# Linux select TCP并发服务器与客户端编程 - 新一

*新一关注 - 15 粉丝 - 60 +加关注*

介绍：运行在ubuntu linux系统，需要先打开一个终端运行服务端代码，这时，可以打开多个终端同时运行多个客户端代码（注意客户端数目要小于MAX_FD）;在客户端输入数据后回车，可以看见服务器收到数据，并回复客户端确认信息，客户端输入：exit，按回车，该客户端关闭，在服务器端显示退出信息;所有客户端关闭后，服务器不会自动关闭，需要按ctrl+c强制关闭。

服务器端代码：

```
#include <stdio.h>

#include <unistd.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/select.h>

#define SERV_PORT 8888
#define SERV_IP "127.0.0.1"        //本地回环接口
#define LIST 20                    //服务器最大接受连接
#define MAX_FD 10                  //FD_SET支持描述符数量
int main(void)
{
        int sockfd;
        int err;
        int i;
        int connfd;

        int fd_all[MAX_FD]; //保存所有描述符，用于select调用后，判断哪个可读

//下面两个备份原因是select调用后，会发生变化，再次调用select前，需要重新赋值
```

```
        fd_set fd_read;                 //FD_SET数据备份

        fd_set fd_select;               //用于select

        struct timeval timeout;          //超时时间备份
        struct timeval timeout_select;  //用于select

        struct sockaddr_in serv_addr;   //服务器地址
        struct sockaddr_in cli_addr;    //客户端地址
        socklen_t serv_len;
        socklen_t cli_len;

       //超时时间设置
        timeout.tv_sec = 10;
        timeout.tv_usec = 0;

        sockfd = socket(AF_INET, SOCK_STREAM, 0);
        if(sockfd < 0)
        {
                perror("fail to socket");
                exit(1);

}
        memset(&serv_addr, 0, sizeof(serv_addr));
        memset(&cli_addr, 0, sizeof(cli_addr));

        serv_addr.sin_family = AF_INET;
        serv_addr.sin_port = htons(SERV_PORT);
        serv_addr.sin_addr.s_addr = inet_addr(SERV_IP);

        serv_len = sizeof(serv_addr);

        err = bind(sockfd, (struct sockaddr *)&serv_addr, serv_len);
        if(err < 0)
        {
                perror("fail to bind");
                exit(1);
        }

        err = listen(sockfd, LIST);

        if(err < 0)
        {
                perror("fail to listen");
                exit(1);
        }
```

```
        //初始化fd_all数组
        memset(&fd_all, -1, sizeof(fd_all));

        fd_all[0] = sockfd;   //第一个为监听套接字
        FD_ZERO(&fd_read);
        FD_SET(sockfd, &fd_read);  //将监听套接字加入fd_set

        char buf[1024];  //读写缓冲区
        int num;
        int maxfd;
        maxfd = fd_all[0];  //监听的最大套接字

        while(1)
        {

                //每次都需要重新赋值
                fd_select = fd_read;
                timeout_select = timeout;


//              err = select(maxfd+1, &fd_select, NULL, NULL, NULL);
                err = select(maxfd+1, &fd_select, NULL, NULL, (struct
timeval *)&timeout_select);
                if(err < 0)
                {
                        perror("fail to select");
                        exit(1);
                }
                if(err == 0)
                        printf("timeout\n");

//检测监听套接字是否可读

                if(FD_ISSET(sockfd, &fd_select))
                {
                        //可读，证明有新客户端连接服务器
                        cli_len = sizeof(cli_addr);
                        connfd = accept(sockfd, (struct sockaddr
*)&cli_addr, &cli_len);
                        if(connfd < 0)
                        {
                                perror("fail to accept");
                                exit(1);
                        }
```

```
//将新连接套接字加入fd_all及fd_read
                for(i=0; i<MAX_FD; i++)
                {
                        if(fd_all[i] != -1)
                        {
                                continue;
                        }

                        else
                        {
                                fd_all[i] = connfd;
                                printf("client fd_all[%d] join\n", i);
                                break;
                        }
                }
                FD_SET(connfd, &fd_read);

                if(maxfd < connfd)
                {
                        maxfd = connfd;  //更新maxfd
                }

        }
        //从1开始查看连接套接字是否可读，因为上面已经处理过
0（sockfd）

        for(i=1; i<maxfd; i++)
        {
                if(FD_ISSET(fd_all[i], &fd_select))
                {
                        printf("fd_all[%d] is ok\n", i);

                        num = read(fd_all[i], buf, 1024);
                        if(num > 0)
                        {
                                if(strncmp("exit", buf, 4) == 0)
                                {

//客户端退出，关闭套接字，并从监听集合清除
                                        printf("client:fd_all[%d] exit\n", i);
                                        FD_CLR(fd_all[i], &fd_read);
                                        close(fd_all[i]);
                                        fd_all[i] = -1;

                                        continue;
```

```
                                }

//收到 客户端数据并打印

                                buf[num] = '\0';
                                printf("receive buf from client fd_all[%d]
is: %s\n", i, buf);
                        }

//回复客户端
                        num = write(fd_all[i], "ok", sizeof("ok"));
                        if(num < 0)
                        {
                                perror("fail to write ");
                                exit(1);
                        }
                        else
                        {
                                printf("send reply\n");
                        }
                }

                else
                {
                        //printf("no data\n");
                }

        }

    }

    return 0;
}
```

客户端代码:

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
```

```
#include <arpa/inet.h>

#define SERV_PORT 8888
#define SERV_IP "127.0.0.1"
int main(void)
{
	int sockfd;
	int err;
	int connfd;
	struct sockaddr_in serv_addr;
	struct sockaddr_in cli_addr;
	socklen_t serv_len;
	socklen_t cli_len;

	sockfd = socket(AF_INET, SOCK_STREAM, 0);
	if(sockfd < 0)
	{
		perror("fail to socket");
		exit(1);
	}
	memset(&serv_addr, 0, sizeof(serv_addr));
	memset(&cli_addr, 0, sizeof(cli_addr));

	serv_addr.sin_family = AF_INET;
	serv_addr.sin_port = htons(SERV_PORT);
	serv_addr.sin_addr.s_addr = inet_addr(SERV_IP);

	serv_len = sizeof(serv_addr);

//客户端不需要绑定，直接连接即可

	err = connect(sockfd, (struct sockaddr *)&serv_addr, serv_len);
	if(err < 0)

{
		perror("fail to bind");
		exit(1);
	}

	char buf[1024];
	int num;
	while(1)
	{
		sleep(2);
		num = read(STDIN_FILENO, buf, 1024);
```

```
            if(num > 0)
            {

//exit代表退出
                    if(strncmp("exit", buf, 4) == 0)
                    {
                        write(sockfd, buf, num);
                        break;
                    }

 }
                    write(sockfd, buf, num);
            }
            num = read(sockfd, buf, 1024);
            if(num > 0)
            {
                    buf[num] = '\0';
                    printf("server reply: %s\n", buf);
            }
            else
                    printf("error to read\n");

    }

    close(sockfd);


    return 0;
}
```