# Chapter Four

# Input/ Output Management

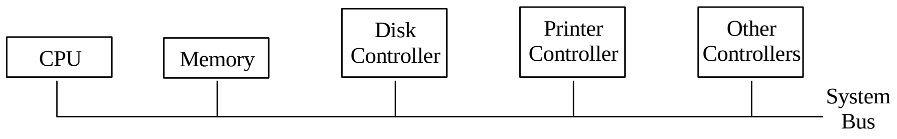
## 4.1. Introduction

I/O is important for communication between the user and computer. The following are functions of the operating system as I/O manager:

- ♦ Control all the computer I/O devices
- ♦ Issue commands to I/O devices, catch interrupts and handle errors
- ♦ Provide an interface between the device and the rest of the system

## 4.2 Principles of I/O hardware

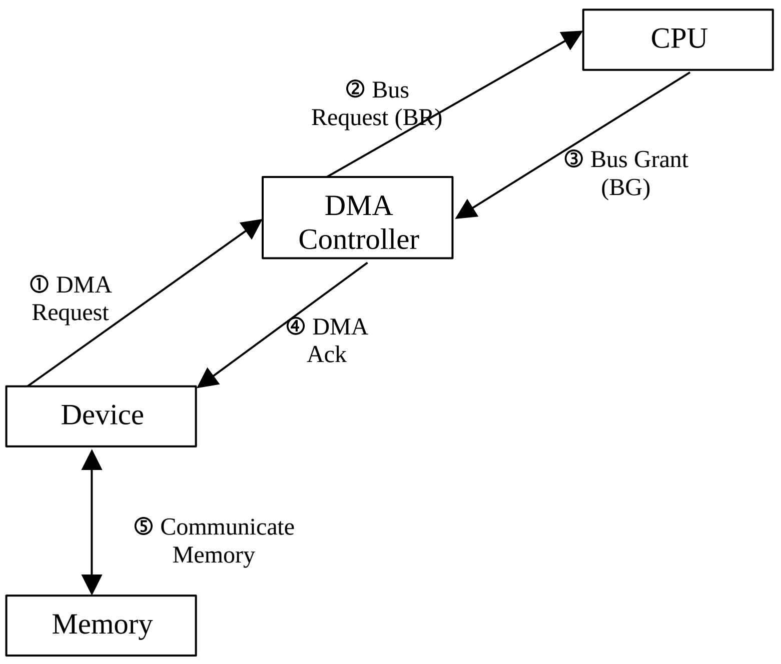
- ♦ **I/O devices**: - are concerned with the way data are handled by the I/O device. There are two types of I/O devices **blocked** and **character**
  - Blocked devices (such as disks) - A block device is one that stores information in fixed-size blocks, each one with its own address. Common block size range from 512 bytes to 32,768 bytes. The essential property of a block device is that it is possible to read or write each block independently of all the other ones. Disks are the most common block devices. A disk is a block addressable device because no matter where the arm currently is, it is always possible to seek to another cylinder and then wait for the required block to rotate under the head.
  - Character devices (such as printers, mouse and NIC) - The other type of I/O device is the character device. A character device delivers or accepts stream of character, without regard to any block structure. It is not addressable and does not have any seek operation.
- ♦ **I/O Unit**: indicates the hardware components. There are two major components – **Electronic Component** (Device controller/Adapter) and the **Mechanical Component** (the moving parts)



- ♦ **Direct Memory Access (DMA)**: is a technique for moving a data directly between main memory and I/O devices without the CPU's intervention. In the absence of DMA reading from a disk is done with the following steps:
  - The controller reads the block (one or more sectors) from the drive serially, bit by bit, until the entire block is in the controller's internal buffer
  - The controller computes the checksum to verify that no read errors have occurred
  - The controller causes an interrupt
  - The OS reads the disk block from the controller's buffer a byte or a word at a time and stores it on memory

**Problem**: Since the OS controls the loop of reading it wastes the CPU time. On the other hand, when DMA is used the Device Controller will do the counting and address tracking activities.

- The following diagram shows the steps to use DMA:



## 4.3 Principles of I/O software

- Layered technique is used
- Goals and issues of I/O software:

1. **Device Independence**: It should be possible to write programs that can read files on a floppy disk, on hard disk, or on a CD-ROM, without having to modify the program for each different device types. It is up to the operating system to take care of the problems caused by the fact that these devices are really different.
2. **Uniform Naming:** the name of a file or a device should simply be a string or an integer and not dependant on the device in any way.
3. **Error handling:** In general, errors should be handled as close to the hardware as possible (at the device controller level). Many errors are transient, such as read errors caused by specks of dust on the read head, and will go away if the operations are repeated.
4. **Transfer:** There are two types of transfer modes – **Synchronous (Blocking)** and **Asynchronous (interrupt –driven)**. In the case of synchronous transfer the program requesting I/O transfer will be suspended until the transfer is completed. In the case of Asynchronous transfer the CPU starts the transfer and goes off to do something until the interrupt that shows the completion of the transfer arrives.
5. **Device types:** There are two device types –

   **Sharable** (such as disk) - used by many users at the same time. No problems are caused by multiple users having open files on the same disk at the same time.

**Dedicated** (tape drives) have to be dedicated to a single user until that user is finished. Having two or more users writing blocks intermixed at random to the same tape will definitely not work

**Layers of I/O software**

- The following are the I/O software layers
  - Interrupt handler (bottom)
  - Device driver
  - Device independent OS software
  - User-level software (top)

1. **Interrupt Handler**
   - Interrupts are unpleasant facts of life and cannot be avoided. They should be hidden away.
   - The best way to hide them is to have every process starting an I/O operation block until the I/O has completed and the interrupt occurs.
   - When the interrupt happens, the interrupt procedure does whatever it has to in order to unblock the process that started it.
2. **Device Driver**
   - All device – dependent code goes in the device driver
   - Each device driver handles one device type, or at most, one class of closely related devices.
   - Each controller has one or more device registers used to give it command
   - The device driver issue the commands and check that they are carried out properly
   - Thus, the disk driver is the only part of the OS that knows how many registers that disk controller has and what they are used for.
   - In general terms, the job of a device driver is to accept requests form the device-independent software above it and sees to it that the request is executed.
   - Steps in carrying out I/O requests:
     - Translate it from abstract to concrete terms
     - Write into the controller's device registers
     - The device driver blocks itself until interrupt comes
     - The blocked driver will be awakened by the interrupt
     - Device driver checks for errors
     - If everything is all right, the driver may have data to pass to the device independent software
     - If nothing is queued, the driver blocks waiting for the next request

**3. Device Independent I/O Software**

- It is large fraction of I/O software
- Functions of the device-independent I/O software

1. **Uniform interfacing for device drivers** – perform I/O function common to all drives
2. **Device Naming** – responsible for mapping symbolic devices names onto the proper driver
3. **Device protection** – prevent users from accessing devices that they are not entitled to access
4. **Providing device-independent block size** – provide uniform block size to higher layers hiding differences in block sizes
5. **Buffering**: if a user process write half a block, the OS will normally keep the data in buffer until the rest of the data are written. Keyboard inputs that arrive before it is needed also require buffering.
6. **Storage allocation on block devices:** when a file is created and filled with data, new disk blocks have to be allocated to the file. To perform this allocation, the OS needs a list of free blocks and used some algorithms for allocation
7. **Allocating and releasing dedicated devices**: It is up to the OS to examine requests for devices usage and accept or reject them.
8. **Error reporting:** Errors handling, by and large, is done by drivers. Most errors are device dependent. After the driver tries to read the block a certain number of times, it gives up and informs the device-independent software. It then reports to the caller.

**4. User Space I/O Software**

- A small portion of the I/O software is outside the OS
- System calls, including the I/O system calls, are normally made by library procedures. E.g. when a C program contains the call count=write (fd, buffer, nbytes); the library procedure write will be linked with the program and contained in the binary program present in memory at run time.
- Formatting of input and output is done by library procedures
- The following is an example how the I/O system works during reading by an application
  - **Step 1**: the user program produce a system call for reading from a file
  - **Step 2**: the device-independent software looks in the block cache; if the needed block is there it calls the device driver.
  - **Step 3:** The device driver issue the request to the hardware and the process will be blocked until the disk operation has been completed
  - **Step 4:** When the disk is finished, the hardware generates an interrupt
  - **Step 5:** The interrupt handler is run to discover what happened, i.e., which device wants attention right now. It then extracts the status form

the device and wakes up the sleeping process to finish off the I/O request and let the user process continue

- The following figure shows the I/O system layers and the main functions of each layer

| | | | |
|---|---|---|---|
| ↓ | User Processes | ↑ | Make I/O call; format I/O; Spooling |
| ↓ | Device-independent software | ↑ | Naming, protection, blocking, buffering, allocation |
| ↓ | Device Drivers | ↑ | Setup devices registers; check status |
| ↓ | Interrupt Handlers | ↑ | Wakeup driver when I/O completed |
| | Hardware | | Perform I/O operation |

## 4.4 Deadlocks

Computer systems are full of resources that can only be used by one process at a time.
E.g. plotter, CD-ROM readers/recorders, tape drive backup systems

**Definition**: A set of processes is **deadlocked** if each process in the set is waiting for an event that only another process in the set can cause.

- Deadlock happens when processes have been granted *exclusive access* to resources such as devices, files, and so forth.
- A **resource** is anything that can only be used by a single process at any instance.
- There are two types of resources
  - **Preemptable** (can be taken away from a process) and
  - **Non-preemptable** (cannot be taken away from its current owner without causing the computation to fail). If a process has began to burn a CD-ROM, suddenly taking the CD recorder away from it and giving it to another process will result in a garbled CD.
- In general deadlock involves non-preemptable resources
- The sequence of events while using resources are:
  - Request the resource
  - Use the resource
  - Release the resource

**Examples:**

1. Suppose that process A asks for CD-ROM drive and gets it. A moment later, process B asks for the flatbed plotter and gets it, too. Now process A asks for the plotter and blocks waiting for it. Finally process B asks for CD-ROM drive and also blocks. This situation is called a deadlock.
2. Suppose process A locks record R1 and process B locks record R2, and then each process tries to lock the other one’s record, this time a deadlock will happen.

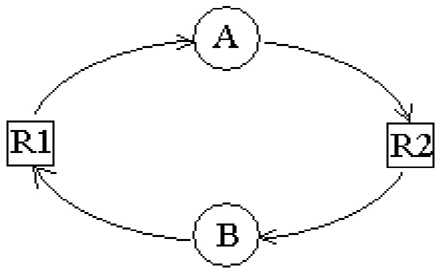
Conditions for deadlock

1. **Mutual Exclusion:** Each resource is either currently assigned to exactly one process or is available
2. **Hold and Wait Condition:** Processes currently holding resources granted earlier can request new resources
3. **No Preemptive Condition:** Resources previously granted cannot be forcibly taken away from a process
4. **Circular Wait Condition:** There must be a circular chain of two or more processes, each of which is waiting for a resource held by the next member of the chain.

**Note:** If one of the four is absent, no deadlock is possible

**Deadlock Modeling**

- Directed graph can be used
- We have two kinds of nodes: Process (circles) and Resources (square)
- An arc from a resource to a process means the resource is currently held by that process
- An arc from process to a resource means the process is currently blocked waiting for that resource
- The following resource allocation graph shows a deadlock

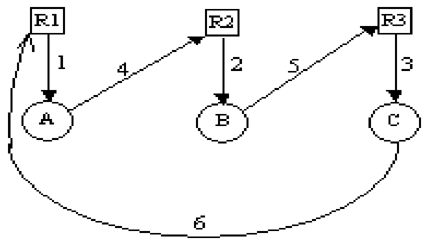


The Cycle is:
B-R1-A-R2-B

**Example**: Let we have three processes A, B, C and three resources R1, R2 and R3. The request and the release of the three processes are as follows:

| **A** | **B** | **C** |
|---|---|---|
| A1: requests R1 | B1: Requests R2 | C1: Requests R3 |
| A2: requests R2 | B2: Requests R3 | C2: Requests R1 |
| A3: releases R1 | B3: Releases R2 | C3: Releases R3 |
| A4: releases R2 | B4: Releases R3 | C4: Releases R1 |

- Let round robin scheduling algorithm is used and instructions are executed in the following order:
  A1→ B1→ C1 → A2 → B2 → C2



- Resource graphs are a tool that let us see if a given request/release sequence leads to deadlock
- If there is any cycle in the graph that means there is a deadlock

**Solving Deadlock Problems**

- In general, four strategies are used for dealing with deadlocks
    - Ignore the problem
    - Detect and recover
    - Dynamically avoid deadlock by careful resource allocation
    - Prevent, by structurally negating one of the four required conditions

1. **The Ostrich Algorithm**
    - It is the simplest approach
    - Stick your head in the sand and pretend there is no problem at all.
        - Mathematicians do not accept it
        - Engineers accept with constraints – frequency (chance of happening)
    - The problem is the price, mostly in terms of putting inconvenient restrictions on processes
2. **Detection and Recovery**
    - Every time a resource is requested or released, the resource graph is updated, and a check is made to see if any cycles exist
    - If cycle exists, one of the processes in the cycle is killed
    - If the method is to check the length of time a process is requesting a resource and if large kill it
3. **Deadlock Prevention**
    - Impose some condition on processes
        - **Spooling** – By spooling printer output, several processes can generate output at the same time but the only process that actually requests the physical printer is the printer daemon.
        - **Disadvantage**: This solution cannot be used for every device
    - If we can prevent processes that hold resources from waiting for more resources, we can eliminate deadlocks. (request before use)
        - **Disadvantage**: Resources will not be used optimally and most processes do not know that resources they need before running
    - Let a process temporarily release all the resources it currently holds when requesting another resource.
    - Let every process is entitled only to a single resource at any moment
        - Disadvantage: Consider a process that copies a huge file from a tape to a printer

- Let resources be requested in predefined order
    - **Disadvantage**: Ordering resources is almost impossible to meet the requirements of all processes

**Summary**

| Condition | Approach |
|---|---|
| Mutual Exclusion | Spool everything |
| Hold and wait | Request all resources initially |
| No preemptive | Take resources away |
| Circular wait | Order resources numerically |

4. **Deadlock Avoidance**
    - We can avoid deadlocks, but only if certain information is available in advance
    - The Banker's algorithm considers each request as it occurs and sees if granting it leads to a safe state. If it does, the request is granted; otherwise, it is postponed until later.
    - To see if a state is safe, the banker checks to see if he has enough resources to satisfy the customer closest to his or her maximum. If so, those loans are assumed to be repaid. If all loans can eventually be repaid, the state is safe and the initial request can be granted.
    - Here there is no imposition of rules on processes like the case in prevention case.

**4.5 Disks**

**Disk**

- All real disks are organized into cylinders, each one containing many tracks. Each of the tracks then will be divided into sectors (equal number of sectors or different number of sectors)
- In the case of equal number of sectors
    - The data density as closer to the center (hub) is high
    - The speed increases as the read/write moves to the outer tracks
- Modern large hard drives have more sectors per track on outer tracks e.g. IDE drives
- Many controllers can read or write on one drive while seeking on one or more other drives, but floppy disk controller can not do that.

**Disk Access Time**

- The time required to read or write a disk block is determined by three factors:
    - The seek time (the time to move the arm to the proper cylinder)
    - The rotational delay (the time for the proper sector to rotate under the head)
    - The actual data transfer time
- For most disks, the seek time dominates the other two times, so reducing the mean seek time can improve system performance substantially.

- While the arm is seeking on behalf of one request, other disk requests may be generated by other process. Many disk drivers maintain a table, indexed by cylinder number, with all the pending requests for each cylinder chained together in a linked list headed by the table entries

**Disk Arm Scheduling Algorithm**

The OS maintains queue of requests for each I/O operation

It uses disk scheduling algorithm

1. **First Come First Served (FCFS) :** Accept a request one at a time and carries them out in that order

   E.g. Track initial position: 11
   Track request: 1,36,16,34,9,12
   Service order: 1,36,16,34,9,12
   Arm motion required: 10, 35, 20, 18, 25, 3,
   Total= 111 tracks
   The simplest and the fairest of all, but it doesn't improve performance

2. **Shortest Seek First (SSF):** It handles the closest (the least disk arm movement) request next, to minimize seek time.

   E.g. Track initial position: 11
   Track request: 1,36,16,34,9,12
   Service order: 12,9,16, 1, 34, 36
   Arm motion required: 1, 3, 7, 15, 33, 2
   Total= 61 tracks
   Advantage: Performance (efficiency), provides better performance
   Disadvantage: Possibility of starvation (it lacks fairness)

3. **SCAN (Elevator) Algorithm :** The disk arm keeps moving in the same direction until there are no more outstanding requests in that direction, then it switches direction

   Direction bit 1= up, 0=down

   E.g. Track initial position: 11
   Track request: 1,36,16,34,9,12
   Direction bit: 1
   Service order: 12, 16, 34, 36, 9, 1
   Disk arm motion: 1, 4, 18, 2, 27, 8
   Total= 60 tracks

   It provides better service distribution

4. **C-SCAN (Modified Elevator) Algorithm**

   It restricts scanning to one direction only. A slight modification of elevator algorithm that has smaller variance in response times is always scan in the same direction. When the highest numbered cylinder with a pending request has been serviced, the arm goes to the lowest-numbered cylinder with a pending request and then continues moving in an upward direction. In effect, the lowest-numbered cylinder is thought of as being just above the highest-numbered cylinder.

   It reduces the maximum delay experienced by new request

**RAM Disk**

- A RAM disk has the advantage of having instant access
- Unix support mounted file system but DOS and Windows do not support
- The RAM disk is split up into n blocks each with a size equal to the real disk
- Finally the transfer will be done
- A RAM disk driver may support several areas of memory used as RAM disk

**Disk Cache**

Memory cache – To narrow the distance between the processor and memory.

Disk cache – To narrow the distance between the processor/ memory and I/O

It uses a buffer kept in main memory that functions as a cache of disk memory and the rest of the main memory.

It contains a copy of some of the sectors on the disk.

It improves performance (by minimizing block transfer. Disk $\Longleftrightarrow$ memory

Design issue

Data transfer - Memory-to-memory

- using shared memory (pointer)

Replacement algorithm - Least Recently used

- Least Frequently Used