实验内容

（1）使用 man 命令查看 fork 、 kill 、 signal、 sleep、 exit 系统调用的帮助手册。

（2）根据流程图（如图 2.1 所示） 编制实现软中断通信的程序： 使用系统调用 fork()创建两个子进程，再用系统调用 signal()让父进程捕捉键盘上发出的中断信号（即 5s 内按下delete 键或 quit 键），当父进程接收到这两个软中断的某一个后，父进程用系统调用 kill()向两个子进程分别发出整数值为 16 和 17 软中断信号，子进程获得对应软中断信号，然后分别输出下列信息后终止：

Child process 1 is killed by parent !! Child process 2 is killed by parent !!

父进程调用 wait()函数等待两个子进程终止后，输出以下信息，结束进程执行： Parent process is killed!!

注： delete 会向进程发送 SIGINT 信号， quit 会向进程发送 SIGQUIT 信号。 ctrl+c 为delete， ctrl+\为 quit 。

（3）多次运行所写程序，比较 5s 内按下 Ctrl+\或 Ctrl+Delete 发送中断，或 5s 内不进行任何操作发送中断， 分别会出现什么结果？分析原因。

（4）将本实验中通信产生的中断通过 14 号信号值进行闹钟中断，体会不同中断的执行样式，从而对软中断机制有一个更好的理解

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

void signal_handler(int signal_num) {
  // printf("Received interrupt signal:
%d\n", signal_num);
}

void child_signal_handler(int signal_num) {
  if (signal_num == 16) {
    printf("Child process 1 is killed by
parent !!\n");
  }
  if (signal_num == 17) {
    printf("Child process 2 is killed by
parent !!\n");
  }
  exit(0);
}

int main() {
  pid_t child1, child2;
```

```
  signal(SIGINT, signal_handler);   //
(Ctrl+C)
  signal(SIGQUIT, signal_handler);  //
(Ctrl+\)

  child1 = fork();
  if (child1 == 0) {
    signal(16, child_signal_handler);
    while (1) {
    }
  }

  child2 = fork();
  if (child2 == 0) {
    signal(17, child_signal_handler);
    while (1) {
    }
  }

  if (child1 > 0 && child2 > 0) {
    alarm(5);
    pause();

    kill(child1, 16);
    kill(child2, 17);

    // wait two child process to exit
    wait(NULL);
    wait(NULL);
    printf("Parent process is killed!!\n");
```

```
    }

    return 0;
}
```

```
cold@ubuntu:~/Desktop/xjtuOSlab$ gcc 2.1.1.c -o 2.1.1
cold@ubuntu:~/Desktop/xjtuOSlab$ ./2.1.1
^CChild process 1 is killed by parent !!
Child process 2 is killed by parent !!
Parent process is killed!!
cold@ubuntu:~/Desktop/xjtuOSlab$ ./2.1.1
^\Child process 2 is killed by parent !!
Child process 1 is killed by parent !!
Parent process is killed!!
cold@ubuntu:~/Desktop/xjtuOSlab$ ./2.1.1
Alarm clock
```

观察三次输出可以看到，子进程结束的顺序并不固定，如果超时会通过alarm来退出。这符合我一开始对于程序的设计。因为超时没有传递中断或者退出子进程就无法被结束。（最重要的是alarm信号没有被处理。）现在我希望改进代码，我增加一个alarm_handle：

```
void signal_handler(int signal_num) {
    if (signal_num == SIGALRM) {
        printf("the signal is timeout proc
stopped auto!!\n");
    }
    // printf("Received interrupt signal:
%d\n", signal_num);
}
```

```
cold@ubuntu:~/Desktop/xjtuOSlab$ gcc 2.1.1.c -o 2.1.1
cold@ubuntu:~/Desktop/xjtuOSlab$ ./2.1.1
the signal is timeout proc stopped auto!!
Child process 1 is killed by parent !!
Child process 2 is killed by parent !!
Parent process is killed!!
```

此时我发现实验手册要求我使用闹钟中断——alarm，我在前面已经实现完毕。只能说：无巧不成书！

kill命令我只调用了两次。调用后一方面是中断了对应的子进程，另一方面发出了一个signal，输出相关信息。

（5）使用 kill 命令可以在进程的外部杀死进程。进程怎样能主动退出？这两种退出方式哪种更好一些？

对于这个问题，我认为可以在进程内调用exit(0)、return等方式来退出。后者更好一些。kill我理解的类似于Windows平台设备管理器的“结束进程”，或者说强制关机，这是一个具有风险的行为，尤其是对于易失性的信息，很可能直接丢失。因此kill不到万不得已不太应该使用。