

**2023~2024学年 第二学期**

**《计算机网络课程设计》**

设 计 报 告

班 级 \_\_\_\_\_计科21-5班\_\_\_\_\_\_\_\_\_

学 号 \_\_\_\_\_2021218257\_\_\_\_\_\_\_\_\_

姓 名 \_\_\_\_\_ 卢含宇 \_\_\_\_\_\_\_\_\_

2024年4月20日

目录

[1 课设选题 3](#_Toc164978930)

[2 设计贡献 3](#_Toc164978931)

[2.1 整体架构设计 3](#_Toc164978932)

[2.2 相关数据结构设计 4](#_Toc164978933)

[2.3 宏 6](#_Toc164978934)

[2.4 多线程函数 6](#_Toc164978935)

[3 使用说明 7](#_Toc164978936)

[4 不足与后续改进之处 7](#_Toc164978937)

[5 个人思考 8](#_Toc164978938)

[6 源代码 8](#_Toc164978939)

# 1 课设选题

设计二.网络文件传输

完全使用c语言，在linux平台进行开发，使用udp协议，在应用层实现多线程收发，具备SR协议滑动窗口，包含客户端和服务端的可靠udp文件传输协议。整体采用推拉式模式，由客户端请求服务器的文件列表，并发送对应的指令请求服务器中的文件，服务器建立多线程，并为每一个线程分配一个独立的socket用于数据流的传输。单次连接可以允许多次文件传输，并且可以对服务器的文件目录进行切换，

服务端：具有接收命令，命令解析，获取并发送文件目录，多线程文件发送等功能。实现对于音视频文件的传输。

客户端：具有发送命令，多线程接收，写文件，数据校验，滑动窗口，保序接收，向服务器发送确认帧等功能。

# 2 设计贡献

# 2.1 整体架构设计

服务器设计：首先创建一个服务器端用于接收来自客户端的连接请求的seversocket，此处暂时使用udp socket进行单连接，后续可以改进为tcp socket连接，实现单服务器，多客户端的多路复用模式。然后接收来自客户端的命令包。其中数据包都命令部分包含服务器需要解析的指令。指令包含以下部分。

get col：获取服务器的工作目录部分所有的文件目录项，本人在设计时使用了Linux关于文件的系统调用dirent.h用于进行目录操作。此方法可以更加灵活，获取所需的目录文件的各种属性内容。或者使用Linux调用指令execl执行ll指令，使用dup2将输出描述符重定向到文件中，将重定向文件作为目录包发送给客户端，此方法更为简便，可定制性较差。

cd …：修改服务器的工作目录，与linux系统的cd命令功能相同，此命令执行时同时向客户端发送新的目录包。

get …: 获取工作目录文件，并分析请求文件的合法性，如果文件不存在，则返回一个错误包，后续可以通过fcntl获取文件更多属性，增加用户读写权限检查等附加功能，如果文件地址合法，则服务器创建多线程，并为线程分配socket，开始与客户端进行文件传输。具体工作流程在多线程函数的实现部分介绍。

客户端设计：

客户端首先创建socket与服务器进行连接，请求文件服务器目录，并发送文件请求包，客户端首先要在本地进行命令有效性检查，如果命令非法，则返回非法输出，减少与服务器的交流次数，减少服务器的压力。进行文件传输时，客户端首先要接收一个文件信息包，用于确定总数据块的数量，之后开启两个线程，其中一个线程负责接收服务器的数据包，并进行md5的抽样校验，当数据包md5与本地md5校验获取的数值相同时，将其送入到缓冲窗口中，此时并不给予确认号。如果md5校验失败，则向发送线程发送一个失败确认帧，发送线程会立刻重复发送该包。另一个为写入线程，负责在本地创建文件描述符，将窗口中的连续部分写入到文件中，并依次对写入后的包的发送线程发送一个带有数据包序号的确认帧，使得发送线程可以继续工作。

# 2.2 相关数据结构设计

（1）滑动窗口window.h

在设计时，滑动窗口有两种设计方案，其一是采用位图的方式，为文件分配一个与包数相同的位图，当数据包到达时，将对应位图位权置1，并暂存起来。当写入线程获得cpu执行权时，依次扫描位图中连续位为1的位置，并向发送线程发送确认帧。这种方法更偏向于底层，且操作起来执行效率更高，空间使用效率也更高。但设计起来十分繁琐，时间缘故抛弃了这一方案。

具体实现采用了带有头节点和尾节点的有序双向链表的方式。当有一个数据包到达时，客户端接收线程会根据数据包的seq字段将数据包插入到链表的对应位置。当窗口满时接收线程将挂起，或者cpu自然挂起接收线程时，控制权交给写入线程。写入线程会依次读取链表前端有序的部分，并依次向发送线程提交ack确认帧。当读取到一个非有序数据包时，写入线程将主动挂起，将控制权转交接收线程，继续接收数据包，直到数据编号再次有序为止。即实现了多发送窗口，多接收窗口，顺序提交的select repeat协议。

（2）数据包格式pack.h

数据包采用统一格式便于代码编写。使用一个结构体，将其看作一个有固定结构的字符数组。此处部分字段理论上应该使用比特表示，但考虑到数据包中的字节对齐以及操作的简便性统一化，故统一使用字节操作。

typedef struct Pack{

    char type;

    unsigned long seq;

    unsigned int dataLen;

    unsigned char md5[33];

    char data[MAX\_PACK\_DATA\_LEN];

    int port;

}\*PACK;

Type：数据包类型字段，0表示确认帧，1表示终止帧，2表示命令帧，3表示目录帧，4表示数据帧，5表示错误帧，6表示信息帧。

Seq：数据包在文件中的序号

Datalen：数据包数据部分的长度

MD5：数据部分的md5校验码，为了方便调试，因此采用明文传输

Data：数据包的数据部分，或信息帧的信息部分，或目录帧的目录信息部分，或错误帧的错误信息

Port：发送改数据帧的socket对应的端口号。

（3）重新封装的socket提供多线程确定socket对应的端口

typedef struct Sock{

    int socket;

    unsigned int port;

}\*SOCK;

Socket：记录服务器主线程创建的socket

Port：记录socket被分配的端口

（4）线程函数参数param.h

三个参数结构体对应三种不同的线程

struct severTransParameter{

    int fd;

    SOCK sock;

    struct sockaddr\_in clientSocketMsg;

    unsigned int clientMsgLen;

};

第一个结构体对应服务器发送线程的参数结构体，fd对应客户端请求文件的文件描述符，sock对应服务器为发送线程分配的socket，clientSocketMsg对应记录返回客户端信息的结构体，可以不需要，clientMsgLen对应客户端信息结构体的长度大小，可以不需要。

struct clientRecvParameter{

    int clientSocket;

    struct sockaddr\_in severSocketMsg;

    unsigned int severMsgLen;

    WINDOW window;

    unsigned long blokeNum;

};

第二个结构体对应客户端接收线程的参数结构体。clientSocket对应接收线程的socket，severSocketMsg对应记录返回服务器信息的结构体，可以不需要，severMsgLen对应服务器信息结构体的长度大小，可以不需要。window对应滑动窗口，blokeNum对应服务器传送文件的所有块的数量。

struct clientWriteParameter{

    int clientSocket;

    const char\* path;

    WINDOW window;

    unsigned long blokeNum;

};

第三个结构体对应客户端写线程的参数结构体，clientSocket用于对服务器发送线程对应的端口发送确认帧。Path对应接收端要写入文件的地址。Window对应滑动窗口，blokeNum对应客户端写线程要写入的所有块的数量。

# 2.3 宏

#ifndef MAX\_PATH\_NUM

#define MAX\_PATH\_NUM 128

#define MAX\_INSTRUCT\_LEN 256

#define MAX\_PATH\_LEN 256

#define MAX\_PACK\_DATA\_LEN 20480

#define MAX\_THREAD\_NUM 8

#define MAX\_WINDOW\_SIZE 10

#define UHUNG\_TIME 20

#endif

MAX\_PATH\_NUM：允许的最大目录条数

MAX\_INSTRUCT\_LEN：允许最大的指令长度

MAX\_PATH\_LEN：允许最大的路径长度

MAX\_PACK\_DATA\_LEN ：允许最大的数据包长度

MAX\_THREAD\_NUM：允许设置的发送线程数

MAX\_WINDOW\_SIZE：允许最大的窗口大小

UHUNG\_TIME：默认线程挂起时间

对于不同的测试需求，可以通过修改宏定义来实现。

# 2.4 多线程函数

对于多线程函数的编写，则是最复杂的部分。首先，线程之间的数据共享，需要有序进行资源使用的调度。因此知道在什么地方上锁则是最困难的部分。同时，上锁时要考虑锁的粒度，防止单一线程对cpu资源的独占，造成其他线程的资源饥饿现象。其次，线程是轻量化的进程，没有自己独立的资源空间，多个线程共享父进程的所有资源，包含父进程的内核资源。这也为一些功能的实现造成了困难，之后会再不足之处中详细解释。下面将介绍已经实现的多线程函数思想。

（1）发送线程函数

void\* serverTransThread(void\* param);

发送线程需要创建两个临时包，其一用来将读取文件块的数据封装成包发送给客户端，其二用来接收客户端发送的确认帧。首先，多线程之间共享父进程打开的文件描述符，因此，多个线程连续的读取同一个文件描述符可以免去创建多个描述符的定位问题。其次因为文件描述符属于共享资源，因此所有与文件描述符相关的操作，都需要上锁。然后，由于不清楚是哪一个线程读取到最后的文件描述符，因此只有当所有的线程读取文件的长度均为0时则说明文件已经彻底读取完毕，此时才可以将发送线程回收。其次，由于发送时可能会出现比特错误，即md5校验失败，因此在发送sendto时，仍然是一个死循环，跳出条件是收到的确认帧序号等于当前包的序号。在计划中，本来是要设计超时重发的机制，但由于时间缘故，我选择使用阻塞式socket，因此并便于准确的计算实际的链路延迟，因此暂时没有实现，在下面改进部分我将给出改进思想。

（2）接收线程函数

void\* clientRecvThread(void \* param);

接收线程仍是一个死循环，跳出条件为当接收到全部的包时即可结束接收线程。接收方接收线程和写入线程共享滑动窗口，因此所有对滑动窗口的操作都要上锁。首先，当滑动窗口满时，要将cpu控制权转交给写入线程，然后对收到的数据包进行md5的数据校验，当校验通过时，则可以将数据包加入到滑动窗口中，因为滑动窗口使用的是有序链表，因此在插入时会保证链表的前一部分必定有序。当md5校验失败时，则根据数据包所携带的端口号，向这一端口发送一个失败校验包，当发送线程收到这个错误包时，则会将当前数据包重复发送一次。

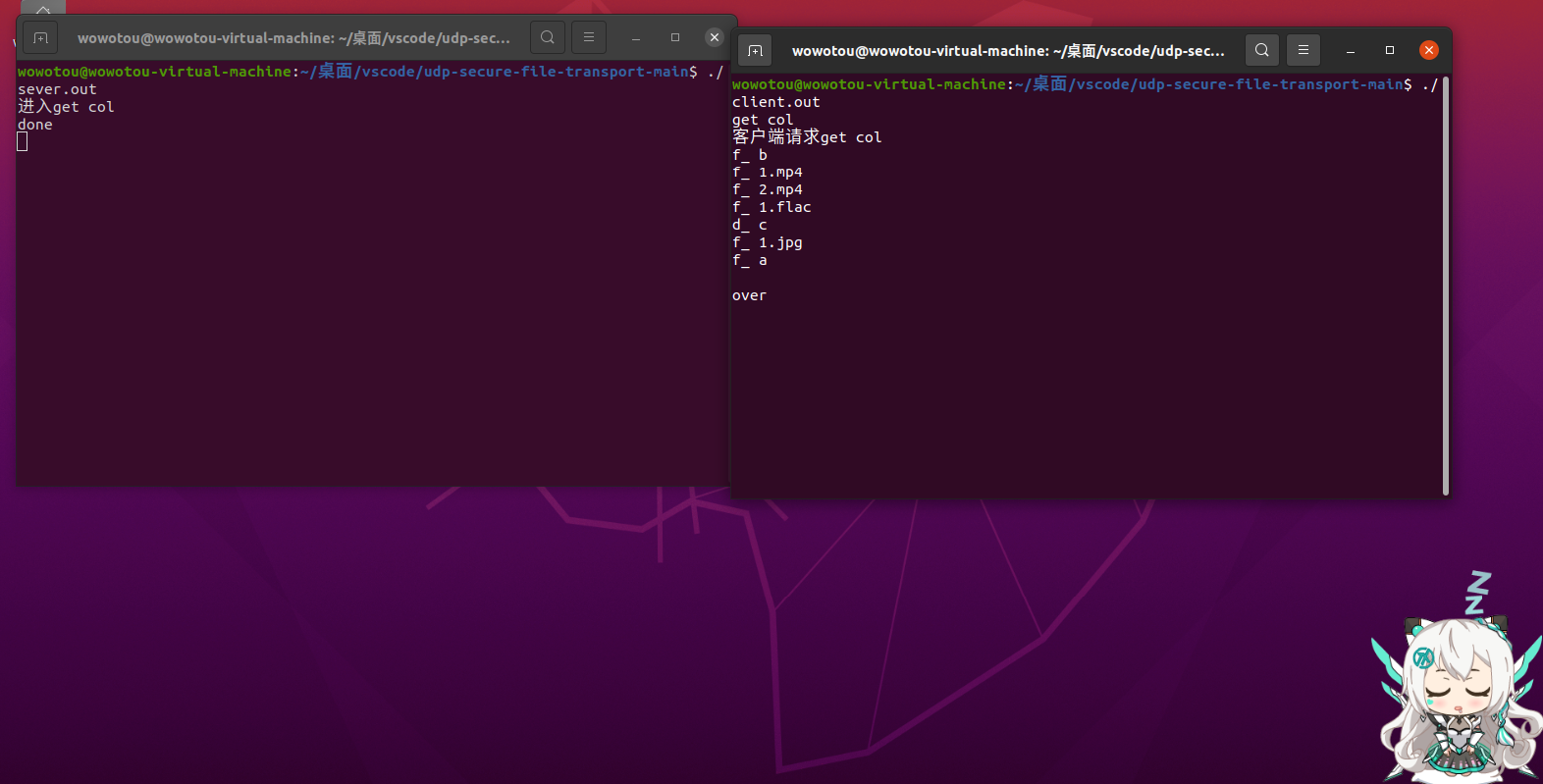
（3）写入线程函数

void\* clientWriteThread(void\* param)

写入线程会连续读取滑动窗口的内容，当读入到的数据包序号等于当前自己需要的数据包序号时，则将该数据包写入到文件中，并且，向发送线程发送一个确认帧，解除发送线程的阻塞状态，发送线程可以继续发送下一个数据包。当读取到的包并不是所需的包的时候，则说明读取到了第一个不连续的包，需要将内核的控制权转交给接收线程让其继续数接收所缺少的数据包。当窗口中的数据被全部读取时，同样立刻将内核控制权转交给接收线程。

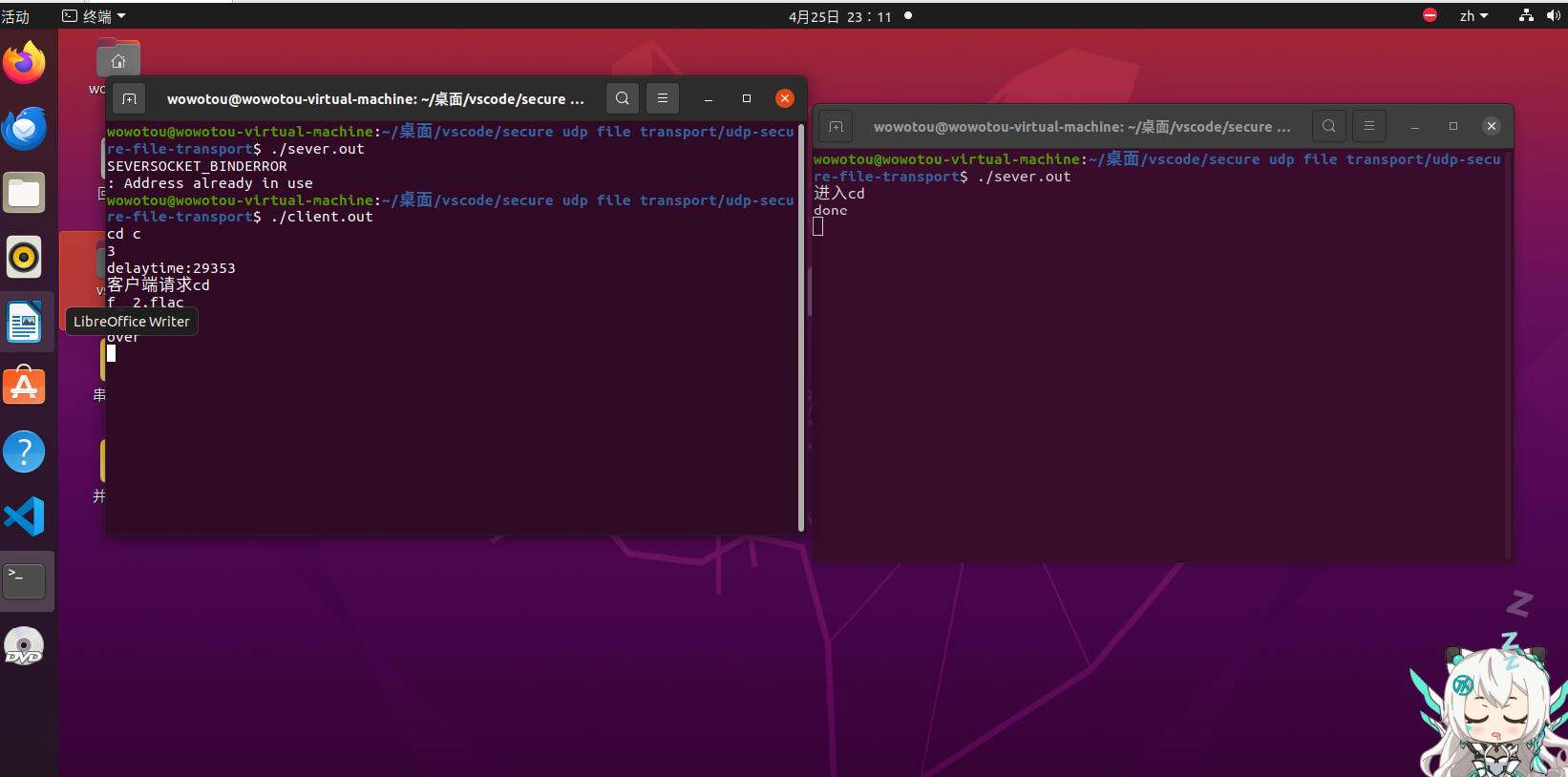
# 3 使用说明

(1)获取服务器目录get col

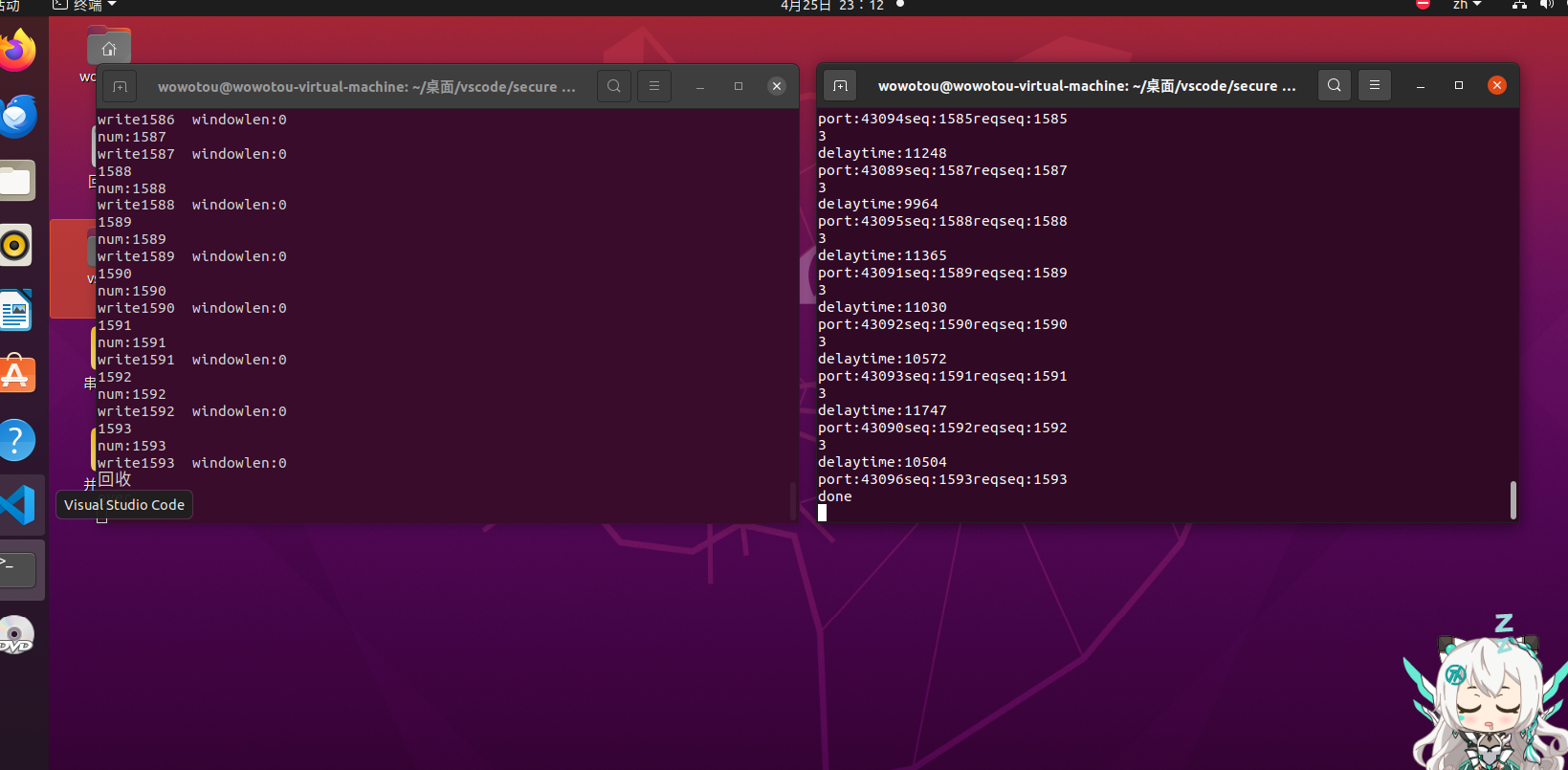


以f开头的是文件，以d开头的是目录

（2）进入服务器目录cd …..



（3）获取文件get … …



Num为客户端接收到的包编号

Delaytime是发送数据包和接收ack之间的延迟

Port是发送线程的端口号

Seq是发送线程发送的包的序号

Reqseq是发送线程接收到的ack包回应的序号

# 4 不足与后续改进之处

上面说过，本协议的最致命的部分即未能实现超时重发机制。其一的原因是，采用的阻塞式socket，在发送后没有接收到确认帧时是完全阻塞状态，无法执行后续的代码，因此，如果单纯时使用两个计时器来记录发送时的时间戳和接收后的时间戳得到的并不是链路上的延迟，且无法在计时器到达设定时间时中断程序。

再之，使用alarm系统调用可以实现内核内的时钟定时，当计时器结束时触发一个alarmSignal，继而调用自定义的软中断处理程序，但由于我采用的是多线程模式，当内核产生时钟中断时，归属于父进程的所有子线程均会收到来自于内核的定时器信号，进而产生竞争现象，无法准确定位真正出现超时的线程，因此也无法实现超时重发机制。个人对此现象的解释是，信号对于父进程下，功能各不相同的线程时，所能触发的信号也是不同的，因此可以通过设置每一个子线程的mask信号屏蔽字来分离不同线程可以处理的内核信号，但我设计的发送线程功能均是相同的，而计时器信号只有一个，因此会产生竞争现象。

而个人所能思考到的解决方法是，将socket设置为非阻塞模式，实现类似select模型的轮询机制，将sendto与recvfrom成对的写入在一个死循环中，每隔一定的时间计算循环执行的时间，当到达宏设定的固定时间时，或使用上一次成功计算出的链路延迟经过特定公式计算的超时时间时，即触发超时重发的操作。

其次，要实现多路io复用，可以将服务器的socket设置为tcp socket来侦听客户端与服务器的连接请求。每当收到一个客户端的连接请求时，可以为其单独创建一个进程，来单独与客户端进行通信，这种方法对于系统资源的需求较大，当客户端请求连接数量较多时，为了保持服务器的响应能力，会将大量的时间浪费在进程切换当中。或者使用一些诸如select，epoll等多路io复用的网络模型。但这种方法需要在服务端维护多个请求响应队列，实现起来较为复杂。

# 5 个人思考

对于多线程传输，并不是并发的线程数量越多越好，对于多核处理器来说，并不是并行的线程越多，传送速度越快。原因是，在局域网内信道资源的数量时有限的。当发送线程过多时，信道中的数据增多，信道也会变得拥挤，数据包丢失的概率也会更大。其次信道变得拥挤，信道冲突的概率也会变得更大，出现比特差错的概率也会更大，重传的次数也会变多。

其次对于可靠传输，握手是一个不错的解决方式，但如果每一次传输都需要握手，对信道资源的消耗也是巨大的，且每一次握手的信号都有丢失的可能性，如果只用握手来确保可靠传输，则会无穷无尽的握手。因此通过增加超时重传的机制也是一个重要的补充部分。除此之外，通过向信道中泛洪，一次性向信道中传送多个相同的数据包也可以确保信息能够被接收方收到，但这种方法最好是在发送一些只需要发送一次的属性信息时比较适合。

# 6 源代码

