Real Time Operating System

Operating System

- Developed to use bare machines (hardware)
- Essential software required to work with a computer
- Manage basic hardware resources and provide interface to users and their programs
- Also Controls the execution of application programs

Features of a Real Time operating system

- Clock and Timer support
  - Clock and timer services with adequate resolution are the most important issues in a RTOS
- Real Time priority levels
  - RTOS must support static priority levels.
- Fast task preemption
  - Time duration for which a higher priority task waits before it is allowed to execute is quantitatively expressed as *task preemption time*
- Predictable and fast interrupt latency
  - Interrupt latency is the occurrence of the interrupt and running of the corresponding subroutine
  - Interrupt latency must be less than a few micro seconds

Process

- "Process is a program in execution." It includes current activity as represented by PC, Registers, Stack, etc.
- Process consists of sequentially executable program (codes) and state control by an OS.
- The state during running of a process is represented by the information of process state (created, running, blocked or finished), process structure (data, objects and resources) and process control block.
- It is a computational unit that processes on a CPU and whose state changes under the control of kernel of an OS.

- Process is that executing unit of computation, which is controlled by some processes of the OS

  - for a scheduling mechanism that lets it execute on the CPU- for a resource
  - management mechanism that lets it use the system-memory and other system
  - resources such as network, file, display or printer
  - for access control mechanism
  - for inter-process communication
  - concurrently

Process State

As process executes it changes state. State is defined by the current activity of the process.

Each process may be in one of the following States:

1. **New**: To start execution of a program, a new process is created in memory.

2. **Ready**: Process is waiting to be assigned to the CPU for further execution
A process which is not waiting for external event or not running is in READY state. When CPU is free OS chooses one from list of the ready state processes and dispatch for execution as per scheduling algorithm

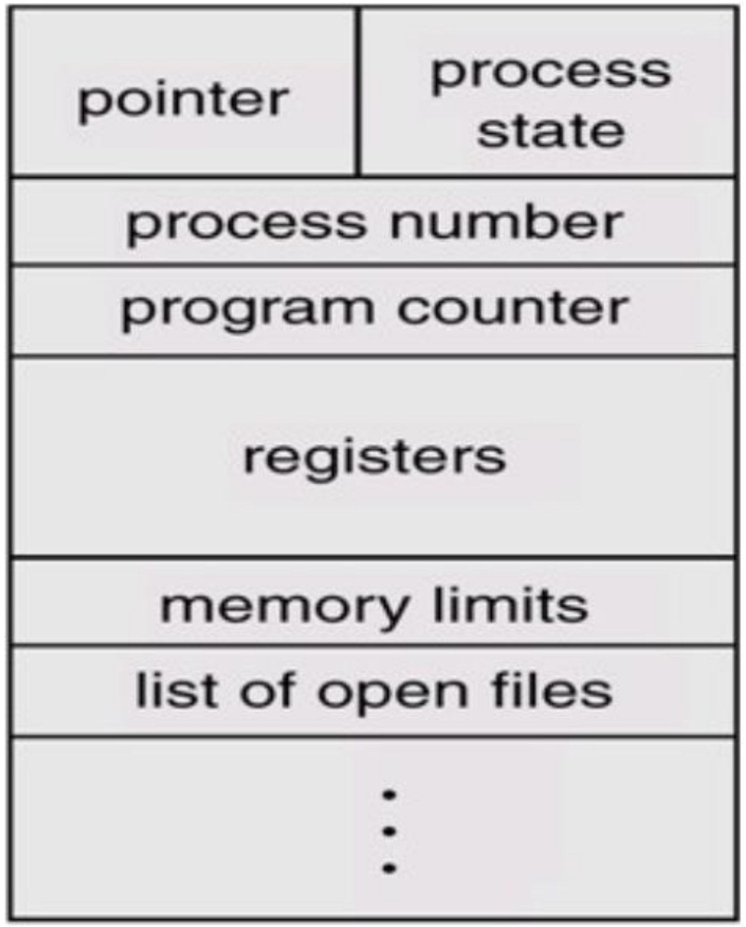
3. **Running:** Instructions are being executed.
Only one process execution by CPU at any given moment.

4. **Blocked/Waiting:** Process is waiting for some event to occur e.g. I/O operation finish.
Blocked process cannot be scheduled for running even if CPU is free.

5. **Terminated:** Process has finished its execution

Process Control Block

- Information about each and every process in our computer is stored into Process Control Block (PCB)
- Also known as Task Control Block.
- Created when a user creates process
- Removed from system when process is terminated
- Stores in protected memory area of the kernel (Memory reserved for OS)



PCB Contents

- **Process State**: **-** Information about various process states such as new, ready, running, waiting, etc.

- **Program Counter**: **-** It shows the address of the next instruction to be executed in the process.

- **CPU registers**: There are various registers in CPU such as accumulators, stack pointer, working register, instruction pointer. PCB stores the state of all these register when CPU switch from one process to another process.

- **CPU Scheduling information**: It includes process priority, pointer to the ready queue and scheduling information.

- **Accounting information**: **-** It includes total CPU time used, real time used, process number, etc.

- **I/O status information**: It includes list of I/O device allocated to the process. It also includes the list of opened file by process in disk. File is opened either for read or write.

- **Memory-management information**: PCB stores the value of registers, address of page table, and other memory management information.

Context Switching

- When CPU switch from one process to another process, CPU saves the information about the one process into PCB (Process Control Block) and then starts the new process..
- The present CPU registers, which include program counter and stack pointer are called context.
- When context saves on the PCB pointed process-stack and register save area addresses, then the running process stops.
- Other process context now loads and that process runs means that the context has switched.

- Context switch is purely overhead because system does not perform any useful work while context switch.

- Context switch times are highly dependent on hardware. Its speed vary from machine to machine depending on the memory speed, registers to be copied and existence of special instructions.

Thread

- Two processes can run at the same time but they do not share memory.

- Suppose we provided software entities that could run at the same time but also shared memory. Such entities exist in many newer operating systems and they are called as threads.

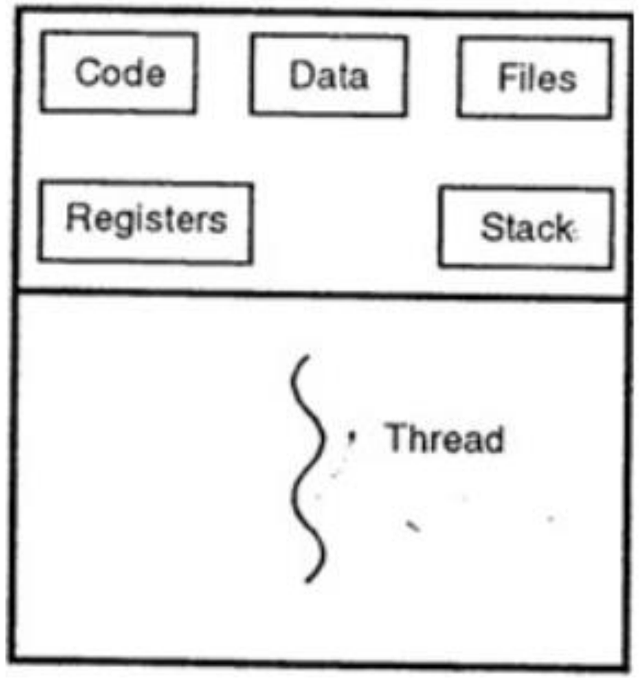
- Many software that run on desktop PCs are multithreaded.

- Example, a word processor may have a thread for displaying graphics, another thread for reading key stroke from the user, and third thread for performing spelling and grammar checking in the background.

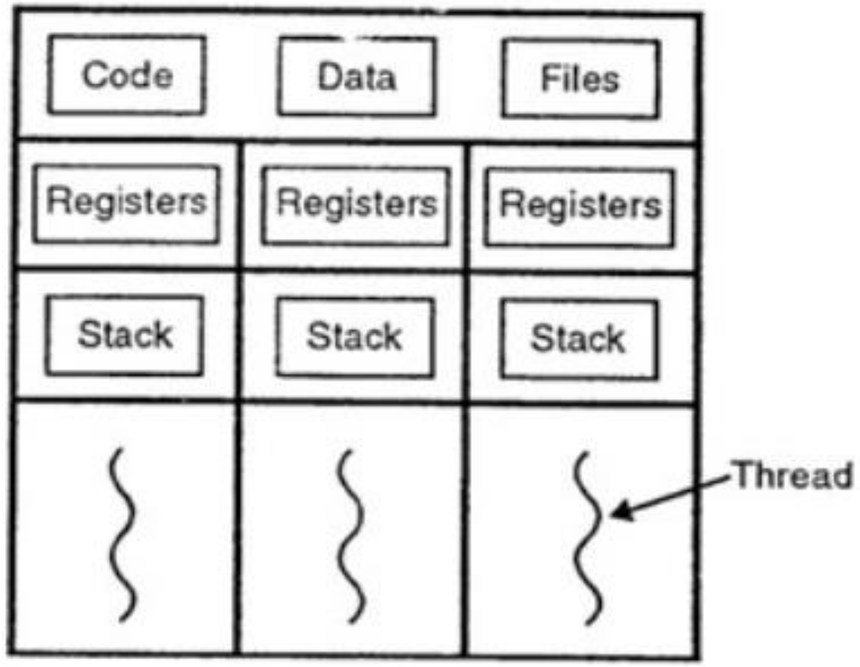
Thread...

- Each thread has independent parameters -- a thread ID, a program counter, a register set, and a stack, priority and its present status.

- A traditional **heavyweight Process (a kernel-level controlled entity)** has a single thread of control. If the process has multiple threads of control, it can do more than one task at a time.



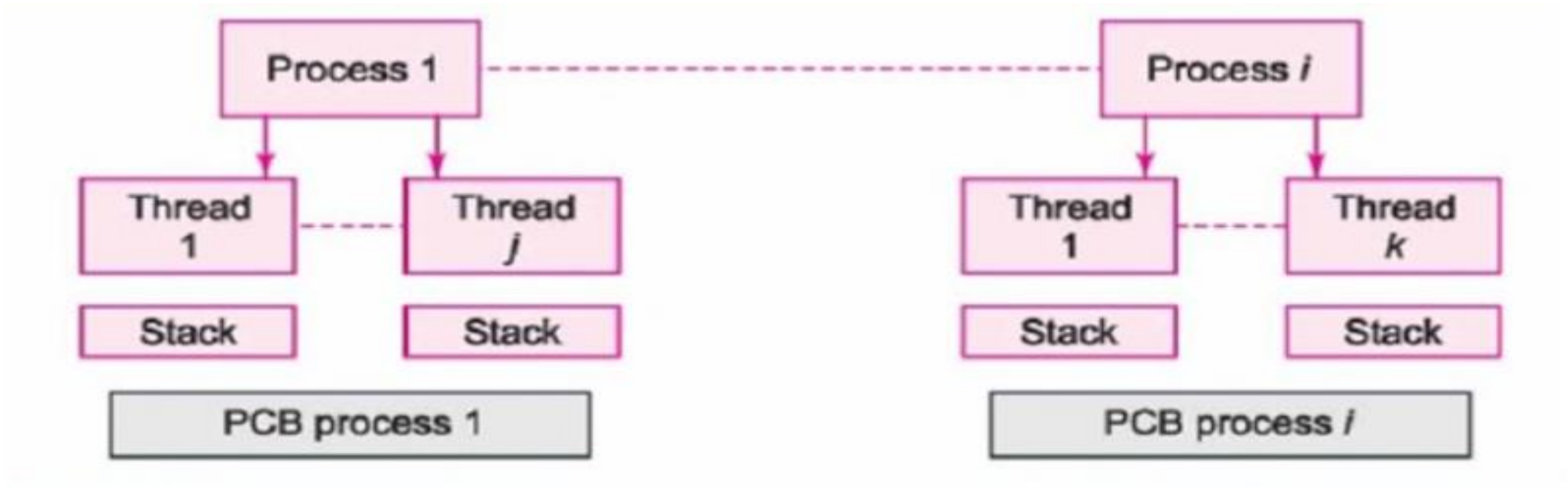
(a) Single threaded



(b) Multithreaded

Thread …

- A thread is a process or subprocess within a process that has its own PC, its own SP and stack, its own priority parameter for its scheduling.
- Thread states─ starting, running, blocked (sleep) and finished.

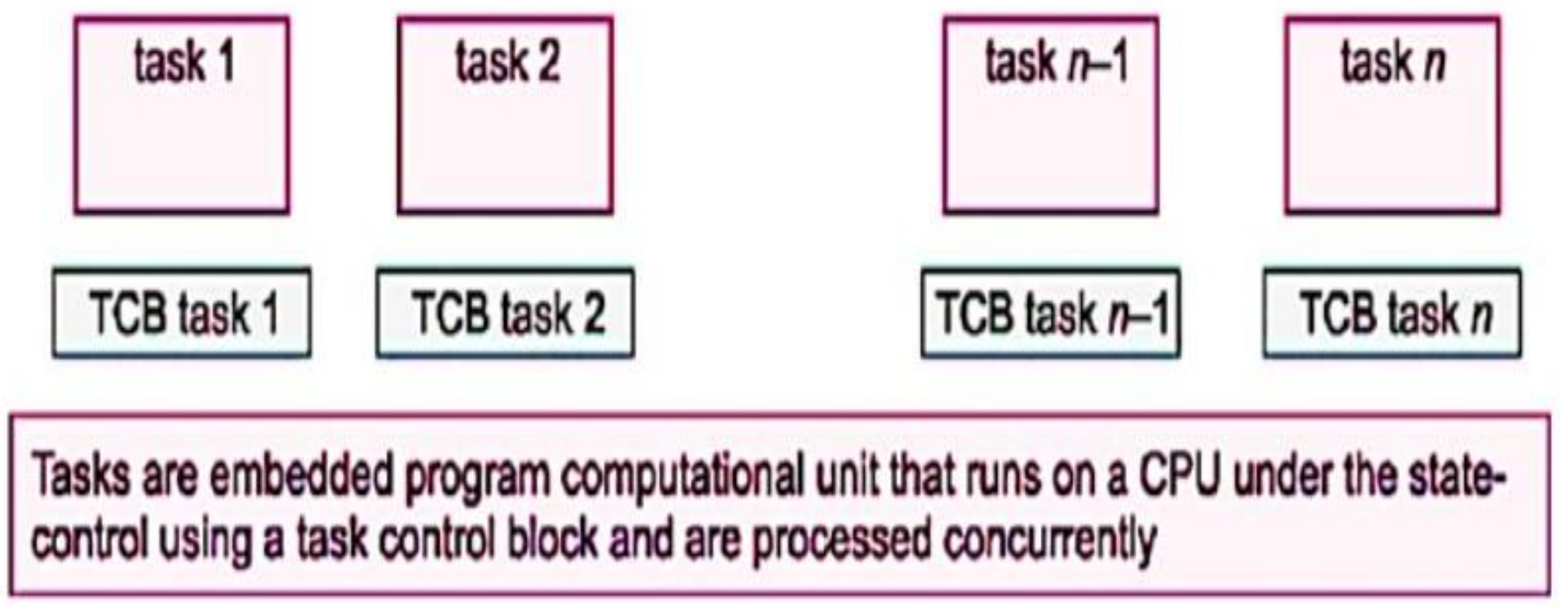


Task

- Task─ term used for the process in the RTOS for the embedded systems. (VxWorks)
- A task is like a process or thread in an OS.
- Runs when it is scheduled to run by the OS (kernel), which gives the control of the CPU on a task request (system call) or a message.
- Runs by executing the instructions and the continuous changes of its state takes place as the program counter (PC) changes
- A task─ an independent process. No task can call another task.

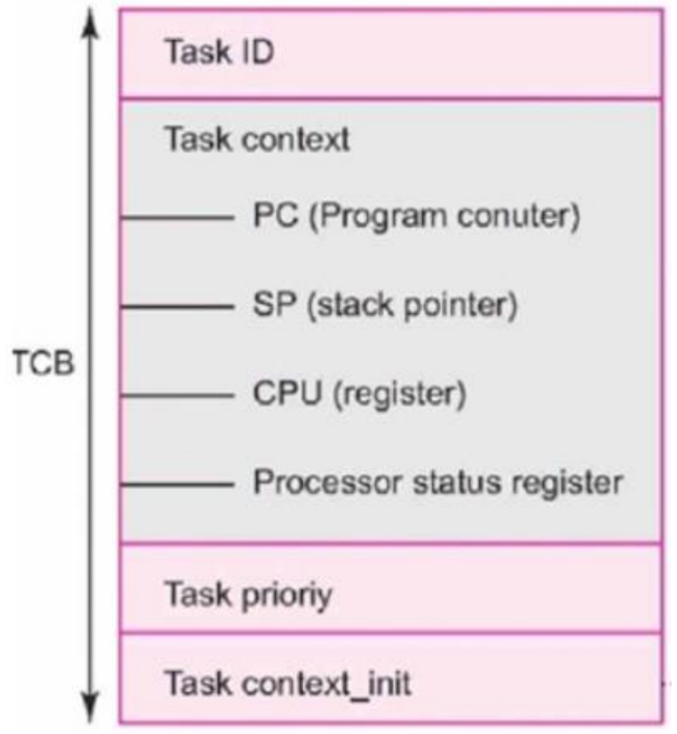
Task…

- Includes task context and TCB
- TCB — A data structure having the information using which the OS controls the process state.
- Stores in protected memory area of the kernel.
- Consists of the information about the task state


task 1
task 2
task *n*–1
task *n*
TCB task 1
TCB task 2
TCB task *n*–1
TCB task *n*
Tasks are embedded program computational unit that runs on a CPU under the state-control using a task control block and are processed concurrently

Task…

- TaskID, e.g. a number between 0 and 255. (index of task)
- task priority, between 0 and 255, represented by a byte
- Each task has its independent values of PC, SP
- Context_init:
  - It has initial parameters of a task



- **Context:**
  - Each task has a context (CPU registers and parameters, which includes registers for the task PC and pointer to the called function)

  - Context continuously updates during the running of a task, and the context is saved before switching occurs to another task

- **Context switch:**
  - Only after saving these registers and pointers does the CPU control switch to any other process or task.

  - Context switching action must happen each time the scheduler blocks one task and runs another task.

Task States

1. Idle state
2. Ready State
3. Running state
4. Blocked (waiting) state

- Number of possible states depends on the RTOS.

Idle state: The task has been created and memory allotted to its structure.

Ready state: The created task is ready and is schedulable by the kernel but not running at present.

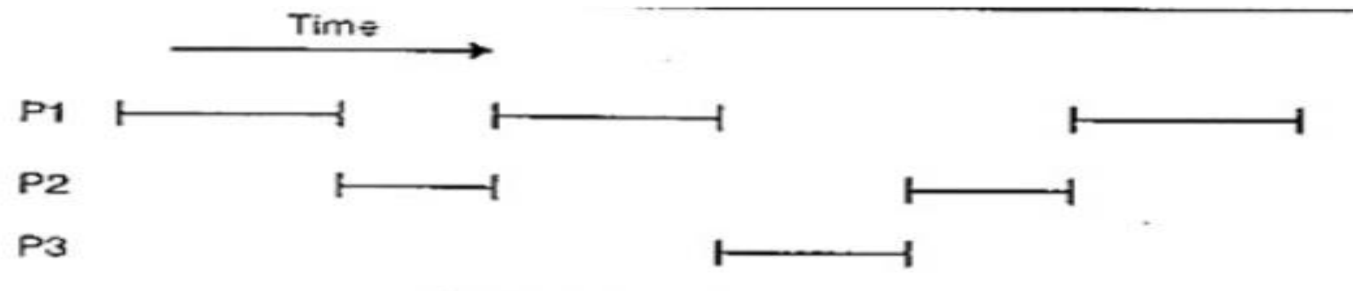
Running state: Executing the codes.

Blocked state: Execution of codes suspends after saving the needed parameters into its context.

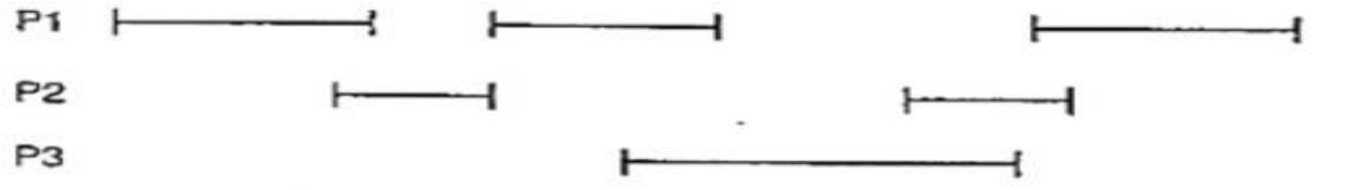
Single vs. Multi-processor

- In single processor multiprogramming system, processes are interleaved in time to yield the appearance of simultaneous execution.

- In multiple processor system, it is possible not only to interleave processes but to overlap them.



(a) Interleaving



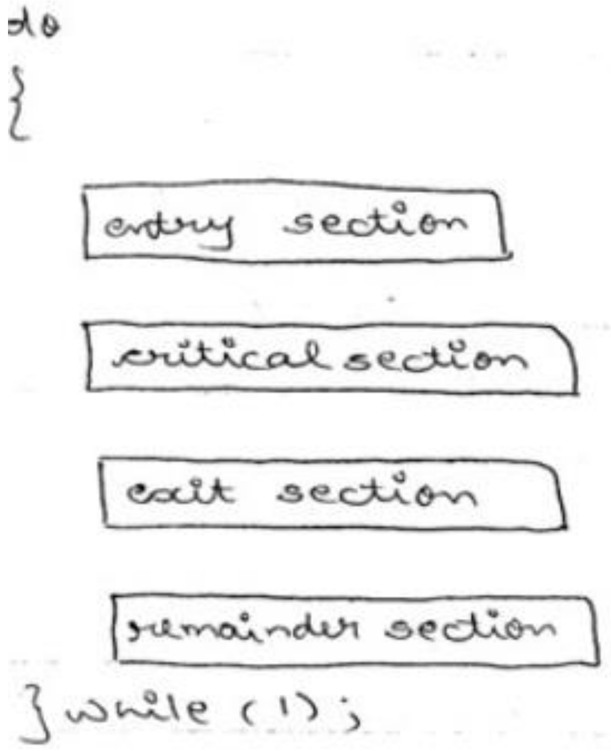
(b) Interleaving and overlapping

Process Cooperation by Sharing (Process Synchronization)

- **Critical Section**

  Each process has a segment of code in which the process may be

-Changing common variables
-Updating a table
-Writing on a file, etc.



Control Requirements to Critical Section

1. **Mutual Exclusion**

   -If a process is inside critical section then another process should not be allowed to enter critical section.

2. **Progress**

   -If there is no process inside critical section AND if a process wants to enter critical section then it should not be stopped by another process.

3. **Bounded Waiting**

   -There exist a limit on the number of times the other process should be allowed to enter the critical section after a process has requested for the same.

All of above three conditions must be satisfied.

Process Cooperation by Communication (Inter-Process Communication)

- IPC is used to generate information about certain sets of computations finishing on one processor and to let other processors waiting for finishing those computations take note of the information.

- IPC means that a process generates some information by setting or resetting a flag or value, or generates an output so that it lets another process take note or use it under the control of an OS.

- Oses provide the software programmer the following IPC functions:

Queue	**to communicate data**
Mailbox/Messages	**”**
Pipes	
Semaphores	**to trigger actions**
Signals	**”**
Remote Procedure Call (RPC)	**for distributed systems**

Semaphore

- Programming language concept proposed for IPC
- Semaphore is a variable upon which three operations are defined:
    1. Count is always initialized to non-negative value
    2. WAIT operation decrements the semaphore value.
    3. SIGNAL operation increments the semaphore.

Semaphore…

- General semaphore (Counting or P-V semaphore)

  -Count can take any integer value

  -Used for process synchronization

- Binary semaphore

  -Value of count 0 or 1, Easier to implement

- Strong semaphore

  -Queue implemented as FIFO queue

- Weak semaphore

  -Queue implemented as priority queue

Signal

- Signal provides the shortest message and it takes shortest possible CPU time.
- The signal is a one bit output from a process for an IPC.
- A signal is the software equivalent of the flag at a register that sets on a hardware interrupt
- When the IPC functions for signal are not provided by an OS, then the OS employs semaphore for the same purpose.

- An important use of signals is to handle exceptions.

- A signal reports an error (‘exception’) during running of a task and then lets the scheduler initiate an error handling process or function.

- Sending a signal is software equivalent of throwing exception in C/C++ or Java program

- Signal is handled only by a very high priority process.

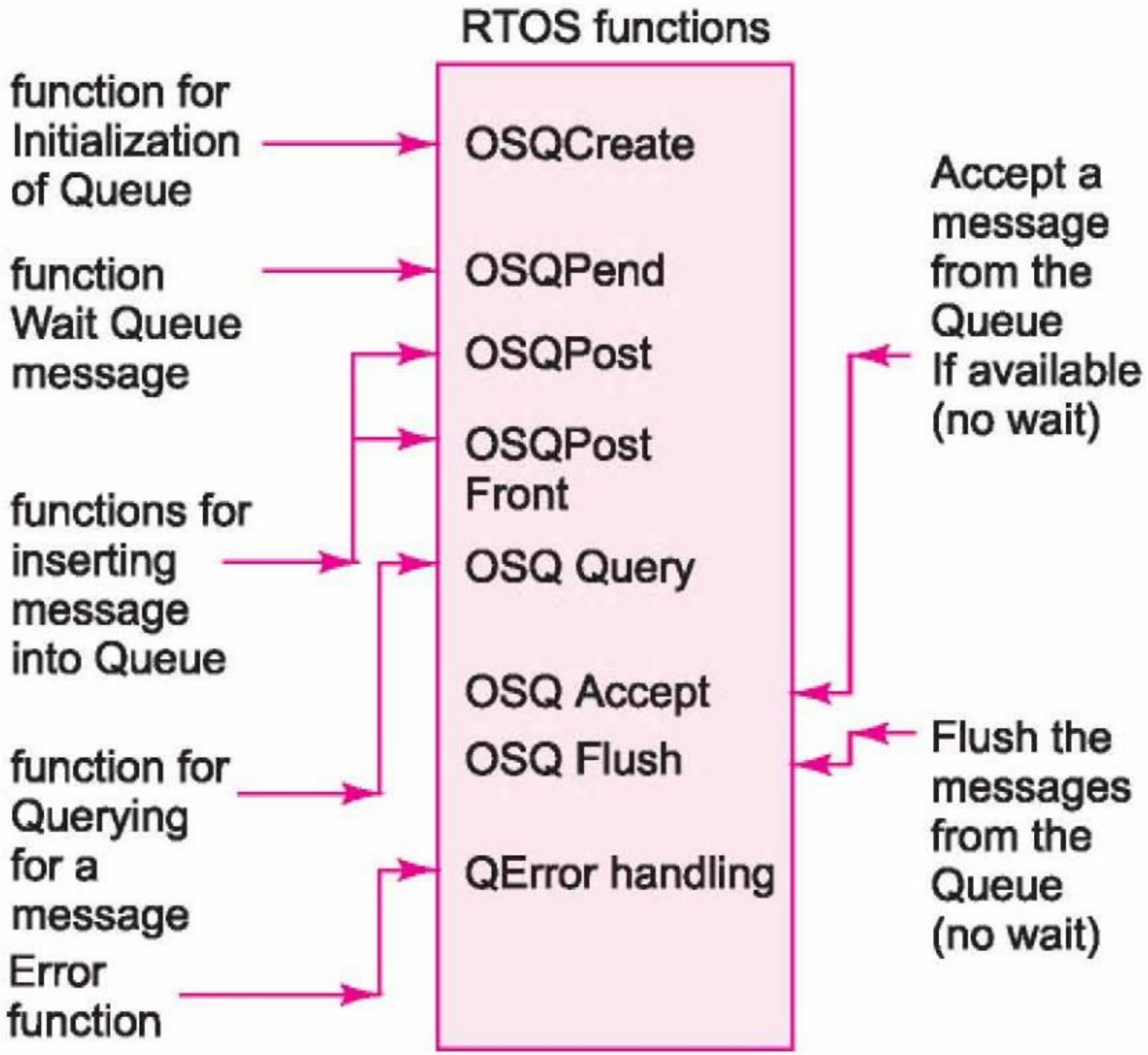
- Queue:

- Each queue for a message may need initialisation before using the functions in the scheduler for the message queue.

- There may be a provision for multiple queues for the multiple types or destinations of messages. Each queue may have an ID.

- Each queue either has a user definable size or a fixed pre defined size assigned by the scheduler.

- When a queue becomes full, there may be a need for error handling and user codes for blocking the task


RTOS functions
function for Initialization of Queue
OSQCreate
function Wait Queue message
OSQPend
OSQPost
OSQPost Front
functions for inserting message into Queue
OSQ Query
OSQ Accept
OSQ Flush
function for Querying for a message
QError handling
Error function
Accept a message from the Queue If available (no wait)
Flush the messages from the Queue (no wait)

- OSQCreate─ to create a queue and initialize the queue message blocks and the contents with front and back as queue-top pointers, *QFRONT and *QBACK, respectively.

- OSQPost ─ to post a message to the message block as per the queue back pointer, *QBACK. (Used by ISRs and tasks)

- OSQPend ─ to wait for a queue message at the queue and reads and deletes that when received. (Used by tasks.)

- OSQAccept ─ to read the present queue front pointer after checking its presence (Used by ISRs and tasks)

- OSQFlush ─ to read queue from front to back, and deletes the queue block, as it is not needed later after the flush

- OSQQuery─ to query the queue message-block when read and but the message is not deleted.

- OSQPostFront ─ to send a message as per the queue front pointer, *QFRONT. Use of this function is made in the following situations. A message is urgent or is of higher priority than all the previously posted message into the queue (Used in ISRs and tasks)

Mailbox

- Mailbox (for message) is an IPC through a message-block at an OS that can be used only by a single destined task.
- Each mailbox usually has one message pointer only, which can point to message.
- OS mailbox functions: Create, Query, Post, Pend, Accept, Delete

- RTOS_BoxCreate: creates a box and initializes the mailbox contents with a NULL pointer

- RTOS_BoxPost : sends a message to the box

- RTOS_BoxPend : waits for a mailbox message, which is read when received

- RTOS_BoxAccept : reads current message pointer after checking the presence. Deletes the mailbox when read and not needed later.

- RTOS_BoxQuery : queries the mailbox when read and not needed or used later.

Pipe

- Pipe is a device used for the inter process communication.
- Pipe has the functions create, connect and delete and functions similar to a device driver (open, write, read, close)
- A message-pipe ─ a device for inserting (writing) and deleting (reading) from that between two given inter-connected tasks or two sets of tasks.
- In a pipe there may be no fixed number of bytes per message with an initial pointer for the back and front.
- A pipe could be used for inserting the byte stream by a process and deleting the bytes from the stream by another process.

- The client makes the call to the function that is local or remote and the server response is either remote or local in the call.
- Both systems work in the peer-to-peer communication mode. Each system in peer-to-peer mode can make an RPC.
- An RPC permits remote invocation of the processes in the distributed systems.

Process management

- Creation of a process means specifying the resources for the process and address spaces (memory blocks) for the created process, stack, data and placing the process initial information at the PCB.
- The process manager allocates a PCB when it creates the process and later manages it.

Management of the created process

- Process manager is a unit of the OS that is the entity responsible for controlling a process execution.
- Process management enables process creation, activation, running, blocking, resumption, deactivation and deletion.
- Process manager executes a process request for a resource or OS service and then grants that request to let the processes share the resources.

Process manager

- Makes it feasible for a process to sequentially (or concurrently) execute when needing a resource and to resume when it becomes available.
- Implements the logical link to the resource manager for resources management.
- Allows specific resources sharing between specified processes only
- Allocates resources as per the resource allocation mechanism of the system
- Manages the processes and resources of the given system

Memory management

- Memory manager of the OS has to be secure, robust and well protected.
- There must be control such that there are no memory leaks and stack overflows.
- Memory leaks means attempts to write in the memory block not allocated to a process or data structure.
- Stack overflow means that the stack exceeds allocated memory blocks when there is no provision for additional stack space.

- Memory manager allocates memory to the processes and manages it with appropriate protection
- There may be static and dynamic allocations of memory
- The manager optimizes the memory needs and memory utilization.

Memory managing strategy for a system

Managing strategy	Explanation
Fixed blocks allocation	Memory address space is divided into blocks with processes having small spaces getting a lesser number of blocks and processes with big address space getting larger number of blocks
Dynamics blocks allocation	Manager allocates variable size blocks dynamically allocated from unused list of memory blocks at different computation phases of a process
Dynamic page allocation	MMU allocates pages (fixed size blocks) dynamically
Dynamic data memory allocation	Manager allocates memory dynamically to different data structures like queues, stack etc.

Memory manager manages:

- Use of memory address space by a process
- Specific mechanisms to share the memory space
- Specific mechanisms to restrict sharing of a given memory space
- Optimization of the access periods of a memory by using an hierarchy of memories(caches, primary and external secondary memories)

Device management

- Device manager coordinates between application process, driver and device controller.
- It manages physical as well as virtual devices like the pipes through a common strategy.

Functions of a device manager

Function	Actions
Device detection and addition	Provides codes for detecting the presence of various devices, then adding them
Device deletion	Provides codes for denying device resources
Device allocation and registration	Allocates and registers port addresses for various devices and also includes codes for detecting any collision between them
Detaching and deregistration	detaches and deregisters port addresses for various devices at distinctly different addresses
Restricting device to a specific process	Access to one process only at an instant

function	actions
Device control	Remote control of the devices from remote server at the service provider
Device access management	Sequential access, random access etc
Device queue management	Queues, circular queues, blocks of queue
Device driver	Software for interface with the device hardware
Backup and restoration	For drivers

- Set of command functions for device management:
    - Create and open
    - Write
    - Read
    - Close and delete

File system organization and implementation

- A file is a named entity on a magnetic disk, optical disk or system memory.
- A file contains data, characters and texts or mix of any of these.
- Each OS may have differing abstractions of a file:
  - A file may be a named entity that is a structured record on a disk having random access in the system.
  - A file may be a structured record on a RAM disk
  - A file may be unstructured record of bits or bytes

Set of command functions

Command in POSIX	Action
open	Creating a file
write	Writing the file
read	Reading the file
close	Closing the file
Set the file pointer	Setting the pointer for the appropriate place in the file for the next read or write

- Windows NT command functions: createFile, ReadFile, WriteFile and SetFilePointer, and CloseHandle.
- A file in Unix has open(), close(), read(), write() functions.
- A file manager creates, opens, reads, seeks a record, writes and closes a file.
- A file has a file descriptor.

Data structure of File descriptor in a typical file system

File descriptor	meaning
Identity	Name by which a file is identified in the application
creator	Program by which it was created
state	‘closed’, ‘saved’, ‘open executing file’ or ‘open file for additions’
Lock and protection fields	O_RDWR, O_RDONLY, O_WRONLY
File info	Current length, when created, when last modified, when last accessed
Sharing permission	Can be shared for execution, reading or writing
count	Number of directories referring to it

I/O subsystems

- I/O ports are the subsystems of OS device management systems
- I/O systems differs in different Oses.
- Two types of IO operations:
  - *Synchronous IO operations* are at certain fixed data transfer rates. A task blocks till completion of the IO
  - *Asynchronous IO operations* are at the variable data transfer rates. It provisions for a process of high priority not blocked during the IOs