

Readings

- Reach chapter 2 of Buttazzo (2011)¹
- Topics
 - Task schedule
 - Task preemption
 - Task timing
 - Metrics of scheduling algorithms



Task Scheduling Taxonomy

Kizito NKURIKIYEYU,
Ph.D.

¹Readings are based on Buttazzo, G. C. (2011). Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications (Real-Time Systems Series, 24) (3rd edition). Springer.

Kizito NKURIKIYEYU, Ph.D.

Task Scheduling Taxonomy

October 13, 2021

1 / 10

Schedule

Given a set of tasks $J = J_1, J_2, \dots, J_n$

- A **schedule** is an assignment of tasks to the processor, such that each task is executed until completion.
- A schedule can be defined as an **integer step function** (Equation (1))

$$\sigma : R \rightarrow N \quad (1)$$

where $\sigma(t)$ denotes the task which is executed at time t . If $\sigma(t) = 0$, then the processor is called idle

- If $\sigma(t)$ changes its value at some time, then the processor performs a **context switch**.
- Each interval, in which $\sigma(t)$ is constant is called a **time slice**.
- A **preemptive schedule** is a schedule in which the running task can be arbitrarily suspended at any time, to assign the CPU to another task according to a predefined scheduling policy.

Example—Task scheduling

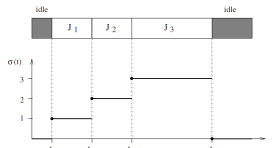


FIG 1. Schedule obtained by executing three tasks J_1 , J_2 , and J_3

- **context switch** at times t_1, t_2, t_3, t_4
- **time slice**—intervals $[t_i, t_{(i+1)})$ in which $\sigma(t)$ is constant

Preemption

Preemption is important for three reasons:

- 1 Tasks performing exception handling may need to preempt existing tasks so that responses to exceptions may be issued in a timely fashion.
- 2 When tasks have different levels of criticality (expressing task importance), preemption permits executing the most critical tasks, as soon as they arrive.
- 3 Preemptive scheduling typically allows higher efficiency, in the sense that it allows executing a real-time task sets with higher processor utilization.

Disadvantages:

- Preemption destroys program locality and introduces a runtime overhead
- This overhead increase the execution time of tasks.

Schedule and Timing

A real-time task τ_i can be characterized by the following parameters:

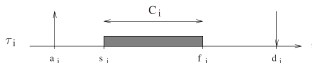


FIG 2. Typical parameters of a real-time task

- **feasible schedule**—if all task can be completed according to a set of specified constraints. Examples of constraints include:
 - Timing constraints—activation, period, deadline, jitter.
 - Precedence—order of execution between tasks.
 - Resources—synchronization for mutual exclusion.
- **schedulable tasks**—If there exists at least one algorithm

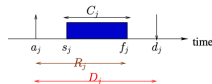
Schedule and Timing

- **arrival time a_i or release time r_i** is the time at which a task becomes ready for execution
- **computation time C_i** is the time necessary to the processor for executing the task without interruption.
- **relative deadline D_i** is the time length between the arrival time and the absolute deadline.
- **absolute deadline d_i** is the time at which a task should be completed. Note that from the above definitions, we have the relation in Equation (2)

$$d_i \geq r_i + C_i \quad (2)$$

Schedule and Timing

- **start time s_i** is the time at which a task starts its execution.
- **finishing time f_i** is the time at which a task finishes its execution.
- **response time R_i** is the time length at which the job finishes its execution after its arrival, which is $f_i - a_i$



- **lateness $L_i = f_i - d_i$** represents the delay of a task completion with respect to its deadline. If a task completes before the deadline, its lateness is negative.
- **tardiness or exceeding time $E = \max(0, L_i)$** is the time a task stays active after its deadline.
- **laxity or slack time $X_i = d_i - a_i - C_i$** is the maximum time a

Schedule and Timing

periodic task τ_i —infinite sequence of identical activities, called **instances or jobs**, that are regularly activated at a constant rate with **period** T_i . The activation time of the first instance is called **phase** ϕ

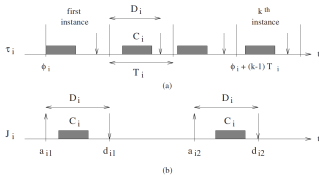
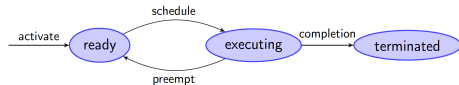


FIG 3. Sequence of instances for a periodic task (a) and an aperiodic job (b)

Scheduling Algorithm

scheduling algorithm—determines the order that jobs execute on the processor



- **preemptive algorithms**—the running task can be interrupted at any time to assign the processor to another active task, according to a predefined scheduling policy.
- **non preemptive algorithm**—a task, once started, is executed by the processor until completion.
- **static algorithms** are those in which scheduling decisions are based on fixed parameters, assigned to tasks before their activation.
- **dynamic algorithms** are those in which scheduling decisions

Metrics of scheduling algorithms

- Average response time, \bar{t}_r (Equation (3))

$$\bar{t}_r = \frac{1}{n} \sum_{i=1}^n (f_i - a_i) \quad (3)$$

- Total completion time, t_c (Equation (4))

$$t_c = \max(f_i) - \min(a_i) \quad (4)$$

- Maximum lateness, L_{max} (Equation (5))

$$L_{max} = \max(f_i - d_i) \quad (5)$$

- Maximum number of late tasks

$$N_{late} = \sum_{i=1}^n \text{miss}(f_i), \quad (6)$$

with $\text{miss}(f_i)$ defined as shown in Equation (7)

$$\text{miss}(f_i) = \begin{cases} 0, & \text{if } f_i \leq d_i. \end{cases} \quad (7)$$

The end