头文件：

#include <poll.h>

函数原型：

int poll(struct pollfd fd[], nfds_t nfds, int timeout);

第一个参数:

一个struct pollfd结构数组;

struct pollfd结构如下：

```
struct pollfd{
    int fd;          //文件描述符
    short events;    //请求的事件
    short revents;   //返回的事件
};
```

events 的初始值为7fff,表示默认对所有事件都感兴趣。

revents的初始值为0，当执行poll函数后，值变为感谢趣的值。

第二个参数nfds：

要监视的描述符的个数。

第三个参数timeout：

是一个用毫秒表示的时间，是指定poll在返回前没有接收事件时应该等待的时间。

timeout = -1，poll不会超时，永远阻塞。

timeout = 0 ，poll立即返回，不阻塞。

timeout > 0 ，阻塞timeout所指定的毫秒时间长度之后返回。

返回值：

>0：数组fds中准备好读、写或出错状态的那些socket描述符的总数量；

=0：数组fds中没有任何socket描述符准备好读、写，或出错；

-1： poll函数调用失败，同时会自动设置全局变量errno；

案例：

```
#include <stdio.h>
#include <linux/input.h>
#include <fcntl.h>
#include <poll.h>

int main(int argc, char *argv)
{
 struct pollfd ev_fd[2];
 struct input_event ev;
 int fd = -1;
 int ret = -1;
 size_t rb = 0;

 fd = open("/dev/input/event5",O_RDONLY);//打开鼠标
 if (fd < 0)
 {
  perror("open error");
```

```
 return -1;
}

ev_fd[0].fd = fd;//将文件描述符添加进数组
ev_fd[0].events = POLLIN;//设置感兴趣事件

while(1)
{
 ret = poll(ev_fd, 1, -1);//设置永远等待，阻塞

 if (ret < 0)
  return -1;
 else if (ret == 0)
  return -1;
 else
 {
  if (ev_fd[0].revents & POLLIN)//如果感兴趣事件发生了
  {
   rb = read(fd, &ev, sizeof(struct input_event));
   if (rb < sizeof(struct input_event))
    return -1;
   printf("ev.type = %x, ev.code = %x, ev.value = %d\n",ev.type,ev.code,ev.value);
  }
 }
}

 return 0;
}
```