Hasso Plattner Institut

IT Systems Engineering | Universität Potsdam

Real-time Scheduling

Verteilte Echtzeitsystem 2010

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

- Different kinds of real-time:
  - **Soft real-time**: Utility degrades with distance from deadline
  - **Firm real-time**: Result has no utility outside deadline window, but system can withstand a few missed results
  - **Hard real-time**: System fails if deadline window is missed

Real-time Systems

- Real-time Systems commonly react on external events
  - **Periodic events**: tasks which are activated at fixed rates (e.g. rotating machinery and control loops)
  - **Aperiodic events**: tasks which are activated at unforeseen times (e.g. button click)
  - **Sporadic events**: tasks which are activated irregularly but a bound rate is known

The Problem

"For a given set of jobs, the general scheduling problem asks for an order according to which the jobs are to be executed such that various constraints are satisfied."

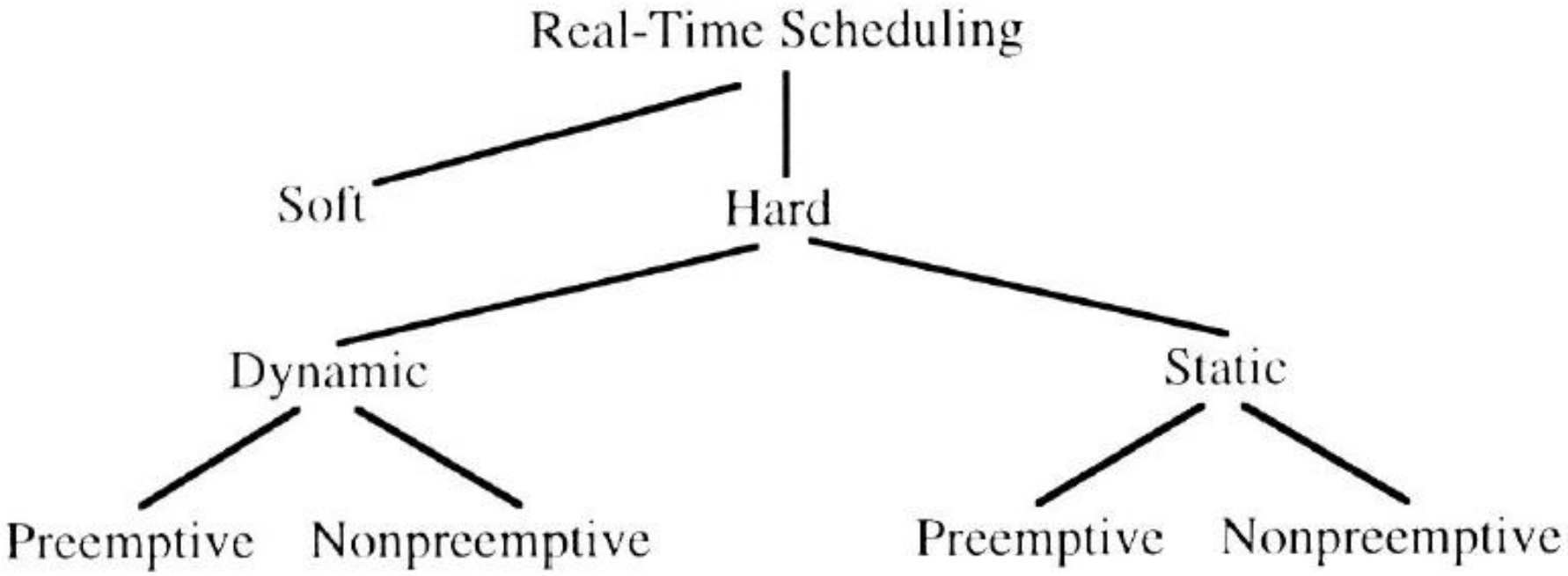
Constraints for real-time scheduling:

- Tasks shell finish before deadline
- Tasks may depend on other tasks finishing before
- Tasks may take precedence over other tasks

→ **Different goals** for real-time scheduling:

- Efficiency and fairness of secondary importance
- Predictability and resilience come first

Classification of Scheduling Algorithms



Uni-Processor Scheduling

- Common assumptions:
  1. No task has any nonpreemptable section and the cost of preemption is negligible.
  2. Only processing requirements are significant; memory, I/O, and other resource requirements are negligible.
  3. All tasks are independent; there are no precedence constraints.

EDF: Earliest Deadline First

- Dynamic-priority scheme
- Algorithm:
  - 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

- Dynamic-priority scheme
- Algorithm:
  - Always execute the task with the smallest laxity (=amount of time before a task *must* begin)

- Performance:
  - Optimal for uniprocessor (supports up to 100% of CPU usage in all situations)
  - 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

Traditional rate-monotonic scheduling

- Static-priority scheme
- Additional assumptions:
  - All tasks are periodic
  - For all T_i: $D_i = P_i$
- Scheduling policy:
  - The tasks with the shortest period gets the highest priority

- Performance:
  - Optimal static-priority uniprocessor algorithm

Traditional rate-monotonic scheduling

- Schedulability criteria:
  - Sufficient condition: total processor utilization is not greater than $n(2^{(1/n)} - 1)$
    - Obvious schedules, that work and have a higher uitilization
  - Neccessary condition:
    - $W_i(t) = \sum_{j=1}^{i} e_j \left\lceil \frac{t}{P_j} \right\rceil$: Total work of Tasks $T_1 \ldots T_i$ in [0,t]
    - $W_i(t) < 1$
- Overload case:
  - Task with longest period misses its deadline
  - Can be manipulated by artificially partitioning a task

Other Principles - Multiple task versions

- System has primary and alternative version of tasks
- Vary in execution time and quality of output:
  - Primary: full-fledged task; top quality output
  - Alternative: bare-bone; lower-quality (acceptable) output; take less much execution time
- Scheduler may pick alternative tasks during overload

Multi-processor Scheduling

- Most algorithms based on Uni-processor ones and add distributing the tasks over the processors

  - **Utilization balancing algorithm**:

```
For each task T_i, do
  Allocate one copy of T_i to each of the r_i least utilized processors
  Update the processor allocation to account for the allocation of T_i
End do
```

  - **Next-Fit algorithm for RMS**

```
For each task T_i, do
  Assign T_i to class C_i
  If T_i is not RM-schedulable on all processors assigned to C_i
    Assign additional processor to C_i
  End if
  Allocate T_i on a C_i processor
End do
```

Multi-processor Scheduling

- **Bin-packing algorithm for EDF**
  - Total utilizations <= a given threshold
  - Threshold: the uniprocessor scheduling algorithm is able to schedule the tasks assigned to each processor
  - Based on classical bin-packing problems

- **Myopic offline scheduling algorithm**
  - Can deal with nonpreemptible tasks
  - Defines a logical schedule tree: height of tree is number of included tasks
  - Depth-search based on:
    - Task selection based on heuristic function H: execution time, deadline, start time, laxity, etc.
    - Development decision: strong feasibility

Multi-processor Scheduling

- **Focused addressing and bidding algorithm**
  - Tasks arrive at the individual processors
  - One processor finds itself unable to meet the deadline or other constraints of new task: tries to offload
    - Send to most probable processor
    - Maybe start bidding process
  - Underloaded processors announce themself and answer to bidding

- **Buddy Strategy**
  - Similar to FAB
  - Targeted on multi-hope networks
  - State-based: underloaded, fully loaded or overloaded
  - If in overloaded state, offload workload onto buddies

Scheduling Problems

- Critical Sections:
  - Source of priority inversion problem
  - **Priority Inheritance**: a lower priority owner of a resource needed by a higher priority task is shortly boosted to higher priority
    - Does not prevent deadlocks in a program with circular lock dependencies
    - A chain of blocking may occur: blocking duration can be substantial, though bounded
  - **Priority Ceiling Protocol**: based on priority inheritance, but uses a *Priority Ceiling* for each critical section
    - Prevents deadlocks and unbound priority inversion

Scheduling Problems

- Mode changes:
  - Problems: when to delete or add tasks in a way to hold-up deadlines
  - Delete tasks finally at end of period of the last run
  - Tasks can be added anytime as long the final task set is still RM-schedulable
  - Priority ceiling protocol:
    - Sempahore ceilings have to be updated
    - If the semaphore is locked, change is execute when semaphore is freed

Scheduling Problems

- Assumptions from the beginning:
  1. No task has any nonpreemptable section and the cost of preemption is negligible.
     - Cost of preemption can be integrated into scheduling formulas(only complicates understanding)
  2. All tasks are independent; there are no precedence constraints.
     - Can be incorporated into worst-case runtime
     - Can be solved by partitioning/fusing the task

Related Lecture

- C.M. Krishna and Kang G. Shin, Real-Time Systems, McGraw-Hill, ISBN: 0-07-057043-4, 1997
- http://www.ifi.uzh.ch/~riedl/lectures/maendli.pdf
- http://www.isiconference.org/2003/resourses/presentation/REAL_TIME_SCHEDULING.PPT