目录

1 实验内容说明

1.1 实验题目

实验 3：基于 UDP 服务设计可靠传输协议并编程实现

1.2 实验说明

利用数据报套接字在用户空间实现面向连接的可靠数据传输，功能包括：建立连接、差错检测、确认重传等。流量控制采用停等机制，完成给定测试文件的传输。

实验要求：

- 数据报套接字：UDP；
- 建立连接：实现类似 TCP 的握手、挥手功能；
- 差错检测：计算校验和；
- 确认重传：rdt2.0、rdt2.1、rdt2.2、rdt3.0 等，亦可自行设计协议；
- 单向传输：发送端、接收端；
- 有必要日志输出。

2 协议设计

停等协议使用 rdt3.0 协议，数据包设计参考 TCP 数据头格式。

2.1 数据包格式

为了实现可靠传输，首先设计数据包格式，不仅包括停等协议，也为后续实验做准备。

图 1: 数据包格式

标志位分别设置了：ACK，SYN，FIN 三项。

2.2 发送端状态机

发送端的等效状态机如图2所示，这是课件的原图。

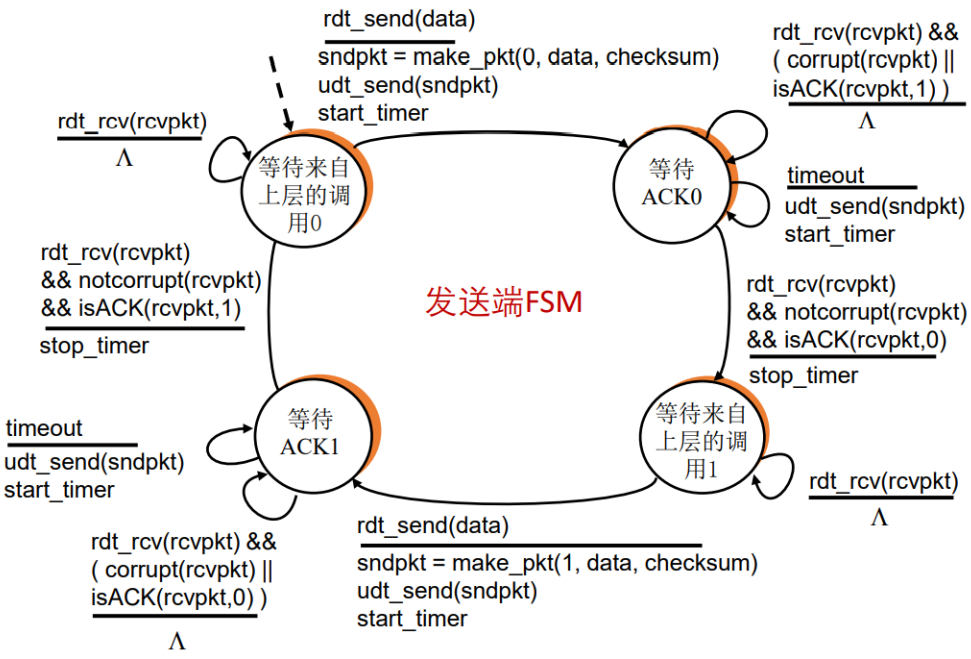

图 2: 发送状态机

2.3 接收端状态机

接收端的等效状态机如图3所示，同样是课件的原图。

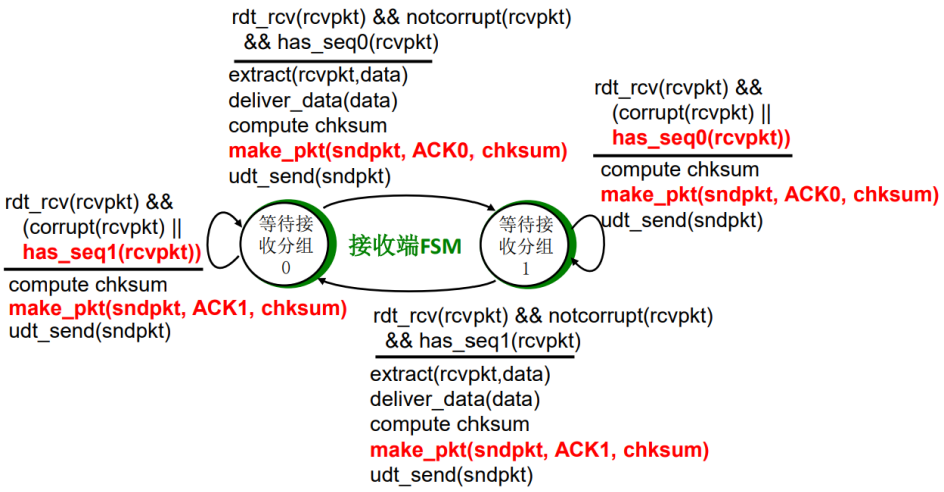

图 3: 接收状态机

3 单向传输实现

3.1 差错检测实现

思路是使用 32 位整数对 IP 头各 16 位进行加和，最后令结果的高 16 位于低 16 位相加。

</> CODE 1: 计算校验和

```
// file: common/RPkg.h
inline uint16_t RPkg::calCheckSum()
{
    uint16_t* words = (uint16_t*)this;
    uint32_t checksum = 0; //计算中溢出位自动进位
    size_t size = this->size();
    int looplen = size / sizeof(uint16_t);
    for (int i = 0; i < looplen; i++)
        checksum += nToH(words[i]);
    //处理数据不对齐
    if (size % sizeof(uint16_t) == 1) {
        uint16_t part = ((uint8_t*)this)[size - 1];
        checksum += part << 8;
    }
    //加上溢出位
    checksum = (checksum >> 16) + (checksum & 0x0000FFFF);
    checksum = (checksum >> 16) + checksum;
    //取反截断
    return (uint16_t)~checksum;
}
```

Note

对于校验和的计算，最多 2 次高低位相加后，高 16 位一定为 0，故高低位只需相加 2 次。

3.2 建立连接

由于只需实现单向传输，这里只实现两次握手，即发送端发送 SYN，接收端回复 ACK 的一来一回的过程。

发送端接收到接收端的 ACK 后，成功建立连接。接收端接收到 SYN 后，成功建立连接。发送端多次超时重发未接收到接收端 ACK，则连接失败。

</> CODE 2: 发送过程代码片段

```
socket.sendto(ptr,sPkg->size(),targetAddr,targetPort);
RPkg* rPkg = recvPkg(); //接收，带超时重发
setState(State::send);
```

3.3 发送端实现

发送端的状态管理由单独的线程负责。发送端的状态有两个：Send、Wait。每个状态又分为 0 和 1 两组，总计 4 种情况，对应2中的 4 中状态。发送端会设置一位变量，记录当前发送的分组 0 和 1 中的某一个。

</> CODE 3: 发送端状态管理代码片段

```
while (true) {
    if (timeToClose()) {    //判断是否应该结束
        closeConnection(); //关闭连接
        break;
    }
    switch (state) {
    case State::send:       //判断当前状态
        doSend();
        break;
    case State::wait:
        doWait();
        break;
    }
}
```

当发送端处于发送状态时，则发送数据包，将 SEQ 为当前分组 (0/1)。当发送端出于接收状态时，则接收数据包。接收过程中需进行超时重传，同时对接收到错误的 ACK 回复不做处理。

</> CODE 4: 发送端重传与差错检测

```
RPkg* RSend::recvPkg()
{
    while (true) {
        auto timeout = milliseconds(getRTO());
        auto rnum = socket.recvfrom(rPkgBuf, maxPkgSize(),
                                    sourceIP, sourcePort, timeout);
        if (rnum != 0) {  //if 未超时
            nTimeout = 0;
            RPkg* rPkg= reinterpret_cast<RPkg*>(rPkgBuf);
            if ((rPkg->calCheckSum()) == 0 || rPkg->checkSum == 0)
                return rPkg; //无差错或无校验,则成功收到数据包
        }//else 超时重传
        if (++nTimeout > maxNTimeout) {
            //超时错误处理
        }
        socket.sendto(sPkgBuf,sPkg->size(),targetAddr,targetPort);
    }
}
```

而接收端成功接收到数据包后，要比对 ACK 和发送是 SEQ 是否相同。如果不同则不做处理，否则设置新的 SEQ。

</> CODE 5: 发送端接收逻辑

```
// file: Sender/RSend.cpp
void RSend::doWait()
{
    RPkg* rPkg = recvPkg();
    uint32_t ack = rPkg->getACK();
    if (!(rPkg->flags & RPkg::F_ACK))
        return;
    if (ack != group)
        return;
    group = group ? 0 : 1;  //设置当前发送的SEQ
    setState(State::send);
}
```

3.4 接收端实现

接收端只有一个 Recv 状态，同样设置一个标记记录当前要确认的 SEQ。接收端接收到有正确的 SEQ 的数据包，则进行数据提交，否则，只进行确认不进行提交。接收端只要收到无差错的数据，就对该 SEQ 进行确认，不论 SEQ 是否与当前标记是否相符。

3.5 连接关闭

发送端发起连接的关闭，发送端在发送缓存区全部发送完毕后，开始关闭过程。首先，向接收端发送 FIN。然后，设置超时重传定时器，并等待接收端回复 ACK。接收到 ACK 则成功关闭。

接收端接收到 FIN 后，回复 ACK 并关闭接收端。

</> CODE 6: 发送端关闭片段

```
// file: Sender/RSend.cpp
void RSend::closeConnection()
{
    //发送FIN
    RPkg* sPkg = new(sPkgBuf) RPkg();
    sPkg->makeFIN(group);
    sPkg->setCheckSum(sPkg->calCheckSum());
    socket.sendto(sPkgBuf,sPkg->size(),targetAddr,targetPort);
    //设置定时器并等待接收
    recvPkg();
    //成功关闭
    setState(State::closed);
}
```

4 效果演示

4.1 发送文件

发送端、接收端操作如下

```
Sender -i 127.0.0.1 -p 10100 -f 1.jpg
Recipient.exe -i 127.0.0.1 -p 10100
```

成功发送文件，速度、平均吞吐率会显示在日志中。

```
send file spent 0.521479 s.
1857359 bytes in total. speed = 3561713 byte ps
success to send the file: 1.jpg.
```

4.2 日志显示

以一个小文件为例，发送端会记录连接、状态转移、关闭等操作。

```
[LOG]: succeed to connect.
[LOG]: ===== state(closed) ---> state(send) =====
[LOG]: Succeed to connect to host:127.0.0.1:10100
[LOG]: user try to send 11 bytes.
[LOG]: user try to send 10 bytes.
[LOG]: user try to close.
[LOG]: manager thread start.
[LOG]: send group=0 seq=0 size=21.
[LOG]: ===== state(send) ---> state(wait) =====
[LOG]: send FIN.
[LOG]: recv ack.
[LOG]: ===== state(wait) ---> state(closed) =====
[LOG]: manager thread end.
[LOG]: close successfully.
```

接收端也会显示接收情况

```
[LOG]: succeed to listen a connect from 127.0.0.1:60188.
[LOG]: manager thread start.
[LOG]: recv group=0 seq=0 deliver=21
[LOG]: user recv 21 bytes.
[LOG]: send ack=0
[LOG]: manager thread end.
[LOG]: user recv 0 bytes.
```