Checkpoint 0 Writeup

My name: 卢郡然

My SUNet ID: 502024330034

This lab took me about 4 hours to do.

Implementation of Webget

using `TCPSocket` library to send HTTP requests.

output the response to std::cout.

```
void get_URL(const string& host, const string& path )
{
  cerr << "Function called: get_URL(" << host << ", " << path << ")\n";

  // cerr << "Warning: get_URL() has not been implemented yet.\n";
  TCPSocket sock{};
  sock.connect(Address(host,"http"));
  sock.write("GET "+path+" HTTP/1.1\r\nHost: "+host+"\r\n\r\n");
  sock.shutdown(SHUT_WR);
  string response;
  while (!sock.eof()) {
    sock.read(response);
    cout<<response;
  }
}
```

Implementation of ByteSteam

Program Structure and Design

The ByteStream implementation leverages the `std::deque` container for its internal buffer, as it provides efficient operations at both ends, allowing for push and pop operations in constant time $O(1)$.

This efficiency is crucial for handling the stream-like behavior, where data is pushed into the buffer by the writer and popped out by the reader.

The writer uses the `push_back()` method of the deque to append data to the buffer. This ensures that the writer can push chunks of data efficiently, even when handling large or partial inputs.

For the reader, `pop_front()` is used to remove data from the front of the buffer, simulating the consumption of the stream. This mechanism ensures that the data is removed once it is read, respecting the natural flow of the stream.

To offer a non-intrusive view of the data for peek operations, I used std::string_view. This C++ feature allows for a read-only view into the string data, which avoids unnecessary copying of the underlying data, improving performance. Using string_view is ideal for the peek operation, where the reader needs a snapshot of the data without modifying the buffer.

Implementation Challenges

The data is stored as segments, which makes the string view inconsistant. So I need to track the front the deque while pushing and poping to obtain a correct string view. I also need to handle the case when the buffer is empty, which is not handled by the string view.

The primary challenge arose from managing the buffer's segmented data. Since the std::deque stores string in segments, the string_view cannot simply point to a contiguous block of memory. This required careful tracking of the front of the deque as data is pushed and popped, ensuring that the string_view provided to the reader is always consistent and reflects the current state of the buffer.

```
void Reader::pop( uint64_t len )
{
  num_bytes_buffered -= len;
  num_bytes_popped += len;
  while ( len >= remain.size() && len != 0 ) {
    len -= remain.size();
    buffer.pop_front();
    remain = buffer.empty() ? ""sv : buffer.front();
  }
  if ( !remain.empty() )
    remain.remove_prefix( len );
}
```

Also handling the case when the buffer is empty is a hard task. Since string_view cannot handle empty or non-existent data natively, additional checks were necessary to manage cases where the buffer was either partially or completely drained.

Managing the end-of-file (EOF) scenario required special handling. The buffer needed to recognize when the writer had closed the stream and ensure that the reader could gracefully finish reading the remaining data.

Experimental Results and Performance.

result of check_webget

```
[proc] Executing command: /home/max/miniconda3/bin/cmake --build /home/max/minnow/build
[build] [1/1 100% :: 2.930] cd /home/max/minnow/build && /home/max/miniconda3/lib/python
[build] Test project /home/max/minnow/build
[build]     Start 1: compile with bug-checkers
[build] 1/2 Test #1: compile with bug-checkers ........   Passed    1.78 sec
[build]     Start 2: t_webget
[build] 2/2 Test #2: t_webget .........................   Passed    1.11 sec
[build]
[build] 100% tests passed, 0 tests failed out of 2
[build]
[build] Total Test time (real) =   2.90 sec
[driver] Build completed: 00:00:03.066
[build] Build finished with exit code 0
```

result of check0

```
[proc] Executing command: /home/max/miniconda3/bin/cmake --build /home/max/minnow/build --config Debug --target check0 --
[build] [1/1 100% :: 3.614] cd /home/max/minnow/build && /home/max/miniconda3/lib/python3.10/site-packages/cmake/data/bin/ctest -
[build] Test project /home/max/minnow/build
[build]       Start  1: compile with bug-checkers
[build]  1/10 Test  #1: compile with bug-checkers ........   Passed    1.85 sec
[build]       Start  2: t_webget
[build]  2/10 Test  #2: t_webget .........................   Passed    1.08 sec
[build]       Start  3: byte_stream_basics
[build]  3/10 Test  #3: byte_stream_basics ...............   Passed    0.01 sec
[build]       Start  4: byte_stream_capacity
[build]  4/10 Test  #4: byte_stream_capacity .............   Passed    0.01 sec
[build]       Start  5: byte_stream_one_write
[build]  5/10 Test  #5: byte_stream_one_write ............   Passed    0.01 sec
[build]       Start  6: byte_stream_two_writes
[build]  6/10 Test  #6: byte_stream_two_writes ...........   Passed    0.01 sec
[build]       Start  7: byte_stream_many_writes
[build]  7/10 Test  #7: byte_stream_many_writes ..........   Passed    0.05 sec
[build]       Start  8: byte_stream_stress_test
[build]  8/10 Test  #8: byte_stream_stress_test ..........   Passed    0.05 sec
[build]       Start 37: compile with optimization
[build]  9/10 Test #37: compile with optimization ........   Passed    0.48 sec
[build]       Start 38: byte_stream_speed_test
[build] 10/10 Test #38: byte_stream_speed_test ...........   Passed    0.04 sec
[build]
[build] 100% tests passed, 0 tests failed out of 10
[build]
[build] Total Test time (real) =   3.59 sec
[driver] Build completed: 00:00:03.726
[build] Build finished with exit code 0
```