Checkpoint 1 Writeup

My name: 赖炜欣

My Student ID: 205220026

I did attend the lab session.

```
private:
  ByteStream output_;
  vector<reassembler_item> buffer_;
  uint64_t unassembled_bytes;
  uint64_t current_index_;
};
```

buffer_是用来存储未组装的数据片段。
unassembled_bytes是用来记录buffer_中所有数据片段的总未组装字节的节数。
current_index_是用来记录已组装并输出的数据的结束索引。

```
uint64_t capacity = output_.writer().available_capacity();
uint64_t left_bound = max(first_index, current_index_);
uint64_t right_bound = min(current_index_ + capacity, first_index + data.size());
if (right_bound < left_bound) {
    return;
}
```

capacity用来记录能获取ByteStream的可用容量。
left_bound和right_bound在这里的作用就是一个边界，对于left_bound是first_index和current_index_中的最大值，但是不能早于current_index_，right_bound取current_index_ + capacity和first_index+data.size()中的最小值，但不能超出ByStream的可用容量或数据的末尾。

```
reassembler_item item = reassembler_item(
    data.substr(left_bound - first_index, right_bound - left_bound),
    left_bound,
    right_bound,
    is_last_substring && right_bound == first_index + data.size()
);
```

创建新的数据片段。

```
unassembled_bytes += item.data.size();
```

更新未组装的字节数。

```
auto insert_iter = buffer_.begin();
while (insert_iter != buffer_.end() && insert_iter->first_index < item.first_index) {
++insert_iter;
```

找到新片段插入的位置。

```
auto iter = insert_iter;
while (iter != buffer_.end() && item.last_index >= iter->first_index) {
    if (item.last_index < iter->last_index) {
        item.data += iter->data.substr(item.last_index - iter->first_index);
        unassembled_bytes -= item.last_index - iter->first_index;
        item.last_index = iter->last_index;
        item.is_last |= iter->is_last;
    } else {
        unassembled_bytes -= iter->data.size();
    }
    iter = buffer_.erase(iter);
}
```

处理重叠的片段，可以分成两个情况部分重叠或者完全重叠。
对于部分重叠，利用if判断，如果新片段部分覆盖了当前片段，首先将当前片段中新片段未覆盖的数据追加到新片段的末尾，然后减少unassembled_bytes，更新新片段的所以就行。
对于完全覆盖，直接更新未组装的字节数。
对于这两个情况都需要删除已经处理的片段。

```
if (insert_iter != buffer_.begin()) {
    iter = insert_iter - 1;
    if (iter->last_index >= item.first_index) {
        if (iter->last_index >= item.last_index) {
            unassembled_bytes -= item.data.size();
        } else {
           iter->data += item.data.substr(iter->last_index - item.first_index);
           unassembled_bytes -= iter->last_index - item.first_index;
           iter->last_index = item.last_index;
           iter->is_last |= item.is_last;
        }
        return;
    }
}
```

这个部分是处理合并阶段，有两种情况，一个是前一个片段完全覆盖了新片段或新片段和前一个片段部分重叠，需要合并。
对于第一个情况，如果完全覆盖了，新片段的数据已经被包含在前一个片段里了，所以只需要减少unassembled_bytes就可以了
对于第二个情况，因为只有部分重叠所以需要合并，将新片段未被前一个片段所覆盖的树蕨加到前一个片段里，然后更新数据且减少unassembled_bytes.

```
buffer_.insert(insert_iter, item);
```

插入新片段。

```
if (buffer_[0].first_index == current_index_) {
    auto& to_write_item = buffer_[0];
    output_.writer().push(to_write_item.data);
    unassembled_bytes -= to_write_item.data.size();
    current_index_ = to_write_item.last_index;
    if (to_write_item.is_last) {
        output_.writer().close();
    }
    buffer_.erase(buffer_.begin());
}
```

用于输出。

```
unassembled_bytes += item.data.size();
```

用于返回当前未组装的总结字数。

测试结果:

```
wx@wx-virtual-machine:~/Desktop/CN_lab$ cmake --build build --target check1
Test project /home/wx/Desktop/CN_lab/build
      Start  1: compile with bug-checkers
 1/17 Test  #1: compile with bug-checkers ........   Passed   13.78 sec
      Start  3: byte_stream_basics
 2/17 Test  #3: byte_stream_basics ...............   Passed    0.04 sec
      Start  4: byte_stream_capacity
 3/17 Test  #4: byte_stream_capacity .............   Passed    0.02 sec
      Start  5: byte_stream_one_write
 4/17 Test  #5: byte_stream_one_write ............   Passed    0.02 sec
      Start  6: byte_stream_two_writes
 5/17 Test  #6: byte_stream_two_writes ...........   Passed    0.02 sec
      Start  7: byte_stream_many_writes
 6/17 Test  #7: byte_stream_many_writes ..........   Passed    0.12 sec
      Start  8: byte_stream_stress_test
 7/17 Test  #8: byte_stream_stress_test ..........   Passed    0.04 sec
      Start  9: reassembler_single
 8/17 Test  #9: reassembler_single ...............   Passed    0.03 sec
      Start 10: reassembler_cap
 9/17 Test #10: reassembler_cap ..................   Passed    0.02 sec
      Start 11: reassembler_seq
10/17 Test #11: reassembler_seq ..................   Passed    0.04 sec
      Start 12: reassembler_dup
11/17 Test #12: reassembler_dup ..................   Passed    0.07 sec
      Start 13: reassembler_holes
12/17 Test #13: reassembler_holes ................   Passed    0.03 sec
      Start 14: reassembler_overlapping
13/17 Test #14: reassembler_overlapping ..........   Passed    0.04 sec
      Start 15: reassembler_win
14/17 Test #15: reassembler_win ..................   Passed    2.43 sec
      Start 37: compile with optimization
15/17 Test #37: compile with optimization ........   Passed    3.39 sec
      Start 38: byte_stream_speed_test
             ByteStream throughput: 1.07 Gbit/s
16/17 Test #38: byte_stream_speed_test ...........   Passed    0.34 sec
      Start 39: reassembler_speed_test
            Reassembler throughput: 1.92 Gbit/s
17/17 Test #39: reassembler_speed_test ...........   Passed    0.52 sec

100% tests passed, 0 tests failed out of 17
```


Implementation Challenges:

一直卡在时间复杂度上面，使用了vector以后就减少了。