difficult to (30) where it is

MODULE - 1

(3)

Real Time System:-

→ Real Time is a quantitative notion of time and is measured using a physical (real) clock. whenever we quantity time using a physical clock, we deal with real time.

→ A system is called a real time system, when we need quantitative expression of time to describe the behaviour of the system.

APLICATIONS OF REAL TIME SYSTEM:-

Real time systems have many applications in wide-ranging areas.

① Industrial Application:-

Exa. process control system, industrial automation systems, SCADA applications, test & measurement equipment and robotic equipment

SCADA:- supervisory control and Data Acquisition

② Medical :-

Exa :- Robots, MRI scanners, Radiation therapy equipment, bedside monitors, and computerized Axial Tomography - (CAT).

③ Peripheral equipment:-

Exa :- Laser printers, digital copiers, Fax machine digital cameras, and scanners.

④ Automotive and Transporation:-

Exa: Automotive engine control systems, road traffic signal control, air-traffic control, high speed train control, car navigation systems.

⑤ Telecommunication Applications:-

Exa cellular systems, video conferencing and cable modems.

⑥ Aerospace:-

Exa avionics, Height simulation, airline cabin management systems, satellite tracking systems, and computer on-board an aircraft

⑦ Internet and multimedia applications:-

Exa:- video conferencing and multimedia multicast, Internet routers and switches.

⑧ Consumer Electronics:-

Exa:- set-top boxes, audio equipment, Internet telephony, microwave ovens, home security systems, air conditioning and refrigeration.

⑨ Defence Applications:-

Exa:- missile guidance systems, anti-missile systems, satellite-based surveillance systems.

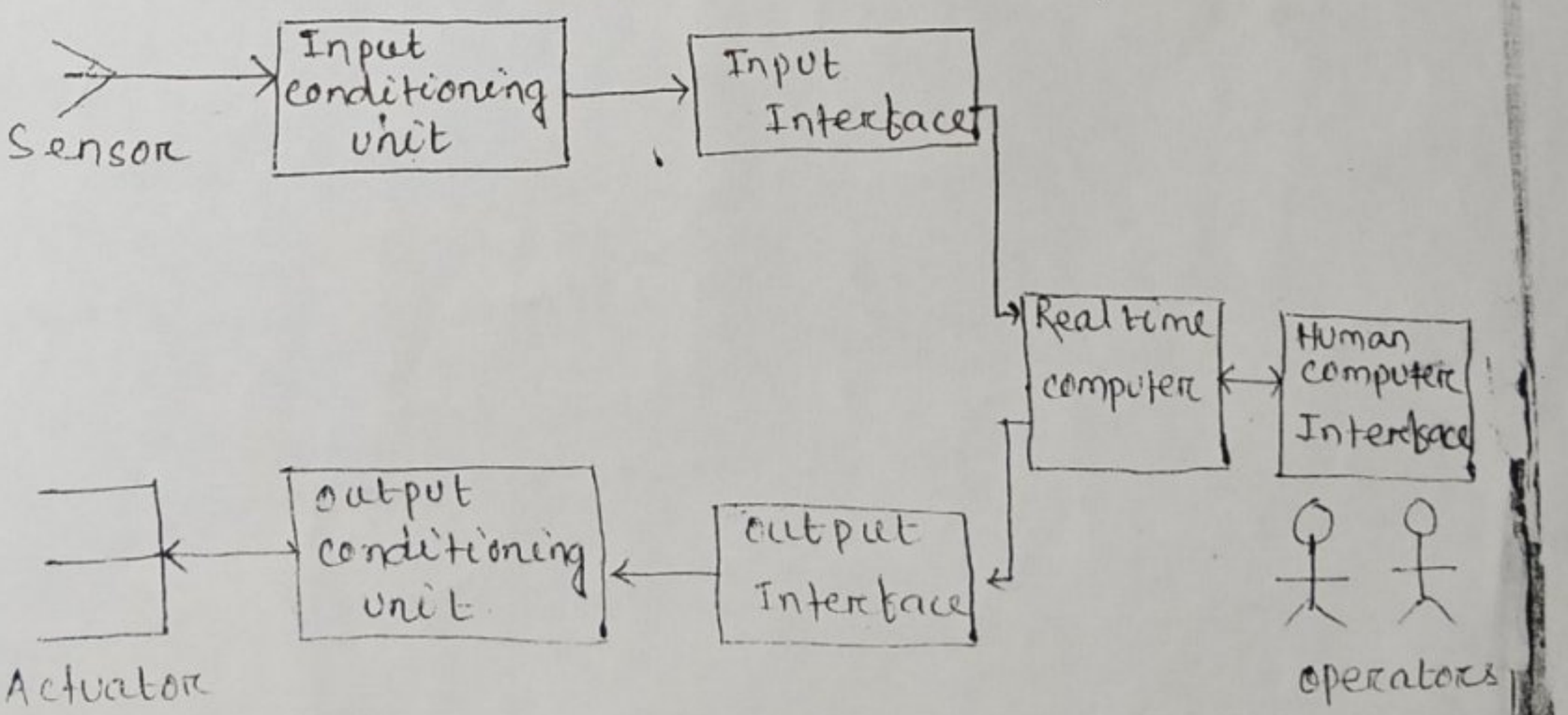
⑩ Miscellaneous Applications:

Exa railway reservation system.

3 Basic Model of a Real Time system:-

The important Functional Blocks of a Real time systems are:-

1. Sensor.
2. ~~Acuta~~ Actuator
3. Signal conditioning units.
4. Interface Unit.



(Fig: Model of a Real time System)

1. Sensor:-

→ A sensor converts some physical characteristics of its environment into electrical signals.
→ An example of a sensor is a photo-voltaic cell which converts light energy into electrical energy.
→ Different temperature and pressure sensors are also used.

A temperature sensor operates on the principle of a thermocouple. It based on other physical principles.

A pressure sensor operates on the piezoelectricity principle.

2. Actuator:-

→ An actuator is any device that takes its inputs from the output interface of a computer & converts these electrical signals into some physical actions on its environment.
→ The physical actions may be in the form of motion, change of thermal, electrical, or physical characteristics of some objects.
→ Exa:- A popular actuator is a Motor.

3. Signal Conditioning Unit:-

→ The electrical signal produced by a computer can be used to directly drive an actuator. The computer signals usually need conditioning before they can be used by the actuator. This is called as output conditioning.
→ similarly, input conditioning is required to be carried out on sensor signals before they can be accepted by the computer.
→ For example, analog signals generated by a photo-voltaic cell are normally in the millivolts range and need to be conditioned before they can be processed by a computer.

uman
omputer
nterface

perators

4. Interface Unit:- 6

→ Normally, commands from the CPU are delivered to the actuator through an output interface.

→ An output Interface converts the stored voltage into analog form and then outputs this to the actuator circuitry. This would require the value generated to be written on a register..

→ In order to produce an analog output, in an output interface, the cpu selects a data register of the output interface and writes the necessary data to it. The interface takes care of the buffering & the handshake control aspects.

Characteristics Of Real Time Systems:-

Different characteristics of Real time systems are:-

→ Time constraints
→ New correctness criterion.
→ Embedded.
→ Safety criticality
→ concurrency.
→ Distributed & Feedback structure
→ Task criticality.
→ Custom Hardware
→ Reactive
→ stability.
→ Exception Handling.

SAFETY AND RELIABILITY:-

Fail-safe state: A fail-safe of a system is one which if entered when the system fails, no damage would result.

→ For example, the fail-safe state of a word processing program is one where the document being processed has been saved onto the disk.

Safety-critical systems:-

↑ safety critical system is one whose failure can cause severe damages.

→ Exa, a navigation system on-board an aircraft. It has no fail-safe states when the computer on-board an aircraft fails, a fail-safe state may not be one where the engine is switched off.

→ In a safety-critical system; the absence of fail-safe state ~~may not be one where the engine is~~ implies that safety can only be ensured through increased reliability.

How to Achieve High Reliability?

High Reliable software can be developed by adopting all the following three important techniques.

① Error Avoidance:

For achieving high reliability, the errors that occure should be minimized during product Development. This can be achieved by adopting suitable CASE tools (Computer Aided Software Engineering).

② Error Detection and Removal:-

Many errors that are appear need to be detected and removed. This can be achieved by conducting through reviews & testing

(3) Fault - Tolerance :-

→ To achieve high reliability, even in situations where errors are present, the system should be able to tolerate the faults and compute the correct results. This is called Fault Tolerance.

→ It can be achieved by incorporating redundancy.

Two methods used to achieve hardware fault-tolerance:

① BIST (Built-in Self Test);-

② TMR (Triple Modular Redundancy):

Software Fault - Tolerance Techniques:-

There are two methods used to achieve software fault - tolerance.

① N - version programming:-

In the N-version programming technique, independent teams develop N different versions of a software component.

→ The redundant modules are run concurrently. The results produced by the different versions of the module are subjected to voting at run time and the result on which majority of the components agree is accepted.

→ This scheme is not very successful in achieving fault-tolerance and the problem can be attributed to statistical correlation of failures. This means that even though individual teams worked in isolation to develop the different version of a software component, even then the different versions fail for identical reasons.

② Recovery Block - Techniques:

In this scheme, the redundant components are called try blocks. Each try block computes the same end result as the others but is intentionally written using a different algorithm compared to the other try blocks.

Difference bet'n N version programming and Recovery Block techniques:

N-Version Programming:	Recovery Block Techniques:
→ In N-version programming the different versions of component are written by different teams of programmers.	→ In Recovery block techniques different algorithms are used in different try blocks.
→ The redundant copies are run concurrently	→ The redundant copies are run one after another

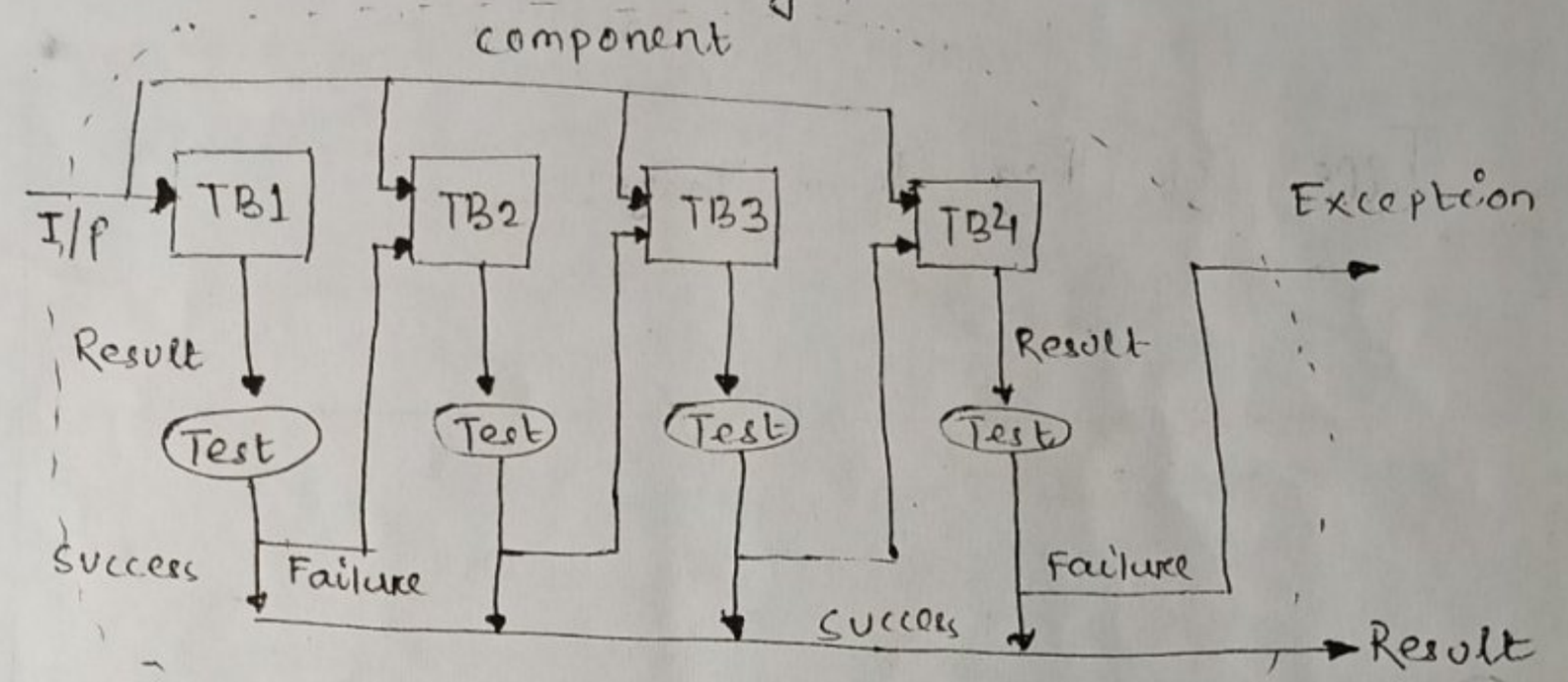
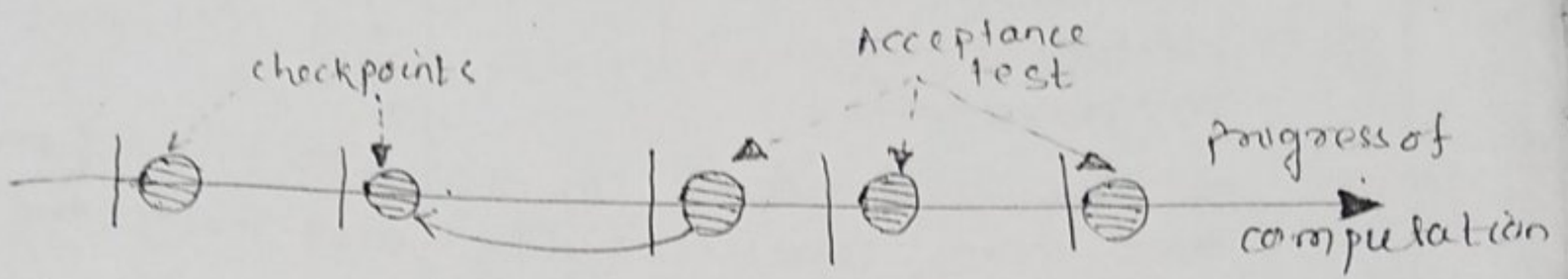


Fig:- (A Software Fault-Tolerance Scheme Using Recovery Blocks)

Checkpointing and Roll-back Recovery:-

→ It is another popular technique to achieve fault-tolerance.
→ In this technique as the computation proceeds the system state is tested each time after some meaningful progress in computation is made.
→ Immediately after a ~~te~~ state-check test succeeds, the state of the system is backed up on a stable storage.
→ In case the next test does not succeed, the system can be made to roll back to the last checkpointed state. After a roll-back, from a checkpointed state a fresh computation can be initiated.

→ This technique 10 is useful, if there is a chance that the system state may be corrupted as the computation proceeds, such as data corruption or processor failure.



(Fig: checkpointing and Roll-back Recovery)

Types of Real Time Tasks:-

A Real Time Task can be classified into 3 broad categories depending on the consequences of a task missing its deadline.

① Hard Real Time Tasks.
② Soft Real Time Tasks.
③ Firm Real Time Tasks.

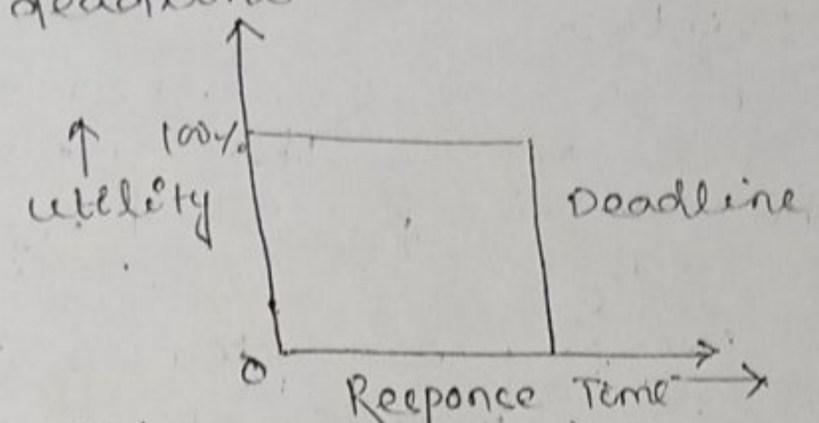
① Hard Real Time Task:-

→ A hard Real Time Task is one that is constrained to produce its results within certain predetermined time bounds.
→ The system is considered to have failed whenever any of its hard real time tasks does not produced its required results before the specified time bound.
→ Exa - Robot, anti-missile system.
→ Applications having hard real time tasks are safety-critical. This means that any failure of a real time task, including its failure to meet the associated deadlines. This makes hard real time tasks extremely critical.

② Firm Real Time Tasks:-

→ Every Firm Real Time Task is associated with some predefined deadline before which it is required to produce its results

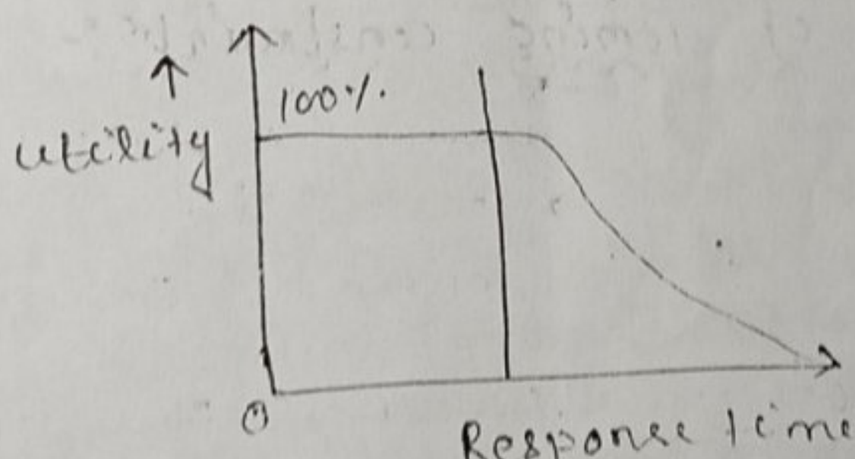
→ A Firm real time ... within its deadline, the system does not fail. The late results are discarded.
→ The utility of the results computed by a firm real time task becomes zero after the deadline.



(Utility of Result of a Firm Real-time task with time)

→ Exa :- Video conferencing.
satellite. Based Tracking of Enemy Movements.

③ Soft Real Time Tasks :-
→ Soft Real Time Tasks also have time bounds associated with them.
→ The timing constraints on soft real time tasks are not expressed as absolute values. Instead the constraints are expressed in terms of the average response times required.
→ Exa - Web browsing.



(Fig: Utility of the results produced by a soft real time task as a function of time).

Non-Real Time Task :- It is not associated with any time bounds.
Exa - batch processing jobs, e-mail & background tasks such as event loggers.

Timing Constraints :-

Events in a Real Time System :-

An event may be generated either by the system or its environment. Based on this, events can be classified into two types.

1. Stimulus Events :-

→ Stimulus Events are generated by the environment and act on the system.

→ These events can be produced asynchronously (i.e aperiodically).

→ Exa :- A user pressing a button on a telephone set generates a stimulus event to act on the telephone system.

Events may be generated by the system themselves on the enviroment of the task of the

2. Response Events :-

→ Response Events act on the environment and are usually produced by the system in response to some stimulus events.

→ Response events can either be periodic or aperiodic.

→ Exa :- Consider a chemical plant where as soon as the temperature exceeds 100°c, the system responds by switching off the heater.

Classification of Timing constraints :-

The classification of Timing constraints help us quickly identify the different timing constraints that can exist from a casual examination of a problem.

→ So Timing Constraints can be classified into ① Performance constraint and ② Behavioural constraints.

① Performance constraint

→ Performance constraints are the constraints that are imposed on the response of the system.

→ It ensures that the computer system performs satisfactorily

② Behavioural constraint

→ B.C are the constraints that are imposed on the stimuli generated by the environment.

→ It ensures that the environment of a system is well behaved.

Each of the constraints are further classified into the following 3 types:-

1. Delay constraint.
2. Deadline constraint.
3. Duration constraint.

1. Delay Constraint :-

→ A Delay Constraints captures the minimum time that must elapse between the occurrence of two arbitrary events e_1 & e_2.

→ After e_1 occures, if e_2 occures earlier than the minimum delay, then a delay violation is said to occure.

→ A Delay constraint on the event e_2 can be expressed $\boxed{t(e_2) - t(e_1) \geq d}$

where $t(e_2)$ and $t(e_1)$ are the time stamps on the events e_2 and e_1. d is the minimum delay specified from e_2.

2. Deadline Constraint :-

→ A Deadline constraint captures the maximum separation between any two arbitrary events e_1 and e_2.

→ A Deadline constraint may be expressed as

$$\boxed{t(e_2) - t(e_1) \leq d}$$

3. Duration Constraint :-

→ A Duration constraint on an event specifies the time period over which the event acts.

→ It may be minimum type or maximum type

→ The minimum type duration constraint require that once the event starts, the event must not end before a certain minimum duration. where as a maximum type duration constra requires that once the event starts, the event must end before a certain maximum duration elapses.

Deadline constraints :-

1) Stimulus - stimulus (SS) :- In this case, the deadline is defined between two stimuli.
→ This is a behavioural constraint, since the constraint is imposed on the second event which is stimulus.
→ Exa:- Once a user completes dialling a digit, he must dial the next digit within the next 5 sec. otherwise, an idle tone is produced.

2) Stimulus - Response (SR) :-
→ In this case, the deadline is defined on the Response event, measured from the occurrence of the corresponding stimulus event.
→ This is a performance constraint, since the constraint is imposed on a response event.
→ Exa :- Once the receiver of the handset is lifted, the dial tone must be produced by the system within a sec. otherwise a beeping sound is produced until the handset is replaced.

3) Response - stimulus (RS) :-
→ Here the deadline is on the production of response counted from the corresponding stimulus. This is a behavioural constraint, since the constraint is imposed on the stimulus event
→ Exa :- Once the dial tone appears, the first digit must be dialled within 30 sec, otherwise the system enters an idle state and an idle tone is produced.

4) Response - Response (RR) :-
→ It is defined on two response events.
→ In this case, once the first response event occurs, the second response event must occure before a certain deadline.
→ This is a performance constraint, since

response event.

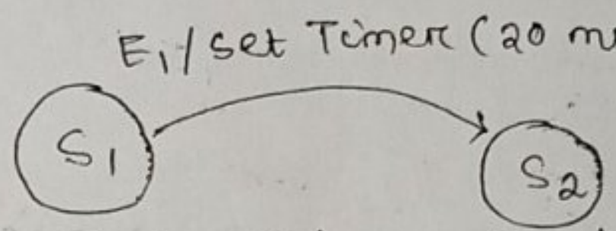
→ Exa:- Once the ring tone is given to the callee, the corresponding ring back tone must be given to the caller within 15 sec, otherwise the call is terminated.

Modelling Timing Constraint:-

→ For modelling a Timing constraint, we use EFSM (Extended Finite state Machine).

→ It extends the traditional FSM, by incoreporating the action of setting a timer and the expiry event of a timer.

→ Exa



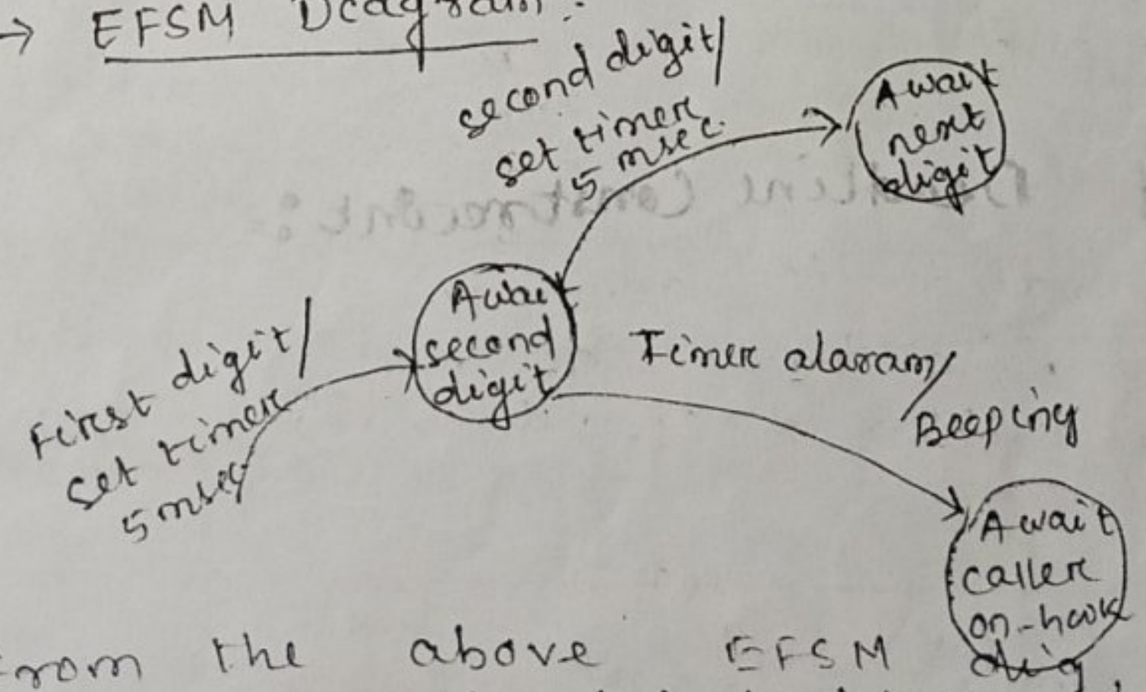
From the above Fig, it describes that, if an event e1 occurs when the currecent state of the system is S_1, then an action will be taken by setting a timer to expire in the next 20msec. and the system transits to state S_2.

Examples

① SS type of Deadline constraint:-

→ Once, the first digit has been dialled on the telephone handset, the next digit must be dialled within the next 5 msec.

→ EFSM Diagram:-



From the above EFSM Fig it is observe that as soon as, the first digit is dialled, the system enters the "Await second digit" state and the timer is set to ~~20~~ 5 msec. If the next digit does not appear within 5 msec, then the timer alaram expires & the system enters the Await caller on-hook state & a beeping sound is produced. If the second digit occurs before 5 msec

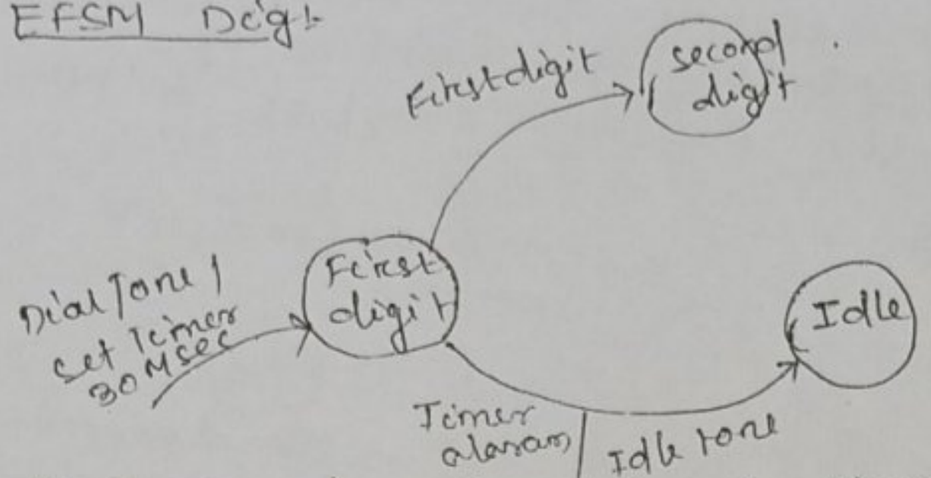
then the system transits to await next digit state.

Example 2

RS Types of Deadline Constraint :-

Once the dial tone appears, the first digit must be dialled within 30 sec. otherwise the system enters an idle state and an idle tone is produced.

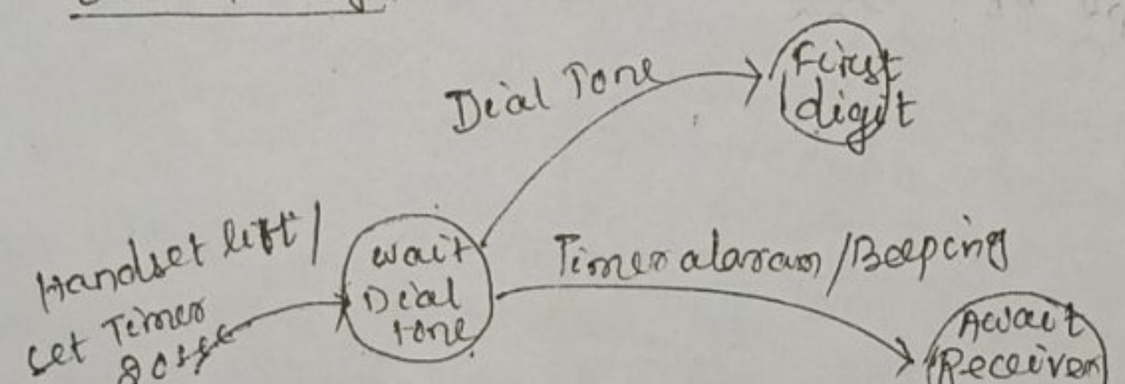
→ EFSM Dig :-



S-R Type of Deadline constraint :-

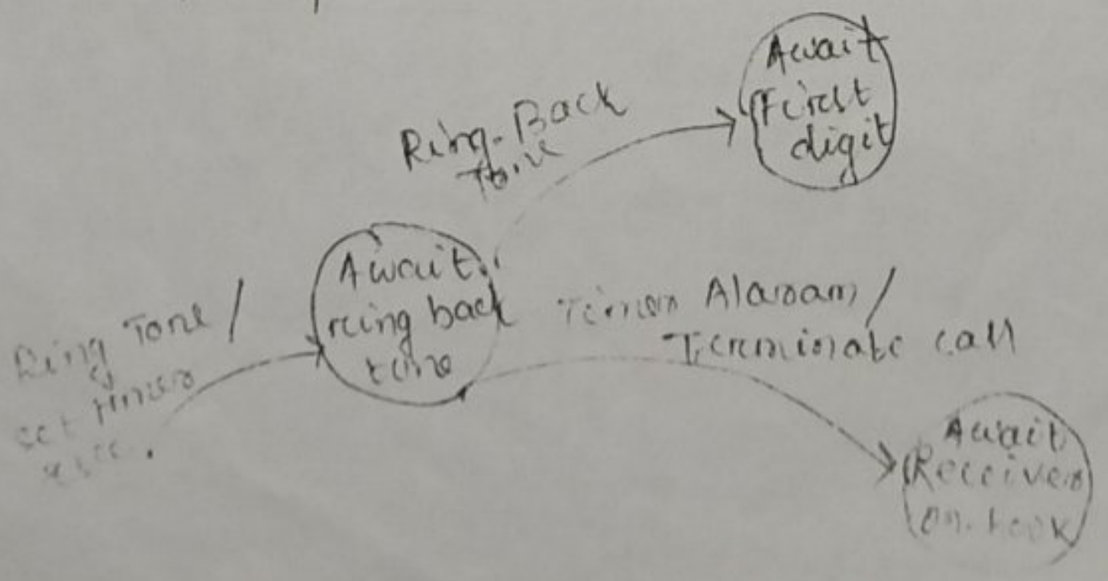
Once the receiver of the handset is lifted, the dial tone must be produced by the system within 20 sec, otherwise a beeping sound is produced until the handset is replaced.

EFSM Dig :-



RR Type of Deadline Constraint :-

Once the ringtone is given to the caller, the corresponding ring back tone must be given to the caller within 2 sec, otherwise the call is terminated.



CHAPTER-2

Real Time Task Scheduling :-

Task Instance :- Each time an event occurs, it triggers the task that handles this event to run. In other words, a task is generated when some specific event occurs.

Response Time :- The Response time is the time duration from the occurrence of the event generating the task to the time the task produces its results.

Task Precedence :- A task is said to precede another task, if the first task must complete before the second task can start.

→ When a Task T_i precedes another task T_j, then each instance of T_i precedes the corresponding instance of T_j i.e. if T_1 precedes T_2, Then $T_1(1)$ precedes $T_2(1)$, $T_1(2)$ precedes $T_2(2)$ & so on.

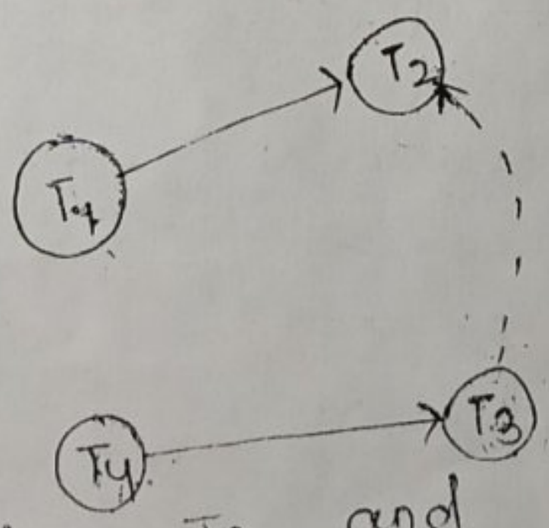
Data sharing :-

The precedence relation between two tasks sometimes called data sharing i.e. first task passing some results to the second task.

→ Data sharing may occure not only when one task precedes the other, but might occure among truly concurrent tasks and overlapping tasks.

→ we represent data sharing among two tasks using a dashed arrow.

→



on this fig. T_2 uses the results of T_3, but T_2 & T_3 may execute concurrently. T_2 may even start executing first, after sometimes it may receive some data from T_3, and continue its execution & so on.

Relative Deadline

→ Relative Deadline is the time interval between the start of the task & the instant at which the deadline occurs.
→ OR. Relative Deadline is the time interval between the arrival of a task & the corresponding Deadline.

Absolute Deadline

→ The absolute Deadline of a task is the absolute time value (counted from time 0) by which the results from the task are expected. Thus Absolute Deadline is equal to the interval of time between the time '0' & the actual instant at which the deadline occurs as measured by some physical clock.

Types of Real Time Task :- [Based on the way realtime task occur over a period of time]

The Real Time Tasks are classified into 3 catagories. ① Periodic ② sporadic ③ aperiodic

① Periodic Task :-
→ A periodic task is one that repeats after a certain fixed time interval
→ A periodic task is sometimes called as clock-driven tasks.
→ The fixed time interval after which a task repeats is called the period of the task.
→ Formally, a periodic Task Ti can be represented by a four tuple (ϕ_i, P_i, e_i, d_i) where
P_i - is the period of task.
e_i - is the worst-case execution time of the task.
d_i - is the relatively deadline of the task.
and if Ti is a periodic Task, then the time from '0' till the occurrence of the first instance of Ti is denoted by ϕ_i.
Eg: In a rocket control software, the track correction task starts 2000 msec. after the launch of the rocket, & it periodically recurs every 50 msec then on each instance of the task requires a processing time of 8 msec & its relative deadline is 50 msec.

(19)

so this task can be formally represented as (20ms, 50 msec, 8 msec, 50 msec).

→ when the deadline of a task equals it's period i.e ($P_i = d_i$) we can omit the lowest tuple. i.e if (20ms, 50 msec, 8 msec). this would automatically $P_i = d_i = 50$ msec.

→ when $\phi_i = 0$, T_i = (20 msec, 100 msec). which indicate a task with $\phi_i = 0$, $P_i = 100$ msec, $e_i = 20$ msec, and $d_i = 100$ msec.

② Sporadic Task:-

→ A sporadic task is one that recurs (to occur again or to recover) at random instants.

→ A sporadic task T_i can be represented by a three tuple: $T_i = (e_i, g_i, d_i)$.

where e_i - it's the worst case execution time of an instance of the task, g_i denotes the minimum separation between two consecutive instances of the task implies that once an instance of a sporadic task occurs, the next instance cannot occur before (g_i) time units have elapsed.

③ Aperiodic Task:-

→ It can arise at random instants.

→ In case of an aperiodic task, the minimum separation g_i between two consecutive instances can be '0'. That is two or more instances of an aperiodic task might occur at the same time instant.

→ Aperiodic tasks are generally soft-real time tasks.

Task Scheduling

→ Real time Task scheduling refers to determining the orders in which the various tasks are to be taken up for execution by the operating system.

Basic Concepts

① Valid Schedule:- A valid schedule for a set of tasks is one where at most one task is assigned to a processor at a time

no task is scheduled before its arrival time, and the precedence and resource constraints of all tasks are satisfied.

② Feasible Schedule:-

A valid schedule is called a feasible schedule, only if all tasks meet their respective time constraints in the schedule.

③ Proficient Scheduler:-

⇒ A task scheduler "sch.1" is said to be more proficient than another scheduler "sch.2", if 'sch1' can feasible schedule all task sets that sch2 can feasibly schedule, but not vice versa.

⇒ If 'sch1' can feasibly schedule all task sets that 'sch2' can feasibly schedule and vice versa, then sch1 & sch2 are called equally proficient schedulers.

④ Optimal Scheduler:-

A real time task scheduler is called optimal, if it can feasibly schedule any task set that can be feasibly scheduled by any other scheduler.

⑤ Scheduling Points:-

The scheduling points of a scheduler are the points on time line at which the scheduler makes decisions regarding which task is to be seen next.

⑥ Preemptive Scheduler:-

⇒ A preemptive scheduler is one which when a higher priority task arrives, suspends any lower priority task that may be executing and takes up the higher priority task for execution.

→ Thus, in a preemptive scheduler, it can not be the case that a higher priority task is ready and waiting for execution, and the lower priority task is executing.

⑦ Utilization :-

→ The processor utilization of a task is the average time for which it executes per unit time interval.

→ In notations, for a periodic task T_i, the utilization $U_i = \frac{e_i}{p_i}$ where e_i is the execution time and p_i is the period of T_i.

→ For a set of periodic tasks $\{T_i\}$, the total utilization due to all tasks,

$$U = \sum_{i=1}^{n} \frac{e_i}{p_i}$$

⑧ **Jitter** :- (difference bet^n the worst case response time and the best case response time)

→ Jitter is the deviation of a periodic task from its strict periodic behaviour.

→ The arrival time Jitter is the deviation of the task from arriving at the precise periodic time of arrival.

→ Completion time Jitter is the deviation of the completion of a task from precise periodic points.

Classification of Real Time Task Scheduling :-

→ There are 3 main types of schedulers :- (Based on scheduling points)

① Clock - Driven

② Event - Driven

③ Hybrid

① Clock - Driven Schedulers :-

→ The clock Driven schedulers are those in which the scheduling points are determined by the interrupts received from a clock.

→ The clock - Driven schedulers are

ⓐ Table - Driven ⓑ cyclic.

→ Clock driven schedulers are simple and efficient. Therefore these are frequently used in embedded applications.

→ Clock driven schedulers make their scheduling decisions regarding which task to run next only at the clock interrupt points.

→ Clock driven schedulers are those for which the scheduling points are determined by timer interrupts

→ They are also called offline schedulers because these schedulers fix the schedule before the system starts to run. i.e the scheduler pre-determines which task will run when. Therefore these schedulers incurr very little run time overhead.

Limitations:-

These schedules can not handle aperiodic & sporadic tasks since the exact time of occurrence of these tasks can not be predicted. For this reason, this type of schedulers are also called a static scheduler.

Table - Driven Scheduling:-

→ Table Driven schedulers precompute which task would run when and store this schedule in a table at the time the system is designed or configured.

→ Rathere than automatic computation of the schedule by the scheduler, the application programmer can be given the freedom to select his own schedule for the set of tasks in the application and store the schedule in a table called Scheduled table to be cesed by the schedulers at run time.

Exa

Task	start time in miliseconds
T_1	0
T_2	3
T_3	10
T_4	12
T_5	17

From the above table, it is observe that, T_1 starts execution at time 0, T_2 would start 3msec. & so on. One Que? may arcise here that, what would be the size of the schedule table that would be required for some given set of periodic real time tasks to be run on a sys pm.

(23) If a set $ST = \{T_i\}$ of n tasks is to be scheduled, then the entries in the table will replicate themselves after LCM $(P_1, P_2 \cdots P_n)$ time units, where $P_1, P_2 \cdots P_n$ are the periods of $T_1, T_2 \cdots$ LCM $(P_1, P_2 \cdots P_n)$ is called the major cycle of the set of tasks ST.

Q)

Task	phase msec	Execut^n time (msec)	Relative Deadline msec	period msec
T_1	20	10	20	20
T_2	40	10	50	50
T_3	70	20	30	80

If the tasks are scheduled using a table-driven scheduler, what is the length of time for which the schedules have to be stored in the precomputed schedule table of the scheduler.

Ans:- The periodic task can be represented by,

$T_i = \{\phi_i, p_i, e_i, d_i\}$, where,

ϕ_i is called the phase of the task.

P_i - Period of the task.

e_i - worst case execution time of the task

d_i - Relative deadline.

From the above Quest^n, since $P_i = d_i$,

so the tasks are represented as,

$T_1 = (20, 20, 10)$

$T_2 = (40, 50, 10)$

$T_3 = (70, 80, 20)$.

Now, we have to calculate the length of the time. so,

LCM $(P_1, P_2 \cdots P_n)$ i.e. LCM (P_1, P_2, P_3)

= LCM (20, 50, 80) = 400 msec.

= ~~250~~ msec. (Ans)

Cyclic schedulers:-

→ cyclic schedulers are simple, efficient and are easy to program.
→ Ena of cyclic schedulers used in • temperature controller.
→ A cyclic scheduler repeats a precomputed schedule.
→ The precomputed schedule needs to be stored only for one major cycle. Each task in the task set to be scheduled repeats in every major cycle.
→ The major cycle is devided into one or more minor cycles. Each minor cycle is also called a Frame.
→ The size of the frame to be used by the scheduler should satisfy the following three constraints.

(1) Minimum Context Switching:-
→ This constraint is imposed to minimize the number of content switches occuring during task execution.
→ The simplest interpretation of this constraint is that a task instance must complete running within its assigned frame.
→ To avoid unnecessary context switches, the selected frame size should be larger than the execution time of each task. so that when a task starts at a frame boundary it should be able to complete within the same frame.
→ Formally we can state this constraints as $\boxed{\max(\{e_i\}) \leq F}$ where e_i is the execution time of the task T_i and F is the frame size. (This constraint defines the lower bound).

(2) Minimization of the Table size:-
→ This constraint requires that the number of entries in the schedule table should be minimum in order to minimize the storage requirement of the schedule table.
(cyclic schedulers are used in small embedded application with very small storage capacity)

(25) → We can formulate this constraint as $\left\lfloor \frac{M}{F} \right\rfloor = \frac{M}{F}$, in otherwords; if the floor of M/F equals M/F, then the major cycle would contain an integral number of frames.

(3) Satisfaction of Task Deadline:-

→ This constraint imposes that between the arrival of a task and its deadline, there must exist at least one full frame.

→ This constraint is necessary since a task should not miss its deadline because, by the time it could be taken up for scheduling the deadline was imminent.

→ The constraint can be written as, for every T_i, $\boxed{2F - GCD(F, P_i) \leq d_i}$

(This constraint defines the upper bound).

Exa A cyclic scheduler is to be used to run the following set of periodic tasks on a uniprocessor $T_1 = (e_1 = 1, P_1 = 4)$, $T_2 = (e_2 = 1, P_2 = 5)$, $T_3 = (e_3 = 1, P_3 = 20$ $T_4 = (e_4 = ?, P_4 = 20)$. Select an appropriate Frame size.

Ans To select the appropriate frame size, it must satisfies the three constraints.

constraint 1:- Let 'F' be an appropriate fram size. then $\max\{e_i\} \leq F$.

$\Rightarrow \max\{1, 2\} \leq F$

$\Rightarrow 2 \leq F \quad \Rightarrow F \geq 2$.

constraint 2:- The Major cycle 'M' for the given task set is given by $M = LCM(4, 5, 20)$

$= 20$

M should be an integral multiple of the fram size F. i.e. $\boxed{M \bmod F = 0}$

$\therefore F = 2, 4, 5, 10, 20$.

constraint 3:- To satisfy this constraint, we need to check whether a selected frame size F satisfies the inequality.

$2F - GCD(F, P_i) \leq d_i$ for each P_i

(26) Let us First try Frame size - 2.

For $\underline{F=2}$ and Task $\underline{T_1}$,

$2 * 2 - GCD(2,4) \leq 4 \equiv 4 - 2 \leq 4$.

$\therefore$ For P_1, the inequality is satisfied.

For $\underline{F=2}$ and Task $\underline{T_2}$,

$2 * 2 - GCD(2,5)$

$\equiv 4 - 1 = 3 \leq 5$ (di)

$\therefore$ For P_2, the inequality is satisfied.

For $\underline{F=2}$ and Task $\underline{T_3}$,

$2 * 2 - GCD(2,20)$

$\equiv 4 - 2 = 2 \leq 20$.

$\therefore$ For P_3, the inequality is satisfied.

For $\underline{F=2}$ and Task $\underline{T_3}$,

$2 * 2 - GCD(2,20)$

$\equiv 4 - 2 = 2 \leq 20$

$\therefore$ For P_3, the inequality is satisfied.

Thus constraint - 3 is satisfied by all tasks for frame size $\underline{2}$.

so, Frame size '2' satisfies all the constraints. Hence '2' is Feasible Frame size

Ex-2 Consider the following set of periodic real time tasks to be scheduled by a cyclic scheduler. $T_1 = (e_1 = 1, P_1 = 4)$

$T_2 = (e_2 = 2, P_2 = 5)$

$T_3 = (e_3 = 5, P_3 = 20)$. Determine a suitable frame size for the task set.

Ans To determine the suitable Frame size, it must satisfies all the three constraints.

constraint - 1 it says that,

$\max\{e_i\} \leq F$

$\Rightarrow \max\{1, 2, 5\} \leq F$

$\Rightarrow \boxed{F \geq 5}$

result to (20) where e_i is the worst-case execution

constraint 2:- It says that,

(20) 'M' should be an integral multiple of Frame size 'F' i.e. $\boxed{M \bmod F = 0}$.

Now, we have to calculate the major cycle.

$M = LCM(P_1, P_2, P_3)$

$= LCM(4, 5, 20)$

$= 20$

$\therefore F = 2, 4, 5, 10, 20$. But constraint '1' is $F \geq 5$. so the values of 'F' may be

$\{5, 10, 20\}$

constraint-3 :- To satisfy this constraint, we need to check whether a selected Frame size 'F' satisfies the inequality or not.

i.e. $2F - GCD(F, P_i) \leq d_i$

Let's first try, Frame size - '5'

For $F=5$ and Task T_1,

$2 \times 5 - GCD(5, 4)$

$\equiv 10 - 1 = 9 \not\leq 4$ (Not satisfy).

Let it does not satisfy the inequality.

Let $F = 10$, Task T_1,

$2 \times 10 - GCD(5, 10)$

$20 - 5 \leq 10$

$15 \not\leq 10$. (Not satisfy).

since, For all the frame size, it does not satisfy the inequality, To overcome this problem, we need to split the task that is making the task set unschedulable.

For the above task, Task T_3 has more execution time so T_3 is split into.

$T_{3.1} = (20, 1, 20)$

$T_{3.2} = (20, 2, 20)$

$T_{3.3} = (20, 2, 20)$

Now the possible values of f are '2' & '4'.

$\therefore$ For $\boxed{F=2}$ it is feasible

(28) A Generalized Task scheduler:-

→ Generalized task scheduler says that, "how aperiodic & sporadic tasks can be accommodated by cyclic schedulers".
→ In a Generalized scheduler, initially a schedule for only periodic tasks is prepared.
→ The sporadic and aperiodic Tasks are schieduled in the slack times that may be available in the frames.
→ slack time in a Frame is the time left in the frame after a periodic task allocated to the Frame completes its execution.

For sporadic Task

→ A sporadic task is taken up for scheduling only if enough slack time is available for the arriving sporadic task to complete before its deadline. Therefore a sporadic task on its arrival is subjected to an acceptance test.
→ The acceptance test checks whether the task is likely to be completed within its deadline when executed in the available slack times.
→ If it is not possible to meet the task's deadline, then the scheduler rejects it & the corresponding recovery routines are run.

For aperiodic task

→ Since aperiodic tasks do not have strict deadlines, they can be taken up for scheduling without any acceptance test & best effort can be made to schedule them in the available slack times.
→ Though for aperiodic task, no acceptance test is done, but no guarantee is given for a tasks completion time & best effort is made to complete the task as early as possible.

Pseudo-code for a Generalized scheduler


```
cyclic scheduler ()
{
current_task T = schedule_Table [K];
   K = K+1;
   K = K Mod N; // N is the total no. of tasks
                   in the schedule table
   dispatch_current_Task (T);
   schedule_sporadic_tasks ();
   schedule_aperiodic_task();
   idle (); // No task to run, idle.
   }
```

Difference bet" cyclic scheduler & Table-Driven Scheduler

cyclic Scheduler	Table-Driven Scheduler
→ A cyclic scheduler needs to set a periodic timer only once at the application initialization time.	→ A timer has to be set every time a tasks starts to run.
→ A cyclic scheduler is more efficient than the table-driven scheduler.	→ A table driven scheduler is more proficient than a cyclic scheduler because the size of the frame that needs to be chosen should be at least as long as the size of the largest execution time of a task in the task set.

Hybrid schedulers:-
→ In hybrid schedulers, the scheduling points are defined ~~through clock interrupts~~ & in case of both through the clock interrupts and the event occurrence.
→ There is a popular hybrid scheduler known as Time-sliced Round Robin Scheduling.

Time-sliced Round Robin Scheduling:-

→ It is a preemptive scheduling method.
→ In Round-Robin scheduling, the ready tasks are held in a circular queue.
→ The tasks are taken up one after the other in a sequence from the queue.
→ once a task is taken up, it runs for a certain fixed interval of time called its time slice.
→ If a task does not complete within its allocated time slice, it is inserted back into the circular queue.
→ It treats all task equally, and all tasks are assigned identical time slices irrespective of their priority, criticality or closeness of deadline. So tasks with short deadlines might fail to complete on time.
→ Thus it is less proficient than table-driven or cyclic schedulers for scheduling real time task.

EVENT DRIVEN SCHEDULING:-

→ In Event Driven scheduling, the scheduling 7 points are defined by certain events which precludes clock interrupts. or defined by task completion and task arrival events.
→ This class of schedulers are preemptive. i.e. a higher priority task when ready, preempts any lower priority task that may be running.
→ This schedulers are less suitable for embedded applications as these are required to be of small size, low cost and consume minimal amount of power.

result to

)
:-
method.
ready
ie.
r the other

for a
ed its

in its
back

all tasks
respective
oseness of
deines

r table-
scheduling

: scheduling
is which
by task

ic
empteive.
eady,
that may

ble for
ee

cost

(30) where ei is the worst ca

(31) → Exa. of Event-Driven schedulers are:
8 (1) Simple priority Based or Foreground-Background scheduler.
(2) Earliest deadline First (EDF)
(3) Rate Monotonic Analysis (RMA)

(1) Simple priority Based or Foreground-Background

→ It is a simple priority-driven preemptive scheduler.
→ In this case, the real-time tasks in an application are run as foreground tasks.
→ The sporadic, aperiodic and non-real time task are run as background tasks.
→ Among the foreground tasks, at every scheduling point, the highest priority task is taken up for scheduling.
→ A background task can run when none of the foreground tasks is ready. In otherwords, the background tasks run at the lowest priority.
→ Let us assume that, in a certain real time system, there are n foreground tasks which are denoted as $T_1, T_2 \cdots T_n$. Since the foreground tasks are periodic, let T_B be the only Background tasks.
Let e_B be the processing time requirement of T_B.
In this case, the completion time (ct_B) for the Background task is given by

$$ct_B = \frac{e_B}{1 - \sum_{i=1}^{n} \frac{e_i}{P_i}}$$

CPU utilizat"

This expr" says that, when any foreground task is executing, the background task waits. The average CPU utilization due to the foreground task T_i is e_i/P_i, since e_i amount of processing time is required ove every P_i period. It follows that all foreground tasks together would result in CPU utilization of $\sum_{i=1}^{n} \frac{e_i}{T_i}$. Therefore the

average time available for execution of the background tasks in every unit of time is

$$1 - \sum_{i=1}^{n} \frac{e_i}{P_i}$$

Ex: Consider a Real-time system in which tasks are scheduled using foreground-background scheduling. There is only one periodic foreground Task T_f ($\phi_f = 0$, $e_f = 50$ msec, $P_f = 100$ msec, $d_f = 100$ msec). & the background task be $T_B = (e_B = 1000$ msec.). Compute the completion time for background task

Ans: To Compute, the task completion time. for the Background task is given by,

$$C t_B = \frac{e_B}{1 - \sum_{i=1}^{2} \frac{e_i}{P_i}}$$

$e_B = 1000$ msec.

$$\therefore \quad C t_B = \frac{1000}{1 - \frac{50}{100}} = 1000 \times 2 = 2000 \text{ msec.}$$

② Earliest Deadline First (EDF) scheduling:-

→ In EDF scheduling, at every scheduling point the task having the shortest deadline is taken up for scheduling.

→ A task set is schedulable under EDF, if and only if it satisfies the condition that the total processor utilization due to the task set is less than 1.

→ For a set of periodic real-time tasks $\{T_1, T_2 \ldots T_n\}$, EDF schedule can be Expressed as

$$\sum_{i=1}^{n} \frac{e_i}{P_i} = \sum_{i=1}^{n} u_i \leq 1 \quad \text{..... (1)}$$

where u_i is the average utilization due to the task T_i & 'n' is the total number of tasks in the task set.

→ In the above Expr^n 1 we assumed that, the period of each task is the same as its deadline.
→ when period of tasks different from deadline, in that case, the schedulability test needs to be changed.
→ If $P_i > d_i$, then, each task needs e_i
10. amount of computing time every $\min(P_i, d_i)$ duration of time. Then,

$$\sum_{i=1}^{n} \frac{e_i}{\min(P_i, d_i)} \leq 1 \quad \text{—— (2)}$$

→ If $P_i < d_i$, It does not satisfy exp^n-2. Hence, Expr^n-2 is conservative and is not a necessary condition, but only a sufficient condition for a given task set to be EDF schedulable.
→ A variant of EDF scheduling is Minimum Laxity First (MLF) scheduling. In MLF, at every scheduling point, a laxity value is computed for every task in the system. & the task having the minimum laxity is executed First.
→ Laxity of a task measures the amount of time that would remain if the task is taken up for execution next. Laxity is a measure of the flexibility available for scheduling a task.

Shortcomings of EDF :-

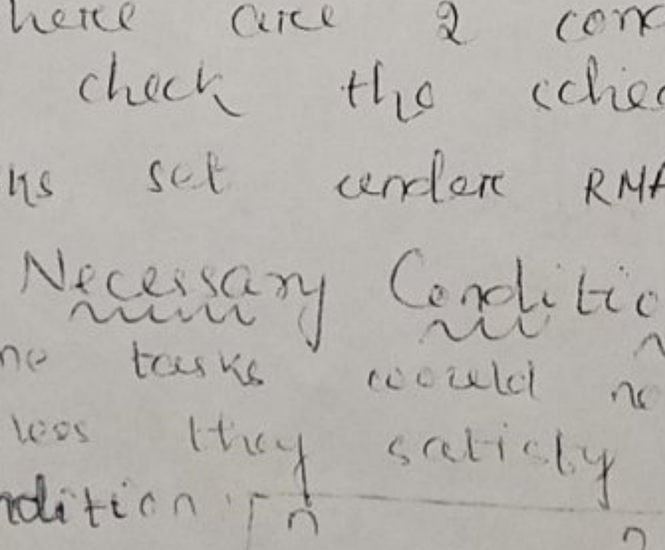
(1) **Transient Overload problem:-**
→ It denotes the overload of a system for a very short time.
→ It occurs when some task takes more time to complete than what was originally planned during the design time.

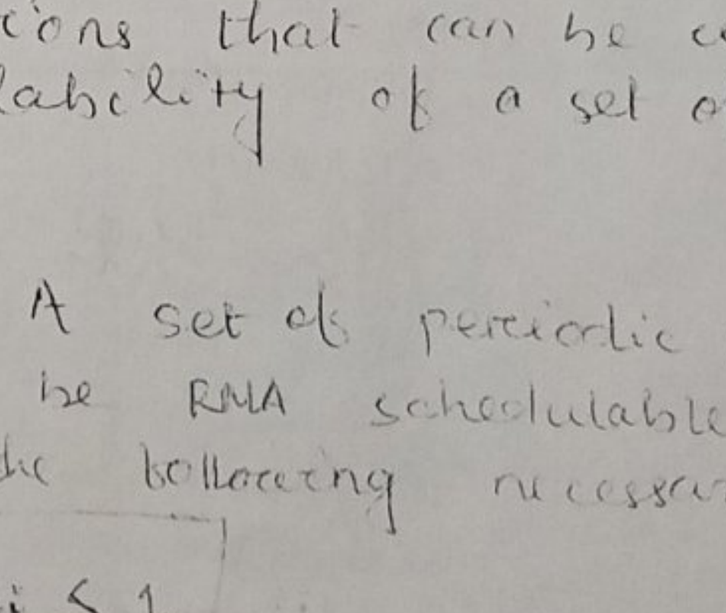
(2) **Resource sharing Problem:-**
when EDF is used to schedule a set of real time tasks high overheads might have to be incurred to support resource sharing among the tasks without making tasks to miss their respective deadlines.

③ Efficient Implementation Problem:-
It is not often feasible as it is difficult to restrict the number of tasks with distinct deadlines to a reasonable number.

③ RATE MONOTONIC ALGORITHM :- (RMA)

→ RMA is a static priority algorithm and is used in practical applications.
→ It assigns priorities to tasks based on their rates of occurrence
→ The lower the occurrence rate of a task, the lower is the priority assigned to it. A task having the highest occurrence rate is accorded the highest priority.
→ In RMA, the priority of a task is directly proportional to its rate or inversely proportional to its period. i.e the priority of any Task T_i is computed as, $\boxed{\text{priority} = \frac{k}{P_i}}$, where P_i is the period of Task T_i & 'k' is a constant.





Schedulability Test for RMA:-

→ Schedulability of a Task Test under RMA can be determined from a knowledge of the worst-case execution times and periods of the task.
→ There are 2 conditions that can be used to check the schedulability of a set of tasks set under RMA

1. Necessary Condition:- A set of periodic Real time tasks would not be RMA schedulable unless they satisfy the following necessary condition:

$$\sum_{i=1}^{n} \frac{e_i}{P_i} = \sum_{i=1}^{n} u_i \leq 1$$

where e_i is the worst-case execution time and P_i is the period of the Task T_i, 'n' is the number of tasks to be scheduled, and u_i is the CPU utilization due to the task T_i.

(8) **Sufficient Condition**:-

A set of 'n' real time periodic tasks are schedulable under RMA, if

$$\boxed{\sum_{i=1}^{n} u_i \leq n\left(2^{\frac{1}{n}} - 1\right)} \quad \text{——— (1)}$$

where u_i is the utilization due to task T_i.

Lets consider the case, where there is only one task in the system. i.e. $n = 1$.

so, substituting $n = 1$, in equ^n (1), we get.

$$\sum_{i=1}^{1} u_i \leq 1\left(2^{\frac{1}{1}} - 1\right).$$

$$\Rightarrow \sum_{i=1}^{1} u_i \leq 1.$$

for $n = 2$,

$$\sum_{i=1}^{2} u_i \leq 2\left(2^{\frac{1}{2}} - 1\right).$$

$$\Rightarrow \sum_{i=1}^{2} u_i \leq 0.824.$$

for $n = 3$,

$$\sum_{i=1}^{3} u_i \leq 3\left(2^{\frac{1}{3}} - 1\right)$$

$$\Rightarrow \sum_{i=1}^{3} u_i \leq 0.78.$$

⋮

for $n = \infty$,

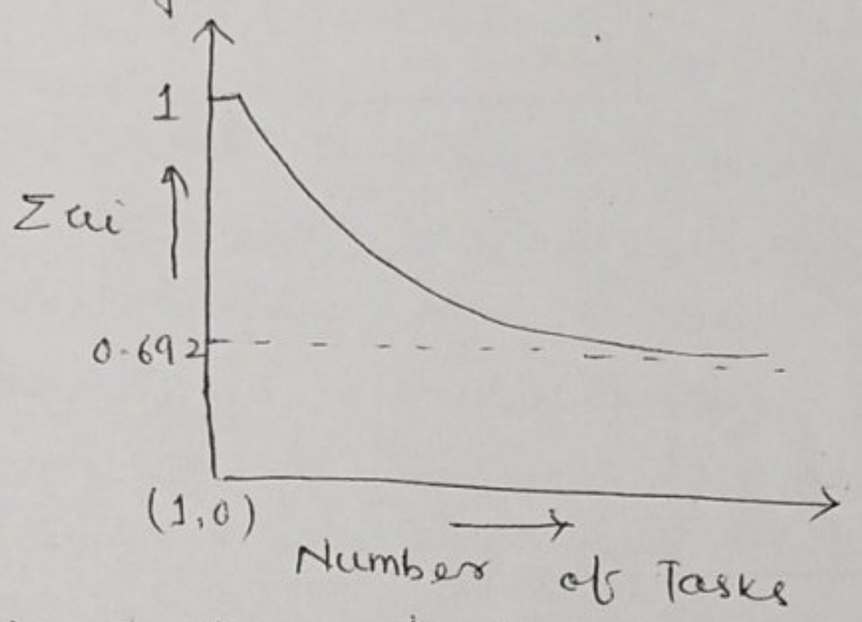
$$\sum_{i=1}^{\infty} u_i \leq \infty\left(2^{\frac{1}{\infty}} - 1\right)$$

$$\Rightarrow \sum_{i=1}^{\infty} u_i \leq \infty \cdot 0$$

So when $n \to \infty$, it gives an indeterminate exp^n.

→ By applying L'Hospitals rule we can verify that the right hand side of the expression evaluates to $\log_e 2 = 0.692$

(36) → From the above computation, it is clear that, the maximum CPU utilization that can be achieved under RMA is 1. This is achieved when there is only a single task in the system. As the number of tasks increases, the achievable CPU utilization falls and as $n \to \infty$, the achievable utilization stabilizes at $\log_e 2$. which is approximately 0.692.



Ex: check whether the following set of periodic real-time tasks is schedulable under RMA on a uniprocessor
$T_1 = (e_1 = 20, P_1 = 100)$, $T_2 = (e_2 = 30, P_2 = 150)$,
$T_3 = (e_3 = 60, P_3 = 200)$.

Sol^n:- The total CPU utilization achieved due to the three given tasks is

$$\sum_{i=1}^{3} u_i = \frac{20}{100} + \frac{30}{150} + \frac{60}{200}$$

$= 0.7$ which is less than 1. which satisfied the necessary condition.
Now checking the sufficient condition,
According to Liu & Layland's condit^n,

$$\sum_{i=1}^{3} u_i \leq 3(2^{\frac{1}{3}} - 1) \leq 0.78.$$

The total utilizat^n has already found that 0.7 which is less than 0.78. which is satisfied the sufficient condition.
Therefore the task set is RMA schedulable.

that,

ved
ystem
ble
ievable

: a real-
: processor

ed due

1. which

nd that
ich is

lulable.

(37) According to Lehoczky's Test, that a given Task is schedulable under RMA:-

→ Let $\{T_1, T_2, \cdots T_i\}$ be the set of tasks to be scheduled. Assume that the tasks have been ordered in descending order of their priority i.e. $(T_1) > pri(T_2) \cdots > Prei(T_i)$. where $Prei(T_i)$ denotes the priority of the Task T_i.

→ Consider that, the Task T_i arrives at the time instant '0'. During the first instance of the task T_i, three instances of the task T_1 have occured. Each time T_1 occurs, T_i has to wait since T_1 has higher priority than T_i.

→ Now determine the exact number of times that T_1 occures within a single instance of T_i This is given by $\left[\frac{P_i}{P_1}\right]$. Since T_1's execution time is e_1, then the total execution time required due to Task T_1 before the deadline of T_i is $\left[\frac{P_i}{P_1}\right] \times e_1$.

→ Therefore the time for which T_i will have to wait due to all its higher priority tasks can be expressed as:

$$\sum_{k=1}^{i-1} \left[\frac{P_i}{P_k}\right] \times e_k \quad \text{---(1)}$$

[It gives the total time required to execute T_i's higher priority tasks for which T_i would have to wait]

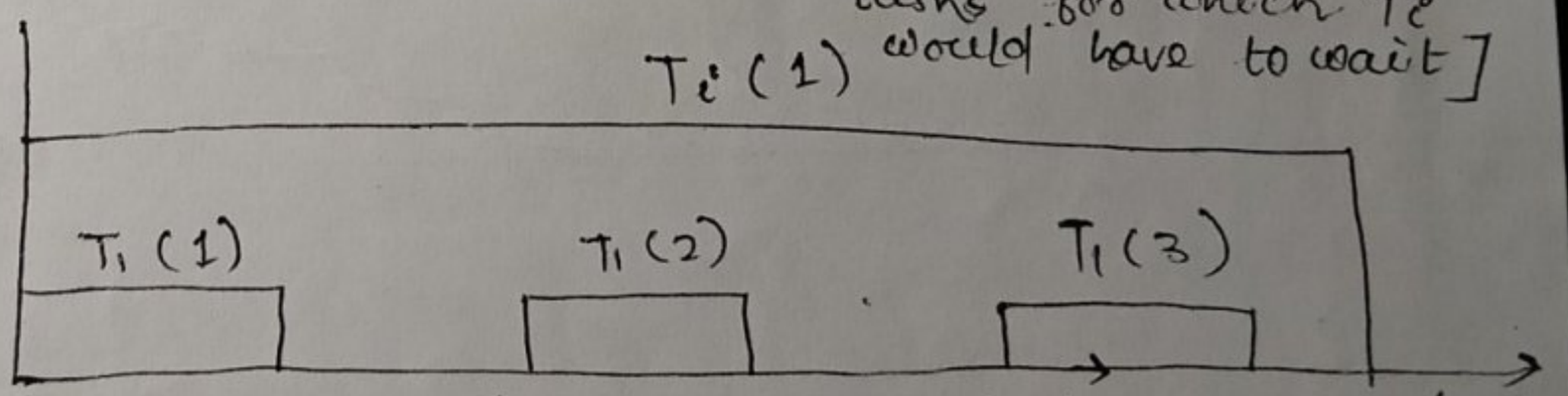


Fig: (3 instances of T_1 over a single instance of T_i)

→ Task T_i would meet its first deadline, if

$$e_i + \sum_{k=1}^{i-1} \left[\frac{P_i}{P_k}\right] \times e_k \leq d_i$$

(39) Advantages & Disadvantages of RMA:-

Adv: → RMA is very commonly used for scheduling real-time tasks in practical applications.
→ RMA is simple and efficient and is also the optimal static priority task scheduling algorithm.
→ RMA possesses good transient overload handling capability. this means that when a lower priority task does not complete within its planned completion time, it can not make any higher priority task to ~~me~~ miss its deadline.

Dis:
→ It is very difficult to support scheduling of aperiodic and sporadic tasks under RMA.
→ RMA is not optimal when task periods and deadlines differ.

Deadline Monotonic Algorithm (DMA):-
→ It is a periodic Real-time tasks, $d_i \neq p_i$, in such that case DMA is used.
→ DMA is essentially a variant of RMA and assigns priorities to tasks based on their deadline rather than assigning priorities based on task periods as done in RMA.
→ DMA assigns higher priorities to task with shorter deadlines.
→ When the relative deadline of every task is proportional to its period, RMA & DMA produce identical solutions.

Exa: Is the following task set schedeeled by DMA? Also check whether it is schedulable using RMA. $T_1 = (e_1 = 10\ msec, P_1 = 50\ msec, d_1 = 35\ msec)$
$T_2 = (e_2 = 15\ msec, P_2 = 100\ msec, d_2 = 20\ msec)$.
$T_3 = (e_3 = 20\ msec, P_3 = 200\ msec, d_3 = 200\ msec)$.

Ans = First check RMA schedulability of the given set of tasks, by checking the Lehoczky's criterion. The tasks are already ordered in descending order of their priorities.

(40) checking for T_1 : 10 msec < 35 msec. Therefore T_1 would meet its first deadline.
checking for T_2 : 10 + 15 $\not\leq$ 20, Therefore, T_2 will miss its first deadline. Hence, the given task set can not be feasibly scheduled under RMA.

Now let us check the schedulability using DMA:

Under DMA, the priority ordering of the tasks is follows: $pr(T_2) > pr(T_1) > pr(T_3)$.
checking for T_2: 10 msec < 35 msec, hence, T_2 will meet its first deadline.
checking for T_1: (15+20) msec <

Theorem For a set of harmonically related tasks $HS = \{T_i\}$. The RMA schedulability criterion is given by $\sum_{i=1}^{n} u_i \leq 1$

Proof:- Let us assume that $T_1, T_2 \ldots T_n$ be the tasks in the given Task set. & assume that, the tasks in the task set $\{T_1, T_2 \ldots T_n\}$ have been arranged in increasing order of their periods.
i.e. for any 'i' & 'j', $P_i < P_j$ whenever $i < j$.
we know, a task T_i meets its deadline if.

$$e_i + \sum_{k=1}^{i-1} \left\lceil \frac{P_i}{P_k} \right\rceil * e_k \leq P_i .$$

Since, the task set is harmonically related. P_i can be written as $m * P_k$ for some m. Using this $\left\lceil \frac{P_i}{P_k} \right\rceil = \frac{P_i}{P_k}$.
For $T_i = T_n$, we can write,

$$e_n + \sum_{k=1}^{n} \frac{P_n}{P_k} * e_k \leq P_n .$$ Devide both side of this expn by P_n, we get the

lat form (43)

required result. the task set would schedulable, if.

$$\sum_{k=1}^{n} \frac{e_k}{P_k} \leq 1, \text{ or}$$

$$\boxed{\sum_{i=1}^{n} u_i \leq 1}$$

Context Switching Overhead:-
In order to take context switching time into consideration, in all schedulable computations, we need to replace e_i by $e_i + 2c$ for each T_i.

Issues in using RMA in practical situation:-
While applying RMA to practical problems, more issues are comes. These are.
(1) Handling Critical Tasks with long period
(2) Handling Aperiodic & sporadic Tasks.
(3) Coping with Limited priority levels.
(4) Dealing with task Jitter.

Classification of Real Time Tasks based on Acceptance test used:-
Based on Acceptance test used, there are two broad categories of task schedulers.
① Planning - Based ② Best effort.
In planning - Based schedulers, when a task arrives the scheduler first determines whether the task can meet its deadlines, if it is taken up for execution. Or not if it is rejected.
→ If the task can meet its deadline and does not cause others already scheduled tasks to miss their respective deadlines, then the task is acceptance for scheduling. otherwise it is rejected.
In best effort, best effort is made to meet its deadline. but no guarantee is given to whether a task's deadline would be met

(12)

classification based on the target platform on which the task are to be run;

According to this scheme, the different classes of scheduling algorithim are.

① Uniprocessor.
② Multiprocessor.
③ Distributed.

lat form

ifferent

(43)

MODULE-II

CHAPTER-3

Handling Resource Sharing & Dependencies Among Real time Tasks:-

Resource sharing Among RTS:-

Non-preemptive Resource or Critical Resources:
If a task is pre-empted from using a resource before it completes using the resource, then the resource can become corrupted. Example of such resources are files, devices & certain data structures. These resources are called non-preemptive resource or critical resources

Serially Reusable Resource:-

A serially Reusable resource is one which is used in exclusive mode, but a task using it may at any time be preempted from using it and then allowed to use it at some latter time without affecting the correctness of the computed results.

Priority Inversion:-

Simple priority Inversion:-

When a lower priority task is already holding a resource, a higher priority task needing the same resource has to wait and can not make progress with its computations. The higher priority task would remain blocked until the lower priority task releases the required non-preemptable resource. In this situation the higher priority task is said to simple priority Inversion on account of lower priority task.

Unbounded priority Inversion-

→ It occurs when a higher priority task waits for a lower priority task to release a resource it needs, and meanwhile intermediate priority tasks preempt the lower priority task from CPU repeatedly.
→ As a result, the lower priority task can not complete its usage of the critical resource and the higher priority task waits indefinitely for its required resource to be released.

Exa

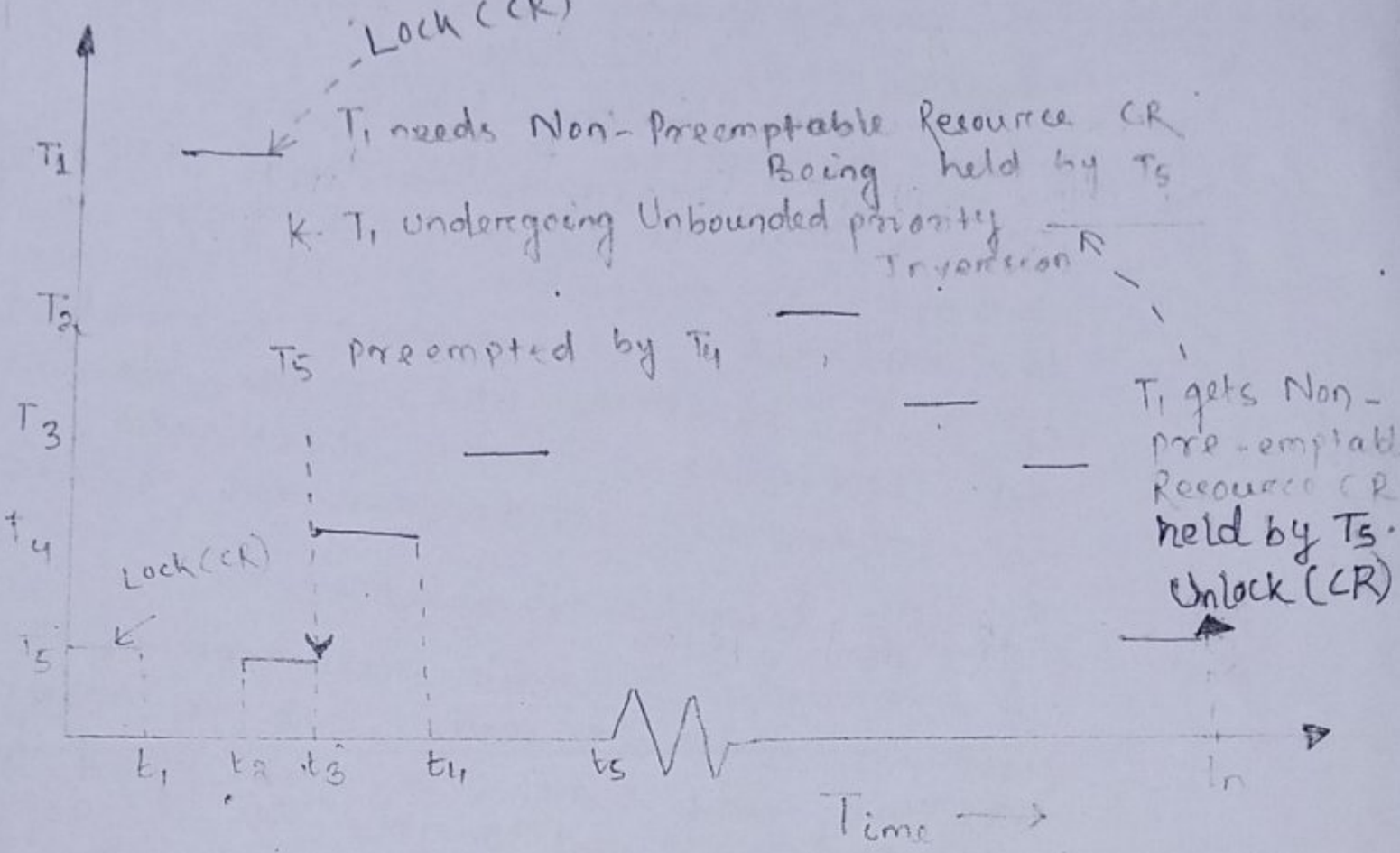


Fig: Explanation of Unbounded priority Inversion using a Timing Diagram

→ In this example, there are five Tasks. T_1, T_2, T_3, T_4, T_5. T_5 is a low priority task & T_1 is a high priority task. Tasks T_2, T_3, T_4 have priorities higher than T_5 but lower than T_1 (called intermediate priority Tasks).

→ At time t_0, the low priority task T_5 is executing and is using a non-premptable resource (CR). After a while (at time instant t_1) the higher priority task T_1 arrives, preempts T_5 and starts to execute.

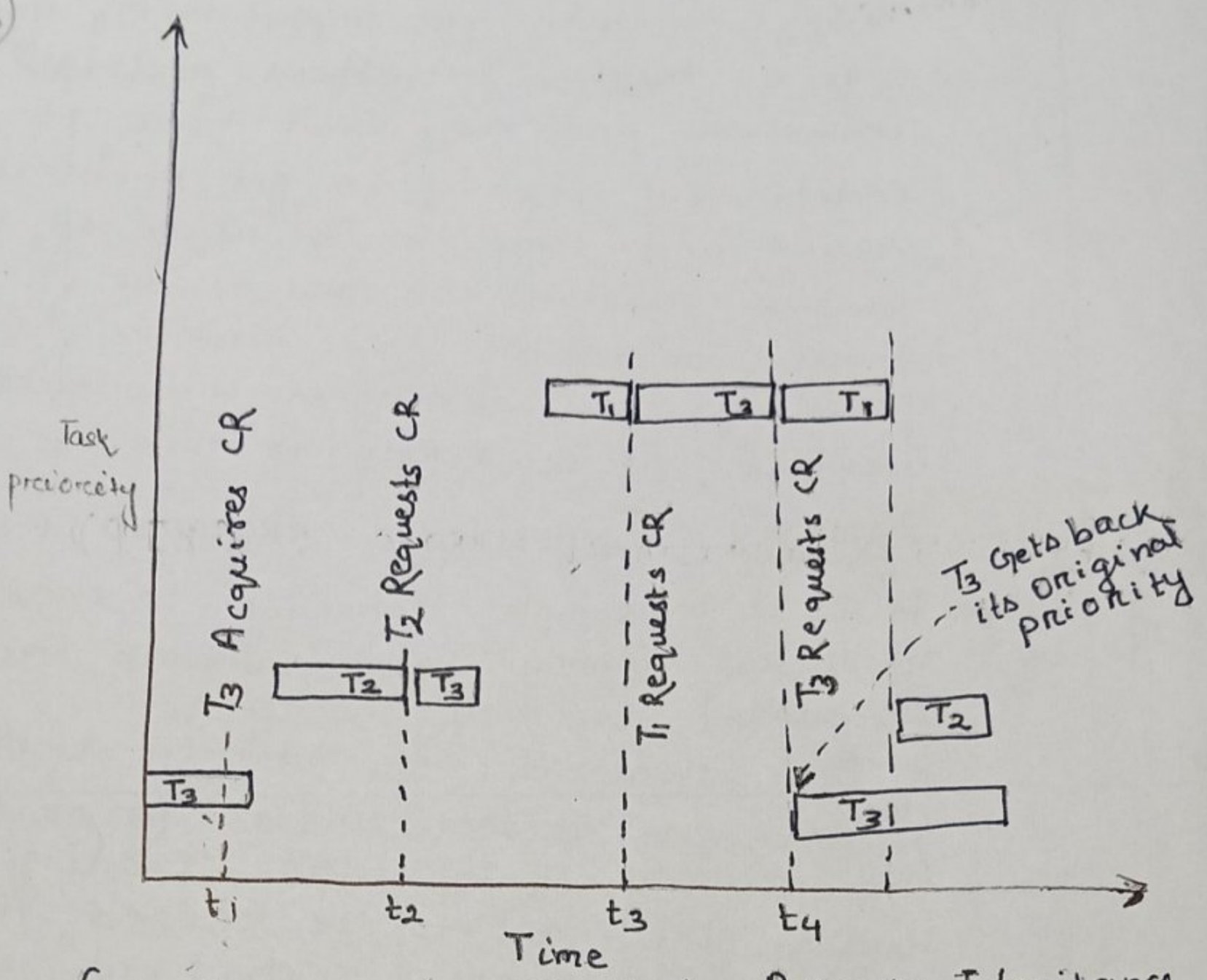
→ Task T_1 requests to use the resource CR at t_2 and blocks since the resource is being held by T_5

→ Since T_1 blocks, T_5 resumes its execution at t_2. However the task T_4 which does not require the non-preemptable resource CR preempts T_5 from CPU usage and starts to execute at time t_3.

(12) → T_4 is in turn pre-empted by T_3 and so on.
→ As a result, T_1 suffers multiple priority inversions and may even have to wait indefinitely for T_5 to get a chance to execute and complete its usage of the critical resource CR and release it. It should be clear that when a high priority task undergoes unbounded priority inversion, it is likely to miss its deadline.

PRIORITY INHERITANCE PROTOCOL:- (PIP)

↳ It is a simple technique to share critical resources among tasks without incurring unbounded priority inversions.
→ The essence of this protocol is that
"Whenever a task suffers priority inversion, the priority of the lower priority task holding the resource is raised, through a priority inheritance mechanism."
→ This enables it to complete its usage of the critical resource as early as possible without having to suffer pre-emptions from the intermediate priority tasks.
When several tasks are waiting for a resource the task holding the resource inherits the highest priority of all tasks waiting for the resource.
→ Since, the priority of the low priority task holding the resource is raised to equal the highest priority of all tasks waiting for the resource being held by it, intermediate priority tasks can not preempt it and unbounded priority inversion is avoided.



(Eg: Priority changes under Priority Inheritance Protocol)

→ In the above fig, the priority changes of tasks are captured on a time line.

→ The task T_3 initially locks the resource CR. After some time, it is pre-empted by T_2. The task T_2 requests the resource CR at t_2. Since, T_3 already holds the resource, T_2 blocks & T_3 inherits the priority of T_2. so T_2 and T_3 drawing at the same priority levels.

→ Before T_3 could complete use of resource CR, it is preempted by a higher priority task T_1, T_1 requests for CR_1 at time t_3 and T_1 blocks as CR_1 is still being held by T_3. so at this point there are two tasks (T_2 and T_1) waiting for the resource.

→ T_3 inherits the priority of the highest priority waiting task. T_3 completes its usage of the resource at t_4 and as soon as it releases the resource, it gets back its oreiginal priority

It suffers from two important problems. Deadlock and chain blocking.

(1) Deadlock :-

T_1 : Lock CR_1, lock CR_2, Unlock CR_2, Unlock CR_1

T_2 : Lock CR_2, lock CR_1, Unlock CR_1, Unlock CR_2.

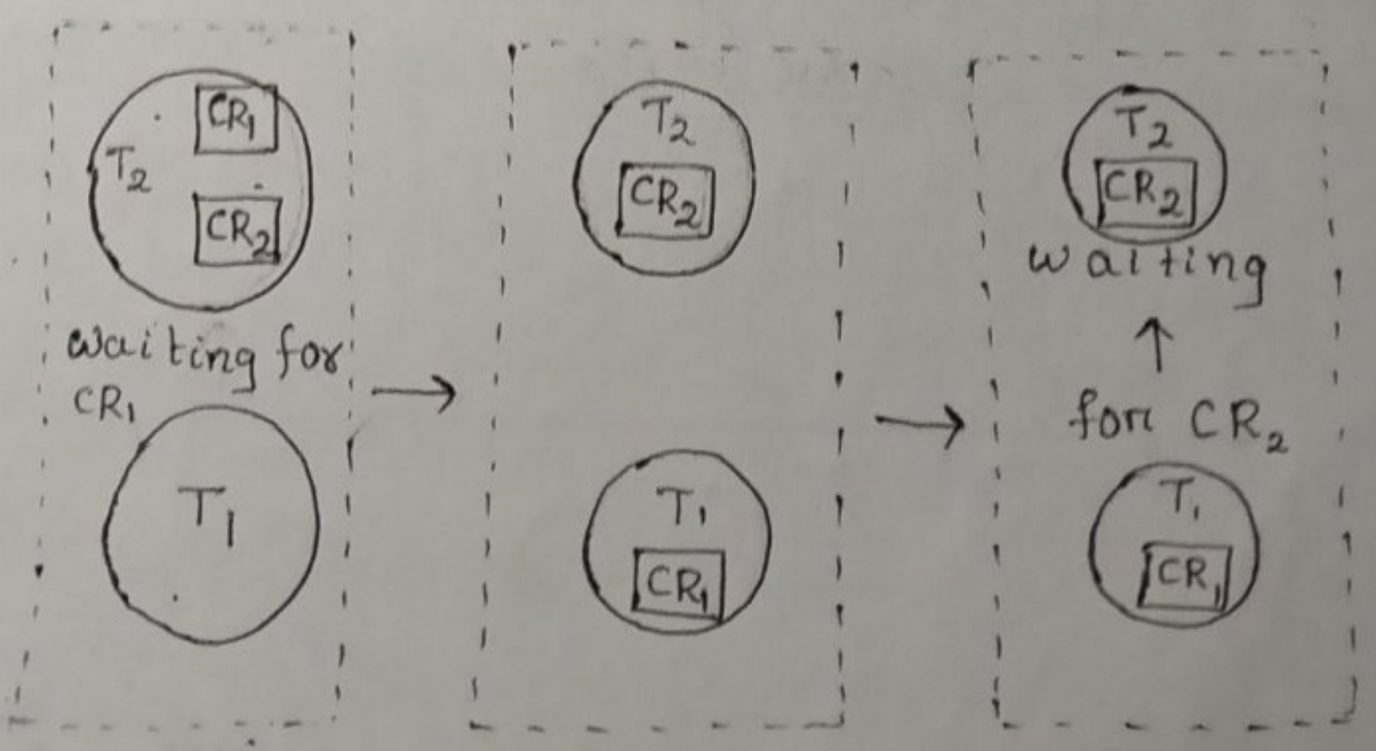
Assume that T_1 has highere priority than T_2. T_2 start running first and locks critical resource CR_2 (T_1 was not ready at that time). After some time, T_1 arrives, preempts T_2, and starts execution

→ After some time, T_1 locks CR_1 and then tries to lock CR_2 which is being held by T_2. As a consequence, T_1 blocks and T_2 inherits T_1's priority according to the priority Inheritance protocol.

→ T_2 Resumes its execution and after sometime needs to lock the resource CR_1 being held by T_1. Now T_1 & T_2 both deadlocked.

(2) Chain Blocking :-

A task is said to undergo chain blocking. If each time it needs a resource, it undergoes priority inversion. Thus if a task needs n resources for its computations, it might have to undergo priority inversions n times to acquire all its resources.



[Fig: Chain Blocking in PIP]

Assume that, a task T_1 needs several resources.
⇒ On the first trig, a low priority Task T_2 is holding two resources CR_1 & CR_2, and a high priority task T_1 arrives and requests to lock CR_1. It undergoes priority inversion and causes T_2 to inherit its priority.
⇒ On the second trig, as soon as T_2 releases CR_1, its priority reduces to its original priority and T_1 is able to lock CR_1.
⇒ On the third trig, after executing for some time, T_1 requests to lock CR_2. This time it again undergoes priority inversion since T_2 is holding CR_2. T_1 waits until T_2 releases CR_2.

From this example, we can see that chain blocking occurs when a task undergoes multiple priority inversions to lock the required resources.

Highest locker Protocol (HLP) :-
→ Under HLP, as soon as a task acquires a resource, its priority is set equal to the ceiling priority of the resource.
⇒ If a task holds multiple resources, then it inherits the highest ceiling priority of all its locked resources.
→ Let the ceiling priority of a resource R_i be denoted by ceil (R_i) and the priority of a task T_j be denoted as pri(T_j).
Then ceil (R_i) can be defined as

$$ceil(R_i) = max(\{pri(T_j) / T_j \text{ needs } R_i\})$$

→ That is, the ceiling priority of a critical Resource R_i is the maximum of the priority of all tasks that may use R_i.

(4a) → This expression holds only when higher priority values indicate higher priority.

→ If higher priority values indicate lower priorities, then the ceiling priority would be the minimum priority of all tasks needing the resource.

i.e $ceil(R_i) = min(\{pri(T_j) / T_j \text{ needs } R_i\})$

→ For Operating systems, supporting time-slice round-Robin scheduling among equal priority tasks and larger priority value indicates higher priority, the rule for computing the ceiling priority is $ceil(R_i) = max(\{pri(T_j) | T_j \text{ needs } R_i\}) + 1$

→ For the case where larger priority value indicates lower priority (and time sliced round-robin scheduling among equal priority tasks), we have to take the minimum of the task priorities i.e

$ceil(R_i) = min(\{pri(T_j) / T_j \text{ needs } R_i\}) + 1$

How priority ceiling of Resources are compute

Consider there are two Resources - CR_1 & CR_2 & tasks are T_1, T_5, T_7 having priorities 10, 5, 2.

~~then ceil (CR1~~ CR_1 is shared by T_1, T_5 & T_7. CR_2 is shared by T_5 & T_2.

Then $ceil(CR_1) = max\{10, 5, 2\} = 10$

$ceil(CR_2) = max\{5, 2\} = 5$

If a task holding more than one resource

its prionity will become maximum of the ceiling prionities of all the resources it is holding.

→ A task holding both CR_1 and CR_2 would inherit the langer of the two ceiling prionities i.e 10.

Shortcomings of HLP:-

→ Though HLP solves the problems of unbocended prionity inveresion, deadlock and chain blocking, it opens up the possibility for inheritance-related inversion

→ Inheritance-related inversion occures when the prionity value of a low prionity task holding a resource is raised to a high value by the ceiling rule, the intermediate prionity tasks not needing the resource can not execute and are said to undergoes inheritance-related prionity inversion.

Prionity Ceiling Protocol (PCP) :-

→ It used to solve the problems of unbocended prionity inveresion, chain blocking and deadlocks as well as minimize the inheritance related inveresions.

→ PCP associates a ceiling value ceil(CRi) with every resource CRi, that is the maximum of the prionity values of all tasks that might use CRi.

→ A variable called csc (current System ceiling) is used to keep the maximum ceiling value of all the resource

that are in use at any instant of time.

→ Thus at any time

CSC = max { ceil (CRi) | CRi is currently in use }

→ At system start, csc is initialized to '0'.

→ Resource sharing among tasks under PCP is regulated using two rules for handling resource requests:

1) Resource Grant and 2) Resource Release.

1) Resource Grant Rule:

It consists of two clauses. These two clauses are applied when a task Ti requests to lock a resource.

(a) Resource Request clause:-

→ If the task Ti is holding a resource whose ceiling priority equals csc, then the task is granted access to the resource.

→ otherwise, Ti will not be granted CRj. unless its priority is greater than csc. [i.e pri(Ti) > csc).

(b) Inheritance clause:-

When a task is prevented from locking a resource by failing to meet the resource grant clause, it blocks & the task holding the resource is less than that of the blocked task.

2) Resource Release Rule:-

If a task releases a critical resource it was holding and if the ceiling priority of this resource equals csc, then csc is

made equal to the maximum of the ceiling value of all other resources in use. else csc remains unchanged.

Different types of Priority Inversions Under PCP :-

There are 3 important types of priority Inversion.

1. Direct Inversion
2. Inheritance Related Inversion.
3. Avoidance inversion

1. Direct priority Inversion:-

- Direct inversion occceres when a higher priority task waits for a lower priority task to release a resource that it needs.

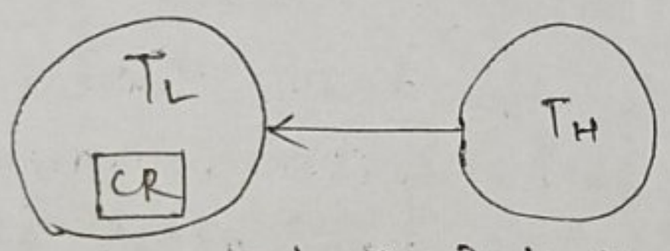


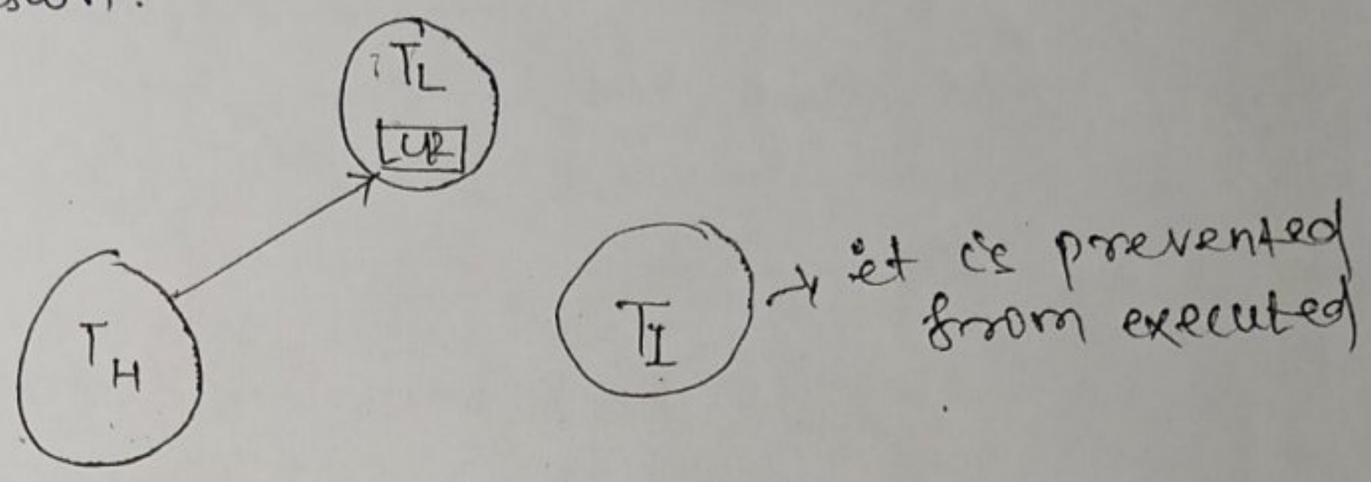
Fig: Direct Priority Inversion.

- In the above figure, suppose a low priority task T_L is holding a resource CR, Now if a highere priority task T_H needs this resource, then it would have to wait tell T_L finishes using CR and release it.

2. Inheritance Related Inversions.-

Consider a situation where a lower priority task is holding a resource and a highere priority task is waiting for it. Then the priority of the lower priority task is raised to that of the waiting higher priority task by the inheritance clause of PCP. As a result, the

(53) the intermediate priority tasks not needing the resource undergo inheritance-related inversion.

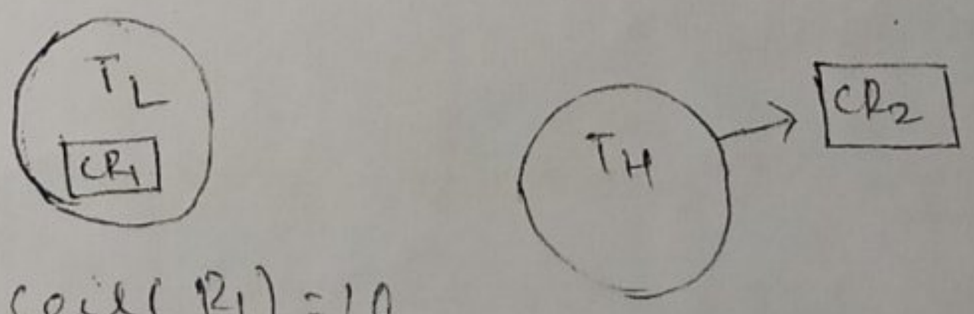


[Fig: Inheritance Related Inversion]

3. Avoidance Related Inversion:-

→ It is also sometimes refererred to as priority-ceiling related inversion. since a higher priority task is not allocaed to execute, not because it requests a resource that is already locked by another task, but becoz, its priority is less than the value of csc.

→ In this case, a high priority task blocks for a resource that is not being used by any of the tasks. So this restriction is necessary to prevent deadlocks. This type of inversion is called avoidance Related inversion.



(Fig: Avoidance Related Inversion)

(54) Handling Task Dependencies :-

If one task depends the result of another task, then in that case, we use the ~~tal~~ following Alg.

Table - Dreiven Alg:-

Step 1 :- Sort task in increasing oredere of their deadlines, without violating any precedence constraints and store the stored tasks in a linked list.

Step. 2:

Repeat

Take rep the task having largest deadline and not yet scheduled (i.e scan the task list of Step. 1 from left).

Schedule it as late as possible

Until all tasks are scheduled

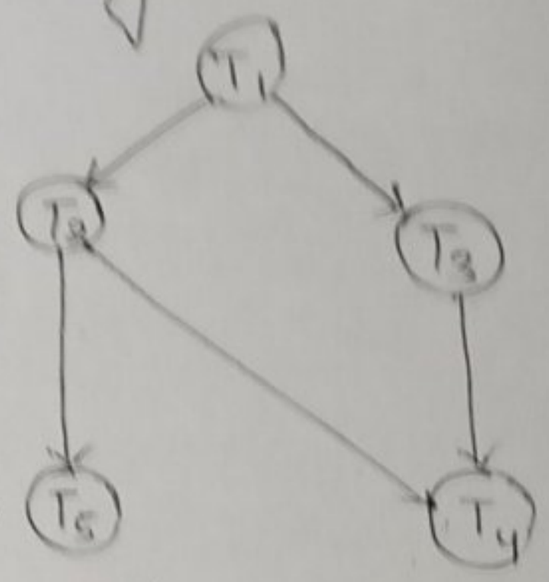
Step. 3

Move the schedule of all tasks to as much left (i.e early) as possible without disturbing their relative positions in the schedule.

Exa: Determine a Feasible schedule for the real time tasks of a task set $\{T_1, T_2 \cdots T_5\}$ for which the precedence relations have been shown the below fig. for use with a table driven scheduler.

The execution times of the tasks $T_1, T_2 \cdots T_5$ are 7, 10, 5, 6, 2 & the

(3a) corresponding deadlines are, 40, 50, 25, 20, 8



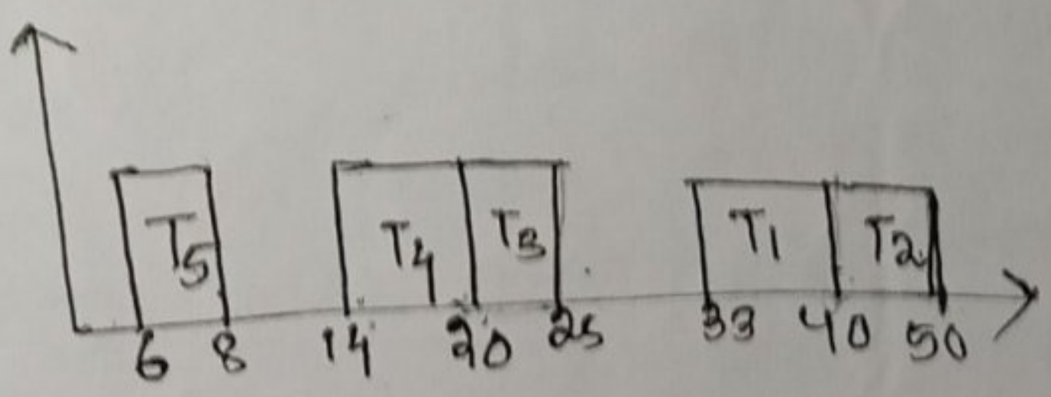
Ans:

Tasks	Ex. time	deadline
T_1	7	40
T_2	10	50
T_3	05	25
T_4	06	20
T_5	02	08

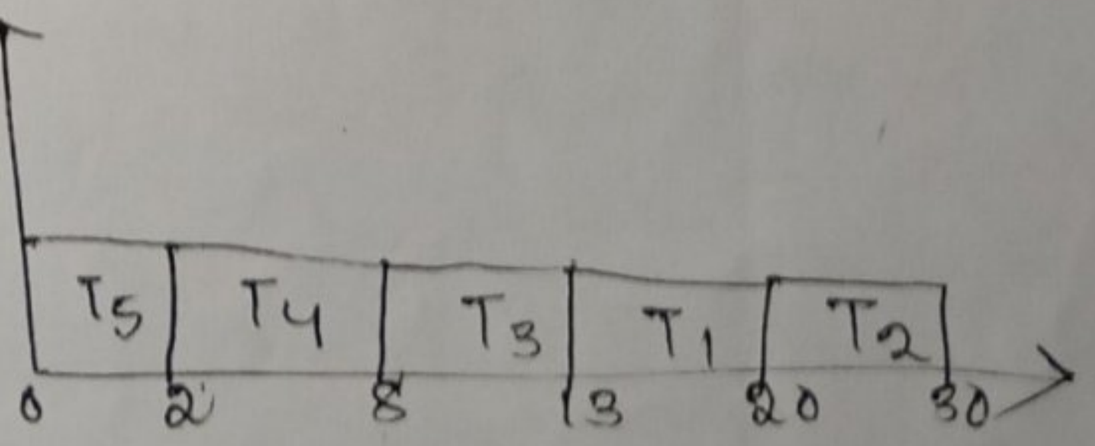
Step. 1: Arrange tasks in decreasing order by their deadline.

T_5 T_4 T_3 T_1 T_2.

Step. 2



Step. 3



(52) Handling Task Dependencies :-

If one task depends the result of another task, then in that case, we use the ~~tal~~ following Alg.

Table - Driven Alg:-

Step 1 :- Sort task in increasing order of their deadlines, without violating any precedence constraints and store the sorted tasks in a linked list.

Step. 2 :

Repeat

Take rep the task having largest deadline and not yet scheduled (i.e scan the task list of Step. 1 from left).

Schedule it as late as possible

Until all tasks are scheduled

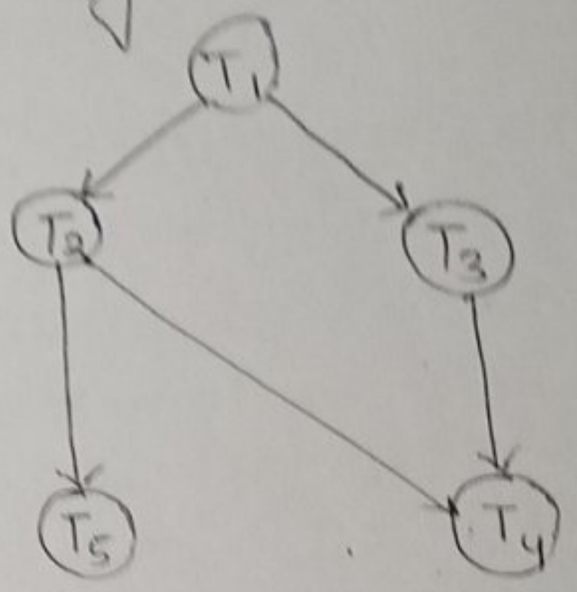
Step. 3

Move the schedule of all tasks to as much left (i.e early) as possible without disturbing their relative positions in the schedule.

Exa: Determine a Feasible schedule for the real time tasks of a task set $\{T_1, T_2 \cdots T_5\}$ for which the precedence relations have been shown the below fig. for use with a table driven scheduler.

The execution times of the tasks $T_1, T_2 \cdots T_5$ are 7, 10, 5, 6, 2 & the

corresponding deadlines are 40, 50, 25, 20, 8



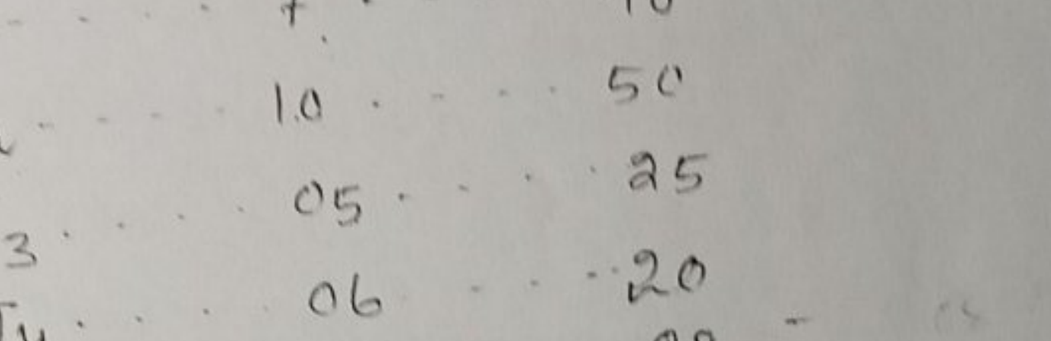
Ans:

Tasks	ex. time	deadline
T_1	7	40
T_2	10	50
T_3	05	25
T_4	06	20
T_5	02	08

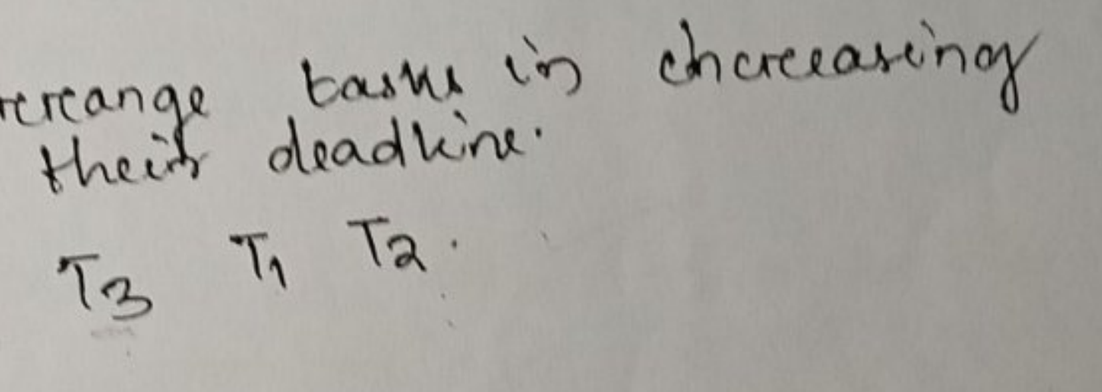
Step-1: Arrange tasks in increasing order by their deadline.

T_5 T_4 T_3 T_1 T_2.

Step-2



Step-3



CHAPTER-4

Scheduling Real-Time Tasks in Multiprocessor & distributed system

diff^n bet^n multiprocessor s/m and distributed s/m :-

Multiprocessor s/m	Distributed s/m
→ Multiprocessor s/ms are known as tightly coupled syetems.	→ Distributed s/ms are known as loosly coupled syetems.
→ In multiprocessor s/m, there is a shared physical m/r present.	→ There is no shared physical m/r present in a distributed s/m.
→ In this s/m, the interprocess communicat^n (IPC) is enexpensive & can be ignored as compared to task execut^n times.	→ It is not true in distributed s/m.
→ Multiprocessor s/m use a centralized dispatcher/ scheduler.	→ distributed s/m can not use centralized scheduler.

Multiprocessor Task Allocation

Multiprocessor Task Allocation Algorcithms are:-

1. Utilization Balancing Alg.
2. Next-fit Alg. for RMA
3. Bin packing Alg. for EDF.

1. Utilizat^n Balancing Alg:-

- This alg. mantains the tasks in a queue in increasing orders of their utilizations
- It removes task one by one from the head of the queue and allocates them to the least utilized processors each time.

(60) → The objective of selecting the least utilized processor each time is to balance utilizat^{n} of the different processores.

→ In a balanced system, the utilizat^{n} u_i, at each processor equals the overall utilizat^{n} of the processors $\bar{u}$ of the system.

→ If ST_i is the set of all tasks assigned to a processor P_i, then the utilizat^{n} of the processor P_i is $u_i = \sum_{j \in ST_i} u_{tj}$, where u_{tj} is the utilizat^{n} due to the task T_j.

→ The objective of any good utilizat^{n} ~~alg~~ balancing alg. is to minimize

$\sum_{i=1}^{n} | (\bar{u} - u_i) |$, where n is no. of processor in the system, $\bar{u}$ is average utilizat^{n} of processors, and u_i is utilizat^{n} of processor P_i.

② Next-Fit Alg. for RMA :-

→ In this alg., a task set is partitioned, so that each partition is scheduled on a uniprocessor using RMA scheduling.

→ It classifies the different tasks into a few classes based on the utilizat^{n} of the task. one or more processore are assigned to each class of tasks.

→ If the tasks are to be devided into m classes, a task Ti belongs to a class j, $0 \leq j < m$, if

$$(2^{\frac{1}{j+1}} - 1) \geq e_i / P_i \leq (2^{\frac{1}{j}} - 1)$$

class 1: $(2^{\frac{1}{2}} - 1) < c_1 \leq (2^{\frac{1}{1}} - 1)$

class 2: $(2^{\frac{1}{3}} - 1) < c_2 \leq (2^{\frac{1}{2}} - 1)$

class 3: $(2^{\frac{1}{4}} - 1) < c_3 \leq (2^{\frac{1}{3}} - 1)$

→ That is a

least
to
nt
ilizat
ie
rs ū
ssigned
t) of
, where
he
at) ~~alg.~~
processor
t) of
ssors Pi.
coned,
on a
a few
task.
; each
n
lasses,
2
,
0

(61) class 4: $0 < C_4 \leq \left(2^{\frac{1}{4}} - 1\right)$

→ So, From the above, the utilizat) greid for the different classes can be found to be:

class 1 : (0.41, 1)
class 2 : (0.26, 0.41)
class 3 : (0.19, 0.26)
class 4 : (0, 0.19)

(3) Bin Packing Alg. For EDF :-

→ This alg. attempts to allocate tasks to the processors such that the tasks on the individual processors can be successfully scheduled using EDF. This means that tasks are to be assigned to processors such that the utilizat) at any processor does not exceed 1.

→ We are given n periodic real time tasks either the individual processors are to be scheduled using the EDF alg., the no. bins necessary can be expressed as: $\left[\sum_{i=1}^{n} u_i\right]$. The bin packing problem is known to be NP-complete.

→ There are two bin packing alg.

1. First-Fit Random Alg:- on this tasks are selected randomly and assigned to processors in an arbitary manner as long as the utilization of a processor does not exceed 1.

2. First-Fit decreasing Alg:- on this, the tasks are stored in non-increasing order of their CPU utilizat) in an ordered list. The tasks are selected one by one from the ordered list and assigned to the bin to which it can fit in.

(62) DYNAMIC ALLOCATION OF TASKS:-

There are two dynamic real-time task allocat^{n} alg.

① Focussed Addressing and Bidding:-

② Buddy Alg.

1. Focussed Addressing and Bidding:-

→ In this method every processor mantains two tables called status table and system load table.

→ The status table of a processor contains information about the task which it has committed to run, including information about the execution time and periods of the tasks.

→ The system load table contains the latest load information of all other processors of the system.

→ This strategy incures high comm^{n} overhead in maintaining the system load table at the individual processors.

→ Window size is an important parameter determining the comm^{n} overhead incurred. If the window size is increased, then the comm^{n} overhead decreases.

② Buddy Algorithim:-

→ This alg. tries to overcome the high communication overhead of the focused addressing and bidding algorithim.

→ In this alg, a processor can be in any of the following two states. underloaded and overeloaded.

→ The states of a node is underloaded if its utilizat^{n} is less than some threshold value.

→ That is a processor Pi is said to be underloaded if ui < Th. The processor is said to be overloaded if its utilizat^{n} is greatere than the thresold value (i.e. ui >/ Th)

Clocks in Distributed Real-time systems

→ Clocks in a system are useful for two main purpose determining timeouts and time stamping.

→ Timeouts are useful to a real time programmer in a variety of siteuations, and its use includes determining the failiere of a task due to the missing of a deadline.

→ Time stamping is useful in several applications. ~~the~~ mostly use in message communication among tasks.

Clock synchronization:-

→ The goal of clock synchronization is to make all clocks in the network agree on their time values.

→ When the clocks of a system are synchronized with respect to one of the clocks of the system, it is called internal clock synchronization. When synchronizat^{n} of a set of clocks with some external clock is performed, it is called external synchronizat^{n}.

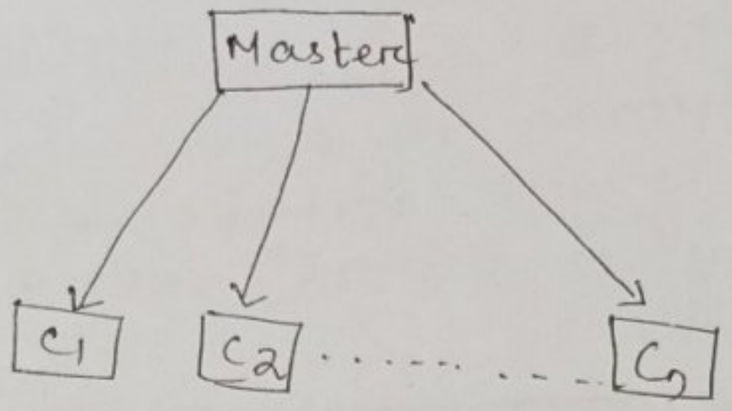
→ There are two main approaches for internal synchronizat^{n}.

1. Centralized clock synchronization
   and
2. Distributed clock synchronizat^{n}

1. Centralized clock synchronization.

→ In centralized clock synchronization, one of the clock is designated as the master clock. The other clocks of the s/m are called slaves and are kept in synchronization with the master clock.

→ The master clock is also sometimes called the time servere.



[Fig: centralized synchronization system]

→ The servere broadcasts its time to all othere clocks for synchronizat' after every ΔT time interval.

→ once the slave clocks receive a time broadcast from the mastere, they set their clock as per the time at the master clock.

→ The parameter ΔT should be carefully choosen. If ΔT is chosen to be too small, then the broadcast from the master is frequent and the slaves remain in good synchronizat" with the master all times, but high communication overhead is occeered.

→ If ΔT is choosen to be too large, then the clocks may drift too much apart.

Disadv.

Any failure of the master clock causes breakdown of the synchronization scheme.

Distributed clock synchronization :-

→ In distributed clock synchronization, there is no master clock with respect to which all slave clocks are to be set. But, all the clocks of a system are made to periodically exchange their clock readings among themselves.

→ Based on the receive time readings each clock in the system computes the synchronized time and sets its clock accordingly.

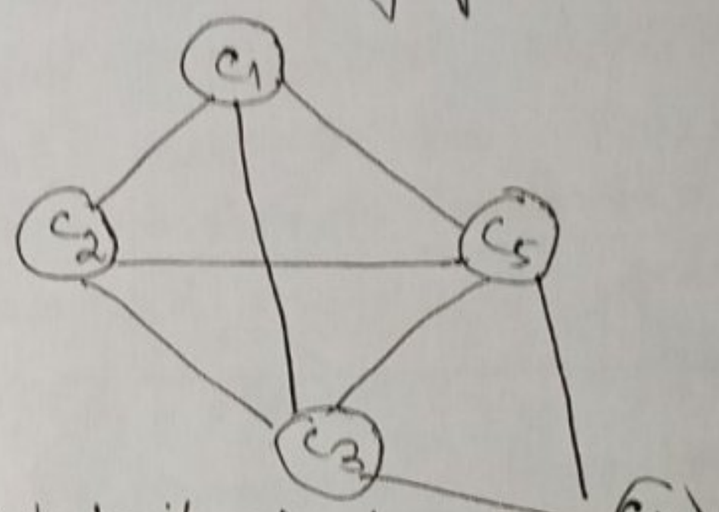
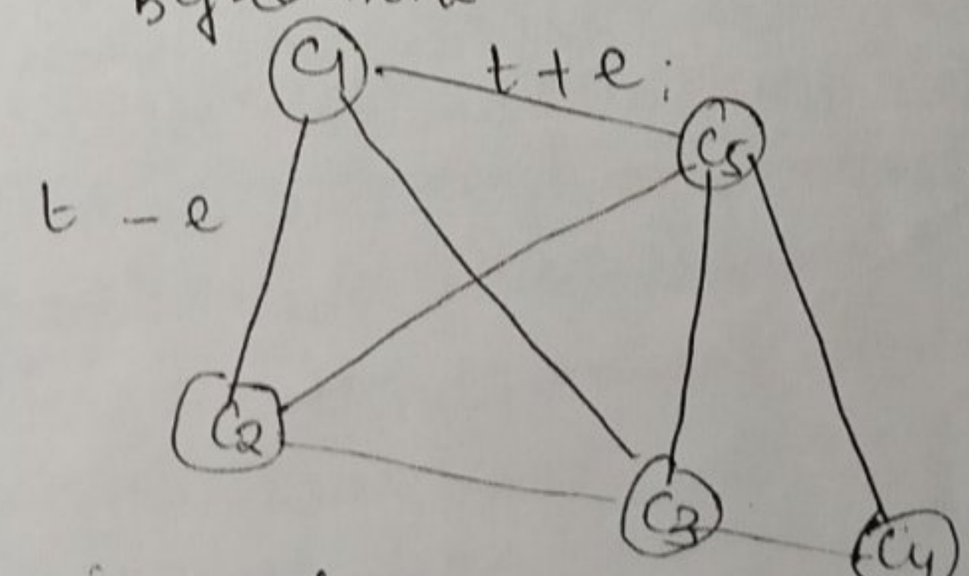


Fig: (Distributed clock synchronizat^n.

Byzantine clocks :-

→ A Byzantine clock is a two faced clocks.

⇒ It can transmit different values to different clocks at the same time.



[Byzantine clock]

→ In the above fig, C_1 is a Byzantine clock that is sending time values $t+e$ to clock C_5 and $t-e$ to clock C_2 at the same time instant

CHAPTER-5 MODULE-III

:Commercial Real Time Operating System:-

N.B:- RTOS are responsible for ensuring that every real time task meets its timeliness requirements.

Time Services:

→ Clocks and time services provided by the RTOS, since accurate & high precision clocks are very important to the successful operation of any real time application.

→ The time services provided by an OS are based on a s/w clock called the system clock. It is maintained by the operating system kernel based on the interrupts received from the h/w clock.

A. Clock interrupt Processing:-

Each time a clock interrupt occures, the handler routine carries out the following activities.

1. Process timer events:-

- RTOS maintain either per-process timer queues or a single system wide timer Queue.
- A timer Queue contains all timers arranged in order of their expiration times. Each timer is associated with a handler routine.
- The handler routine is the function that needs to be invoked when the timer expires.
- At each clock interrupt, the kernel checks the timer data structures in the timer queue to see whether any timer event has occured.
- If it finds that a timer event has occured, then it queues the corresponding handler routine in the ready Queue.

2. Update Ready List:-
- Since the occurrence of the last clock event, some tasks might have arrived or become ready due to the fulfillment of certain conditions they were waiting for.
- The tasks in the wait queue are checked to see if any task has become ready meanwhile. The tasks which are found to have become ready. are queued in the ready queue. If a task having higher priority than the currently running task is found to have become ready, then the currently running task is preempted and the scheduler is invoked.

3. Update execution budget:-
- At each clock interrupt, the scheduler decrements the time slice remaining for the executing tasks. If the remaining budget for the task becomes zero and the task is not yet complete, then the task is pre-empted and the scheduler is invoked to select another task to run.

B. Providing high clock Resolution:-
A way to provide sufficiently fine clock resolution is by mapping a hardware clock into the address space of applications. An application can then read the h/w clock directly through a normal memory read operation without having to make a system call.

Q) :

Task	Start time	Processing time	P_i	d_i
T_1	20	25	150	150
T_2	40	10	50	50
T_3	60	50	200	200

Sol

e_1	P_1	d_1	e_2	P_2	d_2	e_3	P_3	d_3
25	150	150	10	50	50	50	200	200

Necessary condition

$$\sum_{i=1}^{n} u_i \leq 1.$$

$$\Rightarrow \frac{25}{150} + \frac{10}{50} + \frac{50}{200}$$

$$\frac{1}{6} + \frac{1}{5} + \frac{1}{4} = \frac{10+12+15}{60} = \frac{37}{60} \leq 1.$$

$$= 0.61$$

Sufficient cond^n:

$$\sum_{i=1}^{n} u_i \leq n\left(2^{\frac{1}{n}} - 1\right)$$

$$\Rightarrow 0.61 \leq 0.78$$

Timers :-

RTOS support two types of timers.

(1) Periodic Timers :-

- These are used mainly for sampling events at regular intervals or performing some activities periodically.
- Once a periodic timer is set, it expires periodically.

(2) Aperiodic or One-shot timers :-

- These timers are set to expire only once.
- Exa :- Watchdog timers :-

These are used extensively in real time programs to detect if a task misses its deadline & then to initiate exception handling process upon a deadline miss.

```
f ()
{
  wd-start (t1, Exception-Handler); ← start
  :
  :
t1 :
  :
  wd-Tickle () ← End
}
```

Fig :- Use of a watchdog timer.

→ In the above fig, a watchdog timer is set at the start of a certain critical function f() through a wd_start(t1) call. which sets the watchdog timer to expire by the specified deadline from the starting of the task

→ If the fun' f() does not complete after t1 time units have elapsed, then the watchdog timer expires. which indicating that the task deadline is missed and the exception handling procedure is initiated.

→ In case the task completes before the watchdog timer expires, then the watchdog timer is reset using a wd-tickle() call

POSIX

(Portable Operating System Interface)

- The letter "x" has been suffixed to the abbreviation to make it sound like UNIX.
- It defines only interfaces to operating system services and the semantic of these services, but does not specify how exactly the services are to be implemented.
- The important part of POSIX are:-
  (1) POSIX.1: System Interfaces and system call parameters.
  (2) POSIX.2: shells & utilities.
  (3) POSIX.3: test methods for verifying conformance to POSIX.
  (4) POSIX.4: real-time extensions.

Real-Time POSIX standard:

- POSIX.4 deals with real time extensions to POSIX and is also popularly known as POSIX-RT.
- The main Requirements of the POSIX-RT are:

(71) taken ...

1. Execution scheduling.
2. Performance Requirements on System calls.
3. Priority levels.
4. Timers.
5. Real time files.
6. Memory locking.
7. Multithreading support.

Survey of Contemporary Real-Time Operating Systems :-

There are some popular Real-time operating systems that are being used in commercial applications.

(1) PSOS :-

- It is used in embedded applications.
- It is a host-target type of RTOS.
- It is used in several commercial embedded products exa :- in the base stations of cell phone systems.
- It supports 32 priority levels which can be assigned to tasks.
- It supports priority Inheritance and priority ceiling protocols, and segmented memory management but does not support virtual memory.

(2) VRTX :-

- It is a POSIX-RT compliant OS from Mentor Graphics.
- It is available in two multitasking kernels. VRTXsa and VRTXmc.
- VRTXsa is used for large & medium sized applications

- It supports virtual memory. (72)
- VRTX 32 is optimized for power consumption and ROM and RAM sizes and it does not support Virtual Memory.

VxWorks :-

- It is host-target type real time OS. & the host can be either a Windows or a UNIX machine.
- It conforms to POSIX-RT and comes with an Integrated Development Environment (IDE) called Tornado.
- Tornado contains VxSim and WindView.
- VxSim simulates a VxWorks target for use as a prototyping and testing environment in the absence of the actual target board.
- WindView provides debugging tools for the simulator environment.

QNX

- It is used in mission-critical applicats in the areas such as medical instrument, Internet routers, telemetric devices, process control applications, and air traffic control systems.
- It is implemented using a Microkernel architecture.

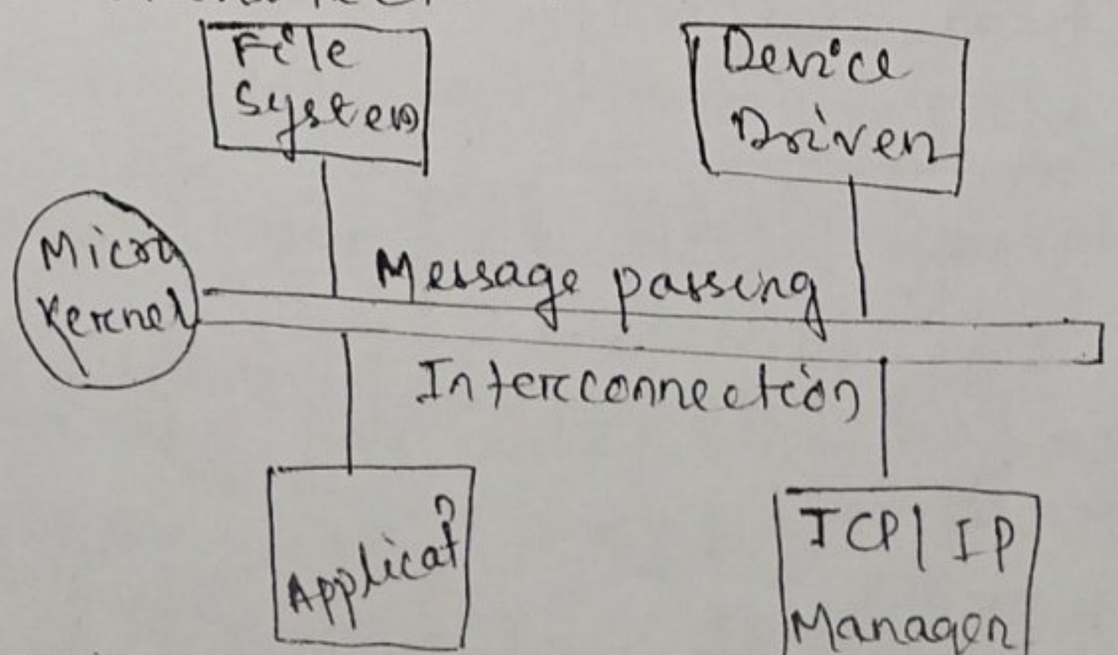


Fig. Microkernel Architecture of QNX.

(73) taken by ...

µC/OS-II :-

- It is written in ANSI C and contains a small portion of assembly code.
- It has been ported to over 100 different processor architectures ranging from 8-bit to 64-bit microprocessors, microcontrollers.
- It has a fully preemptive kernel.
- It allows upto 64 tasks to be created.
- It uses a partitioned memory management scheme.

Rhealstone Metric :-

There are six Metrics :-

(1) Task Switching Time (t_{tc}) :-
It is defined as the time it takes for one context switch among equal priority tasks.

(2) Task Pre-emption Time (t_{tp}) :-
It is defined as the time it takes to start execution of a higher priority task, after the condition enabling the task occures.

(3) Interrupt latency time (t_{il}) :-
It is defined as the time it takes to start the execution of the required ISR, after an interrupt occures.

(4) Semaphore Shuffling time (t_{ss}) :-
It is defined as the time that elapses between a lower priority task releasing a semaphore and a higher priority task to start running.

(5) Unbounded Priority Inversion Time (t_{up}) :-

- It is computed as the time it takes for the OS to recognize priority inversion (t_1) and run the task holding the resource and (start T_2 after T_1 completes (t_2)

$$t_{up} = t_1 + t_2$$

(6) Datagram throughput time (t_{dt}) :-

This parameter indicates the number of kilobytes of data that can be transferred between two tasks without using shared memory or pointers.

UNIT-4

(75)

Real-Time Communication:-

Real-Time communication is defined as one where an application can make specific quality of service demands such as maximum permissible delay, maximum loss rate, etc.

Difference betⁿ Hared, Firm & soft R.T.C:-

Hared - Real-time communication	Firm - Real-Time communication
→ In hared real time communication unless an absolute delay bound is met, the communication is assumed to have failed.	→ In a firm real-time commⁿ, if occasionally a message delayed, then the delayed message is discarded without affecting the system. ~~to any great extent.~~
→ Many of the industrial and embedded real-time applications require hard real time commⁿ.	→ such types of commⁿ are required in multimedia applicatns.

Soft Real-time commⁿ

→ It is best effort commⁿ.

Some examples in which Real-time communicatⁿ can be applied:-

→ Manufacturing Automation.
→ Automated chemical Factory.
→ Internet-Based Banking Applications.
→ Multimedia multicast.
→ Internet telephony.

Classification of Real-Time computer commⁿ n/w:-

There are 3 types of n/w are mainly used in Real-time communication.

① Controller-Area-Networks (CAN)
② Local-Area-Networks (LAN)
③ packet-switched-Networks.

① Controller Area Networks :-

→ It is a very small network.
→ It is used to connect different components of an embedded controller.
→ The end-to-end length of a CAN is usually less than 50 meters.
→ since, the propagation time of a CAN is very small, functionally a CAN behaves like a local Bus in a computer.
→ CAN specifies only the physical and data link layers of ISO/OSI model.
→ It is robustness. [If one link becomes unusable then it does not give impact to the entire system].
→ Application areas are industrial automation systems, trains, ships, agricultural machinery etc.

② Local Area Networks :-

→ This type of network is deployed in a building or a campus.
→ For example, a LAN can be used to connect a number of computers within an organization to share data and other resources such as a files, printers, FAX services etc
→ Generally LAN operate at data rates exceeding 10 ~~MPb~~ Mbps but presently it operates at 1 gbps.

③ packet switched Network :-

→ packet switching refers to protocols in which messages are divided into small units of data called packets.
→ Each packet is then transmitted individually and a packet can even follow a route different from that

taken by other packets to reach its destination.
→ The individual packets are routed through a network based on the destination address contained within each packet
→ once all the packets forming a message arrive at the destination, they are recompiled into the original message.
→ packet switched networks are usually wide-area networks.
→ Exa. Internet.

Quality of Service. (QoS)

The parameters of QoS are:-
1) Bandwidth 2) Maximum transmission delay.
3) delay jitter 4) loss rate and
5) Blocking probability.

① Bandwidth:-
→ It determines an application's maximum throughput and also determines the bounds on the end-to-end delay.
→ The higher the allocated bandwidth, the lower is the delay.

② Delay Jitter:-
→ Jitter is defined as the maximum variation in delay experienced by messages or packets in a single session.
→ or, it is the difference between the maximum and minimum delays that messages may encounter.
→ The size of the buffer required at the receive end would given by

peak rate x delay jitter.

③ Loss Rate:- Loss Rate denotes the percentage of all transmitted packets that may be lost during transmission.

④ Blocking Probability:-

Blocking probability is the probability of a new connection being rejected by the admission control mechanism of a network.

Traffic Categorization:-

There are commonly three traffic

① CBR traffic:-

→ This traffic arises due to constant bit rate data generation by a source.
→ Data generation and transmission involved in hard real-time applications are often CBR traffic.
→ In this case, fixed sized messages are transmitted over fixed intervals.

② VBR traffic:-

→ It consists of different rates of data transmission at different times.
→ VBR sources can be of several types.
→ Exa. 1) compressed audio signals generated by speech signals.
2) compressed video signals where variable sized data is generated periodically.

③ Sporadic traffic:-

→ It consists of a special type of variable sized packet transmissions.
→ In this, the packets are generated in bursts followed by long periods of silence.
→ Exa. traffic consisting of certain command, control & alarm msgs generated.

tage

(79) Real-Time communication on a LAN:-

↳ On a LAN, there is a single shared channel and only one node can transmit at any time.

↳ The exact time when a node can transmit on the channel is determined by the access arbitration policy of the n/w.

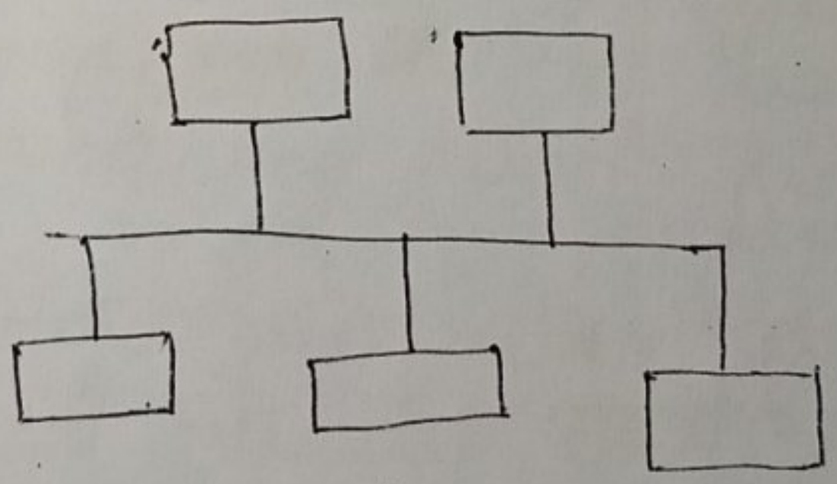
↳ The transmission control policy determines how long a node can transmit. These two policies are called the Access control techniques and form the media Access control (MAC) layers protocol.

LAN Architecture:-

There are two LAN architectures are used.

① Bus Architecture ② Ring Architecture

Bus Architecture:-



(Fig: A Bus interconnection network)

↳ In Bus based Architecture, nodes are connected to the n/w cable using T-shaped network interface connectors

→ Terminating points are placed at each end of the network cable.

→ There is a single shared channel (Bus) for which the transmitting nodes contend

↳ Nodes are communicated using broadcasting.
↳ The protocol used in BUS ntw is CSMA/CD (carrier sense multiple Access with collision Detection).
↳ CSMA/CD networks are also called multiple access networks.
→ In CSMA/CD networks, when two or more nodes transmit packets, the transmission overlap in time & the resulting signals get garbled. such an event is called as collision.

Ring Architecture:-

→ In a ring Architecture, the nodes are connected using MSAUs. (Multistational Access Units).
→ A MSAU is a hub or a connector that connects two or more no. of computers to the ring.
→ All the nodes are placed along the ring.
→ All the nodes are transmitted in turn.
→ Each node usually transmits a certain period of predetermined period of time.
→ Therefore, packet transmission delays are become predictable and can be made sufficiently small as per requirement. As a result, ring-based architectures are often preferred in real time applications.

- Its advantages :-

→ Undirectional traffic.

→ Any break in the ring can bring the whole network down.

→ It is a poor fit to the linear topology found in most assembly lines and other applications.

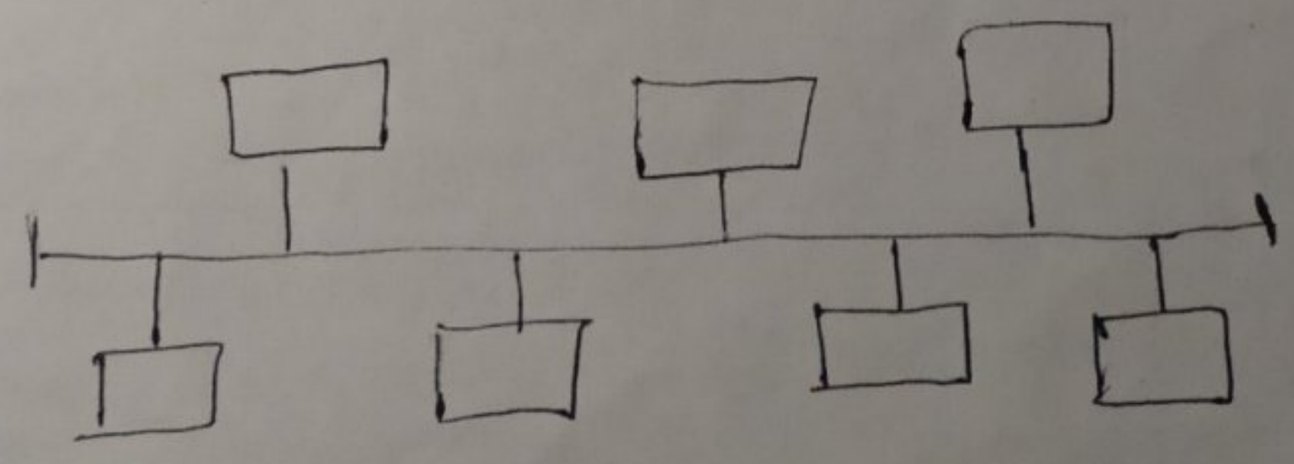
Token - Bus Architecture :-

→ A token bus is a bus based architecture, where the stations on the bus are logically arranged in a ring with each station knowing the address of the station to its left & right.

→ When the logical ring is initialized, the highest numbered station gets a chance to start its transmission.

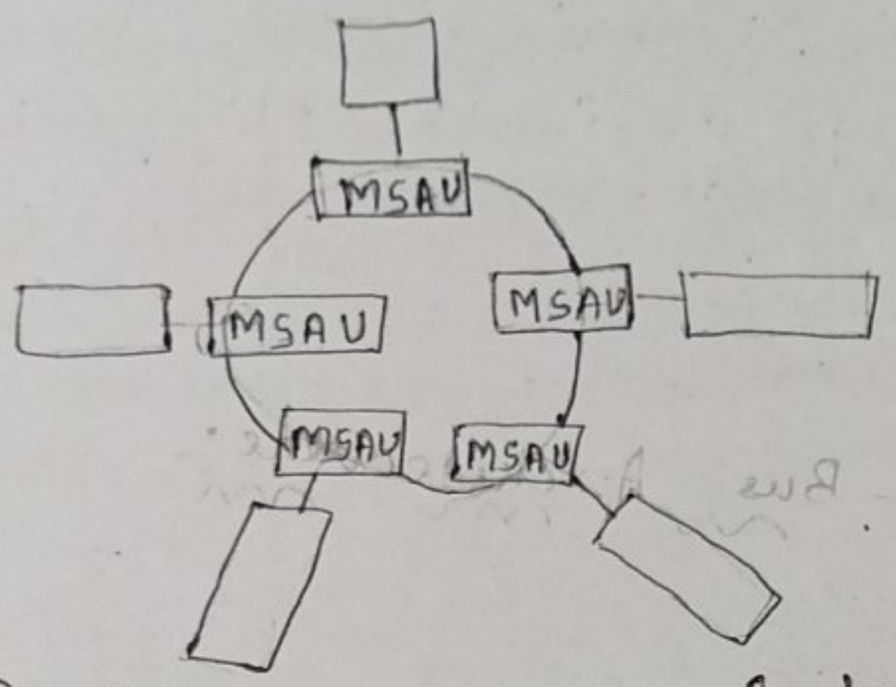
→ After transmitting for a predetermined duration, the station passes the transmission permission to its immediate neighbour by sending a special control frame called a token.

→ The token propagates around the logical ring. At any time, only the token holder is permitted to transmit packets. since only one station at a time holds the token in a ring n/w, collisions can not occure.



(Fig : A local ring in a Token Bus)

Fig of Ring Network



Soft Real-time communication in a LAN:

→ soft real time communications are not expected to provide any absolute QoS guarantees to the application.

→ It only ensures prioritized for messages so that message deadline miss ratio will be minimum.

— The techniques of soft Real time comm are.

① Fixed Rate traffic smoothing alg:-

— It is based on the considerations of limits on transmissing capacity of networks.

— The traffic smoother, which is placed between the MAC layer and TCP IP Layer, smooth a non-real time stream so that guarantees to real time messages coould not be violated.

— It used a leaky bucket algonithim known as credit bucket depth (CBD).

— It has two parameteres

① CBD

② Rp.

(83) CBD : (Credit bucket Depth)
- It is the maximum no. of credits that added to the bucket at every time
- or It is the maximum no. of credits that the bucket can hold.

RP : It is the Refresh period with which the bucket is replenished with new credits.

→ The ratio of CBD/RP is the average throughput of non real time messages.
→ The no. of credits present in the bucket at every time denoted by CNS (current network share).
→ The CNS may be +ve or -ve. It is -ve, if credits have been borrowed.
→ At every refresh, the no. of credits can be updated as

$$CNS = \min(CNS + CBD, CBD)$$

Disadvantage:
- It is not very flexible
- Increase in the no of stations, leads to a decrease in traffic generations limits of the station.

Adaptive Traffic smoothing:
→ The main idea behind adaptive traffic smoothing is that to provide reasonable throughput for non real time messages, in the presence of VBR real traffic sources, the non-real time traffic transmission rate can be allowed to adopt itself to the load condn of the underlying n/w.

→ That means, the nodes are allowed to increase their transmission limit if the utilizatn of the n/w is low.

→ when the utilizatn of the n/w high the nodes transmitting non real time messages are made to decrease their transmission limits.

– To implement such a rate adaptive traffic smoother, the two problems must be resolved.

① How to detect a change in n/w utilization.

② How to adopt the transmission limits to a detected change in n/w utilization.

– The solutn that has been proposed is that the no. of collisions per unit time can be used as a measure of n/w utilization.

Hard Real-Time communication in a LAN

– On a LAN, hard real time communicatn are normally supported using any of the following three classes of protocols.

① Global priority based protocols.

② Calendar-based.

③ bounded-access protocols.

① Global priority based protocol:-

→ In a global priority based protocol, each message is assigned a priority value

→ It ensures that, the channel is serving the highest priority message in the network

e allowed
ion limits
is low.
ow high
time
their

daptive
ems must

n/w

sion

n/w

posed
cerit
s

on a LAN

unicat',
ny of
cols.

protocal.
ity
is
age

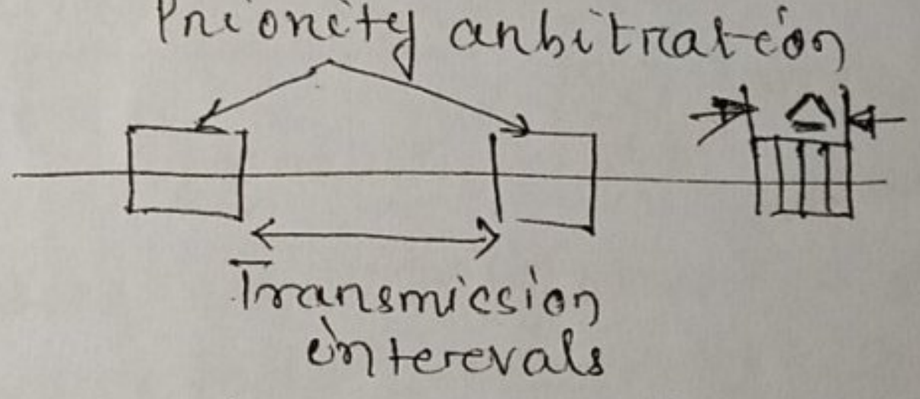
(8) → Global priority based protocols are
(a) count-down protocol
(b) Virtual-Time protocol.
(c) IEEE 802.5
(d) Window-based protocol.

@ count-down protocol:-

→ In this protocol, the timeline is devided into a fixed size intervals called slots.

→ At the start of every slot, the priority arbitration is carried out to determine the heighest priority message on the network.

→ As soon as the priority anbitration is complete, the node having the heighest priority message is allowed to transmit.



How priority arbitration is achieved in count-down protocol?

- The priority anbitration in countdown protocal is achieved in the following ways.

→ Over each slot, every node that has a pending message transmits the priority value of its highest priority pending message with MSB first

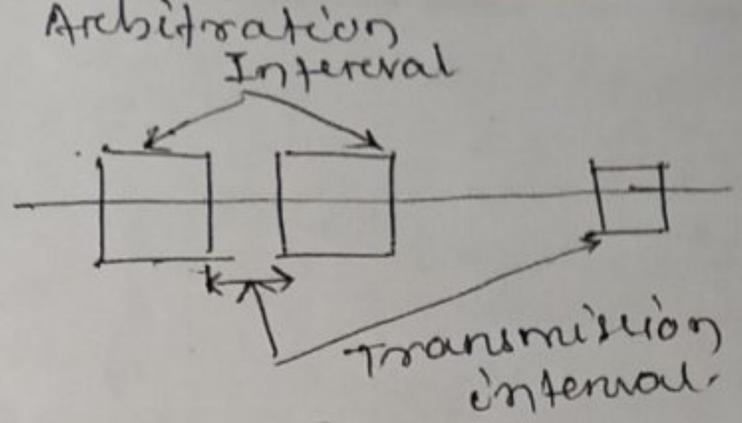
- Since simultaneous transmission follows, a node transmits 0 and receives 1, knows that there atleast one high priority pending message, and drops out of the contention.

of

d
face

(24) - The node that transmits last without any collision, concludes that no nodes have any priority message and can begin its transmission.
- An important parameter in the efficient working of this protocol is the slot size. So, the slot size should be selected carefully.
- Each slot size is made equal to the end-to-end propagation delay of the medium.
- If the slot size made smaller, then the priority arbitration scheme would not work since when a collision occures, it would not be detected.
- If the slot size made larger, this lead to an increase in channel idle time during every slot.

(b) Virtual Time protocol:-
- In this protocol, a node uses the state of channel to reason about the pending packets residing at other nodes.
- Each node with a packet to send waits for an interval of time & inversely proportional to the priority of the highest priority message it has.
- This protocol assigns priority to each nodes. The priority of a node is equal to the highest priority message it has.
- If the channel is busy, then this would imply that a higher priority message is being transmitted & it would need to wait until an idle period.



→ Assume that two nodes N_1 and N_2 having their priorities differ by 1.
→ After N_1 stants transmitting, if N_2 waits the time shorter than the propagation time, then it can not be detected the transmission of N_1 and would stants transmitting. This would lead to a collision and result in incorrect priority anbitration
— Thus propagation time effectively bound the difference between the wait times of two nodes that have consecutive priorities.

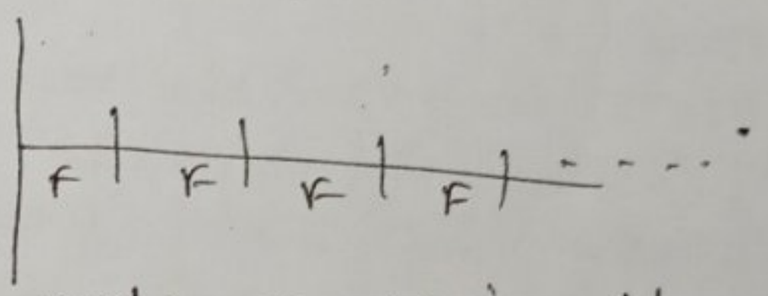
⑥ IEEE 802.5 :-
→ It is a priority based token-ring protocol.
→ In this protocol, the header of the node is devided into two fields.
(1) Reservation field & (2) mode field.
— A node can any time determine the mode of the token by examining the mode bit in the header of the token.
— In this protocol, the messages to be transmitted are devided into number of frames. The token contains a frame as a payload
→ It supports assigning priorities to the messages. The priority of the message that is being transmitted is recorded in the reservation field of the header.
→ As the token passes through the ring, every node with pending message inspects the reservation field in the token header.

- A node having a higher priority message than that it is being transmitted registers its preority in the priority field of the header.
- When the token returns to the sending node, it observes the reservation mode by taking another node. puts the token in the free mode & releases it.
- As the free mode token passes through the ring, the node that made the reservation seizes the token, puts it onto the reservation mode & starts transmitting.

④ Window based protocol:-

→ This is also a global priority based protocol.

→ In this protocol, the timeline is devided into frames.



→ Each node maintains the current transmission by defining the priority value (low, high).

→ A node that has a message in window, can start transmission.

→ At the start of every frame, there may be more than one node can eligible for transmitting & transmitting it. So the result becomes collision.

→ On a collision, every node increments the value of low & on a free frame every node decrements the value of low

② Calender-based protocol:-
→ on this protocol, each node mantains a calender data structure to where information about the access time reservations of all guaranteed messages of all nodes are mantained.
- when a message for which no reservations has yet been made arrives at a node, the node first determines a free slot by consulting its local data structure.
- it then attempts to reserve a suitable free future time interval by broadcasting a message to all nodes. Every node on receipt of a control messages, updates its calender accordingly.
- it becomes very difficult to handle a periodic & sponadic messages in this protocol. Therefore this protocol is used only in very simple n/ws such as CANs.

③ Bounded Access protocols For LAN:-
- There are two protocols used in this.
① IEEE 802.4
② RETHER
④ IEEE 802.4:-
- it is used in token ring and token bus network.
- sometimes it is referred as timed token protocol.
- In this protocol, a node transmits. it it holds the token.
- This protocol has been incorporated in FDDI (Fiber Distributed Data Interface)
TTRT: (Target token Rotation Time):-
- it is defined as the expected time b.tn two consecutive visits of the token to a node.

(9a) - It is used as a design parameter.
- On the network utilization, it is specified by network administrator depending upon the character of the message that would transmit over the network.

Synchronous Bandwidth in TTRT:
- Individual nodes are allocated in a portion of TTRT, known as Synchronous Bandwidth.
- This allotment is made according to the timing characteristics of synchronous message originating at all other nodes taken together.
- Let the synchronous bandwidth of a node Ni be Hi unit time.
- Each time a token visits a node Ni, it can transmit its synchronous message for at most Hi unit.
- Let SN be the set of all nodes on the n/w.
- Then TTRT can be represented as

$$TTRT = \Theta + \sum_{N_i \in SN} H_i$$

, where Θ is the propagation time. and Hi is the token holding time at node Ni.
- When a node receives the token. It first transmits its synchronous traffic for a time bounded by its synchronous bandwidth.
- After transmitting synchronous traffic, it transmits asynchronous traffic only.

(91) if the time since the previous departure of the token from the same node is less than TTRT.
- The time for which asynchronous frames are transmitted is called asynchronous overrun.
- It reduces the effective bandwidth available to transmit asynchronous messages & delays the time betn consecutive arrival of the token at a node.
- Due to asynchronous overrun, the worst-case time betn two successive visits of the token to a node is

$$2 \times TTRT$$

- The synchronous bandwidth allocated to a node N_i is given by the following expression

$$H_i = (TTRT - \Theta) * \frac{C_i / T_i}{\sum C_i / T_i}$$

where C_i is the size of the message (in bits) that node N_i requires to transmit over T_i interval.
C_i / T_i is the channel utilization due to node N_i.

@ RETHER :-

Real Time Communicat? over Packet Switched N/W :-

Module - III

Real Time Database management System:

Use or Application of RDBMS:

Real Time Database can be used in.

1) Network management system.
2) Industrial control system.
3) Autopilot system.
4) Internet service management.

ACID properties:-

1) Atomicity: Either all or none of the operations of a transcation are performed
2) consistency: A transcation need to mantain the integrity constraint on a database.
3) Isolation: Transcations are executed concurrently as long as they do not interfere with each others ~~transcation~~ computations
4) Durability: All changes made by a committed transcation become permanent on the database.

Real-Time vs Traditional database:-

The main difference bet" the traditional database and the conventional database lie on the following 3 characteristics.

① temporal data.
2) Timing constraints on the database operations.
3) Performance Metrics.

① Temporal data:-

data whose validity is loss after some prespecified time interval is called temporal data.

⑧ Timing constraints on the database oper
Tasks and Transactions are similar
abstractions in the sense that both are
the unit of work and as well as schedul
A transaction execution requires so
many data records in an exclusive mode.

⑤ Performance Metrics:-
The most commonly used performance
metrics for real as well as non-real time
is transcation is transcation response tim
It is the no. of transactions missing their
deadlines per unit time.

characteristics of Temporal data:-

Temporal Consistency:-

-Temporal consistency of data requires the
actual state environment and the state
represented by the database.
- It has two main requirements.
(1) Absolute validity.
(2) Relative Consistency
④ Absolute validity:- This is the notion
of the consistency between the environment
and its reflection in the database
given by the collection of data collected by
the system about the environment.
② Relative consistency:- This is the
notion of consistency among the data
required that are derive new data.

(14) How to represent data items in a Real time database :-?

- A data items 'd' can be represented in a real time database as a triple.

d (value, avi, timestamp)

- The three components are denoted as

(dvalue, davi, dtimestamp)

- where dvalue - is the value recorded for 'd'.

davi - absolute validity interval of 'd'

dtimestamp - time when measurement of 'd' takes place.

Condition for absolute validity:-

A data item d is absolutely valid if

(current time − dtimestamp) ≤ davi

condition for Relative Consistency:-

A set 'd' of data items is relatively consistent if

|dxtimestamp − dytimestamp| ≤ dRvi

where dRvi is the relative validity of 'd'

Concurrency control in Real Time database:-

→ The main idea behind the concurrency control is to ensure the non-interference (isolation and atomicity) among different transaction by serializability.

→ A concurrent execution of a set of transaction is said to be serializability if the database operation carried out by the given data is equivalent to some series of execution of these transaction.

- Types of concurrency control are:-

(95) ① Locking Based concurrency control.
② optimistic concurrency control (OCC)
③ speculative concurrency control (SCC)

1- Locking Based concurrency control:-
There are so many protocols under this control.
ⓐ 2PL
ⓑ 2PL-(WP)
ⓒ 2PL - HP
ⓓ priority ceiling protocol. (PCP)

ⓐ <u>2PL</u> :- [Two phase locking]
- This type of protocol restricts the degree of concurrency in a database.
- It has two main phase.
(i) Growing phase.
(ii) Shrinking phase.
- In Growing phase, a transaction acquires the lock on the desired data items.
- In the shrinking phase, a transaction release the lock.
- Once, a lock is released by a transaction, its shrinking phase is start and no further lock is acquired on the database.

<u>Strict 2PL</u>:
- A strict 2PL is implemented in the commercial database.
- The main restriction is that, the transaction can not release any lock until ~~its~~ after its termination.

(96) Is the conventional API is suitable for Real-Time Application?

Ans:- The conventional API is unsatisfactory for Real time applications for the following reasons.

1. priority inversion.
2. Long blocking delays.
3. Lack of considerations for timing information.
4. Deadlock.

1. possibility of priority inversion:-
A priority inversion can occure, when a low priority transaction holding a lock on the data item and a high priority transaction needs the data item & waits until the low priority transaction it's release.

2. Long Blocking delays:
A transaction might go long blocking delays, since any other transaction which might acquires the data before it would hold it till its completion.

3. deadlock:
Consider two transactions T_1 and T_2 holding the data item 'd1' and 'd2'. and, consider the following sequences.

T_1: lock d1, lock d2, unlock d2, unlock d1

T_2: Lock d2, lock d1, unlock d1, unlock d2

Assume that T_1 is the higher priority than T_2. let T_2 starts running and lock d2 ~~and then lock d1~~. when T_1 starts running, it lock d1 & then tries to lock d2 which is held by T_2. Then T_2 tries to lock d1 which is held by T_1.

(97) (b) 2PL wp (wait promote) :-

-pseudocode for this protocol is

```
if (Pri(TR) > Pri(TH))
then
    TR waits;
    TH inherits the priority of TR;
else
    TR waits;
end if.
```

- from the above pseudocode, it is observe that. when a high priority transaction request the lock on the data item, which is held by the low priority transaction, then the low priority transaction inherits the priority from high priority transaction

(c) 2PL-HP (High priority) :-

- on this protocol, when a transaction request a lock on the data object which held by any low priority transaction in conflict mode, then the lock holding the low priority transaction is aborted!

- This protocol is also known as priority Abort (PA).

- This protocol is free from priority inversion as well as deadlock.

- Pseudocode:-

```
if (no conflict) then TR access
    else
if (Pri(TR) > Pri(TH)) then TH abort.
    else
        TR waits for locking.
```

(98) (1) priority ceiling protocols:
- The main concept behind this protocol is that establishment of total priority ordering among all transactions.
- It associates the following three values to the data object.
① Read ceiling.
② Absolute ceiling.
③ Read-write ceiling.
① Read-ceiling:-
This indicates the priority value of high priority transaction which may write to the data object.
② Absolute ceiling:-
This is the highest priority transaction that may read and write on the data object.
③ Read-write ceiling:-
- Its value defines dynamically at run time.
- A read-write ceiling is set equal to absolute ceiling, when a transact" needs to write operat".
- A read-write ceiling is set equal to Read-ceiling, when a transact" needs to read operh.

② Optimistic Concurrency control:-(OCC)
- It does not prevent any transaction that it access any data items it requires.
- It can also deride into 3 parts.
① Foreward OCC.
② OCC-Broadcast commit.
③ OCC-sacrifice.

(99) ① Forward occ:

- On this protocol, each transactions read and update data items freely.
- When the update data items are stored on a private work place.
- These data items are made public during the transaction commited time.
- Before a transaction allowed to commit, it goes for validation test.
- The validation test checks whether there is any conblict bet' the committed transaction and other transaction.
- A transaction is abont, if it does not pass the validation test.

② OCC Broadcast commit:

- On this protocol, coher a transaction committs, it notifies all its intention to other transaction that currently running.
- Each of these transaction are carries out a test to check cohether ~~there~~ it has any conblict with the committed currently transaction.
- If it detects any conflict, then the committed transaction abont itself and restanted.
- On occ-BC, once a transaction reaches its validatation stage, it is guanantees to committed.

③ OCC- sacrifice:

- This protocol consideres the priority of transactions and also introduce a concept of conblict set.

read
ned
mit,
n
itted
es not
nsaction
on to
nning.
then
the
the
and
it is
of
ncept

(100) - A conflict set is a set of concurrent currently running transaction that conflict with the validated transaction.

(3) Speculative concurrency protocol:- (SCC)
- on SCC, conflicts are checked at every read and write operations.
- whenever a conflict is detected, a new version (called the shadow version) of each of these transactions initiated.

Comparision of Different protocols:-
→ locking based algorithims tend to reduce the degree of concurrent execution of trancaction as they construct serializable schedules.
→ optimistic approaches attempt to increase parallelism to its maximum.
→ occ protocols have the advantage of being non-blocking and free from deadlock.