

CSE-381: Systems 2

Exercise #8

Max Points: 20

Objective: The objective of this exercise is to:

1. Review issues with passing references to local variables to threads
 - Convert code to a multithreaded program that does not need synchronization
2. Understand the use of detached threads (a.k.a background threads)
 - Understand the advantage of multithreaded web-server
3. Observe and fix race conditions in a multi-threaded program

Submissions:

1. This document saved as a PDF file and named `MUID.pdf` (where `MUID` is your unique ID, example: `raodm.pdf`)
2. The C++ program you modified as part of this exercise.

Fill in answers to all of the questions. For some of the questions you can simply copy-paste appropriate text from the terminal/output window into this document. You may discuss the questions with your instructor.

Name: Noah Dunn



Wait for your instructor to cover the concept of background threads prior to proceeding with this exercise. The coverage will take about 15 minutes.

Part #1: Incorrect to pass local variables to threads

Estimated time: $(4 \times 5 \text{ minutes}) = 20 \text{ minutes}$

Background: In all programming languages, the scope (or lifetime) of local variables is limited – that is, once the scope ends, the local variable(s) are destroyed. Consequently, passing references to local variables to threads that operate beyond the scope of a local variable is incorrect and will cause program to abort with errors during runtime.

Exercise:

1. Study the `spinUp` method in the code fragment below. The `spinUp` method incorrectly passes reference to a local variable to threads:

```
using std::vector<std::thread> ThreadList;

void thrMain(int j, std::string &s) {
    std::this_thread::sleep_for(2s); // Sleep 2 seconds
    s.append("result " + std::to_string(j)); // Use reference
}
```

```
void spinUp() {
    ThreadList thrList;
    for (int i = 0; (i < 10); i++) {
        std::string result = "";
        thrList.push_back(std::thread(thrMain, i, std::ref(result)));
    }
    for (auto &t : thrList) {
        t.join();
    }
}
```

In the space below, briefly describe (in your own words) why the `spinUp` method will not work correctly -- **ensure you highlight the difference between the 2 arguments passed to `thrMain` method:**

Manipulating a value passed by reference in a thread is likely going to create some form of Address/Memory sanitization issue when two different threads try to access the same memory block at the same time.

The first parameter, a local variable, cannot be modified outside the scope of the method, it will be fine. The second value, a reference being manipulated by 10 different threads will break, as it is destroyed every single loop.

2. The source code below illustrates the minimal change (it is 1 line code change) to `spinUp` method that would enable passing a valid reference.

```
void spinUp() {
    ThreadList thrList;
    std::string result = "";
    for (int i = 0; (i < 10); i++) {
        thrList.push_back(std::thread(thrMain, std::ref(result)));
    }
    for (auto &t : thrList) {
        t.join();
    }
}
```

In the space below, briefly describe (in your own words) how the above change to the `spinUp` method resolves the issue with invalid reference:

We are no longer destroying the result instance at the termination of every loop.

3. The revised version of `spinUp` method above suffers from a race condition. Briefly describe the race condition in the space below:

We have 10 threads all attempting to manipulate the same memory location at the same time, if any of them try at the same time, we'll hit a race condition.

4. To fix the source of race condition, revise the `thrMain` method by using a mutex. Show a revised version of the `thrMain` method that addresses the race condition (hint: use a `std::mutex` and a `std::lock_guard`) below:

SKIP THIS QUESTION

```
void thrMain(int j, std::string &s) {  
  
}
```

Part #2: Multithreading a web-server using detached threads

Estimated time: 30 minutes

Background: Web-servers often use multiple threads to process requests from users. Moreover, most web-browsers count on multithreaded servers to often use multiple concurrent connections to obtain data.

In a web-server, typically, no synchronization is required when processing multiple requests. Each request can take a varying amount of time to complete and hence, waiting (via `join`) to requests to complete is not efficient. The combination of the aforementioned two characteristics of this application makes it a good candidate for using detached threads (*a.k.a* background threads).

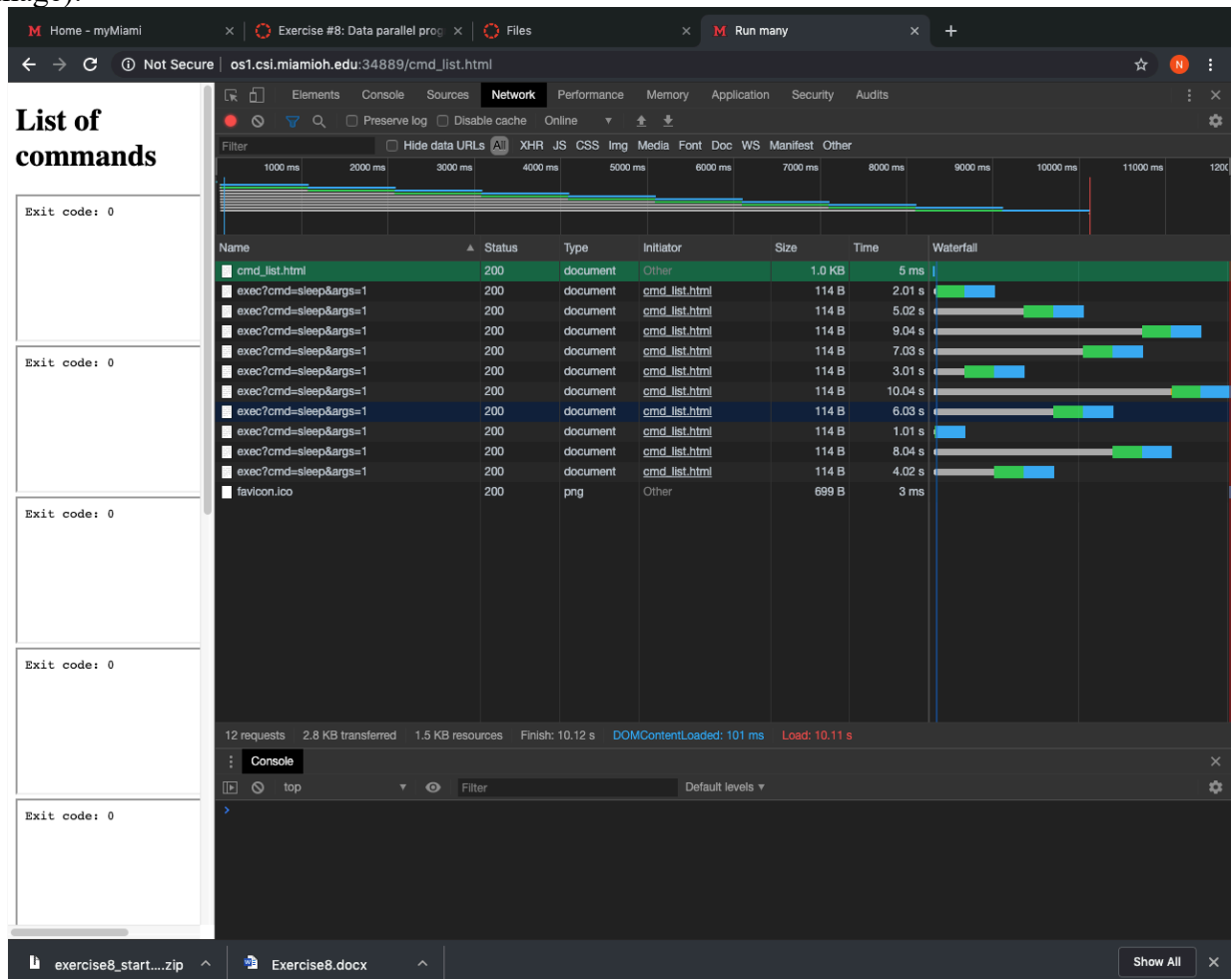
Exercise: Convert the supplied starter code for this exercise to operate using multiple threads via the following procedure:

1. Create a `Miami University C++ Project` in NetBeans called `exercise8` on the Linux server used for this course. Download and `scp` the supplied `exercise8_start_code.zip` file to the NetBeans project directory for your exercise on the Linux server.
2. The supplied zip file contains several files for use by the web-server. From a terminal (doing it in a NetBeans terminal would make your life easier), unzip the files on the Linux server (the Zip file should be in your NetBeans project directory) as shown below:

```
$ cd ~/NetBeansProjects/exercise8  
$ unzip exercise8_starter_code.zip
```

3. Now add the starter code `exercise8.cpp` to your NetBeans project. This is essentially the solution to homework #5 with extension to also stream files, in addition to running programs. You can study this starter code in more detail later on.
4. Compile the starter code in NetBeans. It should compile fine if you have your project setup correctly.

- Run the starter code from NetBeans. It will print the port number where the server is listening for connections.
- In a Chrome web-browser on your local computer, open the Network tab in Developer tools (click on: BrowserMenu→More Tools→Developer Tools→Network). The Network tab enables you to observe interaction between a web-browser and your server.
- In your web-browser access the following URL (you will need to change your port number appropriately) to observe the requests from your browser:
`http://os1.csi.miamioh.edu:4000/cmd_list.html`
- Copy-paste a screenshot of your browser's Network tab below (replacing the sample image):



- Using the information in the Network tab, indicate how much time took to "load" the page:
Time taken to Load the page (at bottom of Network tab): 10.11 s

10. You will note that the web page takes about 10 seconds to load. Why do you think the web-page takes 10 seconds to load all of its contents?

There are 10 calls to sleep for 1s listed in the network tab, taking at minimum 10 seconds to load all the information

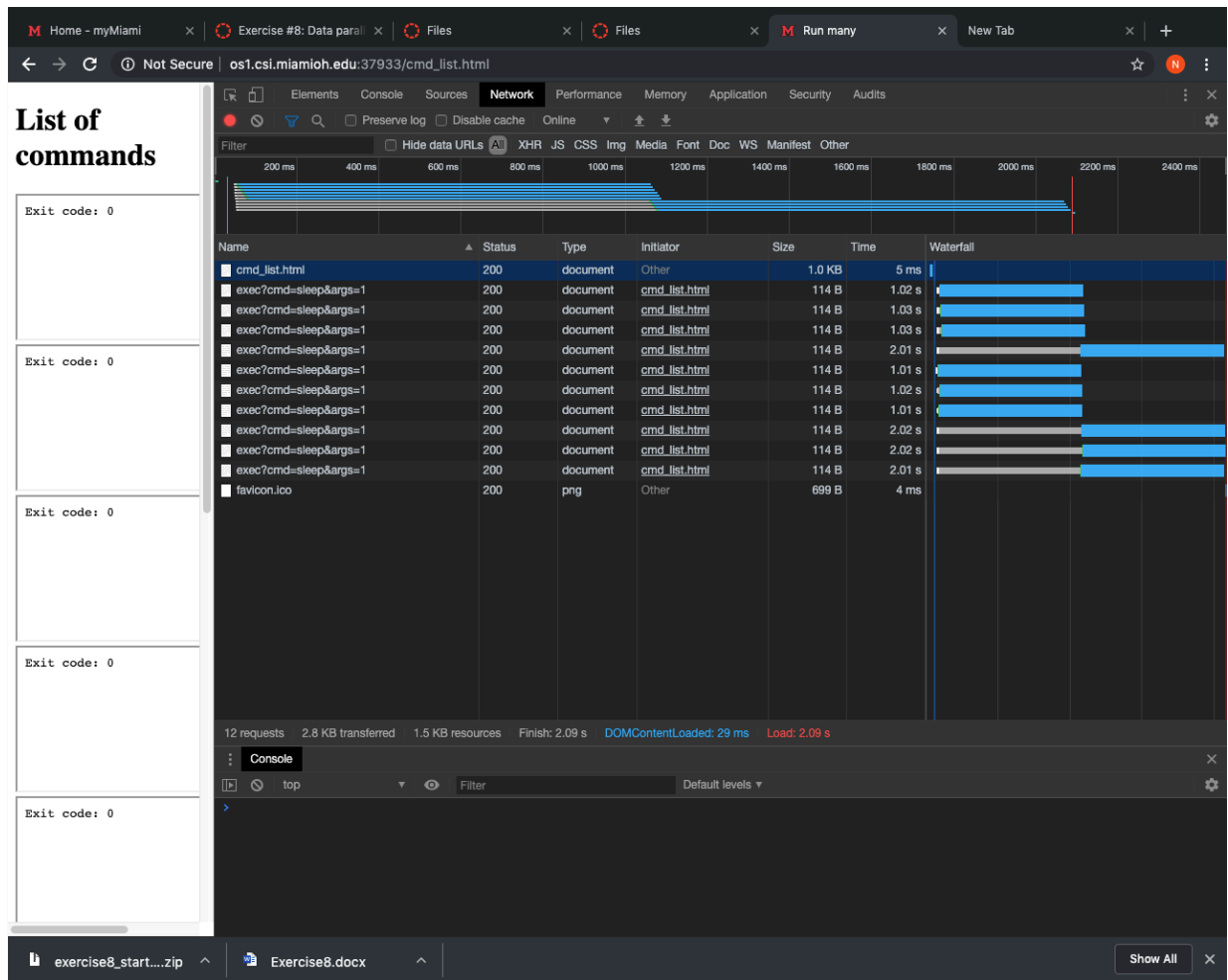
11. A 10 second loading time is an eternity on the Internet these days. So, let us try to see how to use threads to make the site load faster. Using the example of detached threads from lecture slides, modify the runServer method to process each connection using a separate thread.

Note: Ensure you use `std::shared_ptr` and `std::make_shared` as shown in the lecture slides. Importantly, **ensure you understand why the use of local variable in client in the lecture slides does not cause a bad reference or a race condition.**

12. Once you have done some basic tests to ensure your server is operating correctly, revisit the following URL while recording requests in Chrome browser's Network tab (you have to press Shift while reloading the page to correctly reload the page):

`http://osl.csi.miamioh.edu:4000/cmd_list.html`

13. Place a screenshot of the Network traffic in the tab below replacing the screenshot below:



14. Using the information in the Network tab, indicate how much time took to "load" the page:

Time taken to Load the page with multiple threads: **2.09 s**

15. Using the information from step #8, briefly discuss the decrease in load time. Also mention how many concurrent threads do you think the browser used.

Our runtime went from 10 seconds down to 2 seconds, leading to a drastic decrease in load time. I would vouch a guess at 2, since a large number of threads finished + or - 0.02 seconds at the 1 second mark and 2 second mark.

Part #3: Understanding capabilities of the web-server

Estimated time: 15 minutes

Background: The web-server developed in this exercise is versatile and can operate as a general-purpose web-server -- that is, it can be used to run any script (say, PHP, Python) or

programs in other languages (Java) that use standard I/O. In fact, this is how almost all PHP and Python scripts are run by web-servers.



It is important that you understand and appreciate the capabilities and sophistication that is already present in the programs you have created as part of this course.

Exercise: Explore the capabilities of your web-server via the following procedure:

1. The starter code (zip file) already includes some files for testing.
2. Ensure your web-server is running. In your web-browser access the following URL (you will need to change your port number appropriately):

```
http://osl.csi.miamioh.edu:4000/exec.html
```

3. Note how the web-server returned the contents of the file! This is a feature that has been added to the `exercise8.cpp` file supplied to you.
4. In the web-form, use the command `php` and parameter `php_info.php` as shown in the adjacent image to have your web-server to use `php` program to run the `php_info.php` script. Your web-server will run the PHP program (it could run any PHP script for that matter, including `Laravel` applications) and show you the results.

Enter command and command-line arguments:

Command to run:

Command-line arguments:

5. Similarly, using the web-form, run the Java program using the following two steps:
 - a. Compile the Java program using command & argument: `javac java_test.java`
 - b. Run the Java program via command & argument: `java java_test`
6. Similarly, using the web-form, run a Python program using the following two steps:
 - a. Run the Java program via command & argument: `python intro.py`
7. Building on the above experience, briefly (3 sentences) describe how you could use the web-server to serve as a "remote Linux terminal" (similar to a standard Linux terminal where the user can type 1-line command at the `$` prompt)

Much like the `python` and `java` scripts we are capable of running on the server, we can use the existing webpage with the `C++` code we wrote several homeworks ago. We can extract the command we want to run with arguments, make a call to the `C++` terminal we wrote, providing the arguments, and return the output we received and display it on the web page. This is exactly the same.

Part #4: Submit files

Estimated time: < 5 minutes

1. This document saved as a PDF file and named `MUID.pdf` (where `MUID` is your unique ID, example: `raodm.pdf`)
2. The C++ program you modified as part of this exercise.

Upload each file individually to Canvas. Do not upload archive files such as zip/7zip/rar/tar/gz etc. Ensure you click the `Submit` button on Canvas once you have uploaded all the necessary files.

Part #5: Optional Part – Code Review

Estimated time: 10 minutes

Review the file processing features in `exercise8.cpp` while paying attention to the following methods:

1. Review the `getMimeType` method. This method returns a MIME type based on a file's extension. You should be able to explain what MIME type is by now (of course, MIME types were covered in CSE-278 as well).
2. Next review the `sendData` method to see how it reads and streams contents of a given input stream `is`. This method is used to stream data from files and processes.
3. See how `sendData` method is used in the program in the following three cases:

- a. To send output from a process in the `exec` method. Review the `exec` method.

Copy-paste the call to `sendData` method from `exec` method below:

```
sendData("text/plain", "200 OK", pid, is, os);
```

- b. Review the `send404` method that sends an HTTP error response back to the client. Copy-paste the call to `sendData` method from `send404` method below:

```
sendData("text/plain", "404 Not Found", -1, is, os);
```

- c. Next, review the `serveClient` method that process a client request. Copy-paste the call to `sendData` method from `serveClient` method below:

```
sendData(getMimeType(path), "200 OK", -1, dataFile, os);
```