CSC 553 Operating Systems

Lecture 11 - I/O Management
and Disk Scheduling

Categories of I/O Devices

- External devices that engage in I/O with computer systems can be grouped into three categories:
  - Human readable
    - Suitable for communicating with the computer user
    - Printers, terminals, video display, keyboard, mouse
  - Machine readable
    - Suitable for communicating with electronic equipment
    - Disk drives, USB keys, sensors, controllers
  - Communication
    - Suitable for communicating with remote devices
    - Modems, digital line drivers

Differences in I/O Devices

- Devices differ in a number of areas:
  - **Data Rate** - There may be differences of magnitude between the data transfer rates
  - **Application** - The use to which a device is put has an influence on the software
  - **Complexity of Control** - The effect on the operating system is filtered by the complexity of the I/O module that controls the device

Differences in I/O Devices

- Devices differ in a number of areas:
  - **Unit of Transfer** - Data may be transferred as a stream of bytes or characters or in larger blocks
  - **Data Representation** - Different data encoding schemes are used by different devices
  - **Error Conditions** - The nature of errors, the way in which they are reported, their consequences, and the available range of responses differs from one device to another

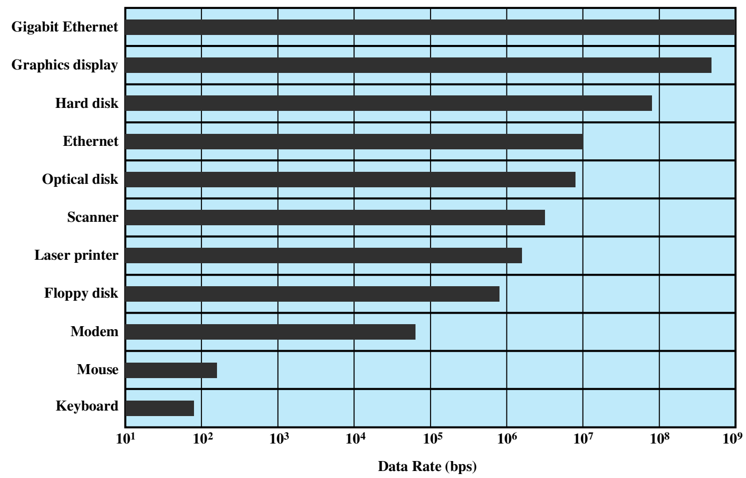


Figure 11.1 Typical I/O Device Data Rates

Organization of the I/O Function

- Three techniques for performing I/O are:
- Programmed I/O
- Interrupt-driven I/O
- Direct Memory Access (DMA)

Organization of the I/O Function

- Programmed I/O
  - The processor issues an I/O command on behalf of a process to an I/O module; that process then busy waits for the operation to be completed before proceeding
- Direct Memory Access (DMA)
    - A DMA module controls the exchange of data between main memory and an I/O module

Organization of the I/O Function

- **Interrupt-driven I/O**
  - The processor issues an I/O command on behalf of a process
    - **If non-blocking** – processor continues to execute instructions from the process that issued the I/O command
    - **If blocking** – the next instruction the processor executes is from the OS, which will put the current process in a blocked state and schedule another process

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)

Evolution of the I/O Function

1. Processor directly controls a peripheral device
2. A controller or I/O module is added
3. Same configuration as step 2, but now interrupts are employed
4. The I/O module is given direct control of memory via DMA
5. The I/O module is enhanced to become a separate processor, with a specialized instruction set tailored for I/O
6. The I/O module has a local memory of its own and is, in fact, a computer in its own right

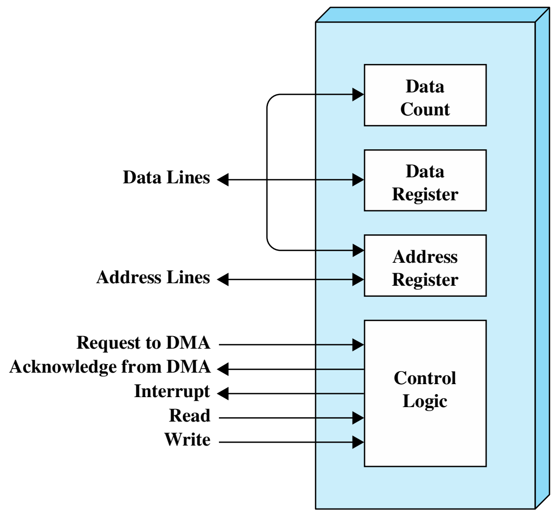


Figure 11.2 Typical DMA Block Diagram

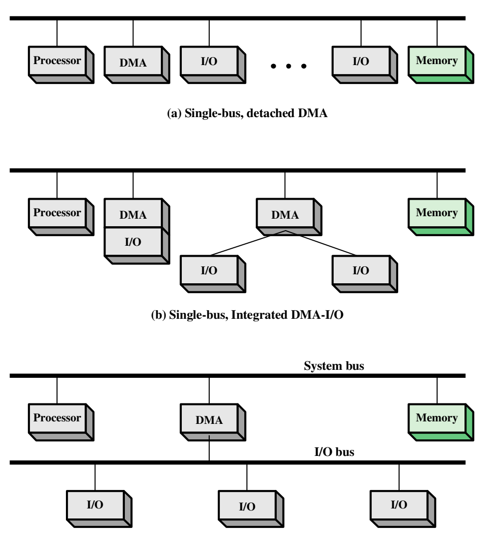


Figure 11.3 Alternative DMA Configurations

Design Objectives

- Efficiency
  - Major effort in I/O design
  - Important because I/O operations often form a bottleneck
  - Most I/O devices are extremely slow compared with main memory and the processor
  - The area that has received the most attention is disk I/O

Design Objectives

- Generality
  - Desirable to handle all devices in a uniform manner
  - Applies to the way processes view I/O devices and the way the operating system manages I/O devices and operations
  - Diversity of devices makes it difficult to achieve true generality
  - Use a hierarchical, modular approach to the design of the I/O function

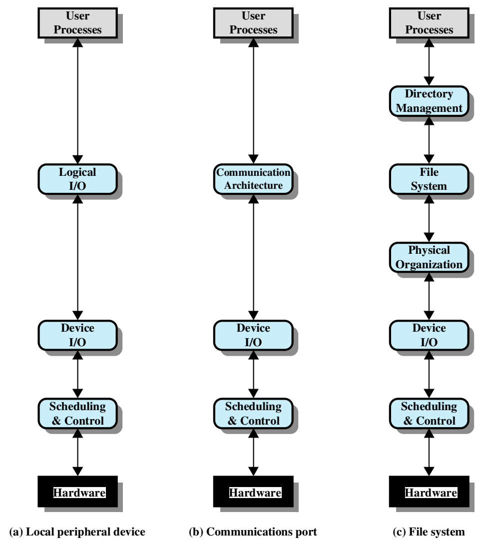


Figure 11.4 A Model of I/O Organization

Buffering

- To avoid overheads and inefficiencies, it is sometimes convenient to perform input transfers in advance of requests being made, and to perform output transfers some time after the request is made

Buffering – Block-Oriented Device

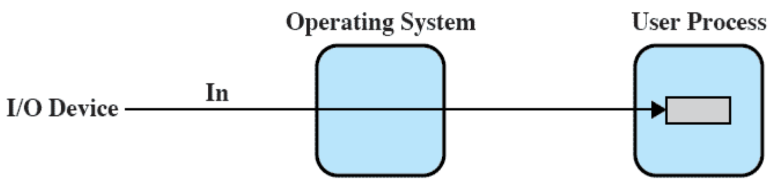
- Stores information in blocks that are usually of fixed size
- Transfers are made one block at a time
- Possible to reference data by its block number
- Disks and USB keys are examples

Buffering – Stream-Oriented Device

- Transfers data in and out as a stream of bytes
- No block structure
- Terminals, printers, communications ports, mouse and other pointing devices, and most other devices that are not secondary storage are examples

No Buffer

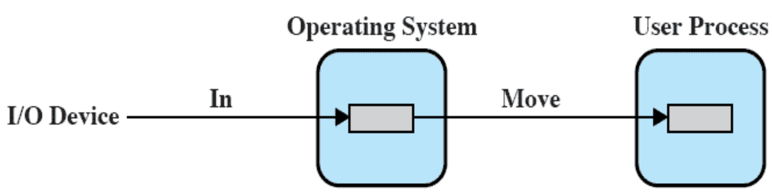
- Without a buffer, the OS directly accesses the device when it needs



(a) No buffering

Single Buffer

- The simplest type of support that the operating system can provide
- When a user process issues an I/O request, the OS assigns a buffer in the system portion of main memory to the operation



(b) Single buffering

Single Buffer

- Input transfers are made to the system buffer
- Reading ahead/anticipated input
  - Is done in the expectation that the block will eventually be needed
  - When the transfer is complete, the process moves the block into user space and immediately requests another block

Single Buffer

- Approach generally provides a speedup compared to the lack of system buffering
  - 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 system memory rather than user process memory
- Disadvantages:
  - Complicates the logic in the operating system
  - Swapping logic is also affected

Single Buffering for Stream-Oriented I/O

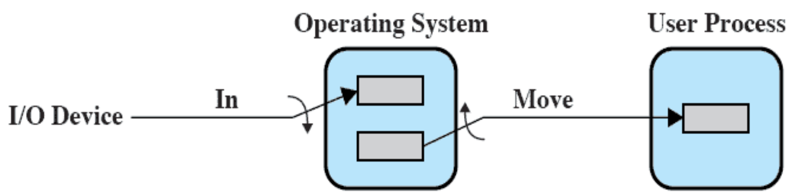
- Can be used in a line-at-a-time fashion or a byte-at-a-time fashion
- Line-at-a-time operation is appropriate for scroll-mode terminals (dumb terminals)
  - With this form of terminal, user input is one line at a time, with a carriage return signaling the end of a line
  - Output to the terminal is similarly one line at a time

Single Buffering for Stream-Oriented I/O

- Byte-at-a-time operation is used on forms-mode terminals, when each keystroke is significant and for many other peripherals, such as sensors and controllers

Double Buffer

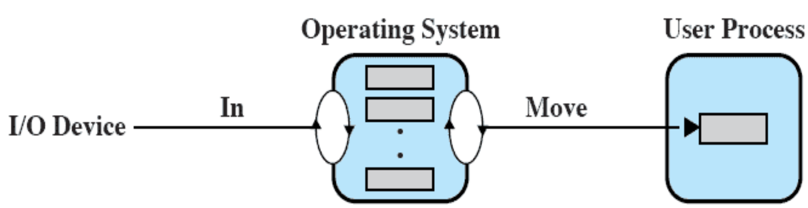
- Assigning two system buffers to the operation
- A process now transfers data to or from one buffer while the operating system empties or fills the other buffer
- Also known as buffer swapping



(c) Double buffering

Circular Buffer

- When more than two buffers are used, the collection of buffers is itself referred to as a circular buffer
- Each individual buffer is one unit in the circular buffer



(d) Circular buffering

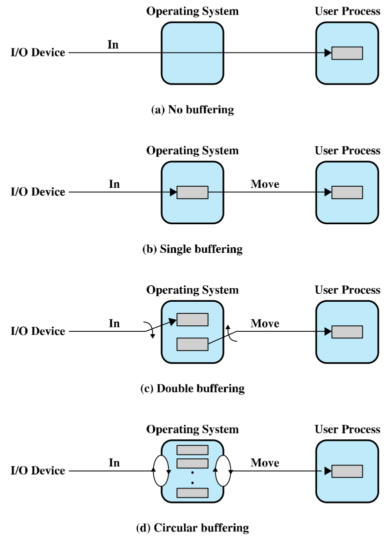


Figure 11.5 I/O Buffering Schemes (input)

Disk Performance Parameters

- The actual details of disk I/O operation depend on the:
  - Computer system
  - Operating system
  - Nature of the I/O channel and disk controller hardware

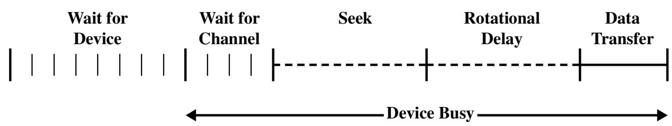


Figure 11.6 Timing of a Disk I/O Transfer

Disk Performance Parameters

- When the disk drive is operating, the disk is rotating at constant speed
- To read or 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

Disk Performance Parameters

- On a movable-head system the time it takes to position the head at the track is known as ***seek time***
- The time it takes for the beginning of the sector to reach the head is known as ***rotational delay***
- The sum of the seek time and the rotational delay equals the ***access time***

Seek Time

- The time required to move the disk arm to the required track
- Consists of two key components:
  - The initial startup time
  - The time taken to traverse the tracks that have to be crossed once the access arm is up to speed
- Settling time
  - Time after positioning the head over the target track until track identification is confirmed

Seek Time

- Much improvement comes from smaller and lighter disk components
- A typical average seek time on contemporary hard disks is under 10ms

Disk Performance

- Rotational delay
  - The time required for the addressed area of the disk to rotate into a position where it is accessible by the read/write head
  - Disks rotate at speeds ranging from 3,6000 rpm (for handheld devices such as digital cameras) up to 15,000 rpm

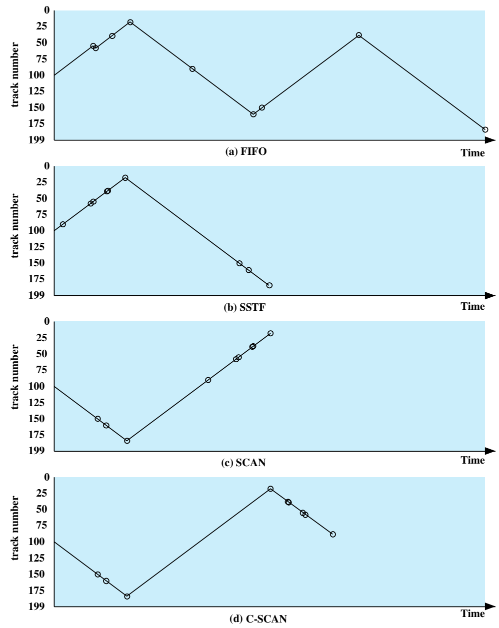
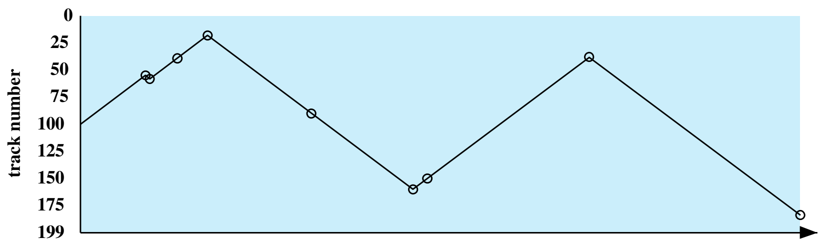


Figure 11.7 Comparison of Disk Scheduling Algorithms (see Table 11.3)

(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

First-In, First-Out

- Processes in sequential order
- Fair to all processes
- Approximates random scheduling in performance if there are many processes competing for the disk



(a) FIFO

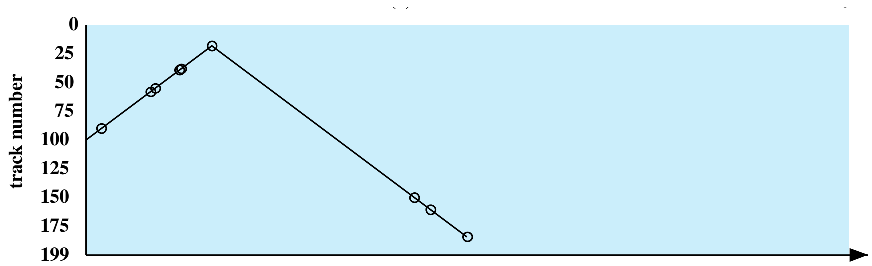
Name	Description	Remarks
Selection according to requestor		
Random	Random scheduling	For analysis and simulation
FIFO	First in first out	Fairest of them all
PRI	Priority by process	Control outside of disk queue management
LIFO	Last in first out	Maximize locality and resource utilization
Selection according 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

Priority (PRI)

- Control of the scheduling is outside the control of disk management software
- Goal is not to optimize disk utilization but to meet other objectives
- Short batch jobs and interactive jobs are given higher priority
- Provides good interactive response time
- Longer jobs may have to wait an excessively long time
- A poor policy for database systems

Shortest Service Time First

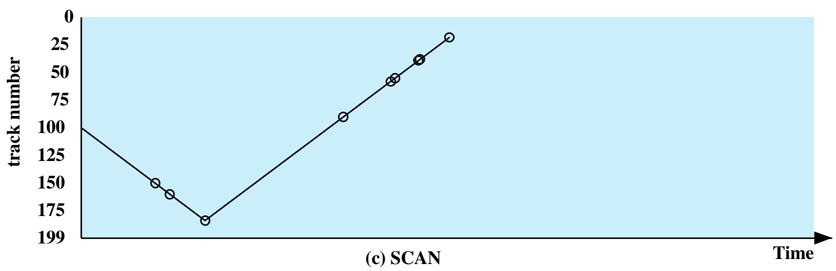
- Select the disk I/O request that requires the least movement of the disk arm from its current position
- Always choose the minimum seek time



(b) SSTF

SCAN

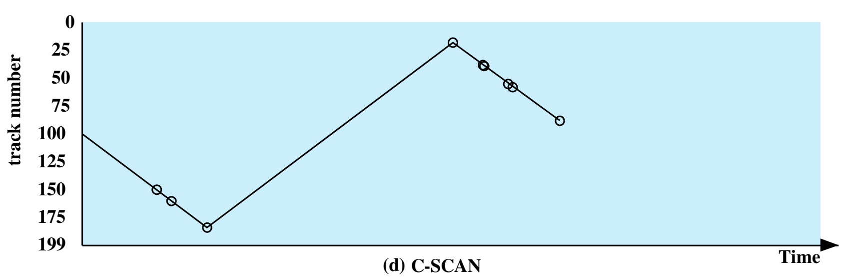
- Also known as the elevator algorithm
- Arm moves in one direction only
  - Satisfies all outstanding requests until it reaches the last track in that direction then the direction is reversed
  - Favors jobs whose requests are for tracks nearest to both innermost and outermost tracks and favors the latest-arriving jobs



(c) SCAN

C-SCAN (Circular 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



(d) C-SCAN

N-Step-SCAN

- Segments the disk request queue into subqueues of length *N*
- Subqueues are processed one at a time, using SCAN
- While a queue is being processed new requests must be added to some other queue
- If fewer than *N* requests are available at the end of a scan, all of them are processed with the next scan

FSCAN

- Uses two subqueues
- When a scan begins, all of the requests are in one of the queues, with the other empty
- During scan, all new requests are put into the other queue
- Service of new requests is deferred until all of the old requests have been processed

RAID

- Redundant Array of Independent Disks
  - Consists of seven levels, zero through six
- Design architectures share three characteristics:
  - RAID is a set of physical disk drives viewed by the operating system as a single logical drive
  - Data are distributed across the physical drives of an array in a scheme known as striping
  - Redundant disk capacity is used to store parity information, which guarantees data recoverability in case of a disk failure

RAID

- The term was originally coined in a paper by a group of researchers at the University of California at Berkeley
  - The paper outlined various configurations and applications and introduced the definitions of the RAID levels
- Strategy employs multiple disk drives and distributes data in such a way as to enable simultaneous access to data from multiple drives
  - Improves I/O performance and allows easier incremental increases in capacity

RAID

- The unique contribution is to address effectively the need for redundancy
- Makes use of stored parity information that enables the recovery of data lost due to a disk failure

Category	Level	Description	Disks required	Data availability	Large I/O data transfer capacity	Small I/O request rate
Striping	0	Nonredundant	N	Lower than single disk	Very high	Very high for both read and write
Mirroring	1	Mirrored	$2N$	Higher than RAID 2, 3, 4, or 5; lower than RAID 6	Higher than single disk for read; similar to single disk for write	Up to twice that of a single disk for read; similar to single disk for write
Parallel access	2	Redundant via Hamming code	$N + m$	Much higher than single disk; comparable to RAID 3, 4, or 5	Highest of all listed alternatives	Approximately twice that of a single disk
	3	Bit-interleaved parity	$N + 1$	Much higher than single disk; comparable to RAID 2, 4, or 5	Highest of all listed alternatives	Approximately twice that of a single disk
Independent access	4	Block-interleaved parity	$N + 1$	Much higher than single disk; comparable to RAID 2, 3, or 5	Similar to RAID 0 for read; significantly lower than single disk for write	Similar to RAID 0 for read; significantly lower than single disk for write
	5	Block-interleaved distributed parity	$N + 1$	Much higher than single disk; comparable to RAID 2, 3, or 4	Similar to RAID 0 for read; lower than single disk for write	Similar to RAID 0 for read; generally lower than single disk for write
	6	Block-interleaved dual distributed parity	$N + 2$	Highest of all listed alternatives	Similar to RAID 0 for read; lower than RAID 5 for write	Similar to RAID 0 for read; significantly lower than RAID 5 for write

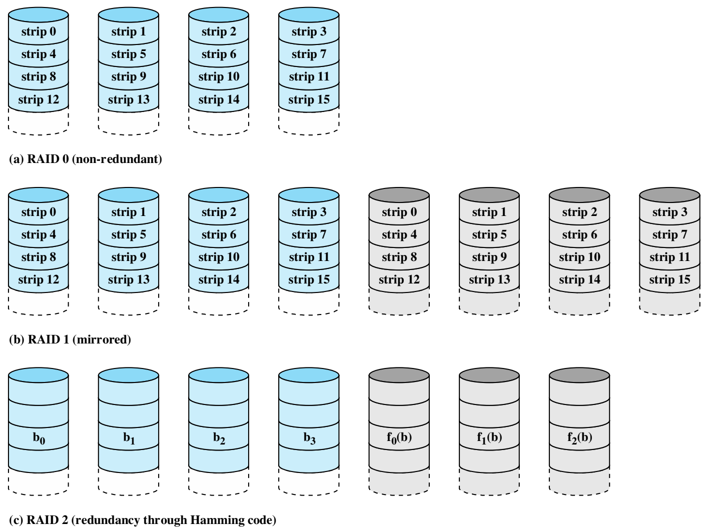


Figure 11.8 RAID Levels (page 1 of 2)

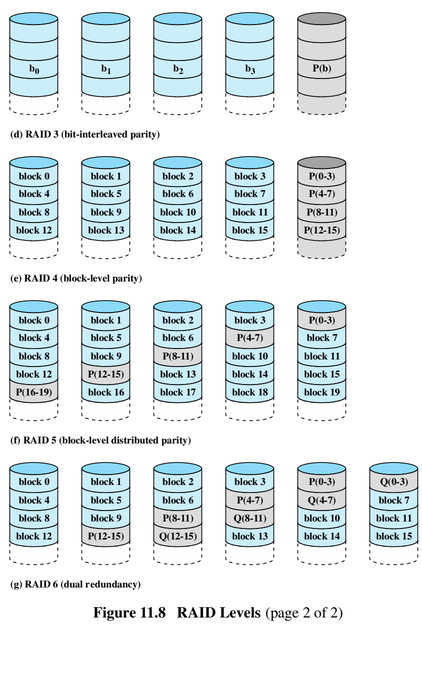
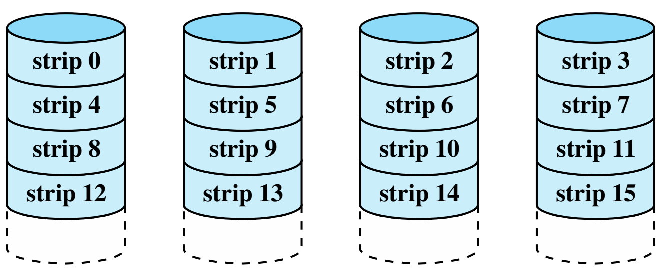


Figure 11.8 RAID Levels (page 2 of 2)

RAID Level 0

- Not a true RAID because it does not include redundancy to improve performance or provide data protection
- User and system data are distributed across all of the disks in the array
- Logical disk is divided into strips

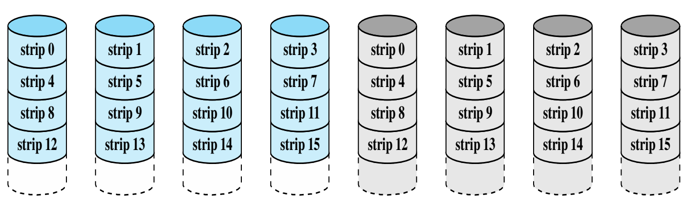


(a) RAID 0 (non-redundant)

RAID Level 1

- Redundancy is achieved by the simple expedient of duplicating all the data
- There is no "write penalty"
- When a drive fails the data may still be accessed from the second drive
- Principal disadvantage is the cost

RAID Level 1

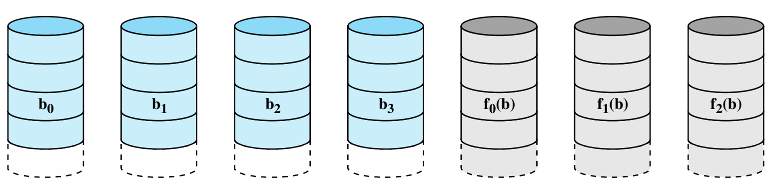


(b) RAID 1 (mirrored)

RAID Level 2

- Makes use of a parallel access technique
- Data striping is used
- Typically a Hamming code is used
- Effective choice in an environment in which many disk errors occur

RAID Level 2

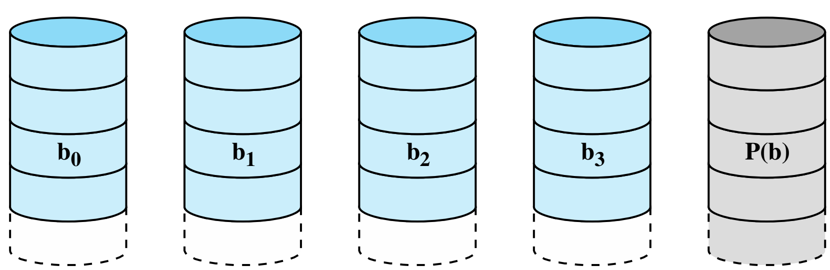


(c) RAID 2 (redundancy through Hamming code)

RAID Level 3

- Requires only a single redundant disk, no matter how large the disk array
- Employs parallel access, with data distributed in small strips
- Can achieve very high data transfer rates

RAID Level 3

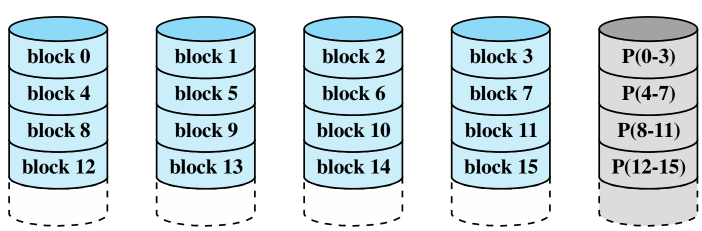


(d) RAID 3 (bit-interleaved parity)

RAID Level 4

- Makes use of an independent access technique
- A bit-by-bit parity strip is calculated across corresponding strips on each data disk, and the parity bits are stored in the corresponding strip on the parity disk
- Involves a write penalty when an I/O write request of small size is performed

RAID Level 4

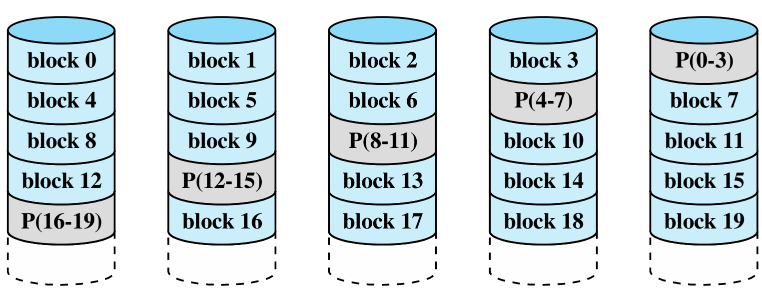


(e) RAID 4 (block-level parity)

RAID Level 5

- Similar to RAID-4 but distributes the parity bits across all disks
- Typical allocation is a round-robin scheme
- Has the characteristic that the loss of any one disk does not result in data loss

RAID Level 5

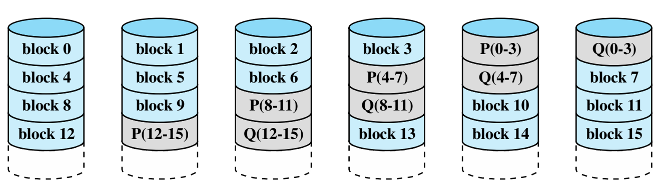


(f) RAID 5 (block-level distributed parity)

RAID Level 6

- Two different parity calculations are carried out and stored in separate blocks on different disks
- Provides extremely high data availability
- Incurs a substantial write penalty because each write affects two parity blocks

RAID Level 6

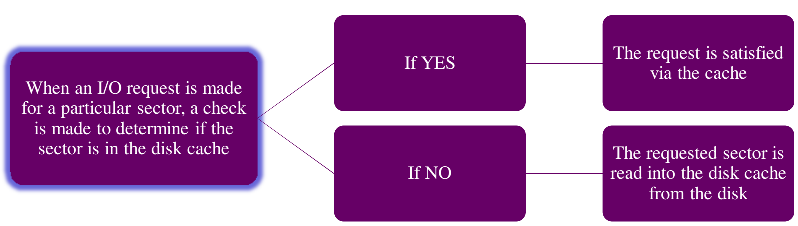


(g) RAID 6 (dual redundancy)

Disk Cache

- *Cache memory* is used to apply to a memory that is smaller and faster than main memory and that is interposed between main memory and the processor
- Reduces average memory access time by exploiting the principle of locality
- *Disk cache* is a buffer in main memory for disk sectors
- Contains a copy of some of the sectors on the disk

Disk Cache



Least Recently Used (LRU)

- Most commonly used algorithm that deals with the design issue of replacement strategy
- The block that has been in the cache the longest with no reference to it is replaced
- A stack of pointers reference the cache
  - Most recently referenced block is on the top of the stack
  - When a block is referenced or brought into the cache, it is placed on the top of the stack

Least Frequently Used

- The block that has experienced the fewest references is replaced
- A counter is associated with each block
- Counter is incremented each time block is accessed
- When replacement is required, the block with the smallest count is selected

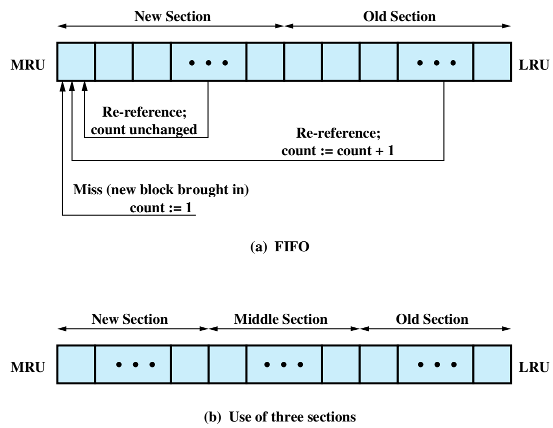


Figure 11.9 Frequency-Based Replacement

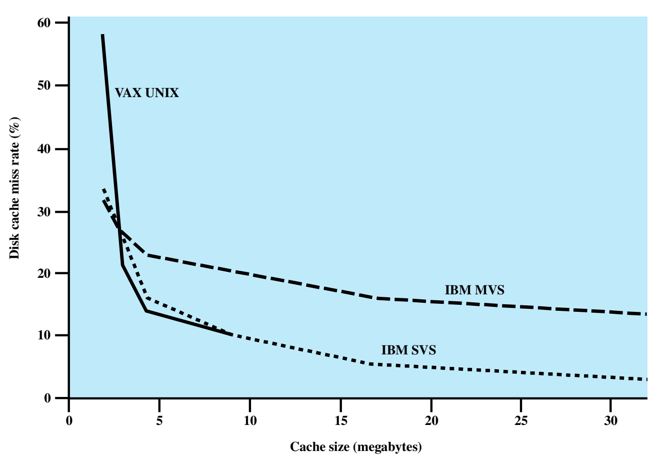


Figure 11.10 Some Disk Cache Performance Results Using LRU

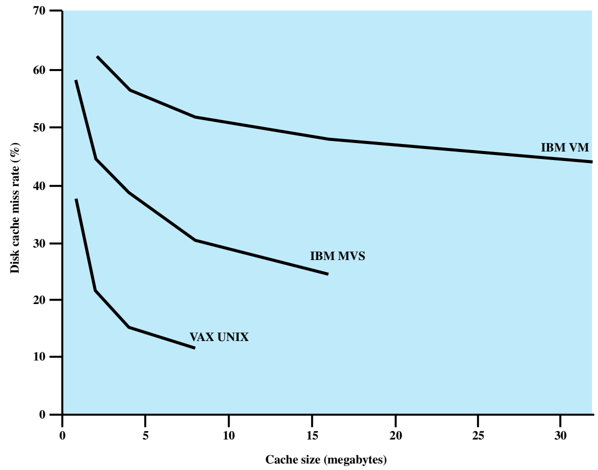


Figure 11.11 Disk Cache Performance Using Frequency-Based Replacement [ROBI90]

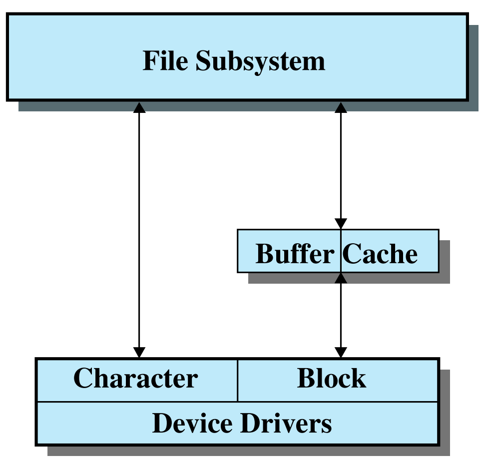


Figure 11.12 UNIX I/O Structure

UNIX Buffer Cache

- Is essentially a disk cache
  – I/O operations with disk are handled through the buffer cache
- The data transfer between the buffer cache and the user process space always occurs using DMA
  – Does not use up any processor cycles
  – Does consume bus cycles

UNIX Buffer Cache

- Three lists are maintained:
  – Free list
    - List of all slots in the cache that are available for allocation
  – Device list
    - List of all buffers currently associated with each disk
  – Driver I/O queue
    - List of buffers that are actually undergoing or waiting for I/O on a particular device

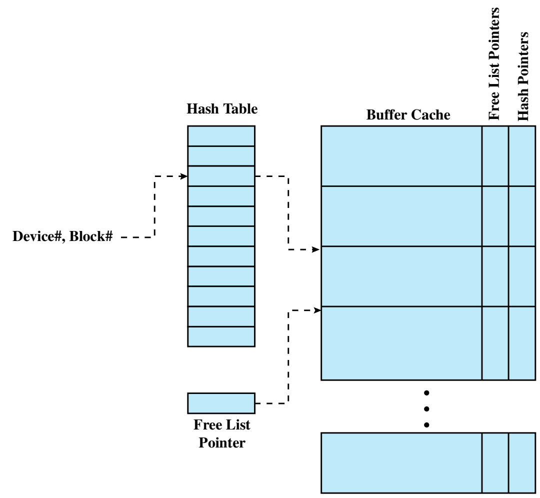


Figure 11.13 UNIX Buffer Cache Organization

Character Queue

Used by character oriented devices

Terminals and printers

Either written by the I/O device and read by the process or vice versa

Producer/consumer model is used

Character queues may only be read once

As each character is read, it is effectively destroyed

Unbuffered I/O

- Is simply DMA between device and process space
- Is always the fastest method for a process to perform I/O
- Process is locked in main memory and cannot be swapped out
- I/O device is tied up with the process for the duration of the transfer making it unavailable for other processes

	Unbuffered I/O	Buffer Cache	Character Queue
Disk drive	X	X	
Tape drive	X	X	
Terminals			X
Communication lines			X
Printers	X		X