OPERATING SYSTEMS - BCS303

Solution

Sub:	Operating System - OS					Sub Code:	BCS303	Branch:	CSE
Date:	06/02/2026	Duration:	180 mins	Max Marks:	100	Sem / Sec:	3 A,B,C		

1.a. Solution	Explain various types of system calls with an example for each. System calls are the interface between a **user program** and the **Operating System (OS)**. They allow a program to request services such as file handling, process control, memory management, etc. System calls are broadly classified into the following types: **Process Control System Calls** These are used to create, execute, and terminate processes. **Functions:** ● Create a process ● End a process ● Load and execute a program ● Wait for a process **Process Control System Calls** These are used to create, execute, and terminate processes. **Functions:** ● Create a process

	• End a process • Load and execute a program • Wait for a process **File Management System Calls** Used to create, open, read, write, and delete files. **Functions:** • Create file • Open file • Read/write file • Close file **Examples:** • open() • read() • write() • close() **Device Management System Calls** Used to interact with hardware devices. **Functions:** • Request device • Release device • Read/write device • Get/set device attributes **Examples:**

	• ioctl() • read() • write() **Information Maintenance System Calls** Used to get or set system and process information. **Functions:** • Get time/date • Get system data • Set system data • Get process attributes • **Examples:** • getpid() • time() • uname() • **communication System Calls** Used for inter-process communication (IPC). **Functions:** • Create communication connection • Send/receive messages • Share memory **Examples:** • pipe() • msgget() • shmget() • socket() **Memory Management System Calls**

1.b	Used to allocate and free memory. **Functions:** ● Allocate memory ● Free memory ● Map memory **Examples:** ● brk() ● mmap() ● munmap() Explain cloud computing its tyes and their services it offers? **Cloud Computing** **Cloud Computing** is the delivery of computing services such as servers, storage, databases, networking, and software over the internet instead of using local computers or physical servers. Instead of buying and maintaining hardware, users can access resources online and pay only for what they use. Example providers: ● Amazon Web Services (AWS) ● Microsoft Azure ● Google Cloud Platform **Public Cloud** ● Owned and operated by third-party providers. ● Resources are shared among multiple users. ● Accessible through the internet. **Private Cloud**

- Used by a single organization.
- Can be hosted on-premise or by a third party.
- Offers higher security and control.

hybrid Cloud

- Combination of Public and Private cloud.
- Data and applications can move between both.

service Models of Cloud Computing

Cloud provides services in **three main models**:

IaaS (Infrastructure as a Service)

Provides virtualized computing resources over the internet.

PaaS (Platform as a Service)

Provides platform and environment to develop, test, and deploy applications.

SaaS (Software as a Service)

Provides ready-to-use software over the internet.

services Offered by Cloud Computing

Cloud computing offers:

1. Storage services (Google Drive, cloud storage)
2. Computing power (virtual machines)
3. Database services
4. Networking services
5. Security services
6. Backup and disaster recovery

1.c	7. Application hosting 8. AI and Machine Learning services What is dual mode operation and what is the need of it? **Dual Mode Operation** **Dual Mode Operation** is a feature of an Operating System that allows the computer to operate in **two different modes**: 1 **User Mode** 2**Kernel Mode (Supervisor Mode)** The mode is controlled by a special hardware mechanism called the **mode bit**. **User Mode** ● Used for running **application programs**. ● Limited access to system resources. ● Cannot execute privileged instructions. **Example:** When you open a browser like Google Chrome, it runs in **user mode**. ✔ Safe ✔ Restricted access **Kernel Mode** ● Used by the **Operating System**. ● Has full access to hardware and memory. ● Can execute privileged instructions. Example OS: Linux Complete control Can manage memory, CPU, devices ● The CPU has a **mode bit**:

2.a	○ `0` → Kernel Mode ○ `1` → User Mode ● When a program requests OS service (via **system call**), control switches from **User Mode → Kernel Mode**. ● After service is completed, it returns to **User Mode**. ● **Need for Dual Mode Operation** Dual mode operation is needed for: **Security** Prevents user programs from directly accessing hardware or critical memory. **Protection** Stops malicious or faulty programs from damaging the system. **Controlled Resource Access** Only the OS can allocate memory, control devices, or manage CPU scheduling. **System Stability** Prevents system crashes due to unauthorized instructions. Explain the different architecture of OS starting from Simple Structure, Layered Structure, Micro Kennels, Modules and Hybrid Systems.? Operating System (OS) architecture defines how different components of the OS are organized and how they interact with hardware and users. Below are the major OS architectures: **Simple Structure** **Explanation:** ● The OS is written as a **single large program**. ● No clear separation between components.

- All system services run in **kernel mode**.

Characteristics:

- Fast execution
- Poor modularity
- Difficult to maintain

Example:

- MS-DOS

In MS-DOS, applications could directly access hardware, so there was **no strong protection mechanism**.

Layered Structure

Explanation:

- OS is divided into **multiple layers**.
- Each layer performs a specific function.
- Each layer uses services of the lower layer.

Characteristics:

- Easy to debug
- Better modularity
- Slower due to layer communication

Example:

- THE operating system

Explanation:

- Only essential services run in kernel.
- Other services run in **user space**.
- Communication happens via **message passing**.

Kernel Contains:

- CPU scheduling
- Memory management
- Inter-process communication
- **Advantages:**

✔ More secure
✔ Reliable
✔ Easy to extend

Example:

- MINIX
- Mach

Modular Structure (Modules)

Explanation:

- Uses **loadable kernel modules**.
- Core kernel + dynamically added modules.
- Combines monolithic speed with modular flexibility.
- **Advantages:**

✔ Flexible
✔ Efficient
✔ Easy to update modules **Example:**

- Linux

Linux allows modules like device drivers to be loaded or removed without rebooting.

Hybrid Systems

- Combination of different architectures.
- Uses microkernel ideas but keeps some services in kernel space for performance.

2.b.	**Advantages:** ✔ Balanced performance ✔ Better security ✔ Practical implementation**Examples:** ● Windows NT ● macOS These systems combine monolithic and microkernel concepts. Discuss the essential properties of the following types of systems : i. Time sharing system ii. Multi-programmed batch systems. A **Time Sharing System** is an operating system that allows multiple users to access a computer system simultaneously by sharing CPU time. The CPU time is divided into small intervals called *time slices* or *quantum*, and each process is executed for a short duration in a cyclic order. This rapid switching between processes creates the illusion that all users are working at the same time. Time-sharing systems are interactive and provide quick response to user commands. **Essential Properties:** 1. **Multi-user environment** – Supports multiple users at the same time. 2. **Time slicing** – CPU time is divided among processes. 3. **Fast response time** – Immediate feedback to users. 4. **Context switching** – CPU switches quickly between processes. 5. **Fair resource allocation** – Equal opportunity for all users. 6. **Interactive processing** – Users directly interact via terminals. 7. **High CPU utilization** – CPU rarely remains idle. 8. **Preemptive scheduling** – OS can interrupt a running process. **Example:**

Operating systems like UNIX and Linux use time-sharing concepts.

A **Multi-programmed Batch System** is an operating system that keeps multiple jobs in memory simultaneously and executes them without user interaction. Users submit jobs in batches, and the system processes them one by one. When one job waits for I/O operations, the CPU switches to another job, thereby improving system efficiency. The main objective is to maximize CPU utilization and increase throughput.

Essential Properties:

1. **Multiple jobs in memory** – Several programs loaded simultaneously.
2. **Batch processing** – Jobs are grouped and processed together.
3. **No direct user interaction** – Non-interactive system.
4. **CPU scheduling** – CPU switches when a job waits for I/O.
5. **High throughput** – Large number of jobs completed.
6. **Efficient resource utilization** – Minimizes idle CPU time.
7. **Long turnaround time** – Output is received after processing completes.
8. **Automatic job sequencing** – Jobs executed in predefined order.

Example:

Early mainframe operating systems such as IBM OS/360 followed this approach.

3.a. Explain inter process communication?

Inter Process Communication (IPC) is a mechanism that allows processes to communicate and synchronize with each other. Since processes run independently in memory, IPC enables them to exchange data and coordinate execution. It is necessary in multi-tasking operating systems where multiple processes run simultaneously and need to share information.

Need for IPC

1. Information sharing between processes
2. Process synchronization
3. Resource sharing

4. Modularity and cooperation among processes

Types of IPC

Shared Memory

In this method, two or more processes share a common memory area. Processes can read and write data in this shared space. It is faster because data is not copied between processes, but synchronization mechanisms like semaphores are required to avoid data inconsistency.

Example: `shmget()` system call in Linux

Message Passing

In this method, processes communicate by sending and receiving messages. The operating system manages the communication. It is easier to implement but slightly slower than shared memory.

Example: `pipe()`, `msgget()` system calls in UNIX

3. b.Discuss Multi level Queue Scheduling Algorithm.?

Multi-Level Queue (MLQ) Scheduling is a CPU scheduling algorithm in which the ready queue is divided into multiple separate queues based on process priority or type. Each queue has its own scheduling algorithm, and processes are permanently assigned to a specific queue depending on their characteristics (such as system process, interactive process, batch process, etc.).

Features:

1 **Multiple Ready Queues** – The ready queue is divided into different queues.
2**Fixed Assignment** – A process is assigned to one queue permanently.
3**Different Scheduling Algorithms** – Each queue may use a different algorithm (e.g., Round Robin for interactive jobs and FCFS for batch jobs).
4**Priority Between Queues** – Queues are scheduled based on priority. Higher-priority queues are served first.
5**No Movement Between Queues** – Processes do not move from one queue to another.

Scheduling Between Queues:

There are two methods:

- **Fixed Priority Scheduling** – Higher-priority queue is executed first. Lower queues execute only when higher ones are empty.

- **Time Slicing Between Queues** – Each queue gets a certain percentage of CPU time.

Advantages:

- Suitable for systems with different types of processes
- Ensures priority-based execution
- Simple to implement

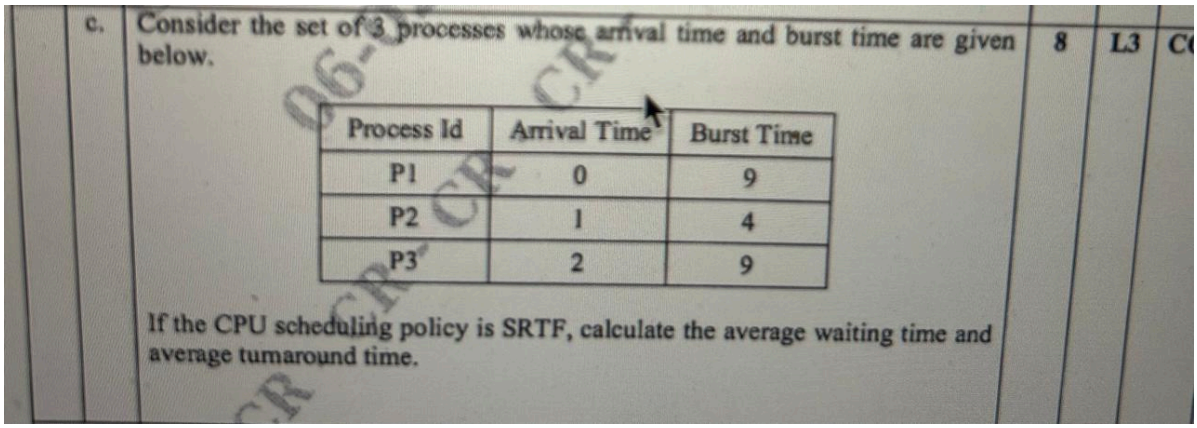
Disadvantages:

- Possibility of starvation for lower-priority processes
- Inflexible (no movement between queues)

eg:-

- System processes → Highest priority
- Interactive processes → Medium priority (Round Robin)
- Batch processes → Lowest priority (FCFS).

3.c.



c. Consider the set of 3 processes whose arrival time and burst time are given below. 8 L3 C(

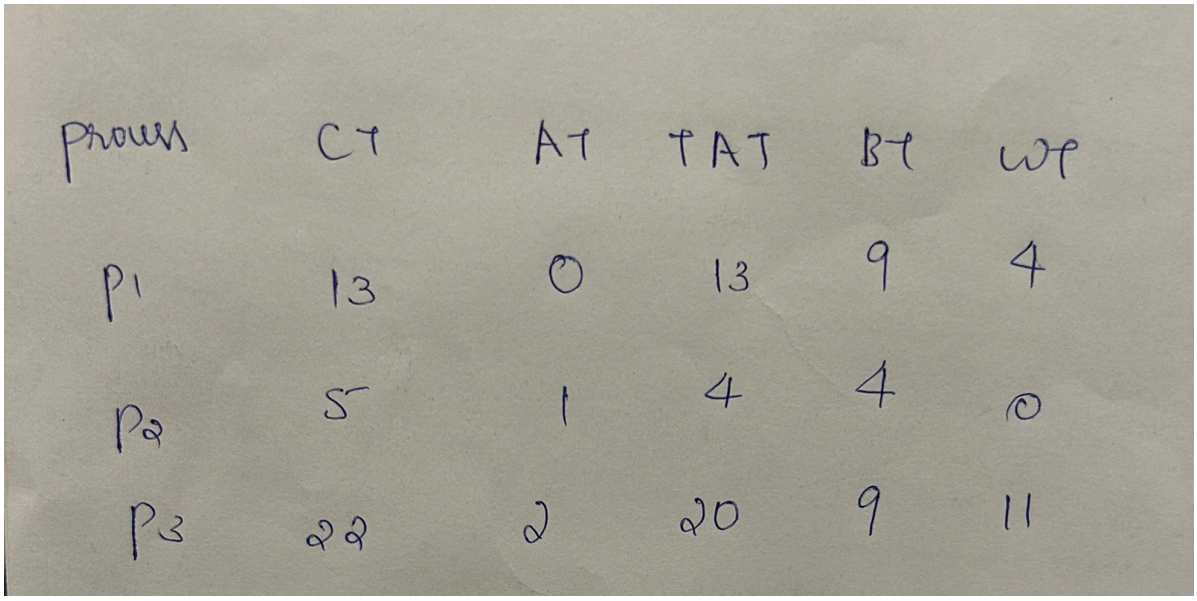
Process Id	Arrival Time	Burst Time
P1	0	9
P2	1	4
P3	2	9

If the CPU scheduling policy is SRTF, calculate the average waiting time and average turnaround time.

Gantt chart

0 1 5 13 22

| P1 | P2 | P1 | P3 |



Process	CT	AT	TAT	BT	WT
P1	13	0	13	9	4
P2	5	1	4	4	0
P3	22	2	20	9	11

Average Waiting Time

(4+0+11)/3=15/3=5

Average Turnaround Time

(13+4+20)/3=37/3=12.33

4.a. Compare User level threads and Kernel level threads. 4M

User Level Threads	Kernel Level Threads
Managed by user thread library	Managed by operating system kernel
No system call required	Requires system calls
If one thread blocks, entire process blocks	If one thread blocks, others can continue
Implemented in user space	Implemented in kernel space

4.b. Illustrate with a neat sketch, process states and process control block(PCB). 8M

Process State

A Process has 5 states. Each process may be in one of the following states –

1. New - The process is in the stage of being created.

2. Ready - The process has all the resources it needs to run. It is waiting to be assigned to the processor.

3. Running – Instructions are being executed.

4. Waiting - The process is waiting for some event to occur. For example, the process may be waiting for keyboard input, disk access request, inter-process messages, a timer to go off, or a child process to finish.

5. Terminated - The process has completed its execution.

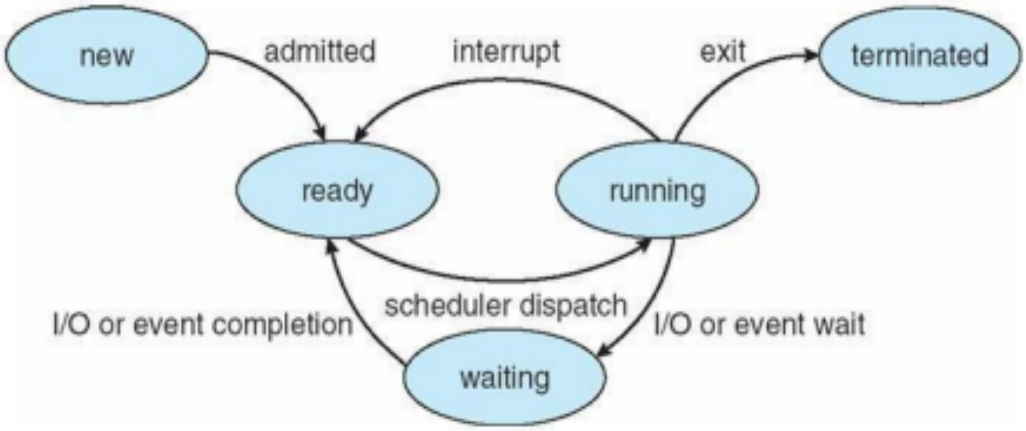


Figure: Diagram of process state

Process Control Block

For each process there is a Process Control Block (PCB), which stores the process-specific information as shown below –

Process State – The state of the process may be new, ready, running, waiting, and so on.

Program counter – The counter indicates the address of the next instruction to be executed for

this process.

CPU registers - The registers vary in number and type, depending on the computer architecture. They include accumulators, index registers, stack pointers, and general-purpose registers. Along with the program counter, this state information must be saved when an interrupt occurs, to allow the process to be continued correctly afterward.

CPU scheduling information- This information includes a process priority, pointers to scheduling queues, and any other scheduling parameters.

Memory-management information – This includes information such as the value of the base and limit registers, the page tables, or the segment tables.

Accounting information – This information includes the amount of CPU and real time used, time limits, account numbers, job or process numbers, and so on.

I/O status information – This information includes the list of I/O devices allocated to the process, a list of open files, and so on.

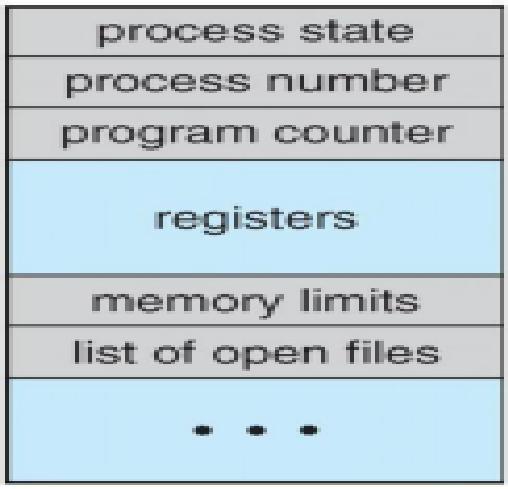


Figure: Process control block (PCB)

4.C.Consider the set of 6 processes, whose arrival time and burst time are given below.

Process ID	Arrival Time	Burst Time
P_1	5	5
P_2	4	6
P_3	3	7
P_4	1	9
P_5	2	2
P_6	6	3

If the CPU scheduling policy is Round Robin with time quantum = 3, calculate Average Waiting time and Turnaround time. 8M

Step 1: Gantt Chart (Round Robin Execution)

Code

```
0   1   4   6   9   12  15  18  21  24  27  30  32  33
|   P4  |P5 |P3 |P2 |P4 |P1 |P6 |P3 |P2 |P4 |P1 |P3 |
```

Step 3: Turnaround Time (TAT)

$$TAT = CT - AT$$

Process	CT	AT	TAT
P1	32	5	27
P2	27	4	23
P3	33	3	30
P4	30	1	29
P5	6	2	4
P6	21	6	15

Step 4: Waiting Time (WT)

$$WT = TAT - BT$$

Process	TAT	BT	WT
P1	27	5	22
P2	23	6	17
P3	30	7	23
P4	29	9	20
P5	4	2	2
P6	15	3	12

Step 5: Average Waiting Time

$$\text{Average WT} = \frac{22 + 17 + 23 + 20 + 2 + 12}{6}$$

$$= \frac{96}{6} = \boxed{16}$$

Step 6: Average Turnaround Time

$$\text{Average TAT} = \frac{27 + 23 + 30 + 29 + 4 + 15}{6}$$

$$= \frac{128}{6} = \boxed{21.33}$$

5.a. Discuss in detail the critical section problem and write the algorithm for the producer consumer problem.

The Critical-Section Problem

Consider a system consisting of n processes {P0, P1, ..., Pn−1}. Each process has a segment of code, called a critical section, in which the process may be changing common variables, updating a table, writing a file, and so on. When one process is executing in its critical section, no other process is allowed to execute in its critical section. The critical-section problem is to design a protocol that the processes can use to cooperate. Each process must request permission to enter its critical section. The section of code implementing this request is the entry section. The critical section may be followed by an exit section. The remaining code is the remainder section.

Code for producer is given below:

```
do {
    . . .
   /* produce an item in next_produced */
    . . .
   wait(empty);
   wait(mutex);
    . . .
   /* add next_produced to the buffer */
    . . .
   signal(mutex);
   signal(full);
} while (true);
```

Code for consumer is given below:

```
do {
   wait(full);
   wait(mutex);
      . . .
   /* remove an item from buffer to next_consumed */
      . . .
   signal(mutex);
   signal(empty);
      . . .
   /* consume the item in next_consumed */
      . . .
} while (true);
```

Consider the following system using data structures in the Bankers Algorithm with resource type ABC Maximum instance present in the system A = 10, B = 5, C = 7.

Process	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	7	5	3	3	3	2
P_1	2	0	0	3	2	2			
P_2	3	0	2	9	0	2			
P_3	2	1	1	2	2	2			
P_4	0	0	2	4	3	3			

i. Calculate Need Matrix
ii. Check whether system is safe or not.

🔹 **Need Matrix**

Process	A	B	C
P0	7	4	3
P1	1	2	2
P2	6	0	0
P3	0	1	1
P4	4	3	1

Step 2: Safety Algorithm

Initial **Available = (3,3,2)**

🔹 **Check P1**

Need (1,2,2) ≤ (3,3,2) ✅

After execution:
Available = (3,3,2) + (2,0,0)
= **(5,3,2)**

Safe sequence: **P1**

🔹 **Check P3**

Need (0,1,1) ≤ (5,3,2) ✅

Available = (5,3,2) + (2,1,1)
= **(7,4,3)**

Safe sequence: **P1 → P3**

🔹 **Check P4**

Need (4,3,1) ≤ (7,4,3) ✅

Available = (7,4,3) + (0,0,2)
= **(7,4,5)**

Safe sequence: **P1 → P3 → P4**

🔹 **Check P0**

Need (7,4,3) ≤ (7,4,5) ✅

Available = (7,4,5) + (0,1,0)
= **(7,5,5)**

Safe sequence: **P1 → P3 → P4 → P0**

🔹 **Check P2**

Need (6,0,0) ≤ (7,5,5) ✅

Available = (7,5,5) + (3,0,2)
= **(10,5,7)**

Safe sequence:

✅ **P1 → P3 → P4 → P0 → P2**

6.a. Outline the solutions of the Dining Philosopher problem.

One simple solution is to represent each chopstick with a semaphore. A philosopher tries to grab a chopstick by executing a wait() operation on that semaphore. She releases her chopsticks by executing the signal() operation on the appropriate semaphores.

Thus, the shared data are

semaphore chopstick[5];

where all the elements of chopstick are initialized to 1. The structure of philosopher i is

shown in Figure

```
do {
   wait(chopstick[i]);
   wait(chopstick[(i+1) % 5]);
      . . .
   /* eat for awhile */
      . . .
   signal(chopstick[i]);
   signal(chopstick[(i+1) % 5]);
      . . .
   /* think for awhile */
      . . .
} while (true);
```

6.b. Describe a Resource Allocation Graph with an example.

Deadlocks can be described in terms of a directed graph called System Resource-Allocation Graph. The graph consists of a set of vertices V and a set of edges E. The set of vertices V is

partitioned into two different types of nodes:

P = {P1, P2, ...,Pn}, the set consisting of all the active processes in the system.

R = {R1, R2, ..., Rm} the set consisting of all resource types in the system.

A directed edge from process Pi to resource type Rj is denoted by Pi → Rj it signifies that

process Pi has requested an instance of resource type Rj and is currently waiting for that resource.

A directed edge from resource type Rj to process Pi is denoted by Rj → Pi it signifies that an instance of resource type Rj has been allocated to process Pi.

A directed edge Pi → Rj is called a Request Edge.

A directed edge Rj → Pi is called an Assignment Edge.

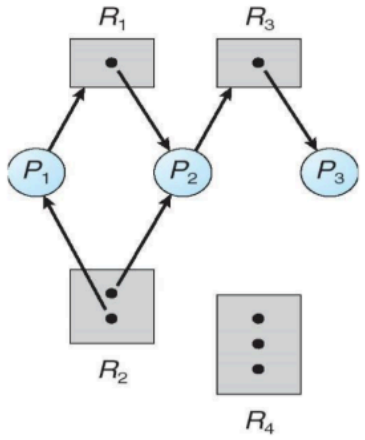
Pictorially each process Pi as a circle and each resource type Rj as a rectangle. Since resource type Rj may have more than one instance, each instance is represented as a dot within the rectangle.

A request edge points to only the rectangle Rj, whereas an assignment edge must also designate one of the dots in the rectangle.

When process Pi requests an instance of resource type Rj, a request edge is inserted in the

resource-allocation graph. When this request can be fulfilled, the request edge is instantaneously transformed to an assignment edge. When the process no longer needs access to the resource, it releases the resource; as a result, the assignment edge is deleted.

The resource-allocation graph shown in Figure depicts the following situation.



The sets P, K and E:

- $P = \{P_1, P_2, P_3\}$
- $R = \{R_1, R_2, R_3, R_4\}$
- $E = \{P_1 \rightarrow R_1, P_2 \rightarrow R_3, R_1 \rightarrow P_2, R_2 \rightarrow P_2, R_2 \rightarrow P_1, R_3 \rightarrow P_3\}$

Resource instances:

- One instance of resource type R_1
- Two instances of resource type R_2
- One instance of resource type R_3
- Three instances of resource type R_4

Process states:

- Process P_1 is holding an instance of resource type R_2 and is waiting for an instance of resource type R_1.
- Process P_2 is holding an instance of R_1 and an instance of R_2 and is waiting for an instance of R_3.
- Process P_3 is holding an instance of R_3.

Using Bankers algorithm, solve the following problem :

Process	Allocation				Max				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P_0	0	0	1	2	0	0	1	2	1	5	2	0
P_1	1	0	0	0	1	7	5	0				
P_2	1	3	5	4	2	3	5	6				
P_3	0	6	3	2	0	6	5	2				
P_4	0	0	1	4	0	6	5	6				

i. Calculate the Need Matrix
ii. Check whether system is safe or not.

✅ Step 1: Need Matrix

$$Need = Max - Allocation$$

Process	A	B	C	D
P0	0	0	0	0
P1	0	7	5	0
P2	1	0	0	2
P3	0	0	2	0
P4	0	6	4	2

Step 2: Safety Check

Initial **Available = (1,5,2,0)**

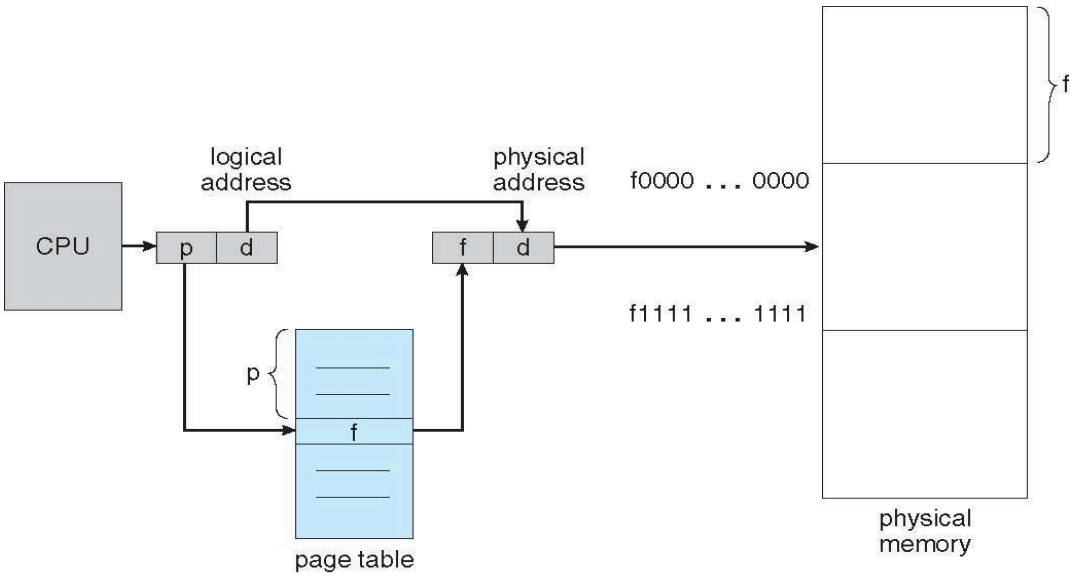
- **P0**

Need (0,0,0,0) ≤ (1,5,2,0) ✅

Available = (1,5,2,0) + (0,0,1,2)
= **(1,5,3,2)**

	Safe sequence: **P0** --- 🔹 **P2** Need (1,0,0,2) ≤ (1,5,3,2) ✅ Available = (1,5,3,2) + (1,3,5,4) = **(2,8,8,6)** Safe sequence: **P0 → P2** --- 🔹 **P3** Need (0,0,2,0) ≤ (2,8,8,6) ✅ Available = (2,8,8,6) + (0,6,3,2) = **(2,14,11,8)** Safe sequence: **P0 → P2 → P3** --- 🔹 **P4** Need (0,6,4,2) ≤ (2,14,11,8) ✅ Available = (2,14,11,8) + (0,0,1,4) = **(2,14,12,12)** Safe sequence: **P0 → P2 → P3 → P4** --- 🔹 **P1** Need (0,7,5,0) ≤ (2,14,12,12) ✅ Available = (2,14,12,12) + (1,0,0,0) = **(3,14,12,12)** Safe sequence: P0→P2→P3→P4→P1

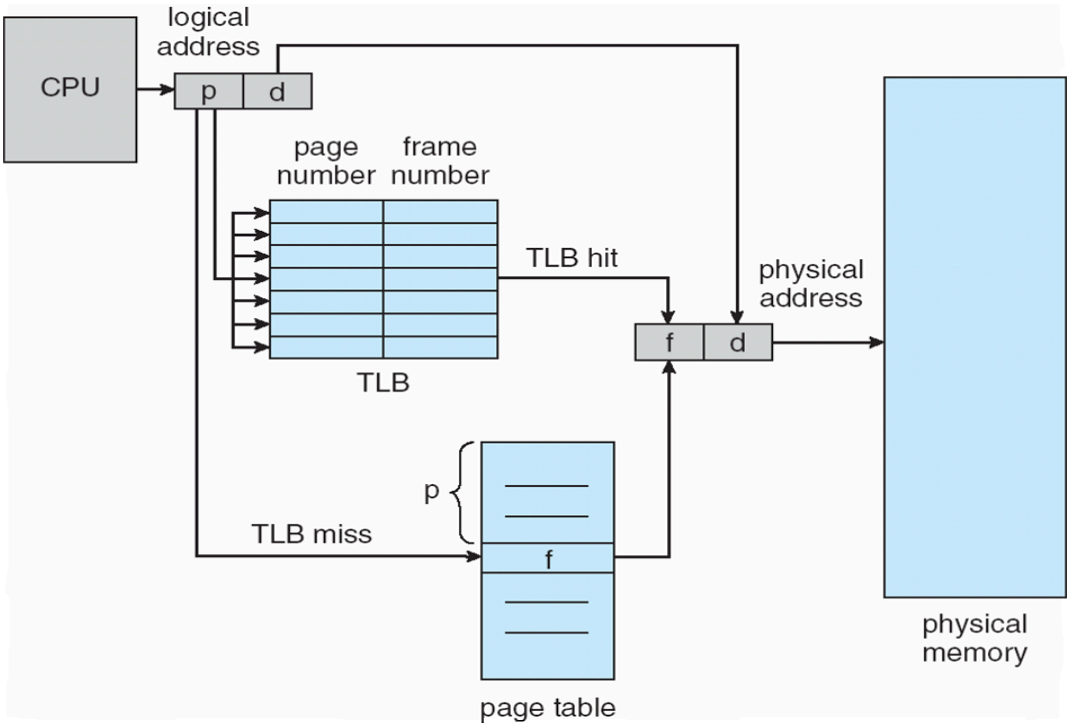
7	a.	Discuss the given memory management technique with diagram. i. Paging ii. Translation Look-Aside Buffer.	6 6	L2	CO4
	b.	Discuss about Contiguous Memory Allocation with a neat diagram.	8	L2	CO4



- Physical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available
- Divide physical memory into fixed-sized blocks called **frames**
  - Size is power of 2, between 512 bytes and 16 Mbytes
- Divide logical memory into blocks of same size called **pages**

Keep track of all free frames

- To run a program of size *N* pages, need to find *N* free frames and load program
- Set up a **page table** to translate logical to physical addresses
- Backing store likewise split into pages

- Still have Internal fragmentation
- 

Modern operating systems use **virtual memory**.

When a program accesses memory:

1. CPU generates a **virtual address**
2. That must be converted into a **physical address**
3. This mapping is stored in a **page table**

But accessing the page table every time is slow.

So we use **TLB** to store recent address translations.

With TLB:

- CPU → Check TLB first
- If found → Immediate translation (Fast!)
- If not found → Go to Page Table → Update TLB

TLB Hit

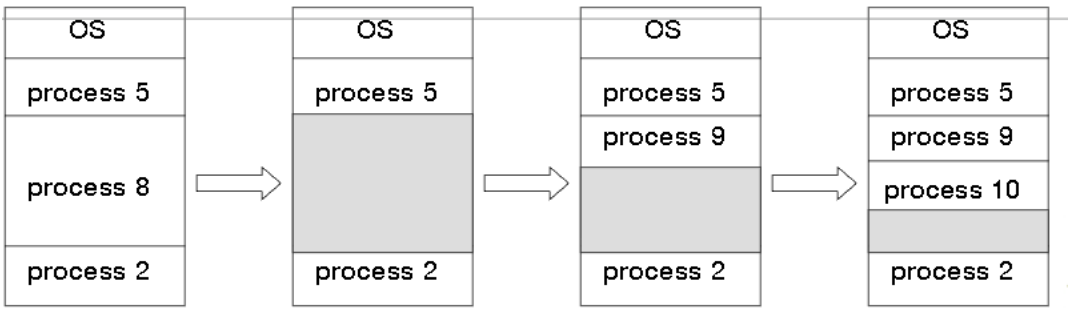
- When required mapping is found in TLB.
  - Fast access

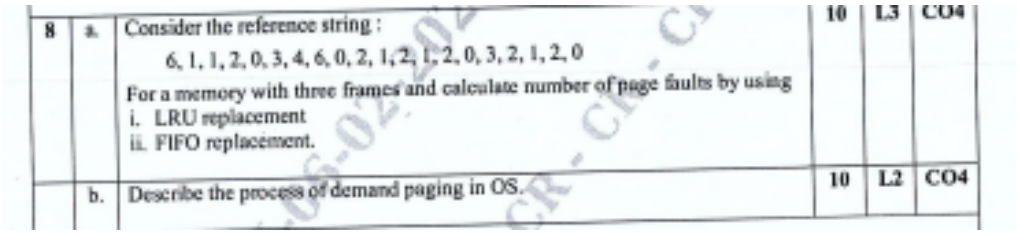
TLB Miss

- When mapping is not found.
  -Slower (needs page table lookup)

Contiguous Memory allocation:-

- Multiple-partition allocation
  - Degree of multiprogramming limited by number of partitions
  - Hole – block of available memory; holes of various size are scattered throughout memory
  - When a process arrives, it is allocated memory from a hole large enough to accommodate it
  - Process exiting frees its partition, adjacent free partitions combined
  - Operating system maintains information about:
    a) allocated partitions b) free partitions (hole)
- **First-fit**: Allocate the *first* partitions (hole) that is big enough
- **Best-fit**: Allocate the *smallest* partitions (hole) that is big enough; must search entire list, unless ordered by size
  - Produces the smallest leftover hole
- **Worst-fit**: Allocate the *largest* partitions (hole) ; must also search entire list
  - Produces the largest leftover hole



8	

8	a.	Consider the reference string : 6, 1, 1, 2, 0, 3, 4, 6, 0, 2, 1, 2, 1, 2, 0, 3, 2, 1, 2, 0 For a memory with three frames and calculate number of page faults by using i. LRU replacement ii. FIFO replacement.	10	L3	CO4
	b.	Describe the process of demand paging in OS.	10	L2	CO4

8.a

LRU (Least Recently Used)

We replace the page that was least recently used.

Step-by-step result (3 frames)

Ref	Frames (after replacement)	Fault?
6	6 - -	F
1	6 1 -	F
1	6 1 -	
2	6 1 2	F
0	0 1 2	F
3	0 3 2	F
4	0 3 4	F
6	6 3 4	F
0	6 3 0	F
2	2 3 0	F
1	2 1 0	F
2	2 1 0	

1	2 1 0	
2	2 1 0	
0	2 1 0	
3	2 3 0	F
2	2 3 0	
1	2 3 1	F
2	2 3 1	
0	2 0 1	F

Total LRU Page Faults = 13

FIFO (First In First Out)

Replace the oldest loaded page.

Step-by-step result

Ref	Frames (after replacement)	Fault?
6	6 - -	F
1	6 1 -	F
1	6 1 -	
2	6 1 2	F
0	0 1 2	F
3	0 3 2	F
4	0 3 4	F
6	6 3 4	F
0	6 0 4	F

2	6 0 2	F
1	1 0 2	F
2	1 0 2	
1	1 0 2	
2	1 0 2	
0	1 0 2	
3	1 3 2	F
2	1 3 2	
1	1 3 2	
2	1 3 2	
0	0 3 2	F

Total FIFO Page Faults = 12

Algorithm	**Page Faults**
LRU	**13**
FIFO	**12**

8.b The basic idea behind demand paging is that when a process is swapped in, its pages are not swapped in all at once. Rather they are swapped in only when the process needs them. (on demand.) This is termed as lazy swapper, although a pager is a more accurate term.
The basic idea behind demand paging is that when a process is swapped in, the pager only loads into memory those pages that is needed presently.
Pages that are not loaded into memory are marked as invalid in the page table, using the invalid bit. Pages loaded in memory are marked as valid.
If the process only ever accesses pages that are loaded in memory (memory resident pages), then the process runs exactly as if all the pages were loaded in to memory.

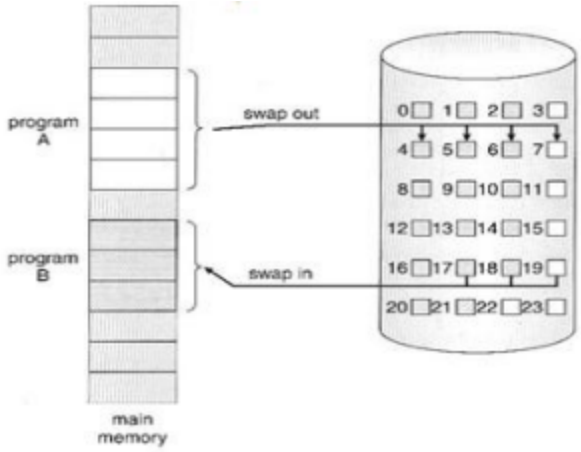


Figure 9.4 Transfer of a paged memory to contiguous disk space.

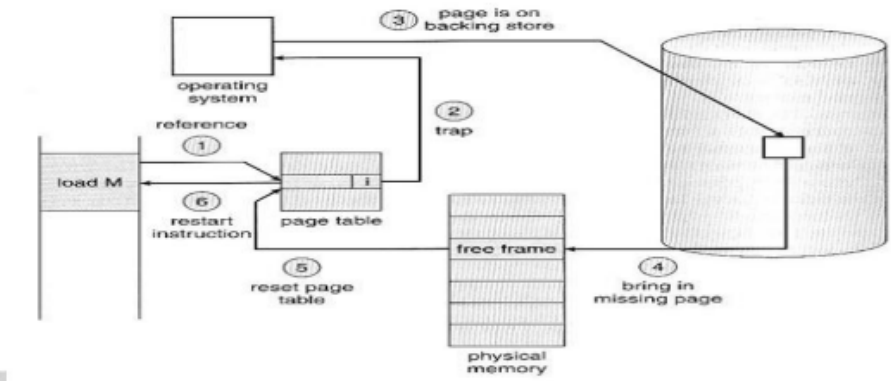
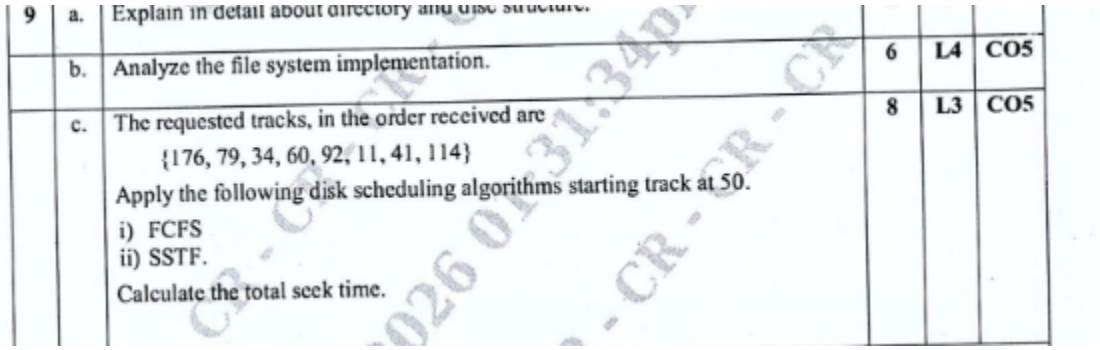


Figure 9.6 Steps in handling a page fault.

On the other hand, if a page is needed that was not originally loaded up, then a page fault trap is generated, which must be handled in a series of steps:
1. The memory address requested is first checked, to make sure it was a valid memory request.
2. If the reference is to an invalid page, the process is terminated. Otherwise, if the page is not present in memory, it must be paged in.
3. A free frame is located, possibly from a free-frame list.
4. A disk operation is scheduled to bring in the necessary page from disk.
5. After the page is loaded to memory, the process's page table is updated with the new frame number, and the invalid bit is changed to indicate that this is now a valid page reference.
6. The instruction that caused the page fault must now be restarted from the beginning.
In an extreme case, the program starts execution with zero pages in memory. Here NO pages are swapped in for a process until they are requested by page faults. This is known as pure demand paging.
The hardware necessary to support demand paging is the same as for paging

	and swapping: A page table and secondary memory.
9	

9	a.	Explain in detail about directory and disc structure.			
	b.	Analyze the file system implementation.	6	L4	CO5
	c.	The requested tracks, in the order received are {176, 79, 34, 60, 92, 11, 41, 114} Apply the following disk scheduling algorithms starting track at 50. i) FCFS ii) SSTF. Calculate the total seek time.	8	L3	CO5

DISK STRUCTURE

Each disk platter is divided into number of tracks and each track is divided into number
of sectors. Sectors is the basic unit for read or write operation in the disk.

Modern disk drives are addressed as a large one-dimensional array. The one-dimensional
array of logical blocks is mapped onto the sectors of the disk sequentially. Sector 0 is the first
sector of the first track on the outermost cylinder. The mapping proceeds in order through that
track, then through the rest of the tracks in that cylinder, and then through the rest of the
cylinders from outermost to innermost.

The disk structure (architecture) can be of two types –

i) Constant Linear Velocity (CLV)

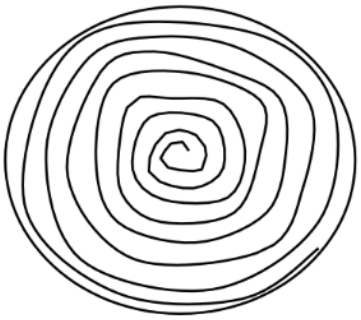
ii) Constant Angular Velocity (CAV)

i) CLV - The density of bits per track is uniform. The farther a track is from the center of
the disk, the greater its length, so the more sectors it can hold. As we move from outer
zones to inner zones, the number of sectors per track decreases. This architecture is
used in CD-ROM and DVD-ROM.

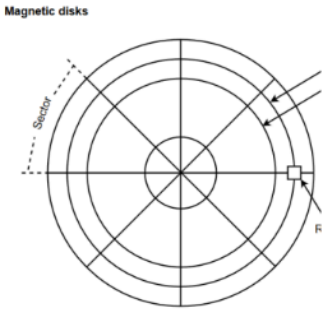
ii) CAV – There is same number of sectors in each track. The sectors are densely packed in

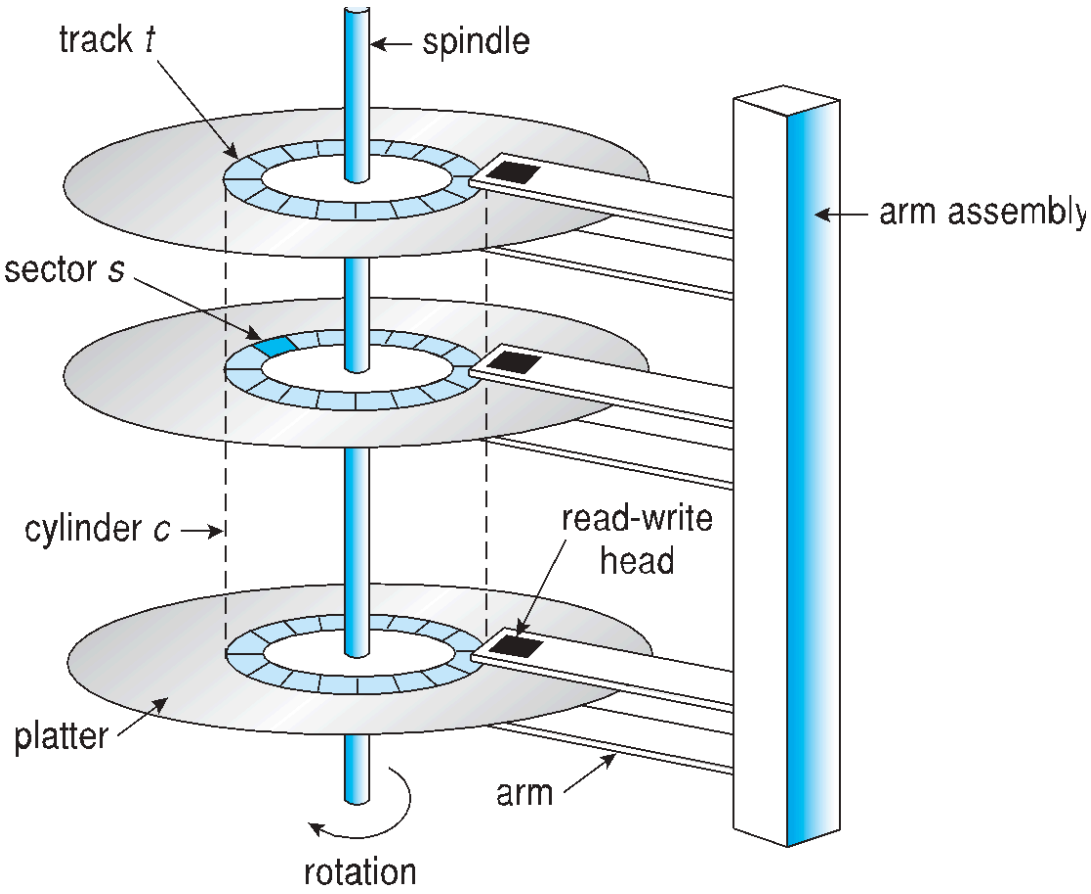
the inner tracks. The density of bits decreases from inner tracks to outer tracks to keep
the data rate constant.

CLV

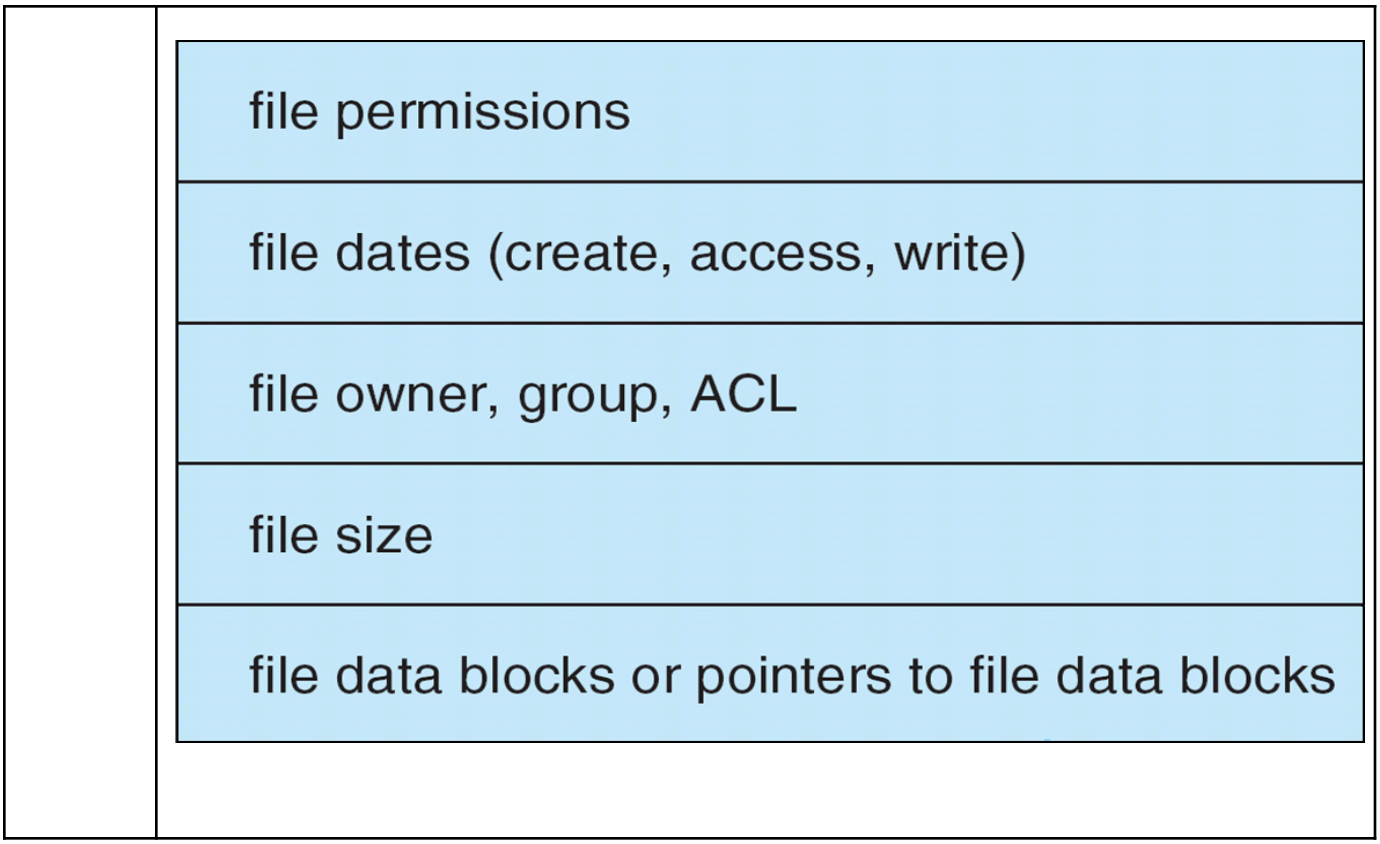


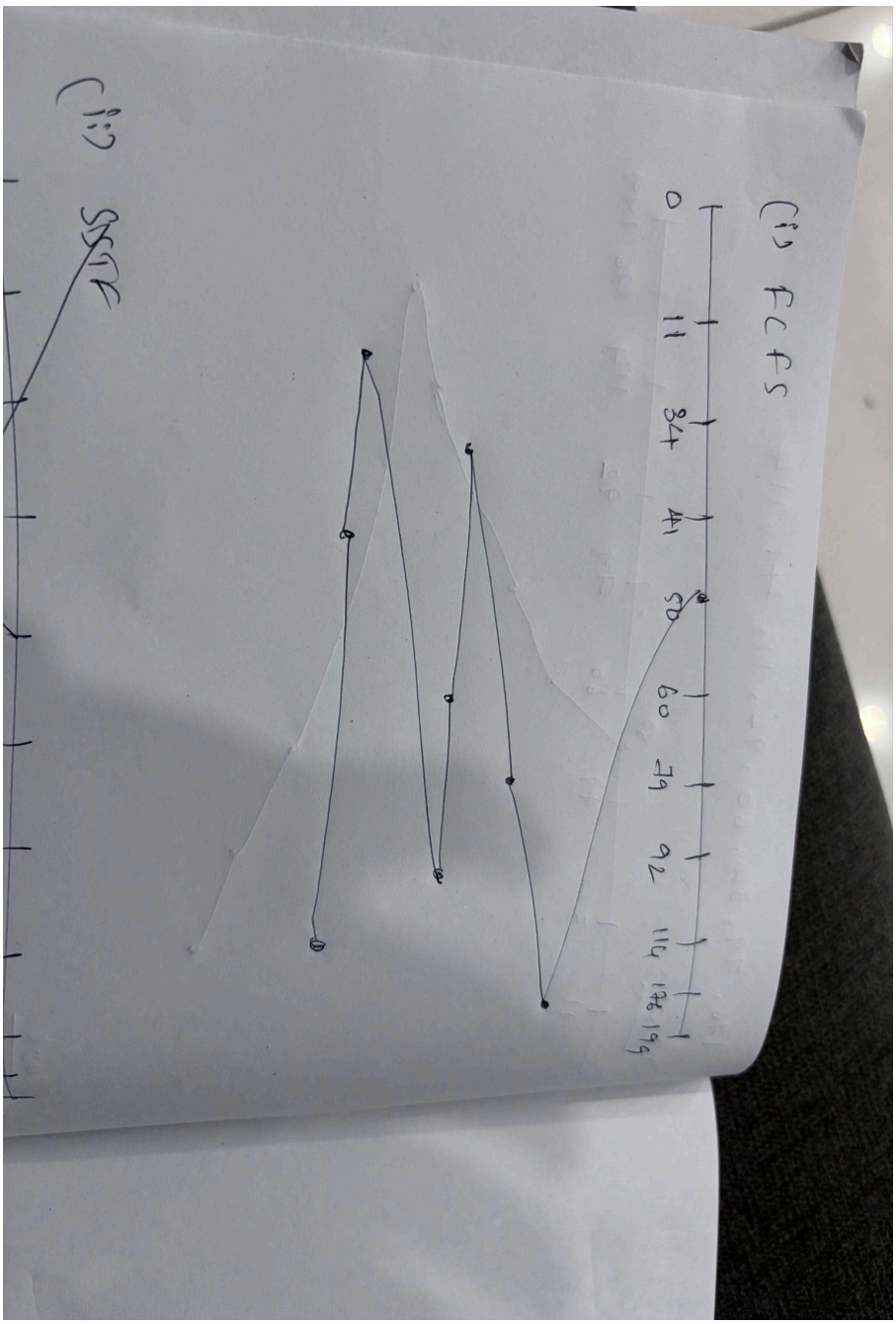
CAV





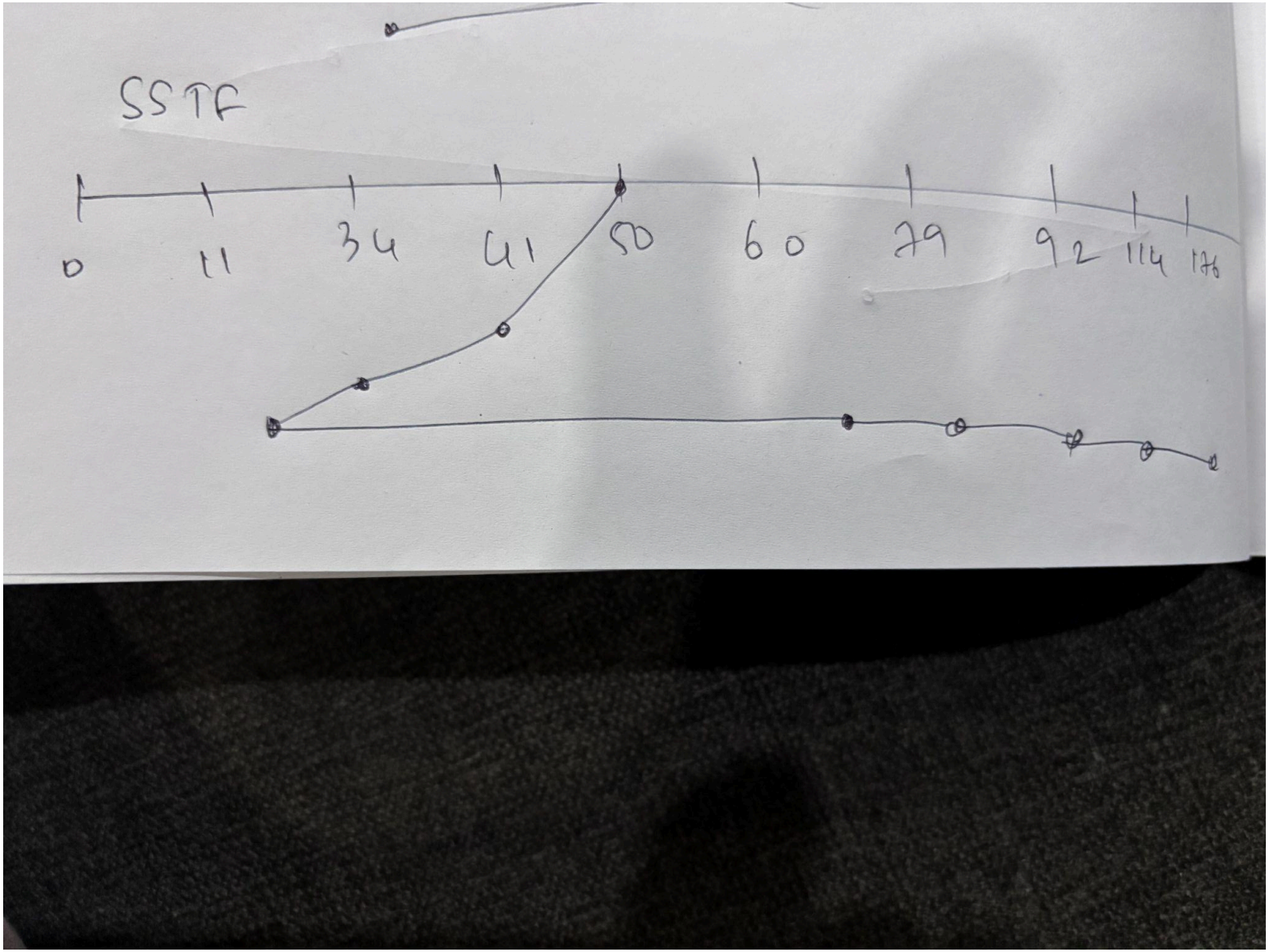
	▪ Bulk of secondary storage for modern computers is **hard disk drives** (**HDDs**) and **nonvolatile memory** (**NVM**) devices ▪ **HDDs** spin platters of magnetically-coated material under moving read-write heads • Drives rotate at 60 to 250 times per second • **Transfer rate** is rate at which data flow between drive and computer • **Positioning time** (**random-access time**) is time to move disk arm to desired cylinder (**seek time**) and time for desired sector to rotate under the disk head (**rotational latency**) • **Head crash** results from disk head making contact with the disk surface -- That's bad ▪ Disks can be removable • 9.b We have system calls at the API level, but how do we implement their functions? • On-disk and in-memory structures • Boot control block contains info needed by system to boot OS from that volume • Needed if volume contains OS, usually first block of volume • Volume control block (superblock, master file table) contains volume details • Total # of blocks, # of free blocks, block size, free block pointers or array • Directory structure organizes the files • Names and inode numbers, master file table • Per-file File Control Block (FCB) contains many details about the file • Inode number, permissions, size, dates • NFTS stores into in master file table using relational DB structures

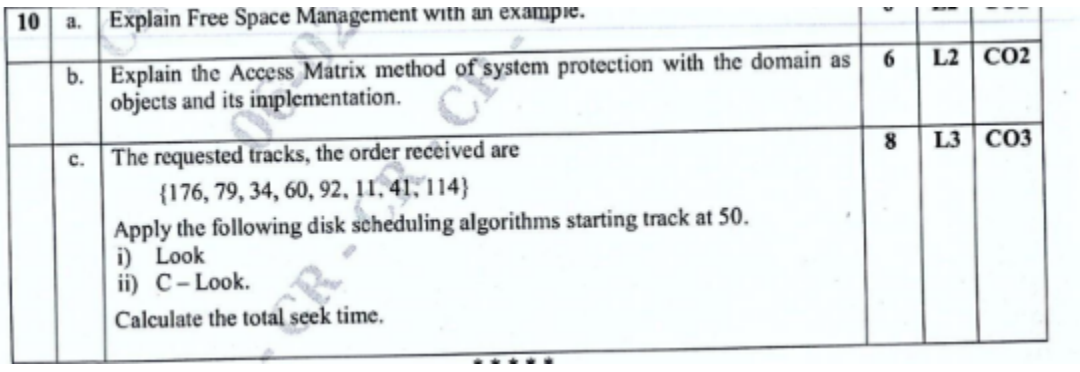

file permissions
file dates (create, access, write)
file owner, group, ACL
file size
file data blocks or pointers to file data blocks

9.c 

Move	Distance
50 → 176	126
176 → 79	97
79 → 34	45

	$34 \to 60$ 26 $60 \to 92$ 32 $92 \to 11$ 81 $11 \to 41$ 30 $41 \to 114$ 73 Total: $$126 + 97 + 45 + 26 + 32 + 81 + 30 + 73 = \boxed{510}$$

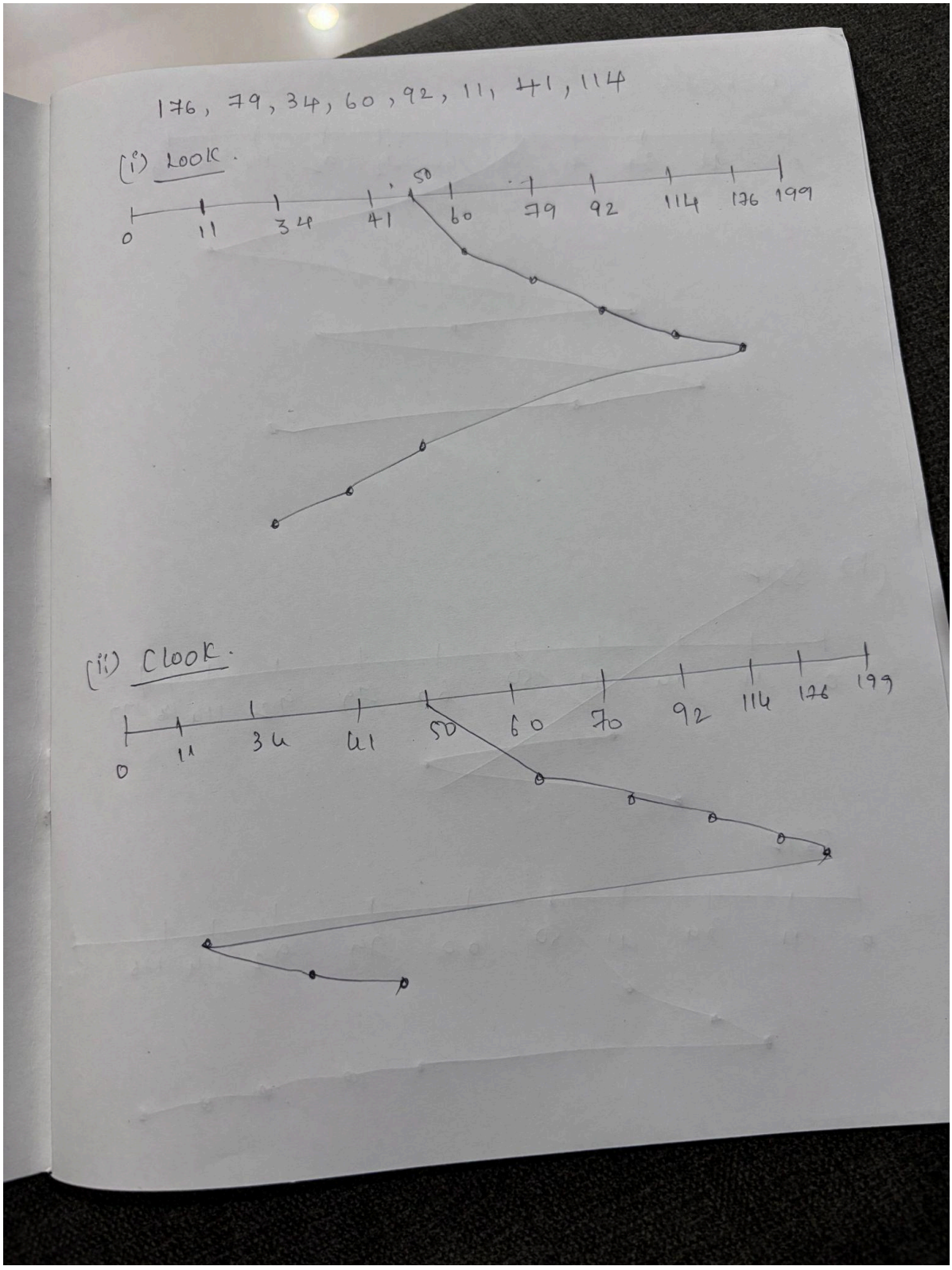

SSTF
0
11
34
41
50
60
79
92
114
176

	Move Distance 50 → 41 9 41 → 34 7 34 → 11 23 11 → 60 49 60 → 79 19 79 → 92 13 92 → 114 22 114 → 176 62 $9 + 7 + 23 + 49 + 19 + 13 + 22 + 62 = \boxed{204}$
10	 10.a The space created after deleting the files can be reused. Another important aspect of disk management is keeping track of free space in memory. The list which keeps

10	a.	Explain Free Space Management with an example.			
	b.	Explain the Access Matrix method of system protection with the domain as objects and its implementation.	**6**	**L2**	**CO2**
	c.	The requested tracks, the order received are {176, 79, 34, 60, 92, 11, 41, 114} Apply the following disk scheduling algorithms starting track at 50. i) Look ii) C – Look. Calculate the total seek time.	**8**	**L3**	**CO3**

	track of free space in memory is called the free-space list. To create a file, search the free-space list for the required amount of space and allocate that space to the new file. This space is then removed from the free-space list. When a file is deleted, its disk space is added to the free-space list. The free-space list, is implemented in different ways as explained below. a) Bit Vector Fast algorithms exist for quickly finding contiguous blocks of a given size One simple approach is to use a bit vector, in which each bit represents a disk block, set to 1 if free or 0 if allocated. For example, consider a disk where blocks 2,3,4,5,8,9, 10,11, 12, 13, 17and 18 are free, and the rest of the blocks are allocated. The free-space bit map would be 00111100111111100011 Easy to implement and also very efficient in finding the first free block or ‘n’ consecutive free blocks on the disk. The down side is that a 40GB disk requires over 5MB just to store the bitmap. b) Linked List A linked list can also be used to keep track of all free blocks. Traversing the list and/or finding a contiguous block of a given size are not easy, but fortunately are not frequently needed operations. Generally the system just adds and removes single blocks from the beginning of the list. The FAT table keeps track of the free list as just one more linked list on the table.

10.C


176, 79, 34, 60, 92, 11, 41, 114
(i) LOOK.
0
11
34
41
50
60
79
92
114
176
199
(ii) CLOOK.
0
11
34
41
50
60
70
92
114
176
199

Look;-

Total seek time:

$$10 + 19 + 13 + 22 + 62 + 135 + 7 + 23 = \boxed{291}$$

C-Look

Total seek time:

$$10 + 19 + 13 + 22 + 62 + 165 + 23 + 7 = \boxed{321}$$

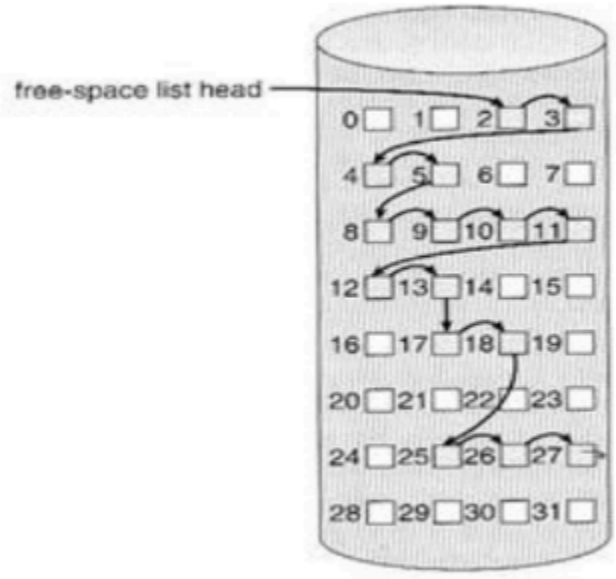


Figure 11.10 Linked free-space list on disk.

c) Grouping

A variation on linked list free lists. It stores the addresses of n free blocks in the
first free block. The first n-1 blocks are actually free. The last block contains the
addresses of another n free blocks, and so on.
The address of a large number of free blocks can be found quickly.

d) Counting

When there are multiple contiguous blocks of free space then the system can keep
track of the starting address of the group and the number of contiguous free blocks.
Rather than keeping al list of n free disk addresses, we can keep the address of first free block and the number of free contiguous blocks that follow the first block.
Thus the overall space is shortened. It is similar to the extent method of allocating
blocks.

10.b

ACCESS MATRIX

Our model of protection can be viewed as a matrix, called an access matrix. It is a
general model of protection that provides a mechanism for protection without imposing a
particular protection policy. The rows of the access matrix represent domains, and the columns
represent objects. Each entry in the matrix consists of a set of access rights. The entry access(i,j)
defines the set of operations that a process executing in domain Di can invoke on object Oj.

domain \ object	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	

Implementation of Access Matrix

Different methods of implementing the access matrix (which is sparse).

- Global Table
- Access Lists for Objects
- Capability Lists for Domains
- Lock-Key Mechanism

Global Table

	This is the simplest implementation of access matrix. A set of ordered triples <domain, object, rights-set>is maintained in a file. Whenever an operation M is executed on an object Oj, within domain Di, the table is searched for a triple <Di, Oj, Rk>. If this triple is found, the operation is allowed to continue; otherwise, an exception (or error) condition is raised. Drawbacks - The table is usually large and thus cannot be kept in main memory. Additional I/O is needed Access Lists for Objects Each column in the access matrix can be implemented as an access list for one object. The empty entries are discarded. The resulting list for each object consists of ordered pairs <domain, rights-set>. It defines all domains access right for that object. When an operation M is executed on object Oj in Di, search the access list for object Oj, look for an entry <Di, Rj> withM eKj. If the entry is found, we allow the operation; if it is not, we check the default set. If M is in the default set, we allow the access. Otherwise, access is denied, and an exception condition occurs. For efficiency, we may check the default set first and then search the access list. Capability Lists for Domains A capability list for a domain is a list of objects together with the operations allowed on those objects. An object is often represented by its name or address, called a capability. To execute operation M on object Oj, the process executes the operation M, specifying the capability for object Oj as a parameter. Simple possession of the capability means that access is allowed.