第 2 部分　实验指导

系统调用函数说明、参数值及定义

1. fork()

创建一个新进程。

```
int fork()
```

返回值的意义如下：

- 0：创建子进程，从子进程返回的 id 值。
- 大于 0：从父进程返回的子进程 id 值。
- －1：创建失败。

2. lockf(files,function,size)

用作锁定文件的某些段或者整个文件。
头文件如下：

```
#include <unistd.h>
```

参数定义如下：

```
int lockf(files,function,size);
int files,function;
long size;
```

其中，files 是文件描述符；function 是锁定/解锁开关，1 表示锁定，0 表示解锁；size 是锁定或解锁的字节数，0 表示从文件的当前位置到文件尾。

3. msgget(key,flag)

获得一个消息的描述符，该描述符指定一个消息队列，以便用于其他系统调用。
头文件如下：

```
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/msg.h>
```

参数定义如下：

```
int msgget(key,flag);
key_t key;
int flag;
```

语法格式如下：

```
msgqid=msgget(key,flag)
```

其中，msgqid 是该系统调用返回的描述符，失败则返回－1；flag 由操作允许权限和控制命令值相或得到，例如 IPC_CREATE|0400（是否该队列应被创建）、IPC_EXCL|0400（是否该队列的创建应是互斥的）。

msgsnd(id,msgp,size,flag)

发送一个消息。

头文件如下：

```
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/msg.h>
```

参数定义如下：

```
int msgsnd(id,msgp,size,flag);
int id,size,flag;
struct msgbuf *msgp;
```

其中，id 是返回消息队列的描述符；msgp 是指向用户存储区的一个构造体指针；size 指示由 msgp 指向的数据结构中字符数组的长度，即消息的长度，这个数组的最大值由系统调用参数 MSG_MAX 确定；flag 规定当核心用尽内部缓冲空间时应执行的动作。若在标志 flag 中未设置 IPC_NOWAIT 位，则当该消息队列中的字节数超过某一最大值或系统范围的消息数超过某一最大值时，调用 msgsnd 进程睡眠；若设置 IPC_NOWAIT，则在此情况下 msgsnd 进程立即返回。

5. msgrcv(id,msgp,size,type,flag)

接收一个消息。

头文件如下：

```
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/msg.h>
```

参数定义如下：

```
int msgrcv(id,msgp,size,type,flag);
int id,size,type,flag;
struct msgbuf *msgp;
struct msgbuf{long mtype;char mtext[];};
```

语法格式如下：

```
count=msgrcv(id,msgp,size,type,flag)
```

其中,id 是消息描述符;msgp 是存放用来接收消息的用户数据结构的地址;size 是 msgp 中数据数组的大小;type 是用户要读的消息类型,为 0 时接收该队列的第一个消息,为正时接收类型 type 的第一个消息,为负时接收小于或等于 type 绝对值的最低类型的第一个消息;flag 规定当该队列无消息时,核心应当做什么事,当设置为 IPC_NOWAIT 时立即返回,当设置为 MSG_NOERROR 且接收的消息大小大于 size 时,核心截断正在接收的消息。

count 是返回消息正文的字节数。

6. msgctl(id,cmd,buf)

查询一个消息描述符的状态,设置它的状态,以及删除一个消息描述符。

头文件如下:

```
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/msg.h>
```

参数定义如下:

```
int msgctl(id,cmd,buf);
int id,cmd;
struct msgid_ds *buf;
```

调用成功时返回 0,调用不成功时返回-1。

参数说明如下:

(1) id 是用来识别该消息的描述符。

(2) cmd 规定命令的类型:

- IPC_STAT,将与 id 关联的消息队列首标读入 buf。
- IPC_SET,为与 id 关联的消息队列设置有效的用户 id 和组 id、操作权限和字节数。
- IPC_RMID,删除与 id 关联的消息队列。

(3) buf 是含有控制参数或查询结果的用户数据结构的地址。

msgid_ds 定义如下:

```
struct msgid_ds
{
    struct ipc_perm msg_perm;        /*许可权结构*/
    short pad1[7];                   /*由系统使用*/
    ushort msg_qnum;                 /*队列中的消息数*/
    ushort msg_qbytes;               /*队列中的最大字节数*/
    ushort msg_lspid;                /*最后发送消息的进程 id*/
    ushort msg_lrpid;                /*最后接收消息的进程 id*/
    time_t msg_stime;                /*最后发送消息的时间*/
    time_t msg_rtime;                /*最后接收消息的时间*/
    time_t msg_ctime;                /*最后更改时间*/
};
```

```
struct ipc_perm
{
    ushort uid;                    /* 当前用户 id */
    ushort gid;                    /* 当前组 id */
    ushort cuid;                   /* 创建用户 id */
    ushort cgid;                   /* 创建组 id */
    ushort mode;                   /* 操作权限 */
    {short pad1;long pad2}         /* 由系统使用 */
};
```

7. shmget(key,size,flag)

获得一个共享存储区。

头文件如下：

```
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/shm.h>
```

语法格式如下：

```
shmid=shmget(key,size,flag)
```

参数定义如下：

```
int shmget(key,size,flag);
key_t key;
int size,flag;
```

其中，size 是存储区的字节数，key 和 flag 与系统调用 msgget 中的参数含义相同。

flag 中包含的操作权限如下(权限值为八进制数)：

- 用户可读：00400。
- 用户可写：00200。
- 小组可读：00040。
- 小组可写：00020。
- 其他可读：00004。
- 其他可写：00002。

flag 中包含的控制命令如下：

- IPC_CREATE：0001000。
- IPC_EXCL：0002000。

例如：

```
shmid=shmget(key,size,(IPC_CREATE|0400));
```

创建一个关键字为 key、长度为 size 的共享存储区。

8. shmat(id,addr,flag)

从逻辑上将一个共享存储区附接到进程的虚拟地址空间上。

头文件如下：

```
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/shm.h>
```

参数定义如下：

```
int * shmat(id,addr,flag);
int id,flag;
char * addr;
```

语法格式如下：

```
virtaddr=shmat(id,addr,flag)
```

其中，id 是共享存储区的标识符；addr 是要附接共享存储区的虚拟地址，若 addr 是 0，系统选择一个适当的地址附接该共享存储区；flag 规定对此区的读写权限以及系统是否可以舍弃用户指定的地址，设置为 shm_rdonly 表示只允许读操作，设置为 shm_rnd 表示操作系统在必要时舍去这个地址。

virtaddr 是附接共享存储区的虚拟地址。

9. shmdt(addr)

把一个共享存储区从指定进程的虚拟地址空间中删除(称为断接)。

头文件如下：

```
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/shm.h>
```

参数定义如下：

```
int shmdt(addr);
char * addr;
```

调用成功时返回 0，调用不成功时返回－1。其中，addr 是系统调用 shmdt 返回的地址。

10. shmctl(id,cmd,buf)

对与共享存储区关联的各种参数进行操作，从而对共享存储区进行控制。

头文件如下：

```
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/shm.h>
```

参数定义如下：

```
int shmctl(id,cmd,buf);
int id,cmd;
```

```
struct shmid_ds *buf;
```

调用成功时返回0，调用不成功时返回－1。

参数说明如下：

(1) id为共享存储区的标识符。

(2) cmd规定操作的类型：

- IPC_STAT：返回包含在指定数据结构中的状态信息，并且把它放置在用户存储区中的*buf指针所指的数据结构中。执行此命令的进程必须有读权限。
- IPC_SET：对于指定的shmid，为它设置有效用户id、组id和操作权限。
- IPC_RMID：删除指定的shmid以及与它相关的共享存储区的数据结构。
- SHM_LOCK：在内存中锁定指定的共享存储区。必须是超级用户才可以进行此项操作。

(3) buf是一个用户级数据结构地址。

shmid_ds定义如下：

```
shmid_ds
{
    struct ipc_perm shm_perm;       /*权限结构*/
    int shm_segsz;                  /*段大小*/
    int pad1;                       /*由系统使用*/
    ushort shm_lpid;                /*最后操作的进程id*/
    ushort shm_cpid;                /*创建者的进程id*/
    ushort shm_nattch;              /*当前附接数*/
    short pad2;                     /*由系统使用*/
    time_t shm_atime;               /*最后附接时间*/
    time_t shm_dtime;               /*最后断接时间*/
    time_t shm_ctime;               /*最后修改时间*/
}
```

11. signal(sig,function)

允许调用进程控制对软中断信号的处理。

头文件如下：

```
#include<signal.h>
```

参数定义如下：

```
signal (sig,function)
int sig;
void (*func)();
```

其中，sig的值如下：

- SIGHVP：挂起。
- SIGINT：在键盘上按Delete键或Break键。
- SIGQUIT：在键盘上按Quit键。
- SIGILL：非法指令。

- SIGTRAP：跟踪的断点。
- SIGIOT：IOT 指令。
- SIGEMT：EMT 指令。
- SIGFPE：浮点运算溢出。
- SIGKILL：要求终止进程。
- SIGBUS：总线错。
- SIGSEGV：段违例。
- SIGSYS：系统调用参数错。
- SIGPIPE：向无读者管道上写。
- SIGALRM：闹钟。
- SIGTERM：软件终止。
- SIGUSR1：用户定义信号。
- SIGUSR2：第二个用户定义信号。
- SIGCLD：子进程终止。
- SIGPWR：电源故障。

function 的值如下：

- SIG_DFL：默认操作。对除 SIGPWR 和 SIGCLD 以外的所有信号的默认操作是进程终止。对信号 SIGQUIT、SIGILL、SIGTRAP、SIGIOT、SIGEMT、SIGFPE、SIGBUS、SIGSEGV 和 SIGSYS，它产生一个内存映像文件。
- SIG_IGN：忽视该信号的出现。

func 是在该进程中的一个函数地址。在系统内核返回用户态时，它以软中断信号的序号作为参数调用该函数，对除了信号 SIGILL、SIGTRAP 和 SIGPWR 以外的信号，系统内核自动重新设置软中断信号处理程序的值为 SIG_DFL。一个进程不能捕获 SIGKILL 信号。

实验 1　计算机系统启动

实验平台

（1）一台运行 Ubuntu 操作系统或可运行 Ubuntu 虚拟机的计算机。

（2）在 Ubuntu 操作系统上运行 QEMU 模拟器，利用它模拟硬件平台进行计算机系统启动实验。

实验目的

（1）熟悉计算机系统启动流程。

（2）深入理解计算机系统启动过程中的关键性步骤。

实验预备内容

了解开源模拟器 QEMU；认识系统启动流程由 BIOS→Bootloader→操作系统内核 3 个关键步骤组成。

实验内容

本实验将设计实现一个简单的BIOS，该BIOS能够为操作系统启动建立基本的运行环境，并从硬盘加载操作系统。本实验中的操作系统选用MIT6.828课程提供的一个小内核（基于xv6，可运行一个shell接收用户输入的命令以及显示输出）。

本实验基于一个开源的BIOS—— SeaBIOS，对SeaBIOS进行简化，得到一个简单的且具备完整核心功能的BIOS——SimpleBIOS，以帮助学生理解计算机系统启动的基本过程和BIOS的原理与实现。

本实验的总体思路为使用QEMU模拟x86平台（例如Intel公司推出的i440FX主板）。QEMU支持指定第三方BIOS启动，因此可指定本实验编写的BIOS进行系统引导过程。本实验将编写一个简化版本的BIOS，在QEMU提供的i440FX主板（支持奔腾处理器Pro、奔腾处理器Ⅱ）上运行该BIOS，实验现象为加载并启动一个带shell的简单操作系统内核。

本实验将包含以下3部分。

1. 计算机启动

这部分的设计可适当弱化，仅向学生展现BIOS的引导过程即可。

2. PCI总线及设备初始化

这部分内容是本实验的主体，将对PCI设备进行遍历，为每个PCI设备分配I/O端口和内存地址空间，配置相应中断，并对设备进行初始化。同时，在这部分内容中需要向学生解释PCI总线与设备、中断、内存映射以及各类相关概念。在实现上，这部分仅保留一些必要的设备初始化（例如硬盘控制器初始化）即可。

3. 操作系统加载与启动

此部分内容的实现去除了其他存储介质加载程序部分，仅考虑从硬盘中加载程序的过程。

实验2　进 程 管 理

实验目的

（1）加深对进程概念的理解，明确进程和程序的区别。

（2）进一步认识并发执行的实质。

（3）分析进程争用资源的现象，学习解决进程互斥的方法。

（4）了解Linux系统中进程通信的基本原理。

实验预备内容

（1）阅读Linux的sched.h源文件，加深对进程管理概念的理解。

(2) 阅读 Linux 的 fork.c 源文件,分析进程的创建过程。

实验内容

1. 进程的创建

编写一段程序,使用系统调用 fork()创建两个子进程。当此程序运行时,在系统中有一个父进程和两个子进程活动。让每一个进程在屏幕上显示一个字符:父进程显示字符 a,两个子进程分别显示字符 b 和 c。观察、记录屏幕上的显示结果,并分析原因。

2. 进程的控制

修改已编写的程序,将每个进程输出一个字符改为每个进程输出一句话,再观察程序执行时屏幕上出现的现象,并分析原因。

如果在程序中使用系统调用 lockf()给每一个进程加锁,可以实现进程之间的互斥,观察并分析出现的现象。

3. 进程的软中断通信

(1) 编制一段程序,使其实现进程的软中断通信。

要求:使用系统调用 fork()创建两个子进程,再使用系统调用 signal()让父进程捕捉来自键盘的中断信号(即按 Delete 键)。当捕捉到中断信号后,父进程使用系统调用 kill()向两个子进程发出信号。子进程捕捉到信号后分别输出下列信息后终止:

```
child process 1 is killed by parent!
child process 2 is killed by parent!
```

父进程等待两个子进程终止后,输出下列信息后终止:

```
parent process is killed!
```

(2) 在上面的程序中增加语句 signal(SIGINT,SIG_IGN)和 signal(SIGQUIT,SIG_IGN),观察执行结果,并分析原因。

4. 进程的管道通信

编制一段程序,实现进程的管道通信。

使用系统调用 pipe()建立一条管道。两个子进程 P1 和 P2 分别向管道各写一句话:

```
Child process 1 is sending a message!
Child process 2 is sending a message!
```

父进程从管道中读出来自两个子进程的信息,显示在屏幕上。

要求父进程先接收子进程 P1 发来的消息,再接收子进程 P2 发来的消息。

思考

(1) 系统是怎样创建进程的?

（2）可执行文件加载时进行了哪些处理？

（3）当首次调用新创建进程时，其入口在哪里？

（4）进程通信有什么特点？

实验 3　进程间通信

实验目的

Linux 系统的进程间通信允许在任意进程间大批量地交换数据。本实验的目的是了解和熟悉 Linux 支持的消息通信机制、共享存储区机制及信息量机制。

实验预备内容

阅读 Linux 系统的 msg.c、sem.c 和 shm.c 等源文件，熟悉 Linux 的 3 种通信机制。

实验内容

1. 消息的创建、发送和接收

（1）使用系统调用 msgget()、msgsnd()、msgrcv()及 msgctl()，编制一个发送和接收长度为 1KB 的消息的程序。

（2）观察上面的程序，说明控制消息队列的系统调用 msgctl()在此起什么作用。

2. 共享存储区的创建、附接和断接

使用系统调用 shmget()、shmat()、shmdt()和 shmctl()，编制一个与上述功能相同的程序。

3. 两种通信机制的比较

比较上述两种通信机制中数据传输的时间。

实验 4　存 储 管 理

实验目的

存储管理的主要功能之一是合理地分配空间。请求页式存储管理是一种常用的虚拟存储管理技术。

本实验的目的是通过完成请求页式存储管理中的页面置换算法模拟设计，了解虚拟存储技术的特点，掌握请求页式存储管理的页面置换算法。

实验内容

（1）通过随机数产生一个指令序列，共 320 条指令。指令的地址按下述原则生成：