嵌入式系统工程师

系统调用与标准I/O库

大纲

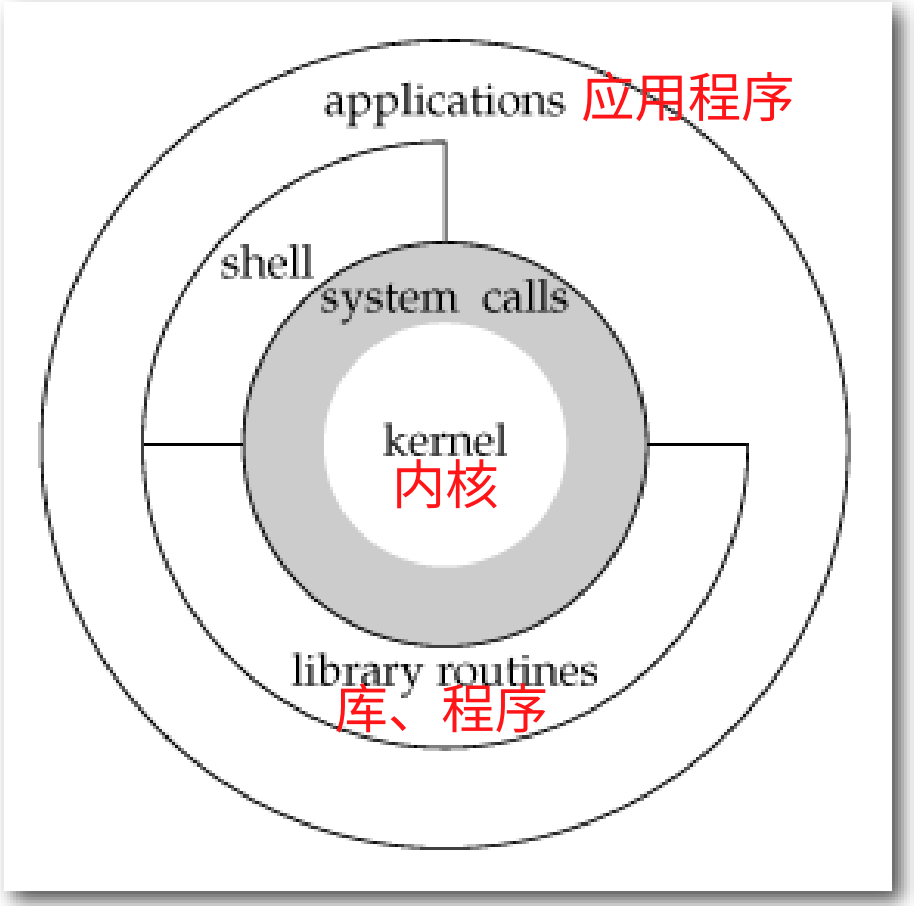
- 系统调用概述
- 系统调用I/O函数
- 系统调用与内核
- 系统调用与库
- 标准I/O库函数

大纲

- 系统调用概述
- 系统调用I/O函数
- 系统调用与内核
- 系统调用与库
- 标准I/O库函数

系统调用与内核

- 类UNIX系统的软件层次

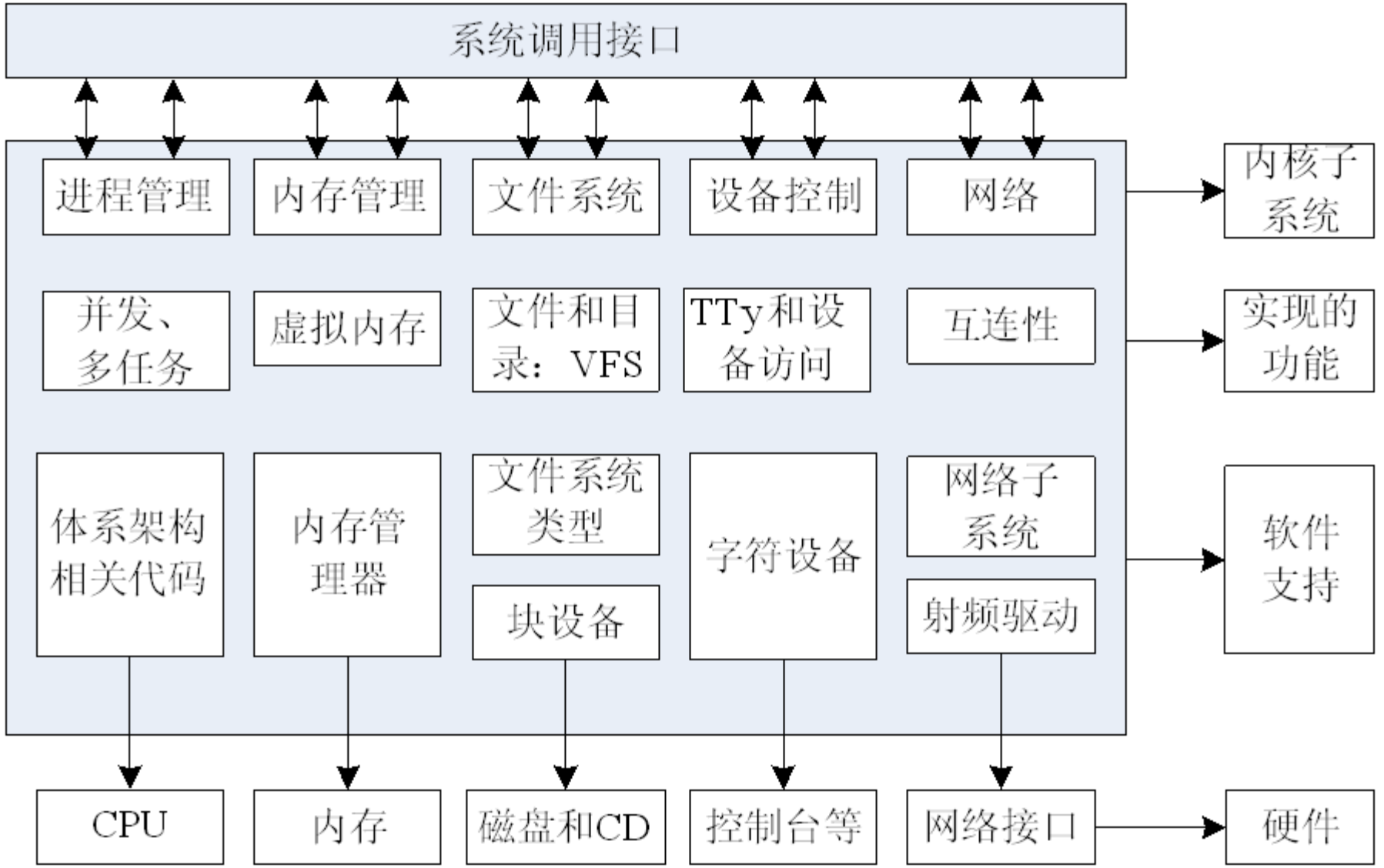


系统调用概述

- 系统调用是操作系统提供给用户程序的一组“特殊”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。

系统调用概述


系统调用接口
进程管理
内存管理
文件系统
设备控制
网络
并发、多任务
虚拟内存
文件和目录：VFS
TTy和设备访问
互连性
体系架构相关代码
内存管理器
文件系统类型
块设备
字符设备
网络子系统
射频驱动
CPU
内存
磁盘和CD
控制台等
网络接口
内核子系统
实现的功能
软件支持
硬件

系统调用概述

- 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。

- 系统调用通常通过函数进行调用。
- 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。

系统调用概述

- 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标准描述了操作系统的函数接口规范（函数名、返回值、参数等）。

  不同操作系统下编写的程序只要遵循POSIX标准，程序都可以直接移植。

  如:

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

大纲

- 系统调用概述
- 系统调用I/O函数
- 系统调用与内核
- 系统调用与库
- 标准I/O库函数

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

- 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。

系统调用I/O函数

- open函数：打开一个文件

```
#include <sys/types.h>   定义新的数据类型
#include <sys/stat.h>    文件信息结构体的定义
#include <fcntl.h>       声明系统调用

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

例：open ("/home/.../text",O_RDONLY)
或：char *p="/home/.../text";
    open (p, O_RDONLY);

系统调用I/O函数

- open函数：打开一个文件

  参数：

  pathname：文件的路径及文件名。

  flags：open函数的行为标志。以什么方式打开

  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

系统调用I/O函数

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
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，且为只读或只写的方式打开，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，此选项为阻塞标志位

系统调用I/O函数

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
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

系统调用I/O函数

- 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

系统调用I/O函数

- 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。

系统调用I/O函数

- 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：

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。

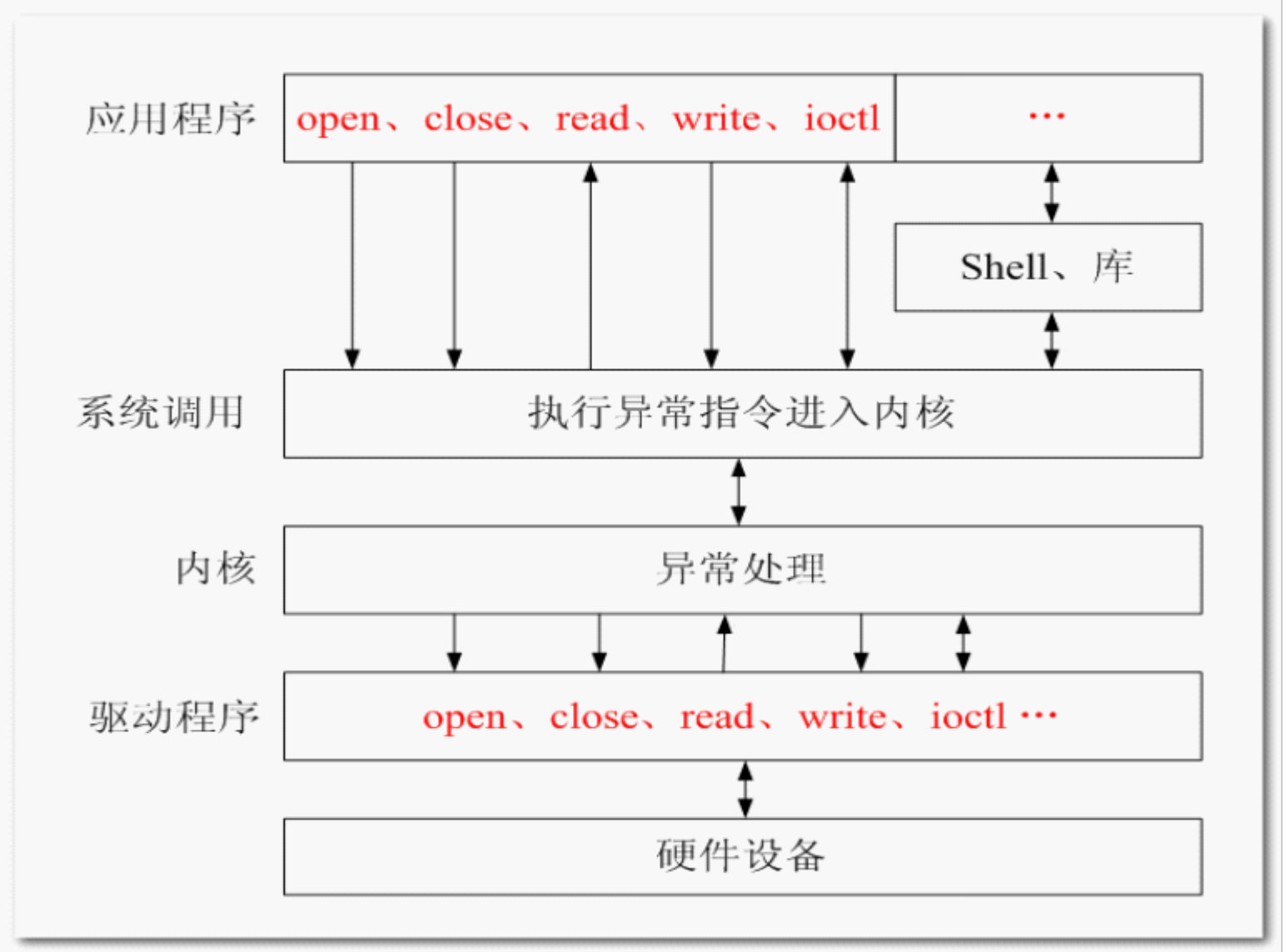
大纲

- 系统调用概述
- 系统调用I/O函数
- 系统调用与内核
- 系统调用与库
- 标准I/O库函数

系统调用与内核

- 为了更好地保护了内核，在Linux中，把程序运行空间分为内核空间和用户空间，它们分别运行在不同的级别上。
- 用户进程在通常情况下不允许访问内核数据，也无法使用内核函数。
- 但在有些情况下，用户空间的进程需要获得一定的系统服务，这时，就必须通过系统调用。

系统调用与内核


应用程序
open、close、read、write、ioctl
…
Shell、库
系统调用
执行异常指令进入内核
内核
异常处理
驱动程序
open、close、read、write、ioctl …
硬件设备

系统调用与内核

- 应用程序运行在用户空间，系统调用需要切换到内核空间，应用程序应该以某种方式通知内核需要切换到内核空间。
- 通知内核的机制是靠软件中断实现的:

  应用程序执行异常指令，引发一个异常，程序进入中断，系统切换到内核态去执行异常处理程序。

  此处的异常处理程序即系统调用处理程序syscall()。
- 所有的系统调用陷入内核的方式都一样，所以仅仅是陷入内核空间是不够的。必须以某种方式通知内核进入异常的原因。

系统调用概述

- Unix系统通过系统调用号通知内核进入异常的原因。
- 系统调用号

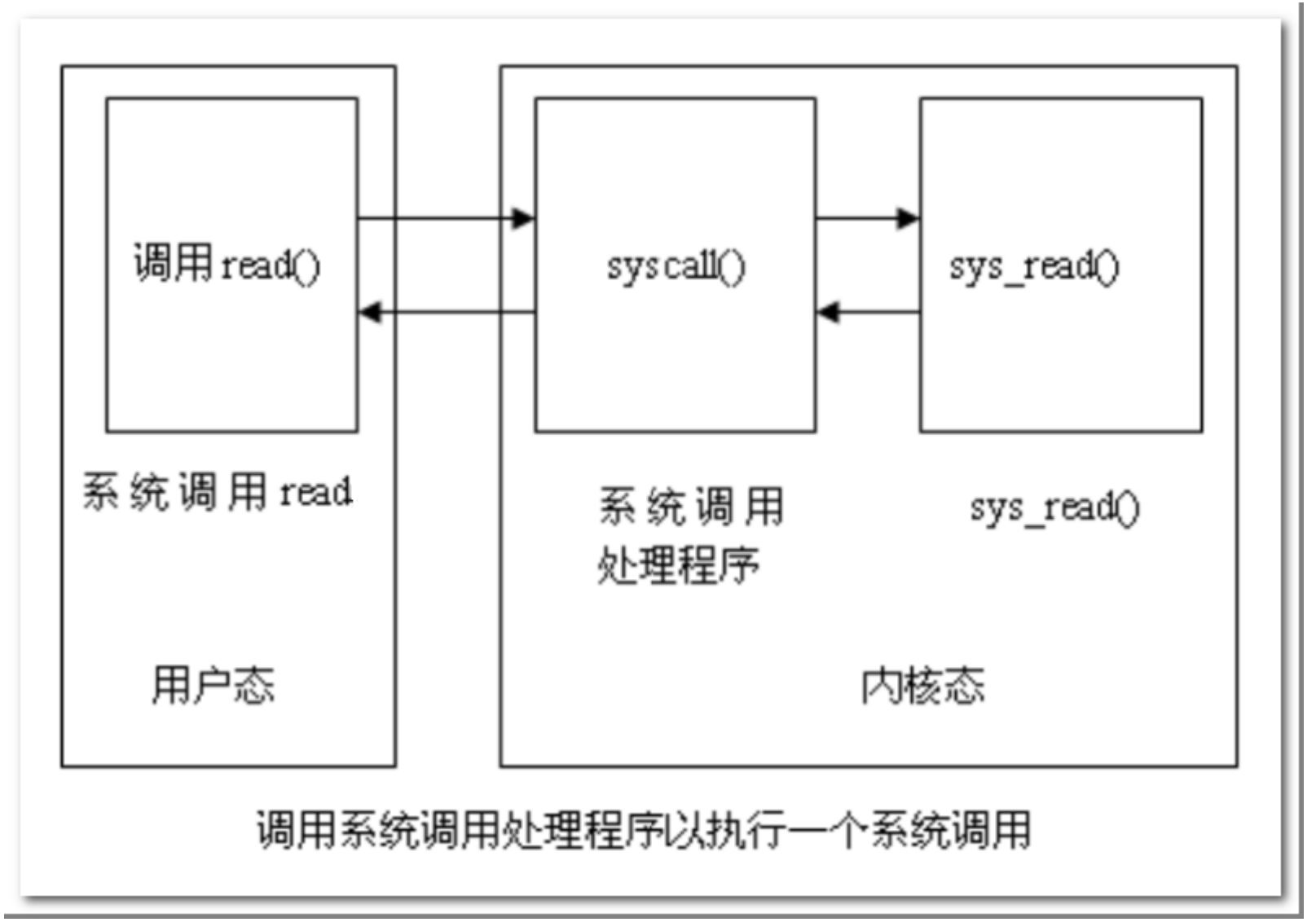
  操作系统给每个系统调用分配了一个唯一的编号，这个编号就是系统调用号。

  用户空间进程执行一个系统调用时，这个系统调用号就被用来指明执行哪个系统调用。

  系统调用号相当关键，一旦分配就不能再有任何变更，否则编译好的应用程序会崩溃。此外，如果一个系统调用被删除，它所占用的系统调用号也不允许被回收利用。
- 路径:

  /usr/include/i386-linux-gnu/asm/unistd_32.h

系统调用与内核



调用系统调用处理程序以执行一个系统调用

大纲

- 系统调用概述
- 系统调用I/O函数
- 系统调用与内核
- 系统调用与库
- 标准I/O库函数

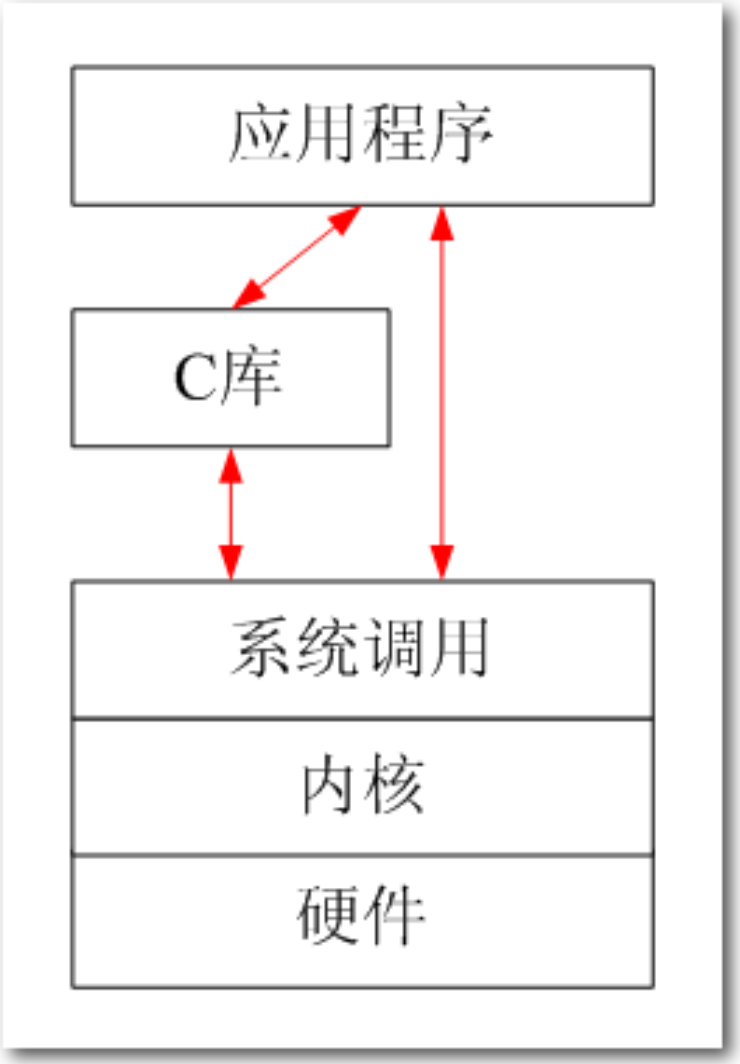
系统调用与库

- 库函数由两类函数组成
  - 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。
  - 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。

系统调用与库



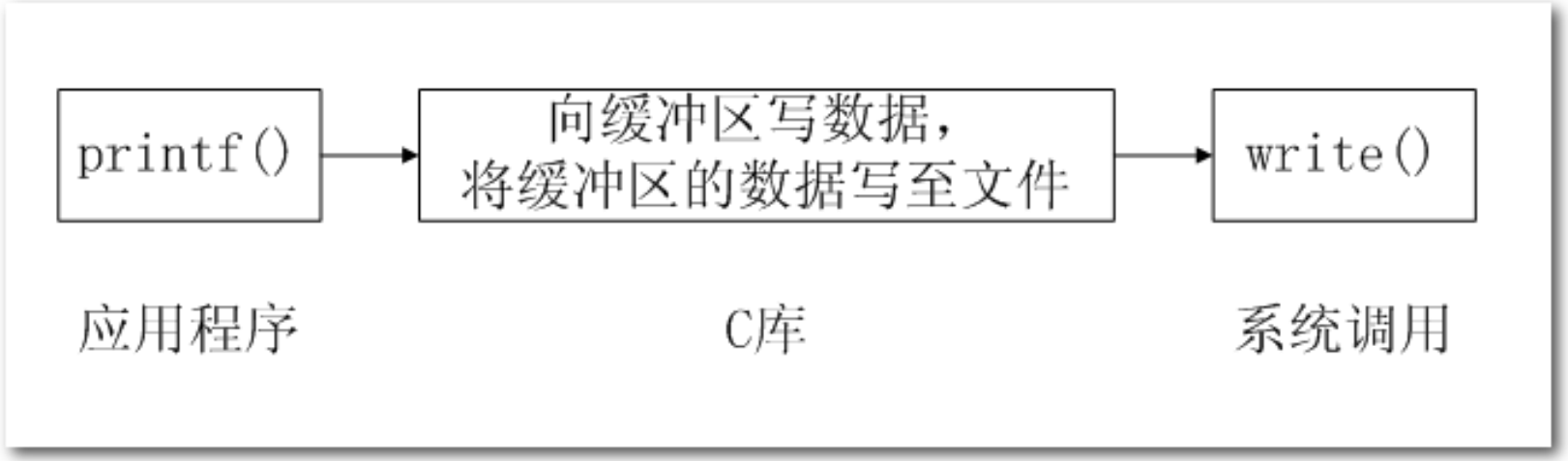
- 库函数与系统调用的关系：

  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。

系统调用与库

- 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，然后转入系统态工作。
  - 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：



大纲

- 系统调用概述
- 系统调用I/O函数
- 系统调用与内核
- 系统调用与库
- 标准I/O库函数

标准I/O库函数

- 无论是编写系统程序还是应用程序，都离不开I/O这个重要的环节。
- 相对于低级的I/O操作(即系统调用级的I/O)，标准I/O库函数处理了很多细节，如缓存分配等。
- 考虑到代码的可移植性，开发人员应该在编写代码时尽可能使用标准库函数。

标准I/O库函数

- I/O的管理分类
  - 由ANSI标准提供的标准IO库函数

    几乎被所有的操作系统支持，如winsdows下编写的程序几乎不用做任何修改就可以在linux下重新编译运行。

    如：fopen、fread、fwrite、fclose。
  - 以系统调用的方式给用户提供函数接口(遵循POSIX标准)

    例如linux操作系统提供的文件IO接口。

    如：open、close、read、write、ioctl。

    系统调用与操作系统直接相关，直接使用系统调用编写的程序的可移植性差。

标准I/O库函数

- 头文件<stdio.h>中声明了标准C的I/O库，标准C的I/O库在所有通用计算机上的C语言实现都是相同的。
- 对于标准I/O操作函数来说，打开或创建一个文件的时候，会返回一个指向FILE结构体的指针。
- FILE结构体包含了I/O函数库为管理文件所需要的尽可能多的信息。包括了用于I/O文件的文件描述符、指向流缓存的指针、缓存长度等。
- 定义路径：/usr/include/libio.h
- 别名(typedef)：/usr/include/stdio.h

标准I/O库函数

- 打开流

头文件: #include <stdio.h>

定义函数:

```
FILE* fopen(const char *pathname,
                                const char *mode);
```

函数说明:

pathname: 文件的路径及文件名。

mode: 流的打开方式。

返回值:

成功: 返回指向该流的指针。

失败: 则返回NULL, 并把错误代码存在errno中

标准I/O库函数

模　式	功　能
r或rb	以只读方式打开一个文本文件（不创建文件）
w或wb	以写方式打开文件（使文件长度截断为0字节，创建一个文件）
a或ab	以添加方式打开文件，即在末尾添加内容，当文件不存在时，创建文件用于写
r+或rb+	以可读、可写的方式打开文件(不创建新文件)
w+或wb+	以可读、可写的方式打开文件 （使文件长度为0字节，创建一个文件）
a+或ab+	以添加方式打开文件，打开文件并在末尾更改文件 （如果文件不存在，则创建文件）

标准I/O库函数

- 关闭流

头文件: #include <stdio.h>

定义函数:

int fclose(FILE *stream);

函数说明:

fclose用来关闭fopen打开的文件。此动作会让缓冲区的数据写入文件中，并释放系统所提供的文件资源。

返回值:

成功返回0; 失败返回EOF，并把错误代码存到errno中。

标准I/O库函数

- 读、写流

  打开了流后，对其进行读写操作的方法:

  - 每次一个字符
  - 每次一行字符
  - 每次一个数据块

标准I/O库函数

- 每次一个字符

```
int getchar(void);
int getc(FILE *stream);
int fgetc(FILE *stream);

int putchar(int c);
int putc(int c, FILE *stream);
int fputc(int c, FILE *stream);
```

标准I/O库函数

- 每次一行字符

```
char *gets(char *buf);
char *fgets(char *buf, int n, FILE *stream);
```

fgets从stream指定的文件中最多读取n-1个字符放到buf所指向的数组中。读到换行符或文件结束符后不再向后读，最后一个字符读入后接着写入一个空字符。

返回值:

成功返回buf；失败返回NULL

注意:

gets()丢掉输入里的换行符。

fgets()存储输入中的换行符

标准I/O库函数

- 每次一行字符

```
int puts(const char *str);
int fputs(const char *str, FILE *stream);
```

fputs函数将字符串写入stream指定的文件中，终止字符串的空字符不写入。

返回值:

成功返回非负数；失败返回EOF

注意:

puts()为输出添加换行符。

fputs()不为输出添加换行符。

标准I/O库函数

- 每次一个数据块

```
size_t fread(void *ptr, size_t size,
                        size_t nobj, FILE *stream);
size_t fwrite(const void *ptr, size_t size,
                        size_t nobj, FILE *stream);
```

ize是数据块大小，nobj指要读取或写入的数据块个数，stream指定要操作的数据流。

注意:

两个函数返回的是实际读或写的数据的个数，而不是整个数据的字节数。