Linux系统编程-目录

2024年5月18日 11:16

基本概念

2024年5月18日　　11:17

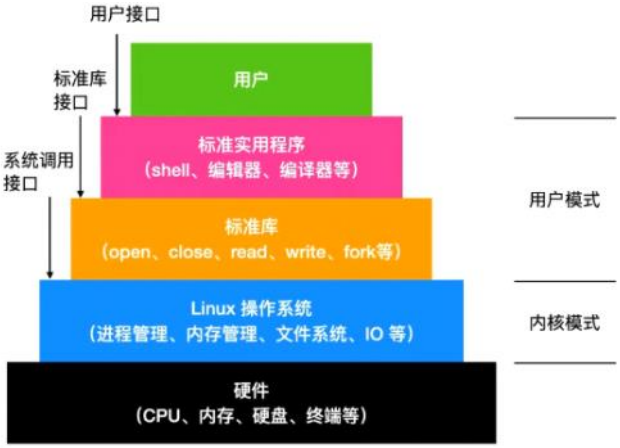
一、 基本概念

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。

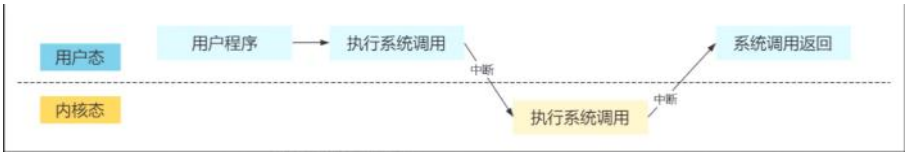


因此，操作系统一般分为两种模式：用户模式（用户态）、内核模式（内核态）

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

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和执行的指令（可执行文件）。

进程：进程是正在执行程序的实例。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。

文件IO

2024年5月27日 15:42

二、 文件IO

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(Nonblock1/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 fdc = 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 fdc = 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

2.4 lseek（文件偏移量）

所有打开的文件都有一个当前文件偏移量(currentfile ofset)，也叫读写偏移量和指针文件偏移量通常是一个非负整数，用于表明文件开始处到文件当前位置的字节数(下一个read()或write()操作的文件起始位置)。文件的第一个字节的偏移量为 0。文件打开时，会将文件偏移量设置为指向文件开始(使用O_APPEND 除外)，以后每次read()和write()会自动对其调整，以指向已读或已写数据的下一字节。因此连续的read()和write()将按顺序递进，对文件进行操作。使用 lseek 函数可以改变文件的偏移量。

```
off_t lseek(int fd, off_t offset, int whence);
```

函数参数:

fd:文件描述符
offset:字节数，以 whence 参数为基点解释 offset
whence:解释 offset 参数的基点
    SEEK SET:文件偏移量设置为 offset
    SEEK CUR:文件偏移量设置为当前文件偏移量加上offset，offset可以为负数
    SEEK END:文件偏移量设置为文件长度加上 offset，offset可以为负数

函数返回值:

若lseek 成功执行，则返回新的偏移量。
失败返回 -1 并设置 errno

Iseek 函数常用操作:

文件指针移动到头部
```
lseek(fd, 0, SEEK_SET);
```
获取文件指针当前位置
```
int len = lseek(fd,0,SEEK CUR);
```
获取文件长度
```
int len = lseek(fd,0,SEEK END);
```
lseek 实现文件拓展
```
lseek(fd,n,SEEK_END);// 扩展n个字节
write();// 扩展后需要执行一次写操作才能扩展成功
```

练习--实现自己的cp函数 -- mycp

```
int main(int argc, char* argv[])
{
    if(argc!=3)
    {
        printf("请输入正确的参数数量: 源文件、目标文件\n");
    }
    // 打开两个文件
    int fd1 = open(argv[1],O_RDONLY);
    // 写入、创建、清空(截断)
    int fd2 = open(argv[2],O_WRONLY|O_CREAT|O_TRUNC,0644);

    // 读取第一个文件的内容
```

```
    char buf[10];
    while(1) // 循环读取
    {
        int read_count = read(fd1,buf,sizeof(buf));
        if(read_count == 0)
        {
            break;
        }
        // 将读取的内容写入第二个文件
        int write_count = write(fd2,buf,read_count);
        printf("write_count = %d\n",write_count); // 打印写入字节
    }

    // 关闭两个文件
    close(fd1);
    close(fd2);
    return 0;
}
```

2.5 阻塞、非阻塞

这里的阻塞、非阻塞是只文件的，而不是进程。该县设置

进程通信

2024年5月28日 20:05

四、进程通信

4.1 管道

4.1.1 匿名管道

文件名匿名

匿名管道只能在有血缘关系的进程之间通讯

命名管道

匿名管道只能用于具有共同祖先的进程(具有亲缘关系的进程)之间的通信，通常，一个管道由一个进程创建，然后该进程调用fork，此后父子进程之间就可应用该管道。如果要实现两个毫不相关进程之间的通信，可以使用命名管道来做到。命名管道就是一种特殊类型的文件，两个进程通过命名管道的文件名打开同一个管道文件，此时这两个进程也就看到了同一份资源，进而就可以进行通信了。命名管道和匿名管道一样，都是内存文件，只不过命名管道在磁盘有一个简单的映像，但这个映像的大小永远为 0，因为命名管道和匿名管道都不会将通信数据刷新到磁盘当中。