DV1619 Assignment 4

Simon Strand
Blekinge Institute of Technology
Karlskrona, Sweden
etta11@hotmail.se

1. Introduction

This report is about how a threaded web server compares to a forked web server in a local environment.
The report will also take up if there is any difference between creating a new thread every call or having a thread pool.

2. Setup

The tests were done on a windows computer running a virtual machine (Oracle VM) with the operative system Ubuntu 20.04.5.
One test was done with Paravirtualization Interface set as “Default” and one with “None”.
The computer has an i7-10700k CPU with a clock speed of 3.79GHz, 2 ddr4 ram sticks with 16 GB storage (total) (the ram is placed in slots 1 and 3 so duel channel is utilized), and a speed of 3200 MHz, one NV2 Gen 4 M.2 SSD with 1 TB capacity (3500 MB/s reading speed and 2100 MB/s writing speed).
The virtual machine used 8 out of 16 processors from the CPU, 8000 MB of the ram memory, and 52.5 GB of the M.2 SSD.
Both client, threaded server and forked server ran on the same computer and talked via ipv4 and the TCP protocol.
The code was compiled with clang version 10.0.
No other program was running on the virtual machine except visual studio code.
One other test was done using a laptop that I got to borrow for 2 hours running the operative system Manjaro.
This laptop had 8 cores and more specifications was not able to be gathered.

3. Method

Multiple tests were conducted using a shell script created by Patrik Arlos[1] but were modified to get more values.
Specifically, a timer was added for both the thread and fork tests and the results are displayed/printed upon completion.

The script worked as such that a tool called “Apache Benchmark tool” would call 10000 requests for a file called “big” to the server.
These requests would be called first 10 times with a concurrency off one, and then 10 more times with a concurrency off two and so on until a concurrency of five have been met.
The values were also tested with a repeat of 32 times and concurrency up to 60.

There is one version of the forked server and two versions of the threaded server.

Forked Server:
When the forked server gets a client a forked process will be created, the parent will close the socket and continue trying to accept new clients.
If a fork could not be created a function will wait until all forked processes are done and remove zombie processes and then try to create a fork process again.
The forked process will start handling the user and after a user has handled a shutdown will be forced and the socket will be closed.

```
void WaitForFork(){
    int status = 0;
    pid_t wpid;
    while ((wpid = wait(&status)) > 0);
}
```

Figure. 1. function for cleaning up child fork processes and zombie processes.

Threaded server
There are two versions of the threaded server.
In the first version, a function would check if a thread is able to be created.
If yes a client can be accepted.
When a client is accepted a thread would be created and run a function with parameters that handle the client.
When the thread is done a reference to the thread is set to done and the main program will join the thread.
The number of threads able to be created is capped based on the CPU.
The second version uses pool threading where x number of threads are created, again based on the CPU.
The threads run a looping function where the threads check if any jobs are available.
When a client joins the thread pool is given a job with parameters and one thread will pick up the job and handle the client.

```
ThreadPool::ThreadPool(){
    nrOfThreads = std::thread::hardware_concurrency() + 2;
    threads.resize(nrOfThreads);
    runningThreads.resize(nrOfThreads, 0);
    std::cout << "nr of Threads:" << nrOfThreads << std::endl;
}
```

Figure. 2. function to decide the max number of threads in both versions of the threaded server.

Handle Clients

All the versions uses the same handle client code, both threaded server versions and the fork server.
The socket gets a timeout timer for 3 seconds both for sending and receiving.
The server starts by trying to receive data from the client.
If nothing is received a warning message will be shown and kick the client.
If the server got something it will convert it to a string and look for a “GET” call in the string.
The server will check for how many slashes there are in the “GET” call, if there are more than 3 an error will be returned to the client and kicked.
The server tries to get the HTTP version and file name.
The server accepts HTTP/1.1 and HTTP/1.0.
The server will read 5000 bytes of the file at a time and send that data to the client.
When the server is done reading the data the server will kick the client.

4. Results

The two versions of the threaded server were compared, and no significant difference was observed. Therefore, only the thread pool version is presented in the results.
Also no significant difference in speed between bigger and smaller returning documents where found.
Figure 3 shows the performance with paravirtualization set to “Default” where the threaded server is quicker than the forked server when the concurrency is below three. When the concurrency is between three and seven they are relative equally but after that, the forked server is faster and the threaded server starts to lose speed.
More details on why this is the case are discussed in the “Conclusion and Discussions” section.
The test took on average 34 minutes and 4 seconds to complete using the threaded server and 20 minutes and 40 seconds using the forked server.
Performance with paravirtualization set to “None” is illustrated in Figure 4. Similar to the results with paravirtualization set to “Default”, the threaded server outperformed the forked server when the concurrency is below 4, but the forked server outperformed the threaded server when the concurrency is higher. However, both servers perform slower than when paravirtualization was set to “Default”. The threaded server completed the test in 42 minutes and 16 seconds, while the forked server finished in 40 minutes and 48 seconds.
Figure 5 shows the performance on the borrowed laptop where the threaded server is faster than the forked server at all concurrencies.
The threaded server took 31 minutes and 3 seconds to complete the test and the forked server took 63 minutes and 48 seconds.

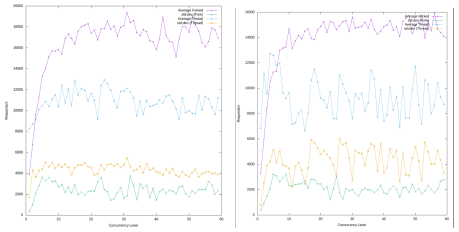

Figure. 3. statistics from thread vs fork test when the paravirtualization is set to “Default”, purple line is the forked server and blue is the threaded server.

Figure. 4. statistics from thread vs fork test when the paravirtualization is set to “None”, purple line is the forked server and blue is the threaded server.

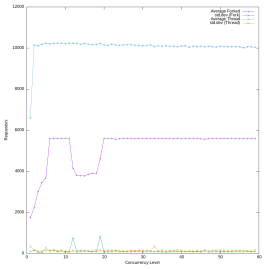

Figure. 5. statistics from thread vs fork test, using the borrowed laptop, purple line is the forked server and blue is the threaded server..

5. Conclusion and Discussions

On the computer that ran the virtual machine, the data appears to favor the forked server when multiple connections are connected, while the threaded server seems to be faster when only one to two clients are connected. While running both servers on the virtual machine a discovery was made, the threaded server would not utilize all available processing power, instead, a core would be staying at 50% and resulting in a slower performance (see figure 6). The forked server, on the other hand, runs as expected at 100% without this issue.
Making a stress test program that is trying to stress the virtual machine and make all cores utilize 100% power using threads was unsuccessful. While with the same stress test program used on windows, all the cores used 100% of the processing power available.
Borrowing a laptop running Linux Manjaro the threaded server both ran faster than the forked

server and made the cores on the threaded server run at 100% confirming the theory.
The reason for the performance difference is unclear.
So why would a threaded server or forked server be faster?
A threaded instance takes a shorter period of time to create than a forked instance, where a forked instance copy's all the data from the parent process to the child process while the threading, all threads share the same memory as the main thread is using. The speed between threads and forks should not be significantly faster than the other proven by Yanis MANSOUR in his article about threads vs forks[3] but the creation is.

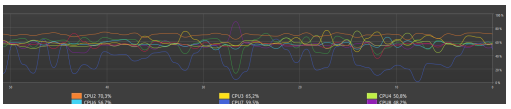

Figure. 6. System Monitor when stress testing on the virtual machine

6. References

[1] Patrik Arlos, *np_assignment4_web/dcollect.sh*, https://github.com/patrikarlos/np_assignment4_web/blob/master/dcollect.sh.
[2] *Forking vs Threading* https://stackoverflow.com/questions/16354460/forking-vs-threading. accessed 2023-02-21.
[3] Threads vs forks in C++ - Speed test (Linux) https://www.yanismansour.com/articles/20210723-Threads-VS-Forks-C++. acceded 2023-02-27