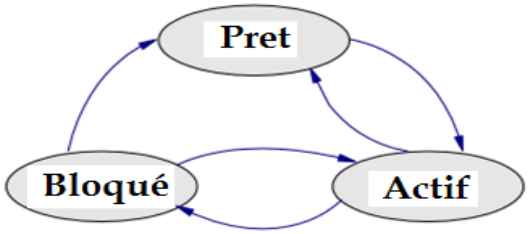
Ordonnancement Temps Réel

1. Définition d'une tache

Une tache est un programme qui s'exécute comme s'il était le seul à utiliser le CPU. Il possède une priorité, du code à effectuer, une partie de la mémoire et une zone de pile.

2. Etats d'une tache

Une tache peut être dans un état :

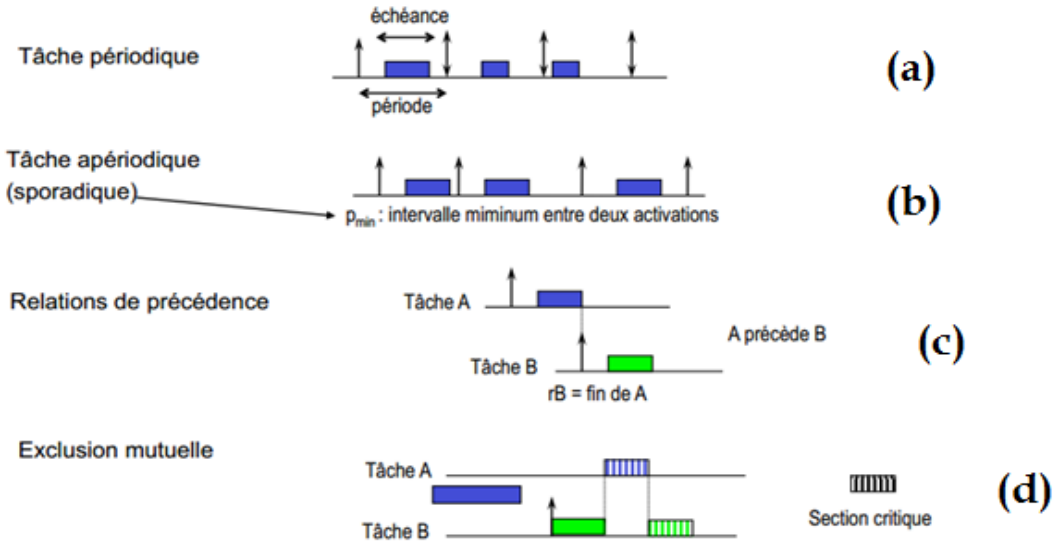


- Pret : La tâche est prête à être exécutée mais n'a pas le CPU,
- actif : La tâche est exécutée par le CPU,
- bloqué : La tâche est en attente d'un signal ou d'une ressource pour poursuivre,

3. Types de taches

Suivant les contraintes temporelles, le partage de ressources et la relation de précédence, on distingue les types de taches suivants :

- Tâches dépendantes (figure c et d) / indépendantes (figure a et b)
- Tâches répétitives / périodiques: activations successives (figure a)
- Tâches répétitives / apériodiques (figure b)



- La figure (a) montre une tache périodique se déclenchant à des intervalles de temps réguliers(périodes)
- La figure © affiche deux taches dépendantes A et B reliées par une relation de precedance causale ; Tache A précède Tache B
- La figure (d) affiche aussi deux taches dépendantes partageant une section critique.

4. Priorités des tâches

En fonction de sa criticité , chaque tache se voit attribuer une priorité. Les taches importantes auront des priorités élevées. Il existe deux types de priorité de taches :

- statiques c'est-à-dire les priorités sont fixes, Chaque tache reçoit une priorité fixe lors de sa création et ne change pendant son exécution ,
- dynamiques c'est-à-dire les priorités sont variables et peuvent être modifier pendant l'exécution.

5. Ordonnanceur

L'Ordonnanceur est le composant du noyau responsable de la distribution du temps CPU entre les différentes taches. Il est basé sur le principe de la priorité : chaque tache possède une priorité dépendant de son importance (plus une tache est importante, plus sa priorité est élevée). Il attribue le processeur à la tache exécutable de plus haute priorité.

6. Critères de performance d'ordonnancement

Le choix d'une politique d'ordonnancement doit tenir compte des points suivants :
- **Utilisation de l'UC** : utiliser la CPU le maximum possible.
- **Débit (Throughput)** – nombre de taches qui terminent leur exécution par unité de temps.
- **Temps de rotation (Turnaround time)** – le temps depuis le lancement de la tache jusqu'à la fin de son exeution (les attentes incluses).
- **Temps de réponse (Response time)** – temps qui s'ecoule entre l'entrée de la tache et le moment où elle commence à être traitée.

– **Temps d'attente (Waiting time)** – temps de la tache dans la file d'attente des taches prêtes.

Il est conseillé de maximiser l'utilisation de l'UC (proche de 100% c'est-à-dire pas de repos de le CPU) et de minimiser le temps de rotation, le temps d'attente et le temps de réponse.

7.Caractéristiques des tâches

r : **date de réveil**: dans certains cas, un traitement doit être déclenché à une date précise relative par rapport au début de l'exécution de la tache ou absolue (plus

rarement). Cette date de réveil n'implique pas obligatoirement l'exécution ; il peut y avoir un délai dû à l'indisponibilité du processeur.

● C : durée d'exécution maximale (**capacité)**

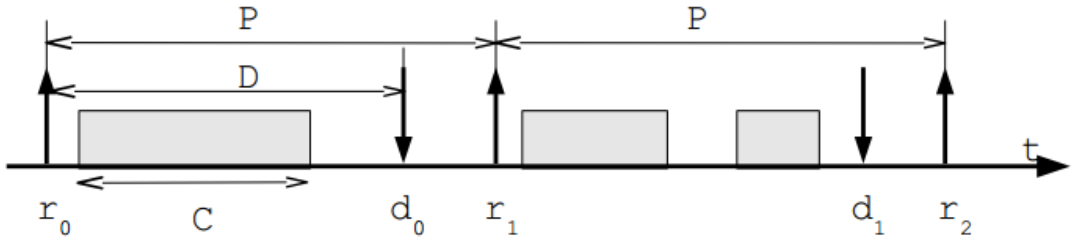
● D : délai critique **(échéance ou deadline**): délai maximum acceptable pour son exécution ; le traitement doit être terminé à un instant spécifié par rapport au début de l'exécution de la tache.

● P : **période** (si tâche périodique)

● d = r+D : échéance (si tâche à contraintes strictes)

● tâche périodique : rk= r0+ k*P

si D = P, tâche à échéance sur requête



La figure presente une tache periodique . on voit qu'il y a un decalage entre la date de reveil de la tache (r0 ou r1) et sa Date de démarrage c'est-à-dire l'instant correspondant à son exécution effective. Aussi , le traitement de tache doit se terminer avant le depassement de son echeance pour eviter tout probleme . la Date de fin d'une tache Ti est l' instant correspondant à la fin de son exécution . elle doit etre à inferieure à di(deadline) et Pi(periode)

8. Paramètres statiques

* U = C/P : facteur d'utilisation du processeur

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

* CH = C/D : facteur de charge du processeur

$$CH = \sum_{i=1}^{n} \frac{C_i}{D_i}$$

9. Algorithmes d'ordonnancement des tâches indépendantes

Pour les tâches indépendantes , il n'y a ni de partage de ressources ni de contraintes de précédence. Leurs ordonnancement est simple . On distingue deux categories de taches independantes :

- Taches independantes periodiques
- Taches independantes aperiodiques

9.1/ Rate Monotonic Analysis (RMA)

● algorithme à priorité constante
● basé sur la période (tâches à échéance sur requête) : la tâche de plus petite période est la plus prioritaire. La priorité d' une tâche est inversement proportionnelle à sa période. Plus la période est petite, plus la priorité est grande
● test d'acceptabilité (condition suffisante de Liu et Layland)

$$\sum_{i=1}^{n} \frac{C_i}{P_i} \leqslant n(2^{1/n}-1)$$

Exemple 1 :

Soit un système temps réel avec 3 tâches ayant les paramètres suivants :

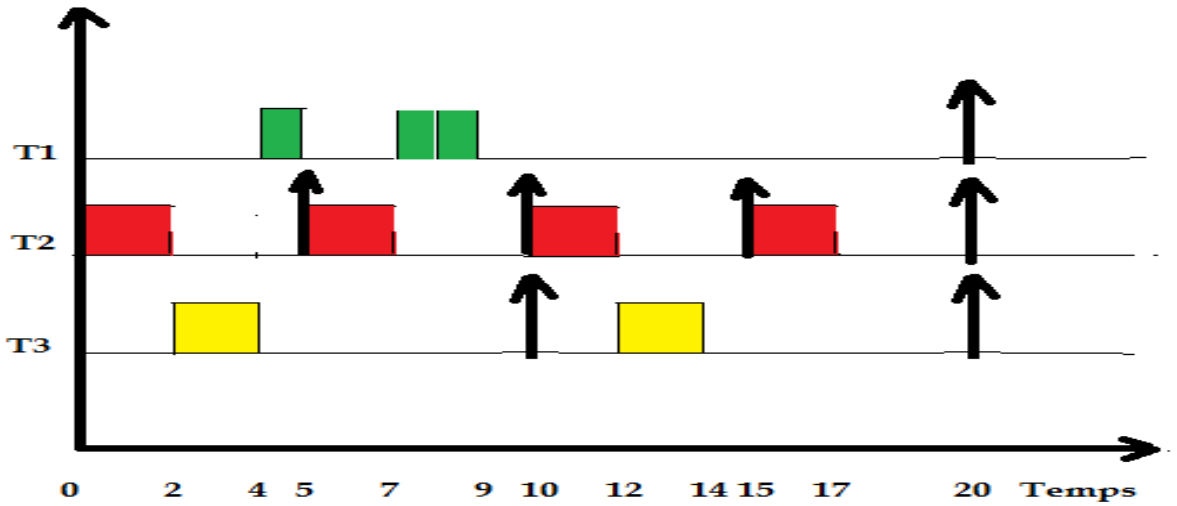
Tâche	Date de réveil (r)	Capacité (C)	Echeance (D)	Période (P)
T1	0	3	20	20
T2	0	2	5	5
T3	0	2	10	10

- Priorité(T1)< Priorité(T3)< Priorite(T2)

  La tache T2 est la tâche la plus prioitaire car sa periode est la plus petite.

- Condition de Lui Layland

  3/20 + 2/5 + 2/10 = 0,75 < 0 ,78 ==➔ les taches sont ordonnancables



Exemple 2 :

Soit un système temps réel avec 3 tâches ayant les paramètres suivants :

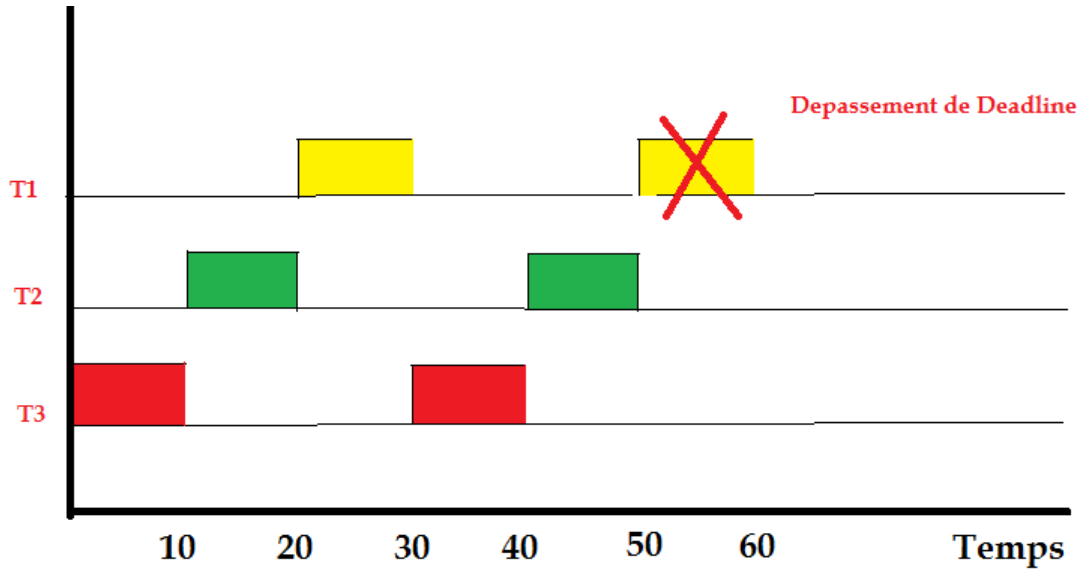
Tâche	Date de réveil (r)	Capacité (C)	Echeance (D)	Période (P)
T1	0	12	50	50
T2	0	10	40	40
T3	0	10	30	30

- Priorite(T1)< Priorite(T2)< Priorite(T3)

  La tacheT3 est la tache la plus prioitaire car sa periode est la plus petite.

- Condition de Lui Layland

  12/50 +10/40 + 10/30 = 0,82 < 0 ,78 ==➔ on ne peut rien dire sur l'ordonnancabilité des taches



On voit sur le graphe que les trois taches sont non ordonnacables . La tache T1 lui manque 2 unités de temps qu'elle doit les executer à partir de l'instant 50. Or cet instant est sa période . Donc , il est impossible qu'elle termine son exécution .

9.2-Deadline Monotonic Analysis (DMA)

● algorithme à priorité constante

● basé sur le délai critique : la tâche de plus petit délai critique est la plus prioritaire . La priorité d' une tâche est inversement proportionnelle à son deadline (plus le deadline est petit, plus la priorité est grande)

● test d'acceptabilité (condition suffisante)

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

● équivalent à RMA dans le cas des tâches à échéance sur requête

Exemple 1 :

Soit un système temps réel avec 2tâches ayant les paramètres suivants :

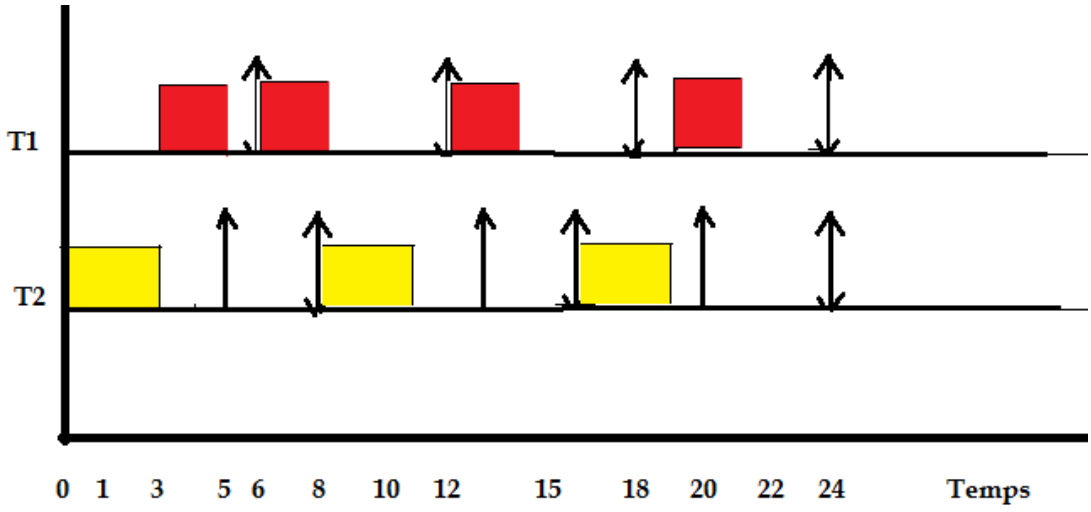
Tâche	Date de réveil (r)	Capacité (C)	Echeance (D)	Période (P)
T1	0	2	6	6
T2	0	3	5	8

- Priorite(T1)< Priorite(T2

  La tacheT2 est la tache la plus prioitaire car son echeance est la plus petite.

- Condition de Lui Layland

  2/6 +3/5 = 0,933 < 0 ,828 ==➔ on ne peut rien dire sur l'ordonnancabilité des taches



Les 2 taches T1 et T2 sont ordonnancables

Exemple 2 :

Soit un système temps réel avec 3 tâches ayant les paramètres suivants :

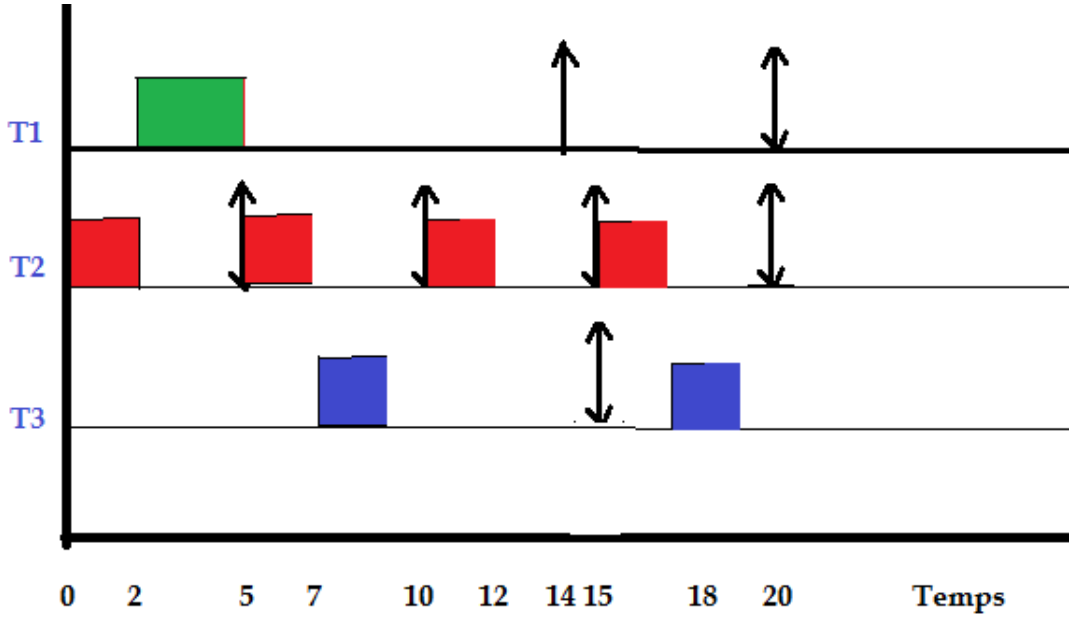
Tâche	Date de réveil (r)	Capacité (C)	Echeance (D)	Période (P)
T1	0	3	14	20
T2	0	2	5	5
T3	0	2	15	15

- Priorite(T3)< Priorite(T1)< Priorite(T2)

La tacheT2 est la tache la plus prioitaire car son deadline est le plus petit.

- Condition de Lui Layland

3/14 +2/5 + 2/15 = 0,747 < 0 ,78 ==➔ les taches sont ordonnancables



9.3.Earliest Deadline First (EDF)

● algorithme à priorité variable ou dynamique

● basé sur l'échéance : à chaque instant (i.e à chaque réveil de tâche), la priorité maximale est donnée à la tâche dont l'échéance est la plus proche. La tâche la plus prioritaire (parmi les tâches prêtes) est celle dont
l' échéance absolue est la plus proche.

•Il est applicable aussi bien pour des tâches périodiques qu' apériodiques.

● test d'acceptabilité

- condition nécessaire

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

-condition suffisante

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

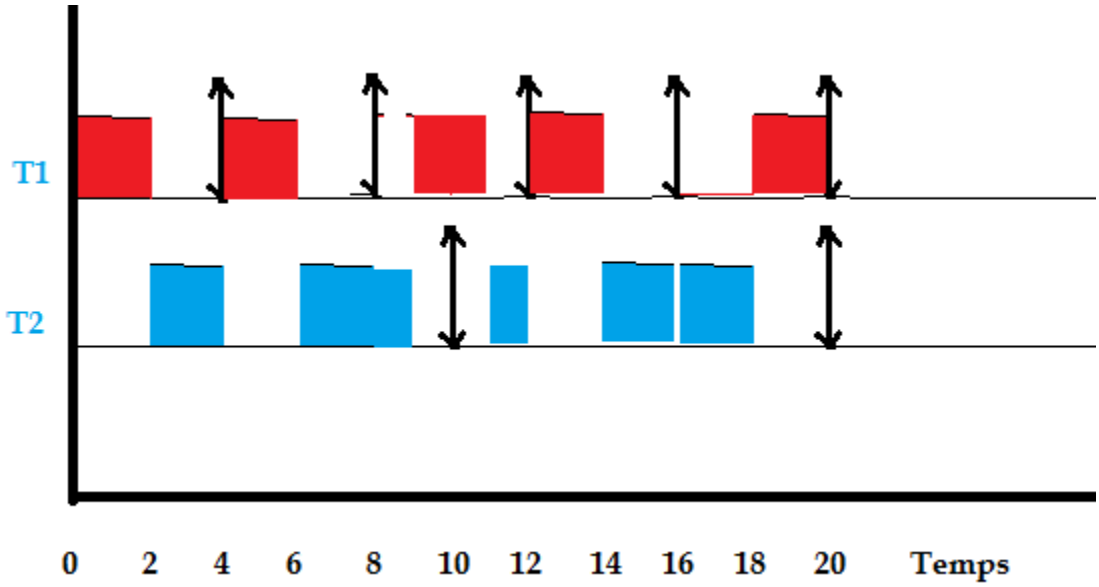
Exemple 1 :

Soit un système temps réel avec 2 tâches ayant les paramètres suivants :

Tâche	Date de réveil (r)	Capacité (C)	Echeance (D)	Période (P)
T1	0	2	4	4
T2	0	5	10	10

- Condition necessaire de Lui Layland

  2/4 +5/10 = 1 ==➔ les taches peuvent etre ordonnancées



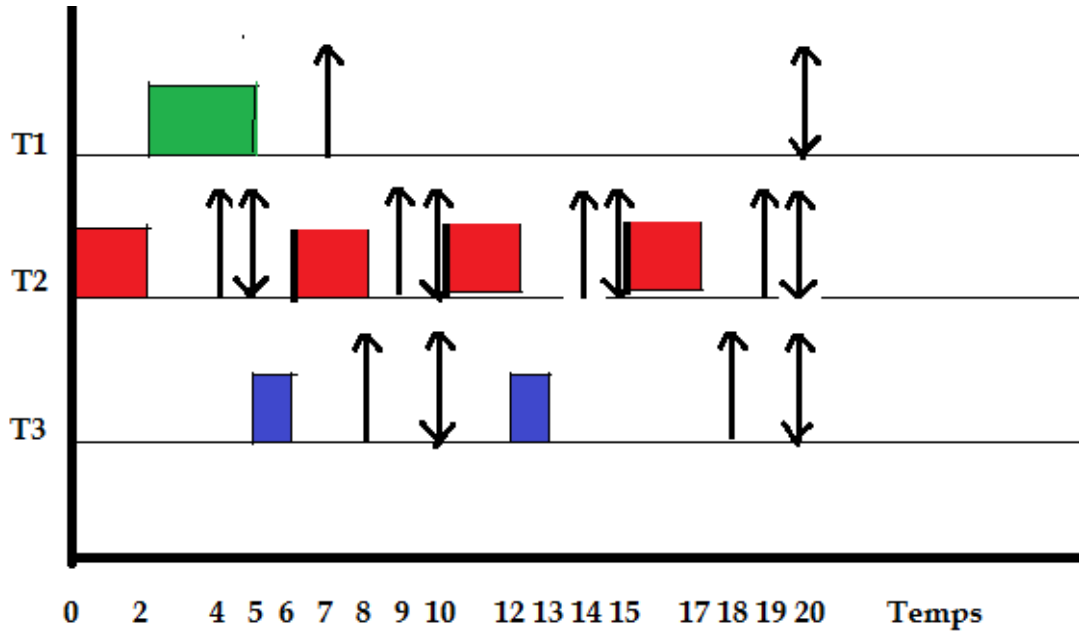
Exemple 2 :

Soit un système temps réel avec 3 tâches ayant les paramètres suivants :

Tâche	Date de réveil (r)	Capacité (C)	Echeance (D)	Période (P)
T1	0	3	7	20
T2	0	2	4	5
T3	0	1	8	10

- Condition de Lui Layland

  3/7 +2/4 +1/8 = 1,05 < 1 non ==➔ on ne peut rien dire sur l'ordonnancabilité des taches



9.4.Traitement des tâches apériodiques à contraintes relatives

9.4.1.Traitement d'arrière-plan

● tâches apériodiques ordonnancées quand le processeur est oisif : les tâches périodiques restent les plus prioritaires (tâches critiques)
● ordonnancement relatif des tâches apériodiques en mode FIFO
● préemption des tâches apériodiques par les tâches périodiques
● traitement le plus simple, mais le moins performant

Exemple :

Soit un système temps réel avec 2 tâches periodiques ayant les paramètres suivants :

Tâche	Date de réveil (r)	Capacité (C)	Echeance (D)	Période (P)
Tp1	0	2	5	5
Tp2	0	2	10	10

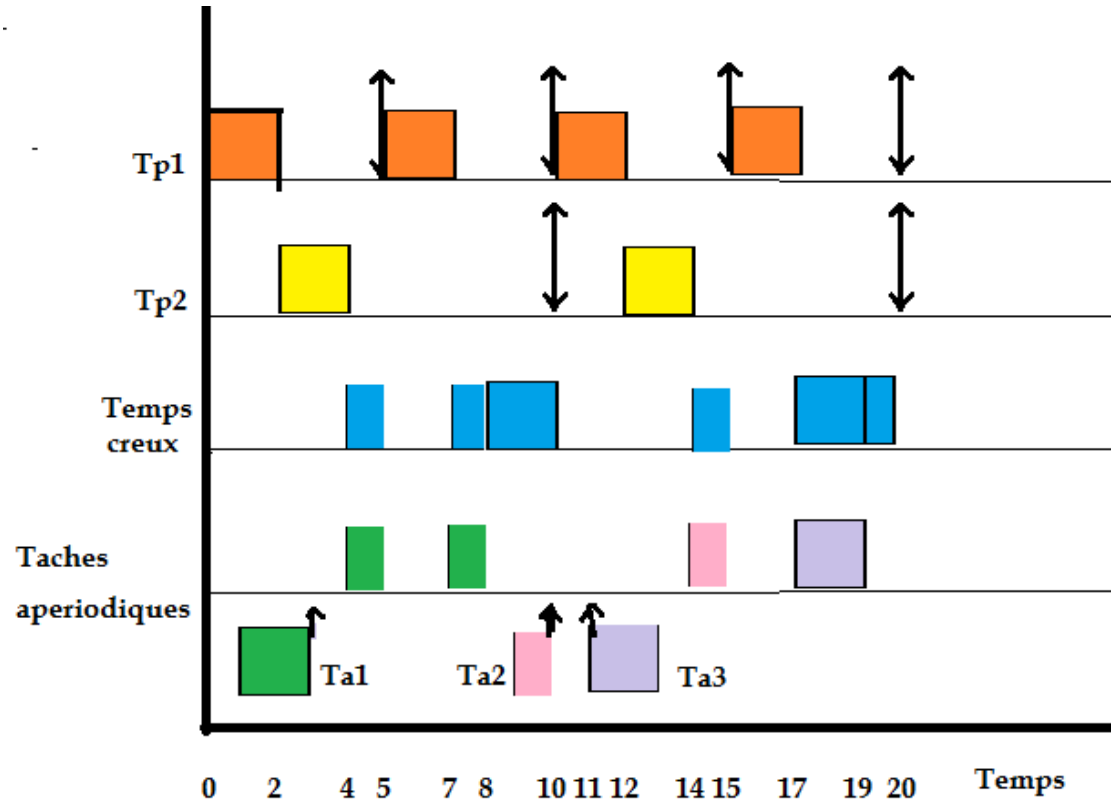
Et 3 tâches aperiodiques ayant les paramètres suivants :

Tâche	Date de réveil (r)	Capacité (C)
Ta1	3	2
Ta2	10	1
Ta3	11	2

Application de l'algorithme RMA pour les taches periodiques

Condition de lui Layland

2/5+2/10= 0,6<0,8284 ===➔ Les 2 taches peuvent etre ordonnancées



9.4.2.Traitement par serveur

● un serveur est une tâche périodique créée spécialement pour prendre en compte les tâches apériodiques

● serveur caractérisé par sa période, son temps d'exécution : capacité du serveur

●serveur généralement ordonnancé suivant le même algorithme que les autres tâches périodiques (RMA)

●une fois actif, le serveur sert les tâches apériodiques dans la limite de sa capacité. ● l'ordre de traitement des tâches apériodiques ne dépend
pas de l'algorithme général

9.4.2.1.Traitement par serveur par scrutation

● à chaque activation, traitement des tâches en suspens jusqu'à épuisement de la capacité ou jusqu'à ce qu'il n'y ait plus de tâches en attente

● si aucune tâche n'est en attente (à l'activation ou parce que la dernière tâche a été traitée) , le serveur se suspend immédiatement et perd sa capacité qui peut être réutilisée par les tâches périodiques (amélioration du temps de réponse)

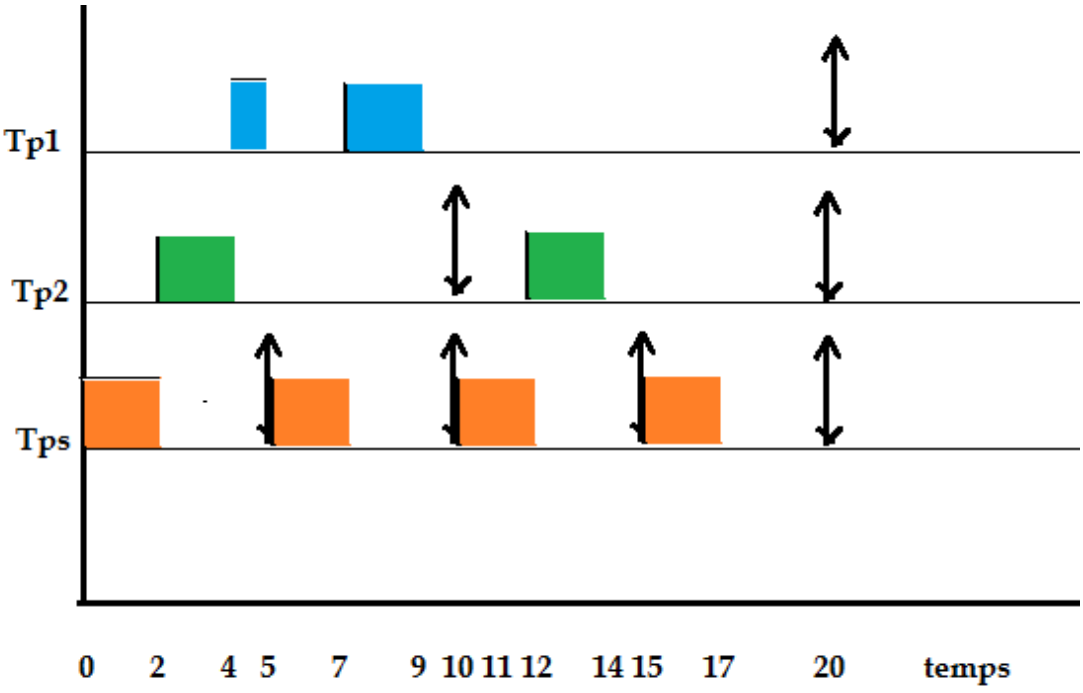
Exemple :

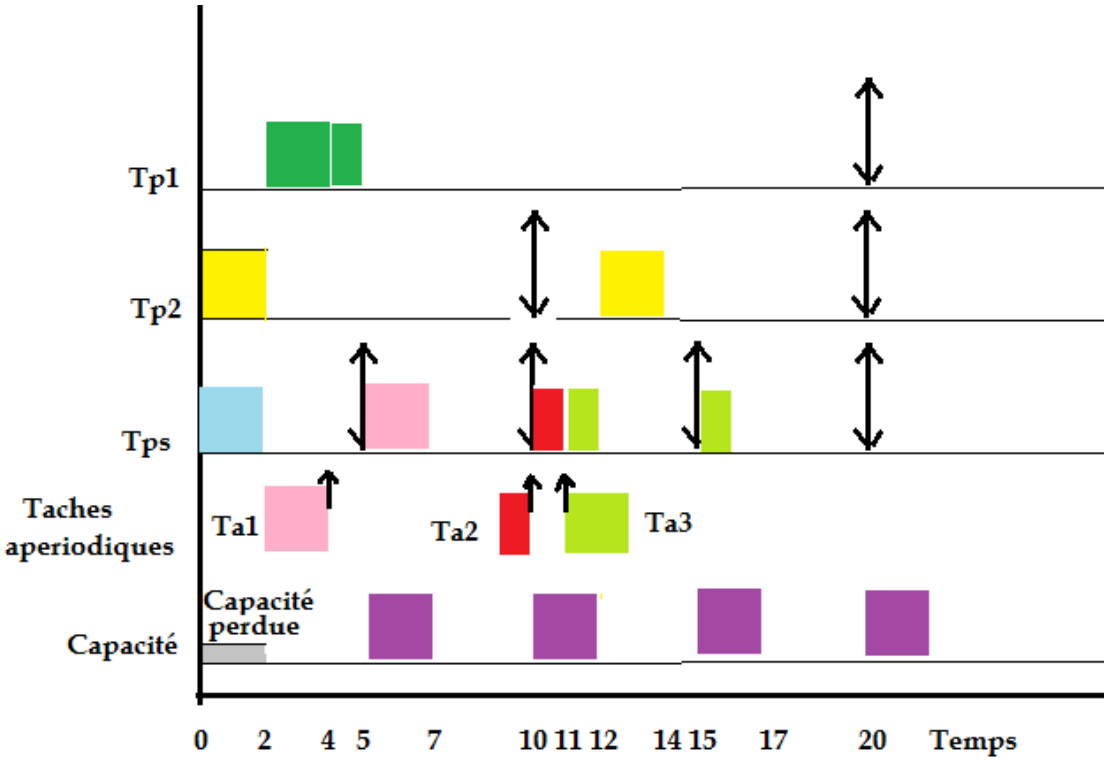
Soit un système temps réel avec 2 tâches périodiques: Tp1 (0, 3, 20) ,Tp2 (0, 2, 10), une tache serveur : Tps (0, 2, 5) et 3 tâches apériodiques : Ta1 (4, 2), Ta2 (10, 1), Ta3 (11, 2)

Application de l'algorithme RMA pour les taches periodiques

Condition de lui Layland

3/20 + 2/10 + 2/5 = 0,75<0,77 ===➔ Les 3 taches peuvent etre ordonnancées





9.4.2.2.Traitement par Serveur sporadique

● améliore le temps de réponse des tâches apériodiques sans diminuer le taux d'utilisation du processeur pour les tâches périodiques
● comme le serveur ajournable mais ne retrouve pas sa capacité à période fixe
● le serveur sporadique peut être considéré comme une tâche périodique « normale » du point de vue des critères d'ordonnancement

● calcul de la récupération de capacité
- le serveur est dit « actif » quand la priorité de la tâche courante Pexe est supérieure ou égale à celle du serveur Ps
- le serveur est dit « inactif » si Pexe < Ps
- RT : date de la récupération
✔ calculée dès que le serveur devient actif (tA)
✔ égale à tA + Ts
- RA : montant de la récupération à effectuer à RT
✔ calculée à l'instant tI où le serveur devient inactif ou que la capacité est épuisée
✔ égal à la capacité consommée pendant l'intervalle [tA, tI]

9.5.Tâches apériodiques à contraintes strictes

● les algorithmes précédents restent possibles

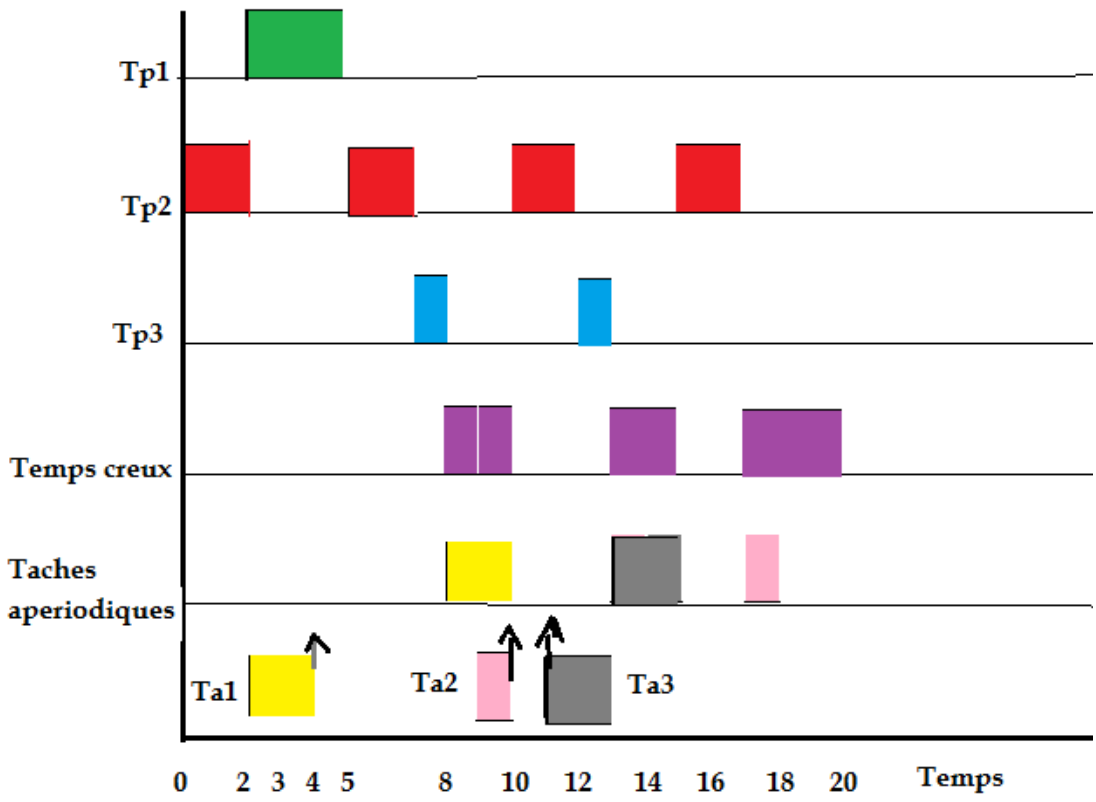
Principe de l'ordonnancement
● ordonnancer les tâches en EDF
● A chaque nouvelle tâche apériodique, faire tourner une "routine de garantie" pour vérifier que toutes les contraintes temporelles seront respectées
- si oui, accepter la tâche.
- si non, refuser la tâche.
● 2 politiques d'acceptation dynamique :
- acceptation dans les temps creux
- ordonnancement conjoint
● favorise les tâches périodiques

9.5.1.acceptation dans les temps creux

● ordonnancement EDF des tâches périodiques
● les tâches apériodiques acceptées sont ordonnancées dans les temps creux des tâches périodiques (méthode d'arrière-plan) selon l'algorithme EDF
● routine de garantie (au réveil d'une tâche apériodique):
- teste l'existence d'un temps creux suffisant entre le réveil et l'échéance de la tâche apériodique)
- vérifie que l'acceptation de la nouvelle tâche ne remet pas en cause le respect des contraintes temporelles des autres tâches apériodiques déjà acceptées et non encore terminées
- si OK, la tâche est ajoutée à la liste des tâches apériodiques

Exemple

Soit un système temps réel avec 3 tâches périodiques: Tp1 (0, 3, 7,20) ,Tp2 (0, 2, 4,5), Tp3 (0, 1, 8,10) et 3 tâches apériodiques : Ta1 (4, 2,10), Ta2 (10, 1,18), Ta3 (11, 2,16)



Que se passe t-il si on modifie la capacité de Ta2 (10,2,18) ?

9.5.2.ordonnancement conjoint

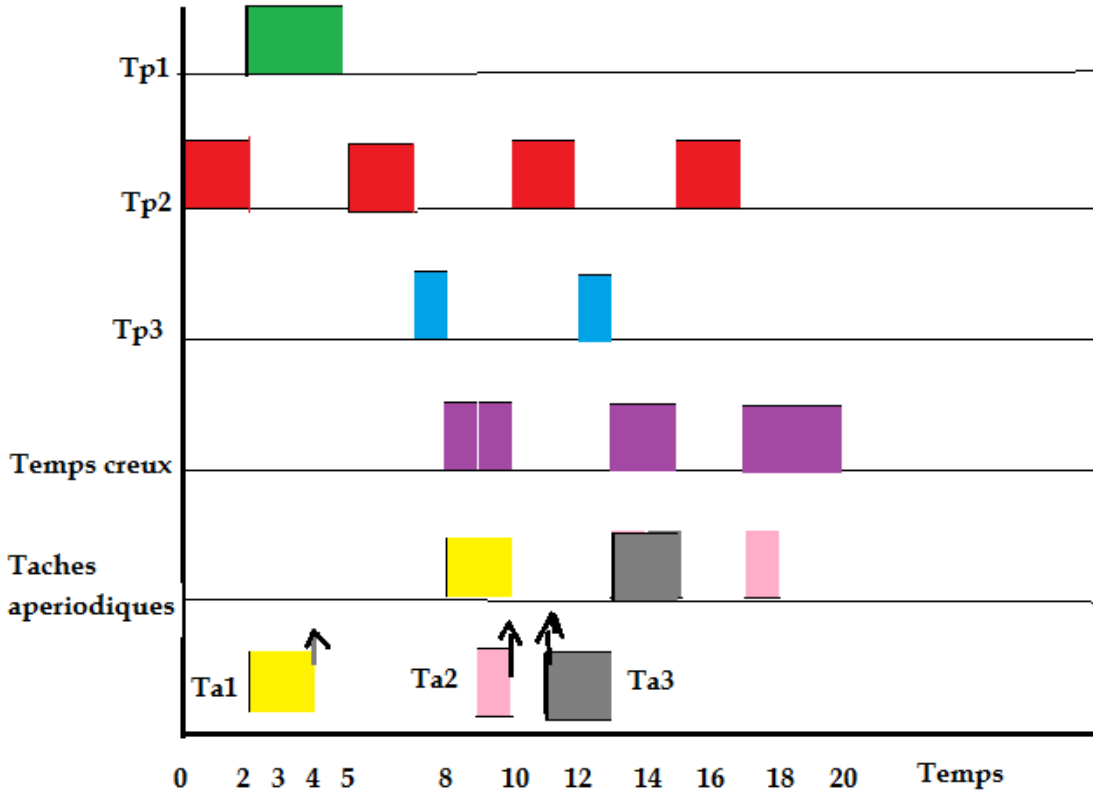
● la séquence des tâches périodiques n'est plus immuable

● à l'arrivée de chaque nouvelle tâche apériodique, construction d'une nouvelle séquence EDF

- si la construction est possible : acceptation de la tâche
- sinon rejet

Exemple

Soit un système temps réel avec 3 tâches périodiques: Tp1 (0, 3, 7,20) ,Tp2 (0, 2, 4,5), Tp3 (0, 1, 8,10) et 3 tâches apériodiques : Ta1 (4, 2,10), Ta2 (10, 1,18), Ta3 (11, 2,16)



Que se passe t-il si on modifie la capacité de Ta2 (10,2,18) ?