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 5m user threads)

- \( N = 1 \) Model

  \[
  \begin{align*}
  \text{user threads} & : N \\
  \text{kernel thread} & : 1 \\
  \end{align*}
  \]

- \( N = M \) Model (thread pool)

  \[
  \begin{align*}
  \text{user threads} & : N \\
  \text{tasks} & : N = M \text{ model (threadpool)} \text{ (workers)} \\
  \end{align*}
  \]
What happens if a user thread blocks?

- How might it block?
  - Synchronization (e.g. SleepLock)
  - Blocking syscalls, exceptions that generate I/O
  - The underlying kernel thread blocks, no user threads can run on the kernel thread in the meantime.

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 (MLFO)? 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, 10-20 ms)
- solid state drive (SSD)
  - more expensive than HDD, cheaper than DRAM
  - no physical movement (faster access: 10us-100us)
Hard Drive

- Platter
- Spindle
- Head
- Actuator Arm
- Actuator Axis
- Power Connector
- Jumper Block
- IDE Connector
- Actuator
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 (depends on how far to move)
  average 10ms

→ wait for the sector to spin under the disk head
  * rotational time: 4-15ms
  assume it takes half a rotation to reach the sector (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 = seek time + rotational time + transfer time (ms)

- cost of reading/writing 1 sector (512 bytes)
  - given 10 ms seek time, 7200 RPM, 120 MB/s
  - total latency = 10 ms + 4 ms + 0.004 ms = 14.004 ms!

- cost of read/write 10 consecutive sectors
  - 10 ms + 4 ms + 0.004 ms = 14.04 ms
  - 1 seek + 1 avg rotation 5120 bytes

- cost of read/write 10 random sectors
  - 14.004 ms x 10 = 140.04 ms
  - Single request 10 times (different tracks, sectors)

- can we improve random access performance?
  - disk head scheduling
  - CSCAN

- cost of reading/writing 512B
  - $\frac{512B \times 1024B/MB}{120MB/s} = 0.004 ms$
  - $\frac{1}{0.12 \text{ RPMs}} = 8.3 \text{ ms per rotation}$
  - 4 ms per half rotation (average)

- total latency = 10 ms + 4 ms = 14 ms
  - $\frac{1}{0.12 \text{ RPMs}} = 8.3 \text{ ms per rotation}$
  - 4 ms per half rotation (average)
Metrics for evaluating disk performance: IOPS

I/O Operations Per Second: \( \frac{\text{# of I/O requests}}{\text{total latency}} \)

→ IOPS for 10 consecutive read requests

\[
\text{IOPS} = \frac{10}{14.34 \text{ ms} \times 1000 \text{ ms/s}} = 712 \text{ IOPS}
\]

→ IOPS for 10 random requests

\[
\text{IOPS} = \frac{10}{140.04 \text{ ms} \times 1000 \text{ ms/s}} = 71 \text{ IOPS}
\]