

**本科实验报告**

|  |  |
| --- | --- |
| 课程名称： | 计算机网络基础 |
| 实验名称： | Lab2 Webget & ByteStream |
| 姓 名： | 祖敬涵 |
| 学 院： | 计算机学院 |
| 系： | 信息安全 |
| 专 业： | 信息安全 |
| 学 号： | 3220102091 |
| 指导教师： |  |

2024年 10月 9日

**浙江大学实验报告**

实验名称： Lab2 Webget 实验类型： 设计实验

# 实验目的：

* 学习掌握Linux虚拟机的用法
* 学习掌握网页的抓取方法
* 学习掌握ByteStream的相关知识
* 学习掌握C++的新特性

# 实验内容

* 安装配置Linux虚拟机并在其上完成本次实验。
* 编写小程序webget，通过网络获取web页面，类似于wget。
* 实现字节流ByteStream：

- 字节流可以从写入端写入，并以相同的顺序，从读取端读取；

- 字节流是有限的，写者可以终止写入。而读者可以在读取到字节流末尾时，产生EOF标志，不再读取；

- 支持流量控制，以控制内存的使用；

- 写入的字节流可能会很长，必须考虑到字节流大于缓冲区大小的情况。

# 主要仪器设备

* 联网的PC机
* Linux虚拟机

# 操作方法与实验步骤

* 在你的电脑上安装配置GNU/Linux：建议安装VirtualBox。安装CS144 VirtualBox虚拟机映像(<https://stanford.edu/class/cs144/vm_howto/vm-howto-image.html>)，也可从学校内网下载 (<http://10.214.131.122:8080/cs_network/cs144_vm.ova> )。如果电脑为2020-21版本的MacBook（带有ARM64M1芯片），请安装UTM虚拟机和ARM64虚拟机映像（[https://stanford.edu/class/cs144/vm\_ howto/](https://stanford.edu/class/cs144/vm_howto/)）。

- 虚拟机映像中用户名为cs144，初始密码为cs144

* 抓取网页: 在正式进行编码工作之前，你需要对本实验第一个任务，即抓取一个网页，有更深刻的理解。

- 在浏览器中，访问<http://cs144.keithw.org/hello>并观察结果。

- 在你的虚拟中运行**telnet cs144.keithw.org http**命令，它告诉telnet程序在你的计算机与另一台计算机（名为cs144.keithw.org）之间打开一个可靠的字节流，并在这台计算机运行一个特定的服务：“http”服务。

- 输入**GET /hello HTTP/1.1**以及回车键，这告诉服务器URL的路径部分。

- 输入**Host: cs144.keithw.org**以及回车键，这告诉服务器URL的主机部分。

- 输入**Connection: close**以及回车键，这告诉服务器你已经完成了HTTP请求。

- 需再次敲击回车键，将返回结果与浏览器的返回结果进行比较。

* 准备工作: 从github上抓取初始代码文件并完成环境搭建。

- 在虚拟机上，获取初始代码文件。运行

**git clone -b lab7-startercode https://github.com/zhanghl-learner/sponge**

- 运行**cd sponge**命令进入sponge目录。

- 运行**mkdir build**命令构建build目录来编译实验代码。

- 运行**cd build**命令进入build目录

- 运行**cmake ..**命令建立搭建系统

- 运行**make**命令编译源代码，需注意每次你对项目进行修改都需运行**make**命令。

- 实验代码的编写以现代C++风格完成，使用最新的2011特性尽可能安全地编程。（<http://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines>）。

* 阅读sponge文档: sponge封装了操作系统函数，请务必在编写实验代码前仔细阅读相关的基础代码文件。

- 阅读入门代码文档（<https://cs144.github.io/doc/lab0>）。

- 阅读 [FileDescriptor](https://cs144.github.io/doc/lab0/class_file_descriptor.html), [Socket](https://cs144.github.io/doc/lab0/class_socket.html), [TCPSocket](https://cs144.github.io/doc/lab0/class_t_c_p_socket.html)以及[Address](https://cs144.github.io/doc/lab0/class_address.html)这些类的文档。

- 阅读描述了这些类的头文件，请查阅**libsponge/util**目录：**file\_descriptor.hh**, **socket.hh**以及**address.hh**。

* 实现webget: 完成webget.cc程序代码的编写以实现抓取网页的功能。要求实现get\_URL函数，功能为向指定IP地址发送HTTP GET请求，然后输出所有响应。可参考配套Doc中[TCPSocket的示例代码](https://cs144.github.io/doc/lab0/class_t_c_p_socket.html)。

- 在build目录下，文本编辑器或IDE下打开**../apps/webget.cc**。

- 在**get\_URL**函数中完成实现，实现代码使用**HTTP**（Web）请求的格式。使用**TCPSocket**类以及**Address**类。

- 运行**make**命令编译程序。

- 运行**./apps/webget cs144.keithw.org /hello**命令进行程序的测试

- 通过上面的测试后运行make check\_webget命令进行自动测试。

注意点：

- 在HTTP中，每行必须以“\r\n”结尾。

- Connection:close这句代码必须包含在客户端的请求中。

- 确保从服务器读取和打印所有的输出，即直到套接字到达“EOF”（文件的末尾），才停止打印输出。

* 可靠字节流：完成ByteStream的代码编写工作。

- 打开**libsponge/byte\_stream.hh**以及**libsponge/byte\_stream.cc**文件，并完成接口内的方法的实现，并按自己的需求增加ByteStream类中的私有成员变量。各个接口的描述可查看注释得到。

- 完成代码的编写后，运行make编译程序。

- 运行**make check\_lab0**命令进行自动测试。

注意点：

- 若看完注释后仍然对接口的实现逻辑不清楚的，可以查看测试样例从而加深理解。路径：sponge/test/，其中byte\_stream\_test\_harness.hh和byte\_stream\_test\_harness.cc为测试所需方法的实现，其余byte\_stream\_xxx.cc为测试样例。

* 温馨提示: 当你在开发代码的时候，可能会遇到无法解决的问题，下面给出解决的办法。

- 运行**cmake .. -DCMAKE\_BUILD\_TYPE=RelASan**命令配置build目录，使编译器能够检测内存错误和未定义的行为并给你很好的诊断。

- 你还可以使用valgrind工具。

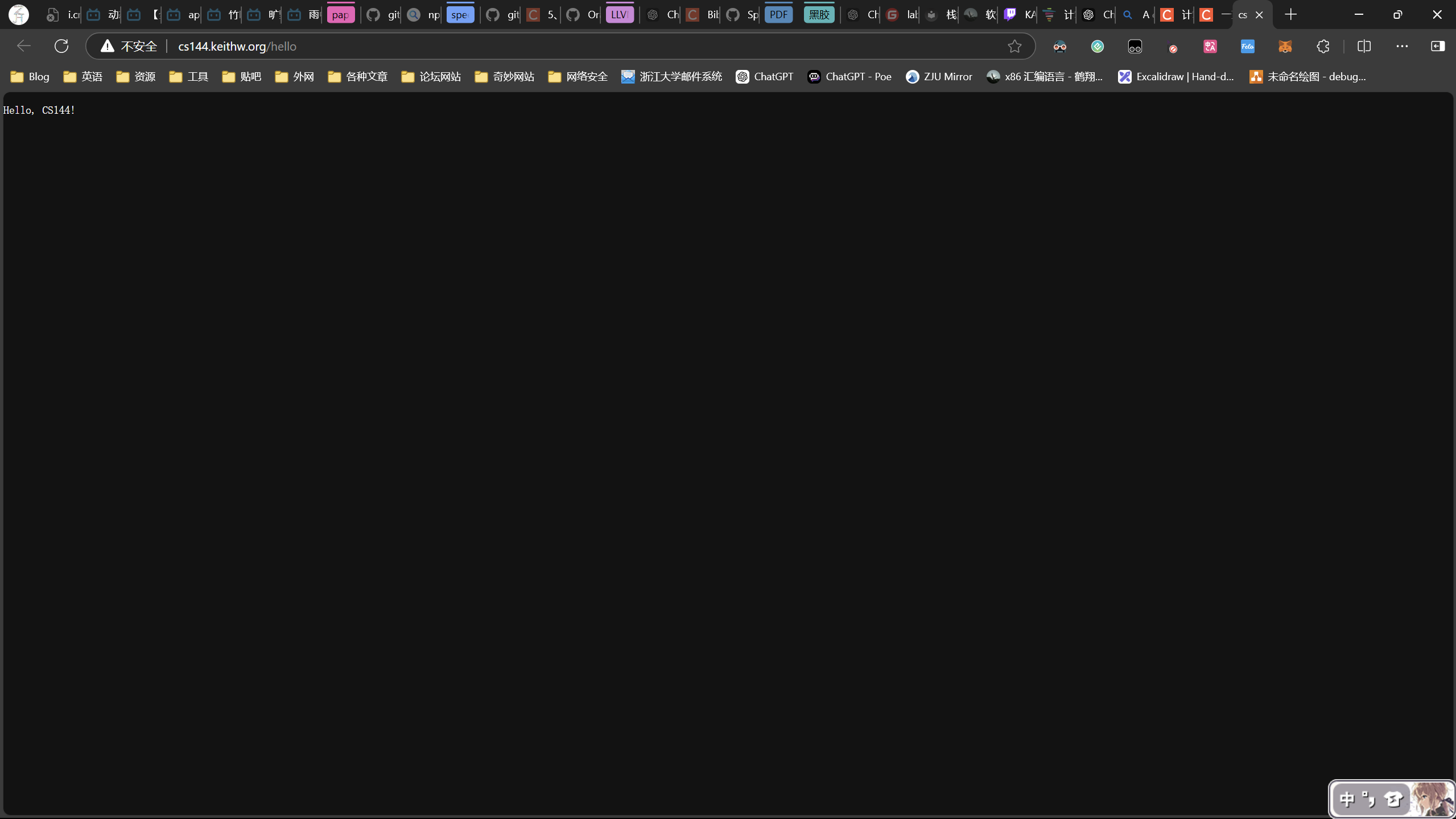
- 你也可以运行**cmake .. -DCMAKE\_BUILD\_TYPE=Debug**命令配置并使用GNU调试器（**gdb**）。

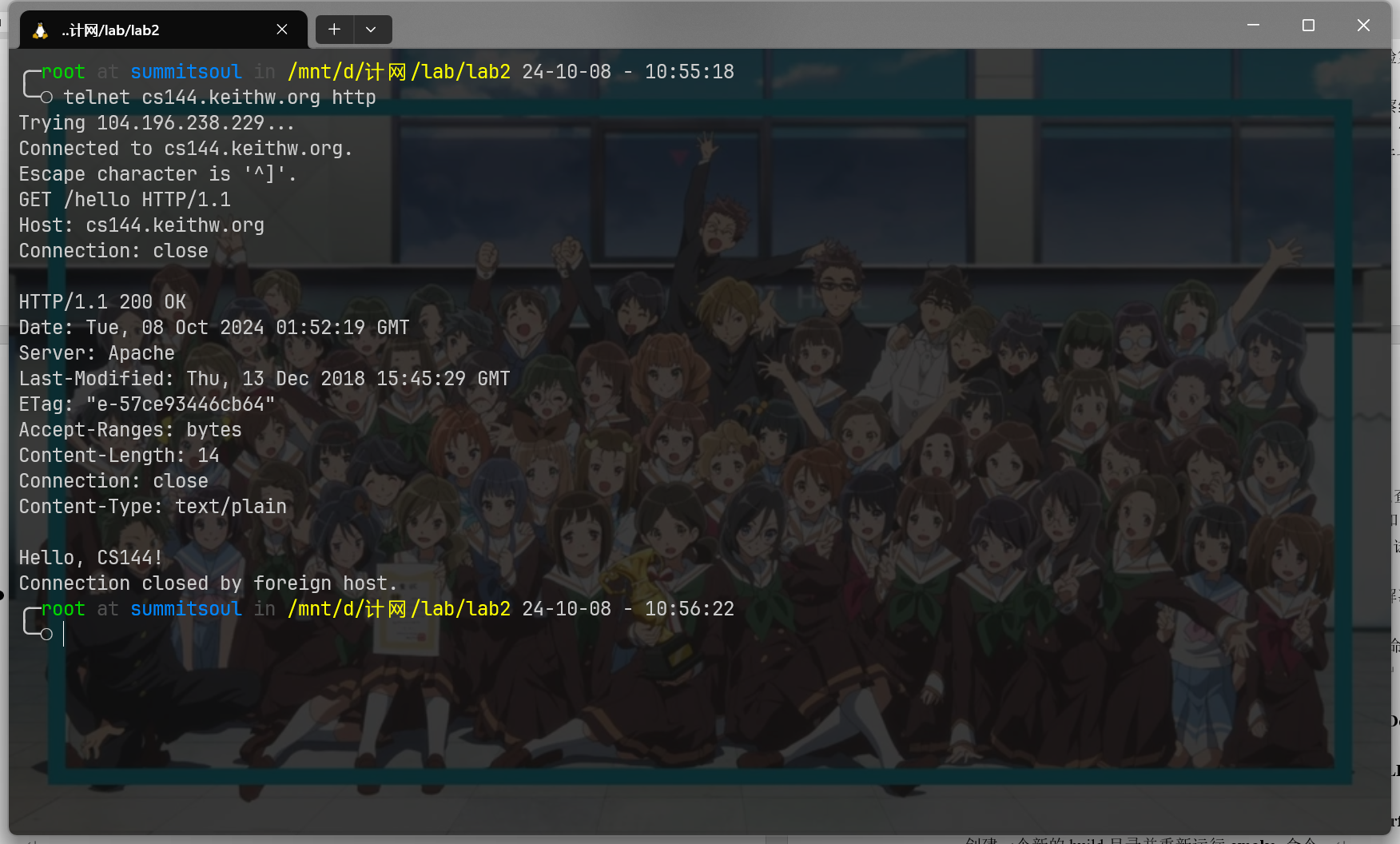
- 你可以运行**make clean**和**cmake .. -DCMAKE\_BUILD\_TYPE=Release**命令重置构建系统。

- 如果你不知道如何修复遇到的问题，你可以运行**rm -rf build**命令删除build目录，创建一个新的build目录并重新运行**cmake..**命令。

# 实验数据记录和处理

* 第二步中抓取网页（通过浏览器和telnet）的运行结果





* 编写的webget关键代码，即get\_URL中的代码

void get\_URL(const string &host, const string &path) {

Address server\_address(host, "http");

    TCPSocket socket;

    socket.connect(server\_address);

    string request = "GET " + path + " HTTP/1.1\r\n";

    request += "Host: " + host + "\r\n";

    request += "Connection: close\r\n\r\n";

    socket.write(request);

    string responce;

*while*(!socket.eof()){

        responce = socket.read();

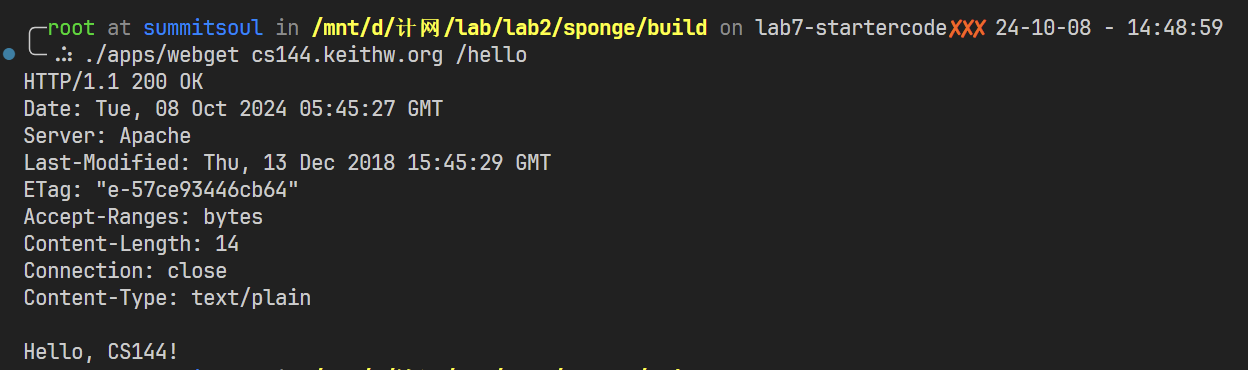
        cout << responce;

    }

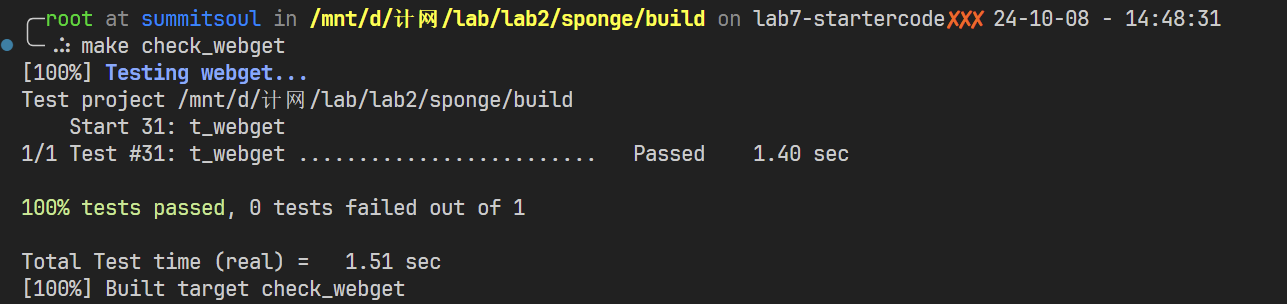
socket.close();

}

* 使用webget抓取网页运行结果



* 运行make check\_webget的测试结果展示

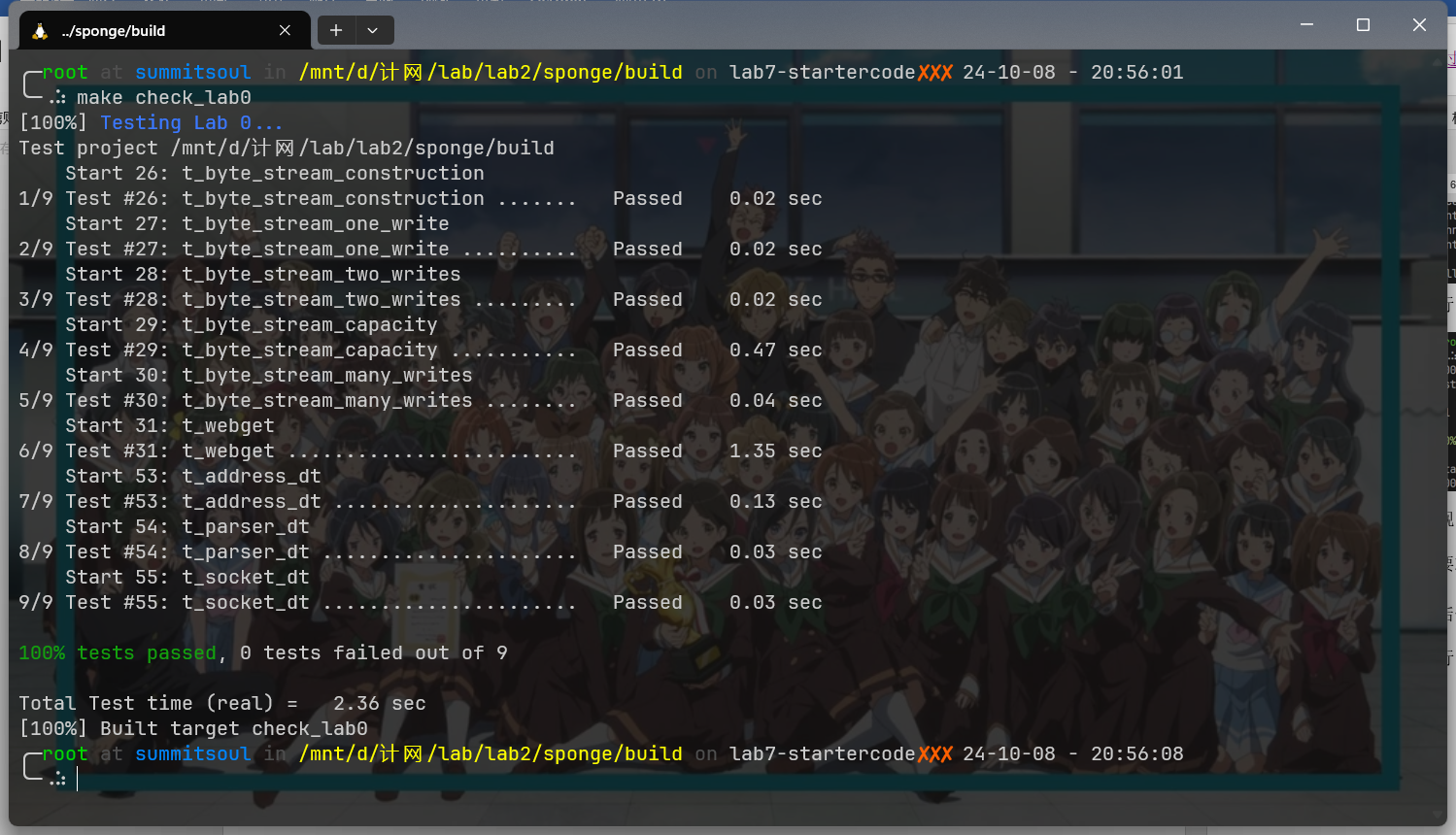


* 实现ByteStream关键代码截图（描述总体，省略细节部分）

主要就是要注意不要越界的问题，其他的到都还好，设置一个queue来当buffer，然后一个变量记录write一个变量记录read，一个记录大小，一个记录是否end。

测试结果如下。

* 运行make check\_lab0测试结果。



# 实验数据记录和处理

根据你编写的程序运行效果，分别解答以下问题（看完请删除本句）：

* 完成webget程序编写后的测试结果和Fetch a Web page步骤的运行结果一致吗？如果不一致的话你认为问题出在哪里？请描述一下所写的webget程序抓取网页的流程

我觉得一致吧，都是得到了那个Hello,CS144!的哦输出。

抓取的过程就像注释写的，显示创建连接，然后发送请求，最后接收到返回的内容并且将内容输出

* 请描述ByteStream是如何实现流控制的？

通过buffer来存储内容，通过bytes\_written来管理写入的内容，不要越界，通过bytes\_read来管理读取的内容，也同样不要越界，同时，要注意异常以及结束的处理。

* 当遇到超出capacity范围的数据流的时候，该如何进行处理？如果不限制流的长度的时候该如何处理？

我这里是只输入在范围内的数据流，后面的直接扔了。

不限制的话，动态分配吧可以，或者设置优先级依次处理这种。

* 你觉得这两种传参方式void Writer::push( string data )和void Writer::push( const string& data )有什么区别（注意：仅针对这一个函数而言）？尝试从性能和语法两方面说明。

Hint: Effective Modern C++ Item 41

根据《Effective Modern C++》中的第41条，讨论这两种传参方式（`void Writer::push(string data)` 和 `void Writer::push(const string& data)`）的区别可以从性能和语法两个方面分析：

性能方面

第一个使用了值传递，这意味着传入的字符串会被复制。在函数调用时，会创建一个新的字符串对象，这可能涉及动态内存分配和数据拷贝，尤其是当字符串很大时，性能开销会显著。如果函数内需要对`data`进行修改（例如追加字符），直接使用这种方式是合适的，因为它不会影响原始对象。

第二个使用了常量引用，传入的字符串不会被复制，而是传递了一个对原始对象的引用。这样做可以避免不必要的复制，提高性能，尤其在处理大对象时。这种方式更适合只读访问，减少了内存使用和拷贝成本。

语法方面

其实都还好吧，感觉都还挺容易读懂的。第二个提高了函数的可读性，减少了不必要的复制。

# 实验数据记录和处理

其实困难是在最开始cmake..跑不起来，会有报错，结果zjgg说要下载最新的或者下新的虚拟机，下了一会虚拟机实在时间有点长，结果下了个包居然好使了。