进程间通信实验报告

518021910109　　王梓帆

1. 实验要求

1. 设计一个程序，在其中创建一个子进程，使父子进程合作，协调完成某一功能。要求在该程序中还要使用进程的睡眠、进程图像改换、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，以便判断数据传送的正确性。

---

2. 父子进程实验

2.1 实验设计

父子进程协同完成计算、保存结果和打印输出的任务。

- 父进程：对1-100求和，将结果保存到文本文件中，并向子进程发送信号。然后父进程等待3秒，不论子进程是否完成任务，都杀死子进程。
- 子进程：处于睡眠状态，接收到父进程信号后被唤醒，读取结果文件，将结果打印到屏幕。

2.2 函数实现

调用信号处理函数和创建子进程

```
signal(SIGUSR1, sig_p);

while((pid = fork()) == -1);
```

在信号处理函数中打印信号ID

```
void sig_p(int dunno)
{
  printf("signal id: %d.\n", dunno);
}
```

父进程在计算和保存结束后向子进程发送信号，并等待3s

```
// send signal to child
printf("signal sent\n");
kill(pid, SIGUSR1);

printf("sleep 3.\n");
sleep(3);
```

杀死子进程

```
if( (waitpid(pid, NULL, WNOHANG)) == 0)
{
  if( (ret = kill(pid, SIGKILL)) == 0)
    printf("killed the child process: %d.\n", pid);
}
```

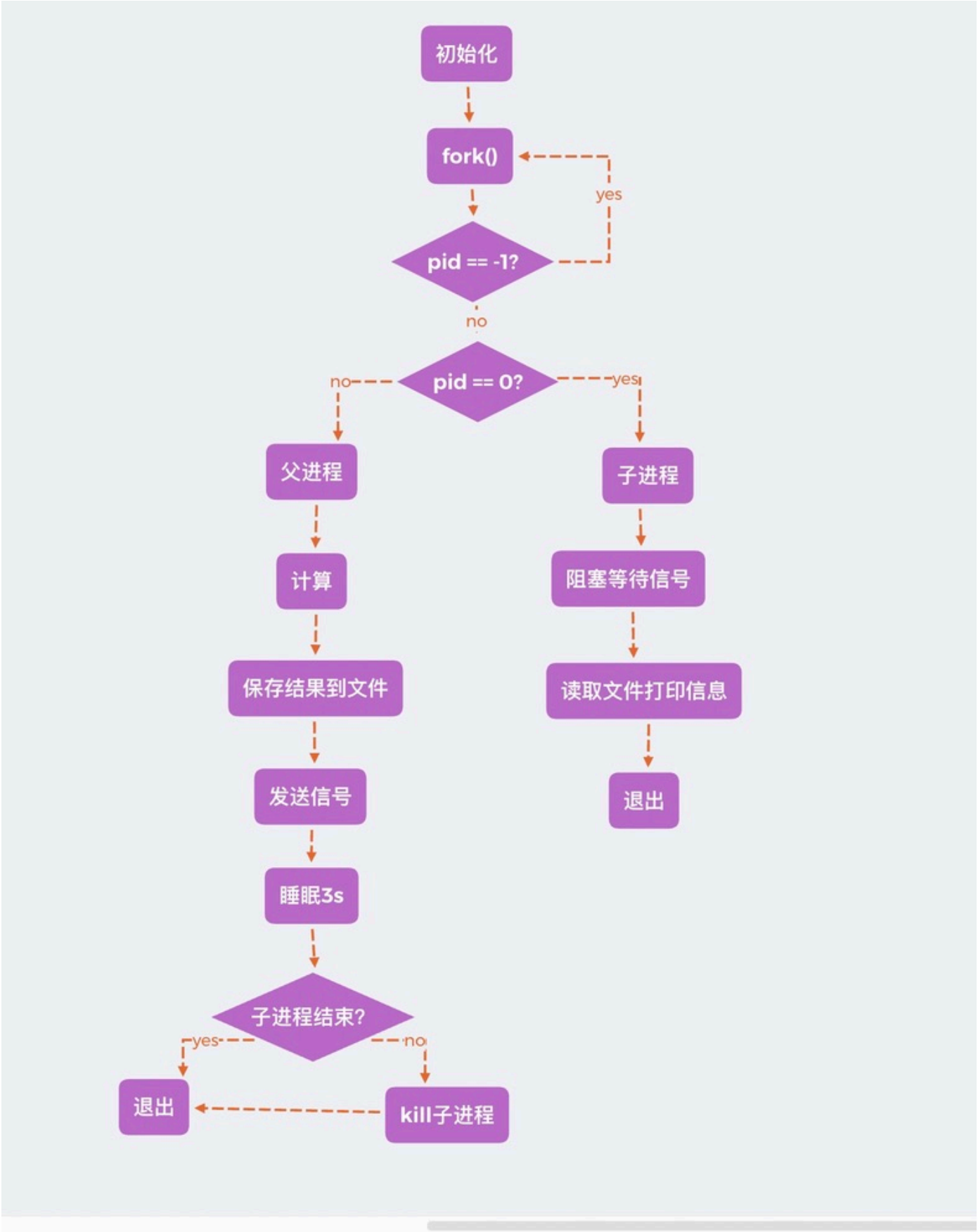
子进程首先挂起等待信号

```
pause();
```

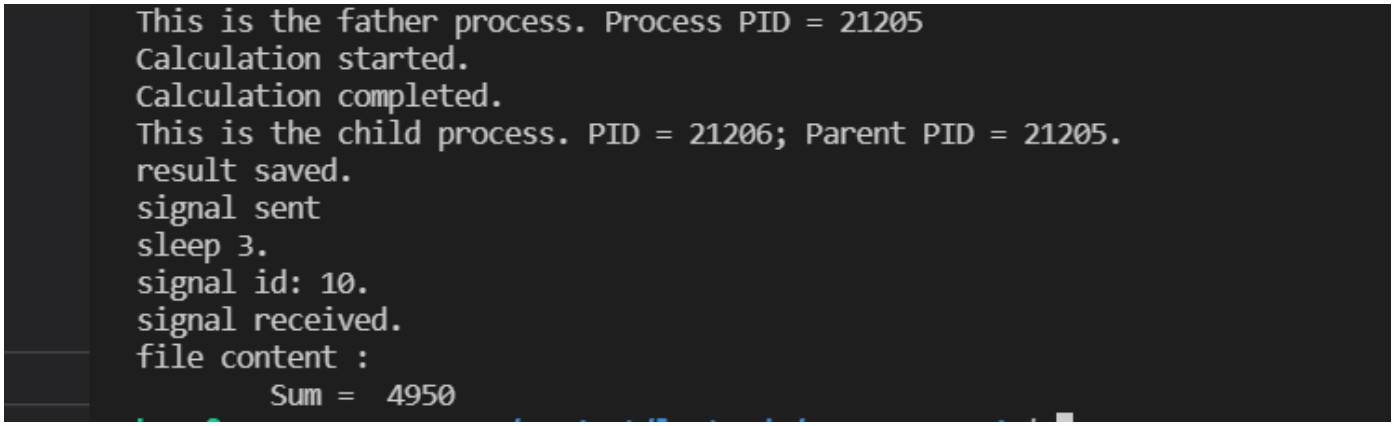
唤醒后读取文件并打印

```
FILE *in = fopen(output, "r");
char readout[50] = {'\0'};

fread(readout, sizeof(char), 50, in);
fclose(in);

printf("file content :\n\t%s\n", readout);
```

2.3 实验流程图


初始化
fork()
yes
pid == -1?
no
pid == 0?
no
yes
父进程
子进程
计算
阻塞等待信号
保存结果到文件
读取文件打印信息
发送信号
退出
睡眠3s
子进程结束?
yes
no
退出
kill子进程

2.4 运行结果



---

3. 进程通信实验

3.1 消息队列通信

3.1.1 实验设计

定义消息结构体

```
struct msg_s {
  long msgtype;                 //消息类型，client to server或server to client
  int length;                   //消息长度
  char text[BUFFER_SIZE];       //消息数组
};
```

client将从文件中读入的消息送入结构体

```
memset(snd.text, 0, sizeof(snd.text));

strcpy(snd.text, buffer);
```

client将结构体放入消息队列

```
msgsnd(qid, &snd, sizeof(snd.text), 0);
```

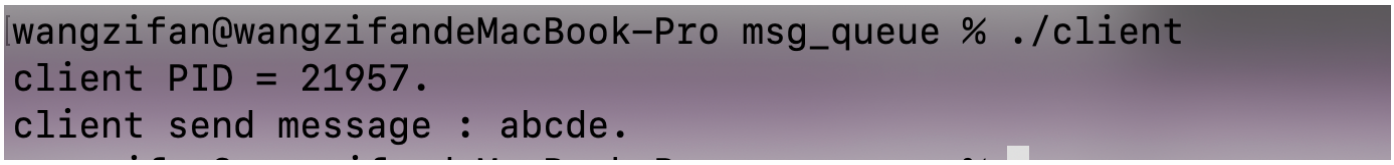
Server通过死循环监听消息队列中的信息

```
while(1)
  {

    // msgtpye from client is C2S: `123`
    msgrcv(qid, &rcv, sizeof(rcv.text), C2S, MSG_NOERROR);

    printf("server received message : %s.\n", rcv.text);
```

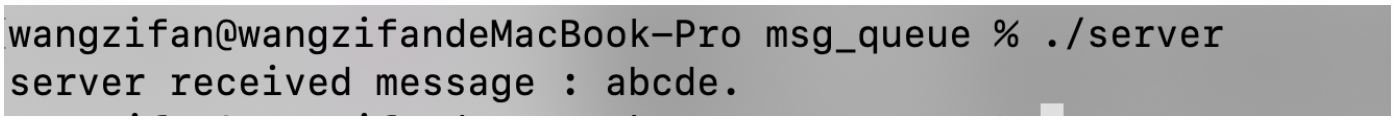
```
    fwrite(rcv.text, sizeof(char), rcv.length, out);  //将收到的信息写入输出文件

    msgsnd(qid, &ret, sizeof(ret.text), 0);           //通知client成功接受

    if(rcv.length < BUFFER_SIZE - sizeof(long))       //没有消息则终止监听
      break;

    memset(rcv.text, 0, sizeof(rcv.text));
  }
```

3.1.2 实验结果

client发送信息



server接收信息



输入、输出文件

3.2 共享内存

3.2.1 实验设计

首先创建共享内存，然后调用fork()创建子进程，直接使父子进程使用同一块内存。

- 互斥：父子进程利用信号量互斥地访问共享内存，分别进行读写操作。
- 同步：父进程读取文本文件内容并写入共享内存。子进程从共享内存中读取内容，写到新的文本文件中。此过程中使用信号量进行同步，即父进程需要等待子进程将内存中的内容读走后才可以写入，子进程需要等待父进程写入内存完毕后才可以进行读取。

3.2.2 实现要点

需要用到三个信号量

```
printf("create semaphores : %d, %d, %d\n", semid_w, semid_r, semid_mutex);
```

- semid_mutex用于父进程与子进程互斥地对共享内存进行操作。
- semid_w：子进程读走内存后调用V(semid_w)，表明内存读取完毕。父进程在写入内存前调用P(semid_w)，根据其结果挂起或者写入。
- semid_r：父进程写入内存后调用V(semid_r)，表明内存写入完毕。子进程在读取内存前调用P(semid_r)，根据其结果挂起或者读取。

进程的互斥

```
P(semid_mutex);                             //对互斥信号量进行判断

char tmpbuffer[SHM_SIZE + 1] = {'\0'};      //内存操作
strcpy(tmpbuffer, memaddr);
tmpbuffer[SHM_SIZE] = '\0';
strcpy(memaddr, tmpbuffer);

V(semid_mutex);                             //信号量+1
```

进程的同步

```
P(semid_r);                                                 //判断是否可以读取

printf("child enter shared memory.\n");                     //读取共享内存
strcpy(buffer_c, memaddr);
buffer_c[SHM_SIZE] = '\0';
fwrite(buffer_c, sizeof(char), strlen(buffer_c), out);
printf("chile leave shared memory\n");

V(semid_w);                                                 //内存现在可以写入
```

```
P(semid_w);                                                 //判断是否可以写入

printf("parent enter shared memory.\n");                    //写入共享内存
strcpy(memaddr, buffer_f);
memset(buffer_f, 0, SHM_SIZE);
printf("parent leave shared memory\n");

V(semid_r);                                                 //内存现在可以读取
```

3.3.3 实验结果

```
[Running] cd "/Users/wangzifan/Desktop/OS/exp2/msg_signal/" && gcc shm_comm.c -o sh
create shared memory, id = 65536.
create semaphores : 82952, 82953, 82954
This is parent process, PID = 22087.
parent enter shared memory.
parent leave shared memory
create shared memory, id = 65536.
create semaphores : 82952, 82953, 82954
This is child process, PID = 22088, parent PID = 22087.
child enter shared memory.
chile leave shared memory

[Done] exited with code=0 in 0.633 seconds
```

输入、输出文件对比