# 信号

信号：信号是一种内容受限的异步通信机制

目的：通信

信号是异步的

信号本质上是一个int类型数字编号（事先定义好的）

信号的发出：

用户在终端按下按键

硬件异常后由操作系统内核发出信号（除零异常）

用户用kill命令向其他进程发出信号

某种软件条件满足后也会发出信号，如alarm闹钟时间到了会产生SIGALARM信号，向一个读端已经关闭的管道write时会产生 SIGPIPE信号

信号的处理

接受的进程：

忽略信号

捕获信号（信号绑定了一个处理函数）

默认处理（当前进程没有明显的管这个信号，默认：忽略或终止进程）

常见信号介绍

/usr/include/signal.h

/usr/include/ /bits/signum.h //信号定义处

1. SIGINT(名字) 2（数字编码） CTRL+C 时OS送个前台进程组的
2. SIGABRT 6 调用abort函数，进程异常终止
3. SIGPOLL SIGIO 8 指示一个异步IO事件
4. SIGKILL 9 杀死进程的终极方法
5. SIGSEGV 11 无效存储访问时OS发出该信号（段错误）
6. SIGPIPE 13 涉及管道和socket（管道可能已经关闭，不能再写或 者读）
7. SIGALARM 14 涉及ALARM函数的实现
8. SIGTERM 15 kill命令发送的OS默认终止信号
9. SIGCHLD 17 子进程终止或者停止时候OS向其父进程发此信号
10. SIGUSR1 10 用户自定义信号，作用和意义由应用自己定义
11. SIGUSR2 12

#include<stdio.h>

#include<signal.h>

#include<stdlib.h>

void func(int sig){

printf("func for signal :%d.\n",sig);

}

int main(){

signal(SIGINT,func);

//signal(SIGINT,SIG\_DFL); //指定默认处理，相当于没有指定，因为本身就是默认处理

tips SIG\_DFL在signum.h中有宏定义

//signal(SIGINT,SIG\_IGN); //忽略处理，不论怎么CTRL+C,都没有用，信号被忽略了。

printf("before while(1).\n")

while(1)

printf("after while(1).\n");

return 0;

}

signal 函数绑定一个捕获函数后信号发生后会自动执行绑定的捕获函数，并且把把信号编号作为传参传给捕获函数

signal的返回值在出错时为SIG\_ERR，绑定成功时返回旧的捕获函数

#include<stdio.h>

#include<signal.h>

#include<stdlib.h>

typedef void (\*singhandler\_t)(int);

int main(){

sighandler\_t ret=(sighandler\_t)-2;

ret=signal(SIGKILL,SIG\_IGN); //编译通过，执行会出错，不让忽略kill信号

if(SIG\_ERR=ret){

perror("signal:")

exit(-1);

}

printf("before while(1).\n")

while(1)

printf("after while(1).\n");

return 0;

}

signal函数优缺点

优点：简单好用，捕获信号常用

缺点：无法简单直接得知之前设置的信号处理方法，并且平台移植性不好

sigaction函数

移植性更好

用法关键是两个sigaction指针

sigaction可以一次得到设置新的捕获函数和获取旧的捕获函数，signal需要通过返回值获取旧的，如果不想修改只想获取旧的捕获函数就要两次，先获取再修改回去，而sigaction可以给新的捕获函数sigaction指针赋值null

alarm 和pause函数

#include<stdio.h>

#include<unistd.h>

#include<signal.h>

void func(int sig){

if(sig==SIGALRM){

printf({"alarm happened .\n");

}

}

int main(void){

unsigned int ret =-1;

struct sigaction act={0};

act.sa\_handler=func;

sigaction(SIGALRM,&act,NULL);

//signal(SIGALRM,func);

ret=alarm(3);

printf("ret=%d.\n",ret);

while(1){};

return 0;

}

int main(void){

unsigned int ret =-1;

struct sigaction act={0};

act.sa\_handler=func;

sigaction(SIGALRM,&act,NULL);

//signal(SIGALRM,func);

ret=alarm(5);

printf(" 1st ret=%d.\n",ret); //ret=0 第一次调用alarm返回值0；

sleep(3)

ret=alarm(5); //ret=2 上一次调用alarm剩余的时间，并重新设置闹钟

printf("2st ret=%d.\n",ret);

sleep(1)

ret=alarm(5); //ret=4

printf("3st ret=%d.\n",ret);

//while(1){}; //浪费CPU

pause(); //让当前进程交出CPU运行权，当alarm信号来临时pause结束

return 0;

}

pause 函数的作用就是让当前进程暂停运行，交出cpu给其他进程去执行，当前进程进入pause状态后当前进程会表现为"卡住、阻塞住"，要退出pause状态需要被信号唤醒。

使用alarm和pause模拟sleep

#include<stdio.h>

#include<unistd.h>

#include<signal.h>

void func(int sig){

}

void mysleep(unsigned int sleep);

int main(){

printf("before mysleep.\n");

mysleep(3);

printf("after mysleep.\n");

}

void mysleep(unsigned int seconds){

struct sigaction act={0};

act.sa\_handler=func;

sigaction(SIGALRM,&act,NULL);

alarm(seconds);

pause();

}