Real Time Scheduling

Slides reference taken from UCLA

CS | Computer Science (ucla.edu)

What are Real-Time Systems?

- whose correctness depends on timing as well as functionality.
- real-time scheduling is more critical and difficult than traditional time-sharing.
- But real-time systems may have a few characteristics that make scheduling easier.

Look back: traditional scheduling algorithms

- metrics were:
  - *turn-around time* (or throughput),
  - *fairness*, and
  - mean *response time*.

Moving forward: real-time scheduling

- Metrics are:
  - *Timeliness*: how closely does it meet its timing requirements (e.g. ms/day of accumulated tardiness)
  - *Predictability*: how much deviation is there in delivered timeliness

Introducing a few new concepts:

- *Feasibility*: whether or not it is possible to meet the requirements for a particular task set
- *hard real-time*: there are strong requirements that specified tasks be run a specified intervals (or within a specified response time). Failure to meet this requirement (perhaps by as little as a fraction of a micro-second) may result in system failure.
- *soft real-time*: we may want to provide very good (e.g. microseconds) response time, the only consequences of missing a deadline are degraded performance or recoverable failures.

Real-Time Scheduling Algorithms

- When the tasks and their execution times are all known, there might not even be a scheduler.
- One task simply call the next.
- makes a great deal of sense in a system where the tasks form a producer/consumer pipeline (e.g. protocol decoding, image decompression).

Contd...

- with a larger (but still fixed) number of tasks that do not function in a strictly pipeline fashion, it may be possible to do *static* scheduling.
- Based on the list of tasks to be run, and the expected completion time for each, we can define a fixed schedule that will ensure timely execution of all tasks.

Complexity

- work-load changes from moment to moment, based on external events.
- require *dynamic* scheduling:
  - two keys:
    - how they choose the next (ready) task to run
      - shortest job first
      - static priority ... highest priority ready task
      - soonest start-time deadline first (ASAP)
      - soonest completion-time deadline first (slack time)
    - how they handle overload (infeasible requirements)
      - best effort
      - periodicity adjustments ... run lower priority tasks less often.
      - work shedding ... stop running lower priority tasks entirely.

Advantages

- enables intelligent scheduling
- *Starvation* (of low priority tasks) may be acceptable
- work-load may be relatively fixed