Real-Time Systems

Event-Driven Scheduling

Hermann Härtig
Outline

• mostly following Jane Liu, Real-Time Systems
• Principles
• Scheduling
• EDF and LST as dynamic scheduling methods
• Fixed Priority schedulers
• Admission based on utilization
• A few multi-processor insights (more later)
• Anomalies
Important Properties

• scheduling decisions are triggered by events (not time instants)
• events are release, completion, blocking, unblocking of jobs
• event triggers: scheduler calls, interrupts, timers, …
• scheduling decisions are made online
• scheduling must therefore be simple
• admission is online or offline
• work-conserving schedulers never leave a resource idle intentionally
some restrictive assumptions of time-driven systems are relaxed:

• fixed inter-release times
  → minimum inter-release times

• fixed number of real-time tasks
  → number of real-time and non real-time tasks can vary

• a priori fairly well known parameters
  → overload, schedule non-RT in the background, …
Principles

At Admission Time

• select scheduler (may depend on the OS)
• check if feasible schedule exists for the selected scheduler
• assign jobs a value as a simple selection criterion: priorities

Scheduling / Dispatching

• at event, select highest prioritized job
Comparing Schedulers

How good are schedulers?

• shorter response times
• more task sets
• higher utilization of resources

Optimality of Schedulers

• A scheduling method X is called *optimal in a class of scheduling methods*, if X produces a feasible schedule whenever there exists a scheduling method Y in this class that produces a feasible schedule.

• X is called *optimal*, if X produces a feasible schedule whenever there exists such a schedule (no matter which method produced it).
Earliest Deadline First

Assign priorities at time when jobs are released: “the earlier the deadline the higher the priority”

Optimality

• one processor,
• jobs are preemptable,
• jobs do not contend for passive resources,
• jobs have arbitrary release times, deadlines,
• then: EDF is optimal (i.e. if there is a feasible schedule, there is also one with EDF)
EDF Example

**T1:** (2,0.9)  **T2:** (5,2.3)
EDF Optimality

Proof (informal)

- assume a feasible, non EDF schedule
- systematically transform it to an EDF schedule (3 steps)

1. 

2. 

3. 

[Least/Minimum] [Slack Time/Laxity] First

Slack Time / Laxity

- time to deadline - remaining execution time required to reach deadline
  \[ d - x - t \]
  - \( d \) absolute deadline
  - \( x \) remaining execution time of a job
  - \( t \) current time

Scheduling Based on Slack

- priority dynamic per job (see example)
- strict version is optimal
Least Slack Time First

scheduler checks slacks of all ready jobs and runs the job with the least slack

Two Versions:

- **Strict**: slacks are computed at all times
  - each instruction (prohibitively slow)
  - each timer tick
- **Non-strict**: slacks are computed only at events (release, completion)
Example: Non-strict LST

Job: (release time, execution time, deadline)

- $J_1 : (0, 2, 3.5)$
- $J_2 : (1, 3, 5)$

$t = 0$: $J_1$ released and scheduled
$t = 1$: $J_2$ released; 
$L(J_1) = 3.5 - 1 - 1 = 1.5$; 
$L(J_2) = 5 - 3 - 1 = 1 \rightarrow J_2$ scheduled
$t = 3.5$: $J_1$ deadline miss

EDF schedules both jobs successfully!
Example: Strict LST

Job: (release time, execution time, deadline)

\[ J_1 : (0, 2, 3.5) \quad \text{and} \quad J_2 : (1, 3, 5) \]

\[ t = 0: \quad J_1 \text{ released and scheduled} \]

\[ t = 1: \quad J_2 \text{ released; } \]
\[ L(J_1) = 3.5 - 1 - 1 = 1.5; \quad L(J_2) = 5 - 3 - 1 = 1 \rightarrow J_2 \text{ scheduled} \]

\[ t = 1.5: \quad L(J_1) = 3.5 - 1 - 1.5 = 1; \quad L(J_2) = 5 - 2.5 - 1.5 = 1 \rightarrow \]
\[ J_1, J_2 \text{ are scheduled and executed in parallel (at half speed)} \]

\[ t = 3.5: \quad J_1 \text{ completes } \rightarrow J_2 \text{ continued at full speed} \]

\[ t = 5: \quad J_2 \text{ completes} \]
Latest Release Time (LRT)

Rationale

- no need to complete real-time jobs before deadline
- use time for other activities

Idea

- backwards scheduling
  (Deadline <-> Release, turn around precedence graph, EDF)
- run as late as possible
- use latest possible release times
- optimal (analog EDF and strict LST)
EDF and Non - Preemptivity

Job: (release time, execution time, deadline)

EDF is not optimal if jobs are not preemptable
EDF and Multiple Processors

Job: (release time, execution time, deadline)

- $J_1: (0,1,2)$
- $J_2: (0,1,2)$
- $J_3: (0,5,5)$

$J_3$ missed deadline

- EDF is not optimal for multiprocessor systems
Assumptions for Next Algorithms

Set of **periodic tasks** with these properties:

- tasks are independent
- one processor
- no aperiodic tasks
- preemptable, context switch overhead is negligibly small
- period = minimum inter-release time
  (release times are not fixed but at least one period apart)

Since tasks are independent, tasks can be added (if admitted) and deleted at any time without causing deadline misses.
Fixed Priority Scheduling

- the shorter the period the higher the priority
  (rate: inverse of period)
- example: (e, p); d = p
Deadline Monotonic Scheduling

Fixed Priority Scheduling

• the shorter the relative deadline the higher the priority
• example: (e, d, p)

Conclusion (no proof):
RM not optimal but DM if d ≤ p for all tasks
T: periodic tasks, independent, preemptable, one CPU

**Deadline Monotonic Scheduling (DMS)**

- relative deadlines \( \leq \) periods, in phase
- if there is any feasible fixed priority schedule for T, then Deadline Monotonic is feasible as well

**Rate Monotonic Scheduling (RMS)**

- relative deadlines = periods
- if there is any feasible fixed priority schedule for T, then Rate Monotonic produces a feasible schedule as well
Admission Based on Utilization

Utilization

- a task \((p,e)\) requires \(e/p\) of the capacity of a processor
- any scheduler can admit at most up to full capacity:
  
  For a task set \(T_1 \ldots T_n: \sum e_i/p_i \leq m\) is a necessary but not sufficient condition for \(m\) processors

Scheduled Utilization / Utilization Bound

- can we establish a maximum utilization bound \(X\) such that:
  
  \(T_1 \ldots T_n: \sum e_i/p_i \leq X\) is sufficient?

- depends on the scheduling algorithm
- the higher the better
Utilization: RMS / EDF

T₁: (2, 1)  T₂: (5, 2.5)  U=1

EDF

RMS

T₂ misses deadline

RMS not optimal in general
Some Schedulable Utilization (SU) Results

- independent tasks, preemptable, relative deadline = period, m = 1 processor
- n: number of tasks
- EDF: SU = 1
- RMS: SU = n \( (2^{1/n} - 1) \)  \( n \to \infty \): \( \ln(2) \)
- RMS with harmonic periods: SU = 1
- harmonic periods (also called simply periodic): for all pairs of tasks Ti, Tj: if \( p_i \leq p_j \) then \( p_j = n_{ij} \times p_i \)
for task sets with $d_i \leq p_i$ (+ some other cases)

**Critical Instant Analysis / Time Demand Analysis**

- critical instant for task $T_i$:
  release of jobs such that they have the maximum response time
- 1 CPU, preemptable, independent:
  critical instant occurs when all tasks are released simultaneously
- it is sufficient to check schedulability for the simultaneous release for the longest involved period
Fixed Priority Schedulability and Blocking

• $T_i$ may have to wait for non-preemptable, lower priority task
• $b_i$: longest non-preemptable portion of all lower priority jobs
• schedulable utilization with $i$ tasks:
  $$U_i = \frac{e_1}{p_1} + \frac{e_2}{p_2} + \ldots + \frac{e_i}{p_i}$$
  $$U_i + \frac{b_i}{p_i} \leq SU(i)$$
Non Negligible Context Switch Time

• For Job level fixed priority schedulers:
  i.e. each job preempts at most one other job
• 2 context switches:
  • release (when it preempts other)
  • completion
• include context switch overhead in WCET:
  \[ WCET_i' := WCET_i + 2 \times \text{context switches} \]
Static and Dynamic Priority

If no new tasks arrive:

• Task static: task T does not change its priority, i.e. all jobs of T have same fixed priority

• Job static: jobs do not change their priorities

• Job dynamic: jobs change their priorities

Careful: job static is a dynamic priority system
Earliest Deadline First, priority assignment:

fixed per job, dynamic at task level: the closer the absolute deadline of a job at release time, the higher the priority

\[ T_1: (0.9, 2) \]
\[ T_2: (2.3, 5) \]
EDF and Overload, Examples

\[ T_1: (1, 2) \quad T_2: (3, 5) \]

U=1.1

\[ T_1 \text{ misses} \]

\[ T_1: (0.8, 2) \quad T_2: (3.5, 5) \]

U=1.1

\[ T_2 \quad T_1 \text{ and } T_2 \text{ misses} \]

No easy way to determine which jobs miss deadline
EDF and Overload, Another Example

\[ T_1: (0.8, 2) \quad T_2: (4.0, 5) \quad U=1.2 \]

In fixed priority systems it is possible to predict which tasks are affected by overruns.
Predictable/Sustainable Execution

Informal Definition

• Given a set of periodic tasks with known minimal and maximal execution times and a scheduling algorithm.

• A schedule produced by the scheduler when the execution time of each job has its maximum (minimum) value is called a maximum (minimum) schedule.

• An execution is called predictable, if for each actual schedule the start and completion times for each job are bound by these times in the minimum and maximal schedules.

• The execution of every job in a set of independent, preemptable jobs with fixed release times is predictable when scheduled in a priority driven manner on one processor.
Lessons Learned

- schedulers: static and dynamic priorities (RMS, EDF, LST)
- schedulability analysis: utilization, critical instant
- RMS and EDF are optimal under simplistic assumptions