Dynamic Priority Scheduling

- Static-priority:
  - Rate Monotonic (RM): “The shorter the period, the higher the priority.” [Liu+Layland ‘73]
  - Deadline Monotonic (DM): “The shorter the relative deadline, the higher the priority.” [Leung+Whitehead ‘82]
- For arbitrary relative deadlines, DM outperforms RM
- Dynamic-priority:
  - EDF: Earliest Deadline First
  - LST: Least Slack Time First
  - FIFO/LIFO
  - others

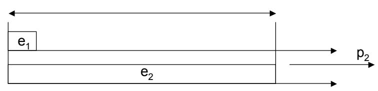
Priority-Driven Scheduling

- FIFO/LIFO do not take into account urgency of jobs
- Static-priority assignments based on functional criticality are typically non-optimal
- We confine our attention to algorithms that assign priorities based on temporal parameters
- Definition [**Schedulable Utilization**]: Every set of periodic tasks with total utilization less or equal than the schedulable utilization of an algorithm can be feasibly scheduled by that algorithm
- The higher the schedulable utilization, the better the algorithm
- Schedulable utilization is always less or equal to 1.0!

Schedulable Utilization of FIFO

- Theorem: $U_{FIFO} = 0$
- Proof: Given any utilization level $\varepsilon>0$, we can find a task set, with utilization ε, which may not be feasibly scheduled according to FIFO
- Example task set:

T_1: $e_1 = \varepsilon/2 * p_1$
T_2: $p_2 = 2/\varepsilon * p_1$
$e_2 = p_1$
$\Rightarrow U = \varepsilon$

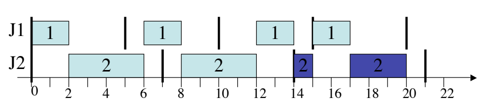


Earliest Deadline First (EDF)

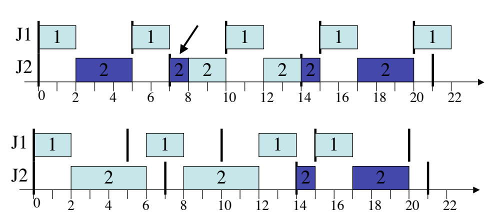
- Online
- Preemptive
- Dynamic priorities
- “Always run the process that is closest to its deadline”
- Requirements:
  - events that lead to release of P_i appear with minimum interarrival interval T_i
  - P_i has a max computation time e_i
  - the process must be finished before its deadline $D_i \leq T_i$
  - processes are independent (do not share resources)
  - the process with shortest absolute deadline (d_i) will run first

EDF

- Earliest deadline first with two tasks
- $C_1=2$, $T_1=D_1=5$
- $C_2=4$, $T_2=D_2=7$
- Earliest Deadline First
  - Optimal
  - Sufficient condition $U \leq 1$
  - Dynamic priority assignment
  - Runs the task with the closest deadline

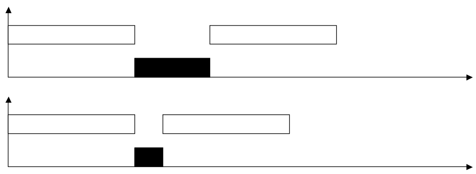


RMS versus EDF



EDF versus RMS

Process	P1	P2
WCET	5	10
Deadline (Di=Ti)	20	12
Arrival times (ri)	0,20,…	0,12,…



Theorem

- A set of periodic tasks $P_1,\ldots,P_n$ for which $D_i = T_i$ is schedulable with EDF iff $U \le 1$
- EDF versus RMS
  - EDF gives higher processor utilization
  - EDF has simpler exact analysis
  - RMS can be implemented to run faster at run-time (ignoring time for context switching)

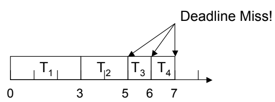
Sufficient Acceptance Test for EDF

- If the deadline ≥ period, then test is both necessary and sufficient
- If the deadline < period, then the test is only a sufficient condition

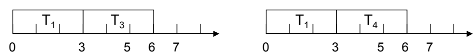
$$Density = \Delta = \sum_{k=1}^{n} \frac{e_k}{\min(D_k, p_k)} \le 1$$

Unpredictability of EDF

- Domino effect during overload conditions
- Example: $T_1(4,3)$, $T_2(5,3)$, $T_3(6,3)$, $T_4(7,3)$



Better schedules:



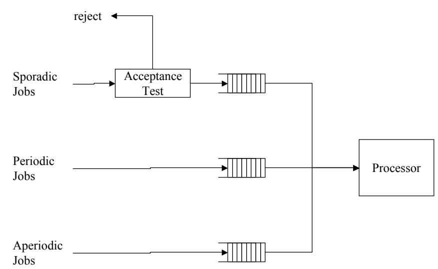
Least Slack Time First (LST)

- Slack of a job at time t: d-t-x
- Scheduler gives jobs with smaller slack higher priority
- Difference to EDF?

Scheduling Aperiodic and Sporadic Jobs

- Given: *n* periodic tasks $T_1, \ldots, T_i = (p_i, e_i), \ldots, T_n$
  priority-driven scheduling algorithm
- We want to determine when to execute aperiodic and sporadic jobs, *i.e.,*
  - sporadic job: acceptance test
    scheduling of accepted job
  - aperiodic job: schedule job to complete ASAP.

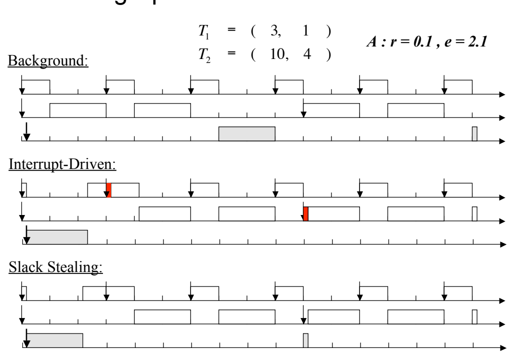
Priority Queues



Executing Aperiodic Jobs

- Background:
  - Aperiodic job queue has always lowest priority among all queues.
  - Periodic tasks and accepted jobs always meet deadlines.
  - Simple to implement.
  - Execution of aperiodic jobs may be unduly delayed.
- Interrupt-Driven:
  - Response time as short as possible.
  - Periodic tasks may miss some deadlines.
- Slack Stealing:
  - Postpone execution of periodic tasks only when it is safe to do so:
    - Well-suited for clock-driven environments.
    - What about priority-driven environments? (quite complicated)

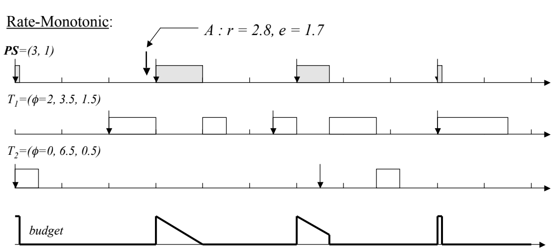
Executing Aperiodic Jobs



Polled Execution, Bandwidth Preserving Servers

- Polling server *(p_s, e_s)*: scheduled as periodic task.
  - p_s: Poller ready for execution every p_s time units.
  - e_s: Upper bound on execution time.
- Terminology:
  - (Execution) budget: e_s
  - Replenishment: set budget to e_s at beginning of period.
  - Poller consumes budget at rate 1 while executing aperiodic jobs.
  - Poller exhausts budget whenever poller finds aperiodic queue empty.
  - Whenever the budget is exhausted, the scheduler removes the poller from periodic queue until replenished.
- Bandwidth-preserving server algorithms:
  - Improve upon polling approach
  - Use periodic servers
  - Are defined by consumption and replenishment rules.

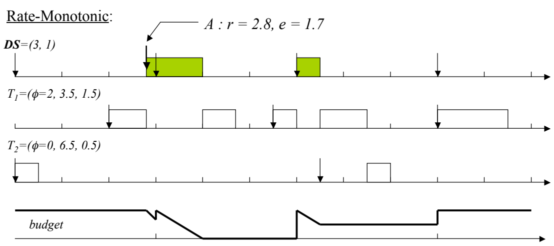
Example: Polling Server



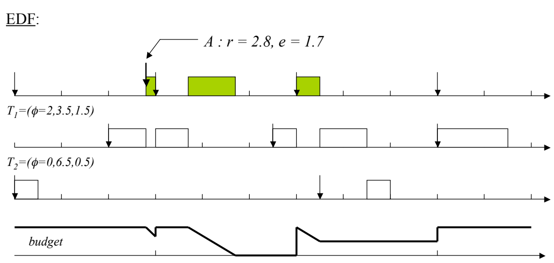
Deferrable Servers

- Rules:
  - Consumption: Execution budget consumed only when server executes.
  - Replenishment: Execution budget of server is set to e_s at each multiple of p_s.
- Preserves budget when no aperiodic job is ready.
- Any budget held prior to replenishment is lost (no accumulation).

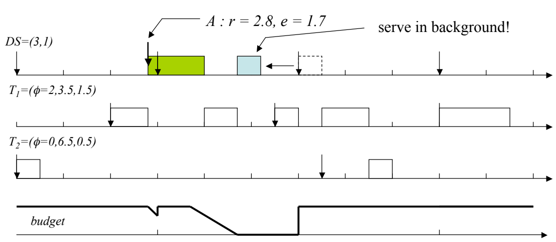
Deferrable Server with RMS



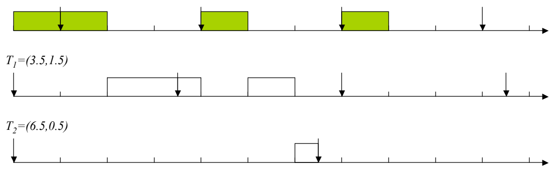
Deferrable Server with EDF



Deferrable Server with Background Server



Why Not Increase The Budget?



Total Bandwidth Server

- Consumption rule:
  - A server consumes its budget only when it executes.
- Replenishment rules:
  - **R1** Initially, set e_s := 0 and *d* :=0.
  - **R2** When an aperiodic job with execution time *e* arrives at time *t* to an empty aperiodic job queue, set $d := max(d,t) + e_s/u_s$, and e_s := *e*.
  - **R3** Upon completion of the current aperiodic job, remove job from queue.
    - *(a)* if the server is backlogged, set $d := d + e/u_s$ and $e_s := e$;
    - *(b)* if the server is idle, do nothing.

TBS: Eliminated Unused Capacity

T_1 = (3, 0.5) T_2 = (4, 1.0) T_3 = (19, 4.5)