HP-UX 参考手册

第 7 节：设备专用文件
第 9 节：常规信息
索引

HP-UX 11i v3

第 10 卷（共 10 卷）

生产部件号：B2355-91046
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 10 卷
目录

第 10 卷
目录

目录
第 10 卷

第 7 节：设备专用文件

条目名称(节)：名称 **说明**

目录
第 10 卷

第 9 节：一般信息

名称

intro - 设备专用文件简介

说明

本节介绍了用于访问 HP 外围设备和设备驱动程序的设备专用文件 (DFS) 和硬件路径。条目名称通常派生自所介绍设备的类型（磁盘、磁带、终端等），而不是设备专用文件或设备驱动程序本身的名称。硬件设备与对应的 HP-UX 设备驱动程序的特性将在适当的位置论述。

设备类型

设备可分为两类设备访问模式，原始模式和 块模式。原始模式或字符模式设备（例如，行式打印机）以不缓冲数据流的形式传输数据并使用字符设备专用文件。

块模式设备，顾名思义，通过系统普通缓冲机制以块的形式传输数据。块设备使用块设备专用文件，并且还可能具有一个字符设备接口。

设备文件命名约定

使用专用文件守护程序 **sfd** 自动创建，或使用 **insf** 、 **mknod** 或 **mksf** 命令显式创建设备专用文件时，该文件的名称会与设备相关联。创建设备专用文件时，建议使用以下标准命名约定:

/dev/*subdir*/*class*#[*options*]

subdir 可选的设备类子目录（例如， **rdisk** 表示磁盘原始设备专用文件， **disk** 表示磁盘块设备专用文件， **rtape** 表示原始磁带设备）。

class 设备的类，例如 **tape** 、 **disk** 或 **lan** 。

操作系统分配给设备的实例编号。每个设备类都有自己的实例编号集，因此，类与实例编号的每个组合才能准确指明一个设备。

options 其他限定符，例如，磁盘分区 (**p#**) 、磁带设备的磁带密度选择或磁光介质的表面规格。

每个设备类型的的命名约定分别在各自的联机帮助页条目中进行介绍。

Legacy 海量存储设备专用文件具有不同的命名约定，该约定会对硬件路径进行编码；该内容在“设备文件类型（海量存储设备）”一节进行介绍。

硬件路径

硬件路径信息以及类名称和实例编号均可由 **ioscan** （请参阅 *ioscan*(1M) ）的输出派生得来。有三种不同类型的设备路径： **Legacy** 硬件路径、 **lunpath** 硬件路径和 **LUN** 硬件路径。所有这三种路径都是硬件组件的数字字符串，从系统总线地址到设备地址按顺序表示。通常每个数字代表硬件组件在设备路径中的位置。

Legacy 硬件路径由用斜线 (**/**) 字符分隔的一系列的总线联结地址组成，这些地址最终指向主机总线适配器 (HBA)。在 HBA 下，其他地址元素由句点 (**.**) 字符进行分隔。所有的元素均以十进制表示。缺省情况下，这是由大多数设备的 **ioscan** 命令输出的格式。 **0/0/2/0.1.7.0** 就是 Legacy 硬件路径的一个示例。

lunpath 硬件路径用于海量存储设备，也称为逻辑单元 (LUN)。它与 Legacy 硬件路径的格式相同，等同于 HBA。在 HBA 下，其他元素是以十六进制输出的。前导元素代表与传输相关联的目标地址，而最终的元素是

LUN 地址，该地址是目标报告的 LUN 标识符的 64 位表示。如果指定了 **-N** 选项，则该格式由 **ioscan** 命令输出。字符串 **0/2/1/0.0x50001fe1500170ac.0x4017000000000000** 是 lunpath 硬件路径的一个示例。

请注意，HBA 下的地址元素可能与物理硬件地址不对应；应将 lunpath 硬件路径看作 句柄，而不是设备的物理路径。

LUN 硬件路径是可以代表单个海量存储设备的多个硬件路径的虚拟路径。有一个含有 64000 地址的虚拟总线联结节点（也称为 虚拟根节点），而不是一系列指向 HBA 的总线联结地址。在该虚拟根节点下的地址是由虚拟总线地址和虚拟 LUN 标识符组成的，由斜线 (*/*) 字符分隔。字符串 **64000/0xfa00/0x22** 是 LUN 硬件路径的一个示例。

作为虚拟路径，LUN 硬件路径仅仅是 LUN 的句柄，而并不表示 LUN 的物理位置；确切的说，它可链接到 LUN 的全球通用标识符 (WWID)。这样，如果添加了新的设备物理路径、现有物理路径被删除或者物理路径有任何更改，它都会保持相同的状态。此 LUN 绑定在重新引导过程中会保持不变，但是并不能保证在安装 — 期间保持不变；也就是说，重新安装系统或安装相同配置的系统可能会创建不同的 LUN 硬件路径集。

设备文件类型（海量存储设备）

海量存储设备（例如磁盘设备和磁带设备）有两种类型的设备文件： 持久性设备专用文件和 **Legacy** 设备专用文件。这两种类型的设备文件都可用于独立访问海量存储设备，并可在同一个系统中共存。

持久性设备专用文件与 LUN 硬件路径相关联，从而可明确支持 Agile 寻址和多路径。换句话说，如果 LUN 从一个 HBA 移动到另一个 HBA、从一个交换器（或集线器）端口移动到另一个端口、通过不同的目标端口到主机进行表示或者使用多个硬件路径进行配置时，持久性设备专用文件不会更改。和 LUN 硬件路径一样，在重新引导期间到设备的设备专用文件绑定保持不变，但是并不能保证在安装过程中会保持不变。设备专用文件名是根据上面的标准命名约定进行命名的，并且次设备号中不包含硬件路径信息。

对于特定物理硬件路径， **Legacy** 设备专用文件被锁定，并且不支持 Agile 寻址。这种设备专用文件在设备文件名和次设备号中包含诸如以下硬件路径信息： SCSI 总线、目标和 LUN。需要特别指出的是，设备专用文件名的 *class* 和 *instance* 部分表示硬件路径信息，格式为 **c#t#d#** ，含义如下:

c# 操作系统分配给接口卡的实例编号，以十进制表示。该编号是 0 到 255 范围内的十进制整数。实例编号与物理插槽编号之间无直接关系。

t# 远程总线上的目标地址（例如 SCSI 地址）。通常为 0 到 15 范围内的十进制整数。

d# 目标地址的设备单元号（例如 SCSI 设备中的 LUN）。通常为 0 到 7 范围内的十进制整数。

请注意，传统命名约定最多支持 256 条外部总线和 32768 个 LUN。超出这些限制的具有海量存储设备的系统将无法使用传统命名约定对其进行寻址。

不推荐使用 Legacy 设备专用文件，并且在 HP-UX 的将来版本中将删除对它们的支持。

查看海量存储

随着持久性设备专用文件和 Legacy 设备专用文件的出现，处理海量存储的命令可在 I/O 系统的两个 视图中选择。表示 **Legacy** 视图的命令使用 Legacy 设备专用文件和 Legacy 硬件路径。 **Agile** 视图使用持久性设备专用文件、lunpath 硬件路径和 LUN 硬件路径。

根据命令，可以显示两种视图，或者可由命令选项或环境变量控制视图的选择。例如， **ioscan** 命令在缺省情况下显示 Legacy 视图，如果指定了 **-N** 选项则切换为 Agile 视图。

举例

示例 **1**

下面是一个持久性设备专用文件名的示例：

/dev/disk/disk3

其中， **disk** 表示块磁盘访问， **disk3** 表示设备类 **disk** 和实例编号 3。缺少 **p#** 表示要访问整个磁盘；有关详细信息，请参阅 *disk*(7) 。

示例 **2**

下面是一个 Legacy 磁盘设备专用文件名的示例：

/dev/dsk/c0t6d0s2

其中， **dsk** 表示块磁盘访问， **c0t6d0** 表示在接口卡实例 0、目标地址 6 和单元 0 的逻辑磁盘访问。 **s2** 表示对磁盘扇区 2 的访问。

示例 **3**

以下是持久性磁带设备专用文件名的示例：

/dev/rtape/tape4QIC150

其中， **rtape** 表示原始磁带， **tape4** 表示磁带设备实例编号 4，而 **QIC150** 将磁带格式识别为 QIC150；有关详细信息，请参阅 *mt*(7) 。

警告

不推荐使用 Legacy 设备专用文件支持，并且在 HP-UX 的将来版本中将删除该支持。

另请参阅

insf(1M)、 ioscan(1M)、 lssf(1M)、 mksf(1M)、 mknod(1M)、 hier(5) 和 introduction(9)。

《HP 系统管理员指南》，位于 **http://docs.hp.com** 。

《The Next Generation Mass Storage Stack》白皮书，位于：
http://docs.hp.com/en/netsys.html#Storage%20Area%20Management 。

名称

arp - 地址解析协议

说明

ARP 是用来在 DARPA Internet 和硬件工作站地址之间进行动态映射的协议。所有 LAN 驱动程序都使用该协议。

ARP 缓存 Internet 到硬件工作站地址映射。当接口请求不在缓存中的地址的映射时，如果已为该接口启用了 **ether** 封装方法，则 ARP 将请求映射的消息排队，并在相关联的网络上广播消息请求地址映射。如果有响应，则缓存新映射并发送任何未决消息。在等待映射请求响应时， ARP 最多可排队一个数据包；只有最新“发送”的数据包才能被保留。

为了便于与不使用 ARP 的系统通信，提供 **ioctl** 调用以输入和删除 Internet 到硬件工作站地址表条目。

实际应用信息:

```
#include <sys/ioctl.h>
#include <sys/socket.h>
#include <net/if.h>
#include <netinet/if_ether.h>
struct arpreq arpreq;

ioctl(s, SIOCSARP, (caddr_t)&arpreq);
ioctl(s, SIOCGARP, (caddr_t)&arpreq);
ioctl(s, SIOCDARP, (caddr_t)&arpreq);
```

每个 **ioctl** 调用采用同样的结构作为参数。 **SIOCSARP** 设置 ARP 条目， **SIOCGARP** 获取 ARP 条目， **SIOCDARP** 删除 ARP 条目。这些 **ioctl** 调用可以应用于任何套接字描述符 *s* ，但是只有超级用户才能使用。 **arpreq** 结构包括:

```
/*
 * ARP ioctl request
 */
struct arpreq {
    int32_t ifindex;
    int32_t arp_flags;          /* flags  */
    int32_t arp_hw_addr_len;  /* hardware address length */
    struct  sockaddr arp_pa;    /* protocol address */
    struct  sockaddr arp_ha;    /* hardware address */
    u_char  arp_pad[242];       /* buffer for link specific info. */
};
/*  arp_flags field values */
#define ATF_COM             0x02        /* ARP on ether */
#define ATF_PERM            0x04        /* permanent entry */
#define ATF_PUBL            0x08        /* publish entry */
```

```
#define ATF_SNAPFDDI      0x200    /* SNAP - FDDI */
#define ATF_SNAP8025      0x400    /* SNAP - 8025 */
#define ATF_IEEE8025      0x800    /* IEEE - 8025 */
#define ATF_FCSNAP        0x4000   /* Fibre Channel SNAP */
```

arp_pa **sockaddr** 的地址系列必须是 **AF_INET** ；对于 *arp_ha* **sockaddr** ，地址系列必须是 **AF_UNSPEC** 。能写入的标志位只有 **ATF_PERM** 和 **ATF_PUBL** 。光纤通道主机仅支持 **ATF_PERM** 标志。 **ATF_PERM** 导致条目成为永久性条目。 **ATF_PUBL** 指定 ARP 代码应该响应来自于其他计算机的指定主机的 ARP 请求。这允许主机充当 *ARP server* ，它在使 ARP计算机与非 ARP 计算机进行通信方面可能非常有用。

ARP 被动监视充当本地主机的主机（即，响应本地主机地址的 ARP 映射请求的主机）。

诊断信息

duplicate IP address!! sent from ethernet address: %x:%x:%x:%x:%x:%x.

控制台屏幕上输出的该消息表示 ARP 已在本地网络发现另一台主机，该主机响应它自己的 Internet 地址的映射请求。

警告

要启用 **ether** 封装方法，请使用 **ifconfig** 命令（请参阅 *ifconfig*(1M) ）。

作者

ARP 由加州大学伯克利分校开发。

另请参阅

ifconfig(1M)、inet(3N)、lan(7)、arp(1M)。

《An Ethernet Address Resolution Protocol》， RFC826 ，Dave Plummer，网络信息中心， SRI 。

名称

autochanger：schgr、eschgr - 介质更换器设备的 SCSI 接口

说明

自动更换器是一种 SCSI 海量存储设备，它由一个机械更换器设备、一个或多个数据转换设备（例如光盘驱动器）和用于数据存储的介质（例如光盘）组成。机械更换器可以在自动更换器内的存储和使用位置之间移动介质。

两个介质更换器驱动程序（ **schgr** 或 **eschgr** ）可提供对介质更换器设备的访问权限； **eschgr** 是当前的首选访问方法，提供 **schgr** 是为了实现与旧的程序的兼容。可通过以上驱动程序直接访问机械更换器设备，以便在自动更换器内移动介质。

schgr 和 **eschgr** 介质更换器设备驱动程序遵循介质更换器设备的 SCSI 规范，它们提供一种常规的介质更换器接口，可以为任何机械更换器、自动唱片点唱机、库或自动更换器设备（MO、磁带、 CD-ROM）构建应用程序级驱动程序。

设备命名约定

通过自动更换器驱动程序的设备命名约定，可访问更换器设备。

旧的字符设备文件名驻留在 **/dev/rac** 内。在该目录中，名称是从 **c#t#d#** 设备命名约定（在 *intro*(7) 中已作说明）派生而来的。这些唯一的设备名称由卡实例、SCSI 更换器设备的目标地址和 SCSI 更换器设备的 LUN 确定。

持久性设备文件名的字符设备格式为 **/dev/rchgr/autochx** 。卡实例、目标地址和 LUN 将不会再在持久性设备文件名自身中进行编码（请参阅 *intro(7)* ）。

主次设备号说明

下面显示 **schgr** 更换器驱动程序使用 Legacy 设备文件访问更换器设备所用的位分配（ **dev_t** 格式）：

0-7	8-15	16-19	20-22	
MAJOR	INSTANCE	TARGET	LUN	

“MAJOR”是指相应驱动程序的主设备号；“INSTANCE”是指装载器设备附加到的 SCSI 接口的卡实例；“TARGET”是指更换器设备的 SCSI 目标地址；“LUN”是指更换器设备的 SCSI LUN。

设备号中的所有字段都以十六进制表示法指定。请注意，此次设备号不支持硬分区（扇区）。如果需要，可以通过 LVM 软分区方案实现分区。

注释：从 HP-UX 11i v3 发行版开始，动态分配更换器驱动程序使用的主设备号：

以下长列表显示与更换器的设备专用文件名关联的主次设备号：

schgr:

crw-rw-rw- 1 root sys 231 0x015000 Apr 22 10:22 /dev/rac/c1t5d0

SCSI 介质更换器设备驱动程序

SCSI 介质更换器设备驱动程序可以在自动更换器内的不同介质位置之间执行移动。每个潜在的介质位置都包括一个特定的元素地址，并且属于以下元素类型之一：

storage 一个位置，用于保存当前未占用的介质单元。通常，大多数介质都位于此类型的元素中。

import/export 一个位置，用于从设备插入和删除介质。将某个介质单元移至此类型的位置将执行有效的弹出操作。将某个介质单元移出此类型的位置属于加载操作。

data transfer 一个位置，用于访问介质数据。这通常是在由介质更换器设备处理的介质上读取和（或）写入数据的设备的位置。移至此类型的位置属于物理介质安装操作。移出此类型的位置属于物理介质卸除操作。

media transport 用于介质移动的位置。通常，仅在实际介质移动期间，介质才暂时位于此类型的元素中。

更换器控制请求

下列读写控制函数及结构定义位于 **<sys/scsi.h>** 中：

```
#define SIOC_INIT_ELEM_STAT             _IO('S', 51)
#define SIOC_ELEMENT_ADDRESSES          _IOW('S', 52, struct element_addresses)
#define SIOC_ELEMENT_STATUS             _IOWR('S', 53, struct element_status)
#define SIOC_RESERVE                    _IOW('S', 54, struct reservation_parms)
#define SIOC_RELEASE                    _IOW('S', 55, struct reservation_parms)
#define SIOC_MOVE_MEDIUM                _IOW('S', 56, struct move_medium_parms)
#define SIOC_EXCHANGE_MEDIUM            _IOW('S', 57, struct exchange_medium_parms)

/* structure for SIOC_ELEMENT_ADDRESSES ioctl */
struct element_addresses {
      unsigned short  first_transport;
      unsigned short  num_transports;
      unsigned short  first_storage;
      unsigned short  num_storages;
      unsigned short  first_import_export;
      unsigned short  num_import_exports;
      unsigned short  first_data_transfer;
      unsigned short  num_data_transfers;
};

/* structure for SIOC_ELEMENT_STATUS ioctl */
struct element_status {
     unsigned short element;          /* element address */
```

```
        unsigned int  resv1:2;
        unsigned int  import_enable:1;     /* allows media insertion (load) */
        unsigned int  export_enable:1;     /* allows media removal (eject) */
        unsigned int  access:1;            /* transport element accessible */
        unsigned int  except:1;            /* is in an abnormal state */
        unsigned int  operatr:1;           /* medium positioned by operator */
        unsigned int  full:1;              /* holds a a unit of media */

        unsigned char resv2;
        unsigned char sense_code;          /* info. about abnormal state */
        unsigned char sense_qualifier;     /* info. about abnormal state */

        unsigned int  not_bus:1;           /* transfer device SCSI bus differs */
        unsigned int  resv3:1;
        unsigned int  id_valid:1;          /* bus_address is valid */
        unsigned int  lu_valid:1;          /* lun is valid */
        unsigned int  sublu_valid:1;       /* sub_lun is valid */
        unsigned int  lun:3;               /* transfer device SCSI LUN */

        unsigned char bus_address;         /* transfer device SCSI address */
        unsigned char sub_lun;             /* sub-logical unit number */

        unsigned int  source_valid:1;      /* source_element is valid */
        unsigned int  invert:1;            /* media in element was inverted */
        unsigned int  resv4:6;

        unsigned short source_element;     /* last storage medium location */
        char          pri_vol_tag[36];     /* volume tag (device optional) */
        char          alt_vol_tag[36];     /* volume tag (device optional) */
        unsigned char misc_bytes[168];     /* device specific */
};

/* structure for SIOC_RESERVE and SIOC_RELEASE ioctls */
struct reservation_parms {
        unsigned short  element;
        unsigned char   identification;
        unsigned char   all_elements;
};

/* structure for SIOC_MOVE_MEDIUM ioctl */
```

```
struct move_medium_parms {
        unsigned short  transport;
        unsigned short  source;
        unsigned short  destination;
        unsigned char   invert;
};

/* structure for SIOC_EXCHANGE_MEDIUM ioctl */
struct exchange_medium_parms {
        unsigned short  transport;
        unsigned short  source;
        unsigned short  first_destination;
        unsigned short  second_destination;
        unsigned char   invert_first;
        unsigned char   invert_second;
};
```

SIOC_INIT_ELEM_STAT

使介质更换器设备制作清单。因此，介质更换器设备可确定各个元素地址的状态，包括介质单元的存在与否。这是一个耗时的机械操作。仅当介质更换器发生严重错误时，才需要使用此函数。

SIOC_ELEMENT_ADDRESSES

确定介质更换器设备支持的元素地址。将为每个元素类型指示第一个有效的元素地址及元素数。可将这些元素地址用作源位置参数和目标位置参数。

SIOC_ELEMENT_STATUS

确定元素的状态。将通过 **element** 字段指定请求了状态信息的元素地址。得到的状态数据指示此元素地址中某个介质单元存在与否，以及与此元素地址有关的其他信息。

SIOC_RESERVE 和 **SIOC_RELEASE**

控制对元素地址的访问。根据设备的不同，保留可能会限制介质更换器设备内元素地址的操作员控制权限。如果每个请求器都带有唯一的保留标识，那么可以保留特定的元素地址来处理多个请求器之间的联锁。**all_elements** 字段中的零值可指定应保留或释放单个元素地址。以这种方式保留的某个元素地址，不能由另一个元素地址保留通过不同的保留标识进行保留。 **reservation** 字段指定保留标识。 **element** 字段指定要保留的元素地址。

all_elements 字段中的值“1”指示应保留所有元素地址。由于在保留所有元素地址时， **reservation** 和 **element** 字段没有意义，因此这两个字段应包含零值。保留所有元素地址的作用主要是为了限制操作员控制权限。

SIOC_MOVE_MEDIUM 和 **SIOC_EXCHANGE_MEDIUM**

重定位介质单元。这可能会导致介质加载、弹出或简单重定位，具体取决于源元素类型和目标元素类型。可以在 **invert** 、 **invert_first** 或 **invert_second** 字段中使用值“1”来“翻转”介质。

SIOC_EXCHANGE_MEDIUM 读写控制可重定位两个不同的介质单元。其中一个介质单元从 **source** 字段指定的元素移至 **first_destination** 字段指定的元素，另一个介质单元从 **first_destination** 字段指定的元素移至 **second_destination** 字段指定的元素。在带有多个更换器机制的自动更换器中，或者介质临时区域中，如果 **source** 和 **second_destination** 字段相同，则会发生交换。

缺省配置

缺省情况下，系统配置 (**/stand/system**) 文件中不包括 **schgr** 和 **schgr** 。

举例

以下示例使用 **SIOC_ELEMENT_ADDRESSES** 和 **SIOC_ELEMENT_STATUS ioctl** 函数获取与 自动更换器设备中的驱动器有关的总线地址信息：

```
int                     last_drive_el;
struct element_addresses    el_addrs;
struct element_status       el_stat; drive[1024];
int fd = -1, error = 0, i = 0;



fd = open("/dev/rchgr/autoch0",O_RDWR);
if ((error = ioctl(fd, SIOC_ELEMENT_ADDRESSES, &el_addrs)) != 0) {
  perror("ioctl: SIOC_ELEMENT_ADDRESSES");
  return -1;
} else {
  last_drive_el = el_addrs.first_data_transfer
            + el_addrs.num_data_transfers - 1;
  for (i = el_addrs.first_data_transfer; i <= last_drive_el; i++) {
    el_stat.element = i;
    if ((error = ioctl(fd, SIOC_ELEMENT_STATUS, &el_stat)) != 0) {
      perror("ioctl: SIOC_ELEMENT_ADDRESSES");
      return -1;
    } else {
      /*
       * You may wish to also check some of the other fields
       * in the el_stat structure to verify that the data is
       * valid.  Fields: el_stat.access (ac accessible),
       * el_stat.except (exception).
       */
      if (! el_stat.not_bus && el_stat.id_valid) {
       drive[i].bus_address = el_stat.bus_address;
        if (! el_stat.lu_valid) {
           drive[i].lun = 0;
```

```
            } else {
               drive[i].lun = el_stat.lun;
            }
          }
        }
      }
    }
```

警告

某些非 HP 介质更换器设备不支持 **SIOC_INIT_ELEM_STAT** 和 **SIOC_ELEMENT_STATUS ioctls** 。

某些早期的介质更换器设备不支持 **SIOC_EXCHANGE_MEDIUM ioctl** 。对于这些设备，如果能够找到适当的临时元素地址，则可以通过多次 **SIOC_MOVE_MEDIUM** 读写控制操作来实现相同的结果。

另请参阅

insf(1M)、mknod(1M)、scsictl(1M)、ioctl(2)、scsi(7)、scsi_ctl(7)、intro(7)。

名称

blmode - 终端块模式接口

说明

此终端接口对目前使用的 *termio*(7) 函数功能进行了增强，从而有效模拟了 MPE 终端驱动程序的功能。最重要的是，它可以添加必要的功能来支持通过 HP 终端进行块模式传输。块模式接口只影响输入处理，而不影响写入请求。将始终按 *termio*(7) 中的说明处理写入请求。在字符模式下，每当键入一个字符时，终端就会将其发送到系统。但在块模式下，每当键入数据时，此数据将被缓冲，并且可能在终端内存中进行本地编辑，然后在按下终端上的 **Enter** 键后，将其作为数据块发送。在块模式数据传输过程中，不回显传入的数据，并且不执行特殊字符处理，但是可以识别数据块终止字符。对于后续字符模式传输，现有的 termio 状态会继续确定回显和字符处理。

块模式协议包括两个部分。第一个部分为块模式握手，其工作方式如下：

- 在读取的起始阶段， *trigger* 字符将发送到终端，以通知此终端，系统正在请求数据块（如果定义了 *trigger* 字符，则在所有读取操作的起始阶段发送此字符，而不论是字符模式读取还是块模式读取。必须为块模块读取定义 *trigger* 字符）。
- 收到 *trigger* 字符后，终端将一直等待，直到用户将数据键入终端的内存，并按下终端上的 **Enter** 键。然后终端将一个 *alert* 字符发送到系统，以通知此系统，终端具有要发送的数据块。
- 系统随后可能会将用户可定义的光标定位或其他数据序列发送到终端。完成此操作后，系统会将另一个 *trigger* 字符发送到终端，并重复此循环。

块模式协议的第二部分为块模式传输。在此数据传输过程中，不回显传入的数据，并且不执行特殊字符处理，但是可以识别数据块终止字符。可以绕过块模式握手，以便在发送第一个 *trigger* 字符后即开始块模式传输。

要防止数据丢失，应当在系统与终端之间使用 XON/XOFF 流量控制。应设置 IXOFF 位，并且终端已适当地进行绑定。如果不使用流量控制，则传入的数据可能会溢出和丢失（注释：某些早期的终端不能正常处理这种流量控制）。

可以混合使用字符模式和块模式数据传输。如果启用了块模式传输，则按块模式传输处理所有传输。如果未启用块模式传输，则按 *termio*(7) 的说明处理字符模式传输。如果未启用块模式传输，但在输入数据中的任何位置收到 *alert* 字符，则传输模式将自动切换到块模式，以执行单一传输。

在完成传输（即终端传输了所有字符）之前，从块模式传输接收数据的读取请求将不会返回。如果通过字节数实现读取，或者发生数据传输错误，则忽略任何后续数据。读取在返回之前，会等待完成数据的传输。

在返回至用户的数据中以及字节数中都包含了数据块终止字符。如果终端执行块模式传输时传输的字节数超出用户请求的字节数，读取将返回请求的字节数，而忽略多余的字节数。用户可以通过检查返回数据的最后一个字符来确定是否忽略了数据。如果最后一个字符不是终止字符，那么表示收到的数据多于请求的数据，因此数据被忽略。

如果需要，可以通过选择 OWNTERM 模式，使应用程序在响应 *alert* 时提供自身的握手机制。选择这种模式后，如果收到 *alert* 字符，驱动程序就会完成读取请求。应用程序发出下一个读取请求后，此驱动程序将发送第二个 *trigger* 。

与块模式一起使用的还有多个专用字符（输入和输出）。下面介绍了用于块模式的这些字符和正常值。这些字符的初始值为 0377，此值可禁用这些字符。

CBTRIG1C (DC1) 是读取请求的起始阶段发送到终端的初始 *trigger* 字符。

CBTRIG2C (DC1) 是收到 *alert* 字符后发送到终端的第二个 *trigger* 字符。

CBALERTC (DC2) 是终端响应第一个 *trigger* 字符后发送的 *alert* 字符。它表示终端可以发送数据块。可以在 *alert* 字符前面加反斜杠 (\) 转义此字符。

CBTERMC (RS) 是完成块模式传输后由终端发送的。它表示计算机数据块的末尾。

适用于块模式的两个 *ioctl*(2) 请求都使用 **<blmodeio.h>** 中定义的 **blmodeio** 结构，此结构包括下列成员：

```
unsigned long   cb_flags;     /* Modes */
unsigned char   cb_trig1c;    /* First trigger */
unsigned char   cb_trig2c;    /* Second trigger */
unsigned char   cb_alertc;    /* Alert character */
unsigned char   cb_termc;     /* Terminating char */
unsigned char   cb_replen;    /* cb_reply length */
char            cb_reply[ ];  /* optional reply */
```

cb_flags 成员控制基本块模式协议：

CB_BMTRANS 0000001 启用强制块模式传输。
CB_OWNTERM 0000002 启用握手机制的用户控制。

仅当已设置 *termio*(7) 中的 **ICANON** 标志时， **CB_BMTRANS** 位才有效。如果清除了 **ICANON** ，则在原始模式下完成所有传输，而不论 **CB_BMTRANS** 位是什么。如果未设置 **CB_BMTRANS** ，将按照 *termio*(7) 中的说明执行输入处理。在此过程中，如果定义了 *alert* 字符，并且可以在输入流中的任何位置检测到此字符，则会刷新输入缓冲区，并调用块模式握手。随后系统向终端发送 *cb_trig2c* 字符，接着执行块模式传输。可以在 *alert* 字符前面加反斜杠 (\) 转义此字符。

如果已设置 **CB_BMTRANS** ，则按块模式传输来处理所有传输。不需要块模式握手，并且读取的数据将按块模式传输数据来处理。仍然可以通过接收某个 *alert* 字符并将其作为第一个收到的字符，来调用块模式握手。如果在设置 **CB_BMTRANS** 位的情况下发出读取请求，则会刷新任何现有的输入缓冲区数据。

如果已设置 **CB_OWNTERM** ，则收到非转义 *alert* 字符后即会终止读取。不执行输入缓冲区刷新，将在读取的数据中返回 *alert* 字符。这允许应用程序代码执行自身的块模式握手。如果位已清除， *alert* 字符会导致使用正常的块模式握手。

初始 **cb_flags** 值的所有位将被清除。

cb_trig1c 字符是读取请求的起始阶段发送到终端的初始 *trigger* 字符。初始值是未定义的 (0377)；即不发送 *trigger* 字符。

cb_trig2c 字符是收到 *alert* 字符后发送到终端的第二个 *trigger* 字符。初始值是未定义的 (0377)。

cb_alertc 字符是终端响应计算机发送的第一个 *trigger* 字符后，发送的 *alert* 字符。它表示终端可以传输数据。初始值是未定义的 (0377)。

cb_termc 字符是完成块模式传输后由终端发送的。它表示计算机数据块的末尾。初始值是未定义的 (0377)。

cb_replen 成员指定 **cb_reply** 数组的字节长度。 **cb_reply** 数组的最大长度为 **NBREPLY** 字节。如果将此长度设置为零，将不使用 *cb_reply* 字符串。此长度的初始设置为零。

cb_reply 数组包含在收到 *alert* 字符之后、但在计算机发送第二个 *trigger* 字符之前所要发出的字符串。任何字符都可以包含在回复字符串中。发送的字符数由 **cb_replen** 指定。 **cb_reply** 数组的最大长度为 **NBREPLY** 字节。 **cb_reply** 数组中所有字符的初始值为空。

在支持进程组控制的系统上，除非针对特定的请求另外作出说明，否则将限制后台进程使用 *ioctl* 请求。尝试从后台进程发出 *ioctl* 请求，将导致此进程阻塞，并且可能会向进程组发送 **SIGTTOU** 信号。

主要的 *ioctl*(2) 调用使用以下格式：

int ioctl(int fildes, int request, struct blmodeio *arg);

使用此格式的请求包括：

CBGETA 获取与块模式接口关联的参数，并将这些参数存储在 *arg* 引用的 *blmodeio* 结构中。允许从后台进程发出此请求。但是，前台进程随后可能会更改信息。

CBSETA 从 *arg* 引用的 *blmodeio* 结构设置与块模式接口关联的参数。更改将立即生效。

返回值

请参考 *read*(2) 、 *write*(2) 和 *ioctl*(2) 。

错误

如果在读取过程中返回了错误值，则可能需要更改用户的缓冲区。在这种情况下，应忽略用户的缓冲区中的数据，原因是这些数据是不完整的。

设置全局变量 *errno* 除了可指示 *read*(2) 、 *write*(2) 和 *ioctl*(2) 中描述的错误外，还可指示以下错误：

[EIO] 在传输块模式数据块的过程中发生了读取错误。

警告

有多种事件会导致由于读取错误而返回 [EIO] 错误。如果发生传输错误、成帧错误、奇偶错误、中断错误和溢出错误，或者内部计时器到期，则读取操作将返回 [EIO] 。计算机发送第二个 *trigger* 字符时将启动内部计时器，计算机收到终止字符时将停止内部计时器。读取时请求的字节数和当前波特率以及额外的十秒钟共同确定此计时器的长度。

作者

blmode 驱动程序由 HP 开发。

另请参阅

termio(7)。

名称

cent - 与 Centronics 兼容的接口

说明

cent 是广泛使用的简单的通信协议，通常与打印机、绘图仪和扫描仪相关联。它是一个八位并行数据接口，具有来自主机的其他控制信号和来自外围设备的状态信号。

cent 接口驱动程序不会进行字符处理；即，它不会解释计算机与外围设备之间传输的数据。因此，所有发送到设备或次设备上接收到的字节都按原样进行处理。 **cent** 接口驱动程序始终以 原始模式运行；因此，任何所需的数据解释必须由用户程序（例如与相应机型文件一起的“lp”后台打印程序）执行。 **cent** 驱动程序支持六种不同的数据传输握手模式。设备专用文件次设备号的最后四位指定所用模式。次设备号格式为：

0x*II000A*

其中，“0x”前缀后的每个字母表示一个十六进制数，如下所示：

II 指定 Centronic 接口的实例编号。

000 始终为零。

A 指定握手模式。握手模式包括：

模式 1 同时使用 ACK 和 BUSY 的自动握手。
次设备号格式： **0x***II0***001**.

模式 2 仅使用 BUSY 的自动握手。
次设备号格式： **0x***II0***002**.

模式 3 用于 ScanJet 的双向读/写。
次设备号格式： **0x***II0***003**.

模式 4 流模式。数据实质上在没有任何握手协议的情况下被传输到外围设备。
次设备号格式： **0x***II0***004**.

模式 5 同时使用 ACK 和 BUSY 的自动握手的脉冲模式。与模式 1 相似，只是数据选通线 **nSTROBE** 被发送方在一定时间内脉冲后才释放。
次设备号格式： **0x***II0***005**.

模式 6 仅使用 BUSY 的自动握手的脉冲模式。与模式 1 相似，只是数据选通线 **nSTROBE** 被发送方在一定时间内脉冲后才释放。
次设备号格式： **0x***II0***006**.

模式 1 和 2 支持大多数 HP *Jet 系列打印机（LaserJet、DeskJet 和 QuietJet 等）。

作者

cent 由 HP 开发。

另请参阅

lp(1)、ioctl(2)、intro(7)、lp(7)。

名称

clone - 打开 STREAMS 驱动程序上的主次设备对

说明

clone 驱动程序是一种“通透式”(pass through) 机制的设备驱动程序，它允许其他驱动程序对每个 **open()** 选择唯一的次设备号。实际上，驱动程序将打开操作传递到其他驱动程序。该机制通过一个设备文件允许多个驱动程序实例，每个实例带有一个不同的次设备号。

当打开 **clone** 驱动程序时，操作系统会向其传递一个主次设备号。主设备号是 **clone** 驱动程序的主设备号 (72)，次设备号是用户希望克隆的驱动程序（这里称为目标驱动程序）的主设备号。 **clone** 驱动程序调用目标驱动程序（该程序具有指明克隆打开的 **CLONEOPEN** 标志）的打开例行程序。目标驱动程序的打开例行程序将分配一个未使用的次设备号。目标驱动程序必须使用 **makedev** 为新创建的设备创建新设备号，并且必须将 ***devp** 设置为 **makedev** 返回的新设备号。新设备号通过 ***devp** 返回到 **clone** 打开。然后 **clone** 打开返回给用户一个指向目标驱动程序的新实例的文件描述符。

echo 驱动程序是可克隆驱动程序的示例。

注释

使用 **clone** 驱动程序，不可能打开多个具有相同主次设备号的设备。这是因为对 **clone** 驱动程序仅提供了要克隆的驱动程序的主设备号，然后该驱动程序将选择一个尚未打开的次设备号。

当使用对应于可克隆驱动程序的路径名调用时，如果对从同一个可克隆驱动程序路径名的 **open()** 返回的文件描述符调用 **stat()** ，则它将返回不同于 **fstat()** 的结果。

返回值

如果为 **clone** 驱动程序提供一个无效次设备号，或如果指定的驱动程序不可克隆，则 **open()** 将会失败并且 **errno** 将被设置为 [ENXIO]。

另请参阅

open(2)、 fstat(2)。

名称

console、 systty、 syscon - 系统控制台接口

说明

/dev/console 为配置为系统控制台的设备提供一个 **termio** 接口。 *init*(1M) 联机帮助页论述了 **/dev/systty** 和 **/dev/syscon** 的用法。

通常由 **/dev/console** 发送到控制台或由内核 **printf()** 生成的输出数据，可以通过 **TIOCCONS ioctl()** 重定向到另一个终端或伪终端设备。有关详细信息，请参阅 *termio*(7) 。

文件

/dev/console
/dev/systty
/dev/syscon

另请参阅

init(1M)、 termio(7)。

符合的标准

console： SVID2、 SVID3 和 XPG2

名称

ddfa - 数据通信和终端控制器 (DTC) 设备文件访问 (DDFA) 软件

说明

数据通信和终端控制器 (DTC) 设备文件访问 (DDFA) 软件允许使用标准的 HP-UX 结构从系统实用程序和用户应用程序访问终端服务器。 DDFA 提供远程 LAN 连接终端服务器端口的接口，此接口与本地直连端口的接口相似。

基本原理是，根据配置文件（专用端口文件）中的信息，为每个配置的终端服务器端口创建一个守护程序。衍生了守护程序后，将通过池生成一个 **pty** ，并创建一个具有与从属 **pty** 相同的主次设备号的设备文件。此设备文件称为“假名”，实用程序和应用程序可使用假名，通过执行标准的 HP-UX 系统函数（ open() 、 **close()** 、 **read()**、 **write()** 和 **ioctl()** ）来访问终端服务器端口。应用程序对假名执行 **open()** 之前，守护程序将一直监听 **pty**。然后，在应用程序对假名执行 **close()** 之前，此守护程序将设置和管理与终端服务器端口的连接。最终结果是通过设备文件编址了终端服务器端口，但是实现此结果的机制对用户是透明的。第二个配置文件（端口配置文件）包含用于配置处理终端服务器端口的信息。

DDFA 包括以下项目：

dp 专用端口文件。此文本文件包含 DDFA 在设置和管理假名与某个终端服务器端口之间的连接时所需的信息。

dp 文件由专用端口分析器 (**dpp**) 分析，此分析器可为文件中指定的每个传出连接衍生传出连接守护程序 (**ocd**) 。 HP-UX Telnet 守护程序 (**telnetd**) 还使用 **dp** 文件标识来自 DTC 的传入连接，并将这些连接映射到假名（Telnet 端口标识功能）。

pcf 端口配置文件。 DDFA 使用此文本文件来配置处理终端服务器端口。此模板文件的一般名称是 **pcf** 。端口配置文件由专用端口文件 (**dp**) 中的一个条目引用。

dpp 专用端口分析器。此命令可分析专用端口文件 (**dp**) ，并为 **dp** 文件中的每个有效条目衍生一个传出连接守护程序 (**ocd**) 。可以从 shell 执行此命令，也可以将它加入系统初始化脚本，以便每次引导系统时，可以自动运行 DDFA 软件。

ocd 传出连接守护程序。此守护程序管理与远程终端服务器端口的连接和数据传输。通常它是专用端口分析器 (**dpp**) 衍生的，但是也可以从 shell 直接运行。

此守护程序启动时，将针对连接创建其假名。正常终止时，它将会删除此假名。如果在守护程序运行期间删除了假名，那么 **ocd** 将会终止，同时返回错误。

ocdebug 传出连接守护程序调试模式。这是包含调试代码的 **ocd** 的特殊版本。必须从 shell 运行此守护程序。

配置

可通过两个基本步骤配置 DDFA 软件：

- 在 **dp** 文件中输入信息。

- 在端口配置文件中输入信息。

配置 **dp** 文件

dp 文件包含要建立的每个传出连接使用的一个行，以及每个传入连接请求使用的一个行。应将缺省文件 **/usr/examples/ddfa/dp** 复制到新文件，还可以根据需要复制编辑过的文件。建议创建一个目录，用于保存 **dp** 及端口配置文件。

dp 文件的每一行必须包含终端服务器端口的位置及假名的位置。此外，对于传出连接，必须指定端口配置文件，同时还可以指定日志记录级别。

配置端口配置文件

可使用端口配置文件来配置各个终端服务器端口。主端口配置文件为 **/usr/examples/ddfa/pcf** 。实际上，根据需要不同配置值的每个端口，会重命名此文件，并且值会针对附加到端口的设备而相应地改变。建议创建一个目录，用于保存端口配置文件及 **dp** 文件。

端口配置文件的每一行必须包括一个变量名称及此变量的值。变量-值对包含有关如何打开与终端服务器端口的连接、如何关闭与终端服务器端口的连接、如何管理与终端服务器端口进行数据传输的信息。

配置系统初始化脚本

要在每次引导时运行 DDFA，可以在系统初始化脚本中加入对 **dpp** 的引用。建议在此环境中运行 **dpp** 时，使用 **-k** 选项。

终止守护程序

请注意，应使用 **kill -15** 终止 **ocd** 。不要使用 **kill -9** 完成此操作，原因是它不能删除设备文件。 **ocd** 可以在尝试使用现有假名之前验证其有效性。 **dpp** 和 **ocd** 在替换某个进程之前，可使用 **/var/adm/utmp.dfa** 文件中存储的数据来验证此进程是否还拥有假名。如果 **ocd** 找到无主假名，则使用此假名。

错误处理

如果 **ocd** 收到一个严重错误，例如 LAN 故障，则它会通过关闭 **pty** 将此错误传输给应用程序。对假名执行 **open()** 、 **close()** 、 **read()** 和 **write()** 中的任一调用，都会返回错误 **0 bytes read** 。如果此假名为应用程序所属组的控制终端，则将 **SIGHUP** 发送到此组中的所有进程，包括此应用程序。

ioctl() 限制

由于缺少通过 LAN 将这些命令传输至远程端口的协议，因此并不是所有的 **ioctl()** 功能都是可用的。

termio 属性限制

termio 属性（请参阅 *termio*(7) ）的主要限制包括调制解调器信号控制和奇偶校验检查。下列项是不可用的：

CBAUD IGNPAR INPCK IXANY IXOFF PARMRK

ioctl() 请求限制

可应用以下 **ioctl()** 请求限制：

CSTOPB 标志	DTC 只支持一个停止位。
CSIZE	DTC 只支持 8 位字符。值不可以修改。

PARODD 标志	DTC 可提供静态配置来处理偶数奇偶校验或奇数奇偶校验。同时还可以针对偶数奇偶校验或奇数奇偶校验处理自动奇偶校验检测。
PARENB 标志	通过静态配置完成启用/禁用。不提供编程接口。
INPCK 标志	无法将输入从输出奇偶校验功能分离出来。
IGNPAR 标志	无法在 DTC 上配置。
PARMRK	将错误的字符转发到了系统，并且没有对这些字符标记 OFFH 或 OH。
CBAUD	速度是静态配置的一部分。
IXOFF 标志	如果 DTC 静态配置指定了 ASCII 访问模式，将启用流量控制。如果选择了二进制，则不提供流量控制。
IXON 标志	如果在静态端口配置中选择了 ASCII 模式，则启用通过编程接口调节输出到终端的速度，选择二进制模式后则禁用。
IXANY 标志	如果以前收到了 XOFF，则 DTC 不提供对已收到的任何字符重新启动输出的功能。
HUPCL 标志	DDFA 不支持在最后一次关闭设备文件后挂断调制解调器信号。如果 DTC 占用的调制解调器信号断开，则关闭连接。
CLOCAL 标志	不支持。
c_flags	不支持 **IENQACK** 。 Telnet 端口标识软件不支持 **OFILL** 、 **OFDEL** 、 **NLDLY** 、 **CRDLY** 、 **TABDLY**、 **BSDLY** 和 **FFDLY** 。
BINARY 模式标志	将在 DTC 管理器中通过选择二进制模式完成静态配置的一部分操作。如果启用了切换，则可以在用户接口级别选择二进制。使用 **telnetd** 时，如果重置了正确的 termio 标志，则无法自动协商二进制模式。可以针对 DDFA 切换二进制/ASCII 模式。 DTC 不支持在纯二进制模式下进行大量读取，因此传输的数据块不应超过 256 个字节。如果已实现带远程确认的半双工，则可支持二进制应用程序。

ioctl() 系统调用请求

可应用以下 **ioctl()** 系统调用限制：

TCSBRK	**telnetd** 或 DDFA 中的系统级别未提供如下功能：即发送一个中断，而不等待发送以前的数据。如果 DTC 收到一个 Telnet 中断命令，则它可以在异步端口上生成一个中断。
TCFLSH	无法刷新 DTC 输出队列。
硬件握手请求	在 DTC 上不受支持。

TCXONC 无法在 DTC 上禁用本地握手。

MCGETA 不支持。

MCSETA、 **MCSETAF**、 **MCSETAW**
无法单独设置 DTC 端口的调制解调器线路。

MCGETT 无法配置调制解调器计时器、CD 计时器、连接计时器和断开连接。

CCITT 简单、直接的呼入或呼出模式
由于调制解调器信号存在编程接口，因此 DTC 无法处理简单模式。如果端口已打开，则不能模拟呼入模式，原因是必须在 2 分钟之内提供调制解调器信号（或呼叫），否则将清除连接。

DACIDY 获取设备适配器信息
无法获取设备适配器信息。

下载 ioctl()**DACRADDR**、 **DACDLADDR**、 **DACDLGO** 、 **DACDLVER** 未提供可下载 DTC 的编程调用。

DACHWSTATUS、 **DACSELFTEST**、 **DACLOADED** 、 **DACISBROKE**
状态未提供可以获取此类信息的编程接口。

DACLOOPBACK DACSUBTEST 端口测试

警告

为了确保命令（比如 *ps* ）显示正确的设备文件名（也就是 *pseudonym* ），所有假名应该放到目录 **/dev/telnet** 中。如果没有在此目录中指定要替换的假名，则不能保证有许多命令的设备文件名正确显示。

此外，为了确保命令（比如 **w** 、 **passwd** 、 **finger** 和 **wall** ）正常工作，每个假名前 17 个字符必须唯一（包括目录前缀 **/dev/telnet/** ）。如果假名前 17 个字符不唯一，则不能保证许多命令正确运行。

同时，为确保可靠处理计时标记协商（并确保在附加到终端服务器的打印机上打印的文件已完全刷新到此打印机），必须将以下行添加到终端服务器所附加的打印机的每个打印机接口脚本中接近末尾的位置：

stty exta <&1 2>/dev/null

打印机接口脚本驻留在目录 **/etc/lp/interface** 中。必须紧邻每个打印机接口脚本中的最终 **exit** 命令之前添加此行。

如果未按照指定要求添加此行，则无法保证终端服务器所附加的打印机的打印可靠性。

文件

/usr/examples/ddfa/dp
/usr/examples/ddfa/pcf
/usr/sbin/dpp
/usr/sbin/ocd
/usr/sbin/ocdebug
/var/adm/dpp_login.bin

/var/adm/utmp.dfa

另请参阅

dpp(1M)、 ocd(1M)、 ocdebug(1M)、 ioctl(2)、 dp(4)、 pcf(4)、 ioctl(5)、 termio(7)。

名称

diag0 - HP-PB I/O 子系统的诊断接口

说明

diag0 是一个诊断伪驱动程序，它为 HP 支持工具提供访问 HP-PB I/O 子系统的权限。硬件监视器和支持工具管理器 (STM) 内的工具使用该驱动程序与通过 HP-PB 连接到系统的外围设备进行交互。I/O 驱动程序还将诊断事件发送到 **diag0** 以便支持工具管理器进行诊断日志记录。

如果没有 **diag0** ，则那些能帮助防止外围设备出现故障的信息将会丢失。此外，如果发生故障，HP 将不具有及时诊断问题原因的工具或数据。这可能增加停机时间并导致将来发生故障。

作者

diag0 由 HP 开发。

文件

/stand/vmunix
/dev/diag/diag0
/dev/diag 包含诊断设备文件的目录

另请参阅

支持工具管理器中的 stm(1M)。

名称

diag1 - PCI I/O 子系统的诊断接口

说明

diag1 是一个诊断伪驱动程序，它为支持工具提供访问 PCI I/O 子系统的权限。支持工具管理器 (STM) 内的工具使用该驱动程序与连接到系统上的 PCI 卡进行交互。如果没有 **diag1** ，则 PCI 卡支持工具将无法操作。

警告

HP-UX 11i v1.5 不支持 **diag1** 。

作者

diag1 由 HP 开发。

文件

/stand/vmunix
/dev/diag/diag1
/dev/diag 包含诊断设备文件的目录

另请参阅

支持工具管理器中的 stm(1M)。

名称

diag2 - 诊断日志记录接口和处理器接口

说明

硬件监视器和支持工具管理器 (STM) 内的工具使用 **diag2** ，通过处理器相关代码 (PDC) 与处理器硬件进行交互。如果没有 **diag2** ，则处理器的支持工具将无法操作。

diag2 也是支持以下功能的重要组件:

I/O 错误日志记录
低优先级计算机检查 (LPMC) 日志记录
内存错误日志记录
主动内存页取消分配。

如果没有上述功能，则那些能帮助防止系统或外围设备出现故障的信息将会丢失。此外，如果发生故障，HP 将不具有及时诊断问题原因的工具或数据。这可能增加停机时间并导致将来发生故障。

作者

diag2 由 HP 开发。

文件

/stand/vmunix
/dev/diag/diag2
/dev/diag2
/dev/diag 包含诊断设备文件的目录

另请参阅

支持工具管理器中的 stm(1M)

名称

disk - 直接磁盘访问

说明

本条目介绍将磁盘当作块专用设备或字符专用（原始）设备时， HP-UX 磁盘驱动程序的操作。

设备文件命名约定

根据以下约定命名标准的磁盘设备文件（请参阅 *intro*(7) ）：

Block-mode Devices	**/dev/disk/disk***N*[**_p***X*]
Character-mode Devices	**/dev/disk/disk***N*[**_p***X*]
Legacy block-mode Devices	**/dev/dsk/c***x***t***y***d***n*[**s***m*]
Legacy character-mode Devices	**/dev/rdsk/c***x***t***y***d***n*[**s***m*]

Legacy 设备专用文件名是指应用于 HP-UX 11i v2 和更早版本上的文件名。这些文件名仍然可以使用，以便向后兼容，但是仅限用于 HP-UX 11i v2 中的部分配置。

设备文件名组成部分可按如下方式构建：

N 必需。它是与操作系统为直接访问设备分配的实例编号相对应的十进制整数。

X 如果指定了 **_p** ，则为必需的。它是与分区编号相对应的十进制整数。

c 必需。将以下十六进制数字标识为接口卡的“实例”。

x 标识控制总线接口的十六进制数字，也称为此接口卡的“实例”。在 *ioscan*(1M) 输出中 H/W 类型所对应的“I”列“INTERFACE”中显示了此实例值。
必需。

t 将以下十六进制数字标识为“驱动器号”或“目标”。
必需。

y 标识驱动器或目标编号（总线地址）的十六进制数字。
必需。

d 将以下十六进制数字标识为“单元号”。
必需。

n 设备内部的十六进制单元号。
必需。

s 可选。缺省为与整个磁盘对应的值。将以下值标识为“扇区号”。

m 如果指定了 **s** ，则是必需的。缺省为扇区 0（零），即整个磁盘。驱动器扇区号。

系统的系统管理员手册中介绍了控制器、驱动器、逻辑单元和扇区号的分配。

块专用访问

块专用设备文件可通过系统的块缓冲区缓存机制访问磁盘。完成缓冲的工作方式是：正确处理通过多个 open 调用进行的并行访问，及安装相同的物理设备，以避免出现操作顺序的错误。块缓冲区缓存允许系统在方便时执行物理 I/O 操作。这意味着物理写入操作在时间上可能比其对应的逻辑写入请求有相当程度的滞后。同时，这也意味着物理读取操作在时间上可能比其对应的逻辑读取请求有相当程度的提前。

可以在忽略物理磁盘记录的前提下读取和写入块设备专用文件。块设备专用文件的 **read()** 和 **write()** 调用需要进行磁盘访问，因此将在磁盘和块缓冲区缓存之间产生一个或多个 **BLKDEV_IOSIZE** 字节（通常为 2048 字节）的传输。使用块专用设备的应用程序应确保其读取或写入操作不超过设备文件中最后一个大小为 **BLKDEV_IOSIZE** 的块的末尾。这是因为接口已被缓冲，超过此点的访问将产生不可预测的行为。

字符专用访问

字符专用设备文件可以在不缓冲的情况下访问磁盘，并支持在磁盘和用户的读取/写入缓冲区之间进行直接的数据传输。通过字符专用文件接口进行磁盘访问，可以在控制从调用返回之前，完成所有物理 I/O 操作。对最多 **MAXPHYS** 字节（通常为 64 KB 或 256 KB）执行单次读取或写入操作将正好导致一个磁盘操作。如果请求大于此数，操作系统将自动分解这些请求。因为通过字符专用文件执行大的 I/O 操作可避免块缓冲区缓存处理过程，并可减少磁盘操作，所以这些操作比类似的块设备专用文件操作更加有效。

对于字符专用文件的 **read()** 和 **write()** 调用，内存中的用户缓冲区对齐问题可能具有与实现相关的限制。此外，每次读取和写入操作都必须在逻辑块边界上开始和结束，同时其大小还必须为整数个逻辑块。逻辑块的大小是一个与硬件相关的值，可通过下文介绍的 **DIOC_DESCRIBE_EXT** 和 **DIOC_DESCRIBE** 读写控制调用进行查询。

字符专用文件接口除了用于读取和写入数据外，还可获取特定设备的信息，及执行特殊的操作。可以使用读写控制调用来控制这些操作。 **<sys/diskio.h>** 中包含了与这些读写控制有关的详细信息。

可使用 **DIOC_DESCRIBE_EXT** 和 **DIOC_DESCRIBE** 读写控制获取特定设备的标识信息。返回的信息包括磁盘的型号标识、磁盘接口类型、最大偏移地址、设备类型和磁盘的逻辑块大小。

可使用 **DIOC_CAPACITY** 读写控制获取磁盘设备的容量，以 **DEV_BSIZE** 为单位（ DEV_BSIZE 在 **<sys/param.h>** 中定义）。

可使用 **DIOC_EXCLUSIVE** 读写控制获取和释放对磁盘设备的独占访问。某些特殊操作（例如，介质重新格式化）必须采用独占访问，在其他某些环境中也可能需要采用此类访问。值 1 指定要求独占访问。值 0 指定应释放独占访问。独占访问导致其他打开请求失败。仅当设备当前未在块模式下打开，并且只打开了此磁盘设备的一个文件表条目（独占访问请求者可以访问的那一个条目）时，才授予独占访问权限。

错误

磁盘设备驱动程序调用可能会返回以下错误：

[EACCES] 设备或操作所需的权限被拒绝。

[EIO] I/O 错误（例如，介质缺陷或设备通信问题）。

[EINVAL] 如果是从 **open()** 调用返回的：则表示此设备不是磁盘设备。对于其他调用：则表示无效的请求或参数。请注意，对于传统的 32 位访问，如果设备的大小超出 **DIOC_DESCRIBE** 或 **DIOC_CAPACITY** 读写控制的参数，则可能导致此错误。

[ENXIO] 如果是从 **open()** 调用返回的结果，这表示指定的地址上没有设备。对于其他调用，这表示指定的地址超出范围，或者不能再访问此设备。

警告

未指定对大小为 **BLKDEV_IOSIZE** 的相同块进行块设备专用文件访问和字符专用文件访问的交互，通常此交互是不可预测的。

对于某些系统，如果在同一设备上同时打开挂接的文件系统和块设备专用文件，可能会导致不可预测的结果；如果可能，则应避免此问题。这是因为，某些文件可能拥有某些系统中的专用缓冲区。

虽然从历史角度来说，磁盘设备的块大小都比较小（通常为 512 字节），但某些磁盘设备（例如，光盘和磁盘阵列）具有相对较大的块大小。使用直接的裸磁盘访问的应用程序应使用 **ioctl()** 调用来确定相应的 I/O 操作大小和对齐。

对于带有可移动介质（例如，软盘或 CD-ROM），并且此介质包含挂接文件系统的任何磁盘，不应在卸除之前将其移除。移除包含了挂接文件系统的磁盘介质很可能导致文件系统错误和系统异常。

作者

disk 由 HP 和 AT&T 联合开发。

另请参阅

ioscan(1M)、 mknod(1M)、 intro(7)。

系统随附的系统管理员手册。

名称

dlpi - 数据链路提供程序接口

说明

该联机帮助页对 DLPI （数据链路提供程序接口）以及如何与 DLPI 提供的 API 集连接作了简短说明。

HP-UX DLPI 用作 OSI 体系结构的第 2 层（数据链路层）。DLPI 用作局域网设备驱动程序和 DLPI 用户之间的接口。DLPI 仅供具有经验的网络用户使用。

HP-UX DLPI 具有两个更大的接口集。第一个接口集按照 DLPI 2.0 标准提供，第二个接口集是该标准的 HP 扩展。

HP-UX DLPI 还为设备驱动程序提供接口，以与 STREAMS 模块和 DLPI 应用程序连接。

STREAMS 模块和 DLPI 应用程序

DLPI 的 HP 实现是样式 2 服务提供程序。样式 2 提供程序需要 DLS 用户使用原始专用附加服务明确标识 PPA。有关 PPA 的详细信息，请参考 *lan*(7) 联机帮助页。

HP DLPI 为 STREAMS 模块和 DLPI 应用程序提供下列服务：

- 克隆（最大值 3992）和非克隆（最大值 100）访问。
- 支持以太网/IEEE802.3、FDDI 和令牌环接口。
- 支持无连接以及连接模式服务（仅在 IEEE802.3 和令牌环上支持连接模式服务）。
- 支持原始模式服务。
- 支持 **I_STR** ioctl 进行特定设备控制和诊断请求。
- 支持第三方设备驱动程序。
- 支持所有级别的混合模式。

HP DLPI 不为 STREAMS 模块和 DLPI 应用程序提供下列服务：

- 服务质量 (QOS) 管理。
- 连接管理 STREAMS：面向连接的 STREAMS 上的 **DL_SUBS_BIND_REQ** 和 **DL_SUBS_UNBIND_REQ** 。
- 已确认的无连接模式服务。

基于 DLPI 2.0 标准的 DLPI 请求在 **<dlpi.h>** 中定义；请参阅 *dlpi*（4）。 DLPI 的 HP 扩展在 **<dlpi_ext.h>** 中定义；请参阅 *dlpi_ext*（4）。

设备文件格式

要通过 DLPI 接口访问局域网驱动程序，DLS 用户必须使用以下设备文件：

名称	类型	主要 #	次要 #	访问类型
/dev/dlpi	c	72	0x77	Clone access
/dev/dlpiX	c	119	0xX	Non-Clone access

设备驱动程序

HP-UX DLPI 属于非本地设计。驱动程序和 DLPI 不是耦合在一起的，并且在系统上作为单个组件存在。非本地 DLPI 支持两种驱动程序。紧密耦合和松散耦合的驱动程序。

DLPI 为紧密耦合和松散耦合的驱动程序提供接口。DLPI 用作 DLS 用户紧密耦合驱动程序的唯一接口。但是，松散耦合的驱动程序仅仅依赖于 DLPI 为用户空间命令 *lanscan*(1M) 和 *nwmgr*(1M) 提供信息以进行显示。

设备驱动程序接口在 **<dlpi_drv.h>** 中定义，请参阅 *dlpi_drv*（4）。

DLPI 为紧密耦合的驱动程序提供下列功能:

- 允许驱动程序与上层 STREAMS 模块或应用程序通信的基础结构。
- 协议、组播及混合处理基础结构。
- 异步控制处理基础结构。
- 入站帧处理。
- 处理启动链路和关闭链路事件。
- 所有注册接口储备库和相关联的信息。
- 传递到物理驱动程序前的出站处理。

DLPI 通过三个导出头文件提供服务。头文件 **<dlpi.h>** 和 **<dlpi_ext.h>** 用于用户空间应用程序和内核级别 STREAMS 模块。头文件 **<dlpi_drv.h>** 用于物理和逻辑驱动程序。

警告

DLPI 的各种实现存在于 HP-UX 专用技术（如 ATM 和 Hyper Fabric 等）中；但是该联机帮助页介绍的是支持局域网类驱动程序（紧密耦合）的 DLPI。

不推荐使用 **lanadmin** 、 **lanscan** 和 **linkloop** 命令。在将来的 HP-UX 版本中将删除这些命令。 HP 建议使用替换命令 *nwmgr*(1M) 来执行所有与网络接口相关的任务。

作者

dlpi 由 HP 根据 DLPI 2.0 标准开发。

另请参阅

lanscan(1M)、 nwmgr(1M)、 dlpi(4)、 dlpi_drv(4)、 dlpi_ext(4)、 lan(7)。

《DLPI Programmer's Guide》，2003 年，HP
《Driver Development Guide》，HP

《Device Driver Reference》, HP

名称

framebuf - 光栅帧缓冲设备信息

概要

#include <sys/framebuf.h>

说明

帧缓冲设备是基于光栅的显示器。这些设备使用内存映射 I/O 来获取性能，此性能要比通过基于 tty 的图形终端所能获取的性能高出许多。虽然建议通过图形库访问帧缓冲设备，但是也可以使用此接口直接访问帧缓冲设备。直接访问帧缓冲设备，需要确切了解所使用的帧缓冲体系结构。由于帧缓冲设备不是串行设备，因此不可以将输入传送或重定向到这些设备。

每个帧缓冲设备都与一个字符专用文件关联。帧缓冲设备的主次设备号与实现有关。这些设备的次设备号表示不同的帧缓冲区。相应的系统管理员手册中介绍了实现特定的详细信息。

与帧缓冲设备的通信由 **open()** 系统调用开始。多个进程可以同时打开一个帧缓冲设备。

close() 可以使与帧缓冲设备关联的文件描述符失效。完成 **close()** 系统调用后，对帧缓冲设备地址范围执行任何访问可能导致内存故障，并向进程发送信号 SIGSEGV （请参阅 *signal*(2) ）。关闭帧缓冲专用文件后，进程无法从其地址空间取消映射帧缓冲区。要取消帧缓冲区，请使用 **GCUNMAP ioctl()** 调用（请参阅下文）。

一旦进程获取了帧缓冲设备的锁，此进程必须在调用 **close()** 之前显式解锁此帧缓冲设备；请参阅下面的 **GCUNLOCK** 。

read() 和 **write()** 系统调用是未定义的，它们始终返回错误。在这种情况下， **errno** 将设置为 [ENODEV] 。

可使用 **ioctl()** 系统调用控制帧缓冲设备。可使用 **select()** 系统调用测试帧缓冲设备，以检查异常情况。图形硬件的中断被视为异常情况。当打开帧缓冲设备的任何进程得到由 **select()** 调用检测到的异常的通知后，此异常情况将自动清除。调用 **select()** ，以便对与帧缓冲设备关联的文件描述符进行读取或写入操作，将在读取或写入位掩码中返回 False（请参阅 *select*(2) ）。

多个进程可同时访问一个帧缓冲设备。但是，除非使用了此处所述的其中一个锁定机制，或者使用了其他某种同步机制，否则每个进程都将覆盖其他进程的输出。此处所述的锁定机制只适用于协作进程。

对于所有帧缓冲区，数据字节将以从左到右、从上到下的顺序扫描。像素是屏幕上可见的点，与帧缓冲区中的某个位置关联。每个设备都将内存中的一个或多个位映射到屏幕上的像素，尽管帧缓冲区中的位不一定是连续的。在 **crt_frame_buffer_t** 数据结构中可以找到描述帧缓冲区结构和属性的信息。 **crt_frame_buffer_t** 数据结构包括下列字段：

```
int crt_id;                     /*display identifier*/
unsigned int crt_attributes;    /*flags denoting attributes*/
char *crt_frame_base;           /*first byte in frame-buffer memory*/
char *crt_control_base;         /*first byte of the control registers*/
char *crt_region [ CRT_MAX_REGIONS ];
                                /*other regions associated with the frame-buffer device*/
```

以下为有效的 **ioctl()** 请求：

GCDESCRIBE 描述帧缓冲区的大小、特性和映射的区域。此信息将返回到 **crt_frame_buffer_t** 数据结构中的调用进程，参数定义为 **crt_frame_buffer_t *arg;** 。虽然某些结构字段包含一个或多个帧缓冲设备区域的地址，但始终不会定义这些字段的值。仅当成功发出了 **GCMAP** 命令（请参阅下文）后，才返回正确的地址，以便用户能够使用返回的地址直接访问帧缓冲区域。

GCID 提供设备标识号。参数定义为 **int *arg;** 。使用此命令后返回的信息是 **GCDESCRIBE** 提供的信息的子集，此处提供此信息的目的只是为了实现向后兼容。

GCON, **GCOFF** 打开或关闭图形。这些操作对于在 **GCDESCRIBE** 命令返回的 **crt_frame_buffer_t** 数据结构的 **crt_attributes** 字段中设置了 CRT_GRAPHICS_ON_OFF 位的设备有效。对于其他情况，这些命令不起作用。

GCAON, **GCAOFF** 打开或关闭 alpha。这些操作对于在 **GCDESCRIBE** 命令返回的 **crt_frame_buffer_t** 数据结构的 **crt_attributes** 字段中设置了 CRT_ALPHA_ON_OFF 位的设备有效。对于其他情况，这些命令不起作用。

GCMAP 使帧缓冲内存、图形控制和其他设备区域可由执行调用的用户进程访问。只有请求此操作的进程才可直接访问帧缓冲内存和控制寄存器。成功完成 **GCMAP** 调用后，**crt_frame_buffer_t** 数据结构（由后续 **GCDESCRIBE ioctl()** 调用返回）中的 **crt_frame_base** 和 **crt_control_base** 字段将保留帧缓冲区的这两个区域的有效地址。对于特定设备，如果要将两个以上的区域映射到用户的地址空间，那么最多会将 CRT_MAX_REGIONS 个额外设备区域的基址依次置于 **crt_region** 数组中。只映射与特定帧缓冲区有关的区域。 **crt_frame_buffer_t** 数据结构中的不相关的区域字段将设置为 **0** 。*arg* 参数的用途与实现相关（请参阅下面的“相关内容”）。帧缓冲区域的基址始终是页对齐的。

GCUNMAP 导致从请求进程中删除对帧缓冲内存、图形控制及其他可能的设备区域的访问权限。将忽略参数 *arg* ，应将其设置为 **0** 。成功完成 **GCUNMAP** 调用后，尝试访问这些内存区域都将导致内存故障，并将信号 SIGSEGV 发送到进程。

GCLOCK 为协作进程提供对帧缓冲设备的独占使用权。调用进程将锁定设备并继续，或者被阻塞。在这种情况下，阻塞意味着：仅当帧缓冲区可用或者信号中断了调用时才返回调用。如果调用中断，它将返回一个错误，并将 **errno** 设置为 [EINTR] 。如果另一个进程事先使用 **GCLOCK** 命令锁定了此帧缓冲区，并且尚未执行 **GCUNLOCK** 命令，则会进行等待。**GCLOCK** 命令不能防止其他非协作进程写入帧缓冲区；因此， **GCLOCK** 只是一个监视锁。将忽略参数 *arg* ，应将其设置为 **0** 。

此调用可防止内置仿真终端 (ITE) 破坏图形硬件的状态（请参阅 *termio*(7) ）。在某些系统上，只要使用 **GCLOCK** 命令锁定了帧缓冲区，ITE 就不会向此帧缓冲区输出文本（请参阅下面的“相关内容”）。同一进程尝试多次锁定设备均会失败，并使 **errno** 设置为 [EBUSY] 。

GCLOCK_NOWAIT 为协作进程提供对帧缓冲设备的独占使用权。此请求与 **GCLOCK** 请求对帧缓冲设备产生的影响相同。但是，此调用不会等待其他进程释放帧缓冲区。如果锁定了帧缓冲设备，进程将不会被阻塞；系统调用将返回一个错误，并使 **errno** 设置为 [EAGAIN] 。将忽略参数 *arg* ，应将其设置为 **0** 。

GCLOCK_BLOCKSIG

为协作进程提供对帧缓冲设备的独占使用权，同时阻塞调用进程的所有传入信号，否则这些信号将被捕获。此调用为 **GCLOCK** 调用的超集。将忽略参数 *arg* ，应将其设置为 **0** 。在获取了显示内容以供独占使用（即，被锁定），发送到该进程的所有信号都会被阻止，直到请求了 **GCUNLOCK** 为止，这些信号本来会在调用 **GCLOCK** 时被该进程捕获。发出 **GCUNLOCK** 请求之前，尝试修改进程的信号掩码（请参阅 *sigsetmask*(2) ）都不会对这些阻塞的信号产生任何影响。在实际获取锁之前，这些信号都不会被阻塞，可以在等待锁的同时接收这些信号。

信号 SIGTSTP 同样会被阻塞，而不论它是否已被捕获。信号 SIGTTIN 和 SIGTTOU 也会在帧缓冲设备上被阻塞，当 ITE 被锁定时，它不会输出到此设备。请参阅下面的“相关内容”。

除了上述三个信号外，此调用既不阻塞进程不需要捕获的信号，也不阻塞无法捕获或无法忽略的信号。此命令不能防止其他非协作进程写入帧缓冲区。

GCLOCK_BLOCKSIG_NOWAIT

为协作进程提供对帧缓冲设备的独占使用权，同时阻塞调用进程的所有传入信号，否则这些信号将被捕获。此请求与 **GCLOCK_BLOCKSIG** 请求对帧缓冲设备产生的影响相同。但是，此调用不会等待其他进程释放帧缓冲区。如果锁定了帧缓冲设备，进程将不会被阻塞，但是，系统调用将返回一个错误，并将 **errno** 设置为 [EAGAIN] 。将忽略参数 *arg* ，应将其设置为 **0** 。

GCUNLOCK 放弃对帧缓冲设备的独占使用权。如果使用 **GCLOCK_BLOCKSIG** 或 **GCLOCK_BLOCKSIG_NOWAIT ioctl()** 请求锁定了设备，则调用进程的信号掩码将恢复到锁定请求之前的状态。

GCRESET 将与帧缓冲设备关联的图形硬件重置到定义的初始状态。通过此调用，帧缓冲设备可以响应此处定义的 **ioctl()** 请求。

GCDMA_OUTPUT 将 DMA 输出发送到帧缓冲设备。可使用此系统调用将数据从用户的数组传输到图形帧缓冲区的矩形区，或者传输到设备的图形控制空间。

DMA 的参数将传入包括下列字段的 **crt_dma_ctrl_t** 数据结构：

```
char *mem_addr;     /* Starting address of data being transferred */
char *fb_addr;      /* Address of framebuffer destination */
int length;         /* Number of bytes to transfer, including those "skipped" */
int linelength;     /* Number of bytes written on each framebuffer row */
```

```
int skipcount;      /* Number of source bytes to ignore after each "linelength" */
unsigned int flags; /* Specified options to the driver */
```

要写入图形帧缓冲区，请将 **fb_addr** 设置为要绘制的矩形的左上角的地址。DMA 将在每个帧缓冲行上写入 **linelength** 个字节，同时忽略内存数据的后面 **skipcount** 个字节，然后在每个后续的帧缓冲行的相同起始位置恢复写入。此操作将一直继续，直到写入或忽略了 **length** 个字节。

要写入图形控制空间，请将 **fb_addr** 设置为要写入的第一个图形控制寄存器的地址。在这种情况下，将忽略 **linelength** 和 **skipcount** 。

flags 参数指定 DMA 的选项。当前，没有支持的标记，因此此参数应设置为零；否则系统调用将失败，并将 **errno** 设置为 [EINVAL] 。

DMA 对帧缓冲设备产生的影响与使用存储指令写入数据产生的影响相同。因此，不同的图形控制寄存器都可能影响 DMA 的结果。用户程序负责执行帧缓冲设备的必要设置，以便 DMA 得到所需的结果。

用户可通过 **skipcount** 参数刷新用户存储在内存中的窗口映像的一部分，在这种情况下，只需要刷新一部分映像。然后，此窗口映像将成为要更新的矩形的超集，因此可能具有不同的尺寸。 **skipcount** 指定较大的窗口映像中，矩形不包括的行的那一部分。因此， **linelength** 加 **skipcount** 就是较大的窗口映像数组的每一行中的字节数。

如果特定的帧缓冲设备支持此系统调用，则设置 **crt_frame_buffer_t** 结构的 **crt_attributes** 字段中的 CRT_DMA_OUTPUT 标记。某些支持 DMA 的帧缓冲设备将限制各个参数的对齐，下面的“相关内容”一节指定了这些帧缓冲设备。内核将确保遵循这些限制，如果无法遵循，则系统调用将失败，并将 **errno** 设置为 [EINVAL] 。

应用程序负责通过刷新处理器的数据缓存，保证系统的物理内存处于最新状态。用户在启动 DMA 传输之前，应使用 **GCDMA_DATAFLUSH** 读写控制确保数据的统一性。

GCDMA_DATAFLUSH

将指定的数据从处理器的数据缓存刷新到系统的主内存。此系统调用适合在 DMA 之前使用，以确保将最新版本的数据传输到帧缓冲区或控制空间。

刷新使用的参数将传入包括下列字段的 **crt_flush_t** 数据结构:

```
char *flush_addr;  /* Starting address of data
                      to be flushed */
int flush_len;     /* Number of bytes to flush */
```

内核将确保由 **flush_addr** 开始的 **flush_len** 个字节，相对于缓存来说，在主内存中是统一的。

GCSLOT

提供有关调用进程参与系统级图形锁定机制的相关信息（请参阅上文的 **GCLOCK** 小节下的介绍）。 **GCSLOT** 请求不执行任何实际锁定功能。锁定信息将返回到 **crt_gcslot_t** 数据

结构中的调用进程。此参数定义为 **crt_gcslot_t *arg;** 。文件 **<sys/framebuf.h>** 中定义了 **crt_gcslot_t** 数据结构。

GCSTATIC_MAP 防止内置仿真终端 (ITE) 修改设备的颜色映射。

GCVARIABLE_MAP 允许内置仿真终端 (ITE) 修改设备的颜色映射。

相关内容

请求 **GCMAP** 时，将忽略参数 *arg* ，应将此参数设置为 **0** 。

所有受支持的 ITE 都将忽略输出的帧缓冲区锁。

错误

[EAGAIN] 操作会导致挂起调用进程，但请求是 **GCLOCK_NOWAIT** 或 **GCLOCK_BLOCKSIG_NOWAIT**。

[EBUSY] 尝试锁定已经由相同进程锁定的设备。

[EINTR] 一个信号中断了 **ioctl()** 调用。

[EINVAL] 执行了无效的 **ioctl()** 命令。

[ENODEV] 在设备上尝试使用 **read()** 或 **write()** 系统调用。

[ENOMEM] 无法为映射分配足够的内存。

[ENOSPC] 无法分配映射所需资源。

[ENXIO] 设备文件上的次设备号引用了不存在的设备。

[EPERM] 请求了 **GCUNLOCK ioctl()** 命令，但是其他进程锁定了设备。

作者

framebuf 由 HP 开发。

另请参阅

mknod(1M)、 close(2)、 ioctl(2)、 lockf(2)、 open(2)、 select(2)、 signal(2)、 sigsetmask(2)、 termio(7)。

名称

gang_sched - 成组调度程序

说明

可通过成组调度程序，将一组 MPI（消息传递接口）进程，或单个进程中的多个线程当作一个组进行并行调度。

要启用和禁用成组调度，可将 **MP_GANG** 环境变量分别设置为 **ON** 和 **OFF** 。

成组调度功能可显著提升超额注册的负荷分时环境中的并行应用程序的性能。如果可运行并行线程、可运行 MPI 进程和其他可运行进程的总数超过了系统中的处理器数，就会发生超额注册。

成组调度还允许在共享内存并行应用程序中的线程间进行低滞后交互。

只能成组调度使用 HP-UX v11.0 MPI 或 pthread 库的应用程序。由于 HP 编译程序并行处理主要是在 pthread 库中构建的，因此使用 HP 编译程序编译的程序可受益于成组调度。

接口

可使用环境变量启用和禁用 HP-UX 成组调度程序。此变量定义为：

MP_GANG [ON | OFF]

将 **MP_GANG** 设置为 **ON** 可启用成组调度，设置为 **OFF** 可禁用成组调度。如果不设置 **MP_GANG** ，或者将其设置为不确定的值，则不执行任何操作。

成组调度是将要由 **fork** （请参阅 *fork*(2) ）所创建的子进程继承的进程属性。只有在调用 **exec** （请参阅 *exec*(2)）后，进程的成组调度的状态才会发生更改。

行为

将 **MP_GANG** 环境变量设置为 **ON** 后，执行和查找此变量的任何 MPI 或 pthread 应用程序都将启用此进程的成组调度。

只有 pthread 和 MPI 库才可查询 **MP_GANG** 变量，但操作系统不可以查询。

成组调度是一个继承的进程属性。如果某个启用了成组调度的进程创建了一个子进程，则会发生以下情况：

- 子进程将继承成组调度属性。
- 将为子进程生成一个新的组。子组不会成为其父组的一部分。

仅当某个组包括多个线程时，成组调度程序才起作用。对于 pthread 应用程序，此时需要创建第二个线程。对于 MPI 应用程序，此时需要添加第二个进程。

当进程创建线程时，如果为此进程启用了成组调度，则新的线程将添加到此进程的组。但是，一旦某个组的大小等于系统中的处理器数，就会发生以下情况：

- 不向组添加新的线程或进程。
- 组保留原样，并继续被成组调度。

- 使用常规的分时策略调度溢出的线程。
- 如果组中的线程退出（因而产生可用空间），则不会将溢出的线程添加到组。但是，当空间可用后，新建的线程将添加到组。

在执行的起始阶段静态分配 MPI 进程。如果 **MP_GANG** 设置为 **ON** ，那么 MPI 应用程序中的所有进程都将成为相同组的一部分。

将按照与分时策略相同的方式管理组的线程优先级和进程优先级。分时优先级调度程序确定何时调度一个组，同时它还遵循分时策略。

尽管在调度一个组时，可能会先占一个或多个高优先级的分时线程，但是从长时间的运行来看，成组调度程序策略一般是合理的。当调度一个组时，组中的所有线程都已成为最高优先级。因为组中的所有线程必须并行执行，所以有些已成为最高优先级的线程将不会执行（这些线程必须一直等待，直到选择了其他所有线程，以允许其他进程先运行）。

在一个时间片内调度组。系统中所有线程的时间片是相同的，而不论是否对此时间片进行成组调度。

如果单个组在系统上执行，则将此组的线程分配到系统中的处理器，而不是迁移到其他处理器。

在带有多个组的超额注册的系统中，将定期移动所有组，以便为每个不同的线程指定均衡的 CPU 时间百分比。每隔几秒就发生此平衡操作。

外部语言环境影响

环境变量

以下环境变量会影响进程的成组调度：

- **MP_GANG** 可启用（如果设置为 **ON** ）和禁用（如果设置为 **OFF** ）进程的成组调度。有关详细信息，请参阅本联机帮助页的“接口”一节。
- **MP_NUMBER_OF_THREADS** 指定处理器数，这些处理器可用于执行为并行执行而编译的程序。如果未设置此项，则缺省值为系统中的处理器数。

性能

成组调度可确保同时调度组中的所有可运行线程和进程。这可以改善并行应用程序中的同步滞后。例如，等待屏蔽的线程不必等待当前未调度的线程。

但是，对于带有冗长的并行区域并且不经常同步的应用程序，不对其进行成组调度时，可使其保持最佳性能。对于这些应用程序，即使没有同时调度所有线程，但也可能调度了某些线程。

成组调度的应用程序的性能受系统上的成组调度的应用程序数，以及每个应用程序中的线程数的影响。成组调度程序可使用一种“最适当的”算法将并行应用程序分配到 CPU，此算法可尝试将应用程序中的 CPU 重叠现象降到最低程度。

在带有复杂的工作负荷（如不同大小的组或者多个大小的奇数组合）的系统上，工作负荷将不能最佳地匹配可用 CPU 的数目。在这种情况下，如果不进行成组调度，则会使某些线程被调度而不是等待将所有线程当作一个组来调度，那么应用程序可能会获取更佳的性能。

调度系统开销

当调度程序收集一组线程、向线程分配一组处理器，以及集中一组线程和处理器以实现并行执行时，成组调度会造成一定的系统开销。

在空闲的系统上，在单个并行应用程序的执行时间内可以显示成组调度的系统开销。

线程的内核阻塞

如果组中的线程在内核中阻塞，则此线程的处理器可用于运行其他非成组调度的线程。如果阻塞的线程恢复操作，并且它的组当前正在运行，那么此线程可以连接其他成组的线程，而不必再次集中。

在多组环境中，线程阻塞可导致吞吐量降低。如果应用程序的线程经常在内核中长时间阻塞，则会发生这种情况。

实时线程的先占权

实时线程可以在执行时先占成组调度的线程。这只影响在实时线程先占的处理器上运行的成组调度线程。此组中余下的线程将继续运行，直到各自的时间片结束。

限制

成组调度的此种实现存在以下限制。将来的发行版可能会删除这些限制中的某些项目。

- 不支持成组调度调试的进程。当调试程序附加到进程后，将禁止对此进程进行成组调度。这可以避免调试程序停止对带有一个或多个线程的进程的成组调度。
- 启用 Process Resource Manager (PRM) 后，将完全关闭成组调度。
- 如果选择了一个要换出的成组调度进程，那么将此进程交换回来后，不再对其进行成组调度。
- 不成组调度实时进程。
- 只有带分时调度策略的进程才支持成组调度。
- 如果某个成组调度的进程包含最大的线程数（或者对于 MPI 应用程序而言，是最大进程数），则不会将此点之后创建的线程或进程当作组的一部分进行调度。有关详细信息，请参阅本联机帮助页的“行为”一节。
- 不使用 MPI 的多进程应用程序不受成组调度程序的支持。
- **PTHREAD_SCOPE_PROCESS** 线程不支持成组调度。从 HP-UX 11i v1.6 开始，线程的缺省调度争用范围为 **PTHREAD_SCOPE_PROCESS** 。如果应用程序创建了任何 **PTHREAD_SCOPE_PROCESS** 线程，则最初的那个线程将被视为 **PTHREAD_SCOPE_PROCESS** 。

文件

以下为提供成组调度时使用的库：

/usr/lib/libpthread.1 pthread 库。

/opt/mpi 包含 MPI 库和 MPI 软件的目录。HP MPI 是一个可选产品。

另请参阅

fork(2)、exec(2)。

名称

hil - HP-HIL 设备驱动程序

概要

#include <sys/hilioctl.h>

说明

HP-HIL（Hewlett-Packard Human Interface Link，HP 人性化接口链接）是用于将个人计算机、终端或工作站与各自的输入设备连接的 HP 标准。 **hil** 支持键盘、鼠标、控制旋钮、 ID 模块、按钮箱、数字转换器、正交信号设备、条码读取器、触摸屏等设备。

在具有单个链路的系统上， HP-HIL 设备文件名使用以下格式：

/dev/hil*n*

其中的 *n* 代表指定 HP-HIL 物理设备地址的单位数，其范围为 1 至 7。例如，使用 **/dev/hil3** 访问第三个 HP-HIL 设备。

在具有多个链路的系统上， HP-HIL 设备文件名使用以下格式：

/dev/hil_*m.n*

其中的 *m* 代表实例号， *n* 代表 HP-HIL 物理设备地址。例如，可以使用 **/dev/hil_0.2** 访问实例号为零的链路上的第二个设备。同样， **/dev/hil_12.7** 引用实例号为十二的链路上的第七个设备。

请注意，只能按照设备附加到链路时的顺序确定 HP-HIL 设备地址。第一个附加到链路的设备将成为设备一，第二个附加到链路的设备将成为设备二，依此类推。

HP-HIL 设备分类为“慢”设备。这意味着捕获的信号可以中断系统调用 **hil** （请参阅 *signal*(5) ）。

hil 只能读取原始键代码模式下的 HP-HIL 键盘。原始键代码模式是指在不过滤的前提下读取所有键盘输入。 HP-HIL 键盘可返回代表按键事件和释放按键事件的键代码。

可使用 *hilkbd*(7) 读取来自 HP-HIL 键盘的映射键代码。可使用 *termio*(7) 中介绍的内置仿真终端 (ITE) 读取来自 HP-HIL 键盘的 ASCII 字符。

系统调用

open(2) 可提供对指定的 HP-HIL 设备的独占访问权限。将忽略任何来自设备的、事先排队的输入。如果此设备为键盘，那么它是在原始键代码模式下打开的。在原始键代码模式下打开键盘的副作用是，在关闭此键盘之前，ITE （请参阅 *termio*(7) ）和映射键盘驱动程序（请参阅 *hilkbd*(7) ）将会丢失来自此键盘的输入。在原始键代码模式下，只有设备实现的自动重复功能才可用（请参阅 HILER1 和 HILER2）。

可设置文件状态标志 O_NDELAY，以启用非阻塞读取（请参阅 *open*(2) ）。

close(2) 可将一个 HP-HIL 键盘返回到映射的键代码模式，使其输入可用于 ITE 或映射的键盘驱动程序（请参阅 *hilkbd*(7) ）。

read(2) 可返回来自指定的 HP-HIL 设备的数据，这些数据的形式为带有时间戳的数据包：

unsigned char packet_length;
unsigned char time_stamp[4];
unsigned char poll_record_header;
unsigned char data[packet_length - 6];

packet_length 指定数据包内的字节数（包括数据包本身），指定值的范围为六至十二个字节。 *time_stamp* 被重新压缩成一个整数后，可指定系统自上次引导后运行的时间，其单位为数十毫秒。时间戳的最重要的字节为 *time_stamp*[0] 。 *poll_record_header* 指示要遵循的信息类型和数量，及报告简单的设备状态信息。数据字节数与设备相关。有关 *poll_record_header* 和设备特定的数据的说明，请参考“另请参阅”中所列的文本。

要读取每个数据包，通常需要两次系统调用。第一次系统调用将读取数据包的长度，第二次系统调用将读取实际的数据包。某些设备始终返回每个数据包中的相同数据量，在这种情况下，同一系统调用既可读取计数，也可读取数据包。

如果设置了文件状态标志 O_NDELAY，并且没有可用的数据，那么 *read*(2) 将返回 **0** 而不是阻塞。

hil 不支持 *write*(2) 。

可使用 *select*(2) 轮询来自 HP-HIL 设备的可用输入。用于处理写入或异常情况的 *select*(2) 始终在文件描述符位掩码中返回一个 False 指示。

可使用 *ioctl*(2) 对 HP-HIL 设备执行特殊操作。 *ioctl*(2) 系统调用使用以下格式：

int ioctl(int fildes, int request, char *arg);

<sys/hilioctl.h> 中定义了以下 *request* 代码：

HILID　　标识和描述

此请求将在 *arg* 指向的 **char** 变量中，返回指定的 HP-HIL 设备所提供的标识和描述记录。标识和描述记录用于确定连接到链路的每个设备的类型和特性。标识和描述记录的长度范围为 2 至 11 个字节。此记录至少包括：

- 设备 ID 字节，
- 以及描述记录头字节。

设备 ID 字节用于标识设备的常规类别，如果设备为键盘或数字键盘，还可标识其所属的国家（地区）。描述记录头字节用于描述设备的位置报告功能。描述记录头字节还指示是否在描述记录的结尾后跟一个 I/O 描述符字节。此外，它还指示是否支持“扩展描述”及“报告安全代码”请求。如果设备能够报告任何坐标，则描述记录将在描述记录头字节后面紧跟设备解析。如果设备可报告绝对坐标，则在设备解析的后面指定每个轴的最大计数。 I/O 描述符字节指示设备具有的按钮数。 I/O 描述符字节还指示设备邻近程度检测功能，并指定提示/确认功能。所有 HP-HIL 设备都支持“标识和描述”请求。

HILPST　　执行自检

此请求使编址的设备执行自检，并在 *arg* 指向的 **char** 变量中返回单字节检测结果。检测结果为零表示检测成功，结果为非零值表示设备特定的失败。所有 HP-HIL 设备都支持“自检”请求。

HILRR 读取寄存器

“读取寄存器”请求需要在 *arg* 指向的 **char** 变量中提供 HP-HIL 设备寄存器地址，此请求可以在 **arg* 中返回此寄存器的单字节内容。扩展描述记录可指示某个设备是否支持读取寄存器请求。

HILWR 写入寄存器

“写入寄存器”请求需要 **arg* 包含带有一个或多个数据包的记录，其中每个数据包都包含 HP-HIL 设备寄存器地址，及要写入此寄存器的一个或多个数据字节。共有两种类型的寄存器写入。可使用第 1 种类型将单个字节写入各个设备寄存器。可使用第 2 种类型将多个字节写入一个寄存器。扩展描述记录可指示某个设备是否支持这两种类型的寄存器写入请求中的一种或两种。

HILRN 报告名称

“报告名称”请求可在 *arg* 指向的字符数组中返回设备描述字符串。此字符串最长为十五个字符。扩展描述记录可指示是否支持报告名称请求。

HILRS 报告状态

“报告状态”请求可在 *arg* 指向的字符数组中返回设备特定的状态信息字符串。此字符串最长为十五个字节。扩展描述记录可指示是否支持报告状态请求。

HILED 扩展描述

“扩展描述”请求可在 *arg* 指向的字符数组中返回扩展描述记录。扩展描述记录最多包含有关其他设备信息的十五个字节。第一个字节为扩展描述头，用于指示设备是否支持“报告状态”、“报告名称”、“读取寄存器”或者“写入寄存器”请求。如果设备实现了“写入寄存器”请求，则指定最大可读寄存器。如果设备支持“写入寄存器”请求，则扩展描述记录将指定设备是否可实现两种类型的寄存器写入中的一种或两种，及是否可实现最大可写寄存器。如果设备支持第 2 种类型的寄存器写入，则指定写入缓冲区的最大大小。扩展描述记录还可包含设备的本地化（语言）代码。在描述记录头字节中指示了是否支持“扩展描述”请求。

HILSC 报告安全代码

“报告安全代码”请求可在 *arg* 指向的字符数组中返回安全代码记录。安全代码记录可以是一至十五个字节的数据，这些数据用于唯一标识特定的设备。应用程序可使用此请求实现一个硬件“钥匙”，使得对此应用程序的每次复制操作都限制为单个计算机或用户。应用程序可以从 HP-HIL ID 模块读取安全代码记录，然后验证此应用程序是否在特定的计算机上运行，或者此应用程序是否正由合法用户使用。设备将在描述记录头中指示是否支持“报告安全代码”请求。

HILER1 启用自动重复速率 = 1/30 秒

可使用此请求启用某些 HP-HIL 键盘和数字键盘设备的固件实现的“重复键”功能。此请求还可将光标键重复速率设置为 1/30 秒。此请求不使用 *arg* 。

HILER2　启用自动重复速率 = 1/60 秒

可使用此请求启用某些 HP-HIL 键盘和数字键盘设备的固件中实现的“重复键”功能。此请求还可将光标键重复速率设置为 1/60 秒。此请求不使用 *arg* 。

HILDKR　禁用键开关自动重复

此请求可关闭某些 HP-HIL 键盘和数字键盘设备的固件中实现的“重复键”功能。此请求不使用 *arg* 。

HILP1..HILP7　提示 1 至提示 7

某些 HP-HIL 设备支持这七个请求，可为用户提供声音响应或可视响应，多半指示系统已就绪，可接受某种类型的输入。设备将在描述记录的 I/O 描述符字节中指定是否接受这些请求。这些请求不使用 *arg* 。

HILP　提示（通用）

此请求可为用户提供通用提示。设备将在描述记录的 I/O 描述符字节中指定是否接受此请求。此请求不使用 *arg* 。

HILA1..HILA7　确认 1 至确认 7

这七个请求用于向用户提供声音响应或可视响应，通常用于确认用户输入。描述记录中的 I/O 描述符字节将指示 HP-HIL 设备是否可实现此请求。这些请求不使用 *arg* 。

HILA　确认（通用）

“确认”请求用于向用户提供声音响应或可视响应。设备将在描述记录的 I/O 描述符字节中指定是否接受此请求。此请求不使用 *arg* 。

错误

[EBUSY]　指定的 HP-HIL 设备已打开。

[EFAULT]　尝试使用系统调用的参数时，检测到一个错误的地址。

[EINTR]　一个信号中断了 *open*(2) 、 *read*(2) 或 *ioctl*(2) 系统调用。

[EINVAL]　*ioctl*(2) 检测到无效的参数。

[ENXIO]　在指定的地址不存在设备；请参阅下面的“警告”。

[EIO]　执行 *ioctl*(2) 系统调用时出现硬件或软件错误。

[ENODEV]　HP-HIL 设备没有实现 *write*(2) 。

警告

在电源故障恢复期间，当 **hil** 正在重新配置链路时，如果进行了访问设备的尝试， *open*(2) 和 *ioctl*(2) 将返回 ENXIO 错误。

hil 无法检测到设备是否执行了 *ioctl*(2) 请求。

HP-HIL 设备不提供用于指示它们是否支持 HILER1、HILER2 或 HILDKR 请求的状态位。

作者

hil 由 HP 开发。

文件

/dev/hil[1-7]
/dev/hil_*.[1-7]

另请参阅

close(2)、 errno(2)、 fcntl(2)、 ioctl(2)、 open(2)、 read(2)、 select(2)、 signal(5)、 hilkbd(7)、 termio(7)。

有关一般条件下的 HP-HIL 硬件和软件的详细信息，请参阅《HP-HIL Technical Reference Manual》。

名称

hilkbd - HP-HIL 映射键盘驱动程序

说明

HP-HIL（即 HP 人机接口链接），是将个人计算机、终端或工作站连接到其输入设备的 HP 标准。 **hilkbd** 提供来自于指定 HP-HIL 链接上所有映射键盘的输入。

hilkbd 将返回映射的键编码，而不是 ASCII 字符。“原始”键编码是单个键按下和弹起的编码，并且对于每种类型的键盘都不同。 **hilkbd** 将原始输入映射到 HP-UX、Pascal 工作站和 BASIC/UX 操作系统所需的键编码和协议中。 **hil** 驱动程序可以侵占 **hilkbd** 提供的键盘，方法是将该键盘从映射模式更改为原始模式。

系统调用

open() 可以提供对键盘的独占访问。如果存在与键盘相关联的 ITE （内部仿真终端）， ITE 会丢失键盘输入直到键盘设备关闭为止。键盘设备的任何先前排队的输入将从输入列队中清除。

close() 将键盘控制权返回给 ITE（如果有）。任何未读取输入将在此时被忽略。

read() 以时间戳包的形式返回键盘的数据：

unsigned char *time_stamp***[4];**
unsigned char *status***;**
unsigned char *data***;**

当重新打包为四字节或更多字节的整数数据类型时， *time_stamp* 指定自过去任意时间点以来的时间（例如，系统启动时间）。该时间点在数据包之间不会更改，但断电时间可能计算在内，也可能不计算在内。时间以数十毫秒为单位。

status 字节对键盘 **Shift** 键和 **Ctrl** 键的状态进行编码：

0x8X	shift 键和 control 键
0x9X	仅 control 键
0xAX	仅 shift 键
0xBX	无 shift 键或 control 键

data 字节包含实际键击。

如果设置了文件状态标记 O_NDELAY， **read()** 在无数据可用时将返回 **0** 而不是阻塞状态。对 HP-HIL 键盘的 **read()** 系统调用被视为“速度缓慢”；即，它可被捕获信号（请参阅 *signal*(2) ）中断。

hilkbd 不支持 **write()** 。

select() 可用于轮询输入以便从 **hilkbd** 设备读取。用于写入或异常情况的 **select()** 始终返回位掩码中的假指示。

ioctl() 用于对设备执行特殊操作。 **ioctl()** 系统调用的格式如下：

int ioctl(int fildes, int request, char *arg);

下列 **hilkbd** 请求代码在 **<sys/hilioctl.h>** 中定义：

KBD_READ_CONFIG

读取配置代码。

该请求将返回 *arg* 指向的 **char** 变量中的单字节配置代码。它包含一个由 **KBD_IDCODE_MASK** 定义的字段，该字段指定键盘标识代码。该字段的可能值在头文件中定义，该标识代码影响对语言代码的解释。目前未定义配置代码中的其他所有字段。

KBD_READ_LANGUAGE

读取语言代码。

从键盘读取时，该请求将返回 *arg* 指向的 **char** 变量中的单字节语言代码。如果有多个键盘，将采用链接上第一个键盘的语言。语言代码的解释受配置代码内键盘标识字段的影响。

KBD_STATUS 读取键盘状态寄存器。

该请求将返回 *arg* 指向的 **char** 变量中包含位标记（指定切换键和控制键的状态）的单字节值：

KBD_STAT_LEFTSHIFT	左 shift 键弹起
KBD_STAT_RIGHTSHIFT	右 shift 键弹起
KBD_STAT_SHIFT	左右 shift 键弹起
KBD_STAT_CTRL	control 键弹起

未定义其他位。

KBD_REPEAT_RATE

设置键盘自动重复速率。

arg 指向的单字节值是以数十毫秒为单位的重复周期的负数。重复速率是重复周期的倒数。参数为 0 将禁用自动重复。

KBD_REPEAT_DELAY

设置键盘自动重复延迟。

arg 指向的单字节值是以数十毫秒为单位的重复延迟的负数。

KBD_BEEP 导致发出嘟嘟声。

arg 指向的单字节值指定在范围 **0** 至 **KBD_MAXVOLUME** 内嘟嘟声的音量。音量小于 **KBD_MAXVOLUME** 离散级别的实现方案会将参数调整至更小的范围。

错误

[EINVAL] **ioctl()** 检测到无效参数。

[EINTR] 在 **read()** 系统调用期间捕获到一个信号。

[ENXIO] 次设备号指定的 HP-HIL 链接上无键盘。

[ENODEV] 试图通过 **hilkbd** 来使用 **write()** 。

[EBUSY] 设备已打开。

作者

hilkbd 由 HP 开发。

文件

/dev/hilkbd*

另请参阅

mknod(1M)、 select(2)、 signal(2)、 hil(7)、 termio(7)。

名称

inet - Internet 协议系列

概要

#include <sys/types.h>

#include <netinet/in.h>

说明

Internet 协议系列是一个位于 “**Internet** 协议” (IP) 网络层顶层的协议集合，它使用 Internet 地址格式。 Internet 系列支持 SOCK_STREAM 以及 SOCK_DGRAM 套接字类型。

地址

Internet 地址是四字节实体。包含文件 **<netinet/in.h>** 将该地址定义为 **struct in_addr** 结构。

绑定到 Internet 协议系列的套接字使用一个名为 **struct sockaddr_in** 的地址结构。只要系统调用请求指向 **struct sockaddr** 的指针，便可以在这些系统调用中使用指向该结构的指针。

该结构中有三个重要字段。第一个是 **sin_family** ，必须设置为 AF_INET。第二个是 **sin_port** ，它指定了要在所需主机上使用的端口号。第三个是 **sin_addr** ，属于 **struct in_addr** 类型，并且指定所需主机的地址。

协议

Internet 协议系列由 IP 网络协议、 Internet 控制消息协议 (ICMP) 、传输控制协议 (TCP) 和用户数据报协议 (UDP) 组成。 TCP 用于支持 **SOCK_STREAM** 套接字类型，而 UDP 用于支持 **SOCK_DGRAM** 套接字类型。 ICMP 消息协议和 IP 网络协议不能直接访问。

本地端口地址从 TCP 和 UDP 套接字的独立域中选择。这表示创建一个 TCP 套接字并将它绑定到本地端口号 10000（例如），不会对同时创建一个 UDP 套接字并将它绑定到本地端口号 10000 产生妨碍。

1-1023 范围内的端口号将保留仅供超级用户使用。非超级用户试图绑定到该范围内的端口号，会失败并导致返回错误。

作者

inet 由加州大学伯克利分校开发。

另请参阅

tcp(7P)、 udp(7P)。

名称

iomap - 物理 I/O 地址映射

概要

#include <sys/iomap.h>

说明

iomap 机制允许将物理 I/O 地址映射到用户进程地址空间，从而实现直接访问。对于 PA-RISC 计算机，物理 I/O 地址空间从 **0xf0000000** 开始并且扩展到 **0xffffffff** 。

iomap 设备的专用（设备）文件是使用动态主设备号分配方案的字符专用文件。

iomap 设备的次设备号格式如下：

0xAAAASM

通过在 0xAAAA 前面加前缀 0xF 并在后面追加 0x000，可形成物理 I/O 地址（这会强制 I/O 地址实现页对齐）。要映射区域的大小由表达式 M*(2^S) 4K 页给出。例如，从 **0xf4000000** 开始、占用 64MB 的设备的次设备号是 **0x4000e1** 。

在首次使用 **iomap** 之前，必须将 **iomap** 驱动程序明确添加到 **/stand/system** 文件中，重建内核，随后重新引导系统。

无论设备专用文件的实际权限如何， I/O 空间始终在既有读取访问权限又有写入访问权限的情况下映射。

多个进程可以同时打开并映射一个 **iomap** 设备。这些进程的责任就是使其访问同步。

连续调用 **iomap** 映射相同的 I/O 空间必须与首次映射相同。相同的映射具有相同的地址和大小。

请注意，进程可以与内核驱动程序额外共享 I/O 空间（由 **iomap** 映射）。但是，仅当驱动程序通过用户读取（或写入）访问权限，使用适当的驱动程序 I/O 映射服务在 I/O 空间映射时，才有可能发生以上情形。由驱动程序通过内核读取（或写入）访问权限映射的任何 I/O 空间，不能由使用 **iomap** 的进程同时映射。

iomap 驱动程序不支持 **read()** 或 **write()** 系统调用。

ioctl() 函数用于控制 **iomap** 设备。下列 **ioctl()** 请求在 **<iomap.h>** 中定义：

IOMAPMAP 将 **iomap** 设备映射到设备地址空间，其位置由 **ioctl()** 的 **(void **)** 第三个参数指向的指针指定。如果该参数指向一个包含空指针的变量，系统将选择适当的地址。 **ioctl()** 随后返回在其中映射了设备的用户地址，将其存储在第三个参数指向的地址中（请参阅下面的“举例”一节）。多个进程可以同时映射同一 **iomap** 设备。

IOMAPUNMAP 从用户地址空间中取消 **iomap** 设备的映射。

close() 关闭与 **iomap** 设备相关联的文件描述符。如果关闭操作是针对设备上的最后一个系统级打开操作，则还会从用户地址空间取消 **iomap** 设备的映射；否则，它保持为映射到用户地址空间（请参阅上面的 **IOMAPUNMAP**）。

警告

创建和使用 **iomap** 设备时应特别小心。对 I/O 设备或 RAM 的不当访问会导致系统崩溃。

错误

[EINVAL] 地址字段超出范围，或 **ioctl** 请求无效。

[ENOMEM] 内存不足，无法分配供映射使用。

[EBUSY] 设备已映射，但该映射与初始映射不完全相同（地址、大小和访问权限相同）。

[ENODEV] 不支持读取和写入调用。

[ENXIO] 在次设备号指定的地址上没有这样的设备。

[ENOSPC] 无法分配映射所需资源。

[ENOTTY] 对该设备类型的 **ioctl** 请求不适当； *fildes* 不是 **iomap** 设备文件的文件描述符。

举例

请考虑以下代码段：

```
#include <sys/iomap.h>
        ...
int fildes;
void *addr;
        ...
   addr = REQUESTED_ADDRESS;
   (void) ioctl(fildes, IOMAPMAP, &addr);
   (void) printf("actual address = 0x%x\n", addr);
```

其中， **fildes** 是设备专用文件的打开文件描述符， **REQUESTED_ADDRESS** 是程序最初请求的地址。

如果 **addr** 是空指针，系统将选择合适的地址，然后在 *addr* 中返回所选的地址。

如果 *addr* 的值不是空指针，则将它用作分配内存时所用的指定地址。如果无法使用指定地址，将返回错误（请参阅“错误”一节）。

另请参阅

mknod(1M)。

名称

IPv6、ipv6、ip6 - Internet 协议第 6 版

概要

#include <sys/socket.h>
#include <netinet/in.h>

s = socket(AF_INET6, SOCK_DGRAM, 0);
s = socket(AF_INET6, SOCK_STREAM, 0);

说明

IPv6 是下一代网络层协议，它设计为用作当前 Internet 协议第 4 版 (IPv4) 的后继版本。它可以为 TCP、UDP 和 ICMPv6 提供数据包传递服务。

在增大的地址空间、简化的头格式、集成的 QoS 支持及强制安全性方面， IPv6 比 IPv4 具有明显的优势。IPv6 还允许在称为“扩展头”的单独头内对可选的 Internet 层信息进行编码，这些头位于 IPv6 头和上层头之间。当前支持的扩展头包括逐跳选项头，目标选项头，段头和路由选择（类型 0）头。IPv6 数据包可携带零个、一个或多个扩展头，每个头由前置头的下一个头字段标识。

IPv6 将地址大小由 32 位扩展到 128 位，它们用“十六进制-冒号”表示法以文字形式表示为 **x:x:x:x:x:x:x:x** ，其中 **x** 是地址的八个 16 位片段的十六进制值。例如， **fedc:83ff:fef6:417a:210:83ff:fef6:3dc0** 。

IPv6 包括三种类型的地址： 单播、 任意播和 组播。

- “单播地址”是单个接口的标识符。传递到单播地址的数据包将被传递给此地址标识的接口。
- “任意播地址”是一组接口的标识符。传递到任意播地址的数据包将被传递给此地址标识的接口之一。
- “组播地址”是一组接口的标识符。传递到组播地址的数据包将被传递给此地址标识的所有接口。

  IPv6 中不包括广播地址，这些地址的功能已由组播地址替代。

每个 IPv6 地址都有一个与自身关联的 范围。此范围是一种拓扑范围，可以在其中将地址用作一个接口或一组接口的唯一标识符。

单播地址包括三个定义的范围：链路本地、站点本地和全局。

- 链接本地地址可唯一标识单个链路中的接口，它带有固定的前缀 **fe80::/10** 。例如， **fe80::210:84c0:ef6f:cd30** 。
- 站点本地地址只可以唯一标识单个站点中的接口，它带有固定的前缀 **fec0::/10** 。例如， **fec0::210:84c0:ef6f:cd30** 。
- 全局地址可唯一标识互联网上的任何接口。

有 2 种专用的单播地址，用于保留低次位 32 位中的嵌入式 IPv4 地址。

- 第一种类型称为“与 IPv4 兼容的 IPv6 地址”，其格式为 **0:0:0:0:0:0:d.d.d.d** 。双堆栈 (IPv4/IPv6) 节点将使用此类型的地址执行 IPv6-over-IPv4 自动通道处理，其中，IPv4 通道端点地址由嵌入在进行通道处理的 IPv6 数据包的 IPv4 兼容目标地址中的 IPv4 地址确定。
- 第二种类型称为“IPv4 映射的 IPv6 地址”，其格式为 **0:0:0:0:0:ffff:d.d.d.d** 。此地址有助于 IPv6 应用程序与 IPv4 应用程序之间的相互操作。如果指定的主机只包括 IPv4 地址，则应用程序可使用 **getaddrinfo()** （请参阅 *getaddrinfo*(3N) ）自动生成此地址。

IPv6 套接字选项

为 IPv6 定义了新的套接字选项，可通过这些选项发送和接收扩展头，以及在内核和应用程序之间交换其他可选的信息。这些选项在 **IPPROTO_IPV6** 协议级别受支持。 *optval* 参数指向的变量的类型在圆括号中进行了指示。

IPV6_UNICAST_HOPS (**integer**) 设置或获取出站单播数据包中使用的跳段限制。如果使用 **setsockopt()** （请参阅 *setsockopt*(2) ）设置了此选项，则指定的新选项值将用作通过此套接字发送的所有后续单播数据包的跳段限制。有效值的范围为 0 至 255（包括两个边界值），缺省值为 64。例如，

```
int hoplimit = 50;
setsockopt(s, IPPROTO_IPV6, IPV6_UNICAST_HOPS,
    &hoplimit, sizeof(hoplimit));
```

可以将此选项与 **getsockopt()** （请参阅 *getsockopt*(2) ）一起使用，以确定系统为通过此套接字发送的后续单播数据包使用的跳段限值。

IPV6_MULTICAST_HOPS (**integer**) 设置或获取出站组播数据包中使用的跳段限制。如果设置了此选项，则指定的新选项值将用作通过此套接字发送的所有后续组播数据包的跳段限制。有效值的范围为 0 至 255（包括两个边界值），缺省值为 1。

IPV6_MULTICAST_IF (**integer**) 设置出站组播数据包使用的接口。此选项值为选中的出站接口的索引。例如，

```
unsigned int index;
index = if_nametoindex("lan0");
setsockopt(s, IPPROTO_IPV6, IPV6_MULTICAST_IF,
    &index, sizeof(index));
```

IPV6_MULTICAST_LOOP

(**boolean**) 启用或禁用通过此套接字发送的组播数据报的 IP 层中的环回。 *optval* 指向的变量的值为零（禁用）或非零（启用）。缺省值：启用。

IPV6_JOIN_GROUP (**struct ipv6_mreq**) 连接指定的本地接口上的组播组。应使用 **<netinet/in6.h>** 中定义的 **struct ipv6_mreq** ，将要连接的组的 IPv6 组播地址以及连接所用的接口的索引指定为：

```
struct ipv6_mreq {
```

```
  struct in6_addr ipv6mr_multiaddr;
            /* IPv6 multicast addr */
  unsigned int  ipv6mr_interface;
            /* interface index */
};
```

如果接口索引指定为 0，则使用缺省的组播接口。

IPV6_LEAVE_GROUP (**struct ipv6_mreq**) 将组播组保留在指定的本地接口上。应使用 **struct ipv6_mreq** 指定要保留的组的 IPv6 组播地址以及接口索引。此接口索引应与连接组时使用的索引匹配。将索引设置为 0 可指定缺省的接口。

IPV6_CHECKSUM (**integer**) 如果设置了此选项，内核将计算出站数据包的校验和，并验证入站数据包的校验和。此选项值为用户数据中的校验和位置的字节偏移量。由于校验和计算对于 **IPPROTO_ICMPV6** 是强制的，因此此选项对于 **IPPROTO_ICMPV6** 无效。缺省值为 -1 （既不计算校验和，也不针对 **IPPROTO_ICMPV6** 以外的协议执行验证）。

IPV6_RECVPKTINFO (**boolean**) 如果启用此选项， **recvmsg()** 会将 **PKTINFO** （目标 IPv6 地址和收到的接口索引）作为辅助数据返回（请参阅 *recvmsg*(2) ）。此信息将在 **struct in6_pktinfo** 结构中返回，并在 **<netinet/in6.h>** 结构中其定义为：

```
struct in6_pktinfo {
    struct in6_addr ipi6_addr;
    uint32_t        ipi6_ifindex;
};
```

缺省情况下，此选项是禁用的。

IPV6_RECVHOPLIMIT (**boolean**) 如果启用此选项， **recvmsg()** 会将入站数据包的跳段限制作为辅助数据返回。例如，

```
int on = 1;
setsockopt(s, IPPROTO_IPV6, IPV6_RECVHOPLIMIT,
        &on, sizeof(on));
```

缺省情况下，此选项是禁用的。

IPV6_RECVDSTOPTS (**boolean**) 如果启用此选项， **recvmsg()** 会将入站数据包的目标选项（如果有的话）作为辅助数据返回。缺省情况下，此选项是禁用的。

IPV6_RECVHOPOPTS (**boolean**) 如果启用此选项， **recvmsg()** 会将入站数据包的逐跳选项（如果有的话）作为辅助数据返回。缺省情况下，此选项是禁用的。

IPV6_RECVRTHDR (**integer; boolean**) 如果启用此选项， **recvmsg()** 会将入站数据包的路由选择选项（如果有的话）作为辅助数据返回。缺省情况下，此选项是禁用的。

IPV6_RECVRTHDRDSTOPTS

(**integer; boolean**) 如果启用此选项， **recvmsg()** 会将显示在路由选择头（如果有的话）前面的入站数据包的目标选项作为辅助数据返回。缺省情况下，此选项是禁用的。

接下来的七个套接字选项既可以与 **setsockopt()** 一起使用，也可以用作 **sendmsg()** （请参阅 *sendmsg*(2) ）的辅助数据中的选项名。

IPV6_PKTINFO (**struct in6_pktinfo**) 用于设置出站数据包的源地址和接口索引。

IPV6_HOPLIMIT (**integer**) 用于设置出站数据包的跳段限制。此跳段限制只对单个输出操作有效。要设置所有单播 IPv6 数据包或组播 IPv6 数据包的跳段限制，请分别使用 **IPV6_UNICAST_HOPS** 或 **IPV6_MULTICAST_HOPS** 选项。

IPV6_NEXTHOP (**struct sockaddr_in6**) 用于设置下一个跳段地址。由此地址标识的节点必须与发送主机相邻。如果此地址与目标 IPv6 地址相同，则选项等效于 **SO_DONTROUTE** 套接字选项。

IPV6_RTHDR (**variable length**) 用于指定出站数据包的路由选择头。目前只支持类型 0 路由选择头。

IPV6_DSTOPTS (**variable length**) 用于指定后续 IPv6 数据包中要发送的一个或多个目标选项。

IPV6_HOPOPTS (**variable length**) 用于指定后续 IPv6 数据包中要发送的一个或多个逐跳选项。

IPV6_RTHDRDSTOPTS (**variable length**) 用于指定路由选择头前面的一个或多个目标选项。发送数据包时，将以无提示方式忽略此选项，除非还指定了一个路由选择头。

IPv6 使用称为“ICMPv6”的增强版 ICMP 来报告处理数据包时遇到的错误，此外还将它用于诊断目的（例如 ping）。ICMPv6 是 IPv6 的组成部分，它的下一个头值为 58。

<netinet/in6.h> 中定义了所有选项及关联的结构，应用程序不需要显式包括此头文件， **<netinet/in.h>** 会自动包括此头文件。

错误

如果套接字操作失败，则可能返回下列错误之一。

[EADDRINUSE] 已经连接了指定的组播组。

[EADDRNOTAVAIL] 指定的 IPv6 地址不是本地接口地址，或者指定的组播地址不存在路由，或者未连接指定的组播组。

[EINVAL] 参数“level”不是 **IPPROTO_IPV6** ，或者 *optval* 为空地址，或者指定的组播地址无效，或者指定的跳段限制不在范围 0 <= x<= 255 内。

[ENOBUFS] 没有为内部系统数据结构提供足够的内存。

[ENOPROTOOPT] 参数 optname 对于 **IPPROTO_IPV6** 级别来说，不是有效的套接字选项。

作者

IP 使用的这些套接字接口由加州大学伯克利分校开发。

另请参阅

bind(2)、 getsockopt(2)、 recv(2)、 send(2)、 socket(2)、 inet6_opt_init(3N)、 inet6_rth_space(3N), inet(7F)、 ndp(7P)。

RFC 2460 Internet 协议第 6 版。
用于 IPv6 的 RFC 2553 Basic Socket Interface Extensions。
用于 IPv6 的 RFC 2292 Advanced Socket Interface Extensions。

名称

IP - Internet 协议

概要

#include <sys/socket.h>

#include <netinet/in.h>

s = socket(AF_INET, SOCK_DGRAM, 0);

说明

IP 是 Internet 协议系列使用的网络层协议。它将 TCP 和 UDP 消息封装到数据报中，由网络接口传输。通常情况下，应用程序不需要通过接口直接与 IP 通信。但是，通过使用 UDP 套接字将选项传递到 **IPPROTO_IP** 协议级别，来控制某些组播套接字选项；通过使用 TCP 或 UDP 套接字将选项传递到 **IPPROTO_IP** 协议级别，来控制服务的 IP 类型（请参阅 *getsockopt*(2) 联机帮助页）。

下列套接字选项在包含文件 **<netinet/in.h>** 中定义。 *optval* 参数指向的变量的类型在括号中指明。数据类型 **struct ip_mreq** 和 **struct in_addr** 在 **<netinet/in.h>** 中定义。

IP_TOS (**unsigned int**) 设置服务的 IP 类型。 optval 的允许值有 4（可获得高可靠性）、 8（可获得高吞吐量）和 16（可获得低延迟）。其他值不会返回错误，但可能会出现不可预知的结果。缺省值： 0。

IP_ADD_MEMBERSHIP (**struct ip_mreq**) 请求系统加入组播组。

IP_DROP_MEMBERSHIP (**struct ip_mreq**) 允许系统退出组播组。

IP_MULTICAST_IF (**struct in_addr**) 通过该套接字发送组播数据报时，指定除缺省使用的接口之外的网络接口。缺省值：组播数据报通过缺省组播路由或缺省路由，从与特定组播组相关联的接口发送。

IP_MULTICAST_LOOP （**unsigned char**；布尔型）对于通过该套接字发送的组播数据报，在 IP 层中启用或禁用环回。 *optval* 指向的变量的值是零（禁用）或非零（启用）。该选项只是为了兼容而提供。通常情况下，如果系统已加入组，则组播数据报始终环回。请参阅下面的相关内容。缺省值：启用。

IP_MULTICAST_TTL (**unsigned char**) 对于通过该套接字发送的组播数据报，指定生存时间值。 *optval* 指向的变量的值可介于 0 和 255 之间。缺省值： 1。

IP_ADD_MEMBERSHIP 请求系统加入指定接口上的组播组。例如：

```
struct ip_mreq mreq;
mreq.imr_multiaddr.s_addr = net_addr("224.1.2.3");
mreq.imr_interface.s_addr = INADDR_ANY;
setsockopt(s, IPPROTO_IP, IP_ADD_MEMBERSHIP, &mreq, sizeof(mreq));
```

系统必须加入接口上的组，才能接收该接口所连接网络上发送的组播数据报。如果 **imr_interface** 设置为 **INADDR_ANY** ，则系统将加入接口上的指定组，该组的数据报将根据路由选择配置从此接口发送。否则，

imr_interface 应为本地接口的 IP 地址。对于每个套接字，应用程序最多可加入 **IP_MAX_MEMBERSHIPS** 个组播组。 **IP_MAX_MEMBERSHIPS** 在 **<netinet/in.h>** 中定义。但是，由于接口资源限制以及系统使用某些链接层组播地址，因此，可能对每个网络接口施加了较小系统级限制。

应用程序还必须绑定到目标端口号，以便接收发送到该端口号的数据报。如果应用程序绑定到地址 **INADDR_ANY** ，则它可以接收发送到该端口号的所有数据报。如果应用程序绑定到组播组地址，则它只能接收发送到该组及端口号的数据报。要向应用程序发送数据报，则应用程序不必加入组播组。

IP_DROP_MEMBERSHIP 允许系统退出组播组。例如：

```
struct ip_mreq mreq;
mreq.imr_multiaddr.s_addr = net_addr("224.1.2.3");
mreq.imr_interface.s_addr = INADDR_ANY;
setsockopt(s, IPPROTO_IP, IP_DROP_MEMBERSHIP, &mreq, sizeof(mreq));
```

系统会一直保留为组播组成员，直到用于加入组的最后一个套接字关闭，或者丢弃了组成员身份。

IP_MULTICAST_IF 指定通过该套接字发送组播数据报时要使用的本地网络接口。例如：

```
#include <arpa/inet.h>
struct in_addr addr;
addr.s_addr = inet_addr("192.1.2.3");
setsockopt(s, IPPROTO_IP, IP_MULTICAST_IF, &addr, sizeof(addr));
```

通常情况下，应用程序不需要指定接口。缺省情况下，组播数据报使用缺省组播路由或缺省路由，从路由选择配置指定的接口（即，与指定组播组相关联的接口）发送。如果 **addr** 设置为地址 **INADDR_ANY** ，则选择缺省接口。否则， **addr** 应为本地接口的 IP 地址。

对于通过该套接字发送的组播数据报， **IP_MULTICAST_LOOP** 可启用或禁用环回。例如：

```
unsigned char loop = 1;
setsockopt(s, IPPROTO_IP, IP_MULTICAST_LOOP, &loop, sizeof(loop));
```

请注意， *optval* 参数的类型是 **unsigned char** 而不是 **int** ，这对布尔套接字选项是通用的。该选项只是为了兼容而提供。通常情况下，如果将组播数据报发送到系统已加入的组，则数据报副本始终环回，并被传递到绑定到目标端口的所有应用程序。请参阅下面的相关内容。

对于通过该套接字发送的组播数据报， **IP_MULTICAST_TTL** 通过设置生存时间值来控制组播范围。例如：

```
unsigned char ttl = 64;
setsockopt(s, IPPROTO_IP, IP_MULTICAST_TTL, &ttl, sizeof(ttl));
```

请注意， *optval* 参数的类型是 **unsigned char** 而不是 **int** ，这对套接字选项是通用的。缺省情况下，生存时间字段 (TTL) 为 1，它表示将组播限于本地网络。如果 TTL 为 0，则组播限于本地系统（环回）。如果 TTL 为 2，则组播最多可通过一个网关转发；依此类推。仅当本地和中间网络上有专用组播路由器时，组播数据报才能转发到其他网络上。

相关内容

IP_MULTICAST_LOOP 的行为取决于网络驱动程序和接口卡。通常情况下，不能禁用环回，即使 **IP_MULTICAST_LOOP** 设置为 0 也是如此，这是因为它出现在驱动程序或网络接口中。但是，如果出站接口是 **lo0** (127.0.0.1)，或者如果 **IP_MULTICAST_TTL** 设置为 0，则将 **IP_MULTICAST_LOOP** 设置为 0 会禁用通过套接字发送的组播数据报的环回。

错误

如果调用 **setsockopt()** 或 **getsockopt()** 失败，则可能返回下列错误之一。

[EADDRINUSE]	对于该套接字，系统已加入指定的组播组。
[EADDRNOTAVAIL]	指定的 IP 地址不是本地接口地址；或没有用于指定组播地址的路由；或系统未加入指定的组播组。
[EINVAL]	参数 *level* 不是 **IPPROTO_IP** ；或 *optval* 是空地址；或指定的组播地址无效。
[ENOBUFS]	内部系统数据结构的内存不足。
[ENOPROTOOPT]	参数 *optname* 不是 **IPPROTO_IP** 级别的有效套接字选项。
[EOPNOTSUPP]	套接字类型不是 **SOCK_DGRAM** 。
[ETOOMANYREFS]	对于某个套接字，尝试加入的组播组数量大于 **IP_MAX_MEMBERSHIPS** 。

作者

IP 套接字接口由加州大学伯克利分校开发。组播扩展由斯坦福大学开发。

另请参阅

bind(2)、 getsockopt(2)、 recv(2)、 send(2)、 socket(2)、 inet(7F)。

名称

kmem - 根据符号名在内核内存上执行 I/O

概要

#include <sys/ksym.h>

int ioctl(
int *kmemfd*,
int *command*,
**void ****rks*
);

说明

当与 **/dev/kmem** (*kmemfd*) 的有效文件描述符一起使用时， **ioctl** 可用于处理内核内存。该处理的具体情况取决于如下 *command* :

MIOC_READKSYM 读取始于 *mirk_symname* 地址的 *mirk_buflen* 字节内核内存到 *mirk_buf* 。 *rks* 是指向 **mioc_rksym** 结构的指针，其定义如下。

MIOC_IREADKSYM 间接读取。读取始于 *mirk_symname* 地址的 **sizeof(void *)** 字节内核内存，并将其用作从中读取 *mirk_buflen* 字节内核内存到 *mirk_buf* 的地址。 *rks* 是指向 **mioc_rksym** 结构的指针。

MIOC_WRITEKSYM 从 *mirk_buf* 将 *mirk_buflen* 字节写入到始于 *mirk_symname* 地址的内核内存中。 *rks* 是指向 **mioc_rksym** 结构的指针。

MIOC_IWRITEKSYM 间接写入。读取始于 *mirk_symname* 地址的 **sizeof(void *)** 字节内核内存，并将其用作向其中写入 *mirk_buf* 的 *mirk_buflen* 字节的内核内存地址。 *rks* 是指向 **mioc_rksym** 结构的指针。

MIOC_LOCKSYM 将名称由指向字符串的指针 *rks* 指定的动态加载模块的保留计数增加 1，从而防止其卸载。

MIOC_UNLOCKSYM 将名称由指向字符串的指针 *rks* 指定的动态加载模块的保留计数减少 1。如果计数因此减少到 0，模块则成为准备卸载的候选模块。

struct mioc_rksym 定义是:

```
struct mioc_rksym {
    char * mirk_modname;      /* limit search for symname
                                 to module modname; if NULL
                                 use standard search order */
    char * mirk_symname;      /* name of symbol whose address
                                 is the basis for this operation */
    void * mirk_buf;          /* buffer into/from which
                                 read/write takes place */
```

```
    size_t mirk_buflen;          /* length (in bytes) of desired operation */
};
```

返回值

ioctl 返回下列值之一：

0 成功完成。

-1 失败。设置 **errno** 以指示错误。

错误

除 *ioctl*(2) 中所述的值，如果检测到相应情况， **kmem ioctl** 还会将 **errno** 设置为下列值之一：

[EINVAL] *modname* 不表示当前加载模块或这是 **MIOC_UNLOCKSYM** 并且保留计数已为 0。

[ENXIO] *kmemfd* 在错误从设备中打开（即，不是 **/dev/kmem**）。

[EBADF] *kmemfd* 打开用于读取，并且这是一个 **MIOC_WRITEKSYM** 。

[ENOMATCH] 未找到 *symname* 。

[ENAMETOOLONG]
modname 长于 **MODMAXNAMELEN** 字符长度，或 *symname* 长于 **MAXSYMNMLEN** 字符长度。

另请参阅

getksym(2)、 ioctl(2)、 ioctl(5)。

名称

lan - 网络 I/O 卡访问信息

说明

本手册条目简单介绍了如何访问 OSI 体系结构第 2 层（数据链路层）上的局域网设备驱动程序。局域网设备驱动程序控制第 1 层（物理层）上的各种局域网接口卡（例如，以太网 /IEEE 802.3、FDDI 和令牌环）。

数据链路提供程序接口 (DLPI) 是所支持的用于访问第 2 层上局域网设备驱动程序的方法。DLPI 仅供行业知识丰富的网络用户使用。有关完整的编程详细信息，请参考《DLPI Programmer's Guide》。

有的 HP 和非 HP 驱动程序和接口卡将提供自己的 DLPI 模块。这些类型的 DLPI 被称为“本地”DLPI。

概述

物理连接点 (PPA) 是一个唯一标识特定设备的数值。PPA 值可从 **nwmgr** 和 **lanscan** 命令获取。 **nwmgr** 输出中的“ClassInstance”标识符是由驱动程序类 (lan) 和 PPA 编号连接而成。 **lanscan** 输出中的“NamePPA”标识符是由接口名称和 PPA 编号连接而成。局域网设备的 **card instance** 值等于该设备的 PPA 编号。

一个硬件设备可以有多个“NamePPA”标识符，这表明该设备支持多种封装方法。对于以太网/IEEE 802.3 链路，“名称” **lan** 用于指定以太网封装，而 **snap** 用于指定 IEEE 802.3 封装。对于其他链接（FDDI 和令牌环），仅使用 **lan** 指定封装。

通过局域网设备在 DLPI 接口上的传输方法包括“原始”、“无连接”和“面向连接”数据传输。

警告

不推荐使用 **lanadmin** 、 **lanscan** 和 **linkloop** 命令。在将来的 HP-UX 版本中将删除这些命令。 HP 建议使用替换命令 *nwmgr*(1M) 来执行所有与网络接口相关的任务。

作者

lan 由 HP 开发。

另请参阅

lanscan(1M)、 lanadmin(1M)、 linkloop(1M)、 nwmgr(1M)。

《DLPI Programmer's Guide》，1995 年，HP

《The Ethernet, A LAN:Data Link Layer and Physical Specification》，第 2 版，1982 年 11 月，Digital Equipment Corporation， Intel Corporation，Xerox Corporation

《CSMA/CD Access Method and Physical Layer Specification》，1996 年，电气与电子工程师协会

《Demand-Priority Access Method，Physical Layer & Repeater Specifications》，1996 年，电气与电子工程师协会

《Fiber Distributed Data Interface (FDDI) Physical Layer Medium Dependent (PMD)》，1995 年，ANSI

《Token Ring Access Method and Physical Layer Specification》，1995 年,电气与电子工程师协会

《802.3u Media Access Control Parameters, Physical Layer, Medium Attachment Units, and Repeater for 100 Mb/s Operation, Type 100BASE-T》，1995 年，电气与电子工程师协会

名称

ldterm - 标准 STREAMS 终端线路规则模块

概要

#include <sys/stream.h>
#include <sys/stropts.h>
#include <sys/termios.h>
#include <sys/bsdtty.h>
#include <sys/ttold.h>
#include <sys/strtio.h>
#include <sys/eucioctl.h>

int ioctl(fd, I_PUSH, "ldterm");

说明

ldterm 是为基于流的终端或伪终端设备驱动程序提供线路规则的 STREAMS 模块。此模块可提供 *termio*(7) 中介绍的常规终端接口的大多数函数，但是，它不能执行 POSIX **termios** 结构或 System V **termio** 结构（分别在 **termios.h** 和 **termio.h** 中定义）定义的 **c_cflag** 字所指定的低级设备控制函数。同时，某些操作需要利用压入到 tty 或 pty（从属）流中 **ldterm** 模块下面的模块和驱动程序的协作关系。本联机帮助页只介绍了 **ldterm** 特定的接口，有关终端接口的详细信息，建议读者参考 *termio*(7) 。

ldterm 模块在内部使用扩展 UNIX 代码 (EUC) 字符编码方案。 **ldterm** 模块可通过此编码方案处理多字节字符和简单的 8 位字符。它能够正确处理多字节 EUC 字符的退格、字清除和制表符扩展。

ldterm 模块可提供符合 POSIX 1003.1 和 System V Interface Definition (SVID) 第三版指定行为的标准终端操作。它还可以提供与 BSD 4.3 线路规则行为的兼容性。请注意，在其他 STREAMS 系统上，BSD 4.3 兼容功能通常是由称作 **ttcompat** 的单独 STREAMS 模块提供的。因此，HP-UX 上的应用程序不需要将 **ttcompat** 压入到 **ldterm** 之上来实现 BSD 4.3 兼容性。事实上，HP-UX 系统上根本没有提供 **ttcompat** 模块。

通常， **ldterm** 模块位于 STREAMS tty 驱动程序的上方，或 STREAMS pty 从属驱动程序的上方。一旦打开了 STREAMS tty 或 STREAMS pty 从属设备，用户便发出 **STREAMS I_PUSH** *ioctl*(2) 系统调用，以便将 **ldterm** 压入到流中。

STREAMS 消息

ldterm 模块可处理各种类型的 STREAMS 消息。线路规则适用于以下任何消息类型。但是，模块收到的其他任何类型的消息都将传递到流中的下一个模块。

读取端行为

ldterm 可处理其输入流上的下列 STREAMS 消息：

M_FLUSH

如果设置了 **FLUSHR** ，则 read put 例行程序将刷新读取队列，同时忽略输入消息缓冲区中的字符，并忽略任何部分缓冲的多字节 EUC 字符。然后，此例行程序将上行转发此消息。

M_BREAK

read put 例行程序根据处理 **BREAK** 事件、奇偶错误、成帧错误和信号生成（有关详细信息，请参阅 *termio*(7) ）的 POSIX 规则来处理此消息。如果消息中没有数据，则假定此消息表示一个输入 **BREAK** 事件，此事件通过字符值为 0（零）的成帧错误表示。如果消息中有数据，则数据值将为一个整数，表示出现了输入 **BREAK** 事件，或者接收字符时遇到奇偶错误或成帧错误。此数据值的低次 8 位是已读取的字节。如果此整数的高阶位中设置了 **TTY_PE** 标记，则表示检测到奇偶错误。如果此整数的高阶位中设置了 **TTY_FE** 标记，则表示检测到成帧错误。

读取数据值后，read put 例行程序将忽略此消息。

M_DATA read put 例行程序根据 POSIX 1003.1 规范，使用针对退格、字清除和制表符扩展的多字节处理（如果适用）来处理此消息。

此例行程序生成回显字符，并将这些字符放入到输出缓冲区中，以便下行发送到写入队列。在处理传入数据时，此例行程序将扫描 **START** 和 **STOP** 字符，并在必要时将 **M_START** 、 **M_STOP** 消息下行发送到写入队列。

如果缓冲的输入字符总数大于高水印，并且设置了 **IXOFF** 则 read put 例行程序将下行发送 **M_STOPI** 消息。如果队列将其后备字符减少到低水印以下，则此例行程序将下行发送 **M_STARTI** 消息。

如果缓冲的输入字符数达到 **MAX_INPUT** ，并且设置了 **IMAXBEL** 标记，那么 read put 例行程序将忽略新的输入字符，并下行发送一个 **BEL** 字符 (**Ctrl-G**) 。如果未设置 **IMAXBEL** ，此例行程序将刷新输入队列。

如果设置了 **ISIG** 标记，那么当遇到适当的信号字符时， read put 例行程序将上行发送 **M_PCSIG** 消息，然后再忽略这些字符。

如果遇到了与 **c_cc[VDISCARD]** 匹配的字符，并且设置了 **IEXTEN** 标记，那么 read put 例行程序将上行发送 **M_FLUSH** (**FLUSHW**) 消息，以刷新所有的写入队列。 **M_FLUSH** 消息由流头部反映，并被下行发送到所有写入队列。

如果此字符表示输入的逻辑终止，那么 read put 例行程序会将当前缓冲的字符上行发送到流头部。

输入的逻辑终止取决于 **ICANON** 标记的状态。如果设置了 **ICANON** ， **ldterm** 模块将处于标准输入模式。此种情况下，read put 例行程序将在某输入行的结尾逻辑终止输入。标准行终止字符包括 **NEWLINE** 、 **EOF** 、 **EOL** 和 **EOL2** 。如果清除了 **ICANON** ，则 **ldterm** 规则模块将处于非标准模式或原始输入模式。此种情况下，当输入消息缓冲区中至少出现了 **VMIN** 字节，或者 **VTIME** 指定的计时器过期时， read put 例行程序将终止输入（有关详细信息，请参阅 *termio*(7) ）。

M_IOCACK

如果此消息确认了 POSIX **termios TCGETS** 命令，read put 例行程序会将控制台驱动程序下行发送的 **c_cflag** 和速度信息从此消息复制到内部 POSIX **termios** 结构。然后，例行程序将内部 POSIX **termios** 结构复制到此消息。

如果此消息确认了 POSIX **termios** set 命令（即 **TCSETS** 、 **TCSETSW** 和 **TCSETSF** ）中的一个，read put 例行程序会将所有数据从此消息复制到内部 POSIX **termios** 结构。

该处理完成后，read put 例行程序将确定此 I/O 控制命令在被 write service 例行程序转换成 POSIX **termios** 命令之前，最初是否为 BSD 4.3 或 System V I/O 控制命令。如果是，它将会恢复初始数据，以便消息确认初始的 I/O 控制命令。然后，该例行程序将上行转发此消息。

M_CTL 此消息是由驱动程序发送的，用于向 **ldterm** 发出特殊请求。 **M_CTL** 消息的结构与 **M_IOCTL** 消息的结构相同。 **M_CTL** 消息块指向包含 **iocblk** 数据结构（在 **<sys/stream.h>** 中定义）的消息缓冲区。此结构的 **ioc_cmd** 成员包含一个命令，就像在 **M_IOCTL** 消息中一样。 **M_CTL** 消息块的 **b_cont** 成员包含一个指向 **M_DATA** 消息块的指针，而所指向的这个消息块包含与 **M_CTL** 消息关联的数据。

read put 例行程序可处理包含下列命令的 **M_CTL** 消息：

MC_NO_CANON
关闭通常针对上行的 **M_DATA** 消息执行的输入处理。此操作有助于使用执行自身的输入处理的模块或驱动程序，例如处于 **REMOTE** 模式、与执行输入处理的程序相连接的伪终端（请参阅 *ptm*(7) 和 *pts*(7) ）。

MC_DO_CANON
打开通常针对上行的 **M_DATA** 消息执行的输入处理。当驱动程序希望 **ldterm** 退出 **REMOTE** 模式时，将发送此消息。

写入端行为

ldterm 可处理其输出流上的下列 STREAMS 消息。此处未列出的消息只下行转发。

M_FLUSH
write put 例行程序将刷新写入队列，并忽略任何缓冲的输出数据。然后，此例行程序将下行转发消息。

M_DATA write service 例行程序根据 POSIX 1003.1 规范的输出标记处理数据。当输出队列已满，并且处理了所有数据后，此例行程序会将处理的字符下行发送到驱动程序。

M_IOCTL
write put 例行程序将验证 **M_IOCTL** 消息的格式，并检查已知的命令。如果消息格式无效，则此例行程序会将 **M_IOCTL** 消息转换成 **M_IOCNAK** 消息，然后上行返回此消息。如果无法识别 I/O 控制命令，则此例行程序将下行转发 **M_IOCTL** 消息，以便其他模块对其进行处理。

write put 例行程序将确定此命令是否为必须按相对于 **M_DATA** 消息的正确顺序处理的命令。如果是，则此例行程序会将 **M_IOCTL** 消息排入到写入队列中，以便 write service 例行程序以后对其进行处理。需要按顺序处理的命令包括：

TCSETSW 、 **TCSETSF** 、 **TCSETAW** 、 **TCSETAF** 、 **TCSBRK**

否则，模块的 write put 例行程序将立即处理此命令。下面的“ioctl 命令”一节提供了前述的 **ioctl** 命令的详细说明。

M_READ 当应用程序发送了读取请求，但流头部上没有足够的排队数据，因而无法满足此请求时，流头部将发送此消息，以通知下行的模块。通常，当 **ldterm** 在非标准输入模式下操作时，将下行发送

M_READ。如果 **VTIME** 为正值，则 write put 例行程序将启动输入计时器。如果计时器过期，此例行程序将上行发送所有缓冲的输入。然后，再下行转发 **M_READ** 消息。

ioctl 命令

ldterm 模块对以下两种类别的 **ioctl** 命令起作用：

- 主终端 I/O 控制命令
- 与 BSD 4.3 兼容的终端 I/O 控制命令

有关使用这些 **ioctls** 的详细说明，可在 *termio*(7) 联机帮助页上找到。 注释： **ldterm** 目前不支持 *termio*(7) 中介绍的 **FIO[xyz] ioctls** 。

主终端 I/O 控制命令

ldterm 模块对下列主终端 I/O 命令起作用：

TCSETS、 **TCSETSW**、 **TCSETSF**
如果 **ldterm** 模块收到 **M_IOCTL** 消息中包含的上述任何命令，那么它将下行转发这些命令。如果此模块收到读取队列中的 **M_IOCACK** 消息，则它会将 POSIX **termios** 信息从此消息复制到内部 POSIX **termios** 结构，然后上行转发此消息。如果模式更改需要更改流头部的选项，则上行发送 **M_SETOPTS** 消息。如果打开或关闭 **ICANON** flag 标记，则流头部的读取模式将分别更改为打开了读取通知 (**SO_MREADON**) 的 message-nondiscard 模式 (**RMSGN**) 或关闭了读取通知 (**SO_MREADOFF**) 的 byte-stream 模式 (**RNORM**) 。如果打开或关闭 **TOSTOP** 标记，则会分别打开 (**SO_TOSTOP**) 或关闭 (**SO_TONSTOP**) 流头部的 tostop 模式。

TCGETS **ldterm** 模块下行转发 **M_IOCTL** 消息。如果此模块收到读取队列中的 **M_IOCACK** 消息，则它会将 **CLOCAL** 标记和速度从此消息复制到内部 POSIX **termios** 结构。然后，此模块将整个结构复制到 **M_IOCACK** 消息，再上行转发此消息。

TCSETA、 **TCSETAW**、 **TCSETAF**
这些命令可设置旧的 System V **termio** 信息。 **ldterm** 模块将消息转换为 POSIX **termios M_IOCTL** 消息，然后使用对应的 POSIX **termios** 命令（即 **TCSETS** 、 **TCSETSW** 和 **TCSETSF** ）转发此消息。将存储初始的 I/O 控制命令和 **M_IOCTL** 消息，以便用于 **M_IOCACK** 。

TCGETA 此命令可获取旧的 System V **termio** 信息。 **ldterm** 模块将消息转换为 POSIX **termios M_IOCTL** 消息，然后使用 **TCGETS** 命令转发此消息。将存储初始的 I/O 控制命令和 **M_IOCTL** 消息，以便用于 **M_IOCACK** 。当 **ldterm** 模块收到匹配的 **M_IOCACK** 消息时，它会根据 **TCGETS** 命令处理此消息，然后将 POSIX **termios** 信息转换为 System V **termio** 信息并作出答复。

TCSBRK **ldterm** 模块将下行转发此命令，以供驱动程序处理，这样，此驱动程序在上行发送 **M_IOCACK** 消息之前，有机会耗尽数据。

TCXONC 此命令可控制输入/输出流量控制的行为。如果参数为 0，并且尚未停止输出，则下行发送 **M_STOP** 消息。如果参数为 1，并且停止了输出，则下行发送 **M_START** 消息。如果参数为 2，并且尚未停止输入，则下行发送 **M_STOPI** 消息。如果参数为 3，并且停止了输入，则下行发送 **M_STARTI** 消息。

TCFLSH 此命令可刷新输入和（或）输出流。如果参数为 0，则下行发送一个标记字节为 **FLUSHR** 的 **M_FLUSH** 消息。驱动程序将上行逆向反映此 **M_FLUSH**(**FLUSHR**) 消息，以刷新整个输入流。如果参数为 1，则上行发送一个标记字节为 **FLUSHW** 的 **M_FLUSH** 消息。流头部将下行反映此 **M_FLUSH**(**FLUSHW**) 消息，以刷新整个输出流。

TIOCSWINSZ

此命令可设置窗口大小变量。此命令的参数使用指向 **winsize** 结构的指针。 **ldterm** 模块不使用窗口大小变量，但此处维护此变量的目的在于向 **TIOCGWINSZ** 命令提供任何必要的答复。此模块将下行转发消息。

TIOCGWINSZ

ldterm 模块收到此命令后，将返回最后一个 **TIOCSWINSZ** 命令设置的窗口大小变量。此命令的参数使用指向 **winsize** 结构的指针。

EUC_WSET

此命令可设置 EUC 字符集的字符宽度和屏幕宽度。此命令的参数使用指向 **eucioc** 结构的指针，此结构包含用于设置 EUC 字符集的字符宽度和屏幕宽度的信息。处理完命令后， **ldterm** 会将此消息下行转发到下一个模块。

EUC_WGET

此命令将返回 EUC 字符集的字符宽度和屏幕宽度。此命令使用指向 **eucioc** 结构的指针，EUC 字符宽度和屏幕宽度信息将通过此结构返回。

EUC_SET_HP15

此命令可使 **ldterm** 进入所谓的 **HP15** 模式， **ldterm** 可通过此模式识别 HP15_SJIS、HP15_BIG5、HP15_CCDC 和 HP15_GB 字符集，并可处理这些字符集，使处理后的字符集的行为类似于 EUC 字符。此命令的参数使用指向某个整数值的指针，此整数值指定上述四个受支持的 HP15 字符集中的一个。如果此参数设置为 HP15_ASCII，则 **ldterm** 将切换回正常的 ASCII 处理。 **EUC_WSET** 与 **EUC_SET_HP15** 互相排斥。

EUC_GET_HP15

此命令将返回通过 **EUC_SET_HP15** 命令设置的当前 HP15 字符。此命令使用指向某个整数的指针，并通过此整数返回结果。如果以前未发送 **EUC_SET_HP15** ，则此命令将返回 HP15_ASCII。

与 **BSD 4.3** 兼容的终端 **I/O** 命令

ldterm 模块对下列 I/O 命令起作用，这些命令与 BSD I/O 环境兼容：

TIOCEXCL

设置“独占使用”模式。在关闭文件之前，不允许进一步执行打开操作。

TIOCNXCL

关闭“独占使用”模式。

TIOCSETD

ldterm 模块除了回复此命令外，不执行其他操作。在 BSD 系统上，此命令用于设置当前的线路规则类型。在 STREAMS 环境中，它没有太大的意义，原因是可通过从流中取出当前模块，然后将另一个模块压入此流，来更改线路规则模块。

TIOCGETD

在 BSD 系统上，此命令用于获取当前的线路规则类型。在 STREAMS 环境中，此命令没有太大的意义。由于 **ldterm** 支持作业控制，对于二进制兼容性， **ldterm** 模块将使用值 2 进行回复。

TIOCFLUSH

此命令与 **TCFLSH** 命令相似，可刷新输入和（或）输出流。参数是指向 **int** 变量的指针。如果参数值为零，则通过上行和下行发送相应的 **FLUSHR/FLUSHW M_FLUSH** 消息来刷新输入和输出流。否则， **int** 的值将被视为由 **<sys/file.h>** 定义的 **FREAD** 和 **FWRITE** 标记的逻辑 **OR** 运算的结果。如果设置了 **FREAD** 标记，则刷新输入流。如果设置了 **FWRITE** 标记，则刷新输出流。然后， **ldterm** 使用 **M_IOCACK** 确认消息。

TIOCOUTQ

此命令使用指向某个整数的指针，并返回 **ldterm** 的输出缓冲区中缓冲的字符数。

TIOCHPCL

此命令可设置 POSIX **termios HUPCL** 标记，以指示当关闭与终端线路关联的最后一个文件描述符后，应断开此终端线路的连接。 **ldterm** 模块可通过下行发送一个包含 **TCSETS** 命令（带有当前的 **termios** 设置）的 **M_IOCTL** 消息，将此命令转换为兼容的 POSIX **termios** I/O 控制命令。

TIOCSTART

此命令可重新启动输出。如果停止了终端，则 **ldterm** 模块将下行发送 **M_START** 消息。

TIOCSTOP

此命令可停止输出。 **ldterm** 模块将下行发送 **M_STOP** 消息。

TIOCSBRK

此命令可设置线路上的中断条件。 **ldterm** 模块可将包含值 1 的 **M_BREAK** 消息作为数据发送到驱动程序，然后回复 **M_IOCACK** 。

TIOCCBRK

此命令可清除线路上的中断条件。 **ldterm** 模块可将包含值 0（零）的 **M_BREAK** 消息作为数据发送到驱动程序，然后回复 **M_IOCACK** 。

TIOCSETP、 TIOCSETN

这些命令可设置在 **<sys/ttold.h>** 中定义的 **sgttyb** 信息。参数是指向 **sgttyb** 结构的指针。 **ldterm** 模块将消息转换为 POSIX **termios M_IOCTL** 消息。然后，使用对应的 POSIX **termios** 命令（即 **TCSETSW** 和 **TCSETS** ）转发 POSIX **termios M_IOCTL** 消息。将存储初始的 I/O 控制命令和 **M_IOCTL** 消息，以便用于 **M_IOCACK** 消息。

TIOCGETP

此命令可根据 **ldterm** 中维护的 POSIX **termios** 结构的当前内容的解释，返回 **sgttyb** 信息。参数是指向 **sgttyb** 结构的指针，信息是通过此结构返回的。

TIOCSETC

此命令可设置 **<sys/strtio.h>** 中定义的 **tchars** 信息。参数是指向 **tchars** 结构的指针。 **ldterm** 模块将消息转换为 POSIX **termios M_IOCTL** 消息。然后，使用对应的 POSIX **termios** 命令（即 TCSETS ）转发 POSIX **termios M_IOCTL** 消息。将存储初始的 I/O 控制命令和 **M_IOCTL** 消息，以便用于 **M_IOCACK** 。

TIOCGETC

此命令可根据 **ldterm** 中维护的 POSIX **termios** 结构的当前内容的解释，返回 **tchars** 信息。参数是指向 **tchars** 结构的指针，信息是通过此结构返回的。

TIOCSLTC

此命令可设置 **<sys/bsdtty.h>** 中定义的 **ltchars** 信息。 **ldterm** 模块将消息转换为 POSIX **termios M_IOCTL** 消息。然后，使用对应的 POSIX **termios** 命令（即 TCSETS ）转发 POSIX **termios M_IOCTL** 消息。将存储初始的 I/O 控制命令和 **M_IOCTL** 消息，以便用于 **M_IOCACK** 。

TIOCGLTC

ldterm 模块可根据 **ldterm** 中维护的 POSIX **termios** 结构的当前内容的解释，返回 **ltchars** 信息。

TIOCLBIS、 **TIOCLBIC**、 **TIOCLSET**

这些命令可设置 **<sys/strtio.h>** 中定义的 BSD 4.3 标记信息。对于 **TIOCLBIS** 和 **TIOCLBIC** ，参数是指向 **int** 的指针，其值是包含了待设置/清除标记的掩码。对于 **TIOCLSET** ，参数是指向 **int** 的指针，其值是要设置的一组新标记。 **ldterm** 模块将消息转换为 POSIX **termios M_IOCTL** ，然后使用对应的 POSIX **termios** 命令（即 TCSETS ）转发 POSIX **termios M_IOCTL** 消息。此模块将存储初始的 I/O 控制命令和 **M_IOCTL** 消息，以便用于 **M_IOCACK** 。

TIOCLGET

ldterm 模块可根据 **ldterm**
中维护的 POSIX **termios** 结构的当前内容的解释返回 BSD 4.3 标记信息。

TIOCSTI 此命令使用的参数是指向某个字符的指针，并且此命令假定此字符是在终端上键入的。对于发出读写控制的控制终端，用户必须具有 **DEVOPS** 权限或具有读取权限。有关对支持精细划分的权限的系统的授权访问的详细信息，请参阅 *privileges*(5) 。

FIONREAD

此命令使用的参数是指向某个整数的指针，它将返回可立即读取的字符的数目。

作者

ldterm 由 HP 和 OSF 联合开发。

另请参阅

ioctl(2)、 privileges(5)、 ptem(7)、 ptm(7)、 pts(7)、 streamio(7)、 termio(7)。

名称

lp - 行式打印机

概要

#include <sys/lprio.h>

备注

本手册条目仅适用于某些打印机。对于 800 系列，它适用于由设备驱动程序 **lpr2** 控制的打印机。它 不适用于 700 系列系统上的任何打印机。

说明

本节介绍各种版本的 HP-UX 操作系统支持的很多行式打印机提供的功能。行式打印机是一种字符专用设备，它可以有选择地将解释应用于数据。

如果字符专用设备文件是使用原始选项创建的（有关使用原始选项创建设备文件的信息，请参阅 HP-UX 系统管理员手册），则数据将以 “原始模式”发送到打印机（例如，处理图形打印操作时）。在原始模式下，不对要打印的数据进行解释，也不执行页面格式化。只是将数据字节发送到打印机，并完全以接收时的原样进行打印。

如果设备文件不包含原始选项，仍能以原始模式将数据发送到打印机。原始模式由 **LPRSET** 请求进行设置和清除。

如果行式打印机设备文件不包含原始选项，则根据以下讨论的规则对数据进行解释。因为打印机页面具有页面长度（以行数计）、行长度（以字符计）以及距左边距的偏移（以字符计），所以驱动程序了解打印机页面的概念。缺省的行长度、缩进、每页行数、打开和关闭跳页以及退格符的处理，均被设置为在打印机打开并被系统首次识别时确定的缺省值。如果打印机未被识别，则缺省的行长度为 132 个字符，缩进为 4 个字符，每页行数为 66 行，关闭时跳 1 页，打开时不跳页，并且根据字符打印机对退格符进行处理。

下列规则描述了数据流的解释：

- 换页符会导致跳页并将行计数器重置为 0。
- 多个连续换页符将被视为单个换页符。
- 换行符将被映射到回车/换行序列，如果指定了偏移，则在回车/换行序列之后将插入很多空白字符。
- 超过页尾的换行符将被转换为换页符。
- 制表符将被扩展为适当数量的空白字符（按当前缩进值进行偏移时，每 8 个字符位置出现一个制表位）。
- 将退格符解释为生成适用于字符打印机或行式打印机的叠印。
- 大于行长度减去缩进（使用上面的缺省值，即 128 个字符）的行将被截断。
- 回车符会导致行叠印。
- 当打开或关闭时，将生成适当数量的跳页。

可以使用两个 *ioctl*(2) 请求，来控制每页行数、每行字符数、缩进、退格符处理，以及打开和关闭时的跳页数。

打开或关闭时，如果没有请求跳页，则纸张不会移动。打开时，行计数和页面计数将开始采用页面顶部条件。

ioctl 请求具有以下格式：

#include <sys/lprio.h>

int ioctl(int fildes, int request, struct lprio *arg);

request 的可能值包括：

LPRGET 获取当前打印机状态信息，并将其存储在 *arg* 指向的 **lprio** 结构中。

LPRSET 通过 *arg* 指向的结构设置当前打印机状态信息。

LPRGET 和 **LPRSET** 请求中使用的 **lprio** 结构体在 **<sys/lprio.h>** 中定义，并包括下列成员：

```
short int  ind;       /* indent */
short int  col;       /* columns per page */
short int  line;      /* lines per page */
short int  bksp;      /* backspace handling flag */
short int  open_ej;   /* pages to eject on open */
short int  close_ej;  /* pages to eject on close */
short int  raw_mode;  /* raw mode flag */
```

上述内容将在打开时被记住，因此，可以使用外部程序设置缩进、页面宽度和页面长度。如果 **col** 字段设置为 0，则在下一次打开时恢复为缺省值。

如果退格符处理标记为 0，则假定为字符打印机，退格符将通过驱动程序毫无改变地进行传递。如果该标记为 1，则假定为行式打印机，并且会生成足够的打印操作以生成适当的重叠字符。

如果原始模式标记为 0，则将根据缩进、每页列数、每页行数、退格符处理以及打开和关闭时的跳页数，对发送到打印机的数据进行格式化。

如果原始模式标记为 1，将不对发送到打印机的数据进行格式化。

如果原始模式标记从 1 更改为 0（原始模式被关闭），并且未修改格式设置（缩进、每页列数等等），则根据以前的格式设置对数据进行格式化。

作者

lp 由 HP 和 AT&T 联合开发。

文件

/dev/lp	某些 HP-UX 命令使用的缺省或标准打印机；
/dev/[r]lp*	打印机专用文件

另请参阅

lp(1)、slp(1)、ioctl(2)、cent(7)、intro(7)。

名称

lvm - 逻辑卷管理器 (LVM)

说明

逻辑卷管理器 (LVM) 是管理磁盘空间的一个子系统。 HP LVM 子系统提供增值功能，例如镜像功能（使用可选的 HP MirrorDisk/UX 软件）、高可用性（使用可选的 HP ServiceGuard 软件），以及提高可用性和性能的条带化功能。

与早期将磁盘划分为固定大小的几个区域的排列方式不同， LVM 允许用户将磁盘（也称为 “物理卷”）视为由相同大小盘区组成的数据存储池（或卷）。一个盘区的缺省大小为 4 MB。

LVM 系统由物理卷的任意分组组成，其组织形式为 “卷组”。一个卷组包括一个或多个物理卷。系统中可以有多个卷组。创建卷组后，卷组（而不是磁盘）成为数据存储的基本单元。因此，早期是用户将磁盘从一个系统移到另一个系统，但对于 LVM 而言，是用户将卷组从一个系统移到另一个系统。鉴于此原因，系统中存在多个卷组通常比较方便。

卷组可细分为虚拟磁盘，称为 “逻辑卷”。一个逻辑卷可以跨越多个物理卷，或仅代表一个物理卷的一部分。一个卷组所代表的磁盘空间池可分为几个不同大小的逻辑卷。逻辑卷的大小由其盘区数确定。创建逻辑卷后，可将逻辑卷视为磁盘分区处理。可以将逻辑卷分配给文件系统，用作交换设备或转储设备，或者用于原始访问。

命令

可以使用下列命令创建、显示和处理 LVM 信息：

lvchange	更改逻辑卷特性
lvcreate	在卷组中创建条带化的逻辑卷
lvdisplay	显示有关逻辑卷的信息
lvextend	为逻辑卷增加空间、增加镜像
lvlnboot	准备逻辑卷使之成为根卷、主要交换卷或转储卷
lvreduce	减小分配给逻辑卷的物理盘区数
lvremove	从卷组中删除一个或多个逻辑卷
lvrmboot	删除指向根卷、主要交换卷或转储卷的逻辑卷链接
pvchange	更改卷组中物理卷的特性
pvcreate	创建物理卷以便在卷组中使用
pvdisplay	显示有关卷组中物理卷的信息
pvmove	将分配的物理盘区从一个物理卷移到其他物理卷
vgcfgbackup	创建或更新卷组配置备份文件
vgcfgrestore	通过备份文件显示或还原卷组配置
vgchange	设置卷组可用性
vgcreate	创建卷组
vgdisplay	显示有关卷组的信息
vgexport	导出卷组及其相关逻辑卷
vgextend	通过添加物理卷扩展卷组

vgimport	将卷组导入系统
vgmodify	修改卷组属性
vgreduce	从卷组中删除物理卷
vgremove	从系统中删除卷组定义
vgscan	扫描卷组的物理卷

如果安装了 HP MirrorDisk/UX 软件，还可以使用下列命令：

lvmerge	将两个逻辑卷合并为一个逻辑卷
lvsplit	将镜像的逻辑卷分为两个逻辑卷
lvsync	同步逻辑卷中的过时镜像
vgsync	同步卷组中的过时逻辑卷镜像

设备专用文件

在此版本的 HP-UX 11i 中，针对用于标识设备的设备专用文件，海量存储堆栈支持两种命名约定（请参阅 *intro*(7) ）。可以使用以下形式表示设备：

- 持久性设备专用文件 (**/dev/disk/disk3**) ，或者
- Legacy 设备专用文件名 (**/dev/dsk/c0t6d6**) 。

尽管 LVM 支持在同一卷组中使用这两种约定，但是，LVM 联机帮助页中显示的示例均使用 Legacy 设备专用文件约定。

备用链接 **(PVLink)**

在此版本的 HP-UX 中，LVM 继续支持设备的备用链接，以便在主链接失败的情况下，可以继续访问该设备。这种多链接或多路径解决方案提高了数据可用性，但是它仍然不允许同时使用多个路径。

HP-UX 11i V3 上的海量存储子系统中引入了一项新功能，该功能支持设备的多个路径，并且允许同时访问多个路径。海量存储子系统将平衡各有效路径之间的 I/O 负载。多路径行为是缺省设置，除非使用 **scsimgr** 命令启用旧的多路径行为，并且活动路径为旧的设备专用文件。有关详细信息，请参阅 *scsimgr*(1M) 。

即使海量存储子系统在此版本的 HP-UX 上对每个物理卷可支持 32 个路径， LVM 不支持任一物理卷具有 8 个以上的路径。因此，诸如 **vgcreate** 和 **vgextend** 的命令为每个物理卷添加 8 个以上的路径时将不会成功。此外，**vgimport** 和 **vgscan** 无法在 **/etc/lvmtab** 文件中为每个物理卷写入 8 个以上的路径。如果用户要使用此 8 个路径以外的任何特定路径，则必须对卷组中的其中一个备用路径执行 **vgreduce** 操作，并使用 **vgextend** 添加此特定路径。

不再需要（或者不建议）为 LVM 配置备用链接。但是，可以保留传统的 LVM 行为。要实现此目的，必须同时满足下面两个条件：

- 只在卷组配置中使用 Legacy 设备专用文件的命名约定。
- **scsimgr** 命令用于启用卷组中各物理卷的旧的多路径行为。

举例

开始使用 LVM 的基本步骤如下:

- 标识要用于 LVM 的磁盘。
- 在每个已标识磁盘上创建 LVM 数据结构（请参阅 *pvcreate*(1M)）。
- 收集所有物理卷以形成新卷组（请参阅 *vgcreate*(1M)）。
- 利用卷组中的空间创建逻辑卷（请参阅 *lvcreate*(1M)）。
- 将每个逻辑卷作为磁盘分区使用（创建文件系统或用于原始访问）。

要将磁盘 **/dev/dsk/c0t0d0** 配置为名为 **vg01** 的新卷组的一部分，请执行下列操作:

首先，使用 **pvcreate** 命令为 LVM 初始化磁盘。

pvcreate /dev/rdsk/c0t0d0

然后，创建 LVM 子系统使用的伪设备文件。

mkdir /dev/vg01
mknod /dev/vg01/group c 64 0x010000

group 文件的次设备号对于系统中的所有卷组而言应该是唯一的。其格式为 **0x***NN***0000** ，其中 *NN* 的范围从 **00** 到 **ff** 。

使用 **vgcreate** 命令创建包含物理卷 **/dev/dsk/c0t0d0** 的卷组 **vg01** 。

vgcreate /dev/vg01 /dev/dsk/c0t0d0

可以使用 **vgdisplay** 命令查看有关新创建的卷组的信息。

vgdisplay -v /dev/vg01

使用 **lvcreate** 命令在该卷组中创建大小为 100 MB、名为 **usrvol** 的逻辑卷。

lvcreate -L 100 -n usrvol /dev/vg01

这将为逻辑卷创建两个设备文件，一个是块设备文件 **/dev/vg01/usrvol** ，另一个是字符（原始）设备文件 **/dev/vg01/rusrvol** 。

可以使用 **lvdisplay** 命令查看有关新创建的逻辑卷的信息。

lvdisplay /dev/vg01/lvol1

允许对磁盘分区执行的任何操作，也允许对逻辑卷执行。因此，可以使用 **usrvol** 保存文件系统。

newfs /dev/vg01/rusrvol
mount /dev/vg01/usrvol /usr

另请参阅

lvchange(1M)、 lvcreate(1M)、 lvdisplay(1M)、 lvextend(1M)、 lvlnboot(1M)、 lvreduce(1M)、 lvremove(1M)、 lvrmboot(1M) 、 pvchange(1M) 、 pvcreate(1M) 、 pvdisplay(1M) 、 pvmove(1M) 、 vgcfgbackup(1M) 、 vgcfgrestore(1M) 、 vgchange(1M) 、 vgcreate(1M) 、 vgdisplay(1M) 、 vgexport(1M) 、 vgextend(1M) 、 vgimport(1M)、 vgmodify(1M)、 vgreduce(1M)、 vgremove(1M)、 vgscan(1M)、 intro(7)。

《管理系统和工作组》。

如果安装了 HP MirrorDisk/UX： lvmerge(1M)、 lvsplit(1M)、 lvsync(1M)、 vgsync(1M)。

如果安装了 HP ServiceGuard： cmcheckconf(1M)、 cmquerycl(1M)、《管理 MC/Serviceguard》。

名称

mem - 主内存映像文件

说明

mem 是计算机主内存映像的专用文件。它可用于，例如，检查和修补系统。

mem 中的字节地址被解释为物理内存地址。试图引用不存在的位置会导致返回错误。

文件 **kmem** 与 **mem** 相同，只是被访问的是内核虚拟内存而不是物理内存。有关 **/dev/kmem** 支持的 ioctl 操作的信息，请参考 *kmem*(7) 。

警告

当出现只读位或只写位时，检查与修补设备寄存器可能会导致意外结果。

文件

/dev/mem

/dev/kmem

另请参阅

kmem(7)。

名称

modem - 异步串行调制解调器线路控制

概要

#include <sys/modem.h>

说明

本节介绍两种模式的调制解调器线路控制和三种类型的终端端口访问。此外还讨论了影响调制解调器线路控制的 *termio* 结构位的作用。在本联机帮助页的结尾讨论了与调制解调器相关的 **ioctl()** 系统调用（请参阅 *ioctl*(2) ）。

定义

下面的讨论中使用了若干个术语，在此定义这些术语以供参考。

“调制解调器控制线” (CONTROL) 通常定义为由驱动程序自动控制的调制解调器传出线路。

“调制解调器状态线” (STATUS) 通常定义为由驱动程序自动监视的调制解调器传入线路。

终端文件的 CONTROL 和 STATUS 随文件的调制解调器线路控制模式的不同而不同（请参阅下面的“调制解调器线路控制模式”一节）。

如果某一端口的 **open()** （请参阅 *open*(2) ）等待同一端口上的另一文件关闭，则将其视为阻塞。

如果某一端口的 **open()** 等待提升 STATUS，则将其视为未决。

如果 **open()** 系统调用返回到调用进程，且未发生错误，则将端口的 **open()** 视为成功。

Open 标记位

当前，驱动程序可识别的 **open()** 标记位只有 **O_NDELAY** 位和 **O_NONBLOCK** 位。如果设置了其中任意一个标记位，对驱动程序执行 **open()** 调用永远不会被阻塞。如果可能， **open()** 会立即以 SUCCESSFUL 的形式返回，同时，驱动程序将继续打开 tty 文件的进程。如果不可能，则 **open()** 调用会立即返回，同时返回相应章节中介绍的相应的错误代码。

Termio 位

设置了 *termios* 或 *termio* 结构（请参阅 *termio*(7) ）中的 **CLOCAL** 位后，可以使用此位取消驱动程序对调制解调器线路的自动监视。但是，用户是否能够控制调制解调器线路只能由正在实行的模式确定，而不是取决于 **CLOCAL** 的状态。通常，驱动程序会监视 STATUS，并要求提升它。除非设置了 **CLOCAL** 位，否则 **open()** 系统调用将提升 CONTROL，并且在完成之前等待 STATUS （如果设置了 **O_NDELAY** 或 **O_NONBLOCK** 位， **open()** 将立即返回，但是驱动程序会根据 **CLOCAL** 位的状态，以正常方式继续监视调制解调器线路）。通常，丢失 STATUS 将导致驱动程序中断调制解调器连接，并降低 CONTROL。但是，如果设置了 **CLOCAL** ， STATUS 发生的任何更改都将被忽略。除非设置了 **CLOCAL** ，否则在读取或写入任何数据之前，需要建立连接。在设置 **CLOCAL** 时，通常情况下正在使用的任何计时器（请参阅下面的“调制解调器线路控制模式”一节和“调制解调器计时器”一节）将停止。

如果 **CLOCAL** 位由清除更改为设置，则驱动程序将假定端口上存在活动设备（例如，调制解调器），而无论 STATUS 如何。如果在那一时刻提升了任何 CONTROL，则它们继续保持此状态。不再主动监视 STATUS。如果 **CLOCAL** 位由设置更改为清除，则驱动程序恢复对 STATUS 的主动监视。如果在那一时刻提升了所有

CONTROL 和 STATUS，则驱动程序将继续保持调制解调器连接。如果没有提升任何 STATUS，则驱动程序的行为就如同丢失了这些信号（如下面“调制解调器线路控制模式”一节所述），如果设备为控制终端，则向控制进程发送一个 *hangup* 信号。如果没有提升任何 CONTROL，则驱动程序将通过降低所有 CONTROL 来中断调制解调器连接。

如果向终端文件发出了最后一个 **close()** 系统调用（请参阅 *close*(2)），*termios* 或 *termio* 结构中的 **HUPCL** 位将确定驱动程序在有关 CONTROL 方面的操作。如果设置了 HUPCL 位，驱动程序将在执行 **close()** 时降低 CONTROL，并且调制解调器连接将会中断。如果未设置 **HUPCL**，并且存在调制解调器连接，那么此调制解调器连接会继续存在，即使在发出 **close()** 后也是如此。驱动程序将不更改 CONTROL。

终端端口访问类型

调制解调器访问有三种类型：呼入连接、呼出连接和直接（无调制解调器控制）连接。通过访问不同的文件，使用所有三种连接类型可以访问给定的端口。终端文件的调制解调器访问类型由此文件的主设备号和（或）次设备号确定。

如果需要通过呼入建立连接，则将使用呼入访问类型。*getty*(1M) 将使用此类型，来接受通过调制解调器完成的登录。向这样的文件发出 **open()** 后，驱动程序会等待呼入，然后根据端口的当前模式（请参阅下文）提升 CONTROL。如果端口已关闭，则驱动程序可能会降低 CONTROL，也可能不会，具体取决于 **HUPCL** 位。

如果需要通过呼出建立连接，则将使用呼出访问类型。*uucp*(1) 等程序将使用此访问类型。向这样的文件发出 **open()** 后，驱动程序将立即提升 CONTROL，然后根据当前使用的模式等待连接。如果端口已关闭，则驱动程序可能会降低 CONTROL，也可能不会，具体取决于 **HUPCL** 位。

如果不需要驱动程序调制解调器控制，则将使用直接访问类型。此类型可用于使用三线连接的直接连接终端，或者在建立连接之前，用于与调制解调器通信。对于第二种情况，程序可以向调制解调器发出拨号指令。 **CLOCAL** 和 **HUPCL** 位对通过直接文件访问的端口没有任何影响（但是，其他类型的文件可以继承这两个位；请参阅下面的“终端端口访问联锁”一节）。对直接文件执行 **open()** 不影响 CONTROL，并且其成功完成不依赖于 STATUS 的任何特定状态。如果文件已关闭，则驱动程序不影响 CONTROL 的状态。如果已建立调制解调器连接，则此连接会继续存在。将直接文件的速度设置为 B0（请参阅 *termio*(7)）将被视为不可能发生的速度更改，因而将被忽略。此设置不影响 CONTROL。

调制解调器线路控制模式

调制解调器线路控制有两种模式： CCITT 模式和简单模式。在任何给定的时间点上，给定端口只能使用这两种模式中的一种模式。尝试以所使用模式之外的模式打开端口（对不同的文件执行 PENDING 或 SUCCESSFUL **open()** 调用），将导致 **open()** 返回 [ENXIO] 错误。终端文件的调制解调器访问类型由此文件的主设备号和（或）次设备号确定。

CCITT 模式用于连接交换线路调制解调器。 CCITT 模式的 CONTROL 为数据终端就绪 (DTR) 和请求发送 (RTS)。 STATUS 为数据设置就绪 (DSR)、数据载波检测 (DCD) 和清除发送 (CTS)。另外，振铃指示 (RI) 信号还可指示是否存在呼入。如果开始了连接（为呼入文件执行了呼入，或者向呼出文件发出了 **open()**），将提升 CONTROL，并启动连接计时器（请参阅下面的“调制解调器计时器”一节）。如果在超过计时时间之前提升了 STATUS，将建立连接，并且 **open()** 请求将成功返回。如果计时时间已过，则降低 CONTROL，并且连接将异常中止。对于呼入文件，驱动程序将等待另一个呼入；对于呼出文件， **open()** 将返回 [EIO] 错误。一旦建立连接，

无论丢失 DSR 还是 CTS，都会导致 CONTROL 降低，并且，如果设备为控制终端，还会向控制进程发送一个 *hangup* 信号。

如果丢失了 DCD，则将启动计时器。如果在超过计时时间之前， DCD 已恢复，则会保留连接。但是，在这段时间内不传输任何数据。驱动程序将停止传输字符，并且驱动程序收到的任何字符将被忽略（但是，对于某些实现，将无法停止数据传输。请参阅“相关内容”一节）。如果在分配的时间内 DCD 未恢复，则连接会像上面的 DSR 和 CTS 介绍中所述的那样发生中断。

如果在发出 **close()** 系统调用（即设置了 **HUPCL** ）后，需要中断调制解调器连接，则会降低 CONTROL，并且 **close()** 将成功返回。但是，只有在调制解调器降低了 DSR 和 CTS 并且挂起计时器（请参阅下面的“调制解调器计时器”一节）过期之后，才允许进一步执行 **open()** 调用。在这段时间内，为响应 **open()** 而执行的操作就如同端口仍然是打开的（请参阅下面的“终端端口访问联锁”一节）。

如果端口处于 CCITT 模式，则驱动程序对调制解调器线路具有完全控制权限，并且不允许用户更改 CONTROL 的设置，或者影响驱动程序主动监视的那些 STATUS （请参阅下面的“调制解调器读写控制”一节）。这是为了严格遵守 CCITT 的建议。

简单模式用于连接只需要简单的调制解调器线路控制方法的设备。这可能包括黑匣子、数据交换机等设备，或者用于系统间连接的设备。此外，此模式还可用于无法在 CCITT 建议条件下工作的调制解调器。简单模式的 CONTROL 只包括 DTR。 STATUS 只包括 DCD。如果发出了 **open()** ，则会提升 CONTROL，但是不启动连接计时器。如果提升了 STATUS，将建立连接，并且 **open()** 请求将以 SUCCESSFUL 的形式返回。一旦建立了连接，丢失 STATUS 将导致 CONTROL 降低，并且，如果设备为控制终端，还会向控制进程发送一个 *hangup* 信号。

如果某个端口处于简单模式，则驱动程序通常会控制调制解调器线路。但是，允许用户更改 CONTROL 的设置（请参阅下面的“调制解调器读写控制”一节）。

终端端口访问联锁

在终端文件的三种访问类型之间提供了一个联锁机制。此机制可防止同时成功打开多个文件，但是它允许某些 **open()** 成功，而使其他 open 调用成为 PENDING，以便当 *getty* 在呼入连接上有未决的 **open()** 时，可以通过呼出连接打开端口。为这三种访问类型指定了优先级，如果针对多个文件发出了 **open()** ，此优先级可确定哪个 **open()** 将会成功。这三种访问类型将按以下方式从最低优先级到最高优先级进行排列：呼入、呼出、直接。

如果向一个已有 SUCCESSFUL **open()** 的端口发出了优先级类型较低的 **open()** ，则新的 **open()** 将返回 [EBUSY] 错误（如果当前正在接收呼入指示，那么尝试对 CCITT 呼出文件调用 **open()** 也会返回 [EBUSY] 。这种情况下，如果对应的 CCITT 呼入文件有一个 PENDING **open()** ，则会完成此 PENDING **open()** ）。如果较低优先级的 **open()** 为 PENDING，那么在可能的情况下，新的 **open()** 将会成功；如果需要等待 STATUS，它将会保留为 PENDING，并且较低优先级的 **open()** 将成为 BLOCKED。如果较高优先级的 **open()** 已成功或者为 PENDING，则新的 **open()** 将成为 BLOCKED，除非新的 **open()** 的 **O_NDELAY** 标记位已被设置，这种情况下， **open()** 将返回 [EBUSY] 错误。如果对一种类型的文件执行的 **open()** 为 SUCCESSFUL，则对较低优先级文件执行的任何 PENDING *open* 将成为 BLOCKED。

如果关闭了某一优先级的文件，则对下一个较低优先级类型的文件执行的 BLOCKED **open()** 将成为活动状态。如果提升了所有 STATUS，则 **open()** 将成为 SUCCESSFUL，否则， **open()** 将成为 PENDING，需要等待 STATUS。如果较低优先级的 **open()** 为 SUCCESSFUL （因为关闭较高优先级文件时保留了连接），较低优先级

文件将继承由较高优先级文件设置的端口特性（速度、奇偶较验等）。如果没有通过 **close()** 保留连接，则端口特性将设置为缺省值。

调制解调器计时器

当前定义了四种可用于调制解调器连接的计时器。前面三种只适用于 CCITT 模式连接。通常，当计时器运行时，更改计时器值所产生的影响是与系统相关的。但是，如果将计时器值设置为零，则一定会禁用计时器，即使此计时器正在运行也是如此。

连接计时器用于限制从开始建立连接起可以等待的时间。当呼入文件收到一个呼入时，或者呼出文件（所涉及的 *open* 均未处于未决状态）已发出 **open()** 时，此计时器将启动。如果连接按时完成，此计时器将异常中止。如果计时时间已过，则连接将异常中止。对于呼入文件，驱动程序将再次等待一个呼入，并且 **open()** 将保留未决状态。对于呼出文件， **open()** 将返回 [EIO] 错误。

载波检测计时器用于限制当 DCD 断开时，使连接断开之前可以等待的时间。如果在这段时间内未重新建立载波，则会发生连接断开。如果在超时之前重新建立了载波，则计时器将异常中止，同时连接得以保留。如果在这段时间内没有提升载波，则不通过线路传输任何数据。

无活动计时器用于限制当线路上没有数据传输时，连接可以保留打开状态的时间。当数据线路由于没有数据传输而进入无提示状态时，将启动此计时器。如果在到达时间限制之前，无论以哪种方向再次通过线路传输了数据，则计时器将异常中止。如果在超时之前出现无活动状况，则驱动程序将断开线路的连接。当数据传输正常或异常停止时，可通过此方式避免长时间、高成本的电话连接。

所定义的最后一种计时器为挂起计时器，它既可用于 CCITT 模式，也可用于简单模式。此计时器可控制在断开调制解调器线路连接之后、允许另一个 **open()** 之前可以等待的时间。应将此时间段设置得足够长，以确保电话交换设备终止连接。如果此时间段不够长，则电话连接可能不会中断，因此后续的 **open()** 可以使用原先的连接完成操作。

HP-UX 调制解调器读写控制

有多个 **ioctl()** 系统调用可用于调制解调器线路的操作。这些系统调用使用 **<sys/modem.h>** 中定义的以下信息：

```
#define NMTIMER 6
typedef unsigned long mflag;
struct mtimer {
    unsigned short m_timers[NMTIMER];
    };
```

mflag long 的每个位对应于下列调制解调器线路之一：

MRTS	请求发送	出站
MCTS	清除发送	入站
MDSR	数据设置就绪	入站
MDCD	数据载波检测	入站
MDTR	数据终端就绪	出站

MRI	振铃指示	入站
MDRS	数据速率选择	出站

计时器值在数组 **m_timers** 中定义。计时器的相对位置、每个计时器的缺省初始值及单位如下：

0	**MTCONNECT**	25 秒
1	**MTCARRIER**	400 毫秒
2	**MTNOACTIVITY**	0 分钟
3	**MTHANGUP**	250 毫秒
4	保留	
5	保留	

如果任何计时器使用零值，则将禁用此计时器。

调制解调器线路 **ioctl()** 系统调用的格式为：

int ioctl(int *fildes***, int** *command***, mflag ****arg***);**

使用此格式的命令包括：

MCGETA 获取入站和出站调制解调器线路的当前状态，并将此状态存储在 **arg** 引用的 **mflag** long 中。某个一位将在适当的位置指示提高的线路。

MCSETA 从 **arg** 引用的 *mflag* long 设置出站调制解调器线路。将出站位设置为一将导致提升此线路，设置为零将降低此线路。设置入站线路位将不起作用。当处于 CCITT 模式时，设置任何位都不起作用。对调制解调器线路的更改可立即生效，当字符仍在输出时使用此格式可能会导致不可预测的结果。

MCSETAW 等待输出完成，按上述方式设置新的参数。

MCSETAF 等待输出完成，然后刷新输入队列，再按上述方式设置新的参数。

计时器值 **ioctl()** 系统调用的格式为：

int ioctl(int *fildes***, int** *command***, mtimer ****arg***);**

使用此格式的命令包括：

MCGETT 获取当前的计时器值设置，并将这些设置存储在 **arg** 引用的 *mtimer* 结构中。

MCSETT 从 **arg** 引用的结构设置计时器值。

对于任何计时器，将计时器值设置为以前的值都不起作用。

SVID3 调制解调器读写控制

System V Interface Definition 第三版 (SVID3) 指定了可操作调制解调器线路的其他 **ioctl()** 系统调用。这些系统调用使用 **<termios.h>** 中定义的信息。

每个 **ioctl()** 都可传递一个整数参数，在此整数参数中，以下每个位定义都将按如下方式对应于调制解调器线路之

一:

TIOCM_RTS	请求发送	出站
TIOCM_CTS	清除发送	入站
TIOCM_DSR	数据设置就绪	入站
TIOCM_CAR	数据载波检测	入站
TIOCM_DTR	数据终端就绪	出站
TIOCM_RNG	振铃指示	入站

此外， **TIOCM_CD** 等效于 **TIOCM_CAR** ， **TIOCM_RI** 等效于 **TIOCM_RNG** 。

调制解调器线路 **ioctl()** 系统调用的格式为:

int ioctl(int *fildes***, int** *command***, int ****arg***);**

使用此格式的命令包括:

TIOCMGET 获取入站和出站调制解调器线路的当前状态，并将此状态存储在 **arg** 引用的 int 中。将在适当的位置使用一个位指示提升的线路。

TIOCMSET 从 **arg** 引用的 int 设置出站调制解调器线路。

TIOCMBIS 提升由 **arg** 引用的 int 的对应位位置中的一指定的控制线路。

TIOCMBIC 降低由 **arg** 引用的 int 的对应位位置中的一指定的控制线路。

请注意，入站线路的设置位不起作用，并且在 CCITT 模式下设置任何位都不起作用。此外，对调制解调器线路的更改可立即生效，当字符仍在输出时使用这些读写控制可能会导致不可预测的结果。

警告

有时，在与收到呼入大致相同的时刻，进程可以打开一个呼出文件。在某些情况下，呼入可以满足呼出连接的条件。但一般来说，这些结果是不确定的。如有必要，可以使用两套调制解调器和端口，其中一套用于呼出连接，另一套用于接收呼入，来避免上述情况。

相关内容

某些硬件实现可能无法访问 MCSETA 支持的所有调制解调器线路。如果特定硬件不支持给定线路，那么设置线路值的尝试将被忽略，并且读取此线路的当前状态将返回零。要确定已安装硬件所支持的线路，应参考相应的 I/O 卡手册。

某些硬件实现可能无法访问 MCSETT 支持的所有计时器。同时，各个计时器的粒度将根据所使用的硬件和系统的不同而异。特定系统应记录设置一个超出范围的计时器，或者设置一个粒度超出此系统功能的计时器会产生的后果。当计时器正在运行时，更改此计时器的值所产生的后果与系统相关，每个系统应记录这些后果。

当计时器正在运行时，设置 **CLOCAL** 位将导致此计时器停止。是否需要重新启动计时器，如果需要重新启动，那么当以后清除了 **CLOCAL** 位后，重新启动计时器需要使用什么值，这些内容都与系统相关。

对于那些支持 HP27140A 6 信道多路复用器的实现，在丢失 DCD 期间将无法停止字符传输。在连接中断之前，驱动程序无法检测到 DCD 丢失。此外，I/O 卡的内部缓冲区中可能还有字符，并且它还会尝试传输这些字符。

作者

modem 由 HP 和 AT&T 联合开发。

文件

/dev/cua*
/dev/cul*
/dev/tty*
/dev/ttyd*

另请参阅

stty(1)、 mknod(1M)、 ioctl(2)、 open(2)、 termio(7)。

名称

mt - 用于 stape 和 estape 的磁带接口和控制

说明

本条目介绍 HP 磁带接口和控制的行为。文件 **/dev/rtape/*** 指的是 estape 驱动程序控制的特定原始磁带机。将动态分配这些设备专用文件的主设备号，且次设备号不会编码任何设备特定信息。

文件 **/dev/rmt/*** 指得是旧的 stape 驱动程序控制的特定原始磁带机，每个给定单元的行为在 DSF 的主设备号和次设备号中指定。不推荐使用旧的驱动程序和 DSF，并且将从 HP-UX 的将来版本中删除。

命名约定

estape 驱动程序的设备专用文件（称为 DSF）具有以下命令约定：

/dev/rtape/tape#_BEST[**n**][**b**]

有四个此类文件（称为持久性 DSF）分别与 **n** 选项和 **b** 选项的四种不同排列相对应。它们受 **estape** 驱动程序支持。有关持久性设备专用文件名的详细信息，请参阅 *intro*(7) 。

Legacy DSF 有两种命名约定。在支持长文件名的系统上使用标准（首选）约定。备用约定是为只能使用短文件名的系统而提供的。建议使用以下标准约定，因为它允许设备名中所有可能的配置选项，且由 *mksf*(1M) 和 *insf*(1M) 使用：

/dev/rmt/c#t#d#[**o**][**z**][**e**][**p**][**s**[#]][**w**]**BEST**[**C**[#]][**n**][**b**]

提供以下备用命名约定是为了支持其中的 **/dev/rmt** 目录需要短文件名的系统。这些 DSF 名称的描述性较差，但保证唯一的设备命名，并由 *mksf*(1M) 和 *insf*(1M) 根据需要使用。

/dev/rmt/c#t#d#[**f#i#**][**n**][**b**]

对于现有的每种磁带设备，安装系统后将自动创建 12 个 DSF。如果通过 **rmsf** 中的 **-L** 选项禁用了 Legacy 模式，安装后将在 **/dev/rtape** 中仅创建 4 个 DSF。它们受 **estape** 驱动程序支持。

将使用以下命名规则在 **/dev/rmt** 目录中创建 Legacy DSF。这些 Legacy DSF 受 **stape** 驱动程序支持。

/dev/rmt/c#t#d#BEST[**n**][**b**].

安装系统之后，将使用 HP-UX 10.0 之前的设备文件命名约定自动创建四个以上的 Legacy DSF，其格式为 **/dev/rmt/#m**[**n**][**b**] 。这包括用于区分此磁带设备与系统中其他磁带设备的任意编号，后跟字母 **m** 。之所以存在四个这样的 DSF，是因为创建了 **n** 和 **b** 选项的四种不同排列中的每一种（见下文）。创建这些文件是为了与 HP-UX 10.0 之前的脚本兼容，供熟悉较早约定的用户使用。

自动创建的每个 DSF 均使用标准或备用命名约定，每个 DSF 均链接到使用 HP-UX 10.0 之前的命名约定的设备文件。也就是说，格式为 **/dev/rmt/#m**[**n**][**b**] 的 DSF 均作为上述 **/dev/rmt/c#t#d#BEST**[**n**][**b**] DSF 对应的硬链接而创建。

因此，使用 HP-UX 10.0 之前命名约定的 DSF 提供的功能与包含密度规范 **BEST** （标准命名约定）的设备文件所提供的功能相同。

选项

此处介绍的选项是所有旧磁带驱动程序的公共选项。 Legacy DSF 中的 **c#t#d#** 表示法派生自 **ioscan** 输出，在 *ioscan*(1M) 和 *intro*(7) 的联机帮助页上进行介绍。

c# 由操作系统为接口卡分配的实例编号。

t# 远程总线上的目标地址（例如 SCSI 地址）

d# 目标地址处的设备单元号（例如 SCSI LUN）。

w 在返回状态之前写入等待操作的物理完成。缺省行为（缓冲模式或即时报告模式）要求磁带设备缓冲数据并以成功状态立即返回。

density 将数据写入磁带时使用的密度或格式。此字段由下列值指定:

BEST 将使用最高容量的密度或格式，如果设备支持压缩，则包括数据压缩。

NOMOD 维护用于以前写入磁带的数据的密度。使用此选项时的行为取决于设备的类型。仅在 DDS 驱动器上支持此选项。

DDS 选择已知的 DDS 格式之一；可以根据需要用于指定 **DDS1** 或 **DDS2** 。

DLT 选择已知的 DLT 格式之一；可以根据需要用于指定 **DLT42500_24** 、 **DLT42500_56** 、 **DLT62500_64** 、 **DLT81633_64** 或 **DLT85937_52** 。

D[*#*] 将密度指定为要在 SCSI 模式选择块描述符中放置的数值。头文件 **<sys/mtio.h>** 包含标准密度代码的列表。此数值仅用于在此列表中找 不到的密度代码。

C[*#*] 在支持数据压缩的磁带机上，以压缩模式写入数据。如果包括了一个数值，则使用它指定特定于设备的压缩算法。请注意，将密度字段设置为 **BEST** 时，也提供压缩。

n 在关闭时不倒带。除非请求此模式，否则关闭时将自动倒带。

b 指定 Berkeley 样式的磁带行为。如果 **b** 不存在，则磁带机遵循 AT&T 样式的行为。在下面的“磁带行为特性”中有详细介绍。

f# 指定在次设备号中编码的格式（或密度）值。此值的含义取决于所用磁带设备的类型（仅用于短文件名表示法）。

i# 指定由磁带驱动程序维护的内部属性表索引值，包含配置选项的数组。此表的内容不可直接进行访问。使用 *lssf*(1M) 命令可确定调用了哪些配置选项（仅用于短文件名表示法）。

o 禁用控制台消息。请参阅 *mksf*(1M) 。

z 关闭兼容的 RTE。请参阅 *mksf*(1M) 。

e 详尽模式。请参阅“相关内容”一节。

p 磁带分区。请参阅“相关内容”一节。

s 固定块模式。请参阅“相关内容”一节。

#m 适用于-HP-UX 10.x 之前的设备文件命名约定。

磁带设备专用文件名示例

对于位于实例 1、目标 2、 LUN 3 的 HP Ultrium-2 驱动器， Legacy DSF 将为 **/dev/rmt/c1t2d3BEST**[**n**][**b**] 。对应的持久性 DSF **/dev/rtape/tape1_BEST**[**n**][**b**] 将假定向该 DSF 分配实例编号“1”。符合 HP-UX 10.0 之前的命名约定的对应设备专用文件将为 **/dev/rmt/0m**[**n**][**b**] 。在此特例中， **0m**[**n**][**b**] 中的 0（零）表示向 DSF 分配的实例编号为 0（零）。格式为 **/dev/rmt/#m**[**n**][**b**] 的文件将作为对应的 **/dev/rmt/c#t#d#BEST**[**n**][**b**] DSF 的硬链接而创建。

使用 *lssf*(1M) 命令可确定实际上与任何设备文件一起使用的配置选项。上面定义的命名约定应该指示所使用的选项，但设备文件可以使用任何用户定义的名称创建。

磁带行为特性

在打开磁带以读取或写入时，假定它是根据需要定位的。

如果关闭为写入而打开的文件，则当且仅当发生对文件的一次或多次写入时，才写入两个连续的 EOF （文件结束）标志。除非指定了不倒带模式，否则将对磁带进行倒带，在这种情况下，会将磁带定位在刚写入的第二个 EOF 之前。

当关闭为读取（仅限读取）而打开的文件，且未设置不倒带位时，将对磁带进行倒带。如果设置了不倒带位，则行为取决于 *style* 模式。对于 AT&T 样式的设备，将磁带定位在刚读取的数据后面的 EOF 之后（除非已在 BOT 或文件标记处）。对于 Berkeley 样式的设备，不以任何方式重新定位磁带。

每个 *read*(2) 或 *write*(2) 调用都读取或写入磁带上的下一条记录。对于写入，记录具有与给定的缓冲区相同的长度（在硬件限制内）。

在读取期间，记录大小作为读取的字节数传递回来，最大为指定的缓冲区大小。由于磁带设备上的最小读取长度为一条完整记录（至下一条记录的标记），因此通过 *ioctl*(2) 的 **MTIOCGET** 调用，已忽略的字节数（对于长度超过指定缓冲区大小的记录）在 **mtget** 结构的 **mt_resid** 字段中是可用的。当前限制要求磁带设备应用程序将 2 字节对齐方式用于缓冲区位置和 I/O 大小。要允许将来更苛刻的限制（4 字节对齐等），并最大限度地提高性能，建议使用页面对齐。例如，如果结构内包含目标缓冲区，则必须注意缓冲区之前的结构元素允许目标缓冲区从偶数地址开始。如果需要，在目标缓冲区之前放置填充整数将确保其在 4 字节边界上的位置。

磁带标记的升序层次结构定义如下：记录标记、文件标记 (EOF)、设置标记和 EOD（数据结束标志）。并不是所有设备都支持所有类型的磁带标记，但层次结构内的定位则支持。每种类型的标记通常用于包含一个或多个较低层的标记。

当分隔开许多特定类型的磁带标记时，较高层的标记（EOD 除外）不会终止磁带移动且包括在计数中。例如，可以使用 MTFSR 忽略记录标记和文件标记。

读取 EOF 标记是作为成功的零长度读取返回的；即，返回的数据计数为零，而且将磁带定位在 EOF 之后，从而使下一次读取可以返回下一条记录。

DDS 设备还支持设置标记，设置标记用于描述文件组（集）。读取设置标记也作为零长度读取返回。文件标记、

设置标记和 EOD 可以由 **mt_gstat** 字段中的唯一位区分。

走带操作（前进或后退、设置标记、文件或记录）将磁带定位到以运动方向走带的对象的更远处。例如，后退一个文件使磁带位于文件标记之前；前进一个文件使磁带位于文件标记之后。这与标准的磁带用法是一致的。

忽略磁带设备上的 *lseek*(2) 类型查找。相反，可以使用下面的 *ioctl*(2) 操作定位磁带并确定其状态。

头文件 **<sys/mtio.h>** 包含用于磁带处理的有用信息。

用于解码选项的宏

持久性磁带设备专用文件的设备 ID (**dev_t**) 中的次设备号将不再编码磁带设备选项（例如，密度、访问方式等）。因此，下面提供的各个宏（在 **<sys/mtio.h>** 头文件中定义）不会正确解释持久性 (agile) DSF 的选项。这些宏包括：

M_INSTANCE(dev)	M_TARGET(dev)
M_LUN(dev)	M_BERKELEY(dev)
M_NO_REWIND(dev)	M_USER_CONFIG(dev)
M_INDEX(dev)	M_INDEX_PUT(dev,index)
M_DFLT_DENSITY(dev)	M_DFLT_DENSITY_PUT(dev,density)
M_TRANSPARENT_MODE(dev)	M_PROP_TBL_ACCESS(dev)

上述各宏将按以前的方式，继续在 Legacy DSF 上运行。

应用程序应使用下述方法来解码持久性设备文件中的磁带设备。

libIO(3X) API **io_dev_to_options** 用于解码持久性设备文件中的设备选项，如下所示：

```
#include <libIO.h>
#include <sys/_inttypes.h>
#include <fcntl.h>
```

注释： **libIO** 调用应包含在对 **io_init()** 和 **io_end()** 的调用之中。有关详细信息，请参考 *libIO*(3X) 联机帮助页。要访问这些 API，应用程序必须链接到 libIO 库。

mt_get_newdev_options() 和 **mt_check_newdev_options()** 是供以下代码段使用的实用程序函数。

```
uint64_t
mt_get_newdev_options(dev_t dev, int dev_type) {
    uint64_t       options;
    int            err;
    err = io_dev_to_options(dev, dev_type, &options);
    if (err == IO_ERROR)
          return 0;
    return (options);
}
```

```
uint64_t
mt_check_newdev_options(dev_t dev, int dev_type, uint64_t bitmask) {
    uint64_t       options;
    int            err;
    err = io_dev_to_options(dev, dev_type, &options);
    if (err == IO_ERROR)
          return 0;
    return (options & bitmask);
}
```

例如，下面提供的宏 **M_BERKELEY_AGILE** 可解码 Legacy DSF 和持久性 (agile) DSF 的设备选项。如果设备 ID 是支持 Berkeley 访问方式的设备专用文件的 ID，则该宏将返回 True。

举例

File test.c :

```
#include <stdlib.h>
#include <sys/libIO.h>
#include <sys/_inttypes.h>
#include <sys/stat.h>
#include <sys/errno.h>
#include <fcntl.h>
#include <sys/mtio.h>

#define MT_IS_LEGACY_DEV 1

#define M_BERKELEY_AGILE(dev)                                   \
        ((io_is_legacy_dev(dev, D_CHR) == MT_IS_LEGACY_DEV) ?   \
        (dev & MT_BSD_MASK)                     :               \
        (mt_check_newdev_options(dev, D_CHR, MT_BSD_MASK)))

/ *
  * It is assumed that definitions of mt_get_newdev_options() and
  * mt_check_newdev_options() are defined by the application and
  * available. Omitted here for the sake of simplicity.
  */

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

```
        struct stat stbuf;
        dev_t dev;

        /* Device special file is passed as argv[1] */

        if (stat(argv[1], &stbuf) < 0)
        {
           perror("stat(): ");
           exit (1);
        }

        dev = stbuf.st_rdev;

        io_init(O_RDWR);

        if(M_BERKELEY_AGILE(dev))
           printf(" This is a Berkeley style device file ");
        else
           printf(" This is not a Berkeley style device file ");

        io_end();

        exit(0);
    }
```

Compile Line: cc -Ae -o test test.c -lIO

Sample Output:

```
# ./test /dev/rtape/tape1_BESTn
This is not a Berkeley style device file
# ./test /dev/rtape/tape1_BESTb
This is a Berkeley style device file
# ./test /dev/rmt/0mnb
This is a Berkeley style device file
# ./test /dev/rmt/c5t4d0BEST
This is not a Berkeley style device file
# ./test /dev/rmt/c5t4d0BESTnb
This is a Berkeley style device file
```

可写入与上述宏相似的宏，以便替换其各自的旧宏，如下所示：

```
#define M_INSTANCE_AGILE(dev)                                          \
    ((io_is_legacy_dev(dev, D_CHR) == MT_IS_LEGACY_DEV)    ?           \
    (((dev) >> MT_INSTANCE_BIT_POS) & MT_INSTANCE_MASK) :              \
    (((mt_get_newdev_options(dev, D_CHR)) >> MT_INSTANCE_BIT_POS) \
                                & MT_INSTANCE_MASK))

#define M_TARGET_AGILE(dev)                                            \
    ((io_is_legacy_dev(dev, D_CHR) == MT_IS_LEGACY_DEV)    ?           \
    (((dev) >> MT_TARGET_BIT_POS) & MT_TARGET_MASK)    :               \
    ((mt_get_newdev_options(dev, D_CHR)) >> MT_TARGET_BIT_POS)   \
                                & MT_TARGET_MASK))

#define M_LUN_AGILE(dev)                                               \
    ((io_is_legacy_dev(dev, D_CHR) == MT_IS_LEGACY_DEV)   ?            \
    (((dev) >> MT_LUN_BIT_POS) & MT_LUN_MASK)          :               \
    ((mt_get_newdev_options(dev, D_CHR) >> MT_LUN_BIT_POS)             \
                                & MT_LUN_MASK))

#define M_USER_CONFIG_AGILE(dev)                                       \
    ((io_is_legacy_dev(dev, D_CHR) == MT_IS_LEGACY_DEV) ?              \
    (dev & MT_USER_CONFIG_MASK)                :                      \
    (mt_check_newdev_options(dev, D_CHR, MT_USER_CONFIG_MASK)))

#define M_INDEX_AGILE(dev)                                             \
    ((io_is_legacy_dev(dev, D_CHR) == MT_IS_LEGACY_DEV) ?              \
    (((dev) & MT_INDEX_MASK) >>  MT_INDEX_BIT_POS)   :                 \
    ((mt_check_newdev_options(dev, D_CHR, MT_INDEX_MASK)) >>       \
                                MT_INDEX_BIT_POS));

#define M_INDEX_PUT_AGILE(dev,index)                                   \
    ((io_is_legacy_dev(dev, D_CHR) == MT_IS_LEGACY_DEV) ?              \
    (((dev) & (~MT_INDEX_MASK))  |                                     \
    (index << MT_INDEX_BIT_POS) |                                      \
     MT_USER_CONFIG_MASK)                  :                           \
    ((mt_check_newdev_options(dev, D_CHR, ~MT_INDEX_MASK)) |       \
     (index << MT_INDEX_BIT_POS)))

#define M_DFLT_DENSITY_PUT_AGILE(dev,density)                          \
    ((io_is_legacy_dev(dev, D_CHR) == MT_IS_LEGACY_DEV) ?              \
```

```
        (((dev) & (~MT_DENSITY_MASK))   |                                   \
        (density << MT_DENSITY_BIT_POS))            :                       \
        ((mt_check_newdev_options(dev, D_CHR, ~MT_DENSITY_MASK)) |  \
                        (density << MT_DENSITY_BIT_POS)))

#define M_TRANSPARENT_MODE_AGILE(dev)                                       \
        ((io_is_legacy_dev(dev, D_CHR) == MT_IS_LEGACY_DEV) ?           \
        (((dev) & MT_TRANSPARENT_MASK) ==                                   \
                        MT_TRANSPARENT_VAL)     :                           \
        ((mt_check_newdev_options(dev, D_CHR, MT_TRANSPARENT_MASK))  \
                        == MT_TRANSPARENT_VAL))
```

以下宏包含在 **<sys/mtio.h>** 中，并描述了可能的磁带操作：

```
/* mag tape I/O control requests */

#define  MTIOCTOP  _IOW('m', 1, struct mtop)        /* do mag tape op */
#define  MTIOCGET  _IOR('m', 2, struct mtget)       /* get tape status */

/* structure for MTIOCTOP - mag tape op command */

struct mtop {
   short mt_op;                 /* operations defined below */
   int32_t mt_count;            /* how many of them */
};

/* operations */

#define MTWEOF     0  /* write filemark (end-of-file record) */
#define MTFSF      1  /* forward space file */
#define MTBSF      2  /* backward space file */
#define MTFSR      3  /* forward space record */
#define MTBSR      4  /* backward space record */
#define MTREW      5  /* rewind */
#define MTOFFL     6  /* rewind and put the drive offline (may eject) */
#define MTNOP      7  /* no operation, may set status */
#define MTEOD      8  /* DDS, QIC and 8MM only - seek to end-of-data */
#define MTWSS      9  /* DDS and 8MM only - write setmark(s) */
#define MTFSS      10  /* DDS and 8MM only - space forward setmark(s) */
```

```
#define MTBSS           11  /* DDS and 8MM only - space backward setmark(s) */
#define MTSTARTVOL      12  /* Start a new volume (for ATS) */
#define MTENDVOL        13  /* Terminate a volume (for ATS) */
#define MTRES           14  /* Reserve Device */
#define MTREL           15  /* Release Device */
#define MTERASE         16  /* Erase media */

/* structure for MTIOCGET - mag tape get status command */

struct mtget {
    long    mt_type;        /* type of magtape device */
    long    mt_resid;       /* residual count */

/* The following two registers are device dependent */

    long    mt_dsreg1;      /* status register (msb) */
    long    mt_dsreg2;      /* status register (lsb) */

/* The following are device-independent status words */

    long    mt_gstat;       /* generic status */
    long    mt_erreg;       /* error register */
    int32_t  mt_fileno;     /* No longer used - always set to -1 */
    int32_t  mt_blkno;      /* No longer used - always set to -1 */
```

有关解码 **mt_type** 字段的信息可以在 **<sys/mtio.h>** 中找到。

Legacy 设备和 Agile 设备的磁带操作的工作方式相同。

其他磁带状态特性

为了高效使用具有大内部缓冲区和即时报告功能的流式磁带机，需要执行下列磁带末尾过程：

如果实际写入了磁带，则在 LEOT （磁带的逻辑末尾）附近的所有写入完成，且不出现错误。磁带驱动程序确定已经过 LEOT 后，就不会发生后续写入，并返回错误消息。

要在该点之外进行写入（请牢记流式驱动器的写入已超出 LEOT 范围），仅须使用 **MTIOCGET** ioctl 查询状态。如果状态反映为 EOT 条件，则该驱动程序将忽略所有写入限制。

当给定写入文件标记（对于所有设备）或写入设置标记（对于支持设置标记的设备）命令，并将计数设置为零时， **estape** 和 **stape** 驱动程序将刷新设备缓冲区。

禁用即时报告时，遇到 LEOT 的写入将返回错误，而且磁带驱动程序自动备份此记录。

在磁带末尾附近读取时，不将 LEOT 通知用户。相反，典型的双 EOF 标记或预先排列的数据模式会指示磁带的逻辑末尾。

由于各种磁带机在 EOT 传感上存在不同（因为传感器的物理放置有差异），因此必须对要求数据从一个磁带的 EOT 区域持续到另一磁带的任何应用程序（如多磁带 *cpio*(1) 备份）加以限制。因此，创建和读取磁带时的磁带机类型和模式应该是相同的。

下列各宏在 **<sys/mtio.h>** 定义，用于对 **MTIOCGET** 返回的状态字段 mt_gstat 进行解码。对于每个宏，输入参数 *x* 为 **mt_gstat** 字段。

GMT_BOT(*x*)	在磁带的开头返回 TRUE。
GMT_EOD(*x*)	如果遇到 DDS、QIC 或 8MM 的数据结束标志，则返回 TRUE。
GMT_EOF(*x*)	在文件结束标记处返回 TRUE。
GMT_EOT(*x*)	在磁带末尾返回 TRUE。
GMT_IM_REP_EN(*x*)	如果启用即时报告模式，则返回 TRUE。
GMT_ONLINE(*x*)	如果驱动器处于联机状态，则返回 TRUE。
GMT_SM(*x*)	如果遇到设置标记，则返回 TRUE。
GMT_WR_PROT(*x*)	如果磁带是写保护的，则返回 TRUE。
GMT_COMPRESS(*x*)	如果启用数据压缩，则返回 TRUE。
GMT_DENSITY(*x*)	返回当前配置的 8 位密度值。支持的值在 **<sys/mtio.h>** 中定义。
GMT_D_800(*x*)	如果在 **mt_gstat** 中编码的密度为 800 bpi，则返回 TRUE。
GMT_D_1600(*x*)	如果在 **mt_gstat** 中编码的密度为 1600 bpi，则返回 TRUE。
GMT_D_6250(*x*)	如果在 **mt_gstat** 中编码的密度为 6250 bpi（有压缩或无压缩），则返回 TRUE。
GMT_D_6250c(*x*)	如果在 **mt_gstat** 中编码的密度为 6250 bpi 并进行压缩，则返回 TRUE。
GMT_D_DDS1(*x*)	如果在 **mt_gstat** 中编码的密度为 DDS1（有压缩或无压缩），则返回 TRUE。
GMT_D_DDS1c(*x*)	如果在 **mt_gstat** 中编码的密度为 DDS1 并进行压缩，则返回 TRUE。
GMT_D_DDS2(*x*)	如果在 **mt_gstat** 中编码的密度为 DDS2（有压缩或无压缩），则返回 TRUE。
GMT_D_DDS2c(*x*)	如果在 **mt_gstat** 中编码的密度为 DDS2 并进行压缩，则返回 TRUE。

GMT_D_DLT_42500_24(*x*)	如果在 **mt_gstat** 中编码的密度为 42500 bpi（24 个磁道对），则返回 TRUE。
GMT_D_DLT_42500_56(*x*)	如果在 **mt_gstat** 中编码的密度为 42500 bpi（56 个磁道对），则返回 TRUE。
GMT_D_DLT_62500_64(*x*)	如果在 **mt_gstat** 中编码的密度为 62500 bpi（有压缩或无压缩），则返回 TRUE。
GMT_D_DLT_62500_64c(*x*)	如果在 **mt_gstat** 中编码的密度为 62500 bpi 并进行压缩，则返回 TRUE。
GMT_D_DLT_81633_64(*x*)	如果在 **mt_gstat** 中编码的密度为 81633 bpi（有压缩或无压缩），则返回 TRUE。
GMT_D_DLT_81633_64c(*x*)	如果在 **mt_gstat** 中编码的密度为 81633 bpi 并进行压缩，则返回 TRUE。
GMT_D_DLT_85937_52(*x*)	如果在 **mt_gstat** 中编码的密度为 85937 bpi（有压缩或无压缩），则返回 TRUE。
GMT_D_DLT_85937_52c(*x*)	如果在 **mt_gstat** 中编码的密度为 85937 bpi 并进行压缩，则返回 TRUE。
GMT_D_3480(*x*)	如果在 **mt_gstat** 中编码的密度用于 3480 设备（有压缩或无压缩），则返回 TRUE。
GMT_D_3480c(*x*)	如果在 **mt_gstat** 中编码的密度用于 3480 设备（有压缩），则返回 TRUE。
GMT_DR_OPEN(*x*)	不适用于当前支持的任何设备。始终返回 FALSE。

HP-UX 以无提示方式对大 I/O 请求强制一个磁带记录分块系数 (**MAXPHYS**)。例如，长度为 **MAXPHYS** 的十倍的用户写入请求，实际上将作为十个单独的记录到达介质。后续读取（长度为 **MAXPHYS** 的十倍）在用户看来好像是单个操作，即使 HP-UX 已将它分解为对驱动程序的十个单独读取请求。在写入期间，分块功能对用户是透明的。除非出现下列情况，否则在读取期间它也是透明的：

- 用户选取大于 **MAXPHYS** 的任意读取长度。
- 用户试图读取包含大于 **MAXPHYS** 的记录的第三方磁带。

由于 **MAXPHYS** 的值相对较大（通常 >= 256KB），因此这通常不是问题。

MTNOP 操作不设置与设备无关的状态字。

举例

假定 *fd* 是有效的文件描述符，以下示例在磁带上写入两个连续的文件标记：

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

```
#include <sys/mtio.h>

struct  mtop mtop;

mtop.mt_op = MTWEOF;
mtop.mt_count = 2;
ioctl(fd, MTIOCTOP, &mtop);
```

如果 *fd* 对于打开的 DDS 驱动器是有效的文件描述符，则以下示例向前走带，刚好超过下一个设置标记：

```
#include <sys/types.h>
#include <sys/mtio.h>

struct  mtop mtop;

mtop.mt_op = MTFSS;
mtop.mt_count = 1;
ioctl(fd, MTIOCTOP, &mtop);
```

假定 *fd* 是已打开磁带设备的有效文件描述符，并且它刚从 *read*(2) 请求返回 0。以下系统调用验证磁带是否刚读取文件标记：

```
#include <sys/types.h>
#include <sys/mtio.h>

struct mtget mtget;

ioctl(fd, MTIOCGET, &mtget);
if (GMT_EOF (mtget.mt_gstat)) {
/* code for filemark detection */
}
```

警告

密度规范 **BEST** （标准命名约定）激活支持压缩的磁带设备上的数据压缩。对于链接到这些文件的使用 HP-UX 10.0 之前命名约定的文件（请参阅上面的“命名约定”），也是这样。

对于持久性磁带 DSF，次设备号不会解码任何配置选项。该次设备号表示持久性内核数据库的索引，配置选项存储在该数据库中。

建议将所有 Legacy 磁带设备文件放在 **/dev/rmt** 目录中。位于 **/dev/rmt** 目录外使用扩展配置选项的 Legacy 设备文件可能无法提供跨系统重新引导的一致行为。

虽然可能会在除 **/dev/rtape** 之外的目录中创建持久性 DSF，HP 仍推荐仅在 **/dev/rtape** 中创建持久性磁带 DSF。

使用 *rmsf*(1M) 命令可清除未使用的设备文件。否则，属性表可能溢出，并导致 *mksf*(1M) 命令失败。

在 **<sys/mtio.h>** 中列出的密度代码具有设备相关的行为。请参阅磁带设备的硬件手册以了解哪些密度是有效的。对于某些设备，这些值可能称为格式，而不是密度。

使用不缓冲模式可能会降低性能并增加介质损耗。

相关内容

用于 **stape** 的驱动程序特定的选项（主设备号 **205**）

为访问 **stape** 驱动程序的磁带机创建 Legacy DSF 时，可以使用下列选项：

e 启用穷举模式（缺省情况下禁用它）。
如果启用穷举模式，则驱动程序将在打开设备时根据需要尝试几个不同的配置选项。第一次尝试紧跟次设备号配置之后，但是如果它失败了，则驱动程序尝试其他可能的配置值。

如果禁用穷举模式，则驱动程序仅尝试一次，使用次设备号中所示的配置来配置设备。

p 指定一个分区磁带，其当前活动的分区是分区 1（距 BOT（磁带的开头）最近）。可选的分区 1 距 BOT 最近，以便有可能用作卷目录。无此选项的缺省分区是分区 0。如果不支持对磁带进行分区，则将整个磁带称为分区 0。

s[#] 指定固定块模式；可选的数值指示块大小。如果此数值不存在，则驱动程序选择适合于设备类型的缺省块大小。

作者

mt 由 HP 和加州大学伯克利分校联合开发。

文件

/dev/rtape/*	estape 驱动程序支持的持久性磁带 DSF
/dev/rmt/*	Legacy 磁带 DSF
<sys/mtio.h>	用于磁带的常量和宏
/etc/mtconfig	用于磁带的配置属性表
/dev/rmt/*config	用于访问配置属性表的设备文件 - 仅供内部使用

另请参阅

dd(1)、 mt(1)、 insf(1M)、 lssf(1M)、 mksf(1M)、 ioscan(1M)、 rmsf(1M)、 ioctl(2)、 lseek(2)、 libIO(3X)、 intro(7)。

《Configuring HP-UX for Peripherals》

名称

ndp - 邻节点发现协议 (NDP)

说明

邻节点发现协议 (NDP) 是主机和路由器使用的一种协议，其作用为：

1. 查找已知要附加到相同链路的邻节点的链路层地址。
2. 查找希望代表其自身转发数据包的相邻路由器。
3. 主动跟踪有哪些邻节点可以到达，哪些不可到达。
4. 当指向某个路由器的路径失败时，搜索备用路由器。

为完成上述任务， NDP 将会定义以下过程：

1. 路由器和前缀发现

路由器发现是一个过程，主机可通过此过程查找相邻路由器，及了解地址自动配置所必需的前缀和其他参数。

主机可使用前缀发现来了解驻留在链路上的 IPv6 地址的范围，无须通过路由器便可到达此地址。

路由器可发送能使主机将其视为缺省路由器的路由器广播。路由器广播还包含用于标识链路上的 IPv6 地址范围的前缀信息选项（子网前缀）。

2. 路由器和主机要求

NDP 中的路由器要求为用作路由器的主机指定了一组规则。这些规则包括：

- 路由器配置变量。

  这些配置变量包括连续主动发送的路由器广播之间的间隔，等等。

- 如何将一个接口变成广播接口。

  将某个接口变成广播接口后，即意味着该节点将要定期发送路由器广播，并且打算代表该链路上的主机转发数据包。

- 路由器广播的消息内容。

  路由器将在广播接口上发送定期和请求的路由器广播。 NDP 指定这些消息的格式。

- 主动发送路由器广播。

  除了因响应路由器请求而发送请求的路由器广播外，路由器还可主动发送路由器广播。例如，可以主动发送路由器广播，以使某个前缀过期，或者广播一个新的前缀，等等。

- 停止某个接口上的路由器广播。

  路由器可以停止在某个接口上广播前缀。当由于只有一个路由器而停止该路由器时，可能会因系统管理决策而发生这种情况。 NDP 将指定在这些情况下该路由器应采取的措施。

- 处理路由器请求消息。

  作为无状态自动配置过程的一部分，主机将发送路由器请求。路由器应通过路由器广播响应这些请求。

- 当路由器的链路本地地址发生更改时应采取的措施。

  通常，不应更改路由器的链路本地地址。但是，NDP 仍然定义了当路由器针对其任何接口更改其链路本地地址后应采取的措施。

主机要求是适用于 IPv6 主机的一组规则。这些规则包括：

- 必须维护的 IPv6 变量。

  这些变量包括重新传输邻节点请求的时间间隔、每个接口的链路 MTU，等等。

- 处理路由器广播。

  此规则说明收到路由器广播后，应采取哪些措施。

- 超时前缀和缺省路由器。

  只要路由器发送路由器广播，这些广播就会包括路由器生命周期及所广播的前缀。 NDP 将指定这些生命周期过期后，主机应采取哪些措施。

- 选择缺省路由器。

  如果链路中有多个路由器，缺省路由器的选择算法则会生效。此算法可根据可达性等因素，帮助选择缺省的路由器。

- 发送路由器请求。

  如果启用了一个接口，则主机不需要等待主动提供的路由器广播，而是发送一个路由器请求，然后获取作为响应的路由器广播。这有助于在启用接口后，立即接收缺省的路由器和前缀信息。

3. 发送数据包的算法

任何 IPv6 主机都必须维护发送数据包的算法所使用的某些数据结构。这些数据结构包括：

邻节点缓存
: 一组条目，可维护刚刚向其发送数据包的邻节点的 IPv6 地址到链路层地址映射。除此之外，它还可维护用于检测邻节点不可达性所需的信息，例如可达性状态等。

目标缓存
: 最近向其发送了数据包的主机的一组条目。这既包括在链路上的主机，也包括不在链路上的主机。此数据结构包含邻节点缓存的间接层次。

前缀列表
: 这是一个前缀列表，用于定义链路上的一组 IPv6 地址。将基于每个接口维护此信息。通常，此列表是通过从路由器收到的路由器广播构建的。

缺省路由器列表

可代表该主机转发数据包的路由器列表。同样，此列表包括一个指向各个路由器的邻节点缓存条目的指针。

一个主机向另一个主机发送数据包时，将使用上述数据结构。下面是将数据包发送到单播目标所使用的概念算法。

a. 在发出一个数据包之前，应确定下一跳。通常不需要对所有数据包确定下一跳。下一跳确定的结果将存储在目标缓存中。主机首先应检查目标缓存，找出与当前目标地址匹配的任何条目。如果找到匹配项，则继续执行下面的步骤“c”。

b. 如果目标缓存中没有目标条目，则将最长的前缀与前缀列表中的所有前缀进行匹配。如果存在匹配项，则确定该目标在链路上，而目标地址被视为下一跳。否则，将从路由选择表中确定下一跳。

c. 一旦确定了下一跳，将针对它执行地址解析过程和邻节点不可达性检测。下一节将说明此过程。

d. 如果已知邻节点是可以到达的，则向该目标发送数据包。

4. 地址解析和邻节点不可达性检测

地址解析是用于确定某个邻节点的链路层地址的过程。通过此过程找到的 IPv6 地址至链路层地址的映射将被缓存于邻节点缓存中。以下是地址解析所包含的步骤。

a. 首先检查邻节点缓存，找出与当前的目标地址匹配的条目。如果不存在该条目，则主机会将一个邻节点请求消息发送到请求节点组播组。此组播地址是基于目标 IPv6 地址派生的，具有特定 IPv6 地址的所有节点必须加入该组。

b. 如果网络中存在具有指定 IPv6 地址的主机，则此主机将使用邻节点广播消息答复此请求。

c. 收到邻节点广播后，节点将在发送方 IPv6 地址的邻节点缓存中搜索一个条目。将在该邻节点缓存中创建一个新条目，同时将可达性标记设置为 REACHABLE。

一旦完成地址解析，便将执行邻节点不可达性检测。此过程取决于邻节点缓存的可达性字段。邻节点缓存中的条目可能处于以下任何一种状态：

INCOMPLETE 正在进行地址解析，并且尚未确定目标的链路层地址。

REACHABLE 到目前为止，该目标是可以到达的。

STALE 已知该目标不再是可以到达的，但是，在必须将数据包发送到该目标之前，不需要执行可达性检测。

DELAY 此状态是一种优化，可为上层协议提供额外的时间，以进行可达性确认。

PROBE 已通过反复发送邻节点请求，主动请求了可达性确认。

在邻节点不可达性检测期间，节点将检查邻节点缓存中的状态。如果目标的状态为 REACHABLE，则发送数据包。否则，将执行以下步骤：

a. 对某个目标执行地址解析后，将在该目标的邻节点缓存中创建一个条目，同时将可达性状态设置为 INCOMPLETE。如果地址解析失败，则删除此条目。

b. 如果地址解析通过，则将使用该目标的链路层地址填充此条目，同时将状态设置为 REACHABLE。

c. 系统维护了一个称为“可达性计时器”的计时器，它能使邻节点缓存中的某个条目的状态过期。一旦此计时器过期，可达性状态将由 REACHABLE 更改为 STALE。

d. 将数据包发送到邻节点缓存中的状态为 STALE 的目标时，节点会将此状态设置为 DELAY，然后启动与该状态关联的计时器。在计时器过期之前，如果该节点收到可达性确认，则状态将设置为 REACHABLE。否则，状态将设置为 PROBE。

e. 一旦条目的状态为 PROBE，节点会将单播邻节点请求发送到该条目中指定的链路层地址。如果节点收到一个响应的邻节点广播，则状态将设置为 REACHABLE。将反复发送此请求，并且可配置最大次数。如果在达到最大请求次数后，仍然没有收到可达性确认，则从邻节点缓存中删除此条目，然后再次执行地址解析。

注释：如果节点收到了主动提供的邻节点广播、路由器广播和路由器请求等，则也可能在邻节点缓存中创建条目。但是，对于在这些情况下创建的条目，可达性状态始终设置为 STALE。

5. 重定向功能

如果路由器在相同的链路上找到更好的下一跳，则会向主机发送一个重定向消息。这是对路由器的要求。

收到路由器重定向消息后，主机应使用新的下一跳地址更新其目标缓存。

作者

NDP 由 Internet Engineering Task Force 的 IPng Working Group 开发。

另请参阅

ifconfig(1M)、 ndp(1M)、 ip6(7P)、 lan(7)。

Neighbor Discovery for IPv6，RFC2461，T. Narten 等。 NDP 邻节点发现协议

名称

nfs、NFS - 网络文件系统

说明

网络文件系统 (NFS) 允许客户端节点通过网络访问透明文件。通过使用 NFS，客户端节点可在位于各种服务器和服务器体系结构上并且跨各种操作系统的文件上运行。客户端上的文件访问调用（例如，读取请求）被转换为 NFS 协议请求并且通过网络发送到服务器系统。服务器接收该请求，执行实际文件系统操作，并且发送响应回客户端。

NFS 使用构建于外部数据表示 (XDR) 协议顶层的远程过程调用 (RPC)，以无状态的方式运行。出于安全目的，RPC 协议允许在网络上交换版本和身份验证参数。

通过将文件系统条目添加到服务器的 **/etc/dfs/dfstab** 文件，服务器授予客户端访问特定文件系统的权限。

客户端使用 **mount** 命令请求文件系统的文件句柄，以获取该文件系统的访问权（请参阅 *mount*(1M) ）（文件句柄是 NFS 标识远程文件的方法）。客户端挂接文件系统后，服务器为将针对客户端访问的每个文件（或目录）向客户端发出文件句柄。如果文件在服务器端被删除，文件句柄将会失效（与已知文件分离），并且服务器返回错误，并将 **errno** 设置为 [ESTALE] 。

对于服务器通过网络挂接的文件系统来说，服务器也可以是客户端；不过，该服务器的客户端不能直接访问那些文件系统。如果客户端尝试挂接一个服务器是 NFS 客户端的文件系统，服务器则以 **errno** 设置为 [EREMOTE] 的形式返回。客户端必须将文件系统直接挂接在文件系统驻留的服务器上。

客户端与服务器上的用户 ID 和组 ID 映射必须一致。不过，在执行客户端访问检查前，服务器会将 UID 0 （超级用户）映射为 UID –2。该进程可防止在远程文件系统上获取超级用户权限。

返回值

通常，将向客户端返回在服务器上检测到的物理磁盘 I/O 错误，以便进行相应的操作。如果服务器已关闭或不可访问，则客户端将收到消息：

NFS: file server xxx not responding: still trying.

其中 **xxx** 是 NFS 服务器的主机名。客户端继续重新发送请求，直到它收到服务器应答为止。因此，无需客户端进行任何特殊操作，服务器可以在崩溃或关机后重新启动。请求 I/O 的客户端进程将被阻塞，但保持对信号的识别能力（除非用 **nointr** 选项挂接），直到服务器恢复为止。不过，如果用 **soft** 选项挂接，客户端进程将会返回错误而不是无限期等待。

作者

nfs 由 Sun Microsystems, Inc. 开发。

另请参阅

exportfs(1M)、 share(1M)、 mount(1M)、 mount_nfs(1M)、 nfsd(1M)、 mount(2)、 fstab(4)、 dfstab(4)。

名称

null - 空文件

说明

写入空专用文件的数据将被忽略。

读取空专用文件始终返回 0 字节。

举例

要创建长度为零的文件，请使用下列任一命令：

cat /dev/null > *file*
cp /dev/null *file*

文件

/dev/null

符合的标准

null：AES、SVID2、SVID3、XPG2、XPG3、XPG4、FIPS 151-2、POSIX.1 和 POSIX.2

名称

pckt - STREAMS pty（伪终端）的 Packet Mode（数据包模式）模块

概要

#include <sys/stropts.h>

int ioctl(fd_slave, I_PUSH, "pckt");

说明

STREAMS pty 设备的 **Packet Mode** 功能允许 pty 设备主端上的用户进程了解 pty 上的状态更改。要在 STREAMS pty 设备上启用 **Packet Mode** ，用户进程必须调用 STREAMS **I_PUSH** *ioctl*(2) 系统调用，将 **pckt** 模块推到 pty 的主端。当 **pckt** 模块被推到 STREAMS pty 主端时，主端上上行发送的某些 STREAMS 消息将会被打包，以便随后可由主端使用 **getmsg** 函数进行检索。

当用户进程写入数据时， **pckt** 模块将消息按原样下行传递到下一个模块或驱动程序。当用户进程读取数据或当 **pckt** 模块接收某些 STREAMS 消息类型时，它将会从消息中构建一个数据包以上行转发。要构建一个消息数据包，模块需创建一条 **M_PROTO** 消息。该 **M_PROTO** 消息包含第一个数据块中的原始消息类型和所需数据块中的原始消息。然后通过调用 **getmsg()** 函数，用户进程可以检索 **M_PROTO** 消息。

pckt 模块将下列 STREAMS 消息类型打包：

M_DATA 、 **M_IOCTL** 、 **M_PROTO** 、 **M_PCPROTO** 、 **M_FLUSH** 、 **M_START** 、 **M_STOP** 、 **M_STARTI** 、 **M_STOPI** 、 **M_READ** 。

所有其他消息将按原样上行传递。

如果消息是 **M_FLUSH** 消息，则 **pckt** 模块将会查看标志并执行下列操作：

- 如果标志是 **FLUSHW** ，则模块在创建 **M_PROTO** 消息并将消息上行传递之前，将其更改为 **FLUSHR** 。这可防止数据流头部的读取列队被原始 **M_FLUSH** 消息刷新。
- 如果标志是 **FLUSHR** ，则模块在创建 **M_PROTO** 消息并将消息上行传递之前，将其更改为 **FLUSHW** 。要正确刷新写入列队，模块还将发送一条设置了 **FLUSHW** 标志的 **M_FLUSH** 消息。
- 如果标志是 **FLUSHRW** ，则模块在创建 **M_PROTO** 消息并将消息上行传递之前，将其更改为 **FLUSHW** 。要正确刷新写入列队，模块还将发送一条设置了 **FLUSHW** 标志的 **M_FLUSH** 消息。

作者

pckt(7) 由 HP 和 OSF 联合开发。

另请参阅

getmsg(2)、 ioctl(2)、 ptm(7)、 pts(7)、 ldterm(7)、 ptem(7) 和 streamio(7)。

名称

poll - 监视多个文件描述符上的 I/O 条件

概要

#include <sys/devpoll.h> #include <fcntl.h>

int open("/dev/poll", O_RDWR);

int write(int filedes, const struct pollfd *buf, size_t nbyte);

int ioctl(int filedes, DP_POLL, struct dvpoll *arg);

int ioctl(int filedes, DP_ISPOLLED, struct pollfd *arg);

说明

/dev/poll 可为事件端口驱动程序提供一个接口，使用户能够同步监视与一组注册的文件描述符关联的一组特定的条件。轮询条件包括在不阻塞的情况下读取或写入数据的功能，及某些异常条件。

可通过 **open()** 、 **write()** 和 **ioctl()** 系统调用访问 **/dev/poll** 。

/dev/poll 事件端口提供相当于 **select(2)** 和 **poll(2)** 系统调用的功能，并支持以下类型的文件描述符：网络 (**AF_INET**) 和 Unix 域 (**AF_UNIX**) 套接字、指定的 FIFO 文件和管道、XTI 端点和 STREAMS 设备。

事件端口驱动程序支持的常规操作包括：

-- 打开事件端口。
-- 在事件端口上注册和取消注册文件描述符。
-- 轮询事件端口上的已注册文件描述符。
-- 检索某个文件描述符的已注册轮询条件。
-- 关闭事件端口。

打开事件端口

每次打开 **/dev/poll** 设备都将启用一个事件端口，可通过此事件端口轮询一组不同的文件描述符。 **open()** 系统调用返回的文件描述符代表该事件端口。需要监视多组文件描述符的用户应多次打开 **/dev/poll** 设备。例如：

int evpfd;

evpfd = open("/dev/poll", O_RDWR);

只有针对 **/dev/poll** 执行了 **open()** 的进程才可以执行常规的事件端口操作。特别是，只能关闭子进程从其父进程继承的任何事件端口文件描述符，或者只能关闭使用 Unix 域套接字访问权限从另一进程接收的任何事件端口文件描述符（请参阅 *send*(2) 联机帮助页中的 *sendmsg* ，或者 *streamio*(7) 联机帮助页中的 STREAMS **I_FDINSERT** ioctl *request* ）。

注册和取消注册文件描述符

可使用 **write()** 系统调用将一组相关的文件描述符和轮询条件注册到事件端口。用户可通过将一个 **pollfd** 结构的数组写入到事件端口，在一个 **write()** 服务调用中注册多个文件描述符。 **<poll.h>** （包括在 **<sys/devpoll.h>** 中）中定义了 **pollfd** 结构和相关的轮询条件。 **<sys/devpoll.h>** 文件中定义了其他标志。有关轮询条件的定义，请参阅

poll(2) 联机帮助页。

要注册一个文件描述符，需要将 **fd** 字段设置为要注册的文件描述符，同时将 **events** 字段设置为一个或多个轮询条件，例如 **POLLIN** 。可以将多个轮询条件一起进行逻辑 或运算。可以将某个给定的文件描述符注册到多个事件端口。将某个文件描述符重新注册到同一个事件端口，会导致指定的轮询条件加入给定文件描述符以前的条件。

要取消注册，需要将 **fd** 设置为要取消注册的文件描述符，同时将 **events** 设置为 **POLLREMOVE** 。**<sys/devpoll.h>** 中定义了 **POLLREMOVE** 。不得将 **POLLREMOVE** 与其他任何轮询条件一起进行逻辑 或运算。

当轮询的文件描述符关闭后，它将被自动取消注册。

以下后续示例将在打开的事件端口上注册两个文件描述符，分别为 **fd1** 和 **fd2** ：

```
struct pollfd pfd[2];
int err;


pfd[0].fd = fd1;
pfd[0].events = POLLIN;
pfd[1].fd = fd2;
pfd[1].events = (POLLIN | POLLRDBAND);
err = write(evpfd, pfd, sizeof(pfd));
```

轮询文件描述符

可通过调用指定了 **DP_POLL** *request* 的 **ioctl()** ，开始轮询某个事件端口的相关集。

ioctl *arg* 参数是指向 **<sys/devpoll.h>** 中定义的 **dvpoll** 结构的指针。它包含以下成员：

```
struct dvpoll {
  pollfd_t *dp_fds;    /* pollfd[] to be used */
  nfds_t    dp_nfds;   /* number of pollfd entries */
  int       dp_timeout; /* milliseconds or -1 */
}
```

dp_fds 是一个指向 **pollfd** 结构的数组的指针。 **dp_nfds** 是要在该数组中返回的 **pollfd** 结构的最大数目。**dp_timeout** 是等待出现至少一个符合事件端口中已注册轮询条件的最长时间，其单位为毫秒。

如果符合任何已注册文件描述符的一个或多个已注册轮询条件， **ioctl()** 将在数组中每个 **pollfd** 结构的 **revents** 中存储有效的轮询条件，每个活动的文件描述符存储一个数组元素。 **ioctl()** 的返回值为有效的 **pollfd** 结构的数目。

如果没有符合的轮询条件，并且 **dp_timeout** 为 **-1** ，则在符合任何已注册文件描述符的轮询条件之前， **ioctl()** 将处于休眠状态。如果 **dp_timeout** 为非负值，则在经过 *dp_timeout* 毫秒，或者符合某个轮询条件之后， **ioctl()** 将会返回。如果时间限制过期，则 **ioctl()** 返回值为 **0** 。

检索某个文件描述符的已注册轮询条件

可通过使用 **DP_ISPOLLED** *request* 调用 **ioctl()** 来确定某个相关集中给定文件描述符的已注册轮询条件。例如，对于文件描述符 **fd1** ：

```
struct pollfd pfd;
int ispolled;

pfd.fd = fd1;
ispolled = ioctl(evpfd, DP_ISPOLLED, &pfd);
```

如果文件描述符注册到了事件端口，则 **ioctl()** 返回值为 **1** ，同时在 **pollfd** 结构的 **events** 成员中返回已注册的轮询条件。

如果未注册文件描述符，或者文件描述符未打开，则 **ioctl()** 返回值为 **0** 。

关闭事件端口

close() 系统调用指定事件端口文件描述符，来关闭某个事件端口。已注册到该事件端口的所有文件描述符将从该事件端口自动取消注册。

返回值

open() 返回事件端口文件描述符。如果 **open()** 系统调用失败，它将返回 **-1** ，同时将 **errno** 设置为错误情况。

write() 在已传入 *buf* 的 **pollfd** 结构的数组中返回字节数。如果 **write()** 返回 **-1** ，则将 **errno** 设置为错误情况。

ioctl(DP_POLL) 返回符合其一个或多个轮询条件的文件描述符的数目。如果在符合任何已注册文件描述符的任何轮询条件之前发生超时，则 **ioctl(DP_POLL)** 返回 **0** 。

如果 **pollfd** 结构中指定的文件描述符已注册，则 **ioctl(DP_ISPOLLED)** 返回 **1** 。如果该文件描述符未注册，或者已关闭，则 **ioctl(DP_ISPOLLED)** 返回 **0** 。

如果 **ioctl()** 返回 **-1**, , 则将 **errno** 设置为错误情况。

错误

事件端口驱动程序可返回以下错误。

如果 **open()** 失败，则将 **errno** 设置为下列值之一。

[EACCES]	传递给 **open()** 的设备文件名的次设备号不是 **0** 。
[EAGAIN]	由于出现临时性的情况，分配内部数据结构失败。再次调用 **open()** 可能会成功。
[EMFILE]	已经打开了进程允许的最大数目的文件描述符。
[ENFILE]	已经打开了系统允许的最大数目的文件。
[ENXIO]	某些必需的文件类型不受 **/dev/poll** 驱动程序的支持。请参阅下面的警告一节。

如果 **write()** 或 **ioctl()** 失败，会将 **errno** 设置为下列值之一。

[EACCES] 调用进程没有打开事件端口。

[EBADF] 传递给 **write()** 的 *filedes* 参数不是一个打开的文件描述符。

[EFAULT] 尝试访问一个位于进程地址空间之外的 **pollfd** 结构。

[EINTR] 一个信号中断了 **ioctl(DP_POLL)** 系统调用。

[EINVAL] 传递给 **write()** 的 *nbyte* 参数小于 **0** 。

[ENODEV] 传递给 **write()** 的 *filedes* 参数不是一个事件端口文件描述符。

举例

以下示例说明如何使用 **/dev/poll** 驱动程序来轮询网络套接字文件描述符上的事件。

要注册一个 TCP 套接字文件描述符 (**sd**) ，以便当有新的连接建立时或输入数据可用时， **ioctl(DP_POLL)** 将通知应用程序:

```
struct pollfd regpfd;
int err;

regpfd.fd = sd;
regpfd.events = POLLIN;
err = write(evpfd, &regpfd, sizeof(regpfd));
```

如果应用程序需要区分收到的带外数据，则需要将 **POLLRDBAND** 与 **POLLIN** 一起进行逻辑 或运算。

要等待一个或多个已注册套接字（最多 100 个连接）上的事件:

```
struct pollfd pollpfd[100];
struct dvpoll dvp;
int npoll;

dvp.dp_fds = pollpfd;
dvp.dp_nfds = 100;
dvp.dp_timeout = -1;
npoll = ioctl(evpfd, DP_POLL, &dvp);
```

如果未完成对某个套接字的无阻塞写入，则可以使用以下示例注册该套接字，以便在以后可以再次写入该套接字时， **ioctl(DP_POLL)** 会通知应用程序。通常，该套接字已经注册，可以接收输入通知。以下示例将添加 **POLLOUT** 通知。

```
struct pollfd regpfd;
int err;

regpfd.fd = sd;
regpfd.events = POLLOUT;
```

```
err = write(evpfd, &regpfd, sizeof(regpfd));
```

最后一次无阻塞写入成功后，应使用以下代码段取消注册 **POLLOUT** ，但是需要继续保持对输入通知的注册。请注意，要删除 **POLLOUT** 注册，必须使用 **POLLREMOVE** 。

```
struct pollfd regpfd[2];
int err;

regpfd[0].fd = sd;
regpfd[0].events = POLLREMOVE;
regpfd[1].fd = sd;
regpfd[1].events = POLLIN;
err = write(evpfd, regpfd, sizeof(regpfd));
```

以下示例将使用 **ioctl(DP_ISPOLLED)** 说明如何在更普遍的情况（例如，应用程序库可能不知道在正常情况下文件描述符是如何注册的）下完成相同的操作。

```
struct pollfd regpfd[2];
int err;

regpfd[0].fd = sd;
regpfd[0].events = POLLREMOVE;
regpfd[1].fd = sd;
err = ioctl(evpfd, DP_ISPOLLED, &regpfd[1]);
regpfd[1].events &= ~POLLOUT;   /* clear POLLOUT */
err = write(evpfd, regpfd, sizeof(regpfd));
```

警告

通常， **/dev/poll** 的性能优于 **select()** 和 **poll()** ，特别是当应用程序注册了大量的文件描述符时。但是，如果大量的已注册文件描述符上有可能同时发生指定的情况，则性能差异将会减小。

如果 **open()** 返回 **-1** ，同时将 **errno** 设置为 [ENXIO] ，这表示尚未安装某些必要的系统修补软件，而系统管理员必须安装支持 **/dev/poll** （事件端口）的文件系统、传输和 STREAMS 修补软件。

如果无法注册或取消注册 **pollfd** 结构中的一个或多个文件描述符，则 **write()** 系统调用不返回任何错误指示。

如果将 **POLLREMOVE** 与已传递给 **write()** 的 **pollfd** 结构中的其他轮询条件一起进行逻辑 或运算，则将忽略 **POLLREMOVE** 。会将其他轮询条件与已注册文件描述符的任何现有轮询条件一起进行逻辑 或运算。

ioctl(DP_POLL) 系统调用只返回第一个 *dp_nfds* 活动文件描述符。如果存在其他活动的文件描述符，则不提供指示。

为了与其他某些供应商的 **/dev/poll** 驱动程序的实现兼容， **ioctl(DP_ISPOLLED)** 系统调用还将在 **pollfd** 结构的 **revents** 成员中返回其结果。

如果 **pollfd** 结构中的文件描述符未打开，则 **ioctl(DP_ISPOLLED)** 系统调用不返回任何错误指示。

关闭了某个事件端口后，如果仍然有大量的已注册文件描述符，则 **close()** 系统调用可能需要经过漫长的时间才能完成。

作者

该事件端口驱动程序由 HP 独立开发。

文件

/dev/poll	驱动程序设备文件
/sbin/init.d/devpoll	创建 **/dev/poll** 的启动脚本
/etc/rc.config.d/devpoll	启动脚本的配置参数

另请参阅

ioctl(2)、 mknod(2)、 open(2)、 pipe(2)、 poll(2)、 select(2)、 send(2)、 socket(2)、 socketpair(2)、 write(2)、 t_open(3)。

名称

ps2、ps2kbd、ps2mouse - PS/2 键盘/鼠标设备驱动程序和文件

概要

#include <sys/ps2io.h>

说明

通过 **ps2** 驱动程序，可以在安装了 PS/2 接口硬件的惠普工作站上使用与 IBM Personal System/2 (PS/2) 兼容的键盘和鼠标设备。

在具有一个接口的系统上， PS/2 设备文件名使用以下格式：

/dev/ps2_*n*

其中的 *n* 代表范围介于 0 和 15 之间的接口端口号。例如，可使用设备文件 **/dev/ps2_1** 访问端口一。

在带有多个接口的系统上， PS/2 设备文件名使用以下格式：

/dev/ps2_*m***.***n*

其中的 *m* 代表接口号， *n* 代表端口号。例如，可使用设备文件 **/dev/ps2_1.2** 访问接口一上的端口二。

在引导时， **ps2** 驱动程序将扫描包括从端口零到实现的最大端口号在内的所有接口端口，并尝试标识附加的 PS/2 设备。 **/dev/ps2mouse** 设备文件将访问 **ps2** 检测到的第一个鼠标。 **/dev/ps2kbd** 设备文件将访问 **ps2** 检测到的第一个键盘。

PS/2 设备分类为“慢”设备。这意味着捕获的信号可以中断对 **ps2** （请参阅 *signal*(5) ）的系统调用。

可将鼠标置于两种输出模式之一。在流模式下，鼠标将生成一个三字节报告数据包，以响应鼠标移动和（或）按键。可通过 **read()** 系统调用（请参阅 *read*(2) ）获取这些报告。在提示模式下， **ioctl()** 请求将轮询鼠标，同时在其地址被当作 **ioctl()** 调用的参数进行传递的缓冲区中返回一个三字节报告数据包。

PS/2 键盘可返回代表按键事件和释放事件的键代码。可使用内置仿真终端 (ITE) 读取来自 PS/2 键盘的 ASCII 字符。 *termio*(7) 中介绍了 ITE 使用的 ASCII 终端接口。

ps2 驱动程序为 PS/2 键盘和鼠标提供了一个低级编程接口。要以独立于硬件的方式访问这些设备，请使用 X Window 编程环境。

系统调用

open() 系统调用提供对指定的 PS/2 设备的独占访问（请参阅 *open*(2) ）。如果某个端口已打开，则对该端口执行的所有 **open()** 调用都将失败，同时将 **errno** 设置为 [EBUSY] （请参阅 *errno*(2) ）。

如果尝试打开一个不存在的端口，则 **open()** 调用将会失败，同时将 **errno** 设置为 [ENXIO]。

如果在系统引导时未检测到键盘，同时尝试对 **/dev/ps2kbd** 调用 **open()** ，或者在系统引导时未检测到鼠标，同时尝试对 **/dev/ps2mouse** 调用 **open()** ，则 **open()** 调用将会失败，并将 **errno** 设置为 [ENXIO]。

尝试打开一个没有连接设备的现有 **ps2** 端口将会成功。

如果成功打开，则忽略任何来自设备的先前排队的输入。缺省情况下，键击将被路由到 ITE。当键盘已打开时，

ITE 将不接收来自该键盘的键击；在关闭该键盘设备之前，ITE 对键盘输入拥有独占访问权限。

可设置文件状态标志 **O_NDELAY** 和 **O_NONBLOCK** ，以启用无阻塞读取（请参阅 *open*(2) ）。

read() 返回来自 PS/2 设备的字节。 HP-UX 为每个端口保留了一个 512 字节的缓冲区。如果该缓冲区已满，则会忽略次设备收到的其他字节。

如果提供了足够的缓冲数据来满足全部请求的字节数，则 **read()** 调用将成功完成，并读取所有请求的数据，同时返回已读取的字节数。

如果没有提供足够的缓冲数据来满足整个请求，但是至少有一个可用字节，则 **read()** 调用将成功完成，并读取所有可用数据，同时返回实际读取的字节数。

如果状态标志 **O_NDELAY** 和 **O_NONBLOCK** 都已清除，并且没有可用的数据，则在数据可用或收到信号之前， **read()** 调用将会阻塞。

如果设置了文件状态标志 **O_NDELAY** ，并且没有可用的数据，则 **read()** 调用将返回零而不是阻塞。

如果设置了文件状态标志 **O_NONBLOCK** ，并且没有可用的数据，则 **read()** 调用将返回 **-1** ，同时将 **errno** 设置为 [EAGAIN] （请参阅 *errno*(2) ）。

ps2 不支持 **write()** 系统调用。

可使用 **select()** 系统调用确定当前是否可以从 **ps2** 端口读取数据。将 **select()** 用于写入或异常情况，始终会在文件描述符位掩码（请参阅 *select*(2) ）中返回一个 False 指示。

可使用 **ioctl()** 系统调用对 PS/2 鼠标和键盘设备执行特殊的操作（请参阅 *ioctl*(2) ）。可以将 **ps2** 驱动程序 **ioctl()** 请求集划分为三组：针对于鼠标和键盘的常规请求、特定于键盘的请求和特定于鼠标的请求。针对键盘使用特定于鼠标的请求，以及针对鼠标使用特定于键盘的请求都将失败，同时返回 **-1** ，并将 **errno** 设置为 [EINVAL]。

针对没有连接 PS/2 设备的端口使用任何 **ioctl()** 请求（ **PS2_PORTSTAT** 除外）将会超时，同时返回 **-1** ，并将 **errno** 设置为 [EIO]。

所有 **ioctl()** 系统调用都使用以下语法：

int ioctl(int fildes, int request, char *arg);

所有需要参数或者返回数据的请求都使用 *arg* 参数编址的 4 字节无符号字符缓冲区。

<sys/ps2io.h> 中定义了跟在后面的 *request* 代码。

键盘和鼠标的常规 **ioctl()** 请求

PS2_PORTSTAT 返回驱动程序状态信息。

将在 *arg* 编址的字符缓冲区中返回两个字节的数据。

字节 0，用于指示已连接设备的类型，它包括以下四个可能的值：

PS2_NONE 未检测到设备。

PS2_MOUSE 检测到鼠标。
PS2_KEYBD 检测到键盘。
PS2_UNKNOWN 检测到未知设备。

字节 1 包含各种驱动程序信息的位标志。文件 **/usr/include/sys/ps2io.h** 中定义了该字节的以下位掩码:

INTERFACE_HAS_ITE 如果已设置，则内置仿真终端 (ITE) 会将包含此端口的接口用于键盘输入。

PORT_HAS_FIRST_KEYBD 如果已设置，会将此端口连接到驱动程序检测到的第一个键盘。

PORT_HAS_FIRST_MOUSE 如果已设置，会将此端口连接到驱动程序检测到的第一个鼠标。

当前，其他所有位都未被使用，并被清除为零。

PS2_DISABLE 禁用 PS/2 设备。

设备自身可阻塞从该设备的更多输出。此请求不使用 *arg* 。某些设备除了可禁用自身外，还可执行其他操作。

键盘可将其内部状态重置为缺省状态，停止扫描键，并等待进一步的命令。

鼠标将停止传输报告，然后禁用自身。

PS2_ENABLE 启用 PS/2 设备

将启用次设备的传输。此请求不使用 *arg* 。

PS2_IDENT 标识 PS/2 设备。

将在 *arg* 编址的 4 字节缓冲区中返回用于标识设备类型的值。键盘将返回两个字节 （*arg*[0]=0xAB 和 *arg***[1]=0x83** ）。鼠标将返回一个字节 (*arg***[0]=0x00**) 。

PS2_SETDEFAULT 将设备设置为其缺省（通电）状态。

设备将返回其缺省的内部状态。此请求不使用 *arg* 。

PS2_RESET 重置 PS/2 设备。

指示设备执行其内部重置例行程序，并执行其通电检测。将在 *arg* 编址的 4 字节缓冲区中返回通电检测的结果。鼠标将返回两个字节以指示重置成功（*arg*[0]=0xAA 和 *arg***[1]=0x00** ）。键盘将返回一个字节 (*arg***[0]=0xAA**) 。

特定于键盘的 **ioctl()** 请求

PS2_SCANCODE 选择键盘扫描代码集

会将键盘使用的扫描代码集作为 *arg* 编址的缓冲区的第一个字节进行传递。下面列出了该字节的有效值:

SCANCODE_1	选择扫描代码集 1。
SCANCODE_2	选择扫描代码集 2。
SCANCODE_3	选择扫描代码集 3。
GET_SCANCODE	返回使用的扫描代码。

如果指定了 **GET_SCANCODE** ，则将键盘使用的扫描代码作为 *arg* 编址的字符缓冲区的第一个字节进行传递。某些键盘不一定支持所有的扫描代码集。

PS2_ALL_TMAT 将所有键设置为键盘重复输入行为。

当键盘使用任何扫描代码集时，可以发出此请求；但是，该请求只影响扫描代码集 3 的操作。不使用 *arg* 参数。可通过 **PS2_RATEDELAY ioctl()** 请求设置键盘重复输入速率和延迟。

PS2_ALL_MK 将所有键设置为仅通行为。

当键盘使用任何扫描代码集时，可以发出此请求；但是，它只影响扫描代码集 3 的操作。不使用 *arg* 参数。

PS2_ALL_MKBRK 将所有键设置为接通或断开行为。

当键盘使用任何扫描代码集时，可以发出此请求；但是，它只影响扫描代码集 3 的操作。不使用 *arg* 参数。

PS2_ALL_TMAT_MKBRK 将所有键设置为键盘重复输入接通或断开行为。

当键盘使用任何扫描代码集时，可以发出此请求；但是，它只影响扫描代码集 3 的操作。不使用 *arg* 参数。可通过 **PS2_RATEDELAY ioctl()** 请求设置键盘重复输入速率和延迟。

PS2_KEY_TMAT 为单个键设置键盘重复输入行为。

会将单个键的扫描代码集 3 中的键代码作为 *arg* 编址的字符缓冲区的第一个字节进行传递。当键盘使用任何扫描代码集时，可以发出此请求；但是，它只影响扫描代码集 3 的操作。可通过 **PS2_RATEDELAY ioctl()** 请求设置键盘重复输入速率和延迟。因为完成此请求后，键盘可能会被置于禁用状态，所以应该在执行 **PS2_KEY_TMAT** 后执行 **PS2_ENABLE** 请求。

PS2_KEY_MAKE 为单个键设置仅通行为。

会将单个键的扫描代码集 3 中的键代码作为 *arg* 编址的字符缓冲区的第一个字节进行传递。当键盘使用任何扫描代码集时，可以发出此请求；但是，它只影响扫描代码集 3 的操作。因为完成此请求后，键盘可能会被置于禁用状态，所以应该在 **PS2_KEY_MAKE** 后执行 **PS2_ENABLE** 请求。

PS2_KEY_MKBRK 为单个键设置接通或断开行为。

会将单个键的扫描代码集 3 中的键代码作为 *arg* 编址的字符缓冲区的第一个字节进行传递。将为该键设置接通或断开行为。当键盘使用任何扫描代码集时，可以发出此请求；但是，它只影响扫描代码集 3 的操作。因为完成此请求后，键盘可能会被置于禁用状态，所以应该在 **PS2_KEY_MKBRK** 后执行 **PS2_ENABLE** 请求。

PS2_INDICATORS 根据已传入由 *arg* 编址的字符缓冲区的第一个字节的值，设置键盘指示灯、 Num Lock、Caps Lock 和 Scroll Lock 的状态。

按如下方式对这些指示灯进行位映射：

NONE_LED	没有活动的指示灯
CAPS_LED	Caps Lock 指示灯是活动的
NUM_LED	Num Lock 指示灯是活动的
SCROLL_LED	Scroll Lock 指示灯是活动的

PS2_RATEDELAY 可通过指定作为 *arg* 编址的字符缓冲区的第一个字节进行传递的值，为所有重复输入的键设置速率和延迟。

第零位至第四位指定了速率。第五位至第六位指定了延迟。第七位（最高有效位）未使用，应设置为零。将通过以下方程式确定以毫秒为单位的延迟，其中 *X* 是第五位至第六位的数值：

delay **= (1+***X***) * 250** *(+|– 20%)*

将通过以下方程式确定以秒为单位的时间段（从一个输出键代码到下一个输出键代码的间隔），其中 *Y* 是第零位至第二位的数值， *Z* 是第三位至第四位的数值：

period **= (8+***Y***) * (2^***Z***) * 0.00417** *(+|– 20%)*

对于使用以上方程式的每个时间段，键盘重复输入速率（以每秒通码表示）为一。缺省的键盘重复输入速率为每秒 10.9 个字符。缺省延迟为 500 毫秒。

特定于鼠标的 **ioctl()** 请求

PS2_SAMPLERATE 可通过指定作为 *arg* 编址的字符缓冲区的第一个字节进行传递的值，设置流模式下使用的鼠标采样速率。

支持七个特定的速率：

SAMPLE_10	最大 10 个报告/秒
SAMPLE_20	最大 20 个报告/秒
SAMPLE_40	最大 40 个报告/秒
SAMPLE_60	最大 60 个报告/秒
SAMPLE_80	最大 80 个报告/秒

SAMPLE_100 最大 100 个报告/秒
SAMPLE_200 最大 200 个报告/秒

缺省速率为最大 100 个报告/秒。此请求只可更新流模式下的鼠标采样速率。如果鼠标处于提示模式，则会忽略此请求。

PS2_PROMPTMODE 将鼠标置于提示模式。

在提示模式下，鼠标可根据移动或按键更新其内部值，但只有在响应 **PS2_REPORT ioctl()** 请求时，才发出报告。不使用 *arg* 参数。

PS2_REPORT 获取处于提示模式下的鼠标的报告。

此请求将轮询鼠标，并获取在 *arg* 参数编址的字符缓冲区中返回的三字节报告。该报告的格式如下：

第 1 字节 按键、符号和溢出的位映射

第 0 位 左键（1 = 按下）
第 1 位 右键（1 = 按下）
第 2 位 中键（1 = 按下）
第 3 位 始终为 1
第 4 位 X 数据符号（1 = 负）
第 5 位 Y 数据符号（1 = 负）
第 6 位 X 数据溢出（1 = 溢出）
第 7 位 Y 数据溢出（1 = 溢出）

第 2 字节 横坐标数据字节

第 3 字节 纵坐标数据字节

以 2 的补数表示横坐标和纵坐标的值。通过 **PS2_2TO1_SCALING ioctl()** 请求指定的比例行为不适用于使用 **PS2_REPORT ioctl()** 请求获取的报告。**PS2_2TO1_SCALING** 只影响在流模式下发送的报告。

PS2_STREAMMODE 将鼠标置于流模式。

当鼠标处于流模式下时，只要从上一个报告后，移动了鼠标，或者按下或释放了鼠标按键，该鼠标就会发送一个三字节报告。可使用 **PS2_SAMPLERATE ioctl()** 请求设置最大报告速率。如果在某个采样间隔内按下鼠标按键，随即又松开按键，则报告为在该间隔结束时按下了该按键。

可通过 **read()** 系统调用（请参阅 *read*(2) ）获取流模式报告。该报告的格式与上述 **PS2_REPORT ioctl()** 请求返回的报告相同。

当处于流模式下时，必须在发送 **PS2_DISABLE** 请求后才能发送其他任何 **ioctl()** 请求。

不使用 *arg* 参数。

PS2_STATUS 获取鼠标状态。

此请求将轮询鼠标，并获取在 *arg* 参数编址的字符缓冲区中返回的三字节报告。

状态报告的格式如下：

第 1 字节　按键和鼠标内部状态的位映射

第 0 位	右键（1 = 按下）
第 1 位	中键（1 = 按下）
第 2 位	左键（1 = 按下）
第 3 位	始终为 0
第 4 位	如果为 0，则比例为 1:1；如果为 1，则比例为 2:1
第 5 位	如果为 0，则禁用；如果为 1，则启用
第 6 位	如果为 0，则采用流模式；如果为 1，则采用提示模式
第 7 位	始终为 0

第 2 字节　当前的分辨率设置

第 3 字节　当前的采样速率

PS2_RESOLUTION 可通过指定作为 *arg* 编址的字符缓冲区的第一个字节进行传递的值，为横坐标值和纵坐标值设置鼠标分辨率。支持四个离散分辨率：

分辨率	**200 DPI**	**320 DPI**
RES_1	1 次/mm	1 次/mm
RES_2	2 次/mm	3 次/mm
RES_3	4 次/mm	6 次/mm
RES_4	8 次/mm	12 次/mm

PS2_2TO1_SCALING 将鼠标比例设置为 2 比 1。在流模式报告中返回的横坐标值和纵坐标值将会加倍，但是，小于六的绝对值将被转换为非线性形式的新值。下表详细说明了此转换：

鼠标内部值	转换值
0	0
+|– 1	+|– 1
+|– 2	+|– 1
+|– 3	+|– 3
+|– 4	+|– 6
+|– 5	+|– 9
All other *n*	2 * *n*

此转换不适用于通过 **PS2_REPORT ioctl()** 请求获取的报告。

不使用 *arg* 参数。

PS2_1TO1_SCALING 将鼠标比例设置为 1 比 1。

不为在鼠标报告中返回的 X 轴值和 Y 轴值指定比例。此请求不使用 *arg* 参数。

错误

如果某个系统调用失败，则根据上面“说明”一节中的说明，将 **errno** 设置为下列值之一:

[EBUSY] 指定的 PS/2 设备已打开。

[EFAULT] 尝试使用系统调用的参数时，检测到一个错误的地址。

[EINTR] 一个信号中断了 **open()** 、 **read()** 或 **ioctl()** 系统调用。

[EINVAL] **ioctl()** 检测到无效的参数。

[EIO] 执行 **ioctl()** 系统调用时出现硬件或软件错误。

[ENODEV] PS/2 设备没有实现 **write()** 。

[ENXIO] 指定的地址上不存在设备。

举例

假定 *fildes* 是已连接到键盘的 **ps2** 端口的有效文件描述符。第一个示例将使键盘指示灯闪烁，选择扫描代码集 3，并在输出键代码时无限循环。

```
#include <sys/ps2io.h>

unsigned char kbdbuf[4];  /* buffer for ioctl operations */
unsigned char inchar;     /* keycode read */

/* flash the LED indicators */
kbdbuf[0] = CAPS_LED | SCROLL_LED | NUM_LED;   /* all on */
if( ioctl( fildes, PS2_INDICATORS, &kbdbuf) < 0){
  perror("ioctl PS2_INDICATORS failed");
  exit(1);
}
printf("Indicators on\n");
sleep(1);

kbdbuf[0] = NONE_LED;  /* all off */
if( ioctl( fildes, PS2_INDICATORS, &kbdbuf) < 0){
  perror("ioctl PS2_INDICATORS failed");
  exit(1);
```

```
}
printf("Indicators off\n");

/* use scancode set 3 */
kbdbuf[0] = SCANCODE_3;
if( ioctl( fildes, PS2_SCANCODE, &kbdbuf) < 0){
  perror("ioctl PS2_SCANCODE failed");
  exit(1);
}

/* identify our scancode set */
kbdbuf[0] = GET_SCANCODE;
if( ioctl( fildes, PS2_SCANCODE, &kbdbuf) < 0){
  perror("ioctl PS2_SCANCODE failed");
  exit(1);
}
printf("Keyboard reports it is using scancode set %d\n",
     (unsigned int) kbdbuf[0]);

/* now, loop forever while printing keycodes */
while( 1){
    read( fildes, &inchar, 1);
    printf("Keycode: %x\n", (unsigned int)inchar);
}
```

以下示例将鼠标置于流模式，将报告限制设置为每秒 80，启用鼠标，并在输出鼠标报告时无限循环。假定 *fildes* 是已连接到鼠标的 **ps2** 端口的有效文件描述符。

```
#include <sys/ps2io.h>

unsigned char buf[3];       /* mouse report buffer */
unsigned char ioctl_buf[4]; /* mouse ioctl buffer */

/* first, disable the mouse */
if (ioctl( fildes, PS2_DISABLE) < 0){
  perror("ioctl PS2_DISABLE failed\n");
  exit(1);
}
printf("Mouse disabled\n");
```

```
/* Put mouse in stream mode */
if (ioctl( fildes, PS2_STREAMMODE) < 0){
  perror("ioctl PS2_STREAMMODE failed\n");
  exit(1);
}
printf("Mouse in stream mode\n");

/* set samplerate */
ioctl_buf[0] = SAMPLE_80;
if (ioctl( fildes, PS2_SAMPLERATE, ioctl_buf) < 0){
  perror("ioctl PS2_SAMPLERATE failed\n");
  exit(1);
}
printf("Mouse sample rate set to SAMPLE_80\n");

/* Enable mouse */
if (ioctl( fildes, PS2_ENABLE) < 0){
  perror("ioctl PS2_ENABLE failed\n");
  exit(1);
}
printf("Mouse enabled.\n");

for (;;) {
  if (read(fildes, &buf[0], 1) != 1){
    perror("Read of report byte 1 failed");
    return 1;
  }
  if (read(fildes, &buf[1], 1) != 1){
    perror("Read of report byte 2 failed");
    return 1;
  }
  if (read(fildes, &buf[3], 1) != 1){
    perror("Read of report byte 3 failed");
    return 1;
  }
  printf("mouse: 0x%02x, %d %d\n", buf[0], buf[1], buf[2]);
}
```

作者

ps2 由 HP 开发。

PS/2 和 Personal System/2 是 International Business Machines, Incorporated 在美国和其他国家/地区的注册商标。

文件

/usr/include/sys/ps2io.h
/dev/ps2_[0-15]
/dev/ps2_*.[0-15]
/dev/ps2mouse
/dev/ps2kbd

另请参阅

close(2)、 errno(2)、 fcntl(2)、 ioctl(2)、 open(2)、 read(2)、 select(2)、 signal(5)、 termio(7)。

《SoftPC User's Guide》

《SoftPC Installation Guide》

《Sun System Administrators Guide for the HP700/RX》

名称

ptem - STREAMS pty（伪终端）模拟模块

概要

#include <sys/stropts.h>

int ioctl(fd_slave, I_PUSH, "ptem");

说明

ptem 是一个 STREAMS 模块，当与 **ldterm** （STREAMS 线路规则）和 **pts** （STREAMS 从属 pty 驱动程序）一起使用时，它模拟终端。 **ptem** 模块通常位于 **pts** 之上、 **ldterm** 之下。在压入 **ldterm** 之前，用户进程必须调用 STREAMS **I_PUSH** *ioctl*(2) 系统调用，将 **ptem** 模块压入 pty 的从属端。 **ptem** 负责处理从 **ldterm** 或 **ptm** （STREAMS pty 主驱动程序）下行传递的所有终端 **ioctl** 命令。

ldterm 和 **ptem** 一起为 STREAMS pty 从属端提供实际的终端行为。但是，某些终端 **ioctl** 命令将被忽略并仅促成对命令的确认，这是因为在 pty 子系统中没有实际的终端或调制解调器。实际上，除非将波特率设置为 0，否则 **termio** 或 **termios** 结构的 **c_clfag** 字段中的所有标志（分别由 **TCSETA** 或 **TCSETS ioctls** 使用）对 pty 没有任何影响。将波特率设置为 0 具有挂断 pty 连接的作用。同样， **c_iflag** 字段中的奇偶校验标志或延迟标志对 pty 根本没有影响。

综上所述， **ptem** 模板执行下列任务：

- 将处理以下 **ioctls** （如果适用），当在 **ptem** 的写入队列中收到它们时，通过上行发送一条 **M_IOCACK** 消息进行确认：

  TCSETA 、 **TCSETAW** 、 **TCSETAF** 、 **TCSETS** 、 **TCSETSW** 、 **TCSETSF** 、 **TCGETA** 、 **TCGETS** 和 **TCSBRK** 。

- 跟踪 **TIOCSWINSZ** 、 **TIOCGWINSZ** 和 **JWINSIZE ioctls** 所需的窗口大小。

- 在写入队列中收到任何其他 **ioctl** 时， **ptem** 将通过上行发送一条 **M_IOCNAK** 消息做出否定性确认。

- 以下 **ioctls** 在处理之后将由 **ptem** 下行传递：

  TCSETA 、 **TCSETAW** 、 **TCSETAF** 、 **TCSETS** 、 **TCSETSW** 、 **TCSETSF** 、 **TCSBRK** 和 **TIOCSWINSZ** 。

- 如果 **pckt** 模块没有被压入 **ptm** ，并且上述 **ioctls** 到达 pty 主 STREAMS 头部，则在 **ptem** 的读取队列中收到的任何 **M_IOCNAK** 消息将被释放，随后会下行发送一条 **M_IOCNAK** 消息。

- 如果已打开 **ptem** ，并且设置控制终端的所有条件都满足，则会将一条 **M_SETOPTS** 消息（已设置 **SO_ISATTY** 标志）上行发送到 STREAMS 头部以分配控制终端。

- 在读取队列中收到类型为 **TCSBRK** 的 **M_IOCTL** 消息时， **ptem** 将下行发送一条 **M_IOCACK** 消息并上行发送一条 **M_BREAK** 消息。

- 当在写入列队中收到 **ioctl** 消息用以将波特率设置为 0 时（例如，将带有 CBAUD 的 **TCSETA** 设置为 B0）， **ptem** 将上行发送一条 **M_IOCACK** 消息并下行发送一条零长度消息，供 pty 主进程读取。

- 当在读取列队中收到类型为 **TIOCSIGNAL** 的 **M_IOCTL** 消息时， **ptem** 将下行发送一条 **M_IOCACK** 消息并上行发送一条 **M_PCSIG** 消息，并将信号编号设置为 **M_IOCTL** 消息中所用的相同值。
- 当在读取列队中收到类型为 **TIOCREMOTE** 的 **M_IOCTL** 消息时， **ptem** 将下行发送一条 **M_IOCACK** 消息并上行发送一条 **M_CTL** 消息（将 ioc_cmd 设置为 **MC_DO_CANON** 或 **MC_NO_CANON** ），来启用或禁用 **ldterm** 上的输入处理。
- 当在读取或写入列队中收到 **M_DELAY** 消息时， **ptem** 只是忽略消息，不采取任何操作。
- 当在写入列队中收到类型为 **JWINSIZE** 的 **M_IOCTL** 消息时，如果 **ptem** 中 **jwinsize** 结构的值不为 0， **ptem** 将使用 **jwinsize** 结构上行发送一条 **M_IOCACK** 消息。如果该值为 0， **ptem** 将上行发送一条 **M_IOCNAK** 消息。
- 当在写入列队中收到类型为 **TIOCGWINSZ** 的 **M_IOCTL** 消息时，如果 **ptem** 中 **winsize** 结构的值不为 0， **ptem** 将使用 **winsize** 结构上行发送一条 **M_IOCACK** 消息。如果该值为 0， **ptem** 将上行发送一条 **M_IOCNAK** 消息。
- 当在写入列队中收到类型为 **TIOCSWINSZ** 的 **M_IOCTL** 消息时，如果窗口大小已更改， **ptem** 将在 **winsize** 结构中保存收到的消息，并将一条 **M_PCSIG** （将信号编号设置为 **SIGWINCH** ）上行发送到 pty 从属进程。
- 当在读取列队中收到类型为 **TIOCGWINSZ** 的 **M_IOCTL** 消息时，如果 **ptem** 中 **winsize** 结构的值不为 0， **ptem** 将使用 **winsize** 结构下行发送一条 **M_IOCACK** 消息。如果该值为 0， **ptem** 将下行发送一条 **M_IOCNAK** 消息。
- 当在读取列队中收到类型为 **TIOCSWINSZ** 的 **M_IOCTL** 消息时，如果窗口大小已更改， **ptem** 将在 **winsize** 结构中保存收到的消息，并将一条 **M_PCSIG** （将信号编号设置为 **SIGWINCH** ）上行发送到 pty 从属进程。
- 上面未提及的所有其他消息将被传递到下一个模块或驱动程序。

作者

ptem 由 HP 开发。

另请参阅

ioctl(2)、 streamio(7)、 ptm(7)、 pts(7)、 ldterm(7)。

名称

ptm - STREAMS 主 pty（伪终端）驱动程序

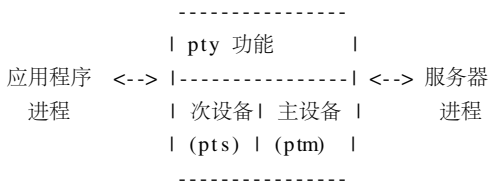
概要

#include <sys/stropts.h>
#include <sys/ptyio.h>
#include <sys/strtio.h>

int open("/dev/ptmx", O_RDWR);

说明

伪终端 (pty) 包括由被称为“主设备”和“次设备”的、紧密耦合的字符设备对组成。 pty 主次设备驱动程序共同协作，可模拟一个终端连接，其中的主设备驱动程序可用于连接伪终端服务器进程，次设备驱动程序可为终端应用程序进程提供终端设备专用文件，如下所述：



压入到顶部（为方便起见而没有显示）的、带有 **ptem** （STREAMS pty 仿真模块）和 **ldterm** （STREAMS 线路规范模块）的从属驱动程序 **pts** 可按 *termio*(7) 中所述提供终端接口。但是，如 *termio*(7) 中所述的提供终端接口的设备，在其后面带有一个硬件设备；与此相反，次设备带有另一个进程，可通过 pty 的主设备端对其进行操作。在主设备上写入的数据将作为输入指定给次设备，而在次设备上写入的数据将作为输入提供给主设备。

要使用 STREAMS pty 子系统，必须安装主 pty 驱动程序 **/dev/ptmx** 的节点，及 *N* 个从属 pty 设备（有关从属 pty 的详细信息，请参阅 *pts*(7) ）。文件系统中没有为单个主设备提供节点。而主驱动程序设置为 STREAMS 克隆驱动程序（请参阅 *clone*(7) ），其主设备号设置为该克隆驱动程序的主设备号，其次设备号设置为 **ptm** 驱动程序的次设备号。将 **/dev/ptmx** 设置为设备文件参数，使用 **open()** 系统调用打开主驱动程序。克隆打开将查找主设备的下一个可用的次设备号。仅当主设备及其对应的次设备尚未打开时，该主设备才可供使用。一个主设备上只允许一个 open 调用，但是一个次设备上可允许多个 open 调用。打开主设备后，将自动锁定对应的次设备（有关如何解锁次设备，及获取次设备名的信息，请参阅 *pts*(7) ）。同时打开主次设备后，将为用户提供两个文件描述符，分别代表由两个流组成的全双工连接的端点。主次设备被打开后，将自动连接这两个流。然后用户可将必要的模块压入到主从流（例如，出于终端语义的需要，将 **ptem** 和 **ldterm,** 压入到 **pts** ；为实现数据包模式功能，将 **pckt** 压入到 **ptm** ）。

主从驱动程序可将所有 STREAMS 消息传递给与它们相邻的驱动程序。只需要对 **M_FLUSH** 消息进行一些特殊的处理，原因是主设备的读取队列已连接到次设备的写入队列，或者情况与此相反。因此，一旦需要将 **M_FLUSH** 消息在主–从链路之间进行传播，就必须将 **FLUSHR** 标志更改为 **FLUSHW** 标志，或者执行与此相反的更改。关闭主设备后，会将一个 **M_HANGUP** 消息发送到对应的次设备，呈现该次设备不可用。当次设备端的进程尝试对该次设备执行 **write()** 系统调用时，将得到 errno [ENXIO] ，但它还能够读取从属流中剩余的任何数

据。最后，当读取了所有数据后， **read()** 系统调用将返回 0（零），指示不再能够使用该次设备。最后一次关闭次设备时，会将一个零长度 **M_DATA** 消息发送到对应的主设备。当主设备端的应用程序发出 **read()** 或 **getmsg()** 系统调用后，将返回 0。由主设备的用户确定是否关闭可分解主设备端上的流的主设备文件。如果主设备保持打开状态，则可以打开对应的次设备，并由另一个用户再次使用。

与次设备不同，主设备不能用作终端。如果设置了 **O_NDELAY** 或 **O_NONBLOCK** ，那么当没有提供数据时，在主设备上执行读取将返回 –1，同时将 errno 设置为 [EAGAIN] ；当流上没有内部流量控制时，在主设备上执行写入将返回 –1，同时将 errno 设置为 [EAGAIN] 。

主 **ptm** 驱动程序支持以下 **ioctl()** 请求：

ISPTM 确定文件描述符是否为打开的主设备的文件描述符。如果成功，它将返回主设备的主次设备号 (type dev_t)，可使用此编号确定对应的次设备的名称。如果失败，它将返回 –1，同时将 errno 设置为 [EINVAL] HP-UX 上的 **ISPTM** 可返回带有负值的有效设备号。例如，如果 STREAMS pty 主设备的主设备号为 0x9c， **ICPTM** 将返回负数 0x9C000000。因此，应用程序必须检查返回值是否为明确的 –1，而不是“< 0”（小于 0）。

ISPTM 由函数 **grantpt()** 、 **unlockpt()** 和 **ptsname()** 使用。通常，用户应用程序不需要调用此读写控制。此读写控制的格式为：

int ioctl(master_fd, ISPTM, 0)

UNLKPT 解锁主设备和对应的次设备。如果成功，它将返回 0。如果失败，它将返回 –1，同时将 errno 设置为 [EINVAL] 。 **UNLKPT** 由函数 **unlockpt()** 使用。通常，用户应用程序不需要调用此读写控制。此读写控制的格式为：

int ioctl(master_fd, UNLKPT, 0)

TIOCREMOTE 此读写控制能使 STREAMS pty 进入和退出远程模式。如果打开了远程模式，输入数据将受到流量控制，并且将在 **ldterm** 中传递，而不必进行任何输入处理，且不管终端模式如何。 pty 主驱动程序收到此读写控制后，会通过 **ptm** 、 **pts** 和 **ptem** 将一个 **M_CTL** 消息向下游发送到 **ldterm** 。 **M_CTL** 消息中的命令将设置为 **MC_NO_CANON** 或 **MC_DO_CANON** ，这取决于是打开远程模式，还是关闭远程模式。此读写控制的格式为：

int ioctl(master_fd, TIOCREMOTE, argument)

如果要打开远程模式，其中的参数将设置为 1；如果要关闭远程模式，则此参数设置为 0。在窗口管理器中执行远程线路编辑时，或者无论何时需要流量受控的输入时，通常需要使用远程模式。每次对主设备进行写入都会为读取次设备的进程生成一个记录边界。在正常使用的情况下，写入数据就类似于在终端上以行的形式键入了该数据；写入 0（零）字节就类似于键入了一个 **EOF** （文件结束）字符。

TIOCSIGNAL 主进程可通过此读写控制将一个信号发送到从属进程。此读写控制的格式为：

int ioctl(master_fd, TIOCSIGNAL, argument)

其中的参数是头文件 **<sys/signal.h>** 中定义的信号编号。例如，主进程可通过执行以下操作，将一个 **SIGINT** 信号发送到从属进程：

ioctl(master_fd, TIOCSIGNAL, SIGINT)

作者

ptm 由 HP 和 OSF 联合开发。

文件

/dev/ptmx	STREAMS pty 主克隆设备
/dev/pts/*N*	STREAMS pty 次设备 (0 <= *N* < **NSTRPTY**)，其中的 **NSTRPTY** 是可以通过 SAM 更改的内核可调参数。

另请参阅

insf(1M)、 getmsg(2)、 ioctl(2)、 open(2)、 read(2)、 write(2)、 grantpt(3C)、 ptsname(3C)、 unlockpt(3C)、 clone(7)、 ldterm(7)、 pckt(7)、 ptem(7)、 pts(7)、 streamio(7)、 termio(7)。

名称

pts - STREAMS 从属 pty（伪终端）驱动程序

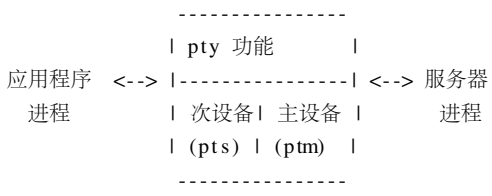
概要

#include <sys/stropts.h>
#include <sys/termios.h>
#include <sys/strtio.h>

int open("/dev/pts/*N*", O_RDWR);

说明

伪终端 (pty) 由紧密耦合的字符设备对（称为主设备和次设备）组成。 pty 主次设备驱动程序一起工作来模拟终端连接，主设备驱动程序提供与伪终端服务器进程的连接，次设备驱动程序提供对终端应用程序进程的终端设备专用文件访问，如下所述:



被压入顶层的、具有 **ptem** （STREAMS pty 模拟模块）和 **ldterm** （STREAMS 线路规则模块）的从属驱动程序 **pts** （为简明起见未显示），提供了如 *termio*(7) 中所述的终端接口。但是， *termio*(7) 中所述的提供终端接口的设备有硬件设备作为后盾；相反，次设备具有通过 pty 的主端对其进行处理的其他进程。写入到主设备中的数据将作为输入提供给次设备，写入到次设备中的数据将作为主设备中的输入。

要使用 STREAMS pty 子系统，必须安装主 pty 驱动程序 **/dev/ptmx** 的节点和 *N* 个从属 pty 设备（有关主 pty 的详细信息，请参阅 *ptm*(7) ）。当打开主设备时，对应的次设备将自动锁定。所有用户都无法打开该次设备，直到更改了其权限（通过 **grantpt()** 函数）并且该设备被解除锁定（通过 **unlockpt()** 函数）。然后，用户调用 **ptsname()** 函数来获取次设备的名称，并调用 **open()** 系统调用来打开次设备。尽管只允许对主设备进行一次打开操作，但允许对次设备进行多次打开操作。主设备和次设备都打开后，用户有两个代表全双工连接端点的文件描述符，该连接由打开主设备和次设备时它们自动连接的两个流组成。然后，用户可以压入所需的模块（例如，对于终端语义而言，是 **pts** 上的 **ptem** 和 **ldterm** ；对于数据包模式功能而言，是 **ptm** 上的 **pckt** ）。

主驱动程序和从驱动程序将所有 STREAMS 消息传递到其临近的驱动程序。只有 **M_FLUSH** 消息需要进行某些特殊处理，这是因为主驱动程序的读取列队连接到从属驱动程序的写入列队，反之亦然。例如，当 **M_FLUSH** 消息通过主–从链接进行传递时， **FLUSHR** 标志会更改为 **FLUSHW** 标志，反之亦然。关闭主设备时，会将一条 **M_HANGUP** 消息发送到对应的次设备，这会使该次设备不可用。当试图对次设备文件进行 **write()** 系统调用，而该文件能够读取从属流中的所有剩余数据时，从属端上的进程将获取错误编号 [ENXIO] 。最后，读取完所有数据后， **read()** 系统调用将返回 0，指示不能再用从属流。最后一次关闭次设备时，会将零长度的 **M_DATA** 消息发送到对应的主设备。当主端上的应用程序发出 *read*(2) 或 *getmsg*(2) 系统调用时，将返回 0（零）。主设备的用户可以决定关闭主设备文件，这会去除主端上的流。如果主设备保持打开状态，则其他用户可以打开并再次使用

对应的次设备。

举例

下面的示例说明了通常如何打开 STREAMS pty 主次设备。

```
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

pts 由 HP 和 OSF 联合开发。

文件

/dev/ptmx	STREAMS pty 主克隆设备
/dev/pts/*N*	STREAMS pty 次设备 (0 <= *N* < **NSTRPTY**)，其中 **NSTRPTY** 是可通过 SAM 更改的内核可调参数（请参阅 *sam*(1M) ）。

另请参阅

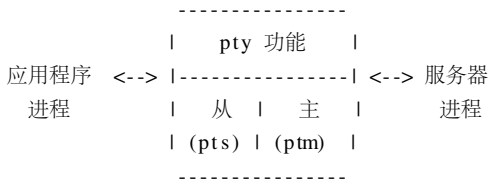
insf(1M)、 sam(1M)、 getmsg(2)、 ioctl(2)、 open(2)、 read(2)、 write(2)、 grantpt(3C)、 ptsname(3C)、 unlockpt(3C)、 ldterm(7)、 ptem(7)、 ptm(7)、 streamio(7)、 termio(7)。

名称

pty - 伪终端驱动程序

说明

pty 驱动程序提供对称为伪终端的设备对的支持。伪终端是一对字符设备：主设备和从设备。从设备为应用程序进程提供与在 *termio*(7) 中所述相同的接口。与提供在 *termio*(7) 中所述接口的所有其他设备不同，从设备在其背后没有硬件设备。相反，它具有通过伪终端的主设备对其进行处理的其他进程。因此，在主设备上写入的任何内容都将作为输入提供给从设备，而在从设备上写入的任何内容都将作为输入显示在主设备上。



打开和关闭处理

pty 的从属端将打开或关闭主端解释为实际终端上的调制解调器连接或断开连接。仅允许打开 **pty** 的主面一次。如果试图打开已打开的主面，则会返回 **-1** ，并将外部变量 **errno** 设置为 [EBUSY] 。如果试图使用打开的文件描述符打开具有从设备的 **pty** 的主面，则会返回 **-1** ，并将 **errno** 设置为 [EBUSY] 。通过使用下一节中讨论的 *clone open* 功能，可以避免在打开时发现 **pty** 繁忙的潜在问题。

如果试图打开不存在的 **pty** ，则会返回 **-1** ，并将 **errno** 设置为 [ENXIO] 。如果未指定 **O_NDELAY** ，则从属面上的打开挂起，直到打开主面为止。如果指定了 **O_NDELAY** ，则在关闭主面时，从属面上的打开将返回错误。关闭主面后在 **pty** 的从属面上发出的任何 **ioctl()** 或 **write()** 请求都将返回 **-1** ，并将外部变量 **errno** 设置为 [EIO]。关闭主面后在 **pty** 的从属面上发出的 **read()** 请求返回 0 字节。关闭 **pty** 的主面会将 **SIGHUP** 挂起信号发送到对应从属面的 tty 进程组编号，并刷新未决的输入和输出。

克隆打开

在典型的 **pty** 用法中， **pty** 对之间没有优先级差异。因此，能够发出在内部打开任何可用 **pty** 的单个 **open()** 是很有用的。 **/dev/ptym/clone** 上的打开会返回可用主 **pty** 设备的打开文件描述符。如果没有可用设备，则打开将返回 **-1** ，并将 **errno** 设置为 [EBUSY] 。通过 **ptsname()** 请求可以找到与打开的主设备对应的从设备的名称。

处理 **ioctl()** 请求

缺省情况下， **pty** 的主面和从属面都可以识别由 *termio*(7) 定义的任何 **ioctl()** 请求。这些 **ioctl()** 请求由 **pty** 驱动程序处理，如 *termio*(7) 指定的那样。此外， **pty** 的主面可识别下面定义的 **ioctl()** 请求。从属面仅识别由 *termio*(7) 定义的 **ioctl()** 请求。关闭主面后在 **pty** 的从属面上发出的 **ioctl()** 请求将返回 **-1** ，并将外部变量 **errno** 设置为 [EIO] 。 **pty** 无法识别的 **ioctl()** 请求返回 **-1** ，并将外部变量 **errno** 设置为 [EINVAL] 。请注意，某些仅限主面的 **ioctl()** 请求影响哪些 **ioctl()** 请求可以由 **pty** 的主面和从属面识别。这些仅限主面的 **ioctl()** 请求还影响 **pty** 驱动程序处理已识别的 **ioctl()** 请求、 **open()** 请求和 **close()** 请求的方式。

下列 **ioctl()** 请求（在 **<sys/ptyio.h>** 中定义）仅适用于 **pty** 的主面：

TIOCSIGSEND 导致将信号从 **pty** 的从属面发送到从属面的当前 tty 进程组。将参数的值作为发送的信号编号。如果指定的信号编号没有引用合法的信号（请参阅 *signal*(5) ），则将返回 [EINVAL] 错误，且不发送信号。请注意，此请求允许服务器进程将信号发送到不是由同一用户 ID 拥有的进程。

TIOCTTY 启用或禁用由 **pty** 进行的所有 **termio** 处理。如果 *arg* 指明的 **int** 不为零，则启用 **termio** 处理；如果 *arg* 指明的 **int** 为零，则禁用该处理。缺省情况下，启用 **termio** 处理。 **termio** 处理是指由 *termio*(7) 所述的输入和输出的处理（如制表符扩展），以及由 *termio*(7) 所述的 **ioctl()** 请求的处理。如果禁用，则使所有输入和输出数据经过 **pty** ，且不进行修改。如果发出 **TIOCTTY ioctl()** 请求，则刷新在伪终端中缓冲的所有数据，并释放被阻塞并等待数据的任何进程。启用和禁用 **TIOCTTY** 将影响下列 **ioctl()** 请求的操作： **TIOCPKT** 、 **TIOCREMOTE** 、 **TIOCBREAK** 、 **TIOCSTOP** 、 **TIOCSTART** 、 **TIOCTRAP** 和 **TIOCMONITOR** 。

如果启用 **TIOCTTY** ，则所有的 **termio ioctl()** 请求如 *termio*(7) 中指定的那样执行，而不管从哪个面发出了 **ioctl()** 请求。如果禁用 **TIOCTTY** ，则主面 **termio ioctl()** 请求将外部变量 **errno** 设置为 [EINVAL] ，并将其返回。如果禁用 **TIOCTTY** ，则像处理任何其他 **ioctl()** 请求那样处理从属面 **termio ioctl()** 请求。尤其是，如果同时禁用 **TIOCTTY** 和 **TIOCTRAP** ，则从属面 **termio ioctl()** 请求将外部变量 **errno** 设置为 [EINVAL] ，并将其返回。（请参阅下面对 **ioctl()** 、 **open()** 和 **close()** 捕获的讨论）。 **TIOCTTY** 的状态不影响不是由 *termio*(7) 定义的 **ioctl()** 请求。

在禁用 **TIOCTTY** 的情况下通过伪终端写入的数据的处理方式与流过管道的数据类似。写入请求将在 **pty** 中被阻塞，直到将所有数据写入 **pty** 。如果没有可用数据，读取请求将被阻塞，除非设置了 **O_NDELAY** 标记（请参阅 *fcntl*(2) ）。当数据可用于读取时，读取请求返回任何可用的数据，且不等待满足请求的字节数。 **pty** 可以在其内部内存中包含的字节数与实现相关，但是在每个方向上至少为 256 字节。例如，对于在 1024 字节 **pty** 的从属面上的写入，在主面上可能由四个读取请求读取，每个请求返回 256 字节。不保证读取的数据块大小是一致的，但是不会丢失数据。

下列 **ioctl()** 请求（在 **<sys/ptyio.h>** 中定义）仅适用于 **pty** 的主面。尤其是，这些 **ioctl()** 请求启用（或禁用） **pty** 驱动程序操作的特定模式。这些 **ioctl()** 请求是与 **TIOCTTY** 相连的；即，模式必须由其 **ioctl()** 请求启用，且必须启用 **TIOCTTY** ，该模式才能起作用。可以启用或禁用模式，而不管 **TIOCTTY** 的状态如何。

TIOCPKT 启用或禁用数据包模式。如果由 *arg* 指明的 **int** 不为零，则启用数据包模式；如果由 *arg* 指明的 **int** 为零，则禁用数据包模式。缺省情况下，禁用数据包模式。如果应用于伪终端的主面，则来自主面的每个后续 **read()** 都将返回在伪终端的从属部分上写入的数据，前置有零字节（以符号形式定义为 **TIOCPKT_DATA** ）或反映控制状态信息的单字节。这样的状态字节的值由零个或更多个位标记组成：

TIOCPKT_FLUSHREAD
已刷新从属面的读取队列。

TIOCPKT_FLUSHWRITE
已刷新从属面的写入队列。

TIOCPKT_STOP
已通过 **^S** 、 **TIOCSTOP** 或 **TCXONC** 停止从 **pty** 的从属面到主面的数据流。

TIOCPKT_START
已重新启动从 **pty** 的从属面到主面的数据流。

TIOCPKT_DOSTOP
停止并启动已设置为 **^S** 或 **^Q** 的字符。

TIOCPKT_NOSTOP
停止并启动设置为除 **^S** 或 **^Q** 之外的内容的字符。

TIOCREMOTE 启用或禁用远程模式。如果 *arg* 的 **int** 值不为零，则启用远程模式；如果 *arg* 的 **int** 值为零，则禁用远程模式。缺省情况下，禁用远程模式。远程模式与数据包模式无关。此模式导致对伪终端的输入进行流控制，但不编辑输入（不管终端模式如何）。对主面的每次写入都将为读取从属面的进程生成记录边界。在正常用法中，写入数据就像将数据作为一行在终端上键入；写入零字节等效于键入文件结束标记字符（即，在 *termio*(7) 中定义的 EOF 字符）。从属面读取的数据与在主面上写入的数据完全相同。在启用 **TIOCREMOTE** 的情况下在从属面上写入的数据和在主面上读取的数据，仍会受到常规的 *termio*(7) 处理。在窗口管理器中执行远程行编辑时，或者每当需要流控制的输入时，可以使用 **TIOCREMOTE** 。发出 **TIOCMONITOR ioctl()** 请求可刷新在伪终端中缓冲的所有数据。

下列 **ioctl()** 请求（在 **<sys/ptyio.h>** 中定义）仅适用于 **pty** 的主面。尤其是，仅当启用 **TIOCTTY** 时才识别这些 **ioctl()** 请求。如果禁用 **TIOCTTY** ，则这些 **ioctl()** 请求将外部变量 **errno** 设置为 [EINVAL] 并将其返回。

TIOCBREAK 导致在 **pty** 的从属面上执行中断操作，好像用户按下了实际终端上的 Break 键。不采用任何参数。

TIOCSTOP 停止从 **pty** 的从属面到主面的数据流（等效于键入 **^S** ）。不采用任何参数。

TIOCSTART 重新启动输出（已由 **TIOCSTOP** 停止或通过键入 **^S** 停止）。不采用任何参数。

流控制输入和输出处理

下列术语用于描述通过伪终端的数据流。 INPUT 是指从 **pty** 的主面到从属面的数据流。 OUTPUT 是指从 **pty** 的从属面到主面的数据流。

如果禁用数据包模式 (**TIOCPKT**) 并停止 INPUT （请参阅 *termio*(7) 中的 IXOFF 输入模式），来自 **pty** 的主面的下一个 **read()** 返回 STOP 字符。重新启动 INPUT 时，来自主面的下一个 **read()** 返回 START 字符。如果启用数据包模式 (**TIOCPKT**) ，则 STOP 或 START 字符前面有一个数据包指示符 (**TIOCPKTDATA**) 。在每个 **write()** 请求正确处理 INPUT 流控制之前， **select()** 应该由主面服务器使用（请参阅 *select*(2) ）。

如果启用 INPUT 流控制，则按如下所述处理 **write()** 和 **select()** ：仅当尚未停止 INPUT 时， **pty** 的主面上的“写入选择”才返回 true。如果在将数据写入 **pty** 的主面时 INPUT 被停止，则写入返回在停止 INPUT 之前写入的字

节数。在停止 INPUT 之后完成的写入立即返回，且写入的字节数为零。

如果禁用数据包模式 (**TIOCPKT**) 并停止 OUTPUT （请参阅 *termio*(7) 中的 IXON 输入模式），则来自 **pty** 的主面的每个后续 **read()** 都在不读取数据的情况下返回。重新启动 OUTPUT 时，来自主面的每个后续 **read()** 都将返回在从属面上写入的数据。如果启用数据包模式 (**TIOCPKT**)，则停止 OUTPUT 之后的第一个 **read()** 返回 **TIOCPKTSTOP** 数据包。停止 OUTPUT 时来自主面的所有后续读取都返回不含数据的 **TIOCPKTDATA** 数据包。重新启动 OUTPUT 时，来自主面的下一个 **read()** 返回 **TIOCPKTSTART** 数据包。来自主面的所有后续读取返回在从属面上写入的数据，其前面有一个 **TIOCPKTDATA** 数据包。在每个 **read()** 正确处理 OUTPUT 流控制之前， **select()** 应该由主面服务器使用。否则，在停止 OUTPUT 时将不阻塞来自 **pty** 的主面的读取。

捕获 **ioctl()**、**open()**、**close()** 请求

如果启用捕获，则其从属面上的应用程序发出 **ioctl()** 、 **open()** 或 **close()** 请求时将通知主面。对于捕获的 **ioctl()** 和 **open()** 请求，将阻塞从属面（即请求不完成），直到其主面上的服务器确认捕获的请求为止。对于捕获的 **close()** 请求，不因未进行确认而阻塞从属面。

select() 应该由主面服务器使用，以接收捕获的 **ioctl()** 、 **open()** 和 **close()** 请求的通知。当捕获这些请求之一时，**select()** 返回为从属面的文件描述符指示的“异常情况”。用于接收已捕获请求的通知的其他机制在下面定义，但是仅当 **select()** 不可用时才应该使用这些机制。

如果禁用捕获（缺省情况），则无法识别的从属 **ioctl()** 请求返回错误，并将外部变量 **errno** 设置为 [EINVAL] 。从属端仅可以识别那些由 *termio*(7) 定义的 **ioctl()** 请求，而且仅当启用 **TIOCTTY** 时才识别。如果禁用 **TIOCTTY** ，则从属面无法识别 **ioctl()** 请求。如果启用捕获且主面关闭，则禁用捕获。如果主面在与从属面握手的中间关闭，则握手自动完成。

捕获以启用或禁用它们的 **ioctl()** 请求标识的两种形式出现 — **TIOCTRAP** 和 **TIOCMONITOR** 。这两种形式可以根据它们所影响的请求类型和它们所提供的功能进行区分。捕获 **open()** 和 **close()** 请求由 **TIOCTRAP** 启用或禁用。捕获未由 *termio*(7) 定义的 **ioctl()** 请求由 **TIOCTRAP** 启用或禁用。仅当还禁用了 **TIOCTTY** 时，捕获由 *termio*(7) 定义的 **ioctl()** 请求才由 **TIOCTRAP** 启用或禁用。如果启用 **TIOCTTY** ，则捕获由 *termio*(7) 定义的 **ioctl()** 请求由 **TIOCMONITOR** 启用或禁用。简单地说， **TIOCTRAP** 和 **TIOCMONITOR** 捕获都允许主面上的服务器检查请求的参数、发出请求的 pid 等。此外， **TIOCTRAP** 捕获允许服务器修改参数和返回 **ioctl()** 请求的值。

下列 **ioctl()** 调用仅适用于 **pty** 的主面，且与捕获 **ioctl()** 、 **open()** 和 **close()** 请求有关。它们在 **<sys/ptyio.h>** 中进行了定义：

TIOCTRAP 启用或禁用在 **pty** 的从属面上捕获应用程序发出的 **ioctl()** 、 **open()** 和 **close()** 请求。如果由 *arg* 指明的 **int** 不为零，则启用捕获；如果 *arg* 指明的 **int** 为零，则禁用捕获。缺省情况下，禁用 **TIOCTRAP** 捕获。

TIOCTRAPSTATUS

检查未决的 **ioctl()** 、 **open()** 或 **close()** 陷阱。如果陷阱是未决的，则该参数指向设置为一的 **int**，如果没有未决的陷阱，则该参数指向设置为零的 **int** 。当 **select()** “异常情况”的首选方法不可用时，使用 **TIOCTRAPSTATUS** 。

TIOCREQCHECK

将捕获的 **ioctl()** 、 **open()** 或 **close()** 信息返回到主面。使用 **TIOCREQCHECK** 来响应 **select()** “异常情况”或指示陷阱未决的 **TIOCTRAPSTATUS** 。 **TIOCREQCHECK** 将未决的 **ioctl()** 、 **open()** 或 **close()** 信息读取到由 **TIOCREQCHECK** 的 *arg* 指向的内存中。信息采用在 **<sys/ptyio.h>** 中定义的以下 **request_info** 结构的形式:

```
struct request_info {
   int request;
   int argget;
   int argset;
   pid_t pgrp;
   pid_t pid;
   int errno_error;
   int return_value;
};
```

request_info 的所有元素都是指 **pty** 的从属面，其中包括:

request 收到的 **ioctl()** 命令。

argget 应用于主面以接收捕获的 **ioctl()** 结构（如果存在一个）的 **ioctl()** 请求（零值表示没有这样的结构）（如果不为零，则 **argget** 是已预先计算大小字段的 **TIOCARGGET** 请求）。

argset 应用于主面以发回得到的 **ioctl()** 结构（如果存在一个）的 **ioctl()** 请求（零值表示没有这样的结构）（如果不为零，则 **argset** 是已预先计算大小字段的 **TIOCARGSET** 请求）。

pgrp 执行操作的进程的进程组编号。

pid 执行操作的进程的进程 ID。

errno_error 由 **ioctl()** 在从属面上返回的 **errno** 外部变量错误代码（已初始化为零）。如果启用打开错误模式，则可以使用 **errno_error** 为捕获的从属 **pty open()** 请求返回错误。有关打开错误模式的进一步信息，请参阅对 **TIOCSMODES ioctl()** 的讨论。

return_value 未设置 **errno_error** 时由 **ioctl()** 在从属面上返回的成功值（已初始化为零）。

如果在从属面上接收的 **ioctl()** 参数不是指针，则其值存储为四个字节，可使用等于 **argget** 的对主面的 **ioctl()** 请求进行检索。

传递 **open()** 或 **close()** 时， **request** 被分别设置为 **TIOCOPEN** 或 **TIOCCLOSE** 。对于 **TIOCOPEN** 和 **TIOCCLOSE** ， **argget** 和 **argset** 都为零，因为没有 **ioctl()** 结构。如果启用 **TIOCTTY** ，则在将 *termio*(7) 对打开/关闭的定

义传递到主面之前首先执行该定义。请注意，在捕获所有打开时， **pty** 仅捕获 **pty** 从属面的特定 i 节点上的最后关闭。

如果没有 **ioctl()** 、 **open()** 或 **close()** 陷阱是未决的，则 **TIOCREQCHECK** 返回外部变量 **errno** 错误 [EINVAL] 。因此，响应 **select()** “异常情况”而返回 [EINVAL] 的 **TIOCREQCHECK** 指示：在 **select()** 返回后，捕获的 **ioctl()** 、 **open()** 或 **close()** 请求被信号终止。

TIOCREQGET 除没有 **ioctl()** 、 **open()** 或 **close()** 陷阱为未决的情况外，与 **TIOCREQCHECK** 相同。 **TIOCREQGET** 将被阻塞，直到捕获从属面 **ioctl()** 、 **open()** 或 **close()** 为止；而 **TIOCREQCHECK** 返回 [EINVAL] 。当 **select()** “异常情况”的首选方法和主面 **ioctl() TIOCTRAPSTATUS** 都不可用时，使用 **TIOCREQGET** 。

TIOCREQSET 完成由以前的 **TIOCREQCHECK** 或 **TIOCREQGET** 启动的握手。该参数应该指向 **request_info** 结构，如 **TIOCREQCHECK** 定义的那样。

执行此 **ioctl()** 请求以完成握手之前，服务器应该将 **errno_error** 设置为要传递回从属面的外部变量 **errno** 错误值。如果没有错误，则可以不理会 **errno_error** ，因为 **pty** 将它初始化为零。此外，在没有错误时，如果需要除零之外的结果，则应该设置 **return_value** 。如果捕获的请求是 **ioctl()** ，则服务器可以设置 **return_value** 和 **errno_error** ；如果启用了打开错误模式，则服务器可以为捕获的 **open()** 设置 **errno_error** 。为捕获的 **close()** 设置 **return_value** 或 **errno_error** 既不影响请求的返回值，也不影响从属面的外部变量 **errno** 值。除非启用打开错误模式，否则为捕获的 **open()** 设置 **return_value** 或 **errno_error** 既不影响请求的返回值，也不影响从属面的外部变量 **errno** 值。打开错误模式允许服务器通过设置 **errno_error** 将错误返回到捕获的从属 **open()**。与 **ioctl()** 请求不同，设置 **return_value** 从不影响从属 **pty open()** 请求。此外，设置 **return_value** 或 **errno_error** 不会导致 **TIOCREQSET** 将错误返回到服务器。

如果发出了 **TIOCREQSET** 请求，而且已传递 **request_info** 结构中的请求值与捕获的值不相等，则设置外部变量 **errno** ，并返回为 [EINVAL] 。如果没有捕获的 **ioctl()** 、 **open()** 或 **close()** 请求，则也返回 [EINVAL] 。如果在服务器执行 **TIOCREQGET** 和 **TIOCREQSET** 的时间之间捕获的请求已被信号中断，则 **TIOCREQSET** 请求返回 [EINVAL] 。

TIOCGFLAGS 获取与捕获的请求关联的文件状态标记。在成功返回时， **ioctl()** 在 *arg* 引用的整数中返回所捕获请求的文件状态标记。可以使用 **<sys/file.h>** 中的标记定义解释标记。如果没有陷阱当前是未决的，则 **TIOCGFLAGS ioctl()** 返回错误，并将外部变量 **errno** 设置为 [EINVAL] 。

TIOCMONITOR

启用或禁用 **termio ioctl()** 请求的只读捕获。如果 *arg* 指明的 **int** 不为零，则启用 **TIOCMONITOR** 捕获；如果 *arg* 指明的 **int** 为零，则禁用该捕获。缺省情况下，禁用 **TIOCMONITOR** 捕获。 **TIOCMONITOR** 是与 **TIOCTTY** 相连的；即，必须启用 **TIOCMONITOR** 捕获且必须启用 **TIOCTTY** ，才能由 **TIOCMONITOR** 捕获 **termio ioctl()** 请求。可以启用或禁用 **TIOCMONITOR** 捕获，而不管 **TIOCTTY** 的状态如何。

如果禁用 **TIOCTTY** ，则 **TIOCMONITOR** 不捕获 **termio ioctl()** 请求。但是，如果禁用 **TIOCTTY** 且启用 **TIOCTRAP** ，则 **ioctl()** 请求由 **TIOCTRAP** 捕获。 **TIOCTRAP** 捕获允许主面服务器修改参数和返回 **ioctl()** 请求的值，而 **TIOCTMONITOR** 捕获则不允许。

TIOCMONITOR 捕获允许主面上的服务器知道 **pty** 中线路规则的特性何时被其从属面上的应用程序更改。 **TIOCMONITOR** 捕获的 **termio** 请求握手的机制，与上面所述的由 **TIOCTRAP** 捕获的请求机制相同（建议在主面上使用 **termio ioctl()** 请求，以查询 **pty** 中线路规则的配置状态。这补偿了未捕获 **termio ioctl()** 请求时启用 **TIOCMONITOR** 之前的时间窗口）。

在 **pty** 的主面上使用 **select()** 时，“异常情况”是指从属面上未决的 **open()** 、 **close()** 或 **ioctl()** 请求，而“可以读取或写入”指示设备可以成功读取或写入了。

在受到捕获的 **ioctl()** 请求中，一次只能处理每个 pty 的一个请求。也就是说，当应用程序向从属面发出 非**termio ioctl()** 请求时，将阻塞对同一 **pty** 从属面的所有其他 **ioctl()** 请求，直到主面重新与第一个请求握手为止（如果启用 **TIOCTTY** 且禁用 **TIOCMONITOR** ，则不阻塞未捕获的 ioctl() 请求，如 **termio** ）。这允许通过从属面上传递到服务器进程的 **ioctl()** 调用实现不可分割的操作。

概括地说，处理捕获的 **ioctl()** 、 **open()** 和 **close()** 请求的以下方法是首选的：

1. 调用 **select()** 。此系统调用阻塞主面，直到捕获从属面 **ioctl()** 、 **open()** 或 **close()** 请求为止。
2. 发出 **TIOCREQCHECK ioctl()** 请求。此步骤返回有关捕获的 **ioctl()** 、 **open()** 或 **close()** 请求的信息。如果 **TIOCREQCHECK** 返回外部变量 **errno** 错误 [EINVAL], , 则循环回 **select()** 调用。
3. 发出 **argget ioctl()** 请求。如果 **argget** 不为零，且服务器希望执行比仅拒绝捕获的从属 **ioctl()** 请求更多的操作，则使用此可选步骤。
4. 发出 **argset ioctl()** 请求。如果 **argset** 不为零，且服务器希望传递回已修改的 **ioctl()** 结构，则执行此可选步骤。它是在通过主面上的服务器处理捕获的 **ioctl()** 请求之后执行的。
5. 设置 **errno_error** 和 **return_value** 。如果捕获的请求为 **ioctl()** ，则相应地设置 **errno_error** 。如果 **errno_error** 的相应值为零，则必须设置 **return_value** 。如果启用了打开错误模式，则将 **errno_error** 设置为非零值，以便将错误返回到捕获的 **open()** 请求。
6. 发出 **TIOCREQSET ioctl()** 请求。此步骤完成捕获的 **ioctl()** 、 **open()** 或 **close()** 请求。

当进程在 **pty** 的从属面中等待服务器完成握手时，它易受接收信号的影响。以下主面 **ioctl()** 请求允许服务器进程控制信号试图中断捕获的 **open()** 或 **ioctl()** 请求时 **pty** 响应的方式：

TIOCSIGMODE

将 **pty** 的信号处理状态设置为以参数形式指定的模式。该模式可以具有三个值，即 **TIOCSIGBLOCK** 、 **TIOCSIGABORT** 和 **TIOCSIGNORMAL** 。

TIOCSIGBLOCK

导致将发往捕获了其 **open()** 或 **ioctl()** 请求的从属面进程的某些信号延迟。如果不延迟信号会导致进程跳转到已安装的信号处理程序，则延迟这些信号。如果信号被延迟会导致进程异常中止或者它们被忽略，则不延迟这些信号。当服务器进程通过 **TIOCREQSET ioctl()**

请求完成握手时，进程返回到调用程序，然后处理任何未决的信号。用户已通过 **sigblock()** 阻塞的任何信号将继续被阻塞。

TIOCSIGABORT
防止重新启动已捕获的 **open()** 或 **ioctl()** 请求。当服务器进程希望中断的请求为调用程序返回 [EINTR] 错误时，它设置此模式。

TIOCSIGNORMAL
这是 **pty** 的缺省模式。如果信号中断了捕获的 **open()** 或 **ioctl()** 请求，则用户的信号处理程序例行程序可以指定是否重新启动该请求。如果重新启动该请求，则它再次从头开始执行，而且服务器必须发出另一 **TIOCREQGET** 请求以再次启动握手。如果用户的信号处理程序例行程序指定不应该重新启动已中断的请求，则在信号处理程序完成时该请求为调用程序返回 [EINTR] 。请注意，重新启动的请求不一定就是要捕获的下一个请求。

在 **<sys/ptyio.h>** 中定义的下列 **ioctl()** 请求提供了获取和设置 **pty** 模式的机制。使用前面讨论的其他 **ioctl()** 请求也可以处理其中的五种模式。请参阅 **ioctl()** 等效项的位定义。通过任一方式启用或禁用它们的效果是相同的。通常，应用程序将使用 **TIOCGMODES ioctl()** 获取当前有效的 **pty** 模式，设置或清除被更改的模式的位，以及发出 **TIOCGMODES ioctl()** 以实现所需的更改。

TIOCGMODES
获取当前有效的 **pty** 模式。 **ioctl()** 在由 *arg* 引用的 long 型值中返回若干位，以指示各种 **pty** 模式的状态。如果设置了位，则启用关联模式。如果清除了位，则禁用关联模式。清除未使用的位。各个位的含义在 **TIOCSMODES ioctl()** 的说明下介绍。

TIOCSMODES 根据 *arg* 引用的 long 型值设置 **pty** 模式。忽略未使用的位，但应该将其设置为零。下面列出了 **pty** 模式的位值。

PM_REMOTE
启用或禁用远程模式。请参阅对 **TIOCREMOTE ioctl()** 的讨论。

PM_TTY
启用或禁用 tty 模式。请参阅对 **TIOCTTY ioctl()** 的讨论。

PM_PKT
启用或禁用数据包模式。请参阅对 **TIOCPKT ioctl()** 的讨论。

PM_TRAP
启用或禁用陷阱模式。请参阅对 **TIOCTRAP ioctl()** 的讨论。

PM_MONITOR
启用或禁用监视器模式。请参阅对 **TIOCMONITOR ioctl()** 的讨论。

PM_OPEN_ERROR
启用或禁用打开错误模式。打开错误模式允许服务器进程通过 **TIOCREQSET ioctl()** 将错误返回到捕获的从属 **pty open()** 。如果启用打开错误模式，则通过在传递到 TIOCREQSET **ioctl()**

request_info 的结构中设置 **errno_error** 字段，服务器可以返回捕获的 **open()** 及错误。如果禁用打开错误模式（缺省状态），则将 **errno_error** 设置为与从属 **open()** 握手不起作用。请注意，与 **ioctl()** 陷阱握手不同，设置 **return_value** 对从属 **open()** 不起作用，而不管打开错误模式的状态如何。有关与捕获的请求握手的进一步详细信息，请参阅对 **TIOCREQSET ioctl()** 的讨论。

警告

从属面无法向主面指示文件结束情况。

使用 **TIOCREMOTE** 时，向主面发出的大于 256 字节的单个 **write()** 请求可能会导致自从属面读取多个更小的记录，而不是仅读取一个记录。

作者

pty 由加州大学伯克利分校开发。

文件

/dev/ptym/pty[**a-ce-su-z**][**0-9**][**0-9**]	主伪终端
/dev/ptym/pty[**a-ce-su-z**][**0-9**][**0-9**][**0-9**]	主伪终端
/dev/ptym/pty[**a-ce-su-z**][**0-9a-f**]	主伪终端
/dev/pty[**pqr**][**0-9a-f**]	主伪终端
/dev/pty/tty[**a-ce-su-z**][**0-9**][**0-9**]	从属伪终端
/dev/pty/tty[**a-ce-su-z**][**0-9**][**0-9**][**0-9**]	从属伪终端
/dev/pty/tty[**a-ce-su-z**][**0-9a-f**]	从属伪终端
/dev/tty[**pqr**][**0-9a-f**]	从属伪终端

另请参阅

close(2)、 fcntl(2)、 ioctl(2)、 open(2)、 read(2)、 select(2)、 sigblock(2)、 write(2)、 ptsname(3C)、 signal(5)、 termio(7)。

名称

random、urandom、rng - 功能强大的随机数生成器

概要

#include <sys/random.h>

说明

字符设备专用文件 **/dev/random** 和 **/dev/urandom** 提供了驻留在内核的随机数生成器 **rng** 的接口。 **/dev/random** 的 **read()** 可能是阻塞的。 **/dev/urandom** 的 **read()** 始终是非阻塞的。 **/dev/urandom** 的数据信息量可能低于 **/dev/random** 的数据信息量。

rng 模块是动态可加载内核模块 (DLKM)。即，具有超级用户权限的管理员可以动态地对其取消配置或重新配置，而无须重新引导系统。

rng 序列具有无限的信息量。相反，由伪随机数生成器（例如 *random*(3M) ）计算生成的序列则具有有限的信息量，仅派生自其初始种子。 **rng** 模块应视为随机性的优良来源。它已通过广泛的统计测试，包括 NIST （美国国家标准和技术研究院）随机性测试。

rng 模块使用由外部事件触发的中断线程的完成时间的不确定性。 **rng** 模块从中断时间戳内提取一系列位。将删除任何现有位偏差，以生成具有 0 和 1 统一分布的序列。所得序列分给了专用文件 **/dev/random** 和 **/dev/urandom** 的对应保留缓冲区。对于 **/dev/random** 和 **/dev/urandom** 的每个 **read()** ，将从对应的保留缓冲区中检索数据。将应用基于 AES（高级加密标准）的散列函数，并将结果放置在用户提供的缓冲区内。将对保留缓冲区内的所有请求进行串行化处理，以确保返回的随机数据不在不同的请求之间共享，即便是多处理器系统上同时发生的请求也是如此。

没有与 **/dev/random** 或 **/dev/urandom** 相关联的 **write()** 函数，所有用户仅能读取这两个设备。为 **/dev/random** 定义了单个 **ioctl()** ，用来进行独立的 **rng** 产品验证。

文件 **/usr/include/sys/random.h** 包含下列定义：

```
/* The maximum request size, for read() or ioctl(), in bytes   */
#define RNG_READMAX    256


/* ioctl() to retrieve data from the entropy collector directly*/
#define RNG_GETRAW     _IOR('Q', 0, uint8_t[RNG_READMAX])
```

如果 **read()** 请求的数据量超过 RNG_READMAX 字节数，则可将它视为刚好为 **RNG_READMAX** 字节。这适用于 **/dev/random** 和 **/dev/urandom** 。

有关 /dev/random 的特定信息

如果在很短的时间间隔内出现大量针对于 **/dev/random** 的请求，则保留缓冲区上的需求会超过 **rng** 提供数据的速率。如果存储在保留缓冲区中的随机数据太少而无法完成请求，则 **/dev/random** 设备上的 **read()** 将阻塞请求线程。在使用足够的随机数据更新保留缓冲区以完成请求之前，线程将保持阻塞状态。

对于 **/dev/random open()** 标记，仅 **O_NONBLOCK** 和 **O_NDELAY** 具有设备特定的操作。如果这两个标记均未设置，则 **/dev/random** 上的 **read()** 将阻塞请求的线程，直到可以返回多达 **RNG_READMAX** 字节的请求数据

量。当请求的字节数不可用，并且设置了上述任一标记时，则立即返回 **read()** 。如果设置了 **O_NONBLOCK** 标记， **read()** 将返回 -1，并且 errno 将设置为 **EAGAIN** 。如果未设置 **O_NONBLOCK** 但设置了 **O_NDELAY** ， **read()** 将返回 0。

在删除偏差之后但在 AES 散列之前， **RNG_GETRAW ioctl()** 允许具有超级用户权限的应用程序从 **/dev/random** 保留缓冲区中直接提取 **RNG_READMAX** 字节的数据。该接口不适用于加密应用程序，而适用于 **/dev/random** 保留缓冲区中数据随机性的统计测试。出于同样的原因， **RNG_GETRAW ioctl()** 采用与 **/dev/random** 上 read 命令的相同方式阻塞请求的线程。如果请求线程没有超级用户权限，将返回 **EACCES** 。

有关 /dev/urandom 的特定信息

要解决受限制的随机数据收集速率问题， **/dev/urandom** 设备应是非阻塞的。 **/dev/urandom** 保留缓冲区是使用随机数据定期更新的，但是大量的读取会减少保留缓冲区中的消息量。在这种情况下， **/dev/urandom** 的数据信息量将稍低于 **/dev/random** 的数据信息量，但是 **/dev/urandom** 仍可视为随机数的较好来源。

对于 **/dev/urandom read()** ，没有导致设备特定操作的 **open()** 标记。

错误

[EAGAIN] 对于 **/dev/random read()** ，当打开 **/dev/random** 时，将设置 **O_NONBLOCK** ，并且保留缓冲区中的内容不足以完成请求。

[EACCES] 对于 **/dev/random RNG_GETRAW ioctl()** ，请求线程没有超级用户权限。

作者

随机数生成器由 HP 开发。

对于偏差消除，生成器使用加州大学 Dr.Yuval Perez 开发的算法。

安全散列使用英国的 Dr.Brian Gladman 提供的 AES 实现。

NIST 统计测试可在以下网站上找到： **http://csrc.nist.gov/rng** 。

文件

/dev/random

/dev/urandom

另请参阅

random(3M)。

名称

route - 内核数据包转发数据库

概要

#include <sys/types.h>
#include <sys/socket.h>
#include <net/route.h>
#include <net/if.h>

s = socket(AF_ROUTE, SOCK_RAW, *family*);

说明

此联机帮助页说明了用来读取和写入内核路由消息的路由套接字接口。

有关如何传输网络数据包的信息由路由信息数据库中的 HP-UX 内核维护，路由信息数据库也称为路由表。通过利用 AF_ROUTE 套接字向内核发送路由消息，用户进程可以读取或更新路由表中的信息。有关消息类型的详细描述，请参阅下面的“消息类型”一节。

“概要”所说明的 套接字系统调用中，*family* 参数可用于过滤调用方接收的路由消息。*family* 的有效值包括：

AF_INET	获取影响 Internet 协议的路由消息。
AF_INET6	获取影响 Internet 协议第 6 版的路由消息。
AF_UNSPEC	获取同时影响 AF_INET 和 AF_INET6 协议的路由消息。

路由表中的条目指定传输数据包时要使用的相应远程主机或网关。这些条目要么是主机特定的，要么适用于位于通用子网中的所有主机，具体由网络掩码值指定。

系统引导之后，每个协议系列都会向配置的每个网络接口的路由表添加条目，并且准备传输网络流量。正常情况下，路由条目指定为到目标主机或网络的 “直接连接”。对于直接路由，网络堆栈的传输层会将数据包直接发送给数据包头中指定的主机。对于非直接路由，接口会将数据包转发给该接口的路由条目中列出的网关。

路由数据包时，内核会尝试为每个目标找到一种最佳的路由。如果多个条目与目标的网络掩码相匹配，则内核会在网络掩码中选择具有更多 1 的路由。

如果无法定位至到某个特定远程主机或网络的其他路由，则会使用缺省（通配符）路由。缺省路由是通过全部为 0 的目标地址值和全部为 0 的网络掩码指定的。缺省路由与路由重定向一起为路由网络流量提供了一种节约开销的机制。

如果找不到路由条目，则该目标会声明为无法访问，并且会对使用路由套接字工具的所有用户进程都生成一条路由缺失消息- (RTM_MISS)，如下所述。

消息类型

创建了路由套接字之后，进程可以通过写入套接字的方式向内核发送命令。进程可以通过读取套接字的方式读取内核中的信息。下面的消息类型可用于在用户进程和内核之间相互传递路由信息：

RTM_ADD	添加路由
RTM_CHANGE	更改网关、跃点数或标记
RTM_DELADDR	要从接口删除的地址
RTM_DELETE	删除路由
RTM_GET	报告跃点数和其他信息
RTM_IFINFO	接口打开、关闭等
RTM_LOSING	内核怀疑路由会失败
RTM_LOCK	锁定指定的跃点数
RTM_MISS	查找此地址失败
RTM_NEWADDR	要添加到接口的地址
RTM_REDIRECT	内核指示使用其他路由
RTM_RESOLVE	请求将目标解析为链路层地址

所有这 12 种消息类型都可用于从内核读取信息。要写入内核，进程可以发出 RTM_ADD、RTM_DELETE 或 RTM_GET 消息类型，以更新路由表中的信息。

HP-UX 不支持消息类型 RTM_CHANGE 和 RTM_LOCK。如果用户进程发出这些消息，则会返回 [EOPNOTSUPP] 错误。

消息结构

消息的构成形式为：消息头，后跟少量套接字地址结构。

要使用什么消息头取决于消息类型。 RTM_IFINFO 消息使用 **if_msghdr** 消息头。 RTM_NEWADDR 和 RTM_DELADDR 消息使用 **ifa_msghdr** 消息头。所有其他消息类型都使用 **rt_msghdr** 消息头。

rt_msghdr 结构包含以下成员：

```
uint16_t rtm_msglen;        /* to skip over unrecognized messages */
uint8_t  rtm_version;       /* future binary compatibility */
uint8_t  rtm_type;          /* message type */
uint16_t rtm_index;         /* index for associated ifp */
int32_t  rtm_flags;         /* flags, incl. kern & message, e.g. DONE */
int32_t  rtm_addrs;         /* bitmask identifying sockaddrs in the message */
pid_t    rtm_pid;           /* identify sender */
int32_t  rtm_seq;           /* for sender to identify action */
int32_t  rtm_errno;         /* error indicator */
int32_t  rtm_use;           /* from rtentry */
uint32_t rtm_inits;         /* which metrics we are initializing */
struct rt_metrics rtm_rmx;  /* metrics themselves */
```

if_msghdr 结构包含以下成员：

```
uint16_t ifm_msglen;        /* to skip over unrecognized messages */
uint8_t  ifm_version;       /* future binary compatibility */
```

```
uint8_t  ifm_type;          /* message type */
int32_t  ifm_addrs;         /* bitmask identifying sockaddrs in the message */
int32_t  ifm_flags;         /* value of if_flags */
uint16_t ifm_index;         /* index for associated ifp */
struct if_data ifm_data;    /* statistics and other data about interface */
```

ifa_msghdr 结构包含以下成员：

```
uint16_t ifam_msglen;       /* to skip over unrecognized messages */
uint8_t  ifam_version;      /* future binary compatibility */
uint8_t  ifam_type;         /* message type */
int32_t  ifam_addrs;        /* bitmask identifying sockaddrs in the message */
int32_t  ifam_flags;        /* value of ifa_flags */
uint16_t ifam_index;        /* index for associated ifp */
int32_t  ifam_metric;       /* value of ifa_metric */
```

为了确定重新传输行为，可靠的协议会使用 **rt_msghdr** 消息头中包含的 **rt_metrics** 结构。 **rt_metrics** 结构包含以下成员：

```
uint32_t rmx_locks;         /* Kernel must leave these values alone */
uint32_t rmx_mtu;           /* MTU for this path */
uint32_t rmx_hopcount;      /* max hops expected */
uint32_t rmx_expire;        /* lifetime for route, e.g. redirect */
uint32_t rmx_recvpipe;      /* inbound delay-bandwidth product */
uint32_t rmx_sendpipe;      /* outbound delay-bandwidth product */
uint32_t rmx_ssthresh;      /* outbound gateway buffer limit */
uint32_t rmx_rtt;           /* estimated round trip time */
uint32_t rmx_rttvar;        /* estimated rtt variance */
uint32_t rmx_pksent;        /* packets sent using this route */
```

if_msghdr 消息头中包含的 **if_data** 结构为每个网络接口定义一个队列，该接口包含以下成员：

```
/* generic interface information */
uint8_t ifi_type;           /* ethernet, tokenring, etc */
uint8_t ifi_physical;       /* AUI, Thinnet, 10base-T, etc */
uint8_t ifi_addrlen;        /* media address length */
uint8_t ifi_hdrlen;         /* media header length */
uint8_t ifi_recvquota;      /* polling quota for receive intrs */
uint8_t ifi_xmitquota;      /* polling quota for xmit intrs */
uint32_t ifi_mtu;           /* maximum transmission unit */
uint32_t ifi_metric;        /* routing metric (external only) */
uint32_t ifi_baudrate;      /* linespeed */
```

```
/* volatile statistics */
uint32_t ifi_ipackets;      /* packets received on interface */
uint32_t ifi_ierrors;       /* input errors on interface */
uint32_t ifi_opackets;      /* packets sent on interface */
uint32_t ifi_oerrors;       /* output errors on interface */
uint32_t ifi_collisions;    /* collisions on csma interfaces */
uint32_t ifi_ibytes;        /* total number of octets received */
uint32_t ifi_obytes;        /* total number of octets sent */
uint32_t ifi_imcasts;       /* packets received via multicast */
uint32_t ifi_omcasts;       /* packets sent via multicast */
uint32_t ifi_iqdrops;       /* dropped on input, this interface */
uint32_t ifi_noproto;       /* destined for unsupported protocol */
uint32_t ifi_hwassist;      /* HW offload capabilities */
uint32_t ifi_unused;        /* XXX was ifi_xmittiming */
struct timeval ifi_lastchange; /* time of last administrative change */
```

（请注意，前面提到的所有数据结构中项目的位置不一定反映结构中成员的顺序）。

消息头的成员 **rtm_addrs** 、 **ifm_addrs** 和 **ifam_addrs** 是一些位掩码，用于指定消息后跟什么套接字地址结构。当消息后跟多个 *sockaddrs* 时，将根据它们在消息中的顺序以及存储在位掩码中的值对其进行解释。向量中的序列顺序从最不重要的到最重要的位。

定义下列常量的目的是为了指明路由消息中存在哪些套接字地址：

```
#define RTA_DST     0x01  /* destination sockaddr present */
#define RTA_GATEWAY 0x02  /* gateway sockaddr present */
#define RTA_NETMASK 0x04  /* netmask sockaddr present */
#define RTA_GENMASK 0x08  /* cloning mask sockaddr present */
#define RTA_IFP     0x10  /* interface name sockaddr present */
#define RTA_IFA     0x20  /* interface address sockaddr present */
#define RTA_AUTHOR  0x40  /* author of redirect sockaddr present */
#define RTA_BRD     0x80  /* for NEWADDR, broadcast or
                           * point-to-point destination address */
```

发送到内核的任何消息都将返回到发出该命令的进程，同时会向所有感兴趣的监听程序发送消息副本。发送方可能会提供其进程 ID，以便存储在消息头中。还有一个序列字段可用于区分未完成的消息。但是，当内核缓冲区用完时，消息回复可能会丢失。

内核生成的任何消息都会将进程 ID 和序列字段设置为零。

为了响应外部事件（如收到重定向命令，或者无法为请求找到合适的路由），内核可能会自发地发出路由消息。进程可以通过对进一步输入执行 *shutdown*(2) 系统调用，从而忽略来自路由套接字的所有消息。

安全性限制

只有具有相应权限的用户才能更改路由表。

注释

消息头结构中的某些字段不在 HP-UX 中使用。这意味着，当内核生成路由消息时，会将这些字段设置为 0。另外，当内核收到路由消息时，它会忽略这些字段中包含的任何值。这种情况适用于下列字段：

结构	未使用的字段
rt_msghdr	rtm_use、rtm_inits rtm_rmx（rtm_rmx.rmx_mtu 和 rtm_rmx.rmx_rtt 除外）
if_msghdr	ifm_data （ ifm_data.ifi_mtu 、 ifm_data.ifi_metric 、 ifm_data.ifi_ipackets 和 ifm_data.ifi_opackets 除外）
ifa_msghdr	ifam_metric

错误

如果内核拒绝某个路由消息，则 **rt_msghdr** 结构中的 **rtm_errno** 字段可能会设置为下列值之一：

[EEXIST]	指定的条目已存在。已请求复制某个现有的条目。
[ENETUNREACH]	网络无法访问。
[ENOENT]	指定的条目不存在。请求删除的条目不存在。
[ENOBUFS]	没有可用的缓冲区空间。可用资源不足，无法安装新路由。
[EOPNOTSUPP]	不支持操作。 HP-UX 不支持消息类型 RTM_CHANGE 和 RTM_LOCK。
[EPERM]	发出命令的权限被拒绝。用户需要相应的权限才能更改路由表。

举例

以下示例程序说明了用户进程如何才能将路由添加到内核的路由表。

```
#include <sys/types.h>
#include <sys/socket.h>
#include <net/route.h>
#include <net/if.h>
#include <netinet/in.h>

int main(int argc, char **argv)
{
    int    s;
    char   buf[1024];
    struct rt_msghdr *rtm;
    struct sockaddr_in *sin1, *sin2;
```

```
if (argc != 3) {
        printf("usage: %s <destinationIP> <gatewayIP>\n",
                argv[0]);
        return -1;
}

if ((s = socket(AF_ROUTE, SOCK_RAW, AF_UNSPEC)) < 0) {
         perror("failed to create socket");
         return -1;
}

rtm = (struct rt_msghdr *)buf;

rtm->rtm_msglen = sizeof(struct rt_msghdr) +
                  (2 * sizeof(struct sockaddr_in));
rtm->rtm_version = RTM_VERSION;
rtm->rtm_type = RTM_ADD;
rtm->rtm_addrs = (RTA_DST | RTA_GATEWAY);
rtm->rtm_rmx.rmx_hopcount = 1;
rtm->rtm_pid = getpid();
rtm->rtm_errno = 0;
rtm->rtm_seq = 0001;

/*
 * the destination address being added follows
 * the routing header
 */
sin1 = (struct sockaddr_in *)(rtm + 1);
sin1->sin_family = AF_INET;
sin1->sin_addr.s_addr = inet_addr(argv[1]);

/*
 * the gateway address being added follows the
 * destination address
 */
sin2 = (struct sockaddr_in *)(sin1 + 1);
sin2->sin_family = AF_INET;
sin2->sin_addr.s_addr = inet_addr(argv[2]);
```

```
        if (write(s, (caddr_t)rtm, rtm->rtm_msglen) < 0) {
            perror("Failed to send routing message");
            return -1;
        }

        return 0;
    }
```

作者

路由套接字接口由 HP 和加州大学伯克利分校联合开发。

另请参阅

route(1M)、 ioctl(2)、 shutdown(2)、 socket(2)、 routing(7)。

名称

routing - 本地网络数据包路由选择的系统支持

说明

HP-UX 的网络设备提供了一般的数据包路由选择支持。路由选择表维护由应用程序进程处理。

路由选择表包括一组数据结构，当传输数据包时，网络设备将使用这些数据结构来选择相应的远程主机或网关。该表包含指向特定网络或主机的每个路由的单个条目，可使用带有 **-r** 或 **-rn** 选项的 **netstat** 命令（请参阅 *netstat*(1)）显示此路由。不显示无效的路由。

```
-------------------------------------------------------------
# netstat -r
Routing tables
Destination     Gateway          Flags  Refs  Use Interface Pmtu
hpindwr.cup.hp.com
                localhost        UH        1   39 lo0       4608
localhost       localhost        UH        0   68 lo0       4608
147.253.56.195  localhost        UH        0    0 lo0       4608
147.253.144.66  localhost        UH        0    0 lo0       4608
default         hpinsmh.cup.hp.com
                                 UG        1   21 lan0      1500
15.13.136       hpindwr.cup.hp.com
                                 U         1   92 lan0      1500
147.253.56      147.253.56.195   U         0    7 lan2      1500
147.253.144.64  147.253.144.66   U         0    7 lan1      1500
-------------------------------------------------------------
# netstat -rn
Routing tables
Destination     Gateway          Flags  Refs  Use Interface Pmtu
15.13.136.66    127.0.0.1        UH        1   39 lo0       4608
127.0.0.1       127.0.0.1        UH        0   68 lo0       4608
147.253.56.195  127.0.0.1        UH        0    0 lo0       4608
147.253.144.66  127.0.0.1        UH        0    0 lo0       4608
default         15.13.136.11     UG        2   30 lan0      1500
15.13.136.0     15.13.136.66     U         1  113 lan0      1500
147.253.56.0    147.253.56.195   U         0    7 lan2      1500
147.253.144.64  147.253.144.66   U         0    7 lan1      1500
-------------------------------------------------------------
# netstat -rv
Routing tables
Dest/Netmask    Gateway          Flags  Refs  Use Interface Pmtu
```

```
hpindwr.cup.hp.com/0xffffffff
                localhost       UH          1    39 lo0        4608
localhost/0xffffffff
                localhost       UH          0    68 lo0        4608
147.253.56.195/0xffffffff
                localhost       UH          0     0 lo0        4608
147.253.144.66/0xffffffff
                localhost       UH          0     0 lo0        4608
default/0x00000000
                hpinsmh.cup.hp.com
                                UG          2    31 lan0       1500
15.13.136/0xfffff800
                hpindwr.cup.hp.com
                                U           1   129 lan0       1500
147.253.56/0xfffffe00
                147.253.56.195  U           0     7 lan2       1500
147.253.144.64/0xfffffff0
                147.253.144.66  U           0     7 lan1       1500
-------------------------------------------------------------------
# netstat -rnv
Routing tables
Dest/Netmask     Gateway         Flags  Refs  Use Interface Pmtu
15.13.136.66/255.255.255.255
                127.0.0.1       UH          1    39 lo0        4608
127.0.0.1/255.255.255.255
                127.0.0.1       UH          0    68 lo0        4608
147.253.56.195/255.255.255.255
                127.0.0.1       UH          0     0 lo0        4608
147.253.144.66/255.255.255.255
                127.0.0.1       UH          0     0 lo0        4608
default/0.0.0.0 15.13.136.11    UG          3    40 lan0       1500
15.13.136.0/255.255.248.0
                15.13.136.66    U           1   153 lan0       1500
147.253.56.0/255.255.254.0
                147.253.56.195  U           0     8 lan2       1500
147.253.144.64/255.255.255.240
                147.253.144.66  U           0     8 lan1       1500
-------------------------------------------------------------------
```

与以下列特别相关:

Destination 目标 Internet 地址：主机名、网络名或 **default** 。 **default** 关键字指示通配符路由；如果没有为特定的远程主机或网络指定路由，则将此通配符路由用作最后手段。请参阅 **Flags** 。

Netmask 网络掩码和目标 Internet 地址共同定义了路由的网关可以到达的一系列 IP 地址。缺省情况下，主机路由拥有全部为 1 的网络掩码。缺省情况下，缺省路由拥有全部为 0 的网络掩码。在选择路由转发 IP 数据包时，也使用网络掩码。请参阅“路由选择算法”一节。

Gateway 用于到达目标的网关：远程网关或本地主机。请参阅 **Flags** 。

Flags 路由的类型：

- **U** 路由“已启动”或可用（请参阅 *ifconfig*(1M) ）。
- **G** 路由将远程主机用作网关；否则，本地主机将显示为网关（请参阅 *route*(1M) ）。
- **H** 目标为主机；否则，目标为网络（请参阅 *route*(1M) ）。

Interface 接口连接：

- **lo0** 系统引导后的本地环回。
- **lan0**, **lan1**,... 在引导时执行 **ifconfig** 命令后，在本地主机上安装的接口卡（请参阅 *ifconfig*(1M) ）。

route 命令中 *count* 和 *destination* 类型字段的值确定 **netstat -r** 显示中是否存在 **G** 和 **H** 标记，并因此确定路由类型，如下表所示。

计数	目标类型	标记	路由类型
=0	网络	**U**	从本地主机直接指向网络的路由
>0	网络	**UG**	经过远程主机网关指向网络的路由
=0	主机	**UH**	从本地主机直接指向远程主机的路由
>0	主机	**UGH**	经过远程主机网关指向远程主机的路由
=0	**default**	**U**	从本地主机开始的直接通配符路由
>0	**default**	**UG**	经过远程主机网关的通配符路由

子网

网络设备可支持长度可变的子网。 Internet 地址由 “网络地址”部分和某个地址的 “主机地址”部分组成，其格式如下：

192.34.17.0

子网地址定义为网络的 Internet 地址的一部分。该方案适用于：

- 以物理方式标识不同网络的网络地址。

- 以物理方式标识相同网络的不同子网的子网地址。

网络管理器可使用主机编号空间将本地网络的 Internet 地址细分为子网。多个物理网络可通过此设备共享单个 Internet 地址。

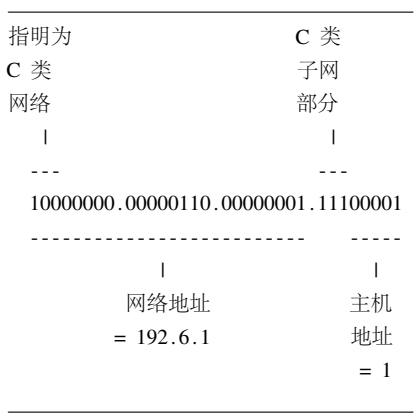
为实现此目的，定义了三个 Internet 类，每一个都可容纳不同数量的网络地址和主机地址。地址类由地址的二进制格式的最重要位定义。

下表列出了网络数、节点数和每个地址类的地址范围：

类	网络数	每个网络 节点数	地址范围
A	127	16777215	0.0.0.1 - 127.225.225.254
B	16383	65535	128.0.0.1 - 191.255.255.254
C	2097151	255	192.0.0.1 - 223.244.244.243
保留	—	—	224.0.0.0 - 255.255.255.255

A 类网络的前 8 位拥有的网络空间只有 127，但还要容纳定义的类中最大可能的节点数。单个 B 类网络包括 16 位网络地址限制，以及用于定义节点的 16 位。

例如，C 类地址空间如下：



可使用 **ifconfig** 命令（请参阅 *ifconfig*(1M) ）指定给定主机的子网，该命令使用带有 32 位子网 *mask* 的 **netmask** 参数。

这三个 Internet 地址类的缺省掩码如下：

A 类：255.0.0.0
B 类：255.255.0.0
C 类：255.255.255.0

C 类网络号的一个示例为 192.34.17.0。最后一个字段指定主机号。因此，带有前缀 192.34.17 的所有主机被识别为在相同的逻辑和物理网络上。

如果子网未被占用，则使用的缺省掩码为 255.255.255.0。

如果使用了子网，并且 8 位主机字段划分为上例所述的子网的 3 位和主机的 5 位，则子网掩码应为 255.255.255.192。

如果某个主机具有多个接口，则它可能属于不同的子网。与以往的发行版不同，子网可具有不同的大小，即使它们的网络地址相同，也是如此。可以对每个主机接口使用不同的网络掩码来获得不同大小的子网。例如，以上 **netstat** 表中显示的 **lan1** 和 **lan2** 接口可连接到同一网络 147.253 的两个不同的子网。 **lan1** 所属的子网最多可拥有 14 个主机，这是因为它的网络掩码为 255.255.255.240。

注释： 子网中的那些 IP 地址的主机部分不能全部为 1 或全部为 0，因此该子网只能支持 14 个主机，而不是 16 个。

lan2 所属的子网最多可拥有 510 个主机，这是因为它的网络掩码为 255.255.254.0。

超网

超网是较小网络的集合。超级联网是使用网络掩码将较小网络的集合汇聚成超网的技术。此技术对 C 类网络特别有用。 C 类网络只能拥有 254 个主机。对于某些公司来说，这可能存在太大的限制性。对于这些公司，只包含一部分网络部分的网络掩码可能会应用到这些 C 类网络中的主机以形成超网。此超网网络掩码应该应用到那些使用 *ifconfig* 命令（请参阅 *ifconfig*(1M)）连接到超网的接口。例如，通过将 IP 地址 192.6.1.1 以及网络掩码 255.255.0.0 配置给主机的接口，该主机可以配置其接口以连接到 C 类超网 192.6。

路由选择算法

共有三种类型的路由选择表条目：

- 特定主机的条目。
- 特定网络上所有主机的条目。
- 用于与前两种类型的条目不匹配的任何目标的通配符条目。

要选择一个用于转发 IP 数据包的路由，网络设备需要从路由选择表中选择“匹配的”路由选择表条目的完全集。如果路由选择条目中的网络掩码和 IP 数据包的目标地址之间的按位逻辑 AND 运算等于路由选择条目中的目标地址，则路由选择表条目被视为一个匹配项。

网络设备将从该集中选择拥有最长网络掩码的路由选择条目。网络掩码的长度定义为从 32 位网络掩码字段中最左边的位位置开始的连续 1 位的数目。换句话说，网络设备将选择指定了 IP 地址的最狭窄范围的路由选择条目。例如，带有目标/网络掩码对 (147.253.56.1, 0xFFFFFFFF) 的主机路由选择条目，比带有目标/网络掩码对 (147.253.56.0, 0xFFFFFE00) 的网络路由选择条目更明确；因此，网络设备将选择主机路由条目。缺省情况下，缺省路由的目标/网络掩码对为 (0,0)。因此，缺省路由能够匹配所有目标，但它也是最不明确的。仅当没有更明确的路由时，才选择缺省路由。

可能还余下了多个路由选择条目。在这种情况下，IP 数据包将通过 **netstat -r** 显示的第一个条目进行路由。所述的多个路由包括：

- 通过不同的网关到某个主机的两个或更多个路由。
- 通过不同的网关到某个网络的两个或更多个路由。

超级用户可通过使用 **route** 命令（请参阅 *route*(1M)），或使用因特网控制消息协议 (ICMP) 重定向消息中已收到的信息，来更改表中的条目。

如果特定网络或子网有多个缺省网关，将按顺序使用每个网关，以便影响不同网关的数据报的平均分布。

警告

在使用虚电路连接或双向数据报传输的情况下，如果要成功路由，则必须在本地主机、目标主机和所有中间主机上执行交互的 **route** 命令。

作者

routing 由加州大学伯克利分校开发。

文件

/etc/hosts

/etc/networks

另请参阅

netstat(1)、 ifconfig(1M)、 route(1M)、 route(7P)。 @@@本地网络数据包路由选择的系统支持

名称

sad - STREAMS 管理驱动程序

概要

```
#include <sys/types.h>
#include <sys/conf.h>
#include <sys/sad.h>
#include <stropts.h>

int ioctl(
        int fildes,
        int command,
        ...
        /* arg */
        );
```

说明

sad 驱动程序使用 **ioctl()** 函数提供 **autopush** 功能的接口。作为接口， **sad** 驱动程序使得管理任务得以在 STREAMS 模块和驱动程序上执行。通过为 **ioctl()** 函数指定 *command* 参数，管理员可以配置设备的 **autopush** 信息、获取有关设备的信息或检查模块列表。

fildes 是通过使用 **open()** 打开 **/dev/sad** 获取的文件描述符。 *command* 指定要执行的管理函数。 *arg* 指向数据结构。如果 *command* 是 **SAD_SAP** 或 **SAD_GAP** ，则 *arg* 指向类型为 **strapush** 的结构。如果 *command* 是 **SAD_VML** ，则 *arg* 指向类型为 **str_list** 的结构。

安全性限制

SAD_SAP ioctl() 仅限于超级用户或具有 **NETADMIN** 权限的用户使用。有关对支持精细权限划分系统的授权访问的详细信息，请参阅 *privileges*(5) 。

ioctl 命令

在 STREAMS 模块或驱动程序上用以执行管理功能的命令由下列 **ioctl()** 命令指定：

SAD_SAP 允许用户配置设备的 **autopush** 信息。 *arg* 参数指向 **strapush** 结构体（在 **<sys/sad.h>** 头文件中定义），其成员如下所示：

```
struct strapush {
    uint sap_cmd;
    long sap_major;
    long sap_minor;
    long sap_lastminor;
    long sap_npush;
    char sap_list[MAXAPUSH][FMNAMESZ+1];
};
```

sap_cmd 允许指定要执行的配置的类型。该字段可以具有下列值:

SAP_ALL 配置所有次设备。

SAP_RANGE
配置次设备范围。

SAP_ONE
配置单个次设备。

SAP_CLEAR
清除先前的设置。使用该命令时，请仅指定 **sap_major** 和 **sap_minor** 字段。如果先前的条目指定了 **SAP_ALL** ，则将 **sap_minor** 字段设置为 0（零）。如果先前的条目指定为 **SAP_RANGE** ，则将 **sap_minor**

字段设置为该范围内最小的次设备号。

sap_major
指定主设备号。

sap_minor
指定次设备号。

sap_lastminor
指定次设备范围。

sap_npush
指定要压入的模块数。该数目不得大于 **MAXAPUSH** ，后者在 **<sad.h>** 中定义。另外，该数目不能超过 **NSTRPUSH** 。

sap_list 按顺序指定要压入的模块的排列方式。

SAD_GAP
允许使用 **sad** 驱动程序获取设备的 **autopush**
配置信息，方法是将 **strapush**
结构体的 **sap_major**
和 **sap_minor**
字段（请参阅 **SAD_SAP** 命令）设置为所查询设备的主设备号和次设备号。

arg 应指向类型为 **strapush** 的 struct。成功完成后， **strapush** 结构将包含用于配置设备的所有信息。模块列表中任何未使用的条目将显示 0（零）值。

SAD_VML
允许检查模块列表。例如，可以确定是否安装了特定模块。 *arg* 参数指向 **str_list** 结构（在 **<stropts.h>** 头文件中定义），其成员如下所示:

struct str_list {

```
    int sl_nmods;
    struct str_mlist *sl_modlist;
};
```

sl_nmods 指定数组中已分配的条目数。

sl_modlist 指向模块名称数组。 **str_mlist** 结构（另请参阅 **<stropts.h>** 头文件）如下所示：

```
struct str_mlist {
    char    l_name[FMNAMESZ+1];
};
```

其中， **l_name** 指定模块名称数组。

如果 **l_name** 数组有效， **SAD_VML** 命令将返回值 0（零）。如果数组包含无效的模块名称，该命令将返回值 1。失败时，该命令将返回值 -1。

注释

作为 STREAMS 驱动程序， **sad** 还支持普通 STREAMS **I_STR ioctl()** ：

```
int ioctl(fildes, I_STR, strp);
int fildes;
struct strioctl *strp;
```

将 **strioctl** 结构中的 **ic_cmd** 字段以该形式指定为 **SAD_SAP** 、 **SAD_GAP** 或 **SAD_VML** 。 **ic_dp** 字段指向 **strapush** 结构（请参阅“说明”一节中的 **SAD_SAP** 命令）。有关详细信息，请参考 *streamio*(7) 参考页。

返回值

除非另行指定，否则成功完成后， **sad ioctl()** 命令将返回值 0（零）。否则，返回值 -1。

错误

如果出现下列任一情况，则 **sad ioctl** 命令将返回对应的值：

SAD_SAP

[EEXIST] 已配置指定的主（或次）设备号对 (**sad_major/sad_minor**) 。

[EFAULT] *arg* 参数指向分配的地址空间之外。

[EINVAL] 主设备号 (**sad_major**) 无效，模块数 (**sap_list[MAXAPUSH][FMNAMESZ+1]**) 无效，或模块名称列表无效。

[ENODEV] 设备未针对 **autopush** 进行配置。该值通过 **SAD_GAP** 命令返回。

[ENOSR] 无法分配内部 **autopush** 数据结构。

[ENOSTR] 主设备不表示 STREAMS 驱动程序。

[ERANGE] 当命令为 **SAP_RANGE** 时，或者 **SAP_CLEAR** 命令中指定的次设备不存在时，**sap_lastminor** 字段小于 **sap_minor** 字段。

[EACCES] 仅允许超级用户或具有 **NETADMIN** 权限的用户执行 **SAD_SAP ioctl()** 。

SAD_GAP

[EFAULT] *arg* 参数指向分配的地址空间之外。

[EINVAL] 主设备号 (**sad_major**) 无效。

[ENODEV] 设备未针对 **autopush** 进行配置。

[ENOSTR] 主设备不表示 STREAMS 驱动程序。

SAD_VML

[EFAULT] *arg* 参数指向分配的地址空间之外。

[EINVAL] 模块名称列表无效。

另请参阅

autopush(1M)、 ioctl(2)、 open(2)、 privileges(5)、 streamio(7)。

名称

scsi - 小型计算机系统接口设备驱动程序

说明

小型计算机系统接口 (SCSI) 是用于互连计算机和外围设备的美国国家标准。 HP-UX 支持针对并行 SCSI 接口（请参阅 ANSI Std X3.131-199X，“SCSI-2”）和光纤通道接口（请参阅 ANSI Std X3.269-199X，“Fibre Channel Protocol for SCSI”）和 Serial Attached SCSI 接口 (SAS) 的 SCSI 设备协议。

SCSI 标准包括很多设备类型的规范。本节介绍适用于所有 SCSI 设备驱动程序的常规 SCSI 接口。有关特定设备类型的信息，可在手册中描述那些设备类型的 SCSI 外围设备驱动程序的章节中找到。

这里介绍的读写控制可在持久性设备文件或 Legacy 设备上执行（请参阅 *intro*(7)）。HP-UX 11i v3 不建议使用 Legacy 设备文件。保留 Legacy 设备文件是为了向后兼容，会在以后的版本中过时。

根据读写控制命令在持久性设备文件还是在 Legacy 设备文件上执行以及 Legacy 设备文件上是否启用多路径，某些读写控制的行为可能会有所不同。通常，向设备发出 SCSI 命令的读写控制可能使用任何可用的 LUN 路径来发送命令。但是，当禁用 Legacy 设备文件上的多路径时（请参阅 *scsimgr*(1M) 中的 **leg_mpath_enable** 属性），读写控制仅尝试使用与 Legacy 设备文件相对应的 LUN 路径。如果该 LUN 路径不可用，即使存在其他可用 LUN 路径，读写控制也会失败。该行为与 Legacy 行为相对应。

所有 SCSI 设备驱动程序都支持 **SIOC_INQUIRY** 读写控制。此读写控制可返回特定于 SCSI 设备的 INQUIRY 命令数据此数据包含设备标识和功能信息。由于查询数据的 SCSI 标准存在多个版本，因此提供了多个版本的查询数据声明。提供 SCSI-1 版本仅仅是为了向后兼容。如果在 Legacy 设备文件上执行，即使 Legacy 设备文件上已启用多路径，该读写控制也仅尝试使用与 Legacy 设备文件相对应的 LUN 路径。

SIOC_CAPACITY 读写控制指示当前的设备大小。设备大小定义为逻辑块大小和逻辑块的某个数目。确定此设备大小数据的方式是与特定的设备类型相关的。如果逻辑块大小和（或）逻辑块数目等于零，则表示：设备大小未知，设备当前无法进行 I/O 操作，或者 I/O 操作对设备没有意义。请注意，对于极大型的设备，读写控制参数可能会溢出。如果是大型的设备，选择 **SIOC_STORAGE_CAPACITY** 比选择 **SIOC_CAPACITY** 更好。另请注意， **DIOC_CAPACITY** 是首选项（请参阅 *disk*(7)）。

头文件 **<sys/scsi.h>** 提供了有关 SCSI 设备的有用信息。以下内容摘自 **<sys/scsi.h>** :

```
#define SIOC_INQUIRY                _IOR('S', 2, union inquiry_data)
#define SIOC_CAPACITY               _IOR('S', 3, struct capacity)
#define SIOC_STORAGE_CAPACITY   _IOR('S', 101, storage_capacity_t)

/* SCSI-1 inquiry structure */
struct inquiry {
    unsigned char       dev_type;
    unsigned int        rmb:1;
    unsigned int        dtq:7;
    unsigned int        iso:2;
    unsigned int        ecma:3;
```

```
        unsigned int            ansi:3;
        unsigned int            resv:4;
        unsigned int            rdf:4;
        unsigned char           added_len;
        unsigned char           dev_class[3];
        char                    vendor_id[8];
        char                    product_id[16];
        char                    rev_num[4];
        unsigned char           vendor_spec[20];
        unsigned char           resv4[40];
        unsigned char           vendor_parm_bytes[32];
};

/* SCSI-2 inquiry structure */
struct inquiry_2 {
        unsigned int            periph_qualifier:3;
        unsigned int            dev_type:5;
        unsigned int            rmb:1;
        unsigned int            dtq:7;
        unsigned int            iso:2;
        unsigned int            ecma:3;
        unsigned int            ansi:3;
        unsigned int            aenc:1;
        unsigned int            trmiop:1;
        unsigned int            resv1:2;
        unsigned int            rdf:4;
        unsigned char           added_len;
        unsigned char           resv2[2];
        unsigned int            reladr:1;
        unsigned int            wbus32:1;
        unsigned int            wbus16:1;
        unsigned int            sync:1;
        unsigned int            linked:1;
        unsigned int            resv3:1;
        unsigned int            cmdque:1;
        unsigned int            sftre:1;
        char                    vendor_id[8];
        char                    product_id[16];
        char                    rev_num[4];
```

```
        unsigned char           vendor_spec[20];
        unsigned char           resv4[40];
        unsigned char           vendor_parm_bytes[32];
} inquiry_2_t;

/* Definition for version description in SCSI-3 inquiry */
typedef uint8_t                 vdesc_t[2];

/* SCSI-3 inquiry structure */
typedef struct inquiry_3 {
        uint32_t        pq              :3;
        uint32_t        pdt             :5;
        uint32_t        rmb             :1;
        uint32_t        rsvd1           :7;
        uint32_t        version         :8;
        uint32_t        aerc            :1;
        uint32_t        obslt1          :1;
        uint32_t        naca            :1;
        uint32_t        hisup           :1;
        uint32_t        rdf             :4;
        uint32_t        added_len       :8;
        uint32_t        sccs            :1;
        uint32_t        rsvd2           :7;
        uint32_t        bque            :1;
        uint32_t        encserv         :1;
        uint32_t        vs1             :1;
        uint32_t        multip          :1;
        uint32_t        mchngr          :1;
        uint32_t        obslt2          :1;
        uint32_t        obslt3          :1;
        uint32_t        addr16          :1;
        uint32_t        reladr          :1;
        uint32_t        obslt4          :1;
        uint32_t        wbus16          :1;
        uint32_t        sync            :1;
        uint32_t        linked          :1;
        uint32_t        obslt5          :1;
        uint32_t        cmdque          :1;
        uint32_t        vs2             :1;
```

```
        uint8_t     vendor_id[8];
        uint8_t     product_id[16];
        uint8_t     rev_num[4];
        uint8_t     vendor_spec[20];
        uint16_t    rsvd3               :4;
        uint16_t    clcking             :2;
        uint16_t    qas                 :1;
        uint16_t    ius                 :1;
        uint16_t    rsvd4               :8;
        vdesc_t     vers_desc[8];
        uint8_t     rsvd6[22];
        uint8_t     vendor_parm_bytes[32];
} inquiry_3_t;

/* union for SIOC_INQUIRY ioctl */
union inquiry_data {
        struct inquiry   inq1;      /* SCSI-1 inquiry */
        struct inquiry_2 inq2;      /* SCSI-2 inquiry */
        inquiry_3_t      inq3;      /* SCSI-3 inquiry */
};

/* structure for SIOC_CAPACITY ioctl */
struct capacity {
        uint32_t lba;
        uint32_t blksz;
};

/* structure for SIOC_STORAGE_CAPACITY ioctl */
typedef struct {
        uint64_t lba;
        uint32_t blksz;
} storage_capacity_t;
```

当有关设备状态和错误的详细信息可用时， **SIOC_XSENSE** 读写控制可返回这些信息。由于 sense（即状态）数据的 SCSI 标准存在多个版本，因此提供了 sense 数据声明的多个版本。提供 SCSI-1 和非对齐版本仅仅是为了向后兼容。如果自从上次 **SIOC_XSENSE** 读写控制调用后，未获取由 CHECK-CONDITION 产生的新 REQUEST SENSE 命令数据，则 **xsense_aligned.error_class** 和 **sense_2_aligned.error_code** 字段将包含零值。某些应用程序需要更精确的 REQUEST SENSE 数据处理，应使用 SCSI 设备控制驱动程序（请参阅 *scsi_ctl*(7) ）。

以下信息摘自 **<sys/scsi.h>** :

```
#define SIOC_XSENSE        _IOR('S', 7, union sense_data)

/* structure for SIOC_XSENSE ioctl */
typedef union sense_data {
      xsense_aligned_t        r_sense1a;   /* SCSI and CCS devices */
      sense_2_aligned_t       r_sense2a;   /* SCSI-2 devices */
      xsense_t                r_sense1;    /* Do not use; for compatibility only */
      sense_2_t               r_sense2;    /* Do not use; for compatibility only */
} sense_data_t;

/*
 * Struct xsense_aligned is for examining the sense data of SCSI-1
 * and CCS devices.
 */
typedef struct xsense_aligned {
      unsigned int    valid                :1;
      unsigned int    error_class          :3;
      unsigned int    error_code           :4;
      unsigned char   seg_num;
      unsigned int    parms:4;
      unsigned int    sense_key            :4;
      unsigned char   lba[4];
      unsigned char   add_len;
      unsigned char   copysearch[4];       /* Unused by HP-UX */
      unsigned char   sense_code;
      unsigned char   resv;
      unsigned char   fru;
      unsigned char   field;
      unsigned char   field_ptr[2];
      unsigned char   dev_error[4];
      unsigned char   misc_bytes[106];
} xsense_aligned_t;

/*
 *  Struct sense_2_aligned is for examining the sense data
 *  of SCSI-2 devices
 */
typedef struct sense_2_aligned {
      unsigned int    info_valid           :1;
```

```
        unsigned int    error_code          :7;
        unsigned char   seg_num;
        unsigned int    filemark            :1;
        unsigned int    eom                 :1;
        unsigned int    ili                 :1;
        unsigned int    resv                :1;
        unsigned int    key                 :4;
        unsigned char   info[4];
        unsigned char   add_len;
        unsigned char   cmd_info[4];
        unsigned char   code;
        unsigned char   qualifier;
        unsigned char   fru;
        unsigned char   key_specific[3];
        unsigned char   add_sense_bytes[113];
} sense_2_aligned_t;

/*
 * Struct xsense is provided for backward source code
 * compatibility only.
 * Struct xsense_aligned is the appropriate struct for
 * examining the sense
 * data of SCSI-1 and CCS devices.
 */
typedef struct xsense {
        unsigned int    valid               :1;
        unsigned int    error_class         :3;
        unsigned int    error_code          :4;
        unsigned char   seg_num;
        unsigned int    parms               :4;
        unsigned int    sense_key           :4;
        unsigned char   lba[4];
        unsigned char   add_len;
        unsigned char   copysearch[4];      /* Unused by HP-UX */
        unsigned char   sense_code;
        unsigned char   resv;
        unsigned char   fru;
        unsigned char   field;
        unsigned short  field_ptr;
```

```
        uint32_t        dev_error;
        unsigned char   misc_bytes[106];
} xsense_t;

/*
 * Struct sense_2 is provided for backward source code
 * compatibility only.
 * Struct sense_2_aligned is the appropriate struct for
 * examining the sense
 * data of SCSI-2 devices.
 */
typedef struct sense_2 {
        unsigned int    info_valid              :1;
        unsigned int    error_code              :7;
        unsigned char   seg_num;
        unsigned int    filemark                :1;
        unsigned int    eom                     :1;
        unsigned int    ili                     :1;
        unsigned int    resv                    :1;
        unsigned int    key                     :4;
        unsigned char   info[4];
        unsigned char   add_len;
        unsigned int    cmd_info;
        unsigned char   code;
        unsigned char   qualifier;
        unsigned char   fru;
        unsigned char   key_specific[3];
        unsigned char   add_sense_bytes[113];
} sense_2_t;
```

错误

调用 SCSI 设备驱动程序可能会导致以下错误：

[EACCES] 设备或操作所需的权限被拒绝。

[ENXIO] 如果是从打开调用返回的结果，这表示指定的地址上没有设备。对于其他调用，这表示指定的地址超出范围，或者不再可以访问该设备。

[EINVAL] 如果是从打开调用返回的结果，这表示设备驱动程序不支持该设备（例如，设备类型不正确）。对于其他调用，这表示该请求或其他某个请求参数无效。如果是从 **SIOC_CAPACITY** 读写控制返回的结果，则表示参数结构中的一个或多个字段可能溢出。

[EBUSY] 这表示该设备尚不可用，或者请求的操作与其他操作相互冲突（例如，当前通过其他设备驱动程序打开了该设备，或者独占访问有效）。

[EIO] 表示出现了 SCSI 协议问题或通信问题，或者某个 SCSI 命令导致了不良的状态。

介绍特定 SCSI 外围设备驱动程序的手册条目可能提供了有关错误结果的其他条件。

警告

使用不受正式支持的设备可能会导致数据丢失、系统异常和设备损坏。 HP-UX 设备驱动程序需要设备与 SCSI-2 兼容。只与 SCSI-CCS 兼容的不受支持设备可能会正常工作，但是不建议使用这些设备。强烈建议不要使用只与 SCSI-1 兼容的不受支持设备。

当系统正在运行时，不支持更改 SCSI 总线连接（重新布线）。当设备连接到不支持电源故障恢复的系统时，不支持打开或关闭 SCSI 设备电源。已知这些活动会导致数据丢失和系统异常。

在支持 **scsi_ctl** 接口的系统上，读写控制 **SIOC_CMD_MODE** 、 **SIOC_SET_CMD** 和 **SIOC_RETURN_STATUS** 已过时（请参阅 *scsi_ctl*(7) ）。通过 **scsi_ctl** 接口直接操作 SCSI 设备可提供更全面的功能，以及更方便地控制低级 SCSI 设备（请参阅 *scsi_ctl*(7) ）。

只支持未提供有效大小的设备的驱动程序可能不支持 **SIOC_CAPACITY** 读写控制。对于某些设备，总设备大小（单位为字节）可超过 $2^{32}-1$ 。

相关内容

esdisk/estape/eschgr/sdisk/schgr/stape

可使用 **SIOC_EXCLUSIVE** 读写控制获取和释放独占访问权限。某些操作必须使用独占访问权限，以防其他应用程序的同时访问，在其他某些情况下也可能需要使用独占访问权限。支持以下独占访问控制参数。相应的值将在 **<sys/scsi.h>** 中定义。如果在持久性设备文件上执行读写控制，则目标和总线实际上会独占访问 LUN 独占访问的结果。

SIOC_REL_LUN_EXCL 释放对逻辑单元 (LUN) 的独占访问权限。

SIOC_SET_LUN_EXCL 获取对逻辑单元 (LUN) 的独占访问权限。

SIOC_REL_TGT_EXCL 释放对关联 SCSI 目标的独占访问权限。

SIOC_SET_TGT_EXCL 获取对关联 SCSI 目标的独占访问权限。

SIOC_REL_BUS_EXCL 释放对关联 SCSI 总线的独占访问权限。

SIOC_SET_BUS_EXCL 获取对关联 SCSI 总线的独占访问权限。

当可移动介质设备中的介质发生更改时， **SIOC_MEDIUM_CHANGED** 读写控制会给出指示。值“1”指示自从上次 **SIOC_MEDIUM_CHANGED** 读写控制调用后，设备介质已发生更改。请注意，在介质更改后，只有第一次调用该读写控制时才会收到该指示。这意味着，如果多个应用程序尝试检测介质更改，则有可能会错过介质更改。如果使用 **SIOC_EXCLUSIVE** 读写控制获取独占访问权限，则可避免此问题。

以下信息摘自 **<sys/scsi.h>** :

```
#define SIOC_MEDIUM_CHANGED  _IOR('S', 42, int)
```

```
#define SIOC_EXCLUSIVE      _IOR('S', 68, int)
```

disc3

可通过 **SIOC_VPD_INQUIRY** 读写控制访问特定设备的详细信息。 **page_code** 字段指定要请求的 SCSI 重要产品数据页。需要使用请求页数据填充 **page_buf** 字段。在 Legacy 设备文件上执行读写控制时，即使 Legacy 设备文件上已启用多路径，该读写控制也仅尝试通过与 Legacy 设备文件相应的 LUN 路径发送 INQUIRY 命令。

以下信息摘自 **<sys/scsi.h>** :

```
#define SIOC_VPD_INQUIRY  _IOWR('S', 10, struct vpd_inquiry)

/* union for SIOC_VPD_INQUIRY ioctl */
struct vpd_inquiry {
    char    page_code;          /* VPD page code            */
    char    page_buf[126];      /* buffer for VPD page info  */
};
```

文件

/usr/include/sys/scsi.h

另请参阅

diskinfo(1M)、 ioctl(2)、 autochanger(7)、 intro(7)、 scsi_ctl(7)、 scsi_disk(7)、 scsi_tape(7)。

名称

scsi_ctl - SCSI 通透式驱动程序 (esctl/sctl)

说明

SCSI 设备由设备专用的驱动程序（如果有）控制。诸如 SCSI 直接访问（磁盘）设备和顺序访问（磁带）设备的驱动程序等设备专用的驱动程序可以协调设备和驱动程序的状态，以便完成正常的逻辑设备行为。通过 SCSI 通透式驱动程序，可以使用这些设备专用的驱动程序通常不支持的 SCSI 设备和命令。

esctl 是 SCSI 通透式驱动程序，并可同持久性设备文件一起使用（请参阅 *intro*(7) ）。 **sctl** 是已用于 HP-UX 11i v3 之前的 HP-UX 版本的 SCSI 通透式驱动程序。在此处保留以便向后兼容，它可用于 Legacy 设备文件。在本文档中， **scsi_ctl** 既指 **esctl** ，又指 **sctl** 。

一旦通过 **scsi_ctl** 驱动程序打开设备，则可使用 ioctl 调用更改 SCSI 通信参数或者尝试 SCSI 命令以及其他 SCSI 操作。由于 SCSI 通透式驱动程序不尝试以逻辑方式识别目标设备，因此不支持 **read()** 和 **write()** 调用。

除非另有说明，否则可通过所有的 SCSI 设备驱动程序（包括设备专用的驱动程序）使用此处所述的读写控制。必须在与这些读写控制关联的数据结构的所有 **reserved** 字段中填充零。

下列特定于并行 SCSI 的读写控制不推荐在 LUN 设备专用文件 (DSF) 上执行。持久性设备专用文件上并不支持这些读写控制。 Legacy 设备专用文件将继续支持它们以实现向后兼容。但是，现在建议直接在并行 SCSI HBA 设备专用文件 (DSF) 上执行这些读写控制或随 HP-UX 11i v3 引入的等效读写控制。

SIOC_GET_TGT_PARMS
SIOC_GET_BUS_PARMS
SIOC_GET_TGT_LIMITS
SIOC_GET_BUS_LIMITS
SIOC_SET_TGT_LIMITS
SIOC_SET_BUS_LIMITS

应直接在并行 SCSI HBA DSF 上执行下列随 HP-UX 11i v3 引入的并行 SCSI 特定读写控制（它们替换了某些现有读写控制，后者从 HP-UX 11i v3 开始不能在 LUN 持久性设备专用文件上执行）：

PSIOC_GET_TGT_LIMITS	替换 **SIOC_GET_TGT_PARMS**
PSIOC_GET_TGT_PARMS	替换 **SIOC_GET_BUS_PARMS**
PDIOC_RSTCLR	替换 **DIOC_RSTCLR**
PSIOC_RESET_DEV	替换 **SIOC_RESET_DEV**

不推荐在 HP-UX 11i v3 上使用 Legacy 设备文件。保留该文件以实现向后兼容，并且可能在以后的版本中会过时（有关 Legacy 设备文件和持久性设备文件的详细信息，请参阅 *intro*(7) ）。推荐新的应用程序使用持久性设备文件。

此处介绍的大多数读写控制都能在外围设备文件或 Legacy 设备文件上执行。根据在持久性设备文件还是在 Legacy 设备文件上执行以及 Legacy 设备文件上是否已启用多路径，某些读写控制的行为可能会有所不同。通常，向设备发出 SCSI 命令的读写控制可能使用任何到设备的可用 LUN 路径来发送命令。但是，如果 Legacy 设备文件上禁用多路径（请参阅 *scsimgr*(1M) 中的 **leg_mpath_enable** 属性），读写控制将仅尝试使用与 Legacy 设

备文件相对应的 LUN 路径。如果该 LUN 路径不可用，即使存在其他可用 LUN 路径，读写控制也会失败。这种行为与 Legacy 行为相对应。

设备专用文件次设备号

通透式驱动程序 (**esctl/sctl**) 是执行读写控制 **SIOC_IO_EXT** （仅对于esctl ）和 **SIOC_IO** 读写控制的首选方法，但不会执行设备专用驱动程序（例如 **esdisk** ）。要完成该操作，必须为通透式驱动程序创建设备专用文件。*mksf*(1M) 是针对 **esctl** 创建通透式设备文件的推荐方法。要为 Legacy 通透式驱动程序 **sctl** 创建设备文件，请使用 *mknod*(1M) 替代在次设备号中指明的值：

/usr/sbin/mknod *name* **c 203 0x***iitl***0o**

其中的次设备号的组成部分可按如下方式构建：

ii 两个十六进制数字，通过其“实例”号标识控制接口卡。“接口”硬件类型的列 **I** 下面的 *ioscan*(1M) 输出中显示“实例”值。

t 一个十六进制数字，标识设备（目标）地址。

l 一个十六进制数字，标识设备内的逻辑单元号 (LUN)。

0 一个十六进制数字零，用作次设备号的保留部分。

o 可选值如下所示：

0 打开时执行 Inquiry，以确保设备存在（建议方法）；或者

2 打开时禁止执行 Inquiry。从 HP-UX 11i v3 开始，不推荐使用选项 2。保留它的目的是为了与已设置为该选项的现有应用程序实现二进制兼容。将在打开时实际发送 Inquiry 命令，而无论该选项是否设置为 2。

SCSI 通信参数

HP-UX 支持针对并行 SCSI 接口和光纤通道接口的 SCSI 设备协议。此处所述的 SCSI 通信参数可能只适用于特定的 SCSI 接口，并且说明中已注明了这一点。

SCSI 通信参数可控制与三个不同范围级别的通信相关的功能：总线（链路）、目标和逻辑单元号 (LUN)。总线通信参数适用于已连接到特定总线的所有目标。目标通信参数适用于与特定目标关联的所有 LUNLUN 通信参数适用于特定的 LUN。 SCSI 通信参数适用于所有设备驱动程序（设备专用的驱动程序和 **scsi_ctl** ）。

在通电时和重置后，所有的并行 SCSI 设备和主机都使用异步数据传输通信。异步数据传输使用请求 (REQ) 和确认 (ACK) 传输信号。对 REQ 和 ACK 信号进行严格排序可简化通信协议，但限制了 I/O 的性能。 SCSI 目标和主机对可能会接受使用同步数据传输以提高 I/O 性能。

同步数据传输通过降低对 REQ 和 ACK 的排序要求，来提高 I/O 性能。通过允许多个未完成的 REQ，将可以更好的接收信号传播延迟和临时速率失衡。为了利用同步数据传输， SCSI 目标和主机必须协商确定彼此可接受的最大 REQ-ACK 偏移量参数和数据传输速率参数。

最大的 REQ-ACK 偏移量参数指示可允许的最大未完成 REQ 数。可使用零值指示异步数据传输。其他值指示同

步数据传输。适当的值通常取决于接收数据 FIFO 的大小。较大的值有助于提高数据传输速率。最大数据传输速率参数指的是“爆发”数据的传输速率（连续两次同步数据传输之间允许的最小时间）。用于启动协商进程的 SCSI 同步数据传输请求 (SDTR) 消息，是与 SCSI 命令的处理相关联的。

在通电时和重置后，所有并行的 SCSI 设备和主机都使用八位数据传输进行通信。 SCSI 目标和主机对可能会接受使用十六位（宽）数据传输来提高 I/O 性能。为了利用宽数据传输， SCSI 目标和主机必须协商确定彼此可接受的数据传输宽度参数。用于启动协商进程的 SCSI 宽数据传输请求 (WDTR) 消息，是与 SCSI 命令的处理相关联的。

某些 SCSI 设备能够同时管理多个活动的命令。这种设备拥有一个能够保留要处理的命令的命令队列。命令队列可通过减少设备等待来自主机的新命令所耗用的时间来提高 I/O 性能。请注意，对于支持“预读”和“即时报告”（请参阅 *scsi_disk*(7) 和 *scsi_tape*(7) ）的设备，命令队列可能无法显著地提高 I/O 性能。 SCSI 设备和主机可使用命令标记来正确管理多个同时活动的命令。无论何时，只要命令队列有效，由特定 LUN 处理的每个活动命令就会带有一个唯一的命令标记。

SCSI 设备将在它们的 SCSI **INQUIRY** 命令数据中，指示它们是否能够支持上述特殊的通信功能。通常，主机和设备可通过 SCSI **INQUIRY** 命令数据和协商协议来确定最佳的通信参数，以最大程度地提高 I/O 性能。

可使用 **SIOC_GET_LUN_PARMS** 、 **PSIOC_GET_TGT_PARMS** （推荐）或 **SIOC_GET_TGT_PARMS** （为了向后兼容）和 **SIOC_GET_BUS_PARMS** 读写控制来确定当前的操作通信参数。

有时，为了解决某个通信问题，或者在确定最佳参数时提供一些外部提示，需要限制 SCSI 通信参数。可使用 **SIOC_SET_LUN_LIMITS** 、 **SIOC_SET_TGT_LIMITS** 和 **SIOC_SET_BUS_LIMITS** 读写控制指定 SCSI 通信参数限制建议。

请注意，指定的通信参数限制建议与当前实际用于通信的对应通信参数之间可能存在很大差异。产生这些差异的原因在于设备专用的驱动程序的功能、接口驱动程序的功能、接口硬件的功能、设备的功能、由协商进程产生的延迟、由当前活动命令产生的延迟、由于等待向设备发送命令而产生的延迟。请注意，如果所有 SCSI 设备驱动程序（设备专用的驱动程序或 **scsi_ctl** ）都不具有已打开的关联 LUN，则通信参数限制建议将无法存活于 **close()** 调用和 **open()** 调用之间。

可以使用 **SIOC_GET_LUN_LIMITS** 、 **SIOC_GET_TGT_LIMITS** 和 **SIOC_GET_BUS_LIMITS** 读写控制确定当前的 SCSI 通信参数限制建议。

可以使用 **SIOC_GET_LUN_PARMS** 、 **SIOC_SET_LUN_LIMITS** 和 **SIOC_GET_LUN_LIMITS** 、 **SIOC_RESET_DEV** 、 **SIOC_RESET_BUS** 读写控制管理逻辑单元通信参数。

SIOC_GET_LUN_PARMS 读写控制指示当前的 LUN 通信参数值。 *max_q_depth* 字段指示是否启用带标记的队列，以及启用时允许同时活动的命令的最大数目。如果 *max_q_depth* 为零，则禁用带标记的队列。如果该值为一，则使用标记，但仍然需要以串行方式处理命令。如果 *max_q_depth* 大于一，则使用标记，并且指定允许同时活动的命令的最大数目。

可使用 **SIOC_SET_LUN_LIMITS** 读写控制来提供 LUN 通信参数限制建议。 *max_q_depth* 字段指示是否应启用带标记的队列，以及启用时允许同时活动的命令的最大数目。 **SIOC_GET_LUN_LIMITS** 读写控制指示当前的 LUN 通信参数限制建议。

可以使用任何关联 HBA DSF 上的 **PSIOC_GET_TGT_PARMS**，或任何关联 LUN 的 **SIOC_GET_TGT_PARMS**、**SIOC_SET_TGT_LIMITS** 和 **SIOC_GET_TGT_LIMITS** 读写控制来管理目标通信参数。

PSIOC_GET_TGT_PARMS 和 **SIOC_GET_TGT_PARMS** 读写控制指示当前的目标通信参数值。*width*、*reqack_offset* 和 *xfer_rate* 字段指示当前已协商的数据传输参数。如果 *width* 为八，则窄传输有效。如果该值为十六，则宽传输有效。如果 *reqack_offset* 为零，则异步传输有效，而 *xfer_rate* 没有意义。如果 *reqack_offset* 为非零值，则同步传输有效，同时最大的“爆发”数据传输速率为每秒 *xfer_rate* 个单词，*width* 中指示了此处所述的单词的大小。

SIOC_SET_TGT_LIMITS 读写控制指定目标通信参数限制建议。*max_width* 字段指定应该用于数据传输的最大总线宽度。*max_reqack_offset* 字段指定在数据传输期间，应该尝试的未完成 REQ 的最大数目。*max_xfer_rate* 字段指定同步数据传输期间内允许的最大“爆发”数据速率。**SIOC_GET_TGT_LIMITS** 读写控制指示当前的目标通信参数限制建议。*width*、*reqack_offset*、*xfer_rate*、*max_width*、*max_reqack_offset* 和 *max_xfer_rate* 字段只适用于并行的 SCSI。

可以使用面向任何关联 LUN 的 **SIOC_GET_BUS_PARMS**、**SIOC_SET_BUS_LIMITS** 和 **SIOC_GET_BUS_LIMITS** 读写控制来管理总线通信参数。

SIOC_GET_BUS_PARMS 读写控制指示当前的总线通信参数值。*max_width* 字段指示将数据传输到与关联总线连接的任何目标设备时，将要尝试的最大数据传输宽度。*max_reqack_offset* 字段指示将数据传输到与关联总线连接的任何目标设备时，将要尝试的未完成 REQ 的最大数目。*max_xfer_rate* 字段指示将数据传输到与关联总线连接的任何目标设备时，将要尝试的最大“爆发”数据传输速率。

SIOC_SET_BUS_LIMITS 读写控制为已连接到关联总线的目标指定总线通信参数限制建议。*max_width* 字段指定将数据传输到与关联总线连接的任何目标设备时，应该尝试的建议最大数据传输宽度。*max_reqack_offset* 字段指定将数据传输到与关联总线连接的任何目标设备时，应该尝试的未完成 REQ 的最大数目。*max_xfer_rate* 字段指定将数据传输到与关联总线连接的任何目标设备时，应该尝试的最大同步“爆发”数据传输速率。**SIOC_GET_BUS_LIMITS** 读写控制指示当前的总线通信参数限制建议。*max_width*、*max_reqack_offset* 和 *max_xfer_rate* 字段只适用于并行的 SCSI。

以下内容摘自 **<sys/scsi.h>**：

```
/* SCSI communication parameter ioctls */
#define SIOC_GET_LUN_PARMS     _IOR('S', 58, struct sioc_lun_parms)
#define SIOC_GET_TGT_PARMS     _IOR('S', 59, struct sioc_tgt_parms)
#define SIOC_GET_BUS_PARMS     _IOR('S', 60, struct sioc_bus_parms)
#define SIOC_GET_LUN_LIMITS    _IOR('S', 61, struct sioc_lun_limits)
#define SIOC_GET_TGT_LIMITS    _IOR('S', 62, struct sioc_tgt_limits)
#define SIOC_GET_BUS_LIMITS    _IOR('S', 63, struct sioc_bus_limits)
#define SIOC_SET_LUN_LIMITS    _IOW('S', 64, struct sioc_lun_limits)
#define SIOC_SET_TGT_LIMITS    _IOW('S', 65, struct sioc_tgt_limits)
#define SIOC_SET_BUS_LIMITS    _IOW('S', 66, struct sioc_bus_limits)
```

```
struct sioc_lun_parms {
        unsigned int flags;
        unsigned int max_q_depth;          /* maximum active I/O's */
        unsigned int reserved[4];          /* reserved for future use*/
} sioc_lun_parms_t;

struct sioc_lun_limits {
        unsigned int flags;
        unsigned int max_q_depth;
        unsigned int reserved[4];          /* reserved for future use */
} sioc_lun_limits_t;

typedef struct sioc_tgt_parms {
        unsigned int flags;
        unsigned int width;                /* bits per word */
        unsigned int xfer_rate;            /* words per second */
        unsigned int reqack_offset;        /* REQ/ACK offset */
        unsigned int tgt_id;               /* target Id */
        unsigned int reserved[3];          /* reserved for future use */
} sioc_tgt_parms_t;

typedef struct sioc_tgt_limits {
        unsigned int flags;
        unsigned int max_width;            /* Bits per word */
        unsigned int max_xfer_rate;        /* Words per second */
        unsigned int max_reqack_offset;    /* REQ/ACK offset */
        unsigned int tgt_id;               /* target Id */
        unsigned int reserved[3];          /* Reserved for future use */
} sioc_tgt_limits_t;

struct sioc_bus_parms {
        unsigned int flags;                /* reserved for future use */
        unsigned int max_width;
        unsigned int max_reqack_offset;
        unsigned int max_xfer_rate;        /* bytes/sec */
        unsigned int reserved[4];          /* reserved for future use */
} sioc_bus_parms_t;

struct sioc_bus_limits {
```

```
    unsigned int flags;                 /* reserved for future use */
    unsigned int max_width;
    unsigned int max_reqack_offset;
    unsigned int max_xfer_rate;         /* bytes/sec */
    unsigned int reserved[4];           /* reserved for future use */
} sioc_bus_limits_t;
```

包含 **<sys/pscsi.h>**的以下内容：

```
#define PSIOC_GET_TGT_PARMS     _IOWR('S', 114, struct sioc_tgt_parms)
#define PSIOC_GET_TGT_LIMITS    _IOWR('S', 115, struct sioc_tgt_limits)
#define PSIOC_RESET_DEV         _IOW('S', 116, int)
#define PDIOC_RSTCLR            _IOW('S', 117, int)
```

SCSI 命令和操作

可通过 **SIOC_IO_EXT** 和 **SIOC_IO** 读写控制向设备发送任意 SCSI 命令。将自动处理 SCSI 命令协议的所有细节。 **SIOC_IO_EXT** 只能在持久性设备文件上执行。它通过任何可用 LUN 路径或选定的 LUN 路径发送 scsi 命令。不推荐使用 **SIOC_IO** 。可在持久性设备文件和 Legacy 设备文件上执行该命令。当在持久性设备文件上执行该命令时，可通过任何可用的 LUN 路径发送 SCSI 命令。

可使用下列标记指定 **SIOC_IO_EXT** 和 **SIOC_IO** 的 标记字段的值，除非另行指明：

SCTL_READ 如果 *data_length* 字段为非零值，则需要执行数据读取操作。如果没有这个标记，则表示当 *data_length* 字段为非零值时，需要执行数据写入操作。

SCTL_INIT_SDTR 应使用此命令尝试同步数据传输请求协商。此标记只适用于并行的 SCSI 并保留以向后兼容。

SCTL_INIT_WDTR 应使用此命令尝试宽数据传输请求协商。此标记只适用于并行的 SCSI 并保留以向后兼容。

SCTL_NO_DISC 未设置 Identify 消息中的 discpriv 位。此标记只适用于并行的 SCSI 并保留以向后兼容。

ESCTL_IO_LPT 在给定的 LUN 路径上执行 SCSI 命令。只能使用 **SIOC_IO_EXT** 读写控制指定该标记。指定时，要使用的 LUN 路径的硬件路径在字段 *lpt_hwp* 中指定。

cdb 字段指定 SCSI 命令字节。命令字节数由 *cdb_length* 字段指定。在 SCSI 命令阶段，会将这些命令字节发送到目标设备。

SCSI 命令数据阶段所用数据区地址由 *data* 字段指定。 *data_length* 字段指定要传输的最大数据字节数。 *data_length* 值为 0 表示不应该出现数据阶段。大多数带有数据阶段的 SCSI 命令需要在命令字节中的某个位置包含数据长度信息。调用方负责正确指定 *data_length* 字段和任何 cdb 数据长度值。长度不能大于

SCSI_MAXPHYS，而且一些实现还会限制该长度。

max_msecs 字段指定设备完成命令所需的最大时间，以毫秒为单位。如果命令尚未完成，但此时间段已过期，则系统可能会尝试恢复过程，以便重新获得设备的关注。这些恢复过程包括异常中止标记、异常中止、设备和总线重置操作。*max_msecs* 字段中的零值表示超时时间是无限的，因此系统应无限期地等待命令的完成。

如果 **SIO_IO_EXT** 或 **SIOC_IO** 读写控制调用返回，则表示已完成所有的命令处理。大多数 **SIOC_IO_EXT/SIOC_IO** 读写控制调用将返回零（成功）。应使用所得到的详细的读写控制数据，从调用方的角度来评估“成功”或“失败”。*cdb_status* 字段指示 **cdb** 命令的结果。如果 *cdb_status* 字段指示一个 **S_CHECK_CONDITION** 状态，则 *sense_status* 字段将指示用于收集关联 sense 数据的 SCSI **REQUEST SENSE** 命令的结果。这些状态字段包含下列值之一：

SCTL_INVALID_REQUEST	SCSI 命令请求无效，因此未尝试此请求。
SCTL_SELECT_TIMEOUT	目标设备对主机 SCSI 接口所作的选择没有应答（该设备不存在或没有响应）。
SCTL_INCOMPLETE	设备对选择作出了应答，但是命令未完成（设备所用时间过长或通信失败）。
S_GOOD	设备成功完成了命令。
S_CHECK_CONDITION	设备指明的 sense 数据可用。
S_CONDITION_MET	设备成功完成了命令，并且请求的（搜索或预提取）操作得到满足。
S_BUSY	设备指明因其正忙于进行其他操作而无法接受命令。
S_INTERMEDIATE	设备成功完成了该命令，它是一系列链接命令中的一个命令（不受支持，请参阅“警告”一节）。
S_I_CONDITION_MET	设备指明 **S_INTERMEDIATE** 和 **S_CONDITION_MET**（不受支持，请参阅“警告”一节）。
S_RESV_CONFLICT	设备指明命令与现有保留相冲突。
S_COMMAND_TERMINATED	设备指明命令在早期已被主机系统终止。
S_QUEUE_FULL	设备指明因其命令队列当前已满而无法接受命令。

data_xfer 字段指明在 **cdb** 命令的数据阶段期间实际传输的数据字节数。仅当 *cdb_status* 字段包含下列值之一时该字段有效：**S_GOOD** 或 **S_CHECK_CONDITION**。*sense_xfer* 字段指明有效 sense 数据字节数。仅当 *cdb_status* 字段包含值 **S_CHECK_CONDITION** 且 *sense_status* 字段包含值 **S_GOOD** 时，该字段有效。

可通过 **SIOC_ABORT** 读写控制将 SCSI **ABORT** 消息发送到 LUN。这样，通过此启动器可清除所有 LUN 的活动命令。

如果 SCSI 传输支持 SCSI 任务管理函数，则 **SIOC_TASK_MGMT** 读写控制会导致执行 SCSI 任务管理函数。可指定下列任务管理函数值。它们定义于 **<sys/scsi.h>** 中：

SIOC_TM_LUN_RESET	Lun 重置
SIOC_TM_WARM_TGT_RESET	暖目标重置
SIOC_TM_COLD_TGT_RESET	冷目标重置

可通过 **SIOC_RESET_DEV** 读写控制重置 SCSI 设备（包括清除所有活动命令）。在并行 SCSI 上，可通过 **PSIOC_RESET_DEV** 和 **SIOC_RESET_DEV** 读写控制向关联的目标发送 SCSI **BUS DEVICE RESET** 消息。在光纤通道上，可通过 **SIOC_RESET_DEV** 读写控制向后跟 Global Process Logout (GPRLO) 的关联目标发送“TARGET RESET”任务管理函数。

系统可通过 **SIOC_RESET_BUS** 读写控制针对关联总线生成一个 SCSI 总线重置条件。可通过 SCSI 总线重置总线上的所有设备（包括清除所有设备上的所有活动命令）。 **SIOC_RESET_BUS** 读写控制不适用于光纤通道。

一般来说，在执行设备控制操作时，禁止其他 SCSI 命令是必要的，或者是有用的。应通过使用 **SIOC_EXCLUSIVE** 读写控制获取独占访问权限来完成此设置。关联参数指向一个整数，这些值中的其中一个是在 **<sys/scsi.h>** 中定义的。请注意，如果读写控制在持久性设备文件上执行，则目标和总线独占访问请求将使系统执行 LUN 独占访问。

SIOC_REL_LUN_EXCL	释放对逻辑单元的独占访问权限
SIOC_SET_LUN_EXCL	获取对逻辑单元的独占访问权限
SIOC_REL_TGT_EXCL	释放对目标的独占访问权限
SIOC_SET_TGT_EXCL	获取对目标的独占访问权限
SIOC_REL_BUS_EXCL	释放对总线的独占访问权限
SIOC_SET_BUS_EXCL	获取对总线的独占访问权限

不推荐在 HP-UX 11i v3 中使用读写控制 **SIOC_PRIORITY_MODE** 。如果调用了该读写控制，它会伪装成功。使用该读写控制可以解决无法设置设备独占访问的问题。它将设备放置在“优先级模式”中。可导致所有专用设备驱动程序的 I/O 操作（例如，文件系统 I/O 和虚拟内存页面交换）受阻塞，以及使对关联的 LUN 执行的所有 SCSI 设备驱动程序打开调用（其中包括通透式驱动程序打开调用）受阻塞。在该优先级模式生效的整个过程中，这些 I/O 操作和打开调用都将阻塞。当优先级模式生效时，只能尝试 **SIOC_IO** 打开调用操作（这些操作不会阻塞）。如果不正确使用 **SIOC_PRIORITY_MODE** 读写控制可很容易导致系统死锁。通常在启用优先级模式之前，要求将调用进程锁入内存（请参阅 *plock*(2) ）。

头文件 **<sys/scsi.h>** 提供了有关 SCSI 设备控制的有用信息。以下内容摘自 **<sys/scsi.h>** ：

```
/* SCSI device control ioctls */
#define SIOC_IO                 _IOWR('S', 22, struct sctl_io)
#define SIOC_RESET_DEV          _IO('S', 16)
#define SIOC_RESET_BUS          _IO('S', 9)
#define SIOC_PRIORITY_MODE      _IOW('S', 67, int)

#define SIOC_IO_EXT             _IOWR('S', 102, esctl_io_t)
#define SIOC_TASK_MGMT          _IOWR('S', 104, sioc_task_mgmt_t)
```

```
/* Structure for SIOC_IO_EXT ioctl */
typedef struct {
    int                     version;
    escsi_sctl_io_flags_t   flags;
    int                     max_msecs;
    uint32_t                cdb_length;
    uint32_t                data_length;
    ptr64_t                 data;
    union sense_data        sense;
    escsi_hw_path_t         lpt_hwp;
    uint32_t                data_xfer;
    uint32_t                sense_xfer;
    uint32_t                cdb_status;
    uint32_t                sense_status;
    uint8_t                 cdb[ESCSI_MAX_CDB_LEN];
    uint32_t                rsvd[32];  /* Reserved for future use */
} esctl_io_t;

/* Structure for SIOC_IO ioctl */
struct sctl_io
{
    unsigned        flags;
    unsigned char   cdb_length;
    unsigned char   cdb[16];
    void            *data;
    unsigned        data_length;
    unsigned        max_msecs;
    unsigned        data_xfer;
    unsigned        cdb_status;
    unsigned char   sense[256];
    unsigned        sense_status;
    unsigned char   sense_xfer;
    unsigned char   reserved[64];
} sctl_io_t;
```

安全性限制

使用这些读写控制需要具有超级用户或 **DEVOPS** 权限，或者设备写入权限。有关对支持精细权限划分的系统的授权访问的详细信息，请参阅 *privileges*(5) 。

举例

假定 *fildes* 是某个 SCSI 设备持久性设备文件的有效文件描述符， *leg_fildes* 是某个 SCSI 设备 Legacy 设备文件的有效文件描述符，并且 *lpt_hwp* 包含一个指向该设备的 LUN 路径的有效硬件路径。第一个示例将尝试执行 SCSI **INQUIRY** 命令：

```
#include <sys/scsi.h>

esctl_io_t esctl_io;
#define MAX_LEN 255
unsigned char inquiry_data[MAX_LEN];

memset(&esctl_io, 0, sizeof(esctl_io));      /* clear reserved fields */
esctl_io.flags = SCTL_READ;                  /* input data expected */
esctl_io.cdb[0] = CMDinquiry;
esctl_io.cdb[1] = 0x00;
esctl_io.cdb[2] = 0x00;
esctl_io.cdb[3] = 0x00;
esctl_io.cdb[4] = MAX_LEN;                   /* allocation length */
esctl_io.cdb[5] = 0x00;
esctl_io.cdb_length = 6;                     /* 6 byte command */
esctl_io.data = &inquiry_data[0];            /* data buffer location */
esctl_io.data_length = MAX_LEN;              /* maximum transfer length */
esctl_io.max_msecs = 10000;                  /* allow 10 seconds for cmd */
if (ioctl(fildes, SIOC_IO_EXT, &esctl_io) < 0) {
    /* request is invalid */
} else {
    if ( esctl_io.cdb_status == S_GOOD) {
        /* success. display inquiry data */
    else {
        /* failure. process depending on cdb_status */
    }
}
```

第二个示例将通过特定 LUN 路径尝试 SCSI **INQUIRY** 命令：

```
#include <sys/scsi.h>

esctl_io_t esctl_io;
#define MAX_LEN 255
unsigned char inquiry_data[MAX_LEN];
```

```
memset(&esctl_io, 0, sizeof(esctl_io));        /* clear reserved fields */
esctl_io.flags = SCTL_READ | SCTL_IO_LPT;  /* input data
                                               * expected and commmand
                                               * to be sent on given
                                               * LUN path
                                               */
memcpy(&esctl_io.lpt_hwp, lpt_hwp, sizeof(lpt_hwp); /* specify
                                               * the hardware path of
                                               * LUN path through which
                                               * command must be sent
                                               */
esctl_io.cdb[0] = CMDinquiry;
esctl_io.cdb[1] = 0x00;
esctl_io.cdb[2] = 0x00;
esctl_io.cdb[3] = 0x00;
esctl_io.cdb[4] = MAX_LEN;                     /* allocation length */
esctl_io.cdb[5] = 0x00;
esctl_io.cdb_length = 6;                       /* 6 byte command */
esctl_io.data = &inquiry_data[0];              /* data buffer location */
esctl_io.data_length = MAX_LEN;                /* maximum transfer length */
esctl_io.max_msecs = 10000;                    /* allow 10 seconds for cmd */
if (ioctl(fildes, SIOC_IO_EXT, &esctl_io) < 0) {
      /* request is invalid */
} else {
      if ( esctl_io.cdb_status == S_GOOD) {
            /* success. display inquiry data */
      else {
            /* failure. process depending on cdb_status */
      }
}
```

以下示例将尝试执行 SCSI **TEST UNIT READY** 命令，然后检查设备是否就绪、未就绪还是处于其他某个状态。

```
#include <sys/scsi.h>

struct sctl_io sctl_io;

memset(&sctl_io, 0, sizeof(sctl_io));          /* clear reserved fields */
sctl_io.flags = 0;                             /* no data transfer expected */
sctl_io.cdb[0] = 0x00;                         /* can use CMDtest_unit_ready */
```

```
sctl_io.cdb[1] = 0x00;
sctl_io.cdb[2] = 0x00;
sctl_io.cdb[3] = 0x00;
sctl_io.cdb[4] = 0x00;
sctl_io.cdb[5] = 0x00;
sctl_io.cdb_length = 6;                 /* 6 byte command */
sctl_io.data = NULL;                    /* no data buffer is provided */
sctl_io.data_length = 0;                /* do not transfer data */
sctl_io.max_msecs = 10000;              /* allow 10 seconds for cmd */
if (ioctl(leg_fildes, SIOC_IO, &sctl_io) < 0) {
    /* request is invalid */
} else if (sctl_io.cdb_status == S_GOOD) {
    /* device is ready */
} else if (sctl_io.cdb_status == S_BUSY ||
     (sctl_io.cdb_status == S_CHECK_CONDITION &&
      sctl_io.sense_status == S_GOOD &&
      sctl_io.sense_xfer > 2 &&
      (sctl_io.sense[2] & 0x0F) == 2)) {
    /* can use sense_data */
    /* device is not ready */
} else {
    /* unknown state */
}
```

警告

不正确地使用 **scsi_ctl** 操作（甚至包括那些尝试访问不存在的设备的操作）可能会导致数据丢失、系统异常和设备损坏。

在尝试 **SIOC_IO** 命令之前，应使用 **SIOC_EXCLUSIVE** 读写控制获取对某个设备的独占访问权限。如果未获取独占访问权限， **SIOC_IO** 命令将与设备专用的驱动程序的命令混用，这可能导致不需要的结果。

设备专用的驱动程序可拒绝不当的或繁琐的 **SIOC_IO_EXT/SIOC_IO** 命令。但是，由于不一定已知或能检测到所有此类操作，因此在使用修改内部设备状态的命令时，应小心谨慎，以避免干扰设备专用的驱动程序。

大多数 SCSI 命令都具有一个逻辑单元号 (LUN) 字段。 HP-UX 操作系统上的并行 SCSI 实现方法将通过 SCSI **IDENTIFY** 消息选择逻辑单元。 cdb 的 LUN 部分通常应设置为 0，即使所访问的 LUN 不为 0 也是如此。

不支持使用链接的命令。

大多数带有数据阶段的 SCSI 命令需要在命令字节中的某个位置加入数据长度信息。要获取命令的正确结果，必须正确地指定 *data_length* 字段值和任何 cdb 数据长度值。

超时值过大（或无限）会导致并行 SCSI 总线（可能是整个系统）挂起。

如果超时过期，可使用设备和（或）总线重置来重新获取某个设备的关注。

重置设备可能导致 I/O 错误和（或）缓存数据丢失。这可能导致数据丢失和（或）系统异常。

一般来说，由于在驱动程序打开调用期间， HP-UX 操作系统上的 SCSI 实现将同步对查询数据的访问，所以，要获取 SCSI **INQUIRY** 数据，最好使用 **SIOC_INQUIRY** 读写控制，而不要使用 **SIOC_IO** 读写控制。

由于通信参数会受到设备特定的驱动程序功能的影响，因此使用设备特定的驱动程序可能会导致更改通信参数。

scsi_ctl 不支持 **SIOC_CAPACITY** 读写控制，这是因为功能的含义与特定的设备相关。

文件

/usr/include/sys/scsi.h
/usr/include/sys/scsi_ctl.h

另请参阅

mknod(1M)、 mksf(1M)、 ioctl(2)、 plock(2)、 privileges(5)、 intro(7)、 scsi(7)。

名称

scsi_disk - SCSI 直接访问设备驱动程序 (esdisk/sdisk)

说明

本节介绍通过字符专用设备驱动程序访问 SCSI 磁盘、CD-ROM 和光盘设备的接口。 **esdisk** 是自 HP-UX 11i v3 开始具有的直接访问设备的缺省驱动程序。 **sdisk** 是在 HP-UX 11i v2 和早期版本上使用的缺省驱动程序。保留该程序是为了向后兼容。

SCSI 直接访问设备内存储了一系列数据块。每个直接访问设备都具有指定的设备大小，其中包括许多数据块和一个逻辑块。所有数据块都具有相同的逻辑块大小。

由于 I/O 操作所使用的大小必须是整数个块，因此一个逻辑块大小是可能的最小 I/O 数量。可使用 **DIOC_DESCRIBE** 、 **DIOC_CAPACITY** 、 **SIOC_CAPACITY** 、 **DIOC_DESCRIBE_EXT** 和 **SIOC_STORAGE_CAPACITY** 读写控制确定设备块大小（请参阅 *disk*(7) 和 *scsi*(7)； **disc3** 上不支持 **SIOC_CAPACITY** ）。对于因介质尚未安装或其他原因而尚未准备就绪的直接访问设备，系统都将其解释为“设备大小为零”。对这种设备执行 **open()** 调用将会成功，但是后续的 **read()** 和 **write()** 调用将会失败。

ioctl(2) 联机帮助页说明了这些操作和实际参数的使用方式。请注意，所使用的 *arg* 通常是特定读写控制 **#define** 语句中引用的形式参数的地址。有关示例代码，请参阅“举例”一节。

为提高性能，许多 SCSI 磁盘设备都具有可用于读取和写入操作的缓存。

读取缓存的使用称为“预读”，它能使磁盘驱动器在预计有读取请求时读取数据。对用户而言，预读仅在所产生的增强性能方面有明显的优势。

写入缓存的使用称为“即时报告”。即时报告可通过在将写入的数据实际提交给介质之前，报告一个已完成写入的状态，来提高 I/O 性能。如果后续的物理写入操作未成功完成，则数据可能会丢失。

通过在磁盘驱动器中使用自动隔离功能，可以很大程度地排除由于介质缺陷而导致的物理写入失败。在即时报告和介质提交之间发生电源故障可能会导致丢失缓存的数据。但是，通常这些事件之间的时间段相对较短，因此不太可能发生这种丢失。

可使用 **SIOC_GET_IR** 读写控制确定设备当前是否使用了即时报告功能。值 **1** 指示启用了即时报告。零值指示禁用了即时报告。可使用 **SIOC_SET_IR** 读写控制来启用或禁用即时报告。零值将禁用即时报告。值 **1** 将启用即时报告。

可使用 **SIOC_SYNC_CACHE** 读写控制将设备中缓存的数据强制到介质。

大多数 SCSI 可移动介质磁盘设备支持“prevent”和“allow”介质移动命令。为避免数据损坏和数据可访问性问题，在可移动介质磁盘设备打开的整个期间，将防止出现介质移动。由于介质移动不受支持，因此 **SIOC_MEDIUM_CHANGED** 读写控制也不受支持。

头文件 **<sys/scsi.h>** 提供了有关直接访问设备控制的有用信息，其中包括：

```
/* ioctl support for SCSI disk devices */
#define SIOC_GET_IR                    _IOR('S', 14, int)
```

```
#define SIOC_SET_IR             _IOW('S', 15, int)
#define SIOC_SYNC_CACHE         -IOW('S', 70, int)
```

SIOC_FORMAT ioctl 可重新格式化整个介质面。执行重新格式化之前，为确保不影响其他应用程序，需要通过 **DIOC_EXCLUSIVE** 读写控制（请参阅 *disk*(7) ）获取对设备的独占访问权限。可使用 **fmt_optn** 字段选择所需的介质几何结构。大多数设备只支持一个介质几何结构。对于这些设备，应使用零值。对于支持多个介质几何结构的设备，还可以使用零值来选择缺省几何结构。可使用 interleave 字段指定扇区交错。零值指定应使用相应的缺省交错。

举例

以下示例代码说明如何使用可影响 **scsi_disk** 的读写控制。

```
#include <stdio.h>
#include <fcntl.h>
#include <sys/errno.h>
#include <sys/diskio.h>
#include <sys/scsi.h>
Describe_ext(dfd)
 int dfd;
{
  int ret;
  disk_describe_type_ext_t disk_descr;
  uint64_t  capacity;

  if ((ret = ioctl (dfd, DIOC_DESCRIBE_EXT, &descr_type)) != 0) {
     exit(1);
  }
  printf("\nSuccessful ioctl DIOC_DESCRIBE_EXT \n");
  printf("  model number: %s\n", disk_descr.model_num);
  printf("  interface:    %d  <20=scsi>\n", disk_descr.intf_type);
  capacity = (disk_descr.maxsva_high << 32) + disk_descr.low_lba;
  printf("  Capacity:    %llu (blocks)\n", capacity);
  printf("  block size:  %u (bytes)\n", disk_descr.lgblksz);
  printf("  Device type:  %u (0=disk, 5=CD, 7=OM)\n",
         disk_descr.dev_type);
  printf("  Write Protected:  %s \n",
         disk_descr.flags & WRITE_PROTECT_FLAG ? "yes" : "No");

}

Describe (dfd)
```

```
  int dfd;
{
  int ret;
  disk_describe_type descr_type;
  if ((ret = ioctl (dfd, DIOC_DESCRIBE, &descr_type)) != 0) {
    exit(1);
  }
  printf ("\nSuccessful ioctl DIOC_DESCRIBE \n");
  printf ("  model number: %s\n", descr_type.model_num);
  printf ("  interface:    %d  <20=scsi>\n", descr_type.intf_type);
}
Exclusive (dfd)
  int dfd;
{
  int ret, flag=1;
  if ((ret = ioctl (dfd, DIOC_EXCLUSIVE, &flag)) != 0) {
    exit(1);
  }
}
Enable_WOE (dfd)
  int dfd;
{
  int ret, flag=1;
  if ((ret = ioctl (dfd, SIOC_WRITE_WOE, &flag)) != 0) {
    exit(1);
  }
  printf ("\nSuccessful ioctl SIOC_WRITE_WOE \n");
}
main (argc, argv)
  int argc;
  char ** argv;
{ int ret, fd; if (argc != 2) {
    printf ("Usage: %s <disk_device> \n", argv[0]);
    exit(1);
  }
  if ((fd = open (argv[1], O_RDWR)) < 0) {
    exit (1);
  }
  Describe_ext(fd);
```

```
 Describe (fd);
 Exclusive (fd);
 Enable_WOE (fd);
}
```

警告

从历史角度来说，磁盘设备的块大小都比较小（通常为 512 字节）；但是，有许多新的磁盘设备（例如光盘和磁盘阵列）具有相对较大的块大小。使用直接原始裸磁盘访问的应用程序应使用 **DIOC_DESCRIBE** 、**DIOC_CAPACITY**、 **DIOC_DESCRIBE_EXT** 或 **SIOC_CAPACITY** 读写控制确定相应的最小 I/O 大小。

当磁盘设备处于打开状态时，移动和插入介质是不受支持的，并且其结果是不可预测的。不要尝试规避介质移动的预防机制。将无法识别由于这种干预而导致的设备容量更改。

通常，I/O 操作大小越大，效率应该越高。但是，相对于设备的缓存来说太大的 SCSI 磁盘 I/O 操作可能会导致缓存空间不足，从而使设备无法保持介质全速的数据传输速率。相对于较小的 I/O 大小来说，这可能会导致 I/O 性能降低。

相关内容

光盘设备

SIOC_VERIFY_WRITES 读写控制控制写入模式。通常，假定写入的数据正确地存储在介质上。可通过验证写入模式来验证写入的数据，以确保数据已正确写入。验证会极大地降低写入性能，因此通常并不需要使用。

可使用 **SIOC_VERIFY_WRITES** 读写控制来启用或禁用写入验证。零值将禁用写入验证。值 **1** 将启用写入验证。尽管写入验证主要用于光学介质，但是有些系统可能也支持对普通磁盘设备进行写入验证。

SIOC_VERIFY 读写控制可验证某个介质区是否包含有效的数据（即正确写入的数据）。尝试读取时，验证的介质不会导致 I/O 错误。可通过 **start_lba** 和 **block_cnt** 字段指定要验证的介质区。尽管验证主要用于光学介质，但是有些系统可能也支持对普通磁盘设备进行验证操作。

SIOC_WRITE_WOE 读写控制控制用于磁光盘设备的写入模式。通常，磁光学写入操作需要进行两次物理磁头传递。第一次传递将清除要写入的介质区。第二次传递将实际写入数据。写入且不清除模式可跳过第一次（清除介质区）传递，因此可以极大地提高写入性能。为确保产生正确的数据，最重要的一点是，只能在已知为空白的介质（事先已清除或从未使用过）上执行写入且不清除操作。可使用 **SIOC_WRITE_WOE** 读写控制来启用或禁用写入且不清除模式。零值将禁用写入且不清除模式。零 **1** 将启用写入且不清除模式。

可通过 **SIOC_ERASE** 读写控制显式清除介质区。可通过 **start_lba** 和 **block_cnt** 字段指定要清除的介质区。可使用写入且不清除模式写入以这种方式清除的介质区。请注意，清除的介质区与写入了某些数据值（例如，零）的介质区是不相同的。不应读取清除的介质区。尝试读取清除的介质区通常会导致 I/O 错误。

SIOC_VERIFY_BLANK 读写控制可验证是否清除了某个介质区，并且是否可以使用写入且不清除模式写入该介质区。可通过 **start_lba** 和 **block_cnt** 字段指定要验证的介质区。

以下特定光盘设备的信息摘自 **<sys/scsi.h>** ：

```
#define SIOC_WRITE_WOE          _IOW('S', 17, int)
#define SIOC_VERIFY_WRITES      _IOW('S', 18, int)
```

```
#define SIOC_ERASE              _IOW('S', 19, struct scsi_erase)
#define SIOC_VERIFY_BLANK       _IOW('S', 20, struct scsi_verify)
#define SIOC_VERIFY             _IOW('S', 21, struct scsi_verify)

/* structure for SIOC_ERASE ioctl */
struct scsi_erase {
      unsigned int    start_lba;
      unsigned short  block_cnt;
};

/* structure for SIOC_VERIFY_BLANK and SIOC_VERIFY ioctls */
struct scsi_verify {
      unsigned int    start_lba;
      unsigned short  block_cnt;
};
```

文件

/usr/include/sys/scsi.h

另请参阅

mediainit(1)、 mknod(1M)、 ioctl(2)、 disk(7)、 scsi(7)。

名称

scsimgr_eschgr - 用于 scsimgr 的 SCSI 类驱动程序 eschgr 插件程序

说明

用于 **scsimgr** 的 SCSI 类驱动程序 **eschgr** 插件程序实现特定于绑定到 **eschgr** 驱动程序的设备类的管理和诊断操作。 **eschgr** 是本地 HP-UX SCSI 类驱动程序，缺省情况下处理所有库/更换器设备。

该插件程序为驱动程序 **eschgr** 处理下列操作：

- 显示和清除驱动程序 **eschgr** 的全局统计数据，以及该驱动程序维护的有关绑定到它的 LUN 实例和相关 LUN 路径的统计数据。
- 显示由驱动程序 **eschgr** 维护的有关绑定到它的 LUN 的状态和其他信息。
- 获取、设置和保存驱动程序 **eschgr** 的全局属性和每个 LUN 实例的属性。

命令

用户可以通过指定 **-d eschgr** 选项将下列 **scsimgr** 命令显式发送到驱动程序 **eschgr** 插件程序：

clear_stat	清除统计数据。
get_attr	显示有关属性的信息。
get_info	显示状态和其他信息。
get_stat	显示统计数据。
save_attr	将属性值保存在持久存储中。
set_attr	设置当前的属性值。

注释：有关以上命令的语法，请参考 *scsimgr*(1M) 。

但是，仅当对驱动程序 **eschgr** 的以下全局对象执行操作时，才需要将命令显式发送给此插件程序：全局统计数据、属性或状态信息。在所有其他情况下，当对绑定到驱动程序 **eschgr** 的 LUN 或其 LUN 路径执行操作时，**scsimgr** 将自动调用此插件程序以执行操作的驱动程序特定部分。

属性

下表列出了驱动程序 **eschgr** 特定的属性。有关属性概念的详细信息，请参考 *scsimgr*(1M) 。

注释：使用下列约定：

- RO 表示只读。
- RW 表示读写。
- uint32 表示 32 位无符号整数。

对象	属性名称	**RO/RW**	类型	说明
全局	version	RO	string	驱动程序 **eschgr** 的版本

LUN	default_secs	RW	uint32	未引用的以下所有命令的超时。 缺省值：30
	move_secs	RW	uint32	移动命令的超时。 缺省值：1200
	readelem_secs	RW	uint32	读取元素状态命令的超时。 缺省值：600
	initelem_secs	RW	uint32	初始化元素状态命令的超时。 缺省值：600
	readaddr_secs	RW	uint32	modesense 0x1D 命令的超时。 缺省值：600
	exchange_secs	RW	uint32	交换命令的超时。 缺省值：600

I/O 负载平衡和多路径策略

eschgr 不支持负载平衡，并且对多路径的支持也十分有限。

在系统引导后首次打开设备时，将选择一个路径，并将保持固定。如果该路径失败，则下次打开设备时将选择一个新路径。

举例

显示 **scsimgr eschgr** 插件程序的通用帮助和支持的命令

scsimgr -h -d eschgr

获取 **eschgr** 驱动程序的全局统计数据：

scsimgr get_stat -d eschgr

清除 **eschgr** 驱动程序的全局统计数据：

scsimgr clear_stat -d eschgr

获取 **eschgr** 驱动程序的全局状态信息：

scsimgr get_info -d eschgr

显示有关 **eschgr** 驱动程序的全局属性的信息：

scsimgr get_attr -d eschgr

作者

用于 **scsimgr** 的 SCSI 类驱动程序 **eschgr** 插件程序由 HP 开发。

另请参阅

scsictl(1M)、 scsimgr(1M)、 autochanger(7)、 intro(7)、 scsi(7)。

名称

scsimgr_esdisk - 用于 scsimgr 的 SCSI 类驱动程序 esdisk 插件程序

说明

用于 **scsimgr** 的 SCSI 类驱动程序 **esdisk** 插件程序实现特定于绑定到 **esdisk** 驱动程序的设备类的管理和诊断操作。 **esdisk** 是本地 HP-UX SCSI 类驱动程序，缺省情况下处理包括直接访问、CD/DVD、一次写入多次读取 (WORM) 和光存储器 (OM) 等类型在内的所有块设备。

该插件程序为驱动程序 **esdisk** 处理下列操作：

- 显示和清除驱动程序 **esdisk** 的全局统计数据以及该驱动程序维护的有关绑定到它的 LUN 实例和相关 LUN 路径的统计数据。
- 显示由驱动程序 **esdisk** 维护的有关绑定到它的 LUN 的状态和其他信息。
- 获取、设置和保存驱动程序 **esdisk** 的全局属性、每个 LUN 实例的属性和绑定到该驱动程序的设备集的属性。

命令：

用户可以通过指定 **-d esdisk** 选项将下列 **scsimgr** 命令显式发送到驱动程序 **esdisk** 插件程序：

clear_stat 清除统计数据。

get_attr 显示有关属性的信息。

get_info 显示状态和其他信息。

get_stat 显示统计数据。

save_attr 将属性值保存在持久存储中。

set_attr 设置当前的属性值。

注释：有关以上命令的语法，请参考 *scsimgr*(1M) 。

但是，仅当对驱动程序 **esdisk** 的全局对象（全局统计数据、属性或状态信息）执行操作时，才需要向该插件程序显式发送命令。在所有其他情况下，当对绑定到驱动程序 **esdisk** 的 LUN 或其 LUN 路径执行操作时， **scsimgr** 将自动调用此插件程序以执行操作的驱动程序特定部分。

属性

下表列出了驱动程序 **esdisk** 特定的属性。此外，在类别“设备集”下，它列出 **esdisk** 驱动程序可在各范围内设置的属性；包括设备类型、供应商标识符、产品标识符以及产品修订版本。有关属性和属性范围的概念，请参考 *scsimgr*(1M)。

注释：使用下列约定：

- RO 表示只读。
- RW 表示读写。

- uint32 表示 32 位无符号整数。
- 列出适用属性的值范围。

对象	属性名称	**RO/RW**	类型	说明
全局	version	RO	string	驱动程序 **esdisk** 的版本
LUN	capacity	RO	uint32	设备容量（以块为单位）
	block_size	RO	uint32	块大小（以字节为单位）
	path_fail_secs	RW	uint32	在第一次 I/O 失败后，声明 LUN 路径处于脱机状态之前的延迟（以秒为单位） 范围：0-600
	load_bal_policy	RW	string	I/O 负载平衡策略 可以为： * **round_robin** * **least_cmd_load** * **cl_round_robin** * **preferred_path**
	infinite_retries _enable	RW	boolean	启用或禁用无限的 I/O 重试功能。 可以为： * true：启用 * false：禁用
	preferred_path	RW	string	将 I/O 负载平衡策略设置为 preferred_path，I/O 传输优先选择的 lunpath 的硬件路径。
设备集	transient_secs	RW	uint32	在 LUN 转换为 **ONLINE** 之外的状态后，在发生 I/O 失败之前等待的秒数。 范围：0-600
	format_secs	RW	uint32	SCSI 命令 FORMAT 的超时（以秒为单位）。 范围：0-0xFFFFFFFF
	start_unit_secs	RW	uint32	SCSI 命令 START UNIT 的超时（以秒为单位）。 范围：0-0xFFFFFFFF
	max_retries	RW	uint32	最大 I/O 重试次数 范围：1-0xFFFFFFFF
	path_fail_secs	RW	uint32	声明 LUN 路径处于脱机状态之前的超时（以秒为单位） 范围：0-600
	esd_secs	RW	uint32	传输 I/O 的最长时间（以秒为单位） 范围：0-0xFFFFFFFF

max_q_depth	RW	uint32	最大队列深度 范围：1 - 0xFFFFFFFE
load_bal_policy	RW	string	I/O 负载平衡策略 可以为： * **round_robin** * **least_cmd_load** * **cl_round_robin** * **preferred_path**
disable_flags	RW	string	一组表示 SCSI 任务管理及其他功能的标记。如果设置了某个标记，则将对设备集禁用对应的功能。 当前定义了下列标记： * **WCE**：写入缓存启用 * **RW16**：**16** 字节 **READ/WRITE CDB** * **ABT**：**SCSI** 任务管理功能中止任务设置 * **CTS**：**SCSI** 任务管理功能清除任务设置 * **LR**：**SCSI** 任务管理功能 **LUN** 重置 * **WTR**：**SCSI** 任务管理功能热重置目标 * **CTR**：**SCSI** 任务管理功能冷重置目标 * **BR**：总线重置 * **PR**：永久保留 * **WERO**：永久保留 **WERO (Write Exclusive Read-Only)** * **AERO**：永久保留 **AERO (Access Exclusive Read-Only)**
infinite_retries _enable	RW	Boolean	启用或禁用无限的 I/O 重试。 可以为： * true：启用 * false：禁用

I/O 负载平衡策略

I/O 负载平衡策略属性 **load_bal_policy** 是一个控制如何在 LUN 的路径之间分配 I/O 的可调参数：

- **round_robin**

  以循环方式选择路径。当设备的所有路径具有相似的 I/O 循环特性时，该参数将更为合适。

- **least_cmd_load**

  选择活动的 I/O 请求数最少的 LUN 路径来执行下一个 I/O。当 LUN 路径表现出不对称延迟特性时，该策略将比较合适。分配负载以优化每个 LUN 路径上的带宽。

- **cl_round_robin**（可识别单元的循环）

此负载平衡策略适用于基于单元的 HP 平台。在启动了 I/O 的 CPU 的位置内以循环方式选择 LUN 路径，以确保优化内存访问延迟。

- **preferred_path**

  I/O 传输优先选择 preferred_path 属性中设置的 I/O 路径。如果该 I/O 路径不可用，或未设置 preferred_path 属性，则 I/O 传输将选择任何其他路径。该策略对于某些磁盘阵列（这些磁盘阵列同时通过 LUN 的多个 I/O 路径传输 I/O 时可能出现性能下降）很有用。

举例

显示 **scsimgr esdisk** 插件程序的通用帮助和支持的命令：

scsimgr -h -d esdisk

获取 esdisk 驱动程序的全局统计数据

scsimgr get_stat -d esdisk

清除 esdisk 驱动程序的全局统计数据

scsimgr clear_stat -d esdisk

获取 esdisk 驱动程序的全局状态信息

scsimgr get_info -d esdisk

显示有关 esdisk 驱动程序的全局属性的信息

scsimgr get_attr -d esdisk

将 disk0 的负载平衡策略设置为 preferred_path，并设置优先选择的 I/O 路径

scsimgr set_attr -D /dev/rdisk/disk0 -a load_bal_policy=preferred_path
-a preferred_path=0/3/1/0.0x21000020371972eb.0x0

添加对应于来自 HP 并且产品标识符为“MSA VOLUME”的所有磁盘的可设置属性范围，以便允许在此范围修改某些可设置属性

scsimgr ddr_add -N "/escsi/esdisk/0x0/HP /MSA VOLUME "

永久更改来自 HP 并且产品标识符为“MSA VOLUME”的所有磁盘的缺省 I/O 负载平衡策略、I/O 超时和最大并行 I/O

scsimgr save_attr -N "/escsi/esdisk/0x0/HP /MSA VOLUME "
-a load_bal_policy=least_cmd_load -a esd_secs=60
-a path_fail_secs=60

禁用绑定到 **esdisk** 驱动程序的所有磁盘的写入缓存、永久保留和 16 字节读/写 CDB

scsimgr set_attr -N /escsi/esdisk -a disable_flags='WCE PR RW16'

作者

用于 **scsimgr** 的 SCSI 类驱动程序 **esdisk** 插件程序由 HP 开发。

另请参阅

scsimgr(1M)、diskinfo(1M)、scsictl(1M)、intro(7)、scsi(7)、scsi_disk(7)。

名称

scsimgr_estape - 用于 scsimgr 的 SCSI 类驱动程序 estape 插件程序

说明

用于 **scsimgr** 的 SCSI 类驱动程序 **estape** 插件程序实现特定于绑定到驱动程序 **estape** 的设备类的管理和诊断操作。 **estape** 是本地 HP-UX SCSI 类驱动程序，缺省情况下处理所有磁带设备。

该插件程序为驱动程序 estape 处理下列操作：

- 显示和清除驱动程序 **estape** 的全局统计数据，以及该驱动程序维护的有关绑定到它的 LUN 实例和相关 LUN 路径的统计数据。
- 显示由驱动程序 **estape** 维护的有关绑定到它的 LUN 的状态和其他信息。
- 获取、设置和保存驱动程序 **estape** 的全局属性和每个 LUN 实例的属性。

命令

用户可以通过指定 **-d estape** 选项将下列 scsimgr 命令显式发送到驱动程序 **estape** 插件程序：

clear_stat 清除统计数据。

get_attr 显示有关属性的信息。

get_info 显示状态和其他信息。

get_stat 显示统计数据。

save_attr 将属性值保存在持久存储中。

set_attr 设置当前的属性值。

注释：有关上述命令的语法，请参考 *scsimgr*(1M) 。

但是，仅当对驱动程序 **estape** 的以下全局对象执行操作时，才需要将命令显式发送给此插件程序：全局统计数据、属性或状态信息。在所有其他情况下，当对绑定到驱动程序 estape 的 LUN 或其 LUN 路径执行操作时，**scsimgr** 将自动调用此插件程序以执行操作的驱动程序特定部分。

属性

下表列出了驱动程序 estape 特定的属性。有关属性概念的详细信息，请参考 *scsimgr*(1M)。

注释：使用下列约定：

- RO 表示只读。
- RW 表示读写。
- VBM 表示变量块模式。
- uint32 表示 32 位无符号整数。
- uint64 表示 64 位无符号整数。
- 列出适用属性的值范围。

对象	属性名称	**RO/RW**	类型	说明
全局	version	RO	string	驱动程序 **estape** 的版本
	norewind_ close_disable	RW	uint32	禁用打开“rewind”设备的功能。 缺省值：0 值：0（禁用） 1（启用）
	st_ats_enabled	RW	uint32	确定是否在打开时保留设备，以及是否在关闭时释放设备。请参阅 *st_ats_enabled*(5)。 缺省值：1 值：0（禁用） 1（启用）
LUN	default_secs	RW	uint32	未引用的以下所有命令的超时。 缺省值：30
	space_secs	RW	uint32	空间命令的超时。 缺省值：1200
	write_secs	RW	uint32	写入命令的超时。 缺省值：600
	read_secs	RW	uint32	读取命令的超时。 缺省值：600
	unload_secs	RW	uint32	卸载命令的超时。 缺省值：600
	rewind_secs	RW	uint32	倒带命令的超时。 缺省值：600
	erase_secs	RW	uint32	清除命令的超时。 缺省值：18000

mt_type	RW	uint32	与某特定 LUN 关联的设备类型。 缺省值：0 值: 0 = 未知 5 = 9 轨 HPIB 6 = DDS1 7 = 所有其他 DDS/DAT 8 = 9 轨 Scsi 9 = QIC 10 = 8mm 11 = IBM 3480、STK 9XXX、STK T10000 12 = Quantum DLT 13 = Sony AIT 14 = IBM 3590 15 = LTO
default_blocksize	RW	uint32	缺省块大小。 缺省值：0 0 = 由定制 DSF 覆盖的变量。
default_ir	RW	uint32	缺省的即时报告。 缺省值：1 值：0（禁用） 1（启用）
close_marks	RW	uint32	表示数据结尾的文件标记的数目。 缺省值：2
num_partitions _supp	RW	uint32	支持的分区数 缺省值：1 范围：1+
characteristics	RW	uint64	驱动程序特性按位逻辑 OR 运算 缺省值：0 1 = 设备支持设置标记 2 = Logpage 31 包含容量信息 4 = Logpage 38 包含容量信息 8 = 设备支持保留/释放

best_density	RW	uint32	写入的磁带密度。 缺省值：0x7f 0xFFFFFFFF = 最佳密度 0x00 = 由设备选择密度 0x7f = 不修改磁带密度。 other = 模式参数块描述符的有效密度代码。
best_compression	RW	uint32	要使用的压缩算法 缺省值：0 0x00 = 禁用压缩 0xDEF = 驱动器的缺省值 other = 数据压缩模式页 (0x0F) 的有效压缩值。
clean_req_ sns_info	RW	uint32	表示“需要清除”的关键字/代码/限定符 缺省值：0xffffffff

I/O 负载平衡和多路径策略

estape 驱动程序不支持负载平衡，并且对多路径的支持也十分有限。

在系统引导后首次打开设备时，将选择一个路径，并将保持固定。如果该路径失败，则下次打开设备时将选择一个新路径。

举例

显示 **scsimgr estape** 插件程序的通用帮助和支持的命令：

scsimgr -h -d estape

获取 **estape** 驱动程序的全局统计数据：

scsimgr get_stat -d estape

清除 **estape** 驱动程序的全局统计数据：

scsimgr clear_stat -d estape

获取 **estape** 驱动程序的全局状态信息：

scsimgr get_info -d estape

显示有关 **estape** 驱动程序的全局属性的信息：

scsimgr get_attr -d estape

作者

用于 **scsimgr** 的 SCSI 类驱动程序 **esctape** 插件程序由 HP 开发。

另请参阅

scsimgr(1M)、 scsictl(1M)、 st_ats_enabled(5)、 intro(7)、 scsi(7)、 scsi_tape(7)。

名称

scsi_tape - SCSI 顺序访问设备驱动程序

说明

SCSI 顺序访问（磁带）设备可以存储一系列数据块。可以使用固定大小或可变大小的块模式读取和写入数据。如果设备支持可变大小的块模式，则通常使用该模式（即使所有块大小相同）。通常仅对于不支持可变大小块的磁带设备，使用固定大小的块模式。在一些支持可变大小块的磁带设备上，可以使用固定大小的块模式来提高 I/O 性能。

通常，通过 **mt** （请参阅 *mt*(7) ）常规磁带设备接口控制 SCSI 磁带设备。本节介绍 SCSI 磁带设备特定的功能。

可以使用 **SIOC_CAPACITY** 读写控制（请参阅 *scsi*(7) ），确定某些磁带设备的剩余磁带容量。 **blksz** 字段指明设备的“固有”块大小。该值可能是设备的当前块大小，也可能不是。由 **lba** 字段指明的块数，是剩余介质上可写入的数据量的估算值。对于未提供剩余容量信息的设备，将返回零大小。实际可写入的数据量可能高于或低于所指明的值，这取决于块大小、介质缺陷、数据压缩以及维护流的能力等因素。

为了提高性能，大多数 SCSI 磁带设备都具有缓存。读取缓存的使用（称为“预读”）可使磁带机在预计有读取请求时读取数据。对用户而言，预读仅在所产生的增强性能方面有明显的优势。写入缓存的使用称为“即时报告”。在将所写入的数据实际提交给介质之前，即时报告通过报告已完成的写入状态，可提高 I/O 性能。这样，应用程序可以提供其他数据，从而可实现连续介质运动，称为“流”。可以使用 **SIOC_GET_IR** 读写控制确定设备当前是否正在使用即时报告功能。值为 1 表示已启用即时报告。缺省情况下，设备驱动程序将尝试启用即时报告。可以使用 **SIOC_SET_IR** 读写控制明确启用或禁用即时报告。值为 0 将禁用即时报告。值为 1 将启用即时报告。可以使用 **MTIOCTOP** 读写控制 **MTNOP** 命令将任何缓存的数据写入（提交）到介质。请注意，由 **SIOC_SET_IR** 读写控制设置的设备即时报告模式将在执行 **close()** 和 **open()** 调用之间继续存在，但在系统重新引导期间则不会继续存在。

SIOC_GET_BLOCK_SIZE 读写控制表示设备的当前块大小。块大小为 0 表示设备处于可变大小的块模式下。块大小不为 0 表示设备处于固定大小的块模式下。

SIOC_SET_BLOCK_SIZE 读写控制将当前块大小更改为指定的字节数。将块大小设置为 0 指定应使用可变大小的块模式。任何不为 0 的块大小指定应使用固定大小的块模式。缺省情况下，设备驱动程序将尝试在打开时将块大小设置为 0。如果设备不支持可变大小的块模式，则驱动程序将选择适当的块大小供固定大小的块模式使用。请注意，由 **SIOC_SET_BLOCK_SIZE** 读写控制设置的设备块大小将在执行 **close()** 和 **open()** 调用之间继续存在，但在系统重新引导期间则不会继续存在。

SIOC_GET_BLOCK_LIMITS 读写控制指明设备的最大和最小固定块大小限制。设备的最小固定块大小由 **min_blk_size** 字段指明。 **max_blk_size** 字段包含设备支持的最大块大小和系统支持的最大块大小 (**MAXPHYS**) 二者之间的较小值。对于所使用的设备、驱动程序和主机系统的特定组合而言，这是最大的有效块大小。

使用 **SIOC_GET_POSITION** 读写控制可以确定某些设备的当前介质位置。对于支持该功能的设备，使用所得到的值可以在将来将介质重新定位到相同位置。

使用 **SIOC_SET_POSITION** 读写控制可以在某些设备上重新定位介质。对于支持该功能的设备，通过该机制完成重新定位介质通常可比使用记录、文件标记或设置标记间隔来以类似方式完成重新定位更快。指定的参数值应

为介质卷的先前 **SIOC_GET_POSITION** 操作的结果。

以下内容摘自 **<sys/scsi.h>** :

```
/* ioctl support for SCSI tape commands */
#define SIOC_GET_IR                 _IOR('S', 14, int)
#define SIOC_SET_IR                 _IOW('S', 15, int)
#define SIOC_GET_BLOCK_SIZE         _IOR('S', 30, int)
#define SIOC_SET_BLOCK_SIZE         _IOW('S', 31, int)
#define SIOC_GET_BLOCK_LIMITS       _IOW('S', 32, struct scsi_block_limits)
#define SIOC_GET_POSITION           _IOR('S', 33, int)
#define SIOC_SET_POSITION           _IOW('S', 34, int)

/* structure for SIOC_GET_BLOCK_LIMITS ioctl */
struct scsi_block_limits {
    unsigned min_blk_size;
    unsigned max_blk_size;
};
```

警告

SCSI 总线和设备重置会导致某些设备将介质重新定位到磁带开头 (BOT)。这种无意的介质重新定位行为会导致数据丢失。 **scsi_tape** 驱动程序会使第一个后续的 **open()** 尝试失败，以此表明可能丢失数据。

如果已经通过编程方式重新定位介质，则 **scsi_tape** 驱动程序关闭时将不写入文件标记。在关闭设备之前重新定位介质的应用程序，应写入所有需要的磁带标记。

另请参阅

mknod(1M)、 mt(7)、 scsi(7)。

名称

sioc_io - SCSI 通透式接口

说明

SCSI 设备由设备专用的驱动程序（如果有的话）控制。诸如适用于 SCSI 直接访问（磁盘）和顺序访问（磁带）设备的驱动程序等设备专用驱动程序可以协调设备与驱动程序状态，以便完成正确的逻辑设备行为。 **sioc_io** 通透式接口可启用通常这些设备专用的驱动程序不支持的 SCSI 设备和命令。它包括两个读写控制： **SIOC_IO_EXT** 和 **SIOC_IO** 。

SIOC_IO_EXT 是 HP-UX 11i v3 版本中介绍的通透式接口。建议使用该接口。应在持久性设备文件上执行该接口（请参阅 *intro*(7) ）。该接口允许通过任何可用的 LUN 路径或特定的 LUN 路径发送 SCSI 命令。

SIOC_IO 是存在于 HP-UX 11i V3 之前版本中的通透式接口。不建议在 HP-UX 11i V3 版本中使用该接口。保留该接口是为了向后兼容。该接口可用于持久性设备路径或 Legacy 设备路径。如果在持久性设备文件上执行，则通过任何可用的 LUN 路径发送 SCSI 命令。如果在 Legacy 设备文件上执行，则将通过任何可用的 LUN 路径发送 SCSI 命令。但是，如果 Legacy 设备文件禁用多路径（请参阅 *scsimgr*(1M) 中的 **leg_mpath_enable** ），将会仅通过与 Legacy 设备文件相对应的 LUN 路径发送 SCSI 命令。

与接口相关联的数据结构中的所有保留字段值必须为零。

通过 **SIOC_IO_EXT/SIOC_IO** 读写控制，可以向设备发送任意 SCSI 命令。 SCSI 命令协议的所有详细信息是自动处理的。

SIOC_IO_EXT/SIOC_IO 读写控制的数据结构摘自 **<sys/scsi.h>** ：

```
/* SCSI device control ioctls */

#define SIOC_IO_EXT             _IOWR('S', 102, esctl_io_t)
#define SIOC_IO                 _IOWR('S', 22, struct sctl_io)

/* Structure for SIOC_IO_EXT ioctl */
typedef struct {
    int                     version;
    escsi_sctl_io_flags_t   flags;
    int                     max_msecs;
    uint32_t                cdb_length;
    uint32_t                data_length;
    ptr64_t                 data;
    union sense_data        sense;
    escsi_hw_path_t         lpt_hwp;
    uint32_t                data_xfer;
    uint32_t                sense_xfer;
    uint32_t                cdb_status;
```

```
    uint32_t            sense_status;
    uint8_t             cdb[ESCSI_MAX_CDB_LEN];
    uint32_t            rsvd[32];  /* Reserved for future use */
} esctl_io_t;

/* Structure for SIOC_IO ioctl */
typedef struct sctl_io {
    unsigned            flags;
    unsigned char       cdb_length;
    unsigned char       cdb[16];
    void                *data;
    unsigned            data_length;
    unsigned            max_msecs;
    unsigned            data_xfer;
    unsigned            cdb_status;
    unsigned char       sense[256];
    unsigned            sense_status;
    unsigned char       sense_xfer;
    unsigned char       reserved[64];
} sctl_io_t;
```

除非另行指明，否则可以使用下列标记指定 **SIOC_IO_EXT** 和 **SIOC_IO** 的 **flags** 字段值：

SCTL_READ 如果 *data_length* 字段不为零，则应该是数据输入阶段。如果 *data_length* 字段不为零，则缺少该标记表示应该是数据输出阶段。

ESCTL_IO_LPT 在给定的 LUN 路径上执行 SCSI 命令。只能使用 **SIOC_IO_EXT** 读写控制指定该标记。指定时，要使用的 LUN 路径的硬件路径在字段 *lpt_hwp* 中指定。

cdb 字段指定 SCSI 命令字节。命令字节数由 *cdb_length* 字段指定。在 SCSI 命令阶段，会将这些命令字节发送到目标设备。

SCSI 命令数据阶段所用数据区地址由 **data** 字段指定。 **data_length** 字段指定要传输的最大数据字节数。 *data_length* 值为 0 表示不应该出现数据阶段。大多数带有数据阶段的 SCSI 命令需要在命令字节中的某个位置包含数据长度信息。调用方负责正确指定 *data_length* 字段和任何 *cdb* 数据长度值。长度不能大于 **SCSI_MAXPHYS** ，而且一些实现还会限制该长度。

max_msecs 字段指定设备完成命令所需的最长时间（以毫秒为单位）。如果命令尚未完成，但此时间段已过期，则系统可能会尝试恢复过程，以重新获得设备的关注。这些恢复过程可能包括异常中止标记、异常中止操作以及设备和总线重置操作。 *max_msec* 字段中的零值表示超时时间是无限的，因此系统应无限期地等待命令完成。

当 **SIOC_IO_EXT/SIOC_IO** 读写控制调用返回时，所有命令处理都已完成。大多数 **SIOC_IO_EXT/SIOC_IO** 读

写控制调用将返回 0（成功）。应该使用所得到的读写控制详细数据从调用方的角度评估“成功”或“失败”。**cdb_status** 字段指示 **cdb** 命令的结果。如果 *cdb_status* 字段指示 **S_CHECK_CONDITION** 状态，则 *sense_status* 字段指示用于收集相关 sense 数据的 SCSI **REQUEST SENSE** 命令的结果。这些状态字段将包含下列值之一：

SCTL_INVALID_REQUEST	SCSI 命令请求无效，因此未尝试此请求。
SCTL_SELECT_TIMEOUT	目标设备对主机 SCSI 接口所作的选择没有应答（该设备不存在或没有响应）。
SCTL_INCOMPLETE	设备对选择作出了应答，但是命令未完成（设备所用时间过长或通信失败）。
S_GOOD	设备成功完成了命令。
S_CHECK_CONDITION	设备指明的 sense 数据可用。
S_CONDITION_MET	设备成功完成了命令，并且请求的（搜索或预提取）操作得到满足。
S_BUSY	设备指明因其正忙于进行其他操作而无法接受命令。
S_INTERMEDIATE	设备成功完成了该命令，它是一系列链接命令中的一个命令（不受支持，请参阅“警告”一节）。
S_I_CONDITION_MET	设备指明 **S_INTERMEDIATE** 和 **S_CONDITION_MET**（不受支持，请参阅“警告”一节）。
S_RESV_CONFLICT	设备指明命令与现有保留相冲突。
S_COMMAND_TERMINATED	设备指明命令在早期已被主机系统终止。
S_QUEUE_FULL	设备指明因其命令队列当前已满而无法接受命令。

data_xfer 字段指明在 **cdb** 命令的数据阶段期间实际传输的数据字节数。仅当 *cdb_status* 字段包含下列值之一时该字段有效：**S_GOOD** 或 **S_CHECK_CONDITION**。*sense_xfer* 字段指明有效 sense 数据字节数。仅当 *cdb_status* 字段包含值 **S_CHECK_CONDITION** 且 *sense_status* 字段包含值 **S_GOOD** 时，该字段有效。

安全性限制

SIOC_IO 读写控制的使用要求超级用户、 **DEVOPS** 权限或设备写入权限。有关对支持精细划分权限系统的授权访问的详细信息，请参阅 *privileges*(5)。

举例

假定 *fildes* 是某个 SCSI 设备持久性设备文件的有效文件描述符， *leg_fildes* 是某个 SCSI 设备 Legacy 设备文件的有效文件描述符，并且 *lpt_hwp* 包含一个指向该设备的 LUN 路径的有效硬件路径。第一个示例将尝试执行 SCSI **INQUIRY** 命令：

```
#include <sys/scsi.h>

esctl_io_t esctl_io;
#define MAX_LEN 255
unsigned char inquiry_data[MAX_LEN];
```

```
memset(&esctl_io, 0, sizeof(esctl_io)); /* clear reserved fields */
esctl_io.flags = SCTL_READ;     /* input data expected */
esctl_io.cdb[0] = CMDinquiry;
esctl_io.cdb[1] = 0x00;
esctl_io.cdb[2] = 0x00;
esctl_io.cdb[3] = 0x00;
esctl_io.cdb[4] = MAX_LEN;                  /* allocation length */
esctl_io.cdb[5] = 0x00;
esctl_io.cdb_length = 6;                    /* 6 byte command */
esctl_io.data = &inquiry_data[0];           /* data buffer location */
esctl_io.data_length = MAX_LEN;             /* maximum transfer length */
esctl_io.max_msecs = 10000;                 /* allow 10 seconds for cmd */
if (ioctl(fildes, SIOC_IO_EXT, &esctl_io) < 0) {
      /* request is invalid */
} else {
      if ( esctl_io.cdb_status == S_GOOD) {
            /* success. display inquiry data */
      else {
            /* failure. process depending on cdb_status */
      }
}
```

第二个示例将通过特定 LUN 路径尝试 SCSI **INQUIRY** 命令：

```
#include <sys/scsi.h>

esctl_io_t esctl_io;
#define MAX_LEN 255
unsigned char inquiry_data[MAX_LEN];
memset(&esctl_io, 0, sizeof(esctl_io));       /* clear reserved fields */
esctl_io.flags = SCTL_READ | SCTL_IO_LPT;  /* input data
                                            * expected and command
                                            * to be sent on given
                                            * LUN path
                                            */
memcpy(&esctl_io.lpt_hwp, lpt_hwp, sizeof(lpt_hwp); /* specify
                                            * the hardware path of
                                            * LUN path through which
                                            * command must be sent
                                            */
```

```
esctl_io.cdb[0] = CMDinquiry;
esctl_io.cdb[1] = 0x00;
esctl_io.cdb[2] = 0x00;
esctl_io.cdb[3] = 0x00;
esctl_io.cdb[4] = MAX_LEN;              /* allocation length */
esctl_io.cdb[5] = 0x00;
esctl_io.cdb_length = 6;                /* 6 byte command */
esctl_io.data = &inquiry_data[0];       /* data buffer location */
esctl_io.data_length = MAX_LEN;         /* maximum transfer length */
esctl_io.max_msecs = 10000;             /* allow 10 seconds for cmd */
if (ioctl(fildes, SIOC_IO_EXT, &esctl_io) < 0) {
    /* request is invalid */
} else {
    if ( esctl_io.cdb_status == S_GOOD) {
        /* success. display inquiry data */
    else {
        /* failure. process depending on cdb_status */
    }
}
```

以下示例将尝试执行 SCSI **TEST UNIT READY** 命令，然后检查设备是否就绪、未就绪还是处于其他某个状态。

```
#include <sys/scsi.h>

struct sctl_io sctl_io;

memset(&sctl_io, 0, sizeof(sctl_io));   /* clear reserved fields */
sctl_io.flags = 0;                      /* no data transfer expected */
sctl_io.cdb[0] = 0x00;                  /* can use CMDtest_unit_ready */
sctl_io.cdb[1] = 0x00;
sctl_io.cdb[2] = 0x00;
sctl_io.cdb[3] = 0x00;
sctl_io.cdb[4] = 0x00;
sctl_io.cdb[5] = 0x00;
sctl_io.cdb_length = 6;                 /* 6 byte command */
sctl_io.data = NULL;                    /* no data buffer is provided */
sctl_io.data_length = 0;                /* do not transfer data */
sctl_io.max_msecs = 10000;              /* allow 10 seconds for cmd */
if (ioctl(leg_fildes, SIOC_IO, &sctl_io) < 0) {
    /* request is invalid */
```

```
}
else if (sctl_io.cdb_status == S_GOOD) {
      /* device is ready */
}
else if (sctl_io.cdb_status == S_BUSY ||
       (sctl_io.cdb_status == S_CHECK_CONDITION &&
        sctl_io.sense_status == S_GOOD &&
        sctl_io.sense_xfer > 2 &&
        (sctl_io.sense[2] & 0x0F) == 2)) {
       /* can use sense_data */
      /* device is not ready */
} else {
      /* unknown state */
}
```

警告

sioc_io 操作（即使是对不存在设备的尝试性访问）使用不当会导致数据丢失、系统异常及设备损坏。

在尝试 **SIOC_IO** 命令之前，应使用 **SIOC_EXCLUSIVE** 读写控制获取对某个设备的独占访问权限。如果未获取独占访问权限， **SIOC_IO** 命令将与设备专用的驱动程序的命令混用，这可能导致不需要的结果。

设备特定的驱动程序可以拒绝不适当的或有问题的 **SIOC_IO** 命令。但是，由于并非所有此类操作都是已知的且可检测到的，因此，当使用修改内部设备状态的命令时，应当小心操作以免中断设备特定的驱动程序。

大多数 SCSI 命令都具有一个逻辑单元号 (LUN) 字段。 HP-UX 操作系统上的并行 SCSI 实现方法将通过 SCSI **IDENTIFY** 消息选择逻辑单元。 **cdb** 的 LUN 部分通常应设置为 0，即使所访问的 LUN 不为 0 也是如此。

不支持使用链接的命令。

大多数带有数据阶段的 SCSI 命令需要在命令字节中的某个位置加入数据长度信息。要获取命令的正确结果，必须正确地指定 *data_length* 字段值和任何 *cdb* 数据长度值。

超时值过大（或无限）会导致并行 SCSI 总线（可能是整个系统）挂起。

如果超时过期，可使用设备和（或）总线重置来重新获取某个设备的关注。

重置设备可能导致 I/O 错误和（或）缓存数据丢失。这可能导致数据丢失和（或）系统异常。

一般来说，由于在驱动程序打开调用期间， HP-UX 操作系统上的 SCSI 实现将同步对查询数据的访问，所以，要获取 SCSI **INQUIRY** 数据，最好使用 **SIOC_INQUIRY** 读写控制，而不要使用 **SIOC_IO** 读写控制。

由于通信参数会受到设备特定的驱动程序功能的影响，因此使用设备特定的驱动程序可能会导致更改通信参数。

文件

/usr/include/sys/scsi.h
/usr/include/sys/scsi_ctl.h

另请参阅

ioctl(2)、 privileges(5)、 intro(7)、 scsi(7)、 scsi_ctl(7)。

名称

slp_syntax - SLP 服务类型语法

说明

SLP API 希望在查询 SLP 服务信息以及注册和取消注册服务时传递服务类型信息。SLP API 还可接受 URL 格式的服务类型信息。

服务类型字符串包含以下信息。

服务类型的名称。

负责服务名称的命名机构。

服务类型字符串的形式为:

service:*abstract-type.naming-authority*:*concrete-type*

abstract-type 是简短的描述服务类型的描述性字符串。

naming-authority 是为该服务命名的组织名称。 *naming-authority* 为可选项，但是如果省略，就会假定 IANA 为命名机构，并且 IANA 会要求注册服务类型（请参阅 RFC 2609）。

concrete-type 也为可选项，它是抽象类型的一种子类型。

例如，

printer 是抽象类型（由 IANA 拥有）， **printer:lpr** 是具体类型（由 IANA 拥有）。

服务类型字符串的正式定义可在 RFC 2609“服务模板和服务方案”中找到。

服务类型字符串示例

weather.nasa:wtp	由 NASA 拥有的使用 WTP 协议的（虚构）天气服务类型。
weather.nasa:swtp	由 NASA 拥有的使用 SWTP 协议的（虚构）天气服务类型。
chat.superchat	由 SuperChat 拥有的交谈服务类型。
printer.samba	samba 打印机服务类型。
ftp	IANA ftp 服务类型。
telnet	IANA telnet 服务类型。

比较服务类型

由于服务类型在确定由 **SLPFindSrvs()** 调用返回的 URL 时非常重要，因此应该了解如何比较服务。假定使用 **SLPReg()** 注册了三项服务，使用的 *srvtype*

分别为 **printer:lpr** 、 **printer** 和 **printer.acme**。如果客户端程序使用 *srvtype* 为 **service:printer** 调用 **SLPFindSrvs()** ，则将返回 **printer:lpr** 和 **printer** 的 URL（不返回 printer.acme 的 URL）。但是，如果使用 *srvtype* 为 **printer:lpr** 或 **printer.acme**

调用 **SLPFindSrvs()** ，则返回 **printer:lpr** 或 **printer.acme** 的 URL。换句话说，如果使用了具体类型，则仅返回具有相同抽象和具体类型的服务。如果仅使用了抽象类型，则返回该抽象类型的所有服务（及命名机构）。

SLP 服务 **URL** 语法

SLP API 接受 URL 语法格式的服务类型字符串。 URL 字符串作为参数传递给 **SLPReg()** 、 **SLPDeReg()** 、 **SLPFindSrvs()** 和 **SLPParseSrvURL()** 函数，并作为 **SLPSrvURLCallback()** 回调函数的结果返回。 SLP 定义了一种名为“服务 URL”的特殊类型的 URL，在调用 SLP API 函数时必须使用该 URL。服务 URL 的语法是：

SLP Service URL = service:*service-type***://***addrspec*

service-type 是上面介绍的服务类型。 *addrspec* 可以是任何符合 URL 语法的地址，并且可以转换为网络地址。 **service:** 和 **://** 字符串是必需的。

服务 **URL** 示例

service:weather.nasa:wtp://weather.nasa.com:12000
service:weather.nasa:swtp://weather.nasa.com:12001
service:chat.superchat://chat.superchat.com;auth=ldap

SLP 要求使用服务 URL。如果不使用，则 API 函数将返回 **SLP_PARSE_ERROR** 。由于 SLP API 程序员不允许将服务类型作为 **SLPDeReg()** 调用的参数传入，因此必须使用服务 URL。如果没有服务类型，则 **SLPDeReg()** 将不允许调用方区分用相同标准 URL 注册的各种类型的服务。

SLPFindSrvs() 函数希望搜索字符串用 LDAPv3 搜索过滤语法传递。

另请参阅

slpd(1M)、libslp(3N)、slp.reg(4)。

名称

socket - 进程间通信

说明

套接字是允许进程进行本地或远程通信的通信端点。通过一系列系统调用，可以访问这些套接字（请参阅 *socket*(2) ）。

下列 **ioctl()** 请求在 **<sys/ioctl.h>** 中定义（请参阅 *ioctl*(2) ）：

FIOSNBIO 如果地址为 *arg* 的整数不为零，则将套接字置于非阻塞模式。否则，将套接字置于阻塞模式。缺省为阻塞模式。尽管不建议使用 **FIONBIO** ，但是 **FIONBIO** 请求等效于 **FIOSNBIO** 请求。有关如何使用非阻塞模式的说明，请参阅 *accept*(2) 、 *connect*(2) 、 *recv*(2) 和 *send*(2) 。

FIONREAD 对于 SOCK_STREAM 套接字，将以地址为 *arg* 的整数形式返回该套接字中的当前可读字节数。对于 SOCK_DGRAM 套接字，将以地址为 *arg* 的整数形式，返回当前可读的字节数以及 *sockaddr* 结构大小（在 **<sys/socket.h>** 中定义）。

SIOCATMARK 对于 SOCK_STREAM TCP 套接字，如果入站 TCP 流已读至带外数据字节开始的位置，则返回时，地址为 *arg* 的整数不为零。否则，入站 TCP 流尚未读至带外数据字节开始的位置。对于除 SOCK_STREAM TCP 套接字之外的套接字，返回时，地址为 *arg* 的整数始终为零。

SIOCSPGRP 该请求将与套接字相关联的进程组或进程 ID 设置为地址为 *arg* 的整数值。当套接字状态发生更改时，将按以下方式用信号通知与套接字相关联的进程组或进程 ID ： 当收到带外数据时，将传递 **SIGURG** ； 如果套接字是异步的，则传递 **SIGIO** ，如下面 **FIOASYNC** 中所述。如果地址为 *arg* 的整数值为正数，则信号将发送到其进程 ID 与指定值相匹配的进程。如果整数值为负数，则信号将发送到其进程组等于指定值的绝对值的所有进程。如果整数值为 0，则不向任何进程发送信号。有必要使用非零整数值发出该请求，以便启用所述的信号传递机制。进程组或进程 ID 的缺省值为零。

SIOCGPGRP 该请求将以地址为 *arg* 的整数形式返回与套接字相关联的进程组或进程 ID 。有关返回的整数值的含义的详细信息，请参阅上面的 **SIOCSPGRP** 说明。

FIOASYNC 如果地址为 *arg* 的整数不为零，则该请求将套接字的状态设置为异步。否则，套接字将置于同步模式（缺省）。如果满足下列任一条件，则异步模式将传递 **SIGIO** 信号。

- 新数据到达。
- 用于面向连接的协议，每当其他传出缓冲区空间变为可用时，或者连接建立或断开时。

与套接字相关联的进程组或进程 ID 必须为非零值，才能发送 **SIGIO** 信号。信号是根据所述的 **SIOCSPGRP** 语义传递的。

套接字支持 *fcntl*(2) **O_NDELAY** 和 **O_NONBLOCK** 标志（在 **<fcntl.h>** 中定义）。如果设置了 **O_NONBLOCK** 标志，则套接字将置于 POSIX样式的非阻塞模式。如果设置了 **O_NDELAY** 标志，则套接字将置于非阻塞模式。否则，套接字将置于阻塞模式。缺省为阻塞模式。有关如何使用这些形式的非阻塞模式的说明，请参阅 *accept*(2)、 *connect*(2) 、 *recv*(2) 和 *send*(2) 。

由于支持 **fcntl() O_NONBLOCK** 和 **O_NDELAY** 标志以及 **ioctl() FIOSNBIO** 请求，以下内容阐明了这些功能如何交互。如果已设置了 **O_NONBLOCK** 或 **O_NDELAY** 标志，则 **recv()** 和 **send()** 请求将进行相应操作，并忽略任何 **FIOSNBIO** 请求。如果 **O_NONBLOCK** 标志和 **O_NDELAY** 标志均未设置，则 **FIOSNBIO** 会请求控制 **recv()** 和 **send()** 的行为。

相关内容

仅适用于 **AF_CCITT**

对于 **af_ccitt** 套接字，仅定义了 **FIOSNBIO** 、 **FIONREAD** 、 **SIOCGPGRP** 和 **SIOCSPGRP ioctl()** 。

作者

socket 由加州大学伯克利分校开发。

另请参阅

fcntl(2)、 getsockopt(2)、 ioctl(2)、 socket(2)。

名称

streamio - STREAMS ioctl 命令

概要

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

int ioctl(int *fildes*, **int** *command*, **... /*** *arg* ***/);**

说明

STREAMS **ioctl** 命令是在流上执行各种控制函数的 **ioctl()** 系统调用的子集。

fildes 是引用流的打开文件描述符。 *command* 确定要执行的控制函数（如下所述）。 *arg* 表示此命令所需的其他信息。 *arg* 的类型取决于命令，但它通常为整数或指向 *command* 特定数据结构的指针。 *command* 和 *arg* 由流头部解释。可能会将这些参数的某些组合传递给流中的模块或驱动程序。

由于这些 STREAMS 命令是 **ioctl** 的子集，因此它们会受到针对 ioctl 所述的错误的影响。除这些错误外，如果在多路复用器下面链接 *fildes* 引用的流，或者如果 *command* 不是流的有效值，则调用将失败，并将 **errno** 设置为 [EINVAL] ，且不会处理控制函数。

此外，如 **ioctl** 中所述，STREAMS 模块和驱动程序可以检测错误。在这种情况下，模块或驱动程序会将错误消息发送到包含错误值的流头部。这会导致后续系统调用失败，并将 **errno** 设置为此值。

下列 **ioctl** 命令会指示错误值，并适用于所有 STREAMS 文件：

I_ATMARK 允许用户查看流头部读取队列上的当前消息是否被某个下行模块“标记”。 *arg* 确定当流头部读取队列上存在多个已标记消息时如何执行检查。它可能采用下列值：

ANYMARK 检查是否已标记消息。

LASTMARK 检查消息是否为队列上标记的最后一条消息。

如果同时设置了 **ANYMARK** 和 **LASTMARK** ，则 **ANYMARK** 将替换 **LASTMARK**。

如果满足标记条件，则返回值为 1；否则为 0。

I_CANPUT 检查某个带是否可写入。 *arg* 设置为上述的优先级带。如果优先级带 *arg* 是流控制的，则返回值为 0；如果带是可写入的，则返回值为 1；如果出现错误，则返回值为 −1。

I_CKBAND 检查在流头部读取队列上是否存在给定优先级带的消息。如果存在给定优先级的消息，则此操作返回 1；如果出现错误，则返回 −1。 *arg* 应该是包含上述优先级带的值的整数。

I_FDINSERT 从用户指定的缓冲区创建消息，添加有关其他流的信息并向下行发送消息。消息包含控制部分和可选的数据部分。要发送的数据部分和控制部分根据在单独缓冲区中的位置进行区分，如下所述。

arg 指向包含下列成员的 **strfdinsert** 结构体：

```
struct strbuf    ctlbuf;
struct strbuf    databuf;
long             flags;
int              fildes;
int              offset;
```

ctlbuf strbuf 结构体（请参阅 *putmsg*(2)）中的 *len* 字段必须设置为指针的大小加上要随消息发送的控制信息的字节数。 **strfdinsert** 结构体中的 *fildes* 指定其他流的文件描述符。 *offset* 必须是字对齐的，它指定超过控制缓冲区（ **I_FDINSERT** 将在其中存储指针）开头的字节数。对于与 **strfdinsert** 结构体中 *fildes* 对应的流，此指针将是驱动程序的读取队列结构的地址。 **databuf strbuf** 结构体中的 *len* 字段必须设置为要随消息发送的数据信息的字节数或零（如果没有要发送的数据部分）。

flags 指定要创建的消息类型。如果 *flags* 设置为 0，则创建普通（非优先级）消息；如果 *flags* 设置为 **RS_HIPRI** ，则创建高优先级消息。对于常规消息，如果流写入队列因内部流控制情况已满，则 **I_FDINSERT** 将会阻塞。对于高优先级消息， **I_FDINSERT** 在此情况下不会阻塞。对于常规消息，当写入队列已满且设置了 **O_NONBLOCK** 时， **I_FDINSERT** 不会阻塞。相反，它将失败，并将 **errno** 设置为 [EAGAIN] 。

除非因缺少内部资源受到保护，否则无论优先级高低或是否已指定 **O_NONBLOCK** ， **I_FDINSERT** 也将阻塞，并等待消息块可用。不发送部分消息。

如果与 **strfdinsert** 结构体中的 *fildes* 对应的流的流头部收到错误消息，则 **I_FDINSERT** 也可能失败。在这种情况下， **errno** 将设置为消息中的值。

I_FIND 将流上当前存在的所有模块的名称与在 *arg* 中指定的名称进行比较。如果模块存在，则命令返回值 1；如果模块不存在，则返回值 0（零）。

I_FLUSH 此请求可刷新所有输入和（或）输出队列，具体取决于 *arg* 的值。有效的 *arg* 值包括：

- **FLUSHRW** 刷新写入队列和读取队列。
- **FLUSHW** 刷新写入队列。
- **FLUSHR** 刷新读取队列。

如果管道或 FIFO 未压入任何模块，则将根据 *arg* 的值刷新任一端上流头部的读取队列。

如果设置了 **FLUSHR** ，而且 *fildes* 是管道，则刷新管道这一端的读取队列，并刷新另一端的写入队列。如果 *fildes* 为 FIFO，则刷新这两个队列。

如果设置了 **FLUSHW** ，而 *fildes* 是管道，且该管道的另一端存在，则刷新管道另一端的读取队列，并刷新这一端的写入队列。如果 *fildes* 为 FIFO，则刷新 FIFO 的两个队列。

如果设置了 **FLUSHRW** ，则刷新所有读取队列，即刷新 FIFO 的读取队列和管道两端的读取队列。

已压入模块的管道或 FIFO 的正确刷新处理是通过 **pipemod** 模块完成的。此模块应该是压入到管道上的第一个模块，以便它位于管道本身的中点。

I_FLUSHBAND

刷新特定消息带。 *arg* 指向具有下列成员的 **bandinfo** 结构体：

```
unsigned char     bi_pri:
int               bi_flag;
```

bi_flag 字段的值可以是针对 **I_FLUSH** 命令所述的 **FLUSHR** 、 **FLUSHW** 或 **FLUSHRW** 。

I_GETBAND 返回 *arg* 引用的整数中流头部读取队列上第一条消息的优先级带。

I_GETCLTIME 返回 *arg* 指定的长整数中的关闭时间延迟。

I_SETCLTIME 允许用户在流正在关闭且写入队列上存在数据时设置流头部将延迟的时间。在关闭每个模块和驱动程序之前，流头部将延迟指定的时间以允许数据流失。如果延迟之后数据仍然存在，则将刷新数据。 *arg* 是指向要延迟的毫秒数（四舍五入至系统上最接近的有效值）的指针。缺省值为 15 秒。

I_GETSIG 返回调用进程已为其注册以接收 **SIGPOLL** 信号的事件。事件以类似为 **I_SETSIG** 命令定义的 *arg* 位掩码形式返回。

I_GRDOPT 在参数 *arg* 指向的 *int* 中返回当前的读取模式设置。读取模式在 *read*(2) 中介绍。

I_GWROPT 在参数 *arg* 指向的 *int* 中返回当前的写入模式设置（如 **I_SWROPT** 中所述）。

I_LINK 连接两个流，其中 *fildes* 是连接到多路复用驱动程序的流的文件描述符， *arg* 是连接到其他驱动程序的流的文件描述符。由 *arg* 指定的流在多路复用驱动程序下进行连接。 **I_LINK** 要求多路复用驱动程序将确认消息发送到与链接操作有关的流头部。如果成功，则此调用返回多路复用器 ID 号（用于与多路复用器断开连接的标识符，请参阅 **I_UNLINK** ）；如果失败，则返回 −1。

I_LIST 允许用户列出流上的所有模块名称，直到并包括最顶部的驱动程序名称。如果 *arg* 为 **NULL** ，则返回值是模块数，其中包括位于 *fildes* 指向的流上的驱动程序。这样，用户就可以为模块名称分配足够的空间。如果 *arg* 不为 **NULL** ，则它应该指向具有下列成员的 **str_list** 结构体：

```
int     sl_nmods;
struct  str_mlist  *sl_modlist;
```

str_mlist 结构体具有下列成员：

```
char  l_name[FMNAMESZ+1];
```

sl_nmods 指示用户已在数组中分配的条目数。如果成功，则返回值为 0； **sl_modlist** 包含模块名称的列表， **sl_nmods** 指示已填写的条目数。

I_LOOK 检索 *fildes* 所指向流的流头部之下紧邻的模块的名称，并将其放置在 *arg* 指向的以空字符结尾的字符串中。 *arg* 指向的缓冲区的长度至少应为 **FNAMESZ+1** 字节。需要进行 **#include <stropts.h>** 声明。

I_NREAD 统计流头部读取队列上第一条消息的数据块中的数据字节数，并将此值放置在 *arg* 指向的位置中。命令的返回值是流头部读取队列上的消息数。例如，如果在 *arg* 中返回零，但 **ioctl** 的返回值大于零，则这表示队列上的下一条消息为零长度消息。

I_PEEK 允许用户进程查看（窥视）流头部读取队列上第一条消息的内容。执行该操作并不清除队列中的消息。 **I_PEEK ioctl** 的操作方式与 **getmsg()** 函数相同，但它不删除消息。 *arg* 参数指向具有下列成员的 **strpeek** 结构（在 **<stropts.h>** 头文件中）：

```
struct strbuf   ctlbuf;
struct strbuf   databuf;
long            flags;
```

ctlbuf 和 **databuf** 指向的 **strbuf** 结构具有下列成员：

```
int     maxlen;
int     len;
char    *buf
```

strbuf 结构体的 *maxlen* 字段必须指定要检索的控制或数据信息的字节数。可以将 *flags* 字段设置为 **RS_HIPRI** 或 0（零）。如果将此字段设置为 **RS_HIPRI** ，则 **I_PEEK ioctl** 将查找队列上的高优先级消息。如果将此字段设置为 0，则 **I_PEEK ioctl** 将查看队列上的第一条消息。

如果检索到消息，则 **I_PEEK** 返回 1；如果找不到消息，则返回值 0（零）；它不会等待消息。成功完成后， **ctlbuf** 将指定控制缓冲区中的控制信息， **databuf** 指定数据缓冲区中的数据信息， *flags* 包含 **RS_HIPRI** 或 0（零）。

I_PLINK 连接两个流，其中 *fildes* 是连接到多路复用驱动程序的流的文件描述符， *arg* 是连接到其他驱动程序的流的文件描述符。由 *arg* 指定的流通过多路复用驱动程序下的持续链接进行连接。 **I_PLINK** 要求多路复用驱动程序将确认消息发送到与链接操作有关的流头部。此调用将创建持久性链接，即使关闭了与多路复用驱动程序的上行流关联的文件描述符，该链接也可以存在。如果成功，则此调用返回多路复用器 ID 号（可以用于与多路复用器断开连接的标识符，请参阅 **I_PUNLINK** ）；如果失败，则返回 −1。

如果 **I_PLINK ioctl** 正在等待多路复用驱动程序确认链接请求，或在 *fildes* 的流头部处收到错误 (**M_ERROR**) 消息或挂起 (**M_HANGUP**) 消息，则它也可能失败。此外， **M_IOACK** 或 **M_IONAK** 消息中可能会返回错误。如果出现这些情况，则 **I_PLINK** 失败，并将 **errno** 设置为消息中的值。

I_POP 删除 *fildes* 所指向流的流头部之下紧邻的模块。从管道中删除模块，要求在从删除它的一侧压入该模块。在 **I_POP** 请求中， *arg* 应为 0。

I_PUNLINK 将由通过持久性链接连接的 *fildes* 和 *arg* 指定的两个流断开连接。 *fildes* 是连接到多路复用驱动程序的流的文件描述符。 *arg* 是在多路复用驱动程序下链接流时由 **I_PLINK** 返回的多路复用器 ID 号。如果 *arg* 是 **MUXID_ALL** ，则将 *fildes* 的持久性链接的所有流断开连接。与在 **I_PLINK** 中相同，此命令要求多路复用驱动程序确认取消链接。

I_PUSH 将 *arg* 指向其名称的模块压入到当前流的顶部，紧邻流头部之下。如果流是管道，则将在管道两端的流头部之间插入模块。然后，它调用新压入模块的打开例行程序。

I_RECVFD 检索与 **I_SENDFD ioctl** 通过流管道发送的消息关联的文件描述符。 *arg* 是一个指针，它指向足以容纳包括下列成员的 **strrecvfd** 数据结构的数据缓冲区：

```
int     fd;
uid_t   uid;
gid_t   gid;
char    fill[8]
```

fd 是整型文件描述符。 **uid** 和 **gid** 分别是发送流的用户 ID 和组 ID。

如果清除了 **O_NONBLOCK** ，则 **I_RECVFD** 将阻塞，直到流头部上出现消息为止。如果设置了 **O_NONBLOCK** ，则当流头部上不存在消息时， **I_RECVFD** 将失败，并将 **errno** 设置为 [EAGAIN] 。

如果流头部上的消息是 **I_SENDFD** 发送的消息，则为消息中包含的文件指针分配新的用户文件描述符。新的文件描述符放置在 **strrecvfd** 结构体的 *fd* 字段中。该结构体将复制至 *arg* 指向的用户数据缓冲区中。

I_SENDFD 请求与 *fildes* 关联的流将包含文件指针的消息发送到位于流管道另一端的流头部。文件指针对应于必须为打开文件描述符的 *arg* 。

I_SENDFD 将 *arg* 转换为对应的系统文件指针。它分配一个消息块，并在该块中插入文件指针。此外，还插入与发送进程关联的用户 ID 和组 ID。将此消息直接放置在其连接到的流管道的另一端上流头部的读取队列上。

I_SETCLTIME 允许用户进程当流正在关闭且写入队列包含数据时设置流头部延迟的时间。 *arg* 参数包含指向要延迟的毫秒数（四舍五入至系统上最接近的合法值）的指针。缺省时间为 15 秒。

在关闭 STREAMS 模块和驱动程序之前，流头部将延迟指定的时间。这样，写入队列上的数据可以流失。如果在延迟后写入队列上仍存在数据，则刷新队列。

I_SETSIG 在与 *fildes* 关联的流上发生特定事件时，通知流头部用户希望内核发出 **SIGPOLL** 信号（请参阅 *signal*(2) ）。 **I_SETSIG** 支持 STREAMS 中的异步处理功能。 *arg* 的值是指定应该通知用户的事件的位掩码。除非另有说明，否则它是下列常量的任意组合的按位

OR :

S_BANDURG 与 **S_RDBAND** 联合使用时，如果优先级消息到达流头部读取队列的前部，则将生成 **SIGURG** ，而不是 **SIGPOLL** 。

S_ERROR **M_ERROR** 消息已到达流头部。

S_HANGUP **M_HANGUP** 消息已到达流头部。

S_HIPRI 在流头部读取队列上存在高优先级消息。即使消息长度为零，也这样设置。

S_INPUT 除 **M_PCPROTO** 之外的任何消息都已到达流头部读取队列。保留此事件是为了与以前版本兼容。即使消息长度为零，也这样设置。

S_MSG 包含 **SIGPOLL** 信号的 STREAMS 信号消息已到达流头部读取队列的前部。

S_OUTPUT 紧邻流头部之下的写入队列不再是满的。这将通知用户队列上存在空间，可用于向下行发送（或写入）数据。

S_RDBAND 优先级带消息（带 > 0）已到达流头部读取队列。即使消息长度为零，也这样设置。

S_RDNORM 普通（非优先级）消息已到达流头部读取队列。即使消息长度为零，也这样设置。

S_WRBAND 在队列下行存在大于 0 的优先级带，并且是可写入的。这将通知用户在队列上存在空间，可用于向下行发送（或写入）优先级数据。

S_WRNORM 此事件与 **S_OUTPUT** 相同。

用户进程可通过将 *arg* 位掩码设置为值 **S_HIPRI** ，选择仅收到高优先级消息的通知。

希望接收 **SIGPOLL** 信号的进程必须使用 **I_SETSIG** ，显式地进行注册以接收这些信号。如果多个进程注册接收同一流上同一事件的信号，则在事件发生时，每个进程都将收到通知。

如果 *arg* 的值为零，则将取消注册调用进程，并且将不再接收 **SIGPOLL** 信号。

I_SRDOPT 使用参数 *arg* 的值设置读取模式（请参阅 *read*(2) ）。有效的 *arg* 值包括：

RNORM 字节流模式（缺省）。

RMSGD 忽略消息模式。

RMSGN 不忽略消息模式。

将 **RMSGD** 和 **RMSGN** 均设置为错误。 **RMSGD** 和 **RMSGN** 将覆盖 **NORM** 。

此外，通过在 *arg* 中设置下列标记，可以更改流头部处理控制消息的方式：

RPROTNORM 如果控制消息位于流头部读取队列的前部，则以 EBADMSG 使 **read** 失败。这是缺省行为。

RPROTDAT 在用户发出 **read** 时将消息的控制部分作为数据传递。

RPROTDIS 在用户发送 **read** 时，忽略消息的控制部分，传递任何数据部分。

I_STR 通过 *arg* 指向的数据构建内部 STREAMS **ioctl** 消息，并下行发送该消息。

提供此机制是为了将用户 **ioctl** 请求发送到下行的模块和驱动程序。它允许通过 **ioctl** 发送信息，并将由下行接收者上行发送的任何信息返回给用户。 **I_STR** 将阻塞，直到系统以肯定或否定的确认消息进行响应，或者直到某个时间段后请求“超时”为止。如果请求超时，则它将失败，并将 **errno** 设置为 ETIME。

在流上，最多只能有一个 **I_STR** 处于活动状态。进一步的 **I_STR** 调用将被阻塞，直到流头部处活动的 **I_STR** 完成为止。这些请求的缺省超时间隔为 15 秒。 **O_NONBLOCK** （请参阅 *open*(2) ）标记对此调用不起作用。

要向下行发送请求， *arg* 必须指向包含下列成员的 **strioctl** 结构体：

```
int     ic_cmd;
int     ic_timout;
int     ic_len;
char    *ic_dp;
```

ic_cmd 是用于下行的模块或驱动程序的内部 **ioctl** 命令； **ic_timout** 是在超时之前 **I_STR** 请求将等待确认的秒数（−1 =无限期， 0 = 使用缺省值，>0 = 使用指定的值）。缺省超时是无限期。 **ic_len** 是数据参数中的字节数， **ic_dp** 是指向数据参数的指针。 **ic_len** 字段有两个用途：在输入时，它包含传入的数据参数的长度；在从命令返回时，它包含返回给用户的字节数（ **ic_dp** 指向的缓冲区应该大到足以包含流中的任何模块或驱动程序可以返回的最大数据量）。流头部将 **strioctl** 结构指向的信息转换为内部的 **ioctl** 命令消息，并将其向下行发送。

I_SWROPT 使用参数 *arg* 的值设置写入模式。 *arg* 的合法位设置包括：

SNDZERO 如果出现写入 0 字节的情况，则向下行发送零长度消息。要在出现写入 0 字节的情况下不发送零长度消息，不得在 *arg* 中设置此位。

I_UNLINK 将由 *fildes* 和 *arg* 指定的两个流断开连接。 *fildes* 是连接到多路复用驱动程序的流的文件描述符。 *arg* 是由 **I_LINK** 返回的多路复用器 ID 号。如果 *arg* 是 **MUXID_ALL** ，则将链接到 *fildes* 的所有流断开连接。与在 **I_LINK** 中相同，此命令要求多路复用驱动程序

确认取消链接。

返回值

除非为命令指定不同的值，否则在成功时， STREAMS **ioctl()** 调用的返回值为 0（零）；失败时返回值为 −1（负一）。

错误

如果出现下列情况，则 STREAMS **ioctl** 命令将失败而不执行函数，并将 **errno** 设置为 [EINVAL]：

- 在多路复用驱动程序之下，链接了 *fildes* 引用的流。
- *command* 参数不是流的有效值。

此外，如果出现下列任一情况，则 STREAMS **ioctl** 命令将返回对应值:

I_ATMARK

[EINVAL] *arg* 包含非法值。

I_CANPUT

[EINVAL] *arg* 包含非法值。

I_CKBAND

[EINVAL] *arg* 包含非法值。

I_FDINSERT

[EINVAL] **strfdinsert** 结构体中的 *fildes* 参数是无效的打开文件描述符。

[EINVAL] 指针大小加上 *offset* 超过了通过 *ctlptr* 指定的缓冲区的 *len* 字段的值。

[EINVAL] *offset* 不指定数据缓冲区中正确对齐的位置。

[EINVAL] *flags* 包含未定义的值。

[EFAULT] *arg* 指向分配的地址空间之外，或者 **ctrlbuf** 或 **databuf** 位于分配的地址空间之外。

[EAGAIN] **ioctl** 请求因要创建非优先级消息而失败，设置了 **O_NONBLOCK** 选项，同时流的写入队列因内部流控制情况是满的。

[ENOSR] 由于 STREAMS 内存资源不足，因此无法为要创建的消息分配缓冲区。

[ENXIO] 在 **I_FDINSERT ioctl** 调用中 *fildes* 指定的流上，或 **strfdinsert** 中 *fildes* 指定的流上收到了挂起。

[ERANGE] 通过 **databuf** 指定的缓冲区的 *len* 字段的值，不在流上最顶部模块的数据包最小大小和最大大小指定的范围内。

[ERANGE] 通过 **databuf** 指定的缓冲区的 *len* 字段的值，大于消息数据部分的最大允许大小。

[ERANGE] 通过 **ctlbuf** 指定的缓冲区的 *len* 字段的值，大于消息控制部分的最大允许大小。

如果由 **strfdinsert** 结构体中 *fildes* 字段指定的流收到错误（**M_ERROR**）消息，则 **I_FDINSERT ioctl** 也可能失败。在这种情况下， **errno** 将设置为错误消息中的错误值。

I_FIND

[EINVAL] *arg* 不包含有效的模块名称。

[EFAULT] *arg* 指向分配的地址空间之外。

I_FLUSH

[ENOSR] 由于缺少 STREAMS 内存资源，无法为刷新操作分配缓冲区。

[EINVAL] *arg* 参数是无效值。

[ENXIO] 在 *fildes* 上收到了挂起。

I_FLUSHBAND

[EINVAL] *bi_pri* 参数值超过了最大带，或者 *bi_flag* 参数不是 **FLUSHR** 、 **FLUSHW** 或 **FLUSHRW** 。

I_GETBAND

[ENODATA] 在流头部读取队列上不存在任何消息。

I_GETSIG

[EINVAL] 用户进程未注册，无法接收 **SIGPOLL** 信号。

[EFAULT] *arg* 指向分配的地址空间之外。

I_GRDOPT

[EFAULT] *arg* 指向分配的地址空间之外。

I_LINK

[EAGAIN] 暂时无法分配存储以执行链接操作。

[EBADF] *arg* 参数不是有效的打开文件描述符。

[ENXIO] 在 *fildes* 上收到了挂起。

[EINVAL] *fildes* 引用的流不支持多路复用。

[EINVAL] *arg* 引用的文件不是流，或者在多路复用器下已链接了流。

[EINVAL] 链接操作将在生成的多路复用配置中产生“循环”。换句话说， *arg* 参数引用的驱动程序已在多个位置链接到此配置。

[ENOSR] 由于 STREAMS 内存资源不足，无法为此命令分配存储。

[ETIME] 超时之前在流头部上未收到确认消息。

如果在收到驱动程序确认之前，在 *fildes* 的流头部处收到了 **M_ERROR** 或 **M_HANGUP** 消息，则 **I_LINK ioctl** 也可能失败。此外，在 **M_IOCACK** 或 **M_IOCNAK** 消息中可能返回错误。如果出现这些情况，则 **I_LINK ioctl** 失败，并将 **errno** 设置为消息中的值。

I_LIST

[EINVAL] **sl_nmods** 小于 1。

[EAGAIN] 无法分配缓冲区。

I_LOOK

[EINVAL] 流中没有模块。

[EFAULT] *arg* 指向分配的地址空间之外。

I_NREAD

[EFAULT] *arg* 指向分配的地址空间之外。

I_PEEK

[EINVAL] *flags* 参数是非法值。

[EFAULT] *arg* 指向分配的地址空间之外，或者 **ctrlbuf** 或 **databuf** 位于分配的地址空间之外。

[EBADMSG] 要查看的消息对 **I_PEEK** 命令无效。

I_PLINK

[ENXIO] 在由 *fildes* 参数引用的流上收到了挂起。

[ETIME] 在流头部上收到确认消息之前产生了超时。

[EAGAIN] 暂时无法分配存储以执行链接操作。

[EBADF] *arg* 不是有效的打开文件描述符。

[EINVAL] *fildes* 引用的流不支持多路复用。

[EINVAL] *arg* 引用的文件不是流，或者在多路复用驱动程序下已进行链接。

[EINVAL] 链接操作将在生成的多路复用配置中产生“循环”。换句话说， *arg* 引用的驱动程序已在多个位置链接到该配置。

I_POP

[EINVAL] 流中没有模块。

[ENXIO] 取出的模块所返回的错误值。

[ENXIO] 在 *fildes* 上收到了挂起。

I_PUNLINK

[ENXIO] 在 *fildes* 上收到了挂起。

[ETIME] 在流头部上收到确认消息之前产生了超时。

[EAGAIN] 暂时无法分配存储以执行链接操作。

[EINVAL] *arg* 是无效的多路复用器 ID 号。

[EINVAL] *fildes* 是管道的文件描述符。

如果 **I_PUNLINK ioctl** 正在等待多路复用器确认取消链接请求，并在 *fildes* 的流头部上收到了错误 (**M_ERROR**) 消息或挂起 (**M_HANGUP**)，则它也可能失败。此外， **M_IOCACK** 或 **M_IOCNAK** 消息中可能会返回错误。如果出现这些情况，则 **P_UNLINK ioctl** 将失败，并将 **errno** 设置为消息中的值。

I_PUSH

[EINVAL] 使用了无效的模块名称。

[EFAULT] *arg* 指向分配的地址空间之外。

[ENXIO] 压入的模块所返回的错误值。压入已失败。

[ENXIO] 在 *fildes* 上收到了挂起。

I_RECVFD

[EAGAIN] 已设置 **O_NONBLOCK** 选项，并且流头部读取队列上不存在任何消息。

[EFAULT] *arg* 参数指向分配的地址空间之外。

[EBADMSG] 流头部读取队列上存在的消息不包含传递的文件描述符。

[EMFILE] 打开文件过多。不允许再打开文件描述符。

[ENXIO] 在 *fildes* 上收到了挂起。

I_SENDFD

[EAGAIN] 发送流的头部无法为文件指针分配消息块。

[EAGAIN] 接收流的头部的读取队列已满，无法接收消息。

[EBADF] *arg* 参数不是有效的打开文件描述符。

[EINVAL] *fildes* 参数未引用流。

[ENXIO] 在 *fildes* 上收到了挂起。

I_SETCLTIME

[EINVAL] *arg* 包含非法值。

I_SETSIG

[EINVAL] 用户进程未注册为接收 **SIGPOLL** 信号。

[EAGAIN] 无法分配存储信号请求的数据结构。

I_SRDOPT

[EINVAL] *arg* 包含非法值。

I_STR

[EINVAL] **ic_len** 字段小于 0（零）字节，或者大于消息数据部分的最大允许大小 (**ic_dp**)。

[EINVAL] **ic_timout** 字段小于 –1。

[EFAULT] *arg* 指向分配的地址空间之外，或者由 **ic_dp** 或 **ic_len** 指定的缓冲区区域位于分配的地址空间之外。

[ENOSR] 由于缺少 STREAMS 内存资源，无法为 **ioctl** 请求分配缓冲区。

[ENXIO] 在 *fildes* 引用的流上收到了挂起。

[ETIME] **ioctl** 请求在收到确认之前超时。

如果流头部收到指示错误 (**M_ERROR**) 或挂起 (**M_HANGUP**) 的消息，则 **I_STR ioctl** 也可能失败。此外， **M_IOCACK** 或 **M_IOCNAK** 消息中可能会返回错误。在这些情况下， **ioctl** 将失败，并将 **errno** 设置为消息中的错误值。

I_SWROPT

[EINVAL] *arg* 参数是非法值。

I_UNLINK

[ENXIO] 在 *fildes* 上收到了挂起。

[ETIME] 在流头部上收到确认消息之前产生了超时。

[EINVAL] *arg* 是一个无效的多路复用器 ID 号，或者在多路复用驱动程序下已链接 *fildes* 。

如果 **I_UNLINK ioctl** 正在等待多路复用器确认取消链接请求，并在 *fildes* 的流头部处收到错误 (**M_ERROR**) 消息或挂起 (**M_HANGUP**) ，则它也可能失败。此外， **M_IOCACK** 或 **M_IOCNAK** 消息中可能会返回错误。如果出现这种情况，则 **I_UNLINK ioctl** 将失败，并将 **errno** 设置为消息中的值。

另请参阅

close(2)、 fcntl(2)、 getmsg(2)、 ioctl(2)、 open(2)、 poll(2)、 putmsg(2)、 read(2)、 write(2)、 signal(5)。

名称

strlog - STREAMS 记录驱动程序

说明

用户级进程、 STREAMS 驱动程序和模块可通过 STREAMS 记录驱动程序执行错误日志记录和事件跟踪。这些任务是通过用户接口和内核接口完成的。此外， STREAMS 记录驱动程序可将错误日志记录和事件跟踪消息发送给网络跟踪和日志记录工具 (NetTL)（请参阅 *nettl*(1M) 、 *netfmt*(1M) 和 *nettlconf*(1M) ） 。

该驱动程序为用户级进程提供的接口是 **ioctl()** 系统调用和 STREAMS 消息格式的子集。这些进程可能是错误记录器、跟踪记录器或者生成错误或事件消息的其他用户进程。用户接口可从记录驱动程序收集日志消息，同时还从用户进程生成日志消息。

该驱动程序还可通过其函数调用接口从内核中的 STREAMS 驱动程序和模块接受日志消息。内核接口可将来自 STREAMS 驱动程序和模块的请求或调用输入到日志消息。

记录驱动程序接受的日志消息还被发送给 NetTL。可使用 NetTL 来控制要记录的消息类型，及格式化和过滤已记录的消息。

内核接口

STREAMS 驱动程序和模块可通过调用 **strlog** 函数生成日志消息。

```
#include <sys/strlog.h>

int strlog (mid, sid, level, flags, fmt [, value ]...);
short mid;
short sid;
char level;
ushort flags;
char *fmt;
int value;
```

mid 指定提交日志消息的驱动程序或模块的 STREAMS 模块 ID 号。

sid 指定与 *mid* 标识的 STREAMS 模块或驱动程序关联的次设备的子 ID 号。

level 指定从跟踪器筛选低级事件消息所使用的级别。

flags 包含可以在不同的组合中设置的多个标记。这些标记如下:

SL_ERROR 该消息用于错误记录器。

SL_TRACE 该消息用于跟踪器。

SL_CONSOLE 该消息将输出到控制台。

SL_FATAL 提供致命错误的通知。

SL_NOTIFY 生成一个将消息的副本发送到系统管理员的请求。

以下为其他标记。 **strerr** 或 **strace** 不使用这些标记。但是，按下面的“STREAMS-NetTL 链接”一节所述，可以使用这些标记将 STREAMS 消息映射到 NetTL 消息。

SL_WARN 该消息为警告。

SL_NOTE 该消息为注释。

fmt 是 **printf** 形式的格式字符串。它接受 **%x** 、 **%l** 、 **%o** 、 **%u** 、 **%d** 、 **%c** 和 **%s** 转换规范。

values 是格式字符串的数字参数或字符参数。无法指定参数的最大数目。

用户接口

用户进程可通过对 **/dev/strlog** 执行 **open()** 调用，来访问记录驱动程序。每次对设备执行 open 调用都将获取一个独立的流。进程打开 **/dev/strlog** 后，将指示这是错误记录器，还是跟踪记录器。该进程是通过使用 **strioctl** 结构的 **ic_cmd** 字段中的相应值，及使用 **trace_ids** 结构中的相应数据和控制信息，发出 **I_STR ioctl()** 系统调用来完成此操作的：

```
struct trace_ids {
   short   ti_mid;
   short   ti_sid;
   char    ti_level;
   short   ti_flags;
};
```

ic_cmd 的值包括：

I_ERRLOG 指示错误记录器。不需要 **trace_ids** 数据。

I_TRCLOG 指示跟踪记录器。必须包括由一个或多个 **trace_ids** 结构的数组组成的数据缓冲区。

如果 **trace_ids** 结构的任何字段包含值 -1，则 **/dev/strlog** 将接受在该字段中收到的任何值。否则，仅当 *mid* 和 *sid* 的值与它们在 **trace_ids** 结构中的对等值相同，并且消息的级别等于或小于 **trace_ids** 结构中的 **level** 值时， **strlog** 才接受消息。

一旦记录器进程发送了 **I_STR ioctl()** 调用，STREAMS 记录驱动程序便开始向该记录器进程发送与限制匹配的日志消息。记录器进程将通过 **getmsg()** 系统调用获取这些日志消息。传入该调用的消息的控制部分包括一个 **log_ctl** 结构：

```
struct log_ctl {
   short   mid;
   short   sid;
   char    level;
   short   flags;
   long    ltime;
```

```
    long    ttime;
    int     seq_no;
};
```

log_ctl 结构指示引导后在时钟周期内提交消息的 *mid* 、 *sid* 和 *level* 时间，1970 年 1 月 1 日以后对应的以秒表示的时间，以及序列号。提供 1970 年 1 月 1 日以后的以秒表示的时间的目的，是为了能够方便地计算消息的日期和时间。提供引导后时钟周期内的时间的目的，是为了能够确定日志消息的相对计时。

除错误记录器或跟踪记录器以外的用户进程可以将一个日志消息发送到 **strlog** 。驱动程序只接受该消息的控制部分内的 **log_ctl** 结构的 **flags** 和 **level** 字段，以及该消息的适当格式化的数据部分。如果消息的数据部分包含一个以 null（空）结尾的格式字符串，并且该字符串的结尾最多接上三个参数，其中每个参数压缩了一个单词，那么该数据部分将被适当地格式化。

为错误和跟踪日志记录流提供了一系列不同的序列号。这些序列号有助于跟踪消息的发送。编号序列中出现空缺表示记录器进程未成功发送这些编号。如果由于种种原因，记录器进程停止发送消息，则可能会发生这种情况（有关详细信息，请参阅 *strace*(1M) 和 *strerr*(1M) 命令的参考页）。消息的数据部分包含格式字符串（以 null 结尾）的文本，其后面最多接上三个参数。

STREAMS-NetTL 链接

STREAMS 错误日志记录和事件跟踪消息都被映射到 NetTL 日志记录消息，并发送到 NetTL。 NetTL 将消息分成四个日志类：DISASTER、ERROR、WARNING 和 INFORMATIVE。 NetTL 日志类是由 **flags** 根据以下规则确定的：

```
If (flags & SL_ERROR)       NetTL log class
then
  if (flags & SL_FATAL)       ====> DISASTER
  if (flags & SL_WARN)        ====> WARNING
  if (flags & SL_NOTE)        ====> INFORMATIVE
  otherwise                   ====> ERROR
else
  all messages                ====> INFORMATIVE
```

缺省情况下，只记录 DISASTER 和 ERROR 消息。可通过 **nettl** 命令或 **nettlconf** 命令（请参阅 *nettl*(1M) 和 *nettlconf*(1M) ）更改此设置。

NetTL 使用的 STREAMS 子系统为 **STREAMS** 。

netfmt 命令（请参阅 *netfmt*(1M) ）可将 NetTL 工具记录的消息格式化为可读的格式。 **netfmt** 接受可用于过滤 STREAMS 模块 ID 和子 ID 的过滤配置文件。 STREAMS 的过滤配置文件语法如下：

STREAMS *module_id sub_id*

module_id 和 *sub_id* 可以是一个十进制数，也可以是作为通配符的 * 。

返回值

除非另有说明，如果成功完成， **strlog ioctl()** 命令将返回值 0（零）。否则，将返回值 -1。

错误

如果出现以下任何情况， **strlog** 驱动程序的 **ioctl()** 命令会将 **errno** 设置为对应的值：

[ENXIO] **I_TRCLOG ioctl()** 调用不包含任何 **trace_ids** 结构。

[ENXIO] 无法识别 **I_STR ioctl()** 调用。

此驱动程序不返回用户进程发送的格式不正确的消息的任何错误。

举例

以下示例说明 **strlog** 接口的某些基本用途。

此示例代码段显示如何使用 STREAMS 将消息输出到控制台：

```
strlog(TMUX,minor(mydev),0,SL_CONSOLE|SL_FATAL,
   "TMUX driver (minor:%d) suffers resource shortage.",
   minor(mydev));
```

此代码示例显示用户进程如何使用 **ioctl()** 命令 **I_ERRLOG** 将其自身注册到 STREAMS 记录驱动程序。

```
struct strioctl iocerr:
int logfd;

if ((logfd = open("/dev/strlog", O_RDWR)) == -1) {
   printf("Cannot open /dev/strlog\n");
   exit(1);
}

iocerr.ic_cmd = I_ERRLOG;
iocerr.ic_timout = 0;
iocerr.ic_len = 0;
iocerr.ic_dp = NULL;
ioctl(logfd, I_STR, &iocerr);
```

此代码示例显示了将消息发送到 **strlog** 驱动程序的用户级进程。

```
struct strbuf control, data;
struct log_ctl log;
char *warning = "Fatal error for user level process";
int logfd;

if ((logfd = open("/dev/strlog", O_RDWR)) == -1) {
      printf("Cannot open /dev/strlog\n");
```

```
        exit(1);
}

control.len = control.maxlen = sizeof(log);
control.buf = (char *)&lc;

data.len = data.maxlen = strlen(warning);
data.buf = warning;

lc.level = 2;
lc.flags = SL_FATAL|SL_CONSOLE;

putmsg(logfd, &control, &data, 0);
```

以下示例说明如何使用 STREAMS 的 NetTL 工具。有关 NetTL 的常规用法，请参阅 *nettl*(1M) 、 *netfmt*(1M) 和 *nettlconf*(1M) 。NetTL 使用的 STREAMS 子系统为 **STREAMS** 。

netfmt 可接受作为命令参数的过滤配置文件。以下过滤配置文件示例用于格式化模块 ID 为 1，子 ID 为 100 的消息：

```
STREAMS    1    100
```

可以使用此过滤配置文件示例显示模块 ID 为 2，子 ID 为 101 的所有消息：

```
STREAMS    2    *
STREAMS    *    101
```

文件

/dev/strlog	指定复制接口。
<sys/strlog.h>	指定 STREAMS 日志记录的头文件。
<stropts.h>	指定 STREAMS 选项和 **ioctl()** 命令的头文件。

另请参阅

netfmt(1M)、 nettl(1M)、 nettlconf(1M)、 strace(1M)、 strerr(1M)、 getmsg(2)、 ioctl(2)、 open(2)、 putmsg(2)、 write(2)、 clone(7)、 streamio(7)。

名称

sttyv6：stty - 用于与 v6/PWB 兼容的终端接口

说明

这些例行程序尝试将 UNIX 分时系统第六版 (v6) 以及 PWB *stty* 和 *gtty* 调用映射到执行相同功能的当前读写控制中。映射并不是完美无缺的。转换功能的方式如下所述。在学习此条目之前，读者应先熟悉 **termio** 。

以下数据结构在包含文件 **<sgtty.h>** 中定义：

```
struct sgttyb {
    char  sg_ispeed;    /* input speed */
    char  sg_ospeed;    /* output speed */
    char  sg_erase;     /* erase character */
    char  sg_kill;      /* kill character */
    int  sg_flags;      /* mode flags */
}
```

在 **sgtty.h** 中定义的标记包括：

```
#define HUPCL      01
#define XTABS      02
#define LCASE      04
#define ECHO       010
#define CRMOD      020
#define RAW        040
#define ODDP       0100
#define EVENP      0200
#define ANYP       0300
#define NLDELAY    001400
#define TBDELAY    002000
#define CRDELAY    030000
#define VTDELAY    040000
#define BSDELAY    0100000
#define CR0        0
#define CR1        010000
#define CR2        020000
#define CR3        030000
#define NL0        0
#define NL1        000400
#define NL2        001000
#define NL3        001400
```

#define TAB0	0
#define TAB1	002000
#define NOAL	004000
#define FF0	0
#define FF1	040000
#define BS0	0
#define BS1	0100000

当执行 **stty** 命令 (*ioctl* **TIOCSETP**) 时，旧的 **sgttyb** 结构中的标记将映射到 **termio** 结构中的新等效项。然后，将执行 **TCSETA** 命令。

下表显示了旧 **sgttyb** 标记与当前 **termio** 标记之间的映射关系。请注意， **termio** 结构中包含的但在下面未提及的标记将被清除。

HUPCL（如果设置）设置 **termio** HUPCL 标记；
HUPCL（如果清除）清除 **termio** 标记；
XTABS（如果设置）设置 **termio** TAB3 标记；
XTABS（如果清除）清除 **termio** TAB3 标记；
TBDELAY（如果设置）设置 **termio** TAB1 标记；
TBDELAY（如果清除）清除 **termio** TAB1 标记；
LCASE（如果设置）设置 **termio** IUCLC、OLCUC 和 XCASE 标记；
LCASE（如果清除）清除 **termio** IUCLC、OLCUC 和 XCASE 标记；
ECHO（如果设置）设置 **termio** ECHO 标记；
ECHO（如果清除）清除 **termio** ECHO 标记；
NOAL（如果设置）设置 **termio** ECHOK 标记；
NOAL（如果清除）清除 **termio** ECHOK 标记；
CRMOD （如果设置）设置 **termio** ICRNL 和 ONLCR
　　标记；同时，如果设置了 CR1，则设置 **termio** CR1 标记；如果设置了 CR2，则设置 **termio** ONOCR 和 CR2 标记；
CRMOD（如果清除）
　　设置 **termio** ONLRET 标记；同时，如果设置了 NL1，则设置 **termio** CR1 标记；如果设置了 NL2，则设置 **termio** CR2 标记；
RAW（如果设置）设置 **termio**
　　CS8 标记，并清除 **termio** ICRNL 和 IUCLC 标记；同时，将分别向 MIN 和 TIME 分配缺省值 6 个字符和 0.1 秒钟；
RAW（如果清除）设置 **termio**
　　BRKINT、IGNPAR、ISTRIP、IXON、IXANY、OPOST、CS7、PARENB、ICANON、和 ISIG 标记；同时，将分别向控制字符 EOF 和 EOL 分配缺省值 control-D 和 null；
ODDP（如果设置）如果还设置了 EVENP，将清除 **termio**
　　INPCK 标记；否则，将设置 **termio** PARODD 标记；

VTDELAY（如果设置）设置 **termio** FFDLY 标记；
VTDELAY（如果清除）清除 **termio** FFDLY 标记；
BSDELAY（如果设置）设置 **termio** BSDLY 标记；
BSDELAY（如果清除）清除 **termio** BSDLY 标记。

此外，还设置 **termio** CREAD 位，并且，如果波特率为 110，则设置 CSTOPB 位。

如果使用 **TIOCSETP** ，则将 **sgttyb** 结构中的 *ispeed* 条目映射到 **termio** CBAUD 字段中的相应速率。 *erase* 和 *kill* **sgttyb** 条目将被映射到 **termio** 的清除和抹行字符。

当执行 **gtty** (*ioctl* **TIOCGETP**) 命令时，将首先执行 **termio TCGETA** 命令。然后，将所得到的 **termio** 结构映射到 **sgttyb** 结构中，随后将其返回给用户。

下表显示了如何将 **termio** 标记映射到旧的 **sgttyb** 结构中。请注意， **sgttyb** 结构中包含的但在下面未提及的所有标记将被清除。

HUPCL（如果设置）设置 **sgttyb** HUPCL 标记；
HUPCL（如果清除）清除 **sgttyb** HUPCL 标记；
ICANON（如果设置）设置 **sgttyb** RAW 标记；
ICANON（如果清除）清除 **sgttyb** RAW 标记；
XCASE（如果设置）设置 **sgttyb** LCASE 标记；
XCASE（如果清除）清除 **sgttyb** LCASE 标记；
ECHO（如果设置）设置 **sgttyb** ECHO 标记；
ECHO（如果清除）清除 **sgttyb** ECHO 标记；
ECHOK（如果设置）设置 **sgttyb** NOAL 标记；
ECHOK（如果清除）清除 **sgttyb** NOAL 标记；
PARODD（如果设置）设置 **sgttyb** ODDP 标记；
PARODD（如果清除）清除 **sgttyb** ODDP 标记；
INPCK（如果设置）设置 **sgttyb** EVENP 标记；
PARODD、INPCK（如果清除二者）设置 **sgttyb**
ODDP 和 EVENP 标记；
ONLCR（如果设置）设置 **sgttyb**
CRMOD 标记；同时，如果设置了 CR1，则设置 **sgttyb** CR1 标记；如果设置了 CR2，则设置 **sgttyb** CR2 标记；
ONLCR（如果设置）如果设置了 CR1，则设置 **sgttyb**
NL1 标记；如果设置了 CR2，则设置 **sgttyb** NL2 标记；
TAB3（如果设置）设置 **sgttyb** XTABS 标记；
TAB3（如果清除）清除 **sgttyb** XTABS 标记；
TAB1（如果设置）设置 **sgttyb** TBDELAY 标记；
TAB1（如果清除）清除 **sgttyb** TBDELAY 标记；

FFDLY（如果设置）设置 **sgttyb** VTDELAY 标记；
FFDLY（如果清除）清除 **sgttyb** VTDELAY 标记；
BSDLY（如果设置）设置 **sgttyb** BSDELAY 标记；
BSDLY（如果清除）清除 **sgttyb** BSDELAY 标记。

如果使用 **TIOCGETP** ，则将 **termio** CBAUD 字段映射到 **sgttyb** 结构的 *ispeed* 和 *ospeed* 条目中。另外，将 **termio** 的清除和抹行字符映射到 *erase* 和 *kill* **sgttyb** 条目中。

请注意，由于 **sgttyb** 和 **termio** 结构之间不存在一对一的映射关系，因此使用较早的 **TIOCSETP** 和 **TIOCGETP** 调用时可能会出现意外结果。因此，在所有将来代码中应分别使用当前等效项 **TCSETA** 和 **TCGETA** 来替换 **TIOCSETP** 和 **TIOCGETP** 。

警告

包含上述工具是为了帮助转换旧程序，因此，不应在新代码中使用。请使用 **termio** 中描述的接口。请注意，以上转换 不适用于从 UNIX 分时系统第 7 版（v7）移植的程序，因为某些 V7 标记以不同方式定义。

另请参阅

stty(2)、 termio(7)。

名称

TCP - Internet 传输控制协议

概要

#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/tcp.h>

s = socket(AF_INET, SOCK_STREAM, 0);
s = socket(AF_INET6, SOCK_STREAM, 0);

说明

TCP 协议提供了可靠且流量可控的双向数据传输。它是用于支持 **SOCK_STREAM** 套接字类型的字节流协议。TCP 可在两个对等实体之间构建虚电路。虚电路包括远程 Internet 地址、远程端口、本地 Internet 地址和本地端口。 IP 使用 Internet 地址在主机之间定向消息，并使用端口号标识特定主机上的 TCP 实体。

使用 TCP 的套接字可以是 主动的或者是 被动的。 **connect()** 可创建主动套接字，这些套接字可启动与被动套接字的连接（请参阅 *connect*(2) ）。要创建一个被动套接字，在将套接字绑定到 **bind()** 系统调用（请参阅 *listen*(2) 和 *bind*(2) ）后，请使用 **listen()** 系统调用。只有被动套接字才可使用 **accept()** 调用来接受传入的连接（请参阅 *accept*(2) ）。

被动套接字可能会 “模糊指定”其位置，以便与匹配来自多个网络的传入连接请求。单个服务器可通过这种称为 “通配符编址”的技术，为多个网络上的客户端提供服务。要创建一个可监听所有网络的套接字，必须针对 AF_INET 系列绑定 Internet 地址 **INADDR_ANY** ，及针对 AF_INET6 系列绑定 Internet 地址 **in6addr_any** 。即使使用了通配符编址，也可指定 TCP 端口。如果将端口指定为零，则系统将分配一个端口。

一旦 **accept()** 与某个连接请求存在一个集合点，则在对等实体之间建立一个虚电路。 **bind()** 可提供本地端口和本地 Internet 地址， **accept()** 可从请求连接的对等实体收集远程端口和远程 Internet 地址。

选项

系统支持下列套接字选项: **TCP_MAXSEG** 、 **TCP_NODELAY** 、 **TCP_ABORT_THRESHOLD** 、 **TCP_CONN_ABORT_THRESHOLD** 、 **TCP_KEEPCNT** 、 **TCP_KEEPIDLE** 、 **TCP_KEEPINTVL** 、 **TCP_TSOPTENA** 和 **TCP_SACKENA** （在包含文件 **<netinet/tcp.h>** 中定义）。 **TCP_MAXSEG** 选项只能用于 **getsockopt()** ；可以使用 **setsockopt()** 设置，及使用 **getsockopt()** （请参阅 *getsockopt*(2) ）测试 **TCP_NODELAY** 、 **TCP_ABORT_THRESHOLD** 、 **TCP_CONN_ABORT_THRESHOLD** 、 **TCP_KEEPCNT** 、 **TCP_KEEPIDLE** 、 **TCP_KEEPINTVL** 、 **TCP_TSOPTENA** 和 **TCP_SACKENA** 。在 **getsockopt/setsockopt** 调用中，这些选项需要将 *level* 设置为 **IPPROTO_TCP** 。

TCP_MAXSEG （非布尔型选项）允许应用程序接收 TCP SOCK_STREAM 套接字的当前段大小。将在 *optval* 中返回当前段大小。

TCP_NODELAY
（布尔型选项）导致立即发送少量的输出。

TCP_ABORT_THRESHOLD

（非布尔型选项）设置处于 ESTABLISHED 状态的连接的第二个阈值计时器。该选项值是以毫秒为单位的阈值时间。

如果由于计时器已过期，TCP 必须重新传输数据包，那么 TCP 将首先比较它针对两个阈值所等待的总时间，如 RFC 1122, 4.2.3.5 中所述。如果 TCP 等待的时间长于等待第二个阈值 (R2) 所花的时间，那么它会终止连接。该选项的缺省值为 ndd 可调参数 **tcp_ip_abort_cinterval** 的当前值。有关 **tcp_ip_abort_interval** 缺省值的详细信息，请参考 *ndd*(1M) 联机帮助。

TCP_CONN_ABORT_THRESHOLD

（非布尔型选项）在建立连接过程中设置第二个阈值计时器。该选项值是以毫秒为单位的阈值时间。

此选项与 **TCP_ABORT_THRESHOLD** 相似，不同之处在于，此值在建立连接的过程中使用。如果由于计时器已过期，TCP 必须重新传输 SYN 数据包，那么 TCP 将首先比较它针对两个阈值所等待的总时间。如果 TCP 等待的时间长于等待第二个阈值所花的时间，则它会终止连接。该选项的缺省值为 ndd 可调参数 **tcp_ip_abort_cinterval** 的当前值。有关 **tcp_ip_abort_cinterval** 缺省值的详细信息，请参阅 *ndd*(1M) 联机帮助。

TCP_KEEPCNT

（非布尔型选项）当启用 **SO_KEEPALIVE** 选项时，TCP 会探查已经空闲了一段时间的连接。如果远程系统不响应保持连接探查，则在连接被认定断开之前， TCP 会重新传输一定次数的探查。 **TCP_KEEPCNT** 选项可用于影响给定套接字的该值，并指定要发送的保持连接探查的最大数量。该选项采用 **int** 值，值范围是 1 到 32767。

TCP_KEEPIDLE

（非布尔型选项）当启用 **SO_KEEPALIVE** 选项时，TCP 会探查已经空闲了一段时间的连接。该空闲时间段的缺省值为 2 小时。 **TCP_KEEPIDLE** 选项可用于影响给定套接字的该值，并指定保持连接探查间空闲时间的秒数。该选项采用 **int** 值，值范围是 1 到 32767。

TCP_KEEPINIT

（非布尔型选项）如果不能在某段时间内建立 TCP 连接，TCP 将使连接尝试超时。此初始连接建立超时的缺省值为 75 秒。 **TCP_KEEPINIT** 选项可用于影响给定套接字的此初始超时时间段，并且指定在连接尝试超时之前要等待的秒数。对于被动连接，从监听套接字继承 **TCP_KEEPINIT** 选项值。该选项采用 **int** 值，值范围是 1 到 32767。

TCP_KEEPINTVL

（非布尔型选项）当启用 **SO_KEEPALIVE** 选项时，TCP 会探查已经空闲了一段时间的连接。如果远程系统不响应保持连接探查，一段时间后， TCP 会重新传输探查。该重新传输间隔的缺省值为 75 秒。 **TCP_KEEPINTVL** 选项可用于影响给定套接字的该值，并指定在重新传输保持连接探查之前要等待的秒数。该选项采用 **int** 值，值范围是

1 到 32767。

TCP_TSOPTENA

（布尔型选项）当启用该选项时，发件人会在每个数据段中放置一个时间戳。如果已经配置接收方来接受时间戳，则接收方会在 ACK 段中将时间戳发送回去。这就为发送方提供了一种测量往返时间的机制。 TCP 提供布尔型选项 **TCP_TSOPTENA** （来自 **<netinet/tcp.h>** 头文件）来启用或禁用该选项。该选项采用 **int** 值。当启用该选项时，同时启用 **TCP_PAWS** 选项。

TCP_PAWS （布尔型选项）当启用了 PAWS（Protect Against Wrapped Sequence 数）选项时，接收方会拒绝收到的任何旧的重复段。该选项仅用于同步的 TCP 连接。 TCP 提供布尔型选项 **TCP_PAWS** （来自 **<netinet/tcp.h>** 头文件）来启用或禁用该选项。该选项采用 **int** 值。该选项可自动启动 **TCP_TSOPTENA** 选项。

TCP_SACKENA

（布尔型选项）当启用了 Selective Acknowledgment (SACK) 选项时，数据接收方可通知发送方有关已成功到达的段的信息。使用该方法，发送方仅需要重新传输实际已经丢失的那些段。在设置了多个段的情况下，该选项很有用。 TCP 提供布尔型选项 **TCP_SACKENA** （来自 **<netinet/tcp.h>** 头文件）来启用或禁用该选项。该选项采用 **int** 值。

如果设置了 **TCP_NODELAY** ，系统将立即发送少量的输出，而不是在收到确认后，将这些输出集中到单个数据包。如果未设置 **TCP_NODELAY** ，那么当没有未完成的未确认数据时，系统将发送所出现的数据。如果有未完成的未确认数据，那么一旦收到确认，系统就将少量的待发送数据集中到单个数据包。对于某些客户端，例如可发送收不到答复的鼠标事件流的窗口管理器，此封包化可能会导致明显的延迟。使用 **TCP_NODELAY** 选项可避免这种情况。但是请注意，设置 **TCP_NODELAY** 选项可能会导致在网络间发送大量的小型数据包。

缺省情况下，创建套接字后不设置 **TCP_NODELAY** 。

通过 **setsockopt()** 或 **getsockopt()** 调用访问 TCP 选项所使用的选项级别是 TCP 的协议号，可从 **getprotobyname()** （请参阅 *getprotoent*(3N) ）获取此协议号。

如果针对建立的 TCP 连接启用了 **SO_KEEPALIVE** 套接字选项，并且该连接已空闲了两个小时，则 TCP 会将一个数据包发送到远程套接字，以期望远程 TCP 确认连接仍然是活动的。如果远程 TCP 没有及时响应，则 TCP 将根据其正常的重新传输算法继续发送保持连接数据包。如果远程 TCP 在特定的时间限制内没有作出响应，则 TCP 将断开连接。下一个套接字系统调用（例如 **recv()** ）将返回错误，同时将 **errno** 设置为 [ETIMEDOUT] 。有关启用 **SO_KEEPALIVE** 的详细信息，请参阅 *getsockopt*(2) 。

发送和接收缓冲区的缺省大小为 32768 字节（请参阅下面的“警告”）。可以使用 **setsockopt()** 系统调用的 **SO_SNDBUF** 和 **SO_RCVBUF** 选项，或者使用 **t_optmgmt()** 系统调用的 **XTI_SNDBUF** 和 **XTI_RCVBUF** 选项，来更改 TCP 流套接字的发送和接收缓冲区大小。有关详细信息，请参考 *getsockopt*(2) 或 *t_optmgmt*(3) 。

TCP 流套接字的传输缓冲区最大大小为 2147483647 字节。 TCP 流套接字的接收缓冲区最大大小为 1073725440 字节。可以使用 ndd 变量 **tcp_xmit_hiwater_max** 和 **tcp_recv_hiwater_max** 降低这些最大值。

错误

如果套接字操作失败，则可能在 **errno** 中返回下列错误之一。有关错误的详细列表，请参阅特定系统调用的联机帮助页。

[EISCONN] 已连接该套接字。

[ENOBUFS] 没有可用于内部数据结构的缓冲区空间。

[ETIMEDOUT] 由于过度的重新传输，连接已断开。

[ECONNRESET]
对等套接字强制关闭了连接。

[ECONNREFUSED]
远程对等实体主动地拒绝了连接建立（通常是由于没有进程监听端口）。

[EADDRINUSE]
指定的地址已被占用。

[EADDRNOTAVAIL]
此计算机上未提供指定的地址。

警告

将来的发行版或修补软件可能会增加套接字缓冲区的缺省大小，恕不另行通知。因此，如果某个应用程序需要使用 **SO_RCVBUF** 调用 **setsockopt()** ，那么它应在调用 **listen()** 之前执行此调用，或者应首先使用 **SO_RCVBUF** 调用 **getsockopt()** ，并确保所需的新接收缓冲区大小不小于当前的缓冲区大小。这些编程约定符合 TCP 协议限制，可以防止在建立连接后减小 TCP 接收窗口。

作者

TCP 使用的这些套接字接口由加州大学伯克利分校开发。

另请参阅

ndd(1M)、 getsockopt(2)、 recv(2)、 send(2)、 socket(2)、 t_open(3)、 t_optmgmt(3)、 socket(7)、 inet(7F)。

RFC 793 传输控制协议
RFC 1122 Internet 主机要求
RFC 1323 高性能的 TCP 扩展
RFC 1878 IPv4 的可变长度子网表
RFC 2018 TCP 选择确认选项
RFC 2414 增大的 TCP 初始窗口
RFC 2582 TCP 快速恢复算法的 NewReno 修改 (NewReno Modifications to TCP's Fast Recovery Algorithm)

名称

tels、telm - 分别为 STREAMS Telnet 从属（伪终端）驱动程序和 STREAMS Telnet 主驱动程序（仅供 telnetd 使用）

概要

#include <sys/termios.h>
#include <sys/strtio.h>

int open("/dev/pts/t*N*", O_RDWR);

说明

Telnet 伪终端由一对紧密耦合的字符设备组成，称为主设备和次设备。主次设备驱动程序一起在服务器端提供一个 Telnet 连接，其中，主设备驱动程序提供 **telnetd** 连接，次设备驱动程序则对 Telnet 应用程序进程提供终端设备专用文件访问权限，如下所述：

具有 **ptem** （STREAMS pty 模拟模块）和 **ldterm** （STREAMS 线路规范模块）的从属驱动程序 **tels** 被推到顶部（为简便起见未显示出来），提供了一个如 *termio*(7) 中所述的终端接口。但是，提供如 *termio*(7) 中所述终端接口的设备后具有硬件设备；相反，次设备则通过 Telnet 伪终端的主端用 **telnetd** 对其进行处理。

文件系统中不包括各主设备的任何节点。相反，主驱动程序设置为 STREAMS *clone*(7) 驱动程序，并将其主设备号设置为驱动程序副本的主编号，次设备号设置为 **telm** 驱动程序的主编号。telnetd 通过 **/dev/telnetm** 将 *open*(2) 系统调用用作设备文件参数来打开主驱动程序。克隆打开找到主设备可用的下一个次设备号。仅当主设备及其相应的次设备尚未打开时，才可使用该主设备。

要使用 STREAMS Telnet 子系统，必须安装主驱动程序 **/dev/telnetm** 的节点和 *N* 个 Telnet 次设备。

次设备的数量由一个名为 **nstrtel** 的内核可调参数设置。可使用 SAM 进行修改；其最小缺省值为 60。 **nstrtel** 的值为可以打开的 telnet 会话数的上限。

允许打开多个 Telnet 次设备。

主从驱动程序将所有 STREAMS 消息传递给其相邻的驱动程序。当 Telnet 客户端关闭连接时，将发送 **M_HANGUP** 消息到对应的次设备，该消息将使该次设备无法使用。当尝试对次设备文件进行 *write*(2) 系统调用时，从属端进程将获取错误编号 **ENXIO** ，但仍能读取从属流中的其余数据。最后，当读取完所有数据后，*read*(2) 系统调用将返回 0，表示不能再使用次设备。

作者

tels() 和 **telm()** 由 HP 开发。

文件

/dev/telnetm	STREAMS Telnet 主克隆设备
/dev/pts/tN	STREAMS Telnet 次设备，其中 *N* 为次设备的次设备号，并且 0< *N* < **nstrtel** 。

另请参阅

insf(1M)、 open(2)、 ioctl(2)、 streamio(7)、 ldterm(7)、 telnetd(1M)、 ptem(7)。

名称

termio、termios - 通用终端接口

说明

无论所涉及的硬件是什么，所有 HP-UX 异步通信端口均使用相同的通用接口。例如 **rlogin** （请参阅 *rlogin*(1) ）的网络连接使用伪终端接口（请参阅 *pty*(7) ）。

此讨论集中于此接口的常用功能。

打开终端文件

打开终端文件时，通常会导致进程一直等待，直到建立连接。实际上，用户的程序很少打开这些文件；而由专用程序（例如， **getty** ，请参阅 *getty*(1M)) ）打开并成为用户的标准输入文件、标准输出文件和标准错误文件。

如果同时清除 **O_NDELAY** 和 **O_NONBLOCK** 标记（请参阅 *open*(2) ），则在完成请求的调制解调器类型连接（请参阅 *modem*(7) ）之前 *open* 将一直阻塞。如果设置了 **O_NDELAY** 或 **O_NONBLOCK** 标记，则 *open* 成功并立即返回，而不等待请求的调制解调器连接完成。 **CLOCAL** 标记（请参阅“控制模式”一节）还可以影响 *open*(2) 。

进程组

终端可以拥有与其关联的前台进程组。此前台进程组在处理信号生成输入字符方面起着特殊的作用。

通过在单个进程组中放置相关进程并将此进程组与终端关联，命令解释程序进程可以将终端分配给不同的 *jobs*（进程组）。如果满足权限要求，则终端的前台进程组可以由进程设置或检查（请参阅 *tcsetpgrp*(3C) 或 *tcgetpgrp*(3C) ）。终端接口通过仅允许不属于前台进程组的进程访问终端的方式来辅助此分配。

如果进程组的每个成员的父级本身是此进程组成员或者它不是此组的会话（请参阅 会话）的成员，则认为此进程组是孤立的。

会话

创建会话的进程（请参阅 *setsid*(2) 或 *setpgrp*(2) ）将成为会话发起者。每个进程组仅属于一个会话。将进程视为其进程组为其成员的会话的成员。新创建的进程加入其父级的会话。进程可以更改其会话成员关系（请参阅 *setpgid*(2) 或 *setpgrp*(2) ）。通常，会话包含作为单个登录的结果而创建的所有进程（包括子进程）。

控制终端

终端可以作为其控制终端属于一个进程。拥有控制终端的每个会话进程都拥有相同的控制终端。终端最多只能是一个会话的控制终端。会话的控制终端由会话发起者分配。如果会话发起者没有控制终端，且在未使用 **O_NOCTTY** 选项（请参阅 *open*(2) ）的情况下打开了尚未与会话关联的终端设备文件，则此终端将成为会话的控制终端，并且将此控制终端的前台进程组设置为会话发起者的进程组。当控制终端与会话关联时，认为会话发起者是控制终端的控制进程。

控制终端在 **fork()** （请参阅 *fork*(2) ）期间由子进程继承。如果进程通过 **setsid()** 或 **setpgrp()** （请参阅 *setsid*(2) 和 *setpgrp*(2) ）创建新会话，或者已经关闭与控制终端关联的所有文件描述符时，此进程将放弃其控制终端。

控制进程终止时，将取消控制终端与当前会话的关联，以允许新的会话发起者获取。将 **SIGHUP** 信号发送到控制终端的前台进程组中的所有进程。可能拒绝早期会话中的其他进程对终端的后续访问（请参阅“终端访问控制”一节），处理对终端的访问尝试时认为已检测到调制解调器断开连接。

终端访问控制

允许从其控制终端的前台进程组的进程执行读取操作（请参阅“输入处理和读取数据”一节）。如果进程不在其控制终端的前台进程组中，则认为此进程及其进程组的所有成员在此控制终端的后台进程组中。将拒绝后台进程组中的进程从其控制终端进行的所有读取尝试。如果拒绝且进行读取的进程忽略或阻塞 **SIGTTIN** 信号，或者进程（在单独实现 *vfork* 和 *fork* 的系统上）对 *vfork*(2) 发出了调用但尚未对 *exec*(2) 发出调用，或者进行读取的进程的进程组是孤立的，则 **read()** 返回 −1，并将 **errno** 设置为 **EIO** ，且不发送信号。在拒绝读取的所有其他情况下，会将 **SIGTTIN** 信号发送到进行读取的进程的进程组。 **SIGTTIN** 信号的缺省操作是停止将其发送到的进程。

如果进程在其控制终端的前台进程组中，则允许写入操作（请参阅“写入数据和输出处理”一节）。如果设置了 **TOSTOP** （请参阅“本地模式’一节），进程没有忽略也没有阻塞 **SIGTTOU** 信号，且进程（在单独实现 *vfork* 和 *fork* 的系统上）没有对 *vfork*(2) 发出调用，也没有对 *exec*(2) 进程发出后续调用，则将拒绝后台进程组中的进程写入其控制终端的尝试。如果拒绝写入且后台进程组是孤立的，则 **write()** 返回 −1， **errno** 设置为 **EIO** 。如果拒绝写入，但后台进程组不是孤立的，则将 **SIGTTOU** 信号发送到进行写入的进程的进程组。 **SIGTTOU** 信号的缺省操作是停止将其发送到的进程。

设置终端参数的某些调用的处理方式与写入相似，不同之处在于忽略 **TOSTOP** ；也就是说，其效果与设置 **TOSTOP** 时终端写入的效果相同。

输入处理和读取数据

与终端设备文件关联的终端设备可以在全双工模式下操作，以便即使在进行数据输出时，数据也可以到达。每个终端设备文件都拥有与其关联的 *input queue* ，系统将在进程读取传入数据前将数据存储在其中。系统要求可以在输入队列中存储的字符数不得大于 **MAX_INPUT** 。此限制取决于特定的实现，但至少为 256。达到输入限制时，将忽略所有已保存的字符，且不进行通知。

所有输入要么在标准模式下处理，要么在非标准模式下处理（请参阅 “标准模式输入处理”和“非标准模式输入处理”）。此外，根据 **c_iflag** （请参阅“输入模式”一节）和 **c_lflag** （请参阅“本地模式”一节）字段处理输入字符。例如，此类处理可以包括 *echoing* ，这通常表示从终端接收到输入字符后，立即将其传输回终端。这对于在全双工模式下操作的终端很有用。

为从终端设备文件读取的进程提供数据的方式取决于终端设备文件处于标准模式还是处于非标准模式。

此外，还取决于 **O_NONBLOCK** 或 **O_NDELAY** 标记是由 *open*(2) 还是 *fcntl*(2) 设置的。如果同时清除 **O_NONBLOCK** 和 **O_NDELAY** 标记，则在数据可用或接收到信号之前，将阻塞读取请求。如果设置了 **O_NONBLOCK** 或 **O_NDELAY** 标记，则读取请求在不阻塞的情况下，以下列三种方式之一完成：

- 如果存在的可用数据足以满足整个请求，则 **read()** 成功完成，读取请求的所有数据，并返回读取的字符数。
- 如果存在的可用数据不足以满足整个请求，则 **read()** 成功完成，读取尽可能多的数据，并返回读取的字符数。
- 如果没有可用数据，则 **read()** 返回 −1，并在设置了 **O_NONBLOCK** 标记时将 **errno** 设置为 **EAGAIN** 。否则，（清除了标记 **O_NONBLOCK** 并设置了 **O_NDELAY** ） **read()** 成功完成，不读取任何数据，并返回计

数 0。

数据的可用性取决于输入处理模式是规范的还是非规范的。以下两节“标准模式输入处理”和“非标准模式输入处理”说明其中每种输入处理模式。

标准模式输入处理（清除和终止处理）

在标准模式输入处理中，以行为单位处理终端输入，其中行由换行符 (NL)、文件结束标记字符 (EOF) 或行结束标记字符 (EOL) 或 (EOL2) 分隔。有关 **NL** 、 **EOF** 、 **EOL** 和 **EOL2** 的详细信息，请参阅 “特殊字符”一节。这表示在键入整个行或接收到信号之前，读取请求不返回。此外，不管在读取调用中请求了多少字符，最多返回一行。但是，不必一次读取一整行；在读取中可以请求任何数目的字符（甚至是一个字符），而不会丢失信息。

MAX_CANON 是对行中字符数的限制。对于每个特定的实现，此限制是不同的，但至少为 256。

达到 **MAX_CANON** 限制时，将忽略当前未分隔行中的所有字符，且不进行通知。

当接收到任一下列三个特殊字符时，将进行清除和终止处理： ERASE、 WERASE 或 KILL 字符（请参阅“特殊字符”）。此处理影响输入队列中尚未由 NL、 EOF、 EOL 或 EOL2 字符分隔的数据。此未分隔的数据构成当前行。 ERASE 字符删除当前行中的最后一个字符（如果存在一个的话）。 WERASE 字符删除当前行中的最后一个字（如果存在一个的话）。字定义为一系列非空字符（制表符相当于空格）。 KILL 字符删除当前行中的所有数据（如果有的话），以及（可选）输出换行符 (NL)。这些字符在键击的基础上进行操作，与可能在其之前的任何退格或制表位无关。如果当前行为空，则 ERASE、 WERASE 和 KILL 字符不起作用。 ERASE、 WERASE 和 KILL 字符不放置在输入队列中。

非标准模式输入处理（**MIN/TIME** 交互）

在非标准模式输入处理中，输入字符不组合为行，且不进行清除和终止处理。 **c_cc** 数组的 **MIN** 和 **TIME** 成员的值（请参阅“termios 结构”一节）用于确定如何处理收到的字符。 **MIN** 表示在 **read()** 成功返回之前应该接收到的最小字符数。 TIME 是精度为 0.10 秒的计时器，用于使突发和短期数据传输超时。下面说明了 MIN 和 TIME 的四种可能情况及其交互。

情况 A： MIN > 0， TIME > 0

在这种情况下， TIME 充当字符间计时器，且在接收到第一个字符后激活。由于它是字符间计时器，因此在接收到每个字符后将重置。 MIN 和 TIME 之间的交互如下所述：

- 接收到一个字符后，将启动字符间计时器。
- 如果在字符间计时器到期之前接收到 MIN 个字符（请记住，在收到每个字符后将重置计时器），则将满足读取。如果在接收到 MIN 个字符之前计时器到期，则将直到此时接收到的字符返回给用户。
- 请注意，如果 TIME 到期，则将返回至少一个字符，因为除非接收到字符，否则不会启用计时器。在这种情况下（ MIN > 0， TIME > 0），在因接收到第一个字符或收到信号而激活 MIN 和 TIME 机制之前，读取一直阻塞。

情况 B： MIN > 0， TIME = 0

在这种情况下，由于 TIME 的值为零，因此计时器不起作用，只有 MIN 是有效的。不会满足挂起的读取，直到完成以前的任何读取后接收到 MIN 个字符（挂起的读取将阻塞，直到接收到 MIN 个字符）或信号。使用此情况

处理基于记录的终端 I/O 的程序可以在读取操作中无限期地阻塞。

情况 C： MIN = 0， TIME > 0

在这种情况下，由于 MIN 的值为零，因此 TIME 不再表示字符间计时器。现在它充当读取计时器，处理 **read()** 函数后，就会激活。接收到单个字符后 *or* 读取计时器到期，就会满足读取。如果计时器到期，则不返回字符。如果计时器未到期，则可以满足读取的唯一方法是接收到字符。读取不能无限期地阻塞等待字符，因为如果在启动读取后 TIME × 0.10 秒内未接收到字符，则 **read()** 将返回零值，且不读取任何数据。

情况 D： MIN = 0， TIME = 0

返回请求的字符数或当前可用的字符数（取其中的较小值），且不等待输入更多字符。如果没有可用的字符，则 **read()** 返回零值，且不读取任何数据。

有关 MIN 和 TIME 的注意事项如下：

1. 在上面的说明中， MIN 和 TIME 的交互不是对称的。例如，当 MIN > 0 且 TIME = 0 时， TIME 不起作用。但是，在相反的情况下（ MIN = 0 且 TIME > 0）， MIN 和 TIME 都起作用，因为接收到单个字符时会满足 MIN。
2. 另请注意，在情况 A （ MIN > 0， TIME > 0）中， TIME 表示字符间计时器，而在情况 C （ MIN = 0， TIME > 0）中， TIME 表示读取计时器。

这两点着重说明了 MIN/TIME 功能的双重用途。情况 A 和 B（其中 MIN > 0）的存在是为了处理突发模式活动（如文件传输程序），其中程序希望一次至少处理 MIN 个字符。在情况 A 中，字符间计时器由用户作为一项安全措施激活，而在情况 B 中则将其关闭。

情况 C 和 D 的存在是为了处理单字符的计时传输。可以轻松地使这些情况适应基于屏幕的应用程序（这些应用程序需要了解在刷新屏幕之前字符是否存在于输入队列中）。在情况 C 中，对读取计时，而在情况 D 中，不对读取计时。

另一重要说明是， MIN 始终为最小值。它不表示记录长度。例如，如果程序启动 20 个字符的读取，而 MIN 为 10 且存在 25 个字符，则会将 20 个字符返回给用户。如果程序请求了所有字符，则将 25 个字符都返回给用户。

此外，如果 TIME 大于零且 MIN 大于 **MAX_INPUT** ，则读取决不会因接收到 MIN 个字符而终止，因为在超过 **MAX_INPUT** 时，将忽略已保存的所有字符，且不进行通知。如果 TIME 为零且 MIN 大于 **MAX_INPUT** ，则除非接收到信号，否则读取决不会终止。

特殊字符

某些字符在输入、输出或这两者中拥有特殊的功能。除非专门拒绝，否则可以更改或禁用每个特殊字符。要禁用字符，请将其值设置为 **_POSIX_VDISABLE** （请参阅 *unistd*(5) ）。这些特殊功能及其缺省字符值如下：

INTR （Rubout 或 ASCII DEL）输入中的特殊字符，如果启用 **ISIG** （请参阅“本地模式”一节），则会识别它。生成 **SIGINT** 信号，此信号发送到前台进程组中终端为其控制终端的所有进程。通常，会强制终止每个这样的进程，但可以进行如下安排：忽略或保留信号，或者在一致同意的位置接收陷阱；请参阅 *signal*(2) 和 *signal*(5) 。如果设置了 **ISIG** ，则在处理时

将忽略 INTR 字符。如果清除了 **ISIG** ，则 INTR 字符将作为普通数据字符处理，且不发送信号。

QUIT （Control-\ 或 ASCII FS）输入中的特殊字符。如果设置了 **ISIG** （请参阅“本地模式”一节），则可识别它。此字符的处理方式与 INTR 字符相似，不同之处在于会生成 **SIGQUIT** 信号，不仅会终止接收此信号的进程，还会在当前工作目录中创建一个名为 **core** 的核心图像文件（如果实现支持核心文件的话）。

SWTCH (ASCII NUL) 输入中的特殊字符，只能由 Shell 层工具 *shl*(1) 使用。Shell 层工具不是通用终端接口的一部分。当遇到 SWTCH 字符时，通用终端接口不执行任何特殊功能。

ERASE (**#**) 输入中的特殊字符，如果启用 **ICANON** （请参阅“本地模式”一节），则会识别它。清除前面的字符。不清除行开头前面的字符，行由 NL、 EOF、 EOL 或 EOL2 字符分隔。如果启用 **ICANON** ，则在处理时忽略 ERASE 字符。如果未启用 **ICANON** ，则将 ERASE 字符视为普通数据字符。

WERASE （禁用）输入中的特殊字符，如果启用 **ICANON** （请参阅“本地模式”一节），则会识别它。清除前面的字。不清除行开头前面的字符，行由 NL、 EOF、 EOL 或 EOL2 字符分隔。如果启用 **ICANON** ，则在处理时忽略 WERASE 字符。如果未启用 **ICANON** ，则将 WERASE 字符视为普通数据字符。

KILL (**@**) 输入中的特殊字符，如果启用 **ICANON** ，则会识别它。 KILL 删除整个行，行由 NL、 EOF、 EOL 或 EOL2 字符分隔。如果启用 **ICANON** ，则在处理时忽略 KILL 字符。如果未启用 **ICANON** ，则将 KILL 字符视为普通数据字符。

EOF （Control-D 或 ASCII EOT）输入中的特殊字符，如果启用 **ICANON** ，则可识别它。 EOF 可以用于从终端生成文件结束标记。等待读取的所有字符在接收到后将立即传递到程序，而不等待换行符，并忽略 EOF。因此，如果没有等待的字符（即 EOF 出现在行开头），则从 **read()** 返回计数为零的字符，表示文件结束。如果启用 **ICANON** ，则在处理时忽略 EOF 字符。如果未启用 **ICANON** ，则将 EOF 字符视为普通数据字符。

NL (ASCII LF) 输入中的特殊字符，如果启用 **ICANON** 标记，则会识别它。它是行分隔符 (**\n**)。如果未启用 **ICANON** ，则将 NL 字符视为普通数据字符。

EOL (ASCII NUL) 输入中的特殊字符，如果启用 **ICANON** ，则会识别它。 EOL 是附加的行分隔符，类似于 NL。通常不使用它。如果未启用 **ICANON** ，则将 EOL 字符视为普通数据字符。

EOL2 （禁用）输入中的特殊字符，如果启用 **ICANON** ，则会识别它。 EOL2 是附加的行分隔符，类似于 EOL。通常不使用它。如果未启用 **ICANON** ，则将 EOL2 字符视为普通数据字符。

SUSP （禁用）输入时识别的特殊字符。如果启用 **ISIG** ，则接收到 SUSP 字符会导致将 **SIGTSTP** 信号发送到前台进程组中终端为其控制终端的所有进程，且在处理时忽略 SUSP 字符。如果未启用 **ISIG** ，则将 SUSP 字符视为普通数据字符。命令解释程序进程通常将 SUSP 设置为

Control-Z。

DSUSP （禁用）输入时识别的特殊字符。如果启用 **ISIG** ，而且前台进程组中的一个进程尝试读取 DSUSP 字符，则将 **SIGTSTP** 信号发送到前台进程组中终端为其控制终端的所有进程，然后忽略 DSUSP 字符。如果未启用 **ISIG** ，则将 DSUSP 字符视为普通数据字符。请注意 DSUSP 类似于 SUSP，不同之处在于，前台进程组中的进程尝试在读取 DSUSP 字符时发送信号，而不是在键入它时发送信号。

STOP （Control-S 或 ASCII DC3）输入和输出中的特殊字符。如果启用 **IXON** （输出控制），则 STOP 字符的处理会暂时挂起终端设备的输出。对于 CRT 终端防止输出在可以读取前消失，这是很有用的。在挂起输出且启用 **IXON** 时，忽略 STOP 字符，且不读取它。如果启用 **IXON** ，则在处理时忽略 STOP 字符。如果未启用 **IXON** ，则将 STOP 字符视为普通数据字符。如果启用 **IXOFF** （输入控制），则输入队列中的未读取字符数接近系统指定的限制时，系统会将 STOP 字符发送到终端设备。这是通过指示终端设备停止发送数据来防止此缓冲区溢出的尝试。

START （Control-Q 或 ASCII DC1）输入和输出中的特殊字符。如果启用 **IXON** （输出控制），则 START 字符的处理会恢复已挂起的输出。在未挂起输出且启用 **IXON** 时，忽略 START 字符，且不读取它。如果启用 **IXON** ，则在处理时忽略 START 字符。如果未启用 **IXON** ，则将 START 字符视为普通数据字符。如果启用 IXOFF （输入控制），则在输入队列下降到某个系统定义级别时，系统将 START 字符发送到终端设备。当输入队列不再有可能溢出的危险时，会出现这种情况。

CR (ASCII CR) 输入中的特殊字符，如果启用 **ICANON** ，则会识别它。当启用 **ICANON** 和 **ICRNL** 但不启用 **IGNCR** 时，此字符转换为 NL ，且拥有与 NL 字符相同的效果。如果启用 **ICANON** 和 **IGNCR** ，则忽略 CR 字符。如果启用 **ICANON** 但不启用 **ICRNL** 和 **IGNCR**，则将 CR 字符视为普通数据字符。

LNEXT （禁用）输入时识别的特殊字符。导致忽略下一个字符的特殊含义。这适用于上面指定的所有特殊字符。它允许输入这样的字符：在其他情况下将由系统解释为拥有特殊的功能。

在打开终端端口时，将为特殊字符分配其缺省字符值。所使用的缺省值是由 System V Interface Definition 第三版 (SVID3) 指定的，但 WERASE (Control-W) 和 LNEXT (Ctrl-V) 字符除外，并将其设置为 **_POSIX_VDISABLE** 以维持与 HP-UX 以前版本的二进制兼容性。通过使用 **stty** 命令（请参阅 *stty*(1) ），可以为基于系统级的所有端口更改在打开端口时分配的缺省字符值。使用 **stty** 命令打开特定端口后，也可以更改其字符值。不能更改或禁用 NL 和 CR 字符。可以更改或禁用剩余特殊字符的字符值，以适合个人喜好。

如果设置了 **ICANON** （请参阅“本地模式”一节），则可以通过前置的 \ 字符对 ERASE、 KILL 和 EOF 字符进行转义，在这种情况下不执行特殊功能。这些字符和剩余的特殊字符也可以通过在其前面添加 LNEXT 字符（请参阅上面的 LNEXT ）来进行转义。

如果两个或多个特殊字符拥有相同的值，则处理字符时执行的函数是未定义的。

调制解调器断开连接

如果控制终端的终端接口检测到调制解调器断开了连接，并且在终端的 **c_cflag** 字段中清除了 **CLOCAL** （请参阅“控制模式”一节），则会将 **SIGHUP** 信号发送到控制终端的控制进程。除非进行了其他安排，否则这会导致控制进程终止。从终端设备的任何后续读取都将返回文件结束标记指示，直到关闭此设备。因此，读取终端文件并测试文件结束标记的进程可以在断开连接后适当地终止。对终端设备的任何后续 **write()** 都将返回 −1 且将 **errno** 设置为 **EIO** ，直到关闭此设备。

关闭终端设备文件

关闭终端设备文件的最后一个进程会导致尚未发送到设备的任何输出发送到设备，即使输出被挂起也是如此。此最后关闭始终阻塞（即使已指定非阻塞 I/O ），直到所有输出都发送到终端设备。忽略已接收到但未读取的任何输出。

写入数据和输出处理

写入字符时，会将字符放置在输出队列中。发送以前写入的字符后，立即将输出队列上的字符传输到终端。根据 **c_oflag** 字段处理这些字符（请参阅“输出模式”一节）。通过在输入字符到达时将其放置在输出队列中的方式，对其进行回显。如果进程为输出生成字符的速度比发送字符的速度快，则在进程的输出队列超过某个限制时，将挂起此进程。当队列下降到某个阈值时，将恢复进程。

termios 结构

通过使用头文件 **<termios.h>** 中定义的 **termios** 结构，需要控制某些终端 I/O 特性的例行程序可以如此操作。结构定义如下：

```
#define NCCS    16
struct  termios  {
    tcflag_t  c_iflag;          /* input modes */
    tcflag_t  c_oflag;          /* output modes */
    tcflag_t  c_cflag;          /* control modes */
    tcflag_t  c_lflag;          /* local modes */
    tcflag_t  c_reserved;       /* reserved for future use */
    cc_t      c_cc[NCCS];       /* control chars */
};
```

特殊字符由数组 **c_cc** 定义。每个特殊字符函数的相对位置和缺省值如下：

INTR	VINTR	DEL
QUIT	VQUIT	Control-|
ERASE	VERASE	#
KILL	VKILL	@
EOF	VEOF	Control-D
EOL	VEOL	NUL
EOL2	VEOL2	禁用

MIN	VMIN	NUL
TIME	VTIME	Control-D
SUSP	VSUSP	禁用
START	VSTART	Control-Q
STOP	VSTOP	Control-S
WERASE	VWERASE	禁用
LNEXT	VLNEXT	禁用
DSUSP	VDSUSP	禁用

termio 结构

termio 结构已由 **termios** 结构替代，提供此结构是为了与以前的应用程序向后兼容（请参阅“termio 警告”一节）。结构是在头文件 **<termio.h>** 中定义的，其定义如下：

```
#define NCC    8
struct  termio  {
    unsigned short  c_iflag;        /* input modes */
    unsigned short  c_oflag;        /* output modes */
    unsigned short  c_cflag;        /* control modes */
    unsigned short  c_lflag;        /* local modes */
    char            c_line;         /* line discipline */
    unsigned char   c_cc[NCC];      /* control chars */
};
```

模式

接下来的四节将说明使用 **termios** 和 **termio** 结构（请参阅“termio 警告”一节）可以设置的特定终端特性。忽略没有在下面显式定义的模式字段中的任何位。但是，应该始终清除它们以防止将来出现兼容性问题。

输入模式

c_iflag 字段说明基本终端输入控制：

IGNBRK	忽略中断条件。
BRKINT	在中断时发出中断信号。
IGNPAR	忽略具有奇偶校验错误的字符。
PARMRK	标记奇偶校验错误。
INPCK	启用输入奇偶校验检查。
ISTRIP	去除字符。
INLCR	将 NL 映射为输入中的 CR。
IGNCR	忽略 CR。
ICRNL	将 CR 映射为输入中的 NL。
IUCLC	将大写字母映射为输入中的小写字母。
IXON	启用开始/停止输出控制。

IXANY	启用任何字符以重新开始输出。
IXOFF	启用开始/停止输入控制。
IMAXBEL	在过长的输入行上启用 BEL。

中断条件定义为继续时间超过发送一个字符的时间的零值位的序列。例如，数据全为零的字符分帧或奇偶校验错误解释为单个中断条件。

如果设置了 **IGNBRK** ，则忽略中断条件。因此任何进程都无法读取中断条件。如果清除了 **IGNBRK** 并设置 **BRKINT** ，则中断条件同时刷新输入队列和输出队列，如果终端是前台进程组的控制终端，则中断条件将为此前台进程组生成单个 **SIGINT** 信号。如果既未设置 **IGNBRK** 也未设置 **BRKINT** ，则将中断条件作为单个 **\0** 字符读取；如果设置了 **PARMRK** ，则将中断条件作为三字符序列 **\377** 、 **\0** 、 **\0** 读取。

如果设置了 **IGNPAR** ，则忽略具有其他分帧和奇偶校验错误 (中断除外) 的字符。

如果设置了 **PARMRK** ，并清除了 **IGNPAR** ，则将具有分帧或奇偶校验错误（中断除外）的字符作为三字符序列读取： **\377** 、 **\0** 、 *X* ，其中 *X* 是出错时接收到的字符的数据。为避免此情况中出现歧义，如果清除了 **ISTRIP**，则将 **\377** 的有效字符读取为 **\377** 、 **\377** 。如果同时清除了 **PARMRK** 和 **IGNPAR** ，则将分帧或奇偶校验错误（中断除外）读取为字符 **\0** 。

如果设置了 **INPCK** ，则启用输入奇偶校验检查。如果清除了 **INPCK** ，则禁用输入奇偶校验检查。启用还是禁用输入奇偶校验检查与启用还是禁用奇偶校验检测无关（请参阅“控制模式”一节）。如果设置了 **PARENB**（请参阅“控制模式”）并清除了 **INPCK** ，则启用奇偶校验生成，但禁用输入奇偶校验检查；终端所连接的硬件将识别奇偶校验位，但终端专用文件将不检查是否正确设置了此位。

下表显示标记 **IGNBRK** 、 **BRKINT** 、 **IGNPAR** 和 **PARMRK** 之间的相互关系。标有 输入的列给出接收的输入字符的各种类型，如下所示：

0	NUL 字符 (**\0**)
C	除 NUL 之外的字符
P	检测到奇偶校验错误
F	检测到分帧错误

括在方括号中的项指示一个或多个条件为真。

如果清除了 **INPCK** 标记，则不按照此表处理接收到的具有奇偶校验错误的字符，而是相反，好像没有出现奇偶校验错误。在标记列下， **Set** 表示已设置标记， **Clear** 表示未设置标记， **X** 表示可能已设置或清除标记。标有 **Read** 的列显示将传递到应用程序代码的结果。 — 表示没有字符或条件传递到应用程序代码。值 **SIGINT** 表示未返回字符，但已将 **SIGINT** 信号发送到控制终端的前台进程组。

输入	1IGNBRK	BRKINT	IGNPAR	PARMRK	读取
0[PF]	设置	X	X	X	—
0[PF]	清除	设置	X	X	**SIGINT**
0[PF]	清除	清除	X	设置	’\377’,’\0’,’\0’
0[PF]	清除	清除	X	清除	’\0’

C[PF]	X	X	设置	X	—
C[PF]	X	X	清除	设置	'\377','\0',C
C[PF]	X	X	清除	清除	'\0'
'\377'	X	X	X	设置	'\377','\377'

如果设置了 **ISTRIP** ，则有效的输入字符首先去除到 7 位，否则将处理全部 8 位。

如果设置了 **INLCR** ，则将接收到的 NL 字符转换为 CR 字符。如果设置了 **IGNCR** ，则忽略（不读取）接收到的 CR 字符。如果清除了 **IGNCR** 并设置了 **ICRNL** ，则将接收到的 CR 字符转换为 NL 字符。

如果设置了 **IUCLC** ，则将接收到的大写字母字符转换为相应的小写字符。

如果设置了 **IXON** ，则启用开始/停止输出控制。接收到的 STOP 字符会导致挂起输出，而收到的 START 字符会导致重新开始输出。如果设置了 **IXANY** 和 **IXON** ，则没有分帧或奇偶校验错误的任何输入字符将导致重新开始已挂起的输出。如果设置了这三个标记，输出已挂起，并接收到具有分帧或奇偶校验错误的输入字符，则输出将重新开始（如果处理导致读取数据）。如果设置了 **IXON** ，则不读取 START 和 STOP 字符，而是仅仅执行流控制功能。如果清除了 **IXON** ，则读取 START 和 STOP 字符。

如果设置了 **IXOFF** ，则启用开始或停止输入控制。当输入队列中的字符数超过了系统定义值（高水印）时，系统将传输 STOP 字符。这旨在导致终端设备停止传输数据，以防止输入队列中的字符数超过 **MAX_INPUT** 。当从输入队列读取了足够的字符（剩余字符数小于另一系统定义值（低水印））时，系统将传输 START 字符，此字符旨在导致终端设备恢复传输数据（没有使输入队列溢出的风险）。为了避免可能出现死锁，只要输入缓冲区中没有行分隔符，就在标准模式下忽略 **IXOFF** 。在这种情况下，在高水印不发送 STOP 字符，但如果接收到分隔符，则将稍后传输。如果从输入队列中读取了所有完整行，仅剩下没有行分隔符的部分行，则发送 START 字符，即使字符数仍大于低水印。在设置了 **ICANON** ，并且输入流在行分隔符之间包含的字符数比高水印允许的大时，不保证 **IXOFF** 可以防止缓冲区溢出和数据丢失，因为可能根本没有及时发送 STOP 字符。

如果设置了 **IMAXBEL** ，则在输入队列溢出时，回显 ASCII BEL 字符。不存储进一步的输入，但是不忽略存在于输入队列中的任何输入。如果清除了 **IMAXBEL** ，则不回显 ASCII BEL 字符，并且在输入队列溢出时，忽略已存在于输入队列中的输入。

清除初始输入控制值的所有位。

输出模式

c_oflag 字段指定系统处理输出的方式：

OPOST	后处理输出。
OLCUC	将小写字母映射为输出中的大写字母。
ONLCR	将 NL 映射为输出中的 CR-NL。
OCRNL	将 CR 映射为输出中的 NL。
ONOCR	在第 0 列上没有 CR 输出。
ONLRET	NL 执行 CR 功能。
OFILL	将填充字符用于延迟。

OFDEL	填充为 DEL，否则为 NUL。
NLDLY	选择换行符延迟：
NL0	无延迟
NL1	延迟类型 1
CRDLY	选择回车符延迟：
CR0	无延迟
CR1	延迟类型 1
CR2	延迟类型 2
CR3	延迟类型 3
TABDLY	选择水平制表符延迟：
TAB0	无延迟
TAB1	延迟类型 1
TAB2	延迟类型 2
TAB3	将制表符扩展为空格。
XTABS	将制表符扩展为空格。
BSDLY	选择退格符延迟：
BS0	无延迟
BS1	延迟类型 1
VTDLY	选择垂直制表符延迟：
VT0	无延迟
VT1	延迟类型 1
FFDLY	选择换页符延迟：
FF0	无延迟
FF1	延迟类型 1

如果设置了 **OPOST** ，则如剩余标记所示后处理输出字符；否则将传输字符，而不进行更改。

如果设置了 **OLCUC** ，则将小写字母字符作为对应的大写字符传输。此功能通常与 **IUCLC** 联合使用。

如果设置了 **ONLCR** ，则将 NL 字符作为 CR-NL 字符对传输。如果设置了 **OCRNL** ，则将 CR 字符作为 NL 字符传输。如果设置了 **ONOCR** ，则不传输第 0 列（第一个位置）上的 CR 字符。如果设置了 **ONLRET** ，则假定 NL 字符执行回车功能；列指针将设置为 0，并且将使用为 CR 指定的延迟。如果清除了 **ONLRET** ，则假定 NL 字符仅执行换行功能；使用为 NL 指定的延迟，且列指针保持不变。对于上述所有情况，如果实际传输了 CR 字符，则列指针始终设置为 0。

延迟位指定在将某些字符发送到终端时传输停止多长时间以允许机械运动或其他运动。 **NL0** 、 **CR0** 、 **TAB0** 、 **BS0** 、 **VT0** 和 FF0 的值指示无延迟。如果设置了 **OFILL** ，则为延迟而不是计时延迟传输填充字符。对于仅需要很短延迟的高波特率终端，这是很有用的。如果设置了 **OFDEL** ，则填充字符为 DEL ；否则为 NUL。

如果指定了换页符或垂直制表符延迟，则持续大约 2 秒。

换行符延迟持续大约 0.10 秒。如果设置了 **ONLRET** ，则使用回车符延迟而不是换行符延迟。如果设置了 **OFILL** ，则传输两个填充字符。

回车符延迟类型 1 取决于当前的列位置；类型 2 约为 0.10 秒；类型 3 约为 0.15 秒。如果设置了 **OFILL** ，则延迟类型 1 传输两个填充字符；类型 2 传输四个填充字符。

水平制表符延迟类型 1 取决于当前的列位置。类型 2 约为 0.10 秒；类型 3 指定将制表符扩展为空格。如果设置了 **OFILL** ，则为任何延迟传输两个填充字符。

退格符延迟持续大约 0.05 秒。如果设置了 **OFILL** ，则传输一个填充字符。

实际延迟取决于线路速度和系统负载。

清除初始输出控制值的所有位。

控制模式

c_cflag 字段说明终端的硬件控制：

CBAUD	波特率：	**CSIZE**	字符大小：
B0	挂起	CS5	5 位
B50	50 波特	CS6	6 位
B75	75 波特	CS7	7 位
B110	110 波特	CS8	8 位
B134	134.5 波特		
B150	150 波特	CSTOPB	发送两个停止位，否则发送一个。
B200	200 波特	CREAD	启用接收器。
B300	300 波特	PARENB	启用奇偶校验。
B600	600 波特	PARODD	启用奇校验，否则启用偶校验。
B900	900 波特	HUPCL	在最后一次关闭时挂起。
B1200	1200 波特	CLOCAL	本地线路，否则使用拨号。
B1800	1800 波特	LOBLK	保留以供 *shl*(1) 使用。
B2400	2400 波特		
B3600	3600 波特		
B4800	4800 波特		
B7200	7200 波特		
B9600	9600 波特		
B19200	19200 波特		
B38400	38400 波特		
EXTA	外部 A		
EXTB	外部 B		

CBAUD 位指定波特率。零波特率 **B0** 用于挂起连接。如果指定 **B0** ，则停止声明调制解调器控制线（请参阅 *modem*(7) ）。通常，这会断开线路连接。对于任何特定硬件，将忽略不可能的速度改变。提供 **CBAUD** 是为了与 **termio** 结构一起使用。使用 **termios** 结构时，几个例行程序可用于设置和获取输入和输出波特率（请参阅“与 termios 结构相关的函数”一节）。

CSIZE 位指定传输和接收的字符大小（以位为单位）。此大小不包括奇偶校验位（如果有的话）。如果设置了 **CSTOPB** ，则使用两个停止位；否则使用一个停止位。例如，对于 110 波特，许多设备需要两个停止位。

如果设置了 **PARENB** ，则启用奇偶校验生成（将奇偶校验位添加到每个输出字符）。此外，启用奇偶校验检测（检查传入字符的奇偶校验是否正确）。如果设置了 **PARENB** ，则 **PARODD** 指定奇校验（如果设置的话）；否则使用偶校验。如果清除了 **PARENB** ，则同时禁用奇偶校验生成和奇偶校验检查。

如果设置了 **CREAD** ，则启用接收器。否则无法接收字符。

HUPCL 和 **CLOCAL** 位的特定效果取决于生效的调制解调器控制的模式和类型。有关详细信息，请参阅 *modem*(7) 。

如果设置了 **HUPCL** ，则在使用打开端口的最后一个进程关闭它或终止时，端口的调制解调器控制线将降低（断开连接）。

如果设置了 **CLOCAL** ，则连接与调制解调器状态线的状态无关。如果清除了 **CLOCAL** ，则监视调制解调器状态线。

正常情况下，对 **read()** 的调用将等待调制解调器连接完成。但是，如果设置了 **O_NDELAY** 或 **O_NONBLOCK** 标记，或者设置了 **CLOCAL** ，则 **open()** 立即返回，而不等待连接。如果设置了 **CLOCAL** ，请参阅“调制解调器断开连接”以了解 **read()** 和 **write()** 对尚未为其建立连接或已经失去连接的那些文件的影响。

LOBLK 由 Shell 层工具使用（请参阅 *shl*(1) ）。Shell 层工具不是通用终端接口的一部分，通用终端接口不检查 **LOBLK** 位。

打开后的初始硬件控制值为 **B300** 、 **CS8** 、 **CREAD** 和 **HUPCL** 。

本地模式

c_lflag 字段用于控制终端功能。

ISIG	启用信号。
ICANON	规范的输入（清除和终止处理）。
XCASE	规范的大写（或小写）表示形式。
ECHO	启用回显。
ECHOE	在更正退格符序列时回显 ERASE。
ECHOK	回显抹行字符后的 NL。
ECHONL	回显 NL。
NOFLSH	中断、退出或挂起后禁用刷新。
TOSTOP	为后台输出发送 SIGTTOU。
ECHOCTL	将控制字符回显为 ^char，将 DEL 回显为 ^?。
ECHOPRT	在清除字符时回显已清除的字符。
ECHOKE	BS SP BS 在终止行时清除整个行。
FLUSHO	正在刷新输出。
PENDIN	在下一读取或输入字符处重新处理挂起的输入。

IEXTEN 启用扩展功能。

如果设置了 **ISIG**，则对照特殊控制字符 INTR、 QUIT、 SUSP 和 DSUSP 检查每个输入字符（请参阅“进程组控制 IOCTL 命令”）。如果输入字符与其中一个控制字符匹配，则执行与此字符关联的功能，并忽略此字符。如果清除了 **ISIG**，则不进行检查，并将字符视为普通数据字符。因此，仅当设置了 **ISIG** 时这些特殊输入功能才是有可能的。

如果设置了 **ICANON**，则启用规范处理。这将启用清除和终止编辑功能，并允许将输入字符组合为由 NL、EOF、 EOL 或 EOL2 分隔的行。如果清除了 **ICANON**，则直接从输入队列满足读取请求。在接收到至少 MIN 个字符或超时值 TIME 在字符之间到期之前，一直阻塞读取（请参阅“非标准模式输入处理（MIN/TIME 交互）”一节）。这样就可以在仍允许单字符输入的情况下，高效读取快速突发的输入。时间值表示 1/10 秒。

如果设置了 **XCASE**，并设置了 **ICANON**，则在输入时接受大写字母，（方法是在其前面添加 **** 字符），在输出时前面带有 **** 字符。在此模式下，在输出时将生成且在输入时将接受下列转义序列：

要获取：	请使用：
‘	\’
|	\!’
{	\(
}	\)
\	\\

例如， **A** 输入为 **\a**， **\n** 输入为 **\\n**， **\N** 输入为 **\\\n**。对于仅支持前六十四个字符有限字符集的终端， **XCASE** 通常与 **IUCLC** 和 **OLCUC** 联合使用。在这种情况下，在输入时 **IUCLC** 的处理在 **XCASE** 之前进行，而在输出时处理在 **XCASE** 之后进行。因此，由于设置了 **IUCLC**，键入 **A** 会导致读取 **a**；由于 **IUCLC** 生成 **\a**，然后由 **XCASE** 处理转换为 **A**，键入 **\A** 会导致读取 **A**。

如果设置了 **ECHO**，则字符在接收到时将重新回显到终端。如果清除了 **ECHO**，则不回显字符。

如果设置了 **ICANON**，则启用规范处理。这将启用清除和终止编辑功能，并允许将输入字符组合为由 NL、EOF、 EOL 和 EOL2 分隔的行，如“标准模式输入处理”中所述。此外，下列回显功能是可能的。

如果设置了 **ECHO** 和 **ECHOE**，则将 ERASE 和 WERASE 字符回显为三字符 ASCII 序列 BS SP BS，这将从 CRT 屏幕清除最后一个字符或字。

如果设置了 **ECHO** 和 **ECHOPRT**，并清除了 **ECHOE**，则序列中的第一个 ERASE 和 WERASE 字符回显一个反斜杠 (\)，后跟要清除的字符。随后的 ERASE 或 WERASE 字符按相反顺序回显清除的字符。下一个非清除字符导致在回显它之前键入斜线 (/)。

如果设置了 **ECHOKE** 和 **ECHO**，则使用由 **ECHOE** 和 **ECHOPRT** 选择的方法从 CRT 屏幕清除行上的每个字符，以回显 KILL 字符。

如果设置了 **ECHOCTL** 和 **ECHO**，则将除 ASCII TAB、 ASCII NL、 START 和 STOP 字符、 ASCII CR 以及 ASCII BS 之外的所有控制字符（其代码介于八进制的 0 和 37 之间的字符）回显为 ^char，其中 char 是通过将八进制的 100 与控制字符的代码相加所给出的字符。

如果设置了 **ECHOK** 但未设置 **ECHOKE** ，则在抹行字符之后回显 NL 字符，以强调要删除此行。

如果设置了 **ECHONL** ，则回显 NL 字符，即使清除了 **ECHO** 。对于设置为本地回显（即半双工）的终端，这是很有用的。

除非进行转义，否则将不回显 EOF 字符。由于 ASCII EOT 是缺省的 EOF 字符，因此这会阻塞响应 EOT 的终端挂起。

如果设置了 **NOFLSH** ，则不执行与退出、中断和挂起字符关联的输入和输出队列的的正常刷新。但是，如果设置了 **BRKINT** ，则 **NOFLSH** 在接收到中断时不影响数据的刷新。

如果设置了 **TOSTOP** 位，则当不属于前台进程组的进程不忽略且不阻塞 **SIGTTOU** 信号时，将拒绝此进程写入其控制终端的尝试。如果拒绝写入且进程是孤立进程组的成员，则 **write()** 返回 –1，并将 **errno** 设置为 **EIO** ，且不发送信号。如果拒绝写入且进程不是孤立进程组的成员，则将 **SIGTTOU** 信号发送到此进程组。

如果设置了 **FLUSHO** ，则忽略写入终端设备的数据。此位由程序设置。程序可以通过清除 **FLUSHO** 来取消 **FLUSHO** 效果。

如果设置了 **PENDIN** ，则在下一个字符作为输入到达时，会重新处理并可能重新回显尚未读取的任何输入。

如果设置了 **ICANON** ，则可以通过前置的 **** 字符对 ERASE、 KILL 和 EOF 字符进行转义，在这种情况下不执行特殊功能。

在允许 **ECHOCTL** 、 **ECHOPRT** 、 **ECHOKE** 、 **FLUSHO** 和 **PENDIN** 功能之前，必须设置 **IEXTEN** 。此外，仅当设置了 **IEXTEN** 时，才允许特殊字符 WERASE 和 LNEXT 。 **IEXTEN** 不影响任何其他功能。

清除初始本地控制值的所有位。

特殊控制字符

特殊控制字符在数组 **c_cc** 中定义。可以更改所有这些特殊字符。下表显示了标准模式和非标准模式下每个元素的下标名称和说明。

下标用法

规范	非规范	说明
VEOF		EOF 字符
VEOL		EOL 字符
VEOL2		EOL2 字符
VERASE		ERASE 字符
VWERASE		WERASE 字符
VINTR	VINTR	INTR 字符
VKILL		KILL 字符
	VMIN	MIN 值
VQUIT	VQUIT	QUIT 字符
VSTART	VSTART	START 字符

VSTOP	VSTOP	STOP 字符
VSUSP	VSUSP	SUSP 字符
VDSUSP	VDSUSP	DSUSP 字符
	VTIME	TIME 值
VLNEXT	VLNEXT	LNEXT 字符

与 **termios** 结构相关的函数

在使用 *termios* 结构时提供了下列函数。请注意，在成功调用 **tcsetattr()** 函数之后， **cfsetispeed()** 和 **cfsetospeed()** 函数对终端设备的作用才会生效。有关详细信息，请参阅相应的手册条目。

termios 结构函数

函数	说明
cfgetospeed()	获取输出波特率
cfgetispeed()	获取输入波特率
cfsetospeed()	设置输出波特率
cfsetispeed()	设置输入波特率
tcgetattr()	获取终端状态
tcsetattr()	设置终端状态

与 **termio** 结构相关的 **OCTL** 命令

几个 **ioctl()** 系统调用适用于使用 **termio** 结构（请参阅“termio 结构”）的终端文件。如果无法识别所请求的命令，则请求返回 –1，并将 **errno** 设置为 [EINVAL] 。

引用 **termio** 结构的 **ioctl()** 系统调用的格式如下：

```
ioctl (fildes, command, arg)
struct termio *arg;
```

使用此格式的命令如下：

TCGETA 获取与终端关联的参数，并将其存储在由 *arg* 引用的 **termio** 结构中。允许从后台进程执行此命令；但是信息可能随后由前台进程更改。

TCSETA 从 *arg* 引用的 **termio** 结构设置与终端关联的参数。更改是即时的。如果在请求命令时输出字符，则结果是未定义的，而且输出可能是乱码。

TCSETAW 在设置新参数之前等待输出完毕。在更改影响输出的参数时应该使用此格式。

TCSETAF 等待输出完毕，然后刷新输入队列并设置新参数。

termio 警告

只能设置或返回前八个特殊控制字符（请参阅“termios 结构”）。索引 VEOL 和 VEOF 的值分别与索引 VTIME 和 VMIN 的值相同。因此，如果设置了 **ICANON** ，则 VEOL 或 VTIME 是附加的行结束标记字符， VEOF 或 VMIN 是文件结束标记字符。如果清除了 **ICANON** ，则 VEOL 或 VTIME 是字符间计时器值， VEOF 或 VMIN 是读取所需的最小字符数。

与结构无关的函数

为控制终端提供了与 **termio** 和 **termios** 结构均无关的下列函数。有关详细信息，请参阅相应的手册条目。

与结构无关的函数

函数	说明
tcsendbreak()	发送中断
tcdrain()	等待，直到输出完毕
tcflush()	刷新输入或输出队列或者刷新这两者
tcflow()	挂起或恢复输入或输出
tcgetpgrp()	获取前台进程组 id
tcsetpgrp()	设置前台进程组 id
tcgetsid()	获取会话 id

系统异步 **I/O IOCTL** 命令

下列 **ioctl()** 系统调用为系统异步 I/O 作准备，其格式如下：

ioctl (fildes, command, arg)
int *arg;

使用此格式的命令如下：

FIOSSAIOSTAT 如果 *arg* 引用的整数不为零，则启用系统异步 I/O；也就是说，每当终端设备文件状态从“没有可用的读取数据”更改为“读取数据可用”时，都允许将 **SIGIO** 发送到当前使用 **FIOSSAIOOWN** （见下文）指定的进程。如果未使用 **FIOSSAIOOWN** 指定进程，则允许将 **SIGIO** 发送到打开终端设备文件的第一个进程。

如果指定的进程已退出，则不将 **SIGIO** 信号发送到任何进程。

如果 *arg* 引用的整数为 0，则禁用系统异步 I/O。

打开终端设备文件时的缺省行为是禁用系统异步 I/O。

FIOGSAIOSTAT 如果启用系统异步 I/O，则将 *arg* 引用的整数设置为 1。否则，将 *arg* 引用的整数设置为 0。

FIOSSAIOOWN 将由于系统异步 I/O 而将接收 **SIGIO** 信号的进程 ID 设置为由 *arg* 引用的整数的值。如果找不到与 *arg* 引用的整数所指定进程相对应的进程，则调用返回 –1，并将 **errno** 设置为 [ESRCH] 。拥有相应权限的用户可以指定任何进程接收 **SIGIO** 信号。如果请求不是由拥有相应权限的用户发出的，且调用进程没有指定其本身或实际、已保存或有效用户 ID 与此进程的实际或有效用户 ID 匹配的其他进程，或者调用进程没有指定是调用进程子代的进程以接收 **SIGIO** 信号，则调用返回 –1，并将 **errno** 设置为 [EPERM] 。有关访问支持精细划分权限系统的权限访问的详细信息，请参阅 *privileges*(5) 。

如果指定的进程随后退出，则不将 **SIGIO** 信号发送到任何进程。

打开终端设备文件时的缺省行为是将执行第一个打开的进程设置为接收 **SIGIO** 信号。

FIOGSAIOOWN 将由 *arg* 引用的整数设置为指定用于接收 **SIGIO** 信号的进程 ID。

行控制 **IOCTL** 命令

几个 **ioctl()** 系统调用控制输入和输出。其中一些调用的格式如下：

```
ioctl (fildes, command, arg)
int arg;
```

使用此格式的命令如下：

TCSBRK 等待输出完毕。如果 *arg* 为 0，则发送中断（零位至少需要 0.25 秒）。 **tcsendbreak()** 函数执行相同的功能（请参阅 *tcsendbreak*(3C) ）。

TCXONC 开始/停止控制。如果 *arg* 为 0，则挂起输出；如果为 1，则重新开始已挂起的输出；如果为 2，则传输 STOP 字符；如果为 3，则传输 START 字符。如果为 *arg,* 指定了任何其他值，则调用返回 –1，并将 **errno** 设置为 [EINVAL] 。 **tcflow()** 函数执行相同的功能（请参阅 *tcflow*(3C) ）。

TCFLSH 如果 *arg* 为 0，则刷新输入队列；如果为 1，则刷新输出队列；如果为 2，则同时刷新输入队列和输出队列。如果为 *arg,* 指定了任何其他值，则调用返回 –1，并将 **errno** 设置为 [EINVAL] 。 **tcflush()** 函数执行相同的功能（请参阅 *tcflush*(3C) ）。

发送 BREAK 是通过在 SPACE 或逻辑零条件处保持数据传输线路至少 0.25 秒完成的。在此时间间隔内，可以将数据发送到设备，但是由于串行数据接口的限制， BREAK 优先于所有数据。因此，在 BREAK 期间发送到设备的所有数据都将丢失。这包括系统生成的用于输入流控制的 XON/XOFF 字符。另请注意，直到终止 BREAK 后才传输 XOFF 流控制字符的延迟，仍可能导致数据溢出，因为流控制字符可能发送得不够快。

其他调用的格式如下：

```
ioctl (fildes, command, arg)
int *arg;
```

使用此格式的命令如下：

FIONREAD 在 *arg* 引用的整数中返回可以立即从终端设备文件读取的字符数。允许从后台进程执行此命令；但是，不能从后台进程读取数据本身。

非阻塞 **I/O IOCTL** 命令

通过在 *open*(2) 和 *fcntl*(2) 中均可用的 **O_NONBLOCK** 和 **O_NDELAY** 标记可轻松提供非阻塞 I/O。本节中的命令是为了与以前开发的应用程序向后兼容提供的。提供不同于 **O_NONBLOCK** 和 **O_NDELAY** 的非阻塞 I/O 样式的 **ioctl()** 系统调用的格式如下：

ioctl (fildes, command, arg)
int *arg;

使用此格式的命令如下:

FIOSNBIO 如果 *arg* 引用的整数不为零，则启用 **FIOSNBIO**样式的非阻塞 I/O；也就是说，以非阻塞方式处理终端设备文件的后续读取和写入（见下文）。如果 *arg* 引用的整数为 0，则禁用 **FIOSNBIO**样式的非阻塞 I/O。

对于读取， **FIOSNBIO**样式的非阻塞 I/O 防止对此设备文件的所有读取请求阻塞，而不管请求成功还是失败。这样的读取请求以下列三种方式之一完成:

- 如果存在的可用数据足以满足整个请求，则读取成功完成，读取所有数据，并返回已读取的字符数;
- 如果存在的可用数据不足以满足整个请求，则读取成功完成，读取尽可能多的数据，并返回已读取的字符数;
- 如果没有可用的数据，则读取返回 –1，并将 **errno** 设置为 [EWOULDBLOCK] 。

对于写入， **FIOSNBIO**样式的非阻塞 I/O 防止对此设备文件的所有写入请求阻塞，而不管请求成功还是失败。这样的写入请求以下列三种方式之一完成:

- 如果系统中存在的可用空间足以缓冲所有数据，则写入成功完成，写出所有数据，并返回已写入的字符数;
- 如果缓冲区中存在的空间不足以写出整个请求，则写入成功完成，写入尽可能多的数据，并返回已写入的字符数;
- 如果缓冲区中没有空间，则写入返回 –1，并将 **errno** 设置为 [EWOULDBLOCK]。

要禁止 **FIOSNBIO**样式的非阻塞 I/O 干扰 **O_NONBLOCK** 和 **O_NDELAY** 标记（请参阅 *open*(2) 和 *fcntl*(2) ） ， **O_NONBLOCK** 和 **O_NDELAY** 的功能始终取代 **FIOSNBIO** 样式的非阻塞 I/O 的功能。也就是说，如果设置了 **O_NONBLOCK** 或 **O_NDELAY** ，则驱动程序根据 **O_NDELAY** 或 **O_NONBLOCK.** 的定义执行读取请求。如果同时清除了 **O_NONBLOCK** 和 **O_NDELAY** ，则 **FIOSNBIO**样式的非阻塞 I/O 的定义将适用。

打开终端设备文件的缺省行为是禁用 **FIOSNBIO**样式的非阻塞 I/O。

FIOGSNBIO 如果启用 **FIOSNBIO**样式的非阻塞 I/O，则将 *arg* 引用的整数设置为 1。否则，将 *arg* 引用的整数设置为 0。

进程组控制 **IOCTL** 命令

使用函数 **tcgetattr()** 、 **tcsetattr()** 、 **tcgetpgrp()** 、 **tcsetpgrp()** 和 **tcsetsid()** （请分别参阅 *tcattribute*(3C) 、 *tcgetpgrp*(3C) 、 *tcsetpgrp*(3C) 和 *tcgetsid*(3C) ）可轻松实现此处所述的进程组控制功能（设置和获取延迟的停止进程字符除外）。

用于进程组控制的以下结构在 **<bsdtty.h>** 中定义：

```
struct ltchars {
    unsigned char t_suspc;     /* stop process character*/
    unsigned char t_dsuspc;    /* delayed stop process character*/
    unsigned char t_rprntc;    /* reserved; must be '_POSIX_VDISABLE'*/
    unsigned char t_flushc;    /* reserved; must be '_POSIX_VDISABLE'*/
    unsigned char t_werasc;    /* reserved; must be '_POSIX_VDISABLE'*/
    unsigned char t_lnextc;    /* reserved; must be '_POSIX_VDISABLE'*/
};
```

所有这些字符的初始值均为 **_POSIX_VDISABLE** ，这导致禁用。每个字符的含义如下：

t_suspc 挂起前台进程组。将 *suspend* 信号 (**SIGTSTP**) 发送到前台进程组中的所有进程。通常，强制每个进程停止，但可以进行这样的安排：忽略或阻塞信号，或者在一致同意的位置接收陷阱；请参阅 *signal*(2) 和 *signal*(5) 。如果启用，则此字符的典型值为 Control-Z 或 ASCII SUB。设置或获取 **t_suspc** 等效于设置或获取 SUSP 特殊控制字符。

t_dsuspc 与 **t_suspc** 相似，不同之处在于，进程在读取字符时发送 *suspend* 信号 (**SIGTSTP**) ，而不是在键入字符时发送此信号。如果启用，则此字符的值通常为 Control-Y 或 ASCII EM。

将任一保留字符设置为除 **_POSIX_VDISABLE** 之外的值的尝试会导致 **ioctl()** 返回 −1，并将 **errno** 设置为 [EINVAL] ，但保留字符的值不变。

使用上述结构的 **ioctl()** 系统调用的格式如下：

ioctl (fildes, command, arg)
struct ltchars *arg;

使用此格式的命令如下：

TIOCGLTC 获取进程组控制字符，并将其存储在由 *arg* 引用的 *ltchars* 结构中。允许从后台进程执行此命令。但是，信息可能随后由前台进程更改。

TIOCSLTC 从 *arg* 引用的结构设置进程组控制字符。

其他进程组控制 **ioctl()** 系统调用的格式如下：

ioctl (fildes, command, arg)
unsigned int *arg;

使用此格式的命令如下：

TIOCGPGRP 在 *arg* 引用的整数中返回与终端关联的前台进程组。允许从后台进程执行此命令。但是，信息可能随后由前台进程更改。使用 **tcgetpgrp()** 函数（请参阅 *tcgetpgrp*(3C) ）可轻松实现此功能。

如果 **ioctl()** 调用失败，则返回 −1，并将 **errno** 设置为下列值之一：

[EBADF] *fildes* 不是有效的文件描述符。

[ENOTTY] 与 *fildes* 关联的文件不是控制终端，或者调用进程没有控制终端。

[EACCES] 与 *fildes* 关联的文件是调用进程的控制终端，但是，没有为控制终端定义前台进程组。

注释： [EACCES] 在将来版本中可能不返回。没有为控制终端定义前台进程组的情况下的行为在 POSIX 标准的将来版本中可能会更改。因此，可移植应用程序不应该依赖于此错误条件。

TIOCSPGRP 将与终端关联的前台进程组设置为 *arg* 引用的值。使用 **tcsetpgrp()** 函数可轻松实现此功能（请参阅 *tcsetpgrp*(3C) ）。

如果 **ioctl()** 调用失败，则返回 −1，并将 **errno** 设置为下列值之一：

[EBADF] *fildes* 不是有效的文件描述符。

[EINVAL] *arg* 引用的进程 ID 不是受支持的值。

[ENOTTY] 调用进程没有控制终端，或者 *fildes* 不是控制终端，或者控制终端不再与调用进程的会话关联。

[EPERM] *arg* 引用的进程 ID 是受支持的值，但其与调用进程所在会话中进程的进程组 ID 不匹配。

TIOCGSID 在 *arg* 引用的整数中返回由 fildes 指定的终端的会话 ID。使用 **tcgetsid()** 函数可轻松实现此功能（请参阅 *tcgetsid*(3C) ）。

如果 **ioctl()** 调用失败，则返回 −1，并将 **errno** 设置为下列值之一：

[EBADF] *fildes* 不是有效的文件描述符。

[ENOTTY] 与 *fildes* 关联的设备不是终端。

[EACCES] *fildes* 是未分配给会话的终端。

TIOCLGET 获取进程组控制模式字，并将其存储在由 *arg* 引用的 int 中。允许从后台进程执行此命令；但是信息可能随后由前台进程更改。

TIOCLSET 将进程组控制模式字设置为由 *arg* 引用的 int 的值。

TIOCLBIS 将 *arg* 引用的 int 用作要在进程组控制模式字中设置的位的掩码。

TIOCLBIC 将 *arg* 引用的 int 用作要在进程组控制模式字中清除的位的掩码。

以下位在进程组控制模式字中定义：

LTOSTOP 发送 **SIGTTOU** 以用于后台写入。

设置或清除 **LTOSTOP** 等效于设置或清除 **TOSTOP** 标记（请参阅“本地模式”）。如果设置了 **LTOSTOP**，并且进程不在其控制终端的前台进程组中，则可能拒绝进程写入其控制终端（请参阅“终端访问控制”）。

终端大小 IOCTL 命令

下列 **ioctl()** 系统调用用于获取和设置有关由 *fildes* 引用的终端的终端大小信息。这些 **ioctl()** 系统调用使用 **winsize** 结构获取和设置终端大小信息。在 **<termios.h>** 中定义的 **winsize** 结构拥有下列成员：

```
unsigned short  ws_row;    /* Rows, in characters */
unsigned short  ws_col;    /* Columns, in characters */
unsigned short  ws_xpixel; /* Horizontal size, in pixels */
unsigned short  ws_ypixel; /* Vertical size, in pixels */
```

终端大小的所有元素的初始值均为零。通用终端接口既不设置也不使用终端大小的值，而且这些值对通用终端接口的功能没有影响。终端大小的值仅由通过终端大小 **ioctl()** 系统调用（请参阅 *ioctl*(2)）访问它们的应用程序设置和使用。

使用上述结构的 **ioctl()** 系统调用的格式如下：

```
ioctl (fildes, command, arg)
struct winsize *arg;
```

使用此格式的命令如下：

TIOCGWINSZ 获取终端大小值，并将其存储在由 *arg.* 引用的 **winsize** 结构中。允许从后台进程执行此命令。

TIOCSWINSZ 从 *arg.* 引用的 **winsize** 结构设置终端大小值。如果任一新值与以前的值不同，则会将 **SIGWINCH** 信号发送到终端的前台进程组中的所有进程。

控制台输出重定向 IOCTL 命令

可能会将通常发送到系统控制台的输出重定向到系统中的任何其他 TTY 设备或伪设备。用于控制控制台输出重定向的 **ioctl()** 系统调用的格式如下：

```
ioctl (fildes, command, arg)
int  arg;
```

使用此格式的命令如下：

TIOCCONS 重定向系统控制台输出。通常通过内核 printf 请求或通过控制台专用文件发送到系统控制台的任何输出，都将改为发送到由 *fildes* 引用的终端。忽略 *arg* 的值。用户必须拥有 **DEVOPS** 权限，才能执行此请求。否则，调用返回 −1，并将 **errno** 设置为 [EPERM]。如果尚未通过稍后调用此命令将控制台输出重定向到其他设备，则在关闭 *fildes* 时将其重定向回物理控制台设备。

警告

多个 HP-UX 实现使用类似终端的非串行接口（如位映射图形显示），或者使用无法实现上述确切功能的“智能卡”。因此，并不是所有的系统都可以完全符合上述标准。每个实现需要在其系统特定的文档中指出与标准的任

何偏差。

FIOSSAIOSTAT 与 BSD 4.2 **FIOASYNC** 类似，但增加了安全性条款。

FIOGSAIOSTAT 出自 HP ，对 **FIOSSAIOSTAT** 进行了补充，并允许保存和恢复命令解释程序进程的系统异步 I/O TTY 状态。

FIOSSAIOOWN 与 BSD 4.2 **FIOSETOWN** 类似，但增加了安全性条款。

FIOGSAIOOWN 与 BSD **FIOGETOWN** 类似。4.2 另请注意以下差异：此功能的 BSD 4.2 版本使用进程组，而 HP-UX 版本仅使用进程。

FIOSNBIO 与 BSD **FIONBIO** 4.2 相似，不同之处在于前者不干扰 **O_NDELAY** 或 **O_NONBLOCK** 以及 **open()** 和 **fcntl()** 标记。

FIOGNBIO 出自 HP，对 **FIOSNBIO** 进行了补充，并允许保存和恢复命令解释程序进程的 **FIOSNBIO**样式非阻塞 I/O TTY 状态。

通用终端接口使用称为 **cblock** 的系统资源，以存储通过通信端口传输或接收的数据。随着数据通过系统，持续地使用这些 cblock 并释放以供重用。如果在系统中配置的 cblock 太少，则 cblock 池可能会暂时或永久地耗尽，这可能导致数据丢失、系统挂起或系统性能降低。

如果怀疑 cblock 已耗尽，您可以使用 **dmesg** （请参阅 *dmesg*(1M) ）检查系统消息缓冲区，以查找指示 cblock 已耗尽的消息。或者，如果检查转储的核心文件，则可以使用 **adb** （请参阅 *adb*(1) ）。消息格式如下：

WARNING: cblock exhaustion occurred n times

其中 n 指示操作系统已请求 cblock 的次数，但无法提供任何 cblock。如果监视到此消息，则应重新配置内核，以生成更大数量的 cblock。

cblock 的长度为 32 字节。系统中配置的 cblock 缺省数目定义为 8292。

通过使用可选的可调系统参数 **nclist** 指定需要在系统中使用的 cblock 数，可以覆盖此数目。

可以使用 **SAM** 或 *kctune*(1M) 更改 **nclist** 值。

相关内容

工作站

工作站计算机上的内置串行端口支持下列其他波特率设置： 57 600 和 115 200。可能需要有 RS-232 到 RS-422 的转换器，才能实现这些波特率上的实际电缆长度（因为 RS-232 最多只能指定 19 200 波特）。

不支持计时延迟。

工作站系统上的内置串行端口具有 RTS 和 CTS 流控制功能、可配置的接收 FIFO 触发器级别和可配置的传输限制。通过次设备文件号中的位、通过 **ioctl()** 调用（请参阅 *termiox*(7) ）或通过 **stty** 命令（请参阅 *stty*(1) ），可以启用 RTS/CTS 硬件握手。

通过从属设备文件号中的两位，可以配置接收 FIFO 触发器级别。接收 FIFO 触发器级别用于设置为系统生成接收中断的级别。为接收 FIFO 触发器级别设置较小的值，使系统可以更快地对字符的接收作出反应。但是，使用

较低的触发器级别会增加处理其他中断的系统开销。使用较高的接收 FIFO 触发器级别会减小大量入站数据流量的系统中断开销，而付出的代价是系统在接收 FIFO溢出之前从硬件读取数据的时间更少了。使用 RTS 流控制时，接收 FIFO 触发器级别还确定硬件在何时降低 RTS 以保护接收 FIFO。使用较高的接收 FIFO 触发器级别还会降低 XOFF 流控制的响应速度，原因是在入站数据流较小的情况下，系统接收到 XOFF 字符会稍微延迟。应该根据如何使用串行端口来选择适当的接收 FIFO 触发器级别。对于大多数应用程序，建议使用的接收 FIFO 触发器级别为 8 (c3,c2 = 10)。

从属设备文件号中的两位指定传输限制，即连续加载到传输 FIFO 中的字符数。设置较小的传输限制，可允许发送器对接收 XOFF 字符或取消对 CTS 的声明产生的流控制作出更快的响应，但代价是增加了系统中断开销。设置较大的传输限制会减小中断开销，但是会降低响应流控制的速度，因为即使在对发送器进行流控制之后，也可以传输传输 FIFO 的剩余部分。与在进行流控制后几乎不容许接收数据的设备进行通信时，必须适当地选择传输限制。

从属设备文件号

工作站从属设备文件号的格式如下：

0x*IIC***0***HM*

其中：

II = 两个十六进制数字（8 位），指示串行接口的实例。

C = 一个十六进制数字（4 位），用于 FIFO 控制。每位的值如下所示：

接收 **FIFO** 触发器级别			传输限制		
c3	**c2**	级别	**c1**	**c0**	限制
0	0	1	0	0	1
0	1	4	0	1	4
1	0	8	1	0	8
1	1	14	1	1	12

H = 一个十六进制数字（4 位），控制诊断访问和硬件流控制。

位	值
h3	诊断电话访问
h2	保留
h1	保留
h0	启用 RTS/CTS 硬件流控制

M = 一个十六进制数字（4 位），用于确定端口访问类型。每位的值如下所示：

位	值
m3	TI/ALP
m2	0 = 简单协议（美国）， 1 = CCITT 协议（欧洲）
m1m0	00 = 直接 01 = 拨出调制解调器 10 = 拨入调制解调器 11 = 无效

服务器

不直接支持计时输出延迟。如果使用，则将输出相应的填充字符数（基于当前的波特率）。输出填充字符的总时间至少与请求的时间一样长。

系统指定的输入流控制值如下：低水印为 60，高水印为 180，允许的最大输入为 512。

HP 98196A （以前为 27140A 选项 800）接口不支持下列硬件设置：

CBAUD **B200** 、 **B38400** 、 **EXTA** 、 **EXTB** 。

HP A1703-60003 和 HP 28639-60001 接口不支持超过 9600 的波特率。此外，不支持从缺省设置（9600 波特、8 位字符、1 个停止位、无奇偶校验）更改端口 0 上的下列硬件设置：

CBAUD 、 **CSIZE** 、 **CSTOPB** 、 **PARENB** 、 **PARODD** 。

HP J2094A 接口不支持超过 19200 的波特率。

HP J2094A 支持 RTS/CTS 流控制。通过从属设备文件号中的位、通过 **ioctl()** 调用（请参阅 *termiox*(7) ）或通过 **stty** 命令（请参阅 *stty*(1) ），可以启用 RTS/CTS 硬件握手。

从属设备文件号

服务器从属设备文件号的格式如下：

0x*IIPPHM*

其中：

II = 两个十六进制数字（8 位），指示串行接口的实例。

PP = 两个十六进制数字（8 位），指示此设备在串行接口上的端口号。

H = 一个十六进制数字（4 位），控制诊断访问和硬件流控制（仅适用于 HP J2094A）。

位	值
h3	卡诊断
h2	端口诊断
h1	保留
h0	启用 RTS/CTS 硬件流控制

M = 一个十六进制数字（4 位），表示端口访问类型。每位的值如下所示：

位	值
m3	TI/ALP
m2	0 = 简单协议（美国）， 1 = CCITT 协议（欧洲）
m1m0	00 = 直接 01 = 拨出调制解调器 10 = 拨入调制解调器 11 = 无效

作者

termios 由 HP 和 IEEE Computer Society 联合开发。

termio 由 HP、AT&T 和加州大学伯克利分校联合开发。

文件

/dev/console
/dev/cua*
/dev/cul*
/dev/tty*
/dev/ttyd*

另请参阅

adb(1)、 shl(1)、 stty(1)、 dmesg(1M)、 kctune(1M)、 mknod(1M)、 fork(2)、 ioctl(2)、 setpgid(2)、 setsid(2)、 signal(2)、 stty(2)、 cfspeed(3C)、 tcattribute(3C)、 tccontrol(3C)、 tcgetpgrp(3C)、 tcgetsid(3C)、 tcsetpgrp(3C)、 privileges(5)、 signal(5)、 unistd(5)、 modem(7)、 sttyV6(7)、 termiox(7)、 tty(7)。

符合的标准

termio：SVID2、SVID3、XPG2

termios：AES、SVID3、XPG3、XPG4、FIPS 151-2、POSIX.1

名称

termiox - 扩展的通用终端接口

概要

#include <sys/termiox.h>

ioctl (int *fildes*, **int** *request*, **struct termiox *** *arg)*

说明

扩展的通用终端接口通过增添对异步硬件流控制和附加异步功能的本地实现，对 *termio*(7) 通用终端接口进行补充。由于硬件或软件限制，一些系统可能不支持所有这些功能。其他系统可能不允许禁用某些功能。在这种情况下，将忽略相应位。如果可以支持这些功能，则必须使用在此介绍的接口。

硬件流控制模式

硬件流控制可补充 termio **IXON**、 **IXOFF** 和 **IXANY** 字符流控制（请参阅 *termio*(7) ）。当一个设备通过在设备之间的数据流中插入控制字符来控制另一设备的数据传输时，将出现字符流控制。当一个设备通过使用异步接口线路（电路）上的电子控制信号来控制另一设备的数据传输时，将出现硬件流控制。可以同时设置字符流控制和硬件流控制。

在异步全双工应用程序中，使用美国电子工业协会的 EIA-232-D 请求发送 (RTS) 和清除发送 (CTS) 电路是硬件流控制的首选方法。

EIA-232-D 标准仅指定了单向硬件流控制，其中数据电路端接设备或数据通信设备 (DCE) 表示要停止传输数据的数据终端设备 (DTE)。通过 termiox 接口，可同时启用单向和双向硬件流控制；当双向硬件流控制启用时， DCE 或 DTE 都可以指示对方通过接口停止传输数据。

时钟模式

不支持等时流控制和时钟模式通信。

终端参数

控制提供 termiox 接口的设备的行为的参数由 **termiox** 结构指定，该结构在 **<sys/termiox.h>** 头文件中定义。多个提取或更改这些参数的 **ioctl()** 系统调用（请参阅 *ioctl*(5) ）可使用 **termiox** 结构，该结构包含下列成员：

```
unsigned short x_hflag;    /* hardware flow control modes */
unsigned short x_cflag;    /* clock modes */
unsigned short x_rflag;    /* reserved modes */
unsigned short x_sflag;    /* spare local modes */
```

x_hflag 字段描述了硬件流控制模式：

RTSXOFF 0000001	在输入中启用 RTS 硬件流控制。
CTSXON 0000002	在输入中启用 CTS 硬件流控制。

建立 CCITT 调制解调器连接时，会使用 RTS 和 CTS 电路。由于 CCITT 调制解调器连接和硬件流控制都会使用 RTS 和 CTS 电路，因此不能同时启用 CCITT 调制解调器和硬件流控制。

通过设置相应位，可以选择各种硬件流控制方法的变化形式。例如，通过设置 **RTSXOFF** 和 **CTSXON** 位，可选

择双向 RTS/CTS 流控制。通过仅设置 **CTSXON** 位，可选择单向 CTS 硬件流控制。

如果设置了 **RTSXOFF** ，则将增加请求发送 (RTS) 电路（线路），如果异步端口需要停止其输入，则它将减少请求发送 (RTS) 线路。如果减少了 RTS 线路，则假定所连接的设备会停止其输出，直到增加了 RTS 为止。

如果设置了 **CTSXON** ，则仅当所连接的设备增加了清除发送 (CTS) 电路（线路）时，才进行输出。如果所连接的设备减少了 CTS 线路，则会暂停输出，直到增加了 CTS 为止。

与 **termiox** 结构相关的 **IOCTL** 命令

引用 **termiox** 结构的 **ioctl()** 系统调用具有如下格式：

ioctl (fildes, command, arg)
struct termiox *arg;

使用该格式的命令包括：

TCGETX 该参数是指向 **termiox** 结构的指针。将提取当前终端参数，并将其存储到该结构中。

TCSETX 该参数是指向 **termiox** 结构的指针。当前终端参数是利用该结构中存储的值设置的。更改会立即生效。可以返回的错误包括：

[EINVAL] 端口不支持硬件流控制。

[ENOTTY] 该端口的文件描述符是针对 CCITT 模式访问进行配置的。 CCITT 模式设备不允许硬件流控制。

TCSETXW 该参数是指向 **termiox** 结构的指针。当前终端参数是利用该结构中存储的值设置的。传输完排队等待输出的所有字符之后，将发生更改。当更改影响输出的参数时，应使用该格式。可以返回的错误包括：

[EINVAL] 端口不支持硬件流控制。

[ENOTTY] 该端口的文件描述符是针对 CCITT 模式访问进行配置的。 CCITT 模式设备不允许硬件流控制。

TCSETXF 该参数是指向 **termiox** 结构的指针。当前终端参数是利用该结构中存储的值设置的。传输完排队等待输出的所有字符之后，将发生更改；将忽略排队等待输入的所有字符，然后又将发生更改。可以返回的错误包括：

[EINVAL] 端口不支持硬件流控制。

[ENOTTY] 该端口的文件描述符是针对 CCITT 模式访问进行配置的。 CCITT 模式设备不允许硬件流控制。

作者

termiox 由 HP 和 AT&T 联合开发。

文件

/dev/tty* 中或位于其下级目录中的文件。

另请参阅

ioctl(2)、termio(7)、modem(7)。

名称

timod - 将 ioctl() 调用转换为传输接口消息的 STREAMS 模块

说明

timod 模块是一个 STREAMS 模块，它将来自于支持传输接口 (TI) 的传输用户的 **ioctl()** 调用转换为支持 TI 的传输协议提供商可使用的消息。这样，用户可以将某些 TI 功能作为原子操作启动。每当执行 *t_open*(3) 时，此版本的 HP-UX 不再自动压入 **timod** 。已经对 TLI 和 XTI 库进行了修改，使其不再需要该模块来执行本联机帮助页中介绍的原子操作。由于模块仍存在于内核中，因此二进制兼容性无关紧要。但是，已重新编译的、并需要自动压入模块的任何应用程序，在不修改代码的情况下可能无法运行。

用户通过调用 STREAMS **I_PUSH ioctl()** 和 **I_POP ioctl()** 函数，可在设备流上放置和删除 **timod** 模块（TLI 函数 **t_open()** 将 **timod** 压入用户的设备流中）。仅应将 **timod** 模块压入由遵循传输接口的传输提供商终止的流中。*tirdwr*(7) 是支持 **read()** 和 **write()** 系统调用的 **timod** 的备用接口。如果已将 **tirdwr** 压入流中，在压入 **timod** 之前，用户应该使用 **I_POP ioctl** 从流中删除 **tirdwr** 模块。

timod 模块以透明方式，将不是由如下所述的 **ioctl()** 命令生成的所有 STREAMS 消息传递给相邻模块或驱动程序。 **timod** 将对其 **strioctl.ic_cmd** 字段为下列值之一的 **I_STR ioctl()** 起作用（有关 **I_STR ioctl** 和 **strioctl** 结构的说明，请参阅 *streamio*(7) ）。

TI_BIND 该 TI 命令将地址绑定到传输协议提供商。模块发送给 **TI_BIND ioctl()** 调用的 STREAMS 消息，等效于 **T_bind_req** 类型的 TI 消息。模块为响应 **TI_BIND ioctl()** 调用的成功完成而返回的 STREAMS 消息，等效于 **T_bind_ack** 类型的 TI 消息。

TI_UNBIND 该 TI 命令从传输协议提供商解除地址绑定。模块发送给 **TI_UNBIND ioctl()** 调用的 STREAMS 消息，等效于 **T_unbind_req** 类型的 TI 消息。模块为响应 **TI_UNBIND ioctl()** 调用的成功完成而返回的 STREAMS 消息，等效于 **T_ok_ack** 类型的 TI 消息。

TI_GETINFO 该 TI 命令从传输协议提供商获取 TI 协议特定的信息。模块发送给 **TI_GETINFO ioctl()** 调用的 STREAMS 消息，等效于 **T_info_req** 类型的 TI 消息。模块为响应 **TI_GETINFO ioctl()** 调用的成功完成而返回的 STREAMS 消息，等效于 **T_info_ack** 类型的 TI 消息。

TI_OPTMGMT 该 TI 命令获取和设置 TI 协议特定的选项，或与传输协议提供商协商该选项。模块发送给 **TI_OPTMGMT ioctl()** 调用的 STREAMS 消息，等效于 **T_optmgmt_req** 类型的 TI 消息。模块为响应 **TI_OPTMGMT ioctl()** 调用的成功完成而返回的 STREAMS 消息，等效于 **T_optmgmt_ack** 类型的 TI 消息。

返回值

如果 **timod** 模块返回 **ioctl()** 调用的错误，则返回值的较低 8 位将是 **<tiuser.h>** 头文件中定义的 TI 错误代码之一。如果 TI 错误类型为 TSYERR，则返回值的第二个 8 位将包含 **<errno.h>** 头文件中定义的错误。当 **ioctl()** 调用产生错误时模块发送的 STREAMS 消息，等效于 **T_error_ack** 类型的 TI 消息。

文件

<xti.h> 定义 XTI 函数的错误代码。

<tiuser.h> 定义 TI 函数的错误代码。

<tihdr.h> 定义 TI 函数的消息类型。

<errno.h> 定义系统错误的错误代码。

另请参阅

ioctl(2)、 t_open(3)、 streamio(7)、 tirdwr(7)。

名称

tirdwr - 供传输接口用户读取和写入的 STREAMS 模块

说明

tirdwr 模块是一个 STREAMS 模块，它为支持传输接口 (TI) 的传输用户提供支持 TI 的传输协议提供商的备用接口。通过该备用接口，传输用户可使用 **read()** 和 **write()** 函数与传输协议提供商进行通信。它也可以继续使用 **putmsg()** 和 **getmsg()** 函数，但这些函数仅在用户进程和设备流之间传输数据消息。不应将 **getpmsg()** 和 **putpmsg()** 与 **tirdwr** 一起使用。

用户通过调用 STREAMS **I_PUSH ioctl()** 函数，将 **tirdwr** 模块放置到设备流中。 **tirdwr** 是 *timod*(7) 的备用接口。如果已将 **timod** 压入到设备流中，则在压入 **tirdwr** 之前，用户应使用 **I_POP ioctl** 次设备流中删除 **timod** 模块。只应将 **tirdwr** 模块压入到由遵循传输接口的传输提供商终止的设备流中。将模块压入到设备流后，用户将无法再调用 TI 功能。如果用户尝试继续调用 TI 函数，则设备流中将发生错误。检测到错误后，后续调用将失败，并将 **errno** 设置为 [EPROTO]。用户通过调用 STREAMS **I_POP ioctl()** 函数，从设备流中删除 **tirdwr** 模块。

压入和取出时的模块行为

将 **tirdwr** 模块压入到设备流时，它将检查确定要发送到用户的任何现有消息，来确定其消息类型。如果现有消息是常规数据消息，则它会将消息转发给用户。它会忽略与进程管理相关的所有消息，例如，为用户生成信号的消息。如果出现任何其他消息，则它将向用户请求返回错误，并将 **errno** 设置为 [EPROTO]。

如果 **tirdwr** 模块从设备流中取出，则它将检查以前是否从传输协议提供商收到了有序释放指示。如果收到了有序释放指示，则它会向传输连接的远程端发送有序释放请求。关闭设备流后， **tirdwr** 模块仍以这种方式操作。

读取和写入的模块行为

当 **tirdwr** 模块从传输协议提供商收到的消息不包含控制部分时（请参阅 *putmsg*(2) 和 *getmsg*(2) 参考页），则它将以透明方式将该消息发送给上行的相邻模块。但零长度的数据消息除外，这种情况下，模块将释放该消息，并且不会将它们向传递给上行相邻模块。

当模块从传输协议提供商收到包含控制部分的消息时，它将采取下列操作之一:

对于含有控制部分的数据消息，它将删除该部份，然后将消息传递给上行的相邻模块。

对于表示加急数据的消息，它将生成错误。进一步的系统调用将失败，并将 **errno** 设置为 [EPROTO]。

对于表示来自传输协议提供商的有序释放指示的消息，它将生成零长度的数据消息，表示文件结束标志 (EOF)，并将该消息向上行发送给读取进程。将释放包含有序释放指示的原始消息。

对于表示来自传输协议提供商的异常断开指示的消息，它将导致所有进一步的 **write()** 和 **putmsg()** 调用失败，并将 **errno** 设置为 [ENXIO]。读取完先前的所有数据后，后续的 **read()** 和 **getmsg()** 调用将返回零长度的消息，表示文件结束标志 (EOF)。

对于其他所有消息，它将生成错误，进一步的调用将失败，并将 **errno** 设置为 [EPROTO]。

另请参阅

getmsg(2)、putmsg(2)、read(2)、write(2)、t_open(3)、streamio(7)、timod(7)。

名称

tty - 控制终端接口

说明

文件 **/dev/tty** 是每个进程中与该进程的进程组相关联的控制终端的同义名称（如果有的话）。它对需要确保无论怎样重定向输出，都可在终端上写入消息的程序或 Shell 序列很有用。它还可以用于那些在需要键入的输出而又很难找到当前所用终端时需要获得输出文件名的程序。

文件

/dev/tty

/dev/tty*

另请参阅

termio(7).

符合的标准

tty：AES、SVID2、SVID3、XPG2、XPG3、XPG4、FIPS 151-2 和 POSIX.1

名称

UDP - Internet 用户数据报协议

概要

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

s = socket(AF_INET, SOCK_DGRAM, 0);
s = socket(AF_INET6, SOCK_DGRAM, 0);

说明

UDP 是一个简单的不可靠数据报协议，用于支持 Internet 协议系列的 **SOCK_DGRAM** 套接字类型。 UDP 套接字是无连接的套接字，通常与 **sendto()** 和 **recvfrom()** 调用一起使用（请参阅 *send*(2) 和 *recv*(2) ）。 **connect()** 调用也可用于模拟连接（请参阅 *connect*(2) ）。按该方式使用时，它固定了将来传输数据包的目标（在这种情况下，可使用 **send()** 或 **write()** 系统调用），并且指定了从中接收数据包的源。如果信息源不重要，则可随时使用 **recv()** 和 **read()** 调用。

UDP 地址格式与 TCP 所用的地址格式相同。尤其是， UDP 除普通 Internet 地址格式外，还需要一个端口标识符。请注意， UDP 端口域与 TCP 端口域是分开的（即， UDP 端口无法与 TCP 端口连接）。

UDP 套接字缺省的发送缓冲区大小为 65535 字节。 UDP 套接字缺省的接收缓冲区大小为 2147483647 字节。 UDP 套接字发送和接收缓冲区大小可通过使用 **setsockopt()** 系统调用的 **SO_SNDBUF** 和 **SO_RCVBUF** 选项，或 **t_optmgmt()** 系统调用的 **XTI_SNDBUF** 和 **XTI_RCVBUF** 选项设定。这些缓冲区的最大大小为 2147483647 字节。使用 **ndd** 参数 **udp_recv_hiwater_max** 可减小接收缓冲区的最大大小。

UDP 数据报套接字的最大消息大小受 IP 数据报最大大小和 UDP 数据报套接字缓冲区大小两者之中较小者的限制。 IP 数据报最大大小将 UDP 消息最大消息大小限制为 65507 字节。因此，使用套接字缓冲区最大大小允许在发送列队中放入多个最大大小的消息。UDP 数据报套接字缺省的入站和出站消息大小限制为 65535 字节。

UDP 广播的最大消息大小受基础链路的 MTU 大小的限制。

错误

如果套接字操作失败，则可能在 **errno** 中返回下列错误之一。有关错误的详细列表，请参阅特定系统调用的联机帮助页。

[EISCONN]	连接套接字以后，尝试发送指定目标地址的数据报。
[ENOBUFS]	没有可用于内部数据结构的缓冲区空间。
[EADDRINUSE]	尝试创建已经分配了端口的套接字。
[EADDRNOTAVAIL]	尝试用不存在网络接口的网络地址创建套接字。

作者

UDP 套接字接口由加州大学伯克利分校开发。

另请参阅

ndd(1M)、 getsockopt(2)、 recv(2)、 send(2)、 socket(2)、 t_open(3)、 t_optmgmt(3)、 inet(7F)、 socket(7)。

RFC 768 用户数据报协议
RFC 1122 Internet 主机要求

名称

UNIX - 本地通信域协议

概要

#include <sys/types.h>

#include <sys/un.h>

说明

本地通信域协议，行业中通常称为 **Unix** 域协议，使用路径名地址格式和 **AF_UNIX** 地址系列。该协议可用作用于在同一节点上执行的进程之间进行通信的 Internet 协议系列（TCP/IP 或 UDP/IP ）的替代。与本地 IP 环回相比，它具有显著的吞吐量优势，这主要是由于它的代码执行开销非常低。数据在协议层（OSI 第 4 层）而不是在驱动程序层（OSI 第 2 层）环回。

AF_UNIX 地址系列仅支持 **SOCK_STREAM** 。

本地通信域协议的 HP-UX 实现在 **recv()** （请参阅 *recv*(2) ）和 **send()** （请参阅 *send*(2) ）中不支持 **MSG_OOB** 标志。

地址

AF_UNIX 套接字地址为路径名。其长度不超过 92 字节，包括一个终止空字节。对 **AF_UNIX** 套接字调用 **bind()** 将使用名为 **struct**sockaddr_un 的寻址结构（请参阅 *bind*(2) ）。只要系统调用需要指向 **struct sockaddr** 的指针，就应该在所有 **AF_UNIX** 套接字系统调用中使用指向该结构的指针。

包含文件 **<sys/un.h>** 定义了该寻址结构。该结构中存在两个值得注意的字段。第一个字段是 *sun_family* ，必须将其设置为 **AF_UNIX** 。第二个是 *sun_path* ，它是以空字符结尾的字符串，指定了与套接字（例如 **/tmp/mysocket**）相关联的文件的路径名。

只有被动（监听）套接字才必须绑定到一个地址。主动套接字连接到该地址，但它不需要其自己的地址。

有关使用 **AF_UNIX** 套接字进行进程间通信的详细信息，请参考《BSD Sockets Interface Programmer's Guide》。

套接字缓冲区大小

对于数据流套接字和数据报套接字，最大发送和接收缓冲区大小为 262142 字节。缺省缓冲区大小为 32768 字节。发送和接收缓冲区大小可通过使用 **setsockopt()** 系统调用的 **SO_SNDBUF** 和 **SO_RCVBUF** 选项进行更改。有关详细信息，请参考 *getsockopt*(2) 。

作者

AF_UNIX 由加州大学伯克利分校开发。

另请参阅

getsockopt(2)、 socket(2)。

名称

VLAN - 虚拟局域网

说明

本联机帮助页简要概述了 VLAN（虚拟 LAN）技术。

VLAN 是可以跨越多个物理网段的逻辑网段或 “虚拟”网段。 VLAN 的主要优势在于它们可以通过确定哪些目标应该接收流量，将广播与组播流量分隔开来，从而更好地利用交换机和端站资源。

通过使用 VLAN 进行逻辑分隔，可对 PC、服务器和其他网络资源进行分组，使其工作方式与连接到同一物理段时相同（即使未连接到同一物理段，也是如此）。

HP-UX VLAN 是 IEEE 802.1p/Q 标准的实现。

可使用 **nwmgr** 命令（请参阅 *nwmgr_vlan*(1M)）或 **lanadmin** 命令（请参阅 *lanadmin_vlan*(1M)）在 HP-UX 服务器中配置 VLAN 接口。对于 HP-UX 11i V3 及以后的版本，HP 建议您使用 **nwmgr** 。还可使用基于 Web 的管理工具 HP-UX System Management Homepage (HP SMH) 来配置接口。

每个 VLAN 接口都被赋予了系统上唯一的 VLAN PPA (VPPA)，以及一个 VLAN ID，该 VLAN ID 可识别其所属的虚拟 LAN。在创建 VLAN 接口的接口上，该 VLAN ID 是唯一的。

警告

不推荐使用 **lanadmin** 、 **lanscan** 和 **linkloop** 命令。在将来的 HP-UX 版本中将删除这些命令。 HP 建议使用替换命令 *nwmgr*(1M) 来执行所有与网络接口相关的任务。

另请参阅

lanadmin(1M)、 lanadmin_vlan(1M)、 lanscan(1M)、 nwmgr(1M)、 nwmgr_vlan(1M)、 smh(1M)。

《HP-UX VLAN Administrator’s Guide》

IEEE 802.1p， IEEE 802.1Q

名称

xopen_networking - X/Open 网络接口

说明

X/Open 已在“《X/Open CAE 规范》，网络服务，第 4 期 (UNIX 95)”、“《X/Open CAE 规范》，网络服务，第 5 期 (UNIX 98)”和“《Single UNIX 规范》，第 3 版，系统接口 (UNIX 03)”中定义了“套接字”和“**IP** 地址解析”接口。

X/Open 还在“《X/Open CAE 规范》，网络服务，第 4 期 (UNIX 95)”和“《X/Open CAE 规范》，网络服务，第 5 期 (UNIX 98)”中定义了 XTI。从 UNIX 03 开始，XTI 不再是《Single UNIX 规范》的一部分。

有关规范的详细信息或 X/Open 网络接口的详细说明，请参考位于 **The Open Group** 网站 **http://www.opengroup.org** 的上述规范。

在 HP-UX 11i v3 之前，HP-UX 在 PA-RISC 和 Integrity 系统上被认证为 UNIX 95。从 HP-UX 11i v3 开始，HP-UX 在 PA-RISC 系统上被认证为 UNIX 95，在 Integrity 系统上被认证为 UNIX 95 和 UNIX 03。

编译环境

可通过两种方法来获得 X/Open 套接字功能：

方法 A 符合 X/Open 编译规范。

方法 B 与 X/Open 编译规范略有不同。但是，方法 B 允许程序既包含编译为 X/Open 套接字规范的对象，又包含编译为 BSD 套接字规范的对象。

可使用 **cc** 、 **c89** 或 **c99** 实用程序。有关详细信息，请参阅 *cc*(1) 。另请注意， **UNIX 03** 中的某些功能仅在使用了 **c99** 的情况下才可用。例如，用于指针的“restrict”限定符仅在使用了 c99 的情况下才可用。

方法 **A)** 严格相符方法

符合 X/Open 要求的应用程序是指，其所有部分都是根据 X/Open 规范进行编译和创建的应用程序。对于此类符合要求的应用程序，此编译方法是适当的。

编译

UNIX 03

应用程序应确保使用值 **600** 定义了功能测试宏 **_XOPEN_SOURCE** 。为了确保可移植性，应用程序应在包含任何头文件之前，在编译命令行中或在每个源模块的开头定义该宏。

例如，使用 **HP ANSI Compiler** 编译一个 64 位对象：

c99 +DD64 -D_XOPEN_SOURCE=600 -c main.c -o main.o

c99 +DD64 -D_XOPEN_SOURCE=600 -c routines.c -o routines.o

UNIX 95

应用程序应确保定义了功能测试宏 **_XOPEN_SOURCE** 和 **_XOPEN_SOURCE_EXTENDED** 。为了确保可移植性，应用程序应在包含任何头文件之前，在编译命令行中或在每个源模块的开头定义该宏。

例如，使用 **HP ANSI Compiler** 编译一个 64 位对象：

c89 +DD64 -D_XOPEN_SOURCE -D_XOPEN_SOURCE_EXTENDED -c main.c -o main.o

c89 +DD64 -D_XOPEN_SOURCE -D_XOPEN_SOURCE_EXTENDED -c routines.c -o routines.o

链接

使用 **Xnet** 库链接程序对象。

例如:

ld main.o routines.o -lxnet -lc -o prog

请注意，如果还在链接行中指定了 **C** 库，则必须在 **C** 库之前指定 **Xnet** 库。否则，X/Open 套接字调用将会解析为 **C** 库中的 BSD 套接字函数，而不是 **Xnet** 库中的 X/Open 套接字函数。

方法 **B)** 可选方法

HP-UX 提供两种形式的套接字 API:

- 缺省的 BSD 套接字
- X/Open 套接字

这两种形式的套接字 API 具有相同的函数名称，但是它们在语义和参数类型上有所不同。例如，X/Open **getsockopt()** 中的 *optlen* 字段为 **size_t** 类型，而 BSD **getsockopt()** 为 **int** 类型。在 64 位模式下， **size_t** 为 64 位，而 int 仍然为 32 位。

在同一个程序中，使用方法 A 中的方法链接编译为 X/Open 套接字规范的对象和编译为 BSD 套接字规范的对象会导致错误地将 BSD 套接字调用解析为 **Xnet** 库中的 X/Open 套接字函数。因此，当程序运行时，它可能会导致应用程序核心转储或异常的套接字错误。通常，在调用 BSD 套接字函数 **accept()** 、 **getpeername()** 、 **getsockname()** 、 **getsockopt()** 、 **recvfrom()** 、 **sendmsg()** 和 **recvmsg()** 时会产生此问题。

对于此类混合程序配置，应使用下面“编译”一节中说明的编译和链接方法。

编译

除在 **UNIX 03** 中定义 **_XOPEN_SOURCE=600** 以及在 **UNIX 95** 中定义 **_XOPEN_SOURCE** 和 **_XOPEN_SOURCE_EXTENDED** 外，还应定义 **_HPUX_ALT_XOPEN_SOCKET_API** 。

例如，使用 **HP ANSI Compiler** 编译一个 64 位 X/Open 套接字对象和一个 64 位 BSD 套接字对象:

UNIX 03

c99 +DD64 -D_XOPEN_SOURCE=600 -D_HPUX_ALT_XOPEN_SOCKET_API -c main.c -o main.o

UNIX 95

c89 +DD64 -D_XOPEN_SOURCE -D_XOPEN_SOURCE_EXTENDED -D_HPUX_ALT_XOPEN_SOCKET_API -c main.c -o main.o

BSD Sockets

cc -Ae +DD64 -c routines.c -o routines.o

使用此方法，X/Open 套接字调用会被 <sys/socket.h> 中的静态套接字函数重新映射到 C 库中的 X/Open 套接字函数的可选集。此可选集在其函数名称中具有前缀 **_xpg_**，例如，**_xpg_getsockopt()**。

因为此可选集具有不同的函数名称，所以在链接时不会混淆 X/Open 套接字调用和 BSD 套接字调用。

除命名差异外，此可选集与“Xnet”库中的 X/Open 套接字函数相同。除添加一个附加宏 **_HPUX_ALT_XOPEN_SOCKET_API** 外，此编译方法符合 X/Open 规范。

链接

使用 C 库而不是“Xnet”库链接。不应将“Xnet”库包含在应用程序链接行中。

例如：

ld main.o routines.o -lc -o prog

因为链接行中未包含“Xnet”库，所以 BSD 套接字调用不会被错误地解析为“Xnet”库中的 X/Open 套接字函数。

未来方向

在将来的发行版中，方法 B 可能会成为缺省的方法。届时，缺省情况下会定义 **_HPUX_ALT_XOPEN_SOCKET_API**。

作者

X/Open **XTI**、套接字和 **IP** 地址解析接口由 HP 和 X/Open Company, Ltd. 联合开发。

另请参阅

XTI：

t_accept(3)、t_alloc(3)、t_bind(3)、t_close(3)、t_connect(3)、t_error(3)、t_free(3)、t_getinfo(3)、t_getprotaddr(3)、t_getstate(3)、t_listen(3)、t_look(3)、t_open(3)、t_optmgmt(3)、t_rcv(3)、t_rcvconnect(3)、t_rcvdis(3)、t_rcvrel(3)、t_rcvudata(3)、t_rcvuderr(3)、t_snd(3)、t_snddis(3)、t_sndrel(3)、t_sndudata(3)、t_strerror(3)、t_sync(3) 和 t_unbind(3)。

套接字：

accept(2)、bind(2)、close(2)、connect(2)、fcntl(2)、fgetpos(3S)、fsetpos(3S)、ftell(3S)、getpeername(2)、getsockname(2)、getsockopt(2)、listen(2)、lseek(2)、poll(2)、read(1)、recv(2)、recvfrom(2)、recvmsg(2)、select(2)、send(2)、sendmsg(2)、sendto(2)、setsockopt(2)、shutdown(2)、sockatmark(3N) socket(2)、socketpair(2) 和 write(1)。

IP 地址解析：

gethostname(2)、endhostent(3N)、endnetent(3N)、endprotoent(3N)、endservent(3N)、freeaddrinfo(3N)、gai_strerror(3N)、getaddrinfo(3N)、gethostbyaddr(3N)、getnameinfo(3N)、getnetbyaddr(3N)、getprotobynumber(3N)、getservbyport(3N)、htonl(3N)、if_freenameindex(3N)、if_indextoname(3N)、if_nameindex(3N)、if_nametoindex(3N)、inet_addr(3N)、ntohl(3N)、sethostent(3N)、setnetent(3N)、setprotoent(3N)、setservent(3N)。

名称

zero - **/dev/zero** 专用文件

说明

/dev/zero 是一个零专用文件。从零专用文件读取将始终返回值为 0（\0 字符）的字符。

在零专用文件上写入的数据将被丢弃或忽略。

在零专用文件上搜索将始终会成功。

当 **/dev/zero** 是通过调用 **mmap()** 映射的内存时，关联的内存对象的行为如同 **MAP_ANONYMOUS** 对象的行为。它将全部被初始化为零。写入对象的操作将修改后续读取操作所读取的该对象的内容。

MAP_SHARED 和 **MAP_PRIVATE mmap()** 都是允许的。

当它为映射为共享时，内存对象仅可供当前进程的派生进程共享。对 **MAP_SHARED** 对象所做的修改仅对于该进程及其派生进程可见。

当它为映射为专用时，在 **fork()** 之后的任何修改仅对进程可见。

举例

在下例中，将使用 **len** \0 字符填充缓冲区 **buf** 。

```
fildes = open("/dev/zero",...)
read(fildes, buf, len)
```

下例中，进程此时在内存位置 **address** 处占据 **len** \0 个字符的范围：

```
fildes = open("/dev/zero",...)
address = mmap(0, len, PROT_READ | PROT_WRITE,
    MAP_PRIVATE, fildes, any_offset)
```

文件

/dev/zero

另请参阅

mmap(2)、 null(7)。

第 9 节

简介和词汇表

第 9 节

简介和词汇表

名称

intro - HP-UX 一般信息部分简介

说明

本节包含有关 HP-UX 的一般信息，包括 HP-UX 和操作系统的简介以及常见 HP-UX 术语表。

另请参阅

glossary(9)、 introduction(9)。

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

名称

glossary - 常用 HP-UX 术语说明

说明

HP-UX 和其他 UNIX 类系统使用专用词汇表，其中某些单词和术语具有非常特殊的含义。本词汇表旨在帮助您准确使用这些专门术语，有时候这些术语在其他环境下可能会有不同的含义。根据需要，本词汇表中还包含对其他 HP-UX 文档的引用。

字体为 斜体且带有括在括号中的罗马数字（有时还带有大写字母）的实体 - 例如 *sh*(1) 、 *wait*(2) 或 *fopen*(3S) ，表示本手册其他各节中的条目。 粗体项目表示本词汇表中的其他条目。使用 计算机系统字体的项目（在联机帮助页中为 粗体）表示文字项，如文件名和环境变量。所有斜体形式的手册名表示系统附带的或单独提供的独立手册。

虽然某些术语和定义也融合了 IEEE POSIX 标准和《X/Open Portability Guide》中的术语和定义，但本版本中的定义仍然是专门针对 HP-UX 操作系统。本词汇表力求用语精确，以便更好地反映 HP-UX 系统的特性。

词汇表条目

. （点）

表示 当前目录的专用文件名。该文件名可以单独使用，也可以用在目录路径名的开头。另请参阅 路径名解析。在 POSIX、Bourne 和 Korn Shell 中 点也作为专用命令，在文本编辑器和格式化程序中解析正则表达式和指定文件名时具有特殊含义。

.. （点-点）

表示 父目录的专用文件名。如果 点-点位于一个 路径名的开头，则它表示当前目录的父目录。如果 点-点出现在路径名中，则表示路径名字符串中位于 点-点之前的目录的父目录。一种特殊情况是，在没有父目录的任何目录（通常为 根目录）中， 点-点表示当前目录。另请参阅 路径名解析。

.o （点-**o**）

通常指定给浮动对象文件的后缀。术语 点-**o** 文件有时用于表示浮动对象文件。这类文件的格式有时称为 点-**o** 格式。请参阅 *a.out*(4) 。

a.out

通常是为 HP-UX 上的可执行对象代码文件指定的名称。其格式取决于不同的计算机，在 *a.out*(4) 中对每种实现进行了描述。尚未链接的对象代码具有相同格式，但称为 **.o** （点-**o** ）文件。 **a.out** 也是链接程序 *ld*(1) 使用的缺省输出文件名。

Agile 寻址

一种寻址方案，其中逻辑单元的地址或路径独立于物理路径。有关详细信息，请参阅 *intro*(7) 。

ASCII

美国信息交换标准委员会 (American Standard Code for Information Interchange) 的首字母缩略词。 ASCII 是传统的 System V 编码字符集，定义了包括控制字符和图形字符在内的 128 个字符，其中每个字符由 7 位二进制值表示（其十进制值的范围为 0 到 127）。

保留的进程组 **ID**

每个进程都拥有保留的进程组 ID，它保留自上一次成功调用 *exec*(2) 而生成的进程组 ID。请参阅 *setpgrp*(2)、*termio*(7) 和 进程组 **ID** 。

保留的设置用户 **ID**

请参阅 保留的用户 **ID** 。

保留的设置组 **ID**

请参阅 保留的组 **ID** 。

保留的用户 **ID**

每个进程都拥有 保留的用户 **ID** ，它保留自上次成功执行 *exec*(2) 或 *setresuid*(2) 调用，或者上次对 *setuid*(2) 执行超级用户调用而生成的进程 有效用户 **ID** 。 *setuid*(2) 允许进程将其有效用户 ID 设置为此记忆值。因此，对于要执行程序的进程，如果已设置了其设置用户 ID 位且所有者 ID 为 5（举例），则该进程可在程序终止之前随时将其有效用户 ID 设置为 5。请参阅 *exec*(2)、 *setuid*(2)、 保留的组 **ID** 、 有效用户 **ID** 和 设置用户 **ID** 位。保留的用户 ID 也称为 保留的设置用户 **ID** 。

保留的组 **ID**

每个进程都拥有保留的组 ID，它保留自上次成功执行 *exec*(2) 或 **setresgid()** 调用（请参阅 *setresuid*(2)），或者上次对 **setgid()** （请参阅 *setuid*(2) 或 *setresuid*(2)）执行超级用户调用而生成的进程 有效组 **ID** 。 **setgid()** 允许进程将其有效组 ID 设置为此记忆值。因此，对于要执行程序的进程，如果已设置了其设置组 ID 位且组 ID 为 5（举例），则该进程可以在程序终止之前随时将其有效组 ID 设置为 5。请参阅 *exec*(2)、 *setuid*(2)、 保留的用户 **ID**、 有效组 **ID** 和 设置组 **ID** 位。保留的组 ID 也称为 保留的设置组 **ID** 。

备份

创建全部或部分文件系统的副本，以便在系统崩溃（通常由电源故障、硬件问题等原因造成）的情况下保留文件系统的过程。强烈建议执行备份操作。

本地惯例

是指按地理地区或地域而产生的一种约定，例如日期、时间和货币形式等的不同。

本地化

调整现有软件以满足特定地区的本地语言、惯例和字符集要求的过程。

本地语言

计算机用户的口头或书面语言，例如，中文、荷兰语、英语、法语、德语、希腊语、意大利语、片假名、朝鲜语、西班牙语、瑞典语、土耳其语等。

本地语言支持 **(NLS)**

HP-UX 的一种功能，为用户提供国际化软件，并为应用程序编程人员提供开发这种软件的工具。

崩溃

程序或系统的意外关闭。如果操作系统发生了崩溃，则为“系统崩溃”，此时需要重新引导系统。

编码字符集

一组意义明确的规则，可建立字符集以及字符集中每个字符与其对应的位表示之间的一一对应关系。 **ASCII** 就是一个 编码字符集。

标准错误

程序发出错误和特殊消息的目标，用于诊断消息。标准错误输出通常表示为 **stderr** ，对于每个调用的命令，将自动打开该输出以写入文件描述符 2。在缺省情况下，用户终端是写入标准错误的所有数据的目标，但也可以将其重定向到其他目标。与永远不会用于向“错误”方向传输数据的标准输入和标准输出不同，标准错误只是偶尔被读取。建议不要执行该操作，因为 I/O 重定向功能可能会中断某个程序来执行它。

标准输出

程序的输出数据的目标。标准输出文件通常称为 **stdout** ，对于每个调用的命令，将自动打开该文件以写入文件描述符 1。在缺省情况下，用户的终端是写入标准输出的所有数据的目标，但可以将其重定向到其他目标。

标准输入

程序的输入数据的来源。标准输入文件通常称为 **stdin** ，对于调用的每个命令，将自动打开该文件来读取文件描述符 0。在缺省情况下，用户的终端是从标准输入读取的所有数据的源，但可以从另一个源对其进行重定向。

补充组 **ID**

除有效组 ID 外，进程还拥有最多 **sysconf(_SC_NGROUPS_MAX)** 用于确定文件访问权限的补充组 ID。在创建一个进程后，该进程的补充组 ID 会设置为其父进程的补充组 ID。请注意， **sysconf(_SC_NGROUPS_MAX)** 返回的值可能大于某些 HP-UX 系统上的 **<limits.h>** 中的 **NGROUPS_MAX** 的值。

CS/80、CS-80

通过名为 **CS/80** （命令集 1980）命令集的一系列命令和数据传输协议，与控制计算机通信的一系列海量存储设备。实现此命令集是为了随着技术发展，在不同型号和不同版本的海量存储设备之间提供更好的向前/向后兼容性。某些海量存储设备仅支持整个 **CS/80** 命令集的子集，通常表示为 **SS/80** （子集 1980）设备。

常规文件

一种 文件类型，其仅包含可随机访问的字节序列，不具有系统强制规定的其他结构。可以扩展其大小。常规文件也称为 普通文件。

超级块

位于每个文件系统海量存储介质上的描述该文件系统的一个块。对于不同的实现，超级块的内容也不同。有关详细信息，请参考您系统附带的系统管理员手册。

超级目录

请参阅 父目录。

超级用户

HP-UX 系统管理员。此用户有权访问所有文件并可执行权限操作。 超级用户的 实际用户 **ID** 和 有效用户 **ID** 是 0；依照惯例，其用户名是 **root** 。

程序

以二进制代码形式向计算机发出的指令序列（由程序源代码的编译和汇编过程生成）。

持久性设备专用文件

海量存储设备的设备文件，与 LUN 硬件路径关联，因此可以透明地支持 **Agile** 寻址和多路径。也就是说，在对 LUN 进行以下操作时持久性设备专用文件都不会发生更改：从一个主机总线适配器 (HBA) 移动至另一个 HBA，从一个 switch/hub 端口移动至另一个端口，通过不同目标端口提供给主机，或使用多个硬件路径进行配置。有关设备专用文件的详细信息，请参阅 *intro*(7) 。

次设备号

是表示专用文件的一种属性的数字，次设备号是在文件创建期间制定，并在访问文件时使用它们来启用到（或来自）特定设备的 I/O 通道。该数字被传递给设备驱动程序，用于从一系列设备中选择要使用的设备，以及可能的某些操作模式。 次设备号的具体形式和含义取决于使用的驱动程序和寻址格式（Legacy 格式或 Agile 格式）。在 Legacy 格式中，次设备号编码路径信息，但在 Agile 格式中，次设备号不透明且基于 WWID。

delta 版

源代码控制系统 (SCCS) 中使用的术语，用于描述对 **SCCS** 文件所做的一处或多处文本更改量。每次编辑 SCCS 文件时，对该文件所做的更改都会单独存储为一个 **delta** 版。然后，会使用 *get*(1) 命令来指定要对 SCCS 文件应用或从该文件中排除哪些 delta 版，从而生成该文件的特定版本。 **vi** 或 **ed** 编辑器与此相反，它们会立即将更改合并到文件中，所以不可能获得该文件的以前版本。 RCS 文件提供了类似功能（请参阅 *rcsintro*(5) ）。

打开文件

当前与文件描述符关联的文件。

大写**-**小写转换

大写字符至其小写形式的转换。

单用户状态

HP-UX 操作系统的一种状态，这种情况下系统控制台仅在系统与其用户之间的提供通信机制。依照惯例，单用户状态通常由 *init*(1M) 指定为具有运行级别 **S** 或 **s** 。不要将 单用户状态与单用户系统相混淆，在单用户状态下，软件会将多用户系统限制为单用户通信；而单用户系统永远不能与多个固定终端进行通信。另请参阅 多用户状态。

当前工作目录

请参阅 工作目录。

当前目录

请参阅 工作目录。

登录

获得对 HP-UX 的访问的过程。登录过程包括成功执行由 *login*(1) 定义的登录序列，这一过程因系统的不同配置而异。登录要求提供 登录名以及（可能）一个或多个口令。

登录目录

您登录后立即进入的目录。文件 **/etc/passwd** 中为每个用户定义了此目录。在您登录后， *login*(1) 立即自动将 Shell 变量 **HOME** 设置为您的 登录目录。请参阅 主目录。

地址

在信息存储或检索过程中，用于指定和标识内存位置的数字。 地址用于标记、定向和指明目标，并指示计算机元素或者与计算机元素通信。

在邮件中， 地址是一种数据结构，传输信息时所涉及的所有元素都可以识别其格式。在本地系统上，地址可能仅仅是用户的 登录名，而在联网的系统中， 地址指定的是网络软件资源的位置。

在文本编辑器（例如， **vi** 、 **ex** 、 **ed** 或 **sed** ）中， 地址会在文件中定位到给定指令将要处理的那一行。

对于 **adb** ， 地址指定使用哪一种汇编语言指令来执行给定命令。

在 **fsdb** 等磁盘实用程序中， 地址可能表示原始文件或 块设备专用文件、 **i** 节点编号、 卷标题或其他文件属性。

在外围设备环境中， 地址表示为计算机指定 I/O 设备位置的一组值。在不同系统上组成地址的具体内容也不同。

地址空间

进程可引用的内存位置的范围。

点

请参阅 **.**（点）。

点**-o**

请参阅 **.o** （点**-o** ）。

点**-o** 格式

请参阅 **.o** （点**-o** ）。

点**-o** 文件

请参阅 **.o** （点**-o** ）。

点**-**点

请参阅 **..** （点**-**点）。

动态加载程序

进程启动时调用的例行程序，可将共享库加载到进程的地址空间中。动态加载程序还可解析程序与共享库之间的符号引用，并初始化共享库的链接表。有关详细信息，请参阅 *dld.sl*(5) （PA-RISC 系统）或 *dld.so*(5) （基于 Itanium(R) 的系统）。

多路复用器 **(MUX)**

多路复用器 (MUX) 是一种高速串行通信多端口产品。它可合并不同信号，以便在单个通道上传输，并为卸载的 CPU 串行通信处理任务提供智能通信功能。

多用户状态

HP-UX 操作系统的状态，在该状态下终端（包括系统控制台）允许系统与其多个用户之间进行通信。依照惯例，多用户运行级别设置为状态 2，该状态通常定义为包含多用户环境中需要的所有终端进程和 守护程序。运行级别是表驱动式的，由 *init*(1M) 指定，它通过查看文件 **/etc/inittab** 设置运行级别。不要将多用户系统与多用户状态相混淆。多用户系统是指系统在处于多用户状态时，可以拥有多个主动与该系统通信的用户。多用户状态可以消除

单用户状态强加的单用户限制（请参阅 单用户状态和 *inittab*(4) ）。

EOF

请参阅 文件结束标记。

二级提示符

Shell 在显示屏上显示的一个或多个字符，指示需要更多输入。遇到此提示符的频率比遇到 Shell 的主提示符（请参阅 提示符）的频率低。出现二级提示符通常是由于省略了字符串的右引号（使 Shell 无所适从），或者是由于您在命令行上输入了 Shell 编程语言控制流结构（例如， **for** 结构）。在缺省情况下，Shell 的二级提示符是大于符号 (**>**) ，不过您可以通过在 **.profile** 文件中相应地设置 Shell 变量 **PS2** 来重新定义它（C Shell 没有二级提示符）。

FIFO 设备专用文件

一种 文件类型。写入 **FIFO** 的数据会按照先进先出的原则进行读取。 *open*(2) 、 *read*(2) 、 *write*(2) 和 *lseek*(2) 中介绍了其他特性。

fork

一种 HP-UX 系统调用（请参阅 *fork*(2) ），当现有进程调用该系统调用时会创建新进程。该新建进程称为 子进程；现有进程则称为 父进程。子进程是通过创建与父进程完全相同的副本创建的。父进程和子进程能够通过其相应 **fork** 调用的返回值（有关详细信息，请参阅 *fork*(2) ）来识别自身。

访问

从存储器获得数据或向存储器放置数据的过程，或使用系统资源的权利。可访问性由三种进程特性来管理：有效用户 ID、有效组 ID 和组访问列表。 *access*(2) 系统调用根据文件的 *amode* 参数中包含的位模式来确定文件的可访问性，构建该参数可判断文件的读取、写入、执行权限或检查该文件是否存在。 *access*(2) 系统调用使用 实际用户 **ID** 而不是 有效用户 **ID** ，并使用 实际组 **ID** 而非 有效组 **ID** 。

访问模式

访问模式是指允许的文件访问形式。每个实现提供独立的读取、写入和执行/搜索访问模式。

访问组

组访问列表是一组用于确定资源可访问性的 补充组 **ID** 。下文中的 文件访问权限中将说明如何执行访问检查。

非占位字符

变音符或重音等字符，用于与其他字符一起组成非英语语言中常用的复合图形符号。

分层目录

目录（或文件系统）的结构，其中每个目录可以包含文件和其他目录。

符号链接

一种间接引用路径名的文件类型。请参阅 *symlink*(4) 。

父进程

如果当前的现有进程创建了新进程（通过 *fork*(2) ），该现有进程就是新建进程的父进程。每个进程只有一个父进程（但 **init** 进程除外，请参阅 **init** ），但每个进程可通过 *fork*(2) 系统调用创建多个新进程。任何进程的父进程 ID 都是其创建者的 进程 **ID** 。

父进程 **ID**

新进程由当前活动的进程创建。在创建者的周期内，进程的 父进程 **ID** 就是其创建者的进程 ID。创建者的周期结束后， 父进程 **ID** 是 **init** 的进程 ID。

父目录

文件层次结构中某一目录的上一级目录。除 根目录 (*/*) 外，所有目录都有且仅有一个父目录。 根目录没有父目录。另请参阅 点和 点**-**点。

复合图形符号

在一个字符位置包含两个或多个其他图形符号组合的图形符号，例如一个变音符和一个基本字母的组合。

根卷

包含引导区（其中包含 HP-UX 内核）以及 HP-UX 文件系统 根目录的海量存储卷。

根目录

(1) 分层文件系统的最高级别目录，所有其他文件分支都位于该目录下。在 HP-UX 中， 斜线 (*/*) 字符表示 根目录。根目录是文件系统中唯一一个其 父目录是其自身的目录。

(2) 每个进程与一个根目录的概念相关联，以便对那些以 斜线 (*/*) 开头的路径实现解析其路径名搜索。进程的根目录不必一定是根文件系统的根目录，并且可以通过 *chroot*(1M) 命令或 *chroot*(2) 系统调用来更改该目录。这样的目录对于所关联的进程而言，其本身就是父目录。

工作目录

每个进程都与当前工作目录的概念相关联。对于 Shell，此目录即为您当前“驻留”的目录。相对路径名（即不是以 **/** 开头的路径名）将从此目录中开始搜索。该目录有时也称为 当前目录或 当前工作目录。

共享库

可由多个不同程序共享的可执行文件。共享库的代码并未通过 *ld*(1) 链接到程序中，而是在运行时由动态加载程序映射到进程的地址空间。共享库必须包含位置无关代码并由 *ld*(1) 创建。通常共享库文件名的后缀为 **.sl** 。

共享内存标识符 **(shmid)**

由 *shmget*(2) 系统调用创建的具有唯一性的正整数。每个 **shmid** 都有一个内存段（称为共享内存段）和与其关联的数据结构。该数据结构表示为 **shmid_ds** ，包含下列成员：

```
struct
ipc_perm  shm_perm;        /* operation permission struct */
size_t    shm_segsz;       /* size of segment */
pid_t     shm_cpid;        /* creator pid */
pid_t     shm_lpid;        /* pid of last operation */
shmatt_t  shm_nattch;      /* number of current attaches */
time_t    shm_atime;       /* last attach time */
time_t    shm_dtime;       /* last detach time */
time_t    shm_ctime;       /* last change time */
                           /* Times measured in secs since 00:00:00 GMT, Jan. 1, 1970 */
```

可以使用 *ftok*(3C) 创建共享库标识符。

shm_perm 是指定 *shmop*(2) 或 *shmctl*(2) 操作（请参阅下文）的权限的 **ipc_perm** 结构。该结构包括下列成员：

```
uid_t   cuid;    /* creator user id */
gid_t   cgid;    /* creator group id */
uid_t   uid;     /* user id */
gid_t   gid;     /* group id */
mode_t  mode;    /* r/w permission */
```

shm_segsz 指定共享内存段的大小。 **shm_cpid** 是创建共享内存标识符的进程的进程 ID。 **shm_lpid** 是上一次执行 *shmop*(2) 操作的进程的进程 ID。 **shm_nattch** 表示当前连接此段的进程的数量。 **shm_atime** 表示上一次执行 **shmat** 操作的时间， **shm_dtime** 表示上一次执行 **shmdt** 操作的时间，而 **shm_ctime** 表示上一次执行 *shmctl*(2) 操作（更改了上述结构成员之一）的时间。

共享内存操作权限

在 *shmop*(2) 和 *shmctl*(2) 系统调用说明中，已指出了每个操作所需的权限。特定进程是否具有对某对象执行 *shmop*(2) 或 *shmctl*(2) 操作的权限，由该对象的权限模式位决定，如下所示：

00400 由用户读取
00200 由用户写入
00060 由组读取和写入
00006 由其他用户读取和写入

如果下列一个或多个条件成立，则会授予进程对 共享内存标识符 (**shmid**) 的 *shmop*(2) 或 *shmctl*(2) 操作的读取或写入权限：

- 进程的有效用户 ID 是超级用户。
- 进程的有效用户 ID 与 **shmid** 的关联数据结构中的 **shm_perm.[c]uid** 匹配，且 **shm_perm.mode** 的“user”部分 (0600) 的相应位已设置。
- 进程的有效用户 ID 与 **shm_perm.[c]uid** 不匹配，而进程的有效组 ID 与 **shm_perm.[c]gid** 匹配，或者 **shm_perm.[c]gid** 之一包含在进程的组访问列表中，且 **shm_perm.mode** 的“group”部分 (00060) 的相应位已设置。
- 进程的有效用户 ID 与 **shm_perm.[c]uid** 不匹配，且进程的有效组 ID 与 **shm_perm.[c]gid** 不匹配，并且任一 **shm_perm.[c]gid** 都不包含在进程的组访问列表中，且 **shm_perm.mode** 的“other”部分 (06) 的相应位已设置。

否则，将拒绝相应的权限。

孤进程

当 父进程因任何原因终止时遗留的 子进程。 **init** 进程（请参阅 *init*(1M) ）将接收所有孤进程（即成为其有效父进程）。

孤进程组

一种进程组，其中每个成员的父进程本身或者是该组的成员，或者不是该组的会话的成员。

关联

请参阅 终端关联。

管道

用于在两个进程间传递数据的进程间 I/O 通道。 **Shell** 通常使用管道将数据从一个进程的标准输出传输到另一个进程的标准输入。在命令行中，管道用竖线 (|) 表示。竖线左侧命令的输出直接作为其右侧命令的标准输入。

归档

由其他文件 - 例如链接程序 *ld*(1) 使用的一组对象文件（即 **.o** ）的内容组成的文件。归档文件由 *ar*(1) 或者类似程序（例如 *tar*(1) 或 *cpio*(1) ）创建和维护。 归档通常称为 库。

国际标准时间 **(UTC)**

请参阅 起始参考时间。

国际化

使软件能够支持用户的 本国语言、 本地惯例和 编码字符集的概念。

过滤

从标准输入读取数据、对数据执行转换并将数据写入标准输出的命令。

行

包含零个或多个非换行符以及终止换行符的文本字符序列。

后台进程组

如果一个会话已经与不属于前台进程组的控制终端建立了连接，则该会话的成员即为后台进程组。

环境

定义的 Shell 变量（如 **EXINIT** 、 **HOME** 、 **PATH** 、 **SHELL** 、 **TERM** 和其他变量）集，用于定义用户命令运行时所处的环境。这些环境可能包括终端特性、主目录和缺省搜索路径。当前进程中的每个 Shell 变量设置会传递给创建的所有 子进程，前提是已通过 **export** 命令（请参阅 *sh*(1) ）导出了每个 Shell 变量设置。未导出的 Shell 变量设置仅对当前进程有意义，并且通过执行 **/etc/profile** 、 **$HOME/.profile** 或 **$HOME/.login** ，创建的任何子进程都将获得某些 Shell 变量的缺省设置。

幻数

a.out 格式文件或归档文件的第一个单词。此单词包含系统 ID，说明文件将在哪台计算机（硬件）上运行，还说明文件的类型（可执行文件、共享的可执行文件、归档文件等）。

换行符

其 ASCII 值为 10（换行）的字符，用于断开各行字符。在 C 语言和各种实用程序中，换行符由 **\n** 表示。终端驱动程序通常将终端发送的回车/换行序列解释为单个换行符（但有关完整的详细信息，请参阅 *tty*(7) ）。

会话

每个进程组都是会话的一个成员。进程被视为其进程组所属的会话的成员。新建的进程会加入其创建者所属的会话。进程可以更改其会话成员关系（请参阅 *setsid*(2) ）。一个会话可以包含多个进程组（请参阅 *setpgid*(2) ）。

会话发起者

创建了会话的进程（请参阅 *setsid*(2) ）。

会话周期

从创建会话开始，到一直作为会话成员的所有进程组的周期结束时为止的一段时间。

i 节点

i 节点是描述文件的一种结构，在系统中由 文件序号标识。每个文件或目录都具有相关联的 **i** 节点。指定可访问文件的用户及访问方式的权限保留在一个 9 位字段中，该字段是 **i** 节点的组成部分。 **i** 节点还包含文件大小、文件的用户 ID 和组 ID、链接数，以及指向文件内容所在磁盘块的指针。 **i** 节点与其在一个或多个目录中的条目之间的每个连接称为一个 链接。

i 节点编号

请参阅 文件序号。

I/O 重定向

HP-UX Shell 提供的一种机制，用于更改标准输入的数据源和/或标准输出与标准错误的数据目标。请参阅 *sh*(1)。

init

执行初始化的 系统进程，是系统中所有其他进程的祖先进程，用于启动 登录进程。 **init** 的 进程 **ID** 通常为 **1** 。请参阅 *init*(1M) 。

ITE

请参阅 内置仿真终端。

僵尸进程

对由于任何原因所终止、而其父进程还未执行等待它终止（通过 *wait*(2) ）的操作的进程所指定的名称。终止的进程会继续占用进程表中的插槽，直到其父进程等待它。但由于该进程已终止，所以在用户空间或内核空间中都不会为其分配其他空间。因此，这种情况产生的危害并不大，并且下次其父进程执行等待时，僵尸进程会自行恢复。 *ps*(1) 命令将僵尸进程标记为 **defunct** 。

交错比

确定海量存储介质上各扇区的访问顺序的数字。可对其进行优化，以提高获取数据的效率。

节点名

本地网络中可唯一标识系统的字节字符串。与 主机名不同的是，节点名不能包含域名。可通过 *uname*(1) 命令对其进行查看和（或）设置。由于应用程序有时可互换使用节点名和主机名，因此，通常将节点名和主机名设置为相同值。

进程

程序的调用或映像的执行（请参阅 映像 ）。虽然所有命令和实用程序都在进程内执行，但并非所有命令或应用程序都与进程一一对应。某些命令（例如 **cd** ）在进程内执行，但并不创建任何新进程。其他命令（例如 **ls | wc -l**）

则会创建多个进程。一个程序可同时运行多个进程，但每个进程可能使用不同数据且处于不同执行阶段。也可以将进程视作 地址空间和在该地址空间内执行的单个控制线程及其所需的系统资源。 进程由另一个进程通过执行 *fork*(2) 函数调用来创建。执行 *fork*(2) 的进程称为 父进程，而 *fork*(2) 创建的新进程称为 子进程。

进程 **1**

请参阅 **init** 。

进程 **ID**

系统中的每个活动进程在其生命周期内由小于或等于 **PID_MAX** 的正整数（称为 进程 **ID** ）唯一标识。在进程生命周期结束之前，系统不能重用其进程 ID。此外，如果存在进程组 ID 与该进程 ID 相同的进程组，则在该进程组的生命周期结束之前，系统不能重用该进程 ID。

进程周期

在使用 *fork*(2) 函数创建一个进程后，该进程被认为是活动进程。在该进程终止前，其控制线程和 地址空间始终存在。然后，进程进入不活动状态，此时某些资源可能会返回给系统，但是 进程 **ID** 等资源仍被使用。当另一进程对非活动进程执行 **wait()** 、 **wait3()** 或 **waitpid()** 函数（请参阅 *wait*(2) ）时，剩余资源将返回给系统。最后将返回给系统的资源是进程 ID。此时，进程周期结束。

进程组

系统中的每个进程都是 进程组的成员。这种分组可表明相关的进程。新建的进程会加入其创建者所属的进程组。

进程组 **ID**

系统中的每个进程组在其周期内由 进程组 **ID** – 即小于或等于 **PIC_MAX** 的正整数唯一标识。在进程组的周期结束之前，系统不能重新使用其 进程组 **ID** 。

进程组创建者

进程组创建者是指其进程 ID 与其进程组 ID 相同的进程。

进程组周期

从创建 进程组开始，到组中剩余的最后一个进程离开该组（由于进程终止或调用了 *setsid*(2) 或 *setpgid*(2) 函数）为止的一段时间。

卷编号

用于设备的地址的组成部分。该数字的含义与软件和设备相关，但常用于指定多卷磁盘驱动器上的特定卷。有关详细信息，请参阅您系统附带的系统管理员手册。

绝对路径名

以斜线 (*/*) 开头的路径名。这种路径名表示相对于 根目录 (*/*) ，的文件位置，且从根目录开始搜索。

可挂接的文件系统

某些海量存储介质上包含的可卸除的块文件系统，具有自己的根目录以及独立的目录和文件层次结构。请参阅 块设备专用文件和 *mount*(1M) 。

可移植的文件名称字符集

下列图形字符集可以在符合 IEEE P1003.1 标准的实现之间移植：

ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
01234567890._-

最后三个字符分别是点、下划线和连字符。不能将连字符用作可移植文件名的第一个字符。

空白字符

一个或多个字符，显示时会使光标或打印头移动，但不会导致显示任何可见图形。 ASCII 代码集中的空白字符包括空格、制表符、换页符、回车符和纵向制表符。特定命令或例行程序可能将部分（但不一定是全部）空白字符解释为字段、单词或命令选项的定界符。

控制进程

建立与 控制终端的连接的会话发起者。如果该终端随后不再作为此会话的控制终端，则会话发起者将停止作为控制进程。

控制终端

与会话关联的终端。每个会话最多可拥有一个与其关联的控制终端，并且一个控制终端只能与一个会话关联。来自控制终端的某些输入序列会使信号发送到与控制终端关联的前台进程组中的所有进程。

控制字符

影响文本的录制、处理、传输或解释的除图形字符以外的字符。在 **ASCII** 字符集中， 控制字符是指值在 0-31 范围内以及值为 127 的字符。可通过按 Control 键（根据您的终端，该键可能标记为 CTRL、CONTROL 或 CNTL），同时按下一个字符键来生成控制字符（正如使用 SHIFT 键一样）。这些双键序列通常书写为以下形式： Control-**D** 、 Ctrl-**D** 或 **^D** ，其中 ^ 代表 Control 键。

口令

由 ASCII 字符组成的字符串，用于验证用户的身份。口令可以与用户和组关联。如果用户拥有口令，则口令会自动被加密并填入 **/etc/passwd** 文件中该用户所在行的第二个字段中。用户可使用 *passwd*(1) 命令创建或更改自己的口令。

库

包含用户程序可访问的一组子例行程序和变量的文件。库可以是归档库或共享库。例如， **/usr/lib/libc.a** 和 **/usr/lib/libc.sl** 是包含《HP-UX 参考手册》的第 2 节和第 3 节中所有标记为 (3C) 和 (3S) 的功能的库。同样， **/usr/lib/libm.a** 和 **/usr/lib/libm.sl** 是包含《HP-UX 参考手册》的第 3 节中所有标记为 (3M) 的功能的库。请参阅 *intro*(2) 和 *intro*(3C) 。

块

(1) HP-UX 用于在海量存储介质上进行访问和存储空间分配的基本信息单位。不同实现以及不同文件系统之间的块大小不同。为了向用户提供更统一的接口，无论介质的实际块大小是多少，大多数系统调用和实用程序都认为 块的大小为 512 字节。本手册中 块的含义也是如此，除非在其他条目中另行规定。

(2) 在写入长度可变的字符串数据的介质（例如 9 轨磁带）上，块表示相应字符串的大小。 块常用于区别 记录；一个块中包含多条记录，而记录数表示分块系数。

块设备专用文件

与使用多字节块而非一系列单个字节来传输数据的海量存储设备（如硬盘或磁带驱动器）相关联的专用文件（请参阅 字符设备专用文件）。可以挂接 块设备专用文件。 块设备专用文件可提供对其硬件特性不可见的设备的访问。

LANG

环境变量，用于通知计算机进程有关用户对 本国语言、 本地惯例和 编码字符集的要求。

Legacy 设备专用文件

与锁定至特定物理 硬件路径的 I/O 设备（磁带、磁盘等）相关的专用文件，它包含硬件路径信息（如设备文件名称和次设备号中的 SCSI 总线、目标和 LUN）。有关详细信息，请参阅 *intro*(7) 。

Legacy 硬件路径

采用 Legacy 格式的硬件路径，即由 **/** （斜线）字符分隔的一系列总线联结地址，这些地址最终指向主机总线适配器 (HBA)。在 HBA 下，其他地址元素由 **.** （句点）字符分隔。所有元素均用十进制表示。有关详细信息，请参阅 *intro*(7) 。

LIF

请参阅 逻辑交换格式。

LUN

LUN 表示一个终端设备，例如磁盘或磁带、磁盘阵列（海量存储术语）中的一块逻辑存储。也称为逻辑单元 (LU)。

LUN 硬件路径

可表示一个海量存储设备的多个路径的虚拟路径。它以一个虚拟总线联结节点（称为 虚拟根节点）开始，其地址为 64000。在该虚拟根节点下，通过一个虚拟总线地址和一个虚拟 LUN 标识符来寻址，其间以 **/** （斜线）字符分隔。有关详细信息，请参阅 *intro*(7) 。

lunpath 硬件路径

LUN 的硬件路径。它由一系列总线联结地址组成，以 **/** （斜线）字符分隔，最终目标指向主机总线适配器 (HBA)。在 HBA 下，其他地址元素以十六进制表示。第一个元素表示取决于传输的目标地址。最后一个元素是 LUN 地址，是以 64 位表示的目标报告的 LUN 标识符。有关详细信息，请参阅 *intro*(7) 。

链接

链接是 目录条目的同义词。它是将文件名与任意类型的文件相关联的对象。构成 链接的信息包括文件名和该文件的内容在海量存储介质上的位置。一个物理文件可以有多个指向它的链接。多个目录条目可以使其名称与一个给定文件关联。如果这些链接出现在不同目录中，则文件在每个链接中可能具有相同名称，也可能具有不同名称。不过，如果这些链接出现在同一个目录中，则每个链接在该目录中必须具有唯一名称。不允许存在指向目录的多重链接（由具有适当权限的用户创建的链接除外）。请参阅 *ln*(1) 、 *link*(2) 、 *unlink*(2) 和 符号链接。

另外，若要准备将要执行的程序，请参阅 链接程序。

链接程序

一种程序，可将一个或多个对象程序组合为一个程序，搜索库来解析用户程序引用，并构建 **a.out** 格式的可执行文件。此时该可执行文件即可通过程序加载器 *exec*(2) 执行。可使用 *ld*(1) 命令调用链接程序。链接程序通常称为链接编辑器。

流

常常与本手册第 3 节中介绍的标准 I/O 库例行程序一起使用的术语。简单地说，流就是由 *fopen*(3S) 库例行程序返回的文件指针（声明为 **FILE *stream** ）。流可能具有，也可能不具有与其关联的缓冲区（在缺省情况下会分配缓冲区，但可以使用 *setbuf*(3S) 改变分配情况）。

路径名

用斜线分隔并以任意文件名结尾的目录名序列。除最后一个文件名外，该序列中的所有文件名都 必须是目录。如果路径名以 斜线 (*/*) 开头，则表示 绝对路径名；否则表示 相对路径名。路径名定义了要在分层文件系统中找到特定文件所需遵循的路径。

具体地说，路径名就是以 null（空）结尾的字符串，其结构如下所示：

<path-name>::=<file-name>|<path-prefix><file-name>|/
<path-prefix>::=<rtprefix>|/<rtprefix>
<rtprefix>::=<dirname>/|<rtprefix><dirname>/

其中，<file-name> 是由 ASCII 斜线和空字符以外的一个或多个字符组成的字符串，<dirname> 是一个或多个字符（除 ASCII 斜线和空字符之外）组成的用于命名目录的字符串。在支持短文件名的系统上，文件名和目录名最多可包含 14 个字符，而在支持长文件名的系统上，文件名和目录名最多可包含 255 个字符。

一个 斜线 (*/*) 表示 根目录。连续的两个或多个斜线 (*////...*) 也被视为单个斜线。

除非特别说明，否则空路径名或长度为零的路径名表示命名不存在的文件。

路径名解析

将路径名解析为 文件层次结构中某个特定文件的路径的过程。多个路径名可解析到同一个文件，这取决于以绝对形式还是相对形式进行解析（请参阅下文）。路径名中的每个文件名都位于它的上级目录所指定的目录中（例如，在路径名 **a/b** 小段中，文件 **b** 位于目录 **a** 中）。如果不能实现这一点，则 路径名解析将失败。

如果路径名以斜线开头，则将路径名中第一个文件名之前的目录视为进程的 根目录，而该路径名称为 绝对路径名。如果路径名不是以斜线开头，则将路径名中第一个文件名前面的目录视为进程的当前工作目录，且该路径名称为 相对路径名。由一个斜线组成的路径名会解析为进程的根目录。

路径前缀

结尾带有表示 目录的可选 斜线的 路径名。

逻辑交换格式 **(LIF)**

在多台 HP 计算机上实现的海量存储的标准格式，帮助提高介质可传输性。有关详细信息，请参阅 *lif*(4) 。

裸磁盘

为存在 字符设备专用文件（允许在磁盘以及用户的读取或写入缓冲区之间进行直接传输）的磁盘指定的名称。一个读取或写入调用仅生成一次 I/O 调用。

msqid

请参阅 消息队列标识符。

命令

执行特定任务的指令。 HP-UX 命令通过称为 **Shell** 的 命令解释程序执行。HP-UX 支持包括 POSIX Shell (*sh-posix*(1)) 、 C Shell (*csh*(1)) 和 Korn Shell (*ksh*(1)) 在内的多种 Shell。有关受支持的 Shell 的详细信息，请参阅 *sh*(1) 。大多数命令由可执行文件（称为 实用程序 ）执行，该文件的形式可能是包含可执行对象代码的独立单元（程序），或者是包含将以给定顺序执行的其他程序列表的文件（Shell 脚本）。脚本可以包含对其他脚本以及对象代码程序的引用。典型的 命令由实用程序名称以及紧接其后的传递给该实用程序的参数组成。例如，在命令 **ls mydirectory** 中， **ls** 是实用程序名称，而 **mydirectory** 是传递给 **ls** 实用程序的参数。

命令集 **1980**

请参阅 **CS/80** 。

命令解释程序

一种程序，可从标准输入（从键盘键入或从文件中读取）读取文本行，并根据要求对文本行进行解释以执行其他程序。 HP-UX 的命令解释程序称为 **Shell** 。请参阅 *sh*(1) 和相关的手册条目。

模式

与文件系统中每个文件关联的一个 16 位单词，存储在 **i** 节点中。 “模式”中的 12 个最低有效位可决定文件所有者、文件组和所有其他用户的读取、写入和执行权限，并包含设置用户 ID、设置组 ID 和粘着位。如果您是文件所有者或超级用户，可以通过 *chmod*(1) 命令设置 12 个最低有效位。这 12 个位有时称为 权限位。4 个最高有效位指定关联文件的文件类型，并设置为 *open*(2) 或 *mknod*(2) 系统调用的结果。

目录

提供文件名与其内容之间的映射的文件，仅由操作系统处理。对于一个目录中包含的所有文件名，该目录包含指向每个文件的 **i** 节点的指针；该指针称为 链接。一个文件可以有多个链接出现在在同一文件系统中的多个位置上。每个用户可根据需要自由地创建目录（使用 *mkdir*(1) ），前提是新目录的 父目录授予了用户新建目录的权限。创建目录后，目录中即可以包含普通文件和其他目录。 HP-UX 目录的命名方式和行为与普通文件几乎完全相同，只有一个不同之处：不允许任何用户（包括超级用户）向目录本身写入数据；只有 HP-UX 操作系统具有此权限。

依照惯例，一个目录至少含有两个链接， **.** 和 **..** ，分别称为 点和 点**-**点。 **.** 表示目录本身，而 **..** 表示其 父目录。仅包含 **.** 和 **..** 的目录被视为空目录。

NLS

请参阅 本地语言支持。

NLSPATH

用于表示消息清单（请参阅 消息清单）的搜索路径的环境变量。

内部指令

请参阅 系统调用。

内核

HP-UX 操作系统。内核是指负责管理计算机资源（例如，分配内存、创建进程和调度要执行的程序等）的可执行代码。每当 HP-UX 运行时，内核都会驻留在 RAM（随机访问存储器）中。

内置仿真终端 **(ITE)**

HP-UX 内核中包含的“设备驱动程序”代码；该代码根据系统处理器的系列和型号，与计算机的内置键盘和显示器或者与计算机所连接的特定键盘和显示器相关联。有关详细信息，请参阅 系统控制台和随您系统附带的系统管理员手册。

PIC

请参阅 位置无关代码。

排序

根据按优先级建立的规则确定的预定义序列中字符串的逻辑顺序。这些规则确定了排序元素的排序序列，并管理包含多个排序元素的字符串的顺序，以便适应本国语言。

排序序列

对 排序元素进行排序时所应用的排序序列。为适应本国语言，可以将 排序序列视为 排序元素的相对顺序，该顺序由 **LANG** 环境变量的当前值设置。可以忽略排序序列中的字符，还可以为两个或多个排序元素指定相同的相对顺序（请参阅 *string*(3C) ）。

排序元素

在排序时用于决定字符串的逻辑排序（即 排序序列）的最小实体。为适应本国语言，排序元素或者由单个字符组成，或者由两个或多个作为单个实体排序的字符组成。 **LANG** 环境变量的当前值决定了当前排序元素集。

普通文件

一种 HP-UX 文件类型，其中包含 ASCII 文本（例如，程序源代码）、二进制数据（例如可执行代码）等。用户可通过 I/O 重定向、编辑器或 HP-UX 命令创建普通文件。

起始参考时间

从 国际标准时间 (**UTC**) 1970 年 1 月 1 日 0 时、0 分、0 秒开始的时间周期。通过增量确定从起始参考时间到所引用时间的时间长度。

在计算起始参考时间到所引用时间之间的秒数时，不考虑以不规则间隔发生的闰秒（1970 年到 1988 年间共发生了十四次闰秒）。

前台进程组

对于已经与控制终端建立连接的每个会话，仅有一个进程组作为该控制终端的前台进程组。前台进程组在访问其控制终端时具有某些后台进程组拒绝的权限。请参阅 *read*(2) 和 *write*(2) 。

前台进程组 **ID**

前台进程组的进程组 ID。

权限位

文件的 模式中 9 个最低有效位称为该文件的 权限位。这些位决定了文件的 所有者、文件所属的 组和所有其他用户的读取、写入和执行权限。这些位分为三部分：所有者、组和其他用户。每个部分用于进程的相应文件类。权限位包含在文件模式中，如 *stat*(5) 所述。 文件访问权限中详细介绍了文件权限位在访问决策中的用法。

权限组

权限组是已对其执行了 **setprivgrp** （请参阅 *getprivgrp*(2) ）操作的组，这样使它能够访问本来只有超级用户可以访问的某些系统调用。请参阅 相应的权限。

缺省搜索路径

sh(1) 、 *time*(1) 和其他 HP-UX 命令在由相对路径名（即不是以 斜线 (**/**) 开头的路径名）搜索文件时所使用的目录前缀序列。该路径由环境变量 **PATH** （请参阅 *environ*(5) ）定义。 *login*(1) 将 **PATH** 设置为等于 **:/usr/bin** ，这表示您的工作目录是搜索的第一个目录，然后搜索 **/usr/bin** 。通过修改 **PATH** 的值可以重新定义搜索路径。通常可在主目录下的 **/etc/profile** 和/或 **.profile** 文件中完成该操作。

__restrict

当应用程序开发人员直接或间接选择 C99 标准时，宏会有选择地应用于函数原型。如果用户选择 C99 标准，**__restrict** 宏将更改为 **restrict** 关键字。否则， **__restrict** 宏将扩展为空字符串。

SCCS

请参阅 源代码版本控制系统。

SCCS 文件

普通的文本文件，已经过修改以便使 源代码版本控制系统 (**SCCS**) 可使用它。此修改由 *admin*(1) 命令自动完成。另请参阅 **delta** 版。

semid

请参阅 信号量标识符。

Shell

HP-UX 操作系统的用户接口。 Shell 通常同时用作命令解释程序和解释性编程语言。系统会为每个登录的用户自动调用 Shell。有关详细信息，请参阅 *sh*(1) 及其相关的手册条目，以及您系统附带的教程。

Shell 程序

请参阅 **Shell** 脚本。

Shell 脚本

Shell 命令和 Shell 编程语言结构组成的序列，存储在文件中并作为用户命令（程序）调用。执行前无需进行编译，因为 Shell 可识别组成 Shell 编程语言的命令和结构。 Shell 脚本通常称为 **Shell** 程序或 命令文件。请参阅《Shells User Guide》。

shmid
请参阅 共享内存标识符。

SS/80
请参阅 **CS/80** 。

stderr
请参阅 标准错误。

stdin
请参阅 标准输入。

stdout
请参阅 标准输出。

设备
计算机外围设备或类似于外围设备应用程序的对象。

设备地址
请参阅 总线地址。

设备文件
请参阅 专用文件。

设置用户 **ID** 位
文件系统中每个文件的模式位中的一个位。如果执行了已设置其 设置用户 **ID** 位的文件，则执行该文件的进程的有效用户 **ID** 会设置为文件所有者的 实际用户 **ID** 。

设置组 **ID** 位
文件系统中每个文件的模式位中的一个位。如果执行了已设置其 设置组 **ID** 位的文件，则执行该文件的进程的 有效组 **ID** 将设置为文件所有者的 实际组 **ID** 。另请参阅 组。

时钟周期
系统中用于调度和计算的频率。其值是表示 **clock_t** 类型值的 **CLK_TCK** 所定义的每秒间隔数。 **CLK_TCK** 以前表示已定义的常量 **HZ** 。

实际用户 **ID**
指定给系统中每个用户的正整数。在 **/etc/passwd** 文件中会给每个有效的登录名 登录名分配一个实际用户 ID。使用修饰词“实际”是因为用户还可以拥有 有效用户 **ID** （请参阅 有效用户 **ID** ）。

每次进程创建子进程（通过 *fork*(2) ）时，子进程的实际用户 ID 都与其父进程的实际用户 ID 相同。这有助于确定进程中的文件访问权限。

实际组 **ID**
为系统中每个用户指定的正整数。用户与其 实际组 **ID** 之间的关联是在 **/etc/passwd** 文件中实现的。之所以要使用修饰词“实际”，是因为用户还可以拥有 有效组 **ID** 。之后可以在 **/etc/group** 文件中将实际组 ID 映射到组名，虽然这样做并不必要。这样，每个用户都会成为某一组（可以为无名组）的成员，即使该组仅包含一个成员也是

如此。

每次进程创建子进程（通过 *fork*(2) ）时，子进程的实际组 ID 都与其父进程的实际组 ID 相同。这有助于确定进程中的文件访问权限。

实用程序

一种可执行文件，可能包含可执行对象代码（即 程序 ），或将以指定顺序执行的 命令清单（即 **Shell** 脚本）。可以自己编写实用程序，作为可执行程序或 Shell 脚本（需要使用 Shell 编程语言编写）。

守护程序 **(daemon)**

在后台运行的进程，通常不受来自终端的终止指令的影响。其作用是执行不同的调度、清理和维护作业。例如，*lpsched*(1M) 就是一个 守护程序。该程序用于对 *lp*(1) 命令生成的行式打印机作业执行这些功能。 *cron*(1M) 是永久性 守护程序（即永远不应终止的守护程序）的一个例子。

守护程序 **(demon)**

UNIX 术语 守护程序 **(daemon)** 的另一种非标准拼写形式。

数据加密

一种对信息进行编码以保护机密数据或专有数据的方法。例如， HP-UX 可自动加密所有用户的口令。 HP-UX 采用的加密方法使用字符 **.** 、 **/** 、 **0-9** 、 **A-Z** 、 **a-z** 将 ASCII 文本转换为 base-64 形式。有关与此字符集相关的等效数字，请参阅 *passwd*(4) 。

死进程

请参阅 僵尸进程。

所有者

文件的所有者通常是该文件的创建者。不过，超级用户或当前所有者可以使用 *chown*(1) 命令或 *chown*(2) 系统调用更改文件所有权。文件所有者可以对其文件执行任何操作，包括删除、复制和移动文件，以及更改文件内容等。所有者还可以更改文件的模式。

tty

最初是电传打字机的缩写；现在一般指 终端。

提示符

Shell 在终端上显示的字符，表明系统已准备好接受命令。对于普通用户，提示符通常为美元符号 (**$**) （在 C Shell 中为 **%** ），对于超级用户则为井字符 (**#**) ，但您可以通过设置相应的 Shell 变量来重新将其定义为任何字符串（请参阅 *sh*(1) 和相关条目）。另请参阅 二级提示符。

图形字符

除控制字符以外的其他字符，在手写、打印或显示时具有可视表示。

退出信号

SIGQUIT 信号（请参阅 *signal*(2) ）。通过键入由电传处理程序定义为退出信号的字符，即可生成退出信号（请参阅 *stty*(1) 、 *ioctl*(2) 和 *termio*(7) ）。缺省值是通过键入 Ctrl-\ 生成的 ASCII FS 字符（ASCII 值为 28）。此信号通常会导致正在运行的程序终止，并生成包含已终止进程的“核心映像”的文件。核心映像可用于进行调试（某些系统不支持核心映像，并且在这些系统上不会生成上述文件）。

UTC

请参阅 起始参考时间。

网络文件系统 **(NFS)**

网络文件系统 (NFS) 允许客户端节点通过网络执行透明的文件访问。

通过使用 NFS，客户端节点可在位于各种服务器和服务器体系结构上并且跨各种操作系统的文件上运行。客户端上的文件访问调用（例如，读取请求）被转换为 NFS 协议请求，并通过网络发送到服务器系统。服务器接收请求，执行实际的文件系统操作，然后将响应发送回客户端。

NFS 使用构建于外部数据表示 (XDR) 协议顶层的远程过程调用 (RPC)，以无状态的方式运行。出于安全目的，RPC 协议允许在网络上交换版本和身份验证参数。

通过将文件系统条目添加到服务器的 **/etc/dfs/dfstab** 文件，服务器授予客户端访问特定文件系统的权限。

位置无关代码 **(PIC)**

无需修改即可在任意虚拟地址运行的对象代码。位置无关代码可以使用 PC 相关的编址模式和/或链接表。该代码最常用于共享库中，在这种情况下链接表由动态加载程序进行初始化。在指定了 **+z** 或 **+Z** 编译程序选项后，会生成位置无关代码。

文本文件

包含组织为一行或多行的字符的文件。这些行不能包含 NUL（空）字符，且任一行的长度（包括终止换行符在内）都不能超过 **LINE_MAX** 个字节。虽然内核或 C 语言实现都不区分文本文件和二进制文件（请参阅 ANSI C 标准 X3-159-19xx），但许多实用程序的行为仅在处理文本文件时才是可预知的。

文件

可进行写入和/或读取的字节流。 文件具有包括权限和类型在内的某些属性。文件类型包括 常规文件、 字符设备专用文件、 块设备专用文件、 **FIFO** 设备专用文件、网络设备专用文件、 目录和 符号链接。每个文件必须具有文件名，使用户（以及许多 HP-UX 命令）可引用该文件的内容。系统不会强制规定文件内容的特定结构，但是某些程序对此有强制规定。可以连续或随机地访问文件（使用字节偏移量作为索引）。对文件内容和结构的解释由访问该文件的程序完成。

文件层次结构

系统上一个或多个可用 文件系统的集合。这些 文件系统中的所有 文件都组织在单个层次结构中，其中所有非终端节点都是 目录。因为多个 链接可指向同一个 文件，所以目录又被恰当地称为定向图。

文件地址偏移量

文件地址偏移量指定文件中下一个 I/O 操作开始的位置。与常规文件或专用文件关联的每个 文件开放性说明都具有 文件地址偏移量。不存在为 管道或 **FIFO** 指定的文件地址偏移量。

文件访问模式

文件开放性说明的一个特性，用于确定所说明的文件是否开放为读取、写入还是同时执行这两种操作。请参阅 *open*(2) 。

文件访问权限

文件层次结构中的每个文件都有一组访问权限。这些权限用于确定某一进程是否可对该文件执行请求的操作（例如，开放文件允许写入）。访问权限是在通过 *open*(2) 或 *creat*(2) 系统调用创建文件时确立的，之后可以通过 *chmod*(2) 调用对其进行更改。这些权限由 *stat*(2) 或 *fstat*(2) 读取。

文件访问可以控制一个文件是否可被读取、写入或执行。目录文件使用执行权限来控制是否可以搜索相应目录。

文件访问权限由系统解释，因为这些权限将应用于三类不同的用户：文件 所有者、文件 组中的用户，以及任何其他用户（“其他用户”）。对于上述每一类用户，每个文件具有一组独立的访问权限。在进行访问检查时，系统会通过检查适用于调用者的访问信息，确定是否应授予权限。

如果满足下列任一条件，则会授予进程对文件的读取、写入以及执行/搜索权限：

- 进程的 有效用户 **ID** 是超级用户。
- 进程的 有效用户 **ID** 与文件所有者的用户 ID 相匹配，且已设置了文件模式的 所有者部分 (0700) 的相应访问位。
- 进程的 有效用户 **ID** 与文件所有者的用户 ID 不匹配，但进程的 有效组 **ID** 与文件的组 ID 相匹配，或者文件的组 ID 包含在进程的组访问列表中，且已设置了文件模式的 组部分 (070) 的相应访问位。
- 进程的 有效用户 **ID** 与文件所有者的用户 ID 不匹配，进程的 有效组 **ID** 与文件的组 ID 不匹配，且文件的组 ID 未包含在进程的组访问列表中，并且已设置了文件模式的“其他”部分 (07) 的相应访问位。

否则，将拒绝相应的权限。

文件更新时间

每个文件具有三个关联的时间值，当访问或修改文件数据后，或者更改了文件状态时这些值将更新。这些值在文件特性结构中返回，如 **<sys/stat.h>** 中所述。对于 HP-UX 中读取或写入文件数据或者更改文件状态的每一功能，相应的与时间相关的文件将标记为“marked-for-update”。在出现更新点时，所有已标记字段会设置为当前时间，且更新标记被清除。当文件不再为任何进程打开时，即会出现这样一个更新点。对 只读文件系统中的文件不会执行更新。

文件结束标记 **(EOF)**

(1) 通过 *stdio*(3S) 例行程序试图读取文件逻辑结尾之后时数据时返回的数据。在这种情况下，严格地说文件结束标记不是一个字符。

(2) ASCII 字符 Ctrl-**D** 。

(3) *stty*(1) 或 *ioctl*(2) （请参阅 *termio*(7) ）定义的字符，用作您终端上的文件结束标记 (EOF)。此标记通常为 Ctrl-**D** 。

(4) *read*(2) 的返回值，指示数据结束。

文件开放性说明

一个进程或一组进程如何访问文件的记录。每个 文件描述符仅引用一个 文件开放性说明 ，但一个 文件开放性说明可被多个文件描述符引用。 文件地址偏移量、 文件状态标记和 文件访问模式都是 文件开放性说明的属性。

文件链接数
: 指向一个特定文件的目录条目的数目。

文件描述符
: 每个进程中唯一的较小非负整数标识符，指向开放为读取和/或写入权限的文件。每个 文件描述符仅引用一个 文件开放性说明。

  文件描述符通过 *creat*(2) 、 *fcntl*(2) 、 *open*(2) 、 *pipe*(2) 或 *dup*(2) 等系统调用获得。 文件描述符用作 *read*(2) 、 *write*(2) 、 *ioctl*(2) 和 *close*(2) 等调用的参数。

  文件描述符的值范围为 0 到系统定义的最大值减 1。系统定义的最大值是 **<sys/param.h>** 中的 **NOFILE** 值。

文件名
: 最大长度为 14 个字节（在支持长文件名的文件系统上为 255 个字节）的字符串，用于表示常规文件、专用文件或目录。不能将字节值 NUL（空）和斜线 (*/*) 用作文件名中的字符。建议不要在文件名中使用 ***** 、 **?** 、 **,** 、 **[** 或 **]** ，因为 Shell 为这些字符赋予了特殊含义（请参阅 *sh*(1) 、 *csh*(1) 或 *ksh*(1) ）。避免使用 **-** 、 **+** 或 **=** 作为文件名开头，因为对于某些程序，这些字符表示后接命令参数。文件名有时称为路径名组成部分。建议不要使用在您的常用硬件上没有相应可打印图形的字符（虽然允许这样做），否则您的终端可能无法正确识别。

文件名可移植性
: 应当使用 可移植的文件名称字符集来构建文件名，因为在某些环境中，使用其他字符会导致混淆或意义不明。

文件其他用户类
: 如果一个进程不属于 文件所有者类或 文件组类，则该进程属于 文件其他用户类。

文件权限位
: 请参阅 权限位。

文件所有者类
: 如果一个进程的 有效用户 **ID** 与文件的用户 ID 相匹配，则该进程属于 文件所有者类。

文件系统
: 驻留在海量存储卷上的 文件和所支持的数据结构的集合。文件系统为引用这些文件的 文件序号提供了命名空间。有关文件系统的实现和维护的详细信息，请参考随系统提供的系统管理员手册。

文件序号
: 给定文件的对于文件系统唯一的标识符，也称为文件的 **i** 节点编号。每个 文件序号仅标识一个 **i** 节点。 文件序号在 文件层次结构中的各 文件系统之间不一定唯一。

文件指针
: 通过任一 *fopen*(3S) 标准 I/O 库例行程序获得的数据元素，该元素“指向”（引用）打开的将进行读取和/或写入的文件，并跟踪该文件中下一个 I/O 操作发生的位置（以相对于文件开头的字节偏移量的形式）。获得文件指针后，在使用任一标准 I/O 库例行程序时，必须使用它来指向打开的文件（有关这些例行程序的列表，请参阅 *stdio*(3S) ）。

文件状态标记

文件开放性说明的一部分。这些标记可用于修改系统调用（用于访问 文件开放性说明所描述的文件）的行为。

文件组类

如果一个进程不属于 文件所有者类，且进程的 有效组 **ID** 或 补充组 **ID** 之一与文件的关联组 ID 相匹配，则该进程属于该文件的 文件组类。

系统

HP-UX 操作系统。另请参阅 内核。

系统调用

通过高级语言（如 FORTRAN、Pascal 或 C）向用户提供的 HP-UX 操作系统内核功能。也称为“内部指令”或“系统内部指令”。《HP-UX 参考手册》》的第 2 节中介绍了可用的系统调用。

系统进程

系统进程是代表系统运行的进程。系统进程可以具有实现所定义的特殊特性。

系统控制台

由 HP-UX 指定了唯一状态且与专用文件 **/dev/console** 关联的键盘和显示器（或称终端）。所有引导 ROM 错误消息、 HP-UX 系统错误消息和某些系统状态消息都会发送到系统控制台。在某些情况下（例如在单用户状态下），系统控制台提供了与 HP-UX 进行通信的唯一机制。有关配置和使用系统控制台的详细信息，请参阅您系统附带的系统管理员手册和用户指南。

系统异步 **I/O**

一种执行 I/O 的方法，即进程通知驱动程序或子系统，它希望知道数据的到达时间或者何时可以执行写入请求。驱动程序或子系统维护一组缓冲区，而进程通过这些缓冲区执行 I/O 操作。有关更多信息，请参阅 *ioctl*(2) 、 *read*(2) 、 *select*(2) 和 *write*(2) 。

下级目录

请参阅 子目录。

相对路径名

不是以 斜线 (*/*) 开头的 路径名。该路径名表示文件相对于当前 工作目录的位置，且搜索将从该目录（而不是 根目录）开始。例如， **dir1/file2** 搜索组合先在当前的工作目录中搜索 **dir1** ； 目录；接着在 **dir1** 目录中搜索文件 **file2** 。

相应的权限

每个实现提供了一种方法，可以将各种权限与要求特殊权限的函数调用和函数调用选项的进程相关联。在 HP-UX 系统中， 相应的权限表示超级用户状态或与权限组关联的权限（请参阅 *setprivgrp*(1M) ）。

消息操作权限

在 *msgop*(2) 和 *msgctl*(2) 系统调用说明中，已指出了每个操作需要的操作权限。特定进程对于某个对象是否具有这些权限，由该对象的权限模式位决定，如下所示：

00400 由用户读取
00200 由用户写入
00060 由组读取和写入
00006 由其他用户读取和写入

如果下列一个或多个条件成立，则会授予进程对 **msqid** 的读取和写入权限：

- 进程的有效用户 ID 是超级用户。
- 进程的有效用户 ID 与 **msqid** 的关联数据结构中的 **msg_perm.[c]uid** 相匹配，且已设置 **msg_perm.mode** 的“user”部分 (0600) 的相应位。
- 进程的有效用户 ID 与 **msg_perm.[c]uid** 不匹配，但进程的有效组 ID 与 **msg_perm.[c]gid** 匹配，或 **msg_perm.[c]gid** 之一包含在进程的组访问列表中，且已设置 **msg_perm.mode** 的“group”部分 (00060) 的相应位。
- 进程的有效用户 ID 与 **msg_perm.[c]uid** 不匹配，且进程的有效组 ID 与 **msg_perm.[c]gid** 不匹配，并且任一 **msg_perm.[c]gid** 都不包含在进程的组访问列表中，且 **msg_perm.mode** 的“other”部分 (06) 的相应位已设置。

否则，将拒绝相应的权限。

消息队列标识符 **(msqid)**

由 *msgget*(2) 系统调用创建的具有唯一性的正整数。每个 **msqid** 都有一个消息队列和与其关联的数据结构。该数据结构表示为 **msqid_ds** 并包含下列成员：

```
struct
ipc_perm  msg_perm;      /* operation permission */
msgqnum_t msg_qnum;      /* number of msgs on q */
msglen_t  msg_qbytes;    /* max number of bytes on q */
msglen_t  msg_cbytes;    /* current number of bytes on q */
pid_t     msg_lspid;     /* pid of last msgsnd operation */
pid_t     msg_lrpid;     /* pid of last msgrcv operation */
time_t    msg_stime;     /* last msgsnd time */
time_t    msg_rtime;     /* last msgrcv time */
time_t    msg_ctime;     /* last change time */
                         /* Times measured in secs since 00:00:00 GMT, Jan. 1, 1970 */
```

可以使用 *ftok*(3C) 创建消息队列标识符。

msg_perm 是指定消息操作权限（请参阅下文）的 **ipc_perm** 结构。该结构包括下列成员：

```
uid_t   cuid;    /* creator user id */
gid_t   cgid;    /* creator group id */
uid_t   uid;     /* user id */
gid_t   gid;     /* group id */
```

```
mode_t   mode;       /* r/w permission */
```

msg_qnum 是队列中的当前消息数。 **msg_qbytes** 是允许队列包含的最大字节数。 **msg_lspid** 是执行 **msgsnd** 操作的上一个进程的进程 ID。 **msg_lrpid** 是执行 **msgrcv** 操作的上一个进程的进程 ID。 **msg_stime** 表示上一次执行 **msgsnd** 操作的时间， **msg_rtime** 表示上一次执行 **msgrcv** 操作的时间，而 **msg_ctime** 表示上一次执行 *msgctl*(2) 操作（更改了上述结构的成员）的时间。

消息清单

程序消息和提示符等程序字符串存储在与特定地理地区相对应的 消息清单中。在 消息清单中检索字符串要根据用户的 **LANG** 环境变量（请参阅 **LANG**）的值。

小数分隔符

将一个数字的整数部分与小数部分分隔开的字符。例如，在美国用法中 小数分隔符是小数点，而在欧洲则是逗号。

小写-大写转换

小写字符至其大写形式的转换。

斜线

文字字符 **/** 。由单个斜线组成的 路径名会解析为进程的 根目录。另请参阅 路径名解析。

斜线分隔符

请参阅 斜线。

信号

发送给进程的软中断信号，通知它发生了特殊情况或事件。也可以指事件本身。请参阅 *signal*(2) 。

信号量标识符 **(semid)**

由 *semget*(2) 系统调用创建的具有唯一性的正整数。每个 **semid** 都有一组信号量和与其关联的数据结构。该数据结构表示为 **semid_ds** ，包含下列成员：

```
struct
ipc_perm  sem_perm;      /* operation permission */
ushort    sem_nsems;     /* number of sems in set */
time_t    sem_otime;     /* last operation time */
time_t    sem_ctime;     /* last change time */
                         /* Times measured in secs since 00:00:00 GMT, Jan. 1, 1970 */
```

可以使用 *ftok*(3C) 创建信号量标识符。

sem_perm 是一种指定信号量操作权限（请参阅下文）的 **ipc_perm** 结构。该结构包括下列成员：

```
uid_t   cuid;      /* creator user id */
gid_t   cgid;      /* creator group id */
uid_t   uid;       /* user id */
gid_t   gid;       /* group id */
```

```
mode_t  mode;       /* r/a permission */
```

sem_nsems 的值等于相应组中信号量的数目。一组中的每个信号量都由名为 **sem_num** 的正整数引用。 **sem_num** 值为从零依次到 sem_nsems 减 1。 **sem_otime** 表示上一次执行 *semop*(2) 操作的时间，而 **sem_ctime** 表示上一次执行 *semctl*(2) 操作（更改了上述结构的成员）的时间。

信号量操作权限

在 *semop*(2) 和 *semctl*(2) 系统调用说明中，已指出了每个操作所需的权限。特定进程对于某个对象是否具有这些权限，由该对象的权限模式位决定，如下所示：

00400 由用户读取
00200 由用户更改
00060 由组读取和更改
00006 由其他用户读取和更改

如果下列一个或多个条件成立，则会授予进程对 **semid** 的读取和更改权限：

- 进程的有效用户 ID 是超级用户。
- 进程的有效用户 ID 与 **semid** 的关联数据结构中的 **sem_perm.[c]uid** 相匹配，且 **sem_perm.mode** 的“user”部分 (0600) 的相应位已设置。
- 进程的有效用户 ID 与 **sem_perm.[c]uid** 不匹配，且 **sem_perm.mode** 的“group”部分 (060) 的相应位已设置。
- 进程的有效用户 ID 与 **sem_perm.[c]uid** 不匹配，且进程的有效组 ID 与 **sem_perm.[c]gid** 不匹配，并且任一 **sem_perm.[c]gid** 都不包含在进程的组访问列表中，且 **sem_perm.mode** 的“other”部分 (06) 的相应位已设置。

否则，将拒绝相应的权限。

引导

加载、初始化和运行操作系统的进程。

引导 **ROM**

位于 ROM（只读存储器）中的程序，在每次打开计算机的电源时执行；用于通过其自身的操作使计算机进入所需状态。引导程序的前几条指令足以从输入设备将该程序的其余部分引入到计算机中，并启动计算所需要的功能。引导 ROM 的功能是对计算机硬件进行运行检测、查找可通过计算机访问的所有设备，然后加载指定的操作系统或根据特定搜索算法找到的第一个操作系统。

引导区域

海量存储介质的一部分，引导操作系统时使用的卷标题和“bootstrap”程序位于该区域。 引导区域专门保留给 HP-UX 使用。

映像

执行命令期间，您的计算机（或多用户计算机系统上属于您的部分）的当前状态。映像通常被视为执行期间任一特定时刻的计算机状态的“快照”。

硬件路径

与系统组件（总线、卡、附加 I/O 设备等）相关并提供与组件位置相关的信息的数字字符串。

用户 **ID**

每个系统用户都由一个称为 用户 **ID** 的整数标识，其范围为零到 **UID_MAX** （包括该值）。根据进程识别用户的方式的不同， 用户 **ID** 值可能表示为 实际用户 **ID** 、 有效用户 **ID** 或 保留的用户 **ID** 。

有效用户 **ID**

进程具有用于确定 文件访问权限的 有效用户 **ID** （如果有效用户 ID 为表示超级用户的 0，则还可确定关于系统调用的其他权限）。进程的有效用户 ID 由该进程正在执行的文件（命令）确定。如果已设置了该文件的设置用户 ID 位（位于文件的模式中，请参阅 模式），则进程的有效用户 ID 会设置为与该文件的用户 ID 相同。这使得进程似乎是该文件的所有者，可以使该进程访问必须访问的文件，以便程序能够成功执行（ **root** 拥有的许多 HP-UX 命令，如 **mkdir** 和 **mail** ，都设置了其设置用户 ID 位，使其他用户能够执行这些命令）。如果未设置该文件的设置用户 ID 位，则进程的有效用户 ID 将继承自其父进程。请参阅 实际用户 **ID** 和 设置用户 **ID** 位。

有效组 **ID**

每个进程都有一个 有效组 **ID** ，用于确定 文件访问权限。进程的 有效组 **ID** 由该进程正在执行的文件（命令）确定。如果已设置了该文件的设置组 ID 位（位于文件模式中，请参阅 模式），则进程的 有效组 **ID** 会设置为与该文件的组 ID 相同。这使得进程似乎属于该文件的组，因此可能使该进程访问必须访问的文件，以便程序能够成功执行。如果未设置该文件的设置组 ID 位，则进程的 有效组 **ID** 将继承自其父进程。进程的 有效组 **ID** 的设置仅在程序执行期间有效，之后，进程的有效组 ID 会设置为与其实际组 ID 相同。请参阅 组、 实际组 **ID** 和 设置组 **ID** 位。

元字符

对于 HP-UX Shell 以及 **ed** 、 **find** 和 **grep** 等命令（请参阅 *ed*(1) 、 *find*(1) 和 *grep*(1) ）具有特殊含义的字符。元字符集包括： **!** 、 **"**、 **&** 、 **'** 、 ***** 、 **;** 、 **<** 、 **>** 、 **?** 、 **[** 、 **]** 、 **‘** 和 **|** 。有关与每个元字符相关的含义，请参考 *sh*(1) 和相关的 Shell 手册条目。另请参阅 正则表达式。

源代码

按照指定的计算机语言的语法编写的基础性高级信息（程序）。对象（机器语言）代码是由源代码派生而来的。在处理 HP-UX Shell 命令语言时， 源代码是命令语言解释程序的输入。术语 **Shell** 脚本与此含义相似。使用 C 语言时， 源代码是 *cc*(1) 命令的输入。 源代码还可以表示满足上述任一条件的源代码的集合。

源代码版本控制系统 **(SCCS)**

一组 HP-UX 命令，用于将对 **SCCS** 文件所做的更改存储为单独“单元”（称为 **delta** 版）。在这些单元中，每个单元都包含对该文件所做的一处或多处文本更改，然后，可对 SCCS 文件应用或排除这些单元以便获得该文件的不同版本。组成 SCCS 的命令包括 *admin*(1) 、 *cdc*(1) 、 *delta*(1) 、 *get*(1) 、 *prs*(1) 、 *rmdel*(1) 、 *sact*(1) 、 *sccsdiff*(1) 、 *unget*(1) 、 *val*(1) 和 *what*(1) 。

粘着位

文件系统中每个文件的模式位中的一个位。如果粘着位设置在 常规文件中，则没有任何意义。

如果对目录设置该位，则只有该文件的所有者、包含该文件的目录的所有者或超级用户才能够删除或重命名该目录中的文件。另请参阅 *chmod*(2) 、 *rename*(2) 、 *rmdir*(2) 和 *unlink*(2) 。

正则表达式

由选择文本的零个或多个字符组成的字符串。该字符串中包含的所有字符都可以是文字字符，这意味着正则表达式仅与其自身相匹配；另外，一个或多个字符可以是 元字符 ，这表示一个正则表达式可以与多个文字字符串匹配。正则表达式通常出现在文本编辑器中（例如， *ed*(1) 、 *ex*(1) 或 *vi*(1) ），用来搜索一段特定的文字；或者出现在执行命令中（最常见的为 *grep*(1) ），用来搜索文件中特殊的字符。在 Shell 中也会遇到正则表达式，特别是在命令行上引用文件名时。

只读文件系统

文件系统的一种特性，可禁止对文件系统进行修改。

中断信号

SIGINT 发送的信号（请参阅 *signal*(2) ）。此信号通常会终止您正在运行的任何程序。可以使用 *ioctl*(2) 或 *stty*(1) 重新定义发送此信号的键（请参阅 *termio*(7) ）。通常为 ASCII DEL (rubout) 字符（DEL 键）或 BREAK 键。还经常使用 Ctrl-**C** 代替。

终端

符合 *termio*(7) 规范的 字符设备专用文件。

终端关联

进程组创建者用于在自身与特定终端之间建立关联的过程。只要进程组创建者执行（直接或间接） *open*(2) 或 *creat*(2) 系统调用来打开终端，一个终端就会与该进程组创建者相关联（而且该进程组创建者后来创建的所有进程都将与其关联，请参阅 终端组）。那么， 如果正在执行 *open*(2) 或 *creat*(2) 的进程是进程组创建者，且该进程组创建者尚未与终端关联，同时正被打开的终端尚未与进程组关联，则会建立关联（不过，还请参阅 **O_NOCTTY** 的 *open*(2) 说明）。

关联的终端通过将进程组的进程组 ID 存储在内部结构中，来跟踪其进程组关联。

终端关联可提供两种好处。第一，从终端发送的所有信号都会发送到终端组中的所有进程。第二，终端组中的所有进程都能够执行出入常规终端驱动程序 **/dev/tty** 的 I/O，该驱动程序可自动选择关联的终端。

当进程组创建者终止时，终端与终端组的关联将会断开，之后，挂起信号会发送到进程组中所有剩余的进程。此外，如果终端组中的进程（不是进程组创建者）通过 *setpgrp*(2) 系统调用成为进程组创建者，其终端关联也会断开。

请参阅 进程组、 进程组创建者、 终端组和 *setpgrp*(2) 。

终端设备

请参阅 终端。

主机名

网络中可唯一标识系统的字节字符串。可通过 *hostname*(1) 命令查看和（或）设置系统的主机名。可在 *hostname*(5) 联机帮助页中获得更多信息。另请参阅 节点名称。

主目录

环境变量 **HOME** 的值指定的目录名。当您第一次登录时， *login*(1) 会自动将 **HOME** 设置为您的 登录目录。您可以随时更改其值。通常在 登录目录中包含的 **.profile** 文件中进行此更改。设置 **HOME** 不会影响您的 登录目

录；只是使您可以更方便地引用您可能最常用的目录。

主设备号

是创建专用文件而使用的专门数字，这些专用文件可启用与特定设备进行通信的 I/O 通道。该数字表明设备将使用的设备驱动程序。有关详细信息，请参考 *mknod*(2) 和您系统附带的系统管理员手册。

专用文件

与 I/O 设备关联的文件。通常称为 设备文件。专用文件的读取和写入与 普通文件相同，但在激活相关设备时要求读取或写入结果。依照惯例并为了确保一致性，这些文件应始终位于 **/dev** 目录中。另请参阅 文件。

专用系统进程

专用系统进程是指那些对基本系统操作很关键的进程，其中包括：调度程序、初始化进程（即 **init** ）和分页程序。

子进程

原有进程通过 *fork*(2) 系统调用所创建的新进程。因此，该新建进程称为原有进程的 子进程。原有进程是新建进程的 父进程。请参阅 父进程和 **fork** 。

子命令集 **1980**

请参阅 **CS/80** 。

子目录

在文件系统层次结构中比给定目录低一级或多级的目录。有时称为 下级目录。

字符

用于组织、控制或表示文本的元素。字符包括 图形字符和 控制字符。

字符集

用于以本国语言或机器语言进行通信的一组字符。

字符设备专用文件

与逐字节传输数据的 I/O 设备相关联的专用文件。其他字节模式的 I/O 设备包括打印机、9 轨磁带驱动器，以及以“原始”模式访问的磁盘驱动器（请参阅 “裸磁盘”）。 字符设备专用文件没有预定义结构。

总线地址

HP-UX 用于定位特定设备的地址所包含的数字。 总线地址由外围设备上的开关设置确定，该设置使计算机能够区分连接到同一接口的两个设备。 总线地址有时称为“设备地址”。

组

请参阅 组 **ID** 。

组 **ID**

可关联必须全部有权访问同一组文件的零个或多个用户。在文件 **/etc/passwd** 和 **/etc/logingroup** （如果存在）中，通过必须介于零到 **UID_MAX** （包括该值）之间的数字组 ID 定义组成员。具有相同组 ID 的用户是同一个组的成员。 ASCII 组名与文件 **/etc/group** 中的每个组 ID 关联。一个组 ID 还与 文件层次结构中的所有文件相关联，且每个文件的模式中包含仅应用于此组的一组权限位。所以，如果您属于一个与文件关联的组，并且文件模

式中已为该组授予了适当权限，则您可以访问该文件。当组标识与进程相关联时，则组 ID 值称为 实际组 **ID** 、有效组 **ID** 、 补充组 **ID** 或 保留的组 **ID** 。另请参阅 权限组和 设置组 **ID** 位。

组访问列表

用于确定资源可访问性的一组 补充组 **ID** 。按照 文件访问权限中的约定访问检查。

作业控制

通过作业控制，用户可以有选择地停止（挂起）进程的执行，并在以后继续（恢复）其执行。

用户通过由系统终端驱动程序和某些 Shell（请参阅 *sh*(1) ）共同提供的交互式界面来使用此功能。终端驱动程序可识别用户定义的“挂起字符”，该字符会使当前的前台进程组停止，以及恢复用户的作业控制 Shell。作业控制 Shell 可以对前台或后台中已停止的进程组提供继续执行的命令。当后台进程组的任何成员试图从用户终端读取或向该终端写入数据时，终端驱动程序也会停止该后台进程组。这样，用户可以结束或挂起 前台进程组而不会引起中断，并在更方便的时间继续执行已停止的 后台进程组。

有关其用法和安装的详细信息，请参阅 *stty*(1) 、 *sh*(1) 和相关的 Shell 条目；有关实现的详细信息，请参阅 Shell 条目以及 *signal*(2) 和 *termio*(7) 。

另请参阅

introduction(9)。

名称

introduction - HP-UX 操作系统和 *HP-UX* 参考手册

简介

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 (SVR4)，并包括 Fourth Berkeley Software Distribution (4BSD) 中的重要功能。

改进的功能包括由 HP 开发的增强功能和其他功能，使 HP-UX 成为功能强大、实用、可靠的操作系统，从而可以支持大量应用程序，范围从简单的文本处理到复杂的工程图形和设计。该操作系统已经可以用来控制各种仪器和其他外围设备。实时功能进一步扩展了 HP-UX 的灵活性，使其成为一种功能强大的工具，用以解决在设计、生产、业务和其他注重响应速度和性能的领域中遇到的难题。

通过支持大量国际语言，HP-UX 可以使用多种语言与用户进行交互操作。 HP-UX 可以轻松地与局域网和资源共享设备相连。通过使用行业标准协议， HP-UX 可以灵活地与其他计算机和操作系统交互操作。可选的软件产品扩展了 HP-UX 的功能，可满足大量的特殊需求。

《HP-UX 参考手册》不是面向初学者的学习工具。它主要是一种参考工具，供熟悉 UNIX 或类似于 UNIX 的系统的有经验用户使用。如果尚不熟悉 UNIX 或 HP-UX，请参考随系统提供或单独提供的初学者指南系列参考资料、教程手册和其他学习文档。有关系统实现和维护的详细信息在《HP-UX System Administrator's Guide》中进行了说明。

其他联机帮助页

本简介和 *intro* 联机帮助页一节描述了随 HP-UX 提供的“核心”联机帮助页。其他联机帮助页可能随可选的 HP-UX 和第三方软件单独提供，它们可能位于核心联机帮助页所在的目录，也可能位于其他目录。

联机帮助页结构

《HP-UX 参考手册》及其相应联机部分的内容由许多称为 联机帮助页的独立条目组成。这些条目也称为 手册条目或 参考页。

为了方便起见，联机帮助页分为八个专用的节。印刷手册还包含每卷的目录和一个复合索引。

每个联机帮助页都包含一个或多个印刷页面，页面上方两角印着联机帮助页的名称和节编号。各个联机帮助页在该参考手册的每一节内部按字母顺序排列，但每一节的开头都有一个 *intro* 页。联机帮助页按名称和节编号来引用，其格式为 页名（ 节编号）。

如果系统提供了联机帮助页，则可以通过 **man** 命令联机获得联机帮助页。有关详细信息，请参考第 1 节的 *man*(1) 联机帮助页。

印刷手册中的每一页都有两个页码，位于页面底部。在每个新的联机帮助页中，中央的页码从第 1 页开始，以正常字体放在两个短线之间。每页底部角落的页码为该卷内的各个页面的序号。如同查阅词典一样，用户通常通过页首的字母标题来查找联机帮助页。

某些联机帮助页介绍两个或更多的命令或例行程序。此时，联机帮助页通常以“名称”一节中的第一个命令或函数命名。有时，联机帮助页名是“名称”一节中的描述符。此时，此名称用于对命令或函数进行概述。例如，带有组描述符 **acct:** 的 *acct*(1M) 联机帮助页描述 **acctdisk** 、 **acctdusg** 、 **accton** 和其他命令，而带有组描述符 **string:** 的 *string*(3C) 联机帮助页则描述许多字符串函数。

HP-UX 参考手册各章节内容

《HP-UX 参考手册》包含下列章节：

卷目录 （印刷卷）

按各节中的顺序完整地列出所有联机帮助页，同时还按字母顺序混杂列出与各个联机帮助页不同的所有命令、函数和功能名称。

第 **1** 节：用户命令

通常直接由用户调用或从命令语言过程（脚本）中调用的程序。

第 **1M** 节：系统管理命令

用于系统安装和维护（包括引导过程、故障恢复、系统完整性测试和其他需求）的命令。此节中的大多数命令都需要超级用户权限。

第 **2** 节：系统调用

HP-UX 内核中的条目（包括 C 语言接口）。这些主题主要供程序员参考。

第 **3** 节：库函数

以二进制格式驻留在各种系统库中的可用子例行程序。这些主题主要供程序员参考。

第 **4** 节：文件格式

各种类型的文件的结构，（例如，头文件）主要供管理员和程序员参考。例如， *a.out*(4) 中介绍链接编辑器输出文件格式。仅由一个命令使用的文件（如汇编程序使用的中间文件）不作介绍。与第 4 节中的格式对应的 C 语言声明位于目录 **/usr/include** 和 **/usr/include/sys** 中。

第 **5** 节：其他主题

各种信息，如字符集、宏程序包和内核参数的说明。

第 **6** 节（未使用）

本节通常用于娱乐。 HP-UX 中未附带任何内容。

第 **7** 节：设备专用文件

用于在 HP-UX 与系统 I/O 设备之间提供链接的设备专用文件 (DSF) 的特性。每个主题的名称通常是指 I/O 设备的类型，而不是各个专用文件的名称。

第 **8** 节：系统管理命令

某些 UNIX 和 Linux 供应商将系统管理命令置于本节中。某些第三方供应商在 HP-UX 中安装本节中提供的各命令。

第 **9** 节：一般信息

在 HP-UX 环境中使用的一般介绍（例如，本节）和术语词汇表。

本节还可供驱动程序开发工具包使用，以便使用 9E、9F 和 9S 小节编号存储其函数和结构联机帮助页。

复合索引 （印刷手册）

在每个联机帮助页的开始部分，按字母顺序列出以“名称”一节为准的关键字和主题以及其他信息，这些信息以交叉引用联机帮助页名和节编号的形式列出。索引还包含对各种命令解释程序 (Shell) 中的内置功能的引用。

联机帮助页格式

所有联机帮助页都遵循一个固定的节标题格式，但并非每个联机帮助页都包含所有节标题。某些联机帮助页有自述性的专用标题。

名称　提供命令、函数或功能的名称，并简要说明其用途。

概要　总结命令或程序实体的语法。使用下列约定：

等宽字符表示应当完全按显示的形式输入的字符。这些字符在联机帮助页中显示为粗体。

斜体字符串表示应当使用适当值来替换的变量元素。

罗马字体的方括号 ([ ]) 表示内容是可选的。

罗马字体的花括号 ({ }) 表示必需的元素（通常在选项中）。

省略号 (...) 表示前一元素及其前置的空白字符可以重复。

注释：以短线 **(-)** 、加号 **(+)** 或等号 **(=)** 开头的参数通常定义为命令选项，即使该参数出现在文件名可能出现的位置上。因此，不要在文件名前面加上 **-** 、 **+** 或 **=** 字符。

可选小节包括：

参数	对前面语法中各参数的说明（针对函数）。
结构成员	对前面语法中各结构元素的说明（针对结构）。
备注	有关特定软件或硬件要求的信息。

说明　讨论每个条目的功能和行为。

可选小节包括：

选项	对选项参数的说明（针对命令）。
操作数	对非选项参数和关键字的说明（针对命令）。
访问控制列表	
多线程应用信息	
安全性限制	有关限制和使用此条目所需的权限的信息。

外部语言环境影响
有关可影响系统行为的外部因素（例如，环境变量）的信息。

可选小节包括：

环境变量	与语言相关的环境变量以及其环境变量对系统行为的影响。
国际代码集支持	是否支持单字节或多字节字符。

网络功能
只有当使用此处描述的网络功能时，该标题下的信息才适用。

可选小节包括：

NFS	有关网络文件系统的信息。

返回值 介绍由函数调用或命令的返回代码 **($?)** 返回的值。

诊断信息
介绍可能生成的诊断信息（针对命令）。但不列出自述性消息。

可选小节包括：

错误

警告

错误 函数错误值（在 **errno** 中设置）及其相应的错误条件（针对函数）。

举例 典型用法的示例。

警告 潜在的问题和缺陷。

相关内容
在 HP-UX 操作中与使用特定硬件、软件或硬件和软件组合相关的变化形式。

作者 指明由此联机帮助页记录的软件的主要开发人员。除非另有说明，否则，条目来源均为 System V。

文件 由程序或命令使用或受其影响的文件名。

另请参阅
提供相关联机帮助页和其他文档资料的参考手册。

符合的标准
HP-UX 组件所符合的标准规范（适用于下列一个或多个行业标准所述的每个命令或子例行程序入口点）。

各种标准包括：

AES	OSF Application Environment Specification

ANSI C	ANSI X3.159-1989
FIPS 151-1	Federal Information Processing Standard 151-1 (National Institute of Standards and Technology)
FIPS 151-2	Federal Information Processing Standard 151-2 (National Institute of Standards and Technology)
POSIX.1	IEEE Standard 1003.1-1988 (IEEE Computer Society) （用于计算机环境的可移植操作系统接口）
POSIX.2	IEEE Standard 1003.2-1990 (IEEE Computer Society) （用于计算机环境的可移植操作系统接口）
POSIX.4	IEEE Standard 1003.1b-1993 (IEEE Computer Society) （用于计算机环境的可移植操作系统接口）
SVID2	System V Interface Definition Issue 2
SVID3	System V Interface Definition Issue 3
XPG2	X/Open Portability Guide Issue 2 (X/Open, Ltd.)
XPG3	X/Open Portability Guide Issue 3 (X/Open, Ltd.)
XPG4	X/Open Portability Guide Issue 4 (X/Open, Ltd.)
XPG4.2	X/Open Portability Guide Issue 4 Version 2 (X/Open, Ltd.)

HP-UX 入门

它简要地概述了如何使用 HP-UX 系统：如何登录和注销、如何通过计算机进行通信以及如何运行程序。

HP-UX 使用 控制字符来执行某些功能。控制字符一般显示为 ^*x* 形式，如 **^D** 表示 Control-D。请在按住 **Control (Ctrl)** 键的同时按字符键。

注释：键名 **Enter** 和 **Return** 表示同一个键。

登录

要登录，必须拥有有效的用户名和口令，可通过系统管理员获得用户名和口令。

建立连接后，系统在终端上显示 **login:** 。键入用户名并按 **Enter** 键。输入口令（系统不回显）并按 **Enter** 键。

在出现第一个提示符前，可能会显示版权声明列表及当天的消息。

尽可能使用小写字母键入登录名，这一点很重要。如果键入大写字母，HP-UX 将假定终端不能生成小写字母，并将输入的大写字母作为小写字母来处理。

成功登录后，系统将启动登录 Shell。缺省值是 POSIX Shell (**/usr/bin/sh**)。POSIX Shell（及其先前版本 Korn Shell 和 Bourne Shell）使用 **$** 作为用户的缺省提示符。 C Shell 使用 **%** 。所有 Shell 都使用 **#** 作为缺省超级用户提示符。

有关登录的详细信息，请参阅 *login*(1)；要更改口令，请参阅 *passwd*(1)；要更改登录 Shell，请参阅 *chsh*(1)。

注销

可以通过键入 **exit** 命令或 **eof** （文件结束标记）字符，来注销 Shell （请参阅下面的“特殊交互式字符”一节）。Shell 将终止，并且 **login:** 提示符将再次出现（如果使用的是 C Shell、Korn Shell 或 POSIX Shell，请参阅 *csh*(1)、 *ksh*(1) 或 *sh-posix*(1)，以了解有关 **ignoreeof** 特殊命令的信息）。

如何通过终端进行通信

HP-UX 会收集键盘输入字符并将其保存到缓冲区中。在键入 **Enter** 键之前，不会将累积的字符传递给 Shell 或其他程序。

HP-UX 终端输入/输出是全双工的。它有完全预读功能，这意味着即使程序正在显示屏幕或终端上打印，也可以随时键入。当然，如果输出过程中进行键入，则输出屏幕将散布有输入字符。但是，所键入的内容都会被保存，并按正确的顺序加以解释。预读量有一定的限制，不过该限制很宽松，不会超出，除非系统严重超负荷运行或操作不正常。如果超出预读限制，系统将放弃 所有已保存的字符。

stty(1) 联机帮助页说明如何描述系统的终端特性。 *profile*(4) 联机帮助页说明如何在每次登录时自动完成该任务。

特殊交互式字符

许多特殊字符用来控制终端的输入和输出。这些字符都有缺省值，可使用 **stty** 命令来重新定义（请参阅 *stty*(1)）。 **stty** 名称的定义位于 *termio*(7) 和 *termiox*(7) 中。

注释：系统管理员可通过使用 **stty** 命令更改 **/dev/ttyconf** 设备文件的特性，来修改系统登录缺省值。

stty 名称	登录时的系统缺省值 字符（**ASCII** 名称；键名称）	一般用户 重新定义
eof	**^D** (EOT)	
erase	**#**	**^H** (BS; Backspace)
kill	**@**	**^U** (NAK)、**^X** (CAN)
intr	**^?** (DEL；Delete、Rub、Rubout)	**^C** (ETX)
quit	**^** (FS)	
start	**^Q** (DC1；X-ON)	
stop	**^S** (DC3；X-OFF)	

eof 字符用于终止由程序或脚本读取的终端“文件”输入。通过扩展， **eof** 还可以终止 Shell （请参阅上述的“注销”一节）。

清除字符可清除键入的最后一个字符。连续使 清除将一直清除到输入行的开头，但不会越过该行的开头。

抹行字符可删除在终端输入行之前键入的所有字符。

intr 字符可生成绕过输入缓冲区的中断信号。该信号一般会使正在运行的程序终止。该字符可用来停止不需要的冗长输出内容。但是，程序可以选择完全忽略该信号，或在该信号发生时收到通知（而不是终止）。例如， **vi** 编

辑器捕获中断信号，然后停止正在执行的操作，而不是终止，这样，中断信号可以用来暂停编辑操作，而不会丢失正在编辑的文件。

quit 字符可生成绕过输入缓冲区和大多数程序陷阱的退出信号，并使正在运行的程序终止。它会在当前目录中引发核心转储。

stop 字符可用来暂停向终端输出内容。它常用在视频终端上，用于在阅读显示的内容时暂停向显示屏幕输出内容。然后，可以通过键入 **start** 字符恢复输出。当使用 **stop** 和 **start** 暂停或恢复输出时，它们会绕过键盘命令行缓冲区，并且不会传递给程序。但是，在键盘上键入的任何字符都会被保存在程序中，用作后续的输入。

如果使用前置的 **** 对 **eof** 、 清除和 抹行字符进行转义，则这些字符可用作正常文本字符，如 **\^D** 。因此，要清除 **** ，需要使用两个 清除。

在输入行中不能对 **intr** 、 **quit** 、 **start** 和 **stop** 字符进行转义。

行结束标记和制表符

除了能够适应终端速度外， HP-UX 还会很灵活地判断您的终端是使用换行键，还是必须使用回车符/换行符进行模拟。在后一种情况下，所有键入的回车符都会变为换行符（标准的行分隔符），并且回车符/换行符对将回显到终端中。如果进入错误模式，请使用 **stty** 命令更正它（请参阅 *stty*(1) ）。

制表符可在 HP-UX 源程序中自由使用。如果终端没有 Tab 功能，则可以选择在输出时将制表符更改为空格，并在输入时将制表符回显为空格。 **stty** 命令可设置或重置该模式。缺省情况下，系统假定制表符以八个字符为间隔设置。如果终端支持制表符，则 **tabs** 命令（请参阅 *tabs*(1) ）可以在终端上设置制表位。

如何运行程序

成功登录 HP-UX 后， Shell 将监视来自终端的输入。 Shell 会接受从终端键入的行，并将它们拆分成命令名和参数，然后执行命令。命令可以是内置 Shell 的名称、可执行的命令脚本或可执行的程序。系统提供的命令没有特殊之处，只是它们保存在可被 Shell 查找的目录中。您也可以将命令保存在自己的目录中，并告知 Shell 在何处查找该命令。

命令名是 Shell 输入行中的第一个单词；命令及其参数之间使用空白（一个或多个空格和/或制表符）进行分隔。

当一个程序终止时，Shell 在正常情况下会重新获得控制，并提示您可以运行其他命令了。 Shell 还有其他许多功能，它们会在下列相应联机帮助页中加以详细介绍：用于 POSIX Shell 的 *sh-posix*(1) 、用于 Korn Shell 的 *ksh*(1) 或用于 C Shell 的 *csh*(1) 。

当前目录

HP-UX 使用目录分层结构来安排文件系统。当系统管理员为您提供了用户名后，该管理员也为您创建了一个目录（正常情况下与您的用户名相同，称为 **login** 目录或 **home** 目录）。登录后，该目录成为您的 当前或 工作目录，并且缺省情况下键入的任何文件名都假定位于该目录中。由于您是该目录的所有者，因此您拥有完全权限，可以读取、写入、更改或删除其内容。您对其他目录和文件拥有的权限由其各自的所有者或系统管理员授予或拒绝。要更改当前工作目录，请使用 **cd** （请参阅 *cd*(1) ）。

路径名

要引用不在当前目录中的文件，必须使用路径名。完全（绝对）路径名的开头是 **/** ，它是整个文件系统 根目录的名称。斜线后面是到达要引用的文件名的每个目录的名称，各个目录包含下一层子目录（后面紧跟 **/** ）（例如，

/usr/ae/filex 指的是目录 **ae** 中的文件 **filex** ，而 **ae** 本身是 **usr** 的一个子目录； **usr** 是根目录的一个子目录）。有关 路径名格式定义的信息，请参阅 *glossary*(9) 。

如果当前目录包含子目录，则这些子目录中的文件的路径名以相应子目录名开头（不含前置的 **/** ）。通常，路径名可以在可使用文件名的任何位置上使用。

修改目录内容的重要命令是 **cp** 、 **mv** 和 **rm** ，它们分别复制、移动（即重命名、重定位或同时执行这两者）和删除文件。要确定文件状态或目录内容，请使用 **ls** 命令。可以使用 **mkdir** 创建目录，使用 **rmdir** 删除目录，使用 **mv** 重命名目录（请参阅 *cp*(1) 、 *ls*(1) 、 *mkdir*(1) 、 *mv*(1) 、 *rm*(1) 和 *rmdir*(1) ）。

编写程序

要将源程序文本输入到 HP-UX 文件中，请使用文本编辑程序，如 **vi** 、 **ex** 或 **ed** （请参阅 *vi*(1) 、 *ex*(1) 和 *ed*(1)）。可以在 HP-UX 中使用的三种主要语言是 C（请参阅 *cc_bundled*(1) 和 *cc*(1) ）、FORTRAN（请参阅 *f77*(1) ）和 aC++（请参阅 *aCC*(1) ）。在使用编辑器输入程序文本并将其写入文件（文件名带有合适的后缀）后，可以将该文件的名称作为参数提供给相应的语言处理程序。通常，语言处理程序的输出保留在当前目录中名为 **a.out** 的文件中。由于后续编译的结果可能也放在 **a.out** 中，从而会覆盖当前输出，您可能希望使用 **mv** 为该输出提供一个唯一的名称。如果使用汇编语言来编写程序，可能需要将库子例行程序与该程序链接（请参阅 *ld*(1)）。 FORTRAN、C 和 aC++ 都会自动调用此链接程序。

如果完成这一完整过程后未遇到任何诊断信息，则可以通过在提示符后为 Shell 提供程序名来运行得到的程序。

程序可以从命令行接收参数，正如系统程序使用 *argc* 和 *argv* 参数来接收参数一样。有关详细信息，请参阅适用于您的《Programmer's Guide》语言版本。

文本处理

几乎所有的文本都是通过文本编辑器输入的。 HP-UX 提供的首选编辑器是 **vi** 编辑器。对于批处理文本文件，**sed** 编辑器的效率会很高。使用 **vi** 时， **ex** 编辑器对于处理某些情况而言很有用，但是，其他大多数编辑器都很少使用，除非在各种脚本中。

下列编辑器虽然使用不同的名称，但实际上是同一个程序： **vi** 、 **view** 和 **vedit** （请参阅 *vi*(1) ）以及 **ex** 和 **edit** （请参阅 *ex*(1) ）。有关 **sed** 流编辑器的信息，请参阅 *sed*(1) 。有关 **ed** 行编辑器的说明，请参阅 *ed*(1) 。

最常用于在终端上显示文本的命令是 **cat** 、 **more** 和 **pr** （请参阅 *cat*(1) 、 *more*(1) 和 *pr*(1) ）。 **cat** 命令仅仅用于将 ASCII 文本复制到终端，而不进行任何处理。 **more** 命令用于在终端上一次显示一整屏文本，然后暂停以等待用户确认，用户确认后才继续运行。 **pr** 命令用于对文本进行分页、提供标题并具有多列输出的功能。 **pr** 最常与 **lp** 命令一起使用（请参阅 *lp*(1) ），以便将格式化的文本通过管道传输到行式打印机。

用户间通信

某些命令可使用户间进行通信。即使您不打算使用这些命令，了解它们也是有好处的，因为其他人可能使用它们与您通信。要与当前已登录的其他用户通信，可以使用 **write** 将文本直接传输到该用户的显示屏幕中（如果该用户已授予执行该操作的权限）。否则，为便于使用，可以使用 **elm** 、 **mailx** 或 **mail** 向其他用户的邮箱发送邮件。然后 HP-UX 会通知该用户邮件已到达（如果当前已登录）或有邮件（用户下次登录时）。有关如何使用这些命令的说明，请参考 *elm*(1) 、 *mail*(1) 、 *mailx*(1) 和 *write*(1) 。

认可

UNIX 是 The Open Group 的注册商标。

另请参阅

cat(1)、 cc_bundled(1)、 cd(1)、 chsh(1)、 cp(1)、 csh(1)、 ed(1)、 ex(1)、 ksh(1)、 ld(1)、 login(1)、 lp(1)、 ls(1)、 mail(1)、 mailx(1)、 man(1)、 mkdir(1)、 more(1)、 mv(1)、 passwd(1)、 pr(1)、 rm(1)、 rmdir(1)、 sed(1)、 sh(1)、 sh-posix(1)、 stty(1)、 tabs(1)、 vi(1)、 write(1)、 a.out(4)、 profile(4)、 glossary(9)。

HP 技术文档资料网站网址为： **http://docs.hp.com** 。