

# Linux系统编程-目录

2024年5月18日 11:16

## 一、基本概念

1.1 操作系统

1.2 库函数和系统调用

1.3 内核

1.4 程序和进程

1.5 虚拟内存

# 基本概念

2024年5月18日 11:17

## 一、基本概念

- 1.1 操作系统
- 1.2 库函数和系统调用
- 1.3 内核
- 1.4 程序和进程
- 1.5 虚拟内存

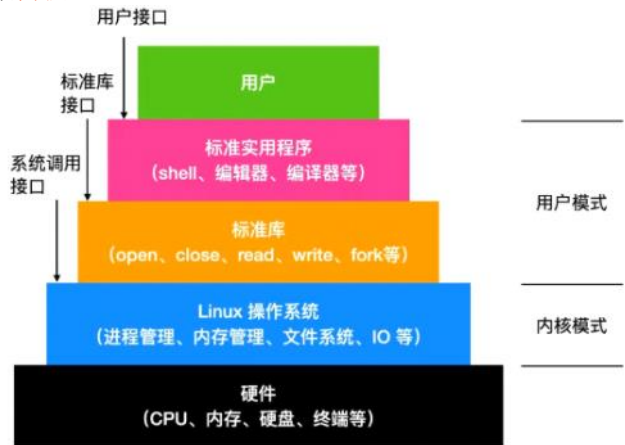
## 一、 基本概念

### 1.1 操作系统

操作系统（operating system, OS）：是管理硬件资源和软件资源的计算机程序。

操作系统通俗解释：计算机是由各种硬件组成，如果没有操作系统，开发人员就需要熟悉对所有计算机硬件的使用，掌握所有硬件的细节，这样程序员就不用写代码了。所以就给计算机安装了一个软件，称为操作系统，能为用户程序提供一个更简单，方便的计算机模型。本质上操作系统也是一个程序，最大的作用就是管理硬件资源。

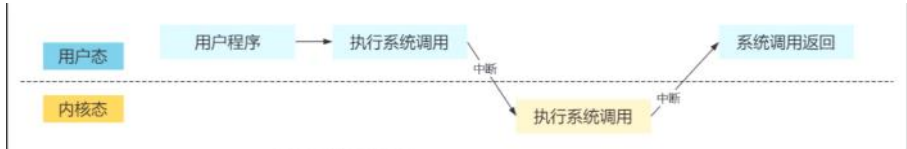
其中管理硬件资源的程序叫做**内核**。



因此，操作系统一般分为两种模式：**用户模式**（用户态）、**内核模式**（内核态）  
一般情况下，内核态可以访问硬件资源，用户态不能访问硬件资源。

例如：printf() 函数在用户态下是无法打印输出的，在函数内部进行了系统调用write() 函数，进入了内核态

可以通过**系统调用**进行内核态。



流程：用户利用程序（开发工具）调用库函数（printf），库函数内部进行系统调用，然后进入内核态，内核态下执行系统调用，返回系统调用，回到用户态。

### 1.2 库函数和系统调用

本质上都是函数

**库函数**：是把函数放到库里，方便其他人使用。目的就是把常用的一些函数放到一个文件了，方便重复使用。一般放在lib文件中。

**系统调用：**是内核的入口，是为了能够进入内核的一个函数。如果需要进入内核空间，就需要通过系统调用，当程序进行系统调用时，会产生一个中断，CPU会中断当前执行的应用程序，跳转到中断处理程序，也就是开始执行内核程序，内核处理完成后，主动触发中断，把CPU执行权交回应用程序。

使用man手册查看系统调用

```
1 可执行程序或 shell 命令
2 系统调用(内核提供的函数)
3 库调用(程序库中的函数)
4 特殊文件(通常位于 /dev)
5 文件格式和规范, 如 /etc/passwd
6 游戏
7 杂项 (包括宏包和规范), 如 man(7), groff(7), man-pages(7)
8 系统管理命令(通常只针对 root 用户)
Manual page man(1) line 1 (press h for help or q to quit)
```

补全man手册

```
sudo apt-get install manpages-dev glibc-doc manpages-posixmanpages-posix-dev
```

### 1.3 内核

内核是操作系统的核心，是应用程序与硬件直接的桥梁，应用程序只关心与内核的交互，不用关系硬件的细节

**内核的能力：**

- (1) 进程调度：管理进程、线程，决定哪个进程、线程使用CPU
- (2) 内存管理：决定内存的分配和回收
- (3) 提供文件系统：提供文件系统，允许对文件进行创建、获取、更新以及删除等操作。
- (4) 管理硬件设备：为进程与硬件设备直接提供通信能力
- (5) 提供系统调用接口：进程可以通过内核入口（也就是系统调用），请求去内核执行各种任务。

内核具有很高的权限，可以控制CUP、内存、硬盘等硬件。而应用程序权限很小，因此大多数操作系统一般将内存分为两个区域：

**内核空间：**只有内核程序才能访问

**用户空间：**专门给应用程序使用

因此CUP可以在两种状态下运行：用户态、内核态，  
用户态下的CPU，只能访问用户空间的内存  
内核态下，内核空间内存与用户空间内存都能访问

### 1.4 程序和进程

**程序：**所有的程序都必须能运行。使用的软件就是程序，或者是自己写的代码经过编译和链接处理，得到计算机能够理解 and 执行的指令（可执行文件）。

**进程：**进程是正在执行程序实例。是操作系统进行资源分配和调度的基本单位。在内核看来，进程是一个个实体，内核需要在他们之间共享各种计算机资源。例如内存资源：内核会在程序开始时为进程分配一定的内存空间，并统筹该进程和整个系统对内存的需求。程序终止时，内核会是否该进程的所有资源，以供其他进程使用。

程序运行起来之后，就叫做进程（运行起来的程序）。

### 1.5 虚拟内存

内核是通过**虚拟内存**来确保进程只访问自己的内存空间的。

进程中，只能访问虚拟内存，操作系统会把虚拟内存翻译为真实的内存地址。这种内存管理方式叫做虚拟内存。

操作系统会给每个进程分配一套独立的虚拟地址。每个进程访问自己的虚拟内存，互不干涉，操作系统会提供虚拟内存和物理地址的映射机制。

段式分配内存：外部内存碎片  
页式分配内存：内部内存碎片

## 1.6 并发与并行

假设电脑里只有一个CPU、并且只有一个核心（core）

并行：是同时进行的。

并发：不是同时进行的，虽然在一个时间段内是一起完成的，但在某一时刻只能有一个进程在执行。

微观上：CPU在进程上来回切换（进程1在阻塞时，就会去执行进程2）

宏观上：多个进程都在运行。

## 二、 文件I/O

### 2.1 文件描述符

打开的程序，使用进程来描述；而打开的文件，使用文件描述符来描述。（非负整数，从0开始，依次递增）

但一个程序打开之后，默认就有三个文件描述符（0：标准输入、1：标准输出、2：标准错误）

所以，在程序中打开的第一个文件的文件描述符应该是3。

### 2.2 打开关闭文件（open和close函数）

#### 1、open函数

使用man 2 open 查看 open函数的定义。

```
SYNOPSIS
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

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

**pathname:** 要打开的文件路径

**flags:** 文件访问模式，有一系列常数值可供选择，可同时选择多个，用按位或（|）连接起来。

**必选项:** 以下三个常数中必须指定一个，且仅允许指定一个

**O\_RDONLY:** 只读打开

**O\_WRONLY:** 只写打开

**O\_RDWR:** 可读可写打开

常用可选项: 可以同时指定0个或多个，和必选项按位或起来使用。

**O\_APPEND:** 追加，如果文件已有内容，这次打开文件所写的数据附加到文件的末尾而不覆盖原来的内容，

**O\_CREAT:** 若此文件不存在则创建它。使用此选项时需要提供第三个参数 mode，表示该文件的访问权限。注: 文件最终权限: mode & ~umask

**O\_EXCL:** 如果同时指定了O\_CREAT，并且文件已存在，则出错返回，

**O\_TRUNC:** 如果文件已存在，将其长度截断为0字节，

**O\_NONBLOCK:** 设置非阻塞模式:

普通文件默认是非阻塞的，内核缓冲区保证了普通文件 I/O 不会阻塞。打开普通文件一般会忽略该参数  
设备、管道和套接字默认是阻塞的，以O\_NONBLOCK方式打开可以做非阻塞I/O (Nonblock I/O)。

**mode:** 创建文件时设置权限

**函数返回值:**

成功: 返回一个最小且未被占用的文件描述符

失败: 返回 -1，并设置 errno 值，

文件描述符指的是打开的文件，而不是文件，文件描述符是由进程维护的。

```
int fd = open("./c.txt", O_RDONLY | O_CREAT, 0644);
```

创建打开c.txt文件，不存在就创建一个，权限为0644（8进制），给的是 用户、用户组、其他人的权限

```
int fd = open("./c.txt", O_RDONLY | O_CREAT, 0666);
```

无法给到其他人读写权限，系统有默认权限，其他人给不到写权限。系统会给一个权限掩码，防止给太高的权限

使用umask即可查看权限掩码

```
● weihong@weihong:~/linux/Day_517$ umask
0002
```

系统会将其二进制 0000 0010 取反 1111 1101 与 你给的 权限进行 按位与 &  
0000 0110  
1111 1101  
=0000 0100

## 2、close函数

```
// close 函数 -- 关闭文件
// int close(int fd);
//
// fd 文件描述符
//
// 函数返回值:
//    成功返回 0
//    失败返回 -1, 并设置 errno 值。
//
```

不使用close函数，在进程结束之后也会关闭文件，因为文件描述符是由进程维护的，进程关了，文件也就关了

## 2.3 文件读写（read和write函数）

### 1、读文件— read函数

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

**函数参数:**

fd: 文件描述符  
buf: 读取的数据保存在缓冲区 buf 中 -- 传出参数  
count: buf 缓冲区存放的最大字节数

**函数返回值:**

>0: 读取到的字节数  
=0: 文件读取完毕  
-1: 出错，并设置 errno

count 应该是 大于等于 ssize\_t

### 2、写文件— write函数

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

**函数参数:**

fd: 文件描述符  
buf: 缓冲区，要写入文件或设备的数据  
count: buf中数据的长度

**函数返回值:**

成功: 返回写入的字节数  
错误: 返回-1 并设置 errno

`ssize_t` 等于 `count`

相同的文件描述符就是追加，不同的文件描述符就是覆盖