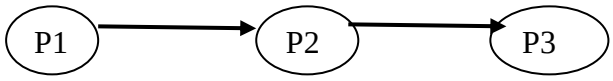
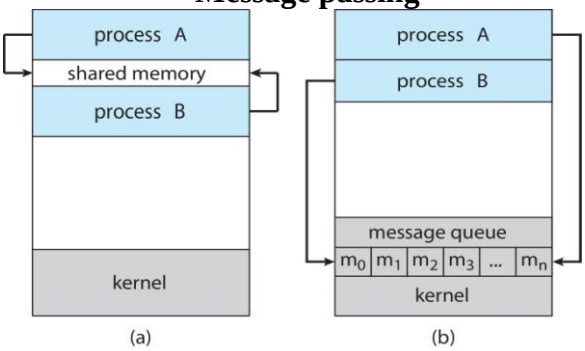
KONGU ENGINEERING COLLEGE, PERUNDURAI 638 060

CONTINUOUS ASSESSMENT TEST – 2

Regulations 2020

Month and Year : Oct 2023	Roll Number :
Programme : B.Tech Branch : IT Semester : V	Date : 7.10.23 Time : 9.15am to 10.45am
Course Code : 20ITT52 Course Name : Operating systems	Duration : 1 ½ Hours Max. Marks : 50

PART - A (10 × 2 = 20 Marks)		
ANSWER ALL THE QUESTIONS		
1.	A multithreaded system consisting of multiple user-level threads mapped to one kernel thread cannot make use of the different processors in a multiprocessor system. Consequently, there is no performance benefit associated with this solution. The multithreaded solution could be faster if the multiple user-level threads are mapped to different kernel threads.	[CO2,K2]
2.	Many-to-One, One-to-One, Many-to-Many and Two level model	[CO2,K1]
3.	▪ Each process has **critical section** segment of code • Process may be changing common variables, updating table, writing file, etc. • When one process in critical section, no other may be in its critical section ▪ The ***critical-section problem*** is **to design a protocol that the processes can use to synchronize their activity so as to cooperatively share data.** ▪ Each process must ask **permission to enter critical section in entry section[**The section of code implementing this request**]**, may follow critical section with **exit section**, then **remainder section** [remaining code]	[CO3,K2]
4.	Minimum value (X) of D will possible when, P2 reads D=100, preempted. P1 executes D=D+20, D=120. P3 executes D=D+10, D=130. Now, P2 has D=100, executes, D = D-50 = 100-50 = 50. P2 writes D=50 final value. **So, minimum value (X) of D is 50.** Maximum value (Y) of D will possible when, P1 reads D=100, preempted. P2 reads D=100, executes, D = D-50 = 100-50 = 50. Now, P1 executes, D = D+20 = 100+20 = 120. And now, P3 reads D=120, executes D=D+10, D=130. P3 writes D=130 final value. **So, maximum value (Y) of D is 130. Therefore, Y - X = 130 - 50 = 80**	[CO3,K3]
5.	No memory barrier instruction x=0 memory barrier instruction x=100	[CO3,K3]
6.	Mutual exclusion but not progress	[CO3,K3]
7.	20-7 -> 13 will be in blocked state, when perform 12 V(S) operation makes 12 more process to get chance for execution from blocked state. So one process will be left in the queue (blocked state).	
8.	There is possibility for executing in any one of the 4 ways. 1. execute P2 process after P1 process 2.execute P1 process after P2 process 3. Pre-empting P1 and executing P2 processes 4. Pre-empting P2 and executing P1 processes 1. Execute P2 process after P1 process, then B = 3 2. Execute P1 process after P2 process, then B = 4 3. Pre-empting P1 and executing P2 processes B=3 4. Pre-empting P2 and executing P1 processes B=2 So, total no. of distinct values that B can possibly take after the execution is 3.	[CO3,K3]
9.		[CO3,K3]

10.	Mutual exclusion locks (mutexes) can prevent data inconsistencies due to race conditions. A race condition often occurs when two or more threads need to perform operations on the same memory area, but the result of computations depends on the order in which these operations are performed.		[CO3,K3]
	PART – B (3 × 10 = 30 Marks) ANSWER ANY THREE QUESTIONS		
11.	Illustrate two fundamental models of interprocess communication. • Processes within a system may be ***independent*** or ***cooperating*** • Cooperating process can affect or be affected by other processes, including sharing data • Reasons for cooperating processes: – Information sharing – Computation speedup – Modularity – Convenience • Cooperating processes need **interprocess communication** (**IPC**) • Two models of IPC – **Shared memory** – **Message passing**  **IPC – Shared Memory** • An **area of memory shared among the processes that wish to communicate** • The **communication is under the control of the users processes** not the operating system. • Major issues is **to provide mechanism that will allow the user processes to synchronize their actions** when they access shared memory. **IPC – Message Passing** • Processes communicate with each other without resorting to shared variables • **IPC facility provides two operations:** – **send(*message*) -receive(*message*)** • The *message* size is either fixed or variable ➢ If processes *P* and *Q* wish to communicate, they need to: ❖ Establish a ***communication link*** between them ❖ Exchange messages via send/receive **Implementation issues:** ✓ How are links established? ✓ Can a link be associated with more than two processes? ✓ How many links can there be between every pair of communicating processes? ✓ What is the capacity of a link? ✓ Is the size of a message that the link can accommodate fixed or variable? ✓ Is a link unidirectional or bi-directional?	(10)	[CO2,K2]

12.	a. The values of Need for processes P0 through P4, respectively, are (0,0, 0, 0), (0, 7, 5, 0), (1, 0, 0, 2), (0, 0, 2, 0), and (0, 6, 4, 2). b. The system is in a safe state. With Available equal to (1, 5, 2, 0), either process P0 or P3 could run. Once process P3 runs, it releases its resources, which allows all other existing processes to run. c. The request can be granted immediately. The value of Available is then (1, 1, 0, 0). One ordering of processes that can finish is P0, P2, P3, P1, and P4.	(10)	[CO3,K3]
13	Software Solutions **Algorithm for Process P_i** **while (true)** **{** **flag[i] = true;** **turn = j;** **while (flag[j] && turn == j) ;** **/* critical section */** **flag[i] = false;** **/*remainder section */** **}** **Algorithm for Process P_j** **while (true)** **{** **flag[j] = true;** **turn = i;** **while (flag[i] && turn == i) ;** **/* critical section */** **flag[j] = false;** **/*remainder section */** **}** It is a humble algorithm to give chance (ie)set turn value to other process **Provable that the three CS requirement are met:** **1. Mutual exclusion is preserved, 2. Progress requirement is satisfied 3. Bounded-waiting requirement is met** **Semaphore** **Semaphore S is an integer variable that, apart from initialization, is** accessed only through two standard atomic operations: **wait() and signal().** Semaphores were introduced by the Dutch computer scientist Dijkstra, such that , the **wait() operation was originally termed P (from the Dutch *proberen, "to test");*** ***signal() was originally called V (from verhogen, "to increment").*** **Definition** of the **wait() operation** **wait(S) {** **while (S <= 0) ;** **// busy wait //** **S--;** **}** **In semaphore, 0- hold and 1 - free** **Definition** of the **signal() operation** **signal(S) {** **S++;** **}**	(10)	[CO3,K1]

Hardware Instructions: The three hardware instructions that provide support for solving the critical-section problem are

1. Memory Barriers
2. Hardware instructions [Test and Set , Compare And Swap(CAS)]
3. Atomic variables

Memory Barriers:

- **Memory model** are the memory guarantees a computer architecture makes to application programs.
- Memory models may be either:
  - **Strongly ordered** – where a memory modification of one processor is immediately visible to all other processors.
  - **Weakly ordered** – where a memory modification of one processor may not be immediately visible to all other processors.
- A **memory barrier is an instruction** that forces any change in memory to be propagated (made visible) to all other processors.

Hardware instructions [Test and Set]

locked(lock=1)
Not locked (lock=0)

Definition of Test and set

```
boolean test_and_set (boolean *target)
    {
        boolean rv = *target;
        *target = true;
        return rv:
    }
```

```
do {
    while (test_and_set(&lock)) ;
        /* do nothing */

            /* critical section */

        lock = false;
        /* remainder section */
    } while (true);
```

P2 →

← P1

```
do {
    while (test_and_set(&lock)) ;
        /* do nothing */

            /* critical section */

        lock = false;
        /* remainder section */
    } while (true);
```

Does it solve the critical-section problem properties?

1. Mutual exclusion is achieved 2. No bounded waiting

P3 P4

locked(lock=1)
Not locked (lock=0)

```
int compare_and_swap(int *value, int expected, int new_value)
    {
     int temp = *value;
      if (*value == expected)
          *value = new_value;
      return temp;
    }
```

P1

```
while (true)
{
 while (compare_and_swap(&lock, 0, 1) != 0) ;

        /* do nothing */

            /* critical section */

            lock = 0;

        /* remainder section */
 }
```

P2

```
while (true)
{
 while (compare_and_swap(&lock, 0, 1) != 0);

        /* do nothing */

            /* critical section */

            lock = 0;

        /* remainder section */
 }
```

14	**Resource**	**Available**	**t=0**	**t=1**	**t=2**	**t=3**	**t=4**	**t=5**	**t=6**	**t=7**	**t=8**	**t=9**	**t=10**		(10)	[CO3,K3]
	R1	**3**	**3**	**3**	**1**	**1**	**0**	**1**	**1**	**1**	**2**	**2**	**3**			
	R2	**2**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**2**			
	R3	**3**	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**2**	**2**	**3**	**3**			
	R4	**2**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**2**	**2**			

Bloom's Taxonomy Level	Remembering (K1)	Understanding (K2)	Applying (K3)	Analysing (K4)	Evaluating (K5)	Creating (K6)
Percentage	20	20	60	-	-	-