传输协议格式

（1）约定实地址为用户端地址，虚地址为服务器端地址，以下请求地址默认为实地址
（2）请求类型等等只占一个字节，下面为了表示方便所以写成字符串
（3）传输的原则是，一次请求对应一次回复。得到回复之后请求就结束。至于请求之间的逻辑关系是服务器和客户端自行负责。例如传输文件分为两步，第一步客户端向服务器发送传输文件请求，服务器给予答复（允许传输或者不允许传输）；第二步服务器向客户端索取文件，客户端传输。
（4）{}表示一项，实际传输中会进行压缩
（5）项分为可变长度和不变长度，不变长度放在前面，这样可以用 struct 很方便地读取出不变的项目，并且可变长度项的长度也会被包含在内，所以根据这些长度读取剩下的内容即可。例如新增文件可以写成类似如下格式：

```
struct acquire
{
    int id;
    int type;
    char file_md5[32];
    int startpos;
    int endpos;
    int contentlen;
};
```

1、删除文件
请求：“{id} {delete} {addr_len}{file_addr}”
回复：{id}{status} （status:删除成功、暂时无法删除、删除失败）

2、 新增文件(只发送文件信息，此时需要打开，直到文件传输完成或者通知不需要传输文件)
请求：“{id}{add}{file_md5}{file_time}{addr_len}{file_addr}”
回复：{id}{status} （status:允许新增、暂时无法新增、无需新增）

3、传输文件（真正进行文件传输）
请求：“{id}{acquire}{file_md5}{start_pos}{end_pos}{addr_len}{file_addr}”
回复：{id}{status}{start_pos}{end_pos}{content}文件传输内容包

4、 修改文件（夹）名
请求：“{id}{modify}{addr_len}{name_len}{file_addr}{file_name}”
回复：“{id}{status}”(修改成功、失败、暂时无法修改)

5、打印目录结构，该目录为虚目录
请求：“{id}{print}{depth}{addr_len}{file_addr}”
回复：“{id}{status}{tree_len}{tree}”

6、注册
请求：“{id}{register}{passwd_len}{name_len}{user_passwd} {user_name}”注册

回复：“{id}{status}”　成功、失败（包括失败原因）

7、登录
请求：“{id}{login}{count_len}{user_count}{user_passwd}”
回复：“{id}{status}”
登录后在服务端创建“socket”——“用户 id”的哈希表

数据库设计

用户信息
User_info

User_id	int	用户编号
User_count	string	用户账户
User_name	string	用户名
User_passwd	string	密码（MD5 加密，密文验证）

服务器文件夹信息
该表记录了共享文件夹的对应关系，当系统启动的时候，会根据这张表生成虚实地址转化树
Folder_info

Folder_info_id	int	文件夹编号
User_id	int	用户编号
Addr_server	string	服务端虚文件夹
Addr_local	string	客户端实文件夹
Isbind	bool	是否绑定

文件存储
所有的真实文件将放在服务器的指定位置，并且每个文件的命名就是该文件的 Md5 码。这些真实文件的 md5 码集中存放在该表中，并记录指向该文件的连接数（当连接数减为 0 时可以看做是放入回收站，当空间不够或者到了指定时长这些连接数为 0 的文件可以被清理）
File_num

File_md5	string	文件 MD5
File_num	int	指向该文件的连接数

存储地址方式
经过仔细考虑，最终决定使用邻接表来存储虚地址和虚文件，即每个节点记录其父亲节点的 ID。该表中同时存储了文件夹和文件，对于文件还会存储文件的 MD5 码和修改时间（以时间戳的形式存储）。
File_md5_info

File_id	int	文件编号	
File_md5	string	文件 MD5 码	default=null
File_name	string	客户端文件名	
File_parent	int	父节点 id	

File_modifytime	int	修改时间的时间戳

日志信息：
日志最终将被存储在数据库中，以方便查找日志。

Log_info

Log_id	int	日志编号
Log_operation	int	操作类型
Log_userid	int	操作者
Log_addr	string	操作对象地址
Log_time	int	操作时间（时间戳）

未完成传输文件信息：
该信息用于断点续传，每当发生一次写入磁盘操作，相应地修改对应行的 acq_finish，知道传输完毕，表明 acquire 这个请求结束，将该项从中删除。当断开重连的时候，首先将继续这些未完成的传输文件。

Unfinished_acquire

acq_id	int	编号
file_md5	string	未完成传输 md5 码
acq_finish	int	已经完成传输字节
file_startp	int	请求传输的文件开始位置
file_endp	int	请求传输的文件结束位置
file_time	int	文件时间（时间戳）

客户端没有数据库，因而需要将以上两张表（日志表和未完成传输文件信息表）写入到文件中。格式相同。

路径解析

在服务器和客户端交互过程中，客户端的实地址需要转化为服务器的虚地址才能在服务器上存储，反之亦然。由于共享目录下的普通文件和文件夹两端是一致的，因此只需要对共享目录进行转化即可。可以建立一棵实地址共享文件夹的树，叶子节点存储虚地址，因而当读入一个实地址的时候，就根据实地址从树根开始移动，移动到叶子节点便将对应实地址改为虚地址即可。反之亦然。

实现断点续传和分块传输

上文中谁需要文件谁发送 request 请求，并且 request 只请求传输该文件的[start_pos,end_pos)范围内的字节。因此这个方法可以用于实现分块传输和断点续传。

每接受指定长度的字节（8K）那么就会发生一次写入磁盘文件，此时传输未完成，并且随时可能被打断，所以要同时更新当前已经传输了多少字节，并且写入磁盘文件或者数据库。一旦发生突发情况，那么重新连接上去的时候根据这些未传输完毕的文件信息即可实现断点续传。

存储方案设计

如上面数据库所设计的那样，我们将会把用户的目录和文件以邻接表方式存入数据库中，其中文件实际上只是一个虚文件，只是一个指向真实文件的连接（如同软连接一样，不过软连接的方式我们经过仔细考虑认为其速度太慢放弃了）。

而所有用户的真实文件我们将会保存在服务器的一个指定位置上，并且就以文件的 MD5 码命名，由于 MD5 码是区分文件的唯一标志（在本问题中），因而这样不会出现冲突。然后在数据库中记录该文件以及指向该文件的链接数，为 0 的时候理论上可以删除该文件（无紧急情况会保留，以防止后面会再次恢复）。

用户目录结构设计

用户目录是由上面的两张表构成，一张记录了服务器和客户端的共享文件夹对应关系，并记录了这一对应关系是否绑定。通过这张表并配合虚实路径转换树（上文中的路径解析）即可将客户端路径转化为虚路径（用户目录和用户文件），在服务器上以邻接表的方式存储该虚目录和虚文件即可。