中南大學

CENTRAL SOUTH UNIVERSITY

作业三-Linux 应用程序开发报告

学生姓名	王雅蓉
学号	8206200602
专业班级	计科 2003
指导教师	沈海澜
学院	计算机学院
完成时间	2023 年 05 月 05 日

计算机学院

2023 年 05 月

作业内容及解析

一、进程控制类函数：fork();wait();waitpid()

1、fork1.c

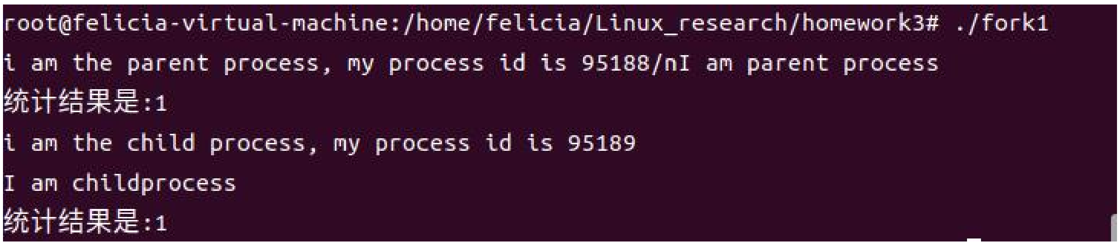
代码：

```
#include <unistd.h>
#include <stdio.h>
int main ()
{
pid_t fpid; //fpid 表示 fork 函数返回的值
int count=0;
fpid=fork();
if (fpid < 0)
printf("error in fork!");
else if (fpid == 0) {
printf("i am the child process, my process id is %d\n",getpid());
printf("I am childprocess\n");
count++;
}
else {
printf("i am the parent process, my process id is %d/n",getpid());
printf("I am parent process\n");
count++;
}
printf("统计结果是: %d/n",count);
return 0;
}
```

程序解析：

该程序主要调用了 fork 函数生成子进程 fpid。该函数在子进程中返回值为 0，父进程中返回值为子进程 pid。getpid()函数可以获取当前进程的 pid。

运行结果：



```
root@felicia-virtual-machine:/home/felicia/Linux_research/homework3# ./fork1
i am the parent process, my process id is 95188/nI am parent process
统计结果是:1
i am the child process, my process id is 95189
I am childprocess
统计结果是:1
```

2、fork2.c

代码：

```c
#include "stdio.h"
#include "sys/types.h"
#include "unistd.h"
 int main()
{
    pid_t pid1;
    pid_t pid2;

    pid1 = fork();
    pid2 = fork();

    printf("pid1:%d, pid2:%d\n", pid1, pid2);
}
```

要求如下：已知从这个程序执行到这个程序的所有进程结束这个时间段内，没有其它新进程执行。

1、请说出执行这个程序后，将一共运行几个进程。

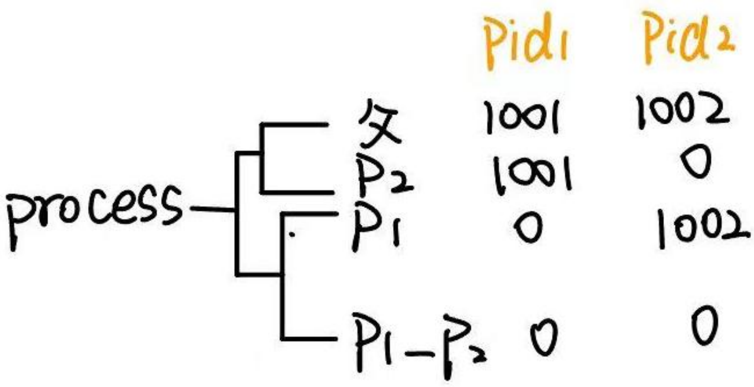
2、如果其中一个进程的输出结果是“pid1:1001, pid2:1002”，写出其他进程的输出结果（不考虑进程执行顺序）

程序解析：

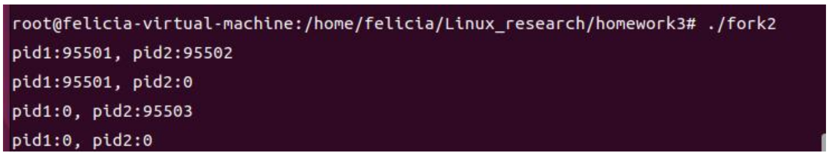
1、执行该程序后，将一共运行 4 个进程。

2、父进程：1001、1002；p2:1001、0；p1:0、1002；p1_p2:0、0；

该程序的进程生成树如下图：



运行结果：



```
root@felicia-virtual-machine:/home/felicia/Linux_research/homework3# ./fork2
pid1:95501, pid2:95502
pid1:95501, pid2:0
pid1:0, pid2:95503
pid1:0, pid2:0
```

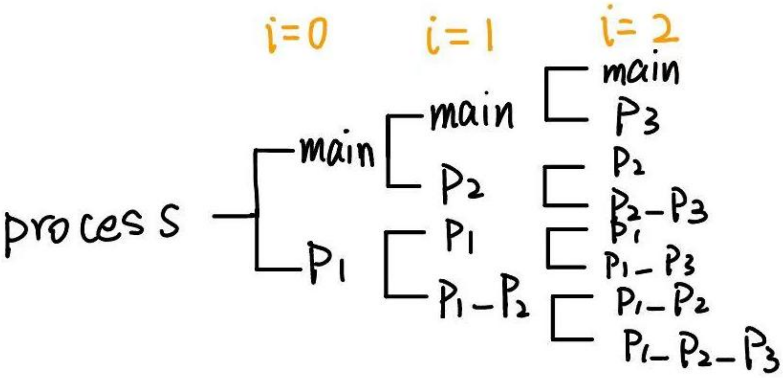
3、fork.c

代码：

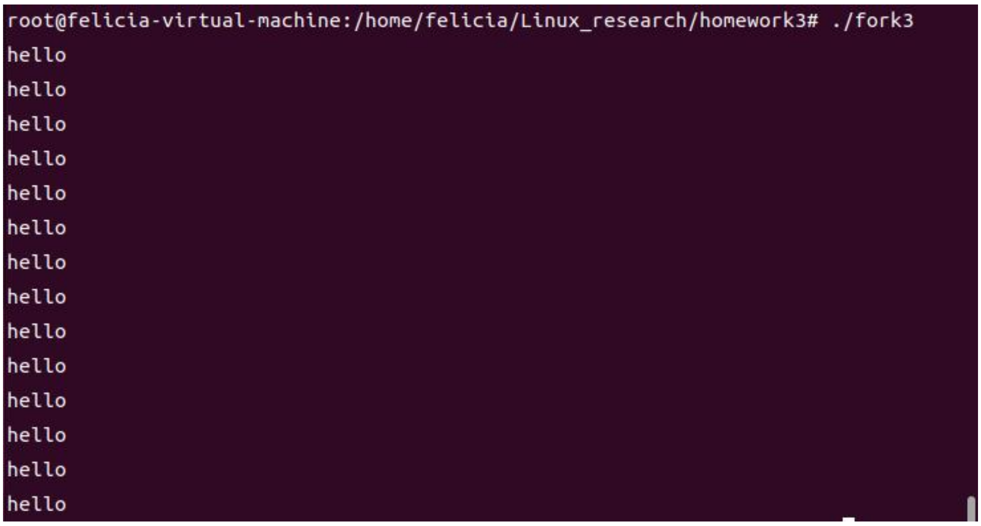
```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main(void)
{
  int i;
  for(i=0; i<3; i++){
    fork();
    printf("hello\n");
  }
  return 0;
}
```

程序解析：

该程序一共会输出 14 个 hello。可以将该进程生成的进程树描绘出来。由下图可知在每一个结点都会输出 hello，故一共输出 14 次 hello。



运行结果：



```
root@felicia-virtual-machine:/home/felicia/Linux_research/homework3# ./fork3
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
```

4、Wait.c

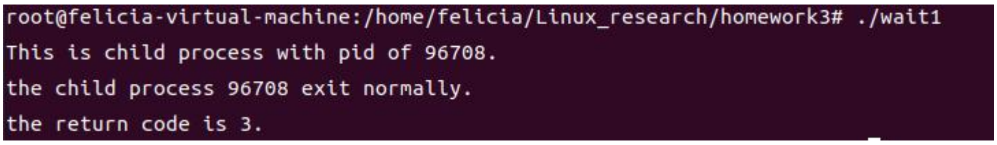
代码：

```
/* wait2.c */
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
int main()
{
int status;
pid_t pc,pr;
pc = fork();
if (pc < 0)
printf("error ocurred!\n");
else if (pc == 0)
{
printf("This is child process with pid of %d.\n",getpid());
exit(3);
}
else
{
pr = wait(&status);
if (WIFEXITED(status))
{
printf("the child process %d exit normally.\n",pr);
printf("the return code is %d.\n",WEXITSTATUS(status));
}
else
printf("the child process %d exit abnormally.\n",pr);
}
}
```

程序解析：

该程序调用 wait 函数，父进程等待子进程结束，并获取结束状态 status。判断子进程是否是正常结束。

运行结果：



```
root@felicia-virtual-machine:/home/felicia/Linux_research/homework3# ./wait1
This is child process with pid of 96708.
the child process 96708 exit normally.
the return code is 3.
```

5、Waitpid.c

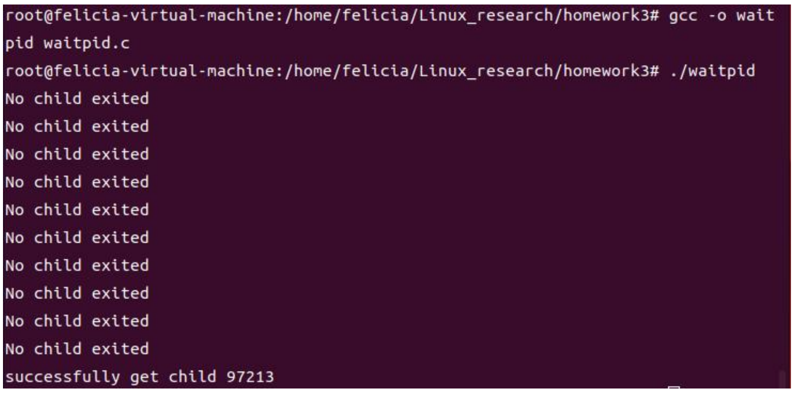
代码：

```c
/* waitpid.c */
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
int main()
{
pid_t pc, pr;
pc = fork();
if (pc < 0)
printf("Error occured on forking.\n");
else if (pc == 0)
{
sleep(10);
exit(0);
}
else
do
{
pr = waitpid(pc, NULL, WNOHANG);
if (pr == 0)
{
printf("No child exited\n");
sleep(1);
}
}
while (pr == 0);
if (pr == pc)
printf("successfully get child %d\n", pr);
else
printf("some error occured\n");
}
```

程序解析：

该程序调用了 waitpid 函数，父进程一直循环等待子进程结束。Pr 为 waitpid 的返回值，如果子进程没有立刻结束则返回值为 0，否则为子进程 id。

运行结果：



```
root@felicia-virtual-machine:/home/felicia/Linux_research/homework3# gcc -o wait
pid waitpid.c
root@felicia-virtual-machine:/home/felicia/Linux_research/homework3# ./waitpid
No child exited
No child exited
No child exited
No child exited
No child exited
No child exited
No child exited
No child exited
No child exited
No child exited
successfully get child 97213
```

二、信号

6、Signal.c

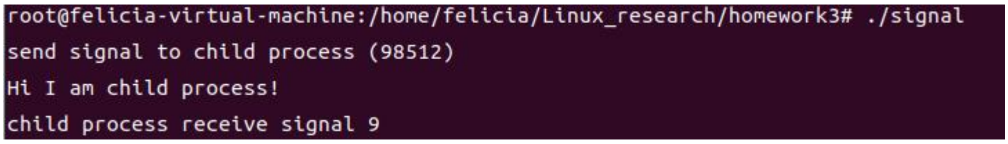
代码：

```
#include <unistd.h>
#include <signal.h>
#include <sys/types.h>
#include<sys/wait.h>

main()
{pid_t pid;
int status;
if(!(pid=fork()))
 {
  printf("Hi I am child process!\n");
  sleep(100);
  return;
 }
else
 {printf("send signal to child process (%d)\n",pid);
  sleep(1);
  kill(pid ,SIGKILL);
  wait(&status);
  if(WIFSIGNALED(status))
    printf("chile process receive signal %d",WTERMSIG(status));
 }
}
```

程序解析：

父进程调用 kill 函数发送进程终止命令给子进程。同时判断子进程结束是否因为信号而结束。

运行结果：



```
root@felicia-virtual-machine:/home/felicia/Linux_research/homework3# ./signal
send signal to child process (98512)
Hi I am child process!
child process receive signal 9
```

三、信号量

7、Semphore1.c

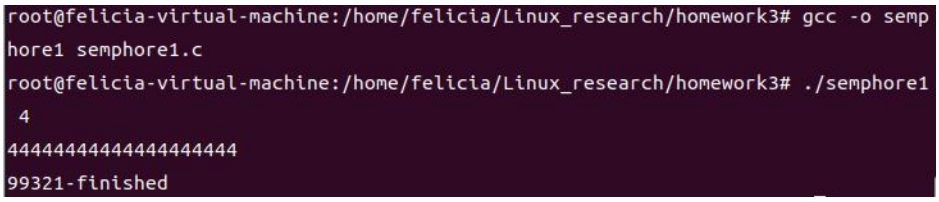
代码：

```c
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sys/sem.h>

union semun
{
    int val;
    struct semid_ds *buf;
    unsigned short *arry;
};

static int sem_id = 0;

static int set_semvalue();
static void del_semvalue();
static int semaphore_p();
static int semaphore_v();

int main(int argc, char *argv[])
{
    char message = 'X';
    int i = 0;

    //创建信号量
    sem_id = semget((key_t)1234, 1, 0666 | IPC_CREAT);

    if(argc > 1)
    {
        //程序第一次被调用，初始化信号量
        if(!set_semvalue())
        {
            fprintf(stderr, "Failed to initialize semaphore\n");
            exit(EXIT_FAILURE);
        }
        //设置要输出到屏幕中的信息，即其参数的第一个字符
        message = argv[1][0];
        sleep(2);
    }
    for(i = 0; i < 10; ++i)
    {
        //进入临界区
        if(!semaphore_p())
            exit(EXIT_FAILURE);
        //向屏幕中输出数据
        printf("%c", message);
        //清理缓冲区，然后休眠随机时间
        fflush(stdout);
        sleep(rand() % 3);
        //离开临界区前再一次向屏幕输出数据
        printf("%c", message);
        fflush(stdout);
        //离开临界区，休眠随机时间后继续循环
        if(!semaphore_v())
            exit(EXIT_FAILURE);
        sleep(rand() % 2);
    }

    sleep(10);
    printf("\n%d - finished\n", getpid());

    if(argc > 1)
    {
        //如果程序是第一次被调用，则在退出前删除信号量
        sleep(3);
        del_semvalue();
    }
    exit(EXIT_SUCCESS);
}

static int set_semvalue()
{
    //用于初始化信号量，在使用信号量前必须这样做
    union semun sem_union;

    sem_union.val = 1;
    if(semctl(sem_id, 0, SETVAL, sem_union) == -1)
        return 0;
    return 1;
}

static void del_semvalue()
{
    //删除信号量
    union semun sem_union;

    if(semctl(sem_id, 0, IPC_RMID, sem_union) == -1)
        fprintf(stderr, "Failed to delete semaphore\n");
}

static int semaphore_p()
{
    //对信号量做减 1 操作，即等待 P（sv）
    struct sembuf sem_b;
    sem_b.sem_num = 0;
    sem_b.sem_op = -1;//P()
    sem_b.sem_flg = SEM_UNDO;
    if(semop(sem_id, &sem_b, 1) == -1)
    {
        fprintf(stderr, "semaphore_p failed\n");
        return 0;
    }
    return 1;
}

static int semaphore_v()
{
    //这是一个释放操作，它使信号量变为可用，即发送信号 V（sv）
    struct sembuf sem_b;
    sem_b.sem_num = 0;
    sem_b.sem_op = 1;//V()
    sem_b.sem_flg = SEM_UNDO;
    if(semop(sem_id, &sem_b, 1) == -1)
    {
        fprintf(stderr, "semaphore_v failed\n");
        return 0;
    }
    return 1;
}
```

程序解析：

用信号量控制，每次输出两个指定字符，最后一共输出 20 个字符。

运行结果：



```
root@felicia-virtual-machine:/home/felicia/Linux_research/homework3# gcc -o semp
hore1 semphore1.c
root@felicia-virtual-machine:/home/felicia/Linux_research/homework3# ./semphore1
 4
44444444444444444444
99321-finished
```

8、nosemphore2.c

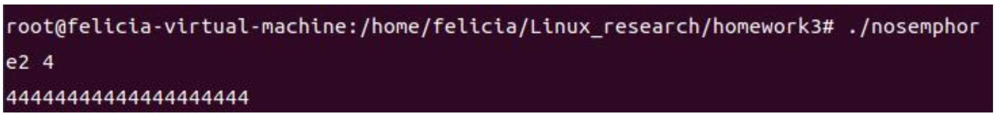
代码：

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    char message = 'X';
    int i = 0;
    if(argc > 1)
        message = argv[1][0];
    for(i = 0; i < 10; ++i)
    {
        printf("%c", message);
        fflush(stdout);
        sleep(rand() % 3);
        printf("%c", message);
        fflush(stdout);
        sleep(rand() % 2);
    }
    sleep(10);
    printf("\n%d - finished\n", getpid());
    exit(EXIT_SUCCESS);
}
```

程序分析：

该程序没有使用信号量，输出会被打断，因此最终输出的字符个数可能不是 20 个。

运行结果：



```
root@felicia-virtual-machine:/home/felicia/Linux_research/homework3# ./nosemphor
e2 4
44444444444444444444
```

四、管道通信

9、pipe.c

```
#include<sys/types.h>
#include<unistd.h>
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int main()
{
 int d1[2];
 int d2[2];
 int d3[2];
 int r,j,k;
 char buff[200];
 printf("please input a string:");
 scanf("%s",buff);
 r=pipe(d1);
 if(r==-1)
 {
  printf("chuangjianguandaoshibai 1\n");
  exit(1);
 }

 r=pipe(d2);
 if(r=-1)
 {
  printf("chuangjianguandaoshibai 2\n");
  exit(1);
 }

 r=pipe(d3);
 if(r==-1)
 {
  printf("chuangjianguandaoshibai 3\n");
  exit(1);
 }

 r=fork();
 if(r)
 {
  close(d1[1]);
  read(d1[0],buff,sizeof(buff));
  if(strlen(buff)%2==1)
  {
   j=fork();
   if(j)
```

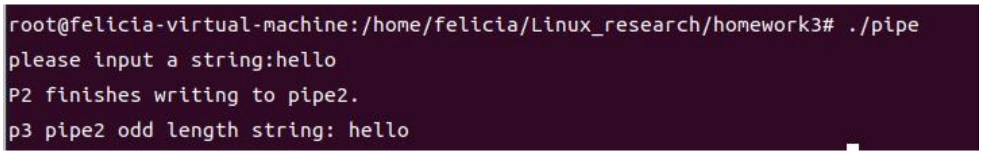
```
     {
       close(d2[1]);
       read(d2[0],buff,sizeof(buff));
       printf("p3 pipe2 odd length string: %s\n",buff);
       close(d2[0]);
       exit(0);
     }
    else
    {
       close(d2[0]);
       write(d2[1],buff,strlen(buff));
       printf("P2 finishes writing to pipe2.\n");
       close(d2[1]);
       exit(0);
    }
  }
  else
  {
    k=fork();
    if(k)
    {
       close(d3[1]);
       read(d3[0],buff,sizeof(buff));
       printf("P4 pipe3 even length string:%s\n",buff);
       close(d3[0]);
       exit(0);
    }
    else
    {
      close(d3[0]);
      write(d3[1],buff,strlen(buff));
      printf("P2 finishes writing to pipe3.\n");
      close(d3[1]);
      exit(0);
    }
  }
}
else
{
  close(d1[0]);
  write(d1[1],buff,strlen(buff));
  close(d1[1]);
  exit(0);
}
```

```
}
```

程序解析：

主要创建了无名管道，并对其进行读写。

运行结果：



```
root@felicia-virtual-machine:/home/felicia/Linux_research/homework3# ./pipe
please input a string:hello
P2 finishes writing to pipe2.
p3 pipe2 odd length string: hello
```

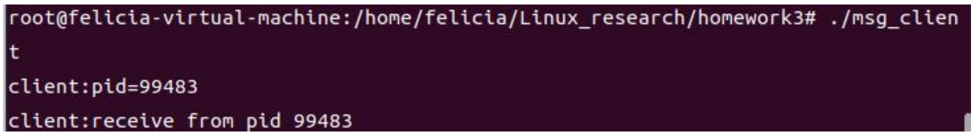
五、消息队列通信

10、msg_Client.c

代码：

```
// 客户端程序 msg_client.c
# include <sys/types.h>
# include <sys/ipc.h>
# include <sys/msg.h>
# include <stdio.h>
# include <unistd.h>
# define MSGKEY 75
struct msgform
{
  long mtype;
  char mtext[256];
};

Int main()
{
  struct msgform msg;
  int msgqid,pid,*pint;
  msgqid=msgget(MSGKEY,0777);
  pid=getpid();
  printf(“client:pid=%d\n”,pid);
  pint=(int*)msg.mtext;
  *pint=pid;
  msg.mtype=1;
  msgsnd(msgqid,&msg,sizeof(int),0);
  msgrcv(msgqid,&msg,256,pid,0);
  printf(“client:receive from pid%d\n”,*pint);
}
```

运行结果：

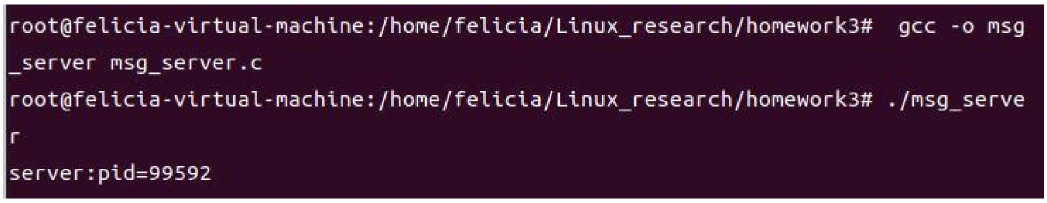


```
root@felicia-virtual-machine:/home/felicia/Linux_research/homework3# ./msg_clien
t
client:pid=99483
client:receive from pid 99483
```

11、msg_server.c

```c
/* 服务端程序 msg_server.c*/
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <signal.h>

#define MSGKEY 75
struct msgform
{
    long mtype;
    char mtext[256];
};
int msgqid;

void cleanup()
{
    msgctl(msgqid, IPC_RMID, 0); /*删除队列*/
    exit(0);
}

int main()
{
    struct msgform msg;
    int pid, *pint, i;

    // extern cleanup();
    for (i = 0; i < 23; i++)
        signal(i, cleanup);
    msgqid = msgget(MSGKEY, 0777 | IPC_CREAT);
    printf("server : pid = % d\n", getpid());
    for (;;)
    {
        msgrcv(msgqid, &msg, 256, 1, 0);
        pint = (int *)msg.mtext;
        pid = *pint;
        printf("server: receive from pid %d\n", pid);
        msg.mtype = pid; /*将接收的客户进程的 pid 为消息类型*/
        *pint = getpid();
        msgsnd(msgqid, &msg, sizeof(int), 0);
    }
}
```

运行结果：



```
root@felicia-virtual-machine:/home/felicia/Linux_research/homework3#  gcc -o msg
_server msg_server.c
root@felicia-virtual-machine:/home/felicia/Linux_research/homework3# ./msg_serve
r
server:pid=99592
```

六、共享存储通信

12、shm.c

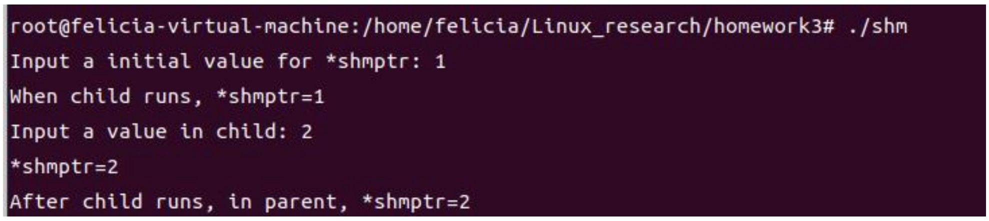
```c
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/shm.h>

int main(void)
{
    int x, shmid;
    int *shmptr;
    if((shmid=shmget(IPC_PRIVATE, sizeof(int), IPC_CREAT|0666)) < 0)
        printf("shmget error"), exit(1);
    if((shmptr=(int *)shmat(shmid, 0, 0)) == (int *)-1)
        printf("shmat error"), exit(1);
    printf("Input a initial value for *shmptr: ");
    scanf("%d", shmptr);
    while((x=fork())==-1);
    if(x==0)
    {
        printf("When child runs, *shmptr=%d\n", *shmptr);
        printf("Input a value in child: ");
        scanf("%d", shmptr);
        printf("*shmptr=%d\n", *shmptr);
    }
    else
    {
        wait();
        printf("After child runs, in parent, *shmptr=%d\n", *shmptr);
    if ( shmctl(shmid, IPC_RMID, 0) < 0 )
```

```
        printf("shmctl error"), exit(1);
    }
        return 0;
}
```

程序解析：

子进程往共享存储中存储输入的数据，此时在父进程中该位置存储的数据也相应改变。

运行结果：



```
root@felicia-virtual-machine:/home/felicia/Linux_research/homework3# ./shm
Input a initial value for *shmptr: 1
When child runs, *shmptr=1
Input a value in child: 2
*shmptr=2
After child runs, in parent, *shmptr=2
```

七、线程

13、thread.c

代码：

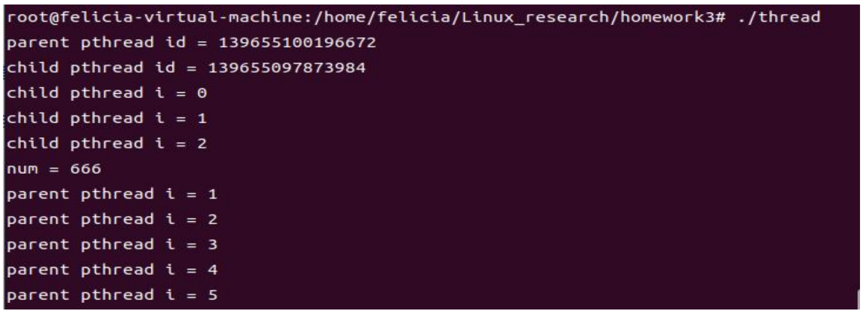
```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

int num = 100;

// myfunc 线程
void *myfunc(void *arg)
{
    printf("child pthread id = %ld\n", pthread_self());
    for (int i = 0; i < 5; i++)
    {
        printf("child pthread i = %d\n", i);
        if (i == 2)
        {
            num = 666; // 验证不同线程可以利用全局变量通信
            // pthread_exit(NULL);  // 不携带数据的退出
            pthread_exit(&num); // 携带数据的退出
        }
    }
    return 0;
}
```

```

int main()
{
    int ret;
    int i = 0;
    pthread_t pthid;
    ret = pthread_create(&pthid, NULL, myfunc, NULL); // 线程创建

    if (ret != 0) // 创建失败判断
    {
        printf("error number is %d\n", ret);
        printf("%s\n", strerror(ret));
    }
    printf("parent pthread id = %ld\n", pthread_self());

    void *ptr = NULL;
    ptr = malloc(32);      // 如果不动态申请内存会发生段错误
    void *tmp = ptr;    // 用 tmp 指向申请的内存来操作内存，以防改变 ptr 的指向导致 free 时产生段错误
    pthread_join(pthid, &tmp); // 第二个参数是 void * 类型的二级指针，可以用来获取 exit 参数携带的数据
    printf("num = %d\n", *(int *)tmp);
    free(ptr);  // 释放掉申请的内存
    ptr = NULL; // 指针指向 NULL 以防后续误操作

    while (i < 5)
    {
        i++;
        printf("parent pthread i = %d\n", i);
    }
    sleep(2);
 return 0;
}
```

程序解析：

第三个子进程携带数据退出时 num 的值改变，说明不同线程可以利用全局变量通信。

运行结果：



```
root@felicia-virtual-machine:/home/felicia/Linux_research/homework3# ./thread
parent pthread id = 139655100196672
child pthread id = 139655097873984
child pthread i = 0
child pthread i = 1
child pthread i = 2
num = 666
parent pthread i = 1
parent pthread i = 2
parent pthread i = 3
parent pthread i = 4
parent pthread i = 5
```