Scalable Servers Experiment

Manuel Gonzales

COMP 6D

British Columbia Institute of Technology

COMP 8005

Aman Abdulla

February 15, 2016

Contents

[Summary 3](#_Toc443298485)

[Introduction 4](#_Toc443298486)

[Theory Behind the Experiment 4](#_Toc443298487)

[Experimental Apparatus and Procedure 5](#_Toc443298488)

[The client 5](#_Toc443298489)

[The servers 5](#_Toc443298490)

[Results and Discussion 6](#_Toc443298491)

[Threaded Server 8](#_Toc443298492)

[Select Server 9](#_Toc443298493)

[Epoll Server 10](#_Toc443298494)

[Client 12](#_Toc443298495)

[Conclusion 14](#_Toc443298496)

# Summary

A short experiment was conducted to compare the speed and efficiency of different types of servers in order to find which one is better. The tests were run under 2 different conditions, one where clients would send 1 requests and other with 10 requests.

The process of testing was successful and it helped clear many of the hypothesis points and it showed better performance on **epoll** and **select** servers; nevertheless, the results weren’t as clear since even though some servers were showing better serving times they were still serving the same number of clients overall.

# Introduction

There are several ways of implementing threads such as creating threads / processes for each client in order to serve them faster, or use some more complex calls that will ensure better resource use such as select or epoll. Nevertheless, which one is best for a specific situation? The following report will focus on testing all server and check which one performs better under which conditions.

The results are expected to show which of these is better overall and therefore should be used over the other to increase the server’s performance.

## Theory Behind the Experiment

Threaded servers are fast enough for a limited client of numbers; nevertheless, proven to fail when the load is too much and the server’s hardware can’t keep with it. As for select and epoll server, these calls are better when dealing with limited resources for the server. Following this thinking with the same amount of resources select and epoll, and specially epoll will outperform the others and serve more connections efficiently.

Before the experiment we expect this to be the case, for epoll to be the fastest followed by select and finally threads.

# Experimental Apparatus and Procedure

In order to compare the performance of the different servers, 3 types of servers were created, one that is multithreaded, another one using select, and finally one using epoll. All of these servers were run on the same environment and serving the same amount of clients. All Tested in a Wireless LAN connection. Each server was tested under 2 different conditions, one where there is a burst of clients echoing 1request only and another one where the requestswere10.

The client.- used for testing was running 2000 processes and 250 threads on each process. Each thread would be running a client making for a total of 50000 “concurrent” clients. As mentioned before clients were run with 1 and 10 requests and would send the payload data of a set of strings e.g. “Hello”.

The servers.- each server has a default value for the number of processes / threads to use; nevertheless, in case these are not enough it will keep growing dynamically on demand.

Threaded server.- initially runs 200 threads and will spawn a new one as soon as one of the existing threads accept a new connection. Threads return after serving the client.

Select server.- initially runs 50 processes and will stop one process and create a new one if the select call returns many errors, in timeout it will clear all sockets. In case of all the processes getting full, a new process would be created.

Select server.- initially runs 50 processes and will stop one process and create a new one if the epoll call returns many errors, in timeout it will clear all sockets. In case of all the processes getting full, a new process would be created.

# Results and Discussion

The full data for these approximations can be found under the **csv folder**  where all the data from connections was stored for each of the tests.

When the test were run with only 1 request per client the results were really different as shown in Figures 1 and 2, The threaded server took a lot of time in average for serving all 400,000 clients compared to select and epoll. As for epoll it showed an impressive timing being close to 2 times faster than the threaded one.

Most of the tests started with really small numbers for serving time on all of the servers (as shown on the csv files); however, over time it went increasing thus generating a higher average time for the duration of the tests.

**Note:** It is also worth noticing that the client side of the tests show better service times than the server. This happens because the client will stop the timer as soon as it closes the connection to the server. While on the other hand, the server still needs to wait for that FIN/RESET in order to stop the timer and close the connection.

As for the tests run with 10 requests the results for all 3 were really similar as shown in figures 3 and 4. They all seemed to perform at the same pace overall and served all 100,00 clients. Epoll and select start with really good times compares to threads early on; nevertheless, once the number current clients start peaking the service time increases dramatically and leads to an average time of 1.8 secs per request instead of the faster results shown by the 1 request test.

Average Served Clients by servers with 1 Request : 400,000 in 120000 msec sample.

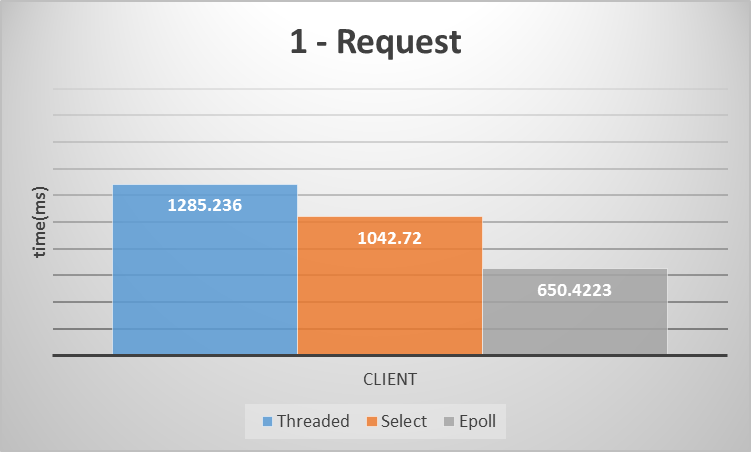


Fig. 1 service time (according to client)

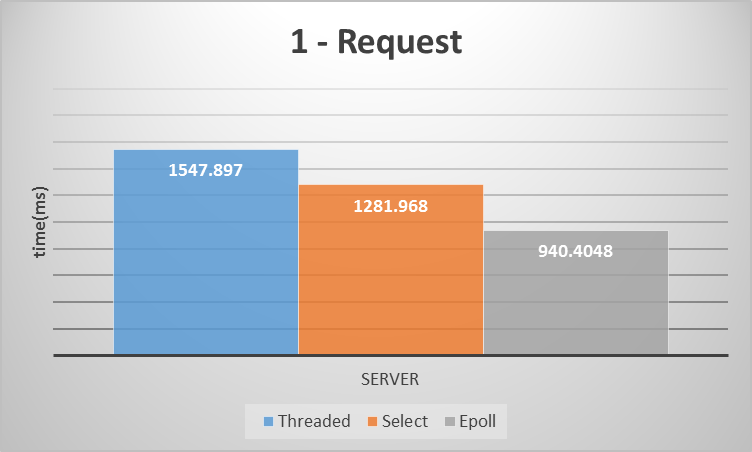


Fig.2 service time (according to server)

Average Served Clients by servers with 10 Request : 100,000 in 120000 msec sample

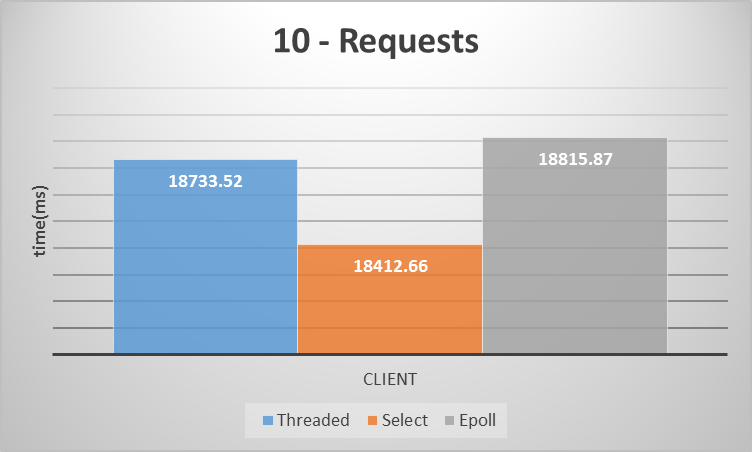


Fig.3 service time (according to client)

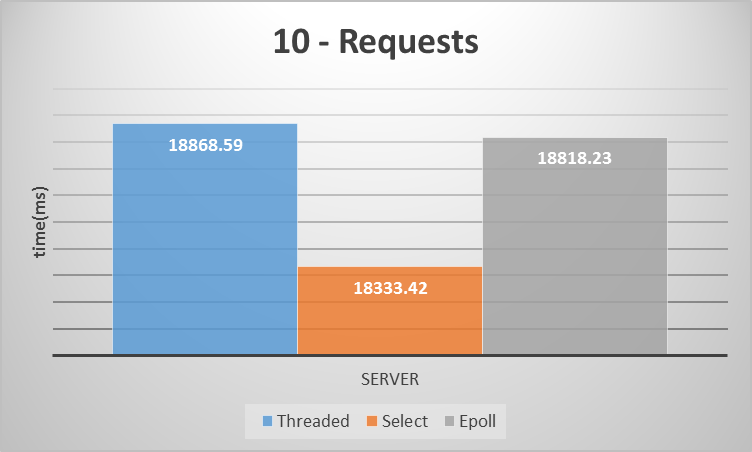


Fig.4 service time (according to client)

## Threaded Server

**It will stop if it reaches maximum file descriptors.** Some stats for the threaded servers are:

Average Memory Use: 18% (shown in test document)

Average Network Transfer:1.5 MiB/sec for Read/Receive(shown in test Document)

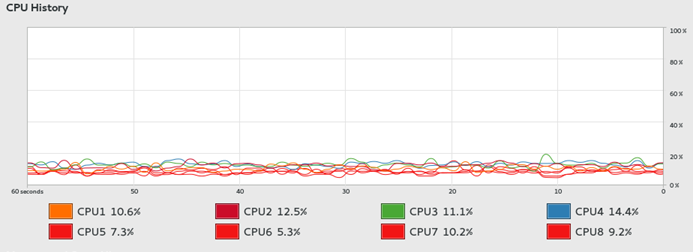


Fig.5 CPU Usage (1 - request)

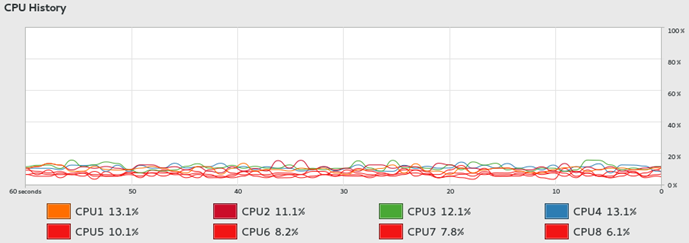


Fig.6 CPU Usage (10 - request)

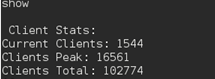


Fig.7 Peak Clients (10 - request)

## Select Server

Some stats for the select servers are:

Average Memory Use: 13% (shown in test document)

Average Network Transfer:1.5 MiB/sec for Read/Receive(shown in test Document)

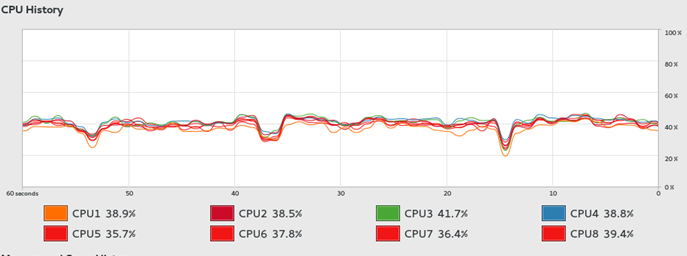


Fig.8 CPU Usage (1 - request)

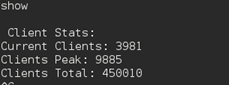


Fig.9 Peak Clients (1 - request)

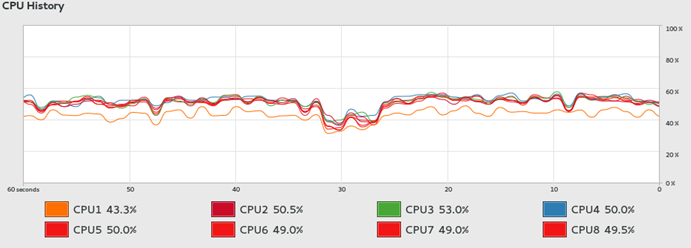


Fig.10 CPU Usage (10 - request)

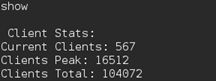


Fig.11 Peak Clients (10 - request)

## Epoll Server

Some stats for the threaded servers are like this:

Average Memory Use: 12% (shown in test document)

Average Network Transfer:1.5 MiB/sec for Read/Receive(shown in test Document)

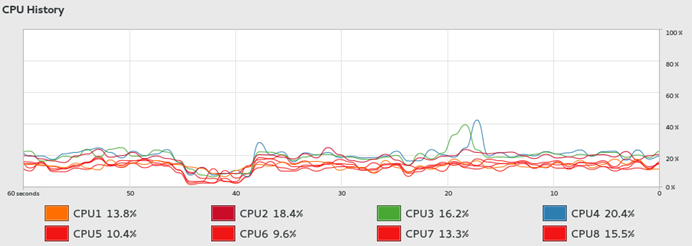


Fig.12 CPU Usage (1 - request)

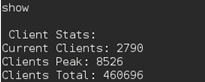


Fig.13 Peak Clients (1 - request)

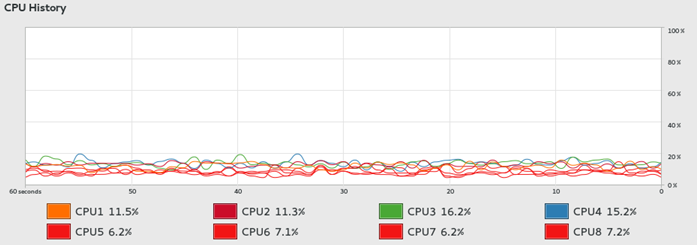


Fig.14 CPU Usage (10 - request)

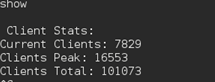


Fig.15Peak Clients (10 - request)

**Note:**

As a note after the demo of the servers and raising some questions about the data printed during the tests.

**Current** **Clients**: Will show the current clients being server at that given time. The count increases only when a new connection has been accepted successfully, and only decreases if the client has been served and the connection is closed, or if there is an error/timeout and the client socket is closed.

**Clients Peak**: It will show the highest number of concurrent clients so far during the test. It uses current clients data and gets replaced when the current clients surpasses the last client peak. This comparison only happens after accepting a new connection successfully and will ensure a right number.

**Clients Total**: It will show the number of clients served in total during the duration of the test. The count increases only when a new connection has been accepted successful just as the current count, but does not decrement at all during the test/session.

This data has the same principle on the client, on connect increment current and total, and on closing decrement current. Nevertheless, clients have a little less control of mutex and semaphores so data can get a little corrupted, but still really approximate to the real count.

## Client

Average Network Transfer: 1.5 MiB/sec for Read/Receive (shown in test Document)

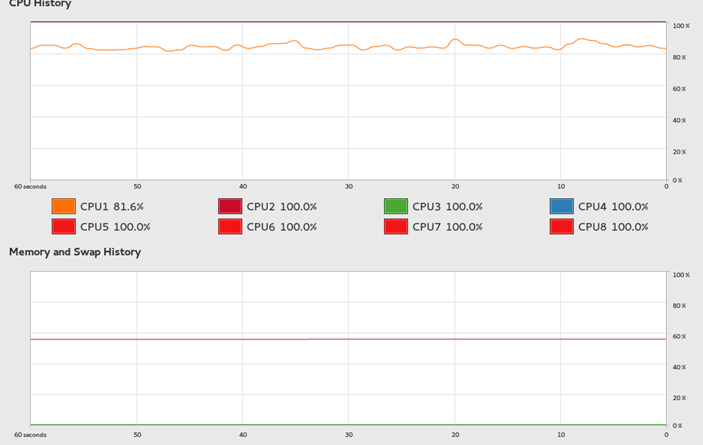


Fig.5 Resources Usage (1 - request)

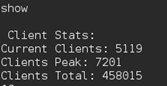


Fig.5 Peak Clients (1 - request)

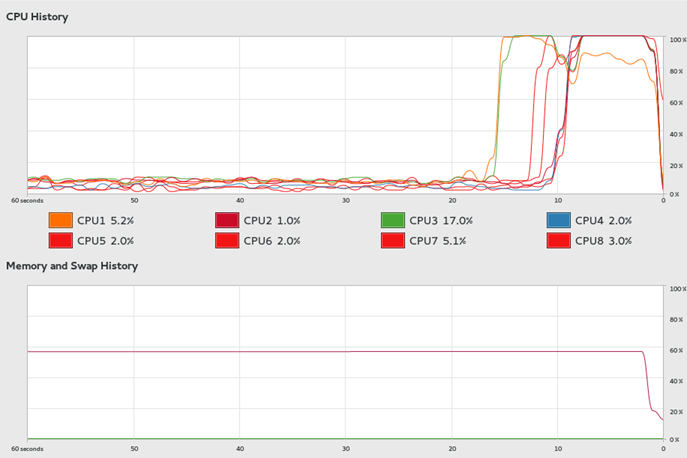


Fig.5 Resources Usage (10 - request)

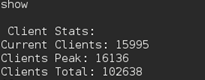


Fig.5 Peak Clients (10 - request)

**Note:** Even though the client was expected to keep 50,000 concurrent connections (current / peak) it wasn’t able to because the servers were serving the clients really fast and would have taken a longer time in order to actually start overwhelming the servers and reaching the expected number.

# Conclusion

From this short experiment after reviewing the results, we would have to agree that in fact epoll calls and select calls are showing better serving times than threads; nevertheless, it seems to only be showing those results when serving one request per client. As per 10 requests, the average serving time becomes really close in all of them which probably suggest a throttle from the client side to the servers.

Threaded servers seemed to use the most resources memory wise due to the number of threads it would keep spawning in order to serve new connections; on the other hand select and epoll servers showed little to no use from the memory.

Regarding CPU usage the threaded and epoll calls seem to use less resources than select and this is probably because of select being level-triggered and requiring more switching.

To conclude, Threads can be used when the number of clients is going to be small and resources are sufficient to serve all efficiently, and select calls can be used when the number of clients is significant and enough resources are available to serve the clients. Nevertheless, epoll servers are the best option of the three since the use of resources is less compared to the others and it’s proven to server clients as fast as twice compared to other types of servers. And on top of all the benefits the epoll implementation doesn’t need to be as complex to be efficient. Time to make the switch into epoll servers.