

实验二 进程和进程通信

学号: 515030910067 姓名: 杨超琪 日期: 2018.5.18

一、实验题目

1. 设计一个程序, 创建一个子进程, 使父子进程合作, 协调地完成某一功能。要求在该程序中还要使用进程的睡眠、进程图象改换、父进程等待子进程终止、信号的设置与传送(包括信号处理程序)、子进程的终止等有关进程的系统调用。

2. 分别利用 UNIX 的消息通信机制、共享内存机制(用信号灯实施进程间的同步和互斥) 实现两个进程间的数据通信。具体的通信数据可从一个文件读出, 接收方进程可将收到的数据写入一个新文件, 以便能判断数据传送的正确性

二、算法设计思路

1. 父子进程合作交互

第一个实验是由父进程调用系统 `fork()` 函数创建一个子进程, 子进程将会复制父进程的扩充控制块, 然后继续从 `fork()` 函数下方开始执行。其中, 子进程将共享父进程的全部打开的文件、信号处理方式等。子进程复制了父进程的数据区、核心栈和用户栈。父子进程程序执行的当前位置、状态、数据区、变量的当前值都是相同的。唯一的区别是父进程的 `fork()` 返回的是子进程的 `pid`, 而子进程的 `fork()` 返回的是 0。

进程映像的改换指的是为了配合 `fork()`, 系统提供了进程映像改换的 `exec` 系列的系统调用(在这里我们主要使用 `execl`)。主要指的是进程用一个可执行文件中的程序和数据取代当前正在运行的程序和数据, 从而使主进程的影响改换成新的映像。

具体思路如下(见图 1):

- 1) 首先父进程初始化 `status`、`pid` 参数与信号处理函数 `sigFunc`;
- 2) 父进程关联信号与信号处理函数;
- 3) 父进程调用 `fork` 函数申请创建一个子进程;
- 4) 父进程的 `fork` 返回子进程的 `pid`, 子进程的 `fork` 返回 0, 因此父

子进程分别进入 if 条件判断语句的两个模块；

- 5) 子进程进入模块沉睡 3 秒钟让父进程先运行，打印子进程状态信息，之后调用 excel 进程进程图像改换，命令结束后子进程自结束，若出错，则打印出错信息，调用 exit 结束；
- 6) 父进程调用 kill 向子进程发送信号，子进程创建时调用信号处理函数，父进程等待子进程结束获得返回状态信息，并打印；
- 7) 父进程结束。

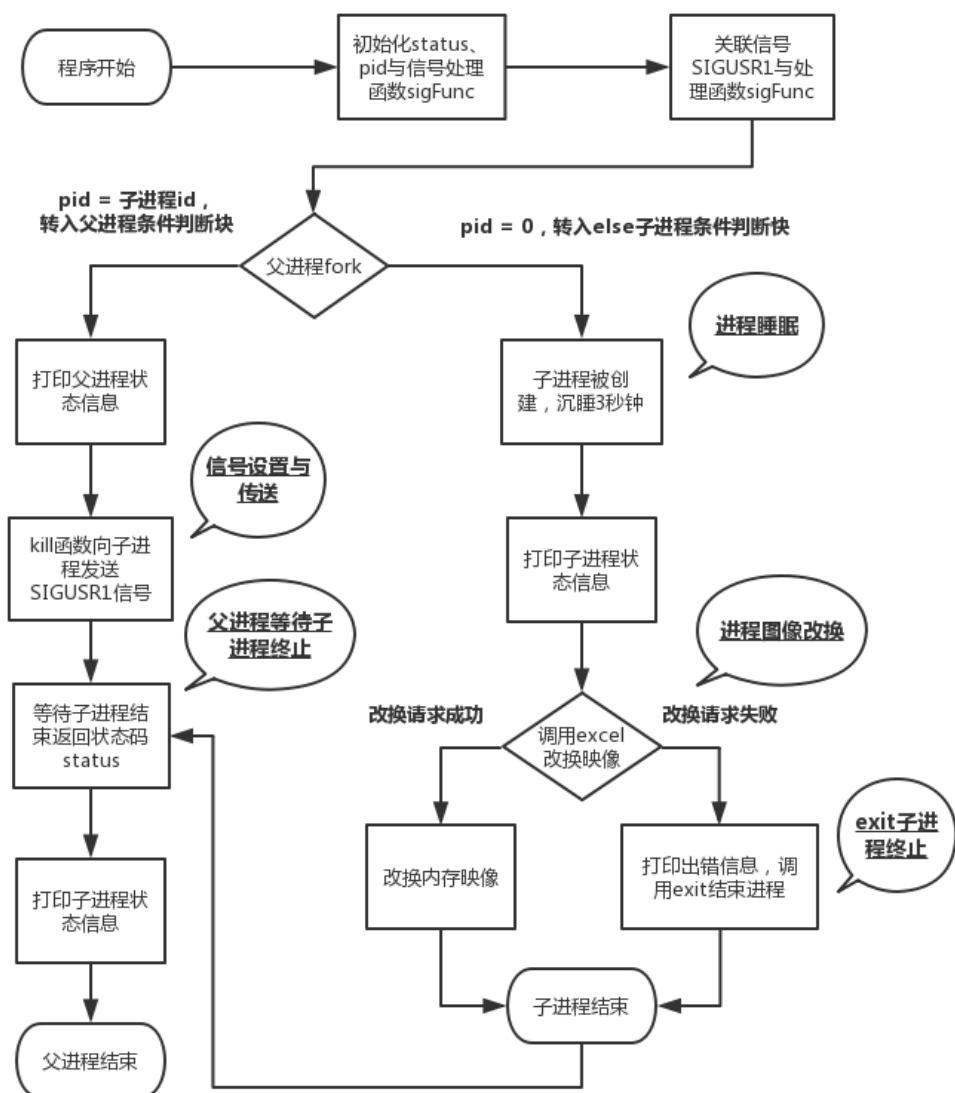


图1：父子进程合作交互流程图

2. 消息队列

消息通信的基本思想是由系统消息通信机构统一管理的一组空闲的消息缓冲区，当一个进程要向另一个进程发送消息，先要向系统申请一个缓冲区，填写消息正文和其他有关消息的特征和控制信息以后，通过消息通信机构将该消息送到接收进程的消息队列中。当接收进程需要接受该消息的时候，就从消息队列中移出一个消息，读取信息以后，在释放消息缓冲区。

消息缓冲区中需要包含消息和相应的控制信息，为了支持消息的发送与接收，操作系统还需要提供 `msgsnd` 和 `msgrcv` 两个系统调用函数。

具体思路如下（见图 2）

Server 端:

- 1) 首先由头文件 `msgcom.h` 定义 `buf` 结构,初始化 `pid` 等其他参数;
- 2) 从消息队列中获取消息;
- 3) 若获取失败则推出, 否则调用 `msgrcv` 将消息按照固定大小逐块存入 `buf` 中, 打印到频幕上, 并且存到 `output.txt` 文件中;
- 4) 若存取失败则推出, 否则循环接受, 直到消息接受完毕, 退出。

Client 端:

- 1) 首先由头文件 `msgcom.h` 定义 `buf` 结构,初始化 `pid` 等其他参数;
- 2) 打开 `input.txt` (事先存好要发送的消息), 若打开错误, 则直接退出, 否则将文件中的内容打印到频幕上, 并按固定大小加入消息队列;
- 3) 循环将固定大小的消息加入消息队列, 直到消息发送完毕;
- 4) 发送完毕后, 客户端调用 `exit` 退出。

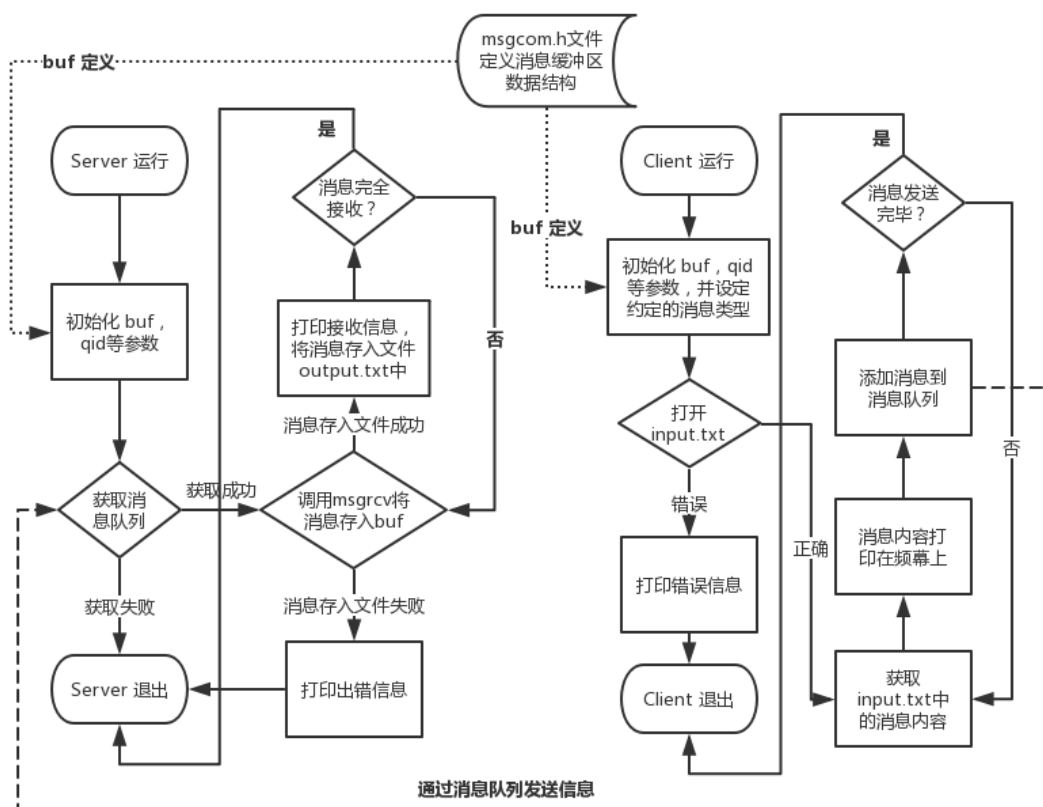


图2：通过消息队列Server和Client进行消息通信

3. 共享内存

共享内存是一种通信机制，可以把内存中的一个区域连入多个进程的虚拟空间，当一个进程把消息写入共享内存以后，另一个内存就可以从中进行存取。

进程请求分配一个共享内存区域端的时候，核心先按照进程相对的共享内存区的大小和存取控制的要求分配空闲页表区和一个空闲内存块，填入共享内存关键字，大小等控制信息，返回描述符。

具体思路如下（见图3）：

- 1) 首先程序设定了一些全局参数，比如：segaddr，segid 等；
- 2) 申请一块共享内存，若申请失败则打印错误信息，程序结束；
- 3) 将共享内存段映射到数据段，并创建两个信号灯 sid1，sid2，初始值分别为 1，0，代表父进程可存入数据，子进程不可访问；
- 4) 父进程调用 fork 创建一个子进程；

- 5) 此时子进程与父进程进入不同的流程；
- 6) 子进程打开文件 `output2.txt`，设置信号量 `sid1`，使子进程无法访问，之后将共享内存中的数据读入文件，再设置信号量 `sid2`，告知父进程可以进行数据更改了，循环；
- 7) 父进程打开文件 `Input2.txt`，设置信号量 `sid2`，令键盘键入消息，将消息存入共享内存，设置信号来量 `sid1`，告知子进程可以获得新的信息了，循环；
- 8) 如果子进程或者父进程文件打开失败，将会直接导致程序的结束。

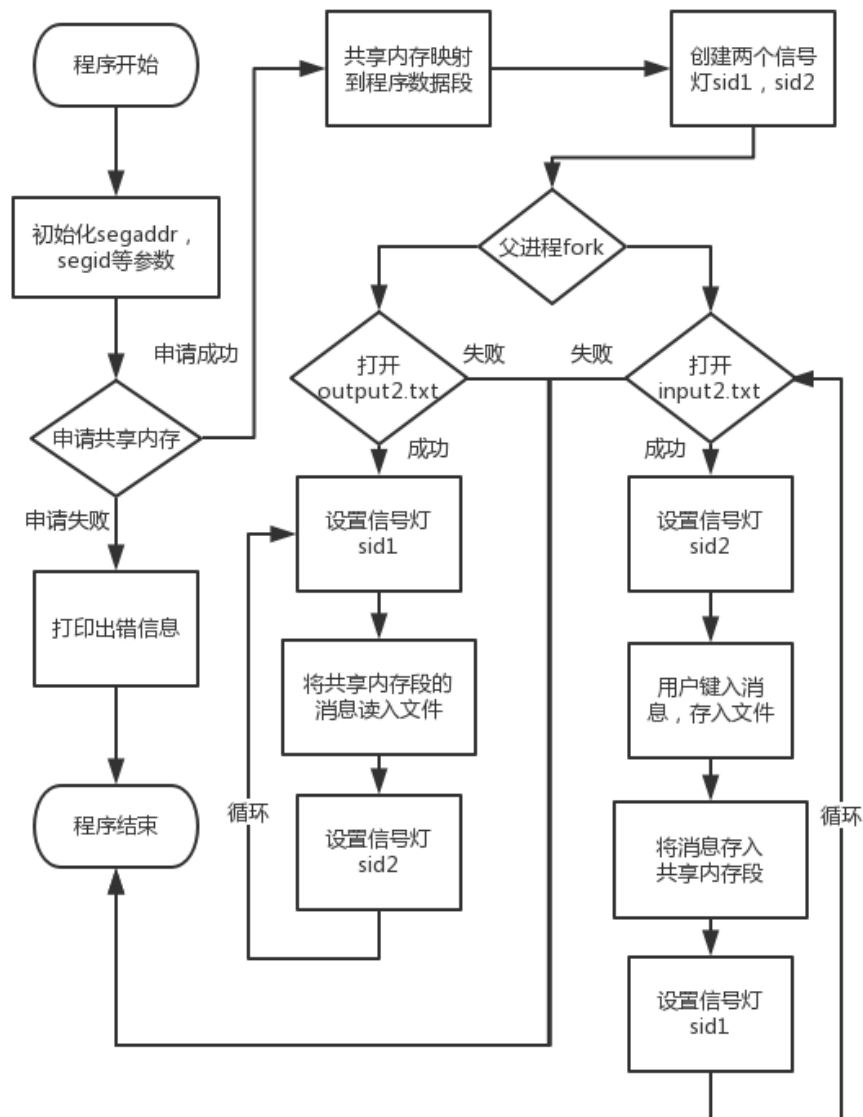


图3：共享内存消息通信

三、模块设计、功能和接口说明

1. 父子进程合作交互

signal(SIGUSR1, sigFunc): 此模块的功能是将信号与函数相关联，一旦信号 SIGUSR1 触发，将会运行函数 sigFunc 函数。SIGUSR1 是我们自己设定的信号标识，不可以取 SIGKILL 和 SIGTOP；而 sigFunc 在这里设定为一个屏幕打印程序，一旦信号触发，将会在屏幕上进行提示。

kill(pid, SIGUSR1): kill 函数不是平常设想的杀掉一个进程的意思，在这里，kill 函数将会把 SIGUSR1 这个信号传递给 pid 这个子进程。

wait(& status): 将会接受子进程推出以后返回的状态信息，存储到 status 中。

execl("/bin/ls", "ls", "-l", (char*)0): 是映像改换函数，将会改变到/bin/ls 目录下执行 ls -l 命令，将此部分的程序代码覆盖原先程序下的代码，叫做映像改换。

2. 消息队列

msgget(MSGKEY, IPC_CREAT|0666): 这个函数用于生成消息队列，并且返回消息队列的 ID。其中的 MSGKEY 为关键字，已在头文件中规定好，IPC_CREAT|0666 则设定了操作的权限，为用户，组，其他均可读可访问，不可执行。

msgsnd(qid, &buf, sizeof(buf.msg), 0): 这个函数用于发送消息，qid 是消息的队列的 ID，&buf 是将要发送的内容，sizeof(buf.msg) 指的是发送消息的长度，0 表示发送的一个标识，如果消息队列满时，进程将会阻塞。

msgrcv(qid, &buf, 1000, 5679, MSG_NOERROR): 这个函数用于接受消息，qid 表示消息的队列的 ID，接收到的内容将会存入 &buf，一次存入 1000 长度的内容，接受消息的类型为 5679，MSG_NOERROR 表示允许接受的长度小于消息的正文段长度。

3. 共享内存

shmget(SHMKEY, SIZE, IPC_CREAT|0666): 这个函数是用于申请共享内存段，其中 SHMKEY 为关键词，SIZE 是申请的大小，IPC_CREAT|0666 表示操作的权限，为用户，组，其他均可读可访问，不可执行。

shmat(segid,0,0): 这个函数将共享内存段的地址进行返回,方便在程序中进行操作。

creatsem(key): 这个函数用于创建初始值为 1 的信号量。

P(sid): 这个函数将会调用 `semcall`,降低信号量 1,与我们所学的 `semWait` 类似,如果此时信号量被减小到小于 0,进程被阻塞。

V(sid): 这个函数将会调用 `semcall`,增加信号量 1,与我们所学的 `semSignal` 类似,如果此时信号量被小于等于 0,进程恢复。

四、重要数据结构和变量说明

1. 父子进程合作交互

Status: 用于存放子进程返回的状态信息(对于这一点,将会在后续问题探讨中进行详细说明)。

Pid: 保存进程 ID,对于父进程来说,将会保存子进程的 ID,对于子进程来说,将会是 0。

2. 消息队列

```
struct msgtype{           // 定义了一个消息的结构体
    long mtype;           // 消息类型的定义
    char msg[1000];       // 消息内容的存储
};
```

buf: 在子进程与父进程中都有的数据结构,用于存储消息片段的所有信息,由 `msgtype` 定义。

qid: 用于存放消息队列的 ID,方便后序 `Server` 与 `Client` 消息队列交互。

MSGKEY: 消息队列的关键词,只有关键词一致才可以进行消息的传递。

3. 共享内存

SHMKEY: 共享内存的关键词。

SEMKEY1: 第一组信号灯的关键词。

SEMKEY2: 第二组信号灯的关键词。

segaddr: 一个字符指针，用于之后绑定共享内存地址。

segid: 共享内存的 ID，将与 segaddr 进行绑定。

sid1: 第一组信号量的值。

sid2: 第二组信号量的值。

五、测试方法与分析

1. 父子进程合作交互

在测试过程中，首先父进程调用 `fork()` 创建一个子进程，我们可以看到，父进程的 `pid` 为 1998，子进程的 `pid` 为 1999，父进程调用 `kill` 将信号传递给子进程，之后子进程运行以后，信号处理函数运行，子进程先进程睡眠，父进程等待子进程终止，子进程进入图像改换以后，终止执行，父进程通过 `wait` 获得返回的 `status`，打印在频幕。测试成功。

```
root@ubuntu:/home/linuxvirtualpc1/lab2# gcc -o test1 test1.c
root@ubuntu:/home/linuxvirtualpc1/lab2# ls
a      client  input2.txt  msgcom.h    output.txt  server.c    test1
a.out  client.c  input.txt   output2.txt  server      share_memory.c  test1.c
root@ubuntu:/home/linuxvirtualpc1/lab2# ./test1
The pid of the Father is 1998.
The pid of the Son is 1999.
Father send a signal to the son.
:::It is signal processing function.
The Son receives the signal and triggers the signal function.
After 3 second's sleep...
The Son change the Image in the inner storage.
total 100
-rwxr-xr-x 1 root root 13536 May 20 06:21 a
-rwxr-xr-x 1 root root 8824 May 20 01:54 a.out
-rwxr-xr-x 1 root root 9024 May 20 03:40 client
-rw-r--r-- 1 root root 674 May 20 03:38 client.c
-rw-r--r-- 1 root root 90 May 20 06:22 input2.txt
-rw-r--r-- 1 root root 216 May 20 04:04 input.txt
-rw-r--r-- 1 root root 156 May 20 03:40 msgcom.h
-rw-r--r-- 1 root root 91 May 20 06:22 output2.txt
-rw-r--r-- 1 root root 1270 May 20 04:04 output.txt
-rwxr-xr-x 1 root root 9008 May 20 03:40 server
-rw-r--r-- 1 root root 884 May 20 03:38 server.c
-rw-r--r-- 1 root root 2616 May 20 06:21 share_memory.c
-rwxr-xr-x 1 root root 9152 May 21 19:35 test1
-rw-r--r-- 1 root root 1110 May 21 19:35 test1.c
Status=0, Father process finished.
root@ubuntu:/home/linuxvirtualpc1/lab2#
```


2. 消息队列

在消息队列的方法中，Client 使用 input.txt 中的文本作为消息传送，利用消息队列的方式，Server 通过关键字接收到消息，并存入 output.txt 文件中，可以看到两者的内容是一样的。测试成功。

```
root@ubuntu:/home/linuxvirtualpc1/lab2# gcc -o server server.c
root@ubuntu:/home/linuxvirtualpc1/lab2# gcc -o client client.c
root@ubuntu:/home/linuxvirtualpc1/lab2# cat input.txt
Here is the example Message:
The Client successfully open the input.txt,
and it sends this message to the Server.
If the Server has successfully received this message,
you can check in the output.txt!
root@ubuntu:/home/linuxvirtualpc1/lab2# ./client
Client has sent the following Message to the Server.
#####
Here is the example Message:
#####

Client has sent the following Message to the Server.
#####
The Client successfully open the input.txt,
#####

Client has sent the following Message to the Server.
#####
and it sends this message to the Server.
#####

Client has sent the following Message to the Server.
#####
If the Server has successfully received this message,
#####

Client has sent the following Message to the Server.
#####
you can check in the output.txt!
#####
```

```
root@ubuntu:/home/linuxvirtualpc1/lab2# ./server
Server has received some data from the Client:
#####
Here is the example Message:
#####

Server has received some data from the Client:
#####
The Client successfully open the input.txt,
#####

Server has received some data from the Client:
#####
and it sends this message to the Server.
#####

Server has received some data from the Client:
#####
If the Server has successfully received this message,
#####

Server has received some data from the Client:
#####
you can check in the output.txt!
#####
```

```
root@ubuntu:/home/linuxvirtualpc1/lab2# cat output.txt
Here is the example Message:
The Client successfully open the input.txt,
and it sends this message to the Server.
If the Server has successfully received this message,
you can check in the output.txt!
root@ubuntu:/home/linuxvirtualpc1/lab2# cat input.txt
Here is the example Message:
The Client successfully open the input.txt,
and it sends this message to the Server.
If the Server has successfully received this message,
you can check in the output.txt!
root@ubuntu:/home/linuxvirtualpc1/lab2#
```

3. 共享内存

在这个实验中，我们采用用户键盘输入，子进程先将文本保存在 input2.txt 文件中，再将其存入共享内存，父进程从共享内存中取出消息，存放在 output2.txt 中。测试成功。

同时，值得一提的是，我们可以看到，在子进程与父进程同时进入运行时，子进程在等待用户的外部输入，而父进程率先将原本保留在共享内存中的 hello 读入了文件。

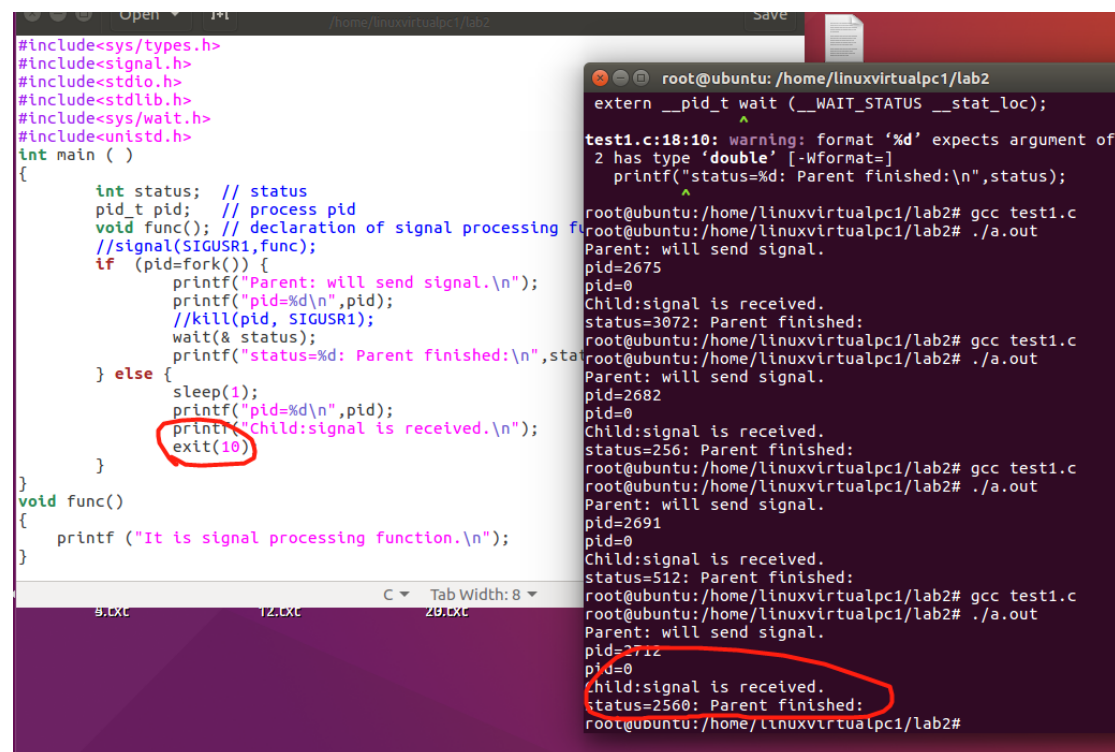
```
root@ubuntu:/home/linuxvirtualpc1/lab2# gcc -o share_memory share_memory.c
root@ubuntu:/home/linuxvirtualpc1/lab2# ./share_memory
The Son has received Message from the Father: hello
Input the Message that you want to send:testMessage1
The Son has received Message from the Father: testMessage1
Input the Message that you want to send:testMessage2
The Son has received Message from the Father: testMessage2
Input the Message that you want to send:^C
root@ubuntu:/home/linuxvirtualpc1/lab2#
```

```
root@ubuntu:/home/linuxvirtualpc1/lab2# cat input2.txt
testMessage1
testMessage2
root@ubuntu:/home/linuxvirtualpc1/lab2# cat output2.txt
hello
testMessage1
testMessage2
```

六、程序及测试的改进与体会

1、存在不足及展望：

1) 首先是存在一个问题，在测试程序的时候，当子进程调用 `exit(x)`；退出的时候，父进程的 `wait` 函数调用，将会接受子进程提供的返回参数，但是很奇怪的是，父进程接收到的将是 `256x`，例如：



```
#include<sys/types.h>
#include<signal.h>
#include<stdio.h>
#include<stdlib.h>
#include<sys/wait.h>
#include<unistd.h>
int main ( )
{
    int status; // status
    pid_t pid; // process pid
    void func(); // declaration of signal processing function
    //signal(SIGUSR1,func);
    if (pid=fork()) {
        printf("Parent: will send signal.\n");
        printf("pid=%d\n",pid);
        //kill(pid, SIGUSR1);
        wait(& status);
        printf("status=%d: Parent finished:\n",status);
    } else {
        sleep(1);
        printf("pid=%d\n",pid);
        printf("Child:signal is received.\n");
        exit(10);
    }
}
void func()
{
    printf ("It is signal processing function.\n");
}
```

```
root@ubuntu: /home/linuxvirtualpc1/lab2
extern __pid_t wait (__WAIT_STATUS __stat_loc);

test1.c:18:10: warning: format '%d' expects argument of
2 has type 'double' [-Wformat=]
    printf("status=%d: Parent finished:\n",status);
    ^
root@ubuntu:/home/linuxvirtualpc1/lab2# gcc test1.c
root@ubuntu:/home/linuxvirtualpc1/lab2# ./a.out
Parent: will send signal.
pid=2675
pid=0
Child:signal is received.
status=3072: Parent finished:
root@ubuntu:/home/linuxvirtualpc1/lab2# gcc test1.c
root@ubuntu:/home/linuxvirtualpc1/lab2# ./a.out
Parent: will send signal.
pid=2682
pid=0
Child:signal is received.
status=256: Parent finished:
root@ubuntu:/home/linuxvirtualpc1/lab2# gcc test1.c
root@ubuntu:/home/linuxvirtualpc1/lab2# ./a.out
Parent: will send signal.
pid=2691
pid=0
Child:signal is received.
status=512: Parent finished:
root@ubuntu:/home/linuxvirtualpc1/lab2# gcc test1.c
root@ubuntu:/home/linuxvirtualpc1/lab2# ./a.out
Parent: will send signal.
pid=2712
pid=0
Child:signal is received.
status=2560: Parent finished:
root@ubuntu:/home/linuxvirtualpc1/lab2#
```

子进程调用 `exit` 返回状态参数的时候，父进程接收到的竟然是 256 倍以后的数值，在网上也没找到合适的解释，希望在下节课询问老师的时候可以得到解答。

2) 第二个问题是，在进行共享内存消息通信的时候，发现父进程往往会先去读入共享内存原有的数据块，因为程序运行远远快于用户的键盘输入，因此在用户输入新的内容覆盖之前，父进程读入共享内存原有的数据块似乎是不可避免的，不知道有没有好的方法可以在子进程用户输入之后，再触发父进程的 `while` 循环，而不是一上来子进程与父进程同时进入 `while` 循环。

2、体会

虽然父子进程交互与消息通信机制在上理论课的时候都讲过了，但是只有经过实验才能对于这些知识刻骨铭心，在做实验的时候经常会有这种时刻：哇！原来是上课讲的是这个意思！同时在做实验的过程中，我也对 C 语言的文件输入输

出部分进行了回顾，果然是熟能生巧，不用会遗忘。

其实这个实验一开始做还是很痛苦的，原本我是在 windows 上运行的，怎么跑也跑不出效果来，之后经过分析与询问同学才发现，这个代码只有在 Linux 中才能运行，因为头文件中存在 sys/等目录，这是 windows 中所没有的。

最后，感觉进行这类底层操作还是比较复杂的，需要掌握许多 Unix 的系统调用，明白参数类型，我们这个只是一个小小的实验，如果要深究的话，将会是一个浩大的工程。

七、源代码及其注释

1. 父子进程合作交互

```
#include<sys/types.h>
#include<signal.h>
#include<stdio.h>
#include<stdlib.h>
#include<sys/wait.h>
#include<unistd.h>
int main ( )
{
    int status; // status
    pid_t pid; // process pid
    void sigFunc(); // declaration of signal processing function
    signal(SIGUSR1,sigFunc); // SIGUSR1 trigger the func
    if (pid=fork()) { // the father process
        printf("The pid of the Father is %d.\n",getpid());
        printf("The pid of the Son is %d.\n",pid);
        printf("Father send a signal to the son.\n");
        kill(pid, SIGUSR1); // send signal SIGUSR1
        wait(& status); // wait for the return status from the son
        printf("Status=%d, Father process finished.\n",status);
    } else { // the son process
        sleep(3);
        printf("The Son receives the signal and triggers the signal function.\n");
        printf("After 3 second's sleep...\n");
        printf("The Son change the Image in the inner storage.\n");
        execl("/bin/ls","ls","-l",(char*)0);
        printf("excel error.\n"); // if some error happens.
        exit(2);
    }
}
```

```

void sigFunc() // signal processing function
{
    printf(":::It is signal processing function.\n");
}

```

2. 消息队列

头文件:

```

#include<errno.h>
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/msg.h>
#define MSGKEY 5678
struct msgtype{
    long mtype;        // message type definition
    char msg[1000];    // message block
};

```

Client 文件:

```

#include <stdio.h>
#include <stdlib.h>
#include "msgcom.h"
int main()
{
    struct msgtype buf; // create a buffer
    int qid;
    qid=msgget(MSGKEY,IPC_CREAT|0666);        // get the queue of message
    buf.mtype=5679;
    FILE *FileOpen=fopen("input.txt","r");    // open the message file
    if(FileOpen){
        while(fgets(buf.msg, 1000, FileOpen)){ // read message by line
            printf("Client has sent the following Message to the Server.\n");
            printf("#####\n");
            printf("%s",buf.msg);
            printf("#####\n\n");
            msgsnd(qid,&buf,sizeof(buf.msg),0); // send the message
        }
    }
    else{
        printf("Open file failed.\n");
        exit(1);
    }
    fclose(FileOpen);
}

```

```
}
```

Server 文件:

```
#include <stdio.h>
#include <stdlib.h>
#include "msgcom.h"
int main()
{
    struct msgtype buf;    // create a message buffer
    int qid;
    if((qid=msgget(MSGKEY,IPC_CREAT|0666))== -1)    // get the message ID
        return(-1);
    while(msgrcv(qid,&buf,1000,5679,MSG_NOERROR)){ // receive message
        FILE *FileOpen=fopen("output.txt","a");    // open the receive file
        if(FileOpen){
            printf("Server has received some data from the Client:\n");
            printf("#####\n");
            printf("%s",buf.msg);                    // print the message

printf("#####\n\n");

            fputs(buf.msg,FileOpen);                // put the message to the file
            fclose(FileOpen);
        }
        else{ // open file failure
            printf("Open file failed.\n");
            exit(1);
        }
    }
}
```

3. 共享内存

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

#define SHMKEY 10000 /* 共享内存关键字 */
#define SIZE 2048 /* 共享内存长度 */
#define SEMKEY1 15001 /* 信号灯组 1 关键字 */
#define SEMKEY2 15002 /* 信号灯组 2 关键字 */
```

```

static void semcall(int sid,int op) // 修改信号灯的值
{
    struct sembuf sb;
    sb.sem_num = 0;
    sb.sem_op = op;
    sb.sem_flg = 0;
    if(semop(sid,&sb,1) == -1) {
        printf("semop error.");
    }
}

int creatsem(key)
    key_t key;
{
    int sid;
    union semun{ /* 如 sem.h 中已定义，则省略 */
        int val;
        struct semid_ds *buf;
        ushort *array;
    } arg;
    if((sid=semget(key,1,0666|IPC_CREAT))==-1){ // 关键词为 1 的信号灯组，
权限 0666
        printf("semget error."); //出错处理
    }
    arg.val=1;
    if(semctl(sid,0,SETVAL,arg)==-1){ // 如果失败将会返回-1
        printf("semctl error."); //出错处理
    }
    return(sid);
}

void P(sid) // 信号灯的 op 值减 1
    int sid;
{
    semcall(sid,-1);
}

void V(sid) // 信号灯的 op 值增 1
    int sid;
{
    semcall(sid,1);
}

```

```

int main()
{
    char *segaddr; // 共享内存映射的地址指针
    int segid,sid1,sid2;
    if((segid=shmget(SHMKEY,SIZE, IPC_CREAT|0666))==-1) { // 共享内存申请失
败
        printf("shmget error.");
    }
    segaddr=shmat(segid,0,0); /* 将共享内存映射到进程数据空间 */
    sid1=creatsem(SEMKEY1); /* 创建两个信号灯，初值为 1 */
    sid2=creatsem(SEMKEY2);
    P(sid2); /* 置信号灯 2 值为 0，表示缓冲区空 */
    if(!fork()){
        while(1) { /* 子进程，接收和输出 */
            FILE *FileOpen1=fopen("output2.txt","a");
            if(FileOpen1){
                P(sid1); // 改变 sid1 的信号值
                fputs(segaddr,FileOpen1); // 共享内存中的值存入 output2.txt
                fputc('\n', FileOpen1); // 增加一个换行符
                printf("The Son has received Message from the Father: %s\n",
segaddr); //显示共享信息
                fclose(FileOpen1);
                V(sid2); // 改变 sid2 的信号值
            }
            else{
                exit(1);
            }
        }
    }
    else{
        while(1){ /* 父进程，输入和存储 */
            FILE *FileOpen2=fopen("input2.txt","a");
            if(FileOpen2){
                P(sid2); // 改变 sid2 的信号值
                printf("Input the Message that you want to send:");
                scanf("%s",segaddr); // 输入需要发送的信息
                fputs(segaddr,FileOpen2); // 将其存入存入 input2.txt
                fputc('\n', FileOpen2); // 增加换行
                V(sid1); // 改变 sid1 的信号值
                fclose(FileOpen2);
            }
            else{
                exit(1);
            }
        }
    }
}

```


}
}
}