OPERATING SYSTEMS IT308 PROJECT REPORT

Project Title: Multi-threaded Web Server in JAVA

Group Members:

201401072 – N. Srisai Karthik
201401080 – P. Pranay Kumar

INTRODUCTION

This project involves the client-server based multi-threaded web server. There is a server which accepts requests from clients continuously and handles these requests accordingly. Socket programming is used to establish connection between the client and server. Before proceeding to the implementation of the web server, we will first look at how the socket is created and a connection is established.

What is a SOCKET?

A socket is the one end-point of a two-way communication link between two programs running over the network. In other words, a socket is generally a data structure that contains an internet address and a port number. The two-way communication link between the two programs running on different computers is done by reading from and writing to the sockets created on these computers. The data read from a socket is the data wrote into the other socket of the link. Reciprocally, the data wrote into a socket is the data read from the other socket of the link. These two sockets are created and linked during the connection creation phase. The link between two sockets is like a pipe that is implemented using a stack of protocols.

How is a network connection created?
A network connection is initiated by a client when it creates a socket to communicate with the server. To create a socket in Java, the client calls the *Socket* constructor, passes the server address and the specific port number to it. By this time, the server must be started on the machine having the specified address and it should be listening for connections on its port number.

The server uses a specific port dedicated only to listen for requests from clients. It can't use this specific port for data communication with a client because the server should be able to accept the client connection at any instant. The server side socket associated with specific port is called *Server Socket*. When a connection request arrives on this socket from the client side, the client and the server establish a connection. This connection is established in the following sequence:

1. When the server receives a connection request, it creates a new socket for the client and binds a port number to it.
2. The server sends this new port number to the client to inform it that the connection is established.
3. The server goes on now using two ports:
    - One port to continue listening for new requests
    - Other port to read and write on established connection with the accepted client.

The server communicates with the client by reading from and writing to the new port (socket). If other connection requests arrive, the server accepts them in a similar way by creating a new port for each request. Thus, at any instant, the server is able to communicate simultaneously with many clients and wait for incoming requests. The communication with each client is done via the sockets created for each communication.

TCP (Transfer Control Protocol) is used for communication. TCP is a connection-oriented protocol. In order to communicate over the TCP protocol, a connection must be first established between two sockets.

IMPLEMENTATION

This web server is implemented using four classes -

- *Initializer* : used for initializing a web server
- *Server* : used to listen for the requests and handle them
- *Client* : used for generating requests to the web server
- *ClientHandler* : used to process the client requests

Initializer: This class contains a main method which is used to start a web server with IP address *"localhost"* or 127.0.0.1. This IP address is a loopback address which refers to the computer which is being used itself. Port number for this web server is taken by default as *8080* (used generally for personally hosted web servers). However, any port number from 0 to 65535 can be specified while initializing the server by giving input through the command line. An instance of the class *Server* is created and initialized with the specified port number. A thread is used to execute the *run()* method of this server instance.

Server: This class uses three main attributes -

- *serverSocket* - of the data type *ServerSocket*, which is used by the server applications to obtain a port and listen for the client requests
- *serverPort* - of the data type *int*, port number for server
- *serverHost* - of the data type *String*, host IP address

Once the server is started by the calling thread, serverSocket is initialized with the given port number and the host IP address. The server displays a message suggesting that the server has started at HOST *localhost* and PORT *8080*.

The server, now, starts to wait for the requests to be made by the clients using the in-built method *serverSocket.accept()*, which listens for a connection to be made to the socket and accepts it. It blocks until a connection is made and returns the *clientSocket* once a connection is made with a client. A variable *clientID* is used to distinguish client requests and keep track of the number of clients connected to the web server from its initialization. The server displays a message suggesting that a connection has been established with client *clientID* at *IP:PORT*.

After a connection is established, the server handovers the *clientSocket* and *clientID* to the *ClientHandler* class which proceeds with handling the request made by the client to the web server and continues to listen for any new client requests.

<u>Client</u>: This class contains a main method which is used to generate a client request for the web server. This class should be run once the server is initialized. On executing the main method, the client is asked to provide one or two or three arguments each separated by single space. If no argument is passed, the program exits displaying an error.

- If the number of arguments is 1, it is assumed to be the IP address of the host (server) to which the client wishes to connect. In this case, the port number is assumed to be 8080 (DEFAULT).
- If the number of arguments is 2, the first argument corresponds to the host IP address and the second argument corresponds to either the port number or the requested file path. If the second argument turns out to be the file path, the port number is assumed to be 8080 (DEFAULT).
- If the number of arguments is 3, the first argument corresponds to the host IP address, the second argument corresponds to the port number and the third one indicates the requested file path.

After reading the required data, a *clientSocket* of the class *Socket* is defined and initialized with the *serverIP* and the *serverPort* mentioned above. Later, for reading bytes from this socket and for writing bytes to this socket, a *socketInStream* and *socketOutStream* are used and initialized using the input and output streams of *clientSocket* as *clientSocket.getInputStream()* and *clientSocket.getOutputStream()*.

The request is now written to the *socketOutStream* using the in-built method *writeBytes()*. To terminate the request line, a carriage return (\r) immediately followed by a line feed (\n) is appended i.e. the string “\r\n” is concatenated to the request line. After sending these bytes, the client waits for a response from the server.

socketInStream is now used to read data from the *clientSocket*. The first line contains the status line (messages such as OK, NOT FOUND). A string builder is used to store all the data read from the *socketInStream* and it is printed to the console. A FileOutputStream *fos* is used to write these bytes into a file using the in-built method *fos.write()*. After the completion, the stream is flushed.

socketInStream, socketOutStream, fos and *clientSocket* are closed after the completion of the execution of a client request. Exceptions are handled accordingly. This class also includes a method *getFileName()* which helps in extracting title of the HTM/HTML file in case of GET HTTP requests. The extracted file name is appended with the string “.htm” or “.html”. Appropriate messages are printed to the console by the client to indicate the status of the execution.

ClientHandler: This class is initialized by the class *Server* when there is a new client connected to the server. The function of this class is to process the requests made by the client. As in the case of the *Client* class, *socketInStream* and *socketOutStream* are used for reading and writing into the *clientSocket*.

socketInStream reads the request line sent by a client and executes it. First of all, the request line is split and analyzed based on the components of the request line. If no file path is mentioned for a GET request, a default file is returned to the client. If a file path is requested, a check is made if the file actually exists on the given server. If no such file exists, the *ClientHandler* sends the NOT FOUND bytes to the *socketOutStream.*

If the file exists, the handler uses a FileInputStream *fis* to read bytes from the requested file and writes these bytes to the *socketOutStream*. After completion of this, *fis* is flushed.

socketInStream, socketOutStream, fis and *clientSocket* are closed after the completion of the execution of a client request. Exceptions are handled accordingly. Appropriate messages are printed to the console by the client to indicate the status of the execution.

CONCLUSION

Java sockets API (Socket and ServerSocket classes) is a powerful and flexible interface for network programming of client/server applications. On the other hand, Java threads is another powerful programming framework for client/server applications. Multi-threading simplifies the implementation of complex client/server applications. However, it introduces synchronization issues if there are any critical sections. These issues are caused by the concurrent execution of critical sections of the program by different threads. The synchronized(this){} statement allows us to synchronize the execution of the critical sections.