Date : / /
Page No.: /

Chapter 4

Introduction To Real-Time Operating System (RTOS).

→ Desired service within a given period of time.
→ RTOS are time driven system and deadline driven.
↳ Bounded response.

General purpose OS (GPOS)	Real time OS (RTOS).
* time insensitive	* time sensitive
* non deterministic (non predictable)	* deterministic (predictable)
* taks will be non prioritized	* they are prioritized
* may not be preemptive	* mostly preemptive
* Virtual memory	* no virtual memory
* General purpose applicatns	* Specific applicatns.
* unbounded response.	* Bounded respons

Types of RTOS:
- → Soft RTOS
- → hard RTOS.

Date : / /
Page No.:

Soft RTOS: Meeting deadline will not lead to undesirable or catastrophic changes but quality reduction may occurs such type of RTOS are called as soft RTOS

Ex: Telecommunicatn applications, media applications.

Hard RTOS: It has to definitely meet deadline

Ex: Nuclear power plant

Key characteristics of RTOS:

→ Reliability: long time service it should give
→ Predictability: deterministic
→ Compactness: as small as possible so that it consumes less memory.
→ Performance: Throughput and response time should be good.
→ Scalable: RTOS should allow to add some other component if required like any network protocol if needed.

Functions of RTOS:

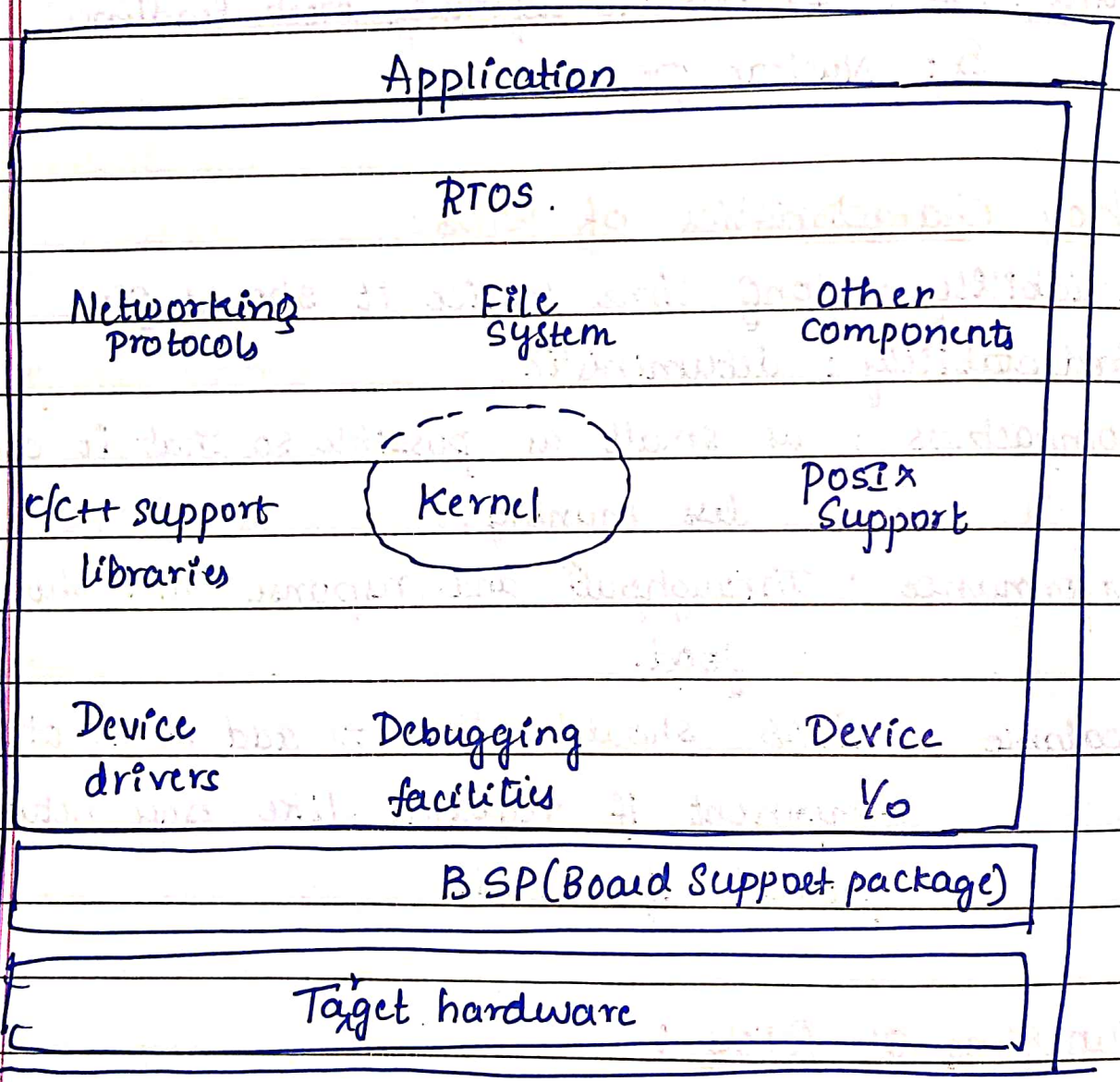
* Task management: managing all tasks
* Scheduling
* Resource management / Resource allocation.
* Interrupt handling:

Date : / /
Page No.:

Applications:

Medical, Telecommunication, defence, security, Air traffic, Automobile.

RTOS



BSP is used to connect some periferals to hardware

→ When we connect peripheral we will in need of some drivers w will be installed in BSP (Board

Date : / /
Page No.:

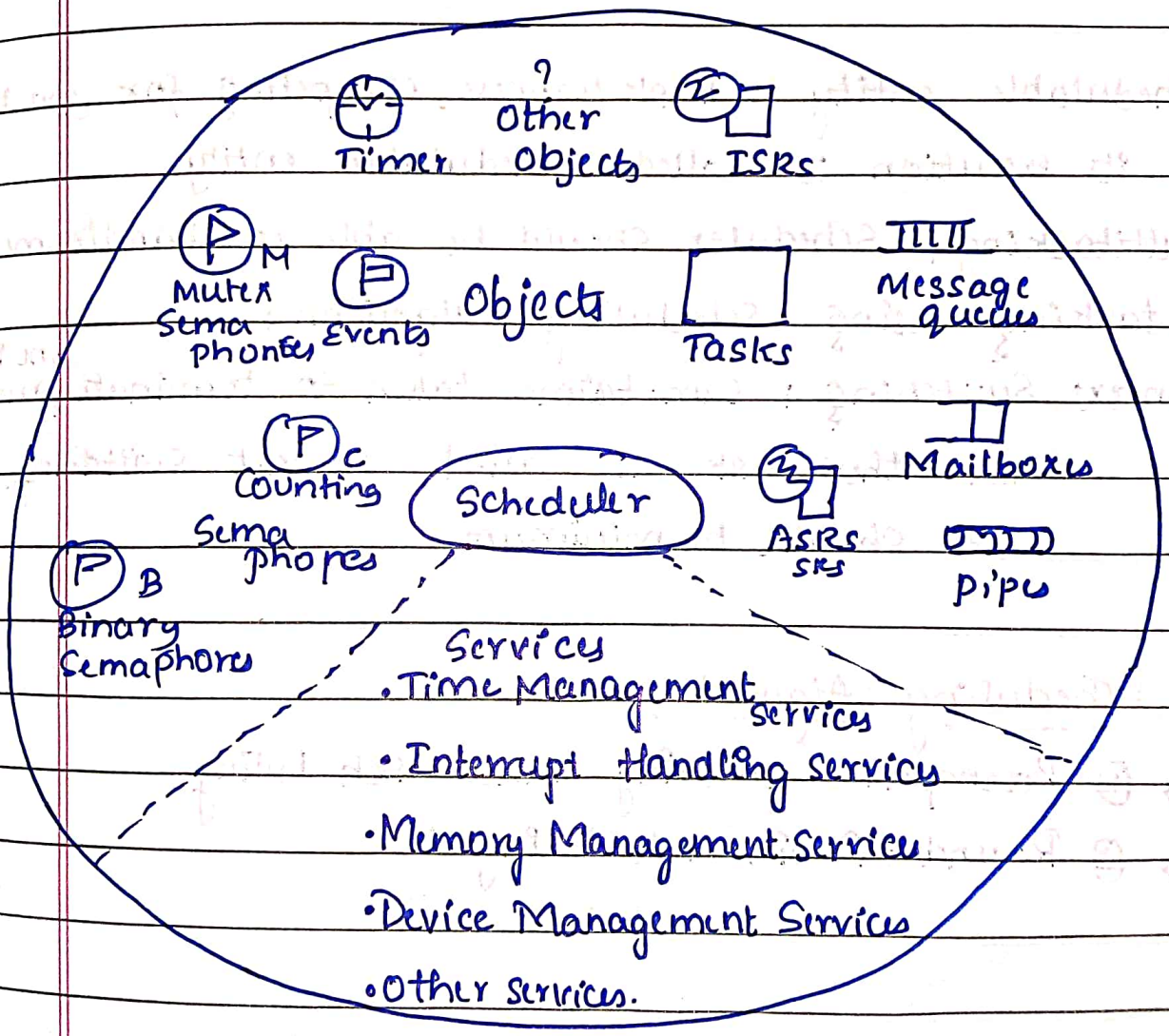
Support package software

Kernel Structure

Kernel is made up of mainly 3 components

* Scheduler * services * objects

→ Scheduler make use of objects and gives services



Date : / /
Page No.:

Scheduler : heart of every kernel.

* Schedulable entity
* provides multitasking
* context Switching
* Dispatching
* make use of Scheduling algorithms

→ Schedulable entity : task / process competing for cpu time for its execution is called Schedulable entity.

→ multitasking : Scheduler should be able to handle multi tasking using Scheduling algorithms.

→ Context Switching : Sam. taime taken to terminate one task and load other task is called context switching It should be minimum.

Scheduling Algorithms

→ ① Preemptive priority-based Scheduling

→ ② Round-Robin scheduling.

Date : / /
Page No.:

RTOS KERNEL OBJECT - TASK

→ By use of objects RTOS gives desired level of services
→ Kernal objects are
① Task / process
② Timer
③ ISR's → Interrupt service routines ⇒ help us to handle Interrupts.
④ Semaphores → for synchronization during multitasking.
⑤ Events / signals
⑥ Queues / pipes
⑦ Mail boxes

TASK

Task : A piece of program ⇒ is meant realizing a operation is called as task.
Task is entity that competes for CPU time for its execution

Every task will to have predefined algorithm / scheduling algorithm..

Date: / /
Page No.:

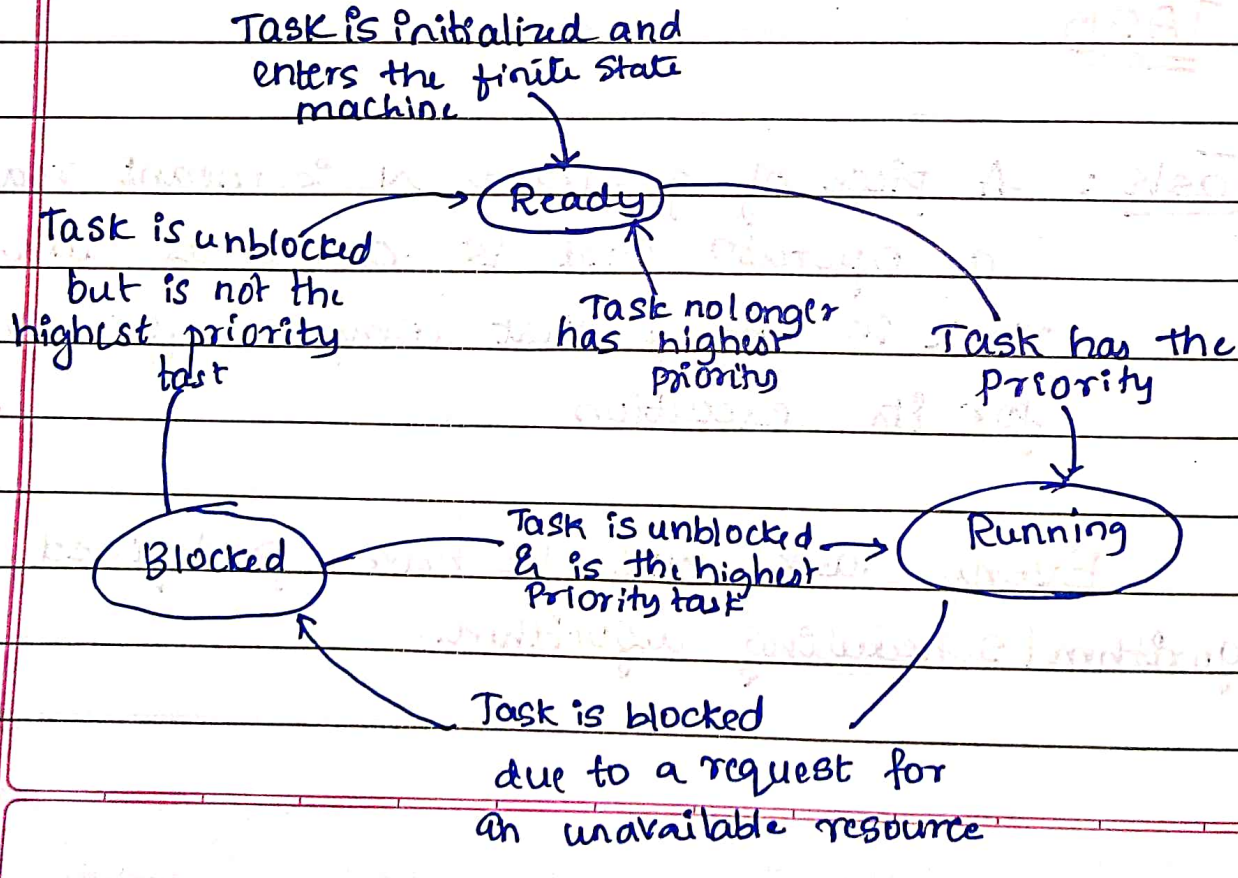
Structure of task:

→ We will have a TCB (Task control block) which is going to store the infor about task like Taskname ID etc. (parameters)

```
Task name/ID
int my_task
{ while(1)
    { =
    }
}
```

→ To story memory which is required to execute a task will be having a stack memory.

TASK STATE DIAGRAM



Date : / /
Page No.:

TYPICAL TASK OPERATIONS.

* Creating & deleting of task
* Scheduling
* Task info

Operations	Description
Suspend	Suspend a task
Resume	Resumes a task
Delay	Delay a task
Restart	Restarts a task
Get priority	Get the current task's priority
Set priority	Dynamically sets a tasks priority.
preemption lock	Locks out higher pri tasks from Preempting current tast
preemption unlock	unlocks preempth lock.

Operath's	Descriptn
Get ID	Get current task's ID
Get TCB	Get the current task's TCB

RTOS KERNEL OBJECT – BASICS OF SEMAPHORE

→ Semaphore : for achieving Synchronizatn, Communicatn and Resource management.
  ↳ act as keytoken for hardware resources whichever task will have this key they will be able to access CPU (resource).

→ 3 types of Semaphore
  ① Binary Semaphore
  ② Counting Semaphore
  ③ Mutex

1] Binary Semaphore : It carries two values [0, 1]
  If it takes 0 : means that resource is unavailable
  If it takes 1 : " " " " "

→ To store the attributes related to Semophore (info like name, ID, tasks) We have SCB (Semaphore control block).

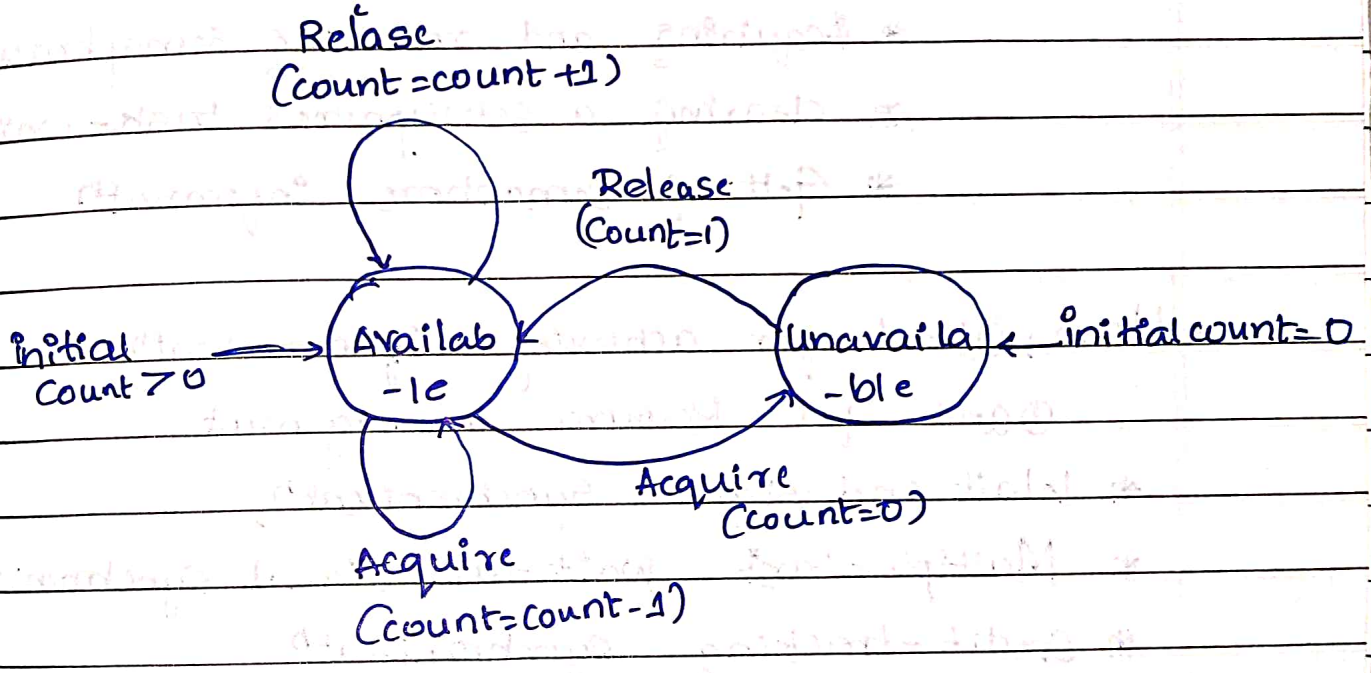
It will have Idname, task weight, list board, Value

Problem of Binary S : no ownership means when someone Access resource any other can release it by deleting taxtsk w may lead to memory leakage.

Date : / /
Page No.:

2] Counting Semaphore : having a count value (integer) How many times Particular Semaphore is being accesd that count value can be maintained (recorded) by this Semaphore.

→ Here also we will have 2 states (available and anavailable).

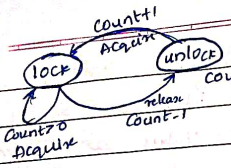


3] Mutex : To avoid problems in previous semophores

↳ 3 main features:

* Ownership is provided
* Task deletion safety
* provides Recurssive access

→ have 2 states : defined as lock and unlock.



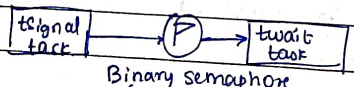
Date : / /
Page No.:

SEMAPHORE OPERATION & USES.

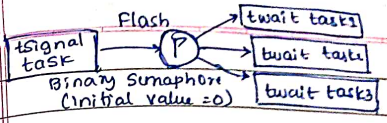
→ Operations: * Creating and deleting semaphores
* Acquiring and releasing semaphores
* clearing a semaphores task-waiting lists
* Getting semaphore information

→ USES: ① Used for achieving Synchronizatn
② and for Resource management.
*. Wait-and-signal Synchronizatn,
* Multiple-task wait-and-Signal Synchronizatn
* Credit-tracking synchronizatn
* Single Shared-resource-access synchronizatn
* Recursive Shared-resource-access synchronizatn
* Multiple Shared-resource-access "

① *



Date : / /
Page No.:



Date : / /
Page No.:

OBJECT - PRIORITY INVERSION.

→ Let us say we have 3 task

Low priority — T_1 | Med pri — T_2 | high priority — T_3

@ one instance of time when let us assume T_1 is in running mode

T_1 utilizes resource called R_1

Now when T_2 comes tries to preempt T_1 as it medium priority and when T_3 comes it also tries to preempt.

When T_2 preempt T_1 then R_1 enters to blocked mode now when T_3 comes it want to execute Though priority is high R_1 has to be released by T_1 so T_3 has to wait due to poor design. This problem is called as priority inversion

→ Solving prority inversion problem.

We use mutex to solve this problem

Mutex solves problem by incorporating priority inheritence and ceiling priority protocol

Date : / /
Page No.:

1. Priority inheritence protocol : means someones priority being inherited

We make T_1 as highest priority so that T_2 cannot prempt T_1 and T_3 can do that its execu

2 Ceiling priority protocol :

highest possible priority among all task will be assigned to runnig low priority task.

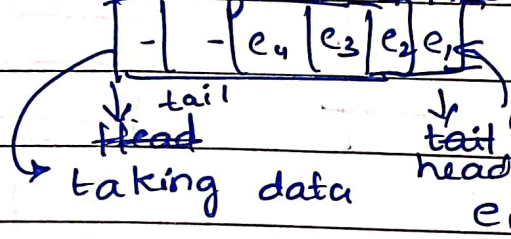
Date : / /
Page No.:

RTOS Kernel Object : Basics of Message Queues

Message Queues :

→ Inter task processes (communicatn) (IPC) :
↳ Message Queue are achieving IPC
Along with message queue we can also use pipes, mail boxes for IPC

→ Message queue is going to carry message.
This Queue is in form FIFO



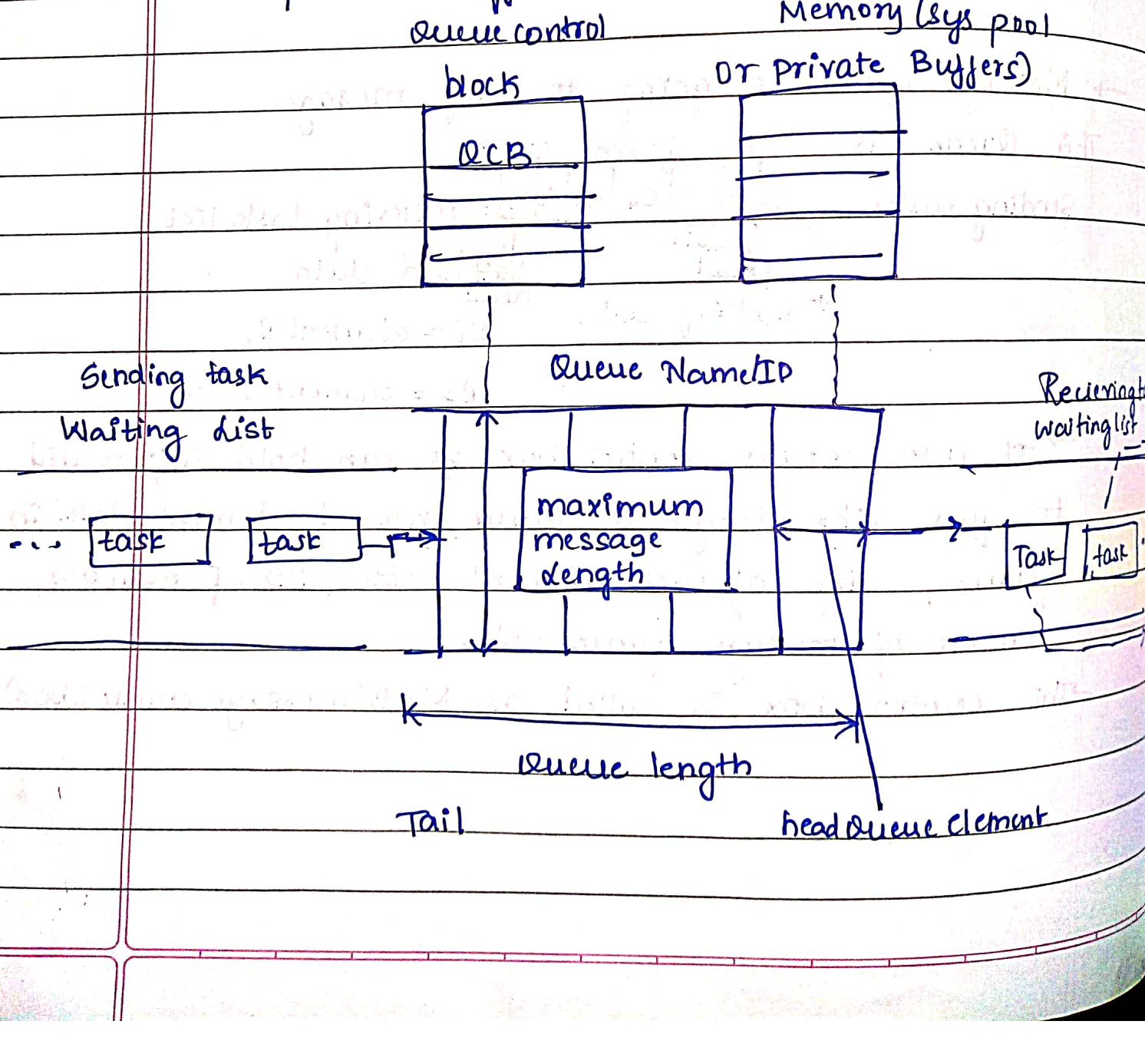
e_1 = element 1,
e_2 = element 2 etc

→ It also creates control box ✓ can hold info related to ques like length of queue, no. of elements(data) in queues, size of each element etc., ID of mail box, name of message queue etc.
This control box is called as MCB(message control box).

Date : / /
Page No.:

→ A message queue is a buffer-like object through which tasks and ISRs send and receive message to communicate and synchronize with data. A message queue is like a pipeline.

→ 2 means of allocating memory to queue:
1. System pool
2. private buffer.



Date : / /
Page No.:

1)* System pools : Using a system pool can be advantageous if it is certain that all message queues will never be filled to capacity @ the same time. The advantage occurs because system pools typically save on memory use.

The downside is that a message queue with large messages can easily use most of the pooled memory, not leaving enough memory for other message queues.

2)* Private Buffers: Using private Buffers, on the other hand, requires enough reserved memory area for the full capacity of every message queue that will be created.

This approach clearly uses up more memory;

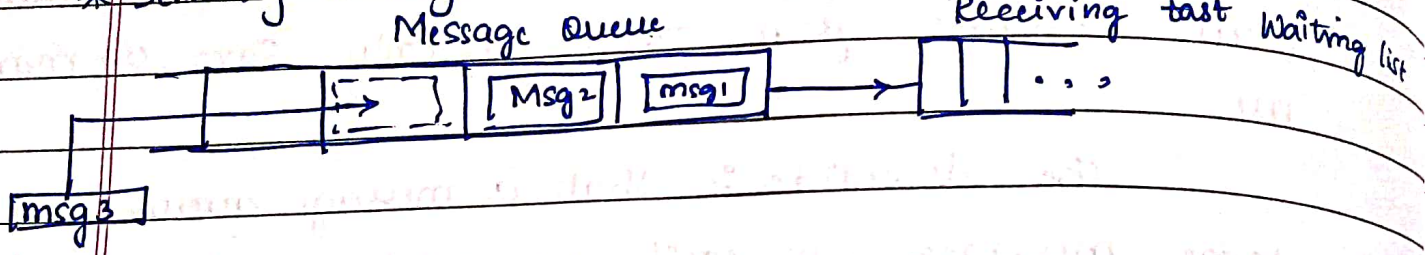
OPERATIONS ON MESSAGE QUEUES

1. Create queues:
2. Deleting queue
3. Sending & Receiving messages:
4. Get info abt message queue.

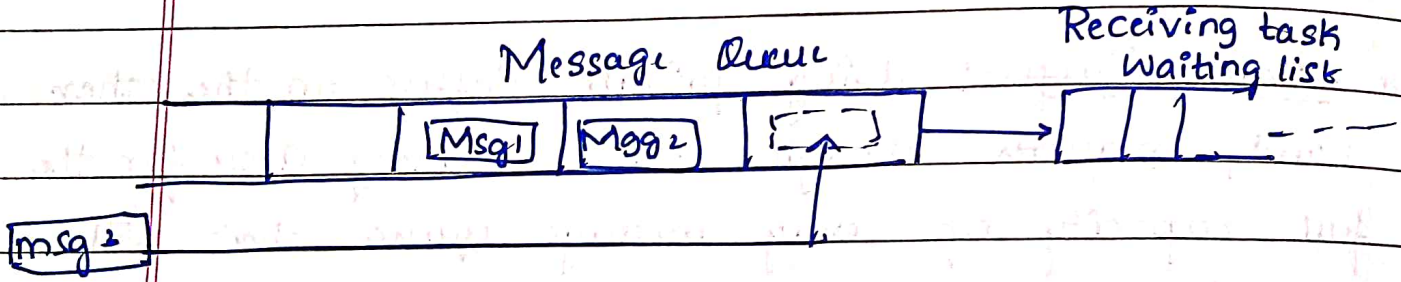
Date : / /
Page No.:

→ Sending & Receiving messages:

* Sending message - First in - First-out (FIFO) order



* Sending message - last-in, first-out (LIFO) order



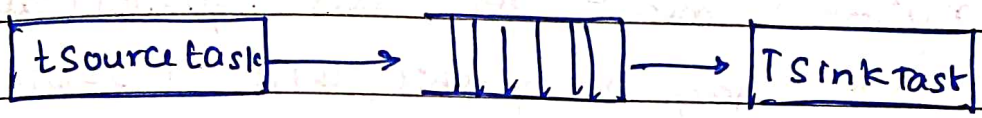
* for Task Waiting list also we have FIFO & priority based order.

Date : / /
Page No.:

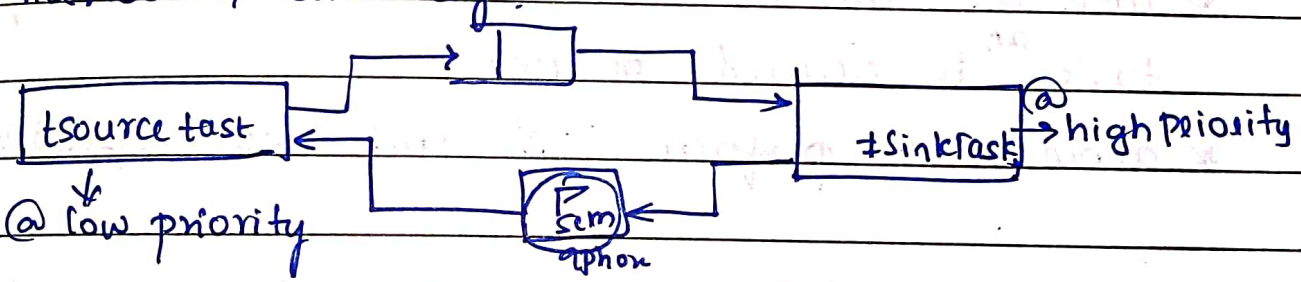
Uses of Message Queues

- Non-interlocked, one-way data communication.
- Interlocked, one-way data communicatn.
- Interlocked, two-way " "
- Broadcast communicatn.

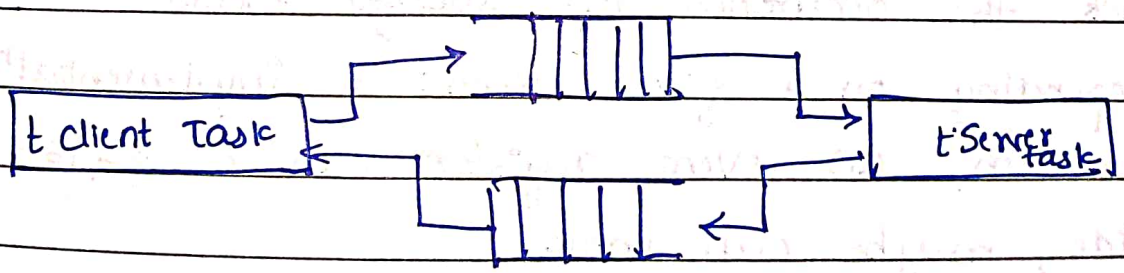
- Non-interlocked, one way



- Interlocked, one way

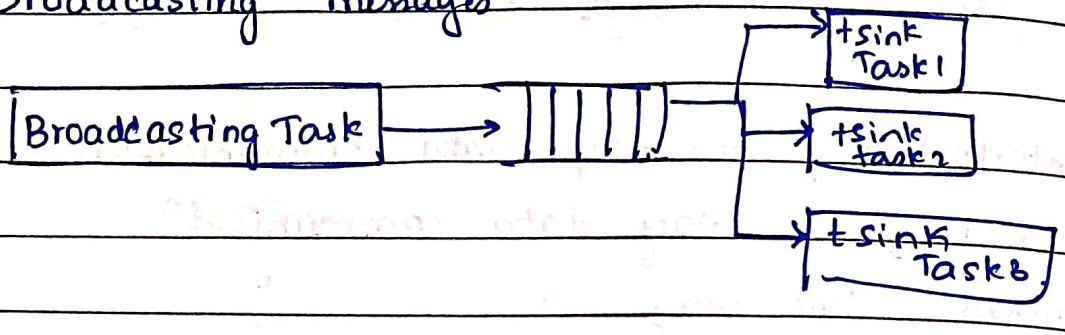


- Interlocked, 2way communicatn.



Date : / /
Page No.:

* Broadcasting messages



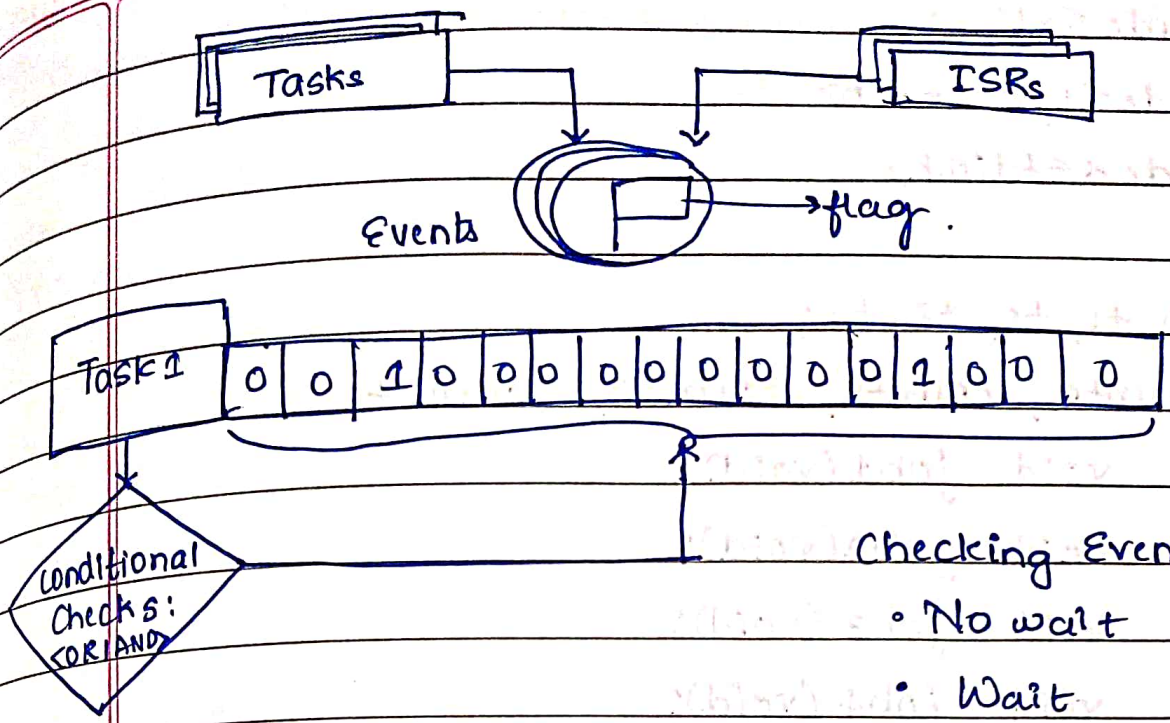
KERNEL OBJECTS - EVENT REGISTERS.

* Event register is meant for task synchronizatn.
* " " may be of 8bit, 16bit or 32bit.
* Each bit will be acting as flag
* Each bit can be set or reset (0 or 1) to say a task can is occured or not
* allow us to perform 'OR' and 'AND' operatns.

* Event register, is an object belonging to a task and consists of a group of binary event flags used to track the occurence of specific event.
* Depending on a given kernel s implementatn of this mechanism, an event register can be 8-, 16-, or 32-bit wide, maybe even more.
* Each bit in this register is treated like a binary flag (also called an event flag) & can be either set or cleared.

Date : / /
Page No.:

Setting Events



Checking Events

- No wait ⊗
- Wait ✓
- Wait with time out