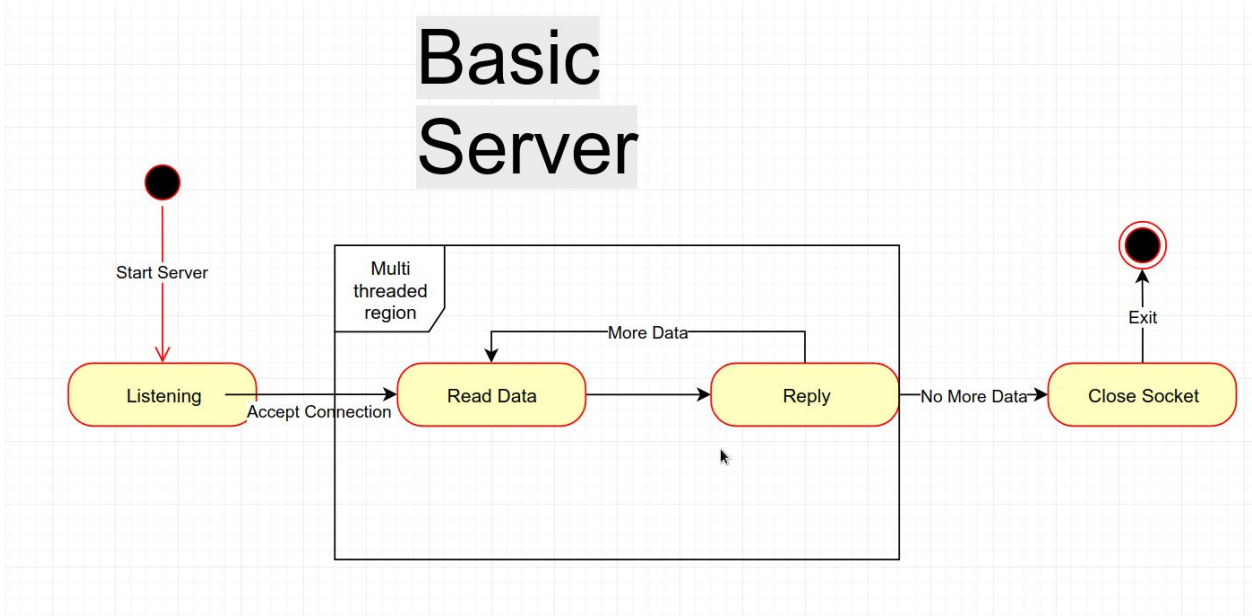
Dimitry Rakhlei
A2 8005 Design

Basic Server:

The basic server was designed to run in a very similar way to the regular echo server we saw in class with a few minor modifications.
In our case the server would log data when connected to, writing data and when terminating connections.

Here is the original FSM used in design.



The threading model used to implement the multi-threaded region was a Thread Pool. The source code for the threadpool can be seen in src/pool. It is taken from the well known repository https://github.com/mbrossard/threadpool.

This model can be improved by using multiple pools to reduce overhead on individual queues.

Used by running *./bin/server basic [buflen]*

After fixing a few bugs and testing I found that the basic server is the fastest of the other two when the number of clients is lower than the total number of cores on the machines. This is obvious because with the basic design each client would get a dedicated core ensuring speed.
On a machine with a 100MB/s NIC:

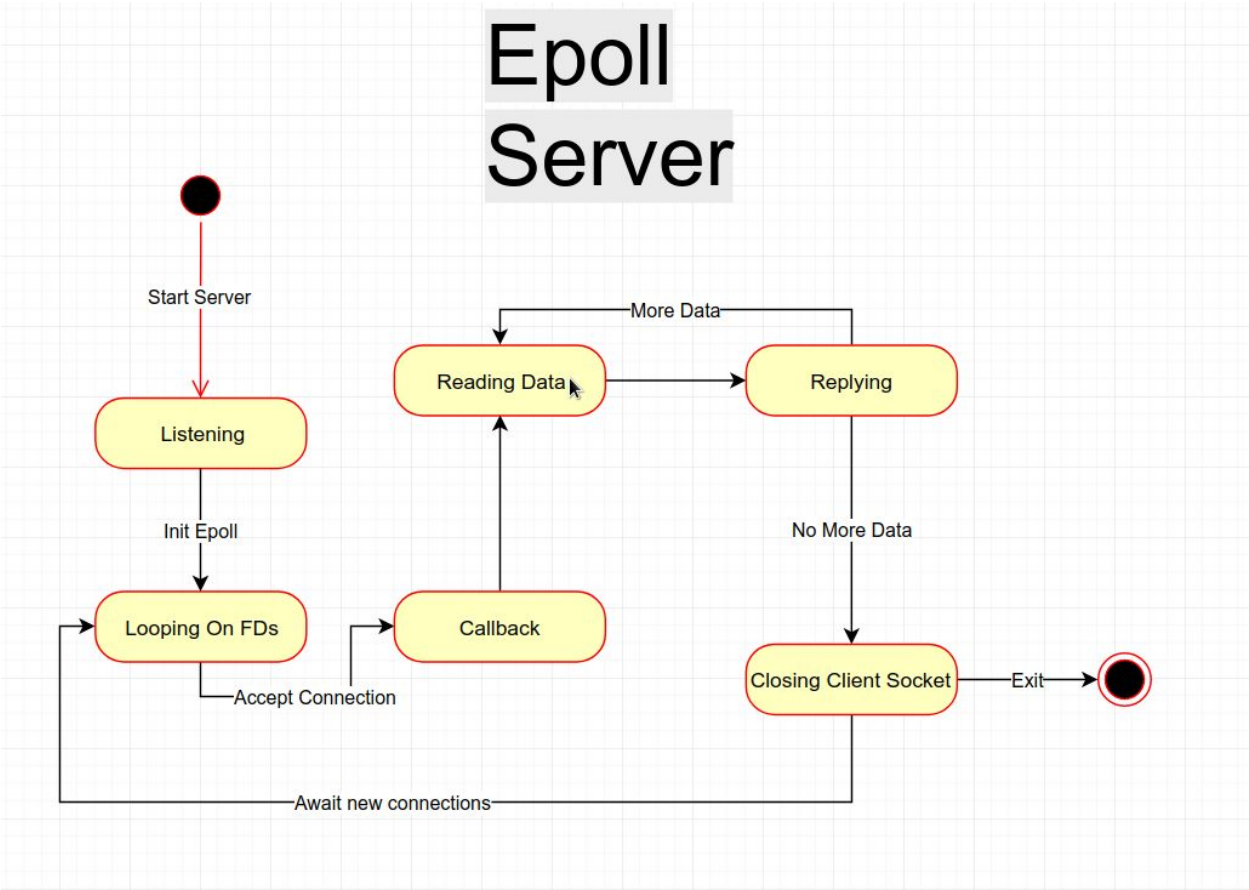
```
TX:             cum:   42.7MB   peak:   32.7Mb   rates:   14.1Kb  3.58Kb  4.72Mb
RX:                    46.0MB           33.3Mb            14.5Kb  4.07Kb  4.80Mb
TOTAL:                 88.7MB           66.0Mb            28.5Kb  7.64Kb  9.51Mb
```

Transfer speeds are at around 40% utilization of the NIC over WIFI.

Epoll Server

The epoll server was written with the assistance of the libev library which focuses on allowing developers access to all of the memory while optimizing epoll's lower level operations. Python uses this library as the back end to their epoll and select statements.

Here is the original design for the epoll server.



When the epoll server has no more connections it simply keeps listening for more connections and never closes unless the user forces the executable to shut down.

Used by running *./bin/server epoll [bufflen]*

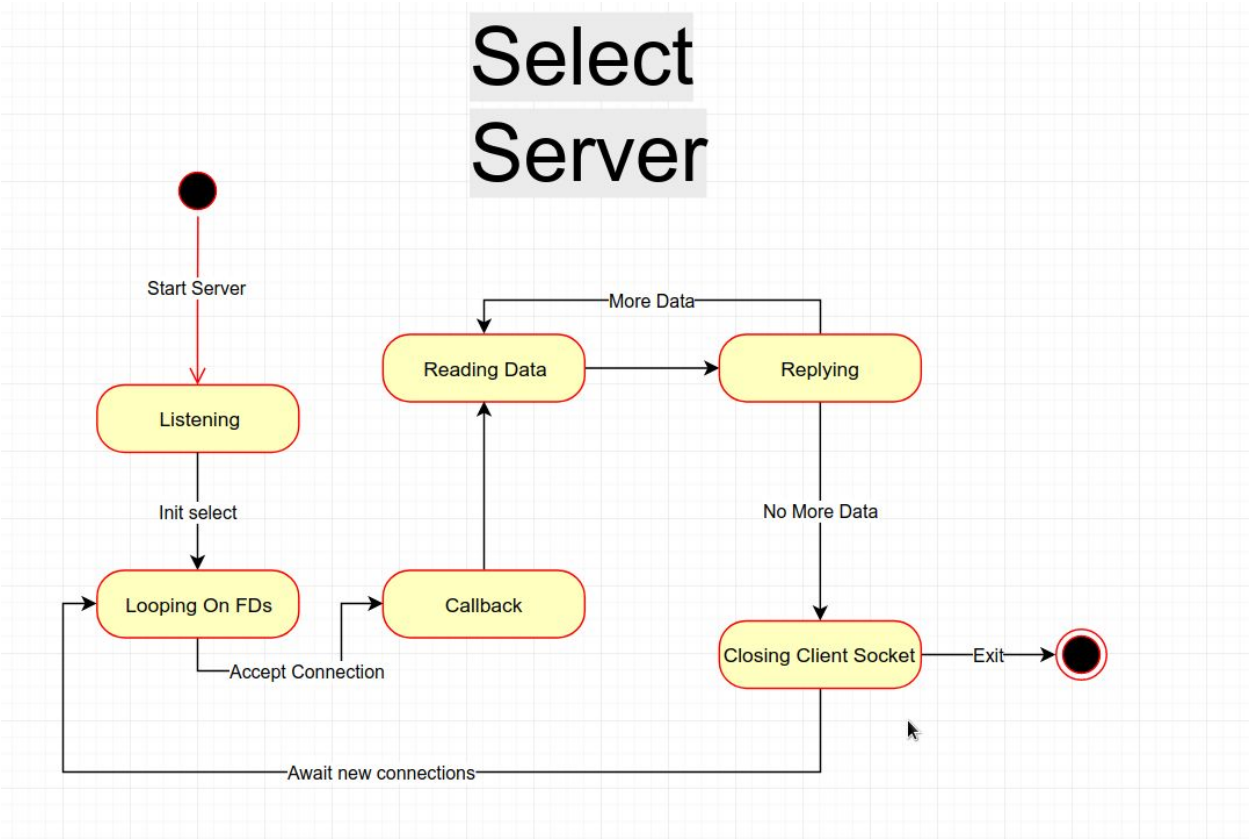
Epoll performs a bit better than select at higher numbers of clients.

```
TX:             cum:   41.8MB   peak:   40.8Mb   rates:   36.6Mb  33.4Mb  23.9Mb
RX:                    42.7MB           39.7Mb            36.3Mb  34.1Mb  24.4Mb
TOTAL:                 84.5MB           80.4Mb            72.8Mb  67.6Mb  48.3Mb
```

Epoll similarly comes close to pinning the NIC under the testing circumstances. There is a noticeably lower impact on the CPU during my testing. The basic server was running at around 30% utilization of every core. Epoll was at 5%.

Select Server

The select server was written with the assistance of the libev library which focuses on allowing developers access to all of the memory while optimizing epoll's lower level operations. Python uses this library as the back end to their epoll and select statements.



Used by running *./bin/server select [buflen]*

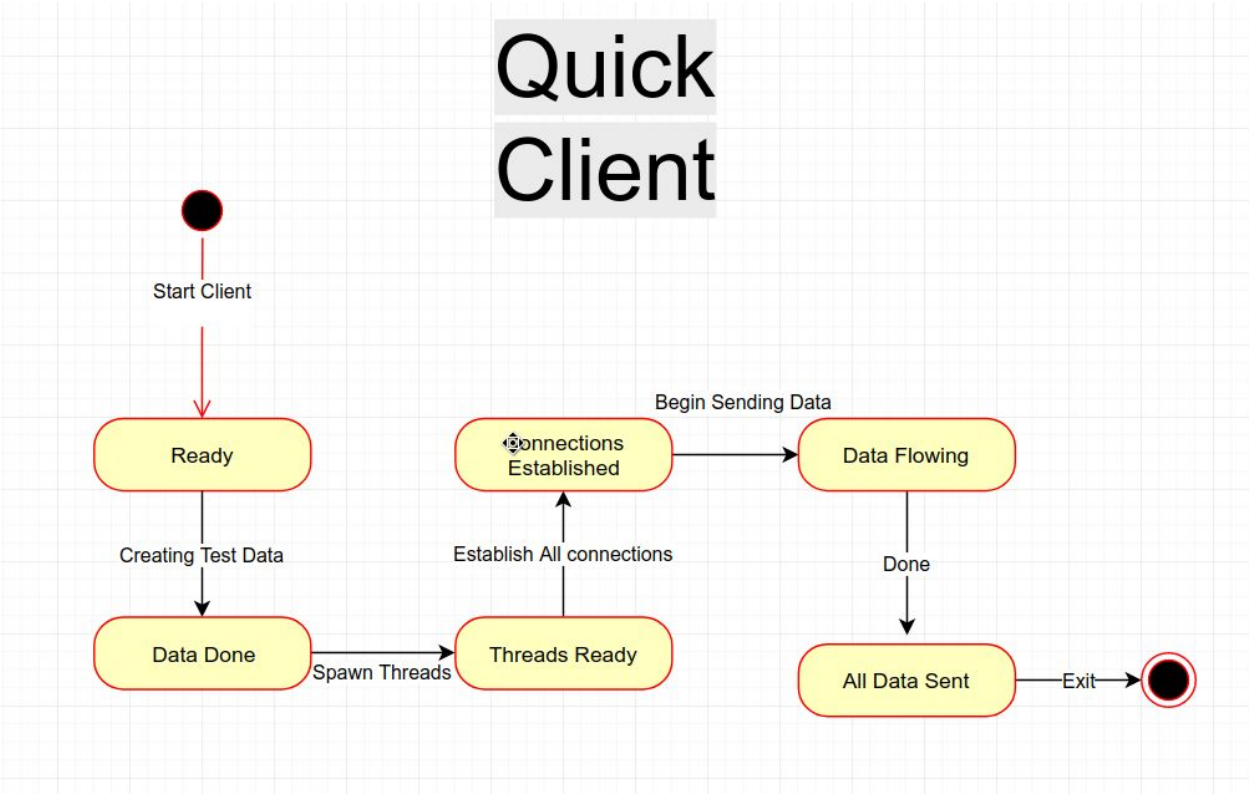
Select seems to be at a point between the basic threaded server and epoll. On one hand it does not use as much CPU power when at lower client counts but on the other hand it suffers in performance at higher client counts.

```
TX:             cum:   22.8MB   peak:   35.4Mb   rates:   35.4Mb  18.2Mb  9.12Mb
RX:                    23.6MB           34.5Mb            34.5Mb  18.9Mb  9.43Mb
TOTAL:                 46.4MB           69.9Mb            69.9Mb  37.1Mb  18.5Mb
```

In my testing I found that epoll outperforms select after around 1000 clients.

Quick Client

The quick client uses python and the default python threading library to generate a number of threads to simultaneously connect to the server.



Used by running

./quick_client [dst_ip] [dst_port] [num_clients] [connection_delay] [num_packets]