IOMULTIPLEXING,

WHAT IS I/O MULTIPLEXING

- Reading from file descriptor is blocking
- Openning new threads/processes is expensive and also requires locking API
- Many APIs (select, poll, pselect, epoll etc..)
- We will focus on two simple APIs: select and poll
- Non-Blocking I/O is a CPU hog
- Check chapters 7.1, 7.2, 7.3 in beej guide for network programming

How to implement chat

Attempt 1 - The process calls recv(2) on one client socket(2)

STARVATION

If client tries to connect while we are stuck on recv - no communication is created

How to implement chat

Attempt 2 - The process calls accept(2) on listening socket(2)

STARVATION

If client tries to communicate while we are stuck on accept - no data is transferred

How to implement chat

Attempt 3 -

Change all sockets to non blocking. i.e recv/accept will fail if no data/new connection (We didn’t study. don’t worry this is very very rarely the solution. as it is not here)

CPU hog

We are stuck on 100% CPU doing nothing!

What do we need

When we listen on one file descriptor - we are idle. The OS wakes us up when there is input. (not a CPU hog!)

We need something similar accept we need to wait on multiple file descriptors.

We need the OS to wake us up if there is input on ANY of them.

We also want the OS to tell us where the input is.

Enter I/O Mux

SELECT(2)

- Oldest API that exist everywhere..
- API is lousy
- Receives 3 sets of File descriptors
- RUINS ITS ARGUMENTS

  i.e. same parameters are used for input and output!
- Returns 3 subsets of the original sets
- ALSO receives timeout (may receive more accurate timeout with pselect(2) and also signal masks)
- Still widely used in the industry. Beej teaches this API on 7.3

POLL(2)

- Receives a vector of file descriptors
- API solves some of select(2) problems

  no longer ruins its arguments

  instead - marks on the array which file descriptors are hot.

Beej teaches this API on 7.2

SELECT SERVER AND POLL SERVER IN BEEJ GUIDE