JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD

Lecture-11 By Vivek Dubey

COMPUTER SCIENCE AND ENGINEERING

CS403PC

COMPUTER SCIENCE AND ENGINEERING (ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING)

CS403PC

BHARAT INSTITUTE OF ENGINEERING AND TECHNOLOGY

Scheme of Operating System

S. No.	Course Code	Course Title	L	T	P	Credits
3	CS403PC	Operating Systems	3	0	0	3
6	CS406PC	Operating Systems Lab	0	0	3	1.5

Lecture-11 By Vivek Dubey

UNIT-2:

- **Process and CPU Scheduling –**
  - Process concepts and scheduling, Operations on Processes, Cooperating Processes, Threads, and Inter Process Communication (IPC), Scheduling Criteria, Scheduling Algorithms, Multiple - Processor Scheduling.
- **System call interface for process management-**
  - fork, exit, wait, waitpid, exec

Lecture-11 By Vivek Dubey

Cooperating Processes and IPC

- A process can either be *independent* or c*o-operate* with other processes.
- Any process that shares data *with other processes* **can be called** *as a cooperating process.*
- Cooperating processes require **an inter-process communication (IPC) mechanism** *for exchanging data and information between processes*.

Cooperating Processes and IPC

- Cooperating processes are those that ***can affect*** or ***are affected*** *by other processes running on the system*.
- Cooperating processes ***may share*** data with each other.
- There may be many reasons *for the requirement of cooperating processes.*
- Some of these are given as follows –
  - ✓Modularity
  - ✓Information Sharing
  - ✓Convenience
  - ✓Computation Speedup

- **Modularity**
  - Modularity involves ***dividing complicated tasks*** into smaller subtasks.
  - These subtasks ***can be completed*** by different cooperating processes.
  - This ***leads*** to faster and more efficient completion ***of the required tasks***.
- **Information Sharing**
  - Sharing of information between multiple processes ***can be accomplished*** using cooperating processes.
  - This ***may include access*** to the same files.
  - A mechanism ***is required*** so that the processes ***can access*** the files in parallel to each other.
- **Convenience**
  - There are many tasks that a user needs to do ***such as compiling, printing, editing etc.***
  - It is convenient if these tasks can be managed ***by cooperating processes.***
- **Computation Speedup**
  - Subtasks of a single task can be performed parallelly ***using cooperating processes***.
  - This increases the computation speedup ***as the task can be executed faster***.
  - However, this is only possible ***if the system has multiple processing elements.***

Methods of Cooperation

- ***There are two fundamental models*** of interprocess communication:

  ✓Cooperation by Sharing

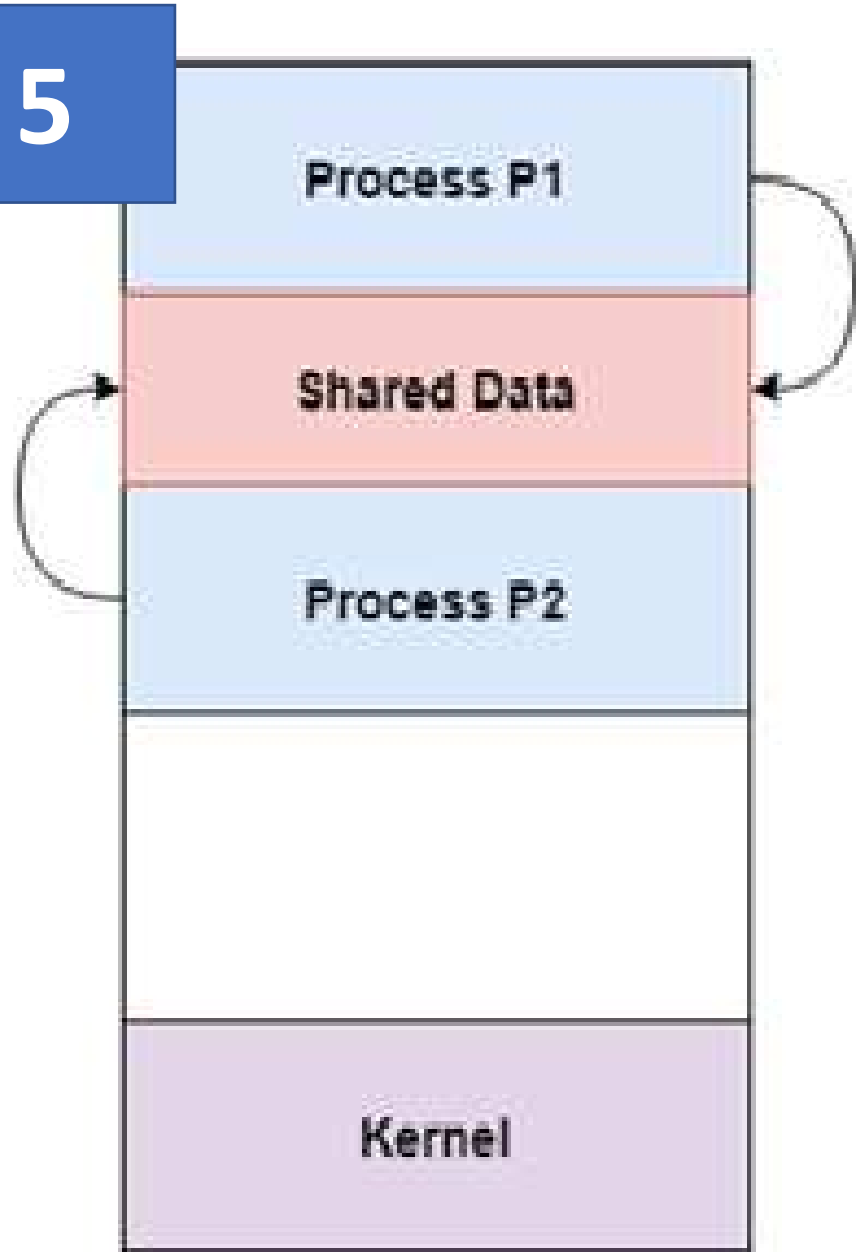
  ✓Cooperating Processes

Cooperation by Sharing

- The cooperating processes ***can cooperate with each other using*** shared data *such as memory, variables, files, databases etc.*

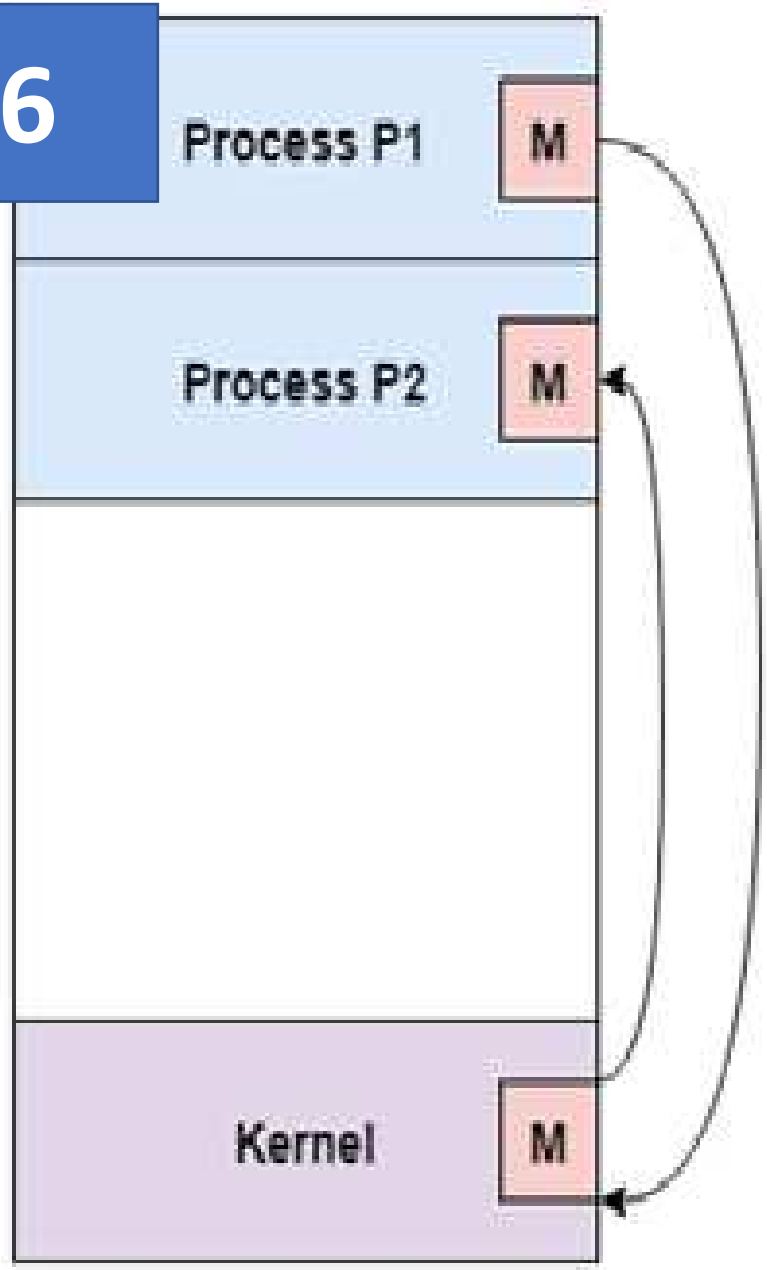
- Critical section ***is used to provide*** *data integrity* and writing is mutually exclusive *to prevent inconsistent data.*

- In the diagram, Process P1 and P2 ***can cooperate*** *with each other using* shared data *such as memory, variables, files, databases etc.*



Cooperating Processes

- The cooperating processes ***can cooperate*** *with each other using messages*.
- **This may lead to deadlock** *if each process is waiting for a message* from the other to perform a operation.
- **Starvation is also possible** *if a process never receives a message*.
- A diagram that demonstrates *cooperation by communication* –



Communication Link

- A communication link **must exist** between sender and receiver.
- For logical link implementation, *the following methods:*
  - ***Direct or indirect*** communication
  - ***Synchronous or asynchronous*** communication
  - ***Automatic or explicit*** buffering

Direct Communication

- Direct communication: *Each process that* wants to communicate ***must explicitly*** name the recipient or sender of the communication.
- For example,
  - send(P, message)—***Send a message to process P.***
  - receive(Q, message)—***Receive a message from process Q.***
- **Its properties are,**
  - A link is established automatically *between every pair of process.*
  - The processes need to know *only other's identity*
  - A link is associated with *exactly two processes*.
  - Between each pair of processes, *there exists exactly one link.*

Indirect Communication

- **Indirect communication**: The messages *are sent to and received from* mailboxes, or ports.
  - Each mailbox has *a unique identification*.
  - Two processes can communicate only *if they have a shared mailbox.*
  - A mailbox may be owned *either by a process or by the operating system*.
- The send() and receive() operations ***are defined as follows***:
  - **send(A, message)**—Send a message to mailbox A.
  - **receive(A, message)**—Receive a message from mailbox A.
- Its properties are,
  - A link is established only *if both members of the pair have a shared mailbox*.
  - A link may be associated with *more than two processes.*
  - Between each pair of communicating processes, *a number of different links* **may exist**, *with each link corresponding to one mailbox*.

Synchronous or Asynchronous Communication

- Message passing may be either synchronous and asynchronous ***also known as blocking or nonblocking***.
  - ***Blocking send***. The sending process is blocked *until the message is received.*
  - ***Nonblocking send***. The sending process sends *the message and resumes operation*.
  - ***Blocking receive.*** The receiver blocks *until a message is available*.
  - ***Nonblocking receive***. The receiver retrieves *either a valid message or a null.*

Automatic or Explicit Buffering

- Messages exchanged ***by communicating processes*** reside in a temporary queue.
- Basically, such queue can be implemented in three ways:
  - **Message system** *with no buffering*:
    1. Zero Capacity.
  - **Message system** *with automatic buffering*:
    2. Bounded Capacity and
    3. Unbounded Capacity

Message System with No Buffering

- **Message system with no buffering:**
  - ✓ *Zero capacity*.
    - No messages are queued on a link.
    - *Sender must wait for receiver.*

Message System with Automatic Buffering

✓ Bounded capacity:

- The queue has ***finite length n***.
- Sender must wait, ***if link is full***.

Message System with Automatic Buffering

✓Unbounded capacity:

- The queue's length is ***infinite***;
- Sender ***never wait.***

CLASSICAL PROBLEM OF SYNCHRONIZATION

- Following are ***some of the classical problem*** faced while process synchronization ***in systems*** where cooperating processes are present:
  - ✓Bounded Buffer Problem
  - ✓The Readers Writers Problem
  - ✓Dining Philosophers Problem

Bounded Buffer Problem

- The bounded-buffer problems (**the producer-consumer problem**) *is a classic example of concurrent access* to a shared resource.

- A bounded buffer **lets** multiple producers and multiple consumers *share a single buffer*.

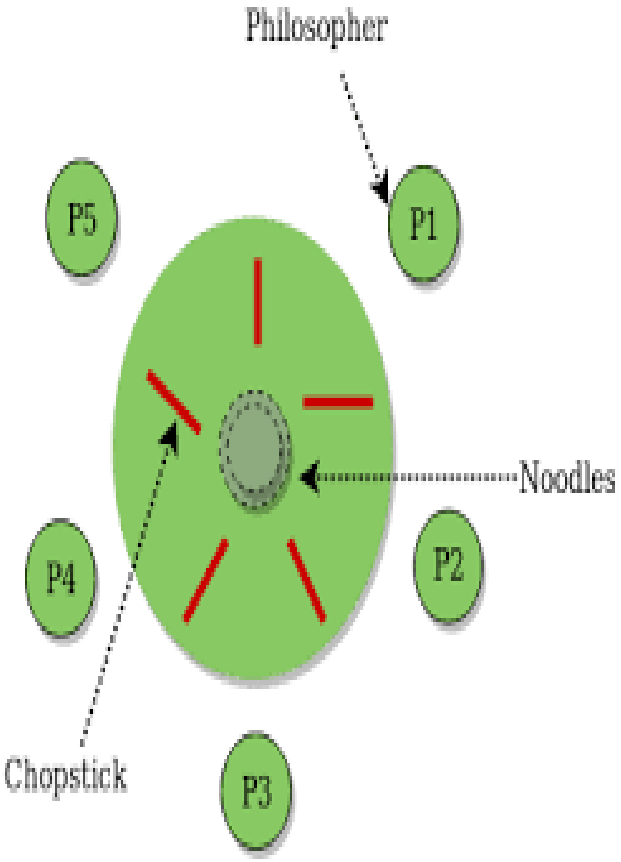
- ***Producers write data to the buffer*** and ***consumers read data from the buffer***:
  - *Producers must block* ***if the buffer is full.***
  - *Consumers must block* ***if the buffer is empty***.

The Readers Writers Problem

- A resource is shared among many processes, *each belonging to one of two processes.* They are either **reader** or **writer**.
- ***In this problem***, any number of readers ***can read*** from the shared resource simultaneously, ***but at a time only*** one writer can write to the shared resource.
- Also, when a writer ***is writing data to the resource***, at that time no other process can access the resource.
- **Readers do not write**, readers only read.
- **If a process is writing**, no other process can read it.

Dining Philosophers Problem

- The dining philosopher's problem involves *the allocation of limited resources from a group of processes* ***in a*** deadlock-free and starvation-free manner.
- There are five philosophers ***sitting around a table***, in which there are five chopsticks ***kept beside them*** and a bowl of rice in the center.
- When a philosopher wants to eat, ***he uses two chopsticks*** - one from their left and one from their right.
- When a philosopher wants to think, ***he keeps down both chopsticks*** at their original place.



Deadlock Problem

- A Process or thread enters a waiting state because a requested system resource is held by another waiting process.

Starvation Problem

- When high priority processes keep executing and low priority processed get blocked for indefinite time.