UC Day06

预习课

预习内容

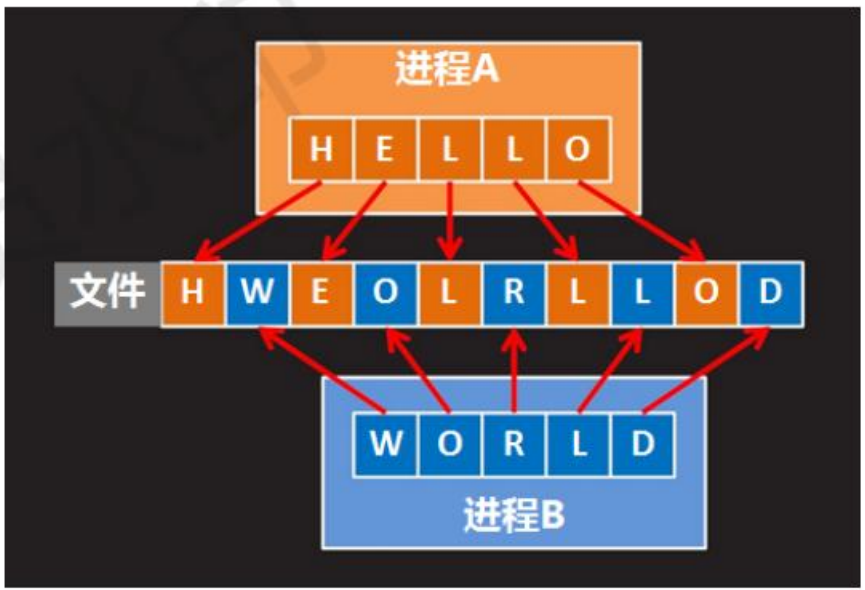
文件的读写冲突

文件的读写冲突

文件的读写冲突

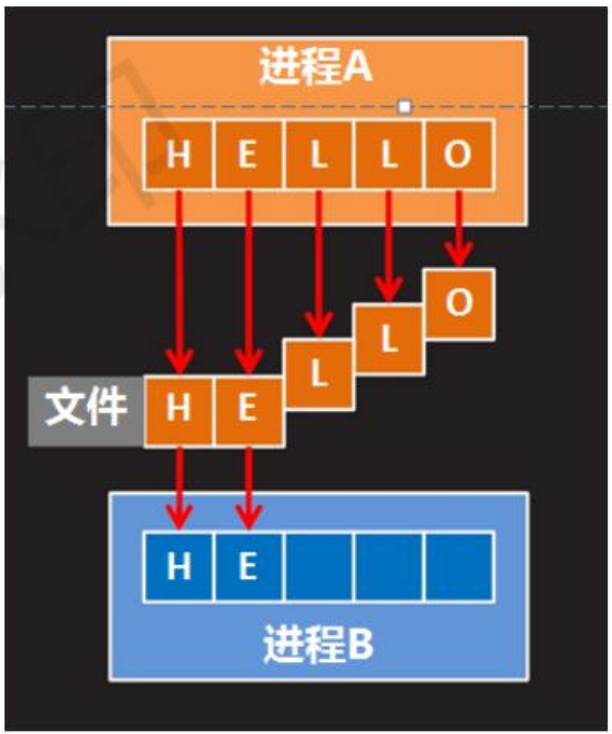
知识讲解

- 如果两个或两个以上的进程同时向一个文件的某个特定区域写入数据，那么最后写入文件的数据极有可能因为写操作的交错而产生混乱



文件的读写冲突

- 如果一个进程写而其它进程同时在读一个文件的某个特定区域，那么读出的数据极有可能因为读写操作的交错而不完整



知识讲解

文件的读写冲突

知识讲解

- 多个进程同时读一个文件的某个特定区域，不会有任何问题，它们只是各自把文件中的数据拷贝到各自的缓冲区中，并不会改变文件的内容，相互之间也就不会冲突
- 由此可以得出结论，为了避免在读写同一个文件的同一个区域时发生冲突，进程之间应该遵循以下规则，如果一个进程正在写，那么其它进程既不能写也不能读，如果一个进程正在读，那么其它进程不能写但是可以读

文件锁

文件锁

知识讲解

- 为了避免多个进程在读写同一个文件的同一个区域时发生冲突，Unix/Linux系统引入了文件锁机制，并把文件锁分为读锁和写锁两种，它们的区别在于，对一个文件的特定区域可以加多把读锁，对一个文件的特定区域只能加一把写锁
- 基于锁的操作模型是：读/写文件中的特定区域之前，先加上读/写锁，锁成功了再读/写，读/写完成以后再解锁

文件锁

- 文件锁的加锁操作

知识讲解

文件的某个区域当前拥有锁	期望加锁	
	读锁	写锁
无任何锁	OK	OK
多把读锁	OK	NO
一把写锁	NO	NO

文件锁

- 假设进程A期望访问某文件的A区，同时进程B期望访问该文件的B区，而A区和B区存在部分重叠，分情况讨论
  - 第一种情况：进程A正在写，进程B也想写

进程A	进程B
打开文件，准备写A区	打开文件，准备写B区
给A区加写锁，成功	
写A区	给B区加写锁，失败，阻塞
写完，解锁A区	从阻塞中恢复，B区被加上写锁
	写B区
	写完，解锁B区
关闭文件	关闭文件

文件锁

- 假设进程A期望访问某文件的A区，同时进程B期望访问该文件的B区，而A区和B区存在部分重叠，分情况讨论
  - 第二种情况：进程A正在写，进程B却想读

进程A	进程B
打开文件，准备写A区	打开文件，准备读B区
给A区加写锁，成功	
写A区	给B区加读锁，失败，阻塞
写完，解锁A区	从阻塞中恢复，B区被加上读锁
	读B区
	读完，解锁B区
关闭文件	关闭文件

知识讲解

文件锁

- 假设进程A期望访问某文件的A区，同时进程B期望访问该文件的B区，而A区和B区存在部分重叠，分情况讨论
  - 第三种情况：进程A正在读，进程B却想写

进程A	进程B
打开文件，准备读A区	打开文件，准备写B区
给A区加读锁，成功	
读A区	给B区加写锁，失败，阻塞
读完，解锁A区	从阻塞中恢复，B区被加上写锁
	写B区
	写完，解锁B区
关闭文件	关闭文件

知识讲解

文件锁

- 假设进程A期望访问某文件的A区，同时进程B期望访问该文件的B区，而A区和B区存在部分重叠，分情况讨论
  - 第四种情况：进程A正在读，进程B也想读

进程A	进程B
打开文件，准备读A区	打开文件，准备读B区
给A区加读锁，成功	
读A区	给B区加读锁，成功
	读B区
读完，解锁A区	
	读完，解锁B区
关闭文件	关闭文件

知识讲解

直播课见

UC

C/C++教学体系

目录

文件锁

文件锁

- #include <fcntl.h>
- int fcntl(int fd, F_SETLK/F_SETLKW，struct flock* lock);
  - 功能：加解锁
  - 参数：F_SETLK 非阻塞模式加锁，F_SETLKW 阻塞模式加锁
    lock 对文件要加的锁
  - 返回值：成功返回0,失败返回-1

知识讲解

文件锁

知识讲解

- struct flock {
  short l_type; // 锁类型：F_RDLCK/F_WRLCK/F_UNLCK
  short l_whence; // 锁区偏移起点：SEEK_SET/SEEK_CUR/SEEK_END
  off_t l_start; // 锁区偏移字节数
  off_t l_len; // 锁区字节数
  pid_t l_pid; // 加锁进程的PID，-1表示自动设置
};
- 通过对该结构体类型变量的赋值，再配合fcntl函数，以完成对文件指定区域的加解锁操作

文件锁

知识讲解

- 几点说明：
  - 当通过close函数关闭文件描述符时，调用进程在该文件描述符上所加的一切锁将被自动解除
  - 当进程终止时，该进程在所有文件描述符上所加的一切锁将被自动解除；
  - 文件锁仅在不同进程之间起作用，同一个进程的不同线程不能通过文件锁解决读写冲突问题；
  - 通过fork/vfork函数创建的子进程，不继承父进程所加的任何文件锁；
  - 通过exec函数创建的新进程，会继承原进程所加的全部文件锁，除非某文件描述符带有FD_CLOEXEC标志。

文件锁

- 从前述基于锁的操作模型可以看出，锁机制之所以能够避免读写冲突，关键在于参与读写的多个进程都在按照一套模式——先加锁，再读写，最后解锁——按部就班地执行。这就形成了一套协议，只要参与者无一例外地遵循这套协议，读写就是安全的。反之，如果哪个进程不遵守这套协议，完全无视锁的存在，想读就读，想写就写，即便有锁，对它也起不到任何约束作用。因此，这样的锁机制被称为劝谏锁或协议锁

知识讲解

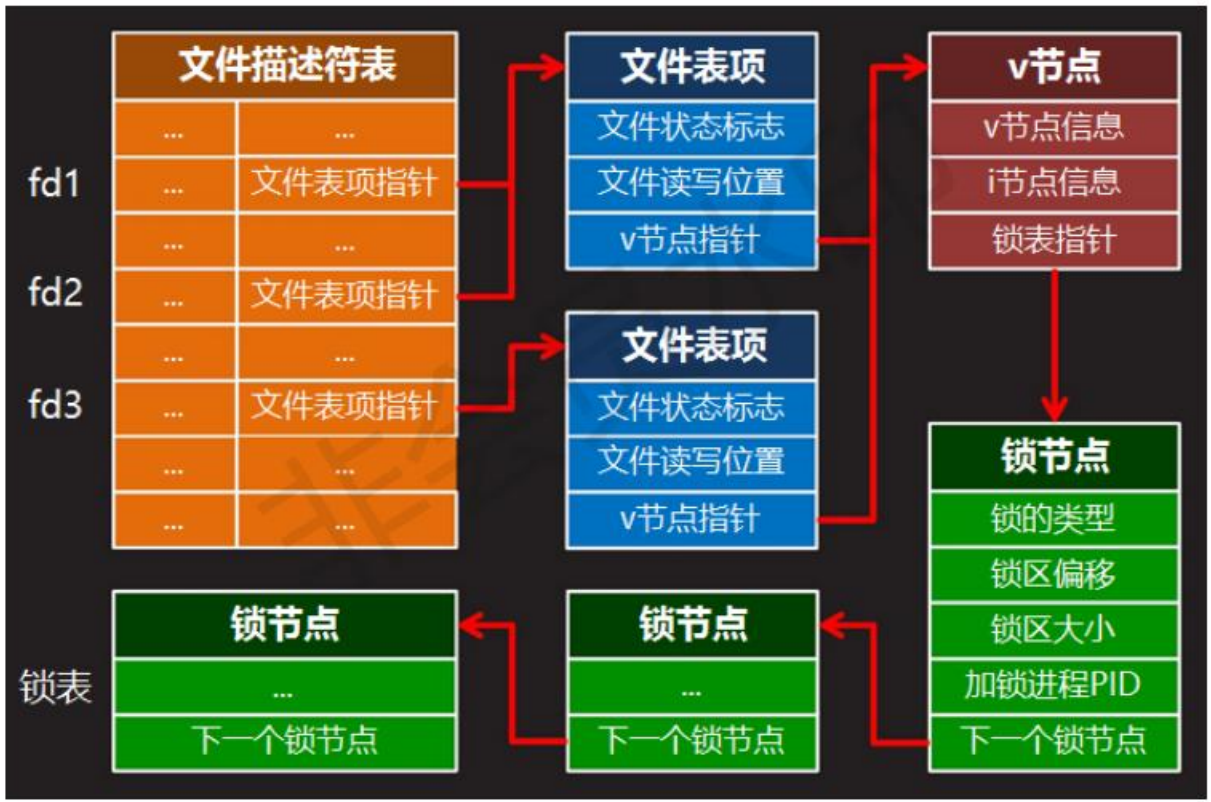
文件锁的内核结构

文件锁的内核结构

知识讲解

- 每次对给定文件的特定区域加锁，都会通过fcntl函数向系统内核传递flock结构体，该结构体中包含了有关锁的一切细节，诸如锁的类型(读锁/写锁)，锁区的起始位置和大小，甚至加锁进程的PID(填-1由系统自动设置)
- 系统内核会收集所有进程对该文件所加的各种锁，并把这些flock结构体中的信息，以链表的形式组织成一张锁表，而锁表的起始地址就保存在该文件的v节点中
- 任何一个进程通过fcntl函数对该文件加锁，系统内核都要遍历这张锁表，一旦发现有与欲加之锁构成冲突的锁即阻塞或报错，否则即将欲加之锁插入锁表，而解锁的过程实际上就是调整或删除锁表中的相应节点

文件锁的内核结构


文件描述符表
fd1
fd2
fd3
文件表项指针
文件表项指针
文件表项指针
文件表项
文件状态标志
文件读写位置
v节点指针
文件表项
文件状态标志
文件读写位置
v节点指针
v节点
v节点信息
i节点信息
锁表指针
锁节点
锁的类型
锁区偏移
锁区大小
加锁进程PID
下一个锁节点
锁表
锁节点
下一个锁节点
锁节点
下一个锁节点

文件的元数据

文件锁的元数据

知识讲解

- #include <sys/stat.h>
- int stat(char const* path, struct stat* buf);
- int fstat(int fd, struct stat* buf);
- int lstat(char const* path, struct stat* buf);
  - 功能：从i节点中提取文件的元数据，即文件的属性信息
  - 参数：path　文件路径
    buf　文件元数据结构
    fd　文件描述符
  - 返回值：成功返回0，失败返回-1

文件锁的元数据

- lstat()函数与另外两个函数的区别在于它不跟踪符号链接。
  - 例:
  - abc.txt ---> xyz.txt abc.txt文件是xyz.txt文件的符号链接
  - stat(“abc.txt”, ...); //得到xyz.txt文件的元数据
  - lstat(“abc.txt”, ...); //得到abc.txt文件的元数据

知识讲解

文件锁的元数据

- stat函数族通过stat结构体，向调用者输出文件的元数据
  - –struct stat {

```
dev_t     st_dev;       // 设备ID
ino_t     st_ino;       // i节点号
mode_t    st_mode;      // 文件的类型和权限
nlink_t   st_nlink;     // 硬链接数
uid_t     st_uid;       // 拥有者用户ID
gid_t     st_gid;       // 拥有者组ID
dev_t     st_rdev;      // 特殊设备ID
off_t     st_size;      // 总字节数
blksize_t st_blksize;   // I/O块字节数
blkcnt_t  st_blocks;    // 存储块数
time_t    st_atime;     // 最后访问时间
time_t    st_mtime;     // 最后修改时间
time_t    st_ctime;     // 最后状态改变时间
}
```

知识讲解

文件锁的元数据

知识讲解

- stat结构的st_mode成员表示文件的类型和权限，该成员在stat结构中被声明为mode_t类型，其原始类型在32位系统中被定义为unsigned int，即32位无符号整数，但到目前为止，只有其中的低16位有意义
- 用16位二进制数(B15...B0)表示的文件类型和权限，从高到低可被分为五组
  - B15 - B12 : 文件类型
  - B11 - B9 ：设置用户ID，设置组ID，粘滞
  - B8 - B6 : 拥有者用户的读、写和执行权限
  - B5 - B3 : 拥有者组的读、写和执行权限
  - B2 - B0 : 其它用户的读、写和执行权限

文件锁的元数据

- 文件类型：B15 - B12

…	B_{17}	B_{16}	B_{15}	B_{14}	B_{13}	B_{12}	B_{11}	B_{10}	B_9	B_8	B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0	
			1				S_IFREG												普通文件
				1			S_IFDIR												目录
			1	1			S_IFSOCK												本地套接字
					1		S_IFCHR												字符设备
				1	1		S_IFBLK												块设备
			1		1		S_IFLNK												符号链接
						1	S_IFIFO												有名管道
			1	1	1	1	S_IFMT												掩码

知识讲解

文件锁的元数据

- 拥有者用户的读、写和执行权限：B8 - B6

知识讲解

文件锁的元数据

- 拥有者组的读、写和执行权限：B5 - B3

知识讲解

文件锁的元数据

- 其他用户的读、写和执行权限：B2 - B0

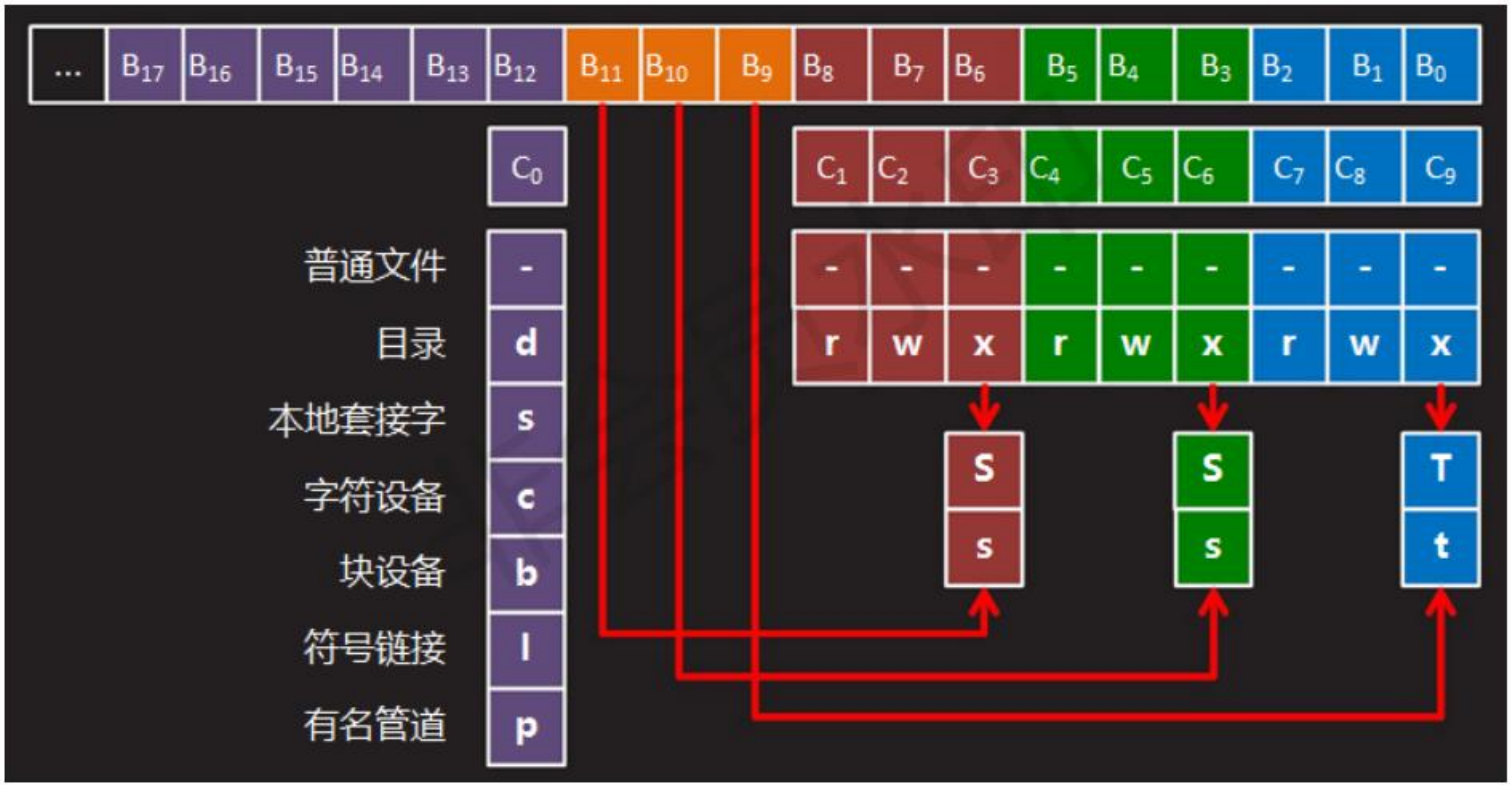
知识讲解

文件锁的元数据

- 辅助分析文件类型的实用宏
  - S_ISREG() ：是否普通文件
  - S_ISDIR() ：是否目录
  - S_ISSOCK()：是否本地套接字
  - S_ISCHR() ：是否字符设备
  - S_ISBLK() ：是否块设备
  - S_ISLNK() ：是否符号链接
  - S_ISFIFO() ：是否有名管道

知识讲解

文件锁的元数据


...
B17 B16 B15 B14 B13 B12 B11 B10 B9 B8 B7 B6 B5 B4 B3 B2 B1 B0
C0
C1 C2 C3 C4 C5 C6 C7 C8 C9
普通文件 -
目录 d
本地套接字 s
字符设备 c
块设备 b
符号链接 l
有名管道 p
- - - - - - - - -
r w x r w x r w x
S s
S s
T t

内存映射文件

内存映射文件

- #include <sys/mman.h>
- void* mmap(void* start, size_t length, int prot, int flags,
  int fd, off_t offset);
  - 功能：建立虚拟内存到物理内存或磁盘文件的映射：
  - 参数：start： 映射区虚拟内存的起始地址，NULL系统自动选定后返回。
    length：映射区字节数，自动按页圆整。
    prot： 映射区操作权限，可取以下值：
    PROT_READ - 映射区可读
    PROT_WRITE - 映射区可写
    PROT_EXEC - 映射区可执行
    PROT_NONE - 映射区不可访问

知识讲解

内存映射文件

- 参数：flags：映射标志，可取以下值
  - MAP_ANONYMOUS- 匿名映射，将虚拟内存映射到物理内存而非文件，忽略fd和offset参数
  - MAP_PRIVATE - 对映射区的写操作只反映到缓冲区中并不会真正写入文件
  - MAP_SHARED - 对映射区的写操作直接反映到文件中
  - MAP_DENYWRITE - 拒绝其它对文件的写操作
  - MAP_FIXED - 若在start上无法创建映射，则失败（无此标志系统会自动调整）
  - fd: 文件描述符
  - offset: 偏移量，以页为单位
- 返回值：成功返回映射区虚拟内存的起始地址，失败返回MAP_FAILED(-1)

内存映射文件

- #include <sys/mman.h>
- int munmap(void* start, size_t length);
  - 功能：解除虚拟内存到物理内存或磁盘文件的映射
  - 参数：start：映射区虚拟内存的起始地址
    length：映射区字节数，自动按页圆整
  - 返回值：成功返回0，失败返回-1

- munmap允许对映射区的一部分解映射，但必须按页处理

知识讲解

谢谢

UC Day06

复习课

设置用户ID位
设置组ID位
粘滞位

文件锁的元数据

知识讲解

- stat结构的st_mode成员表示文件的类型和权限，该成员在stat结构中被声明为mode_t类型，其原始类型在32位系统中被定义为unsigned int，即32位无符号整数，但到目前为止，只有其中的低16位有意义
- 用16位二进制数(B15...B0)表示的文件类型和权限，从高到低可被分为五组
  - B15 - B12 : 文件类型
  - B11 - B9 ：设置用户ID，设置组ID，粘滞
  - B8 - B6 : 拥有者用户的读、写和执行权限
  - B5 - B3 : 拥有者组的读、写和执行权限
  - B2 - B0 : 其它用户的读、写和执行权限

文件锁的元数据

- 设置用户ID、设置组ID和粘滞：B11 - B9

知识讲解

文件锁的元数据

- 设置用户ID、设置组ID和粘滞：B11 - B9
  - 系统中的每个进程其实都有两个用户ID，一个叫实际用户ID，一个叫有效用户ID
  - 进程的实际用户ID继承自其父进程的实际用户ID。当一个用户通过合法的用户名和口令登录系统以后，系统就会为他启动一个Shell进程，Shell进程的实际用户ID就是该登录用户的用户ID。该用户在Shell下启动的任何进程都是Shell进程的子进程，自然也就继承了Shell进程的实际用户ID

知识讲解

文件锁的元数据

- 设置用户ID、设置组ID和粘滞：B11 - B9
  - 一个进程的用户身份决定了它可以访问哪些资源，比如读、写或者执行某个文件。但真正被用于权限验证的并不是进程的实际用户ID，而是其有效用户ID。一般情况下，进程的有效用户ID就取自其实际用户ID，可以认为二者是等价的
  - 但是，如果用于启动该进程的可执行文件带有设置用户ID位，即B11位为1，那么该进程的有效用户ID就不再取自其实际用户ID，而是取自可执行文件的拥有者用户ID

知识讲解

文件锁的元数据

- 设置用户ID、设置组ID和粘滞：B11 - B9
  - 系统管理员常用这种方法提升普通用户的权限，让他们有能力去完成一些本来只有root用户才能完成的任务。例如，他可以为某个拥有者用户为root的可执行文件添加设置用户ID位，这样一来无论运行这个可执行文件的实际用户是谁，启动起来的进程的有效用户ID都是root，凡是root用户可以访问的资源，该进程都可以访问。当然，具体访问哪些资源，以何种方式访问，还要由这个可执行文件的代码来决定。作为一个安全的操作系统，不可能允许一个低权限用户在高权限状态下为所欲为。如通过passwd命令修改口令
  - 带有设置用户ID位的不可执行文件：没有意义
  - 带有设置用户ID位的目录文件：没有意义

知识讲解

文件锁的元数据

- 设置用户ID、设置组ID和粘滞：B11 - B9
  - 与设置用户ID位的情况类似，如果一个可执行文件带有设置组ID位，即B10位为1，那么运行该可执行文件所得到的进程，它的有效组ID同样不取自其实际组ID，而是取自可执行文件的拥有者组ID
  - 带有设置组ID位的不可执行文件：某些系统上用这种无意义的组合表示强制锁
  - 带有设置组ID位的目录文件：在该目录下创建文件或子目录，其拥有者组取自该目录的拥有者组，而非创建者用户所隶属的组

知识讲解

文件锁的元数据

- 设置用户ID、设置组ID和粘滞：B11 - B9
  - 带有粘滞位(B9)的可执行文件,在其首次运行并结束后， 其代码区被连续地保存在磁盘交换区中，而一般磁盘文件的数据块是离散存放的。因此，下次运行该程序可以获得较快的载入速度
  - 带有粘滞位(B9)的不可执行文件:没有任何意义
  - 带有粘滞位(B9)的目录： 除root以外的任何用户在该目录下，都只能删除或者更名那些属于自己的文件或子目录，而对于其它用户的文件或子目录，既不能删除也不能改名，如/tmp目录

知识讲解

下节课见