****



**网络聊天室实验报告**

**学生姓名 李丰杰**

**班 级 计科四班**

**学 号 3019244196**

1. 需求分析

在学习过应用层使用SOCKET编程的相关知识后，本次实验为让我们更好地理解TCP/UDP socket的原理，理解网络应用层协议架构，掌握相应架构的设计与实现方法，提升编程与实现能力，让我们通过使用Socket编程实现网络聊天室。

本次实验的目标是实现一个可供多用户使用的即时聊天室，支持发送文字消息、文件传输、语音聊天等功能，同时可以解决高并发、粘包、掉线等情况。

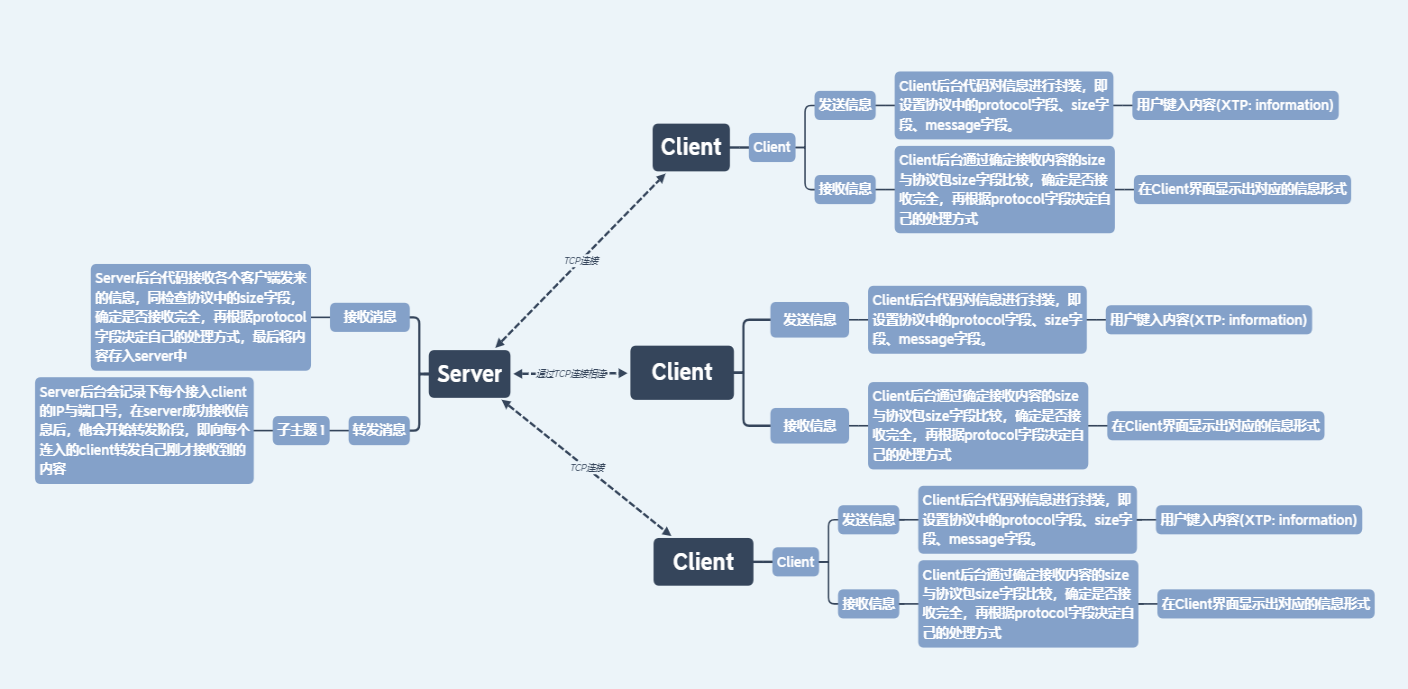
|  |  |
| --- | --- |
| 目标 | 是否完成 |
| 服务端可建立网络聊天室并等待客户端连接 | 完成 |
| 客户端可连入，并能将自己的消息发往服务端 | 完成 |
| 多客户端能及时受到并显示彼此的聊天消息 | 完成 |
| 新用户加入与旧用户退出不会影响聊天室的运行 | 完成 |
| 能正确处理粘包、掉线等异常情况 | 完成 |
| 系统进行高并发优化 | 完成 |
| 系统支持文件传输 | 完成 |
| 系统支持语音聊天 | 完成 |

1. 系统设计

该聊天室的架构为典型的CS架构，即多用户通过一个服务器来进行通信。为解决高并发问题，本实验需要对Server进行并发模式的设计；为了处理粘包问题和同时支持文本、文件、语音传输，我们需要设计协议头文件，来确定每次发送的内容格式、确定本次发送数据的内容；同时也要在Server中处理用户掉线后的正常处理。

在代码实现过程中，在实现高并发过程中通过对单线程Accept+多线程读写进行优化，通过主线程执行非阻塞Accept，每当客户端发来一个Connect，主线程就开始建立连接，然后新建线程来处理客户端的读写业务。

通过设计三个不同的协议：MTP、FTP、VTP来分别传递消息、文件、音频。同时通过对协议头的设置来确定每次数据传输的数据大小来解决粘包问题。在对语言问题传输的过程中，由于该实验在Linux中实现，所以调用ALSA声卡驱动来实现对声音的采集。

下图为系统的整个架构：

1. 协议设计部分

因为本实验需要正确处理粘包问题，且要支持信息、文件、语音传输的功能。所以本次实现将chatroom中发送的消息类型分为三类：信息、文件、语音，分别定义为MTP、FTP、VTP。Chatroom的协议头文件包括三个部分：protocol(包括MTP、FTP、VTP三类)、message(即传输的具体内容)、size(本次文件传输内容的大小)。

通过对协议头的分析，即可得到对内容的处理方式，可以正确接/发发送内容。因为在传输的过程中，原本的字符都会变成bit流，在接受后需要重新将其转化为UTF8字符流，然后可以根据协议头字段来具体分析如何处理本次接收内容(对消息、文件、语音的处理肯定是不同的)。

通过对size的分析可以得到本次传输内容的大小，接受信息的人可以通过确定接收内容的大小来确定文件是否完全达到，可以正确处理粘包问题。

其实可以继续丰富协议包的内容，比如加入校验位和对protocol字段可以不使用字符而使用数字来表示，但因为在lib.rs以及head.rs中已经将其用字符串表示了，再更改可能涉及到很多函数，可能造成意外的bug，所以就仍使用了字符，不过这样更有利于对整个系统的分析、debug，更加直接明了。但会使协议头的长度增加，但也可以接受。

协议在对于信息的传输，可以在确定完全接收后直接转发给各个客户端；但对于文件传输就比较复杂，首先客户端要将要发送文件放在client\_files目录下，然后进行上传。服务器在接受到后也要进行下载，存入server\_files目录下，然后再转发给所有客户。客户端在接受文件时也要经历和服务器端类似的过程。

另外需要额外处理一下文件名相同的问题，在lib.rs中存在一个rename\_filename的函数，可以将文件名后面加上数字来将文件名更改可以正常保存。

在客户端发送：用户需要自己键入本次要发送内容的类型：信息、文件还是语音(即键入MTP: 或FTP: 或 VTP: 然后再输入内容)。然后客户端后台会有代码来find信息中的类型，设置协议中的protocol部分是MTP还是FTP或VTP；也会有代码来计算本次发送信息的内容大小，并设置协议中的size部分的数值。另外对于语言传输会调用系统的alsa声卡来采集声音，然后再进行发送。

在服务器端接收：服务器会一直运行并监听，如果接受到信息，就会首先根据协议中的size值与本次接收的大小进行比较，决定是否继续接收；然后根据协议中的protocol值决定server对本次内容的处理形式(MTP的话可直接转发、FTP则需要下载文件后再转发、VTP也需要下载内容再转发)。

在服务器端转发：服务器有一个消息队列，其中存储了所有接入服务器的客户端的IP与端口，在服务器接收到信息并处理后，服务器会重新自己构建一个信息(添加发送客户的IP和端口、以及协议中的传输类型、大小、内容等字段)，然后转发给客户端。

在客户端接收：客户端也会像服务器接收一样存在一个接受传输内容的过程，首先根据协议中的size值与本次内容的处理形式来确定是否继续接受；然后根据协议中的protocol字段确定自己对接受内容的处理(MTP直接显示、FTP下载文件、VTP接受语音)，另外会在前面加上发送者的IP和端口号。

1. 实现方式

本次实验在Ubuntu 20.04系统下进行，使用RUST编程语言、在进行对语音传输支持的部分使用了ALSA声卡调用库(在Cargo.toml文件的dependencies中声明)、另外也使用了标准库中的多生产者单消费者以及多线程库函数。

下面为源代码的目录结构与文件：

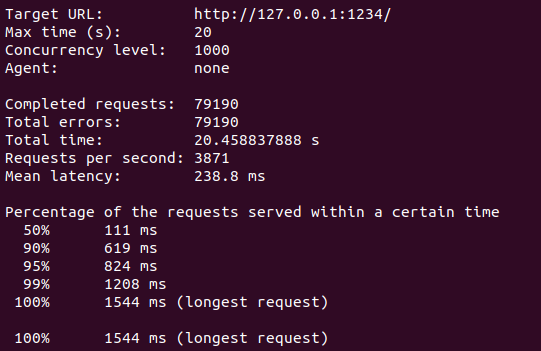


1. 性能评价报告

|  |  |
| --- | --- |
| 目标 | 是否完成 |
| 服务端可建立网络聊天室并等待客户端连接 | 完成 |
| 客户端可连入，并能将自己的消息发往服务端 | 完成 |
| 多客户端能及时受到并显示彼此的聊天消息 | 完成 |
| 新用户加入与旧用户退出不会影响聊天室的运行 | 完成 |
| 能正确处理粘包、掉线等异常情况 | 完成 |
| 系统进行高并发优化 | 完成 |
| 系统支持文件传输 | 完成 |
| 系统支持语音聊天 | 完成 |

系统可行性评价：

性能评价：



通过loadtest压力测试程序测量，可以得知server最多每秒处理3871个request，平均延迟为238.8ms,其中50%在111ms内处理；90%在619ms内处理;95%在824ms内处理;99%在1208ms内处理;100%在1544ms(最长)内处理。

六、总结

本聊天室实现了预期的规划目标，完成了扩展功能。你可以进入Chatroom目录后执行cargo run - - bin server 运行服务器，执行 cargo run - - bin client 执行客户端。

本次实验是第一次使用RUST语言编写代码，同时也是第一次进行socket编程。通过本次实验，首先让我对RUST语言有了更深的了解，同时也学习了很多标准库之外的内容。在编写代码的过程中常常遇到了很多的不解与困惑，通过查阅网上资料，跟随着一个rust网络编程教学逐步实现了基本功能。在实现扩展功能时和一名也使用rust语言的同学进行了学习交流，逐步完成了自己的代码。今后我一定会继续在RUST和C++语言方面学习，争取精通其底部构造。

通过实验我也逐渐明晰了socket编程的知识，明晰了应用层协议设计的根本就是通过socket来编写自己的数据传输规则。虽然对程序中的很多API并不理解，但就像应用层从来不会去理解底层一样，不了解底层依旧可以编写出聊天室的协议。这可能就是互联网协议分层的意义的一个体现吧。

另外在此过程中，我也深深体会到了编写文档的重要。在之前，我都是不管三七二十一，先开始写代码再说，但通过这次跟着课程规划一步一步地写文档，感觉对这个工程有了一个不一样的视角。在编写代码之前可以对它有一个全局的、比较深刻的认识，明晰了自己的目标，在实现目标时，查阅资料也变得更加有目的性。感觉用软件工程的思想写project真的会很不一样！

另外在压力测试的时候也学习了很多先进的测试程序，开拓了自己的知识面。

今后会继续努力，学习课堂知识，学习rust和c++，编写xv6系统，学习底层构造，成为一个合格的底层人才。