Real-Time Scheduling
Characteristics of a RTS

• Large and complex

• OR small and embedded
  – Vary from a few hundred lines of assembler or C to millions of lines of high-level language code
  – Concurrent control of separate system components
    • Devices operate in parallel in the real-world, hence, better to model this parallelism by concurrent entities in the program

• Facilities to interact with special purpose hardware
  – Need to be able to program devices in a reliable and abstract way
Characteristics of a RTS

• Extreme reliability and safety
  – Embedded systems typically control the environment in which they operate
  – Failure to control can result in loss of life, damage to environment or economic loss

• Guaranteed response times
  – We need to be able to predict with confidence the worst case response times for systems
  – Efficiency is important but predictability is essential
    • In RTS, performance guarantees are:
      – Task- and/or class centric
      – Often ensured a priori
    • In conventional systems, performance is:
      – System oriented and often throughput oriented
      – Post-processing (... wait and see ...)
Typical Components of a RTS

- Real-Time Clock
- Database
- Operator’s Console
- Algorithms for Digital Control
- Data Logging
- Data Retrieval and Display
- Operator Interface
- Interface
- Remote Monitoring System
- Display Devices
- Engineering System

Real-Time Computer
Terminology

• **Scheduling**
  define a policy of how to order tasks such that a metric is maximized/minimized
  – Real-time: guarantee hard deadlines, minimize the number of missed deadlines, minimize lateness

• **Dispatching**
  carry out the execution according to the schedule
  – Preemption, context switching, monitoring, etc.

• **Admission Control**
  Filter tasks coming into the systems and thereby make sure the admitted workload is manageable

• **Allocation**
  designate tasks to CPUs and (possibly) nodes. Precedes scheduling
Preliminaries

Scheduling is the issue of ordering the use of system resources

- A means of predicting the worst-case behaviour of the system
Non-Real-Time Scheduling

• Primary Goal: maximize performance
• Secondary Goal: ensure fairness
• Typical metrics:
  – Minimize response time
  – Maximize throughput
  – E.g., FCFS (First-Come-First-Served), RR (Round-Robin)
Example: Workload Characteristics

- Tasks are preemptable, independent with arbitrary arrival (=release) times
- Times have *deadlines* \((D)\) and known *computation times* \((C)\)
- Tasks execute on a uni-processor system

Example Setup
Example:
Non-preemptive FCFS Scheduling

T1

T2

T3

T4

Missed deadline!!
Example:
Round-Robin Scheduling
Real-Time Scheduling

• Primary goal: ensure predictability
• Secondary goal: ensure predictability
• Typical metrics:
  – Guarantee miss ration = 0 (hard real-time)
  – Guarantee Probability(missed deadline) < X% (firm real-time)
  – Minimize miss ration / maximize completion ration (firm real-time)
  – Minimize overall tardiness; maximize overall usefulness (soft real-time)
• E.g., EDF (Earliest Deadline First, LLF (Least Laxity First), RMS (Rate-Monotonic Scheduling), DM (Deadline Monotonic Scheduling)
• Recall: Real-time is about enforcing predictability, and does not equal to fast computing!!!
Scheduling: Problem Space

- Uni-processor / multiprocessor / distributed system
- Periodic / sporadic / aperiodic tasks
- Independent / interdependent tasks

- Preemptive / non-preemptive
- Tick scheduling / event-driven scheduling
- Static (at design time) / dynamic (at run-time)
- Off-line (pre-computed schedule), on-line (scheduling decision at runtime)
- Handle transient overloads
- Support Fault tolerance
Task Assignment and Scheduling

- Cyclic executive scheduling (→ later)
- Cooperative scheduling
  - scheduler relies on the current process to give up the CPU before it can start the execution of another process
- A *static priority-driven scheduler* can preempt the current process to start a new process. Priorities are set pre-execution
  - E.g., Rate-monotonic scheduling (RMS), Deadline Monotonic scheduling (DM)
- A *dynamic priority-driven* scheduler can assign, and possibly also redefine, process priorities at run-time.
  - E.g., Earliest Deadline First (EDF), Least Laxity First (LLF)
Simple Process Model

- **Fixed** set of processes (tasks)
- Processes are **periodic**, with known periods
- Processes are **independent** of each other
- System overheads, context switches etc, are ignored (zero cost)
- Processes have a **deadline equal to their period**
  - i.e., each process must complete before its next release
- Processes have **fixed worst-case execution time** (WCET)
Terminology: Temporal Scope of a Task

- $C$ - Worst-case execution time of the task
- $D$ - Deadline of tasks, latest time by which the task should be complete
- $R$ - Release time
- $n$ - Number of tasks in the system
- $\pi$ - Priority of the task
- $P$ - Minimum inter-arrival time (period) of the task
  - Periodic: inter-arrival time is fixed
  - Sporadic: minimum inter-arrival time
  - Aperiodic: random distribution of inter-arrival times
- $J$ - Release jitter of a process
Performance Metrics

• Completion ratio / miss ration
• Maximize total usefulness value (weighted sum)
• Maximize value of a task
• Minimize lateness
• Minimize error (imprecise tasks)
• Feasibility (all tasks meet their deadlines)
Scheduling Approaches (Hard RTS)

- **Off-line scheduling / analysis** (static analysis + static scheduling)
  - All tasks, times and priorities given a priori (before system startup)
  - Time-driven; schedule computed and hardcoded (before system startup)
  - E.g., [Cyclic Executives](#)
  - Inflexible
  - May be combined with static or dynamic scheduling approaches

- **Fixed priority scheduling** (static analysis + dynamic scheduling)
  - All tasks, times and priorities given a priori (before system startup)
  - Priority-driven, dynamic(!) scheduling
    - The schedule is constructed by the OS scheduler at run time
  - For hard / safety critical systems
  - E.g., [RMA/RMS (Rate Monotonic Analysis / Rate Monotonic Scheduling)](#)

- **Dynamic priority scheduling**
  - Tasks times may or may not be known
  - Assigns priorities based on the current state of the system
  - For hard / best effort systems
  - E.g., [Least Completion Time (LCT), Earliest Deadline, First (EDF), Least Slack Time (LST)](#)

Cyclic Executive Approach

- Clock-driven (time-driven) scheduling algorithm
- Off-line algorithm
- Minor Cycle (e.g. 25ms) - gcd of all periods
- Major Cycle (e.g. 100ms) - lcm of all periods

Construction of a cyclic executive is equivalent to bin packing

<table>
<thead>
<tr>
<th>Process</th>
<th>Period</th>
<th>Comp. Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>25</td>
<td>10</td>
</tr>
<tr>
<td>B</td>
<td>25</td>
<td>8</td>
</tr>
<tr>
<td>C</td>
<td>50</td>
<td>5</td>
</tr>
<tr>
<td>D</td>
<td>50</td>
<td>4</td>
</tr>
<tr>
<td>E</td>
<td>100</td>
<td>2</td>
</tr>
</tbody>
</table>
Cyclic Executive (cont.)

loop
  Wait_For_Interrupt;
  Procedure_For_A;
  Procedure_For_B;
  Procedure_For_C;
  Wait_For_Interrupt;
  Procedure_For_A;
  Procedure_For_B;
  Procedure_For_D;
  Procedure_For_E;

end loop;
Cyclic Executive: Observations

• No actual processes exist at run-time
  – Each minor cycle is just a sequence of procedure calls

• The procedures share a common address space and can thus pass data between themselves.
  – This data does not need to be protected (via semaphores, mutexes, for example) because concurrent access is not possible

• All ‘task’ periods must be a multiple of the minor cycle time
Cyclic Executive: Disadvantages

With the approach it is difficult to:

• incorporate **sporadic processes**;

• incorporate processes with **long periods**;
  – Major cycle time is the maximum period that can be accommodated without secondary schedules (=procedure in major cycle that will call a secondary procedure every \( N \) major cycles)

• **construct** the cyclic executive, and

• handle processes with **sizeable computation** times.
  – Any ‘task’ with a sizeable computation time will need to be split into a fixed number of fixed sized procedures.
Online Scheduling

- Ready
  - task activation
  - scheduling
  - preemption
  - signal from resource
- Waiting
  - wait on busy resource
- Run
  - termination
Schedulability Test

Test to determine whether a feasible schedule exists

• **Sufficient Test**
  – If test is passed, then tasks are definitely schedulable
  – If test is not passed, tasks may be schedulable, but not necessarily

• **Necessary Test**
  – If test is passed, tasks may be schedulable, but not necessarily
  – If test is not passed, tasks are definitely not schedulable

• **Exact Test** (= *Necessary* + *Sufficient*)
  – The task set is schedulable *if and only if* it passes the test.
Rate Monotonic Analysis: Assumptions

A1: Tasks are periodic (activated at a constant rate).
   Period \( P_i \) = Intervall between two consecutive activations of task \( T_i \)

A2: All instances of a periodic task have the same computation time \( C_i \)

A3: All instances of a periodic task have the same relative deadline, which is equal to the period \( D_i = P_i \)

A4: All tasks are independent
   (i.e., no precedence constraints and no resource constraints)

Implicit assumptions:
A5: Tasks are preemptable
A6: No task can suspend itself
A7: All tasks are released as soon as they arrive
A8: All overhead in the kernel is assumed to be zero (or part of \( C_i \))
Rate Monotonic Scheduling: Principle

Principle
- Each process is assigned a (unique) priority based on its period (rate); always execute active job with highest priority.
- The shorter the period the higher the priority.
- $P_i < P_j \Rightarrow \pi_i > \pi_j$ (1 = low priority)
- W.l.o.g. number the tasks in reverse order of priority.

<table>
<thead>
<tr>
<th>Process</th>
<th>Period</th>
<th>Priority</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>25</td>
<td>5</td>
<td>T1</td>
</tr>
<tr>
<td>B</td>
<td>60</td>
<td>3</td>
<td>T3</td>
</tr>
<tr>
<td>C</td>
<td>42</td>
<td>4</td>
<td>T2</td>
</tr>
<tr>
<td>D</td>
<td>105</td>
<td>1</td>
<td>T5</td>
</tr>
<tr>
<td>E</td>
<td>75</td>
<td>2</td>
<td>T4</td>
</tr>
</tbody>
</table>
Example: Rate Monotonic Scheduling

- Example instance

- RMA - Gant chart
Example: Rate Monotonic Scheduling

\[ T_i = (P_i, C_i) \quad P_i = \text{period} \quad C_i = \text{processing time} \]

- \( T_1 = (4,1) \)
- \( T_2 = (5,2) \)
- \( T_3 = (7,2) \)

Response time of job \( J_{3,1} \)

Deadline Miss
Utilization

\[ U_i = \frac{C_i}{P_i} \quad \text{Utilization of task } T_i \]

Example: \( U_2 = \frac{2}{5} = 0.4 \)