# 双向停等协议 (Bidirectional Stop-and-Wait Protocol)

## 实验简介

本实验是Lab2-1的选做内容1,在原有单向停等协议基础上改进,支持双向数据传输。

## 改进要点

### 与单向协议的区别

| 特性 | 单向协议 | 双向协议 |
| --- | --- | --- |
| 数据传输方向 | 服务器→客户端 | 双向 |
| 序列号管理 | 单一序列号 | 发送和接收各维护序列号 |
| 帧类型 | 数据帧、ACK帧 | 数据帧、ACK帧(统一格式) |
| 复杂度 | 简单 | 中等 |

### 关键改进

1. **双序列号机制**

* unsigned char sendSeq = 0; // 发送序列号  
  unsigned char recvSeq = 0; // 接收序列号
  + 发送和接收独立维护序列号
  + 避免冲突和混淆

1. **统一帧格式**

* struct DataFrame {  
   unsigned char seq; // 序列号  
   char data[1024]; // 数据  
   unsigned char flag; // 0=数据, 1=结束, 2=ACK  
  };
  + 使用flag字段区分帧类型
  + 数据帧和ACK帧使用相同结构

1. **双向传输能力**
   * 服务器可以向客户端发送数据
   * 客户端可以向服务器发送数据
   * 支持同时进行双向通信

## 编译说明

cd Lab2-1/bidirectional  
  
# 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

服务器启动后会显示:

========================================  
双向停等协议服务器已启动  
监听端口: 12340  
支持双向数据传输  
========================================

### 2. 启动客户端

./client.exe

### 3. 可用命令

#### 客户端命令

* **-time**: 获取服务器时间
* **-send [丢包率]**: 客户端向服务器发送数据
  + 示例: -send 0.2 (20% ACK丢失率)
* **-recv [丢包率]**: 接收服务器发送的数据
  + 示例: -recv 0.3 (30%丢包率)
* **-both [丢包率]**: 双向传输综合测试
  + 示例: -both 0.2
* **-quit**: 退出客户端

## 测试示例

### 测试1: 客户端→服务器

**客户端**:

请输入命令: -send 0.2

**预期结果**:

* 客户端发送4个数据包到服务器
* 服务器接收并发送ACK
* 可能出现ACK丢失(20%概率)
* 超时后重传
* 最终所有数据成功传输

### 测试2: 服务器→客户端

**客户端**:

请输入命令: -recv 0.2

**预期结果**:

* 服务器发送4个数据包到客户端
* 客户端接收并发送ACK
* 模拟20%数据包或ACK丢失
* 触发重传机制
* 所有数据正确接收

### 测试3: 双向传输

**客户端**:

请输入命令: -both 0.2

**预期结果**:

[双向传输综合测试]  
丢包率: 20%  
========================================  
  
阶段1: 客户端->服务器  
----------------------------------------  
[发送->服务器] Seq=0, 数据="客户端: Hello Server!", 尝试=1  
[接收ACK] ACK=0  
[成功] 收到正确ACK  
...  
  
阶段2: 服务器->客户端  
----------------------------------------  
[接收] 服务器消息1: 双向通信测试  
[接收] 服务器消息2: 停等协议  
...  
  
========================================  
[完成] 双向传输测试完成!  
========================================

## 运行示例输出

### 服务器端

========================================  
双向停等协议服务器已启动  
监听端口: 12340  
支持双向数据传输  
========================================  
  
等待消息...  
  
[接收<-客户端] Seq=0, 数据="客户端消息1: 你好服务器"  
[接受] 序列号正确  
[发送ACK->客户端] ACK=0  
[数据] 客户端消息1: 你好服务器  
  
[接收<-客户端] Seq=1, 数据="客户端消息2: 双向通信"  
[接受] 序列号正确  
[发送ACK->客户端] ACK=1  
[数据] 客户端消息2: 双向通信  
...  
  
[命令] 来自 127.0.0.1 - -send  
  
[双向传输测试] 服务器->客户端  
======================================  
[发送->客户端] Seq=0, 数据="服务器消息1: 双向通信测试", 尝试=1  
[接收ACK] ACK=0  
[成功] 收到正确ACK  
...

### 客户端

========================================  
双向停等协议客户端  
服务器: 127.0.0.1:12340  
支持双向数据传输  
========================================  
  
请输入命令: -send 0.2  
  
[双向传输测试] 客户端->服务器  
ACK丢失率: 20%  
======================================  
[发送->服务器] Seq=0, 数据="客户端消息1: 你好服务器", 尝试=1  
[接收ACK] ACK=0  
[成功] 收到正确ACK  
  
[发送->服务器] Seq=1, 数据="客户端消息2: 双向通信", 尝试=1  
[模拟] ACK丢失!  
[超时] 准备重传...  
[发送->服务器] Seq=1, 数据="客户端消息2: 双向通信", 尝试=2  
[接收ACK] ACK=1  
[成功] 收到正确ACK  
...  
======================================  
[完成] 客户端数据发送完毕

## 协议工作流程

### 客户端→服务器流程

客户端 服务器  
 | |  
 |------ 数据帧(Seq=0) ----------->|  
 | | 检查序列号  
 | | 发送ACK  
 |<------- ACK(Seq=0) -------------|  
 | 收到ACK |  
 | 切换sendSeq |  
 | |  
 |------ 数据帧(Seq=1) ----------->|  
 | (ACK丢失模拟) |  
 |<------- ACK(Seq=1) ----X |  
 | 超时 |  
 |------ 数据帧(Seq=1) ----------->| (重传)  
 |<------- ACK(Seq=1) -------------|  
 | |

### 服务器→客户端流程

服务器 客户端  
 | |  
 |------ 数据帧(Seq=0) ----------->|  
 | | (数据丢失模拟)  
 X | 不发送ACK  
 | 超时 |  
 |------ 数据帧(Seq=0) ----------->| (重传)  
 | | 检查序列号  
 |<------- ACK(Seq=0) -------------|  
 | 收到ACK |  
 | 切换sendSeq |  
 | |

## 关键技术实现

### 1. 双序列号管理

// 发送数据时使用sendSeq  
DataFrame frame;  
frame.seq = sendSeq;  
// 发送成功后切换  
sendSeq = 1 - sendSeq;  
  
// 接收数据时检查recvSeq  
if (frame.seq == recvSeq) {  
 // 接受数据  
 recvSeq = 1 - recvSeq;  
}

### 2. 帧类型识别

if (frame.flag == 2) {  
 // 这是ACK帧  
 处理ACK();  
} else if (frame.flag == 1) {  
 // 这是结束数据帧  
 处理数据并结束();  
} else {  
 // 这是普通数据帧  
 处理数据();  
}

### 3. 超时重传机制

// 使用select实现超时  
fd\_set readSet;  
FD\_ZERO(&readSet);  
FD\_SET(sock, &readSet);  
  
timeval timeout;  
timeout.tv\_sec = 2;  
timeout.tv\_usec = 0;  
  
if (select(0, &readSet, NULL, NULL, &timeout) == 0) {  
 // 超时,重传  
 重传数据帧();  
}

## 验证要点

* ✓ 客户端可以向服务器发送数据
* ✓ 服务器可以向客户端发送数据
* ✓ 发送和接收序列号独立管理
* ✓ ACK丢失触发重传
* ✓ 数据包丢失触发重传
* ✓ 序列号正确切换
* ✓ 双向同时通信测试通过

## 改进总结

### 相比单向协议的优势

1. **更实用**: 支持双向通信,更接近实际应用
2. **更灵活**: 任意一方都可主动发送数据
3. **更完整**: 完整实现了停等协议的双向特性

### 实现难点

1. **序列号管理**: 需要分别维护发送和接收序列号
2. **帧类型区分**: 需要正确识别数据帧和ACK帧
3. **并发处理**: 需要处理可能同时到达的数据和ACK

## 注意事项

1. 发送和接收序列号必须独立维护
2. ACK帧也使用相同的DataFrame结构
3. 需要正确处理ACK超时和数据超时
4. 双向传输时要避免死锁
5. 测试时建议先单向再双向

## 扩展方向

1. 实现piggybacking(捎带ACK)
2. 优化双向同时传输的效率
3. 添加流量控制机制
4. 支持更复杂的应用层协议