] 。如果后续调用时没有将 *inbuf* 设置为空指针，或者没有设置为指向空指针的指针，那么将从转换描述符的当前状态发生转换。

对于状态相关编码，将更新转换描述符，以反映最后一次成功转换的字节序列结尾处有效的切换状态。

返回值

iconv_open() 如果成功完成， **iconv_open()** 返回后续调用 **iconv()** 时使用的转换描述符。否则， **iconv_open()** 返回 **(iconv_t)–1** 并设置 **errno** 以指明错误。

iconv() **iconv()** 更新参数指向的变量，以反映转换的范围，并返回执行的不同转换的数目。如果转换了输入缓冲区中的整个字符串， *inbytesleft* 指向的值为零。如果发生错误， **iconv()** 将返回 (**size_t**)–1 ，并设置 **errno** 以指明错误。

iconv_close() 成功完成后， **iconv_close()** 将返回值 0；否则，它将返回 –1 并设置 **errno** 以指明错误。

错误

如果遇到以下任何情况， **iconv_open()** 将会失败：

[ENOMEM] 未提供足够的存储空间。

[EINVAL] 不支持 *fromcode* 和 *tocode* 指定的转换，或者无法正常读取或加载配置文件中指定的表或方法。如果配置文件自身有缺陷，也会发生此错误。

如果遇到以下任何情况， **iconv()** 将会失败：

[EILSEQ] 由于输入字符不属于输入代码集，或者如果转换表不包含与此输入字符对应的条目，并且没有为该特定表定义预定义字符，因此输入转换停止。

[E2BIG] 由于输出缓冲区中的空间不足，输入转换停止。

[EINVAL] 由于输入缓冲区结尾有不完整的字符或 shift 序列，输入转换停止。

[EBADF] *cd* 参数不是有效的开放转换描述符。

如果遇到以下任何情况， **iconv_close()** 将会失败：

[EBADF] 转换描述符无效。

举例

以下示例说明如何在可能的情况下，将 **iconv()** 接口用于转换。

```
#include <iconv.h>
#include <errno.h>

main()
{
    ...
    convert("roman8", "iso88591", fd);
    ...
}

int
convert(tocode, fromcode, Input)
char *tocode;                   /* tocode name */
char *fromcode                  /* fromcode name */
int Input;                      /* input file descriptor */
```

```
{
   extern void error();              /* local error message */

   iconv_t cd;                       /* conversion descriptor */
   unsigned char *table;             /* ptr to translation table */
   int bytesread;                    /* num bytes read into input buffer */
   unsigned char inbuf[BUFSIZ];              /* input buffer */
   unsigned char *inchar;            /* ptr to input character */
   size_t inbytesleft;               /* num bytes left in input buffer */
   unsigned char outbuf[BUFSIZ];             /* output buffer */
   unsigned char *outchar;           /* ptr to output character */
   size_t outbytesleft;              /* num bytes left in output buffer */
   size_t ret_val;                   /* number of conversions */

   /* Initiate conversion -- get conversion descriptor */
   if ((cd = iconv_open(tocode, fromcode)) == (iconv_t)-1) {
      error(FATAL, BAD_OPEN);
   }

   inbytesleft = 0;                  /* no. of bytes converted */
   /* translate the characters */
   for ( ;; ) {
            /*
             * if any bytes are leftover, they will be in the
             * beginning of the buffer on the next read().
             */

            inchar = inbuf;          /* points to input buffer */
            outchar = outbuf;        /* points to output buffer */
            outbytesleft = BUFSIZ;           /* no of bytes to be converted */
            if ((bytesread = read(Input, inbuf+inbytesleft,
                        (size_t)BUFSIZ-inbytesleft)) < 0) {
                  perror("prog");
                  return BAD;
            }
            if (!(inbytesleft += bytesread)) {
                  break;         /* end of conversions */
            }
```

```
        ret_val = iconv(cd, &inchar, &inbytesleft,
                    &outchar, &outbytesleft);

        if (write(1, outbuf, (size_t)BUFSIZ-outbytesleft) < 0) {
            perror("prog");
            return BAD;
        }

        /* iconv() returns the number of non-identical conversions
         * performed.If the entire string in the input buffer is
         * converted, the value pointed to by inbytesleft will be
         * zero.If the conversion stopped due to any reason, the
         * value pointed to by inbytesleft will be non-zero and
         * errno is set to indicate the condition.
         */
        if ((ret_val == -1) && (errno == EINVAL)) {
            /* Input conversion stopped due to an incomplete
             * character or shift sequence at the end of the
             * input buffer.
             */
            /* Copy data left, to the start of buffer */
            memcpy((char *)inbuf, (char *)inchar,
                (size_t)inbytesleft);
        } else if ((ret_val == -1) && (errno == EILSEQ)) {
            /* Input conversion stopped due to an input byte
             * that does not belong to the input codeset.
             */
            error(FATAL, BAD_CONVERSION);
        } else if ((ret_val == -1) && (errno == E2BIG)) {
            /* Input conversion stopped due to lack of space
             * in the output buffer. inbytesleft has the
             * number of bytes to be converted.
             */
            memcpy((char *)inbuf, (char *)inchar,
                (size_t)inbytesleft);
        }
        /* Go back and read from the input file. */
    }
```

```
    /* end conversion & get rid of the conversion table */
    if (iconv_close(cd) == BAD) {
        error(FATAL, BAD_CLOSE);
    }
    return GOOD;
}
```

警告

如果使用了 **iconv()** ，则在编译（或链接）PA-RISC 系统上的应用程序归档文件时，请注意 **iconv()** 具有 **libdld.sl** 的相关性，这需要更改编译/链接命令：

编译：

cc -Wl,-a,archive -Wl,-E -Wl,+n -l:libdld.sl -o *outfile source*

或使用 **CCOPTS** 和 **LDOPTS** 编译：

export CCOPTS="-Wl,-a,archive *options* **-Wl,-E -l:libdld.sl"**

export LDOPTS="*options* **-E +n -l:libdld.sl"**

cc -o *outfile source*

选项 **-Wl,-a,archive** 与位置相关，应在编译行的起始处应用。为获得将来发行版的最佳兼容性，应避免与其他共享库（为满足以上情况而使用的 libdld.sl 除外）一起使用归档文件 libc。

多字节字符存在一种极端状况 (corner-case)， **iconv()** 不能正常处理这种情况。如果所转换的文件中，最后一个字符是无效的多字节字符， **iconv()** 将返回 [EINVAL] ，而不是 [EILSEQ] 。应用程序可以通过检查是否到达 EOF，或者是否在转换最后一个缓冲区，来避免发生此状况。在这种情况下，应将 [EINVAL] 视为 [EILSEQ] 。

作者

iconv() 由 HP 开发。

文件

包含操作系统支持的代码集名称的	**/usr/lib/nls/iconv/system.config.iconv** 系统 **iconv** 配置文件。
/usr/lib/nls/iconv/config.iconv	包含其他代码集名称的用户可自定义的 **iconv** 配置文件。
	iconv_open() 会依次在 **/usr/lib/nls/iconv/system.config.iconv** 和 **/usr/lib/nls/iconv/config.iconv** 中搜索代码集名称，以检查是否支持请求的转换。如果是，则 **iconv_open()** 会确定将哪个表和（或）方法用于转换。
/usr/lib/nls/iconv/tables	包含用于转换的表的目录。
/usr/lib/nls/iconv/methods	包含用于转换的方法的目录。

另请参阅

genxlt(1)、 iconv(1)、 thread_safety(5)。

http://docs.hp.com/zh_cn/index.html 上提供了《HP-UX 11.0 - 11i 国际化特性白皮书》

符合的标准

iconv_open()： XPG4

iconv()： XPG4

iconv_close()： XPG4

名称

idcok — 启用或禁用硬件插入和删除字符功能

概要

#include <curses.h>

void idcok(WINDOW *_win_, bool _bf_);

说明

idcok() 函数指定实现是否可以在 *win* 中使用硬件插入和删除字符功能（如果终端配备了这样的功能）。如果 *bf* 为 TRUE，则在 *win* 中启用这些功能。如果 *bf* 为 FALSE，则在 *win* 中禁用这些功能。初始状态为 TRUE。

返回值

idcok() 函数不返回任何值。

错误

没有定义任何错误。

另请参阅

clearok(3X)、 doupdate(3X)、 <curses.h>。

历史变更记录

在 X/Open Curses 第 4 期中首次发布。

名称

if_nameindex()、if_nametoindex()、if_indextoname()、 if_freenameindex() - 在接口名称和索引值之间进行映射的函数

概要

#include <net/if.h>

unsigned int if_nametoindex(const char *ifname);

char *if_indextoname(unsigned int ifindex, char *ifname);

struct if_nameindex *if_nameindex(void);

void if_freenameindex(struct if_nameindex *ptr);

说明

通常，接口是通过“lan0”和“vlan200”等名称进行识别的，索引是分配给接口的唯一正整数值。索引值由 1 开始， 0 为无效的索引。

if_nametoindex()

该函数可将接口名映射为与其对应的索引。如果指定的接口名不存在，则将返回值设置为 0，同时将 **errno** 设置为 [ENXIO] 。如果存在系统错误，则返回值为 0，同时将 **errno** 设置为相应的值。例如，如果系统的内存不足，则将 **errno** 设置为 [ENOMEM] 。

if_indextoname()

该函数可将接口索引映射为与其对应的名称。 *ifname* 参数必须指向一个至少有 **IF_NAMESIZE** 字节，并且对应于指定索引的接口名将要返回到的缓冲区。该指针还是该函数的返回值。 **IF_NAMESIZE** 在 **<net/if6.h>** 中定义，其值包括接口名末尾的结尾空字节。如果没有对应于指定索引的接口，则返回 NULL，同时将 **errno** 设置为 [ENXIO] 。如果存在系统错误，则返回值为 0，同时将 **errno** 设置为相应的值。

if_nameindex()

该函数将返回作为 **if_nameindex** 结构数组的所有接口名和索引，其中每个接口对应一个结构。 **if_nameindex** 结构在 **<net/if.h>**. 中定义。

```
struct if_nameindex {
    unsigned int   if_index;  /* 1, 2, ...*/
    char         *if_name;   /* null terminated name:"lan0" */
};
```

结构数组的末尾以 *if_index* 为 0 且 *if_name* 为 NULL 的结构表示。出错时，该函数将返回 NULL 指针，同时将 **errno** 设置为相应的值。

该结构数组以及 *if_name* 成员指向的接口名所使用的内存是以动态方式获取的。下一个 **if_freenameindex()** 函数可释放该内存。

if_freenameindex()

该函数可释放 **if_nameindex()** 分配的动态内存。该函数的参数必须是 **if_nameindex()** 返回的指针。

历史变更记录

函数原型和 **struct if_nameindex** 以前在 HP-UX 11i v3 之前的 **<net/if6.h>** 中定义。现在它们在 **<net/if.h>** 中定义。

从 HP-UX 11i v3 开始，这些函数可在 IPv4 接口和 IPv6 接口上传递，并且它们不再驻留在 IPv6 库中，它们当前驻留在 C 库中。因此，使用这些函数的应用程序无须再链接到 **-lipv6** 。使用其他 IPv6 接口且驻留在 IPv6 库中的应用程序，并且仍需要链接到 **-lipv6** 。

作者

if_nametoindex() 、 **if_indextoname()** 、 **if_nameindex()** 和 **if_freenameindex()** 由 HP、IETF 和 X/Open Company Limited 联合开发。

错误

如果 **if_nametoindex()** 或 **if_indextoname()** 失败， **errno** 中将设置以下错误消息。

[ENXIO] 指定的接口或索引不存在。

另请参阅

ndp(1M)、inet6_opt_init(3N)、inet6_rth_space(3N)、ip6(7P)、ndp(7P)。

符合的标准

socket()：UNIX 03

名称

ilogb()、ilogbf()、ilogbl()、ilogbw()、ilogbq() - 与基数无关的指数函数

概要

```
#include <math.h>

int ilogb(double x);
```

仅适用于 **Integrity** 服务器

```
int ilogbf(float x);

int ilogbl(long double x);

int ilogbw(extended x);

int ilogbq(quad x);
```

说明

ilogb() 函数计算浮点值 *x* 的指数。在标准情况下，返回值是带符号的整数值 |*x*| 的 *r* 的对数底的整数部分，对于非零 *x* 而言，其中 *r* 是计算机的浮点运算的基数。在 HP-UX 系统中，基数 *r* 为 2。

如果 *x* 不标准，则在确定指数前将其规范化。

注释：对于除 NaN、±INFINITY 和零之外的所有 *x* 的值而言， **ilogb(***x***)** 等效于 **(int)logb(***x***)** 。

仅适用于 **Integrity** 服务器

ilogbf() 是 **ilogb()** 的 **float** 版本；它使用 **float** 参数。

ilogbl() 是 **ilogb()** 的 **long double** 版本；它使用 **long double** 参数。

ilogbw() 是 **ilogb()** 的一个 **extended** 版本；它使用 **extended** 参数。

在 HP-UX 系统中， **ilogbq()** 等效于 **ilogbl()** 。

用法

要使用这些函数，请使用缺省的 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。

（对于 Integrity 服务器）要使用 **ilogbw()** 或 **ilogbq()** ，也可以使用 **-fpwidetypes** 选项进行编译。

确保程序包含 **<math.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

有关详细信息，请参阅《HP-UX floating-point guide for HP Integrity servers》（位于以下站点：**http://www.hp.com/go/fp** ）。

返回值

如果 *x* 为 NaN，则 **ilogb()** 返回 **FP_ILOGBNAN** 。

如果 *x* 为 ±INFINITY，则 **ilogb()** 返回 **INT_MAX** 。

如果 *x* 为零，则 **ilogb()** 返回 **FP_ILOGB0** 。

仅适用于 **Integrity** 服务器

如果 *x* 是 NaN、+-INFINITY 或零，则会引发无效的浮点异常（根据 C99 TC2 的指定）

宏 **FP_ILOGBNAN** 和 **FP_ILOGB0** 都定义在 **<math.h>** 中。

错误

没有定义任何错误。

另请参阅

frexp(3M)、 logb(3M)、 scalb(3M)、 scalbn(3M)、 scalbln(3M)、 math(5)。

符合的标准

ilogb()： XPG4.2 和 ISO/IEC C99（包括附件 F“IEC 60559 floating-point arithmetic”）

ilogbf() 和 **ilogbl()**： ISO/IEC C99（包括附件 F“IEC 60559 floating-point arithmetic”）

名称

immedok — 启用或禁用立即终端刷新

概要

#include <curses.h>

void immedok(WINDOW ****win*, bool** *bf***);**

说明

当更改 win 指向的窗口时， **immedok()** 函数指定是否刷新屏幕。如果 *bf* 为 TRUE，则针对每个这种更改隐式刷新窗口。如果 *bf* 为 FALSE，则不隐式刷新窗口。初始状态为 FALSE。

返回值

immedok() 函数不返回任何值。

错误

没有定义任何错误。

实际应用信息

immedok() 函数对作为仿真终端使用的窗口有用。

另请参阅

clearok(3X)、 doupdate(3X)、 <curses.h>。

历史变更记录

在 X/Open Curses 第 4 期中首次发布。

名称

inch、mvinch、mvwinch 和 winch — 从窗口输入单字节字符和呈现

概要

#include <curses.h>

chtype inch(void);

chtype mvinch(int *y*, **int** *x*);

chtype mvwinch(WINDOW ****win*, **int *y*, **int** *x*);

**chtype winch(WINDOW ****win*);

说明

这些函数在当前或指定窗口中的当前或指定位置返回 *chtype* 类型的字符和呈现方式。

返回值

成功完成后，这些函数返回指定的字符和呈现方式。否则，它们返回 (**chtype**)ERR。

错误

没有定义任何错误。

实际应用信息

这些函数只保证对这样的字符集进行可靠操作：其中每个字符都属于单个字节，只能用带有 A_ 前缀的常量来表示其属性。

另请参阅

<curses.h>。

历史变更记录

在 X/Open Curses 第 2 期中首次发布。

X/Open Curses 第 4 期

为清楚起见，重新编写了该条目。将 **inch()** 函数的参数列表显式声明为 **void**。

名称

inchnstr、inchstr、mvinchnstr、mvinchstr、 mvwinchnstr、mvwinchstr、winchnstr、winchstr — 从窗口中输入单字节字符及呈现方式的数组

概要

#include <curses.h>

int inchnstr(chtype ****chstr*, int** *n***);**

int inchstr(chtype ****chstr*);**

int mvinchnstr(int *y***, int** *x***, chtype ****chstr***, int** *n***);**

int mvinchstr(int *y***, int** *x***, chtype ****chstr***);**

int mvwinchnstr(WINDOW ****win*, int** *y***, int** *x***, chtype ****chstr***, int** *n***);**

int mvwinchstr(WINDOW ****win*, int** *y***, int** *x***, chtype ****chstr***);**

int winchnstr(WINDOW ****win*, chtype ****chstr***, int** *n***);**

int winchstr(WINDOW ****win*, chtype ****chstr***);**

说明

这些函数将从当前窗口或所指定的窗口中将字符及呈现方式放入 *chstr* 所指向的数组中，起止位置如下：从当前位置或所指定的位置开始，到行尾结束。

inchnstr() 、 **mvinchnstr()** 、 **mvwinchnstr()** 和 **winchnstr()** 函数最多可将当前窗口或所指定窗口中的 *n* 个元素存储到 *chstr* 所指向的数组中。

返回值

成功完成后，这些函数将返回 OK。否则，它们将返回 ERR。

错误

未定义错误。

实际应用信息

对于 *chstr* 所指向的数组而言，当用 **inchstr()** 、 **mvinchstr()** 、 **mvwinchstr()** 或 **winchstr()** 读取导致其溢出的行时，可产生不可预测的结果。建议分别使用 **inchnstr()** 、 **mvinchnstr()** 、 **mvwinchnstr()** 或 **winchnstr()** 。

另请参阅

inch(3X)、 <curses.h>。

历史变更记录

在 X/Open Curses 第 4 期中首次发布。

名称

inet : inet_addr()、 inet_lnaof()、 inet_makeaddr()、 inet_netof()、 inet_network()、 inet_ntoa()、 inet_ntoa_r() - Internet 地址操作例行程序

概要

#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

in_addr_t inet_addr(const char *_cp_);

in_addr_t inet_lnaof(struct in_addr _in_);

struct in_addr inet_makeaddr(in_addr_t _net_, in_addr_t _lna_);

in_addr_t inet_netof(struct in_addr _in_);

in_addr_t inet_network(const char *_cp_);

char *inet_ntoa(struct in_addr _in_);

备注

在下面的“过时的接口”一节中描述了 **inet_ntoa_r()** 例行程序。

说明

inet_addr() 和 **inet_network()**

说明以 Internet 标准的“点分”表示法表示的、代表数字的字符串。

inet_addr() 返回适合用作 Internet 地址的数字。

inet_network() 返回适合用作 Internet 网络号的数字。

通过类似于下面的方法，可将返回值赋给 **struct in_addr** （在 **/usr/include/netinet/in.h** 中定义）：

```
struct in_addr addr;
char *cp;

addr.s_addr = inet_addr(cp);
```

inet_ntoa() 使用一个 Internet 地址，同时返回一个 ASCII 字符串，表示以点分 (**.**) 表示法表示的地址。

inet_makeaddr() 使用一个 Internet 网络号和一个本地网络地址，同时通过该地址构建一个 Internet 地址。

inet_netof() 分解 Internet 主机地址，同时返回网络号部分。

inet_lnaof() 分解 Internet 主机地址，同时返回本地网络地址部分。

将以网络顺序（按从左至右的顺序排列字节）返回所有 Internet 地址。将所有网络号和本地地址部分作为计算机格式整数值返回。 HP-UX 系统中的字节按从左至右的顺序排列。

Internet 地址

使用点分表示法指定的值采用下列格式之一:

a.*b*.*c*.*d*
a.*b*.*c*
a.*b*
a

如果指定了四个部分，则将每个部分解释为数据的一个字节，并且按从左至右的顺序将其分配给 Internet 地址的四个字节。

如果指定了有三个部分的地址，则将最后一个部分解释为 16 位的量，并且将其置于网络地址的最右边两个字节内。这样就可以使用具有三个部分的地址格式方便地指定 B 类网络地址，例如， **128.net.host** 。

如果提供具有两个部分的地址，则将最后一个部分解释为 24 位的量，并且将其置于网络地址的最右边三个字节内。这样就可以使用具有两个部分的地址格式方便地指定 A 类网络地址，例如， **net.host** 。

如果只指定了一个部分，则将直接在网络地址中存储该值，而不会重新排列任何字节。

作为点分表示法的部分提供的所有数字，都可以是 C 语言中指定的十进制、八进制或十六进制数（也就是说，前导 0x 或 0X 表示十六进制数；前导 0 表示八进制数；否则，该数字解释为十进制数）。

在多线程应用程序中， **inet_ntoa()** 使用每次调用可再使用的特定线程存储。对于每个线程，返回值（字符串）应该是唯一的，线程执行下一次 **inet_ntoa()** 调用之前，应该按需要保存该返回值。

过时的接口

以下重入接口已从 **libc** 移至 **libd4r** 。

int inet_ntoa_r(struct in_addr *in***, char ****buffer***, int** *buflen***);**

包括该接口是为支持现有应用程序，在将来版本中可能会将其删除。新的多线程应用程序应该使用常规 API（不带 **_r** 后缀）。

重入接口与不带 **_r** 后缀的常规接口的工作方式相同。但是，需要为 **inet_ntoa_r()** 传递字符缓冲区的地址，同时该接口将在提供的位置中存储结果。如果缓冲区的长度不足，则将返回 **-1** 。如果操作成功，则返回结果字符串的长度（不包括以 null 结尾的字符）。

返回值

这些例行程序返回的值在“说明”一节中进行了描述。 **inet_addr()** 和 **inet_network()** 将返回 **-1** (**INADDR_NONE**) 。

警告

inet_addr() 函数的返回值不能区分失败 (**-1**) 与本地广播地址 (255.255.255.255)。可以通过使用 **inet_pton()** 函数，而不是 **inet_addr()** 函数来处理这种情况。

作者

inet 例行程序由加州大学伯克利分校开发。

另请参阅

gethostent(3N)、 getnetent(3N)、 inet6(3N)、 hosts(4)、 networks(4)、 thread_safety(5)。

名称

inet_pton()、inet_ntop() - IP Version 4 及更高版本的 Internet 地址操作例行程序

概要

```
#include <sys/socket.h>
#include <arpa/inet.h>

int inet_pton(int af, const char *src, void *dst);

const char *inet_ntop(int af, const void *src,
        char *dst, size_t size);
```

说明

inet_pton() 和 **inet_ntop()** 是 IP Version 6 (IPv6) 上的新函数，它们也适用于 IP Version 4 (IPv4) 和 IPv6 地址。字母“p”和“n”表示 **presentation** 和 **numeric** 。地址的表示格式通常采用 ASCII 字符串，数字格式是将要加入套接字地址结构的二进制值。

ASCII 字符串可以是用点表示的 IPv4 地址，也可以是用冒号表示的 IPv6 地址。ASCII 字符串的示例包括 **15.10.43.255** 或 **1080::2538:400:25:800:200C:417A** 。

二进制值是 IPv4/IPv6 地址的十六进制表示形式。对于 IPv4 地址，此二进制值驻留在 **in_addr** 结构中；对于 IPv6 地址，此二进制值驻留在 **in6_addr** 结构中（有关详细信息，请参阅 *inet*(3N) 联机帮助页）。

函数 **inet_pton()** 和 **inet_ntop()** 是彼此相对的。函数 **inet_pton()** 将 ASCII 地址字符串转换为二进制地址；函数 **inet_ntop()** 将二进制地址转换为 ASCII 地址字符串。此外，它们分别与 **inet_addr()** 和 **inet_ntoa()** 等效。

inet_pton() 函数

inet_pton() 函数可将地址的标准文本表示格式转换为数字二进制格式。

af 参数指定地址的系列。目前支持 **AF_INET** 和 **AF_INET6** 地址系列。

src 参数指向被传入的 IPv6/IPv4 地址字符串。

dst 参数指向函数将在其中存储数字地址的缓冲区。该地址将按网络字节顺序返回。

如果转换成功， **inet_pton()** 将返回 **1** ；如果输入不是有效的 IPv4 点分十进制字符串或有效的 IPv6 地址字符串，则返回 **0** ；如果 *af* 参数未知，则返回 **-1** ，并将 errno 设置为 [EAFNOSUPPORT] 。

调用应用程序必须确保 *dst* 引用的缓冲区大小足以保存数字地址（例如， **AF_INET** 的 4 个字节，或 **AF_INET6** 的 16 个字节）。

如果 *af* 参数为 **AF_INET** ，则函数可接受标准的 IPv4 点分十进制格式的字符串：

ddd.ddd.ddd.ddd

其中 *ddd* 是介于 0 和 255 之间的一至三位十进制数字。请注意，现有的 **inet_addr()** 和 **inet_aton()** 函数的许多实现都接受非标准输入：八进制数字、十六进制数字，以及少于四位的数字。而 **inet_pton()** 不接受这些格式。

如果 *af* 参数为 **AF_INET6** ，则函数可接受其中一种标准的 IPv6 文本格式的字符串（有关详细信息，请参考下文

所述的 IPv6 地址表示法）。

inet_ntop() 函数

inet_ntop() 函数可将数字地址转换为适当形式的文本字符串。

af 参数指定地址的系列。系列可能是 **AF_INET** 或 **AF_INET6** 。

如果 *af* 参数为 **AF_INET** ，则 *src* 参数指向保存 IPv4 地址的缓冲区；如果 *af* 参数为 **AF_INET6** ，则指向保存 IPv6 地址的缓冲区。

dst 参数指向函数将在其中存储得到的文本字符串的缓冲区。

size 参数指定该缓冲区的大小。应用程序必须指定一个非 NULL *dst* 参数。对于 IPv6 地址，缓冲区至少必须为 46 个八位字节。对于 IPv4 地址，缓冲区至少必须为 16 个八位字节。为了使应用程序方便地声明用来存储字符串格式 IPv4 和 IPv6 地址的具有适当大小的缓冲区， **<netinet/in.h>** 中定义了以下两个常量：

```
#define INET_ADDRSTRLEN   16
#define INET6_ADDRSTRLEN   46
```

如果转换成功， **inet_ntop()** 函数将返回一个指向包含文本字符串的缓冲区的指针；否则，返回 NULL。在失败的情况下，如果 *af* 参数无效，会将 errno 设置为 [EAFNOSUPPORT] ；如果结果缓冲区的大小不足，则将 errno 设置为 [ENOSPC] 。

IPv6 地址表示法

IPv6 地址的长度为 **128** 字节。 IPv6 地址的示例如下：

FEDC:BA98:7654:3210:FEDC:BA98:7654:3210

128 位编写为冒号分隔的八个 16 位整数。用四个十六进制数字表示每个整数。

如果在初始阶段，所有 128 位都未被使用，则很有可能是存在多个零。因此，这种实例中的 IP 地址类似于：

1080:0:0:0:0:800:200C:417A

可以通过紧凑的形式表示上面的地址，即用两个冒号来代替一组连续的 16 位空数字。这样，上面的地址可重新编写如下：

1080::800:200C:417A

对这种简略形式进行扩展是很简单的。将双冒号左边的整个部分与原地址的左边对齐：这些是 16 位前导字。然后将双冒号右边的整个部分与原地址的右边对齐，再填充零。

双冒号约定只能在某个地址内使用一次。

为了支持对 IPv4 的转换，可支持两个特殊的 IPv6 地址。它们分别是 与 **IPv4** 兼容的地址和 由 **IPv4** 映射的地址。

与 **IPv4** 兼容的地址说明和示例

与 IPv4 兼容的地址与早期的 IPv4 网络地址格式可以互相转换。当 IPv6 系统之间需要互相通信，但中间隔有 IPv4 网络时，便可以使用与 IPv4 兼容的地址。

与 IPv4 兼容的地址的构成方法是，在有效的 32 位 IPv4 地址前面添加 96 个为零的位。例如，要将 IPv4 地址

1.2.3.4

转换为 IPv6 地址，可以首先将每个以点（“.”）分隔的十进制数转换为 2 位十六进制值，然后连接成冒号（“:”）分隔的 4 位十六进制值，其格式如下。

::0102:0304

此外，可以省略冒号之间的每个十六进制数的前导零。因此，这个 IPv4 地址可编写为以下 IPv6 地址:

::102:304

可以保留点分的十进制格式，将上述地址重写为

::1.2.3.4

以上地址也是与 IPv4 兼容的有效 IPv6 地址。

由 **IPv4** 映射的地址说明和示例

由 IPv4 映射的地址用于指示不支持 IPv6 的系统。它们只限于 IPv4。IPv6 主机可以使用由 IPv4 映射的 IPv6 地址与 IPv4 主机通信。

由 IPv4 映射的地址的构成方法是，在有效的 32 位 IPv4 地址前面添加 80 个为零的位，及 16 个为一的位。将 IPv4 地址

1.2.3.5

映射至 IPv6 后，将成为:

::FFFF:1.2.3.5

或者

::FFFF:102:305

作者

这些 **inet** 例行程序由加州大学伯克利分校开发。

另请参阅

gethostent(3N)、 getnetent(3N)、 inet(3N)、 hosts(4)、 networks(4)。

名称

inet6_opt_init() 、 inet6_opt_append() 、 inet6_opt_find() 、 inet6_opt_finish() 、 inet6_opt_get_val() 、 inet6_opt_next()、 inet6_opt_set_val() - IPv6 逐跃点选项及目的地选项操作函数。

概要

```
#include <netinet/in.h>

int inet6_opt_append(void *extbuf, size_t extlen, int prevlen,
                     uint8_t type, size_t len, uint_t align,
                     void **databufp);

int inet6_opt_find(void *extbuf, size_t extlen, int prevlen,
                   uint8_t type, size_t *lenp,
                   void **databufp);

int inet6_opt_finish(void *extbuf, size_t extlen, int prevlen);

int inet6_opt_get_val(void *databuf, size_t offset, void *val,
                      int vallen);

int inet6_opt_init(void *extbuf, size_t extlen);

int inet6_opt_next(void *extbuf, size_t extlen, int prevlen,
                   uint8_t *typep, size_t *lenp,
                   void **databufp);

int inet6_opt_set_val(void *databuf, size_t offset, void *val,
                      int vallen);
```

说明

应用程序可以使用这些函数来构建和分析 IPv6 逐跃点选项和目的地选项标头。逐跃点选项标头用于沿数据包传递路径携带每个节点必须检查的可选信息，目的地选项标头用于传送只需要由数据包目的地节点检查的可选信息。

要接收逐跃点选项，应用程序必须启用 **IPV6_RECVHOPOPTS** 套接字选项：

```
int  on = 1;
setsockopt(fd, IPPROTO_IPV6, IPV6_RECVHOPOPTS, &on, sizeof(on));
```

要发送逐跃点选项标头，应用程序需要将标头指定为调用 **sendmsg()** 时的辅助数据，或者将标头指定为使用 **setsockopt()** 。

应用程序使用零选项长度调用 **IPV6_HOPOPTS** 的 **setsockopt()** ，可以删除任何依附的逐跃点扩展标头。

要接收显示在路由选择标头之后（或没有路由选择标头的数据包内）的目的地选项，应用程序必须启用 **IPV6_RECVDSTOPTS** 套接字选项：

```
int  on = 1;
setsockopt(fd, IPPROTO_IPV6, IPV6_RECVDSTOPTS, &on, sizeof(on));
```

要接收显示在路由选择标头之前的目的地选项，应用程序必须启用 **IPV6_RECVRTHDRDSTOPTS** 套接字选项：

```
int  on = 1;
setsockopt(fd, IPPROTO_IPV6, IPV6_RECVRTHDRDSTOPTS, &on, sizeof(on));
```

要发送目的地选项标头，应用程序需要将它指定为调用 **sendmsg()** 时的辅助数据，或者将其指定为使用 **setsockopt()** 。

应用程序使用零选项长度调用 **IPV6_RTHDRDSTOPTS/IPV6_DSTOPTS** 的 **setsockopt()** ，可以删除任何依附的目的地扩展标头。

inet6_opt_init()

此函数返回不带任何选项的空扩展标头所需的字节数。如果 *extbuf* 不为 NULL，它还将初始化扩展标头，使之具有正确的长度字段。如果 *extlen* 值不是 8 的正倍数，函数将失败，并返回 -1。

inet6_opt_append()

此函数返回添加带有长度 *len* 和对齐 *align* 的选项时所需的最新总长度。如果 *extbuf* 不是 NULL，除了返回长度外，此函数还会插入任何必要的填充选项，初始化选项（设置类型和长度字段），并返回指向 *databufp* 中选项内容的位置的指针。如果 *type* 指定的选项不适合扩展标头缓冲区，函数将返回 -1。

第三个参数 *prevlen* 应该是 **inet6_opt_init()** 返回的或上一个 **inet6_opt_append()** 返回的长度。

参数 *type* 是 8 位选项类型，它必须是介于 2 和 255 之间的值（包括边界值）。0 和 1 分别为 Pad1 和 PadN 选项保留。

参数 *len* 是不包括选项类型和选项长度字段在内的选项数据的长度。它的值必须介于 0 和 255 之间（包括边界值），用于指定选项标头之后的选项数据的长度。

align 参数的值必须为 1、2、4 或 8。 *align* 值不可超过 *len* 值。

一旦调用 **inet6_opt_append()** ，应用程序可以直接使用 *databuf* ，或者使用 **inet6_opt_set_val()** 来指定（设置）选项的内容。

inet6_opt_finish()

此函数返回包括扩展标头最终填充量（为使标头成为 8 字节的倍数）在内的最新总长度。

如果 *extbuf* 不是 NULL，函数还将通过插入适当长度的 Pad1 或 PadN 选项来初始化选项。如果必要的填充不适合扩展标头缓冲区，函数将返回 -1。

参数 *prevlen* 应该是 **inet6_opt_init()** 或 **inet6_opt_append()** 返回的长度。

inet6_opt_set_val()

此函数在选项的数据部分插入不同大小（1、2、4 或 8 字节）的数据项目。插入完成后，函数将返回下一个字段 (*offset* + *vallen*) 的偏移量，编写带有多个字段的选项内容时可以使用该偏移量。参数 *val* 指向要插入数据，参数 *offset* 指定应该在其中插入值的选项的数据部分中的位置。通过指定零偏移量，可以访问选项类型和长度之后的第一个字节。

参数 *databuf* 应该是 **inet6_opt_append()** 返回的指针。

inet6_opt_next()

此函数可分析应用程序收到的扩展标头选项，及返回下一选项的偏移量。

参数 *extbuf* 和 *extlen* 指定扩展标头。

参数 *prevlen* 可能是零（针对第一个选项），也可能是以前调用 **inet6_opt_next()** 或 **inet6_opt_find()** 后返回的长度。 *prevlen* 指定从此处继续扫描扩展缓冲区的位置。可通过更新参数 *databufp typep* 和 *lenp* 返回下一个选项。此函数返回通过越过已返回选项而计算出的最新“上一个”长度。然后可以将此返回的“上一个”长度传递给 **inet6_opt_next()** 的后续调用。

此函数不返回任何 Pad1 或 PadN 选项。如果没有更多的选项，或者选项扩展标头的格式不当，则返回值为 -1。

inet6_opt_find()

此函数与 **inet6_opt_next()** 函数相似，但是此函数允许调用方指定需要搜索选项类型，而不是让选项类型始终在扩展标头中返回下一个选项。

如果找到指定的 *type* 的选项，函数将返回最新的“上一个”总长度。此“上一个”总长度是通过越过已返回选项，及越过任何与类型不匹配的选项而计算出来的。然后，所返回的上一个 *length* 被传递给 **inet6_opt_find()** 的后续调用，此调用是为了查找下一个出现的相同选项类型。

如果未找到指定的 *type* 的选项，或者选项扩展标头的格式不当，则返回值为 -1。

inet6_opt_get_val()

此函数在选项的数据部分提取不同大小（1、2、4 或 8 字节）的数据项目，并返回下一个字段 (*offset* + *vallen*) 的偏移量，提取带有多个字段的选项内容时可以使用该偏移量。

第一个参数 *databuf* 应该是包含选项数据部分的 **inet6_opt_next()** 或 **inet6_opt_find()** 返回的指针。

参数 *offset* 指定应该从其中提取值的选项数据部分中的位置。通过指定零偏移量，可以访问选项类型和长度之后的第一个字节。

val 指向已提取数据的目的地。

要使用这些函数，必须通过以下方式编译应用程序：

-D_HPUX_SOURCE -D_XOPEN_SOURCE_EXTENDED -lxnet -lipv6

这些 API 是根据互联网草案 Advanced Sockets API for IPv6 <draft-ietf-ipngwg-2292bis-02.txt> 实现的。可以根据将来对文档的更改来更新、替换或废弃这些 API。

返回值

出现错误时， **inet6_opt_init()** 、 **inet6_opt_append()** 、 **inet6_opt_finish()** 、 **inet6_opt_next()** 和 **inet6_opt_find()** 将返回 -1。

举例

Advanced Sockets API for IPv6 <draft-ietf-ipngwg-2292bis-02.txt> 文档的附录 C 提供了综合示例。

作者

这些 API 由 HP 开发。

另请参阅

ip6(7P)。

Advanced Sockets API for IPv6 <draft-ietf-ipngwg-2292bis-02.txt> 。

名称

inet6_rth_add() 、 inet6_rth_getaddr() 、 inet6_rth_init() 、 inet6_rth_reverse() 、 inet6_rth_segments() 、 inet6_rth_space() - IPv6 路由标头选项操作函数。

概要

#include <netinet/in.h>

size_t inet6_rth_space(int type, int segments);

void *inet6_rth_init(void *bp, int bp_len, int type, int segments);

int inet6_rth_add(void *bp, const struct in6_addr *addr);

int inet6_rth_reverse(const void *in, void *out);

int inet6_rth_segments(const void *bp);

struct in6_addr *inet6_rth_getaddr(const void *bp, int index);

说明

应用程序可使用这些函数来构建和检查 IPv6“路由”标头。IPv6 源可以使用“路由”标头来列出通往数据包目标时将要访问的一个或多个中间节点。

这三个函数可构建一个“路由”标头：

inet6_rth_space() 可返回“路由”标头所需的字节数。

inet6_rth_init() 初始化“路由”标头的缓冲区数据。

inet6_rth_add() 将一个 IPv6 地址添加到“路由”标头。

以下三个函数处理返回的“路由”标头：

inet6_rth_reverse() 可反转“路由”标头。

inet6_rth_segments() 可返回“路由”标头中的段数。

inet6_rth_getaddr() 从“路由”标头提取一个地址。

要接收“路由”选项，应用程序必须启用 **IPV6_RECVRTHDR** 套接字选项：

```
int on = 1;
setsockopt(fd, IPPROTO_IPV6, IPV6_RECVRTHDR, &on, sizeof(on));
```

要发送“路由”标头，应用程序需要将它指定为调用 **sendmsg()** 时的辅助数据，或者将其指定为使用 **setsockopt()**（请分别参阅 *send*(2) 和 *getsockopt*(2)）。

inet6_rth_space()

此函数可返回保存 *type* 指定的，并且包含指定的 *segments*（地址）数目的路由标头所需的字节数。对于 IPv6 类型 0 的“路由”标头，段数必须介于 0 和 127（包括边界值）之间。返回值正好是“路由”标头使用的空间。如果应用程序使用辅助数据，则必须将返回的长度传递给 **CMSG_LEN()**，以确定辅助数据对象

（包括 cmsghdr 结构）所需的内存量。

如果返回值为 0，则表示要么此实现不支持“路由”标头的类型，要么段数对此类型的“路由”标头无效。

此函数将返回大小，但不分配辅助数据所需的空间。

inet6_rth_init()

此函数可初始化 *bp* 指向的缓冲区，使之包含指定的 *type* 的“路由”标头。 *bp_len* 只用于验证缓冲区的大小是否足够。

调用方必须分配缓冲区，而缓冲区大小可通过调用 **inet6_rth_space()** 来确定。

如果成功，返回值将是指向缓冲区 (*bp*) 的指针，以后可以将此指针用作 **inet6_rth_add()** 函数的第一个参数。如果出现错误，则返回值为 NULL。

inet6_rth_add()

此函数可以将 *addr* 指向的 IPv6 地址添加到所构建的“路由”标头的结尾。

如果成功，“路由”标头的 segleft 成员将更新为“路由”标头中的新地址的帐户，同时函数的返回值为 0。如果出现错误，函数的返回值为 -1。

inet6_rth_reverse()

此函数将使用第一个参数 *in* 指向的“路由”标头的扩展标头，并写入新的“路由”标头。新的“路由”标头将沿着该线路相反的方向发送数据报。此函数可反转地址的顺序，并将新路由标头中的 segleft 成员设置为段数。允许这两个参数都指向同一个缓冲区（也就是说，可以随地发生反转）。

如果成功，此函数的返回值为 0，如果出现错误，返回值为 -1。

inet6_rth_segments()

此函数可返回 *bp* 描述的“路由”标头包含的段（地址）数，该数目可能是 0 或更大值。

如果出现错误，此函数的返回值为 -1。

inet6_rth_getaddr()

此函数可返回一个指向 IPv6 地址的指针，而该地址是由 *bp* 描述的“路由”标头中的索引（必须是介于 0 和 **inet6_rth_segments()** 返回的值减一之间的值）指定的。

应用程序应首先调用 **inet6_rth_segments()** 来获取“路由”标头中的段数。

如果出现错误，此函数的返回值为 NULL。

要使用这些函数，必须通过以下方式编译应用程序：

-D_HPUX_SOURCE -D_XOPEN_SOURCE_EXTENDED -lxnet -lipv6

这些 API 是根据互联网草案 Advanced Sockets API for IPv6 <draft-ietf-ipngwg-2292bis-02.txt> 实现的。根据将来对此文档的更改，可能会更新、替换或废弃这些 API。

举例

Advanced Sockets API for IPv6 <draft-ietf-ipngwg-2292bis-02.txt> 的附录 B 提供了综合示例。

作者

这些 API 由 HP 开发。

另请参阅

Advanced Sockets API for IPv6 <draft-ietf-ipngwg-2292bis-02.txt> 。

名称

initgroups( ) - 初始化组访问列表

概要

#include <unistd.h>

int initgroups(const char *name, gid_t basegid);

说明

initgroups() 读取登录组文件 **/etc/logingroup** ，并使用 *setgroups*(2) 系统调用为 *name* 指定的用户设置组访问列表。如果 *basegid* 的值为零或正数，则其自动包含在该组列表中。该值通常作为口令文件的组号给定。如果登录组文件为空，则 *basegid* 是该列表的唯一成员。

返回值

如果 **initgroups()** 不是由具有相应特权的用户调用，则它返回 **-1** 。

警告

initgroups() 使用基于 *getgrent*(3C) 的例行程序。如果调用程序使用所有这些例行程序，则调用 **initgroups()** 会覆盖组结构。随后调用带有相同 *name* 参数的 **initgroups()** 会覆盖以前调用的操作。

在许多系统中，似乎没有一个系统能保持 **/etc/logingroup** 是最新的。

initgroups() 使用动态名称服务交换（请参阅 *nsswitch.conf*(4) ）。使用该接口的应用程序不能完全归档绑定。

网络功能

NFS

如果将 **/etc/logingroup** 链接到 **/etc/group** ，则 **initgroups()** 尝试使用以加号 **(+)** 开头的条目的网络信息服务 (NIS)。如果 *name* 的组成员关系由 NIS 管理，并且 NIS 服务器没有响应，则直到服务器有响应时调用 **initgroups()** 才返回。这使得诸如 *login*(1) 和 *su*(1) 等命令无限期等待。

请参阅 *group*(4) ，以了解正确的语法与操作。

作者

initgroups() 由加州大学伯克利分校开发。

文件

/etc/logingroup 登录组文件

另请参阅

login(1)、su(1)、getgroups(2)、setgroups(2)、group(4)、 thread_safety(5)。

名称

initscr 和 newterm — 屏幕初始化函数

概要

#include <curses.h>

WINDOW *initscr(void);

SCREEN *newterm(char *_type_, FILE *_outfile_, FILE *_infile_);

说明

initscr() 函数确定终端类型并初始化所有实现数据结构。 **TERM** 环境变量指定终端类型。 **initscr()** 函数还会引发第一次刷新操作以清除屏幕显示。如果发生错误， **initscr()** 将向标准错误中写入一个适当的错误消息并退出。仅可以在 **initscr()** 或 **newterm()** 之前调用的函数是 **filter()** 、 **ripoffline()** 、 **slk_init()** 、 **use_env()** 以及那些原型在 **<term.h>** 中定义的函数。可移植应用程序决不能调用 **initscr()** 两次。

newterm() 函数可以根据附加终端设备需要的次数任意调用。 *type* 参数指向指定终端类型的字符串，只不过 *type* 是一个空指针，需要使用 **TERM** 环境变量。*outfile* 和 *infile* 参数分别是用于向终端输出和从终端输入的文件指针。未指定 Curses 是否修改这些文件指针的缓冲模式。对于每个终端都应调用一次 **newterm()** 函数。

initscr() 函数等效于:

```
newterm(getenv("TERM"), stdout, stdin);
return stdscr;
```

如果信号 SIGINT、SIGQUIT 或 SIGTSTP 的当前配置是 SIGDFL，则 **initscr()** 还可以安装信号的处理程序，这会在进程生命周期内保持其有效性或者直到进程更改信号的配置为止。

initscr() 和 **newterm()** 函数用于启动 *cur_term* 外部变量。

返回值

成功完成后， **initscr()** 将返回一个指向 *stdscr* 的指针，否则，将什么也不返回。

成功完成后， **newterm()** 将返回一个指向指定终端的指针。否则，返回空指针。

错误

没有定义任何错误。

实际应用信息

输出到多个终端的程序应对每个终端使用 **newterm()** ，而不是 **initscr()** 。需要错误状态的指示才能继续以面向行的模式运行的程序，如果终端不能支持面向屏幕的程序，则也需要使用此函数。

应用程序在调用 **initscr()** 之前，应对 SIGINT、SIGQUIT 或 SIGTSTP 信号执行所需的处理。

另请参阅

delscreen(3X)、 doupdate(3X)、 del_curterm(3X)、 filter(3X)、 slk_attroff(3X)、 use_env(3X)、 terminfo(4)，参阅选择终端、<curses.h>。

历史变更记录

在 X/Open Curses 第 2 期中首次发布。

X/Open Curses 第 **4** 期

newterm() 函数与此条目合并。在以前各期中，它出现在自己的条目中。

为清楚起见，重新编写了此条目。 **initscr()** 函数的参数列表已明确地声明为 **void**。

名称

innstr、instr、mvinnstr、mvinstr、mvwinnstr、mvwinstr、winnstr 和 winstr — 从窗口输入多字节字符型字符串

概要

#include <curses.h>

int innstr(char ****str*, int** *n***);**

int instr(char ****str*);**

int mvinnstr(int *y***, int** *x***, char ****str***, int** *n***);**

int mvinstr(int *y***, int** *x***, char ****str***);**

int mvwinnstr(WINDOW ****win*, int** *y***, int** *x***, char ****str***, int** *n***);**

int mvwinstr(WINDOW ****win*, int** *y***, int** *x***, char ****str***);**

int winnstr(WINDOW ****win*, char ****str***, int** *n***);**

int winstr(WINDOW ****win*, char ****str***);**

说明

这些函数从当前或指定窗口将字符串置于 *str* 所指向的数组中，其位置为在当前或指定位置开始，至行末尾结束。

innstr() 、 **mvinnstr()** 、 **mvwinnstr()** 和 **winnstr()** 函数在 *str* 所指向的字符串中最多存储 *n* 个字节。

innstr() 、 **mvinnstr()** 、 **mvwinnstr()** 和 **winnstr()** 函数仅存储与字符关联的整个多字节序列。如果数组至少包含一个字符，则使用完整的字符填充该数组。如果数组无法包含任何完整字符，则函数失败。

返回值

成功完成后， **instr()** 、 **mvinstr()** 、 **mvwinstr()** 和 **winstr()** 返回 OK。

成功完成后， **innstr()** 、 **mvinnstr()** 、 **mvwinnstr()** 和 **winnstr()** 返回实际读入字符串中的字符数。

否则，所有这些参数都将返回 ERR。

错误

没有定义任何错误。

实际应用信息

由于某些多字节字符可能已处理，因此屏幕上的列位置与返回的字节数可能不会一一对应。

这些函数不返回重发信息。

使用 **instr()** 、 **mvinstr()** 、 **mvwinstr()** 或 **winstr()** 读取溢出 *str* 指向的数组的行会产生不确定的结果。建议分别使用 **innstr()** 、 **mvinnstr()** 、 **mvwinnstr()** 或 **winnstr()** 。

另请参阅

<curses.h>。

历史变更记录

在 X/Open Curses 第 4 期中首次发布。

名称

innwstr、inwstr、mvinnwstr、mvinwstr、mvwinnwstr、mvwinwstr、winnwstr、winwstr — 从窗口输入宽字符的字符串

概要

#include <curses.h>

int innwstr(wchar_t ****wstr*, int** *n***);**

int inwstr(wchar_t ****wstr*);**

int mvinnwstr(int *y***, int** *x***, wchar_t ****wstr***, int** *n***);**

int mvinwstr(int *y***, int** *x***, wchar_t ****wstr***);**

int mvwinnwstr(WINDOW ****win*, int** *y***, int** *x***, wchar_t ****wstr***, int** *n***);**

int mvwinwstr(WINDOW ****win*, int** *y***, int** *x***, wchar_t ****wstr***);**

int winnwstr(WINDOW ****win*, wchar_t ****wstr***, int** *n***);**

int winwstr(WINDOW ****win*, wchar_t ****wstr***);**

说明

这些函数将从当前窗口或指定的窗口中将 **wchar_t** 字符串放入 *wstr* 所指向的数组中，起止位置为从当前光标位置或所指定的光标位置开始，到行尾结束。

这些函数将仅存储与占位复合字符关联的整个宽字符序列。如果数组足够大，至少可包含一个完整的占位复合字符，则数组中将填充完整的字符。如果数组不够大，无法包含任何完整的字符，就会出现错误。

在 *wstr* 所指向的数组中， **innwstr()** 、 **mvinnwstr()** 、 **mvwinnwstr()** 和 **winnwstr()** 函数最多可存储 *n* 个字符。

返回值

成功完成后， **inwstr()** 、 **mvinwstr()** 、 **mvwinwstr()** 和 **winwstr()** 将返回 OK。

成功完成后， **innwstr()** 、 **mvinnwstr()** 、 **mvwinnwstr()** 和 **winnwstr()** 将返回实际读入字符串中的字符数。

否则，所有这些函数都将返回 ERR。

错误

未定义错误。

实际应用信息

对于 *wstr* 所指向的数组而言，当用 **inwstr()** 、 **mvinwstr()** 、 **mvwinwstr()** 或 **winwstr()** 读取导致其溢出的行时，可产生不可预测的结果。建议分别使用 **innwstr()** 、 **mvinnwstr()** 、 **mvwinnwstr()** 或 **winnwstr()** 。

这些函数不会返回呈现方式信息。

另请参阅

<curses.h>。

历史变更记录

第一版发行在 X/Open Curses 第 4 期中。

名称

insch、mvinsch、mvwinsch 和 winsch — 向窗口插入单字节字符和呈现方式

概要

#include <curses.h>

int insch(chtype *ch***);**

int mvinsch(int *y***, int** *x***, chtype** *ch***);**

int mvwinsch(WINDOW ****win*, int** *y***, int** *x***, chtype** *ch***);**

int winsch(WINDOW ****win*, chtype** *ch***);**

说明

这些函数将 *ch* 的字符和呈现方式插入当前或指定位置的当前或指定窗口。

这些函数不执行换行操作。这些函数不会使光标位置向前移动。这些函数执行特殊的字符处理操作，但例外情况是：如果将换行符插入窗口的最后一行，并启用滚动，则未指定此操作。

返回值

成功完成后，这些函数返回 OK。否则，它们返回 ERR。

错误

没有定义任何错误。

实际应用信息

这些函数只保证对这样的字符集进行可靠操作：其中每个字符都属于单个字节，只能用带有 A_ 前缀的常量来表示其属性。

另请参阅

ins_wch(3X)、 <curses.h>。

历史变更记录

在 X/Open Curses 第 2 期中首次发布。

X/Open Curses 第 **4** 期

为清楚起见，重新编写了该条目。

名称

insdelln 和 winsdelln — 从窗口删除行或向窗口插入行

概要

#include <curses.h>

int insdelln(int *n***);**

int winsdelln(WINDOW ****win*, int** *n***);**

说明

insdelln() 和 **winsdelln()** 函数执行以下操作：

- 如果 *n* 为正，则这些函数在当前或指定窗口中的当前行之前插入 *n* 行。不再显示最后 *n* 行。
- 如果 *n* 为负，则这些函数从当前或指定窗口中删除自当前行开始的 *n* 行，并将剩余各行移到光标位置。最后 *n* 行被清除。

当前光标位置与以前相同。

返回值

成功完成后，这些函数返回 OK。否则，它们返回 ERR。

错误

没有定义任何错误。

另请参阅

deleteln(3X)、 insertln(3X)、 <curses.h>。

历史变更记录

在 X/Open Curses 第 4 期中首次发布。

名称

insertln 和 winsertln — 向窗口插入行

概要

#include <curses.h>

int insertln(void);

int winsertln(WINDOW *_win_);

说明

insertln() 和 **winsertln()** 函数在当前或指定窗口中的当前行之前插入空白行。不再显示底行。光标位置没有改变。

返回值

成功完成后，这些函数返回 OK。否则，它们返回 ERR。

错误

没有定义任何错误。

另请参阅

insdelln(3X)、 <curses.h>。

历史变更记录

在 X/Open Curses 第 2 期中首次发布。

X/Open Curses 第 **4** 期

为清楚起见，重新编写了该条目。将 **insertln()** 函数的参数列表显式声明为 **void**。

名称

insnstr、insstr、mvinsnstr、mvinsstr、mvwinsnstr、mvwinsstr、winsnstr、winsstr — 将多字节字符型字符串插入窗口

概要

#include <curses.h>

int insnstr(const char *_str_, int _n_);

int insstr(const char *_str_);

int mvinsnstr(int _y_, int _x_, const char *_str_, int _n_);

int mvinsstr(int _y_, int _x_, const char *_str_);

int mvwinsnstr(WINDOW *_win_, int _y_, int _x_, const char *_str_, int _n_);

int mvwinsstr(WINDOW *_win_, int _y_, int _x_, const char *_str_);

int winsnstr(WINDOW *_win_, const char *_str_, int _n_);

int winsstr(WINDOW *_win_, const char *_str_);

说明

这些函数将在当前窗口或所指定窗口的当前位置或所指定位置之前插入一个字符串（按行上最多能放置的字符计）。

这些函数不会使光标位置向前移动。这些函数执行特殊字符处理操作。 **innstr()** 和 **innwstr()** 函数执行环绕。**instr()** 和 **inswstr()** 函数不执行环绕。

insnstr() 、 **mvinsnstr()** 、 **mvwinsnstr()** 和 **winsnstr()** 函数可插入最多 *n* 字节。如果 *n* 小于 1，则插入整个字符串。

返回值

成功完成后，这些函数将返回 OK。否则，它们将返回 ERR。

错误

未定义错误。

实际应用信息

由于字符串中可能包含多字节字符，因此在字符所占用的列位置数与字符串的字节数之间可能没有一一对应的关系。

另请参阅

<curses.h>。

历史变更记录

第一版发行在 X/Open Curses 第 4 期中。

名称

ins_nwstr、ins_wstr、mvins_nwstr、mvins_wstr、mvwins_nwstr、mvwins_wstr、 wins_nwstr、wins_wstr — 将宽字符的字符串插入窗口

概要

#include <curses.h>

int ins_nwstr(const wchar_t *_wstr_, int _n_);

int ins_wstr(const wchar_t *_wstr_);

int mvins_nwstr(int _y_, int _x_, const wchar_t *_wstr_, int _n_);

int mvins_wstr(int _y_, int _x_, const wchar_t *_wstr_);

int mvwins_nwstr(WINDOW *_win_, int _y_, int _x_, const wchar_t *_wstr_, int _n_);

int mvwins_wstr(WINDOW *_win_, int _y_, int _x_, const wchar_t *_wstr_);

int wins_nwstr(WINDOW *_win_, const wchar_t *_wstr_, int _n_);

int wins_wstr(WINDOW *_win_, const wchar_t *_wstr_);

说明

这些函数将 **wchar_t** 字符串（按一行中最多能放下的 **wchar_t** 字符计）立即插入到当前窗口或指定窗口内的当前位置之前或指定位置之前。

字符串中的任何非占位字符都与非占位字符前的字符串中的第一个占位字符关联。如果字符串中的第一个字符为非占位字符，这些函数就会失败。

这些函数不执行换行操作。这些函数不会使光标位置向前移动。这些函数执行特殊字符处理操作。

ins_nwstr() 、 **mvins_nwstr()** 、 **mvwins_nwstr()** 和 **wins_nwstr()** 函数最多可插入 *n* 个 **wchar_t** 字符。如果 *n* 小于 1，则插入整个字符串。

返回值

成功完成后，这些函数将返回 OK。否则，它们将返回 ERR。

错误

未定义错误。

另请参阅

<curses.h>。

历史变更记录

第一版发行在 X/Open Curses 第 4 期中。

名称

insque()、remque() - 在队列中插入或删除元素

概要

#include <search.h>

void insque(void *element, void *pred);

void remque(void *element);

说明

insque() 和 **remque()** 函数可以操作在双链接列表中构建的队列。使用这些函数的应用程序必须定义这样一个结构：结构的前两个成员是指向同一类型结构的指针。结构的其他附加成员则是面向具体的应用程序。结构的前两个成员将用于向前和向后的指针。结构的名称及指针的名称不受任何限制。

insque() 函数将在队列中由 *pred* 参数所指向的对象后面插入 *element* 参数所指向的对象。

remque() 函数则从队列中删除 *element* 参数所指向的对象。

作者

insque() 和 **remque()** 由 HP 和加州大学伯克利分校联合开发。

另请参阅

thread_safety(5)。

符合的标准

insque(): XPG4.2

remque(): XPG4.2

名称

ins_wch、mvins_wch、mvwins_wch、wins_wch — 向窗口中插入复合字符和呈现方式

概要

#include <curses.h>

int ins_wch(const cchar_t ****wch*);**

int wins_wch(WINDOW ****win*, const cchar_t ****wch***);**

int mvins_wch(int *y***, int** *x***, const cchar_t ****wch***);**

int mvwins_wch(WINDOW ****win*, int** *y***, int** *x***, const cchar_t ****wch***);**

说明

这些函数将在当前窗口或所指定窗口的当前光标位置或所指定的光标位置插入复合字符 ***wch*** 及其呈现方式。

这些函数不执行换行操作。这些函数不会使光标位置向前移动。这些函数可执行特殊字符处理，但以下情况除外：如果将换行符插入窗口的最后一行但未启用滚动功能，则行为将不可预测。

返回值

成功完成后，这些函数将返回 OK。否则，它们将返回 ERR。

错误

未定义错误。

实际应用信息

对于非占位字符，可以使用 **add_wch()** 将非占位字符添加到窗口中现有的占位复合字符中。

另请参阅

add_wch(3X)、 <curses.h>。

历史变更记录

在 X/Open Curses 第 4 期中首次发布。

名称

intrflush — 启用或禁用中断时刷新

概要

#include <curses.h>

int intrflush(WINDOW ****win*, bool** *bf***);**

说明

intrflush() 函数将指定按中断键（interrupt、suspend 或 quit）时是否刷新与当前屏幕相关联的输入缓冲区。如果 *bf* 为 TRUE，则按中断键时将刷新此输入缓冲区。如果 *bf* 为 FALSE，则按中断键时将不刷新此输入缓冲区。选项的缺省值是从显示器驱动程序的设置中继承而来。*win* 参数必须指向与当前屏幕相关联的窗口。

返回值

成功完成后， **intrflush()** 将返回 OK。否则，它将返回 ERR。

错误

未定义错误。

实际应用信息

在 Curses 以外，使用《X/Open System Interface Definitions, Issue 4, Version 2》规范 (General Terminal Interface) 中指定的 NOFLSH 本地模式标志可达到相同的效果。

另请参阅

curses_intro(3X)，参阅输入处理小节、 <curses.h>、《X/Open System Interface Definitions, Issue 4, Version 2》规范第 9.2 节： Parameters That Can Be Set 。

历史变更记录

在 X/Open Curses 第 2 期中首次发布。

X/Open Curses 第 **4** 期

为了清楚说明，此条目已经过重新编写。

名称

in_wch、mvin_wch、mvwin_wch 和 win_wch — 从窗口输入组合字符和呈现方式

概要

#include <curses.h>

int in_wch(cchar_t ****wcval*);**

int mvin_wch(int *y***, int** *x***, cchar_t ****wcval***);**

int mvwin_wch(WINDOW ****win*, int** *y***, int** *x***, cchar_t ****wcval***);**

int win_wch(WINDOW ****win*, cchar_t ****wcval***);**

说明

这些函数将当前或指定窗口中的当前或指定位置的组合字符和呈现方式提取到 *wcval* 指向的对象中。

返回值

成功完成后，这些函数返回 OK。否则，返回 ERR。

错误

没有定义任何错误。

另请参阅

<curses.h>。

历史变更记录

在 X/Open Curses 第 4 期中首次发布。

名称

in_wchnstr、in_wchstr、mvin_wchnstr、mvin_wchstr、 mvwin_wchnstr、mvwin_wchstr、win_wchnstr、win_wchstr — 从窗口输入由复合字符及呈现方式组成的数组

概要

#include <curses.h>

int in_wchnstr(cchar_t *_wchstr_, int _n_);

int in_wchstr(cchar_t *_wchstr_);

int mvin_wchnstr(int _y_, int _x_, cchar_t *_wchstr_, int _n_);

int mvin_wchstr(int _y_, int _x_, cchar_t *_wchstr_);

int mvwin_wchnstr(WINDOW *_win_, int _y_, int _x_, cchar_t *_wchstr_, int _n_);

int mvwin_wchstr(WINDOW *_win_, int _y_, int _x_, cchar_t *_wchstr_);

int win_wchnstr(WINDOW *_win_, cchar_t *_wchstr_, int _n_);

int win_wchstr(WINDOW *_win_, cchar_t *_wchstr_);

说明

这些函数将从当前窗口或所指定的窗口中提取字符（起止位置如下：从当前位置或所指定的位置开始，到行尾结束），并将其放入 *wchstr* 所指向的数组中。

in_wchnstr() 、 **mvin_wchnstr()** 、 **mvwin_wchnstr()** 和 **win_wchnstr()** 最多可向数组中填入 *n* 个 **cchar_t** 元素。

返回值

成功完成后，这些函数将返回 OK。否则，它们将返回 ERR。

错误

未定义错误。

实际应用信息

对于 *wchstr* 所指向的数组而言，当用 **in_wchstr()** 、 **mvin_wchstr()** 、 **mvwin_wchstr()** 或 **win_wchstr()** 读取导致其溢出的行时，可产生不可预测的结果。建议分别使用 **in_wchnstr()** 、 **mvin_wchnstr()** 、 **mvwin_wchnstr()** 或 **win_wchnstr()** 。

另请参阅

in_wch(3X)、 <curses.h>。

历史变更记录

在 X/Open Curses 第 4 期中首次发布。

名称

isastream( ) - 确定文件描述符指向的是流设备还是基于流的管道

概要

#include <stropts.h>

int isastream(int fd);

说明

isastream() 函数将测试打开的文件描述符 (*fd*) 是对应流设备还是对应基于流的管道。

返回值

成功完成后，如果 *fd* 所指定的打开文件的文件描述符是流设备或是基于流的管道， **isastream()** 函数将返回值 1；如果此描述符不是流，但属于有效的打开文件描述符，则返回 0（零）。否则，它将返回值 -1 并设置 **errno** 以指示错误。

错误

如果出现以下任何一种情况， **isastream()** 函数就会将 **errno** 设置为相应的值:

[EBADF] 指定的文件描述符未指向有效的打开文件。

另请参阅

fattach(3C)、fdetach(3C)、streamio(7)。

符合的标准

isastream(): SVID3

名称

isendwin — 确定屏幕是否已刷新

概要

```
#include <curses.h>

bool isendwin(void);
```

说明

isendwin() 函数表示自上次调用 **endwin()** 以来屏幕是否已刷新。

返回值

如果已调用 **endwin()** 而随后没有进行任何刷新，则 **isendwin()** 函数返回 TRUE。否则返回 FALSE。

错误

没有定义任何错误。

另请参阅

endwin(3X)、 <curses.h>。

历史变更记录

在 X/Open Curses 第 4 期中首次发布。

名称

isfinite( ) - 浮点有限值宏

概要

#include <math.h>

int isfinite(*floating-type* **x);**

说明

isfinite() 宏可确定其参数是否包含有限值（0、去标准化值、标准化值、非无限值或 NaN）。参数必须是浮点类型，并且分类是基于参数类型的。对于 HP Integrity 服务器，参数可为任何浮点类型。对于 PA-RISC，参数必须是 **double** 或 **float** 。

isfinite() 宏实现 IEEE-754 标准为浮点算术运算所建议的 **finite()** 功能。

用法

要使用 **isfinite()** 宏，请使用缺省的 **-Ae** 选项或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。确保程序包含 **<math.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

返回值

当且仅当参数包含有限值的情况下， **isfinite()** 宏将返回非零值。该宏不会引发浮点异常。

错误

没有定义任何错误。

举例

继续操作前，确保值为有限数：

```
#include <math.h>
/*...*/
  float x;
  /*...*/
  if (isfinite(x))
    /*...*/
```

另请参阅

fpclassify(3M)、 isinf(3M)、 isnan(3M)、 isnormal(3M)、 signbit(3M)、 math(5)。

符合的标准

isfinite()： ISO/IEC C99

名称

isgreater( ) - 浮点静态比较宏 (>)

概要

#include <math.h>

int isgreater(*floating-expr* **x,** *floating-expr* **y);**

说明

isgreater() 宏确定其第一个参数是否大于第二个参数。 **isgreater(***x***,***y***)** 的值始终等于 **(***x***) > (***y***)** ； 但是与 **(***x***) > (***y***)** 不同，当 *x* 和 *y* 无序时， **isgreater(***x***,***y***)** 不引发非法异常。

参数必须是浮点类型。对于 HP Integrity 服务器，参数可为任何浮点类型。对于 PA-RISC，每个参数都必须是 **double** 或 **float** 。

用法

要使用 **isgreater()** 宏，请使用缺省的 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。确保程序包含 **<math.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

返回值

isgreater() 宏返回 **(***x***) > (***y***)** 的值。它不引发任何浮点运算异常。

错误

没有定义任何错误。

另请参阅

isgreaterequal(3M)、 isless(3M)、 islessequal(3M)、 islessgreater(3M)、 isunordered(3M)、 math(5)。

符合的标准

isgreater()： ISO/IEC C99

名称

isgreaterequal( ) - 浮点静态比较宏 (>=)

概要

#include <math.h>

int isgreaterequal(*floating-expr* **x,** *floating-expr* **y);**

说明

isgreaterequal() 宏确定其第一个参数是否大于或等于第二个参数。 **isgreaterequal(***x***,***y***)** 的值始终等于 **(***x***) >= (***y***)** ；但是与 **(***x***) >= (***y***)** 不同，当 *x* 和 *y* 无序时， **isgreaterequal(***x***,***y***)** 不引发非法异常。

参数必须是浮点类型。对于 HP Integrity 服务器，参数可为任何浮点类型。对于 PA-RISC，每个参数都必须是 **double** 或 **float** 。

用法

要使用 **isgreaterequal()** 宏，请使用缺省的 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。确保程序包含 **<math.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

返回值

isgreaterequal() 宏返回 **(***x***) >= (***y***)** 的值。它不引发任何浮点运算异常。

错误

没有定义任何错误。

另请参阅

isgreater(3M)、 isless(3M)、 islessequal(3M)、 islessgreater(3M)、 isunordered(3M)、 math(5)。

符合的标准

isgreaterequal()： ISO/IEC C99

名称

isinf( ) - 浮点无穷大测试

概要

#include <math.h>

int isinf(floating-type x);

说明

isinf() 宏确定其参数是否为无穷大。参数必须是浮点类型，并且分类必须基于该参数的类型。对于 HP Integrity 服务器，参数可为任何浮点类型。对于 PA-RISC，参数必须是 **double** 或 **float** 。

isinf() 宏替换过时的或不再支持的 **isinf()** 和 **isinff()** 函数。

用法

要使用 **isinf()** 宏，请使用缺省的 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。确保程序包含 **<math.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

返回值

如果 *x* 为 ±INFINITY，则 **isinf()** 宏返回非零值。否则返回零。此宏不引发任何浮点运算异常。

错误

没有定义任何错误。

举例

如果 **x** 为无穷大，则执行以下特定操作：

```
#include <math.h>
/*...*/
  float x;
  /*...*/
  if (isinf(x))
    /*...*/
```

另请参阅

fpclassify(3M)、 isfinite(3M)、 isnan(3M)、 isnormal(3M)、 signbit(3M)、 math(5)。

符合的标准

isinf()： ISO/IEC C99

名称

isless( ) - 浮点静态比较宏 (<)

概要

#include <math.h>

int isless(*floating-expr* **x,** *floating-expr* **y);**

说明

isless() 宏确定其第一个参数是否小于第二个参数。

isless(*x***,***y***)** 的值始终等于 **(***x***) < (***y***)** ；但是与 **(***x***) < (***y***)** 不同，当 *x* 和 *y* 无序时， **isless(***x***,***y***)** 不引发非法异常。

参数必须是浮点类型。对于 HP Integrity 服务器，参数可为任何浮点类型。对于 PA-RISC，每个参数都必须是 **double** 或 **float** 。

用法

要使用 **isless()** 宏，请使用缺省的 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。确保程序包含 **<math.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

返回值

isless() 宏返回 **(***x***) < (***y***)** 的值。它不引发任何浮点运算异常。

错误

没有定义任何错误。

另请参阅

isgreater(3M)、 isgreaterequal(3M)、 islessequal(3M)、 islessgreater(3M)、 isunordered(3M)、 math(5)。

符合的标准

isless()： ISO/IEC C99

名称

islessequal( ) - 浮点静态比较宏 (<=)

概要

#include <math.h>

int islessequal(*floating-expr* **x,** *floating-expr* **y);**

说明

islessequal() 宏确定其第一个参数是否小于或等于第二个参数。 **islessequal(***x***,***y***)** 的值始终等于 **(***x***) <= (***y***)** ；但是与 **(***x***) <= (***y***)** 不同，当 *x* 和 *y* 无序时， **islessequal(***x***,***y***)** 不引发非法异常。

参数必须是浮点类型。对于 HP Integrity 服务器，参数可为任何浮点类型。对于 PA-RISC，每个参数都必须是 **double** 或 **float** 。

用法

要使用 **islessequal()** 宏，请使用缺省的 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。确保程序包含 **<math.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

返回值

islessequal() 宏返回 **(***x***) <= (***y***)** 的值。它不引发任何浮点运算异常。

错误

没有定义任何错误。

另请参阅

isgreater(3M)、 isgreaterequal(3M)、 isless(3M)、 islessgreater(3M)、 isunordered(3M)、 math(5)。

符合的标准

islessequal()： ISO/IEC C99

名称

islessgreater( ) - 浮点静态比较宏 (<>)

概要

#include <math.h>

int islessgreater(*floating-expr* **x,** *floating-expr* **y);**

说明

islessgreater() 宏确定其第一个参数是否小于或大于第二个参数，当 *x* 和 *y* 无序时，不引发非法异常。

参数必须是浮点类型。对于 HP Integrity 服务器，参数可为任何浮点类型。对于 PA-RISC，每个参数都必须是 **double** 或 **float** 。

用法

要使用 **islessgreater()** 宏，请使用缺省的 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。确保程序包含 **<math.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

返回值

如果 **(***x***) < (***y***)** 或 **((***x***) > (***y***)** ，则 **islessgreater()** 宏返回值 1，否则，返回零。此宏不引发任何浮点运算异常。

错误

没有定义任何错误。

另请参阅

isgreater(3M)、 isgreaterequal(3M)、 isless(3M)、 islessequal(3M)、 isunordered(3M)、 math(5)。

符合的标准

islessgreater()：ISO/IEC C99

名称

is_linetouched、is_wintouched、touchline、untouchwin、wtouchln — 窗口刷新控制函数

概要

#include <curses.h>

bool is_linetouched(WINDOW *_win_, int _line_);

bool is_wintouched(WINDOW *_win_);

int touchline(WINDOW *_win_, int _start_, int _count_);

int untouchwin(WINDOW *_win_);

int wtouchln(WINDOW *_win_, int _y_, int _n_, int _changed_);

说明

touchline() 函数将仅处理 *count* 行（从 *start* 行开始）。

untouchwin() 函数将标记窗口中所有自上次刷新操作以来未发生更改的行。

如果 *changed* 为 1，则调用 **wtouchln()** 时将处理所指定窗口中的 *n* 行（从第 *y* 行开始）。如果 *changed* 为 0，**wtouchln()** 就会将这些行标记为自上次刷新操作以来未发生更改。

is_wintouched() 函数决定是否处理所指定的窗口。 **is_linetouched()** 函数决定是否处理所指定窗口的第 *line* 行。

返回值

如果自上次刷新操作以来所指定的任何行或任何窗口已被处理， **is_linetouched()** 和 **is_wintouched()** 函数就会返回 TRUE。否则，它们将返回 FALSE。

成功完成后，其他函数将返回 OK。否则，它们将返回 ERR。有关它的异常信息，请参阅前面的函数说明。

错误

未定义错误。

实际应用信息

使用重叠的窗口时，有时需要调用 **touchwin()** 或 **touchline()** ，原因是对一个窗口所做的更改会影响到另一个窗口。但是，有关另一窗口中哪些行进行过更改的记录并不能反映出更改内容。

另请参阅

doupdate(3X)、 touchwin(3X)、 curses_intro(3X)、 Screens, Windows and Terminals 、 <curses.h>。

历史变更记录

第一版发行在 X/Open Curses 第 4 期中。

名称

isnan( ) - 浮点 NaN 测试

概要

#include <math.h>

int isnan(*floating-type* x);

说明

isnan() 宏确定其参数是否为 NaN。参数必须是浮点类型。对于 HP Integrity 服务器，参数可为任何浮点类型。对于 PA-RISC，参数必须是 **double** 或 **float** 。

isnan() 宏实现浮点算术的 IEEE-754 标准建议的 **isnan()** 功能。

isnan() 宏替换过时的或不再支持的 **isnan()** 和 **isnanf()** 函数。

用法

要使用 **isnan()** 宏，请使用缺省的 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。确保程序包含 **<math.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

返回值

当且仅当其参数具有 NaN 值时， **isnan()** 宏才返回非零值。此宏不引发任何浮点运算异常。

错误

没有定义任何错误。

举例

如果 **x** 不是 NaN，则执行以下特定操作：

```
#include <math.h>
/*...*/
  double x;
  /*...*/
  if (!isnan(x))
    /*...*/
```

另请参阅

fpclassify(3M)、 isfinite(3M)、 isinf(3M)、 isnormal(3M)、 signbit(3M)、 math(5)。

符合的标准

isnan()： ISO/IEC C99

名称

isnormal( ) - 浮点规范化值测试

概要

#include <math.h>

int isnormal(*floating-type* **x);**

说明

isnormal() 宏确定其参数是否为规范化值（既不是零、非规范化值和无穷大，也不是 NaN）。参数必须是浮点类型，并且分类必须基于该参数的类型。对于 HP Integrity 服务器，参数可为任何浮点类型。对于 PA-RISC，参数必须是 **double** 或 **float** 。

用法

要使用 **isnormal()** 宏，请使用缺省的 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。确保程序包含 **<math.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

返回值

当且仅当其参数具有规范化值时， **isnormal()** 宏才返回非零值。此宏不引发任何浮点运算异常。

错误

没有定义任何错误。

举例

在继续对值进行操作之前，确保其为规范化值：

```
#include <math.h>
/*...*/
  float x;
  /*...*/
  if (isnormal(x))
    /*...*/
```

另请参阅

fpclassify(3M)、 isfinite(3M)、 isinf(3M)、 isnan(3M)、 signbit(3M)、 math(5)。

符合的标准

isnormal()：ISO/IEC C99

名称

isunordered( ) - 浮点比较宏（无序）

概要

#include <math.h>

int isunordered(*floating-expr* **x,** *floating-expr* **y);**

说明

isunordered() 确定其参数是否是无序的。如果至少有一个参数为 NaN，则其参数就是无序的。

参数必须是浮点类型。对于 HP Integrity 服务器，参数可为任何浮点类型。对于 PA-RISC，每个参数都必须是 **double** 或 **float** 。

用法

要使用 **isunordered()** 宏，请使用缺省的 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。确保程序包含 **<math.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

返回值

如果其参数是无序的（即如果有一个参数为 NaN），则 **isunordered()** 宏返回 1，否则返回 0。此宏不引发任何浮点运算异常。

错误

没有定义任何错误。

另请参阅

isgreater(3M)、 isgreaterequal(3M)、 isless(3M)、 islessequal(3M)、 islessgreater(3M)、 math(5)。

符合的标准

isunordered()：ISO/IEC C99

名称

j0()、j0f()、j1()、j1f()、jn()、jnf() - 第一类贝塞尔函数

概要

```
#include <math.h>

double j0(double x);

double j1(double x);

double jn(int n, double x);
```

仅适用于 **HP Integrity** 服务器

```
float j0f(float x);

float j1f(float x);

float jnf(int n, float x);
```

说明

j0() 和 **j1()** 分别返回命令 0 与 1 的第一类 *x* 的贝塞尔函数。 **jn()** 返回顺序 *n* 的第一类 *x* 的贝塞尔函数。

j0f()、 **j1f()** 和 **jnf()** 是 **j0()**、 **j1()** 和 **jn()** 的 **float** 版本；每一个都采用 **float** 参数 *x* ，并返回 **float** 结果。

用法

要使用这些函数，请使用缺省的 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。确保程序包含 **<math.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

返回值

如果 *x* 为 NaN，则 **j0()**、 **j1()** 和 **jn()** 返回 NaN。

错误

没有定义任何错误。

另请参阅

y0(3M)、 math(5)。

M. Abramowitz 和 I. Stegun 合作编写的《Handbook of Mathematical Functions》（纽约：Dover Publications，1972 年）。

符合的标准

j0()：SVID3 和 XPG4.2

j1()：SVID3 和 XPG4.2

jn()：SVID3 和 XPG4.2

名称

keyname()、key_name() - 获取密钥名

概要

#include <curses.h>

char *keyname(int *c***);**

char *key_name(wchar_t *c***);**

说明

keyname() 和 **key_name()** 函数将生成一个字符串，它的值将对密钥 *c* 进行说明。 **keyname()** 的 *c* 参数可以是一个 8 位字符或是密钥代码。而 **key_name()** 的 *c* 参数则必须是一个宽字符。

字符串将根据下表中第一个适用的行来确定所用的格式：

输入	返回字符串的格式
可见字符	同一字符
控制字符	^*X*
元字符（仅限 **keyname()**）	**M-***X*
<curses.h> 中的密钥值（仅限 **keyname()**）	**KEY_**名称
其他所有情况	**UNKNOWN KEY**

只有在启用元字符的情况下，才能使用上面所示的元字符形式。

返回值

成功完成后， **keyname()** 将返回一个指向字符串的指针（如上所述）。否则，它将返回空指针。

错误

未定义错误。

实际应用信息

keyname() 和 **key_name()** 的返回值可指向一个静态区域：当后面再调用其中的某个函数时，将覆盖此静态区域。

在应用程序处理元字符时，通常不会将它们放到窗口中。如果某个应用程序将元字符放到窗口中并试图按宽字符进行检索， **keyname()** 将检测不到元字符，原因是宽字符不支持元字符。

另请参阅

meta(3X)、 <curses.h>。

变更历史记录

第一版发行在 X/Open Curses 第 4 期中。

名称

键盘 — 启用或禁用功能键的缩写

概要

#include <curses.h>

int keypad(WINDOW ****win*, bool** *bf***);**

说明

keypad() 函数控制键盘转换。如果 *bf* 为 TRUE，则键盘转换打开。如果 *bf* 为 FALSE，则键盘转换关闭。初始状态为 FALSE。

该函数影响任何提供键盘输入的函数的行为。

如果使用中的终端要求用命令启用它，以在按下功能键时传输与众不同的代码，那么在首次启用键盘转换后，实现将此命令传输到该终端（在受影响的输入函数尝试从该终端读取任何字符之前）。

返回值

成功完成后， **keypad()** 返回 OK。否则返回 ERR。

错误

没有定义任何错误。

另请参阅

curses_intro(3X)，参阅键盘处理小节、 <curses.h>。

历史变更记录

在 X/Open Curses 第 2 期中首次发布。

X/Open Curses 第 **4** 期

为清楚起见，重新编写了该条目。

名称

lckpwdf()、 ulckpwdf() - 控制对 /etc/passwd 和 /etc/shadow 文件的访问

概要

#include <shadow.h>

int lckpwdf (void)

int ulckpwdf (void)

说明

lckpwdf() 和 **ulckpwdf()** 例行程序用于调整口令文件 **/etc/passwd** 和影子口令文件 **/etc/shadow** 的修改访问权限。这两个例行程序所用的锁定文件为 **/etc/.pwd.lock** 。进程首先将调用 **lckpwdf()** ，以获取对修改的独占访问权限。完成修改后，将调用 **ulckpwdf()** 以释放 **/etc/.pwd.lock** 上的锁。这种机制可以防止同时对文件进行修改。

返回值

成功完成后， **lckpwdf()** 例行程序将返回 0。如果无法获取锁，它就会返回 –1 并设置 **errno** 以指示错误。

成功完成后， **ulckpwdf()** 例行程序将返回 0。如果已释放锁， **ulckpwdf()** 就会返回 –1 并设置 **errno** 以指示错误。

文件

/etc/passwd
/etc/shadow
/etc/.pwd.lock

另请参阅

getpwent(3C)、 passwd(4)、 thread_safety(5)。

名称

_ldecvt( )、_ldfcvt( )、_ldgcvt( ) - 将长双精度型浮点数转换为字符串

概要

```
#include <stdlib.h>

char *_ldecvt(long_double value, int ndigit, int *decpt, int *sign);

char *_ldgcvt(long_double value, int ndigit, char *buf);

char *_ldfcvt(long_double value, int ndigit, int *decpt, int *sign);
```

过时的接口

```
int _ldecvt_r(
    long_double value,
    int ndigit,
    int *decpt,
    int *sign,
    char *buffer,
    int buflen);

int _ldfcvt_r(
    long_double value,
    int ndigit,
    int *decpt,
    int *sign,
    char *buffer,
    int buflen);
```

说明

_ldecvt() 将 *value* 转换为一个以 null 结尾的 *ndigit* 数字字符串，并返回一个指向该字符串的指针。除非值为零，否则高位数字非零。低位数字舍入。通过 *decpt* （负值表示位于返回数字的左边）间接存储小数分隔符相对于字符串开始的位置。在返回的字符串中不包含小数分隔符。如果结果的符号为负，那么 *sign* 指向的字为非零；否则它为零。

_ldfcvt() 与 **_ldecvt()** 类似，不同之处在于，为 printf *ndigit* 指定的数位的 **%Lf** （FORTRAN F-格式）输出而舍入了正确的数字。

_ldgcvt() 将 *value* 转换为由 *buf* 指向的数组中以 null 结尾的字符串，并返回 *buf* 。如果可能，它生成 *ndigit* FORTRAN F-格式的有效数字，否则生成 E-格式。如果需要，返回字符串可以包含减号和小数分隔符。取消零结尾。小数分隔符由当前加载的 NLS 环境决定（请参阅 *setlocale*(3C) ）。如果没有成功调用 **setlocale()** ，则使用缺省的 NLS 环境“C”（请参阅 *lang*(5) ）。缺省环境指定句点 **(.)** 作为小数分隔符。

过时的接口

_ldecvt_r() 和 **_ldfcvt_r()** 将长双精度型浮点数转换为字符串。

返回值

不是数字时返回 **NaN** ，无穷大时返回 **±INFINITY** 。

警告

由 **_ldecvt()** 和 **_ldfcvt()** 返回的值指向数据，随后由相同的线程调用这些接口来改写其内容。

_ldecvt_r() 和 **_ldfcvt_r()** 是过时接口，只是为了兼容现有的 DCE 应用程序而进行支持。新的多线程应用程序应当使用 **_ldecvt()** 和 **_ldfcvt()** 。

外部语言环境影响

语言环境

LC_NUMERIC 类型决定小数分隔符。

国际代码集支持

支持单字节字符代码集。

作者

_ldecvt() 、 **_ldfcvt()** 和 **_ldgcvt()** 由 HP 开发。

另请参阅

setlocale(3C)、 printf(3S)、 lang(5)、 thread_safety(5)。

名称

ldexp( )、ldexpf( )、ldexpl( )、ldexpw( )、ldexpq( ) - 缩放浮点数的指数

概要

#include <math.h>

double ldexp(double x, int exp);

仅适用于 **HP Integrity** 服务器

float ldexpf(float x, int exp);

long double ldexpl(long double y, int exp);

extended ldexpw(extended x, int exp);

quad ldexpq(quad x, int exp);

说明

ldexp() 函数将计算 $x * 2^{exp}$ 的值。

ldexp() 与 **scalbn()** 计算相同的值。

仅适用于 **Integrity** 服务器

ldexpf() 是 **float** 版的 **ldexp()**；它的第一个参数采用 **float**，并返回 **float** 结果。

ldexpl() 是 **long double** 版的 **ldexp()**；它的第一个参数采用 **long double**，并返回 **long double** 结果。

ldexpw() 是 **extended** 版的 **ldexp()**；它的第一个参数采用 **extended**，并返回 **extended** 结果。

ldexpq() 等效于 HP-UX 系统上的 **ldexpl()**。

用法

（对于 Integrity 服务器）要使用 **ldexpf()**，请使用缺省的 **-Ae** 选项或 **-Aa** 选项进行编译。

（对于 Integrity 服务器）要使用 **ldexpl()**、**ldexpw()** 或 **ldexpq()**，请使用缺省的 **-Ae** 选项或 **-Aa** 和 **-D_HPUX_SOURCE** 选项编译。

（对于 Integrity 服务器）要使用 **ldexpw()** 或 **ldexpq()**，也可以使用 **-fpwidetypes** 选项进行编译。

要使用上面的任何函数，请确保程序包含 **<math.h>**，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

有关详细信息，请参阅《HP-UX floating-point guide for HP Integrity servers》（位于以下站点：**http://www.hp.com/go/fp**）。

返回值

成功完成后，**ldexp()** 函数将返回 $x * 2^{exp}$。

如果 x 为 ±无穷大、0 或 NaN，则 **ldexp()** 返回 x。

ldexp() 将返回带相应符号的无穷大 (等于±HUGE_VAL) 以代替量值太大的值，并引发溢出及不精确异常。

当结果太小（本质上异常或为零），因此会丧失准确性时， **ldexp()** 将引发下溢和不精确异常；如果结果只是太小，即可能引发这些异常。

错误

如果正确的值会溢出， **ldexp()** 则将 **errno** 设置为 [ERANGE]。

仅适用于 **Integrity** 服务器

缺省情况下，Integrity 服务器的 HP-UX **libm** 函数不设置 **errno** 。对于 **errno** 设置，可以使用 **+Olibmerrno** 选项进行编译。

另请参阅

frexp(3M)、 ilogb(3M)、 logb(3M)、 scalb(3M)、 scalbln(3M)、 scalbn(3M)、 math(5)。

符合的标准

ldexp() ： SVID3、XPG4.2、ANSI C、ISO/IEC C99 （包括附件 F，“IEC 60559 floating-point arithmetic”）

ldexpf() 、 **ldexpl()** ： ISO/IEC C99（包括附件 F“IEC 60559 floating-point arithmetic”）

名称

lgamma( )、 lgammaf( )、 lgammal( )、 lgammaw( )、 lgammaq( )、 lgamma_r( )、 lgammaf_r( )、 lgammal_r( )、 lgammaw_r( )、 lgammaq_r( )、 gamma( )、 gammaf( )、 gammal( )、 gammaw( )、 gammaq( )、 signgam - 记录伽玛函数

概要

```
#include <math.h>

double lgamma(double x);

double lgamma_r(double x, int *sign);

double gamma(double x);

extern int signgam;
```

仅适用于 **HP Integrity** 服务器

```
float lgammaf(float x);

long double lgammal(long double x);

extended lgammaw(extended x);

quad lgammaq(quad x);

float lgammaf_r(float x, int *sign);

long double lgammal_r(long double x, int *sign);

extended lgammaw_r(extended x, int *sign);

quad lgammaq_r(quad x, int *sign);

float gammaf(float x);

long double gammal(long double x);

extended gammaw(extended x);

quad gammaq(quad x);
```

说明

lgamma() 和 **gamma()** 可返回 **ln**(|Γ(*x*)|)，其中的 Γ(*x*) 定义为整数，随着 *t* 从零变为无穷大， **exp**(−*t*) 根据幂数 (*x*−1) 乘 *t* 。

在外部整数 **signgam** 中返回 Γ(*x*) 的符号。

可以使用 Integrity 服务器上提供的 **tgamma()** 函数计算 Γ(*x*)。也可以使用以下 C 程序段计算 Γ(*x*)：

```
if ((y = lgamma(x)) > LN_MAXDOUBLE)
    error( );
y = signgam * exp(y);
```

在该程序段中，如果 *y* 大于 **<values.h>** 头文件中定义的 **LN_MAXDOUBLE** ，则 **exp()** 将返回范围错误（请参阅 *exp*(3M) ）。

记录伽玛函数 **lgamma()** 不是重入函数，原因是它使用全局变量 **signgam** 。函数 **lgamma_r()** 是可用于多线程应用程序中的 **lgamma()** 的重入形式。函数 **lgamma_r()** 可存储第二个参数 *sign* 指向的对象中的 Γ(*x*) 的符号。如果 Γ(*x*) 是正数，则 *sign* 指向的值为 +1；如果是负数，则所指向的值为 -1。

gamma() 函数在功能上等效于 **lgamma()** 。

lgammaf() 、 **gammaf()** 和 **lgammaf_r()** 分别是 **lgamma()** 、 **gamma()** 和 **lgamma_r** 的 **float** 形式；这些函数使用 **float** （第一个）参数，并返回一个 **float** 结果。

lgammal() 、 **gammal()** 和 **lgammal_r()** 分别是 **lgamma()** 、 **gamma()** 和 **lgamma_r** 的 **long double** 形式；这些函数使用 **long double** （第一个）参数，并返回一个 **long double** 结果。

lgammaw() 、 **gammaw()** 和 **lgammaw_r()** 分别是 **lgamma()** 、 **gamma()** 和 **lgamma_r()** 的 **extended** 形式；这些函数使用 **extended** （第一个）参数，并返回一个 **extended** 结果。

lgammaq() 、 **gammaq()** 和 **lgammaq_r** 分别等效于 HP-UX 系统上的 **lgammal()** 、 **gammal()** 和 **lgammal_r()** 。

用法

要使用这些函数，请使用缺省的 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。

（对于 Integrity 服务器）要使用 **lgammaw()** 、 **lgammaw_r()** 、 **gammaw()** 、 **lgammaq()** 、 **lgammaq_r()** 或 **gammaq()** ，还应使用 **-fpwidetypes** 选项进行编译。

确保程序包含 **<math.h>** ，并通过在编译程序或链接程序命令行上指定的 **-lm** 链接到数学库。

有关详细信息，请参阅《HP-UX floating-point guide for HP Integrity servers》（位于以下站点：**http://www.hp.com/go/fp** ）。

返回值

lgamma(1) 返回 +0。

lgamma(2) 返回 +0。

如果 x 为负整数或零，则 **lgamma**(x) 将返回 +Inf，同时引发以零为除数的浮点异常。

lgamma(-Inf) 返回 +Inf。

lgamma(+Inf) 返回 +Inf。

如果 *x* 为 NaN，则 **lgamma()** 和 **gamma()** 将返回 NaN。

lgamma() 可返回无穷大（等于 **HUGE_VAL** ）来代替数量级过大的值，并可引发溢出异常和不精确异常。

如果该函数没有引发其他异常，则无法确定 **lgamma()** 是否会引发不精确异常。

警告

在多线程应用程序中使用 **lgamma()** 和 **gamma()** 是不安全的。 **lgamma_r()** 是多线程安全的，因此应改用此函数。

另请参阅

exp(3M)、 log(3M)、 tgamma(3M)、 math(5)。

符合的标准

lgamma()： SVID3、XPG4.2、ISO/IEC C99 （包括附件 F“IEC 60559 floating-point arithmetic”）

gamma()： SVID3、XPG4.2

lgammaf()、 **lgammal()**： ISO/IEC C99（包括附件 F“IEC 60559 floating-point arithmetic”）

名称

libIO : io_block_to_char_dsf() 、 io_block_to_raw() 、 io_char_to_block_dsf() 、 io_dev_to_node() 、 io_dev_to_options() 、 io_end() 、 io_error 、 io_get_devs() 、 io_get_legacy_mode() 、 io_get_mapping() 、 io_get_node_relation()、 io_hw_compare()、 io_hw_compare_ext()、 io_hw_path_to_node()、 io_hw_path_to_str()、 io_init() 、 io_init_hw_path() 、 io_is_hwpath_legacy() 、 io_is_legacy_dev() 、 io_is_legacy_token() 、 io_is_option_set()、 io_legacy_to_new_dev()、 io_legacy_to_new_dsf()、 io_legacy_to_new_hwpath()、 io_mkdev()、 io_mkdev_ext() 、 io_new_to_legacy_devs() 、 io_new_to_legacy_dsfs() 、 io_new_to_legacy_hwpath() 、 io_node_to_hw_path()、 io_query()、 io_query_array()、 io_query_batch()、 io_raw_to_block()、 io_search()、 io_search_array()、 io_search_array_batch()、 io_str_to_hw_path()、 io_strerror()、 - 与内核 I/O 子系统进行交互的接口

概要

#include <sys/libIO.h>

备注

“名称”中的每个函数都在下面各自的“libIO 函数”部分中进行了说明。

说明

libIO 库提供了多个 API，以便通过 **dev_config** 驱动程序访问 I/O 子系统。本联机帮助页介绍了每个 **libIO** API 以及关联的数据结构。

每个 API 下的“错误”一节列出了在 API 发生故障时设置的可能的 **io_errno** 值。在联机帮助页结尾的“LIBIO_ERROR”一节提供了对每个错误值的说明。

要使用 **libIO** 库 **libIO.so** （适用于基于 Itanium(R) 的系统）或 **libIO.sl** （适用于 PA-RISC 系统），程序必须使用 **-lIO**链接到对应的库。

I/O 子系统术语

以下是在本联机帮助页中使用的一些常见的 I/O 子系统术语和结构。

I/O 节点或标记	用于唯一标识 I/O 子系统中的对象的句柄（用 **io_token_t** 表示）。
I/O 树	用于表示一组 I/O 节点的内核数据结构。
io_errno	某个 **libIO** API 发生故障时，用于存储该 **libIO** 特定错误编号的外部变量，其用法类似于 **errno** （请参阅 *errno*(2) ）。
dev_t	请参阅 *mknod*(5) 。
leg_hw_path_t	用于存储 HP-UX 11i v3 之前发行版所采用的硬件路径的数据结构。此结构只有 14 个 **unsigned char** 类型的元素。
hw_path_t	用于存储 HP-UX 11i v3 或更高发行版所采用的硬件路径的数据结构。此结构最多可包含 64 个 **hw_addr_t** 类型的元素。

LibIO API

下面列出了为 HP-UX 11i v3 提供的 **libIO** API。

io_block_to_char_dsf()	将块设备专用文件映射到字符设备专用文件
io_block_to_raw()	返回给定块 **dev_t** 的字符 **dev_t**
io_char_to_block_dsf()	将字符设备专用文件映射到块设备专用文件
io_dev_to_node()	将 **dev_t** 转换为标记
io_dev_to_options()	返回给定 **dev_t** 和 *dev_type* 的设备选项
io_end()	终止与 **dev_config** 驱动程序的连接
io_error()	提供 **libIO** 错误消息
io_get_devs()	返回给定节点的 **dev_t**
io_get_legacy_mode()	返回 Legacy 模式的状态
io_get_mapping()	返回给定节点和其他 I/O 节点之间的映射
io_get_node_relation()	检索给定节点标记的关系
io_hw_compare()	比较 HP-UX 11i v3 之前发行版所用的两个硬件路径结构
io_hw_compare_ext()	比较 HP-UX 11i v3 或更高发行版所用的两个硬件路径结构
io_hw_path_to_node()	将硬件路径转换为标记
io_hw_path_to_str()	将硬件路径转换为字符串
io_init()	建立与 **dev_config** 驱动程序的连接
io_init_hw_path()	使用传递的标记参数初始化硬件路径
io_is_hwpath_legacy()	确定给定硬件路径的格式是 Legacy 还是 Agile
io_is_legacy_dev()	确定给定的 **dev_t** 是否为 Legacy
io_is_legacy_token()	确定标记是 Legacy 还是 Agile
io_is_option_set()	确定是否为 **dev_t** 设置了选项
io_legacy_to_new_dev()	将 Legacy **dev_t** 映射到 Agile **dev_t**
io_legacy_to_new_dsf()	将 Legacy 设备专用文件映射到持久性设备专用文件
io_legacy_to_new_hwpath()	将 HP-UX 11i v3 之前发行版的硬件路径的数据结构转换为 HP-UX 11i v3 或更高发行版的硬件路径的数据结构。
io_mkdev()	如果提供了一个节点和选项字符串，则构建一个 **dev_t**

io_mkdev_ext()	返回给定节点、 *dev_type* 和设备选项的 **dev_t**
io_new_to_legacy_devs()	将一个 Agile **dev_t** 映射到一个或多个 Legacy **dev_t**
io_new_to_legacy_dsfs()	将一个 Agile 设备专用文件映射到一个或多个 Legacy 设备专用文件
io_new_to_legacy_hwpath()	将 HP-UX 11i v3 或更高发行版所用的硬件路径的数据结构转换为 HP-UX 11i v3 之前发行版所用的硬件路径的数据结构。
io_node_to_hw_path()	将标记转换为硬件路径
io_query()	获取有关标记的信息
io_query_array()	获取有关标记的信息（多个字段）
io_query_batch()	创建多个关键字以调用 **io_query_array()**
io_raw_to_block()	返回给定字符 **dev_t** 的块 **dev_t**
io_search()	搜索 I/O 子系统数据结构
io_search_array()	搜索 I/O 系统数据结构
io_search_array_batch()	检索与搜索条件匹配的标记的数组
io_str_to_hw_path()	将表示硬件路径的字符串转换为 **hw_path_t** 结构
io_strerror()	提供 **libIO** 错误消息

这些 API 将在下面各节中进行详细说明。

libIO 函数

名称

io_init() 、 **io_end()** - 建立/终止与 **dev_config** 驱动程序的连接

概要

int io_init(int flag);

void io_end();

说明

libIO 库使用 **dev_config** 驱动程序访问内核 I/O 数据结构中的信息。

io_init() 打开 **/dev/config** 设备专用文件，该操作会对 **dev_config** 驱动程序执行 *open*(2) 。

必须在调用 **libIO** 库中的任何其他例行程序之前调用 **io_init()** 。

io_end() 对 **dev_config** 驱动程序执行 *close*(2) 。

必须在使用 **libIO** 库例行程序之后调用 **io_end()** 。

请注意，每个 **io_init()** 调用应与一个 **io_end()** 调用成对进行。

参数

flag 指定要传递给对 **/dev/config** 设备专用文件执行的 *open*(2) 操作的标记。

定义的值包括：

O_RDONLY
以只读模式打开。

O_WRONLY
以只写模式打开。

O_RDWR 以读写模式打开。

举例

```
if (io_init(O_RDWR) == -1) exit(1);

... 用户代码 ...

io_end();

exit(0);
```

返回值

io_init() ：

IO_SUCCESS - 成功。

IO_ERROR - 失败。设置 **io_errno** 以指明错误。

io_end() 不返回值。

错误

[IO_E_DCONF_NOEXIST]

[IO_E_OPEN_FLAG]

[IO_E_SYSCALL]

libIO 函数

名称

io_error() 、 **io_strerror()** 、 **io_errno - libIO** 错误消息

概要

void io_error(char *str);

char *io_strerror(int errnum);

extern int io_errno;

说明

io_error() 将一个错误消息写入标准错误 (**stderr**) ，说明在调用 **libIO** API 期间遇到的上一个错误。参数 *str* 通常包含调用该 **libIO** API 并引发错误的程序（或函数）的名称。外部变量 *io_errno* 包含由 **libIO** API 导致的上一个错误（如果有）的错误值。应该在发生 API 故障后立即检查该变量的值，它不会被任何其他 **libIO** API 重置。

有关对每个错误值的说明，请参阅下面的“LIBIO_ERROR”一节。

输出的消息的格式如下:

*str***:***libIO error message* **<cr>**

如果 *str* 为 NULL，则消息的格式为:

libIO error message **<cr>**

io_strerror() 返回指向含有 **libIO** 错误消息的字符串的指针，该错误消息字符串映射到已传递的参数 *errnum*。

参数

str 由调用方指定的随错误状态一起输出的字符串

errnum 由调用方指定的映射到错误消息字符串的错误编号

返回值

io_strerror() :

指向消息字符串的指针 - 成功。

NULL - 失败。

io_error() 不返回值。

错误

无。

libIO 函数

名称

io_dev_to_node() - 将 **dev_t** 转换为标记

概要

#include <sys/types.h>

io_token_t io_dev_to_node(dev_t dev,int dev_type);

说明

io_dev_to_node() 返回与指定的 **dev_t** *dev_type* 对应的标记。请参阅 *mknod*(5) 。

参数

dev 映射到 *node* 的 **dev_t** 。

dev_type 可能的值:

D_CHR 字符 **dev_t** 。
D_BLK 块 **dev_t** 。

返回值

标记 - 成功。
NULL - 失败。设置 **io_errno** 以指明错误。

错误

[IO_E_DCONF_OPEN]
[IO_E_PARM]
[IO_E_DCONF_ACCESS]
[IO_E_NODE_NOEXIST]
[IO_E_SYSCALL]

libIO 函数

名称

io_hw_compare() - 比较 HP-UX 11i v3 之前发行版所采用的两个硬件路径结构

概要

int io_hw_compare(leg_hw_path_t *path1, leg_hw_path_t *path2);

说明

io_hw_compare() 比较两个 **leg_hw_path_t** 结构的内容。如果发现它们相等，则返回成功；否则，返回错误。
leg_hw_path_t 是 HP-UX 11i v3 之前发行版用于存储硬件路径的结构。

参数

path1、 *path2* 指向 **leg_hw_path_t** 结构的指针。

返回值

IO_SUCCESS - 相等。
IO_ERROR - 不相等。

错误

无。

libIO 函数

名称

io_hw_compare_ext() - 比较 HP-UX 11i v3 或更高发行版所采用的硬件路径结构

概要

int io_hw_compare_ext(hw_path_t *path1, hw_path_t *path2);

说明

io_hw_compare_ext() 比较两个 **hw_path_t** 结构的内容。如果发现它们相等，则返回成功；否则，返回错误。**hw_path_t** 是 HP-UX 11i v3 或更高发行版用于存储硬件路径的结构。

参数

path1、 *path2* 指向 **hw_path_t** 结构的指针。

返回值

IO_SUCCESS - 相等。
IO_ERROR - 不相等。

错误

[IO_E_DCONF_ACCESS]
[IO_E_DCONF_OPEN]
[IO_E_NODE_NOEXIST]
[IO_E_PARM]
[IO_E_SYSCALL]

libIO 函数

名称

io_hw_path_to_node() - 将硬件路径转换为标记

概要

io_token_t io_hw_path_to_node(hw_path_t *hw_path);

说明

io_hw_path_to_node() 返回指定的硬件路径的标记。

参数

hw_path 指向硬件路径的指针。

返回值

标记 - 成功。
NULL - 失败。设置 **io_errno** 以指明错误。

错误

[IO_E_DCONF_ACCESS]
[IO_E_DCONF_OPEN]
[IO_E_NODE_NOEXIST]
[IO_E_PARM]
[IO_E_SYSCALL]

libIO 函数

名称

io_hw_path_to_str() - 将硬件路径转换为字符串

概要

int io_hw_path_to_str(char *str,hw_path_t *hw_path);

说明

io_hw_path_to_str() 将由 *hw_path* 指定的硬件路径转换为 *str* 指向的字符串。 *str* 必须具有足够的空间（最大硬件路径长度 **MAX_HW_PATH_STR** ）来存储转换后的字符串。如果 *str* 为 NULL，则不传输数据，而是返回字符串的长度。

参数

str 指向表示硬件路径的字符串的指针。

hw_path 输入硬件路径结构。

举例

以下示例输出由 **my_node** 表示的 I/O 树节点的硬件路径:

```
hw_path_t hw_path;
char my_string[MAX_HW_PATH_STR];
if (io_init(O_RDWR) == -1) exit(1);
io_node_to_hw_path(my_node,&hw_path);
io_hw_path_to_str(my_string,&hw_path);
printf("%s", my_string);
io_end();
exit(0);
```

返回值

字符串的字节数 - 成功。
IO_ERROR - 失败。设置 **io_errno** 以指明错误。

错误

[IO_E_DCONF_ACCESS]
[IO_E_DCONF_OPEN]
[IO_E_PARM]
[IO_E_SYSCALL]

libIO 函数

名称

io_str_to_hw_path() - 将表示硬件路径的字符串转换为 **hw_path_t** 结构

概要

int io_str_to_hw_path(char *str,hw_path_t *hw_path);

说明

如果提供了表示传递的路径参数的字符串， **io_str_to_hw_path()** 将使用该硬件路径填充 *hw_path* 。

参数

str	指向表示硬件路径的字符串的指针。
hw_path	硬件路径。

举例

```
char *str_path="4/16.4.0"

hw_path_t hw_path;

if (io_init(O_RDWR) == -1) exit(1);

io_str_to_hw_path(str_path, &hw_path);

io_end();

exit(0);
```

返回值

IO_SUCCESS - 成功。
IO_ERROR - 失败。设置 **io_errno** 以指明错误。

错误

[IO_E_DCONF_OPEN]
[IO_E_PARM]
[IO_E_PATH_STR]
[IO_E_SYSCALL]

libIO 函数

名称

io_is_option_set() - 确定是否为 **dev_t** 设置了选项

概要

int io_is_option_set(dev_t dev, int dev_type, char *options);

说明

io_is_option_set() 调用 **io_mkdev()** （无论是否带选项）均生成选项掩码。然后，此掩码与 *dev* 参数进行按位“AND”运算，并返回其布尔型结果。此例行程序不区分一个或多个选项设置或多位选项。

参数

dev 用于检查选项设置的 **dev_t** 。

dev_type 可能的值：

D_CHR 字符 **dev_t** 。
D_BLK 块 **dev_t** 。

options 用于指明设备专用选项的字符串（以 NULL 结尾）。

返回值

1 - 设置了一个或多个指定的选项。
IO_ERROR - 失败。

错误

无。

libIO 函数

名称

io_mkdev() - 如果提供了一个节点和选项字符串，则构建一个 **dev_t**

概要

int io_mkdev(io_token_t node,int dev_type,char *options,dev_t *dev);

说明

io_mkdev() 通过给定节点和选项构建一个由 *dev* 指向的 **dev_t** 。

参数

node 与某个 I/O 树节点对应的标记。

dev_type 可能的值：

D_CHR 字符 **dev_t** 。
D_BLK 块 **dev_t** 。

dev 指向 **dev_t** 的指针。

返回值

IO_SUCCESS - 成功。
IO_ERROR - 失败。设置 **io_errno** 以指明错误。

错误

[IO_E_DCONF_ACCESS]
[IO_E_DCONF_OPEN]
[IO_E_NODE_NOEXIST]
[IO_E_PARM]
[IO_E_SYSCALL]

libIO 函数

名称

io_node_to_hw_path() - 将标记转换为硬件路径

概要

int io_node_to_hw_path(io_token_t node,hw_path_t *hw_path);

说明

io_node_to_hw_path() 使用给定节点的硬件路径填充 *hw_path* 。

参数

node 与 I/O 节点对应的标记。

hw_path 节点的硬件路径。

返回值

IO_SUCCESS - 成功。
IO_ERROR - 失败。设置 **io_errno** 以指明错误。

错误

[IO_E_DCONF_ACCESS]
[IO_E_DCONF_OPEN]
[IO_E_NODE_NOEXIST]
[IO_E_SYSCALL]

libIO 函数

名称

io_query() - 获取有关标记的信息

概要

int io_query(io_token_t token,int type,char *key,void *ptr);

说明

io_query() 用于获取以前通过调用 **io_search()** 获得的标记的相关信息。与关键字对应的数据被复制到 *ptr* 所指向的缓冲区中。 关键字必须是 I/O 子系统可识别的字符串之一。请注意，并非所有的关键字都是为所有查询定义

的。如果 *ptr* 为 NULL，则 **io_query()** 仅返回已复制的字节数。请参阅 **io_search()** 中的示例。

用户应确保 *ptr* 所指向的缓冲区足够大，以包含请求的信息。

参数

token 对 **io_search()** 调用返回的标记。

type 指定查询的类型。

只能指定一个类型。定义的值包括：

S_IOTREE	查询 I/O 树。
S_IOTREE_EXT	查询 Agile I/O 树。
S_BDEVSW	查询块设备交换表。
S_CDEVSW	查询字符设备交换表。

key 与查询字段对应的字符串。

ptr 指向缩查询数据的副本的指针。

返回值

已复制的字节数 - 成功。

IO_ERROR - 失败。设置 io_errno 以指明错误。

错误

[IO_E_DCONF_ACCESS]
[IO_E_DCONF_OPEN]
[IO_E_KEY_TOKEN_DEF]
[IO_E_PARM]
[IO_E_SYSCALL]

libIO 函数

名称

io_query_batch() - 创建多个关键字以调用 **io_query_array()**

概要

```
int io_query_batch(io_token_t token, int type, char *key1,
        void *dat1,char *key2, void *dat2, ...,
        char *keyN,void *datN,NULL);
```

说明

此例行程序将一对或多对 key/dat 参数复制到数组，并使用传递的搜索参数调用 **io_query_array()** 。

参数

token 由先前对 **io_search()** 的调用返回的标记。

type 指定类型（请参阅 **io_query()** ）。

key 查询字段字符串。

dat 查询字段数据。

返回值

请参阅 **io_query_array()** 的“返回值”一节。

错误

请参阅 **io_query_array()** 的“错误”一节。

libIO 函数

名称

io_query_array() - 获取有关标记的信息（多个字段）

概要

int io_query_array(io_token_t token, int type,
int nkeys, char *key[], void *ptr[]);

说明

此例行程序是对 **io_query()** 的改进。它可针对一个标记一次返回多个字段。

参数

token 由先前对 **io_search()** 的调用返回的标记。

type 查询的类型（请参阅 **io_query()** ）。

nkeys 查询中关键字的数目。

key[ ] 查询字段字符串数组。

ptr[ ] 相应的查询字段数据数组。

返回值

IO_SUCCESS - 成功。
IO_ERROR - 失败。设置 io_errno 以指明错误。

错误

[IO_E_DCONF_OPEN]
[IO_E_DCONF_ACCESS]
[IO_E_PARM]
[IO_E_SYSCALL]

libIO 函数

名称

io_search()、 **io_search_array()** - 搜索 I/O 子系统和系统数据结构

概要

io_token_t io_search(io_token_t token, int type, int qual,
char *key1, void *dat1, char *key2, void *dat2,...,
char *keyN, void *datN,NULL);

io_token_t io_search_array(io_token_t token, int type, int qual,
char *key[], void *dat[]);

说明

可使用此函数搜索 I/O 子系统信息。返回的标记对应于 I/O 树的一个条目，或者可在 type 中指定的一个表。标记可与 **io_query()** 命令一起使用来获取有关标记的信息。

参数

token 由先前对 **io_search()** 的调用返回的标记，或者为 NULL（如果是首次搜索）。

type 指定搜索的类型（请参阅 **io_query()** ）。

qual 搜索的限定符。可以通过对 *qual* 中的值按位“OR” 运算来指定一个或多个限定符。定义的值包括：

Q_SW 与 I/O 树节点关联的软件。

Q_HW 与 I/O 树节点关联的硬件。

Q_PSEUDO 是伪设备驱动程序。

Q_DEVSW 具有交换表入口点。

Q_SUBTREE 在进行 I/O 树搜索时，返回匹配节点下的整个子树。

Q_NEW 新发现的硬件。

Q_TRANS 允许返回透明节点。

Q_CONSOLE 匹配控制台设备。

Q_DUMP 匹配转储设备。

Q_BOOT 匹配引导设备。

Q_ROOT 匹配根文件系统设备。

Q_SWAP 匹配交换设备。

Q_SAVE_CONF 在 ioconfig 文件中保存的 I/O 树节点信息（请参阅 **ioconfig(4)** ）。

key 与已定义的查询字段对应的字符串。对于 **io_search_array()**，它是一个字符串数组（以 NULL 结尾）。

IO_MAX_SEARCH_KEYS 是最大数组大小。

dat 指向与 key 中指明的查询字段相匹配的数据的指针。必须为每个列出的 key 指定一个 dat。**IO_MAX_DATA_SIZE** 是最大数据大小。

返回值

token - 成功。
NULL - 失败。设置 **io_errno** 以指明错误。

错误

[IO_E_DCONF_ACCESS]
[IO_E_DCONF_OPEN]
[IO_E_KEY_TOKEN_DEF]
[IO_E_MATCH]
[IO_E_MEM_ALLOC]
[IO_E_PARM]
[IO_E_SYSCALL]

libIO 函数

名称

io_search_array_batch() - 检索与搜索条件匹配的标记的数组

概要

io_search_array_batch (io_token_t token, int type, int qual,
char *key[],void *dat[],io_token_t *ret_token);

说明

此例行程序是对 **io_search_array()** 的改进，它返回一个标记数组。

参数

token 由先前对 **io_search()**的调用返回的标记，或者为 NULL（如果是首次搜索）。

type 指定搜索的类型（请参阅 **io_query()** ）。

qual 搜索的限定符（请参阅 **io_search()** ）。

key[] 查询字段字符串数组。

dat[] 相应的查询字段数据数组。

ret_roken[] 返回的标记数组。

返回值

IO_SUCCESS - 成功。
IO_ERROR - 失败。设置 io_errno 以指明错误。

错误

[IO_E_DCONF_ACCESS]
[IO_E_DCONF_OPEN]

[IO_E_NODE_PARM]
[IO_E_SYSCALL]

libIO 函数

名称

io_mkdev_ext() - 返回给定节点、 *dev_type* 和设备选项的 **dev_t**

概要

int io_mkdev_ext(io_token_t node, int dev_type, uint64_t options,
dev_t *dev);

说明

io_mkdev_ext() 通过给定节点、设备类型和选项构建一个 **dev_t** 。此 API 仅用于为 Agile 节点创建 **dev_t** 。如果对 Legacy 节点调用，则 **io_errno** 将设置为 **IO_E_FUNC_NOT_SUPPORTED** 。

参数

node 与 I/O 节点对应的标记。

dev_type 可能的值:

D_CHR 字符 **dev_t** 。
D_BLK 块 **dev_t** 。

options 设备专用选项。

dev 创建的 **dev_t** 存储在 *dev* 所指向的内存中。

返回值

IO_SUCCESS - 成功。
IO_ERROR - 失败。设置 io_errno 以指明错误。

错误

[IO_E_DCONF_OPEN]
[IO_E_PARM]
[IO_E_DCONF_ACCESS]
[IO_E_SYSCALL]
[IO_E_NODE_NOEXIST]
[IO_E_FUNC_NOT_SUPPORTED]

libIO 函数

名称

io_dev_to_options() - 返回给定 **dev_t** 和 *dev_type* 的设备选项

概要

int io_dev_to_options(dev_t dev, int dev_type, uint64_t *options);

说明

如果提供了 **dev_t** 和 *dev_type* ， **io_dev_to_options()** 将以 **uint64_t** 位掩码格式返回其设备选项。

参数

dev 用于检索设备选项的 **dev_t** 值。

dev_type 可能的值:

D_CHR 字符 **dev_t** 。
D_BLK 块 **dev_t** 。

options 表示设备选项的无符号 64 位掩码存储在 options 所指向的位置中。

返回值

IO_SUCCESS - 成功。
IO_ERROR - 失败。设置 io_errno 以指明错误。

错误

[IO_E_DCONF_OPEN]
[IO_E_PARM]
[IO_E_DCONF_ACCESS]
[IO_E_SYSCALL]

libIO 函数

名称

io_is_legacy_dev() - 确定给定的 **dev_t** 是否为 Legacy

概要

int io_is_legacy_dev(dev_t dev, int dev_type);

说明

该 API 确定指定 **dev_t** 为 Legacy 还是 Agile **dev_t** 。在查看指定设备专用文件为持久性或旧的文件时，请在该设备专用文件上执行 **stat()** 调用，并向该 API 传递 **st_rdev** （请参阅 *stat*(2) ）。如果 API 返回 **IO_TOKEN_NEW**，则该设备专用文件属于持久性类型；如果 API 返回 **IO_TOKEN_LEGACY** ，该文件则属于旧类型。

参数

dev I/O 节点的 **dev_t** 。

dev_type 可能的值:

D_CHR 字符 **dev_t** 。
D_BLK 块 **dev_t** 。

返回值

IO_TOKEN_NEW - **dev_t** 是 Agile I/O 节点。

IO_TOKEN_LEGACY - **dev_t** 是 Legacy I/O 节点。
IO_ERROR - 失败。设置 io_errno 以指明错误。

错误

[IO_E_DCONF_OPEN]
[IO_E_PARM]
[IO_E_DCONF_ACCESS]
[IO_E_SYSCALL]

libIO 函数

名称

io_is_legacy_token() - 确定标记是 Legacy 还是 Agile

概要

int io_is_legacy_token(io_token_t token);

说明

此 API 接受一个标记，并确定该标记的类型是 Agile 还是 Legacy。

参数

token I/O 节点标记。

返回值

IO_TOKEN_NEW - **dev_t** 是 Agile 节点。
IO_TOKEN_LEGACY - **dev_t** 是 Legacy 节点。
IO_ERROR - 失败。设置 io_errno 以指明错误。

错误

[IO_E_DCONF_OPEN]
[IO_E_PARM]
[IO_E_DCONF_ACCESS]
[IO_E_SYSCALL]

libIO 函数

名称

io_get_legacy_mode() - 返回 Legacy 模式的状态

概要

int io_get_legacy_mode (unsigned int *mode)

说明

调用该 API 以检索 Legacy 模式的状态。参数 模式中的值将指定该 Legacy 模式当前是启用还是禁用（请参阅 *rmsf*(1M) 和 *insf*(1M) ）。

参数

mode 指向无符号整数的指针。

1 启用 Legacy 模式时设置的值。
0 禁用 Legacy 模式时设置的值。

返回值

IO_SUCCESS - 成功。
IO_ERROR - 失败。设置 io_errno 以指明错误。

错误

[IO_E_DCONF_ACCESS]
[IO_E_DCONF_OPEN]
[IO_E_FUNC_NOT_SUPPORTED]

libIO 函数

名称

io_get_mapping() - 返回给定节点和其他 I/O 节点之间的映射

概要

int io_get_mapping(io_token_t node, io_map_type_t map_type,
int *count, io_token_t *token_arry);

说明

调用此 API 可检索映射到给定节点标记的一个或多个节点。

如果给定了一个 *map_type* 为 **IO_LEGACY_2_NEW** 的 Legacy 标记，此例行程序将返回一个映射到该标记的 Agile 标记。

如果给定了一个 *map_type* 为 **IO_NEW_2_LEGACY** 的 Agile 标记，此例行程序将返回一个或多个 Legacy 标记。

如果给定了一个 *map_type* 为 **IO_LUN_2_LUNPATH** 的 LUN 标记，此例行程序将返回一个或多个映射到该标记的 lunpath 标记。

同样地，如果给定了一个 lunpath 标记且 *map_type* 为 **IO_LUNPATH_2_LUN** ，则返回一个 LUN 标记。

参数

node 返回映射所指向节点标记。

map_type 可能的值：

IO_LEGACY_2_NEW
从给定 Legacy 标记获得 Agile 节点标记。这是一对一映射。

IO_NEW_2_LEGACY
从给定 Agile 节点标记获得 Legacy 节点标记。这可以是一对多映射。

IO_LUN_2_LUNPATH

从给定 lun 节点标记获得 lunpath 标记。这可以是一对多映射。

IO_LUNPATH_2_LUN

从给定 lunpath 标记获得 lun 节点标记。这是一对一映射。

token_arry 可能的值：

INPUT:

指向 **io_token_t** 数组的指针。建议的大小为：

(sizeof(io_token_t)*10)

OUTPUT:

将返回的 *io_token_t* 复制到此数组。

count 可能的值：

INPUT:

在 *token_array* 中返回的标记的数目。

对于初始计数，建议使用 10。

OUTPUT:

返回的 *token_array* 中的实际 *io_token_t* 的数目。如果数组不足够大，则：

1) 不返回标记。
2) 将计数设置为要返回的 *io_token_t* 的实际计数。
3) API 返回 **IO_ERROR** 。
4) 将 **io_errno** 设置为 **IO_E_BUF_TOO_SMALL**。

   在这种情况下，调用方需要利用返回的计数值并使用：

   (sizeof(io_token_t)*count)

   来重新计算所需要的缓冲区，然后分配更大的缓冲区，并且需要使用该新缓冲区和计数值来重新调用此 API。

返回值

IO_SUCCESS - 成功。
IO_ERROR - 失败。设置 **io_errno** 以指明错误。

错误

[IO_E_DCONF_OPEN]
[IO_E_PARM]
[IO_E_DCONF_ACCESS]
[IO_E_NODE_NOEXIST]

[IO_E_SYSCALL]
[IO_E_BUF_TOO_SMALL]
[IO_E_MEM_ALLOC]

libIO 函数

名称

io_is_hwpath_legacy() - 确定给定硬件路径是 Legacy 还是 Agile

概要

int io_is_hwpath_legacy(char *path);

说明

此 API 接受一个字符串格式的硬件路径，并确定它是 Legacy 路径还是 Agile 硬件路径。

参数

path 指向表示硬件路径的字符串的指针。

返回值

IO_HWPATH_NEW - 路径是一个 Agile 路径。
IO_HWPATH_LEGACY - 路径是一个 Legacy 路径。
IO_ERROR - 失败。设置 io_errno 以指明错误。

错误

[IO_E_DCONF_OPEN]
[IO_E_PARM]
[IO_E_SYSCALL]
[IO_E_PATH_STR]
[IO_E_NODE_NOEXIST]

libIO 函数

名称

io_get_node_relation() - 检索给定节点标记的关系

概要

int io_get_node_relation(io_token_t node, int relationship,
uint64_t addr, io_token_t *relative);

说明

调用此 API 可获得由 node 指定的 I/O 节点的关系。参数 relationship 指明关系的类型。有效值包括：

- **IO_REL_CHILD**
- **IO_REL_PARENT**
- **IO_REL_SIBLING**

关系的节点标记在参数 *relative* 中返回。如果指定了 **IO_REL_CHILD** 或 **IO_REL_SIBLING** ，则可使用参数 *addr* 和 *relative* 指定子节点或同级节点。

如果参数 *relative* 为 NULL，则可使用 *addr* 标识某个特定的子节点或同级节点。

如果参数 *relative* 不为 NULL，则忽略 *addr* ，此外，如果 *relative* == node，则返回第一个子节点或同级节点，否则，返回由 relative 标识的节点的下一个子节点或同级节点。

参数

node I/O 节点的标记。

relationship 要查找的关系：

IO_REL_CHILD 查找子节点。
IO_REL_PARENT 查找父节点。
IO_REL_SIBLING 查找同级节点。

addr 子节点或同级节点（相对于父节点）的 relative 的地址（可选）。

relative NULL 或找到的上一个关系的句柄。

请注意，NULL 表示关系指针指向 NULL。它并不指 NULL 指针。

返回值

IO_SUCCESS - 成功。
IO_ERROR - 失败。设置 io_errno 以指明错误。

错误

[IO_E_DCONF_OPEN]
[IO_E_PARM]
[IO_E_SYSCALL]

libIO 函数

名称

io_legacy_to_new_hwpath() - 将 HP-UX 11i v3 之前发行版的硬件路径的数据结构转换为 HP-UX 11i v3 或更高发行版的硬件路径的数据结构。

概要

int io_legacy_to_new_hwpath (leg_hw_path_t *from, hw_path_t *to);

说明

io_legacy_to_new_hwpath() 将 HP-UX 11i v3 之前发行版的硬件路径 (**leg_hw_path_t**) 的数据结构转换为 HP-UX 11i v3 或更高发行版的硬件路径 (**hw_path_t**)的数据结构。

参数

from 指向 **leg_hw_path_t** 数据结构的指针。

to 指向 **hw_path_t** 数据结构的指针。

返回值

IO_SUCCESS - 成功。
IO_ERROR - 失败。设置 io_errno 以指明错误。

错误

[IO_E_PARM]

libIO 函数

名称

io_new_to_legacy_hwpath() - 将 HP-UX 11i v3 或更高发行版的硬件路径的数据结构转换为 HP-UX 11i v3 之前发行版所用的硬件路径的数据结构

概要

int io_new_to_legacy_hwpath (hw_path_t *from, leg_hw_path_t *to);

说明

io_new_to_legacy_hwpath() 将 HP-UX 11i v3 或更高发行版的硬件路径 (**hw_path_t**) 的数据结构转换为 HP-UX 11i v3 之前发行版的硬件路径 (**leg_hw_path_t**) 的数据结构。如果 **hw_path_t** 包含的元素多于 14 个或者任一元素的长度大于 8 位，则此 API 会将 **io_errno** 设置为 **IO_E_PARM** 。

参数

from 指向 **hw_path_t** 数据结构的指针。

to 指向 Legacy **leg_hw_path_t** 数据结构的指针。

返回值

IO_SUCCESS - 成功。
IO_ERROR - 失败。设置 io_errno 以指明错误。

错误

[IO_E_PARM]

libIO 函数

名称

io_get_devs() - 返回给定节点的 **dev_t**

概要

int io_get_devs(io_token_t node, io_dev_info_t *dev_arry, int *count);

说明

io_get_devs() 获取给定节点的所有 **dev_t** （包括字符和块 **dev_t**）。它将通过 *dev_arry* 返回所有 **dev_t**、设备专用选项和 **dev_t** 类型。 *dev_arry* 中的 *dev_type* 字段将被更新，以说明 **dev_t** 类型是 **D_CHR** （字符）还是 **D_BLK** （块）。

用户需要为该数组分配内存，并在 *count* 中指明 *io_dev_info_t* 的大小。如果此数组足够大，此接口便会将数据复制到此数组中。如果它太小，则不返回任何数据，并设置计数来指明所需大小（ *io_dev_info_t* 的数目）。调用方

需要重新分配内存并重新调用该 API。

参数

node 用于检索其 **dev_t** 的 I/O 节点标记。

dev_arry 可能的值:

INPUT:

指向 *io_dev_info_t* 数组的指针。此数组由调用方分配。初始大小可以为 **IO_MAX_DEVS_IN_IOQ** （当前设置为 48）。

OUTPUT:

将返回的 **io_dev_info_t** 复制到此数组。

count 可能的值:

INPUT:

所需的 *io_dev_info_t* 的数目。初始设置应为 **IO_MAX_DEVS_IN_IOQ** 。

OUTPUT:

返回的数组中的 *io_dev_info_t* 的数目（如果数组足够大）。如果数组不够大，则:

1) 不返回标记。
2) 将计数设置为要返回的 *io_dev_info_t* 的实际计数。
3) API 返回 **IO_ERROR** 。
4) 将 **io_errno** 设置为 **IO_E_BUF_TOO_SMALL** 。

   在这种情况下，调用方需要利用返回的计数值并使用:

   (sizeof(io_dev_info_t)*count)

   来重新计算所需的缓冲区，然后分配更大的缓冲区，并且需要使用该新缓冲区和计数值来重新调用此 API。

返回值

IO_SUCCESS - 成功。
IO_ERROR - 失败。设置 io_errno 以指明错误。

错误

[IO_E_DCONF_OPEN]
[IO_E_NODE_NOEXIST]
[IO_E_DCONF_ACCESS]
[IO_E_SYSCALL]
[IO_E_MEM_ALLOC]
[IO_BUF_TOO_SMALL]

libIO 函数

名称

io_block_to_raw() - 返回给定块 **dev_t** 的字符 **dev_t**

概要

int io_block_to_raw(dev_t bdev, dev_t *rdev);

说明

可使用此 API 检索与块 **dev_t** 对应的字符 **dev_t** 。

参数

bdev 需要检索字符 **dev_t** 的块 **dev_t** 。

rdev 为给定的块 **dev_t** 检索到的字符 **dev_t** 通过指针存储在此处。

如果出现错误，则为 **NODEV**。

返回值

IO_SUCCESS - 成功。

IO_ERROR - 失败。设置 io_errno 以指明错误。

错误

[IO_E_DCONF_OPEN]

[IO_E_PARM]

[IO_E_SYSCALL]

[IO_E_DCONF_ACCESS]

libIO 函数

名称

io_block_to_char_dsf() - 将块设备专用文件映射到字符设备专用文件

概要

int io_block_to_char_dsf(char *block_dsf, char *char_dsf);

说明

此 API 将一个给定的块设备专用文件映射到对应的字符设备专用文件。

参数

block_dsf 指向要映射到字符设备专用文件的块设备专用文件的指针。

char_dsf 指向大小为 **MAXPATHLEN** 的缓冲区的指针。

返回值

IO_SUCCESS - 成功。

IO_ERROR - 失败。设置 io_errno 以指明错误。

错误

[IO_E_DCONF_OPEN]
[IO_E_PARM]
[IO_E_DCONF_ACCESS]
[IO_E_SYSCALL]
[IO_E_BUF_TOO_SMALL]

libIO 函数

名称

io_char_to_block_dsf() - 将字符设备专用文件映射到块设备专用文件

概要

int io_char_to_block_dsf(char *char_dsf, char *block_dsf);

说明

此 API 将一个给定的字符设备专用文件映射到对应的块设备专用文件。

参数

char_dsf 指向要映射到块设备专用文件的字符设备专用文件的指针。

block_dsf 指向大小为 **MAXPATHLEN** 的缓冲区的指针。

返回值

IO_SUCCESS - 成功。
IO_ERROR - 失败。设置 io_errno 以指明错误。

错误

[IO_E_DCONF_OPEN]
[IO_E_PARM]
[IO_E_DCONF_ACCESS]
[IO_E_SYSCALL]
[IO_E_BUF_TOO_SMALL]

libIO 函数

名称

io_raw_to_block() - 返回给定字符 **dev_t** 的块 **dev_t**

概要

int io_raw_to_block(dev_t rdev, dev_t *bdev);

说明

可使用此 API 检索与字符 **dev_t** 对应的块 **dev_t** 。

参数

rdev 需要检索块 **dev_t** 的字符 **dev_t** 。

bdev 为给定的字符 **dev_t** 检索到的块 **dev_t** 通过指针存储在此处。如果出现错误，则为 **NODEV**。

返回值

IO_SUCCESS - 成功。
IO_ERROR - 失败。设置 io_errno 以指明错误。

错误

[IO_E_DCONF_OPEN]
[IO_E_PARM]
[IO_E_SYSCALL]
[IO_E_DCONF_ACCESS]

libIO 函数

名称

io_legacy_to_new_dev() - 将 Legacy **dev_t** 映射到 Agile **dev_t**

概要

int io_legacy_to_new_dev(dev_t legacy_dev, int dev_type,
dev_t *new_dev);

说明

如果给定了一个 Legacy **dev_t** 和 *dev_type*，此 API 将检索其 Agile **dev_t** 。

参数

legacy_dev 要检索 Agile **dev_t** 的 Legacy **dev_t** 。

dev_type 可能的值:

D_CHR 字符 **dev_t** 。
D_BLK 块 **dev_t** 。

new_dev Agile **dev_t** 存储在此处。

返回值

IO_SUCCESS - 成功。
IO_ERROR - 失败。设置 io_errno 以指明错误。

错误

[IO_E_DCONF_OPEN]
[IO_E_PARM]
[IO_E_DCONF_ACCESS]
[IO_E_SYSCALL]

libIO 函数

名称

io_legacy_to_new_dsf() - 将 Legacy 设备专用文件映射到持久性设备专用文件

概要

int io_legacy_to_new_dsf(char *legacy_dsf, char *new_dsf);

说明

此 API 将给定的 Legacy 设备专用文件映射到持久性设备专用文件。

参数

legacy_dsf 要查找所对应的持久性设备专用文件的 Legacy 设备专用文件。

new_dsf 指向长度为 **MAXPATHLEN** 的缓冲区的指针。

返回值

IO_SUCCESS - 成功。

IO_ERROR - 失败。设置 io_errno 以指明错误。

错误

[IO_E_DCONF_OPEN]

[IO_E_PARM]

[IO_E_DCONF_ACCESS]

[IO_E_SYSCALL]

[IO_E_MATCH]

[IO_E_BUF_TOO_SMALL]

libIO 函数

名称

io_new_to_legacy_devs() - 将 Agile **dev_t** 映射到一个或多个 Legacy **dev_t**

概要

int io_new_to_legacy_devs(dev_t new_dev, int dev_type, dev_t *dev_arry,
int *count);

说明

此 API 将 Agile **dev_t** 映射到其 Legacy **dev_t** 。可以为一对多映射。

参数

new_dev 指向要获取所对应的 Legacy **dev_t** 的 Agile **dev_t** 的指针。

dev_type 可能的值:

D_CHR 字符 **dev_t** 。

D_BLK 块 **dev_t** 。

dev_arry 可能的值:

INPUT:

指向 Legacy **dev_ts** 数组的指针。该数组由调用方分配，建议的初始大小为:

(IO_MAX_DEVS_IN_IOQ * sizeof (dev_t))

OUTPUT:

将返回的 **dev_t** 复制到此数组。

count 可能的值:

INPUT:

所需的 Legacy **dev_t** 的数目（建议为 **MAX_DEVS_IN_IOQ** (48)）。

OUTPUT:

返回的数组中的 Legacy **dev_t** 的数目。如果数组不足够大，则:

1) 不返回标记。
2) 将计数设置为要返回的 **dev_t** 的实际计数。
3) API 返回 **IO_ERROR** 。
4) 将 **io_errno** 设置为 **IO_E_BUF_TOO_SMALL** 。

   在这种情况下，调用方需要利用返回的计数值并使用:

   (sizeof (dev_t)*count)

   来重新计算所需要的缓冲区，然后分配更大的缓冲区，并且需要使用该新缓冲区和计数值来重新调用此 API。

返回值

IO_SUCCESS - 成功。
IO_ERROR - 失败。设置 io_errno 以指明错误。

错误

[IO_E_DCONF_OPEN]
[IO_E_PARM]
[IO_E_DCONF_ACCESS]
[IO_E_SYSCALL]
[IO_E_BUF_TOO_SMALL]
[IO_E_MEM_ALLOC]

libIO 函数

名称

io_new_to_legacy_dsfs() - 将 Agile 设备专用文件映射到一个或多个 Legacy 设备专用文件

概要

int io_new_to_legacy_dsfs(char *new_dsf, char *legacy_dsf,int *count);

说明

此 API 检索与持久性设备专用文件对应的 Legacy 设备专用文件。可以为一对多映射。

参数

new_dsf 指向要检索其 Legacy 设备专用文件的持久性设备专用文件。

legacy_arry 可能的值：

INPUT:

指向用于存储返回的 Legacy 设备专用文件的缓冲区的指针。此缓冲区由调用方分配。缺省情况下，此缓冲区的大小应为 **(MAXPATHLEN *10)** ，并且计数应设置为 10。

OUTPUT:

将返回的设备专用文件复制到此数组。每个设备专用文件以分号分隔。

count 可能的值：

INPUT:

所需的 Legacy 设备专用文件的数目。初始大小应为 10。

OUTPUT:

返回的数组中的 Legacy 设备专用文件的数目（如果数组足够大）。如果数组不足够大，则：

1) 不返回标记。
2) 将计数设置为要返回的设备专用文件的实际计数。
3) API 返回 **IO_ERROR** 。
4) 将 **io_errno** 设置为 **IO_E_BUF_TOO_SMALL** 。

   在这种情况下，调用方需要利用返回的计数值并使用：

   (MAXPATHLEN)*count)

   来重新计算所需要的缓冲区，然后分配更大的缓冲区，并且需要使用该新缓冲区和计数值来重新调用此 API。

返回值

IO_SUCCESS - 成功。
IO_ERROR - 失败。设置 io_errno 以指明错误。

错误

[IO_E_DCONF_OPEN]
[IO_E_PARM]
[IO_E_DCONF_ACCESS]
[IO_E_SYSCALL]
[IO_BUF_TOO_SMALL]
[IO_E_MEM_ALLOC]

libIO 函数

名称

io_init_hw_path() - 使用传递的标记参数初始化硬件路径

概要

int io_init_hw_path(hw_path_t *hw_path, int flags);

说明

此 API 根据指定的 *flags* 参数初始化硬件路径结构 **hw_path_t** 。在初始化硬件路径时， **IO_TREE_LEGACY** 应该作为 *flags* 值进行传递。请参阅 *intro*(7)。在所有其他情况下，可以将零作为 *flags* 值进行传递。

参数

hw_path	指向硬件路径结构的指针。
flags	硬件路径的标记字段。 有效的标记是 0 和 **IO_TREE_LEGACY** 。

返回值

IO_SUCCESS - 成功。
IO_ERROR - 失败。

错误

无。

LIBIO_ERROR

以下内容说明了各个 **io_errno** 值，这些值会在发生 **libIO** API 故障后进行设置。

[IO_E_BUF_TOO_SMALL]	用户缓冲区太小。
[IO_E_DCONF_ACCESS]	**/dev/config** 权限被拒绝。
[IO_E_DCONF_NOEXIST]	**/dev/config** 不存在。
[IO_E_DCONF_OPEN]	**/dev/config** 未打开。
[IO_E_DYN_MAJOR]:	没有为指定节点定义动态主对象。

[IO_E_FUNC_NOT_SUPPORTED]	不支持的功能。
[IO_E_KEY_TOKEN_DEF]	没有为指定标记定义关键字，或者标记为 NULL。
[IO_E_MATCH]	未发现匹配项。
[IO_E_MEM_ALLOC]	无法分配内存。
[IO_E_NO_LCK]	正在进行另一个 IO 事件。
[IO_E_NODE_DESTROY]	无法破坏指定的节点或节点的派生节点。
[IO_E_NODE_NOEXIST]	I/O 树节点不存在。
[IO_E_NODE_PARM]	节点已存在，或者检测到参数错误。
[IO_E_OPER_FAIL]	操作失败。
[IO_E_OPEN_FLAG]	打开标记无效。
[IO_E_PARM]	参数错误。
[IO_E_PATH_STR]	字符串表示的路径无效。
[IO_E_SYSCALL]	系统调用错误。
[IO_E_UNEXPECTED]	意外错误。
[IO_E_WOULDBLOCK]	调用会在内核中阻塞。

多线程应用信息

这些 **libIO** API 不是线程安全的。

警告

libIO 中的许多 API 都是针对特定发行版的。在将来的 HP-UX 发行版中可能会删除这些 API，或更改其含义。

作者

libIO 由 HP 开发。

文件

/usr/include/sys/libIO.h	**libIO** 数据结构和标识符。
/usr/lib/libIO.sl	适用于 PA-RISC 系统的 32 位 **libIO** 。
/usr/lib/pa20_64/libIO.sl	适用于 PA-RISC 系统的 64 位 **libIO** 。
/usr/lib/hpux32/libIO.so	适用于基于 Itanium 的系统的 32 位 **libIO** 。
/usr/lib/hpux64/libIO.so	适用于基于 Itanium 的系统的 64 位 **libIO** 。

另请参阅

insf(1M)、 ioscan(1M)、 rmsf(1M)、 ioconfig(4)、 intro(7)。

名称

libkrb5、 libkrb5.sl、 libkrb5.so、 libcom_err、 libcom_err.sl、 libcom_err.so、 libk5crypto、 libk5crypto.sl、 libk5crypto.so - Kerberos 客户端库

概要

32-Bit Libraries on Itanium®-based Systems

/usr/lib/hpux32/libkrb5.so

/usr/lib/hpux32/libcom_err.so

/usr/lib/hpux32/libk5crypto.so

64-Bit Libraries on Itanium®-based Systems

/usr/lib/hpux64/libkrb5.so

/usr/lib/hpux64/libcom_err.so

/usr/lib/hpux64/libk5crypto.so

PA-RISC 系统上的 **32** 位库

/usr/lib/libkrb5.sl

/usr/lib/libcom_err.sl

/usr/lib/libk5crypto.sl

PA-RISC 系统上的 **64** 位库

/usr/lib/pa20_64/libkrb5.sl

/usr/lib/pa20_64/libcom_err.sl

/usr/lib/pa20_64/libk5crypto.sl

说明

Kerberos 是 MIT 开发的网络验证协议。现在是 IETF 标准 RFC 1510，Kerberos Network Authentication Service (V5)。共享库 **libkrb5.so/libkrb5.sl** 、 **libcom_err.so/libcom_err.sl** 和 **libk5crypto.so/libk5crypto.sl** 可遵照 Kerberos V5 规范支持验证、完整性和保密性服务。

Kerberos 通过使用传统的（共享私用密钥）加密机制，以第三方验证服务的方式执行验证。它提供了验证主体身份的方法，可不必依赖主机操作系统的验证，或者将信任基于主机地址。假如能够随意读取、修改和插入网络间传输的数据包，该协议便能够工作，而不需要网络上的所有主机具有物理安全性。

libkrb5.so/libkrb5.sl 是 Kerberos 主库，为验证、验证凭证、创建验证器、环境管理、缓存和重放缓存管理、keytab 文件管理、内存管理、主体名称形式映射和操作系统专用的调用提供 API。应该在通过 **libkrb5.so/libkrb5.sl** 库使用 API 的应用程序中加入 **<krb5.h>** 头文件。

与 **libkrb5.so/libkrb5.sl** 链接的 **libk5crypto.so/libk5crypto.sl** 可以提供加密和解密 API。用户不应将此库直接链接

到应用程序。为了添加验证，应用程序可能需要调用 Kerberos 库的一个或多个 API，以便传输实现验证所必需的消息。

libcom_err.so/libcom_err.sl 实现 Kerberos 库错误代码表。提供了数据库单独的错误代码表，以及幻数和 ASN.1 API。用户可以根据 API 失败，使用相应的 **com_err()** API 从这些表中获取错误。应该在通过 **libcom_err.so/libcom_err.sl** 库使用例行程序的应用程序中加入 **<com_err.h>** 头文件。要加入 **com_err** 库，必须将可执行文件链接到 **-lcom_err** 。

下文介绍了在 Kerberos 客户端库中实现的 API 的功能。

krb5_context 管理 API

环境用于表示每进程状态。与特定“环境”有关的全局参数存储在此结构中。此结构包含缺省领域、缺省加密类型、缺省配置文件等等。通过这些 API，可以使用完全访问权限访问存储在环境中，不应由开发人员直接访问的数据结构。其中一些常见 API 包括 **krb5_init_context()** 、 **krb5_init6_context()** 、 **krb5_free_context()** 和 **krb5_set_default_in_tkt_etypes()** 。

应该由调用者释放从 **context** 检索的且存储在 **etypes** 中的加密类型。

krb5_auth_context 管理 API

auth_context 是一个每连接环境，由直接与客户端/服务器验证有关的各个 API 使用。存储在此环境中的某些数据包括 keyblock、地址、序列号、验证器、校验和类型和重放缓存指针。其中一些常见 API 包括 **krb5_auth_con_init()** 、 **krb5_auth_con_free()** 、 **krb5_auth_con_setaddrs()** 、 **krb5_auth_con_setports()** 、 **krb5_auth_con_setflags()** 、 **krb5_auth_con_getlocalsubkey()** 和 **krb5_auth_con_genaddrs()** 。

应该使用 **krb5_auth_con_free()** 释放 **auth_context** 结构。应用程序开发人员负责使用 **krb5_free_authenticator()** 来释放分配给验证器的内存。应用程序开发人员还必须使用 **krb5_free_keyblock()** 来释放为了存储本地子 keyblock 而分配的内存。

主体访问 API

主体是参与网络通信的，带有唯一名称的客户端或服务器实例。可以通过这些 API 创建、修改和访问 Krb5_principal 的某些部分。其中一些常见 API 包括 **krb5_parse_name()** 、 **krb5_unparse_name()** 、 **krb5_free_principal()**、 **krb5_princ_realm()** 、 **krb5_copy_principal()** 等。

由于接口可能随时更改，其中一些 API 是内部函数，不适合应用程序使用。尽管可以直接访问结构中的数据元素，仍然建议使用这些 API。应该使用 **krb5_free_principal()** 释放返回的主体。

凭据缓存管理 API

这些 API 用于处理半永久存放区中，以后不同程序将要使用的存储凭据（凭证、会话密钥及其他识别信息）。凭据存储可能是硬盘或内存存储。其中一些常见 API 包括 **krb5_cc_resolve()** 、 **krb5_cc_default()** 、 **krb5_cc_initialize()** 、 **krb5_cc_destroy()** 、 **krb5_cc_store_cred()** 、 **krb5_cc_retrieve_cred()** 、 **krb5_cc_remove_cred()** 和 **krb5_cc_set_flags()** 。

应该使用 **krb5_free_credentials()** 释放检索的凭据。

重放缓存管理 **API**

这些 API 用于验证 **AP_REQ** 是否不包含重复的验证器。在验证器的站点确定的有效性时间段内，存储必须是非易失性的。其中一些常见 API 包括 **krb5_auth_to_rep()** 、 **krb5_rc_register_type()** 、 **krb5_rc_default()** 、 **krb5_rc_initialize()** 、 **krb5_rc_close()** 、 **krb5_rc_store()** 和 **krb5_rc_resolve()** 。

krb5_rc_resolve() 初始化重放缓存的专用数据。必须在调用其他重放缓存 API 之前调用此 API。应该使用 **krb5_rc_close()** 释放分配的内存。

应用程序一般不使用这些 API。

Keytab 管理 **API**

这些 API 用于存储和检索参与验证交换的无人值守服务所使用的服务密钥。 Keytab 例行程序都是原子性的。所有 keytab 类型都支持多个并行依序扫描。其中一些常见 API 包括 **krb5_kt_register()** 、 **krb5_kt_resolve()** 、 **krb5_kt_default()** 、 **krb5_kt_add_entry()** 、 **krb5_kt_close()** 、 **krb5_kt_free_entry()** 和 **krb5_kt_next_entry()** 。

要释放资源，用户应该使用 **krb5_kt_free_entry()** 。

内存管理 **API**

这些 API 用于取消分配各个例行程序所分配的内存。建议开发人员使用这些例行程序来释放数据结构。所有 API 都以 **krb5_free** 前缀开头。其中一些常见 API 包括 **krb5_free_principal()** 、 **krb5_free_data()** 、 **krb5_free_authenticator()** 、 **krb5_free_ticket()** 、 **krb5_free_cred()** 、 **krb5_free_pa_data()** 和 **krb5_free_tgt_creds()** 。

操作系统专用的 **API**

这些 API 为 **libkrb5** 库的其他部分与操作系统之间提供接口。通过某些 API 可以访问配置专用的信息、基于磁盘的 I/O 操作、基于网络的操作和操作系统专用的访问 API。其中一些常见 API 包括 **krb5_set_config_file()** 、 **krb5_get_default_realm()** 、 **krb5_get_krbhst()** 、 **krb5_gen_portaddr()** 、 **krb5_read_message()** 、 **krb5_kuserok()** 、 **krb5_timeofday()** 和 **krb5_read_passwd()** 。

应用程序专用的 **API** 和其他 **API**

这些 API 可处理通往 Kerberos 服务器的 KRB5 协议消息的发送和接收，及凭证管理和其他调用。其中一些常见 API 包括 **krb5_get_cred_from_kdc()** 、 **krb5_get_credentials()** 、 **krb5_get_in_tkt_with_password()** 、 **krb5_rd_rep()** 、 **krb5_mk_error()** 、 **krb5_sendauth()** 和 **krb5_recvauth()** 。

警告

强烈建议用 GSS-API 取代 Kerberos 调用。 Kerberos 库不是线程安全的。

作者

Kerberos 客户端库由麻省理工学院开发。该版本的库与 MIT1.3.5 兼容。

另请参阅

kdestroy(1)、 kinit(1)、 klist(1)、 kpasswd(1)、 ktutil(1)、 kvno(1)、 krb5.conf(4)、 gssapi(5)、 kerberos(5)。

名称

libslp： SLPOpen()、SLPClose()、SLPReg()、SLPDereg()、SLPDelAttrs()、SLPFindSrvs()、 SLPFindSrvTypes()、SLPFindAttrs()、 SLPParseSrvURL()、 SLPEscape()、 SLPUnescape()、 SLPFree()、 SLPGetRefreshInterval()、SLPFindScopes()、 SLPGetProperty()、SLPSetProperty() - SLP（服务定位协议）库例行程序

概要

```
#include <slp.h>
cc [ flag... ] file... -lslp [ library... ]

SLPError SLPOpen(
    cnst char *lang,
    SLPBoolean isasync,
    SLPHandle *hslp,
);

void SLPClose(
    SLPHandle *hslp
);

SLPError SLPReg(
    SLPHandle hslp,
    const char* srvurl,
    unsigned short lifetime,
    const char* srvtype,
    const char* attrs,
    SLPBoolean fresh,
    SLPRegReport callback,
    void* cookie
);

SLPError SLPDereg(
    SLPHandle hslp,
    const char* srvurl,
    SLPRegReport callback,
    void* cookie
);

SLPError SLPDelAttrs(
    SLPHandle hslp,
    const char* srvurl,
    const char* attrs,
    SLPRegReport callback,
    void* cookie
```

```
);

SLPError SLPFindSrvs(
    SLPHandle hslp,
    const char* srvtype,
    const char* scopelist,
    const char* filter,
    SLPSrvURLCallback callback,
    void* cookie
);

SLPError SLPFindSrvTypes(
    SLPHandle hslp,
    const char* namingauthority,
    const char* scopelist,
    SLPSrvURLCallback callback,
    void* cookie
);

SLPError SLPFindAttrs(:
    SLPHandle hslp,
    const char* srvurlorsrvtype,
    const char* scopelist,
    const char* attrids,
    SLPSrvURLCallback callback,
    void* cookie
);
```

说明

SLP （服务定位协议）库例行程序为编写支持 SLP 的应用程序提供标准接口。SLP API 是允许程序员编写客户端和服务器应用程序，以提供动态服务发现和选择的接口。

C 语言绑定可以提供直接映射到协议以实现最低开销。每个协议请求都对应于一个 C 语言函数，但 **SLPDereg()** 和 **SLPDelAttrs()** 函数除外，它们可以映射到 SLP 取消注册请求的不同用途。参数是针对大部分字符缓冲区的。如果让客户端分配大多数内存，并要求客户端回调函数将输入参数复制到客户端代码分配的内存，就能够简化内存管理。直接从 API 函数返回的任何内存将使用 **SLPFree()** 函数取消分配。

为符合标准 C 语言的做法，通过 API 传递和返回的所有字符型字符串必须以 null 结尾，即使 SLP 协议不使用以 null 结尾的字符串时，也是如此。作为参数传递的字符串必须为 UTF-8 编码，但也可以把它们作为 C 字符串（以 null 结尾的字节序列）传递。必须通过 API 客户端将转义字符编码为 UTF-8。一般情况下，对于 US-ASCII，常见的一个字符一个字节形式的 C 字符串都是有效的。提供了帮助转义和取消转义字符串的 API 函数。

除非另有说明，API 函数和回调的参数都为非 NULL 形式。某些参数可能还有其他限制。如果有任何参数无法满

足对其值所设的限制，操作将返回 **PARAMETER_BAD** 错误。

参数

lang 指向要使用的语言标记。

isasync 表示是以异步方式，还是以同步方式执行要求的操作。目前只支持同步操作。

srvurl 需要注册/取消注册的 URL。

lifetime 要注册的服务 URL 的周期（单位为秒），该值小于或等于 **SLP_LIFETIME_MAXIMUM** ，且大于零。

srvtype *service*:*scheme* 格式中的服务类型名称。

attrs 服务类型属性的列表。

fresh 与 SLP 服务注册一起使用，表示注册请求是新请求，还是更新现有注册的请求。目前不支持更新现有注册。

注释：目前不支持异步操作和递增注册。因此，必须始终将 *isasync* 设置为 **SLP_FALSE** ，以及将 *fresh* 设置为 **SLP_TRUE** 。否则会返回 **SLP_NOT_IMPLEMENTED** 错误。

callback 一个指针，指向处理响应时将要调用的相应类型的回调函数。

cookie 一个指针，指向从应用程序传递到 SLP 库，以便回调填充返回值的内存。

scopelist 范围的逗号分隔列表。

filter 一种模式，与 LDAPv3 搜索过滤器语法（服务器将用它来过滤结果）中的属性生成的表达式匹配。

namingauthority
要搜索的命名权限。

回调

回调函数允许应用程序开发人员控制库调用结果的处理。通过异步编程调用，将回调函数作为其中一个参数传递到库函数的程序代码，可以继续执行其作业，同时回调函数还能保持计算该操作的结果。

只要 API 库具有需要报告的结果，就会调用回调函数。回调代码在查看其他参数之前，需要检查错误代码参数。如果错误代码不是 **SLP_OK** ，则其他参数可能是 NULL 或者无效。API 库可以终止任何发生了错误的未完成操作。同样，回调函数也可以指示应该终止操作，其方法是传回 **SLP_FALSE** ，以表示它不希望收到更多的结果。不允许回调函数以递归方式调入同一个 **SLPHandle** 上的 API。

如果尝试调用 API，API 函数将返回 **SLP_HANDLE_IN_USE** 。禁止对相同句柄进行递归回调，可以简化线程安全代码的实现，这是因为句柄第二次调出期间，句柄上保留的锁将不存在。

通过设置 *net.slp.maxResults* 参数，可以控制收到结果的总数。

最后一次调用回调时，无论传递到回调的状态代码是异步还是同步，都会得到值 **SLP_LAST_CALL** 。

主要例行程序

SLPOpen()

在以 *lang* 参数传入的语言环境的 *hslp* 参数中，返回一个 **SLPHandle** 句柄。该句柄可以封装通过句柄发出的 SLP 请求的语言环境，以及实现所要求的其他任何资源的语言环境。函数的返回值为表示操作状态的 **SLPError** 代码。如果失败， *hslp* 参数将设置为 NULL。

一次只能将 **SLPHandle** 用于一个 SLP API 操作。如果以异步方式启动原始操作，当原始操作处于挂起状态时，尝试在句柄上启动其他操作都会导致 API 函数返回 **SLP_HANDLE_IN_USE** 错误。

目前不支持异步操作。如果 *isasync* 参数设置为 **SLP_TRUE** ，将返回 **SLP_NOT_IMPLEMENTED** 。

lang 是指向包含 RFC 1766 语言标记的以 null 结尾的字符数组的指针。传入 NULL 或空字符串 "" ，以使用配置计算机后能够使用的语言环境。

目前只支持英文语言标记。向 langtag 传递其他任何值将导致返回 **SLP_NOT_IMPLEMENTED** 。

返回值为:

SLP_OK
SLP_PARAMETER_BAD
SLP_NOT_IMPLEMENTED
SLP_MEMORY_ALLOC_FAILED

SLPClose()

释放与句柄 *hslp* 关联的所有资源。如果句柄无效，函数将以无提示方式返回。任何未完成的同步或异步操作都被取消，以便不再调用它们的回调函数。

SLPReg()

在包含周期 *lifetime* 的 *srvurl* 中注册 URL，以及在 *attrs* 中注册属性列表。 *attrs* 列表是属性赋值的线格式（包括保留字符的转义）逗号分隔列表。 *lifetime* 参数必须为非零值，并且小于或等于 **SLP_LIFETIME_MAXIMUM** 。如果 *fresh* 参数为 **SLP_TRUE** ，则注册是新注册（已设置 **FRESH** 标志），可以替换任何现有的注册。 *srvtype* 参数是一个服务类型名称， *service*:*scheme*. 不包含的服务 URL 中可以加入此参数。

如果 URL 位于 *service*:*scheme* 中，则忽略 *srvtype* 参数。如果 *fresh* 参数为 **SLP_FALSE** ，则更新现有的注册。

要注册的服务 URL 必须符合 SLP 服务 URL 语法，它不可以是空字符串或 NULL。如果 *srvurl* 不是 SLP 服务 URL，则返回 **SLP_INVALID_REGISTRATION** 。

如果 *lifetime* 设置为 **SLP_LIFETIME_MAXIMUM** ，在调用进程的周期内，它将保持为已注册状态。

因为 *srvurl* 参数所需的服务 SRL 语法将封装服务类型，所以会始终忽略 *srvtype* 参数。

属性列表由已注册服务的逗号分隔属性赋值表达式组成。如果不需要提供属性，注册服务时将使用空字符串 "" 。

Example: (*attr1***=***val1***),(***attr2***=***val2***),(***attr3***=***val3***)**

SLPDereg()

在其中以所有语言环境注册了服务的所有范围中，取消 URL *srvurl* 的广告。取消注册不只是限于 **SLPHandle** 的语言环境，还适用于所有语言环境。要在通过配置获取的所有范围中执行操作，需要具有 API 库。

SLPFindSrvs()

发出服务查询。

SLPFindSrvTypes()

SLPFindSrvTypes() 函数可以为 *scopelist* 所指定的范围内的服务类型，发出 SLP 服务类型的请求。结果是通过回调参数返回的。服务类型与语言环境无关，但这一点只适用于其中一个范围内注册的服务，以及指定的命名权限。

如果命名权限为“*”，则返回所有命名权限的结果。如果命名权限为空字符串，即 "" ，则使用缺省的命名权限“IANA”。不可以将“IANA”显式指定为命名权限，它只能取缺省值。如果显式加入的话，将返回 **SLP_PARAMETER_BAD** 错误。

服务类型名称将随保留原样的命名权限一起返回。如果命名权限为缺省值（也就是空字符串），会将此命名权限看作分隔号“**.**”而将其省略。 *service***:***scheme* 的 URL 中的服务类型名称将随保留原样的 *service***:** 前缀一起返回。

SLPFindAttrs()

该函数返回指定服务 URL 或服务类型的 attrids 相匹配的服务属性。如果 *srvurlorsrvtype* 是服务类型名称，则返回该服务类型所有注册的属性。

回调函数

上述所有主要的 SLP API 都需要具有回调函数。下面列出了这些回调函数及其语义。

参数包括：

hslp SLP 句柄。

errcode 一个变量，库函数用它将操作状态传递到回调函数。

cookie 一个指针，指向应用程序代码传递到 SLP 库调用的内存。

srvurl 包含所请求服务的 SLP 服务 URL。

lifetime 服务的周期（单位为秒）。

attrlist 一个指针，指向包含了属性 ID/赋值的 SLP 线格式 null 结尾逗号分隔列表的缓冲区 (*attr-id***=***attr-value-list***)** 。

函数包括：

typedef void SLPRegReport(SLPHandle hslp, SLPError errcode,
void* cookie)

SLPRegReport 回调类型用作传入 **SLPReg()** 、 **SLPDeReg()** 和 **SLPDelAttrs()** 函数的回调函数的类型。此回调函数不返回任何值。

typedef SLPBoolean SLPSrvURLCallback(SLPHandle hslp,
const char* srvurl,
unsigned short lifetime,
SLPError errcode,
void* cookie)

SLPSrvURLCallback 类型是 **SLPFindSrvs()** 函数的回调函数参数的类型。如果未异步打开 *hslp* 句柄参数，可能会通过回调取消分配返回的结果。如果需要更多数据，回调应返回 **SLP_TRUE** 。回调可能持续返回 **SLP_TRUE** ，直到使用 errcode **SLP_LAST_CALL** 调用该回调。如果没有请求更多的数据，回调应返回 **SLP_FALSE** 。

typedef SLPBoolean SLPAttrCallback(SLPHandle hslp,
const char* attrlist,
SLPError errcode,
void* cookie)

SLPAttrCallback 类型是作为 **SLPFindAttrs()** 函数的参数传递的回调函数的类型。取决于属性请求是由服务 URL 还是服务类型发出的，库行为将有所差异。

如果使用服务 URL 调用 **SLPFindAttrs()** 函数，则无论句柄是以异步方式打开的，还是以同步方式打开的，只会调用回调一次。 *attrlist* 参数将包含属性的逗号分隔列表。

如果使用服务类型调用 **SLPFindAttrs()** 函数，则持续调用回调，直到没有可用的结果。 attrlist 参数将包含属性的逗号分隔列表。如果以同步方式调用了 **SLPFindAttrs()** ，将为 SA 和 DA 的响应排序，以删除重复的响应。但如果以异步方式调用，attrlist 可以返回重复项。

如果需要更多数据，回调应返回 **SLP_TRUE** 。回调可能持续返回 **SLP_TRUE** ，直到使用 errcode **SLP_LAST_CALL** 调用该回调。如果没有请求更多的数据，回调应返回 **SLP_FALSE** 。

其他例行程序

以下是分析服务 URL 的例行程序。

SLPParseSrvURL(const char* srvurl, SLPSrvURL parsedurl)**

分析作为 null 结尾字符串传入的服务 URL，同时在指向动态分配 **SLPSrvURL** 结构的指针中返回结果。应该由 **SLPFree()** 释放 *parsedurl* 中返回的指针。

按以下方式在 **<slp.h>** 中定义 **SLPSrvURL** 结构:

```
typedef struct srvurl
{
  char *s_pcSrvType;
  char *s_pcHost;
  int   s_iPort;
  char *s_pcNetFamily;
```

```
    char *s_pcSrvPart;
  } SLPSrvURL;
```

SLPError SLPEscape(const char* unescaped,
char escaped, SLPBoolean istag)**

处理输入的字符串，以转义任何 SLP 保留字符，同时在动态分配内存中返回转义的字符串。如果 *istag* 参数为 **SLP_TRUE** ， **SLPEscape()** 将查看错误的标记字符。

SLPError SLPUnescape(const char* escaped, char unescaped,**
SLPBoolean istag)

处理输入的 **escaped** 字符串，以取消转义任何 SLP 保留字符。在动态分配内存中返回取消转义的字符串，在 **unescaped** 中返回指向该动态分配内存的指针。不再需要此内存时，应通过调用 SLPFree() 来释放它。如果 *istag* 参数为 **SLP_TRUE** ， **SLPUnEscape()** 将查看错误的标记字符。

void SLPFree (void* mem)

释放从 **SLPParseSrvURL()** 、 **SLPFindScopes()** 、 **SLPEscape()** 和 **SLPUnescape()** 返回的内存。

可以使用以下例行程序来确定本地计算机和网络上的 SLP 配置。

unsigned short SLPGetRefreshInterval(void)

返回网络上所有 SLP SA 和 DA 代理都可以接受的最小刷新间隔。

此调用不会实现，并且始终返回 **SLP_NOT_IMPLEMENTED** 。

SLPError SLPFindScopes(SLPHandle hslp,
char scopelist)**

将 *scopelist* 参数设置为指向所有可用范围值的逗号分隔列表的指针。最需要的值始终放在列表中的第一个位置。列表中还有一个始终会存在的值“DEFAULT”。

scopelist 的内存是动态分配的。不再需要此内存时，应通过调用 **SLPFree()** 来释放它。

const char* SLPGetProperty(const char* name)

返回对应 SLP 属性名称的值。返回的字符串由库拥有，并且不可将其释放。指向字符串的返回指针是指向静态内存的指针。

void SLPSetProperty(const char* name, const char* value):

可以假定该函数允许调用方设置 SLP 属性，但不可能以能够达到远程线程安全的方式与 **SLPGetProperty()** 一起实现该函数。因此，SLP 实现将完全忽略对 **SLPSetProperty()** 的所有调用，以便在线程应用程序中使用 **SLPGetProperty()** 。注释：

1. 如果返回 **SLP_NETWORK_INIT_FAILED** ，则可能是未运行 **slpd** 。出于灵活性的考虑，**OpenSLP()** 不需要在所有主机上运行 **slpd** ，但是在需要注册和取消注册服务的主机上，必须使其运行。
2. 目前未实现 **SLPDelAttrs()** 函数。编写支持 SLP 的代码的开发人员可以不必调用 **SLPDelAttrs()** ，只需使用 **SLPDeReg()** 来取消注册整个服务，然后调用 **SLPReg()** 来重新注册服务，而不必使用不需要的属性。

返回值

如果成功，则返回 **SLP_OK** ，否则返回其中一个 **SLPError** 代码。有关错误返回值的信息，请参阅 *SLPError*(3N) 。

作者

SLP 由 Caldera Systems, Inc. 开发。

另请参阅

slpd(1M)、SLPError(3N)、slp.conf(4)、slp.reg(4)、slp_syntax(7)。

符合的标准

RFC 2614，RFC2608

名称

LINES — 终端屏幕上的行数

概要

#include <curses.h>

extern int LINES;

说明

外部变量 *LINES* 表示终端屏幕上的行数。

另请参阅

initscr(3X)、<curses.h>。

历史变更记录

在 X/Open Curses 第 4 期中首次发布。

名称

llrint( )、llrintf( )、llrintl( )、llrintw( )、llrintq( ) - 舍入最接近的 long long 函数

概要

#include <math.h>

long long llrint(double x);

仅适用于 **HP Integrity** 服务器

long long llrintf(float x);

long long llrintl(long double x);

long long llrintw(extended x);

long long llrintq(quad x);

说明

llrint() 将根据当前的舍入方向将参数舍入为最接近的整数值。

llrint() 等效于 **lrint()** ，不同之处在于它舍入 **long long** ，而非 **long int** 。

仅适用于 **Integrity** 服务器

llrintf() 是 **float** 版的 **llrint()** ；它采用 **float** 参数。

llrintl() 是 **long double** 版的 **llrint()** ；它采用 **long double** 参数。

llrintw() 是 **extended** 版的 **llrint()** ；它采用 **extended** 参数。

llrintq() 等效于 HP-UX 系统上的 **llrintl()** 。

用法

要使用这些函数，请使用缺省的 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。

（对于 Integrity 服务器）要使用 **llrintw()** 或 **llrintq()** ，也可以使用 **-fpwidetypes** 选项进行编译。

确保程序包含 **<math.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

有关详细信息，请参阅《HP-UX floating-point guide for HP Integrity servers》（位于以下站点：**http://www.hp.com/go/fp** ）。

返回值

如果四舍五入后的值位于 **long long** 的范围以外，数字结果就是最大或最小的 **long long** 值，并引发无效的浮点异常。在 Integrity 服务器上，如果四舍五入后的值位于 **long long** 的范围以外，数字结果就是最小的 **long long** 值。

如果 **llrint()** 未引发其他浮点异常，但结果与参数不同，函数就会引发不精确的浮点异常。

错误

没有定义任何错误。

另请参阅

ceil(3M)、 floor(3M)、 fabs(3M)、 fmod(3M)、 fegetround(3M)、 fesetround(3M)、 llround(3M)、 lrint(3M)、 lround(3M)、 rint(3M)、 round(3M)、 trunc(3M)、 math(5)、fenv(5)。

符合的标准

llrint()、 **llrintf()**、 **llrintl()**： ISO/IEC C99（包括附件 F“IEC 60559 floating-point arithmetic”）

名称

llround( )、llroundf( )、llroundl( )、llroundw( )、llroundq( ) - 四舍五入至 long long 函数

概要

#include <math.h>

long long llround(double x);

仅适用于 **HP Integrity** 服务器

long long llroundf(float x);

long long llroundl(long double x);

long long llroundw(extended x);

long long llroundq(quad x);

说明

llround() 将其参数四舍五入至最接近的整数值。无论当前四舍五入方向如何，位于两个整数中间的参数将向零以外四舍五入。向零以外四舍五入的原则还适用于函数 **round** 和 **lround** 。

llround() 等效于 **lround()** ，不同之处在于它四舍五入至 **long long** ，而非 **long int** 。

仅适用于 **Integrity** 服务器

llroundf() 是 **float** 版的 **llround()** ；它采用 **float** 参数。

llroundl() 是 **long double** 版的 **llround()** ；它采用 **long double** 参数。

llroundw() 是 **extended** 版的 **llround()** ；它采用 **extended** 参数。

llroundq() 等效于 HP-UX 系统上的 **llroundl()** 。

用法

要使用这些函数，请使用缺省的 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。

（针对 Integrity 服务器）要使用 **llroundw()** 或 **llroundq()** ，还应使用 **-fpwidetypes** 选项进行编译。

确保程序包含 **<math.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

有关详细信息，请参阅《HP-UX floating-point guide for HP Integrity servers》（位于以下站点：**http://www.hp.com/go/fp** ）。

返回值

如果四舍五入后的值位于 **long long** 的范围以外，数字结果就是最大或最小的 **long long** 值，并引发无效的浮点异常。在 Integrity 服务器上，如果四舍五入后的值位于 **long long** 的范围以外，数字结果就是最小的 **long long** 值。

如果 **llround()** 未引发其他浮点异常，但结果与参数不同，函数就可能会引发不精确的浮点异常。

错误

没有定义任何错误。

另请参阅

ceil(3M)、 floor(3M)、 fabs(3M)、 fmod(3M)、 fegetround(3M)、 fesetround(3M)、 lrint(3M)、 llrint(3M)、 lround(3M)、 rint(3M)、 round(3M)、 trunc(3M)、 math(5)、fenv(5)。

符合的标准

llround()、 **llroundf()**、 **llroundl()**： ISO/IEC C99（包括附件 F“IEC 60559 floating-point arithmetic”）

名称

localeconv() - 查询当前语言环境的数字格式约定

概要

#include <locale.h>

struct lconv *localeconv(void);

说明

localeconv() 函数可根据程序的当前语言环境（请参阅 *setlocale*(3C) ）的规则，使用适用于数量（货币和其他）格式的值来设置 **struct lconv** 类型（在 **<locale.h>** 中定义）的对象的组成部分。

类型为 **char *** 的结构的成员为字符串，其中每个字符串（除 **decimal_point** 之外）都可以指向“”（空字符串），表示该值在当前的语言环境中不可用，或者其长度为零。类型为 **char** 的成员为非负数，其中每个数都可以是 **CHAR_MAX** （在 **<limits.h>** 中定义），表示该值在当前的语言环境中不可用。这些成员包括以下项：

char *decimal_point
用于设置非货币数量的小数点字符的格式。该值与使用 **RADIXCHAR** 作为参数（请参阅 *nl_langinfo*(3C) ）调用 **nl_langinfo()** 后返回的值相同。

char *thousands_sep
用于分隔已设置格式的非货币数量中小数点字符左侧的数字组的字符。该值与使用 **THOUSEP** 作为参数（请参阅 *nl_langinfo*(3C) ）调用 **nl_langinfo()** 后返回的值相同。

char *grouping
一个字符串，其中每个字节的数值指示已设置格式的非货币数量中每组数字的大小。

char *int_curr_symbol
适用于当前的语言环境的国际货币符号。前三个字符包含符合“ISO 4217 Codes for the Representation of Currency and Funds”规定的按字母顺序排列的国际货币符号。第四个字符（紧靠在空字符前面）用于将国际货币符号与货币数量分隔开来。

char *currency_symbol
适用于当前的语言环境的本地货币符号。使用 **CRNCYSTR** 作为参数（请参阅 *nl_langinfo*(3C) ）调用 **nl_langinfo()** 后将返回该值，同时还返回定位信息。

char *mon_decimal_point
用于设置货币数量格式的小数点。

char *mon_thousands_sep
已设置格式的货币数量中小数点左侧的数字组的分隔符。

char *mon_grouping
一个字符串，其中每个字节的数值指示已设置格式的货币数量中每个数字组的大小。

char *positive_sign
用于指示值为正数的已设置格式的货币数量的字符串。

char *negative_sign
用于指示值为负数的已设置格式的货币数量的字符串。

char int_frac_digits
将在按国际标准设置格式的货币数量中显示的小数位数（小数点右侧的位数）。

char frac_digits
将在按本地标准设置格式的货币数量中显示的小数位数（小数点右侧的位数）。

char p_cs_precedes
如果在 **currency_symbol** 的前面或后面接上已设置格式的非负货币数量的值，则分别设置为 1 或 0。

char p_sep_by_space
设置为可指示 **currency_symbol** 、符号字符串和已设置格式的非负货币数量的值之间分隔的值。

char n_cs_precedes
如果在 **currency_symbol** 的前面或后面接上已设置格式的负货币数量的值，则分别设置为 1 或 0。

char n_sep_by_space
设置为可指示 **currency_symbol** 、符号字符串和已设置格式的负货币数量的值之间分隔的值。

char p_sign_posn
设置为可指示已设置格式的非负货币数量中 **positive_sign** 的定位信息的值。

char n_sign_posn
设置为可指示已设置格式的负货币数量中 **negative_sign** 的定位信息的值。

下列成员还可用于符合 UNIX 2003 要求的应用程序：

char int_p_cs_precedes
如果在 **int_currency_symbol** 的前面或后面接上按国际标准设置格式的非负货币数量的值，则分别设置为 1 或 0。

char int_p_sep_by_space
设置为可指示 **int_currency_symbol** 、符号字符串和按国际标准设置格式的非负货币数量的值之间分隔的值。

char int_n_cs_precedes
如果在 **int_currency_symbol** 的前面或后面接上按国际标准设置格式的负货币数量的值，则设置为 1 或 0。

char int_n_sep_by_space
设置为可指示 **int_currency_symbol** 、符号字符串和按国际标准设置格式的非负货币数量的值之间分隔的值。

char int_p_sign_posn
设置为可指示按国际标准设置格式的非负货币数量的 **positive_sign** 的定位信息的值。

char int_n_sign_posn
设置为可指示按国际标准设置格式的负货币数量的 **negative_sign** 的定位信息的值。

根据以下方法解释 **grouping** 和 **mon_grouping** 的每个字节的数值:

CHAR_MAX
不再执行分组。

0 余下的数字将反复使用以前的字节。

其他 该值为构成当前组的数字的数目。将检查下一个字节，以确定当前组左侧的下一个数字组的大小。

根据以下方法解释 **p_sep_by_space** 、 **n_sep_by_space** 、 **int_p_sep_by_space** 和 **int_n_sep_by_space** 的值:

0 货币符号和值之间没有空格。

1 如果货币符号和符号字符串相邻，则它们与值之间有空格；否则，货币符号和值之间有空格。

2 如果货币符号和符号字符串相邻，则它们之间有空格；否则，符号字符串和值之间有空格。

对于符合 UNIX 2003 的应用程序的 **int_p_sep_by_space** 和 **int_n_sep_by_space** ，使用 **int_curr_symbol** 的第四个字符，而不是空格。

根据以下方法解释 **p_sign_posn** 和 **n_sign_posn** 的值:

0 圆括号括起数量和 **currency_symbol** 或 **int_curr_symbol** 。

1 符号字符串加在数量和 **currency_symbol** 或 **int_curr_symbol** 的前面。

2 符号字符串接在数量和 **currency_symbol** 或 **int_curr_symbol** 的后面。

3 符号字符串紧靠在 **currency_symbol** 或 **int_curr_symbol** 的前面。

4 符号字符串紧接在 **currency_symbol** 或 **int_curr_symbol** 的后面。

localeconv() 在工作时如同没有库函数调用 **localeconv()** 。

外部语言环境影响

语言环境

LC_NUMERIC 类别会影响调用 **localeconv()** 后返回的指针所引用的结构的 **decimal_point** 、 **thousands_sep** 和 **grouping** 成员。

LC_MONETARY 类别会影响该结构的所有其他成员。

国际代码集支持

支持单字节字符代码集和多字节字符代码集。

返回值

localeconv() 返回指向已填充的 **struct lconv** 的指针。

举例

下表说明了五种语言的货币数量的格式。

国家/地区	正数格式	负数格式	国际格式
en_US.iso88591	$1,234.56	-$1,234.56	USD 1,234.56
it_IT.iso88591	L.1.234	-L.1.234	ITL.1.234
nl_NL.iso88591	F 1.234,56	F -1.234,56	NLG 1.234,56
no_NO.iso88591	kr1.234,56	kr1.234,56-	NOK 1.234,56
pt_PT.iso88591	1,234$56	-1,234$56	PTE 1,234$56

对于这五种语言， **localeconv()** 返回的结构的货币成员的值分别为：

iso88591	**en_US.** iso88591	**it_IT.** .iso88591	**nl_NL.** iso88591	**no_NO.** iso88591	**pt_PT.**
int_curr_symbol	USD	ITL.	NLG	NOK	PTE
currency_symbol	$	L.	F	kr	$
mon_decimal_point	.	""	,	,	$
mon_thousands_sep	,	.	.	.	,
mon_grouping	\3	\3	\3	\3	\3
positive_sign	""	""	""	""	""
negative_sign	-	-	-	-	-
int_frac_digits	2	0	2	2	2
frac_digits	2	0	2	2	2
p_cs_precedes	1	1	1	1	0
p_sep_by_space	0	0	1	0	0
n_cs_precedes	1	1	1	1	0
n_sep_by_space	0	0	1	0	0
p_sign_posn	1	1	1	1	1
n_sign_posn	4	1	4	2	1

警告

不应通过调用程序来修改 **localeconv()** 返回的结构。使用类别 **LC_ALL** 、 **LC_MONETARY** 或 **LC_NUMERIC** 调用 **setlocale()** 可能会覆盖 **localeconv()** 返回时指向的结构的内容（请参阅 *setlocale*(3C) ）。

作者

localeconv() 由 OSF 和 HP 联合开发。

另请参阅

nl_langinfo(3C)、 setlocale(3C)、 localedef(4)、 langinfo(5)、 thread_safety(5)。

符合的标准

localeconv()： AES、SVID3、XPG4、ANSI C

名称

log10()、log10f()、log10l()、log10w()、log10q() - 常用对数函数

概要

#include <math.h>

double log10(double x);

float log10f(float x);

仅适用于 **HP Integrity** 服务器

long double log10l(long double x);

extended log10w(extended x);

quad log10q(quad x);

说明

log10() 返回 *x* 对于基数 10 的对数。

log10f() 是 **log10()** 的 **float** 版本；它使用 **float** 参数，并返回 **float** 结果。

仅适用于 **Integrity** 服务器

log10l() 是 **log10()** 的 **long double** 版本；它使用 **long double** 参数，并返回 **long double** 结果。

log10w() 是 **log10()** 的 **extended** 版本；它使用 **extended** 参数，并返回 **extended** 结果。

在 HP-UX 系统中， **log10q()** 等效于 **log10l()** 。

用法

要使用这些函数，请使用缺省的 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。

（对于 Integrity 服务器）要使用 **log10w()** 或 **log10q()** ，也可以使用 **-fpwidetypes** 选项进行编译。

要使用上面的任何函数，请确保程序包含 **<math.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

有关详细信息，请参阅《HP-UX floating-point guide for HP Integrity servers》（位于以下站点：**http://www.hp.com/go/fp** ）。

仅适用于 **PA-RISC** 系统

log10() 函数的 Millicode 版本可用。数学库函数的 Millicode 版本通常比标准库中对应的 Millicode 版本的速度快。要使用这些版本，可以使用 **+Olibcalls** 或 **+Oaggressive** 优化选项编译程序。

特殊情况下，Millicode 版本返回在“返回值”一节中描述的值，但不设置 **errno** 。

返回值

如果 *x* 为 +INFINITY，则 **log10()** 返回 +INFINITY。

如果 *x* 为零，则 **log10()** 返回 **–HUGE_VAL** （等于 –INFINITY），并引发除数为零的异常。

如果 *x* 小于零，则 **log10()** 返回 NaN，并引发非法异常。

如果 *x* 为 NaN，则 **log10()** 返回 NaN。

如果舍入的结果不等于精确结果， **log10()** 就引发不精确异常。

错误

如果 *x* 小于零，则 **log10()** 将 **errno** 设置为 [EDOM]。

仅适用于 **Integrity** 服务器

缺省情况下，Integrity 服务器上的 HP-UX **libm** 函数不设置 **errno** 。对于 **errno** 设置，可以使用 **+Olibmerrno** 选项进行编译。

另请参阅

exp(3M)、 exp10(3M)、 log(3M)、 log1p(3M)、 log2(3M)、 pow(3M)、 math(5)。

符合的标准

log10()： SVID3、XPG4.2、ANSI C 和 ISO/IEC C99 （包括附件 F“IEC 60559 floating-point arithmetic”）

log10f() 和 **log10l()**： ISO/IEC C99（包括附件 F“IEC 60559 floating-point arithmetic”）

名称

log1p()、 log1pf()、 log1pl()、 log1pw()、 log1pq() - 一加参数函数的自然对数

概要

#include <math.h>

double log1p(double x);

仅适用于 **HP Integrity** 服务器

float log1pf(float x);

long double log1pl(long double x);

extended log1pw(extended x);

quad log1pq(quad x);

说明

log1p() 函数计算对数函数 **log(1 +** *x***)** ，不过当 *x* 值非常小时，计算得可能更精确。

在 *x* 值非常小的情况下， **expm1()** 和 **log1p()** 函数对确保 **(((1+***x***)*****n***)-1)/***x** 的财务计算非常有用，即：

expm1(*n****log1p(***x***))/***x**

当计算每天小的利率时，前面的示例将非常适用。另请参阅 **compound()** 和 **annuity()** 。

仅适用于 **Integrity** 服务器

log1pf() 是 **float** 版的 **log1p()** ；它使用 **float** 参数，并返回 **float** 结果。

log1pl() 是 **long double** 版的 **log1p()** ；它使用 **long double** 参数，并返回 **long double** 结果。

log1pw() 是 **extended** 版的 **log1p()** ；它使用 **extended** 参数，并返回 **extended** 结果。

在 HP-UX 系统中， **log1pq()** 等效于 **log1pl()** 。

用法

要使用这些函数，请使用缺省的 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。

（对于 Integrity 服务器）要使用 **expw()** 或 **expq()** ，也可以使用 **-fpwidetypes** 选项进行编译。

确保程序包含 **<math.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

有关详细信息，请参阅《HP-UX floating-point guide for HP Integrity servers》（位于以下站点：**http://www.hp.com/go/fp** ）。

返回值

如果 *x* 为 +INFINITY，则 **log1p()** 返回 +INFINITY。

如果 *x* 为 = -1.0，则 **log1p()** 返回 **–HUGE_VAL** （等于 –INFINITY），并引发除数为零的异常。

如果 *x* < -1.0，则 **log1p()** 返回 NaN，并引发非法异常。

如果 *x* 为 NaN，则 **log1p()** 返回 NaN。

当它没有引发其他异常时，未指定 **log1p()** 是否引发不精确的异常。

错误

如果 *x* < -1.0，则 **log1p()** 将 **errno** 设置为 [EDOM] 。

仅适用于 **Integrity** 服务器

缺省情况下，Integrity 服务器上的 HP-UX **libm** 函数不设置 **errno** 。对于 **errno** 设置，可以使用 **+Olibmerrno** 选项进行编译。

另请参阅

annuity(3M)、 compound(3M)、 expm1(3M)、 log(3M)、 math(5)。

符合的标准

log1p()： XPG4.2、ANSI C 和 ISO/IEC C99 （包括附件 F“IEC 60559 floating-point arithmetic”）

log1pf() 和 **log1pl()**： ISO/IEC 和 C99（包括附件 F“IEC 60559 floating-point arithmetic”）

名称

log2( )、log2f( )、log2l( )、log2w( )、log2q( ) - 基数为 2 的对数函数

概要

#include <math.h>

double log2(double x);

float log2f(float x);

仅适用于 **HP Integrity** 服务器

long double log2l(long double x);

extended log2w(extended x);

quad log2q(quad x);

说明

log2() 返回 *x* 对于基数 2 的对数。

log2f() 是 **log2()** 的 **float** 版本；它使用 **float** 参数，并返回 **float** 结果。

仅适用于 **Integrity** 服务器

log2l() 是 **log2()** 的 **long double** 版本；它使用 **long double** 参数，并返回 **long double** 结果。

log2w() 是 **log2()** 的 **extended** 版本；它使用 **extended** 参数，并返回 **extended** 结果。

在 HP-UX 系统中， **log2q()** 等效于 **log2l()** 。

用法

要使用这些函数，请使用缺省的 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。

（对于 Integrity 服务器）要使用 **log2w()** 或 **log2q()** ，也可以使用 **-fpwidetypes** 选项进行编译。

确保程序包含 **<math.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

有关详细信息，请参阅《HP-UX floating-point guide for HP Integrity servers》（位于以下站点：**http://www.hp.com/go/fp** ）。

返回值

如果 *x* 为 +INFINITY，则 **log2()** 返回 +INFINITY。

如果 *x* 为零，则 **log2()** 返回 **–HUGE_VAL** （等于 –INFINITY），并引发除数为零的异常。

如果 *x* 小于零，则 **log2()** 返回 NaN，并引发非法异常。

如果 *x* 为 NaN，则 **log2()** 返回 NaN。

如果舍入的结果不等于精确结果， **log2()** 就引发不精确异常。

错误

如果 *x* 小于零，则 **log2()** 将 **errno** 设置为 [EDOM]。

仅适用于 **Integrity** 服务器

缺省情况下，Integrity 服务器上的 HP-UX **libm** 函数不设置 **errno** 。对于 **errno** 设置，可以使用 **+Olibmerrno** 选项进行编译。

另请参阅

exp(3M)、 exp10(3M)、 log(3M)、 log1p(3M)、 log2(3M)、 pow(3M)、 math(5)。

符合的标准

log2() 、 **log2f()** 和 **log2l()** ： ISO/IEC C99（包括附件 F“IEC 60559 floating-point arithmetic”）

名称

log( )、logf( )、logl( )、logw( )、logq( ) - 自然对数函数

概要

#include <math.h>

double log(double x);

float logf(float x);

仅适用于 **HP Integrity** 服务器

long double logl(long double x);

extended logw(extended x);

quad logq(quad x);

说明

log() 返回 *x* 的自然对数值。

logf() 是 **float** 版的 **log()** ；它采用 **float** 参数，并返回 **float** 结果。

仅适用于 **Integrity** 服务器

logl() 是 **long double** 版的 **log()** ；它采用 **long double** 参数，并返回 **long double** 结果。

logw() 是 **extended** 版的 **log()** ；它采用 **extended** 参数，并返回 **extended** 结果。

logq() 等效于 HP-UX 系统上的 **logl()** 。

用法

要使用这些函数，请使用缺省的 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。

（对于 Integrity 服务器）要使用 **logw()** 或 **logq()** ，也可以使用 **-fpwidetypes** 选项进行编译。

要使用上面的任何函数，请确保程序包含 **<math.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

有关详细信息，请参阅《HP-UX floating-point guide for HP Integrity servers》（位于以下站点：**http://www.hp.com/go/fp** ）。

仅适用于 **PA-RISC** 系统

可以使用 Millicode 版的 **log()** 和 **logf()** 函数。 Millicode 版的数学库函数通常比标准库中对应的函数运行速度快。要使用这些版本，请使用 **+Olibcalls** 或 **+Oaggressive** 优化选项编译程序。

特殊情况下，Millicode 版本返回在“返回值”一节中描述的值，但不设置 **errno** 。

返回值

如果 *x* 为正无穷大， **log()** 将返回正无穷大。

如果 *x* 为 0， **log()** 将返回 **–HUGE_VAL** （等于负无穷大）并引发除数为零的异常。

如果 *x* 小于 0， **log()** 就会返回 NaN 并引发无效异常。

如果 *x* 为 NaN， **log()** 将返回 NaN。

如果未引发其他异常，则说明未指定 **log()** 是否引发不准确异常的情况。

错误

如果 *x* 小于 0， **log()** 就会将 **errno** 设置为 [EDOM]。

仅适用于 **Integrity** 服务器

缺省情况下，Integrity 服务器上的 HP-UX **libm** 函数不设置 **errno** 。对于 **errno** 设置，可以使用 **+Olibmerrno** 选项进行编译。

另请参阅

clog(3M)、 exp(3M)、 log10(3M)、 log2(3M)、 log1p(3M)、 pow(3M)、 math(5)。

符合的标准

log()： SVID3、XPG4.2、ANSI C、ISO/IEC C99 （包括附件 F“IEC 60559 floating-point arithmetic”）

logf() 、 **logl()**： ISO/IEC C99（包括附件 F“IEC 60559 floating-point arithmetic”）

名称

logb( )、logbf( )、logbl( )、logbw( )、logbq( ) - 与基数无关的指数函数

概要

```
#include <math.h>

double logb(double x);
```

仅适用于 **HP Integrity** 服务器

```
float logbf(float x);

long double logbl(long double x);

extended logbw(extended x);

quad logbq(quad x);
```

说明

logb() 函数将计算浮点值 *x* 的指数值。形式上，对于非零的 *x* 而言，返回值就是以 *r* 为底数， |*x*| 的对数值（为带符号的浮点值）的整数部分，其中 *r* 是计算机浮点算术运算的基数。基数 *r* 在 HP-UX 系统上为 2。

如果 *x* 异常，则在确定指数前将按规范化值对待。

仅适用于 **Integrity** 服务器

logbf() 是 **float** 版的 **logb()** ；它采用 **float** 参数，并返回 **float** 结果。

logbl() 是 **long double** 版的 **logb()** ；它采用 **long double** 参数，并返回 **long double** 结果。

logbw() 是 **extended** 版的 **logb()** ；它采用 **extended** 参数，并返回 **extended** 结果。

logbq() 等效于 HP-UX 系统上的 **logbl()** 。

用法

要使用这些函数，请使用缺省的 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。

（对于 Integrity 服务器）要使用 **logbw()** 或 **logbq()** ，也可以使用 **-fpwidetypes** 选择进行编译。

确保程序包含 **<math.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。有关详细信息，请参阅《HP-UX floating-point guide for HP Integrity servers》（位于以下站点： **http://www.hp.com/go/fp** ）。

返回值

成功完成后， **logb()** 将返回 *x* 的指数值。

如果 *x* 为 NaN， **logb()** 将返回 NaN。

如果 *x* 为 ±INFINITY， **logb()** 就会返回 +INFINITY。

如果 *x* 为 0， **logb()** 将返回 **–HUGE_VAL** （等于 –INFINITY）并引发除数为零的浮点异常。

错误

如果 *x* 为零， **logb()** 就会将 **errno** 设置为 [EDOM]。

仅适用于 **Integrity** 服务器

缺省情况下，Integrity 服务器上的 HP-UX **libm** 函数不设置 **errno** 。对于 **errno** 设置，可以使用 **+Olibmerrno** 选项进行编译。

另请参阅

frexp(3M)、 ilogb(3M)、 scalb(3M)、 scalbn(3M)、 scalbln(3M)、 math(5)。

符合的标准

logb() ： SVID3、XPG4.2、ANSI C、ISO/IEC C99 （包括附件 F“IEC 60559 floating-point arithmetic”）

logbf() 、 **logbl()** ： ISO/IEC C99（包括附件 F“IEC 60559 floating-point arithmetic”）

名称

logname( ) - 返回用户的登录名

概要

#include <unistd.h>

char *logname(void);

说明

logname() 返回指向以空字符结尾的登录名的指针；它从用户的环境中提取 **$LOGNAME** 变量。

警告

logname() 返回指向每个后续调用所覆盖的静态数据的指针。

确定登录名的此方法受伪造的影响。

文件

/etc/profile

另请参阅

env(1)、 login(1)、 profile(4)、 environ(5)、 thread_safety(5)。

符合的标准

logname()： SVID2 和 XPG2

名称

longname — 获取当前终端的详细说明

概要

```
#include <curses.h>

char *longname(void);
```

说明

longname() 函数生成当前终端的详细说明。详细说明的最大长度为 128 字节。仅在调用 **initscr()** 或 **newterm()** 之后定义它。

返回值

成功完成后， **longname()** 返回指向上面所指定的说明的指针。否则，它返回错误的空指针。

错误

没有定义任何错误。

实际应用信息

longname() 的返回值可以指向后续调用 **newterm()** 所覆盖的静态区域。

另请参阅

initscr(3X)、 <curses.h>。

历史变更记录

在 X/Open Curses 第 2 期中首次发布。

X/Open Curses 第 4 期

为清楚起见，重新编写了该条目。将 **longname()** 函数的参数列表显式声明为 **void**。

名称

lrint( )、lrintf( )、lrintl( )、lrintw( )、lrintq( ) - 舍入到最接近的长整型函数

概要

#include <math.h>

long int lrint(double x);

仅适用于 **HP Integrity** 服务器

long int lrintf(float x);

long int lrintl(long double x);

long int lrintw(extended x);

long int lrintq(quad x);

说明

lrint() 根据当前舍入方向将其参数舍入到最接近的整数值。

除了舍入到 **long int** 而非 **long long** 之外， **lrint()** 等效于 **llrint()** 。

仅适用于 **Integrity** 服务器

lrintf() 是 **lrint()** 的 **float** 版本；它使用 **float** 参数。

lrintl() 是 **lrint()** 的 **long double** 版本；它使用 **long double** 参数。

lrintw() 是 **lrint()** 的 **extended** 版本；它使用 **extended** 参数。

在 HP-UX 系统中， **lrintq()** 等效于 **lrintl()** 。

用法

要使用这些函数，请使用缺省的 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。

（对于 Integrity 服务器）要使用 **lrintw()** 或 **lrintq()** ，也可以使用 **-fpwidetypes** 选项进行编译。

确保程序包含 **<math.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

有关详细信息，请参阅《HP-UX floating-point guide for HP Integrity servers》（位于以下站点：**http://www.hp.com/go/fp** ）。

返回值

如果舍入值超出了 **long int** 的范围，则数值结果为最大或最小 **long int** 值，并引发无效的浮点异常。在 Integrity 服务器上，如果舍入值超出了 **long int** 的范围，则数值结果为最小 **long int** 值。

当 **lrint()** 没有引发其他浮点运算异常，并且结果与参数不同时，此函数就会引发不精确的浮点运算异常。

错误

没有定义任何错误。

另请参阅

ceil(3M)、 floor(3M)、 fabs(3M)、 fmod(3M)、 fegetround(3M)、 fesetround(3M)、 llrint(3M)、 llround(3M)、 lround(3M)、 rint(3M)、 round(3M)、 trunc(3M)、 math(5)、fenv(5)。

符合的标准

lrint()、 **lrintf()** 和 **lrintl()**： ISO/IEC C99（包括附件 F“IEC 60559 floating-point arithmetic”）

名称

lround( )、lroundf( )、lroundl( )、lroundw( )、lroundq( ) - 舍入到长整型函数

概要

#include <math.h>

long int lround(double x);

仅适用于 **HP Integrity** 服务器

long int lroundf(float x);

long int lroundl(long double x);

long int lroundw(extended x);

long int lroundq(quad x);

说明

lround() 将其参数舍入到最接近的整数值。无论当前的舍入方向是什么，正好位于两个整数中间的参数舍去零。舍去零还适用于函数 **round** 和 **llround** 。

除了舍入到 **long int** 而非 **long long** 之外， **lround()** 等效于 **llround()** 。

仅适用于 **Integrity** 服务器

lroundf() 是 **lround()** 的 **float** 版本；它使用 **float** 参数。

lroundl() 是 **lround()** 的 **long double** 版本；它使用 **long double** 参数。

lroundw() 是 **lround()** 的 **extended** 版本；它使用 **extended** 参数。

在 HP-UX 系统中， **lroundq()** 等效于 **lroundl()** 。

用法

要使用这些函数，请使用缺省的 **-Ae** 选项，或 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。

（对于 Integrity 服务器）要使用 **lroundw()** 或 **lroundq()** ，也可以使用 **-fpwidetypes** 选项进行编译。

确保程序包含 **<math.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

有关详细信息，请参阅《HP-UX floating-point guide for HP Integrity servers》（位于以下站点：**http://www.hp.com/go/fp** ）。

返回值

如果舍入值超出了 **long int** 的范围，则数值结果为最大或最小 **long int** 值，并引发无效的浮点异常。在 Integrity 服务器上，如果舍入值超出了 **long int** 的范围，则数值结果为最小 **long int** 值。

当 **lround()** 没有引发其他浮点运算异常，并且结果与参数不同时，此函数就会引发不精确的浮点运算异常。

错误

没有定义任何错误。

另请参阅

ceil(3M)、 floor(3M)、 fabs(3M)、 fmod(3M)、 fegetround(3M)、 fesetround(3M)、 lrint(3M)、 llrint(3M)、 llround(3M)、 rint(3M)、 round(3M)、 trunc(3M)、 math(5)、fenv(5)。

符合的标准

lround()、 **lroundf()** 和 **lroundl()**： ISO/IEC C99（包括附件 F“IEC 60559 floating-point arithmetic”）

名称

lsearch( )、lfind( ) - 线性搜索和更新

概要

```
#include <search.h>

void *lsearch(
    const void *key,
    void *base,
    size_t *nelp,
    size_t width,
    int (*compar)(const void *, const void *)
);

void *lfind(
    const void *key,
    const void *base,
    size_t *nelp,
    size_t width,
    int (*compar)(const void *, const void *)
);
```

说明

lsearch() 是一个从 Knuth (6.1) Algorithm S 泛化的线性搜索例行程序。它将指针返回到一个指示可以到何处查找数据的表中。如果该数据没有出现，则其将被添加到表的末尾。

key 指向要在表中查找的数据。

base 指向表中的第一个元素。

nelp 指向一个包含表中当前元素数的整数。如果在表中添加数据，则该整数将增大。

compar 用户必须提供的比较函数的名称 （例如，**strcmp()** ）。它是通过两个参数调用的，这两个参数指向用于比较的元素。如果这两个元素相等并且非零，此函数必须返回零。

lfind() 与 **lsearch()** 一样，如果未找到数据，它将不会被添加到表中。相反，只返回一个 NULL 指针。

注释

基础表的关键字和元素的指针应该是类型 pointer-to-element，它将被强制转换为类型 pointer-to-character。

比较函数不必比较每一字节，所以除了要比较的值外，在元素中还可以包含任意数据。

虽然以前是作为类型 pointer-to-character 声明的，但是应该将返回值强制转换为类型 pointer-to-element。

示例

此代码段读入长度 ≤ **ELSIZE** 的 ≤ **TABSIZE** 字符串，并将其存储在表中，同时去除副本。

```
#include <stdio.h>

#define TABSIZE 50
#define ELSIZE 120

  char line[ELSIZE], tab[TABSIZE][ELSIZE], *lsearch( );
  size_t nel = 0;
  int strcmp( );
      . . .
  while (fgets(line, ELSIZE, stdin) != NULL &&
    nel < TABSIZE)
      (void) lsearch(line, (char *)tab, &nel,
            ELSIZE, strcmp);
      . . .
```

返回值

如果找到了搜索数据，则 **lsearch()** 和 **lfind()** 将分别对其返回一个指针。否则， **lfind()** 将返回 NULL，而 **lsearch()** 将对新添加的元素返回指针。

警告

如果表中没有添加新项的足够空间，则可能出现不确定的结果。

另请参阅

bsearch(3C)、hsearch(3C)、tsearch(3C)、 thread_safety(5)。

符合的标准

lsearch()：AES、SVID2、SVID3、XPG2、XPG3、XPG4

lfind()：AES、SVID2、SVID3、XPG2、XPG3、XPG4

名称

ltostr()、ultostr()、ltoa()、ultoa() - 将长整数转换为字符串

概要

```
#include <stdlib.h>

char *ltostr(long n, int base);

char *ultostr(unsigned long n, int base);

char *ltoa(long n);

char *ultoa(unsigned long n);
```

过时的接口

```
int ltostr_r(long n, int base, char *buffer, int buflen);

int ultostr_r(unsigned long n, int base, char *buffer, int buflen);

int ltoa_r(long n, char *buffer, int buflen);

int ultoa_r(unsigned long n, char *buffer, int buflen);
```

说明

ltostr() 将带符号 long 整型数转换为对应的指定基数的字符串表示。参数 *base* 必须在 2 和 36 之间，并包含这两个数。

ultostr() 将无符号 long 整型数转换为对应的指定基数的字符串表示。参数 *base* 必须在 2 和 36 之间，并包含这两个数。

ltoa() 将带符号 long 整型数转换为对应的基数为 10 的字符串表示，返回一个指向结果的指针。

ultoa() 将无符号 long 整型数转换为对应的基数为 10 的字符串表示，返回一个指向结果的指针。

这些函数比 **sprintf()** 要小，并且比使用它进行简单转换要快（请参阅 *printf*(3S)）。

过时的接口

ltostr_r()、**ultostr_r()**、**ltoa_r()**、**ultoa_r()** 将长整数转换为字符串。

错误

如果 *base* 的值不在 2 和 36 之间，那么 **ltostr()** 和 **ultostr()** 返回 NULL 并将外部变量 **errno** 设置为 ERANGE。

警告

ltostr()、**ultostr()**、**ltoa()** 和 **ultoa()** 的返回值指向的数据的内容被同一线程随后对这些函数进行的调用改写。

ltostr_r()、**ultostr_r()**、**ltoa_r()** 和 **ultoa_r()** 是过时的接口，只是为了兼容现有的 DCE 应用程序而进行支持。新的多线程应用程序应当使用 **ltostr()**、**ultostr()**、**ltoa()** 和 **ultoa()**。

作者

ltostr() 、 **ultostr()** 、 **ltoa()** 和 **ultoa()** 由 HP 开发。

另请参阅

strtol(3C)、printf(3S)、 thread_safety(5)。

名称

malloc()、 alloca()、 calloc()、 free()、 mallinfo()、 mallopt()、 memorymap()、 realloc()、 valloc() - 主存分配器

概要

#include <stdlib.h>

void *malloc(size_t size);

void *calloc(size_t nelem, size_t elsize);

void free(void *ptr);

void *realloc(void *ptr, size_t size);

void *valloc(size_t size);

void memorymap(int show_stats);"（已过时）

alloca()

#include <alloca.h>

void *alloca(size_t size);

System V 概要

#include <malloc.h>

char *malloc(unsigned size);

char *calloc(unsigned nelem, unsigned elsize);

void free(char *ptr);

int mallopt(int cmd, int value);

char *realloc(char *ptr, unsigned size);

struct mallinfo mallinfo(void);

备注

旧版 **malloc()** 程序包中的功能已合并到 **malloc()** 中。与 **-lmalloc** 链接程序选项对应的库 (**/usr/lib/libmalloc.a**) 现在为空库，且仅存在于基于 PA-RISC 的系统中；在基于基于 .if t Itanium®- 的系统。基于 PA-RISC 的系统上引用此库的生成文件依然有效，但在基于基于 .if t Itanium®- 的系统。

过时的接口

不建议在 HP-UX 11i v1 中使用 **memorymap()** ，现已过时。 **mallinfo()** 函数对于 **malloc** 统计数据作用更大。

说明

此处介绍的函数可提供一种简单、通用的内存分配程序包：

malloc() 为至少包含 *size* 字节的块分配空间，但不初始化此空间。

calloc() 为 *nelem* 元素数组分配空间，每个大小为 *elsize* 字节，并将此空间初始化为零。分配的实际空间容量将至少为 *nelem* * *elsize* 字节。

realloc() 将 *ptr* 指向的块的大小更改为 *size* 字节，并返回（可能已移动）块的指针。现有内容包括新旧大小的较小部分均未更改。如果 *ptr* 是空指针，则 **realloc()** 的行为与指定大小的 **malloc()** 类似。如果 *size* 为零，并且 *ptr* 不是空指针，则将释放其指向的对象，并返回 NULL。不支持经过特殊调整的块（例如 **valloc()** 创建的块）的 **realloc()** 。

valloc() 为至少包含 *size* 字节的块分配空间，该块从与 **sysconf(***__SC_PAGESIZE***)** 返回的多个值对齐的边界开始。不会初始化此空间。

free() 取消分配 *ptr* （指向 **malloc()** 、 **realloc()** 、 **calloc()** 或 **valloc()** 以前分配的块的指针）指向的空间，使空间对其他分配可用。如果 *ptr* 是空指针，则不执行操作。

mallopt() 用于控制 **malloc()** 程序包内的分配算法和其他选项。 *cmd* 的可用值包括：

M_MXFAST 将 *maxfast* 设置为 *value* 。算法以大群组的方式为所有块分配小于 *maxfast* 的大小，然后迅速地为它们分配少量的空间。 *maxfast* 的缺省值为零。

M_NLBLKS 将 *numlblks* 设置为 *value* 。上述每个“大群组”均包含 *numlblks* 个块。 *numlblks* 必须大于 1。 *numlblks* 的缺省值为 **100** 。

M_GRAIN 将 *grain* 设置为 *value* 。小于 *maxfast* 的所有块的大小都需要四舍五入为最接近 *grain* 的倍数。 *grain* 必须大于零。 *grain* 的缺省值是可以适应任何数据类型调整的最小字节数。如果设置了 *grain* ， *value* 将四舍五入为缺省值的倍数。

M_BLOCK 阻塞 **malloc()** 、 **realloc()** 、 **calloc()** 和 **free()** 中所有可阻塞的信号。此选项是为那些需要编写分配内存的信号处理程序的人员提供的。设置后，可以从信号处理程序（现为可重入）内部调用 **malloc()** 例行程序。缺省操作是 不阻塞所有可阻塞的信号。

注释：如果设置了 **M_BLOCK** 选项，则 **malloc()** 的性能将受到极大影响。

M_UBLOCK 不要阻塞 **malloc()** 、 **realloc()** 、 **calloc()** 和 **free()** 中所有可阻塞的信号。该选项可取消 **M_BLOCK** 选项启动的信号阻塞操作。

M_REL_LAST_FBLK

启用 “将最后一个可用块释放到堆”功能。该选项适用于带一个活动区域的 32 位应用程序。只有当可用块的大小大于阈值时，才释放最后一个块。

阈值定义为 **2 * (arena expansion factor) * 4096** 个字节。如果 **M_MXFAST** 值大于或等于阈值，则将禁用此功能。缺省操作是不释放

最后一个块。

这些值在 **<malloc.h>** 头文件中定义。

可以反复调用 **mallopt()**；但是一旦分配了第一个小块，就不可以更改 **M_MAXFAST**、**M_NLBLKS** 和 **M_GRAIN** 值。

mallinfo() 提供描述空间使用情况的方法，但在分配第一个小块之前，无法调用该函数。它可以返回结构 **mallinfo**：

arena : total space in arena
fsmblks : number of bytes in free small blocks
fordblks : number of bytes in free ordinary blocks
hblks : number of holding blocks
hblkhd : number of bytes in holding block headers
keepcost : cost of enabling keep option
ordblks : number of ordinary blocks
smblks : number of small blocks
uordblks : number of bytes in ordinary blocks in use
usmblks : number of bytes in small blocks in use

mallinfo 结构在 **<malloc.h>** 头文件中定义。

每个分配例行程序都会返回用于存储任何类型对象而适当对齐的（可能经过指针强制转换之后）空间的指针。

memorymap() 只显示 HP-UX 32 位操作系统内存分配器的内容。将地址和块说明的列表写入到标准输出（使用 **printf()**）。如果 *show_stats* 参数的值为 1，则还会输出与使用的块和大小数量有关的统计数据。如果值为零，则只输出内存映射。

memorymap() 显示的地址和大小可能与某个应用程序请求的地址和大小不对应。块大小（分配器查看过的）包括标头信息，以及为适当对齐块而提供的填充数据。地址还会使用特定的数量进行弥补，以容纳标头信息。

不建议在 HP-UX 11i v1 中使用 **memorymap()**，现已过时。

alloca() 从调用方堆栈为至少包含 *size* 字节的块分配空间，但不初始化此空间。调用例行程序退出时，将自动释放此空间。

alloca() 返回的内存与其他内存分配函数分配的内存无关。 **alloca()** 将地址作为参数返回到其他内存函数的行为是未定义的。

该例行程序的实现与系统有关，不建议使用该例行程序。

返回值

成功完成后， **malloc()**、 **realloc()**、 **calloc()** 和 **valloc()** 将返回用于存储任何类型对象而适当对齐的（可能经过指针强制转换之后）空间的指针。否则，返回空指针。如果 **realloc()** 返回空指针，原始指针指向的内存将保留原样。

mallopt() 在成功时返回零，在失败时返回非零值。

诊断信息

如果没有可用内存，或者能检测到 **malloc()** 管理的内存已损坏，则 **malloc()** 、 **realloc()** 、 **calloc()** 和 **valloc()** 将返回空指针。如果数据存储在块的边界以外，或者将一个无效指针（不是由 **malloc()** 、 **realloc()** 、 **calloc()** 或 **valloc()** 生成的指针）当作参数传递到 **free()** 或 **realloc()** ，则该内存可能会损坏。

如果分配了任何小块后调用 **mallopt()** ，并且 *cmd* 未设置为 **M_BLOCK** 或 **M_UBLOCK** ，或者 *cmd* 或 *value* 无效，则将返回非零值。否则，将返回零。

错误

[ENOMEM] 出现内存不足的情况时， **malloc()** 、 **realloc()** 、 **calloc()** 和 **valloc()** 将 **errno** 设置为 [ENOMEM]，同时返回空指针。

[EINVAL] 如果能检测到 **malloc()** 管理的内存已损坏，则 **malloc()** 、 **realloc()** 、 **calloc()** 和 **valloc()** 将 **errno** 设置为 [EINVAL] ，同时返回空指针。

外部语言环境影响

对于基于 PA-RISC 的系统，可以通过环境变量 **_M_ARENA_OPTS** 、 **_M_SBA_OPTS** 和 **_M_CACHE_OPTS** 来调整 API **malloc()** 系列的性能。

对于 Itanium® 的系统，除 **_M_ARENA_OPTS** 、 **_M_SBA_OPTS** 和 **_M_CACHE_OPTS** 之外，有三个新的全局变量可用于调整性能，分别为 **__hp_malloc_maxfast** 、 **__hp_malloc_num_smallblocks** 和 **__hp_malloc_grain** 。

对于线程应用程序， **malloc()** 将使用多个活动区域。从不同线程发出的内存请求由不同的活动区域处理。可以使用 **_M_ARENA_OPTS** 来调整活动区域的数量，以及假定页面大小为 4096 个字节的情况下，活动区域每次扩展自身的页数（扩展系数）。通常，应用程序中的线程越多，为获取最佳性能所应使用的活动区域就越多。线程应用程序的活动区域数介于 1 至 64 之间。

对于非线程应用程序，只使用一个活动区域。如果未设置环境变量，或者活动区域数的设置超出范围，则将使用缺省数量 8。扩展系数介于 1 至 4096 之间，缺省值为 32。同样，如果系数超出范围，则将使用缺省值。

以下是有关如何使用 **_M_ARENA_OPTS** 的示例：

$ export _M_ARENA_OPTS = 16:8

这表示活动区域数为 16，扩展大小为 8*4096 字节。通常，使用的活动区域越多，扩展系数就越小，反之亦然。

_M_SBA_OPTS 用于打开小块分配器，并且设置小块分配器的参数，即 *maxfast* 、 *grain* 和 *numlblks* 。一般来说，小块分配器已打开的应用程序，比小块分配器关闭的应用程序的运行速度更快。可以通过 **mallopt()** 打开小块分配器；但是，对 C++/Java 应用程序来说，以这种方式打开的速度显得不够快。环境变量在应用程序启动之前会将它打开。仍然能够以同样的方式使用 **mallopt()** 调用。如果已设置环境变量，但未使用小块分配器，则后续的 **mallopt()** 调用仍然可以覆盖通过 **_M_SBA_OPTS** 设置的任何内容。如果已设置环境变量，并且已使用了小块分配器，则 **mallopt()** 将不起作用。

要使用该环境变量，

$ export _M_SBA_OPTS = 512:100:16

这表示 *maxfast* 大小为 512，小块的数量为 100，粒度大小为 16。必须按此顺序提供全部 3 个值。否则，将改用缺省值。

对于基于 Itanium 的系统，引入了三个新的全局变量： **__hp_malloc_maxfast** 、 **__hp_malloc_num_smallblocks** 和 **__hp_malloc_grain** ，用来覆盖 **_M_SBA_OPTS** 环境选项。在应用程序中初始化这三个变量后，**_M_SBA_OPTS** 将会无效。通过这种方式，经过微调的应用程序可以在不同的用户环境中锁定性能。但是，对于 **_M_ARENA_OPTS** 来说，在分配任何内存块之前，后续调用 **mallopt()** 将会更改 **malloc()** 的行为。缺省情况下，在启动时，这三个变量将被初始化为零。这与以下设置等效

```
extern    int    __hp_malloc_maxfast=512;
extern    int    __hp_malloc_num_smallblocks=100;
extern    int    __hp_malloc_grain=16;
```

缺省情况下，在基于 Itanium 的系统上，SBA（小块分配）是打开的。这有助于提高应用程序的性能。用户可以设置

```
extern    int    __hp_malloc_maxfast=-1;
```

这将关闭 SBA。

有关其他所有可能值的信息，请参考 **mallopt()** 。

_M_CACHE_OPTS 用于打开线程本地缓存。使用此选项可对其访问为非线程的每个线程设置专用的缓存。这样减少了活动区域中的争用现象。对于某些多线程应用程序而言，这可以明显地提高其性能。

线程本地缓存用于保存之前使用的各种大小的块，因此这些块可供再次请求使用。此缓存的大小是可配置的。此缓存可由两种大小不一的存储桶组成，例如，一种存储桶用来存储大小在 64 至 127 字节之间的所有块；另一种存储桶用来存储大小在 128 至 255 字节之间的块，等等。

通过对 **_M_CACHE_OPTS** 环境变量进行如下设置可以调整线程本地缓存:

_M_CACHE_OPTS=bucket_size:buckets:retirement_age:
max_cache_misses:num_global_slots

必须严格按照指示的顺序提供这些值。其中前三个参数为必选参数，后两个参数为可选参数。

bucket_size 表示每个存储桶缓存的指针数量。如果 **bucket_size** 为 0，则禁用线程本地缓存。 **bucket_size** 的最大值为 32768。

buckets 表示存储桶的数量。可以缓存的最大块大小为 2^**buckets** 。 **buckets** 的范围为 8 至 32。

retirement_age 以分钟为单位，指明未使用的缓存中的块将在多长时间后释放到活动区域。 **retirement_age** 只是一个提示，缓存在指定的时间段后可能会过期，也可能不会过期。如果 **retirement_age** 为 0，则禁用过期。**retirement_age** 的最大值为 1440（即，24 小时）。

max_cache_misses 允许线程之间相互交换缓存的块。如果线程大量分配某一大小的块，则不久便会耗尽其专用缓存中的块。通过相互交换，线程可以从全局池中借用可用的块。这可能比释放回活动区域更为有效。

max_cache_misses 是对缓存算法的一个提示，它指出缺少的缓存的数量，随后它将搜索全局池以查找相应大小的块。该算法还可将未使用的缓存块释放到全局池中。如果 **max_cache_misses** 为零或者未设置，则将关闭缓存交换。

num_global_slots 表示全局缓冲池的大小。此参数仅在 **max_cache_misses** 打开时有效。 **num_global_slots** 的缺省值为 8。

为每个线程缓存的最大块数为 **bucket_size * buckets** 。

下面是如何使用 **_M_CACHE_OPTS** 的示例。

$ export _M_CACHE_OPTS = 1024:32:20:4:8

这表明 **bucket_size** 为 1024、 **buckets** 为 32、 **retirement_age** 为 20 分钟、 **max_cache_misses** 为 4、 **num_global_slots** 为 8。

$ export _M_CACHE_OPTS = 1024:32:20

这是一个有效的配置，其中未激活全局池。

_M_ARENA_OPTS 和 **_M_CACHE_OPTS** 不会影响非线程的应用程序，但 **_M_SBA_OPTS** 会影响。

注释：如果修改这些变量，则将增加现有用户出现内存缺陷（例如缓冲区溢出）的机率。

警告

malloc() 函数使用 **brk()** 和 **sbrk()** （请参阅 *brk*(2) ）来增加进程的地址空间。因此，使用 **brk()** 或 **sbrk()** 的应用程序决不能使用它们来减少地址空间，因为这将与 **malloc()** 函数混淆。

free() 和 **realloc()** 不检查其指针参数的有效性。

以下操作被视为错误的编程做法，应避免执行这些操作。结果无法预测，可能是不需要且不受支持的结果。不希望的结果的示例包括数据丢失、内存错误、总线错误和死循环。

- 尝试 **free()** 或 **realloc()** 不是由调用 **malloc()** 、 **realloc()** 、 **calloc()** 或 **valloc()** 生成的指针。
- 读取或写入的数据超出分配块的边界。
- 尝试 **realloc()** 一个对齐的块，例如 **valloc()** 的结果。

强烈建议不要执行以下操作，将来的 **malloc()** 版本中可能不支持这些操作：

- 尝试两次 **free()** 相同的块。
- 释放块后，取决于该块的未修改的内容。
- 释放块后，尝试 **realloc()** 该块。

未重复使用早期内存分配器的未记录的功能。使用上述任何错误编程做法或非建议做法的应用程序，不保证在将来的版本中仍然可以正常运行。

兼容性

旧的 **malloc()** 分配器与 **malloc()** 分配器之间的唯一外部差异在于，旧的分配器可以为零字节请求返回空指针。**malloc()** 分配器将返回有效的内存地址。对大多数应用程序来说，这个问题无关紧要。

缺省情况下，在基于 Itanium 的系统上， SBA （小块分配）是打开的，而在 PA-RISC 系统上，SBA 是关闭的。这是出于性能考虑而采取的配置。有关详细信息，请参考联机帮助页的“外部语言环境影响”一节。

另请参阅

brk(2)、 errno(2)、 thread_safety(5)。

符合的标准

malloc()： AES、SVID2、SVID3、XPG2、XPG3、XPG4、FIPS 151-2、POSIX.1、ANSI C

calloc()： AES、SVID2、SVID3、XPG2、XPG3、XPG4、FIPS 151-2、POSIX.1、ANSI C

free()： AES、SVID2、SVID3、XPG2、XPG3、XPG4、FIPS 151-2、POSIX.1、ANSI C

mallinfo()： SVID2、XPG2

mallopt()： SVID2、SVID3、XPG2

realloc()： AES、SVID2、SVID3、XPG2、XPG3、XPG4、FIPS 151-2、POSIX.1、ANSI C

名称

mbrlen( ) - 获取字符中的字节数（可重新启动）

概要

#include <wchar.h>

size_t mbrlen(const char *__restrict s, size_t n, mbstate_t *__restrict ps);

说明

如果 *s* 不是一个空指针，则 **mbrlen()** 确定由 *s* 所指向字符组成的字节数。它等效于：

```
mbstate_t internal;
mbrtowc(NULL, s, n, ps != NULL ? ps : &internal);
```

如果 *ps* 是一个空指针，则 **mbrlen()** 函数使用其自身的内部 **mbstate_t** 对象，它在程序开始时被初始化为其初始转换状态。否则， *ps* 所指向的 **mbstate_t** 对象用于完整地描述相关字符序列的当前转换状态。

实际应用信息

如果应用程序满足下列条件，则可以使用该函数的原型：

a. 符合 **c99** 的要求。

b. 使用值 >=500 的 **-D_XOPEN_SOURCE** 宏进行编译。

c. 使用值 >= 200112 的 **-D_POSIX_C_SOURCE** 宏进行编译。

外部语言环境影响

环境变量

该函数的行为受当前语言环境的 **LC_CTYPE** 类别的影响。

返回值

mbrlen() 函数返回以下各项中其适用的第一个数值：

0	如果接下来的 *n* 个或更少的字节完成与空宽字符相对应的字符。
positive	如果接下来的 *n* 个或更少的字节完成一个有效字符；返回数值是完成该字符的字节数。
(*size_t*)-2	如果接下来的 *n* 个字节有助于一个非完成但可能有效的字符，并且已经处理了全部 *n* 个字节。当 *n* 拥有至少 **MB_CUR_MAX** 宏的数值时，仅当 *s* 指向一个冗余移位序列（用于实现与状态有关的编码）时才会发生这一情况。
(*size_t*)-1	如果发生编码错误，在此情况下，接下来的 *n* 个或更少的字节无助于一个完整且有效的字符。在此情况中， [EILSEQ] 存储于 **errno** 中，并且未定义转换状态。

错误

如果发生以下情况，则 **mbrlen()** 函数可能失败：

[EILSEQ]	检测到无效的字符序列。

[EINVAL] *ps* 指向包含无效转换状态的对象。

作者

mbrlen() 由 HP 和 Mitsubishi Electric Corporation 联合开发。

另请参阅

mbrtowc(3C)、 mbsinit(3C)、 glossary(9)。

名称

mbrtowc( ) - 将字符转换为宽字符代码（可重新启动）

概要

#include <wchar.h>

size_t mbrtowc(wchar_t *__restrict pwc, const char *__restrict s, size_t n, mbstate_t *__restrict ps);

说明

如果 *s* 是空指针，则 **mbrtowc()** 函数等效于调用：

mbrtowc(NULL, "", 1, ps)

在这种情况下，将忽略参数 *pwc* 和 *n* 的值。

如果 *s* 不是空指针，则 **mbrtowc()** 函数从 *s* 指向的字节开始，至多检查 *n* 个字节，以确定完成下一个字符所需的字节数（包括任何 shift 序列）。如果函数确定已完成下一个字符，则将确定对应的宽字符的值，如果 *pwc* 不是空指针，就会将该值存储在 *pwc* 指向的对象中。如果对应的宽字符为空宽字符，则描述的生成状态为初始转换状态。

如果 *ps* 是空指针，则 **mbrtowc()** 函数将使用自身的，在程序启动时将初始化为初始转换状态的内部 mbstate_t 对象。否则，将使用 *ps* 指向的 mbstate_t 对象来完整地描述关联的字符序列的当前转换状态。

实际应用信息

如果应用程序满足下列条件，则可以使用该函数的原型：

a. 符合 **c99** 的要求。

b. 使用值 >=500 的 **-D_XOPEN_SOURCE** 宏进行编译。

c. 使用值 >= 200112 的 **-D_POSIX_C_SOURCE** 宏进行编译。

外部语言环境影响

环境变量

该函数的行为受当前语言环境的 **LC_CTYPE** 类别的影响。

返回值

mbrtowc() 函数返回下列项中可适用的第一项：

0	如果后面的 *n* 个或更少的字节完成了与空宽字符（存储的值）对应的字符，则将返回该值。
Positive	如果后面的 *n* 个或更少的字节完成了有效的字符（存储的值），则将返回该值；返回值为完成字符的字节数。
(*size_t*)-2	如果后面的 *n* 个字节构成了不完整的但可能有效的字符，并且已处理了所有的 *n* 字节（未存储任何值），则将返回该值。如果 *n* 至少包含了 **MB_CUR_MAX** 宏的值，则仅当 *s* 指向一系列冗余的转换序列（适用于带有状态相关编码的实现）时，才出现这种情况。

(*size_t*)-1　如果出现了编码错误，则将返回该值。在这种情况下，后面的 *n* 个或更少的字节不会构成完整且有效的字符（未存储任何值）。此外， [EILSEQ] 将存储在 **errno** 中，并且转换状态是不确定的。

错误

出现以下情况时， **mbrtowc()** 函数可能失败：

[EILSEQ]　检测到无效的字符序列。

[EINVAL]　*ps* 指向包含无效转换状态的对象。

警告

除 ASCII 字符外，宽字符（类型为 **wchar_t** ）的代码值特定于 **LC_CTYPE** 环境变量指定的有效语言环境。这些值可能无法与通过指定目前支持，或者将来可能支持的其他语言环境而获取的值兼容。建议不要使用宽字符常量和宽字符串文字（请参阅《C Reference Manual》），也不要将宽字符代码值存储在文件或设备中，原因是将来的标准可能要求更改宽字符的代码赋值。但是，可以放心地使用与 ASCII 代码集的字符对应的宽字符常量和宽字符串文字，这是因为它们的值肯定与其 ASCII 代码集的值相同。

作者

mbrtowc() 由 HP 和 Mitsubishi Electric Corporation 开发。

另请参阅

mbsinit(3C)、 glossary(9)。

名称

mbsinit() - 确定转换对象状态

概要

#include <wchar.h>

int mbsinit(const mbstate_t *ps);

说明

如果 *ps* 不是空指针，则 **mbsinit()** 函数确定 *ps* 指向的对象是否描述初始转换状态。

实际应用信息

在当前语言环境的 **LC_CTYPE** 类别的特定设置规则下，mbstate_t 对象用来描述从特定字符序列到宽字符序列（反之亦然）的当前转换状态。

不论在哪个方向转换，初始转换状态都对应于初始转换状态中的新字符序列的开头。值为零的 mbstate_t 对象至少是一种描述初始转换状态的方式。值为零的 mbstate_t 对象可用于初始化涉及任何字符序列的转换。

如果应用程序满足下列条件，则可以使用该函数的原型：

a. 符合 **c99** 的要求。

b. 使用值 >=500 的 **-D_XOPEN_SOURCE** 宏进行编译。

c. 使用值 >= 200112 的 **-D_POSIX_C_SOURCE** 宏进行编译。

外部语言环境影响

环境变量

该函数的行为受当前语言环境的 **LC_CTYPE** 类别的影响。

返回值

如果 *ps* 是空指针或者所指向的对象描述初始转换状态，则 **mbsinit()** 函数返回非零值；否则，返回零。

在下列情况中该行为是未定义的：任何描述为“重新启动的”函数更改了 mbstate_t 对象，然后将其与不同的字符序列一起使用，或者采用其他转换方向，或者 **LC_CTYPE** 类别设置与以前的函数调用不同。

错误

没有定义任何错误。

作者

mbsinit() 由 HP 和 Mitsubishi Electric Corporation 联合开发。

另请参阅

mbrlen(3C)、 mbrtowc(3C)、 mbsrtowcs(3C)、 wcrtomb(3C)、 wcsrtombs(3C)。

名称

mbsrtowcs( ) - 将字符串转换为宽字符字符串（可重新启动）

概要

#include <wchar.h>

size_t mbsrtowcs(wchar_t *__restrict dst, const char **__restrict src,
size_t len, mbstate_t *__restrict ps);

说明

mbsrtowcs() 函数可以将从 *ps* 指向的对象描述的转换状态开始的一系列字符，由 *src* 间接指向的数组转换为一系列对应的宽字符。如果 *dst* 不是空指针，则将转换的字符存储在 *dst* 指向的数组内。转换将持续进行，直到转换了同样需要存储的结尾空字符为止。出现以下任一情况时，转换将提前停止：

- 遇到了一系列不是来自有效字符的字节时。
- 已将 *len* 代码存储到 *dst* 指向的数组（并且 *dst* 不是空指针）中时。

类似于通过调用 **mbrtowc()** 函数执行每次转换。

如果 *dst* 不是空指针，则向 *src* 指向的指针对象分配一个空指针（如果由于到达结尾空字符而停止转换），或紧接在最后一个已转换字符后面的地址（如果有的话）。如果由于到达结尾空字符而停止转换，并且 *dst* 不是空指针，则描述的生成状态将是初始转换状态。

如果 *ps* 是空指针，则 **mbsrtowcs()** 函数将使用自身的，在程序启动时将初始化为初始转换状态的内部 mbstate_t 对象。否则，将使用 *ps* 指向的 mbstate_t 对象来完整地描述关联的字符序列的当前转换状态。实现的行为类似于没有任何该规范中定义的函数调用 **mbsrtowcs()** 。

实际应用信息

如果应用程序满足下列条件，则可以使用该函数的原型：

a. 符合 **c99** 的要求。

b. 使用值 >=500 的 **-D_XOPEN_SOURCE** 宏进行编译。

c. 使用值 >= 200112 的 **-D_POSIX_C_SOURCE** 宏进行编译。

外部语言环境影响

环境变量

该函数的行为受当前语言环境的 **LC_CTYPE** 类别的影响。

返回值

如果输入转换遇到一系列不构成有效字符的字节，则将出现编码错误。在这种情况下， **mbsrtowcs()** 函数将宏 [EILSEQ] 的值存储在 **errno** 中，同时返回 (*size_t*)-1 ；转换状态不确定。否则，该函数将返回成功转换的字符数，该数目不包括结尾空字符（如果有的话）。

错误

出现以下情况时， **mbsrtowcs()** 函数可能失败：

[EILSEQ] 检测到无效的字符序列。

[EINVAL] *ps* 指向包含无效转换状态的对象。

警告

除 ASCII 字符外，宽字符（类型为 **wchar_t** ）的代码值特定于 **LC_CTYPE** 环境变量指定的有效语言环境。这些值可能无法与通过指定目前支持，或者将来可能支持的其他语言环境而获取的值兼容。建议不要使用宽字符常量和宽字符串文字（请参阅《C Reference Manual》），也不要将宽字符代码值存储在文件或设备中，原因是将来的标准可能要求更改宽字符的代码赋值。但是，可以放心地使用与 ASCII 代码集的字符对应的宽字符常量和宽字符串文字，这是因为它们的值肯定与其 ASCII 代码集的值相同。

作者

mbsrtowcs() 由 HP 和 Mitsubishi Electric Corporation 联合开发。

另请参阅

mbrtowc(3C)、 mbsinit(3C)、 glossary(9)。

名称

memalign() - 分配对齐的内存

概要

#include <stdlib.h>

void *memalign(size_t boundary, size_t size);

说明

memalign() 将为大小为 *size* 字节的块分配空间，其地址是 *boundary* 的倍数。此空间未初始化。 *boundary* 必须为 2 的乘方。

外部语言环境影响

memalign() 使用的是 **malloc()** 分配器。有关调整的信息，请参阅 *malloc*(3C) 联机帮助页。

返回值

成功完成后， **memalign()** 将返回一个指针，指向与 *boundary* 的倍数对齐的空间。否则，它将返回空指针。

诊断信息

如果没有可用内存或 *boundary* 的值不是 2 的乘方， **memalign()** 就会返回空指针。

错误

[ENOMEM]

当出现内存不足的情况时， **memalign()** 就会将 **errno** 设置为 [ENOMEM] ，并返回空指针。

[EINVAL]

如果 *boundary* 的值不是 2 的乘方，则 **memalign()** 就会将 **errno** 设置为 [EINVAL] ，并返回空指针。

[EINVAL]

当检测到 **malloc()** 所管理的内存已损坏时， **memalign()** 就会将 **errno** 设置为 [EINVAL] ，并返回空指针。

警告

有关警告，请参阅 *malloc*(3C) 联机帮助页。

另请参阅

thread_safety(5)。

符合的标准

目前似乎还没有适用于 **memalign()** 的标准。有些实现不会检查 *boundary* 是否为 2 的乘方。 HP-UX 实现并非从任何先前的实现派生而来。

名称

内存: memccpy()、 memchr()、 memcmp()、 memcpy()、 memmove()、 memset()、 bcmp()、 bcopy()、 bzero()、 ffs() - 内存操作

概要

```
#include <string.h>

void *memccpy(void *__restrict s1, const void *__restrict s2, int c,
        size_t n);

void *memchr(const void *s, int c, size_t n);

int memcmp(const void *s1, const void *s2, size_t n);

void *memcpy(void *__restrict s1, const void *__restrict s2, size_t n);

void *memmove(void *s1, const void *s2, size_t n);

void *memset(void *s, int c, size_t n);

#include <strings.h>

int bcmp(const void *s1, const void *s2, size_t n);

void bcopy(const void *s1, void *s2, size_t n);

void bzero(void *s, size_t n);

int ffs(int i);
```

备注

bcmp()、 **bcopy()**、 **bzero()**、 **ffs()** 和 **<strings.h>** 是专门为实现 BSD 应用程序的可移植性而提供的，对于可移植性非常重要的新应用程序，不建议使用这些函数。而对于可移植的应用程序，请分别使用 **memcmp()**、 **memmove()** 和 **memset()**。 **ffs()** 没有可移植的等效项。

说明

这些函数可以在内存区域（以计数定界的字节的数组，但不是以空字节结尾）中尽量以最有效的方式工作。它们不检查任何接收内存区域的溢出。

<string.h> 头文件中定义了所有这些函数、类型 **size_t** 和常量 **NULL**。

memccpy() 将 *s2* 指向的对象的字节复制到 *s1* 指向的对象，以便在复制了第一个出现的字节 *c* 后，或者复制了 *n* 个字节后停止复制（无论哪种情况首先发生）。如果在重叠的对象之间进行复制，则该行为将是不确定的。 **memccpy()** 在复制了 *s1* 中的 *c* 后，将返回指向该字节的指针；如果在最前面的 *n* 个字节的 *s2* 中未找到 *c*，则返回空指针。

memchr() 在 *s* 指向的对象的最初 *n* 个字节（每个都解释为 **unsigned char**）中，查找第一个出现的 *c*（已转换为 **unsigned char**）。 **memchr()** 将返回指向找到的字节的指针；如果对象中没有出现该字节，则返回空指针。

memcmp() 将 *s1* 指向的对象的最前面的 *n* 个字节与 *s2* 指向的对象的最前面 *n* 个字节进行比较。 **memcmp()** 根据 *s1* 指向的对象是大于、等于还是小于 *s2* 指向的对象，返回一个大于、等于或小于零的整数。非零返回值的符号是由进行比较的对象中不同的第一对字节（两者都已解释为 **unsigned char** ）的差值的符号确定的。

memcpy() 从 *s2* 指向的对象复制 *n* 个字节到 *s1* 指向的对象。如果在重叠的对象之间进行复制，则该行为将是不确定的。 **memcpy()** 将返回 *s1* 的值。

memmove() 从 *s2* 指向的对象复制 *n* 个字节到 *s1* 指向的对象。复制时，假定首先将 *s2* 指向的对象的 *n* 个字节，复制到不与 *s1* 和 *s2* 指向的对象重叠的 *n* 个字节的临时数组，然后将临时数组的 *n* 个字节复制到 *s1* 指向的对象。 **memmove()** 将返回 *s1* 的值。

memset() 将 *c* （已转换为 **unsigned char** ）的值复制到 *s* 指向的对象的最前面 *n* 个字节的每个字节。 **memset()** 将返回 *s* 的值。

bcopy() 从 *s2* 指向的区域复制 *n* 个字节到 *s1* 指向的区域。

bcmp() 对于 *s1* 指向的区域和 *s2* 指向的区域，将两者的最前面 *n* 个字节进行比较。如果所比较的字节相同， **bcmp()** 将返回零；否则返回非零值。假定这两个区域的长度均为 *n* 个字节。

bzero() 可通过将 *s* 指向的区域中的 *n* 个字节设置为零，来清除这些字节。

ffs() 查找第一个设置的位（以最不重要的位开头），并返回该值的索引。位从 1 开始编号。返回值为 0 表示 *i* 为零。

国际代码集支持

这些函数只支持单字节字节代码集。

警告

<string.h> 中定义的函数以前在 **<memory.h>** 中定义。

文件

/usr/include/string.h

另请参阅

string(3C)、 thread_safety(5)、 glossary(9)。

符合的标准

memccpy()：AES、SVID2、SVID3、XPG2、XPG3、XPG4

memchr()：AES、SVID2、SVID3、XPG2、XPG3、XPG4、ANSI C

memcmp()：AES、SVID2、SVID3、XPG2、XPG3、XPG4、ANSI C

memcpy()：AES、SVID2、SVID3、XPG2、XPG3、XPG4、ANSI C

memmove()：AES、SVID3、XPG4、ANSI C

memset()：AES、SVID2、SVID3、XPG2、XPG3、XPG4、ANSI C

名称

meta — 启用或禁用 meta 键

概要

#include <curses.h>

int meta(WINDOW *_win_, bool _bf_);

说明

最初，不论终端返回 7 位还是 8 位，输入中的重要位都取决于显示驱动程序的控制模式（请参阅《X/Open System Interface Definitions, Issue 4, Version 2》规范和 General Terminal Interface ）。要强制返回 8 位，可以调用 *meta*(*win*,TRUE)。要强制返回 7 位，可以调用 *meta*(*win*, FALSE)。通常忽略 *win* 参数。如果为终端定义 **terminfo** 功能 **smm** (meta_on) 和 **rmm** (meta_off)，则当调用 *meta*(*win*, TRUE) 时发送 **smm**，当调用 *meta*(*win*, FALSE) 时发送 **rmm**。

返回值

成功完成后， **meta()** 返回 OK。否则返回 ERR。

错误

没有定义任何错误。

实际应用信息

使用在《X/Open System Interface Definitions, Issue 4, Version 2》规范 (General Terminal Interface) 中指定的 CS7 或 CS8 控制模式标志, 在 Curses 外部实现相同的效果。

meta() 函数用于与带有 7 位字符设置和“meta”键（可用于设置第八位）的终端一起使用。

另请参阅

getch(3X)、 curses_intro(3X)，参阅输入处理小节、 <curses.h>，《X/Open System Interface Definitions, Issue 4, Version 2》规范的第 9.2 节和 Parameters That Can Be Set （ISTRIP 标志）。

历史变更记录

在 X/Open Curses 第 4 期中首次发布。

名称

mkdirp()、rmdirp() - 创建、删除路径中的目录

概要

#include <libgen.h>

int mkdirp (const char ****path*, mode_t** *mode***);**

int rmdirp (char ****d*, char ****d1***);**

说明

mkdirp 将在给定的 *path* 中按给定的 *mode* 创建所有缺失的目录（有关 *mode* 的值，请参阅 *chmod*(2) ）。 *mode* 参数的保护部分将由进程的文件创建掩码进行修改（请参阅 *umask*(2) ）。

rmdirp 将删除路径 *d* 中的目录。此删除操作从路径的末尾开始，并向根目录的方向尽可能后退。如果出现错误，则将剩余的路径存储到 *d1* 中。只有在 **rmdirp** 可以删除路径中的所有目录的情况下，它才会返回 0。

要使用这些接口，应指定 **-lgen** ，以链接到 libgen 库中，例如：

cc foo.c -lgen

返回值

如果无法创建所需的目录，则 **mkdirp** 将返回 -1，并将 **errno** 设置为 **mkdir** 错误编号之一。如果所有以其开头的目录都已创建或存在，则它将返回零。

举例

```
/* create scratch directories */
if(mkdirp("/tmp/sub1/sub2/sub3", 0755) == -1) {
   fprintf(stderr, "cannot create directory");
   exit(1);
}
chdir("/tmp/sub1/sub2/sub3");
.
.
.
/* cleanup */
chdir( /tmp");"
rmdirp("sub1/sub2/sub3");
```

警告

mkdirp 使用 **malloc** 为字符串分配临时空间。

如果路径中有 "**.**" 或 "**..**"，则 **rmdirp** 将返回 **-2** 。如果尝试删除当前目录，则返回 **-3** 。如果出现这两个错误以外的错误，则 **rmdirp** 将返回 **-1** 。

另请参阅

chmod(2)、 mkdir(2)、 rmdir(2)、 umask(2)、 thread_safety(5)。

名称

mkfifo( ) - 创建 FIFO 文件

概要

#include <sys/stat.h>

int mkfifo(char *path, mode_t mode);

说明

mkfifo() 创建一个新 FIFO （先入先出）文件，其路径名由 *path* 指定。当通过该过程的文件创建掩码更改时，新文件的文件权限位根据 *mode* 参数进行初始化：对于在该过程的文件模式创建掩码中设置的每个位，在新文件的模式中对应的位被清除（请参阅 *umask*(2) ）。 *mode* 中的位除了文件权限位外都被忽略。

FIFO 所有者 ID 设置为该过程的 effective-user-ID 。如果在父目录上设置 set-group-ID 位，则 FIFO 组 ID 就设置为该目录的组 ID 。否则将 FIFO 组 ID 设置为该过程的有效组 ID 。

有关管道的 I/O 行为的详细信息，请参阅 *read*(2) 和 *write*(2) 。

以下符号常量在 **<sys/stat.h>** 头文件中定义，并应该用来构造 *mode* 参数的值。传递的值应当是所需权限的逐位或运算所得到的：

S_IRUSR	由所有者读取。
S_IWUSR	由所有者写入。
S_IRGRP	由组读取。
S_IWGRP	由组写入。
S_IROTH	由其他用户读取。
S_IWOTH	由其他用户写入。

返回值

mkfifo() 成功完成后返回 0。否则，它返回 -1，没有创建 FIFO ，并设置 **errno** 以指明错误。

错误

如果遇到下列的任何一条件，则 **mkfifo()** 失败，并且不创建新文件：

[EACCES]	路径前缀的某部分拒绝搜索权限。
[EEXIST]	命名的文件已存在。
[EFAULT]	参数 *path* 指向该过程外的已分配地址空间。可靠地检测该错误事与实现相关。
[ELOOP]	转换路径名时，遇到过多的符号链接。
[ENAMETOOLONG]	尽管 **_POSIX_NO_TRUNC** 在起作用，但指定路径名的长度超过了 **PATH_MAX** 个字节，或路径名的某部分长度超过了 **NAME_MAX** 个字节。
[ENOENT]	路径前缀的某部分不存在。

[ENOENT] 参数 *path* 为空。

[ENOSPC] 文件系统上的空间不足。

[ENOTDIR] 路径前缀的某部分不是一个目录。

[EROFS] 创建文件所在目录位于一个只读文件系统中。

作者

mkfifo() 由 HP 开发。

另请参阅

chmod(2)、mknod(2)、pipe(2)、stat(2)、umask(2)、 mknod(5)、stat(5)、 thread_safety(5)、 types(5)。

符合的标准

mkfifo()：AES、SVID3、XPG3、XPG4、FIPS 151-2、POSIX.1

名称

mktemp()、 mkstemp() - 创建唯一的文件名

概要

#include <stdlib.h>

char *mktemp(char *template);

int mkstemp(char *template);

备注

提供这些函数只是为了应用程序的向后兼容性和可导入性，对于可移植性比较重要的新应用程序不推荐使用。对于可移植应用程序，请改用 **tmpfile()** （请参阅 *tmpfile*(3S) ）。

说明

mktemp() 通过唯一的文件名替代 *template* 所指向字符串的内容，并返回 *template* 的地址。 *template* 中的字符串应当类似于以 6 个 **X** 结尾的文件名； **mktemp()** 利用可移植文件名字符集中的单字节字符代替每个 **X** 。选择名称的原则是获得的名称不与现有文件重名。

mkstemp() 对模板进行同样的替换，但打开进行读写的文件后，也返回模板文件的文件描述符。因此 **mkstemp()** 可以防止测试文件是否存在和打开它进行使用二者之间出现竞争状态的可能性。

返回值

除了用完字母或基础服务失败之外（在这种情况下，结果是一个指向空字符串 **""** 的指针）， **mktemp()** 将返回自己的参数。

成功完成后， **mkstemp()** 返回一个打开文件的描述符，或者如果不能创建适合的文件，则返回 -1。

警告

有可能用完字母。

mktemp() 和 **mkstemp()** 不进行检查来确定 *template* 的文件名部分是否超过最大的可允许文件名长度。

将 **process_id_max** 可调参数设置为大于 99999 的值可以影响 **mktemp()** 和 **mkstemp()** 中临时文件名的构成。

mktemp() 和 **mkstemp()** 的将来版本可能不支持使用进程 ID 的临时文件名的构成。

另请参阅

getpid(2)、 open(2)、 tmpfile(3S)、 tmpnam(3S)、 thread_safety(5)。

符合的标准

mktemp()： SVID2、 SVID3、 XPG2

名称

mktimer - 分配每个进程计时器

概要

#include <sys/timers.h>

timer_t mktimer(int clock_type, int notify_type, void *itimercbp);

说明

mktimer() 函数以所指定的系统级内的时钟作为计时基准，用于分配每个进程计时器。 **mktimer()** 将返回类型为 **timer_t** 的唯一性计时器 ID ，用于标识计时器请求中的计时器（请参阅 *gettimer*(3C) ）。 **clock_type** 将指定系统级内的时钟，用作新计时器的计时基准。 **notify_type** 则指定当计时器过期时通知进程的机制。

mktimer() 支持 **clock_type** 为 **TIMEOFDAY** ， **notify_type** 为 **DELIVERY_SIGNALS** 的每个进程计时器。

如果 **notify_type** 为 **DELIVERY_SIGNALS** ，则系统会在每次计时器过期时，向进程发送 **SIGALRM** 信号。

对于 **clock_type TIMEOFDAY** ，与计算机关联的时钟精确度和最大值分别为 **1/HZ** 和 **MAX_ALARM** 秒。这些常量在 **<sys/param.h>** 中定义。

返回值

成功完成后， **mktimer()** 将返回 **timer_t** ，后者可传递给每个进程计时器调用。如果未成功，则 **mktimer()** 将返回值 (**timer_t**)−1 ，并设置 **errno** 以指明错误。

错误

如果满足下列任一条件，则 **mktimer()** 将失败：

[EAGAIN] 调用进程已分配完其所允许使用的所有计时器。

[EINVAL] **clock_type** 是未定义的，或不允许使用指定的通知机制。

文件

/usr/include/sys/timers.h

/usr/include/sys/param.h

另请参阅

timers(2)、 getclock(3C)、 gettimer(3C)、 reltimer(3C)、 rmtimer(3C)、 setclock(3C)、 thread_safety(5)。

符合的标准

mktimer()：AES

名称

modf( )、modff( )、modfl( )、modfw( )、modfq( ) - 分解浮点数

概要

#include <math.h>

double modf(double x, double *iptr);

仅适用于 **HP Integrity** 服务器

float modff(float x, float *iptr);

long double modfl(long double x, long double *iptr);

extended modfw(extended x, extended *iptr);

quad modfq(quad x, quad *iptr);

说明

modf() 函数将把参数 *x* 分解为整数部分和小数部分，且每个部分与参数具有相同的符号。它将在 *iptr* 所指向的对象中以 **double** 格式存储整数部分。

仅适用于 **Integrity** 服务器

modff() 是 **float** 形式的 **modf()** ；它采用 **float** 和 **float *** 参数，并返回 **float** 结果。

modfl() 是 **long double** 形式的 **modf()** ；它采用 **long double** 和 **long double *** 参数，并返回 **long double** 结果。

modfw() 是 **extended** 形式的 **modf()** ；它采用 **extended** 和 **extended *** 参数，并返回 **extended** 结果。

modfq() 等效于 HP-UX 系统上的 **modfl()** 。

用法

（对于 Integrity 服务器）要使用 **modff()** ，请使用缺省的 **-Ae** 选项或 **-Aa** 选项进行编译。

（对于 Integrity 服务器）要使用 **modfl()** 、 **modfw()** 或 **modfq()** ，请使用缺省的 **-Ae** 选项或使用 **-Aa** 和 **-D_HPUX_SOURCE** 选项进行编译。

（对于 Integrity 服务器）要使用 **modfw()** 或 **modfq()** ，也可以使用 **-fpwidetypes** 选项进行编译。

要使用这些函数，请确保程序包含 **<math.h>** ，并通过在编译程序或链接程序命令行上指定 **-lm** 链接到数学库。

有关详细信息，请参阅《HP-UX floating-point guide for HP Integrity servers》（位于以下站点：**http://www.hp.com/go/fp** ）。

返回值

modf(±*x*,*iptr***)** 将返回与 *x* 具有相同符号的结果。

modf(±Inf,*iptr***)** 将返回 ±0 并在 *iptr* 所指向的对象中存储 ±Inf 。

modf(NaN,*iptr***)** 将在 *iptr* 所指向的对象中存储 NaN 并返回 NaN。

错误

没有定义任何错误。

另请参阅

frexp(3M)、 ldexp(3M)、 rint(3M)、 trunc(3M)、 scalb(3M)、 scalbln(3M)、 scalbn(3M)、 math(5)。

符合的标准

modf() ： SVID3、XPG4.2、ANSI C、ISO/IEC C99 （包括附件 F “IEC 60559 floating-point arithmetic”）

modff() 、 **modfl()** ： ISO/IEC C99（包括附件 F“IEC 60559 floating-point arithmetic”）

名称

monitor( ) - 准备执行配置文件

概要

```
#include <mon.h>

void monitor(
    void (*lowpc)(),
    void (*highpc)(),
    WORD *buffer,
    int bufsize,
    int nfunc
);
```

说明

由 **cc -p** 自动创建的可执行程序包括调用带有缺省参数的 **monitor()**；除良好地控制配置处理之外，不需要显式调用 **monitor()**。

monitor() 是 *profil*(2) 的接口， *lowpc* 和 *highpc* 是两个函数的地址： *buffer* 是 *bufsize* WORDs 的数组（用户提供的）的地址（定义在 **<mon.h>** 头文件中）。该地址应该适当对齐以转换为 **<mon.h>** 中的类型 **struct hdr** 和 **struct cnt**。

monitor() 负责在缓冲区中记录程序计数器的定期采样值的直方图，以及某些函数的调用计数的直方图。采样的低位地址是 *lowpc* 的地址，而高位地址正好在 *highpc* 之下。使用 *monitor* 时 *lowpc* 不必为 0。保持调用计数不超过 *nfunc*；只记录使用 **cc** 的配置处理选项 **-p** 编译的函数的调用（当还使用 **cc -p** 记录调用计数时，提供 C 库和数学库）。

由于结果非常重要，尤其是在大量使用小例行程序时，建议缓冲区只比采样的位置范围小几倍。

要对整个程序进行配置处理，这已够用了。

```
extern etext;
    ...
monitor ((int (*)())2, ((int(*)())& etext, buf, bufsize, nfunc);
```

etext 正好在程序文本之上（请参阅 *end*(3C)）。

要停止执行监视并将结果写入文件 **mon.out**，请使用

```
monitor ((int (*)())0, (int(*)())0, 0, 0, 0);
```

然后使用 *prof*(1) 来检查这些结果。

文件

/usr/lib/libc.a
/usr/lib/libm.a
mon.out

另请参阅

cc_bundled(1)、prof(1)、profil(2)、end(3C)。

符合的标准

monitor()：SVID2、SVID3、XPG2

名称

mount - 跟踪远程挂接的文件系统

概要

#include <rpcsvc/mount.h>

说明

程序号

MOUNTPROG

以下是所提供的 **xdr** 例行程序:

```
xdr_exportbody(xdrs, ex)
  XDR *xdrs;
  struct exports *ex;

xdr_exports(xdrs, ex);
  XDR *xdrs;
  struct exports **ex;

xdr_fhandle(xdrs, fh);
  XDR *xdrs;
  fhandle_t *fp;

xdr_fhstatus(xdrs, fhs);
  XDR *xdrs;
  struct fhstatus *fhs;

xdr_groups(xdrs, gr);
  XDR *xdrs;
  struct groups *gr;

xdr_mountbody(xdrs, ml)
  XDR *xdrs;
  struct mountlist *ml;

xdr_mountlist(xdrs, ml);
  XDR *xdrs;
  struct mountlist **ml;

xdr_path(xdrs, path);
  XDR *xdrs;
  char **path;
```

Procs

MOUNTPROC_MNT **xdr_path** 的参数；返回 **fhstatus** 。要求 UNIX 身份验证。

MOUNTPROC_DUMP

没有参数；返回 **struct mountlist**

MOUNTPROC_UMNT

xdr_path 的参数；没有结果。要求 UNIX 身份验证。

MOUNTPROC_UMNTALL

没有参数；没有结果。要求 UNIX 身份验证。卸除发送方的所有远程挂接。

MOUNTPROC_EXPORT

没有参数；返回 **struct exports**

MOUNTPROC_EXPORTALL

没有参数；返回 **struct exports**

版本

MOUNTVERS_ORIG

结构

```
struct mountlist {          /* what is mounted */
   char *ml_name;
   char *ml_path;
   struct mountlist *ml_nxt;
   };

struct fhstatus {
   int fhs_status;
   fhandle_t fhs_fh;
   };

/*
 * List of exported directories
 * An export entry with ex_groups
 * NULL indicates an entry which is exported to the world.
 */

struct exports {
   dev_t         ex_dev;          /* dev of directory */
   char         *ex_name;         /* name of directory */
   struct groups  *ex_groups;     /* groups allowed to mount this entry */
   struct exports *ex_next;
   };
```

```
struct groups {
  char        *g_name;
  struct groups  *g_next;
  };
```

作者

mount(3N) 由 Sun Microsystems, Inc. 开发。

另请参阅

mount(1M)、 mountd(1M)、 showmount(1M)。

名称

move 和 wmove — 窗口光标位置函数

概要

#include <curses.h>

int move(int *y***, int** *x***);**

int wmove(WINDOW ****win*, int** *y***, int** *x***);**

说明

move() 和 **wmove()** 函数将与当前或指定窗口关联的光标移到 (*y*, *x*) 相对于该窗口的初始位置。在执行下一次刷新之前，该函数不移动终端的光标。

返回值

成功完成后，这些函数返回 OK。否则，它们返回 ERR。

错误

没有定义任何错误。

另请参阅

doupdate(3X)、 <curses.h>。

历史变更记录

在 X/Open Curses 第 2 期中首次发布。

X/Open Curses 第 **4** 期

为清楚起见，重新编写了该条目。

名称

mblen( )、mbtowc( )、mbstowcs( )、wctomb( )、wcstombs( ) - 多字节字符和字符串转换

概要

#include <stdlib.h>

int mblen(const char *s, size_t n);

int mbtowc(wchar_t *__restrict pwc, const char *__restrict s, size_t n);

int wctomb(char *s, wchar_t wchar);

size_t mbstowcs(wchar_t *__restrict pwcs, const char *__restrict s, size_t n);

size_t wcstombs(char *__restrict s, const wchar_t *__restrict pwcs, size_t n);

说明

多字节字符由表示字符编码中的“整个”字符的一个或多个字节组成。宽字符（类型为 **wchar_t** ）由固定数量的字节组成，这些字节的代码值可以表示字符编码中的任何字符。

mblen() 确定 *s* 指向的多字节字符中的字节数。等效于：

mbtowc((wchar_t *)0, s, n);

如果 *s* 是空指针，则 *mblen* 将根据多字节字符编码是否包括与状态有关的编码，分别返回非零值或零值。由于 HP-UX 当前支持的字符编码都不是与状态有关的，因此在这种情况下，始终返回零值。但是，为了最大程度地实现与其他系统之间的可移植性，应用程序不应依赖于此返回值。

在 *s* 不是空指针的情况下，如果后面的 *n* 个或更少的字节构成了一个有效的多字节字符，则 *mblen* 将返回多字节字符中的字节数；如果这些字节不能构成有效的多字节字符，则返回 -1。如果 *s* 指向空字符，则 *mblen* 将返回 0。

mbtowc() 确定 *s* 指向的多字节字符中的字节数，再确定与此多字节字符对应的 **wchar_t** 类型值的代码，然后将此代码存储在 *pwc* 指向的对象中。与空字符对应的代码的值为零。从 *s* 指向的字符开始最多检查 *n* 个字符。

如果 *s* 是空指针，则 **mbtowc()** 将根据多字节字符编码是否包括与状态有关的编码，分别返回非零值或零值。由于 HP-UX 当前支持的字符编码都不是与状态有关的，因此在这种情况下，始终返回零值。但是，为了最大程度地实现与其他系统之间的可移植性，应用程序不应依赖于此返回值。

在 *s* 不是空指针的情况下，如果后面的 *n* 个或更少的字节构成了一个有效的多字节字符，则 **mbtowc()** 将返回已转换的多字节字符中的字节数；如果这些字节不能构成有效的多字节字符，则返回 -1。如果 *s* 指向空字符，则 **mbtowc()** 将返回 0。返回的值始终不大于 *n* 或 **MB_CUR_MAX** 宏的值。

wctomb() 确定要表示对应于其值为 *wchar* 的代码的多字节字符所需的字节数，然后将多字节字符的表示形式存储在 *s* 指向的数组对象中。最多可存储 **MB_CUR_MAX** 个字符。

如果 *s* 是空指针，则 **wctomb()** 将根据多字节字符编码是否包括与状态有关的编码，分别返回非零值或零值。由于 HP-UX 当前支持的字符编码都不是与状态有关的，因此在这种情况下，始终返回零值。但是，为了最大程度地实现与其他系统之间的可移植性，应用程序不应依赖于此返回值。

如果 *s* 不是空指针，则 **wctomb()** 将返回与 *wchar* 的值对应的多字节字符中的字节数；如果 *wchar* 的值不对应于有效的多字节字符，则返回 -1。返回的值始终不大于 **MB_CUR_MAX** 宏的值。

mbstowcs() 将 *s* 指向的数组中的多字节字符序列转换为对应代码的序列，然后将这些代码存储到 *pwcs* 指向的数组中，以便在存储了 *n* 个代码或值为零的代码（转换的空字符）后停止操作。转换每个多字节字符的方法类似于调用 **mbtowc()** 。在 *pwcs* 指向的数组中修改的元素不超过 *n* 个。

如果遇到无效的多字节字符，则 **mbstowcs()** 将返回 (size_t)－1。否则， **mbstowcs()** 将返回已修改的数组元素的数量，不包括结尾零代码（如果有的话）。如果返回的值为 *n* ，则数组不是以空字符或零结尾。如果 *pwcs* 是空指针，则 **mbstowcs()** 将返回宽字符代码数组所需的元素数。

wcstombs() 将与 *pwcs* 指向的数组中的多字节字符对应的代码序列转换为多字节字符序列，然后将它们存储到 *s* 指向的数组中。以便某个多字节字符超出了 *n* 个总字节数的限制，或者存储了空字符后停止操作。转换每个代码的方法类似于调用 **wctomb()** 。在 *s* 指向的数组中修改的字节不超过 *n* 个。

如果遇到与有效的多字节字符不对应的代码，则 **wcstombs()** 将返回 (size_t)－1。否则， **wcstombs()** 将返回已修改的字节数，不包括结尾空字符（如果有的话）。如果返回的值为 *n* ，则数组不是以空字符或零结尾。如果 *s* 是空指针，则 **wcstombs()** 将返回字符数组所需的字节数。

外部语言环境影响

语言环境

LC_CTYPE 类别可确定多字节字符和字符串函数的行为。

错误

如果满足下列条件，则 **mblen()** 、 **mbstowcs()** 、 **mbtowc()** 、 **wcstombs()** 和 **wctomb()** 可能失败，同时将设置 **errno** ：

[EILSEQ] 已找到无效的多字节序列或宽字符代码。

警告

除 ASCII 字符外，宽字符（类型为 **wchar_t** ）的代码值特定于 **LC_CTYPE** 环境变量指定的有效语言环境。这些值可能无法与通过指定目前支持，或者将来可能支持的其他语言环境而获取的值兼容。建议不要使用宽字符常量和宽字符串文字（请参阅《C Reference Manual》），也不要将宽字符代码值存储在文件或设备中，原因是将来的标准可能要求更改宽字符的代码赋值。但是，可以放心地使用与 ASCII 代码集的字符对应的宽字符常量和宽字符串文字，这是因为它们的值肯定与其 ASCII 代码集的值相同。

作者

本条目中的多字节函数由 OSF 和 HP 开发。

另请参阅

setlocale(3C)、 wctype(3C)、 thread_safety(5)、 glossary(9)。

符合的标准

mblen()：AES、SVID3、XPG4、ANSI C

mbstowcs()：AES、SVID3、XPG4、ANSI C

mbtowc()：AES、SVID3、XPG4、ANSI C

wcstombs()：AES、SVID3、XPG4、ANSI C

wctomb()：AES、SVID3、XPG4、ANSI C

名称

mvcur - 将光标移动命令输出到终端

概要

#include <curses.h>

int mvcur(int *oldrow*, **int** *oldcol*, **int** *newrow*, **int** *newcol*);

说明

mvcur() 函数将向终端输出一个或多个命令，用于将终端的光标移动到终端屏幕上的绝对位置 (*newrow*, *newcol*) 处。 (*oldrow*, *oldcol*) 参数指定前一光标位置。对于不提供基于坐标的移动命令的终端而言，指定前一光标的位置是必要的。在提供这些命令的终端上，Curses 可以选择更有效的方式来根据前一位置移动光标。如果 (*newrow*, *newcol*) 对于所用的终端而言为无效地址，则 **mvcur()** 将失败。如果 (*oldrow*, *oldcol*) 与 (*newrow*, *newcol*) 相同，则 **mvcur()** 可以成功执行，而不采取任何操作。如果 **mvcur()** 输出光标移动命令，则它将更新与终端上的光标位置有关的信息。

返回值

成功完成后， **mvcur()** 将返回 OK。否则，它将返回 ERR。

错误

未定义任何错误。

实际应用信息

使用 **mvcur()** 后，Curses 所维护的终端状态的模式可能与终端的实际状态并不匹配。在恢复 Curses 的常规用法之前，应用程序应当处理并刷新窗口。

另请参阅

doupdate(3X)、 is_linetouched(3X)、 <curses.h>。

历史变更记录

在 X/Open Curses 第 4 期中首次发行。

名称

mvderwin — 定义窗口坐标转换

概要

#include <curses.h>

int mvderwin(WINDOW **win*, int *par_y*, int *par_x*);

说明

mvderwin() 函数指定了字符的映射方式。此函数将标识指定窗口的父窗口的一个映射区域，其大小与指定窗口的大小相同，且原点位于父窗口的 (*par_y*, *par_x*) 坐标位置。

- 刷新指定的窗口时，在终端显示区域的此窗口中所显示的字符即来自该映射区域。
- 引用所指定窗口中的字符时将获取或修改映射区域中的字符。

也就是说， **mvderwin()** 定义了从映射区域的各个位置到所指定窗口中的相应位置（距离原点的偏移量具有相同的 *y*、*x* 值）的坐标转换关系。

返回值

成功完成后， **mvderwin()** 将返回 OK。否则，它将返回 ERR。

错误

未定义错误。

另请参阅

derwin(3X)、 doupdate(3X)、 dupwin(3X)、 <curses.h>。

历史变更记录

在 X/Open Curses 第 4 期中首次发布。

名称

mvprintw、mvwprintw、printw 和 wprintw — 在窗口打印格式化的输出

概要

#include <curses.h>

int mvprintw(int *y*, **int** *x*, **char ****fmt*, **...);**

int mvwprintw(WINDOW ****win*, **int *y*, **int** *x*, **char ****fmt*, **...);**

int printw(char ****fmt*, **...);

int wprintw(WINDOW ****win*, **char ****fmt*, **...);

说明

mvprintw() 、 **mvwprintw()** 、 **printw()** 和 **wprintw()** 函数类似于 **printf()** 。这些函数的作用与以下操作相同：使用 **sprintf()** 格式化字符串，然后使用 **waddstr()** 将多字节字符串添加到当前或指定光标位置的当前或指定窗口。

返回值

成功完成后，这些函数返回 OK。否则，它们返回 ERR。

错误

没有定义任何错误。

另请参阅

addnstr(3X)、 fprintf() （位于《X/Open System Interfaces and Headers, Issue 4, Version 2》规范中）、 <curses.h>。

历史变更记录

在 X/Open Curses 第 2 期中首次发布。

X/Open Curses 第 **4** 期

为了清楚起见，重新编写了此条目，并将其名称从 **printw()** 更改为 **mvprintw()** 。

名称

mvscanw、mvwscanw、scanw 和 wscanw — 从窗口转换格式化的输入

概要

#include <curses.h>

int mvscanw(int *y***, int** *x***, char ****fmt***, ...);**

int mvwscanw(WINDOW ****win*, int** *y***, int** *x***, char ****fmt***, ...);**

int scanw(char ****fmt*, ...);**

int wscanw(WINDOW ****win*, char ****fmt***, ...);**

说明

这些函数类似于 **scanf()** 。它们的作用与以下操作相同：调用 **mvwgetstr()** 以从当前或指定光标位置的当前或指定窗口获取多字节字符串，然后使用 **sscanf()** 来解释和转换该字符串。

返回值

成功完成后，这些函数返回 OK。否则，它们返回 ERR。

错误

没有定义任何错误。

另请参阅

getnstr(3X)、 printw(3X)、 fscanf() （位于《X/Open System Interfaces and Headers, Issue 4, Version 2》规范中）、 wcstombs() （位于《X/Open System Interfaces and Headers, Issue 4, Version 2》规范中）、 <curses.h>。

历史变更记录

在 X/Open Curses 第 2 期中首次发布。

X/Open Curses 第 4 期

为了清楚起见，重新编写了此条目，并将其名称从 **scanw()** 更改为 **mvscanw()** 。

名称

mvwin — 移动窗口

概要

#include <curses.h>

int mvwin(WINDOW *_win_, int _y_, int _x_);

说明

mvwin() 函数将指定的窗口移动到其初始位置 (*y*, *x*)。如果移动导致窗口的任一部分的扩展超出了屏幕的任何边界，则此函数将失败，并且不移动此窗口。

返回值

成功完成后， **mvwin()** 返回 OK。否则返回 ERR。

错误

没有定义任何错误。

实际应用信息

应用程序不应该通过调用 **mvwin()** 来移动子窗口。

另请参阅

derwin(3X)、 doupdate(3X)、 is_linetouched(3X)、 <curses.h>。

历史变更记录

在 X/Open Curses 第 2 期中首次发布。

X/Open Curses 第 **4** 期

为清楚起见，重新编写了该条目。