Classical scheduling algorithms

Tullio Facchinetti
<tullio.facchinetti@unipv.it>

24 novembre 2023

http://robot.unipv.it/toolleeo

The scheduling

the problem of resource allocation arose before the issues related with real-time scheduling

some common scheduling algorithms are:

- First Come First Served (FCFS)
- Shortest Job First (SJF)
- Round Robin (RR)

these algorithms do not work well under timing constraints

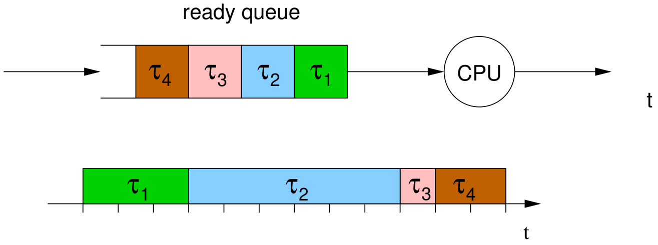
First Come First Served

the processor is assigned on the basis of
the task arrival time

features:

- non-preemptive
- dynamic (no assumptions regarding other parameters of tasks)
- online
- best effort

First Come First Served



- the scheduling pattern is determined by the task arrival time
- earlier tasks have higher priority
- tasks are inserted into the ready queue using a First In First Out policy (FIFO), without any further sorting

Response time

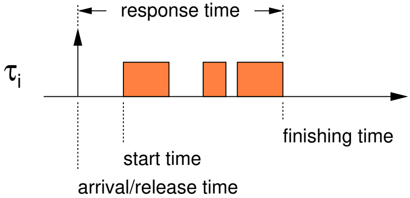
for non real-time algorithms, performance can be assessed by the response time

the response time R_i of the i-th task is

$$R_i = f_i - a_i$$

where

- f_i is the finishing time
- a_i is the arrival time



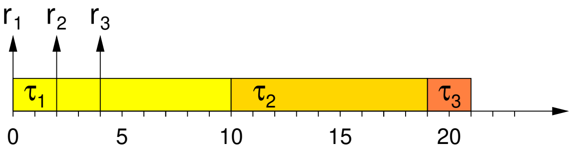
First Come First Served

the completion time of a task depends on:

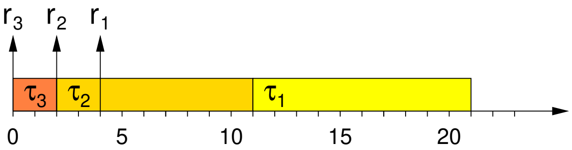
- its arrival time
- the duration of all earlier tasks

no a-priori guarantees on the task completion time (response time)

First Come First Served



$$R_1 = 10 - 0 = 10$$
$$R_2 = 19 - 2 = 17$$
$$R_3 = 21 - 4 = 17$$



$$R_1 = 21 - 4 = 17$$
$$R_2 = 11 - 2 = 9$$
$$R_3 = 2 - 0 = 2$$

the response time of a task depends on the order of arrival

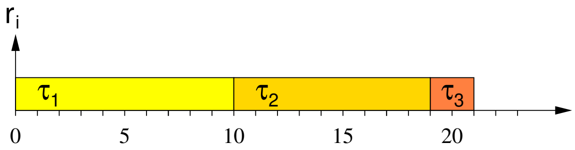
Shortest Job First (SJF)

higher priorities are assigned to tasks having shorter execution times (durations)

features:

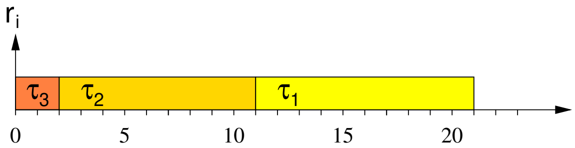
- both preemptive and non-preemptive version
- static (the duration is constant)
- can be both online and offline
- **minimizes the average response time**

Shortest Job First (SJF)



$R_1 = 10 - 0 = 10$
$R_2 = 19 - 0 = 19$
$R_3 = 21 - 0 = 21$
$\overline{R} = 16.667$

This schedule IS NOT the one produced by SJF



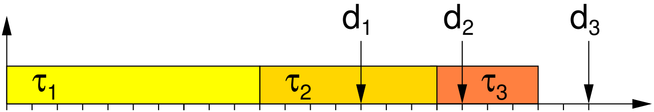
$R_1 = 21 - 0 = 21$
$R_2 = 11 - 0 = 11$
$R_3 = 2 - 0 = 2$
$\overline{R} = 11.333$

This schedule IS the one produced by SJF

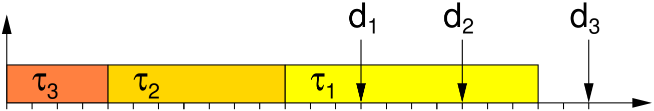
It can be *formally proved* that
the *average response time* is *minimized*.

Shortest Job First with temporal constraints

example of task set that is schedulable with an algorithm different from SJF



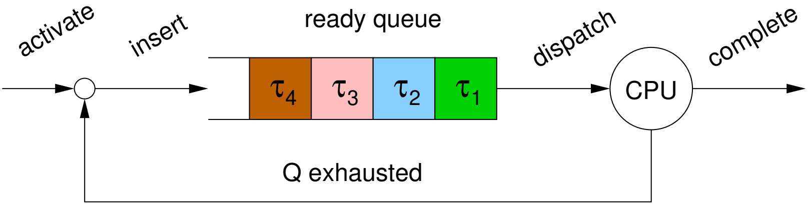
the same task set is not schedulable by SJF



SJF is not optimal (regarding the schedulability)

Round Robin (RR)

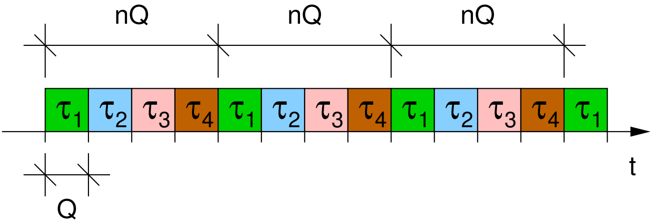
each task can not run continuously for more than an allotted time, called Q (time quantum)



- the ready queue is managed with FIFO policy (First In First Out)
- if a task exhausts its time quantum, it is interrupted and re-inserted into the ready queue

Round Robin (RR)

the Round Robin algorithm is behind the so-called time-sharing systems

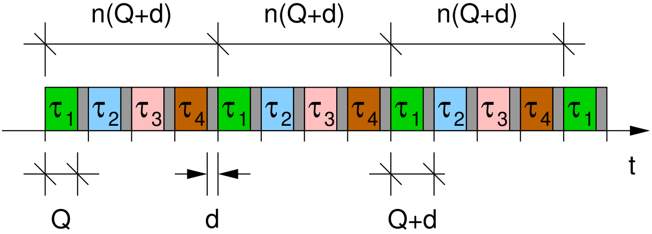


$$R_i \cong (nQ)\frac{C_i}{Q} = nC_i$$

the response time of each task is equal to that of the same task executed on a processor n times slower

Round Robin (RR)

- if $\max(C_i) \leq Q$ then RR $\equiv$ FCFS
- if the scheduling overhead d due to the context switch is taken into account:



$$R_i \cong n(Q+d)\frac{C_i}{Q} = nC_i\frac{Q+d}{Q}$$

Earliest Due Date (EDD)

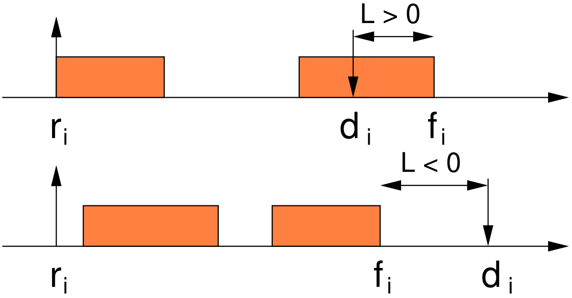
the task having the shortest relative deadline
becomes the highest priority task

assumptions:

- simultaneous arrival of all tasks
- fixed priority (relative deadlines are known in advance)
- preemption is not an issue (simultaneous arrival)
- **minimizes the maximum lateness** L_{max}

Lateness

$$L_i = f_i - d_i$$



How much the completion of a task is late w.r.t. its absolute deadline

EDD: schedulability test

given the definition of lateness:

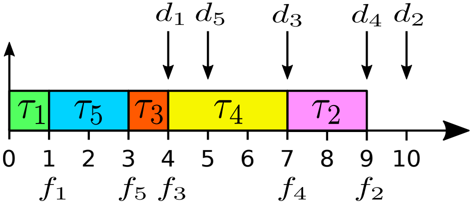
$$L_i = f_i - d_i$$

to check the schedulability of the task set it suffices to check that every task τ_i has lateness $L_i \leq 0$ (or, similarly, $\max_i\{L_i\} \leq 0$)

- the absolute deadline d_i is known for every task
- the finishing time f_i of τ_i can be computed by summing the duration of all tasks executed before τ_i plus C_i

EDD: example of schedulability test (1)

	τ_1	τ_2	τ_3	τ_4	τ_5
C_i	1	2	1	3	2
d_i	4	10	7	9	5



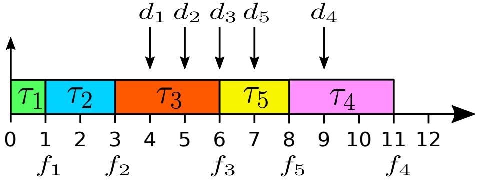
Tasks are sorted and scheduled in increasing deadline (decreasing priority) order.

task	τ_1	τ_5	τ_3	τ_4	τ_2
params (C_i, d_i)	$(1,4)$	$(2,5)$	$(1,7)$	$(3,9)$	$(2,10)$
f_i	1	3	4	7	9
$L_i = f_i - d_i$	-3	-2	-3	-2	-1

$\max_i\{L_i\} = -1$: task set schedulable

EDD: example of schedulability test (2)

	τ_1	τ_2	τ_3	τ_4	τ_5
C_i	1	2	3	3	2
d_i	4	5	6	9	7



Tasks are sorted and scheduled in increasing deadline (decreasing priority) order.

task	τ_1	τ_2	τ_3	τ_5	τ_4
params (C_i, d_i)	$(1,4)$	$(2,5)$	$(3,6)$	$(2,7)$	$(3,9)$
f_i	1	3	6	8	11
$L_i = f_i - d_i$	-3	-2	0	1	2

$\max_i\{L_i\} = 2$: the task set is not schedulable

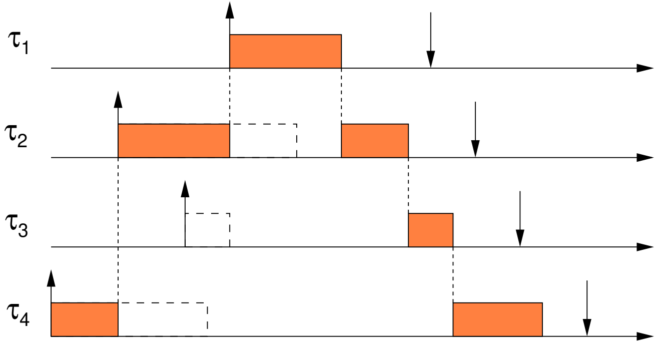
Earliest Deadline First (EDF)

the task having the closest absolute deadline
becomes the highest priority task

assumptions:

- tasks can arrive at any moment
- dynamic priorities: d_i depends on the arrival time
- full-preemptive system
- **minimizes the maximum lateness** L_{max}

EDF: example of scheduling

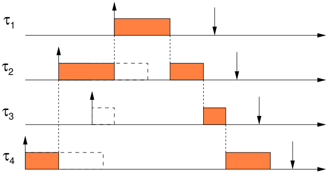


- we consider a preemptive system

EDF: scheduling decisions and schedulability test

Two events trigger a scheduling decision:

- A new task is released (becomes ready for execution)
- The running task completes its execution



The schedulability test is applied everytime a new task is released

- Goal: verify if the set of tasks composed by already guaranteed tasks and the new task is schedulable
- If adding the new task makes the set of tasks not schedulable, then the new task is rejected (the simplest solution)

EDF: schedulability test

Basic idea: check if there is enough time to fit each task τ_i in the time interval between the current time t and the absolute deadline d_i, considering the "interference" of tasks with higher priority.

Tasks are sorted in decreasing priority order
(τ_1 has the highest priority, τ_n has the lowest priority)

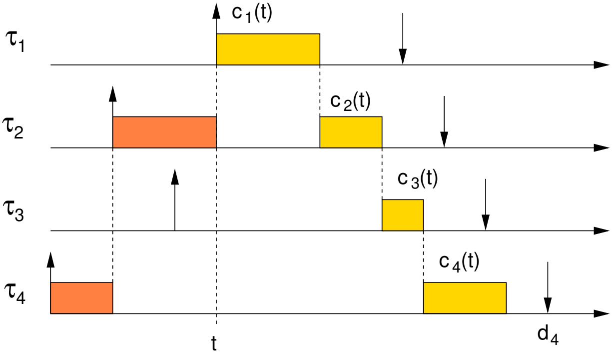
$$\forall \tau_i \quad \sum_{k=1}^{i} c_k(t) \leq d_i - t$$

where:

- t : current time (the arrival time of a new task)
- $c_k(t)$: remaining execution time of task τ_k at time t

EDF: schedulability test (example)

$$\forall \tau_i \quad \sum_{k=1}^{i} c_k(t) \leq d_i - t$$



E.g. for task τ_4 : $c_1(t) + c_2(t) + c_3(t) + c_4(t) \leq d_4 - t$

Comparing the complexity of EDD and EDF

EDD

- $O(n \log n)$ to sort the task set
- $O(n)$ to check the schedulability

EDF

- $O(n)$ to insert a new task into the ready queue
- $O(n)$ to check the schedulability

Optimality of EDF

EDF is optimal in the sense of schedulability: it is guaranteed to find a schedule if one exists

in other words

- if an optimal algorithm (in the sense of schedulability) fails to generate a feasible schedule, then no other algorithm can find a feasible schedule
- if an algorithm minimizes the maximum lateness L_{max}, then it is optimal (in the sense of schedulability)
- the opposite does not hold

Optimality of EDF: formal proof

the proof is due to Dertouzos (1974)

1. the proof starts from a feasible schedule σ^A generated by an algorithm $A \neq EDF$
2. a procedure is applied to transform σ^A into σ^{EDF}
3. it is shown that the procedure does not change the timing constraints of the schedule
4. it is shown that σ^{EDF} is feasible

Optimality of EDF: formal proof

- the procedure holds for a generic schedule
- this proves that EDF is able to generate a feasible schedule if one exists

the above feature is plainly
the definition of optimality

- therefore EDF is optimal

Dertouzos's algorithm

for all $t \in [0, D-1]$ **do**
  if $\sigma(t) \neq E(t)$ **then**
    $\sigma(t_E) = \sigma(t)$
    $\sigma(t) = E(t)$
  end if
end for

- the timeline is divided into time slices
- t is the time corresponding to a slice
- D is the largest deadline among all tasks
- $\sigma(t)$ is the running task at slice t
- $E(t)$ is the active task having the closest deadline at slice t
- $t_E \geq t$ is the closest instant to t in which $E(t)$ is executed

1. for each time slice, it is checked if it belongs to the task having the closest absolute deadline
2. if so, the schedule is already the same as the one produced by EDF
3. otherwise, the time slices are swapped

Dertouzos's algorithm

for all $t \in [0, D-1]$ **do**
  if $\sigma(t) \neq E(t)$ **then**
    $\sigma(t_E) = \sigma(t)$
    $\sigma(t) = E(t)$
  end if
end for

- the timeline is divided into time slices
- t is the time corresponding to a slice
- D is the largest deadline among all tasks
- $\sigma(t)$ is the running task at slice t
- $E(t)$ is the active task having the closest deadline at slice t
- $t_E \geq t$ is the closest instant to t in which $E(t)$ is executed

important notes

- the computation time of tasks is not affected (in case, time slices are swapped)
- arrival times and deadlines are not affected
- each time slice is delayed to – at most – time t_E

Dertouzos's algorithm

for all $t \in [0, D-1]$ **do**
  if $\sigma(t) \neq E(t)$ **then**
    $\sigma(t_E) = \sigma(t)$
    $\sigma(t) = E(t)$
  end if
end for

- the timeline is divided into time slices
- t is the time corresponding to a slice
- D is the largest deadline among all tasks
- $\sigma(t)$ is the running task at slice t
- $E(t)$ is the active task having the closest deadline at slice t
- $t_E \geq t$ is the closest instant to t in which $E(t)$ is executed

now, it holds

1. $t_E + 1 \leq d_E$ since σ^A is feasible
2. $d_E \leq d_i$ (d_E is the closest absolute deadline)
3. therefore $t_E + 1 \leq d_E \leq d_i$ in σ^{EDF}

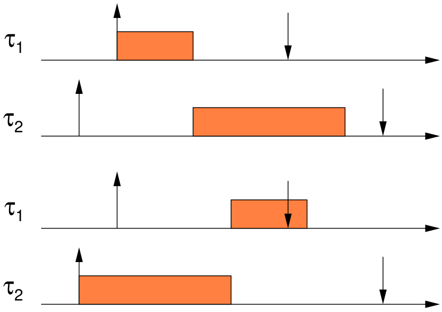
- each task terminates before its deadline $\Rightarrow \sigma^{EDF}$ is feasible
- EDF is optimal

Non-preemptive scheduling

in case of non-preemptive systems
EDF is not optimal

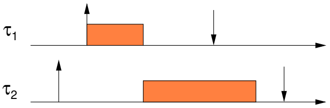
feasible schedule

EDF



Non-preemptive scheduling

- to ensure optimality in a non-preemptive system, the algorithm should be “clairvoyant”
- it should be able to decide to leave the CPU unused even in presence of ready tasks



although τ_2 becomes ready before τ_1, it is not executed

> if the possibility to leave the processor idle when there are ready task is forbidden, then EDF is optimal for this class of algorithms (work-conserving)

Non-preemptive scheduling: heuristics approach

the problem of finding a feasible schedule has NP-hard complexity

- heuristic techniques are adopted to obtain good results in reasonable time
- the computation is done offline
- typical methods are based on the exploration of graphs