**一、回顾前面的select**

select优点：

目前几乎在所有的平台上支持，其良好跨平台支持也是它的一个优点

select缺点：

1. 每次调用 select()，都需要把 fd 集合从用户态拷贝到内核态
2. 单个进程能够监视的文件描述符的数量存在最大限制，在 Linux 上一般为 1024

**二、poll函数概述**

select() 和 poll() 系统调用的本质一样，但是 poll() 没有最大文件描述符数量的限制

poll()函数介绍

头文件：

#include <poll.h>

函数体：

int poll(struct pollfd \*fds, nfds\_t nfds, int timeout);

功能：

监视并等待多个文件描述符的属性变化

参数：

fds:指向一个结构体数组的第0个元素的指针，每个数组元素都是一个struct pollfd结构

struct pollfd{

int fd; //文件描述符

short events; //等待的事件

short revents; //实际发生的事件

};

fd：文件描述符。

events：

指定监测fd的事件（输入、输出、错误），每一个事件有多个取值，如下：

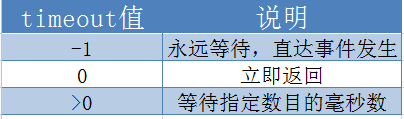


revents：

revents 域是文件描述符的操作结果事件，内核在调用返回时设置这个域。events 域中请求的任何事件都可能在 revents 域中返回.

nfds:用来指定第一个参数数组元素个数

timeout: 指定等待的毫秒数，无论 I/O 是否准备好，poll() 都会返回.



返回值：

成功时，poll() 返回结构体中 revents 域不为 0 的文件描述符个数；如果在超时前没有任何事件发生，poll()返回 0；  
失败时，poll() 返回 -1

**三、poll示例举例**

用poll实现udp同时收发

代码：

#include <string.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/select.h>

#include <sys/time.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <poll.h>

int main(int argc,char \*argv[])

{

    int udpfd = 0;

    struct pollfd fds[2];//监测文件描述结构体数组：2个

struct sockaddr\_in saddr;

    bzero(&saddr,sizeof(saddr));

    saddr.sin\_family = AF\_INET;

    saddr.sin\_port   = htons(8000);

    saddr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

    //创建套接字

    if( (udpfd = socket(AF\_INET,SOCK\_DGRAM, 0)) < 0)

    {

        perror("socket error");

        exit(-1);

    }

    //套接字端口绑字

    if(bind(udpfd, (struct sockaddr\*)&saddr, sizeof(saddr)) != 0)

    {

        perror("bind error");

        close(udpfd);

        exit(-1);

}

    printf("input: \"sayto 192.168.220.X\" to sendmsg to somebody\033[32m\n");

    fds[0].fd = 0;      //标准输入描述符

    fds[1].fd = udpfd;  //udp描述符

    fds[0].events = POLLIN; // 希望监测的事件，普通或优先级带数据可读

    fds[1].events = POLLIN; // 希望监测的事件，普通或优先级带数据可读

    while(1)

    {

        // 监视并等待多个文件（标准输入，udp套接字）描述符的属性变化（是否可读）

        // 没有属性变化，这个函数会阻塞，直到有变化才往下执行，这里没有设置超时

        int ret = poll(fds, 2, -1);

        write(1,"UdpQQ:",6);

        if(ret == -1){ // 出错

            perror("poll()");

        }

        else if(ret > 0){ // 准备就绪的文件描述符

            char buf[100] = {0};

            if( ( fds[0].revents & POLLIN ) ==  POLLIN ){ // 标准输入

                struct sockaddr\_in caddr;   // 新建一个目标地址的socket

                bzero(&caddr,sizeof(caddr));

                caddr.sin\_family  = AF\_INET;

                caddr.sin\_port    = htons(8000);

                fgets(buf, sizeof(buf), stdin);

                buf[strlen(buf) - 1] = '\0';

                if(strncmp(buf, "sayto", 5) == 0)

                {

                    char ipbuf[16] = "";

                    inet\_pton(AF\_INET, buf+6, &caddr.sin\_addr);//给addr套接字地址再赋值.

                    printf("\rsay to %s\n",inet\_ntop(AF\_INET,&caddr.sin\_addr,ipbuf,sizeof(ipbuf)));

                    continue;

                }

                else if(strcmp(buf, "exit")==0)

                {

                    close(udpfd);

                    exit(0);

                }

                // 发生udp数据

                sendto(udpfd, buf, strlen(buf), 0, (struct sockaddr\*)&caddr, sizeof(caddr));

            }

            else if( ( fds[1].revents & POLLIN ) ==  POLLIN ){ //udp套接字

                struct sockaddr\_in addr;

                char ipbuf[INET\_ADDRSTRLEN] = "";

                socklen\_t addrlen = sizeof(addr);

                bzero(&addr,sizeof(addr));

                recvfrom(udpfd, buf, 100, 0, (struct sockaddr\*)&addr, &addrlen);

                printf("\r\033[31m[%s]:\033[32m%s\n",inet\_ntop(AF\_INET,&addr.sin\_addr,ipbuf,sizeof(ipbuf)),buf);

            }

        }

        else if(0 == ret){ // 超时

            printf("time out\n");

        }

    }

    return 0;

}