# 信号

## 概念

软中断信号（signal，又简称为信号）用来通知进程发生了异步事件，改变程序正常的执行流程。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。注意，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。

## 性质

信号是异步事件：

1、不可预见

2、信号有自己的名称和编号

3、信号和异常处理机制

## 发送

信号什么时候发出：

1、用户的动作 (如用户按键：Ctrl+c等)

2、硬件异常 （如：除数为0，无效内存引用等）

3、Kill系统调用或者kill命令

4、软件设置的条件 (如gdb程序给另外一个程序设置断点执行等)

## 常用信号

常用信号31个，每个信号都有自己的编号和对应的宏定义：

SIGTSTP 20 前台进程挂起(Ctrl+z)

SIGINT 2 前台进程中断(Ctrl+z)

SIGTERM 15 kill命令发送到默认终止信号

SIGSTOP 19 停止进程(不能被捕捉或忽略)

SIGKILL 9 杀死进程(不能被捕捉或忽略)

SIGCHLD 17 进程终止或停止时，将该信号发送给其父进程

SIGCONT 18 让当前处于停止状态的信号继续运行

SIGFPE 8 算术异常信号，如被除数为0等

SIGPIPE 13 若写到管道时读进程终止，则发生此信号，管道破裂

SIGUSR1 10 用户自定义信号

## signal

上述的信号系统提供了默认的处理方式，如果需要自定义信号处理函数，则可以调用signal：

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

参数：

sig： 信号编号

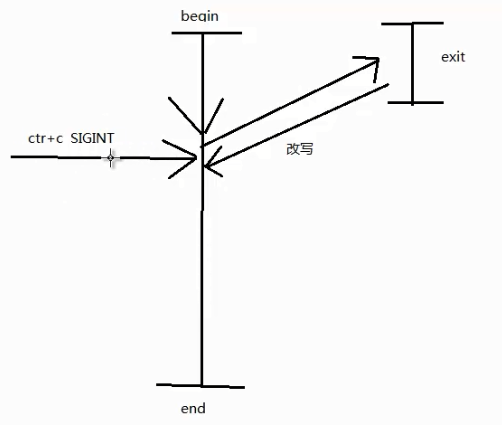
void (\* func )(int)：信号处理函数

返回值：类型是一个信号处理函数的函数指针

SIG\_ERR 错误返回结果 (void(\*)(int))(-1)

SIG\_IGN 忽略某个信号 (void(\*)(int))(0)

SIG\_DFL 默认处理某个信号 (void(\*)(int))(1)



实例：

#include <unistd.h>

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <errno.h>

#include <signal.h>

void sig\_handler(int signo)

{

printf("pid:%d,signum:%d-------------signal occured\n",getpid(),signo);

}

int main(void)

{

if(signal(SIGINT,sig\_handler) == SIG\_ERR)

{

fprintf(stderr,"sigal : %s\n",strerror(errno));

exit(1);

}

int i = 0;

while(1)

{

printf("i:%d\n",i++);

sleep(1);

}

return 0;

}

## 信号处理方式

### 程序启动

程序启动，所有信号的状态都是系统默认或忽略

一个进程原先要捕捉的信号，当其执行一个新程序后，就不在捕获

### 进程创建

当一个进程调用fork时，子进程继承父进程的信号处理方式。

实例：

#include <unistd.h>

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <errno.h>

#include <signal.h>

void sig\_handler(int signo)

{

printf("pid:%d catch child",getpid());

}

void out(int n)

{

int i = 0;

for(i;i < n;i++)

{

sleep(1);

printf("pid:%d,i:%d\n",getpid(),i);

}

}

int main(void)

{

if(signal(SIGTSTP,sig\_handler) == SIG\_ERR)

{

fprintf(stderr,"sigal : %s\n",strerror(errno));

exit(1);

}

pid\_t pid;

pid = fork();

if(pid < 0)

{

fprintf(stderr,"fork error:%s\n",strerror(errno));

exit(1);

}

else if(pid > 0) //parent

{

out(100);

}else //child

{

out(100);

}

return 0;

}

# 可靠性

信号的可靠性讨论：

连续发送多次信号，内核对信号的处理(由信号屏蔽字和未决字决定)发送同一个信号多次信号只处理前两次，后面的都被忽略，并且第二次发送的信号会等待第一次信号处理完再处理交错发送不同的信号马上处理新来的信号。内核中对信号的处理是由信号屏蔽字和未决字决定共同决定的，信号不排队等待。

代码片段

void sig\_int();

...

signal(SIGINT,sig\_int);

...

void sig\_int()

{

signal(SIGINT,sig\_int);

}

依赖于信号处理函数的代码，最好放到信号处理函数中,否则是不可靠的。

代码片段

void sig\_int();

...

signal(SIGINT,sig\_int);

...

while(sig\_int\_flag == 0)

pause();

...

void sig\_int(){

signal(SIGINT,sig\_int);

...

sig\_int\_flag = 1;

}

# 可重入性

中断的系统调用

进程调用“慢”系统调用时，如果发生了信号，内核会终止系统调用。

慢系统调用：

可能会永远阻塞的系统调用

从终端设备、管道或网络设备上的文件读取

向上述文件写入

某些设备上的文件打开

pause和wait系统调用

一些设备的ioctl操作

一些进程间通信函数

函数的可重入性讨论

对于进程调用“慢”的系统调用时，如果发生了信号中断，内核会终止系统调用，重新执行。

对于用户函数中静态存储区(或全局数据区)的变量接受到信号的中断处理后可导致函数不可重入。正常流程和中断处理流程会对变量的数值相互覆盖影响。

可重入函数

int double(int a){

return a\*2;

}

不可重入函数

void foo(){

static int intarray[28];

static int index;

if(index > 19)return;

intarray[index] = 9;

index++;

}

# 信号集

对信号集进行信号的清空、加入、删除等操作。

int sigemptyset(sigset\_t \*set);

将set集合置空

int sigfillset(sigset\_t \*set)；

将所有信号加入set集合

int sigaddset(sigset\_t \*set,int signo)

将signo信号加入到set集合

int sigdelset(sigset\_t \*set，int signo);

从set集合中移除signo信号

int sigismember(const sigset\_t \*set,int signo);

signo判断信号是否存在于set集合中

# 信号屏蔽

设置或获取信号集的信号屏蔽字：

int sigprocmask(int how, const sigset\_t \*restrict set,

sigset\_t\*restrito set);

返回值：成功返回0，出错返回-1

SIG\_BLOCK 追加屏蔽某个信号

SIG\_UNBLOCK 取消屏蔽某个信号

SIG\_SETMASK 设置新的屏蔽字信号集

如果屏蔽所有信号，可以进行如下设置：

sig\_set\_t set;

sigfillset(&set);

sigprocmask(SIG\_BLOCK, &set, NULL)；

等价于如下设置：

sig\_set\_t set;

sigfillset(&set);

sigprocmask(SIG\_SETMASK, &set, NULL);

如果清空所有信号屏蔽字集合，可以进行如下设置：

sig\_set\_t set;

sigfillset(&set);

sigprocmask(UN\_BLOCK, &set, NULL);

等价于如下设置：

sig\_set\_t set;

sigemptyset(&set);

sigprocmask(SIG\_SETMASK, &set, NULL);

获取当前信号屏蔽字信号集：

sig\_set\_t set;

sigemptyset(&set);

sigprocmask(SIG\_BLOCK, NULL, &set );

信号未决字

连续发送多个相同信号，则信号未决字标记保存，未处理的信号，即当同类型的信号已经被屏蔽的状态下才设置未决字，处理后将被取消未决状态。所以连续发送多个相同信号只能处理两次，后面的都会被丢掉忽略。

int sigpending(sigset\_t \*set)

返回值：成功返回0，出错返回-1

获取信号未决字集合

sig\_set\_t set;

sigemptyset(&set);

sigpending(&set);

# 定时器

alarm计时器

unsigned int alarm(unsigned int seconds);

函数说明 ：alarm()用来设置信号SIGALRM在经过参数seconds指定的秒数后传送给目前的进程。如果参数seconds 为0，则之前设置的闹钟会被取消，并将剩下的时间返回。

返回值： 返回之前闹钟的剩余秒数，如果之前未设闹钟则返回0。

useconds\_t ualarm(useconds\_t useconds, useconds\_t interval);

使用方法同alarm，但其一个参数为第一次计时器的延时，第二个参数为每次计时的间隔。采用的计时方式为循环计时。单位为us微秒。

kill和raise函数

kill和raise是用来发送信号的

kill把信号发送给进程或进程组

raise把信号发送给(进程)自身

函数原型如下:

#include<signal.h>

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

int raise(int signo);

返回值：成功则返回0, 出错则返回-1

从原型上可以看出, raise函数是可以通过kill实现的.

raise(signo); 等价于: kill(getpid(), signo);