目录

[信号集 1](#_Toc200967716)

[信号屏蔽字 和 未决信号集 1](#_Toc200967717)

[信号集系统函数 2](#_Toc200967718)

[定义信号集 2](#_Toc200967719)

[清空信号集 2](#_Toc200967720)

[设置所有信号 2](#_Toc200967721)

[添加信号 2](#_Toc200967722)

[移除信号 2](#_Toc200967723)

[检查信号 2](#_Toc200967724)

[更改当前进程的信号屏蔽字 2](#_Toc200967725)

[读取未决信号集 3](#_Toc200967726)

[处理信号 3](#_Toc200967727)

[signal 3](#_Toc200967728)

[sigaction 3](#_Toc200967729)

# 信号集

信号集本质上是一组二进制位

## 信号屏蔽字 和 未决信号集

信号屏蔽字：计划要阻塞的信号(集合)，又称阻塞信号集

未决信号集：已经阻塞的信号的集合

如果将某个信号加入信号屏蔽字，那么当这个信号到来时，不会触发相应的信号处理器，而是被加入到未决信号集等待后续处理。

一旦该信号从信号屏蔽字中移除，操作系统会检查未决信号集，根据情况处理之前被阻塞的信号。

普通信号在未决信号集中最多只会记录一次，实时信号可以被多次记录

## 信号集系统函数

### 定义信号集

sigset\_t set;

### 清空信号集

将信号集二进制位全部置0

int sigemptyset(sigset\_t \*set)

### 设置所有信号

将信号集二进制位全部置1

int sigfillset(sigset\_t \*set)

### 添加信号

将信号signum添加到集合中，即对应位置1

int sigaddset(sigset\_t \*set, int signum):

### 移除信号

将信号signum从集合中移除，即对应位置0

int sigdelset(sigset\_t \*set, int signum)

### 检查信号

判断信号signum是否在集合set中。若在返回1，若不在返回0

int sigismember(const sigset\_t \*set，int signum):

### 更改当前进程的信号屏蔽字

int sigprocmask(int how, const sigset\_t \*set, sigset\_t \*oldset)

how：指定如何修改当前的信号屏蔽字。它可以取以下值之一：

SIG\_BLOCK：将 set 中指定的信号添加到当前的信号屏蔽字中

SIG\_UNBLOCK：从当前的信号屏蔽字中移除 set 中指定的信号

SIG\_SETMASK：用 set 中指定的信号集替换当前的信号屏蔽字(旧的信号屏蔽字将被完全替换为新的信号集)

set：指向包含新信号屏蔽字的 sigset\_t 类型的指针。如果 set 为 NULL，则不改变当前的信号屏蔽字。

oldset：指向一个 sigset\_t 类型的变量，用于保存之前的信号屏蔽字。如果不需要保存旧的信号屏蔽字，可以传入 NULL。

### 读取未决信号集

int sigpending(sigset\_t \*set)

set：传出的未决信号集

## 处理信号

### signal

该系统函数在不同Linux版本中有差异，推荐使用下面的sigaction

#include <signal.h>

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

sighandler\_t signal(int signum, sighandler\_t handler);

signum 是具体信号，handler是处理方式

handler可以是以下三种：

SIG\_IGN: 忽略该信号

SIG\_DFL: 以该信号默认处理方式进行处理

函数 : 执行指定的函数

如果处理方式是函数，会发生以下两件事之一：

1.执行函数前，把该信号处理方式设置为 SIG\_DFL ，然后执行函数

2.执行函数前，阻塞该信号，然后执行函数。函数执行完成后，解除该信号阻塞

这样处理目的是防止处理一个信号的过程中又接收到另一个相同的信号，从而导致无限递归、信号混乱等情况

### sigaction

#include <signal.h>

int sigaction(int signum, const struct sigaction \*\_Nullable restrict act, struct sigaction \*\_Nullable restrict oldact);

signum 是具体信号，act是处理方式(NULL则不改变处理方式)，oldact记录上一个处理方式(NULL则不记录上一个处理方式)

struct sigaction

{

void (\*sa\_handler)(int); // 传统信号处理函数

void (\*sa\_sigaction)(int, siginfo\_t \*, void \*); // 实时扩展处理函数(需要 sa\_flags 设为 SA\_SIGINFO)

sigset\_t sa\_mask; // 信号处理期间要屏蔽的其他信号

int sa\_flags; // 标志位，控制行为

void (\*sa\_restorer)(void); // 不提供给用户使用

};

处理信号会引起原来执行位置的中断，因此需要设置中断处理

例如:

多进程并发服务器需要回收已经结束的子进程，因此设置sigaction绑定SIGCHLD信号。当出现SIGCHLD信号时，系统发生中断去处理SIGCHLD信号，这时正在等待连接的accept()被SIGCHLD信号中断，产生EINTR错误。这意味着如果有子进程结束，accept()就会被中断，这时需要让accept()重新运行

判断错误类型是否为EINTR(即是否为中断)，如果是中断则手动恢复运行:

if(errno==EINTR) continue;