实验六 进程通信——信号

1、通过键盘发送SIGINT信号

> 当用户通过按下`Ctrl-C`组合键给出SIGINT信号时，函数ouch将被调用。程序会在中断函数ouch结束后继续执行，但SIGINT信号处理动作恢复为默认动作。当程序接收到第二个SIGINT信号会结束运行

1. 发送信号：通过键盘的`Ctrl-C`发送终止进程的信号。试通过键盘发送其他信号，观察程序是否有反应。

```
[bexh0lder@XuBinHan_Sun Apr 17_09:35_~/OsHomework/experiment_6/SIGNAL1]$./signal
1_1
Hello World!
Hello World!
Hello World!
^COUCH! - I got signal 2
Hello World!
Hello World!
Hello World!
^C
[bexh0lder@XuBinHan_Sun Apr 17_09:35_~/OsHomework/experiment_6/SIGNAL1]$./signal
1_1
Hello World!
Hello World!
Hello World!
^Z
[1]+  Stopped                 ./signal1_1
[bexh0lder@XuBinHan_Sun Apr 17_09:35_~/OsHomework/experiment_6/SIGNAL1]$
```

2. 预置对信号的处理方式：通过signal设置信号的处理方式，删除语句①或②，观察程序变化

```
[bexh0lder@XuBinHan_Sun Apr 17_09:38_~/OsHomework/experiment_6/SIGNAL1]$./signal
1_2
Hello World!
Hello World!
Hello World!
^COUCH! - I got signal 2
Hello World!
Hello World!
^COUCH! - I got signal 2
Hello World!
Hello World!
^COUCH! - I got signal 2
Hello World!
Hello World!
^Z
[2]+  Stopped                 ./signal1_2
[bexh0lder@XuBinHan_Sun Apr 17_09:38_~/OsHomework/experiment_6/SIGNAL1]$./signal
1_3
Hello World!
Hello World!
Hello World!
^C
[bexh0lder@XuBinHan_Sun Apr 17_09:38_~/OsHomework/experiment_6/SIGNAL1]$
```


删除①后按`Ctrl-C`组合键无法退出进程，删除②后按`Ctrl-C`组合键不会执行ouch函数而是直接退出进程

3. 接收信号的进程按事先规定完成对相应事件的处理，函数`ouch(int sig)`安排设置接收到信号后实施的动作

```
void (__cdecl *signal(int sig, void (*func)(int)))(int)
void (
    __cdecl *signal(
```

```
        int sig,/*信号值*/
        void (*func)(int)
        /*func是一个函数指针， func函数指针指向一个带一个整型参数,并且返回值为void的一个函数*/
    )(int)
    /*signal()函数的返回值也为一个函数指针,这个函数指针指向一个带一个整型参数,并且返回值为void的一个函数*/
)

/*如果参数func不是函数指针，则必须是下列两个常数之一：
    SIG_IGN 忽略参数signum指定的信号
    SIG_DFL 将参数signnum指定的信号量重设为核心预设的信号处理方式*/

/*简化*/
typedef void Sigfunc(int)
/*Sigfunc就代表的就是一个 返回值是一个无返回值，有一个int参数的函数*/
Sigfunc *signal(int, Sigfunc*)

/*简单实例*/
void ouch(int sig)
{
    printf("OUCH! - I got signal %d\n", sig);
  //恢复SIGINT信号的处理动作
   (void) signal(SIGINT, SIG_DFL);
}
void) signal(SIGINT, ouch);/*设置SIGINT信号的处理动作为响应ouch函数*/
```

2、闹钟，通过系统调用`alarm()`定时发送信号

思考：理解程序通过`alarm()`定时发送信号

```
unsigned int alarm（
    unsigned int seconds/*指定秒数*/
    );
/*当定时器指定的时间到时，它向进程发送SIGALRM信号*/
```

```
[bexh0lder@XuBinHan_Sun Apr 17_10:35_~/OsHomework/experiment_6/SIGNAL1]$./signal
2
sleep 1 ...
sleep 2 ...
sleep 3 ...
hello
sleep 4 ...
sleep 5 ...
sleep 6 ...
[bexh0lder@XuBinHan_Sun Apr 17_10:35_~/OsHomework/experiment_6/SIGNAL1]$
```

程序设置每三秒向进程发送一次SIGALRM信号，在此之前已经用`signal`设置SIGALRM信号的处理函数，每次发送SIGALRM信号都会输出一个hello

3、使用`setitimer()`和`getitimer()`设置定时器和获得定时器状态

```
[bexh0lder@XuBinHan_Sun Apr 17_11:09_~/OsHomework/experiment_6/SIGNAL1]$./signal
3
process id is 4664
Catch a signal -- SIGVTALRM
Catch a signal -- SIGVTALRM
Catch a signal -- SIGVTALRM
Catch a signal -- SIGALRM
Catch a signal -- SIGVTALRM
Catch a signal -- SIGVTALRM
Catch a signal -- SIGVTALRM
Catch a signal -- SIGVTALRM
Catch a signal -- SIGALRM
Catch a signal -- SIGVTALRM
Catch a signal -- SIGVTALRM
Catch a signal -- SIGVTALRM
Catch a signal -- SIGVTALRM
Catch a signal -- SIGALRM
Catch a signal -- SIGVTALRM
Catch a signal -- SIGVTALRM
Catch a signal -- SIGVTALRM
Catch a signal -- SIGVTALRM
^C
[bexh0lder@XuBinHan_Sun Apr 17_11:13_~/OsHomework/experiment_6/SIGNAL1]$
```

1. 理解`sys\time.h`中关于`struct itimerval`的定义

```
struct itimerval {

        struct timeval it_interval; /* 第一次之后每隔多长时间 */

        struct timeval it_value; /* 第一次调用要多长时间 */

};
```

2. 了解定时器`ITIMER_REAL`和`ITIMER_VIRTUAL`的差异，理解实际时间和进程执行时间的概念

```
int setitimer(
    int which, /*which指定定时器类型*/
    const struct itimerval *new_value, /*是结构itimerval的一个实例，结构
itimerval形式*/
    struct itimerval *old_valuev/*可不做处理*/
);
```

`ITIMER_REAL`是以系统真实时间来计算的，并且发送给进程的信号是`SIGALRM`

`ITIMER_VIRTUAL`是以进程在用户态执行时间计算的，并且发送给给进程的信号是`SIGVTALRM`

`ITIMER_PROF`是以进程运行或系统代表进程运行时间计算的，并且发送给给进程的信号是`SIGPROF`

系统真实时间：进程从开始执行到最后结束的时间，包括阻塞 + 就绪 + 运行的时间。称为 wall clock time/墙上时钟时间/elpapsed time，是我们跑程序实际等待的时间；

进程在内核态执行时间：用户进程获得CPU资源后，在内核态的执行时间，如write，read等系统调用；

进程在用户态执行时间：用户进程获得CPU资源后，在用户态执行的时间，主要是我们自己编写的代码；

其中可大致认为：

系统真实时间 = 阻塞时间 + 就绪时间 + CPU运行时间

CPU运行时间 = 进程在用户态执行时间 + 进程在内核态执行时间

4、父子进程间通过kill发送信号

> 用`fork()`创建两个子进程，子进程在等待5秒后用系统调用kill()向父进程发送SIGALARM信号，父进程调用`signal()`捕捉SIGLARM信号。

```
int kill(
    pid_t pid,
    /*信号发射对象
    pid：可能选择有以下四种
    pid大于零时，pid是信号欲送往的进程的标识。
    pid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。
    pid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。*/
    int sig
    /*所发送的信号值，假如其值为零则没有任何信号送出，但是系统会执行错误检查，通常会利用sig值为零来检验某个进程是否仍在执行*/
);
```

1. 理解进程调用`kill()`和`signal()`的功能和使用方法

   子进程调用`kill()`向父进程发信号，父进程调用`signal()`设置处理子进程发来的信号的处理函数

2. 掌握父子进程获取对方进程号的方法

   子进程通过`getppid()`获取父进程进程号，父进程获取子进程号可以通过在调用`fork()`生成子进程的第一个返回值来获取

3. 取消语句①，观察变化，解释原因

   ```
   [bexh0lder@XuBinHan_Sun Apr 17_12:26_~/OsHomework/experiment_6/SIGNAL1]$./signal4_2
   alarm application starting
   waiting for alarm to go off
   done
   [bexh0lder@XuBinHan_Sun Apr 17_12:26_~/OsHomework/experiment_6/SIGNAL1]$
   ```

   执行这个还会让我的Ubuntu出问题，没有语句①的pause()会使父进程不等待子进程直接执行，从而父进程提前结束，子进程成为孤儿进程

4. 取消语句①，保留语句②，观察变化，解释原因

   ```
   [bexh0lder@XuBinHan_Sun Apr 17_12:28_~/OsHomework/experiment_6/SIGNAL1]$./signal4_3
   alarm application starting
   waiting for alarm to go off
   Ding!
   done
   [bexh0lder@XuBinHan_Sun Apr 17_12:29_~/OsHomework/experiment_6/SIGNAL1]$
   ```

   语句②的wait()会起到等待子进程结束的效果，从而让父进程能挂起等待子进程执行完，从而使父进程能正常接收到信号正常执行

5、进程使用信号通信

1. 了解父进程利用fork的返回值，向子进程发送信号

   父进程调用fork()后fork()第一次返回值返回的是子进程的进程id，从而使父进程可以根据进程号向子进程传递信号

2. 观察下图各种可能的输出，尝试进行互斥控制，保证两个子进程的输出不交替

```
[bexh0lder@XuBinHan_Sun Apr 17_14:10_~/OsHomework/experiment_6/SIGNAL1]$./signal
5
^CP2 is killed by parent 1
P1 is killed by parent 1
P1 is killed by parent 2
P2 is killed by parent 2
parents is killed
[bexh0lder@XuBinHan_Sun Apr 17_14:11_~/OsHomework/experiment_6/SIGNAL1]$./signal
5
^CP1 is killed by parent 1
P1 is killed by parent 2
P2 is killed by parent 1
P2 is killed by parent 2
parents is killed
[bexh0lder@XuBinHan_Sun Apr 17_14:25_~/OsHomework/experiment_6/SIGNAL1]$./signal
5
^CP1 is killed by parent 1
P2 is killed by parent 1
P2 is killed by parent 2
P1 is killed by parent 2
parents is killed
[bexh0lder@XuBinHan_Sun Apr 17_14:25_~/OsHomework/experiment_6/SIGNAL1]$
```

6、使用sigaction注册信号

```
[bexh0lder@XuBinHan_Sun Apr 17_16:39_~/OsHomework/experiment_6/sigaction]$./siga
ction1 38
wait for the signal
wait for the signal
wait for the signal
wait for the signal
wait for the signal
wait for the signal
wait for the signal
wait for the signal
wait for the signal
wait for the signal
wait for the signal
^C
[bexh0lder@XuBinHan_Sun Apr 17_16:39_~/OsHomework/experiment_6/sigaction]$./siga
ction1 2
wait for the signal
^Creceive signal 2
wait for the signal
^Creceive signal 2
wait for the signal
^Creceive signal 2
^Creceive signal 2
wait for the signal
```

1. 了解sigaction结构的定义

```
struct sigaction {
    union{
        __sighandler_t _sa_handler;
        void (*_sa_sigaction)(int,struct siginfo_t *, void *);
        /*由_sa_sigaction 是指定的信号处理函数带有三个参数，是为实时信号而设的（当然同 样支持非实时信号），它指定一个 3 参数信号处理函数。第一个参数为信号值，第三个参数 没有使用，第二个参数是指向 siginfo_t 结构的指针*/
    }_u
        /*联合数据结构中的两个元素_sa_handler 以及*_sa_sigaction 指定信号关联函数，即用户指定的信号处理函数。除了可以是用户自定义的处理函数外，还可以为 SIG_DFL(采用缺 省的处理方式)，也可以为 SIG_IGN（忽略信号）,二选一*/
        sigset_t sa_mask;
    /*sa_mask 指定在信号处理程序执行过程中，哪些信号应当被阻塞。缺省情况下当前信 号本身被阻塞，防止信号的嵌套发送，除非指定 SA_NODEFER 或者 SA_NOMASK 标志位。*/
        unsigned long sa_flags;
    /*sa_flags 中包含了许多标志位，包括刚刚提到的 SA_NODEFER 及 SA_NOMASK 标 志位。另一个比较重要的标志位是 SA_SIGINFO，当设定了该标志位时，表示信号附带的参 数可以被传递到信号处理函数中，因此，应该为 sigaction 结构中的 sa_sigaction 指定处理函 数，而不应该为 sa_handler 指定信号处理函数，否则，设置该标志变得毫无意义。即使为 sa_sigaction 指定了信号处理函数，如果不设置 SA_SIGINFO，信号处理函数同样不能得到 信号传递过来的数据，在信号处理函数中对这些信息的访问都将导致段错误（Segmentation fault）。*/
}
```

2. 了解使用`sigaction()`函数注册信号

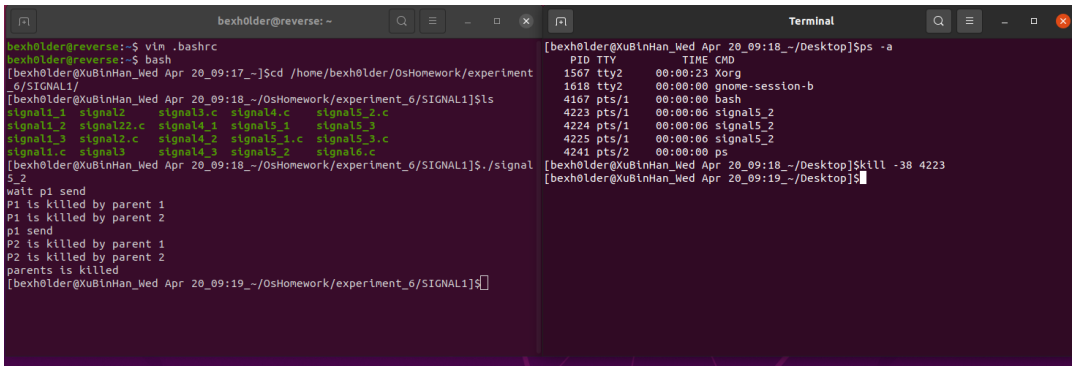
```
int sigaction(
    int sig,
    /*信号量的值，可以为除SIGKILL和SIGSTOP之外的任意一个特定有效信号，安装这两个信号会导致信号安装错误*/
    const struct sigaction *act,
    /*执行sigaction结构体的指针，由sigaction结构体确定对信号量的处理，如果为空，进程会以缺省方式对信号处理*/
    struct sigaction *oact
    /*保存返回的原来对相应信号的处理，可指定为NULL*/
);
```

3. 理解信号注册，信号发送，信号处理函数三个重要环节
   - 方法一：程序首先设置sigaction结构体，然后使用`sigaction()`函数注册信号，再等待信号的发出，在接收到信号之后对信号进行相应处理

```
#include<stdio.h>
#include<signal.h>
#include<unistd.h>
#include<stdlib.h>

int wait_mark;
void waiting(), stop();

int main()
{
    int p1, p2;
    //signal(SIGINT, stop);
```

```
    while((p1 = fork()) == -1);
    if(p1 > 0)
    {
        while((p2 = fork()) == -1);
        if(p2 > 0)
        {
            signal(38, stop);
            wait_mark = 1;
            waiting();
            kill(p1, 16);
            printf("wait p1 send\n");
            fflush(stdout);
            wait_mark = 1;
            waiting(); /*等待子进程发信号表示输出完再向另一个子进程发信号，保证子
进程之间输出不交替*/
            kill(p2, 17);
            wait(0);
            wait(0);
            printf("parents is killed \n");
            exit(0);
        }
        else
        {
            wait_mark = 1;
            signal(17, stop);
            waiting();
            //lockf(1,F_LOCK,100);
            printf("P2 is killed by parent 1\n");
            fflush(stdout);
            sleep(1);
            printf("P2 is killed by parent 2\n");
            fflush(stdout);
            //lockf(1,F_ULOCK,100);
            // kill(getppid(), SIGINT);
            exit(0);
        }
    }
    else
    {
        wait_mark = 1;
            signal(16, stop);
            waiting();
            //lockf(1,F_LOCK,100);
            printf("P1 is killed by parent 1\n");
            fflush(stdout);
            sleep(1);
            printf("P1 is killed by parent 2\n");
            fflush(stdout);
            //lockf(1,F_ULOCK,100);
            printf("p1 send\n");
            fflush(stdout);
            kill(getppid(), 38);
            exit(0);
    }
}
void waiting()
{
    while(wait_mark != 0);
```

```
}

void stop()
{
    wait_mark = 0;
}
```



- 方法二：使用系统调用 lockf 函数

```
#include<stdio.h>
#include<signal.h>
#include<unistd.h>
#include<stdlib.h>

int wait_mark;
void waiting(), stop();

int main()
{
    int p1, p2;
    signal(SIGINT, stop);
    while((p1 = fork()) == -1);
    if(p1 > 0)
    {
        while((p2 = fork()) == -1);
        if(p2 > 0)
        {
            //signal(38, stop);
            wait_mark = 1;
            waiting();
            kill(p1, 16);
            //printf("wait p1 send\n");
            //fflush(stdout);
            //wait_mark = 1;
            //waiting(); /*等待子进程发信号表示输出完再向另一个子进程发信号，保证子进程之间输出不交替*/
            kill(p2, 17);
            wait(0);
            wait(0);
            printf("parents is killed \n");
            exit(0);
        }
        else
        {
            wait_mark = 1;
            signal(17, stop);
            waiting();
```

```
            lockf(1,F_LOCK,100);
            printf("P2 is killed by parent 1\n");
            fflush(stdout);
            sleep(1);
            printf("P2 is killed by parent 2\n");
            fflush(stdout);
            lockf(1,F_ULOCK,100);
            // kill(getppid(), SIGINT);
            exit(0);
        }
    }
    else
    {
        wait_mark = 1;
            signal(16, stop);
            waiting();
            lockf(1,F_LOCK,100);
            printf("P1 is killed by parent 1\n");
            fflush(stdout);
            sleep(1);
            printf("P1 is killed by parent 2\n");
            fflush(stdout);
            lockf(1,F_ULOCK,100);
            //printf("p1 send\n");
            //fflush(stdout);
            //kill(getppid(), 38);
            exit(0);
    }
}
void waiting()
{
    while(wait_mark != 0);
}

void stop()
{
    wait_mark = 0;
}
```

```
[bexh0lder@XuBinHan_Wed Apr 20_09:19_~/OsHomework/experiment_6/SIGNAL1]$./signal
5_3
^CP1 is killed by parent 1
P1 is killed by parent 2
P2 is killed by parent 1
P2 is killed by parent 2
parents is killed
[bexh0lder@XuBinHan_Wed Apr 20_09:20_~/OsHomework/experiment_6/SIGNAL1]$
```

7、使用sigqueue发送信号并传递附加信息

1. 了解`union sigval`的定义

```
typedef union sigval
{
    int sival_int; /*通信时传递的数值数据*/
    void *sival_ptr; /*通信时传递的字符串数据*/
}sigval_t;
/*联合数据结构中的两个元素sival_int 以及*sival_ptr 指定传递的数据，数值数据和字符串数据二选一*/
```

2. 了解使用`sigqueue()`函数发送信号

```
int sigqueue(
    pid_t pid,/*指定接收信号的进程id,不能发送信号给一个进程组。如果 signo=0，将会执行错误检查，但实际上不发送任何信号，0值信号可用于检查 pid 的有效性以及当前进程是否有权限向目标进程发送信号*/
    int sig, /*确定即将发送的信号*/
    const union sigval value/*一个联合数据结构union sigval，指定了信号传递的参数，即通常所说的4字节值。*/
);
```

3. 观察进程给自身发送信号

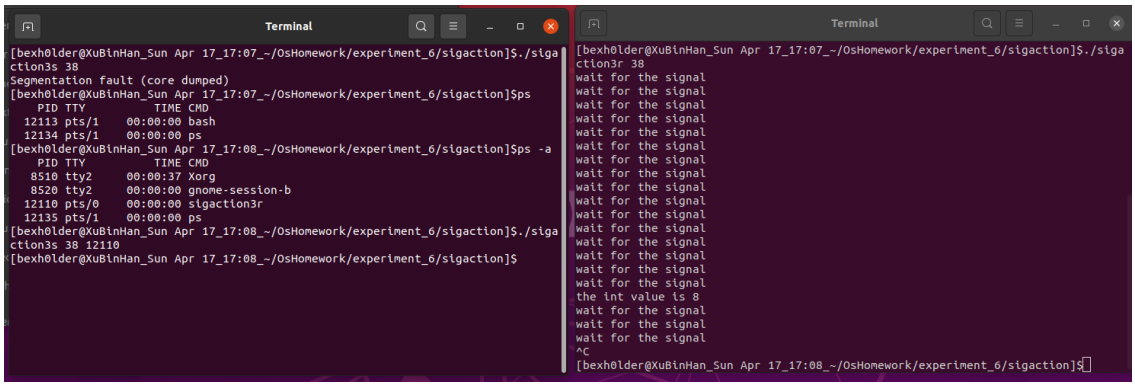
```
[bexh0lder@XuBinHan_Sun Apr 17_17:00_~/OsHomework/experiment_6/sigaction]$./siga
ction2 38
wait for the signal
$ $ $ $ $ $ $ $ $ $
handle signal 38 over;

wait for the signal
$ $ $ $ $ $ $ $ $ $
handle signal 38 over;

wait for the signal
$ $ $ $ $ $ $ $ $ $
handle signal 38 over;

wait for the signal
$ $ $ $ $ $ $ $ $ $
handle signal 38 over;

wait for the signal
$ $ $ $ $ $ $ $ $ $
handle signal 38 over;
```

8、修改前面的例程，把信号发送和接收放在两个程序种，并且在发送过程种传递整型参数

1. 理解发送进程和接收进程
   发送进程通过往`union sigval`结构体写入内容并将其和信号量一起通过`sigqueue()`函数发送给接收进程来传递信息

   接收进程通过`sigaction()`安装信号处理函数来处理和信号一同接收到的信息

2. 按程序要求在运行时附加合适的参数，并运行检验



编程题

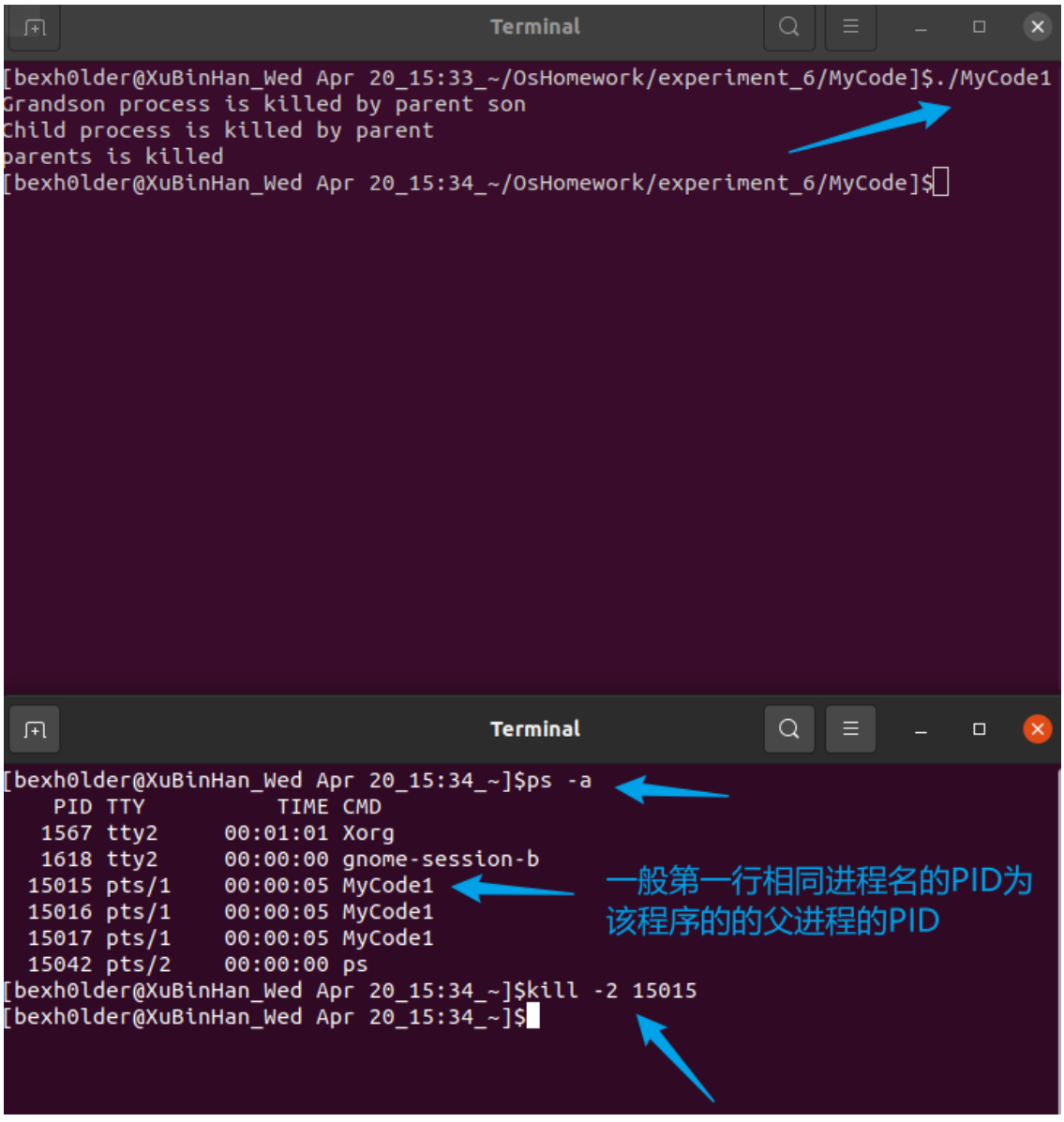
1.

```c
#include<stdio.h>
#include<signal.h>
#include<unistd.h>
#include<stdlib.h>
#include<sys/wait.h>

int wait_mark;
void waiting(), stop();

int main()
{
    int p1, p2;
    while((p1 = fork()) == -1);
    if(p1 > 0) //父进程区域
    {
        wait_mark = 1;
        signal(SIGINT, stop);
        waiting();
        kill(p1, 16);//父进程向子进程（p1）发送信号
        wait(0);
        printf("parents is killed \n"); //有\n所以不用fflush(stdout)
        exit(0);
    }
    else
    {
        while((p2 = fork()) == -1);
        if(p2 > 0) //子进程区域
        {
            wait_mark = 1;
            signal(16, stop);
            waiting();
            kill(p2, 17);//子进程向孙子进程（p2）发送信号
            wait(0);
            printf("Child process is killed by parent\n");
            sleep(1);
            exit(0);
        }
        else //孙子进程区域
        {
            wait_mark = 1;
            signal(17, stop);
```

```
            waiting();
            printf("Grandson process is killed by parent son\n");
            sleep(1);
            exit(0);
        }
    }
}
void waiting()
{
    while(wait_mark != 0);
}

void stop()
{
    wait_mark = 0;
}
```

注意这里不能直接通过运行程序后按CTRL + C来给父进程信号，因为这样的话CTRL + C给出的信号可能会发送给所有进程，包括父进程、子进程和孙子进程，所有这里推荐打开另一个shell输入下列命令来给父进程发送信号

```
kill -2 父进程的PID
PID可以使用 ps -a 来查看
```



2.

```c
#include<stdio.h>
#include<signal.h>
#include<unistd.h>
#include<stdlib.h>

int wait_mark;
int count;
void waiting(), stop();

int main()
{
    int p1, p2;
    while((p1 = fork()) == -1);
    if(p1 > 0) //父进程区域
    {
        count = 0;
        signal(16, stop);
        while(1)
        {
            sleep(1);
            kill(p1, 17);//父进程向子进程发送信号
            wait_mark = 1;
            waiting();
            count ++;
            printf("parent process caught signal #%d\n",count);
        }
    }
    else
    {
        count = 0;
        signal(17, stop);
        while(1)
        {
            wait_mark = 1;
            waiting();
            count ++;
            printf("child process caught signal #%d\n",count);
            sleep(1);
            kill(getppid(), 16);//子进程向父进程发送信号
        }
    }
}

void waiting()
{
    while(wait_mark != 0);
}

void stop()
{
    wait_mark = 0;
}
```

```
[bexh0lder@XuBinHan_Wed Apr 20_11:42_~/OsHomework/experiment_6/MyCode]$./MyCode2
child process caught signal #1
parent process caught signal #1
child process caught signal #2
parent process caught signal #2
child process caught signal #3
parent process caught signal #3
child process caught signal #4
parent process caught signal #4
child process caught signal #5
parent process caught signal #5
child process caught signal #6
parent process caught signal #6
child process caught signal #7
parent process caught signal #7
child process caught signal #8
```

3.

```c
#include<stdio.h>
#include<signal.h>
#include<unistd.h>
#include<stdlib.h>
#include<sys/wait.h>
#include<string.h>

int wait_mark;
void waiting(), stop();

int main()
{
    int p1, p2;
    while((p1 = fork()) == -1);
    if(p1 > 0) //父进程区域
    {
        wait_mark = 1;
        struct sigaction act;
        sigemptyset(&act.sa_mask);
        act.sa_sigaction=stop;//三参数信号处理函数
        act.sa_flags=SA_SIGINFO;//信息传递开关，允许传送参数信息给new_op
        if(sigaction(SIGINT,&act,NULL) < 0)
        {
            printf("install sigal error\n");
        }
        union sigval mysigval;
        //char data[10];
        //memset(data,0,sizeof(data));
        //for(int i=0;i < 10;i++)
        //  data[i]='$';
        //mysigval.sival_ptr=data;
        mysigval.sival_int=233;//不代表具体含义，只用于说明问题
        //char data[] = "you success!1";
        //mysigval.sival_ptr=data;
        waiting();
        sigqueue(p1,16,mysigval);//父进程向子进程（p1）发送信号，并传递附加信息
        wait(0);
        printf("parents is killed \n"); //有\n所以不用fflush(stdout)
        sleep(1);
        exit(0);
    }
```

```
    else
    {
        while((p2 = fork()) == -1);
        if(p2 > 0) //子进程区域
        {
            wait_mark = 1;
            struct sigaction act;
            sigemptyset(&act.sa_mask);
            act.sa_sigaction=stop;//三参数信号处理函数
            act.sa_flags=SA_SIGINFO;//信息传递开关，允许传送参数信息给new_op
            union sigval mysigval;
            //char data[10];
            //memset(data,0,sizeof(data));
            //for(int i=0;i < 10;i++)
            //  data[i]='$';
            //mysigval.sival_ptr=data;
            mysigval.sival_int=888;//不代表具体含义，只用于说明问题
            //char data[] = "you success2!";
            //mysigval.sival_ptr=data;
            if(sigaction(16,&act,NULL) < 0)
            {
                printf("install sigal error\n");
            }
            waiting();
            sigqueue(p2,17,mysigval);//父进程向子进程（p1）发送信号，并传递附加信息
            wait(0);
            printf("Child process is killed by parent\n");
            sleep(1);
            exit(0);
        }
        else //孙子进程区域
        {
            wait_mark = 1;
            struct sigaction act;
            sigemptyset(&act.sa_mask);
            act.sa_sigaction=stop;//三参数信号处理函数
            act.sa_flags=SA_SIGINFO;//信息传递开关，允许传送参数信息给new_op
            if(sigaction(17,&act,NULL) < 0)
            {
                printf("install sigal error\n");
            }
            waiting();
            printf("Grandson process is killed by son\n");
            sleep(1);
            exit(0);
        }
    }
}
void waiting()
{
    while(wait_mark != 0);
}

void stop(int signum,siginfo_t *info,void *myact)
{
    int i;
    if(signum == 2) printf("Oh my father, I see you, you are my real
father\n");
```

```
    else printf("Oh my father, I am %d, you have sent a number %d\n",
signum, info->si_int);
    //for(i=0;i<10;i++){
    //  printf("%c ",(*( (char*)((*info).si_ptr)+i)));
    //}
    // printf("\nhandle signal %d over;\n\n",signum);
    wait_mark = 0;
}
```

这里的操作和编程一相同