CIT 595 Spring 2017
Project #1

In this project, you will build a web app in C that allows a user to access a data file via a web browser from across the Internet, and then perform basic operations on the data, such as searching and filtering.

In completing this assignment, you will learn how to:

- Develop an application that communicates using HTTP
- Use socket programming in C
- Design and implement a large C program with another student
- Use threads to perform numerous operations simultaneously
- Apply algorithms such as sorting to C data structures

You may work with **one** other student on this project; groups will be formed during the recitation on Tuesday, February 7. Once a group is formed, you may not change it without the instructor's permission.

You may of course collaborate with the student with whom you are working, but otherwise you should not be working with any other students or resources not approved by the instructor. Although you are free to discuss the specification with other students, and to help people with general concepts, you should not be sharing or discussing code with anyone else. Be sure to see the section on academic honesty below.

This project is due on **Tuesday, February 28**. You will do a demo to a member of the instruction staff that day and then will have an opportunity to make final changes before submitting the code for grading.

Logistics

You can download the starter code for this assignment in Canvas. Go to "Files", then "Homework Files", then "Project #1."

For this assignment, you also need to write and turn in a Makefile that compiles your code. Your code will be graded on either a Linux machine (such as eniac) or Mac; if you plan on using Windows for this project, please notify the instructor immediately.

The TAs will also use Valgrind to check your code for memory leaks. Be sure to free/deallocate any heap memory your program is using before it terminates. The TAs will also use Helgrind to look for any race conditions.

Before you begin: Web server basics

As discussed in class, a web server works like this:

1. It reads the incoming HTTP request and figures out what's being requested
2. If what is being requested is a file, it figures out the location of that file on the local system, then reads the file byte-by-byte and sends the bytes back to the client, preceded by the appropriate HTTP header
3. If what is being requested is dynamic content, it gets any necessary arguments from the HTTP request, generates the content, and sends the bytes back to the client, preceded by the appropriate HTTP header

The content sent back to the client is often in the HTML format. As you may already know, HTML is a simple plain-text format that uses tags to tell the browser how to display information. Here is a very simple HTML file:

```
<html>
Hello world!
<p>
This text is <b>bold</b>.
</html>
```

Before the server sends the contents of the HTML file, it needs to add the appropriate header to the response, so that the client knows how to treat the content. In particular, all responses that are HTML content should start like this:

```
HTTP/1.1 200 OK
Content-Type: text/html
```

with an empty space between the last line of the header and the first line of the HTML.

We have provided links for specific things you need for this assignment, but you may want to take a look at https://www.tutorialspoint.com/html/html_overview.htm for a good overview of other HTML features.

Getting Started

The *httpserver.c* file that we distributed is a very simple implementation of a web server that accepts an HTTP request from a web browser, prints the entire request to standard out locally, and then sends an HTTP response back to the browser consisting of some basic HTML. Make sure you are able to understand and compile this code before you proceed.

When you run the program, you need to specify a port number on which the server should listen. This should be given as a command-line argument. Note, however, that you cannot use port numbers below 1024 on most systems.

Check that you are able to test the server by connecting to the host:port using a web

browser. For instance, if you start the server on port 3001, you could open a browser on the same computer, request **http://localhost:3001/test.html**, and see the HTML request echoed back as a response.

Note that in the code we provided, the response is always the same regardless of what is actually being requested, but you should see
`GET /test.html HTTP/1.1`
at the top of the HTTP request that is printed to standard out, indicating that this was a "GET" request for "/test.html" using the HTTP/1.1 protocol.

After sending back the HTML response, the server then closes the connection and shuts down. You will need to modify this behavior later so that it can serve requests continuously.

Note! After your program stops, you may see this if you try to restart it right away:
`Unable to bind: Address already in use`
This just means that the operating system hasn't yet realized that your program is done and that it's okay for another process to use that port number. You can choose a different port number, or just wait a few seconds and the port number should become available again.

Please be sure that you are able to get this code running and see it working before proceeding!

Part 1: Handling multiple requests

The starter code we gave you only handles one request before shutting down the server. Modify the code so that it continues waiting for and handling new requests in a loop. You may need to experiment a bit to figure out what goes into the loop and what should only be done once.

For this part, it is okay to have an infinite loop and to use Ctrl-C to terminate the program. You will add a feature to gracefully stop the program in another part of the assignment.

Note! If you use Firefox or Chrome to test your web server, you may see requests for a file called "favicon.ico". This is automatically requested by the browser so that it can put a little icon in the address bar. If this is causing you problems, it's okay to send nothing back to the browser when that file is requested, e.g. ignore the request, send back an empty response, etc.

Part 2: Display content from a text file

Now it's time to add some content!

Modify your program so that it reads input from a text file and then, on each HTTP request, sends back the content of the file as HTML so that it appears in the web browser.

We have a provided a file called *course_evals.txt* that you can use, or you can use another input file of your choosing, as long as:

- The file contains data in tabular form, e.g. comma-separated, tab-separated, JSON, etc.
- There are at least 100 records (rows) in the data
- Each record (row) contains at least five fields

You can display the data in the web browser in any manner you choose, even unmodified, as long as there is clear separation between each record in the data and each field in the records. Note that in HTML, newline characters are simply treated as whitespace and you need to use "
" to start a new line and "<p>" to have a paragraph break.

You may want to use HTML lists (https://www.tutorialspoint.com/html/html_lists.htm) or tables (https://www.tutorialspoint.com/html/html_tables.htm) but you do not need to make it more advanced than that.

It is okay to assume that the data file will not change once the program is started, meaning that you don't have to re-read it on every HTTP request: it may make more sense to read the file once, create the HTML as a string, and then send back that string as part of each HTTP response.

For this part of the assignment, you may add other functions and other source/header files as necessary. Be sure to modify your Makefile to include all source files.

Last, make sure that you modify the *main* function so that it takes the name of the data file as an argument; do not hardcode the name of this file.

Part 3: Displaying processed data

Modify the program so that it is able to do each of the following, based on the request from the user via the web browser:

- **Sort** the data according to one of the fields and display the sorted results
- **Filter** the data so that some records are omitted and display the results that are included
- Perform a **calculation** on the data and display the results

For sorting:

- There must be at least two sorting options. For instance, you can sort based on a single field either ascending or descending; or you can sort based on one field or on another.
- You do not have to worry about "secondary" sorts, e.g. if you sort on field #2, you don't have to worry about how to sort the records that have the same value for that field.

For filtering:

- The user must be able to specify some value to be used in the filtering. This can be a threshold ("show all records where some field is above some specified value") or an exact match ("show all records where some field equals some specified value").

For performing the calculation:

- There must be at least two calculations performed. For instance, you can calculate the average, min/max, most frequent value, etc.
- All records must be included in performing each calculation.

Aside from actually getting your program to do these things, the big challenge here is figuring out how to let the user specify the different actions and arguments via the web browser.

A simple way is to just use links (https://www.tutorialspoint.com/html/html_text_links.htm) in the HTML, with each link including something in the URL that indicates what functionality. The server can then figure out what is being requested by inspecting the HTTP response; this is known as an HTTP "get" and there is more information about this at http://www.w3schools.com/tags/ref_httpmethods.asp

However, since the user will need to be able to specify a value for the filtering option, and should not have to do so by modifying the URL, you will need to use HTML forms: https://www.tutorialspoint.com/html/html_forms.htm

A simple text input field and button should suffice, but you can use other form widgets if you'd like. For this part, you may want to use an HTTP "post" as described at http://www.w3schools.com/tags/ref_httpmethods.asp

As in Part 2, it is okay to assume that the data file will not change once the program is started, meaning that you don't have to do all the processing for each HTTP request: it may make more sense to read the file, do all the processing, generate the HTML as a string, store it in some sort of data structure, and then send back the appropriate string as part of the HTTP response.

Part 4: Stopping the server

Modify your code so that it is possible to gracefully stop the server from the command line where you started it.

In particular, add a new thread to the program that prompts the user to enter some input and waits/blocks until the user enters the letter 'q'. Once it sees that, the program should then wait for the server to handle its final page request, close the socket, and then terminate.

Note that you may find that your server thread is blocking on "accept" when you indicate that you want to stop the program. It is okay if you require that one more HTTP request come in before ending the program. There are other ways around this (including the pthread_cancel and pthread_kill commands), but this is perhaps the most straightforward.

Part 5: Handling multiple concurrent requests

One limitation of the program you have written so far is that if it is handling an HTTP request and another request comes along, the second request needs to wait until the first one is complete before it will be handled.

Modify your program so that, after accepting a new incoming request, the program starts a new thread to handle that request and send back the response. You will need to consider how to pass the necessary information to the thread and how to make sure your code is threadsafe.

It can be tricky to test this part of the code if requests are handled so quickly that you never have two being handled at the same time. To test this, add some code for one of the features (e.g., sorting the data) that makes it sleep for 10 seconds or longer. Then use your browser to request that content, open up a new tab, and make a request for different content: you should see the second response right away, and then the response for the first request a few seconds later. If your second request has to wait for the first one to finish, you'll know you haven't done this part correctly!

Using Piazza...

Although you are encouraged to post questions on Piazza if you need help or if you need clarification, please be careful about accidentally revealing solutions.

In particular, please do not post public questions that reveal your solutions to the assignment, e.g. how you used threads, how you determined the feature from the HTTP request, etc.

If you think your question might accidentally reveal too much, please post it as a private question and we will redistribute it if appropriate to do so.

Academic Honesty and Collaboration

You may discuss the assignment specification with students in other groups, as well as your general implementation strategy and observations, but *you absolutely must **not** discuss or share code with students in other groups* under any circumstances.

Please see the course policy on academic honesty (posted in Canvas on the "Syllabus" page) for more information. Suspected violations of the policy will result in a score of zero for the project (which would be really bad) and/or be reported to the Office of

Student Conduct at the instructor's discretion.

Although you can, of course, look online for help with the HTTP standard and C libraries and things like that, the work you submit **must** be your own. Copy/pasting code written by other people, even people not in this class, will be considered plagiarism and will be treated as academic dishonesty, even if you were "just looking at it to see how it's done."

Additionally, you may **not** use third-party libraries (i.e., pieces of code that are not readily available to a standard Linux C compiler) without permission from the instructor.

If you run into problems, please ask a member of the teaching staff for help before trying to find help online!

Grading

This project is worth a total of 100 points.

Part 1 is worth 5 points.

Part 2 is worth 20 points. If you are unable to complete this part of the project, you can receive partial credit if you are able to show that your program can:

- read the data file and display the contents to standard out, but not send them to the browser
- read the data file and send the contents to the browser but without any attempt at formatting them

Part 3 is worth 60 points. Each of the three things you need to do (sorting, filtering, and performing a computation) is worth 20 points. If you are unable to complete Part 2, you can still get full credit for Part 3 if you show that the appropriate content is displayed to standard out upon requests from the web browser.

Part 4 is worth 5 points.

Part 5 is worth 10 points.

The grader may also deduct up to 10 points at her discretion for things like not submitting a Makefile, submitting a Makefile that does not compile the code, egregious violations of C coding conventions, memory leaks, potential race conditions or other problems with threads, etc.

Submission

This assignment is due on **Tuesday, February 28**. During recitation that day, you will do a demo to a member of the instruction staff, who will then assess your implementation and provide feedback. If further changes are needed, you will have until the end of the day to submit your code.

Late submissions will be penalized by 10% if submitted up to 24 hours late, 20% for 24-48 hours late, and so on, up to one week, after which they will no longer be accepted.

To submit your assignment, put all of your code and your data file into a *single* zip file. Include your Makefile, too. Keep in mind that your Makefile must properly compile your code on Linux or Mac.

If you know that your program does not work correctly, e.g. potentially incorrect behavior, things that make the program crash, or other known bugs, please describe these in a plain-text or PDF file and include it in your submission. This will greatly help the TAs grade your assignment and make sure that they give you credit for the things that you *did* get working.

Submit the zip file into the "Project #1" assignment in Canvas. You may submit as many times as you'd like; only the last version will be graded.

Updated: 7 Feb 2017, 2:21pm