OS lab - 3
Readme file

-Harshita Kalani (B20CS019)

File Description:

Unzip **B20CS019_OS_LAB3**.zip file

To execute question 1

a) cd Q1

b) sh producer_consumer.sh - producer_consumer.sh contains the script to run the file, if you want
any modifications you can make it there.

c) producer.cpp is the producer file, consumer.cpp is the consumer file.

To execute Question 2

a) cd Q2

b) This question contains 3 versions

i) Version1

(1) cd Version1

(2) sh run_server.sh in 1 terminal,

(3) sh run_client.sh in another terminal,

where run_server.sh and run_client.sh contains the code for server 1 and client processes.

ii) Version2

(1) cd Version2

(2) sh run_server.sh in 1 terminal,

(3) sh run_client.sh in another terminal,

where run_server.sh and run_client.sh contains the code for server2 and client processes.

iii) Version3

(1) cd Version3

(2) sh run_server.sh in 1 terminal,

(3) sh run_client.sh in another terminal,
where run_server.sh and run_client.sh contains the code for server3 and client processes.

We have implemented a client-server communication system where a client can communicate with a dummy math server of 3 kinds using the concept of socket communication. A socket allows multiple hosts to communicate with each other using a network. A socket has such a description (IP address: Port number) where the IP is known by the network and port by the host.

Working: The socket present at the server-side waits for connection requests from a client. To do this, the server first establishes (binds) an address that clients can use to find the server. When the address is established, the server waits for clients to request a service. The client-to-server data exchange takes place when a client connects to the server through a socket. The server performs the client's request and sends the reply back to the client.

Server 1:

- ❖ In this server, only 1 client is allowed to connect with the server at a given time.
- ❖ Compile and run the server-side code using runserver.sh file
- ❖ Compile and run the client-side code using runclient.sh file
- ❖ Open a new terminal for handling the server and client separately.
- ❖ first run the server code then the client code since, if the server is not there, no one will be able to host the client.
- ❖ The user will be prompted to enter a string, the server will return the computed answer for the same.
- ❖ If another client requests to get connected it will receive a message which says “line busy !”
- ❖ Here 5555 is the port number and 127.0.0.1 is the IP address.
- ❖ Until ctrl+c is hit by the user, the connection will not be broken.

NOTE: Please disconnect the client(don’t abandon) and server, before testing for other parts of the question since it may cause some errors then, in such cases close all the terminals and run that particular server from scratch.

Server 2:

❖ In this server, multiple clients will be able to communicate with the server, the server here will fork a process whenever it receives a request from a new client.
❖ Compile and run the server-side code using runserver.sh file
❖ Compile and run the client-side code using runclient.sh file
❖ Open a new terminal for handling the server and client separately.
❖ first run the server code then the client code since, if the server is not there, no one will be able to host the client
❖ Run another client in a different terminal to see multiple clients requests being handled
❖ The user will be prompted to enter a string, the server will return the computed answer for the same.
❖ Here 5555 is the port number and 127.0.0.1 is the IP address.
❖ Until ctrl+c is hit by the user, the connection will not be broken.

Server 3:

❖ In this, we have implemented a multithreaded server that uses select system calls to handle multiple client requests concurrently.
❖ Compile and run the server-side code using runserver.sh file
❖ Compile and run the client-side code using runclient.sh file
❖ Open a new terminal for handling the server and client separately.
❖ Compile the server-side code using: gcc server_3.c -o server_3
❖ first run the server code then the client code since, if the server is not there, no one will be able to host the client
❖ Run another client in a different terminal to see multiple clients requests being handled
❖ The user will be prompted to enter a string, the server will return the computed answer for the same
❖ Here 5555 is the port number and 127.0.0.1 is the IP address.

❖ Until ctrl+c is hit by the user, the connection will not be broken.

Extra information:

❖ The **bzero()** function places n byte null bytes in the string s. This function is used to set all the socket structures with null values.

❖ The **htons**() function takes the port number (5555) converts the unsigned
short integer host short from host byte order to network byte order.

❖ We have used **INADDR_ANY** to specify the IP address.

❖ **bind**() function binds the socket to the address and port number specified in addr.

❖ The **listen**() call indicates a readiness to accept incoming client connection requests. It transforms an active socket into a passive socket.

❖ The **accept**() call is used by a server to accept a connection request from a client. When a connection is available, the socket created is ready for use to read data from the process that requested the connection. The call accepts the first connection on its queue of pending connections for the given socket.

❖ The **close**() function shuts down the socket associated with socket descriptors and frees resources allocated to the socket.
