Chapitre III
Techniques d'ordonnancement temps réel

I. Introduction

L'ordonnancement des processus

Les processus concurrents doivent se partager le processeur, la mémoire et les entrées/sorties.
- Dans les systèmes anciens, les systèmes de traitement **par lots** mais aussi les systèmes d'exploitation pas vraiment multitâches tels que Windows 9x, l'ordonnancement était de type coopératif. L'ordonnanceur n'intervenait que lorsque le processus en cours se terminait ou se bloquait.
- Actuellement, sur les systèmes interactifs multitâches, parfois même multi utilisateurs et multi processeurs, l'ordonnancement doit être préemptif.

II. Critères d'ordonnancement des processus

- L'utilisation intensive du processeur
Le système perd son efficacité si le processeur passe trop de temps à attendre des E/S.
- L'équité
Tous les processus doivent avoir la possibilité d'utiliser le processeur.
- Utilisation répartie de l'ensemble des ressources
Une bonne stratégie consiste à évaluer et à bien répartir l'emploi de l'ensemble des ressources.
*Certains critères s'appliquent plus particulièrement aux systèmes de traitement **par lots** :*
- Le nombre de processus par unité de temps
Ce critère dépend cependant de la longueur des processus.
- La durée de rotation
C'est le délai moyen entre l'admission du processus et la fin de son exécution.
- Le temps d'attente
C'est le temps moyen qu'un processus passe à attendre. Il s'obtient en soustrayant la durée d'exécution du processus de sa durée de rotation. Cette durée est indépendante de la durée d'exécution du processus lui-même.
D'autres critères qui ne s'appliquent qu'aux systèmes interactifs :
- Le temps de réponse
C'est la vitesse de réaction aux interventions extérieures.
- La prévisibilité
Un système qui d'habitude réagit rapidement aux commandes mais qui parfois prend un temps beaucoup plus long sera perçu comme moins stable que s'il répondait à chaque fois dans un temps comparable même s'il est globalement plus lent.

III. Techniques d'ordonnancement

III.1. FCFS (first come first Served)

Dans cet algorithme ; connu sous le nom FIFO (First In, First Out) ; les processus sont rangés dans la file d'attente des processus prêts selon leur ordre d'arrivée.

1er cas: les processus ont un même temps d'arrivée

Suppose that the processes arrive at time 0, in the order: P1 , P3 , P2 , P4

Draw Gantt Chart and calculate the average waiting time using the given table ??

Process	Burst Time
P1	3
P2	9
P3	5
P4	7

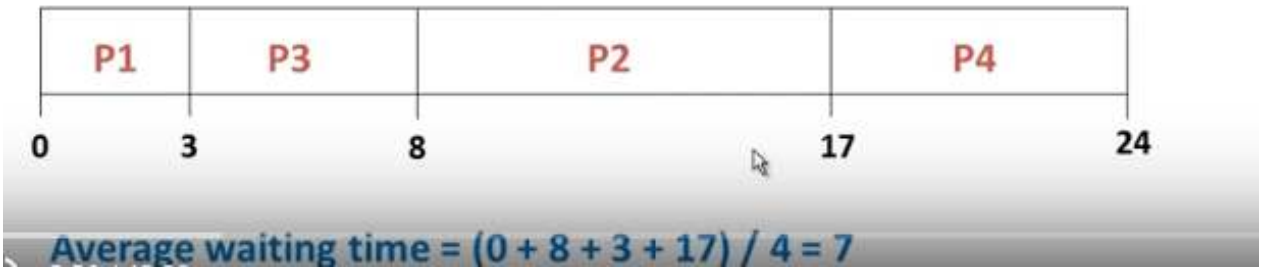
Waiting time :

P1 = 0

P2 = 8

P3 = 3

P4 = 17



Average waiting time = (0 + 8 + 3 + 17) / 4 = 7

2eme cas: les processus ont des temps d'arrivée différents

Draw Gantt Chart and calculate the average waiting time using the given table ??

Process	Burst Time	Arrival Time
P1	20	0
P2	12	3
P3	4	2
P4	9	5

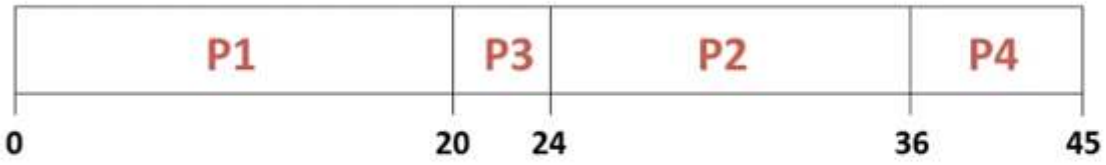
Waiting time : start time – arrival time

P1 = 0 – 0 = 0

P2 = 24 – 3 = 21

P3 = 20 – 2 = 18

P4 = 36 – 5 = 31



Average waiting time = (0 + 21 + 18 + 31) / 4 = 70 / 4

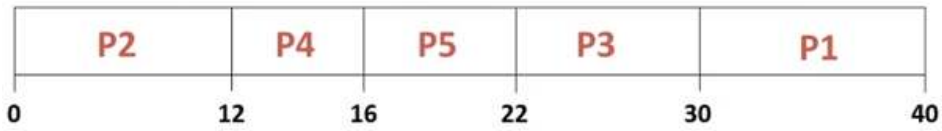
III.2.a. SJF (Short Job First) Non-preemptive

Draw Gantt Chart and calculate the average waiting time using the given table ??

Process	Burst Time	Arrival Time
P2	12	0
P3	8	3
P4	4	5
P1	10	10
P5	6	12

Waiting time : start time – arrival time

P1 = 30 – 10 = 20
P2 = 0 – 0 = 0
P3 = 22 – 3 = 19
P4 = 12 – 5 = 7
P5 = 16 – 12 = 4



Average waiting time = (20 + 0 + 19 + 7 + 4) / 5 = 50 / 5 = 10

III.2.b. SJF (Short Job First) Preemptive

Appelé aussi **SRT -** ***Shorted Remaining Time*** **= l'algorithme du temps restant le plus court**

Draw Gantt Chart and calculate the average waiting time using the given table ??

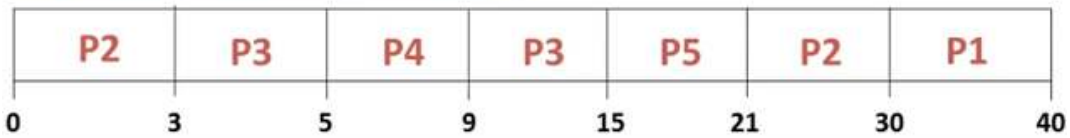
Process	Burst Time	Arrival Time
P2	12	0
P3	8	3
P4	4	5
P1	10	10
P5	6	12

Draw Gantt Chart and calculate the average waiting time using the given table ??

Process	Burst Time	Arrival Time
~~P2~~	~~12~~ 9	~~0~~
~~P3~~	~~8~~ 6	~~3~~
~~P4~~	~~4~~	~~5~~
~~P1~~	~~10~~	~~10~~
~~P5~~	~~6~~	~~12~~

Waiting time : start time – arrival time

P1 = 30 – 10 = 20
P2 = (0 – 0) + (21 - 3) = 18
P3 = (3 – 3) + (9 - 5) = 4
P4 = (5 – 5) = 0
P5 = 15 – 12 = 3



Average waiting time = (20 + 18 + 4 + 0 + 3) / 5 = 45 / 5 = 9

III.3. Round Robbin (le tourniquet)

Draw Gantt Chart and Calculate The Average Waiting Time , where **Quantum = 5 ms**

Process	Burst Time
P1	12
P2	8
P3	4
P4	10
P5	5

Process	Burst Time
~~P1~~	~~12~~ 7 2
~~P2~~	~~8~~ 3
~~P3~~	~~4~~
~~P4~~	~~10~~ 5
~~P5~~	~~5~~

Waiting time :

P1 = 0 + (24 - 5) + (37 - 29) = 27
P2 = 5 + (29 - 10) = 24
P3 = 10
P4 = 14 + (32 - 19) = 27
P5 = 19

Average waiting time = (27 + 24 + 10 + 27 + 19) / 5 = 107 / 5 = 21.4

IV. Techniques d'ordonnancement temps réel

Un algorithme d'ordonnancement étant défini comme un algorithme capable de donner une description (séquence) du travail à effectuer par le ou les processeurs,

- une séquence est dite **valide** si les échéances des tâches sont respectées.
- Un algorithme est dit **fiable** pour une configuration de tâches s'il produit une séquence valide sur une durée infinie quelles que soient les valeurs des premières dates de déclenchement des différentes tâches.
- Une configuration est dite **ordonnançable** s'il existe au moins un algorithme fiable.

Dans un contexte de tâches et d'algorithmes d'ordonnancement (affectation de priorités), nous allons qualifier l'algorithme d'ordonnancement étudié selon deux aspects :

– **optimalité**: si la configuration de tâches est ordonnançable dans cette catégorie d'algorithmes, alors elle le sera avec l'algorithme étudié ;

– **ordonnançabilité:** la capacité à pouvoir prévoir l'ordonnancement de la configuration de tâches en se basant sur des conditions nécessaires et/ou suffisantes ou des simulations de l'exécution.

IV.1. Modélisation des tâches périodiques

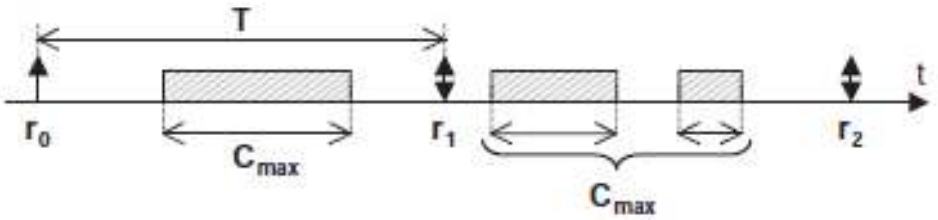
Ces tâches correspondent par exemple à des tâches de scrutation de capteurs pour effectuer des mesures régulières de grandeurs physiques. La modélisation des tâches périodiques va reposer sur trois paramètres temporels :
– **r0:** date de réveil de la tâche, c'est-à-dire la première date à laquelle la tâche demande le processeur ;
– **C**=Cmax: durée d'exécution maximale définie avec les restrictions exposées précédemment ;
– **T**: période d'exécution, c'est-à-dire la fréquence de renouvellement de la demande d'exécution de la tâche.
La date de réveil rk de la kième instance ou occurrence d'une tâche est donc définie par :

$$rk = r0 + kT$$

Nous considérons un environnement d'exécution stricte où la tâche doit avoir terminé son exécution avant la fin de sa période d'exécution; ***la tâche est dite à échéance sur requêt***e.

La figure ci-dessous présente l'exécution de deux occurrences d'une tâche périodique à échéance sur requête dans un diagramme de Gantt.

- Nous pouvons noter que la tâche s'exécute de façon complète dans la première occurrence, c'est-à-dire que, lorsqu'elle obtient le processeur, elle garde pendant toute sa durée d'exécution **C= Cmax** .
- En revanche, lors de la deuxième occurrence, la tâche est préemptée une fois lors de son exécution et son exécution s'effectue alors en deux fois.

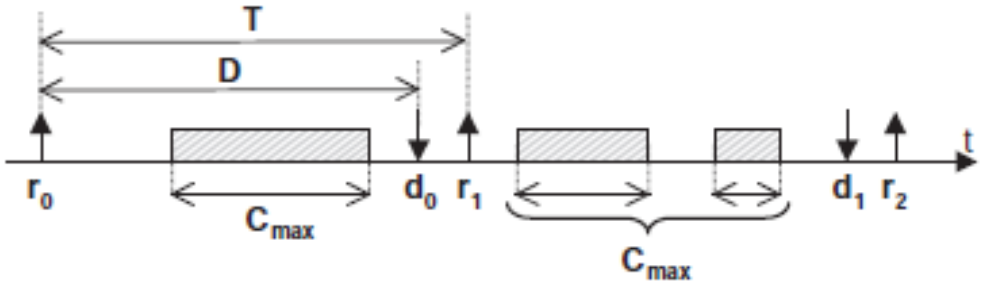


Représentation de l'exécution d'une tâche périodique à échéance sur requête.

La modélisation **des tâches périodiques strictes** va donc reposer sur quatre paramètres temporels :
– **r0** : date de réveil de la tâche, c'est-à-dire la première date à laquelle la tâche demande le processeur ;
– **C=Cmax** : durée d'exécution maximale définie avec les restrictions exposées précédemment ;
– **D** : délai critique, c'est-à-dire le délai au bout duquel la tâche doit être terminée par rapport à la date de réveil de l'instance en cours.
– **T** : période d'exécution, c'est-à-dire la fréquence de renouvellement de la demande d'exécution de la tâche.
Dans ce contexte, il est possible de définir l'échéance **dk** de la **k** ième instance d'une tâche par l'équation suivante :

$$dk = rk + D = r0 + kT + D$$

La figure ci-dessous représente l'exécution d'une telle tâche pour deux occurrences dans un diagramme de Gantt.



Représentation de l'exécution d'une tâche périodique avec une échéance plus petite que la période.

IV.2. Ordonnancement des tâches indépendantes périodiques

IV.2.1. Algorithmes d'ordonnancement à priorités fixes

Algorithme d'ordonnancement « Rate Monotonic » (RM)

Dans un contexte de tâches indépendantes, périodiques, à échéance sur requête et à départ simultané, nous allons effectuer une affectation des priorités aux tâches selon la période :

- ✓ **plus la période de la tâche est petite, plus la priorité de la tâche est grande**.
- ✓ **La tâche conserve cette priorité pendant toute son exécution**.

Cette règle d'affectation des priorités est appelée l'algorithme d'ordonnancement « Rate Monotonic ou RM ». Concernant cet algorithme d'ordonnancement pour une configuration de **n** tâches ayant les paramètres (**ri**, **Ci**, **Di**, **Ti**) avec **ri**=0 pour tout i et **Di=Ti**, nous avons les résultats suivants :

– l'algorithme d'ordonnancement RM est **optimal** dans la classe des algorithmes à priorités fixes, c'est-à-dire que, si une configuration de tâches est ordonnançable, elle le sera en affectant les priorités selon RM ;

– une **condition suffisante d'ordonnançabilité** d'une configuration est obtenue pour un facteur d'utilisation **U** du processeur suivant l'inégalité suivante :

$$U = \sum_{i=1}^{n} C_i / T_i \leq n\left(2^{\frac{1}{n}} - 1\right) \quad \textbf{(1)}$$

- Prenons l'exemple de la configuration à trois tâches décrite dans le tableau 1. Nous vérifions que nous sommes en présence d'une configuration de tâches indépendantes, périodiques, à échéance sur requête et à départ simultané.
- Les priorités ont été affectées selon l'algorithme d'ordonnancement RM. Le facteur d'utilisation **U** de cette configuration est donné par:

$$U = \sum_{i=1}^{3} C_i / T_i = \frac{20}{100} + \frac{40}{150} + \frac{100}{350} \approx 0{,}75 \leq 3\left(2^{\frac{1}{3}} - 1\right) \approx 0{,}779$$

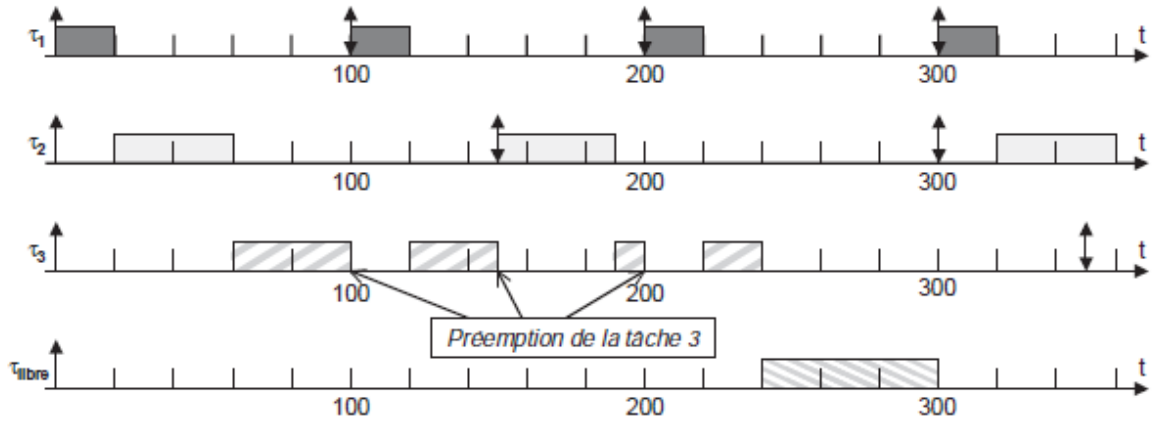
Par conséquent, nous pouvons en déduire que cette configuration de tâches est ordonnançable. Nous pouvons construire une partie de la séquence qui aura pour période **H** :

$$H = PPCM\{T_i\}_{i \in [1,3]} = PPCM\{100,150,350\} = 2100$$

Tableau 1. Exemple d'une configuration de trois tâches périodiques avec une affectation des priorités selon RM.

Tâche	r_i	C_i	D_i	T_i	Priorité selon RM
τ_1	0	20	100	100	3
τ_2	0	40	150	150	2
τ_3	0	100	350	350	1

Cette séquence, représentée sur la figure, montre en particulier les trois préemptions de la tâche de plus faible priorité **t3** alors que la tâche de plus forte priorité **t1** s'exécute dès sa demande (dates de réveil).



Exemple d'une partie de la séquence d'exécution d'une configuration de trois tâches traitée avec l'algorithme RM.

Nous pouvons noter aussi la présence de temps libres du processeur qui sur l'ensemble de la période d'étude sont :

$$T_{\text{libre}} = (1 - 0{,}75238) \times 2100 = 520$$

- La condition, exprimée par l'équation (1), est très restrictive (valeur minimale de **U**). Nous pouvons utiliser le théorème de la zone critique qui exprime le fait que si toutes les tâches sont à départ simultanée et si elles respectent leur première échéance alors la configuration est ordonnançable quel que soit l'instant d'arrivée des tâches dans la suite.
- Cette condition est nécessaire et suffisante si toutes les tâches sont à départ simultané et suffisante si les tâches sont à départ différé.

Un ensemble de **n** tâches T {**t1**, **t2**, …, **tn**} ordonnées suivant les priorités (**t1** la tâche la plus prioritaire et **tn** la tâche la moins prioritaire) définies par les paramètres temporels (**ri, Ci, Di, Ti**) est ordonnançable si et seulement si :

$$\forall i,\ 1 \le i \le n \quad \min_{0 \le t \le D_i} \sum_{j=1}^{i} \frac{C_j}{t} \left\lceil \frac{t}{T_j} \right\rceil \le 1 \qquad \textbf{(2)}$$

Cette condition correspond à tester intervalle de temps par intervalle de temps la possibilité d'exécuter les différentes tâches.

- La valeur [t/**Tj**] correspond au nombre de réveils de la tâche **tj** dans l'intervalle de temps [0,t].
- Le terme **Cj** [t/**Tj**] représente la demande processeur de la tâche **tj** dans ce même intervalle.
- Le temps processeur demandé jusqu'à l'instant t par la tâche **tj** et toutes les tâches **tj** de plus forte priorité (j ∈[0,i-1]) doit être inférieur à l'échéance **di** (**Di =Tj**) de la tâche **ti** , soit :

$$\sum_{j=1}^{i-1} C_j \left\lceil \frac{t}{T_j} \right\rceil + C_i \leq D_i$$

Prenons l'exemple de la configuration à trois tâches décrite dans le tableau 2. Nous vérifions que nous sommes en présence d'une configuration de tâches indépendantes, périodiques, à échéance sur requête et à départ simultané. Les priorités ont été affectées selon l'algorithme d'ordonnancement RM. Le facteur d'utilisation **U** de cette configuration est donnée par :

$$U = \sum_{i=1}^{3} C_i / T_i = \frac{1}{4} + \frac{2}{6} + \frac{2}{8} \approx 0{,}833 > 3\left(2^{\frac{1}{3}} - 1\right) \approx 0{,}779$$

La condition suffisante vue précédemment n'est pas satisfaite. Pour savoir si cette configuration est ordonnançable, il est donc nécessaire de procéder à une simulation complète sur la période d'étude (H=24) ou d'utiliser la condition, dite de la zone critique exprimée par **l'équation (2)**. Le calcul est effectué jusqu'au temps Di à chaque nouvelle activation d'une tâche pour les trois tâches de la configuration. Ainsi, pour la première tâche avec D1=4, nous avons:

$$\frac{C_1}{t}\left\lceil \frac{t}{T_1} \right\rceil \quad \text{pour } t = 4 \quad \frac{1}{4}\left\lceil \frac{4}{4} \right\rceil = \frac{1}{4} \rightarrow \text{minimum} \leq 1$$

Pour la deuxième tâche avec D2 =6, nous avons le minimum inférieur ou égal à 1 :

$$\frac{C_1}{t}\left\lceil \frac{t}{T_1} \right\rceil + \frac{C_2}{t}\left\lceil \frac{t}{T_2} \right\rceil \quad \text{pour } t = 4 \quad \frac{1}{4}\left\lceil \frac{4}{4} \right\rceil + \frac{2}{4}\left\lceil \frac{4}{6} \right\rceil = \frac{3}{4} \rightarrow \leq 1$$

$$\text{pour } t = 6 \quad \frac{1}{6}\left\lceil \frac{6}{4} \right\rceil + \frac{2}{6}\left\lceil \frac{6}{6} \right\rceil = \frac{2}{3} \rightarrow \geq 1$$

Pour la troisième tâche avec D3=8, nous avons le minimum inférieur ou égal à 1 :

$$\frac{C_1}{t}\left\lceil \frac{t}{T_1} \right\rceil + \frac{C_2}{t}\left\lceil \frac{t}{T_2} \right\rceil + \frac{C_3}{t}\left\lceil \frac{t}{T_3} \right\rceil \quad \text{pour } t = 4 \quad \frac{1}{4}\left\lceil \frac{4}{4} \right\rceil + \frac{2}{4}\left\lceil \frac{4}{6} \right\rceil + \frac{2}{4}\left\lceil \frac{4}{8} \right\rceil = \frac{5}{4} \rightarrow > 1$$

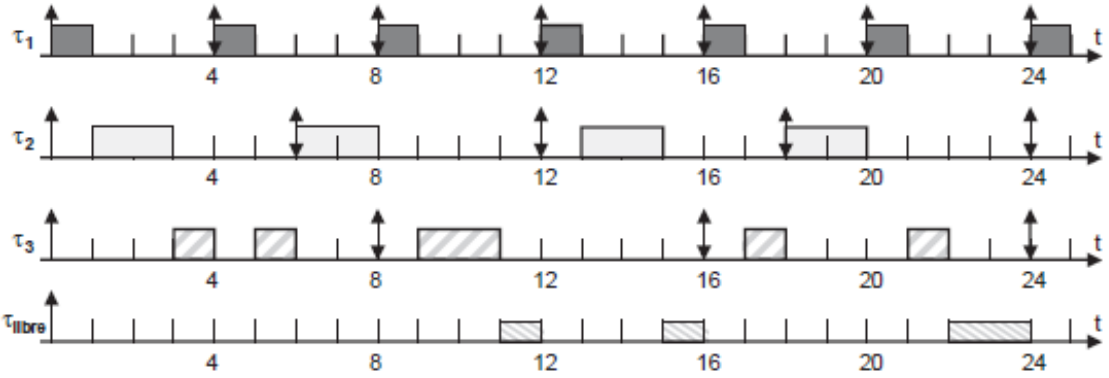
$$\text{pour } t = 6 \quad \frac{1}{6}\left\lceil \frac{6}{4} \right\rceil + \frac{2}{6}\left\lceil \frac{6}{6} \right\rceil + \frac{2}{6}\left\lceil \frac{6}{8} \right\rceil = 1 \rightarrow = 1$$

$$\text{pour } t = 8 \quad \frac{1}{8}\left\lceil \frac{8}{4} \right\rceil + \frac{2}{8}\left\lceil \frac{8}{6} \right\rceil + \frac{2}{8}\left\lceil \frac{8}{8} \right\rceil = 1 \rightarrow = 1$$

Tableau 2. Exemple d'une configuration de trois tâches périodiques avec une affectation des priorités selon RM.

Tâche	r_i	C_i	D_i	T_i	Priorité selon RM
τ_1	0	1	4	4	3
τ_2	0	2	6	6	2
τ_3	0	2	8	8	1

Nous pouvons donc conclure que la configuration est ordonnançable sans avoir à construire la séquence, représentée sur la figure ci-après, sur la période d'étude **H**=24.



Séquence d'exécution d'une configuration de trois tâches donnée dans le tableau 2 et traitée avec l'algorithme RM.

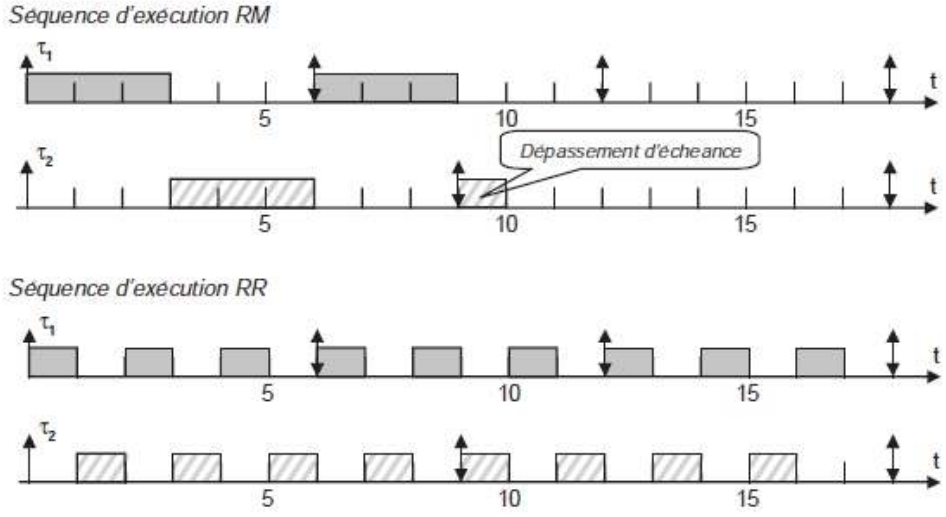
❖ Comparaison des algorithmes d'ordonnancement « Rate Monotonic » et Round Robbin

Les systèmes d'ordonnancement à priorité utilisent pour gérer des tâches dans une même file de priorité un ordonnancement de type «Round Robbin».
Dans le cadre de notre étude où la priorité est affectée en fonction des paramètres temporels de la tâche, il est intéressant de comparer l'ordonnancement «Round Robbin » et l'ordonnancement à priorité fixe de type RM.
Etudions deux exemples et comparons les ordonnancements dans les deux cas. Il est important de noter que l'ordonnancement « à tourniquet » est très dépendant des deux paramètres : **gestion de la file d'attente et quantum d'affectation processeur**. La file d'attente sera gérée selon une file de **type FIFO.**
De plus nous supposons dans nos exemples que le quantum est fixé par le pas de temps discret d'exécution des tâches « 1 ».
Remarquons aussi que, dans le cas de l'ordonnancement « à tourniquet », la notion de période d'étude n'est plus à considérer étant donné la gestion de **la file d'attente asynchrone des périodes des tâches.**

- Prenons l'exemple de la configuration à deux tâches décrite dans le tableau 4. Nous vérifions que nous sommes en présence d'une configuration de tâches indépendantes, périodiques, à échéance sur requête et à départ simultané.
- Les priorités ont été affectées selon l'algorithme d'ordonnancement RM et le facteur d'utilisation est **U** = 0,94.
- La file d'attente du tourniquet est ordonnée au départ avec la tâche **t1** en premier et la tâche **t2** en second. Dans ces conditions, le tracé de la séquence d'exécution montre que la séquence d'exécution, basée sur l'ordonnancement RM, ne permet pas de respecter les échéances (tâche **t2** échoue).
-

Tableau 4. Exemple d'une configuration de deux tâches périodiques avec une affectation des priorités selon RM : comparaison des ordonnancements « Round Robbin » et RM.

Tâche	r_i	C_i	D_i	T_i	Priorité selon RM
τ_1	0	3	6	6	2
τ_2	0	4	9	9	1



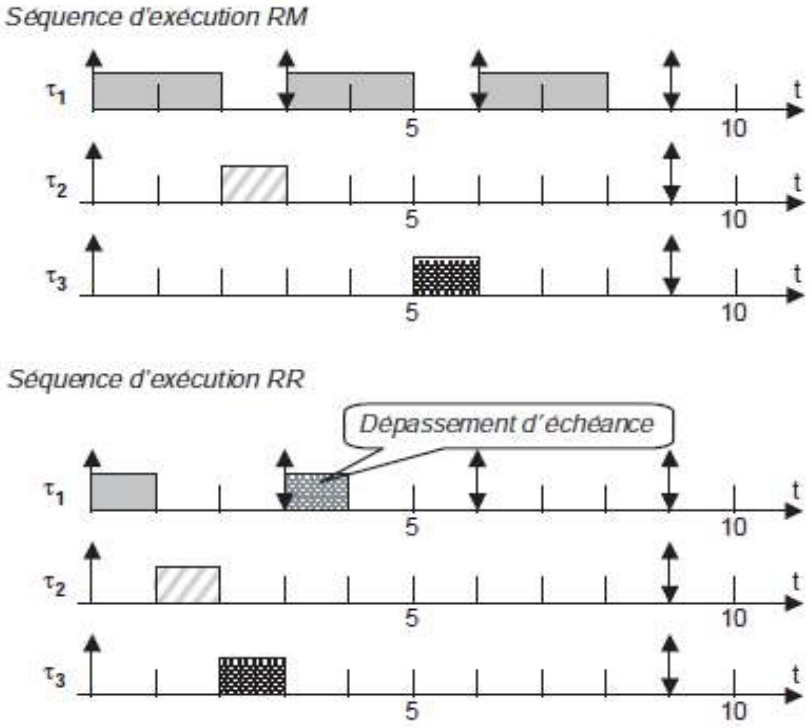
Séquence d'exécution d'une configuration de deux tâches donnée dans le tableau 5 et traitée avec les deux algorithmes RM et « à tourniquet ».

Conclusion: *De cet exemple, nous pouvons conclure que l'algorithme basé sur le tourniquet permet d'ordonnancer plus de configurations que l'algorithme RM.*

Mais, considérons l'exemple de la configuration à trois tâches décrites dans le tableau 5. Nous vérifions que nous sommes en présence d'une configuration de tâches indépendantes, périodiques, à échéance sur requête et à départ simultané. Les priorités ont été affectées selon l'algorithme d'ordonnancement RM avec une liberté de choix pour les tâches **t2** et **t3**. Le facteur d'utilisation est **U**= 0,88. La file d'attente du tourniquet est composée au départ des tâches **t1** en premier, ensuite **t2** et enfin **t3** en dernier.

Tableau 5. Exemple d'une configuration de trois tâches périodiques avec une affectation des priorités selon RM : comparaison des ordonnancements « à tourniquet » et RM.

Tâche	r_i	C_i	D_i	T_i	Priorité selon RM
τ_1	0	2	3	3	3
τ_2	0	1	9	9	2
τ_3	0	1	9	9	1



Dans ces conditions, le tracé de la séquence d'exécution montre que la séquence d'exécution, basée sur l'ordonnancement « à tourniquet », ne permet pas de respecter les échéances (tâche **t1** échoue).
Conclusion: *À partir de ce deuxième exemple, nous pouvons donc conclure que l'algorithme RM permet d'ordonnancer* ***plus*** *de configurations que l'algorithme basé sur le Round Robbin. En réalité de nombreuses applications peuvent être ordonnancées à la fois par l'algorithme RM et l'algorithme RR.*

IV.2.2. Algorithmes d'ordonnancement à priorités variables

Nous avons jusqu'à présent considéré que la priorité affectée à une tâche restait constante pendant toute la durée de l'application. Nous allons nous intéresser à une autre catégorie d'algorithme d'ordonnancement pour laquelle la priorité des tâches varie au cours de l'exécution d'une tâche. Cette priorité est fonction d'une caractéristique temporelle dynamique de la tâche.

Algorithme d'ordonnancement « Earliest Deadline First »

Dans le cas de l'algorithme « Earliest Deadline First » ou EDF, la priorité des tâches est variable au cours de leur exécution et fonction de la prochaine échéance. Pour une instance **k** d'une tâche **ti**, la priorité est liée à la prochaine échéance **di, k** de cette tâche. À un instant t, la priorité peut être calculée à partir du délai critique dynamique **Di(t)** qui s'exprime sous la forme :

$$D_i(t) = d_{i,k} - t = r_{i,k} + D_i - t$$

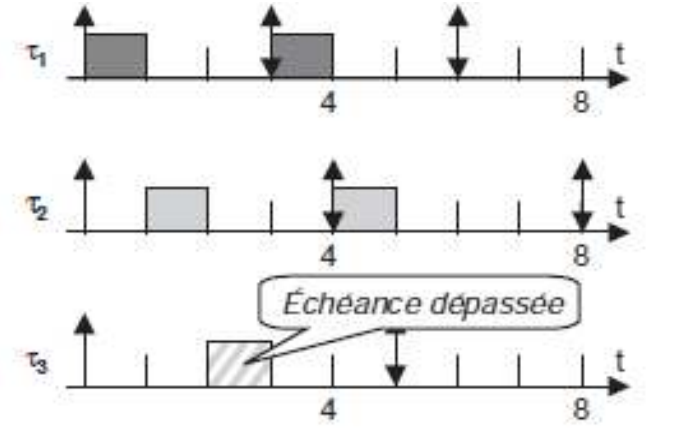
Nous pouvons faire les deux remarques suivantes :
– la priorité est variable, elle change au cours de l'exécution ;
– la priorité augmente si le délai critique dynamique de la tâche diminue.
Deadline First (EDF) donne la priorité à la tâche ayant l'échéance la plus proche. A chaque fois qu'une tâche est réveillée, l'ordonnanceur réévalue les tâches prêtes et sélectionne celle ayant l'échéance la plus courte.
Considérons l'exemple de la configuration à trois tâches décrite dans le tableau 6.

- Les priorités initiales ont été affectées en fonction du délai critique qui correspond à la première échéance.
- Le facteur d'utilisation de la configuration est **U**= 0,983 et la période d'étude est **H**=60. Par conséquent le temps libre est de 1.

Cette configuration n'est pas ordonnançable par l'algorithme à priorité fixe RM comme le montre le diagramme de Gantt de la figure

Tableau 6. Exemple d'une configuration de trois tâches périodiques pour l'étude de l'ordonnancement EDF.

Tâche	r_i	C_i	D_i	T_i	Priorité selon D_i
τ_1	0	1	3	3	3
τ_2	0	1	4	4	2
τ_3	0	2	5	5	1



Séquence d'exécution d'une configuration de trois tâches donnée dans le tableau 6 et traitée avec l'algorithme RM : non ordonnançable.

Par conséquent, étant donné la propriété d'optimalité de l'algorithme RM, cette configuration n'est ordonnançable par aucun algorithme à priorité fixe. En revanche, la figure présente la séquence d'exécution de cette configuration sur une partie de la période d'étude. L'évolution du **délai critique dynamique**, qui conditionne la priorité des tâches, est notée sur chaque séquence.

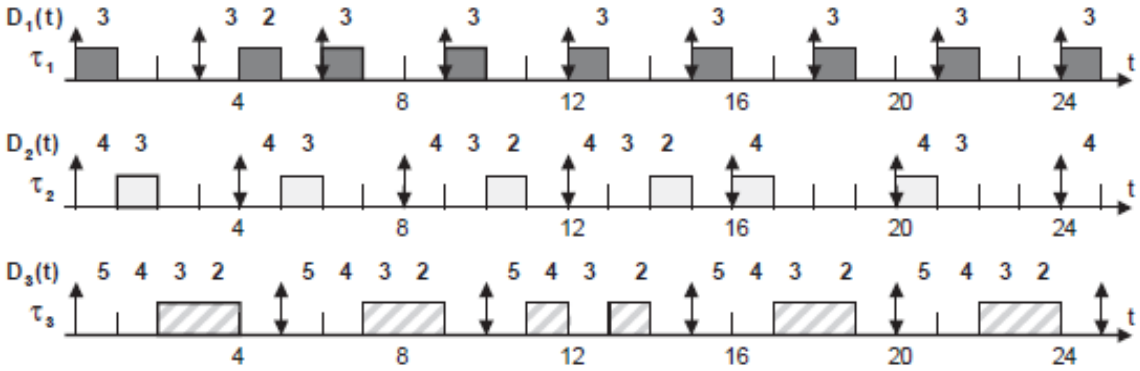
Comme pour les algorithmes à priorité fixe, nous disposons d'une condition d'ordonnançabilité pour l'algorithme EDF.

Pour une configuration de tâches indépendantes, périodiques, à échéance sur requête et à départ simultané, la condition nécessaire et suffisante d'ordonnançabilité est exprimée par :

$$U = \sum_{i=1}^{n} C_i / T_i \leq 1 \qquad \textbf{(3)}$$

L'expression (3) montre la puissance d'ordonnançabilité de l'algorithme EDF puisque le processeur peut être utilisé à 100 % et la configuration validée formellement. L'algorithme EDF est optimal dans la catégorie des algorithmes à priorité variable.

Tâche	r_i	C_i	D_i	T_i	Priorité selon D_i
τ_1	0	1	3	3	3
τ_2	0	1	4	4	2
τ_3	0	2	5	5	1



il faut corriger la séquence de la tache3.

Référence:

Systèmes temps réel de contrôle-commende conception et implémentation. Francis Cottet Emmanuel Grolleau. Dunod Paris 2005.