上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

实验报告

LAB REPORT

操作系统

实验二 进程和进程通信

姓 名: 裘炜程

学 号: 516030910287

班 级: F1603602

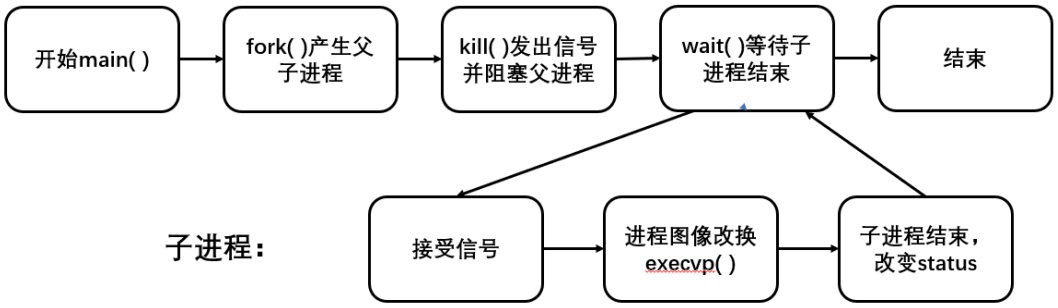
一、实验要求

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（对文件操不熟悉的同学可不必通过读写文件，只要键盘输入和输出至屏幕进行比较即可）。

二、实验原理及代码

1、父子进程之间的通信



其中的主要函数:

signal( )：第一个参数是处理的信号，可以用kill -l命令查看；第二个参数是处理的方式，可以自定义一个函数。

fork( )：若成功调用一次则返回两个值，子进程返回0，父进程返回子进程标记；否则，出错返回-1，fork将运行着的程序分成2个几乎完全一样的进程，每个进程都启动一个从代码的同一位置开始执行的线程。

kill( )：第一个参数pid大于零时，pid是信号欲送往的进程的标识；等于零时，信号将送往所有与调用kill()的那个进程属同一个使用组的进程；等于-1时，信号将送往所有调用进程有权给其发送信号的进程，除了进程1(init)；pid小于-1时，信号将送往以-pid为组标识的进程。

wait( )：用在父进程中等待回收子进程的资源，如果其所有子进程都还在运行，则阻塞；

如果一个子进程已经终止，正等待父进程获取其终止状态，则取得该子进程的终止状态立即返回。

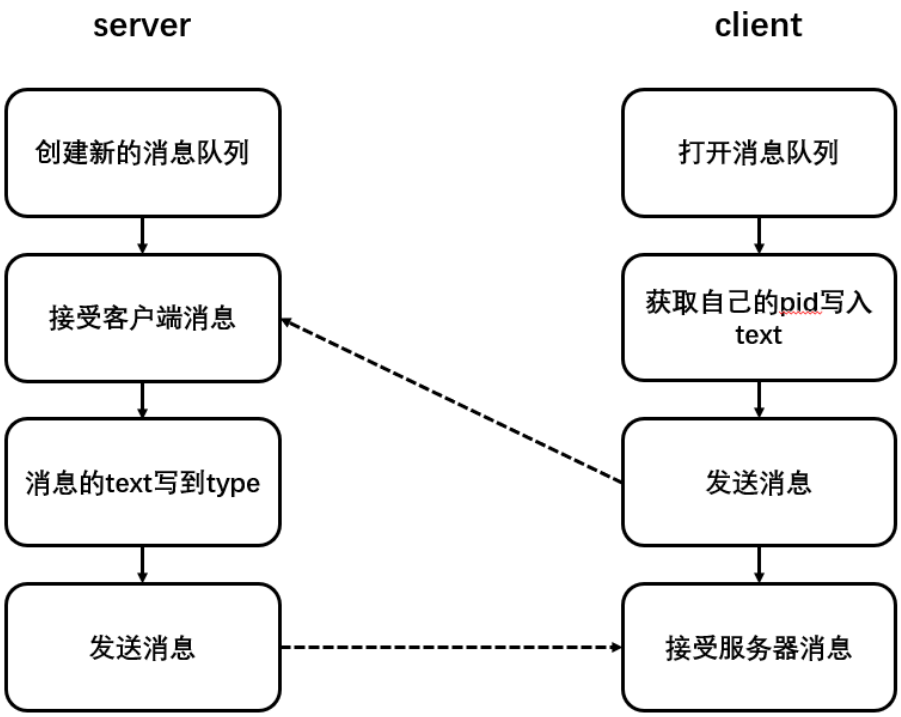
execvp( )：会从PATH 环境变量所指的目录中查找符合参数file 的文件名，找到后便执行该文件，然后将第二个参数argv传给该欲执行的文件。

代码如下：

```
#include <stdlib.h>
#include <sys/types.h>
#include <signal.h>
#include <stdio.h>

main()
{
    int pid,status=1;
    //为信号定义处理函数
    void func();
    signal(SIGUSR1,func);
    //fork()
    while ((pid=fork())==-1);
    printf("Pid:%d\n",pid);
    if (pid>=1)    //父进程
    {
        printf("Parent Process!Call Child!\n");
        kill(pid,SIGUSR1);//发送信号
        pid=wait(&status);//阻塞，等待子进程
        printf("Child process pid:%d,status:%d\n",pid,status);
    }
    else
    {
        sleep(2);
        printf("Child Process!Signal is received.\n");
        //进程图像改换
        char *argv[]={"ls","-l","test",0};
        printf("Pid:%d,Status:%d\n",pid,status);
        execvp("ls",argv);
        printf("Execvp goes wrong.\n");  //execvp 出错，实际没有走到
        exit(2);
    }
    printf("Parent Process Finish!\n");
}

void func()
{
```

```
    system("ls -l test2");
}
```

2、消息通信机制



其中的主要函数:

msgget( ): 用于创建一个新的或打开一个已经存在的消息队列, 此消息队列与 key 相对应。IPC_CREAT:创建新的消息队列。IPC_NOWAIT:读写消息队列要求无法满足时，不阻塞。返回值：调用成功返回队列标识符，否则返回-1。

msgsnd( )：将一个新的消息写入队列。第一个参数是消息队列的识别码；第二个是指向消息缓冲区的指针；第三个是消息大小；第四个是控制函数行为的标志，取值可以是：IPC_NOWAIT, 如果消息队列为空, 则返回一个 ENOMSG, 并将控制权交回调用函数的进程。

msgrcv( )：从消息队列中读取消息。前三个参数和 send 相同，第四个 msgtyp 等于 0 则返回队列的最早的一个消息，msgtyp 大于 0，则返回其类型为 msgtyp 的第一个消息，msgtyp 小于 0,则返回其类型小于或等于 mtype 参数的绝对值的最小的一个消息，第五个参数也是控制函数行为的标志, 取值可以是: MSG_NOERROR, 如果函数取得的消息长度大于 msgsz, 将只返回 msgsz 长度的信息，剩下的部分被丢弃了。

代码如下:

server.c:

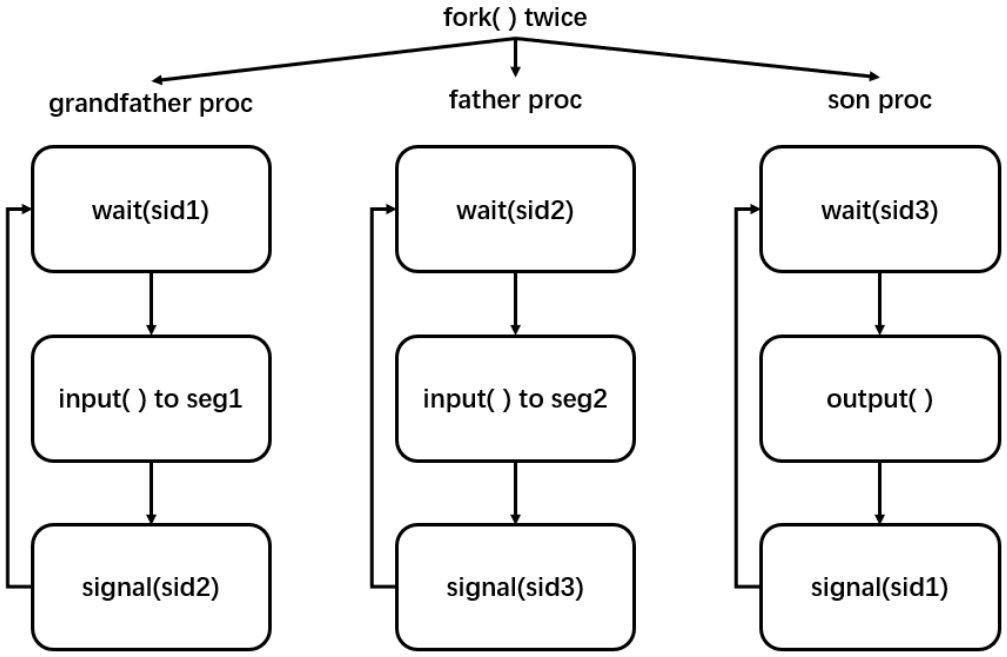
```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <stdlib.h>
#include <stdio.h>

#define MSGKEY 8888

//创建消息结构体
struct msgtype
{
    long type;
    int text;
}


main()
{
    struct msgtype message;
    int qid;
    if ((qid=msgget(MSGKEY,IPC_CREAT|0666))==-1)
    //创建消息队列，加上权限控制
    {
        printf("wrong");  //出错的情况
    }
    while (1)
    {
        msgrcv(qid,&message,512,1,MSG_NOERROR); //接受消息
       printf("Server received a message from process %d\n",message.text);
        message.type=message.text;   //将收到消息的内容作为 type 以响应
client
        msgsnd(qid,&message,sizeof(int),IPC_NOWAIT);   //发送消息
    }
}
```

client.c:

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <stdlib.h>
#include <stdio.h>

#define MSGKEY 8888

struct msgtype
```

```
{
    long type;
    int text;
}


main()
{
    struct msgtype message;
    int qid,pid;
    qid=msgget(MSGKEY,IPC_CREAT|0666);  //打开队列
    message.type=1;
    message.text=pid=getpid();   //获得 pid 作为消息
    msgsnd(qid,&message,sizeof(message.text),IPC_NOWAIT); //发送消息
    msgrcv(qid,&message,512,pid,MSG_NOERROR);  //接受消息
    printf("Client received a message from server, type
is: %ld\n",message.type);
}
```

3、内存共享机制



其中的主要函数:

semop( )：操作一个或一组信号，用来实现 P 和 V。第一个参数是信号集的识别码，第二个是指向存储信号操作结构的数组指针，第三个是信号操作结构的数量，恒大于或等于 1。

semget( )：创建一个新信号量或取得一个已有信号量。第一个参数 key 是整数值（唯一非零），不相关的进程可以通过它访问一个信号量；第二个参数指定需要的信号量数目，它的

值几乎总是 1；第三个参数是一组标志，当想要当信号量不存在时创建一个新的信号量，可以和值 IPC_CREAT 做按位或操作。设置了 IPC_CREAT 标志后，即使给出的键是一个已有信号量的键，也不会产生错误。而 IPC_CREAT | IPC_EXCL 则可以创建一个新的，唯一的信号量，如果信号量已存在，返回一个错误。semget()函数成功返回一个相应信号标识符（非零），失败返回-1。

semctl( )：用来直接控制信号量信息。第一、二个参数跟 semget 一样，第三个是一个命令，如果是 SETVAL 就是用来把信号量初始化为一个已知的值，这个值通过 union semun 中的 val 成员设置，其作用是在信号量第一次使用前对它进行设置，第四个参数是一个 union semum 结构。

shmget( )：用于 Linux 进程通信共享内存，它得到一个共享内存标识符或创建一个共享内存对象并返回共享内存标识符。第一个参数是共享内存的键值，第二个数是新建的内存大小，第三个是 IPC_CREAT|IPC_EXCL 的话，如果内核中不存在键值与 key 相等的共享内存，则新建一个共享内存；如果存在这样的共享内存则报错。

shmat( )：用于 Linux 进程通信共享内存，连接共享内存标识符为 shmid 的共享内存，连接成功后把共享内存区对象映射到调用进程的地址空间，随后可像本地空间一样访问。第一个参数 shmid，第二个是指定共享内存出现在进程内存地址的什么位置，直接指定为 NULL 让内核自己决定一个合适的地址位置；第三个 0 表示读写模式。

程序中 sid1 表示父父进程与子进程的信号量，sid2 表示父进程与子进程的信号量，sid3 表示输入与打印之间的信号量。

代码如下：

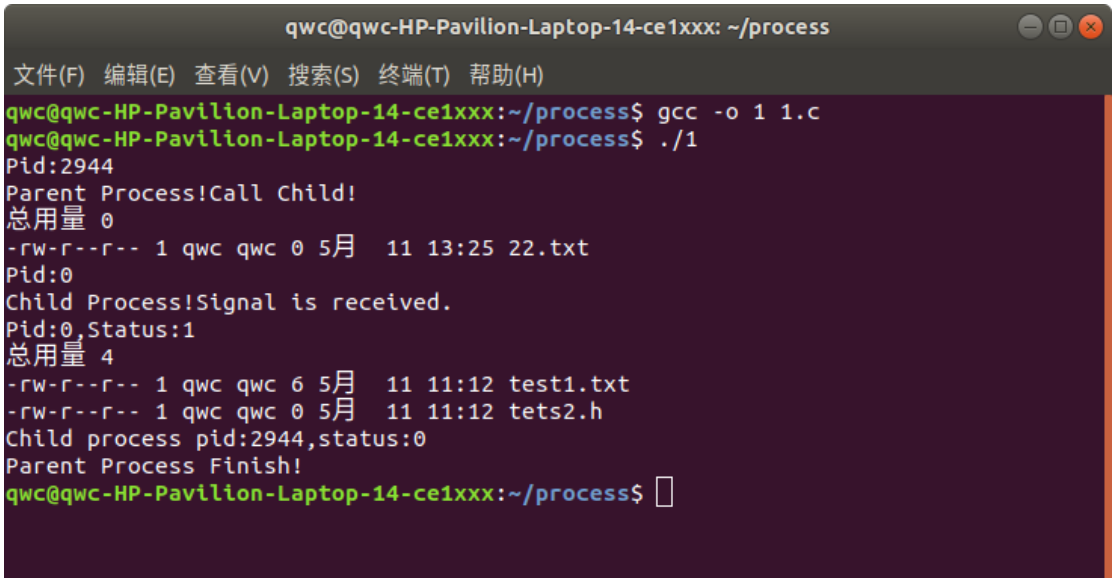
```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/shm.h>
#include <stdlib.h>
#include <stdio.h>

#define SHMKEY1 5000  //共享内存键值 1
#define SHMKEY2 5001  //共享内存键值 2
#define SEMKEY1 10001 //信号量键值 1
#define SEMKEY2 10002 //信号量键值 2
#define SEMKEY3 10003 //信号量键值 3

static void sempv(int sid,int op)    //处理 pv
```

```
{
    struct sembuf buf;
    buf.sem_num=0;
    buf.sem_op=op;
    buf.sem_flg=0;

    if (semop(sid,&buf,1)==-1)
        {perror("semop");}
};

int createsem(int key)    //信号量创建和初始化
{
    int sid;
    union semun
    {
        int val;
        struct semid_ds *buf;
        ushort *array;
    }arg;

    if ((sid=semget(key,1,0666|IPC_CREAT))==-1)
        {perror("semget");}
    arg.val=1; //定义初始值
    if (semctl(sid,0,SETVAL,arg)==-1)
        {perror("semctl");}

    return (sid);
}

void P(int sid)   //P,-1
{
    sempv(sid,-1);
}

void V(int sid)   //V,+1
{
    sempv(sid,1);
}

main()
{
    char *segaddr1,*segaddr2;
    int segid1,segid2,sid1,sid2,sid3;
    if ((segid1=shmget(SHMKEY1,512,IPC_CREAT|0666))==-1)   //创建共享内存
```

```
段 1
        {perror("semget");}
    if ((segid2=shmget(SHMKEY2,512,IPC_CREAT|0666))==-1)   //创建共享内存
段 2
        {perror("semget");}

    segaddr1=shmat(segid1,0,0);  //将共享内存映射到进程数据空间
    segaddr2=shmat(segid2,0,0);
    sid1=createsem(SEMKEY1); //创建信号量
    sid2=createsem(SEMKEY2);
    sid3=createsem(SEMKEY3);

    P(sid3);   //缓冲区为空
    P(sid2);

    if (fork())
    {   //父父进程
        while(1)
        {
            P(sid1);
            scanf("%s",segaddr1);
            V(sid2);
        }
    }
    else if (fork())
    {   //父进程
        while(1)
        {
            P(sid2);
            scanf("%s",segaddr2);
            V(sid3);
        }
    }
    else
    {   //子进程
        while(1)
        {
            P(sid3);
            printf("Parent Process Data:%s\n",segaddr2);
            printf("Grandparent's Process Data:%s\n",segaddr1);
            V(sid1);
        }
    }
}
```

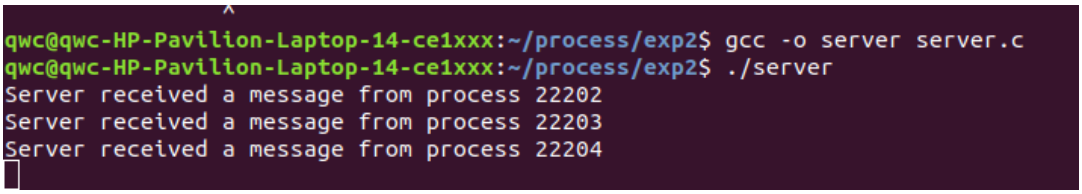
三、实验结果

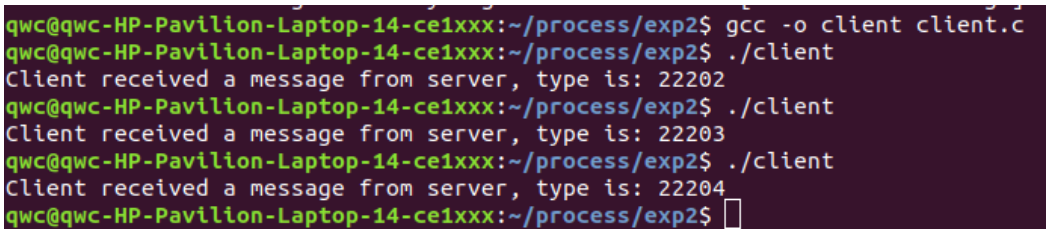
1、父子进程通信



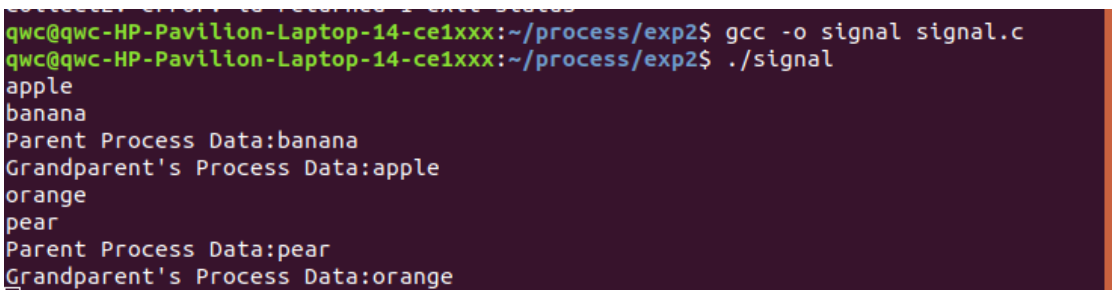
可以看到父进程向子进程发送信息，为函数 system（“ls -l test2”）即显示 test2 文件夹中的文件信息；子进程的影像被改成 system（“ls -l test”）即显示 test 文件夹中的文件信息。从输出中，可以观察到 fork 两个返回值的情况，并且可以看到 status 和 pid 的变化，符合之前的理解。

2、消息通信机制





3、共享内存机制



四、实验总结

这次实验运用了大部分进程通信的机制，让我对通信的不同方法、用到的不同函数有了了解。在课堂中大多是理论的学习，但是运用到实践中，结合具体操作系统的时候还是很难下手，因此这样的实践很有价值。做完实验后，我还是对同步和互斥之间的关系不是很理解，希望老师能再讲一下。