Dynamic Priority Scheduling

Real-Time Operative Systems Course

Paulo Pedreiras, DETI/UA/IT

October 24, 2022

Last lecture

Fixed-priority online scheduling

- Rate-Monotonic scheduling
- Deadline-Monotonic and arbitrary priorities
- Analysis:
  - The CPU utilization bound
  - Worst-Case Response-Time analysis

Agenda for today

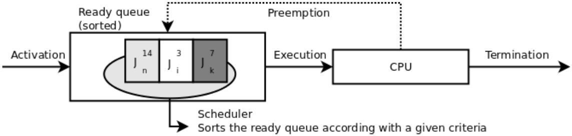
- Online scheduling with dynamic priorities
  - Earliest Deadline First scheduling
  - Analysis: CPU utilization bound and CPU Load Analysis
- Optimality and comparison with RM
  - Schedulability level, number of preemptions, jitter and response time
- Other dynamic priority criteria
  - Least Slack First, First Come First Served

On-line scheduling with dynamic priorities

Dynamic priorities

Ready Queue sorted by instantaneous priorities and dynamically re-sorted (if/when necessary)

- Scheduling is based in dynamic criteria, i.e. one that is known only at run-time
- The dynamic parameter used to sort the ready tasks can be understood as a dynamic priority
- The ready queue is (re)sorted according with decreasing priorities whenever there is a priority change. Executes first the task that has the greater instantaneous priority
  - Complexity O(n.log(n))



On-line scheduling with dynamic priorities

Pros

- Scales well (wrt SCS)
    - Changes made to the task set are immediately seen by the scheduler
- Accommodates easily sporadic tasks (wrt SCS)
- Allows higher utilization levels than Fixed Priorities

Cons

- More complex implementation (wrt SCS and FP)
- Higher overhead (wrt SCS and FP)
    - Re-sorting of ready queue; depends on the algorithm
- "Unpredictability" on overloads (wrt FP)
    - It is not possible to know a priory which tasks will fail deadlines

On-line scheduling with dynamic priorities

Priority allocation approaches

- Inversely proportional to the time to the deadline
  - EDF – Earliest Deadline First
    - Optimal among all dynamic priority criteria
- Inversely proportional to the laxity or slack
  - LSF (LST or LLF) – Least Slack First
    - Optimal among all dynamic priority criteria
- Inversely proportional to the service waiting time
  - FCFS –First Come First Served
    - Not optimal with respect to meet deadlines; extremely poor real-time performance
- etc.

On-line scheduling with dynamic priorities

Schedulability tests

- Since the schedule is built online it is important to determine a priori if a given task set meets or not its temporal requirements
- There are three types of schedulability tests:
    - Based on CPU utilization
    - Based on CPU load (processor demand)
    - Based on response time

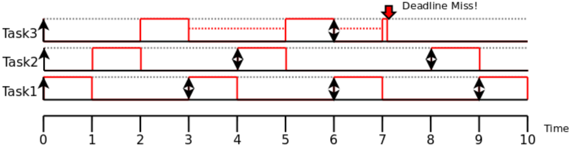
EDF Scheduling

EDF tests based on CPU utilization (n independent tasks, with full preemption)

- $D = T$
  - $U(n) = \sum_{i=1}^{n} (\frac{C_i}{T_i}) \leq 1 \Leftrightarrow$ Task set is schedulable
  - Allows using 100% of CPU with timeliness guarantees
- $D < T$
  - $U(n) = \sum_{i=1}^{n} (\frac{C_i}{D_i}) \leq 1 \Rightarrow$ Task set is schedulable
  - Sufficient condition, only.
  - Pessimistic test (as for RM, inflates the utilization)

RM Scheduling - example

τ	C	T
1	1	3
2	1	4
3	2.1	6

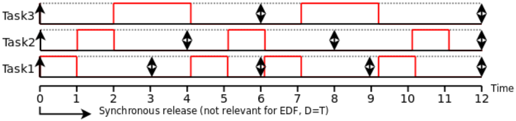


- $U = \frac{1}{3} + \frac{1}{4} + \frac{2.1}{6} = 0.93 > 0.78 \Rightarrow$ 1 activation per period NOT guaranteed.
- In fact τ_3 fails a deadline!

EDF Scheduling – same example

Same example as before!

τ	C	T
1	1	3
2	1	4
3	2.1	6

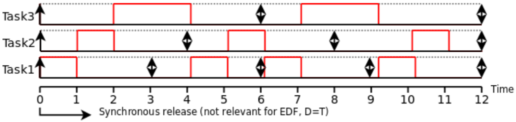


- $U = \frac{1}{3} + \frac{1}{4} + \frac{2.1}{6} = 0.93 \leq 1 \Leftrightarrow$ one activation per period IS guaranteed.

EDF Scheduling – same example

Same example as before!

τ	C	T
1	1	3
2	1	4
3	2.1	6



Note:

- No deadline misses
- Less preemptions
- Higher jitter on quicker tasks
- The worst-case response time does not coincide necessarily with the synchronous release

EDF Scheduling

Notion of **fairness**

- Be fair on the attribution of resources (e.g. CPU)
- EDF is intrinsically fairer than RM, in the sense that tasks see its relative deadline increased as the absolute deadline approaches, independently of its period or any other static parameter.
- Consequences:
  - Deadlines are easier to met
  - As the deadline approaches preemptions suffered by a given task are reduced
  - The slack of tasks that are quick but have large deadlines can be used by other task (higher jitter on tasks with shorter periods)

CPU Load Analysis

- For $D < T$, the biggest period during which the CPU is permanently used (i.e. without interruption, idle time) corresponds to the scenario in which all tasks are activated synchronously. This period is called **synchronous busy period** and has duration L
- L can be computed by the following iterative method, which returns the first instant since the synchronous activation in which the CPU completes all the submitted jobs

Computation of L

$$L(0) = \sum_i (C_i)$$

$$L(m+1) = \sum_i (\lceil \frac{L(m)}{T_i} \rceil \cdot C_i)$$

Stop condition: $L(m+1) = L(m)$

CPU Load Analysis

- Knowing L, we have to guarantee the **Load Condition** , i.e.

$$\boxed{h(t) \leq t, \forall t \in [0, L[\Rightarrow \textit{ Task set is schedulable}}$$

- The Load Condition refers to all the work that must be **completed by t** .
- How to compute h(t)?
- Example:
  - Task set $\Gamma = \{(2,4),(2,8),(3,16)\}$
- Draw the Gantt chart and mark the points where tasks must complete
- Write a suitable equation

CPU Load Analysis

Knowing L we have to guarantee the load condition, i.e.

$$h(t) \leq t, \forall t \in [0, L[$$

And $h(t)$ can be computed as follows:

$$\boxed{h(t) = \sum_{i=1..N} max(0, 1 + \lfloor \frac{t - D_i}{T_i} \rfloor) \cdot C_i}$$

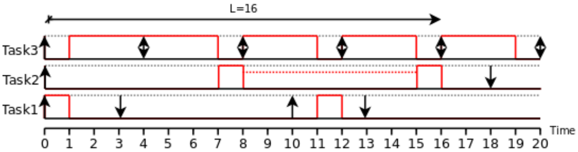
- The computation of h(t) for all values of $t \in [0, L[$ is unfeasible.
  - However it is enough computing the load condition for the instants in which the load function varies, i.e.

$$\boxed{S = \bigcup_i (S_i), S_i = \{m \cdot T_i + D_i\}, m = 0, 1, \ldots}$$

- Note: there are other, possibly shorter, values for L

EDF Scheduling

τ	C	D	T
1	1	3	10
2	2	18	20
3	3	4	4



- $\sum_{i=1..N} \frac{C_i}{D_i} = \frac{1}{3} + \frac{2}{18} + \frac{3}{4} = 1.194 > 1 \Rightarrow$ Schedulability not guaranteed
- But the CPU load analysis indicates that the task set is schedulable!

LSF Scheduling

Least Slack First : Executes first the task with smaller slack $\left(L_i(t) = d_i - c_i(t)\right)$

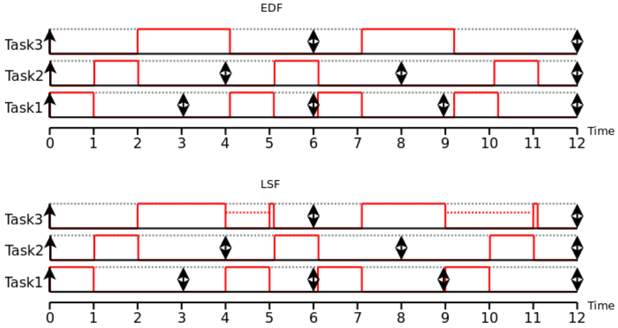
LSF vs EDF short comparison

- LS is optimal (as EDF)
- As the slack $\uparrow$ the priority $\downarrow$
- Priority of ready tasks increases as time goes by
  - Rescheduling only on instants where there are activations or terminations. **Why?**
- Priority of the task in the running state does not change
  - In EDF the priorities of all tasks (ready and executing) increase equally as time goes by
- Causes an higher number of preemptions than EDF (and thus higher overhead)
- No significant advantages with respect to EDF!

LSF scheduling example

LSF vs EDF

τ	C	T
1	1	3
2	1	4
3	2.1	6



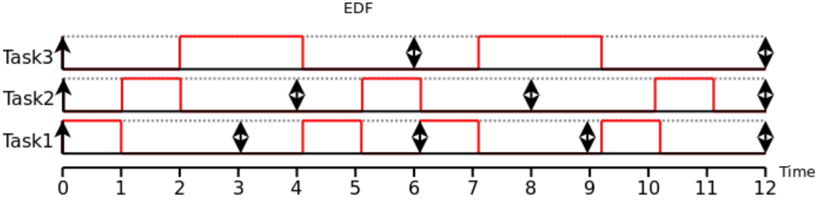
FCFS Scheduling

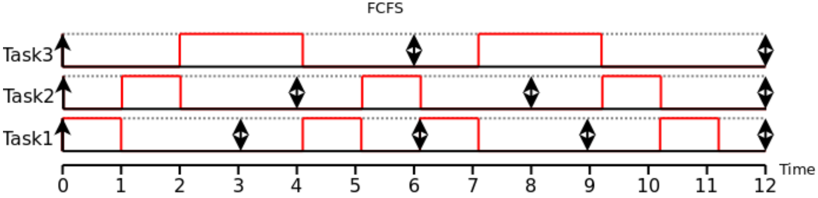
Execute tasks as they arrive. Priority depends on the arrival order.

A brief comparison between FCFS and EDF/LLF

- Non optimal
  - May lead to deadline misses even with very low CPU utilization rates
- Job age ↑ Priority ↑
- Priority of the ready and running tasks increases as time goes by (an in EDF)
- New jobs always get the lower priority
- There are no preeemptions (smaller overhead and facilitates the implementation)
- Very poor temporal behavior!

FCFS – same example





- When the "age" is the same the tie break criteria is decisive!

Summary

- On-line scheduling with dynamic priorities
- The EDF - Earliest Deadline First criteria: CPU utilization bound and CPU Load Analysis
- Optimality of EDF and comparison with RM:
  - Schedulability level, number of preemptions, jitter and response time
- Other dynamic priority criteria:
  - LLF (LST) - Least Laxity (Slack) First
  - FCFS - First Come First Served