**实验4 信号机制**

实验指导

1. Linux信号机制

**1.信号概念**

信号是进程在运行过程中，由自身产生或由进程外部发过来的消息（事件）。信号是硬件中断的软件模拟（软中断）。每个信号用一个整型常量宏表示，以SIG开头，比如SIGCHLD、SIGINT等，它们在系统头文件<signal.h>中定义，也可以通过在shell提示符下键入kill –l查看信号列表，信号说明如下。

|  |  |  |  |
| --- | --- | --- | --- |
| 值 | 名 字 | 说 明 | 默认处理 |
| 01 | SIGHUP | 挂起（hangup），中断关闭时产生这个信号 | 进程终止 |
| 02 | SIGINT | 中断，用户从键盘按Ctrl+c键 | 进程终止 |
| 03 | SIGQUIT | 退出，用户从键盘按Ctrl+\键，不同于SIGINT，这个会产生core dump文件。 | 进程终止并产生core文件 |
| 04 | SIGILL | 非法指令，这个信号用户不能去捕捉它。 | 进程终止 |
| 05 | SIGTRAP | 跟踪陷阱（trace trap），遇到调试断点。 | 进程终止并产生core文件 |
| 06 | SIGIOT | 执行I/O时产生硬件错误 | 进程终止并产生core文件 |
| 07 | SIGBUS | 总线错误 | 进程终止并产生core文件 |
| 08 | SIGFPE | 浮点运算溢出，算术运算异常，除0等。 | 进程终止并产生core文件 |
| 09 | SIGKILL | 终止进程（不可屏蔽），这个信号用户不能去捕捉它。 | 进程终止 |
| 10 | SIGUSR1 | 用户自定义信号1 | 进程终止 |
| 11 | SIGSEGV | 内存非法访问，默认打印出segment fault。 | 进程终止并产生core文件 |
| 12 | SIGUSR2 | 用户自定义信号2 | 进程终止并产生core文件 |
| 13 | SIGPIPE | 向某个非读管道中写入数据，即管道文件只有写进程，没有读进程。 | 进程终止 |
| 14 | SIGALRM | 定时报警信号，闹钟timer到期。 | 进程终止 |
| 15 | SIGTERM | 用kill系统调用发出的软件终止信号 | 进程终止 |
| 17 | SIGCHLD | 某个子进程终止 | 进程终止 |
| 19 | SIGSTOP | 进程暂停运行 | 进程终止 |
| 30 | SIGPWR | 电源故障 | 进程终止 |

**2.信号产生**

信号的生成来自内核，让内核生成信号的请求来自如下三个方面：

* 用户：用户能够通过输入Ctrl+c、Ctrl+\，或者是终端驱动程序分配给信号控制字符的其他任何键来请求内核产生信号。
* 内核：当进程执行出错时，内核会给进程发送一个信号，例如非法段存取（内存访问违规）、浮点数溢出等。
* 进程：进程可以通过系统调用kill给另一个进程发送信号，进程也可以通过信号和另外一个进程进行通信。

**3.信号处理**

进程接收到信号以后，可以有如下三种选择进行处理：

* **接收默认处理**：接收默认处理的进程通常会导致进程本身消亡。例如连接到终端的进程，用户按下Ctrl+c，将导致内核向进程发送一个SIGINT的信号，进程如果不对该信号做特殊的处理，系统将采用默认的方式处理该信号，即终止进程的执行。
* **忽略信号**：进程可以通过代码，显式地忽略某个信号的处理，例如：signal(SIGINT,SIGDEF)；但是某些信号是不能被忽略的：SIGKILL和SIGSTOP。即进程接收到这两个信号后，只能接受系统的默认处理，即终止进程。
* **捕捉信号并处理**：进程可以事先注册信号处理函数，当接收到信号时，由信号处理函数自动捕捉并且处理信号。

当进程处于核心态时，即使收到软中断也不予理睬；只有当它返回到用户态后，才处理软中断信号。对软中断信号的处理分三种情况进行：

* 如果进程收到的软中断是一个已决定要忽略的信号，进程不做任何处理便立即返回。
* 进程收到软中断后便退出；
* 执行用户设置的软中断处理程序。

**4.信号特点**

信号具有如下几个特点：

* 信号是软中断，既可作为进程间通信的一种机制，也可用于处理一些非正常情况。
* 信号是异步的，进程并不知道信号什么时候到达。
* 进程既可以处理信号，也可以发送信号给特定进程。
* 每个信号都有一个名字，这些名字都以SIG开头。

1. Linux信号处理函数
2. signal函数

用来通知内核如何处理某个特定信号（忽略、捕捉、默认处理）。预置进程对信号的处理方式，允许调用进程控制软中断信号。

1. 函数定义

头文件：

#include <signal.h>

系统调用格式：

signal(sig,handler)

参数定义：

int signum;

void (\*hander) (int)

参数说明：signum表示要捕捉的信号类型，handler是函数指针，表示要对该信号进行捕捉的函数，该参数的取值范围有三种：

* SIG\_IGN：表示当进程运行中接收到指定信号时，忽略它，不做任何处理。
* SIG\_DFL：表示交由系统缺省处理，使用系统自带的处理程序完成信号的处理。
* 函数名func：对接收到的信号处理执行一个函数。在该取值情况下，当进程接收到类型为sig的信号时，不管当前进程正在执行哪部分程序，都立即把控制权转给func函数。当func完成后，进程的控制权将返回原进程中断处。

返回值：如果调用成功，返回以前该信号的处理函数的地址，否则返回SIG\_ERR。

1. 使用举例

#include <stdio.h>

#include <signal.h>

#include <unistd.h>

#include <sys/wait.h>

#include<stdlib.h>

int main()

{

signal(SIGINT,SIG\_IGN); /\* 设置忽略键盘中断信号\*/

while(1)

{

printf("No answer to ‘Ctrl+c’ \n");

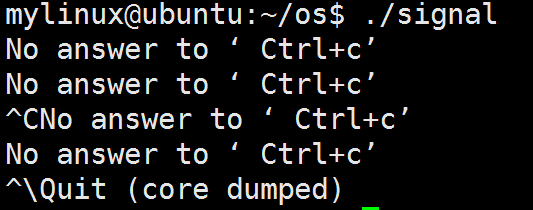
sleep(2);

}

return 0;

}

运行结果：



该程序运行后，将Ctrl+c产生的SIGINT信号被忽略了，所以键入Ctrl＋c将不能终止该进程。要终止该进程，可以向进程发送SIGQUIT信号，即组合键Ctrl+\。

1. kill()函数

用于发送指定类型的信号。

（1）函数定义

头文件：

#include <signal.h>

#include <sys/types.h>

系统调用格式：

int kill(pid,sig)

参数定义：

int pid,sig;

参数说明：pid是将要接收信号的进程ID；sig是要发送的软中断信号。

函数说明：将参数sig指定的信号传递给pid标记的进程。

参数pid的取值分析如下：

* pid>0：表示一个进程ID号，核心将信号sig发送给进程号ID为pid的进程。
* pid=0：表示同一进程组的进程，核心将信号sig发送给当前进程所属进程组的所有进程。
* pid<-1：表示进程组ID为pid绝对值的所有进程，核心将信号sig发送给进程组ID等于pid绝对值的所有进程。
* pid=-1：表示除发送者外，所有pid>1的进程，核心将信号sig发送给除1号进程和自身以外的所有进程。

返回值：如果发送成功，返回0，否则为-1。

（2）使用举例

#include <stdio.h>

#include <signal.h>

#include <unistd.h>

#include <sys/wait.h>

#include<stdlib.h>

void fun( )

{

printf("child received signal\n");

printf("child new exiting!\n");

exit(0);

}

int main( )

{

int pid;

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

if (pid>0)

{ printf("In parent process\n");

sleep(2);

kill(pid,SIGUSR1); /\*发送软中断信号给子进程\*/

wait(0); /\*等待子进程终止\*/

printf("In parent:Child has exited\n");

exit(0);

}

else

{

signal(SIGUSR1,fun); /\*设置收到信号的处理方式为执行函数fun\*/

while(1)

{

printf("In child process\n");

sleep(1);

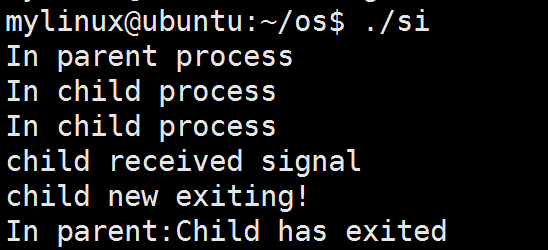
}

}

return 0;

}

运行结果：



该程序运行后，父进程创建子进程，然后给子进程发送SIGUSR1信号，子进程设置对SIGUSR1信号的处理方式是执行函数fun，没收到信号时子进程执行while循环，收到信号后执行fun，终止子进程。父进程等子进程终止后结束。