Computer Networks Lab2

黄嘉祺 221220108

1 Program Structure and Design

1.1 Wrap32

- 实验要求我们完成 Wrap32 类型用于存储 seqno 和 absolute seqno 、 stream index 之间的转换
- 具体来说，我们需要在 src/wrapping_integers.cc 中实现两个成员函数 wrap 与 unwrap ， 分别用于接收一个 int64_t 类型的数 n 与 Wrap32 类型的 zero_point 作为 ISN 并根据 ISN 将 n 转化为 Wrap32 类型，以及接收一个 Wrap32 类型的数 zero_point （ ISN ）和 int64_t 类型的 check_point ， 根据 ISN 的值将调用该成员函数的 Wrap32 类型的数转化为 int64_t 类型的数
- 这样就实现了 seqno （ Wrap32 类型）和 absolute seqno （ uin64_t 类型）间的转换，而 stream index （ int64_t 类型）和 absolute seqno （ uin64_t 类型）间的转换只需要在值上加减 1 即可
- wrap 函数的实现思路是：
  - 直接将 n 加上 zero_point 的值，然后对 2^32 取模（具体通过强制类型转换为 uint32_t 来实现），将取模后的值作为 Wrap32 类型的 seqno 返回

```
Wrap32 Wrap32::wrap( uint64_t n, Wrap32 zero_point )
{
  // Your code here.
  // (void)n;
  // (void)zero_point;
  return Wrap32 { (uint32_t)( zero_point.raw_value_ + n ) };
}
```

- unwrap 函数的实现思路是：
  - 首先计算 this 与 zero_point 的差值（若为负数则设为模 2^32 同余的第一个非负数）
  - 用 check_point 除以 2^32 得到商，将刚刚的差加上商乘以 2^32 ， 接下来考虑该值是否是距离 check_point 最近的模 2^32 同余的数
  - 若该值与 check_point 的距离小于等于 2^31 ， 则该值已是距离 check_point 最近的，否则将该值加/减去 2^32 直到距离 check_point 最近
  - 返回这个最近的值

```cpp
uint64_t Wrap32::unwrap( Wrap32 zero_point, uint64_t checkpoint ) const
{
  // Your code here.
  // (void)zero_point;
  // (void)checkpoint;
  // return {};
  int64_t diff = this->raw_value_ - zero_point.raw_value_;
  if ( diff < 0 ) {
    diff += UINT32_MAX + 1ULL;
  }
  uint64_t udiff = (uint64_t)diff;
  uint64_t factor = checkpoint / ( UINT32_MAX + 1ULL );
  uint64_t ans = factor * ( UINT32_MAX + 1ULL ) + udiff;
  if ( ans > checkpoint ) {
    if ( ans - checkpoint > ( UINT32_MAX + 1ULL ) / 2 && ans > UINT32_MAX ) {
      ans -= ( UINT32_MAX + 1ULL );
    }
  } else {
    if ( checkpoint - ans > ( UINT32_MAX + 1ULL ) / 2 && ans < UINT64_MAX - UINT32_MAX ) {
      ans += ( UINT32_MAX + 1ULL );
    }
  }
  return ans;
}
```

1.2 TCPReceiver

- 接下来实验要求我们完成 TCPReceiver 类，用于在接收方接收数据流时将其中的 seqno 转换为 stream_index 从而可以用于 Reassembler ， 此外它还需要向发送方发送 ackno 以及 window size
- 具体来说，我们需要在 src/tcp_receiver.cc 中实现成员函数 receive 和 send ， 我首先在 src/tcp_receiver.hh 的 TCPReceiver 类的定义中补充成员变量 bool 类型的 syn_ 用于指示是否接受到 SYN 包， Wrap32 类型的 seqno_ 用于存储 ISN
- receive 函数的实现思路是：
  - 若接收到的 message.RST 为真，调用 set_error 函数将 bytestream 的 error 设为真
  - 将收到 SYN 包前收到的数据包全部丢弃
  - 若此时未收到 SYN 包且当前收到的包为 SYN 包，将 syn_ 设为真，并将 seqno_ 设为 message.seqno
  - 对于接收到的 message.seqno ， 对齐调用 unwrap 函数将其转化为 absolute seqno （ zero_point 为 seqno_ ， check_point 为 first_unassembled （ writer().bytes_pushed() ） ）
  - 再将 absolute seqno 转化为 stream_index ， 并调用 reassembler.insert 函数将其插入数据流中

```cpp
void TCPReceiver::receive( TCPSenderMessage message )
{
  // Your code here.
  // (void)message;
  if ( message.RST ) {
    reader().set_error();
  }
  if ( ( !message.SYN && !syn_ ) || ( message.SYN && syn_ ) ) {
    return;
  }
  if ( message.SYN && !syn_ ) {
    syn_ = true;
    seqno_ = message.seqno;
  }
  uint64_t absolute_seqno = message.seqno.unwrap( seqno_, writer().bytes_pushed() );
  uint64_t index = absolute_seqno - !message.SYN;
  reassembler_.insert( index, message.payload, message.FIN );
}
```

- send 函数的实现思路是：
  - 首先调用 bytestream 中的 has_error 函数检测是否需要重设，若需要则将 message.RST 设为真
  - 将 message.window_size 设为 writer().available_capacity() 和 UINT16_MAX 中的较小值
  - 将 absolute_seqno 设为 first_unassembled （ writer().bytes_pushed() ）加 1，
    若 bytestream 已关闭则再加 1，并调用 wrap 函数（ zero_point 为 seqno_ ）
    将 absolute_seqno 转化为 seqno
  - 将 message.ackno 设为 seqno
  - 返回 message

```cpp
TCPReceiverMessage TCPReceiver::send() const
{
  // Your code here.
  TCPReceiverMessage message;
  if ( reader().has_error() ) {
    message.RST = true;
  }
  uint64_t window_size = writer().available_capacity();
  message.window_size = window_size <= UINT16_MAX ? window_size : UINT16_MAX;
  uint64_t first_unassembled = writer().bytes_pushed();
  if ( !syn_ ) {
    message.ackno = {};
  } else {
    uint64_t absolute_seqno = first_unassembled + 1 + writer().is_closed();
    message.ackno = Wrap32::wrap( absolute_seqno, seqno_ );
  }
  return message;
}
```

2 Implementation Challanges

- 在实现 unwrap 的时候，一开始我错误认为 2^64 和 2^32 之间的差为 2^32，导致我认为计算完 diff 后最多只需要加一次 2^32，因此计算出的 absolute seqno 并不是离 check_point 最近的，后来我改为先计算 factor，并加 factor 个 2^32，成功解决了这个问题

- 在实现 `send` 的时候，我一开始在计算 `absolute_seqno` 的时候只加了 `1`，没有再加一个 `writer().is_closed()`，然而实际上由于 `stream index` 并不会包括 `FIN`，也就是说 `absolute_seqno` 的最后一位比 `stream index` 的最后一位还要再多一位，因此导致了 `absolute_seqno` 与 `stream index` 不匹配的问题，后来我加上了 `writer().is_closed()`，解决了这个问题

3 Remaining Bugs

目前我的代码已经通过了全部的测试，暂时未发现明显的bug

4 Experimental Results and Performance

- Bytestream, reassembler, wrapper and tcp receiver: Passed all 29 tests

```
13/29 Test #14: reassembler_overlapping .........   Passed    0.02 sec
      Start 15: reassembler_win
14/29 Test #15: reassembler_win .................   Passed    0.55 sec
      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    0.90 sec
      Start 20: wrapping_integers_extra
19/29 Test #20: wrapping_integers_extra ..........   Passed    0.17 sec
      Start 21: recv_connect
20/29 Test #21: recv_connect .....................   Passed    0.01 sec
      Start 22: recv_transmit
21/29 Test #22: recv_transmit ....................   Passed    0.37 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    0.02 sec
      Start 25: recv_reorder_more
24/29 Test #25: recv_reorder_more ................   Passed    1.67 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    0.02 sec
      Start 37: compile with optimization
27/29 Test #37: compile with optimization ........   Passed    0.45 sec
      Start 38: byte_stream_speed_test
             ByteStream throughput: 2.51 Gbit/s
28/29 Test #38: byte_stream_speed_test ...........   Passed    0.10 sec
      Start 39: reassembler_speed_test
             Reassembler throughput: 6.94 Gbit/s
29/29 Test #39: reassembler_speed_test ...........   Passed    0.16 sec

100% tests passed, 0 tests failed out of 29

Total Test time (real) =   5.32 sec
Built target check2
```