Chapitre 2, Section 3: Ordonnancement dynamique

- **Jusqu'à maintenant, on a vu que des algorithmes statiques. Pour terminer on va regarder un exemple d'algorithme dynamique.**
- **On va regarder la politique EDF (Earliest-Deadline-First) et on va voir en exercices l'algorithme du LCT (*Least Compute Time)* et du LST (*least slack time*).**
- **Il s'agit d'une politique mise à jour du *deadline* après chaque tick d'horloge. La plus grande (petite) priorité est assignée à la tâche qui a le plus petit (grand) *deadline* non nul. La tâche la plus prioritaire est ensuite choisi pour être exécuter.**

Ordonnacement dynamique

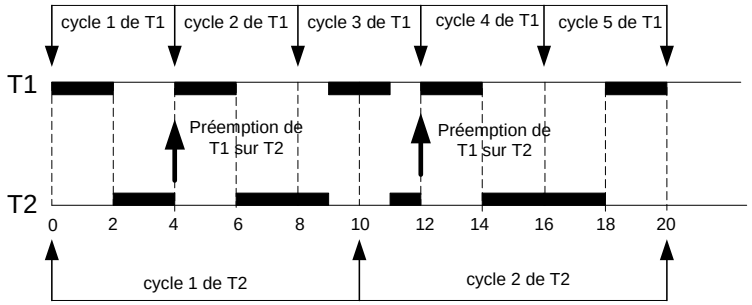
- **Exemple 1. Soit les tâches suivantes:**

Process	Period, *T*	Deadline, *D*	Computation Time, *C*
T1	4	4	2
T2	10	10	5

- **2/4 + 5/10 = 1, donc on ne peut rien dire avec le test d'ordonnancement (Liu and Layland).**

Ordonnacement dynamique

- **La figure suivante illustre la trace d'ordonnancement selon l'algo du EDF (ce qui prouve qu'un ordonnancement existe):**



Chap 2, Section 3.2, page 38

Ordonnacement dynamique

- **Sur la figure précédente, on remarque:**
  - **T1 démarre en premier car il possède le plus petit *deadline* (4 vs 10);**
  - **À la période 1, le *deadline* de T1 (3) est toujours plus petit que T2 (9), donc T1 poursuit son exécution;**
  - **À la période 2, T1 a terminé son cycle 1 et son *deadline* devient 0. Par contre, T2 qui n'a pas encore démarrer a maintenant un *deadline* de 8. T2 a donc le plus petit *deadline* non nul et il démarre;**
  - **À la période 3, le deadline de T2 (7) est toujours plus petit car le *deadline* de T1 est encore 0, donc T2 poursuit son exécution;**
  - **À la période 4, T2 a maintenant un *deadline* de 6 et T1 retrouve son deadline de 4 (début de son 2e cycle). Donc T1 a une préemption sur T2;**

Chap 2, Section 3.2, page 39

Ordonnacement dynamique

- **À la période 5, le deadline de T1 (3) est toujours plus petit que T2 (5), donc T1 poursuit son exécution;**
- **À la période 6, T1 a terminé son cycle 2 et son *deadline* devient 0. Par contre, T2 a maintenant un deadline de 4. T2 a donc le plus petit *deadline* non nul et il redémarre son exécution;**
- **À la période 7, T2 a maintenant un *deadline* de 3 et T1 a encore un deadline de 0, donc T2 poursuit son exécution;**
- **À la période 8, T2 a maintenant un *deadline* de 2 et T1 retrouve son deadline de 4 (début de son 3^{e} cycle). Cependant puisque 2 < 4, T2 poursuit son exécution.**
- **À la période 9, T2 a terminé son cycle 1 et donc son *deadline* devient 0. Par conséquent, T1 qui a un deadline de 3 démarre son cycle 3;**
- **Etc.**

Ordonnacement dynamique

- **On remarque aussi que que ce n'est qu'à la période 20 que les deux tâches terminent un cycle en même temps (ce qui n'est pas le cas à la période 10).**
- **On peut donc considérer que le même ordonnancement se répètera entre les périodes 20 et 40, 40 et 60, 60 et 80, etc.**
- **C'est en fait le plus petit commun multiple entre les deux deadlines qui déterminent se point de rencontre.**
- **Finalement on remarque que le CPU est utilisé à 100% du temps (l'exemple suivant montre qu'on peut avoir moins que 100%).**

Ordonnacement dynamique

- **Exemple 2. Soit les tâches suivantes:**

Process	Period, *T*	Computation Time, *C*
T1	3	1
T2	4	1
T3	5	2

- **1/3 + 1/4 + 1/5 = 0.983, donc on ne peut rien dire avec le test d'ordonnancement (Liu and Layland).**

Ordonnacement dynamique

- **Dans ce qui suit on imprime la trace d'exécution pour le petit petit commun multiple (60) et montre dans la colonne de droite que les *deadlines* sont toujours respectés. Un ordonnancement existe donc.**

Time	*Running process*	*Deadlines*
0	P1	
1	P2	
2	P3	P1
3	P3	P2
4	P1	P3
5	P2	P1
6	P1	
7	P3	P2

8	P3	P1
9	P1	P3
10	P2	
11	P3	P1, P2
12	P3	
13	P1	
14	P2	P1, P3
15	P1	P2
16	P2	
17	P3	P1
18	P3	
19	P1	P2, P3
20	P2	P1
21	P1	
22	P3	
23	P3	P1, P2
24	P1	P3
25	P2	
26	P3	P1
27	P3	P2
28	P1	
29	P2	P1, P3
30	P1	
31	P3	P2
32	P3	P1
33	P1	
34	P2	P3
35	P3	P1, P2
36	P1	
37	P2	
38	P3	P1

Chap 2, Section 3.2, page 44

39	P1	P2, P3
40	P2	
41	P3	P1
42	P1	
43	P3	P2
44	P3	P1, P3
45	P1	
46	P2	
47	P3	P1, P2
48	P3	
49	P1	P3
50	P2	P1
51	P1	P2
52	P3	
53	P3	P1
54	P2	P3
55	P1	P2
56	P2	P1
57	P3	
58	P3	
59	idle	P1, P2, P3

3X4X5=60,

Donc 59/60 comme taux d'occupation du CPU.

Chap 2, Section 3.2, page 45

En résumé

- **On a vu 2 algorithmes d'ordonnancement dont le rôle est d'assigner des priorités aux tâches (e.g. processus ou thread) selon le *deadline* (e.g. fixé par l'usager):**
  - ***Rate monotonic assignement* (D = T)**
  - ***Deadline monotonic assignement* (D < T)**
- **Ces algorithmes sont dits *statiques* en ce sens que l'ordonnancement est fixé au départ et reste inchangé au cours de l'exécution.**

En résumé

- **Il existe aussi des ordonnancements dynamiques. Par exemple:**
  - ***Earliest deadline***
  - ***Least slack time***

  Référence: Baker T.P. (1991). *Stack-based scheduling of real-time processes,* Real-Time Systems 3(1).
- Compléter les no. 12 et 15 des exercices pour le final (si le temps le permet faire aussi le no 2 du final aut. 2003).