OSN Assignment 5

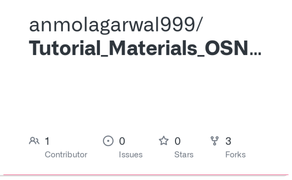
Shreyash Jain

2020101006

Report : Question 3

- I have used the template of server_prog.cpp and client_sim.cpp provided in the tutorial resources. Can be found here :

Tutorial_Materials_OSNW_M-21/netw at master · anmolagarwal999/Tutorial_Materials_OSNW_M-21
Contribute to anmolagarwal999/Tutorial_Materials_OSNW_M-21 development by creating an account on GitHub.
https://github.com/anmolagarwal999/Tutorial_Materials_OSNW_M-21/tree/master/netw


anmolagarwal999/
Tutorial_Materials_OSN...
1 Contributor · 0 Issues · 0 Stars · 3 Forks

- I have made changes to these two files only to satisfy the requirements of the assignment.
- We can divide the assignment into 2 parts, client-side and server-side.

A. Client side

Code in client.cpp

- I take the input, create `num_requests` threads called `users` which are handled by `user_routine` function and have a parameter index passed into it. These threads are the multiple users accessing the server.
- In the user_routine function, I make the respective user thread sleep for the time specified in input, then I send and get a response from the server side, via the functions `send_string_on_socket` and `read_string_from_socket` which I print as the output in the specified format.

B. Server side

Code in server.cpp

- I have created a map `map<int, string> dictionary` which acts like a dictionary storing key, value pair.
- I also have created a queue `queue<int> fd_queue` which stores the file descriptor of the client socket pushed into it. Got this idea from the hint provided in the assignment pdf.
- I make use of mutex locks `dictionary_key_locks[key]` which secure the data stored in the dictionary for the corresponding key. Given : 0≤key≤100
- I create `num_workers` (taken as a command line argument) threads called `workers` which follows the `worker_routine`.
- I have also used a semaphore `queue_sem` with initial value of counter as zero, as soon as a file descriptor is pushed to the `fd_queue`, I execute `sem_post` on this semaphore. And all the worker threads accessing the queue for client socket file descriptor will be waiting for a fd using the `sem_wait` function on this semaphore.
- In worker routine I wait for the queue to be filled with a `fd` by using `sem_wait` and pop it out from the `queue` once I get the `fd` and use the function `handle_connection` to handle the request from client.
- In `handle_connection` , I read the request string from the client side via the function `read_string_from_socket`.
- I tokenize the request string received from client side and handle it with all possible inputs to generate the message to be sent back to client.
- I sleep for 2 seconds, as mentioned in the pdf before sending the message string to the socket.
- Then I send the message string from the server to client side via the socket by using the function `send_string_on_socket`.