COMP 8005 - Assignment #2

Mario Enriquez, Charles Kevin Tan

British Columbia Institute of Technology

COMP 8005, COMP 6D

Aman Abdullah

2016-02-15

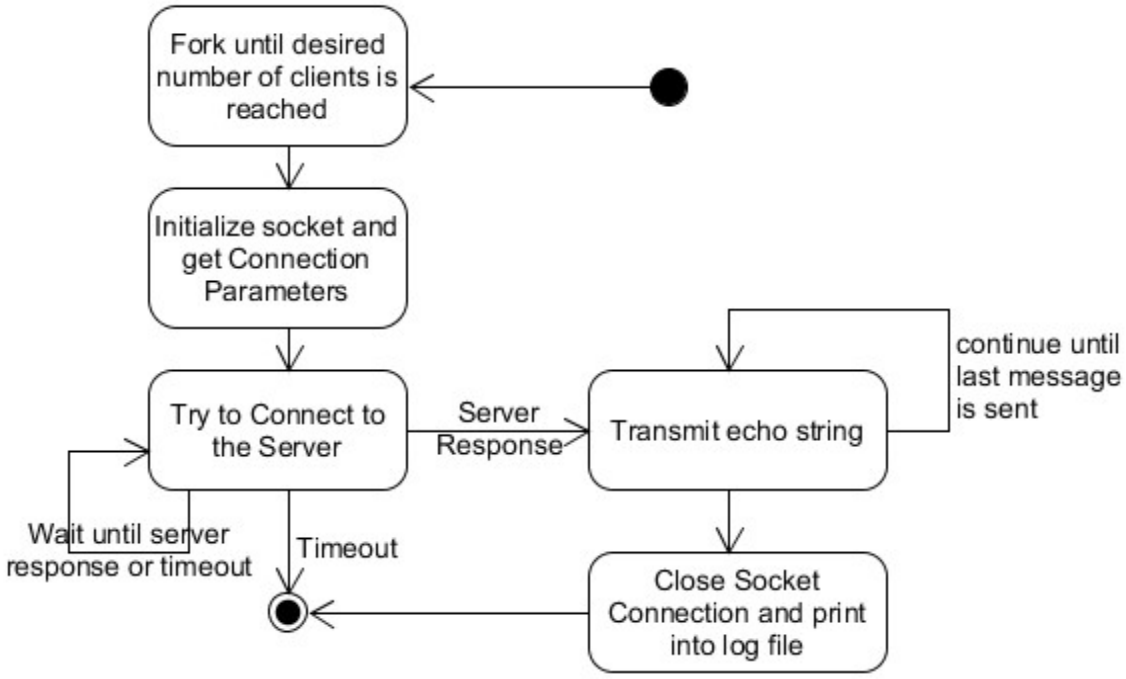
Contents

Introduction

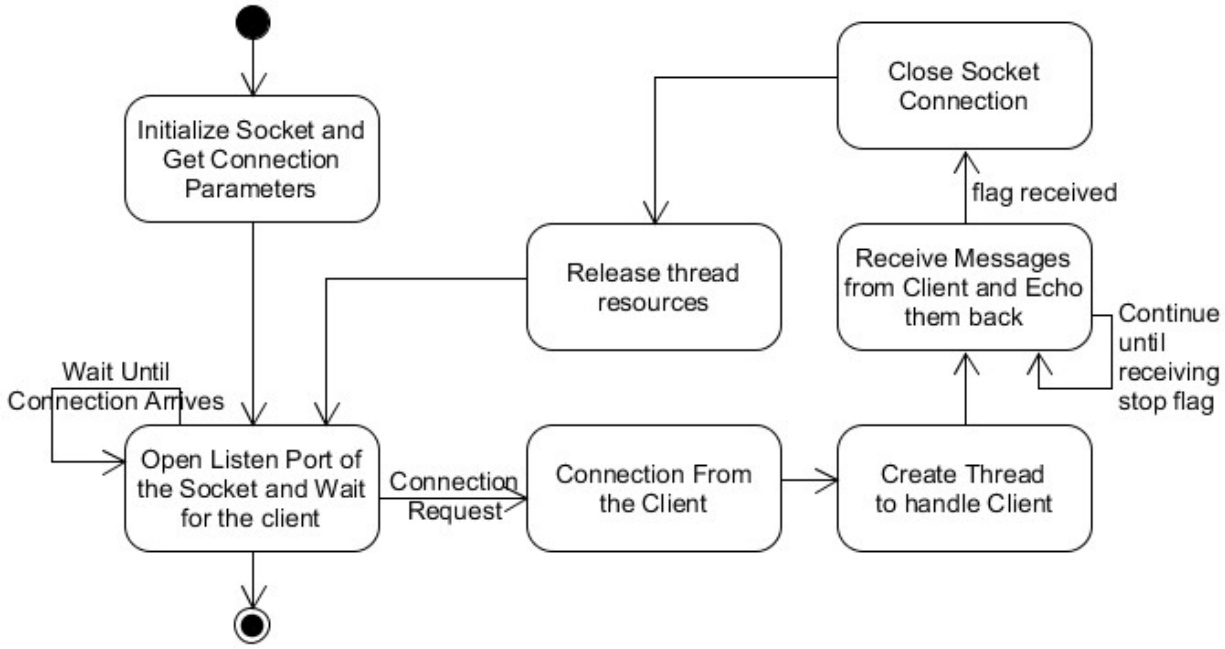
In this assignment, we will be implementing three types of servers: multithreaded, select and epoll. The goal is to determine the scalability and performance of these three servers by continuously increasing the load of the servers. To do that, we will be initializing how many connection requests to be sent to the server starting from 5,000 and increase the load to 10,000 to see a difference in performance between the three servers. Additionally we will vary the number of request per client from 200 requests to 300, to make the connections last longer.

Design Work

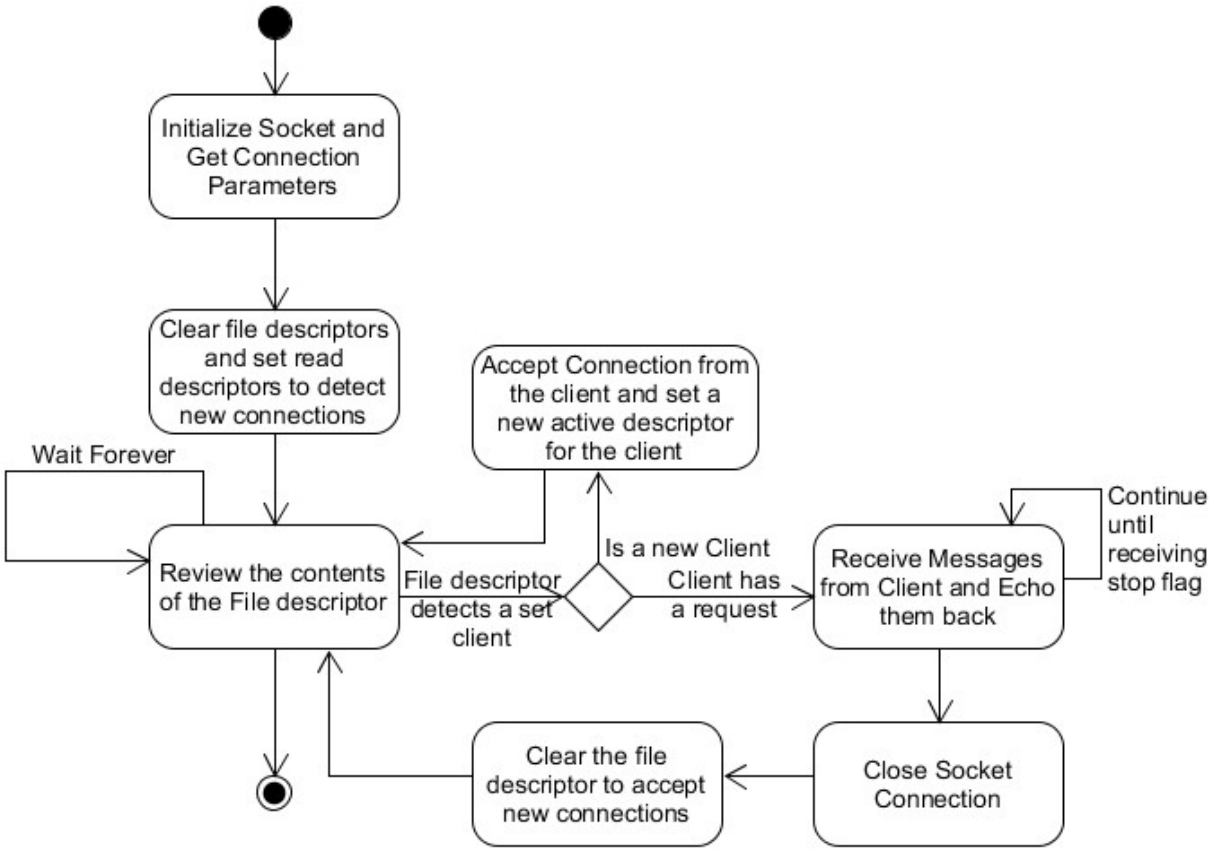
Client



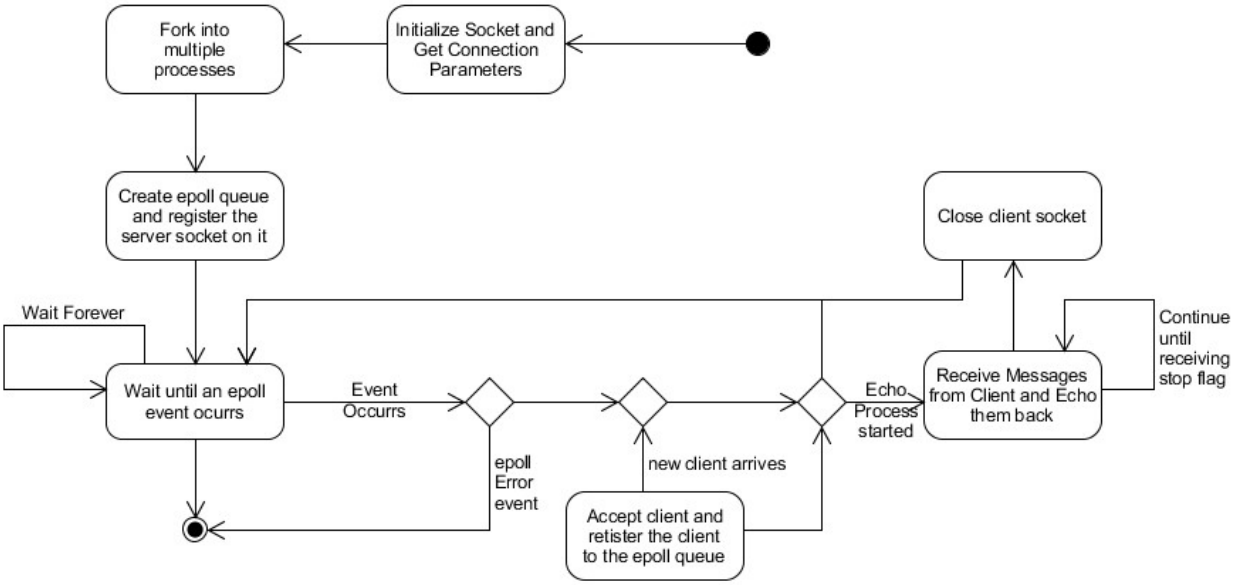
Multi-threaded Server



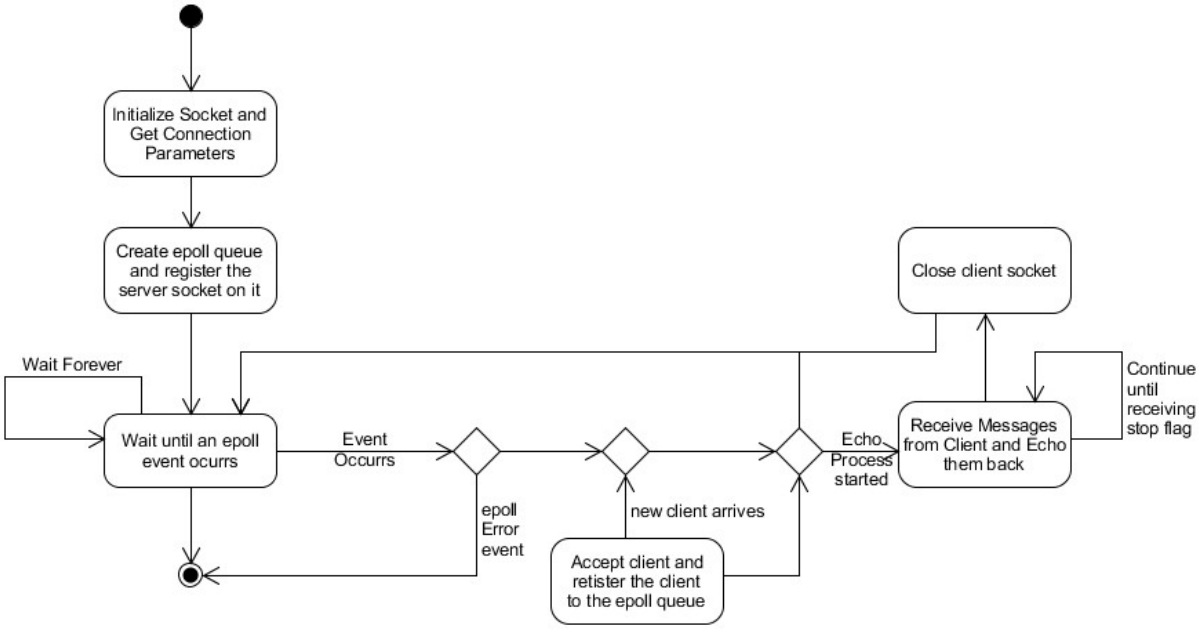
Select Server



Select Server with fork

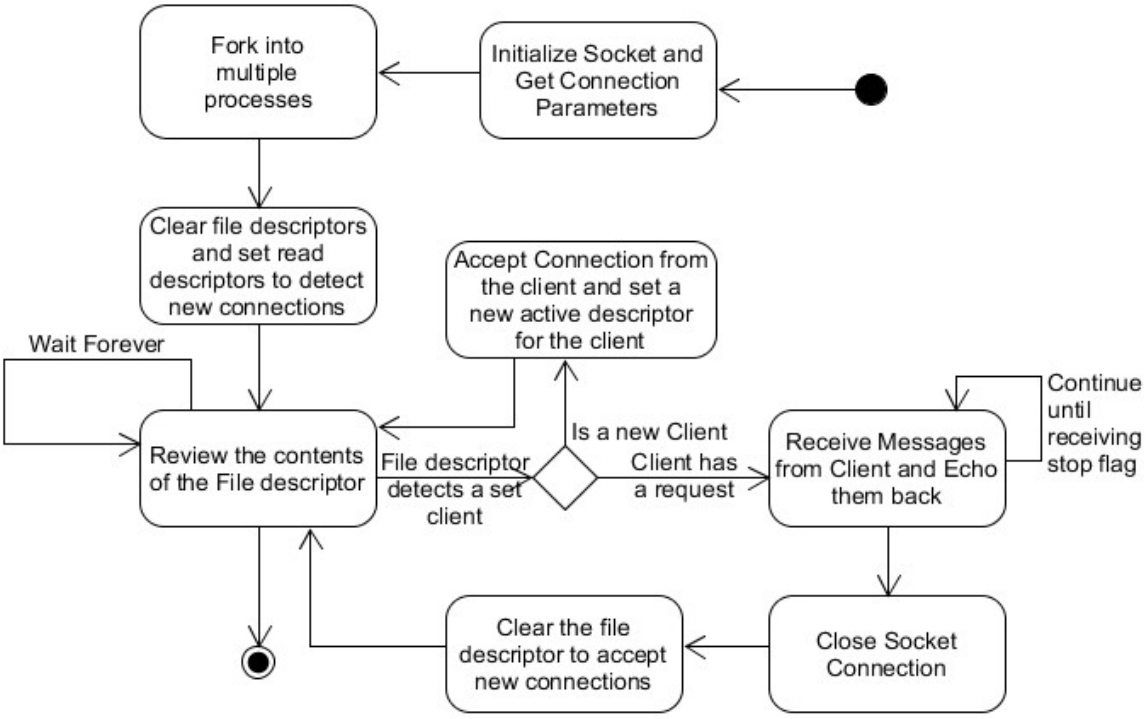


Epoll Server



The design of the servers were based on the class/book designs.

Epoll Server with fork



Instructions

Run in c:

Client:
./client host [port] [client no] [iterations] [message]

Select:
./select [port]

Multi-threading:
./threading [port]

Epoll
./epoll [port]

Test Cases

Test	Description	Tool/Server	Expected Results	Actual Results

1	Test the performance of the server when it accepts 5000 clients with 200 exchanges between them	Multithread Server	The server has no problem handling this load	Pass, The server has no problem handling this load
2	Test the performance of the server when it accepts 5000 clients with 200 exchanges between them	Select Server	The server has no problem handling this load	Pass, The server has no problem handling this load
3	Test the performance of the server when it accepts 5000 clients with 200 exchanges between them	Epoll Server	The server has no problem handling this load	Pass, The server has no problem handling this load
4	Test the performance of the server when it accepts 5000 clients with 300 exchanges between them	Multithread Server	Test a load that my cause trouble to the server	Pass, The server has no problem handling this load
5	Test the performance of the server when it accepts 5000 clients with 300 exchanges between them	Select Server	Test a load that my cause trouble to the server	Pass, The server has no problem handling this load
6	Test the performance of the server when it accepts 5000 clients with 300 exchanges between them	Epoll Server	Test a load that my cause trouble to the server	Pass, The server saw a little drop on performance
7	Test the performance of the server when it accepts 10000 clients with 200 exchanges between them	Multithread Server	The server has no problem handling this load of clients and requests	Pass, The server has no problem handling this load
8	Test the performance of the server when it accepts 10000 clients with 200 exchanges between them	Select Server	The server has no problem handling this load of clients and requests	Pass, The server has no problem handling this load
9	Test the performance of the server when it accepts 10000 clients with 200 exchanges between them	Epoll Server	The server has no problem handling this load of clients and requests	Pass, The server saw a little drop on performance
10	Test the performance of the server when it accepts 10000 clients with 300 exchanges between them	Multithread Server	Test a relatively heavy load of clients and requests	Pass, the server passed without problems

11	Test the performance of the server when it accepts 10000 clients with 300 exchanges between them	Select Server	Test a relatively heavy load of clients and requests	Pass, the server started having trouble handling the load
12	Test the performance of the server when it accepts 10000 clients with 300 exchanges between them	Epoll Server	Test a relatively heavy load of clients and requests	Pass, the server started having trouble handling the load

Test 1, 2, 3

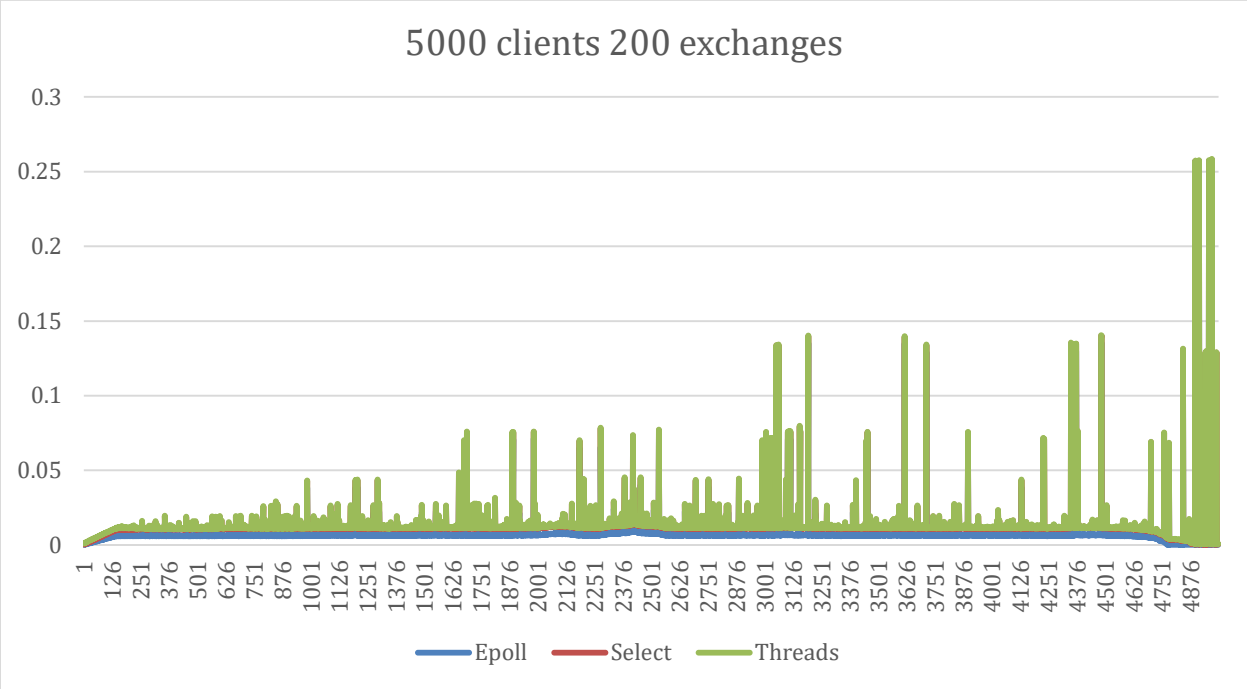


Figure 1 Client Response time Multi-threaded Server

Test 4, 5, 6

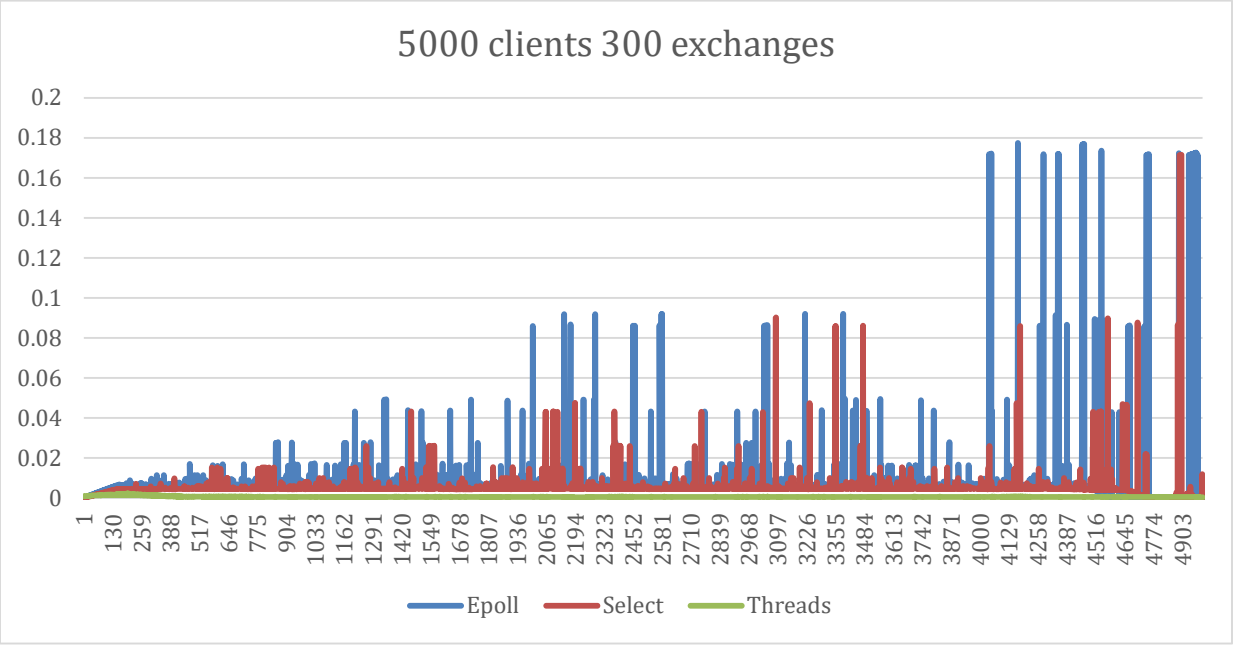


Figure 2 Client Response time Multi-threaded Server

Test 7, 8, 9

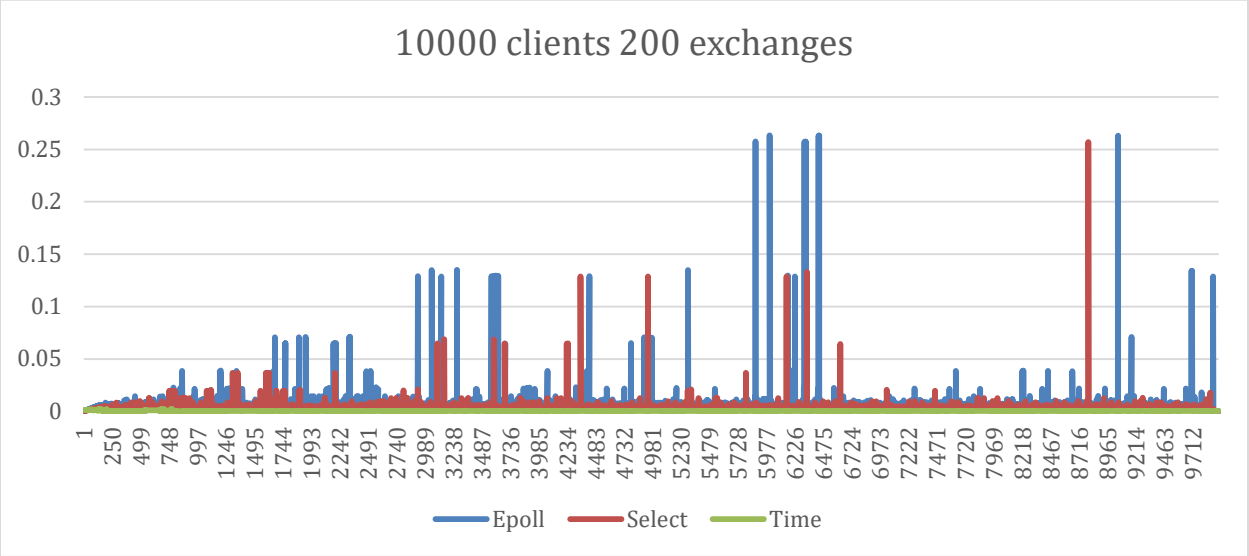


Figure 3 Client Response time Multi-threaded Server

Test 10, 11, 12

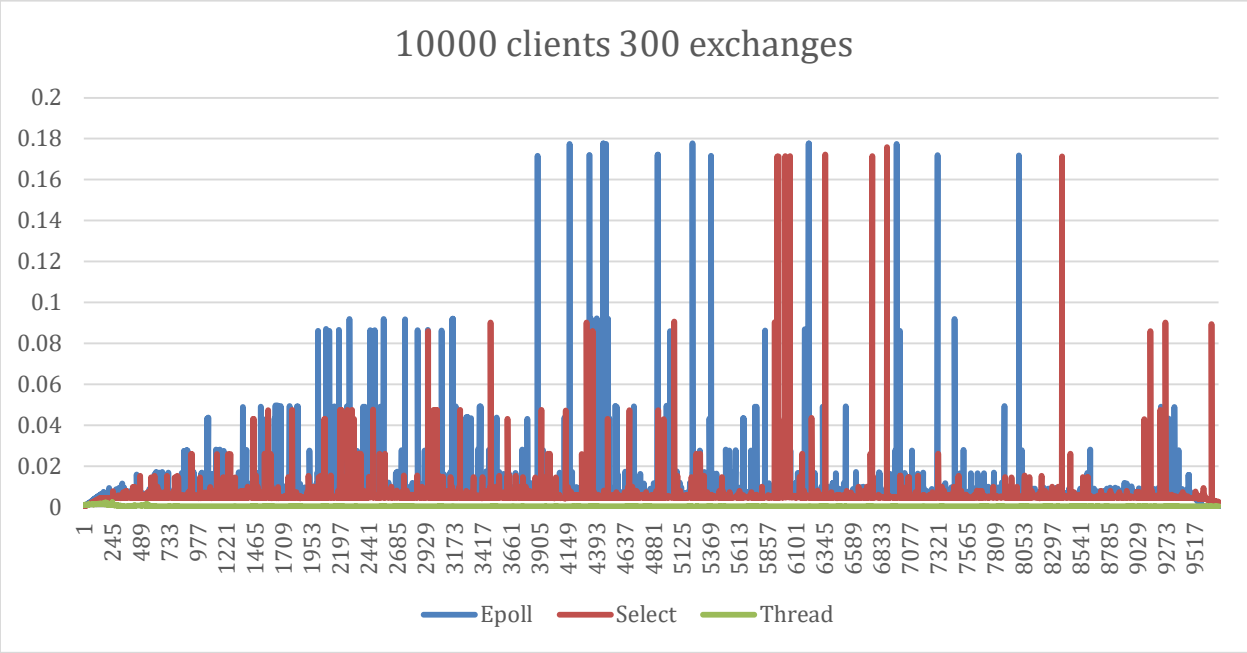


Figure 4 Client Response time Multi-threaded Server

Observations:

	Epoll	Multithread	Select
5000 Clients / 200 Exchanges	0.008002	0.00045	0.00497
5000 Clients / 300 Exchanges	0.008421	0.005281	0.000613
10000 Clients / 200 Exchanges	0.007212	0.000311	0.004897
10000 Clients / 300 Exchanges	0.007262	0.000453	0.005218

The three processes with multi-threading capabilities performed better than the single-threaded ones, and actually used the 4 cores of the CPU, in comparison with the single-threaded that barely used them.

The best performer in this tests was the multi-threaded server, which had response time lower than the other two threads and never in any other test had a problem handling multiple connections for extended time. The worst performance of this server came in the shortest test when it was most volatile.

Epoll even though it had some connections timeouts, these consisted in less than 1% loss most of the time and they happened when the connections lasted a long time. They were probably caused by a mishandling in the implementation and can be improved to have a heavier workloads. Otherwise it works pretty well and has the most consistent times between all the three processes.

Select caused the most trouble as single-process, but adding multiprocessing let the select process work a lot better than expected. It worked mostly in a scaled manner with the more connections and exchanges, the more it took.

Multithreading and Multiprocessing really make a difference when handling a numerous number of clients. From 3000 clients handled by a single process server, the multi-threading server can handle about 5000-10000 clients now.

At the end, there are many factors that influence the performance of the test in addition to the size of data sent/received and number of clients. The traffic on a Network, the processing capabilities of the server to handle the clients, how the network is implemented, and the language used as some functions like epoll don't work as well in some languages.

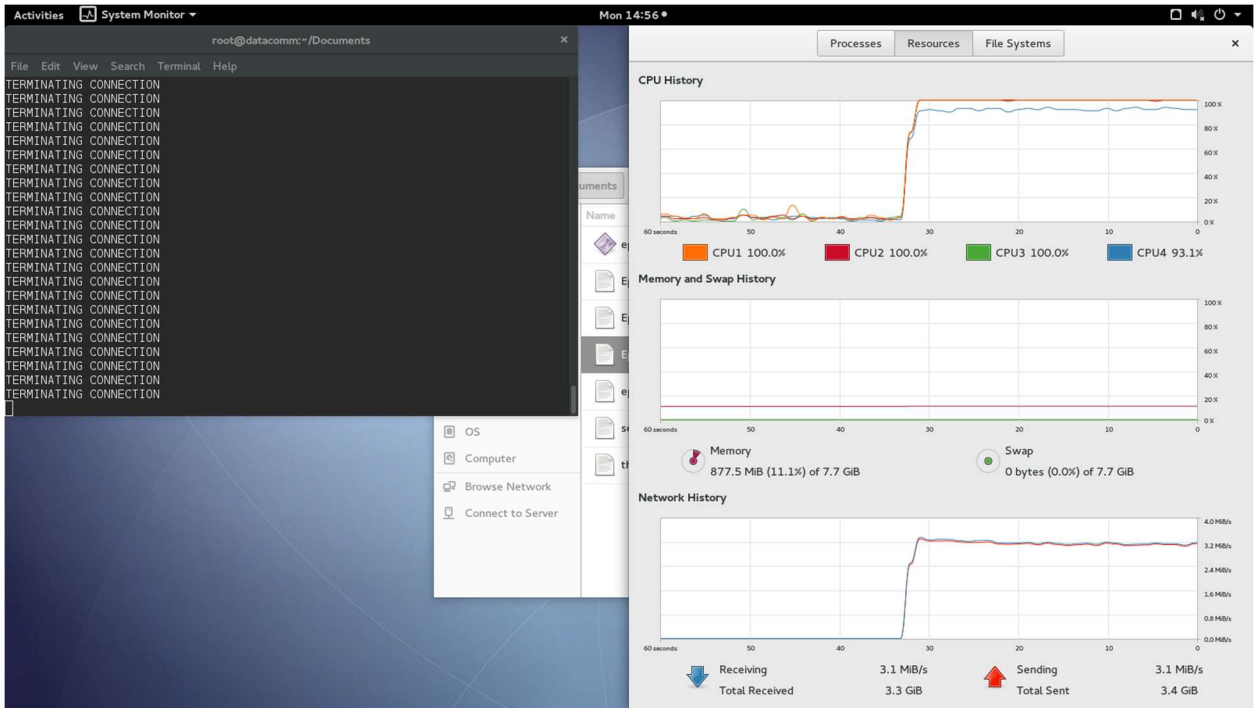


Figure 5 Epoll

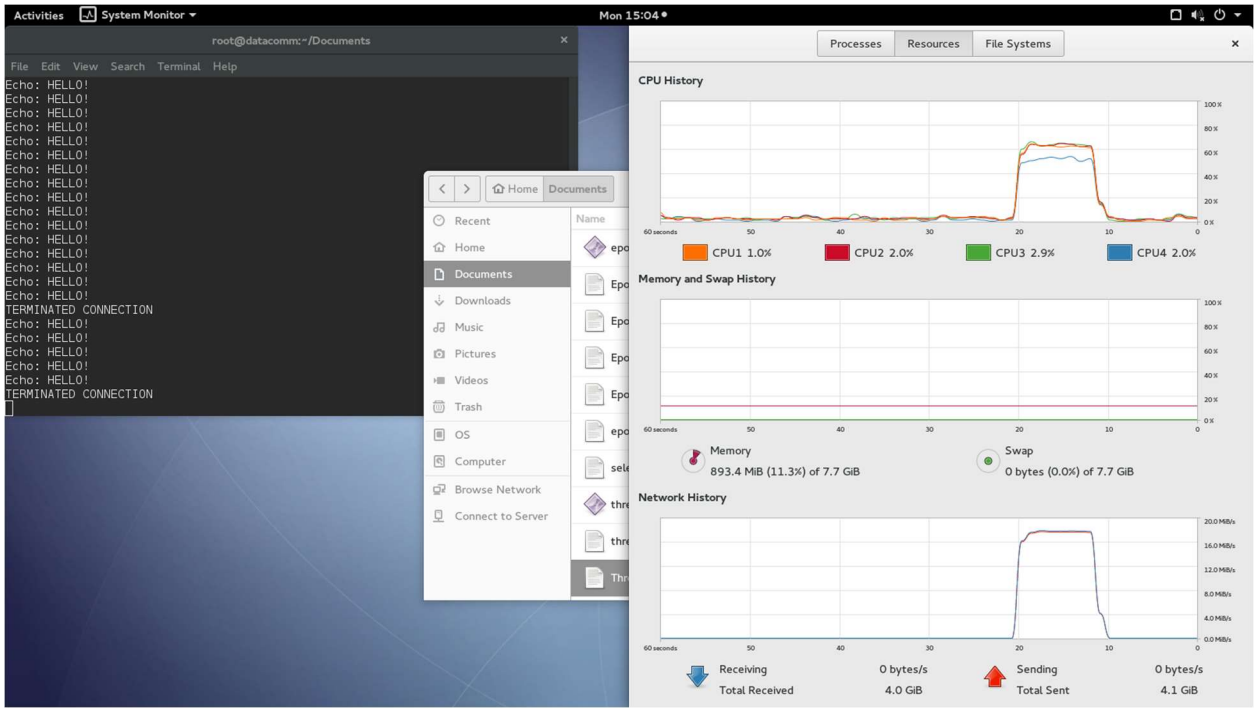


Figure 6 Multi-Thread

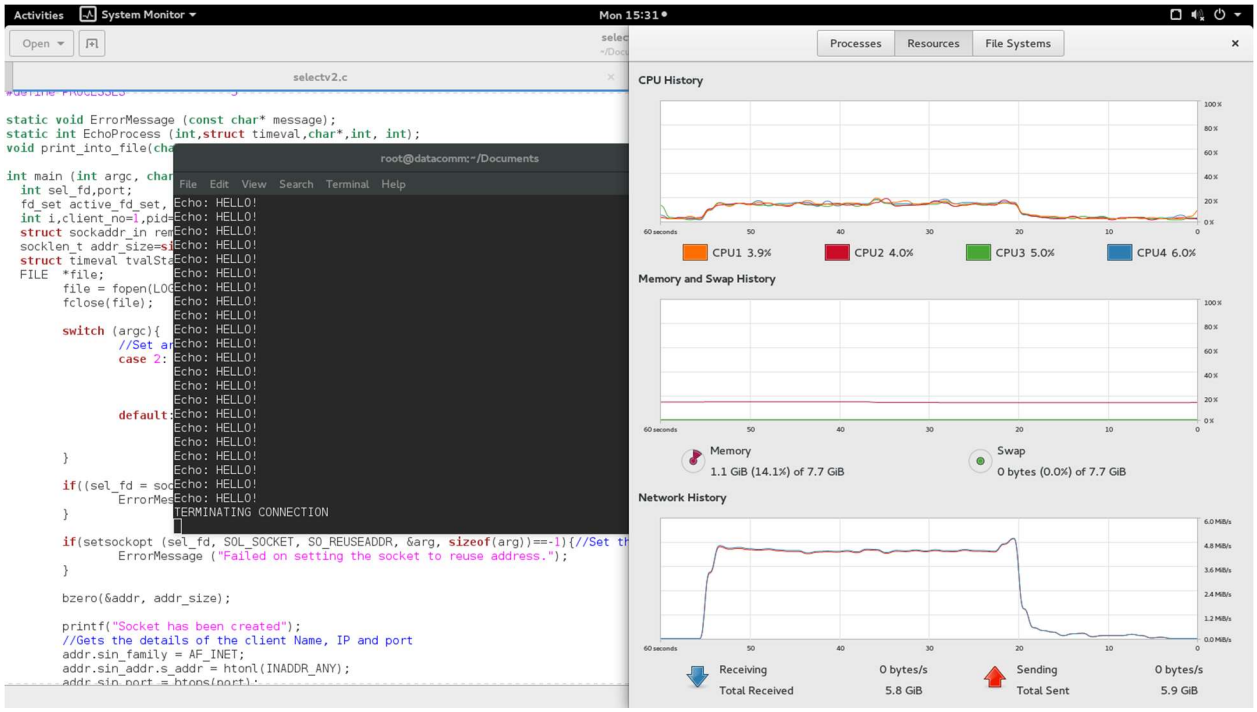


Figure 7 Select non multi-threaded

Pseudo code:

threaded.c

```
Include standard library header files;
Define Variables

Call Function

int main
{

        Initialize variables;
        Initialize structures;

        Create socket;

        Set address family
        Set port number
        Set ip address

        Bind socket to address, port number and ip address
        Listening incoming connections
        Create pthreads

        while (client socket still accepting) {
        Join thread so it does not detach
        }

}


EchoProcess

{
        Initialize variables

        While (flag is still 1) {

                                read the data from client
                                        if (flag is 1) {
                                                        send data to client
                                                }
                                        else            {
                                                        terminate connection
                                                }
                                flush data
                                }
        Close socket
}
```

select.c

```
Include standard library header files;
Define Variables

Call Function

read from client {
```

```
Initialize variables

        While (flag is still 1) {

                                read the data from client
                                        if (flag is 1) {
                                                send data to client
                                                }
                                        else    {
                                                terminate connection
                                                }
                                flush data
                                }
                Close socket

}


int main {
        Initialize variables
        Initialize structures

        if file descriptor of socket = -1 {
        return 1
        }

        Set all values in a buffer to zero


        Set address family
        Set port number
        Set ip address

        Binds socket to an address of current host and port number
        If the process is multi process, fork
        Initialize set of active sockets
        while (flag is still 1){
                Block until input arrives on one or more active sockets

}
        if (i value equals to the socket value) {
                listen for connections and accept
                set bit for file descriptor
        else {
                Close socket
                Clear bit for file descriptor in the file descriptor set
        }


}
```

epoll.c

```
Include standard library header files;
Define Variables
```

```
Call functions

int main
{
        Initialize variables
        Initialize structures

        Port being used

        if (signal = -1) {
                Failed to set interruption signal
        }

        if (socket = -1) {
                Failed to create socket
        }

        Get port to be reused after exit
        (if socket level = -1) {
                Port cannot be reused
        }

        Set to non blocking

        Set address family
        Set port number
        Set ip address

        Binds socket to an address of current host and port number

        If bind, listen are different that -1 do a fork if it's multiprocess, after if epoll queue is -1, cannot
be set

        Add descriptors to epoll

        while(1) {
                Wait for input/output on epoll file descriptor
                for (i less than epoll wait) {
                        if (error) {
                                close
                        }
                        Receiving connection request
                        Set to non blocking
                        Events containing read, write, stream socket close connection and edge
triggered behavior
                        Print out address and port number

                        If socket has read data
                        Close
                }

        }
        Close connection
        Exit
}
```

```
set_non_blocking
{
        if file descriptor = -1
        Print error message
}

read from client {

Initialize variables

        While (flag is still 1) {

                                read the data from client
                                        if (flag is 1) {
                                                send data to client
                                                }
                                        else    {
                                                terminate connection
                                                }
                                flush data
                                }
                Close socket

}
ErrorMessage
{
        Prints error message
        Exit program
}

close_connection
{
        Close connection
        Exit
}
```