RAMA UNIVERSITY

www.ramauniversity.ac.in

FACULTY OF ENGINEERING & TECHNOLOGY

BCS-501 Operating System

Lecturer-09

Manisha Verma
Assistant Professor
Computer Science & Engineering

Process Synchronization

Background
The Critical-Section Problem

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.

Producer

```
while (true)
{
            /* produce an item in next produced */

            while (counter == BUFFER_SIZE) ;

                        /* do nothing */

            buffer[in] = next_produced;

            in = (in + 1) % BUFFER_SIZE;

            counter++;
```

Consumer

```
while (true)
 {
                while (counter == 0)

                                ; /* do nothing */

                next_consumed = buffer[out];

                out = (out + 1) % BUFFER_SIZE;

      counter--;

                /* consume the item in next consumed */
}
```

Critical Section Problem

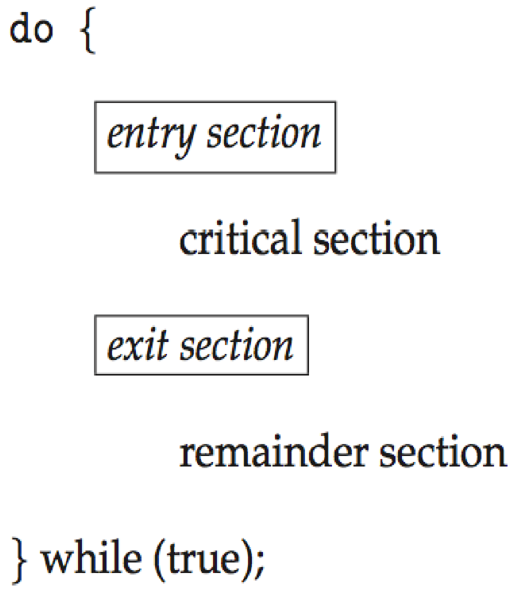
•Consider system of *n* processes $\{p_0, p_1, \ldots p_{n-1}\}$

•Each process has critical section segment of code

- ➢Process may be changing common variables, updating table, writing file, etc
- ➢When one process in critical section, no other may be in its critical section

•Critical section problem is to design protocol to solve this…

➢Each process must ask permission to enter critical section in entry section, may follow critical section with exit section, then remainder section structure of process P_i



```
do {
    entry section
        critical section
    exit section
        remainder section
} while (true);
```

Solution to Critical-Section Problem

1. Mutual Exclusion - If process P_i 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 granted
   - Assume that each process executes at a nonzero speed
   - No assumption concerning relative speed of the *n* processes

Critical-Section Handling in OS:----------

Two approaches depending on if kernel is preemptive or non- preemptive

- Preemptive – allows preemption of process when running in kernel mode
- Non-preemptive – runs until exits kernel mode, blocks, or voluntarily yields CPU
  - Essentially free of race conditions in kernel mode

MCQ

Mutual Exclusion enter…

A. Critical section single process
B. Critical section no process
C. Critical section atleast two process
A. None

Mutual Exclusion is allowed..

A. Interleaving
B. Sharable
C. Mutable lock
D. None

Preemptive – allows preemption of process when…..

A. running in kernel mode
B. running in user mode
C. running in compiler mode
D. None

Concurrent access to……….

A. shared data
B. Not sharable data
C. Mutable data
D. None

Kernel is a….

A. Os
B. Compiler
C. Process
D. CPU