CONSOLE BASED MULTI USER CHAT APPLICATION USING SOCKET PROGRAMMING

Socket is an end point of a two-way communication between two programs running on a network. They provide a means for inter process communication between hosts in the same or different network.

Here, we use Berkeley Sockets which is available as a module in Python, for our application. The primary socket API functions and methods in this module are: **socket(), bind(), listen(), accept(), connect(), send(), recv(), close(**). We use IPv4 addresses and Transmission control protocol (TCP) as the standards for our program. TCP is preferred because it is reliable, and it has in order data delivery mechanism.

Now we discuss the internals of the server and client sides.

**Server Side**: First we create a socket. In python this socket creation can be done with the help of socket function that takes in parameters such as internet address family type and socket type. For this project we simply use the loop back address as the IP and any port number from 1024 to 65535. This is done because port number 0 is reserved and port number 1-1023 can only be used in privileged mode. The next step would be to bind this newly created socket with the IP and the port. We can use a try except block to catch any error that can be thrown while binding. Now we are making the socket a listening socket. A listening socket detects connection from clients. We can set the maximum number of clients we want the server to be connected to. The function accept() suspends the program. When a client requests a connection the server calls accept(). Now the three-way handshake happens and completes the connection. Once the connection has been established, we can send and receive messages between the server and the client. To terminate the connection the server calls close() and the port is released.

**Client Side**: The client-side connection is much simpler than server side. We simply create a socket object as we did earlier and make a connection request to the server following which, message transmission and reception is viable. We can also terminate a socket existence by calling the close() command as done earlier.

One key thing to note here is that the socket that has been used for connection establishment is different from the socket that will be used to communicate with the client. The communication socket is returned by the accept() command.

**Threading:**

The key objective of our application is the concurrent handling of multiple connections. There are various ways to achieve concurrency, with threading being the most traditional and tested option.

A thread is a lightweight process. It is the smallest unit that can be run on a CPU. Threads of same process share the data and code section. Stack, registers, program counter etc. are not shared and are unique to each thread. Threading has multiple benefits. It ensures efficient use of resources. Because the code and data sections are shared, the context switch times are negligibly small. It is extremely useful to achieve high level of concurrency in the system. Multi-threading in a multi-processor system gives highest degree of concurrency. In our application we make use of multi-threading to establish separate connections from the clients to the server. Each thread runs independently of the other and quitting one thread has no effect on other running threads. A practical usage of this feature of threads can be thought of as exit of one user from a multi-user chat doesn’t terminate the entire conversation.

A close up of text on a black surface

Description automatically generated

A working demo of the program is shown above. 3 clients join (send connection requests to the server) at different instants of time and leave at separate instants too and their texts are visible on the console running the server code. It is similar to web-based chat applications albeit simpler and on a much smaller scale.