PROCESS SYNCHRONIZATION

UNIT-III

Process Synchronization-

Process synchronization involves using tools that control access to

shared data to avoid race conditions.

- These tools must be used carefully,as their incorrect use can result in poor system performance, including deadlock.

Process Synchronization-

When multiple processes execute concurrently sharing system resources, then inconsistent results might be produced.

- Process Synchronization is a mechanism that deals with the synchronization of processes.
- It controls the execution of processes running concurrently to ensure that consistent results are produced.

Critical Section-

-

Critical section is a section of the program where a process access the shared resources during its execution.

Example-

-

The following illustration shows how inconsistent results may be produced if multiple processes execute concurrently without any synchronization.

-

It is clear from here that inconsistent results may be produced if multiple processes execute concurrently without any synchronization.

Race Condition-

-

Race condition is a situation where-

- The final output produced depends on the execution order of instructions of different processes.
- Several processes compete with each other.
- To guard against the race condition above, we need to ensure that only one process at a time can be manipulating the variable count .

Critical Section

-

Each process has a segment of code called critical section.

- It is that part of code where process may be changing common variables,updating table, writing a file etc.
- Critical section problem means designing a protocol that the processes can use to cooperate.

Critical Section Problem-

-

If multiple processes access the critical

section concurrently, then results produced might be inconsistent.

- This problem is called as critical section problem

Synchronization Mechanisms-

-

Synchronization mechanisms allow the processes to access critical section in a synchronized manner to avoid the inconsistent results.

- For every critical section in the program, a synchronization

mechanism adds-

- **An entry section** before the critical section
- **An exit section** after the critical section

Synchronization Mechanisms-

General structure of a typical process.

Entry Section-

- It acts as a gateway for a process to enter inside the critical section.
- It ensures that only one process is present inside the critical section at any time.
- It does not allow any other process to enter inside the critical section if one process is already present inside it.

Exit Section

- It acts as an exit gate for a process to leave the critical section.
- When a process takes exit from the critical section, some changes are made so that other processes can enter inside the critical section.

Criteria For

Synchronization Mechanisms-

-

A solution to the critical-section problem must satisfy the following three requirements:

1. Mutual Exclusion-

-

If process P is executing in its critical section, then no other processes can be executing in their critical sections.

- Only one process is present inside the critical section at any time.
- No other process can enter the critical section until the process already present inside it completes.

2. Progress-

-

If no process is executing in its critical section and some processes wish to enter their critical sections, then only

those processes that are not executing in their remainder sections can participate in deciding which will enter its critical section next, and this selection cannot be postponed indefinitely

3. Bounded Wait-

-

There exists a bound, or limit, 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 granted.

Peterson’s Solution to critical section problem

Peterson’s solution is restricted to two processes that alternate execution between their critical sections and remainder sections.

- The processes are numbered P 0 and P 1.
- P_i represents a process then P_j represents another process.

Data items shared between the two processes.

- turn indicates whose turn it is to enter its critical section.
- If Pi == i, Pi is allowed to execute in its critical section..

-

Flag-----> is an array indicates if a process is ready to enter its critical section.

- If flag[i] == True,then Pi can enter its critical section.

We have to prove that,

- Mutual exclusion is preserved.
- The progress requirement is satisfied.
- The bounded waiting requirement is met.

Example for race condition

Producer -Consumer Problem.

Producer process

Consumer process

Race condition

Synchronization Hardware

-

Hardware features can make any programming task easier and improve system efficiency .

- Many modern computer systems provide special hardware instructions.
- We can use these instructions to solve the critical section probelm.

- If the machine supports **TestandSet()** instruction,we can implement mutual exclusion by declaring boolean variable **lock,** initialized to **false.**

Solution to Critical-section Problem Using Locks

Explanation-

-

The occurrence of the following scenes may lead to two processes present inside the critical section at the same time.

The characteristics of this synchronization mechanism are-

Test and Set Lock –

- Test and Set Lock (TSL) is a synchronization mechanism.
- It uses a test and set instruction to provide the synchronization among the processes executing concurrently.

test_and_set Instruction

Solution using test_and_set()

Working procedure

This synchronization mechanism guarantees mutual exclusion.

Mutual-exclusion implementation with the Swap () instruction.

Another algorithm using the TestAndSet ( ) instruction that satisfies all the critical-section requirements.

-

The common data structures are

- boolean waiting[n] ;
- boolean lock;
- These data structures are initialized to false.

semaphores

It is used to provide synchronization among multiple processes running concurrently.

- TestAndset() and swap() instructions are complicated for application programmers to use. To overcome this difficulty synchronization tool called Semaphore is used.
- Semaphore is an integer variable, accessed only through two atomic operations,

. Wait()---[P]
. Signal()----[V]

-

All the modifications to the integer value of the semaphore in the wait () and signalO operations must be executed indivisibly. That is, when one-process modifies the semaphore value, no other process can simultaneously modify that same semaphore value.

Types of Semaphores

Counting Semaphores-

-

counting semaphore can range over an unrestricted domain.

- A counting semaphore has two components-
- 1. An integer value
- 2. An associated waiting list (usually a queue)

- The value of counting semaphore may be positive or negative.

-

Counting semaphore is initialized to the number of resources available.

- A process that wishes to use a resource performes wait() operation on the semaphore(decementing count).
- When a process releses a resource, it performs a signal() operation on the semaphore(incrementing the count).
- When the count or the

semaphore goes to 0, all resources are being used. After that, processes that wish to use a resource will block until the count becomes greater than 0.

PRACTICE PROBLEMS BASED ON COUNTING SEMAPHORES IN OS

-

A counting semaphore S is initialized to 10. Then, 6 P operations and 4 V operations are performed on S. What is the final value of S?

- P operation also called as wait operation decrements the value of semaphore variable by 1

- V operation also called as signal operation increments the value of semaphore variable by 1
- Final value of semaphore variable S
- = 10 – (6 x 1) + (4 x 1)
- = 10 – 6 + 4
- =8

A counting semaphore S is initialized to 7. Then, 20 P operations and 15 V operations are performed on S. What is the final value of S?

-

Final value of semaphore variable S

- = 7 – (20 x 1) + (15 x 1)
- = 7 – 20 + 15
- = 2

Binary semaphore

.

can range only between 0 and 1.

. binary semaphores are known as mutex locks.

-

If the value of binary semaphore is 1,

- The value of binary semaphore is set to 0.
- The process is allowed to enter the critical section.
- If the value of binary semaphore is 0,
- The process is blocked and not allowed to enter the critical section.
- The process is put to sleep in the waiting list.

Disadvantages of the above semaphore

-

While process is in its critical section,any oter process that tries to enter its critical section must LOOP CONTINUOUSLY in the entry code.

- Busy waiting wastes CPU CYCLES that some other process might be able to use productively.
- This type of semaphore is also called as SPINLOCK because the process”spins” while waiting for the lock.

-

To overcome the need for busy waiting,modification done on wait() and signal() semaphore operations.

- Instead of looping continuously in the entry code, that process is put into waiting queue.

Semaphore(modified)

Classic problem of synchronization