TTIT62 Real-time Process Control

Lecture 5: Scheduling (II)

Simin Nadjm-Tehrani

Real-time Systems Laboratory

Department of Computer and Information Science
Linköping university (*)

(*) Thanks to Calin Curescu for improvements to some slides

Recall from last lecture

- Rate monotonic scheduling
  - Preemptive
  - Periods and priorities are static
  - WCET known for all tasks
  - Decision is on-line

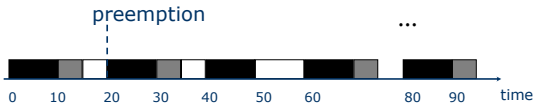
Recall: Example

■ P1 □ P2 ▒ P3

Period (Ti)	20	50	30
WCET (Ci)	10	10	5
Priority	high	low	medium

Consider following scenario:

arrival time	process
0	P1, P2, P3
20	P1
30	P3
40	P1
50	P2
60	P1, P3



Schedulability test

Theorem: (sufficient condition)

For ***n*** processes, RMS will guarantee their schedulability if the total utilisation
$U = C_1/T_1 + \ldots + C_n/T_n$
does not exceed the guarantee level
$G = n\,(2^{1/n} - 1)$

For this example

U = 10/20+10/50+5/30 = 0,87

n = 3 $\Rightarrow$ G = $3(2^{1/3} - 1)$ = 0,78

Schedulability is not guaranteed!

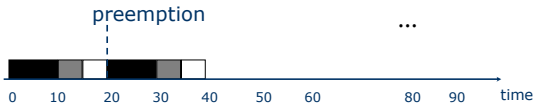
(but processes may still meet their deadlines...)

When the test fails

- Try with the critical instant: Assume that all processes are released simultaneously at time 0, and then arrive according to their periods
- Check whether each process meets its deadline for all releases before the first deadline for the process with lowest priority

In the given case:

arrival time	process
0	P1, P2, P3
20	P1
30	P3
40	P1
50	P2
60	P1, P3



Response time analysis

- Response time: the time between the release and the completion time
- Mathematical equations for computing worst case response times R_i for each process
- Exact analysis: Process set schedulable if $R_i \leq T_i$ for all processes

Response time analysis

- Tasks suffer interference from higher priority tasks

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

- Iterative formula for calculating response time

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

- Assumptions?

Otherwise...

- Change U by reducing C_i (code optimisation, faster processor, ...)

or

- Increase T_i for some process (can one do this? Which one?)

Intuitively...

Utilisation based test:

$$G = n \left(2^{1/n} - 1 \right)$$

For a given *n*, the highest ceiling under which we only find schedulable task sets

(irrespective of release times, with all possible C_i, T_i)

Theorems

- **Optimality**: RMS is optimal among methods with fixed priority
  (in what sense?)
- **Lowest upper bound**: For arbitrarily large *n*, it suffices that processor utilisation is < 0.69

[Nice proofs in Buttazzo book]

Example (2)

	P_1	P_2	P_3
Period (T_i)	20	50	30
WCET (C_i)	7	10	5

U = 7/20 + 10/50 + 5/30 = 0,72
>0,69 but...
< G = 0,78

The schedulability of this task set is guaranteed!

Dynamic priorities

- To deal with
  - processes with long T_i and short deadline
  - Process dependencies: when processes share resources and must be synchronised
- We will look at two methods that change priorities dynamically

Earliest deadline first (EDF)

- Preemptive
- Online decision
- Dynamic priorities

Policy: Always run the process that is *closest* to its deadline

Parameters and conditions

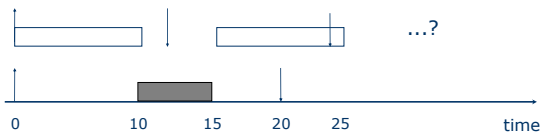
- The process with nearest absolute deadline (d_i) will run first

For pre-analysis:

- Events that lead to release of P_i appear with minimum inter-arrival interval T_i
- P_i has a WCET C_i
- The process must be finished before its deadline $D_i \leq T_i$
- Processes are independent (do not share resources)

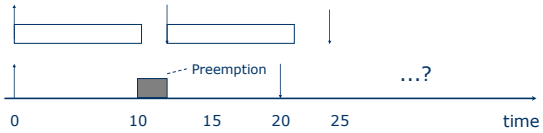
Example (3)

Consider following processes:	P_1	P_2
WCET (C_i)	5	10
Deadline ($D_i = T_i$)	20	12
Arrival times (r_i)	0, 20,...	0, 12,...



Compare to RMS

For same task set:	P_1	P_2
WCET (C_i)	5	10
Deadline ($D_i = T_i$)	20	12
Arrival times (r_i)	0, 20,...	0, 12,...



Theorem

A set of *periodic* tasks $P_1,...,P_n$ for which $D_i = T_i$ is schedulable with EDF **iff**
$U = C_1/T_1+...+C_n/T_n \leq 1$

For Example 3:
$C_1/T_1 + C_2/T_2 = 5/20 + 10/12 = 1,08!$

Example (4)

Consider following task set:	P_1	P_2
WCET (C_i)	2	4
Deadline ($D_i = T_i$)	5	7

Is it schedulable?

$U = 2/5 + 4/7 = 0,97$

Yes!

Under which assumptions?

EDF vs. RMS

- EDF gives higher processor utilisation (Example 4 not schedulable with RMS!)
- EDF has simpler exact analysis
- RMS can be implemented to run faster at run-time (ignoring time for context switching)

Sharing resources

- Assume that processes synchronise using semaphores
- We schedule the processes with fixed priorities but relax the independence requirement

Priority Inversion

- A low priority process (P_1) locks the resource
- A high priority process (P_2) has to wait on the semaphore (blocked state)
- A medium priority process (P_3) preempts P_1 and runs to completion before P_2!

How to avoid it?

- When P_2 is blocked by P_1 one raises the priority of P_1 to the same level as P_2 temporarily
- Afterwards, when the semaphore is released by P_1, it goes back to its prior priority level
- P_3 can not interrupt P_1 any more!

Priority inheritance

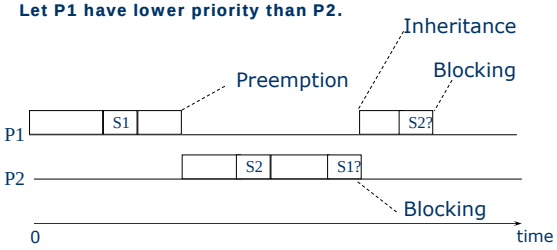
- Is transitive
- Guarantees upper bound for blocking time, since high priority process P_2 is blocked only under the time that P_1 uses the resource

But ...

Does not avoid deadlock!

Example

Let P1 have lower priority than P2.



Here Si denotes the process locks semaphore Si.

Terminology

Note that:

- *blocked* – when waiting for a resource (other than CPU)
- *not dispatched* or *preempted* - when waiting for CPU

Ceiling Protocols

e.g. Immediate priority Ceiling Protocol (ICP):

- A process that obtains its first resource inherits the *resource's ceiling priority* - the highest priority among all processes that can possibly claim that resource
- Dynamic priority for a process is the max of own (fixed) priority and the ceiling values of all resources it has locked
- When a resource is released, the process priority returns to the normal level (or to another engaged resource's ceiling)

Properties

- A process is blocked max once by another process with lower priority
- The blocking delay is a function of the length of the critical section
- Do not even need to use semaphores!

ICP & Deadlock

- The ICP prevents deadlocks (How?)
- Moreover, it prevents starvation (How?)