



Prototyping a session-based web server

Prepared By:

Sahil Nepal

Syed Konain Abbas

Prepared To:

Prof. Daqing Hou



Table of Contents

<i>Project Description</i>	3
<i>Conceptualization</i>	3
Session	3
Cookies	3
Implementation	3
<i>Achievements</i>	4
<i>Task distribution and team assessment</i>	4
Session Management (By Sahil Nepal in server.cpp)	4
Cookie Management (By Syed Konain Abbas in browser.cpp)	4
<i>Testing</i>	5
Unit Testing	5
Integration Testing	5
Error Handling	5
Performance Testing.....	5
Test Cases	5
<i>Implementation</i>	10
Task 1: File Operation for Implementi ng Session/Cookie.....	10
Task 2: Multithreading.....	10
Task 3: Feature Improvement	10
Coding Requirement	10
<i>Frequently Asked Questions (FAQ)</i>	11
<i>GitHub URL</i>	12



Project Description

The session-based web server project displays concurrent communication between multiple browser clients and a server. The project consists of two main components: a server program and a browser program. The server is responsible for accepting connections from multiple browser clients and handling their messages concurrently. The browser program allows users to connect to the server, send messages, and display responses received from the server.

Conceptualization

Session

In context of this project, a session is initiated by the server to allow browser to connect and interact with server. Session here refers to the period of interaction between a browser client and the server. Whenever a browser connects to the server, a session is established such that we can track and manage the communication between server and browser. Each session has unique session id that helps server to identify different sessions. Session stores data like user input, interaction, and other relevant details.

Cookies

Cookies in general are pieces of information sent by server and stored on the client's device. In our case, the cookies contain session state and helps in managing sessions. Whenever a browser connects to a server, the server sends cookies that consists of session id. The browser then sends the information from these cookies whenever it connects to the server or performs any interaction. This helps session in identifying the past interactions and details and uses it to provide seamless experience.

Implementation

The basic instance of implementation of cookies and session in our project are highlighted below:

- The `load_cookie()` function loads the session ID from the cookie file (`browser.cookie`) if it exists. If no cookie file exists, it assigns the session ID to be -1.
- The `save_cookie()` function saves the session ID to the cookie file (`browser.cookie`).
- When the browser connects to the server, it sends the session ID stored in the cookie to the server. This allows the server to identify the user's session.
- If the server receives a session ID of -1 (indicating a new session), it assigns a new session ID to the browser and sends it back. The browser then saves this session ID to the cookie file for future use.
- Throughout the interaction between the browser and the server, the session ID is used to maintain the state of the user's session, allowing for continuity and personalized experiences.

Achievements

As a team, we were able to achieve the requirement of this project. We successfully worked on the development of both the server-side and the client-side components. We integrated the server and browser components and ensured that there is a smooth interaction. Currently, the system is capable to handle concurrent sessions from multiple browser clients. We have managed to keep track of the cookie such that the browser sends the detail of it every time it connects to the server. We have also managed to work on error handling and the system is able to throw appropriate errors when the user provides invalid requests. We tested all the provided test-case scenarios and our system managed to successfully execute the needed criteria. Finally, we've pushed the code in GitHub and managed appropriate documentation for it.

Task distribution and team assessment

As a group of two, we distributed the execution in two different sections: session management and cookie handling. I (Sahil Nepal) handled the session management and Syed Konain Abbas worked on the cookie handling aspect. Furthermore, I tested Syed's code changes, and he checked the changes made by me. We also performed testing on the same and different devices to ensure that the test cases were met. The task distribution is thoroughly explained below:

Session Management (By Sahil Nepal in server.cpp)

- Worked on keeping track of multiple browser sessions concurrently, identified by the session ID.
- Accepting connections from multiple browsers and assigning them to appropriate sessions.
- Implementing load_all_sessions() and save_session().
- Implementing lock mechanism.
- Maintaining session state and creating session file.
- Documentation.

Cookie Management (By Syed Konain Abbas in browser.cpp)

- Maintaining the cookie details sent by the server.
- Parsing incoming cookie and sending them back to server in future requests on basis of the session ID.
- Handling appropriate error message details and ensuring seamless interaction with user.
- Creating appropriate browser cookie file.
- Documentation

Testing

We performed both the unit testing as well as integration testing for this project. Both of us took time to review each other's code and tested on individual as well as multiple devices. The further detailing of the testing procedures is listed below.

Unit Testing

We ensured that our server could handle incoming connections, manage connections, and process messages from browsers. We tried making multiple browser connections to the server and verified that each connection is accepted and handled correctly. We also sent different message types to verify that the server handled input correctly.

Integration Testing

We simulated multiple browsers connecting to the server, sending the messages from browsers, and receiving responses from the server. We verified that server could handle concurrent connections from multiple browsers without errors. We tried this across multiple devices too.

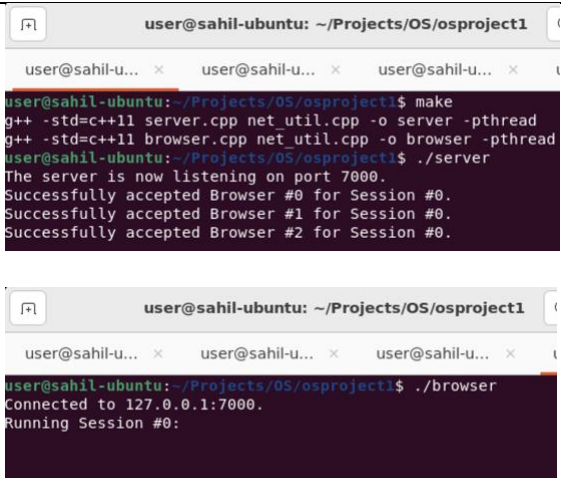
Error Handling

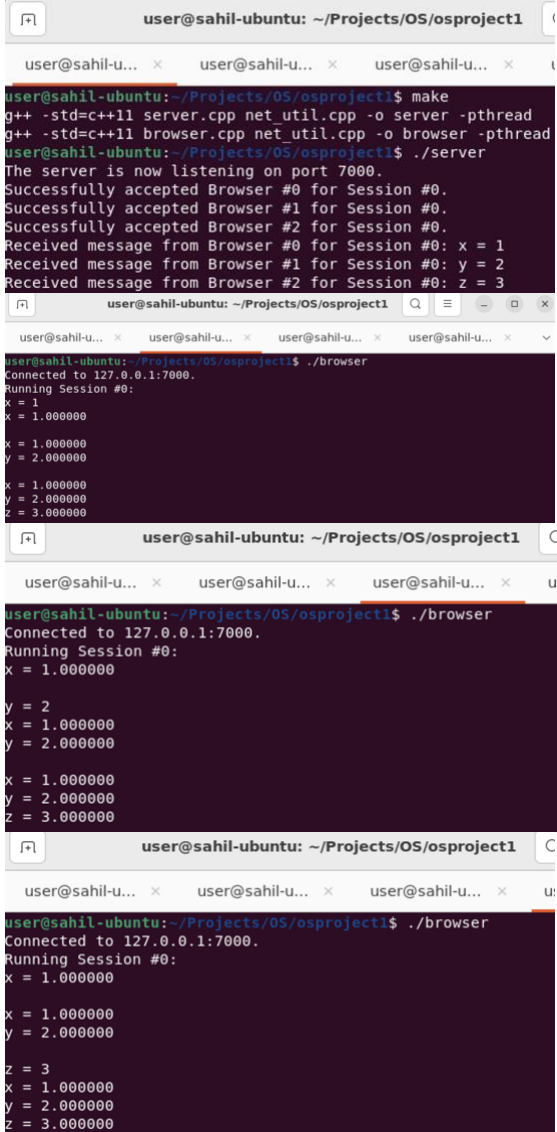
We also provided inaccurate inputs to validate appropriate error messages was being sent and ensured in seamless interaction between user-browser-server.

Performance Testing

We manually tested for memory leak and saw no leak during the entire execution and all the components functions exactly as it was expected.

Test Cases

Test_case_id	Description	Result	Image
T1	Start a server and two or more browsers on the same machine using the default IP and port.	Pass	 <p>The image shows two screenshots of a terminal window and a web browser. The top screenshot shows the terminal output for building and running the server. The bottom screenshot shows the terminal output for running the browser, which connects to the server.</p> <pre>user@sahil-ubuntu: ~/Projects/OS/osproject1 user@sahil-ubuntu: ~/Projects/OS/osproject1\$ make g++ -std=c++11 server.cpp net_util.cpp -o server -pthread g++ -std=c++11 browser.cpp net_util.cpp -o browser -pthread user@sahil-ubuntu: ~/Projects/OS/osproject1\$./server The server is now listening on port 7000. Successfully accepted Browser #0 for Session #0. Successfully accepted Browser #1 for Session #0. Successfully accepted Browser #2 for Session #0. user@sahil-ubuntu: ~/Projects/OS/osproject1 user@sahil-ubuntu: ~/Projects/OS/osproject1\$./browser Connected to 127.0.0.1:7000. Running Session #0:</pre>

T2	Test some commands on each browser and check if everything runs smoothly.	Pass	 <p>The screenshot displays three windows from a user@saahil-ubuntu system. The top window is a terminal showing the compilation and execution of a C++ program. It uses 'make' to compile 'server.cpp' and 'browser.cpp' into 'server' and 'browser' executables. The 'server' is then run, displaying messages for accepted browsers and received data (x=1, y=2, z=3). The middle and bottom windows are web browsers. The middle browser window shows the output of running the 'browser' executable, which connects to 127.0.0.1:7000 and displays the received data. The bottom browser window shows the same output, confirming the successful execution and data transfer between the server and browser.</p>
----	---	------	--



T3	Exit and restart the browsers one by one to check if they can reconnect to the previous session properly and all previously assigned variables in that session are reloaded correctly.	Pass	<pre>user@sahil-ubuntu:~/Projects/OS/osproject1\$ make g++ -std=c++11 server.cpp net_util.cpp -o server -pthread g++ -std=c++11 browser.cpp net_util.cpp -o browser -pthread user@sahil-ubuntu:~/Projects/OS/osproject1\$./server The server is now listening on port 7000. Successfully accepted Browser #0 for Session #0. Successfully accepted Browser #1 for Session #0. Successfully accepted Browser #2 for Session #0. Received message from Browser #0 for Session #0: x = 1 Received message from Browser #1 for Session #0: y = 2 Received message from Browser #2 for Session #0: z = 3 Received message from Browser #2 for Session #0: a=4 Received message from Browser #2 for Session #0: EXIT Browser #2 exited. Successfully accepted Browser #2 for Session #0. Received message from Browser #2 for Session #0: a = 5 Received message from Browser #0 for Session #0: EXIT Browser #0 exited. Successfully accepted Browser #0 for Session #0. Received message from Browser #0 for Session #0: b = 7 Received message from Browser #1 for Session #0: EXIT Browser #1 exited. Successfully accepted Browser #1 for Session #0. Received message from Browser #1 for Session #0: c = 9 EXIT Closed the connection to 127.0.0.1:7000. user@sahil-ubuntu:~/Projects/OS/osproject1\$./browser Connected to 127.0.0.1:7000. Running Session #0: b = 7 a = 5.000000 b = 7.000000 x = 1.000000 y = 2.000000 z = 3.000000 a = 5.000000 b = 7.000000 c = 9.000000 x = 1.000000 y = 2.000000 z = 3.000000 user@sahil-ubuntu:~/Projects/OS/osproject1\$./browser Connected to 127.0.0.1:7000. Running Session #0: a = 5 a = 5.000000 x = 1.000000 y = 2.000000 z = 3.000000 a = 5.000000 b = 7.000000 x = 1.000000 y = 2.000000 z = 3.000000 a = 5.000000 b = 7.000000 c = 9.000000 x = 1.000000 y = 2.000000 z = 3.000000</pre>
----	--	------	--



			<pre>user@sahil-ubuntu:~/Projects/OS/osproject1\$./browser Connected to 127.0.0.1:7000. Running Session #0: c = 9 a = 5.000000 b = 7.000000 c = 9.000000 x = 1.000000 y = 2.000000 z = 3.000000</pre>
T4	Handling invalid input	Pass	<pre>Received message from Browser #0 for Session #0: xyz Received message from Browser #0 for Session #0: x = Received message from Browser #0 for Session #0: 1 + 2 Received message from Browser #0 for Session #0: x = y * user@sahil-ubuntu:~/Projects/OS/osproject1\$./browser Connected to 127.0.0.1:7000. Running Session #0: xyz Invalid input! x = Invalid input! 1 + 2 Invalid input! x = y * Invalid input!</pre>
T5	Start a server and two or more browsers on two different machines.	Pass	<pre>Socket bind failed: Address already in use (base) abbas@CAMP115-SA1:/media/abbas/Expansion/downloads/os/osproject1\$./server --port 7002 The server is now listening on port 7002. Successfully accepted Browser #0 for Session #0. Successfully accepted Browser #1 for Session #0. Successfully accepted Browser #2 for Session #0. (base) abbas@CAMP115-SA1:/media/abbas/Expansion/downloads/os/osproject1\$./browser --port 7002 Connected to 127.0.0.1:7002. Running Session #0: user@sahil-ubuntu:~/Projects/OS/osproject1\$./browser --host 128.153.11.219 --port 7002 Connected to 128.153.11.219:7002. Running Session #0:</pre>
T6	Test some commands on each browser and check if everything runs smoothly.	Pass	<pre>(base) abbas@CAMP115-SA1:/media/abbas/Expansion/downloads/os/osproject1\$./server --port 7000 The server is now listening on port 7000. Successfully accepted Browser #0 for Session #0. Successfully accepted Browser #1 for Session #1. Received message from Browser #0 for Session #0: x = 1 Received message from Browser #1 for Session #1: y = 2 (base) abbas@CAMP115-SA1:/media/abbas/Expansion/downloads/os/osproject1\$./browser Connected to 127.0.0.1:7000. Running Session #0: x = 1 x = 1.000000</pre>

			 <pre> user@sahil-ubuntu: ~/Projects/OS/osproject1 user@sahil-ubuntu:~/Projects/OS/osproject1\$./browser --host 128.153.11.219 --port 7000 Connected to 128.153.11.219:7000. Running Session #1: y = 2 y = 2.000000 </pre>
T7	Exit and restart the browsers one by one to check if they can reconnect to the previous session properly and all previously assigned variables in that session are reloaded correctly.	Pass	 <pre> Received message from Browser #0 for Session #0: EXIT Browser #0 exited. Successfully accepted Browser #0 for Session #0. Received message from Browser #0 for Session #0: a = 10 Received message from Browser #1 for Session #1: EXIT Browser #1 exited. Successfully accepted Browser #1 for Session #1. Received message from Browser #1 for Session #1: b = 20 EXIT Closed the connection to 127.0.0.1:7000. (base) abbas@CAMP115-SA1:/media/abbas/Expansion/downloads/os/osproject1\$ (base) abbas@CAMP115-SA1:/media/abbas/Expansion/downloads/os/osproject1\$./browser Connected to 127.0.0.1:7000. Running Session #0: a = 10 a = 10.000000 x = 1.000000 EXIT Closed the connection to 128.153.11.219:7000. user@sahil-ubuntu:~/Projects/OS/osproject1\$./browser --host 128.153.11.219 --port 7000 Connected to 128.153.11.219:7000. Running Session #1: b = 20 b = 20.000000 y = 2.000000 </pre>
T8	Test some commands on each browser across both devices and check if everything runs smoothly.		 <pre> xyz Invalid input! x = Invalid input! 1 + 2 Invalid input! x = y * Invalid input! xyz Invalid input! x = Invalid input! 1 + 2 Invalid input! x = y * Invalid input! Received message from Browser #0 for Session #0: xyz Received message from Browser #0 for Session #0: x = Received message from Browser #0 for Session #0: 1 + 2 Received message from Browser #0 for Session #0: x = y * Received message from Browser #1 for Session #1: xyz Received message from Browser #1 for Session #1: x = Received message from Browser #1 for Session #1: 1 + 2 Received message from Browser #1 for Session #1: x = y * </pre>

Implementation

Task 1: File Operation for Implementing Session/Cookie

- Successful implementation of load_all_sessions() in server.c.
- Successful implementation of save_session() in server.c.
- Successful implementation of load_cookie() in browser.c
- Successful implementation of save_cookie() in browser.c

Task 2: Multithreading

- Successful implementation of multithreading in server.
- Implementation of locks in critical section of register_browser() in server.c.
- Successful implementation of multithreading in browser.

Task 3: Feature Improvement

- Accurately handling invalid inputs with appropriate error messages.
- Proper implementation of hashmap for session_list in server.c.

Coding Requirement

- No signature of any given functions was changed. #include<assert.h> library was introduced in header of both server and browser.
- Code is running with no issues. All the files are in the provided .cpp format. Additional sessions and browser cookies files are generated as mentioned.
- No memory leak is identified during manual verification and identification for memory leak.
- All coding requirements are met.

Frequently Asked Questions (FAQ)

- Why do we need locks for browser_list and session_list in browser_handler()? What could happen if there is no lock?

Answer: Locks are synchronization mechanism in session-based server, that allows only one thread to access a critical section of code at a time.

The purpose of using locks for browser_list and session_list in browser_handler() is to maintain separate spaces for each active session's browser and session information. By using locks, we prevent concurrent access to these lists, thus only one thread can modify them at a time. This separation is critical to avoid scenarios where browsers could accidentally access and alter data belonging to other browsers within specific sessions.

Without locks, multiple threads could concurrently access and modify the shared lists, leading to race conditions. In such scenarios, browsers might unintentionally overwrite or interfere with each other's data within sessions, resulting in unpredictable behavior and data corruption. Therefore, locks are very important for maintaining data consistency and avoiding race conditions in the browser_handler() function.

- Can we use multiprocessing instead of multithreading here? What are the differences between multiprocessing and multithreading?

Answer: No, we cannot use multiprocessing instead of multithreading in our case as these are two distinctively different concepts with different areas of application.

The differences in multiprocessing and multithreading are:

Multiprocessing	Multithreading
Additional CPUs are used to increase the computing power.	Multiple threads are created of a single process to increase the computing power.
Multiple processes are executed simultaneously.	Many threads of a process are executed simultaneously.
Process creation is comparatively time-consuming.	Creating threads is more economical in terms of overhead.
Each process has its own distinct address space.	Common address space is shared by all threads.

- What are the advantages and the disadvantages of using a hashmap/dictionary over an array or a linked list?

Answer: The advantages of using hashmap/dictionary over an array or a linked list are:

1. Data retrieval in hashmap/dictionary is relatively quicker and can be proven effective specially in large datasets.
2. Hashmap uses key-value approach which can prove effective in large number of algorithms and is often used in cache and database.
3. Hashmap uses dynamic memory allocation technique and thus can prove effective over array which defines fixed size. This structure reduces the wastage of space.

The disadvantages of using hashmap/dictionary over an array or a linked list are:

1. Hashmap needs to store hash table, needs to store hash codes, and maintain internal data structures. This can prove ineffective for datasets with less data.



2. Hashmap encounters hash collisions, where different keys map to same hash code. To resolve this, additional processing is required, thus impacting performance.
 3. Arrays and linked lists are usually ordered and can prove effective for systems that need sorted collection in comparison to hashmap which does not guarantee any specific order of key-value pairs.
- Summarize the changes you made to the original code to make use of hashmap/dictionary in the current code.

Answer: The changes made to the original code to make use of hashmap/dictionary in current code are:

1. Replaced the array 'session_list' with 'session_hash'.
2. Updated 'get_session()' function to retrieve sessions from 'session_hash'.
3. Modified 'process_message()', 'load_all_sessions()', and 'save_session()' to use 'session_hash' instead of 'session_list'.
4. Used mutex locks when accessing hashmaps to prevent race conditions.

GitHub URL

<https://github.com/nepal-s/Prototyping-a-Session-based-Web-Server>