

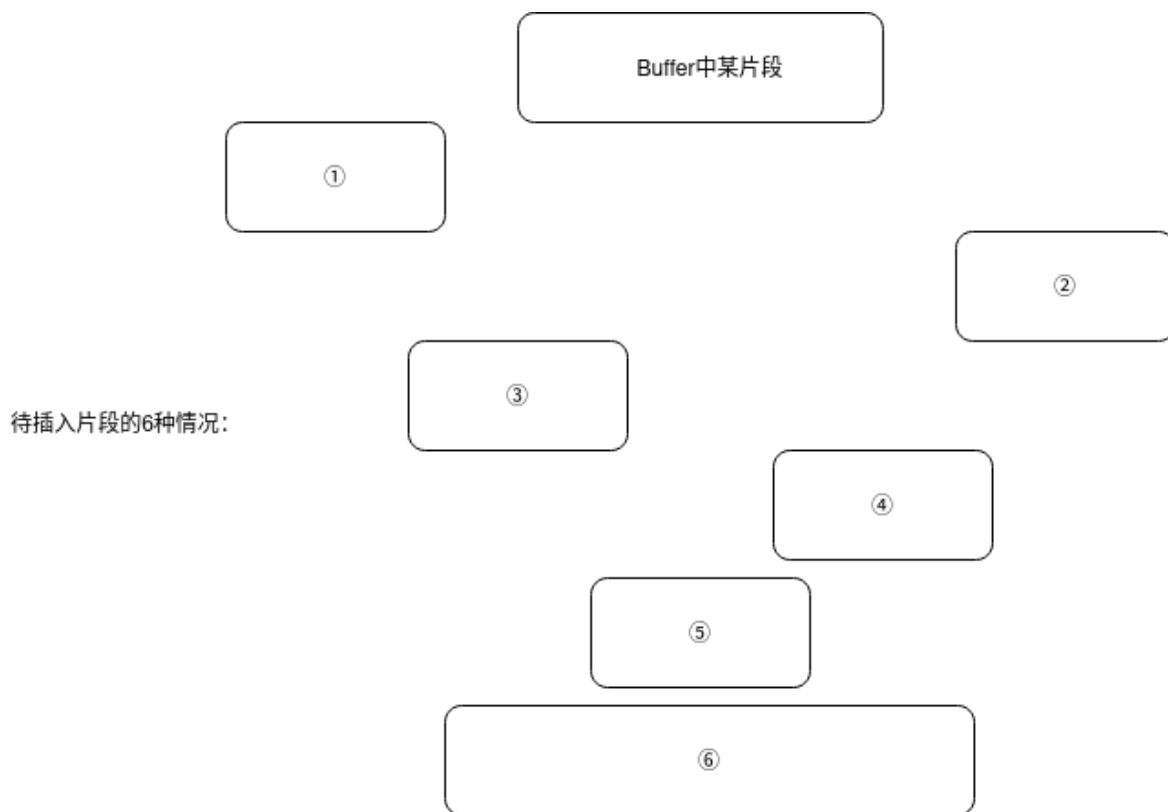
1. Program Structure and Design

1. `void Reassembler::insert()` 函数:

使用 `vector<pair<uint64_t, string>` 类型作为 `buffer` 的数据结构。

- 优点: 存储的数据可以按 `first_index` 排为有序
- 缺点: `vector` 的插入和删除操作均为 $O(n)$ 复杂度

如图所示为待插入片段与 `buffer` 中某片段的6种交叉关系示意图:



根据六种情况, 实行不同的策略。

- ①: 直接插入, `break;`
- ②: 跳到下一个片段
- ③: 合并为整体, 并删除 `Buffer` 中片段
- ④: 合并为整体, 并删除 `Buffer` 中片段
- ⑤: 不插入, `break;`
- ⑥: 删除 `Buffer` 中片段

完成插入操作后, 开始写入 `ByteStream`, 判断 `Buffer` 中片段的 `first_index <= first_unassembled_index` 即可选择插入, 并删除 `Buffer` 中片段。

2. `uint64_t Reassembler::bytes_pending()` 函数:

逐个遍历 `Buffer` 中片段, 将其长度求和即可。

3. `void Reassembler::closeCheck()` 函数:

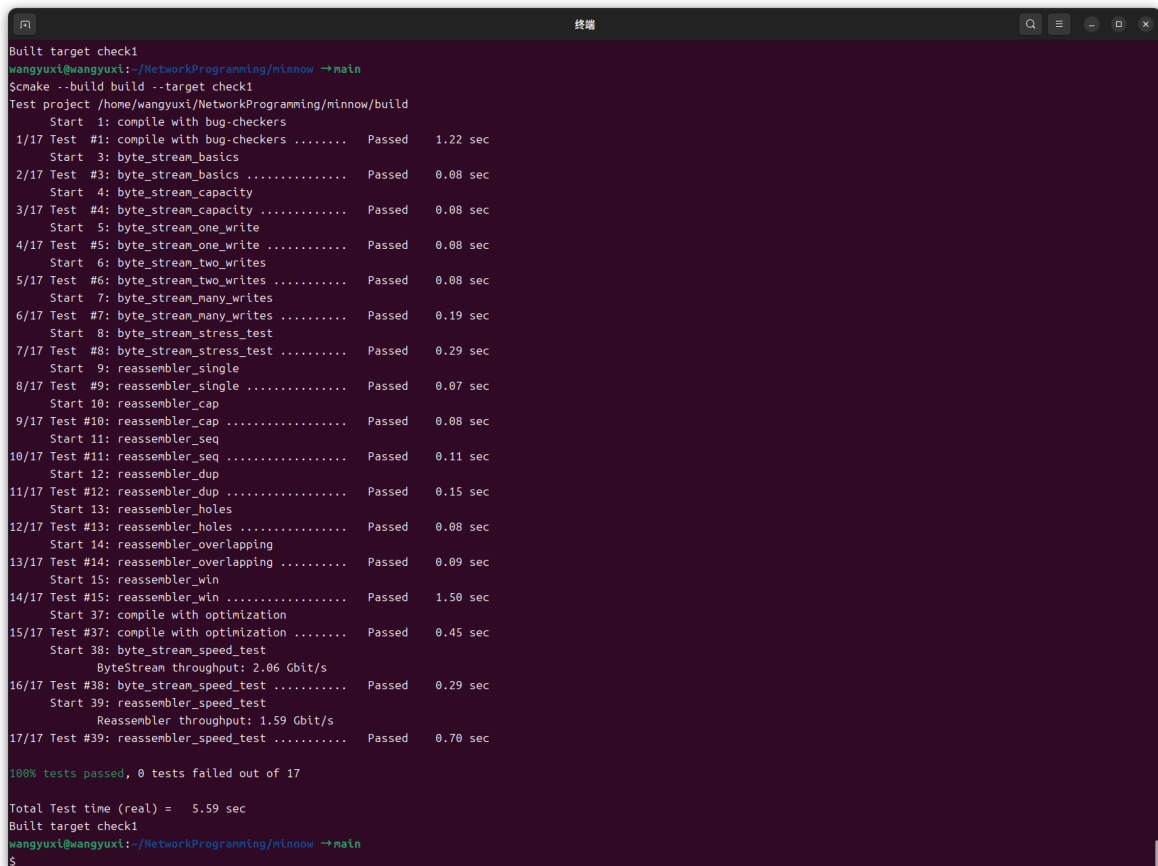
新增的非框架函数，用于检查关闭数据流，需要在 `Reassembler` 类中额外添加成员变量 `endingIdx_`，记录最后一片长度的末尾，当 `endingIdx_==first_unassembled_index` 时，关闭数据流。

2.Implementation Challenges

1. 选择数据结构时，在 `map` 与 `vector` 之间选择了 `vector`，因为其有序，在片段合成过程中，逻辑性更清晰，但速度较 `map` 会慢。
2. 在罗列所有种情况，并用位置关系表达的时候，会出现不严谨以及各类型出现交集的情况，需要再仔细一点。
3. 在 `cmake --build build --target check0` 运行测试程序时，总会报 `AddressSanitizer:DEADLYSIGNAL` 错误，在qq群的FAQ文件中找到了解决方法：终端输入 `echo 0 | sudo tee /proc/sys/kernel/randomize_va_space`，禁止linux地址空间布局随机化ASLR。

3. Experimental results and performance

测试截图：



```

Built target check1
wangyuxi@wangyuxi:~/NetworkProgramming/minnow -> main
$ cmake --build build --target check1
Test project /home/wangyuxi/NetworkProgramming/minnow/build
  Start 1: compile with bug-checkers
1/17 Test #1: compile with bug-checkers ..... Passed    1.22 sec
  Start 3: byte_stream_basics
2/17 Test #3: byte_stream_basics ..... Passed    0.08 sec
  Start 4: byte_stream_capacity
3/17 Test #4: byte_stream_capacity ..... Passed    0.08 sec
  Start 5: byte_stream_one_write
4/17 Test #5: byte_stream_one_write ..... Passed    0.08 sec
  Start 6: byte_stream_two_writes
5/17 Test #6: byte_stream_two_writes ..... Passed    0.08 sec
  Start 7: byte_stream_many_writes
6/17 Test #7: byte_stream_many_writes ..... Passed    0.19 sec
  Start 8: byte_stream_stress_test
7/17 Test #8: byte_stream_stress_test ..... Passed    0.29 sec
  Start 9: reassembler_single
8/17 Test #9: reassembler_single ..... Passed    0.07 sec
  Start 10: reassembler_cap
9/17 Test #10: reassembler_cap ..... Passed    0.08 sec
  Start 11: reassembler_seq
10/17 Test #11: reassembler_seq ..... Passed    0.11 sec
  Start 12: reassembler_dup
11/17 Test #12: reassembler_dup ..... Passed    0.15 sec
  Start 13: reassembler_holes
12/17 Test #13: reassembler_holes ..... Passed    0.08 sec
  Start 14: reassembler_overlapping
13/17 Test #14: reassembler_overlapping ..... Passed    0.09 sec
  Start 15: reassembler_win
14/17 Test #15: reassembler_win ..... Passed    1.50 sec
  Start 37: compile with optimization
15/17 Test #37: compile with optimization ..... Passed    0.45 sec
  Start 38: byte_stream_speed_test
        ByteStream throughput: 2.06 Gbit/s
16/17 Test #38: byte_stream_speed_test ..... Passed    0.29 sec
  Start 39: reassembler_speed_test
        Reassembler throughput: 1.59 Gbit/s
17/17 Test #39: reassembler_speed_test ..... Passed    0.70 sec

100% tests passed, 0 tests failed out of 17

Total Test time (real) =  5.59 sec
Built target check1
wangyuxi@wangyuxi:~/NetworkProgramming/minnow -> main
$
```