Disk Scheduling

- Scheduling algorithms only relevant to Hard-Disks
    - HDDs have non-uniform *random access* times
    - SSDs have constant *random access* time
- **HDD Structure:**
    - **Platter:** Disk composed of *many* platters stacked atop each other.
    - **Track:** Each *platter* divided into circular rings called *track*.
    - **Sector:** Each *track* divided into sections called *sectors.*
    - **Cylinder:** *Tracks* on all platters at the same distance from center form a *Cylinder.*
    - **Arm:** Arm(s) move back-and-forth; used to access *sector* on the corresponding *platter.*
- **Performance Metrics:** HDD performance can be quantified as:
    - **Transfer Rate:** Speed at which data can be *transferred.*
    - **Random Access Time:** Time taken to reach a particular piece of data.
    - **Seek Time:** Time taken to move disk *arm* to the right position.
    - **Rotational Latency:** Time taken to rotate platters to right position
- **NOTE:** *Seek Time* and *Rotation Latency* are generally of about same importance - both are in *milliseconds* range.
- **Average Access Time:** Time taken to *read* data on the disk.
    - For every *track* access, we can calculate the *Average Access Time.*
    -
    - **T_s:** Average Seek Time
    - **r:** Rotational Speed
    - **b:** Bytes to be accessed from the *track*
    - **N:** Bytes in the *track.*
- **Sequential Storage:** Data is stored sequentially on same track, on same cylinder.
    - Reading a new *track* in same *cylinder* requires additional *rotational latency*, and *transfer time*
    - Reading a new *tracks* in another *cylinder* requires additional *seek time, rotational latency*, and *transfer time*
- **Disk Scheduling Algorithms:**
    - We aim to improve disk performance by optimizing the *Seek Time.*
    - OS is *not* responsible for handling *Rotation Latency*
    - **First come, First serve:**
        - FIFO queue for request storage
        - No optimization attempt made
        - Distance moved by *arm* is high
        - No advantage taken of *spatial locatity*
        - No *starvation*
    - **Shortest Seek Time First:**
        - Re-organizes the request queue to begin with closest *track* first
        - *Reduces,* but *doesn't* minimize total distance moved by the *arm*
        - Takes advantage of *spatial locality.*
        - Subject to *starvation* if newly incoming requests closer than older items in request queue
    - **SCAN:**
        - *Arm* moves in one direction, then *reverses* its movement.
            - *Arm* moves till the *end* of the disk reached.
        - Covers less *Arm* distance than *Shortest Seek Time First*
        - No *starvation*
        - Does *not* take advantage of *spatial locality*
    - **C-SCAN (Circular-SCAN):**
        - *Arm* moves in one direction, then *restarts* its movement from the disk's beginning.
        - No *starvation*

        - Does *not* take advantage of *spatial locality*
        - Reduces wait for *tracks* at the *periphery.*
            - *e.g.* assume there are 200 *tracks,* and a periphery-request is on *track* 1.
            - **SCAN:** Assuming that the *scan* happens from the *lower-to-higher count*, and assuming that the original arm position is at *track* 2, disk will take *2t* time to reach the periphery-request of low-track number.
            - **C-SCAN:** Time taken witll be $t + s\_max$, where *s_max* the maximum *seek time*
            - **NOTE:** *seek time* and *scan time* are *NOT* the same.
    - **LOOK:**
        - Modification of SCAN and C-SCAN
        - *Arm* changes direction after encountering *last* request in that direction
- **Optimizations:**
    - **Double Bufferring:**
        - SCAN algorithm can be optimized by using *double-buffering*
        - While the the incoming requests are received, previously stored requests are being served
        - Prevents the algortihm-run to be interrupted.
        - Assuming the buffer sizes are *C* requests*:*
            - *Smaller* the buffer, more the algorithm represents the *First-come First Served.*
            - *Larger* the buffer, more the algorithm represents the SCAN algorihtm
    - **Real-time Processing:**
        - *Performance* should not be the goal, predictibility must be.
        - *Higher priority* processes must *not* have to wait for *Lower Priority* if priority difference is too high.
            - Especially important in *Real-time Systems.*