Parallel & Distributed Real-Time Systems

Lecture #6

Professor Jan Jonsson

Department of Computer Science and Engineering
Chalmers University of Technology

Administrative issues

Course evaluation:

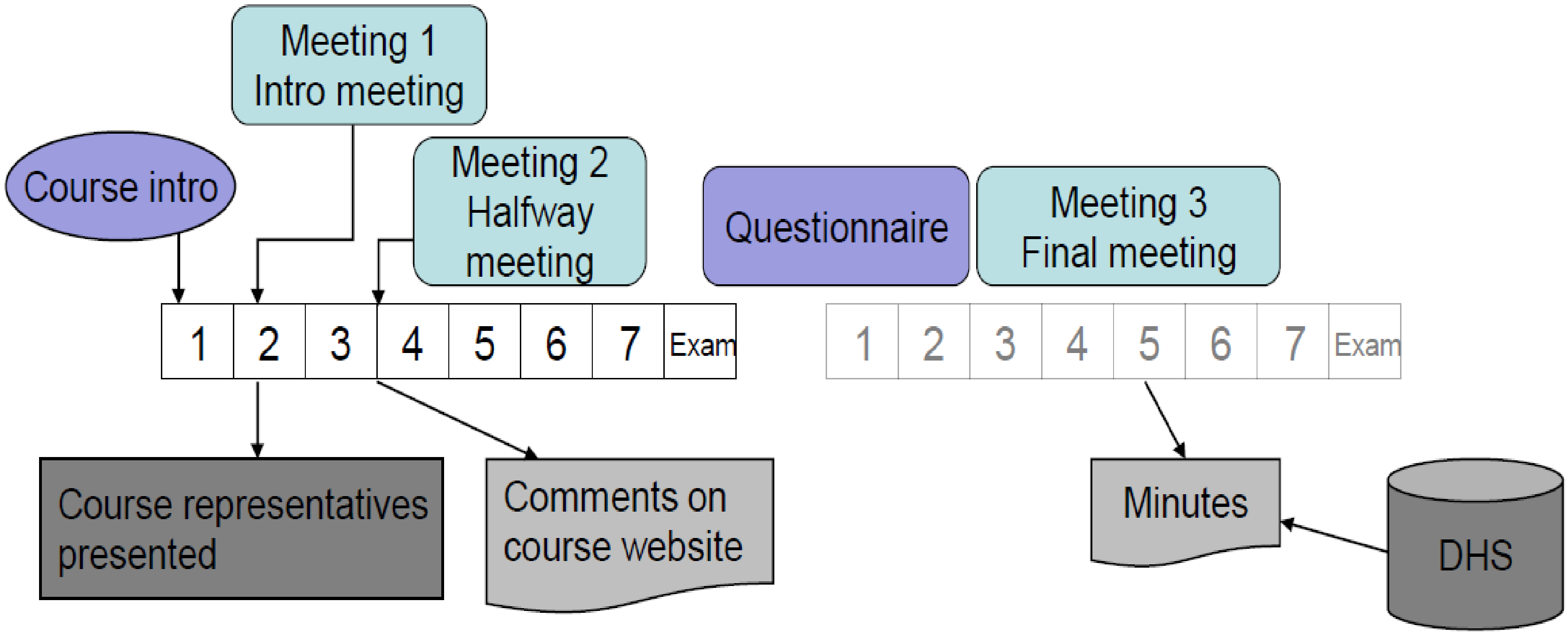
- The following students have been selected to be course representatives:
    - Johan Gustafsson (MPCSN)
    - Fredrik Hidstrand (MPCSN)
    - Henrik Hugo (MPCSN)
    - Emil Lindqvist (MPEES)
    - Eyþór Sigmundsson (MPCSN)

Please contact them whenever you have comments or suggestions for improvements. Contact information is available on the course home page.

Administrative issues

Course evaluation:

- What's in the procedure:



Feasibility testing

What techniques for feasibility testing exist?

- Hyper-period analysis (for static and dynamic priorities)
  - In a simulated schedule no task execution may miss its deadline
- Guarantee bound analysis (for static and dynamic priorities)
  - The fraction of processor time that is used for executing the task set must not exceed a given bound
- Response time analysis (for static priorities)
  - The worst-case response time for each task must not exceed the deadline of the task
- Processor demand analysis (for dynamic priorities)
  - The accumulated computation demand for the task set under a given time interval must not exceed the length of the interval

Feasibility testing

What techniques for feasibility testing exist?

- Hyper-period analysis (for static and dynamic priorities)
  - In a simulated schedule no task execution may miss its deadline
- Guarantee bound analysis (for static and dynamic priorities)
  - The fraction of processor time that is used for executing the task set must not exceed a given bound
- Response time analysis (for static priorities)
  - The worst-case response time for each task must not exceed the deadline of the task
- Processor demand analysis (for dynamic priorities)
  - The accumulated computation demand for the task set under a given time interval must not exceed the length of the interval

Response-time analysis

Response time:

- The response time R_i for a task τ_i represents the worst-case completion time of the task when execution interference from other tasks are accounted for.
- The response time for a task τ_i consists of:

  C_i The task's uninterrupted execution time (WCET)

  I_i Interference from higher-priority tasks

$$R_i = C_i + I_i$$

Response-time analysis

Interference:

- For static-priority scheduling, the interference term is

$$I_i = \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

where $hp(i)$ is the set of tasks with higher priority than τ_i.

- The response time for a task τ_i is thus:

$$R_i = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

Response-time analysis

Response-time calculation:

- The equation does not have a simple analytic solution.
- However, an iterative procedure can be used:

$$R_i^{n+1} = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i^n}{T_j} \right\rceil C_j$$

- The iteration starts with a value that is guaranteed to be less than or equal to the final value of R_i (e.g. $R_i^0 = C_i$)
- The iteration completes at convergence ($R_i^{n+1} = R_i^n$) or if the response time exceeds the deadline D_i

Response-time analysis

Schedulability test: (Joseph & Pandya, 1986)

- An exact condition for static-priority scheduling is

$$\forall i: R_i \leq D_i$$

- The test is only valid if all of the following conditions apply:
  1. Single-processor system
  2. Synchronous task sets
  3. Independent tasks
  4. Periodic tasks
  5. Tasks have deadlines not exceeding the period $(D_i \leq T_i)$

Response-time analysis

Time complexity:

Response-time analysis has pseudo-polynomial time complexity

Proof:

- calculating the response-time for task τ_i requires no more than D_i iterations
- since $D_i \leq T_i$ the number of iterations needed to calculate the response-time for task τ_i is bounded above by $Q_i^{\max} = T_i$
- the procedure for calculating the response-time for all tasks is therefore of time complexity $O(\max\{T_i\})$
- the longest period of a task is also the largest number in the problem instance

Response-time analysis

Accounting for blocking:

- Blocking caused by critical regions
  - Blocking factor B_i represents the length of critical region(s) that are executed by processes with lower priority than τ_i
- Blocking caused by non-preemptive scheduling
  - Blocking factor B_i represents largest WCET (not counting τ_i)

$$R_i = C_i + B_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

Observation: the feasibility test is now only sufficient since the worst-case blocking will not always occur at run-time.

Response-time analysis

Accounting for blocking: (using PCP or ICPP)

- When using priority ceiling a task τ_i can only be blocked once by a task with lower priority than τ_i .
- This occurs if the lower-priority task is within a critical region when τ_i arrives, and the critical region's ceiling priority is higher than or equal to the priority of τ_i .
- Blocking now means that the start time of τ_i is delayed (= the blocking factor B_i)
- As soon as τ_i has started its execution, it cannot be blocked by a lower-priority task.

Response-time analysis

Accounting for blocking: (using PCP or ICPP)

Determining the blocking factor for τ_i

1. Determine the ceiling priorities for all critical regions.

2. Identify the tasks that have a priority lower than τ_i and that calls critical regions with a ceiling priority equal to or higher than the priority of τ_i .

3. Consider the times that these tasks lock the actual critical regions. The longest of those times constitutes the blocking factor B_i .

Processor-demand analysis

Processor demand:

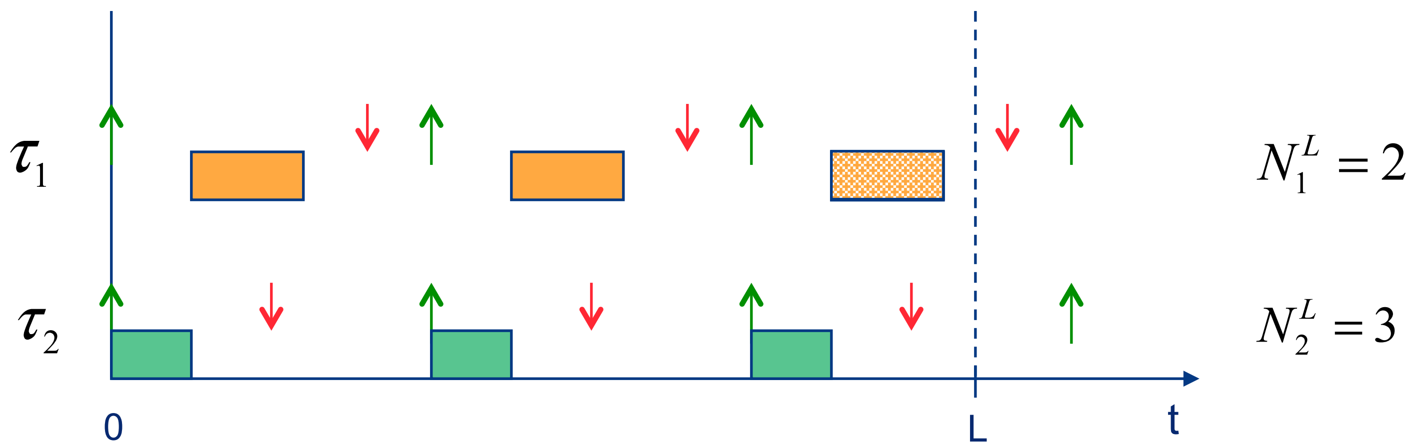
- The processor demand for a task τ_i in a given time interval $[0, L]$ is the amount of processor time that the task needs in the interval in order to meet the deadlines that fall within the interval.
- Let N_i^L represent the number of instances of τ_i that must complete execution before L .
- The total processor demand up to L is

$$C_P(0, L) = \sum_{i=1}^{n} N_i^L C_i$$

Processor-demand analysis

Number of relevant task arrivals:

- We can calculate N_i^L by counting how many times task τ_i has arrived during the interval $[0, L - D_i]$.
- We can ignore instance of the task that has arrived during the interval $[L - D_i, L]$ since $D_i > L$ for these instances.



Processor-demand analysis

Processor-demand analysis:

- We can express N_i^L as

$$N_i^L = \left\lfloor \frac{L - D_i}{T_i} \right\rfloor + 1$$

- The total processor demand is thus

$$C_P(0, L) = \sum_{i=1}^{n} \left(\left\lfloor \frac{L - D_i}{T_i} \right\rfloor + 1 \right) C_i$$

Processor-demand analysis

Schedulability test: (Baruah et al., 1990)

- A sufficient and necessary condition for EDF scheduling is

$$\forall L \in K : C_P(0, L) \leq L$$

- The test is only valid if all of the following conditions apply:
  1. Single-processor system
  2. Synchronous task sets
  3. Independent tasks
  4. Periodic tasks
  5. Tasks have deadlines not exceeding the period ($D_i \leq T_i$)

Processor-demand analysis

Schedulability test: (Baruah et al., 1990)

- The set of control points K is

$$K = \left\{ D_i^k \,\middle|\, D_i^k = kT_i + D_i,\ D_i^k \le L_{\max},\ 1 \le i \le n,\ k \ge 0 \right\}$$

$$L_{\max} = \max\left\{ D_1, \ldots, D_n, \frac{\sum_{i=1}^{n} (T_i - D_i)\, U_i}{1-U} \right\}$$

Observation:

$$L_{\max} \le \max\left\{ \max\{D_i\}, \frac{U}{1-U} \max\{T_i - D_i\} \right\} \le \max\left\{ \max\{T_i\}, \frac{U}{1-U} \max\{T_i\} \right\}$$

Processor-demand analysis

Time complexity:

Processor-demand analysis has pseudo-polynomial time complexity if total task utilization is less than 100%

Proof:

- the number of control points needed to check the processor demand is bounded above by

$$Q_L^{\max} = \max\left\{\max\{T_i\}, \frac{U}{1-U}\max\{T_i\}\right\} = \max\left\{1, \frac{U}{1-U}\right\} \bullet \max\{T_i\}$$

- since $U/(1-U)$ is a constant the procedure for calculating the processor demand is therefore of time complexity $O(\max\{T_i\})$
- the longest period of a task is also the largest number in the problem instance

Processor-demand analysis

Accounting for blocking: (using Stack Resource Policy)

Tasks are assigned static preemption levels:

- The preemption level of task τ_i is denoted π_i
- Task τ_i is not allowed to preempt another task τ_j unless $\pi_i > \pi_j$
- If τ_i has higher priority than τ_j and arrives later, then τ_i must have a higher preemption level than τ_j .

Note:

- The preemption levels are static values, even though the tasks priorities may be dynamic.
- For EDF scheduling, suitable levels can be derived if tasks with shorter relative deadlines get higher preemption levels, that is:

$$\pi_i > \pi_j \quad \Leftrightarrow \quad D_i < D_j$$

Processor-demand analysis

Accounting for blocking: (using Stack Resource Policy)

Resources are assigned dynamic resource ceilings:

- Each shared resource is assigned a ceiling that is always equal to the maximum preemption level among all tasks that may be blocked when requesting the resource.
- The protocol keeps a system-wide ceiling that is equal to the maximum of the current ceilings of all resources.
- A task with the earliest deadline is allowed to preempt only if its preemption level is higher than the system-wide ceiling.

Note:

- The original priority of the task is not changed at run-time.
- The resource ceiling is a dynamic value calculated at run-time as a function of current resource availability.

Processor-demand analysis

Accounting for blocking: (using Stack Resource Policy)

- Blocking factor B_i represents the length of critical / non-preemptive regions that are executed by tasks with lower preemption levels than τ_i
- Tasks are indexed in the order of increasing preemption levels, that is: $\pi_i > \pi_j \Leftrightarrow i < j$

$$\forall L \in K, \forall i \in [1, n] : C_P^i(0, L) \leq L$$

$$C_P^i = \sum_{k=1}^{i} \left(\left\lfloor \frac{L - D_k}{T_k} \right\rfloor + 1 \right) C_k + \left(\left\lfloor \frac{L - D_i}{T_i} \right\rfloor + 1 \right) B_i$$

Processor-demand analysis

Accounting for blocking: (using Stack Resource Policy)

Determining the blocking factor for τ_i

1. Determine the worst-case resource ceiling for each critical region, that is, assume the run-time situation where the corresponding resource is unavailable.

2. Identify the tasks that have a preemption level lower than τ_i and that calls critical regions with a worst-case resource ceiling equal to or higher than the preemption level of τ_i.

3. Consider the times that these tasks lock the actual critical regions. The longest of those times constitutes the blocking factor B_i.