# 文件传输应用 (基于停等协议)

## 实验简介

本实验是Lab2-1的选做内容2,基于停等协议实现了一个完整的C/S结构文件传输应用,支持可靠的文件上传功能。

## 系统架构

客户端 服务器  
 | |  
 |------ 文件信息帧 ----------->|  
 |<------- ACK -----------------|  
 | |  
 |------ 数据帧1 (Seq=0) ------>|  
 |<------- ACK(0) --------------|  
 | |  
 |------ 数据帧2 (Seq=1) ------>|  
 |<------- ACK(1) --------------|  
 | |  
 | ... |  
 | |  
 |------ 数据帧N (结束) -------->|  
 |<------- ACK(N) --------------|  
 | |

## 协议设计

### 数据帧格式

struct DataFrame {  
 unsigned char seq; // 序列号 (0或1)  
 unsigned int fileSize; // 文件总大小(仅首帧)  
 unsigned int offset; // 数据在文件中的偏移  
 unsigned short dataLen; // 当前帧数据长度  
 char data[1020]; // 数据内容  
 unsigned char flag; // 帧类型标志  
};

### 帧类型

* **flag = 0**: 普通数据帧
* **flag = 1**: 最后一个数据帧
* **flag = 2**: ACK确认帧
* **flag = 3**: 文件列表帧(保留)
* **flag = 4**: 文件信息帧(传输开始前)

### 传输流程

1. **建立连接**
   * 客户端发送文件信息帧(文件名、大小)
   * 服务器发送ACK确认
2. **数据传输**
   * 客户端按DATA\_SIZE(1020字节)分割文件
   * 每个数据帧包含：序列号、偏移、长度、数据
   * 使用停等协议逐帧传输
3. **可靠传输**
   * 每发送一帧等待ACK
   * 超时未收到ACK则重传(最多5次)
   * 序列号0和1交替使用
4. **传输完成**
   * 最后一帧设置flag=1
   * 服务器收到后返回ACK
   * 显示传输统计信息

## 编译说明

cd Lab2-1/file\_transfer  
  
# 编译服务器  
g++ server.cpp -o server.exe -lws2\_32  
  
# 编译客户端  
g++ client.cpp -o client.exe -lws2\_32

## 运行说明

### 1. 启动服务器

./server.exe

服务器启动后:

* 监听端口12340
* 自动创建received\_files目录存放接收的文件
* 等待客户端连接

========================================  
文件传输服务器(基于停等协议)  
监听端口: 12340  
数据帧大小: 1020 字节  
========================================  
  
等待客户端请求...

### 2. 启动客户端

./client.exe

### 3. 客户端命令

#### 上传文件

-upload <文件路径> [丢包率]

示例:

请输入命令: -upload test\_file.txt 0.2

参数说明:

* <文件路径>: 要上传的文件路径(相对或绝对)
* [丢包率]: 可选,模拟丢包率(0.0-1.0),默认0.0

#### 其他命令

* -time: 获取服务器时间
* -quit: 退出客户端

## 测试示例

### 测试1: 正常文件传输(无丢包)

**客户端**:

请输入命令: -upload test\_file.txt 0

**预期输出**:

[上传] 文件: test\_file.txt  
大小: 567 字节  
丢包率: 0%  
========================================  
[发送] 文件信息帧  
[确认] 服务器准备接收  
[发送] 包#1, Seq=0, 偏移=0, 大小=567 字节, 尝试=1  
[ACK] 收到ACK=0  
[成功] ACK正确  
[进度] 567/567 (100%)  
  
========================================  
[完成] 文件上传成功!  
总大小: 567 字节  
总包数: 1  
========================================

**服务器输出**:

[请求] 来自 127.0.0.1 - 上传文件  
文件名: test\_file.txt  
文件大小: 567 字节  
[接收] 开始接收文件...  
保存路径: received\_files\test\_file.txt  
[信息] 文件大小: 567 字节  
[接收] 包#1, Seq=0, 偏移=0, 数据=567字节  
[ACK] 发送ACK=0  
[进度] 567/567 (100%)  
  
[完成] 文件接收完成!  
总大小: 567 字节  
总包数: 1  
[成功] 文件保存到: received\_files\test\_file.txt

### 测试2: 模拟丢包传输

**客户端**:

请输入命令: -upload test\_file.txt 0.3

**预期观察**:

* 30%概率ACK丢失
* 触发超时重传机制
* 最终文件完整传输
* 传输时间增加

**示例输出**(部分):

[发送] 包#1, Seq=0, 偏移=0, 大小=567 字节, 尝试=1  
[超时] 准备重传...  
[发送] 包#1, Seq=0, 偏移=0, 大小=567 字节, 尝试=2  
[ACK] 收到ACK=0  
[成功] ACK正确

### 测试3: 大文件传输

创建大文件用于测试:

# Windows下创建1MB测试文件  
fsutil file createnew large\_test.bin 1048576

上传:

请输入命令: -upload large\_test.bin 0.1

**预期观察**:

* 文件被分成多个1020字节的数据帧
* 显示传输进度百分比
* 序列号0和1交替出现
* 所有数据帧按序传输

### 测试4: 验证文件完整性

**Windows下**:

# 比较原文件和接收文件的MD5  
certutil -hashfile test\_file.txt MD5  
certutil -hashfile received\_files\test\_file.txt MD5

**预期结果**: 两个文件的MD5值完全相同

## 运行截图示例

### 服务器端

========================================  
文件传输服务器(基于停等协议)  
监听端口: 12340  
数据帧大小: 1020 字节  
========================================  
  
等待客户端请求...  
  
[请求] 来自 127.0.0.1 - 上传文件  
文件名: test\_file.txt  
文件大小: 567 字节  
[接收] 开始接收文件...  
保存路径: received\_files\test\_file.txt  
[信息] 文件大小: 567 字节  
[接收] 包#1, Seq=0, 偏移=0, 数据=567字节  
[ACK] 发送ACK=0  
[进度] 567/567 (100%)  
  
[完成] 文件接收完成!  
总大小: 567 字节  
总包数: 1  
[成功] 文件保存到: received\_files\test\_file.txt  
  
等待客户端请求...

### 客户端

========================================  
文件传输客户端(基于停等协议)  
服务器: 127.0.0.1:12340  
========================================  
  
命令:  
 -time 获取服务器时间  
 -upload <文件路径> [丢包率] 上传文件  
 示例: -upload test.txt 0.2  
 -quit 退出  
========================================  
  
请输入命令: -upload test\_file.txt 0.2  
  
[上传] 文件: test\_file.txt  
大小: 567 字节  
丢包率: 20%  
========================================  
[发送] 文件信息帧  
[确认] 服务器准备接收  
[发送] 包#1, Seq=0, 偏移=0, 大小=567 字节, 尝试=1  
[ACK] 收到ACK=0  
[成功] ACK正确  
[进度] 567/567 (100%)  
  
========================================  
[完成] 文件上传成功!  
总大小: 567 字节  
总包数: 1  
========================================

## 关键技术实现

### 1. 文件分块读取

char buffer[DATA\_SIZE];  
while (!inFile.eof()) {  
 inFile.read(buffer, DATA\_SIZE);  
 streamsize bytesRead = inFile.gcount();  
  
 // 构造数据帧  
 DataFrame frame;  
 frame.dataLen = (unsigned short)bytesRead;  
 memcpy(frame.data, buffer, bytesRead);  
  
 // 发送...  
}

### 2. 超时重传机制

int attempts = 0;  
const int MAX\_ATTEMPTS = 5;  
  
while (attempts < MAX\_ATTEMPTS) {  
 // 发送数据帧  
 sendto(...);  
  
 // 等待ACK(超时2秒)  
 select(..., timeout);  
  
 if (收到正确ACK) {  
 break;  
 }  
  
 attempts++;  
}

### 3. 进度显示

float progress = (float)(offset + bytesRead) / fileSize \* 100;  
cout << "[进度] " << (offset + bytesRead) << "/" << fileSize  
 << " (" << progress << "%)" << endl;

### 4. 文件写入

ofstream outFile(savePath, ios::binary);  
  
while (接收数据帧) {  
 outFile.write(frame.data, frame.dataLen);  
 receivedSize += frame.dataLen;  
  
 if (frame.flag == 1) { // 最后一帧  
 break;  
 }  
}  
  
outFile.close();

## 功能特性

### 已实现功能

* ✓ 基于停等协议的可靠文件传输
* ✓ 支持任意大小文件
* ✓ 超时重传机制
* ✓ 序列号交替(0/1)
* ✓ 传输进度显示
* ✓ 丢包模拟测试
* ✓ 文件完整性保证
* ✓ 错误处理和恢复

### 可扩展功能

* ⚠ 支持文件下载(服务器→客户端)
* ⚠ 文件列表查询
* ⚠ 断点续传
* ⚠ 多文件批量传输
* ⚠ 文件校验和验证(MD5/SHA256)
* ⚠ 传输速率统计
* ⚠ 压缩传输

## 性能分析

### 传输效率

对于文件大小为 F 字节:

* **数据帧数**: N = ⌈F / 1020⌉
* **总传输时间**: T = N × (发送时间 + RTT + 处理时间)
* **有效吞吐量**: Throughput = F / T

### 丢包影响

* 丢包率p下,平均重传次数: R = 1 / (1 - p)
* 实际传输时间: T' = T × R
* 效率降低: η = 1 / R

示例:

* p=0.1 (10%丢包): R≈1.11, 效率降低11%
* p=0.3 (30%丢包): R≈1.43, 效率降低43%
* p=0.5 (50%丢包): R=2.00, 效率降低50%

## 与原始停等协议对比

| 特性 | 基础停等协议 | 文件传输应用 |
| --- | --- | --- |
| 应用场景 | 简单数据传输 | 文件传输 |
| 数据分块 | 固定测试数据 | 动态文件分块 |
| 元数据 | 无 | 文件名、大小 |
| 进度跟踪 | 无 | 实时进度显示 |
| 文件操作 | 无 | 文件读写 |
| 实用性 | 演示用 | 实际可用 |

## 验证要点

* ✓ 小文件(<1KB)传输正确
* ✓ 大文件(>1MB)传输正确
* ✓ 文件内容完全一致(MD5校验)
* ✓ 丢包情况下可靠传输
* ✓ 超时重传机制有效
* ✓ 序列号正确交替
* ✓ 进度显示准确
* ✓ 错误处理完善

## 使用建议

1. **测试文件准备**
   * 使用提供的test\_file.txt
   * 创建不同大小的测试文件
   * 准备各种类型文件(文本、二进制、图片等)
2. **丢包率设置**
   * 开发阶段: 0 (无丢包)
   * 功能测试: 0.1-0.2 (轻度丢包)
   * 压力测试: 0.3-0.5 (重度丢包)
3. **性能优化**
   * 调整DATA\_SIZE(当前1020字节)
   * 优化超时时间
   * 考虑网络RTT
4. **安全注意**
   * 验证文件路径安全性
   * 限制文件大小上限
   * 防止目录遍历攻击

## 常见问题

### Q1: 为什么数据帧大小是1020字节?

A:

* 以太网MTU: 1500字节
* IP头部: 20字节
* UDP头部: 8字节
* 剩余: 1472字节
* 减去DataFrame其他字段: ≈1020字节安全

### Q2: 大文件传输很慢怎么办?

A:

* 这是停等协议的固有限制
* 考虑升级到GBN或SR协议
* 或增大数据帧大小

### Q3: 如何验证传输正确性?

A:

# Windows  
certutil -hashfile original.file MD5  
certutil -hashfile received\_files\original.file MD5  
  
# 比较两个MD5值

### Q4: 传输中断怎么办?

A: 当前实现不支持断点续传,需要重新传输。可扩展实现:

* 记录已传输偏移
* 客户端请求从指定偏移续传
* 服务器支持追加写入

## 改进方向

1. **功能增强**
   * 支持文件下载
   * 实现断点续传
   * 添加文件校验
2. **性能优化**
   * 动态调整超时时间
   * 自适应数据帧大小
   * 流水线传输(过渡到GBN)
3. **用户体验**
   * GUI界面
   * 传输速率显示
   * 剩余时间估算
4. **安全性**
   * 用户认证
   * 传输加密
   * 访问控制

## 总结

本文件传输应用完整展示了如何基于停等协议构建实用的C/S应用,具有以下特点:

* ✅ 协议设计合理
* ✅ 可靠性保证
* ✅ 易于测试验证
* ✅ 代码结构清晰
* ✅ 注释完善
* ✅ 可扩展性强

是学习网络协议和文件传输的优秀实践项目!