

# 操作系统 实验二 进程与进程通信

---

## 操作系统 实验二 进程与进程通信

### 摘要

#### 算法思想和概要设计

##### 算法思想

##### PART1

##### PART2

##### 消息通信

##### 共享内存

#### 概要设计

##### PART1

##### PART2

##### 消息通信

##### 共享内存

#### 数据结构与变量说明

##### 消息缓冲区

##### 信号灯组

### 源程序

#### PART1

##### 主程序

#### PART2

##### 消息通信

##### 消息缓冲区与预处理宏定义

##### Server 进程

##### Client 进程

##### 共享内存

##### 信号灯组与预处理宏定义

##### 信号灯组初始化函数

##### 信号灯semWait与semSignal函数

##### 共享内存重要函数

##### 主程序

### 测试

#### PART1

#### PART2

##### 消息通信

##### 共享内存

### 改进与体会

#### 改进

#### 体会

# 摘要

---

在本试验 **PART1** 中，我自己设计了一个程序，该程序创建一个子进程，使父子进程合作，协调地完成收发信号、打印目录功能。在该程序中使用了进程的睡眠、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。在本试验 **PART2** 中，我分别利用 *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与简单的信息处理。

## 算法思想和概要设计

---

### 算法思想

#### PART1

父进程设置信号 **SIGUSR1** 的信号处理函数，子进程继承父进程的信号处理方式。子进程创建后进入 10s 的低优睡眠；父进程先打印相关信息，在发送软中断信号前先进入 1s 的低优睡眠，防止在子进程进入睡眠前就发送信号。父进程完成睡眠后将软中断信号传送给子进程，唤醒低优睡眠中的子进程，触发自定义的信号处理程序。之后子进程正常运行，执行图像改换，列出当前路径下的目录。父进程等待子进程终止。

#### PART2

##### 消息通信

本部分分为 **Server** 进程和 **Client** 进程两个进程。**Server** 和 **Client** 通过同一个关键字获得相同消息队列的标识符。之后 **Client** 通过消息通信向 **Server** 发送信息，内容为自己的进程号，并接收 **Server** 通过消息通信返回的信息。**Server** 保持监听消息队列，收到 **Client** 发送的消息后就打印出来，并将消息经过一定处理后返回 **Client**。

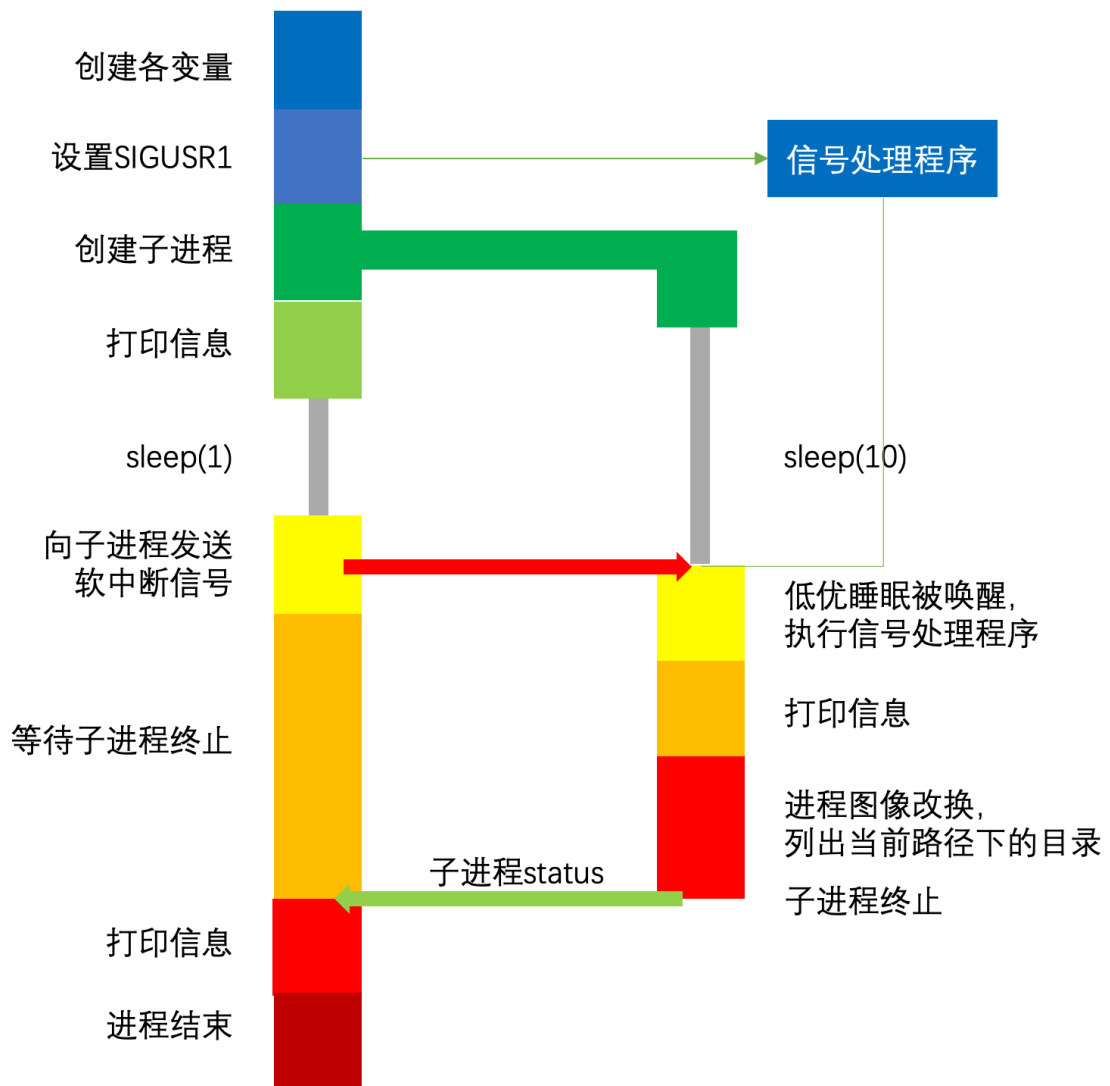
##### 共享内存

父进程创建 1kb 大小的共享内存与同步信号灯组，得到共享内存标识符与信号灯组标识符。父进程把共享内存连接到当前进程的地址空间。由于本部分只有两个进程，所以任一信号灯的信号量不可能超过 1，因此信号灯组只需要两个信号灯，不需要互斥信号灯。信号灯组含读与写两个信号灯，写信号灯初始化为 1，读信号灯初始化为 0。然后父进程创建子进程，子进程继承之前所述参数。之后父进程向共享内存中输入信息，子进程从共享内存中读出并处理信息，完成父子进程间的共享内存通信。

### 概要设计

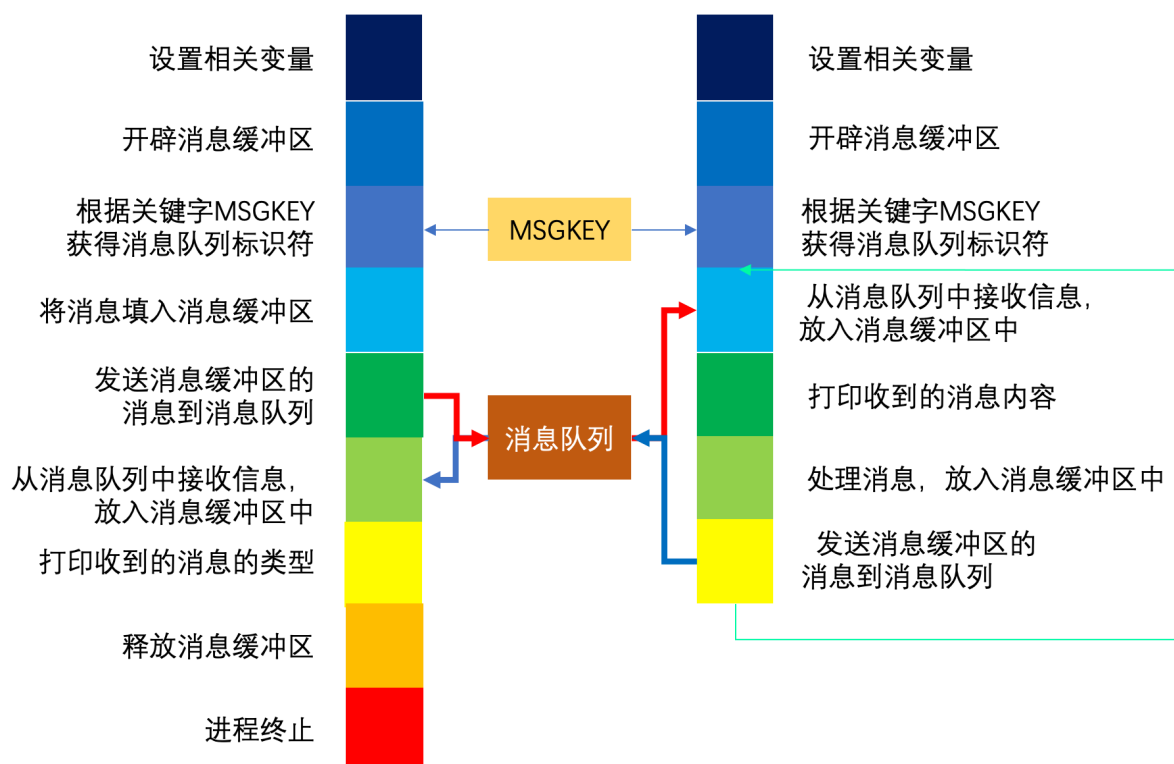
以下将以流程图来展示对该程序分配与释放内存的过程概要设计。

#### PART1

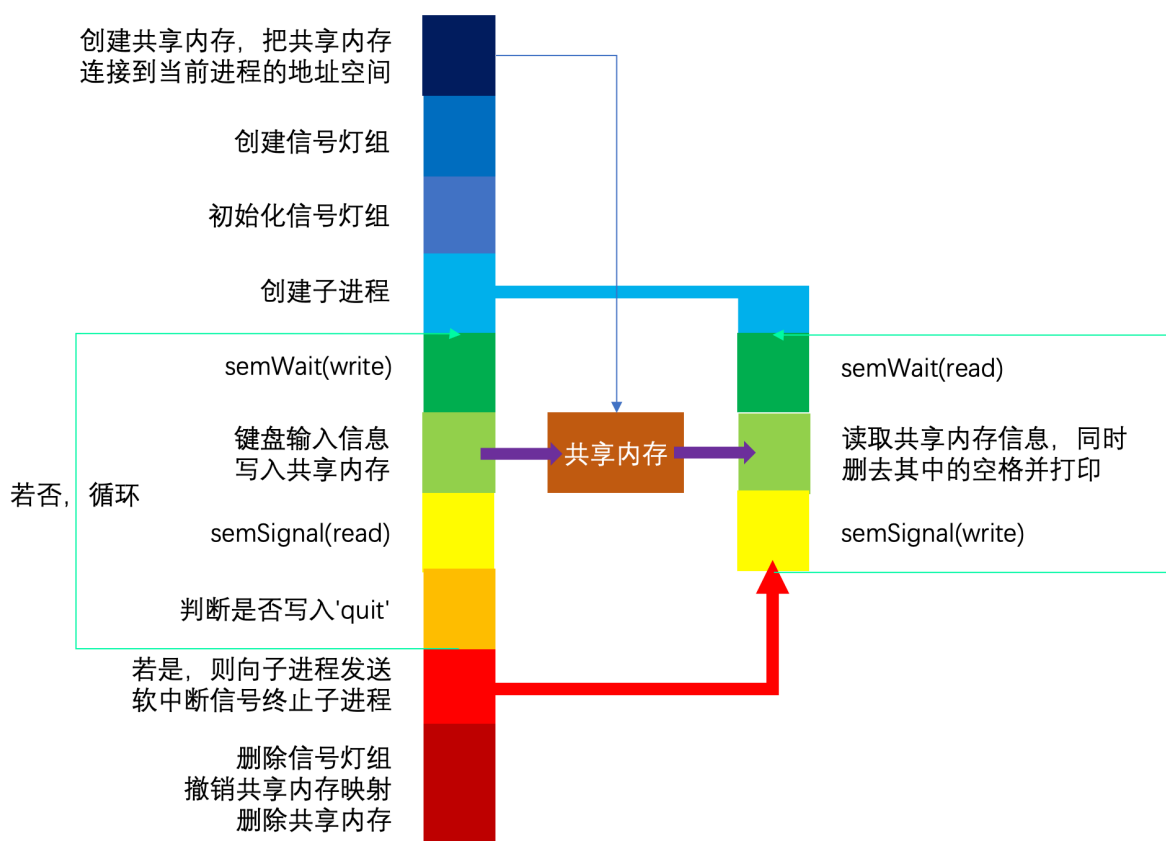


## PART2

消息通信



## 共享内存



# 数据结构与变量说明

## 消息缓冲区

```
1 struct msgtype{
2     long mtype;    // 消息类型
3     int text;      // 消息正文
4 };
```

## 信号灯组

```
1 typedef union semunion{
2     int          val;    /* Value for SETVAL */
3     struct semid_ds *buf; /* Buffer for IPC_STAT, IPC_SET */
4     unsigned short *array; /* Array for GETALL, SETALL */
5     struct seminfo *__buf; /* Buffer for IPC_INFO(Linux specific) */
6 }semunion;
```

## 源程序

### PART1

#### 主程序

```
1  #include <sys/types.h>
2  #include <signal.h>
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <stdarg.h>
6  #include <unistd.h>
7
8  int main(){
9      int status;
10     pid_t pid;
11     void func();
12     signal(SIGUSR1, func);    // 设置信号SIGUSR1的中断处理函数为func
13     while((pid = fork()) == -1); // 产生子进程
14     if(pid){                  // 父进程
15         printf("It is the parent process.\n");
16         printf("Parent: will send signal.\n");
17         sleep(1);             // 向子进程传送信号前先睡眠一秒，保证信号到达时子
                                // 进程正在低优睡眠中
18         kill(pid, SIGUSR1);    // 父进程向子进程传送SIGUSR1软中断信号
19         pid = wait(&status);    // 父进程等待子进程终止
20         printf("Child process %d, status = %d.\n", pid, status);
21     }
```

```

22     else{                                // 子进程
23         sleep(10);
24         printf("It is the child process.\n");
25         printf("Child: signal is received.\n");
26         execl("/bin/ls", "ls", "-l", (char*)0); // 进程图像改换, 将子进程图像改
换为ls命令
27         printf("execl error\n");           // 图像改换失败
28         exit(2);
29     }
30     printf("Parent process finish.\n");
31 }
32
33 void func(){ // 自己设置的信号SIGUSR1的中断处理函数
34     printf("It is a signal processing function.\n");
35     system("date");
36 }

```

## PART2

### 消息通信

#### 消息缓冲区与预处理宏定义

```

1  /* msgcom.h */
2  #include <errno.h>
3  #include <sys/types.h>
4  #include <sys/ipc.h>
5  #include <sys/msg.h>
6  #define MSGKEY 5678
7
8  struct msgtype{
9      long mtype; // 消息类型
10     int text;    // 消息正文
11 };

```

#### Server 进程

```

1  /* Server 进程 */
2  #include "msgcom.h"
3  void main()
4  {
5      struct msgtype buf; // 创建消息缓冲区
6      int qid;
7      if((qid=msgget(MSGKEY, IPC_CREAT|0666)) == -1) // 获得消息队列标识符
8          return(-1);
9      while(1){
10         msgrcv(qid, &buf, 512, 1, MSG_NOERROR); // 接收消息队列消息存入消息缓冲区
11         printf("Server receive a request from process %d\n", buf.text); // 打印
消息内容(对方进程的进程号)

```

```

12     buf.mtype = buf.text;    // 处理消息，将消息类型赋值为消息内容
13     msgsnd(qid, &buf, sizeof(int), 0);    // 将消息缓冲区的消息发送给消息队列
14 }
15 }

```

## Client 进程

```

1  /* Client 进程 */
2  #include "msgcom.h"
3  void main()
4  {
5      struct msgtype buf;    // 创建消息缓冲区
6      int qid, pid;
7      qid = msgget(MSGKEY, IPC_CREAT|0666);    // 获得消息队列标识符
8      buf.mtype = 1;    // 赋消息类型为1
9      buf.text = pid = getpid();    // 赋消息内容为本进程进程号
10     msgsnd(qid, &buf, sizeof(buf.text), 0); // 将消息缓冲区的消息发送给消息队列
11     msgrcv(qid, &buf, 512, pid, MSG_NOERROR); // 接收消息队列消息存入消息缓冲区
12     printf("Request received a massags from server, type is:%d\n ",
13           buf.mtype); // 打印消息内容(本进程进程号)
13 }

```

## 共享内存

### 信号灯组与预处理宏定义

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4  #include <signal.h>
5  #include <sys/types.h>
6  #include <sys/ipc.h>
7  #include <sys/shm.h>
8  #include <sys/sem.h>
9  #include <wait.h>
10 #include <string.h>
11 #define SEM_WRITE 0    // 写信号灯编号
12 #define SEM_READ 1    // 读信号灯编号
13 #define SIZE 1024    // 共享内存容量
14 #define P_OPERATION -1    // P操作
15 #define V_OPERATION 1    // V操作
16
17 typedef union semunion{
18     int val;    /* Value for SETVAL */
19     struct semid_ds *buf;    /* Buffer for IPC_STAT, IPC_SET */
20     unsigned short *array;    /* Array for GETALL, SETALL */
21     struct seminfo *__buf;    /* Buffer for IPC_INFO(Linux specific) */
22 }semunion;

```

## 信号灯组初始化函数

```
1  /*
2  func:对num个信号灯赋值
3  para:
4      semid 信号灯组id
5      sema_value[] 信号灯值缓冲区
6      num 信号灯个数
7
8  */
9  void init_sema_value(int semid,int *sema_value,int num){
10
11      semunion semu;
12      for(int i = 0; i < num; i++){
13          semu.val = sema_value[i];
14          semctl(semid, i, SETVAL, semu);
15      }
16  }
```

## 信号灯semWait与semSignal函数

```
1  /*
2  func: semWait与semSignal操作封装
3  para:
4      semid 信号灯组id
5      whichnum 信号灯编号
6      op 对信号灯的操作
7
8  */
9  void operate(int semid, int whichnum, int op){
10      struct sembuf opbuf;
11      opbuf.sem_num = whichnum;
12      opbuf.sem_op = op;
13      opbuf.sem_flg = 0;
14      semop(semid, &opbuf, 1);
15  }
```

## 共享内存重要函数

```
1  int shmget(key_t key, size_t size, int shmflg);
```

第一个参数，程序需要提供一个参数 *key*（非 0 整数），它有效地为共享内存段命名，**shmget** 函数成功时返回一个与 *key* 相关的共享内存标识符（非负整数），用于后续的共享内存函数。调用失败返回 -1。

不相关的进程可以通过该函数的返回值访问同一共享内存，它代表程序可能要使用的某个资源，程序对所有共享内存的访问都是间接的，程序先通过调用 **shmget** 函数并提供一个键，再由系统生成一个相应的共享内存标识符（**shmget** 函数的返回值），只有 **shmget** 函数才直接使用信号量键，所有其他的信号量函数使用由 **semget** 函数返回的信号量标识符。



第二个参数, *size* 以字节为单位指定需要共享的内存容量

第三个参数, *shmflg* 是权限标志, 它的作用与 **open** 函数的 *mode* 参数一样, 如果要想在 *key* 标识的共享内存不存在时, 创建它的话, 可以与 *IPC\_CREAT* 做或操作。共享内存的权限标志与文件的读写权限一样, 举例来说, 0644, 它表示允许一个进程创建的共享内存被内存创建者所拥有的进程向共享内存读取和写入数据, 同时其他用户创建的进程只能读取共享内存。

```
1 void *shmat(int shm_id, const void *shm_addr, int shmflg);
```

第一次创建完共享内存时, 它还不能被任何进程访问, **shmat** 函数的作用就是用来启动对该共享内存的访问, 并把共享内存连接到当前进程的地址空间。

第一个参数, *shm\_id* 是由 *shmget* 函数返回的共享内存标识。

第二个参数, *shm\_addr* 指定共享内存连接到当前进程中的地址位置, 通常为 *NULL*, 表示让系统来选择共享内存的地址。

第三个参数, *shm\_flg* 是一组标志位, 通常为 0。

调用成功时返回一个指向共享内存第一个字节的指针, 如果调用失败返回 *-1*。

```
1 int shmdt(const void *shmaddr);
```

该函数用于将共享内存从当前进程中分离。注意, 将共享内存分离并不是删除它, 只是使该共享内存对当前进程不再可用。参数 *shmaddr* 是 **shmat** 函数返回的地址指针, 调用成功时返回 0, 失败时返回 *-1*。

```
1 int shmctl(int shm_id, int command, struct shmid_ds *buf);
```

第一个参数, *shm\_id* 是 **shmget** 函数返回的共享内存标识符。

第二个参数, *command* 是要采取的操作, 它可以取下面的三个值:

*IPC\_STAT*: 把 **shmid\_ds** 结构中的数据设置为共享内存的当前关联值, 即用共享内存的当前关联值覆盖 **shmid\_ds** 的值。

*IPC\_SET*: 如果进程有足够的权限, 就把共享内存的当前关联值设置为 **shmid\_ds** 结构中给出的值

*IPC\_RMID*: 删除共享内存段

第三个参数, *buf* 是一个结构指针, 它指向共享内存模式和访问权限的结构<sup>1</sup>。

主程序

```
1 int main(void)
2 {
3     int shm_id, sem_id, sema_value[2];
4     pid_t pid;
5     char *addr = NULL;
6     key_t Key;
7     Key = ftok(".", 'm'); // 获得关键字, 之后用于创建共享内存与信号灯组
8     // 创建共享内存
9     if((shm_id=shmget(Key, SIZE, IPC_CREAT|0666)) < 0){
```

```

10     perror("shmget");
11     exit(-1);
12 }
13 // 创建信号灯组, 含两个信号灯
14 if((semid=semget(Key, 2, IPC_CREAT|0666|IPC_EXCL)) < 0){
15     perror("semget");
16     goto error1;
17 }
18 // 信号灯初始化
19 sema_value[SEM_WRITE] = 1;
20 sema_value[SEM_READ] = 0;
21 init_sema_value(semid, sema_value, 2);
22 // 映射共享内存
23 if((addr=(char *)shmat(shmid, NULL, 0)) < 0){
24     perror("shmat");
25     goto error2;
26 }
27 if((pid=fork()) == 0){
28     char *new, *old;
29     while(1)    // 子进程循环
30     {
31         operate(semid, SEM_READ, P_OPERATION); // 对读信号灯的semWait操作
32         new = old = addr;
33         while(*old){                                // 删除信息中的所有空格
34             if((*old) != ' '){
35                 (*new++) = (*old);
36             }
37             old++;
38         }
39         (*new)='\0';
40         printf("Kid process %d is receiving : %s\n", getpid(), addr);
41         operate(semid, SEM_WRITE, V_OPERATION); // 对写信号灯的semSignal操作
42     }
43 }
44 else
45 {
46     while(1)    // 父进程循环
47     {
48         operate(semid, SEM_WRITE, P_OPERATION); // 对读信号灯的semWait操作
49         printf("Parent process %d is sending :\n", getpid());
50         fgets(addr, SIZE, stdin);
51         operate(semid, SEM_READ, V_OPERATION); // 对读信号灯的semWait操作
52         if(strcmp(addr, "quit\n")==0)          // 发现输入“quit”后发送软中断信
号给子进程使子进程终止, 父进程也推出循环
53         {
54             kill(pid, SIGUSR1);
55             break;
56         }
57     }

```

```

58     }
59
60 error2:
61     semctl(semid, 0, IPC_RMID);    //删除信号灯集
62     shmdt(addr);                  //撤销共享内存映射
63 error1:
64     shmctl(shmid, IPC_RMID, NULL); //删除共享内存
65
66     return 0;
67 }

```

## 测试

### PART1

```

hongxing@ubuntu:~$ cd Desktop/
hongxing@ubuntu:~/Desktop$ cd experiment2nd_part2/
hongxing@ubuntu:~/Desktop/experiment2nd_part2$ gcc -o part1 main.c
main.c: In function 'main':
main.c:20:15: warning: implicit declaration of function 'wait'; did you mean 'main'? [-Wimplicit-function-declaration]
    pid = wait(&status);
              ^~~~~~
hongxing@ubuntu:~/Desktop/experiment2nd_part2$ ./part1
It is the parent process.
Parent: will send signal.
It is a signal processing function.
Wed Apr 29 01:54:26 PDT 2020
It is the child process.
Child: signal is received.
total 116
-rwxr-xr-x 1 hongxing hongxing 8536 Apr 26 08:47 Client
-rw-r--r-- 1 hongxing hongxing 663 Apr 26 08:42 ClientProcess.c
-rw-r--r-- 1 hongxing hongxing 846 Apr 26 08:23 main.c
-rw-r--r-- 1 hongxing hongxing 1910 Apr 26 09:27 MEMsh.c
-rw-r--r-- 1 hongxing hongxing 513 Apr 26 08:32 msgcom.h
-rwxr-xr-x 1 hongxing hongxing 8768 Apr 29 01:54 part1
-rwxr-xr-x 1 hongxing hongxing 8760 Apr 28 04:39 read
-rwxr-xr-x 1 hongxing hongxing 8496 Apr 26 08:48 Server
-rw-r--r-- 1 hongxing hongxing 668 Apr 26 08:41 ServerProcess.c
-rw-r--r-- 1 hongxing hongxing 1524 Apr 28 04:35 shmdata.h
-rwxr-xr-x 1 hongxing hongxing 13224 Apr 28 20:29 shmExp
-rw-r--r-- 1 hongxing hongxing 3623 Apr 28 20:28 shmExperiment.c
-rw-r--r-- 1 hongxing hongxing 1108 Apr 28 04:35 shmread.c
-rw-r--r-- 1 hongxing hongxing 1912 Apr 28 04:35 shmwrite.c
-rwxr-xr-x 1 hongxing hongxing 12952 Apr 28 05:23 write
Child process 39891, status = 0.
Parent process finish.
hongxing@ubuntu:~/Desktop/experiment2nd_part2$

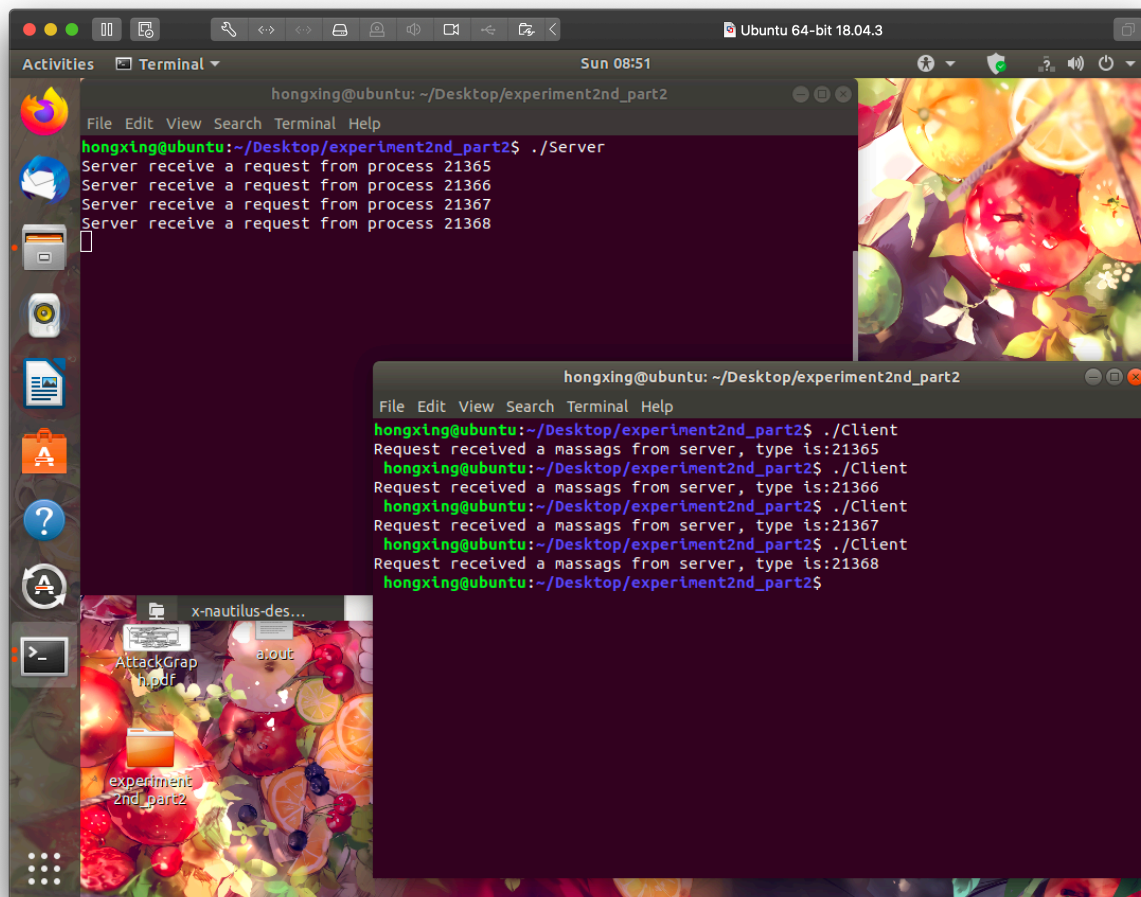
```

过程描述与分析：在以下两句打印出来之后，打印停止了 1s，之后立刻将剩余的所有输出全部打印完毕，子进程并没有完成等待 10s，且打印出了自定义中断处理程序中设置的内容，说明子进程成功被信号唤醒。

- 1 It is the parent process.
- 2 Parent: will send signal.

## PART2

### 消息通信

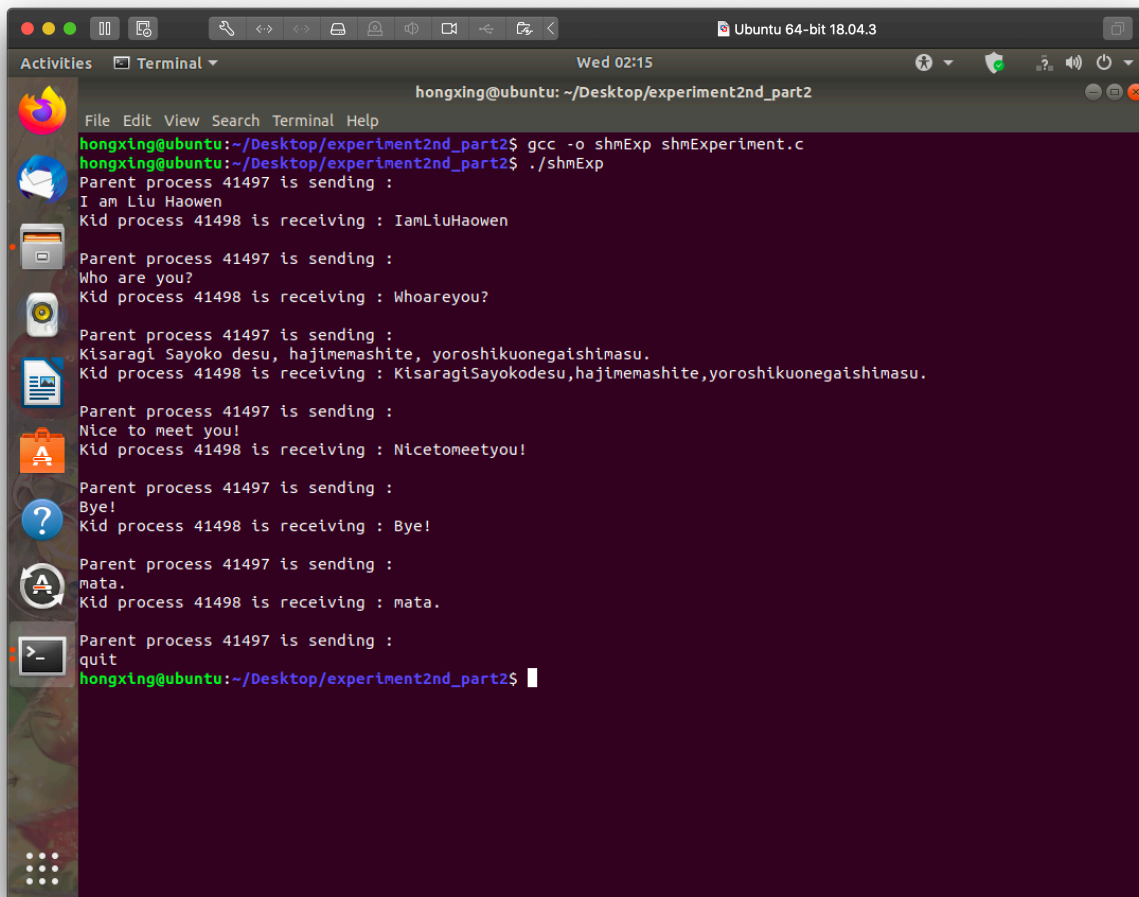


```
hongxing@ubuntu: ~/Desktop/experiment2nd_part2
File Edit View Search Terminal Help
hongxing@ubuntu:~/Desktop/experiment2nd_part2$ ./Server
Server receive a request from process 21365
Server receive a request from process 21366
Server receive a request from process 21367
Server receive a request from process 21368
hongxing@ubuntu:~/Desktop/experiment2nd_part2$

hongxing@ubuntu: ~/Desktop/experiment2nd_part2
File Edit View Search Terminal Help
hongxing@ubuntu:~/Desktop/experiment2nd_part2$ ./Client
Request received a massags from server, type is:21365
hongxing@ubuntu:~/Desktop/experiment2nd_part2$ ./Client
Request received a massags from server, type is:21366
hongxing@ubuntu:~/Desktop/experiment2nd_part2$ ./Client
Request received a massags from server, type is:21367
hongxing@ubuntu:~/Desktop/experiment2nd_part2$ ./Client
Request received a massags from server, type is:21368
hongxing@ubuntu:~/Desktop/experiment2nd_part2$
```

过程描述与分析：**Server** 常开，**Client** 发送并接收一次消息后即关闭。进行多次实验，发现两边输出均为 **Client** 进程号，说明实验成功。

### 共享内存



```
hongxing@ubuntu: ~/Desktop/experiment2nd_part2
File Edit View Search Terminal Help
hongxing@ubuntu:~/Desktop/experiment2nd_part2$ gcc -o shmExp shmExperiment.c
hongxing@ubuntu:~/Desktop/experiment2nd_part2$ ./shmExp
Parent process 41497 is sending :
I am Liu Haowen
Kid process 41498 is receiving : IamLiuHaowen

Parent process 41497 is sending :
Who are you?
Kid process 41498 is receiving : Whoareyou?

Parent process 41497 is sending :
Kisaragi Sayoko desu, hajimemashite, yoroshikuonegaishimasu.
Kid process 41498 is receiving : KisaragiSayokodesu,hajimemashite,yoroshikuonegaishimasu.

Parent process 41497 is sending :
Nice to meet you!
Kid process 41498 is receiving : Nicetomeetyou!

Parent process 41497 is sending :
Bye!
Kid process 41498 is receiving : Bye!

Parent process 41497 is sending :
mata.
Kid process 41498 is receiving : mata.

Parent process 41497 is sending :
quit
hongxing@ubuntu:~/Desktop/experiment2nd_part2$
```

可见父进程号为 41497，子进程号为 41978。父进程发送消息，子进程将接受到的消息去掉空格后打印出来。测试结果完全正确，程序能正确运行。

## 改进与体会

### 改进

一开始（父进程发送信号前没有睡眠一秒）**PART1** 中遇到了一个问题：父进程发送的信号虽然能出发信号处理程序，但无法唤醒子进程的 10s 低优睡眠；但若让父进程先睡眠 1s 后就能正确唤醒子进程。与同学交流后发现大家都有这个现象。之后同学们在课程微信群中的讨论让问题逐渐明朗：原因是由于进程运行的不确定性，父进程发送软中断信号给子进程时，子进程可能还没有进入睡眠，也就不存在唤醒子进程的问题了，子进程之后还会照样睡眠十秒。所以父进程在发送信号给子进程前先睡眠一秒，使操作系统进行进程调度能保证会运行到一部分子进程代码，即让子进程进入睡眠。

### 体会

这次实验让我非常有效地复习了进程的关键知识，加深了对进程调度、进程控制、进程的创建和映像转换、进程的同步与互斥、进程间的数据通信等知识点的理解。同时也入门了 **Linux** 上的 **C** 语言编程与系统调用。对理解操作系统的原理与规则非常有帮助。

by 刘浩文 517021911065