Priority-driven Scheduling of Periodic Tasks



- Recall that a periodic task T_i is defined by the 4-tuple (φ_i, p_i, e_i, D_i) and that the utilization of T_i, $u_i = e_i / p_i$
- For a system of periodic tasks $\mathcal{T} = \{T_i, i = 1..n\}$, the total utilization, $U(\mathcal{T}) = u_1 + u_2 + \ldots + u_n$
- A scheduling algorithm can feasibly schedule any system $\mathcal{T}$ of periodic tasks on a processor if $U(\mathcal{T})$ is equal to or less than the ***schedulable utilization*** of the algorithm
- Schedulable utilization is an attribute of a scheduling algorithm; we will denote it as $\mathcal{U}_{ALG}$
- If $\mathcal{U}_{ALG} = 1$, the algorithm is optimal
- Why is the knowledge of $\mathcal{U}_{ALG}$ important?
  - Validation of your system then depends upon simply showing that for your system $\mathcal{T}$, $U(\mathcal{T}) \leq \mathcal{U}_{ALG}$

Priority-driven Scheduling of Periodic Tasks



- The schedulable utilization of the EDF algorithm
  - Theorem: A system $\mathcal{T} = \{T_i, _I = 1..n\}$ of independent, preemptable periodic tasks with $D_i = p_i$ can be feasibly scheduled on one processor if and only if $U(\mathcal{T}) \leq 1$. (proof covered next Tuesday)
  - Corollaries:
    - A system $\mathcal{T}$ of independent, preemptable periodic tasks with $D_i > p_i$ can be feasibly scheduled on a processor as long as $U(\mathcal{T}) \leq 1$
    - $\mathcal{U}_{EDF}(n)$ for n independent, preemptable periodic tasks with $D_i \geq p_i$ is 1.
    - $\mathcal{U}_{LST}(n)$ for n independent, preemptable periodic tasks with $D_i \geq p_i$ is 1.
  - Note that all of these results are independent of φ_i

Priority-driven Scheduling of Periodic Tasks



- What happens if $D_i < p_i$ for some i?
  - Define $\tau_i = \min(D_i, p_i)$
  - The density for T_i, $\delta_i = e_i / \tau_i$
  - The density of the system, $\Delta(\mathcal{T}) = \delta_1 + \delta_2 + \ldots + \delta_n$
  - Theorem: A system $\mathcal{T}$ of independent, preemptable periodic tasks can be feasibly scheduled on one processor if $\Delta(\mathcal{T}) \leq 1$.
    - Note that this is not a necessary condition, it is simply sufficient – i.e. a system may be feasible when $\Delta(\mathcal{T}) > 1$.

Priority-driven Scheduling of Periodic Tasks



- Schedulability testing
  - A test for the purpose of validating that the given application system meets all its hard deadlines when scheduled according to a particular scheduling algorithm is a ***schedulability test***.
  - If a schedulability test is efficient, then it can be used as an on-line acceptance test.
- Schedulability test for EDF
  - $\Delta(\mathcal{T}) \leq 1$
  - What do we conclude if this test is not satisfied?
    - If $D_i \geq p_i$ for all i, then the system is not schedulable
    - If $D_i < p_i$ for some i, the system may not be schedulable
  - This test is robust – i.e. the test holds true if some jobs execute for less than their maximum execution times; it also holds true if the interrelease times of jobs in a task are longer than the period (minimum interrelease time)

Priority-driven Scheduling of Periodic Tasks



- How can you use this in practice?
  - Assume that you have decided to use EDF to schedule multiple periodic tasks
    - You know the execution times for jobs in each periodic task
    - You can choose the periods for the tasks such that the schedulability test is met
  - Example – digital robot controller
    - Control-law computation takes 8 ms
    - Desired sampling rate is 100 Hz => period is 10 ms => utilization is .8
    - Suppose that a built-in self test task is to be included, and that the computation of the test takes 50 ms; schedulability test tells us that as long as the period for this task is 250 ms or more, the total utilization remains ≤ 1; if a period of 1 sec is chosen, utilization is 0.05
    - If we now wish to add another task, as long as its utilization is ≤ 0.15, the system is still feasible

Priority-driven Scheduling of Periodic Tasks



- Optimality of the RM and DM algorithms
  - We've already seen examples wherein these fixed-priority algorithms are not optimal
  - In fact, if the periods of the tasks in the system are related appropriately, then the RM and DM algorithms are optimal
  - A system of periodic tasks is ***simply periodic*** if for every pair of tasks T_i and T_k in the system and $p_i < p_k$, p_k is an integer multiple of p_i. (Recall our avionics example from lecture 2.)
  - Theorem: A system of simply periodic, independent, preemptable tasks whose relative deadlines are ≥ their periods is schedulable on one processor according to the RM algorithm iff its total utilization is ≤ 1.
  - Corollary: The same is true for the DM algorithm.
  - Since fixed-priority algorithms are more constrained, why would one choose to use them?
    - They often lead to more predictable and stable systems

Priority-driven Scheduling of Periodic Tasks



- Schedulable utilization of the RM algorithm
  - Assume $D_i = p_i$ for all I
  - Arbitrary relationships between relative deadlines of tasks
  - $\mathcal{U}_{RM}(n) = n (2^{1/n} – 1)$
  - For large n, approaches ln 2 (0.693)
  - $U(\mathcal{T}) \leq \mathcal{U}_{RM}(n)$ is a necessary condition – i.e. if $U(\mathcal{T}) > \mathcal{U}_{RM}(n)$, the RM algorithm (or better yet, the DM algorithm) may be able to find a feasible schedule

n	2	4	6	8	10	12	14	16	18	20	100
$\mathcal{U}_{RM}(n)$	0.828	0.757	0.735	0.724	0.718	0.714	0.710	0.708	0.707	0.705	0.696

Priority-driven Scheduling of Periodic Tasks



- What happens if the relative deadlines for tasks are not equal to their respective periods?
  - Assume that $D_k = \nu\, p_k$ for some number ν
  - $\mathcal{U}_{RM}(n, \nu)$ has the following values:

ν	$\mathcal{U}_{RM}(n, \nu)$
$0 \leq \nu \leq 0.5$	ν
$0.5 \leq \nu \leq 1$	$n ((2\nu)^{1/n} – 1) + 1 - \nu$
$\nu = 2, 3, \ldots$	$\nu (n – 1) ((1+1/\nu)^{1/n-1} – 1)$

Priority-driven Scheduling of Periodic Tasks



$n\backslash\nu$	4.0	3.0	2.0	1.0	0.9	0.8	0.7	0.6	0.5
2	0.944	0.928	0.898	0.828	0.783	0.729	0.666	0.590	0.500
3	0.926	0.906	0.868	0.779	0.749	0.708	0.656	0.588	0.500
4	0.917	0.894	0.853	0.756	0.733	0.698	0.651	0.586	0.500
5	0.912	0.888	0.844	0.743	0.723	0.692	0.648	0.585	0.500
6	0.909	0.884	0.838	0.734	0.717	0.688	0.646	0.585	0.500
7	0.906	0.881	0.834	0.728	0.713	0.686	0.644	0.584	0.500
8	0.905	0.878	0.831	0.724	0.709	0.684	0.643	0.584	0.500
9	0.903	0.876	0.829	0.720	0.707	0.682	0.642	0.584	0.500
∞	0.892	0.863	0.810	0.693	0.687	0.670	0.636	0.582	0.500

Priority-driven Scheduling of Periodic Tasks



- Practical factors
  - A ready job is ***blocked*** when it is prevented from executing by a lower-priority job
  - A ***priority inversion*** occurs whenever a lower-priority job executes while some ready higher-priority job waits
  - Nonpreemptability
    - Many reasons why a job may have nonpreemptable sections
      - Critical section over a resource
      - Some system calls are nonpreemptable
      - Disk scheduling
    - If a job becomes nonpreemptable, priority inversions may occur
    - When attempting to understand whether a task meets all of its deadlines, we must consider not only all the tasks that have higher priorities than that task, also the nonpreemptable portions of lower-priority tasks

Priority-driven Scheduling of Periodic Tasks



- Practical factors (continued)
  - Nonpreemptability (continued)
    - Define $b_i(np)$ as the longest amount of time for which any job in the task T_i can be blocked each time it is blocked due to nonpreemptive lower-priority tasks
  - Self-suspension
    - A job may invoke an external operation (e.g. request and I/O operation), during which time the job is suspended
  - Context Switches
    - Assume we know the maximum number of context switches K_i for a job in T_i
    - Can add $2(K_i + 1)\ t_{CS}$ to the execution time of T_i