

实 验 报 告

课程名称：操作系统试验

实 验 一：进程管理与进程间通信

班 级：计算机科学与技术 02 班

学生姓名：白文强

学 号：20191060064

专 业：计算机科学与技术

指导教师：杨旭涛

学 期：2021—2022 学年秋季学期

成 绩：

云南大学信息学院

一、实验目的

- 1、熟悉 linux 系统下 fork()函数的使用，并观察系统中进程创建和执行的并发执行情况；
- 2、观察和了解软中断实现的进程通信；
- 3、观察和了解通过消息队列实现进程通信的过程；
- 4、观察和了解通过共享存储区实现进程通信的过程。
- 5、掌握进程对共享存储区或变量访问时进程互斥的实现方法。

二、知识要点

- 1、创建进程函数 fork()；
- 2、进程的并发执行；
- 3、中断调用，软中断与硬件中断；
- 4、消息队列通信，共享存储区通信；
- 5、进程的互斥访问。

三、实验预习（要求做实验前完成）

- 1、了解 linux 系统中常用命令的使用方法；
- 3、掌握进程的并发执行、系统调用、中断、进程通信、进程互斥、进程同步的基本概念。

四、实验内容和试验结果

结合课程所讲授内容以及课件中的试验讲解，完成以下试验。请分别对试验过程和观察到的情况做描述和总结，并将试验结果截图附后。

1、观察进程的并发执行：用 fork 系统调用创建多个进程，各进程输出不同的内容。观察进程的行为。

①每个进程只输出一次的情况下：

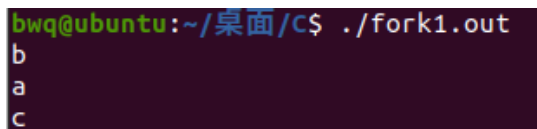
```
1. #include <stdio.h>
2. #include <unistd.h>
3.
4. int main()
5. {
6.     int p1 = 1, p2 = 0;
7.     //创建一个子进程
8.     while( (p1 = fork() )== -1 );
9.
```

```

10.     if( p1 == 0 ){
11.         //这里是子进程
12.         printf("b\n");
13.
14.     }else{
15.         //父进程
16.         while( (p2 = fork()) == -1);
17.         if( p2 == 0 ){
18.             //另一个子进程
19.             printf("c\n");
20.         }else{
21.             printf("a\n");
22.         }
23.     }
24.     return 0;
25. }

```

实验结果截图：



```

bwq@ubuntu:~/桌面/C$ ./fork1.out
b
a
c

```

三个进程都输出了自己应输出的内容。大概由于虚拟化的原因，每次运行的结果都是bac，而实际中由于进程之间的流转，可能存在多种不同的输出顺序。

②每个进程输出多次的情况下

```

1. #include <stdio.h>
2. #include <unistd.h>
3.
4. int main()
5. {
6.     int p1 = -1, p2 = -1;
7.     //创建一个子进程
8.     while( (p1 = fork() )== -1 );
9.
10.    if( p1 == 0 ){
11.        //这里是子进程
12.        for(int i = 0; i < 500; i++){
13.            printf("son %d\n", i);
14.        }
15.
16.    }else{
17.        //父进程
18.        while( (p2 = fork()) == -1);

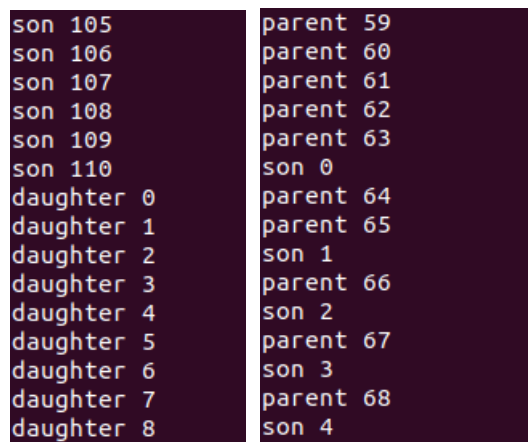
```

```

19.         if( p2 == 0 ){
20.             //另一个子进程
21.             for(int i = 0; i < 100; i++){
22.                 printf("daughter %d\n", i);
23.             }
24.         }else{
25.             for(int i = 0; i < 100; i++){
26.                 printf("parent %d\n", i);
27.             }
28.         }
29.     }
30. }

```

实验结果截图:



```

son 105      parent 59
son 106      parent 60
son 107      parent 61
son 108      parent 62
son 109      parent 63
son 110      son 0
daughter 0   parent 64
daughter 1   parent 65
daughter 2   son 1
daughter 3   parent 66
daughter 4   son 2
daughter 5   parent 67
daughter 6   son 3
daughter 7   parent 68
daughter 8   son 4

```

让每个进程都输出多次信息，可以看到，各个进程轮流使用 CPU，轮流输出信息。猜测由于虚拟机的问题，实验结果不能完全反映出真实的 CPU 调度。

2、软中断通信：用 fork 创建两个子进程，父进程响应键盘上来的中断信号（ctrl+c），调用 kill 系统调用向两个子进程发出信号（16、17 号软中断），子进程收到信号后，输出信息并结束。子进程结束后，父进程输出信息并结束。

```

1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <unistd.h>
4. #include <signal.h>
5. #include <sys/types.h>
6. #include <sys/wait.h>
7.
8. int wait_mark;
9.
10. void waiting(){
11.     while(wait_mark!=0);
12. }
13.
14. void stop(){
15.     wait_mark = 0;

```

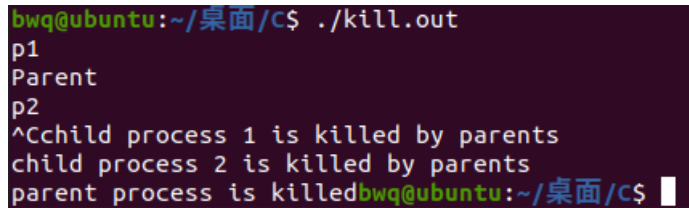
```
16. }
17.
18. int main()
19. {
20.     int p1 = 1, p2 = 0;
21.     //创建一个子进程
22.     while( (p1 = fork() )== -1 );
23.
24.     if( p1 > 0 ){
25.         while( (p2 = fork() )== -1 );
26.         if(p2 > 0){
27.             //父进程
28.             printf("Parent\n");
29.             wait_mark = 1;
30.             signal(SIGINT, stop);
31.             waiting();
32.             kill(p1,16);
33.             kill(p2,17);
34.             wait(0);
35.             wait(0);
36.             lockf(1,1,0);
37.             printf("parent process is killed");
38.             lockf(1,0,0);
39.             exit(0);
40.         }else{
41.             //子进程 2
42.             printf("p2\n");
43.             signal(SIGINT,SIG_IGN);
44.             wait_mark = 1;
45.             signal(17,stop);
46.             waiting();
47.             lockf(1,1,0);
48.             printf("child process 2 is killed by parents\n");
49.             lockf(1,0,0);
50.             exit(0);
51.         }
52.     }else{
53.         //子进程 1
54.         printf("p1\n");
55.         signal(SIGINT, SIG_IGN); //忽略
56.         wait_mark = 1;
57.         signal(16,stop);
58.         waiting();
59.         lockf(1,1,0);
```

```

60.     printf("child process 1 is killed by parents\n");
61.     lockf(1,0,0);
62.     exit(0);
63. }
64. return 0;
65. }

```

实验结果截图：



```

bwq@ubuntu:~/桌面/c$ ./kill.out
p1
Parent
p2
^Cchild process 1 is killed by parents
child process 2 is killed by parents
parent process is killedbwq@ubuntu:~/桌面/c$

```

由实验结果看出，子进程与父进程都输出完自己的身份信息后（p1, Parent, p2），主进程收到中断信号后，向两个子进程发起中断，两个子进程收到中断信号，发出被中断的消息，随后父进程结束。

3、消息队列通信：用 fork 创建两个子进程，第一个子进程（Server）创建消息队列，等待接收消息；第二个子进程（Client）打开消息队列，向消息队列中写长度为 1KB 的消息，循环 10 次。Server 从消息队列接收每一条消息。

```

1. #include <stdio.h>
2. #include <sys/types.h>
3. #include <sys/msg.h>
4. #include <sys/ipc.h>
5. #include <stdlib.h>
6. #include <unistd.h>
7. #include <sys/wait.h>
8.
9. #define MSGKEY 75
10.
11. struct msgform
12. {
13.     long mtype;
14.     char mtrex[1024];
15. } msg;
16.
17. int msgqid, i;
18.
19. void CLIENT()
20. {
21.     int i;
22.     msgqid = msgget(MSGKEY, 0777 | IPC_CREAT);
23.     for (int i = 10; i >= 1; i--)
24.     {

```

```

25.     msg.mtype = i;
26.     printf("(client)sent:");
27.     printf("%d\n", msg.mtype);
28.     msgsnd(msgqid, &msg, 1024, 0);
29. }
30. exit(0);
31. }
32.
33. void SERVER()
34. {
35.     msgqid = msgget(MSGKEY, 0777 | IPC_CREAT);
36.     do
37.     {
38.         msgrcv(msgqid, &msg, 1024, 0, 0);
39.         printf("(server)received: ");
40.         printf("%d\n", msg.mtype);
41.     }while (msg.mtype >= 1);
42.     msgctl(msgqid, IPC_RMID, 0);
43.     exit(0);
44. }
45. int main()
46. {
47.     while ((i = fork()) == -1)
48.         ;
49.     if (!i)
50.         SERVER();
51.     else
52.     {
53.         while ((i = fork()) == -1)
54.             ;
55.         if (!i)
56.             CLIENT();
57.         else
58.         {
59.             wait(0);
60.             wait(0);
61.         }
62.     }
63.     return 0;
64. }

```

实验结果截图：

```
bwq@ubuntu:~/桌面/C$ ./msg.out
(client)sent:10
(server)received: 10
(client)sent:9
(server)received: 9
(client)sent:8
(server)received: 8
(client)sent:7
(server)received: 7
(client)sent:6
(server)received: 6
(client)sent:5
(server)received: 5
(client)sent:4
(server)received: 4
(client)sent:3
(server)received: 3
(client)sent:2
(server)received: 2
(client)sent:1
(server)received: 1
```

Client 每发送一条消息，Server 就接收到了这一条消息。

采用管道通信：

```
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <unistd.h>
4. #include <signal.h>
5. #include <sys/wait.h>
6.
7. int pid1, pid2;
8.
9. int main(){
10.     int fd[2];
11.     char outpipe[100], inpipe[100];
12.     pipe(fd);
13.     while( (pid1=fork()) == -1 );
14.     if( pid1 == 0 ){
15.         //子进程 1
16.         printf("p1\n");
17.         lockf(fd[1], 1, 0);
18.         sprintf(outpipe, "child 1 process is sending a message\n");
19.         write(fd[1], outpipe, 50);
20.         sleep(1);
21.         lockf(fd[1], 0, 0);
22.         exit(0);
23.     }else{
24.         while( (pid2=fork()) == -1 );
25.         if( pid2 == 0 ){
```

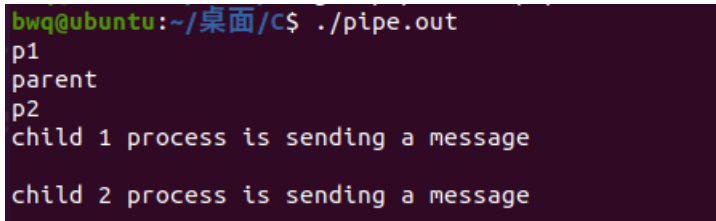


```

26.          //子进程 2
27.          printf("p2\n");
28.          lockf(fd[1], 1, 0);
29.          sprintf(outpipe, "child 2 process is sending a message\n"
    );
30.          write(fd[1],outpipe, 50);
31.          sleep(1);
32.          lockf(fd[1], 0, 0);
33.          exit(0);
34.      }else{
35.          //父进程
36.          printf("parent\n");
37.          wait(0);
38.          read(fd[0], inpipe,50);
39.          printf("%s\n", inpipe);
40.          wait(0);
41.          read(fd[0], inpipe,50);
42.          printf("%s\n", inpipe);
43.          exit(0);
44.      }
45.  }
46. }

```

实验结果截图:



```

bwq@ubuntu:~/桌面/CS$ ./pipe.out
p1
parent
p2
child 1 process is sending a message
child 2 process is sending a message

```

父进程成功收到了两个子进程的消息，并将其打印出来。

4、共享存储区通信: 用 fork 创建两个子进程，两个子进程之间使用共享存储区进行通信。

```

1. #include <stdio.h>
2. #include <sys/types.h>
3. #include <sys/shm.h>
4. #include <sys/ipc.h>
5. #include <stdlib.h>
6. #include <unistd.h>
7. #include <sys/wait.h>
8.
9. #define SHMKEY 75
10.
11. int shmids, i;
12. int *addr;

```

```

13.
14. void CLIENT(){
15.     int i;
16.     shmid = shmget(SHMKEY, 1024, 0777|IPC_CREAT);
17.     addr = shmat(shmid,0,0);
18.     for(i = 9; i >= 0; i--){
19.         while(*addr != -1);
20.         printf("(client)sent:%d\n",i);
21.         *addr=i;
22.     }
23.     exit(0);
24. }
25.
26. void SERVER(){
27.     shmid=shmget(SHMKEY, 1024, 0777|IPC_CREAT);
28.     addr = shmat(shmid,0,0);
29.     do{
30.         *addr = -1;
31.         while(*addr == -1);
32.         printf("(server)received:%d\n",*addr);
33.     }while(*addr);
34.     shmctl(shmid,IPC_RMID,0);
35.     exit(0);
36. }
37.
38. int main(){
39.     while((i=fork()) == -1);
40.     if( !i ){
41.         SERVER();
42.     }
43.     else{
44.         while((i=fork()) == -1);
45.         if(!i) {
46.             CLIENT();
47.         }else{
48.             wait(0);
49.             wait(0);
50.         }
51.     }
52.     return 0;
53. }

```

实验结果截图：

```

bwq@ubuntu:~/桌面/C$ ./shared_mem.out
(client)sent:9
(server)received:9
(client)sent:8
(server)received:8
(client)sent:7
(server)received:7
(client)sent:6
(server)received:6
(client)sent:5
(server)received:5
(client)sent:4
(server)received:4
(client)sent:3
(server)received:3
(client)sent:2
(server)received:2
(client)sent:1
(server)received:1
(client)sent:0
(server)received:0

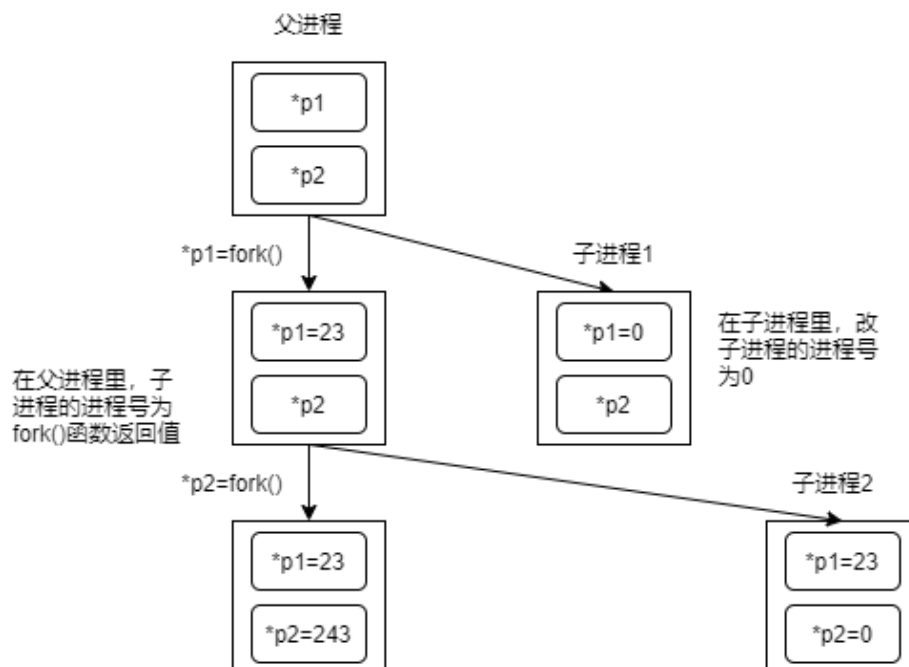
```

通过共享队列，client 进程负责发送信息，server 进程负责接受信息，client 发送的消息，在 server 中都接收到了

五、问题讨论

1、解释 fork 系统调用的工作过程，试着画出流程图。

fork()用于创建一个子进程，其返回值表示该子进程的进程号，在该子进程内部，该进程号为 0。子进程会继承父进程的变量及值，这里之所以在子进程内部的进程号与父进程中的进程号不同，是因为在子进程继承父进程变量后，再将该进程号修改为 0。



2、说明如何用消息队列发送/接收消息。

首先利用 `msgget()` 获取一个消息队列，返回其 `id` 值，一个 `id` 可以唯一标识该消息队列。

然后进程可以利用 `msgsnd()` 将消息发送至该队列，在另一个进程中，可以利用 `msgrcv()` 接收消息队列中的内容。

可以利用 `msgctl()` 控制消息队列，其形参中有 `msgid` 参数、`command` 参数等，若该参数为 `IPC_RMID` 则可以表示删除 `id` 值为 `msgid` 的消息队列。

3、说明进程间使用共享存储区发送/接收信息的过程。

两个进程使用同一个存储区作为公共存储区，一个进程作为发送方可以向存储区中存入数据，另一个进程作为接收方可以从存储区中取出数据。为了实现对存储区的互斥访问，采用 `*addr` 的值作为信号量，`*addr` 为 -1 时，对存储区写数据，`*addr` 不为 -1 时，对存储区进行读数据。

过程：

1. 使用 `shmget()` 创建一个共享存储区
2. 使用 `shmat()` 把共享内存区对象映射到调用进程的地址空间
3. 使用 `shmctl()` 对共享存储区进行操作。

4、在共享存储区通信的实验中，如何使两个进程互斥访问临界资源。

两个进程共享同一个存储区地址即 `addr`，在 `CLIENT` 中，每次循环将 `*addr` 赋值为循环变量 `i`，在 `SERVER` 中，循环中首先将 `*addr` 赋值为 -1，随后等待 `*addr` 被 `CLIENT` 修改为非 -1，随后才可以进行后续的接收操作，随后再次进入循环将 `*addr` 赋值为 -1，将此进程阻塞。在 `*addr` 为 -1 时，`CLIENT` 中的 `for` 循环中的 `while` 语句退出，可以执行后面的 `send` 操作，并将 `*addr` 赋值为 `i`。

由此，两个进程之间通过 `*addr` 的值是否为 -1 实现对共享存储区的互斥访问。