Combined Scheduling of
Periodic and Non-Periodic Tasks
(hard and soft tasks)



## Problem to solve

- Hard-deadline tasks may be
  - Periodic or
  - Sporadic (with a known minimum arrival time)
  - Aperiodic/Event-driven – e.g. ABS-break
- Soft-deadline tasks (and/or non RT) may be
  - Various types (mostly aperiodic/event-driven)
- 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
- We also want to estimate the worst-case response times for non-periodic tasks if possible



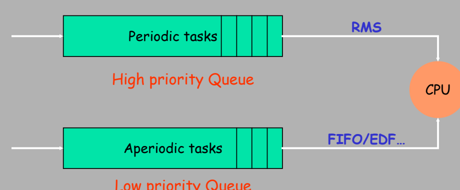
## Combined Scheduling

- 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)
  - Sporadic Server (bandwidth preserving)



## Background Scheduling Algorithm

- No server is created.
- Aperiodic tasks are executed when there is no periodic task to execute.
- Simple, but no guarantee on aperiodic schedulability

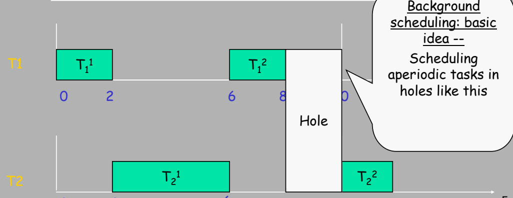





## Normal RMS schedule: Notice the holes

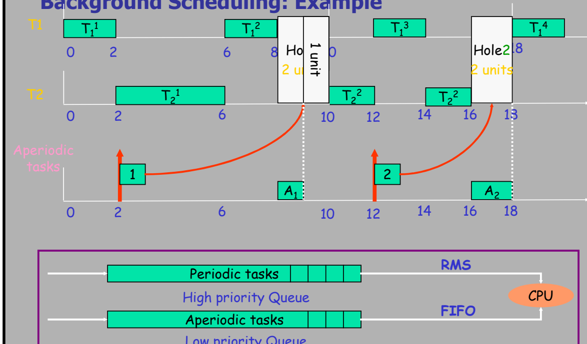
Task set: $T_i = (c_i, p_i)$
T1 = (2,6) and T2 = (4,10)

Schedulability check:
2/6 + 4/10 = 0.33 + 0.40 = 0.73 ≤ UB(2) = 0. 82






## Background Scheduling: Example



## Background scheduling works well, but

- How to estimate the worst case response times for aperiodic tasks if they have hard deadlines?
  - We need Periodic servers
- How to improve the average response times for aperiodic tasks in case they have soft deadlines?
  - Sporadic tasks with small $T_{min}$ 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



## Dual Priority Scheduling

- Idea: there is no benifit in early completion of hard tasks. Use three ready queues:
  - HIGH, MIDDLE and LOW Corresponding priorities
- Run Time Behaviour:

  Hard tasks ➔LOW ➔ HIGH ➔Running

  Soft tasks ➔ MIDDLE
  - A hard task will be placed in the LOW queue initially and after a delay say X (priority promotion delay), it is promoted and put in the HIGH queue
  - All soft tasks are put in the MIDDLE queue
  - Run tasks in the queues according to the priorities: H,M,L
- how to calculate the promotion delay X ?



## Calculate the promotion delay

- Remember

$$Ri= Bi + Ci + \sum_{j \in HP(i)} \lceil Ri/Tj \rceil * Cj$$

- Thus

$$Xi = Di - Ri$$

(this may not work, why? How to calculate $X_{i+1}$?)



## Polling Server (PS)

- Idea:
  - Consider that all hard tasks are periodic
  - Create a periodic task (a periodic 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
- Assume one-server for one aperiodic task, how to calculate the Worst-case response time?



## 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
- Assume one-server for one aperiodic task, how to calculate the Worst-case 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)
- Assume one-server for one aperiodic task, how to calculate the Worst-case response time?



## Other solutions

- Slack stealing server (similar to Dual priority sch.)
  - Steal the slack $Si(t)=Di-t-Ci(t)$ for aperiodic tasks
    - $Si(t)$ is the slack time for task $i$ at time $t$
    - $Ci(t)$ is the remaining computing time for task $i$ at time $t$
- Sproadic server (similar to PS)
  - The server replenishes its capacity only after it has been consumed by aperiodic task execution ('consumed' implies more arrivals of sporadic tasks)



## 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 worst-case response times
- How to schedule mixed task sets
  - Improve response times for soft tasks
  - Calculation of worst-case response times