

**实验报告**

|  |  |
| --- | --- |
| **课程名称：** | 研究生公共实验课 |
| **姓 名：** | 汤明空 |
| **学 号：** | 2017110980 |
| **班 级：** | 2017级18班计算机学硕 |

合肥工业大学

2018.11.16

**编写一个包含服务器与客户端的小程序反应Linux下的TCP通信过程**

**1：安装ubuntu16.04**

去Ubuntu的官网https://www.ubuntu.com/download/alternative-downloads下载ubuntu16.04LTS版本，然后使用[UltraISO](https://cn.ultraiso.net/jiaocheng/jian-rong-xing.html)软碟通工具将ubuntu的iso镜像文件写入U盘中。将电脑从U盘启动，即可进入ubuntu系统的安装。选择中文版进行安装，分区划分选择自定义，由于电脑是UEFI系统所以先分配100M空间给UEFI系统分区并将此分区的物理地址挂载在系统启动项上；然后分配16G空间给SWAP交换空间，此为逻辑分区；分配100G给/系统根分区，此为主分区。剩下的/home、/usr等没有具体分配，都包含在/系统根分区里。点击安装系统便可完成ubuntu16.04LTS系统的安装。

**2：使用vim编写client.c和server.c**

进入ubuntu系统后，按Ctrl+Alt+t可打开系统命令行。在命令行里键入vi client.c进入vi环境，此时按i即可编写client.c文件。编写完后按esc退出插入模式，键入：wq保存并退出完成对文件的编写。对server.c文件的编写同理。如图2.1所示

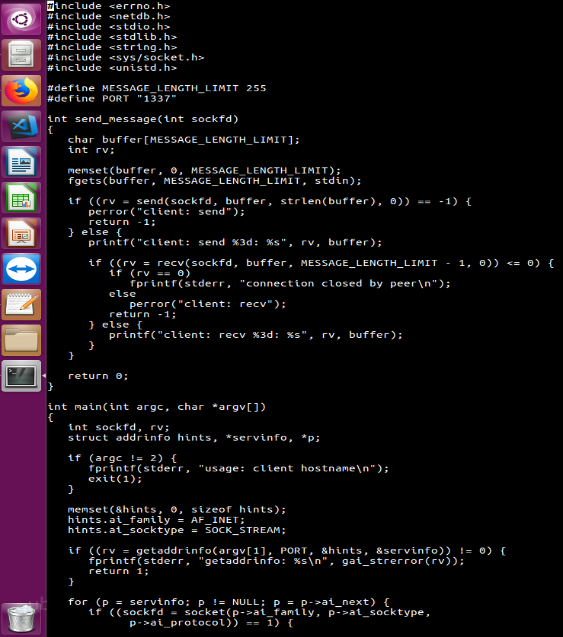
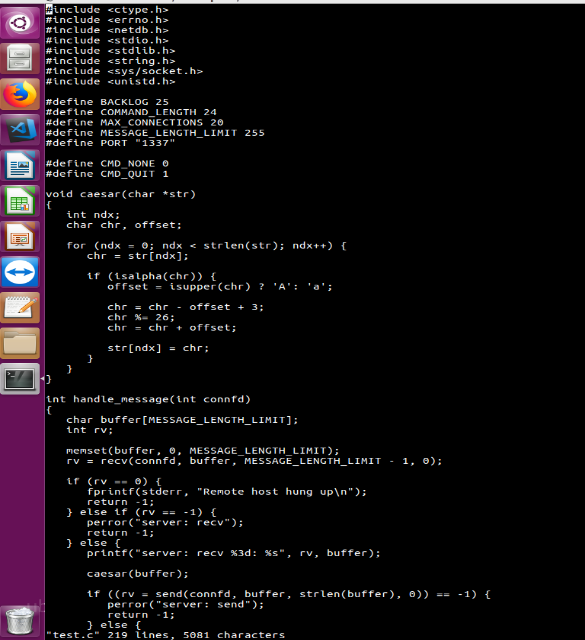


图2.1 vim编写client.c和server.c文件

**3：编译client.c和server.c**

**3.1 使用gcc编译client.c和server.c**

使用linux自带的编译工具gcc编译编写好的client.c和server.c文件，在命令行键入命令gcc -o client client.c编译client.c文件，如果文件没有语法错误则会生成名为client的可执行文件，同理编译server.c文件gcc -o server server.c生成server的可执行文件。

**3.2 使用Makefile编译client.c和server.c**

一个工程中的源文件不计其数，其按类型、功能、模块分别放在若干个目录中，makefile定义了一系列的规则来指定，哪些文件需要先编译，哪些文件需要后编译，哪些文件需要重新编译，甚至于进行更复杂的功能操作，因为 makefile就像一个[Shell脚本](https://baike.baidu.com/item/Shell%E8%84%9A%E6%9C%AC)一样，其中也可以执行操作系统的[命令](https://baike.baidu.com/item/%E5%91%BD%E4%BB%A4/8135974)。

编写Makefile文件来编译client.c和server.c生成对应的可执行程序。

CCFLAGS=-O2

all: server client

server: server.c

$(CC) $(CCFLAGS) -o server server.c

client: client.c

$(CC) $(CCFLAGS) -o client client.c

clean:

$(RM) client server

其中各处意思为CCFLAGS为最优化编译文件、all: server client为Makefile基本格式

target ... : prerequisites ...

command

...

其中,

target - 目标文件, 可以是 Object File, 也可以是可执行文件

prerequisites - 生成 target 所需要的文件或者目标

command - make需要执行的命令 (任意的shell命令), Makefile中的命令必须以 [tab] 开头。

$(CC) $(CCFLAGS)为系统命令表示系统cc编译工具CCFLAGS为最优化编译，$(RM)表示删除编译过程中的临时文件。

**4：系统编程实现**

**4.1服务器端的实现**

首先实现服务器和客户端的TCP连接方式，其中AF\_INET表示IPV4连接，

struct addrinfo hints;

memset(&hints, 0, sizeof hints);

hints.ai\_family = AF\_INET;

hints.ai\_socktype = SOCK\_STREAM;

hints.ai\_flags = AI\_PASSIVE;

SOCK\_STREAM表示是面向连接的SOCKET，是基于TCP的，数据传输比较有保障。AI\_PASSIVE表示调用者将在bind()函数调用中使用返回的地址结构。当此标志不置位时，表示将在connect()函数调用中使用。Getaddrinfo()解决把主机名和服务名转换成套接口地址结构的问题,接着通过sockfd = socket(p->ai\_family, p->ai\_socktype, p->ai\_protocol)建立socket连接此为IPV4面向连接的TCP通信，使用bind(sockfd, p->ai\_addr, p->ai\_addrlen) 把一个本地协议地址赋予一个套接字绑定指定地址和端口，通过listen(sockfd, BACKLOG)开启服务器端对指定地址和端口的监听。通过fd\_set readfds; 设置文件描述符集类型和select(max\_fd + 1, &readfds, NULL, NULL, NULL);函数来监听stdin、客户端连接请求和客户端发来的信息变化，如果变化出现则通过FD\_ISSET(conn\_fds[ndx], &readfds)检测，FD\_ISSET可以过滤掉那些没有出现变化的文件描述。接着如果有客户端请求连接则通过connfd = accept(sockfd, (struct sockaddr \*) &remote\_addr, &addr\_size)函数建立与服务器端的连接。如果已建立连接的客户端发送数据通过rv = recv(connfd, buffer, MESSAGE\_LENGTH\_LIMIT - 1, 0);函数接收客户端的数据，并通过rv = send(connfd, buffer, strlen(buffer), 0))函数将客户端的数据加密后发送给对应的客户端。如果服务端收到“quit”退出命令，则结束服务端程序。

**4.2客户端的实现**

客户端的实现比较简单，首先就是建立与服务器端对应的通信协议，

struct addrinfo hints;

memset(&hints, 0, sizeof hints);

hints.ai\_family = AF\_INET;

hints.ai\_socktype = SOCK\_STREAM;

其中各字段的含义与服务器端相同，表示IPV4的TCP通信协议。接着通过getaddrinfo()函数把主机名和服务名转换成套接口地址结构，然后即可通过sockfd = socket(p->ai\_family, p->ai\_socktype, p->ai\_protocol))函数建立对应协议的socket通信，通过connect(sockfd, p->ai\_addr, p->ai\_addrlen)函数连接到服务器端，此时服务器端会收到连接请求。最后通send(sockfd, buffer, strlen(buffer), 0) 和recv(sockfd, buffer, MESSAGE\_LENGTH\_LIMIT - 1, 0)函数发送与接收服务器端信息。

**4.3系统实现演示截图**

其中服务器和客户端的通信过程如图4.1所示

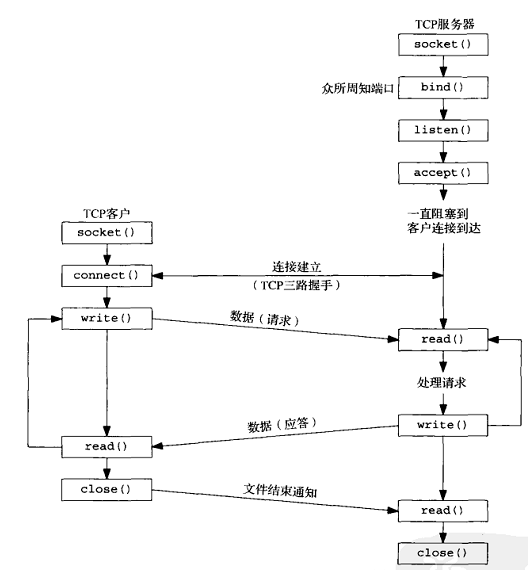
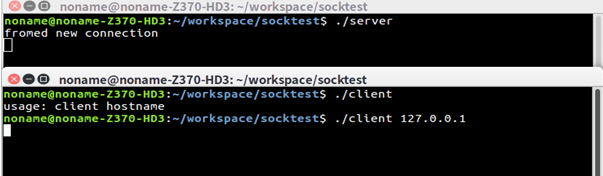
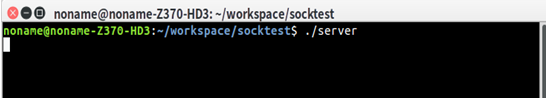
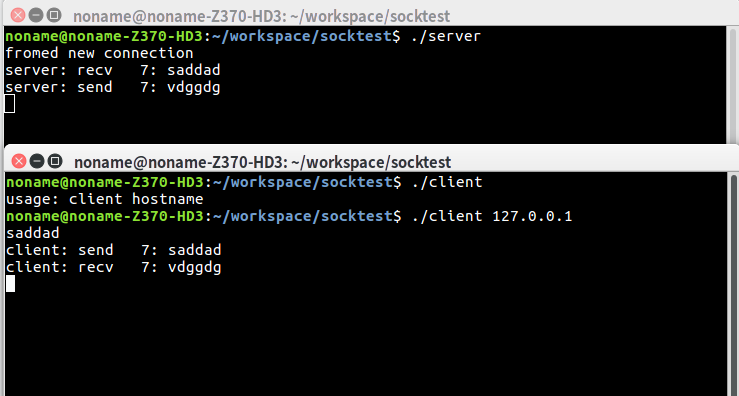


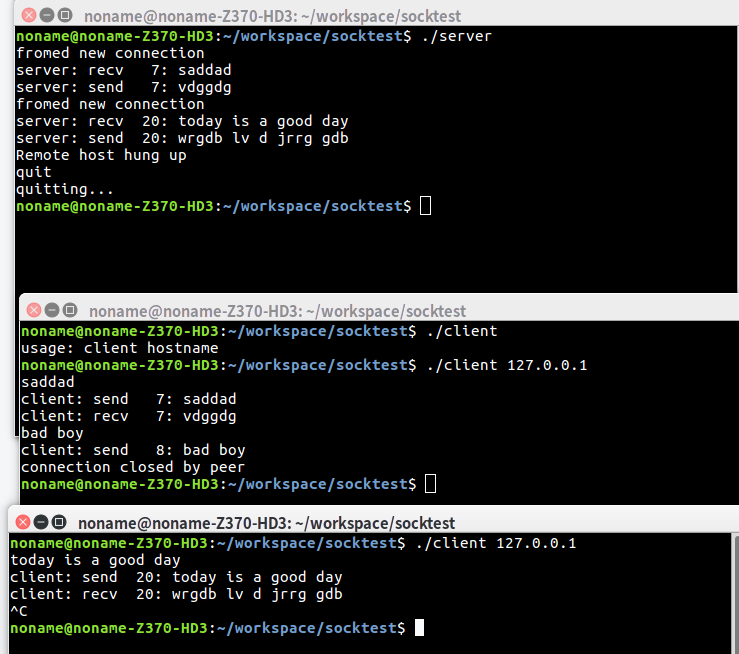
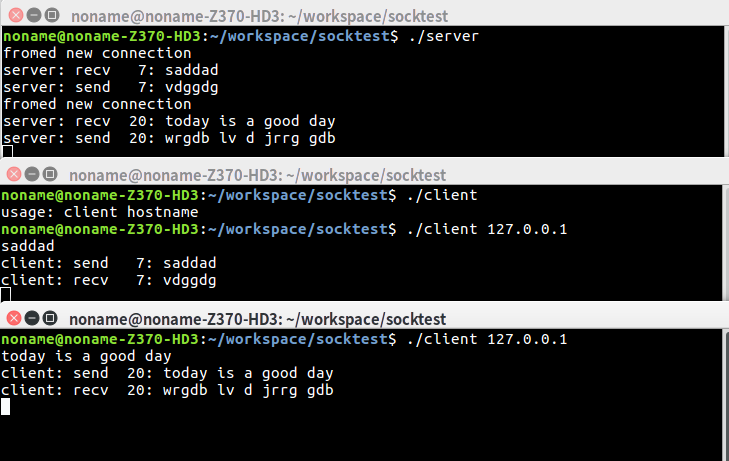
图4.1 服务器和客户端的通信过程

打开Linux系统的命令行运行第三步编译好的可执行文件，先运行服务器端，等待客户端的连接

接着运行客户端的程序，要传一个客户端的地址，由于是本机所以此处传的值为127.0.0.1本机ip地址，如果这里服务器端和客户端在同一局域网的不同主机上则需要传服务器脚本所在的ip地址。此处不需要出端口值，端口值是内置在脚本中的，服务器和客户端需一致。

可以看到服务器端显示已于客户端实现连接，接下来客户端可以发送信息给服务器实现通信

看到客户端对服务器端发送了一段信息，服务器端对该信息进行加密后发送给客户端，加密方式可在源代码中查看为caesar()函数。

接下来我们再打开一个客户端，与服务器端建立连接并通信，通过下图可以看见第二个客户端对服务器端发送信息“today is a good day”。服务器端显示收到一个新的客户端连接，并对第二个客户端返回加密后的信息“wrgdb lv d jrrg gdb”，在此过程中第一个客户端也可以对服务器发送信息，课件客户端之间是不相互影响的。

通信完成后第二个客户端程序退出，服务器会收到该客户端已断开连接，接着将服务器程序退出，此时第一个客户端若再次发送信息给服务器，则会发送错误，显示服务器已断开连接。

具体实现可在源代码文件中查看。

**5：研究生阶段涉及开发工作中，最欣喜沮丧的一个瞬间**

在开发嵌入式摄像头此为HI3516A，视频行为监控时，在参照HI3516A自带的开发说明文档，接通SDK可以实现对摄像头码流转换，此为YUV420转为RGB图像后。有一项工作是将程序检测到的视频中特点物体进入自己定的一个封闭区域后，传该物体的坐标位置给服务器端，此处有一台电脑作为服务器控制多个摄像头客户端。在该处由于物体一旦进入防区就会大概率持续在防区里面，所以客户端会一直向服务器传物体位置信息，由于项目对实时性要求较高，并且由于服务器需要控制多个客户端，使得服务器端接收信息会出现位置信息的紊乱，表现为客户端发送的信息连续性一致，但是长度不定，可是明明客户端在send()函数那里规定了发送多少字节的信息。这个问题很明显是由于多个客户端是多线程工作造成的，虽然在一开始我们怀疑是send和recv函数数据接收数的控制问题，可是在试了循环接收信息后，问题依然存在，我们马上意识到是由多线程通信造成的。奇怪的是在服务器端加了线程阻塞通信后，问题依然存在。问题一直持续到半夜还是没有解决。第二天，在我们重新检查整个项目的代码后，发现是昨天多线程通信控制那里函数使用出现问题。在解决后，看到客户端源源不断发送过来的连续的并且位置信息丝毫不乱的时候，无疑是最高兴的，可是，高兴中还夹杂着沮丧的情绪，因为就是不是很大的问题竟然花费了我们那么多的时间，痛定思痛后决定将今后工作过程中出现的大问题及其解决方案进行写博客的方式记录下来。