TTIT62 Real-time Process Control

Lecture 4: Scheduling (I)

Simin Nadjm-Tehrani

Real-time Systems Laboratory

Department of Computer and Information Science
Linköping university (*)

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

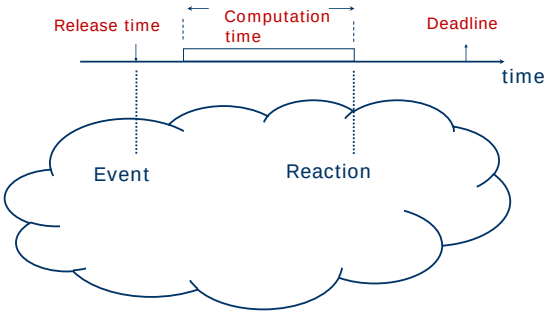
From lecture 1

- Scheduling: Allocating resources among computational processes, especially the CPU time
- Real-time scheduling: guarantees timing requirements
  - Hard RTS: are all deadlines met?
  - Soft RTS: what % of deadlines are met? (miss ratio)

Scheduling algorithms

- The scheduler decides which process is to be executed next

- Inputs: parameters for each process



Scheduling

- Performed off-line or on-line
- With information available statically or dynamically
- Preemptive or non-preemptive

Which parameters?

- How to find the maximum computation time for each process?
- How to determine deadlines?
  - This course: Control applications dictate the deadline through physics of the application
- When (how often) is a process released?

Release times

- Periodic tasks: when reading and reacting to continuous signals
  - Most processes in this course
- Sporadic tasks: when recognising/reacting to some aperiodic events
  - Use minimum inter-arrival time

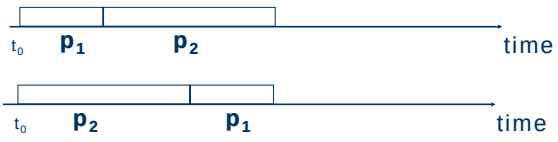
Schedulability Analysis

- To confirm that the deadlines for a *set of processes* can be guaranteed using a given policy
- The process set is then considered "schedulable"

Order matters!

Consider following processes: p_1 p_2

Computation time (C_i)	5 ms	10 ms
Deadline (D_i)	20 ms	12 ms



Schedulability Test

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

Cyclic scheduling

- An off-line schedule is created based on static analysis and fixed parameters
- When executing: Run the processes in pre-determined order using a table look-up.
- To run processes in the "right" frequency find
  - Minor cycle
  - Major cycle

Cyclic Executive

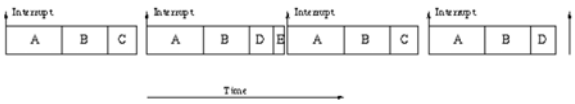
Process	Period	Comp. Time
A	25	10
B	25	8
C	50	5
D	50	4
E	100	2

```
loop
    Wait_For_Interrupt;
    Procedure_For_A;
    Procedure_For_B;
    Procedure_For_C;

    Wait_For_Interrupt;
    Procedure_For_A;
    Procedure_For_B;
    Procedure_For_D;
    Procedure_For_E;

    Wait_For_Interrupt;
    Procedure_For_A;
    Procedure_For_B;
    Procedure_For_C;

    Wait_For_Interrupt;
    Procedure_For_A;
    Procedure_For_B;
    Procedure_For_D;
end loop;
```

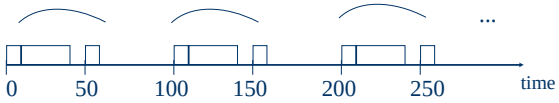


Example (1)

Consider following processes: P_1 P_2

Period/Deadline(Ti)	50	100
Worst case execution time (Ci)	10	30

Note: repetition!



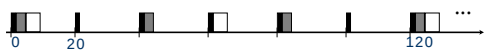
Finding Minor/Major Cycle

Minor cycle: greatest common devisor (sv. sgd)

Major cycle: least common multiplier (sv. mgn)

Example (2):

process	A	B	C
period	20	40	60



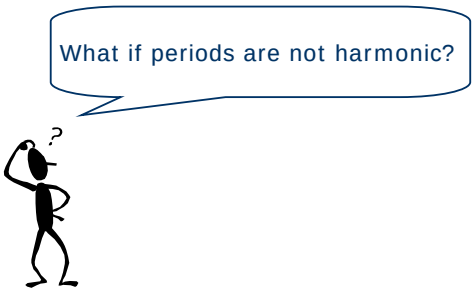
Iterative construction

- Off-line analysis in order to fix the schedule might be iterative
  - Check: Will the chosen minor/major cycle work with the natural periods and computation times?
  - Do the processes fit in?
  - Otherwise, change the parameters!
  - Which parameters can we change?

Harmonic processes

Recall example 2:

process	A	B	C
period	20	40	60



How to construct?

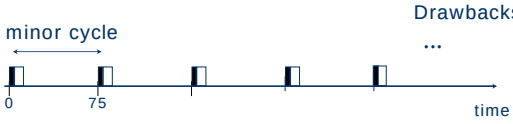
- Recall: In a cyclic schedule processes are intended to output their results periodically
- If the periods are not harmonically related
  - change the periods
  - all processes should be run *at least* as often as every (original) T_i
- Place the processes in *minor cycle* and *major cycle* until repetition appears

Example (3.1)

process	A	B
period	75	100

Alternative 1:
Run process B more often than necessary, e.g. once every 75 time units.

Drawbacks?
...



Example (3.2)

process	A	B
period	75	100

Alternative 2:
Choose minor cycle as greatest common divisor, and move processes backwards in time when they clash.

Drawbacks?
...

Jitter control

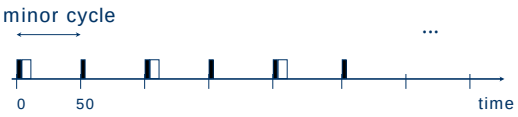
- Many applications need to minimise jitter in reading data from sensors or producing output to actuators

Example (3.3)

process	A	B
period	75	100

Alternative 3:
A mix of the last two

Drawbacks?
...



Schedulability test

- Sum of processes' execution times (WCET) in each minor cycle is less than the cycle's length, and processes run at the "right" frequency

If succeeded:

- the schedule, whose length corresponds to the major cycle, is repeated for all executions

If they don't fit?

- Break some process that does not fit into two or more processes and run the different parts in different minor cycles

Drawbacks?

What if dependent?

- So far we assumed all processes are independent
- Dependence can be due to sharing resources or computation precedence requirements
- In either case, the fixed order has to respect dependencies

Summary

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

But...

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

Exercises

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

Better methods needed

For:

- Processes with long WCET
- Sporadic events
- Processes with long period but short deadline
- Process dependence (sharing resources and overruns)

Priority-based scheduling

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

"Most important process first!"

RMS

Rate Monotonic Scheduling:

- On-line
- Preemptive
- Priority-based with static priorities

Priorities

- Each process has a period T_i that is the shortest interval between its arrival times
- Processes are assigned priorities dependent on length of T_i, the shorter T_i the higher the priority

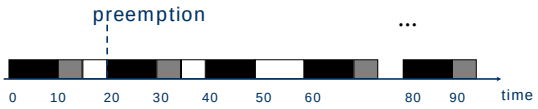
Example (4)

■ 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

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