# 实验四 进程通信——信号

**实验目的：**

1. 理解IPC通信中的信号通信原理和基本技术；
2. 掌握在Linux环境中构造信号通信机制的方法和步骤。

**实验内容：**

1. 在Linux环境下，用C语言编程；
2. 实现进程间通过信号进行通信。

**实验步骤：**

信号是一种异步通信机制，进程不需要执行任何操作来等待信号的到达。

进程之间可以互相通过系统调用kill发送软中断信号。内核也可以因为内部事件而给进程发送信号，通知进程发生了某个事件。

信号异步通知接收信号的进程发生了某个事件，然后操作系统将会中断接收到信号的进程的执行，转而去执行相应的信号处理程序。进程通过系统调用signal来指定进程对某个信号的处理行为。

收到信号的进程对各种信号有不同的处理方法。处理方法可以分为三类：第一种是类似中断的处理程序，对于需要处理的信号，进程可以指定处理函数，由该函数来处理。第二种方法是，忽略某个信号，对该信号不做任何处理，就象未发生过一样。第三种方法是，对该信号的处理保留系统的默认值。

**信号只是用来通知某进程发生了什么事件，并不给该进程传递任何数据。**

有关信号的系统调用：

* 系统调用signal用来设定某个信号的处理方法：

void (\*signal(int signum, void (\*handler)(int)))(int);

* 系统调用kill用来向进程发送一个信号：

int kill(pid\_t pid, int sig);

* 系统调用pause的作用是等待一个信号：

int pause(void);

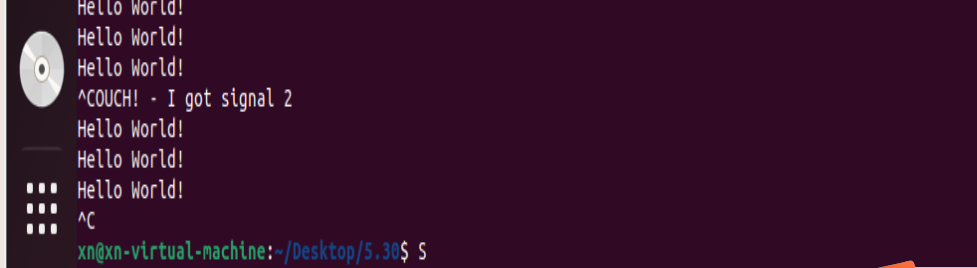
* 系统调用alarm的功能是设置一个定时器，当定时器计时到达时，将发出一个信号给进程：

unsigned int alarm(unsigned int seconds);

1. **从键盘发送信号**

|  |
| --- |
| #include <signal.h>  #include <stdio.h>  #include <unistd.h>  void ouch(int sig)  {  printf("OUCH! - I got signal %d\n", sig);  (void) signal(SIGINT, SIG\_DFL); //恢复SIGINT信号的处理动作  }  int main()  {  (void) signal(SIGINT, ouch); //设置SIGINT信号的处理动作为响应ouch函数  while(1) { //主函数每隔1秒输出字符串  printf("Hello World!\n");  sleep(1);  }  } |

思考：



1. 编译连接通过后，多次运行程序，查看输出结果。
2. 请说明系统调用signal( )的功能和使用方法。

signal()函数用于设置进程的信号处理方式。它的原型如下：

void (\*signal(int sig, void (\*func)(int)))(int);

第一个参数 sig：指定要处理的信号。

第二个参数 func：指定当信号发生时，用的信号处理函数。如果设置为SIG\_IGN，则忽略该信号；如果设置为SIG\_DFL，则恢复为默认的信号处理行为。

返回值：如果signal()成功，它返回之前信号处理函数的地址；如果失败，返回SIG\_ERR。

1. 体会如何从键盘发送中断信号给进程。

Ctrl+C。这会向前台进程组发送一个SIGINT信号

1. **闹钟**

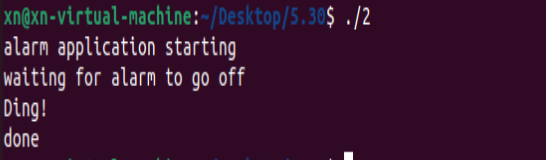
用fork( )创建两个子进程，子进程在等待5秒后用系统调用kill（）向父进程发送SIGALARM信号，父进程用系统调用signal( )捕捉SIGLARM信号。参考程序如下所示。

|  |
| --- |
| #include <signal.h>  #include <stdio.h>  #include <unistd.h>  #include <stdlib.h>  static int alarm\_fired = 0; //闹钟未设置  void ding(int sig) //模拟闹钟  {  alarm\_fired = 1; //设置闹钟  }  int main()  {  int pid;  printf("alarm application starting\n");  if((pid = fork( )) == 0)  { //子进程5秒后发送信号SIGALRM给父进程  sleep(5);  kill(getppid(), SIGALRM);  exit(0);  }  //父进程安排好捕捉到SIGALRM信号后执行ding函数  printf("waiting for alarm to go off\n");  (void) signal(SIGALRM, ding);  pause(); //挂起父进程，直到有一个信号出现  if (alarm\_fired)  printf("Ding!\n");  printf("done\n");  exit(0);  } |

使用信号的程序

思考：

1. 编译连接通过后，多次运行程序，查看输出结果。



1. 请说明系统调用kill（）的功能和使用方法。

kill(getppid(), SIGALRM);getpid()获得接受信号的进程ID, SIGALRM是将要发送的信号

如果成功，kill() 返回 0。

如果失败，返回 -1 并设置 errno 以指示错误。

1. 请说明系统调用signal( )的功能和使用方法。

(void) signal(SIGALRM, ding); SIGALRM是将要发送的信号，ding是信号发生时调用的函数

1. **进程使用信号通信**

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来的中断信号（即按^c键）；捕捉到中断信号后，父进程用系统调用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列信息后终止：

Child process1 is killed by parent!

Child process2 is killed by parent!

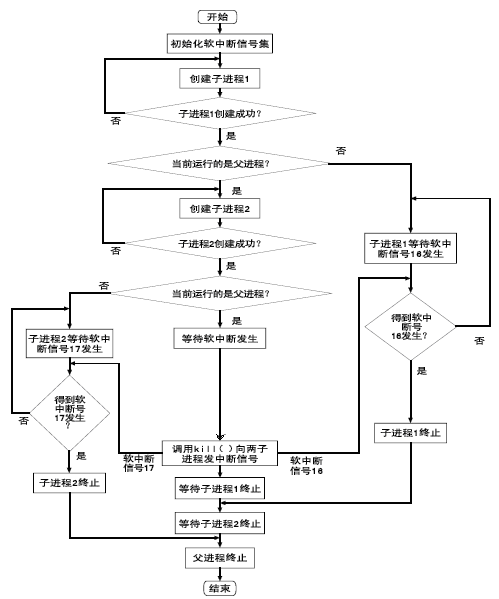
父进程等待两个子进程终止后，输出如下的信息后终止：

Parent process is killed!

参考程序和流程图分别如下所示。

|  |
| --- |
| #include<stdio.h>  #include<signal.h>  #include<unistd.h>  int wait\_mark;  #include <sys/wait.h>  void waiting( ),stop( );  void main( )  {  int p1,p2;  signal(SIGINT,stop);  while((p1=fork())==-1);  if(p1>0) //主进程的处理  {  while((p2=fork())==-1);  if(p2>0) //主进程的处理  {  wait\_mark=1;  waiting( ); //等待接收ctrl+c信号  kill(p1,16); //向p1发出信号16  kill(p2,17); //向p2发出信号17  wait(0); //同步  wait(0);  printf("parents is killed \n");  exit(0);  }  else //p2进程的处理  {  wait\_mark=1;  signal(17,stop);  waiting(); //等待信号17  printf("P2 is killed by parent \n");  exit(0);  }  }  else //p1进程的处理  {  wait\_mark=1;  signal(16,stop);  waiting( ); //等待信号16  printf("P1 is killed by parent \n");  exit(0);  }  }  void waiting( )  {  while(wait\_mark!=0);  }  void stop( )  {  wait\_mark=0;  } |

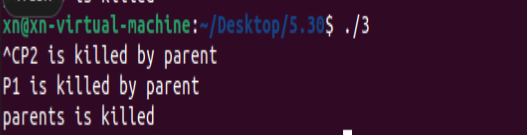
进程软中断通信



进程软中断通信程序的流程图

思考：

1. 编译连接通过后，多次运行程序，查看输出结果。



1. 请说明系统调用kill（）的功能和使用方法。

kill(p1,16); //向p1发出信号16

kill(p2,17); //向p2发出信号17

1. 父进程的处理中有两个wait（0）；他们有什么作用？

第一个 wait(0);：等待第一个子进程（p1 或 p2）结束。一旦有一个子进程结束，无论是哪一个，wait() 将返回该子进程的进程ID，并继续执行。

第二个 wait(0);：等待第二个子进程结束。在第一个子进程结束后，这个调用将等待剩余的子进程结束。

1. 每个进程退出时都使用了exit（0）；他们有什么作用？

在 exit(0); 中，参数 0 表示进程正常退出。退出状态 0 通常表示成功或没有错误。父进程通过 wait() 或 waitpid() 函数可以获取子进程的退出状态，并据此进行相应的处理。

**（三）编写程序**

用fork( )创建一个子进程，在子进程中再次调用fork( )创建孙子进程。调用系统调用signal( )，让父进程捕捉键盘上来的中断信号（即按^c键）；捕捉到中断信号后，父进程用系统调用kill( )向子进程发出信号，子进程捕捉到信号后，调用kill( )向孙子进程发出信号，孙子进程捕捉到信号后，输出下列信息后终止：

Grandson process1 is killed by son!

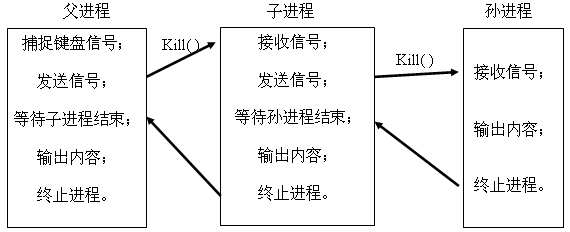
子进程等待孙子进程终止后，输出如下的信息后终止：

Child process is killed by parent!

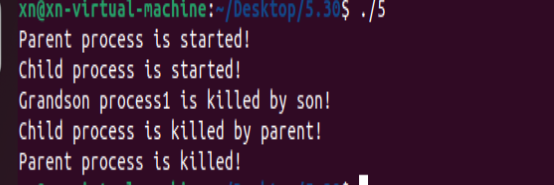
父进程等待子进程终止后，输出如下的信息后终止：

Parent process is killed!

提示：过程如下所示。



三代进程间通信流程图



#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <signal.h>

#include <sys/types.h>

#include <sys/wait.h>

void grandson\_process() {

printf("Grandson process1 is killed by son!\n");

exit(0);

}

void child\_process() {

printf("Child process is started!\n");

pid\_t pid = fork();

if (pid == 0) { // 子进程

grandson\_process();

} else {

wait(NULL);

printf("Child process is killed by parent!\n");

exit(0);

}

}

void parent\_process() {

printf("Parent process is started!\n");

signal(SIGINT, SIG\_IGN); // 忽略键盘中断信号

pid\_t pid = fork();

if (pid == 0) { // 子进程

child\_process();

} else {

wait(NULL);

printf("Parent process is killed!\n");

exit(0);

}

}

void handler(int signum) {

printf("Caught interrupt signal (^C)\n");

pid\_t pid = getpid();

kill(pid + 1, SIGINT); // 向子进程发送信号

}

int main() {

signal(SIGINT, handler); // 注册中断处理函数

parent\_process();

return 0;

}