Disk I/O Schedulers
(Theory and in context of Linux)

How A Disk Works

- A disk is a bunch of data blocks.
- A disk has a disk head.
- Data can only be accessed when the head is on that block.
- Disk head continues to move for incoming disk access (read/write) requests.
- Data transfer is fast (once the disk head gets there).

Our Problem

- Suppose we have multiple requests for disk blocks …. Which should we access first?

  Yes, order does matter … a lot.

Problem Statement

- Input:
  - A set of requests.
  - The current disk head location.
  - Algorithm state (direction??)
- Output:
  - Next disk block to get – order of request completion.
  - Total seek distance in terms of cylinders/sectors.
  - Disk Bandwidth

The Goals

- Maximize throughput
  - Operations per second.
- Maximize fairness
  - All disk requests treated equally.
- Avoid Starvation
  - And very very long waits.
- Real Time Concerns (if any)

Priorities are Everywhere

- Every Operating system assigns priorities to all sorts of things
  - Requests for RAM
  - Requests for the CPU
  - Requests for network access
  - **Disk Request priority.**

Possible Algorithms

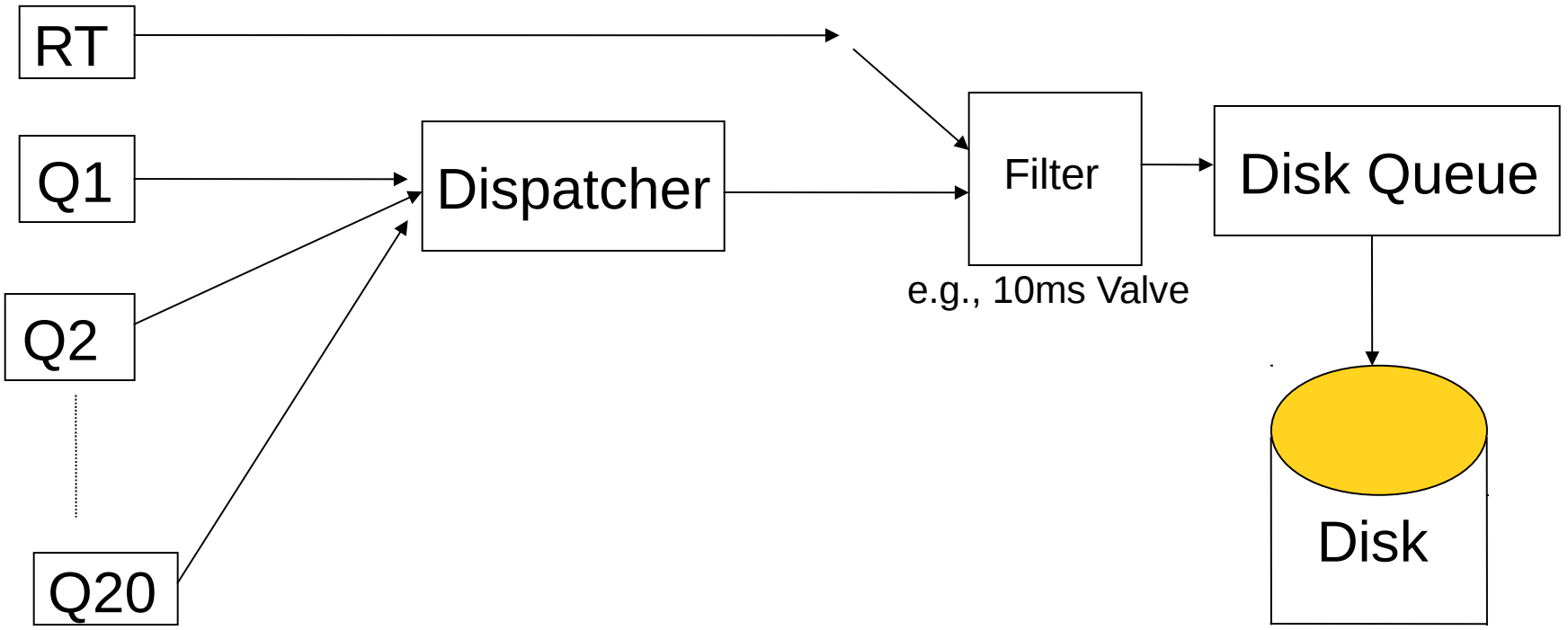
- Pessimist - Choose the disk request furthest from the current disk head position.
- Optimal - Choose the disk request closest to the current disk head position.
- FCFS - Serve the requests in their arrival order.
- Elevator - Move back and forth, solving requests as you go.
- Cyclic Elevator - Move toward the bottom, solving requests as you go.
  - When you've solved the lowest request, seek all the way to the highest request.
  - Performance penalty occurs here.
- Deadline Scheduler - Each request has a deadline.
  - Service requests using cyclic elevator.
  - When a deadline is threatened, skip directly to that request.

Anticipatory Scheduling

- Concept
  - Assume that an I/O request will be closely followed by another nearby one.
- Algorithm
  - After servicing a request … WAIT.
  - Yes, this means do nothing even though there is work to be done.
  - If a nearby request occurs soon, service it.
  - If not … cyclic elevator (C-SCAN/C-LOOK).

Completely Fair Queuing (CFQ)

- Real Time needs always come first. Otherwise, no user should be able to use the disk drive.
- Priorities are OK.



Why this is important to know?

- Because Linux has three/four very different schedulers, and you as a user have a choice to do a selection.

Linux Disk Scheduling

- Linux ships with different I/O schedulers.
  - Deadline
  - Noop
  - Anticipatory, and
  - CFQ

Check which I/O schedulers your Linux distribution is shipped with?

Clearly mention it as a part of your assignment report.

Linux Disk Scheduling

- There are many differences (discussed earlier) between these scheduling algorithms (summary below):
  - CFQ: This is the default algorithm in most Linux distributions. It attempts to distribute all I/O bandwidth evenly among all processes requesting I/O. It is ideal for most purposes.
  - NOOP: The noop algorithm attempts to use as little cpu as possible. It acts as a basic FIFO queue expecting the hardware controller to handle the performance operations of the requests.
  - Anticipatory: This algorithm attempts to reorder all disk I/O operations to optimize disk seeks. It is designed to increase performance on systems that have slow disks.
  - Deadline: This scheduling algorithm places I/O requests in a priority queue so each is guaranteed to be ran within a certain time. It is often used in real-time operating systems.

Few Useful commands

$ sudo -s

$ cd /dev // change to devices directory

$ ls s* // check storage devices

$ lsblk -o KNAME,TYPE,SIZE,MODEL

$ lshw -class disk

$ lshw // check how it is different than previous

$ lshw -class disk -class storage

$ lshw -class disk -class storage -short

$ lsusb // gives you the usb ports info. on your m/c

$ lspci // lists all pci devices/controllers

Concept of device files and its numbers

cd /dev

ls // listing of device files

ls -l sda // your hard disk

ls -l sda1 // what's the difference than above?

Block and char devices – where is the info.?

cd /sys/dev

Look into the block and char directories

What you have to do today?

- Since the 2.6.10 kernel user has been given the option to easily adjust the disk scheduler on the fly without a reboot.
- You can see which scheduler you are currently running by using the following command:

cat /sys/block/sda/queue/scheduler

noop anticipatory deadline [cfq] // all or some of them on your linux distribution?

- In this case sda is the block device in question.
- We can see it is currently running the cfq scheduler.

- Changing the Disk I/O scheduler

echo noop > /sys/block/sda/queue/scheduler

cat /sys/block/sda/queue/scheduler

[noop] anticipatory deadline cfq

- Changing schedulers on the fly allows you to test and benchmark the algorithms for your specific application.
- Once the change is issued, any current I/O operations will be executed before the new scheduler goes into effect, so the change will not be instantaneous.

Algorithm Analysis

- Optimal
  - It is known to have highest performance in operations per second.
  - It’s unfair to requests toward the edges of the disk.
  - It allows for starvation.
- FCFS
  - It’s fair and avoids starvation.
  - It’s medium lousy performance.
  - Some simple OSs use this.
- Elevator (Non-Cyclic by DEFAULT)
  - Performance good
  - Fair, but files near the middle of the disk get more attention.

Algorithm Analysis ...cont'd

- Cyclic Elevator (C-SCAN, C-LOOK)
  - It’s fair and starvation-proof.
  - It’s very good performance - Almost as good as elevator.
  - It’s used in real life (and every textbook).
- Deadline Scheduler
  - Gives Priority to Real Time Processes, Fair otherwise.
  - No starvation, unless a real time process misbehaves.
- Anticipatory Scheduling
  - Fair, No starvation. No support for real time.
  - Makes assumptions about how processes work in real life.
- CFQ
  - Has Real Time support. Fair.
  - No starvation is possible. Allows for priorities.

Benchmarks

- In Real Operating Systems …
  - Performance is never obvious.
  - No one uses the textbook algorithm.
  - Benchmarking is everything.
    - Theory is useful, if it helps you benchmark better.

Try the following experiments

Copy 1GB file

- On disk from one directory location to another.
- On USB from one directory location to another.
- From usb to disk.
- From disk to usb.
- Compare performance of different scheduling policies and report the result for your machine for the linux distribution you have installed on your machine.

Benchmark 1: Copy 1 GB file on disk			
Policy	Real time	Sys time	User time
CFQ			
Deadline			
Noop			

Use the table Template on left for Other benchmarks Listed above.
Your report should have 4 tables each for the results obtained on VM and host machine.

Example Benchmarks

```
time (find kthread.c -type f | xargs cat > /dev/null)
```

```
time (cp 1-gig-file foo ; sync)
```

```
time (cp 1-gig-file foo)
```

```
time (ssh testbox xterm -e true)
```