INTERNAL ASSESSMENT TEST 2 – JUNE 2021
SOLUTION

Sub:	**OPERATING SYSTEMS**				Sub Code:	17CS64	Branch:	**ISE & CSE**
Date:	22-06-2021	Duration:	90 min's	Max Marks:	50	Sem / Sec:	VI	OBE

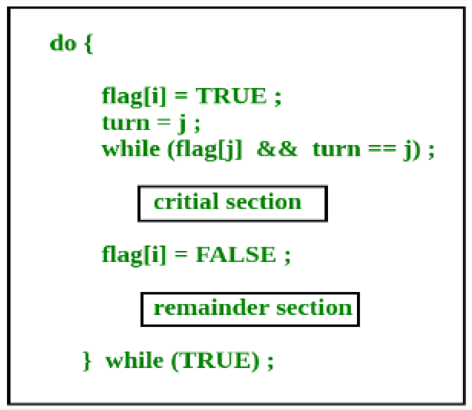
Answer any 5 Questions (5 X 10 = 50)

1 (a)	Illustrate with an example Peterson's solution for Critical section problem and prove that mutual exclusion property is preserved.	[06] [4+2]	CO 2	L2

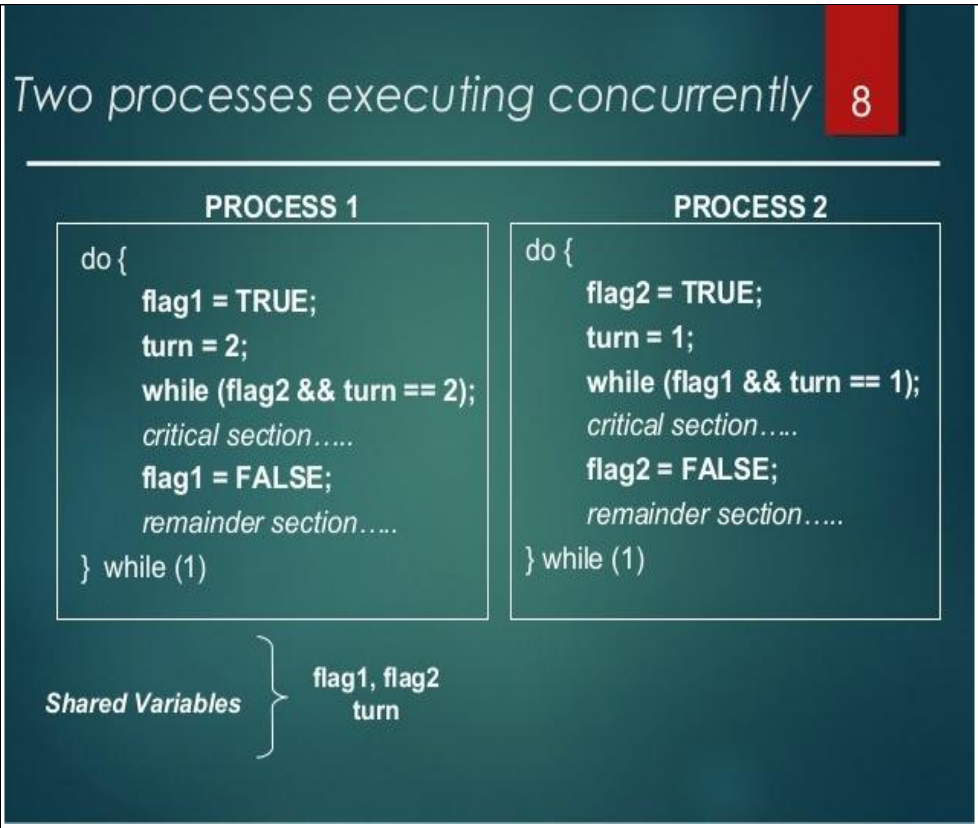
ANS:

In Peterson's solution, we have two shared variables:

- boolean flag[i] :Initialized to FALSE, initially no one is interested in entering the critical section
- int turn : The process whose turn is to enter the critical section.



```
do {

      flag[i] = TRUE ;
      turn = j ;
      while (flag[j]  &&  turn == j) ;

          critial section

      flag[i] = FALSE ;

          remainder section

   }  while (TRUE) ;
```

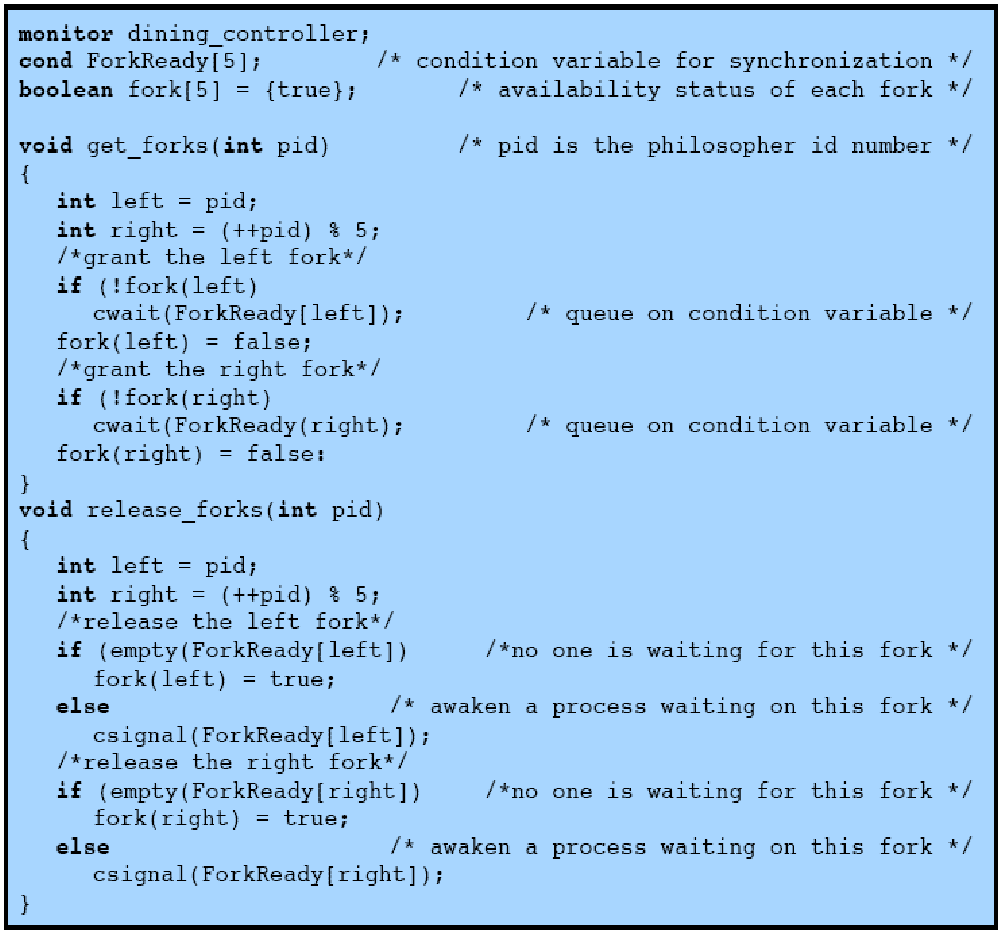
Two processes executing concurrently

```
PROCESS 1                              PROCESS 2
do {                                   do {
    flag1 = TRUE;                          flag2 = TRUE;
    turn = 2;                              turn = 1;
    while (flag2 && turn == 2);            while (flag1 && turn == 1);
    critical section…..                    critical section…..
    flag1 = FALSE;                         flag2 = FALSE;
    remainder section…..                   remainder section…..
} while (1)                            } while (1)
```

Shared Variables } flag1, flag2
turn

(b)	What is a Semaphore? What is the difference between Binary and counting Semaphore ANS: Semaphore is an integer value used for signalling among processes. Semaphore is simply a variable that is non-negative and shared between threads. This variable is used to solve the critical section problem and to achieve process synchronization in the multiprocessing environment. Difference between Binary and counting Semaphore 1. **Binary Semaphore –** This is also known as mutex lock. It can have only two values – 0 and 1. Its value is initialized to 1. It is used to implement the solution of critical section problems with multiple processes. 2. **Counting Semaphore –** Its value can range over an unrestricted domain. It is used to control access to a resource that has multiple instances.	[04] [02+02]	CO 2	L2
2 (a)	What is Race condition? Race condition is a situation where several processes access and manipulate shared data concurrently. The final value of the shared data depends upon which process finishes	[02]	CO 2	L1

	last.To prevent race conditions, concurrent processes must be synchronized.			
(b)	Apply Monitors to solve the Dining Philosophers problem and explain. ANS: Characteristics of a Monitor: ▪ Local data variables accessible only by Monitor Procedures ▪ Process enter Monitor by invoking one of its procedure ▪ Only one process may be executing in a Monitor at a time. Vector of 5 conditional variables defined, one per Fork Boolean vector records the availability of a Fork.(True means Fork is available) • Two procedures- 1. get_forks – seize his left and right fork 2. release_forks - make two forks available One philosopher process in Monitor at a Time	[08]	CO 2	L2



```
monitor dining_controller;
cond ForkReady[5];          /* condition variable for synchronization */
boolean fork[5] = {true};        /* availability status of each fork */

void get_forks(int pid)          /* pid is the philosopher id number */
{
   int left = pid;
   int right = (++pid) % 5;
   /*grant the left fork*/
   if (!fork(left)
      cwait(ForkReady[left]);          /* queue on condition variable */
   fork(left) = false;
   /*grant the right fork*/
   if (!fork(right)
      cwait(ForkReady(right);          /* queue on condition variable */
   fork(right) = false:
}
void release_forks(int pid)
{
   int left = pid;
   int right = (++pid) % 5;
   /*release the left fork*/
   if (empty(ForkReady[left])      /*no one is waiting for this fork */
      fork(left) = true;
   else                        /* awaken a process waiting on this fork */
      csignal(ForkReady[left]);
   /*release the right fork*/
   if (empty(ForkReady[right])     /*no one is waiting for this fork */
      fork(right) = true;
   else                        /* awaken a process waiting on this fork */
      csignal(ForkReady[right]);
}
```

3 (a)	What is Reader-Writer problem? Explain how Semaphore will give solution to the Reader-Writer problem.	[06] [2+4]	CO 2	L1

ANS:

The readers-writers problem is a classical problem of process synchronization, it relates to a data set such as a file that is shared between more than one process at a time. Among these various processes, some are Readers - which can only read the data set; they do not perform any updates, some are Writers - can both read and write in the data sets.

If two or more than two readers want to access the file at the same point in time there will be no problem. However, in other situations like when two writers or one reader and one writer wants to access the file at the same point of time, there may occur some problems, hence the task is to design the code in such a manner that if one reader is reading then no writer is allowed to update at the same point of time, similarly, if one writer is writing no reader is allowed to read the file at that point of time and if one writer is updating a file other writers should not be allowed to update the file at the same point of time. However, multiple readers can access the object at the same time.

Solution using Semaphore:

The code for the writer process looks like this:

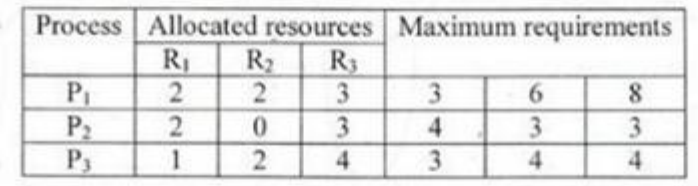
```
while(TRUE)
{
    wait(w);

   /* perform the write operation */

   signal(w);
}
```

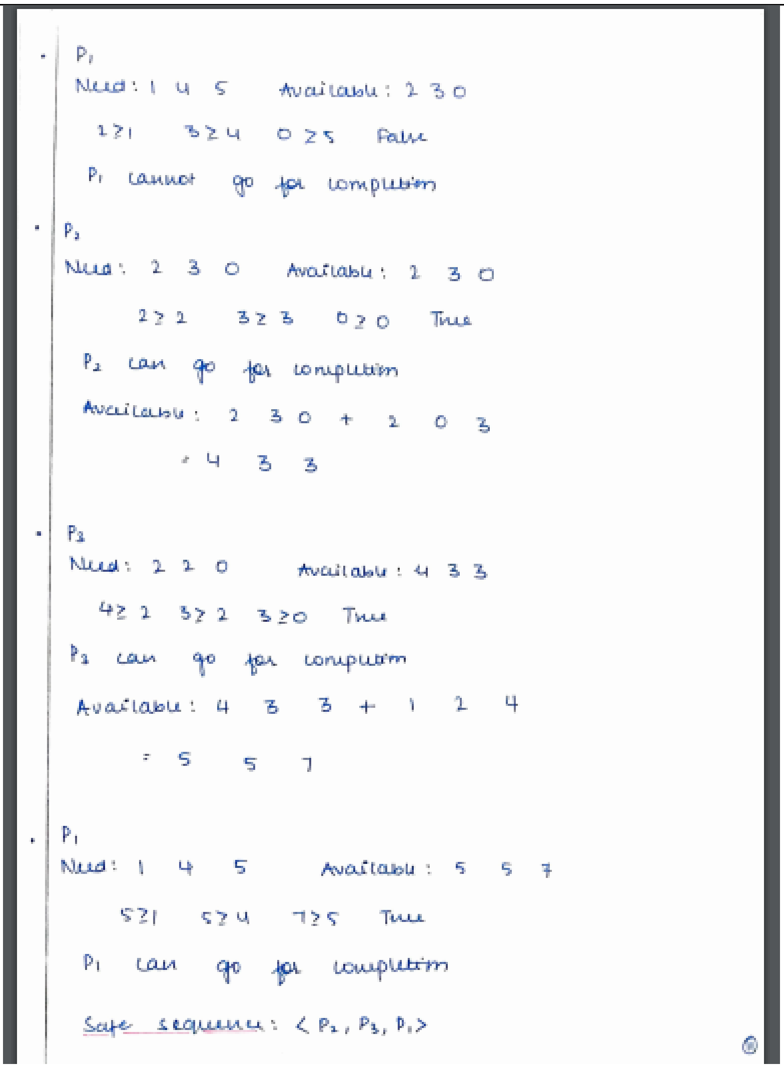
And, the code for the reader process looks like this:

```
while(TRUE)
{
    //acquire lock
    wait(m);
    read_count++;
    if(read_count == 1)
```

```
      wait(w);


   //release lock
   signal(m);


   /* perform the reading operation */


   // acquire lock
   wait(m);
   read_count--;
   if(read_count == 0)
      signal(w);


   // release lock
   signal(m);
}
```

- As seen above in the code for the writer, the writer just waits on the w semaphore until it gets a chance to write to the resource.
- After performing the write operation, it increments w so that the next writer can access the resource.
- On the other hand, in the code for the reader, the lock is acquired whenever the read_count is updated by a process.
- When a reader wants to access the resource, first it increments the read_count value, then accesses the resource and then decrements the read_count value.
- The semaphore w is used by the first reader which enters the critical section and the last reader which exits the critical section.
- The reason for this is, when the first readers enters the critical section, the writer is blocked from the resource. Only new readers can access the resource now.
- Similarly, when the last reader exits the critical section, it signals the writer using the w semaphore because there are zero readers now and a writer can have the

<table>
<tr><td></td><td>chance to access the resource.</td><td></td><td></td><td></td></tr>
<tr><td>(b)</td><td>Define Deadlock. Write short note on 4 necessary conditions that arise deadlocks
ANS:
A set of blocked processes each holding a resource and waiting to acquire a resource held by another process in the set is called Deadlock.
4 necessary conditions that arise deadlocks:
1. Mutual exclusion: Only one process may use a resource at a time
2. Hold-and-wait: A process may hold allocated resources while awaiting assignment of other resources
3. No preemption: No resource can be forcibly removed form a process holding it
4. Circular wait: A closed chain of processes exists, such that each process holds at least one resource needed by the next process in the chain</td><td>[04]
[1+3]</td><td>CO 2</td><td>L1</td></tr>
<tr><td>4.</td><td>For the following snapshot, find the safe sequence using Banker’s algorithm. The number of resource units available in R1, R2 and R3 are 7, 7, 10 respectively.

<table>
<tr><th>Process</th><th colspan="3">Allocated resources</th><th colspan="3">Maximum requirements</th></tr>
<tr><th></th><th>R₁</th><th>R₂</th><th>R₃</th><th></th><th></th><th></th></tr>
<tr><td>P₁</td><td>2</td><td>2</td><td>3</td><td>3</td><td>6</td><td>8</td></tr>
<tr><td>P₂</td><td>2</td><td>0</td><td>3</td><td>4</td><td>3</td><td>3</td></tr>
<tr><td>P₃</td><td>1</td><td>2</td><td>4</td><td>3</td><td>4</td><td>4</td></tr>
</table>
i) If there is a request from Process p2 for a unit of resource R3, can it be granted Immediately?
ii) If there is a request from Process p1 for a 2 units of resource R3, can it be granted Immediately?
Justify your answer.
ANS:</td><td>[10]
[6+2+2]</td><td>CO 2</td><td>L3</td></tr>
</table>

	 P1 Need: 1 4 5 Available: 2 3 0 $2 \geq 1$ $3 \geq 4$ $0 \geq 5$ False P1 cannot go for completion P2 Need: 2 3 0 Available: 2 3 0 $2 \geq 2$ $3 \geq 3$ $0 \geq 0$ True P2 can go for completion Available: 2 3 0 + 2 0 3 = 4 3 3 P3 Need: 2 2 0 Available: 4 3 3 $4 \geq 2$ $3 \geq 2$ $3 \geq 0$ True P3 can go for completion Available: 4 3 3 + 1 2 4 = 5 5 7 P1 Need: 1 4 5 Available: 5 5 7 $5 \geq 1$ $5 \geq 4$ $7 \geq 5$ True P1 can go for completion Safe sequence: <P2, P3, P1> i) If there is a request from Process p2 for a unit of resource R3, can it be granted Immediately? ANS: Cannot be granted as P2 is asking for a resource more than its claim. ii) If there is a request from Process p1 for a 2 units of resource R3, can it be granted Immediately? ANS: cannot be granted because the requested resource is not available now.			
5 (a)	Deadlock exits if a cycle exists. Yes or no? Justify your answer with a suitable example. ANS: YES	[06] [1+5]	CO2	L2

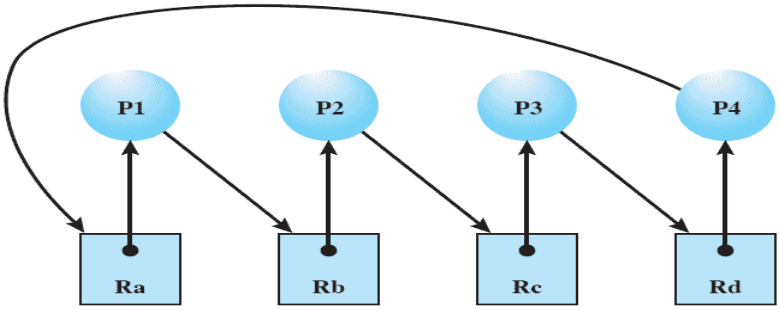


Figure 6.6 Resource Allocation Graph for Figure 6.1b

If a resource-allocation graph contains no cycles, then the system is not deadlocked.If a resource-allocation graph does contain cycles **AND** each resource category contains only a single instance, then a deadlock exists as shown in above diagram.

(b) Explain different methods to recover from Deadlocks.

[04]
[1x4]
CO2
L1

ANS:

1. Abort all deadlocked processes

   (commonly used by OS)

2. Back up each deadlocked process to some previously defined checkpoint, and restart all process:

   a)Original deadlock may occur

   b)Roll back and Restart mechanisms must be built in.

3. Successively abort deadlocked processes until deadlock no longer exists

4. Successively preempt resources until deadlock no longer exists

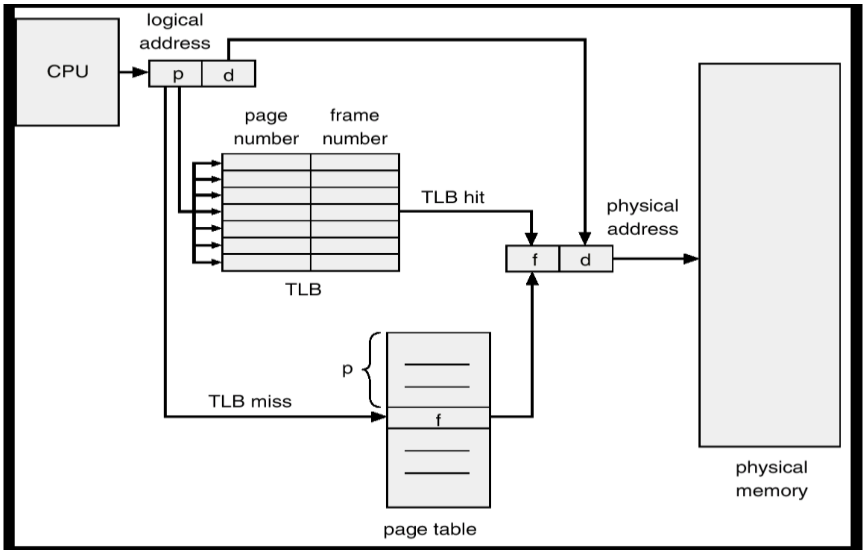
6 (a) Given memory partitions: 40K, 90K, 120K , 20K, 200K, 160K

Apply Best fit, First fit and Worst fit algorithms to place 35K, 10K, 120K, 80K.

[04]
[1x4]
CO2
L3

ANS:

	Best	First	Worst
35K	40K	40K	200K
10K	20K	40K	200K
120K	120K	120K	200K
80K	90K	90K	200K

(b)	What is Paging? Explain how Address translation happens in paging using translation look-aside buffers. ANS: Paging is Partition memory into small equal fixed-size chunks and divide each process into the same size chunks. • The chunks of a process are called pages • The chunks of memory are called frames  To overcome this problem a high-speed cache is set up for page table entries called a Translation Lookaside Buffer (TLB). Translation Lookaside Buffer (TLB) is nothing but a special cache used to keep track of recently used transactions. TLB contains page table entries that have been most recently used. Given a virtual address, the processor examines the TLB if a page table entry is present (TLB hit), the frame number is retrieved and the real address is formed. If a page table entry is not found in the TLB (TLB miss), the page number is used as index while processing page table. TLB first checks if the page is already in main memory, if not in main memory a page fault is issued then the TLB is updated to include the new page entry.	[06] [2+4]	CO2	L1