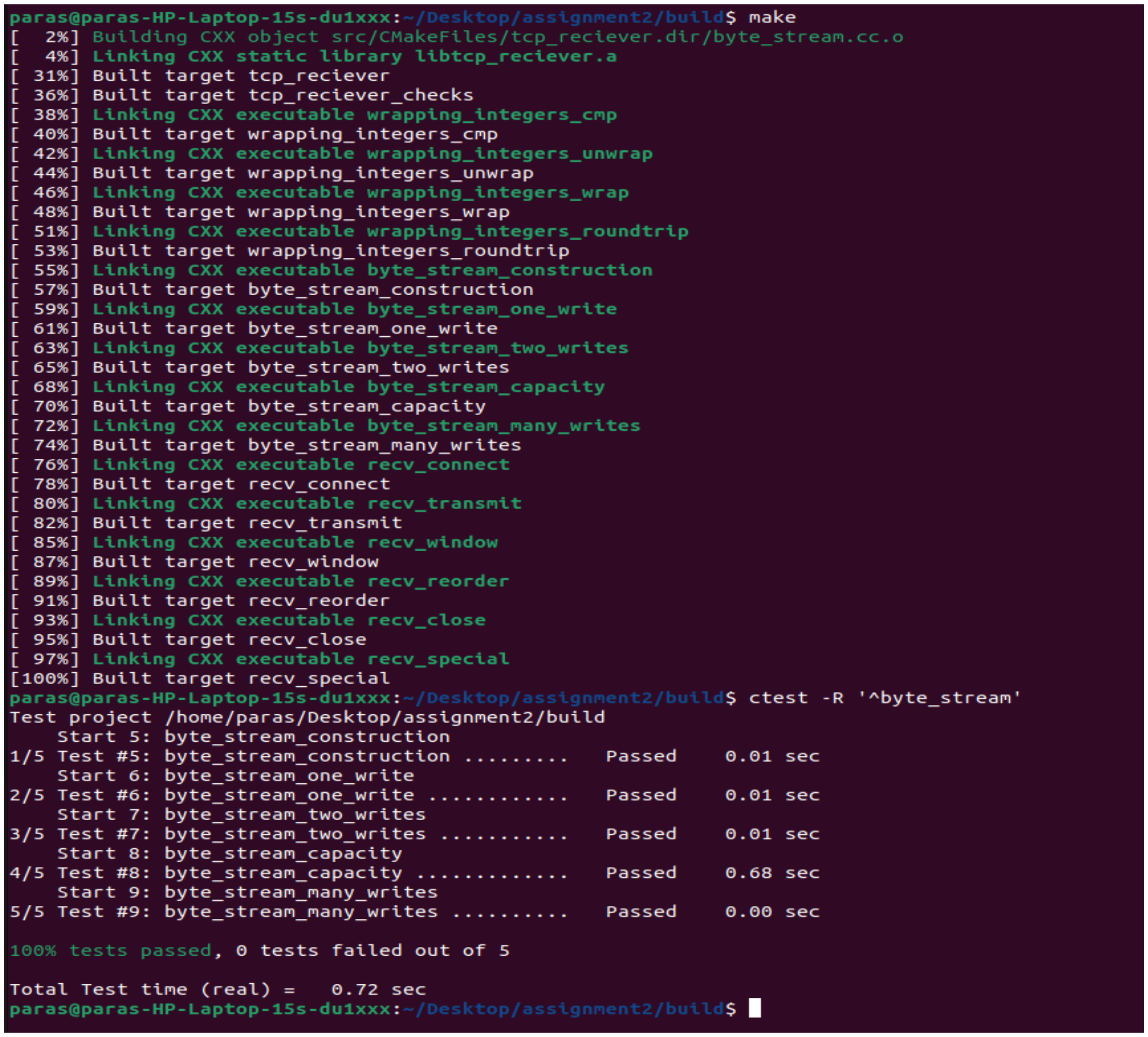
Indraprastha Institute of Information Technology Delhi (IIITD)

ASSIGNMENT-2

Name-PARAS DHIMAN || Roll no-2021482

Computer Networks - CSE232

Question :



```
paras@paras-HP-Laptop-15s-du1xxx:~/Desktop/assignment2/build$ make
[  2%] Building CXX object src/CMakeFiles/tcp_reciever.dir/byte_stream.cc.o
[  4%] Linking CXX static library libtcp_reciever.a
[ 31%] Built target tcp_reciever
[ 36%] Built target tcp_reciever_checks
[ 38%] Linking CXX executable wrapping_integers_cmp
[ 40%] Built target wrapping_integers_cmp
[ 42%] Linking CXX executable wrapping_integers_unwrap
[ 44%] Built target wrapping_integers_unwrap
[ 46%] Linking CXX executable wrapping_integers_wrap
[ 48%] Built target wrapping_integers_wrap
[ 51%] Linking CXX executable wrapping_integers_roundtrip
[ 53%] Built target wrapping_integers_roundtrip
[ 55%] Linking CXX executable byte_stream_construction
[ 57%] Built target byte_stream_construction
[ 59%] Linking CXX executable byte_stream_one_write
[ 61%] Built target byte_stream_one_write
[ 63%] Linking CXX executable byte_stream_two_writes
[ 65%] Built target byte_stream_two_writes
[ 68%] Linking CXX executable byte_stream_capacity
[ 70%] Built target byte_stream_capacity
[ 72%] Linking CXX executable byte_stream_many_writes
[ 74%] Built target byte_stream_many_writes
[ 76%] Linking CXX executable recv_connect
[ 78%] Built target recv_connect
[ 80%] Linking CXX executable recv_transmit
[ 82%] Built target recv_transmit
[ 85%] Linking CXX executable recv_window
[ 87%] Built target recv_window
[ 89%] Linking CXX executable recv_reorder
[ 91%] Built target recv_reorder
[ 93%] Linking CXX executable recv_close
[ 95%] Built target recv_close
[ 97%] Linking CXX executable recv_special
[100%] Built target recv_special
paras@paras-HP-Laptop-15s-du1xxx:~/Desktop/assignment2/build$ ctest -R '^byte_stream'
Test project /home/paras/Desktop/assignment2/build
    Start 5: byte_stream_construction
1/5 Test #5: byte_stream_construction .........   Passed    0.01 sec
    Start 6: byte_stream_one_write
2/5 Test #6: byte_stream_one_write ............   Passed    0.01 sec
    Start 7: byte_stream_two_writes
3/5 Test #7: byte_stream_two_writes ...........   Passed    0.01 sec
    Start 8: byte_stream_capacity
4/5 Test #8: byte_stream_capacity .............   Passed    0.68 sec
    Start 9: byte_stream_many_writes
5/5 Test #9: byte_stream_many_writes ..........   Passed    0.00 sec

100% tests passed, 0 tests failed out of 5

Total Test time (real) =   0.72 sec
paras@paras-HP-Laptop-15s-du1xxx:~/Desktop/assignment2/build$
```

Building a TCP Receiver - ByteStream

Introduction

The objective of this assignment is to build a TCP receiver, which involves three primary components:

1. Building the `ByteStream` class.
2. Building a reassembler to assemble segments received from the sender in the correct order.
3. Building the actual TCP receiver by stitching together the `ByteStream` and reassembler.

In this report, we will focus on Part I, which is the implementation and explanation of the `ByteStream` class. The `ByteStream` class is responsible for managing a collection of bytes in an ordered manner, allowing bytes to be written on the input side and read from the output side.

Part I: Building ByteStream

Properties of ByteStream

1. **Bytes Management:** The `ByteStream` class manages bytes, allowing writing from one side and reading from the other.
2. **Finite Stream:** The stream is finite, and once the writer ends input, no more bytes can be written.
3. **End of File (EOF):** When the reader reaches the end of the stream, it encounters EOF, indicating no more available bytes to read.
4. **Capacity:** The stream has a capacity that limits the total amount of bytes that can be held in memory at once.
5. **Capacity Enforcement:** The writer cannot write more bytes if it exceeds the storage capacity.
6. **Continuous Operation:** The writer can continue to write as the reader reads bytes.
7. **Single-Threaded:** The `ByteStream` is used in a single-threaded context, so there are no concerns about readers, writers, locking, or race conditions.

Class Members

The `ByteStream` class has the following private members:

- `deque<char> myDeque`: A deque to store the bytes.
- `size_t _capacity`: Capacity of the stream.

- `string _buffer`: A buffer for reading.
- `size_t _bytes_read`: Count of bytes read.
- `size_t _bytes_write`: Count of bytes written.
- `bool _eof`: A flag indicating the end of the stream.
- `mutable bool _error`: A mutable flag indicating an error in the stream.

Constructor

The constructor initializes the `ByteStream` object with a given capacity.

Input Interface

- `write(const std::string &data)`: Writes a string of bytes into the stream, returning the number of bytes accepted.
- `remaining_capacity() const`: Returns the number of additional bytes that the stream has space for.
- `end_input()`: Signals the end of input.
- `set_error()`: Indicates that the stream has suffered an error.

Output Interface

- `peek_output(const size_t len) const`: Peeks at the next `len` bytes of the stream without removing them.
- `pop_output(const size_t len)`: Removes `len` bytes from the buffer.
- `read(const size_t len)`: Reads (copies and then pops) the next `len` bytes of the stream.
- `input_ended() const`: Returns `true` if the stream input has ended.
- `error() const`: Returns `true` if the stream has suffered an error.
- `buffer_size() const`: Returns the current size of the buffer.
- `buffer_empty() const`: Returns `true` if the buffer is empty.
- `eof() const`: Returns `true` if the output has reached the end.

General Accounting

- `bytes_written() const`: Returns the total number of bytes written.
- `bytes_read() const`: Returns the total number of bytes read.

Testing

To test the `ByteStream` class, we build the project using `make` and run the tests using `ctest -R '^byte_stream'`.

Conclusion

In conclusion, the `ByteStream` class is a fundamental component for building a TCP receiver. It manages bytes, enforces capacity limits, handles input and output, and keeps track of errors and the end of the stream. Proper implementation and testing of the `ByteStream` class are crucial for the overall functionality of the TCP receiver.