计网 Lab2 实验报告

221220134 佟一飞

一、 程序结构与设计

首先是序号转换部分，分别实现了 64 位绝对序号转 32 位序号和 32 位序号转 64 位绝对序号。

64 位转 32 位只需截取低 32 位并加上起始 32 位序号：

```
Wrap32 Wrap32::wrap( uint64_t n, Wrap32 zero_point )
{
  // Your code here.
  return zero_point+(uint32_t)(n%(1ll<<32));
}
```

32 位转 64 位则首先找出和起始序号的差，然后找到离 checkpoint 最近的两个绝对序号，选其中离的更近的返回即可：

```
uint64_t Wrap32::unwrap( Wrap32 zero_point, uint64_t checkpoint ) const
{
  // Your code here.
  uint32_t tmp=raw_value_-zero_point.raw_value_;
  uint64_t tmp1=tmp;
  if(checkpoint<tmp1) return tmp1;
  else{
    uint64_t cnt=(checkpoint-tmp1)/(1ll<<32);
    uint64_t ans1=tmp1+cnt*(1ll<<32),ans2=tmp1+(cnt+1)*(1ll<<32);
    if(ans2-checkpoint<checkpoint-ans1) return ans2;
    else return ans1;
  }
}
```

然后是 TCP 接收者的 receive 方法和 send 方法。

Receive 方法首先根据读入的 RST 设置字节流的错误位，然后如果有 SYN 则设置当前的 32 位序号为起始序号，然后向 Reassembler 输入字符串即可。输入字符串的起始下标为当前 32 位序号转 64 位序号的结果，checkpoint 为已经读入字节流的字节数加 1，同时得到的 64 位序号如果当前不包含 SYN 则需要减 1 得到真正的起始下标：

```
void TCPReceiver::receive( TCPSenderMessage message )
{
  // Your code here.
  if(message.RST==1){
    rst=1;
    reader().set_error();
  }
  if(message.SYN==1&&!init) zero=message.seqno,init=1;
  reassembler_.insert(message.seqno.unwrap(zero,writer().bytes_pushed()+1)-1+(message.SYN),message.payload,message.FIN);
}
```

然后 send 方法返回字节流是否有错误，当前可以继续输入的容量以及请求的起始 32 位序号。当前可以继续输入的容量是 UINT16_MAX 和当前字节流剩余容量的最小值，请求的起始序号是当前字节流已经读入的容量加 1，如果字节流已经关闭了则再加 1：

```cpp
TCPReceiverMessage TCPReceiver::send() const
{
  // Your code here.
  TCPReceiverMessage res;
  if((rst==1)||reader().has_error()) res.RST=1;
  if(writer().available_capacity()>UINT16_MAX) res.window_size=UINT16_MAX;
  else res.window_size=(uint16_t)writer().available_capacity();
  if(init){
    if(!writer().is_closed()) res.ackno.emplace(zero+(writer().bytes_pushed()+1));
    else res.ackno.emplace(zero+(writer().bytes_pushed()+2));
  }
  return res;
}
```

二、 实验结果

```
      Start 16: wrapping_integers_cmp
15/29 Test #16: wrapping_integers_cmp ............   Passed    0.02 sec
      Start 17: wrapping_integers_wrap
16/29 Test #17: wrapping_integers_wrap ...........   Passed    0.01 sec
      Start 18: wrapping_integers_unwrap
17/29 Test #18: wrapping_integers_unwrap .........   Passed    0.01 sec
      Start 19: wrapping_integers_roundtrip
18/29 Test #19: wrapping_integers_roundtrip ......   Passed    1.65 sec
      Start 20: wrapping_integers_extra
19/29 Test #20: wrapping_integers_extra ..........   Passed    0.30 sec
      Start 21: recv_connect
20/29 Test #21: recv_connect .....................   Passed    0.08 sec
      Start 22: recv_transmit
21/29 Test #22: recv_transmit ....................   Passed    0.51 sec
      Start 23: recv_window
22/29 Test #23: recv_window ......................   Passed    0.02 sec
      Start 24: recv_reorder
23/29 Test #24: recv_reorder .....................   Passed    0.03 sec
      Start 25: recv_reorder_more
24/29 Test #25: recv_reorder_more ................   Passed    1.56 sec
      Start 26: recv_close
25/29 Test #26: recv_close .......................   Passed    0.02 sec
      Start 27: recv_special
26/29 Test #27: recv_special .....................   Passed    0.03 sec
      Start 37: compile with optimization
27/29 Test #37: compile with optimization ........   Passed    1.04 sec
      Start 38: byte_stream_speed_test
             ByteStream throughput: 0.77 Gbit/s
28/29 Test #38: byte_stream_speed_test ...........   Passed    0.20 sec
      Start 39: reassembler_speed_test
             Reassembler throughput: 4.63 Gbit/s
29/29 Test #39: reassembler_speed_test ...........   Passed    0.24 sec
```