实验二 进程管理

一、 实验类型

本实验为设计性实验。

二、 实验目的与任务

1）掌握进程的概念，明确进程的含义

2）掌握进程创建方法，认识并了解并发执行的实质

3）熟悉进程的睡眠、同步、撤消等进程控制方法

4）分析进程竞争资源的现象，学习解决进程互斥的方法

三、 预习要求

1）进程的概念

2）进程控制的概念及内容

3）进程的并发执行

4）熟悉进程同步互斥的概念

5）用到的 Linux 函数有：fork()，exec( )，wait()，exit()，lockf()等。

四、 实验基本原理

使用 fork()系统调用创建一个子进程，如果 fork( )调用成功，它向父进程返回子进程的 PID，并向子进程返回 0，即 fork( )被调用了一次，但返回了两次。此时 OS 在内存中建立一个新进程，所建的新进程是调用 fork( )父进程（parent process）的副本，称为子进程（child process）。子进程继承了父进程的许多特性，并具有与父进程完全相同的用户级上下文，父进程与子进程并发执行。fork( )只是将父进程的用户级上下文拷贝到新进程中，而 exec( )系列可以将一个可执行的二进制文件覆盖在新进程的用户级上下文的存储空间上，以更改新进程的用户级上下文。exec( )系列中的系统调用都完成相同的功能，它们把一个新程序装入内存，来改变调用进程的执行代码，从而形成新进程。如果 exec( )调用成功，调用进程将被覆盖，然后从新程序的入口开始执行，这样就产生了一个新进程，新进程的进程标识符 id 与调用进程相同。使用 lockf()系统调用对临界区进行加锁操作，实现对共享资源的互斥使用。

五、 实验仪器与设备（或工具软件）

实验设备：计算机一台，软件环境要求：安装 Red Hat Linux 操作系统和 gcc 编译器。

六、 实验内容

1）进程的创建

（1）编写程序，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'a'，子进程分别显示字符'b'和字符'c'。试观察记录屏幕上的显示结果，并分析原因。

参考程序如下：

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

运行结果：bca(或 bac)

分析：从进程并发执行来看，输出 bac，acb 等情况都有可能。

原因：fork()创建进程所需的时间要多于输出一个字符的时间，因此在主程序创建进程的同时，进程 2 就输出了”b”，而进程 2 和主程序的输出次序是随机的，所以出现上述结果。

（2）修改程序，每一个进程循环显示一句话。子进程显示'daughter …'及'son ……'，父进程显示 'parent ……'，观察结果，分析原因。

参考程序如下：

```
#include <stdio.h>
main( )
{
 int p1,p2,i;
 while((p1=fork( ))= = -1);                /*创建子进程 p1*/
 if (p1= =0)
    for(i=0;i<10;i++)
       printf("daughter    %d\n",i);
```

```
 else
    {
      while((p2=fork( ))= = -1);     /*创建子进程 p2*/
      if(p2= =0)
         for(i=0;i<10;i++)
         printf("son    %d\n",i);
      else
         for(i=0;i<10;i++)
       printf("parent    %d\n",i);
      }
}
```

运行结果：略

分析：由于函数 printf()输出和字符串之间不会被中断，因此字符串内部的字符顺序输出不变。但是由于进程并发执行时的调度顺序和父进程的抢占处理机问题，输出字符串的顺序和先后随着执行的不同而发生变化。

2）进程的互斥

（1）修改程序，用 lockf( )来给每一个进程加锁，以实现进程之间的互斥

进程加锁后的参考程序如下：

```
#include <stdio.h>
#include <unistd.h>
main( )
{
      int p1,p2,i;
      int n=500;
      while((p1=fork( ))= = -1);          /*创建子进程 p1*/
      if (p1= =0)
      {
       lockf(1,1,0); /*加锁，这里第一个参数为 stdout（标准输出设备的描述符）*/
       for(i=0;i<n;i++)
       printf("daughter %d\n",i);
       lockf(1,0,0);                          /*解锁*/
      }
else
    {
      while((p2=fork( ))= =-1);   /*创建子进程 p2*/
```

```
    if (p2= =0)
    {
     lockf(1,1,0);              /*加锁*/
     for(i=0;i<n;i++)
     printf("son %d\n",i);
     lockf(1,0,0);                    /*解锁*/
     }
     else
         {
           lockf(1,1,0);              /*加锁*/
           for(i=0;i<n;i++)
           printf(" parent %d\n",i);
           lockf(1,0,0);              /*解锁*/
         }
    }
}
```

运行结果：大致与未上锁的输出结果相同，也是随着执行时间不同，输出结果的顺序有所不同。

分析：上述程序执行时，不同进程之间不存在共享临界资源（其中打印机的互斥性已由操作系统保证）问题，所以加锁与不加锁效果相同。

（2）分析下列程序结果（用 cat　to_be_locked.txt 查看输出结果）

```
#include<stdio.h>
#include<unistd.h>
main()
{
    int p1,p2,i;
    int n=500;
    int *fp;
    fp = fopen("to_be_locked.txt","w+");
    if(fp==NULL)
     {
        printf("Fail to create file");
        exit(-1);
     }
    while((p1=fork( ))== -1);          /*创建子进程 p1*/
```

```
        if (p1==0)
        {
            lockf(*fp,1,0);                    /*加锁*/
            for(i=0;i<n;i++)
            fprintf(fp,"daughter %d\n",i);
            lockf(*fp,0,0);                    /*解锁*/
         }
       else
        {
            while((p2=fork( ))==-1);   /*创建子进程 p2*/
          if (p2==0)
            {
              lockf(*fp,1,0);              /*加锁*/
              for(i=0;i<n;i++)
              fprintf(fp,"son %d\n",i);
              lockf(*fp,0,0);              /*解锁*/
             }
         else
           {
              wait(NULL);
              lockf(*fp,1,0);              /*加锁*/
              for(i=0;i<n;i++)
              fprintf(fp,"parent %d\n",i);
              lockf(*fp,0,0);              /*解锁*/
           }
        }
      fclose(fp);
}
```

七、 实验步骤

1）进入 vi 编辑器

2）在编译器中输入所要运行的程序代码

3）退出编辑器，返回命令行输入方式，使用 gcc 编译器编译程序，获得能运行的目标程序。

4）运行目标程序，查看运行结果。

八、 注意事项

1）如果使用 gcc 编译程序有错的话，需要重新修改程序，直到无错为止。

2）注意系统是如何创建进程的？

3）查看结果是否是交替输出，如果修改输出的次数是否会出现交替现象？

4）相关函数的介绍见第一部分的介绍。

九、 实验报告要求

需要列出运行了的程序清单及相应结果，并对结果进行分析和讨论。对结果的分析主要讨论结果为什么会交替出现？并发进程是如何执行的？