COMP 8005 Assignment 2 Design

Alex Zielinski – A00803488

03/05/2018

Contents

Overview

The purpose of this assignment is to evaluate server performance among three different server techniques (multi-threaded, multiplexed and asynchronous). The assignment project folder contains four executables in the ***/bin*** folder. The client program executable is called ***clt_thread***. This program creates worker threads in order to simulate multiple client connections (makes use of openMP). There are three server program executables. The first one is called ***srv_thread***. This program represents a traditional multithreaded server where the server creates a new thread to accommodate each new client connection (the server makes use of Posix threads). The second server executable is called ***srv_poll***. This program represents a multiplexed server and makes use of the ***poll()*** API call (which is level triggered). And lastly, the third server executable is called ***srv_epoll***. This program represents an asynchronous server and makes use of the ***epoll()*** API call (which is edge triggered).

The client program and each server program track some basic statistics regarding each transmission session. These statistics are then written to their respective log files at the end of each session. The log files are saved within the folder called ***/data.***

Note: Please refer to the user guide found in this document for detailed instructions on how to compile and run the programs. Please refer to the Report document for an analysis regarding the performance among the three servers.

User Guide

Program Macros

There are two important user defined macros that need to be discussed first. The client's header file called ***clt_thread.h*** (found in the folder ***/includes***) contains a macro called ***TIMEOUT*** on line 15. This macro determines how long each client will maintain a connection with the server. So, if the macro is set to 20 then each client will send packets to the server and read the echo from the server for 20 seconds. Once the 20 seconds have finished then the client disconnects from the server and shuts down.

All of server header files as well as the client header file contains a user defined macro called ***PKTSIZE***. In ***clt_thread.h*** the ***PKTSIZE*** macro is found on line 14 and tells the client how big of a packet it should send to the server (in bytes). In all of the server header files (***srv_thread.h , srv_poll.h , srv_epoll.h***) the ***PKTSIZE*** macro is found on line 12 and tells the server how big of a packet it should be expecting to receive. Now, the ***PKTSIZE*** macro in the server header files and the client header file need to be the same otherwise unexpected behavior will occur.

Compiling the Code

In order to compile the code, open up a terminal and navigate to the assignment's project directory. In this directory there is a makefile. Within terminal simply type ***make*** and press enter. This will compile the client and server code and will create four executables within the ***/bin*** folder. Enter ***makeclean*** the clean the project if need be.

Running the Code (Client)

The client and server programs should be run on different machines. So, the client code should be run on ***machine A*** and the server code should be run on ***machine B***.

In order to run the client program on machine A, within terminal, navigate to the ***/bin*** folder of the assignment's project directory. Then enter the following command:

./clt_thread <HOST IP> <PORT> <NUM OF CLIENTS>

The client program takes three command line arguments. The first argument specifies the server IP that a client will connect to. The second argument specifies the server's listening port that a client will connect to. And the third argument specifies how many threads the client program will spawn. Where each thread represents a separate client. Each thread will connect to the server and transmit data. During the transmission session each client tracks some basic data regarding the exact time they connected to the server, the number of requests that were processed and the amount of data that was transferred (in bytes). The client program will end transmission and shutdown once ***TIMEOUT*** has occurred.

*Note: if the user has not navigated to the **/bin** folder to run the program and executes the program from a different directory then the program will not run as there will be an error regarding the creation of log files.*

The lab machine firewall may block the client and server programs from interacting with each other. In such a case reset the firewall on each machine that is running an instance of either a client or server program and set the default policy to ACCEPT via the following commands:

iptables -F
iptables -X
iptables -P INPUT ACCEPT
iptables -P OUTPUT ACCEPT
iptables -P FORWARD ACCEPT

*Also, the **ulimit** (limits the allowed number open files) may be set to **1024** which is not enough. Set the ulimit max to a larger number as by running the following command on each machine running an instance of either a client or server program:*

ulimit -n 50000

Running the Code (Server)

As mentioned earlier the client and server programs should be run on different machines. In order to run any of the server programs on machine B, navigate to the ***/bin*** folder within the assignment project directory with terminal. Depending on which server you would like to start, enter the respective command:

./srv_thread <HOST IP> <PORT>

./srv_poll <HOST IP> <PORT>

./srv_epoll <HOST IP> <PORT>

An important note here is that the multiplexed and asynchronous servers have a timeout that is 5 seconds. This means that if either of the servers are inactive for 5 seconds they shutdown. The multi-threaded server however has no timeout and one must use ***ctrl + c*** to shutdown the server.

Project folder directory listings

/bin

|------ clt_thread
|------ srv_epoll
|------ srv_poll
|------ srv_thread

/data

|------ client_log
|------ srv_thread_log
|------ srv_poll_log
|------ srv_epoll_log

/include

|------ clt_thread.h
|------ srv_thread.h
|------ srv_poll.h
|------ srv_epoll.h
|------ socket.h
|------ log.h

/src

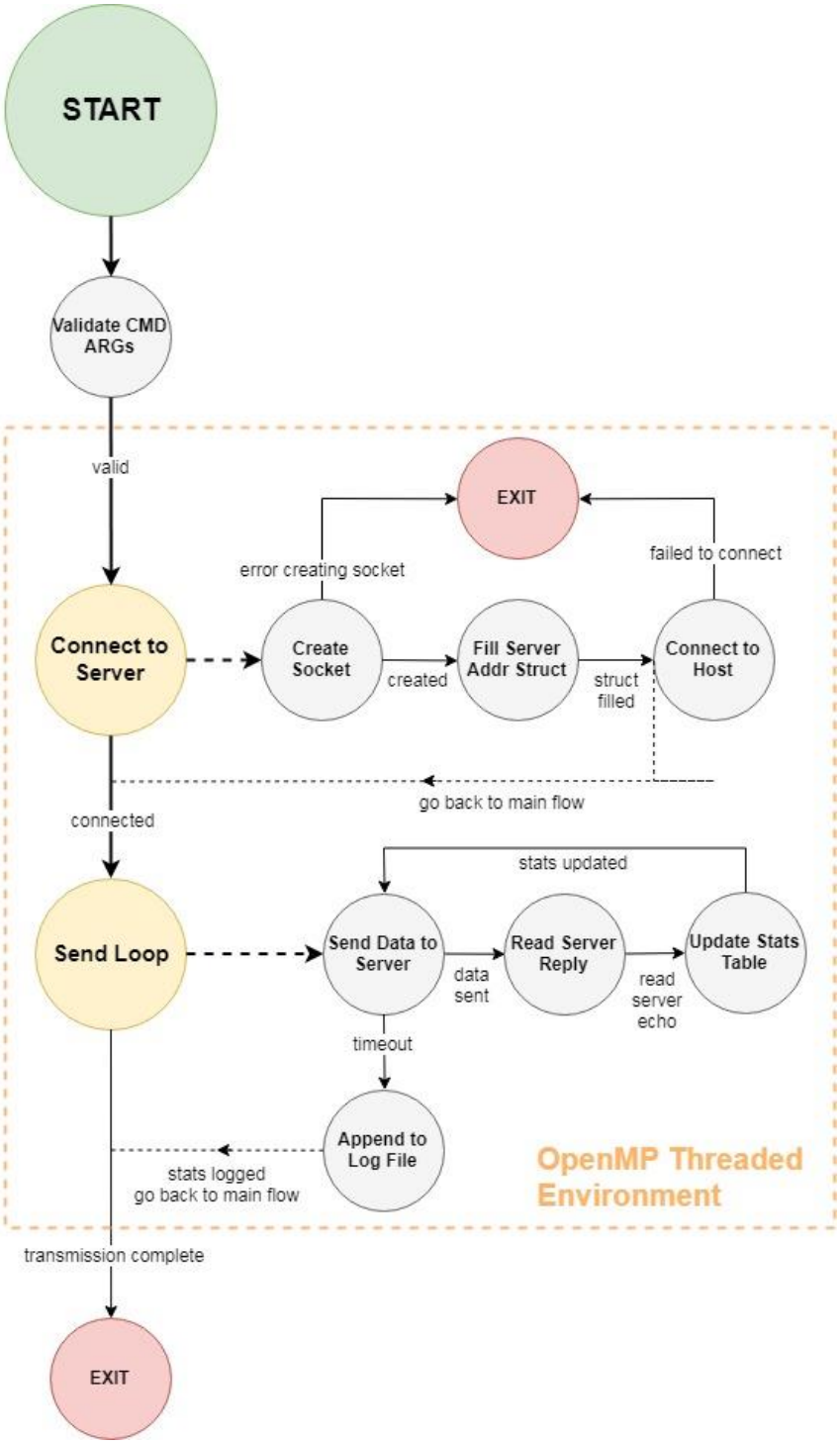
|------ clt_thread.c
|------ srv_thread.c
|------ srv_poll.c
|------ srv_epoll.c
|------ socket.c
|------ log.c

makefile
Design Doc
Testing Doc
Report Doc

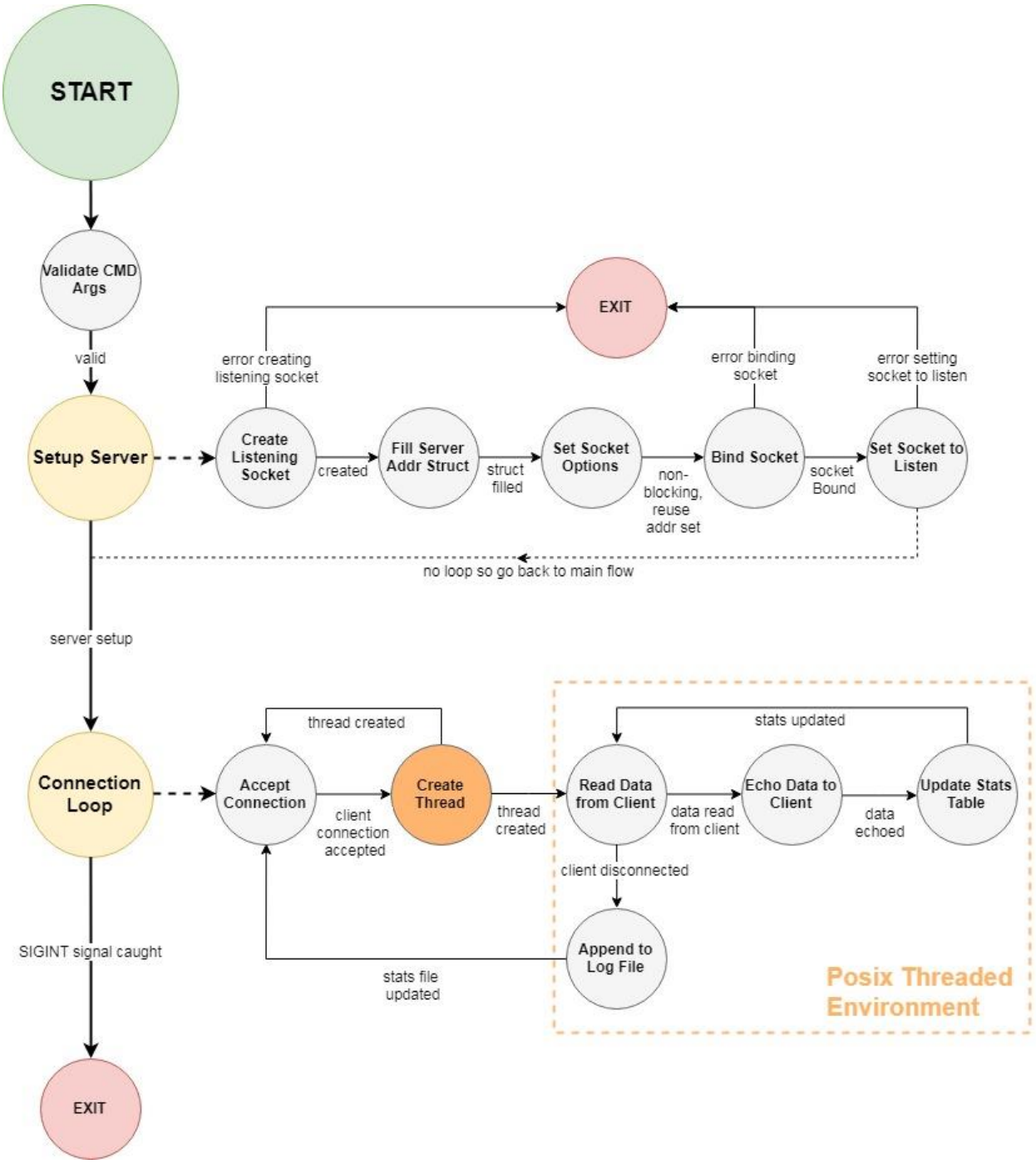
*Note: by default, there are no files within **/data**. The log files that are placed here are only created once either a server of client program have run. However, for the purpose of showing the project folder directory listings I included all of the log files that may exist inside **/data**.*

Finite State Machines

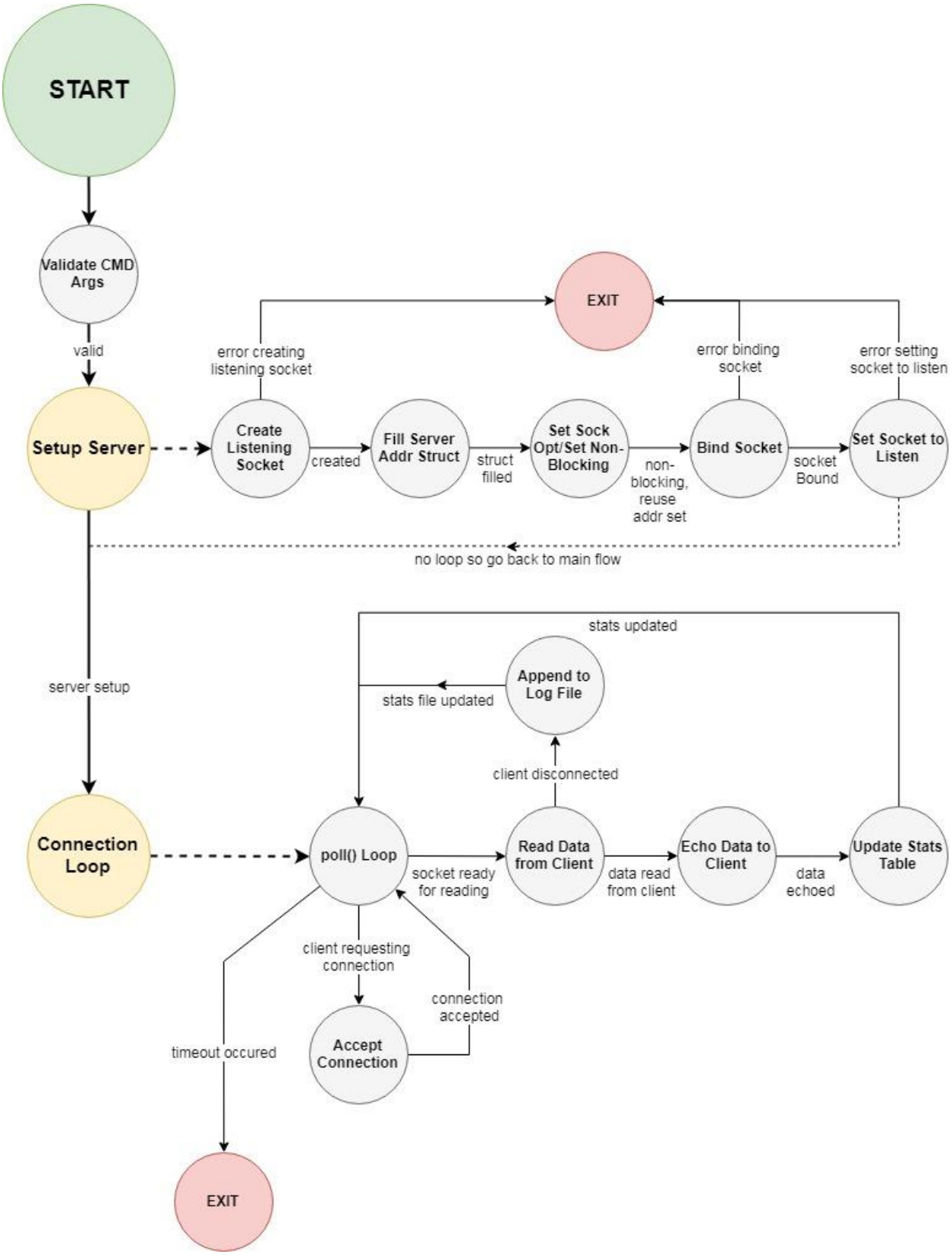
Multi-Threaded Client



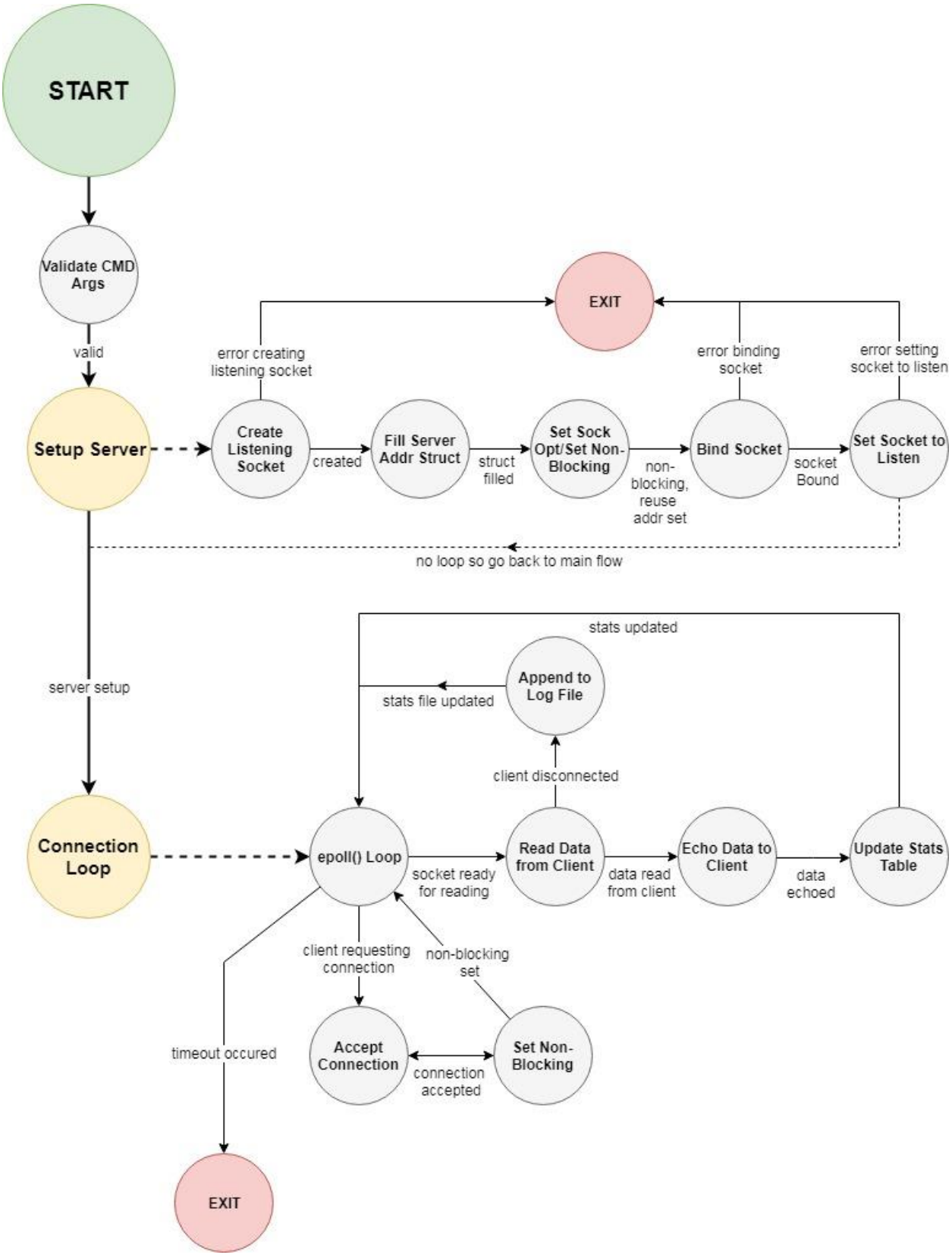
Multi-Threaded Server



Multiplexed Server (poll)



Asynchronous Server (epoll)



Pseudo Code

Multi-Threaded Client

Validate CMD ARGs

Check if the number of CMD ARGs is four
If not, then EXIT and print usage statement
Check if each character of the 'port' argument is a digit
If not, then EXIT with an error message
Check if each character of the 'number of clients' argument is a digit
If not, then EXIT with an error message, otherwise go to **Create Socket**

openMP SCOPE START (code from here until '*scope END*' will execute within an openMP threaded environment)

Create Socket

Create a socket that will be used to connect to server
If an error occurred, then EXIT with an error message, otherwise go to **Fill Server Addr Struct**

Fill Server Addr Struct

Fill in the addr struct that specifies the host and port to connect to and go to **Connect To Host**

Connect to Host

Connect to server via the created socket using the addr struct that specifies connection details
If an error occurred, then EXIT with an error message
Get the current date and time and save the date to the stats table, then **Send Data to Server**

Send Data to Server

Check if timeout has occurred
If timeout occurred then **Append to Log File**, close the socket and EXIT
Fill a packet with an arbitrary character
Send the packet to the server
If an error occurred, then EXIT with an error message
Updates the number of requests made within the stats table and go to **Read Server Reply**

Read Server Reply

Read echo reply from server
If an error occurred, then EXIT with an error message
Update current total bytes transferred within the stats table
Update the current average server response time and go to **Send Data to Server**

Append to Log File

Open or create log file
If an error occurred, then EXIT with an error message
Append stats data to log file

openMP SCOPE END

Multi-Threaded Server

Validate CMD ARGs

Check if the number of CMD ARGs is two
If not, then EXIT and print usage statement
Check if each character of the 'port' argument is a digit
If not, then EXIT with an error message, otherwise go to **Create Listening Socket**

Create Listening Socket

Create a socket that will be used to listen for connections
If an error occurred, then EXIT with an error message, otherwise go to **Fill Server Addr Struct**

Fill Server Addr Struct

Fill in the addr struct that specifies the listening port and the addresses to accept
Then go to **Set Socket to Reuse Addr**

Set Socket to Reuse Addr

Set the options for the listening socket to reuse the address
If an error occurred, then EXIT with an error message, otherwise go to **Bind Socket**

Bind Socket

Bind the listening socket using the addr struct that specifies the port to bind to
If an error occurred, then EXIT with an error message, otherwise go to **Set Socket to Listen**

Set Socket to Listen

Set the socket to listen for connections
If an error occurred, then EXIT with error message, otherwise go to **Accept Connection Loop**

Accept Connection Loop

Accept a client connection request
If an error occurred, then EXIT with an error message
Otherwise get the current date and time and save the date to the stats table
Update total number of connected clients within the stats table
Save client IP
Create a new thread to accommodate the new client connection ***(Posix threaded environment)***
If an error occurred, then EXIT with an error message, otherwise go to **Read Data from Client**

Read Data from Client

Read packets coming from client
If an error occurred, then EXIT with an error message
Otherwise update current total number of client requests within the stats table
If client disconnected then go to **Append to Log File**, close connected socket and EXIT thread
Otherwise go to **Echo Data to Client**

Echo Data to Client

Send the received data back to the client
If an error occurred, then EXIT with an error message
Update current total amount of bytes transmitted within the stats table
Then go to **Read Data from Client**

Append to Log File

Open or create log file
If an error occurred, then EXIT with an error message
Append stats data to log file

Multiplexed Server (poll)

Validate CMD ARGs

Check if the number of CMD ARGs is two
If not, then EXIT and print usage statement
Check if each character of the 'port' argument is a digit
If not, then EXIT with an error message, otherwise go to **Create Listening Socket**

Create Listening Socket

Create a socket that will be used to listen for connections
If an error occurred, then EXIT with an error message, otherwise go to **Fill Server Addr Struct**

Fill Server Addr Struct

Fill in the addr struct that specifies the listening port and the addresses to accept
Then go to **Set Socket to Reuse Addr**

Set Socket to Reuse Addr

Set the options for the listening socket to reuse the address
If an error occurred, then EXIT with an error message,
Otherwise go to **Set Socket to Non-Blocking**

Set Socket to Non-Blocking

Set the listening socket to non-blocking (return immediately on blocking calls)
If an error occurred, then exit with an error message, otherwise go to **Bind Socket**

Bind Socket

Bind the listening socket using the addr struct that specifies the port to bind to
If an error occurred, then EXIT with an error message, otherwise go to **Set Socket to Listen**

Set Socket to Listen

Set the socket to listen for connections
If an error occurred, then EXIT with error message, otherwise go to **POLL Loop**

POLL Loop

Add listening socket to poll table

Loop 1

Wait for a socket event to occur
If an error occurred, then close listening socket and exit with an error message
If timeout occurred, then close listening socket and exit
Otherwise check for connection request
If there is a new connection request, then go to **Accept Connection**

Loop 2

Check if socket is available for reading, then go to **Read Data from Client**
if not then leave loop 2

Loop 2

Loop 1

Accept Connection

Accept new client connection
Add new client socket to poll table
Get the current date and time and save the date to the stats table
Update total number of connected clients within the stats table and save the clients IP

Read Data from Client

Read packet from client
If an error occurred, then close the client socket and remove socket from poll table
If client disconnected from server then go to **Append to Log File**
Otherwise update current total number of client requests within the stats table
Go to **Echo Data to Client**

Echo Data to Client

Send received data back to client
Update current total amount of bytes transmitted within the stats table

Append to Log File

Open or create log file
If an error occurred, then EXIT with an error message
Append stats data to log file

Asynchronous Server (epoll)

Validate CMD ARGs

Check if the number of CMD ARGs is two
If not, then EXIT and print usage statement
Check if each character of the 'port' argument is a digit
If not, then EXIT with an error message, otherwise go to **Create Listening Socket**

Create Listening Socket

Create a socket that will be used to listen for connections
If an error occurred, then EXIT with an error message, otherwise go to **Fill Server Addr Struct**

Fill Server Addr Struct

Fill in the addr struct that specifies the listening port and the addresses to accept
Then go to **Set Socket to Reuse Addr**

Set Socket to Reuse Addr

Set the options for the listening socket to reuse the address
If an error occurred, then EXIT with an error message, otherwise go to **Bind Socket**

Set Socket to Reuse Addr

Set the options for the listening socket to reuse the address
If an error occurred, then EXIT with an error message,
Otherwise go to **Set Socket to Non-Blocking**

Bind Socket

Bind the listening socket using the addr struct that specifies the port to bind to
If an error occurred, then EXIT with an error message, otherwise go to **Set Socket to Listen**

Set Socket to Listen

Set the socket to listen for connections
If an error occurred, then EXIT with error message, otherwise go to **POLL Loop**

EPOLL Loop

Create epoll socket
If an error occurred, close the listening socket and EXIT with error message
Otherwise instantiate epoll table
If an error occurred, close the listening socket and EXIT with error message
Loop 1
Wait for a socket event to occur
If an error occurred, then close listening socket and exit with an error message
If timeout occurred, then close listening socket and exit, otherwise socket events occurred
Loop 2
If there is a new connection request, then go to **Accept Connection**
If there is data ready to be read, go to **Read Data from Client**
If there are not more events, then leave loop 2
Loop 2
Loop 1

Accept Connection

Loop 1
Accept new client connection
Add new client socket to epoll table
Get the current date and time and save the date to the stats table
Update total number of connected clients within the stats table and save the clients IP
If no more clients to accept then leave loop 1
Loop 1

Read Data from Client

Read packet from client
If an error occurred, then close the client socket and remove socket from poll table
If client disconnected from server then go to **Append to Log File**
Otherwise update current total number of client requests within the stats table
Go to **Echo Data to Client**

Echo Data to Client

Send received data back to client
Update current total amount of bytes transmitted within the stats table

Append to Log File

Open or create log file
If an error occurred, then EXIT with an error message
Append stats data to log file