CS241 Lawrence Angrave L24 – Epoll. Server&Port know-how. Intro to Scheduling

Review:

What is htons? ntohs? Why do we need them? What do their names stand for?

What are the "four calls" to set up the server? What is their order? And what is their purpose?

Quick comment: How to use freeaddrinfo

struct addrinfo hints, \*result;

...

getaddrinfo( addr, port, &hints, &result);

freeaddrinfo(result);

What is port hijacking? What steps does the O/S take to prevent port hijacking?

When I restart my program how can I reuse the same port immediately?

Writing high-performance servers; handling 1000s of concurrent sockets The *select* – *poll* – *epoll* story

Differences between select/poll and epoll?

Heuristics:

What would be a good use-case for *select*?

What would be a good use-case for *poll*?

What would be a good use-case for *epoll*?

**Useful Socket/Port Know-how for developers**

getaddrinfo(NULL, "0", &hints, &result); // ANY Port

Later...

struct sockaddr\_in sin;

socklen\_t socklen = sizeof(sin);

if (getsockname(sock\_fd, (struct sockaddr \*)&sin, &socklen) == -1)

perror("getsockname");

else

printf("port number %d\n", \_\_\_\_\_(sin.sin\_port));

***Client IP address?***

struct sockaddr\_in client\_info;  
 int size = sizeof(client\_info);

int client\_fd = **accept**(sock\_fd, (struct sockaddr\*) &client\_info, &size);

char \*connected\_ip= inet\_ntoa(client\_info.sin\_addr); // Does this look thread-safe to you?

int port = ntohs(client\_info.sin\_port);

printf("Client %s port %d\n", connected\_ip, port);

**Scheduling**. Some terms...

How shall I compare thee?

"Turnaround time"

"Waiting time"

"Response time"

"Throughput"

"Latency"

"Starvation?

Good for Batch? Good use of CPU/IO resources?

Good for Interactive?

Good for real-time systems?

FCFS (aka.....)

SJF

RR

Priority-scheduling

Choosing an appropriate time-quantum

What does Linux use?

Notes

select:

old, cross-platform - Even available on tiny embedded linux systems

Requires simple but O(N) linear scan- so does not scale well

Hard-limit on number of selectors

<1ms timeout

poll

Also O(N) scan

OSX support

Good for short-lived sockets or 100s of sockets

can detect closed sockets

1ms+ timeout

Cannot close sockets during poll

event based

epoll – newest. linux specific; not Macosx (use kqueue instead)

good for large (1000s) of long-lived sockets per thread

long-lived = multi I/O requests per connection

1ms+ timeout

event based

Each accept'ed call needs to be added to the set

.. what if I have 100s of long-lived sockets on Linux? poll vs epoll? Ans: There may not be a significant difference between either approach. Try both and benchmark.

An excellent in-depth article about the differences between select, poll and epoll:

http://www.ulduzsoft.com/2014/01/select-poll-epoll-practical-difference-for-system-architects/