TDDD07 – Real-time Systems Lecture2: Scheduling II

Simin Nadjm-Tehrani

Real-time Systems Laboratory Department of Computer and information Science



Recap from last lecture

- Real-time systems have well-defined timing requirements, some soft/hard
- We look at the extreme case
 - Computational systems that need to meet every deadline for every instance of each process



Release times

- Reading and reacting to continuous signals
 Periodicity
- Recognising/reacting to some *aperiodic* events
 - Minimum inter-arrival time
 - Sporadic processes



Computation time (WCET)

- Depends on
 - Hardware
 - Application code (algorithm)
 - Compiler
 - Data



Cyclic scheduling



Cyclic scheduling

- A schedule is created based on statically known and fixed parameters (period, WCET)
- Off-line decision on which task runs & when it runs
 - When executing: Run the processes in predetermined order using a table look-up
- To run processes in the "right" frequency find
 - Minor cycle
 - Major cycle



Example (1)

Consider following processes: P_1 P_2 Period(Ti)/Deadline 50 100

Worst case execution time (Ci)1030

Note: repetition!





```
A cyclic executive
every major cycle do{
   read all in signals;
   run minor cycle 1 processes;
   wait for interrupt;
                                   End of minor cycle
   write all out signals;
   . . .
   read all in signals;
   run minor cycle n processes;
   wait for interrupt;
   write all out signals;
                                   End of minor cycle
```

Finding Minor/Major Cycle

```
First try:
Minor cycle: greatest common divisor
(sv. sgd)
```

```
Major cycle: least common multiplier
(sv. mgm)
```







Construction of a cyclic schedule

Off-line analysis in order to fix the schedule might be iterative

- Each process P_i is run periodically every T_i (i.e. should be completed once every T_i)
- Processes are placed in *minor cycle* and *major cycle* until repetition appears
- Check: Are all process instances runnable with the given periods and estimated WCET?
- If not, reconsider the minor/major cycle and/or some process parameters



When is the schedule correct?

- All processes should be run at least as often as every (original) $\rm T_{i}$
- All processes fit in the minor cycles they are placed in

and

• Repetition appears!



Harmonic processes

• Easy to find minor/major cycle









Next try...

- In either case we need to
 - change the task set parameters
 - recall that all processes should be run at least as often as every (original) T_i
- Place the processes in *new* minor cycle and major cycle until repetition appears
- If there is no option, $T_{i}\ \text{can}\ \text{be}\ \text{increased}\ \text{in}\ \text{cases}\ \text{where}\ \text{the}\ \text{application}\ \text{allows}\ \text{it}$





Alternative 1: Choose minor cycle as greatest common divisor, and move processes in time when they clash.





Drawbacks?

Jitter control

• Many applications need to minimise jitter in reading data from sensors or producing output to actuators



Example (3.2)

process period



Alternative 2: Run process B more often than necessary, e.g. once every 75 time units.







Alternative 3: A mix of the last two

Drawbacks?





If they don't fit?

• Break some process that does not fit into two or more processes and run the different parts in different mino cles

Creates new processes out of the old one!



Note: No preemption

Process A B	Period 25 25	Comp. Time 10 8	loop Wait_For_Interrupt; Procedure_For_ A ; Procedure_For_ B ; Procedure_For_ C ;	Wait_For_Interrupt; Procedure_For_ A ; Procedure_For_ B ; Procedure_For_ C ;
С	50	5	Wait_For_Interrupt;	Wait_For_Interrupt;
D	50	4	Procedure_For_ B ;	Procedure_For_ B ;
Е	100	2	Procedure_For_ D ; Procedure_For_ E ;	Procedure_For_ D ; end loop;

Interropt			Interropt				Interrupt			ł	Interrupt			
А	В	С	A	В	D	Е	А	В	С		А	В	D	

Time



Now let's check!

You should be able to answer how the typical scheduling questions are answered in a cyclic schedule context ...

During run-time:

- What is the de facto "deadline" for each process?
- How does one know that processes meet their deadlines?
- What happens if they don't?



What if dependent?

- So far we assumed all processes are independent
- Dependence can be due to sharing resources or computation precedence requirements
- In a cyclic schedule:
 - Computation precedence automatically taken care of as each instance of a process reads the inputs at the beginning of a minor cycle (produced by another process at the end of some prior minor cycle)
 - Mutual access to resources does not take place as each process is running alone with no interruptions



Summary

- Cycles can be hard to determine and can become looong ...
- Long WCET can create problems
- Sporadic processes are run periodically
 - Can lead to high processor utilisation
- Very inflexible!



But...

- Simple at run-time
- No overheads for context switching
- Processes can exchange data without the need for explicit (dynamic) synchronisation



Better methods needed

For:

- Processes with long WCET
- Sporadic events
- Processes with long period but short deadline
- Run-time process dependence
 - specially in terms of overruns



Priority-based scheduling

Rate-Monotonic Scheduling (RMS)



Priority-base scheduling

• A preemptive method where the priority of the process determines whether it continues to run or it is disrupted

"Most important process first!"



Rate Monotonic Scheduling (RMS)

- On-line
- Preemptive
- Priority-based with fixed (static) priorities



Priorities

- Each process has a period T_i that is the shortest interval between its release times
- Processes are assigned priorities dependent on length of $\mathrm{T_{i}}$
 - The shorter T_i the higher the priority



Example (4)

P1 P2 P3

Period (Ti)205030WCET (Ci)10105Priorityhighlowmedium

Let's assume Di = Ti



Autumn 2024

Consider following scenario:



Schedulability Tests

- Sufficient
 - if test is passed, then tasks are definitely schedulable
 - if test is not passed, we don't know
- Necessary
 - if test is not passed, tasks are definitely not schedulable
 - if test is passed, we don't know
- Exact test:
 - sufficient & necessary at the same time



RMS Schedulability

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 the example (4)

$$U = 10/20 + 10/50 + 5/30 = 0.87$$

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

Schedulability is not guaranteed!

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



When the test fails

- Try testing 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 of the process with lowest priority



Example (4) - Revisited

P1 P2 P3

Period (Ti) WCET (Ci) Priority

20 50 3010 10 5high low medium



Autumn 2024

Consider the scenario for example 4:



Exact schedulability test

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



Response time analysis

• Tasks suffer interference from higher priority tasks

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

- Iterative formula for calculating response time
- Assumptions?

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

[Joseph & Pandya 1986]



Not schedulable task set

When **response time analysis** gives a "no" answer:

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

or

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



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





What does the test mean?

Utilisation based test: $G = n (2^{1/n} - 1)$

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)



Example (5)

$$\begin{array}{cccccccccc} P_{1} & P_{2} & P_{3} \\ Period (T_{i}) & 20 & 50 & 30 \\ WCET (C_{i}) & 7 & 10 & 5 \end{array}$$

The schedulability of this task set is guaranteed!



Better methods needed

For:

- Processes with long WCET
- Sporadic events
- Processes with long period but short deadline
- Run-time process dependence
 - specially in terms of overruns





Summarising RMS

- Processes with long WCET
 - RMS does not require splitting the code
- Sporadic events
 - RMS only runs them when they arrive
- Processes with long period but short deadline
 - Can allocate fixed priorities based on deadlines for the cases $D_i \le T_i$ Deadline monotonic scheduling
- Run-time process dependence
 - Overruns: highest priority task not affected!

How about mutual access?

Priority-based scheduling

Earliest Deadline First (EDF)



Dynamic priorities

• Next: We look at regimes that change priorities dynamically



Earliest deadline first (EDF)

- Online decision
- Preemptive
- Dynamic priorities

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



Assumptions on process set

- Event that leads to release of process P_i appears with minimum inter-arrival interval T_i
- Each P_i has a max computation time C_i
- The process must be finished before its relative deadline $D_i \leq T_i$
- Processes are independent (do not share resources other than CPU)
- **EDF:** The process with nearest absolute deadline (d_i) will run first



Preparatory reading

- Background reading on deadlocks (announced on the web, see Chapter in Silberschatz, Galvin & Gagne)
- Specially important if you do not recall the deadlock related notions as part of your earlier OS course!
 - Deadlock prevention, avoidance, detection





www.ida.liu.se/~TDDD07

