AADL : about scheduling analysis

Scheduling analysis, what is it ?

- **Embedded real-time critical systems** have temporal constraints to meet (e.g. deadline).
- Many systems are built with operating systems providing multitasking facilities … Tasks may have deadline.
- **But, tasks make temporal constraints analysis difficult to do :**
  - We must take the task scheduling into account in order to check task temporal constraints.
  - Scheduling (or schedulability) analysis.

Summary

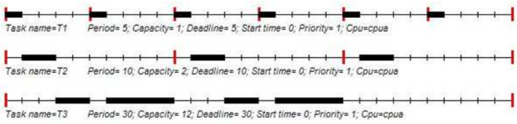
1. Issues about real-time scheduling analysis : AADL to the rescue
2. Basics on scheduling analysis : fixed-priority scheduling for uniprocessor architectures
3. AADL components/properties to scheduling analysis

Real-Time scheduling theory

1. **A set of simplified tasks models** (to model functions of the system)
2. **A set of analytical methods** (called feasibility tests)
   - **Example:**

$$R_i \leq Deadline \qquad R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{P_j} \right\rceil \cdot C_j$$

3. **A set of scheduling algorithms**: build the full scheduling/GANTT diagram



Real-Time scheduling theory is hard to apply

- Real-Time scheduling theory
  - Theoretical results defined from 1974 to 1994: feasibility tests exist for uniprocessor architectures
- Now supported at a decent level by POSIX 1003 real-time operating systems, ARINC653, …
- Industry demanding
  - Yet, hard to use

Real-Time scheduling theory is hard to apply

- Requires strong theoretical knowledge/skills
  - Numerous theoretical results: how to choose the right one ?
  - Numerous assumptions for each result.
  - How to abstract/model a system to verify deadlines?
- How to integrate scheduling analysis in the engineering process ?
  - When to apply it ? What about tools ?

It is the role of an ADL to hide those details

AADL to the rescue ?

- AADL helps modeling a full system, including hardware, task sets, connections, operating system features, …
- All of these elements are mandatory to apply real-time scheduling theory
  - Examples: an AADL model can include
    - Task execution time or task deadline or task release times
    - Scheduling parameters
- However, in many cases, the models stay too complex
  - Multiprocessor architectures, shared buffers or buses, …

Summary

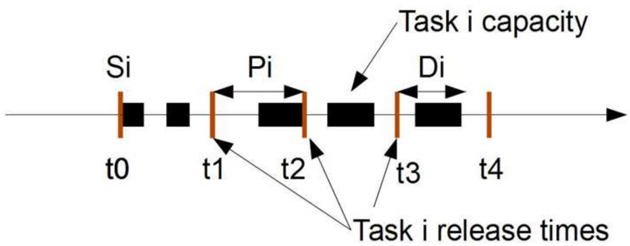
1. Issues about real-time scheduling analysis : AADL to the rescue
2. Basics on scheduling analysis : fixed-priority scheduling for uniprocessor architectures
3. AADL components/properties to scheduling analysis

Real-time scheduling theory : models of task

- **Task simplified model:** sequence of statements + data.

- **Usual kind of tasks:**
  - Independent tasks or dependent tasks.
  - Periodic and sporadic tasks (critical functions) : have several jobs and release times
  - Aperiodic tasks (non critical functions) : only one job and one release time

Real-time scheduling theory : models of task



- **Usual parameters of a periodic task i:**
  - **Period:** Pi (duration between two release times). A task starts a job for each release time.
  - **Deadline to meet:** Di, timing constraint to meet.
  - **First task release time (first job):** Si.
  - **Worst case execution time of each job:** Ci (or capacity or WCET).
  - **Priority:** allows the scheduler to choose the task to run

Real-time scheduling theory : models of task

- **Assumptions for the next slides (synchronous periodic task with deadlines on requests):**
  - All tasks are periodic.
  - All tasks are independent.
  - ∀i : Pi=Di : a task must end its current job before its next release time.
  - ∀i : Si=0 => called critical instant (worst case on processor demand).

Uniprocessor fixed priority scheduling

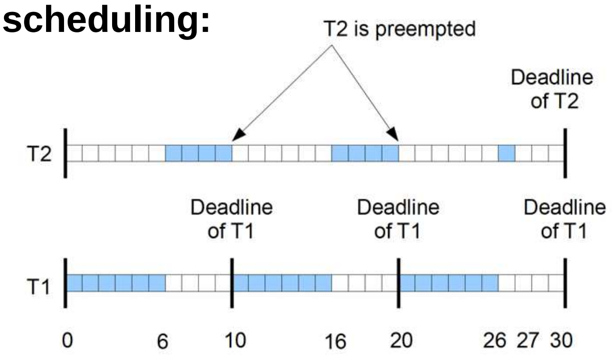
- **Fixed priority scheduling :**
  - Scheduling based on fixed priority => priorities do not change during execution time.
  - Priorities are assigned at design time (off-line).
  - Efficient and simple feasibility tests.
  - Scheduler easy to implement into real-time operating systems.
- **Rate Monotonic priority assignment :**
  - Optimal assignment in the case of fixed priority scheduling and uniprocessor.
  - Periodic tasks only.

Uniprocessor fixed priority scheduling

- **Two steps:**
  1. **Rate monotonic priority assignment:** the highest priority tasks have the smallest periods. Priorities are assigned off-line (e.g. at design time, before execution).
  2. **Fixed priority scheduling**: at any time, run the ready task which has the highest priority level.

Uniprocessor fixed priority scheduling

- **Rate Monotonic assignment and preemptive fixed priority scheduling:**



- Assuming VxWorks priority levels (high=0 ; low=255)
- T1 : C1=6, P1=10, Prio1=0
- T2 : C2=9, P2=30, Prio2=1

Uniprocessor fixed priority scheduling

- **Feasibility/Schedulability tests <u>to predict on design-time if deadline will be met</u>:**

  1. **Run simulations on hyperperiod** = [0,LCM(Pi)]. Sufficient and necessary condition.
  2. **Processor utilization factor test:**
     $U = \sum_{i=1}^{n} Ci/Pi \leq n.(2^{\frac{1}{n}}\text{-}1)$ (about 69%)
     Rate Monotonic assignment and preemptive scheduling. Sufficient but not necessary condition.
  3. **Task worst case response time, noted Ri** : delay between task release time and task completion time. Any priority assignment but preemptive scheduling.

Uniprocessor fixed priority scheduling

- **Compute Ri, task i worst case response time:**
  - Task i response time = task i capacity + delay the task i has to wait for higher priority task j. Or:

$$R_i = C_i + \sum_{j \in hp(i)} waiting\ time\ due\ to\ j \qquad or\ R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{P_j} \right\rceil \cdot C_j$$

  - hp(i) is the set of tasks which have a higher priority than task i.
  - $\lceil x \rceil$ returns the smallest integer not smaller than x.

Uniprocessor fixed priority scheduling

- To compute task response time: compute wi^k with:

$$wi^n = Ci + \sum_{j \in hp(i)} \lceil wi^{n-1}/Pj \rceil . Cj$$

- Start with wi^0=Ci.
- Compute $wi^1, wi^2, wi^3, \ldots wi^k$ upto:
  - If wi^k >Pi. No task response time can be computed for task i. Deadlines will be missed !
  - If wi^k = wi^{k-1}. wi^k is the task i response time. Deadlines will be met.

Uniprocessor fixed priority scheduling

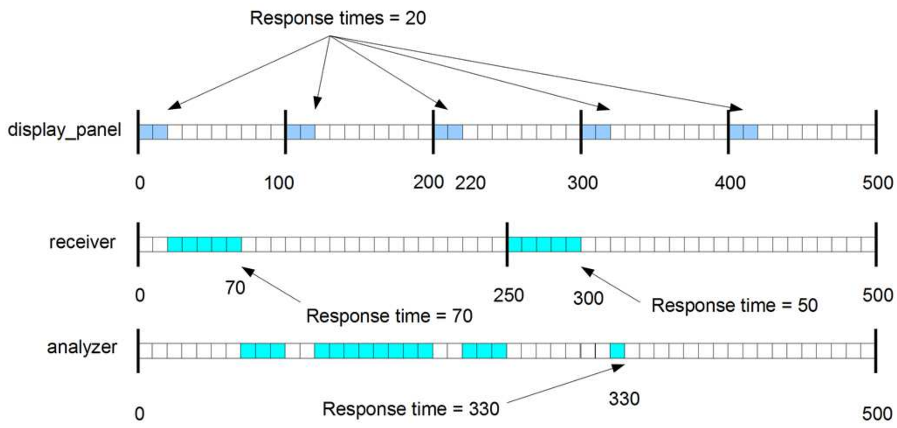
- **Example:** T1(P1=7, C1=3), T2 (P2=12, C2=2), T3 (P3=20, C3=5)

$$w1^0 = C1 = 3 \Rightarrow r1 = 3$$
$$w2^0 = C2 = 2$$
$$w2^1 = C2 + \left\lceil \frac{w2^0}{P1} \right\rceil . C1 = 2 + \left\lceil \frac{2}{7} \right\rceil . 3 = 5$$
$$w2^2 = C2 + \left\lceil \frac{w2^1}{P1} \right\rceil . C1 = 2 + \left\lceil \frac{5}{7} \right\rceil . 3 = 5 \Rightarrow r2 = 5$$
$$w3^0 = C3 = 5$$
$$w3^1 = C3 + \left\lceil \frac{w3^0}{P1} \right\rceil . C1 + \left\lceil \frac{w3^0}{P2} \right\rceil . C2 = 10$$
$$w3^2 = C3 + \left\lceil \frac{w3^1}{P1} \right\rceil . C1 + \left\lceil \frac{w3^1}{P2} \right\rceil . C2 = 13$$
$$w3^3 = C3 + \left\lceil \frac{w3^2}{P1} \right\rceil . C1 + \left\lceil \frac{w3^2}{P2} \right\rceil . C2 = 15$$
$$w3^4 = C3 + \left\lceil \frac{w3^3}{P1} \right\rceil . C1 + \left\lceil \frac{w3^3}{P2} \right\rceil . C2 = 18$$
$$w3^5 = C3 + \left\lceil \frac{w3^4}{P1} \right\rceil . C1 + \left\lceil \frac{w3^4}{P2} \right\rceil . C2 = 18 \Rightarrow r3 = 18$$

Uniprocessor fixed priority scheduling

- **Example with the AADL case study:**
  - “display_panel” thread which displays data. P=100, C=20.
  - “receiver” thread which sends data. P=250, C=50.
  - “analyser” thread which analyzes data. P=500, C=150.
- **Processor utilization factor test:**
  - U=20/100+150/500+50/250=0.7
  - Bound=3.$(2^{\frac{1}{3}}-1)$=0.779
  - U≤Bound => deadlines will be met.
- **Task response time:** $R_{analyser}$=330, $R_{display\_panel}$=20, $R_{receiver}$ =70.
- **Run simulations on hyperperiod:** [0,LCM(Pi)] = [0,500].

Uniprocessor fixed priority scheduling



Fixed priority and shared resources

- Previous tasks were independent … does not really exist in true life.

- **Task dependencies :**
  - Shared resources.
    - E.g. with AADL: threads may wait for AADL protected data component access.
  - Precedencies between tasks.
    - E.g with AADL: threads exchange data by data port connections.

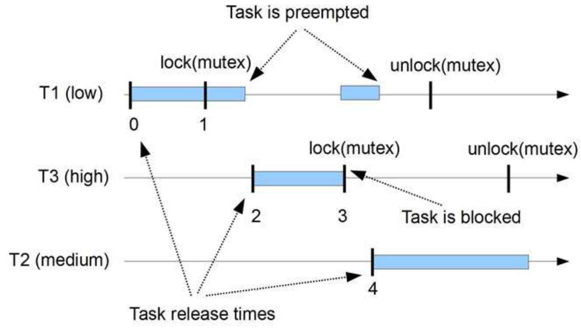
Fixed priority and shared resources

- Shared resources are modeled by semaphores for scheduling analysis.
- **We use specific semaphores implementing inheritance protocols:**
  - To take care of priority inversion.
  - To compute worst case task waiting time for the access to a shared resource. Blocking time Bi.

- **Inheritance protocols:**
  - PIP (Priority inheritance protocol**)**, can not be used with more than one shared resource due to deadlock.
  - PCP (Priority Ceiling Protocol) , implemented in most of real-time operating systems (e.g. VxWorks).
  - Several implementations of PCP exists: OPCP, ICPP, …

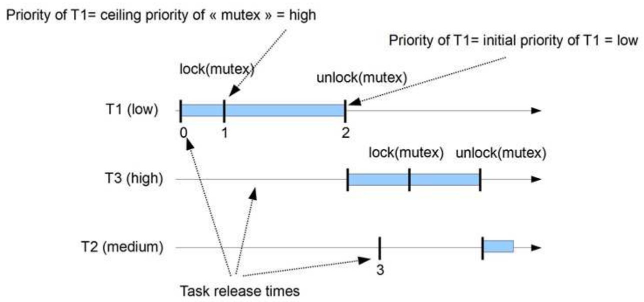
Fixed priority and shared resources

- **What is Priority inversion:** a low priority task blocks a high priority task



- B_i = worst case on the shared resource waiting time.

Fixed priority and shared resources



- **ICPP (Immediate Ceiling Priority Protocol):**
  - Ceiling priority of a resource = maximum fixed priority of the tasks which use it.
  - Dynamic task priority = maximum of its own fixed priority and the ceiling priorities of any resources it has locked.
  - B_i=longest critical section ; prevent deadlocks

Fixed priority and shared resources

- **How to take into account the waiting time Bi:**

  - Processor utilization factor test :

$$\forall i, 1 \leq i \leq n : \sum_{k=1}^{i-1} \frac{Ck}{Pk} + \frac{Ci+Bi}{Pi} \leq i.(2^{\frac{1}{i}} - 1)$$

  - Worst case response time :

$$R_i = B_i + C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{P_j} \right\rceil \cdot C_j$$

To conclude on scheduling analysis

- **Many feasibility tests:** depending on task, processor, scheduler, shared resource, dependencies, multiprocessor, hierarchical, distributed, …

$$R_i = B_i + C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{P_j} \right\rceil \cdot C_j$$

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{P_j} \right\rceil \cdot C_j$$

$$R_i = w_i + J_i$$

$$w_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i + J_j}{P_j} \right\rceil \cdot C_j$$

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{P_j} \right\rceil \cdot C_j + \max(C_k \forall k \in hp(i))$$

- **Many assumptions :** require preemptive, fixed priority scheduling, synchronous periodic, independent tasks, deadlines on requests …

Many feasibility tests …. Many assumptions …

How to choose them?

Summary

1. Issues about real-time scheduling analysis : AADL to the rescue
2. Basics on scheduling analysis : fixed-priority scheduling for uniprocessor architectures
3. AADL components/properties to scheduling analysis

AADL to the rescue ?

- **Issues:**
  - Ensure all required model elements are given for the analysis
  - Ensure model elements are compliant with analysis requirements/assumptions
- **AADL helps for the first issue:**
  - AADL as a pivot language between tools. International standard.
  - Close to the real-time scheduling theory: real-time scheduling analysis concepts can be found. Ex:
    - Component categories: thread, data, processor
    - Property: `Deadline, Fixed Priority, ICPP, Ceiling Priority, …`

Property sets for scheduling analysis

- **Properties related to processor component:**

```
Preemptive_Scheduler : aadlboolean applies to (processor);

Scheduling_Protocol:
 inherit list of Supported_Scheduling_Protocols
 applies to (virtual processor, processor);
-- RATE_MONOTONIC_PROTOCOL,
-- POSIX_1003_HIGHEST_PRIORITY_FIRST_PROTOCOL, ..
```


Property sets for scheduling analysis

- **Properties related to the threads/data components:**

```
Compute_Execution_Time: Time_Range
 applies to (thread, subprogram, …);

Deadline: inherit Time => Period applies to (thread, …)

Period: inherit Time applies to (thread, …);

Dispatch_Protocol: Supported_Dispatch_Protocols
 applies to (thread);
--  Periodic, Sporadic, Timed, Hybrid, Aperiodic, Backg
...

Priority: inherit aadlinteger applies to (thread, …, da

Concurrency_Control_Protocol:
 Supported_Concurrency_Control_Protocols applies to (da
--  None, PCP, ICPP, …
```

Property sets for scheduling analysis

- **Example:**

```
thread implementation receiver.impl
  properties
    Dispatch_Protocol => Periodic;
    Compute_Execution_Time => 31 ms .. 50 ms;
    Deadline => 250 ms;
    Period => 250 ms;
end receiver.impl;

data implementation target_position.impl
  properties
    Concurrency_Control_Protocol
        => PRIORITY_CEILING_PROTOCOL;
end target_position.impl;
```

```
process implementation processing.others
  subcomponents
    receiver : thread receiver.impl;
    analyzer : thread analyzer.impl;
    target : data target_position.impl;
    . . .
processor implementation leon2
  properties
    Scheduling_Protocol =>
          RATE_MONOTONIC_PROTOCOL;
      Preemptive_Scheduler => true;
end leon2;

system implementation radar.simple
subcomponents
  main : process processing.others;
  cpu : processor leon2;
  . . .
```


Cheddar : a framework to access schedulability of AADL models

- **Cheddar tool =** analysis framework (queueing system theory & real-time scheduling theory)
  + internal ADL (architecture description language)
  + various standard ADL parsers (AADL, MARTE UML)
  + simple model editor
  + …

- **Two versions :**
  - Open source (Cheddar) : educational and research.
  - Commercial product (AADLInspector) : Ellidiss Tech product.

- **Supports :** Ellidiss Tech., Conseil régional de Bretagne, BMO, EGIDE/Campus France, Thales Communication, BPI France

Cheddar : a framework to access schedulability of AADL models

- **Demos:**
  - Scheduling analysis of the radar example with Cheddar
  - .. And with AADLInspector also