

同济大学计算机系操作系统课程作业

进程管理 一

2023-11-13

学号 2152118 姓名 史君宝

一、（1）注释 PPT10~11，写出 read 系统调用的执行细节（2）画 2 张图，补全随后 write 系统调用的执行细节。不必面面俱到，不清楚的地方红笔标出来，本周四前完成。

3、系统调用

```
#include <fcntl.h>
char buffer[2048];
int version = 1;

.....

copyOperation (old,new)
int old,new;
{
    int count;
    N: while ((count=read(old,buffer,sizeof(buffer)))>0)
        write(new,buffer,count);
}
```

用户态

核心态

进程返回用户态，执行应用程序处理buffer数组保存的文件数据

进程执行read系统调用，读磁盘文件。将文件数据送buffer数组

应用程序执行 read 系统调用。负责执行这个程序的进程陷入内核，读取磁盘文件数据，将其送入用户空间，buffer数组。完成后，系统调用结束。进程返回用户态，执行应用程序处理 buffer数组中的文件数据，把它写进另一个文件 new。

操作系统
drong2004@tongji.edu.cn 15921642146

电信学院计算机系 邓蓉

9

read 系统调用的执行细节 1

T0 时刻，PA 执行应用程序，子程序 CopyOperation 执行 read 系统调用读 old 文件数据。

应用程序

0x 80h

处理 buffer 中的文件数据

系统调用入口程序

```
.....
Sys_Read()
.....
```

Sys_Read()

上半段：...向磁盘发读命令
Sleep()入睡： p_stat = SSLEEP, p_wchan = &***, Swtch()

下半段：将内存get的磁盘数据送入用户空间 buffer数组.....

T0

CPU 磁盘

CPU执行其它进程

操作系统
drong2004@tongji.edu.cn 15921642146

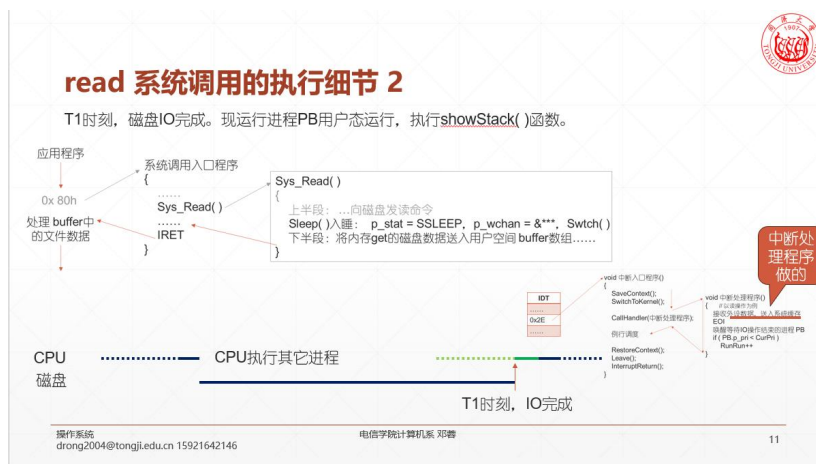
电信学院计算机系 邓蓉

10

P10 的注释：

在原程序 PA 的执行过程中，其子程序 CopyOperation 的执行需要采用 read 系统调用读 old 文件的数据，在要执行 read 读文件时触发系统调用 0X 80h。系统调用属于中断，因此会停止当前用户态指令执行，从用户态转换到核心态中，会执行中断隐指令，保存硬件现场。之后进入对应的中断入口程序即系统调用入口程序。

在系统调用入口程序中会进行保护软件现场等操作，执行到 Sys_Read()函数的时候相当于中断处理程序，会发送命令给磁盘读取文件数据，同时调用 Sleep 函数暂停当前进程。保存 Sleep 的栈帧，并记录核心栈的 ESP 和 EBP 指针，在进程 PA 的 proc 表中将 p_stat 设置为 SSLEEP，记录 PA 进程下台的原因为等待磁盘数据。之后会调用 Swtch()函数执行一次例行调度，切换到优先级最高的就绪进程中。



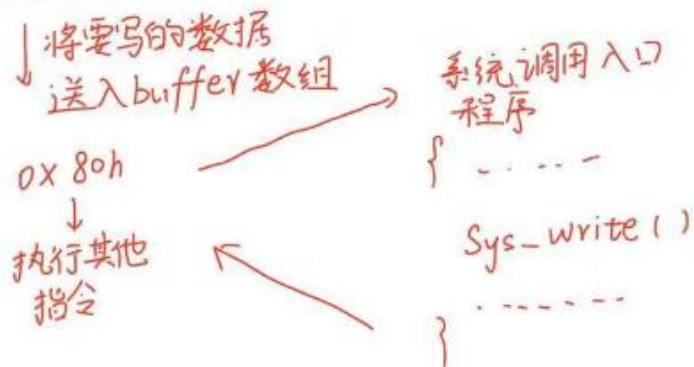
P11 的注释：

在 T1 时刻，此时正在执行的进程为 PB，进程 PA 之前在等待磁盘数据而睡眠。由于 T1 时 PA 所需的磁盘 IO 操作已经完成，会向 CPU 发送中断请求。CPU 得到请求后，会执行相关的中断操作，即获取中断号和查询中断向量表。之后执行中断隐指令保存硬件现场，进入核心态，执行中断入口程序，保存软件现场后执行对应的磁盘 IO 中断处理程序。系统缓存会记住 IO 读取的磁盘数据，并向中断控制器发送 EOI 信号，唤醒进程 PA，并进行进程的例行调度。

如果进程 PA 通过例行调度上台，就会执行 Sys_read 中未完成的部分，将获取的磁盘数据写入 buffer，并退出 Sys_read() 函数，继续执行系统调用入口程序。最后经过例行调度，并通过 IRET 返回用户态执行。

write 的执行过程

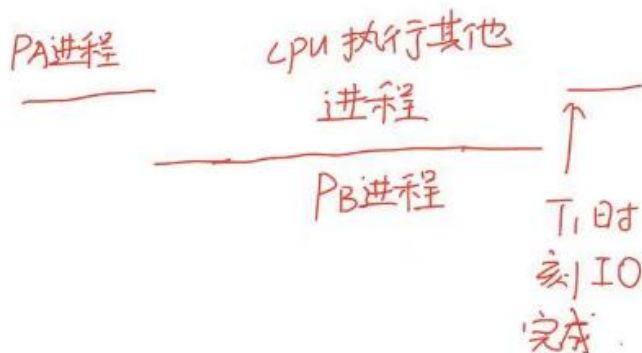
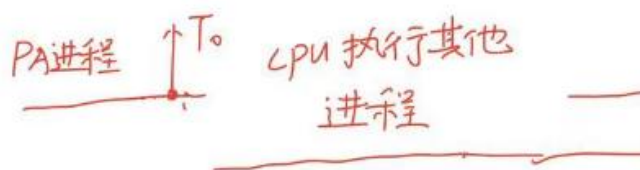
应用程序



`sys_write()`

```
{ 上半段向磁盘发生写的命令  
sleep() 入睡, p_stat = SSLEEP  
p_wchan = &xxx  Switch()  
下半段: 执行写完毕的操作  
}
```

执行过程



二、修改 Kernel.cpp 中的 GetUser() 函数，用 ESP 寄存器计算得到现运行进程 user 结构的起始地址（调通系统和我说一声，加分）。评价系统性能。

```
166 User& Kernel::GetUser()  
167 {  
168     return *(User*)USER_ADDRESS;  
169 }
```

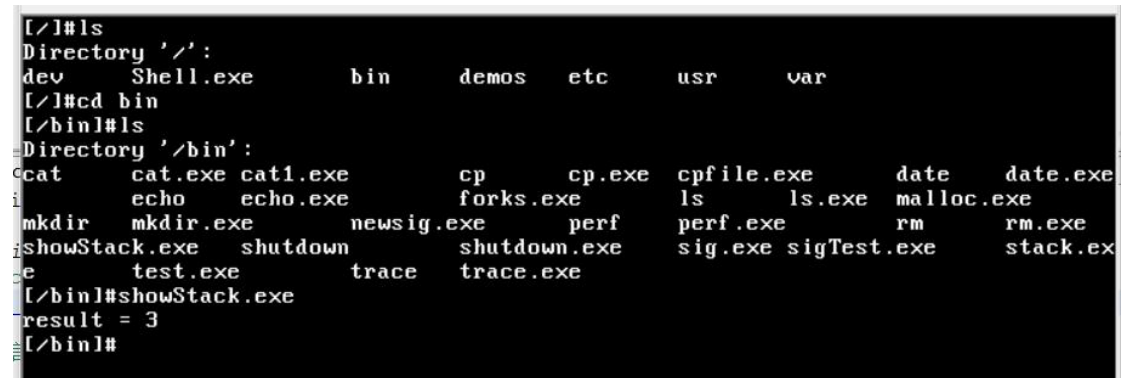
修改内容：

```
User& Kernel::GetUser()  
{  
    //return *(User*)USER_ADDRESS;  
  
    //修改部分，通过ESP获得user结构起始地址  
    unsigned int esp_number;  
    __asm__ __volatile__ ( "mov %%esp, %0" : "=r"(esp_number));  
    unsigned int result = esp_number & 0xFFFFF000;  
    return *(User*)result;  
}
```

由于在核心态下 esp 是指向核心栈的，所以其对应与 User 页号相同，所以取 esp 所指的地址，与 0xFFFFF000 相与取其前 20 位即可就是对应的 User 结构的起始地址。

系统评价：

这种方式下不仅需要一次取值，取 esp 所指地址，还要进行一次位运算，与原来直接取值相比速度比较慢。



```
[/]\#ls  
Directory '/':  
dev      Shell.exe      bin      demos      etc      usr      var  
[/]\#cd bin  
[/bin]\#ls  
Directory '/bin':  
cat      cat.exe  cat1.exe      cp      cp.exe  cpfile.exe  date      date.exe  
echo      echo.exe  forks.exe      ls      ls.exe  malloc.exe  
mkdir     mkdir.exe  newsig.exe     perf     perf.exe  rm          rm.exe  
showStack.exe  shutdown  shutdown.exe   sig.exe  sigTest.exe  stack.exe  
test.exe  trace      trace.exe  
[/bin]\#showStack.exe  
result = 3  
[/bin]\#
```