Sep 26th: Flash, SEDA

Last time:

- Basics of Dist Sys.
  \[ \Rightarrow \text{comm: u-net} \Rightarrow \text{perf.} \]
  \[ \Rightarrow \text{Failure: everything breaks} \left\{ \Rightarrow \text{Time/concurrency} \right. \]
  \[ \Rightarrow \text{causality \Rightarrow what can affect what?} \]

Techniques:

- Case study (end)
  work back to:
  what do I need to know?

Start: Server Concurrency Architecture
Server: what is it?
   (how different than app?)
   => # of simultaneous users
   => how long they run for
       server: runs "forever"
           (?) harder to build
           correctly
       => workload:

Concurrency Architecture:
=> Processes => running program
       (address space, I/O, CPU state, etc.)

=> Threads => ~ process
       \[ N \text{ threads} \rightarrow 1 \text{ addr. space} \]
       => OS not aware
       => OS fully aware, schedules each thread
How to construct server?

Option #1: "Process per request"

fork()

Problems

⇒ overhead
⇒ memory
⇒ create/schedule
⇒ hard to share (caches, etc.)
⇒ control (of scheduling)
⇒ OS: does scheduling

Positives:

⇒ "easy" to program (isolation: bug, ...)
⇒ (no concurrency bugs)
⇒ straight-line code
⇒ overload

Thread (kernel threads)
"Thread Pool"

goals:
- "easy" to program (but concurrency)
- share addr. space; shared caches, etc.

negatives
- synch: necessary
- overhead (less so)
- sched. control (same)
- overload

```
work: while (1) {
    set next request();
    reqs
    queue (prod/cons)
    fixed "pool" of threads
}
```
3 positives => avoids overload: controlled

amt. of conc.

=> gives some sched control

=) "easy" : similar to before

leave some open questions:


thread pool size

N CPUs => ? (depends:

workload

CPU<->I/O

=) scale?

Event-based Programming

simplest form:

\[ \xi \text{ thread} \]
while (1) full control oversched. get something to do do it

what is an event?

\[ \Rightarrow \text{stringent in definition:} \]

\[ \text{non-blocking} \]

(do not issue some req. and go to sleep)

e.g. read a file

threads:

\[ \text{read( ); } \]

// ok if blocks

\[ \text{events:} \]

1) read_begin()

// starts I/O

asynch. I/O

later:

2) later why
Problems:

(page fault)

""

implies:

harder to program

(no stack)

single CPU model:

concurrency?

But, $>1$ CPU

implies: synch. (again)

Flash:

$\Rightarrow$ lays out early alternatives (concurrent arch)

$\Rightarrow$ handles lack

SEDA

$\Rightarrow$ common

workflows in

servers

$\Rightarrow$ many
of async I/O stages)

Question:
can you really implement your own scheduling policy in SEDA?

=> isolation

scheduled in shared storage

performance: scheduling
We are doing analysis of existing storage servers, and (how to fix).