
南開大學
Nankai University

计算机网络

实验报告

# 类 FTP 服务器的实现

作者：李寅昊
学号：1611303
专业：计算机科学与技术

# 实验要求

实现一个类 FTP 功能的协议，实现文件的上传和下载和服务器共享目录的查看。

要求：

i) 下层使用 UDP 服务。

ii) 支持多用户。

iii) 多用户文件的上传和下载可以仅支持当前目录。

iv) 给出协议的具体内容。（5）给出收发双方的交互日志。

v) 使用 c 系列语言进行实现。

# 目录

# 第 1 章　协议内容声明

## 1.1　自定消息类型

在本程序中，我们自定消息类型，并将其作为客户端和服务器之间交互信息的载体。我们所定消息的定义如下:

```
//自定信息类型
struct mymessage
{
    int type;//消息类型
    char mss_buffer[BUF_SIZE];//发送数据
    char file_name[BUF_SIZE];//发送文件名
    int len;//发送数据长度
    int seq;//发送数据序号
    int ACK;//申请下一个数据包序号
    bool end_flag;//结束标志位
};
```

图 1.1: mymessage

此结构体定义分别被写在 myclient 类中以及 myserver 中，从而保证两端信息定义的一致性。在 mymessage 中，我们分别存放了数据信息，用来发送数据，以及一些必要的标记信息，用来客户端与服务器的交互。

## 1.2　自定协议交互过程

在本次实验中，我们使用一下宏定义信息作为 mymessage 的 type 位，用来标记发送信息的类型

```
#define BUF_SIZE 1000
#define APPLY_FILE_DIC 0
#define AGREE_FILE_DIC 1
#define APPLY_UPLOAD_FILE 2
#define AGREE_UPLOAD_FILE 3
#define START_UPLOAD_FILE 4
#define END_UPLOAD_FILE 5
#define RECEIVE_UPLOAD_FILE 6
#define APPLY_DOWNLOAD_FILE 7
#define AGREE_DOWNLOAD_FILE 8
#define START_DOWNLOAD_FILE 9
#define END_DOWNLOAD_FILE 10
```

图 1.2: mymessage 的 type 位信息

协议交互过程如下：

1) 当用户点击获取共享目录时，客户端则会向服务器端发送一个 type 为 APPLY_FILE_DIC 的消息，服务器端收到此消息后则会回复一个 type 为 AGREE_FILE_DIC 的消息，并将服务器端的共享目录下的文件加载到发送信息中。
2) 当用户在客户端选择好要上传的文件并点击开始上传之后，客户端便向服务器发送一条 type 为 APPLY_UPLOAD_FILE 的消息，并将要上传的文件名包含在消息中。服务器端收到此消息后，便在共享目录创建一个对应的文件，然后发送一条 type 为 AGREE_UPLOAD_FILE 的消息给客户端。然后客户端逐包发送带有 seq 的消息，并将 type 设置为 START_UPLOAD_FILE。服务器端收到后将收到的包的数据写入创建好的文件，并发送带 ACK 的消息，将 type 设置为 AGREE_UPLOAD_FILE。在客户端发送到最后一个数据包时，会将该条消息的结束标志位 end_flag 设置为 true。服务器端收到这条消息后将创建的文件流关闭，并向客户端回复一条 type 为 END_UPLOAD_FILE 的消息，客户端收到后，则跳出。
3) 当用户在共享目录中选择好要下载的文件，并点击开始下载按钮后，客户端便向服务器端发送一条 tpye 为 APPLY_DOWNLOAD_FILE 的消息，在这条消息中同样包含了要下载文件的文件名。当服务器端收到这条消息后，则使用文件流打开客户端想要下载的文件，并给客户端打开相应的文件流，回复一条 type 为

AGREE_DOWNLOAD_FILE 的消息，并把此消息的 seq 置为 -1。而当客户端收到 type 未 AGREE_DOWNLOAD_FILE 的消息后，客户端便开始下载工作。每次发送一条 type 为 START_DOWNLOAD_FILE 的消息给服务器端，客户端收到后回复相应的数据包，客户端收到后将其写入打开的文件流，最后结束后将文件流关闭（此过程与上传过程基本一致）。

# 第 2 章　协议的具体实现

## 2.1　客户端代码实现

### 2.1.1　总体框架

我们在窗口类初始化的时候调用另一个封装好的类 myclient, 用来绑定端口以及处理交互。在窗口类中主要负责实现点击按钮的具体功能以及调用线程持续监听消息。而在 myclient 中负责处理接收消息后的处理，以及根据接收到的消息发送对应的信息。

### 2.1.2　多线程机制

我们在窗口类初始化的时候即调用另一个线程，在另一个线程中调用 handleReceive 函数，使得 myclient 能够持续接收到消息。

```
HANDLE hThread = CreateThread(NULL, 0, handleReceive, (LPVOID)client, 0, NULL);//利用多线程
```

图 2.1: 多线程

相应代码如下：

```
//线程调用函数
DWORD WINAPI CClientDlg::handleReceive(LPVOID lpParameter)
{
        myclient* t = (myclient*)lpParameter;
        CString log;
        while(1)
        {
                log.Empty();
                t->handleInteraction();
                CClientDlg* dlg=(CClientDlg*)AfxGetApp()->GetMainWnd(); //获取主窗口
                dlg->m_log+=log;
                dlg->GetDlgItem(IDC_LOG)->SetWindowTextW(dlg->m_log);
        }
        return (DWORD)NULL;
}
```

### 2.1.3　按钮处理函数

当用户按下获取文件目录按钮时，客户端向服务器发送 type 为 APPLY_FILE_DIC 的消息，并且在 log 中显示对应信息，代码如下：

```
//点击获取共享目录按钮
void CClientDlg::OnBnClickedFileDic()
{
        mymessage send_mss;
        send_mss.type = 0;
        //sendto(c_socket,(char*)&temp,sizeof(message),0,(SOCKADDR*)&addrFrom,len);
        sendto(client->c_socket,(char*)&send_mss,sizeof(mymessage),
    0,(SOCKADDR*)&servAddr,sizeof(SOCKADDR));
        CClientDlg* dlg=(CClientDlg*)AfxGetApp()->GetMainWnd(); //获取主窗口
        dlg->m_log += L"开始交互\r\n";
        dlg->m_log+=L"发送查看共享目录请求\r\n";
        dlg->GetDlgItem(IDC_LOG)->SetWindowTextW(dlg->m_log);
        // TODO: 在此添加控件通知处理程序代码
}
```

点击开始上传按钮时，客户端用文件流打开相应文件，客户端便向服务器发送一条 type 为 APPLY_UPLOAD_FILE 的消息，并将要上传的文件名包含在消息中。并向 log 中显示对应消息。代码如下：

```
//点击开始上传按钮
void CClientDlg::OnBnClickedStartUpload()
{
        CClientDlg* dlg=(CClientDlg*)AfxGetApp()->GetMainWnd(); //获取主窗口
        dlg->m_log += L"开始交互\r\n";
        dlg->m_log+=L"发送上传文件命令\r\n";
        dlg->GetDlgItem(IDC_LOG)->SetWindowTextW(dlg->m_log);
        //用文件流打开所选择的文件
        CString upload_file = dlg->m_upload;
        CStringA file_stra(upload_file.GetBuffer(0));
        upload_file.ReleaseBuffer();
        string s = file_stra.GetBuffer(0);
        const char* file_con = s.c_str();
        if (!(client->u_file = fopen(file_con, "rb")))
        {
                AfxMessageBox(_T("创建本地文件失败！"));
        }
        mymessage send_mss;
        int j = 0;
        for(int i = 0;i<upload_file.GetLength();i++)//获取文件名
        {
                if(upload_file[i] == '\\')
                {
                        j = 0;
                }
                send_mss.file_name[j] = upload_file[i];
                j++;
        }
```

```
29          send_mss.file_name[j] = 0;
30          //设置消息类型为申请上传文件
31          send_mss.type = APPLY_UPLOAD_FILE;
32          send_mss.seq = 0;
33          sendto(client->c_socket,(char*)
34      &send_mss,sizeof(mymessage),0,(SOCKADDR*)&servAddr,sizeof(SOCKADDR));
35          if(dlg->m_upload.GetLength()==0)//如果文件选择栏为空，则弹出对话框
36          {
37                  MessageBox(_T("上传文件为空！"));
38          }
39          dlg->GetDlgItem(IDC_LOG)->SetWindowTextW(dlg->m_log);
40          // TODO: 在此添加控件通知处理程序代码
41  }
```

当用户点击开始下载按钮后，客户端便向服务器端发送一条 tpye 为 APPLY_DOWNLOAD_FILE 的消息，在这条消息中同样包含了要下载文件的文件名，并且 log 中显示相应消息，代码如下：

```
1   //点击开始下载按钮
2   void CClientDlg::OnBnClickedStartDownload()
3   {
4           CClientDlg* dlg=(CClientDlg*)AfxGetApp()->GetMainWnd(); //获取主窗口
5           dlg->m_log += L"开始交互\r\n";
6           dlg->m_log+=L"发送下载文件命令\r\n";
7           mymessage send_mss;
8           //设置消息类型为申请下载
9           send_mss.type = APPLY_DOWNLOAD_FILE;
10          //获取文件名
11          for(int i=0;i<m_download.GetLength();i++)
12          {
13                  send_mss.file_name[i] = m_download[i];
14          }
15          send_mss.file_name[m_download.GetLength()]=0;
16          if(dlg->m_download.GetLength()==0)//如果文件选择栏为空，则弹出对话框
17          {
18                  MessageBox(_T("下载文件为空！"));
19          }
20          char path[MAX_PATH + 1] = { 0 };
21          GetModuleFileNameA(NULL, path, MAX_PATH);
22          (strrchr(path, '\\'))[0] = 0; // 删除文件名，只获得路径字串
23          client->full_ACK = 0;
24          strcat(path,"\\");
25          client->d_file = fopen(strcat(path,send_mss.file_name),"wb");
26          sendto(client->c_socket,(char*)
27      &send_mss,sizeof(mymessage),0,(SOCKADDR*)&servAddr,sizeof(SOCKADDR));
28          dlg->GetDlgItem(IDC_LOG)->SetWindowTextW(dlg->m_log);
29  }
```

### 2.1.4 交互的实现

当客户端收到 type 为 AGREE_FILE_DIC 的消息时，客户端便将收到的数据（即共享文件目录）显示在相应窗口中，并在 log 中添加相应消息。代码如下：

```
//服务器同意上传文件
case AGREE_UPLOAD_FILE:
{
        mymessage send_mss;
        //用文件流读文件文件
        int batch_num = fread(send_mss.mss_buffer,1, BUF_SIZE , u_file);
        send_mss.type = START_UPLOAD_FILE;
        int j = 0;
        dlg->m_log += L"上传文件中\r\n";
        CString upload_file = dlg->m_upload;
        send_mss.len = batch_num;
        send_mss.seq = recv.ACK;
        //读出内容长度小于，则证明上传完成BUF_SIZE
        if(batch_num < BUF_SIZE)
        {
                dlg->m_log += L"上传文件完成\r\n";
                send_mss.end_flaaaaaaa[P-=ag = true;
        }
        else
        {
                send_mss.end_flag = false;
        }
        sendto(c_socket,(char*)&send_mss, sizeof(mymessage), 0,
                                (SOCKADDR*)&addrFrom, sizeof(SOCKADDR));
        dlg->GetDlgItem(IDC_LOG)->SetWindowTextW(dlg->m_log);

        return;
}
```

当客户端收到 type 为 AGREE_UPLOAD_FILE 的消息时，客户端逐包发送带有 seq 的消息，并将 type 设置为 START_UPLOAD_FILE。相应代码如下：

```
//服务器同意上传文件
case AGREE_UPLOAD_FILE:
{
        mymessage send_mss;
        //用文件流读文件文件
        int batch_num = fread(send_mss.mss_buffer,1, BUF_SIZE , u_file);
        send_mss.type = START_UPLOAD_FILE;
        int j = 0;
        dlg->m_log += L"上传文件中\r\n";
        CString upload_file = dlg->m_upload;
```

```
11          send_mss.len = batch_num;
12          send_mss.seq = recv.ACK;
13          //读出内容长度小于，则证明上传完成BUF_SIZE
14          if(batch_num < BUF_SIZE)
15          {
16                  dlg->m_log += L"上传文件完成\r\n";
17                  send_mss.end_flag = true;
18          }
19          else
20          {
21                  send_mss.end_flag = false;
22          }
23          sendto(c_socket,(char*)&send_mss, sizeof(mymessage), 0,
24                                  (SOCKADDR*)&addrFrom, sizeof(SOCKADDR));
25          dlg->GetDlgItem(IDC_LOG)->SetWindowTextW(dlg->m_log);
26
27          return;
28  }
```

当客户端收到 type 为 END_UPLOAD_FILE 的消息时，直接跳出。代码如下：

```
1   //上传完成，则跳出
2   case END_UPLOAD_FILE:
3   {
4
5           c_posting = false;
6           return;
7   }
```

当客户端收到 type 为 AGREE_DOWNLOAD_FILE 的消息时，客户端便开始下载工作。代码如下：

```
1   //服务器同意下载文件
2   case AGREE_DOWNLOAD_FILE:
3   {
4           if(recv.seq == -1)
5           {
6                   dlg->m_log += L"开始下载文件\r\n";
7           }
8           mymessage send_mss;
9           send_mss.type = START_DOWNLOAD_FILE;
10          //与相等，则与相加后继续发送ACKseqACKlen
11          if(full_ACK == recv.seq)
12          {
13                  fwrite(recv.mss_buffer,recv.len,1,d_file);
14                  dlg->m_log += L"下载文件中\r\n";
15                  send_mss.type = AGREE_UPLOAD_FILE;
```

```
16                  full_ACK = send_mss.ACK = full_ACK + recv.len;
17          }
18          else
19          {
20                  send_mss.ACK = full_ACK;
21          }
22          //结束为为且不处于开始发送状态，则关闭文件，下载完成true
23          if(recv.end_flag && recv.seq!=-1)
24          {
25                  fclose(d_file);
26                  send_mss.type = END_DOWNLOAD_FILE;
27                  dlg->m_log += L"下载文件完成\r\n";
28          }
29          else
30          {
31                  send_mss.type = START_DOWNLOAD_FILE;
32          }
33          sendto(c_socket,(char*)&send_mss,sizeof(mymessage),
34      0,(SOCKADDR*)&addrFrom,sizeof(SOCKADDR));
35          return;
36  }
```

## 2.2 服务器端代码实现

### 2.2.1 总体框架

在服务器端我们采用同样的框架，在窗口类初始化的时候调用 myserver 类使其绑定相应端口并处理交互数据。利用与客户端一样的多线程机制使其能够持续监听消息。

### 2.2.2 交互的实现

当服务器端收到 type 为 APPLY_FILE_DIC 的消息时，之后服务器端查看共享目录的文件列表，将其放在发送报文中，并将其 type 设置为 AGREE_FILE_DIC，并将其发送给客户端。代码如下：

```
1   //客户端申请访问共享目录
2   case APPLY_FILE_DIC:
3   {
4           CServerDlg* dlg=(CServerDlg*)AfxGetApp()->GetMainWnd(); //获取主窗口
5           dlg->m_log+=L"收到请求共享目录消息.\r\n";
6           send_mss.type = AGREE_FILE_DIC;
7           for (int j = 1; j < BUF_SIZE; j++)
8           {
9                   send_mss.mss_buffer[j] = 0;
10          }
11          char szFilePath[MAX_PATH + 1] = { 0 };
```

```
12          GetModuleFileNameA(NULL, szFilePath, MAX_PATH);
13          (strrchr(szFilePath, '\\'))[0] = 0; // 删除文件名，只获得路径字串
14          string p;
15          //文件句柄
16          long    hFile = 0;
17          //文件信息
18          struct _finddata_t fileinfo;
19          if ((hFile = _findfirst(p.assign(szFilePath)
20      .append("\\*").c_str(), &fileinfo)) != -1)
21          {
22                  do
23                  {
24                          strncat(send_mss.mss_buffer,
25          strncat(fileinfo.name,"\r\n",5), BUF_SIZE);
26                  } while (_findnext(hFile, &fileinfo) == 0);
27                  _findclose(hFile);
28          }
29          send_mss.mss_buffer[BUF_SIZE-1] = '\n';
30          //displayShared(sendBuf);
31          sendto(s_socket, (char*)&send_mss, sizeof(mymessage),
32      0,(SOCKADDR*)&clntAddr, sizeof(SOCKADDR));
33          end = true;
34          dlg->m_log+=L"发送共享文件目录\r\n";
35          break;
36  }
```

当服务器端收到 type 为 APPLY_UPLOAD_FILE 的消息时，客户端便根据发来消息的文件名创建文件，并把回复信息的 type 设置为 AGREE_UPLOAD_FILE。代码如下：

```
1   //客户端申请上传文件
2   case APPLY_UPLOAD_FILE:
3   {
4           CServerDlg* dlg=(CServerDlg*)AfxGetApp()->GetMainWnd(); //获取主窗口
5           dlg->m_log+=L"收到请求上传文件消息.\r\n";
6           //设置发送信息的类型为AGREE_UPLOAD_FILE
7           mymessage send_mss;
8           send_mss.type = AGREE_UPLOAD_FILE;
9           send_mss.ACK = 0;
10          sendto(s_socket, (char*)&send_mss, strLen, 0,
11      (SOCKADDR*)&clntAddr, sizeof(SOCKADDR));
12          char path[MAX_PATH + 1] = { 0 };
13          GetModuleFileNameA(NULL, path, MAX_PATH);
14          (strrchr(path, '\\'))[0] = 0; // 删除文件名，只获得路径字串
15          full_ACK = 0;
16          u_file = fopen(strcat(path,recv_mss.file_name),"wb");
17          end = true;
```

```
18          dlg->m_log+=L"发送同意上传文件消息\r\n";
19          break;
20  }
```

当服务器端收到 type 为 START_UPLOAD_FILE 的消息时，客户端向已创建的文件中写入发送消息包含的文件数据，并根据发过来的 seq 回复对应的 ACK，代码如下：

```
1   //客户端开始上传文件
2   case START_UPLOAD_FILE:
3   {
4           CServerDlg* dlg=(CServerDlg*)AfxGetApp()->GetMainWnd(); //获取主窗口
5           //发送的与初始相等，则继续发送ACKseq
6           if(recv_mss.seq == full_ACK)
7           {
8                   fwrite(recv_mss.mss_buffer,recv_mss.len,1,u_file);
9                   dlg->m_log += L"接收上传文件中\r\n";
10                  send_mss.type = AGREE_UPLOAD_FILE;
11                  full_ACK = send_mss.ACK = full_ACK + recv_mss.len;
12          }
13          //如果不相等，则继续发送上一个ACK
14          else
15          {
16                  send_mss.ACK = full_ACK;
17          }
18          if(recv_mss.end_flag)
19          {
20                  fclose(u_file);
21                  send_mss.type = END_UPLOAD_FILE;
22                  dlg->m_log += L"接收上传文件完成\r\n";
23          }
24          else
25          {
26                  send_mss.type = AGREE_UPLOAD_FILE;
27          }
28          sendto(s_socket, (char*)&send_mss,
29      sizeof(mymessage), 0,(SOCKADDR*)&clntAddr, sizeof(SOCKADDR));
30          end = true;
31          break;
32  }
```

当服务器端收到 type 为 APPLY_DOWNLOAD_FILE 的消息时，打开发送消息所包含的文件名，并将回复信息的 type 设置为 AGREE_DOWNLOAD_FILE，代码如下：

```
//客户端申请下载文件
case APPLY_DOWNLOAD_FILE:
{
        CServerDlg* dlg=(CServerDlg*)AfxGetApp()->GetMainWnd(); //获取主窗口
        dlg->m_log+=L"收到请求下载文件消息.\r\n";
        mymessage send_mss;
        send_mss.type = AGREE_DOWNLOAD_FILE;
        send_mss.seq = -1;
        sendto(s_socket, (char*)&send_mss, sizeof(mymessage),
    0,(SOCKADDR*)&clntAddr, sizeof(SOCKADDR));
        dlg->m_log+=L"发送同意下载文件消息.\r\n";
        //dlg->m_log+=CString(recv_mss.mss_buffer);
        char path[MAX_PATH + 1] = { 0 };
        GetModuleFileNameA(NULL, path, MAX_PATH);
        (strrchr(path, '\\'))[0] = 0; // 删除文件名，只获得路径字串
        full_ACK = 0;
        strcat(path,"\\");
        //用文件流打开文件
        d_file = fopen(strcat(path,recv_mss.file_name),"rb");
        end = true;
        break;

}
```

当服务器端收到 type 为 START_DOWNLOAD_FILE 的消息时，服务器读取之前打开的的文件流，并将其放到发送的消息中，并根据 ACK 发送 seq，当读取文件结束时，将结束标志位置 1，代码如下：

```
//客户端开始下载文件
case START_DOWNLOAD_FILE:
{
        mymessage send_mss;
        int batch_num = fread(send_mss.mss_buffer,1, BUF_SIZE , d_file);
        CServerDlg* dlg=(CServerDlg*)AfxGetApp()->GetMainWnd(); //获取主窗口
        dlg->m_log+=L"开始下载文件.\r\n";
        send_mss.len = batch_num;
        send_mss.seq = recv_mss.ACK;
        //如果文件流读出来的大小小于，则证明下载文件完毕，设置结束标志位。BUF_SIZE
        if(batch_num < BUF_SIZE)
        {
                dlg->m_log += L"下载文件完成\r\n";
                send_mss.end_flag = true;
        }
        else
        {
                send_mss.end_flag = false;
        }
```

```
20          send_mss.type = AGREE_DOWNLOAD_FILE;
21          sendto(s_socket,(char*)&send_mss, sizeof(mymessage), 0,
22                                  (SOCKADDR*)&clntAddr, sizeof(SOCKADDR));
23          end = true;
24          break;
25  }
```

当服务器端收到 type 为 END_DOWNLOAD_FILE 的消息时，直接跳出。代码如下：

```
1  //客户端下载结束
2  case END_DOWNLOAD_FILE:
3  {
4          break;
5  }
```

## 2.3 可靠性的保证

首先我们模仿 FTP 设置了 seq 和 ACK 值只有两者相对应时才能进行传输，从而保证了没有乱序。此外，我们设置了超时重传机制，将超时时间设置为 2s，从而超时时，将上一次数据重新发送，代码如下：

```
1  //设置超时重传
2  int timeout = 2000;        //设置超时时间为2s
3  setsockopt(s_socket,SOL_SOCKET,SO_RCVTIMEO,(const char*)&timeout,sizeof(int));
4  recv_mss.type = -1;
```

当超时时，上一个要发送的数据则被重新发送。

# 第 3 章　实验结果展示

## 3.1　客户端和服务器端的初始界面

运行客户和服务器两个工程，我们便得到了初始界面，如下：

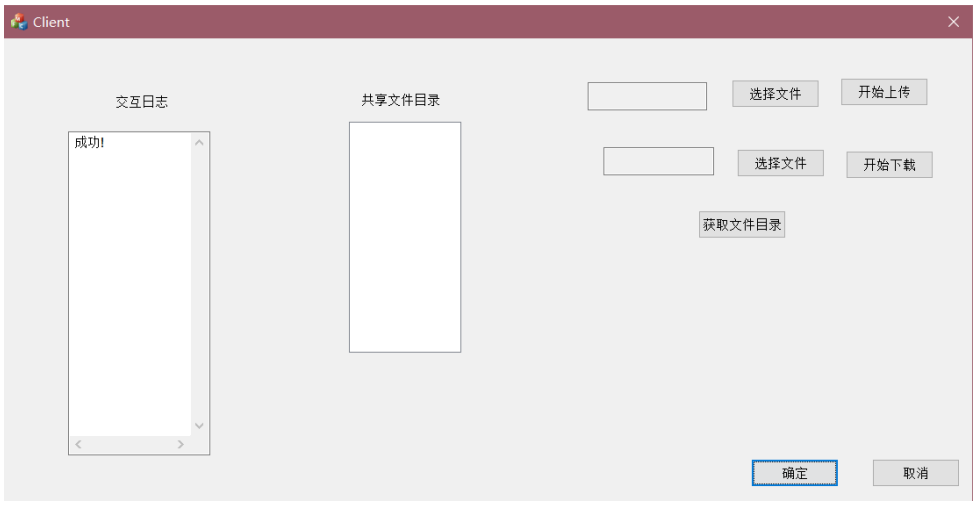


图 3.1: 客户端初始界面

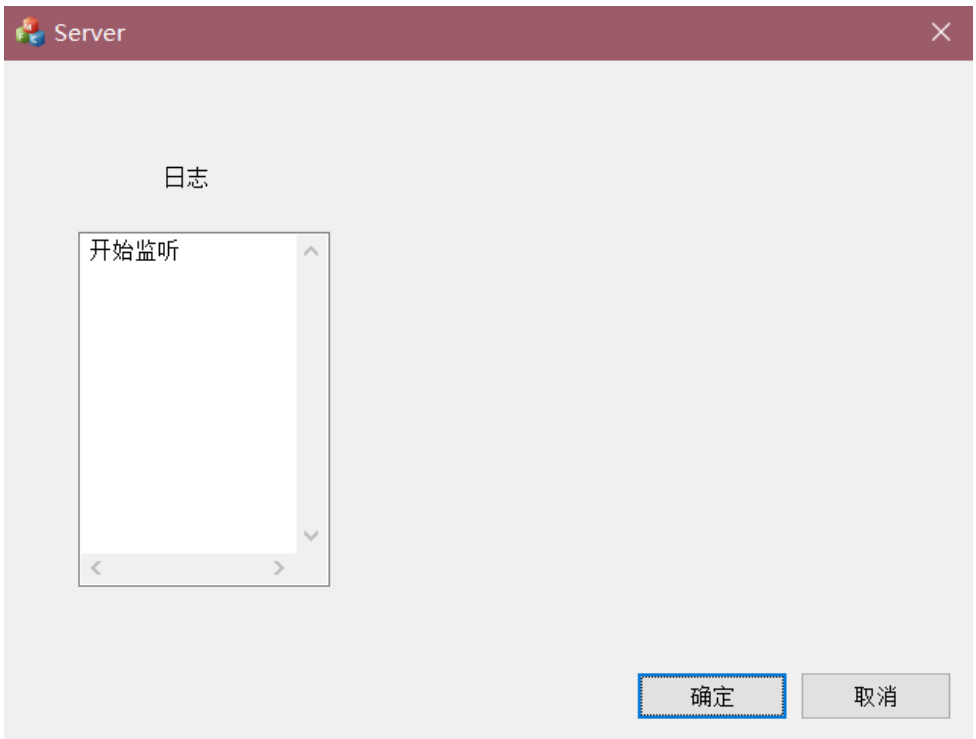


图 3.2: 服务器端端初始界面

## 3.2 共享目录的获取

我们在客户端点击获取共享目录按钮，便可以获得服务器端的共享目录，两端变化如下：

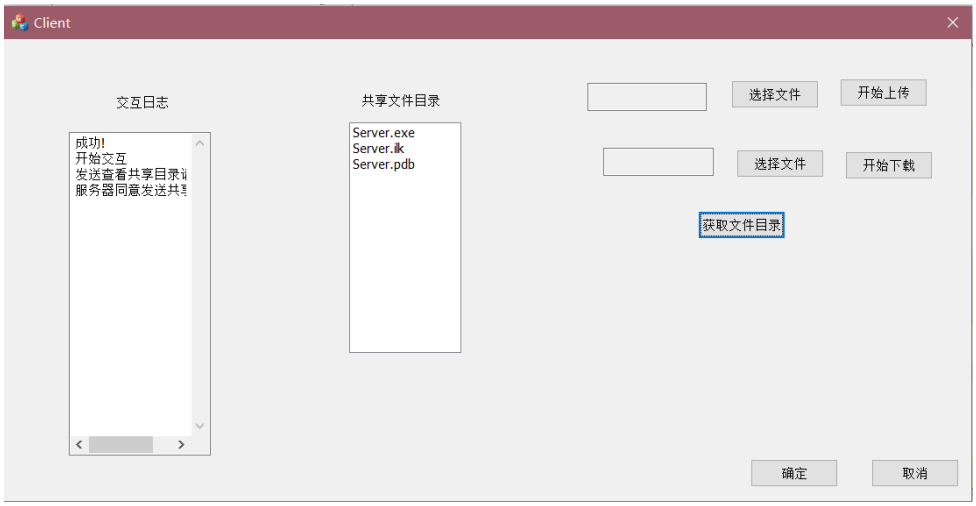


图 3.3: 客户端获取共享目录

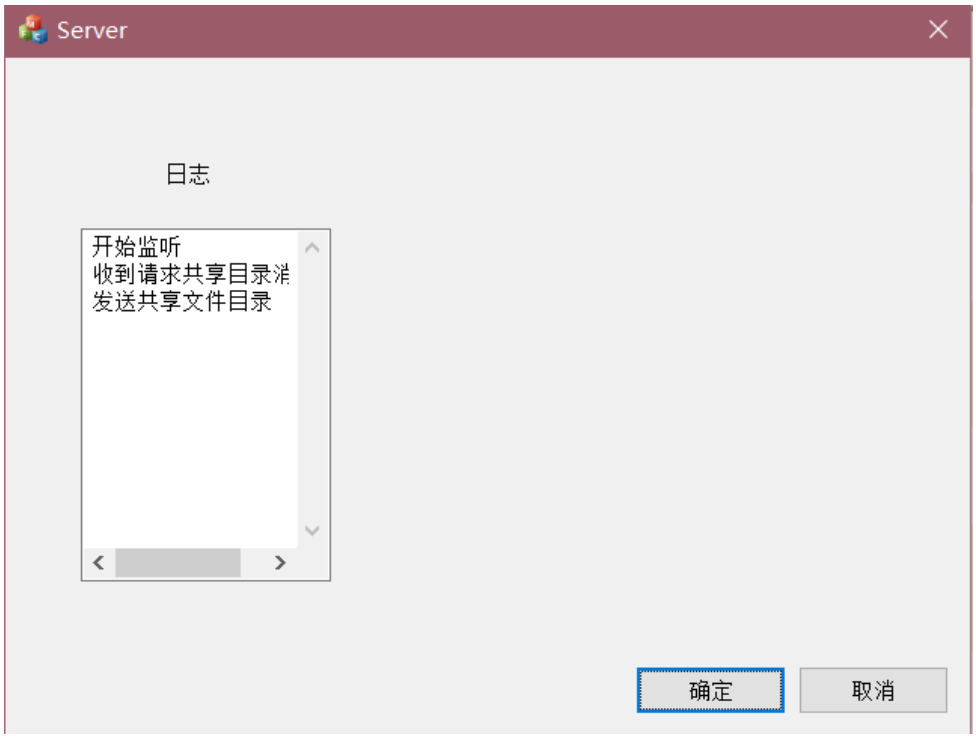


图 3.4: 服务器端获取共享目录

## 3.3 客户端上传文件

我们选择上传文件后点击“开始上传”按钮，我们便可以进行文件的上传了


图 3.5: 客户端上传前

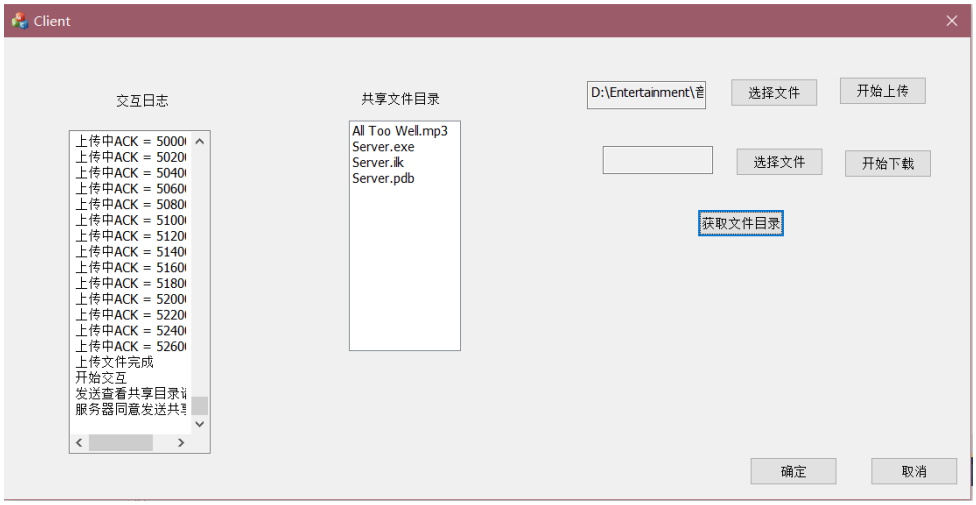


图 3.6: 客户端上传完成

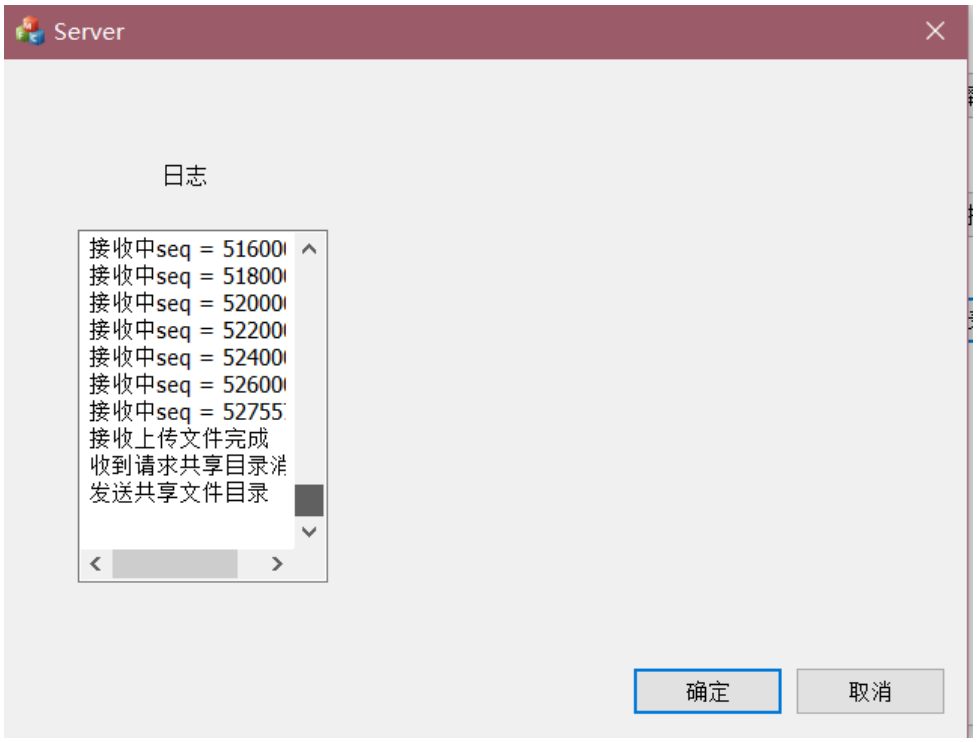


图 3.7: 服务器端接收上传完成

可以看到，共享目录增加了上传的文件，实验成功。

## 3.4 客户端下载文件

我们在客户端的共享目录中选择要下载的文件，然后点击“开始下载”按钮后，我们便可以进行文件的下载了。

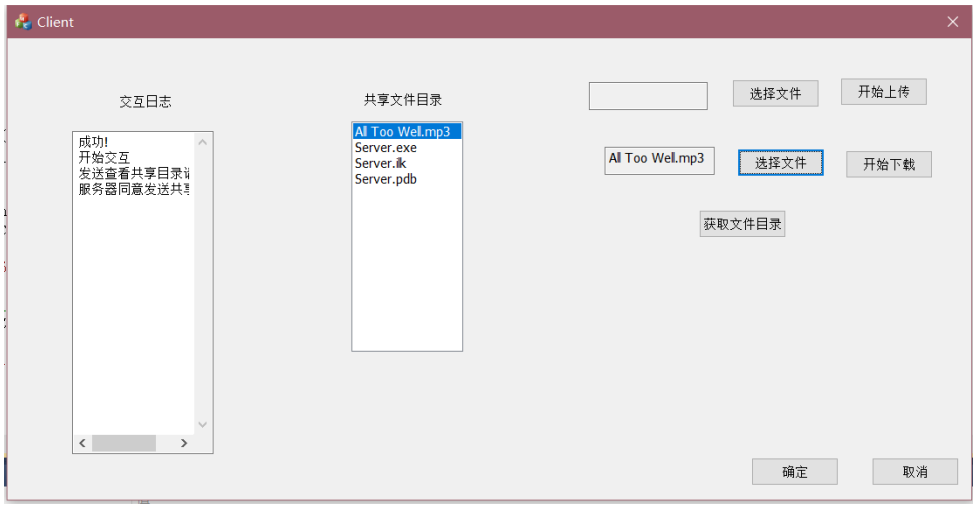


图 3.8: 客户端下载前

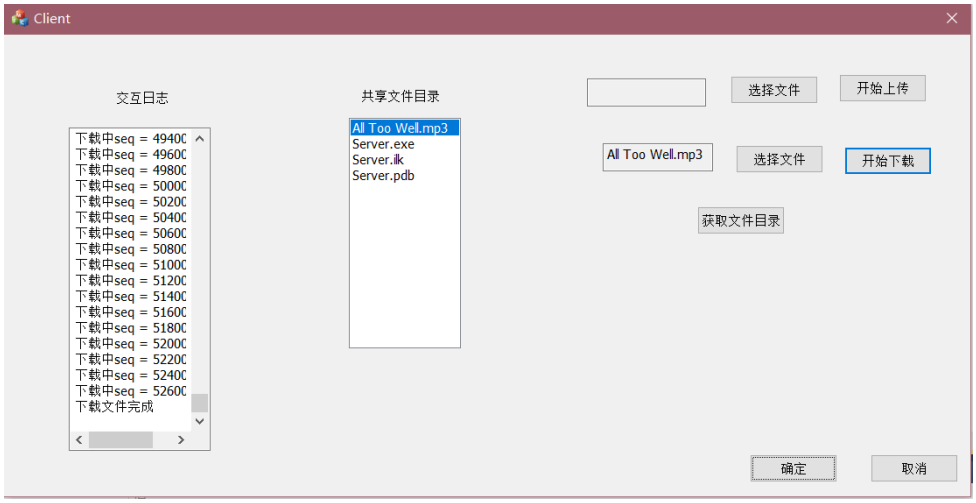


图 3.9: 客户端下载完成后

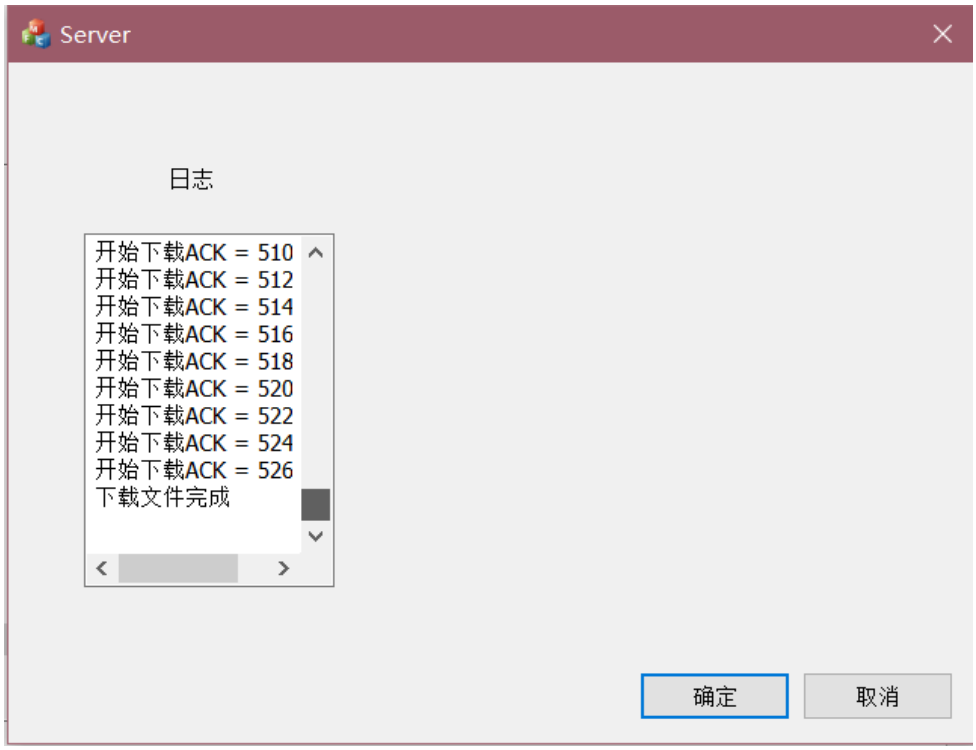


图 3.10: 服务器端下载完成后

我们在客户端文件夹中可以看到获取了正确的文件，实验成功。

## 3.5 多用户

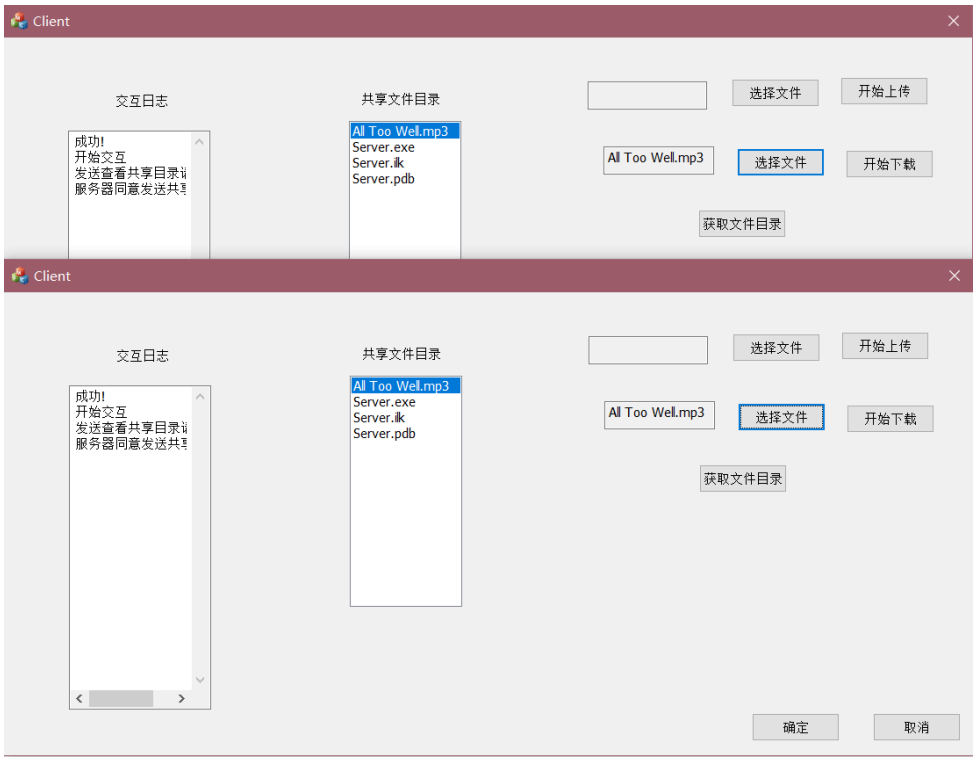


图 3.11: 多用户下载前

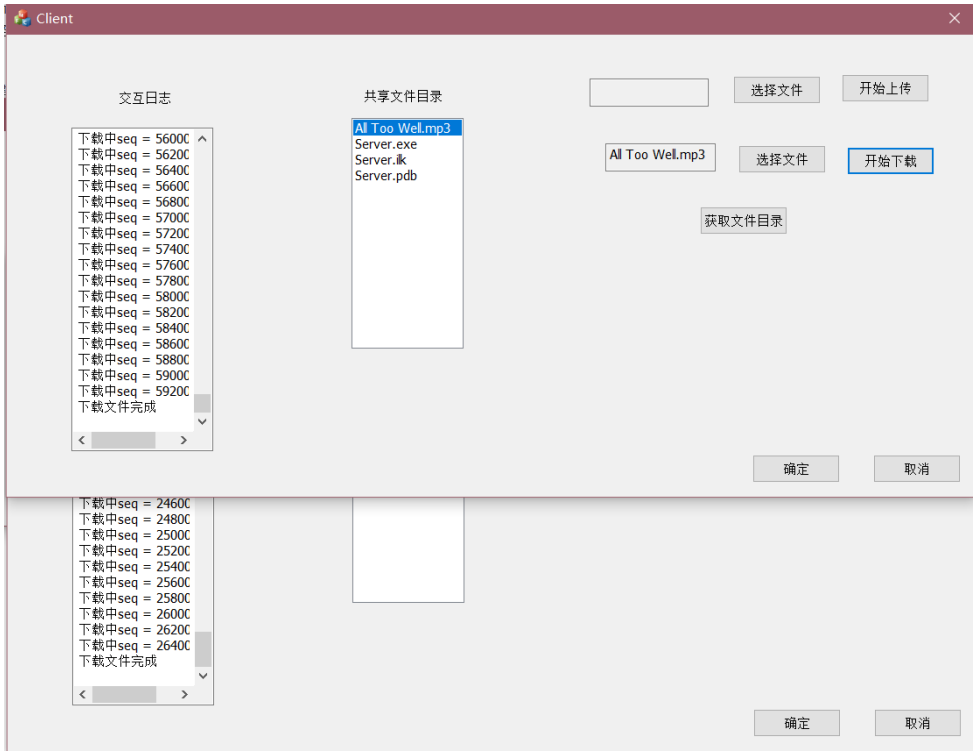


图 3.12: 多用户下载完成后

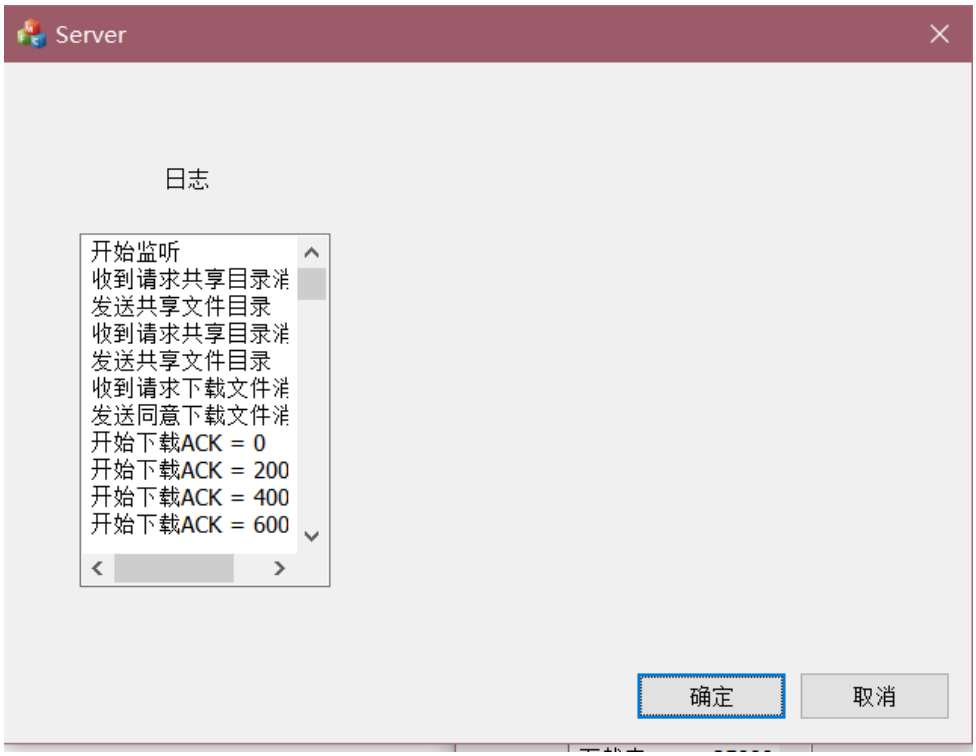


图 3.13: 多用户服务器端下载完成后

可以看到在服务器端显示了两条下载完成记录，两个客户端也正确完成了下载。