***RTOS Problems and Solutions***

**1.**

 Because sporadic jobs may have varying release times and execution times, the periodic task model may be too inaccurate and can lead to undue under utilization of the processor even when the inter release times of jobs are bounded from below and their executions are bounded from above. As an example, suppose we have a [stream](javascript:void(0);) of sporadic jobs whose inter release times are uniformly distributed from 9 to 11. Their execution times are uniformly distributed from 1 to 3.  
  
a. What are the parameters of the periodic task if we were to use such a task to model the stream?  
  
Sol:  **For the periodic task model we model a task using the lower bound on its period and the upper bound           on its execution time (the worst case). In this case, the period, p = 9, and the exeuction time, e = 3.**  
  
b. [Compare](javascript:void(0);) the utilization of the periodic task in part (a) with the average utilization of the sporadic job [stream](javascript:void(0);).  
  
Sol: **The utilization of a periodic task is its execution time divided by its period. In this case:**  
**Uperiodic = eperiodic/pperiodic = 3/9 = 0.3333**  
 **Modeling the job as a**[**stream**](javascript:void(0);)**of periodic jobs, the execution time is a random variable E uniformly distributed from 1 to 3 time units, and the period is a random variable P uniformly distributed from 9 to 11. Utilization is a random variable that is a function of E and P. In particular, Usporadic = E/P. In general we can find the average value of U, E[U], we need to integrate u ⋅ fu(u), the probability density function of U from -infinity to infinity.**  
 **You can use the rules of probability to determine fu(u) from fe(e) and fp(p). In this case, after a bit more math than I anticipated we find:**

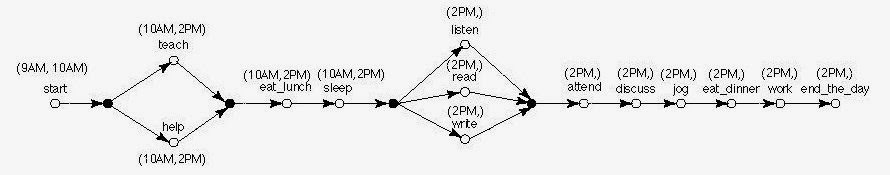
|  |  |  |
| --- | --- | --- |
|  | **0,** | **u < 1/11** |
| **121/8 - 1/(8u2),** | **1/11 ≤ u < 1/9** |
| **fu(u) =** | **5,** | **1/9 ≤ u < 3/11** |
|  | **9/(8u2) - 81/8,** | **3/11 ≤ u < 1/3** |
| **0,** | **1/3 ≤ u** |

1. **After integrating we find Usporadic = E[U] ≈ 0.20.**  
   **The utilization with the periodic task model is about 13 % more than if we use the average utilization.**  
     
    **Next Topic:**  
   **2.**

Consider the real-time program described by the psuedo code below. Names of jobs are in italic.  
  
At 9 [AM](javascript:void(0);), *start*: have breakfast and go to office;  
At 10 [AM](javascript:void(0);),  
     if there is class,  
     *teach*;  
     Else, *help* students;  
When *teach* or *help* is done, *eat\_lunch*;  
Until 2 PM, *sleep*;  
If there is a seminar,  
     If topic is interesting,  
*listen*;  
     Else, *read*;  
Else  
     *write* in office;  
When seminar is over, *attend* social hour;  
*discuss*;  
*jog*;  
*eat\_dinner*;  
*work* a little more;  
*end\_the\_day*;  
  
  
a) Draw a task graph to capture the dependencies among jobs.  
b) Use as many precedence graphs as needed to represent all the possible paths of the program.

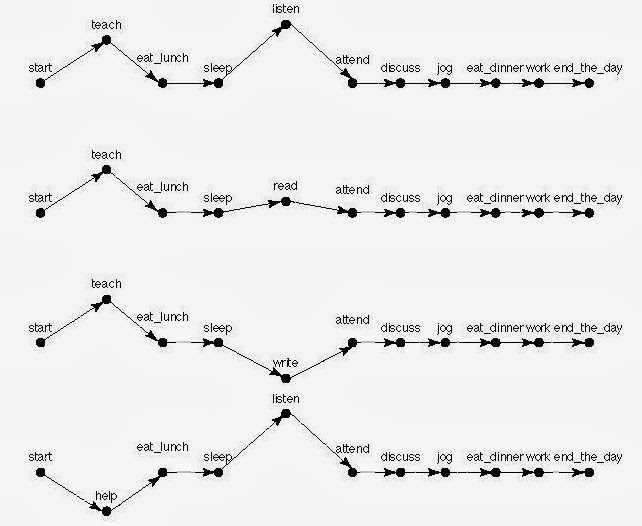
Sol:  **The book was a bit vague on some points, so there will be much flexibility in grading here. I've seen these drawn a number of different ways in different papers... The important part was to capture the timing and dependecies and make clear which dependencies were conditional.**

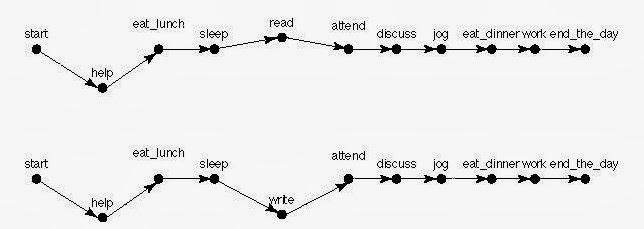
**The start time of *start*, *teach*, and *help* is given, so showing the feasible interval for them is important. The only timing constraint is that sleep has to end at 2PM, though so the deadline is looser than what one would expect. No timing constraints are given for *eat\_lunch*, so they could be left out or (10 AM, 2PM) would be reasonable. The deadline for sleeping is given, but no release time is given for *sleep* or*eat\_lunch*, so 10AM is the latest time we are bounded by. There is no mention of any time after *sleep* so we have no information on what the deadlines of any other tasks should be, unless we take "end of day" to be the literal end of day at midnight.**

[](http://1.bp.blogspot.com/-BYsKSIDqgbY/Um9hYyyoHUI/AAAAAAAAAA4/2Dr9EAbKocY/s1600/neerajblognow3.jpg)

b) Use as many precedence graphs as needed to represent all the possible paths of the program.

Sol: **Classical precedence graphs don't have conditional branches, so we have to draw each path separately... Also there is no timing information.**

[](http://4.bp.blogspot.com/-Qrn98vTL8wQ/Um9hlgh76sI/AAAAAAAAABA/TVWptNCTGiw/s1600/neerajblognow4.jpg)

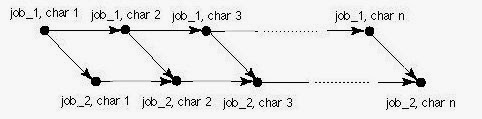
[](http://3.bp.blogspot.com/-hYXEST0VxAQ/Um9htlDpSfI/AAAAAAAAABI/uf6qnAmpkpc/s1600/neerajblognow5.jpg)

3.

*job\_1* | *job\_2* denotes a pipe: The result produced by *job\_1* is incrementally consumed by *job\_2*. (As an example, suppose that *job\_2* reads and displays one character at a time as each handwritten character is recognized and placed in a buffer by *job\_1*.) Draw a precedence constraint graph to represent this producer-consumer relation between the jobs.

Sol:

**To show the pipeline relationship *job\_1* is broken into smaller jobs, one per character with each job depending on the preceding one. Likewise, *job\_2* is broken up, but in addition to depending on the previous character, each job in *job\_2* depending on the corresponding character from *job\_1*.**

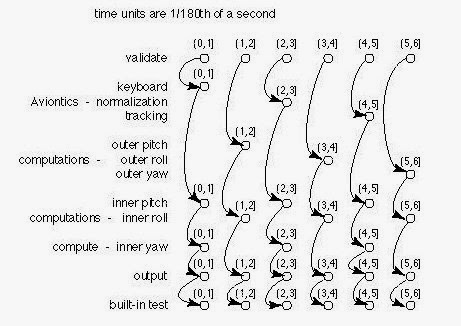
[](http://4.bp.blogspot.com/-9ywRGLoue3g/Um9lBQ8AWyI/AAAAAAAAABU/HzNgACBOE3U/s1600/neerajblognow6.jpg)

4.

 Draw a task graph to represent the flight control system described in Figure 1-3.

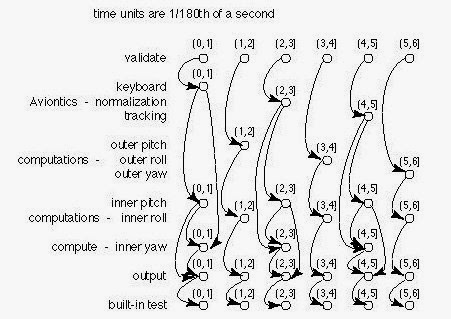
a) Assume the producers and consumers do not explicitly synchronize (i.e., each consumer uses the latest result generated by each of its producers but does not wait for the completion of the producer.)

Sol: **Producers and consumers do not synchronize, so there are no precedence constraints between producers and consumers. You may have drawn arrows to show precedence constraints between each job with the same release time, implied by the program listing. The whole schedule repeats every 6/180ths = 1/30th of a second.**

[](http://4.bp.blogspot.com/-scP4vlYjYb8/Um9lxYwoh6I/AAAAAAAAABc/-qCMqN1S9aE/s1600/neerajblognow7.jpg)

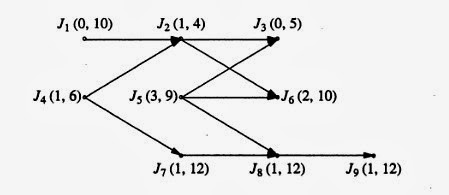
b) Repeat part (a), assuming that producers and consumers do synchronize.

Sol:  **The text says inner loops depend on outer loops and avionics tasks, output depends on inner loops. If you drew the constraints based on program order, only a few additional arcs need to be drawn because the program order causes the dependencies to be satisfied.**

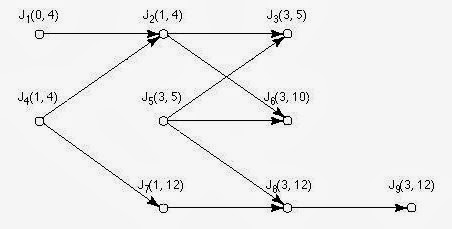
[](http://2.bp.blogspot.com/-5LH68DXYaYY/Um9mIbLmobI/AAAAAAAAABk/YJ7OKlA0v8E/s1600/neerakblognow8.jpg)

5.

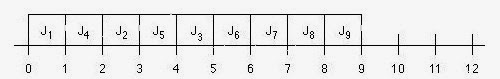
 The feasible interval of each job in the precedence graph in figure 4P-1 is given next to its name. The execution time of [all jobs](javascript:void(0);) are equal to 1.

[](http://3.bp.blogspot.com/-Z-Ip7Zwr64w/Um-tc87eHTI/AAAAAAAAACM/RSvd3nmGx8A/s1600/neerajblognow9.jpg)

a) Find the effective release times and deadlines of the jobs in the precendence graph in Figure 4P-1.  
  
Sol:  

[](http://4.bp.blogspot.com/-AMzrXHhIUT4/Um-sOZODiQI/AAAAAAAAAB0/oi7IEJHy7Ow/s1600/neerajblognow9+(1).jpg)

b) Find an EDF schedule of the jobs.  
  
Sol:

[](http://4.bp.blogspot.com/-W2z8i5WmGuY/Um-soGhkztI/AAAAAAAAAB8/fxDyTpLDrIc/s1600/neerajblognow9+(2).jpg)

c) A  job is said to be at level i if the length of the longest path from the job to jobs that have no successors is i. So, jobs J3, J6 and J9 are at level 0, jobs J2, J5 and J8 are at level 1, and so on. Suppose that the priorities of the jobs are assigned based on their levels, the heigher the level, the higher the priority. Find a priority-drive schedule of the jobs in Figure 4P-1 according to this priority assignment.

Sol:

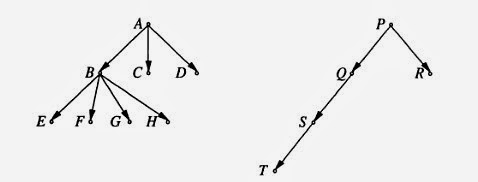
[](http://2.bp.blogspot.com/-qDvpW3dtKPc/Um-s_Deg_mI/AAAAAAAAACE/LBMwofLikN8/s1600/neerajblognow9+(3).jpg)

**Explanation:**

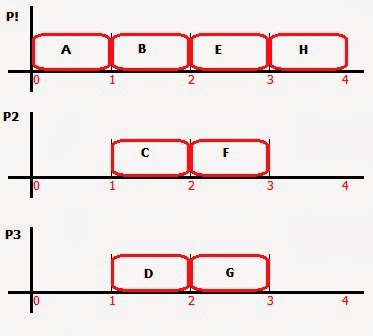
* **J1 is the only job released at t=0, so it goes first.**
* **At t=1, J2, J4, and J7 have been released. J4 has a level of 3, so it goes first.**
* **At t=2, J4is done. J7 has the next highest level (2), so it goes next.**
* **At t=3, J7 is done. J3, J5, J8, and J9 are released. J5 has the next highest level (2), so it runs.**
* **At t=4, J5 is done. Either J2 or J8 could run because both have a level of 1 and both have had their precedence contraints met. At this point J2 has already missed its deadline...**
* **At t=5, either J2 or J8, whichever was run at t=4, is done. The one that was not previously run gets to run. There are no more level 1 jobs.**
* **At t=6, J3, J6, and J9 are all eligible to run and are all at level 0. They can run in any order occording to this scheduling algorithm.**
* **J2 and J3 miss their deadlines. This is *not* an optimal**[**scheduling algorithm**](http://targetiesnow.blogspot.in/2013/10/real-time-system-by-jane-w-s-liu_9175.html)**.**

6.

 The execution times of the jobs in the precedence graph in figure 4P-2 are all equal to 1, and their release times are identical. Give a non preemptive optimal schedule that minimizes the completion time of [all jobs](javascript:void(0);) on three processors. Describe briefly the algorithm you used to find the schedule.

[](http://1.bp.blogspot.com/-0nIMwjEd6nU/Um-wzGqyiBI/AAAAAAAAACY/M06UxkI7GYM/s1600/neerajblognow10.jpg)

Sol:  **Execution time of**[**all jobs**](javascript:void(0);)**equal to 1. Release times are identical, non preemptive optimal solution:**

[](http://4.bp.blogspot.com/-h-At2riolpo/Um-0lCk_o5I/AAAAAAAAACk/lyQeIpj7zuI/s1600/neerajblognow11.jpg)

7.

 Consider a system that has five periodic tasks, A, B, C, D, and E, and three processors P1, P2, P3. The periods of A, B, and C are 2 and their execution times are equal to 1. The periods of D and E are 8 and their execution times are 6. The phase of every task is 0, that is, the first job of the task is released at time 0. The relative deadline of every task is equal to its period.  
  
a) Show that if the tasks are scheduled dynamically on three processors according to the LST algorithm, some jobs in the system cannot meet their deadlines.  
  
Sol:    
  
**At t=0, A, B, and C have 1 time unit of slack. D and E each have a slack of 2, so A, B, and C run first.**

**At t=1, A, B, and C are done running and the slack of D and E is 1, so D and E both get to run.**

**At t=2, A, B, and C are released again. Their slack is 1, as are the slacks of D and E. Assuming that once a job starts running on a processor, it cannot change processors, D and E will run round-robin on**[**two processors**](http://targetiesnow.blogspot.in/2013/10/real-time-system-by-jane-w-s-liu_1862.html)**with two of A, B, and C with the third running alone.**

**By time t=4, A, B, and C will have completed, and D and E will have completed one time unit of work.**

**At t=4, new jobs in A, B, and C are released with a slack of 1, but D and E have 0 slack. D and E run on two processors and A, B, and C run round-robin on the third.**

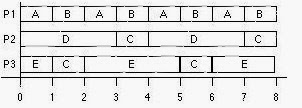
**At t=5.5 the A, B, and C's slack has fallen to 0. At that point all 5 tasks have a slack of 0 (i.e., they require the processor time from now until their deadline), but there are five jobs and only three processors. At least one job will finish past its deadline.**

**If jobs are allowed to change processors once they start, things are a bit more complicated. The five jobs run round-robin on three processors.**

**At t=1.6667, A, B, and C finish. D and E continue to run until t=2. Every 2 time units D and E only execute for 1.3333 time units, so at t=6 they have not completed.**

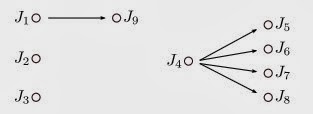
b) Find a feasible schedule of the five tasks on three processors.

Sol:

[](http://3.bp.blogspot.com/-XOIz-rv9jJk/Um-5jgCOQkI/AAAAAAAAAC0/_g-UkZv9Mjs/s1600/neerajblognow12.jpg)

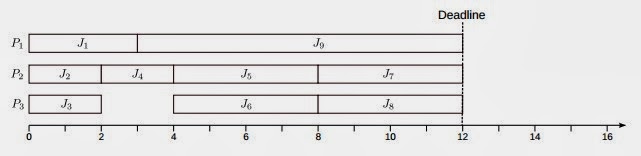
8.

 A system contains nine non-preemptable jobs named Ji, for i = 1, 2, ..., 9. Their execution times are 3, 2, 2, 2, 2, 4, 4, 4, 4, and 9, respectively, their release times are equal to 0, and their deadlines are 12. J1 is the immediate predecessor of J9, and J4 is the immediate predecessor of J5, J6, J7, and J8. There are no other precedence constraints. For all the jobs, Ji has a higher priority than Jk if i < k.  
  
a) Draw the precedence graph of the jobs.  
  
Sol:

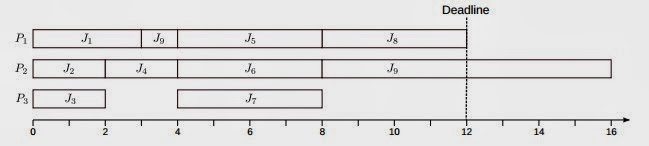
[](http://4.bp.blogspot.com/-5FIYwNVGV7U/Um_uIawH14I/AAAAAAAAADE/DjXikuw_n0U/s1600/neerajblognow13+(1).jpg)

b) Can the jobs meet their deadlines if they are scheduled on three processors? Explain your answer.

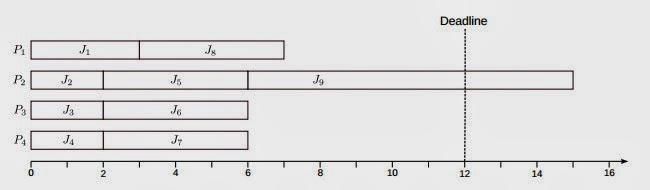
Sol: **All jobs meet their deadline.**

[](http://2.bp.blogspot.com/-dlQWpI7PQf8/Um_uYzM418I/AAAAAAAAADM/2Fh1S4KmLbM/s1600/neerajblognow13+(2).jpg)

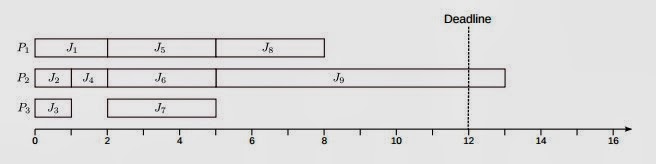
c) Can the jobs meet their deadlines if we make them preemptable and schedule them preemptively? Explain your answer.  
  
Sol: **Job J9 does not meet its deadline.**

[](http://3.bp.blogspot.com/-qfr3eu4MgT4/Um_uusJMBPI/AAAAAAAAADU/njw8IzBsaHU/s1600/neerajblognow13+(3).jpg)

d) Can the jobs meet their deadlines if they are scheduled nonpreemptively on four processors? Explain your answer.  
  
Sol: **Job J9 does not meet its deadline.**

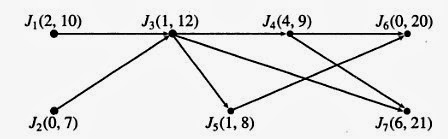
[](http://2.bp.blogspot.com/-qH4GfkUG04k/Um_vQ2TFieI/AAAAAAAAADc/p1-z2JC6CMU/s1600/neerajblognow13+(4).jpg)

e) Suppose that due to an [improvement](javascript:void(0);) of the three processors, the execution time of every job is reduced by 1. Can the jobs meet their deadlines? Explain your answer.  
  
Sol: **Job J9 does not meet its deadline.**

[](http://4.bp.blogspot.com/-hLFPtRIqYRo/Um_vkCwGgmI/AAAAAAAAADk/BT364hmVwK8/s1600/neerajblognow13+(5).jpg)

9.

 Consider the set of jobs in Figure 4-3. Suppose that the jobs have identical execution time. What maximum execution time can the jobs have and still can be feasible scheduling on one processor? Explain your answer.

[](http://4.bp.blogspot.com/-PZLD1UxpB7I/Um_yFtNwdUI/AAAAAAAAADw/Dbe1LDygbBk/s1600/neerajblognow14.jpg)

Sol: **Jobs with their effective release time and deadline are:**  
 **J1 (2,8)     J2 (0,7)     J3 (2,8)     J4 (4,9)     J5 (2,8)     J6 (4,20)     J7 (6,21)**  
 **Between 2 to 9, four jobs need to be fit.**  
**Hence, maximum execution time of each job is 1.75.**

10.

 Each of the following systems of periodic tasks is scheduled and executed according to a cyclic schedule. For each system, choose an appropriate frame size. Preemptions are allowed, but the number of preemption should be kept small.  
  
a) (6, 1), (10, 2), and (18, 2)  
  
Sol: **The frame size has to meet all three criteria discussed in the chapter.**

1. **f ≥ max(ei), 1 ≤ i ≤ n  
   f ≥ 2**
2. **f divides at least one of the periods evenly:  
   f ∈ {2, 3, 5, 6, 9, 10, 18}**
3. **2f - gdc(f, pi) ≤ Di, 1 ≤ i ≤ n**

**f = 2**

**2 × 2 - gcd(2, 6) = 2 - 2 = 0 ≤ 2  
2 × 2 - gcd(2, 10) = 2 - 2 = 0 ≤ 5  
2 × 2 - gcd(2, 18) = 2 - 2 = 0 ≤ 5**

**f = 3**

**2 × 3 - gcd(3, 6) = 6 - 3 = 3 > 2**

**f = 5**

**2 × 5 - gcd(5, 6) = 10 - 1 = 9 > 2**

**f = 6**

**2 × 6 - gcd(6, 6) = 12 - 6 = 6 > 2**

**f = 9**

**2 × 9 - gcd(9, 6) = 18 - 3 = 15 > 2**

**f = 18**

**2 × 10 - gcd(10, 6) = 20 - 2 = 18 > 2**

**f = 18**

**2 × 18 - gcd(18, 6) = 36 - 6 = 30 > 2**

**The only frame size that works for this set of tasks is f = 2.**

b) (8, 1), (15, 3), (20, 4), and (22, 6)

Sol: **The frame size has to meet all three criteria discussed in the chapter.**

1. **f ≥ max(ei), 1 ≤ i ≤ n  
   f ≥ 6**
2. **f divides at least one of the periods evenly:  
   f ∈ {1, 2, 3, 4, 5, 8, 10, 11, 15, 20, 22}**
3. **2f - gdc(f, pi) ≤ Di, 1 ≤ i ≤ n**

**f = 8**

**2 × 8 - gcd(8, 8) = 16 - 8 = 8 ≤ 8  
2 × 8 - gcd(8, 15) = 16 - 1 = 15 ≤ 15  
2 × 8 - gcd(8, 20) = 16 - 4 = 12 ≤ 20  
2 × 8 - gcd(8, 22) = 16 - 2 = 14 ≤ 22**

**f = 10**

**2 × 10 - gcd(10, 8) = 20 - 2 = 18 > 8**

**f = 11**

**2 × 11 - gcd(11, 8) = 22 - 1 = 21 > 8**

**f = 15**

**2 × 15 - gcd(15, 8) = 30 - 1 = 29 > 8**

**f = 20**

**2 × 20 - gcd(20, 8) = 40 - 4 = 36 > 2**

**f = 22**

**2 × 22 - gcd(22, 8) = 44 - 2 = 42 > 2**

**The only frame size that works for this set of tasks is f = 8.**

Clock-driven Cyclic Scheduler

• Since the parameters of [all jobs](javascript:void(0);) with hard deadlines are known can construct a static cyclic schedule in advance – Processor time allocated to a job equals its maximum execution time – Scheduler dispatches jobs according to the static schedule, repeating each hyperperiod – Static schedule guarantees that each job completes by its deadline  
• No job overruns ⇒ all deadlines are met  
• Schedule calculated off-line ⇒ can use complex algorithms – Run-time of the [scheduling algorithm](http://targetiesnow.blogspot.in/2013/10/real-time-system-by-jane-w-s-liu_5855.html) irrelevant – Can search for a schedule that optimizes some characteristic of the system  
 e.g. a schedule where the idle periods are nearly periodic; accommodating aperiodic jobs

Structured Cyclic Schedules

• Arbitrary table-driven cyclic schedules flexible, but inefficient – Relies on accurate timer interrupts, based on execution times of tasks – High scheduling overhead  
• Easier to implement if structure imposed: – Make scheduling decisions at periodic intervals (frames) of length f – Execute a fixed list of jobs with each frame, disallowing pre-emption except at frame boundaries – Require phase of each periodic task to be a non-negative integer multiple of the frame size  
• The first job of every task is released at the beginning of a frame  φ = k⋅f where k is a non-negative integer  
• Gives two benefits: – Scheduler can easily [check for](javascript:void(0);) overruns and missed deadlines at the end of each frame – Can use a periodic clock interrupt, rather than programmable timer

 Each of the following systems of periodic tasks is scheduled and executed according to a cyclic schedule. For each system, choose an appropriate frame size. Preemptions are allowed, but the number of preemptions should be kept small.

c)  (4, 0.5), (5, 1.0), (10, 2), and (24, 9)   
  
Sol: **The frame size has to meet all three criteria discussed in the chapter.**

1. **f ≥ max(ei), 1 ≤ i ≤ n  
   f ≥ 9**
2. **f divides at least one of the periods evenly:  
   f ∈ {2, 3, 4, 5, 6, 8, 10, 12, 24}**
3. **2f - gdc(f, pi) ≤ Di, 1 ≤ i ≤ n**

**f = 10**

**2 × 10 - gcd(10, 4) = 20 - 2 = 18 > 4**

**f = 12**

**2 × 12 - gcd(12, 4) = 24 - 4 = 20 > 4**

**f = 5**

**2 × 24 - gcd(24, 4) = 48 - 4 = 44 > 4**

**None of the possible frame sizes becuase e4 = 9 is too long. We have to split T4 into two smaller tasks. First try e4,1 = 4, and e4,2 = 5.**

1. **f ≥ max(ei), 1 ≤ i ≤ n  
   f ≥ 5**
2. **f divides at least one of the periods evenly:  
   f ∈ {2, 3, 4, 5, 6, 8, 10, 12, 24}**
3. **2f - gdc(f, pi) ≤ Di, 1 ≤ i ≤ n**

**f = 5**

**2 × 5 - gcd(5, 4) = 10 - 1 = 9 > 4**

**A frame size of 5 is still too big, as is a frame size of 4.5. We cannot make the frame size any smaller unless we break up the taks into smaller pieces. Try dividing T4 into three equal sized pieces with e4 = 3.**

1. **f ≥ max(ei), 1 ≤ i ≤ n  
   f ≥ 3**
2. **f divides at least one of the periods evenly:  
   f ∈ {2, 3, 4, 5, 6, 8, 10, 12, 24}**
3. **2f - gdc(f, pi) ≤ Di, 1 ≤ i ≤ n**

**f = 3**

**2 × 3 - gcd(3, 4) = 6 - 1 = 5 > 4**

**Even three is too big. We need to break up T4 further, try four tasks with execution time 2 and one with execution time 1.**

1. **f ≥ max(ei), 1 ≤ i ≤ n  
   f ≥ 2**
2. **f divides at least one of the periods evenly:  
   f ∈ {2, 3, 4, 5, 6, 8, 10, 12, 24}**
3. **2f - gdc(f, pi) ≤ Di, 1 ≤ i ≤ n**

**f = 2**

**2 × 2 - gcd(2, 4) = 4 - 2 = 2 ≤ 4  
2 × 2 - gcd(2, 5) = 4 - 1 = 3 ≤ 4  
2 × 2 - gcd(2, 10) = 4 - 2 = 2 ≤ 4  
2 × 2 - gcd(2, 24) = 4 - 2 = 2 ≤ 4**

**With this set of tasks f = 2 works.**

d)  (5, 0.1), (7, 1.0), (12, 6), and (45, 9)

Sol: **The frame size has to meet all three criteria discussed in the chapter.**

1. **f ≥ max(ei), 1 ≤ i ≤ n  
   f ≥ 9  
   The smallest period is 5, which is less than the longest execution time. We cannot have a frame size larger than the period, so at this point we know we have to split the (45, 9) task and the (12, 6) task. Splitting (45, 9) into two tasks does not leave many frame size choices. Try (45, 9) => (45, 3), (45, 3), (45, 3) and (12, 6) => (12, 3), (12, 3)  
   f≥ 3**
2. **f divides at least one of the periods evenly:  
   f ∈ {1, 2, 3, 4, 5, 6, 7, 9, 12, 15, 45}**
3. **2f - gdc(f, pi) ≤ Di, 1 ≤ i ≤ n**

**f = 3**

**2 × 3 - gcd(3, 5) = 6 - 1 = 5 ≤ 5  
2 × 3 - gcd(3, 7) = 6 - 1 = 5 ≤ 7  
2 × 3 - gcd(3, 12) = 6 - 3 = 3 ≤ 12  
2 × 3 - gcd(3, 45) = 6 - 3 = 3 ≤ 45**

**f = 4**

**2 × 4 - gcd(4, 5) = 8 - 1 = 7 > 5**

**f = 5**

**2 × 5 - gcd(5, 5) = 10 - 5 = 5 ≤ 5**

**2 × 5 - gcd(5, 7) = 10 - 1 = 9 > 7**

**The only frame size that works for this set of tasks is f = 3 (assuming the last two tasks are split as described above.)**

### Scheduling Aperiodic Jobs

• Aperiodic jobs are scheduled in the background after [all jobs](javascript:void(0);) with hard deadlines scheduled in each frame have completed

– Delays execution of aperiodic jobs in preference to periodic jobs – However, note that there is often no advantage to completing a hard real-time job early, and since an aperiodic job is released due to an event, the sooner such a job completes, the more responsive the system

• Hence, minimizing response times for aperiodic jobs is typically a design goal of real-time schedulers

### Slack Stealing

• Periodic jobs scheduled in frames that end before their deadline; there may be some slack time in the frame after the periodic job completes

• Since we know the execution time of periodic jobs, can move the slack time to the start of the frame, running the periodic jobs just in time to meet their deadline

• Execute aperiodic jobs in the slack time, ahead of periodic jobs – The cyclic executive keeps track of the slack left in each frame as the aperiodic jobs execute, preempts them to start the periodic jobs when there is no more slack – As long as there is slack remaining in a frame, the cyclic executive returns to examine the aperiodic job queue after each slice completes

• Reduces response time for aperiodic jobs, but requires accurate timers

11.

 Each of the following systems of periodic tasks is scheduled and executed according to a cyclic schedule. For each system, choose an appropriate frame size. Preemptions are allowed, but the number of preemptions should be kept small.  
  
  
e)  (5, 0.1), (7, 1.0), (12, 6), and (45, 9)  
  
Sol: **The frame size has to meet all three criteria discussed in the chapter.**

1. **f ≥ max(ei), 1 ≤ i ≤ n  
   f ≥ 9  
   The smallest period is 5, which is less than the longest execution time. We cannot have a frame size larger than the period, so at this point we know we have to split the (45, 9) task and the (12, 6) task. Splitting (45, 9) into two tasks does not leave many frame size choices. Try (45, 9) => (45, 3), (45, 3), (45, 3) and (12, 6) => (12, 3), (12, 3)  
   f≥ 3**
2. **f divides at least one of the periods evenly:  
   f ∈ {1, 2, 3, 4, 5, 6, 7, 9, 12, 15, 45}**
3. **2f - gdc(f, pi) ≤ Di, 1 ≤ i ≤ n**

**f = 3**

**2 × 3 - gcd(3, 5) = 6 - 1 = 5 ≤ 5  
2 × 3 - gcd(3, 7) = 6 - 1 = 5 ≤ 7  
2 × 3 - gcd(3, 12) = 6 - 3 = 3 ≤ 12  
2 × 3 - gcd(3, 45) = 6 - 3 = 3 ≤ 45**

**f = 4**

**2 × 4 - gcd(4, 5) = 8 - 1 = 7 > 5**

**f = 5**

**2 × 5 - gcd(5, 5) = 10 - 5 = 5 ≤ 5**

**2 × 5 - gcd(5, 7) = 10 - 1 = 9 > 7**

**The only frame size that works for this set of tasks is f = 3 (assuming the last two tasks are split as described above.)**

f)  (7, 5, 1, 5), (9, 1), (12, 3), and (0.5, 23, 7, 21)   
  
Sol: **The frame size has to meet all three criteria discussed in the chapter.**

1. **f ≥ max(ei), 1 ≤ i ≤ n  
   f ≥ 7  
   The smallest period is 5, which is less than the longest execution time. We cannot have a frame size larger than the period, so at this point we know we have to split the (0.5, 23, 7, 21) task. Splitting it into two tasks does not work (try it, to see). Split the long task into three (0.5, 23, 3, 21), (0.5, 23, 3, 21), and (0.5, 23, 2, 21)  
   f≥ 3**
2. **f divides at least one of the periods evenly:  
   f ∈ {1, 2, 3, 4, 5, 6, 9, 12, 23}**
3. **2f - gdc(f, pi) ≤ Di, 1 ≤ i ≤ n**

**f = 3**

**2 × 3 - gcd(3, 5) = 6 - 1 = 5 ≤ 5  
2 × 3 - gcd(3, 9) = 6 - 3 = 3 ≤ 9  
2 × 3 - gcd(3, 12) = 6 - 3 = 3 ≤ 12  
2 × 3 - gcd(3, 23) = 6 - 1 = 5 ≤ 21**

**f = 4**

**2 × 4 - gcd(4, 5) = 8 - 1 = 7 > 5**

**f = 5**

**2 × 5 - gcd(5, 5) = 10 - 5 = 5 ≤ 5**

**2 × 5 - gcd(5, 9) = 10 - 1 = 9 ≤ 9  
2 × 5 - gcd(5, 12) = 10 - 1 = 9 ≤ 12**

**2 × 5 - gcd(5, 23) = 10 - 1 = 9 ≤ 21**

**Either f = 3 or f = 5 may work, assuming the last two tasks are split as described above. We need to make a schedule to verify the tasks can be scheduled with those frame sizes.**

### Scheduling Sporadic Jobs

• We assumed there were no sporadic jobs

• Sporadic jobs have hard deadlines, release and execution times that are not known a priori – Hence, a clock-driven scheduler cannot guarantee a priori that sporadic jobs complete in time

• However, scheduler can determine if a sporadic job is schedulable when it arrives – Perform an acceptance test to [check](javascript:void(0);) whether the newly released sporadic job can be feasibly scheduled with all the jobs in the system at that time – If there is sufficient slack time in the frames before the new job’s deadline, the new sporadic job is accepted; otherwise, it is rejected

• Can be determined that a new sporadic job cannot be handled as soon as that job is released; earliest possible rejection – If more than one sporadic job arrives at once, they should be queued for acceptance in EDF order

### Practical Considerations

• Handling overruns: – Jobs are scheduled based on maximum execution time, but failures might cause overrun – A robust system will handle this by either: 1) killing the job and starting an error recovery task; or 2) preempting the job and scheduling the remainder as an aperiodic job

• Depends on usefulness of late results, dependencies between jobs, etc.

• Mode changes: – A cyclic scheduler needs to know all parameters of real-time jobs a priori – Switching between modes of operation implies reconfiguring the scheduler and bringing in the code/data for the new jobs – This can take a long time: schedule the reconfiguration job as an aperiodic or sporadic task to ensure other deadlines met during mode change

• Multiple processors: – Can be handled, but off-line scheduling table generation more complex

12.

 A system uses the cyclic EDF algorithm to schedule sporadic jobs. The cyclic schedule of periodic tasks in the system uses a frame size of 5, and a major cycle contains 6 frames. Supose that the initial amounts of slack time in the frames are 1, 0.5, 0.5, 0.5, 1, and 1.

1. Suppose that a sporadic job S(23, 1) arrives in frame 1, sporadic jobs S2(16, 0.8) and S3(20, 0.5) arrive in frame 2. In which frame are the accepted sporadic jobs scheduled?

**Sol:**

**S1(23, 1)**

**Since S1 arrives in frame 1, scheduling decisions about it are made at the start of frame 2. Frame 2 has a slack of 0.5, as does frame 3. Frame 3 ends at t=15 which is well before S1's deadline. The scheduler accepts S1 at the start of frame 2. If no other jobs arrive it would finish at the end of frame 3.**

**S2(16, 0.8)**

**The scheduler examines S2 at the start of frame 3 (t=10). The deadline, 16, is in frame 4, but there is only 0.5 slack in frame 3, so there is no way S2 can finish before its deadline. The scheduler rejects S2 at the start of frame 3.**

**S3(20, 0.5)**

**The scheduler examines S3 at the start of frame 3 (t=10). It's deadline is 20, which is the start of frame 5. There are 1.0 units of slack between frame 3 and frame 5, so the scheduler needs to see if S3can be scheduled without making any currently scheduled jobs miss their deadlines. S3 has an earlier deadline than S1 so if S3 were accepted, it would run for 0.5 time units at the end of frame 3 and S1would run for 0.5 time units at the end of frame 4. Since S1 has already executed for 0.5 time units at the end of frame 2, it will meet its deadline. S3 is accepted at the start of frame 3.**

1. Suppose that an aperiodic job with exeuction time 3 arrives at time 1. When will it be completed, if the systems does not do slack stealing?

Sol: **Call the aperiodic job A. When all the periodic jobs complete at the end of frame 1, the scheduler will let A execute until the start of frame 2, 1 time unit later. Frames 2, 3, and 4 have no slack because S1 and S3, from part (a), consume all of it. The scheduler runs A in the slack at the ends of frames 5 and 6. A completes at t=30, the end of frame 6.**

### Pro and Cons of Clock Driven Scheduling

• Simplicity  
 –Easy extension of frame based decision to event driven  
 –Decision made at clock ticks  
 • Events are queued  
 • Time driven polling  
 • Hard to maintain and modify  
 • Fixed release time and must be known in advance

### Real Time System by Jane W. S. Liu Chapter 5.3 Solution

13.

 Draw a network flow graph that we can use to find a preemptive cyclic schedule of the periodic tasks  
                                               
                                     T1 = (3,7,1);   T2 = (4,1);   T3 = (6,2.4,8).  
  
Sol: 

1. **f ≥ max(ei), 1 ≤ i ≤ n  
   f ≥ 2.4**
2. **f divides at least one of the periods evenly:  
   f ∈ {3, 4, 6,12}**
3. **2f - gdc(f, pi) ≤ Di, 1 ≤ i ≤ n**

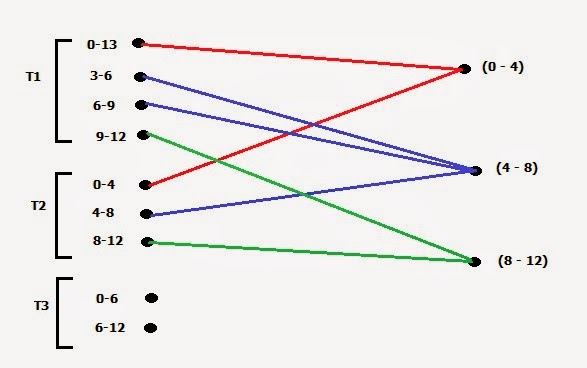
**Pi               Di             f = 3             f = 4           f = 6**

**3                7                3                   7                9**

**4                4                5                   4                10**

**6                8                3                   6                 6**

**Hence for f = 4, Network flow graph is-**

**[](http://3.bp.blogspot.com/-ij5jQ7qRtOI/Um__qWR2-BI/AAAAAAAAAEA/UnxsK98Z4q0/s1600/neerajblognow15.jpg)**

**T3 can't be scheduled.**

#### Assumptions for Clock-driven scheduling

• Clock-driven scheduling applicable to deterministic systems

• A restricted periodic task model: – The parameters of all periodic tasks are known a priori – For each mode of operation, system has a fixed number, n, periodic tasks

• For task Ti each job Ji,k is ready for execution at its release time ri,k and is released pi units of time after the previous job in Ti such that ri,k = ri,k-1 + pi • Variations in the inter-release times of jobs in a periodic task are negligible – Aperiodic jobs may exist • Assume that the system maintains a single queue for aperiodic jobs

• Whenever the processor is available for aperiodic jobs, the job at the head of this queue is executed – There are no sporadic jobs

#### Notation for Clock-driven scheduling

• The 4-tuple Ti = (φi, pi, ei, Di) refers to a periodic task Ti with phase φi, period pi, execution time ei, and relative deadline Di – Default phase of Ti is φi = 0, default relative deadline is the period Di = pi.

Omit elements of the tuple that have [default values](http://targetiesnow.blogspot.in/2013/10/real-time-system-by-jane-w-s-liu_4437.html)

#### The clock-driven approach has many advantages:

- conceptual simplicity;

- we can take into account complex dependencies, communication delays, and resource contentions among jobs in the choice and [construction](javascript:void(0);) of the static schedule;

- static schedule stored in a table; change table to change operation mode;

- no need for concurrency control and synchronization mechanisms;

- [context switch](http://targetiesnow.blogspot.in/2013/10/real-time-system-by-jane-w-s-liu_4437.html) overhead can be kept low with large frame sizes. It is possible to further simplify clock-driven scheduling

- sporadic and aperiodic jobs may also be time-triggered (interrupts in response to external events are queued and polled periodically);

- the periods may be chosen to be multiples of the frame size.

- Easy to validate, test and certify (by exhaustive simulation and testing).

- Many traditional real-time applications use clock-driven schedules.

- This approach is suited for systems (e.g. small embedded controllers) which are rarely modified once built.

14.

 A system contains the following periodic tasks:   
  
T1 = (5,1);     T2 = (7,1,9);     T3 = (10,3) and T4 = (35,7).  
  
If the frame size constraint (5-1) is ignored, what are the possible frame sizes ?  
  
Sol:

1. **f ≥ max(ei), 1 ≤ i ≤ n  
   This step is ignored, here.**
2. **f divides at least one of the periods evenly:  
   f ∈ {2, 5, 7, 10, 14, 35}**
3. **2f - gdc(f, pi) ≤ Di, 1 ≤ i ≤ n**

**Pi          Di          f = 2        f = 5         f = 7        f = 10         f = 14         f = 35**

**5           5             3             5              13(x)       15(x)          27(x)           65(x)**

**7           9             3             9               7            19(x)          21(x)           63(x)**

**10         10           2             5              13(x)       10              26(x)            65(x)**

**35         35           3             5               7            15              20                35(x)**

## Cyclic scheduling: frame size

• Decision points at regular intervals(frames);

• Within a frame the processor may be idle to accommodate aperiodic jobs

• The first job of every task is released at the beginning of some frame

• How to determine the frame size f ?

• The following 3 constraints should be satisfied:

1. f ≥ max(ei)            (for 1 ≤ i ≤ n) (n tasks)

• each job may start and complete within one frame: no job is preempted

2. [pi /f]- pi /f = 0     (for at least one i)

• to keep the cyclic schedule short, f must divide the hyperperiod H; this is true if f divides at least one pi

3.  2f – gcd(pi,f) ≤ Di    (for 1 ≤ i ≤ n)

• to have at least one whole frame between the release time and the deadline of every job (so the job can be feasibly scheduled in that frame)

#### Constructing a cyclic schedule

 Design steps and decisions to consider in the process of constructing a cyclic schedule:

ƒdetermine the hyperperiod H,

ƒdetermine the total utilization U (if >1 schedule is unfeasible),

ƒchoose a frame size that meets the constraints,

ƒpartition jobs into slices, if necessary,

ƒplace slices in the frames.

#### • The clock-driven approach has many disadvantages:

- brittle: changes in execution time or addition of a task often require a new schedule to be constructed;

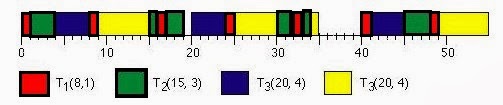
- release times must be fixed (this is not required in priority-driven systems);

- all combinations of periodic tasks that might execute at the same time must be known a priori: it is not possible to reconfigure the system on line (priority-driven systems do not have this restriction);

- not suitable for many systems that contain both hard and soft real-time applications: in the clock-driven systems previously discussed, aperiodic and sporadic jobs were scheduled in a priority driven manner (EDF).

15.

 A system **T** contains for periodic tasks, (8, 1), (15, 3), (20, 4), and (22, 6). Its total utilization is 0.80. Construct the initial segment in the time interval (0, 50) of a rate-monotonic schedule of the system.  
  
Sol: **The scheduling will be as-**

[](http://4.bp.blogspot.com/-JVhLYEQ6BeQ/UnCBGXqvzJI/AAAAAAAAAEQ/QDLhRwk3X9Y/s1600/neerajblognow16.jpg)

## Schedulability Test for RMA

An important problem that is addressed during the design of a uniprocessor-based real-time system is to [check](javascript:void(0);) whether a set of periodic real-time tasks can feasibly be scheduled under RMA. Schedulability of a task set under RMA can be determined from a knowledge of the worst-case execution times and periods of the tasks. A pertinent question at this point is how can a system developer determine the worst-case execution time of a task even before the system is developed. The worst-case execution times are usually determined experimentally or through simulation studies.

The following are some important criteria that can be used to [check](javascript:void(0);) the schedulability of a set of tasks set under RMA.

### Necessary Condition

A set of periodic real-time tasks would not be RMA schedulable unless they satisfy the following necessary condition:

i= ∑ei / pi = ∑ui ≤ 1

where ei is the worst case execution time and pi is the period of the task Ti, n is the number of tasks to be scheduled, and ui is the [CPU utilization](http://targetiesnow.blogspot.in/2013/10/real-time-system-by-jane-w-s-liu_1931.html)due to the task Ti. This test simply expresses the fact that the total CPU utilization due to all the tasks in the task set should be less than 1.

### Sufficient Condition

The derivation of the sufficiency condition for RMA schedulability is an important result and was obtained by Liu and Layland in 1973. A formal derivation of the Liu and Layland’s results from first principles is beyond the scope of this [discussion](javascript:void(0);). We would subsequently refer to the sufficiency as the Liu and Layland’s condition. A set of n real-time periodic tasks are schedulable under RMA, if i=∑ui ≤ n(21/n − 1) (3.4/2.10)

where ui is the utilization due to task Ti. Let us now examine the implications of this result. If a set of tasks satisfies the sufficient condition, then it is guaranteed that the set of tasks would be RMA schedulable

16.

 Which of the following systems of periodic tasks are schedulable by the rate-monotonic algorithm? By the earliest-deadline-first algorithm? Explain your answer.

1. T = {(8, 3), (9, 3), (15, 3)}

Sol:   **URM(3) ≈ 0.780**

**U = 3/8 + 8/9 + 3/15 = 0.908 > URM  
schedulable utilization test is indeterminateFor RM, shortest period is highest priority**  
**w1(t) = 3, W1 = 3 ≤ 8, ∴ T1 is schedulable  
w2(t) = 3 + ⌈t/8⌉⋅3 = t  
W2 = 6 ≤ 9, ∴ T2 is schedulable  
w3(t) = 3 + ⌈t/8⌉⋅3 + ⌈t/9⌉⋅3 = t  
W3 = 15 ≤ 15, ∴ T3 is schedulable.**  
**All tasks are schedulable under RM, therefore the system is schedulable under RM.**  
**U ≤ 1, ∴ the system is schedulable under EDF**

1. T = {(8, 4), (12, 4), (20, 4)}

Sol:**U = 4/8 + 4/12 + 4/20 ≈ 1.03 > 1  
∴ this system is not schedulable by any**[**scheduling algorithm**](http://targetiesnow.blogspot.in/2013/11/real-time-system-by-jane-w-s-liu_6564.html)

1. T = {(8, 4), (10, 2), (12, 3)}

Sol:  **U = 4/8 + 2/10 + 3/12 = 0.95 > URM(3)**  
**Schedulable utilization test is indeterminate, use time-demand analysis,  
w1(t) = 4, W1 = 4 ≤ 8  
∴ T1 is schedulable  
w2(t) = 2 + ⌈ t/8 ⌉⋅4 = t  
W2 = 6 ≤ 10  
∴ T2 is schedulable  
w3(t) = 2 + ⌈ t/8 ⌉⋅4 + ⌈ t/10 ⌉⋅2 = t  
W3 = 15 > 12  
∴T3 misses its deadline**  
**This system is not schedulable under RM**  
**U ≤ 1 ∴ this system is schedulable under EDF**

## Earliest Deadline First (EDF) Scheduling

In Earliest Deadline First (EDF) scheduling, at every scheduling point the task having the  shortest deadline is taken up for scheduling. This basic principles of this algorithm is very intuitive and simple to understand. The schedulability test for EDF is also simple. A task set is schedulable under EDF, if and only if it satisfies the condition that the total processor utilization due to the task set is less than 1.

EDF has been proven to be an optimal uniprocessor scheduling algorithm. This means that, if a set of tasks is not schedulable under EDF, then no other scheduling algorithm can feasibly schedule this task set. In the simple schedulability test for EDF, we assumed that the period of each task is the same as its deadline. However, in practical problems the period of a task may at times be different from its deadline. In such cases, the schedulability test needs to be changed.

A more efficient implementation of EDF would be as follows. EDF can be implemented by maintaining all ready tasks in a sorted priority queue. A sorted priority queue can efficiently be implemented by using a heap data structure. In the priority queue, the tasks are always kept sorted according to the proximity of their deadline. When a task arrives, a record for it can be inserted into the heap in O(log2 n) time where n is the total number of tasks in the priority queue.

At every scheduling point, the next task to be run can be found at the top of the heap. When a task is taken up for scheduling, it needs to be removed from the priority queue. This can be achieved in O(1) time.

17.

 a) Use the time demand analysis method to show that the rate-monotonic algorithm will produce a feasible schedule of the tasks (6,1), (8,2) and (15,6).  
  
Sol:**U = 1/6 + 2/8 + 6/15 = 0.816**  
 **TDA analysis-**  
 **w1(6) = 1,                                                          W1 = 1 ≤ 6,             ∴ T1 is schedulable**  
**w2(6) = 2 + ⌈6/6⌉⋅1 = 3                                    W2 = 3 ≤ 6,             ∴ T2 is schedulable   
w3(6) = 6 + ⌈6/6⌉⋅1 + ⌈6/8⌉⋅2 = 9**

**w3(12) = 6 + ⌈12/6⌉⋅1 + ⌈12/8⌉⋅2 = 12**

b) Change the period of one of the tasks in part (a) to yield a set of tasks with the maximal total utilization which is feasible when scheduled using the rate-monotonic algorithm. (Consider only integer values for period)

Sol: **Change P1 such that-**

**w3(15) = 6 + ⌈12/P1⌉⋅1 + 4 = 15          =>            P1 = 3**

c) Change the execution time of one of the tasks in part (a) to yield a set of tasks with the maximum total utilization which is feasible when scheduled using the rate-monotonic algorithm. (Consider only register values for the execution time).

Sol:**Change the execution time of tasks such that maximum possible utilization**

**w3(15) =  e3 + ⌈15/6⌉⋅1 + ⌈15/8⌉⋅2 = 15**

**=>                 e3 + 3 + 4 = 15**

**=>                 e3= 8**

**=>                 T3 = (15,8)**    

## Rate Monotonic Scheduling

The term rate monotonic derives from a method of assigning priorities to a set of processes as a monotonic function of their rates.  While rate monotonic [scheduling systems](javascript:void(0);) use rate monotonic theory for actually scheduling sets of tasks, rate monotonic analysis can be used on tasks scheduled by many different systems to reason about schedulablility. We say that a task is schedulable if the sum of its preemption, execution, and blocking is less than its deadline. A system is schedulable if all tasks meet their deadlines. Rate monotonic analysis provides a mathematical and scientific model for reasoning about schedulability.

#### Assumptions

Reasoning with rate monotonic analysis requires the presence of the following assumptions :

• Task switching is instantaneous.

• Tasks account for all execution time.

• Task interactions are not allowed.

• Tasks become ready to execute precisely at the beginning of their periods and relinquish the CPU only when execution is complete.

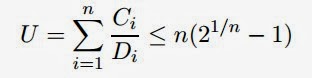
• Task deadlines are always at the start of the next period.

• Tasks with shorter periods are assigned higher priorities; the criticality of tasks is not considered.

• Task execution is always consistent with its rate monotonic priority: a lower priority task never executes when a higher priority task is ready to execute.

18.

 Give two different explanation of why the periodic tasks (2,1), (4,1) and (8,2) are schedulable by the rate monotonic algorithm.  
  
Sol:   **The priorities to tasks are assigned statically, before the actual execution of the task set. Rate Monotonic scheduling scheme assigns higher priority to tasks with smaller periods. It is preemptive (tasks are preempted by the higher priority tasks). It is an optimal**[**scheduling algorithm**](http://targetiesnow.blogspot.in/2013/11/real-time-system-by-jane-w-s-liu_2.html)**among ﬁxed-priority algorithms; if a task set cannot be scheduled with RM, it cannot be scheduled by any ﬁxed-priority algorithm.**  
**The sufﬁcient schedulability test is given by:**

[](http://4.bp.blogspot.com/-p93hOWtLgjI/UnCFW3JtBAI/AAAAAAAAAEc/VaXq3boRNTY/s1600/targeties1.jpg)

**The term U is said to be the processor utilization factor (the fraction of the processor time spent on executing task set). n is the number of tasks.**

**In our case: 1/2 + 1/4 + 2/8 = 1  which is not less than 0.78**

**The above condition is not necessary; we can do a somewhat more involved sufﬁcient and necessary condition test, as follows.**

**We have to guarantee that all the tasks can be scheduled, in any possible instance. In particular, if a task can be scheduled in its critical instances, then the schedulability  guarantee condition holds (a critical instance of a task occurs whenever the task is released simultaneously with all higher priority tasks). For that, we have to use the method as mentioned in Exercise 6.5.**

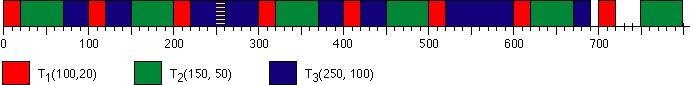
19.

This problem is concerned with the performance an behavior of rate-monotonic an earliest-deadline-first algorithms.

1. Construct the initial segments in the time interval (0, 750) of a rate-monotonic schedule and an earliest-deadline-first schedule of the periodic tasks (100, 20) (150, 50), and (250, 100) whose total utilization is 0.93.

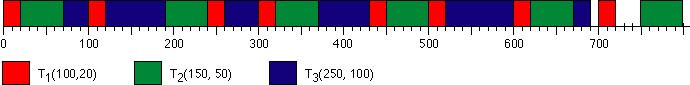
**Sol:**

**RM**

****

**Note, the third task (the blue one) runs past its deadline from t = 250 to t = 260.**

**EDF**

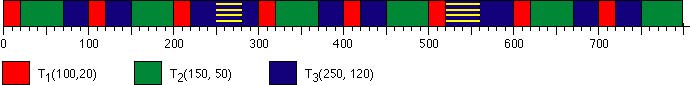
****

**There are no missed deadlines in this schedule.**

1. Construct the initial segments in the time interval (0, 750) of a rate-monotonic schedule and an earliest-deadline-first schedule of the periodic tasks (100, 20) (150, 50), and (250, 120) whose total utilization is 1.01.

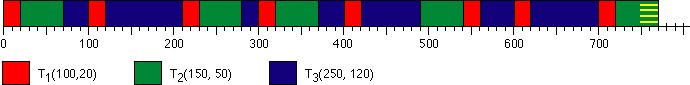
**Sol:**

**RM**

****

**The third task (the blue one) runs past its deadline from 250 to 280 and from 520 to 560. The third task will continue to be backlogged farther and farther each time a new job in the task is released, but the first and second task are not affected.**

**EDF**

****

**Task 2 eventually misses its deadline. Once jobs start missing deadlines, almost every job is going to miss its deadline.**

## Rate Monotonic vs. EDF

Since the ﬁrst results published in 1973 by Liu and Layland on the Rate Monotonic (RM) and Earliest Deadline First (EDF) algorithms, a lot of progress has been made in the schedulability analysis of periodic task sets. Unfortunately, many misconceptions still exist about the properties of these two scheduling methods, which usually tend to favor RM more than EDF. Typical wrong statements often heard in technical conferences and even in [research papers](javascript:void(0);) claim that RM is easier to analyze than EDF, it introduces less runtime overhead, it is more predictable in overload conditions, and causes less jitter in task execution. Since the above statements are either wrong, or not precise, it is time to clarify these issues in a systematic fashion, because the use of EDF allows a better exploitation of the available resources and signiﬁcantly improves system’s performance.

Most commercial RTOSes are based on RM. RM is simpler to implement on top of commercial (fixed priority) kernels.

EDF requires explicit kernel support for deadline scheduling, but gives other advantages.

Less overhead due to preemptions.

More uniform jitter control

Better aperiodic responsiveness.

Two different types of overhead:

**Overhead for job release**

EDF has more than RM, because the absolute deadline must be updated at each job activation

**Overhead for context switch**

RM has more than EDF because of the higher number of preemptions

**Resource access protocols:**

For RM

Non Preemptive Protocol (NPP)

Highest Locker Priority (HLP)

Priority Inheritance (PIP)

Priority Ceiling (PCP)

Under EDF

Non Preemptive Protocol (NPP)

Dynamic Priority Inheritance (D-PIP)

Dynamic Priority Ceiling (D-PCP)

Stack Resource Policy (SRP)

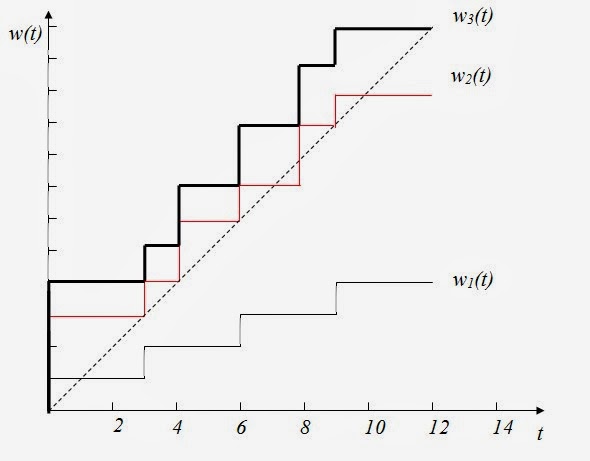
20.

 The Periodic Tasks (3,1), (4,2), (6,1) are scheduled according to the rate-monotonic algorithm.

a)      Draw Time Demand Function of the tasks

|  |
| --- |
|  |
|  |  |

       Sol:

[](http://4.bp.blogspot.com/-jn0-Ta3XVwg/UnDFNPHDWSI/AAAAAAAAAEs/JK4uCphd2Dc/s1600/targeties2.jpg)

         b)     Are the tasks schedulable? Why or why not ?

         Sol: **No. Based on the Time Demand Function graph, Task 3 did not touch or                    go below the dash line by its deadline at time 6. In another word, it can                      not meet its deadline and therefore not schedulable.**

c)     Can this graph be used to determine whether the tasks are schedulable according to an arbitrary priority-driven algorithm?

Sol: **No. This graph is fundamentally based on fixed priority driven algorithm which assigns the same priority to**[**all jobs**](javascript:void(0);)**in each task. In the graph, T2 is built on top of T1 since**[**all jobs**](javascript:void(0);)**in T1 have a higher priority than**[**all jobs**](javascript:void(0);)**in T2. T3 is built on top of T1 and T2since**[**all jobs**](javascript:void(0);)**in T1 and T2 have a higher priority than all jobs in T3. This graph does not depict dynamic priority driven algorithm, such as earliest deadline first (EDF). In EDF, any job in a task can have a higher priority at a specific moment depending on its deadline compared to the jobs of other tasks. Therefore, this graph cannot be used to determine the schedulability of an arbitrary priority-driven algorithm.**

## Time-demand Analysis

Simulate system behaviour at the critical instants. For each job Ji,c released at a critical instant, if Ji,c and all higher [priority tasks](http://targetiesnow.blogspot.in/2013/11/real-time-system-by-jane-w-s-liu_4718.html) complete executing before their relative deadlines the system can be scheduled. Compute the total demand for processor time by a job released at a critical instant of a task, and by all the higher-priority tasks, as a function of time from the critical instant; [check](javascript:void(0);) if this demand can be met before the deadline of the job:

 Consider one task, Ti, at a time, starting highest priority and working down to lowest priority.   [Focus](javascript:void(0);) on a job, Ji, in Ti, where the release time, t0, of that job is a critical instant of T

[Compare](javascript:void(0);) time-demand function, wi(t), and available time, t:

• If wi(t) ≤ t at some t ≤ Di, the job, Ji, meets its deadline, t0 + Di

• If wi(t) > t for all 0 < t ≤ Di then the task probably cannot complete by its deadline; and the system likely cannot be scheduled using a fixed priority algorithm

• Note that this is a sufficient condition, but not a necessary condition. Simulation may show that the critical instant never occurs in practice, so the system could be feasible

Use this method to [check](javascript:void(0);) that all tasks are can be scheduled if released at their critical instants; if so conclude the entire system can be scheduled. The time-demand, wi(t), is a staircase function with steps at multiples of higher priority task periods Plot the time-demand versus available time graphically, to get intuition into approach

21.

   Which of the following fixed-[priority task](http://targetiesnow.blogspot.in/2013/11/real-time-system-by-jane-w-s-liu_3.html) is not schedulable? Explain your answer.

T1(5,1)            T2(3,1)            T3(7,2.5)         T4(16,1)

|  |
| --- |
|  |
|  |  |

Sol:**If Wi(t) <= t, the task is schedulable.**

**Assume RM/DM**[**scheduling algorithm**](http://targetiesnow.blogspot.in/2013/11/real-time-system-by-jane-w-s-liu_3.html)**is used. Priority: T2>T1>T3>T4**

**Index, i, is assigned to each task according to its priority.**

**T2: i = 1, T1: i = 2, T3: i = 3., T4: i = 4**

**Check: t = 3, 5, 6, 7, 9, 10, 12, 14, 15, 16**

**W1(t) = 1 < t,             t= 3, 5, 6, 7, 9, 10, 12, 14, 15, 16          => Schedulable**

**W2(t) = 1 + ⌈t/3⌉**

**W2(3) = 1+1 = 2 <=3                              => Schedulable**

**W3(t) = 2.5 + ⌈t/3⌉ + ⌈t/5⌉     (check: 2.5+1+1=4.5 => min t =5)**

**W3(5) = 2.5+2+1 = 5.5**

**W3(6) = 2.5+2+2 = 6.5**

**W3(7) = 2.5+3+2 = 7.5 => Miss deadline, 7 Not Schedulable**

**W4(t) = 1 + ⌈t/3⌉ + ⌈t/5⌉ + 2.5⌈t/7⌉      (check: 1+1+1+2.5=5.5 => min t =6)**

**W4(6) = 1+2+2+2.5 = 7.5**

**W4(7) = 1+3+2+2.5 = 8.5**

**W4(9) = 1+3+2+5 = 11**

**W4(10) = 1+4+2+5 = 12**

**W4(12) = 1+4+3+5 = 13**

**W4(14) = 1+5+3+5 = 14 <= 14                    => Schedulable**

**T3 is not a schedulable task**

## Time Bound in Fixed-Priority Scheduling

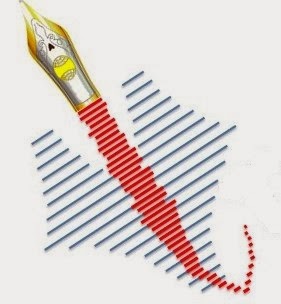
Since worst-case response times must be determined repeatedly during the interactive design of real-time application systems, repeated exact computation of such response times would slow down the design process considerably. In this research, we identify three desirable properties of estimates of the exact response times: continuity with respect to system parameters; efﬁcient computability; and approximability. We derive a technique possessing these properties for estimating the worst case response time of sporadic task systems that are scheduled using ﬁxed priorities upon a preemptive uniprocessor .

When a group of tasks share a common resource (such as a processor, a communication medium), a scheduling policy is necessary to arbitrate access to the shared resource. One of the most intuitive policies consists of assigning Fixed Priorities (FP) to the tasks, so that at each instant in time the resource is granted to the highest priority task requiring it at that instant. Depending on the assigned priority, a task can have longer or shorter response time, which is the time elapsed from request of the resource to the completion of the task.

Since worst case response times must be determined repeatedly during the interactive design of real-time application systems, repeated exact computation of such response times would slow down the design process considerably. In this research, we identify three desirable properties of estimates of the exact response times: continuity with respect to system parameters, efficient computability, and approximability. We derive a technique possessing these properties for estimating the worst-case response time of sporadic task systems that are scheduled using fixed priorities upon a preemptive uniprocessor.

22.

 Find the maximum possible response time of tasks T4 in the following fixed-priority system by solving the equation w4(t) = t, iteratively  
  
                    T1 = (5,1),   T2 = (3,1),     T3 = (8,1.6),   and   T4 = (18,3.5)  
  
Sol:  **Iteration 1:**  
**w4(t=1)(1)  = 3.5 + ⌈ 1/5 ⌉⋅1 + ⌈ 1/3 ⌉⋅1 + + ⌈ 1/8 ⌉⋅1.6**  
**= 3.5 + 1 + 1 + 1.6**  
**= 7.1**  
**Iteration 2:**  
**w4(t=7)(2)  = 3.5 + ⌈ 7/5 ⌉⋅1 + ⌈ 7/3 ⌉⋅1 + + ⌈ 7/8 ⌉⋅1.6**  
**= 3.5 + 2 + 3 + 1.6**  
**= 10.1**  
**Iteration 3:**  
**w4(t=10)(3)  = 3.5 + ⌈ 10/5 ⌉⋅1 + ⌈ 10/3 ⌉⋅1 + + ⌈ 10/8 ⌉⋅1.6**  
**= 3.5 + 2 + 4 + 3.2**  
**= 12.7**  
**Iteration 4:**  
**w4(t=12.7)(4)  = 3.5 + ⌈ 12.7/5 ⌉⋅1 + ⌈ 12.7/3 ⌉⋅1 + + ⌈ 12.7/8 ⌉⋅1.6**  
**= 3.5 + 3 + 5 + 3.2**  
**= 14.7**  
**Iteration 5:**  
**w4(t=14.7)(5)  = 3.5 + ⌈ 14.7/5 ⌉⋅1 + ⌈ 14.7/3 ⌉⋅1 + + ⌈ 14.7/8 ⌉⋅1.6**  
**= 3.5 + 3 + 5 + 3.2**  
**= 14.7**  
 **Max possible response time = 14.7**

[](http://3.bp.blogspot.com/-IVbr6M_dxbA/UnXzMH4_2OI/AAAAAAAAARQ/--7u6lUDo1I/s1600/SnapCrab_NoName_2013-10-31_14-3-37_No-00.jpg)

## Time Bound in Fixed-Priority Scheduling

Since worst-case response times must be determined repeatedly during the interactive design of real-time application systems, repeated exact computation of such response times would slow down the design process considerably. In this research, we identify three desirable properties of estimates of the exact response times: continuity with respect to system parameters; efﬁcient computability; and approximability. We derive a technique possessing these properties for estimating the worst case response time of sporadic task systems that are scheduled using ﬁxed priorities upon a preemptive uniprocessor .

When a group of tasks share a common resource (such as a processor, a communication medium), a scheduling policy is necessary to arbitrate access to the shared resource. One of the most intuitive policies consists of assigning Fixed Priorities (FP) to the tasks, so that at each instant in time the resource is granted to the highest priority task requiring it at that instant. Depending on the assigned priority, a task can have longer or shorter response time, which is the time elapsed from request of the resource to the completion of the task.

Since worst case response times must be determined repeatedly during the interactive design of real-time application systems, repeated exact computation of such response times would slow down the design process considerably. In this research, we identify three desirable properties of estimates of the exact response times: continuity with respect to system parameters, efficient computability, and approximability. We derive a technique possessing these properties for estimating the worst-case response time of sporadic task

systems that are scheduled using fixed priorities upon a preemptive uniprocessor.

23.

 Find the length of an in-phase level-3 busy interval of the following fixed-priority tasks:

              T1 = (5, 1), T2 = (3,1), T3 = (8, 1.6), and T4 = (18, 3.5)

Sol: **The level-3 busy interval is based on T1, T2, and T3**  
**t = ⌈ t/5 ⌉⋅1 + ⌈ t/3 ⌉⋅1 + ⌈ t/8 ⌉⋅1.6**  
**t = 4.6 = length of in-phase level-3 busy interval**

## Busy Intervals

Definition:  A level-πi  busy interval (t0, t] begins at an instant t0 when

(1) [all jobs](javascript:void(0);) in Ti released before this instant have completed, and

(2) a job in Ti is released.

The interval ends at the first instant t after t0 when [all jobs](javascript:void(0);) in Ti released since t0 are complete. For any t that would qualify as the end of a level-πi busy interval, a corresponding t0 exists. During a level-πi busy interval, the processor only executes tasks in Ti other tasks can be ignored.

Definition: We say that a level-πi busy interval is in phase if the first job of all tasks that execute in the interval are released at the same time. For systems in which each task’s relative deadline is at most its period, we argued that an upper bound on a task’s response time could be computed by considering a “critical instant” scenario in which that task releases a job together with all higher-priority tasks. In other words, we just consider the first job of each task in an in-phase system. For many

years, people just assumed this approach would work if a task’s relative deadline could exceed its period. Lehoczky showed that this “folk wisdom”  that only each task’s first job must be considered is false by means of a counterexample.

The [general](javascript:void(0);) schedulability test hinges upon the assumption that the job with the maximum response occurs within an in-phase busy interval.

24.

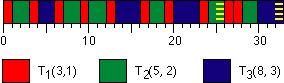
 A system consists of three periodic tasks: (3, 1), (5, 2), and (8, 3).

1. What is the total utilization?

Sol: **U = 1/3 + 2/5 + 3/8 ≈ 1.11**

1. Construct an earliest-deadline-first schedule of this system in the interval (0, 32). Label any missed deadlines.

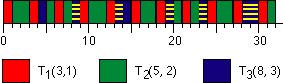
Sol:



**Yellow stripes indicates missed deadlines.**

1. Suppose we want to reduce the execution time of the task with period 3 in order to make mthe task system schedulable according to the earliest-deadline-first algoorithm. What is the minimum amount of reduction mecessary for the system to be schedulable by the earliest-deadline-first algorithm?

Sol:



**Yellow stripes indicates missed deadlines.**

1. Suppose we want to reduce the execution time of the task with period 3 in order to make the task system schedulable according to the earliest-deadline-first algoirthm. What is the minimum amount of reduction necessary for the system to be schedulable by the earliest-deadline-first algorithm?

Sol:        **U = (1-x)/3 + 2/5 + 3/8 ≤ 1  
              x ≥ 0.325**

### Utilization Bounds for EDF Scheduling

        The utilization bound for Earliest Deadline First scheduling is extended

 from uniprocessors to homogeneous multiprocessor systems with partitioning strategies. First results are provided for a basic task model, which includes periodic and independent tasks with deadlines equal to periods. n bounds depend on the allocation algorithm, diﬀerent allocation algorithms have been considered, ranging from simple heuristics to optimal allocation algorithms.

        As multiprocessor utilization bounds for EDF scheduling depend strongly on task sizes, all these bounds have been obtained as a function of a parameter which takes task sizes into account.Theoretically, the utilization bounds for multiprocessor EDF scheduling can be considered a partial solution to the bin-packing problem, which is known to be NP-complete. The basic task model is extended to include resource sharing release jitter, deadlines less than periods, aperiodic tasks, non-preemptive sections, [context switches](http://targetiesnow.blogspot.in/2013/11/real-time-system-by-jane-w-s-liu_9426.html) and mode changes.

25.

 a) Use the time-demand analysis method to show that the set of periodic tasks {(5, 1), (8, 2), (14, 4)} is schedulable according to the rate-monotonic algorithm.  
  
Sol:  **Shortest period has the highest priority...**

**T1 (5, 1): w1(t) = 1  
                                                 W1 = 1 ≤ 5  
                                                ∴ T1 is schedulable**

**T2 (8, 2): w2(t) = 2 + ⌈t/5⌉⋅1  
                                                W2 = 3 ≤ 8  
                                                ∴ T2 is schedulable**

**T3 (14, 4): w3(t) = 4 + ⌈t/5⌉⋅1 + ⌈t/8⌉⋅2  
                                                 W3 = 8 ≤ 14  
                                                 ∴ T3 is schedulable**

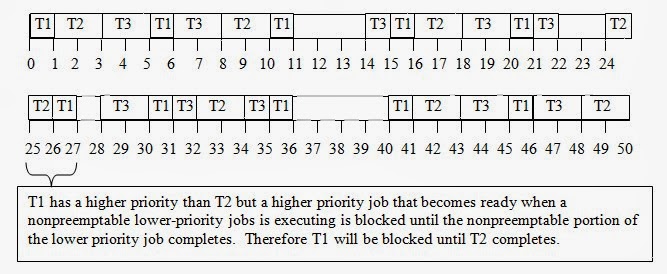
b)    Suppose that we want to make the first x units of each request in the task (8,2) nonpreemptable.  What is the maximum value of x so that the system remains schedulable according to the rate-monotonic algorithm?

Solution 1:

**T={(5,1)(8,2)(14,4)}**

**T1=(5,1)          T2=(8,2)          T3=(14,4)  (in order of priority, T1 being highest)**

|  |
| --- |
|  |
|  |  |

**[](http://2.bp.blogspot.com/-jUOg3_rxJhk/UnDTgqvbCKI/AAAAAAAAAE8/Z7Exf-TlMW8/s1600/targeties3.jpg)**

**T2 (8, 2) can be made nonpreemptable for the first 2 time units (its entire duration) and still allow the system to be scheduled on time.**

Solution 2:

**If we make the first x units of Task (8, 2) nonpreemptable: T3 is unaffected by this change since T2 is a higher**[**priority task**](http://targetiesnow.blogspot.in/2013/11/real-time-system-by-jane-w-s-liu_1209.html)**anyway.  T2 is also unaffected.  Its response time will not be affected by the change (if anything it would improve)**

**W1= x+1 <=5, x<=4**

**x can be at most 4 time units.  But since Task 2 (8, 2), only has an execution time of 2 time units, x can be 2 time units**

26.

 A system contains tasks T1 = (10,3), T2 = (16,4), T3 = (40,10) and  T4 = (50,5). The total blocking due to all factors of the tasks are b1 = 5, b2 = 1, b3 = 4 and b4 = 10, respectively. These tasks are scheduled on the EDF basis. Which tasks (or task) are (or is) schedulable? Explain your answer.  
  
Sol:  **For ith task to be scheduled by EDF basis**

**[http://4.bp.blogspot.com/-FcFKhMruZEU/UnDYNx35FRI/AAAAAAAAAFI/qL6ZK7OkGcs/s1600/targeties4.jpg](http://4.bp.blogspot.com/-FcFKhMruZEU/UnDYNx35FRI/AAAAAAAAAFI/qL6ZK7OkGcs/s1600/targeties4.jpg)**

**= 3/10 + 4/16 + 10/40 + 5/50**  
**= 0.3 + 0.25 + 0.25 + 0.1**  
**= 0.9**  
**for T1:**  
**0.9 + b1/10 = 0.9 + 5/10 = 1.4 > 1       ...not schedulable**  
 **for T2:**  
**0.9 + 1/16 = 0.9 + 0.0625 = 0.9625 < 1  ...schedulable**  
 **for T3:**  
**0.9 + 4/40 = 0.9 + 0.01 = 1 <= 1           ...schedulable**

**for T4:**  
**0.9 + 10/50 = 0.9 + 0.2 = 1.1 > 1          ...not schedulable**

# Earliest deadline first scheduling

**Earliest deadline first** (**EDF**) or **least time to go** is a dynamic [scheduling algorithm](http://en.wikipedia.org/wiki/Scheduling_algorithm) used in real-time operating systems to place processes in a priority queue. Whenever a scheduling event occurs (task finishes, new task released, etc.) the queue will be searched for the process closest to its deadline. This process is the next to be scheduled for execution.

when the system is overloaded, the set of processes that will miss deadlines is largely unpredictable (it will be a function of the exact deadlines and time at which the overload occurs.) This is a considerable disadvantage to a real time [systems designer](http://targetiesnow.blogspot.in/2013/11/real-time-system-by-jane-w-s-liu_5.html). The algorithm is also difficult to implement in hardware and there is a tricky issue of representing deadlines in different ranges (deadlines must be rounded to finite amounts, typically a few bytes at most). If a modular arithmetic is used to calculate future deadlines relative to now, the field storing a future relative deadline must accommodate at least the value of the (("duration" {of the longest expected time to completion} \* 2) + "now"). Therefore **EDF** is not commonly found in industrial real-time [computer systems](javascript:void(0);).

Instead, most real-time [computer systems](javascript:void(0);) use fixed [priority scheduling](http://targetiesnow.blogspot.in/2013/11/real-time-system-by-jane-w-s-liu_5.html) (usually rate-monotonic scheduling). With fixed priorities, it is easy to predict that overload conditions will cause the low-priority processes to miss deadlines, while the highest-priority process will still meet its deadline.

EDF is an *optimal* [scheduling algorithm](http://targetiesnow.blogspot.in/2013/11/real-time-system-by-jane-w-s-liu_5.html) on preemptive uniprocessors, in the following sense: if a collection of independent *jobs,* each characterized by an arrival time, an execution requirement and a deadline, can be scheduled (by any algorithm) in a way that ensures all the jobs complete by their deadline, the **EDF** will schedule this collection of jobs so they all complete by their deadline.

27.

  Interrupts typically arrive sporadically. When an interrupt arrives, interrupt handling is serviced (i.e., executed on the processor) immediately and in a nonpreemptable fashion. The effect of interrupt handling on the schedulability of periodic tasks can be accounted for in the same manner as blocking time. To illustrate this, consider a system of four tasks: T1 = (2.5, 0.5), T2 = (4, 1), T3 = (10, 1), and T4 = (30, 6). Suppose that there are two streams of interrupts. The interrelease time of interrupts in one [stream](javascript:void(0);) is never less than 9, and that of the other [stream](javascript:void(0);) is never less than 25. Suppose that it takes at most 0.2 units of time to service each interrupt. Like the periodic tasks interrupt handling tasks (i.e., the [stream](javascript:void(0);) of interrupt handling jobs) are given fixed prioriteies. They have higher priorities than the periodic tasks, and the one with a higher rate (i.e., shorter minimum interrelease time) has a higher priority.

1. What is the maximum amount of time each job in each periodic task may be delayed from completion by interrupts?

Sol:**If an interrupt comes while a job is running or a higher priority job is running, the job's completion time will be delayed by the interrupt service time, eint. The maximum delay for Ti comes when all tasks with higher priority than Ti release jobs at the same time as Ti. Interrupts behave like a higher priority task.**

**b1 = ⌈e1/pint,1⌉⋅eint,1 + ⌈e1/pint,2⌉⋅eint,2 = ⌈0.5/9⌉⋅0.2 +⌈0.5/25⌉⋅0.2 = 0.4**

**w2(t) = 1 + 0.5⋅⌈t/2.5⌉ + 0.2⋅⌈t/9⌉ + 0.2 ⋅⌈t/25⌉ = t**

**W2 = 1.9**

**The amount of time taken by interrupt handlers between the release of the first job in T2 and it's completion time is:**

**b2 = 0.2 ⋅⌈1.9/9⌉ + 0.2 ⋅⌈1.9/25⌉ = 0.4**

**w3(t) = 1 + 0.5⋅⌈t/2.5⌉ + 1⋅⌈t/4⌉ + 0.2⋅⌈t/9⌉ + 0.2 ⋅⌈t/25⌉ = t**

**W3 = 3.4**

**b3 = 0.2 ⋅⌈3.4/9⌉ + 0.2 ⋅⌈3.4/25⌉ = 0.4**

**w4(t) = 6 + 0.5⋅⌈t/2.5⌉ + 1⋅⌈t/4⌉ + 1⋅⌈t/10⌉ + 0.2⋅⌈t/9⌉ + 0.2 ⋅⌈t/25⌉ = t**

**W4 = 17.1**

**b4 = 0.2 ⋅⌈17.1/9⌉ + 0.2 ⋅⌈17.1/25⌉ = 0.6**

1. Let the maximum delay suffered by each job in Ti in part (a) be bi, for i = 1, 2, 3, and 4. Compute the time-demand functions of the tasks and use the time-demand analysis method to determine whether every periodic task Ti can meet all its deadlines if Di is equal to pi.

Sol: **(This is a bit redundant given part (a) above.)**

**w1(t) = b1 + e1 = 0.4 + 0.5 = 0.9 = t**

**W1 = 0.9 ≤ 2.5**

**∴T1 is schedulable**

**w2(t) = 0.4 + 1 + 0.5⋅⌈t/2.5⌉ = t**

**W2 = 1.9 ≤ 4**

**∴T2 is schedulable**

**w3(t) = 0.4 + 1 + 0.5⋅⌈t/2.5⌉ + 1⋅⌈t/4⌉ = t**

**W3 = 3.4 ≤ 10**

**∴T3 is schedulable**

**w4(t) = 0.6 + 6 + 0.5⋅⌈t/2.5⌉ + 1⋅⌈t/4⌉ + 1⋅⌈t/10⌉ = t**

**W4 = 17.1 ≤ 30**

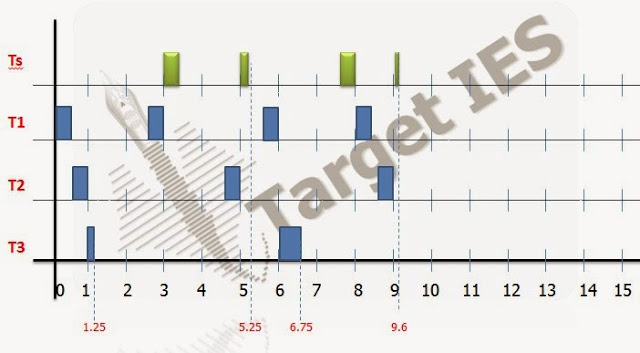
**∴T4 is schedulable**

1. In one or two sentences, explain why the answer you obtained in (b) about the schedulability of the periodic tasks is correct and the method you use works not only for this system but also for all independent preemptive periodic tasks.

Sol:**The interrupt behavior described in the problem is the same as the behavior of a high priority periodic task. Therefore, the amount of time taken handling interrupts can be analyzed with the same method as high priority tasks.**

28.

 A system contains three periodic tasks. They are (2.5,1), (4,0.5), (5,0.75), and their total utilization is 0.475.  
  
a) The system also contains a periodic server (2,0.5). The server is scheduled with the periodic tasks rate-monotonically.  
        1) Suppose that the periodic server is a basic sporadic server. What are the response time of the following two aperiodic jobs: One arrives at 3 and has execution time 0.75, and one arrives at 7.5 and has execution time 0.6.  
  
Sol:

[](http://1.bp.blogspot.com/-_XhUs0vP8c0/UnHrBfmAZwI/AAAAAAAAAHQ/UcSFRlbwjG0/s1600/targeties10+(1).jpg)

**WA1 = 4.25 - 3   = 1.25**

**WA2 = 9.6 - 7.5 = 2.1**

                   2) Suppose that the periodic server is a deferrable server. What are the response times of the above two aperiodic jobs.

Sol:

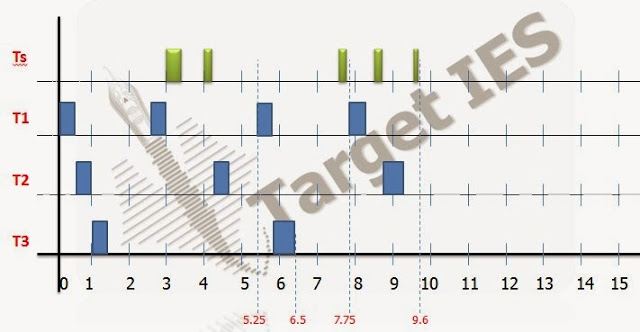
[](http://1.bp.blogspot.com/-sWfBs518IXE/UnHrJHerp_I/AAAAAAAAAHY/Bvs37Uuk9TA/s1600/targeties10+(2).jpg)

**WA1   = 4.25 - 3  = 1.25**

**WA2 = 8.1 - 7.5 = 0.6**

b) Note that the utilization of the periodic server in part (a) is 0.25. We can give the server different periods while keeping [its utilization](http://targetiesnow.blogspot.in/2013/11/real-time-system-by-jane-w-s-liu_6.html) fixed at 0.25. Repeat (1) and (2) in part (a) if the period of the periodic server is 1.

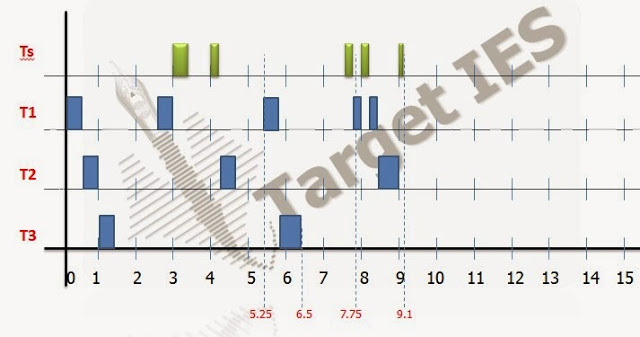
Sol: **Case 1:**

**[](http://2.bp.blogspot.com/-MbIhdAL2AXE/UnHrQZcnjyI/AAAAAAAAAHg/9_hoJxMX1zU/s1600/targeties10+(3).jpg)**

**WA1 = 5.25 - 3  = 2.25**

**WA2 = 9.6 - 7.5 = 2.1**

**Case 2:**

[](http://2.bp.blogspot.com/-gwKKZ4ROwXQ/UnHrZ0p34JI/AAAAAAAAAHo/ph8j46KmZHc/s1600/targeties10+(4).jpg)

**WA1 = 5.25 - 3  = 2.25**

**WA2 = 9.1 - 7.5 = 1.6**

c) Can we improve the response times by increasing the period of the periodic server ?

Sol: **Lengthening the period can improve response time because a layer period allows us to increase the execution time so that more of the sporadic job can run but period must remain short enough to replenish the budget before the next aperiodic job arrives and short enough to remain the highest**[**priority task**](http://targetiesnow.blogspot.in/2013/11/real-time-system-by-jane-w-s-liu_6.html)**.**

d) Suppose that as a designer you were given (1) the characteristics of the periodic tasks, that is, (p1,e1), (p2,e2),.....(pn,en), (2) the minimum interval pa between arrivals of aperiodic jobs, and (3) the maximum execution time required to complete any aperiodic job. Suppose that you are asked to choose the execution budget and period of a deferrable server. Suggest a set of good design rules.

Sol:  **1. Periodic Starts**

**2. Maximum Pa**

**3. Minimum  ea**

**If Ps = Pa and es = ea, then all aperiodic jobs will finish as soon as possible. If the system is unschedulable, make the period as long as possible (and the budget), so that as many jobs can finish as soon as possible.**

29.

 A system contains three periodic tasks. They are (3,1), (4,0.5), (5,0.5).

   The task system also contains a sporadic server whose period is 2. The sporadic server is scheduled with the periodic tasks rate-monotonically. Find the maximum utilization of the server if all deadlines of periodic tasks are surely met.

           1) Suppose that the server in part (a) is a pure polling server. What are the response time of the following two aperiodic jobs: one arrives at 2.3 and has [execution time](http://targetiesnow.blogspot.in/2013/11/real-time-system-by-jane-w-s-liu_2439.html) 0.8 , and one arrives at 12.7 and has execution time 0.6 ?

Sol:   **Find max es so system is schedulable.**

**W1(t) = 1 + ⌈t/2⌉.es = t**

**=> W2(t) = 1 +  ⌈3/2⌉.es = 3  =>  es = 1**

**W2(t) = 0.5 +  ⌈t/2⌉.es + ⌈t/3⌉ = t**

**=> es = 7/4**

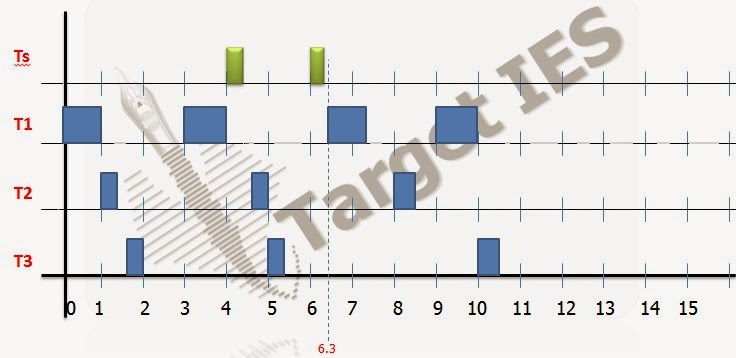
**W3(t) = 1 +  ⌈t/2⌉.es + ⌈t /3⌉ + ⌈t/4⌉.0.5 = t**

**=> es = 0.5**

**Hence, es =< 0.5,**

**U = es/Ps = 0.5/2 = 1/4**

**so,**

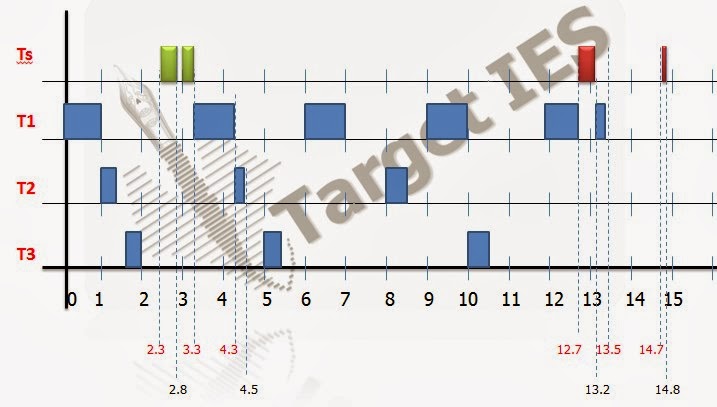
**[](http://1.bp.blogspot.com/-0ovw8mRWM5k/UnHsDR8KwEI/AAAAAAAAAHw/4Rxrml1jZAs/s1600/targeties11+(1).jpg)**

**WA1 = 6.3 - 2.3 = 4**

**WA2 = 16.1 - 12.7 = 3.4**

             2) Suppose that the server in part (a) is a basic sporadic server. What are the response time of the above two aperiodic jobs?

             Sol:

[](http://2.bp.blogspot.com/--NlRg4lToZg/UnHsKKKjxeI/AAAAAAAAAH4/eJGegZ4wua8/s1600/targeties11+(2).jpg)

**WA1 = 3.3 - 2.3 = 1**

**WA2 = 14.8 - 12.7 = 2.1**

:About the Sporadic Server:

• DS may delay lower-priority tasks.

• Sporadic Servers (SS) rules ensure that each sporadic server (ps, es) never demands more processor time than the periodic task (ps, es).

Sporadic Server in Fixed-Priority Systems: Notations

• T: system of n independent, preemptable periodic tasks.

• TH: subset of periodic tasks with higher priorities than the server priority.

• T/ TH are either busy or idle.

• Server busy interval: [an aperiodic job arrives at an empty queue, the queue becomes empty again].

• tr: the latest (actual) replenishment time.

• tf: the first instant after tr at witch server begins to execute.

• te: the latest effective replenishment time.

• At any time t:

–BEGIN: beginning instant of the earliest busy interval among the latest contiguous equence of busy intervals of TH that started before t.

–END: end of the latest busy interval if this interval ends before t, infinity if the interval ends after t.

 Simple Sporadic Server

• Consumption Rules: At any t > tr, budget is consumed at the rate of 1 per unit time until budget is exhausted when

– C1: the server is executing OR

– C2: the server has executed since tr and END < t.

• Replenishment Rules:

– R1: Initially when system begins execution and each time when budget is replenished, budget = es and tr = current time.

– R2: At time tf,

if END = tf then te = max(tr, BEGIN),

 if END < tf then te = tf. Next replenishment time is te + ps.

– R3: a) If te + ps is earlier than tf, budget is replenished as soon as it is exhausted.

        b) If T becomes idle before te + ps and becomes busy again at tb, budget is replenished at min(te + ps, tb).

30.

 Consider a system containing the following periodic tasks: T1 = (10,2), T2 = (14,3) and T3 = (21,4). A periodic server of period 8 is used to schedule aperiodic jobs.  
  
a) Suppose that the server and the tasks are scheduled rate-monotonically.  
       
         1) If the periodic server is a deferrable server, how large can its maximum execution budget be ?  
  
Sol:**For sufficient condition (because Ps is highest priority server)**  
 **U <= URM**  
  
**U <= (n-1)[(u + 2)/(u+1) - 1]**  
**1/3**

**2/10 + 3/14 + 4/21 + Us  <=  3[(u + 2)/(u+1) - 1]**

**Us  <= 0.11059**

**es = Us.Ps = 0.8879**

          2) If the periodic server is a sporadic server, how large can its maximum execution budget be ?

Sol:**If we try to find es using TDA.**

**Suppose final job completes exactly at deadline.**

**21 = w3(21) = 4 + es + es⋅⌈(21 - es)/8⌉ + 1⋅⌈21/10⌉ + 3⋅⌈21/14⌉**

**5 =  es + + es⋅⌈(21 - es)/8⌉**

**by solving,  es = 1.25**

**So, we can take maximum execution budget as 1.25.**

b) Suppose that the server and the tasks are scheduled on the EDF basis. Repeat the (a).

Sol: **tasks are schedulable if,**

**U <= URM**  
  
**<= (n-1)[(u + 2)/(u+1) - 1]**  
  
**2/10 + 3/14 + 4/21 + es⋅⌈(8 - es)/21  +  1⌉ <= 1**  
 **by solving, es = 2.506**

**So, we can take maximum execution budget as 2.506.**

31.

 Consider a system that contains two periodic tasks T1 = (7,2) and T2 = (10,3). There is a bandwidth preserving server whose period is 6. suppose that the periodic tasks and the server are scheduled rate-monotonically.

        a) Suppose that the server is deferrable server.

                1) What is the maximum [server size](http://targetiesnow.blogspot.in/2013/11/real-time-system-by-jane-w-s-liu_8.html) ?

       Sol:   **Max size:**

**W1(t) = 2 + es + ⌈(t - es)/6⌉.es = t**

**2 + es + ⌈(t - es)/6⌉.es < 7**

**es + ⌈(t - es)/6⌉.es < 5**

**simplifies to-**

**1 - X + 2(1 - X) < 5**

**3 - 3X  < 5**

**es + es  < 5**

**es = 2.5**

**W2(t) = 3 + es + ⌈(t - es)/6⌉.es + ⌈t/2⌉.es = t**

**3 + es + ⌈(10 - es)/6⌉.es + ⌈10/2⌉.es < 10**

**put, es = 4 - X,**

**es < 4,    3 + 4 - X + 2(4 - X) + 4 < 10**

**15 - 3X + 4 < 10**

**3X < 9**

**X = 3**

**es <= 1**

**es > 4,    3 + 4 + X + 2(4 + X) + 4 < 10**

**13 + 2X < 10**

**2X = -3**

**X = -1.5**

**put es = 4 + X**

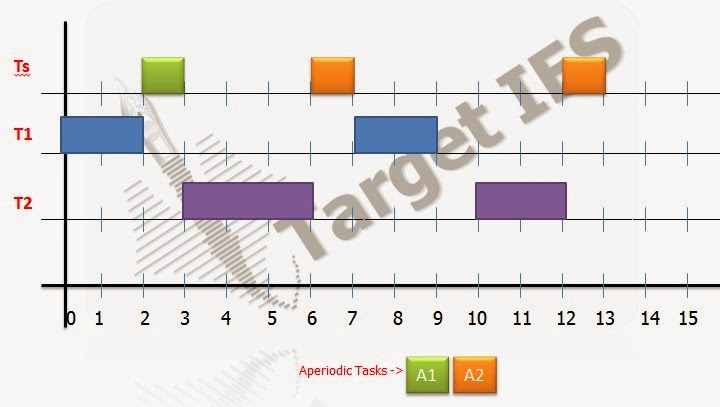
**es <= 4 - 1.5 = 2.5**

**Hence,**

**es = 1  =>  U = es/Ps = 1/6**

               2) Consider two aperiodic jobs A1 and A2. The execution times of the jobs are equal to 1.0 and 2.0, respectively. Their arrival times are 2 and 5. What are their response time ?

                Sol:

[](http://2.bp.blogspot.com/-PYieuS4KH8g/UnHs8xVWAVI/AAAAAAAAAIQ/yM1eGZTm-lc/s1600/targeties12+(1).jpg)

**WA1 = 3 - 2 = 1**

**WA2 = 13 - 5 = 8**

            b)  Suppose that the server is a simple sporadic or SpSL sporadic server.

                       1) What is the maximum server size ?

                   Sol:

**W1(t) = 2 +  ⌈t/6⌉.es = t**

**2 +  ⌈7/6⌉.es < 7**

**es <= 2.5**

**W2(t) = 3 +  ⌈t/6⌉.es  + ⌈t/7⌉.2 = t**

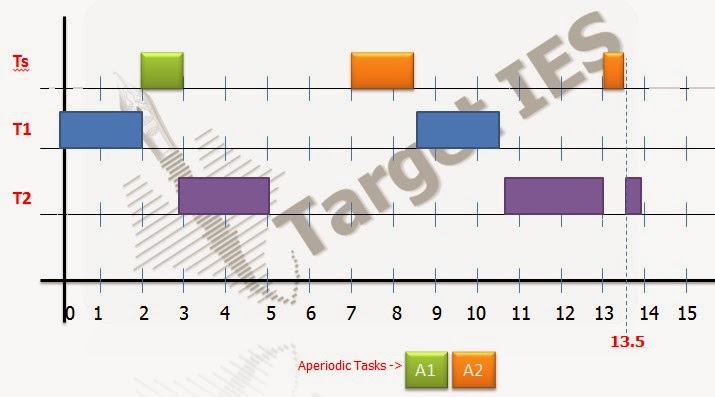
**3 +  ⌈10/6⌉.es  + ⌈10/7⌉.2 < 10**

**es <= 1.5**

**U = es/Ps = 1.5/6**

                       2) Find the response times of jobs A1 and A2 in part (a) if the server is a SpSL server.

                    Sol:

[](http://3.bp.blogspot.com/-A5GBzJmIg3Y/UnHtDDOo1bI/AAAAAAAAAIY/1TYvpvUbXu0/s1600/targeties12+(2).jpg)

**WA1 = 3 - 2 = 1**

**WA2 = 13.5 - 5 = 8.5**

32.

 Suppose that the periodic tasks in the previous problem are scheduled along with a server on the earliest deadline first basis.

a) What is the maximum [server size](http://targetiesnow.blogspot.in/2013/11/real-time-system-by-jane-w-s-liu_5481.html) if the server size if the server is a deferrable server? Is this size a function of the period of the server? If not, why not? If yes, what is the best choice of server size?

b) What is maximum server size if the server is a total bandwidth server?

Sol: **Someone pointed out that problem 4 has a period of 6, and problem 7.5 says to repeat the setup in the previous problem. Using ps = 6 greatly simplifies the problem and probably was what was intended by the author.**

**In part (c) when repeating part (b) with the total bandwidth server. The second aperiodic job that arrives at time t=5 has execution time of 2, so the server's deadline should be set to 5 + 2 / (29/70) ≈ 9.83. The deadline of the other active job in the system is 10, so the aperiodic job is still scheduled first at t = 5. (i.e., the graph is still correct except for the deadline).**

**2/7 + 3/10 + es/Ps (1 + (Ps - es)/Di) <= 1**

**2/7 + 3/10 + es/Ps (1 + (Ps - es)/7) <= 1**

**and 2/7 + 3/10 + es/Ps (1 + (Ps - es)/10) <= 1**

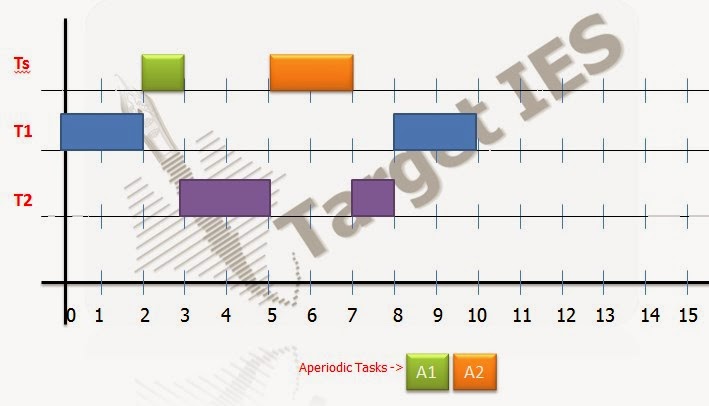
**by solving above two equations,**

**we can conclude. Us depends on the Ps.**

**For**[**maximum size**](http://targetiesnow.blogspot.in/2013/11/real-time-system-by-jane-w-s-liu_5481.html)**server, Ps -> 0. => Us = 0.41 (max).**

**However, we want Ps to be long enough to keep**[**context switch**](http://targetiesnow.blogspot.in/2013/11/real-time-system-by-jane-w-s-liu_5481.html)**time low. In this case, choose Ps close to interrelease time of the aperiodic jobs and es close to the execution time of the aperiodic jobs. In this case choose Ps = 3.2, which gives Us = 0.31 and es =1.0.**

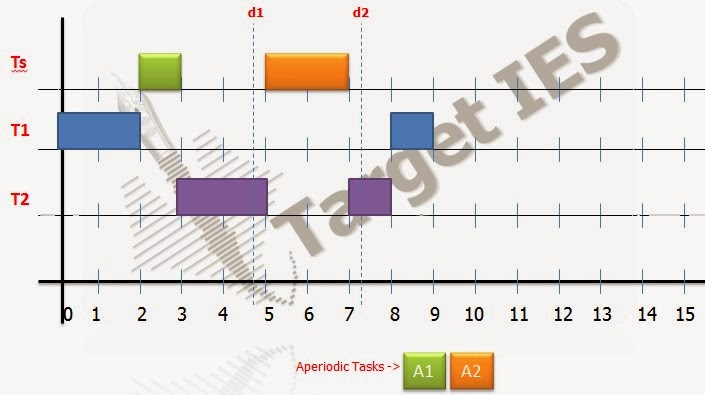
c) Find the response time of A1 and A2 in problem 7.4 for servers in part (a) and (b).  
  
Sol:   
 **1)**

[](http://2.bp.blogspot.com/-ILjQPxEoP9U/UnHtgisBKDI/AAAAAAAAAIg/frt8GxmrnG8/s1600/targeties13+(1).jpg)

**WA1 = 3 - 2 = 1**

**WA2 = 7 - 5 = 2**

**2)**

[](http://3.bp.blogspot.com/-iOlZXTsXvCE/UnHtm7TkOEI/AAAAAAAAAIo/h0N6IdZOu40/s1600/targeties13+(2).jpg)

**at t = 2,**  
**d1 = 2 + 1/(29/70) = 2 + 70/29 = 4.41**  
 **at t = 5,**  
**d2 = 5 + 1/(29/70) = 5 + 70/29 = 7.41**

**WA1 = 3 - 2 = 1**

**WA2 = 7 - 5 = 2**

33.

 Davis *et a*l., suggested a dual priority scheme for scheduling aperiodic jobs in the midst of periodic tasks. According to dual priority scheme, the system keeps three bands of priority, each containing one or more [priority levels](http://targetiesnow.blogspot.in/2013/11/real-time-system-by-jane-w-s-liu.html). The highest band contains real time priorities: they are for hard real time tasks. Real time priorities are assigned to hard real time tasks according to some fixed priority scheme. The middle priority band is for aperiodic jobs. The lowest priority band is also hard real time tasks. Specifically, when jobs Ji,k in a periodic task Ti = (pi, ei, Di) is released , it has a priority in the lowest priority band until [its priority](http://targetiesnow.blogspot.in/2013/11/real-time-system-by-jane-w-s-liu.html) promotion time. At its priority promotion time, its priority is raised to its real time priority. Let Wi denote the maximum response time of [all jobs](javascript:void(0);) in Ti when they execute at the real time priority of the task. The priority promotion time of each job is Yi = Di - Wi from its release time. Since Wi can be computed off line or at admission control time, the release promotion time Yi for jobs in each tasks Ti needs to be computed only once. By delaying as much as possible the scheduling of every hard real time jobs at its real time priority, the scheduler automatically creates slacks for aperiodic jobs.

a) Give an intuitive argument to support the claim that this scheme will not cause any periodic job to miss its deadline if the system of periodic tasks is schedulable.

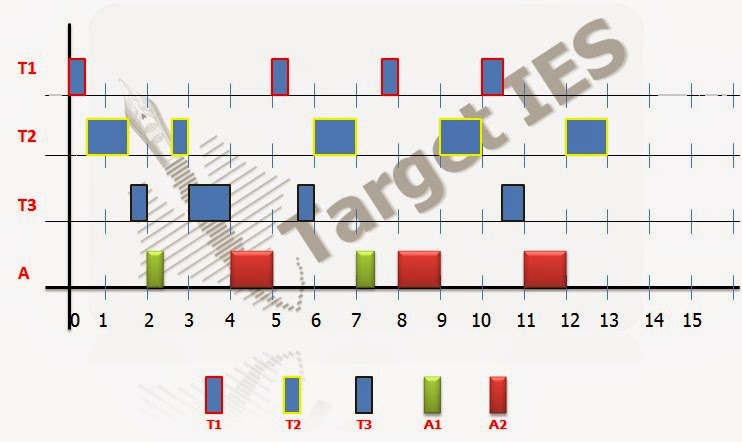
Sol:  **In the worst case, any periodic job in in Ti will take at most Wi time to execute leaving Yi = Di - Wi of slack. If the job does not finish in the first Yi time units, it still has Wi left before its deadline. The job will still finish before its deadline because Wi is the longest time it can possibly take.**

b) A system contains three periodic task: They are  (2.5,0.5), (3,1) and (5,0.5). Compute the priority promotion times for jobs in each of the tasks if the tasks are scheduled rate-monotonically.

Sol:   **Worst case Wi when**[**all jobs**](javascript:void(0);)**released at same time.**  
 **W1 = 0.5             Y1 = D1 - W1 = 2.5 - 0.5 = 2**  
 **w2(t) = 1 + [t/2.5].0.5 = t**  
**1 + 0.5 = 1.5 = t**  
**W2 = 1.5             Y2 = D2 - W2 = 3 - 1.5 = 1.5**  
 **w3(t) = 0.5 + [t/2.5].0.5 + [t/3] = t**  
**1 + 0.5 + 0.5 = 2 = t**  
**W3 = 2                Y3 = D3 - W3 = 5 - 2 = 3**

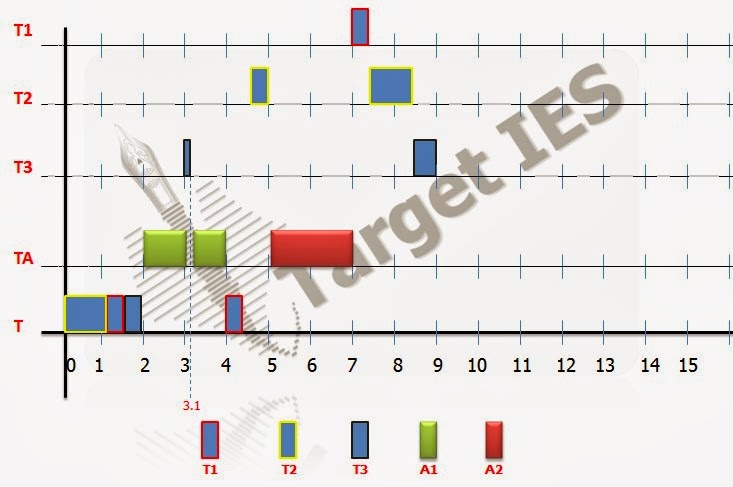
c) Suppose that there are two aperiodic jobs. One arrives at 1.9, and the other arrives at 4.8. Their execution times are 2. Compute the response times of these jobs in a dual priority system in which there is only one priority level in the middle priority band one priority level in the lowest priority band. How much [improvement](javascript:void(0);) is gained over simply scheduling the aperiodic jobs in the background of rate monotonically scheduled periodic tasks?

Sol:

[](http://2.bp.blogspot.com/-BafwbpF6AC0/UnItQseH7NI/AAAAAAAAAJ4/QJ4nxw0KtxQ/s1600/targeties14+(1).jpg)

**WA1 = 7.5 - 1.9 = 5.6**

**WA2 = 12 - 4.8 = 7.2**

[](http://4.bp.blogspot.com/-T3FyIBzrWRM/UnItxa5ch9I/AAAAAAAAAKA/hONFdOwBsUo/s1600/targeties14+(2).jpg)

**WA1 = 4 - 1.9 = 2.1**

**WA2 = 7 - 4.8 = 2.2**

d) Can the dual priority scheme be modified and used in a system where periodic tasks are scheduled according to the EDF algorithm? If no, briefly explain why; if yes, briefly describe the necessary modification.

Sol:  **To make the dual priority scheme with EDF, we need to complete Wi, which is mere difficult because tasks priorities can change. One approach would be to calculate Wi by adding the time demanded by tasks with shorter relative deadline (which can be computed offline) to the execution time Ti and the amount of time needed to complete jobs with relative deadlines longer than Di that have been released and have absolute deadlines before the most recently released job in Ti.**

34.

 Suppose that the intervals between arrivals of sporadic jobs are known to be in the [range](javascript:void(0);) (a,b). The execution time of each sporadic job is at most e (<= a) units.  
Suppose relative deadlines of sporadic jobs are equal to a. You are asked to design a bandwidth preserving server that will be scheduled rate monotonically with other periodic tasks. Sporadic jobs waiting to be completed are executed on the first in first out basis in the time intervals where the periodic server is scheduled. Choose the period and utilization of this server so that all sporadic jobs will be completed by their deadlines and the utilization of the sporadic server is as small as possible.  
  
Sol: **For simple sporadic server,**  
**[a/Ps].es  >= a**  
 **as we know,**  
**a/Ps >= [a/Ps]**  
**=>**  
**a/Ps.es >= a**  
**=>                   Us  >= a**  
 **This makes sense because of jobs arrive every "a" time units and take "a" time units to complete. They will need all available processor power.**  

:About the Sporadic Server:

• DS may delay lower-[priority tasks](http://targetiesnow.blogspot.in/2013/11/real-time-system-by-jane-w-s-liu_9.html).

• Sporadic Servers (SS) rules ensure that each sporadic server (ps, es) never demands more processor time than the periodic task (ps, es).

Sporadic Server in Fixed-Priority Systems: Notations

• T: system of n independent, preemptable periodic tasks.

• TH: subset of periodic tasks with higher priorities than the server priority.

• T/ TH are either busy or idle.

• Server busy interval: [an aperiodic job arrives at an empty queue, the queue becomes empty again].

• tr: the latest (actual) replenishment time.

• tf: the first instant after tr at witch server begins to execute.

• te: the latest effective replenishment time.

• At any time t:

–BEGIN: beginning instant of the earliest busy interval among the latest contiguous equence of busy intervals of TH that started before t.

–END: end of the latest busy interval if this interval ends before t, infinity if the interval ends after t.

 Simple Sporadic Server

• Consumption Rules: At any t > tr, budget is consumed at the rate of 1 per unit time until budget is exhausted when

– C1: the server is executing OR

– C2: the server has executed since tr and END < t.

• Replenishment Rules:

– R1: Initially when system begins execution and each time when budget is replenished, budget = es and tr = current time.

– [R2](http://targetiesnow.blogspot.in/2013/11/real-time-system-by-jane-w-s-liu_9.html): At time tf,

if END = tf then te = max(tr, BEGIN),

 if END < tf then te = tf. Next replenishment time is te + ps.

– [R3](http://targetiesnow.blogspot.in/2013/11/real-time-system-by-jane-w-s-liu_9.html): a) If te + ps is earlier than tf, budget is replenished as soon as it is exhausted.

        b) If T becomes idle before te + ps and becomes busy again at tb, budget is replenished at min(te + ps, tb).

35.

 A system contains five jobs. There are three resources X, Y and Z. The resources required of the jobs are listed below.

J1: [X;2]

 J2 : NONE

 J3 : [Y;1]

         J4 : [X;3 [Z;1]]

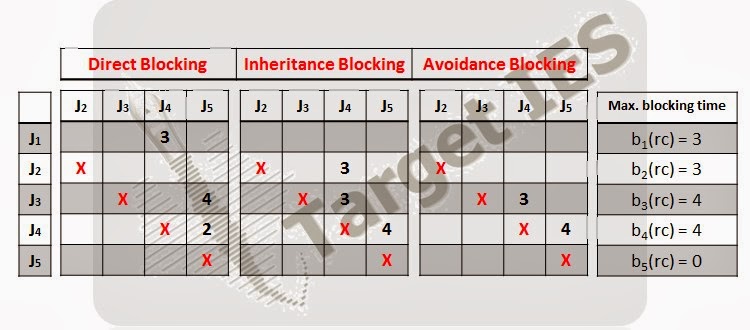
         J5 : [Y;4 [Z;2]]

The priority Ji is higher than the priority of Jj for i < j. What are the maximum blocking times of the jobs under the nonpreemptable critical section protocol and under the priority ceiling protocol?

Sol:                 **Nonpreemptive critical section**

**b1 = maxi > 1(ci) = 4  
                                 b2 = maxi > 2(ci) = 4  
                                 b3 = maxi > 3(ci) = 4  
                                 b4 = maxi > 4(ci) = 4  
                                 b5 = maxi > 5(ci) = 0**

**Priority Ceiling**

[](http://3.bp.blogspot.com/-pURdlaMRw2s/UnLEz6GH02I/AAAAAAAAAKQ/xWa33Tz-rlk/s1600/targeties15.jpg)

### Priority Inheritance Protocol

 Works with any priority-driven scheduling algorithm

 Uncontrolled priority inversion cannot occur

 Protocol does not avoid deadlock

 External mechanisms needed to avoid deadlock

#### Assumption:

 All resources have only one unit

 Definitions:

 The priority of a job according to the scheduling algorithm is its assigned priority

 At any time t, each ready job J is scheduled and executes at its current priority π (t), which may differ from its assigned priority and vary with time.

#### Priority Inheritance

The current priority πl(t) of a job Jl may be raised to the higher priority πh(t) of a job Jh

    When this happens, we say that the lower-priority job Jlinherits the priority of higher-priority job Jh, and that Jl executes at its inherited priority πh(t)

#### Scheduling Rule

Ready jobs are scheduled on the processor preemptively in a priority-driven manner according to their current priorities.

At its release time t, the current priority π(t) of every job J is equal to its assigned priority.

 The job remains at this priority except under the condition stated in the priority-inheritance rule

#### Allocation Rule

 When a job J requests a resource R at time t,

a) if R is free, R is allocated to J until J releases the resource, and

b) if R is not free, the request is denied and J is blocked

#### Priority-Inheritance Rule

When the requesting job J becomes blocked, the job Jl which blocks J inherits the current priority π(t) of J.

 The job Jl executes at its inherited priority π(t) until it releases R

 At that time, the priority of Jl returns to its priority πl(t’) at the time t’ when it acquired the resource R.

36.

 A system contains the following four periodic tasks. The tasks are scheduled by the rate monotonic algorithm and the priority ceiling protocol.  
  
                                T1 = (3,0.75)          b1 = 0.9  
                                T2 = (3.5,1.5)         b2 = 0.75  
                                T3 = (6,0.6)            b3 = 0.9  
                                T4 = (10,1)   
bi is the blocking time of Ti. Are the tasks schedulable ? Explain your answer.  
  
Sol: **Rate monotonic algorithm:**  
 **for U[T1] = 0.75/3 + 0.9/3 = 0.55 <1**  
 **U[T2] = 0.75/3 + 1.5/3 + 0.75/3.5 = 0.89 > URM(2) = 0.828**  
 **Hence, Condition fails.**  
 **So, by using TDA,**  
 **W1[3] = 0.75 + 0.9 = 1.65 < 3                            -Schedulable**  
 **W2[3] = 1.5 + 0.75 + [3/3]0.75 = 3 < 3.5            -Schedulable**  
 **W3[3.5] = 0.6 + 1 + 2x2.75 +1.5 +3 = 4.1 < 6      -Schedulable**  
 **W3[6] = 0.6 + 1 + 1.5 +3  = 6.1 > 6                     - Not Schedulable**

### The Priority Ceiling Protocol

Common real-time operating systems rely on priority-based, preemptive scheduling. Resource sharing in such systems potentially leads to priority inversion.  processes of high priority can be prevented from entering a critical section and be delayed by processes of lower priority. Since uncontrolled priority inversion can cause high-priority processes to

miss their deadlines, a real-time operating system must use resource-sharingg mechanisms that limit the effects of priority inversion. The priority ceiling protocol is one such mechanism. It ensures mutual exclusion and absence of deadlocks, and minimizes the length of priority inversion periods. This paper presents a formal speciﬁcation and analysis of the protocol using PVS and the rigorous proof of associated schedulability results.

The problem with the basic protocol is that job priorities are not taken into account when

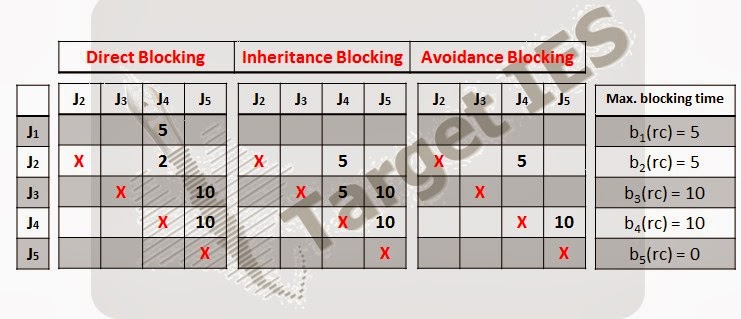
access to critical sections is granted. If a semaphore S is free, a job j executing P(S) obtains access to S, irrespective of any other jobs already in a critical section. Two jobs j1 and j2 can then lock two distinct semaphores S1 and S2. If the two semaphores are requested later on by a job k of higher priority than j1 and j2, two successive blocking periods will occur. To prevent such a situation, the priority ceiling protocol enforces stronger rules foraccessing a critical section. If two jobs j1 and j2 could potentially block a common job k via two semaphores S1 and S2, the protocol does not [grant](javascript:void(0);) S1 to j1 and S2 to j2 at the same time. For example, if j1 has lower priority than j2 and enters a critical section ﬁrst, then the request P(S2) by j2 will not be granted even though S2 may be free. To decide whether it is safe to allocate a semaphore S to a job j, the protocol must have some information about the other jobs that might request S in the future. For this purpose,each semaphore S is assigned a ﬁxed ceiling that is equal to the highest priority among the jobs that need access to S. If S is allocated to j then jobs of priority higher than j and lower than or equal to the ceiling of S might become blocked by j. The rule for entering critical sections is based on the priority of the requesting job and the ceiling of the semaphores already locked:

*A job j executing P(S) is granted access to S if the priority of j is strictly higher than the ceiling of any semaphore locked by a job other than j. Otherwise, j becomes blocked and S is not allocated to j.*

Apart from the new rule for accessing semaphores, the priority ceiling protocol works like the basic priority inheritance protocol. A job k is said to block j if k has lower priority than j and owns a semaphore of ceiling at least equal to the priority of j. Such a job k prevents j from entering a critical section. If j requests access to a semaphore, then j becomes blocked and k inherits j’s priority. An essential property of the protocol is that j cannot have more than one blocker k.

37.

 Consider a fixed priority system in which there are five tasks Ti, for i = 1, 2, 3, 4 and 5, with decreasing priorities. There are two resources X and Y. The critical sections of T1,T2, T4  and T5 are [Y;3], [X;4], [Y;5 [X;2]] and [X;10] respectively. (Note that T3 does not require any resource.) Find the blocking time bi(rc) of the tasks.  
  
Sol: **Priority Ceiling (with Blocking time)**

[](http://4.bp.blogspot.com/-uGQGWCb4ZSs/UnLL2Eiw74I/AAAAAAAAAKg/KjreGMhq6EM/s1600/targeties16.jpg)

### Access to Shared Resources

In Real Time System, some shared resources must be protected from concurrent accesses. The various tasks may demand for the same resources and this may leads to failure of the system. The Well-known solutions ensuring mutual exclusion exists, those are semaphores, mutex locks, etc. Before entering a critical section, the job executes a lock operation such as wait operation on a binary semaphore.

At the exit, an unlock operation is performed such as signal operation on a [binary](http://targetiesnow.blogspot.in/2013/11/real-time-system-by-jane-w-s-liu_2315.html) semaphore. Multiple-unit resources and multiple-unit resource requests can be dealt within the same framework. Nested critical sections are possible, but the resources must be accessed in a Last-In-First-Out order , such that the critical sections must be properly nested. The length of a critical section may include the length of other critical sections (the case of nested critical sections).

### Resource Conﬂicts and Blocking

When the scheduler/resource manager cannot [grant](javascript:void(0);) a lock request because of a resource conﬂict, the requesting job is blocked and it is removed from the ready queue. Once the resource becomes available again, it is unblocked and thus it is moved back to the ready queue. In priority-driven system, the blocking occurs only when a high-priority job requests a shared resource which is currently in use by a low-priority job.

As a result of blocking, the priority inversion phenomenon occurs: a high-priority real-time job is delayed by a low-priority job because of resource conﬂicts. In [general](javascript:void(0);), priority inversion is unavoidable if we want to enforce mutual exclusion. Timing anomalies may occur as a result of priority inversion: reducing the execution time of tasks may hurt feasibility. Further, uncontrolled priority inversion may lead to unbounded delays and deadline misses. Deadlock is another potential problem. Hence to avoid such situations, Resource access protocols are designed to limit the adverse effects of the priority inversion.

### Resource Access Protocols

We have seen the importance and the need of the [Resource access](http://targetiesnow.blogspot.in/2013/11/real-time-system-by-jane-w-s-liu_2315.html) protocols, these are the set of rules that govern:

•  When and under what conditions each request for a resource is granted

•  How jobs requiring resources are scheduled

The main objective is to avoid unbounded priority inversion and the secondary objective is to reduce the blocking times as much as possible

38.

 A system contains the following five periodic tasks. The tasks are scheduled rate monotonically.

T1 = (6, 3, [X;2])

 T2 = (20, 5, [Y;2])

           T3 = (200, 5, [X;3 [Z;1]])

           T4 = (210, 6, [Z;5 [Y;4]])

[Compare](javascript:void(0);) the schedulability of the system when the priority ceiling protocol is used versus the NPCS protocol.

Sol:       **Nonpreemptive Critical Section:**

**b1 = 5**

**b2 = 5**

**b3 = 5**

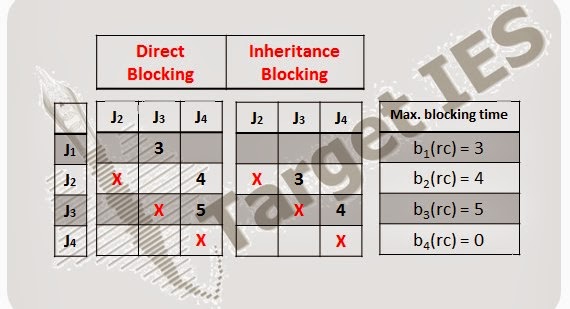
**b4 = 5**

**w1(t) = 3 + 5 = 8**

**W1 = 8 > p1 = 6**

**therefore, T1 is not schedulable**

**Priority Ceiling:**

**[](http://3.bp.blogspot.com/-hA-SNJuXbD8/UnLQA2l4r2I/AAAAAAAAAK0/S96SkhTuxWQ/s1600/targeties17.jpg)**

**w1(t) = 3 + 3 = 6  
                                              W1 = 6 ≤ p1 = 6**

**Continue using time demand analysis, to show that:**

**W2 = 18 ≤ p2 = 20  
                                                    W3 = 52 ≤ p3 = 200  
                                                    W4 = 53 ≤ p4 = 210  
             Therefore, all tasks are schedulable**

### Resource [Access Control](javascript:void(0);) Protocol

**Need a set of requirements (protocol) that determine the behavior of scheduler when jobs want access to a common resource. Resource contention between various jobs in the system may cause higher priority jobs to be blocked by a lower priority job. This is known as priority inversion. This causes timing anomalies and higher priority tasks could miss their deadline.  Deadlock maybe caused as result of non-preemptive jobs holding serial resources.  When two jobs require two resources, a possible deadlock situation will arise when each holds one of the two resources, each waiting for the other job to release the resource.  It is not possible to prevent priority inversion, so our goal is to reduce delays caused by priority inversion. Deadlock avoidance is another goal of some resource**[**access control**](javascript:void(0);)**protocols. Wait-for-graph is used to model resource contention. Every serial reusable resource is modeled. Every job which requires a resource is modeled by vertex with arrow pointing towards the resource. Every job holding a resource is represented by a vertex pointing away from the resource and towards the job. A cyclic path in a wait-for-graph indicates deadlock. A minimum of two system resources are required in a deadlock.**

### Non-Preemptive Critical Section protocol (NPCS)

**Any job requesting a resource is always given the resource. When a job has the resource, it executes at a priority higher than the priority of other jobs until it completes execution of its critical section. Because no job is ever preempted in a system using this protocol, deadlock can not occur in such systems.**

#### Characteristics of the NPCS protocol

**Simple to implement**

**Can be used in static priority as well as dynamic priority systems**

**Performs well when all critical sections are relatively short**

**Every job, regardless of its priority can be blocked by any other job**

39.

 Given a system consisting of the following tasks whose periods, execution times and resource requirements are given below.

T1 = (2, 0.4, [X, 3; 0.3])

                 T2 = (3, 0.75, [X, 1; 0.3][Y, 1; 0.4])

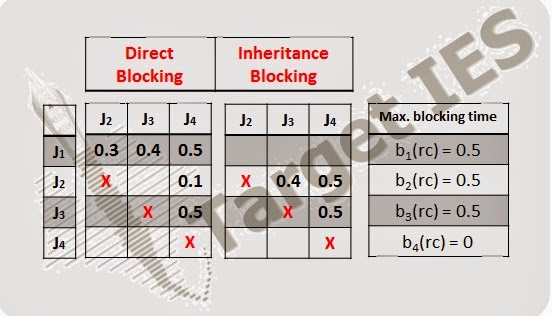
                               T3 = (6, 1.0, [Y, 1; 0.4][Z, 1; 0.5 [X, 1; 0.4]])

                               T4 = (8, 1.0, [X, 1; 0.5][Y, 2; 0.1] [Z, 1; 0.4])

There are 3 units of X, 2 units of Y and 1 unit of Z. The tasks are scheduled by EDF algorithm and the stack based protocol.

a) Find the preemption ceiling of each resource and the maximum blocking time for each task.

Sol: **Preemption Ceilings**

[](http://3.bp.blogspot.com/-5Z3ekq4cnoU/UnLTO9u5FHI/AAAAAAAAALA/ELtT7CHJVUw/s1600/targeties18.jpg)

b) Are the tasks schedulable according to the earliest deadline first algorithm? Why ?

Sol:

**Schedulability test**

**T1 : (e1 + b1) / p1 + Σi≠1(ei / pi) = (0.4 + 0.5)/2 + 0.75/3 + 1/6 + 1/8 ≈ 0.92  
         T2 : 0.4/2 + (0.75+0.5)/3 + 1/6 + 1/8 ≈ 0.91  
         T3 : 0.4/2 + 0.75/3 + (1+0.5)/6 + 1/8 ≈ 0.825  
         T4 : 0.4/2 + 0.75/3 + 1/6 + (1+0.5)/8 ≈ 0.80**

**Therefore, all tasks are schedulable by EDF**

### Stack Resource Policy(SRP)

#### Extend definition of priority ceiling:

In priority ceiling priorities are replaced by preemption levels. This allows EDF priorities to be handled without requiring to recompute ceilings at run time. Cceilings are defined for multiunit resources, subsuming both binary semaphores and read/write lock

#### Abstract ceilings

if J is currently executing or can preempt the currently executing job, and may request an allocation of R that would be blocked directly by the outstanding allocation of R, then [R] ≥ π (J)

#### Specific ceilings

[R]VR = max({0} ∪{π(J) | VR < µR (J)})

VR units of R available.

µR(J) is the maximum num ber of units of R that  job J may need to hold at any one time

#### Current ceiling

π = max({[Ri]| i = 1,…,m} ∪{π(Jc})

if there’re no jobs currently execute, π = 0 the SRP requires that a job execution request J be blocked from starting execution until π < π(J). Once J has started execution, all subsequent resource request by J are granted immediately doesn’t restrict the resource acquiring order, and allocate only when requests.

release resources when they are not need. JH is free to preempt until J actually requests enough of R to block JH (without being blocked).

#### Blocking properties of the SRP

If no job J is permitted to start until π’ < π(J) =>

(a) No job can be blocked after it starts

(b) There can be no transitive blocking or deadlock

(c) If the oldest highest-priority job is blocked, it will become unblocked no later than the first instant that the currently executing job isn’t holding any nonpreemptable resources.