- 将下面的程序编译运行，并解释现象。

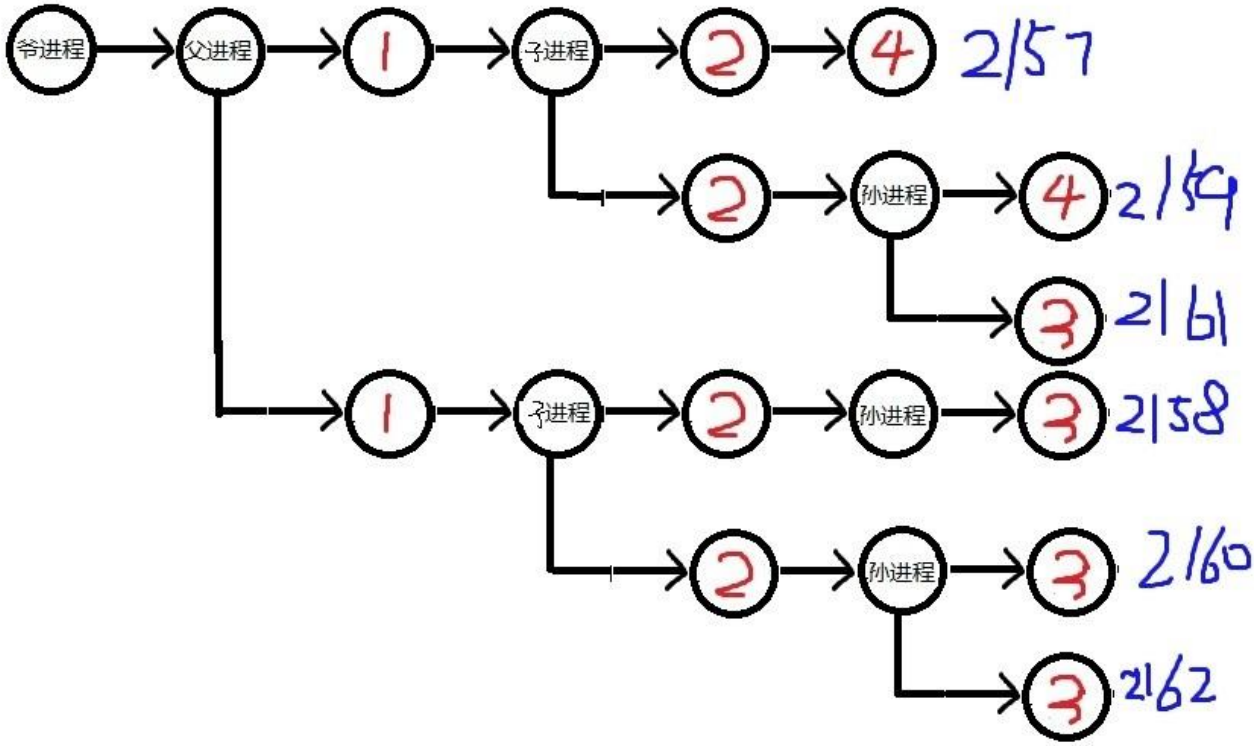
```
void main(){
  int pid1=fork();
  printf("**1**\n");
  int pid2=fork();
  printf("**2**\n");
  if(pid1==0){int pid3=fork();printf("**3**\n");}
  else printf("**4**\n");
}
```

1.cpp

```
#include <unistd.h>
#include <stdio.h>
using namespace std;

int main(){
        int pid1=fork();
        printf("**1** --processid:%d\n",getpid());
        int pid2=fork();
        printf("**2** --processid:%d\n",getpid());
        if(pid1==0){
                int pid3=fork();
                printf("**3** --processid:%d\n",getpid());
        } else
                printf("**4** --processid:%d\n",getpid());
        return 0;
}
```

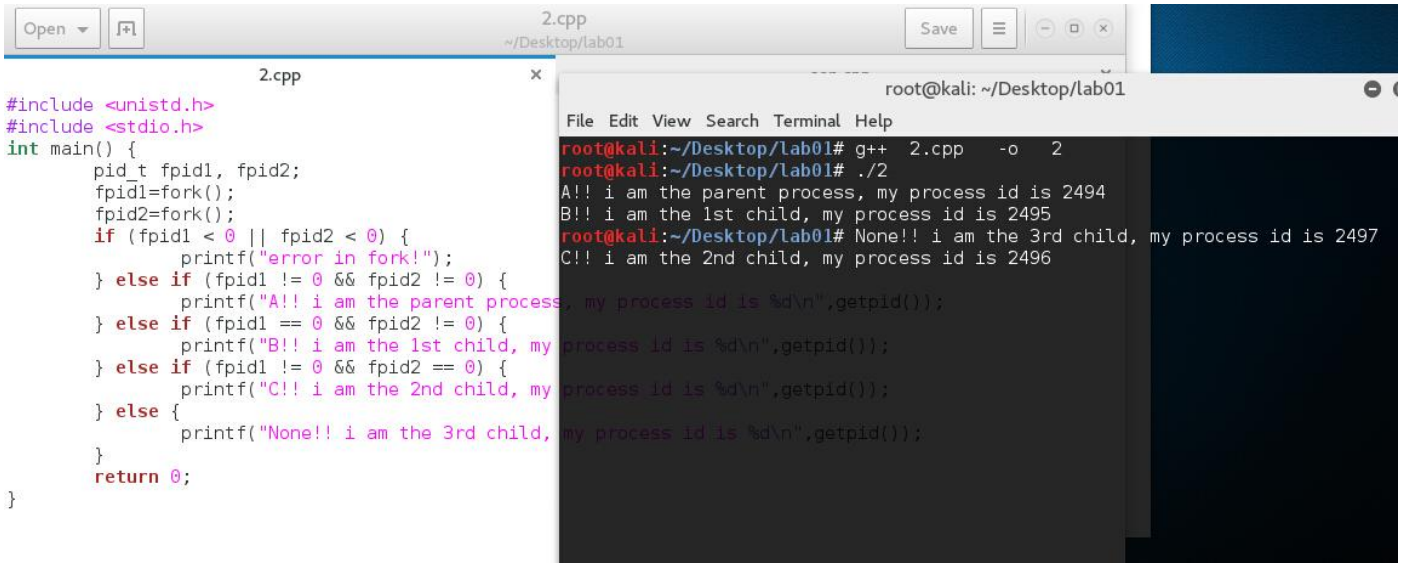
```
File  Edit  View  Search  Terminal  Help
root@kali:~/Desktop/lab01# g++  1.cpp   -o   1
root@kali:~/Desktop/lab01# ./1
**1** --processid:2157
**2** --processid:2157
**4** --processid:2157
**2** --processid:2159
**4** --processid:2159
root@kali:~/Desktop/lab01# **1** --processid:2158
**2** --processid:2158
**3** --processid:2158
**3** --processid:2161
**2** --processid:2160
**3** --processid:2160
**3** --processid:2162
```

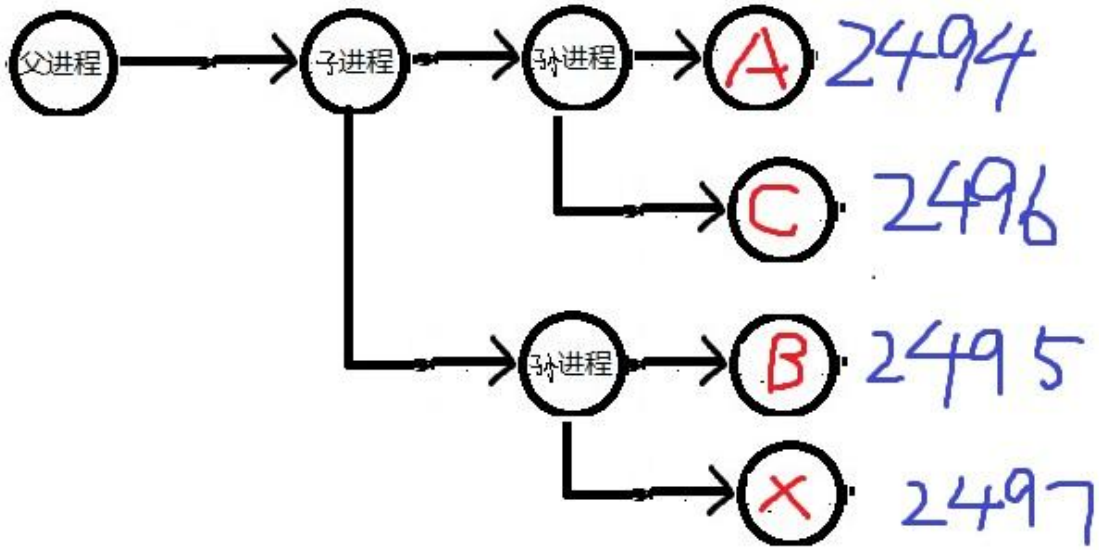


运行结果如上

fork 进程会有竞争，每次顺序可能不一致

- 编写一段程序，使用系统调用fork()创建两个子进程。当此程序运行时，在系统中有一个父进程和两个子进程活动。让每一个进程在屏幕上显示一个字符；父进程显示字符“a”；子进程分别显示字符“b”和字符“c”。试观察记录屏幕上的显示结果，并分析原因。





运行结果如上

主进程 **pid=2494** 打印 **A**

子进程 **pid=2495** 打印 **B**

孙进程 **pid=2496** 打印 **C**

孙进程 **pid=2497** 不打印

下面程序将在屏幕上输出的字符‘X’、数字“1”和“0”各多少个？为什么？

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main(void)
{
  int i, a=0;
  pid_t pid;
if((pid=fork()))a=1;
  for(i=0; i<2; i++){
    printf("X");
  }
  if(pid==0)printf("%d\n",a);
  return 0;
}
```

Open 3.cpp ~/Desktop/lab01

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main(void)
{
    int i, a=0;
    pid_t pid;
if((pid=fork()))a=1;
    for(i=0; i<2; i++){
        printf("X");
    }
    if(pid==0)printf("%d\n",a);
    return 0;
}
```

root@kali: ~/Desktop/lal

File Edit View Search Terminal Help

```
root@kali:~/Desktop/lab01# g++  3.cpp   -o   3
root@kali:~/Desktop/lab01# ./3
XXXX0
root@kali:~/Desktop/lab01#
```

运行结果如上

主进程 pid != 0; a=1；打印 XX 不打印 a=1

子进程 pid ==0; a=0；打印 XX 不打印 a=0

如果将上面main函数修改如下，则屏幕上输出的字符‘X’、数字“1”和“0”各多少个？为什么？

```
int main(void)
{
  int i, a=0;
  pid_t pid[2];
  for(i=0; i<2; i++){
  if((pid[i]=fork()))a=1;
    printf("X");
  }
  if(pid[0]==0)printf("%d\n",a);
  if(pid[1]==0)printf("%d\n",a);
  return 0;
}
```

Open ~/Desktop/lab01 4.cpp

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main(void)
{
    int i, a=0;
    pid_t pid[2];
    for(i=0; i<2; i++){
    if((pid[i]=fork()))a=1;
        printf("X");
    }
    if(pid[0]==0)printf("%d\n",a);
    if(pid[1]==0)printf("%d\n",a);
    return 0;
}
```

root@kali: ~/Desktop/lab

File Edit View Search Terminal Help

```
root@kali:~/Desktop/lab01# g++  4.cpp   -o   4
root@kali:~/Desktop/lab01# ./4
XXXX1
root@kali:~/Desktop/lab01# XX1
XX0
0
```

运行结果如上

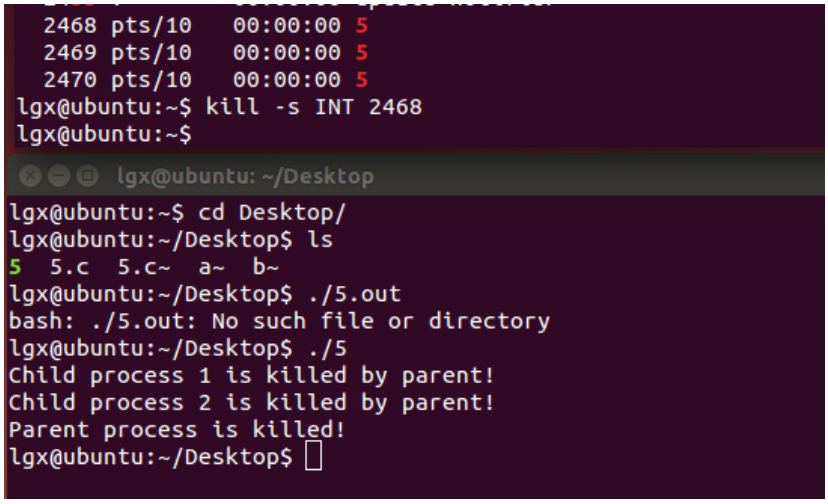
主进程 **pid [0]!=0, pid [1]!=0, a=1，打印 2 个 X，不打印**

子进程 **pid [0]==0, pid [1]!=0, a=1，打印 2 个 X，打印一个 a=1**

孙进程 **pid [0]==0, pid [1]==0, a=0，打印 2 个 X，打印两个 a=0**

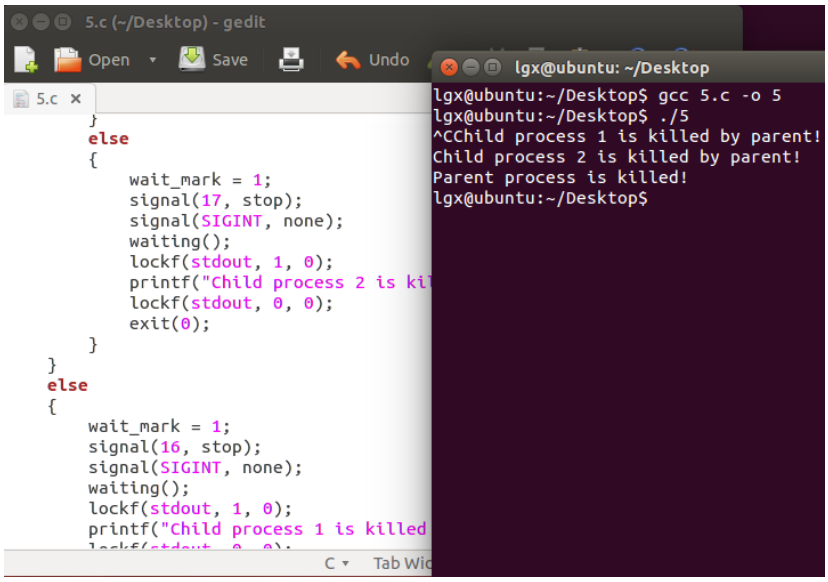
子进程 **pid [0]!=0, pid [0]==0, a=0，打印 2 个 X，打印一个 a=1**

- (a)编制一段程序，使用系统调用fork()创建两个子程序，再用系统调用signal()让父进程捕捉键盘上来的中断信号(即按Ctrl C键)，当捕捉到中断信号后，父进程调用kill()向两个子进程发出信号，子进程捕捉到信号后，分别输出下面信息后终止：
  child process 1 is killed by parent!
  child process 2 is killed by parent!
  父进程等待两个子进程终止后，输出以下信息后终止：
  parent process is killed!



运行结果如上

单独向主进程 **2468** 发送终止信号



稍做修改，让子进程屏蔽 **CTRL+C** 则实现题目功能，在主进程按 **CTRL+C** 显示 **3** 行

- (b)在上述（a）中的程序中增加语句signal(SIGINT, SIG_IGN)和signal(SIGQUIT, SIG_IGN)，观察执行结果并分析原因。这里signal(SIGINT, SIG_IGN)和signal(SIGQUIT, SIG_IGN)分别为忽略“Ctrl Z”键信号以及忽略“Ctrl C”中断信号。

5.c (~/Desktop) - gedit

```c
int main(int argc, const char *argv[])
{
    int p1, p2;
    int stdout;
    while ((p1 = fork()) == -1);
    if (p1 > 0)
    {
        while ((p2 = fork()) == -1);
        if (p2 > 0)
        {
            wait_mark = 1;
            signal(SIGINT, SIG_IGN);
            signal(SIGQUIT, SIG_IGN);
            waiting();
            kill(p1, 16);
            wait(0);
            kill(p2, 17);
            wait(0);
            printf("Parent process is killed!\n");
            exit(0);
```

lgx@ubuntu: ~/Desktop

```
lgx@ubuntu:~/Desktop$ gcc 5.c -o 5
lgx@ubuntu:~/Desktop$ ./5
^C
^Z
[3]+  Stopped                 ./5
lgx@ubuntu:~/Desktop$
```

运行结果如上

屏蔽了 **CTRL+C** 但 **CTRL+Z** 仍结束，不过提示信息没了

3.进程间共享内存实验

- 完成课本第三章的练习3.10的程序
  - 共享内存的系统调用的使用例见课本

5.c × share_example.c × 6.c ×

```c
#include <errno.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#define PERM S_IRUSR|S_IWUSR    //(见注1)
int main(int argc, char **argv)
{
    key_t shmid;
    char   *p_addr, *c_addr;
    pid_t pid;
    if(argc != 2) {
        fprintf(stderr, "Usage:%s\n\a", argv[0]);
        exit(1);
    }
    if( (shmid = shmget(IPC_PRIVATE, 1024, PERM)) == -1
        fprintf(stderr, "Create Share Memory Error:%s\n\
        exit(1);
    }
    pid = fork();
    if(pid > 0) {
        p_addr = shmat(shmid, 0, 0);
        memset(p_addr, '\0', 1024);
        strncpy(p_addr, argv[1], 1024);
        wait(NULL);
        exit(0);
    }
    else if (pid == 0){
        sleep(1);
        c_addr = shmat(shmid, 0, 0);
        printf("Client get %s\n", c_addr);
        exit(0);
    }
}
```

lgx@ubuntu: ~/Desktop

```
lgx@ubuntu:~$ cd Desktop/
lgx@ubuntu:~/Desktop$ gcc share_example.c -o se
lgx@ubuntu:~/Desktop$ ./se
Usage:./se
lgx@ubuntu:~/Desktop$ ./se lalala
Client get lalala
lgx@ubuntu:~/Desktop$
```

测试内存共享函数

```c
#define MAX_SEQUENCE 10
typedef struct {
        long fib[MAX_SEQUENCE];
        int size;
} shared_data, *pshared_data;

shared_data sd;

void calculatefib() {
        sd.fib[0] = 1;
        sd.fib[1] = 1;
        int i;
        for (i = 2; i < MAX_SEQUENCE; i++)
                sd.fib[i] = sd.fib[i-1] + sd.fib[i-2];
        sd.size = MAX_SEQUENCE;
}
```

定义对象和对象指针、计算方法

5.c × share_example.c × 6.c ×

```c
pid = fork();
if(pid > 0) {
    p_addr = shmat(shmid, 0, 0);
    memset(p_addr, '\0', 1024);
    wait(NULL);

    printf("shared memo:");
    pshared_data psd = (pshared_data)p_addr;
    for (i = 0; i < MAX_SEQUENCE; i++)
            printf("%ld ", psd->fib[i]);
    printf("\nsize=%d ", psd->size);
    printf("parent over\n");
    exit(0);
} else if (pid == 0){
    sleep(5);
    c_addr = shmat(shmid, 0, 0);

    calculatefib();
    printf("check data:\n");
    for (i = 0; i < MAX_SEQUENCE; i++)
            printf("%ld ", sd.fib[i]);
    printf("\n\n\n");

    pshared_data psd = (pshared_data)c_addr;
    psd->size = sd.size;
    for (i = 0; i < MAX_SEQUENCE; i++)
            psd->fib[i] = sd.fib[i];

    printf("son over\n");
    exit(0);
}
```

lgx@ubuntu: ~/Desktop

```
lgx@ubuntu:~/Desktop$ gcc 6.c -o 6
lgx@ubuntu:~/Desktop$ ./6
check data:
1 1 2 3 5 8 13 21 34 55


son over
shared memo:1 1 2 3 5 8 13 21 34 55
size=10 parent over
lgx@ubuntu:~/Desktop$
```

子进程进行计算和验算，并将结构数据复制到共享内存（用指针规范化复制）

主进程输出

4.实现shell的要求

- 完成课本上第三章的项目：实现shell。除此之外满足下面要求：
  - 在shell下，按ctrl+C时不会终止shell；

shell.c (~/Desktop) - gedit

```
printf("COMMAND->\n");
setup(inputBuffer,args,&background);

string cmd = args[0];
cout << "     cmd=" << cmd << endl;
if (cmds.size() >= 10) {
  cmds.pop_front();
cout << "     pop" << endl;
}
cmds.push_back(cmd);
cout << "     push->" << cmd << endl;
cout << "     size=" << cmds.size() << en

pid_t pid;
pid = fork();
if (pid < 0)
{
 fprintf(stderr, "Fork Failed");
 exit(-1);
}
else if (pid == 0)
{
 cmd = args[0];
if (cmd == "his") {
       for (pos = cmds.begin(); pos != cm
              cout << *pos << endl;
 } else
```

lgx@ubuntu: ~/Desktop

```
lgx@ubuntu:~/Desktop$ g++ shell.c -o shell
lgx@ubuntu:~/Desktop$ ./shell
COMMAND->
ls
     cmd=ls
     push->ls
     size=1
COMMAND->
5    5.c~  6.c   a~     b~  share_example.c   shell    shell.c~
5.c  6     6.c~  a.out  se  share_example.c~  shell.c
dir
     cmd=dir
     push->dir
     size=2
COMMAND->
5    5.c~  6.c   a~     b~  share_example.c   shell    shell.c~
5.c  6     6.c~  a.out  se  share_example.c~  shell.c
his
     cmd=his
     push->his
     size=3
COMMAND->
ls
dir
his
COMMAND->
```

```
lgx@ubuntu: ~/Desktop
COMMAND->
5     5.c~   6.c    a~      b~   share_example
5.c   6      6.c~   a.out   se   share_example
dir
      cmd=dir
      pop
      push->dir
      size=10
COMMAND->
5     5.c~   6.c    a~      b~   share_example
5.c   6      6.c~   a.out   se   share_example
his
      cmd=his
      pop
      push->his
      size=10
COMMAND->
ls
dir
dir
ls
ls
his
ls
dir
dir
his
COMMAND->
```

显示命令和历史记录，输入 **his** 显示之前信息

```
shell.c (~/Desktop) - gedit
Open  Save  Undo
shell.c  *Untitled Document 1
}

int main(void)
{
 char inputBuffer[MAX_LINE];
    int background;
    char *args[MAX_LINE/2+1];
signal(SIGINT, SIG_IGN);
    while (1)
 {
```

```
lgx@ubuntu: ~/Desktop
lgx@ubuntu:~/Desktop$ g++ shell.c -o shell
lgx@ubuntu:~/Desktop$ ./shell
COMMAND->
^C^C^C^C^C^C^C^C^C^C^C^C^C^C^C^[^A
```

屏蔽 **CTRL+C**