Embedded Systems

3. Real-Time Models
Contents of Course

1. Embedded Systems Introduction
2. Software Introduction
3. Real-Time Models
4. Periodic/Aperiodic Tasks
5. Resource Sharing
6. Real-Time OS
7. System Components
8. Communication
9. Low Power Design
10. Models
11. Architecture Synthesis
12. Model Based Design

Software and Programming
Processing and Communication
Hardware
Distinctive Embedded System Attributes

- Reactive: computations occur in response to external events
  - Periodic events (e.g. rotating machinery and control loops)
  - Aperiodic events (e.g. button click)

- Real Time: correctness is partially a function of time
  - Hard real time
    - Absolute deadline, beyond which answer is useless
    - May include minimum time as well as maximum time
  - Soft real time
    - Approximate deadline
    - Utility of answer degrades with time difference from deadline
Basic Terms

- **Real-time systems**
  - **Hard**: A real-time task is said to be hard, if missing its deadline may cause catastrophic consequences on the environment under control. Examples are sensory data acquisition, detection of critical conditions, actuator servoing.
  
  - **Soft**: A real-time task is called soft, if meeting its deadline is desirable for performance reasons, but missing its deadline does not cause serious damage to the environment and does not jeopardize correct system behavior. Examples are command interpreter of the user interface, displaying messages on the screen.
Real-Time Review

- **Real time is not just “real fast”**
  - Real time means that correctness of result depends on both functional correctness and time that the result is delivered

- **Soft real time**
  - Utility degrades with distance from deadline

- **Hard real time**
  - System fails if deadline window is missed

- **Firm real time**
  - Result has no utility outside deadline window, but system can withstand a few missed results
Type of Real-Time Scheduling

Taxonomy of Real-Time Scheduling
Type of Real-Time Scheduling

- **Dynamic vs. Static**
  - Dynamic schedule computed at run-time based on tasks really executing
  - Static schedule done at compile time for all possible tasks

- **Preemptive permits one task to preempt another one of lower priority**
Schedulability

If the sufficient schedulability test is positive, these tasks are definitely schedulable.

If the necessary schedulability test is negative, these tasks are definitely not schedulable.

Sufficient schedulability test

Exact schedulability test

Necessary schedulability test

Increasing Task Set Complexity

Necessary and sufficient schedulability test
Schedule

- Given a set of tasks $J = \{J_1, J_2, \ldots\}$:
  - 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 $\sigma : R \rightarrow N$ 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.
Schedule and Timing

- A schedule is said to be **feasible**, if all task can be completed according to a set of specified constraints.
- A set of tasks is said to be **schedulable**, if there exists at least one algorithm that can produce a feasible schedule.
- **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.
- **Deadline** $d_i$ is the time at which a task should be completed.
- **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.
Schedule and Timing

- Using the above definitions, we have \( d_i \geq r_i + C_i \).
- **Lateness** \( L_i = f_i - d_i \) represents the delay of a task completion with respect to its deadline; note that if a task completes before the deadline, its lateness is negative.
- **Tardiness or exceeding time** \( E_i = \max(0, L_i) \) 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 task can be delayed on its activation to complete within its deadline.
Schedule and Timing

- **Periodic task** $\tau_i$: 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 $\Phi_i$.

![Diagram showing schedule and timing concepts, including relative deadline, phase, period, and aperiodic task examples.](image-url)
Example

Computation times: $C_1 = 9, C_2 = 12$
Start times: $s_1 = 0, s_2 = 6$
Finishing times: $f_1 = 18, f_2 = 28$
Lateness: $L_1 = -4, L_2 = 1$
Tardiness: $E_1 = 0, E_2 = 1$
Laxity: $X_1 = 13, X_2 = 11$
Precedence Constraints

- *Precedence relations* between graphs can be described through an acyclic directed graph $G$ where tasks are represented by nodes and precedence relations by arrows. $G$ induces a partial order on the task set.

- There are different interpretations possible:
  - All successors of a task are activated (concurrent task execution).
  - One successor of a task is activated (non-deterministic choice).
Precedence Constraints

Example (concurrent activation):

- Image acquisition \( acq1 \) \( acq2 \)
- Low level image processing \( edge1 \) \( edge2 \)
- Feature/contour extraction \( shape \)
- Pixel disparities \( disp \)
- Object size \( H \)
- Object recognition \( rec \)
Classification of Scheduling Algorithms

- With *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.

- With a *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 are based on dynamic parameters that may change during system execution.
Real-time Scheduling Paradigms - RM Issue

- Allocate time slots for tasks onto processor(s). [i.e., Where and When a given task executes]
- Objective: **predictably meeting task deadlines.** (schedulability check, schedule construction)

![Diagram showing real-time task scheduling with static and dynamic paradigms, and further breakdown into table-driven, priority-driven, planning-based, and best effort methods.](image)
Preemptive vs Non-preemptive scheduling

- **Preemptive Scheduling**
  - Task execution is preempted and resumed later.
  - Preemption takes place to execute a higher priority task.
  - Offers higher schedulability.
  - Involves higher scheduling overhead due to context switching.
Preemptive vs Non-preemptive scheduling

- **Non-preemptive Scheduling**
  - Once a task is started executing, it completes its execution.
  - Offers lower schedulability.
  - Has less scheduling overhead because of less context switching.
Optimal scheduling -- definition

- A **static scheduling** algorithm is said to be **optimal** if, for any set of tasks, it always produces a schedule which satisfies the constraints of the tasks (feasible schedule) whenever any other algorithm can do so.

A **dynamic scheduling** algorithm is said to be **optimal** if it always produces a **feasible schedule** whenever a **static algorithm with complete prior knowledge of all the possible tasks** can do so.
Other RM Issues

● Resource Reclaiming
  – Resource reclaiming refers to the problem of utilizing resources left unused by a task.

● Fault Tolerance
  – Fault-tolerance is informally defined as a system’s ability to deliver the expected service even in the presence of faults.
Other RM Issues

- Communication
  - Any form of communication that involves timing constraints.
  - Real-time WAN: Routing and Scheduling
  - Real-time LAN: MAC protocols
Architectural Issues

- Predictability in: Instruction execution time, Memory access, Context switching, Interrupt handling.

- RT systems usually avoid caches and superscalar features.

- Support for error handling (self-checking circuitry, voters, system monitors).

- Support for fast and reliable communication (routing, priority handling, buffer and timer management).
Architectural Issues

- Support for scheduling algorithms (fast preempt ability, priority queues).
- Support for RTOS (multiple contexts, memory management, garbage collection, interrupt handling, clock synchronization).
- Support for RT language features (language constructs for estimating worst-case execution time of tasks).
Requirement, Specification, Verification

- **Functional requirements**: Operation of the system and their effects.
- **Non-Functional requirements**: e.g., timing constraints.
- F & NF requirements must be precisely defined and together used to construct the specification of the system.
Requirement, Specification, Verification

- A **specification** is a mathematical statement of the properties to be exhibited by a system. It is abstracted such that
  - it can be checked for conformity against the requirement.
  - its properties can be examined independently of the way in which it will be implemented.

- The usual approaches for specifying computing system behavior entail enumerating events or actions that the system participates in and describing orders in which they can occur. It is not well understood how to extend such approaches for real-time constraints.
Real-time Languages

● Support for the management of time
  – Language constructs for expressing timing constraint, keeping track of resource utilization.

● Schedulability analysis
  – Aid compile-time schedulability check.

● Reusable real-time software modules
  – Object-oriented methodology.

● Support for distributed programming and fault-tolerance
Classification of Scheduling Algorithms

- An algorithm is said **optimal** if it minimizes some given cost function defined over the task set.
- An algorithm is said to be **heuristic** if it tends toward but does not guarantee to find the optimal schedule.

Domino effect, if acceptance test wrongly accepted a new task.
Metrics

- Average response time:

\[ t_r = \frac{1}{n} \sum_{i=1}^{n} (f_i - r_i) \]

- Total completion time:

\[ t_c = \max_{i} (f_i) - \min_{i} (r_i) \]

- Weighted sum of completion time:

\[ t_w = \sum_{i=1}^{n} \frac{\sum_{i=1}^{n} w_i (f_i - r_i)}{\sum_{i=1}^{n} w_i} \]

- Maximum lateness:

\[ L_{\text{max}} = \max_{i} (f_i - d_i) \]

- Maximum number of late tasks:

\[ N_{\text{late}} = \sum_{i=1}^{n} \text{miss}(f_i) \]

\[ \text{miss}(f_i) = \begin{cases} 0 & \text{if } f_i \leq d_i \\ 1 & \text{otherwise} \end{cases} \]
Example

Computation times: $C_1 = 9$, $C_2 = 12$

Start times: $s_1 = 0$, $s_2 = 6$

Finishing times: $f_1 = 18$, $f_2 = 28$

Lateness: $L_1 = -4$, $L_2 = 1$

Tardiness: $E_1 = 0$, $E_2 = 1$

Laxity: $X_1 = 13$, $X_2 = 11$
Metrics Example

Average response time:
Total completion time:
Weighted sum of completion time:
Maximum number of late tasks:
Maximum lateness:

\[ \overline{t_r} = \frac{1}{2} (18 + 24) = 21 \]
\[ t_c = 28 - 0 = 28 \]
\[ w_1 = 2, w_2 = 1: \quad t_w = \frac{2 \cdot 18 + 24}{3} = 20 \]
\[ N_{\text{late}} = 1 \]
\[ L_{\text{max}} = 1 \]


**Scheduling Example**

- In (a), the maximum lateness is minimized, but all tasks miss their deadlines.

- In (b), the maximal lateness is larger, but only one task misses its deadline.

![Diagram showing scheduling example with tasks J1, J2, J3, J4, J5 and their deadlines and latenesses.](image)