Programmation Temps Réel

Ordonnancement de tâches périodiques

Yann Thoma

Reconfigurable and Embedded Digital Systems Institute
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

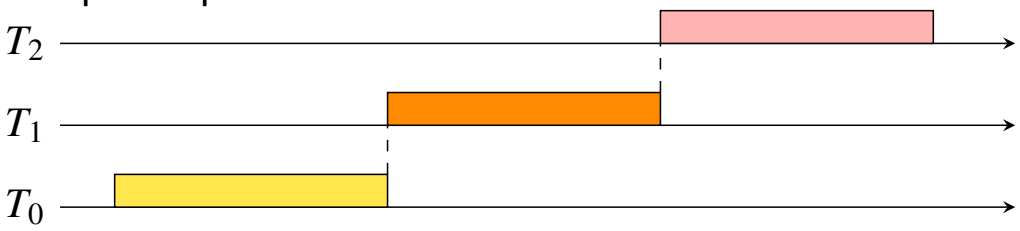
septembre 2017

Ordonnancement non temps réel

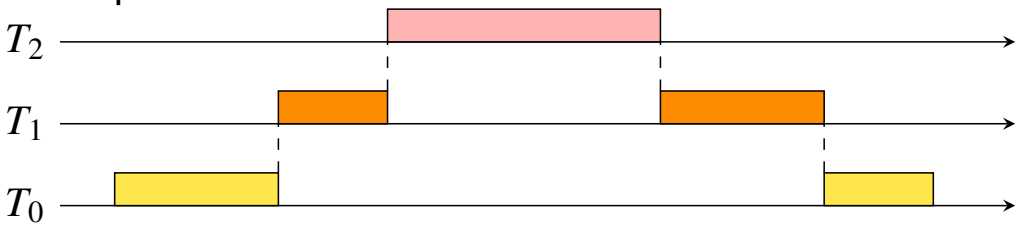
- L'ordonnancement correspond à l'action de choisir l'ordre d'exécution des tâches
- L'ordonnanceur en est responsable
- Un système d'exploitation standard possède un ordonnanceur

Taxonomie des ordonnancements (1)

- Monoprocesseur/multiprocesseur
- Non-préemptif



- Préemptif



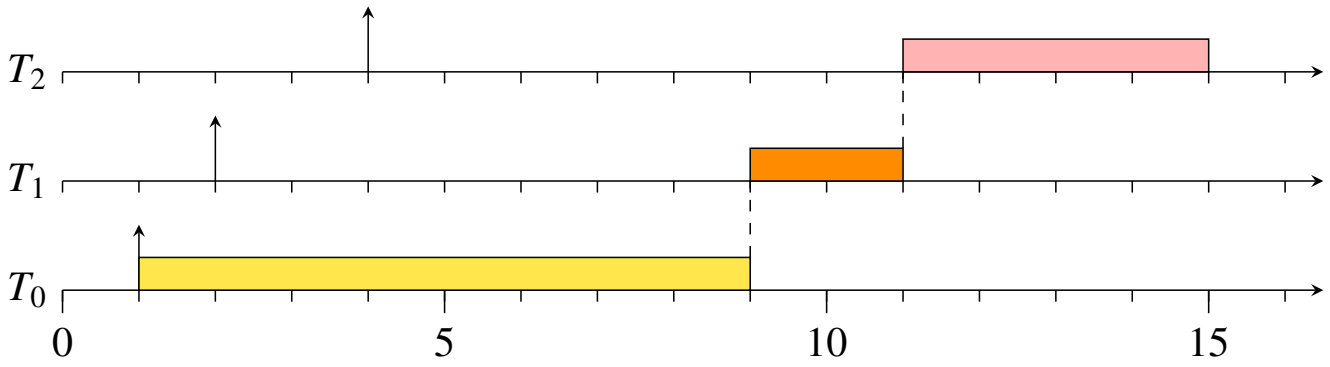
Taxonomie des ordonnancements (2)

- En-ligne/hors-ligne
  - Hors-ligne: réalisé avant le lancement du système
  - En-ligne: réalisé pendant le fonctionnement
- Statique/Dynamique
  - Statique: les propriétés des tâches ne changent pas
  - Dynamique: les propriétés des tâches peuvent changer (priorité)
- Optimal/Non optimal
  - Optimal: fourni un ordonnancement si un ordonnancement existe
  - Non optimal (best effort): essaie de se rapprocher de l'optimal

Algorithmes non préemptifs

- FIFO : Premier arrivé premier servi

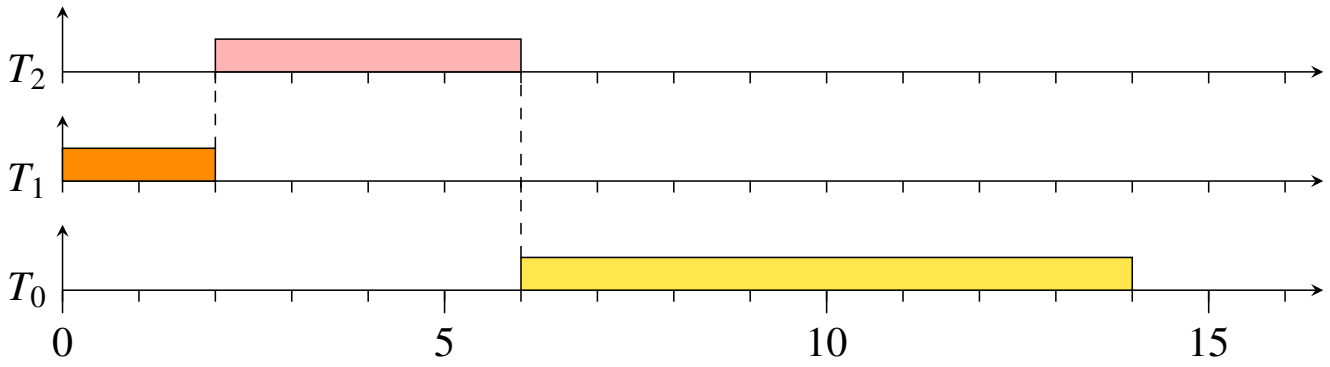
Tâche	Coût	Arrivée
T_0	8	1
T_1	2	2
T_2	4	4



Algorithmes non préemptifs

- Shortest job first

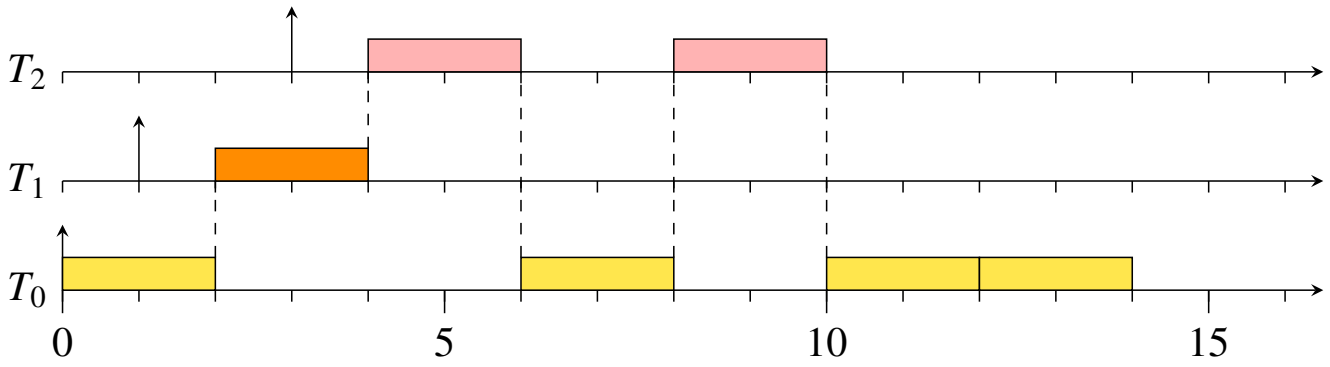
Tâche	Coût	Arrivée
T_0	8	0
T_1	2	0
T_2	4	0



Algorithmes préemptifs

- Round robin/tourniquet

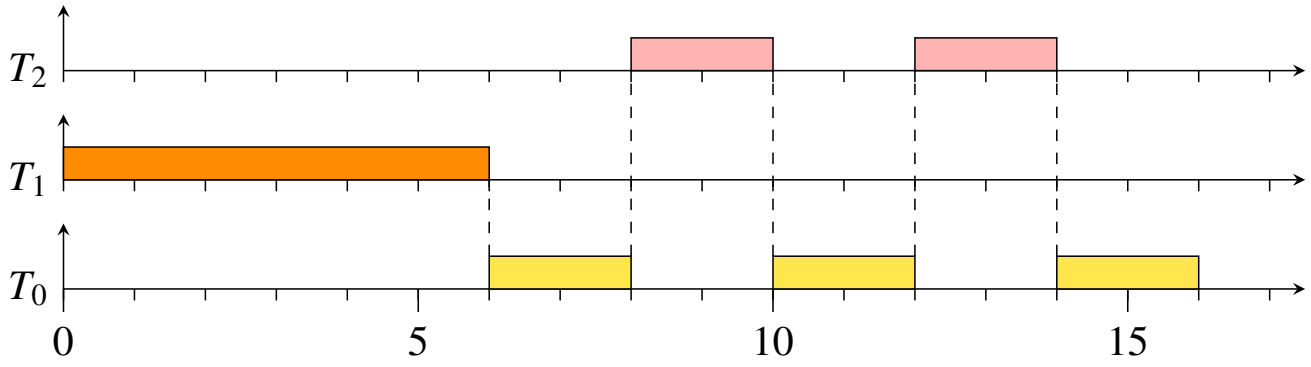
Tâche	Coût	Arrivée
T_0	8	0
T_1	2	1
T_2	4	3



Algorithmes préemptifs

- Priorité fixe

Tâche	Coût	Priorité	Arrivée
T_0	6	1	0
T_1	6	2	0
T_2	4	1	0



Ordonnancement temps réel

- Partant d'un ensemble de tâches Γ
- Les tâches ayant:
  - Un temps d'arrivée
  - Un temps d'exécution
  - *Une échéance*
- L'ordonnancement vise à trouver un moyen d'allouer le processeur aux différentes tâches de manière à *respecter les contraintes de temps*

Ordonnancement temps réel

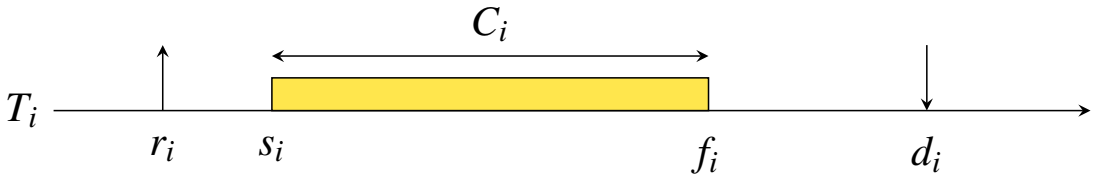
- Les algorithmes d'ordonnancement sont classés selon les critères suivants:
  - A priorité fixe/variable
  - Préemptif/non préemptif
  - Mono processeur/multi processeur (pas abordé dans ce cours)
- L'ensemble de tâches peut contenir des tâches:
  - Uniquement périodiques
  - Périodiques et apériodiques
  - Uniquement apériodiques
  - Dépendantes/indépendantes
  - Avec/sans ressources partagées

Tâches périodiques

Variable	Description
Γ	Un ensemble de tâches périodiques
τ_i	Une tâche périodique générique
$\tau_{i,j}$	La j$^{\text{ème}}$ instance de la tâche périodique τ_i
$r_{i,j}$	Le temps d'arrivée de la j$^{\text{ème}}$ instance de la tâche périodique τ_i
Φ_i	Le déphasage de la tâche τ_i; il s'agit du temps d'arrivée de $\tau_{i,0}$ ($\Phi_i = r_{i,0}$)
D_i	Echéance relative de la tâche τ_i
$d_{i,j}$	Echéance absolue de la j$^{\text{ème}}$ instance de la tâche τ_i ($d_{i,j} = \Phi_i + (j-1)P_i + D_i$)
$s_{i,j}$	Temps de début d'exécution de la j$^{\text{ème}}$ instance de la tâche τ_i
$f_{i,j}$	Temps de fin d'exécution de la j$^{\text{ème}}$ instance de la tâche τ_i

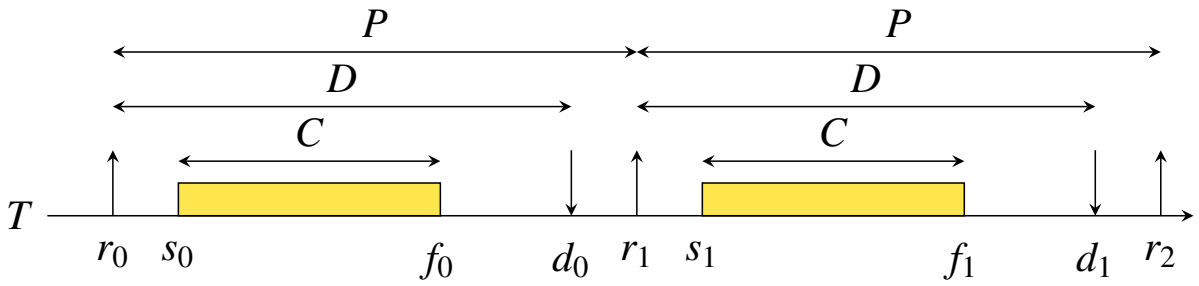
Paramètres d'une tâche

- r: la date de réveil de la tâche (ou date de demande d'activation)
- C: sa durée d'exécution, calculée en temps processeur. Il s'agit du pire temps d'exécution.
- d: son échéance, au-delà de laquelle le résultat est jugé comme étant non pertinent
- s: date de début d'exécution de la tâche
- f: date de fin d'exécution de la tâche



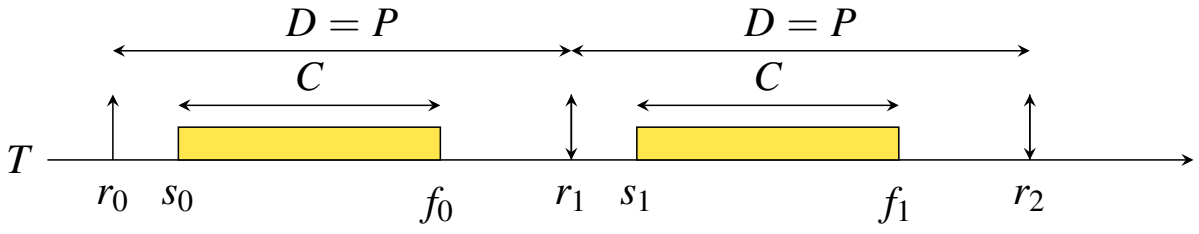
Paramètres d'une tâche périodique

- D: son délai critique, au-delà duquel le résultat est jugé comme étant non pertinent
- P: sa période, pour le cas d'une tâche périodique
- d: pour une tâche à contraintes strictes, son échéance, calculée comme étant $d = r + D$



Paramètres d'une tâche périodique à échéance sur requête

- $D = P$: son délai critique, au-delà duquel le résultat est jugé comme étant non pertinent



Autres paramètres

- $u = \frac{C}{P}$: son facteur d'utilisation du processeur
- $ch = \frac{C}{D}$: son facteur de charge du processeur
- $L = D - C$: sa laxité nominal. Indique le retard maximum que peut prendre la tâche sans dépasser son échéance
- $D(t) = d - t$: son délai critique résiduel au temps t
- $C(t)$: sa durée d'exécution résiduelle au temps t
- $L(t) = D(t) - C(t)$: sa laxité résiduelle au temps t
- $TR = f - r$: son temps de réponse. L'échéance est respectée si $tr \leq D$

Nous pouvons noter les propriétés suivantes:

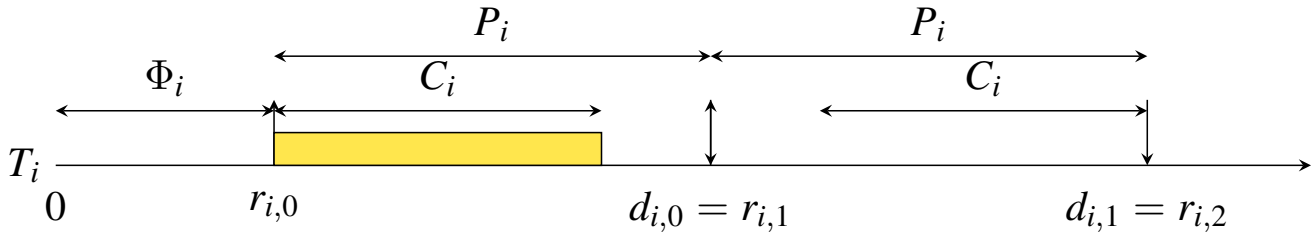
- $u \leq 1$
- $ch \leq 1$
- $0 \leq D(t) \leq D$
- $0 \leq C(t) \leq C$
- $L(t) = D(t) - C(t) = D + r - t - C(t)$

Tâches périodiques

Pour les tâches à échéances sur requête, nous pouvons calculer les temps d'arrivée des tâches, ainsi que leurs échéances respectives de cette manière:

$$r_{i,j} = \Phi_i + (j-1)P_i \quad (1)$$

$$d_{i,j} = r_{i,j} + P_i = \Phi_i + jP_i \quad (2)$$



Définitions

- Une tâche est dite *faisable* si toutes ses instances peuvent se terminer en respectant leur échéance
- L'ensemble Γ des tâches à ordonnancer est dit *ordonnançable* (ou *faisable*) si toutes ses tâches de Γ sont faisables.

Hypothèses

	Pour la suite nous faisons les hypothèses suivantes
H1	Les instances d'une tâche périodique sont activées avec une période constante
H2	Toutes les instances d'une tâche ont le même pire temps d'exécution C_i
H3	Toutes les instances d'une tâche ont la même échéance relative D_i
H4	Toutes les tâches sont indépendantes. Il n'y a pas de dépendances entre tâches
H5	Une tâche ne peut se suspendre elle-même
H6	La surcharge liée aux opérations du noyau est négligée

Taux d'utilisation du processeur

- Pour une tâche τ_i, le facteur d'occupation est défini par:

$$u_i = \frac{C_i}{P_i} \tag{3}$$

- Pour l'ensemble des tâches Γ, le facteur d'utilisation est dès lors:

$$U = \sum_{i=1}^{n} \frac{C_i}{P_i} \tag{4}$$

- Une condition nécessaire (mais pas suffisante) pour qu'un ensemble de tâches soit ordonnançable:

$$U = \sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1 \tag{5}$$

U

- U doit être plus petit que 1
- Mais il dépend des caractéristiques des tâches et de l'algorithme appliqué
- Il existe un $U_{ub}(\Gamma, A)$ (Upper Bound) au-delà duquel les tâches ne sont pas ordonnançables
- Lorsque $U = U_{ub}(\Gamma, A)$, l'ensemble Γ *utilise entièrement* le processeur
  - L'augmentation du temps d'exécution d'une des tâches rend l'ensemble non ordonnançable
- Pour un algorithme A donné, il existe une valeur minimale de U_{ub} (Least Upper Bound):

$$U_{lub}(A) = \min_{\Gamma} U_{ub}(\Gamma, A) \tag{6}$$

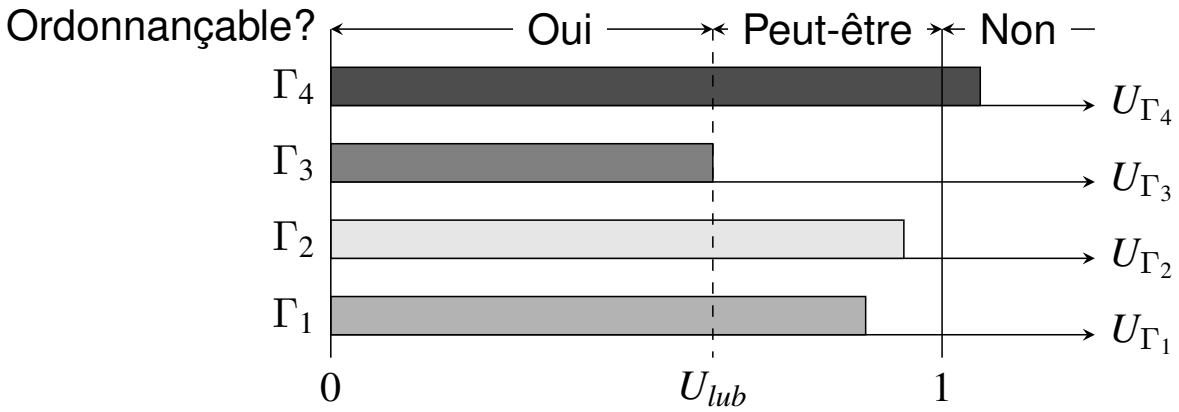
U

De ce fait, un ensemble de tâches Γ est ordonnançable si son facteur d'utilisation du processeur est inférieur à cette limite minimale:

$$U_\Gamma = \sum_{i=1}^{n} \frac{C_i}{P_i} \leq U_{lub}(A) = \min_{\Gamma} U_{ub}(\Gamma, A) \Rightarrow \Gamma \text{est ordonnançable} \quad (7)$$

Il s'agit d'une condition *suffisante*, mais pas nécessaire.

Least Upper Bound



Cyclic scheduling: Introduction

Hypothèse

Nous ne disposons pas d'un noyau multi-tâche

- Question: Comment ordonnancer les tâches (traitements)?
  - Chaque tâche possède:
    - Un temps d'exécution maximal
    - Une période
    - Une échéance (= Période)
  - But: Ordonnancer les traitements
- Une solution: *Cyclic scheduling* (aussi appelé *Timeline scheduling*)

Cyclic scheduling

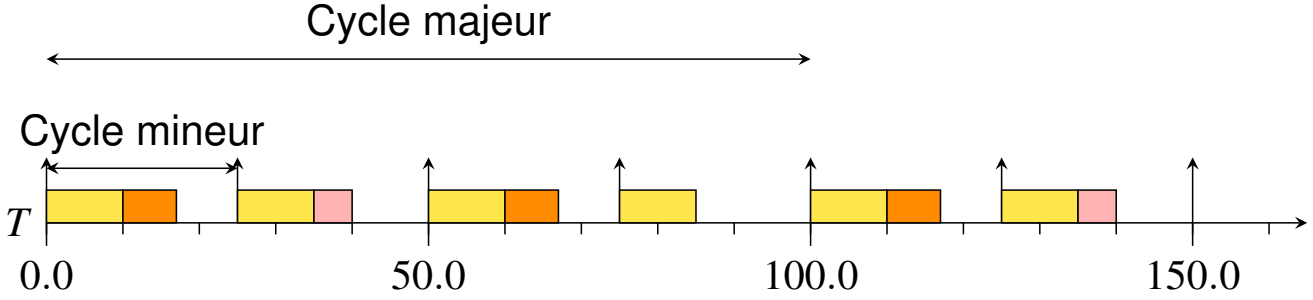
- Approche la plus utilisée dans le contrôle de trafic aérien
- permet de gérer un ensemble de tâches temps-réel
- Méthode:
  - Une tâche périodique principale (slice)
  - A chaque période, une ou plusieurs tâches sont exécutées
  - Un timer synchronise l'activation de la tâche principale
  - Les tâches sont en fait des procédures
  - Problème: Ordonnancer ces tâches
- Avantages:
  - Déterminisme
- Désavantages:
  - Programmation pouvant être laborieuse
  - Gestion des dépassements d'échéances

Cyclic scheduling

- Cycle mineur: $pgcd(P_i)$
- Cycle majeur: $ppcm(P_i)$
- La tâche principale est exécutée tous les cycles mineurs
- Le système se répète tous les cycles majeurs
- Il faut arranger les appels de procédure de manière à respecter les échéances

Exemple

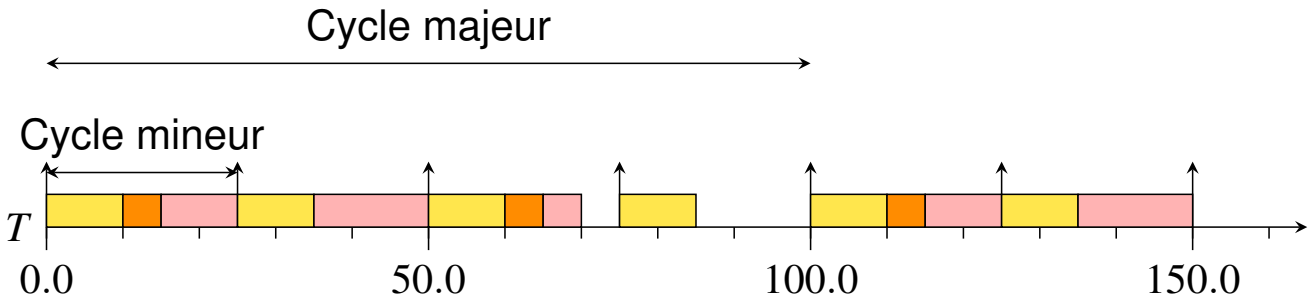
Tâche	Temps d'exec.	Fréquence	Période
T_a	10 ms	40 Hz	25 ms
T_b	7.5 ms	20 Hz	50 ms
T_c	5 ms	10 Hz	100 ms



Cyclic scheduling

- Les tâches peuvent être décomposées en sous-tâches (pas forcément aisé)

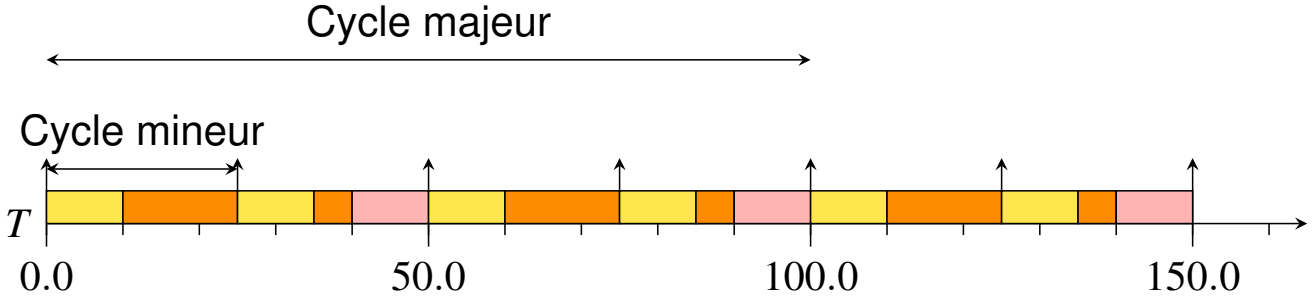
Tâche	Coût	Période
T_1	10	25
T_2	5	50
T_3	30	100



Taux d'utilisation du processeur

- Un taux d'utilisation de 1 est gérable

Tâche	Coût	Période
T_1	10	25
T_2	20	50
T_3	20	100



Problèmes

Tâche	Temps d'exec.	Fréquence	Période
T_a	10 ms	40 Hz	25 ms
T_b	7.5 ms	20 Hz	45 ms
T_c	5 ms	10 Hz	100 ms

- Cycle mineur: PGCD(Périodes)=5 ms. Période du slice
- Cycle majeur: PPCM(Périodes)= 900 ms
- Problèmes:
  - Cycle majeur très long, comprenant 45 cycles mineurs
  - Comment gérer la tâche T_a?

Implémentation (1)

Exemple 1

```
while (1) {
    rt_task_wait_period();
    procedure1();procedure2();
    rt_task_wait_period();
    procedure1();procedure3();
    rt_task_wait_period();
    procedure1();procedure2();
    rt_task_wait_period();
    procedure1();
}
```

Implémentation (2)

Exemple 2

```
while (1) {
    rt_task_wait_period();
    switch(counter) {
    case 0:procedure1();procedure2();break;
    case 1:procedure1();procedure3();break;
    case 2:procedure1();procedure2();break;
    case 3:procedure1();break;
    default: break; // Aie aie aie
    }
    counter = ( counter + 1 ) % taille_cycle;
}
```


Implémentation (3)

Exemple 3

```
int tab_exec[taille_cycle][4];

while (1) {
    rt_task_wait_period();
    if (tab_exec[cycle][0])
      procedure0();
    if (tab_exec[cycle][1])
      procedure1();
    if (tab_exec[cycle][2])
      procedure2();
    if (tab_exec[cycle][3])
      procedure3();
    cycle = ( cycle + 1 ) % taille_cycle;
}
```

Implémentation (4)

Exemple 4

```
while (1) {
    rt_task_wait_period();
    for(int t=0;t<nbTask;t++) {
        if (tab_exec[t][cycle])
            tab_exec[t][cycle]();
    cycle = ( cycle + 1 ) % taille_cycle;
}
```

- De quel type est `tab_exec`?

Conclusion sur Cyclic scheduling

- Avantages
  - Utile si pas de noyau
  - Bien prédictible
- Désavantages
  - Délicat à implémenter en fonction des fréquences
  - Gestion des dépassements

$\Rightarrow$ Les systèmes multi-tâches!

Algorithmes

- Rate Monotonic (RM)
- Deadline Monotonic (DM)
- Earliest Deadline First (EDF)
- Least Laxity First (LLF)

Rate Monotonic

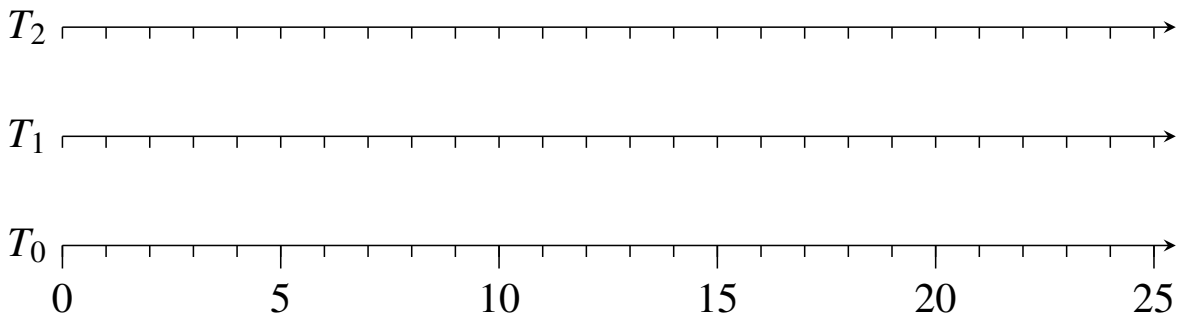
- Hypothèses
  - Tâches périodiques
  - Tâches à échéance sur requête
  - Priorité fixe
  - Préemption

Règle

Plus la période d'une tâche est petite, plus sa priorité est grande

Rate Monotonic: Exemple

Tâche	Coût	Période
T_0	2	6
T_1	3	8
T_2	4	24

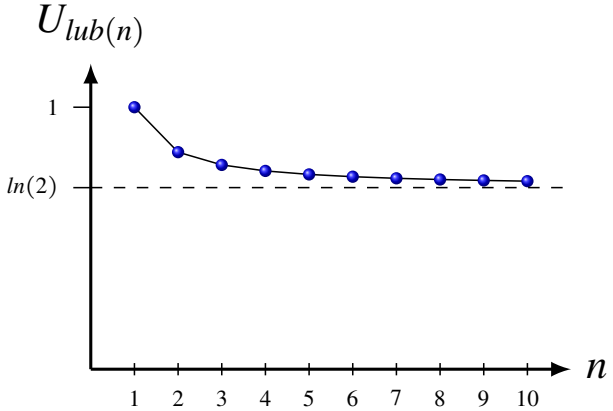


Rate Monotonic: ordonnançabilité

Condition suffisante d'ordonnançabilité (Liu et Layland)

$$\sum_{i=1}^{n} \frac{C_i}{P_i} \leq U_{lub_{RM}}(n) = n(2^{\frac{1}{n}} - 1) = n(\sqrt[n]{2} - 1) \quad (8)$$

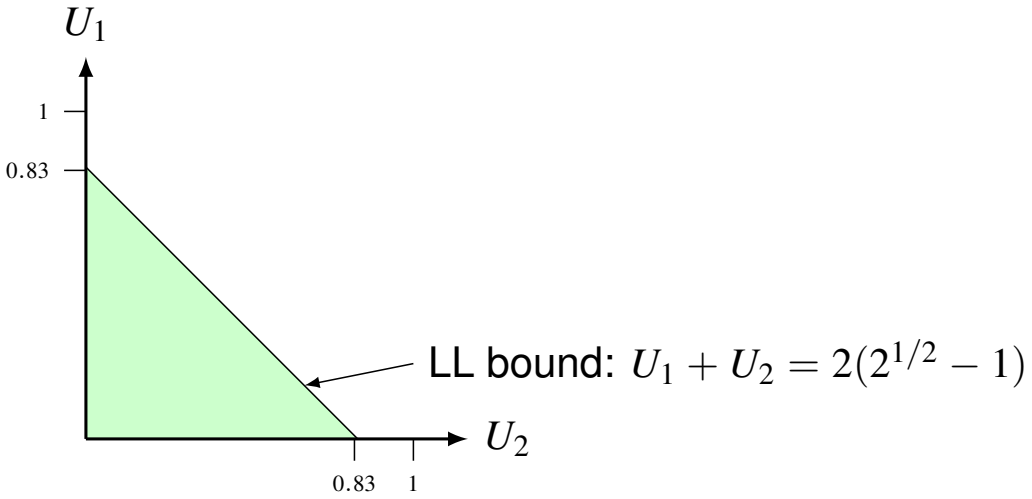
n	U_{lub}
1	1.000
2	0.828
3	0.780
4	0.757
5	0.743



U_{lub} pour de grandes valeurs de n:

$$U_{lub} = \lim_{n \to \infty} n(2^{\frac{1}{n}} - 1) = \ln 2 \simeq 0.69 \quad (9)$$

Rate Monotonic: ordonnançabilité



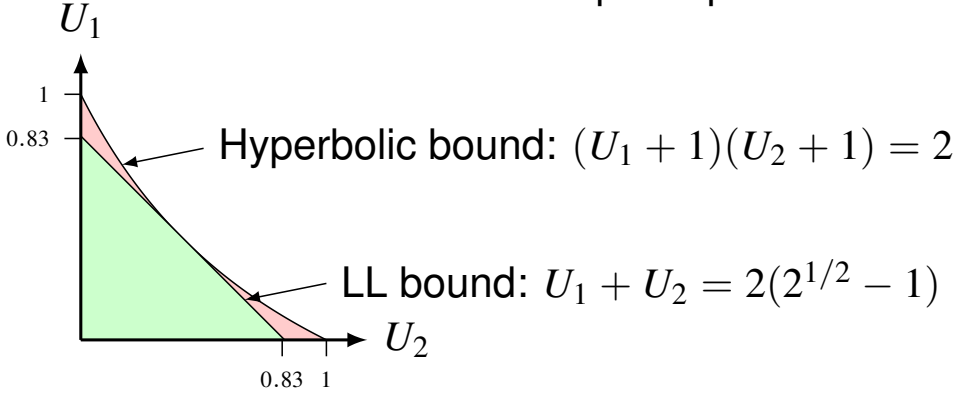
Rate Monotonic: ordonnançabilité

- En 2001, une nouvelle condition d'ordonnançabilité a été proposée: *Hyperbolic Bound*

Condition suffisante d'ordonnançabilité

$$\prod_{i=1}^{n}(U_i + 1) \leq 2 \tag{10}$$

- Cette condition est moins restrictive que la précédente.



Question

- Vous savez que vous avez besoin de 4 heures de travail à la maison pour finaliser un laboratoire PTR
- Vous devez sortir votre chien incontinent toutes les 2 heures pour une promenade de 30 minutes
- Vous devez jouer avec votre chat toutes les heures pendant 15 minutes afin d'éviter qu'il ne ruine votre canapé
- Sachant que vous commencez à travailler à 8 heure du matin et que vous ne ferez pas de pause repas, à quelle heure finirez-vous votre laboratoire?

Deadline Monotonic

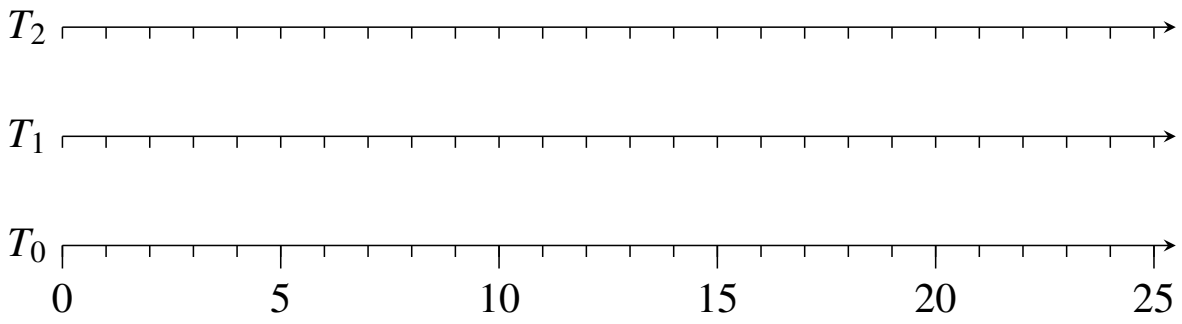
- Hypothèses
  - Tâches périodiques
  - Priorité fixe
  - Préemption

Règle

Plus le délai critique d'une tâche est petit, plus sa priorité est grande

Deadline Monotonic: Exemple

Tâche	Coût	Période	Echéance
T_0	2	6	5
T_1	3	8	4
T_2	4	24	20



Deadline Monotonic: ordonnançabilité

Condition suffisante d'ordonnançabilité

$$\sum_{i=1}^{n} \frac{C_i}{D_i} \leq n(2^{\frac{1}{n}} - 1) \tag{11}$$

- Pas optimal
  - Utilisation du processeur surestimée

Deadline Monotonic: ordonnançabilité

- Nous pouvons observer:
  1. Le pire cas au niveau des demandes d'utilisation du processeur se trouve au moment où toutes les tâches sont activées simultanément;
  2. Le pire temps de réponse d'une tâche correspond à la somme de son temps d'exécution et des interférences des tâches de priorité supérieure.
- Hypothèse: les tâches sont ordonnées selon l'ordre ascendant de leurs échéances
- Ordonnançables si:

$$\forall i : 1 \leq i \leq n \quad C_i + I_i \leq D_i \tag{12}$$

  Où I_i est l'interférence mesurée sur la tâche τ_i:

$$I_i = \sum_{j=1}^{i-1} \left\lceil \frac{D_i}{P_j} \right\rceil C_j \tag{13}$$

- Test suffisant, mais pas nécessaire

Ordonnançabilité: test en zone critique

- Si:
  - $U_{lub} <$ le facteur d'utilisation du processeur ≤ 1
  - Le test précédent à échoué
- Un test plus proche de la réalité d'exécution peut être conduit (Audlsey et al.)
- Applicable à des priorités fixes (RM et DM)
- Même observation que précédemment:
  1. Le pire cas au niveau des demandes d'utilisation du processeur se trouve au moment où toutes les tâches sont activées simultanément;
  2. Le pire temps de réponse d'une tâche correspond à la somme de son temps d'exécution et des interférences des tâches de priorité supérieure.

Ordonnançabilité: test en zone critique

$$R_i = C_i + I_i \tag{14}$$

Où

$$I_i = \sum_{j=1}^{i-1} \left\lceil \frac{R_i}{P_j} \right\rceil C_j \tag{15}$$

Donc

$$R_i = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{R_i}{P_j} \right\rceil C_j \tag{16}$$

- Test suffisant ET nécessaire
- Comment calculer R_i?

Ordonnançabilité: test en zone critique

1. Premier point d'approximation:

$$R_i^0 = \sum_{j=1}^{i} C_j \tag{17}$$

2. Si $R_i^0 > D_i$, la tâche n'est pas ordonnançable
3. Calcul du point d'approximation suivant:

$$R_i^{n+1} = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{R_i^n}{P_j} \right\rceil C_j \tag{18}$$

4. Si $R_i^{n+1} = R_i^n$, nous avons atteint le pire temps de réponse pour la tâche, qui peut alors être comparé à D_i. Sinon, nous reprenons au point 3.

Ordonnançabilité: test en zone critique

- Exemple

Tâche	Coût	Période	Echéance	Priorité
T_0	40	100	100	
T_1	40	150	150	
T_2	100	350	350	

Earliest Deadline First

- Hypothèses
  - Tâches périodiques
  - Tâches à échéance sur requête
  - Préemption

Règle

La tâche à l'échéance *absolue* la plus proche est la plus prioritaire

Earliest Deadline First: ordonnançabilité

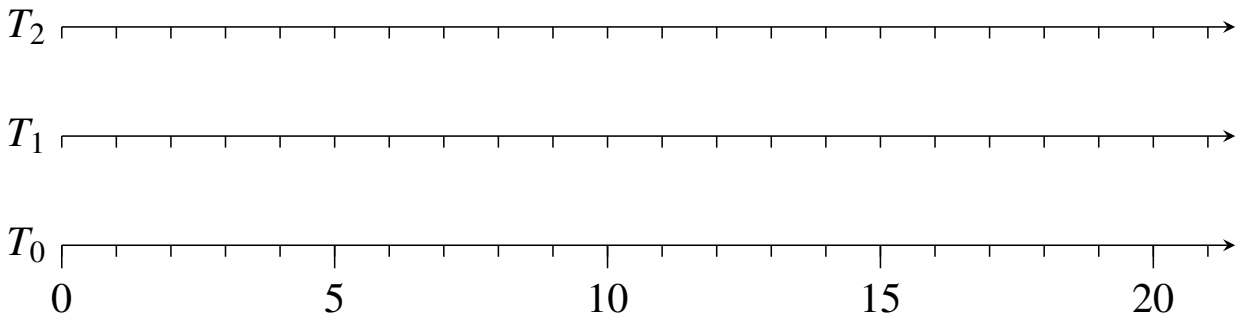
Condition suffisante et nécessaire d'ordonnançabilité

$$\sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1 \tag{19}$$

- Même analyse que pour le cas apériodique

Earliest Deadline First: Exemple

Tâche	Coût	Période
T_0	2	5
T_1	3	7
T_2	1	10



Earliest Deadline First

- EDF peut être exploité pour des tâches où $D < P$
  - Condition nécessaire et suffisante d'ordonnançabilité:

$$\forall L \in \Delta : L \geq \sum_{i=1}^{n} \left\lfloor \frac{L + T_i - D_i}{T_i} \right\rfloor C_i$$

  - où

$$\Delta = \{d_k | d_k \leq min(L^*, H)\}$$

  - et

$$L^* = \frac{\sum_{i=1}^{n} (T_i - D_i) U_i}{1 - U}$$

$$H = lcm(d_k)$$

Comparaison RM - EDF

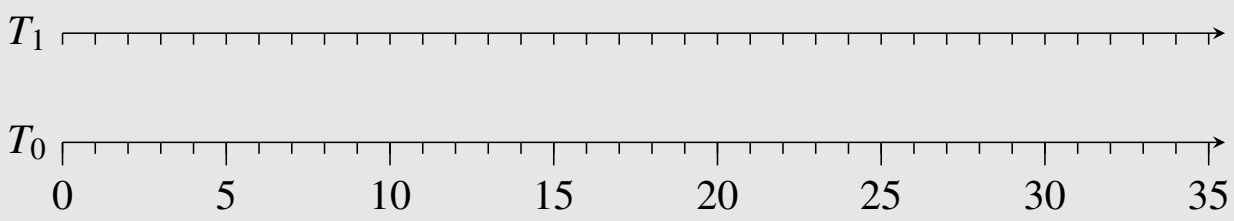
- Soient les tâches suivantes:

Tâche	Coût	Période
T_0	2	5
T_1	4	7

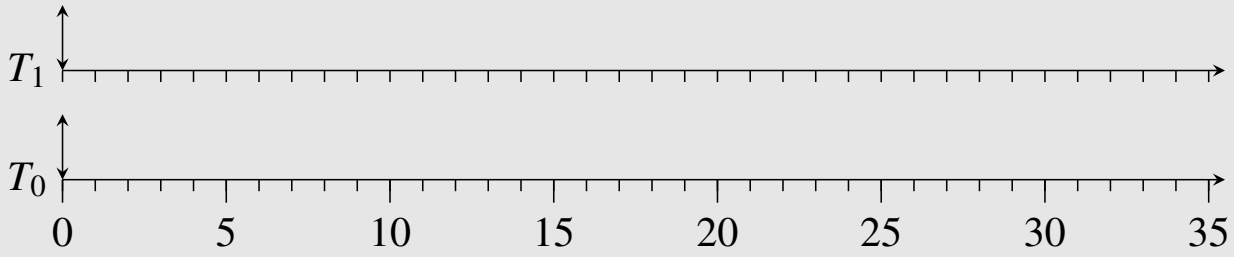
$$U = \qquad (20)$$

Comparaison RM - EDF

RM

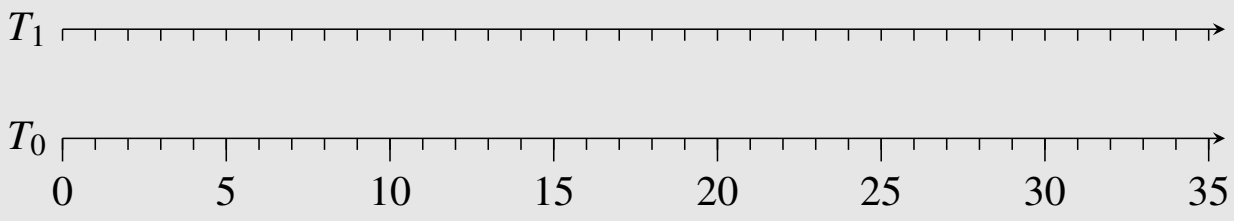


EDF

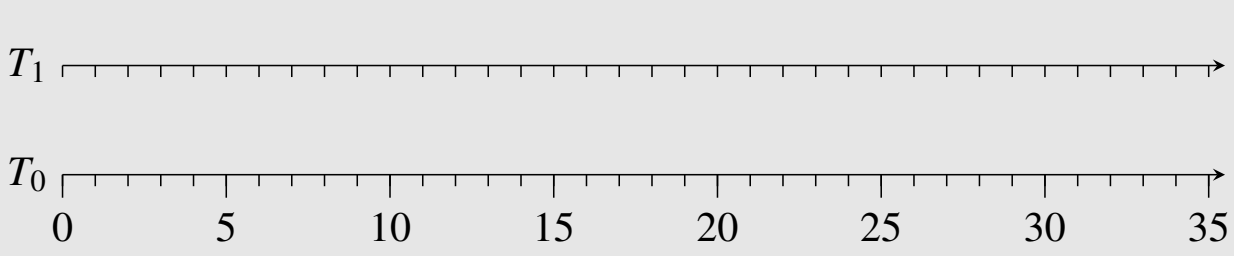


Comparaison RM - EDF

RM



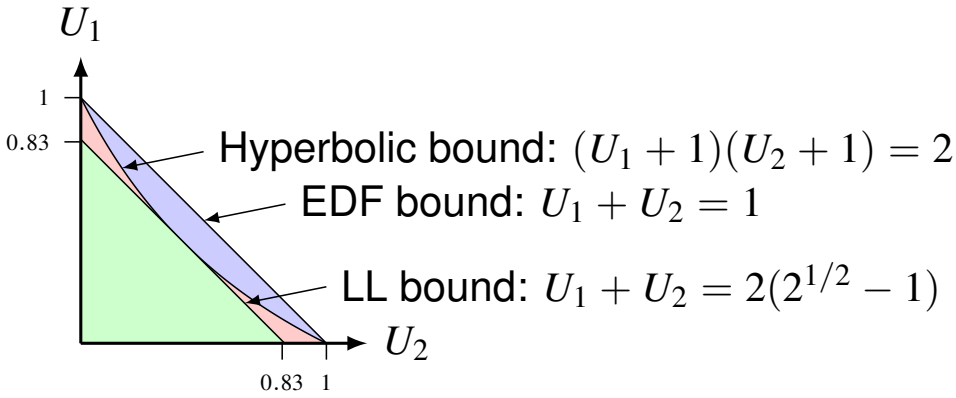
EDF



EDF: ordonnançabilité

Condition suffisante d'ordonnançabilité

$$U_1 + U_2 = 1 \tag{21}$$



Least Laxity First

- Aussi appelé *Least Slack Time*, LST
- Hypothèses
  - Tâches périodiques
  - Préemption

Règle

La tâche dont la laxité est la plus faible est la plus prioritaire

Least Laxity First

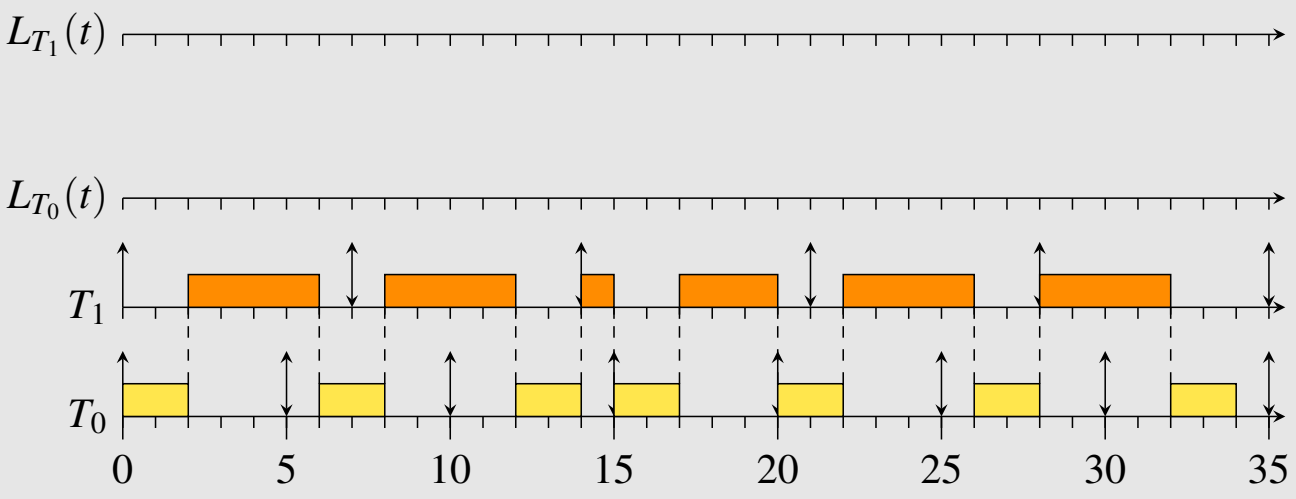
Définitions

- $L = D - C$: sa laxité nominal. Indique le retard maximum que peut prendre la tâche sans dépasser son échéance
- $D(t) = d - t$: son délai critique résiduel au temps t
- $C(t)$: sa durée d'exécution résiduelle au temps t
- $L(t) = D(t) - C(t)$: sa laxité résiduelle au temps t

- Deux options
  - Laxité résiduelle calculée au lancement de la tâche $\rightarrow$ EDF
  - Laxité résiduelle calculée à chaque instant t

Laxité: Exemple

! Ordonnancement EDF !



Least Laxity First: ordonnançabilité

Condition suffisante et nécessaire d'ordonnançabilité

$$\sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1 \tag{22}$$

- Même condition d'ordonnançabilité que pour EDF

Evaluation des performances

Nom	**Fonction**
Temps de réponse moyen	$\overline{t_r} = \frac{1}{n}\sum_{i=1}^{n}(f_i - r_i)$
Temps d'exécution total	$t_c = \max_i(f_i) - \min_i(r_i)$
Somme pondérée des fin d'exécution	$t_w = \sum_{i=1}^{n} w_i f_i$
Maximum lateness	$L_{max} = \max_i(f_i - d_i)$
Nombre maximum de tâche en retard	$N_{late} = \sum_{i=1}^{n} miss(f_i)$ $miss(f_i) = \begin{cases} 0 & \text{si } f_i \leq d_i \\ 1 & \text{sinon} \end{cases}$

Période d'étude

- Périodicité des tâches $\Rightarrow$ Ordonnancement cyclique
- L'algorithme d'ordonnancement consistera à trouver une séquence de tâches caractérisée par une périodicité de longueur L
- La période de longueur L est appelée période d'étude ou période de base.
- Ce qui signifie que si l'échéance de chacune des tâches est respectée dans la période d'étude, toutes les autres occurrences le seront. La séquence ainsi déterminée peut être reproduite à l'infini.
- Evaluation
  - Valide quel que soit l'algorithme d'ordonnancement
  - La période d'étude peut s'avérer très longue selon les relations entre périodes.
  - Adapté aux ordonnancements hors-ligne (simulation)

Période d'étude

- Si les tâches sont synchrones (débutent au même instant)
  - $L = [0, PPCM(P_i)]$
- Si les tâches sont asynchrones (ne débutent pas au même instant)
  - $L = [\min\{r_{i,0}\}, 2 \times PPCM(P_i) + \max\{r_{i,0}\}]$
  - Où $r_{i,0}$ est la date d'activation de la première occurence de la tâche périodique T_i

Références


E. BINI, G.C. BUTTAZZO and G.M. BUTTAZZO.
« Rate Monotonic Analysis: The Hyperbolic Bound ».
IEEE Transactions on Computers, 52(7):933–942, jul 2003.