Metode şi Algoritmi de Planificare (MAP)

2009-2010

Curs 6

Planificare in Sisteme de Timp Real

Real-Time System

- Definition:
  - Any system in which the time at which output is produced is significant.
  - This is usually because the input corresponds to some movement in the physical world, and the output has to relate to the same movement.
  - The lag from input time to output time needs to be sufficiently small for acceptable timeliness
- Correct function at correct time
- Usually embedded
- Often distributed
- Deadlines
  - Hard real-time systems
  - Soft real-time systems

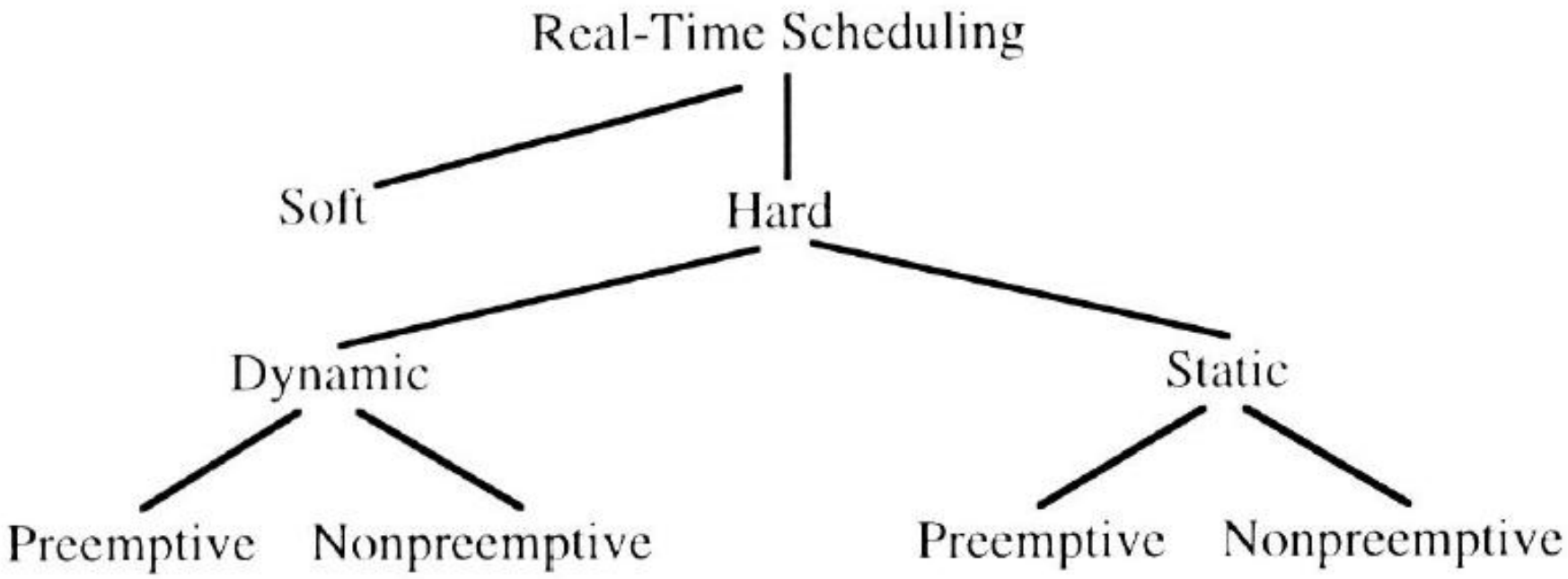
Distinctive Embedded System Attributes

- Reactive: computations occur in response to external events
  - Periodic events (e.g. rotating machinery and control loops)
  - Aperiodic events (e.g. button click)
- Real Time: correctness is partially a function of time
  - Hard real time
    - Absolute deadline, beyond which answer is useless
    - May include minimum time as well as maximum time
  - Soft real time
    - Approximate deadline
    - Utility of answer degrades with time difference from deadline

Real-Time Review

- **Real time is *not* just “real fast”**
  - Real time means that correctness of result depends on both functional correctness and time that the result is delivered
- **Soft real time**
  - Utility degrades with distance from deadline
- **Hard real time**
  - System fails if deadline window is missed
- **Firm real time**
  - Result has no utility outside deadline window, but system can withstand a few missed results

Type of Real-Time Scheduling



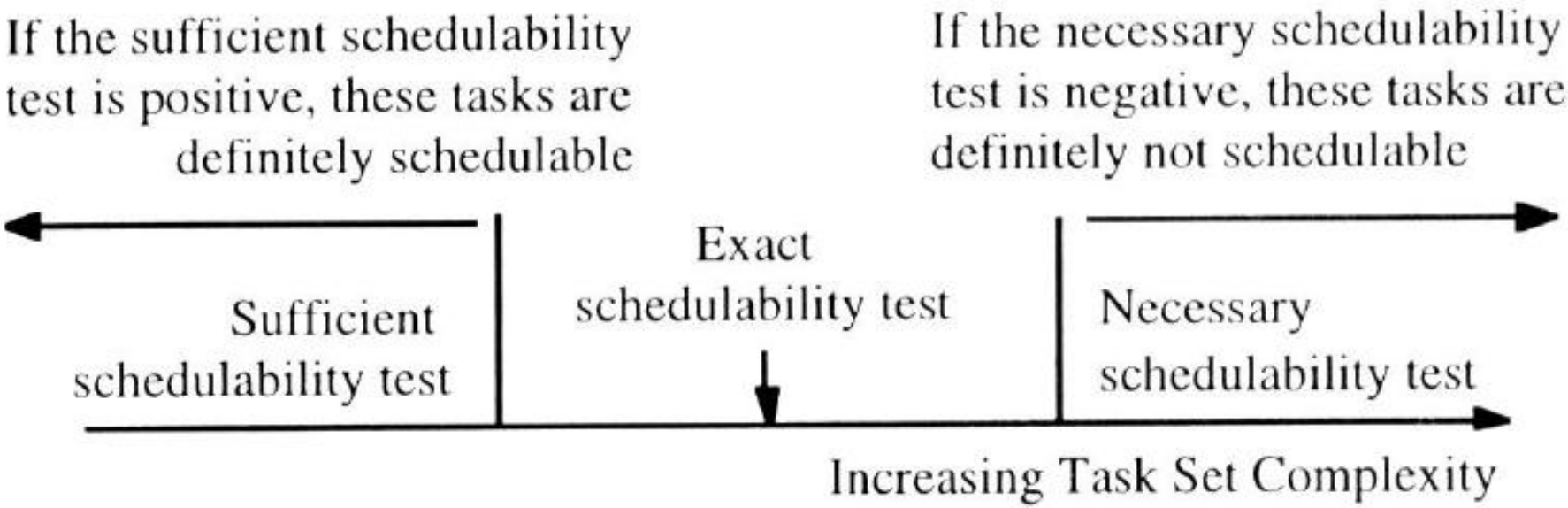
Taxonomy of Real-Time Scheduling

Type of Real-Time Scheduling

- **Dynamic vs. Static**
  - Dynamic schedule computed at run-time based on tasks really executing
  - Static schedule done at compile time for all *possible* tasks
- **Preemptive permits one task to preempt another one of lower priority**

- NP-hard if there are any resources dependencies !
  - Prove it definitely cannot be scheduled
  - Find a schedule if it is easy to do
  - Stuck in the middle somewhere

Schedulability ☺



Necessary and sufficient schedulability test

Scheduling Parameters for RTS

- **Assume N CPUs available for execution of a single task set**
- **Set of tasks {*Ti*}**
  - Periods *ti*
  - Deadline *di* (completion deadline after task is queued)
  - Execution time *pi* (amount of CPU time to complete)
- **Handy values:**
  - Laxity *li* = *di* – *pi* (amount of slack time before *Ti must* begin execution)
  - Utilization factor *ui* = *pi*/*ti* (portion of CPU used)

Static Schedule

- **Assume non-preemptive system with 5 restrictions:**
  1. Tasks {*Ti*} are periodic, with hard deadlines and no jitter
  2. Tasks are completely independent
  3. Deadline = period (*ti* = *di*)
  4. Computation time *pi* is known and constant
  5. Context switching is free (zero cost) INCLUDING network messages to send context to another CPU(!)

Static Schedule

- **Consider least common multiple of periods ti**
  - This considers all possible cases of period phase differences
  - Worst case is time that is product of all periods; usually not that bad
  - If you can figure out (somehow) how to schedule this, you win
- **Performance**
  - Optimal if all tasks always run; can get up to 100% utilization
  - If it runs once, it will always work

EDF: Earliest Deadline First

- **Assume a *preemptive* system with *dynamic* priorities, and (same 5 restrictions)**
- **Scheduling policy:**
  - Always execute the task with the nearest deadline
- **Performance**
  - Optimal for uniprocessor (supports up to 100% of CPU usage in all situations)
  - If you're overloaded, ensures that a lot of tasks don't complete
    - Gives everyone a chance to fail at the expense of the later tasks

Least Laxity

- **Assume a *preemptive* system with *dynamic* priorities, and (same 5 restrictions)**
- **Scheduling policy:**
  - Always execute the task with the smallest laxity
- **Performance:**
  - **Optimal for uniprocessor (supports up to 100% of CPU usage in all situations)**
    - Similar in properties to EDF
  - **A little more general than EDF for multiprocessors**
    - Takes into account that slack time is more meaningful than deadline for tasks of mixed computing sizes
  - Probably more graceful degradations
    - Laxity measure permits dumping tasks that are hopeless causes

EDF/Least Laxity Tradeoffs

- **Pro:**
  - If it works, it can get 100% efficiency (on a uniprocessor)
- **Con:**
  - **It is not always feasible to prove that it will work in all cases**
    - And having it work for a while doesn't mean it will always work
  - **Requires dynamic prioritization**
  - **The laxity time hack for global priority has limits**
    - May take too many bits to achieve fine-grain temporal ordering
    - May take too many bits to achieve a long enough time horizon

Rate Monotonic

- **Assume a *preemptive* system with *static* priorities, and (same 5 restrictions) plus**

$$\mu = \sum \mu_i = \sum \frac{p_i}{t_i} \leq N(2^{\frac{1}{N}} - 1) \qquad ; \mu \approx 0.7 \text{ for large N}$$

- **Scheduling policy:**
  - Highest static priority goes to shortest period; always execute highest priority
- **Performance:**
  - Provides a *guarantee* for schedulability with CPU load of ~70%
    - – Even with arbitrarily selected task periods
    - – Can do better if you know about periods & offsets
  - If all periods are multiple of shortest period, works for CPU load of 100%

Data Dependencies

- **If two tasks access the same data** (violates Assumption #2)
  - Use a semaphore to ensure non-simultaneous access
  - Scheduling can get really nasty…
- **Priority inversion**
  - **Low-priority task holds resource that prevents execution by high-priority task**
    - Task 3 acquires lock
    - Task 1 needs resource, but is blocked by Task 3, so Task 3 is allowed to execute
    - Task 2 preempts Task 3, because it is higher priority
    - BUT, now we have Task 2 delaying Task 3, which delays Task 1
- **Priority ceiling protocols**

Real Time Operating Systems

- **Unix “feel”**
  - QNX
  - LinxOS
- **“Smaller” RTOSs**
  - OS9, Microware
  - VxWorks pSOS
  - RTX, VenturCom
- **A few research/freeware RTOS systems:**
  - RTEMS, Redstone Military Arsenal
  - RT-Mach, CMU
  - … others
- **Windows:** WinCE; embedded NT, Windows + RT add-on products

Exam's quizzes

- **1.** Descrieţi pe scurt noţiunea de planificare în timp real.
- **2.** Ce se înţelege prin noţiunea de "Schedulability"?
- **3.** Descrieţi pe scurt 3 algoritmi de planificare pentru sistemele de timp real.