Comp 8005 Asst #2

server performance

By: Peyman

& John Warren

For: Aman Abdulla

Due: March 5, 2017

# Design Work

## Due:

February 26, 1200 hrs. You may work in groups of two.

(extended to March 5 due to personal matters)

## Objective:

To compare the scalability and performance of the select-, multi-threaded-, and epoll- based client-server implementations.

## Requirements

Design and implement three separate servers:

1. A multi-threaded, traditional server

2. A select (level-triggered) multiplexed server

3. An epoll (edge-triggered) asynchronous server

all servers must

- handle multiple connections

- transfer specified data to connecgted client

echo client

- send variable length strings

- send configurable quantity of messages

- maintain connection

- \*\*resultant\*\* varying duration

Logging

- track when performance degrades

- track number of connections

- track turnaround time (performance) as load increases

- load vs performance

- maintain stats on both server and client

- multiple instances on multiple machines

Note that you will need to have the client maintain the connection for varying time durations (depending on how much data and iterations). The idea is to keep increasing the load on the server until its performance degrades quite significantly. In other words we want to measure how many (scalability) connections the server can handle, and how fast (performance) it can deliver the data back to the clients.

## Restraints

- The server will maintain a list of all connected clients (host names) and store the list together with the number of requests generated by each client and the amount of data transferred to each client.

- Each client will also maintain a record of how many requests it made to the server, the amount of data sent to server, and the amount of time it took for the server to respond (i.e.,

echo the data back).

- You are required to summarize all your data and findings in a properly formatted technical report. Make extensive use of tables and graphs to support your findings and conclusions.

## Additional Notes

have threads ready for connections don't make them as connections come in

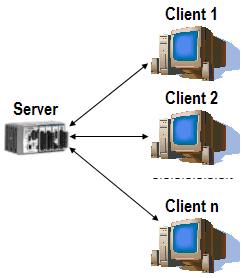
use netstat to make sure that the connections aren’t timed out

richard stevestons v1 chapter6 pool good place to start

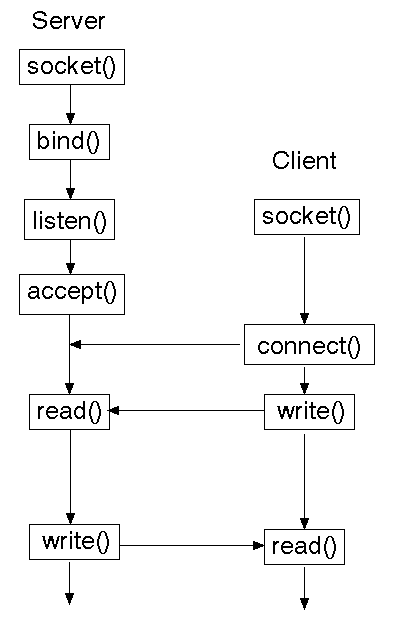
nmap for checking network, less intensive than wireshark

strace -e trace=network

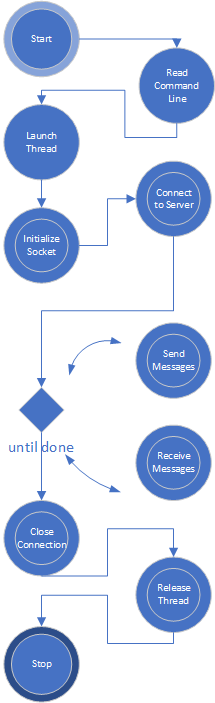
## Diagrams



System Diagram

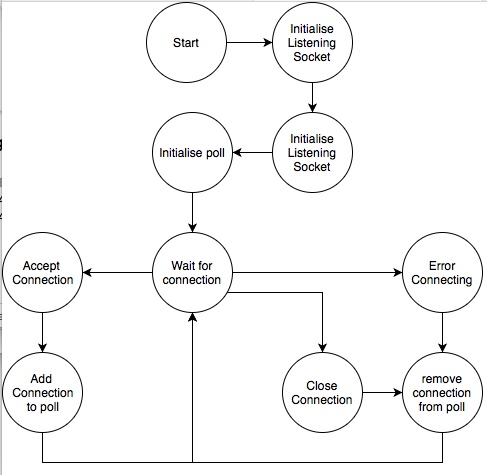


System Calls

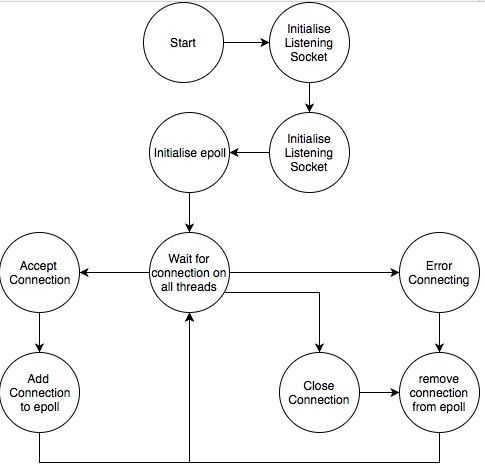


Client

Traditional – Threads



Select



Epoll

## Pseudo Code

## Client

// INPUT: server-ip server-port sequence payload-size duration startup delay

// OUTPUT: client IP, client instance number, requests sent, total data sent, lowest RTT, highest RTT, avgRTT

Initialize:

build/define message

for i=1 to n-clients

start thread

build and configure socket,

open connection,

capture start time, determine end time = start time + duration

while < end-time

sendtime=now()

send packet()

listen

receive packet()

rcvtime=now()

update times;

latest = sendtime-rcvtime

average += latest

if latest < min then min=latest

if latest > max then max=latest

count++

end while

capture end time

avg /= count

log (client-number, end time - start time, (= total running time), requests-sent, payload\*count (=data-sent), min, max, avg)

close connection

close thread

end for

## Threaded Server

## Signal Server

create socket for listening for clients

create epoll struct

add socket to epoll

wait for epoll event

if error in event

remove client from data strctires

if a file discriptor has data to read

if it's a new connection

make connection non blocking

add connection to epoll

else it's an message

echo back

if event is we can write echo harder

## ePoll Server

create socket for listening for clients

create epoll struct

add socket to epoll

on all physical threads

wait for epoll event

if error in event

remove client from data strctires

if a file discriptor has data to read

if it's a new connection

make connection non blocking

add connection to epoll

else it's an message

echo back

if event is we can write echo harder

# Test Plan

# Results

## Traditional / Threaded Server

Statistics

IO

## Signal Server

Statistics

Summary created by Wireshark 2.4.4 (v2.4.4)

File:

Name: /var/tmp/wireshark\_any\_20180302160504\_OdXH6I.pcapng

Length: 21720439338 bytes

Format: Wireshark/... - pcapng

Encapsulation: Linux cooked-mode capture

Time:

First packet: 2018-03-02 16:05:04

Last packet: 2018-03-02 16:12:18

Elapsed: 00:07:13

Capture:

Capture HW: Intel(R) Core(TM) i5-4590 CPU @ 3.30GHz (with SSE4.2)

OS: Linux 4.15.3-300.fc27.x86\_64

Capture application: Dumpcap (Wireshark) 2.4.4 (v2.4.4)

any:

Dropped packets: 54264 (0.482%)

Capture filter: none

Link type: Linux cooked-mode capture

Packet size limit 262144 bytes

Statistics:

Packets: 11246609

Between first and last packet:433.326 sec

Avg. packets/sec: 25954.178

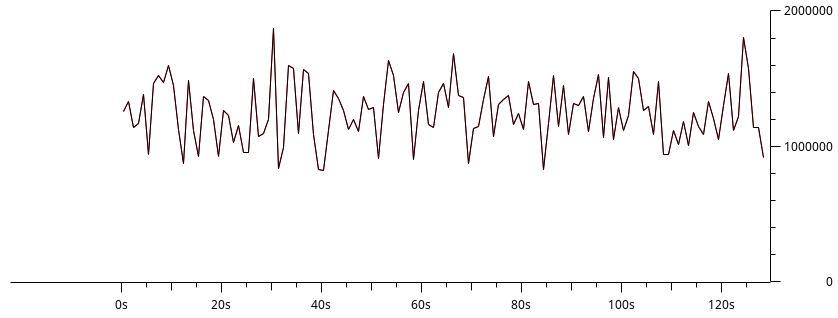
Avg packet size: 1899.288 bytes

Bytes: 21360544988

Avg bytes/sec: 49294448.094

Avg Mbit/sec: 394.356

IO



## Epoll Server

Statistics

Summary created by Wireshark 2.4.4 (v2.4.4)

File:

Name: /root/Documents/anaconda-ks.cfg2.pcapng

Length: 247156550 bytes

Format: Wireshark/... - pcapng

Encapsulation: Linux cooked-mode capture

Time:

First packet: 2018-03-02 15:30:32

Last packet: 2018-03-02 15:46:43

Elapsed: 00:16:10

Capture:

Capture HW: Intel(R) Core(TM) i5-4590 CPU @ 3.30GHz (with SSE4.2)

OS: Linux 4.15.3-300.fc27.x86\_64

Capture application: Dumpcap (Wireshark) 2.4.4 (v2.4.4)

any:

Dropped packets: unknown

Capture filter: none

Link type: Linux cooked-mode capture

Packet size limit 262144 bytes

Statistics:

Packets: 39011

Between first and last packet:970.510 sec

Avg. packets/sec: 40.196

Avg packet size: 6303.553 bytes

Bytes: 245907922

Avg bytes/sec: 253380.206

Avg Mbit/sec: 2.027

IO

