2070-10-5

Chapter 4

Commonly Used approaches to Real time scheduling

Three commonly used approaches to real time scheduling are:-

→ Clock-driven approach
→ Weighted round-robin approach and
→ Priority-driven approach.

Clock-driven Approach

→ As the name implies, in clock-driven approach, the jobs are scheduled in specified clock time so it is also known as time driven approach.

→ The specific time instants to execute jobs, are chosen a priori before the system begins execution. i.e.

→ In clock-driven system, all the parameters of hard real-time jobs are fixed and known.

→ A schedule of the jobs is computed off-line and is stored for use at run time.

→ One of the benefits of using clock-driven approach is to minimize scheduling overhead during run time.

→ A frequently adopted choice is to make scheduling decisions at regularly spaced time instants. Example :

Time	Job
At 1 Sec	J_1
At 2 Sec	J_2
At 3 Sec	J_3

Weighted Round Robin Algo:-

The round-robin approach is commonly used for scheduling time-shared applications. When jobs are scheduled on a round-robin basis, every job joins a first-in first-out (FIFO) queue when it becomes ready for execution.

	Burst Time
J_1	24
J_2	3
J_3	3

Quantum time = 4.

J_1	J_2	J_3	J_1	J_1	J_1	J_1	J_1

0 4 7 10 14 18 22 26 30

→ A time slice is the basic granule of time that is allocated to jobs. (order of 10's of milliseconds).

→ If the job does not complete by the end of the time slice, it is preempted and placed at the end of the queue waiting for next turn.

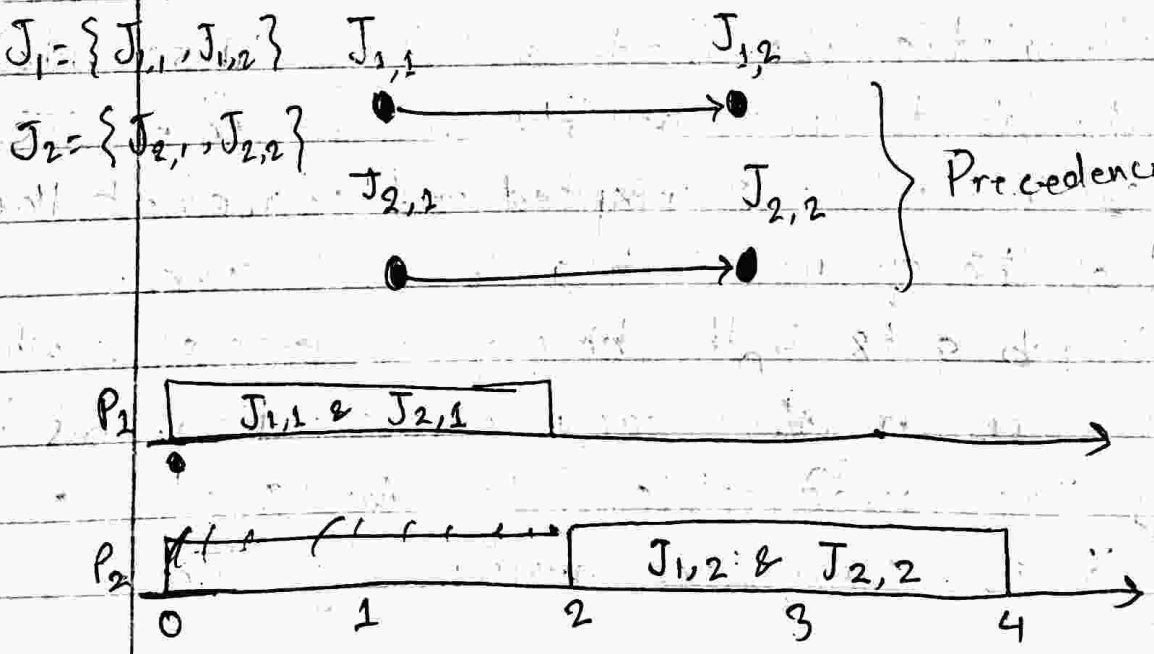
→ Each job gets $1/n$th share of the processor when there are n jobs ready for execution. This is why, the round-robin algorithm is also known as processor-sharing algorithm.

→ The weighted round-robin algorithm has been used for scheduling real-time traffic in high-speed switched networks. It builds on basic round-robin scheme. Rather than giving equal time slice to all ready jobs, different jobs may be given different weights.

→ The weight of a job (wt) refers to fraction of processor time allocated to the job.

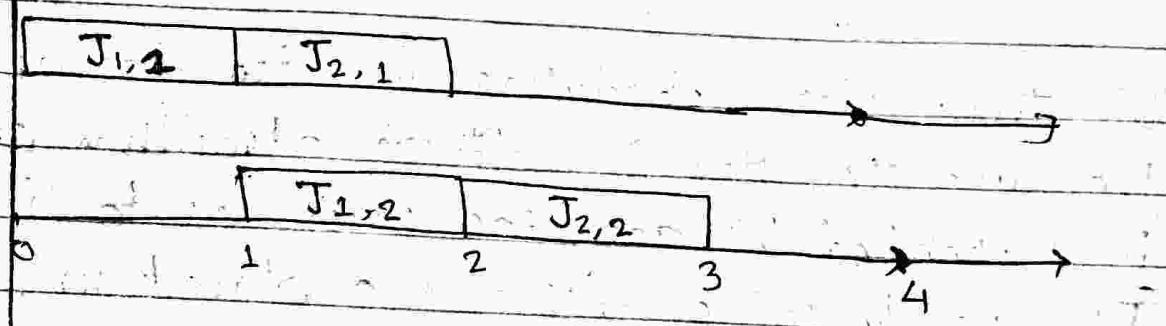
→ A job with the weight wt gets wt time slices every round, and the length of a round is equal to the sum of the weights of all the ready jobs.

→ By adjusting the weight of the ready jobs, we can speed up or retard the progress of each job towards its completion.



(a) shows both sets of jobs completes approximately at time 4, if the jobs are scheduled in

a weighted round-robin manner when the jobs have ~~same~~ same the time slice i.e. 1.



(b) Shows that if the jobs on each processor are executed one after the other, one of the chain can complete at time 2, while the other can complete at time 3.

Priority-Driven Approach

The term priority-driven algorithms refers to a large class of scheduling algorithms that never leave any resource idle intentionally. The priority-driven algorithm is also known as event-driven since scheduling decisions are made when events such as releases and completions of jobs occurs.

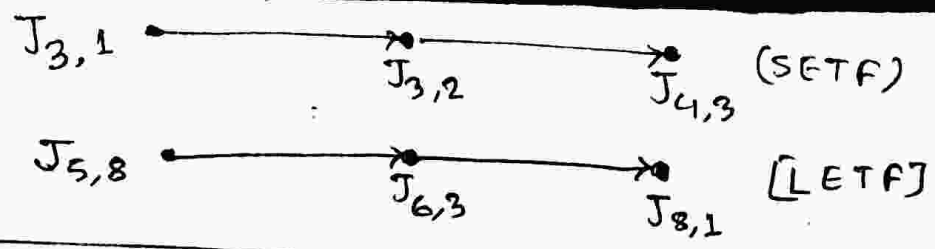
Other commonly used names for this approach are greedy scheduling, list scheduling and work-conserving scheduling.

→ A priority-driven algorithm is greedy since it tries to make locally optimal decisions.

The term list scheduling is also descriptive because any priority-driven algorithm can be implemented by assigning priorities to jobs. Jobs ready for execution are placed in one or more queues ordered by the priorities of the jobs. At any scheduling decision time, the jobs with the highest priorities are scheduled and executed on the available processors. Hence, the priority-driven scheduling algorithm is ~~driven~~ defined to a great extent by the list of priorities it assigns to jobs. So it is termed as list scheduling.

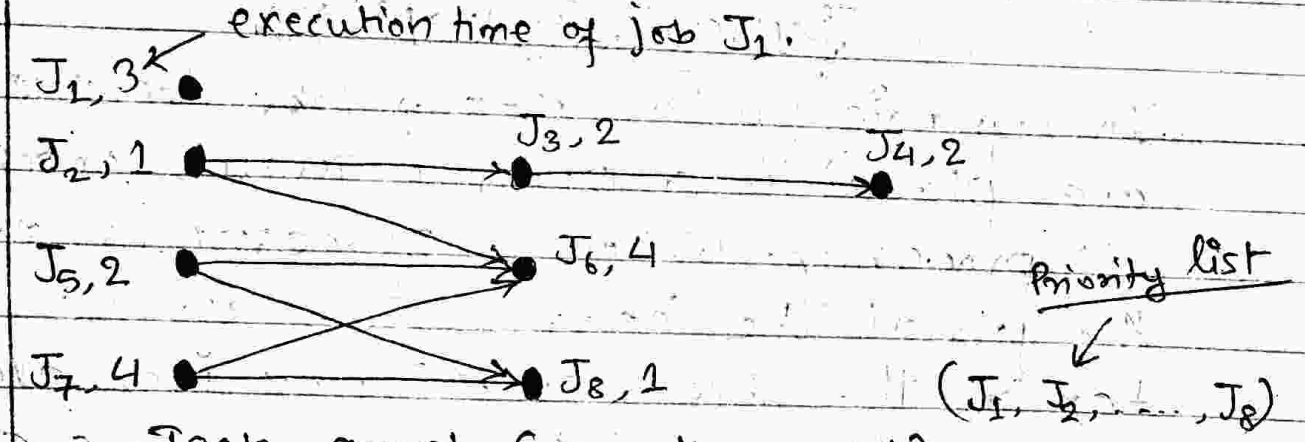
→ Most of the non-real time scheduling algorithms are priority-driven.
for example :- FIFO, LIFO, LETE and SETE algorithms.
FIFO (first-in ~~and~~ first-out) and Last-in-first-out (LIFO) algorithms, which assign priorities of jobs according to their release times.

The SETE (shortest-Execution-Time-first)



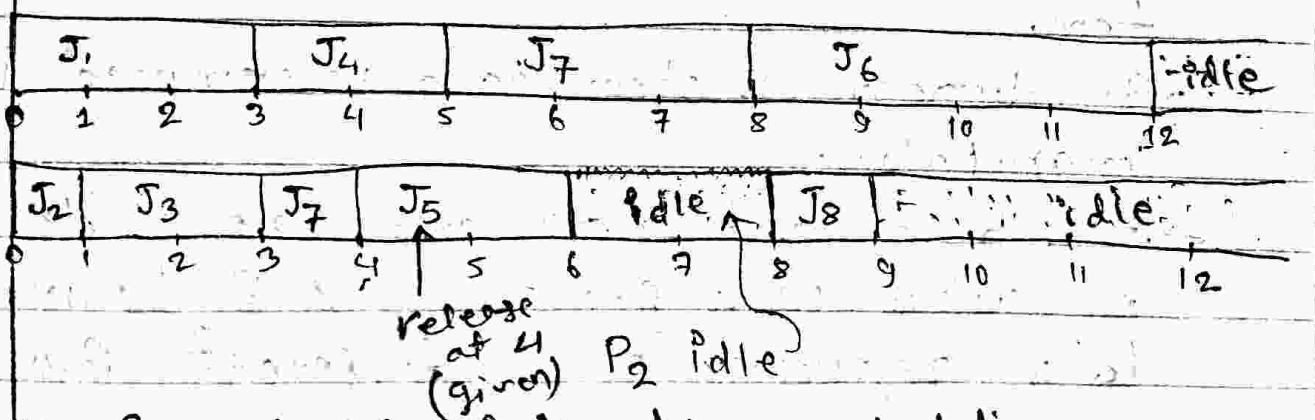
and LETE (Longest-Execution-Time-first) algorithms, which assign priorities on the basis of job execution times.

→ Coffman and Garey [CoGa] recently proved that when there are two processors, the minimum makespan achievable by nonpreemptive algorithm is never more than 4/3 times the minimum makespan achievable by preemptive algorithms when the cost of ~~of~~ premption is negligible.

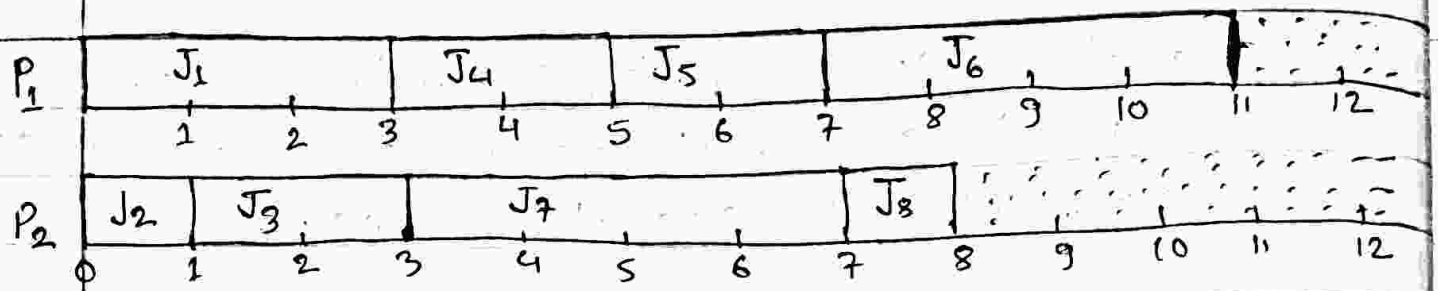


Task graph (precedence graph)

Note:- J_i has higher priority than J_k if $i < k$.



(a) Example of priority-driven scheduling (Preemptive)



(b) Non-preemptive

Note:- The entire set completes 1 unit of time earlier in non-preemptive schedule.

fig (4-2)

Dynamic Versus Static Systems

Dynamic (Job) System

→ In dynamic System, ready jobs for execution are placed in a priority queue common to all processors. When a processor is available, the job at the head of the queue executes on the processor. Since jobs are dynamically dispatched to processors, we call such a multiprocessor system as a dynamic system.

→ In dynamic system, each preempted job is migratable.

→ A job is said to be migratable if it starts execution on a processor, is preempted, and later resumes on a different processor.

Static System

In multiprocessor and distributed systems, the scheduling is to partition the jobs into the subsystems and assign and bind the subsystems statistically to the processors.
The jobs are moved among processors only when the system must be reconfigured, that is, when the operation mode of the system changes or some processor fails.
Such a system is called static system.
[because the system is statically configured.]

→ If the jobs on different processors are dependent, the schedulers on the processors must synchronize the jobs according to some synchronization and resource access-control protocol.

Example

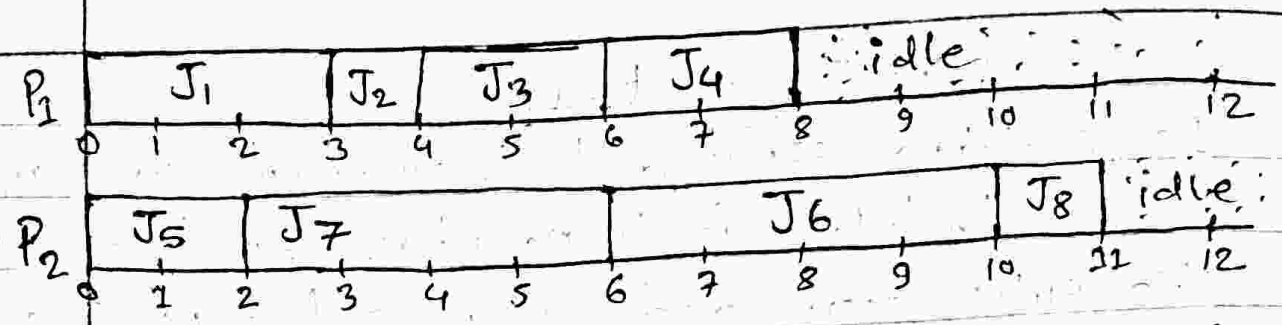
Dynamic Vs. Static Systems

In above example, the priority list ($J_1, J_2, \ldots, J_8$) is segmented into two parts:-

(J_1, J_2, J_3, J_4)

and (J_5, J_6, J_7, J_8)

→ The scheduler of processor P_1 uses the former while the scheduler of processor P_2 uses the latter.



→ The jobs on P_1 complete by time 8 and on P_2 complete by time 11.

→ Therefore, the response of the static system is just as good as that of the dynamic system.

Why most of the hard real time systems built today are static?

Ans:- While dynamic systems may be more responsive on the average, their worst-case real-time performance may be poorer than static systems. More importantly, we do not yet have reliable techniques to validate the timing constraints of dynamic systems while such techniques exist for static systems. For this reason, most hard real-time systems built today are static.

Imp

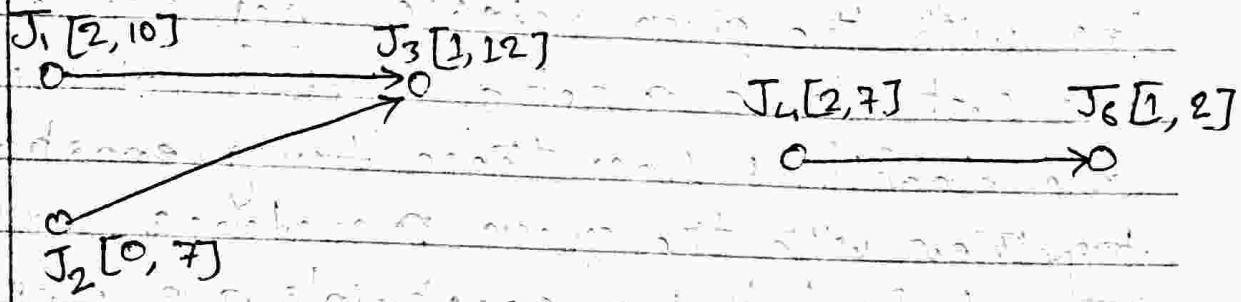
* EFFECTIVE Release Time & Deadline

→ The effective release time of a job without predecessor is equal to its given release time.

The effective release time of all the jobs can be computed in one pass through the precedence graph in $O(n^2)$ time ~~where~~ and effective deadlines can be computed in $O(n^2)$ where n is number of jobs.

The effective release time of a job with predecessor is equal to maximum value among its given release time.

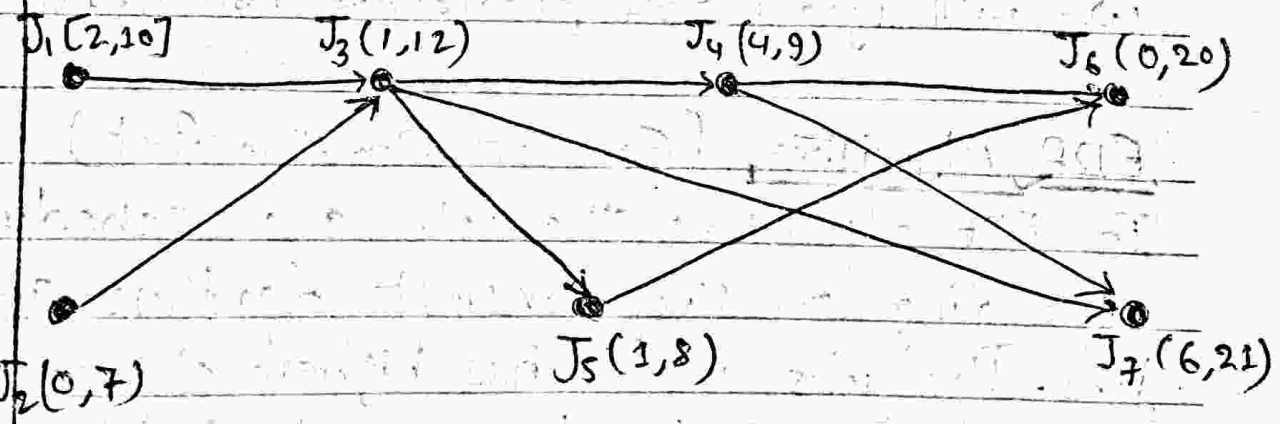
→ The effective deadline of a job without a successor is equal to its deadline & with successor is equal to minimum value.



from the diagram, calculate effective release time & deadline.

J_1	J_2	J_3	J_4	J_6
R.T = 2	R.T = 0	RT = 2	RT = 2	RT = 2
D = 10	D = 7	D = 12	D = 7 2	D = 2

Ans



Find effective release time & deadline.

Why we have to compute Effective release time & deadline

Note:- The given release time and deadlines are sometimes inconsistent with the precedence constraints of the jobs. It means that the release time of a job may be later than that of its successors, and its deadline may be earlier than that of its predecessors. Therefore, rather than working with the given release times and deadlines, we first derive a set of effective release times and deadlines from these timing constraints, together with the given precedence constraints. Thus derived timing constraints are consistent with the precedence constraints. This can be done when there is only one processor.

condition

Note:- Release time → Effective release time
dead lines → Effective dead lines

- For only one processor and jobs are preemptable, we will ignore the precedence constraints.

EDF Algorithm (Earliest Deadline first)

In EDF algorithms, the jobs are scheduled depending on the earliest deadline among jobs. The jobs, which has highest chance of executing fast due to earliest deadline, is

given highest priority.

Example:- $J_1 (0, 10] \rightarrow J_2 (2, 12] \rightarrow J_3 (3, 18]$

deadline.

Scheduling the following jobs according to EDF.

J_1	J_2	J_3	J_4
$rt = 0$	$rt = 1$	$rt = 2$	$rt = 3$
deadline = 5	deadline = 3	deadline = 1	deadline = 2

Let's assume, J_3 has first deadline, then J_4, J_2 & J_1.

$J_3 \rightarrow J_4 \rightarrow J_2 \rightarrow J_1$ (As deadline)

J_3 is earliest which has deadline 1
J_4 is 2nd " " " 2
J_2 is 3rd " " " 3
J_1 is 4th " " " 5.

LST Algorithms (Least slack time)

release time

$J_1, 3 (0, 6]$

deadline

execution time

$J_2, 5 (10, 12)$

what is execution time of J_2? $\Rightarrow 5$

$d - t - [(e - t)]$

(Without preemption)

$$\text{Slack} = 6 - t - (3 - t) \quad (\text{when } t = t)$$
$$= 6 - \cancel{t} - 3 + \cancel{t}$$
$$= 3$$

$$S = d - t - (e - t)$$

changes when preempts.

Preempts

J_1	J_3	J_1

2 4

$$d - t - (e - t)$$
$$S = 6 - 4 - (3 - 2)$$

finish start

$$= 6 - 4 - 1$$
$$= 1 \quad (\text{slack of } J_1 \text{ at } 4)$$

$J_3, 3(2, 6)$

$J_1, 3(0, 6)$

J_1	J_3	J_1

2 4

t_1 t_2

$$\text{Slack} = d - t_2 - (e - t_1)$$
$$= 6 - 4(3 - 2)$$
$$= 2 - 1$$
$$= 1$$

The LST algorithm assigns priorities of jobs based on their slacks, the smaller the

slack, the higher the priority.

What happens when preemption is not applied in LST?

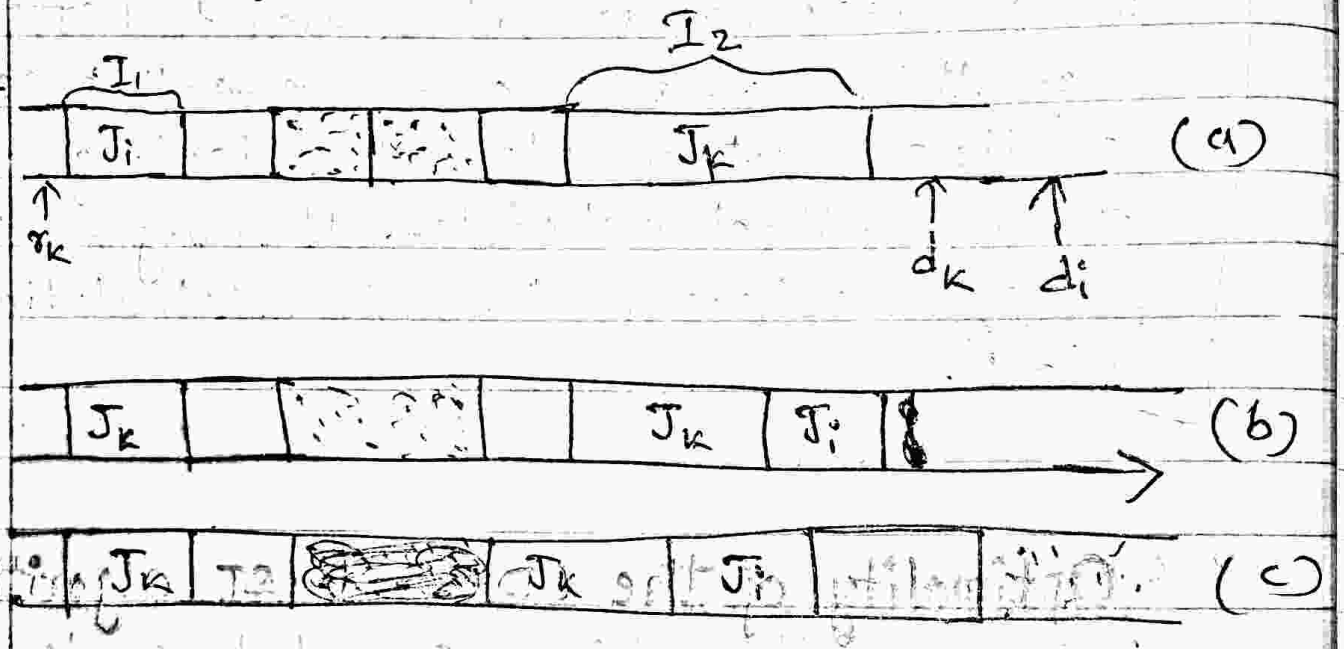
→ **Non optimal** (no any change)
When preemption is not applied to LST, the algorithm ~~leading to some non opti~~ becomes similar to other algorithm leading to some non-optimal schedules. It may create simply wastage of time for calculating slack.

4.6 Optimality of the EDF and LST algorithms

The priority-driven scheduling algorithm based on "earlier the deadline - higher the priority" is called the Earliest-Deadline-First (EDF) algorithm. This algorithm is important because it is optimal when used to schedule jobs on a processor as long as preemption is allowed and jobs donot contend for resources.

Theorem 4.1: When preemption is allowed and jobs donot contend resources, the EDF algorithm can produce a feasible schedule of a set J of jobs with arbitrary release times and deadlines on a processor if and only if J has feasible schedules.

Proof:-



Proof fact:- Any feasible schedule of J can be systematically transformed into an EDF schedule (i.e. a schedule produced by the EDF algorithm). To see why, suppose that in a schedule, parts of J_i & J_k ~~may be later than~~ are scheduled in intervals I_1 and I_2 respectively. furthermore, the deadline d_i of J_i is later than the deadline d_k of J_k, but I_1 is earlier than I_2 as shown in fig.

There are two cases. In the first case, the release time of J_k may be later than the end of I_1. J_k cannot be scheduled in I_1: the two jobs are already scheduled on the EDF basis in these intervals. Hence, we need to consider only the second case where the release time r_k of J_k is before the end of I_1; without loss of generality, we assume that r_k is no later than the begining of I_1.

To transform the given schedule, we swap J_i and J_k. Specifically, if the interval I_1 is shorter than I_2, as shown in figure 4-4, we move the portion of J_k that fits in I_1 forward to I_1 and move the entire portion of J_i scheduled in I_1 backward to I_2 and place it after J_k. The result is as shown in fig 4-4(b). Clearly, this swap is always possible. We can do a similar swap if the interval I_1 is longer than I_2: we move the entire portion of J_k scheduled in I_2 to I_1 and place it before J_i and move the portion of J_i that fits in I_2 to the interval. The result of this swap is that these two jobs are now scheduled on the EDF basis.

We repeat this transformation for every pair of jobs that are not scheduled on the EDF basis according to the given non-EDF schedule until no such pair exists.

fig (b) is not still EDF schedule since it has some interval is left idle. We can eliminate such an idle interval by moving one or more of these jobs forward into the idle interval and leave the interval where the jobs were scheduled idle. This clearly always possible. We repeate this process if necessary until the processor never idles when there are jobs ready for execution as in fig (c)

Hence proved.

LRT Algorithm (Latest Release Time)

In LRT algorithm, the priorities are based on the release times of jobs: the later the release time, the higher the "priority". Because it may leave the processor idle when there are jobs ready for execution, the LRT algorithm is not a priority-driven algorithm

Working:-

This algorithm treats release time as deadlines and deadlines as release times and schedules the jobs backwards, starting from the latest deadline of all jobs, in

"priority- ~~based~~ driven" manner, to the current time.

LRT

Reason for EDF algorithm :-

When the goal of scheduling is to meet deadlines, there is no advantage to completing any job sooner than necessary. So, sometimes, we postpone the execution of hard real-time jobs [for some reasons & eg. to enable soft-real-time jobs, whose response times are important, to complete earlier). For this reason, we sometimes use the LRT algorithm (or Reverse EDF algorithm).

Example of LRT algorithm

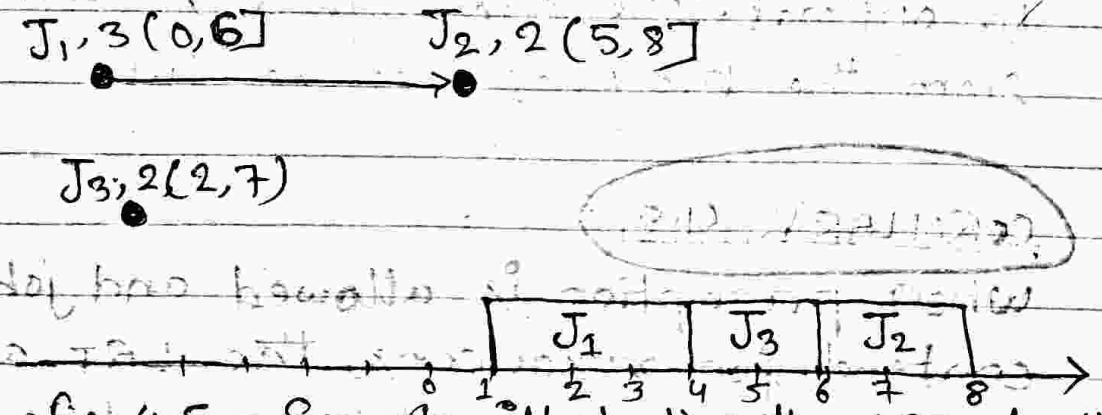


fig 4-5 Example illustrating the LRT algorithm

The latest deadline among jobs is 8. Hence time starts at 8 and goes backward to 0. At time 8, J_2 is "ready" and is scheduled. At time

7, J_3 is also "ready" to be scheduled, but because J_2 has a later release time, it has a higher priority. Consequently, J_2 is scheduled from 7 to 6. [Since its execution time is 2, so, J_2 schedules 8 to 6 backward]

When J_2 completes at time 6, J_1 is "ready" [Since it's deadline as release time is also 6]. However, J_3 has a higher priority and is, therefore, scheduled from 6 to 4.

Finally J_1 is scheduled from 4 to 1. This result is a feasible schedule.

<u>Note</u>- The LRT algorithm is also optimal under the same conditions that the EDF algorithm is optimal. Its proof follows straightforwardly from the proof of theorem 4.1.

COROLLARY 4.2

When preemption is allowed and jobs do not contend for resources, the LRT algorithm can produce a feasible schedule of a set J of jobs with arbitrary release times and deadlines on a processor iff feasible schedules of J exist.

MLF (Minimum-Laxity-First) algorithm.

LST Algorithm (Least-slack-time-First)

The LST algorithm assigns priorities to jobs based on their slack: the smaller the slack, the higher the priority.

Slack (or laxity) of a job :-

At any time t, the slack (or laxity) of a job with deadline at d is equal to

$d - t -$ (the time required to complete the remaining portion of the job.)

i.e.

$$\boxed{S = \underbrace{d}_{\text{deadline}} - t - \underbrace{(e - t)}_{\text{execution time}} \quad \text{for some } t.}$$

For example :-

At $t = 0$, $\quad J_1, 3\,(0, 6]$

$\therefore S = 6 - t - (3 - t)$ $\quad$ (3: execution time, 0: release time, 6: deadline)

$= \cancel{\text{At } t=0,}\ 6 - \cancel{t} - 3 + \cancel{t}$

$= 3$

Now, suppose J_1 is preempted by job

$J_3, 2\,(2, 7]$

which executes from time 2 to 4.

i.e.

J_1	J_3	J_1

0 1 2 3 4 5

t_1 t_2

$S = d - t_2 - (e - t_1)$ when job J_1 is preempted.

$= 6 - 4 - (3 - 2)$ → remaining time to complete the remaining portion of the job.

$= 1$

* The following theorem states that LST algorithm is also optimal.

Theorem 4.3

when preemption is allowed and jobs do not contend for resources, the LST (MLF) algorithm can produce a feasible schedule of a set J of jobs with arbitrary release times and deadlines on a processor iff feasible schedules of J exist.

(Limitations)

Disadvantages of LST algorithm:-

* It requires the knowledge of the execution times of jobs (which EDF algo does not.)

* The actual execution times of jobs are often not known until the jobs completes.

Thus it is impossible to calculate the actual amounts of slack under this circumstance. So, we typically calculate the slack of each job based on its maximum execution time e_i^+ when the range $[e_i^-, e_i^+]$ of execution time e_i of every job is relatively small. We ~~should~~ require the maximum (or actual) execution time of each sporadic or aperiodic job become known upon its arrival.

(time) Since this knowledge is needed for slack computation.

Non optimality of The EDF and LST Algorithm

When preemption is not allowed or there is more than one processor then the ~~LST~~ EDF and LST algorithm provide nonoptimality.

Case 1: non-preemption.

For example:- $J_1, 3(0, 10]$, $J_2, 6[2, 14]$ and $J_3, 4(4, 12]$ are three independent, nonpreemptable jobs. Figure (a) shows the schedule produced by the EDF algorithm. In particular, when J_1 completes at time 3, J_2 has already been released but not J_3. Hence, J_2 is scheduled. When J_3 is released at time 4, J_2 is executing. Even though J_3 has an earlier deadline and, hence, a higher priority, it must wait until J_2 completes because preemption is not allowed. As a result, J_3 misses its deadline. Similarly, it can be the result of infeasible scheduling with LST algorithm.

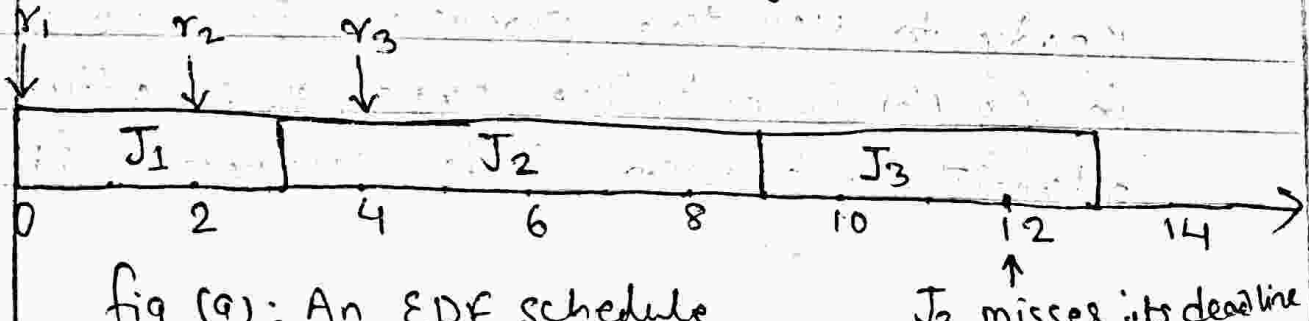


fig (a): An EDF schedule

fig (a): when jobs are nonpreemptive, the EDF algorithm is non-optimal (produce infeasible schedule because J_3 misses its deadlines)

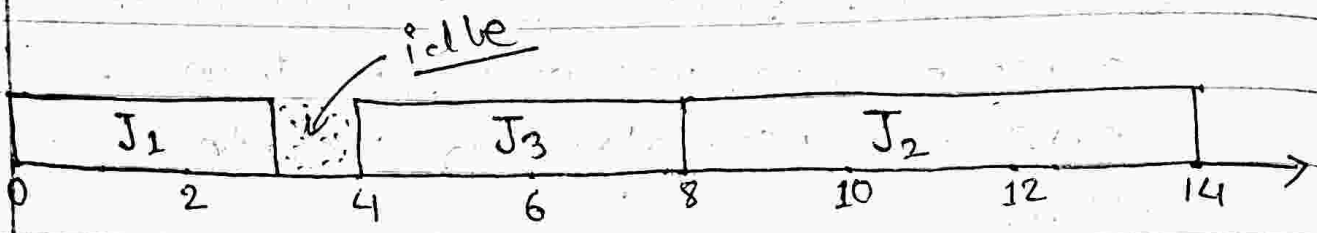


fig (b): feasible schedule) A non-EDF schedule

The fact that these three jobs can meet their deadline is demonstrated by the feasible schedule in fig (b).
At time 3 when J_1 completes, the processor is left idle, even though J_2 is ready for execution. Consequently, when J_3 is released at 4, it can be scheduled ahead of J_2, allowing both jobs to meet their deadlines.

By definition, priority-driven algorithm never leaves a processor idle when there are jobs ready to use the processor. So, the schedule in fig (b) is not the case of priority-driven algorithm. Hence, this example illustrates

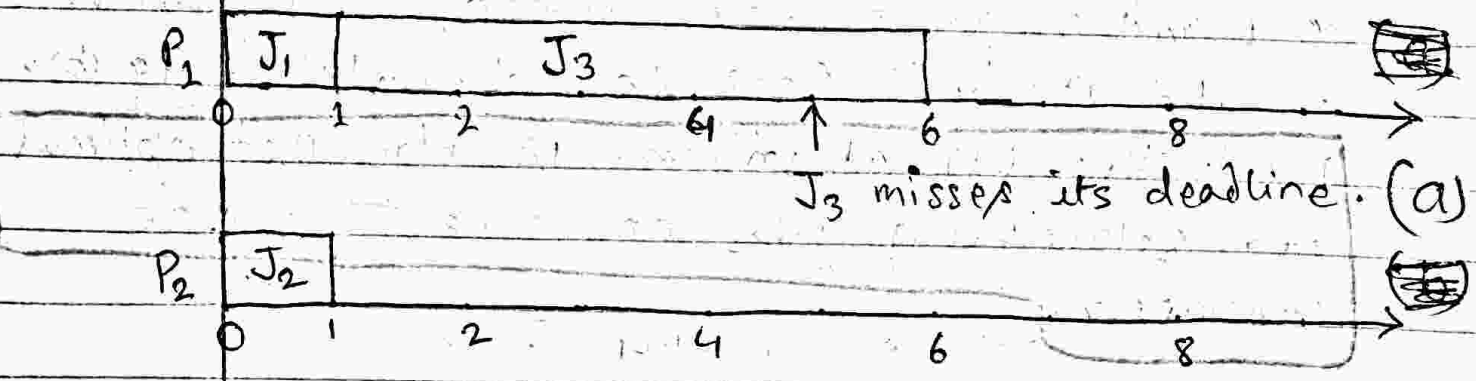
that "Not only non preemptive EDF and LST algorithms are not optimal, but also no ~~pr~~ nonpreemptive priority-driven algorithm is optimal when job have arbitrary release times, execution times, and deadlines."

Case 2:- More than one processor:- The jobs are non preemptive and scheduled on two processor P_1 and P_2.

$J_1, 1 (0, 1]$

$J_2, 1 (0, 2]$

$J_3, 5 (0, 5]$



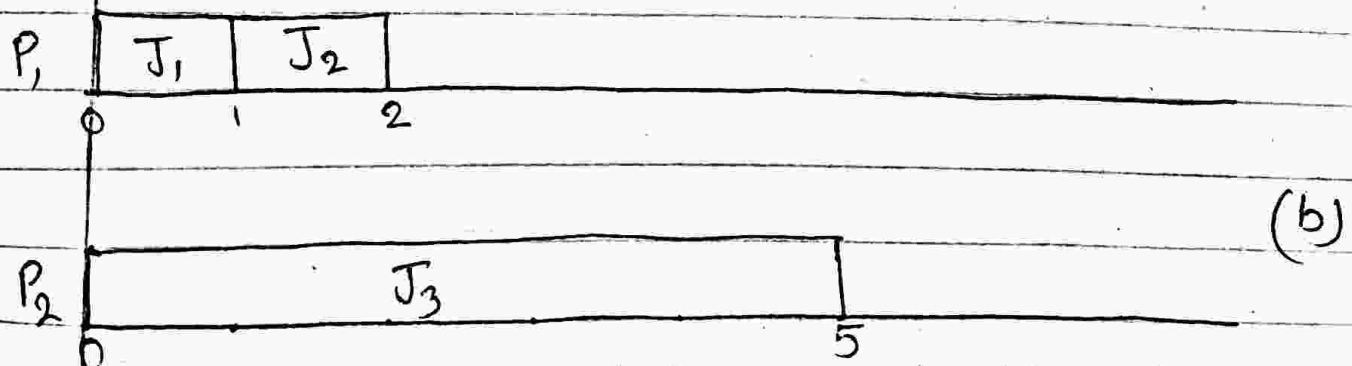


Fig:- Example illustrating nonoptimality of the EDF algorithm for multiprocessor scheduling. (a)

Fig:- (a) The EDF schedule (b) A feasible schedule.

Explanation

The system has two processor P_1 & P_2. According to EDF algorithm, J_1 and J_2 are scheduled on the processors at time 0 because they have higher priorities. The result is the schedule in fig (a), where J_3 misses its deadline.

On the other hand, an algorithm which assigns a higher priority to J_3 in this case can feasibly schedule the jobs. An example of such algorithms is the LST algorithm. The slacks of the J_1, J_2 and J_3 in fig: are 0, 1 and 0 respectively. Hence, this algorithm would produce feasible schedule in fig (b).

But the LST algorithm is also not optimal for scheduling jobs on more than one processor.

Problem 4.4 See.

CHALLENGES IN VALIDATING TIMING constraints in priority-driven system

a) Anamalous Behaviour of Priority-Driven ~~scheduling~~ Systems.
b) Predictability of Executions
c) Validation Algorithms and their performance.

→ A clock-driven scheduler requires the information on the release times and execution times of the jobs a priori in order to decide when to ~~decide~~ schedule them.
In contrast, a priority-driven scheduler does not require most of this information, making it much better suited for applications with varying time and resource requirements.

→ The priority-driven approach has not been widely used in hard real-time systems, especially safety-critical systems.

a) **Anomalous Behaviour of Priority Driven Systems**
The release times, deadlines, and execution times of the jobs are listed in the table. The execution times of all the jobs are fixed and known, except for J_2. Its execution time can

be any value in the range [2, 6].

	r_i	d_i	$[e_i^-, e_i^+]$
J_1	0	10	5
J_2	0	10	[2, 6]
J_3	4	15	8
J_4	0	20	10

Here jobs are preempted but never migrated.

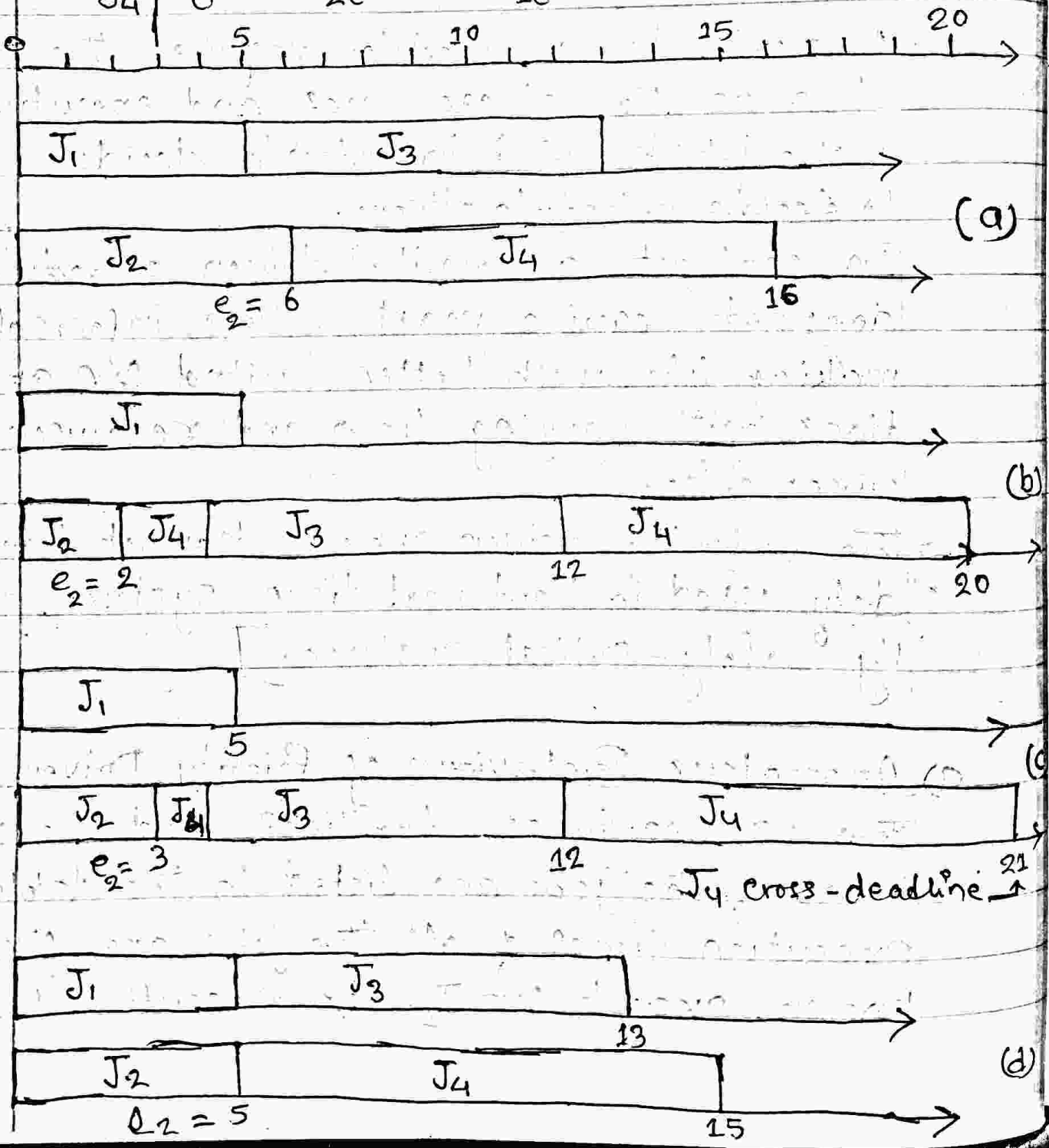


fig 4-8 ; Example illustrating scheduling anomalies.

→ when $e_2 = 6$, ~~and~~ as shown in fig (a) & $e_2 = 2$ as shown in fig (b), all jobs meet their deadlines and completion time jitters are sufficiently small.

→ when $e_2 = 3$, the schedule is as shown in fig 4-8 (c), then the Job ~~Schedule~~ J_4 misses its deadline. (Worst-case)

→ The best case schedule for J_4 is as shown in fig (d) when $e_2 = 5$. Then J_4 can complete as early as time 15.

Hence the phenomenon illustrated by this example is known as a scheduling anomaly, since it shows an unexpected timing behavior of priority-driven systems.

→ These anomalies make validating a priority-driven system difficult whenever job parameters may vary.

Note → If the maximum range of execution times of all n jobs in a system is X, the time required to find the latest and earliest completion times of all jobs is $O(X^n)$.

(b) Predictability of Executions :-

→ **Maximal & minimal schedule :-**

The schedule of J produced by the given scheduling algorithm when the execution time of every job has its maximum value is called maximal schedule of J.
And when it is minimum value then it is called minimal schedule.

→ **Actual schedule :-**

When the execution time for every job has its actual value, the resultant schedule is the actual schedule of J.

example:
- fig (a) has maximal schedule.
- and fig (b) has minimal "
- and all other schedules are possible actual schedules.

> **Note :-** Release times of all jobs are fixed and known for this discussion

→ The execution of J under the given priority-driven scheduling algorithm is predictable if the actual start time and actual completion time of every job according to the actual schedule are bounded by its start times and completion times accor-

No anomalies ~~for~~ to be predictable Scheduling.

ding to the maximal and minimal schedules. Predictability can be defined by:

① start-time predictable

② Completion-time predictable.

Let $S(J_i)$ be the instant of time at which the execution of J_i begins according to the actual schedule of J. $S(J_i)$ is the (actual) start time of J_i. Let $s^+(J_i)$ and $s^-(J_i)$ be the ~~actual~~ start times of J_i according to the maximal and minimal schedule of J, respectively.

→ We say that J_i is start-time predictable

if $s^-(J_i) \le S(J_i) \le s^+(J_i)$.

for example:- for job J_4 in fig above, $s^-(J_4)$ is 2. $s^+(J_4)$ is 6. Its actual start time is in the range [2,6]. Therefore J_4 is start-time predictable.

→ Let $f(J_i)$ be the actual completion time (also called finishing time) of J_i according to the actual schedule of J. Let $f^+(J_i)$ and $f^-(J_i)$ be the completion times of J_i according to the maximal schedule and minimal schedule of J, respectively. We say that J_i is completed time predictable

if $f^-(J_i) \le f(J_i) \le f^+(J_i)$.

Predictability of ~~job~~ behavior depends not only on the parameter of jobs in the set but also on the algorithm used to schedule.

→ Any job J_i is predictable if J_i is both start time and completion time predictable.

→ J is said to be predictable if each J_i is predictable.

<u>for example</u>: in the figure above:

$$f^-(J_4) = 20 \quad \text{and} \quad f^+(J_4) = 16$$

So, $20 \le f(J_4) \le 16$ which not holds.

Therefore, J_4 is not complete-time predictable.

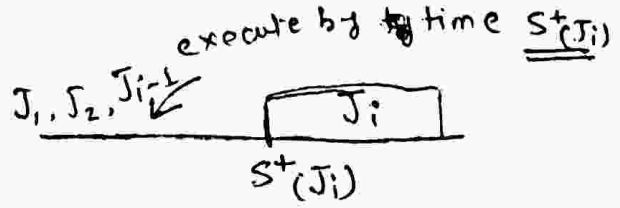
<u>Theorem 4.4</u>:-

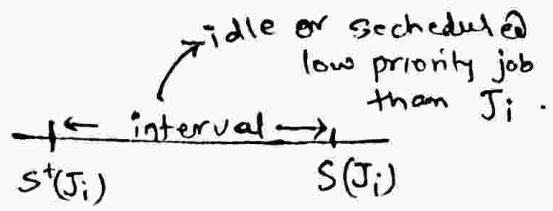
The execution of every job in a set of independent, preemptable jobs with fixed release times is <u>predictable</u> when scheduling in a priority-driven manner on one processor.

<u>Proof</u>: That the highest priority job J_1 is predictable is obvious: It always starts at its release time, and its maximum execution time is larger than its actual execution time. Suppose that all the $i-1$ highest priority jobs $J_1, J_2, \ldots, J_{i-1}$ are predictable. ~~We now show by contra~~

We now show by contradiction that J_i, which has a lower priority than them but a higher priority than all the other jobs in the system, is also predictable.

Suppose that $s^-(J_i) \le s(J_i) \le s^+(J_i)$ is not true. In particular, we suppose that $s(J_i) > s^+(J_i)$. Because the scheduler never schedule





a job before its release time, $s^+(J_i)$ is no earlier than the release r_i of J_i. Because the scheduling algorithm is priority-driven, every job whose release time is at or earlier than $s^+(J_i)$ and whose priorities are higher than J_i has completed by $s^+(J_i)$ according to the maximal schedule. By induction hypothesis, we know that every such job has completed by $s^+(J_i)$ according to the actual schedule as well.

$s(J_i) > s^+(J_i)$ means ~~that~~ either that the processor is left idle in the interval $[s^+(J_i), s(J_i)]$ or a job with a priority lower than J_i is scheduled in this interval. This contradicts the fact that the algorithm used to schedule the jobs is priority-driven.

Hence, $s(J_i) > s^+(J_i)$ cannot be true.

Similarly, $s^-(J_i) > s(J_i)$ cannot be true.

Therefore, J_i is start time predictable.

To show that J_i is also complete time predictable, we note that since

$$s(J_i) \le s^+(J_i),$$

there is more available time in the interval $[s(J_i), f^+(J_i)]$ than in the interval $[s^+(J_i), f^+(J_i)]$. Moreover, the actual execution time of every job is never larger than the

maximum execution time of the job. If J_i remains incomplete at $f^+(J_i)$ according to the actual schedule while it can complete by this time according to the maximal schedule, it must be delayed by one or more jobs with priorities lower than J_i executing in $[s(J_i), f^+(J_i)]$, or the processor must be left idle for some time in this interval. This contradicts the fact that the scheduling algorithm is priority-driven. ~~We can therefore conclude that the scheduling algorithm is priority-driven~~ We can therefore conclude that $f(J_i) \leq f^+(J_i)$.

Similarly, $f^-(J_i) \leq f(J_i)$

Hence J_i is also completion time predictable.

How does

(c) Validation algorithms and their performance? challenge for Priority-driven Sched

See book or photo copy.

Q.9 Define Off-line scheduling Vs. On-line Scheduling: what are their adv. and disadvantages

Q(2) Distinguish between EDF, LST & clock driven approach.

Assignment 3

Q.2 What happens when preemption is not applied in LST?

2070-10-29

Chapter 5

Clock-Driven Scheduling

Notations and Assumptions.

1. There are n periodic tasks in a system and are fixed when in operation mode.
2. All parameters of the periodic tasks are known before execution. The variation in inter release times between jobs are negligibly small. [i.e. P_i units of time for all jobs]
3. Each jobs $J_{i,k}$ is ready for execution at its release time $r_{i,k}$.

Notations:- A periodic Task ~~are~~ is represented by 4-tuples (ϕ_i, p_i, e_i, D_i)

phase, period, execution time, Deadline.

for example:-

$T_1 = (1, 10, 3, 6)$ $T_2 = (10, 3, 6)$ & $T_3 = (10, 3)$

$\Downarrow$

$\phi_1 = 1$ $\boxed{\phi_2 = 0}$ ← Default → $\boxed{\phi_3 = 0}$

$p_1 = 10$ $p_2 = 10$ $p_3 = 10$

$e_1 = 3$ $e_2 = 3$ $e_3 = 3$

$D_1 = 6$ $D_2 = 6$ $\boxed{D_3 = 10}$

Static Timer Scheduler

Definition

→ Aperiodic jobs are queued and maintained by OS. Whenever the parameters of jobs with hard deadlines are known before the system begins to execute, a straightforward way to ensure that they meet their deadlines is to construct a static schedule of the jobs, off-line. This schedule specifies exactly when each job executes. According to the schedule, the amount of processor time allocated to every job is equal to its maximum execution time, and every job completes by its deadline. During run time, the scheduler dispatches the jobs according to this schedule.

Working

→ Straight forward way to implement the scheduler is to store the precomputed schedule as a table. Each entry $(t_k, T(t_k))$ in this table gives a decision time t_k, which is an instant when a scheduling decision is made, and $T(t_k)$, which is either the name of the task whose job should start at t_k or I (idle). During initialization (say at time 0), the operating system creates all the tasks that are to be executed. The scheduler makes use of a timer. Immediately after

all the tasks have been created and initialized and then at every scheduling decision time, the scheduler sets the timer so the timer will expires and request an interrupt at the next decision time. Upon receiving the timer interrupt at t_k, the scheduler sets the timer to expire at t_{k+1} and prepare the task $T(t_k)$ for execution. It then suspends itself letting the task have the processor and ~~execute~~ execute. When the timer expires again, the scheduler repeats this operation.

for example:-

A system has four independent periodic tasks.

$T_1 = (4, 1)$, $T_2 = (5, 1.8)$, $T_3 = (20, 1)$ & $T_4 = (20, 2)$

Their utilizations are:- $\left(\frac{e_i}{p_i}\right)$

$u_1 = 0.25$, $u_2 = 0.36$, $u_3 = (0.05)$ and $u_4 = 0.1$

Therefore, total utilization = 0.76

Hyper period (H) = L.C.M of 4, 5, 20, 20

= 20

Therefore, entire schedule consists of replicated segments of length 20. Fig 5-1, shows such a schedule segment.

Each hyper periods consists of $\left(\frac{20}{4} + \frac{20}{5} + \frac{20}{20} + \frac{20}{20}\right)$

= 11 tasks as in figure.

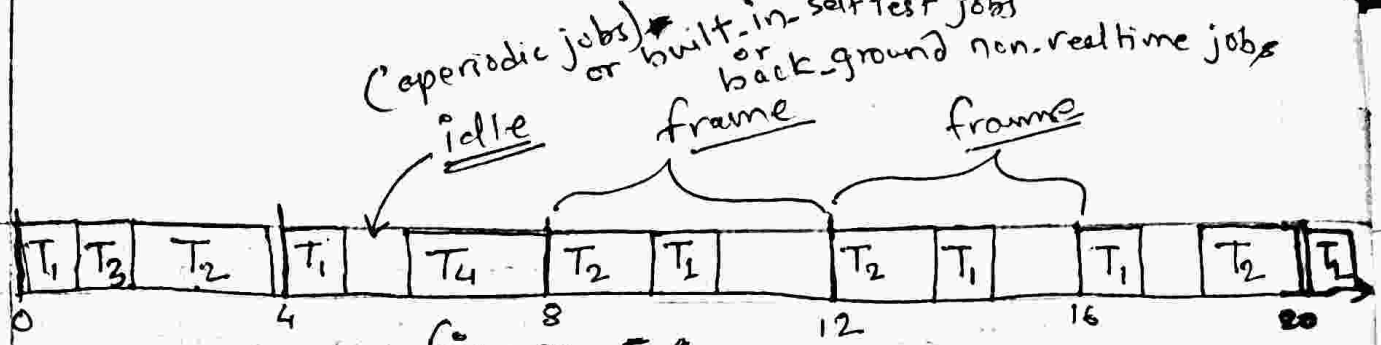


figure 5-1

(An arbitrary static schedule)

Here T_1 start execution at time 0, 4, 9.8, 13.8 and so on; T_2 starts execution at 2, 8, 12, 18 and so on.

In the example above, the stored table contains 17 entries. They are $(0, T_1)$, $(1, T_3)$, $(2, T_2)$, $(3.8, I)$ $(4, T_1)$, --- $(19.8, I)$. Hence the timer is set to expires at 0, 1, 2, 3.8, and so on. At these times the ~~timer~~ scheduler schedules the execution of tasks T_1, T_3, T_2, and an aperiodic or background job respectively.

Cyclic Schedule :-

A periodic static schedule is called a cyclic schedule.

And this approach to scheduling hard real time jobs is called the clock-driven or time-driven scheduling, because each scheduling decision is made at a specific time, ~~independent of events~~

General Structure of cyclic Schedules.

It consists of following characteristics:-

a) frames and Major cycles

b) frame size constraints

c) Job slices.

a) Frames and Major cycles

General structure of cyclic schedule impose a restriction that scheduling decisions are made periodically, rather than arbitrary times. The scheduling decisions times partition the time line into intervals called frames. Every frame has length f (frame size). Because scheduling decisions are made only at the begining of every frame, there is no preemption within each frame. And the first job of every task is released at the begining of some frame.

Major cycles:-

To keep the length of the cyclic schedule as short as possible, a frame size f, ~~should be~~ is so choosen that it divides H. This condition is met when any p_i is divided by f.

i.e. $p_i / f = 0$.

Then, the Resulting hyperperiod is called major cycle that consists of integer number of frames.

b) <u>Frame size constraints</u>:- The structure of cyclic schedule is constrained by frame size in order to achieve:-

① → every job starts and completes its execution within a frame. (no job will be preempted)

Given by relation —

$$\boxed{f \geq \max_{1 \leq i \leq n} (e_i)} \quad \text{—— (1)}$$

→ makes f larger

② To keep length of cyclic schedule as short as possible. [create major cycle]

i.e. $\boxed{P_i / f = 0}$ <u>for atleast one P_i</u> (2)

when this condition is met, there is an integer number of frames in each hyper period.

③ To ensure deadlines of every jobs. This is achieved by ~~reducing~~ sufficiently reducing the frame size. (fig 5-4 illustrate this.)

And at least one frame should be enclosed by release time and dead line of every job.

This condition is met by

$$\boxed{2f - \gcd(P_i, f) \leq D_i} \quad \text{—— (3)}$$

$\gcd(P_i, f) = t' - t$

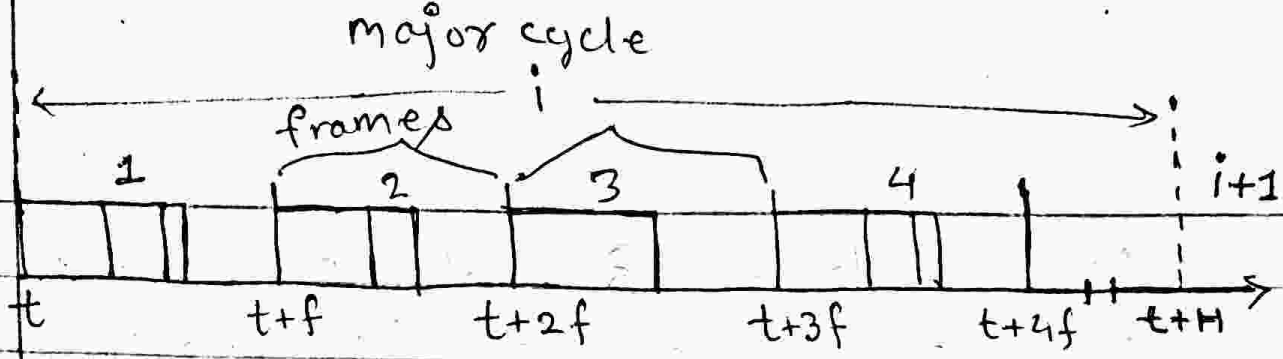


Fig 5-3 General structure of a cyclic schedule.

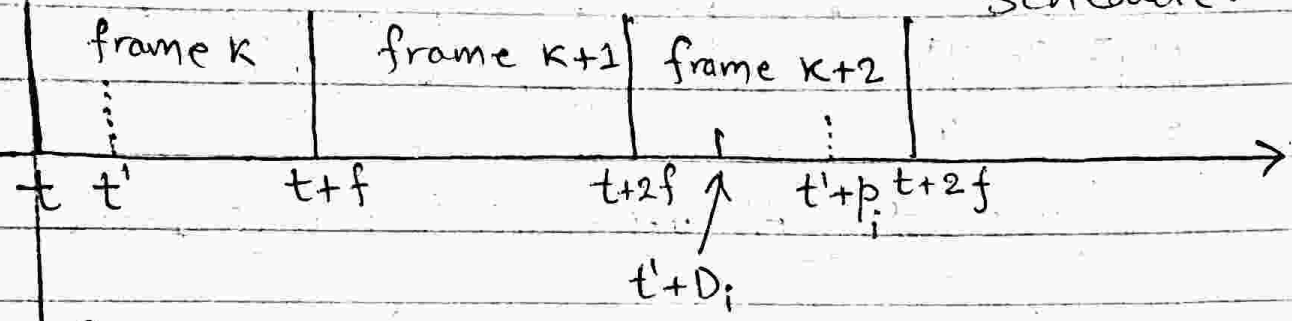


figure (5-4) :- A constraint on the value of frame size.

We refer to Eqs (5.1), (5.2) and (5.3) as the frame-size constraints.

~~Job slices~~ Example:

$\{T_1 = (4,1),\ T_2 = (5, 1.8),\ T_3 = (20, 1)$ and $T_4 = (20, 2)$

To be frame size constraint :-

(1) $f \geq \max(e_i) \Rightarrow \max\{1, 1.8, 1, 2\}$

Therefore, according to eqn (1), frame size should be no less than 2.

(2) According to eqn (2). i.e. $p_i / f = 0$ (for at least one p_i)

since $H = 20$, so possible frame size are $\boxed{2, 4, 5, 10, 20}$ [$\because f \geq 2$ and f divides H

③ According to eqn ③, only 2 satisfies the condition. Hence we must choose frame size = 2. As shown in fig 5.5

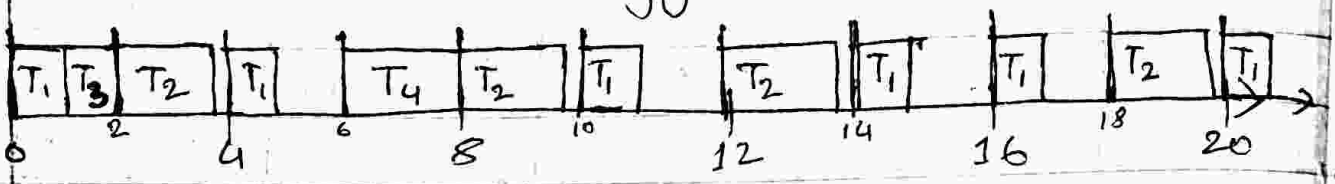


Figure 5-5 A cyclic schedule with frame size 2.

ⓒ Job slices

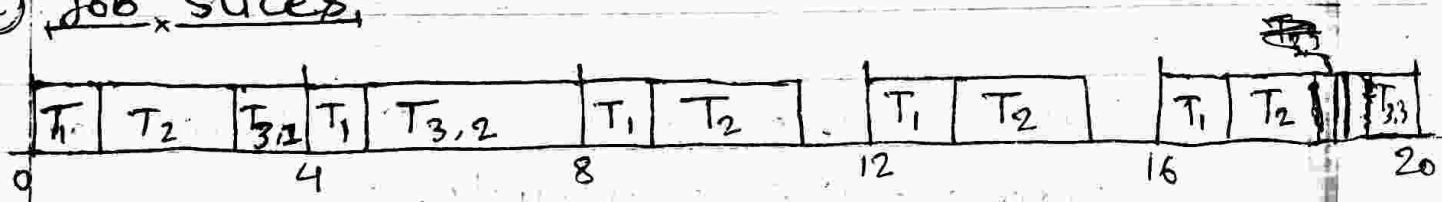


Fig 5-6 A preemptive cyclic schedule of $T_1 = (4,1)$ $T_2 = (5,2,7)$ and $T_3 = (20,5)$

Sometimes, the given parameters of some task cannot meet all ~~these~~ their frame size constraints simultaneously. So in order to satisfy all the conditions of frame size constraints we need to slice ~~the~~ Jobs of given task into ~~sub tasks~~ smaller partitions. This is called Job slices.

In the process of cyclic schedule, we have three kinds of design decisions: choosing a frame size, partitioning jobs into slices and

placing slices in the frames. In general this decisions cannot be made independently. The more slices the job is, the higher the context switch and communication overhead. Therefore, we need to partition each job into as few slices as necessary to meet the frame size constraints.

for example:- $T_1(4,1)$, $T_2(5,2,7)$ & $T_3(20,5)$

Here, (3) For eqn (1) to be true, we must have $f \geq 5$, but to satisfy eqn (3), we must have $f \leq 4$.

Therefore we need to divide each job in $T_3(20,5)$ into a chain of three slices with execution time 1, 3, 1 so as to get sub tasks (20,1), (20,3) & (20,1). Now the resultant system has five tasks with frame size 4. This can be scheduled with ~~three~~ frame size constraints now as shown in fig. 5-6.

$\boxed{T_3 = T_{3,1}, T_{3,2} \text{ and } T_{3,3}}$ Note

5.4 Cyclic Executive :- The clock driven scheduler described above must be modified to accomodate the restrictions that

scheduling decision are made only at frame boundries. In real time systems literature, the term "cyclic executive" refers to a scheduler that deterministically interleaves and sequentialize the periodic tasks on a CPU according to a given cyclic schedule.

In essence, the cyclic executive takes over the processor and executes at each of the clock interrupts, which occurs at the beginning of frame.

See fig 5.7 Page 105

5.5 Improving the average Response Time of a aperiodic Jobs :-

Slack Stealing :

Simply, we can improve the response times of aperoidic jobs by executing the aperiodic jobs ahead of the periodic jobs whenever possible. This approach is called slack stealing was originally proposed for priority-driven systems.

Re-Exam mid-2014 2014-feb 17

class work:-
① Network-flow diagram
② Procs and cons of Clock-driven Scheduling.

- To make the ~~responsive~~ ~~AAO~~ system responsive, we need to minimize the average response of the aperiodic jobs. The following example illustrates this:

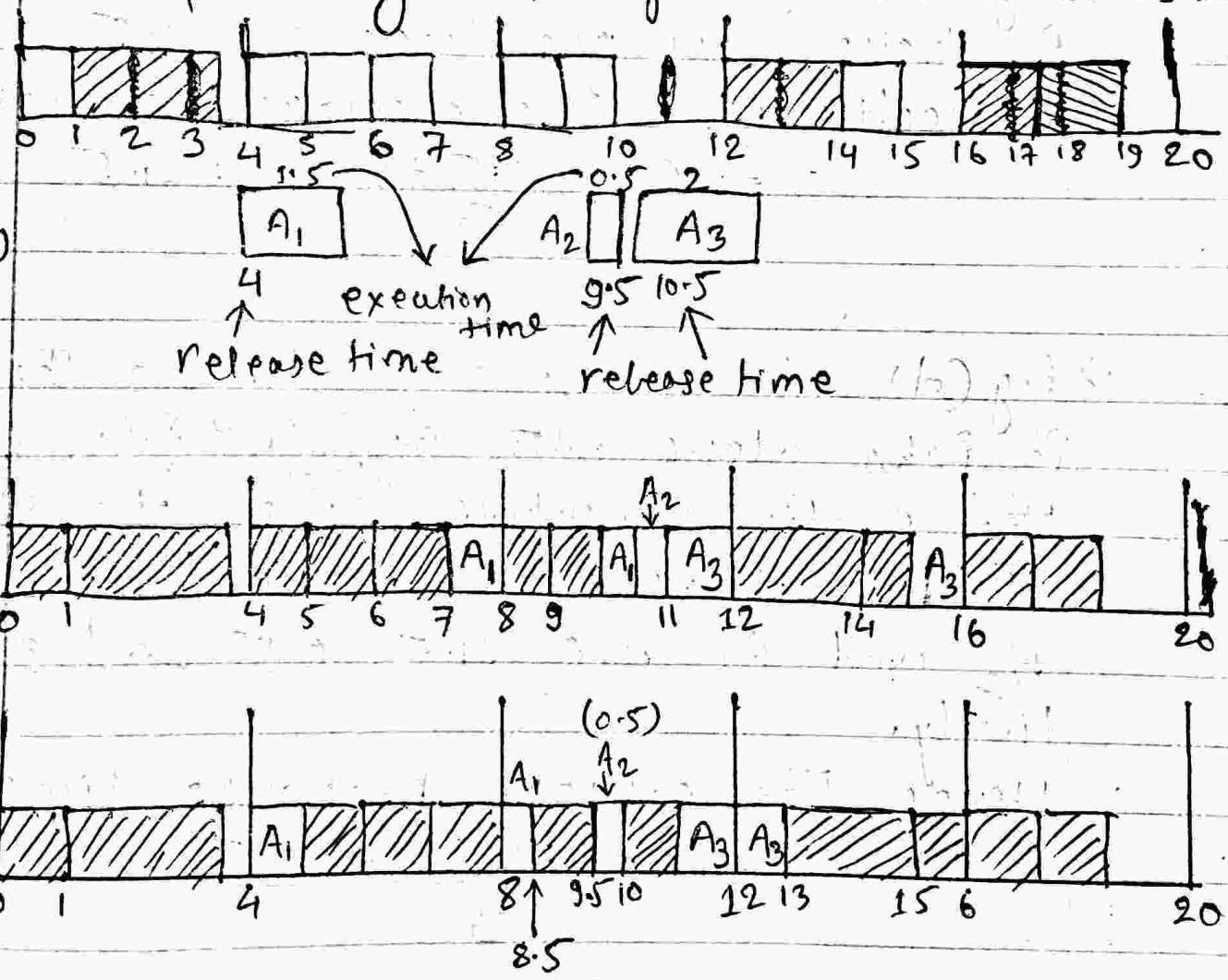


fig:- Example illustrating slack stealing.
(5-8)

Remaining Soln Back

Explanation :-

→ Fig (a) shows the first major cycle in the cyclic schedule of the periodic tasks.

→ Fig (b) shows three aperiodic job A_1, A_2 and A_3 with release time immediately before 4, 9.5, and 10.5 and their execution time 1.5, 0.5 and 2 respectively.

→ Fig (c), shows when the aperiodic jobs are executed in the slack periods of fig (a) by the cyclic executive. Here A_1, A_2 and A_3 have response times 6.5, 1.5 and 5.5 respectively. So the average response time

$$= \frac{6.5 + 1.5 + 5.5}{3} \Rightarrow 4.5$$

→ Fig (d) shows the scheduling of the aperiodic jobs along with periodic jobs, when the cyclic executive ~~steals~~ slides the slack stealing. Here, the response time of A_1, A_2 and A_3 are 4.5, 0.5 and 2.5 respectively.

Hence the average response time

$$= \frac{4.5 + 0.5 + 2.5}{3}$$

$$= 2.5$$

Therefore, slack stealing reduces the average response time of the aperiodic jobs.

gentle

Chapter 6

Priority-Driven Scheduling.

Theorem 6.1, 6.2, 6.3, 6.4

6.5 A Schedulibility Test for fixed-PRIORITY TASKS with ~~out~~ short Response Times

6.5.1. * Critical Instants Page 131

* Theorem 6.5
* Time demand analysis

6.6. Schedulibility Test for fixed-priority Tasks. with arbitrary response times.

* Busy Intervals: (Page 140)
* General Schedulibility (Page 141)

Chapter - 10

Model of Realtime communication

→ Architecture of RTC

→ Packet switch network

Busy Interval :-

A level-π_i busy interval $(t_0, t]$ begins at an instant t_0 when

(1) all jobs in T_i released before the instant have completed and

(2) a job in T_i is released.

In other words, in the interval $[t_0, t]$, the processor is busy all the time executing ~~interval, and at the end~~ jobs with priorities π_i or higher, all the jobs executed in the busy interval are released in the interval, and at the end of the interval there is no backlog of jobs to be executed afterwards.

Chapter 11

Architectural Overview

The circle marked TPH and NACH are the TP and NAC handlers, The directed lines show the data paths. when requested to send a message by a local application task, the source TP handlers places the message in the output queue. from there, each outgoing message is delivered to the network under the control of the NAC handler. After the message has traversed the network, the destination NAC handler places the message in the input queue and notifies the destination TP handler. The destination TP handler then moves the message to the address space of the destination TP handler. The destination TP handler then moves the message to the address space of the destination application task and notifies the application task of the arrival of the message.

Figure 11-1

A7 → IEEE 802.5 Token Rings.

→ IEEE 802.6 Distributed scheme for scheduling queue.

A6 → RSVP Protocol (Resource-Reservation Protocol)

A6 → Issues in Resource Reservation.

CSMA/CD - Carrier Sense Multiple Access with Collision detection.

A6 → ST-II, Internet Stream Protocol.
A5 → RTCP control protocol
A7 → Packet switching Network
A7 - Weighted Round-Robin service Disciplines

T.U.

1. Multiple priority ceiling protocol 366
2. Stack stealing in deadline-driven system with suitable example. ~~317~~ 247
3. Greedy weighted Round Robin and time weighted Round-Robin discipline with example. (459)
4. Schedulibility of sporadic jobs in deadline driven systems. (264)
5. Constant utilization server. (232)
6. RTCP control protocol (497)
7. Total Bandwidth server algorithm. (237)
8. Real time protocol (494)
9. Priority Based service disciplines for switched networks.
10. Communication in multiprocessor system. (499

Chapter 10

Real time communication.

Packet :- Messages are fragmented into segments for their suitable transmission through the communication network. Such segments of messages are called packets.

→ Is a basic transmission unit and is non preemptable. [Such unit is called a frame, a packet, a cell etc with respect to type of networks)

Example

In ATM network → a packet is 53 byte cell.
In IEE 802-5 token ring → it is a frame of tens of kilobits. etc.

* What is the time unit required to transmit a 53 byte packet over a 100-MHz link? (Ans. 4.24 micro sec) + Processing time.

→ length of message is measured by no. of packets.
For eg. a message of length 100 = 100 × 53 byte.
length of time = 100 time units.
= ? (100 × 4.24 micro sec)

Real Time Traffic models:-

The term realtime traffic typically means isochronous (or Synchronous) traffic, consisting of message streams that are generated by their source on a continuous basis and

delivered to their respective ~~source tasks~~ destination on a continuous basis. Such traffic includes periodic & sporadic messages, which require some degree of guarantee for on-time delivery. Also there are aperiodic (asynchronous) messages which have soft timing constraints, ~~and~~ are deliver on a best effort basis.

Periodic & Aperiodic messages:-

Periodic messages / periodic message streams are generated and consumed by periodic tasks and they have characteristics like their respective source tasks.

- Notation for periodic message M_i by the tuple (P_i, e_i, D_i) where,

  $P_i \rightarrow$ period (i.e. interarrival times of instance

  $e_i \rightarrow$ the maximum length of instances in M

  $D_i \rightarrow$ relative deadline of M_i.

The traffic model in which message M_i ~~with~~ (P_i, e_i, D_i) is transmitted with the constants P_i, e_i and D_i is called <u>peak rate model</u>.

Aperiodic Messages

→ There are also aperiodic message streams; the transmission of an aperiodic message stream is an aperiodic task. like aperiodic tasks, an aperiodic message stream does not have a relative deadline but it is desirable to have average delay as small as possible.

Sporadic Messages:-

In general, we can characterize a sporadic message stream M_i in the same way we characterizes a sporadic task.

[- Three models that are commonly used in real-time communication literature:

- the FeVe model
- the (λ, β) model
- the leaky bucket model.]

- According to FeVe model, a sporadic message M_i is characterized by a 5-tuple $(p_i, \bar{p}_i, I_i, e_i, D_i)$ where -

  $\bar{p}_i$ → is the average interarrival time of the instances of M_i, where the average is taken over an interval of length I_i.

- In switched network, a switch typically transmits each packet without waiting for the arrivals of later packets in the same message instance. The FeVe model in this case simplifies to $M_i = (p_i, \bar{p}_i, I_i, D_i)$.

page 273 to study further

→ A message stream meets the (Λ, E) leaky bucket constraint if it can be produced by a (Λ, E) leaky bucket filter.

(input rate, size)

See figure: 7-22 P-274

Many sporadic message that have seemingly different characteristic may satisfy this constraint. Among their common characteristic is that the length of any instance never exceeds E and the total length of all instances arriving in any time interval of length t never exceeds $E + \Lambda t$.

→ Similarly, according to (λ, β) model, the number of packets that arrive in any time interval of length t never exceeds $\beta + \lambda t$.

The miss rate of a message stream or a set of message streams is the fraction of all message instances or packets that are delivered to their destination too late.

Miss, loss and Invalid Rates

Delay Jitter, Buffer Requirement, and Throughput.

the fraction of message instances in the streams that are dropped en route for flow and congestion control reasons.

Priority-Based Service Disciplines for Switched Networks:-

Weighted Fair-Queueing Discipline. (WFQ)

Delay Jitter :- The delay jitter of a message stream is the variation (eg. the maximum or average absolute difference) in the delays suffered by different message instances or packets in the stream. ~~More jitter~~

larger delay Jitter of a message stream means that more buffer needed in the destination. So in Real time communication, we minimize delay jitter to small using some nongreedy synchronization protocols.

Throughput :- The throughput is a measure of the number of packets that the network can deliver per unit time.

Real time connection and Services Disciplines

According to the connection-oriented approach, a logical simplex connection from the source to the destination is set up for the transmission of each message stream.

→ The assumption is that all the packets on each connection are sent along a fixed route.

~~→ The fixed route for traffic on each connection provide high~~

→ Circuit switching provides the fixed route for packet transmission which enables high transmission rate in the network. which Packet Switching may not provide.

Packet switching Networks:-

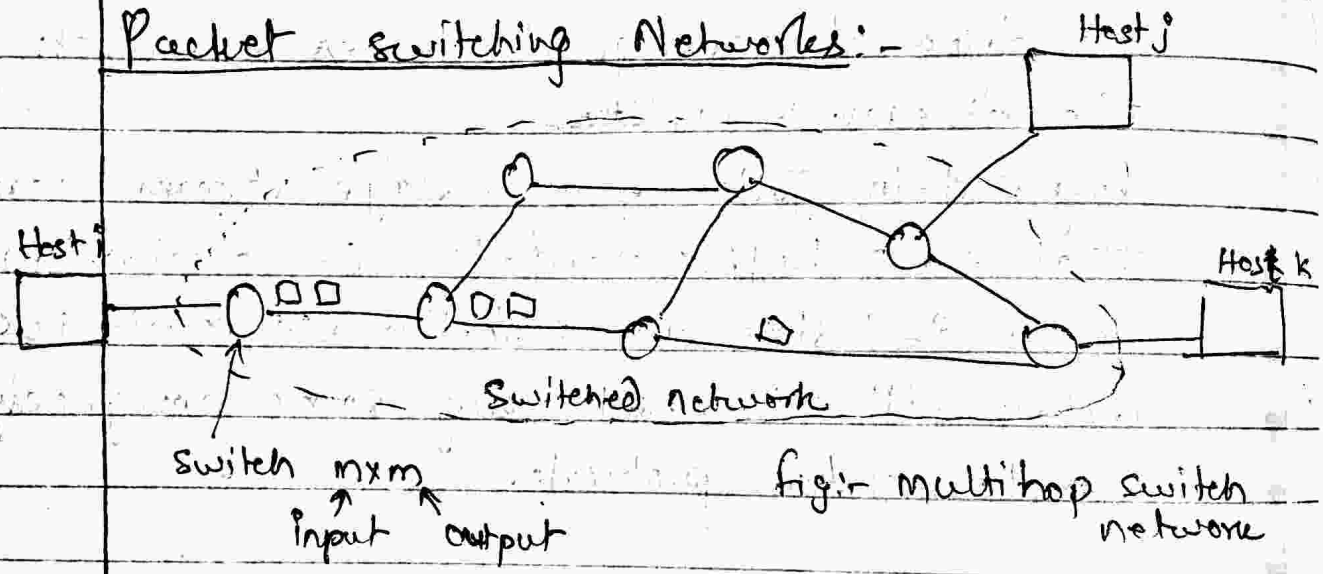


fig:- multihop switch network

→ Routes packets on input links.

→ We can represent switching patterns by a permutation of the m-tuples (1, 2, ... m).

for example:- for a 4×4 switch, the 4-tuple (2, 4, 1, 3) means the link like.

2 → 1

4 → 2

1 → 3

3 → 4

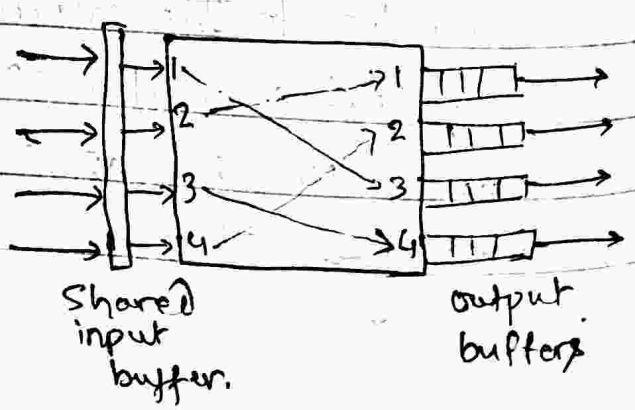


fig:- a 4×4 Switch.

→ The time required to transmit the packet onto the output link (packet transmission time) ~~and hence the~~ plus the waiting time at the output queue is called the per hop delay of the packet

→ The combination of an acceptance test and admission control protocol, an (execution) synchronization protocol and a scheduling algorithm used for a purpose of rate control, (delay) jitter control and scheduling of packet transmissions is called a service descipline.

(WFQ algorithm)

Waited fair - Queueing Discipline :-

The original version of weighted fair-Queueing (WFQ) algorithm, which is non preemptive, was proposed by Demers, for scheduling packet transmissions in switched networks. It is also known as the packet by packet generalized processor-sharing algorithm.

Here, n → denoted number of connections.

Each connection i → is allocated u_i of the link bandwidth.

$U = \sum_{i=1}^{n} u_i$ is the total link bandwidth allocated to n connections.

Scheduling Packets in WFQ :-

→ The switch is assumed to be output buffered.

→ Packets on output buffer with connection i being transmitted or waiting to be transmitted in FIFO basis.

→ The packet becomes ready for transmission when it reaches the head of the queue.

→ Connection i is idle when no connection-i packet is waiting or is being transmitted, otherwise, the connection is backlogged.

Scheduling ready packets on backlogged connections

SFN queue

→ Each backlogged connection (i) has an entry (fn_i, i) in the priority queue.

→ The entry gives the finish number fn_i of the connection; (or more precisely, the finish number of the ready packet on the connection)

→ It also gives the ID i of the connection.

→ The entries are sorted in order of finish numbers: the smaller the number, the earlier in the queue. Therefore, we call this queue the SFN (smallest

Finish Number → The finish number of a server gives the number of the round in which the server budget would be exhausted if the backlogged servers were scheduled. (consumed or used up the whole of)

finish Number first) queue.

→ The currently transmitting entry ~~scheduling~~ has highest order since scheduling is non preemptive. (exception)

→ Ready packets on backlogged connections are transmitted in the order of given SFN queue.

Page 444

Scheduling according to WFQ algorithms

→ when first packet in a busy interval of the output link arrives, the scheduler computes its finish number and enters this number and connection ID in the SFN queue. It then commences the transmission of the packet immediately.

→ During a link busy interval, the scheduler computes the finish number of each packet that arrives on an idle connection and inserts the corresponding entry into the SFN queue.

→ when transmission of packet completes then its entry is deleted and next packet is chosen in the following manner:-

(i) If connection i is still backlogged, the sche-

SFN → Smallest finish Number first.

duler computes the finish number of its new ready packet and inserts this number and connection ID in the SFN queue.

ii) It then commences the transmission of the ready packet on the connection identified by the first entry in the SFN queue.

Computing finish Numbers:-

Here,

U_b → total bandwidth of all backlogged connections.

FN → finish number of the link.

t → current time

t-1 → previous time instant when FN & U_b are updated.

n → no. of connection on output link of interest here

* Rules for computing first finish Number of a Link Busy Interval,

I1 for as long as the link is idle, FN = 0, U_b = 0, t_0-1 = 0 and finish number fn_i of connection i, for every i = 1, 2, ..., n, is 0.

I2 when the first packet of length e arrives and starts a busy interval of the link,

(a) Set t-1 = t; and

(b) The packet being a connection-i packet

GPS → General Processor Sharing Algorithm.

increment U_b by $\tilde{u}_i$, compute $fn_i = fn_i + e/\tilde{u}_i$ and insert the entry (fn_i, i) in the SFN queue.

* **Rules for computing Subsequent finish Numbers during a link Busy Interval:-**

R1 For every i, when a connection-i packet arrives at t during a link busy interval, if connection i was idle prior to this arrival,

a) increment FN by $(t - t_{-1})/U_b$, ~~compute~~ compute $fn_i = \max(FN, fn_i) + e/\tilde{u}_i$, and insert entry (fn_i, i) in the SFN queue; and

b) Set $t_{-1} = t$ and increment U_b by $\tilde{u}_i$.

R2 For every i, when the transmission of a connection-i packet completes.

a) If the connection remains backlogged, increment fn_i by $e/\tilde{u}_i$, where e is the length of the new ready connection-i packet, and insert the entry (fn_i, i) in the SFN-queue;

b) If connection i becomes idle,

(i) increment the link finish number FN by $(t - t_{-1})/U_b$; and

(ii) set $t_{-1} = t$ and decrement U_b by $\tilde{u}_i$.

See fig 7-14(d) (page 244)

Maximum delay and Queue length at the first switch :-

The delay between when a packet becomes ready (i.e. when it reaches the head of the FIFO queue of its connection) and when its transmission completes is called its latency.

~~Condition~~

Corollary 7.9

The latency of a packet on connection i that is allocated the fraction $\tilde{u}_i$ of link bandwidth is bound above by

$$\boxed{L_i = e_i/\tilde{u}_i + 1}$$

where e_i is the maximum length of all connection-i packet

$\frac{e_i}{\tilde{u}_i}$ → gives the amount of time required to transmit the largest connection-i packet on a slower link of bandwidth $\tilde{u}_i$ times that of the physical ~~length~~ link.

1 → maximum blocking time due to non-preemptivity

Because the rate-control function provided by a WFQ scheduler, this upper bound holds independent of the arrival rates of all other connections.

$\boxed{l \text{ or } 1}$ — ?? 1

Per hop delay = queueing delay + latency.

We now examine -

<u>Maximum per hop delay $W_i(1)$</u> that can be incurred by any packet on connection i as the packet traverses the first switch.

- The arrivals of packets on connection i satisfy the (u_i, E_i) leaky bucket constraint.
- The total time required ~~by~~ to complete the transmission of all packets that arrive at the start of the busy interval is no more than

$$E_i/u_i + 1$$

- from this observation, the per hop delay of any connection-i packet through the first switch suffered by a packet of length e is bounded from above by

$$\boxed{W_i(1) = (e + E_i)/\tilde{u}_i + 1}$$

Also the maximum queue length at first switch

$$= E_i + \tilde{u}_i$$

Since each packet is held in the buffer until its last bit is transmitted, the required buffer size is $E_i + e_i + \tilde{u}_i$. (when $\tilde{u}_i = u_i$)

chapter 7

Constant Utilization Server Algorithm :-

for the purpose of executing aperiodic jobs there is a basic constant utilization server. The server is defined by its size, which is its instantaneous utilisation $\tilde{u}_s$; the fraction of processor time is reserved for the execution of aperiodic jobs. As with deferrable servers, the deadline d of a constant utilization server is always defined. The server is eligible and ready for execution only when its budget is nonzero. When the server is ready, it is scheduled with the periodic tasks on the EDF basis. A constant utilization server emulates a sporadic task with a constant instantaneous utilisation.

Consumption & Replenishment Rules :-

From Previous chapter 5

~~RTS~~ ~~Send-Up Exam-2014~~

~~F.M:- 80~~
~~P.M:- 32~~
~~Time: 3 hr~~

Scheduling Sporadic Jobs.

Syllabus

- → See slide
- ✓ Acceptance Test (Slide).
- → EDF scheduling of the Accepted Jobs. (with example)
- → Implementation of the acceptance Test. X

EDF Scheduling of The Accepted sporadic Jobs:-

→ Since EDF can give optimal scheduling, it is good way to ~~to~~ schedule accepted sporadic jobs in EDF scheduling. Therefore, the scheduler maintains a queue of accepted sporadic jobs in non-decreasing order of their deadlines and inserts each newly accepted sporadic job into this queue in this order.

→ Whenever all the slices of periodic tasks scheduled in each frame are completed, the cyclic executive lets the jobs in the sporadic job queue execute in the order they appear in the queue.

→ Here, the cyclic executive integrates the

scheduling of sporadic jobs with that of aperiodic and periodic jobs. The cyclic executive assumes that each newly released sporadic job is placed in a waiting queue without the intervention of the scheduler.

→ The scheduler allows aperiodic jobs to execute only when the accepted sporadic job queue is empty. Again, whenever a server or job completes, the cyclic executive wakes up to execute.

<u>For example</u>:- The example illustrate the process.

Here $f = 4$, and shaded box represents the scheduled ~~spo~~ aperiodic task.

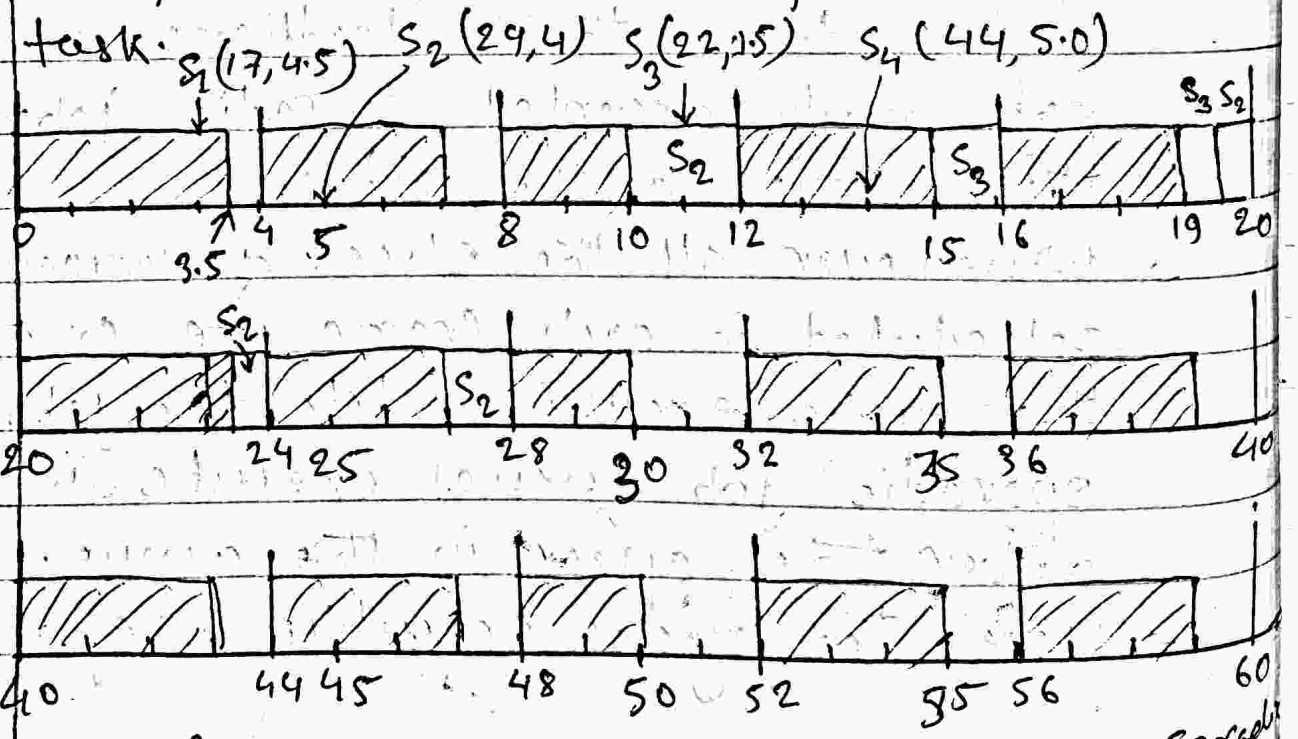


fig 5-11 Example of scheduling sporadic jobs

deadline
execution time

① Suppose that at time 3, a sporadic job S_1 (17, 4.5) is released. The acceptance test is done at time 4 for this job [i.e. the begining of frame 2.]. S_1 must be scheduled ~~on~~ in frames 2, 3 and 4. In these frames, the total amount of slack time is only 4 which is less than 4.5 [execution time of S_1]. So, the scheduler rejects this job.

② At time 5, S_2 (29, 4) is released. Frame 3 to 7 end before its deadline. During the acceptance test at 8, the scheduler finds that the total amount of slack in these frames is 5.5. Hence, it accepts S_2. The first part of S_2 with execution time 2 executes in the current frame.

③ At time 11, S_3 (22, 1.5) is released. At time 12, the scheduler finds 2 units of slack in frames 4, and 5, where S_3 can be scheduled. Moreover, there still is enough slack to complete S_2 even though S_3 executes ahead of S_2. Consequently, the scheduler accepts S_3. This job executes in frame 4 & 5.

primitive → primary.
slide

(4) Suppose that at time 14, S_4 (44, 5) is released. At time 16 when the acceptance test is done, the scheduler finds only 4.5 units of time available in frames before deadlines of S_4, after it has accounted for the slack time that has already been used by the remaining portions of S_2 and S_3 [which has earlier deadline than S_4]. Therefore it rejects S_4.
When remaining portion of S_3 completes in current frame, S_2 executes until the begining of next frame.

(5) The last portion of S_2 executes in frames 6 and 7.

Assignment 3

(1) Network flow graph - Book - P 122
Soft. Book → P 108

(2) Review of [EDF, LST] ~~Dynamic~~ Algorithm with example.

(3) Assumptions of priority - driven scheduling

(4) RM & DM algorithm with example

Independent Preemtable task scheduling.

algo (iterative network-flow)

INF algorithm :- INF algorithm is used to find out the feasible schedule for those tasks, in which some task may have relative deadlines ~~not follow the general~~ shorter than their periods. and it is constrained to have the general structure of cyclic schedule.

- A theorem says that, a system of independent, preemptable periodic tasks whose ~~release~~ relative deadlines are equal to or greater than their respective periods is schedulable if and only if the total utilization of the tasks is no greater than 1.

- This algorithm has key assumptions that tasks can be preempted at any time and are independent.

→ ~~After finding the feasible~~ The possible frame sizes may meet the frame size constraints condition ② & ③ but may not satisfy ① but INF ~~works~~ iteratively tries to find the feasible schedule and if neccessary it decomposes the task ~~that~~ into subtasks in order to make feasible schedule.

Note:- If this algorithm does not find a feasible schedule then there is no exist of feasible schedule even frame size constraints are met

after decomposing the task.

Network-flow graph :-

The INIF algorithm uses Network-flow graph during each iteration in order to find out the suitable frame.

- Assumptions of NFG

① There is a job vertex J_i representing each job J_i, $i = 1, 2, \ldots N$.

② There is a frame vertex named j representing each frame j in the major cycle, for $j = 1, 2, \ldots F$.

③ There are two special vertices named source & sink.

④ There is a directed edge (J_i, j) from a vertex J_i to a frame vertex j if the job J_i can be scheduled in the frame j, and the capacity of the edge is the frame size f.

⑤ There is a directed edge from the source vertex to every job vertex J_i and the capacity of this edge is the

execution time e_i of the job.

⑥ There is a directed edge from every frame vertex to the sink, and the capacity of this edge is f.

→ A flow of ~~the~~ an edge is a nonnegative number that satisfies the following constraints :-

(i) It is no greater than the capacity of ~~an~~ the edge and

(ii) With the exception of the source & sink, the sum of the flows of all the edges into every vertex is equal to the sum of the flows of all the edges out of the vertex.

→ A flow of a network-flow graph is the sum of ~~the~~ all the edges from the source; should be equal to the sum of the flows of all the edges into the sink.

→ The Time complexity of straight forward Network flow graph is $O((N+F)^3)$.

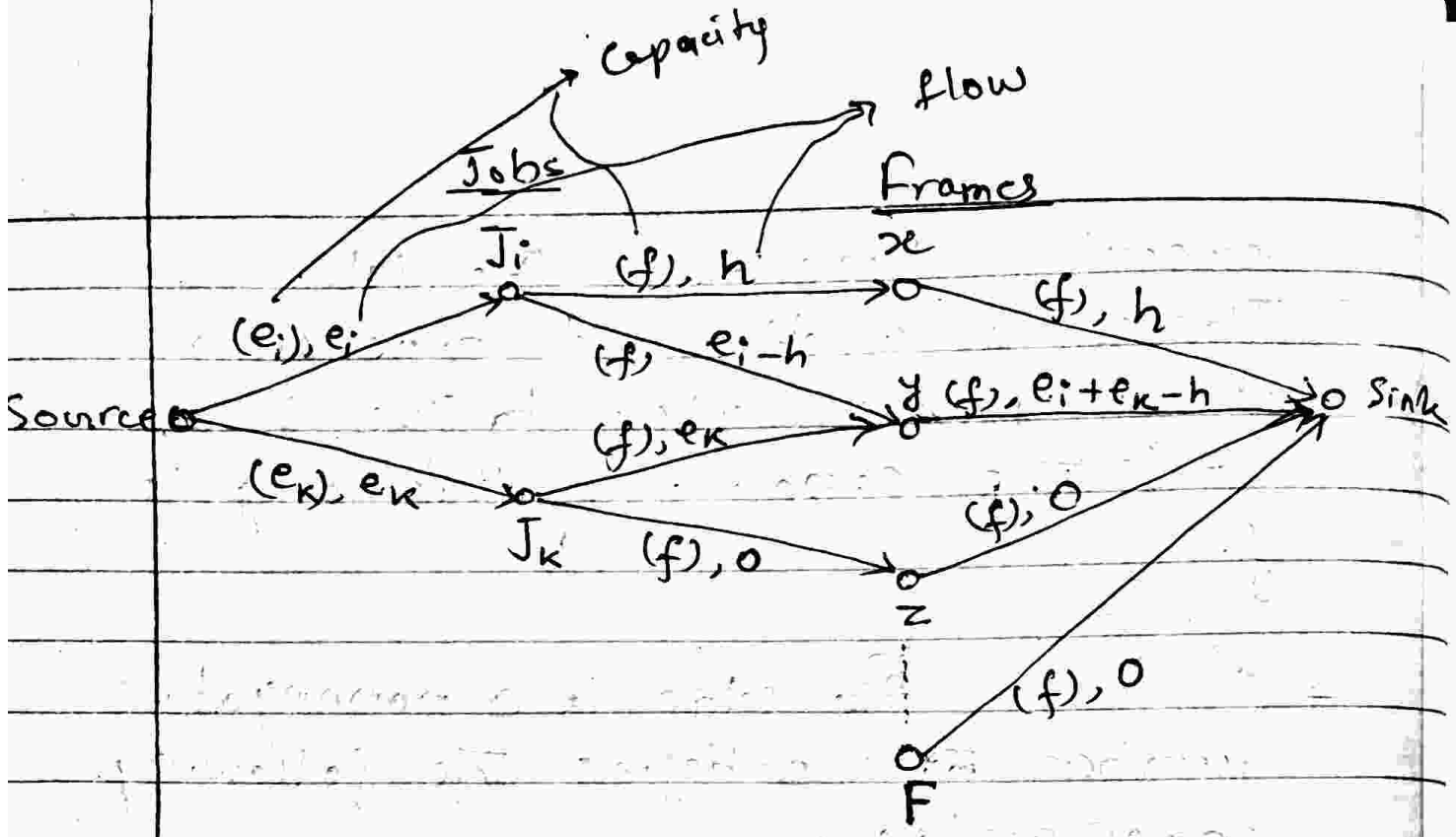


fig (5-4):- Part of a network-flow graph.

* <u>Maximum flow and feasible preemptive schedule</u>

- the maximum flow of a network-flow graph is at most equal to the sum of execution times of all the jobs to be scheduled in a major cycle.

for example:- $T_1 = (4, 1)$

$T_2 = (5, 2, 7)$

$T_3 = (20, 5)$.

The possible frame sizes are 4 & 2. The network-flow graph used in the first iteration is as in the figure below:

fig 5-15 page 124

- In the figure the frame size is 4. we want every job slice to be scheduled in a frame which begins ~~from the job~~ no sooner than its release time & ends no later than its deadline. [This is reason for the edges from the job vertices to the frame vertices].
- The maximum flow of this graph is 18, which is the total execution time of all jobs in a hyperperiod.
- Hence the flow of the edges from the job vertices to the frame vertices represent a schedule of the tasks.

5.5 Improving the average Response time of a aperiodic Jobs :-

Slack stealing :

Simply, we can improve the response times of aperoidic jobs by executing the aperiodic jobs ahead of the poroidic jobs whenever possible. This approach is called slack stealing was originally proposed for priority-driven systems.

Re-Exam mid-2014 2014-feb 17

class work:-

① Network-flow diagram

② Procs and cons of Clock-driven Scheduling.

- To make the ~~responsive~~ ~~AAO~~ system responsive, we need to minimize the average response of the aperiodic jobs. The following example illustrates this:

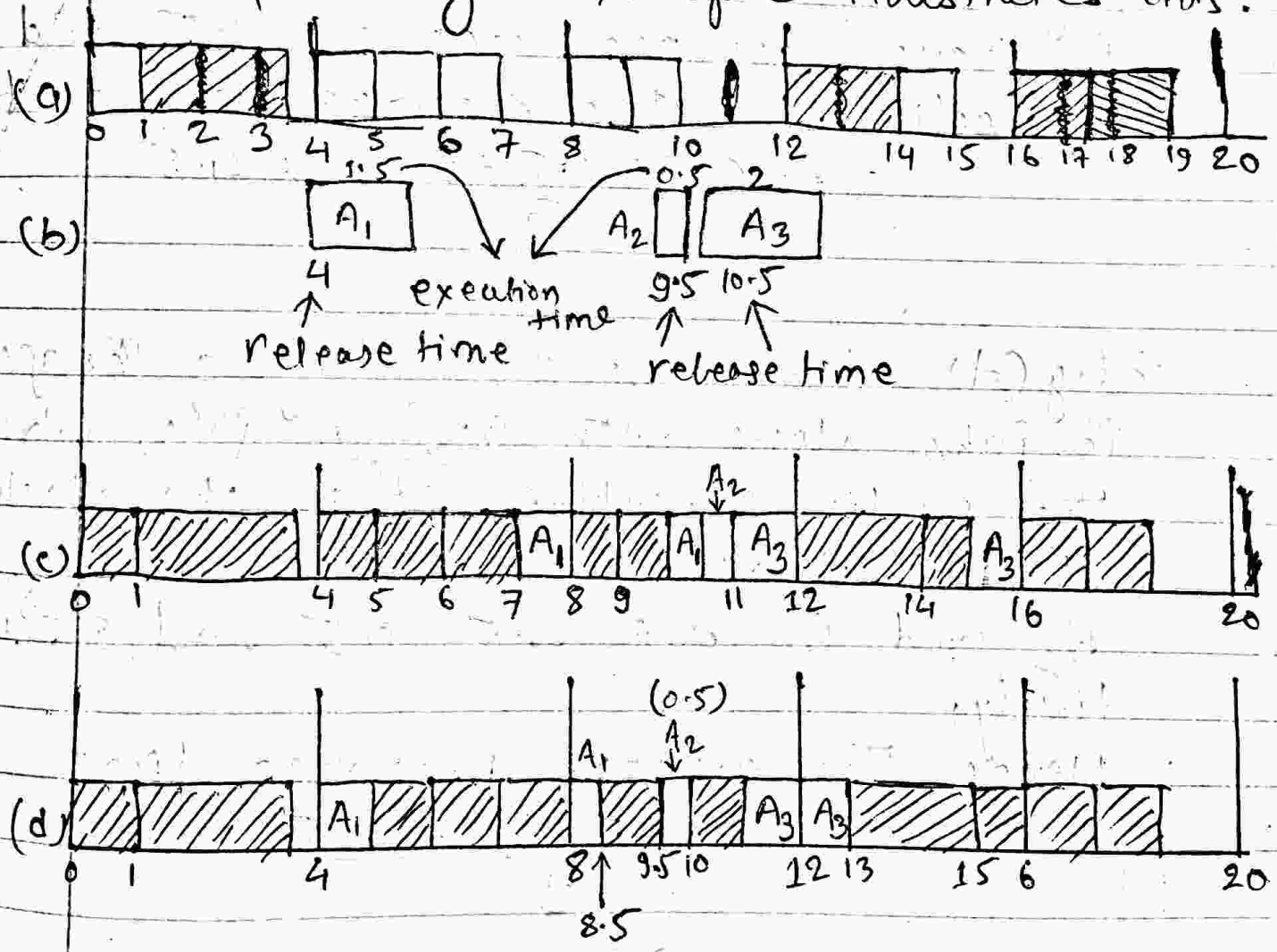


fig:- Example illustrating slack stealing.

(5-8)

Remaining Sol Back ⇒

Explanation :-

→ Fig (a) shows the first major cycle in the cyclic schedule of the periodic tasks.

→ Fig (b) shows three aperiodic job A_1, A_2 and A_3 with release time immediately before 4, 9.5, and 10.5 and their execution time 1.5, 0.5 and 2 respectively.

→ Fig (c), shows when the aperiodic jobs are executed in the slack periods of fig (a) by the cyclic executive. Here A_1, A_2 and A_3 have response times 6.5, 1.5 and 5.5 respectively. So the average response time

$$= \frac{6.5 + 1.5 + 5.5}{3} \Rightarrow 4.5$$

→ Fig (d) shows the scheduling of the aperiodic jobs along with periodic jobs, when the cyclic executive ~~steals~~ does the slack stealing. Here, the response time of A_1, A_2 and A_3 are 4.5, 0.5 and 2.5 respectively.

Hence the average response time

$$= \frac{4.5 + 0.5 + 2.5}{3}$$

$$= 2.5$$

Therefore, slack stealing reduces the average response time of the aperiodic jobs.

Sampling period.

The time between any two consecutive measurements of $y(t)$ and $r(t)$ is called sampling period.

when T is small it better approximate the analog behaviour but demands higher processor time.

when T is large it is difficult to recover the original signal and system will oscillate but utilize less processor time. To choose the sampling period the rise time is taken into consideration.

TS

The rise time is the amout of time that the plant takes to reach some small neighbourhood around the final state in response to a state change in the reference input.

If R is the rise time and T is the sampling period then the ratio are R/T should be between 10-20 sampling period within the rise-time.

Therefore, $10 \leq R/T \leq 20$.

The rise time is also used to measure the ~~representation~~ responsiveness of overall system. It can also be expressed in term

of Bandwidth. The bandwidth of the system is $\omega = \frac{1}{2R}$ Hz. Therefore, the sampling rate

which is the inverse of the sampling period is equal to 20 – 40 times the bandwidth. Theoritically, the lower limit of ~~signally~~ sampling rate is described by Nyquist sampling theorem. This theorem states that any time-continuous signal of bandwidth ω can be reproduced ~~faithfully~~ from its sampled values if and only if the sampling rate is 2ω or higher.

Assignment for exam not attending Students.

(1) Why is Digital control needed in Real time system?? ~~How~~ -

(2) How a digital controller designed in Sample data system.

(3) why There is need of multirate system ~~Describe with well to~~ in Real time System.

(4) How does nearest neighbouring algorithm ~~had~~ role in Data ~~acq~~ acquisition tracking?

✓ Differentiate between absolute and Relative Temporal consistency in Real-Time database.

+

exam Questions.