Mixing Hard and Soft Tasks

Problem to solve

- Hard-deadline tasks may be
  - Periodic or
  - Sporadic (with a known minimum arrival time)
  - (or non periodic, difficult to solve)
- Soft-deadline tasks (and/or non RT) may be
  - Various types (periodic or non periodic etc)
- We want to shedule the mixed task set so that
  - All hard tasks meet their deadlines
  - All soft tasks get average response times as low as possible

Simple solution: Background scheduling

- Schedule all hard tasks as usul (e.g. RMS) and Run non RT tasks whenever CPU is free (i.e put non RT tasks in the background)
- This is fine if we don't care average response time for soft tasks

Potential improvements

- Sporadic tasks with small Tmin and low rate (low CPU utilization): RM analysis will be too pessimistic
- Periodic tasks with low CPU utilization: this fact may be used to improve response times for soft-tasks
- Hard deadlines are not necessarily met as early as possible

Eample

- Hard task: (C,D,T)
  - Task H =(3,9,10)
  - Task L =(4,14,15)
- One soft task: (C,D)
  - Task S=(3,5)
- Assume that they all arrive at time 0
  - If H and L are executed first as they have hard deadlines, S will miss its deadline 5
  - If S is executed first, and then H, L, all deadlines will be met

Combined Scheduling

- Creating a *periodic server* Ts=(Cs, Ps) for processing aperiodic workload. Create one or more server tasks.
- Aperiodic tasks are scheduled in the periodic server's time slots. This policy could be based on deadline, arrival time, or computation time.
- Algorithms – all algorithms behave the same manner when there are enough aperiodic tasks to execute
  - Polling Server (bandwidth non-preserving)
  - Deferrable Server (bandwidth preserving)
  - Priority Exchange Server (bandwidth preserving)

Polling Server (PS)

- Idea:
  - Consider that all hard tasks are periodic
  - Create a periodic task (a server) with period Ts and capacity Cs (the allowed computing time in each period)
  - Schedule the server as a periodic task (Cs, Ts)
- Run time behaviour:
  - Once the server is active, it serves all pending (buffered) aperiodic requests within its capacity Cs according to other algorithms e.g FCFS, SJF etc
  - If no aperiodic requests, the capacity is lost: if a request arrives after the server has been suspended, it must wait until the next polling period

Deferrable server (PS preserving capacity) [Lehoczky and Sha et al, 87,95]

- It is similar to Polling server
- The only difference is that the capacity of DS will be preserved if no pending requests upon the activation of the server. The capacity is maintained until the end of the server
  - within the period, an aperiodic request will be served; thus improving average response time

Priority Exchange (interesting!)

- Similar to PS and DS, PE has a periodic server (usually with high priority) for serving aperiodic tasks. The difference is in the way how the capacity of the server is preserved
- Run Time Behaviour:
  - If the PE server is currently the task with highest priority but there is no aperiodic request pending, then
    - the periodic task with next highest priority runs and
    - the server is assigned with the periodic task's lower priority
  - Thus the capacity of the server is not lost but preserved with a lower priority (the exchange continues until new aperiodic requests arrive)

So far, we should know

- How to schedule aperiodic task sets
  - Optimal scheduling algorithms
  - Precedence constraints
- How to schedule periodic task sets
  - Schedulability tests
  - Calculation of response time
- How to schedule mixed task sets
  - Improve response times for soft tasks
- How to avoid un-bounded priority inversion
  - Resource access protocols
  - Calculation of blocking time