select.c

```
/* ****************************************************************************
 *       Filename:  02select.c
 *    Description:
 *        Version:  1.0
 *        Created:  2019年02月24日 15时31分24秒
 *       Revision:  none
 *       Compiler:  gcc
 *         Author:  YOUR NAME (),
 *        Company:
 * ****************************************************************************/


#include <stdio.h>
#include <sys/select.h>
#include <sys/types.h>
#include <unistd.h>
#include "wrap.h"
#include <sys/time.h>
#define PORT 8888
int main(int argc, char *argv[])
{
    //创建套接字,绑定
    int lfd = tcp4bind(PORT,NULL);
    //监听
    Listen(lfd,128);
    int maxfd = lfd;//最大的文件描述符
    fd_set oldset,rset;
    FD_ZERO(&oldset);
    FD_ZERO(&rset);
    //将lfd添加到oldset集合中
    FD_SET(lfd,&oldset);
    while(1)
    {
        rset = oldset;//将oldset赋值给需要监听的集合rset

        int n = select(maxfd+1,&rset,NULL,NULL,NULL);
        if(n < 0)
        {
            perror("");
            break;
        }
        else if(n == 0)
        {
            continue;//如果没有变化,重新监听
        }
        else//监听到了文件描述符的变化
        {
            //lfd变化 代表有新的连接到来
            if( FD_ISSET(lfd,&rset))
            {
                struct sockaddr_in cliaddr;
```

```
                socklen_t len =sizeof(cliaddr);
                char ip[16]="";
                //提取新的连接
                int cfd = Accept(lfd,(struct sockaddr*)&cliaddr,&len);
                printf("new client ip=%s
port=%d\n",inet_ntop(AF_INET,&cliaddr.sin_addr.s_addr,ip,16),
                        ntohs(cliaddr.sin_port));
                //将cfd添加至oldset集合中,以下次监听
                FD_SET(cfd,&oldset);
                //更新maxfd
                if(cfd > maxfd)
                    maxfd = cfd;
                //如果只有lfd变化,continue
                if(--n == 0)
                    continue;

            }


            //cfd  遍历lfd之后的文件描述符是否在rset集合中,如果在则cfd变化
            for(int i = lfd+1;i<=maxfd;i++)
            {
                //如果i文件描述符在rset集合中
                if(FD_ISSET(i,&rset))
                {
                    char buf[1500]="";
                    int ret = Read(i,buf,sizeof(buf));
                    if(ret < 0)//出错,将cfd关闭,从oldset中删除cfd
                    {
                        perror("");
                        close(i);
                        FD_CLR(i,&oldset);
                        continue;
                    }
                    else if(ret == 0)
                    {
                        printf("client close\n");
                        close(i);
                        FD_CLR(i,&oldset);

                    }
                    else
                    {
                        printf("%s\n",buf);
                        Write(i,buf,ret);

                    }

                }


            }


        }
```

```
    }


    return 0;
}
```

wrap.c

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <strings.h>

void perr_exit(const char *s)
{
    perror(s);
    exit(-1);
}

int Accept(int fd, struct sockaddr *sa, socklen_t *salenptr)
{
    int n;

again:
    if ((n = accept(fd, sa, salenptr)) < 0) {
        if ((errno == ECONNABORTED) || (errno == EINTR))//如果是被信号中断和软件层次中断,不能退出
            goto again;
        else
            perr_exit("accept error");
    }
    return n;
}

int Bind(int fd, const struct sockaddr *sa, socklen_t salen)
{
    int n;

    if ((n = bind(fd, sa, salen)) < 0)
        perr_exit("bind error");
```

```
    return n;
}

int Connect(int fd, const struct sockaddr *sa, socklen_t salen)
{
    int n;

    if ((n = connect(fd, sa, salen)) < 0)
        perr_exit("connect error");

    return n;
}

int Listen(int fd, int backlog)
{
    int n;

    if ((n = listen(fd, backlog)) < 0)
        perr_exit("listen error");

    return n;
}

int Socket(int family, int type, int protocol)
{
    int n;

    if ((n = socket(family, type, protocol)) < 0)
        perr_exit("socket error");

    return n;
}

ssize_t Read(int fd, void *ptr, size_t nbytes)
{
    ssize_t n;

again:
    if ( (n = read(fd, ptr, nbytes)) == -1) {
        if (errno == EINTR)//如果是被信号中断,不应该退出
            goto again;
        else
            return -1;
    }
    return n;
}

ssize_t Write(int fd, const void *ptr, size_t nbytes)
{
    ssize_t n;

again:
    if ( (n = write(fd, ptr, nbytes)) == -1) {
        if (errno == EINTR)
            goto again;
```

```
        else
            return -1;
    }
    return n;
}

int Close(int fd)
{
    int n;
    if ((n = close(fd)) == -1)
        perr_exit("close error");

    return n;
}

/*参三: 应该读取固定的字节数数据*/
ssize_t Readn(int fd, void *vptr, size_t n)
{
    size_t  nleft;              //usigned int 剩余未读取的字节数
    ssize_t nread;              //int 实际读到的字节数
    char   *ptr;

    ptr = vptr;
    nleft = n;

    while (nleft > 0) {
        if ((nread = read(fd, ptr, nleft)) < 0) {
            if (errno == EINTR)
                nread = 0;
            else
                return -1;
        } else if (nread == 0)
            break;

        nleft -= nread;
        ptr += nread;
    }
    return n - nleft;
}
/*:固定的字节数数据*/
ssize_t Writen(int fd, const void *vptr, size_t n)
{
    size_t nleft;
    ssize_t nwritten;
    const char *ptr;

    ptr = vptr;
    nleft = n;
    while (nleft > 0) {
        if ( (nwritten = write(fd, ptr, nleft)) <= 0) {
            if (nwritten < 0 && errno == EINTR)
                nwritten = 0;
            else
                return -1;
        }
```

```
        nleft -= nwritten;
        ptr += nwritten;
    }
    return n;
}

static ssize_t my_read(int fd, char *ptr)
{
    static int read_cnt;
    static char *read_ptr;
    static char read_buf[100];

    if (read_cnt <= 0) {
again:
        if ( (read_cnt = read(fd, read_buf, sizeof(read_buf))) < 0) {
            if (errno == EINTR)
                goto again;
            return -1;
        } else if (read_cnt == 0)
            return 0;
        read_ptr = read_buf;
    }
    read_cnt--;
    *ptr = *read_ptr++;

    return 1;
}

ssize_t Readline(int fd, void *vptr, size_t maxlen)
{
    ssize_t n, rc;
    char    c, *ptr;

    ptr = vptr;
    for (n = 1; n < maxlen; n++) {
        if ( (rc = my_read(fd, &c)) == 1) {
            *ptr++ = c;
            if (c  == '\n')
                break;
        } else if (rc == 0) {
            *ptr = 0;
            return n - 1;
        } else
            return -1;
    }
    *ptr  = 0;

    return n;
}

int tcp4bind(short port,const char *IP)
{
    struct sockaddr_in serv_addr;
    int lfd = Socket(AF_INET,SOCK_STREAM,0);
```

```
    bzero(&serv_addr,sizeof(serv_addr));
    if(IP == NULL){
        //如果这样使用 0.0.0.0,任意ip将可以连接
        serv_addr.sin_addr.s_addr = INADDR_ANY;
    }else{
        if(inet_pton(AF_INET,IP,&serv_addr.sin_addr.s_addr) <= 0){
            perror(IP);//转换失败
            exit(1);
        }
    }
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port   = htons(port);
    int opt = 1;
    setsockopt(lfd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt));

    Bind(lfd,(struct sockaddr *)&serv_addr,sizeof(serv_addr));
    return lfd;
}
```