TDDC47: Real-time and Concurrent Programming
Lecture 5: Real-time Scheduling (I)

Simin Nadjm-Tehrani
Real-time Systems Laboratory
Department of Computer and Information Science
Linköping University

Evaluation actions

- After muddy cards:
  - Made adjustments to schedule for a better synchronisation lecture-lesson

Recall: course overview

- The notion of Process and related concepts (3,5 lectures)
  - Resource sharing & Synchronisation
  - Deadlocks, livelocks, and starvation
- Real-time Resource allocation: scheduling (2,5)
- Real-time communication networks (1)
- Fault management and dependability(1)
- Guest lecture from industry
- RE: Summary and on-demand question session

This lecture

- Introduction to Real-time systems
- CPU as a resource: Scheduling
  - Cyclic scheduling
  - Rate monotonic scheduling

Real-time processes

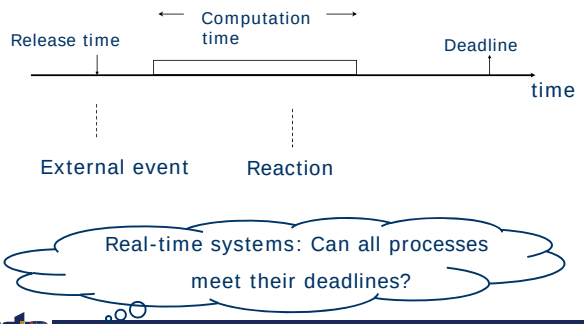
- In (desktop) operating systems scheduler's role is to ensure that each process gets a share of the CPU
  - Lab2 in the course
- With real-time systems it is not enough that processes get a share some time

  The time that the result of the computation is delivered is as important as the result itself
- Predictability!

Predictable is not "fast"!

The film…

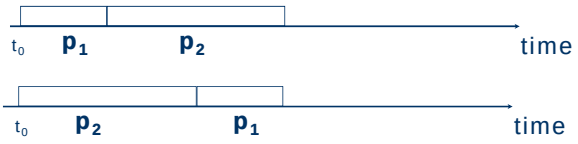
What is meant by predictable?



Order matters!

Consider following processes: p_1 p_2

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



Deadlines

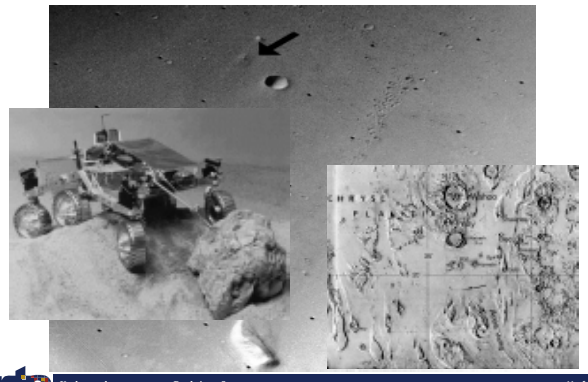
- Hard: Not meeting any deadline is a failure of the system
- Soft: It is desirable that deadlines are met, but OK if they are missed every now and then
- Firm: It is OK that they are missed now and again, but after the deadline the result is of no use

How often?

Typical application area

- Vehicle electronics
  - Power train and chassis
  - Infotainment/telematics
  - Body electronics
- A modern car configuration has over 40 ECUs, distributed over several buses
- Several applications share each ECU that shares the bus

Really good example



This lecture

- Course overview
- Introduction to Real-time systems
- CPU as a resource: Scheduling
  - Cyclic scheduling
  - Rate monotonic scheduling

Scheduling

... is about allocating resources, specially the CPU time, among all computational processes such that the timeliness requirements are met.

If all processes meet their deadlines then the process set is **schedulable**.

Scheduling

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

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 test:
  - sufficient & necessary at the same time

Which parameters?

Scheduling policy induces an order on executions using an algorithm and a set of parameters for the task set:

- Worst case execution time (WCET)
- Deadline
- Release time
- ...

Process parameters

- How to find the maximum computation time for each process?
- How to determine deadlines?
- When (how often) is a process released?

Release times

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

Cyclic scheduling

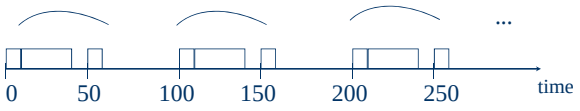
- A schedule is created based on statically known and fixed parameters
- Off-line decision on which task runs when
  - When executing: Run the processes in pre-determined order using a table look-up
- To run processes in the "right" frequency find
  - Minor cycle
  - Major cycle

Example (1)

Consider following processes: P_1 P_2

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

Note: repetition!



A cyclic executive

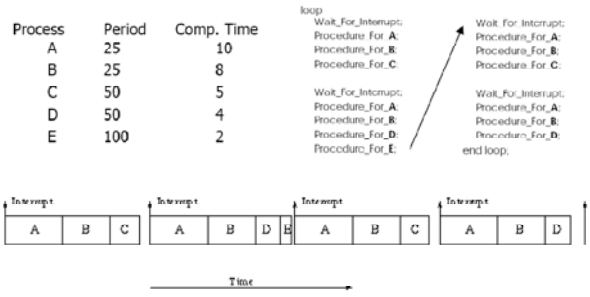
```
every_major_cycle do{
      read all in_signals;
      run_minor_cycle_1_processes;
      wait_for_interrupt;
      write all out_signals;
      ...
      read all in_signals;
      run_minor_cycle_n_processes;
      wait_for_interrupt;
      write all out_signals;
      }
```

End of minor cycle

End of minor cycle

No preemption!

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



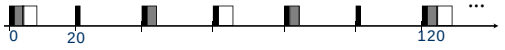
Finding Minor / Major Cycle

First try:
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
  - Each process P_i is run as if strictly periodic (i.e. should be completed once every T_i)
  - Place the processes in *minor cycle* and *major cycle* until repetition appears
  - Check: Will the schedule work with the natural periods and computation times?

When does it work?

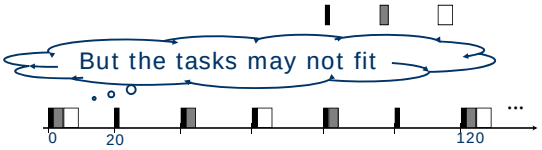
- All processes should be run *at least* as often as every (original) T_i
- All processes fit in their minor cycles
- Otherwise, change the parameters!
- Which parameters can we change?

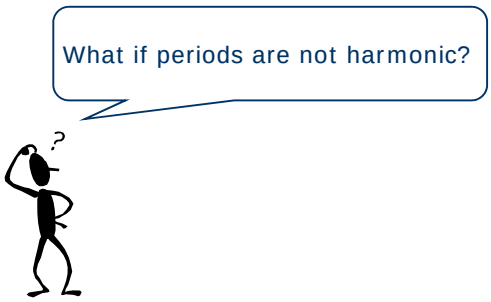
Harmonic processes

Easy to find minor/major cycle

Recall example 2:

process	A	B	C
period	20	40	60





Next try...

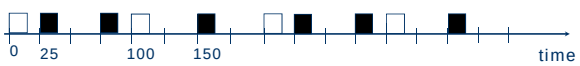
- In either case we need to
  - change the periods
  - recall that all processes should be run *at least* as often as every (original) T_i
- Place the processes in *new* minor cycle and major cycle until repetition appears

Example (3.1)

process	A	B
period	75	100

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

Drawbacks?
...



Jitter control

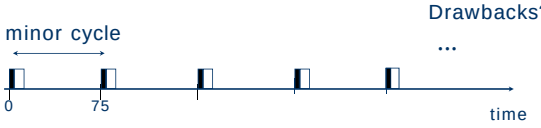
- Many applications need to minimise jitter in reading data from sensors or producing output to actuators

Example (3.2)

process	A	B
period	75	100

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

Drawbacks?
...

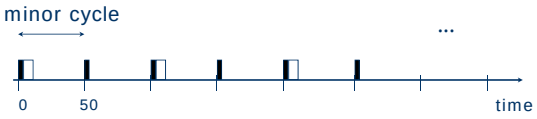


Example (3.3)

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

Creates new processes out of the old one!

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
- 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

Run-time behaviour

Try to work out:

- 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
- Dealing with 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 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 T_i
  - The shorter T_i the higher the priority

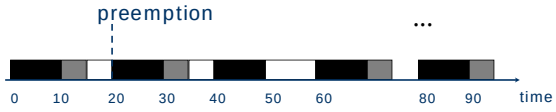
Example (4)

■ 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

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