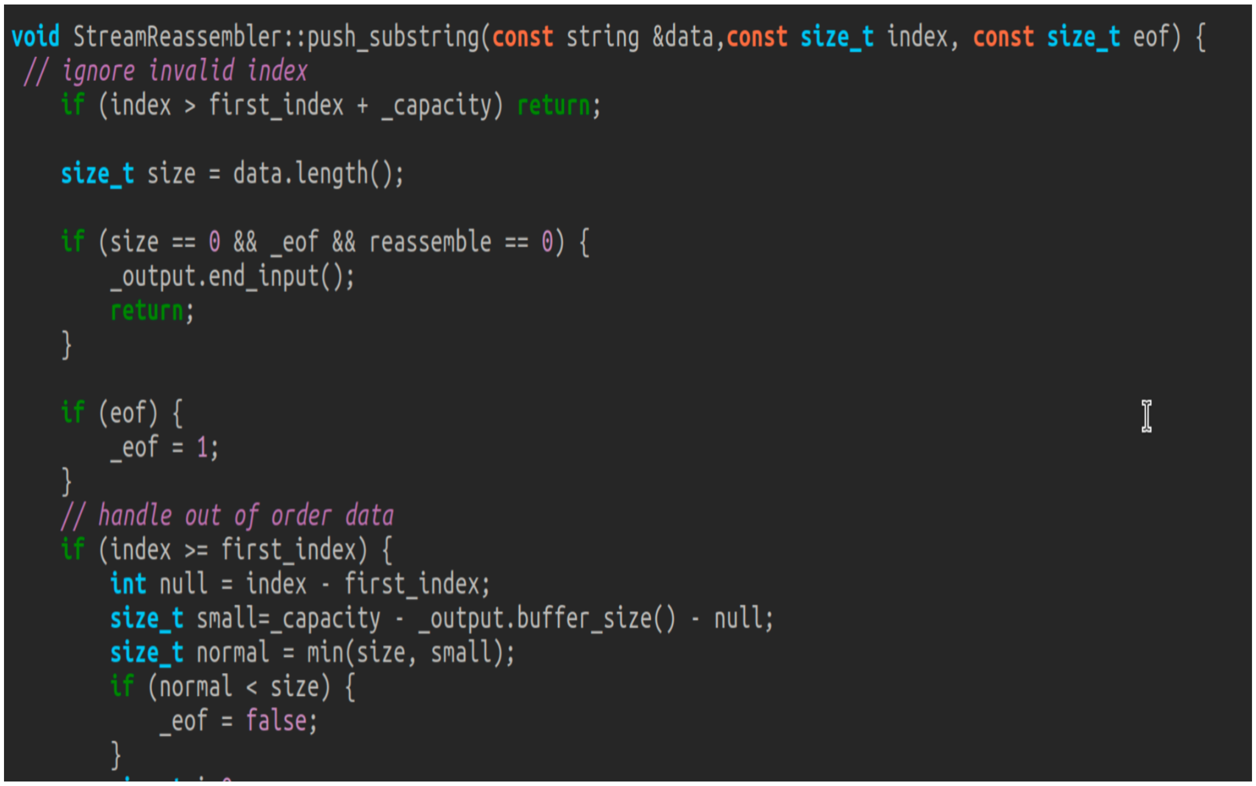
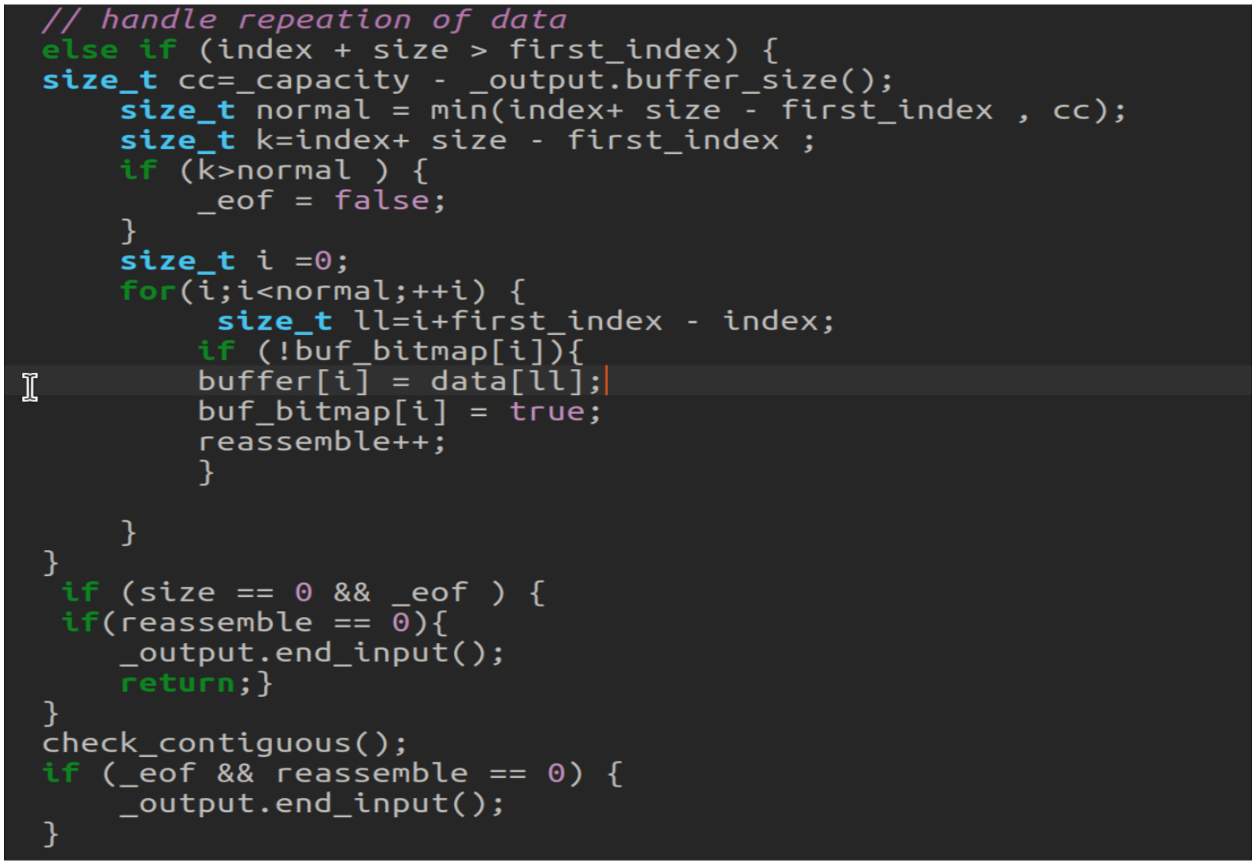
CN assignment 2

Aryan Sharma 2021025

Part 2: Stream Reassembler:
Function: push_substring



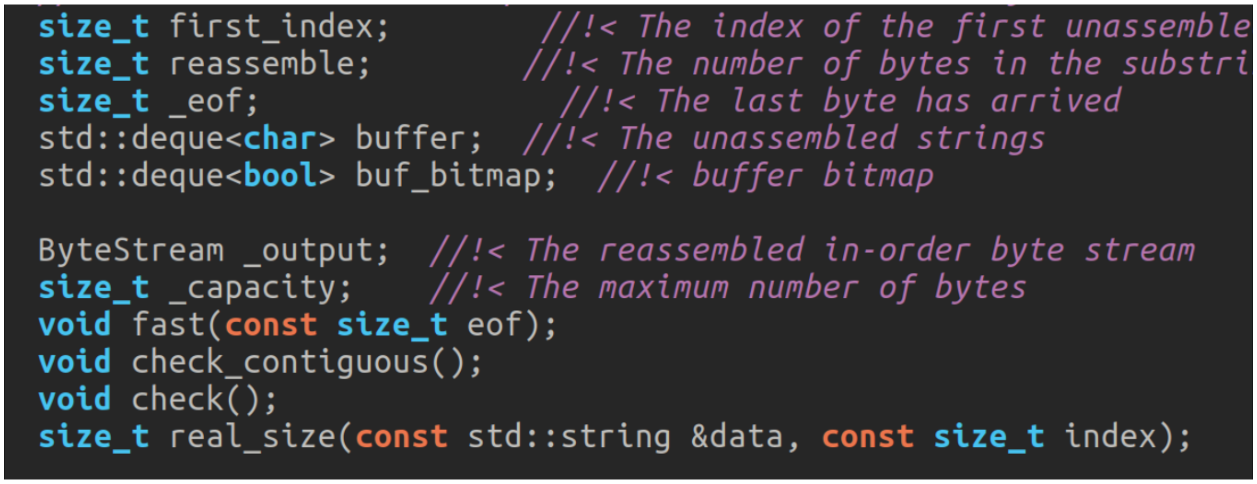
```
void StreamReassembler::push_substring(const string &data,const size_t index, const size_t eof) {
 // ignore invalid index
    if (index > first_index + _capacity) return;

    size_t size = data.length();

    if (size == 0 && _eof && reassemble == 0) {
        _output.end_input();
        return;
    }

    if (eof) {
        _eof = 1;
    }
    // handle out of order data
    if (index >= first_index) {
        int null = index - first_index;
        size_t small=_capacity - _output.buffer_size() - null;
        size_t normal = min(size, small);
        if (normal < size) {
            _eof = false;
        }
```



```
// handle repeation of data
else if (index + size > first_index) {
size_t cc=_capacity - _output.buffer_size();
    size_t normal = min(index+ size - first_index , cc);
    size_t k=index+ size - first_index ;
    if (k>normal ) {
        _eof = false;
    }
    size_t i =0;
    for(i;i<normal;++i) {
         size_t ll=i+first_index - index;
        if (!buf_bitmap[i]){
        buffer[i] = data[ll];
        buf_bitmap[i] = true;
        reassemble++;
        }

    }
}
 if (size == 0 && _eof ) {
 if(reassemble == 0){
    _output.end_input();
    return;}
}
check_contiguous();
if (_eof && reassemble == 0) {
    _output.end_input();
}
```

-Code handles the edge cases like repeated data, invalid index, out of order data and it successfully reassembles the string.

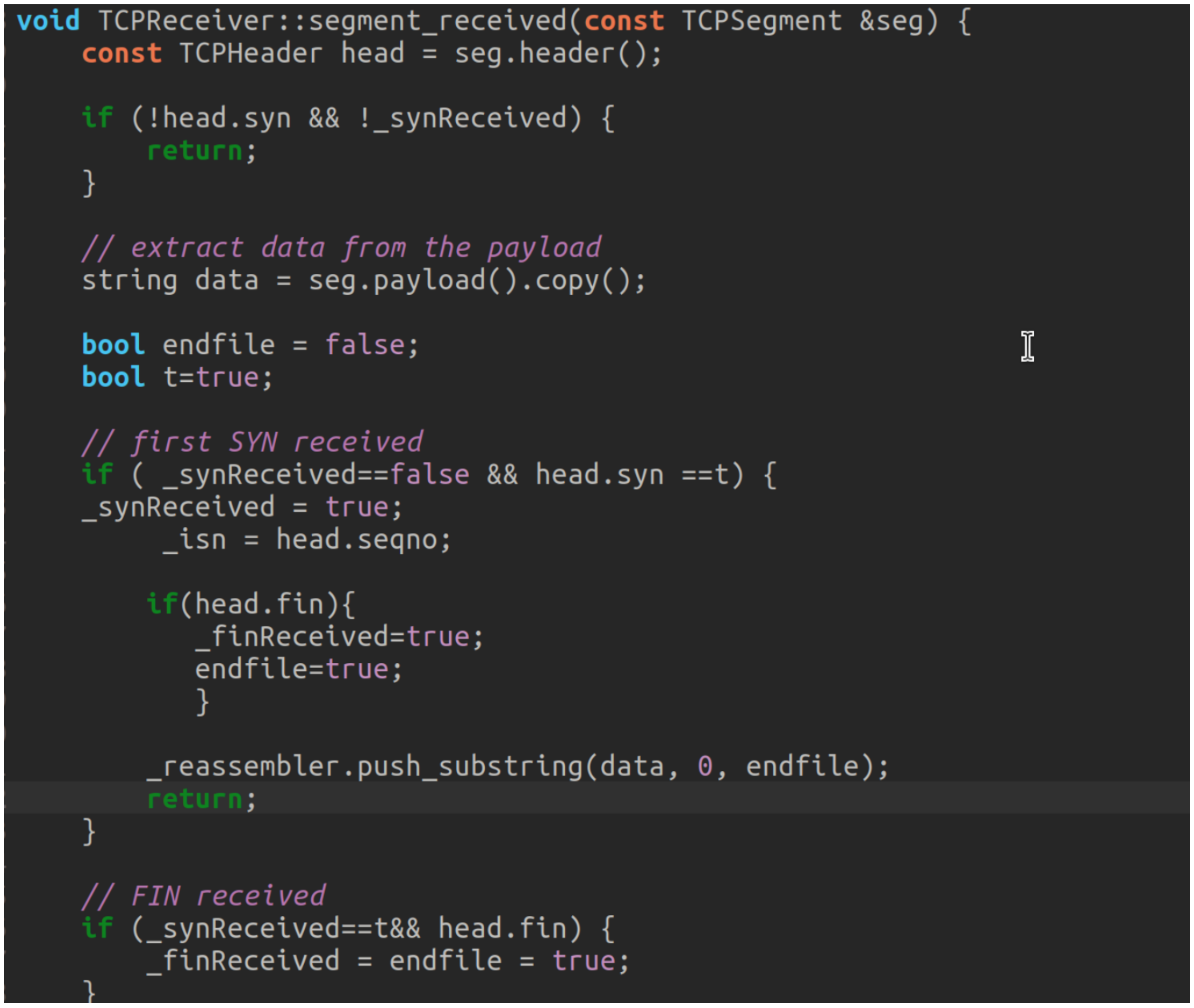
Stream_reassembler.hh



```
size_t first_index;        //!< The index of the first unassemble
size_t reassemble;        //!< The number of bytes in the substri
size_t _eof;                //!< The last byte has arrived
std::deque<char> buffer;  //!< The unassembled strings
std::deque<bool> buf_bitmap;  //!< buffer bitmap

ByteStream _output;  //!< The reassembled in-order byte stream
size_t _capacity;    //!< The maximum number of bytes
void fast(const size_t eof);
void check_contiguous();
void check();
size_t real_size(const std::string &data, const size_t index);
```

All the private variables are declared in the Stream_reassembler.hh file.

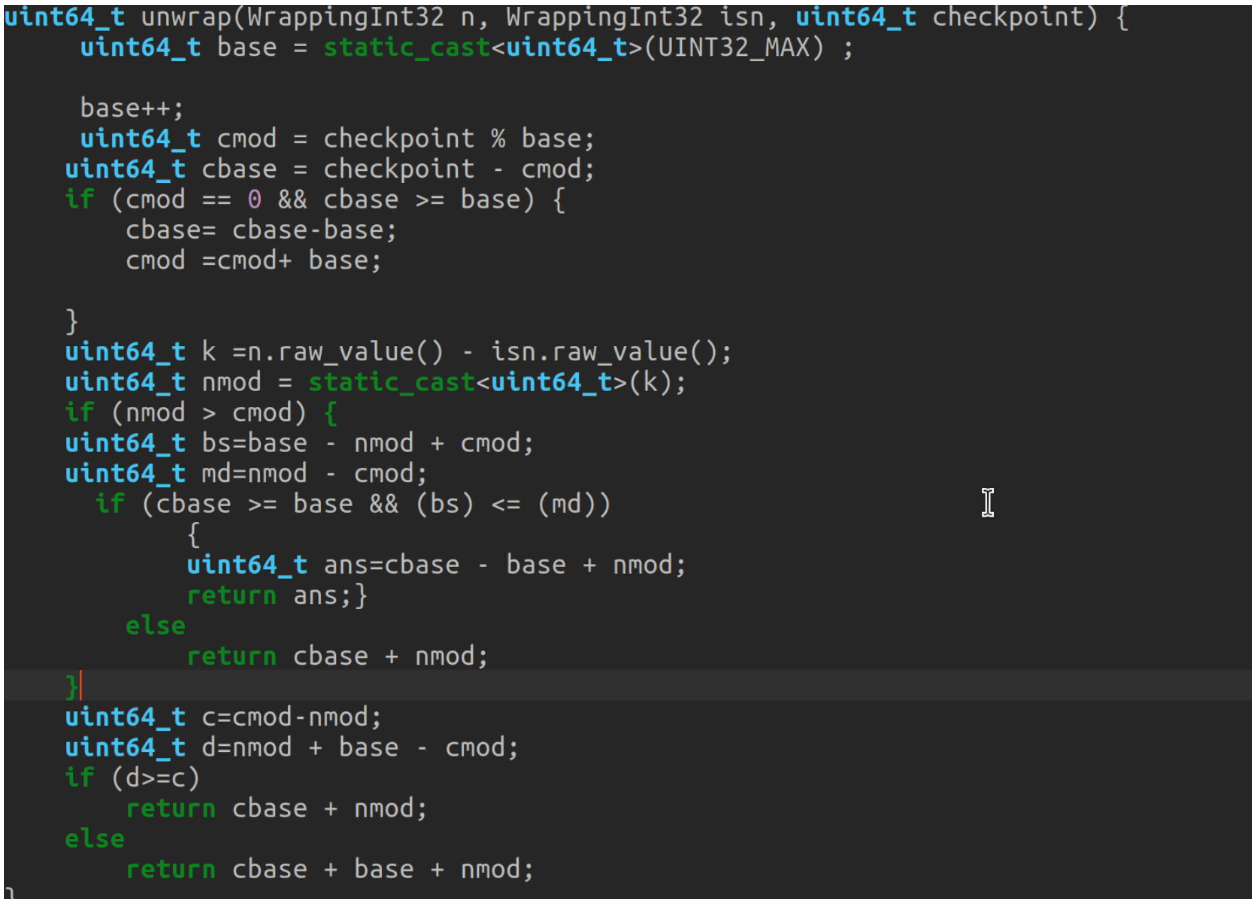
TCP Receiver



```
void TCPReceiver::segment_received(const TCPSegment &seg) {
    const TCPHeader head = seg.header();

    if (!head.syn && !_synReceived) {
        return;
    }

    // extract data from the payload
    string data = seg.payload().copy();

    bool endfile = false;
    bool t=true;

    // first SYN received
    if ( _synReceived==false && head.syn ==t) {
    _synReceived = true;
         _isn = head.seqno;

        if(head.fin){
           _finReceived=true;
           endfile=true;
           }

        _reassembler.push_substring(data, 0, endfile);
        return;
    }

    // FIN received
    if (_synReceived==t&& head.fin) {
        _finReceived = endfile = true;
    }
```

TCP Receiver: takes TCP Segments from the sender and returns it along with an unassembled base index then the unassembled data is transmitted to the reassembler, which reassembles it.

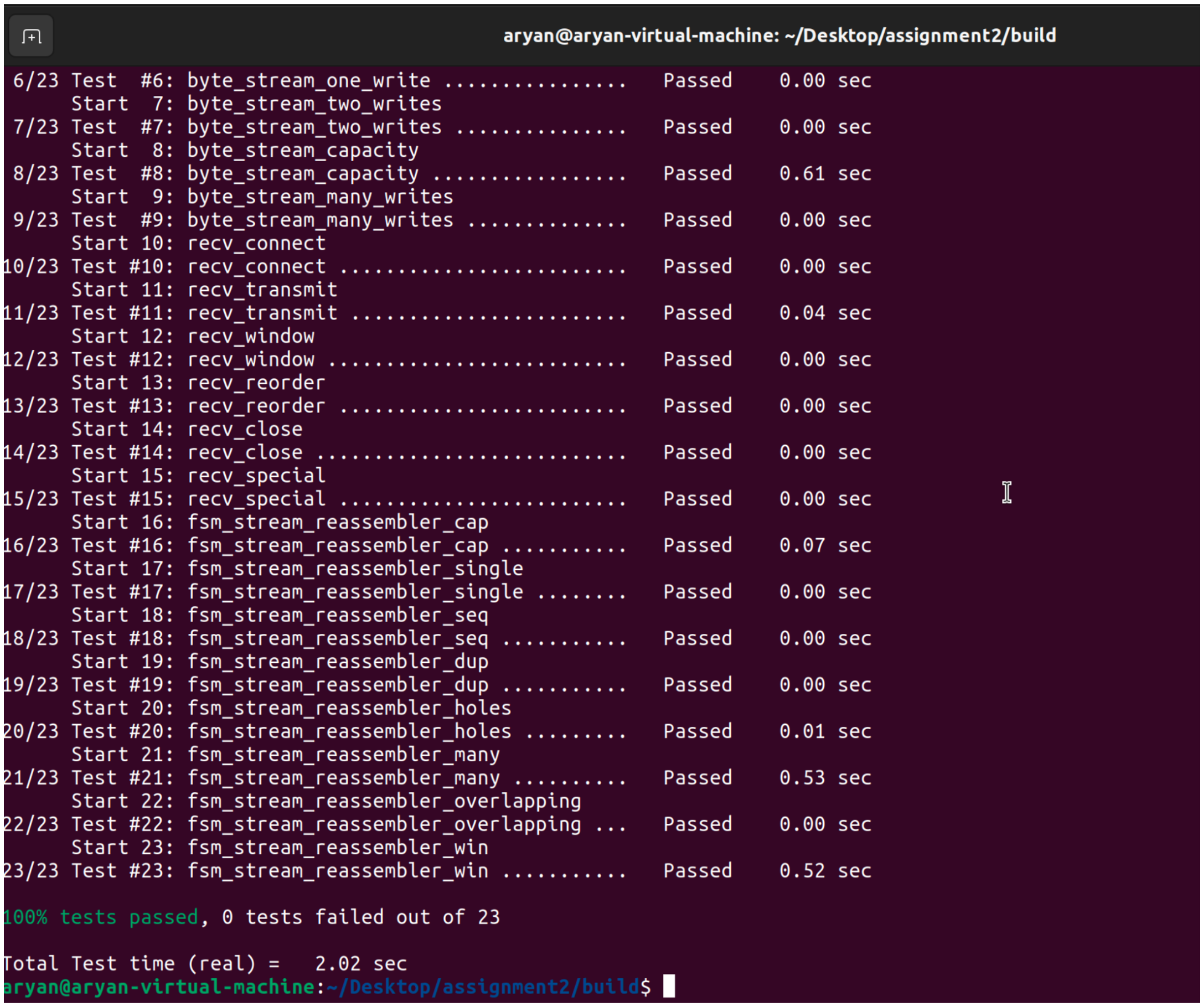
Wrapping_integer.cc



```
uint64_t unwrap(WrappingInt32 n, WrappingInt32 isn, uint64_t checkpoint) {
     uint64_t base = static_cast<uint64_t>(UINT32_MAX) ;

     base++;
     uint64_t cmod = checkpoint % base;
    uint64_t cbase = checkpoint - cmod;
    if (cmod == 0 && cbase >= base) {
        cbase= cbase-base;
        cmod =cmod+ base;

    }
    uint64_t k =n.raw_value() - isn.raw_value();
    uint64_t nmod = static_cast<uint64_t>(k);
    if (nmod > cmod) {
    uint64_t bs=base - nmod + cmod;
    uint64_t md=nmod - cmod;
      if (cbase >= base && (bs) <= (md))
            {
            uint64_t ans=cbase - base + nmod;
            return ans;}
        else
            return cbase + nmod;
    }
    uint64_t c=cmod-nmod;
    uint64_t d=nmod + base - cmod;
    if (d>=c)
        return cbase + nmod;
    else
        return cbase + base + nmod;
```

Wrapping: TCP uses a random number generator to assign sequence numbers to each segment. This is done to make it more difficult for attackers to guess the sequence numbers and to prevent confusion with older segments that may still be in transit.

OUTPUT At Terminal:



aryan@aryan-virtual-machine: ~/Desktop/assignment2/build

```
 6/23 Test  #6: byte_stream_one_write ................   Passed    0.00 sec
      Start  7: byte_stream_two_writes
 7/23 Test  #7: byte_stream_two_writes ...............   Passed    0.00 sec
      Start  8: byte_stream_capacity
 8/23 Test  #8: byte_stream_capacity .................   Passed    0.61 sec
      Start  9: byte_stream_many_writes
 9/23 Test  #9: byte_stream_many_writes ..............   Passed    0.00 sec
      Start 10: recv_connect
10/23 Test #10: recv_connect .........................   Passed    0.00 sec
      Start 11: recv_transmit
11/23 Test #11: recv_transmit ........................   Passed    0.04 sec
      Start 12: recv_window
12/23 Test #12: recv_window ..........................   Passed    0.00 sec
      Start 13: recv_reorder
13/23 Test #13: recv_reorder .........................   Passed    0.00 sec
      Start 14: recv_close
14/23 Test #14: recv_close ...........................   Passed    0.00 sec
      Start 15: recv_special
15/23 Test #15: recv_special .........................   Passed    0.00 sec
      Start 16: fsm_stream_reassembler_cap
16/23 Test #16: fsm_stream_reassembler_cap ...........   Passed    0.07 sec
      Start 17: fsm_stream_reassembler_single
17/23 Test #17: fsm_stream_reassembler_single ........   Passed    0.00 sec
      Start 18: fsm_stream_reassembler_seq
18/23 Test #18: fsm_stream_reassembler_seq ...........   Passed    0.00 sec
      Start 19: fsm_stream_reassembler_dup
19/23 Test #19: fsm_stream_reassembler_dup ...........   Passed    0.00 sec
      Start 20: fsm_stream_reassembler_holes
20/23 Test #20: fsm_stream_reassembler_holes .........   Passed    0.01 sec
      Start 21: fsm_stream_reassembler_many
21/23 Test #21: fsm_stream_reassembler_many ..........   Passed    0.53 sec
      Start 22: fsm_stream_reassembler_overlapping
22/23 Test #22: fsm_stream_reassembler_overlapping ...   Passed    0.00 sec
      Start 23: fsm_stream_reassembler_win
23/23 Test #23: fsm_stream_reassembler_win ...........   Passed    0.52 sec

100% tests passed, 0 tests failed out of 23

Total Test time (real) =   2.02 sec
aryan@aryan-virtual-machine:~/Desktop/assignment2/build$
```