# Scheduling Basics

## Real-Time Operative Systems Course

Paulo Pedreiras, DETI/UA/IT

October 10, 2022

1 Preliminaries

2 Basic concepts

3 Scheduling Algorithms

4 Static Cyclic Scheduling

# Last lecture

- Computation models and Real-Time Kernels and Operating Systems
- Computation models:
  - Tasks with specific temporal constraints;
  - The Event- and Time-Trigger paradigms
- Real-Time OSs and kernels:
  - General architecture
  - Task states
  - Basic components

# Agenda for today

Scheduling basics

- Task scheduling - basic concepts and taxonomy
- Basic scheduling techniques
- Static cyclic scheduling

1 Preliminaries

2 Basic concepts

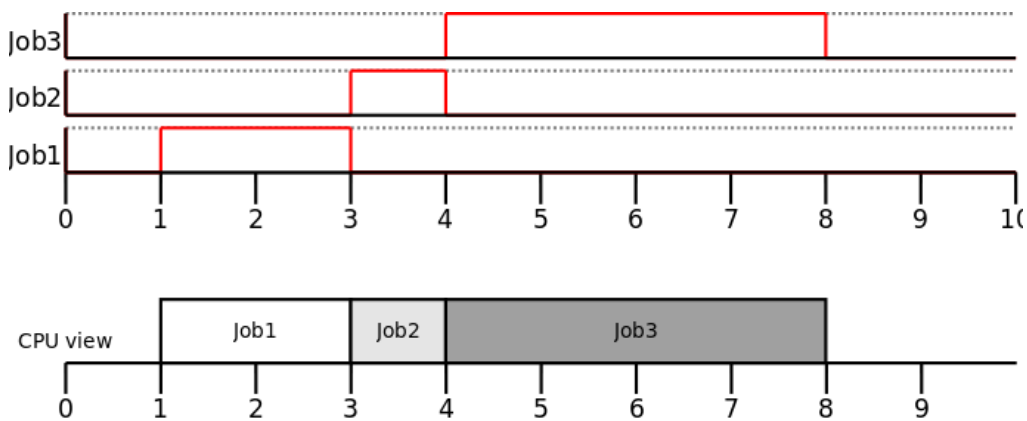
3 Scheduling Algorithms

4 Static Cyclic Scheduling

# Scheduling Definition

Task scheduling (also applies to messages, with due adaptations)

- Sequence of task executions (jobs) in one or more processors
- Application of $\mathbb{R}^+$ (time) in $\mathbb{N}_0$ (task set), assigning to each time instant "t" a task/job "i" that executes in that time instant
  $$\sigma(t) : \mathbb{R}^+ \rightarrow \mathbb{N}_0$$
  $i = \sigma(t), t \in \mathbb{R}^+$, i=0 means that the processor is idle
- $\sigma(t)$ is a step function, which can be expressed e.g. as a Gantt graph



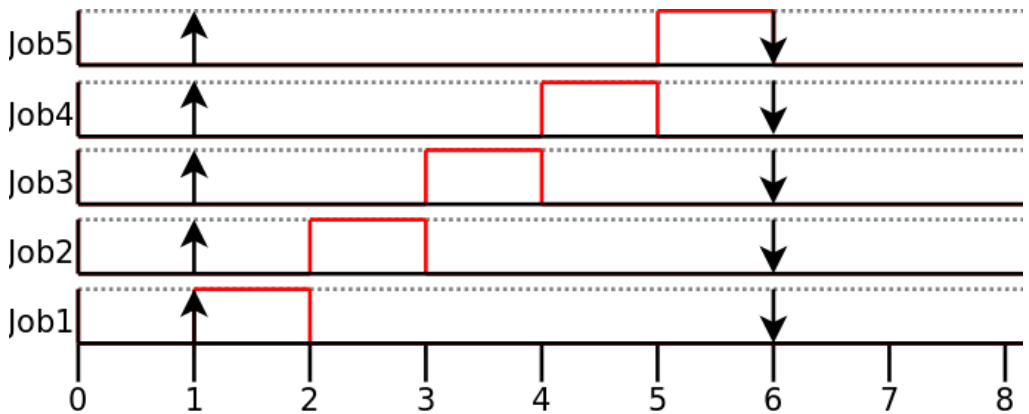
# Scheduling Definition

- A schedule is called **feasible** if it fulfills all the task requirements
  - temporal, non-preemption, shared resources, precedences, ...
- A task set is called **schedulable** if there is at least one feasible schedule for that task set

# The scheduling problem: easy to formulate, but hard to solve!

- Given:
  - A task set
  - Requirements of the tasks (or cost function)
- Find a time attribution of processor(s) to tasks so that:
  - Tasks are completely executed, and
  - Meet they requirements (or minimize the cost function)

E.g. $J = J_i(C_i = 1, a_i = 1, D_i = 5), i = \{1..5\}$



# Scheduling problem

Exercise:

- Build a Gantt diagram for the execution of the following periodic tasks, admitting $D_i = T_i$ and no preemption.
  - $\tau = \{(1, 5)(6, 10)\}$
- Is the execution order important? Why?

1 Preliminaries

2 Basic concepts

3 Scheduling Algorithms

4 Static Cyclic Scheduling

# Scheduling algorithms

- A **scheduling algorithm** is a method for solving the scheduling problem.
  - Note: don't confuse scheduling algorithm (the process/method) with schedule (the result)
- Classification of scheduling algorithms:
  - Preemptive vs non-preemptive
  - Static vs dynamic (priorities)
  - Off-line vs on-line
  - Optimal vs sub-optimal
  - With strict guarantees vs best effort

# A short note on temporal complexity

- Measurement of the **growth** of the execution time of an algorithm as a function of the problem size ( e.g. the number of elements of a vector, the number of tasks of a real-time system)
- Expressed via the $O()$ operator (big O notation)
- $O()$ arithmetic, n=problem dimension, k=constant
  - $O(k) = O(1)$
  - $O(kn) = O(n)$
  - $O(k_1 \cdot n^m + k_2 \cdot n^{m-1} + ... + k_{m+1}) = O(n^m)$

Compl. $= O(N)$

```
for (k=0;k<N;k++)
    a[k]=0;
```

Compl. $= O(N^2)$

```
for (k=0;k<N-1;k++)
    for (m=k;m<N;m++)
        if a[k]<a[m]
            swap(a[k],a[m]);
```

Compl. $= O(N^N)$

Computation of the permutations of a set $A = a_i, i = 1..N$

# Basic algorithms

EDD - Earliest Due Date (Jackson, 1955)

- Single instance tasks fired synchronously:$J = Ji(Ci, (ai = 0, )Di), i = 1..n$
- Executing the tasks by non-decreasing deadlines minimizes the maximum lateness ($L_{max}(J) = max_i(f_i - d_i)$)
- Complexity: O(n.log(n))

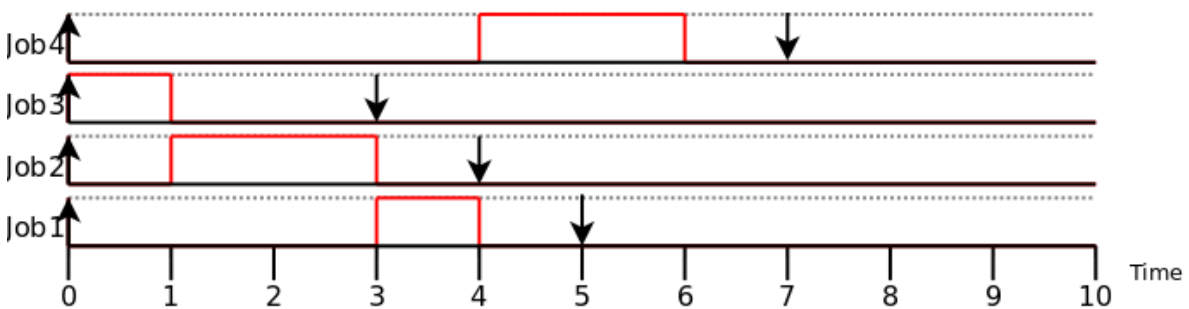
E.g. $J = \{J_1(1,5), J_2(2,4), J_3(1,3), J_4(2,7)\}$
Determine the maximum lateness!

# Basic algorithms

EDD - Earliest Due Date (Jackson, 1955)

- Single instance tasks fired synchronously: $J = Ji(Ci, (ai = 0, )Di), i = 1..n$
- Executing the tasks by non-decreasing deadlines minimizes the maximum lateness ($L_{max}(J) = max_i(f_i - d_i)$)
- Complexity: O(n.log(n))

E.g. $J = \{J_1(1,5), J_2(2,4), J_3(1,3), J_4(2,7)\}$
Determine the maximum lateness!




Lmax,EDD(J) = -1

# Basic algorithms

EDF - Earliest Deadline First (Liu and Layland, 1973; Horn, 1974)

- Single instance or periodic, asynchronous arrivals, preemptive: $J = J_i(C_i, a_i, D_i), i = 1..n$
- Always executing the task with shorter absolute deadline minimizes the maximum latency $L_{max}(J) = max_i(f_i - d_i)$
- Complexity: $O(n.log(n))$, **Optimal** among all scheduling algorithms of this class

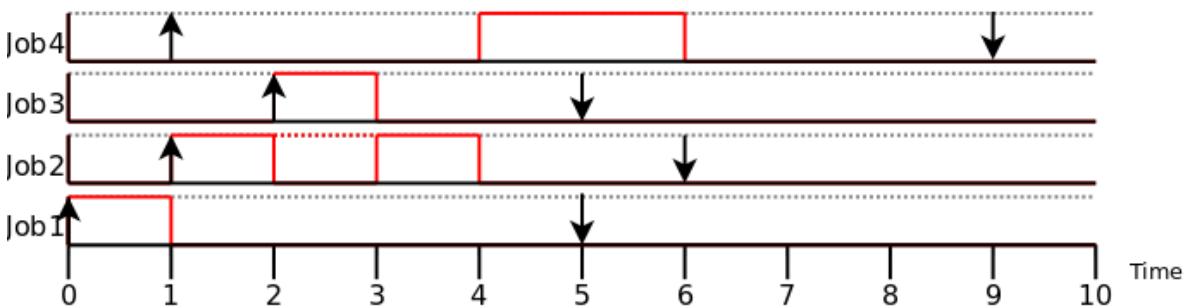
E.g. $J = \{J_1(1, 0, 5), J_2(2, 1, 5), J_3(1, 2, 3), J_4(2, 1, 8)\}$
Determine the maximum lateness!

# Basic algorithms

EDF - Earliest Deadline First (Liu and Layland, 1973; Horn, 1974)

- Single instance or periodic, asynchronous arrivals, preemptive: $J = J_i(C_i, a_i, D_i), i = 1..n$
- Always executing the task with shorter absolute deadline minimizes the maximum latency $L_{max}(J) = max_i(f_i - d_i)$
- Complexity: $O(n.log(n))$, **Optimal** among all scheduling algorithms of this class

E.g. $J = \{J_1(1, 0, 5), J_2(2, 1, 5), J_3(1, 2, 3), J_4(2, 1, 8)\}$
Determine the maximum lateness!




Lmax,EDD(J) = -2

# Basic algorithms

BB – Branch and Bound (Bratley, 1971)

- Single instance or periodic tasks, asynchronous arrivals, non-preemptive: $J = J_i(C_i, a_i, D_i), i = 1..n$
- Based on building an exhaustive search in the permutation tree space, finding all possible execution sequences:
- Complexity: O(n!)

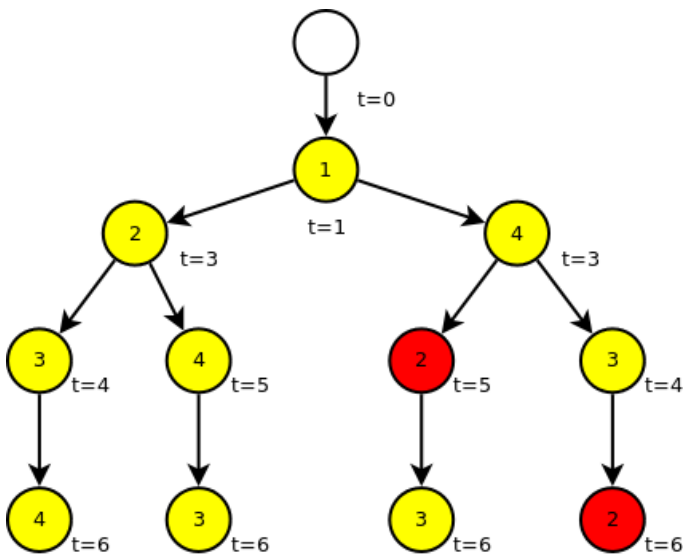
E.g. $J = \{J_1(1, 0, 5), J_2(2, 1, 3), J_3(1, 2, 4), J_4(2, 1, 7)\}$

# Basic algorithms

BB – Branch and Bound (Bratley, 1971)

- Single instance or periodic tasks, asynchronous arrivals, non-preemptive: $J = J_i(C_i, a_i, D_i), i = 1..n$
- Based on building an exhaustive search in the permutation tree space, finding all possible execution sequences:
- Complexity: O(n!)

E.g. $J = \{J_1(1, 0, 5), J_2(2, 1, 3), J_3(1, 2, 4), J_4(2, 1, 7)\}$



# Periodic task scheduling

Periodic tasks

- The release/activation instants are known **a priori**
- $\Gamma = \{\tau_i(C_i, \phi_i, T_i, D_i)\}, i = \{1...n\}$
- $a_{i,k} = \phi_i + (k-1) \cdot T_i, k = 1, 2, ...$

Thus, in this case the schedule can be built:

Online Tasks to execute are selected as they are released and finish, during normal system operation (addressed in next class)

Offline The task execution order is computed before the system enters in normal operation and stored in a table, which is used at runtime time to execute the tasks (static cyclic scheduling).

1 Preliminaries

2 Basic concepts

3 Scheduling Algorithms

4 Static Cyclic Scheduling

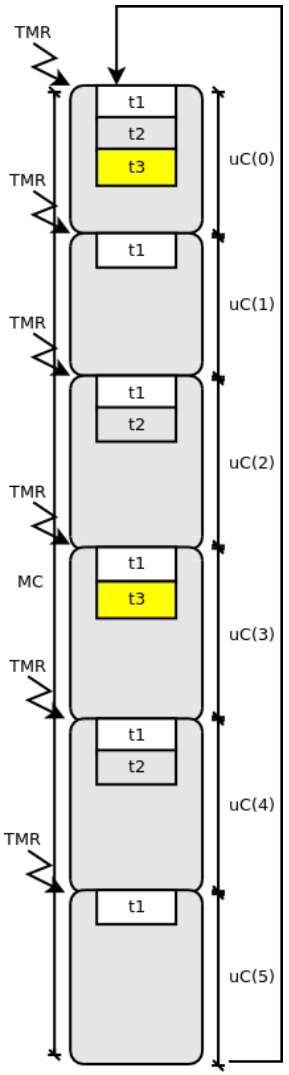
# Static cyclic scheduling

- The table is organized in micro-cycles ($\mu C$) with a fixed duration. This way it is possible to release tasks periodically
- The micro-cycles are triggered by a Timer
- Scanning the whole table repeatedly generates a periodic pattern, called macro-cycle (MC)

$$\Gamma = \{\tau_i(C_i, \phi_i, T_i, D_i)\}, i = \{1...n\}$$
$$\mu C = GCD(T_i); MC = mCM(T_i)$$

Example:

- $\phi_i = 0$
- $T_1 = 5ms$; $T_2 = 10ms$; $T_3 = 15ms$



# Static cyclic scheduling

## Pros

- Very simple implementation (timer+table)
- Execution overhead very low (simple dispatcher)
- Permits complex optimizations (e.g. jitter reduction, check precedence constraints)

## Cons

- Doesn't scale (changes on the tasks may incur in massive changes on the table. In particular the table size may be prohibitively high)
- Sensitive to overloads, which may cause the "domino effect", i.e., sequence of consecutive tasks failing its deadlines due to a bad-behaving task.

# Static cyclic scheduling - Algorithm

How to build the table:

- Compute the micro and macro cycles ($\mu C$ and $MC$)
- Express the periods and phases of the tasks as an integer number of micro-cycles
- Compute the cycles where tasks are activated
- Using a suitable scheduling algorithm, determine the execution order of the ready tasks
- Check if all tasks scheduled for a give micro-cycle fit inside the cycle. Otherwise some of them have to be postponed for the following cycle(s)
- It may be necessary to break a task in several parts, so that that each one of them fits inside the respective micro-cycle

# Summary

- The concept of temporal complexity
- Definition of schedule and scheduling algorithm
- Some basic scheduling techniques (EDD, EDF, BB)
- The static cyclic scheduling technique

# Homework

Consider the following message set ( syntax (C,T), with D=T):

$$\Gamma = \{(1,5);(2,10);(3,10);(4,20)\}$$

- Compute the utilization of each task and the global utilization
- Compute the micro and macrocycle
- Build the scheduling table