



网络应用程序设计

模拟 FTP 服务系统

姓名: xxx

学号: xxx

2021 年 6 月 29 日

目录

1	实验内容	1
2	实验环境	1
3	内容分析	1
4	具体实现	1
4.1	文件清单	1
4.2	准备工作	2
4.2.1	TCP 消息的读取	2
4.2.2	密钥分发算法	3
4.2.3	加解密算法	3
4.3	FTP 功能	3
4.3.1	列出所有可访问文件	4
4.3.2	删除文件	4
4.3.3	上传文件	4
4.3.4	下载文件	5
5	结果展示	5
6	局限与展望	6

1 实验内容

本次实验中，我们在真实的网络环境下搭建安全的文件传输系统，使得客户端与服务器之间能够进行任意类型文件的安全传输。

2 实验环境

客户机使用 Ubuntu 20.04.2 LTS 操作系统，服务器使用 Ubuntu 18.04.5 LTS 操作系统。

3 内容分析

考虑到文件传输需要高可靠性，本次实验中我们使用 `socket/tcp` 完成数据传输。

双方首先通过 `socket` 套接字建立连接，随后通过对称密钥分发办法（如 DH、Elgalma 等）分享会话密钥，此后的即可使用该密钥对文件进行加解密。

服务器支持如下四种操作：

- 列出当前目录下所有文件的详细信息
- 删除当前目录下某个文件
- 上传某文件到服务器
- 下载某文件到客户端

4 具体实现

4.1 文件清单

下面给出所有文件及文件夹的说明。

- | | |
|---|---|
| 1 | <code>Makefile</code> : <code>makefile</code> 文件 |
| 2 | <code>rqdmap.h</code> : 个人编程时惯用的一些宏定义 |
| 3 | <code>ftp.h</code> , <code>ftp.cpp</code> : 定义了一些便于SOCKET通讯的结构体、函数等 |

```
4 des/: 文件夹内包含了DES实现的源码
5 run_des.o: 封装了DES加解密的接口
6 server.cpp, server: 服务器源码及主程序
7 client.cpp, client: 客户端源码及主程序
8 pic.gif: 测试用例, 检验图片是否能完好的传输
```

4.2 准备工作

4.2.1 TCP 消息的读取

TCP 是面向字节流的协议, 因而在接下来的设计中我们大量使用

{报文类型, 32 位整数 N, N 字节数据流}

的形式来传输数据。而实际网络中存在各种情况, 因而有可能存在这样的情形: 消息接收方一次读取消息后得到了即将应该接收的字节数 N , 但是暂时只收集到了不足 n 的字节流, 此时接收方应该不断循环读取, 直到读到的字节数不小于 N 才停止。为此, 我们实现了一个基于**循环队列**的消息队列, 用于不断地读取消息, 并能反馈出目前读取字节的长度。下面给出该消息队列的接口信息:

```
1 struct MQ{
2     unsigned char queue[M + 10];
3     int head, tail;
4     int sockfd;
5
6     void init(int sockfd); //使用连接套接字初始化该消息队列
7     void recv(); //读取一次不超过剩余空间的TCP消息
8     int size(); //返回队列中字节数
9     bool empty(); //返回是否为空
10    unsigned char front(); //返回队头元素
11    void pop(); //弹出队头元素
12    void prt(); //【调试用】顺序打印队列的所有元素值
13 };
```

4.2.2 密钥分发算法

这里我们使用 DH 算法分发对称加密密钥，为此需要实现快速幂算法来进行 $O(\log N)$ 幂次计算。

接口信息如下：

```
1 //return a^p % M
2 unsigned long long quick_pow(ull a, ull p, const ull M);
```

4.2.3 加解密算法

考虑到本次实验仅仅是对安全 FTP 系统的一次简单模拟，故使用较为简单的 DES 作为对称加密算法以说明原理。加解密算法单独成为一套接口 `run_des.o`，可以通过如下方式进行调用：

```
1 ./run_des.o -g <OutKeyFile> //产生密钥，存放在OutKeyFile中
2 ./run_des.o -e <KeyFile> <From> <To> //使用KeyFile加密From文件，产出到To中
3 ./run_des.o -d <KeyFile> <From> <To> //使用KeyFile解密From文件，产出到To中
```

4.3 FTP 功能

FTP 功能一共有四个：列出所有可访问文件、删除文件、上传文件、下载文件，下面将逐一进行说明。

首先给出通讯过程中用到的功能关键字的宏定义：

```
1 #define LIST 0
2 #define UPLOAD 1
3 #define DOWNLOAD 2
4 #define CONTENT 3
5 #define ACK 4
6 #define NAK 5
7 #define FIN 6
8 #define DELETE 7
```

4.3.1 列出所有可访问文件

客户端首先发送功能关键字 LIST 进行请求，服务器端接到后使用系统调用 `system("ls -al")`，读取调用结果，并将结果返回给客户端。客户端根据数据流的格式解析出数据内容，将结果显示在控制台。

双方通讯中字节流的交互流程如下所示：

Client 发送: `LIST`

Server 发送: `CONTENT + {数据内容} + FIN`

Client 本地显示数据内容

4.3.2 删除文件

客户端发送功能关键字 DELETE 和想要删除的文件名给服务器，服务器接收到后首先进行合法性检验，确保用户删除的文件存在且不得超出当前文件夹范围，随后再进行文件删除，返回状态给客户。

双方通讯中字节流的交互流程如下所示：

Client: `DELETE + {文件名}`

Server: `ACK/NAK`

ACK 表示删除成功，NAK 表示删除失败。

4.3.3 上传文件

客户端发送功能关键字 UPLOAD 和想要上传的文件名及其大小，服务端进行文件创建并开辟空间，随后客户端再传输具体的字节流，服务端接受字节流并写入文件。

双方通讯中字节流的交互流程如下所示：

Client 发送: `UPLOAD {文件名}`

Server 在本地创建该文件，并反馈: `ACK/NAK`

如果接收到 `ACK`，Client 发送: `{文件大小}`

Server 在本地为该文件开辟空间，并反馈: `ACK/NAK`

如果接收到 `ACK`，Client 加密原始文件，发送: `{密文文件数据流} + FIN`

Server 不断读密文数据并保存到文件中，解密密文文件，如果成功则反馈: `ACK`

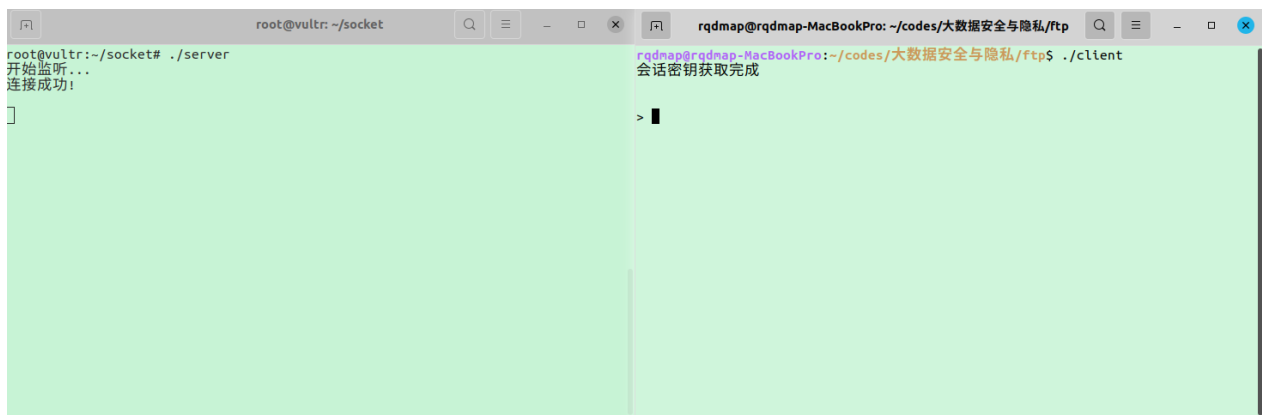
Client 接收: `ACK`, 认为文件传输成功。

4.3.4 下载文件

下载文件与上传文件类似。客户发送想要下载的文件名，服务端检查该文件是否存在，如果存在，则使用与上传文件类似的交互方式完成文件传输，便不予赘述。

5 结果展示

运行两个程序 (*client*, *server*), 双方连接成功, 交换密钥成功。



```
root@vultr: ~/socket
root@vultr:~/socket# ./server
开始监听...
连接成功!

rqdmap@rqdmap-MacBookPro: ~/codes/大数据安全与隐私/ftp
rqdmap@rqdmap-MacBookPro:~/codes/大数据安全与隐私/ftp$ ./client
会话密钥获取完成

> █
```

客户端使用 *ls* 指令查看服务器上所有可供访问的资源。



```
root@vultr: ~/socket
root@vultr:~/socket# ./server
开始监听...
连接成功!

就绪中...
读取到 buf.op: 0
List response finished.

rqdmap@rqdmap-MacBookPro: ~/codes/大数据安全与隐私/ftp
rqdmap@rqdmap-MacBookPro:~/codes/大数据安全与隐私/ftp$ ./client
会话密钥获取完成

> ls

total 100
drwxr-xr-x  2 root root  4096 Jun 13 17:38 .
drwxr-xr-x 10 root root  4096 Jun 13 17:38 ..
-rw-r--r--  1 root root  4257 Apr  9 15:35 ftp.h
-rw-r--r--  1 root root   19 Jun 13 19:31 key
-rw-r--r--  1 root root   686 Apr  9 15:35 rqdmap.h
-rwxr-xr-x  1 root root   44 Apr  8 13:05 run.sh
-rwxr-xr-x  1 root root 25904 Apr  9 15:35 run_des.o
-rwxr-xr-x  1 root root 24384 Apr  9 15:39 server
-rw-r--r--  1 root root 12800 Apr  9 15:35 server.cpp
-rw-r--r--  1 root root    8 Jun 13 19:31 std.key

> █
```

客户端上传本地的一张图片 *pic.gif*, 上传完成后显示花费时间。

```
root@vultr:~/socket# ./server
开始监听...
连接成功!

就绪中...
读取到 buf.op: 0
List response finished.

就绪中...
读取到 buf.op: 1
即将上传文件 pic.gif
文件大小 51869 bytes

Upload finished.

> ls
total 100
drwxr-xr-x  2 root root 4096 Jun 13 17:38 .
drwx----- 10 root root 4096 Jun 13 17:38 ..
-rw-r--r--  1 root root 4257 Apr  9 15:35 ftp.h
-rw-r--r--  1 root root  19 Jun 13 19:31 key
-rw-r--r--  1 root root  686 Apr  9 15:35 rqdmap.h
-rwxr-xr-x  1 root root  44 Apr  8 13:05 run.sh
-rwxr-xr-x  1 root root 25904 Apr  9 15:35 run_des.o
-rwxr-xr-x  1 root root 24384 Apr  9 15:39 server
-rw-r--r--  1 root root 12800 Apr  9 15:35 server.cpp
-rw-r--r--  1 root root  8 Jun 13 19:31 std.key

> up pic.gif
将要上传 51869 bytes
已经传输 51869 bytes

上传完成
总共上传耗时 1.25893 s

>
```

客户端使用 *ls* 指令检查是否成功上传图片，并使用 *rm* 指令删除该图片。

```
就绪中...
读取到 buf.op: 1
即将上传文件 pic.gif
文件大小 51869 bytes

Upload finished.

就绪中...
读取到 buf.op: 0
List response finished.

就绪中...
读取到 buf.op: 7
Remove response finished.

> ls
Assert of line 107 failed.
total 152
drwxr-xr-x  2 root root 4096 Jun 13 19:32 .
drwx----- 10 root root 4096 Jun 13 17:38 ..
-rw-r--r--  1 root root 4257 Apr  9 15:35 ftp.h
-rw-r--r--  1 root root  19 Jun 13 19:31 key
-rw-r--r--  1 root root 51869 Jun 13 19:32 pic.gif
-rw-r--r--  1 root root  686 Apr  9 15:35 rqdmap.h
-rwxr-xr-x  1 root root  44 Apr  8 13:05 run.sh
-rwxr-xr-x  1 root root 25904 Apr  9 15:35 run_des.o
-rwxr-xr-x  1 root root 24384 Apr  9 15:39 server
-rw-r--r--  1 root root 12800 Apr  9 15:35 server.cpp
-rw-r--r--  1 root root  8 Jun 13 19:31 std.key

> rm pic.gif
删除完成.

>
```

6 局限与展望

- FTP 服务器暂时不支持文件夹操作
- 不同用户之间没有进行认证与分类，全部共用一套公用的文件夹
- 加密算法基于文件而不是对字节流进行加密，会导致部分操作码的明文泄漏，有一定的安全问题
- 尚未实现多用户功能。由于多用户涉及到消息队列的异步读取、父子进程的协调工作等，本次实验时间较紧，故暂不支持多用户系统。