unit-3

Reference model of real time system

The model that focuses on the relevant characteristics such as timing properties, resource requirements of the system components and the way in which the operating system allocates the available system resources among them is called reference model of RTS.

→ Reference model is characterized by:

1. A workload model that describes applications running on the systems / supported by the system. Represented by a task graph.
2. A resource model that describes the resources available to ~~these~~ the applications. Represented by a resource graph.
3. A scheduling algorithm that define how the applications execute and use the resources. Represented by types of scheduling algorithm i.e. online vs offline, static vs dynamic

* Jobs characteristics:

Each job is characterized by

1. Temporal parameters: tell us its timing constraints and behavior.
2. Functional parameters: specify the intrinsic properties of the job.
3. Resource parameters: give us its resource requirements.
4. Interconnection parameters: describe how it depends on other jobs and how other jobs depend on it.

Temporal parameters of real time workload

The temporal parameters of a job is the parameters which describes its timing constraints and behavior. The temporal parameter of Job J_i are

- Release time (r_i)
- Relative deadline (D_i)
- Absolute deadline (d_i)
- Feasible interval $[r_i, d_i]$

↳ time interval b/t release time & absolute deadline means the interval that begins immediately after r_i time & ends at d_i

r_i^- → earliest release time r_i^+ → latest RT

Workload specifies the no. of jobs assign to a processor. The real time work load parameter are:

1. Number of tasks or jobs in the system
2. Run-time system

Fixed, Jitter and sporadic Release Time

→ In many system, we do not know the actual release time r_i of each job J_i. But, we know that r_i is in a range $[r_i^-, r_i^+]$. r_i can be as early as r_i^- and as late as r_i^+. When only the range of r_i is known, then this range is called jitter in r_i or the release time jitter.

→ When the release time jitter is negligible (very small) compared to other temporal parameters, the the actual release time of each job is given by its earliest r_i^- or latest r_i^+ release time then we can call it a fixed release time.

→ Sporadic jobs/aperiodic jobs are released at random time instants. So the release time of sporadic/aperiodic jobs can be represented by random variables, and can be modeled as a random variable with some probability distribution, $A(x)$.
- $A(x)$ gives the probability that the release time of the job is not later than x, for all valid values of x.

Periodic Task model

It is a well-known deterministic work load model used to describe hard real time application such as digital control and real time monitoring system.

In periodic task model following types of task are incorporated.

1. Periodic Task

2. Aperiodic
3. sporadic

Periodic Task

A set of jobs that are executed repeatedly at regular time intervals is called periodic task. At all times, the period and execution time of every periodic task in the system are known.

Assume that the tasks in the system are $T_1, T_2, \ldots, T_n$. Each periodic task T_i is a sequence of jobs $J_{i,j}$ (i.e. $J_{i,1}, J_{i,2}, \ldots, J_{i,n}$) and can be represented by a 4 tuple (ϕ_i, P_i, e_i, D_i), where;

- ϕ_i called the phase of Task T_i. It is equal to $r_{i,1}$ i.e. it is the release time of the first job $J_{i,1}$ in the task T_i i.e. $\phi_i = r_{i,1}$.
- P_i is called the period of Task T_i. It is the minimum length of all time intervals between release times of consecutive jobs in T_i.
- e_i is called the execution time (or worst case execution time) of T_i. It is the maximum execution time of all jobs in the periodic task T_i.
- D_i is called the relative deadline of the task T_i.

Hyperperiod (H) / Major cycle of Tasks :

The hyper-period of a set of periodic tasks is the least common multiple of their periods i.e. $H = LCM(P_i)$ for $i = 1, 2, \ldots, n$

→ The (maximum) number of jobs N in each hyperperiod is equal to $\sum_{i=1}^{n} \frac{H}{P_i}$.

E.g. The length of a hyperperiod of three periodic task with periods 3, 4, and 10 is 60.

The total no. of jobs in the hyperperiod (N) $= \left(\frac{60}{5} + \frac{60}{4} + \frac{60}{10}\right) = 41$

Utilization (U_i):

The ratio of execution time and period is called utilization of a task T_i.

$$U_i = \frac{e_i}{P_i}$$

- U_i is equal to the fraction of time a truly periodic task with period P_i and execution time e_i keeps a processor busy.

Total utilization (U):

The total utilization of a system is the sum of the utilizations of all tasks in a system.

i.e. $U = \sum_{i=1}^{n} U_i$

e.g.

→ If the execution times of the three periodic tasks are 1, 1 and 3, and their periods are 3, 4, and 10, respectively, then their utilization are 0.33, 0.25 and 0.3

→ The total utilization of the task is 0.88; these task can keep a processor busy at most 88 percent of the time.

- T_1 : $P_1 = 3$, $e_1 = 1$
- T_2 : $P_2 = 5$, $e_2 = 2$

A system of periodic task repeats after the hyper-period, $H = LCM(P_i)$ for $i = 1, 2, \ldots n$.

The number of jobs in the hyperperiod $= \sum_{i=1}^{n} \frac{H}{P_i}$.

$= \left(\frac{H}{P_1} + \frac{H}{P_2}\right) = \frac{15}{3} + \frac{15}{5}$

$= 8$

Total utilization $= \frac{e_1}{P_1} + \frac{e_2}{P_2} = \frac{1}{3} + \frac{2}{5} = 0.733.$

Aperiodic & sporadic Tasks

The jobs executed in response to an external events is called aperiodic or sporadic jobs. They are released at random times

→ Each aperiodic or sporadic task is a stream of aperiodic or sporadic jobs respectively

→ If the tasks containing jobs that are released at random time instants and have hard deadlines then they are called sporadic task. sporadic tasks are treated as hard real-time tasks. For e.g.,

- An autopilot is required to respond to a pilot's command to disengage the autopilot and switch to manual control within a specified time.
- A fault tolerant system may be required to detect a fault and recover from it in time to prevent disaster.

→ A task is aperiodic if the jobs in it are released at random time instants, and have either soft deadlines or no deadlines. For e.g.,

Date
Page

• The task to adjust radar's sensitivity is an example. We want the system to be responsive, i.e, to complete each adjustment as soon as possible. on the other hand, a late response is annoying but tolerable.

Precedence Constraints and Data Dependency

→ The jobs in a task, whether periodic, aperiodic or sporadic, may be constrained to execute in a particular order. This is known as a precedence constraint.

→ We use a partial-order relation $<$, called a precedence relation, over the set of jobs to specify the precedence constraints among jobs.

- A job J_i is a predecessor of another job J_k (and J_k is a successor of J_i) if J_k cannot begin until the execution of J_i completes. It is denoted as $J_i < J_k$.
- J_i is an immediate predecessor of J_k if $J_i < J_k$ and there is no other job J_j such that $J_i < J_j < J_k$.
- J_i and J_k are independent when neither $J_i < J_k$ nor $J_k < J_i$. It means they can execute in any order.

→ A job with a precedence constraint becomes ready for execution once when its release time has passed and when all predecessors have completed.

Examples:
- Radar surveillance system
- Information system:
  ↳ Suppose that before each query is processed and the requested information retrieved, its authorization to access

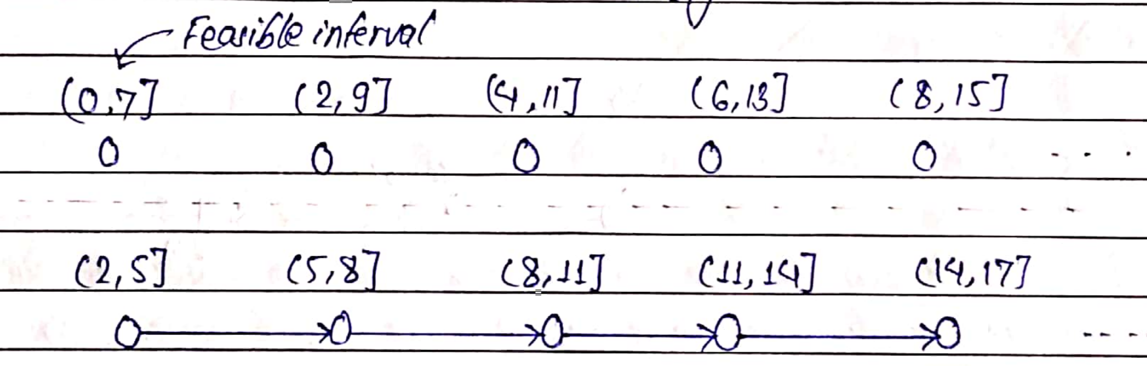
the requested information is first checked. The retrieval job cannot begin execution until the authentication job completes. The communication job that forwards the information to the requester cannot begin until the retrieval job completes.

Precedence Graph

A precedence graph is a directed acyclic graph $G = (J, <)$ which represents the precedence constraints among jobs in a set J. Each vertex in this graph represents a job in J. There is a directed edge from the vertex J_i to the vertex J_k when the job J_i is an immediate predecessor of the job J_k.

e.g.

Jobs are shown as circle in this figure.



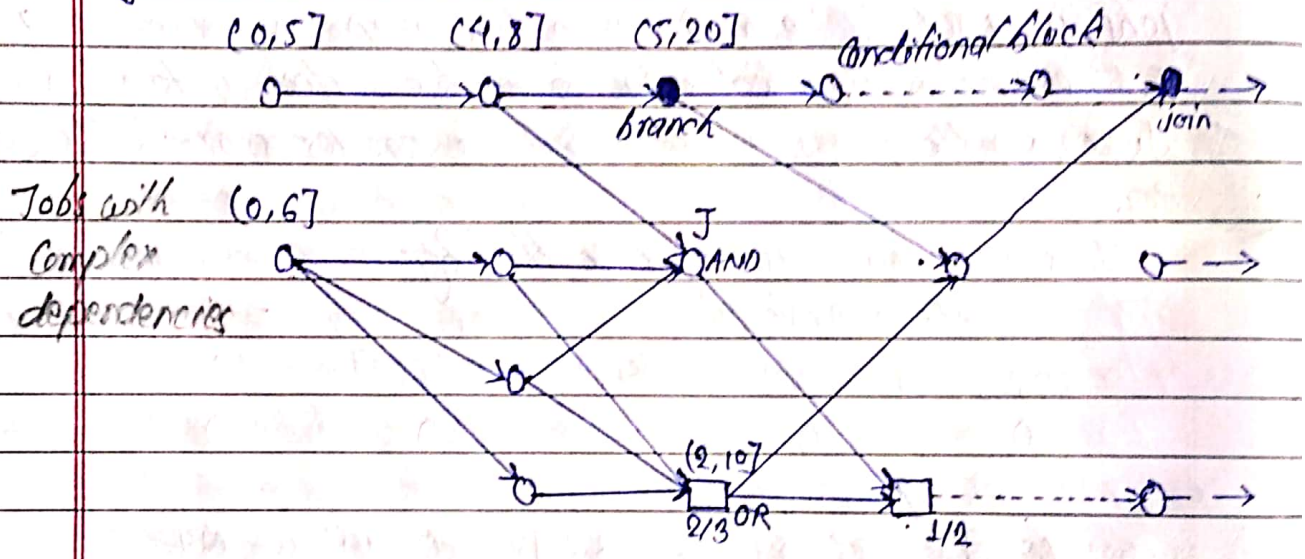
- The task whose jobs are represented by the vertices in the top row has phase 0, period 2 and relative deadline 7. The jobs in it are independent as there is no edge to or from these jobs.
- The task in the second row of figure represents jobs in a periodic task with phase 2, period 3, and relative deadline 3. The jobs in it are dependent. The first job is the immediate predecessor of the second job, the second job is the immediate predecessor of the third job, and so on.

A task graph, which gives us a general way to describe an application system, is an extended precedence

Task Graph

A task graph is an extended precedence graph. The vertices in the task graph represents jobs, numbers in brackets above each job represents feasible interval and the edges represents dependencies among the jobs. But if all the edges are precedence edges representing precedence constraints then the task graph also becomes a precedence graph.

E.g.



Why Task Graph?

→ A task graph like the example is only really necessary when the system contains components which have complex dependencies.

→ Many types of interactions and communication among jobs are not captured by a precedence graph but can be captured by a task graph.

→ Unlike a precedence graph, a task graph can contain different types of edges representing different types of dependencies.

Data Dependency

→ Data dependency can't be captured by the precedence graph. In the real time system, the jobs communicate through shared data hence data of one job is dependent with other and called as data dependency. Often, the designer chooses not to synchronize producer and consumer jobs. Instead, each producer places the data generated by it in a shared address space to be used by the consumer at any time. In this case precedence graph will show that the producer and consumer are independent because they are not explicitly constrained to execute in turn. E.g.

In a task graph, data dependencies among jobs are represented explicitly by data dependency edges among jobs. There is a data-dependency edge from a vertex J_i to vertex J_k in the task graph if the job J_k consumes data generated by J_i or the job J_i sends message to J_k. A parameter of an edge from J_i to J_k is the volume of data from J_i to J_k.

↳ E.g.

In an avionics system, the navigation job updates the location of the airplane periodically. These data are placed in a shared space. Whenever the flight management job needs navigation data, it reads the most current data produced by the navigation job. There is no precedence constraint between the navigation job and the flight management job.

Other Types of Dependencies

→ Temporal dependency
→ AND/OR precedence constraints
→ Conditional branches
→ Pipeline Relationship

→ The value of temporal distance parameter of the job is infinite if there is no dependency edge between the jobs.

Temporal Dependency

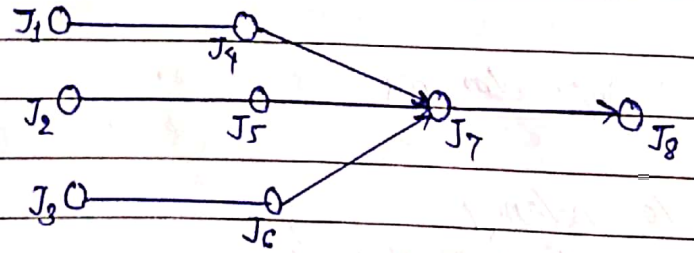
Some jobs may be constrained to complete within a certain amount of time relative to one another. The difference in the completion time of two jobs is called the temporal distance between them. Jobs are said to be temporal distance constraint if their temporal distance must be not more than some finite values. Jobs with temporal distance constraints may or may not have deadlines.

E.g. Lip synchronization

In a task graph, temporal distance constraints among jobs are represented by temporal dependency edges. There is a temporal-dependency edges from a vertex J_i to a vertex J_k if the job J_k must be completed within a certain time after J_i completes.

AND/OR Precedence Constraints

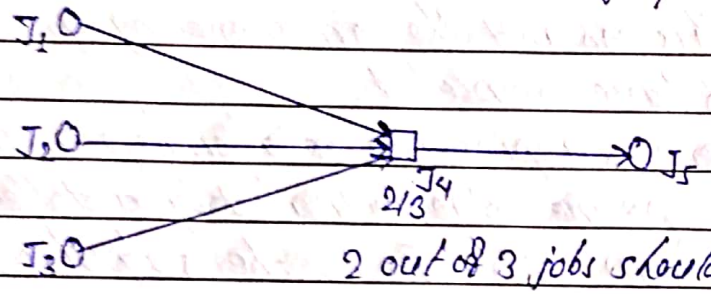
→ If a job having more than one predecessor needs that all the immediate predecessor must have been completed before its execution can begin, then such jobs are called AND jobs. These dependencies are called AND precedence constraints. AND jobs are represented by unfilled circles in the task graph.



⇒ These predecessor J_4, J_5, J_6 may execute in any order relative to each other, but they must all be completed

before the job J_7 begin.

→ If a job having more than one immediate predecessor can begin execution at or after its release time if only one or some of its immediate predecessor have completed execution, then the job is called an or job. This type of dependency constraint is called or precedence constraint. OR jobs are represented by unfilled square vertices in the task graph.



2 out of 3 jobs should complete before J_4 begins

Conditional Branches

If only one of all the immediate successors of a job (whose outgoing edges express OR constraints) is to be executed. such a job is called a branch job. In a task graph, there is a join job associated with each branch job. In a task graph, branch job and associated join job are represented by filled circles.

The subgraph that begins from a vertex representing a branch job and ends at the vertex representing the associated join job is called a conditional block. Only one conditional branch in each conditional block is to be executed.

Fig. previous from task graph.

→ The conditional block in fig. has two conditional branches: Either the upper conditional branch containing a chain of jobs, or the lower conditional branch containing only one job, is to be executed.

Pipeline Relationship

A dependency between a pair of producer-consumer jobs that are pipelined can be represented by a precedence graph. Consumer (job) can begin execution when producer (job) have completed. In it's precedence graph, the vertices are the granules of the producer and the consumer. Each granule of the consumer can begin execution when the previous granule of this job and the corresponding granule of the producer job have completed.
In the task graph, a pipeline relationship among job is shown by a pipeline edge, indicated by a dotted edge. There is an edge from J_i to J_k if the output of J_i is piped into J_k.

Functional Parameters of Real Time Jobs

The functional parameter affect the scheduling and resource access control algorithms. A workload model should describe these parameters, they are:
1. Preemptivity of jobs
2. Criticality of jobs
3. Optional execution
4. Laxity type and Laxity function.

Preemptivity of jobs

Preemptivity of jobs means whether a currently executing job can be momentarily stopped and a new job having higher priority can be executed or not.

→ A job is preemptable if its execution can be suspended at any time to allow the execution of other jobs, and later on can be resumed from the point of suspension. (This interruption of job execution is called preemption.) E.g: Computations by CPU.
(- When higher priority job completes execution, the preemptive jobs resumes execution. During this process, currently executing job is stored in the memory and the job with higher priority is brought into the processor. This process is called context switch & time required to perform context switch is called a context switch time.)

→ A job is non-preemptable if it must be executed from start to completion without interruption. E.g. Data transmission

Criticality of Jobs

The importance or criticality of a job is a positive number that indicates how critical (important) the job is with respect to other job. The more critical the job, the larger its importance. Priority and weights are used to indicate the importance. The more important a job, the higher its priority or the larger its weight. During overload, less critical jobs are sacrificed so that critical jobs meet their deadline. For this, scheduling and resource access algorithm uses, weighted average response time or weighted average tardiness.
E.g. In a flight control management system the job that controls the flight of the ~~air craft~~ aircraft is more critical than the navigation job (current position).

Optional Execution

It defines the portion of jobs that are either optional or mandatory. Optional job or optional part of job can be discarded even when executing. But the not optional or mandatory jobs must be executed to completion. If an optional job completes late or is not executed at all, the system performance may degrade but still functions satisfactorily.

Laxity Type and Laxity Function

Laxity type of a job means whether its timing constraints are hard or soft. Laxity type is supplemented by a usefulness function / laxity function. Laxity function gives the usefulness of the result produced by the job as a function of its tardiness. The tardiness is the difference between the completion time and deadline of the job.

→ Hard RT Jobs : usefulness becomes zero or negative as soon as job is tardy. "Better never than late".

→ Soft RT Jobs : usefulness decreases gradually.

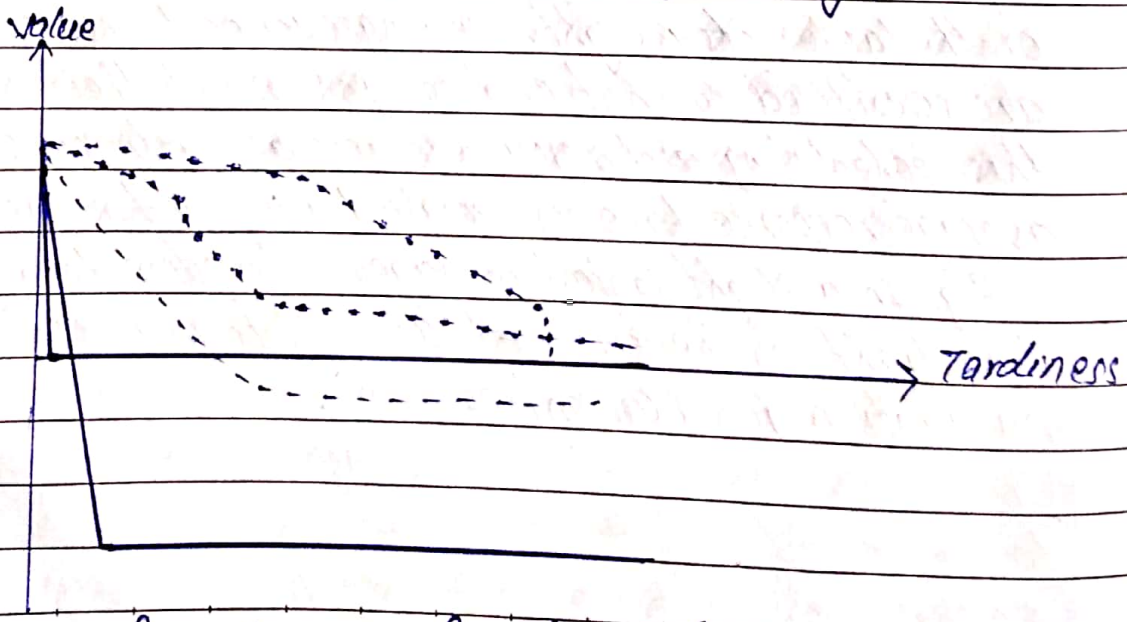


Fig : Examples of usefulness functions

→ In the graph, the dark lines are called hard real time jobs. In this case, the result becomes zero or negative as soon as the job is tardy. Therefore, in the second case, it is better not to execute the job for e.g. release of bomb by a fighter plane.
→ In the graph, the dotted and dashed lines ~~shows~~ are called soft real time system. The dotted lines shows that the usefulness decreases gradually and becomes zero. For e.g. withdrawing money from ATM machines.
The dashed line is a function that decreases faster and becomes negative. Such case can occur while obtaining the stock price.

Resource Parameters of Jobs and Parameters of Resources

A job requires a processor and some resources throughout its execution. The resource parameters of each job give us the type of processor, the unit of each resource type ~~required~~ and the time intervals during its execution required by the job. The information provided by resource parameter is used to support the resource management decision.
The two resource parameters are:
1. Preemptivity of Resources
2. Resource Graph

Preemptivity of Resources

→ A resource parameter is preemptivity.
→ A resource is non-preemptable if each unit of the resource is constrained to be used serially. Once a unit of resource is allocated to a job, other jobs needing it must wait until the job completes its use. E.g. lock on a data object in a database
→ If a job can use every unit of a resource in an interleaved

fashion, the resource is preemptable. E.g. memory.

Resource Graph

A graph that is used to describe the configuration of the resource is called resource graph. In a resource graph, there is a vertex R_i for every processor or resource R_i in the system.

The attributes of the vertex are the parameters of the resource.

- The resource type of a resource tells us whether the resource is a processor or a (passive) resource.
- Number gives us the number of available units of that resource.

Edges in a resource graph represent the relationship among resources. There are two types of edges in resource graph:

1. is-a-part-of edge:

An edge from vertex R_i to R_k means R_k is a component of R_i. This edge is called is-a-part-of-edge. E.g. A memory is a part of a computer, and so is a monitor.

2. Accessibility edge:

Some edges in resource graph represent connectivity between components. These edges are called accessibility edges. E.g.: If there is a connection between two CPUs in the two computers, then each CPU is accessible from the other computer, and there is an accessibility edge from each computer to the CPU on the other computer.

Scheduling Hierarchy

Jobs are scheduled and allocated resources according

to a chosen set of scheduling algorithms and resource access-control protocols.

→ The scheduler is a module that implements scheduling algorithms. It specifically assigns jobs to processors.

→ A schedule is an assignment of all jobs in the system on the available processors.

→ If a scheduler produces only valid schedules then it is called correctness of a schedule. A valid schedule is the one that satisfies the following conditions:
- Every processor is assigned to at most one job at any time.
- Every job is assigned at most one processor at any time.
- No job is scheduled before its release time.
- The total amount of processor time assigned to every job is equal to its maximum or actual execution time.
- All the precedence and resource usage constraints are satisfied.

Feasibility, Optimality & Performance measures of scheduling

→ A valid schedule is a feasible schedule if every job completes by its deadline or meets its timing constraints.

→ A hard real time scheduling algorithm is optimal if the algorithm always produces a feasible schedule if the given set of jobs has feasible schedules.

→ The following are the performance measure of real time system
- Miss rate : Percentage of jobs executed but completed late.
- Loss rate : percentage of jobs discarded.
- Invalid rate : sum of miss and loss rate.
- Makespan : If all jobs have same release time and deadline then the completion time of last job is the makespan.

- Max or average response times
- Max or average tardiness / lateness
- Lateness (L) : L = completion time - deadline (L > 0 if deadline is not met)
- Tardiness (E) : Zero if completion time <= deadline, otherwise equals (completion time - deadline).
  E = max {L, 0} (E = 0 if deadline is met).

Two levels of scheduling

- In the higher level, the application system is scheduled on the resources.
- In the lower level, the jobs that execute in order to implement the resources are scheduled on the processors and resources needed by them.

Jayanta Poudel