Inovace tohoto kurzu byla v roce 2011/12 podpořena projektem CZ.2.17/3.1.00/33274 financovaným Evropským sociálním fondem a Magistrátem hl. m. Prahy.

Evropský sociální fond
Praha & EU: Investujeme do vaší budoucnosti
Embedded and Real-time Systems
Scheduling – Basic Concepts

http://d3s.mff.cuni.cz

Department of Distributed and Dependable Systems

Tomáš Bureš
<bures@d3s.mff.cuni.cz>

CHARLES UNIVERSITY IN PRAGUE
Faculty of Mathematics and Physics
Short summary of lecture 1

- Real-time systems
  - Correctness of the system depends on time
  - Close interaction with the environment (via sensors and actuators)
  - Real-time ≠ fast, real-time = predictable
  - Handling of several activities at the same time
  - Possibility to prioritize among activities

- Some classification of real-time systems
  - Event triggered / time triggered
  - Hard / soft
Tasks and scheduling

- RT system is typically responsible for a number of concurrent activities
- Each activity has its deadline
  - Activity \sim Task
- The key problem in RT systems is how to schedule the tasks such as each meets the deadline
Tasks and scheduling

- Tasks compete for processor(s)
- They run according to a schedule
  - assigns a particular task to a particular processor at given time
- Formally (for uniprocessor):
  - Given a set of tasks, \( J = \{ J_1, \ldots, J_n \} \), a schedule is an assignment of tasks defined as a function \( \sigma : \mathbb{R}^+ \rightarrow \mathbb{N} \) such that \( \forall t \in \mathbb{R}^+ \), \( \exists t_1, t_2 \) such that \( t \in [t_1, t_2) \) and \( \forall t' \in [t_1, t_2) : \sigma(t) = \sigma(t') \)
  - \( \sigma(t) = k \) with \( k > 0 \) means that task \( J_k \) is executing at time \( t \), while \( \sigma(t) = 0 \) means that the CPU is idle
Tasks and scheduling

- At time $t_1, t_2, t_3,$ and $t_4$, the processor performs a context switch.
- If the intervals were disjointed – preemptive schedule.

A schedule is feasible is all tasks can be completed according to a set of specified constraints.

A set of tasks is schedulable, if there exists at least one algorithm that can produce a feasible schedule.
Timing constraints

- Arrival time $a_i$ – time at which a task becomes ready for execution (also called release time – $r_i$)
- Computation time $C_i$ – time necessary for executing the task without interruption (typically determined by WCET analysis)
- Absolute deadline $d_i$ – time before which a task should be completed
- Relative deadline $D_i$ – equals to $d_i - r_i$
- Start time $s_i$ – time at which a task starts its execution
- Finishing time $f_i$ – time at which a task finishes its execution
Timing constraints

- Response time $R_i$ – difference $f_i - r_i$

- Lateness $L_i$ – equals to $f_i - d_i$, represents the delay of a task completion with respect to its deadline
  - if a task completes in time, it is negative

- Tardiness (Exceeding time) $E_i$ – equals to $\max(0, L_i)$, is the time a task stays active after its deadline

- Laxity (Slack time) $X_i$ – equals to $d_i - a_i - C_i$, is the maximum time a task can be delayed to still complete within deadline
Regularity of the activation

- **Periodic task**
  - consists of an infinite sequence of identical activities (instances or jobs) activated regularly at a constant rate
  - denoted $\pi_i$
  - activation time of the first instance – phase $\Phi_i$
  - period of the task $T_i$
  - activation time of $k_{th}$ instance is $\Phi_i + (k - 1)T_i$

- **Aperiodic task**
  - also infinite number of instances, however activation is not regular
  - jobs with minimum inter-arrival time are called *sporadic*
Periodic task execution

- Typically realized this way:
  - RTOS kernels with support for periodic tasks
    ```c
    period_T = 50;
    void task_T() {
        /* do some work */
    }
    ```
  - RTOS kernels with no support for periodic tasks
    ```c
    void task_T() {
        while (1) {
            /* do some work */
            /* wait until next period */
        }
    }
    ```
Precedence constraints

- Sometimes there is a precedence constraint between tasks
  - one task must complete before other task may start running
  - e.g. low-level image processing task may start only after task for image acquisition completes
Resource constraints

- **Resource** is any software structure that can be used by the process to advance its execution.
- **Mutually exclusive resources** must not be used by two tasks at the same time.
- **Critical section**
- **Job must wait for the resource to become available**
General scheduling problem

- A set of \( n \) tasks \( J = \{ J_1, J_2, \ldots, J_n \} \),
  a set of \( m \) processors \( P = \{ P_1, P_2, \ldots, P_m \} \),
  a set of \( r \) resources \( R = \{ R_1, R_2, \ldots, R_r \} \),
  precedence relations in the form of an acyclic graph, timing on tasks
- Scheduling means assigning processors from \( P \) and resources from \( R \) to tasks from \( J \) in order to complete all tasks under the imposed constraints
- In general form, NP-complete
Classification of scheduling algorithms

- **Preemptive** – a running task can be interrupted at any time
- **Non-preemptive** – a task once started is executed until completion
- **Static** – scheduling decisions are based on fixed parameters
- **Dynamic** – parameters of scheduling decisions may change during runtime
- **Off line** – scheduling algorithm executed on the entire task set before actual task activation (schedule determined in advance)
  - stored in a table
- **On line** – scheduling algorithm decides at runtime every time a new task enters the system or when running task terminates
- **Optimal** – minimizes some given cost functions, when no cost function is given it is optimal if it always finds a feasible schedule provided such schedule exists
Scheduling Anomalies

• Theorem (Graham)
  - If a task set is optimally scheduled on a multiprocessor with some priority assignment, a fixed number of processors, fixed execution times, and precedence constraints, then increasing the number of processors, reducing execution times, or weakening the precedence constraints can increase the schedule length.
Scheduling Anomalies – Example I

- Original optimal schedule

![Diagram showing the original optimal schedule with nodes J1 to J9 and their priorities, and a table representing the schedule on three processors P1, P2, and P3. The table shows the time slots assigned to each job on each processor, with J3 highlighted on P3. The diagram includes a line indicating priority(Ji) > priority(Jj) for i < j.]
• Number of processors increased
Scheduling Anomalies – Example I

- Computation times reduced
Scheduling Anomalies – Example I

- Precedence constraints weakened

priority(J_i) > priority(J_j)
Scheduling Anomalies – Example II

- Anomalies under resource constraints
  - J2 and J4 share the same resource

- When computation time of J1 is reduced