Checkpoint 1 Writeup

My name: 耿天成

My SUNet ID: 221900006

I was surprised by or edified to learn that: string.substr(pos, count)

Describe Reassembler structure and design.

Describe data structures and approach taken.

用 `std::map<uint64_t, std::string> mdata` 保存 reassembler 的数据. 不变量:

1. `bytes_pending() < availiable_capacity()`
2. `bytes_pending() = mdata` 中所有 `string` 的长度之和.
3. 若 `&[i, is]` 和 `&[j, js]` 是 `mdata` 中相邻迭代器, 则 `i + is.size() <= j`.
4. 若 `mdata` 非空, `bytes_pushed() < mdata.begin()->first`.
5. 由 3, `mdata` 可导出一个维护的区间 I, 和函数 $m : I \to \langle chars \rangle$. 则 $m(i) = c$ 当且仅当存在某次 `insert(fi, data, ...)` 满足 `i >= bytes_pushed()` $\wedge$ `data[i - fi] = c`.
6. `bytes_pushed() == N` $\to$ `is_closed()`. N 是最后一块数据的末尾下标.

核心是维护 (3), 即维持 `mdata` 中字符串不交叉. 需要

1. 删除被 `data` 覆盖的字符串,
2. 插入 `data` 截断至 [prev.right, next.left).

(6) 的 corner case. `bytes_pushed()` 改变和 `N` 知晓后都需要检查关闭流条件.

Describe alternative designs considered or tested.

考虑了优先队列, 这样会重复存储字符串; 看到网上有类似插入排序的方案, 效率差不多.

Describe benefits and weaknesses of your design compared with alternatives –perhaps in terms of simplicity/complexity, risk of bugs, asymptotic performance, empirical performance, required implementation time and difficulty, and other factors. Include any measurements if applicable. 暂无.

Implementation Challenges: 正确性.

Remaining Bugs: 未知.

```
~/playground/minnow main ?2 > cmake --build build --target check1      08:39:05
Test project /home/isapo/playground/minnow/build
      Start  1: compile with bug-checkers
 1/17 Test  #1: compile with bug-checkers ........   Passed    0.15 sec
      Start  3: byte_stream_basics
 2/17 Test  #3: byte_stream_basics ...............   Passed    0.01 sec
      Start  4: byte_stream_capacity
 3/17 Test  #4: byte_stream_capacity .............   Passed    0.01 sec
      Start  5: byte_stream_one_write
 4/17 Test  #5: byte_stream_one_write ............   Passed    0.01 sec
      Start  6: byte_stream_two_writes
 5/17 Test  #6: byte_stream_two_writes ...........   Passed    0.01 sec
      Start  7: byte_stream_many_writes
 6/17 Test  #7: byte_stream_many_writes ..........   Passed    0.06 sec
      Start  8: byte_stream_stress_test
 7/17 Test  #8: byte_stream_stress_test ..........   Passed    0.06 sec
      Start  9: reassembler_single
 8/17 Test  #9: reassembler_single ...............   Passed    0.01 sec
      Start 10: reassembler_cap
 9/17 Test #10: reassembler_cap ..................   Passed    0.01 sec
      Start 11: reassembler_seq
10/17 Test #11: reassembler_seq ..................   Passed    0.01 sec
      Start 12: reassembler_dup
11/17 Test #12: reassembler_dup ..................   Passed    0.05 sec
      Start 13: reassembler_holes
12/17 Test #13: reassembler_holes ................   Passed    0.01 sec
      Start 14: reassembler_overlapping
13/17 Test #14: reassembler_overlapping ..........   Passed    0.01 sec
      Start 15: reassembler_win
14/17 Test #15: reassembler_win ..................   Passed    0.17 sec
      Start 37: compile with optimization
15/17 Test #37: compile with optimization ........   Passed    0.08 sec
      Start 38: byte_stream_speed_test
            ByteStream throughput: 21.42 Gbit/s
16/17 Test #38: byte_stream_speed_test ...........   Passed    0.05 sec
      Start 39: reassembler_speed_test
            Reassembler throughput: 11.98 Gbit/s
17/17 Test #39: reassembler_speed_test ...........   Passed    0.09 sec

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

Figure 1: check1 result