CHALMERS

Verification

Since timeliness is such an important characteristic of a real-time system: how do we verify that the timing constraints are met for a given system implementation?

… so we don't miss that hard deadline …

… so we don't miss too many soft deadlines …

… while we at the same time avoid analyzing all possible software execution scenarios

CHALMERS

Verification

What is needed for formal verification?

- A good timing model
  - Enables expressing the timing properties of the application in a syntactically unambiguous way
  - Enables timing constraints to be reflected at all design levels: from specification level (end-to-end constraints) to implementation level
- A good schedulability analysis
  - Enables prediction of required processing capacity, e.g. # and speed of processors, of the hardware (when software is known)
  - Enables prediction of required resource usage from the software (when hardware implementation is known)

CHALMERS

Verification

What sources of uncertainty exist in formal verification?

- Non-determinism in tasks' WCET (undisturbed execution)
  - Input data and internal state controls execution paths
  - Memory access patterns control delays in processor architecture (pipelines and cache memories)
- Non-determinism in tasks' execution interference (pseudo-parallel execution)
  - Run-time execution model controls interference pattern
- Conflicts in tasks' demands for shared resources
  - (Pseudo-)parallel task execution may give rise to uncontrolled blocking of shared hardware and software resources

CHALMERS

Verification

How do we simplify formal verification?

- Concurrent and reactive programming paradigm
  - Suitable schedulable unit of concurrency (task, thread, …)
  - Language constructs for expressing application constraints for schedulable unit (priorities, delays, …)
  - WCET must be derivable for schedulable unit (special caution with usage of dynamic language constructs)
- Deterministic task execution
  - Time tables or static/dynamic task priorities
  - Preemptive task execution
  - Run-time protocols for access to shared resources (dynamic priority adjustment and non-preemptable code sections)

CHALMERS
UNIVERSITY OF TECHNOLOGY

Verification

How do we perform schedulability analysis?

- Introduce abstract models of system components:
  - Task model (computation requirements, timing constraints)
  - Processor model (resource capacities)
  - Run-time model (task states, dispatching)
- Predict whether task executions will meet constraints
  - Use abstract system models
  - Make sure that computation requirements never exceed resource capacities
  - Generate (partly or completely) run-time schedule resulting from task executions and detect worst-case scenarios

CHALMERS
UNIVERSITY OF TECHNOLOGY

Task model

Implementation → Abstract model

```
void task1(Object *self, int p) {
    Action1();
    SEND(Period1, Deadline1, self, task1, p);
}

void task2(Object *self, int p) {
    Action2();
    SEND(Period2, Deadline2, self, task2, p);
}

void kickoff(Object *self, int p) {
    AFTER(Offset1, &app1, p);
    AFTER(Offset2, &app2, p);
}

main() {
    TINYTIMBER(&app_main, kickoff, 0);
}
```

τ_1 $\quad \tau_1 = \{C_1, T_1, D_1, O_1\}$

τ_2 $\quad \tau_2 = \{C_2, T_2, D_2, O_2\}$

CHALMERS
UNIVERSITY OF TECHNOLOGY

Task model

The task model expresses the timing behavior of a task:

- The static parameters describe characteristics of a task that apply independent of other tasks.
  - Derived from the specification or implementation of the system
  - For example: period, deadline, WCET
- The dynamic parameters describe effects that occur during the execution of the task.
  - Is a function of the run-time system and the characteristics of other tasks
  - For example: start time, completion time, response time

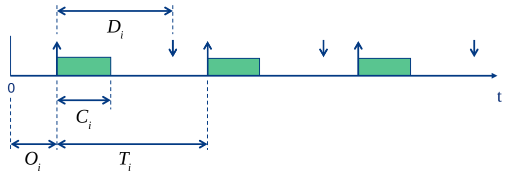
CHALMERS
UNIVERSITY OF TECHNOLOGY

Task model

Static task parameters:

τ_i $\quad \tau_i = \{C_i, T_i, D_i, O_i\}$

C_i : (undisturbed) WCET
T_i : period
D_i : (relative) deadline
O_i : (absolute) time offset



CHALMERS
UNIVERSITY OF TECHNOLOGY

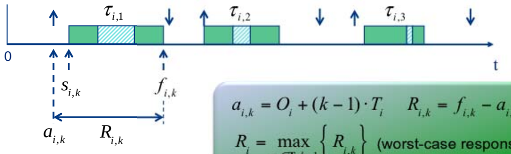
Task model

Dynamic task parameters:

τ_i $\quad \tau_i = \{ C_i, T_i, D_i, O_i \}$

$a_{i,k}$: arrival time of k^{th} instance
$s_{i,k}$: start time of k^{th} instance
$f_{i,k}$: completion time of k^{th} instance
$R_{i,k}$: response time of k^{th} instance

$\tau_{i,k}$: the k^{th} instance of τ_i



$a_{i,k} = O_i + (k-1) \cdot T_i \quad R_{i,k} = f_{i,k} - a_{i,k}$

$R_i = \max_{\tau_i \in \mathrm{T}, k \geq 1} \{ R_{i,k} \}$ (worst-case response time)

CHALMERS
UNIVERSITY OF TECHNOLOGY

Task model

Different types of tasks:

- Periodic tasks
  - A periodic task arrives with a time interval T_i
- Sporadic tasks
  - A sporadic task arrives with a time interval ≥ T_i
- Aperiodic tasks
  - An aperiodic task has no guaranteed minimum time between two subsequent arrivals

⇒ *A priori* schedulable real-time systems can only contain periodic and sporadic tasks.

CHALMERS
UNIVERSITY OF TECHNOLOGY

Processor model

Homogeneous processors:

- Identical processors
  - WCET is a constant

Heterogeneous processors:

- Uniform processors
  - WCET is the product of a basic execution time and a scaling factor
- Unrelated processors
  - WCET is not related for different processors

CHALMERS
UNIVERSITY OF TECHNOLOGY

Run-time model

Task states:

- Waiting
  - Task has not yet arrived for the first time, or has finished executing but not re-arrived
- Ready
  - Task has arrived and can potentially execute on the processor (kept waiting in a ready queue)
- Running
  - Task is currently executing on the processor

Dispatcher:

- A run-time mechanism that takes the first element (task) in the ready queue and executes it on the processor.

CHALMERS
UNIVERSITY OF TECHNOLOGY

Scheduling

- Application constraints can be met through scheduling.
- Scheduling used in many disciplines ("operations research")
  - Production pipelines
  - Real-time systems
  - Classroom scheduling
  - Airline crew scheduling
  - ...

  Schedule = resources + operations on a time line
- An important part of real-time system design is to choose a scheduling technique that generates a good schedule (that fulfills the application constraints).

CHALMERS
UNIVERSITY OF TECHNOLOGY

Evaluating a real-time system

How do we measure and compare performance?

- Quantify system performance
  - Choose useful performance measures (metrics)
- Perform objective performance analysis
  - Choose suitable evaluation methodology
  - Examples: theoretical and/or experimental analysis
- Compare performance of different designs
  - Make trade-off analysis using chosen performance measures
- Identify fundamental performance limitations
  - Find "bottleneck" mechanisms that affect performance

CHALMERS
UNIVERSITY OF TECHNOLOGY

Performance measures

"Yardsticks" by which the performance of a system is expressed.

Why do we need it?

- To objective evaluate different design solutions and choose the "best" one
- To rubberstamp a system with performance potential or quality guarantees (cf. "Intel inside", "ISO 9000")

CHALMERS
UNIVERSITY OF TECHNOLOGY

Performance measures

What is required by a performance measure?

- Must be concise to avoid ambiguity
  - preferably a single number
    - use a weighted sum of constituent local performance measures
  - should reflect user-perceived utility
    - no artificial measures should be used
  - some measures are contradictory
    - processing speed vs. power consumption in a handheld computer
  - some measures are misleading
    - MIPS (million instructions executed per second)

CHALMERS
UNIVERSITY OF TECHNOLOGY

Performance measures

What is required by a performance measure?

- Must provide efficient coding of information
  - determine relevance of individual pieces
- Must provide objective basis for ranking
  - use same set of applications for evaluations
- Must provide objective optimization criteria for design
  - identify application-sensitive criteria
- Must provide verifiable facts
  - use measures that can be derived for a real system

CHALMERS
UNIVERSITY OF TECHNOLOGY

Performance measures

Traditional performance measures:

Throughput
Average # of operations/data processed by system per time unit

Reliability
Probability that system will not fail in a given time interval

Availability
Fraction of time for which system is up (providing service)

These measures do not take deadlines into account!

CHALMERS
UNIVERSITY OF TECHNOLOGY

Performance measures

Suitable real-time performance measures:

Laxity $X = \min_{\tau_i \in \mathbf{T}}\{D_i - C_i\}$
Amount of time that the start of a task can be delayed without it missing its deadline (calculated before scheduling)

Lateness $L = \max_{\tau_i \in \mathbf{T}}\{R_i - D_i\}$
Amount of time by which a task completes after its deadline (calculated after scheduling)

Successful tasks $N_{\text{success}} = |\{\tau_i \in \mathbf{T} : R_i - D_i \le 0\}|$
Number of tasks that complete on or before their deadline (calculated after scheduling)

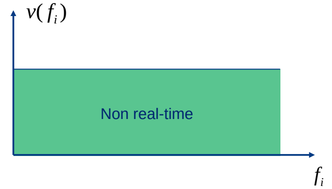
Jitter $J_{\text{output}} = \max_{\tau_i \in \mathbf{T}, k \ge 1}\{|(f_{i,k+1} - f_{i,k}) - T_i|\}$
Amount of deviation from expected periodicity of a task's completion (calculated after scheduling)

CHALMERS
UNIVERSITY OF TECHNOLOGY

Performance measures

Cost function – a general real-time performance measure

Cumulative value: $C = \sum_{\tau_i \in \mathbf{T}} v(f_i)$
Value associated with a task as a function of its completion time



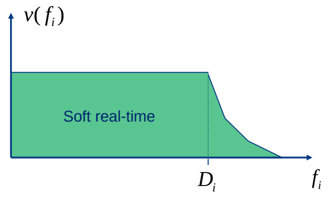
CHALMERS
UNIVERSITY OF TECHNOLOGY

Performance measures

Cost function – a general real-time performance measure

Cumulative value: $C = \sum_{\tau_i \in \mathbf{T}} v(f_i)$

Value associated with a task as a function of its completion time



CHALMERS
UNIVERSITY OF TECHNOLOGY

Performance measures

Cost function – a general real-time performance measure

Cumulative value: $C = \sum_{\tau_i \in \mathbf{T}} v(f_i)$

Value associated with a task as a function of its completion time