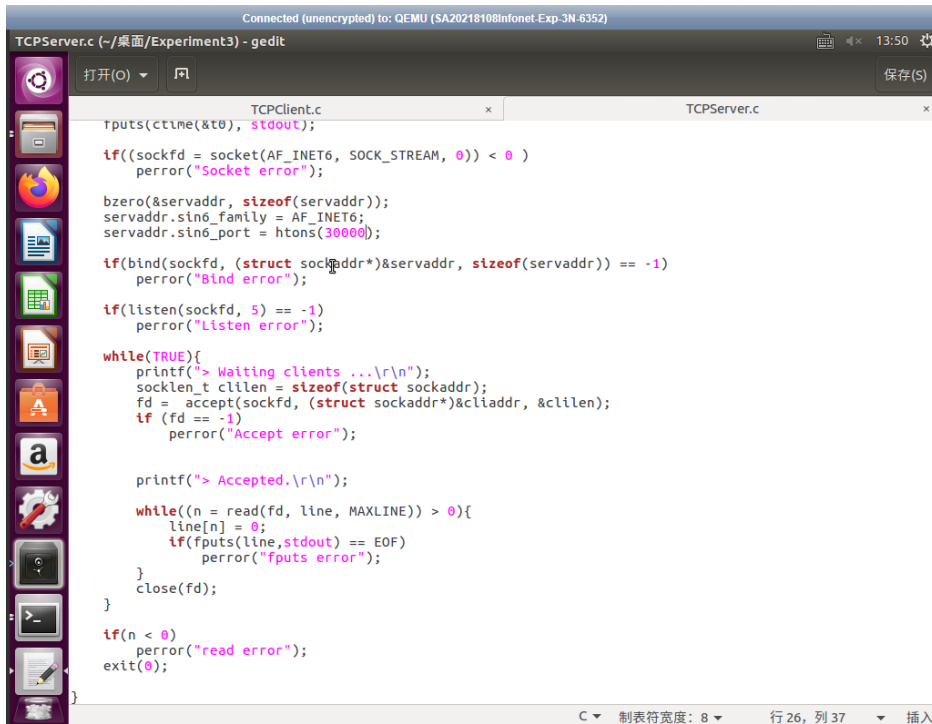


## 实验三 IPv6 网络编程实验——入门

### 一、实验结果

#### 1. 服务器和客户端代码截图：

服务器：



```
Connected (unencrypted) to: QEMU (SA20218108Infonet-Exp-3N-6352)
TCPClient.c (~/.桌面/Experiment3) - gedit
TCPClient.c x TCPServer.c x

rputs(ctime(&t0), stdout);

if((sockfd = socket(AF_INET6, SOCK_STREAM, 0)) < 0 )
    perror("Socket error");

bzero(&servaddr, sizeof(servaddr));
servaddr.sin6_family = AF_INET6;
servaddr.sin6_port = htons(30000);

if(bind(sockfd, (struct sockaddr*)&servaddr, sizeof(servaddr)) == -1)
    perror("Bind error");

if(listen(sockfd, 5) == -1)
    perror("Listen error");

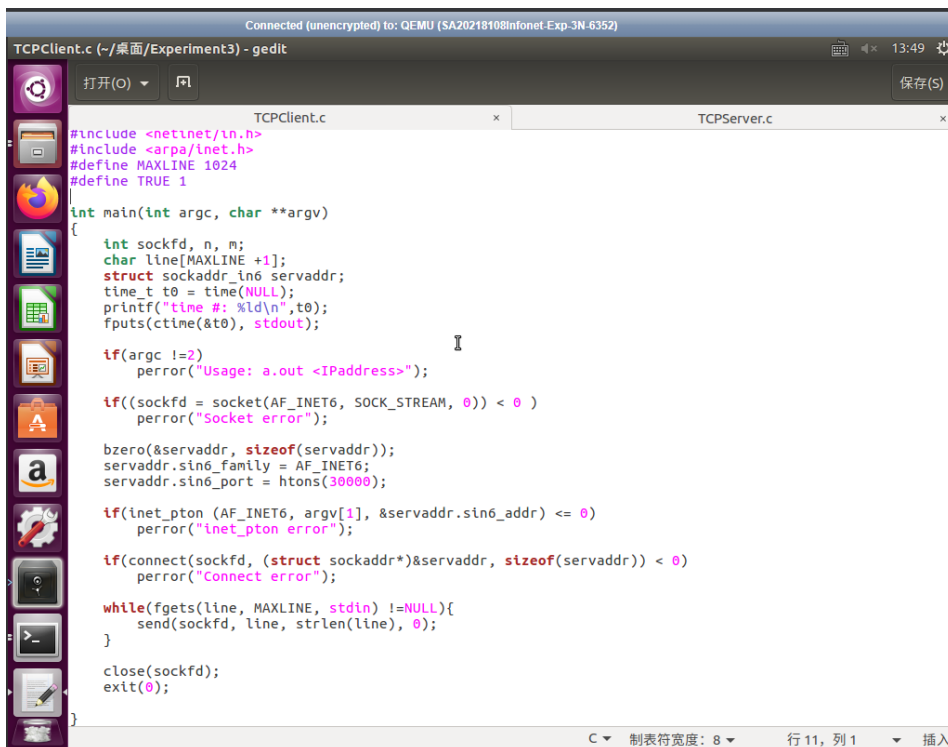
while(TRUE){
    printf("> Waiting clients ...\r\n");
    socklen_t clien = sizeof(struct sockaddr);
    fd = accept(sockfd, (struct sockaddr*)&cliaddr, &clielen);
    if (fd == -1)
        perror("Accept error");

    printf("> Accepted.\r\n");

    while((n = read(fd, line, MAXLINE)) > 0){
        line[n] = 0;
        if(fputs(line, stdout) == EOF)
            perror("fputs error");
    }
    close(fd);
}

if(n < 0)
    perror("read error");
exit(0);
}
```

客户端：



```
Connected (unencrypted) to: QEMU (SA20218108Infonet-Exp-3N-6352)
TCPClient.c (~/.桌面/Experiment3) - gedit
TCPClient.c x TCPServer.c x

#include <netinet/in.h>
#include <arpa/inet.h>
#define MAXLINE 1024
#define TRUE 1

int main(int argc, char **argv)
{
    int sockfd, n, m;
    char line[MAXLINE + 1];
    struct sockaddr_in6 servaddr;
    time_t t0 = time(NULL);
    printf("time #: %ld\n", t0);
    fputs(ctime(&t0), stdout);

    if(argc != 2)
        perror("Usage: a.out <IPaddress>");

    if((sockfd = socket(AF_INET6, SOCK_STREAM, 0)) < 0 )
        perror("Socket error");

    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin6_family = AF_INET6;
    servaddr.sin6_port = htons(30000);

    if(inet_pton (AF_INET6, argv[1], &servaddr.sin6_addr) <= 0)
        perror("inet_pton error");

    if(connect(sockfd, (struct sockaddr*)&servaddr, sizeof(servaddr)) < 0)
        perror("Connect error");

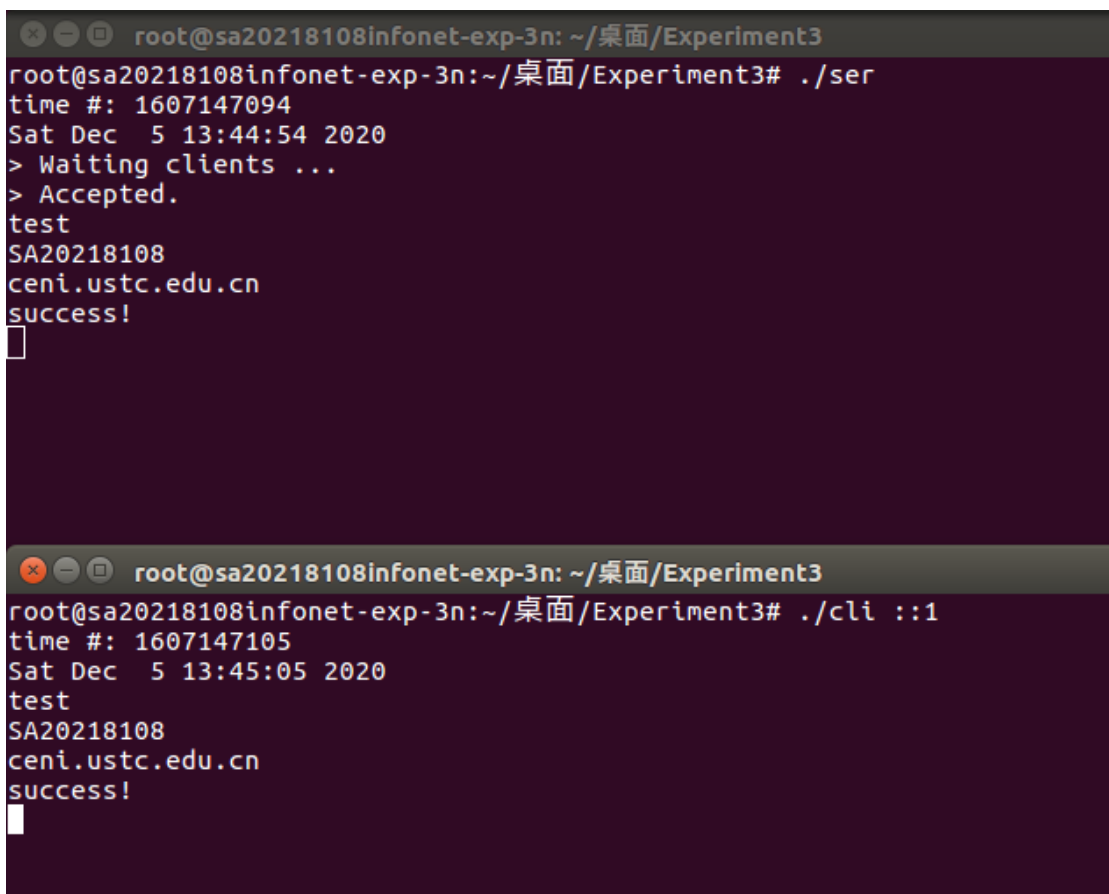
    while(fgets(line, MAXLINE, stdin) != NULL){
        send(sockfd, line, strlen(line), 0);
    }

    close(sockfd);
    exit(0);
}
```

2. 编译结果截图:



3. 运行结果截图:



4. 代码解释:

客户端代码:

```

int main(int argc, char **argv){

    int sockfd, n, m;
    char line[MAXLINE + 1];
    struct sockaddr_in6 servaddr;

    time_t t0 = time(NULL);
    printf("time #: %ld\n", t0); //输出机器时间
    fputs(ctime(&t0), stdout); //将机器时间转换为格式化时间

    if(argc != 2) //对输入参数的判断，如未输入IPaddress则报错
        perror("usage: a.out <IPaddress>");

    if((sockfd = socket(AF_INET6, SOCK_STREAM, 0)) < 0)
        /**
         * socket()函数将在通信域中创建一个未绑定的套接字，并返回一个文件描述符，
         * 该文件描述符可在以后对套接字进行操作的函数调用中使用。
         * domain: 指定要在其中创建套接字的通信域
         *      AF_INET6, IPv6 和 IPv4的Internet系列
         * type: 指定要创建的套接字的类型
         *      SOCK_STREAM类型，提供顺序的，可靠的，双向的连接模式字节流，
         *      并可以提供带外数据流的传输机制
         * protocol: 指定要与套接字一起使用的特定协议
         *      指定为0，表示函数使用于请求的套接字类型为默认类型
         */
        perror("socket error");

    bzero(&servaddr, sizeof(servaddr)); //置零操作
    servaddr.sin6_family = AF_INET6; //通信域设置为IPv6和IPv4
    servaddr.sin6_port = htons(20000); //设置套接字的通信端口

    if(inet_pton(AF_INET6, argv[1], &servaddr.sin6_addr) <= 0)
        //判断输入参数是否是一个ip地址
        perror("inet_pton error");

    if(connect(sockfd, (struct sockaddr*)&servaddr, sizeof(servaddr)) < 0)
        //判断是否连接成功
        perror("connect error");

    while(fgets(line, MAXLINE, stdin) != NULL){ //输入通信信息
        send(sockfd, line, strlen(line), 0); //发送通信信息
    }

    close(sockfd);
    exit(0);
}

```

服务器端代码:

```

time_t t0 = time(NULL);
printf("time #: %ld\n", t0); //输出机器时间
fputs(ctime(&t0), stdout); //将机器时间转换为格式化时间

if((sockfd = socket(AF_INET6, SOCK_STREAM, 0)) < 0)
    //判断是否成功创建套接字
    perror("socket error");

bzero(&servaddr, sizeof(servaddr)); //置零操作
servaddr.sin6_family = AF_INET6; //设置IPaddress类型
servaddr.sin6_port = htons(20000); //设置端口号
servaddr.sin6_addr = in6addr_any; //地址使用通配符

if(bind(sockfd, (struct sockaddr*)&servaddr, sizeof(servaddr)) == -1)
    //为套接字分配地址，并判断绑定函数是否成功
    perror("bind error");

if(listen(sockfd, 5) == -1)
    /*
     * 判断监听是否成功
     * int s:描述符，用于表示绑定的未连接的套接字
     * sockfd, 前文中绑定的套接字
     * int backlog: 挂起的连接队列可能增长的最大长度
     * 5, 最多同时允许5个客户端同服务器连接
     */
    perror("listen error");

while(TRUE) {
    //服务器开启
    printf("> Waiting clients ...\r\n");
    socklen_t clilen = sizeof(struct sockaddr);
    fd = accept(sockfd, (struct sockaddr*)&cliaddr, &clilen);
    if(fd == -1){
        perror("accept error");
    }

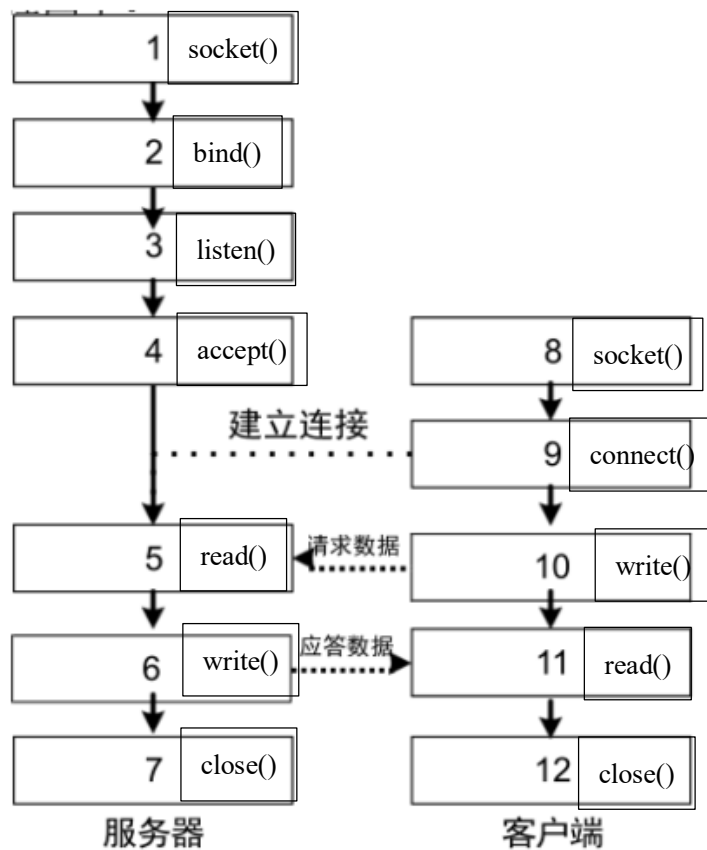
    printf("> Accepted.\r\n");

    while((n = read(fd, line, MAXLINE)) > 0){ //读取客户端发送的文件
        line[n] = 0;
        if(fputs(line, stdout) == EOF) //按行读取文件信息，并输出，并判断文件是否结束
            perror("fputs error");
    }
    close(fd);
}
if(n < 0) perror("read error");
exit(0);

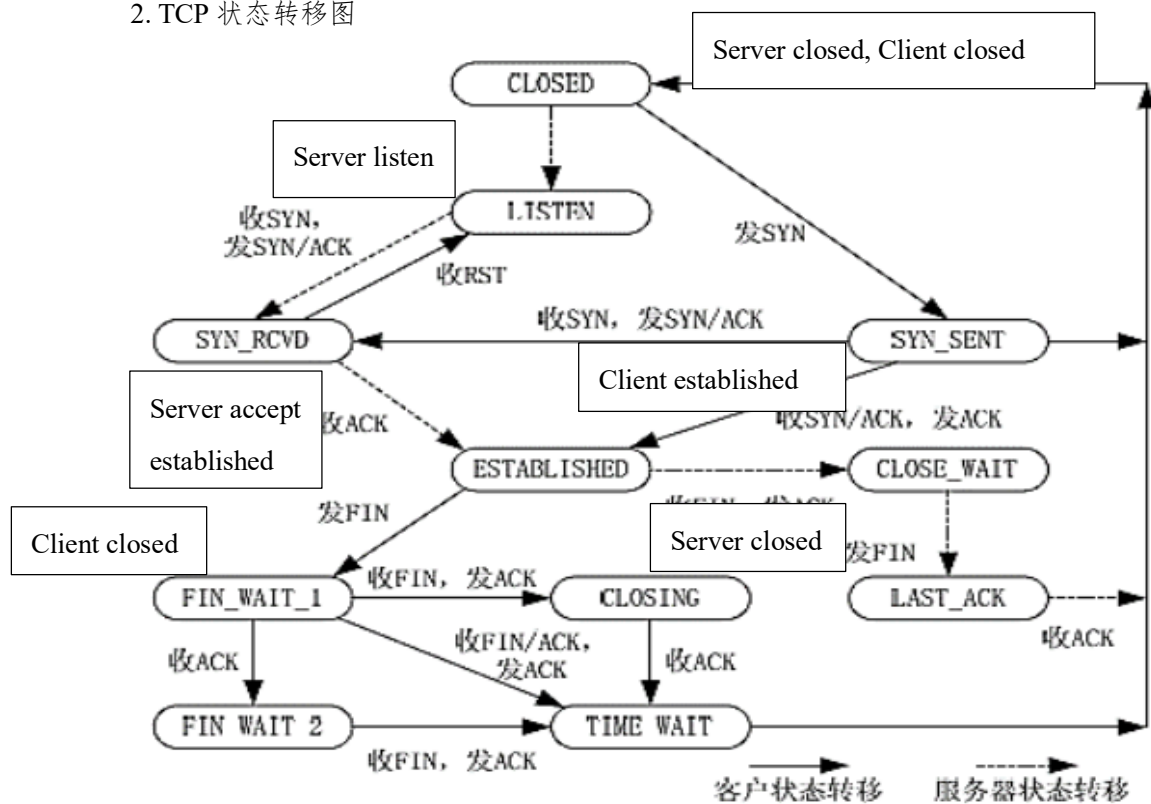
```

## 二、回答问题

### 1. C/S 模型流程图



2. TCP 状态转移图



**SYN Flood 攻击：**通过向服务器所在端口非法发送大量的连接报文抢占服务器的半开连接队列，使得其他合法用户无法访问服务器。

### 三、收获

学习了 **Linux** 下的套接字编程,成功手动的实现了基于 **C/S** 架构的客户端和服务器的连接,并成功进行信息传输;仔细对源代码进行了分析,对客户端和服务端连接的 **TCP** 三次握手协议有了更进一步的细节认识。