UNIT-5

TASK COMMUNICATION

CONTENTS

5.1. TASK COMMUNICATION

- In a multitasking system, multiple tasks/processes run concurrently(pseudo parallelism) and each process may or may not interact between.
- Based on degree of interaction, the processes running on an OS are classified as,

i) Co-operating process: In this model, one process requires the inputs from other processes to complete its execution

ii) Competing process: it do not share anything among themselves but they share the system resources

- ✓ Co-operating processes exchanges information and communicate through the following methods:

i) Co-operation through sharing: The co-operating process exchange data through some shared resources.

ii) Co-operation through communication: no data is shared between the processes. But they communicate for synchronisation

- The mechanism through which processes/tasks communicate each other is known as inter process/ Task communication (IPC)
- IPC is essential for process co-ordination.
- The various types of IPC mechanisms adopted by process are kernel(OS) dependent.
- Some of the important IPC mechanisms adopted by various kernel are: Shared memory(pipes), message passing(Message queue, mailbox, Signalling), RPC

5.1(a). SHARED MEMORY

- Processes share some area of the memory to communicate among them

- Information to be communicated by the process is written to the shared memory area
- Other processes which require this information can read the same from the shared memory area.
- The implementation of shared memory concept is kernel dependent.

- Different mechanisms are adopted by different kernels for implementing this shared memory concept. Namely

a) pipes

b) Memory mapped objects

a) Pipes

- ‘pipe’ is a section of the shared memory used by processes for communicating
- Pipes follow the client-server architecture
- A process which creates a pipe is known as a pipe server and a process which connects to a pipe is known as pipe client
- a pipe can be considered as a conduitfor information flow and has two conceptual ends

- It can be unidirectional, allowing information flow in one direction or bidirectional allowing bi-directional information flow.

- A unidirectional pipe allows the process connecting at one end of the pipe to write the pipe and the process connected at the other end of the pipe to read the data.

- Whereas a bi-directional pipe allows both reading and writing at one end

- The unidirectional pipe can be visualised as

The implementation of 'pipes'is also OS dependent

Microsoft windows Desktop OS supports two types of 'pipes' for IPC. They are,

i) Anonymous pipes (Unidirectional)

ii) Named pipes(uni or bi-directional)

b) Memory mapped objects

- Memory mapped object is a shared memory technique adopted by certain RTOS for allocating a shared block of memory which can be accessed by multiple process simultaneously
- In this approach, a mapping object is created and physical storage for it is reserved and committed

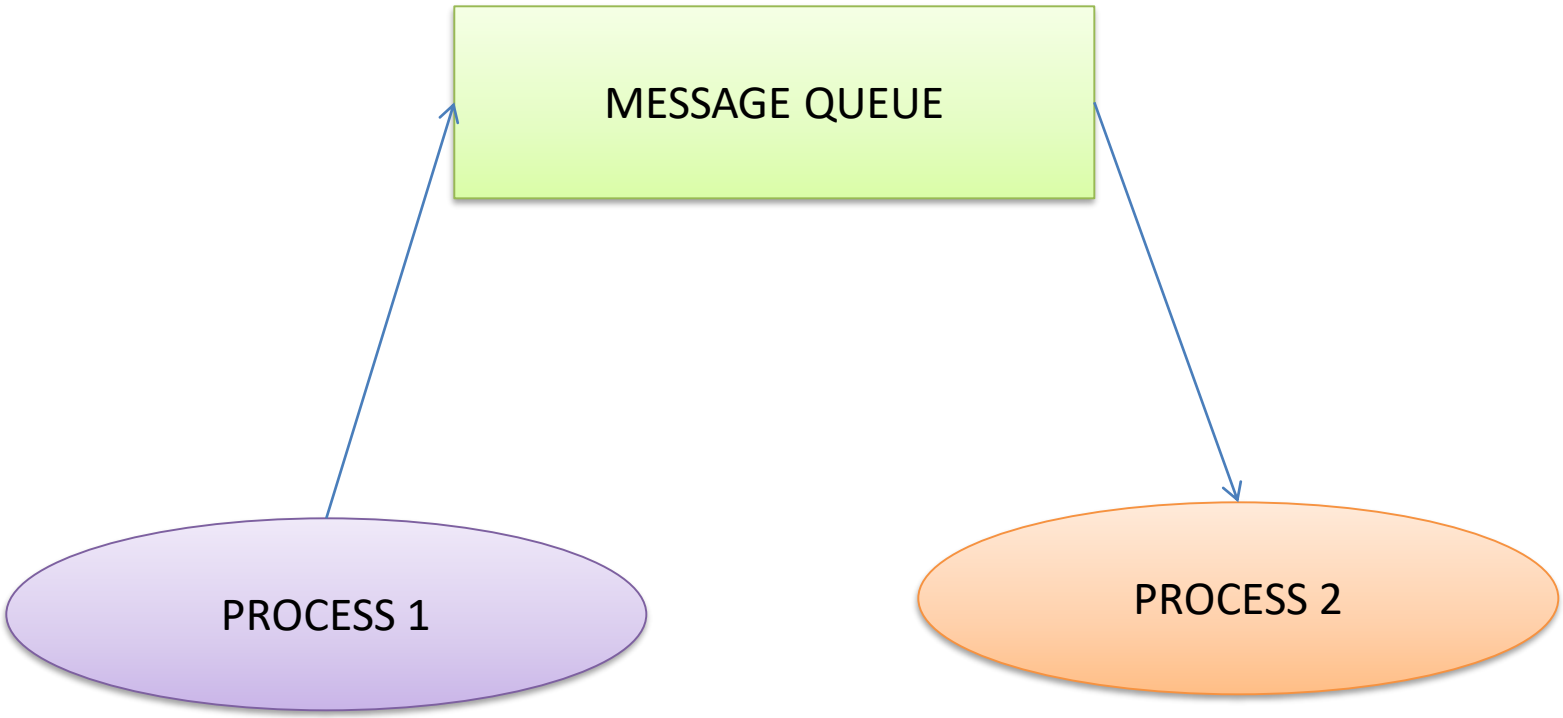
5.2(b). Message passing

- Message passing is an (a)synchronous information exchange mechanism used for inter process/thread communication
- The major difference between shared memory and message passing technique is that, though shared memory lots of data can be shared whereas only limited amount of info/data is passed through the message passing
- Also message passing is relatively fast
- Message passing operation between the processes is classified into

  a) Message queue b)mailbox c)Signalling

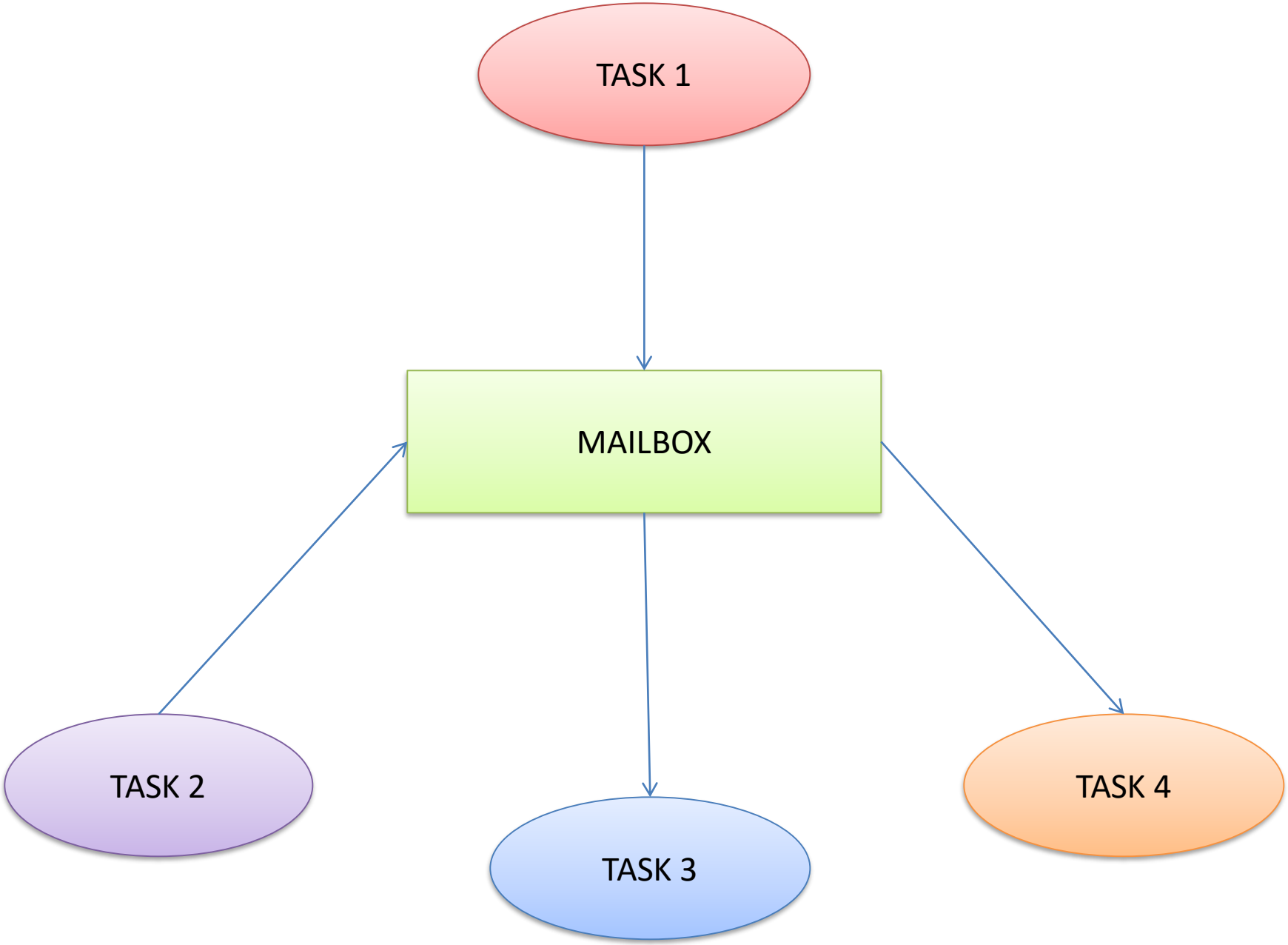
a) Message queue

- Usually the process which wants to talk to another process posts the message to a First-In-First-Out queue is called ‘message queue’, which stores the messages temporarily in a system defined memory to pass it to desired process.
- Messages are send and received through SEND and RECEIVE methods.
- Send and Receive is also OS kernel dependent.


MESSAGE QUEUE
PROCESS 1
PROCESS 2

b) mailbox

- Mailbox is an alternate form of ‘message queues’ and it is used in certain RTOS for IPC
- Mailbox for IPC in RTOS is usually used for one way messaging.
- The task/thread which wants to send a message to other task/threads creates a mailbox for a posting the messages.
- Mailbox server---> taks which are send message
- Mailbox client --- > tasks which are receive message.


TASK 1
MAILBOX
TASK 2
TASK 3
TASK 4

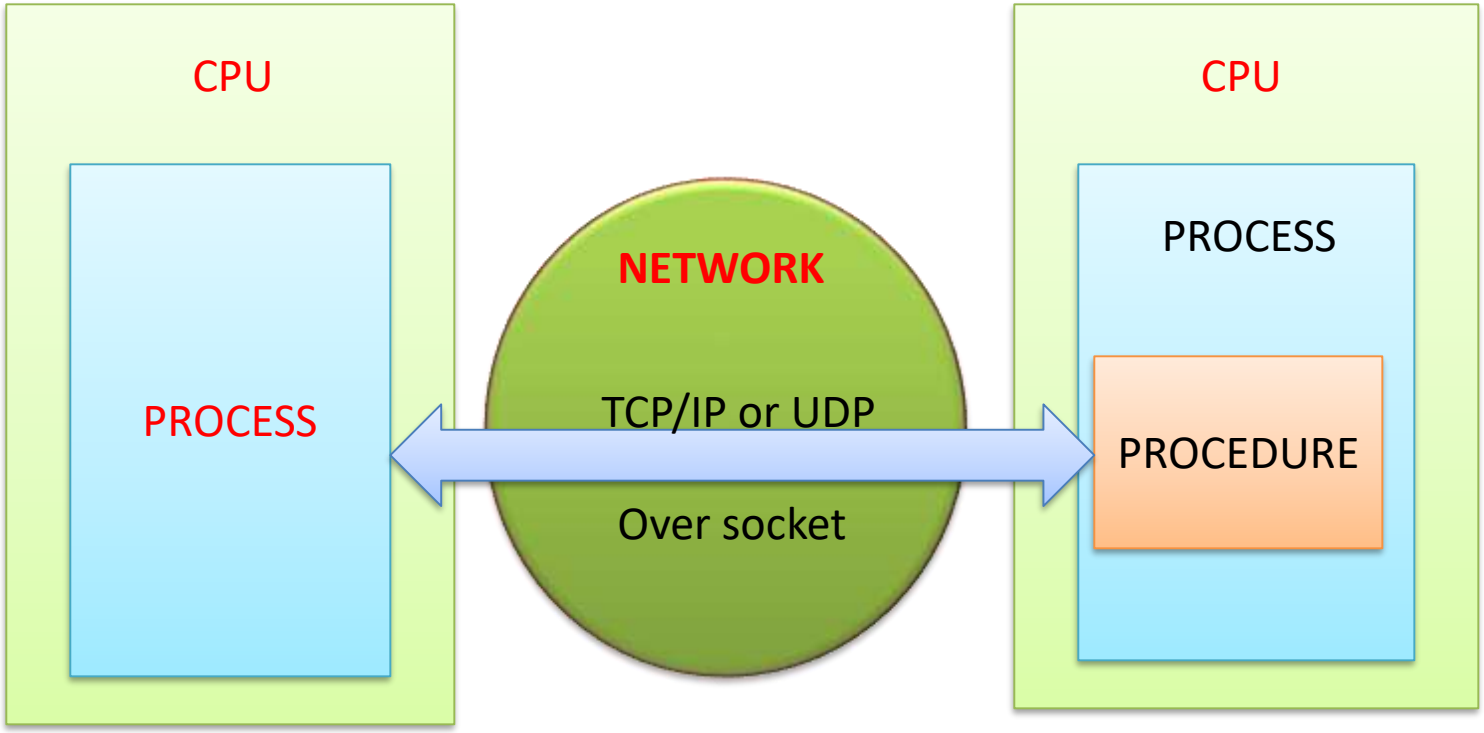
c) Signalling

- Signalling is a primitive way of communication between processes/threads
- Signals are used for asynchronous notifications where one process/thread fires a signal, indicating the occurence of a scenario which the other process(es)/Thread(s) is waiting
- Signals are not queued and they do not carry any data.

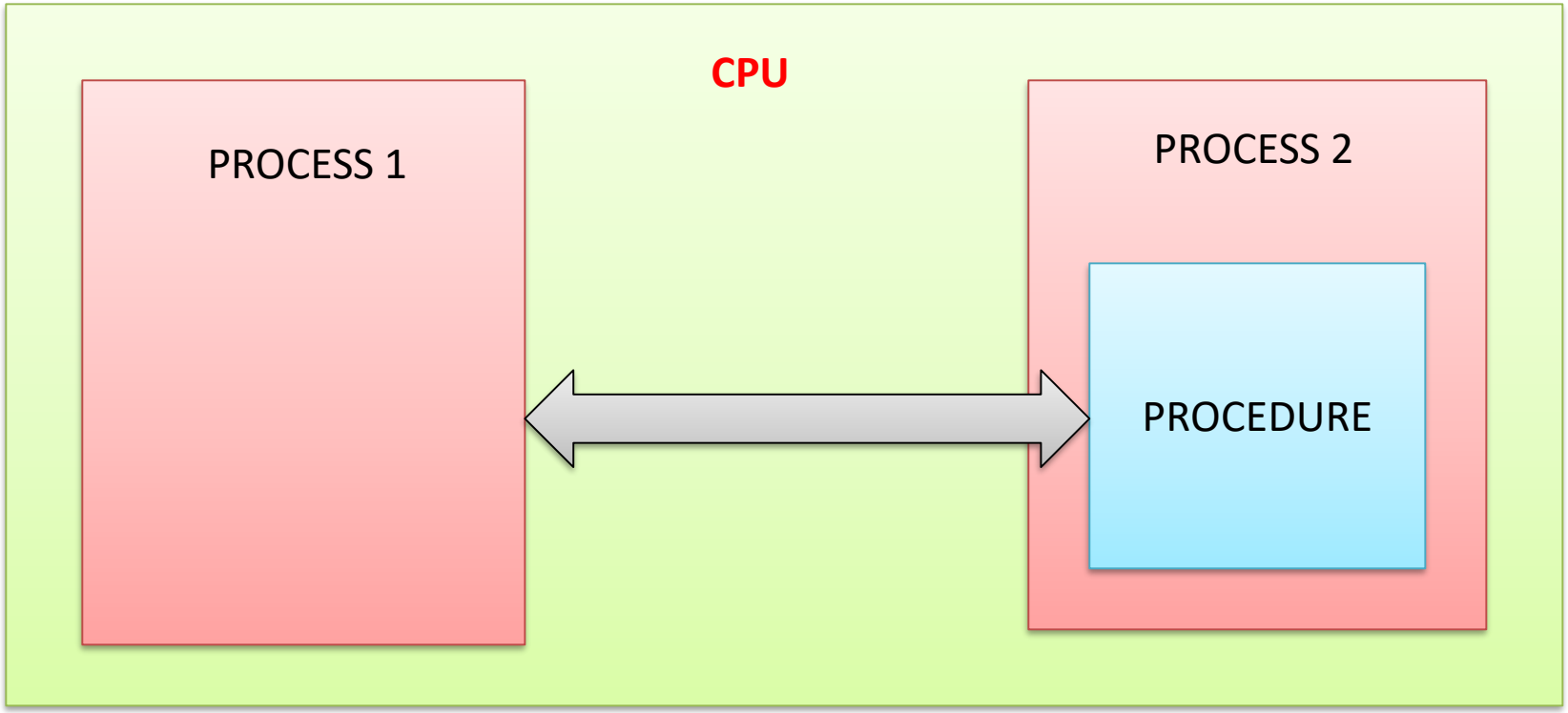
5.1(c). RPC and SOCKETS

- RPC is the IPC mechanism used by a process to call a procedure of another process running on the same CPU or on the different CPU which is interconnected in a network.

- RPC is mainly used for distributed applications like client-server applications.



Processes running on different CPUs Which are Networked



Processes running on the CPU

5.2. TASK SYNCHRONISATION

- In a multitasking environment, multiple processes run concurrently and share the system resources.
- Each process has its own boundary wall and they communicate with each other with different IPC mechanisms including shared memory and variables
- Imagine a situation where two processes try to access a shared memory area where one process tries to write when the other process is trying to read from this memory

- What could be the result in these scenarios? Obviously unexpected results.
- How these issues can be addressed? The solution is, make each process aware of the access of shared resource either directly or indirectly.

  ‘ The act of making processes aware of the access of shared resources by each process to avoid conflicts is knows as, “TASK SYCNCHRONIZATION” ‘

5.2(a). Task sychronisation/communication issues

a) Racing: Racing or Race condition is the situation in which multiple processes compete(Race) each other to access and manipulate shared data concurrently.

b) Deadlock: it is the condition in which a process is waiting for a resource held by another process which is waiting for resource held by the first process

Deadlock Handling:

Ignore deadlocks

Detect and Recover

Avoid deadlocks

Prevent Deadlocks

C) Dining Philosophers problem

d) producer-consumer/ Bounded Buffer Problem

e) Readers-Writers Problem

f) Priority inversion

5.2(b). Task synchronisation Techniques

a) Mutual Exclusion through Busy waiting/spin lock

b) Mutual Exclusion through sleep & wakeup

i) Semaphore

ii) Binary semaphore (Mutex)

iii) Critical section objects

iv) Events

5.3. DEVICE DRIVERS

- Device driver is a piece of software that acts as a bridge between the operating system and the hardware.
- All the device related access should flow through the OS kernel and the OS kernel routes it to the concerned hardware peripheral.
- OS provides interfaces in the form of Application Programming Inteface (API) for accessing the hardware.

5.4. How to choose RTOS

5.4(a) Functional Requirements:

i) Processor support
ii) Memory requirements
iii) Real-time concepts
iv) Kernel and interrupt latency
v) IPC and task synchronisation
vi) Modularisation support
vii) Support for networking and communication
viii) Development Language support

5.4(b) Non-functional Requirements

i) Custom developed or Off the shelf

ii) Cost

iii) Development and debugging tools availability

iv) Ease of erase

v) After sales