Reference model of real-time systems

Chapter 3 of Liu

Michal Sojka

Czech Technical University in Prague,
FEE and CIIRC

September 20, 2022

Some slides are derived from lectures by Steve Goddard and James H. Anderson

Reference model of RT systems

In order to analyze a RT system/application, it is necessary to create its model.

Main parts of RT system models

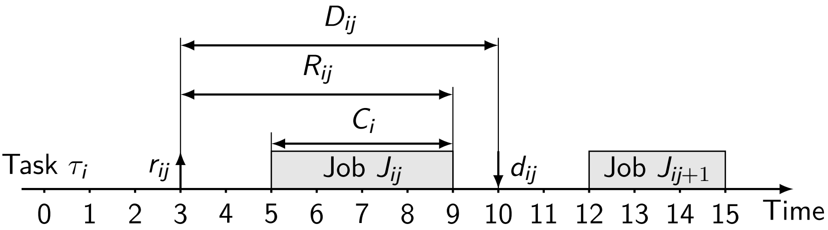
- Workload model describes the applications in the system.
- Resource model describes available system resources.
- Algorithms that define how the system resources are used.

Outline

Real-Time Applications Categories

- Purely periodic
  - Every task is released periodically
  - Constant or almost constant demand for system resources
  - **Examples:** digital controller, flight control, real-time monitoring
- Mostly periodic
  - Most of the tasks are released periodically
  - System has to respond to external asynchronous events
  - **Examples:** modern avionics or control systems
- Asynchronous and predictable
  - Most of the tasks are aperiodic
  - Requirements for system resources can change dramatically for the consecutive task activations, but there are limits known in advance or their statistical distribution is known.
  - **Examples:** multimedia communication, radar signal processing and tracking facilities
- Asynchronous and non-predictable
  - Most of the events are asynchronous
  - Task with high level of complexity
  - **Examples:** real-time control with artificial intelligence, real-time simulation, virtual reality

Job and task description



- ↑ = release time (r_{ij}); the job is released at time 3.
- ↓ = absolute deadline (d_{ij}); the job has to be completed before deadline; equal to 10 for this case.
- Relative deadline (D_{ij}) is 7.
- Response time (R_{ij}) is 6.

Terminology – detail

- Task τ_i: A set of jobs executed in order to perform certain function in the system, e.g. airplane stabilization.
- Job J_{ij}: An instance of task.
- Jobs need resources.
  - **Examples of resources:** CPU, network, critical section, shovel
  - Resources that can perform some work are called processors.
- Release time r_{ij}: Time instant when a job is ready to be executed.
- Deadline d_{ij}: Time instant by which the job has to be finished.
- Relative deadline D_i: Difference between deadline and release time.
- Response time R_{ij}: Completion time minus release time.
- Execution (computation) time C_{ij}: Time needed to execute a job if runs alone on a processor.
- Feasible interval of a job: Interval between r_{ij} and d_{ij}.

Hard Real-Time Systems

- Hard Deadline is a deadline that has to be met under all circumstances.
  - If a hard deadline is missed, the behavior of the system is wrong and it often has catastrophic consequences.
  - We need mathematical apparatus for verifying that deadlines are met.
  - But: "There is nothing like a hard deadline in the real world."
- Hard Real-Time System: is a real-time system, where all deadlines are hard.
  - This course is focused on hard real-time systems. They are easier to analyze. Why?
- **Examples:** Nuclear power plant, aircraft control.

Soft Real-Time Systems

- Soft Deadline (required completion time) can be missed occasionally.
    - **Question:** How to define the term "occasionally"?
- Soft Real-Time System: a real-time system where all deadlines are soft.
- **Example:** Multimedia applications, telephone exchanges (but what about emergency calls?).

Reference model of RT systems

- Each job J_i is characterized by its
  - timing parameters,
  - functional parameters,
  - resource describing parameters and
  - dependencies between individual jobs.
- Each job J_i has its release time r_i, deadline d_i, relative deadline D_i, computation time C_i (often called execution time or worst-case execution time, WCET).
- Occasionally, some parameters are defined as ranges. E.g $r_i \in \langle r_i^-, r_i^+ \rangle$. The size of the interval is called release-time jitter.
- Similarly, execution time can be given as interval $\langle C_i^-, C_i^+ \rangle$.
  - Determination of exact value of C_i might be difficult. Why?

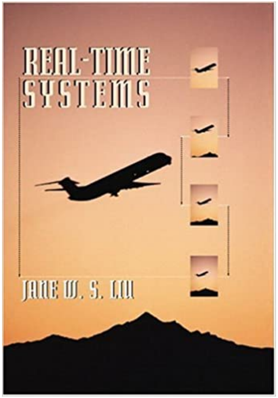
Periodic, sporadic and aperiodic task models

- Periodic task model – deterministic workload model, well suited for many hard real-time applications.
- Periodic task:
  - Each task τ_i has its period T_i. Task τ_i is composed of sequence of jobs.
  - T_i is minimal inter-arrival time between consecutive jobs.
  - Task computation time is the maximum computation time among all jobs of τ_i.
- Sporadic and aperiodic tasks – released at arbitrary times.
  - Sporadic tasks have hard deadlines.
  - Aperiodic tasks have no or soft deadlines.

Liu vs. rest of the world

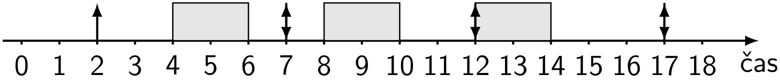
Beware!

- What Liu calls "periodic" the rest of the world calls "sporadic".
- For others period T_i of task τ_i means exact time between activations of two consecutively released jobs.



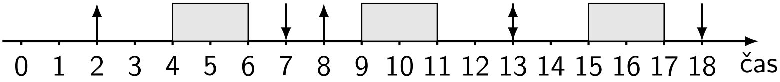
Examples

Periodic task τ_i with $r_i = 2,\ T_i = 5,\ C_i = 2,\ D_i = 5$ can be executed like this (continues until infinity).



Legend: ↑ = job release time , ↓ = deadline.

According to Liu, this task can execute, for example, like this:



The rest of the world calls this sporadic task.

Some definitions for periodic task systems

- Number of tasks is n.
- The jobs of task τ_i are denoted $J_{i,1}$, $J_{i,2}$, ...
- $\Phi_i = r_{i,1}$ (release time of $J_{i,1}$) is called the phase τ_i.
  - Synchronous system: Each task has phase of 0.
  - Asynchronous system: Phases are arbitrary.
  - What is more common?
- Hyperperiod: Least common multiple of $\{T_1, ..., T_n\}$.
- Task utilization: $u_i = \frac{C_i}{T_i}$.
- System utilization: $U = \sum_{i=1,...,n} u_i$

Task/job dependencies

- Data flow and control dependencies between the jobs can constrain the order in which the jobs can be executed.
- Two main types of dependencies:
  - Mutual exclusion (critical sections)
  - Precedence constraints – e.g.: Job J_i can start only after another job J_k finishes.
- Tasks without any dependency on other tasks are called independent.
  - In the initial lectures, we will only consider independent tasks.
  - Software tasks running under a (RT)OS are rarely independent.

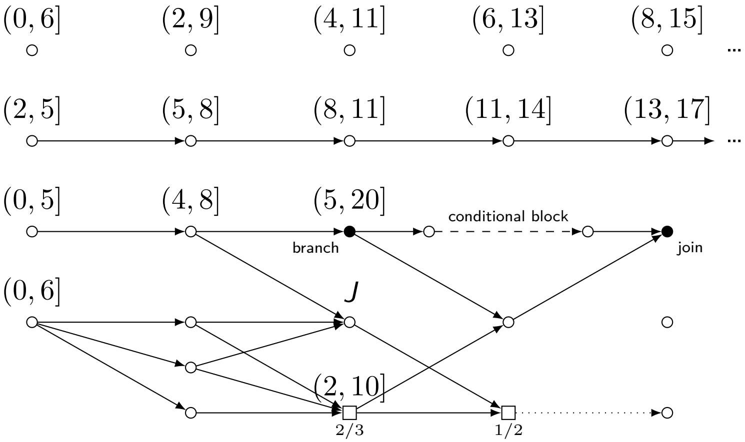
Job dependencies

- Precedence relation on a set of jobs is a relation, that determines precedence constrains among individual jobs.
- Job J_i is a predecessor of another job J_k (and J_k is successor of job J_i), if J_k cannot be started before J_i is finished.
- A job with predecessor is ready to be executed, when current time is greater than its release time and all its predecessors are completed.

Task graph

- Precedence graph – directed graph $G = (J, <)$, where each node represents a job from set J and if job J_i is immediate predecessor of J_k (relation $<$), there is a directed edge from node J_i to node J_k.
  - Data dependencies cannot be captured in the precedence graph.
- Task graph is an extended precedence graph. It can contain other types of dependencies.
  - Type of an edge connecting two nodes and other parameters of the edge is called interconnection parameters of the jobs.
  - Data dependencies are represented explicitly by data-dependency edges. An interconnection parameter can be, for example, the amount of data passed between the jobs.
  - Task graphs are rarely used periodic-task systems.

Task graph – example



- Numbers above a job give its feasible interval.

Other types of dependencies

- Time dependency (distance) is difference of job completion times.
- AND/OR precedence constraints – dependence among immediate job predecessors.
  - AND job – node *J*
  - OR jobs – square nodes marked $2/3$ a $1/2$.
- Conditional branches represent conditional execution of jobs.
  - Branch is a job represented by filled circles.
  - Conditional block – subgraph starting in a *branch* node and ending at next *join* job.
- Pipe relation is dependency among a pair of jobs that are in produce-consumer relation (dotted hrana).

Functional parameters

- Preemptivity of jobs
    - Preemptive
    - Non-preemptive
- Criticality of jobs
- Optional execution
- Laxity type and laxity function

Outline

Terminology

- Processors P_i (active resources) execute machine instructions, move data, read files etc.
  (CPU, communication links, disks, database servers)
- Resources R_i (passive resources) – additional resources needed by jobs to perform their task (memory, mutexes, semaphores). By resources we usually understand "reusable resources".
- Non-reusable resource is, for example, Energy (power-aware scheduling).

Resource parameters

- Processors
  - Speed of a processor
  - Topology of CPU interconnect/network-on-chip
- Preemptivity of resources (CPU, network, ...)
- Memory hierarchy (caches, DRAMs, ...)
- Resource graph
- Wake-up delay from power-saving state
- ...

Outline

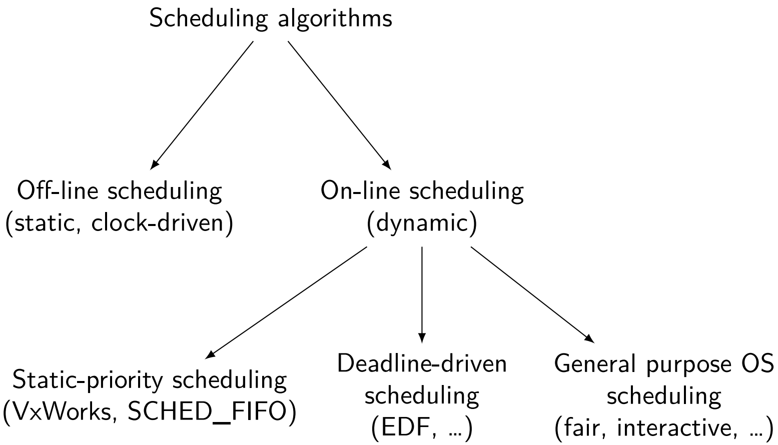
Scheduling algorithms

We are interested in two types of algorithms:

1. Scheduling algorithm, which produces the schedule of jobs (maybe at runtime).
    - In real-time systems, this algorithm is usually simple.
2. Schedulability analysis algorithm, which verifies whether all timing constraints are met.
    - This algorithm is typically more complex.

Classification of scheduling algorithms

(used in real-time systems)



Feasibility and optimality

- A valid schedule is a feasible schedule if every job completes by its deadline (or, in general, meets its timing constraints).
- A set of jobs τ is schedulable according to scheduling algorithm $\mathcal{A}$ if when using the algorithm scheduler always produces a feasible schedule for τ.
- Hard real-time scheduling algorithm is optimal if the algorithm always produces a feasible schedule if the given set of jobs has feasible schedules.
    - Similarly, we can define optimality for a class of schedulers – e.g.. "optimal scheduler for static priorities".

Outline

Model of a real-time system

Comprises of the following parts:

1. Workload model
   - Set of tasks/jobs and their parameters (C_i, D_i, resource dependencies, etc.)
   - Precedence graph or task graph
   - etc.
2. Resource model
   - Description of resources (CPU, memory, network, etc.), their types and relations among them.
   - Often: resource model is just "Uni-processor".
3. Algorithms
   - Fixed-priority scheduler + priority inheritance
   - Off-line scheduler

Real-Time system model – example