|  |
| --- |
| CSE 522 |
| ASSIGNMENT 2 |
| REPORT |

|  |
| --- |
| Nirmal Patel  3-7-2015 |

In this assignment, you are required to develop

1. An analysis program implementing various schedulability testing approaches for fixed priority schedule algorithms.

2. A comparative analysis of the schedulability of RM and DM algorithms, and fixed priority scheduling with least slack time using synthetic tasks sets

Part 1:

Analysis of various scheduling algorithms for task scheduling:

For fixed priority scheduling we can use three algorithms:

1. **Rate monotonic**
2. **Deadline monotonic**
3. **Least slack time scheduling**

**Rate monotonic:**

Its static-priority scheduling. The static priorities are assigned on the basis of the cycle duration of the job: the shorter the cycle duration is, the higher is the job's priority.

Some assumptions are as follow

* No resource sharing (processes do not share resources, *e.g.* a resource, a queue, or any kind of semaphore blocking or non-blocking or busy wait)
* Deterministic deadlines are exactly equal to periods
* Static priorities (the task with the highest static priority that is runnable immediately preempts all other tasks)
* Static priorities assigned according to the *rate monotonic* conventions (tasks with shorter periods/deadlines are given higher priorities)
* Context switch times and other thread operations are free and have no impact on the model

**RMA is the optimal fixed-priority algorithm as long as Deadline <= period.** If a task set cannot be scheduled using the RMA algorithm, it cannot be scheduled using any fixed-priority algorithm.

Implementation in assignment:

1. Obtain data from input file
2. Store in to structure containing wcet, period, deadline, priority
3. Sort structure elements in increasing order of period.
4. Run utilization test
5. If it fails to get schedulable task call for RT test from utilization test function itself.
6. Get output whether it all tasks are schedulable or not.

**Deadline monotonic:**

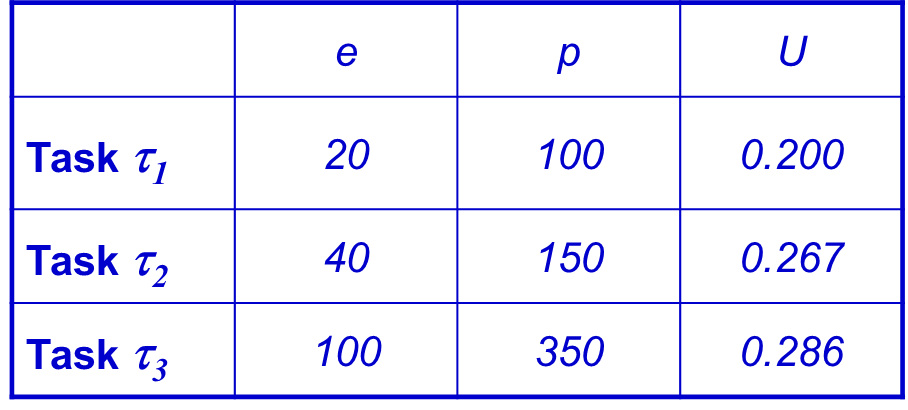
**Deadline-monotonic priority assignment** is a priority assignment policy used with fixed priority preemptive scheduling.

This priority assignment policy is optimal for a set of periodic or sporadic tasks which comply with the following restrictive system model:

1. All tasks have deadlines less than or equal to their minimum inter-arrival times (or periods).
2. All tasks have (WCET) that are less than or equal to their deadlines.
3. Scheduling overheads (switching from one task to another) are zero.
4. All tasks have zero release jitter (the time from the task arriving to it becoming ready to execute).

Deadline monotonic priority assignment is **not** optimal for fixed priority non-pre-emptive scheduling.

Example for part 1:



Total execution time = 100 + 40 + 20

Sufficient condition for UT test = 

For the given example ut test passes for all cases.

If we change value of execution time for task t1 from 20 to 40. Our sufficient condition will not meet and we have to do RT test in order to know whether the whole task set is schedulable or not.

RT test method:

* Summation of execution time
* Sum(Ej/ Pj) + Ei/min(Di, Pi) + Sum(Ek)/min(Di, Pi)
* Your function over runs dedline which means RT test failed
* If your response time < period or response time(for n) = response time(for n-1) then RT test is passed.

Quick performance comparison of all scheduling algorithms

1. RM- period based; DM- deadline based; least slack Time: difference of deadline – wcet.
2. Performance expectation: LST> DM> RM

As LST uses scheduling limit as (deadline - wcet), it will get more time to schedule for the given period compared to RM and DM .

1. Also performance of RM and DM is same if Deadline = Period
2. DM is better than RM in case of Deadline < period because of the availability of more time

Part 2:

Synthetic test case generation:

* Use unifast function to compute number of utilization within given utilization limit.
* Decide number of tasks i.e. 12, 24, 48
* Period is computed with random number generation within range of 10-100, 100-1000, 1000-10000.
* Deadline should be random number between [C, T] and for second case it should be

[C + (T-C)/2, T].

This is the output I got from running 1000 task with .05 utilization and task 12. I couldn’t verify the output for 10000 tasks with different conditions.