Real-Time Systems: Assignment 1
Orion Lowy (1449248)

(1) EDF and Queue/Stack disciplines

"Queue discipline" is another term for FIFO, while "Stack discipline" refers to LIFO. An EDF scheduler does not always obey the FIFO (queue) discipline, because tasks with shorter deadlines that are added later are allowed to preempt (or at least cut in front of) tasks with longer deadlines that are added earlier. EDF schedulers are also not required to obey the LIFO (stack) discipline, because a task that is added later but has a long deadline will have to wait for all previously added tasks with shorter deadlines to finish. EDF schedules are not directly related to arrival order, but instead use deadlines as the parameter for deciding execution order.

Example of how EDF does not obey FIFO (queue) discipline:

Task	Arrival	Exec Time	Deadline
J_1	0	5	10
J_2	0	5	15
J_3	2	5	10

The order of these tasks will be J_1, J_3, J_2. Even though J_2 arrived before J_3, it will not be scheduled until after J_3 because J_3's deadline is sooner than J_2's.

Example of how EDF does not obey LIFO (stack) discipline:

Task	Arrival	Exec Time	Deadline
J_1	0	5	11
J_2	0	5	10
J_3	2	5	15

The order of these tasks will be J_2, J_1, J_3. Even though J_3 arrived last, it will not be chosen as the next task to run after J_2 is finished because J_1's deadline is sooner than J_3's.

(2) Subtask scheduling

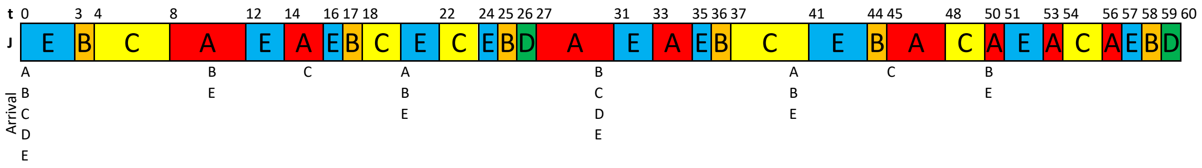
Each subtask may have to wait until there is just enough time before the deadline to run itself and all the other remaining subtasks. This moment would correspond to laxity = 0 for the subtask relative to the set of subtasks. A formula to represent the latest allowable start time could be:

$$D - \sum_{k=i}^{n} C_k$$

for each subtask J_i.

(3) LLF scheduling

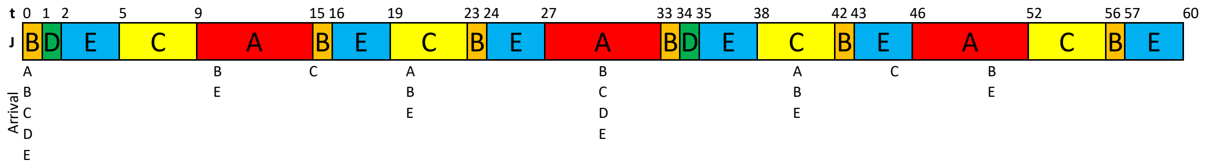
Yes, the task set is schedulable using LLF. The diagram below shows a schedule.



Real-Time Systems: Assignment 1
Orion Lowy (1449248)

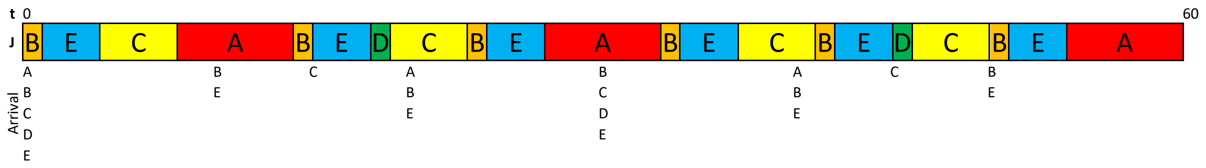
(4) Scheduling a given task set with multiple schedulers

FCFS schedule:



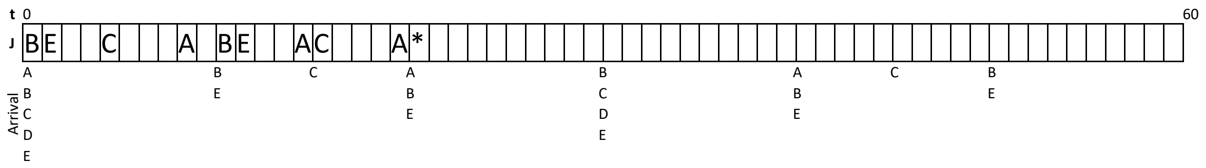
*To break ties, a SJF policy was used.

EDF schedule:



*To break ties, a SJF policy was used. If deadlines for newly arrived tasks tied with the running task, the running task was not preempted.

RM schedule:



This task set is not RM-schedulable. Task A misses its deadline at time = 20. Schedulability test 3 shows this because:

- $c_B + c_E + c_C + c_A + c_D = 1 + 3 + 4 + 6 + 1 = 15 > 10 = p_B = p_E$.
- $2c_B + 2c_E + c_C + c_A + c_D = 2 + 6 + 4 + 6 + 1 = 19 > 15 = p_C$.
- $2c_B + 2c_E + 2c_C + c_A + c_D = 2 + 6 + 8 + 6 + 1 = 23 > 20 = p_A$.
- $3c_B + 3c_E + 2c_C + 2c_A + c_D = 3 + 9 + 8 + 12 + 1 = 33 > 30 = p_D$.

(5) Scheduling with precedence constraints

Revise the deadlines to integrate information about precedence.

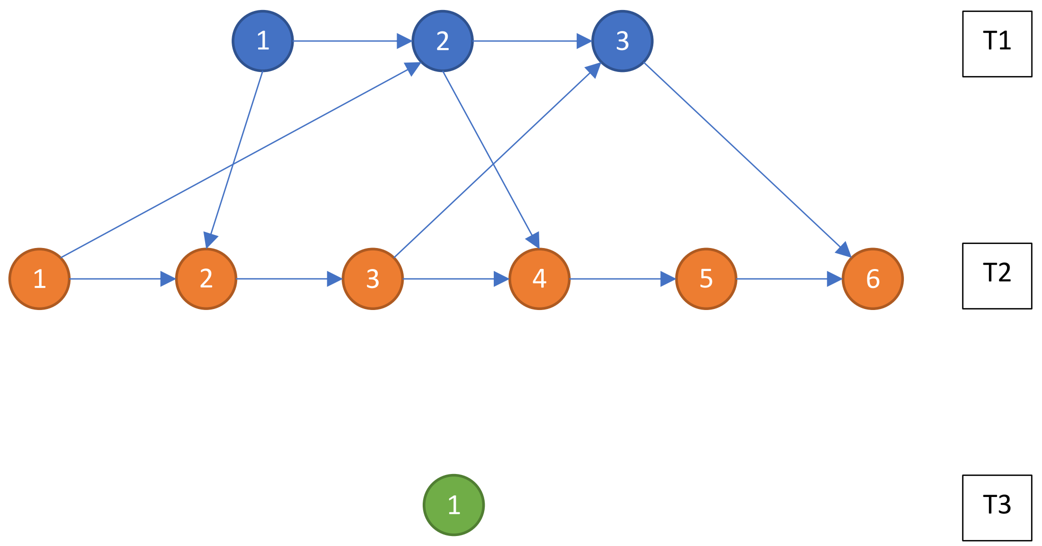
- Map out scheduling blocks over a timespan = LCM of periods of all tasks.

  LCM(42, 42, 14) = 42

- Assign an absolute deadline to each block.

  $d_{1,1} = 42$; $d_{1,2} = 42$; $d_{1,3} = 42$; $d_{2,1} = 12$; $d_{2,2} = 12$; $d_{2,3} = 26$; $d_{2,4} = 26$; $d_{2,5} = 40$; $d_{2,6} = 40$; $d_{3,1} = 42$

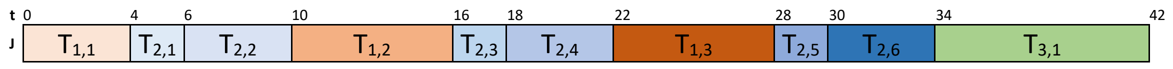
- Topologically order the blocks and then reverse it.
  - Draw a precedence graph.



- A forward topological could be: ($T_{3,1}$ $T_{2,1}$ $T_{1,1}$ $T_{2,2}$ $T_{1,2}$ $T_{2,3}$ $T_{1,3}$ $T_{2,4}$ $T_{2,5}$ $T_{2,6}$)
- Reversing that = ($T_{2,6}$ $T_{2,5}$ $T_{2,4}$ $T_{1,3}$ $T_{2,3}$ $T_{1,2}$ $T_{2,2}$ $T_{1,1}$ $T_{2,1}$ $T_{3,1}$)

- Work through the reverse topological order and adjust deadlines for blocks with outgoing edges in the precedence graph.
  - $T_{2,6}$ has no outgoing edges. $d'_{2,6} = d_{2,6} =$ **40**.
  - $T_{2,5}$ has an edge to $T_{2,6}$. $d'_{2,5} = \min(d_{2,5}, d'_{2,6} - c_{2,6}) = \min(40, 40 - 4) = \min(40, 36) =$ **36**.
  - $T_{2,4}$ has an edge to $T_{2,5}$. $d'_{2,4} = \min(d_{2,4}, d'_{2,5} - c_{2,5}) = \min(26, 36 - 2) = \min(26, 34) =$ **26**.
  - $T_{1,3}$ has an edge to $T_{2,6}$. $d'_{1,3} = \min(d_{1,3}, d'_{2,6} - c_{2,6}) = \min(42, 36 - 4) = \min(42, 32) =$ **32**.
  - $T_{2,3}$ has edges to $T_{1,3}$ and $T_{2,4}$. $d'_{2,3} = \min(d_{2,3}, d'_{2,4} - c_{2,4}, d'_{1,3} - c_{1,3})$
    $= \min(26, 26 - 4, 42 - 6) = \min(26, 22, 36) =$ **22**.
  - $T_{1,2}$ has edges to $T_{1,3}$ and $T_{2,4}$. $d'_{1,2} = \min(d_{1,2}, d'_{1,3} - c_{1,3}, d'_{2,4} - c_{2,4})$
    $= \min(42, 32 - 6, 26 - 4) = \min(42, 26, 22) =$ **22**.
  - $T_{2,2}$ has an edge to $T_{2,3}$. $d'_{2,2} = \min(d_{2,2}, d'_{2,3} - c_{2,3}) = \min(12, 26 - 2) = \min(12, 24) =$ **12**.
  - $T_{1,1}$ has edges to $T_{1,2}$ and $T_{2,2}$. $d'_{1,1} = \min(d_{1,1}, d'_{1,2} - c_{1,2}, d'_{2,2} - c_{2,2})$
    $= \min(42, 22 - 6, 12 - 4) = \min(42, 16, 8) =$ **8**.
  - $T_{2,1}$ has edges to $T_{1,2}$ and $T_{2,2}$. $d'_{2,1} = \min(d_{2,1}, d'_{1,2} - c_{1,2}, d'_{2,2} - c_{2,2})$
    $= \min(12, 22 - 6, 12 - 4) = \min(12, 16, 8) =$ **8**.
  - $T_{3,1}$ has no outgoing edges. $d'_{3,1} = d_{3,1} =$ **42**.

- We can schedule the adjusted task blocks with EDF. Original arrival times are respected.



(6) Task set density

The density of the task system is defined as the sum of the densities of all the tasks in the system. If there is only one task, the system is trivially schedulable, since the single task may start as soon as it arrives, and it is an absurdity to have a task whose deadline cannot be met even if execution is immediate and uninterrupted. Say we have a system S of n – 1 many tasks with density D <= 1 which is schedulable. Consider a task J_n with density D_n such that D + D_n is still <= 1. This implies D <= 1 - D_n. This means that there is an average of >= c_n units of time not used by tasks in S over each window of width d_n within the hyperperiod of S. Since the tasks are independent and preemptable, move their execution blocks so that at least c_n units of idle time are placed between each arrival time of an instance of J_n and the instance's deadline, even if the idle time is not contiguous. Then this idle time can be filled by J_n, and it will meet its deadline. So, the new system S' = S + J_n is still schedulable.

(7) RM and EDF schedulability tests

Since all 3 systems have 3 tasks, the expression from Schedulability Test 2 = 3 * ($2^{1/3}$ – 1) = 0.78

System (a):
U = (4 / 10) + (4 / 16) + (4 / 20) = 0.4 + 0.25 + 0.2 = 0.85
Yes, system (a) is EDF schedulable by Schedulability Test 4.

U > 0.78, so we need to try Schedulability Test 3 to check if the system is RM schedulable.

- J_1 is RM schedulable because c_1 = 4 <= 10 = p_1.
- J_2 is RM schedulable because c_1 + c_2 = 8 <= 10 = p_1.
- J_3 is RM schedulable because 2*c_1 + c_2 + c_3 = 16 <= 16 = p_2.

So yes, system (a) is RM schedulable.

System (b):
U = (2 / 5) + (3 / 10) + (3 / 12) = 0.4 + 0.3 + 0.25 = 0.95
Yes, system (b) is EDF schedulable by Schedulability Test 4.

- J_1 is RM schedulable because c_1 = 2 <= 5 = p_1.
- J_2 is RM schedulable because c_1 + c_2 = 5 <= 5 = p_1.
- J_3 is RM schedulable because 2*c_1 + c_2 + c_3 = 10 <= 10 = p_2.

So yes, system (b) is RM schedulable.

System (c):
U = (3 / 7) + (5 / 9) + (3 / 15) = 0.43 + 0.45 + 0.2 = 1.08
No, system (c) is not EDF schedulable, by Schedulability Test 4.

System (c) is not RM schedulable either, since EDF is an optimal scheduler.

(8) Constructing task set

This problem is equivalent to creating a task set that is RM schedulable, since EDF and LLF are optimal and can schedule any task set that is schedulable by another method.

J_1: c = 2; p = 4
J_2: c = 2; p = 6
J_3: c = 2; p = 12

$3*c_1 + 2*c_2 + c_3 = 6 + 4 + 2 = 12 <= 12 = p_3$ so the task set is RM schedulable.

(9) Non-simple schedulable utilization (RM schedulability test 3)

The task set shown in problem (8) does not pass schedulability test 2 because its utilization is (2/4) + (2/6) + (2/12) = 0.50 + 0.33 + 0.17 = 1.00, which is greater than the utilization for a set of 3 tasks = 0.78 (already calculated in problem (7)). But it is RM schedulable, as shown in problem (8).

(10) Multiprocessor scheduling

The problem of finding a set of n^4 tasks grows quickly as n increases. Any set of this many tasks needs to have very widely spaced periods in order to be schedulable. According to the utilization formula in schedulability test 10, the period of each task J_i should be at least $c_i * n^3$ in order for the sum of all ratios c_i/p_i to be <= n. For example, with n = only 3, the minimum period of any task with c = 1 would have to be $3^3 = 27$, while with n = 4, this minimum would be $4^3 = 64$, etc. The minimum period for tasks with longer computation times would be even greater. Any tasks with periods narrower than this minimum for a given n would make the task set non-schedulable. So it is possible, but not very likely when n is not trivial.

(11) RM and EDF equivalence

The RM and EDF algorithms are equivalent when the tasks with the narrowest periods also have the earliest deadlines every cycle. This can occur when all tasks have deadlines = periods and each period is an integer multiple of all lesser periods of tasks in the set. However, since the EDF algorithm breaks ties for earliest deadline using an arbitrary strategy, a task with a wider period could be chosen in favor of the task with the narrowest period to break the tie during the last cycle before a new instance of the task with the wider period arrives, or the latest instance of the task with a narrow period could fail to preempt the running instance of a task with a wider period. So the tie-breaking strategy of the EDF algorithm would also have to be based on least period first in order for the algorithms to be equivalent all the time.

(12) Is EDF ever not optimal?

EDF is always optimal on a uniprocessor system, but there are cases where EDF is not optimal on a multiprocessor system. This frequently happens when there is a task with a late deadline but also a very high computation time, resulting in a low value of slack/laxity.

For example, say a system has 3 processors. There are 4 tasks ready, including $J_1 = \{d_1 = 8, c_1 = 8\}$, $J_2 = \{4, 1\}$, $J_3 = \{4, 3\}$ and $J_4 = \{5, 4\}$. If we use EDF to schedule, then the tasks J_2 through J_4 will be scheduled first across the 3 processors. Task J_2 will finish first, at time = 1, leaving room on processor 1 for task J_1 to begin executing. But J_1 will miss its deadline if it waits for even one time unit before it executes, because it has 8 computation units to complete before time = 8. This means that the task set is not schedulable with EDF on this system.

In contrast, using LLF will recognize that task J_1 has laxity = 0, so that task will be scheduled at time = 0 along with tasks J_3 (L = 1) and J_4 (L = 1). When J_3 completes early at time = 3 on processor 2, there is just enough time to execute task J_2 before its deadline. Meanwhile, the other processors complete their tasks right before their deadlines. So the task set is schedulable with LLF. This means that EDF is not optimal, since another scheduler could schedule a task set that EDF failed to schedule.

(13) RM schedulability test 3

Let's test the set of tasks using schedulability test 2 first. The desired utilization is $n * (2^{1/n} - 1) = 6 * (2^{1/6} - 1) = 0.735$; the utilization of the system is $(5/50) + (11/70) + (14/100) + (15/120) + (26/140) + (60/300) = 0.908$ so test 2 does not work. We need to try test 3 to check if the set is RM schedulable.

$S_1 = T_1$ is trivially schedulable.

$S_2 = S_1 + T_2$ is schedulable because $c_1 + c_2 = 5 + 11 = 16 <= 50 = p_1$.

$S_3 = S_2 + T_3$ is schedulable because $c_1 + c_2 + c_3 = 5 + 11 + 14 = 30 <= 50 = p_1$.

$S_4 = S_3 + T_4$ is schedulable because $c_1 + c_2 + c_3 + c_4 = 5 + 11 + 14 + 15 = 45 <= 50 = p_1$.

$S_5 = S_4 + T_5$ is schedulable because $2c_1 + 2c_2 + c_3 + c_4 + c_5 = 10 + 22 + 14 + 15 + 26 = 87 <= 100 = p_3$.

$S_6 = S_5 + T_6$ is **not** schedulable because:

- $c_1 + c_2 + c_3 + c_4 + c_5 + c_6 = 5 + 11 + 14 + 15 + 26 + 60 = 131 > 50 = p_1$.
- $2c_1 + c_2 + c_3 + c_4 + c_5 + c_6 = 10 + 11 + 14 + 15 + 26 + 60 = 136 > 70 = p_2$.
- $2c_1 + 2c_2 + c_3 + c_4 + c_5 + c_6 = 10 + 22 + 14 + 15 + 26 + 60 = 147 > 100 = p_3$.
- $3c_1 + 2c_2 + 2c_3 + c_4 + c_5 + c_6 = 15 + 22 + 28 + 15 + 26 + 60 = 166 > 120 = p_4$.
- $3c_1 + 2c_2 + 2c_3 + 2c_4 + 2c_5 + c_6 = 15 + 22 + 28 + 30 + 52 + 60 = 207 > 140 = p_5$.
- $6c_1 + 5c_2 + 3c_3 + 3c_4 + 3c_5 + c_6 = 30 + 55 + 42 + 45 + 78 + 60 = 310 > 300 = p_6$.

So no, the system with all 6 tasks is not RM schedulable.