Explain critical section problem

Ans-

The **critical section problem** is used to design a protocol followed by a group of processes, so that when one process has entered its critical section, no other process is allowed to execute in its critical section.

The **critical section** refers to the segment of code where processes access shared resources, such as common variables and files, and perform write operations on them.

Since processes execute concurrently, any process can be interrupted mid-execution. In the case of shared resources, partial execution of processes can lead to data inconsistencies. When two processes access and manipulate the shared resource concurrently, and the resulting execution outcome depends on the order in which processes access the resource; this is called a race condition.

**Race conditions** lead to inconsistent states of data. Therefore, we need a synchronization protocol that allows processes to cooperate while manipulating shared resources, which essentially is the critical section problem.

Graphical user interface, text, application

Description automatically generated

**Solutions to the critical section problem**

Any solution to the critical section problem must satisfy the following requirements:

* **Mutual exclusion**: When one process is executing in its critical section, no other process is allowed to execute in its critical section.
* **Progress**: When no process is executing in its critical section, and there exists a process that wishes to enter its critical section, it should not have to wait indefinitely to enter it.
* **Bounded waiting**: There must be a bound on the number of times a process is allowed to execute in its critical section, after another process has requested to enter its critical section and before that request is accepted.

Graphical user interface, text, application, email

Description automatically generated

Explain producer consumer problem

Ans –

The **Producer-Consumer problem** is a classic synchronization problem in operating systems.

The problem is defined as follows: there is a fixed-size buffer and a Producer process, and a Consumer process.

The **Producer** process creates an item and adds it to the shared buffer. The **Consumer** process takes items out of the shared buffer and “consumes” them.

Certain conditions must be met by the Producer and the Consumer processes to have consistent data synchronization:

1. The Producer process must not produce an item if the shared buffer is full.
2. The Consumer process must not consume an item if the shared buffer is empty.
3. Access to the shared buffer must be **mutually exclusive**; this means that at any given instance, only one process should be able to access the shared buffer and make changes to it.

**Solution**

The solution to the Producer-Consumer problem involves three *semaphore* variables.

* **semaphore Full**: Tracks the space filled by the Producer process. It is initialized with a value of 00 as the buffer will have 00 filled spaces at the beginning
* **semaphore Empty**: Tracks the empty space in the buffer. It is initially set to **buffer\_size** as the whole buffer is empty at the beginning.
* **semaphore mutex**: Used for mutual exclusion so that only one process can access the shared buffer at a time.

Explain Bekary Algorithm.

Ans -

The Bakery Algorithm is a simple solution for the critical section problem. To recall, the critical section is a section of the code that can be accessed by more than one process. If the critical section is accessed and changed by two or more processes at the same time, this would lead to inconsistency in the data as both processes will try to change the same data at the same time.

Hence, this is a very important concern when dealing with two or more processes that can access a section of the code. The Bakery Algorithm provides a way to make sure that only one process can access the critical section at a time to ensure consistency. The Bakery Algorithm provides a general solution to the critical section problem, which means it provides a solution when there are N processes that can access the critical section.

Advantages of Lamport's bakery algorithm

Lamport's bakery algorithms are free from starvation.

Lamport's Bakery algorithm follows a FIFO.

Lamport's Bakery algorithm works with atomic registers.

Lamport's Bakery algorithm is one of the simplest known solutions to the mutual exclusion problem for the general case of the N process.

This algorithm ensures the efficient use of shared resources in a multithreaded environment.

Disadvantages of Lamport's bakery algorithm

Lamport's bakery algorithm is unreliable as a failure of any one of the processes will halt progress. It has a high message complexity of 3(N - 1) messages per entry/exit into the critical section.

What is deadlock? Explain deadlock prevention and deadlock avoidance.

Ans -

What is Deadlock Prevention

Deadlock prevention is the mechanism to ensure that at least one of the necessary conditions for deadlock can never occur. Usually, it is impossible to implement deadlock prevention. To prevent a deadlock situation, it is not necessary to know the number of all existing resources, availability and requests. Non-blocking synchronization algorithms and serializing tokens are some deadlock prevention algorithms. Furthermore, the resource allocation for deadlock prevention is conservative.

What is Deadlock Avoidance

A system is safe when it is possible to allocate resources to all processes in some order without causing a deadlock. Deadlock is the mechanism to ensure that the system does not enter an unsafe state. In other words, it ensures that the system does not have a deadlock. To avoid a system from reaching an unsafe state, the system should know about the number of existing resources, availability and requests.

### **Algorithms**

Non-blocking synchronization algorithms and serializing tokens are some deadlock prevention algorithms while Banker’s algorithm is the most common deadlock avoidance algorithm

### **Conclusion**

In brief, deadlock is a situation that occurs due to a set of processes in which each process holds a resource and waits to acquire a resource held by another process in the set. Deadlock prevention and deadlock avoidance are two mechanisms related to deadlocks. The main difference between deadlock prevention and deadlock avoidance is that the deadlock prevention ensures that at least one of the necessary conditions to cause a deadlock will never occur, while deadlock avoidance ensures that the system will not enter an unsafe state.