Real-Time Scheduling: Some Results and Open Problems

Risat Mahmud Pathan

Chalmers University of Technology, Sweden

Introduction

- Multiprocessors, specifically CMPs, are considered for many embedded real-time systems (e.g., automotive)
- The application of real-time systems are often modeled as a collection of recurrent tasks (e.g., control applications)
- Hard real-time systems must meet all the deadlines of its application tasks during runtime
- Problem: How can we guarantee that all the tasks deadlines are met on m identical processors?

Task Model

- We consider a set of **recurrent** real-time task set

$$\Gamma = \{\tau_1, \tau_2, \ldots \tau_n\}$$

- Each task τ_i has three parameters (C_i, D_i, T_i)
  - ***Implicit-deadline*** if $D_i = T_i$
  - ***Constrained-deadline*** if $D_i \leq T_i$
  - ***Total utilization*** $U = \sum u_i = \sum \frac{C_i}{T_i}$
- Tasks are given fixed priorities
- Tasks are scheduled on m identical processors

Scheduling Paradigms

- Global Scheduling: task can execute on any processor even when resumed after preemption
- Partitioned Scheduling: task can execute in exactly one processor to which it is assigned
- Task-Splitting: few tasks are allowed to migrate (global scheduling flavor) and each of the remaining tasks executes on a fixed processor to which they are assigned (partitioned scheduling flavor).

Global Fixed-Priority Scheduling

The challenge for global FP scheduling

Two Problems

- **Priority Assignment: How to assign the fixed priorities for a given task set?**
- **Schedulability Test: How to guarantee the schedulability of a given task set?**

Our work @ ECRTS 2011

Priority Assignment and Utilization Bound Test

Proposed new fixed-priority assignment policy, called `ISM-US`, and derived the schedulability utilization bound

Priority Assignment and Iterative Test

Proposed an improved fixed-priority assignment policy and iterative schedulability test

- Utilization bound test: Compare the total utilization of a task set with the guarantee bound (i.e., one test).
- Iterative test: Apply the test to one by one task (i.e., n tests)

Utilization Bound Test

Priority Assignment Policy `ISM-US`

Hybrid (Slack-Monotonic) Priority Assignment (`HPA`)

A subset of the tasks are given slack-monotonic priority and the other tasks are given the highest fixed-priority

Slack-Monotonic (SM)

Task τ_i has higher SM priority than task τ_k if and only if $(T_i - C_i < T_k - C_k)$

Priority Assignment Policy `ISM-US`

Policy `ISM-US`

If $u_i > u_{ts}$, then task τ_i is given the highest fixed-priority, otherwise, task τ_i is given slack-monotonic priority

Threshold Utilization

$$u_{ts} = \frac{3m - 2 - \sqrt{5m^2 - 8m + 4}}{2m - 2}$$

Priority Assignment Policy `ISM-US`

Policy `ISM-US`

If $u_i > u_{ts}$, then task τ_i is given the highest fixed-priority, otherwise, task τ_i is given slack-monotonic priority

Threshold Utilization

$$u_{ts} = \frac{3m - 2 - \sqrt{5m^2 - 8m + 4}}{2m - 2}$$

Theorem (Utilization Bound)

If $\mathbf{U \leq m \cdot \min\{0.5, u_{ts}\}}$, then all the deadlines of task set Γ are met using global FP scheduling

State-of-the-art utilization bound

RM-US[$\frac{1}{3}$] — M. Bertogna et. al., OPODIS 2005

If $u_i > \frac{1}{3}$, then task τ_i is given the highest fixed-priority, otherwise, task τ_i is given *rate-monotonic* priority

Utilization Bound: $\frac{m+1}{3}$

State-of-the-art utilization bound

RM-US[$\frac{1}{3}$] — M. Bertogna et. al., OPODIS 2005

If $u_i > \frac{1}{3}$, then task τ_i is given the highest fixed-priority, otherwise, task τ_i is given *rate-monotonic* priority

Utilization Bound: $\frac{m+1}{3}$

SM-US[$\frac{2}{3+\sqrt{5}}$] — B. Andersson, OPODIS 2008

If $u_i > \frac{2}{3+\sqrt{5}}$, then task τ_i is given the highest fixed-priority, otherwise, task τ_i is given *slack-monotonic* priority

Utilization Bound: $\frac{2m}{3+\sqrt{5}}$

State-of-the-art utilization bound

RM-US[$\frac{1}{3}$] — M. Bertogna et. al., OPODIS 2005

If $u_i > \frac{1}{3}$, then task τ_i is given the highest fixed-priority, otherwise, task τ_i is given *rate-monotonic* priority

Utilization Bound: $\frac{m+1}{3}$

SM-US[$\frac{2}{3+\sqrt{5}}$] — B. Andersson, OPODIS 2008

If $u_i > \frac{2}{3+\sqrt{5}}$, then task τ_i is given the highest fixed-priority, otherwise, task τ_i is given *slack-monotonic* priority

Utilization Bound: $\frac{2m}{3+\sqrt{5}}$

State-of-the-art Utilization Bound

- **If $m \leq 6$, then RM–US[$\frac{1}{3}$] is the best**
- **If $m > 6$, then SM–US[$\frac{2}{3+\sqrt{5}}$] is the best**

Comparison with our bound

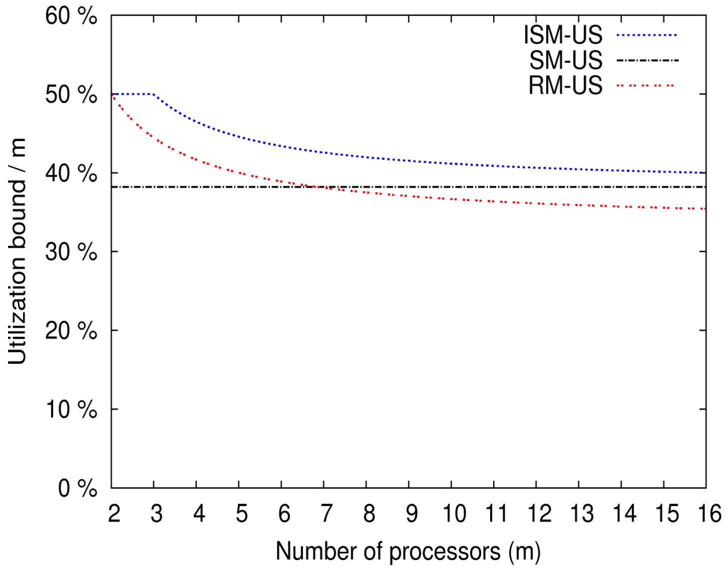


Figure: Utilization bounds of RM−US[$\frac{1}{3}$], SM−US[$\frac{2}{3+\sqrt{5}}$] and proposed ISM−US

HPA policy and Global Scheduling

Separation of Concern

- During schedulability analysis, each highest priority task τ_i's WCET is set to T_i and one processor is (virtually) dedicated to τ_i ***without any concern***.
- The problem now ***reduces*** to the schedulability of the other (lower) priority tasks on $(m - m')$ processors (m' is the number of ***heavy*** tasks)

Iterative Schedulability Test

Iterative Schedulability test

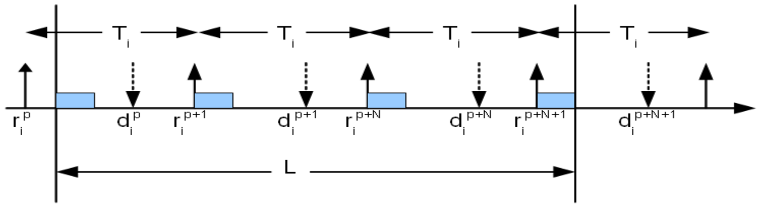
- We consider ***constrained-deadline*** task systems
- We improved the priority assignment policy for an iterative test, called the `DA-LC` test, proposed by Davis and Burns (RTSJ, 2011).

Interference and Workload

When considering the schedulability of a lower priority task τ_k within the ***scheduling window***, the `DA-LC` test considers

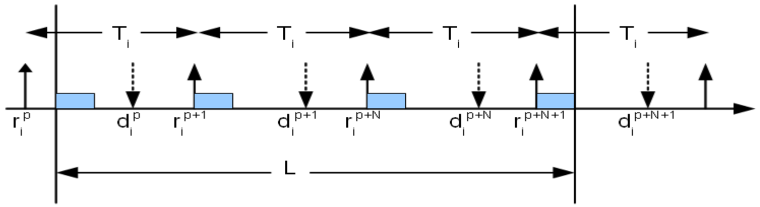
- the ***interference*** of each higher priority task $\tau_i \in hp(k)$
- based on the ***workload*** of each higher priority task τ_i in set $hp(k)$
- where each higher priority task τ_i is considered either a ***carry-in*** or a ***non carry-in*** task

Carry-in and Non Carry-in Interference

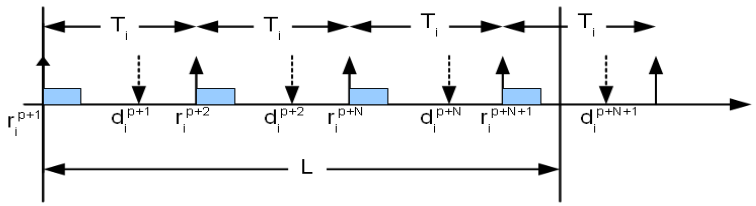


$I_{i,k}^{C}$ = **carry-in interference** of task τ_i on τ_k

Carry-in and Non Carry-in Interference



$I^{C}_{i,k}$ = **carry-in interference** of task τ_i on τ_k



$I^{NC}_{i,k}$ = **non carry-in interference** of task τ_i on τ_k

The DA−LC test

- The DA−LC test (Davis et al. RTSJ 2011) for task τ_k is given as follows:

$$D_k \geq C_k + \left\lfloor \frac{I_k}{m} \right\rfloor$$

The DA−LC test

- The DA−LC test (Davis et al. RTSJ 2011) for task τ_k is given as follows:

$$D_k \geq C_k + \left\lfloor \frac{I_k}{m} \right\rfloor$$

- The function I_k is calculated as follows:

$$I_k = \sum_{i \in hp(k)} I^{NC}_{i,k} + \sum_{i \in Max(k,m-1)} I^{DIFF}_{i,k}$$

The DA−LC test

- The DA−LC test (Davis et al. RTSJ 2011) for task τ_k is given as follows:

$$D_k \geq C_k + \left\lfloor \frac{I_k}{m} \right\rfloor$$

- The function I_k is calculated as follows:

$$I_k = \sum_{i \in hp(k)} I^{NC}_{i,k} + \sum_{i \in Max(k,m-1)} I^{DIFF}_{i,k}$$

- where
  - $Max(k, m-1)$ is the set of $(m-1)$ higher priority tasks in $hp(k)$ that have the largest value of $I^{DIFF}_{i,k}$, and

The DA−LC test

- The DA−LC test (Davis et al. RTSJ 2011) for task τ_k is given as follows:

$$D_k \geq C_k + \left\lfloor \frac{I_k}{m} \right\rfloor$$

- The function I_k is calculated as follows:

$$I_k = \sum_{i \in hp(k)} I^{NC}_{i,k} + \sum_{i \in Max(k,m-1)} I^{DIFF}_{i,k}$$

- where
  - $Max(k, m-1)$ is the set of $(m-1)$ higher priority tasks in $hp(k)$ that have the largest value of $I^{DIFF}_{i,k}$, and
  - $I^{DIFF}_{i,k} = I^{C}_{i,k} - I^{NC}_{i,k}$

The DA−LC test

R. Davis and A. Burns (RTSJ, 2011) have showed that

- Audsley's Optimal Priority Assignment(OPA) algorithm is applicable to the DA−LC test
- Empirically shown that DA−LC+OPA outperforms all other existing test

OPA+DA−LC is the state-of-the-art iterative schedulability tests

Audsley's OPA for multiprocessors (RTSS, 2009)

Algorithm OPA (Taskset A, number of processors $\hat{m}$, Test S)

1. for each priority level k, lowest first
2. for each priority unassigned task $\tau \in A$
3. If τ is schedulable using S on $\hat{m}$ processors at priority k
4. assign τ to priority k
5. break (continue outer loop)
6. return "unschedulable"
7. return "schedulable"

OPA+DA−LC (RTSJ, 2011)

Call OPA (Γ, m, DA−LC)

Our Observation @ ECRTS 2011

- OPA +DA−LC is proved optimal (RTSJ, 2011).

Our Observation @ ECRTS 2011

- OPA +DA−LC is proved optimal (RTSJ, 2011).
- This combination is optimal only under the assumption that it is applied to the entire task set and to all processors
  - i.e.,Call OPA($\Gamma, m,$ DA−LC)

Our Observation @ ECRTS 2011

- OPA +DA−LC is proved optimal (RTSJ, 2011).
- This combination is optimal only under the assumption that it is applied to the entire task set and to all processors
  - i.e.,Call OPA($\Gamma, m,$ DA−LC)

Scope for Improvement?

- Is it possible to obtain a more effective priority assignment if
  - OPA+DA−LC is applied to a **subset** of the entire task set and on a **lower** number of processors
  - while other tasks are assigned the highest priorities based on HPA and predictability?

Interesting Observation

- Recall the DA−LC test for task τ_k:

$$D_k \geq C_k + \left\lfloor \frac{I_k}{m} \right\rfloor$$

- I_k depends on $(m-1)$ carry-in terms

$$I_k = \sum_{i \in hp(k)} I^{NC}_{i,k} + \sum_{i \in Max(k,m-1)} I^{DIFF}_{i,k}$$

Interesting Observation

- Recall the DA−LC test for task τ_k:

$$D_k \geq C_k + \left\lfloor \frac{I_k}{m} \right\rfloor$$

- I_k depends on $(m-1)$ carry-in terms

$$I_k = \sum_{i \in hp(k)} I^{NC}_{i,k} + \sum_{i \in Max(k,m-1)} I^{DIFF}_{i,k}$$

Observation

- If we remove one task, say τ_h, from $hp(k)$ and
- reduce the number of processors from m to $(m-1)$, and
- apply the OPA+DA−LC test on $(\Gamma - \{\tau_h\})$ and on $(m-1)$ processors,
- then I_k depends on $(m-2)$ carry-in tasks in $(hp(k) - \{\tau_h\})$

Example

- Consdier $\Gamma = \{\tau_1, \ldots \tau_4\}$ and $m = 3$
- $(C_i, D_i, T_i) = \{(23, 33, 33), (106, 210, 214), (58, 216, 217), (46, 60, 64)\}$
- **OPA (Γ, $m = 3$, DA−LC) returns "unschedulable"**
- I_3 considers $(m-1) = 2$ as carry-in task

Example

- Consdier $\Gamma = \{\tau_1, \ldots \tau_4\}$ and $m = 3$
- $(C_i, D_i, T_i) = \{(23, 33, 33), (106, 210, 214), (58, 216, 217), (46, 60, 64)\}$
- **OPA (Γ, $m = 3$, DA−LC) returns "unschedulable"**
- I_3 considers $(m-1) = 2$ as carry-in task

- The highest density (i.e.,C_i/D_i) task τ_4 is given the highest priority
- ***OPA*** **(**$\{\tau_1, \tau_2, \tau_3\}$**,** $m = 2$**, *DA−LC*) returns "schedulable"**
- I_3 considers $(m-1) = 1$ task as carry-in task

HPA+OPA +DA−LC

Algorithm HybridOPA (Γ, m)

1. **for** $m' = 0$ **to** $(m-1)$
2. remove m' highest desnity tasks from given task set Γ
3. **if** OPA (Γ, $m - m'$, DA−LC) returns "schedulable" then
4. **return** "schedulable"
5. **end for**
6. **return** "unschedulable"

We call this test HP−DA−LC test

Task Splitting Algorithm

Task Splitting

Background

- **Global** and **partitioned** method cannot guarantee system utilization more than 50% for all task sets (Lecture 7)

  —Partitioned scheduling has task assignment step.

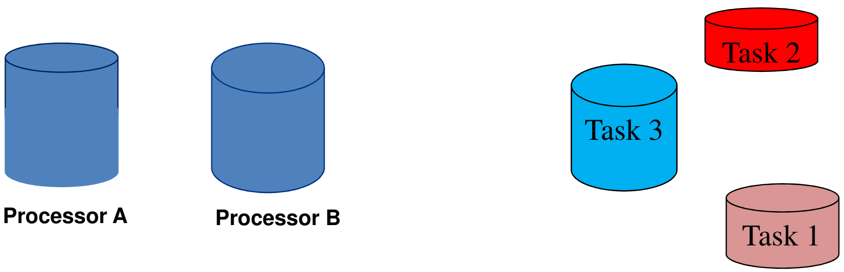
  —Task assignment to processors is generally done with a bin-packing algorithm.

Task Splitting

Background (cont.)

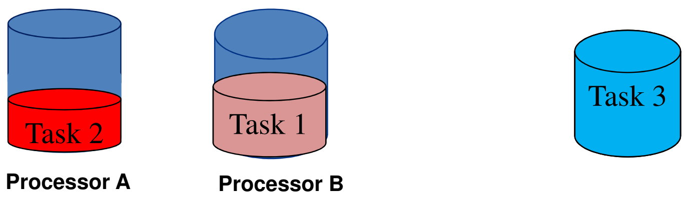
- A ***variation of partitioned scheduling*** using **task-splitting** approach can achieve more than 50% system utilization for all task sets.
- **History**: task-splitting for static-priority were first proposed in July 2009 at CMU

Traditional Partitioned Scheduling



We assume Task 2, Task 1 and Task 3 be the ordering of the tasks to assign to the processors A and B.
Size of each task is proportional to the utilization of the task.

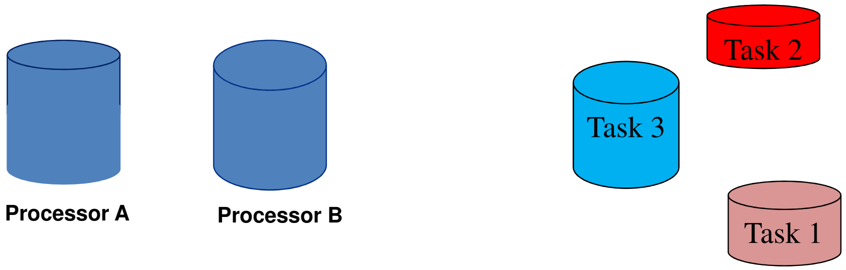
Traditional Partitioned Scheduling



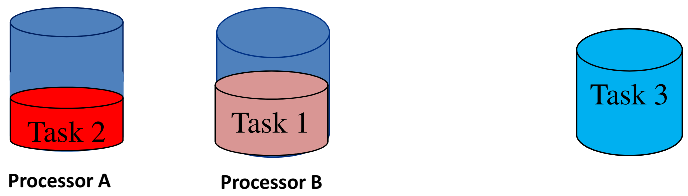
Partition Fails!

Task 3 cannot be assigned to any processor because size of Task 3 is too large

Task-Splitting Partitioned Scheduling

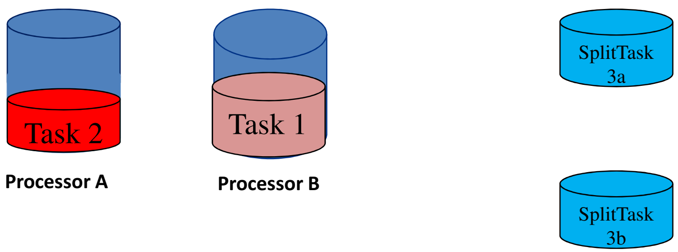


Task-Splitting Partitioned Scheduling



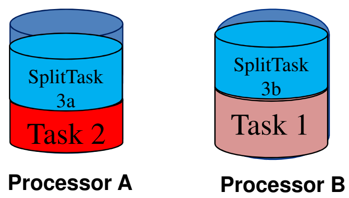
Different subtasks of Task 3 can be assigned to different processors.
To construct the subtasks, we split Task 3.

Task-Splitting Partitioned Scheduling



Different subtasks of Task 3 can be assigned to different processors.
To construct the subtasks, we split Task 3.

Task-Splitting Partitioned Scheduling



Partition Success!

Challenges in Task-Splitting

- **How to design the task assignment algorithm?**
  - How many splits of each task?
  - How many tasks to split?
  - How to ensure that subtasks of a split task do not execute in parallel?
- **How to find the guarantee bound for given task assignment algorithm?**

Dual-Priority Scheduling
(uniprocessor)

Some Results on Task Splitting

- ECRTS 2009, CMU: Utilization bound 65%
  - Unsorted version: 60%
  - Number of split tasks is (m-1)
  - A task can be splitted in (m-1) parts
- IPDPS 2009, CHALMERS (Our Work):
  - Utilization bound 55.2%
  - Number of split tasks is m/2
  - A task can be splitted in at most 2 parts
- RTA 2010, UPPSALA
  - (Sorting) Utilization bound 69.3%
  - Number of split tasks is (m-1)
  - A task can be splitted in (m-1)parts

Motivation for Dual-Priority

- RM is the optimal fixed-priority algorithm with guarantee bound 69.3%
  - Each task is assigned a fixed priority
- EDF is the optimal dynamic priority algorithm with guarantee bound 100%
  - Each job/instance has a fixed-priority,
  - Different instances of the same task may have different priority

Motivation for Dual-Priority

- In EDF, the instances of a task can have n differnt priorities
  - Sometime priority level 1, sometime priority level 2, … Sometime priority level n
- In RM, all the instances of a task have exactly one unique priority
  - **Problem:** How can we introduce minimum dynamic-priority behaviour such that higher utilization bound is possible?

Dual-Priority Scheduling (EXAMPLE)

	C	T	U
τ_1	3	6	50%
τ_2	2	8	25%
τ_3	3	12	25%

- Using RM scheduling on uniprocessor, the task set is not schedulable

$\tau_{1,1}$			$\tau_{2,1}$		$\tau_{3,1}$	$\tau_{1,2}$			$\tau_{2,2}$		$\tau_{3,1}$
1	2	3	4	5	6	7	8	9	10	11	12

- The first instant of τ_3 misses its deadline at t=12

Dual Priority Scheduling

- Where is the problem ?

$\tau_{1,1}$			$\tau_{2,1}$		$\tau_{3,1}$	$\tau_{1,2}$			$\tau_{2,2}$		$\tau_{3,1}$
1	2	3	4	5	6	7	8	9	10	11	12

	C	T
τ_1	3	6
τ_2	2	8
τ_3	3	12

  - The second instance of task τ_2 can be delayed to allow the first instance of task τ_3 to complete before deadline
  - How to do it?
    - We can promote the priority of task τ_3 over other tasks at the beginning of time instant 11.

Dual Priority Scheduling

- New Priority and Promotion Point

	C	T	U	Non-Promoted Priority	Promoted Priority	When to promote?
τ_1	3	6	50%	2		
τ_2	2	8	25%	3		
τ_3	3	12	25%	4	1	11

Promotion point

Promotion point

$\tau_{1,1}$			$\tau_{2,1}$		$\tau_{3,1}$	$\tau_{1,2}$			$\tau_{2,2}$	$\tau_{3,1}$		$\tau_{1,3}$			$\tau_{2,2}$	$\tau_{2,3}$		$\tau_{1,4}$			$\tau_{3,2}$		
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

Dual Priority Scheduling

- Research Questions (a potential MS thesis work):
  - What is the **priority ordering** before and after promotion?
    - Possibly RM priority: before (n+1, … 2n) and after (1, … n)
  - How the **promotion points** have to be calculated for each task?
    - **Heuristic:** Start with promotion point equal to the deadline and then decrease it if not successful.
  - OPEN PROBELM: Does dual-priority scheduling have 100% **utilization bound**?
    - We did a lot of simulation and get YES answer for all.

Mixed-Criticality Systems

Mixed-Criticality System

- An active research area in Cyber-physical systems
- Many **safety-critical** systems are considering integrating multiple functionalities on a single platform (multicore)
  - hosting functionalities with **multiple criticality levels**
- The design is often subjected to **certification requirements** by **certification authority** (CA)
  - e.g., FAA or EASA for avionics

The Challenge

- The certification authority (CA) is very pessimistic in comparison to the system designer
- The CA is only concerned about the correctness of the ***safety-critical part***
- The system designer is concerned about the correctness of the ***entire system***
- **Challenge:** Coming up with a scheduling strategy that satisfies both the CA and the system designer

Current Research on MC

- Consider a particular aspect of the run-time behavior of the system: the **Worst-Case Execution Time** (WCET) of pieces of code
- The CA assumes ***high*** value for WCET
- The system designer assumes relatively ***lower*** value for WCET

Example

- Consider uniprocessor system
- Fixed-priority scheduling
- Three jobs J1, J2, and J3
- All are released at time zero

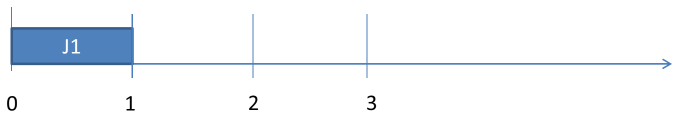
Jobs	Critical?	WCET (CA)	WCET(Designer)	Deadline
J1	NO	-	1	2
J2	YES	1.5	1	3.5
J3	YES	1.5	1	3.5

- Dual-Criticality Systems

Traditional Fixed-Priority Schedule

Jobs	Critical?	WCET (CA)	WCET(Designer)	Deadline
J1	NO	-	1	2
J2	YES	1.5	1	3.5
J3	YES	1.5	1	3.5

- If J1 is the highest priority task, then

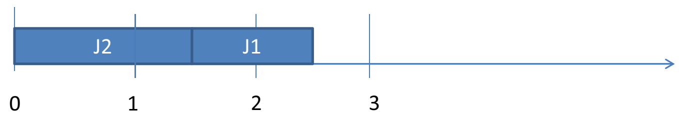


one of J2 or J3 misses its deadline.

Traditional Fixed-Priority Schedule

Jobs	Critical?	WCET (CA)	WCET(Designer)	Deadline
J1	NO	-	1	2
J2	YES	1.5	1	3.5
J3	YES	1.5	1	3.5

- If J1 is the medium priority task, then

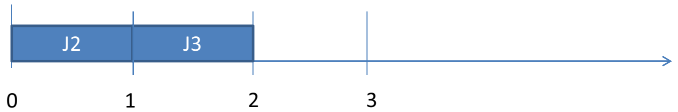


J3 misses its deadline

Traditional Fixed-Priority Schedule

Jobs	Critical?	WCET (CA)	WCET(Designer)	Deadline
J1	NO	-	1	2
J2	YES	1.5	1	3.5
J3	YES	1.5	1	3.5

- If J1 is the lowest priority task, then



Job J1 misses its deadline even if both J2 and J3 executes for 1 time unit.

Traditional Fixed-Priority Schedule

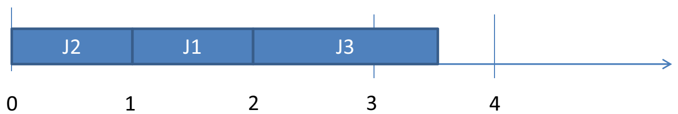
Jobs	Critical?	WCET (CA)	WCET(Designer)	Deadline
J1	NO	-	1	2
J2	YES	1.5	1	3.5
J3	YES	1.5	1	3.5

- Job J2 and J3 are schedulable if they are given the highest two priority levels
  - But J1 misses its deadline even if J2 and J3 execute for only 1 time unit
- Traditional Fixed-priority scheduling is not suitable to satisfy both the system designer and the CA.

A New Scheduling Scheme

Jobs	Critical?	WCET (CA)	WCET(Designer)	Deadline
J1	NO	-	1	2
J2	YES	1.5	1	3.5
J3	YES	1.5	1	3.5

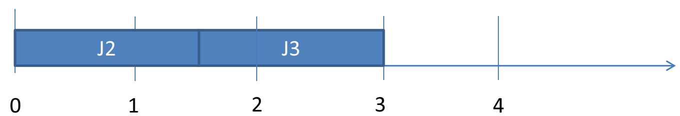
- Execute J2 over [0,1). If J2 completes by 1, then execute J1 and then J3



A New Scheduling Scheme

Jobs	Critical?	WCET (CA)	WCET(Designer)	Deadline
J1	NO	-	1	2
J2	YES	1.5	1	3.5
J3	YES	1.5	1	3.5

- If J2 does not complete by 1, then **drop** J2 and execute J2 over [1,1.5) and then J3 over [1.5,3).



A New Scheduling Scheme

Jobs	Critical?	WCET (CA)	WCET(Designer)	Deadline
J1	NO	-	1	2
J2	YES	1.5	1	3.5
J3	YES	1.5	1	3.5

- **Priority Assignment:** Assign the highest priority to J2, medium priority to J1 and the lowest priority to J3.
- **Dispatching:**
  - Execute J2 within [0,1).
  - If J2 completes, then execute J1 within [1,2) and J3 within [2,3) or [2,3.5)
  - If J2 does not complete, **drop J1**. Execute J2 for additional [1,1.5) and J2 within [1.5,3).

Conclusion

- There is a **gap** between 38% and 50% guarantee bound for global fixed-priority scheduling.
- The **optimal priority assignment** for global fixed-priority scheduling is still unknown.
- The **maximum achievable guarantee** bound for task-splitting with fixed-priority is not known.
- Dual-priority scheduling is very useful for industry, e.g, in CAN, if the **utilization bound** is 100%.
- Analysis for **certifiable mixed-criticality** systems on multiprocessors needs to be developed.

Mixed-Criticality Sporadic Tasks Scheduling on Multiprocessor

- **Each task is recurrent**
  - Three parameters (WCET, Deadline, Period)
- **Priority assignment**
  - How to assign fixed-priorities to the tasks?
- **Schedulability analysis and test**
  - How can we guarantee in offline that a MC task set is schedulable (satisfies both CA and the designer)?
- **Multiple criticality levels**
  - How to deal with multiple criticality levels?