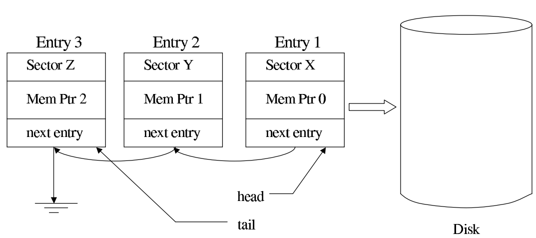
Disk Scheduling

CS 537 - Introduction to Operating Systems

Disk Queues

- Each disk has a queue of jobs waiting to access disk
  - read jobs
  - write jobs
- Each entry in queue contains the following
  - pointer to memory location to read/write from/to
  - sector number to access
  - pointer to next job in the queue
- OS usually maintains this queue

Disk Queues

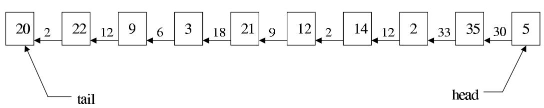


First-In, First-Out (FIFO)

- Do accesses in the order in which they are presented to the disk
- This is very fair to processes
- This is very simple to implement
- Approximates random accesses to disk
  - gives rated, average latency for every read
  - will have large average seeks between each access
- Not a good policy

FIFO

•Reference String: 5, 35, 2, 14, 12, 21, 3, 9, 22, 20



•Calculation of total seek distance:

30 + 33 + 12 + 2 + 9 + 18 + 6 + 12 + 2 = 124

FIFO

- Obviously, reordering the accesses to the disk could greatly reduce the seek distance
  - seek distance ~ seek time
- Want to put close accesses next to each other in the queue

Disk Scheduling

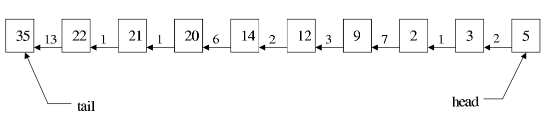
- Recall, statistical average seek time is 9 ms
  - randomly accessing all over disk
- Multiple requests to disk will arrive while one is being serviced
- Can drastically reduce average seek time by intelligent scheduling accesses

Shortest Seek Time First (SSTF)

- When a new job arrives, calculate its seek distance from the current job
- Place it in disk queue accordingly
- Service the next closest access when the current job finishes

SSTF

•Reference String: 5, 35, 2, 14, 12, 21, 3, 9, 22, 20



•Calculation of total seek distance:

2 + 1 + 7 + 3 + 2 + 6 + 1 + 1 + 13 = 35

SSTF

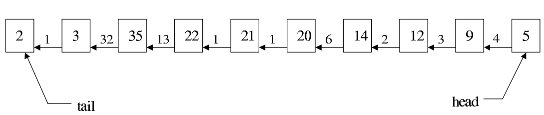
- Provides substantial improvement in seek time over FCFS
- One major issue
  - STARVATION
- What if some accesses are on far end of disk from current access
  - jobs are constantly arriving in a real system
  - jobs closest to the current access will keep getting serviced
  - jobs on the far end will starve

Elevator Algorithm

- Similar to SSTF
- One major difference
  - next job scheduled is closest to current job but in one particular direction
  - all jobs in other direction are put at the end of the list
- Similar to an elevator
  - it goes up first and then comes back down

Elevator Algorithm

•Reference String: 5, 35, 2, 14, 12, 21, 3, 9, 22, 20



•Calculation of total seek distance:
4 + 3 + 2 + 6 + 1 + 1 + 13 + 32 + 1 = 63

Elevator Algorithm

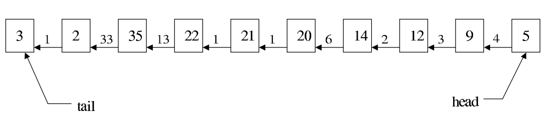
- Avoids starvation
- Provides very good performance
- Still has one major issue
  - FAIRNESS
- Jobs in the middle of the disk get serviced twice as much as jobs at the ends

One-Way Elevator Algorithm

- Exactly like elevator algorithm except scheduling is done in only one direction
  - for example, elevator always goes “up”
- This will require one long seek after finished going up
  - have to go back to the beginning
- This is okay because one long seek doesn’t take very long
  - IBM disk: 15 ms from one end to the other
- This long seek is done infrequently

One-Way Elevator Algorithm

•Reference String: 5, 35, 2, 14, 12, 21, 3, 9, 22, 20



•Calculation of total seek distance:
4 + 3 + 2 + 6 + 1 + 1 + 13 + 33 + 1 = 64