RAMA UNIVERSITY

www.ramauniversity.ac.in

FACULTY OF ENGINEERING & TECHNOLOGY

BCA-307 Operating System

Lecturer-11

Manisha Verma
Assistant Professor
Computer Science & Engineering

Synchronization

Background
Peterson’ s Solution
Synchronization Hardware

Synchronization

- To present the concept of process synchronization.
- To introduce the critical-section problem, whose solutions can be used to ensure the consistency of shared data
- To present both software and hardware solutions of the critical-section problem
- To examine several classical process-synchronization problems
- To explore several tools that are used to solve process synchronization problems

Background

- Processes can execute concurrently
  - May be interrupted at any time, partially completing execution
- Concurrent access to shared data may result in data inconsistency
- Maintaining data consistency requires mechanisms to ensure the orderly execution of cooperating processes
- Illustration of the problem:

Suppose that we wanted to provide a solution to the consumer-producer problem that fills ***all*** the buffers. We can do so by having an integer **counter** that keeps track of the number of full buffers. Initially, **counter** is set to 0. It is incremented by the producer after it produces a new buffer and is decremented by the consumer after it consumes a buffer.

Peterson's Solution

- Good algorithmic description of solving the problem

- Two process solution

Assume that the load and store machine-language instructions are atomic; that is, cannot be interrupted.

The two processes share two variables:

int turn;
Boolean flag[2]

The variable turn indicates whose turn it is to enter the critical section
The flag array is used to indicate if a process is ready to enter the critical section. flag[i] = *true* implies that process P_i is ready!

```
do {
        flag[i] = true;
        turn = j;
        while (flag[j] && turn = = j);
                critical section
        flag[i] = false;
                remainder section
    } while (true);
```

Solution (Cont.)

Provable that the three CS requirement are met:

1. Mutual exclusion is preserved

   P_i enters CS only if:

   either **flag[j] = false** or **turn = I**

2. Progress requirement is satisfied
3. Bounded-waiting requirement is met

Synchronization Hardware

Many systems provide hardware support for implementing the critical section code.

All solutions below based on idea of **locking**
- Protecting critical regions via locks

Uniprocessors – could disable interrupts
- Currently running code would execute without preemption
- Generally too inefficient on multiprocessor systems
  - Operating systems using this not broadly scalable

Modern machines provide special atomic hardware instructions
  - **Atomic** = non-interruptible
- Either test memory word and set value
- Or swap contents of two memory words

```
do {
        acquire lock
                critical section
        release lock
                remainder section
} while (TRUE);
```

MCQ

To enforce ………………….. two functions are provided enter-critical and exit-critical, where each function takes as an argument the name of the resource that is the subject of competition.

A) Mutual Exclusion
B) Synchronization
C) Deadlock
D) Starvation

In ………………. only one process at a time is allowed into its critical section, among all processes that have critical sections for the same resource.

A) Mutual Exclusion
B) Synchronization
C) Deadlock
D) Starvation

Which of the following is/are the disadvantages of machine instruction approach to enforce mutual exclusion.
i) Busy waiting employees ii) hard to verify iii) starvation is possible iv) Deadlock is possible
A) i, ii and iii only
B) ii, iii and iv only
C) i, iii and iv only
D) All i, ii, iii and iv

……………………. techniques can be use to resolve conflicts, such as competition for resources, and synchronize processes so that they can co-operate.
A) Mutual Exclusion
B) Synchronization
C) Deadlock
D) Starvation