Sep 26th : Flash, SEDA

Last time:

Basics of Dist Sys.

=> Comm : U-net

=> perf.

[=> Failure : everything breaks]

=> Time / Concurrency

causality => 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.)

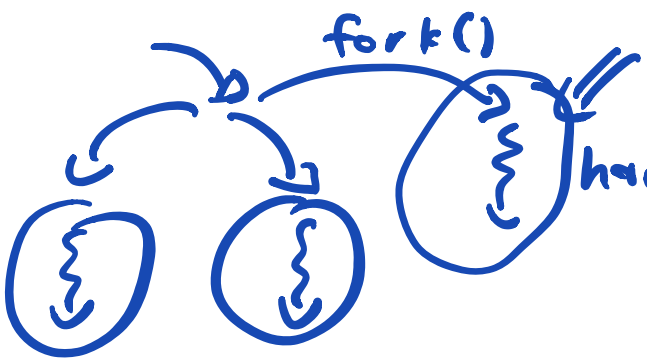
⇒ **Threads** ⇒ ~ process
[N threads ⇒ 1 addr. space]

→ user-level ⇒ OS not aware

→ kernel-level ⇒ OS fully aware, schedules each thread

How to construct server?

Option #1: "Process per request"



separate addr. space

handle req. in entirety

Problems

⇒ overhead
- ⇒ memory
- ⇒ create/switch

⇒ hard to share (caches, etc.)

⇒ control (of scheduling)

⇒ OS: does scheduling

Positives:

⇒ "easy" to program (isolation: bug, ...) (no concurrency bugs) straightline code

⇒ overload

Thread (kernel threads)

"thread per request"

positives

⇒ "easy" to program (but concurrency)

⇒ share addr. space: shared caches, etc.

negatives

synch: necessary

⇒ overhead (less so)

⇒ sched. control (same)

⇒ overload

"Thread Pool"

queue (prod/cons)

fixed "pool" of threads

```
work:
    while (1) {
        get next request();
```

```
process();
}
```

positives

amt. of conc.

=> avoids overload : controlled

=> 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:

```
{ thread
```

while (1)

get something to do ← fell control over sched.

do it

what is an event? ⇒ stringent in definition: non-blocking (do not issue some req. and go to sleep)

e.g. read a file

threads:

read(): ← // ok if blocks

≡

events:

1) read-begin() // starts I/O

asynch. I/O

later:

2) later when

2) later, when I/O is done, do next steps

Problems: (page fault)

" "

=> "harder" to program (no stack)

single CPU model: concurrency? easy, no synch.

But, > 1 CPU

=> synch. (again)

Flash:

=> lays out early alternatives (conc. arch)

=> handles lack

SEDA

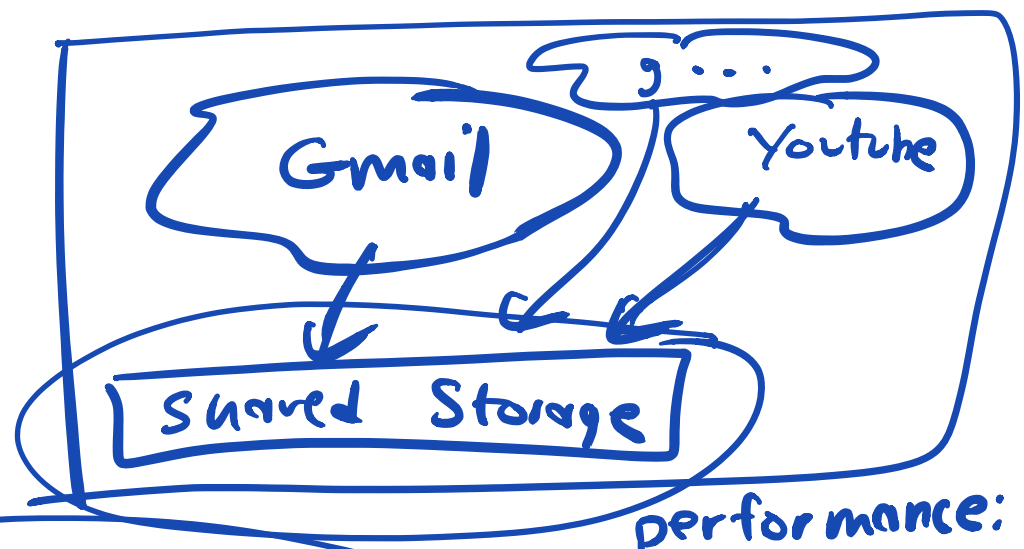
=> common workflows in servers

(many

of asynch I/O stages)

⇒ Question:

can you really implement your own scheduling policy in SEDA?



⇒ isolation

performance: scheduling

what are sched in shared storage ... is important

we doing

=> analysis of existing storage servers

=> and [how to fix]