Topics on Multiprocessor Scheduling

Cláudio Maia, Eduardo Tovar

MESCC

December 21, 2021

Table of contents

Task Characterization

- Workload is (typically) being generated by a finite collection of **recurrent** (and independent) tasks or processes
- A **task** generates an unbounded sequence of jobs
- A **job** is a piece of code that is executed by a given core
- Tasks can be characterized according to when their jobs are released
  - **Periodic**: Each job arrives at a constant inter-arrival time
  - **Sporadic**: There is a minimum inter-arrival time between successive jobs, but there is no upper bound on this value
  - **Aperiodic**: Jobs arrive in the system at any given time
- Each job is characterized by an **Arrival/Release Time**, a **Worst-Case Execution Time (WCET)** and a **Deadline**

Task models

- **Liu and Layland**: Tasks are characterized by two parameters: WCET C_i and period T_i ($\tau_i = (C_i, T_i)$)
- **Sporadic Task Model**: Generalization of the previous model, with the addition of the relative deadline D_i ($\tau_i = (C_i, D_i, T_i)$)
- Relation between relative deadline and period
  - **Implicit deadline**: The relative deadline of each task is equal to the task's period: $D_i = T_i, \forall \tau_i \in \tau$
  - **Constrained deadline**: The relative deadline of each task is no larger than the task's period: $D_i \leq T_i, \forall \tau_i \in \tau$
  - **Arbitrary deadline**: The relative deadlines do not have to satisfy any constraint with respect to the task's period
- **Utilization**: Ratio of the task's WCET to its period; **Density**: ratio of the task's WCET to the smaller of its period and relative deadline

Feasibility and Schedulability

- **Feasibility**: For a given platform, there exist schedules that meet all timing constraints for all the collections of jobs that could legally be generated by the given task system
- In hard real-time systems, it is not sufficient to know that a task system is feasible; **schedulability** is also needed
- **Schedulability**: For a given algorithm and platform, a task system is said to be schedulable if the algorithm meets all deadlines when scheduling each of the potentially infinite different collections of jobs that could be generated by the task system. A **schedulability test** accepts as input the specifications of a the task system and a multiprocessor platform, and determines whether the task system is schedulable. Can be **exact** or **sufficient**

Priority

- Priority deals with the question "Which job or set of jobs should be executed at a given time?"
- Depending on how the priorities are assigned, scheduling algorithms can be classified as:
  - **Fixed task priority (FTP)**: each task is assigned a unique priority, and each job inherits the priority of the task that generated it, e.g., Rate Monotonic
  - **Fixed job priority (FJP)**: different jobs of the same task may be assigned different priorities, but the priority of a job may not change after assignment, e.g., Earliest Deadline First
  - **Dynamic priority (DP)**: there are no restrictions on how priorities are assigned to jobs, e.g., Least Laxity First

Synchronous Release and Utilization Bound

- A synchronous arrival sequence of jobs occurs if all tasks in the task system release a job at the same instant in time and subsequent jobs are generated as rapidly as permitted
- Relevant in uniprocessor scheduling because there are many situations for which it is known that this moment represents the worst-case behavior for a sporadic task system
- If an algorithm can schedule a task set considering this synchronous arrival sequence then it is able to schedule all other possible releases of jobs that can legally be generated by the task system
- **Utilization bound**: The largest number U such that all task systems with utilization $\leq$ U (and each task having utilization ≤ 1) is successfully scheduled by an algorithm.

Multiprocessor Scheduling

Motivation

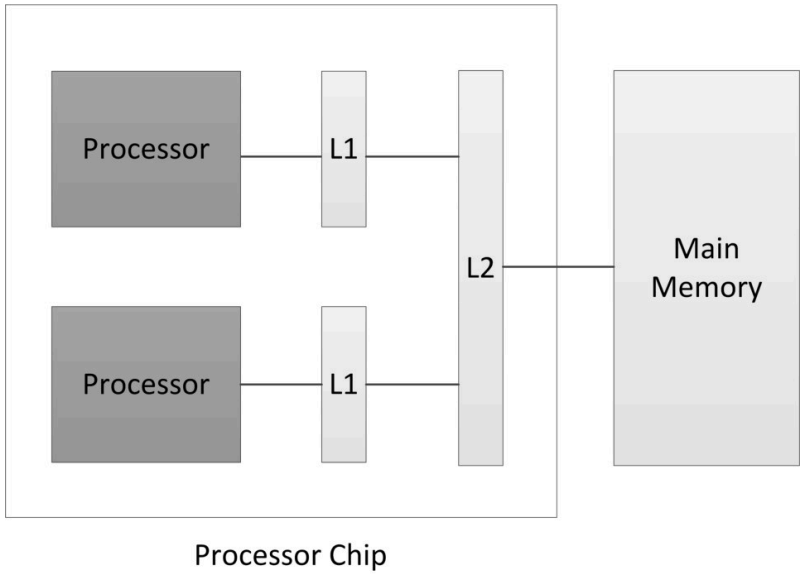
- Moore's Law: the density of transistors on a chip doubles every 24 months
  - Higher number of logic gates that can fit on a chip
  - Higher working frequency
- But, is this scalable?
  - Decrease in size leads to an increase in power dissipation (consumption)
  - Increase in power consumption leads to an increase in heat
    - If not properly dissipated can cause damage to the circuits
    - There are limits in the capabilities of current cooling systems

Solution

Build chips with multiple cores working at lower clock frequencies

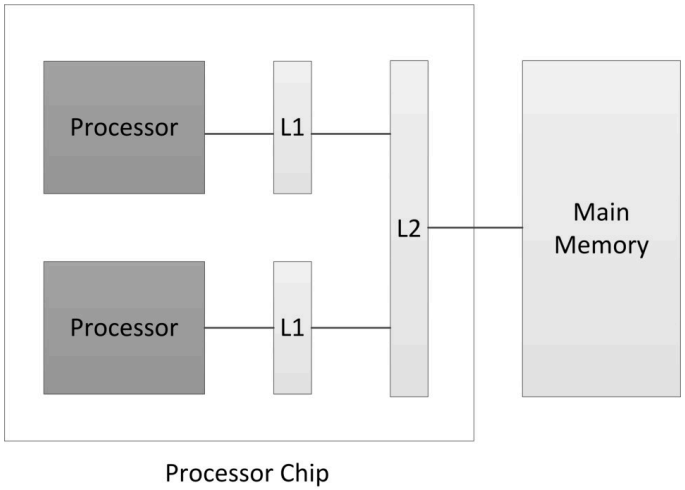
Multicore Chips

Simple Multicore System



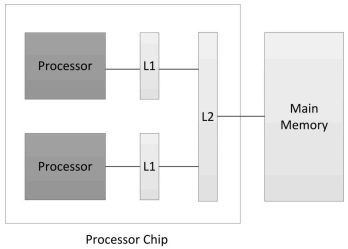
What are the scheduling challenges that a designer has to deal with when designing a real time application running on such systems?

Scheduling challenges



- How to split the application into parallel segments so that they can be executed simultaneously?
- How to allocate the different segments into the different processor cores?

Scheduling challenges cont'd



How about the shared resources (i.e., memory and I/O devices)?

Single core systems vs Multicore systems

- Sequential execution implies that the access to physical resources is serialized
- With multicore processors resources are shared. This may lead to resource contention due to simultaneous access. The side effect is that it may lead to blocking, increased response times and **deviations from WCET**

Scheduling challenges cont'd

The performance of an application can vary significantly depending on how tasks are allocated and scheduled on a multicore platform

- How to allocate and schedule concurrent tasks in a multicore platform?
- How to analyze real-time applications to guarantee timing constraints while accounting for communication delays and interference?
- How to reduce interference?
- How to optimize resource allocation?
- Assuming intra-task parallelism exists, how to deal with it?

Classification of multiprocessor platforms

- **Identical**: Each processor in the platform has the same computing capabilities as every other processor. Example: Symmetric Multiprocessor systems
- **Uniform (or related)**: Each processor has its own computing capacity s. Thus, a job that executes on a processor of computing capacity s for t time units completes $s * t$ units of execution
- **Unrelated**: Each job may have a different execution rate on each processor. Thus, if a job has an execution rate r specified for a processor then executing it on that processor for t time units completes $r * t$ units of execution. Example: Heterogeneous Chips with multiple CPUs, DSPs and GPUs

Preemption and Migration

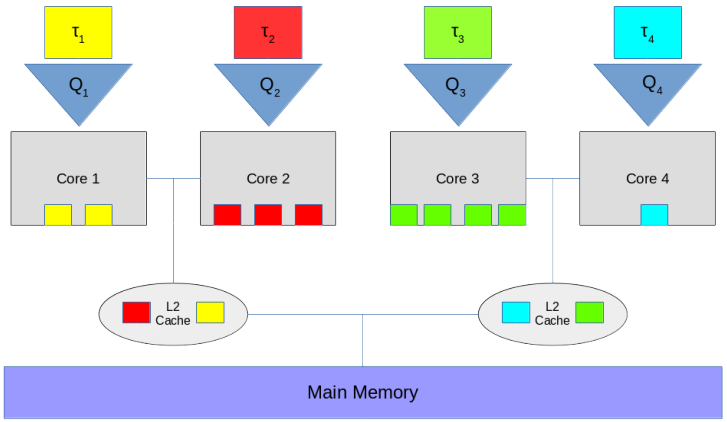
- **Preemption** property that can used either in single/multi processor systems.
  - **Preemptive Scheduling**: A job executing on a given processor may be interrupted by the scheduler and have its execution resumed at a later point in time
  - **Non-Preemptive Scheduling**: A job executing on a given processor cannot be interrupted by the scheduler, i.e., it executes until completion without any interruption.
- **Migration**: property that only exists in multiprocessor or multicore systems. It concerns to where a given task is allowed to execute in such systems.

Types of Scheduling Algorithms

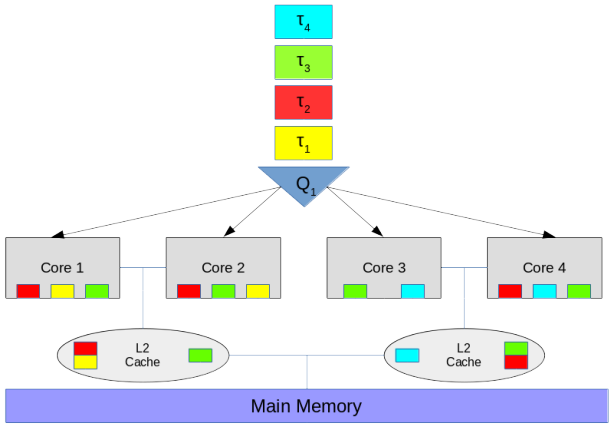
- **Global scheduling**: allows any job to execute on any given processor. **Job Migrations are allowed.**
- **Partitioned Scheduling**: A task is mapped to a particular processor and is only allowed to execute its jobs on the processor to which it has been mapped. **Job Migrations are not allowed.**
- **Clustered Scheduling**: The processors of a given platform are partitioned in to clusters and each task is mapped to one of those clusters. **Job Migrations are allowed within a cluster.**
- **Semi-partitioned**: A few tasks are allowed to migrate while others are mapped to processors (not allowed to migrate). Similar restrictions may be applied by different algorithms.

Types of Scheduling Algorithms

Partitioned Scheduling



Global Scheduling



Types of Scheduling Algorithms

Partitioned Scheduling

- + Each processor is a single processor system and all theory of single processor scheduling applies to each processor
- - Assigning tasks to processors can be complicated and inefficient if done carelessly which leads to weak performance

Global Scheduling

- + Better utilization and overall system performance
- - Migration

It is not possible to compare **Partitioned scheduling** approaches with **Global scheduling approaches**. There are some task sets that are feasible under partitioned approaches and unfeasible under global approaches and vice-versa.

Partitioned Scheduling

- For a given sporadic task set τ and an m-processor platform, the tasks in the task set τ are partitioned into m disjoint subsets, with each subset assigned to execute on a different processor
- Task partitioning is equivalent to the **bin-packing problem** and consequently, the problem is nondeterministic polynomial time (NP)-hard in the strong sense
- Unless P=NP, it is unlikely to be possible to design partitioning algorithms that have both optimal resource utilization and efficient. Thus, approximation algorithms are used to solve this problem.

Bin Packing

Bin packing decision problem

- Given a number of bins B, a bin capacity C, and a set of n items, $x_1, x_2, ..., x_n$, with sizes $s_1, s_2, ..., s_n$, does a packing of items $x_1, x_2, ..., x_n$ exist such that it fits into B bins?

Bin packing optimization problem

- Given a bin capacity C and a set of n items, $x_1, x_2, ..., x_n$, with sizes $s_1, s_2, ..., s_n$, assign each item to a bin such that the number of bins is minimized.

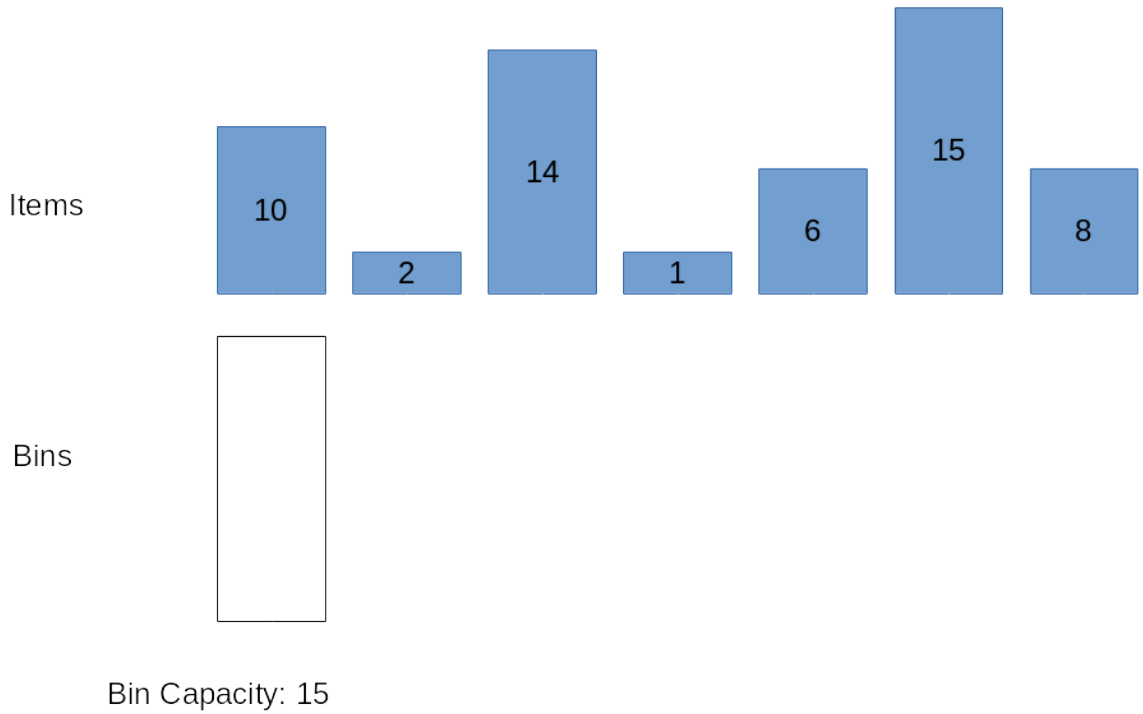
Bin Packing

- Simple approach to solve the partitioning problem:
  - Specify the order of the tasks and processors to be considered in the partitioning
  - After selecting the task, follow the chosen order for the processors in order to allocate it
  - A task is successfully allocated on a processor if it fits on that processor
- SUCCESS: All tasks were allocated; FAILURE: otherwise
- In partitioned scheduling, fit means that after allocating a task to the processor, the task utilization does not exceed the processor capacity
- Current Processor capacity = sum of the utilization of all tasks that were allocated in that processor

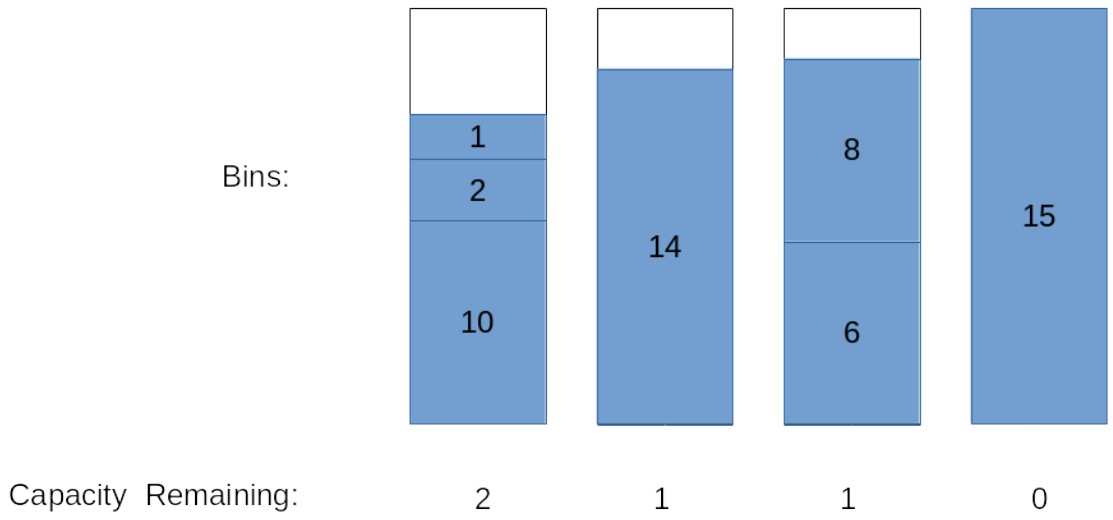
Common Bin Packing Heuristics

- **First-fit (FF)**: The processors are considered ordered in some manner and the task is assigned to the first processor on which it fits
- **Worst-fit (WF)**: A given task is assigned to the processor with the maximum remaining capacity
- **Best-fit (BF)**: A given task is assigned to the processor with the minimum remaining capacity (but still has enough capacity to fit its weight)
- Tasks can be ordered in an increasing or decreasing order

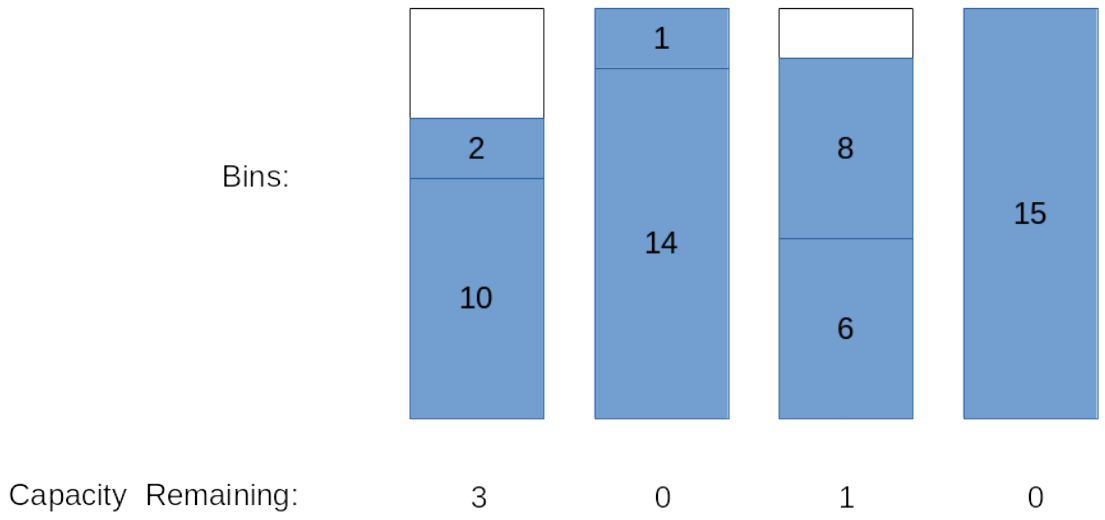
Bin Packing Example



Bin Packing Example - First Fit



Bin Packing Example - Best Fit

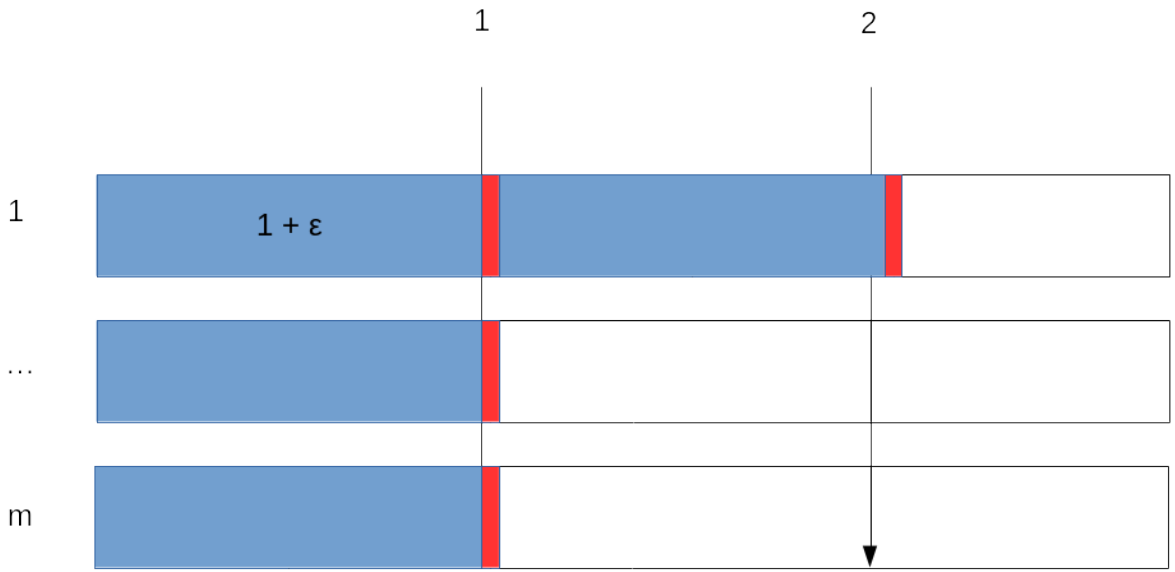


Partitioned Scheduling Upper Utilization Bound

There exist task sets with utilizations arbitrarily close to $\frac{m+1}{2}$ that cannot be partitioned.

- $m + 1$ tasks
- Each task has the following parameters:
- $T_i = 2$, $WCET = 1 + \epsilon$, $U = \frac{(1+\epsilon)}{2}$
- Total taskset utilization: $(m + 1) \cdot \frac{(1+\epsilon)}{2}$

Partitioned Scheduling Upper Utilization Bound



Global Scheduling

- While in partitioned scheduling migrations are forbidden, in global scheduling, tasks are allowed to migrate between the different cores
- Considering global scheduling, Horn established the following result:

Horn

Any implicit-deadline sporadic task set τ satisfying $\sum(U_\tau) \leq m$ and $\max(U_\tau) \leq 1$ is feasible on a platform comprised of m unit-capacity processors.

Global Approaches

Global EDF

- At run-time, assign priorities according to the deadline of the active jobs: closer the deadline, higher the priority
- At any time, execute the m highest priority jobs on the m processors.

Global Rate Monotonic

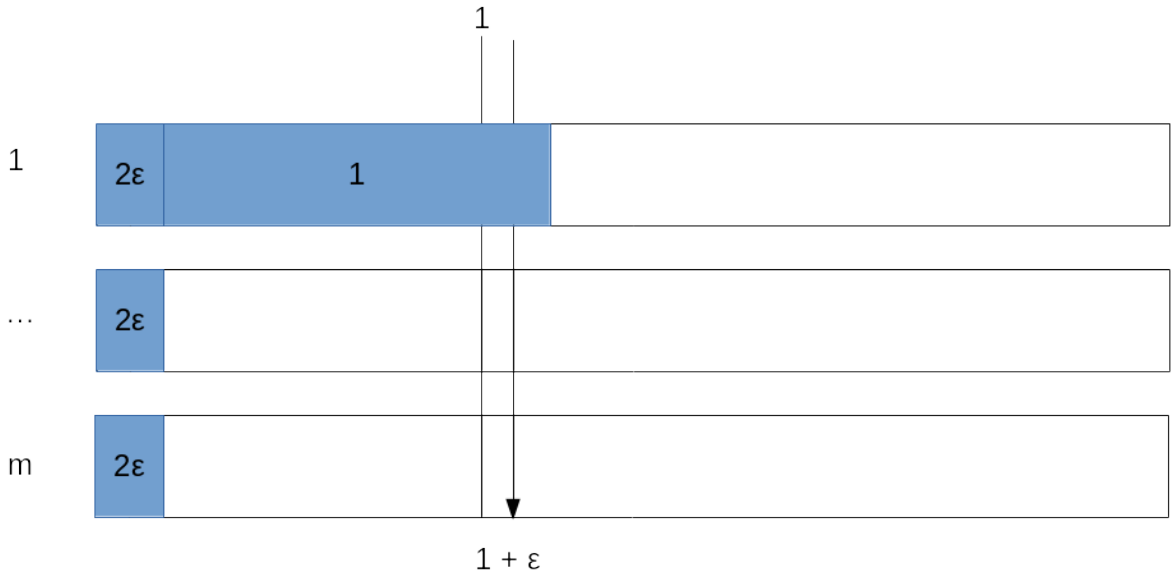
- Statically assign priorities according to the period of the tasks: the smaller the period, the higher the priority
- At any time, execute the m highest priority jobs on the m processors.

Global Scheduling - Dhall's Effect

There exist task sets with utilizations closer to 1 that cannot be scheduled with Global RM or Global EDF.

- $m + 1$ tasks
- First m tasks have the following parameters:
- $T = 1$, $WCET = 2 \cdot \epsilon$
- $m + 1$ task has the following parameters: $T = 1 + \epsilon$, $C = 1$
- What is the total taskset utilization?

Global Scheduling - Dhall's Effect



Unknown Critical Instant

Exercise: Schedule the above task set with synchronous release, $\phi = 0$ or using ϕ as the offset parameter using G-EDF.

- $\tau_i = (\phi_i, C_i, T_i)$
- $\tau_1 = (2, 6, 10)$
- $\tau_2 = (3, 2, 9)$
- $\tau_3 = (2, 1, 5)$
- $\tau_4 = (3, 3, 9)$
- $\tau_5 = (0, 7, 12)$
- Number of processors $= 2$

Summary

- There are different types of scheduling approaches to schedule a set of tasks upon a multiprocessor/multicore platform
- For each type of approach, several algorithms may be found in the literature
- An algorithm should be selected/designed as a function of the parameters of the given tasks
- Recent trends include Mixed Criticality Tasks; Tasks with intra-task parallelism, etc.
- Each year several scheduling algorithms are published in conferences such as RTSS, ECRTS, RTAS, RTCSA, ...; so feel free to look into their program to find novel and interesting ideas
- Feel free to discuss them with people at CISTER

References

- John Carpenter et al., "A Categorization of Real-Time Multiprocessor Scheduling Problems and Algorithms"
- Robert Davis and Alan Burns, "A survey of hard real-time scheduling for multiprocessor systems"
- Sanjoy Baruah, Marko Bertogna and Giorgio Buttazzo, "Multiprocessor Scheduling for Real-Time Systems", **Book used as a reference to extract information to write these slides.**