Real time systems

Fabien Calcado, Thomas Megel

Email: fabien.calcado@gmail.com
thomas.megel@fr.thalesgroup.com

EFREI 2015 - 2016

Outlines

- **Monoprocessor scheduling**
  – Definitions and scheduling strategies for some task models
- **Multiprocessor scheduling**
  – From a graph of task dependency

Scheduling

- **Introduction**
  – The application is a set of n tasks that we call a task system
    - Simultaneous start (same first release date) or spread over the time
  – Terminology (reminder)
    - The **scheduling** is the organisation of task execution on the CPU(s)
      » *Sequencing, interleaving…*
    - The **scheduling policy** is the organization rule to execute tasks on the CPU(s)
      » *Citations from… ? :*
        » "All we have to decide is what to do with the time that is given to us."
        » "… is never late, nor is he early, he arrives precisely when he means to"

Scheduling

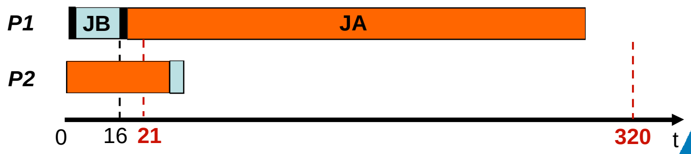
- **Introduction**
  – The application is a set of n tasks that we call a task system
    - Simultaneous start (same first release date) or spread over the time
  – Terminology (reminder)
    - The **scheduling** is the organisation of task execution on the CPU(s)
      » *Sequencing, interleaving…*
    - The **scheduling policy** is the organization rule to execute tasks on the CPU(s)
      » *Citations from… Gandalf :*
        » "All we have to decide is what to do with the time that is given to us."
        » "… is never late, nor is he early, he arrives precisely when he means to"

Scheduling

Introduction

- 2 jobs to execute
  - Job A : Duration = 270 , Deadline = 320
  - Job B : Duration = 15 , Deadline = 21
- Two different resources
  - P1 : speed = 1 , switching duration = 1 , priority = to deadline
  - P2 : speed = 10 , switching duration = 0 , priority = to the first

JA arrives before JB at t = 0



Scheduling

Execution duration (denoted by C)

- Time necessary for a processor to execute the code of a job without any interrupt
  - The execution duration depends on processor speed
  - The execution duration est theretical (non constant in practice)
    - » *worst case execution time (**WCET**)*
    - » *best case execution time*
  - ➔ In general, job execution time corresponds to WCET
- Methods to evaluate the execution duration
  - On-line measure, off-line analysis
- Difficulties
  - Complexity and range of execution paths
  - Processor complexity

Scheduling

Temporal properties of a job

- Earliest start time (request / release) : Tr
  - Time when the job is ready (arrival time ou release time)
- Latest end time (**deadline**) : Td
  - Time when the job shall be finished
- Execution duration of a job « i » (requirement) : Ci

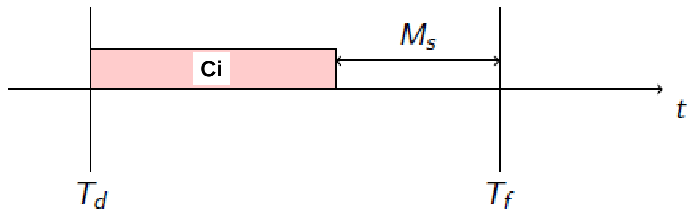
Derived parameters

- Critical delay (temporal windows)
  - Maximum acceptable delay to execute a task
- Latest start time
- Duration until the latest start time (laxity)
- Earliest end time
- Start time and end time of a job
  - Possible preemptions

Scheduling

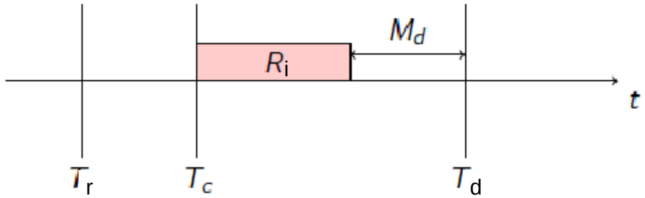
Non-preemptive case

- Static margin : Ms = (Td - Tr) - Ci
  - Ms ≥ 0
  - If Ms = 0, no choice



Scheduling

- **Preemptive case**
  - Dynamic margin or laxity : Md = (Td – Tc) – Ri
    - Tc = current time
    - Ri = remaining execution time for job i
    - Md remains the same for an active job
    - Md decrease dynamically for inactive jobs



Scheduling

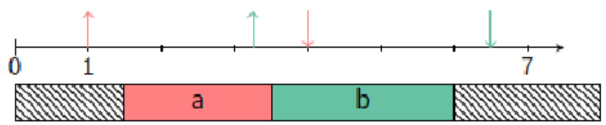
- **Task notion**
  - A task is an entity that releases jobs
    - The task is released and shall execute a job
  - Three main classes
    - Periodic : release a job at each period T (> 0)
      - » *Implicit deadline: Deadline = period*
      - » *Constrained deadline : Deadline ≤ period*
    - Sporadic : job releases separated with minimum duration (defined)
    - Aperiodic : most general case, job releases shall be known

Scheduling

- **Interval of analysis (periodic case)**
  - The execution lasts indefinitely but the configuration behavior is periodic
  - **Interval of analysis** is [0 , LCM (Pi)]
    - Pi = period of any tasks
    - Only valid for periodic case with simultaneous releases

Scheduling

- **Schedule**
  - It is a method to forsee the allocation of resources (conception step)
  - A schedule is **optimal** if any temporal constraints are met



Scheduling

- **Schedule**
  - There is **overload** when the amount of task to execute is such that any schedules lead to miss at least one constraint for one task
    - There is no optimal schedule
  - A scheduling **algorithm** is optimal, given a class of problems, when it generates optimal schedules

Scheduling

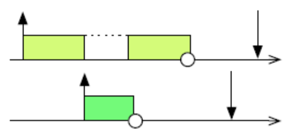
- **Types of scheduling algorithms**
  - Static
    - Schedule is decided before execution (off-line)
      - » *Scheduling sequence is pre-processed based on temporal characteristics of the tasks*
  - Dynamic
    - Schedule changes during the execution (on-line)
      - » *Scheduling choices are taken over the application execution by the scheduler*
    - Non preemptive or preemptive
    - Fixed priority (static ou dynamic)

Scheduling

- **Off-line: advantages / drawbacks**
  - It requires the knowledge of the system and its temporal characteristics (fixed release dates)
    - Lack of flexibility but strong predictability (unique instruction flow)
  - Adapted to periodic model
    - Schedule processed on one cyle (lcm of task periods): cyclic scheduler (loop programming)
  - Simplicity of the scheduler
    - Low execution overheads
  - Processing efficiency not requested for the generation of the schedule
    - Optimal algorihms implementation + extra constraints can be taken into account (task precedence, task synchronization, arbitrary release dates, etc.)
  - Execution regularity
    - Inflexible (cannot adapt to the environment)

Scheduling

- **On-line: advantages / drawbacks**
  - Flexible
    - Adapted to dynamic and evolving systems (able to make decision at time t)
  - Processing efficiency required
    - Simple scheduling policies
  - Difficulty to take into account various constraints
    - Tricky analysis and often pessimistic
    - A priori less safe ➔ need proofs
  - Non conservative
    - The processor is always used is a task is ready



Off-line example: loop programming

- T1 : C=10 , Period=30
- T2 : C=8 , Period=30
- T3 : C=6 , Period=60
- T4 : C=10 , Period=60
- T5 : C=4 , Period=120

Models – assumptions

- On tasks
  - Types : sequential, parallel
  - Relations : mutually dependent, independent
  - Values: identical / different, constant / dependent on end time
  - Abort: if misses on mandatory constraints, authorized, forbidden
- On resources
  - Multiplicity : one or several (equivalent or not)
  - Access mode: centralized ou distributed (memory resources)
  - Requisition (preemption)
    - » *Always possible (with or without loss)*
    - » *Possible at times*
    - » *Impossible*

Models – assumptions

- On event laws
  - Totally or partially known (non real time)
  - Predetermined or statistical
- On release law of tasks
  - Release times of tasks
  - Access times to resources
  - Possible /impossible request intervals of resources

Models – assumptions

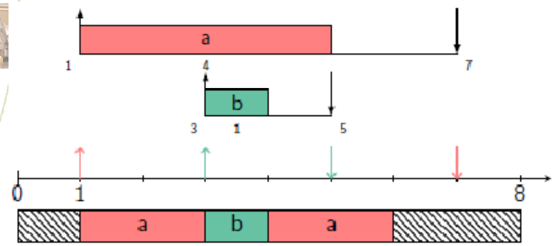
- Problem statement = a choice among all assumptions
  - Exhibition of several classes of problems with known solutions, often optimal ones (hypothesis / constraints context)
  - Following by:
    - » *H1 = hypothesis from problem statement: to identify the class of problem*
    - » *H2 = condition on quantitative data of the problem*
      - ➔ Hypothesis of feasibility to verify

Scheduling algorithms

- **Well-known algorithms**
  - Fundamental algorithms
    - Most of the time, the other ones are a mix between them
  - Static
    - FP
    - RMS
  - Dynamic
    - FCFS (First Come Fisrt Serve)
    - RR (Round Robin)
    - EDF (Earliest Deadlines First)
    - LLF (Least Laxity First)

Static scheduling algorithms

- **Preemptive fixed priority scheduling (FP)**
  - Tasks have a fixed priority
  - Priority for the task having the highest one
    - Should be able to be executed
    - Possible preemption
  - Example



a : Priority 1
b : Priority 2

Static scheduling algorithms

- **Preemptive fixed priority scheduling (FP)**
  - Very close to HW (asynchronous I/O)
    - Scheduling can be done at hardware level
  - Punctuality for a task
    - And for the others ?
  - Low security
  - Weak liveness
    - If there are a lot of higher priority tasks, risk of starvation
    - Drawbacks improvements
      - » *In-line change of priority (Unix / Windows)*
  - Analysis remain complicated to handle control on processing durations
    - => classic system because no urgency notion

Static scheduling algorithms

- **RMS (Rate Monotonic Scheduling)**
  - Created by Liu & Layland
  - Model hypothesis (H1)
    - Fixed priority algorithm (constant over the time)
    - Possible preemption
    - Each task is periodic
    - No dependency between tasks
    - Deadline is period
    - The priority is the inverse of the period
  - Feasibility hypothesis (H2)
    - Sufficient condition exists (CS)
      - » *Theoretical bound corresponding to the worst case*
    - Safe scheduling if this criteria is verified

Static scheduling algorithms

- **RMS (Rate Monotonic Scheduling)**
  - Theoretical criteria
    - n tasks
    - Ci = execution duration
    - Ti = period = job deadline
    - Analysis if the CPU use rate (W) : $\sum_i \frac{C_i}{T_i}$
    - If W ≤ U(n) , RMS is optimal

$$
\begin{aligned}
U(n) &= n*(2^{1/n}-1)\\
U(1) &= 1\\
U(2) &= 0.83\\
U(3) &= 0.78\\
U(\infty) &= 0.69 = \log 2
\end{aligned}
$$

Static scheduling algorithms

- **RMS (Rate Monotonic Scheduling)**

$$W = \sum C_i / T_i$$

$$U(n) = n*(2^{1/n}-1)$$

  - Necessary condition: W ≤ 1 (overload otherwise)
  - Sufficent condition: W ≤ U(n)
  - For W between CS and CN : impossible to know if there is a solution or not
    - One must « manually » produce the schedule on the interval of analysis

Static scheduling algorithms

- **RMS conclusion**
  - Advantages
    - Can be extended to an aperiodic task
    - Can be extended to handle overload
      - » *Highest priority tasks are not impacted by the lowest ones*
    - Easy to implement
      - » *Very close to loop programming*
  - Drawbacks
    - Very simple hypothesis (rarely used in practice)
    - Starvation if bug in a high priority task
      - » *To check the duration taken by each task*

Dynamic scheduling algorithms

- **First Come/First Served**
  - Advantages
    - Very easy and basic
      - » *Non preemptive*
    - Liveness (if tasks ends ofc…)
  - Drawbacks
    - Punctuality (=> classical system)
      - » *Can penalize short duration tasks if a long duration one is already executed*
    - Security
  - Useful to keep an implicit order of processing (for I/O)
    - Spooler of printers
  - FCFS is the « by default » strategy most of the time for a lot of resources (memory, TCP/IP ports …)

Dynamic scheduling algorithms

- **Shortest first**
  - Advantages
    - Very easy and basic (Non preemptive)
    - Liveness (if tasks ends ofc…)
  - Drawbacks
    - Punctuality (=> classical system)
      - » *Long tasks are penalized*
    - Security
    - Necessary to know task durations
      - » *Something we do not necessarily know (classical system)*
  - Shortest duration first
    - « shortest first » with preemption version
    - Preemption when the shorter execution time of a task becomes ready

Dynamic scheduling algorithms

- **Round Robin (tourniquet)**
  - We share time in a « fair » way between any tasks that are ready to execute
    - We let (at most) « K » units of time to a task (quantum)
    - After consuming its quantum of time, we put the task at the end of the waiting queue of ready tasks
  - Advantages
    - Liveness
    - Parallelize for I/O and processing parts

Dynamic scheduling algorithms

- **Round Robin (tourniquet)**
  - Drawbacks
    - Punctuality (=> classical system)
    - Performances depends on the sie of the time quantum
      - » *Too large* ➔ *a task can wait a long time to have acces to the processor (response time)*
      - » *Too small* ➔ context switches are too numerous and their overheads become non negligible
  - In pratice, RR is linked with fixed priority

Dynamic scheduling algorithms

- **EDF (Earliest Deadline First)**
  - Model hypothesis
    - Dynamic
    - Aperiodic task
      - » *Periodic problems included*
    - No dependency between tasks
    - Known deadlines
    - Unknown durations
    - Priority is reverse to the relative deadline
    - Preemption
  - Hypothesis of feasibility
    - Optimal if no overload ($W \leq 1$)

Dynamic scheduling algorithms

- **LLF (Least Laxity First)**
  - Model hypothesis
    - Dynamic
    - Aperiodic task
      - » *Periodic problems included*
    - No dependency between tasks
    - Known deadlines
    - Known durations
    - Priority is reverse to the laxity
      - » *On-line, the scheduler computes the laxity and executes the one with the least laxity*
    - Preemption
  - Hypothesis of feasibility
    - Optimal if no overload (W ≤ 1)

Dynamic scheduling algorithms

- **Difference between EDF and LLF**
  - If laxity values used are those computed at release times of the tasks
    - Equivalent scheduling
  - If laxity values are computed at each time
    - LLF leads to more context switches
      - » *Laxity of the executed task remains constant while the other ready ones decrease their laxity*

Dynamic scheduling algorithms

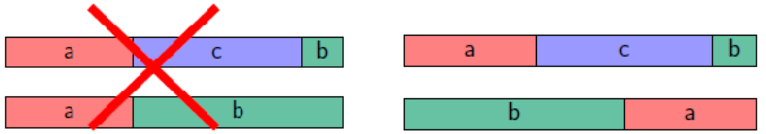
- **Difference between theory and practice**
  - Unfortunately, previous hypothesis are almost never verified
    - Preemption
      - » *Takes time*
    - Integration of critical / non critical tasks
      - » *Often in overload*
    - Processing independance
      - » *Resources sharing where the use is reduced to mutual exclusion only: critical resources*
      - » *Precedence constraints that exhibit synchronization and communication between tasks*
        - ➔ *Dependency graphs*

Scheduling algorithm and synchronization

- **Priority and synchronization**
  - Livelock possibility
    - If a lower priority task takes a mutex and then one with higher priority requests it
  - Solution : priority inheritance
    - The lock owner inherits priority from the requesting one until it releases the lock
    - In an important system: any tasks often end up with the same priority
    - Increase the difficulty of scheduling analysis

Multiprocessor scheduling

- **Scheduling on multiprocessor**
  - At any time t
    - A job is executed by at most one processor
    - A processor executes at most one job
  - Non-cumulative scheduling
    - It becomes more complex compared to the monoprocessor case

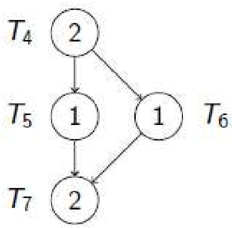


Multiprocessor scheduling

- **Scheduling on multiprocessor**
  - Latencies are not negligible
    - Communication, migrations
  - In monoprocessor
    - « On-line » optimal algorithm (EDF, LLF)
  - In multiprocessor
    - Necessary « to know the future » for optimal scheduling… (Global EDF is non optimal)
    - Different strategies
      - » *Partitionned (forbidden migrations)*
      - » *Global algorithms*
      - » *Semi-partitionned (current trend, showing best results)*

Multiprocessor scheduling

- **Dependencies between jobs**
  - Jobs can be linked by a dependency graph
    - Expresses precedency and concurrency
    - Concurrent accesses to resources, serializations



Multiprocessor scheduling

- **Dependencies between jobs**
  - Task release date is <u>superior</u> to any release dates of its direct predecessors increased by their execution duration
    - A task will only be released if all its predecessors are finished
  - Task deadline shall be <u>inferior</u> to any deadlines of its direct successors decreased by their execution duration
  - Example for the graph (T1) ➔ (T2) ➔ (T3)
    - Tr(T1) + C(T1) = Td(T2), earliest start time
    - Td(T3) - C(T3) = Td(T2), latest end time

Multiprocessor scheduling

- **Hypothesis on a concrete case**
  - Dependency graph
  - Two resources are available (two processors)
  - No preemption
  - Completion in 6 units of time?