Devices and I/O

key concepts

device registers, device drivers, program-controlled I/O, DMA, polling, disk drives, disk head scheduling

reading

Three Easy Pieces: Chapters 36-37

Sys/161 Device Examples

- timer/clock - current time, timer, beep
- disk drive - persistent storage
- serial console - character input/output
- text screen - character-oriented graphics
- network interface - packet input/output

Device Register Example: Sys/161 timer/clock

Offset	Size	Type	Description
0	4	status	current time (seconds)
4	4	status	current time (nanoseconds)
8	4	command	restart-on-expiry
12	4	status and command	interrupt (reading clears)
16	4	status and command	countdown time (microseconds)
20	4	command	speaker (causes beeps)

Device Register Example: Serial Console

Offset	Size	Type	Description
0	4	command and data	character buffer
4	4	status	writeIRQ
8	4	status	readIRQ

Device Drivers

- a device driver is a part of the kernel that interacts with a device
- example: write character to serial output device

```
// only one writer at a time
P(output device write semaphore)
// trigger the write operation
write character to device data register
repeat {
    read writeIRQ register
} until status is ``completed''
// make the device ready again
write writeIRQ register to ack completion
V(output device write semaphore)
```

- this example illustrates *polling*: the kernel driver repeatedly checks device status

Using Interrupts to Avoid Polling

Device Driver Write Handler:

```
// only one writer at a time
P(output device write semaphore)
// trigger write operation
write character to device data register
```

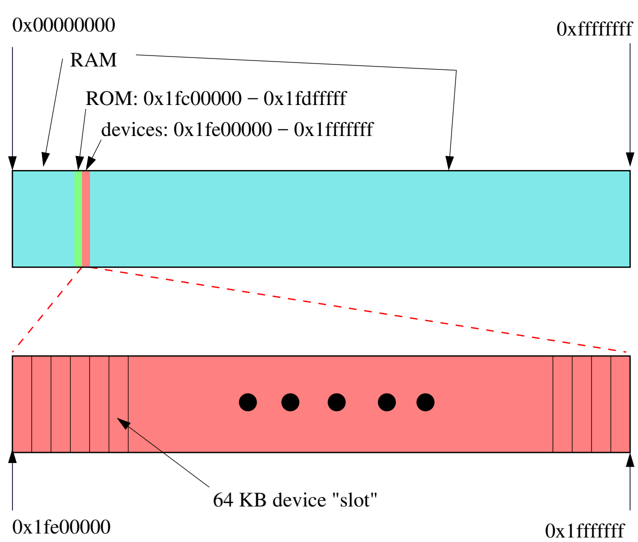
Interrupt Handler for Serial Device:

```
// make the device ready again
write writeIRQ register to ack completion
V(output device write semaphore)
```

Accessing Devices

- how can a device driver access device registers?
- Option 1: special I/O instructions
  - such as `in` and `out` instructions on x86
  - device registers are assigned "port" numbers
  - instructions transfer data between a specified port and a CPU register
- Option 2: memory-mapped I/O
  - each device register has a physical memory address
  - device drivers can read from or write to device registers using normal load and store instructions, as though accessing memory

MIPS/OS161 Physical Address Space

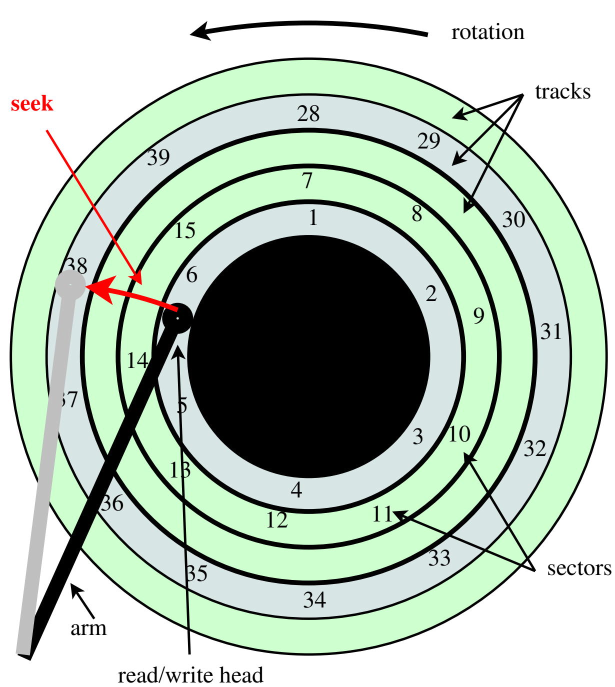


Each device is assigned to one of 32 64KB device "slots". A device's registers and data buffers are memory-mapped into its assigned slot.

Logical View of a Disk Drive

- disk is an array of numbered blocks (or sectors)
- each block is the same size (e.g., 512 bytes)
- blocks are the unit of transfer between the disk and memory
  - typically, one or more contiguous blocks can be transferred in a single operation
- storage is *non-volatile*, i.e., data persists even when the device is without power

A Disk Platter's Surface



Cost Model for Disk I/O

- moving data to/from a disk involves:

  seek time: move the read/write heads to the appropriate cylinder
  - depends on distance (in tracks) between previous request and current request - called the *seek distance*

  rotational latency: wait until the desired sectors spin to the read/write heads
  - depends on the rotational speed of the disk

  transfer time: wait while the desired sectors spin past the read/write heads
  - depends on the rotational speed of the disk and the amount of data being read/written
- request service time is the *sum* of seek time, rotational latency, and transfer time

Seek, Rotation, and Transfer

- Seek time:
  - If the next request is for data on the same track as the previous request, seek distance and seek time will be zero.
  - In the worst case, e.g., seek from outermost track to innermost track, seek time may be 10 milliseconds or more.
- Rotational Latency:
  - Consider a disk that spins at 12,000 RPM
  - One complete rotation takes 5 millseconds.
  - Rotational latency ranges from 0ms to 5ms.
- Transfer Time:
  - Once positioned, the 12,000 RPM disk can read/write all data on a track in one rotation (5ms)
  - If only X% of the track's sectors are being read/written, transfer time will be X% of the complete rotation time (5ms).

Performance Implications of Disk Characteristics

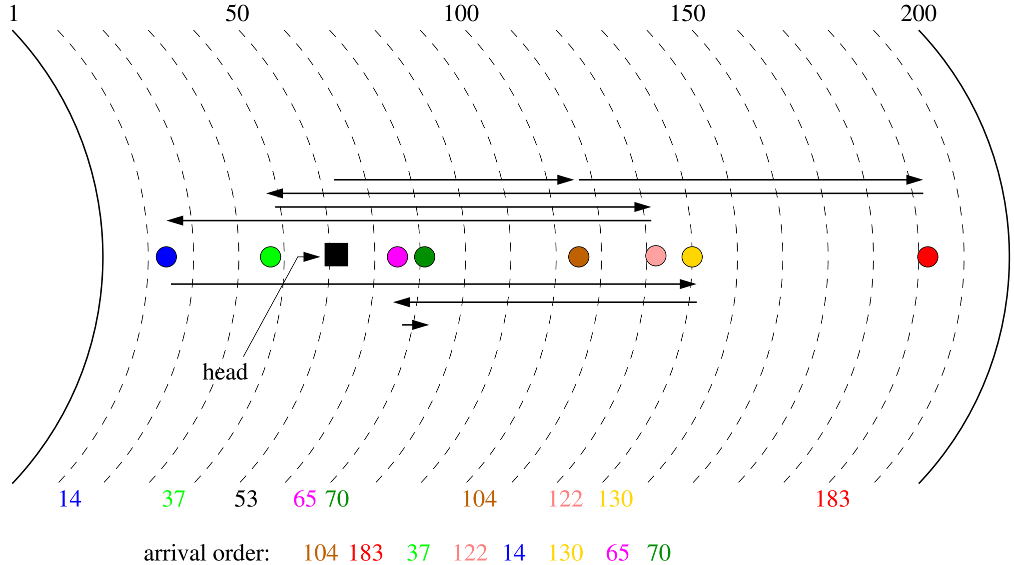
- larger transfers to/from a disk device are *more efficient* than smaller ones. That is, the cost (time) per byte is smaller for larger transfers. (Why?)
- sequential I/O is faster than non-sequential I/O
  - sequential I/O operations eliminate the need for (most) seeks

Disk Head Scheduling

- goal: reduce seek times by controlling the order in which requests are serviced
- disk head scheduling may be performed by the device, by the operating system, or both
- for disk head scheduling to be effective, there must be a queue of outstanding disk requests (otherwise there is nothing to reorder)
- an on-line approach is required: new I/O requests may arrive at any time

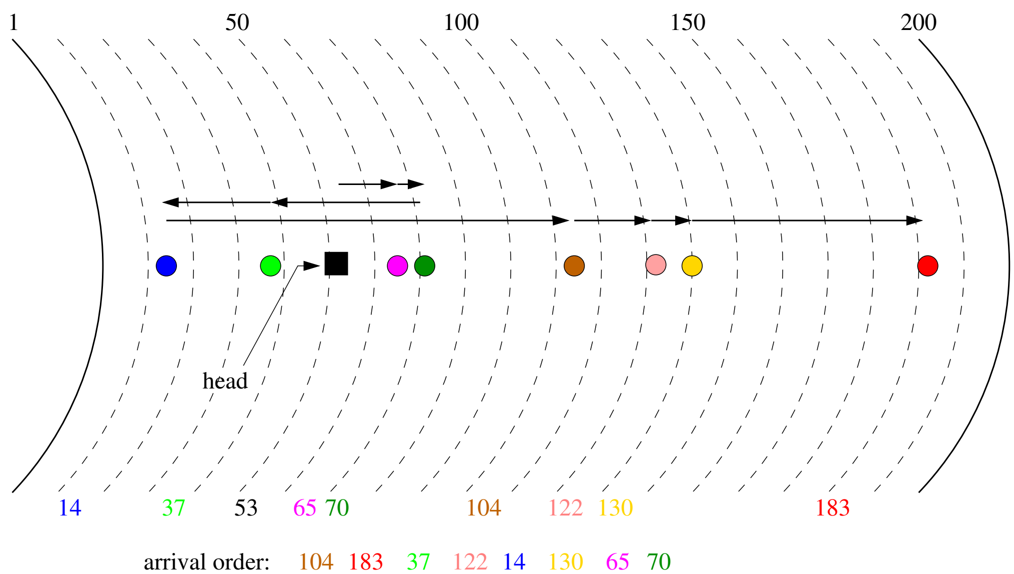
FCFS Disk Head Scheduling

- handle requests in the order in which they arrive
- fair and simple, but no optimization of seek times



Shortest Seek Time First (SSTF)

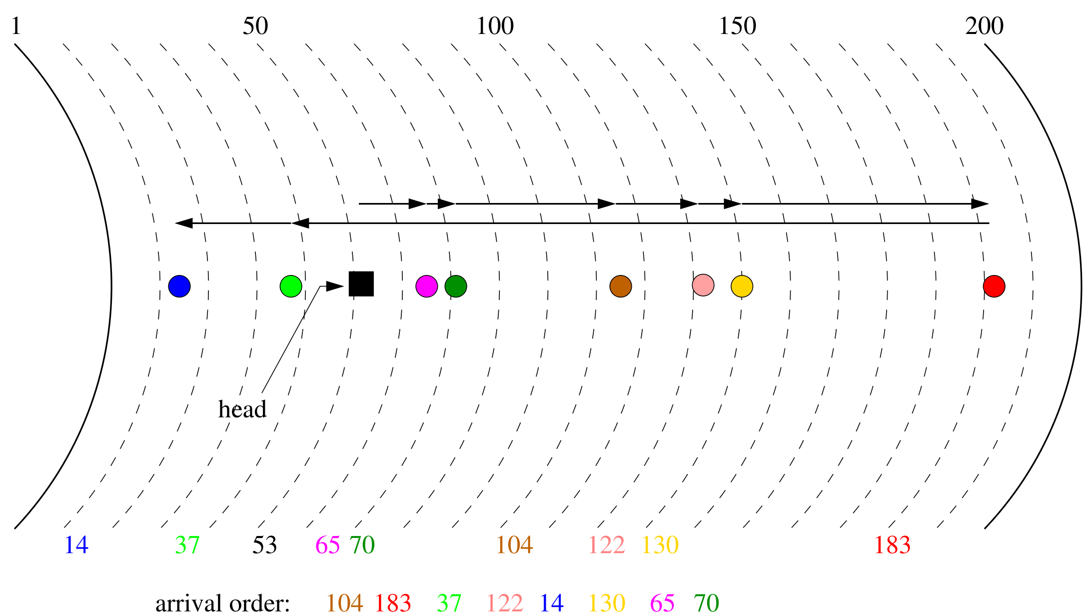
- choose closest request (a greedy approach)
- seek times are reduced, but requests may starve



Elevator Algorithms (SCAN)

- Under SCAN, aka the elevator algorithm, the disk head moves in one direction until there are no more requests in front of it, then reverses direction.
- there are many variations on this idea
- SCAN reduces seek times (relative to FCFS), while avoiding starvation

SCAN Example



Data Transfer To/From Devices

- Option 1: *program-controlled I/O*
  The device driver moves the data between memory and a buffer on the device.
  - Simple, but the CPU is *busy* while the data is being transferred.
- Option 2: *direct memory access (DMA)*
  - The device itself is responsible for moving data to/from memory. CPU is *not busy* during this data transfer, and is free to do something else.

Sys/161 disks do program-controlled I/O.

Device Register Example: Sys/161 disk controller

Offset	Size	Type	Description
0	4	status	number of sectors
4	4	status and command	status
8	4	command	sector number
12	4	status	rotational speed (RPM)
32768	512	data	transfer buffer

Writing to a Sys/161 Disk

Device Driver Write Handler:

```
// only one disk request at a time
P(disk semaphore)
copy data from memory to device transfer buffer
write target sector number to disk sector number register
write ``write'' command to disk status register
// wait for request to complete
P(disk completion semaphore)
V(disk semaphore)
```

Interrupt Handler for Disk Device

```
// make the device ready again
write disk status register to ack completion
V(disk completion semaphore)
```


Reading From a Sys/161 Disk

Device Driver Read Handler:

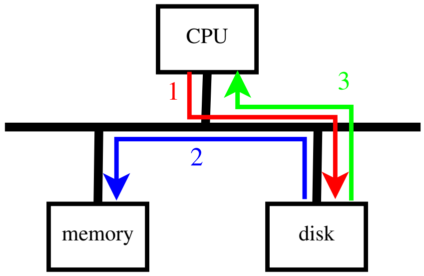
```
// only one disk request at a time
P(disk semaphore)
write target sector number to disk sector number register
write ``read'' command to disk status register
// wait for request to complete
P(disk completion semaphore)
copy data from device transfer buffer to memory
V(disk semaphore)
```

Interrupt Handler for Disk Device

```
// make the device ready again
write disk status register to ack completion
V(disk completion semaphore)
```

Direct Memory Access (DMA)

- DMA is used for block data transfers between devices (e.g., a disk) and memory
- Under DMA, the CPU initiates the data transfer and is notified when the transfer is finished. However, the device (not the CPU) controls the transfer itself.



1. CPU issues DMA request to device
2. device directs data transfer
3. device interrupts CPU on completion

Solid State Drives(SSD)

- no mechanical parts; use integrated circuits for persistant storage instead of magnetic surfaces
- DRAM: requires constant power to keep values
  - transistors with capacitors
  - capacitor holds microsecond charge; periodically refreshed by primary power
- Flash Memory: traps electrons in quantum cage
  - floating gate transistors
  - usually NAND (not-and gates)

SSD Data Arrangement

- logically divided into blocks and pages
  - 2, 4 or 8KB pages
  - 32KB-4MB blocks
- reads/writes at page level
  - pages are initialized to 1s; can transition $1 \rightarrow 0$ at page level (i.e., write new page)
  - a high voltage is required to switch $0 \rightarrow 1$ (i.e., overwrite/delete page)
  - cannot apply high voltage at page level, only to blocks
    - $*$ overwriting/deleting data must be done at the block level

Writing and Deleting from Flash Memory

- Naive Solution (slow):
  - read whole block into memory
  - re-initialize block (all page bits back to 1s)
  - update block in memory; write back to SSD
- SSD controller handles requests (faster):
  - mark page to be deleted/overwritten as invalid
  - write to an unused page
  - update translation table
  - requires garbage collection

Wear Leveling

- SSDs are not impervious
- blocks have limited number of write cycles
  - if block is no longer writeable; it becomes ready-only
  - when a certain % of blocks are read-only; disk becomes read-only
- SSD controller wear-levels; ensuring that write cycles are evenly spread across all blocks

Defragmentation

- defragmentation takes files spread across multiple, non-sequential pages and makes them sequential
  - it re-writes many pages of memory, possibly several times
  - SSD random and sequential access have approximately the same cost
    - no clear advantage to defragmenting
    - extra, unnecessary writes performed by defragmenting—causes pre-mature disk aging