Checkpoint 2 Writeup

My name: 赖炜欣

My student ID : 205220026

I did attend the lab session.
这个实验修改了wrapping_integers和tcp_receiver的文件。

wrapping_integers的主要功能是32位序列号与64位序列号之间的转换。

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

计算n的低32位+zero_point.raw_value_的出来的结果，自动从64无符号对象变成wrap32对象

```
13 uint64_t Wrap32::unwrap( Wrap32 zero_point, uint64_t checkpoint ) const
14 {
15   const uint64_t MODULO = 1ULL << 32;
16   uint64_t offset = (static_cast<uint64_t>(raw_value_) + MODULO - zero_point.raw_value_) %
   MODULO;
17   uint64_t base = checkpoint & ~(MODULO - 1);
18   uint64_t cand = base + offset;
19   if (cand + (MODULO / 2) < checkpoint) {
20         cand += MODULO;
21     } else if (cand > checkpoint + (MODULO / 2)) {
22         if (cand >= MODULO) {
23             cand -= MODULO;
24         }
25     }
26     return cand;
27 }
28
```

在unwrap函数里实现的是将32位的值拆解为靠近checkpoint的64位整数。
首先，先计算offset = (raw_value+ 2^32- zero_point.raw_value) mod 2^32
然后计算base = checkpoint ~ (2^32-1)
计算cand=base+offset，选一个cand和checkpoint的差值最小的数，所以利用了if判断，如果cand比checkpoint小很多就需要加2^32，反之大太多的话就需要减 2^32。
最后得到的cand是最接近checkpoint的。

tcp_receiver的主要功能是负责接受来自TCP发送端的信息和向发送端提供反馈。

```cpp
void TCPReceiver::receive( TCPSenderMessage message )
{
  // Your code here.
  //(void)message;
  if ( message.RST ) {
    reassembler_.reader().set_error();
    return;
  }

  if ( message.SYN && !is_zero_point_set ) {
    zero_point = message.seqno;
    message.seqno = message.seqno + 1;
    is_zero_point_set = true;
  }

  if ( !is_zero_point_set ) {
    return;
  }

  uint64_t first_index = message.seqno.unwrap( zero_point,
reassembler_.writer().bytes_pushed() );

  if ( first_index == 0 ) {
    return;
  }else{
    first_index--;
  }

  reassembler_.insert( first_index, message.payload, message.FIN );

  next_acknum
    = zero_point + is_zero_point_set + reassembler_.writer().bytes_pushed() +
reassembler_.writer().is_closed();
}
```

这个函数处理收到的信息。
首先先判断RST，如果收到了RST信号，表示发送端请求重置连接，所以需要标记错误并停止处理。
如果收到SYN信号，且zero_point没被设置，表示连接建立，所以需要初始化zero_point，zero_point设置成message.seqno，message.seqno将移动到下一个位置，然后把is_zero_point_set已设置。
如果还没设置好zero_point，消息会无法处理所以需要等待收到SYN信号。
first_index设置需要使用Unwrap函数将循环序列号变成绝对序列号。
如果序列号为0表示消息无效可以直接忽略，如果大于0就减1。
更新next_acknum，next_acknum是下一个希望接收到的序列号。

```cpp
TCPReceiverMessage TCPReceiver::send() const
{
  // Your code here.
  //return {};
  TCPReceiverMessage ReceiverMessage;

  if ( is_zero_point_set ) {
    ReceiverMessage.ackno = next_acknum;
  }

  ReceiverMessage.RST = reassembler_.reader().has_error();

  if ( reassembler_.writer().available_capacity() <= UINT16_MAX ) {
    ReceiverMessage.window_size = reassembler_.writer().available_capacity();
  } else {
    ReceiverMessage.window_size = UINT16_MAX;
  }

  return ReceiverMessage;

}
```


如果zero_point被设置，就将当前的next_acknum赋值给ReceiverMessage.ackno，否则就会保持默认值。
调用reassembler_.reader().has_error()检查是否存在错误，如果存在错误就设置ReceiverMessage.RST为true。
要计算window_size的大小，先获取当前接受缓冲区的剩余容量，如果剩余容量小于UINT16_MAX就直接使用剩余容量作为窗口大小，但是如果大于UINT16_MAX，就将窗口大小设置为UINT_16MAX。

测试结果:

```
11/29 Test #12: reassembler_dup ..................   Passed    0.10 sec
      Start 13: reassembler_holes
12/29 Test #13: reassembler_holes ................   Passed    0.04 sec
      Start 14: reassembler_overlapping
13/29 Test #14: reassembler_overlapping ..........   Passed    0.06 sec
      Start 15: reassembler_win
14/29 Test #15: reassembler_win ..................   Passed    0.91 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    0.02 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    1.69 sec
      Start 20: wrapping_integers_extra
19/29 Test #20: wrapping_integers_extra ..........   Passed    0.25 sec
      Start 21: recv_connect
20/29 Test #21: recv_connect .....................   Passed    0.02 sec
      Start 22: recv_transmit
21/29 Test #22: recv_transmit ....................   Passed    0.41 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    2.48 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    1.02 sec
      Start 38: byte_stream_speed_test
             ByteStream throughput: 1.57 Gbit/s
28/29 Test #38: byte_stream_speed_test ...........   Passed    0.27 sec
      Start 39: reassembler_speed_test
             Reassembler throughput: 2.71 Gbit/s
29/29 Test #39: reassembler_speed_test ...........   Passed    0.41 sec

100% tests passed, 0 tests failed out of 29
```