Code No: 154BR **R18**

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD
B. Tech II Year II Semester Examinations, April/May - 2023
OPERATING SYSTEMS
(Common to CSE, IT, CSBS, CSIT, ITE, CE(SE), CSE(CS), CSE(AI&ML), CSE(DS), CSE(IOT), CSE(N))

Time: 3 Hours **Max. Marks: 75**

Note: i) Question paper consists of Part A, Part B.
ii) Part A is compulsory, which carries 25 marks. In Part A, answer all questions.
iii) In Part B, Answer any one question from each unit. Each question carries 10 marks and may have a, b as sub questions.

PART – A

(25 Marks)

1.a) Define the essential properties of parallel operating systems. [2]
b) How does multiprogramming increase CPU utilization? [3]
c) Write about wait command. [2]
d) How does priority scheduling differ from round robin method? [3]
e) What is a message queues? [2]
f) Give an example of the situation describing deadlock. [3]
g) Define segmentation. [2]
h) What is the purpose of paging the page tables? [3]
i) What is a file? [2]
j) List down various file attributes. [3]

PART – B

(50 Marks)

2.a) In a multiprogramming and time-sharing environment, several users share the system
simultaneously. This situation can result in various security problems. What are two
such problems?
b) Can we ensure the same degree of security in a time-shared machine as in a dedicated machine? Explain your answer. [5+5]

OR

3.a) Under what circumstances would a user be better off using a timesharing system rather than a PC or single-user workstation.
b) Distinguish between the client–server and peer-to-peer models of distributed systems. [5+5]
4.a) Describe the differences among short-term, medium-term, and long-term scheduling.
b) Can a multithreaded solution using multiple user-level threads achieve better performance on a multiprocessor system than on a single-processor system? [5+5]

OR

5.a) Describe the actions taken by a thread library to context switch between user-level threads.
b) Why is it important for the scheduler to distinguish I/O-bound programs from CPU-bound programs? [5+5]

OR

6.a) Demonstrate that monitors and semaphores are equivalent as they can be used to implement the same types of synchronization problems.
b) What is critical-section problem? Give a classic Peterson's solution to the critical-section problem. [5+5]

OR

7. Discuss the tradeoff between fairness and throughput of operations in the readers-writers problem. Propose a method for solving the readers-writers problem without causing starvation. [10]
8. Given five memory partitions of 100 KB, 500 KB, 200 KB, 300 KB, and 600 KB (inorder), how would each of the first-fit, best-fit, and worst-fit algorithms place processes of 212 KB, 417 KB, 112 KB, and 426 KB (in order)? Which algorithm makes the most efficient use of memory? [10]

OR

9. Explain the concept of Least Recently Used memory page replacement method and how it is different from First In First Out (FIFO) page replacement method. [10]

10.a) What are the advantages of Contiguous allocation? What are the drawbacks of contiguous allocation of disk space?

b) Explain the following commands: lseek, stat, ioctl. [4+6]

OR

11. Explain in detail about the common schemes for defining the logical structure of a
directory. [10]

Answer Key

1.a) Define the essential properties of parallel operating systems.

Parallel operating systems are designed to manage the simultaneous execution of multiple processes across multiple CPUs. The essential properties include:

1.Concurrency: Ability to execute multiple processes simultaneously, enhancing computational speed and efficiency.

2.Synchronization: Mechanisms to coordinate processes, ensuring they operate correctly without interfering with each other.

1.b)How does multiprogramming increase CPU utilization?
Multiprogramming increases CPU utilization by:

Running multiple programs concurrently: When one program is waiting for I/O operations, another program can use the CPU.

1.Maximizing resource use: Ensures that the CPU is not idle and is always executing instructions from some process.

2.Reducing idle time: Overlapping CPU and I/O operations reduce the total waiting time for processes.

1.c) Write about the wait command.

The wait command in operating systems, particularly in Unix/Linux, is used to make a script or process pause until all background processes have completed. It is often used in shell scripts to ensure that certain tasks are completed before moving on to subsequent commands.

1.d) How does priority scheduling differ from the round-robin method?

Priority Scheduling: Processes are assigned a priority, and the CPU is allocated to the process with the highest priority. It can lead to starvation of low-priority processes.

Round-Robin Scheduling: Each process is assigned a fixed time slot (quantum) in a cyclic order. It ensures fair sharing of the CPU and is more suitable for time-sharing systems.

1.e) What is a message queue?

A message queue is an inter-process communication (IPC) mechanism that allows processes to exchange messages. It enables asynchronous communication where messages are stored in a queue until they are retrieved and processed by the receiving process.

1.f) Give an example of the situation describing deadlock.

A classic example of a deadlock is the "dining philosophers problem" where multiple philosophers sit at a table with a fork between each pair. Each philosopher needs two forks to eat. If every philosopher picks up the fork on their right at the same time, no one can pick up the second fork, causing a deadlock where none can eat.

1.g) Define segmentation.

Segmentation is a memory management technique that divides a program's memory into different segments, each of which may contain a different type of data or code (e.g., stack segment, data segment, code segment). It allows for more efficient and logical use of memory.

1.h) What is the purpose of paging the page tables?

Paging the page tables, also known as multi-level paging, is used to:

Reduce memory usage: Page tables are split into multiple levels to avoid the need for a large, contiguous block of memory for a single page table.
Improve efficiency: By organizing page tables in a hierarchical manner, the system can handle larger address spaces more efficiently.
Minimize overhead: It reduces the overhead of managing a large page table by breaking it down into manageable parts.

1.i) What is a file?
A file is a collection of data stored on a storage device, identified by a filename. It can contain text, binary data, or executable code and is used to organize and manage data on a computer.

1.j) List down various file attributes.
File attributes typically include:
Name: The human-readable identifier for the file.
Type: The file format or extension (e.g., .txt, .exe).
Size: The amount of data stored in the file.
Location: The file's physical or logical address on the storage medium.
Permissions: Access control settings (read, write, execute) for different users.
Timestamps: Information about creation, last modification, and last access times.

2.a) In a multiprogramming and time-sharing environment, several users share the system simultaneously. This situation can result in various security problems. What are two such problems?
Data Privacy and Confidentiality Breaches:
1.Unauthorized Access: In a time-sharing environment, multiple users have access to the same system resources. This can lead to unauthorized access to sensitive data if proper access controls are not in place.
2.Shared Resources: Users may inadvertently leave sensitive information in shared areas (e.g., temporary files or shared memory), which can be accessed by other users, leading to privacy breaches.
3.System Integrity Violations:
4.Malicious Code Execution: Malicious users might exploit system vulnerabilities to inject and execute harmful code, affecting other users' processes or the overall system integrity.
5.Resource Misuse: One user can consume excessive system resources (CPU, memory, disk space), potentially leading to a denial of service for other users

and degrading system performance.

2.b) Can we ensure the same degree of security in a time-shared machine as in a dedicated machine? Explain your answer.

Ensuring the same degree of security in a time-shared machine as in a dedicated machine is challenging but can be approached through several **measures:**

Isolation Mechanisms:

1.Virtualization: Using virtualization technologies, each user can be given a separate virtual machine (VM) environment, effectively isolating their processes and data from others. This can provide security akin to dedicated machines.

2.Containers: Containers can also isolate applications at the operating system level, providing a secure environment for each user's processes and minimizing the risk of interference.

Access Control and Authentication:

3.Strong Authentication: Implementing robust authentication mechanisms (e.g., multi-factor authentication) can help ensure that only authorized users access the system.

4.Role-Based Access Control (RBAC): RBAC ensures that users have only the permissions necessary for their roles, reducing the risk of unauthorized access to critical resources.

Monitoring and Auditing:

5.Continuous Monitoring: Real-time monitoring of user activities can help detect and respond to security incidents promptly.

6.Auditing and Logging: Maintaining detailed logs of user activities can help in post-incident analysis and ensure compliance with security policies.

Data Encryption:

7.Encryption at Rest and in Transit: Encrypting data both at rest and in transit can protect sensitive information from unauthorized access, even if physical security is compromised.

Regular Security Updates and Patching:

8.System Updates: Regularly updating and patching the operating system and applications can close security vulnerabilities that could be exploited in a time-shared environment.

3.a) Under what circumstances would a user be better off using a timesharing system rather than a PC or single-user workstation?

Resource Sharing:

Cost Efficiency: When multiple users need to share expensive resources like powerful processors, large memory, or specialized software, a time-sharing

system can be more cost-effective than providing each user with a dedicated workstation.
Collaboration: Environments requiring collaborative work, such as software development teams or research groups, can benefit from shared access to common data and applications.
1.Workload Variability:
Fluctuating Workloads: Users with sporadic or varying workloads can take advantage of the flexible resource allocation of time-sharing systems, ensuring that computing resources are utilized efficiently.
2.Maintenance and Support:
Centralized Maintenance: Time-sharing systems centralize maintenance and updates, reducing the overhead and complexity associated with managing multiple individual PCs or workstations.
Technical Support: Organizations with limited IT support can benefit from the streamlined management and troubleshooting offered by centralized time-sharing systems.
3.Accessibility:
Remote Access: Users who need access to computing resources from multiple locations can benefit from the remote accessibility provided by time-sharing systems.
Thin Clients: In environments where users only need basic input/output devices, using thin clients connected to a time-sharing system can be more practical and cost-effective.
4.Application Requirements:
Specialized Applications: Applications that require substantial processing power, memory, or other resources might be more efficiently run on a time-sharing system, allowing users to leverage shared high-performance computing resources.
3.b) Distinguish between the client–server and peer-to-peer models of distributed systems.
Architecture:
Client-Server Model: This model involves a central server that provides services and resources to multiple clients. Clients request services, and the server responds, managing resources centrally.
Peer-to-Peer (P2P) Model: In a P2P model, each node (peer) in the network acts as both a client and a server. Resources and services are distributed across the peers without a centralized server.
1.Scalability:

Client-Server Model: Scalability can be limited by the server's capacity. As the number of clients grows, the server may become a bottleneck, requiring more powerful hardware or additional servers.
P2P Model: P2P systems are generally more scalable as each peer contributes resources. The system can grow dynamically as more peers join, distributing the load more evenly.
2.Reliability and Fault Tolerance:
Client-Server Model: If the central server fails, the entire system can become inoperative. Redundancy and backup servers can mitigate this risk but add complexity and cost.
P2P Model: P2P systems are inherently more fault-tolerant. If one peer fails, the system can continue to operate using the resources of other peers.
3.Resource Management:
Client-Server Model: Resource management is centralized, with the server controlling access to resources. This can simplify security and resource allocation but may lead to bottlenecks.
P2P Model: Resource management is decentralized, with each peer managing its resources. This can improve distribution and utilization but may complicate coordination and security.
4.Examples:
Client-Server Model: Web applications, email systems, and database services typically use a client-server model.
P2P Model: File-sharing networks (e.g., BitTorrent), decentralized cryptocurrencies (e.g., Bitcoin), and some collaborative platforms use a P2P model.

4.a) Describe the differences among short-term, medium-term, and long-term scheduling.
Short-Term Scheduling (CPU Scheduling):
Purpose: Determines which process from the ready queue will be executed by the CPU next.
Frequency: Occurs frequently, typically every few milliseconds.
Decision Criteria: Based on factors such as process priority, CPU burst time, and fairness.
Impact: Directly affects system responsiveness and CPU utilization.
1.Medium-Term Scheduling (Swapping):
Purpose: Manages the degree of multiprogramming by moving processes in and out of the main memory.

Frequency: Occurs less frequently than short-term scheduling, depending on system load and memory availability.
Decision Criteria: Based on the need to free up memory, balance load, or manage I/O-bound and CPU-bound processes.
Impact: Affects overall system performance and memory utilization.
2.Long-Term Scheduling (Job Scheduling):
Purpose: Controls the admission of new jobs into the system, determining which jobs will be brought into the ready queue.
Frequency: Occurs infrequently, typically when a new job is submitted or at specific intervals.
Decision Criteria: Based on job priorities, resource availability, and system workload.
Impact: Affects the mix of I/O-bound and CPU-bound jobs, system throughput, and job turnaround time.
4.b) Can a multithreaded solution using multiple user-level threads achieve better performance on a multiprocessor system than on a single-processor system?
Yes, a multithreaded solution using multiple user-level threads can achieve better performance on a multiprocessor system compared to a single-processor system. Here’s why:
Parallelism:
1.Multiprocessor System: Multiple processors can execute multiple threads simultaneously, leading to true parallel execution. This can significantly improve performance, especially for CPU-bound tasks that can be divided into independent threads.
Single-Processor System: Threads must be time-multiplexed on a single processor, leading to context switching overhead without real parallelism. Performance improvement is limited compared to multiprocessor systems.
2.Resource Utilization:
Multiprocessor System: Efficient use of multiple CPUs leads to higher overall system throughput and better resource utilization. Threads can run concurrently, making full use of the available processing power.
Single-Processor System: Only one thread can execute at a time, limiting resource utilization to the capacity of a single CPU. Context switching can add overhead and reduce performance.
3.Scalability:
Multiprocessor System: Scalability is enhanced as adding more processors can proportionally increase the capacity to handle more threads and workload,

improving performance further.

Single-Processor System: Scalability is constrained by the single CPU's capacity. Adding more threads can lead to diminishing returns due to increased context switching and overhead.

4.Concurrency:

Multiprocessor System: Higher concurrency can be achieved as multiple threads can be executed in parallel, improving the responsiveness of applications and reducing latency for concurrent operations.

Single-Processor System: Concurrency is limited by the need to switch between threads on a single processor, resulting in less responsive applications compared to a multiprocessor environment.

Examples:

Multiprocessor System: Multithreaded web servers, scientific computations, and data processing applications can see significant performance gains on multiprocessor systems.

Single-Processor System: Multithreading can still improve performance for I/O-bound tasks by overlapping I/O operations with computation but will not achieve the same level of improvement as on a multiprocessor system.

5.a) Describe the actions taken by a thread library to context switch between user-level threads.

Saving the State of the Current Thread:

1.Register Values: The thread library saves the current thread's CPU register values, including the program counter, stack pointer, and other processor state information, to its thread control block (TCB).

2.Stack Pointer: The current position of the stack pointer is stored to ensure the thread's stack can be resumed correctly later.

3.Updating Thread Control Block (TCB):

Status Update: The thread library updates the current thread's status in its TCB to indicate that it is no longer running (e.g., changing its state to ready or waiting).

4.Selecting the Next Thread:

Scheduling Policy: The thread library uses its scheduling policy (e.g., round-robin, priority-based) to select the next thread to run from the ready queue.

Next Thread's TCB: The TCB of the selected thread is retrieved to access its saved state.

5.Restoring the State of the Next Thread:

Register Values: The thread library loads the saved register values of the selected thread from its TCB, restoring the program counter, stack pointer, and other processor state information.
Stack Pointer: The stack pointer is set to the saved value to ensure the thread's stack is correctly positioned for execution.
6.Updating the System Stack:
System Stack: If the system stack is used to manage the threads' context, it is updated to reflect the new thread's stack context, ensuring a smooth transition.
1.Resuming Execution:
Switch Execution: The CPU execution context is switched to the new thread's context, allowing it to resume execution from where it was last interrupted.
2.Handling Synchronization:
Locks and Semaphores: If the thread library supports synchronization primitives, it ensures that locks, semaphores, and other synchronization mechanisms are correctly managed during the context switch.

5.b) Why is it important for the scheduler to distinguish I/O-bound programs from CPU-bound programs?
Efficient Resource Utilization:
I/O-Bound Programs: These programs spend more time waiting for I/O operations to complete than using the CPU. By recognizing I/O-bound programs, the scheduler can ensure that the CPU is not left idle while these programs wait for I/O.
CPU-Bound Programs: These programs require intensive CPU processing. Identifying them helps the scheduler allocate CPU time effectively, ensuring that these programs get the processing time they need without monopolizing the CPU.
1.Improved System Throughput:
Balanced Scheduling: By differentiating between I/O-bound and CPU-bound programs, the scheduler can interleave their execution, maximizing the overall system throughput. While I/O-bound programs wait for I/O, CPU-bound programs can utilize the CPU, and vice versa.
2.Reduced Waiting Time:
Fairness: Ensuring that both types of programs get appropriate attention reduces the waiting time for each. I/O-bound programs can get their I/O requests serviced quickly, while CPU-bound programs can make steady progress, leading to a more balanced system.
3.Enhanced Responsiveness:

Interactive Systems: In systems with interactive users, responsiveness is critical. I/O-bound programs, which often correspond to user interactions, need to be given higher priority to maintain system responsiveness, ensuring that user requests are handled promptly.

4.Avoiding Starvation:

Resource Allocation: Properly distinguishing between the two types of programs helps prevent scenarios where CPU-bound programs starve I/O-bound programs of CPU time or vice versa. This balance ensures that all programs make progress without indefinite delays.

6.a) Demonstrate that monitors and semaphores are equivalent as they can be used to implement the same types of synchronization problems.

Monitors:

1.Monitors are high-level synchronization constructs that allow safe access to shared resources.

2.A monitor encapsulates shared variables, procedures, and the synchronization code (such as condition variables) to manage concurrent access.

3.Only one process can execute a monitor procedure at a time.

Semaphores:

1.Semaphores are low-level synchronization primitives used to control access to shared resources.

2.A semaphore can be either a counting semaphore or a binary semaphore (mutex).

3.Semaphores use P (wait) and V (signal) operations to manage concurrent access.

Equivalence:

1.Monitors using Semaphores:

Monitors can be implemented using semaphores by using a binary semaphore (mutex) to ensure mutual exclusion.

Condition variables in monitors can be implemented using semaphores to block and wake up processes.

```
// Monitor implementation using semaphores
semaphore mutex = 1;  // Binary semaphore for mutual exclusion
semaphore next = 0;   // Semaphore for condition variable
int next_count = 0;

monitor {
    P(mutex);          // Enter monitor
    // Critical section code
    if (next_count > 0) {
        V(next);       // Wake up waiting process
    } else {
        V(mutex);      // Release monitor
    }
}
```

Semaphores using Monitors:
Semaphores can be implemented using monitors by defining P and V operations within the monitor.
The monitor ensures mutual exclusion for the P and V operations and maintains the semaphore count.

```
// Semaphore implementation using monitor
monitor Semaphore {
    int value;
    condition waitQueue;

    procedure P() {
        if (value == 0) {
            waitQueue.wait();
        }
        value--;
    }

    procedure V() {
        value++;
        waitQueue.signal();
    }
}
```

6.b) What is the critical-section problem? Give a classic Peterson's solution to the critical-section problem.

Critical-Section Problem:
The critical-section problem involves ensuring that multiple processes or threads can access shared resources without causing data inconsistency.
A critical section is a segment of code where shared resources are accessed.

The solution must satisfy three requirements: mutual exclusion, progress, and bounded waiting.

1.Peterson's Solution:

Peterson's solution is a classic algorithm to address the critical-section problem for two processes.

It uses two shared variables: flag and turn.

2.Mutual Exclusion: Only one process can enter the critical section at a time, as the other process will be stuck in the busy wait loop.

3.Progress: The processes make a decision to enter the critical section based on the value of turn and the flag array.

4.Bounded Waiting: Each process will eventually get a chance to enter the critical section, as the turn variable ensures alternation.

7. Discuss the tradeoff between fairness and throughput of operations in the readers-writers problem. Propose a method for solving the readers-writers problem without causing starvation.

Fairness vs. Throughput:

Fairness: Ensures that all processes (readers and writers) get a chance to execute without being indefinitely delayed. Fairness prevents starvation, ensuring every process eventually gets access to the shared resource.

Throughput: Refers to the overall rate of completed operations in a system. High throughput aims to maximize resource utilization and process more requests in a given time period.

In the readers-writers problem, achieving fairness can sometimes lead to reduced throughput and vice versa:

Writer Priority: Prioritizing writers ensures they get immediate access once they request it, reducing their waiting time. However, it can lead to reader starvation if there are continuous writer requests.

Reader Priority: Prioritizing readers allows multiple readers to access the resource concurrently, improving throughput for read operations. However, it can lead to writer starvation if there are continuous reader requests.

No Priority: Alternating between readers and writers can balance the system but may reduce throughput if switching overhead is significant.

Proposed Solution: To avoid starvation and balance fairness and throughput, a common solution is to use a fair read-write lock. Here's a method to implement it:

1.Use a Queue for Requests:

Maintain a queue of waiting readers and writers.

Serve requests in the order they arrive to ensure fairness

8. Given five memory partitions of 100 KB, 500 KB, 200 KB, 300 KB, and 600 KB (in order), how would each of the first-fit, best-fit, and worst-fit algorithms place processes of 212 KB, 417 KB, 112 KB, and 426 KB (in order)? Which algorithm makes the most efficient use of memory?

Processes: 212 KB, 417 KB, 112 KB, 426 KB

1.First-Fit:

212 KB: Placed in 500 KB partition (remaining: 288 KB)
417 KB: Placed in 600 KB partition (remaining: 183 KB)
112 KB: Placed in 288 KB partition (remaining: 176 KB)
426 KB: Cannot be placed (no partition large enough)

2.Final Allocation:

Partitions: [100 KB, 176 KB, 200 KB, 300 KB, 183 KB]
Unallocated Process: 426 KB

3.Best-Fit:

212 KB: Placed in 300 KB partition (remaining: 88 KB)
417 KB: Placed in 500 KB partition (remaining: 83 KB)
112 KB: Placed in 200 KB partition (remaining: 88 KB)
426 KB: Placed in 600 KB partition (remaining: 174 KB)

4.Final Allocation:

Partitions: [100 KB, 83 KB, 88 KB, 88 KB, 174 KB]

5.Worst-Fit:

212 KB: Placed in 600 KB partition (remaining: 388 KB)
417 KB: Placed in 388 KB partition (remaining: 183 KB)
112 KB: Placed in 500 KB partition (remaining: 388 KB)
426 KB: Cannot be placed (no partition large enough)

6.Final Allocation:

Partitions: [100 KB, 388 KB, 200 KB, 300 KB, 183 KB]
Unallocated Process: 426 KB

7.Most Efficient Use of Memory:

Best-Fit algorithm makes the most efficient use of memory as it leaves the smallest fragments of unused space, ensuring better utilization of available partitions. In this example, all processes are allocated memory without any left unallocated, which demonstrates optimal memory usage.

9. Explain the concept of Least Recently Used memory page replacement method and how it is different from First In First Out (FIFO) page replacement method.

Least Recently Used (LRU):

Concept: LRU page replacement algorithm replaces the page that has not been used for the longest period of time.

Mechanism: Keeps track of page usage history. When a page needs to be replaced, the page with the oldest last access time is chosen.

Implementation:

Counter Method: Each page entry has a counter; every time a page is referenced, its counter is updated. The page with the smallest counter is replaced.

Stack Method: Maintains a stack of page references. When a page is accessed, it is removed from its current position and placed at the top of the stack. The bottom of the stack holds the least recently used page.

First In First Out (FIFO):

Concept: FIFO page replacement algorithm replaces the oldest page in memory, i.e., the page that has been in memory the longest.

Mechanism: Maintains a queue of pages in memory. When a page needs to be replaced, the page at the front of the queue (the oldest page) is removed and the new page is added to the back of the queue.

Implementation:

Queue Method: Pages are maintained in a queue. When a new page is loaded, it is added to the end of the queue, and the page at the front is removed when replacement is needed.

Differences:

1.Replacement Criteria:

LRU: Replaces the least recently used page, reflecting actual usage patterns and providing better performance in practical scenarios.

FIFO: Replaces the oldest page, without considering how frequently or recently a page has been accessed.

2.Efficiency:

LRU: Requires more complex data structures (counters or stacks) and additional overhead to track page usage, making it more accurate but computationally expensive.

FIFO: Simpler to implement with less overhead, using a straightforward queue structure, but may not always lead to optimal performance.

3.Performance:

LRU: Generally performs better in scenarios where the most recently used pages are likely to be accessed again soon, reducing page faults.
FIFO: Can lead to higher page fault rates, especially in cases where frequently accessed pages are removed simply because they are the oldest (Belady's anomaly).

10.a) What are the advantages of contiguous allocation? What are the drawbacks of contiguous allocation of disk space?
Advantages of Contiguous Allocation:
1.Simplicity: Contiguous allocation is straightforward to implement and manage, with files stored in consecutive blocks.
2.Performance: Sequential access is very fast, as the entire file is stored contiguously, minimizing seek time and rotational latency.
3.Direct Access: Random access is efficient because the starting block and file size allow easy calculation of the physical address of any block within the file.
Drawbacks of Contiguous Allocation:
1.Fragmentation: Over time, the file system can become fragmented, with free space scattered in small blocks, making it difficult to find contiguous blocks for new files.
2.Space Allocation: Requires knowing the file size in advance to allocate a contiguous block, which can lead to wasted space if the file is smaller than the allocated block or insufficient space if the file grows.
3.Compaction: Periodic compaction is needed to consolidate free space, which can be time-consuming and resource-intensive.
10.b) Explain the following commands: lseek, stat, ioctl.
lseek:
Purpose: The lseek system call repositions the file offset of an open file descriptor, allowing random access within the file.
off_t lseek(int fd, off_t offset, int whence);
fd: File descriptor of the open file.
offset: Number of bytes to move the file offset.
whence: Specifies the reference point for offset (SEEK_SET, SEEK_CUR, SEEK_END).
stat:
Purpose: The stat system call retrieves information about a file, such as its size, permissions, and timestamps.
int stat(const char *pathname, struct stat *buf);
pathname: Path to the file.

buf: Pointer to a struct stat where file information will be stored.
ioctl:
Purpose: The ioctl system call performs device-specific input/output operations, allowing communication between user-space programs and kernel drivers.
int ioctl(int fd, unsigned long request, ...);
fd: File descriptor of the open device.
request: Device-specific request code.
Additional arguments depend on the request.

11. Explain in detail about the common schemes for defining the logical structure of a directory.
The logical structure of a directory is crucial in organizing and managing files efficiently within a file system. Various schemes can be employed to define this structure, each with its advantages and drawbacks. The most common schemes include:
1.Single-Level Directory:
Description: All files are contained in a single directory.
Advantages:
Simple to implement and understand.
Easy to locate any file as there is only one directory.
Drawbacks:
Name conflicts: Files must have unique names as all are in the same directory.
Difficult to manage as the number of files increases
Two-Level Directory:
Description: Each user has a separate directory, and all files for that user are contained within their directory.
Advantages:
Files can have the same name if they belong to different users.
Improved organization and reduced name conflicts.
Drawbacks:
More complex than a single-level directory.
Users can only see their files, not others' files, unless permissions are explicitly set
Tree-Structured Directory:
Description: A hierarchy of directories where each directory can contain files and subdirectories, forming a tree structure.
Advantages:
Hierarchical structure provides efficient organization and navigation.

Can easily group related files together.
Supports nested directories, making it scalable for large systems.
Drawbacks:
More complex to implement and manage.
Can become deep and difficult to navigate if not properly organized.