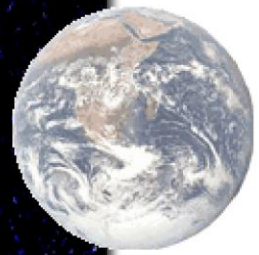
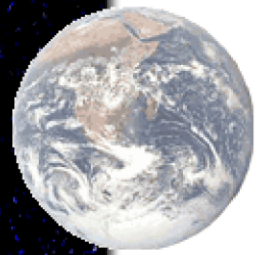
I/O Management and Disk Scheduling

Chapter 11



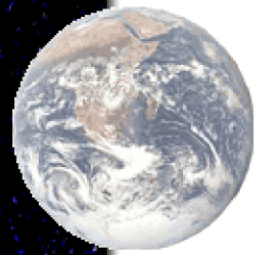
Categories of I/O Devices

- Human readable
  - Used to communicate with the user
  - Printers
  - Video display terminals
    - Display
    - Keyboard
    - Mouse



Categories of I/O Devices

- Machine readable
  - Used to communicate with electronic equipment
  - Disk and tape drives
  - Sensors
  - Controllers
  - Actuators



Categories of I/O Devices

- Communication
  - Used to communicate with remote devices
  - Digital line drivers
  - Modems

Differences in I/O Devices

- Data rate
  - May be differences of several orders of magnitude between the data transfer rates

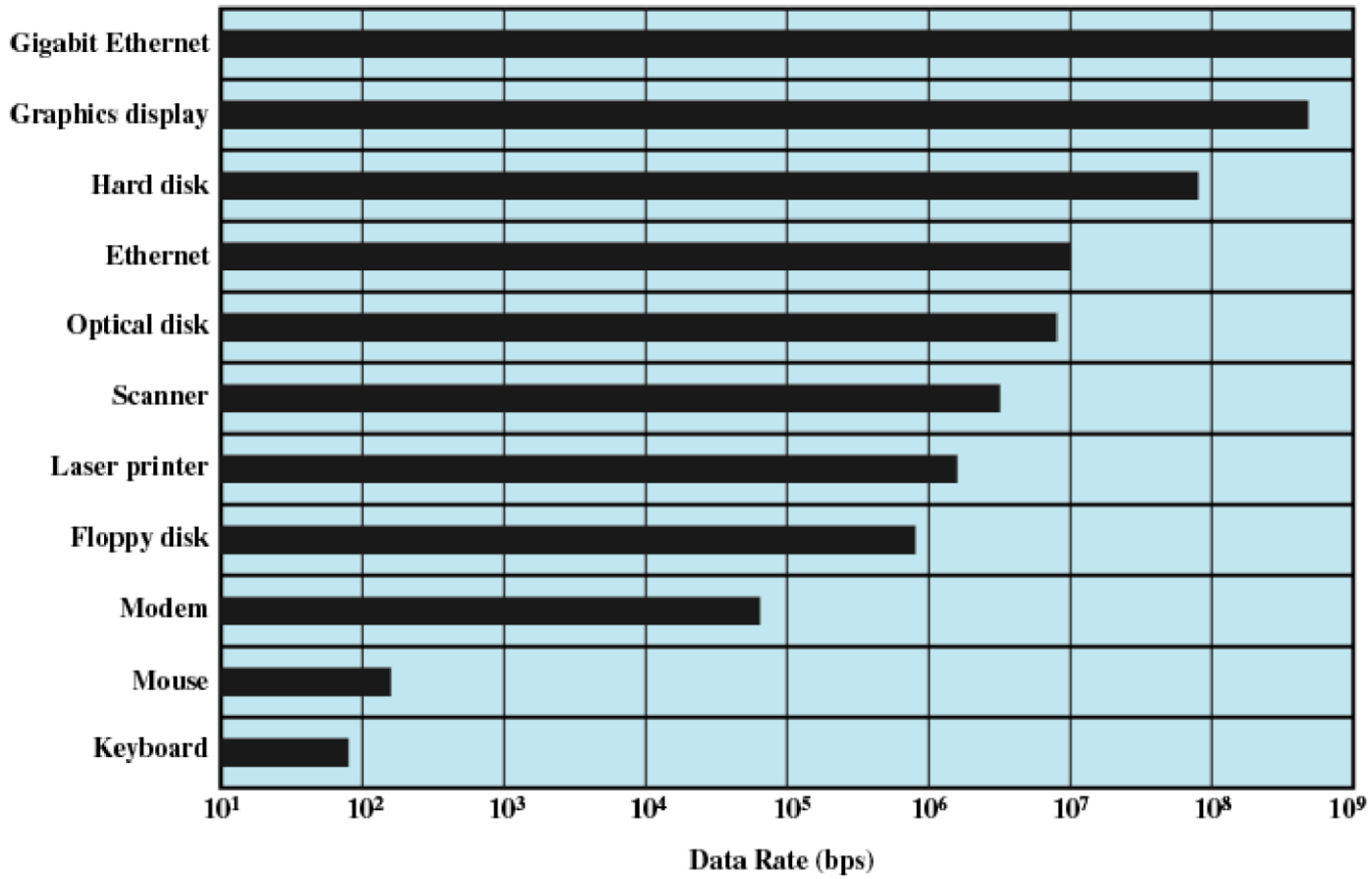


Figure 11.1 Typical I/O Device Data Rates

Differences in I/O Devices

- Application
  - Disk used to store files requires file management software
  - Disk used to store virtual memory pages needs special hardware and software to support it
  - Terminal used by system administrator may have a higher priority

Differences in I/O Devices

- Complexity of control
- Unit of transfer
  - Data may be transferred as a stream of bytes for a terminal or in larger blocks for a disk
- Data representation
  - Encoding schemes
- Error conditions
  - Devices respond to errors differently

Interrupts Revisited

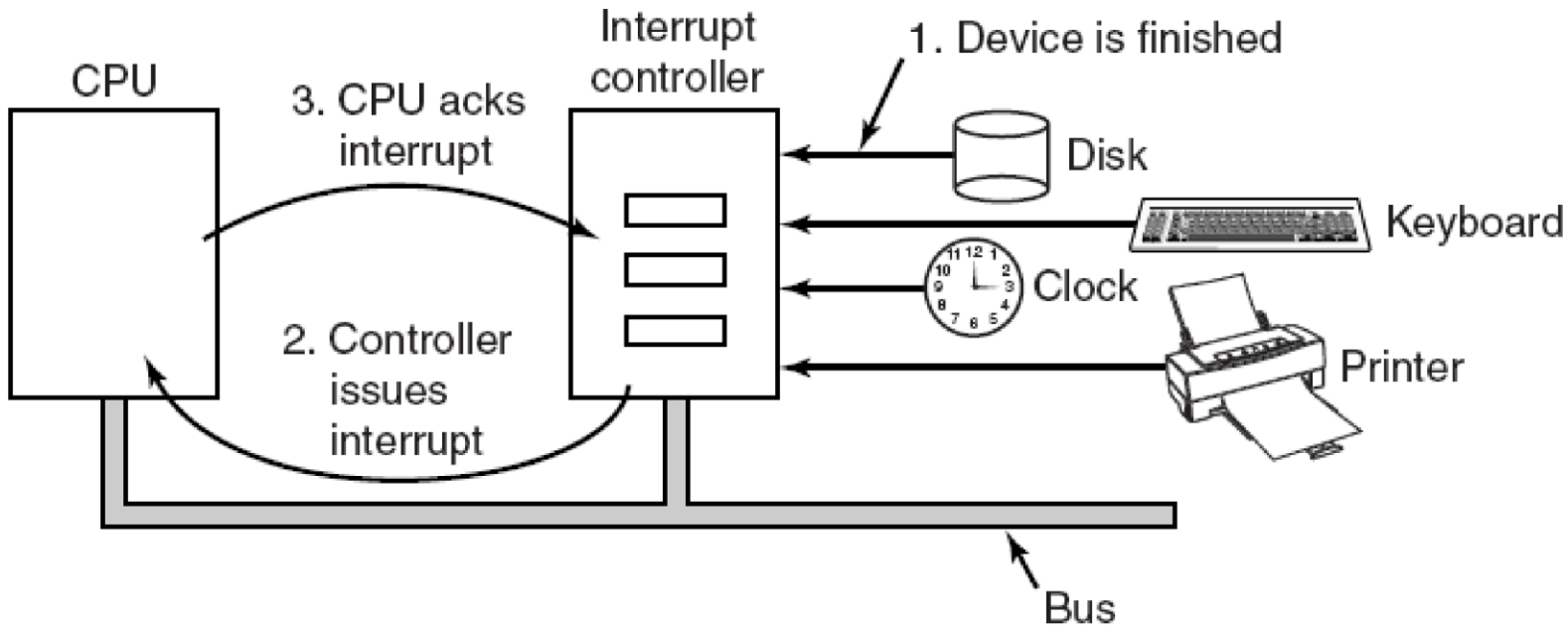
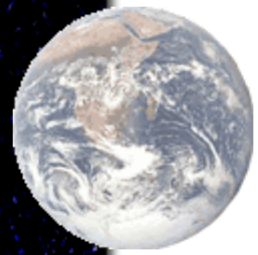


Figure 5-5. How an interrupt happens. The connections between the devices and the interrupt controller actually use interrupt lines on the bus rather than dedicated wires.



Performing I/O

- Programmed I/O
  - Process is busy-waiting for the operation to complete
- Interrupt-driven I/O
  - I/O command is issued
  - Processor continues executing instructions
  - I/O module sends an interrupt when done

Performing I/O

- Direct Memory Access (DMA)
  - DMA module controls exchange of data between main memory and the I/O device
  - Processor interrupted only after entire block has been transferred

Relationship Among Techniques

Table 11.1 I/O Techniques

	No Interrupts	Use of Interrupts
I/O-to-memory transfer through processor	Programmed I/O	Interrupt-driven I/O
Direct I/O-to-memory transfer		Direct memory access (DMA)

Programmed I/O (1)

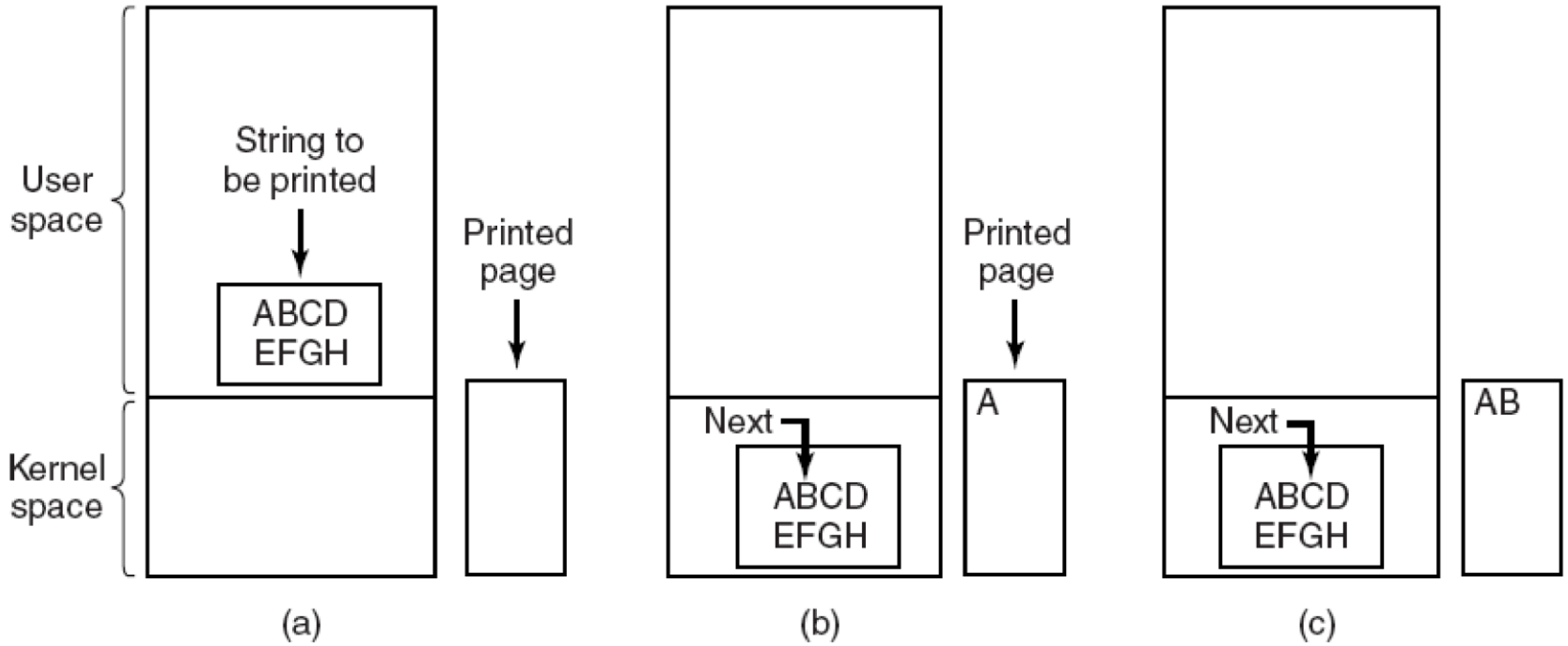


Figure 5-7. Steps in printing a string.

Programmed I/O (2)

```
copy_from_user(buffer, p, count);                    /* p is the kernel buffer */
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( );
```

Figure 5-8. Writing a string to the printer using programmed I/O.

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)

Figure 5-9. Writing a string to the printer using interrupt-driven I/O. (a) Code executed at the time the print system call is made. (b) Interrupt service procedure for the printer.

I/O Using DMA

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

(a)

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

(b)

Figure 5-10. Printing a string using DMA. (a) Code executed when the print system call is made. (b) Interrupt service procedure.

Direct Memory Access (DMA)

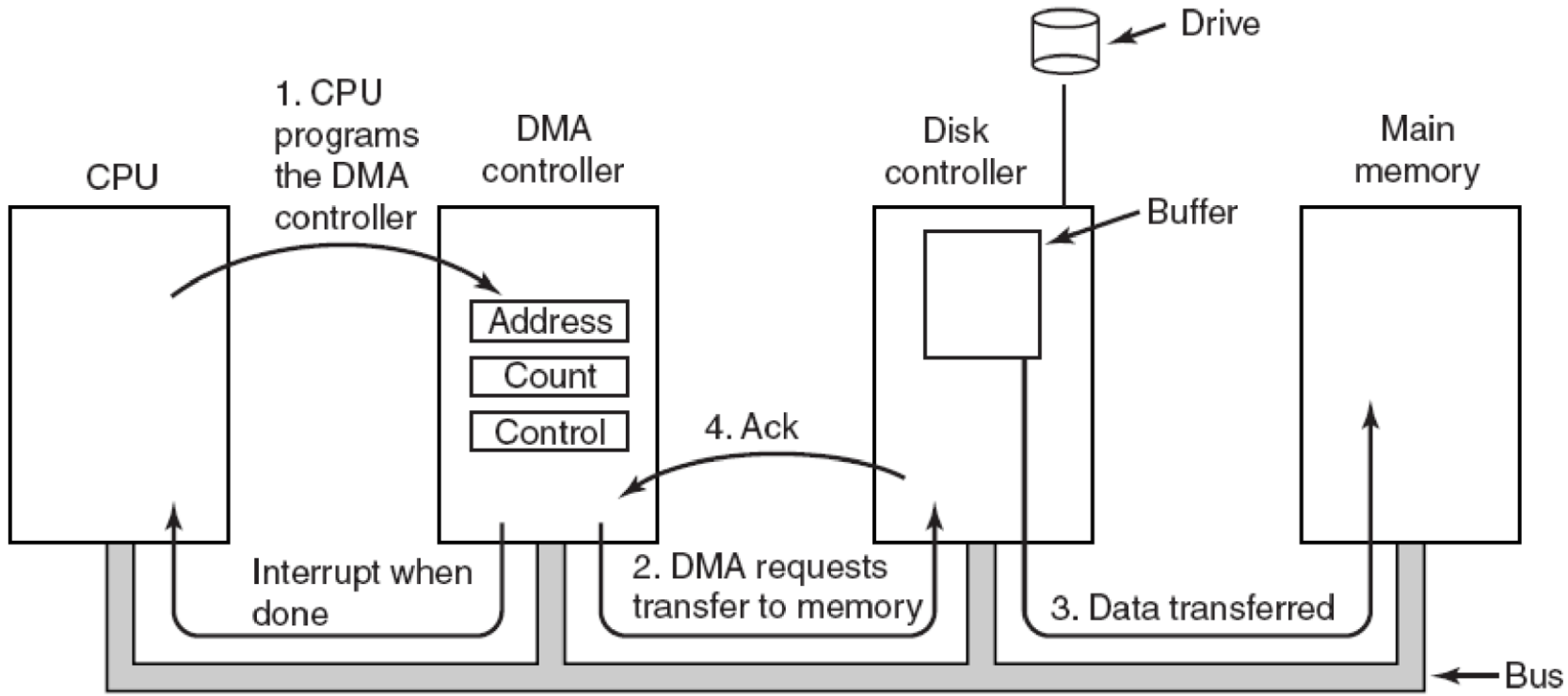
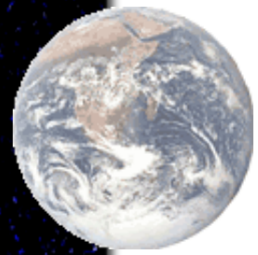


Figure 5-4. Operation of a DMA transfer.

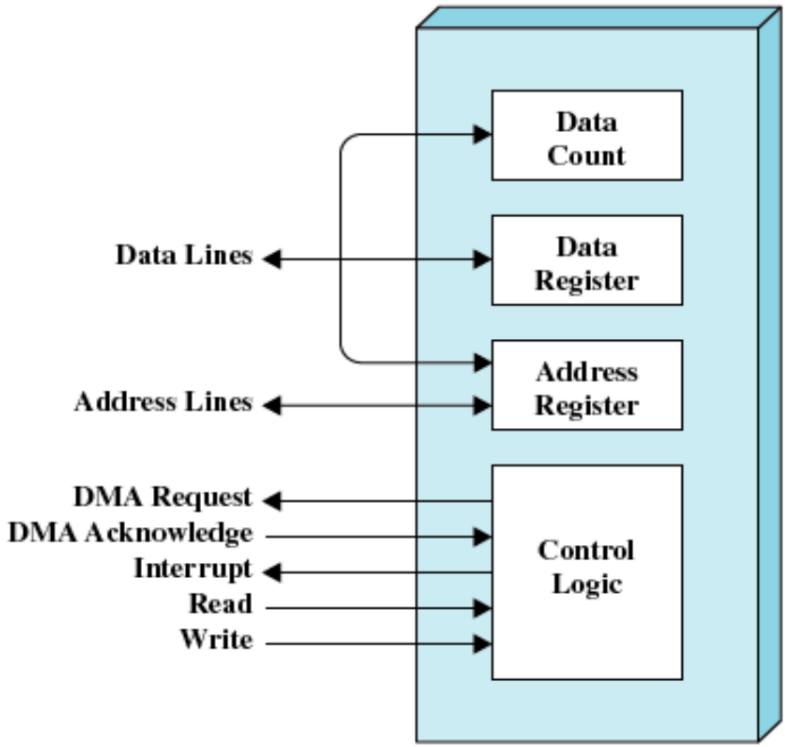


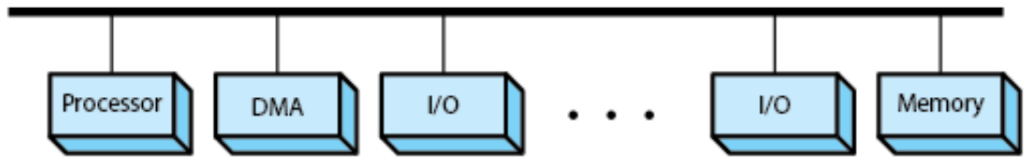
Direct Memory Access

- Processor delegates I/O operation to the DMA module
- DMA module transfers data directly to or form memory
- When complete DMA module sends an interrupt signal to the processor

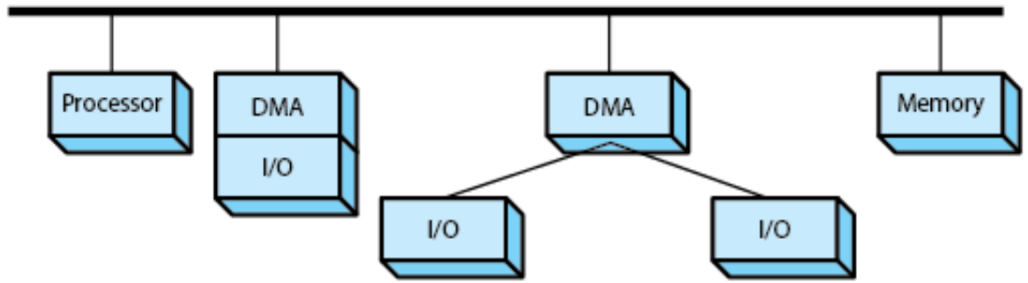
DMA Configurations

DMA

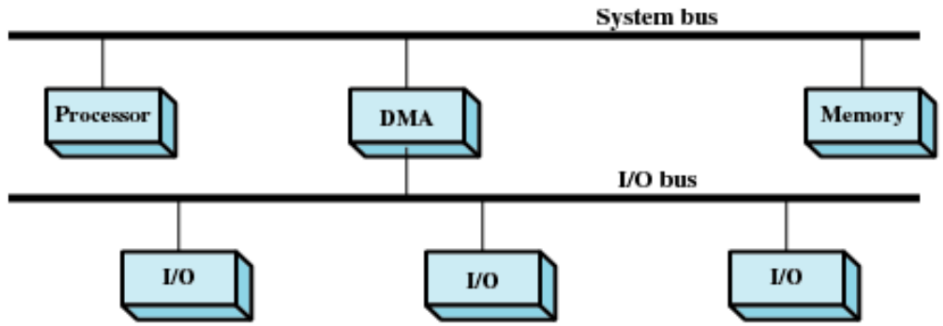




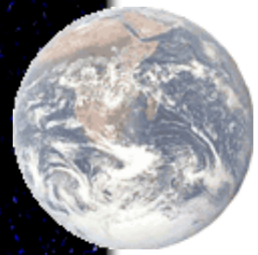
(a) Single-bus, detached DMA



(b) Single-bus, Integrated DMA-I/O

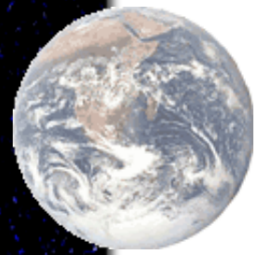


(c) I/O bus



Evolution of the I/O Function

- Processor directly controls a peripheral device
- Controller or I/O module is added
  - Processor uses programmed I/O without interrupts
  - Processor does not need to handle details of external devices



Evolution of the I/O Function

- Controller or I/O module with interrupts
  - Processor does not spend time waiting for an I/O operation to be performed
- Direct Memory Access
  - Blocks of data are moved into memory without involving the processor
  - Processor involved at beginning and end only

Evolution of the I/O Function

- I/O module is a separate processor
- I/O processor
  - I/O module has its own local memory
  - Its a computer in its own right

Operating System Design Issues

- Efficiency
  - Most I/O devices extremely slow compared to main memory
  - Use of multiprogramming allows for some processes to be waiting on I/O while another process executes
  - I/O cannot keep up with processor speed
  - Swapping is used to bring in additional Ready processes which is an I/O operation

Operating System Design Issues

- Generality
  - Desirable to handle all I/O devices in a uniform manner
  - Hide most of the details of device I/O in lower-level routines so that processes and upper levels see devices in general terms such as read, write, open, close, lock, unlock

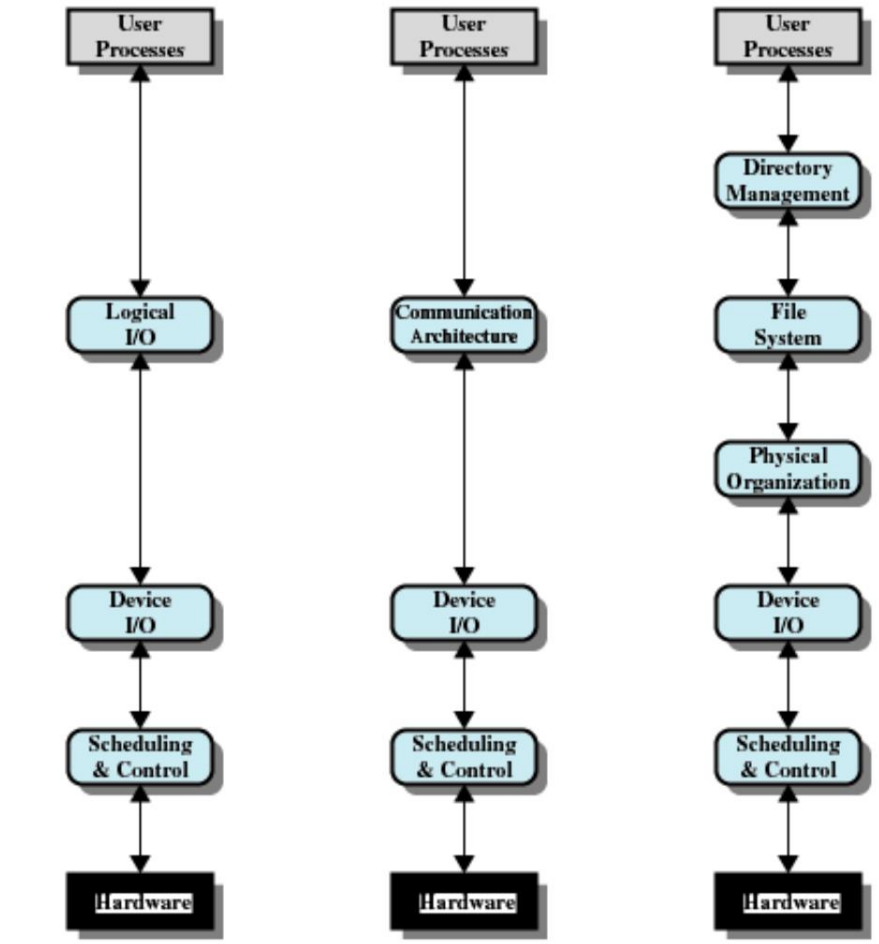


Figure 11.4 A Model of I/O Organization

I/O Software Layers

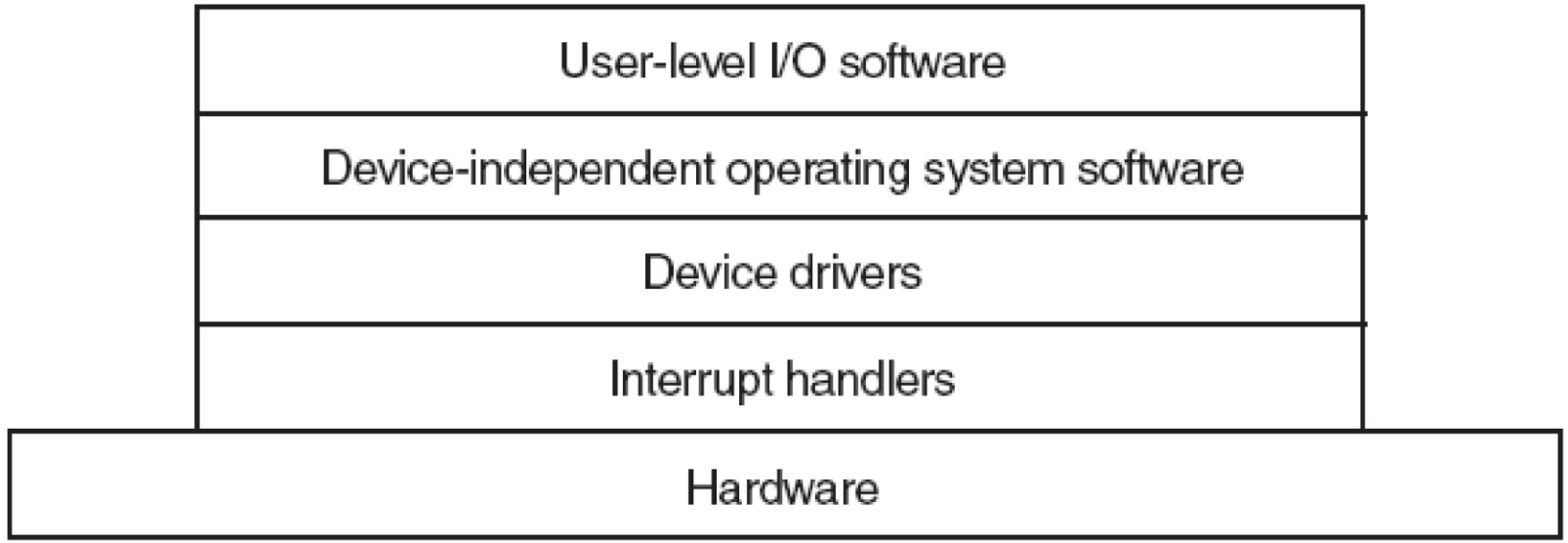


Figure 5-11. Layers of the I/O software system.

Interrupt Handlers (1)

1. Save registers not already been saved by interrupt hardware.
2. Set up a context for the interrupt service procedure.
3. Set up a stack for the interrupt service procedure.
4. Acknowledge the interrupt controller. If there is no centralized interrupt controller, reenable interrupts.
5. Copy the registers from where they were saved to the process table.

Interrupt Handlers (2)

6. Run the interrupt service procedure.
7. Choose which process to run next.
8. Set up the MMU context for the process to run next.
9. Load the new process' registers, including its PSW.
10. Start running the new process.

Device Drivers

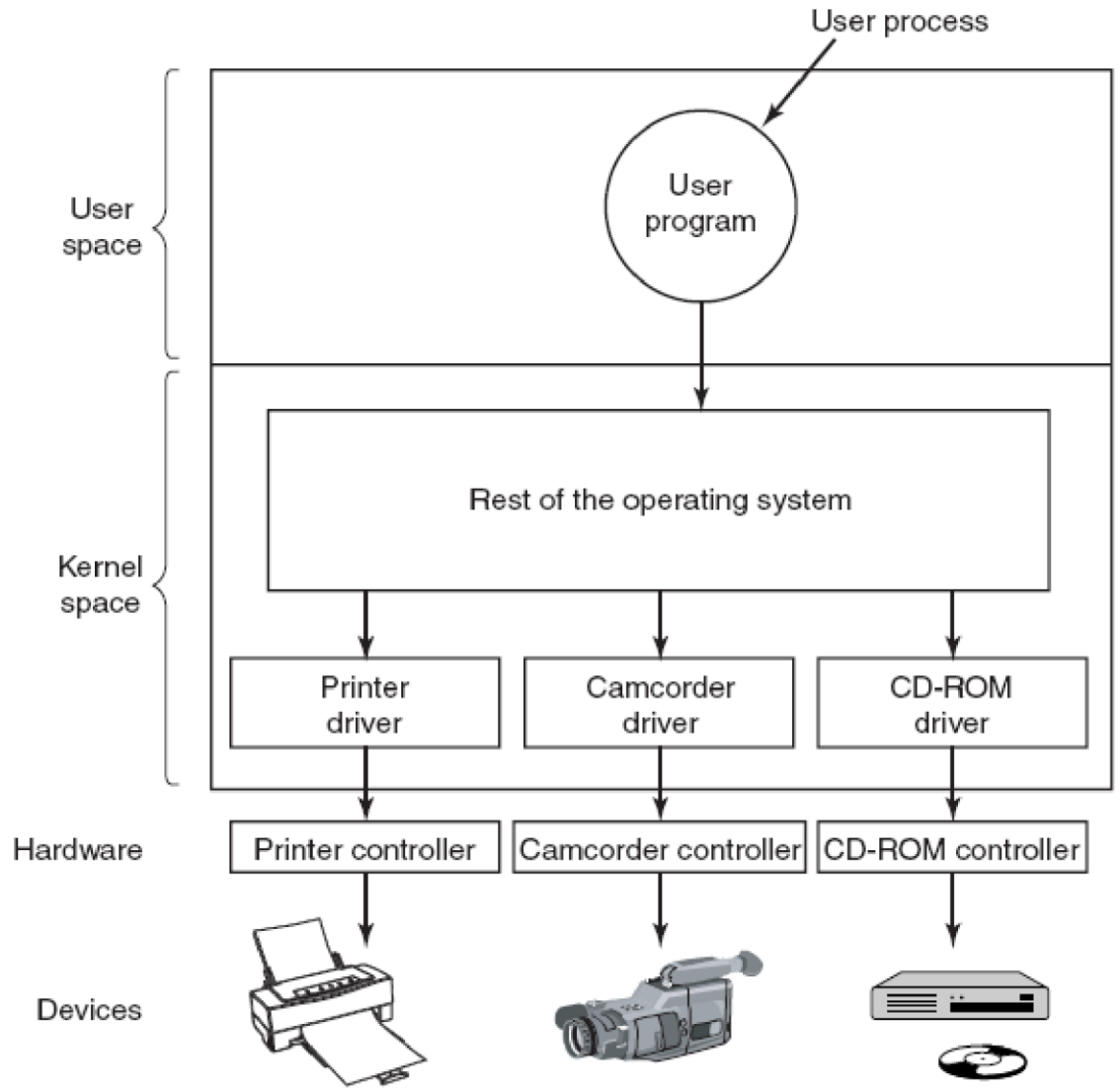


Figure 5-12. Logical positioning of device drivers. In reality all communication between drivers and device controllers goes over the bus.

Device-Independent I/O Software

Uniform interfacing for device drivers
Buffering
Error reporting
Allocating and releasing dedicated devices
Providing a device-independent block size

Figure 5-13. Functions of the device-independent I/O software.

Uniform Interfacing for Device Drivers

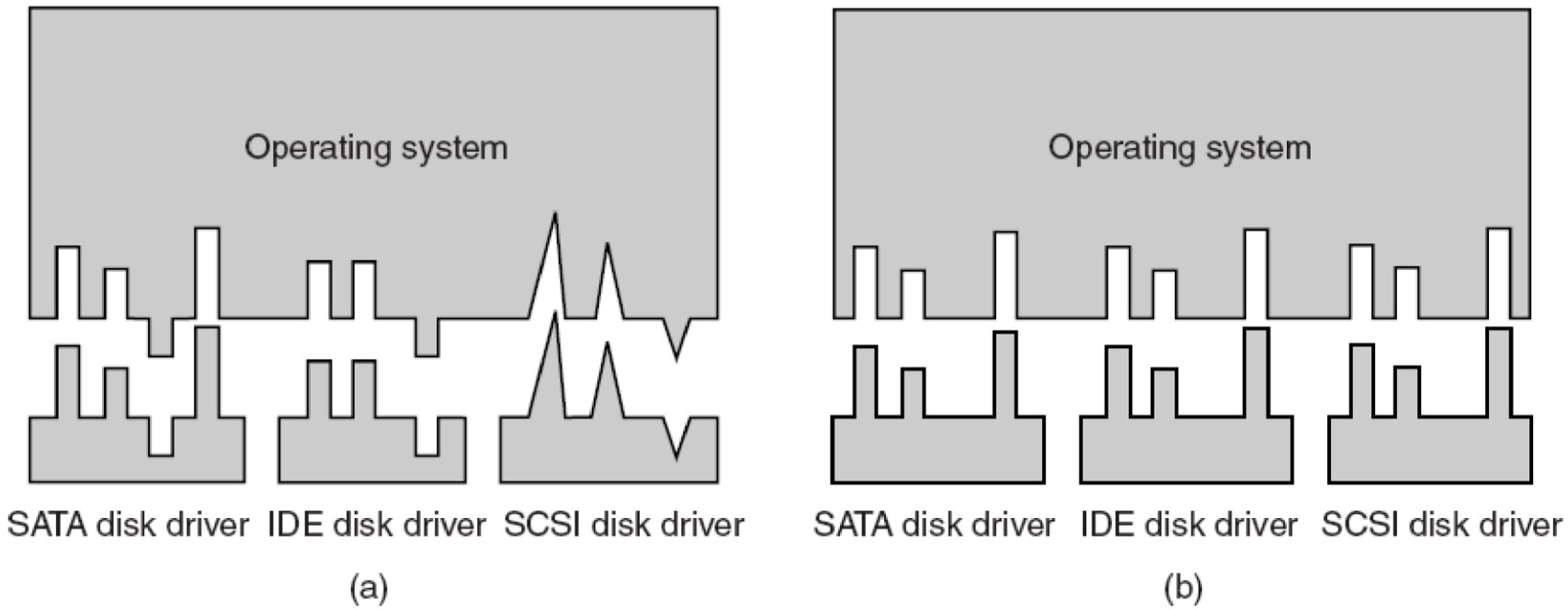


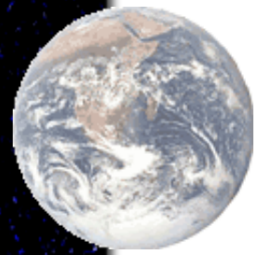
Figure 5-14. (a) Without a standard driver interface. (b) With a standard driver interface.

I/O Buffering

- Reasons for buffering
  - Processes must wait for I/O to complete before proceeding
  - Certain pages must remain in main memory during I/O

I/O Buffering

- Block-oriented
  - Information is stored in fixed sized blocks
  - Transfers are made a block at a time
  - Used for disks and tapes
- Stream-oriented
  - Transfer information as a stream of bytes
  - Used for terminals, printers, communication ports, mouse and other pointing devices, and most other devices that are not secondary storage



Single Buffer

- Operating system assigns a buffer in main memory for an I/O request
- Block-oriented
  - Input transfers made to buffer
  - Block moved to user space when needed
  - Another block is moved into the buffer
    - Read ahead

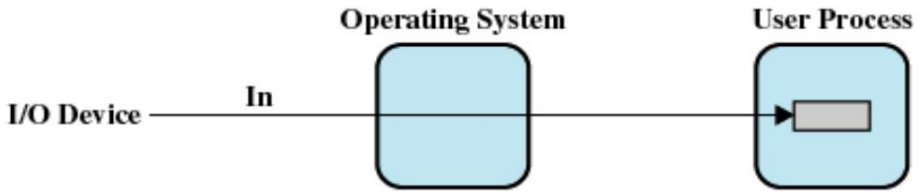
Single Buffer

- Block-oriented
  - User process can process one block of data while next block is read in
  - Swapping can occur since input is taking place in system memory, not user memory
  - Operating system keeps track of assignment of system buffers to user processes

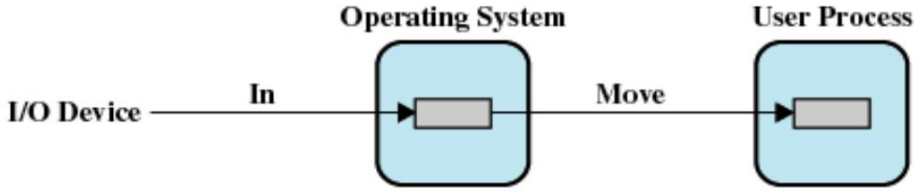
Single Buffer

- Stream-oriented
  - Used a line at time
  - User input from a terminal is one line at a time with carriage return signaling the end of the line
  - Output to the terminal is one line at a time

I/O Buffering



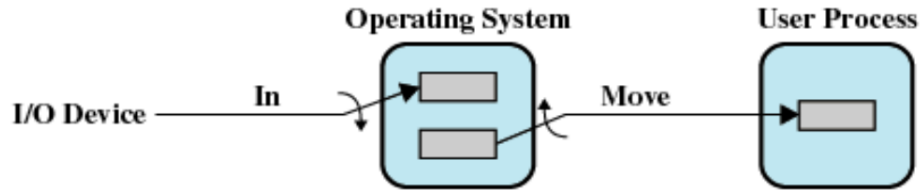
(a) No buffering



(b) Single buffering

Double Buffer

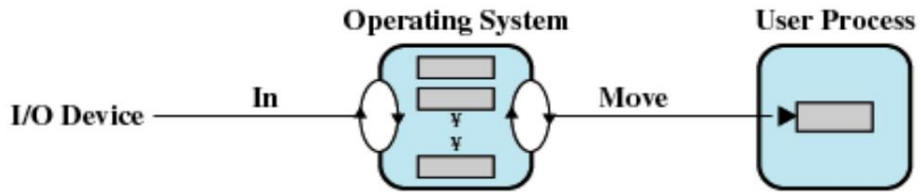
- Use two system buffers instead of one
- A process can transfer data to or from one buffer while the operating system empties or fills the other buffer



(c) Double buffering

Circular Buffer

- More than two buffers are used
- Each individual buffer is one unit in a circular buffer
- Used when I/O operation must keep up with process



(d) Circular buffering

User-Space I/O Software

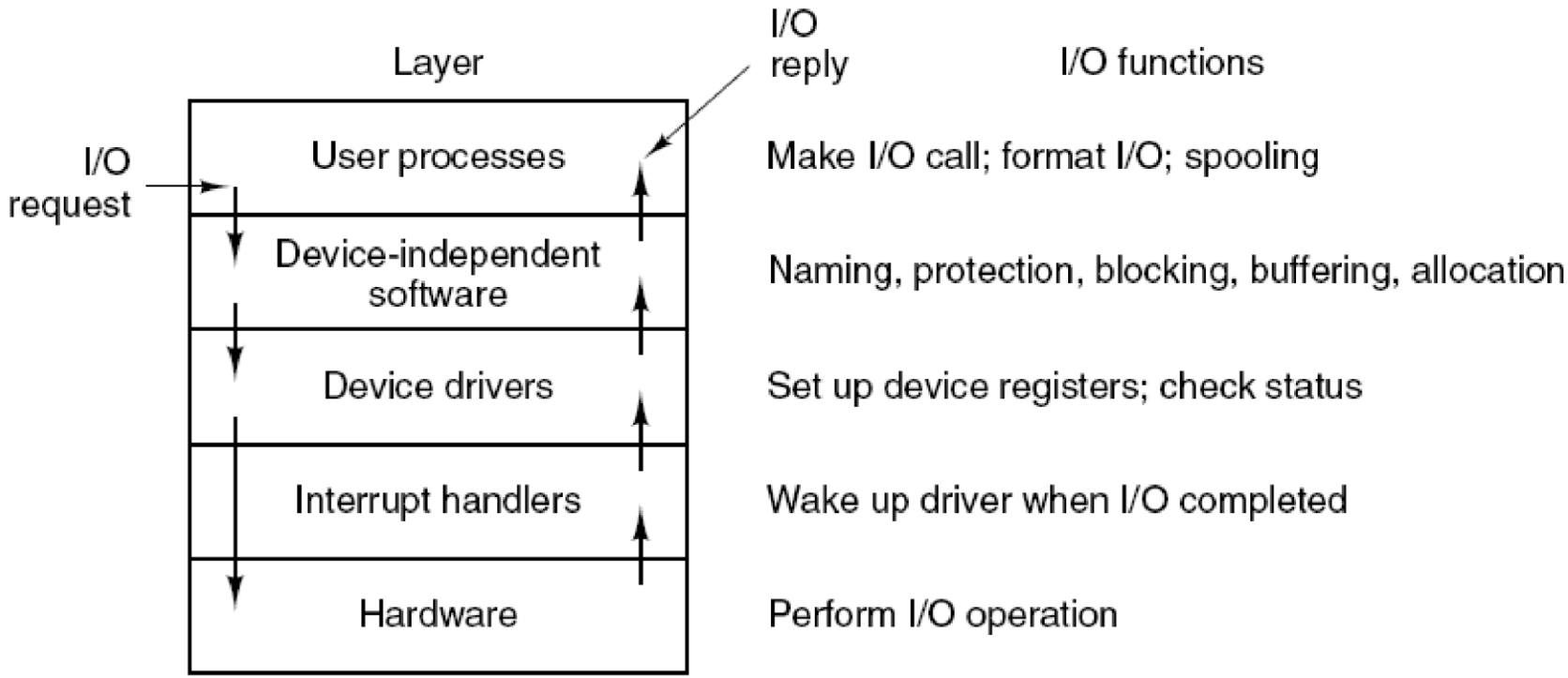


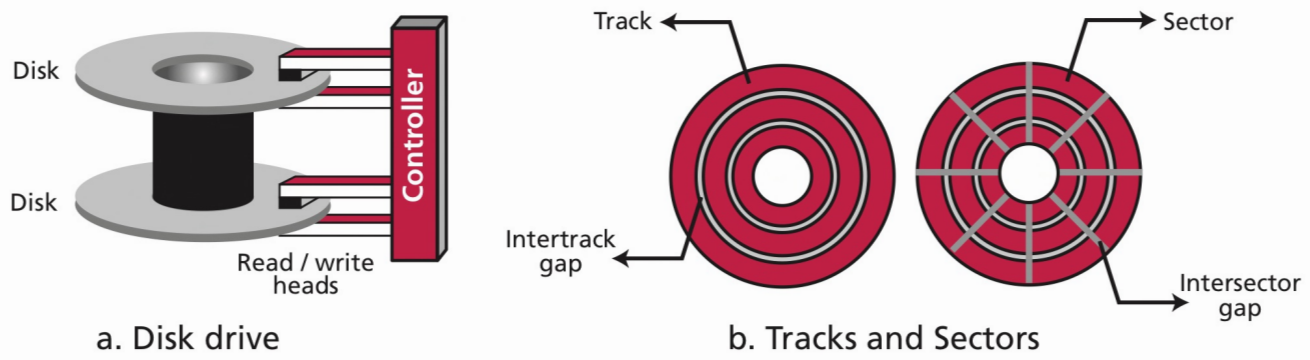
Figure 5-17. Layers of the I/O system and the main functions of each layer.

Magnetic Disks (1)

Parameter	IBM 360-KB floppy disk	WD 18300 hard disk
Number of cylinders	40	10601
Tracks per cylinder	2	12
Sectors per track	9	281 (avg)
Sectors per disk	720	35742000
Bytes per sector	512	512
Disk capacity	360 KB	18.3 GB
Seek time (adjacent cylinders)	6 msec	0.8 msec
Seek time (average case)	77 msec	6.9 msec
Rotation time	200 msec	8.33 msec
Motor stop/start time	250 msec	20 sec
Time to transfer 1 sector	22 msec	17 μsec

Figure 5-18. Disk parameters for the original IBM PC 360-KB floppy disk and a Western Digital WD 18300 hard disk.

Figure 5.6 *A magnetic disk*



a. Disk drive

b. Tracks and Sectors

Timing of a Disk I/O Transfer

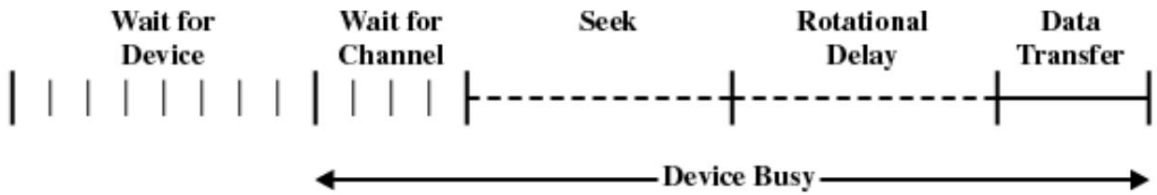
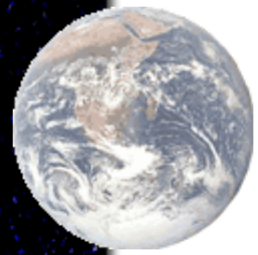


Figure 11.6 Timing of a Disk I/O Transfer

Disk Arm Scheduling Algorithms (1)

Read/write time factors

1. Seek time (the time to move the arm to the proper cylinder).
2. Rotational delay (the time for the proper sector to rotate under the head).
3. Actual data transfer time.

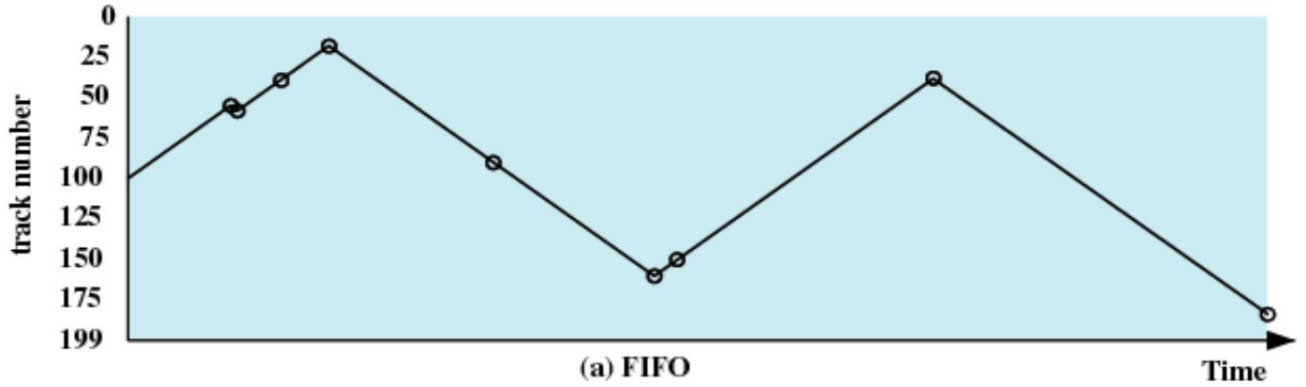


Disk Scheduling Policies

- Seek time is the reason for differences in performance
- For a single disk there will be a number of I/O requests
- If requests are selected randomly, we will poor performance

Disk Scheduling Policies

- First-in, first-out (FIFO)
  - Process request sequentially
  - Fair to all processes
  - Approaches random scheduling in performance if there are many processes

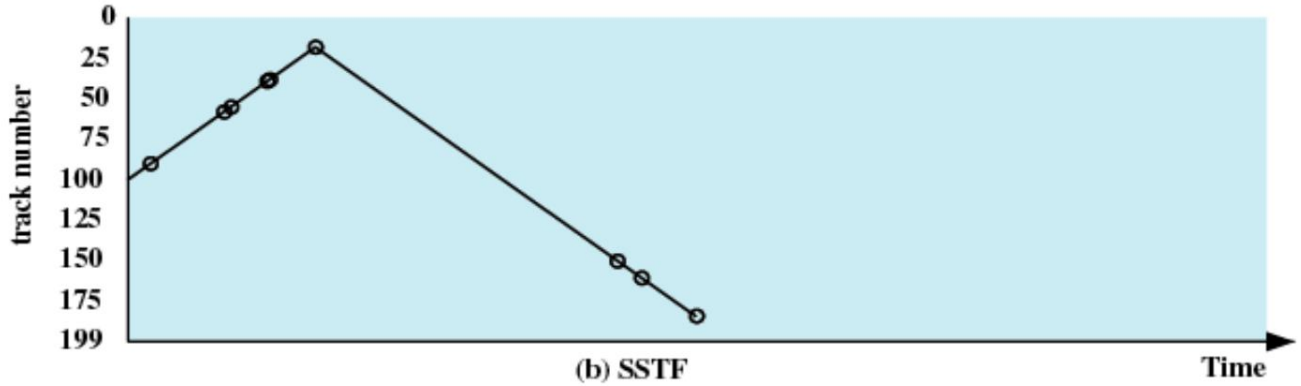


Disk Scheduling Policies

- Priority
  - Goal is not to optimize disk use but to meet other objectives
  - Short batch jobs may have higher priority
  - Provide good interactive response time

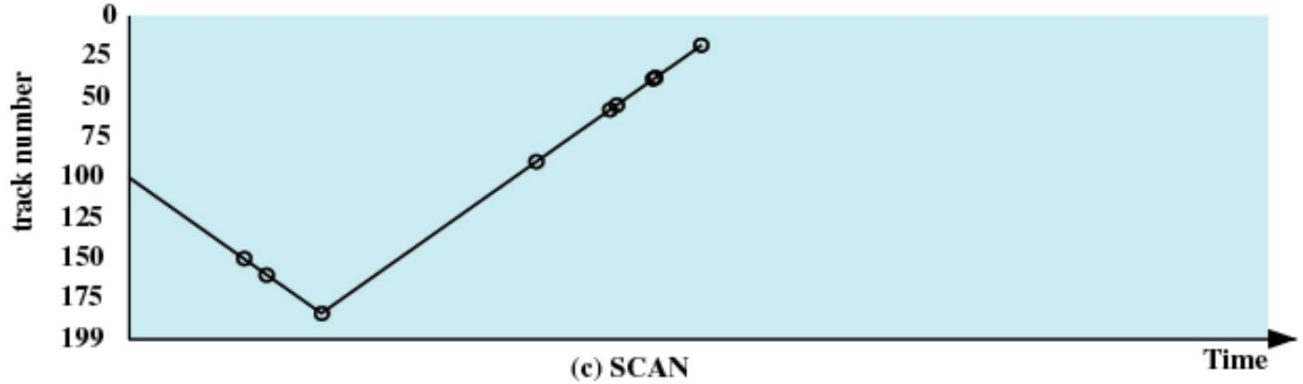
Disk Scheduling Policies

- Shortest Service Time First
  - Select the disk I/O request that requires the least movement of the disk arm from its current position
  - Always choose the minimum Seek time



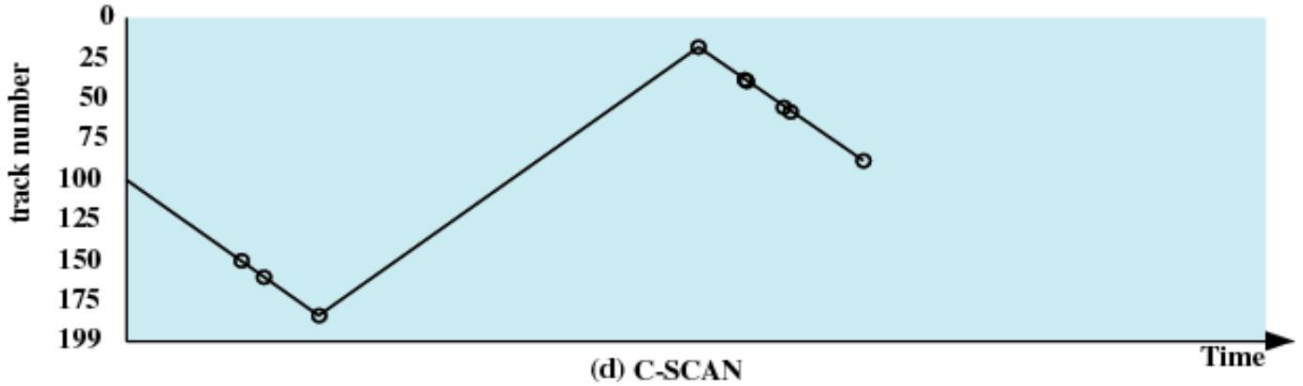
Disk Scheduling Policies

- SCAN
  - Arm moves in one direction only, satisfying all outstanding requests until it reaches the last track in that direction
  - Direction is reversed



Disk Scheduling Policies

- C-SCAN
  - Restricts scanning to one direction only
  - When the last track has been visited in one direction, the arm is returned to the opposite end of the disk and the scan begins again



Disk Scheduling Algorithms

Table 11.2 Comparison of Disk Scheduling Algorithms

(a) FIFO (starting at track 100)		(b) SSTF (starting at track 100)		(c) SCAN (starting at track 100, in the direction of increasing track number)		(d) C-SCAN (starting at track 100, in the direction of increasing track number)	
Next track accessed	**Number of tracks traversed**	**Next track accessed**	**Number of tracks traversed**	**Next track accessed**	**Number of tracks traversed**	**Next track accessed**	**Number of tracks traversed**
55	45	90	10	150	50	150	50
58	3	58	32	160	10	160	10
39	19	55	3	184	24	184	24
18	21	39	16	90	94	18	166
90	72	38	1	58	32	38	20
160	70	18	20	55	3	39	1
150	10	150	132	39	16	55	16
38	112	160	10	38	1	58	3
184	146	184	24	18	20	90	32
Average seek length	55.3	**Average seek length**	27.5	**Average seek length**	27.8	**Average seek length**	35.8

Error Handling

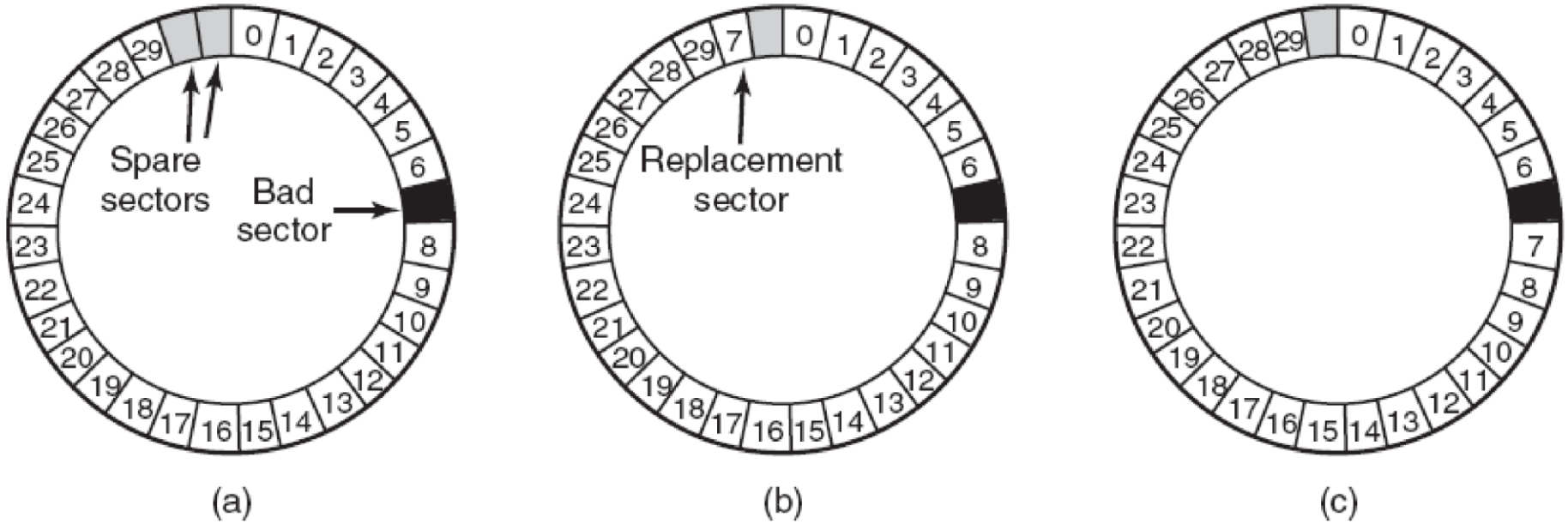


Figure 5-30. (a) A disk track with a bad sector.
(b) Substituting a spare for the bad sector.
(c) Shifting all the sectors to bypass the bad one.