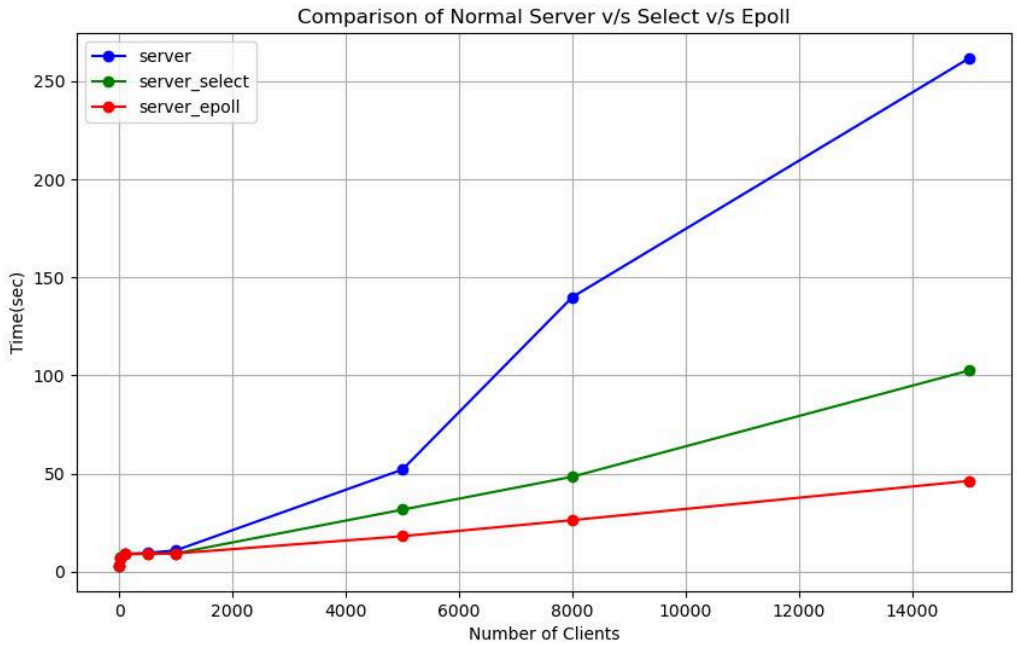
4a. network programming 101

4.)



Server.c

Clients	Time(seconds)
1	3.006905355
10	7.017566757
100	9.113325512
500	9.587915006
1000	10.898518200
5000	52.078291198
8000	139.970608033
15000	261.627769729

Server_Select.c

Clients	Time(seconds)
1	3.004761042
10	7.014678905
100	9.035061600
500	9.178066118
1000	9.368747179
5000	31.644475457
8000	48.450742196
15000	102.561783357

Server_Poll.c

Clients	Time(seconds)
1	3.004995632
10	7.005764809
100	9.030703949
500	9.148651671
1000	9.327743248
5000	18.136480357
8000	26.306689392
15000	46.302540011

As from the above plot and values we can observe that on increasing the number of clients server_epoll perform better as compared to the others.
Epoll is more scalable as compared to the normal server implementation and select.
Server_select performs better than normal server implementation.

5.)

After executing $sudo sysctl net.ipv4.tcp_syn_retries=1

These are the minimum number of connections for which the connect request is refused for at least 1 client due to time out.

The backlog is set to 5.

Due to max number for syn retries being set to 1 the queued clients for whom connect request is not accepted by the server receive no acknowledge for connection client terminate with printing no connection established.

kserver_epoll.c- 1890 clients
kserver_select.c- 1150 clients
kserver.c- 10 clients

6.) kserver_select.c

Client	Time(sec)
1	3.007378816
10	7.010172124
100	9.038260598
500	9.192300517
1000	9.373344831
5000	28.081200957
8000	51.679720661
15000	140.714767911

kserver_epoll.c

Client	Time(sec)
1	3.005080089
10	7.005485333
100	9.032772276
500	9.159126384
1000	9.306237680
5000	18.183914996
8000	28.130934268
15000	46.567079755

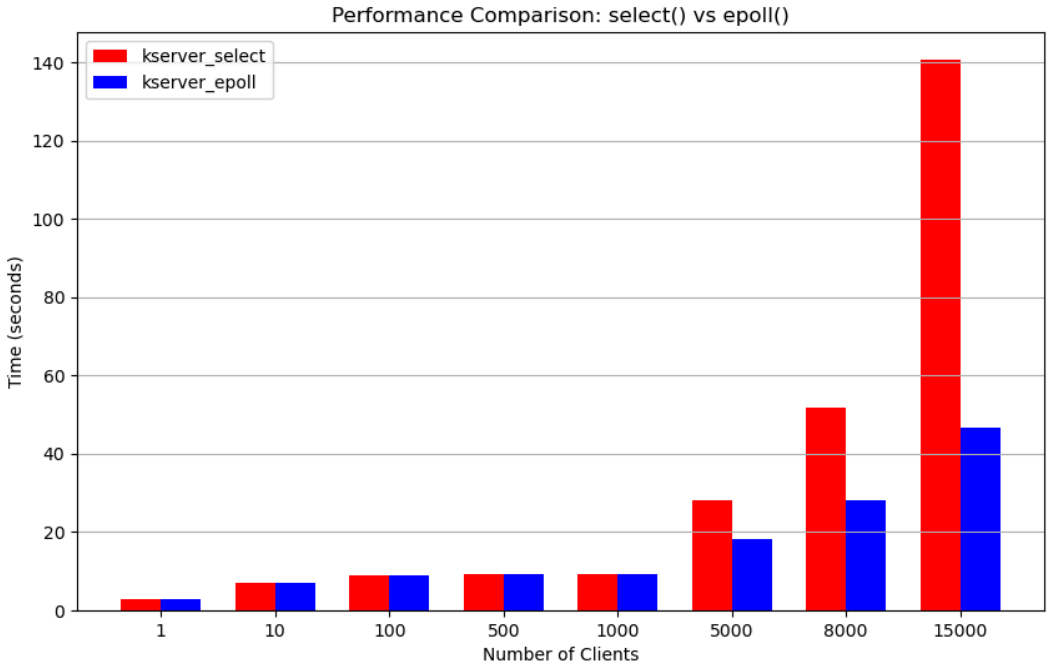
Analysis:----------

`epoll()` clearly scales better than `select()` as the number of clients increases. This is expected since `select()` meaning it needs to iterate over all file descriptors from fd 0 to fdmax-1 to determine which file descriptors are ready leading it to time complexity of O(n) , while `epoll()` is more efficient, especially for a large number of connections, because it uses an event-driven model with O(1) complexity epoll() make an instance in kernel space which keeps tracks of all the ready file descriptors.

For large number of clients—
For select_server,
N=15000,time= 140.714767911
throughput=total_clients/total_time
=15000/140.714767911
=106.598619481697 clients/sec
For epoll_server,
N=15000,time=46.567079755
throughput=15000/46.567079755
=322.1159685966655 clients/sec

Comparing throughputs of both the implementations we can say that for a large number of clients epoll_server is more scalable and has more throughput as compared to select_server.

select_server is having more overheads as compared to epoll as it always iterates over the set to find the fds that are ready for read/write.



Task 4B- 0. Threads are fun!

Output give by processes.c:

PID of Process: 2479651
Value of x: 2
PID of Process: 2479652
Value of x: 7

Output give by threads.c:

PID of Foo: 2479675
PthreadID : 124358469093056
Value of x: 7
PID of Bar: 2479675
PthreadID : 124358458607296
Value of x: 7

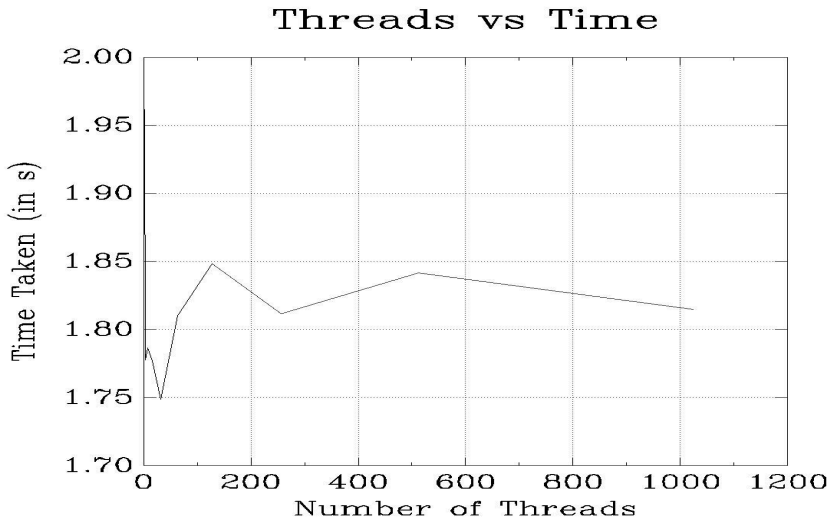
Reason for this:-

1.Since processes have separate memory spaces, changes to variable x are isolated. Each process operates independently of the other in terms of memory.

2.Threads share the same memory space, so changes to x by one thread are reflected in all threads.

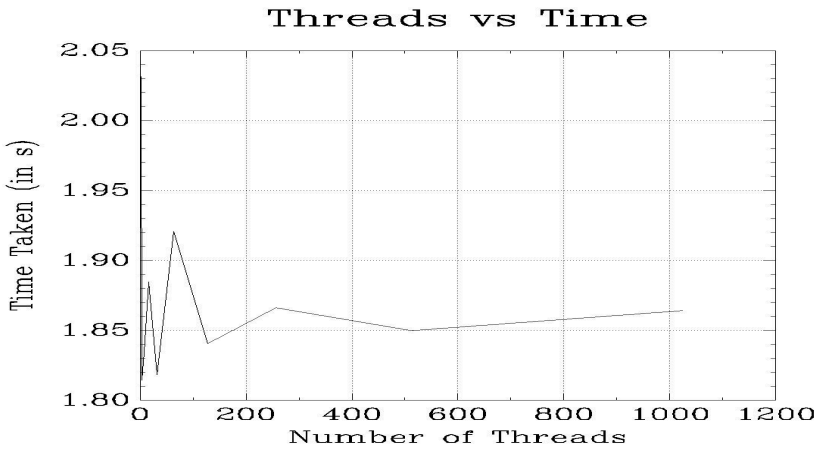
Task 4-1B The Sweet Spot!

So does increasing the number of threads result in better performance?



Client	Time(sec)
2	1.961591
4	1.777826
8	1.786684
16	1.777066
32	1.748611
64	1.810346
128	1.848496
256	1.811514
512	1.841501
1024	1.814753

As from the above values increasing the number of threads here don't improve performance as such due to higher thread counts, the overhead of context switching, synchronization, and resource contention may be overhead to extra time.
Simply increasing the number of threads does not always lead to better performance.



After getting the previous plot on again executing the program we got that increasing threads do decrease the time, but on increasing the number of threads it also increases the execution time.