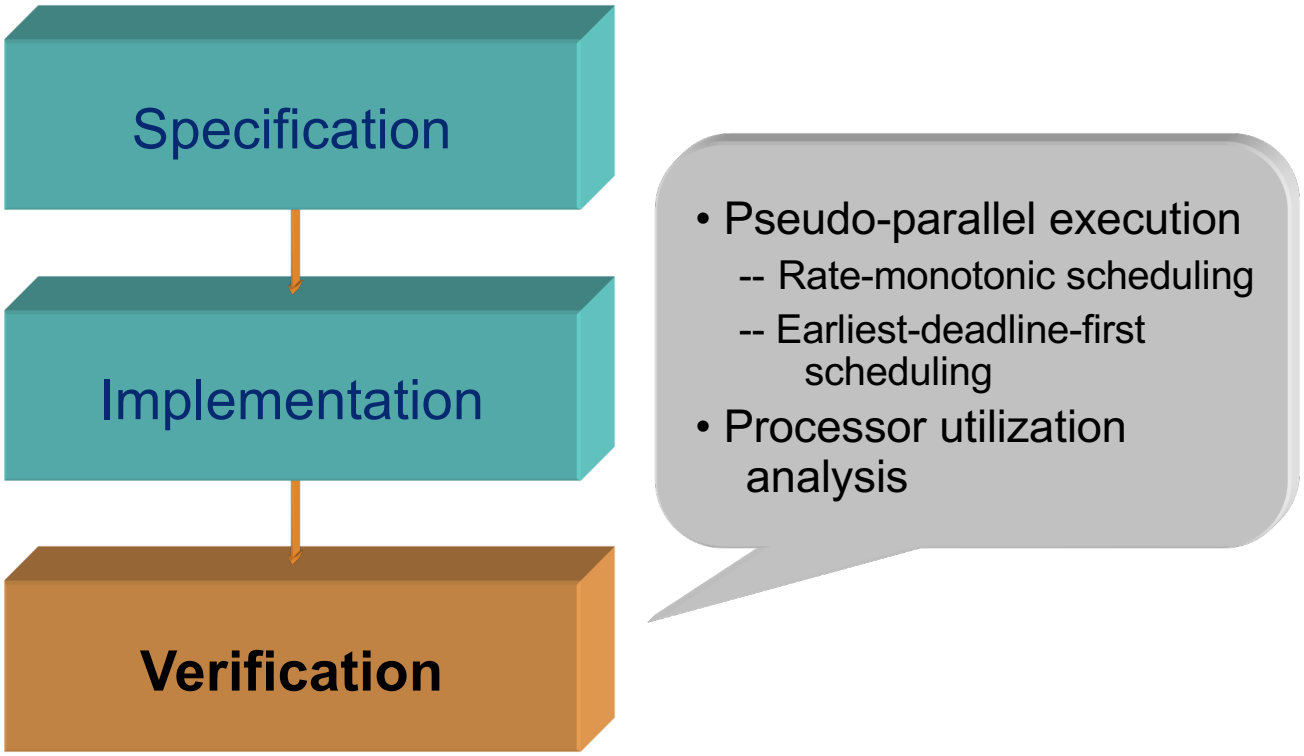
Real-Time Systems

Lecture #11

Professor Jan Jonsson

Department of Computer Science and Engineering
Chalmers University of Technology

Real-Time Systems



Pseudo-parallel execution

General properties:

- On-line schedule generation
  - Schedule determined by run-time behavior controlled by priorities or time quanta to resolve access to the processor
  - Feasibility must be tested off-line by predicting run-time behavior
- Mutual exclusion must be handled on-line
  - Support for mutual exclusion needed in run-time system (e.g., mutex objects, disabling of interrupts)
- Precedence constraints must be handled on-line
  - Dependent tasks must synchronize using semaphores, or by adding suitable time offsets

Pseudo-parallel execution

Advantages:

- High flexibility
  - Schedule can easily adapt to changes in the task set of the system environment, e.g. new tasks can be added dynamically
- External events are handled efficiently
  - I/O-based events handled via interrupt which activates a task immediately through call-back functionality
- Efficient for different types of tasks
  - Sporadic tasks can be easily supported, via suitable priority assignment
  - Scheduling algorithms are often optimal

Pseudo-parallel execution

Disadvantages:

- Complicates communication between tasks
  - Exact time of data availability is not known in advance, which requires extra synchronization between tasks
  - Task execution is difficult to adapt to existing time-slot-based network protocols (but works well with many priority-based network protocols, e.g. CAN and Token Ring)
- Task execution becomes non-deterministic
  - Temporary deviations (”jitter”) in task periodicity may occur
  - Exact feasibility tests often have high time complexity
  - Low observability (difficult to debug)

Pseudo-parallel execution

How is task scheduling done?

- Using static or dynamic priorities:
  - Ready tasks are stored in a queue, sorted by priority
  - At scheduling decisions, the task with highest priority is selected
- Using time quanta: (”round-robin”)
  - Ready tasks are stored in a circular FIFO queue
  - Each task gets access to the processor for a certain time interval (quantum); real-time clock is used for interrupting the execution
  - New scheduling decisions can be taken sooner if the executing task terminates or gets blocked

In this course, we only study pseudo-parallel execution using task priorities.

Pseudo-parallel execution

How are task priorities assigned?

- Static assignment:
  - Rate-monotonic (RM) scheduling
  - Deadline-monotonic (DM) scheduling
  - Weight-monotonic (WM) scheduling
  - Slack-monotonic (SM) scheduling
- Dynamic assignment:
  - Earliest-deadline-first (EDF) scheduling
  - Least-laxity-first (LLF) scheduling

In this course, we only study rate-monotonic, deadline-monotonic and earliest-deadline-first scheduling.

Pseudo-parallel execution

How is the scheduler implemented?

- Use a queue for the ready tasks
  - The elements in the queue (i.e., the tasks) are sorted according to task priorities; if multiple tasks have equal priority, the sorting is arbitrary (e.g., FIFO)
- The queue is updated at external or internal events
  - An external event: for example, an I/O unit generates an interrupt because data has become available at a sensor
  - An internal event: for example, a system timer generates an interrupt because a certain point in time has been reached
- Run tasks to completion
  - The scheduler normally does not terminate task executions that miss their deadlines; it is assumed that schedulability analysis has been used to verify timing correctness.

Pseudo-parallel execution

Programming with the TinyTimber kernel (p. 17):

```
int main() {
   INSTALL( &sonar, echo, IRQ_ECHO_DETECT );
   return TINYTIMBER( &sonar, tick, 0 );
}
```

TinyTimber uses both deadlines and baselines as input to its scheduling algorithm, although it is actually only the deadlines that pose any real challenge to the scheduler. However, missed deadlines are not trapped at run-time; if such behavior is desired it must be programmed by means of a separate watchdog task.

Judging whether a TinyTimber program will meet all its deadlines at run-time is an interesting problem, that can only be solved using a separate schedulability analysis and known worst-case execution times for all methods. That topic, however, is beyond the scope of the present text.

7 Summary of the TinyTimber interface

- `#include "TinyTimber.h"`

Rate-monotonic scheduling

Properties:

- Uses static priorities
  - Priority is determined by task frequency (rate): the task with the highest rate (= shortest period) receives highest priority
- Theoretically well-established
  - Sufficient feasibility test can be performed in linear time (under certain simplifying assumptions)
  - Exact feasibility test is an NP-complete problem (pseudo-polynomial time with response-time analysis)
  - RM is optimal among all scheduling algorithms that use static task priorities for implicit-deadline tasks (with $D_i = T_i$) (shown by C. L. Liu and J. W. Layland in 1973)

Earliest-deadline-first scheduling

Properties:

- Uses dynamic priorities
  - Priority is determined by how critical the task is at a given point in time: the task whose absolute deadline is earliest in time receives highest priority
- Theoretically well-established
  - Exact feasibility test can often be performed in linear time (under certain simplifying assumptions)
  - Exact feasibility test is in general an NP-complete problem (pseudo-polynomial time with processor-demand analysis)
  - EDF is optimal among all scheduling algorithms that use dynamic task priorities (shown by C. L. Liu and J. W. Layland in 1973)

Feasibility tests

What types of feasibility tests exist?

- Hyper period analysis (for any type of scheduler)
  - In an existing schedule no task execution may miss its deadline
- Processor utilization analysis (static/dynamic priority scheduling)
  - The fraction of processor time that is used for executing the task set must not exceed a given bound
- Response time analysis (static priority scheduling)
  - The worst-case response time for each task must not exceed the deadline of the task
- Processor demand analysis (dynamic priority scheduling)
  - The accumulated computation demand for the task set under a given time interval must not exceed the length of the interval

Processor utilization analysis

The utilization U for a set of periodic tasks is the fraction of the processor's capacity that is used for executing the tasks.

Since C_i / T_i is the fraction of processor time that is used for executing task τ_i the utilization for n tasks is

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i}$$

Simple feasibility test for RM

(Sufficient condition)

A sufficient condition for RM scheduling of synchronous task sets, based on the utilization U is

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq n\left(2^{1/n} - 1\right)$$

where n is the number of tasks.

This is a classic feasibility test presented by C. L. Liu and J. W. Layland in 1973.

Simple feasibility test for RM

(Sufficient condition)

Observe that it is possible to derive a conservative lower bound on utilization by letting $n \to \infty$.

$$\lim_{n \to \infty} n\left(2^{1/n} - 1\right) = \ln 2 \approx 0.693$$

This means that a set of tasks (regardless of number of tasks) whose total utilization does not exceed 0.693 is always schedulable with RM!

Simple feasibility test for RM

(Sufficient condition)

The test is valid under the following assumptions:

1. All tasks are independent
   - There must not exist dependencies due to precedence or mutual exclusion
2. All tasks are periodic or sporadic
3. All tasks have identical offsets (= synchronous task set)
4. Task deadline equals the period (= implicit-deadline tasks)
5. Task preemptions are allowed

Simple feasibility test for RM

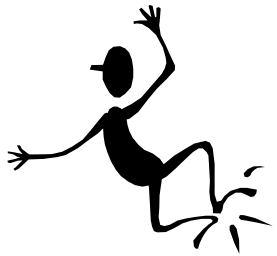
(Sufficient condition)

The proof of the test includes the following observation:

> The response time for a task is maximized at a special task-arrival pattern, referred to as the critical instant.

The feasibility test is derived using an analysis of this special case. It is shown that if the task set is schedulable for the critical instant case, it is also schedulable for any other case.

NOTE: For single-processor systems (assumed in this proof) the critical instant occurs when the analyzed task arrives at the same time as all tasks with higher priority.



Simple feasibility test for EDF

(Sufficient and necessary condition)

A sufficient and necessary condition for EDF scheduling of synchronous task sets, based on the utilization U is

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1$$

where n is the number of tasks.

This is another classic feasibility test presented by C. L. Liu and J. W. Layland in 1973. The test is exact!

Simple feasibility test for EDF

(Sufficient and necessary condition)

The test is valid under the following assumptions:

1. All tasks are independent
   - There must not exist dependencies due to precedence or mutual exclusion
2. All tasks are periodic or sporadic
3. All tasks have identical offsets (= synchronous task set)
4. Task deadline equals the period (= implicit-deadline tasks)
5. Task preemptions are allowed

Example: RM/EDF scheduling

Problem: Assume a system with tasks according to the figure below. The timing properties of the tasks are given in the table. Consider scheduling the tasks using rate-monotonic (RM) and earliest-deadline-first (EDF) scheduling, respectively.

a) What is the utilization of the task set?

b) What do Liu & Layland's feasibility tests for RM and EDF say?

c) Are the tasks are schedulable using RM and EDF, respectively?

Task	C_i	O_i	T_i
τ_1	1	0	3
τ_2	1	0	4
τ_3	1	0	5