Deadline Monotonic Algorithm (DMA)

RMA no longer remains as an optimal algorithm for periodic real-time tasks when task deadlines are such that $d_i \neq p_i$.

For such cases we can employ DMA which is more efficient than RMA.

Basic Idea of DMA - *Assigns higher priorities to tasks based on their deadlines*, rather than assigning priorities based on task periods as done by RMA.

Thus, DMA assigns higher priorities to tasks with shorter deadlines!

Deadline Monotonic Algorithm (DMA)

Q: *What can you say when relative deadline of every task is proportional to its period ?*

- RMA and DMA would produce identical results

Q: *What happens when relative deadlines are arbitrary?*

- DMA is more proficient than RMA in the sense that it can sometimes generate feasible schedule when RMA fails.

Note that whenever DMA fails RMA always fail.

Deadline Monotonic Algorithm (DMA)

Exercise 4.8: Determine whether the task set given below is RMA and/or DMA schedulable.

$T_1 = (e_1=10, p_1=50, d_1=35);$
$T_2 = (e_2=15, p_2=100, d_2=20);$
$T_3 = (e_3=20, p_3=200, d_3=200);$

Quick note on Real-time Event Driven Schedulers

A *foreground-background scheduler (FBS)* is the simplest priority-driven preemptive scheduler.

Foreground (FG) tasks - RT periodic tasks
Background (BG) tasks - Sporadic, aperiodic and non-RT tasks

A BG task can run when none of the FG tasks are ready to run. BG tasks are treated as low priority tasks.

Let there be **n** FG tasks. Let T_B be the only BG task and let e_B be the execution time of BG task. Thus the completion time CT_B of BG task is:

$$CT_B = (e_B) / (1 – \Sigma_{i=1,n}(e_i / p_i))$$

Quick note on Real-time Event Driven Schedulers

The above expression means the following:

When any FG task is executing currently, the
BG task waits.

The average CPU utilization due to a FG task T_i is (e_i / p_i).
This means all FG tasks would consume $\Sigma_{i=1,n}(e_i / p_i)$.
Thus, the available time for BG tasks in every unit of time is given by $1 - \Sigma_{i=1,n}(e_i / p_i)$.

Precedence constraints – Static approach

- **List Scheduling Algorithm** – Many DAG algorithms use this idea

Some of the scheduling algorithms which consider the inter-task communication assume the availability of unlimited number of processors - UNC (unbounded number of clusters) scheduling algorithms.

Some other algorithms assume a limited number of processors - BNP (bounded number of processors) scheduling algorithms.

List scheduling – Static approach (cont'd)…

The basic idea of list scheduling is to make a scheduling list (a sequence of nodes for scheduling) by assigning them some *priorities*, and then repeatedly execute the following two steps until all the nodes in the graph are scheduled:

1) Remove the first node from the scheduling list(ready);

2) Allocate the node to a processor which allows the earliest start-time.

How to decide about the priorities?

List scheduling (cont'd)…

- There are various ways to determine the priorities of nodes such as HLF (Highest level First), LP (Longest Path), LPT (Longest Processing Time) and CP (Critical Path).

- In a traditional scheduling algorithm, the scheduling list is statically constructed before node allocation begins, and most importantly the sequencing in the list is not modified.

- In contrast, after each allocation, recent algorithms re-compute the priorities of all unscheduled nodes which are then used to rearrange the sequencing of the nodes in the list.

List scheduling (cont'd)…

Following three-step approach is used:

1) Determine new priorities of all unscheduled nodes;

2) Select the node with the highest priority for scheduling;

3) Allocate the node to the processor which allows the earliest start-time.

How do we re-compute the priorities?

Scheduling algorithms which employ this three-step approach can potentially generate better schedules.

List scheduling (cont'd)…

- Two frequently used attributes for assigning priority are: t-level (top level) and b-level (bottom level)
- What is a t-level approach?

The **t-level** of a node *n* is the length of a longest path (there can be more than one longest path) from an entry node to (excluding) that node n. Here, the length of a path is the *sum of all the node and edge weig*hts along the path.

As such, the t-level of n_i highly correlates with its earliest start-time, denoted by $T_s(n_i)$, which is determined after it is scheduled to a processor. This is because after it is scheduled, it is simply the length of the longest path reaching it.

List scheduling (cont'd)…

- What is b-level approach?

The **b-level** of a node n is the length of a longest path from n to an exit node. The b-level of a node is bounded from above by the length of a critical path.

A critical path (CP) of a DAG, which is an important structure in the DAG, is a longest path in the DAG.

Clearly, a DAG can have more than one CP.

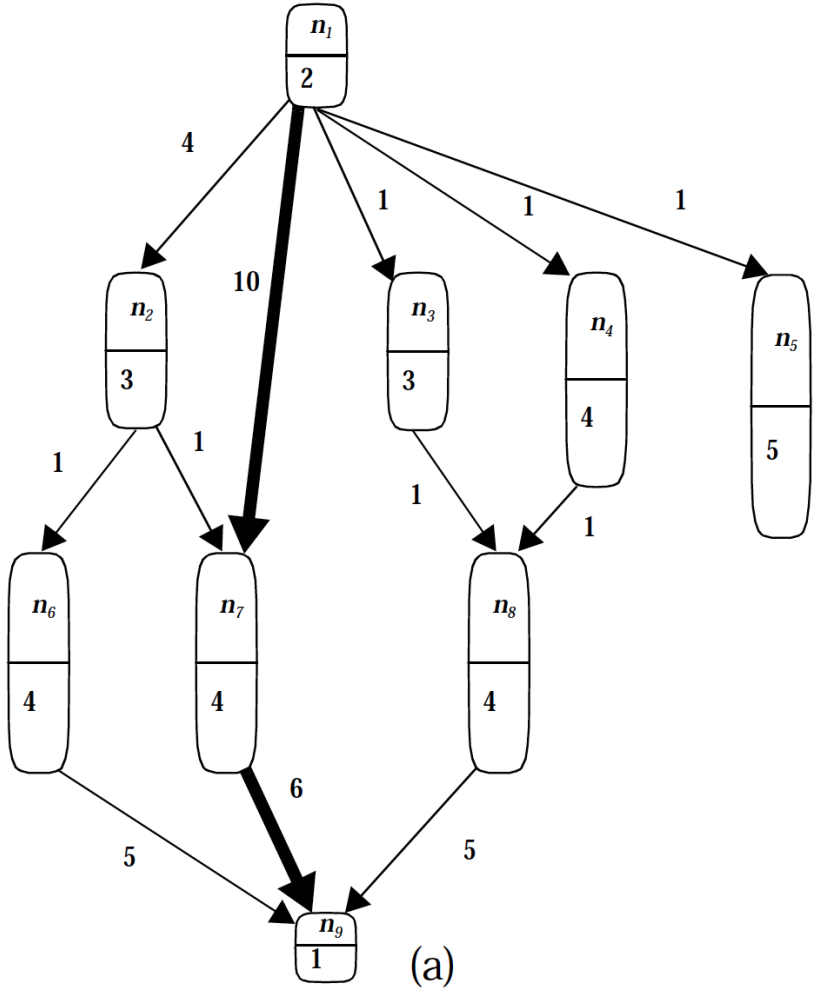
List scheduling (cont'd)…

Different algorithms use the t-level and b-level in different ways.

Some algorithms assign a higher priority to a node with a smaller t-level while some algorithms assign a higher priority to a node with a larger b-level. Still some algorithms assign a higher priority to a node with a larger (b-level – t-level).

In general, scheduling in a descending order of b-level tends to schedule critical path nodes first, while scheduling in an ascending order of t-level tends to schedule nodes in a topological order.

Example - List scheduling (cont'd)…



(a)

node	sl	t-level	b-level
$^*n_1$	11	0	23
n_2	8	6	15
n_3	8	3	14
n_4	9	3	15
n_5	5	3	5
n_6	5	10	10
$^*n_7$	5	12	11
n_8	5	8	10
$^*n_9$	1	22	1

(b)

Sl – Static level – path length from a node to exit node using only execution times along the longest execution path

Final remarks: Achieving predictability

- The operating system is the part most responsible for a predictable behavior.
- Concurrency control must be enforced by:
  - appropriate scheduling algorithms
  - appropriate synchronization protocols
  - efficient communication mechanisms
  - predictable interrupt handling

Chapter 6 – We will see some deadlock, synchronization protocols and interrupt Handling mechanisms