What is a real-time system?

- Correctness depends not only on the logical result but also the time it was delivered
- Failure to respond is as bad as the wrong response
  - Predictability
- Usually a component in a larger engineering system
  - Embedded computer system
- Usually a critical system
  - Safety-critical
  - Dependable
  - Fault-tolerant

What is scheduling?

- Process as an abstraction
- Allocating resources, especially the CPU time, among all computational processes such that the timeliness requirements are met.
- The scheduler decides which process is to be executed next.

Non-real-time scheduling

- Primary goal
  - Maximize interactive performance
- Secondary goal
  - Ensure fairness
    - Prevent deadlock, starvation
- Typical metrics and goals:
  - Minimize response time
  - Maximize throughput
- Examples
  - FCFS (First-Come-First-Served)
  - RR (Round-Robin)

Real-time scheduling

- Primary goal:
  - Ensure predictability !!!
- Secondary goals
  - All the others
- Typical metrics and goals:
  - Guarantee ratio (hard real-time)
  - Maximize completion ratio / minimize miss ratio (hard / firm)
  - Minimize overall tardiness (soft real-time)
- Examples
  - Cyclic Scheduling
  - EDF (Earliest Deadline First)
  - LS (Least Slack)
  - RM (Rate-Monotonic)

Where do these parameters come from?

- Release time
- Computation time
- Deadline
- Event
- Reaction
Scheduling: problem space

- Single processor / multiprocessor / distributed system
- Periodic / sporadic / aperiodic
- Independent / interdependent
- Preemptive / non-preemptive
- Can handle transient overloads?
- Does it support fault tolerance?

Simple process model

- Fixed set of processes, uniprocessor system
- Processes are periodic, with known periods
- Processes are independent of each other
  - Critical instant assumption!
- System overheads, context switches etc. are ignored
- Processes have a deadline equal to their period
- Each process must complete before its next release
- Worst-case execution time is known

Simple process model

- Fixed set of processes, uniprocessor system
- Processes are periodic, with known periods
- Processes are independent of each other
  - Critical instant assumption!
- System overheads, context switches etc. are ignored
- Processes have a deadline equal to their period
- Each process must complete before its next release
- Worst-case execution time is known

Scheduling & schedulability analysis

- Offline vs. online scheduling
  - Offline (static)
    - Complete a priori knowledge of the task set and its constraints is available
    - Hard/safety-critical systems
  - Online (dynamic)
    - Partial taskset knowledge, runtime predictions
    - Firm/soft/best-effort systems

- Schedulability analysis
  - Tells if the scheduling algorithm can schedule a taskset
    - Can be used also with dynamic scheduling
    - A priori analysis very important for hard-rt systems

Schedulability Test

- Test to determine whether a feasible schedule exists
- 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

- Off-line scheduling
- Doesn’t use the process abstraction of the OS
- Manually created table of procedures to be called
  - Waits for a periodic interrupt for synchronisation
    - Minor cycle
  - Loops the execution of the procedures in the table
    - Major cycle

Cyclic Executive

- What happens if the periods are not harmonic?
General Procedure

- Calculate Minor Cycle
  - Greatest common divisor of task periods
  - Why?
    - Because an event can arrive only at the edge of a minor cycle
    - Events
      - Task ready
      - Deadline reached
    - Remember: all tasks start at the same time: critical instant
- Calculate Major cycle
  - Least common multiplier of task periods
  - Why:
    - Cause everything repeats itself afterwards

General procedure (cont’d)

- Populate all the minor cycles of a first major cycle with tasks
  - If a task cannot fit (its WCET) in a minor cycle it needs to be split
    - Over several minor cycles
    - Hopefully the programming technique allows it
  - Every task should run to completion once per period
    - Period might span several minor cycles
    - A new task instance cannot be scheduled before its natural release

Schedulability analysis

- Schedulability analysis?
  - See if a taskset is schedulable.
- Sum \( C_i/T_i \) <= 1 ?
  - NO
    - The taskset is not schedulable using cyclic scheduling.
  - YES
    - If we assume a simple task model, and that tasks can be split in arbitrarily small pieces.

Task dependencies & cyclic

- Dependences due to
  - Sharing resources
  - Computation precedence requirements
- Make sure the running order of the tasks is respected
- A taskset that is schedulable if tasks are independent might not be schedulable if dependencies are introduced.

Cyclic executives observations

- No actual processes exist at run-time
  - Each minor cycle is just a sequence of procedure calls
- The procedures share a common address space
  - Can thus pass data between themselves.
- This data does not need to be protected
  - Via a semaphore, for example
  - Because concurrent access is not possible.
- All process’ periods must be a multiple of the minor cycle
  - Why?

Disadvantages of cyclic

- Difficult to incorporate sporadic processes
- Incorporate processes with long periods
- Problematic for tasks with dependencies
  - Think about an OS without synchronisation primitives
- Time consuming to construct the cyclic executive
  - Or adding a new task to the taskset
- Handle processes with sizeable computation times.
  - How do you split them?
  - Can you split them?
**Disadvantages of cyclic**

- “Manual” scheduler construction
- Cannot deal with any runtime changes
  - What happens if we add a task to the set?
- Denies the advantages of concurrent programming
  - Which?

**Processes & scheduling**

- States of a process
  - Ready
  - Running
  - Waiting
  - Terminated

**Priority-based scheduling**

- Every task has an associated priority
- Run the process of the task with the highest priority
  - At every scheduling decision moment
- Examples
  - Rate Monotonic (RM)
    - Static priority assignment
  - Earliest Deadline First (EDF)
    - Dynamic priority assignment
  - And many others...

**Rate Monotonic**

- Each process is assigned a (unique) priority based on its period; the shorter the period, the higher the priority
- Assumes the “Simple task model”
- Fixed priority scheduling
- Preemptive

**Example 1**

- Assume we have the following task set
  - OBS: not scheduled yet...

**Example 1 (cont’d)**

- Scheduled with RM
Utilisation-based schedulability test for RM

- Sufficient, but not necessary:
  \[ \sum_{i=1}^{N} \left( \frac{C_i}{T_i} \right) \leq N \left( 2^{1/N} - 1 \right) \]

- Necessary, but not sufficient:
  \[ \sum_{i=1}^{N} \left( \frac{C_i}{T_i} \right) \leq 1 \]

Example 2

- Taskset: P1, P2, P3
- Period (T_i): 20, 50, 30
- WCET (C_i): 7, 10, 5

\[ U = \frac{7}{20} + \frac{10}{50} + \frac{5}{30} = 0.72 < 0.78 \]

- The schedulability of this task set is guaranteed!

Example 3

- Taskset:
- Utilisation test? Does not work!
- Gantt chart:

Exact schedulability test

- Theorem
  - If all tasks meet their first deadline, then they will meet all future ones.
    - Proof: paper by Liu and Layland, 1973

- How do we show this?
  - Assume worst case: critical instant
    - i.e. all tasks are released at moment 0
  - Draw Gantt-Charts for testing the schedulability.
  - Perform response time analysis for the first period for all the tasks.

Response time analysis

- Tasks suffer interference from higher priority tasks
- Response time: the time that passes since the task is released and until it finishes
  \[ R_i = C_i + I_i \]

- Iterative formula for calculating response time
  \[ W_i^{(t+1)} = C_i + \sum_{j=1}^{t} \frac{W_j^{(t)}}{T_j} C_j \]

Optimality of scheduling algorithms

- “An optimal scheduling algorithm is guaranteed to always find a feasible schedule, given that a feasible schedule does exist.”
Optimality of RM

- Rate Monotonic is optimal among fixed priority schedulers
  - If we assume the “Simple Process Model” for the tasks

What to do if not schedulable

- Change the task set utilisation
  - by reducing $C_i$
    - code optimisation
    - faster processor
  - Increase $T_i$ for some process
    - If your program and environment allows it

RM characteristics

- Advantage:
  - Easy to implement
  - Easy to analyse

- Drawback:
  - May not give a feasible schedule even if processor is idle at some points.