Review

- Input / Output
  - Types of I/O Devices – block, character device
  - I/O Device Structure – electrical, mechanical components, device driver
  - Device Controllers
  - How CPU communicate with Device controller
    - Memory-Mapped I/O – Indirect
    - Non-Memory-Mapped I/O - Direct
    - Hybrid
  - Direct Memory Access (DMA)

Preview

- Principle of I/O Software
  - Goal of I/O Software
  - Implementation of I/O
    - Programmed I/O
    - Interrupt-Driven I/O
    - I/O using DMA
- Input /Output Software Layer
  - Interrupt Handler
  - Device Driver
- HDD

Principle of I/O Software
(Goal of I/O Software)

- **Device independent**
  - I/O management software must be able to access any I/O devices .
    ex) I/O software should be able to read a file from a hard disk, CD-ROM, DVD or USB stick without modifying the program for each devices.
  - It is up to operating system to take care of the problems caused by different device requires different command sequence to read or write (i.e. non-memory I/O need device dependent instruction set for device driver)

Principle of I/O Software
(Goal of I/O Software)

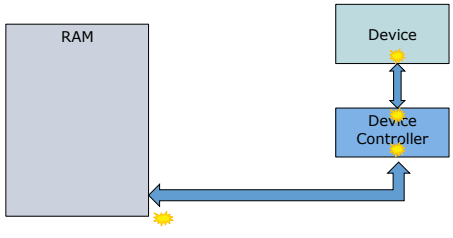
- **Uniform Naming**
  - Name of a file or device should be named with a string (path) or an integer, not depend on the types of devices.

Ex) In UNIX or LINUX, all disks can be integrated in the file system hierarchy.
A USB stick can be mounted on the top of a directory (ex. /usr/ast/backup). So a file can copy from any disk to the USB stick with path name.

Principle of I/O Software
(Goal of I/O Software)

- **Error Handling**
  - Error must be handled as close to the hardware possible.
  - That means I/O error must detected by controller or device itself.
  - Operating System only take care upper level error for I/O access when arrived to memory.

Principle of I/O Software
(Goal of I/O Software)



Principle of I/O Software
(Goal of I/O Software)

- Synchronous vs. Asynchronous Transfer
  - **Synchronous Transfer** – CPU is blocked until finish transfer data from a device to device.
  - **Asynchronous Transfer** (interrupt driven transfer). Once transfer start between devices, CPU can take care other process until interrupt arrive (ex. DMA).

Principle of I/O Software
(Goal of I/O Software)

- Buffering
  - For some devices, data cannot be stored directly in its final destination.
  - It must be saved in a buffer and check error or decoded proper form then send to destination.

    Ex) OS cannot directly take care data comes in off the network until it stored in buffer and examined it.

Principle of I/O Software
(Goal of I/O Software)

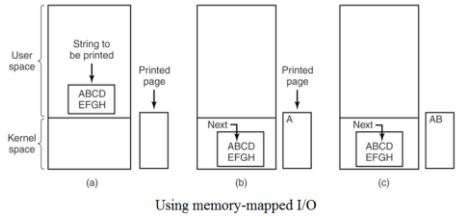
- Sharable vs. Dedicated Device I/O Software
  - Hard disk is able to share with multiple processes (multiple read/write heads for each disk)
  - Tape Drive – have to be dedicated to a single user until that user is finished

Principle of I/O Software

- Three Fundamental ways that I/O can be performed.
  - Programmed I/O – CPU is blocked until an I/O operation is completed
  - Interrupt-Driven I/O – CPU blocks for each step of a job
  - I/O using DMA – CPU inform to DMA, CPU works for other job. When the job is done DMA inform CPU. CPU continue for the job.

Principle of I/O Software
(Programmed I/O)

- Programmed I/O – tying up the CPU full time until all the I/O is done



Using memory-mapped I/O

Principle of I/O Software
(Programmed I/O)

```
copy_from_user(buffer, p, count);          /* p is the kernel bufer */
for (i = 0; i < count; i++) {              /* loop on every character */
    while (*printer_status_reg != READY) ; /* loop until ready */
    *printer_data_register = p[i];         /* output one character */
}
return_to_user( );
```

- Programmed I/O is simple but had the disadvantage of tying up the CPU full time until all the I/O is done.

Principle of I/O Software
(Interrupt-Driven I/O)

- If a printer takes 10ms to print one character, the CPU will sit in idle for 10ms waiting to be allowed to print the next character.
- Interrupt-Driven I/O : the way let the CPU to do something else while waiting for the I/O device to becomes ready.

Principle of I/O Software
(Interrupt-Driven I/O)

```
copy_from_user(buffer, p, count);
enable_interrupts( );
while (*printer_status_reg != READY) ;
*printer_data_register = p[0];
scheduler( );
```

(a)

```
if (count == 0) {
    unblock_user( );
} else {
    *printer_data_register = p[i];
    count = count – 1;
    i = i + 1;
}
acknowledge_interrupt( );
return_from_interrupt( );
```

(b)

Principle of I/O Software
(I/O with DMA)

- Disadvantage of Interrupt-Driven I/O is that an interrupt occurs on every character -> wasting CPU time
- A solution is DMA.
  - The CPU initiate an I/O
  - DMA take over the I/O job.
  - Once I/O finish, DMA issue acknowledge interrupt.
  - OS unblock the process waiting for I/O

Principle of I/O Software
(I/O with DMA)

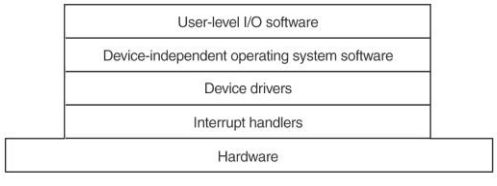
```
copy_from_user(buffer, p, count);
set_up_DMA_controller( );
scheduler( );
```

(a)

```
acknowledge_interrupt( );
unblock_user( );
return_from_interrupt( );
```

(b)

Input / Output Software Layer



Input / Output Software Layer
(Interrupt Handler)

- An **interrupt handler**, also known as an **interrupt service routine** (**ISR**), is a subroutine in an operating system or device driver whose execution is triggered by the reception of an interrupt.
- Interrupt handlers have a multitude of functions, which vary based on the reason the interrupt was generated and the speed at which the Interrupt Handler completes its task.
- For example, pressing a key on a computer keyboard, or moving the mouse, triggers interrupts that call interrupt handlers which read the key, or the mouse's position, and copy the associated information into the computer's memory.

Input / Output Software Layer
(Interrupt Handler)

Step for Interrupt handler

1. Save any registers currently running process
2. Set up a context for interrupt service procedure
3. Set up a stack for the interrupt service procedure
4. Acknowledge the interrupt controller
5. Copy the registers
6. Run the interrupt service procedure
7. Select which process to run next
8. Set up MMU context for the process to run next
9. Load the new process's register
10. Start running the new process.

Input / Output Software Layer
(Device Driver)

Device Driver

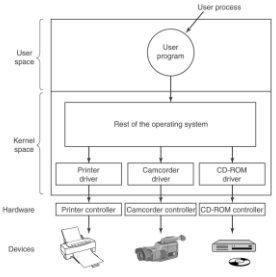
- Since each I/O device has different roles, number of registers and commands used in different I/O devices are different.
- A device driver is a device-specific code for controlling an specific I/O devices.
- A device driver is written by the device's manufacturer.
- A device driver is the interface between OS and a I/O controller.

Input / Output Software Layer
(Device Driver)

Device Driver (Continue)

- In order to access the device's hardware, each of device deriver has to be part of the operating system kernel.
- But, it is still possible to construct a driver which is run in user's space, with system calls for reading and writing the device registers (micro-kernel).
- But, Modern OS expect drivers to run in kernel's space.
- OS designer need to consider an architecture that will allows driver installation in it.

Input / Output Software Layer
(Device Driver)

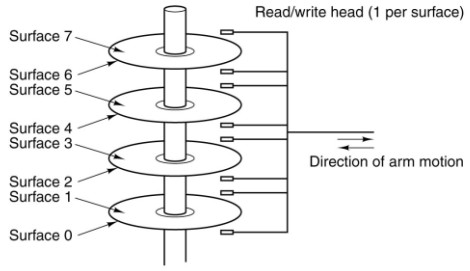


Input / Output Software Layer
(Device Driver)

- OS define a standard interface for block devices and for character devices.
- **Standard interfaces** – a number of procedures that the rest of the OS can call to get the deriver to do work for it.
- An OS with a single binary program (UNIX) that contain all of device driver need to be recompile if new driver is added.
- In PC, OS went over to a model where drivers were dynamically loaded into the system during execution.
- Different system use different way to load drivers.

Hard Disk Drive

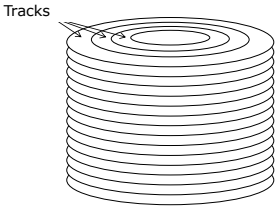
Hard Disk Drive



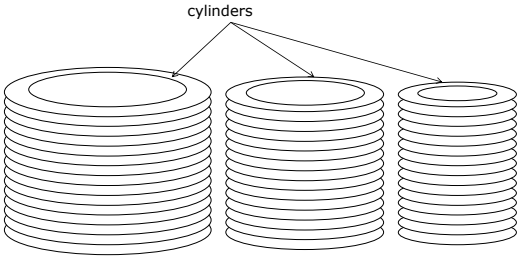
Hard Disk Drive

- Modern disk drives are addressed as large one-dimensional arrays of logical blocks.
- The size of a logical block is usually 512 Bytes.
- At any given arm position, each of heads can read (by rotation) and annular region called a **track**.
- All the tracks for a given arm position forms a **cylinder.**
- Each track is divide into some number of **sector.**
- The one-dimensional array of logical blocks mapped onto the sectors of the disk sequentially.
- We can convert a logical block number into an disk address that consists of a **cylinder number**, a **track number** and a **sector number**.

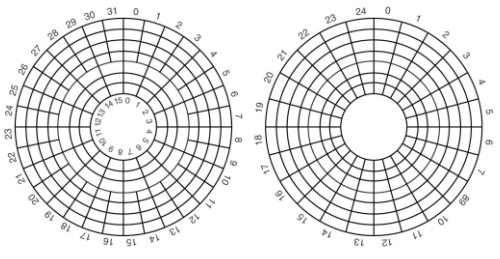
Hard Disk Drive



Hard Disk Drive



Hard Disk Drive



Hard Disk Drive
(Disk Scheduling)

- One of the responsibilities of the operating system is to use the hardware efficiently.
- With disk drivers, operating system try to minimize the disk access time.
- The major component of disk access times
  - **Seek time** –the time for the disk arm to move the heads to the cylinder containing the desired sector.
  - **Rotation time** – to rotate the desired sector to the disk head.
- Disk scheduling algorithm is for scheduling movement of disk arm between cylinders.

Hard Disk Drive
(Disk Scheduling: FCFS)

- FCFS –First Come First Serve

Ex) consider a disk queue with requests for I/O to blocks on cylinders

98, 183, 37, 122, 14, 124, 65, 67.

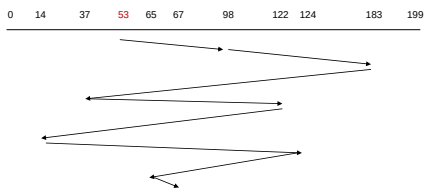
If the disk head is initially at cylinder 53

FCFS : 98, 183, 37, 122, 14, 124, 65, 67

Total Head movement = (98 -53) + (183-98) + (183-37)+ (122-37) + (122-14) + (124-14) + (124-65) + (67-65)

Hard Disk Drive
(Disk Scheduling: FCFS)

98, 183, 37, 122, 14, 124, 65, 67



FCFS disk Scheduling

Hard Disk Drive
(Disk Scheduling: SSTF)

- Shortest-Seek-Time-First (SSTF)– serve close to the current location.

Ex) consider a disk queue with requests for I/O to blocks on cylinders

98, 183, 37, 122, 14, 124, 65, 67.

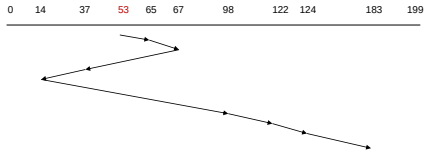
If the disk head is initially at cylinder 53

SSTF : 65, 67, 37, 14, 98, 122, 124, 183

Total Head movement = (65-53) + (67-65) + (67-37) + (37-14) + (98-14) + (122-98) + (124-122) + (183-124)

Hard Disk Drive
(Disk Scheduling: SSTF)

98, 183, 37, 122, 14, 124, 65, 67



SSTF Disk Scheduling

Hard Disk Drive
(Disk Scheduling: SCAN)

- Elevator Scheduling (SCAN) – the disk arm starts at one end of the disk, and moves toward the other end.

Ex) consider a disk queue with requests for I/O to blocks on cylinders

98, 183, 37, 122, 14, 124, 65, 67.

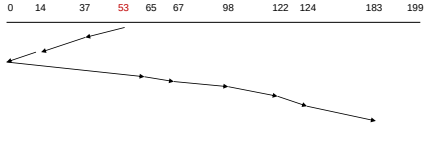
If the disk head is initially at cylinder 53

Elevator1: 37, 14, 0, 65, 67, 98, 122, 124, 183

Total Head movement = (53-37) + (37-14) + (14-0) + (65-0) + (67-65) + (98-67) + (122-98) + (124-122) + (183-124)

Hard Disk Drive
(Disk Scheduling: SCAN)

98, 183, 37, 122, 14, 124, 65, 67



SCAN (Elevator) Disk Scheduling

Hard Disk Drive
(Disk Scheduling: C-SCAN)

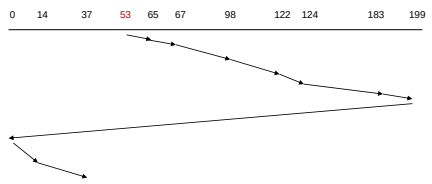
- C-Scan – similar with the Elevator algorithm. Move the head from one end of disk to the other. When the head reaches the other end, it immediately return to the beginning of the disk without servicing any request on the return trip.

Ex) consider a disk queue with requests for I/O to blocks on cylinders
98, 183, 37, 122, 14, 124, 65, 67.

If the disk head is initially at cylinder 53

Hard Disk Drive
(Disk Scheduling: C-SCAN)

98, 183, 37, 122, 14, 124, 65, 67



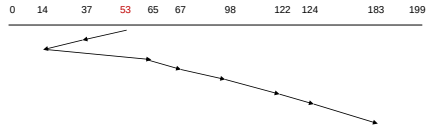
C- SCAN

Hard Disk Drive
(Disk Scheduling: LOOK)

- LOOK – Both Elevator and C-SCAN move the disk arm across the full width of the disk.
- In practice, neither algorithm is implemented this way.
- More commonly, the arm goes only as far as the final request in each direction, then it reverses direction immediately without going all the way to the end of the disk.
- This version of SCAN and C-SCAN algorithm are called LOOK and C-LOOK.

Hard Disk Drive
(Disk Scheduling: LOOK)

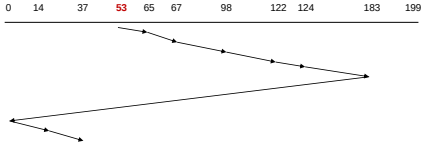
98, 183, 37, 122, 14, 124, 65, 67



LOOK Disk Scheduling

Hard Disk Drive
(Disk Scheduling: C-LOOK)

98, 183, 37, 122, 14, 124, 65, 67



C-LOOK Disk Scheduling

Disk Management

Disk Formatting

- The format consists of a series of concentric tracks, each tack containing sectors, with short gaps between sectors

- **Preamble** – start with bit pattern, the cylinder number, sector number, and other information
- **Data** – size of data portion is determined by formatting program. (usually 512 Byte)
- **ECC** (error correcting code) – information used for error correction.

Disk Management
(Bad Blocks Management)

- Bad block should also be managed by OS some how.
- The MS-DOS (or Window)–
  - *format* command does a logical format,
  - scan the disk to find bad block.
  - If a bad block is found, special value is written into the corresponding file allocation table entry.

Disk Management
(Bad Blocks Management)

Bad Block Management

- SCSI (Small Computer System Interface) disk- used in workstations and servers
  - The disk controller maintains a list of bad blocks on the disk.
  - The list is initialized during the low-level format at the factory
  - It is updated over the life of the disk

Disk Management
(Swap Space Management)

- Swap Space is used in various ways by different OS. Based on memory management algorithm
  - Swapping – need swap entire process space
  - Virtual memory – Paging, Segmentation, Segmentation with paging
- Swap Space Location
  - Swap space can be carved out of the normal file system
  - Swap space can be in a separate partition.