Code No: 183BV R22

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD B. Tech II Year I Semester Examinations, June/July-2025
OPERATING SYSTEMS (Common to CSE(AI&ML), CSE(IOT), AI&ML) (Common

Time: 3 Hours
Note: This question paper contains two parts A and B.
i) Part- A for 10 marks, ii) Part - B for 50 marks.
Part-A is a compulsory question which consists of ten sub-questions from all units carrying equal marks.
Part-B consists of ten questions (numbered from 2 to 11) carrying 10 marks each. From each unit, there are two questions and the student should answer one of them. Hence, the student should answer five questions from Part-B.

PART-A

(10 Marks)

1.a) What is thread?
b) Discuss about time shared operating system
What is Preemptive. Scheduling?
Define Turnaround time.
What is Named Pipe?
Define the usage of counting Semaphore.
What is translation look aside buffer?
What is fragmentation?
Discuss about ioctl system call.
j) Discuss about exclusive lock.

PART-B (50 Marks)

2.a) Discuss about process and its states.

2.a) Discuss about Process and its States. (5 Marks)

A **process** is a program in execution. It is an active entity that consists of program code, data, program counter, CPU registers, stack, and allocated resources. The operating system manages processes to execute multiple tasks efficiently.

Process States

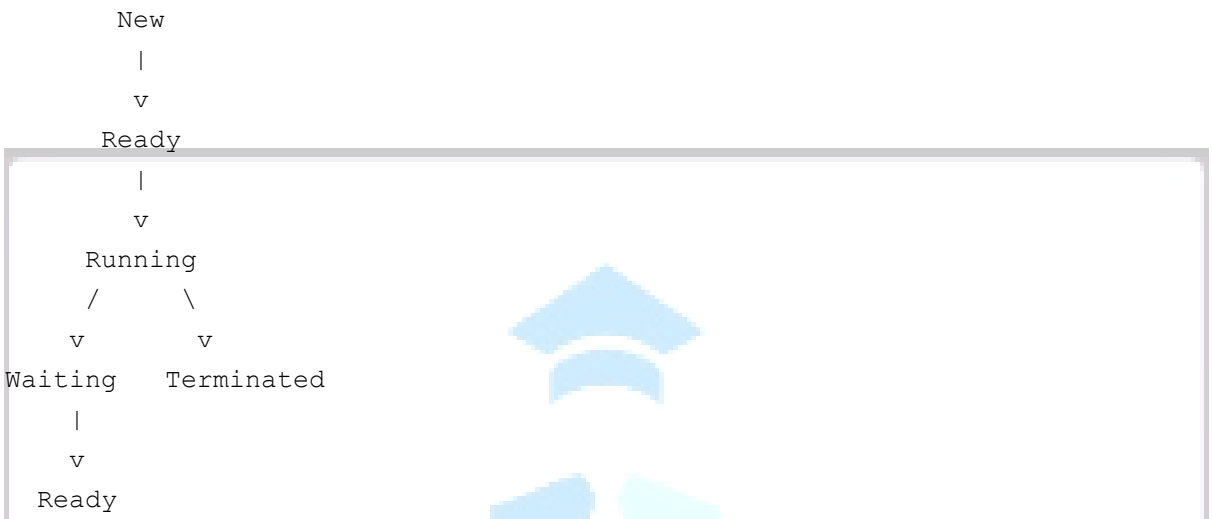
A process passes through different states during its lifetime:

1. **New State**
   - The process is being created.
   - Required resources are allocated.
2. **Ready State**
   - The process is ready to execute and waiting for CPU allocation.
3. **Running State**

- The CPU is currently executing the process instructions.

4. **Waiting (Blocked) State**
    - The process waits for an event such as I/O completion or resource availability.
5. **Terminated State**
    - The process has completed execution and is removed from the system.

Process State Diagram



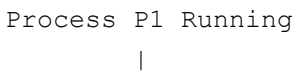
b) Explain the switching of CPU from process to process. [5+5]

The switching of the CPU from one process to another is called **Context Switching**. It allows the operating system to share the CPU among multiple processes.

Steps in Context Switching

1. The currently running process is interrupted due to:
    - Time slice expiration
    - I/O request
    - Arrival of a higher-priority process
2. The operating system saves the current process status, including:
    - Program Counter (PC)
    - CPU Registers
    - Process State
    - Memory Management Information
3. This information is stored in the **Process Control Block (PCB)** of the process.
4. The scheduler selects another ready process from the ready queue.
5. The saved context of the selected process is loaded from its PCB.
6. The CPU starts executing the new process.

Context Switching Diagram



```
   Save Context
        |
        v
   Scheduler Selects
        |
        v
   Load Context of P2
        |
        v
Process P2 Running
```

Advantages

- Enables multiprogramming.
- Improves CPU utilization.
- Supports multitasking and time-sharing systems.

OR

3.a) Discuss about how the system calls are used.

A **system call** is a mechanism through which a user program requests services from the operating system kernel. It acts as an interface between the user process and the operating system.

How System Calls are Used

1. A user program needs a service such as reading a file, writing data, creating a process, or communicating with hardware.
2. The program invokes a system call.
3. Control is transferred from **user mode** to **kernel mode**.
4. The operating system performs the requested service.
5. After completing the service, control returns to the user program.

Working of System Call

```
User Program
      |
      v
 System Call
      |
      v
Operating System Kernel
      |
      v
```

```
Requested Service
      |
      v
Return Result to User Program
```

Types of System Calls

1. **Process Control**
   - create process(), exit(), wait()
2. **File Management**
   - open(), read(), write(), close()
3. **Device Management**
   - ioctl(), read(), write()
4. **Information Maintenance**
   - getpid(), gettimeofday()
5. **Communication**
   - pipe(), socket(), send(), receive()

Advantages

- Provides secure access to hardware resources.
- Enables communication between user programs and the operating system.
- Simplifies resource management.

b) Explain about different types of system calls. [5+5]

System calls provide an interface between user programs and the operating system. They allow processes to request services from the OS kernel.

1. Process Control System Calls

These system calls are used to create, terminate, and manage processes.

Examples:

- `fork()`
- `exit()`
- `wait()`
- `exec()`

Functions:

- Create new processes
- Terminate processes
- Load and execute programs

2. File Management System Calls

These calls handle file operations such as creating, reading, writing, and deleting files.

Examples:

- `open()`
- `read()`
- `write()`
- `close()`

Functions:

- Access files
- Modify file contents
- Manage file attributes

3. Device Management System Calls

These system calls control and manage I/O devices.

Examples:

- `read()`
- `write()`
- `ioctl()`

Functions:

- Request device access
- Read from and write to devices
- Configure device parameters

4. Information Maintenance System Calls

These calls obtain or modify system and process information.

Examples:

- `getpid()`
- `alarm()`
- `time()`

Functions:

- Get process information
- Obtain date and time
- Set system attributes

---

5. Communication System Calls

These system calls enable communication between processes.

Examples:

- `pipe()`
- `socket()`
- `send()`
- `receive()`

Functions:

- Exchange data between processes
- Support networking and IPC

4.a) Consider the following Processes, with the length of CPU burst given in milli seconds

Process	Burst Time
P1	24
P2	3
P3	3

Assume all processes arrived at the same time. Calculate the average waiting time given example using round robin scheduling algorithm.

For **Round Robin Scheduling**, the standard textbook example usually assumes a **time quantum = 4 ms**.

Given

Process	Burst Time (ms)
P1	24
P2	3
P3	3

Time Quantum = 4 ms

Gantt Chart

```
0     4     7    10    14    18    22    26    30
| P1 | P2 | P3 | P1 | P1 | P1 | P1 | P1 |
```

Completion Times

- P2 completes at **7 ms**
- P3 completes at **10 ms**
- P1 completes at **30 ms**

Waiting Time Calculation

Waiting Time = Completion Time − Burst Time − Arrival Time

(All processes arrive at time 0)

Process	Completion Time	Burst Time	Waiting Time
P1	30	24	$30 - 24 = 6$
P2	7	3	$7 - 3 = 4$
P3	10	3	$10 - 3 = 7$

Average Waiting Time

Average Waiting Time=6+4+73=173=5.67 ms\text{Average Waiting Time} = \frac{6 + 4 + 7}{3} = \frac{17}{3} = 5.67 \text{ ms}Average Waiting Time=36+4+7=317=5.67 ms

Answer

Average Waiting Time=5.67 ms\boxed{\text{Average Waiting Time} = 5.67 \text{ ms}}Average Waiting Time=5.67 ms

Result: Using Round Robin Scheduling with a time quantum of **4 ms**, the **average waiting time is 5.67 ms**.

b) Elaborate how time quantum affects the context switches in round robin sched algorithm.

Round Robin (RR) is a preemptive CPU scheduling algorithm in which each process is assigned a fixed time interval called the **time quantum**. After the time quantum expires, the CPU switches to the next process in the ready queue.

Effect of Time Quantum on Context Switches

1. Small Time Quantum

- CPU switches frequently between processes.

- Number of context switches increases.
- Provides better response time for interactive users.
- However, excessive context switching increases overhead and reduces CPU efficiency.

Example: If time quantum = 2 ms, processes are switched very often.

2. Large Time Quantum

- Processes run for longer periods before being switched.
- Number of context switches decreases.
- Context-switch overhead is reduced.
- If the quantum is too large, Round Robin behaves like **FCFS (First Come First Served)** scheduling.

Example: If time quantum = 50 ms, most processes may finish before the quantum expires.

Relationship

```
Small Time Quantum
        ↓
More Context Switches
        ↓
Higher Overhead

Large Time Quantum
        ↓
Fewer Context Switches
        ↓
Lower Overhead
```

Advantages of Choosing an Appropriate Quantum

- Balances response time and CPU utilization.
- Reduces unnecessary context switching.
- Improves overall system performance

OR

5.a) What is deadlock? Describe the necessary conditions for a deadlock to occur.

Definition of Deadlock

A **deadlock** is a situation in which two or more processes are permanently blocked because each process is waiting for a resource that is held by another process.

Example: Process P1 holds Resource R1 and waits for R2, while Process P2 holds R2 and waits for R1. Neither process can proceed, resulting in a deadlock.

Necessary Conditions for Deadlock

According to Coffman's conditions, the following four conditions must occur simultaneously for a deadlock to exist:

1. Mutual Exclusion

- At least one resource must be non-shareable.
- Only one process can use the resource at a time.

Example: A printer can be used by only one process at a time.

2. Hold and Wait

- A process holding one or more resources is waiting for additional resources held by other processes.

Example: A process holds a printer and waits for a scanner.

3. No Preemption

- Resources cannot be forcibly taken away from a process.
- The resource must be released voluntarily by the process holding it.

4. Circular Wait

- A circular chain of processes exists where each process is waiting for a resource held by the next process.

Example:

```
P1 → waits for P2
P2 → waits for P3
P3 → waits for P1
```

Diagram

```
P1 ----> R2
^         |
|         v
R1 <---- P2
```

b) Discuss how deadlocks Discuss how deadlocks can be prevented.

Deadlock prevention is a technique used to ensure that the system never enters a deadlock state. It is achieved by denying at least one of the four necessary conditions for deadlock.

Methods of Deadlock Prevention

1. Eliminate Mutual Exclusion

- Make resources shareable whenever possible.
- Multiple processes can use the resource simultaneously.

Example: Shared read-only files.

2. Eliminate Hold and Wait

- Require a process to request all required resources at once before execution.
- A process is not allowed to hold one resource while waiting for another.

3. Eliminate No Preemption

- If a process requests a resource that is unavailable, it must release all resources currently held.
- The resources are reassigned when available.

4. Eliminate Circular Wait

- Assign a numerical order to all resource types.
- Processes must request resources in increasing order only.

Example:

```
R1 → R2 → R3
```

A process holding R1 can request R2, but not vice versa.

Summary Table

Deadlock Condition	Prevention Method
Mutual Exclusion	Make resources sharable
Hold and Wait	Request all resources at once
No Preemption	Allow resource preemption
Circular Wait	Impose resource ordering

6.a) A solution to Critical Section Problem must satisfy three requirements: Discuss about it.

The **Critical Section Problem** occurs when multiple processes share common data and each process has a code segment called the **critical section**, where shared resources are accessed. A correct solution must satisfy the following three requirements:

1. Mutual Exclusion

- At most **one process** can be in its critical section at any given time.
- If one process is executing in the critical section, other processes must wait.

Example: If Process P1 is updating a shared file, Process P2 cannot access that file simultaneously.

2. Progress

- If no process is in the critical section and some processes wish to enter it, the selection of the next process must not be postponed indefinitely.
- Only processes that want to enter the critical section can participate in the decision.

Example: If P1 leaves the critical section and P2 wants to enter, P2 should be allowed without unnecessary delay.

3. Bounded Waiting

- There must be a limit on the number of times other processes can enter their critical sections after a process has requested entry.
- This prevents **starvation**.

Example: If P1 requests access, it should not wait forever while other processes repeatedly enter the critical section.

Summary Table

Requirement	Description
Mutual Exclusion	Only one process can execute in the critical section at a time.
Progress	A waiting process should not be postponed indefinitely.
Bounded Waiting	Every requesting process gets a fair chance to enter the critical section.

b) Discuss about Peterson's Solution to critical section problem. [5+5]

Peterson's Solution is a software-based method used to solve the **Critical Section Problem** for **two processes**. It ensures that only one process enters the critical section at a time and satisfies the three requirements: **Mutual Exclusion, Progress, and Bounded Waiting**.

Shared Variables

Peterson's solution uses two shared variables:

C
```
boolean flag[2];
int turn;
```

- `flag[i] = true` indicates that process `Pi` wants to enter the critical section.
- `turn` indicates whose turn it is to enter the critical section.

Algorithm for Process Pi

C
```
flag[i] = true;
turn = j;

while(flag[j] && turn == j)
; // wait

/* Critical Section */

flag[i] = false;

/* Remainder Section */
```

where `j = 1 - i`.

Working

1. A process sets its flag to **true**, indicating its desire to enter the critical section.
2. It gives priority to the other process by setting `turn`.
3. If the other process also wants to enter and it is the other's turn, the process waits.
4. After completing the critical section, it sets its flag to **false** and exits.

Example

Suppose:

- Process P0 and P1 both want to enter the critical section.
- Both set their flags to true.
- The value of `turn` decides which process enters first.
- The other process waits until the first process exits.

Advantages

- Simple and easy to understand.
- Provides **Mutual Exclusion**.
- Ensures **Progress**.
- Provides **Bounded Waiting**.

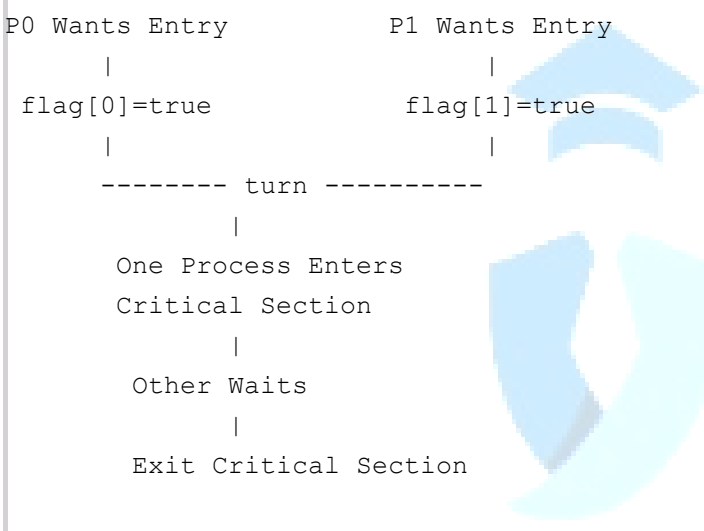
Limitations

- Works only for **two processes**.
- Uses busy waiting (CPU time is wasted while waiting).
- Not suitable for modern multiprocessor systems.

Conclusion

Peterson's Solution is a classical software solution for synchronizing two processes. It guarantees **Mutual Exclusion, Progress, and Bounded Waiting**, making it an important solution to the Critical Section Problem.

Diagram



OR

7.a) What is Producer Consumer Problem?

The **Producer-Consumer Problem** is a classic synchronization problem in operating systems. It involves two processes:

- **Producer**: Produces data/items and places them into a shared buffer.
- **Consumer**: Removes and consumes data/items from the shared buffer.

The challenge is to ensure that the producer does not add data when the buffer is full and the consumer does not remove data when the buffer is empty.

Working

1. The producer generates an item and places it in the buffer.
2. The consumer takes the item from the buffer and uses it.
3. Both processes share the same buffer, so synchronization is required to avoid data inconsistency.

Problems

- **Buffer Overflow**: Producer tries to insert data into a full buffer.
- **Buffer Underflow**: Consumer tries to remove data from an empty buffer.
- **Race Conditions**: Producer and consumer access the buffer simultaneously.

Solution Using Semaphores

Three semaphores are commonly used:

- **Mutex**: Ensures mutual exclusion.
- **Empty**: Counts empty buffer slots.
- **Full**: Counts filled buffer slots.

Diagram

```
Producer  ----->  Shared Buffer  ----->  Consumer
   |                                        |
Produces Data                          Consumes Data
```

Example

- A printer spooler system:
    - Users (producers) submit print jobs.
    - Printer (consumer) prints the jobs from the queue.

b) Explain the solution to producer consumer problem using semaphores. [5+5]

The **Producer-Consumer Problem** can be solved using **semaphores** to synchronize access to a shared buffer and prevent race conditions.

Semaphores Used

Assume the buffer size is **N**.

1. **Mutex = 1**
    - Provides mutual exclusion.
    - Only one process can access the buffer at a time.

2. **Empty = N**
    - Counts the number of empty slots in the buffer.
3. **Full = 0**
    - Counts the number of filled slots in the buffer.

Producer Process

C

```
do {
produce_item();

wait(empty);
wait(mutex);

add item to buffer;

signal(mutex);
signal(full);

} while(1);
```

Consumer Process

C

```
do {
wait(full);
wait(mutex);

remove item from buffer;

signal(mutex);
signal(empty);

consume_item();

} while(1);
```

Working

Producer

1. Produces an item.

2. Checks whether the buffer has an empty slot (`wait(empty)`).
3. Enters the critical section (`wait(mutex)`).
4. Inserts the item into the buffer.
5. Leaves the critical section (`signal(mutex)`).
6. Increases the count of filled slots (`signal(full)`).

Consumer

1. Checks whether the buffer contains an item (`wait(full)`).
2. Enters the critical section (`wait(mutex)`).
3. Removes an item from the buffer.
4. Leaves the critical section (`signal(mutex)`).
5. Increases the count of empty slots (`signal(empty)`).
6. Consumes the item.

Diagram

Plain Text

Producer

|

v

+------------------+

| Shared Buffer |

+------------------+

|

v

Consumer

Semaphores:

Mutex = 1

Empty = N

Full = 0

Advantages

- Prevents **buffer overflow**.
- Prevents **buffer underflow**.
- Ensures **mutual exclusion**.
- Avoids **race conditions**.

8.a) What is Paging? Discuss Paging technique in detail.

Definition of Paging

Paging is a memory management technique in which the logical address space of a process is divided into fixed-size blocks called **pages**, and the physical memory is divided into fixed-size blocks called **frames**. The operating system loads pages into available frames, eliminating the need for contiguous memory allocation.

Paging Technique

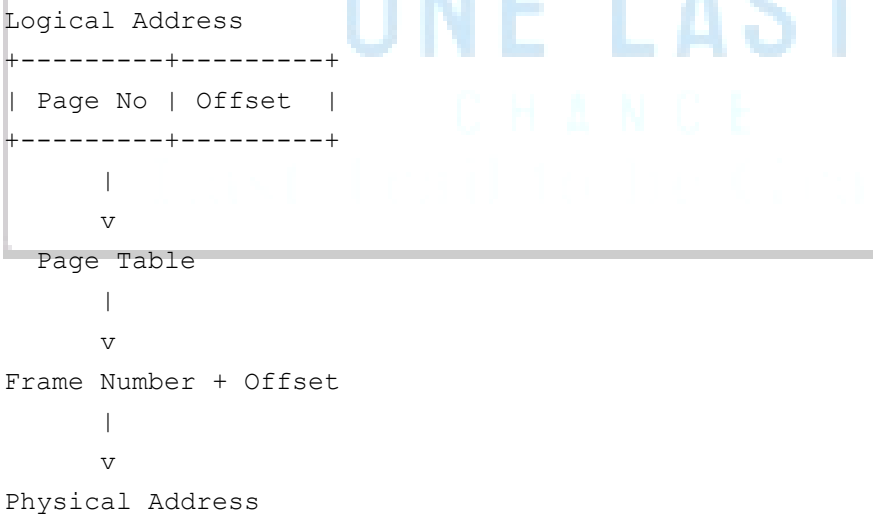
1. Division of Memory

- **Logical Memory** → Divided into **Pages**
- **Physical Memory** → Divided into **Frames**
- The size of a page and a frame is always the same.

2. Page Table

- Each process has a **Page Table**.
- The page table stores the mapping between page numbers and frame numbers.
- It helps in translating logical addresses into physical addresses.

Address Translation



Example

Suppose:

- Page Size = 1 KB
- Process Size = 4 KB

The process is divided into:

Page	Stored in Frame
P0	F2
P1	F5
P2	F1
P3	F7

Thus, a process can be stored in non-contiguous memory locations.

Advantages of Paging

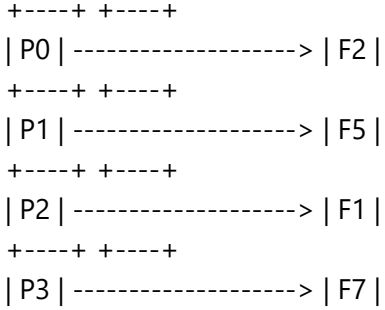
1. Eliminates **External Fragmentation**.
2. Allows efficient memory utilization.
3. Simplifies memory allocation.
4. Supports virtual memory implementation.

Disadvantages of Paging

1. Causes **Internal Fragmentation**.
2. Requires additional memory for page tables.
3. Address translation increases memory access time.

Diagram

Plain Text
Logical Memory Physical Memory



+----+ +----+

b) Discuss inverted page table in detail. [5+5]

An **Inverted Page Table (IPT)** is a memory management technique that maintains **one page table for the entire system** instead of having a separate page table for each process. Each entry in the inverted page table corresponds to a **physical frame** rather than a virtual page.

It is mainly used to reduce the memory required to store page tables in systems with large virtual address spaces.

Structure of Inverted Page Table

Each entry of the inverted page table contains:

- Process ID (PID)
- Virtual Page Number (VPN)
- Control bits (valid bit, protection bit, etc.)

Since there is one entry for each physical frame, the size of the table depends on the number of frames in physical memory.

Example

Frame Number	PID	Page Number
0	P1	2
1	P2	5
2	P1	1
3	P3	4

This table indicates which process page is currently stored in each physical frame.

Working of Inverted Page Table

1. The CPU generates a logical address consisting of:
    - Page Number
    - Offset
2. The operating system searches the inverted page table for a matching:
    - Process ID
    - Page Number
3. If a match is found:

    - The corresponding frame number is obtained.
    - The physical address is generated.
4. If no match is found:
    - A page fault occurs.

Address Translation

Plain Text
Logical Address
(Page Number + Offset)
|
v
Inverted Page Table
|
v
Frame Number
|
v
Physical Address

Advantages

1. Significantly reduces memory required for page tables.
2. Suitable for systems with very large virtual address spaces.
3. Only one page table is maintained for the entire system.

Disadvantages

1. Searching the table can be slower.
2. Additional techniques such as hashing are often required.
3. More complex than conventional page tables.

Diagram

Plain Text
Logical Address
+-------------+
| Page Offset |
+-------------+
|
v

Inverted Page Table
|
+--------+--------+
| PID | Page No. |
+--------+--------+
|
v
Frame Number
|
v
Physical Address

OR

9.a) Consider the following page reference string:

7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1

How many page faults occur for the following example using FIFO page replacement algorithm and assuming three frames?

Using **FIFO (First-In First-Out) Page Replacement** with **3 frames**:

Page Reference String

```
7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1
```

FIFO Calculation

Page	Frames	Fault/Hit
7	7 - -	Fault (1)
0	7 0 -	Fault (2)
1	7 0 1	Fault (3)
2	2 0 1	Fault (4)
0	2 0 1	Hit
3	2 3 1	Fault (5)
0	2 3 0	Fault (6)
4	4 3 0	Fault (7)
2	4 2 0	Fault (8)
3	4 2 3	Fault (9)
0	0 2 3	Fault (10)
3	0 2 3	Hit
2	0 2 3	Hit
1	0 1 3	Fault (11)
2	0 1 2	Fault (12)

0	0 1 2	Hit
1	0 1 2	Hit
7	7 1 2	Fault (13)
0	7 0 2	Fault (14)
1	7 0 1	Fault (15)

Total Page Faults

15\boxed{15}15

Answer

Number of page faults using FIFO page replacement algorithm with 3 frames = 15 page faults

b) What is virtual memory? How it is implemented? [5+5]

Definition of Virtual Memory

Virtual Memory is a memory management technique that allows the execution of processes that may not be completely loaded into the main memory (RAM). It creates an illusion of a large memory space by using a portion of secondary storage (hard disk) as an extension of RAM.

Thus, programs larger than physical memory can be executed efficiently.

Need for Virtual Memory

- Allows execution of large programs.
- Increases the degree of multiprogramming.
- Improves memory utilization.
- Enables efficient sharing of memory among processes.

Implementation of Virtual Memory

Virtual memory is mainly implemented using:

1. Demand Paging

- Pages are loaded into memory only when they are required.
- If a required page is not in memory, a **page fault** occurs.

- The operating system loads the page from disk into an available frame.

Steps in Demand Paging

1. CPU generates an address.
2. Check if the page is in memory.
3. If present, execution continues.
4. If absent, a page fault occurs.
5. OS loads the page from disk into RAM.
6. Page table is updated.
7. Execution resumes.

---

2. Demand Segmentation

- Memory is divided into logical segments such as code, data, and stack.
- Segments are loaded only when needed.
- Similar to demand paging but based on segments.

Virtual Memory Diagram

Plain Text

Logical Address Space

+--------------------+

| Program Pages |

+--------------------+

|

v

Page Table

|

+---------------+

| Main Memory |

+---------------+

|

Page Fault

|

v

Secondary Storage

(Disk)

Advantages

1. Executes programs larger than RAM.
2. Better memory utilization.
3. Increases multiprogramming.
4. Reduces memory wastage.

Disadvantages

1. Page faults can slow down execution.
2. Requires additional hardware and software support.
3. Excessive paging may lead to thrashing.

10.a) Discuss about files and its attributes..

File

A **file** is a named collection of related information stored on secondary storage such as a hard disk. Files are used to store data, programs, documents, images, and other information permanently.

File Attributes

A file has several attributes that describe its properties:

1. **Name**
   - The symbolic name used to identify the file.
   - Example: `student.txt`
2. **Identifier**
   - A unique number assigned by the operating system to identify the file.
3. **Type**
   - Indicates the kind of file.
   - Examples: `.txt`, `.doc`, `.jpg`, `.exe`
4. **Location**
   - Specifies where the file is stored on the storage device.
5. **Size**

    - Indicates the current size of the file in bytes.
6. **Protection**
    - Defines access rights such as read, write, and execute permissions.
7. **Time and Date**
    - Stores information about file creation, modification, and last access times.
8. **Owner/User Identification**
    - Identifies the owner of the file and associated user information.

Diagram

Plain Text
File
|

| | | | | |
Name Type Size Owner Location Protection

b) Explain how different operations can be performed on files using its system calls. [5+5]

File operations are performed through **system calls**, which provide an interface between user programs and the operating system for managing files.

Common File Operations and System Calls

1. Create a File

- A new file is created in the file system.
- **System Call:** `create()`

2. Open a File

- Opens an existing file for reading, writing, or both.
- **System Call:** `open()`

3. Read a File

- Reads data from a file into memory.
- **System Call:** `read()`

4. Write to a File

- Writes data from memory into a file.
- **System Call:** `write()`

5. Reposition File Pointer

- Moves the file pointer to a specified location within the file.
- **System Call:** `lseek()`

6. Close a File

- Closes the file after operations are completed.
- **System Call:** `close()`

7. Delete a File

- Removes a file from the file system.
- **System Call:** `delete()` or `unlink()`

File Operation Flow

Plain Text
Create
↓
Open
↓
Read / Write
↓
Reposition (if needed)
↓
Close
↓
Delete (optional)

Summary Table

Operation	System Call
Create File	`create()`
Open File	`open()`
Read File	`read()`
Write File	`write()`
Reposition Pointer	`lseek()`
Close File	`close()`
Delete File	`unlink()`

OR

11.a) Discuss about file allocation methods in detail.

File allocation methods determine how files are stored on secondary storage devices. The operating system uses these methods to allocate disk space efficiently.

There are three main file allocation methods:

1. Contiguous Allocation

In this method, all file blocks are stored in **consecutive memory locations**.

Example

Plain Text
File A : Blocks 10, 11, 12, 13

Advantages

- Simple to implement.
- Fast sequential and direct access.

Disadvantages

- Causes external fragmentation.
- File size must be known in advance.

---

2. Linked Allocation

In this method, file blocks can be stored anywhere on the disk. Each block contains a pointer to the next block.

Example

Plain Text
File A

10 → 25 → 7 → 40 → NULL

Advantages

- No external fragmentation.
- Files can grow dynamically.

Disadvantages

- Pointer space is required.
- Direct access is difficult.
- If a pointer is lost, data may become inaccessible.

3. Indexed Allocation

In this method, each file has a separate **index block** containing addresses of all file blocks.

Example

Plain Text
Index Block

10
25
7
40

File Blocks:
10, 25, 7, 40

Advantages

- Supports both direct and sequential access.
- No external fragmentation.

Disadvantages

- Extra space is needed for the index block.
- Wastage of space for small files.

Comparison of Allocation Methods

Method	Advantages	Disadvantages
Contiguous	Fast access	External fragmentation
Linked	Dynamic growth	No direct access
Indexed	Direct access supported	Extra index block required

Diagram

Plain Text
File Allocation Methods
|

| | |
Contiguous Linked Indexed
Allocation Allocation Allocation

b) Discuss how access can be controlled to files and protection can be provided. [5+5]

File protection is the mechanism used by the operating system to prevent unauthorized access to files and ensure data security. Access control determines who can use a file and what operations can be performed on it.

Access Rights

The operating system provides different types of access permissions:

1. **Read (R)**
   - Allows viewing or reading the contents of a file.
2. **Write (W)**
   - Allows modifying or updating the file contents.
3. **Execute (X)**
   - Allows executing a program file.
4. **Append (A)**
   - Allows adding data to the end of a file.
5. **Delete (D)**
   - Allows removing a file from the system.

Methods of File Protection

1. Access Control Matrix

- Specifies the access rights of users for each file.
- Rows represent users and columns represent files.

Example

Plain Text
File1 File2
User A RW R
User B R RWX

2. Access Control List (ACL)

- Each file maintains a list of users and their permitted operations.

Example

Plain Text
File1:
User A → Read, Write
User B → Read
User C → Execute

3. Password Protection

- A password is assigned to a file.
- Only users with the correct password can access the file.

Benefits of File Protection

- Prevents unauthorized access.
- Protects sensitive data.
- Ensures data integrity.
- Provides controlled sharing of files.

ONE LAST CHANCE

a) What is a thread?
A **thread** is the smallest unit of CPU execution within a process.

b) Discuss about time-shared operating system.
A **time-shared operating system** allows multiple users/processes to share the CPU by allocating each a small time slice.

c) What is Preemptive Scheduling?
Preemptive scheduling is a CPU scheduling method in which the operating system can interrupt a running process and assign the CPU to another process.

d) Define Turnaround Time.
Turnaround time is the total time taken from process submission to its completion.
Formula: Turnaround Time = Completion Time − Arrival Time

e) What is Named Pipe?
A **named pipe** is an inter-process communication (IPC) mechanism that enables data exchange between processes using a unique pipe name.

f) Define the usage of Counting Semaphore.
A **counting semaphore** is used to control access to multiple instances of a shared resource.

g) What is Translation Lookaside Buffer (TLB)?
A **Translation Lookaside Buffer (TLB)** is a high-speed cache that stores recent virtual-to-physical address translations.

h) What is Fragmentation?
Fragmentation is the wastage of memory caused by inefficient allocation and deallocation of memory blocks.

i) Discuss about ioctl system call.
The **ioctl (Input/Output Control)** system call is used to control and configure device-specific operations that are not handled by standard read/write calls.

j) Discuss about Exclusive Lock.
An **exclusive lock** allows only one process or thread to access a resource at a time, preventing simultaneous access by others.

ONE LAST CHANCE