**Operating System (OS) 2nd Internal**

Answer any five questions. Each carries two marks. 2X5-10

1. List three requirements that a solution to critical section problem must satisfy.

Ans) . Mutual Exclusion - If process Pi 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 grante

1. What is semaphore?

Ans) Synchronization tool that does not require busy waiting • Semaphore S – integer variable • Two standard operations modify S: wait() and signal() – Originally called P() and V() • Less complicated • Can only be accessed via two indivisible (atomic) operations

wait (S)

{

while (S <= 0) ; // busy wait

S--;

}

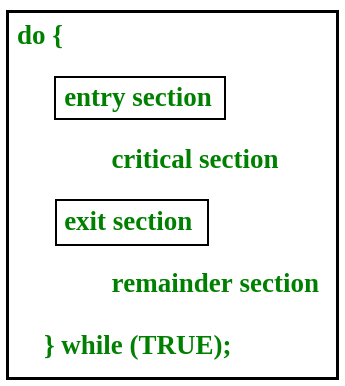
signal (S

) { S++; }

1. List 3 classical problems of synchronization.

The classical problems of synchronization are as follows:

1. Bound-Buffer problem
2. Sleeping barber problem
3. Dining Philosophers problem
4. Readers and writers problem
5. Write code for Critical Section.



1. List advantages of Priority Scheduling algorithm.

* Easy to use scheduling method
* Processes are executed on the basis of priority so high priority does not need to wait for long which saves time
* This method provides a good mechanism where the relative important of each process may be precisely defined.
* Suitable for applications with fluctuating time and resource requirements.

1. List disadvantages of Round Robin scheduling algorithm.

**Disadvantages of Round Robin Algorithm**

* Low slicing time reduces processor output.
* Spends more time on context switching.
* Performance depends on time quantum.
* Processes don't have priorities.

II. Answer any four questions. Each carries five marks. 5X4-20

1. Explain producer consumer problem & Solution Using semaphore

**Solution for Producer –**

do{

//produce an item

wait(empty);

wait(mutex);

//place in buffer

signal(mutex);

signal(full);

}while(true)

When producer produces an item then the value of “empty” is reduced by 1 because one slot will be filled now. The value of mutex is also reduced to prevent consumer to access the buffer. Now, the producer has placed the item and thus the value of “full” is increased by 1. The value of mutex is also increased by 1 because the task of producer has been completed and consumer can access the buffer.

**Solution for Consumer**

do{

wait(full);

wait(mutex);

// remove item from buffer

signal(mutex);

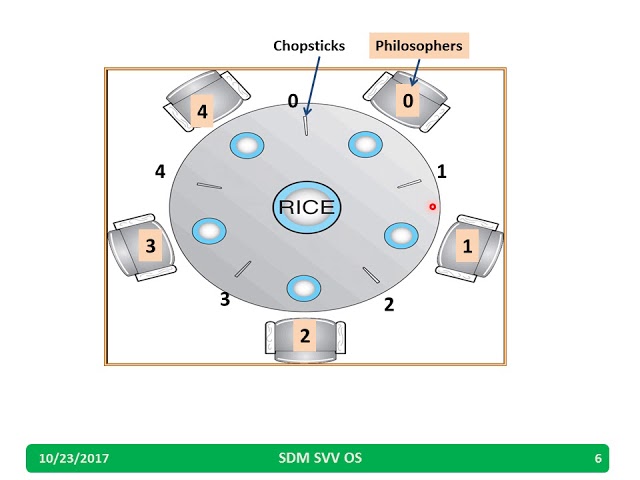
signal(empty);

// consumes item

}while(true)

As the consumer is removing an item from buffer, therefore the value of “full” is reduced by 1 and the value is mutex is also reduced so that the producer cannot access the buffer at this moment. Now, the consumer has consumed the item, thus increasing the value of “empty” by 1. The value of mutex is also increased so that producer can access the buffer now.

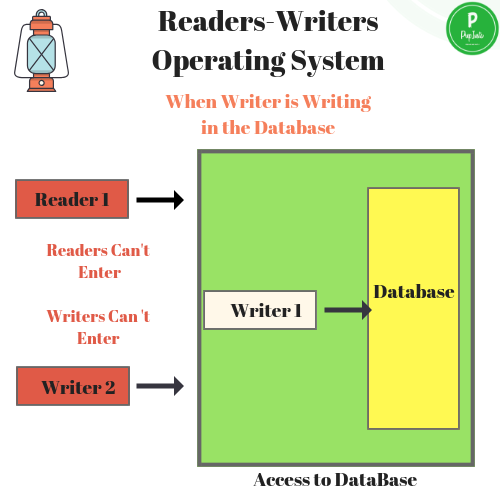
1. Explain dinning philosopher & reader- writer Problems with diagram.



The dining philosophers problem states that there are 5 philosophers sharing a circular table and they eat and think alternatively. There is a bowl of rice for each of the philosophers and 5 chopsticks. A philosopher needs both their right and left chopstick to eat. A hungry philosopher may only eat if there are both chopsticks available.Otherwise a philosopher puts down their chopstick and begin thinking again.

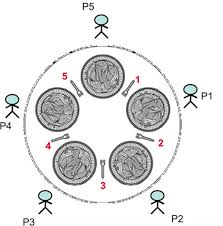
## Solution of Dining Philosophers Problem

A solution of the Dining Philosophers Problem is to use a semaphore to represent a chopstick. A chopstick can be picked up by executing a wait operation on the semaphore and released by executing a signal semaphore



* A data set is shared among a number of concurrent processes
  + Readers – only read the data set; they do ***not*** perform any updates
  + Writers – can both read and write
* Problem – allow multiple readers to read at the same time
  + Only one single writer can access the shared data at the same time
  + ***First***variation – no reader kept waiting unless writer has permission to use shared object
  + ***Second***variation – once writer is ready, it performs write ASAP
  + Both may have starvation leading to even more variations
  + Problem is solved on some systems by kernel providing reader-writer locks

1. Write solution to dining philosopher problem using monitor.(NS)



**Monitor-based Solution to Dining Philosophers**

We illustrate monitor concepts by presenting a deadlock-free solution to the dining-philosophers problem. Monitor is used to control access to state variables and condition variables. It only tells when to enter and exit the segment. This solution imposes the restriction that a philosopher may pick up her chopsticks only if both of them are available.

To code this solution, we need to distinguish among three states in which we may find a philosopher. For this purpose, we introduce the following data structure:

**EATING –** When philosopher has got both the forks, i.e., he has entered the section

**THINKING –** When philosopher doesn’t want to gain access to either fork.

**HUNGRY –** When philosopher wants to enter the critical section.