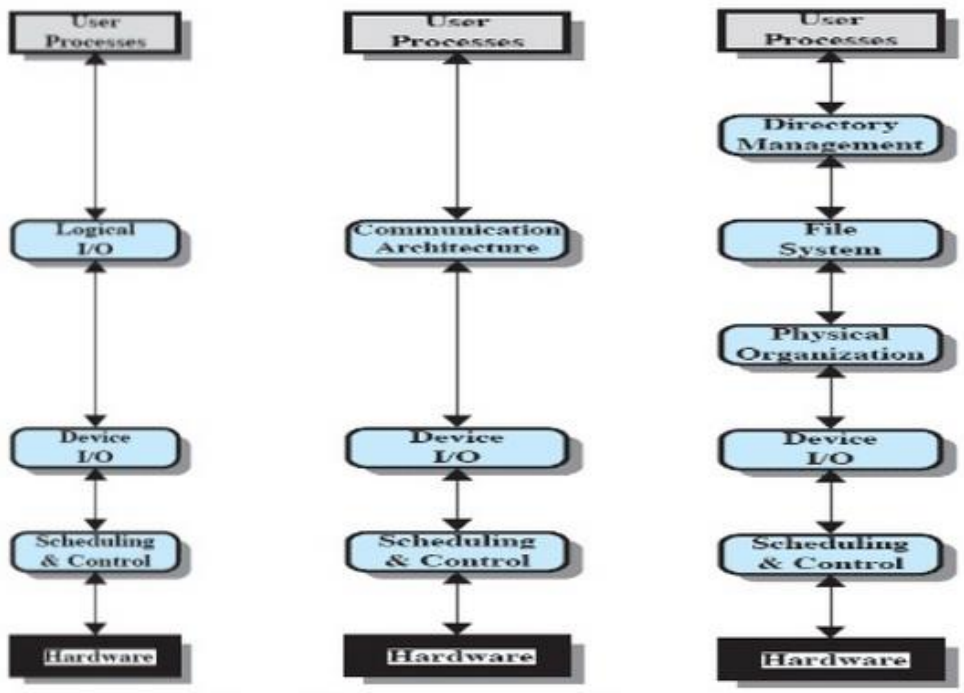
Contents

Hierarchical Structure of an OS

Modern OSs are hierarchical in nature and this means that they are organised into a series of layers, separated according to their complexity, characteristic time scale and their level of abstraction.

Each layer performs a related subset of the functions required of the OS and relies on the next lower layer to perform more primitive functions and to conceal the details of those functions. It provides services to the next higher level and changes in one layer should not affect other layers.



(a) Local peripheral device *(b) Communications Port* *(c) File system*

Figure 1: A Model of I/O Organisation

The three most important logical structures, of most operating systems, are shown in Figure 1: A Model of I/O Organisation above.

(a) Logical I/O deals with the device as a logical resource and is concerned with managing general I/O functions on behalf of user processes, allowing them to deal with the device in terms of a device identifier and simple commands: Open, Close, Read, and Write.

(b) Device IO: The requested operations and data are converted into appropriate sequence of I/O instructions. Buffering techniques may be used to improve use.

(c) Scheduling and control: Performs the actual queuing and scheduling of I/O operations at this level.

For (b) the Communication device, the I/O structure looks much the same as the layers involved in (a) the Logical peripheral device just described. The main difference is that the logical I/O module is replaced by a communications architecture, which may itself consist of a number of layers, such as is the case for the TCP/IP layer.

The representation of the structure for managing I/O devices on secondary storage is by using file systems. As shown in part (c) of Figure 1: A Model of I/O Organisation, this has three extra layers:

- **Directory management**: Symbolic file names are converted to identifiers. This level also concerned about user operations that affect the directory of files, such as Add, Delete, and Reorganisation of the users file system
- **File system**: Deals with logical structures and operation of files. Open, Close, Read, Write. Access rights are handled in this level.
- **Physical organisation**: Logical names of files are converted to physical secondary storage addresses. Allocation of secondary storage space and main storage buffer is handled in this level.

The I/O function is generally broken up into a number of layers, with lower layers dealing with details that are closer to the physical functions to be performed and higher layers dealing with the I/O in a logical and generic fashion. The result is that changes in hardware parameters need not affect most of the I/O software.

A key aspect of I/O is the use of buffers that are controlled by I/O utilities rather than by application processes.

Buffering

Processes must wait for I/O to complete before proceeding. To avoid deadlock certain pages must remain in main memory during I/O

It may be more efficient to perform input transfers in advance of requests being made and to perform output transfers some time after the request is made and this technique is known as ***buffering***.

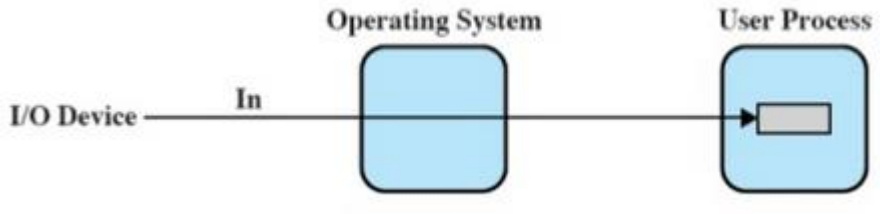
There are two types of I/O devices:

1. A **block-oriented device** stores information in fixed size blocks and transfers are made one block at a time. Generally it's possible to reference data by its block number. For example: disks and USB keys.
2. A **stream-oriented device** transfers data in and out as a stream of bytes with no block structure. For example printers, communications ports, mouse and other pointing devices and most other devices that are not in secondary storage are stream oriented.

Approaches to Buffering

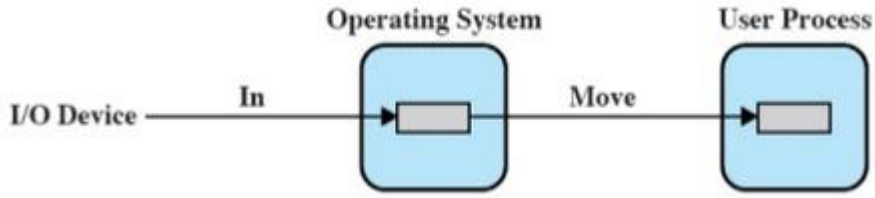
i) No Buffering

The OS directly accesses the device as and when it needs it.



ii) Single Buffer

OS assigns a buffer in main memory for an I/O request



iii) Block Oriented Single Buffer

Input transfers are made to the buffer. When the transfer is complete, the block is moved to user space and immediately requests the next block which is moved into the buffer, called ***reading ahead*** or ***Anticipated Input*** as it is done in the expectation that the block will eventually be needed. For many types of computation, it is often a reasonable assumption as data is usually accessed sequentially.

The user process can be processing one block of data while the next block is being read in. The OS is able to swap the process out because the input operation is taking place in the system memory rather than user process memory. This speeds up processing but adds complexity to the logic in the OS as the OS must keep track of the assignment of system buffers to user processes.

iv) Stream-oriented Single Buffer

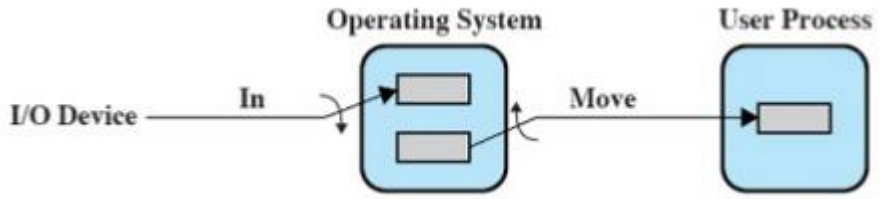
The single buffering scheme can be used in a “Line-at-time” or “Byte-at-a-time” fashion.

E.g.: Terminals often deal with one line at a time with carriage return signalling the end of the line

Byte-at-a-time suites devices where a single keystroke may be significant as well as for sensors and controllers

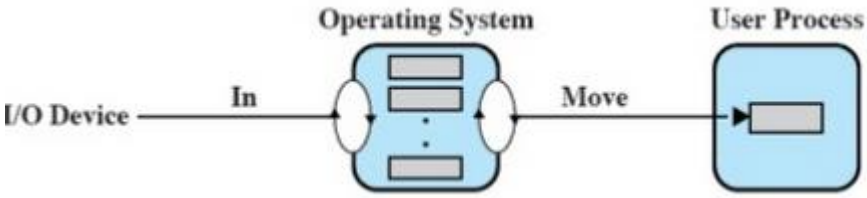
v) Double Buffer

An improvement over single buffering as two system buffers are used instead of one. A process can transfer data to or from one buffer while the operating system empties or fills the other buffer. This is also known as ***buffer swapping***. This can smooth out the flow of data between the I/O device and a process but it may be inadequate if the process performs rapid bursts of IO.



vi) Circular Buffer

The problem described for a double buffer can be alleviated by this. More than two buffers are used, with each individual buffer is one unit in a circular buffer. This is used when I/O operation must keep up with process



Buffering smoothes out the differences between the internal speeds of the computer system and the speeds of the I/O devices. The use of buffers also decouples the actual I/O transfer from the address space of the application. This allows the OS more flexibility in performing its memory management functions. With a variety of I/O and process activities to service as happens in a multiprogramming environment, buffering can increase the efficiency of the OS and the performance of individual processes.

But with enough demand eventually all buffers, even with multiple buffers, become full and their advantage is lost. The process will have to wait after processing each chunk of data.

Disk Scheduling

There is and has been a lot of work done in trying to bridge the gap between the speed of processors and main memory and the speed of disk access and in effect improving the performance of disk storage.

Disk Performance Parameters

The actual details of disk I/O operation depend on the computer system, the OS and the nature of the I/O channel and disk controller hardware

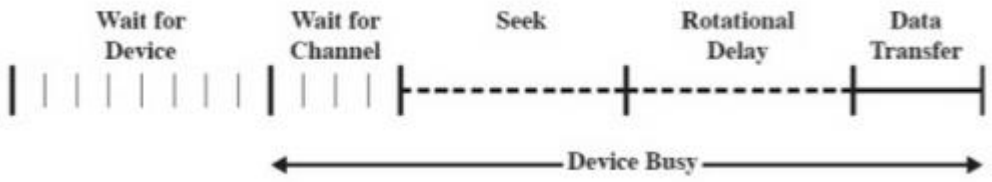


Figure 2: General Timing diagram of Disk I/O transfer

When the (HDD) disk drive is operating, the disk is rotating at constant speed.

To read/write, the head must be positioned at the desired track and at the beginning of the desired sector on that track.

Track selection involves moving the head in a movable-head system or electronically selecting one head on a fixed-head system. The time it takes to position the head at the track is known as the ***seek time***.

Rotational delay or rotational latency is the time it takes for the beginning of the sector to reach the head

Access Time is the time it takes to get into position to read/write.

Access time = Seek time + Rotational delay

Once the head is in position, the read/write operation is then performed as the sector moves under the head, which is the data transfer portion of the operation.

Transfer Time is the time taken to transfer the data

Aswell as the access time and transfer time, there are several queuing delays normally associated with a disk I/O operation.

> When a process issues an I/O request, it must first wait in a queue for the device to be available. Then the device is assigned to the process. If the device shares a single I/O channel or a set of I/O channels with other disk drives, there may be an additional wait for the channel to be available – the seek is performed to begin disk access.

There are ways to control over the data / sector placement for a file.

However, the OS has to deal with multiple I/O requests competing for the same files. Thus, it is important to study the disk scheduling policies

Disk Scheduling Policies

The reason for the difference in performance above can be traced to seek time. If sector access requests involve selection of tracks at random, then the performance of the disk I/O system will be as poor as possible. To improve matters, the average time spent on seeks needs to be lowered.

Name	Description	Remarks
Selection according to requestor		
RSS	Random scheduling	For analysis and simulation
FIFO	First in first out	Fairest of them all
PRI	Priority by process	Control outside of disk queue management
LIFO	Last in first out	Maximize locality and resource utilization
Selection according to requested item		
SSTF	Shortest service time first	High utilization, small queues
SCAN	Back and forth over disk	Better service distribution
C-SCAN	One way with fast return	Lower service variability
N-step-SCAN	SCAN of N records at a time	Service guarantee
FSCAN	N-step-SCAN with N = queue size at beginning of SCAN cycle	Load sensitive

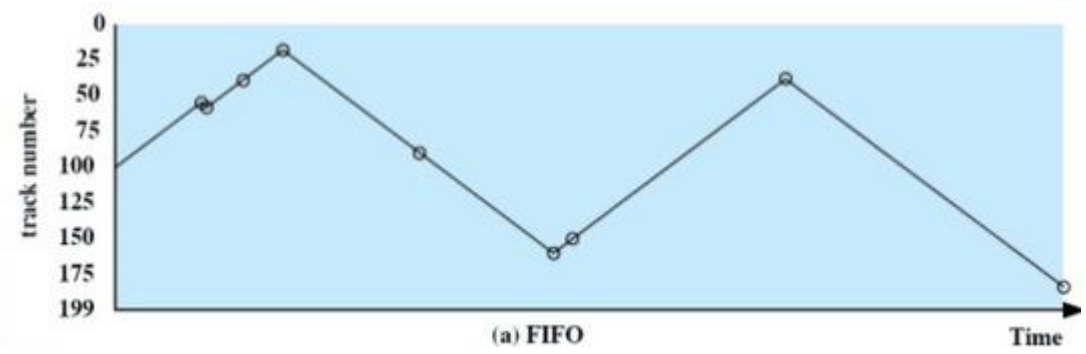
To compare various schemes, consider a disk head is initially located at track 100.

- Assume a disk with 200 tracks and that the disk request queue has random requests in it.
- The requested tracks, in the order received by the disk scheduler, are:

55, 58, 39, 18, 90, 160, 150, 38, 184

i) First-in, first-out (FIFO) Disk Scheduling

This is the simplest form of scheduling which processes items from the queue sequentially. It's fair to all processes. It's suitable is there are only a few processes that require access to the disk. But, it is similar to random scheduling in performance if there are many processes competing for the disk and therefore a more sophisticated scheduling policy would be more suitable.



ii) Priority

With this the control of the scheduling is outside the control of the disk management software. Its goal is not to optimise disk use but to meet other objectives within the OS. Short batch jobs and interactive jobs may have higher priority and so, priority scheduling provide good interactive response time

Short jobs will the sent through the system quickly but longer jobs may have to wait an excessively long time, so, for example, this would be a poor policy for database systems

iii) Last-in, first-out

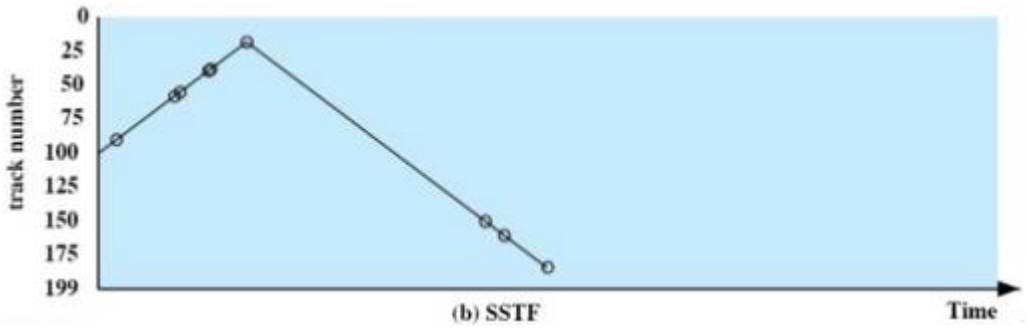
With LIFO, the device is given to the most recent user so there should be little arm movement for moving through the sequential file, and therefore this can improve throughput and reduce the queue lengths. Good for transaction processing systems. Possibility of starvation since a job may never regain the head of the line in the case of the disk being kept busy because of a large workload (it can't regain the head of the line unless the queue in front of it empties).

iv) Shortest Service Time First

This scheme selects the disk I/O request that requires the least movement of the disk arm from its current position

It will always choose the minimum seek time – this does not guarantee that the average seek time over a number of arm movements will be minimum. However, this should provide better performance that FIFO (a random tie-breaking algorithm is applied if two arms are equal distances.

Looking at the example here, the first track accessed is 90 because this is the closest request track to the starting position. The next tract accessed is 58 – it's the closest of the remaining requested tracks to the current position of 90 and so on



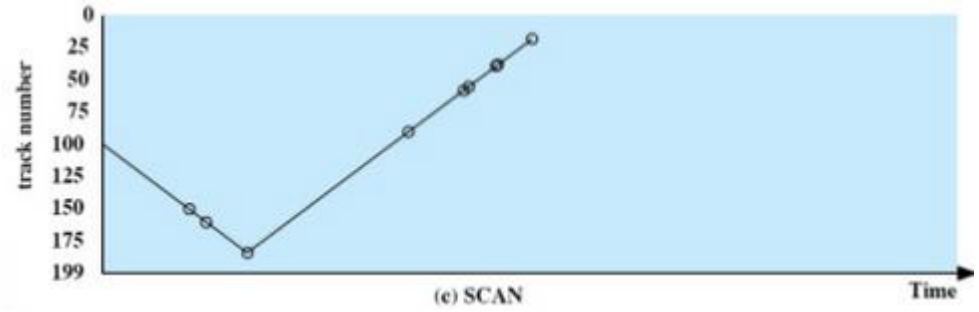
v) SCAN

With the exception of FIFO, all of the policies described so far can leave some request unfilled until the entire queue is emptied – there may always be new requests arriving that will be chosen before an existing request, which can lead to starvation.

SCAN algorithm is also known as the elevator algorithm because it operates like an elevator.

The arm moves in one direction only, satisfying all outstanding requests until it reaches the last track in that direction. Then the direction is reversed and the scan works in the opposite direction picking up all requests in order.

The policy behaves almost identically to the SSTF policy (remember that the example is static so we don't see any new processes being added to the queue).



vi) C-SCAN (Circular-SCAN)

This restricts scanning to one direction only. Thus 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.

This reduces the max. delay experienced by new requests.

vii) N-step-SCAN

In SSTF, SCAN and C-SCAN, it is possible that the arm may not move for a considerable period of time: if one or a few processes have high access rates to one track, they can monopolise the entire device by repeated request to that track.

To avoid this, the disk request queue can be segmented into sub-queues of length N, with one segment at a time being processed completely using SCAN.

New requests are added to other queue when queue is processed

viii) FSCAN

With this, two subqueues are created. When a scan begins, all of the requests are in one of the queues, with the other empty. All new requests are put into the other queue and service of the new requests is deferred until all of the old requests have been processed.

(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

Figure 3: Comparison of Disk Scheduling Algorithms

The aspect of I/O that has the greatest impact on overall system performance is disk I/O and accordingly, there has been great efforts and research put into this area.

Two of the most widely used approaches to improve disk I/O performance are ***disk scheduling*** and the ***disk cache***.

At any time there may be a queue of requests for I/O on the same disk. It is the object of the scheduling techniques to satisfy these requests in a way that minimises the mechanical seek time of the disk and therefore improve performance.