**Solution**

1. Question 1 is compulsory.
2. Attempt any three from remaining five questions.
3. Assume suitable data where required.

<table>
<thead>
<tr>
<th>No</th>
<th>Questions</th>
<th>Marks</th>
</tr>
</thead>
</table>
| 1  | a. Discuss Operating System as a resource manager  
    • Main Memory management  
    • I/O management  
    • Secondary memory management  
    b. Draw process state diagram and explain the following states:
    1. New  
    2. Ready  
    3. Running  
    4. Wait  
    5. Suspended ready  
    6. Suspended wait  
<p>| [5] |
|    | c. Describe Microkernel with a diagram. | [5] |</p>
<table>
<thead>
<tr>
<th>Level of Thread</th>
<th>User-Level Threads</th>
<th>Kernel-Level Threads</th>
</tr>
</thead>
<tbody>
<tr>
<td>Definition</td>
<td>User threads are supported above the kernel by a thread library at the user level. The library provides support for thread creation, scheduling, and management in kernel space.</td>
<td>Kernel threads are supported directly by the operating system.</td>
</tr>
<tr>
<td>Creation</td>
<td>User threads are managed in user space, each process has a separate thread table to track the threads in that process.</td>
<td>No non-time system is needed in each. Also, there is no thread table in each process. Instead, the kernel has a thread table that keeps track of all the threads in the system.</td>
</tr>
<tr>
<td>Management</td>
<td>User threads do not require any new non-blocking system call. The current thread is just blocked in the kernel, or synchronization is used. User threads are the threads that are active. For example, if a thread needs to wait for the thread to be brought in from the disk.</td>
<td>Kernel threads do not require any new thread call. Instead, the kernel can easily check to see if the thread is ready to be brought in from the disk.</td>
</tr>
<tr>
<td>Speed</td>
<td>User-level threads are generally fast to create and manage.</td>
<td>The kernel-level threads are slow and inefficient. For instance, thread operations are hundreds of times slower than that of user-level threads.</td>
</tr>
</tbody>
</table>
For the Banker's algorithm to work, it needs to know three things:
- How much of each resource each process could possibly request [MAX]
Unblock is done by another task (a.k.a. wake up, release, V)
Block is a.k.a. sleep, request, P

(Short Term Scheduling)

(a) Calculate AWT, ATAT, Response Time and Throughput of the following processes using Shortest Job First (Non-Pre-emptive).

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival Time (ms)</th>
<th>Burst Time (ms)</th>
<th>AWT (ms)</th>
<th>ATAT (ms)</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>1</td>
<td>7</td>
<td></td>
<td>0</td>
<td>0.6667</td>
</tr>
<tr>
<td>P2</td>
<td>2</td>
<td>5</td>
<td></td>
<td>0</td>
<td>0.5</td>
</tr>
<tr>
<td>P3</td>
<td>3</td>
<td>1</td>
<td></td>
<td>0</td>
<td>0.3333</td>
</tr>
<tr>
<td>P4</td>
<td>4</td>
<td>2</td>
<td></td>
<td>0</td>
<td>0.25</td>
</tr>
<tr>
<td>P5</td>
<td>5</td>
<td>8</td>
<td></td>
<td>0</td>
<td>0.5</td>
</tr>
</tbody>
</table>

- AWT=6ms
- ATAT=10,6ms
- Throughput=20.8
- Response Time: P1=4; P2=9; P3=5; P4=4; P5=11

(b) What are Semaphores? Differentiate between Counting and Binary Semaphores. Discuss Readers Writers Problem. Write code of its solution using semaphores.

- Synchronization tool used for synchronization. It uses wait and signal.
- Binary semaphore is either 0 or 1; in Counting semaphore can be initialized to more than 1.
- How much of each resource each process is currently holding [ALLOCATED]
- How much of each resource the system currently has available [AVAILABLE]

Resources may be allocated to a process only if the amount of resources requested is less than or equal to the amount available; otherwise, the process waits until resources are available.

Some of the resources that are tracked in real systems are memory, semaphores and interface access. The Banker's Algorithm derives its name from the fact that this algorithm could be used in a banking system to ensure that the bank does not run out of resources, because the bank would never allocate its money in such a way that it can no longer satisfy the needs of all its customers. By using the Banker's algorithm, the bank ensures that when customers request money the bank never leaves a safe state. If the customer's request does not cause the bank to leave a safe state, the cash will be allocated, otherwise the customer must wait until some other customer deposits enough.

Basic data structures to be maintained to implement the Banker's Algorithm:

Let n be the number of processes in the system and m be the number of resource types. Then we need the following data structures:

- Available: A vector of length m indicates the number of available resources of each type. If Available[j] = k, there are k instances of resource type R_j available.
- Max: An n x m matrix defines the maximum demand of each process. If Max[i,j] = k, then P_i may request at most k instances of resource type R_j.
- Allocation: An n x m matrix defines the number of resources of each type currently allocated to each process. If Allocation[i,j] = k, then process P_i is currently allocated k instances of resource type R_j.
- Need: An n x m matrix indicates the remaining resource need of each process. If Need[i,j] = k, then P_i may need k more instances of resource type R_j to complete the task.

Note: Need[i,j] = Max[i,j] - Allocation[i,j], n x m - a.

---

b. Compare the performance of FIFO, LRU and Optimal based on number of page hit for the following string. Frame size = 3; String (pages): 1 2 3 4 5 2 1 3 3 2 4 5

FIFO: No of Hits=2; page fault=10
LRU: No of Hits=2; page fault=10
Although there are other algorithms that reduce the seek time of all

TYPES OF DISK SCHEDULING ALGORITHMS

Interrupt driven I/O

Programmed I/O

Explain interrupt driven I/O and discuss the advantages of interrupt driven I/O over

Cpl to memory

Cpl to CPU

Read

Execute

Poll

Read
e

Poll

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU

Programmed I/O

I/O to CPU
Elevator (SCAN)
Circular SCAN (C-SCAN)
LOOK
C-LOOK

These algorithms are not hard to understand, but they can confuse someone because they are so similar. What we are striving for by using these algorithms is keeping Head Movements (# tracks) to the least amount as possible. The less the head has to move the faster the seek time will be. I will show you and explain to you why C-LOOK is the best algorithm to use in trying to establish less seek time.

Given the following queue -- 95, 180, 34, 119, 11, 123, 62, 64 with the Read-write head initially at the track 50 and the tail track being at 199 let us now discuss the different algorithms.

1. *First Come - First Serve*

   ![Diagram](image)

   *(FCFS)*  

   All incoming requests are placed at the end of the queue. Whatever number that is next in the queue will be the next number served. Using this algorithm doesn't provide the best results. To determine the number of head movements you would simply find the number of tracks it took to move from one request to the next. For this case it went from 50 to 95 to 180 and so on. From 50 to 95 it moved 45 tracks. If you tally up the total number of tracks you will find how many tracks it had to go through before finishing the entire request. In this example, it had a total head movement of 640 tracks. The disadvantage of this algorithm is noted by the oscillation from track 50 to track 180 and then back to track 11 to 123 then to 64. As you will soon see, this is the worse algorithm that one can use.

2. *Shortest Seek Time First*
This approach works like an elevator does. If scans down towards the nearest end and then when it hits the bottom it scans up serving the requests in order. This is not an optimal one. There is a great chance that some requests will take place. The reason for this is if there were a lot of requests close to each other the other requests will never be handled since the distance will be always be greater.

Although 2 requests between them and not 18 if were to go the other way. Although 12 requests away from 62 instead of 34. The next case would be to move from 62 to 64 instead of 34 since there are only 2 requests between them and not 18 if were to go the other way. For example, the next shortest distance would be 62 instead of 34 since it is only 12 requests away from 62 and 16 moves away from 34. This process is only 12 requests away from 62 and 16 moves away from 34. The next shortest distance would be 62 instead of 34 since it is only 12 requests away from 62 and 16 moves away from 34. The process would continue until all the processes are taken care of. For example, the next shortest distance according to next shortest distance, request is served.
is more optimal than the previous algorithm, but it is not the best.

4. Circular Scan (C-SCAN)
Circular scanning works just like the elevator to some extent. It begins its scan toward the nearest end and works it way all the way to the end of the system. Once it hits the bottom or top it jumps to the other end and moves in the same direction. Keep in mind that the huge jump doesn't count as a head movement. The total head movement for this algorithm is only 187 track, but still this isn't the most sufficient.

5. C-LOOK
This is just an enhanced version of C-SCAN. In this the scanning doesn't go past the last request in the direction that it is moving. It too jumps to the other end but not all the way to the end. Just to the furthest request. C-SCAN had a total movement of 187 but this scan (C-LOOK) reduced it down to 157 tracks.

From this you were able to see a scan change from 644 total head movements to just 157. You should now have an understanding as to why
<table>
<thead>
<tr>
<th>Disk Allocation</th>
<th>Indexed Allocation</th>
<th>Linked Allocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Files are allocated block by block.</td>
<td>Files are allocated sequentially and a pointer is needed to access the file.</td>
<td>Files are allocated sequentially and a pointer is needed to access the file.</td>
</tr>
<tr>
<td>Indexed in case of direct access files.</td>
<td>Efficient in case of sequential access files.</td>
<td>Efficient in case of sequential access files.</td>
</tr>
<tr>
<td>No external fragmentation.</td>
<td>No external fragmentation.</td>
<td>No external fragmentation.</td>
</tr>
<tr>
<td>Block header points to first block of a file.</td>
<td>Each file contains a list of links to disk blocks.</td>
<td>Each file contains a list of links to disk blocks.</td>
</tr>
<tr>
<td>Each block is indexed by a pointer to disk blocks.</td>
<td>Linked Allocation</td>
<td>Indexed Allocation</td>
</tr>
<tr>
<td>Indexed Allocation</td>
<td>Linked Allocation</td>
<td>Indexed Allocation</td>
</tr>
<tr>
<td>Indexed Allocation</td>
<td>Linked Allocation</td>
<td>Indexed Allocation</td>
</tr>
</tbody>
</table>

**Indexed Allocation**
- Files are allocated sequentially and a pointer is needed to access the file.
- Indexed in case of direct access files.
- No external fragmentation.
- Each block is indexed by a pointer to disk blocks.

**Linked Allocation**
- Files are allocated sequentially and a pointer is needed to access the file.
- Each file contains a list of links to disk blocks.
- Linked Allocation

**Sequential Access**
- Files are accessed sequentially in the manner in which they are stored.
- Sequential access

**Direct/Random Access**
- Files are accessed randomly.
- Direct/Random access

**Indexed Sequential Access**
- Files are accessed sequentially via an index.
- Indexed Sequential Access

**Several Ways to Access Files**
- The access mechanism refers to the manner in which the records of a file may be accessed.

6. Discuss various file allocation mechanisms and their advantages.

(1)