

# CHAP1: Introduction aux systèmes temps réel.

① Définition: Un système temps réel est un système classique où le respect des délais (temps) est critique. Il ne s'agit pas d'avoir des résultats rapide mais les obtenir avant une échéance donnée.

exemple :

- Système non temps réel: calculer les salaires des employés d'une entreprise. Le temps n'a pas d'impact sur le résultat.
- Système temps réel: détecter et corriger un dysfonctionnement dans un ordre en vol, un retard impacte le résultat.

## ② Classification:

Temps réel dur (critique): le non-respect d'une échéance est considéré comme une erreur fatale. → Système de contrôle de trafic aérien

Temps réel souple: les échéances sont importantes mais non critique. → projection vidéo (décalage entre le son et l'image)

Temps réel ferme: un respect des échéances est requis, mais des erreurs mineures sont tolérées. → projection vidéo (Perte de quelques images)

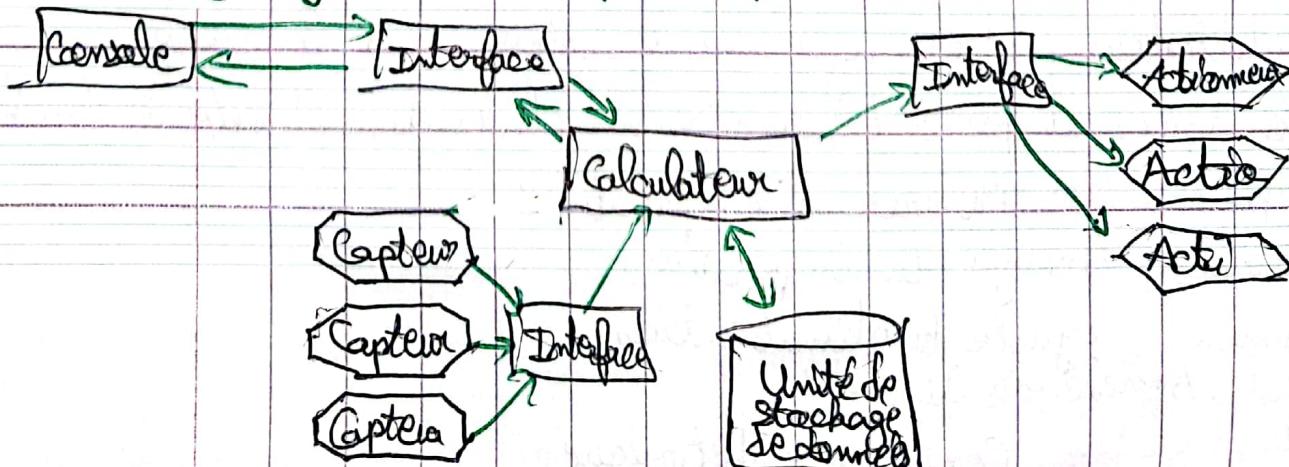
## ③ Caractéristiques du Temps réel:

Taille et complexité: Ses systèmes interagissent avec un environnement. La certification garantit un fonctionnement conforme aux spécifications.

Simulation, prototypage: vérification précoce de la conception.

Plates-formes d'essai: vérification a posteriori du bon fonctionnement.

## ④ Système Embarqué/Temps réel:



①

① **Console:** est une interface utilisateurs qui permet l'entrée de commandes. L'utilisateur peut interagir avec le sys pour config ou surveiller les opérations.

② **L'affichage des données:** les résultats ou les alertes du système sont affichés.

③ **Capteurs:** sont des dispositifs qui collectent des données de l'environnement physique comme la température, la pression, mouvement.  $\Rightarrow$  Voiture (Capteur pour distance)

④ **Interface:** les interfaces font le pont entre différents éléments du système. Elles transmettent les données brutes des capteurs vers le calculateur.

Elles envoient les commandes aux actionneurs.

⑤ **Calculateur:** Il traite les données, il prend des décisions en fonction de ces données. Il envoie des commandes aux actionneurs pour effectuer une action.

$\Rightarrow$  Système de freinage, le calculateur détecte si il faut activer les freins en fonction de la distance mesurée par le capteur.

⑥ **Actionneurs:** Effectuer les actions commandées par le calculateur, comme activer un moteur, déclencher une alarme, ouvrir une vanne.  $\Rightarrow$  Dans un avion un actionneur peut ajuster les ailes pour stabiliser un système d'aérodynamique.

⑦ **Unité de stockage de données:** Elle sert à conserver les données importantes pour l'analyse ou le diagnostic du système.

**Les étapes:**  
1- Les capteurs collectent des données de l'environnement.  
2- Ces données sont transmises au calculateur via une interface.  
3- Le calculateur analyse les données, prend une décision, puis envoie des commandes aux actionneurs.

4- L'utilisateur peut interagir avec le système via la console.

5- Des données importantes sont enregistrées dans l'unité de stockage.

**Exemple Capteur:** Mesurer la température

Calculateur: Analyse la température

Actionneur: ajuste la climatisation

Console: Affiche les données

Unité de stockage: conserve l'historique

Modèle RTS: Il y a ~~Il y a~~ un ~~Il y a~~ exemple d'un système de surveillance de la température dans une chambre. Ce système peut être programmé pour mesurer la température toutes les heures. À chaque heure, une tâche est exécutée pour lire la température et éventuellement ajuster le chauffage ou la climatisation si nécessaire.

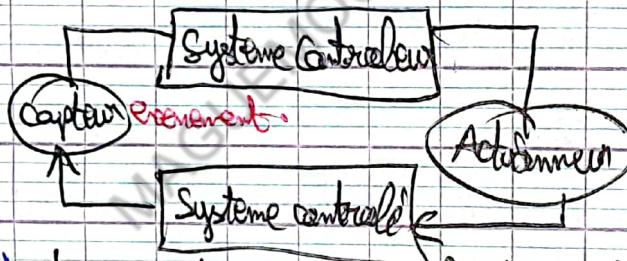
Paradigme basé sur les événements: système de contrôle de la circulation.

Lorsqu'un capteur détecte qu'un véhicule approche d'un feu de circulation cela déclenche un événement. Ce capteur envoie alors un signal au système pour changer la luminosité du feu de signalisation à vert.

Tout système réel (RTS) peut être vu comme un échange constant d'information entre 2 parties principales:

système contrôlé: c'est ce qu'on veut surveiller ou déranger (une voiture, machine, ...)

système contrôleur: c'est celui qui analyse et décide ce qu'il faut faire pour garder le système contrôlé dans un bon état.



1. Observation (Capteurs): des capteurs sont placés sur le système contrôlé pour mesurer ce qui se passe.  $\rightarrow$  capteur de température mesure la chaleur dans un four.

2. Système contrôleur: les données des capteurs sont envoyées au système contrôleur (ordinateur), ce contrôleur effectue ce qu'il faut faire pour maintenir un bon fonctionnement  $\rightarrow$  Si la température est trop basse, il faut chauffer.

3. Actionneur: le système contrôleur envoie des commandes aux actionneurs (les actionneurs font quelque chose pour corriger la situation).  $\rightarrow$  La résistance chauffe le four.

		Système Temps réel	Système embarqué
		- Système réactif avec certains délais critiques	Système dédié intégré dans un dispositif matériel
		Respecter les contraintes de temps	effectuer une tâche spécifique efficacement

Exemples d'applications Temps réel : Transports, modules supervisés en mode décalé, systèmes de production industrielle.

### ⑤ Caractéristiques des Applications Temps réel

Utilisation du Temps Concret : le temps réel est utilisé pour planifier et terminer des actions et gérer les fautes temporelles dynamiquement.

Décomposition en tâches : les actions sont modélisées en tâches concourantes, synchronisées par message partagé ou filtres de messages.

Respect des contraintes Temporelles : les ressources énumérées sont allouées aux processus les plus urgents

Ordonnancement : l'ordonnanceur gère les tâches en fonction de leur priorité et leur échéances.

Algorithmes d'ordonnancement : RM et EDF priorisent les tâches selon leurs périodes ou échéances respectives.

### ⑥ Langages pour le Temps Réel

Famille de langages possibles : Inclut langage assembleur, séquentiels (C, ADA) et langage concourantes / formels (SCADE, Lustre)

Caractéristiques des SRT : généralistes, multi-applications, non spécifiques, avec interactions via appareil système.

Inconvénients d'un OS généraliste RTS : Gourmand en ressources, assez matériel (émetteur), manque de services d'interaction sensibles.

Caractéristiques d'un ATOS : spécialisé, dédié à une application, avec un code compact et des performances adaptées.

```

4 #include <sys/types.h>
5 #include <sys/resource.h>
6 // N A I N
7 int main ()
8 {
9     int s, res;
10    struct sched_param p, q;
11    struct timespec interval;
12    // Find its scheduling policy
13    s= sched_getscheduler(getpid());
14    switch (s) {
15        case SCHED_FIFO : printf( "\nprocess %d in SCHED_FIFO ", getpid() );
16        break ;
17        case SCHED_RR : printf( "\nprocess %d in SCHED_RR\n", getpid() ); break ;
18        case SCHED_OTHER : printf( "\nprocess %d in SCHED_OTHER\n", getpid() );
19    }
20    printf( "\nPriority of this pid is %d \n", sched_get_priority_min(s));
}

```

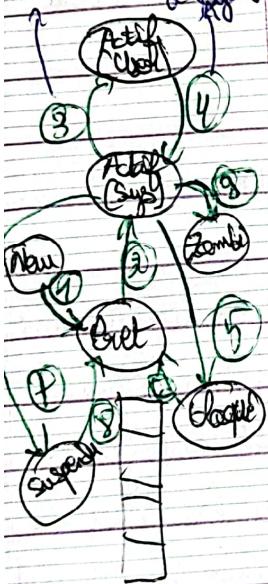
## CHAPITRE 2: Ordénancement sous Linux

① Définition: Un processus Linux est défini par son état, priorité, ses signaux, son PID, son EID, son processus père, sa session, ses identifiants utilisateurs, sa politique d'ordénancement, son temps processeur et sa date de création.

Le processus exécute une op. user  
Le processus a une op. système

② Etats d'un processus Linux:

- 1 / Nouveau → prêt: Lorsqu'un processus est créé, le système lui alloue des ressources (mémoire, id des processus) une fois que il est placé dans la file d'attente des processus prêts.



- 2 / Prêt → Actif: Un processus est sélectionné par l'ordonnanceur pour être exécuté sur le processeur, ce choix dépend de sa capacité et de la politique d'ordénancement.

- 3 / Actif (Système) → Actif (User): le processus passe de l'état système à utilisateur lorsqu'il termine une opération système et continue l'exécution d'une opération user.

- 4 / Actif (User) → Actif (Système): lorsqu'il effectue un appel système qui nécessite une ressource protégée.

liste des processus prêts  
entraînant une transmission vers le noyau

- 5 / Actif → Bloqué: Le processus attend un événement externe comme la fin d'une opération entrée/sortie. Exemple: l'utilisateur de tente attend la réponse de disque pour confirmer que le fichier a été écrit.

- 6 / Bloqué → prêt: L'événement attendu (comme la fin B/S) est produit, le processus peut maintenant exécuter à nouveau.

①

$C_f = 1$

$\Delta T = 40 - 0 - 0 = 40$

40

7/ Actif  $\rightarrow$  Suspendu: Un processus actif est suspendu par un signal (SIGSTOP).

8/ Suspendu  $\rightarrow$  brief: Le processus suspendu est réveillé par un signal (SIGCONT) et retourne dans la file des processus prêts.

9/ Actif  $\rightarrow$  zombie: Le processus a terminé son exécution mais ses données finales n'ont pas encore été récupérées par son processus parent.

### ③ Politique d'affectement Linux:

- Les processus SCHED\_FIFO et SCHED\_RR ont toujours la priorité sur SCHED\_OTHER.

- Le processus SCHED\_OTHER partage équitablement la CPU centralement à SCHED\_FIFO où le processus s'exécute jusqu'à ce qu'il termine.

- Les processus SCHED\_OTHER sont les moins prioritaires et sont gérés de manière dynamique et ne peuvent pas être exécutés si les files SCHED\_FIFO et SCHED\_RR sont vides (utilise la technique tourniquet).

Remarque: Dans le SCHED\_FIFO le processus peut se faire arrêter dans les cas suivants:

- Un autre processus plus prioritaire va l'éjecter dans la file d'attente.

- Le processus demande de faire une entrée/sortie SCHED\_Yield: abandonner le processus.

`S = sched_getscheduler(getpid());`: la fonction retourne la priorité de l'exécution en cours par pid pourtant, SCHED\_FIFO, SCHED\_RR, SCHED\_OTHER.

`sched_get_priority_min()` et `sched_get_priority_max()` pour avoir la priorité Min et Max de la priorité.

`sched_setscheduler()`: pour modifier les paramètres d'exécution.

`sched_getparam()`: pour fetcher la priorité.

`sched_rr_get_interval()`: pour obtenir la taille du quantum de temps Sched\_RR.

CHAPITRE 3 : ordonnancement centralisé de tâches temps réel  
de périodiques connaissance des instants d'arivée des requêtes (date de renvoi)

### I/ ordonnancement de Tâches périodique :

#### \* Algorithme Rate Monotonic (RM) pour les Tâches périodiques :

- priorité de la Tâche en fonction de sa période. Priorité constante,

- La Tâche avec plus petite période est la tâche la plus prioritaire

- Pour un ensemble de  $n$  tâches périodiques à échéance sur requête  $T_{Pi}(t_0, C_i, R_i, P_i)$  sachant que  $R_i$  est  $R_i'$ .

Un test d'acceptabilité est requis :

$$\sum_{i=1}^n \frac{C_i}{P_i} \leq n(2^{\frac{1}{m}} - 1) \quad \text{f. condition suffisante (CS)}$$

Si elle est vérifiée donc la configuration est ordonnable par RM, sinon on ne peut rien conclure (faudra tracer le chronogramme).

#### Algorithme Timed RoundRobin (TRR) :

## CHAPITRE 3 : Exécution centralisée de tâches temps réel

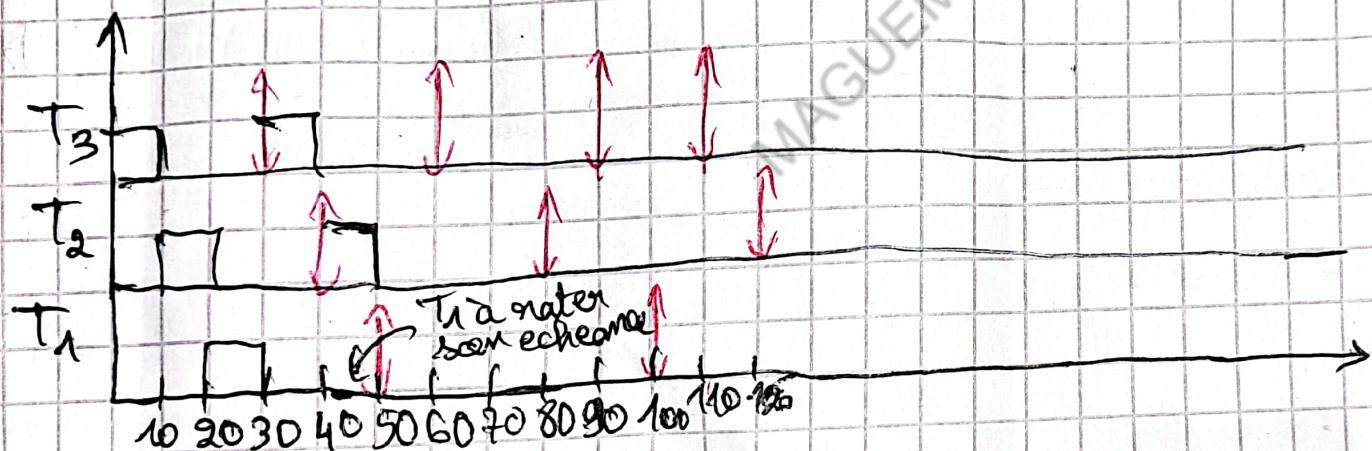
$\Rightarrow$  Tâche périodique : connaissance des instants d'arrêts des requêtes (dates de réveil)

## I/ ordonnancement de tâches périodique :

RM:  $T_1 (b=0; C=12; P=50)$ ,  $T_2 (b=10; C=10; P=40)$ ,  $T_3 (b=0; C=10; P=30)$

$$CS = \sum_{i=1}^n \frac{c_i}{e_i} \leq m(2^{\frac{1}{m}} - 1) \Rightarrow \sum_{i=1}^3 \frac{c_i}{e_i} = \frac{12}{50} + \frac{10}{40} + \frac{10}{30} = 0,82 \quad \left\{ \begin{array}{l} 0,82 > 0,78 \text{ donc } CS \text{ n'est pas vérifiée} \\ \text{on ne peut rien faire} \end{array} \right.$$

price:  $T_3 > T_2 > T_1$        $H_P = [0, 600]$



T<sub>1</sub> a raté son échéance  
il lui manque 2 UC  
donc : le chronogramme  
n'est pas réalisable.

## Algorithm Inverse Deadline (IDM):

- Priorité de la tâche en fonction de son délai critique. Priorité constante
- La tâche avec le plus petit délai critique est la tâche la plus prioritaire.
- Pour un ensemble de  $n$  tâches périodiques à échéance sur requête  $T_p$  ( $b_i, c_i, R_i, p_i$ ).

Un test d'acceptabilité est requis:

$$\sum_{i=1}^n \frac{c_i}{R_i} \leq m \left( \alpha^{\frac{1}{m}} - 1 \right) \quad \text{condition suffisante (CS)}$$

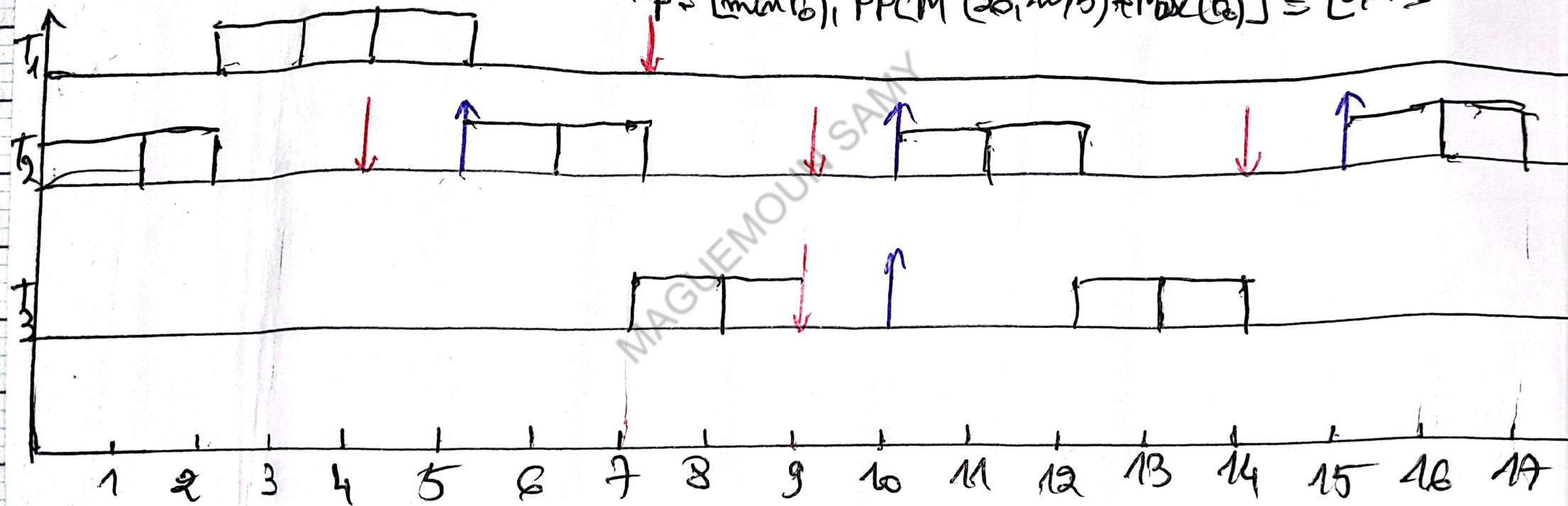
- Si elle est vérifiée donc la configuration est ordonnable par IDM, sinon on ne peut rien conclure (faudra tracer le chronogramme).

DM:  $T_1(t_0=0; C=3; D=7; P=20)$ ,  $T_2(t_0=0; C=2; D=4; P=5)$ ,  $T_3(t_0=0; C=2; D=9; P=10)$

$$\sum_{i=1}^3 \frac{C_i}{P_i} > \frac{3}{7} + \frac{2}{4} + \frac{2}{9} = 1.15 \quad m(2^{\frac{1}{m}} - 1) = 3(2^{\frac{1}{3}} - 1) = 0.46 \text{ donc on ne peut rien conclure}$$

Price  $\downarrow$

$$H_P = [\min(t_0), \text{PPCM}(20, 10, 5) + \max(t_0)] = [0, 20]$$



## Algorithme Earliest Deadline (EDF):

Buorité de la Tâche est en fonction de son échéance. Priorité dynamique

À l'instant  $t$ , la Tâche avec la plus proche échéance est la Tâche la plus prioritaire.

$D_i = P_i$

on a 2 Cas:

1. Pour un ensemble de  $n$  Tâches  
ordiquées à échéance sur requête  
 $(i \leq P_i)$ ,  $T_{Pi}(r_i, C_i, R_i, P_i)$ , (échéance =  
= periode)

Test d'acceptabilité est requis:  
 $\frac{C_i}{P_i} \leq 1 \}$  Condition nécessaire  
et suffisante (CNS)

elle est vérifiée donc la config  
corde avec EDF Si non  
config n'est pas corde

$D_i < P_i$

Cas 2: Pour un ensemble de  $n$  Tâches périodiques quelconques ( $P_i \leq P_i$ ),  $T_{Pi}(r_i, C_i, R_i, P_i)$ .

Test d'acceptabilité est requis:

$$\sum_{i=1}^n \frac{C_i}{P_i} \leq 1 \} \text{ condition suffisante (CS)}$$

Si elle est vérifiée donc la configuration  
est commandable par EDF, Sinon on ne  
peut rien conclure. (tracer le chrono)

$$\sum_{i=1}^n \frac{C_i}{P_i} \leq 1 \} \text{ condition nécessaire (CN)}$$

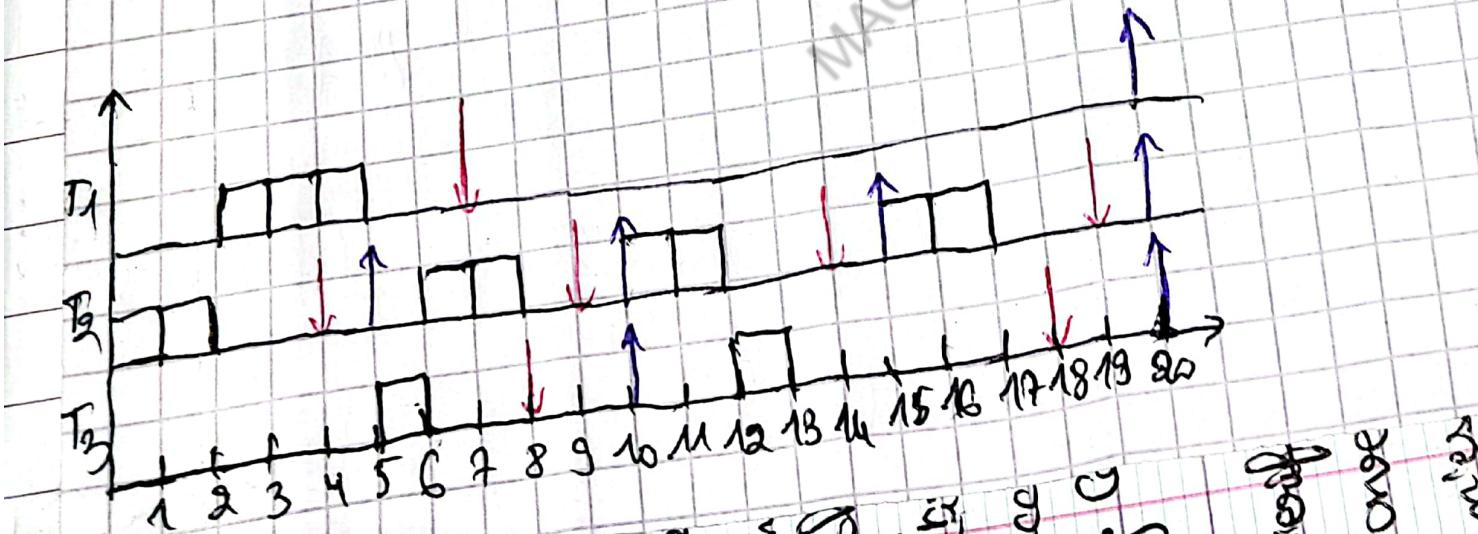
Si elle est vérifiée donc on ne peut pas  
conclure, Sinon la configuration n'est  
pas commandable par EDF. (tracer le  
chrono).

EDF:  $T_1(t_0=0; C=3; D=7; P=20)$ ,  $T_2(t_0=0; C=2; D=6; P=5)$ ,  $T_3(t_0=0; C=1; D=3; P=10)$   
 Non vérifiable, on ne peut rien conclure

$CS = \sum_{i=1}^m \frac{C_i}{R_i} \leq 1 \Rightarrow CS = \frac{3}{7} + \frac{3}{4} + \frac{1}{8} = 1,05 > 1$  Non vérifiable, on ne peut rien conclure

$CN = \sum_{i=1}^m \frac{C_i}{R_i} \leq 1 \Rightarrow CN = \frac{3}{20} + \frac{2}{5} + \frac{1}{10} = 0,65 < 1$  Vérifiable, on peut en conclure

Puis: change à chaque calcul, il dépend l'échéance la plus proche.  
 $t \in [0, 20]$



### Algorithm Least Latency First (LLF)

- Priorité de la Tâche en fonction de la bouteille. Priorité dynamique.
- A l'instant  $t$ , la Tâche avec la plus petite bouteille (marge) est la tâche la plus prioritaire.

$$\text{marge (bouteille)} = \text{échéance} - \frac{\text{temps de calcul}}{\text{Temps courant Restant}}$$

$$D_i = P_i$$

cas 1 : Pour un ensemble de  $m$  Tâches périodiques à échéance sur requête ( $R_i = P_i$ ),  $Tpi(b_i, C_i, R_i, p_i)$ .

Un Test d'acceptabilité est requis :

$$\sum_{i=1}^m \frac{C_i}{p_i} \leq 1 \quad \text{condition nécessaire et suffisante (en)}$$

Pour le 2<sup>eme</sup> Cas Même chose que EDF quand ( $R_i \leq P_i$ )

Hypothèse :  $tp = [\min(t_0), \text{PPCM(Boucle)} + \text{Max}(t_0)]$

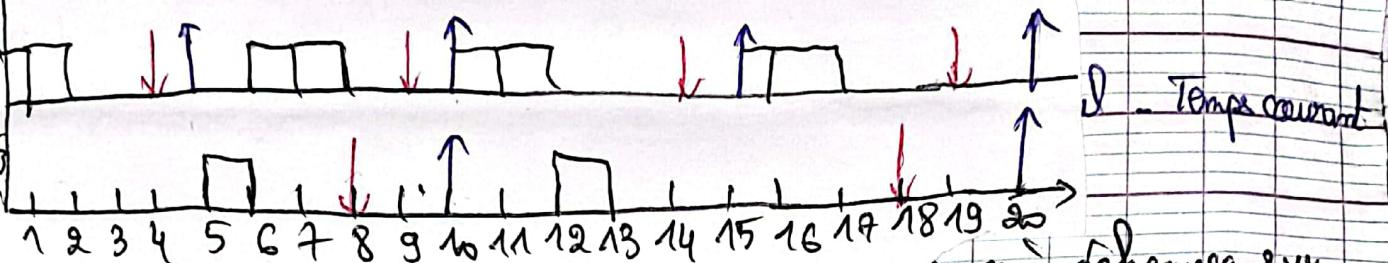
$$\text{PPCM } (a, b) = \frac{a \times b}{\text{PGCD}(a, b)}$$

LLF:  $T_1(t_0=0; C=3; D=7; P=20)$ ,  $T_2(t_0=0; C=2; D=4; P=5)$ ,  $T_3(t_0=0; C=1; D=8; P=10)$

$$CS = \sum_{i=1}^m \frac{C_i}{P_i} \leq 1 \Rightarrow CS = \frac{3}{7} + \frac{2}{4} + \frac{1}{8} = 1,05 > 1$$

$$CN = \sum_{i=1}^m \frac{C_i}{P_i} \leq 1 \Rightarrow CN = \frac{3}{20} + \frac{2}{5} + \frac{1}{10} = 0,65 < 1$$

Rése: change en fonction de la lenteur la plus petite



$t=0 \begin{cases} t_1 = 7 - 3 - 0 = 4 \\ t_2 = 4 - 2 - 0 = 2 \\ t_3 = 8 - 1 - 0 = 7 \end{cases}$

$$t=1 \begin{cases} t_1 = 7 - 3 - 1 = 3 \\ t_2 = 4 - 1 - 1 = 2 \\ t_3 = 8 - 1 - 1 = 6 \end{cases}$$

$$t=2 \begin{cases} t_1 = 7 - 3 - 2 = 2 \\ t_2 = 1 \\ t_3 = 8 - 1 - 2 = 5 \end{cases}$$

$$t=3 \begin{cases} t_1 = 7 - 2 - 3 = 2 \\ t_2 = 1 \\ t_3 = 8 - 1 - 3 = 4 \end{cases}$$

$$t=4 \begin{cases} t_1 = 7 - 1 - 3 = 1 \\ t_2 = 1 \\ t_3 = 8 - 1 - 4 = 3 \end{cases}$$

$$t=5 \begin{cases} t_1 = 1 \\ t_2 = 9 - 2 - 5 = 2 \\ t_3 = 8 - 1 - 5 = 2 \end{cases}$$

$$t=6 \begin{cases} t_1 = 1 \\ t_2 = 9 - 2 - 6 = 1 \\ t_3 = 1 \end{cases}$$

$$t=7 \begin{cases} t_1 = 1 \\ t_2 = 9 - 1 - 7 = 1 \\ t_3 = 1 \end{cases}$$

~~$$t=8 \begin{cases} t_1 = 1 \\ t_2 = 1 \\ t_3 = 1 \end{cases}$$~~

$$t=9 \begin{cases} t_1 = 1 \\ t_2 = 1 \\ t_3 = 1 \end{cases}$$

$$t=10 \begin{cases} t_1 = 1 \\ t_2 = 14 - 2 - 10 = 2 \\ t_3 = 18 - 1 - 10 = 7 \end{cases}$$

$$t=11 \begin{cases} t_1 = 1 \\ t_2 = 14 - 1 - 14 = 2 \\ t_3 = 18 - 1 - 11 = 6 \end{cases}$$

$$t=12 \begin{cases} t_1 = 1 \\ t_2 = 1 \\ t_3 = 18 - 1 - 12 = 5 \end{cases}$$

$$t=13 \begin{cases} t_1 = 1 \\ t_2 = 1 \\ t_3 = 1 \end{cases}$$

$$t=14 \begin{cases} t_1 = 1 \\ t_2 = 1 \\ t_3 = 1 \end{cases}$$

$$t=15 \begin{cases} t_1 = 1 \\ t_2 = 19 - 2 - 15 = 2 \\ t_3 = 1 \end{cases}$$

$$t=16 \begin{cases} t_1 = 1 \\ t_2 = 19 - 1 - 16 = 2 \\ t_3 = 1 \end{cases}$$

$$t=17 \begin{cases} t_1 = 1 \\ t_2 = 1 \\ t_3 = 1 \end{cases}$$

$$t=18 \begin{cases} t_1 = 1 \\ t_2 = 1 \\ t_3 = 1 \end{cases}$$

$$t=19 \begin{cases} t_1 = 1 \\ t_2 = 1 \\ t_3 = 1 \end{cases}$$

ques à échéance sur

necessary et suffisante (cas)

DP quand ( $R_i \leq P_i$ )

bride] + Max( $t_i$ )

que:

instants d'arrêts

→ Traitement en arrière plan Spéciale

→ Traitement par Série

Scrutat-

ion

## II/ordonnancement de tâches aperiodique:

Tâche aperiodique : on ne connaît pas les instants d'arrivée des requêtes (dates d'appel de recul).

ya 2 types de Tâches aperiodique

Tâches aperiodique à contraintes relatives

( ) strictes

→ Traitement en arrière plan Spor

→ Traitement par Service

Ser  
ice



## ① Tâches aperiodiques à contraintes relatives

Maximiser le Temps de Réponse de Toute la configuration

### 1- Traitement par accès plan

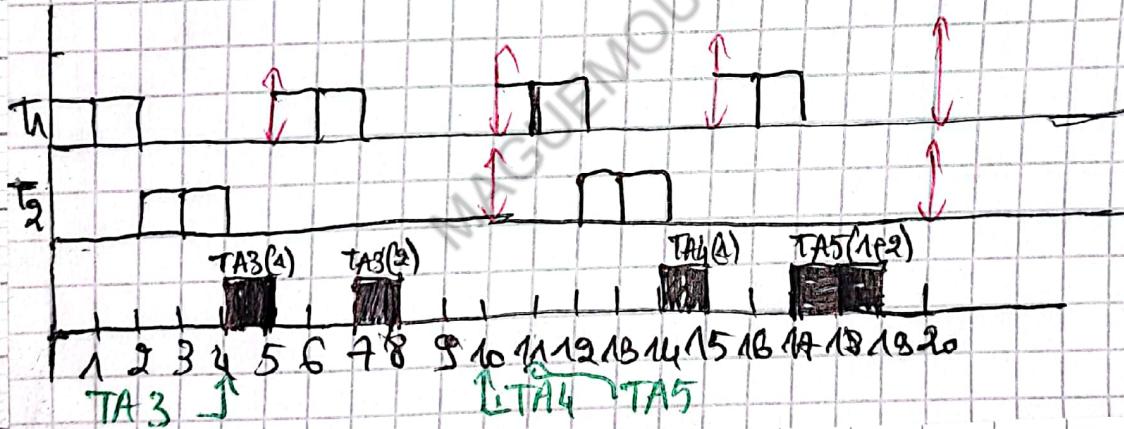
- Les Tâches aperiodiques sont ordonnancées quand le processeur est libre. Si plusieurs tâches attendent, elles sont traitées en mode FIFO. Le plus simple mais le moins performant

Exemple avec PMA:  $T_1: (r=0; C=2; P=5)$ ,  $T_2: (r=0; C=2; P=10)$

$T_3: (r=4; C=2)$ ,  $T_4: (r=10; C=1)$ ;  $T_5: (r=11; C=2)$

$$CS = \sum_{i=1}^m \frac{P_i}{P_1} \leq m \left( 2^{\frac{m}{m-1}} - 1 \right) \Rightarrow 0,6 \leq 0,82 \text{ ordonnable.}$$

Priore  $T_1 > T_2$        $t_{fp} = [0, 20]$



### 2- Traitement par Serveur

Un Serveur est une tâche périodique (non ordonnable). Le Serveur est caractérisé par une période  $P$ , Temps d'exécution  $C$ , un premier réveil. Le Serveur est ordonné suivant le même algorithme que les tâches périodiques. Une fois actif (passé de la priorité), le Serveur sort les Tâches aperiodiques durant son temps d'exécution ( $C$ ), dans la limite de sa capacité. L'ordre de traitement des tâches aperiodiques ne dépend pas de l'algorithme général, il dépend de l'approche de traitement pour Serveur (1)

## \*Sensuel par soumission:

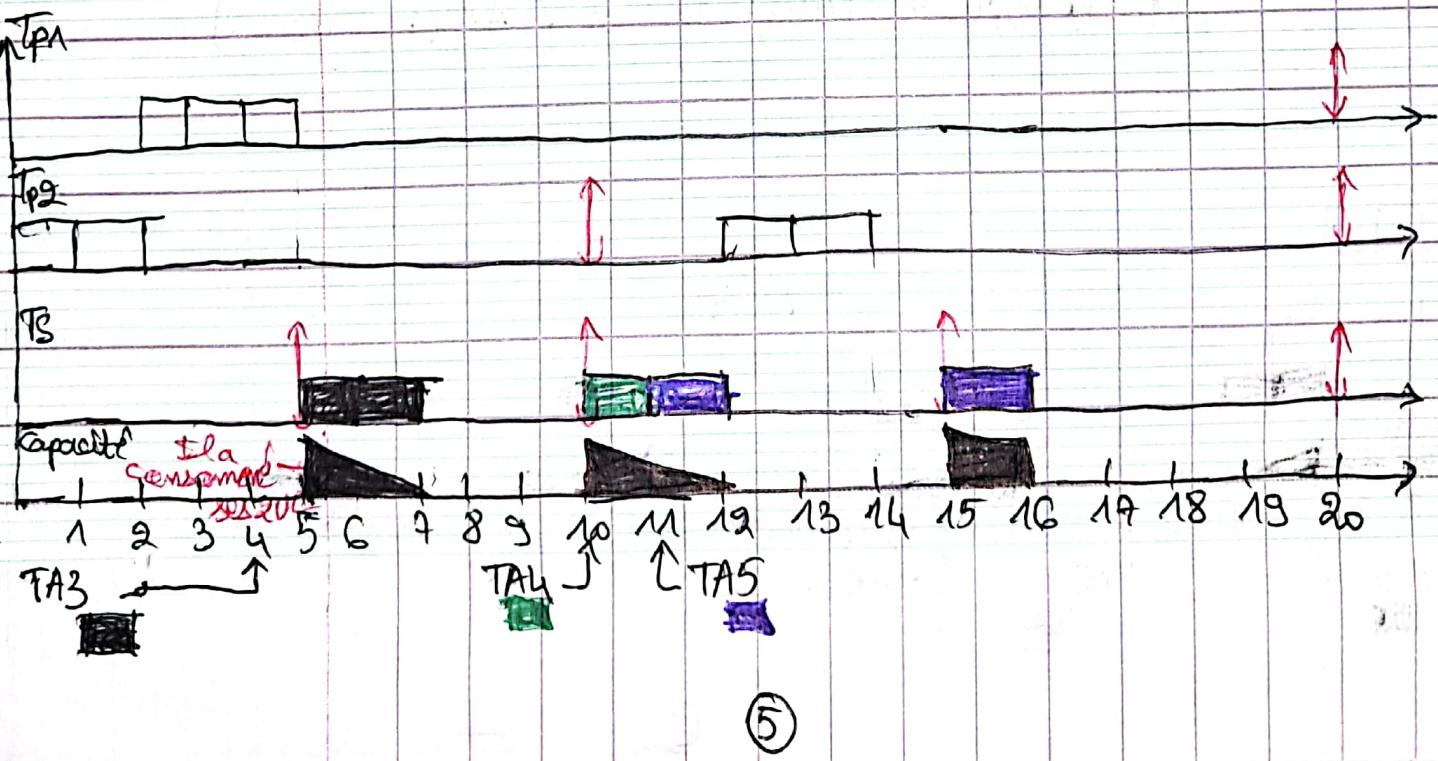
Coordonnancement des tâches aperiodiques pour serveur dont le principe:  
 À chaque activation du serveur, il exécute les tâches aperiodiques jusqu'à épuisement de la capacité ( $C$ ) ou jusqu'à ce qu'il n'y ait plus de tâches en attente. Si aucune tâche aperiodique n'est en attente, le serveur se suspend immédiatement.

1. Il perd le processeur (donc les tâches périodiques peuvent reprendre leur exécution).

2. Il perd sa capacité (donc si après un instant de temps au plus court une tâche aperiodiques se présente, cette dernière ne peut pas être exécutée, elle doit attendre une prochaine activation du serveur).

Exemple avec RMA: TP1 ( $b_0=0; C=3; P=20$ ), TP2 ( $b_0=0; C=2; P=20$ ),

TS ( $b_0=0; C=2; P=5$ ), TA3 ( $r=4; C=2$ ), TA4 ( $r=10, C=1$ ), TA5 ( $r=11, C=2$ )  
 $tp = [0, 20]$



## Serveur Sporadique :

Traitement par serveur tel que le serveur par rotation sauf que :

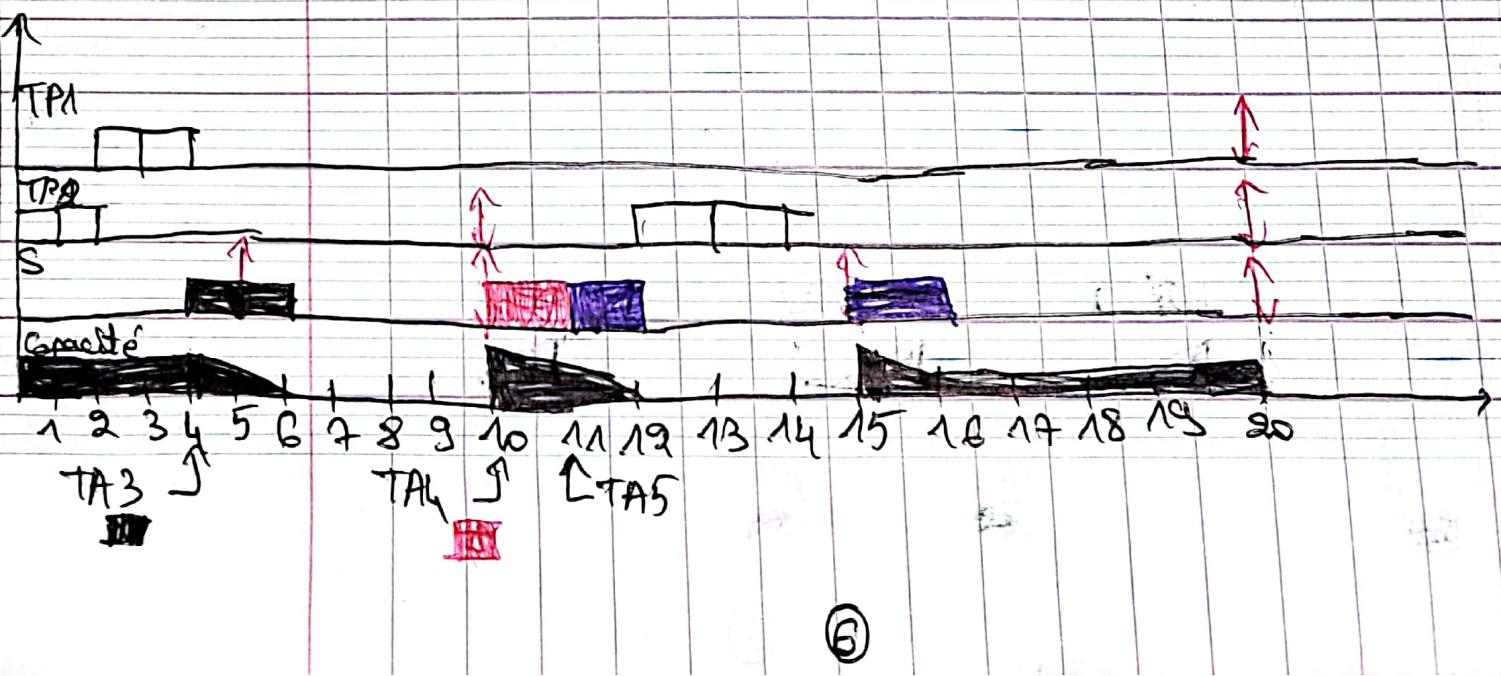
- Si aucune tâche n'est en attente, le serveur se suspend immédiatement.

1- Il perd le processeur.

2- Il ne perd pas sa capacité (donc si après un instant de temps ou plus dont une tâche aperiodique se présente, cette dernière est exécutée immédiatement (elle ne doit pas attendre une prochaine activation du serveur).

- La capacité est conservée si elle n'est pas épuisée et donc peut être utilisée même après la perte du processeur. Le serveur sporadique améliore le temps de réponse des tâches aperiodiques sans diminuer le taux d'utilisation du processeur pour les tâches périodiques.

Exemple avec RMA : TP1 ( $t_0=0; C=3; P=20$ ), TP2 ( $t_0=0; C=2; P=10$ ), TS ( $t_0=0; C=2; P=5$ )  
TA3 ( $n=4; C=2$ ), TA4 ( $n=10; C=1$ ), TA5 ( $n=11; C=2$ )



⑥

## ② Tâches aperiodiques à contraintes strictes:

Méthode de test de garantie: ordonnance les tâches en EDF.

- À chaque nouvelle tâche aperiodique, faire exécuter un "Test de garantie" pour: vérifier toutes les contraintes temporelles seront respectées de toutes les tâches périodiques et aperiodiques.

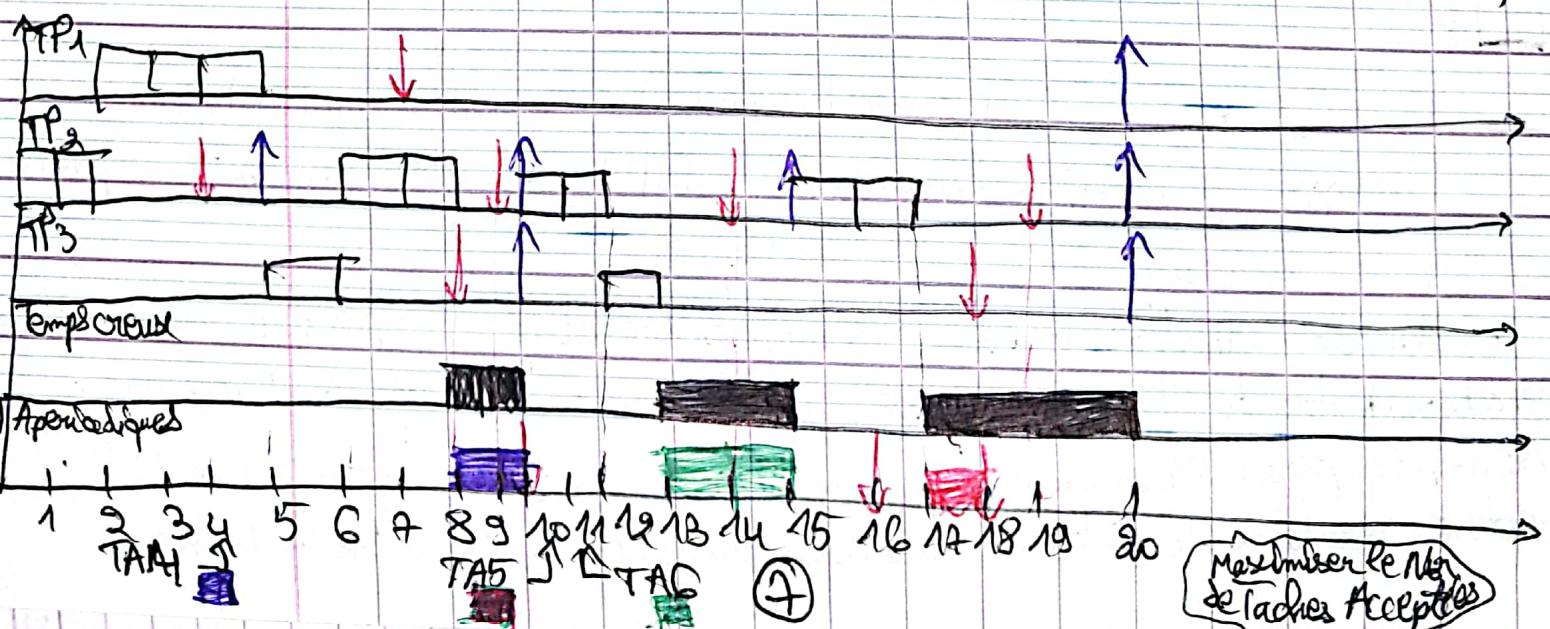
- Si oui: la nouvelle tâche aperiodique est acceptée

- Si non: on refuse la nouvelle tâche aperiodique.

### Acceptation dans les temps creux:

ordonnancement des tâches périodiques par EDF. Les tâches aperiodiques acceptées sont ordonnancées dans les temps creux des tâches périodiques si: à la présence de chaque tâche aperiodique, on exécute le test de garantie: si le test est vérifié alors, tester l'existence d'un temps creux suffisant. Si OK, la tâche est acceptée sinon elle est rejetée.

Exemple:  $T_P1(r=0; C=3; D=7; F=20)$ ,  $T_P2(r=0; C=2; D=4; P=5)$ ,  $T_P3(r=0; C=1; D=8; P=10)$ ,  $TA4(r=4; C=2; Q=10)$ ,  $TA5(r=10; C=1; D=18)$ ,  $TA6(r=11; C=2; D=16)$



### III/ Ordénancement avec contraintes de ressources

#### ① Problème d'inversion de priorité:

L'inversion de priorité survient lorsque :

- Une tâche de haute priorité ( $T_1$ ) est bloquée par une tâche de faible priorité ( $T_2$ ) qui détient une ressource partagée.
- Une tâche intermédiaire ( $T_3$ ) ayant une priorité moyenne s'exécute, en retardant ainsi  $T_2$  et par conséquent  $T_1$ .

demande R1      Exemple:

→ blocage



$T_1$  (priorité haute) veut accéder à une ressource R1 détenu par

$T_2$  (priorité basse)

-  $T_3$  (priorité moyenne) s'exécute, empêchant  $T_2$  de terminer.

Résultat:  $T_1$  attend initialement parce que  $T_3$  interfère. Lien quelle demande R1 libération soit moins prioritaire que  $T_1$

#### ② Problème Interbloque:

Un Interbloque (deadlock) survient lorsque plusieurs tâches

se retrouvent dans une situation où chacune attend l'autre pour une ressource détenu par l'autre, formant une boucle circulaire.

bloque

P(A2)

P(A1)

Exemple:

Tâche  $T_1$  demande R1 puis P2.      Résultat:

bloque

- Tâche  $T_2$  demande P2 puis R1

?  $T_1$  attend P2 détenu par  $T_2$

$T_2$  attend R1 détenu par  $T_1$

condition nécessaire pour un Interbloque:

Exclusion mutuelle: Une ressource ne peut être utilisée que par une seule tâche à la fois.

Préemption: Une tâche détient une ressource tout en attendant une autre.

Non-préemption: Les ressources ne peuvent pas être retirées de force à une tâche.

Attente circulaire: Une chaîne circulaire d'attente existe entre les tâches.

## Solution aux interblocages:

- Tâche d'attente circulaire: imposer un ordre global d'accès aux ressources et libérer les ressources si une tâche ne peut pas obtenir toutes celles dont elle a besoin.

- Détection et reprise: Identifier les interblocages à l'aide de graphes d'attente et libérer ou redemander certaines tâches.

- Préemption prioritaire: Retirer une ressource détenue par une tâche bloquante et la redistribuer à une tâche prioritaire.

- Utilisation de protocoles: PCP et PIP

## ③ Section critique non interruptible:

Une section critique non interruptible est une méthode simple pour garantir l'accès exclusif à une ressource partagée pendant son exécution, la tâche accédant à la ressource ne peut être interrompue par une autre tâche.

Avantages: Simplicité et facilité de mise en œuvre, cette section critique évite les interruptions inutiles, réduisant les conflits lors de l'accès aux ressources partagées.

Inconvénients: Retards possibles pour les tâches prioritaires, surtout si la section critique est longue. Moins adapté aux systèmes multitâches complexes avec de plus dynamiques priorités.

Exemple : Cours page 69.

## Utilisation:

- Convientable pour des systèmes simples avec des interruptions courtes.

- Moins efficace dans des environnements complexes où les priorités doivent être strictement respectées.

DMA

exemple

cours page 73

T1  
TD  
cen exo 7,8

#### ④ Protocole de l'héritage de priorité (PIP):

Le PIP résout l'inversion de priorité en faisant hériter la priorité de  $T_1$  pendant qu'elle utilise  $R_1$ .

Fonctionnement:

- Si  $T_1$  (priorité haute) veut accéder à  $R_1$  détenue par  $T_2$  (priorité basse),  $T_2$  hérite immédiatement de la priorité de  $T_1$ .
- $T_3$  (priorité moyenne) est bloquée et ne peut pas préempter  $T_2$ .
- Une fois  $T_2$  libérée de  $R_1$ , elle reprend sa priorité initiale.

Avantage: Limite le temps de blocage à la durée d'utilisation de la ressource partagée.

Inconvénient: Complexité accrue si plusieurs tâches réclament la même ressource.

#### ⑤ Protocole de priorité plafond (PCP):

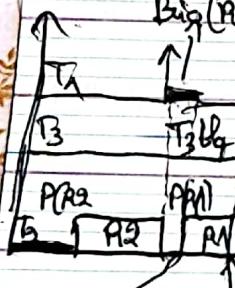
Si PCP améliore PIP en attribuant un plafond prioritaire à chaque ressource. Ce plafond est égal à la priorité la plus élevée parmi toutes les tâches susceptibles d'utiliser cette ressource.

$p_{max}(T_1) > p_{max}(T_2)$

Fonctionnement:

$p(R_1) \Rightarrow b_{R_1}$

$b_{R_1}(R_1) > p_{max}(T_2)$  - Lorsqu'une tâche  $T_2$  accède à une ressource  $R_1$ , elle prend automatiquement la priorité plafond de  $R_1$ .



Une tâche  $T_3$  ne peut pas préempter  $T_2$  si sa priorité est inférieure au plafond.

Si  $T_1$  (plus prioritaire que  $T_2$ ) veut  $R_1$ , elle attend que  $T_2$  termine.

T2 hérite Avantages: Empêche les interblocages, réduit le nombre de préemptions.

des priorités Inconvénients: Complexité accrue pour calculer et gérer les plafonds initiaux.

## IV/ Ordénancement avec contraintes de dépendances:

L'ordénancement avec contraintes de dépendances consiste à gérer des tâches liées par des relations de précédence (une tâche doit être terminée avant qu'une autre puisse commencer) tout en respectant leurs contraintes temporelles.

### Réte Monotonie (RM):

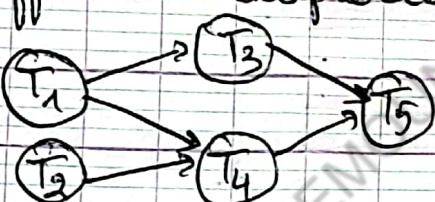
Les dépendances sont transformées en modifiant les dates de début et les délais des tâches pour les rendre indépendantes.

$$\begin{cases} \text{j: prédecesseur} \\ \text{i: membre actuel.} \end{cases} \quad t_i^* = \max\{t_i, t_j + f\} \text{ pour tous les } j \text{ tels que } T_j \rightarrow T_i$$

Si  $T_j \rightarrow T_i$  alors  $\text{préc}(j) > \text{préc}(i)$  dans le respect de la règle d'affectation des priorités par RM.

Pas de but

exemple :



Tâche	i	$t_i^*$	$w_i$	$p_i$	
$T_1$	0	1	0	1	$\rightarrow \max\{0, 1\} = 1$
$T_2$	5	2	5	1	$\rightarrow \max\{5, 1\} = 5$
$T_3$	0	2	0	2	$\rightarrow \max\{0, 2\} = 2$
$T_4$	0	1	5	2	$\rightarrow \max\{0, 1\} = 5$
$T_5$	0	3	5	3	$\rightarrow \max\{0, 3\} = 5$

### Inverse Deadline (IDM):

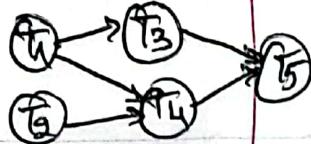
La transformation se passe hors ligne sur la date finale et sur les délais critiques.

$$\begin{cases} \text{j: prédecesseur} \\ \text{i: membre actuel} \end{cases} \quad t_i^* = \max\{t_i, t_j + f\} \text{ pour tous les } j \text{ tels que } T_j \rightarrow T_i$$

$$D_i^* = \max\{D_i, D_j + f\} \text{ pour tous les } j \text{ tels que } T_j \rightarrow T_i$$

Si  $T_j \rightarrow T_i$  alors  $\text{préc}(j) < \text{préc}(i)$  dans le respect de la règle d'affectation des priorités par DM.

exemple:



Tache	$t_i^*$	$C_i$	$D_i^*$	$t_i^*$	$D_i^*$	$P_i$
T1	0	1	5	0	5	1
T2	5	2	7	5	7	3
T3	0	2	4	0	5	2
T4	0	1	5	3	7	4
T5	0	3	6	5	7	5

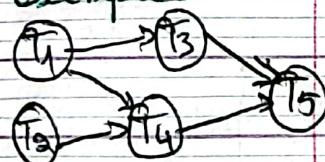
Même chose comme AM  
Valable pour DM  
pour les t<sub>i</sub>\* et les D<sub>i</sub>\*

### Earliest Deadline First (EDF):

La modification des échéances de façon à ce qu'une tâche soit toujours un D<sub>i</sub> inférieure à celui de ses successeurs. Une tâche ne doit être activée que si tous ses prédecesseurs ont terminé leur exécution. Modification de la date de début et de l'échéance.

$$\begin{cases} \text{j: predecessor} \\ \text{i: precedent actual} \\ \text{j: immed actual} \\ \text{j: successor} \end{cases} \quad \begin{cases} h_j^* = \max\{t_i, \max_{j \neq i} t_j + C_j\} \text{ pour tous les } j \text{ tels que } T_j \rightarrow T_i \\ d_i^* = \min\{d_i, \min_{j \neq i} t_j + C_j\} \text{ pour tous les } j \text{ tels que } T_i \rightarrow T_j \end{cases}$$

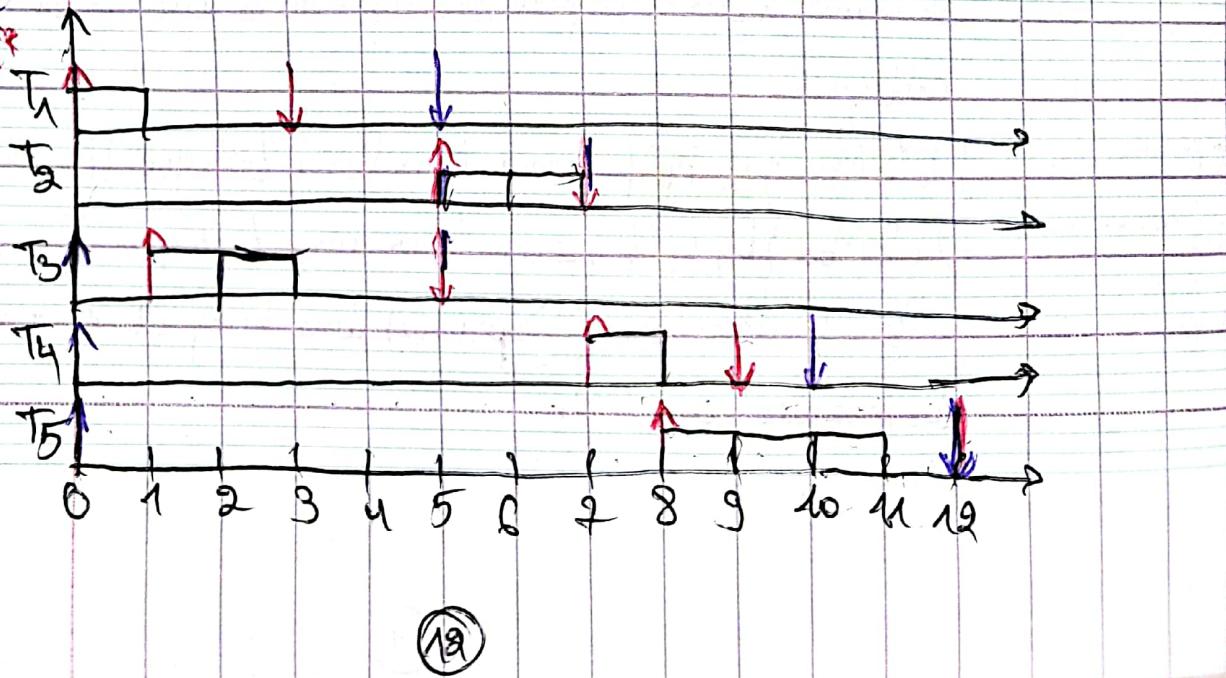
Exemple:



$$\begin{aligned} & \begin{cases} T_1 \text{ a } T_3 \text{ et } T_4 \text{ comme pred} \\ T_2 \text{ a } T_1 \text{ et } T_4 \text{ comme pred} \\ T_3 \text{ a } T_4 \text{ comme pred} \\ T_4 \text{ a } T_5 \text{ comme pred} \end{cases} \quad \begin{cases} T_1 \text{ n'a pas de pred } t_1^* = 0 \\ T_2 \text{ a } T_3 \text{ et } T_4 \text{ comme succ} \\ T_3 \text{ a } T_4 \text{ comme succ} \\ T_4 \text{ a } T_5 \text{ comme succ} \end{cases} \\ & \begin{cases} d_5^* = \max(t_5, \max(t_3^* + C_3, t_4^* + C_4)) = 8 \\ t_5^* = \max(t_5, \max(t_3^* + C_3, t_4^* + C_4)) = 8 \\ t_4^* = \max(t_4, \max(t_1^* + C_1, t_3^* + C_3)) = 7 \\ t_3^* = \max(t_3, \max(t_1^* + C_1, t_2^* + C_2)) = 5 \\ t_2^* = \max(t_2, \max(t_1^* + C_1, t_4^* + C_4)) = 7 \end{cases} \quad \begin{cases} T_1 \text{ a } T_1 \text{ comme pred } t_1^* = \max(t_3^* + C_3, t_4^* + C_4) = 7 \\ T_2 \text{ a } T_1 \text{ comme pred } t_2^* = \max(t_1^* + C_1, t_3^* + C_3) = 5 \\ T_3 \text{ a } T_1 \text{ comme pred } t_3^* = \max(t_1^* + C_1, t_2^* + C_2) = 5 \\ T_4 \text{ a } T_1 \text{ comme pred } t_4^* = \max(t_1^* + C_1, t_3^* + C_3) = 5 \\ T_5 \text{ a } T_1 \text{ comme pred } d_5^* = \min(d_5, \min(d_5 - C_5)) = 5 \end{cases} \end{aligned}$$

- ancien:  $t_i, d_i$

- nouveau:  $t_i^*, d_i^*$



# Questions de cours ptr

**1-expliquer pourquoi les politiques d'ordonnancement sous linux ne sont pas adéquates pour les applications temps réels?**

**Les politiques d'ordonnancement sous Linux ne garantissent pas des délais stricts ni une gestion déterministe des priorités, ce qui est essentiel pour les applications temps réel. Les latences d'exécution peuvent être variables et l'accès aux ressources partagées peut être retardé. Les systèmes temps réel nécessitent des garanties de prévisibilité et de respect des deadlines, que Linux ne peut pas offrir de manière fiable.**

**2-Quel est le principal défaut de l'algorithme LLF?**

**L'algorithme LLF (Least Laxity First) planifie les tâches en fonction de leur laxité, c'est-à-dire le temps restant avant la date limite. Son principal défaut est qu'il peut entraîner des inversions de priorité, où une tâche à faible priorité peut empêcher l'exécution d'une tâche plus urgente. Cela peut causer des violations de délais et rendre l'ordonnancement inefficace dans des systèmes avec beaucoup de tâches.**

**3-Expliquer la différence entre un système temps réel et un système embarqué ?**

**Un système temps réel est conçu pour répondre à des événements dans des délais précis, avec des contraintes de temps strictes. Un système embarqué est un système informatique intégré dans un dispositif pour accomplir une fonction spécifique. Tous les systèmes embarqués ne sont pas en temps réel, mais les systèmes temps réel peuvent être embarqués.**

**4-le fait de donner des priorités différentes aux tâches temps réel ne garantit pas qu'une tâche ayant la priorité maximale ne puisse reprendre le contrôle à l'instant désiré comment appelle-t-on ce phénomène ?dans quel cas se produit-il?**

**Le phénomène où une tâche ayant la priorité maximale ne peut pas reprendre le contrôle à l'instant désiré s'appelle l'inversion de priorité. Ce phénomène se produit lorsqu'une tâche de haute priorité est bloquée par une tâche de basse priorité qui détient une ressource partagée. Si la tâche de basse priorité ne termine pas son exécution, la tâche de haute priorité devra attendre, ce qui viole les contraintes temporelles des systèmes temps réel.**

**5-expliquez les avantages de l'algorithme DM par rapport a l'algorithme RM?**

**L'algorithme Deadline Monotonic (DM) priorise les tâches en fonction de leurs deadlines, offrant ainsi une meilleure gestion des tâches avec des deadlines variables et une garantie de respect des deadlines. Il est plus flexible et adapté aux systèmes où les deadlines ne sont pas fixes. En revanche, Rate Monotonic (RM) priorise les tâches selon leurs périodes et est moins efficace pour les tâches avec des deadlines différentes. DM utilise mieux les ressources et s'adapte mieux aux exigences temporelles strictes.**

**6-classez les systèmes suivants dans l'une des classes systeme tr dur ,mou ou ferme  
contrôle aérien vod transaction boursière control d'une centrale nucléaire**

**Contrôle aérien : Système temps réel dur**

Le contrôle aérien nécessite des réponses strictes et garanties dans des délais précis pour assurer la sécurité des vols, ce qui en fait un système temps réel dur.

**VOD (Vidéo à la demande) : Système temps réel mou**

Bien que la VOD nécessite des délais relativement courts pour la lecture des vidéos, les retards occasionnels n'entraînent pas de conséquences graves. Il s'agit donc d'un système temps réel mou, où les délais sont importants mais peuvent être tolérés dans une certaine mesure.

**Transaction boursière : Système temps réel mou**

Les transactions boursières exigent des délais de réponse rapides, mais un petit retard ou une petite perte de données ne met pas en danger le système. Cela en fait un système temps réel mou.

**Contrôle d'une centrale nucléaire : Système temps réel dur**

Le contrôle d'une centrale nucléaire doit garantir des délais de réponse extrêmement stricts, car un retard ou une erreur pourrait avoir des conséquences . C'est donc un système temps réel dur.

**7-donnez une définition pour chacun des termes suivants:surcharge transitoire,problème d'inversion de priorité ?**

**Surcharge transitoire :** Augmentation temporaire de la charge de travail d'un système, entraînant des retards ou une dégradation des performances avant un retour à la normale.

**Problème d'inversion de priorité :** Lorsque une tâche de faible priorité bloque l'exécution d'une tâche de haute priorité, entraînant des violations de délais dans un système temps réel.

**8-les exécutifs temps réel proposent de nombreux outils de synchronisation et de communication donner 2 exemples de chacun d'eux ?**

## Outils de synchronisation :

- **Sémaphores** : Utilisés pour gérer l'accès concurrent aux ressources partagées. Un sémaphore peut être binaire (0 ou 1) ou compter le nombre de ressources disponibles.
- **Verrous (Mutex)** : Permettent de garantir qu'une seule tâche accède à une ressource partagée à un moment donné, en bloquant les autres tâches jusqu'à ce que la ressource soit libérée.

## 2. Outils de communication :

- **Files de messages** : Permettent l'échange de messages entre différentes tâches ou processus, souvent utilisés pour transmettre des données d'une tâche à une autre de manière asynchrone.
- **Mémoire partagée** : Utilisée pour permettre à différentes tâches ou processus d'accéder directement à la même zone mémoire, facilitant la communication rapide entre eux.

## Questions générales ( chapitre 1)

1. Quelle est la différence entre une application classique et une application temps réel ?
    - o Une application classique ne dépend pas du temps pour l'exactitude de ses résultats, tandis qu'une application temps réel doit produire ses résultats dans un délai spécifié (échéance) pour être considérée correcte.
  2. Quelles sont les caractéristiques principales des systèmes temps réel ?
    - o Respect des échéances temporelles.
    - o Fonctionnement lié à des événements externes.
    - o Importance de la fiabilité et de la gestion des ressources limitées.
  3. Pourquoi est-il crucial de respecter les échéances dans un système temps réel ?
    - o Le non-respect des échéances peut entraîner des erreurs graves, comme des défaillances dans des systèmes critiques (ex. : aviation, santé).
- 

## Questions sur les définitions

4. Comment définiriez-vous un système temps réel ?
    - o C'est un système dont l'exactitude dépend non seulement des calculs réalisés, mais aussi du moment où les résultats sont produits.
  5. Quels sont les trois types de systèmes temps réel selon la classification (dur, souple, ferme) ? Donnez des exemples pour chacun.
    - o Temps réel dur : Une échéance manquée entraîne une défaillance critique (ex. : contrôle du trafic aérien).
    - o Temps réel souple : Une échéance manquée entraîne une dégradation acceptable (ex. : décalage audio-vidéo).
    - o Temps réel ferme : Une échéance manquée est tolérable de façon exceptionnelle (ex. : perte de trames vidéo).
- 

## Questions sur les applications

6. Quels sont quelques exemples d'applications temps réel critiques et non critiques ?
    - o Critiques : Contrôle de missile, supervision médicale.
    - o Non critiques : Streaming vidéo, gestion des feux de circulation.
  7. Pourquoi la redondance est-elle importante dans des applications critiques comme celles utilisées dans l'aviation ?
    - o Elle permet de garantir la fiabilité en cas de panne, en utilisant du matériel et du logiciel redondants.
-

## Questions techniques

8. Quels sont les deux principaux paradigmes d'exécution pour les systèmes temps réel ?
  - Systèmes pilotés par le temps (time-driven) : Actions déclenchées à des moments précis basés sur une horloge interne.
  - Systèmes pilotés par les événements (event-driven) : Actions déclenchées par des événements externes.
9. Quelles sont les étapes pour respecter les contraintes temporelles dans un système temps réel ?
  - Gérer efficacement les ressources.
  - Prioriser les tâches urgentes.
  - Répartir les processus via un ordonnanceur.
10. Quels sont les rôles d'un ordonnanceur dans un système temps réel ?
  - Il gère l'exécution des tâches en respectant leurs priorités, échéances, et autres caractéristiques, garantissant ainsi la satisfaction des contraintes temporelles.

---

## Questions sur les algorithmes et outils

11. Quelle est la différence entre les algorithmes d'ordonnancement RM (Rate Monotonic) et EDF (Earliest Deadline First) ?
  - RM : Les priorités sont fixes, et la tâche ayant la période la plus courte est prioritaire.
  - EDF : Les priorités sont dynamiques, et la tâche avec la plus petite échéance est prioritaire.
12. Quels types de langages sont utilisés pour développer des applications temps réel, et pourquoi ?
  - Langages assembleurs : Pour un contrôle précis du matériel.
  - Langages séquentiels avec bibliothèques système (ex. : C, Ada) : Pour la flexibilité et l'efficacité.
  - Langages concurrents de haut niveau (ex. : SCADE, Lustre) : Pour la modélisation et la vérification formelle.
13. Quelles sont les caractéristiques d'un RTOS (système d'exploitation temps réel) comparé à un système d'exploitation classique ?
  - Plus spécialisé, avec une taille réduite.
  - Dédié à une application spécifique.
  - Offre des primitives pour la gestion des ressources et des interruptions.

---

## Questions de synthèse

14. Pourquoi le respect du temps concret (horloge) est-il fondamental dans une application temps réel ?

- Il permet de synchroniser les actions avec des échéances précises, garantissant ainsi que les résultats sont produits au bon moment.

**15. Quels sont les défis spécifiques dans la conception et l'implémentation des systèmes temps réel ?**

- Respect des échéances malgré des ressources limitées.
- Gestion de la complexité due à l'interaction avec des environnements dynamiques.
- Assurance de la fiabilité et de la robustesse.

**1. Quels sont les éléments principaux qui caractérisent un processus Linux ?**

- Les éléments principaux incluent : l'état du processus, la priorité, les signaux en attente et masqués, le PID, le GID, le processus père, les identifiants utilisateur (réel et effectif), la politique d'ordonnancement utilisée, le temps processeur consommé, et la date de création.

**2. Quels sont les états possibles d'un processus Linux ?**

- Les états possibles sont : nouveau, prêt, actif (mode utilisateur ou système), bloqué, suspendu, et zombi.

---

Questions sur les politiques d'ordonnancement

**3. Quelles sont les trois politiques d'ordonnancement sous Linux et à quoi servent-elles ?**

- **SCHED\_FIFO** : Pour les processus temps réel non préemptibles.
- **SCHED\_RR** : Pour les processus temps réel préemptibles avec une exécution limitée par un quantum de temps.
- **SCHED\_OTHER** : Pour les processus ordinaires non temps réel.

**4. Quelle est la différence entre SCHED\_FIFO, SCHED\_RR et SCHED\_OTHER ?**

- **SCHED\_FIFO** exécute les processus sans interruption sauf pour des raisons spécifiques.
- **SCHED\_RR** utilise un tourniquet pour alterner entre les processus en fonction d'un quantum de temps.
- **SCHED\_OTHER** est utilisé pour les processus classiques et dépend également d'un quantum de temps.

**5. Pourquoi les processus SCHED\_OTHER ne s'exécutent-ils que si les files SCHED\_FIFO et SCHED\_RR sont vides ?**

- Parce que **SCHED\_OTHER** est destiné aux processus ordinaires, qui ont une priorité inférieure à celle des processus temps réel (**SCHED\_FIFO** et **SCHED\_RR**).

---

Questions spécifiques à chaque politique

6. Dans quels cas un processus SCHED\_FIFO peut-il être interrompu avant sa fin ?
    - Si un autre processus SCHED\_FIFO plus prioritaire est inséré dans la file.
    - Si le processus effectue une opération d'entrée-sortie.
    - Si le processus abandonne le processeur avec `sched_yield()`.
  7. Combien de temps un processus SCHED\_RR est-il exécuté avant qu'un autre processus ne prenne le relais ?
    - Un processus SCHED\_RR est exécuté pour une durée d'un quantum de temps, typiquement 100 ms sous Linux.
  8. Comment fonctionne la technique du tourniquet (Round Robin) dans le cadre de SCHED\_RR et SCHED\_OTHER ?
    - Les processus sont exécutés à tour de rôle pour un quantum de temps fixe. Lorsqu'un processus termine son quantum, il est replacé en fin de file et le prochain processus est exécuté.
- 

Questions techniques

9. Quelle est la plage de priorités pour les processus SCHED\_FIFO et SCHED\_RR ?
    - La priorité est comprise entre 1 et 99 pour ces deux politiques.
  10. Quels appels système permettent de définir ou d'obtenir la politique d'ordonnancement et la priorité d'un processus ?
    - `sched_setscheduler()` pour définir la politique et la priorité.
    - `sched_getscheduler()` pour obtenir la politique d'un processus.
    - `sched_get_priority_min()` et `sched_get_priority_max()` pour obtenir les priorités minimales et maximales.
  11. À quoi sert la fonction `sched_rr_get_interval()` dans le contexte de SCHED\_RR ?
    - Elle permet d'obtenir la durée du quantum de temps pour les processus SCHED\_RR.
- 

Questions pratiques

12. Comment modifier la politique d'ordonnancement d'un processus avec `sched_setscheduler()` ?
  - On utilise la fonction avec les paramètres suivants : PID du processus, la nouvelle politique (ex. : SCHED\_RR), et une structure contenant la priorité.
  - Exemple :  
c  
Copier le code

```
struct sched_param p;
```
  - `p.sched_priority = 16;`

- `sched_setscheduler(getpid(), SCHED_RR, &p);`

13.

14. Donnez un exemple d'utilisation de `sched_getscheduler()` pour afficher la politique d'un processus en cours.

- Exemple :

- `C`

Copier le code

```
int s = sched_getscheduler(getpid());  
switch (s) {  
    case SCHED_FIFO: printf("SCHED_FIFO\n"); break;  
    case SCHED_RR: printf("SCHED_RR\n"); break;  
    case SCHED_OTHER: printf("SCHED_OTHER\n"); break;  
}
```

15.

16. Pourquoi est-il important de connaître les priorités minimales et maximales pour chaque politique d'ordonnancement ?

- Cela permet de configurer correctement la priorité d'un processus pour garantir qu'il respecte les exigences de sa politique et qu'il interagit correctement avec les autres processus.

17. Qu'est-ce que l'ordonnancement dans le contexte des tâches temps réel ?

- L'ordonnancement est la planification de l'exécution des tâches, déterminant l'ordre d'allocation du processeur pour respecter les contraintes temporelles des applications.

18. Qu'est-ce qu'une tâche dans le cadre des systèmes temps réel ?

- Une tâche est une entité d'exécution, une instance dynamique d'un programme exécutable (processus, thread).

19. Quelle est la différence entre un ordonnancement préemptif et non préemptif ?

- L'ordonnancement préemptif permet de réquisitionner le processeur pour une tâche plus prioritaire, tandis que l'ordonnancement non préemptif exige qu'une tâche termine avant qu'une autre puisse être exécutée.

---

## Questions sur les modèles et types de tâches

4. Quels sont les deux principaux types de tâches dans les systèmes temps réel ?

- Tâches périodiques : Réveil régulier à intervalles fixes (P unités de temps).
- Tâches apériodiques : Réveil aléatoire en réponse à des événements imprévisibles.

5. Comment modélise-t-on une tâche périodique stricte ?

- Une tâche périodique stricte est décrite par :  
 $T(r_0, C, R, P)T(r_0, C, R, P)T(r_0, C, R, P)$ , où :
  - $r_0$  : date initiale de réveil,

- CCC : durée d'exécution maximale,
- RRR : délai critique (temps jusqu'à l'échéance),
- PPP : période d'exécution.

**6. Comment modélise-t-on une tâche apériodique stricte ?**

- Une tâche apériodique stricte est décrite par :
- Tap(r,C,R)T\_{ap}(r, C, R)Tap(r,C,R), où :
- rrr : date aléatoire de réveil,
  - CCC : durée d'exécution maximale,
  - RRR : délai critique.
- 

Questions sur les algorithmes d'ordonnancement

**7. Quels sont les principaux algorithmes pour les tâches périodiques ?**

- Rate Monotonic (RM) : Priorité inversement proportionnelle à la période.
- Deadline Monotonic (DM) : Priorité inversement proportionnelle à l'échéance relative.
- Earliest Deadline First (EDF) : Priorité dynamique basée sur l'échéance absolue la plus proche.

**8. Quelle est la condition de faisabilité pour l'algorithme RM ?**

- Pour un ensemble de nnn tâches périodiques à échéance sur requête, la condition suffisante est :  $\sum_{i=1}^n C_i P_i \leq n \cdot (2^{1/n} - 1)$

**9. Comment fonctionne l'algorithme EDF pour les tâches périodiques et non périodiques ?**

- L'algorithme EDF exécute en priorité la tâche ayant l'échéance absolue la plus proche. C'est un algorithme préemptif et optimal pour les tâches indépendantes.

**10. Qu'est-ce que la laxité et quel algorithme l'utilise ?**

- La laxité est définie comme  $\text{laxité} = \text{échéance} - \text{temps de calcul restant}$ . L'algorithme Least Laxity First (LLF) priorise les tâches avec la plus petite laxité.
- 

Questions sur les contraintes de ressources

**11. Qu'est-ce qu'une ressource critique dans le contexte de l'ordonnancement ?**

- Une ressource critique est une ressource partagée entre plusieurs tâches, nécessitant une gestion d'exclusion mutuelle pour éviter les conflits.

**12. Qu'est-ce que l'inversion de priorité et comment peut-on la résoudre ?**

- L'inversion de priorité se produit lorsqu'une tâche de haute priorité est bloquée par une tâche de faible priorité utilisant une ressource partagée. Elle peut être résolue par :
  - Protocole d'héritage de priorité (PIP).

■ Protocole de priorité plafond (PCP).

---

13. Qu'est-ce que l'analyse d'ordonnançabilité ?

- C'est une méthode pour vérifier que les tâches respectent leurs contraintes temporelles. Elle peut être réalisée hors ligne (avant exécution) ou en ligne (pendant exécution).

14. Quelle est la différence entre les conditions nécessaire, suffisante, et nécessaire et suffisante dans l'analyse d'ordonnançabilité ?

- Condition nécessaire : Si elle n'est pas vérifiée, le système n'est pas faisable.
- Condition suffisante : Si elle est vérifiée, le système est faisable.
- Condition nécessaire et suffisante : Si elle est vérifiée, le système est faisable, et si elle ne l'est pas, le système ne l'est pas.

MAGUEMOUN SAMY