# UNIT-IV REAL TIME SYSTEMS

## Structure of Real Time System

⇒ Real time system is an example of a general purpose operating system. This system is used when the requirements are inelastic for data flow or operations of processors, So, it is often used as a control device in dedicated applications.

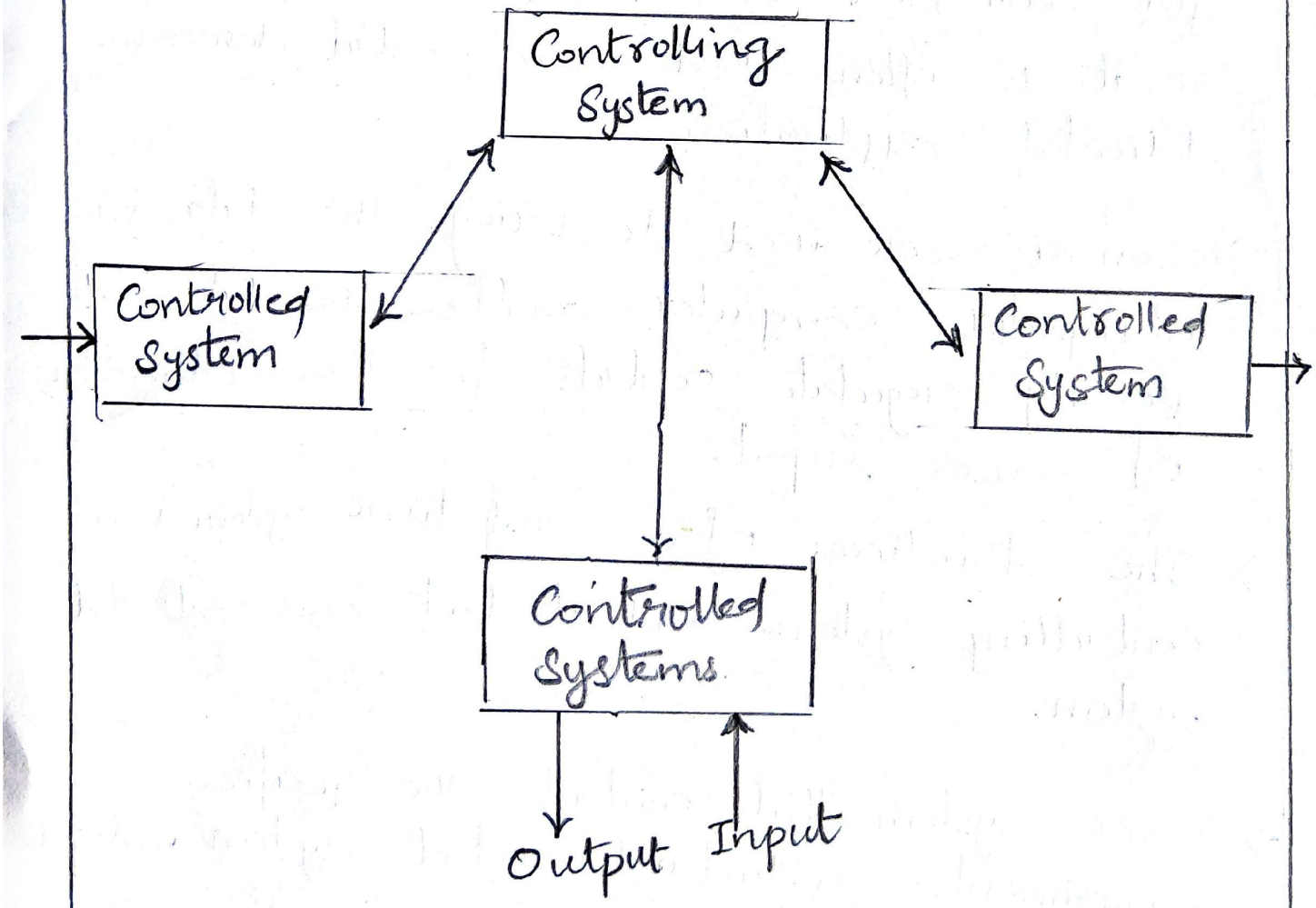
⇒ Sensors are used to bring the data in computer. Computer analyze the data and possibly regulate controls for the modification of sensor input.

⇒ The structure of a real time system is a controlling system and at least one controlled system.

⇒ Some system that control the specific experiments, industrial control system, medical imaging system and some display systems are different forms of real time system.

⇒ It also contains some systems like automobile engines fuel injection systems, weapon systems and home appliances controller.

⇒ In real time system there is fixed time constriants. Processing is required to be done within defined contraints otherwise the system will fail.




⇒ The functionality of the real time system is considered to be correct only if it returns the correct results with in any time constrains.

⇒ Control a device using actuator, based on sampled sensor data. It also control loop compares measured value and reference value. Reference input, accuracy of measurements depends on correct control law computation.

⇒ Time between measurements of $y(t)$, $r(t)$ is the sampling period, T. small T gives better approximates analogue control but large T needs less processor time: if T is too large, oscillation will result as the system fails to keep up with changes in the input.

⇒ Analysis of a control system involves the determination of the system response. This can be carried out experimentally, or by estimating the response on the basis of a system model. Then, it is the task of control system design to achieve a desired overall system response by modifying the controller

⇒ The controller will have its control strategy upon the disturbed signal and might drive the plant into an undesire state. In many designs, the sampler is preceded by an anti-alias filter to avoid the effect of aliasing.

## Hard Real Time System:

⇒ A hard real-time system is one where the response time is specified as an absolute value. This time is normally dictated by the environment.

⇒ A system is called a hard real-time if tasks always must finish execution before their deadlines or if message always can be delivered within a specified time interval.

⇒ Hard-real time is often associated with safety critical applications.

⇒ Missing a deadline may be catastrophic. Critical deadline is called hard deadline.

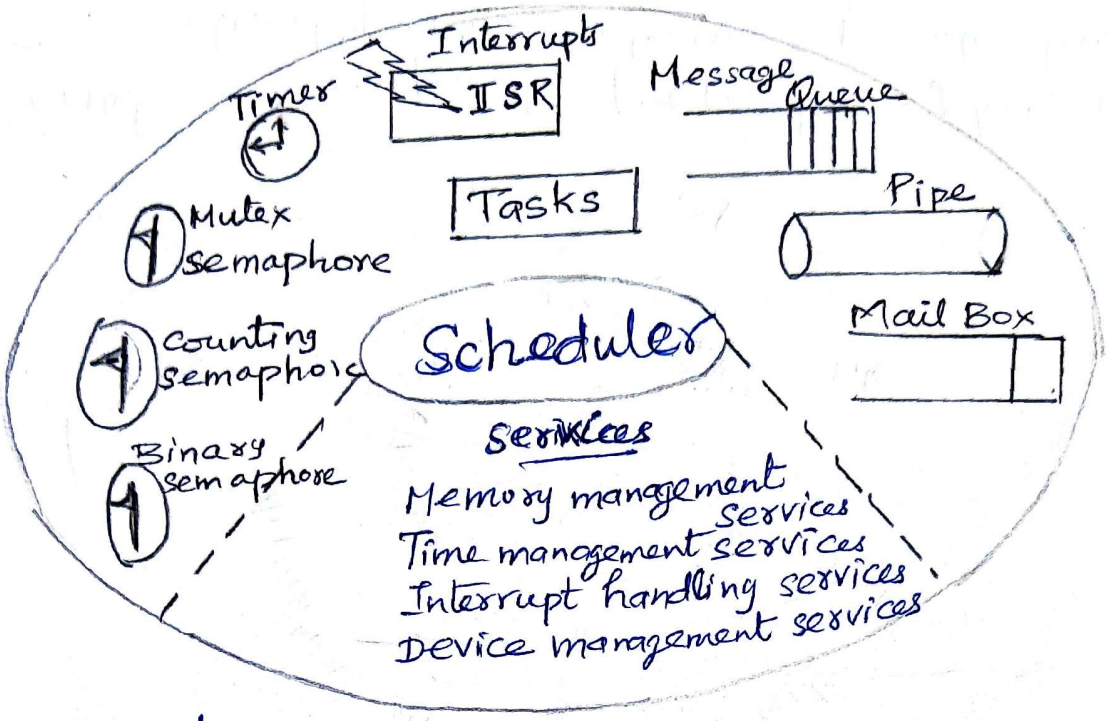
## Soft Real Time System:

⇒ A soft real-time system is one where the response time is normally specified as an average value. The time is normally dictated by the business or market.

⇒ A single computation arriving late is not significant to the operation of the system, though many late arrivals might be.

⇒ Soft real time means that only the precedence and sequence for the task-operations are defined, interrupt latencies and context switching latencies are small but there can be few deviations between expected latencies of the task & observed time constraints and a few deadline misses are accepted.

# KERNEL




## Kernal

The various objects of kernel are Tasks, Task scheduler, Interrupt service Routines, Semaphores, Mutexes, Mail Boxes, Message Queues, Pipes, Timers etc,

## Scheduler

The schedular is the heart of every kernel. A scheduler provides the algorithms needed to determine which task executes when.

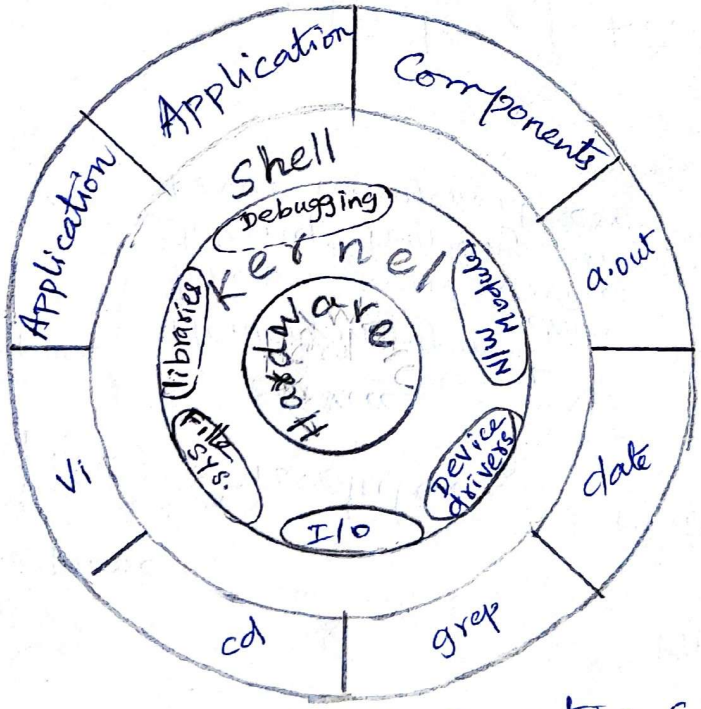
**Schedulable Entities:** It is a kernel object that can be completed for execution time on a system based on predefined scheduling algorithm. Tasks and Process are example of schedulable entities found in most kernel.

# RTOS Architecture

A real-time OS is a program that schedules execution in a timly manner, manages system resources, provides a consistant foundation for developing application code.

RTOS is a multi-tasking OS intended for real time applications to produce high and guaranteed throughput on disired time.

**Hardware** : It consists of all peripherals, Processor & Memory




**Kernel**: Core components of Operating System, interact directly with hardware & provide low level services to layer components

**shell**: An interface to kernel, hiding complexity of kernel's function from users. Takes commands from user & executes kernel's functions.

# UNIT-IV REAL-TIME SYSTEM

## REAL-TIME CHARACTERISTICS

Real-Time System is one whose logical correctness is based on both correctness of the outputs and their timeliness

⇨ Real-time systems are those systems in which the overal correctness of the system depends on both the functional correctness and the timing correctness.

⇨ Real-time system also have a substantial knowledge of the system it controls and the applications running on it.

⇨ Deadline dependent.

⇨ Predictability is important.

⇨ Deadline – a time within which the task should be completed.

⇨ Hard Real-Time System – system fails if deadline window is missed
Ex: aircraft control

⇨ Soft Real-Time System – system will be undesirable for performace reason if deadline window is missed.
Ex: multimedia applications

⇨ Firm deadline – Missing a deadline makes the task useless (similar to hard RT), however the deadline may be missed occasionally (similar to soft deadline).

⇨ Most systems : combination of both hard & soft deadlines

⇨ generalization : cost function associated with missing each deadline.

## Characteristics of RT systems :

⇨ Real-time systems are often Embedded systems.

⇨ They often require concurrent processing of multiple inputs.
- concurrent task must be created & managed in order to fulfill the functions of the system.

⇨ Task scheduling is one of the important aspects of managing concurrency
- Since tasks will compete for the same resources (such as processors)

⇨ Real-time systems need to respond to Synchronous events (i.e. periodic events) as well as asynchronous events (i.e. aperiodic events).

⇨ Real-time systems often requires high Reliability & safty requirements.

⇨ Environmental factors such as temperature, shock, vibration, size limits & weight limits usually have an impact on the system hardware & software requirements.

⇨ Fault-tolerant requirements & Exception handling have special consideration due to the high reliability & critical timing requirements.

⇨ Interfacing requirements. The devices which are typically interfaced to a RTS.

⇨ Guaranteed response times

- We need to able to predict with confidence the worse case response times for system: efficiency is important but predictability is essential.

⇨ Job; unit of work that is scheduled & executed by the system

- Computation of a FFT [Fast Fourier Transform]
- Transmission of a data packet

⇨ Task; a set of related jobs which jointly provide some system function.

⇨ Release time;

- The instant of time at which the job becomes available for execution
- Job have no release time if all the jobs are released when the system begins execution

⇨ Response time;

- The length of time from the release time of the job to the instant when it completes

⇨ Relative deadline:

- The maximum allowable response time of a job

⇨ Deadline or Absolute Deadline;

- The instant of time by which its execution is required to be completed
- Equal to the release time plus the relative deadline.
- A job has no deadline if its deadline is at infinity.

• The set of rules that determines the order in which tasks are executed is called a scheduling algorithm.

- A schedule is feasible if all tasks can be completed according to the timing constraints
- A set of tasks is schedulable if there exists at least one algorithm that can produce a feasible schedule.

⇨ A Periodic task is executed repeatly at regular time intervals and each invocation is called a job or instance
- often time-driven

⇨ A Aperiodic task is executed to response to external events and to respond, it executes aperiodic jobs whose release time are not know a priori.
- Often event-driven
- Off-line guarantee of aperiodic tasks must make proper assumption on the environments; that is, by assuming a maximum arrival rate for each event (i.e. minimum interarrival time)
- Aperiodic tasks characterized by a minimum interarrival time are called Sporadic tasks.

Types of Scheduling

⇨ Preemptive: the running tasks can be interrupted to assign the processor to another task.

⇨ Non-preemtive; A task, once started, is executed until completion.

⇨ Static; the scheduling decisions are based on fixed parameters and assigned to tasks before their activation

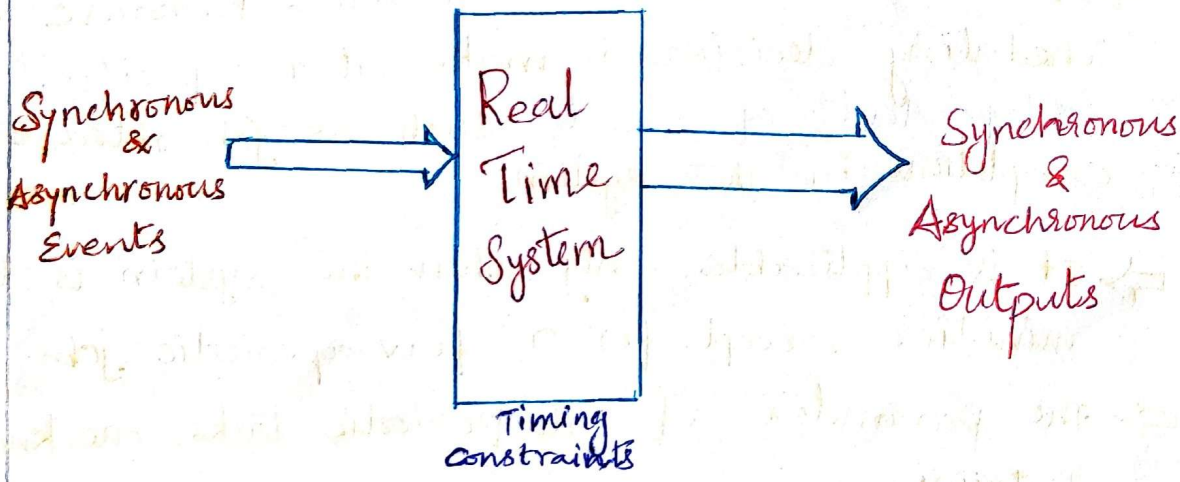
⇨ Dynamic; the scheduling decisions are based on dynamic parameters that may change during system evolution.

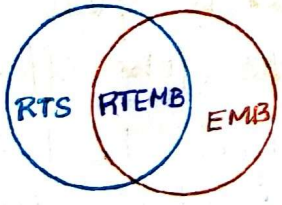
⇒ Off-line :- a scheduling algorithm is executed on the entire task set before actual task activation
- The schedule may be stored in a table and later executed by a dispatcher.

⇒ On-line : scheduling decisions are taken at runtime every time a new task enter the system or when a running task terminates.

⇒ Optimal : an algorithm is optimal if it minimizes some given cost function defined over the task set.

⇒ Heuristic : an algorithm is heuristic if it tends towards but does not guarantee to find the optimal schedule.




Railway monitoring & scheduling : RTS

Cell phone : EMB

Heart pacemaker : RTEMB

# Approaches to Real-Time Scheduling

* Clock-Driven Approach
* Weighted Round-Robine Approach
* Priority-Driven Approach
* Dynamic versus Static Systems
* Effective Release Times & Deadlines
* Earliest Deadline First (EDF) Algorithm
* Least Slack Time First (LST) Algorithm
* Validation of schedules

## CLOCK-DRIVEN APPROACH

⇨ It is also called time-driven. Because each scheduling decision is make at a specific time, independent of events, such as job releases or completions in the system

⇨ It is applicable only when the system is deterministic, except for a few aperiodics jobs.

⇨ The parameters of all periodic tasks are known a priori.

⇨ Therefore, the scheduler is often constructed by a static schedule of the jobs off-line.

- Use a hardware timer to trigger the scheduling decision. The timer is set to expire periodically without intervention of the scheduler.

⇨ When the system is initialized, the scheduler selects & schedules the jobs that will execute until the next scheduling decision time & then blocks itself waiting for the expiration of the timer.

⇨ When the timer expires, the scheduler awakes & repeats these actions.

## Round-robin approach

⇨ Generally used for scheduling time-shared applications.

⇨ The jobs are scheduled in a round-robin system so that every job joins a first-in-first-out (FIFO) queue

⇨ The job at head of the queue executes first at most one time slice. Otherwise it is pre-empted and placed at the end of the queue to wait for its next turn.

⇨ When there are 'n' ready jobs in the queue, each job gets one time slice in the total time (n), in every round.

⇨ Each job gets $\frac{1}{n}^{th}$ share of the processor when there are 'n' jobs ready for execution. Hence it is known as processor-sharing algorithm.

## Weighted round-robin approach

⇨ The weighed round-robin algorithm has been used for scheduling real-time traffic in high speed switched networks.

⇨ Rather than giving all the read jobs equal shares of the processor, different jobs may be given different weights.

⇨ weight = the time slice allocated to the job.

⇨ A job with weight Wt get same time slice in every round.

⇨ The length of the round equals the sum of the weights of all the ready jobs.

⇨ By adjusting the weights of jobs we can speed-up or slow-down the progress of each job towards its completion.

⇨ By giving each job a fraction of the processor, a round robin scheduler delays the completion of every job.

⇨ If it is used to schedule priority based jobs, the response time ~~is not suitable for scheduling~~ of a chain of jobs can be very large.

⇨ For this reason, the weighted round-robin approach is not suitable for scheduling such jobs.

⇨ But for Unix pipe, weighted round-robin scheduling may be a reasonable approach, since a job & its successors can execute concurrently in a pipelined fashion.

Example

- For example consider two sets of jobs

  $J_1 = \{J_{1,1}, J_{1,2}\}$ & $J_2 = \{J_{2,1}, J_{2,2}\}$

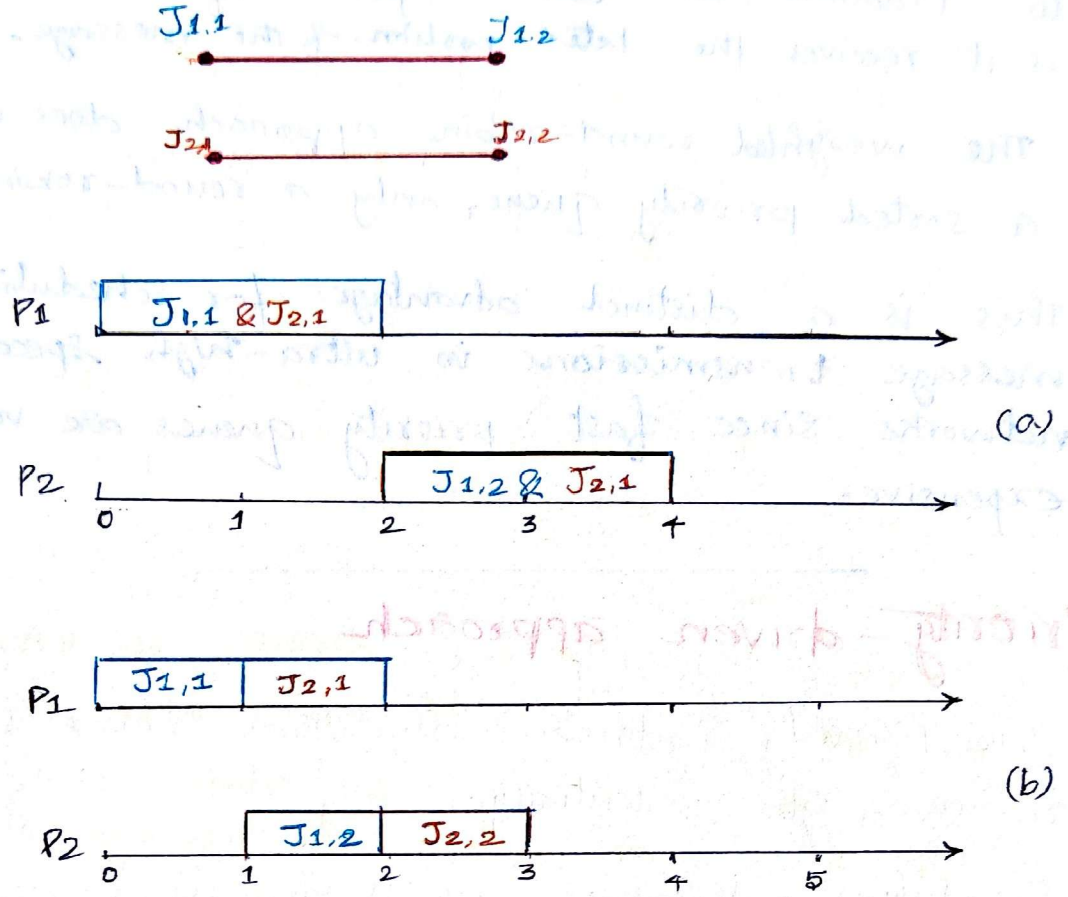
- The release times of all jobs are 0
- The execution times of all jobs are 1

  $J_{1,1}$, $J_{2,1}$ executes on processor $P_1$

  $J_{1,2}$, $J_{2,2}$ executes on Processor $P_2$

- Suppose that $J_{1,1}$ is the predecessor of $J_{1,2}$ & $J_{2,1}$ is the predecessor of $J_{2,2}$

Example Illustration of Round-robin scheduling of precedence-constrained jobs.




* Figure (a) shows that both sets of complete approximately at time 4 if the jobs are scheduled in a weighted round-robin manner.
* In contrast, if the jobs on each processor are executed one after another, one of the chains can complete at time 2, while the other can complete at time 3.
* Suppose that the result of the first job in each set is piped to the second job in the set. The latter can execute after each one or a few time slices of the former complete.
* Then the round-robin approache is better because both sets can complete a few time slices after time 2.

* In a switched network a downstream switch can begin to transmit an earlier portion of the message as soon as it receives the later portion of the message.

* The weighted round-robin approach does not require a sorted priority queue, only a round-robin queue.

* This is a distinct advantage for scheduling message transmissions in ultra-high speed networks since fast priority queues are very expensive.

---

## Priority-driven approach

⇒ The priority driven algorithm never leave any resource idle intentially

⇒ Scheduling decisions are made when events of jobs occur (Ex: releases & completion of jobs). Hence priority driven algorithms are event-driven one.
& other commonly used terms of this approache are greedy scheduling, list scheduling & work-conserving scheduling.

⇒ It is greedy because it takes best decisions.

⇒ So that it makes some jobs wait even when they are ready to execute and the resources they require are available.

⇒ priorities are assigned to jobs, and jobs ready for execution are placed in one or more queues.

⇒ At any scheduling decision time, the jobs with the highest priorites are scheduled and executed on the available processors.

⇨ Hence a priority-driven scheduling algorithm is defined largely by the list of priorities it assigns to jobs

⇨ The priority list & other rules such as whether preemption is allowed, defined the scheduling algorithm completely.

⇨ Most non-real-time scheduling algorithms are priority-driven.

Examples:

- FIFO (first-in-first-out) and LIFO (last-in-first-out) algorithms which assign priorities to jobs based on their release times.
- SETE (Shortest-execution-time-first) and LETF (longest-execution-time-first) algorithms which assign priorities based on job execution times.

⇨ Because we can directly change the priorities of jobs, even round-robin scheduling can be thought of as priority driven.

⇨ The priority of the executing job has executed for the time slice.

⇨ The task graph shown here is a classical precedance graph: all its edges represent precedence constraints.

⇨ The number next to the name of each job is its execution time.

⇨ $J_5$ is released at time 4, all other jobs are released at time 0.

⇨ The two processors $P_1$ & $P_2$ are used to schedules the jobs & they are communicated through shared memory

⇨ Hence the costs of communication among jobs are negligible no matter where they are executed.

⇨ The schedulers of the processors keep one common priority queue of ready jobs.

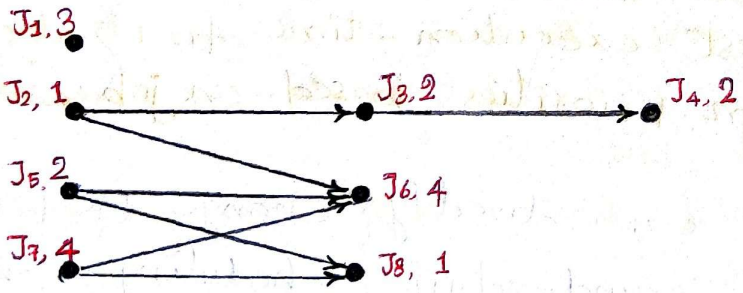
⇨ The priority list is given next to the graph.

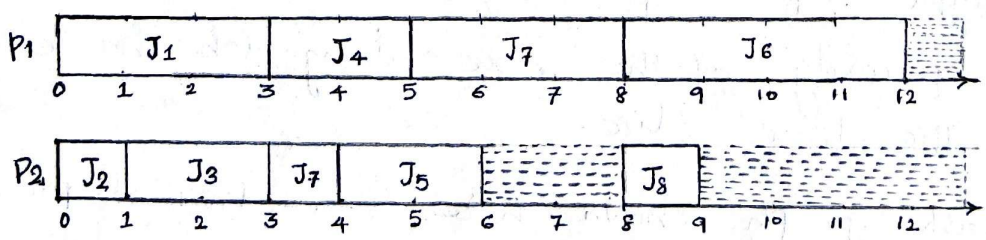
⇨ 'Ji' has a higher priority than Jk if $i < k$. All the jobs are are preemptable.

⇨ scheduling decisions are made whenever some jobs becomes ready for execution or some jobs completes.

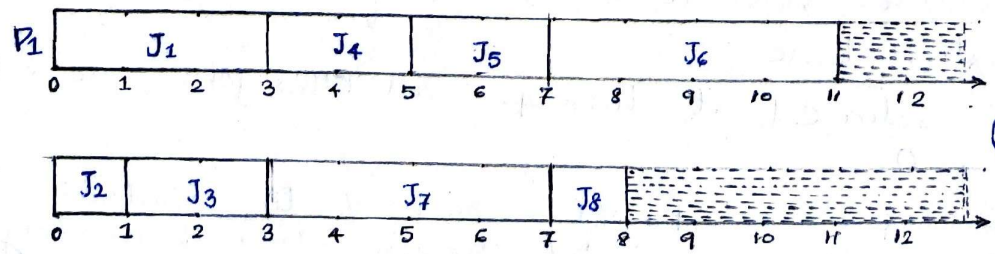
⇨ Figure (a) below shows the schedule of the jobs on the two processors generated by the priority-driven algorithm following this priority assignment.

⇨ At time 0, jobs $J_1$, $J_2$ & $J_7$ are ready for execution. Since $J_1$ & $J_2$ have higher priority than $J_7$, they are ahead of $J_7$ in the queue & hence are scheduled.




(a) Preemptive




(b) Non-preemptive

Example of priority-driven scheduling

⇨ They are the only jobs in the common priority queue at this time.

⇨ Since $J_1$ & $J_2$ have higher priorities than $J_7$, they are ahead of $J_7$ in the queue and hence they are scheduled.

⇨ The processors continue to execute the jobs scheduled on them except when the following decisions are made.

* At time 1, $J_2$ completes and hence $J_5$ becomes ready. $J_3$ is placed in the priority queue ahead of $J_7$ and is scheduled on $P_2$, the processor freed by $J_2$.
* At time 3, both $J_1$ & $J_3$ complete. $J_5$ is still not released. $J_4$ & $J_7$ are scheduled.
* At time 4, $J_5$ is released. Now there are three ready jobs. $J_7$ has the lowest priority among them. Consequently, it is preempted. $J_4$ & $J_5$ have the processors.
* At time 5, $J_4$ completes. $J_7$ resumes on processor $P_1$
* At time 6, $J_5$ completes. Because $J_7$ is not yet completed, both $J_6$ & $J_8$ are not ready for execution. Consequently, processor $P_2$ becomes idle.
* $J_7$ finally completes at time 8. $J_6$ & $J_8$ can be scheduled.

⇨ Figure (b) shows a non-preemptive schedule according to the same priority assignment.

⇨ Before 4, this schedule is the same as the preemptive schedule.

⇨ However, at time 4 when $J_5$ is released, both processors are busy. It has to wait until $J_4$ completes (at time 5) before it can begin execution.

⇨ It turns out that for this system this postponement of the higher priority job benefits the set of jobs as a whole.

⇨ The entire set completes 1 unit of time earlier according to the non-preemptive schedule.

## Estimating Program Run Time:

- A real-time program is defined as a program for which the correctness of operation depends on the logical results of the computation and the time at which the results are produced.
- In general, there are three types of programming **Sequential, Multi-tasking & Real-time.**

⇨ Estimating program run time depends on the following factors:

1. Source Code: Source code that is carefully tuned and optimized takes less time to execute.
2. Compiler: It maps source level code into a machine level program.
3. Machine architecture: Executing program may require much interaction between the processor and the memory and I/O devices.
4. Operating System: OS determines such issues as task scheduling and memory management. Both have major impact on memory management.

# Analysis of Source Code:

⇨ Cosider the following code:

```
a := b x c;
b := d + e;
d := e - f;
```

⇨ This is straight line code. The total execution time is given by

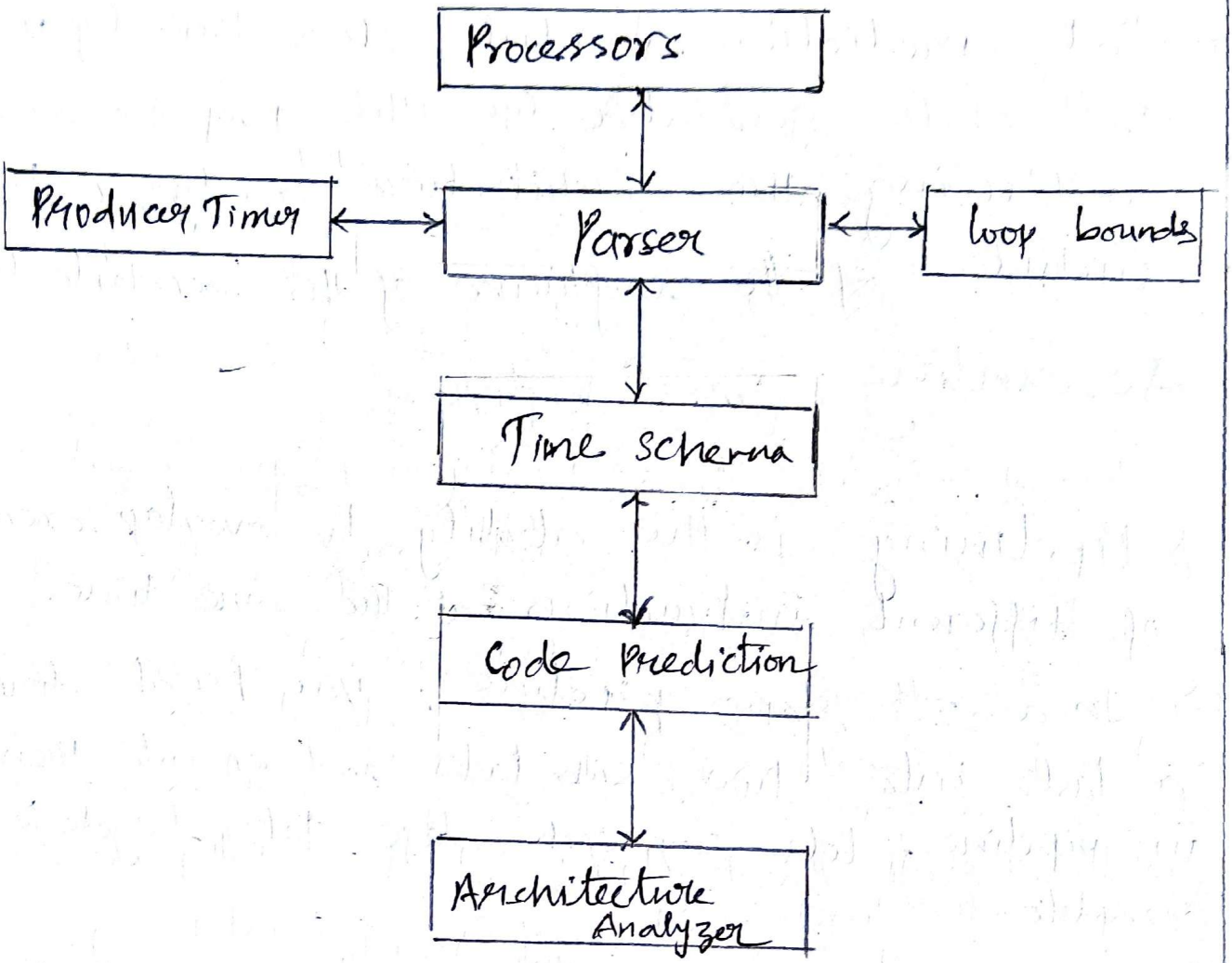
$$\sum_{i=1}^{3} T_{exec}(L_i)$$

where $T_{exec}(L_i)$ is the time needed to execute $L_i$.

⇨ Execution time analysis is any structured method or tool applied to the problem of obtaining information about the execution time of a program or parts of a program.

⇨ The fundamental problem that a timing analysis has to deal with is the following: The execution time of a typical program is not a fixed constant, but rather varies with different probability of occurrence across a range of times.

⇨ The following schematic of timing estimation system.




Schematic of timing estimation system.

- Preprocessor produces compiled assembly language code. Parser analyze input source program.
- Producer timer maintain a table of procedures and their execution times.
- The loop bounds module obtaines bounds on the number of iterations for the various loops in the system.

- time schema is independent of the system, it depends only on the language.
- Code prediction module does this by using the code generated by the preprocessor and using the architecture analyzer to include the influence of the architecture.

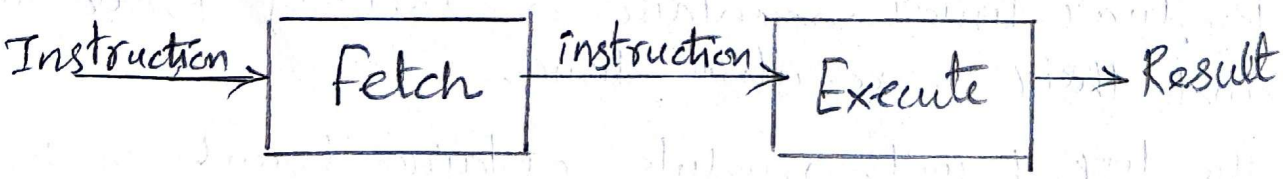
## Accounting for pipeling

⇒ Pipelining is the ability to overlap execution of different instructions at the same time.

⇒ In a 2nd stage pipeline, you break down a task into two sub-tasks and execute them in pipeline. Lets say each stage takes 1 cycle to complete.

⇒ This means in a 2-stage pipeline, each task will take 2 cycles to complete.

⇒ An instruction has a no. of stages. The various stages can be worked on simultaneously through various blocks of production. This is a pipeline. This is also reffered as instruction pipeling




Pipeline of two independent stages

# Task Assignment and scheduling

⇨ scheduling real time tasks on distributed and microprocessor systems consists of two subproblems.

1. Task allocation to the processor.

(a) The task assignment problem is concerned with how to partition a set of tasks and then how to assign these tasks to processors – task assignment can be: 1. static or 2. Dynamic.

(b) In the static allocation scheme, the allocations of tasks to nodes is permanent and does not change with time.

(c) In the dynamic task assignment, tasks are assigned to the nodes as they arise, different instances of tasks may be allocated to different nodes.

2. Scheduling of tasks on the individual processors: uniprocessor scheduling algorithms can be used for the task set allocated to a particular processor.

## static allocation algorithms:

⇨ The tasks are pre-allocated to processors.

⇨ No overhead incurs during run time since tasks are permanently assigned to processors at the system initialization time.

1. Next-Fit Algorithm for RMA
2. Bin Packing Algorithm for EDF
3. Utilization Balancing Algorithm.

⇨ Dynamic allocation Algorithms:

- In many applications tasks arrive sporadically at different nodes.
- The taks are assigned to processor as and when they arise.
- The dynamic approach incurs high run time overhead since the allocator component running at every node needs to keep track of the instantaneous load position at every other nodes.

1. Focussed Addressing and Bidding (FAB)
2. The Buddy strategy Algorithm

## Utilization - Balancing Algorithm

⇨ This algorithm attempts to balance processor utilization, and proceeds by allocating the tasks one by one and selecting the least utilized processor.

⇨ Objective to balance processor utilization, and proceeds by allocating the tasks one by one and selecting the least utilized processor.

⇨ Maintains the tasks in a queue in increasing order of their utilizations.

⇨ It removes task one by one from the head of the queue and allocates them to the least utilized processor each time.

⇨ The objective of selecting the least utilized processor is to balance the utilization of different processors.

$$\frac{\sum_{i=1}^{P} (u_i^B)^2}{\sum_{i=1}^{P} (u_i^*)^2} \leq \frac{9}{8}$$

Where $u_i^*$ = $P_i$'s utilization under an optimal algorithm that minimizes $\sum$ utilization$^2$

$u_i^B$ = $P_i$'s utilization under bet-fit algorithm.

## Next-Fit Algorithm for RM-scheduling :

- This is a utilization-based allocation heuristic.
- The task set has the same properties as for the RM uniprocessor scheduling algorithms.
- 'M' is picked by user.
- Corresponding to each task class is a set of processors that is only allocated to task of that class.

- It is possible to show that this approach uses no more than 'N' times the minimum possible number of processors.

- There are 'm' classes of tasks such that, each class of tasks are assigned to a corresponding set of processors.

  $T_i$ belongs to class $j < m$ if

$$\frac{1}{2^{1+j} - 1} < \frac{e_i}{P_i} \leq \frac{1}{2^{i/j} - 1}$$

  $T_i$ belongs to class m otherwise.

## Bin-Packing Assignment for EDF.

⇨ Same assumptions on tasks and processors as at Next-fit algorithm.

⇨ Problem : Schedule a set of periodic independent preemptible tasks on a multiprocessor system consisting of identical processors.

⇨ The task deadlines equal their periods and tasks require no other processors.

⇨ The task deadlines equal their periods and tasks require no other resources.

- Solution: EDF – scheduling on a processor and task set is EDF – schedulable if $U \leq 1$
- Assign tasks such that $U \leq 1$ foll all processors.
- The problem reduces to making task assignments to processors with the property that the sum of the utilizations of the tasks assigned to a processor does not exceed one.

## Focused Addressing and Bidding (FAB) Algorithm.

⇨ It uses dynamic allocations.

⇨ FAB is a simple algorithm that can be used as an online procedure for task sets consisting of both critical and non-critical real-time tasks.

⇨ Critical tasks must have sufficient time reserved for them so that they continue to execute successfully, even if they need their worst case execution time.

⇨ Non-critical tasks are either processed or not, depending on the system's ability to do so.

⇨ The guarantee can be based on the expected run time of the task rather than the worse-case run time. (non critical task)

⇨ THE UNDERLYING SYSTEM MODE IS:
When an noncritical task arrives at processor Pi, the processor checks to see if it has the resources and time to execute the task without missing any deadlines of the critical tasks or the previously guaranteed noncritical tasks -- if yes, Pi accepts this new noncritical task and adds it to its list of tasks to be executed and reserves time for it.

⇨ The FAB ALGORITHM IS USED WHEN
Pi determines that it does not have the resources or time to execute the task in this case, it tries to ship that task out to some other processor in the system.

⇨ Every processor maintains two tables called: status table and load table.

⇨ STASUS TABLE: indicates which tasks have been already committed to run including the set of critical tasks and any additional noncritical tasks that have been accepted at the different processors can be determined.

⇨ LOAD TABLE contains the latest load information of all other processors of the system, the surplus computing capacity available at the different processors can be determined.

* THE TIME AXIS is divided into windows, which are intervals of fixed duration, at the end of each window, each processor broadcasts to all other processors the fraction of computing power in the next window for which it has no committed tasks.

1. Every processor on receiving a broadcast from a node about the load position updates the system load table

2. Since the system is distributed, this information may never be completely up to date.

3. As the result, when a task arrives at a node, the node first checks whether the task can be processed locally, if yes, it updates its status table if not, it looks for a processor to offload the task.

* THE PROCESS OF OFF LOADING A TASK is based on the content of the system load table, an overloaded processor checks its surplus information and:

1. select a processor (called focused processor) Ps that is believed to be the most likely to be able to successfully execute that task by its deadline.

2. The system load table information might be out of date

• The RFB (Requests For Bids) contains the vital statistics of the task.

3. The RFB asks any processor that can be successfully execute the task to send a bid to the focused Processor.

4. An RFB is only sent out if the sending processor Ps estimates that there will be enough time for timely response to it.

5. Specifically, two times $t_{bid}$ & $t_{offload}$ are calculated → if $t_{bid} \leq t_{offload}$ then the RFB is sent out.

## TIME CALCULATION BY FAB ALGORITHM.

1. $t_{bid}$ = (Estimated time taken by RFB to reach its destination) + (The estimated time taken by the destination to respond with a bid) + (The estimated time taken to transmit the bid to the focused processor);

2. $t_{offload}$ = (Task deadline) − [(Current time) + (time to move the task) + (Task-execution time)].

* If $t_{bid} \leq t_{offload}$; then RFB is sent out.

* When a processor Pt receives an RFB, it checks to see if it can meet the task requirements and still execute its already-scheduled tasks successfully.

# Buddy Strategy:

* The buddy strategy tries to solve the same problem as the FAB algorith, soft real-time arrive at the various processors of a multiprocessor and, if an individual processor finds itself overloaded, it tries to off load some tasks onto less lightly loaded processors.

* The buddy strategy differs from the FAB algorithm in the manner in which the target processors are found.

STRETEGY:

⇒1 Each processor has 3 thresholds of loading: Under loaded (TU), fully loaded (TF), and over loaded (TV)

⇒ 2. The loading is determined by the no. of jobs awaiting service in the processor's queue. If the queue length is Q, the processor is said to be in:

a. State U (underloaded) if $Q \leq TU$;
b. State F (Fully loaded) if $TF < Q \leq TV$;
c. State V (Overloaded) if $Q > TV$;

# Fault Tolerance Techniques

- Fault-tolerance is defined informally as the ability of a system to deliver the expected service even in the presence of faults.
- A common misconception about real-time computing is that fault-tolerance is orthogonal to real-time requirements. It is often assumed that the availability and reliability requirements of a system can be addressed independent of its timing constrains.
- A real-time system may fail to function correctly either because of errors in its hardware and/or software or because of not responding in time to meet the timing requirement that are usually imposed by its "**environment**".
- Hardware fault is some physical defects that can cause a program to fail for a given set of inputs.
- An error is a manifestation of a fault. The fault latency is the duration between the onset of a fault and its manifestation as an error.

- An error latency is the duration between when an error is produced and when it is either recognized as an error or causes the failure of the System.
- Error recovery is the process by which the system attempts to recover from the effects of an error.
- Recovery from an error is fundamental to fault tolerance.
- Two main forms of recovery
  1. Forward error recovery
  2. Backward error recovery.

## Forward error recovery

- Forward recovery attempt to bring system to a new stable state from which it is possible to proceed
- Forward error recovery continues from an erroneous state by making selective corrections to the system state.
- This include making safe the controlled environment which may be hazardous or damaged because of the faiture.

⇨ It is system specific and depends on accurate predictions of the location and cause of errors. (i.e. damage assessment)

⇨ Example : Redundant pointers in data structures and the use of self-correcting codes such as Hamming codes.

**Advantages forward-error recovery :**

1. Less overhead

**Disadvantages of forward recovery**

1. In order to work, all potential errors need to be accounted for up-front.
2. limited use
3. Cannot be used as general mechanism for error recovery.
4. Design specifically for a particular system.

**Backward recovery :**

- most extensively used in distributed systems and generally safest. It can be incorporated into middleware layers.
- Backward recovery is complicated in the case of process, machine or network failure but no guarantee that same fault may occur again.

- It can not be applied to irreversible operations, e.g. ATM withdrawal.

## Advantage backward - error recovery

1. Simple to implement.
2. Can be used as general recovery mechanism.
3. Capable of providing recovery from arbitrary damage.

## Disadvantage of backward recovery.

1. Checkpointing can be very expensive - especially when errors are very rare.
2. Performance penalty.
3. No guarantee that fault does not occur again.
4. Some components cannot be recovered.

## Causes of failure:

- There are three causes of failure,
  1. Errors in the specification or design.
  2. Defects in the components.
  3. Environmental effects.

- Mistake in the specification and design are very difficult to guard against. Many hardward failures and all software failures occurs such mistake.
- If the specification is wrong, everything that processeds pro it, design and implementation, likely to be unsatisfactory.

## Fault Types

- Fault are classified as temporal behaviours and output behaviours.

### 1. Temporal behaviours classification.

- Fault are of three types: permanent, intermittent & transient.

- **Transient faults:** These occur once and then disappear. For example, a network message transmission times out but works fine when attempted a second time.
- **Intermittent faults:** Those are the most annoying of component faults. This fault is characterized by a fault occuring then vanishing again then occuring. As example of this kind of fault is a loose connection.

Permanent faults:

* This fault is persistent: It continues to exist until the faculty components is repaired or replaced. Examples of this fault are disk head crashes, software bugs, and burnt-out hardware.

## 2. output behaviours classification.

- Malicious faults: Inconsistent output.
- Nonmalicious fault: Consistent output errors.
- Fail stop: Responds to up to a certain maximum numbers of failures by simply stoping, rather than putting out incorrect outputs. The component simply stoping, rather than putting out incorrect outputs. The component simply stops working. For instance, a hard disk which refuse to read or write.

**Fail safe:** Its failure mode is biased so that the application process does not suffer catastrophe upon failure. A component under too much load is likely to fail. A fail safe system, on detecting a large amount of load processes such request slower to avoid failure.

## 3. Independence and Correlation:

- Components failure may be independent and correlated.
- Independent : A failure is said to be independent it does not directly or indirectly cause another failure.
- Correlated : If the failure is said to be correlated if they are related in some way.

## Reliability Evaluation:

⇨ Reliability refers to the property that a system can run continuously without failure. In contrast availability is defined interms of a time interval instead of an instant in time.

⇨ A highly reliable system is one that will most likely continue to work without interruption during a relatively long period of time. This is a subtle but important difference when compared to availability.

⇨ If a system goes down on average for one, seemingly random millisecond every hour, it has an availability of more than 99.9999 percent, but it still unreliable.

- Two methods are used for finding device failure rates: Collecting field data or life cycle testing in the laboratory.
- The most common accelerate is temperature. The higher the temperature the greater the failure rate. The acceleration factor is given by the following equation.

$$R(T) = Ae^{-Fa/kT}$$

where A is a constant, Fa is the activation energy and depends largely on the logic family used, k is the Boltzmann constant.

- To measure how quickly an error can propagate, we use fault injection. This is best done on a prototype. special-purpose hardware is used ~~fault~~ to simulate a fault on a selected line. The status of related lines is monitored using logic analyzers to determine how far the error propagates and how quickly. If a prototype is not available, a software simulation can be substituted.

# Clock Synchronization:

- Clock $C_i$ is a mapping.

  $C_i$ : Real time → clock time

- At real time t, $C_i(t)$ is the time told by clock $C_i$. The inverse function $C_i(t)$ is the real time at which clock $C_i$ tells time T.
- Clock drife rate is the rate at which the clock can gain or lose time.
- Why do we want the drift rate to be as small as possible?
- There are two reasons:

1. How the clock rate of computer is determined The speed of the computer is a function of the clock rate. The clock period is choosen to be just long enough for signal propagation along the critical path of a computer circuit.

2. Some synchronization algorithm adjust the clock simultaneously.

If a synchronization algorithm is employed in the system to compensate the time error, two main sources of error remains; the information about the time of a clock degrade due to