CN Assignment 2

Madhava Krishna
2020217

The client programs can be found in Client_Programs and the server programs in Server_Programs.
Compilation Instructions: *make* in the main directory will compile all the required files in their original C file directory.
Running instructions: Parallel client implementation can be found in 'Parallel client' and sequential in 'Sequential client' directory. For servers, 'Concurrent server', 'Non-blocking server' and 'Sequential server' contain the (fork and thread), (select, poll, epoll), and (sequential) implementations respectively.

Reference was taken from the Linux Manpages for select, poll and epoll implementations.
Note: instead of having 10 clients, I implemented 20 clients for the final submission.

Q1. Clients use a write_to_server program which takes in the socket fd and uses the write syscall. In addition to this, the parallel client program uses a main_function for threads to execute.

Q2. Each program uses the following functions (with minor adjustments):

- factorial : trivially calculates factorial
- read_write_to_client: for reading and writing, reads from the fd and writes to a FILE* using fprintf.
- serv_functions: for performing server functions (mostly for pthread and Non-blocking server approaches for its simplicity).

In addition, various *struct*s have also been created to facilitate threading and ease of operation. Most notable of which is *thread_data*, which takes in everything it needs to print to a file (socket fd, client sockaddr_in structure and file descriptor for writing).
There are parameters which can be used to tune the max number of connections (QUEUE for max number of connections, LIMIT for maximum number of factorials to process).
Error handling has been done using if-else statements and exits.
Outputs are taken to a CSV file in the root directory.

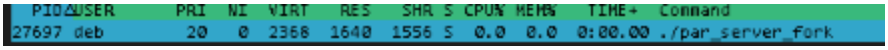
Q3.

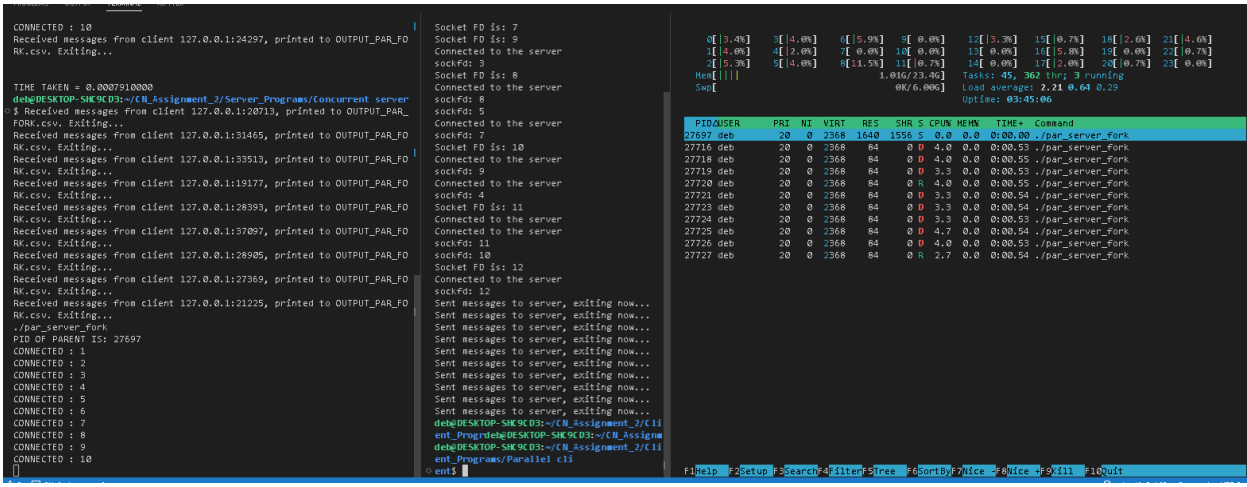
a) **Concurrent Server using Fork:**

i)

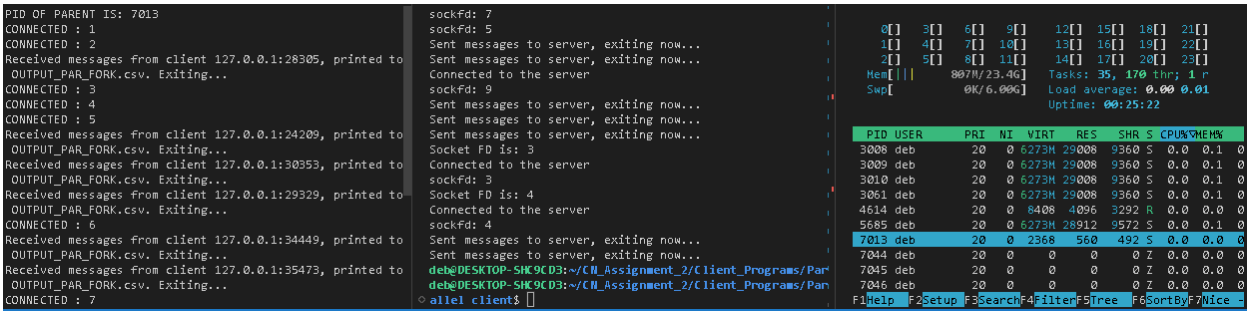
```
TIME TAKEN = 0.0005860000
Received messages from client 127.0.0.1:62701, printed to OUTPUT_PAR_FOR
K.csv. Exiting...
Received messages from client 127.0.0.1:1262, printed to OUTPUT_PAR_FORK
.csv. Exiting...
Received messages from client 127.0.0.1:4846, printed to OUTPUT_PAR_FORK
.csv. Exiting...
```

ii)



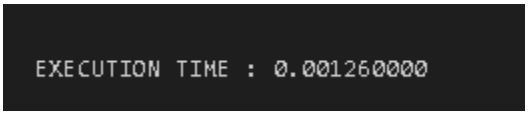


The execution time for the main server was 0.0005 seconds calculated using the clock() function. This means that only 0.0005 seconds of CPU time was utilized by the server. It was not possible to monitor the memory and CPU usage using htop reliably. I increased the number of values for which factorial was to be calculated and obtained the above result. The server sat idle most of the times and the forked processes handled each client.



b) Concurrent Server using PThreads:

i)

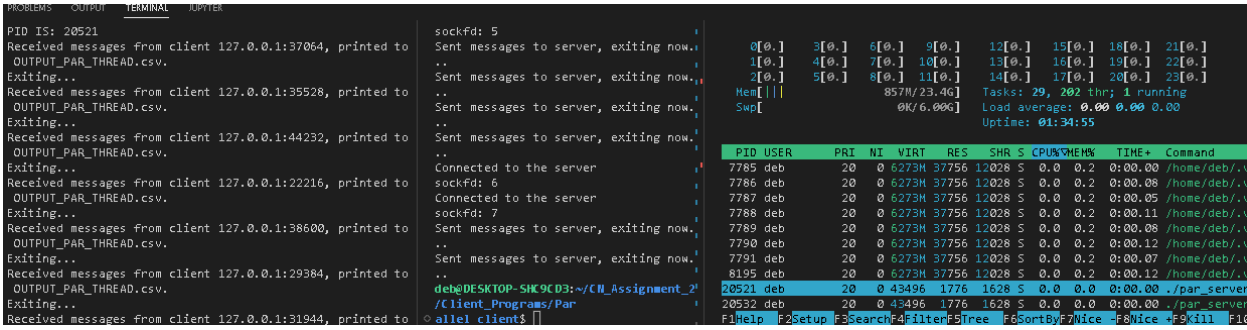


ii)

It uses 43496 KB virtual memory, 1776 KB resident memory and 1628KB shared memory.

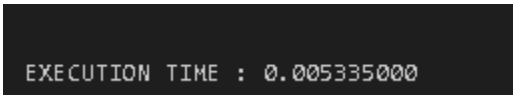
Again, it was not possible to show the CPU utilisations because the program

terminated too quickly. The CPU util was shown to be 0%.

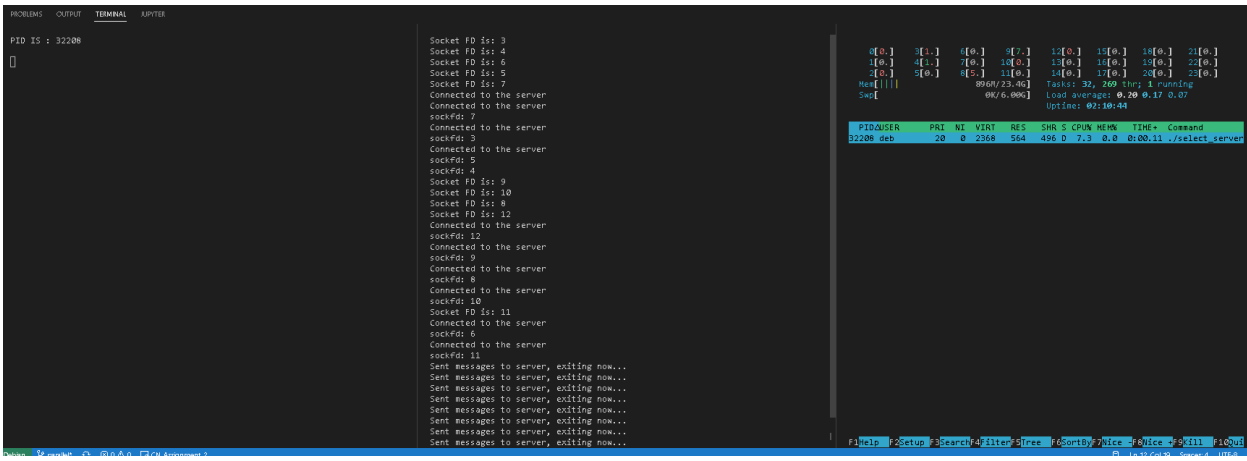


c) Server using Select:

i)



ii)

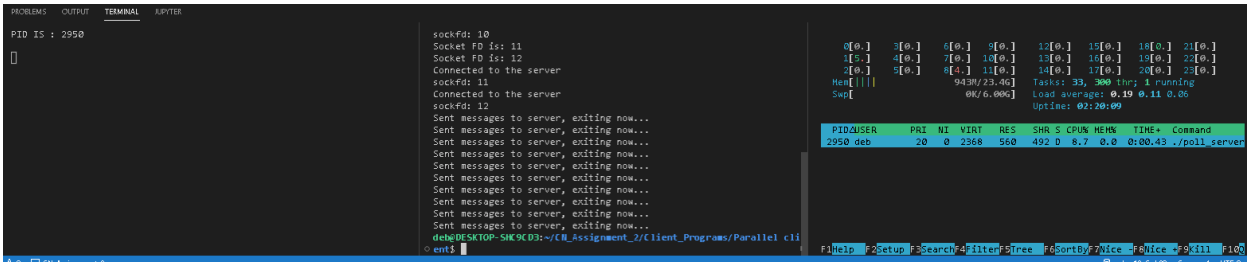


On increasing the number of values sent, it was possible to observe a difference in CPU utilisation. The time taken were significantly increased relative to pthreads and fork implementations.

d) Server using poll:

i)

ii)



```
  PID⏃USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
2950 deb        20   0  2368   560   492 R  8.0  0.0  0:01.48 ./poll_server
```

```
PID IS : 2950
```

e) Server using Epoll

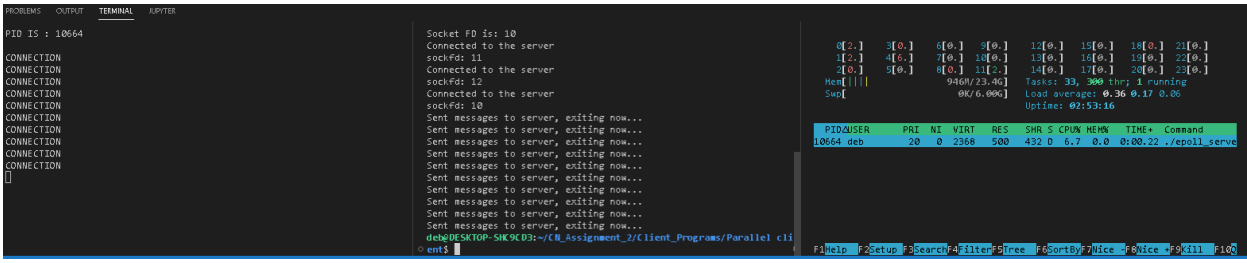
i)

```
EXECUTION TIME : 0.003610000
```

ii)

```
PID IS : 10664
```

```
  PID⏃USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
10664 deb       20   0  2368   500   432 D  6.7  0.0  0:00.22 ./epoll_serve
```



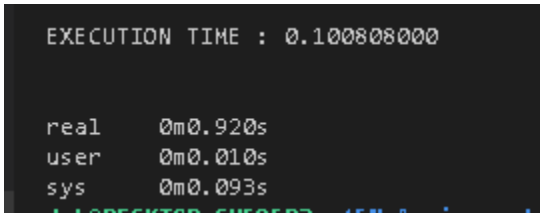
Q4.

With respect to the time required by the server to process all client requests:

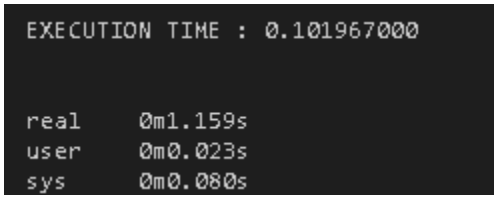
- The server using fork() has the least time because once it gets connected to clients, it goes to idle (in a while(1) loop), waiting for its children to exit. Otherwise, it should take roughly the same time as the pthreads program.
- The pthreads program uses parallelism and executes in 0.001 second. The time is higher than fork() because the CPU utilisation by the process is more (threads are spawned within the same process, fork() creates new processes).
- For the non-blocking counterparts, the execution time is significantly higher. This is because of the program operating on a single thread.
    - Select has the highest time, because of how many redundant operations are being performed. Resetting the fd set and iterating upon them takes significant time.
    - Poll takes less time than select, because there is no need to reset the fd set. It resets the flags on its own.

- Epoll limits the set of FDs, though my implementation may have caused inefficiencies to arise (using a set for mapping client sockaddr_in structure for passing to a function). It is said to be faster than poll() because of the limited number of fds to iterate over.

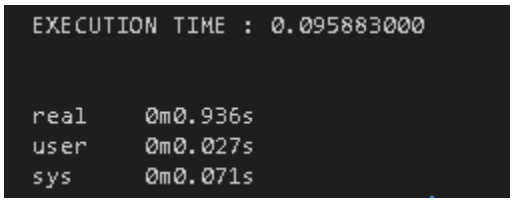
Trying out with 200 clients using ‘time’ when executing



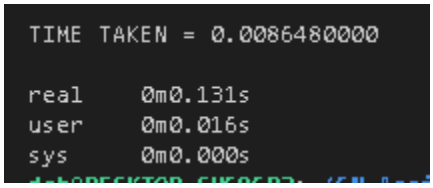
epoll server with 200 clients



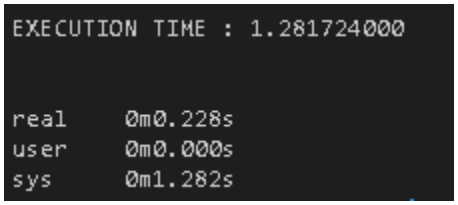
poll server with 200 clients



select with 200 clients



fork with 200 cilents



thread with 200 clients

We notice that pthread takes the least time, but the maximum CPU time (because of parallelism). Fork takes significantly more time, because of creating a whole new process, which is more expensive than threading.
epoll takes about the same time as select, which may be because of my inefficiency (using array and linearly scanning).
poll takes the longest time because of the same reason.