USN | | | | | | | | | |

Internal Assessment Test 2 – January 2024

**SCHEME AND SOLUTION**

| Sub: | **Operating Systems** | | | | | Sub Code: | **BCS303** | Branch: | **CSE** |
|---|---|---|---|---|---|---|---|---|---|
| Date: | **19/01/24** | Duration: | **90 minutes** | Max Marks: | **50** | Sem / Sec: | **III / A, B, C** | | **OBE** |

| | | **Answer any FIVE FULL Questions** | **MARKS** | **CO** | **RBT** |
|---|---|---|---|---|---|
| 1 | a | What is Critical Section Problem? Draw the general structure of a process with critical section. If you are to provide a solution for Critical Section Problem, explain the requirements that you have to satisfy. | [5] | 2 | L2 |

- Consider a system consisting of 11 processes $\{P_o, P_I, \ldots, P_{n-1}\}$. 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.
- The important feature is that, when one process is executing in its critical section, no other process is to be allowed to execute in its critical section. i.e., no two processes are executing in their critical sections at same time.
- 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**.

```
do {
    entry section
        critical section
    exit section
        remainder section
} while (TRUE);
```

**Figure 6.1** General structure of a typical process $P_i$.

**1. Mutual Exclusion** - If process $P_i$ is executing in its critical section, then no other processes can be executing in their critical sections

**2. Progress** - If no process is executing in its critical section and there exist some processes that wish to enter their critical section, then the selection of the processes that will enter the critical section next cannot be postponed indefinitely

| | | | | | |
|---|---|---|---|---|---|
| | | **3. Bounded Waiting -** A bound must exist on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted | | | |
| | b | Write the logic of using locks to solve Critical Section Problem. Explain how Swapping helps to solve the CSP<br>do {<br>    *acquire lock*<br>        critical section<br>    *release lock*<br>        remainder section<br>} while (TRUE);<br><br>void Swap (boolean *a, boolean *b)<br>{<br>    boolean temp = *a;<br>    *a = *b;<br>    *b = temp:<br>}<br>Shared Boolean variable lock initialized to FALSE;<br>Each process has a local Boolean variable key.<br>Solution:<br>do {<br>    key = TRUE;<br>    while ( key == TRUE)<br>        Swap (&lock, &key );<br><br>        // critical section<br>    lock = FALSE;<br>        // remainder section<br>} while ( TRUE); | [5] | 2 | L3 |
| 2 | a | Explain in detail about Semaphore as a Synchronization tool.<br>Semaphore is a synchronization tool that controls the access to shared resources among multiple processes. It limits the number of processes that can access the resource simultaneously, preventing data inconsistencies.<br>Semaphore S – integer variable<br>Two standard operations modify S:<br>Wait () and signal ()<br>Originally called P() and V()<br>Less complicated | [5] | 2 | L2 |

| | | | | | |
|---|---|---|---|---|---|
| | | Can only be accessed via two indivisible (atomic) operations<br>wait (S)<br>{<br>while S <= 0;<br>//No Operation<br>S--;<br>}<br>signal (S)<br>{<br>S++;<br>} | | | |
| | b | In a multi-threaded environment, different processes may need to access shared resources concurrently. This could lead to conflicts & potential data inconsistency. Propose a synchronization mechanism that allows multiple readers simultaneous access to shared data while ensuring exclusive access for a single writer. Discuss how this approach avoids race conditions and guarantees data consistency.<br>• Shared Data<br>○ Data set<br>○ Semaphore mutex initialized to 1.<br>○ Semaphore wrt initialized to 1.<br>○ Integer readcount initialized to 0.<br>• **The structure of a writer process**<br>do {<br>wait (wrt) ;<br><br>// writing is performed<br>signal (wrt) ;<br>} while (true)<br>• **The structure of a reader process**<br>do {<br>wait (mutex) ;<br>readcount ++ ;<br>if (readcount == 1) wait (wrt) ;<br>signal (mutex)<br><br>// reading is performed<br>wait (mutex) ; | [5] | 2 | L2 |

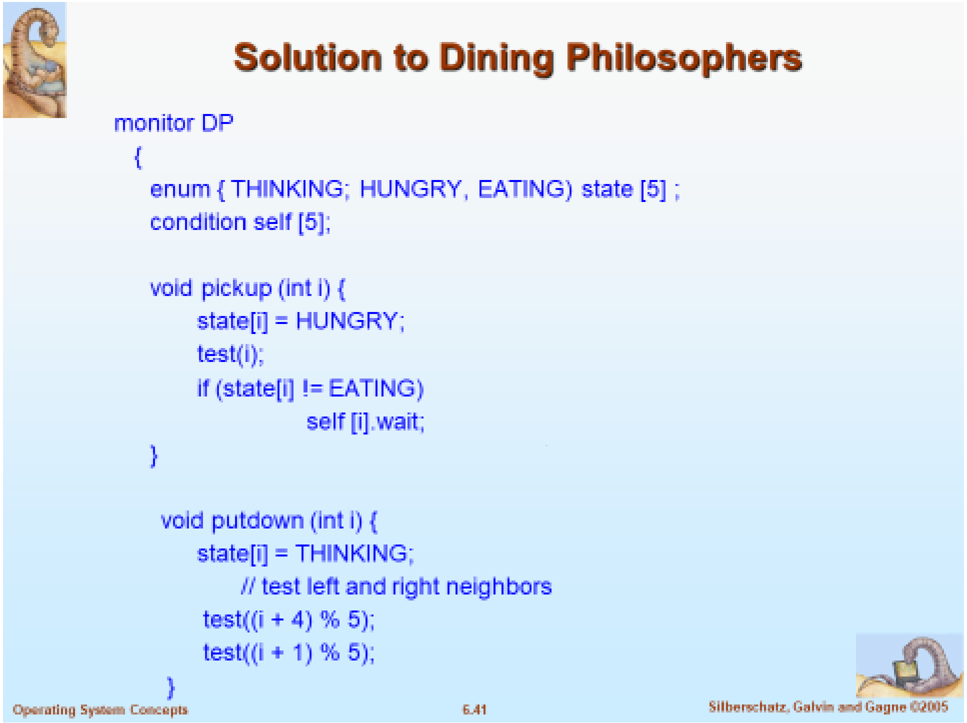
<table>
<tr><td></td><td></td><td>

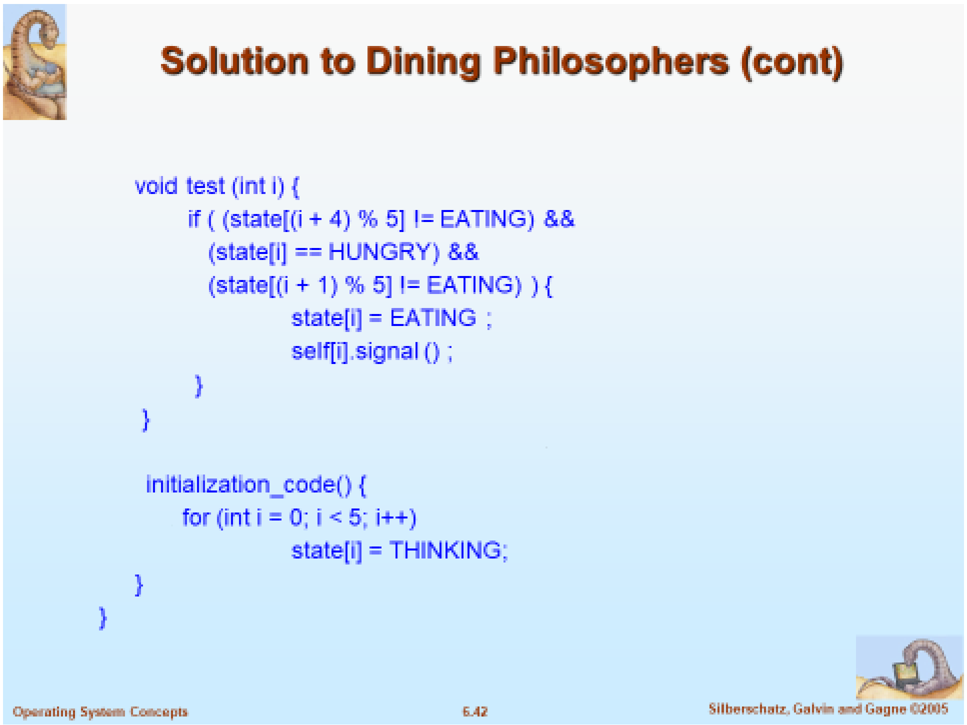
```
        readcount - - ;
        if (readcount == 0)  signal (wrt) ;
        signal (mutex) ;
    } while (true)
```

</td><td></td><td></td><td></td></tr>
<tr><td>3</td><td>a</td><td>

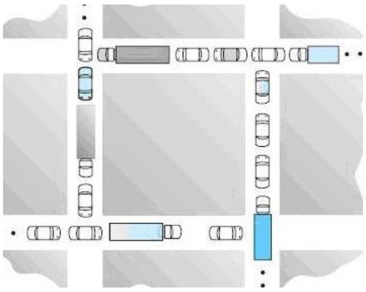
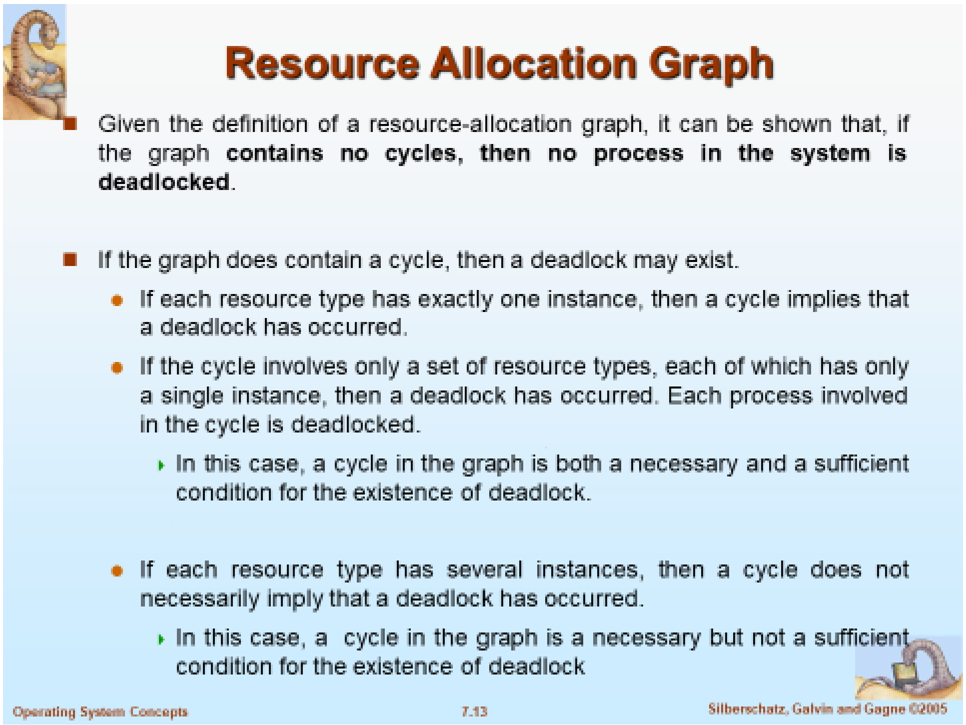
What are Monitors? Help the Dining Philosophers to solve the problem of synchronization using Monitors.

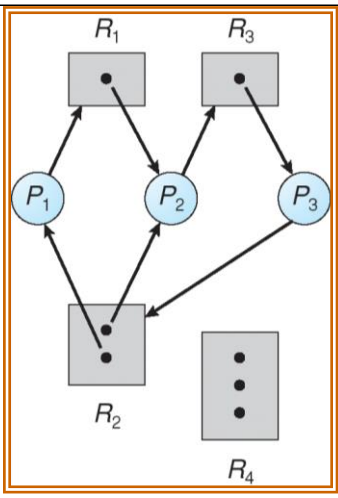
Monitors are high-level synchronization constructs that encapsulate shared data and the operations that can be performed on that data in a single unit called a monitor.

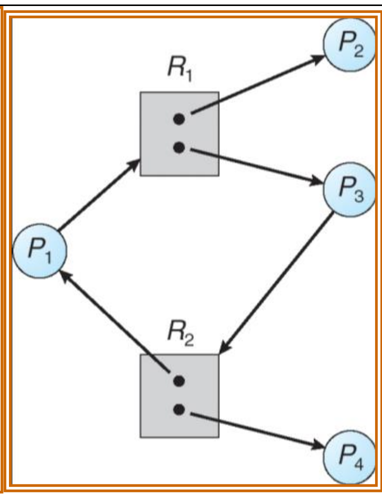



</td><td>[5]</td><td>2</td><td>L2</td></tr>
</table>

| | | | | | |
|---|---|---|---|---|---|
| | b | Consider the traffic depicted in the figure.<br><br>What is a deadlock? Show that the four necessary conditions for deadlock indeed hold in this example.<br>1. **Mutual exclusion:** only one process at a time can use a resource.<br>2. **Hold and wait:** a process holding at least one resource is waiting to acquire additional resources held by other processes.<br>3. **No preemption:** a resource can be released only voluntarily by the process holding it, after that process has completed its task.<br>4. **Circular wait:** there exists a set $\{P_0, P_1, \ldots, P_0\}$ of waiting processes such that $P_0$ is waiting for a resource that is held by $P_1$, $P_1$ is waiting for a resource that is held by $P_2$, …, $P_{n-1}$ is waiting for a resource that is held by $P_n$, and $P_n$ is waiting for a resource that is held by $P_0$. | [5] | 2 | L2 |
| 4 | a | What is Resource Allocation Graph? Explain how RAG is very useful in describing deadlock by considering own example.<br><br> | [5] | 2 | L3 |






- If graph contains no cycles ⇒ no deadlock.
- If graph contains a cycle ⇒
  - if only one instance per resource type, then deadlock.
  - if several instances per resource type, possibility of deadlock.

b

| Process | Allocation | | | Max | | | Available | | |
|---|---|---|---|---|---|---|---|---|---|
| | A | B | C | A | B | C | A | B | C |
| P0 | 0 | 0 | 2 | 0 | 0 | 4 | 1 | 0 | 2 |
| P1 | 1 | 0 | 0 | 2 | 0 | 1 | | | |
| P2 | 1 | 3 | 5 | 1 | 3 | 7 | | | |
| P3 | 6 | 3 | 2 | 8 | 4 | 2 | | | |
| P4 | 1 | 4 | 3 | 1 | 5 | 7 | | | |

Answer the following questions using the banker's algorithm:

(i) Can a request (0,0,2) from process P2 be granted immediately?

| | Max | Allocation | Need |
|---|---|---|---|
| P0 | 0 0 4 | 0 0 2 | 0 0 2 |
| P1 | 2 0 1 | 1 0 0 | 1 0 1 |
| P2 | 1 3 7 | 1 3 5 | 0 0 2 |
| P3 | 8 4 2 | 6 3 2 | 2 1 0 |
| P4 | 1 5 7 | 1 4 3 | 0 1 4 |

| | Need | ≤ Available | ? |
|---|---|---|---|
| P0 | 0 0 2 | 1 0 2 | ✓ |
| P1 | 1 0 1 | 1 0 4 | ✓ |
| P2 | 0 0 2 | 2 0 4 | ✓ |
| P3 | [illegible] | 3 3 9 | ✓ |
| P4 | 1 4 3 | 9 6 11 | ✓ |

Safe Sequence <P0, P1, P2, P3, P4>

Request of (0,0,2) from P2 ?

1) Req ≤ Need ⇒ 0 0 2 ≤ 0 0 2 True

2) Req ≤ Available ⇒ 0 0 2 ≤ 9 6 11 True

3) Pretend like Allocated and update the Matrix

| Process | Max | Alloc | Need |
|---|---|---|---|
| P0 | 0 0 4 | 0 0 2 | 0 0 2 |
| P1 | 2 0 1 | 1 0 0 | 1 0 1 |
| P2 | 1 3 7 | 1 3 7 | 0 0 0 |
| P3 | 8 4 2 | 6 3 2 | 2 1 0 |
| P4 | 1 5 7 | 1 4 3 | 0 1 4 |

Avail: 1 0 0

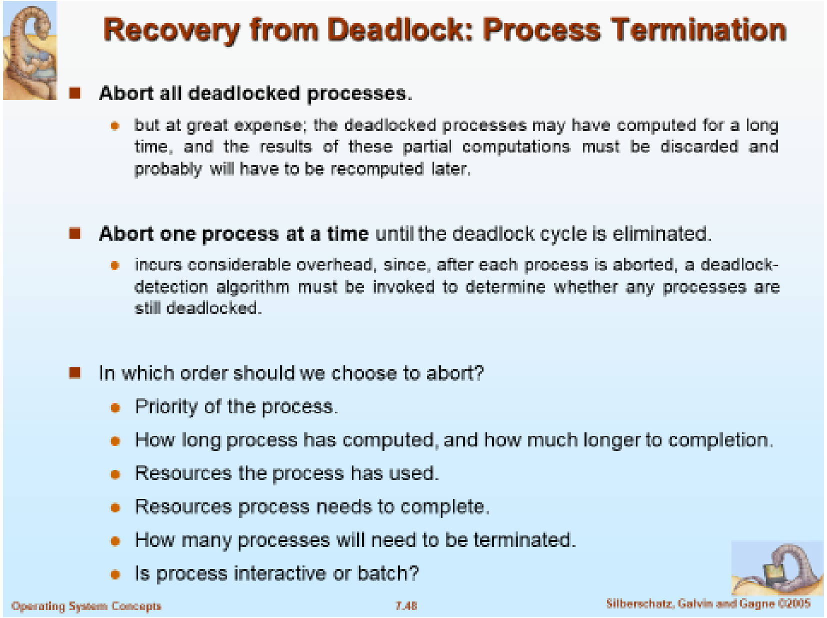
| | Need | ≤ Avail | ? |
|---|---|---|---|
| P0 | 0 0 2 | 1 0 0 | ✗ |
| P1 | 1 0 1 | 1 0 0 | ✗ |
| P2 | 0 0 0 | 1 0 0 | ✓ |
| P3 | 2 1 0 | 2 3 7 | ✓ |
| P4 | 0 1 4 | 8 6 9 | ✓ |
| P0 | 0 0 2 | 8 10 12 | ✓ |
| P1 | 1 0 1 | 9 10 14 | ✓ |

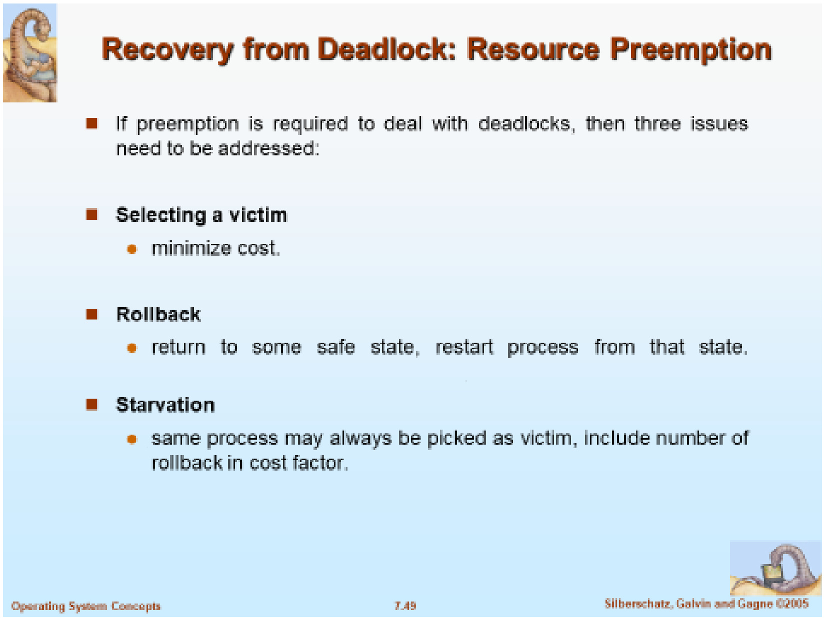
Safe Sequence ↓ (P2, P3, P4, P0, P1) or (P2, P0, P1, P3, P4)

So, Request from P2 can be granted

[5] 2 L3

<table>
<tr>
<td rowspan="2">5</td>
<td>a</td>
<td>
In a complex computer system with multiple interconnected processes, occasional resource conflicts may occur, leading to a situation where processes are unable to progress. Describe the measures that can be taken to resolve such conflicts and bring the system back to a stable state.



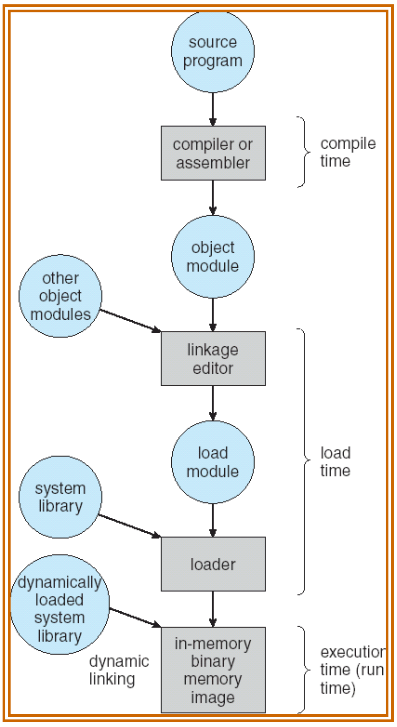
</td>
<td>[5]</td>
<td>2</td>
<td>L2</td>
</tr>
<tr>
<td>b</td>
<td>
Consider the process of preparing a program for execution, where instructions and data need to be associated with specific memory locations. Discuss the different stages with neat diagram, that can be employed to establish this association, ensuring efficient and reliable execution of the program.

Address binding of instructions and data to memory addresses can happen at three different stages

- **Compile time**: If memory location known a priori, *absolute code* can be generated; must recompile code if starting location changes
</td>
<td>[5]</td>
<td>2</td>
<td>L2</td>
</tr>
</table>

<table>
<tr><td></td><td></td><td>

- **Load time**: Must generate *relocatable code* if memory location is not known at compile time
- **Execution time**: Binding delayed until run time if the process can be moved during its execution from one memory segment to another. Need hardware support for address maps (e.g., *base* and *limit registers*).




</td><td></td><td></td><td></td></tr>
<tr><td>6</td><td>a</td><td>

Define the following.

(i) Logical Address and Physical Address

**Logical address** – generated by the CPU; also referred to as *virtual address*

**Physical address** – address seen by the memory unit

(ii) Hole

Hole – block of available memory; holes of various size are scattered throughout memory

(iii)First fit and Best fit

**First-fit**: Allocate the *first* hole that is big enough

**Best-fit**: Allocate the *smallest* hole that is big enough; must search entire list, unless ordered by size. Produces the smallest leftover hole.

(iv)External Fragmentation

**External Fragmentation** – total memory space exists to satisfy a request, but it is not contiguous

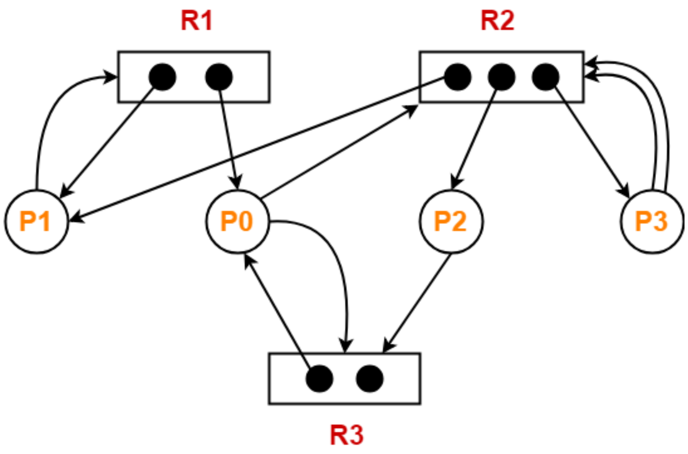
(v) Internal Fragmentation

**Internal Fragmentation** – allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used

</td><td>[5]</td><td>3</td><td>L2</td></tr>
</table>

b

Consider the resource allocation graph in the figure-




Find if the system is in a deadlock state otherwise find a safe sequence.

The given resource allocation graph is multi instance with a cycle contained in it.
So, the system may or may not be in a deadlock state.

Using the given resource allocation graph, we have-

| Process | **Allocation** | | | **Need** | | |
|---|---|---|---|---|---|---|
| | **R1** | **R2** | **R3** | **R1** | **R2** | |
| **P0** | 1 | 0 | 1 | 0 | 1 | |
| **P1** | 1 | 1 | 0 | 1 | 0 | |
| **P2** | 0 | 1 | 0 | 0 | 0 | |
| **P3** | 0 | 1 | 0 | 0 | 2 | |

Available = [ R1 R2 R3 ] = [ 0 0 1 ]

**<u>Step-01:</u>**

With the instances available currently, only the requirement of the process P2 can be satisfied.
So, process P2 is allocated the requested resources.
It completes its execution and then free up the instances of resources held by it.

[5] 2 L3

Then-

Available

= [ 0 0 1 ] + [ 0 1 0 ]

= [ 0 1 1 ]

**Step-02:**

- With the instances available currently, only the requirement of the process P0 can be satisfied.
- So, process P0 is allocated the requested resources.
- It completes its execution and then free up the instances of resources held by it.

Then-

Available

= [ 0 1 1 ] + [ 1 0 1 ]

= [ 1 1 2 ]

**Step-03:**

With the instances available currently, only the requirement of the process P1 can be satisfied.

- So, process P1 is allocated the requested resources.
- It completes its execution and then free up the instances of resources held by it.

Then-

Available

= [ 1 1 2 ] + [ 1 1 0 ]

= [ 2 2 2 ]

**Step-04:**

With the instances available currently, the requirement of the process P3 can be satisfied.

- So, process P3 is allocated the requested resources.
- It completes its execution and then free up the instances of resources held by it.

Then-

Available

= [ 2 2 2 ] + [ 0 1 0 ]

= [ 2 3 2 ]

Thus,

There exists a safe sequence P2, P0, P1, P3 in which all the processes can be executed.

So, the system is in a safe state.

**CI** **CCI** **HoD**

----------------------------------------------------------All the Best----------------------------------------------------------

| CO-PO Mapping | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Course Outcomes** | | **Modules covered** | **PO1** | **PO2** | **PO3** | **PO4** | **PO5** | **PO6** | **PO7** | **PO8** | **PO9** | **PO10** | **PO11** | **PO12** | **PSO1** | **PSO2** | **PSO3** | **PSO4** |
| CO1 | Describe the Operating System Structure and Services. | **1** | 3 | - | - | - | - | - | - | - | - | - | - | 3 | - | 2 | - | - |
| CO2 | Summarize the Process Management concepts like Processes, Threads, CPU Scheduling, Process Synchronization and Deadlocks | **1, 2** | 3 | 2 | 2 | - | - | - | - | - | - | - | - | 3 | - | 2 | - | - |
| CO3 | Interpret the Memory Management concepts with respect to Main Memory and Virtual Memory. | **3, 4** | 3 | 2 | 2 | - | - | - | - | - | - | - | - | 3 | - | 2 | - | - |
| CO4 | Discuss the Storage Management concepts like File-System Interface, File-System Implementation and Mass-Storage Structure | **4, 5** | 3 | 2 | 2 | - | - | - | - | - | - | - | - | 3 | - | 2 | - | - |
| CO5 | Elucidate the Protection features in Operating System and case study in Linux OS. | **5** | 3 | 2 | 2 | - | - | - | - | - | - | - | - | 3 | - | 2 | - | - |