Policies and Guidelines

- ***Assignments' deadlines come with a 72 hours grace period. Submitting the assignment during the grace period will be accepted with a 20% penalty. Submissions after the grace period will be accepted with a 50% penalty. No submission will be accepted after the last day of classes on Wednesday, May 7, 2025, at 11:59 pm.***
- Every project has required submission guidelines. Please read the submission requirements carefully. Any deviations from specifications on projects will result in point deductions or incomplete grades.
- ***Instructors will not 'fix' your code to see what works***. If your code does not run due to any type of errors, it is considered non-functioning.
- ***You cannot resubmit 'fixed' code after the deadline***, regardless of how small the error is.
- If you use the University resources for development and/or testing, then keep in mind that others in the University may need to use the same resources, so please use these resources wisely. You should also follow the rules and restrictions associated with using the University resources.
- ***Using any external/third-party library and/or framework to implement the project requirements is not permitted.***

Academic Honesty Expectations and Violation Penalty

- ***The programming assignments are team assignments; each team cannot have more than two members***. Each team must do the vast majority of the work on its own. It is permissible to consult with other teams to ask general questions, help discover and fix specific bugs, and talk about high-level approaches in general terms. ***Giving or receiving answers or solution details from other teams is not permissible***.
- You may research online for additional resources; however, ***you may not use code explicitly written to solve the problem you have been given***. ***You may not have anyone else help you write the code or solve the problem***. You may use code snippets found online, providing that they are appropriately and clearly cited within your submitted code.
- ***If you use a distributed version control service such as GitHub to maintain your project, you should always set your project repository to private***. If you make your project repository public during the semester or even after the semester, it may cause a violation of the academic honesty policy if other students find and use your solution.

- If you are involved in plagiarism or cheating, you will receive a score of "0" for the involved project for the first offense. You will receive an "F" grade for the course and be reported to the university for any additional offense.
- Cheating and copying will **NOT** be tolerated. Anything submitted as a programming assignment must be the student's original work.
- ***The use of generative AI tools or apps for assignments, quizzes, and exams in this course, including tools like ChatGPT and other AI writing or coding assistants, is prohibited.*** The use of generative AI to complete any coursework may be considered use of an unauthorized aid, which is a form of cheating. This course policy is designed to promote your learning and intellectual development and to help you reach course learning outcomes.
- ***We reserve the right to use plagiarism detection tools to detect plagiarism in the programming assignments.***

Implementation

In this project, you will learn socket programming for TCP connections in C++: how to create a socket, bind it to a specific address and port, as well as send and receive an HTTP packet. You will also learn working with different HTTP header formats.

Part-1: Simple Web Server

In this part, you will develop a web server that handles HTTP requests sequentially, one at a time. For every HTTP request from the client, your web server should:

- Set up a TCP connection.
- Receive and parse the HTTP request.
- Get the requested file from the web server's file system.
- Create an HTTP response message containing the requested file preceded by header lines. If the requested file is not in the web server, the server should send an HTTP "404 Not Found" message back to the client.
- Send the response directly to the client.
- Close the TCP connection and wait for the subsequent request.

Since most browsers work with HTTP/1.1, the web server response should also use HTTP/1.1. Even though HTTP/1.1 is known as Persistent-HTTP, each TCP connection should handle only a single HTTP request and response pair for this project.

The web browser on the client machine should successfully render the HTTP response message received from the web server. Your web server should support handling HTTP

requests for HTML, JPEG, and PDF files. It should also receive the web server port number as a command-line argument.

Testing the Web Server Locally

Put the provided HTML, JPEG, and PDF files in the same directory that the web server source file is in. Run the web server program from a terminal. The IP address of the localhost is “127.0.0.1”, and the port number is what you passed as a command line argument to the web server (e.g., 28000). Now open a web browser and provide the corresponding URL. For example:

http://127.0.0.1:28000/home.html

“home.html” is the file you placed in the web server directory. Note also the use of the port number after the colon. If you omit ":28000", the web browser will assume port 80, and you will get the web page from the web server only if your web server is listening at port 80. The web browser should then display the contents of “home.html”. If you click on the PDF file hyperlink, it should be displayed in the browser.

Try also to get a file that is not present on the web server. You should get a “404 Not Found” message.

Part-2: Multi-threaded Web Server

In Part-1, the web server handles only one HTTP request at a time. Extend your Part-1 implementation to support a multi-threaded web server capable of simultaneously serving multiple requests. Using threading, first, create a main thread in which your modified web server listens for clients at a fixed socket (welcome socket). When it receives a TCP connection request from a client, it will set up the TCP connection through another socket (data socket) and service the client request in a separate thread. Each HTTP request and response pair should be serviced using a separate TCP connection in a separate thread.

Further, you should implement a safe termination feature for the web servers' main thread, such that when the user presses “CTRL+c” the web server safely terminates. To achieve a safe termination, ensure all TCP connections are closed, and all threads are completed before terminating the main thread.

Part-3: Network Demonstration

In this part, you should demonstrate that your implementation of Part-2 works across two machines connected to the campus network. Machine one is expected to run the web server code, which waits for requests. Machine two (the client) is expected to use a web browser to

request the “home.html” file. All requested files should be fully transmitted to the client and correctly rendered in the client's web browser without any issues.

Submission

- Complete the "README.md" file. You may write “n/a” where applicable (bugs, references, etc.).
- Please submit the following files for Project-2 on Brightspace using the same file name and extension as here:
    - webserver1.cc (or .c/.cpp/.h/.hpp) for Part-1.
    - webserver2.cc (or .c/.cpp/.h/.hpp) for Part-2.
    - README.md
    - [Optional] To submit a makefile for your project, append a ".txt" extension to the file and then submit.
- You must submit your project before the deadline to be considered on time. Otherwise, it will be considered late even if your project was completely working before the deadline. ***Please note that on Brightspace, we maintain all your submissions but only grade your most recent submission.***

Grading Environment

All submissions will be tested and graded under the Ubuntu 24.04 LTS operating system. To run a submission, we will follow the instructions provided by the group in the README.md file. We will test Part-2 of each submission thoroughly to:

- Check the correctness of the implementation.
- Check if the implementation supports HTML, JPEG, and PDF files.
- Check the multi-threading is started correctly, works as expected, and terminates safely. We may use tools other than the web browser for testing, such as shell scripts.
- Check there is no compile time, run time, or memory leak errors using Valgrind.

Grading Rubric

Total: 100 points

- Part-1: 35 points.
- Part-2: 40 points.
- Part-3: 10 points.
- Submission: 15 points

    - README.md: 10 points.
    - Naming Requirements: 5 points.
- ***If submission didn't pass the plagiarism test(s):*** -100 points.
- ***Groups with two members:*** 3 additional points will be deducted ***"per"*** detected problem.