Real Time System

- A real time system has well-defined, fixed time constraints. Processing must be done within the defined constraints, or the system will fail.
- A real time system functions correctly only if it returns the correct result within its time constraints.
- This system is used when we require quick response against input.
- In this system, task is completed within specified time.
- It is classified in following two types-

1. **Hard Real Time System:** In hard Real Time system task will be fail if response time against input is more than specified time.
2. **Soft Real Time System:** In Soft Real Time system inaccurate result will be found if response time against input is more than specified time.

2. Real Time Purpose Operating System (RTOS):

- Operating Systems, which are deployed in embedded systems demanding real-time response
- Deterministic in execution behavior. Consumes only known amount oftime for kernel applications
- Implements scheduling policies for executing the highest priority task/application always
- Implements policies and rules concerning time-critical allocation of a system's resources
- Windows CE, QNX, VxWorks , MicroC/OS-II etc are examples of Real Time Operating Systems (RTOS)

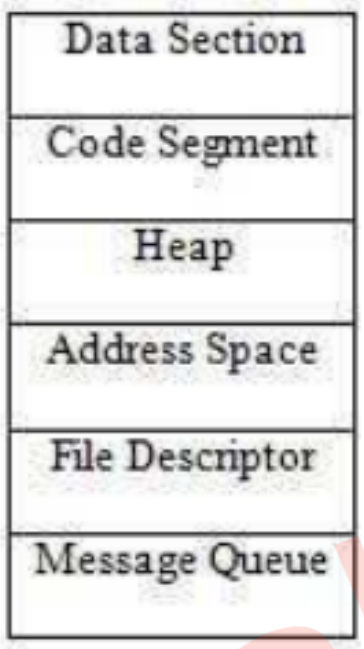
Thread: Light weight Process is called Thread.

Thread Vs Process

Thread	Process
Light weight Process	Heavy Weight Process
Thread Switching does not need interaction with OS	Process Switching needs interaction with OS
Thread can share Code segment, Data Section, File Descriptor, address space, heap etc	Processes own their separate Code segment, Data Section, File Descriptor, address space, heap etc

Threads has its own

Threads of same process share



Note: Local variables of the Process are stored in Stack, Global variables are stored in Data Section and dynamically created variables are stored in Heap.

Time Sharing or Multi-Tasking System

- It is also called fair share system or multi programming with round robin System.
- It is extension of multi Programming System.
- In this system CPU switches between processes so quickly that it gives an illusion that all executing at same time.
- This system can be used to handle multiple interactive tasks.

Advantages

- ✓ High CPU utilization.
- ✓ Less waiting time, response time etc.
- ✓ Useful in current scenario when load is high.

Disadvantages

- ✓ Process scheduling is difficult.
- ✓ Main memory management is required.
- ✓ Problems like memory fragmentation may occur.

Multiprocessing System

- ❖ A system is called multiprocessing system if two or more CPU within a single computer communicate with each other and share system bus memory and I/O devices.
- ❖ It provides true parallel execution of processes.

Type of Multiprocessing system

1. **Asymmetric Multiprocessing System:** In this system, each processor is assigned a specific task. A master processor controls the system. The other processors called slave processors either look to master for instruction or have predefined tasks i.e. master-slave relationship hold in this type of system.

2. **Symmetric Multiprocessing System:** In this system each processor performs all tasks within the operating system i.e. all processor are peers, no master-slave relationship exists between processors.

Advantages:

- ✓ **Increased Throughput:** By increasing number of processors, we expect to get more work done in less time.
- ✓ **Economy of Scale:** multiprocessor system can cost less than equivalent multiple single processor systems.
- ✓ **Increased Reliability:** Since work is distributed among several processors; failure of one processor will not halt the system only slow it down.

Disadvantages

- ✓ It is complex system.
- ✓ Process scheduling is difficult in this system.
- ✓ It required large size of Main memory.

CPU SCHEDULING

Removal of the running process from the CPU and selection of another process on the basis of particular strategy is called CPU Scheduling.

Non Preemptive Vs Preemptive

Non Preemptive Scheduling or Cooperative Scheduling: In non-preemptive scheduling, once the CPU is allocated to a process, keeps the CPU until it releases the CPU either by terminating or by switching to the waiting state i.e. when a process reaches to running state it can either switches to waiting state or terminates.

Preemptive Scheduling: Preemptive scheduling allows a process to switch from running state to ready state or waiting state to ready state. In Preemptive scheduling, once the CPU is allocated to a process, process can release the CPU before terminating the process.

Inter process communication

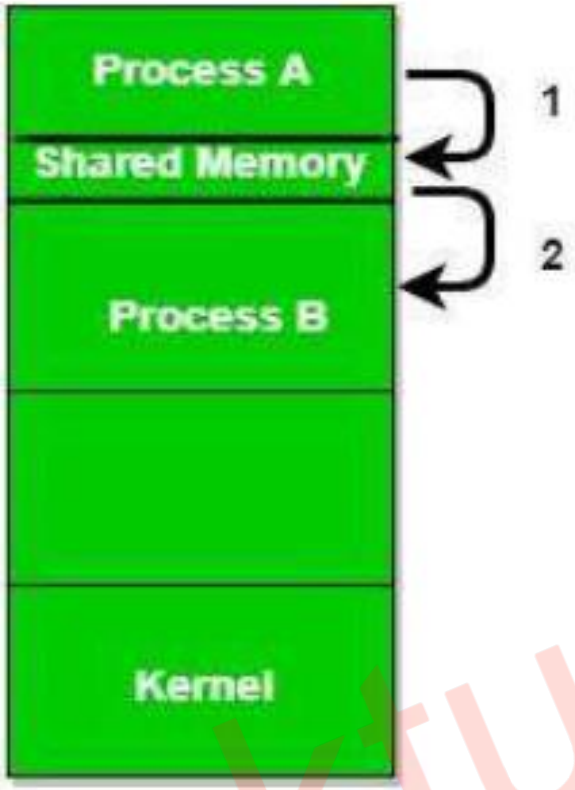
Inter process communication (IPC) is a mechanism which allows processes to communicate each other and synchronize their actions. Processes can communicate with each other using these two ways:

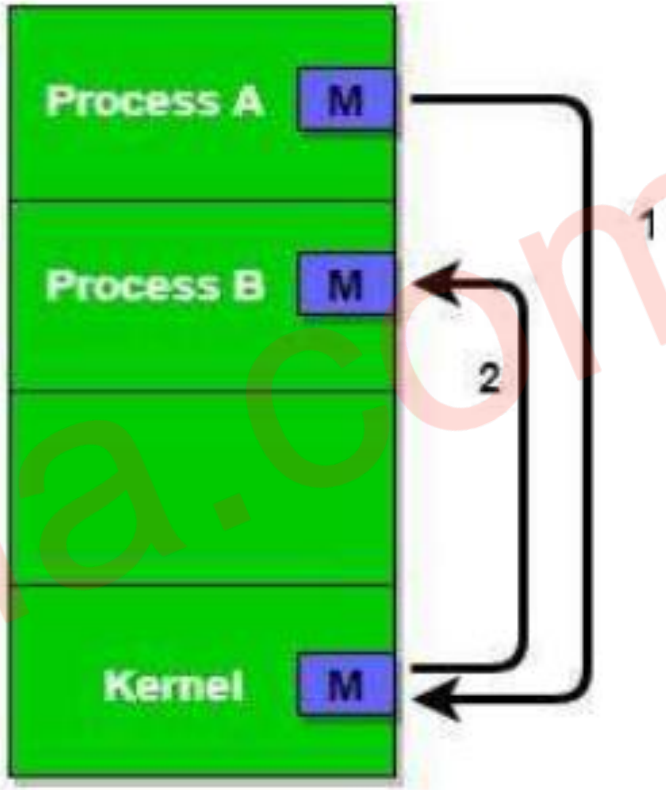
1. Shared Memory
2. Message passing

Shared Memory Method

Shared memory requires processes to share some variable and it completely depends on how programmer will implement it.

One way of communication using shared memory can be- Suppose process1 and process2 are executing simultaneously and they share some resources or use some information from other process, process1 generate information about certain computations or resources being used and keeps it as a record in shared memory. When process2 need to use the shared information, it will check in the record stored in shared memory and take note of the information generated by process1 and act accordingly. Processes can use shared memory for extracting information as a record from other process as well as for delivering any specific information to other process.





Messaging Passing Method

In this method, processes communicate with each other without using any kind of of shared memory. If two processes p1 and p2 want to communicate with each other, they proceed as follow:

- Establish a communication link (if a link already exists, no need to establish it again.)

- Start exchanging messages using basic primitives.

  We need at least two primitives:
  – **send**(message, destinaion) or **send**(message)
  – **receive**(message, host) or **receive**(message)

The message size can be of fixed size or of variable size. If it is of fixed size, it is easy for OS designer but complicated for programmer and if it is of variable size then it is easy for programmer but complicated for the OS designer. A standard message can have two parts: **header and body.** The **header part** is used for storing Message type, destination id, source id, and message length and control information. The control information contains information like what to do if runs out of buffer space, sequence number, priority. Generally, message is sent using FIFO style.

Semaphore

A semaphore S is an integer variable that, apart from initialization, is accessed only through two standard atomic operations: wait () and signal (). The wait () operation is also called P and signal () is also called V.

Definition of wait ()

```
wait(S) {
            while(S <= 0);
            S--;
        }
```

Definition of signal ()

```
signal(S)
        {
            S++;
        }
```

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

Application of Semaphore

1. Solving Multi process Critical Section Problem
2. Resource allocation among various processes
3. Ordering Execution of processes.

Types of Semaphore

1. **Binary Semaphore -** This is also known as **mutex lock**. It can have only two values – 0 and 1. Its value is initialized to 1. It is used to implement solution of critical section problem with multiple processes.
2. **Counting Semaphore -** Its value can range over an unrestricted domain. It is used to control access to a resource that has multiple instances.

Interrupt routine

An interrupt routine, also known as an Interrupt Service Routine (ISR), is a specialized function in computer systems that handles asynchronous events or interrupts. When an external event occurs, such as hardware signaling or timer expiration, the processor suspends its current task and jumps to the ISR to address the event promptly. ISRs are typically short and focused on handling the interrupt as efficiently as possible to minimize system latency. In real-time operating systems (RTOS), ISRs are crucial for maintaining responsiveness and meeting timing requirements in embedded systems. They often prioritize interrupts based on their urgency and can preempt lower-priority tasks to ensure timely handling of critical events. Proper management of interrupt routines is essential for the reliable operation of real-time systems.

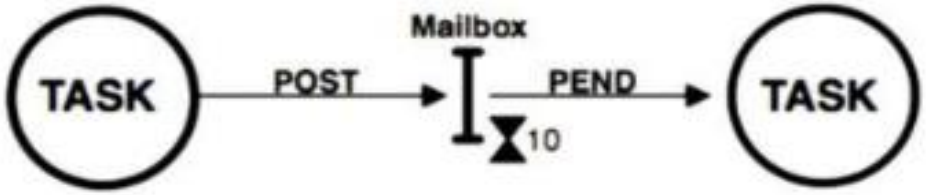
Message Queues

- The message queue is a buffer used in non-shared memory environments, where tasks communicate by passing messages to each other rather than accessing shared variables.
- Tasks share a common buffer pool.
- The message queue is an unbounded FIFO queue protected from concurrent access by different threads.
- Many tasks can write messages into the queue, but only one can read messages from the queue at a time.
- The reader waits on the message queue until there is a message to process.
- Messages can be of any size.

Mailboxes

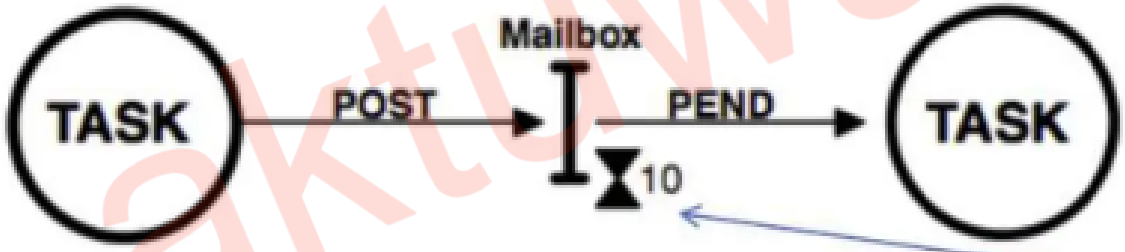
- Tasks can also communicate by sending messages via mailboxes
- Mutual exclusion of the mailbox is handled by the operating system
- A mailbox is a special memory location that one or more tasks can use to transfer data, or generally for synchronization

  The tasks rely on the kernel to allow them to
  - Write to the mailbox via a post operation
  - Read from it via a pend operation
  - Direct access to any mailbox is not allowed
  - A mailbox can only contain one message



Generally, three types of operations can be performed on a mailbox

- Initialize (with or without a message)
- Deposit a message (POST)
- Wait for a message (PEND)



Optional timeout; number of clock ticks the the task will wait for a message

Pipes

- The pipe() system call in OS facilitates interprocess communication by creating a unidirectional communication channel between two processes.
- It allows one process to write data into the pipe, while another process can read from it.
- This mechanism is particularly useful for achieving coordination and data transfer between processes, such as in pipelines or filters.
- Pipes are a fundamental building block for implementing more complex communication and synchronization mechanisms in Unix-like OS Linux and macOS.
- They provide a way for processes to exchange data without the need for shared memory or explicit file operations, enhancing the modularity and efficiency of process communication in a multitasking environment.

	Priority Inversion	Priority Inheritance
1.	In priority inversion, a higher-priority process is preempted by a lower-priority process.	It is a method that is used to eliminate the problems of Priority inversion.
2.	It is the inversion of the priorities of the two processes	With the help of this, a process scheduling algorithm increases the priority of a process, to the maximum priority of any other process waiting for any resource.
3.	It can cause a system to malfunction in our system.	Priority inheritance can lead to poorer worst-case behavior when there are nested locks.
4.	Priority inversions can lead to the implementation of corrective measures.	Priority inheritance can be implemented such that there is no penalty when the locks do not contend,
5.	To deal with the problem of priority inversion we can have several techniques such as **Priority ceiling, Random boosting, etc.**	It is the basic technique at the application level for managing priority inversion.

VxWorks

- **VxWorks** is a real-time operating system (or RTOS) developed as proprietary software by Wind River Systems. First released in 1987, VxWorks is designed for use in embedded systems requiring real-time, deterministic performance and in many cases, safety and security certification for industries such as aerospace, defense, medical devices, industrial equipment, robotics, energy, transportation, network infrastructure, automotive, and consumer electronics

- **Micro-Controller Operating Systems** (**MicroC/OS**, stylized as **µC/OS**, or **Micrium OS**) is a real-time operating system (RTOS) designed by Jean J. Labrosse in 1991. It is a priority-based preemptive real-time kernel for microprocessors, written mostly in the programming language C. It is intended for use in embedded systems.
- **RTLinux** is a hard realtime real-time operating system (RTOS) microkernel that runs the entire Linux operating system as a fully preemptive process. The hard real-time property makes it possible to control robots, data acquisition systems, manufacturing plants, and other time-sensitive instruments and machines from RTLinux applications.

Made By :- AKTU WALA (Satyam Sahu)

- Website :- Extramarkslibrary.com
  :- Aktuwala.com