Task 1

1. Explain what process synchronization is.

Process Synchronization is the coordination of execution of multiple processes in a multi-process system to ensure that they access shared resources in a controlled and predictable manner. It aims to resolve the problem of race conditions and other synchronization issues in a concurrent system.

2. Explain the classic problems of process synchronization and Compare between the alternative approaches in critical section solutions.

- These problems are used for testing nearly every newly proposed synchronization scheme. The following problems of synchronization are considered as classical problems:

    - **Bounded-buffer (or Producer-Consumer) Problem**
      The problem describes two processes, the producer and the consumer, which share a common, fixed-size buffer used as a queue. The producer's job is to generate data, put it into the buffer, and start again. At the same time, the consumer is consuming the data (i.e. removing it from the buffer), one piece at a time.

    - **Dining-Philosophers Problem**
      The Dining Philosopher Problem states that K philosophers are seated around a circular table with one chopstick between each pair of philosophers. There is one chopstick between each philosopher. A philosopher may eat if he can pick up the two chopsticks adjacent to him. One chopstick may be picked up by any one of its adjacent followers but not both.

- **Readers-Writers Problem**
  The Dining Philosopher Problem states that K philosophers are seated around a circular table with one chopstick between each pair of philosophers. There is one chopstick between each philosopher. A philosopher may eat if he can pick up the two chopsticks adjacent to him. One chopstick may be picked up by any one of its adjacent followers but not both.

- **Sleeping Barber Problem**
  Barber shop with one barber, one barber chair and N chairs to wait in. When no customers the barber goes to sleep in barber chair and must be woken when a customer comes in. When barber is cutting hair new customers take empty seats to wait, or leave if no vacancy. This is basically the Sleeping Barber Problem.

- **In the Bakery Algorithm**, each process is assigned a number (a ticket) in a lexicographical order. Before entering the critical section, a process receives a ticket number, and the process with the smallest ticket number enters the critical section. If two processes receive the same ticket number, the process with the lower process ID is given priority.

- Critical section solutions

  Semaphore

  Concept: A generalization of a mutex, a semaphore maintains a count, allowing a specified number of threads to access the critical section simultaneously.

  Pros:

  More flexible than mutex for handling multiple resources.
  Can be used for signaling between threads.

  Cons:

  Requires careful handling to avoid deadlocks.
  More complex than mutex.

Mutex (Mutual Exclusion)

Pros:

Generalization: Mutexes can be used for synchronization among multiple processes or threads.

Efficiency: Some modern implementations use efficient techniques like spinlocks to minimize overhead.

Cons:

Complexity: Implementations might be more complex than Peterson's Solution.

Potential Deadlocks: Care must be taken to avoid deadlocks.

Peterson's Solution

Concept: rests on the assumption that the instructions are executed in a particular order and memory accesses can be achieved atomically.

Pros:

Relatively simple, making it easy to understand and implement.

The algorithm provides a fair solution, ensuring that both processes get an opportunity to enter the critical section.

Cons:

One significant limitation of Peterson's Solution is that it is designed for exactly two processes.

The algorithm uses busy waiting, where a process repeatedly checks a condition in a loop until it can proceed.

Monitors:

Concept: Combines data with procedures (methods) operating on that data, enforcing mutual exclusion by automatically locking and unlocking the monitor for each method call.

Pros:

Simplifies synchronization by encapsulating shared data and methods.

Reduces the risk of deadlock.

Cons:

Less flexible than other approaches.

3. Define critical section, Mutual Locks, Semaphores.

- **Critical section** refers to a segment of code that is executed by multiple concurrent threads or processes, and which accesses shared resources. These resources may include shared memory, files, or other system resources that can only be accessed by one thread or process at a time to avoid data inconsistency or race conditions.

- **Mutual exclusion locks (mutex)** is a program object that prevents multiple threads from accessing the same shared resource simultaneously. A shared resource in this context is a code element with a critical section, the part of the code that should not be executed by more than one thread at a time.

- **Semaphore** generalization of a mutex used to control access to a common resource by multiple threads and avoid critical section problems in a concurrent system such as a multitasking operating system.

Task 2:

4. Explain Deadlock characteristics. And Explain the Deadlock Detection and Avoidance process.

- **Deadlock characteristics**
  i. **Mutual exclusion**. Only one process at a time can use the resource. If another process requests that resource, the requesting process must be delayed until the resource has been released.
  ii. **Hold and wait**. A process must be holding at least one resource and waiting to acquire additional resources that are currently being held by other processes.
  iii. **No preemption.** Resources cannot be preempted; that is, a resource can be released only voluntarily by the process holding it, after that process has completed its task.
  iv. **Circular wait.** A set {P0, P1, ..., Pn} of waiting processes must exist such that P0 is waiting for a resource held by P1, P1 is waiting for a resource held by P2, ..., Pn−1 is waiting for a resource held by Pn, and Pn is waiting for a resource held by P0.

- **Deadlock detection**
  The deadlock detection is a technique used in operating systems to identify the presence of deadlocks in a system and take corrective actions to resolve them.

- **Avoidance process**
  The deadlock avoidance algorithm dynamically examines the resource allocation state to ensure that there can never be a circular wait condition

5. Explain the recovery from deadlocks methods.

Process Termination: To eliminate deadlocks by aborting a process, we use one of two methods. In both methods, the system reclaims all resources allocated to the terminated processes. The first method is Abort all deadlocked processes, the second one is Abort one process at time until the deadlock cycle is eliminated.

Resource Preemption: To eliminate deadlocks using resource preemption, we successively preempt some resources from processes and give these resources to other processes until the deadlock cycle is broken. If preemption is required to deal with deadlocks, then three issues need to be addressed like Selecting a victim, Rollback and Starvation

6. Explain the low level implementation of memory management. Explain the concepts of page table and swapping

- Low-level implementation of memory management involves techniques such as paging, page tables, and swapping to efficiently manage the use of memory in a computer system.

- **Page Table** is a data structure used by the virtual memory system to store the mapping between logical addresses and physical addresses. Logical addresses are generated by the CPU for the pages of the processes therefore they are generally used by the processes.

- **Swapping** is the process of bringing a process into memory and then temporarily copying it to the disc after it has run for a while. The purpose of swapping in an operating system is to access data on a hard disc and move it to RAM so that application programs can use it.

7. Explain segmentation and paging process. Explain the memory allocation methods.

- **Segmentation** is a memory-management scheme that supports this programmer view of memory. A logical address space is a collection of segments. Each segment has a name and a length. The addresses specify both the segment name and the offset within the segment. The programmer therefore specifies each address by two quantities: a segment name and an offset.

- **Paging** is another memory-management scheme that offers this advantage. However, paging avoids external fragmentation and the need for compaction, whereas segmentation does not. It also solves the considerable problem of fitting memory chunks of varying sizes onto the backing store.

- **A. Contiguous Memory Allocation:** The memory is usually divided into two partitions: one for the resident operating system and one for the user processes. We can place the operating system in either low memory or high memory. The major factor affecting this decision is the location of the interrupt vector. Since the interrupt vector is often in low memory, programmers usually place the operating system in low memory as well. Thus, in this text, we discuss only the situation in which the operating system resides in low memory. The development of the other situation is similar.

  B. Non-contiguous Memory Allocation: on the other hand, is a technique where the operating system allocates memory to a process in non-contiguous blocks. The blocks of memory allocated to the process need not be contiguous, and the operating system keeps track of the various blocks allocated to the process. Non-contiguous memory allocation is suitable for larger memory sizes and where efficient use of memory is important.