ORDONNANCEMENT TEMPS REEL

1 Ordonnancement des processus

Dans la littérature plusieurs algorithmes et méthodologies ont été proposés pour améliorer la prévisibilité des systèmes temps réel. Afin de présenter ces résultats, nous devons définir quelques concepts de base. Nous commençons par l'entité logicielle la plus importante traitée par n'importe quel système d'exploitation, le *processus*. Un processus est un triatement exécuté par le CPU de manière séquentielle. Dans ce contexte, le terme processus est utilisé comme synonyme de tâche et de thread. Cependant, il vaut la peine de dire que certains auteurs préfèrent les distinguer et définir un processus comme une entité plus complexe qui peut être composée de plusieurs tâches (ou threads) concurrentes partageant un espace mémoire commun.

Lorsqu'un seul processeur doit exécuter un ensemble de tâches concurrentes, c'est-à-dire des tâches qui peuvent se chevaucher dans le temps, la CPU doit être affecté aux différentes tâches selon un critère prédéfini, appelé politique d'ordonnancement (*scheduling policy*). L'ensemble des règles qui, à tout moment, déterminent l'ordre dans lequel les tâches sont exécutées est appelé un algorithme d'ordonnancement. L'opération spécifique d'allocation de la CPU à une tâche sélectionnée par l'algorithme d'ordonnancement est appelée répartition (*dispatching*).

Un ordonnancement est dit réalisable, si toutes les tâches peuvent être accomplies selon un ensemble de contraintes spécifiées. Les contraintes typiques qui peuvent être spécifiées sur les tâches en temps réel sont : les contraintes de temps, les relations de précédence et les contraintes d'exclusion mutuelle sur les ressources partagées.

Deux techniques de base sont adoptées dans la plupart des systèmes :

- Ordonnancement ***préemptif*** : est un ordonnancement dans lequel la tâche en cours peut être arbitrairement suspendue à tout moment, pour affecter la CPU à une autre tâche selon une politique d'ordonnancement prédéfinie. Dans les ordonnanceurs préemptifs, les tâches peuvent être exécutées dans des intervalles de temps disjoints.
- Ordonnancement ***coopératif*** : dans un ordonnancement coopératif, il n'y a pas de préemption. Le processus élu garde la possession du processeur jusqu'à la fin de son exécution.

1.1 Technique FIFO

C'est un algorithme élémentaire qui ne prend en compte aucune notion de priorité, ni d'échéance pour un processus donné. Les processus s'exécutent dans l'ordre et sans interruption.

1.2 Algorithme à priorité

Chaque processus est muni d'une priorité. A chaque quantum, c'est le processus prêt de forte priorité qui est élu. En cas d'égalité d'autres règles peuvent s'appliquer (FIFO par exemple).

1.3 Algorithme du Tourniquet (Round robin)

Les processus prêts ont droit à tour de rôle à un quantum (τ).

- si la tâche se termine ou se bloque avant le quantum τ, on passe à la suivante ;

- sinon, on remet la tâche en cours dans la file d'attente des tâches éligibles et on passe à la suivante.

2 Modélisation des tâches

Afin de pouvoir analyser de manière rigoureuse l'ordonnançabilité d'une configuration des tâches ; il est nécessaire d'avoir un modèle mathématique. Ce modèle doit permette de prendre en compte toutes les caractéristiques opérationnelles et temporelles d'une tâche.

2.1 Modélisation des tâches périodiques

Le modèle d'une tâche périodique repose sur trois paramètres :

- $\boldsymbol{r_k}$: date de réveil, c'est l'instant où la tâche est prête à être exécutée.

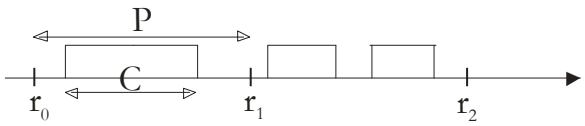
- $\boldsymbol{C = C_{max}}$: le temps d'exécution (de calcul) est le temps nécessaire au processeur pour exécuter la tâche sans interruption.

- $\boldsymbol{P}$: Période d'exécution de la tâche.

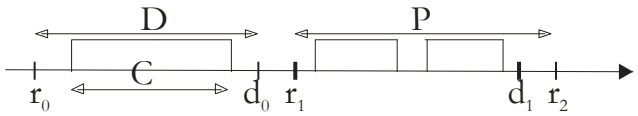
La date de réveil $\boldsymbol{r_k}$ de la $k^{ième}$ instance : $\boldsymbol{r_k = r_0 + kP}$

- Echéance absolue $\boldsymbol{d_i}$ est le temps avant lequel une tâche doit être terminée pour éviter d'endommager le système ;

Normalement la tâche doit avoir terminé son exécution avant la fin de la période. Si $d_i = P$, la tâche est dite à échéance sur requête.



- L'échéance peut être plus courte que la fin de la période. Dans ce cas on définit un nouveau paramètre «le délai critique D». Le délai critique correspond à la durée au bout de laquelle la tâche doit être terminée.



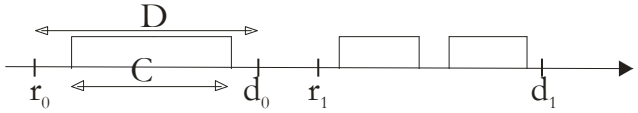
L'échéance $\boldsymbol{d_k}$ de la $\boldsymbol{k_{ième}}$ instance : $\boldsymbol{d_k = r_k + D = r_0 + kP + D}$

- Hyperpériode (Période d'étude). C'est l'intervalle de temps minimum après lequel le programme se répète. Si H est la longueur d'un tel intervalle, alors le programme dans $[0, H]$ est le même que celui dans $[kH, (k+1)H]$ pour tout entier $k > 0$. Pour un ensemble de tâches périodiques activées de manière synchrone à temps t = 0, l'hyperpériode est donnée par le plus petit commun multiple des périodes :

$$H = PPCM(T1, \ldots, Tn)$$

2.2 Modélisation des tâches apériodiques

Pour les tâches apériodiques le seul paramètre connu est le temps d'exécution $\boldsymbol{C}$. Le temps de réveil est aléatoire. Dans un système temps réel il faut connaitre aussi l'échéance.



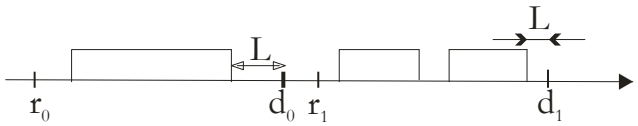
2.3 Autres paramètres

2.3.1 Les paramètres statiques

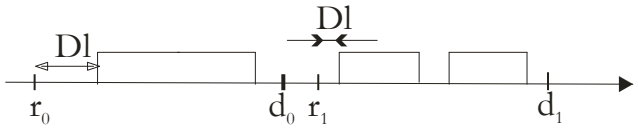
- $s_{i,j}$ désigne l'heure de début de la j[ième] instance de la tâche T_i ; c'est-à-dire l'heure à laquelle la tâche commence à s'exécuter.
- $f_{i,j}$ ou $(e_{i,j})$ désigne l'heure de fin de la j[ième] instance de la tâche T_i ; c'est-à-dire l'heure à laquelle l'exécution de la tâche se termine.
- Temps de réponse au travail. C'est l'heure (mesurée à partir de la date de réveil) à laquelle le travail est terminé : $R_{i,k} = f_{i,k} - r_{i,k}$: temps de réponse à la k^{ieme} instance de la tâche T_i.
- Temps de réponse de la tâche. C'est le temps de réponse maximum parmi tous les jobs :

$$R_i = \max_k R_{i,k}$$

- La laxité L, c'est le temps restant entre la fin d'exécution de la tâche et son échéance.



- Délai de latence (Dl), c'est la durée de temps avant le début d'exécution de la tâche.



$$Dl \leq D - C \; ; \quad Dl_{max} = D - C$$

2.3.2 Les paramètres dynamiques

- Le temps d'exécution restant

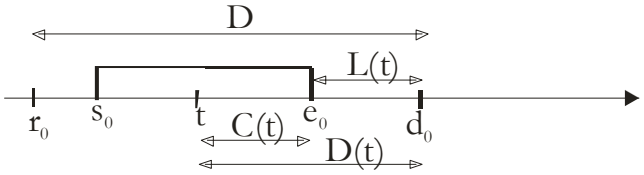
  $C(t) = C - C_{exécuté}$

- Délai critique dynamique

  $D(t) = d - t$

- Laxité dynamique

  $L(t) = D(t) - C(t) = d - t - C(t) = D + r - t - C(t)$



2.3.3 Facteur d'utilisation et facteur de charge du processeur

Étant donné un ensemble Γ de n tâches périodiques, le facteur d'utilisation du processeur U est la fraction du temps processeur consacré à l'exécution de l'ensemble de tâches. Puisque C_i/P_i est la fraction du temps processeur consacré à l'exécution de la tâche i, le facteur d'utilisation pour n tâches est donné par :

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

Le facteur d'utilisation du processeur fournit une mesure de la charge de calcul sur le processeur en raison de l'ensemble de tâches périodiques. Bien que l'utilisation de la CPU puisse être améliorée en augmentant les temps de calcul des tâches ou en diminuant leurs périodes, il existe une valeur maximale de U en dessous de laquelle Γ est ordonnançable et au-dessus de laquelle Γ n'est pas ordonnançable. Une telle limite dépend de l'ensemble de tâches (c'est-à-dire des relations particulières entre les périodes des tâches) et de l'algorithme utilisé pour planifier les tâches. Soit $U_{ub}(\Gamma, A)$ la borne supérieure du facteur d'utilisation du processeur pour un ensemble de tâches Γ sous un algorithme donné A. Lorsque $U = U_{ub}(\Gamma, A)$, on dit que l'ensemble utilise pleinement le processeur. Dans cette situation, Γ est ordonnançable par A, mais une augmentation du temps de calcul dans l'une des tâches rendra l'ensemble infaisable.

Nous pouvons écrire aussi :

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

Le facteur (H/P_i) représente le nombre (entier) de fois où T_i est exécuté dans l'hyperpériode, tandis que la quantité $(H/P_i)\ C_i$ est le temps de calcul total demandé par T_i dans l'hyperpériode. Par conséquent, la somme sur le côté gauche représente le temps de calcul total demandé par la tâche définie dans $[0, H)$.

De même en défini le facteur de charge du processeur par :

- Facteur de charge du processeur par T_i

$u_{hi} = C_i/D_i$

- Facteur de charge du processeur

$U_h = \sum C_i/D_i$

3 Ordonnancement des tâches périodiques

Lorsqu'une application de contrôle se compose de plusieurs tâches périodiques simultanées avec des contraintes de temps individuelles, le système d'exploitation doit garantir que chaque instance périodique est régulièrement activée à son rythme et terminée dans les échéances (qui, en général, peut être différente de son délai). Dans ce chapitre, trois algorithmes de base pour la gestion des tâches périodiques sont décrits en détail : Rate Monotonic, Earliest Deadline First et Deadline Monotonic. Une analyse d'ordonnancement est effectuée pour chaque algorithme afin de dériver un test de garantie pour les ensembles de tâches génériques.

3.1 Ordonnancement à Priorité fixe

3.1.1 Rate Monotonic (RM)

L'algorithme d'ordonnancement Rate Monotonic (RM) est une règle simple qui attribue des priorités aux tâches en fonction de leurs taux de demande. Plus précisément, les tâches avec des taux de demande plus élevés (c'est-à-dire avec des périodes plus courtes) auront des priorités plus élevées. Les périodes étant constantes, RM est une Algorithme à priorité fixe : une priorité p_i est affectée à la tâche avant exécution et ne change pas dans le temps. De plus, RM est intrinsèquement préemptif : la tâche en cours d'exécution est préemptée par une tâche nouvellement arrivée avec une période plus courte. En 1973, Liu et Layland ont montré que RM est optimal parmi toutes les Algorithmes à priorité fixe dans le sens où aucun autre algorithme à priorité fixe ne peut ordonnancer un ensemble de tâches qui ne peut pas être ordonnancé

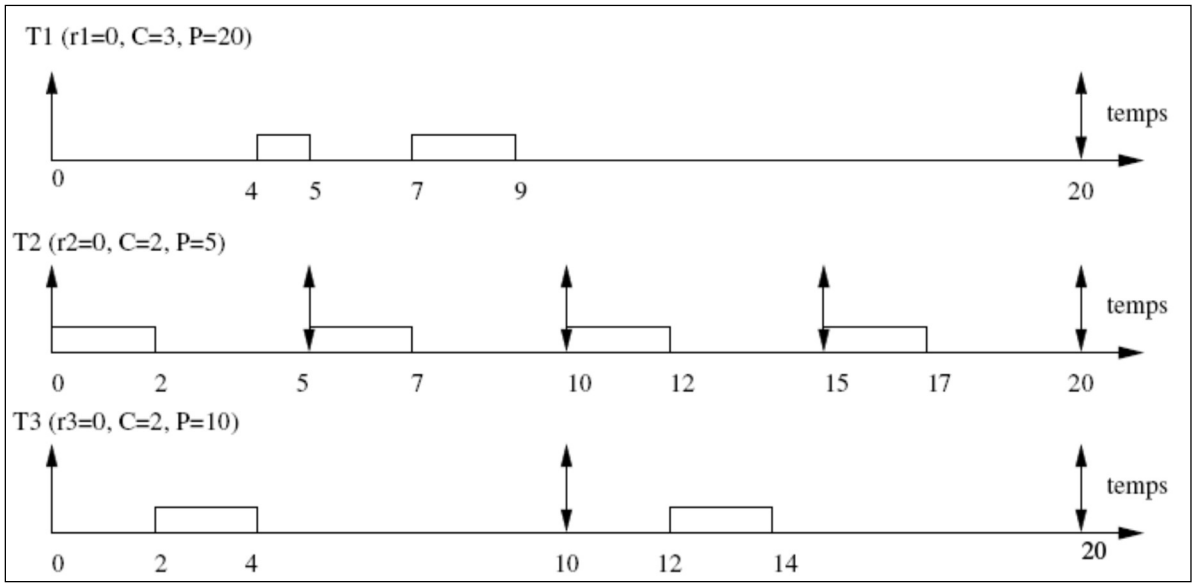
par RM. Liu et Layland ont également dérivé la limite supérieure (Limit Upper Bound) du facteur d'utilisation du processeur pour un ensemble générique de n tâches périodiques.

$$U_{lub} = n\left(2^{1/n} - 1\right)$$

Condition suffisante d'ordonnançabilité de n tâches :

$$\sum_{i=1}^{n} \frac{C_i}{P_i} \leq U_{lub}$$

La figure suivante illustre un exemple d'ordonnancement par RM



3.1.2 InverseDeadline ou DeadlineMonotonic (DM)

L'échéance sur requête permet à une instance d'être exécutée n'importe où dans sa période. Cette condition ne pouvait pas toujours être souhaitée dans les applications temps réel. Un modèle de processus plus flexible, qui pourrait être adopté pour gérer des tâches avec des contraintes de gigue ou des activités avec des temps de réponse courts par rapport à leurs périodes. L'algorithme Deadline monotonic (DM) a été proposé pour la première fois en 1982 comme une extension de Rate Monotonic, où les tâches peuvent avoir des délais relatifs inférieurs ou égaux à leur périodes (c'est-à-dire des délais contraints). Concrètement, chaque tâche périodique T_i est caractérisée par quatre paramètres :

- Une phase Φ_i , c'est-à-dire la date de réveil de sa première instance ($\Phi_i = r_{i,0}$) ;
- Un temps de calcul au pire des cas C_i (constant pour chaque instance) ;
- Un délai relatif D_i (constant pour chaque instance) ;
- Une période P_i.

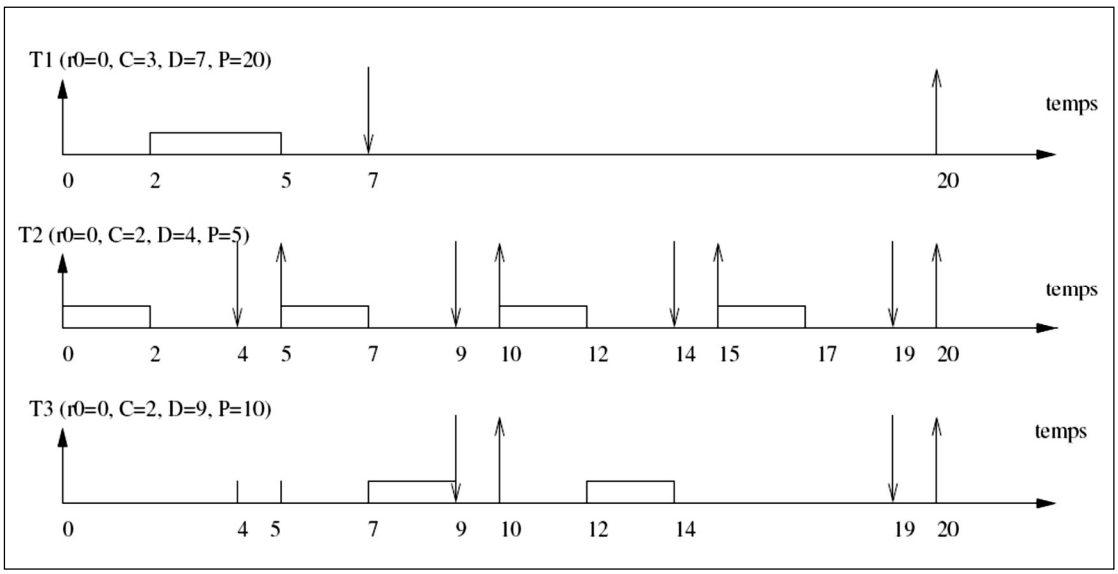
$$C_i \leq D_i \leq T_i$$
$$r_{i,k} = \Phi_i + kT_i$$
$$d_{i,k} = r_{i,k} + D_i.$$

Selon l'algorithme DM, chaque tâche se voit attribuer une priorité fixe inversement proportionnelle à son échéance relative D_i. Ainsi, à tout instant, la tâche dont l'échéance relative est la plus courte est exécutée. Étant donné que les délais relatifs sont constants, DM est un algorithme à priorité statique.

La faisabilité d'un ensemble de tâches avec des délais contraints pourrait être garantie à l'aide du test RM, en réduisant les périodes des tâches à des délais relatifs.

$$\sum_{i=1}^{n} \frac{C_i}{D_i} = n\left(2^{1/n} - 1\right)$$

La figure suivante illustre un exemple d'ordonnancement par DM.



3.1.3 Analyse du temps de réponse

Selon la méthode proposée par Audsley et al., le plus long temps de réponse R_i d'une tâche périodique T_i est calculé, à l'instant critique, comme la somme de son temps de calcul et de l'interférence I_i des tâches les plus prioritaires :

$$R_i = C_i + I_i$$

Où

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

D'où

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

Aucune solution simple n'existe pour cette équation puisque R_i apparaît des deux côtés. Ainsi, le temps de réponse le plus défavorable de la tâche T_i est donné par la plus petite valeur de R_i qui satisfait l'équation ci-dessus. Notez, cependant, que seul un sous-ensemble de points dans l'intervalle $[0, D_i]$ doit être examiné pour la faisabilité.

En fait, l'interférence sur T_i n'augmente que lorsqu'il y a déclenchement d'une tâche de priorité plus élevée.

Pour simplifier la notation, soit $R_i^{(k)}$ la k-ième estimation de R_i et $I_i^{(k)}$ l'interférence sur la tâche T_i dans l'intervalle [0, $R_i^{(k)}$] :

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

Ensuite, le calcul de R_i est effectué comme suit :

1. L'itération commence $R_i^{(0)} = \sum_{j=1}^{i} C_j$, qui est le premier moment où T_i pourrait éventuellement se terminer.
2. L'interférence réelle $I_i^{(k)}$ dans l'intervalle [0, $R_i^{(k)}$] est calculée par l'équation ci-dessus.
3. Si $I_i^{(k)} + C_i = R_i^{(k)}$ alors $R_i^{(k)}$ est le temps de réponse réel dans le pire des cas de la tâche T_i ; c'est-à-dire $R_i = R_i^{(k)}$. Sinon, l'estimation suivante est donnée par :
$$R_i^{(k+1)} = I_i^{(k)} + C_i$$
et l'itération continue à partir de l'étape 2.

Une fois R_i calculé, la faisabilité de la tâche T_i est garantie si et seulement si $R_i \leq D_i$.

Pour clarifier le test d'ordonnançabilité, considérons l'ensemble des tâches périodiques présentées dans le tableau suivant, activées simultanément au temps t = 0.

	C_i	P_i	D_i
T_1	1	4	3
T_2	1	5	4
T_3	2	6	5
T_4	1	11	10

Afin de garantir T_4 , nous devons calculer R_4 et vérifier que $R_4 \leq D_4$. L'ordonnancement produit par DM est illustré à la figure suivante, et les étapes d'itération sont présentées ci-dessous.

1) $R_4^{(0)} = \sum_{j=1}^{4} C_j = 5$; $I_4^{(0)} = \sum_{j=1}^{3} \left\lceil \frac{R_4^{(0)}}{P_j} \right\rceil C_j = \left\lceil \frac{5}{4} \right\rceil * 1 + \left\lceil \frac{5}{5} \right\rceil * 1 + \left\lceil \frac{5}{6} \right\rceil * 2 = 5$ et $I_4^{(0)} + C_4 = 6 > R_4^{(0)}$ alors T_4 n'est pas terminée en $R_4^{(0)}$.

2) $R_4^{(1)} = 6$; $I_4^{(1)} = \sum_{j=1}^{3} \left\lceil \frac{R_4^{(1)}}{P_j} \right\rceil C_j = \left\lceil \frac{6}{4} \right\rceil * 1 + \left\lceil \frac{6}{5} \right\rceil * 1 + \left\lceil \frac{6}{6} \right\rceil * 2 = 6$ et $I_4^{(1)} + C_4 = 7 > R_4^{(1)}$ alors T_4 n'est pas terminée en $R_4^{(1)}$.

3) $R_4^{(2)} = 7$; $I_4^{(2)} = \sum_{j=1}^{3} \left\lceil \frac{R_4^{(2)}}{P_j} \right\rceil C_j = \left\lceil \frac{7}{4} \right\rceil * 1 + \left\lceil \frac{7}{5} \right\rceil * 1 + \left\lceil \frac{7}{6} \right\rceil * 2 = 8$ et $I_4^{(2)} + C_4 = 9 > R_4^{(2)}$ alors T_4 n'est pas terminée en $R_4^{(2)}$.

4) $R_4^{(3)} = 9$; $I_4^{(3)} = \sum_{j=1}^{3} \left\lceil \frac{R_4^{(3)}}{P_j} \right\rceil C_j = \left\lceil \frac{9}{4} \right\rceil * 1 + \left\lceil \frac{9}{5} \right\rceil * 1 + \left\lceil \frac{9}{6} \right\rceil * 2 = 9$ et $I_4^{(3)} + C_4 = 10 > R_4^{(3)}$ alors T_4 n'est pas terminée en $R_4^{(3)}$.

5) $R_4^{(4)} = 10$; $I_4^{(4)} = \sum_{j=1}^{3} \left\lceil \frac{R_4^{(4)}}{P_j} \right\rceil C_j = \left\lceil \frac{10}{4} \right\rceil * 1 + \left\lceil \frac{10}{5} \right\rceil * 1 + \left\lceil \frac{10}{6} \right\rceil * 2 = 9$ et $I_4^{(4)} + C_4 = 10 = R_4^{(4)}$ alors T_4 se termine $R_4 = R_4^{(4)} = 10$.

Puisque $R_4 \leq D_4$, T_4 est programmable dans son échéance. Si $R_i \leq D_i$ pour toutes les tâches, nous concluons que l'ensemble de tâches est ordonnançable par DM.

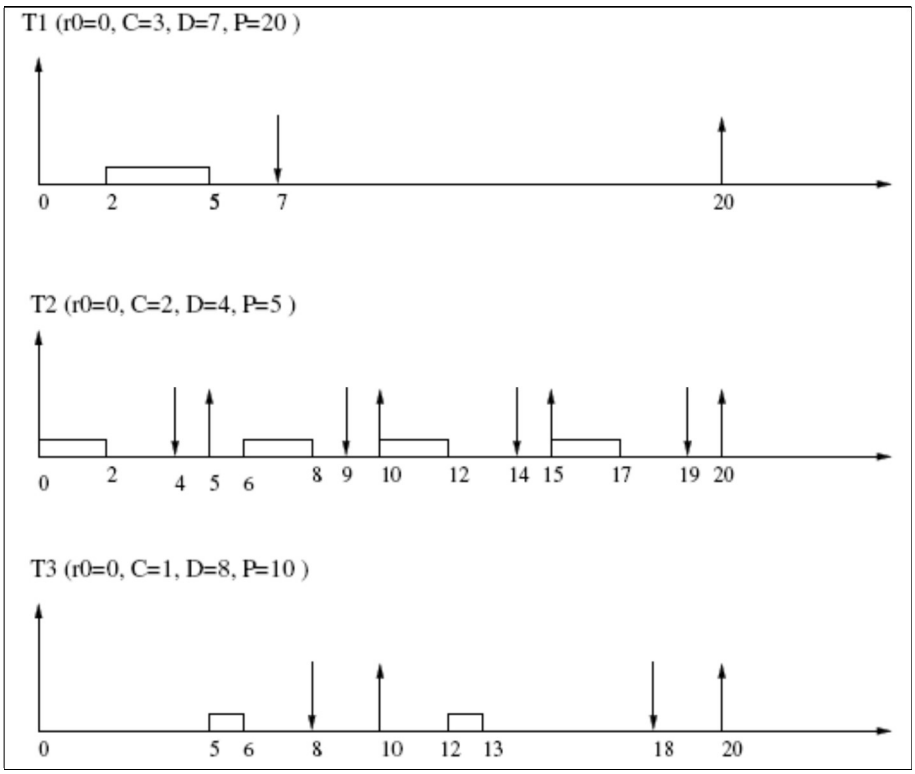
3.2 Ordonnancement à priorité dynamique

3.2.1 Earliest Deadline First (EDF)

L'algorithme EDF est une règle de d'ordonnancement dynamique qui sélectionne les tâches en fonction de leurs échéances absolues. Plus précisément, les tâches dont les échéances sont plus rapprochées seront exécutées à des priorités plus élevées.

Un ensemble de tâches périodiques est ordonnançable avec EDF si et seulement si :

$$\sum_{i=1}^{n} \frac{C_i}{\min(D_i, P_i)} \leq 1$$



3.2.2 EDF avec contrainte de délai

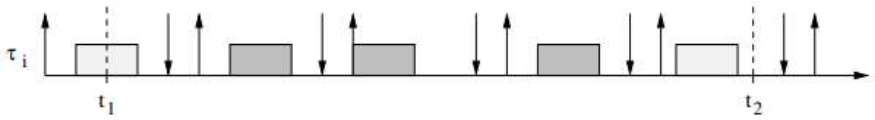
Sous EDF, l'analyse des tâches périodiques avec des échéances inférieures ou égales aux périodes, peut être réalisée en utilisant le critère de demande du processeur.

En général, la demande du processeur d'une tâche T_i dans un intervalle $[t1, t2]$, est le temps de traitement $g_i(t1, t2)$ demandé par les instances de T_i, activées dans $[t1, t2]$ et, qui doivent être terminées dans $[t1\ , t2]$. C'est-à-dire,

$$g_i(t_1, t_2) = \sum_{r_{i,k} \geq t_1, d_{i,k} \leq t_2} C_i$$

Pour l'ensemble des tâches, la demande du processeur dans $[t1, t2]$ est donnée par :

$$g(t_1, t_2) = \sum_{i=1}^{n} g_i(t_1, t_2)$$



La faisabilité d'un ensemble de tâches est garantie si et seulement si dans un intervalle de temps la demande du processeur ne dépasse pas le temps disponible ; c'est-à-dire si et seulement si :

$$\forall t_1, t_2 \quad g(t_1, t_2) \leq (t_1, t_2)$$

Si les délais relatifs ne sont pas plus grands que les périodes et que les tâches périodiques sont activées simultanément au temps $t = 0$ (c'est-à-dire, $\Phi_i = 0$ pour toutes les tâches), alors le nombre d'instances contribuant à la demande dans un intervalle $[0, L]$ peut être exprimé comme :

$$\eta_i(0, L) = \left\lfloor \frac{L - D_i}{T_i} + 1 \right\rfloor$$

Ainsi, la demande du processeur dans $[0, L]$ peut être calculée comme :

$$g(0, L) = \sum_{i=1}^{n} \left\lfloor \frac{L - D_i}{T_i} + 1 \right\rfloor C_i \leq L$$

La fonction $g(0, L)$ est appelée demande cumulée d'exécution des tâches (Demand Bound Function "dbf")

$$dbf(t) \stackrel{\text{def}}{=} \sum_{i=1}^{n} \left\lfloor \frac{t - D_i}{T_i} + 1 \right\rfloor C_i$$

Ainsi, un ensemble synchrone des tâches périodiques avec des échéances relatives inférieures ou égales à des périodes est programmable par EDF si et seulement si :

$$\forall t > 0 \quad dbf(t) \leq t \qquad (**)$$

RÉDUCTION DES INTERVALLES DE TEST

La condition (**) peut être simplifié en réduisant le nombre d'intervalles dans lesquels elle doit être vérifiée. On constate d'abord que :

1. si les tâches sont périodiques et sont activées simultanément à l'instant t = 0, alors l'ordonnancement se répète à chaque hyperpériode H ; ainsi la condition (**) n'a besoin d'être vérifiée que pour les valeurs de L inférieures ou égales à H.

2. $g(0, L)$ est une fonction échelon dont la valeur augmente lorsque L dépasse une échéance d_k et reste constante jusqu'à la prochaine échéance d_{k+1}. Cela signifie que si la condition $g(0, L) < L$ est vérifiée pour $L = d_k$, alors elle est aussi vraie pour tout L tel que $d_k \leq L < d_{k+1}$. Par conséquent, la condition (**) n'a besoin d'être vérifiée que pour des valeurs de L égales à des échéances absolues.

Le nombre de points de test peut être encore réduit en notant que :

$$\left\lfloor \frac{L + T_i - D_i}{T_i} \right\rfloor \leq \left(\frac{L + T_i - D_i}{T_i} \right)$$

Et définir :

$$G(0, L) = \sum_{i=1}^{n} \frac{L + T_i - D_i}{T_i} C_i = \sum_{i=1}^{n} \frac{T_i - D_i}{T_i} C_i + \frac{L}{T_i} C_i$$

$$\forall L > 0, \qquad g(0, L) \leq G(0, L)$$

D'où

$$G(0,L) = \sum_{i=1}^{n}(T_i - D_i)U_i \ + LU$$

On peut noter que $G(0,L)$ est une fonction croissante de L de pente U. Ainsi, si $U < 1$, il existe un $L = L^*$ pour lequel $G(0,L) = L$. Clairement, pour tout $L \geq L^*$, nous avons $g(0,L) \leq G(0,L) \leq L$, ce qui signifie que l'ordonnançabilité de l'ensemble de tâches est garantie. Par conséquent, il n'est pas nécessaire de vérifier la condition (**) pour les valeurs de $L \geq L^*$.

La valeur deL^* est l'instant auquel $G(0,L^*) = L^*$ est :

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

Qui donne :

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

Théorème : Un ensemble de tâches périodiques synchrones avec des échéances relatives inférieures ou égales à des périodes peut être programmé par EDF si et seulement si $U < 1$ et $\forall t > \mathcal{D} \quad dbf(t) \leq t$

Avec $\mathcal{D} = \{d_k |\ d_k \leq \min[H, \max(D_{max}, L^*)]\}$

Et

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

Exemple :

Pour illustrer le critère de demande du processeur, considérons l'ensemble de tâches présenté dans le tableau suivant, où trois tâches périodiques avec des délais inférieurs aux périodes doivent être garanties dans le cadre d'EDF.

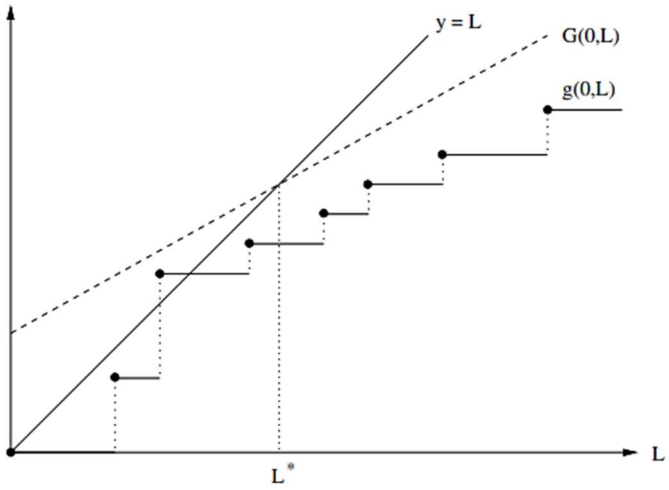
	C_i	P_i	D_i
T_1	2	6	4
T_2	2	8	5
T_3	3	9	7

A partir des paramètres spécifiés, il est facile de calculer que :

$$U = \frac{2}{6} + \frac{2}{8} + \frac{3}{9} = \frac{11}{12}$$

$$L^* = \frac{\sum_{i=1}^{n}(T_i - D_i)U_i}{1-U} = 25$$

$$H = PPCM(6,8,9) = 72$$



L	$g(0, L)$	result
4	2	OK
5	4	OK
7	7	OK
10	9	OK
13	11	OK
16	16	OK
21	18	OK
22	20	OK

Par conséquent, la condition (**) doit être testée pour toute échéance inférieure à 25, et l'ensemble des points de contrôle est donné par $D = \{4, 5, 7, 10, 13, 16, 21, 22\}$. Le tableau ci-dessus présente les résultats du test et la figure suivante illustre le planning produit par EDF pour l'ensemble de tâches.