实验二：进程控制

实验简介：本次实验主要理解程序和进程的关系，进程的创建，多进程的运行以及同步互斥的控制。

实验目标：

（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）分析进程竞争资源现象，学习解决进程互斥的方法。

实验内容：

（1）进程的创建

编写一段源程序，使系统调用 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”;子进程分别显示字符“b”和字符“c”。试观察纪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，在观察程序执行时屏幕出现的现象，并分析原因。

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

（3）进程的管道通信

编制一段程序，实现进程的管理通信。

使用系统调用 pipe()建立一条管道线；两个子进程 P1 和 P2 分别向管道中写一句话：

Child 1 is sending a message!

Child 2 is sending a message!

而父进程则从管道中读出来自于两个子进程的信息,显示在屏幕上。

要求父进程先接收子进程P1发来的消息，然后再接收子进程P2发来的消息。

实验所需基础：

操作系统：Linux RHEL 6.0

实验是否需要联网：否

实训步骤：

步骤一：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”，子进程分别显示字符“b”和“c”。试观察记录屏幕上的显示结果，并分析原因。

〈程序〉

```
#include<stdio.h>
main()
{
int p1,p2;
while((p1=fork())==-1);
if(p1==0)                /*子进程创建成功*/
       putchar('b');
else
 {
      while((p2=fork())==-1);
    if(p2==0)              /*子进程创建成功*/
         putchar('c');
```

```
      else
        putchar('a');            /*父进程执行*/
}
}
```

步骤二：进程的控制

修改已编写好的程序,将每个程序的输出由单个字符改为一句话,再观察程序执行时屏幕上出现的现象,并分析其原因。如果在程序中使用系统调用lockf()来给每个程序加锁,可以实现进程之间的互斥,观察并分析出现的现象。

〈程序 1〉

```
#include<stdio.h>
main()
{
 int p1,p2,i;
 while((p1=fork())==-1);
   if(p1==0)
     for(i=0;i<50000;i++)
       printf("child %d\n",i);
   else
        {

        while((p2=fork())==-1);
        if(p2==0)
          for(i=0;i<50000;i++)
              printf("son %d\n",i);
        else
          for(i=0;i<50000;i++)
              printf("daughter %d\n",i);
}
}
```

〈程序 2〉

```
include<stdio.h>
main()
{
 int p1,p2,i;
 while((p1=fork())==-1);
```

```
    if(p1==0)
      {
        lockf(1,1,0);
        for(i=0;i<50000;i++)
        printf("child %d\n",i);
        lockf(1,0,0);
     }
    else
         {

         while((p2=fork())==-1);
         if(p2==0)
                       {
           lockf(1,1,0);
          for(i=0;i<50000;i++)
              printf("son %d\n",i);
           lockf(1,0,0);
                       }
         else
                     {
           lockf(1,1,0);
          for(i=0;i<50000;i++)
              printf("daughter %d\n",i);
            lockf(1,0,0);
                  }
  }
  }
```

比较<程序 1>和<程序 2>的运行结果，分析 lockf()函数的作用。

步骤三：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()建立一条管道线。两个子进程 p1 和 p2 分别向通道个写一句话：

child1 process is sending message!

child2 process is sending message!

而父进程则从管道中读出来自两个进程的信息，显示在屏幕上。

〈程序〉

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

```
#include <signal.h>
#include <stdio.h>
int pid1,pid2;

main( )
{
int fd[2];
char outpipe[100],inpipe[100];
pipe(fd);                          /*创建一个管道*/
while ((pid1=fork( ))==-1);
if(pid1==0)
  { lockf(fd[1],1,0);
    sprintf(outpipe,"child 1 process is sending message!");
    /*把串放入数组 outpipe 中*/
    write(fd[1],outpipe,50);     /*向管道写长为 50 字节的串*/
    sleep(5);                  /*自我阻塞 5 秒*/
    lockf(fd[1],0,0);
    exit(0);
   }
else
  { while((pid2=fork( ))==-1);
    if(pid2==0)
  {     lockf(fd[1],1,0);           /*互斥*/
        sprintf(outpipe,"child 2 process is sending message!");
        write(fd[1],outpipe,50);
        sleep(5);
        lockf(fd[1],0,0);
        exit(0);
     }
     else
     {   wait(0);               /*同步*/
         read(fd[0],inpipe,50);   /*从管道中读长为 50 字节的串*/
         printf("%s\n",inpipe);
         wait(0);
         read(fd[0],inpipe,50);
         printf("%s\n",inpipe);
         exit(0);
    }
  }
}
```

分析 wait（）系统调用、sleep（）系统调用的作用。