```
#include <signal.h>

int sigprocmask(int how, const sigset_t *set, sigset_t *oldset);
int sigemptyset(sigset_t *set);
int sigfillset(sigset_t *set);
int sigaddset(sigset_t *set, int signum);
int sigdelset(sigset_t *set, int signum);
                                    返回：如果成功则为 0，若出错则为 -1。
int sigismember(const sigset_t *set, int signum);
                        返回：若 signum 是 set 的成员则为 1，如果不是则为 0，若出错则为 -1。
```

sigprocmask 函数改变当前阻塞的信号集合(8.5.1 节中描述的 blocked 位向量)。具体的行为依赖于 how 的值：

SIG_BLOCK：把 set 中的信号添加到 blocked 中(blocked=blocked | set)。

SIG_UNBLOCK：从 blocked 中删除 set 中的信号(blocked=blocked &~set)。

SIG_SETMASK：block=set。

如果 oldset 非空，那么 blocked 位向量之前的值保存在 oldset 中。

使用下述函数对 set 信号集合进行操作：sigemptyset 初始化 set 为空集合。sigfillset 函数把每个信号都添加到 set 中。sigaddset 函数把 signum 添加到 set，sigdelset 从 set 中删除 signum，如果 signum 是 set 的成员，那么 sigismember 返回 1，否则返回 0。

例如，图 8-32 展示了如何用 sigprocmask 来临时阻塞接收 SIGINT 信号。

```
1       sigset_t mask, prev_mask;
2
3       Sigemptyset(&mask);
4       Sigaddset(&mask, SIGINT);
5
6       /* Block SIGINT and save previous blocked set */
7       Sigprocmask(SIG_BLOCK, &mask, &prev_mask);
8       . . .  // Code region that will not be interrupted by SIGINT
9       /* Restore previous blocked set, unblocking SIGINT */
10      Sigprocmask(SIG_SETMASK, &prev_mask, NULL);
11
```

图 8-32　临时阻塞接收一个信号

8.5.5　编写信号处理程序

信号处理是 Linux 系统编程最棘手的一个问题。处理程序有几个属性使得它们很难推理分析：1)处理程序与主程序并发运行，共享同样的全局变量，因此可能与主程序和其他处理程序互相干扰；2)如何以及何时接收信号的规则常常有违人的直觉；3)不同的系统有不同的信号处理语义。

在本节中，我们将讲述这些问题，介绍编写安全、正确和可移植的信号处理程序的一些基本规则。

1. 安全的信号处理

信号处理程序很麻烦是因为它们和主程序以及其他信号处理程序并发地运行，正如我们在图 8-31 中看到的那样。如果处理程序和主程序并发地访问同样的全局数据结构，那