I/O Systems

1DV512 – Operating Systems

Dr. Kostiantyn Kucher

kostiantyn.kucher@lnu.se

December 8, 2020

Based on the Operating System Concepts slides by Silberschatz, Galvin, and Gagne (2018)

Suggested OSC book complement: Chapters 12 & 13

Agenda

- Motivation and Introduction
- Storage Devices
- Storage Device Support
- I/O Support
- Summary

Motivation

- As discussed in the previous lectures, data and code instructions are stored in CPU registers, CPU cache, and main memory (modern implementation: RAM) ⇒ *primary storage*
- Main memory is typically *volatile* ⇒ cannot store data in a *persistent* way across system restarts, disconnected parts, power failures. . .
  (exception: *ROM* used for BIOS, etc.)
- Additionally, main memory volume is often not enough for storing larger amounts of data (and executable files) ⇒ the need for additional *mass storage*!
- Compared to primary storage, such storage types (*secondary*, *tertiary*, *offline*. . .) require access via input/output operations and are considerably slower than main memory
- In this lecture, we will focus mainly on secondary storage (disk drives) and support for I/O in operating systems

Agenda

- Motivation and Introduction
- Storage Devices
- Storage Device Support
- I/O Support
- Summary

Hard Disk Drives

- Bulk of secondary storage for modern computers is *hard disk drives* (HDDs; commercially available since 1950s) and *nonvolatile memory* (NVM) devices
- HDDs spin platters of magnetically-coated material under moving read-write heads
- *Tracks* divided into *sectors*; set of *tracks* under the current disk arm position $\Rightarrow$ a *cylinder*
- *Transfer rate* $\Rightarrow$ rate at which data flow between drive and computer
- *Positioning time* (*random-access time*) $\Rightarrow$ time to move disk arm to desired cylinder (*seek time*) + time for desired sector to rotate under the disk head (*rotational latency*)
- A*verage access time* = average seek time + average latency
- Average I/O time = average access time + (amount to transfer / transfer rate) + controller overhead

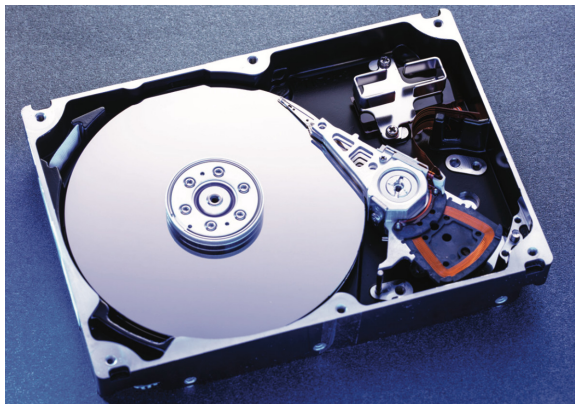


Fig. 11.2 in OSC book

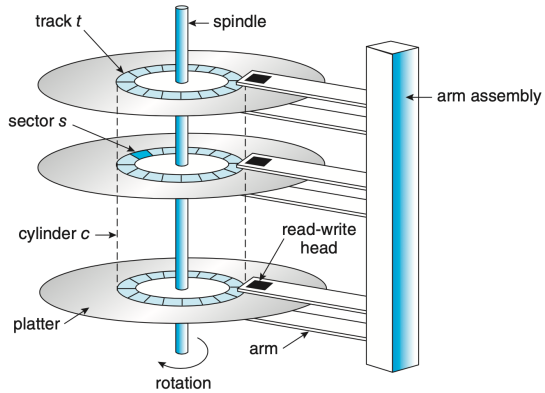


Fig. 11.1 in OSC book

Non-Volatile Memory Devices

- If disk-drive like ⇒ called *solid-state disks* (SSDs); also USB drives and flash memory cards
- SSDs ⇒ typically lower volume and more expensive than HDDs, but much faster!
- Traditional HDD interfaces can be too slow ⇒ connect directly to PCI for example
- No moving parts, so no seek time or rotational latency ⇒ but have their own challenges
- Read and written in "page" increments (think sector) but can't overwrite in place ⇒ must first be erased, and erases happen in larger "block" increments of several "pages"
- With no overwrite, pages end up with mix of valid and invalid data
- To track which logical blocks are valid, controller maintains *flash translation layer* (FTL) table
- Each cell has lifespan, so *wear leveling* needed to write equally to all cells

Fig. 11.3 in OSC book

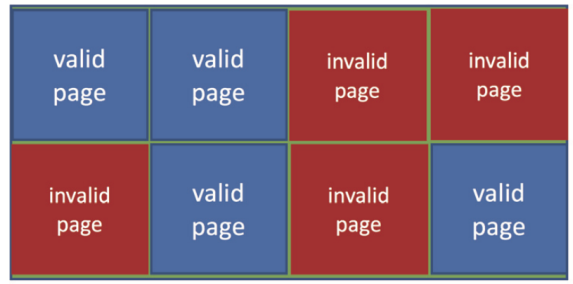


Fig. 11.4 in OSC book

Disk Attachment

- Host-attached storage accessed through I/O ports talking to I/O busses
- Multiple bus standards available, including *advanced technology attachment* (ATA), *serial ATA* (SATA ⇒ most common one), *eSATA*, *serial attached SCSI* (SAS), *universal serial bus* (USB), and *fibre channel* (FC ⇒ used primarily for servers in data centers)
- Because NVM much faster than HDD ⇒ new fast interface for NVM called *NVM express* (NVMe), connecting directly to PCI bus
- Data transfers on a bus carried out by special electronic processors called *controllers* (or *host-bus adapters*, HBAs)
  - Host controller on the computer end of the bus, device controller on device end
  - Computer places command on host controller, using memory-mapped I/O ports
  - Host controller sends messages to device controller
  - Data transferred via *direct memory access* (DMA) between device and computer DRAM

Agenda

- Motivation and Introduction
- Storage Devices
- Storage Device Support
- I/O Support
- Summary

Disk Address Mapping

- Disk drives are addressed as large 1-dimensional arrays of *logical blocks* $\Rightarrow$ the smallest unit of transfer
- Low-level formatting creates logical blocks on physical media
- The 1-dimensional array of logical blocks is mapped into the sectors of the disk sequentially
  - Sector 0 is the first sector of the first track on the outermost cylinder
  - Mapping proceeds in order through that track, then the rest of the tracks in that cylinder, and then through the rest of the cylinders from outermost to innermost
  - Logical to physical address should be easy $\Rightarrow$ except for bad sectors!
  - Also, non-constant # of sectors per track due to constant angular velocity

HDD Scheduling

- The operating system is responsible for using hardware efficiently ⇒ for the disk drives, this means having a fast access time and disk bandwidth
- Minimize seek time; seek time ≈ seek distance
- Disk *bandwidth* = total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer
- There are many sources of disk I/O requests: OS, system processes, user processes
- Each I/O request specifies: input or output mode, disk address, memory address, number of sectors to transfer
- OS maintains queue of requests, per disk or device
- Idle disk can immediately work on I/O request, busy disk means work must queue
- Optimization algorithms only make sense when a queue exists
- In the past, OS responsible for queue management and disk drive head scheduling
- Now, built into the storage device controllers ⇒ just provide LBAs and handle sorting of requests
- Note that drive controllers have small buffers and can manage a queue of I/O requests (of varying "depth")

HDD Scheduling Algorithms

- Multiple algorithms exist to schedule the servicing of disk I/O requests ⇒ here, we use an example with a request queue for cylinders in range 0–199, with the head pointer initially at 53
- *First come, first served* (FCFS) scheduling: simplest and intrinsically fair ⇒ but a lot of extra head movement! (here, 640 cylinders)
- *SCAN* scheduling (the *elevator algorithm*): the disk arm goes in one direction first ⇒ less extra movement (here, 208 cylinders), but extra waiting for the cylinders on the other side...
- *Circular SCAN* (C-SCAN) scheduling: the disk arm goes in one direction and returns to the beginning directly ⇒ provides a more uniform waiting time
- Further algorithms exist, including options with separate queues for reading and writing
- For NVM/SSD drives ⇒ no disk arm movement, but adjacent LBA requests can be merged!

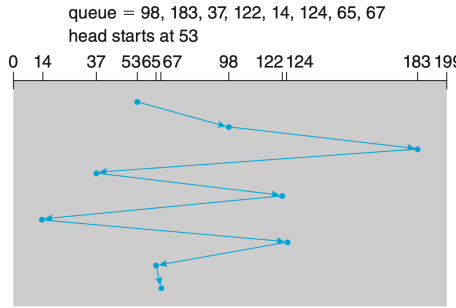


Fig. 11.5 in OSC book

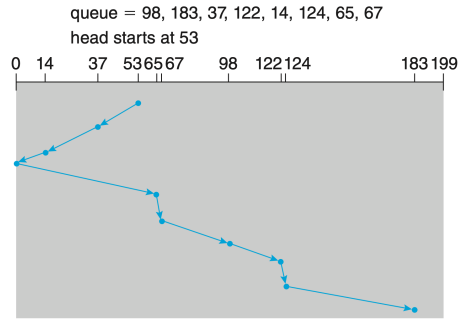


Fig. 11.7 in OSC book

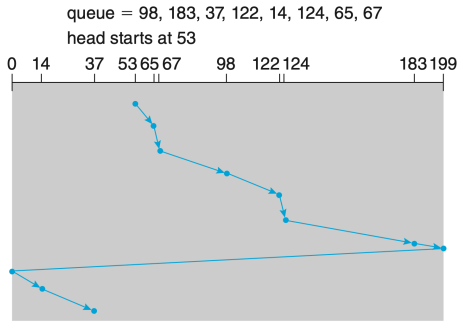


Fig. 11.8 in OSC book

Storage Device Management

- *Low-level formatting*, or *physical formatting* ⇒ dividing a disk into sectors that the disk controller can read and write
  - Each sector can hold header information + data + *error correction code* (ECC)
  - Usually 512 bytes of data but can be selectable
- To use a disk to hold files, OS still needs to record its own data structures on the disk
  - *Partition* the disk into one or more groups of cylinders ⇒ each treated as a *logical disk*
  - *Logical formatting* ⇒ making a file system
  - To increase efficiency, most FSs group blocks into *clusters* ⇒ disk I/O done in blocks, file I/O done in clusters

Storage Device Management (cont.)

- *Root partition* contains the OS ⇒ other partitions can hold other OSs, other FSs, or be *raw* (e.g., for efficient DB usage)
- When the computer starts ⇒ use *bootstrap loader* from ROM/firmware, which then loads the full bootstrap program from the secondary storage
- *Boot block* can point to boot volume, *boot loader* set of blocks (⇒ code for loading the OS kernel), or *boot management program*
- *Swap space management* ⇒ can be in raw partition or a file within a file system (for convenience of adding)

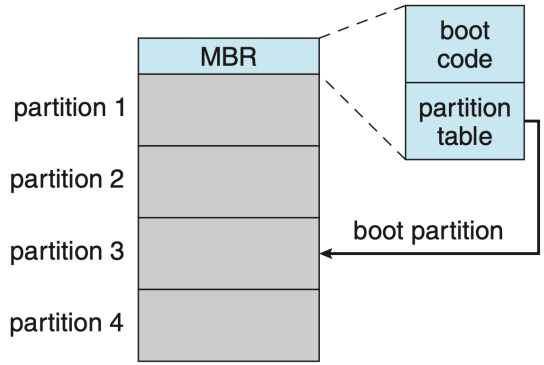


Fig. 11.10 in OSC book

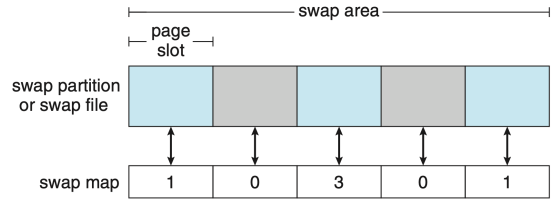


Fig. 11.11 in OSC book

RAID Structure

- *RAID* ⇒ redundant array of independent (or "inexpensive") disks
- Multiple disk drives provide *reliability* via *redundancy* ⇒ increases the mean time to failure
- Disk *striping* uses a group of disks as one storage unit
- Implementation can be hardware- , firmware and driver- , or software-based
- Multiple RAID schemes (*levels*):
  - RAID 0 ⇒ striping, but no redundancy
  - RAID 1 (*mirroring* or*shadowing*) ⇒ keeps duplicate of each disk
  - RAID 1+0 (striped mirrors) or RAID 0+1 (mirrored stripes) ⇒ provides high performance and high reliability
  - RAID 4, 5, 6 (*block interleaved parity*) ⇒ uses much less redundancy
  - Multidimensional RAID 6 ⇒ RAID 6 implemented in several directions

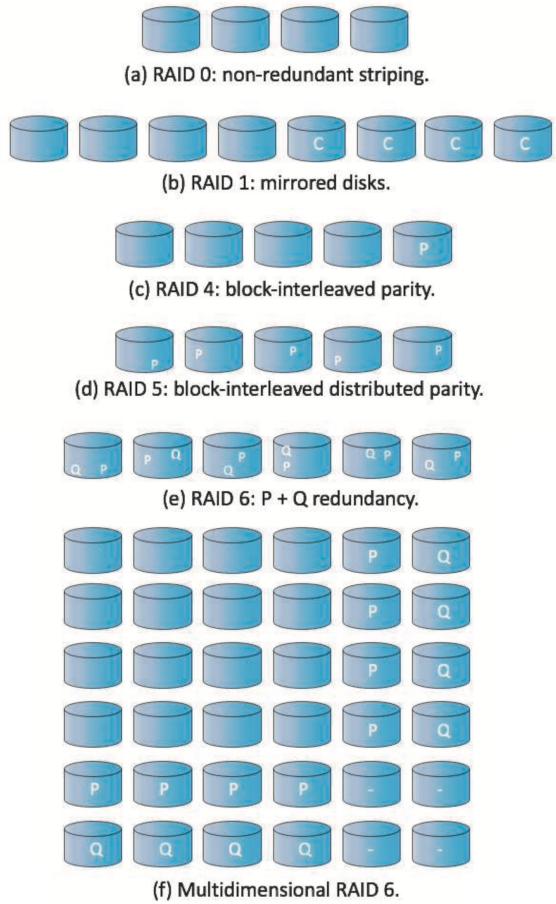
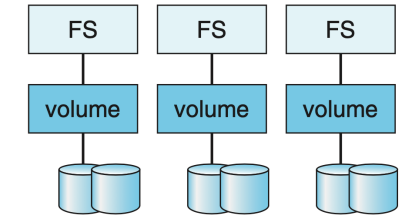


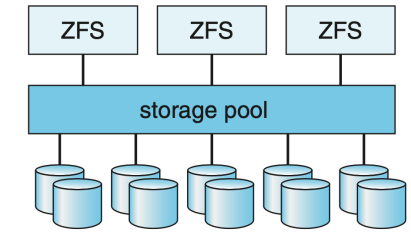
Fig. 11.15 in OSC book

Alternatives to RAID

- RAID alone does not prevent or detect data corruption or other errors, just disk failures
- Modern file systems provide software-based alternatives to many of RAID features
- ZFS (supported on Solaris, Linux, FreeBSD...)
  $\Rightarrow$ combines file-system management and volume management, with disks organized in *pools*
- ZFS also maintains internal checksums of all blocks, including data and metadata
- Also supports *snapshots*, *copy on write* (CoW), and so on
- Other options: LVM (logical volume management), Btrfs ("better FS"; actually, " b-tree file system"), ...



(a) Traditional volumes and file systems.



(b) ZFS and pooled storage.

Fig. 11.18 in OSC book

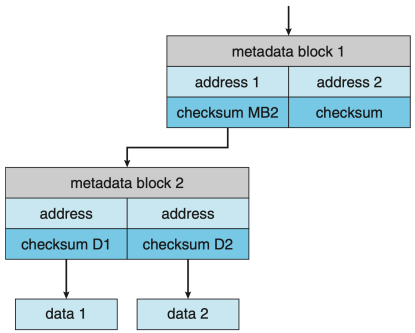


Fig. 11.17 in OSC book

Agenda

- Motivation and Introduction
- Storage Devices
- Storage Device Support
- I/O Support
- Summary

I/O Hardware

- *Port* ⇒ connection point for device
- *Bus* ⇒ daisy chain or shared direct access
  - *PCI* / *PCIe* bus common in modern PCs and servers
  - Expansion bus connects relatively slow devices
- *Controller* (*host adapter*) ⇒ electronics that operate port, bus, device
- I/O instructions control devices
- Devices usually have *registers* (typically 1–4 bytes long) where device driver places commands, addresses, and data to write, or read data from registers after command execution
- Devices have I/O port addresses, used by
  - Direct I/O instructions
  - *Memory-mapped I/O* ⇒ especially for large address spaces (graphics)

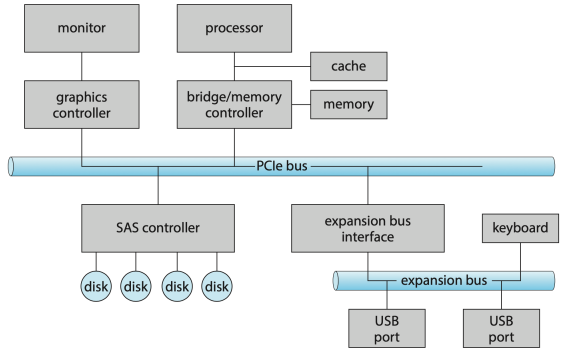


Fig. 12.1 in OSC book

I/O address range (hexadecimal)	device
000–00F	DMA controller
020–021	interrupt controller
040–043	timer
200–20F	game controller
2F8–2FF	serial port (secondary)
320–32F	hard-disk controller
378–37F	parallel port
3D0–3DF	graphics controller
3F0–3F7	diskette-drive controller
3F8–3FF	serial port (primary)

Fig. 12.2 in OSC book

Interrupts

- One way of checking for the data on controller is *polling* ⇒ loop at the host system (CPU) that repeatedly checks the status
- More efficient approach is *interrupts* ⇒ controller should notify the CPU when ready
- *CPU Interrupt-request line* triggered by I/O device ⇒ checked by processor after each instruction; typically two lines, one for *maskable* and one for *nonmaskable* interrupts
- *Interrupt handler* receives interrupts ⇒ *maskable* to ignore or delay some interrupts
- *Interrupt vector* to dispatch interrupt to the correct handler
- Interrupt mechanism also used for *exceptions* ⇒ terminate process, hardware error. . .
- System call executes via *trap* to trigger kernel to execute request
- Efficient interrupt management is important!

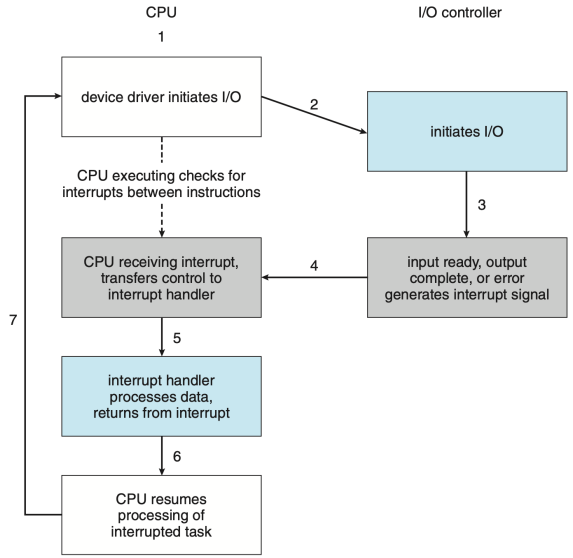


Fig. 12.3 in OSC book

```
Fri Nov 25 13:55:59                                    0:00:10
                        SCHEDULER      INTERRUPTS
------------------------------------------------------------
total_samples                13           22998

delays <  10 usecs           12           16243
delays <  20 usecs            1            5312
delays <  30 usecs            0             473
delays <  40 usecs            0             590
delays <  50 usecs            0              61
delays <  60 usecs            0             317
delays <  70 usecs            0               2
delays <  80 usecs            0               0
delays <  90 usecs            0               0
delays < 100 usecs            0               0
total  < 100 usecs           13           22998
```

Fig. 12.4 in OSC book

Direct Memory Access (DMA)

- DMA used to avoid programmed I/O (one byte at a time) for large data movement
- Requires hardware support ⇒ a DMA controller
- Bypasses CPU to transfer data directly between I/O device and memory

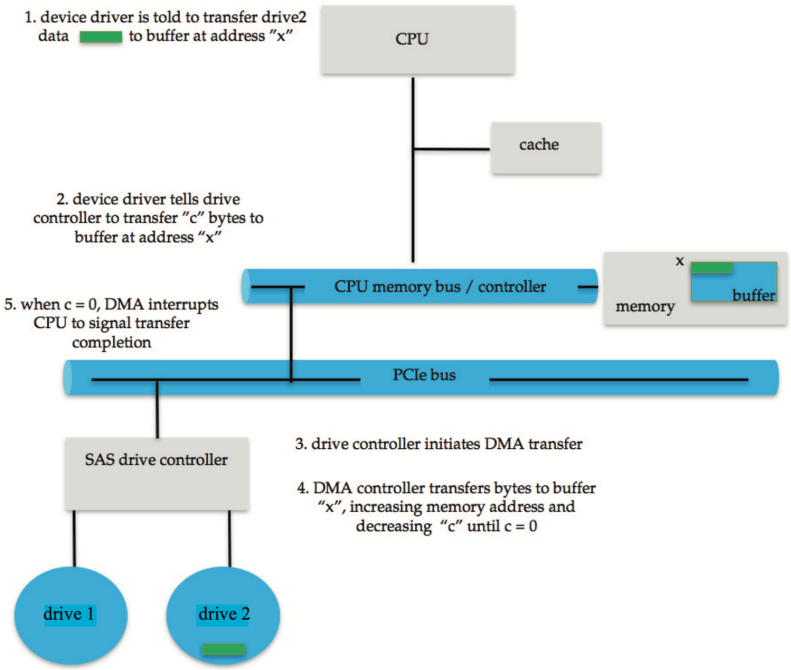


Fig. 12.6 in OSC book

OS I/O Interfaces

- I/O system calls encapsulate device behaviors in generic classes: block I/O (disk drives), character I/O (stream; e.g., keyboards, serial ports. . .), memory-mapped file access, and network sockets
- Device-driver layer hides differences among I/O controllers from kernel ⇒ new devices using existing protocols need no extra work!
- Main strategies for I/O:
  - *Blocking* ⇒ process suspended until I/O completed; easy to implement, but inefficient for some tasks
  - *Nonblocking* ⇒ I/O calls return as much as possible; implemented with multi-threading, used for UI, etc.
  - *Asynchronous* ⇒ process runs while I/O executes, and I/O subsystem signals process when I/O completed; efficient, but more difficult to use!

aspect	variation	example
data-transfer mode	character block	terminal disk
access method	sequential random	modem CD-ROM
transfer schedule	synchronous asynchronous	tape keyboard
sharing	dedicated sharable	tape keyboard
device speed	latency seek time transfer rate delay between operations	
I/O direction	read only write only read–write	CD-ROM graphics controller disk

Fig. 12.8 in OSC book

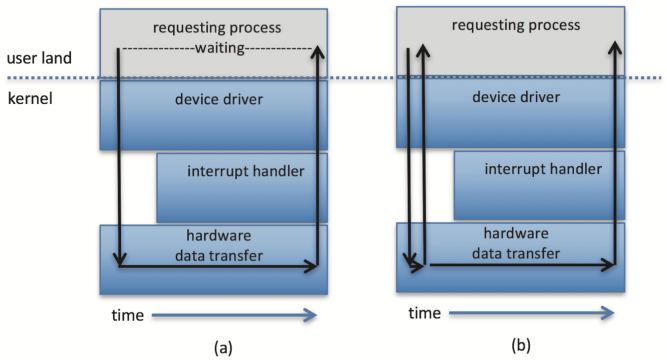


Fig. 12.9 in OSC book

Kernel I/O Subsystem

- Kernels provide many services related to I/O, including: scheduling, buffering, caching, spooling, device reservation, and error handling
- *Device-status table* ⇒ an entry for each I/O device
- *Buffering* ⇒ store data in memory while transferring between devices
- *Caching* ⇒ faster device holding copy of data
- *Spooling* ⇒ hold output for a device if device can serve only one request at a time (e.g., a printer)
- *Device reservation* ⇒ provides exclusive access to a device

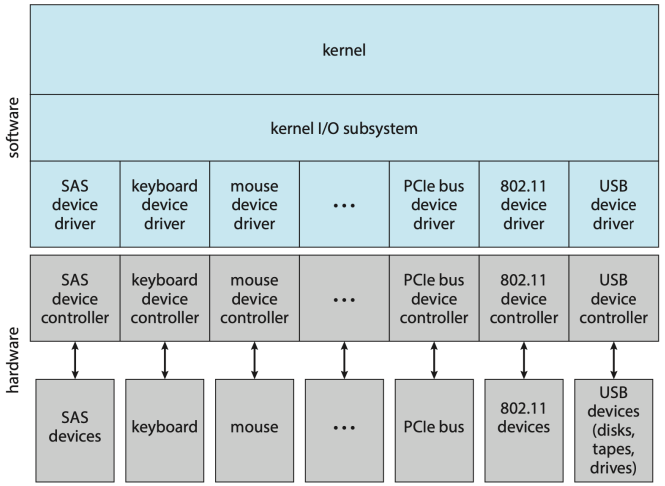


Fig. 12.7 in OSC book

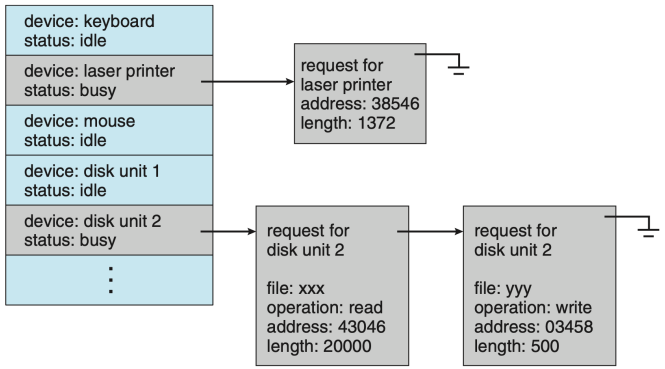


Fig. 12.10 in OSC book

I/O Request Lifecycle

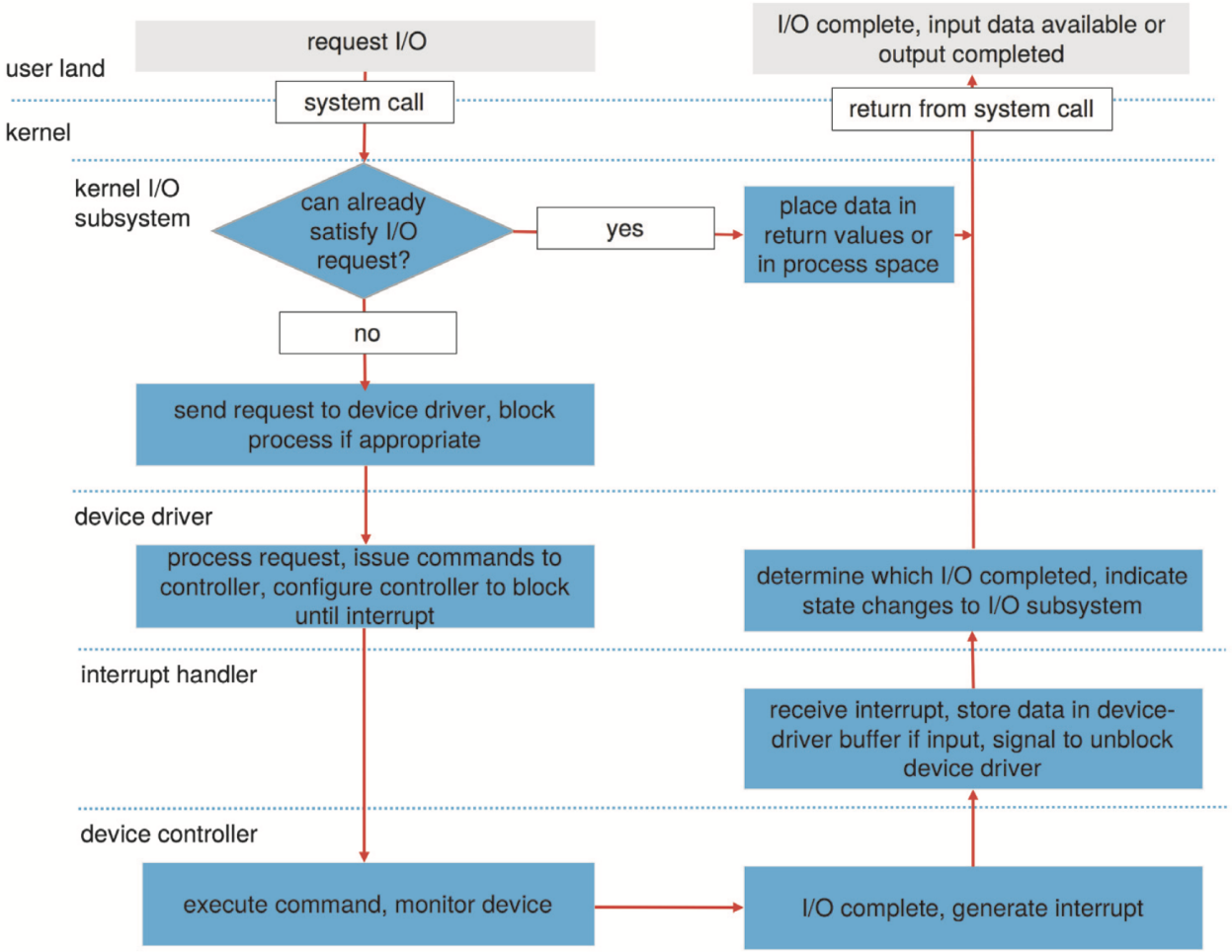


Fig. 12.14 in OSC book

Agenda

- Motivation and Introduction
- Storage Devices
- Storage Device Support
- I/O Support
- Summary

Summary

- Modern secondary storage (typically HDD or SSD) is structured as large one-dimensional arrays of logical blocks $\Rightarrow$ FS and virtual memory management refer to such logical block addresses
- While disc-scheduling algorithms can improve performance for HDDs, there are often no real benefits for SSDs
- RAID algorithms can be used to provide combined operation and redundancy for secondary storage
- The basic hardware elements involved in I/O are buses, device controllers, and the devices themselves
- The kernel module that controls a device is a device driver
- The kernel I/O subsystem provides I/O scheduling, buffering, caching, spooling, device reservation, error handling, name translation, etc.
- I/O system calls are costly due to many layers of software between the application and the physical device