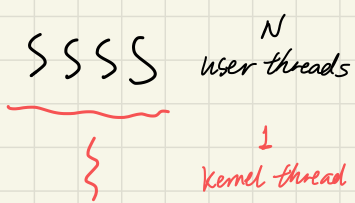
5/6/24 User Level Threads & Storage Device

- managed & scheduled by the user libraries / runtime
- **why use them?**
  - cheaper to create & schedule
  - custom scheduling policies
    - app specific decisions (important lock holders stay running, deadline driven policies)
    - cooperative scheduling
      - schedule only when threads voluntarily yield
- **how does a user level thread run?**
  - on top of a kernel level thread (switches b/wn user threads)
  - N : 1 Model



(tasks) N : M model (threadpool) (workers)

→ **What happens if a user thread blocks?**

→ how might it block?

→ synchronization (e.g. user sleeplock) ↗ transfers control to user schedaler & schedules a different user thread.

→ blocking syscalls, exceptions that generate I/O

→ the underlying kernel thread blocks, no user threads can run on the kernel thread in the meantime.

Golang
→ goroutines
go { ..... }

☆ How can we mitigate this?

→ use nonblocking syscalls when possible

→ keep some back up kernel threads sleeping, wake one up to take over the rest of user threads when a user thread is about to make a blocking syscall.

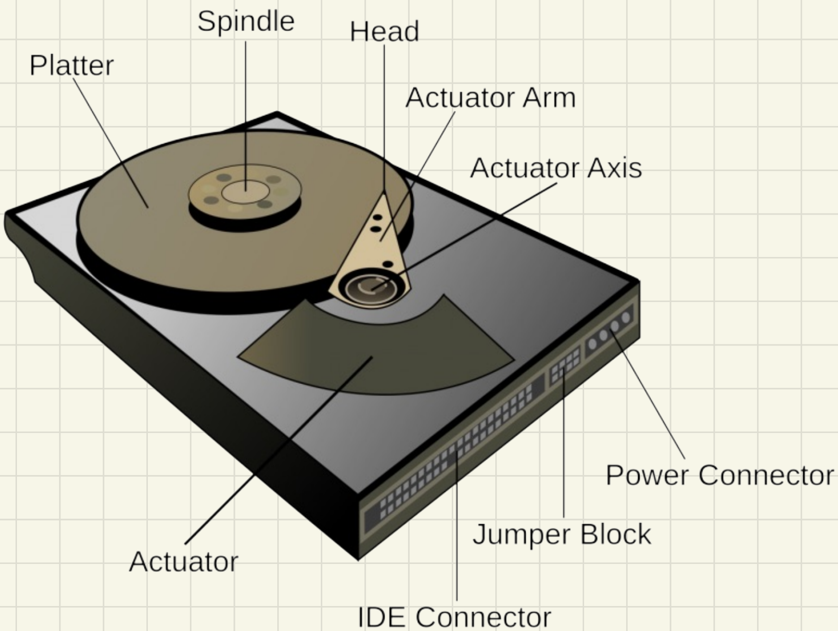
→ class discussion: delay the blocking syscall user thread & run other user threads until close to the end of our time slice

→ challenges: how do we know how much time slice is left? what if it changes (MLFQ)? what if syscall is on the crucial path of execution?

Storage Devices

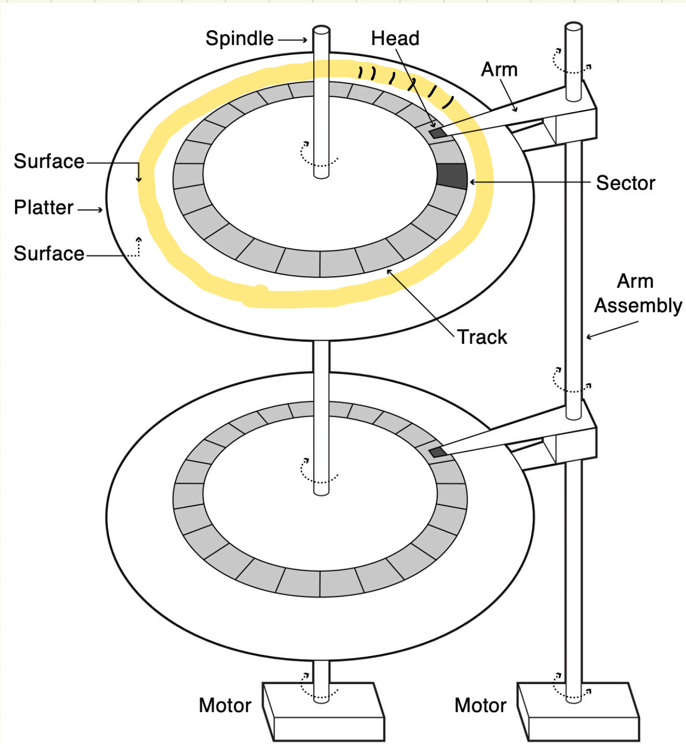
- → persistent (non volatile), large capacity, block addressable
- → hard drive / spinning disk (HDD)
  - → cheap per GB ($10-20 per TB)
  - → physical movement needed (slow access latency, 10-20ms)
- → solid state drive (SSD)
  - → more expensive (~3x) than HDD, cheaper than DRAM
  - → no physical movement (faster access = 10us – 100us)

Hard Drive


Spindle
Head
Platter
Actuator Arm
Actuator Axis
Power Connector
Jumper Block
Actuator
IDE Connector

Sectors

→ 512 bytes, unit of read & write



Disk Request

→ host (CPU) sends a request to the disk controller

→ disk moves arm to the specific track

★ seek time : 1-20ms (depending on how far to move)
average 10ms

→ wait for the sector to spin under the disk head

★ rotational time : depends on RPM, 4-15 ms
assume it takes half a rotation to reach the sector (on avg)

→ transfer data back to host (for a read req).

★ transfer time : depends on # of bytes transferred & disk bandwidth (80-160 MB/s)

Request Latency

→ total time (ms) = seek time + rotational time + transfer time

→ cost of reading / writing 1 sector (512 bytes)

→ given 10ms seek time, 7200 RPM, 120 MiB/s

= 120 RPS ↳ $\frac{1}{0.12 \text{ RPMS}}$ = 8.3 ms per rotation

→ 4 ms per half rotation (average)

→ $\frac{512B}{120MiB/s} \times 1024^2 B/MiB \times 1000$ ms/s = 0.004ms

→ total latency = 10ms + 4ms + 0.004ms = 14.004ms!

→ cost of read/write 10 consecutive sectors

10ms + 4ms + 0.04ms = 14.04 ms

1 seek, 1 avg rotation, 5120 bytes

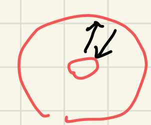
→ cost of read/write 10 random sectors

14.004ms × 10 = 140.04 ms

single request 10 times (different tracks, sectors)

☆ can we improve random access performance?

→ disk head scheduling!



serve requests w/ less seek time.

→ CSCAN

Metrics for evaluating disk performance : IOPS

I/O Operations Per Second : $\dfrac{\#\text{ of I/O requests}}{\text{total latency}}$

→ IOPS for 10 consecutive read requests

$$= \frac{10}{14.04\,ms} \times 1000\,ms/s = 712 \text{ IOPS}$$

→ IOPS for 10 random requests

$$= \frac{10}{140.04\,ms} \times 1000\,ms/s = 71 \text{ IOPS}$$