Course 02224

Scheduling

Basic Notions

Hans Henrik Løvengreen

DTU Compute, Technical University of Denmark

General Scheduling Notions

- Given
  - A number of *activities* (tasks, jobs) to be done
    Each may be one-instance or recurrent
  - A number of *resources* to be utilized
  - A specification of the resources needed for each task
- A *schedule* is a mapping over time between tasks and resources consistent with the specification.
- *Scheduling* is the process of determining a schedule
- Scheduling is done by a *scheduler* according to a *policy*
- Scheduling may be *static* (planning) or *dynamic*

Scheduling Objectives

Activity Types

- *Batch* activities — High CPU/IO ratio
- *Interactive* activities — Low CPU/IO ratio
- *Real-time* activities — deadlines, timeliness

Goals

- *Fairness* — every process/user/client get a fair share
- Good *utilization* — no resource is unnecessarily idle
- High *throughput*
- Acceptable *response times*
- *Deadlines* must be met

Real-Time Scheduling Problem

Real-time Programs

- A *real-time program* is a reactive program with real-time requirements.
- Real-time requirements may be *hard*, *firm*, or *soft*.

Real-Time Scheduling

- Given a real-time program, the *scheduling problem* is to ensure that the (hard) real-time requirements are met given a limited number of HW resources
- Traditionally, mostly CPU-resources have been considered.
- Other resources may also be considered:

  Communication bandwith, power consumption ...

Real-Time Scheduling Notions

Schedulability

- Real-time requirements can usually be stated as *deadlines*
- A *feasible schedule* is a schedule which meets the deadlines

Scheme

- A *scheduling scheme* consists of:
  - A *scheduling policy* determining the possible schedules
  - A *schedulability test* ensuring that all schedules are feasible
- A test may be *sufficient* and/or *necessary*
- A test is *sustainable* if improving conditions preserves feasability

Optimality

- A scheduling policy is *optimal* (for a given class of scheduling problems) iff it can find a feasible schedule, whenever one exists

The Task Execution Model

An arbitrary program is too complex to analyse.

To facilitate analysis, an *abstract execution model* is used:

- The program consists of a fixed number N of computational *tasks*
- Tasks are recurrent: **loop**

  ```
  await next release
  compute task
  respond
  ```
- Tasks are *released* periodically with fixed *periods* T_i
- Tasks have fixed *worst-case execution times* C_i
- Tasks have fixed response *deadlines* D_i ($C_i \leq D_i \leq T_i$)
- Tasks may have fixed *start times* (offsets) S_i ($S_i \geq 0$)

The Simple Scheduling Problem

- Tasks have no suspension points
- *Overheads* (context switch, scheduling, ...) are ignored
- Only a execution on a *single CPU* resource is considered
- Tasks are assumed to be *independent*:
  - No sharing of resources other resources than CPU(s)
  - No precedence relation
  - No assumption on start times
- No aperiodic tasks
- Tasks must run within their period: $D_i = T_i$

Static Scheduling

Idea

- A feasible, repeatable schedule is determined statically.
- The schedule is executed by a single *cyclic executive* driven by a regular hardware tick.

Properties

\+ Simple implementation not requiring an operating system

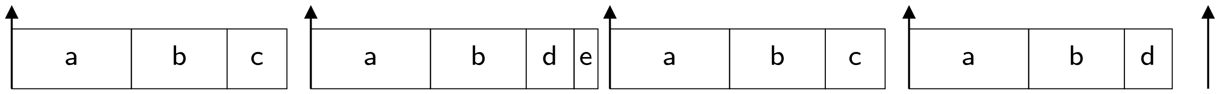
− Very difficult to adapt to task modifications

− Cannot utilize spare time for other activities

− Not amendable for multi-processing

Cyclic Executive

Task	T	C
a	25	10
b	25	8
c	50	5
d	50	4
e	100	2



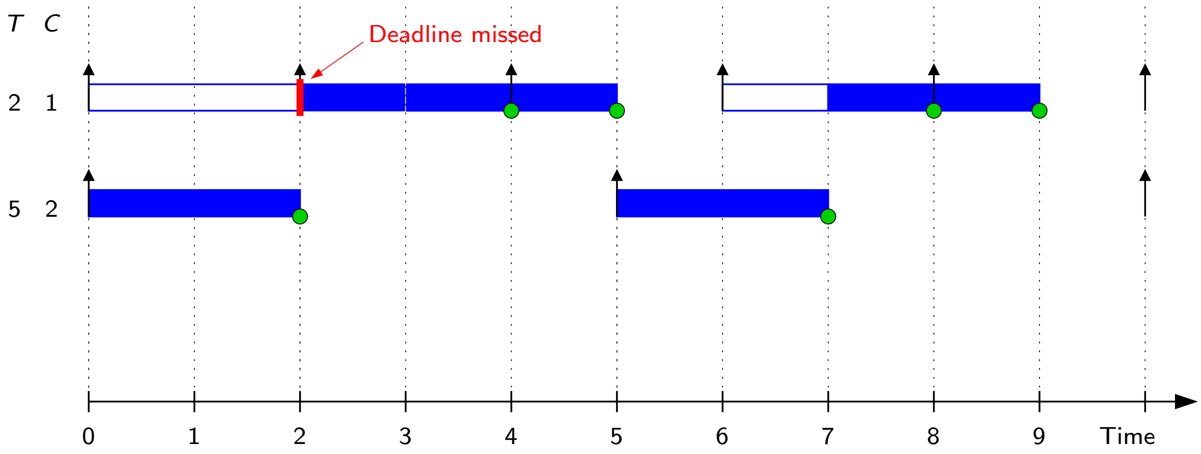
Thread Based Task Scheduling

- Tasks are performed repeatedly by dedicated threads scheduled by an OS.
- A task may be *ready* or *waiting*.
- A *scheduler* dynamically chooses a ready task for execution
- General policies:
  - Fair scheduling, e.g. *round robin* time-sharing
  - Priority based scheduling
- Fair scheduling not suited for hard real-time
- Priorities may be *fixed* or *dynamic*.

Round-Robin Scheduling

- Ready tasks are executed in *time slots* in a cyclic way

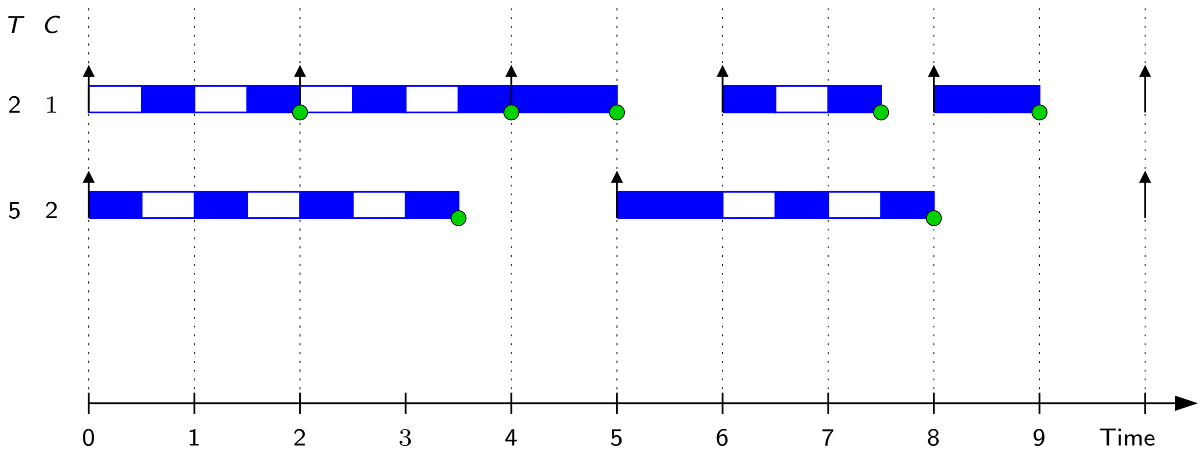
Example time slot = 2.0



Round-Robin Scheduling

- Ready thrads are executed in *time slots* in a cyclic way

Example time slot = 0.5

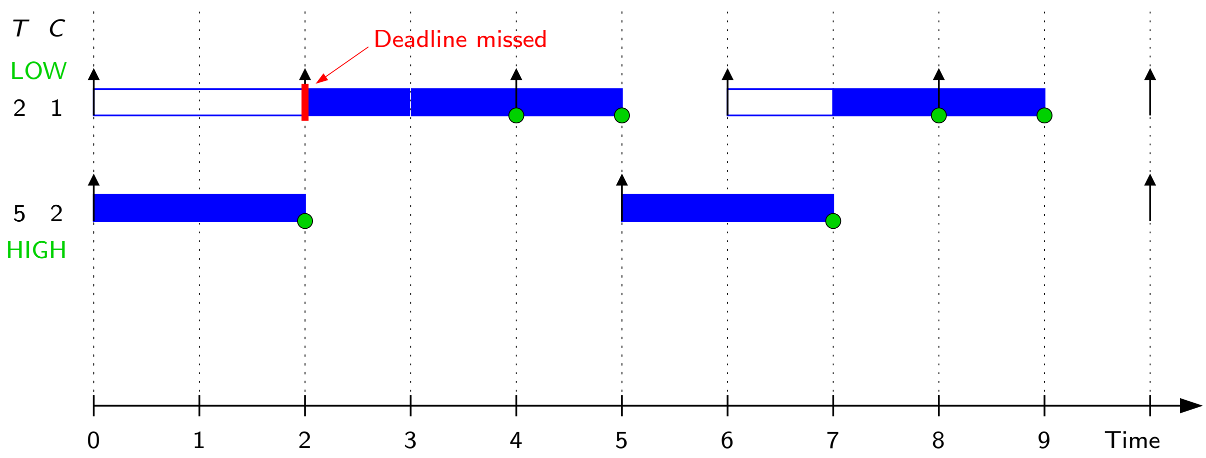


Fixed Priority Scheduling (FPS)

- Idea: Assign fixed priorities to tasks
- Scheduler needs to know about priorities (only)
- Always choose highest priority task when rescheduling
- Scheduling points:
  - *Non-preemptive scheduling:* Let current task run till completion
  - *Periodic scheduling:* Schedule at periodic intervals
  - *Deferred scheduling:* Schedule after a given time
  - *Preemptive scheduling:* Preempt current task if higher prioritized task becomes ready
- How should equal priority tasks be treated?
- What is the best way of assigning priorities?

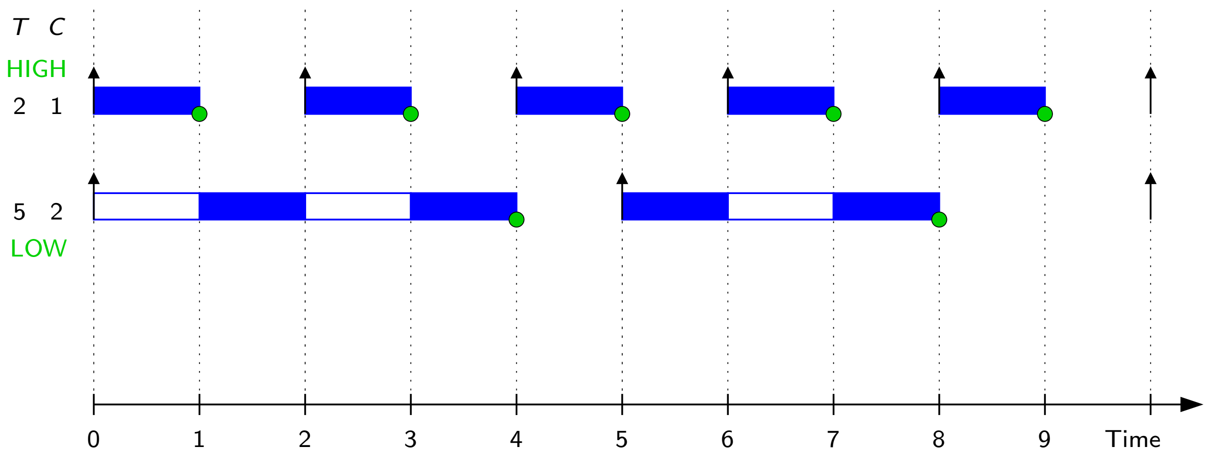
Fixed Priority Scheduling

Example



Fixed Priority Scheduling

Example



Rate-Monotonic Priority Assignment

- A rate-monotonic priority assignment (RMA) satisfies:

$$T_i < T_j \;\Rightarrow\; P_i > P_j$$

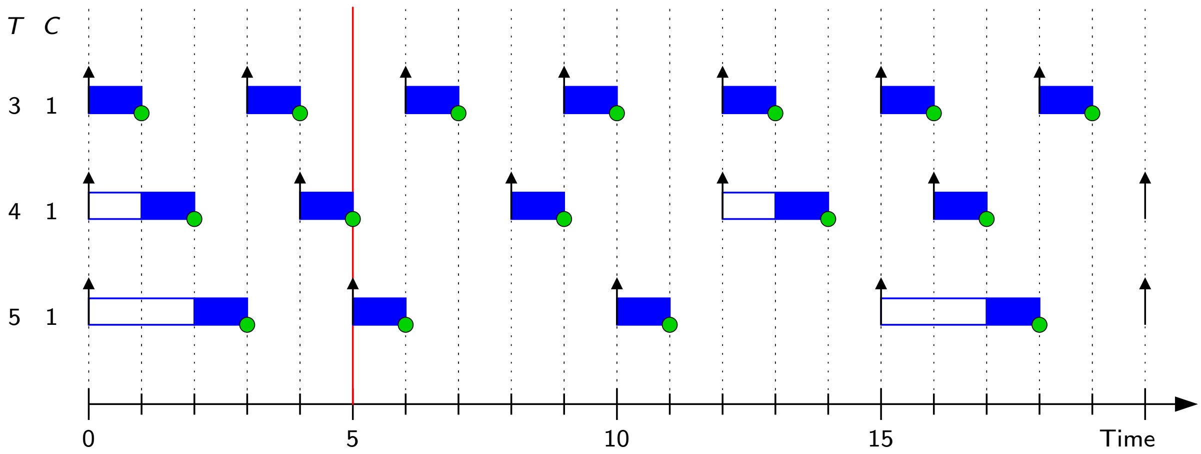
- Optimal for the simple scheduling problem:

  If a set of tasks is schedulable for some fixed priority assignment, then:

  The set of tasks is schedulable using rate-monotonic priority assignment.

Fixed Priority Scheduling

RMA Example



- Feasible if all deadlines met from critical instant till longest period

Utilization

- The processor *utilization* (load) contributed by task i:

$$U_i = \frac{C_i}{T_i}$$

- For a set of tasks running on a single CPU:

$$\sum_{i=1}^{N} \frac{C_i}{T_i} \leq 1$$

- How high can we go while ensuring schedulability?

Utilization based check

In 1973, Liu and Layland showed a sufficient condition for schedulability (using rate-monotonic priority assignment):

$$\sum_{i=1}^{N} \frac{C_i}{T_i} < N(2^{1/N} - 1)$$

- For various N this gives:

N	Utilization (%)	
1	100.0	
2	82.8	
3	78.0	
4	75.7	
10	71.8	
∞	69.3	($ln\,2$)

- Bound sometimes too pessimistic
- Refinement: N = number of *task families* (having harmonic periods)

Utilization based check

In 2006, Bini et.al. showed an alternative sufficent condition:

$$\prod_{i=1}^{N} \left(\frac{C_i}{T_i} + 1 \right) \leq 2$$

Example

- Consider again:

Task	T	C
a	3	1
b	4	1
c	5	1

- $(1/3 + 1) \cdot (1/4 + 1) \cdot (1/5 + 1) = (4 \cdot 5 \cdot 6)/(3 \cdot 4 \cdot 5) = 2$ $\surd$

Response Time Analysis (RTA)

Idea

- Assume some priority assignment given.
- Calculate worst-case *response time* R_i for each task.
- If $R_i \leq T_i$ for all i, the tasks are schedulable.

The response time satisfies:

$$R_i^{\,k+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i^{\,k}}{T_j} \right\rceil C_j$$

- R_i can be found by an iterative process.

Response Time Analysis

Example

	T	C	R^0	R^1	R^2	R^3	...	
a	50	10	10	$\underline{10}$				
b	70	15	15	25	$\underline{25}$			
c	110	40	40	65	75	90	$\underline{90}$	
d	200	30	30	95	120	170	195	$\underline{195}$

Meeting Deadlines

- Some tasks may require $D_i < T_i$
- If $R_i \leq D_i$ for all processes, all deadlines are met
- A *deadline monotonic* priority assignment satisfies:

$$D_i < D_j \Rightarrow P_i > P_j$$

- Deadline monotonic priority assignment is optimal for independent processes

Proof (sketch)

- Given a feasible priority assignment W
- Swap any two adjacent tasks with $P_i > P_j$ but $D_i > D_j$
- When none left, assignment is feasible and deadline monotonic
- Corrollary: RMA is optimal for $D_i = T_i$

Sporadic Tasks

- *Aperiodic tasks* are released at arbitrary times by external or internal events
- *Sporadic tasks* have a *minimum arrival time* T_i
- Sporadic tasks can be treated as periodic tasks with period T_i
- Alternatively, aperiodic and sporadic tasks may be handled by a *server task* given a certain fraction of the processing time
- Sporadic tasks often have $D_i < T_i$ (eg. alarms)

Dynamic Priority Scheduling

- Idea: Base scheduling choice on current situation
- Scheduler needs to be aware of more task parameters
- Better exploitation of processor time

Common Schemes

- Least Completion Time (LCT)
- Least Slack time (LST)
- Earliest Deadline First (EDF)

Earliest Deadline First (EDF)

Principle

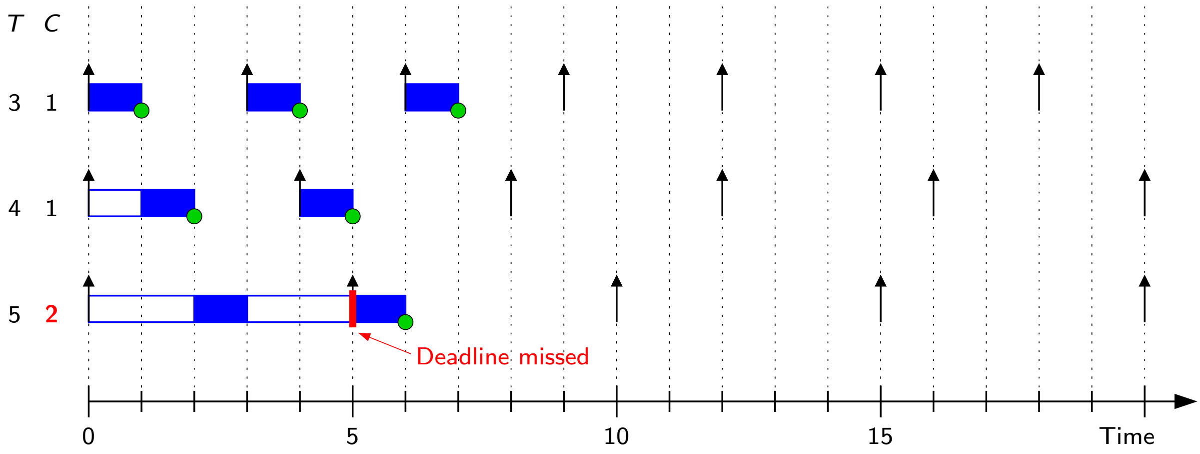
- Always run tasks with first-coming deadline

Properties

+ Optimal algorithm: Feasible iff $\sum U_i \leq 1$ (for $D_i = T_i$)
− Scheduler must know about and maintain deadlines
− Performs poorly on overload
− Difficult to analyse response times
− Difficult to analyse for non-preemptive processes

EDF Scheduling

Example FPS



EDF Scheduling

Example EDF