

第二章：系统调用

2.1 系统编程概述

操作系统的职责

操作系统用来管理所有的资源，并将不同的设备和不同的程序关联起来。

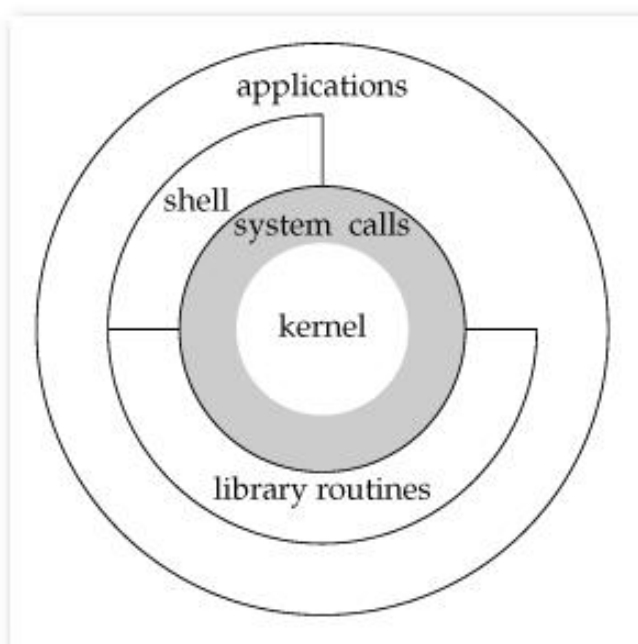
什么是 Linux 系统编程

在有操作系统的环境下编程，并使用操作系统提供的系统调用及各种库，对系统资源进行访问。

学会了 C 语言再知道一些使用系统调用的方法，就可以进行 Linux 系统编程了。

2.2 系统调用概述

类 UNIX 系统的软件层次



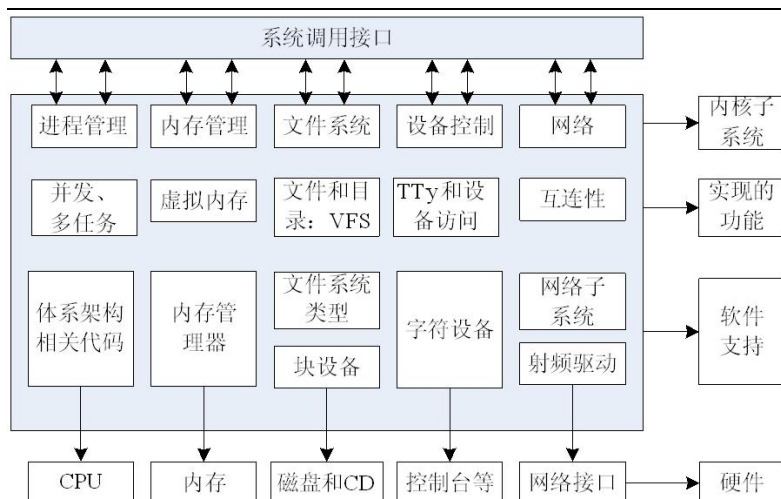
系统调用是操作系统提供给用户程序的一组“特殊”函数接口。

Linux 的不同版本提供了两三百个系统调用。

用户程序可以通过这组接口获得操作系统（内核）提供的服务。

例如：

用户可以通过文件系统相关的系统调用，请求系统打开文件、关闭文件或读写文件。



系统调用按照功能逻辑大致可分为:

进程控制、进程间通信、文件系统控制、系统控制、内存管理、网络管理、socket 控制、用户管理。

系统调用的返回值:

通常, 用一个负的返回值来表明错误, 返回一个 0 值表明成功。错误信息存放在全局变量 `errno` 中, 用户可用 `perror` 函数打印出错信息。

系统调用遵循的规范

在 Linux 中, 应用程序编程接口(API)遵循 POSIX 标准。

POSIX 标准基于当时现有的 UNIX 实践和经验, 描述了操作系统的系统调用编程接口(实际上就是 API), 用于保证应用程序可以在源代码一级上在多种操作系统上移植运行。

如:

linux 下写的 `open`、`write`、`read` 可以直接移植到 unix 操作系统下。

2.3 系统调用 I/O 函数

系统调用中操作 I/O 的函数, 都是针对文件描述符的

通过文件描述符可以直接对相应的文件进行操作。

如: `open`、`close`、`write`、`read`、`ioctl`

2.3.1 文件描述符

文件描述符是非负整数。打开现存文件或新建文件时, 系统(内核)会返回一个文件描述符。文件描述符用来指定已打开的文件。

```
#define STDIN_FILENO    0    //标准输入的文件描述符
#define STDOUT_FILENO   1    //标准输出的文件描述符
#define STDERR_FILENO   2    //标准错误的文件描述符
```

程序运行起来后这三个文件描述符是默认打开的。

2.3.2 open 函数

打开一个文件

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

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

```
#include <fcntl.h>
```

当文件存在时使用：

```
int open(const char *pathname, int flags);
```

当文件不存在时使用：

```
int open(const char *pathname, int flags, mode_t mode);
```

参数：

pathname：文件的路径及文件名。

flags：open 函数的行为标志。

mode：文件权限(可读、可写、可执行)的设置。

返回值：

成功返回打开的文件描述符。

失败返回-1，可以利用 **perror** 去查看原因。

flags 的取值及其含义	
取值	含义
O_RDONLY	以只读的方式打开
O_WRONLY	以只写的方式打开
O_RDWR	以可读、可写的方式打开
flags 除了取上述值外，还可与下列值位或	
O_CREAT	文件不存在则创建文件，使用此选项时需使用 mode 说明文件的权限
O_EXCL	如果同时指定了 O_CREAT，且文件已经存在，则出错
O_TRUNC	如果文件存在，则清空文件内容
O_APPEND	写文件时，数据添加到文件末尾
O_NONBLOCK	当打开的文件是 FIFO、字符文件、块文件时，此选项为非阻塞标志位

mode 的取值及其含义		
取值	八进制数	含义
S_IRWXU	00700	文件所有者的读、写、可执行权限
S_IRUSR	00400	文件所有者的读权限
S_IWUSR	00200	文件所有者的写权限
S_IXUSR	00100	文件所有者的可执行权限
S_IRWXG	00070	文件所有者同组用户的读、写、可执行权限

做真实的自己，用良心做教育

S_IRGRP	00040	文件所有者同组用户的读权限
S_IWGRP	00020	文件所有者同组用户的写权限
S_IXGRP	00010	文件所有者同组用户的可执行权限
S_IRWXO	00007	其他组用户的读、写、可执行权限
S_IROTH	00004	其他组用户的读权限
S_IWOTH	00002	其他组用户的写权限
S_IXOTH	00001	其他组用户的可执行权限

2.3.3 close 函数

关闭一个文件

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

```
int close(int fd);
```

参数:

fd 是调用 open 打开文件返回的文件描述符。

返回值:

成功返回 0。

失败返回-1，可以利用 perror 去查看原因。

2.3.4 write 函数

把指定数目的数据写到文件

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

```
ssize_t write(int fd, const void *addr, size_t count);
```

参数:

fd: 文件描述符。

addr: 数据首地址。

count: 写入数据的字节个数。

返回值:

成功返回实际写入数据的字节个数。

失败返回-1，可以利用 perror 去查看原因。

2.3.5 read 函数

把指定数目的数据读到内存

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

```
ssize_t read(int fd, void *addr, size_t count);
```

参数:

做真实的自己，用良心做教育

fd: 文件描述符。

addr: 内存首地址。

count: 读取的字节个数。

返回值:

成功返回实际读取到的字节个数。

失败返回-1, 可以利用 perror 去查看原因。

2.3.6 remove 库函数

删除文件

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

```
int remove(const char *pathname);
```

参数:

pathname : 文件的路名+文件名。

返回值:

成功返回 0。

失败返回-1, 可以利用 perror 去查看原因。

2.4 系统调用与库函数

库函数由两类函数组成

2.4.1 不需要调用系统调用

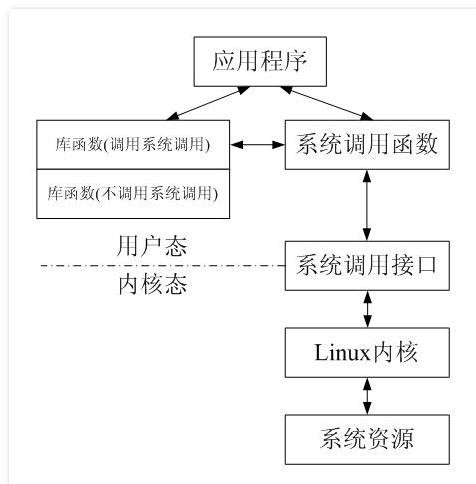
不需要切换到内核空间即可完成函数全部功能, 并且将结果反馈给应用程序, 如 strcpy、bzero 等字符串操作函数。

2.4.2 需要调用系统调用

需要切换到内核空间, 这类函数通过封装系统调用去实现相应功能, 如 printf、fread 等。

2.4.3 库函数与系统调用的关系:

并不是所有的系统调用都被封装成了库函数, 系统提供的很多功能都必须通过系统调用才能实现。

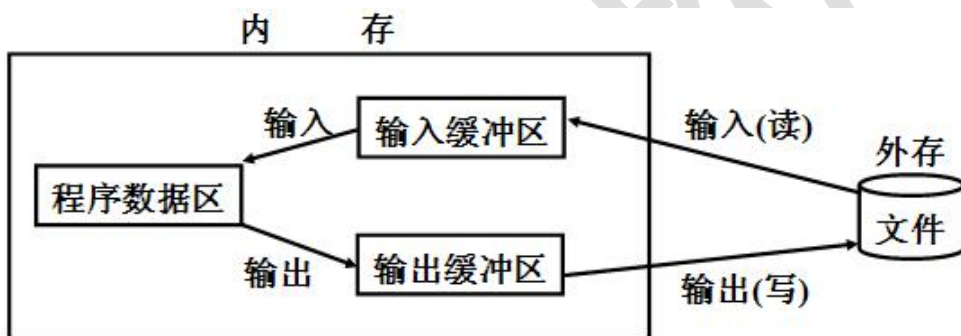


系统调用是需要时间的，程序中频繁的使用系统调用会降低程序的运行效率。

当运行内核代码时，CPU 工作在内核态，在系统调用发生前需要保存用户态的栈和内存环境，然后转入内核态工作。

系统调用结束后，又要切换回用户态。这种环境的切换会消耗掉许多时间。

库函数访问文件的时候根据需要，设置不同类型的缓冲区，从而减少了直接调用 IO 系统调用的次数，提高了访问效率。



应用程序调用 `printf` 函数时，函数执行的过程，如下图

