Block Device Driver

Pradipta De
pradipta.de@sunykorea.ac.kr
• Block Devices
  – Structure of devices
  – Kernel components
  – I/O Scheduling

• USB Device Driver Basics
Block Devices

- Block Devices are characterized by random (non-sequential) access

- Key terms to remember:
  - Sector: unit of storage (or read/write granularity)
    - Hard disk: 512 bytes, CD: 2KB
  - IO Block: Unit of storage represented by OS
    - Can be any multiple of sector, but less than page size
  - IO buffers: data structure that represents a block
Hard Disk Structure

Rotation Time: time to move head to a sector on a track
Seek Time: time to move head to a track
Transfer Time: Time to transfer bytes based on transfer bandwidth

Disk IO Time = Seek Time + Rotation Time + Transfer Time
Implication of HD structure

• Read/write from a process specifies
  – Input or output operation
  – Disk address (hdaX or sdaX)
  – Memory address
  – Number of sectors to transfer

• When requests arrive fast, these are queued up
  – How to dispatch the requests from the queue to ensure high throughput?
Kernel Components

Determines the file block numbers. Invokes a filesystem specific function that accesses files disk inode and determines logical block numbers.

Hides the differences of each hardware block device. Uses “BIO” data structure: descriptor for ongoing IO

Sorts the pending IO requests

Block device drivers initiates actual data transfer by sending suitable commands to the hardware interfaces of the disk controllers
IO Scheduling: General Solutions

- **FCFS**: dispatch the request for disk blocks in the order they arrive
  - may incur large seek time

- **Shortest Seek Time First**:  
  - similar to “Shortest Job First” process scheduling  
  - Pick the request with the shortest seek time  
    (note: we can compute the seek time unlike job completion time)  
  - May lead to starvation of requests
IO Scheduling: General Solutions

• SCAN scheduling:
  – move the disk head from one end to other, and service requests in that order

• C-SCAN: Circular SCAN
  – Move the disk back to the starting point instead of doing a reverse scan
  – Why? After a scan, where are the requests likely to be concentrated
  – Incremental improvement: move the arm till the point of last serviceable request (LOOK and C-LOOK)
Linux IO Scheduling
(c.f. Robert Love’s book)

• [2.4] Linus Elevator:
  – Insert request in the queue in order of sector number
    \(\Rightarrow\) sorted on sector number, then pick sequentially from the list
  – Can merge adjacent sector requests
  – Starvation prevention: check if a request has waited longer than a threshold
    \(\Rightarrow\) the new request cannot be ahead of this request

• Starvation of requests was still a problem
  – Many requests to a region of a disk may lead to low throughput for other requests
Linux IO Scheduling
(c.f. Robert Love’s book)

• Deadline IO Scheduler:
  – Writes can wait longer than reads. Reads for sectors are typically clustered
  – Assign shorter deadline for read (500 ms) and longer for write (5 sec)

• Anticipatory Scheduler:
  – Under heavy writes, deadline IO scheduler suffers in throughput ⇒ have to jump back and forth between servicing read and write requests
  – Anticipation heuristic: wait (with no activity) expecting a read (6 ms) ⇒ may receive a request close to the current read ⇒ saves two seek times
Complete Fair Queueing IO Scheduler

- One IO request queue per process
  - P1’s and P2’s requests are sorted sector-wise in their respective IO queues
  - Apply round-robin to service the queues
  - Service a pre-defined (configurable) number of requests from each process IO queue
  - Idea is to provide fair slice of disk bandwidth to each process
    - Application scenario is multimedia workload
    - Works well for most other workload, including desktop
A block device driver may handle several block devices
- IDE device driver manages several IDE disks
- Each partition of a disk is treated as a separate block device managed by the same device driver
USB Device Drivers
USB Hardware Layout

- Universal Serial Bus (USB) Features
  - It is actually a point to point connector
  - Laid out as a tree connecting only two nodes
  - A non-leaf node in the tree is a USB hub
  - The host controller implements a USB hub
USB Kernel Components

Interface to access and control USB hardware by hiding differences among various USB hardware controllers
Typical Data Transfer

- USB code uses a data structure called URB (USB Request Block) to communicate from kernel to USB devices.

- The lifecycle of a URB:
  - Created by a USB device driver.
  - Assigned to a specific endpoint of a specific USB device.
  - Submitted to the USB core, by the USB device driver.
  - Submitted to the specific USB host controller driver for the specified device by the USB core.
  - Processed by the USB host controller driver that makes a USB transfer to the device.
  - When the urb is completed, the USB host controller driver notifies the USB device driver.
Putting It Together

• Block Device
  – Anatomy
  – Device driver software components
  – IO Schedulers

• USB device driver in Linux