Systèmes Temps-Réel et Systèmes Embarqués

Problématique 1

Certains problèmes sont très liés au temps :

- un distributeur de billets ne doit pas mettre 5 minutes à délivrer les billets
- une balance ne doit pas peser en 30 secondes
- un radar ne doit pas mettre 2 secondes à réagir
- un système de freinage ABS ne doit pas mettre plus de 150ms pour acquérir l'information et 1s pour réagir

Problématique 2

Satisfaire les contraintes de temps des transactions temps réel en allouant de façon efficace :

- Ressources CPU,
- Ressources de communication,
- L'accès aux données.

Problématique 3

- La terminologie temps réel (real-time) cache la notion de temps de réaction lié à la dynamique du procédé industriel à contrôler.
- Systèmes réactifs : Le rythme est donc déterminé par l'environnement et non le système.
- Un système fonctionne en temps réel s'il est capable d'absorber toutes les informations d'entrée sans qu'elles soient trop vieilles pour l'intérêt qu'elles représentent et de calculer des commandes à temps pour qu'elles aient un sens.
  Une absence de réaction peut amener dans une situation catastrophique. On parle alors de systèmes critiques.

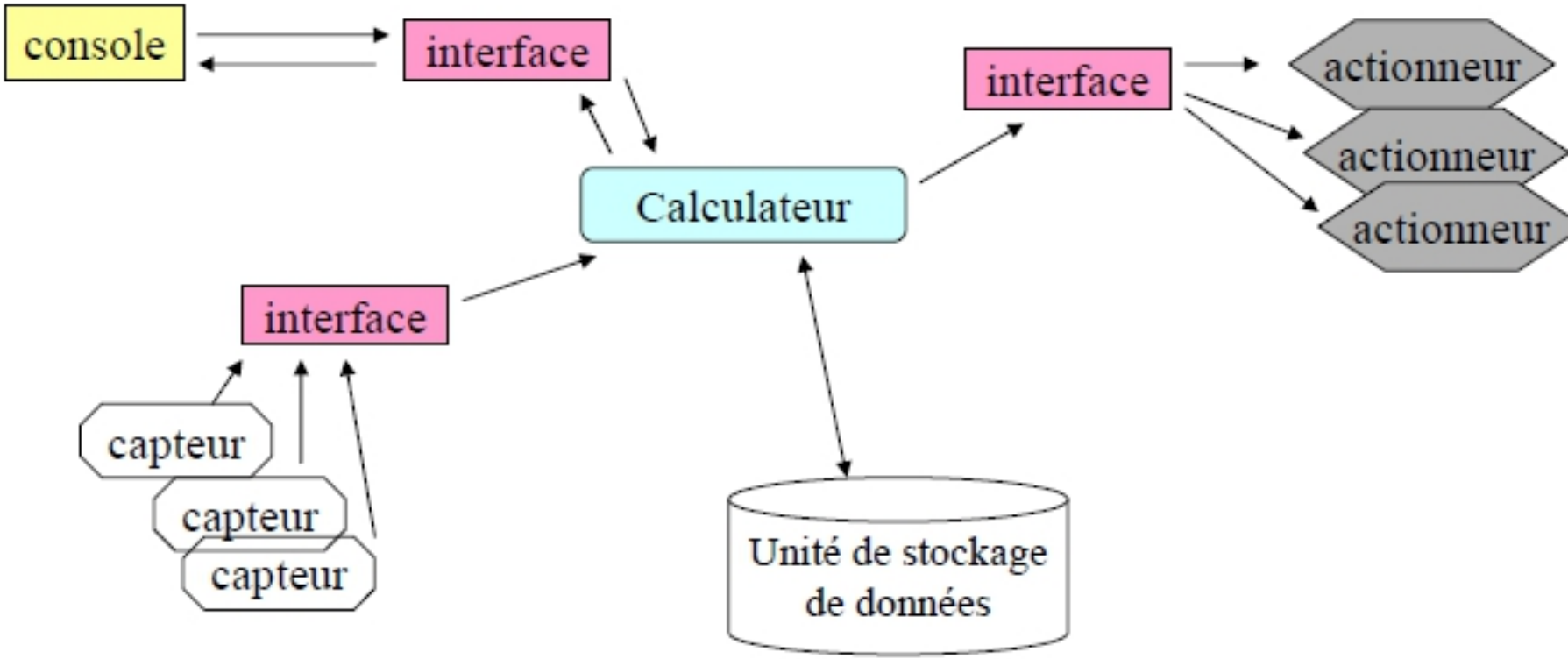
Problématique 4

Temps réel $\neq$ aller vite : satisfaire les contraintes temporelles (les contraintes de temps dépendent de l'application et de l'environnement alors que la rapidité dépend de la technologie utilisée, celle du processeur par exemple).
De quel temps s'agit-il ?

- Temps de l'environnement = temps chronométrique (le temps réel)
- Temps du système informatique = temps chronologique, constitué de la suite des événements ou des instructions du systèmes (pas/étape du temps réel vu par le système)

$\Longrightarrow$ Exigence du temps réel = concordance entre le temps chronologique de l'environnement et le temps chronométrique du système

Exemple : un système temps réel embarqué



Classification

- **Temps réel dur ou critique (hard real-time) :** le non respect des contraintes temporelles entraîne la faute du système.
  Ex. : contrôle de trafic aérien, système de conduite de missile, etc.
- **Temps réel souple (soft real-time) :** le respect des échéances est important mais le non respect des échéances ne peut occasionner de graves conséquences.
  Ex. : projection vidéo (décalage entre le son et l'image).
- **Temps réel ferme (firm real-time) :** temps réel souple avec le manquement occasionnel des échéances.
  Ex. : projection vidéo (perte de quelques trames d'images).

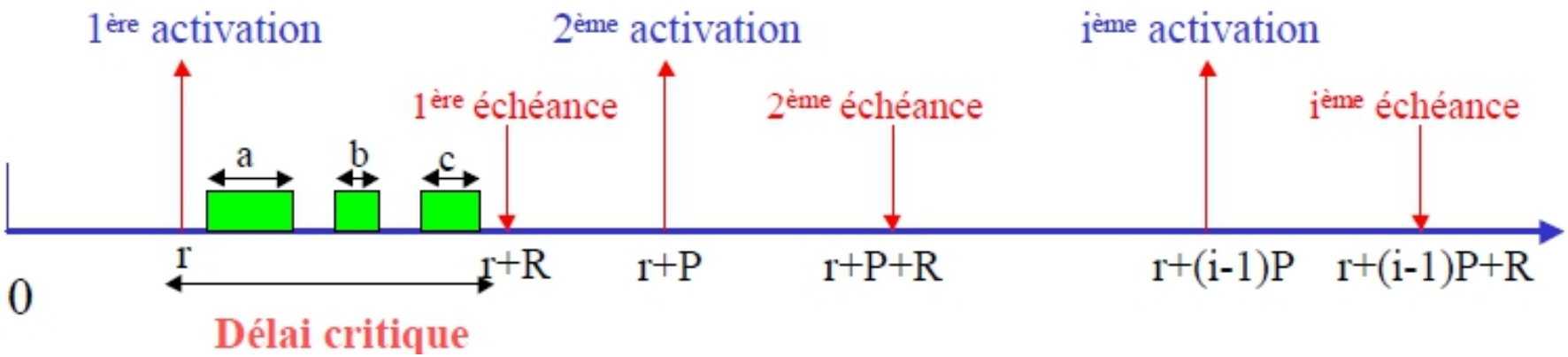
Définitions 1

- Tâches **dépendantes** ou **indépendantes** : les tâches indépendantes ne partagent que le processeur et les tâches dépendantes partagent d'autres ressources ou sont reliées par des contraintes de précédence,
- Ordonnancement **préemptif** ou **non préemptif** : un ordonnanceur **préemptif** peut interrompre une tâche au profit d'une tâche plus prioritaire et un ordonnanceur non préemptif n'arrête pas l'exécution de la tâche courante,
- Ordonnancement **hors ligne** ou **en ligne** : un ordonnancement hors ligne est pré-calculé avant exécution puis est exécuté avec ou sans préemption et un ordonnancement en ligne décide dynamiquement avec ou sans préemption de l'exécution des tâches.

Définitions 2

- Priorité
  - élément utilisé par l'ordonnanceur pour élire une tâche
  - Priorité statique/priorité dynamique
- Ordonnancement optimal : algorithme qui produit un ordonnancement pour tout ensemble de tâches ordonnancées (si un algorithme le fait, lui le fait aussi)
- Test d'ordonnancement : formule qui fournit une condition nécessaire et/ou suffisante pour qu'un algorithme satisfasse les contraintes temporelles d'un ensemble de tâches

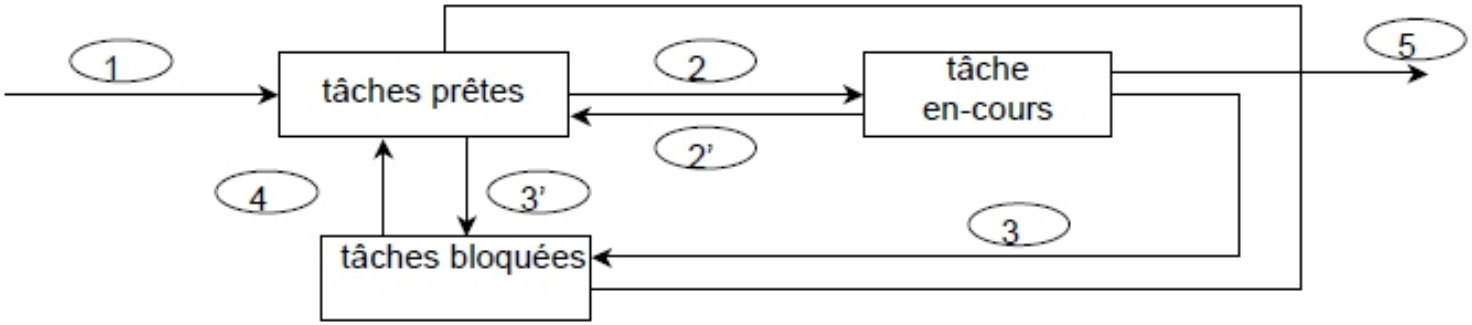
Modèle simple (tâches périodiques)



- r : date de première activation
- C : pire durée d'exécution (WCET) = Max(a)+Max(b)+Max(c)
- R : échéance,
- P : période.

Ordonnanceur

Rôle : choix de la tâche à exécuter parmi les tâches prêtes,



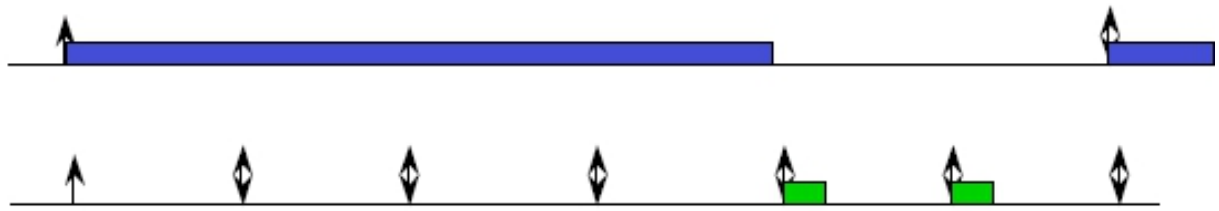
1 : création de la tâche
2 : élection
2' : préemption
3 : demande de ressource occupée, attente événement non présent, auto-suspension
3' : suspention
4 : ressource libérée, évenement arrivé, réactivation
5 : fin

Pourquoi ordonnancer ? 1

Soit une application de contrôle composée d'une activité de contrôlr moteur et d'une activité de monitoring implémentées par deux tâches périodiques (contrôle et display). l'architecture s'appuie sur un micro-contrôler (40 Mhz).

Tâche *Control*
Période= 60 ms
Durée = 40 ms

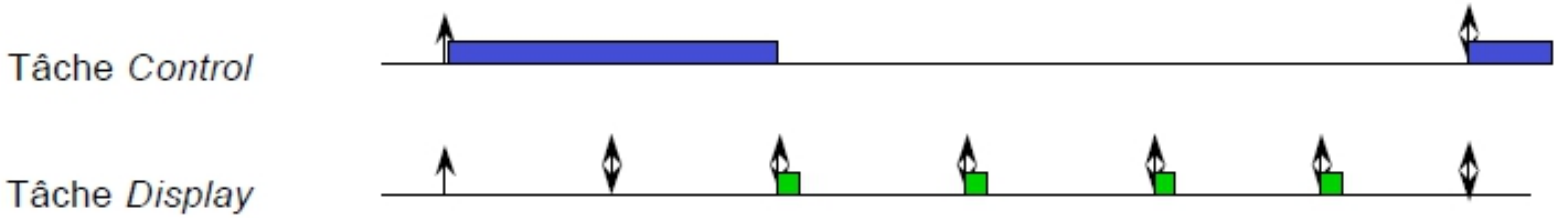
Tâche *Display*
Période = 10 ms
Durée = 2 ms



$\Longrightarrow$ L'information clignote à l'écran

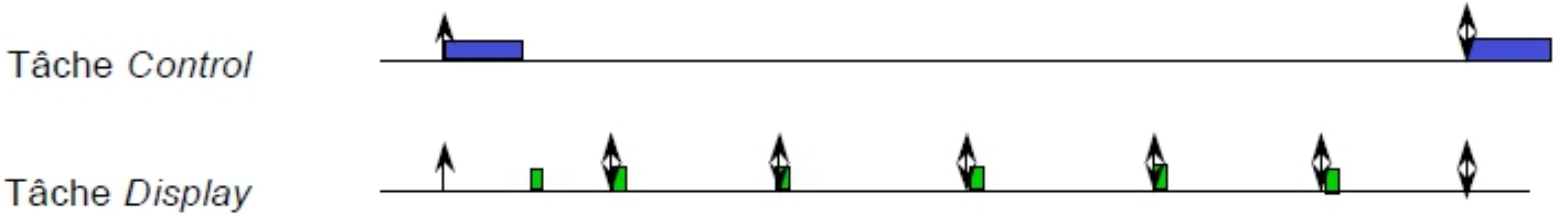
Pourquoi ordonnancer ? 2

- Processeur plus rapide : 80 Mhz



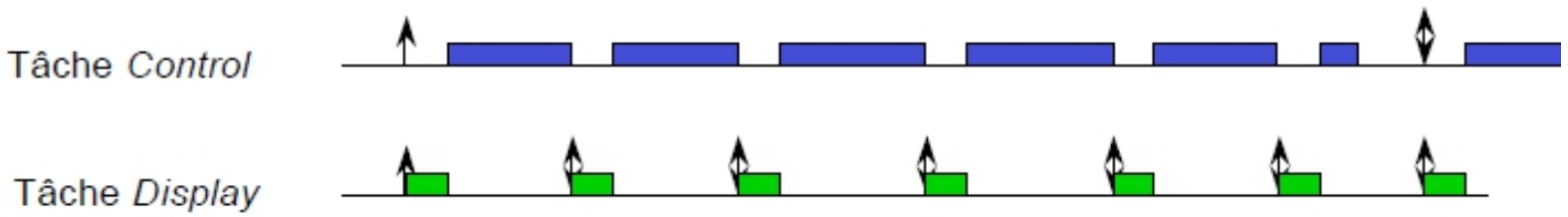
$\Longrightarrow$ L'information clignote à l'écran

- Processeur encore plus rapide : 320 Mhz



$\Longrightarrow$ L'information s'affiche mais solution très couteuse

Pourquoi ordonnancer ? 3

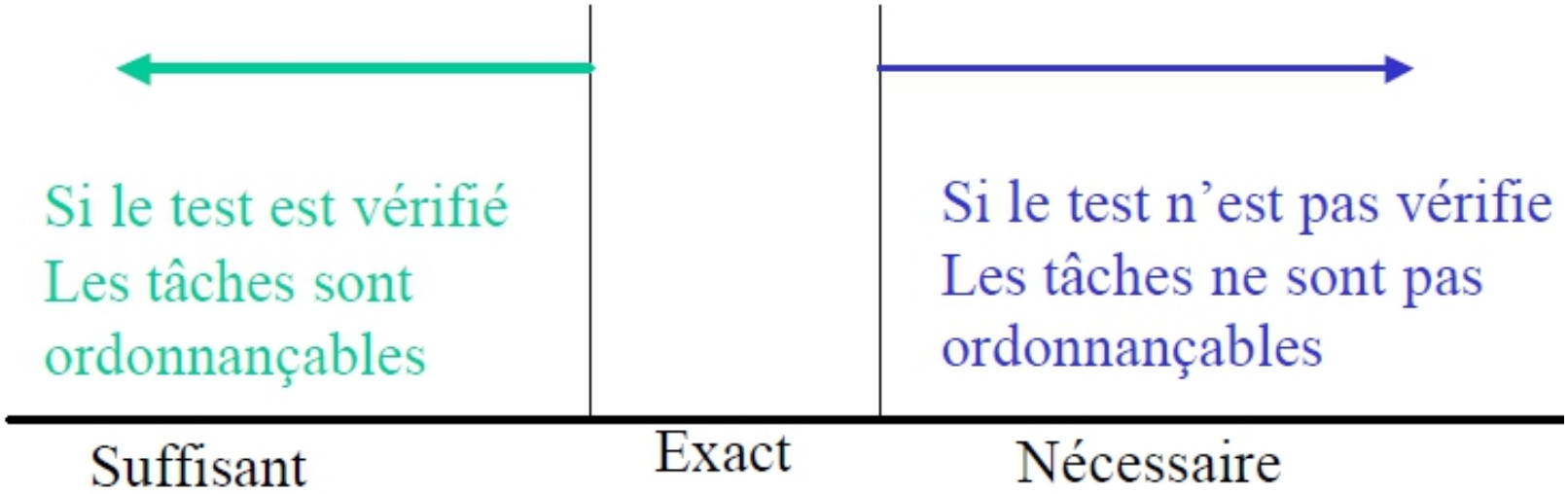


Solution temps réel :

- Meilleur ordonnancement,
- La priorité est liée à l'urgence temporelle
- Processeur bas coût

Test d'ordonnançabilité

Déterminer si il existe un algorithme qui satisfasse les contraintes temporelles d'un système



$\Longrightarrow$ Une condition nécessaire et suffisante pour l'ordonnancement d'une seule tâche périodique est : $C \leq T$

Panorama des algorithmes d'ordonnancement

- Ordonnancements de tâches périodiques
  - Ordonnancement cyclique
  - Ordonnancements préemptif à priorité fixe
  - Ordonnancements préemptif à priorité dynamique
- Ordonnancements de tâches apériodiques
  - Serveur à scrutation
  - Serveur différé

Le modèle :

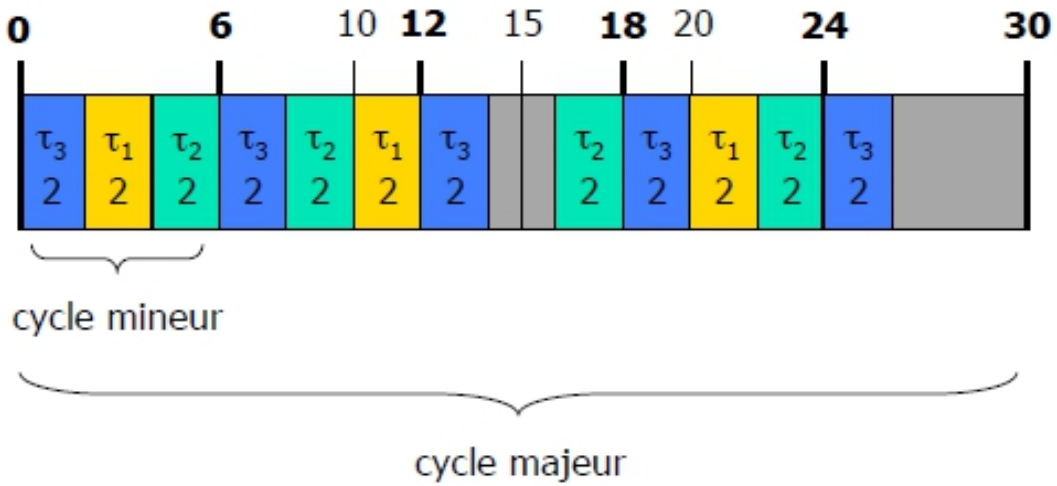
- Un nombre fixé de tâches
- toutes les tâches sont périodiques
- tâches indépendantes,
- temps de changement de contexte supposé égale à zéro.

Ordonnancement Cyclique

- Cycle majeur = PPCM des périodes
- Cycle mineur = bloc non-préemptible
- Le cycle mineur divise le cycle majeur
- Un ordonnanceur cyclique boucle sur le cycle majeur en exécutant la séquence de cycles mineurs
- Le cycle mineur correspond à un point de contrôle permettant de vérifier le respect des contraintes

Ordonnancement Cyclique

	période	échéance	calcul	utilisation
τ_1	10	10	2	0,200
τ_2	15	15	4	0.267
τ_3	6	6	2	0.333



Ordonnancement Cyclique

Avantages

- Mise en oeuvre efficace
- pas besoin de protection de régions critiques

Inconvénients

- Cycles longs difficiles à gérer,
- Manque de flexibilité : traitement de tâches apériodiques,

...

Ordonnancement par priorité statique
Rate Monotonic Scheduling

Hypothèses

- Activation en début de période (si = 0)
- Echéance en fin de période (Di=Ti)

Principe

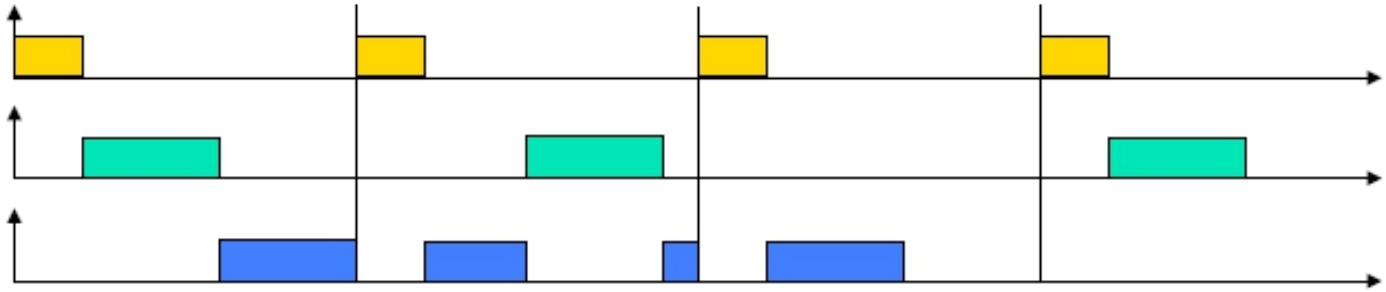
- Une activation de tâche réveille l'ordonnanceur
- Il choisis la tâche éligible de plus courte période

Test d'ordonnançabilité

- Condition suffisante $U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{\frac{1}{n}} - 1)$
  Quand $n \to \infty$ $U \to \lim(2) = 0.69$
- Condition nécessaire et suffisante en $O(n^2)$

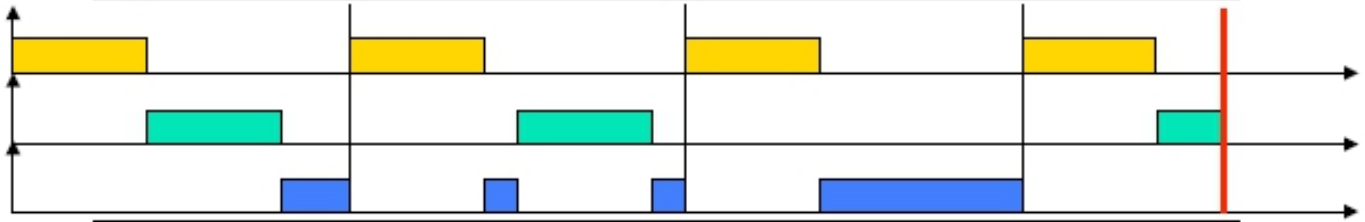
Ordonnancement par priorité statique
Rate Monotonic Scheduling - Exemple 1

$3 \times (2^{1/3} - 1) \approx 0.78$	période	calcul	utilisation
τ_1	10	2	0.200
τ_2	15	4	0.267
τ_3	36	12	0.333

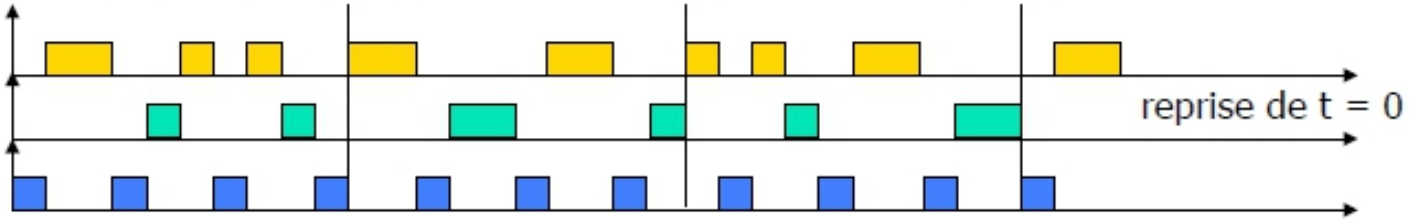


Ordonnancement par priorité statique
Rate Monotonic Scheduling - Exemple 2

$3x(2^{1/3}-1)=0.78$	période	calcul	utilisation
τ_1	10	4	0.400
τ_2	15	4	0.267
τ_3	36	12	0.333



$3x(2^{1/3}-1)=0.78$	période	calcul	utilisation
$\tau'_1 = \tau_1 / 2$	**5**	**2**	**0.400**
τ'_2	15	4	0.267
$\tau'_3 = \tau_3 / 12$	**3**	**1**	**0.333**



Ordonnancement par priorité statique
Rate Monotonic Scheduling

Avantages

- Simplicité de mise en oeuvre
- Optimal pour les ordonnancements à priorité statique
- Répandu dans les exécutifs classiques

Inconvénients

- Surdimensionnement possible du système

Rate Monotonic Scheduling
Calcul du temps de réponse

Principe : pour une tâche i, on cherche à évaluer, au pire cas, son temps d'exécution + son temps d'attente lorsque des tâches plus prioritaires s'exécutent. Où encore :

$$r_i = C_i + I_i = C_i + \sum_{j \in hp(i)} \lceil \frac{r_i}{Tj} \rceil Cj$$

où I_i est le temps pendant lequel la tâche i est suspendue par des tâches plus prioritaires et $hp(j)$ est l'ensemble des tâches de plus forte priorité que j.

Rate Monotonic Scheduling
Calcul du temps de réponse - Technique de calcul

De façon itérative : $w_i^{n+1} = C_i + \sum_{j \in hp(i)} \lceil \frac{w_i^n}{Tj} \rceil Cj$
on démarre avec $w_i^0 = C_i$.
Condition d'arrêt :

- Echec si $w_i^n > T_i$
- Réussite si $w_i^{n+1} = w_i^n$

Rate Monotonic Scheduling
Calcul du temps de réponse - Exemple

T1=7 ; C1=3 ; T2=12 ; C2=2 ; T3=20 ; C3=5

- $w_1^0 = 3$
- $w_2^0 = 2$
- $w_2^1 = 2 + \lceil \frac{2}{7} \rceil 3 = 5$
- $w_2^1 = 2 + \lceil \frac{5}{7} \rceil 3 = 5 \Rightarrow r_2 = 5$
- $w_3^0 = 5$
- $w_3^1 = 5 + \lceil \frac{5}{7} \rceil 3 + \lceil \frac{5}{12} \rceil 2 = 10$
- $w_3^2 = 5 + \lceil \frac{10}{7} \rceil 3 + \lceil \frac{10}{12} \rceil 2 = 13$
- $w_3^3 = 5 + \lceil \frac{13}{7} \rceil 3 + \lceil \frac{13}{12} \rceil 2 = 15$
- $w_3^4 = 5 + \lceil \frac{15}{7} \rceil 3 + \lceil \frac{15}{12} \rceil 2 = 18$
- $w_3^1 = 5 + \lceil \frac{18}{7} \rceil 3 + \lceil \frac{18}{12} \rceil 2 = 18 \Rightarrow r_3 = 5$

Ordonnancement par priorité dynamique
Earliest Deadline First

Hypothèses

- Activation en début de période (si = 0)
- Echéance en fin de période (Di=Ti)

Principe

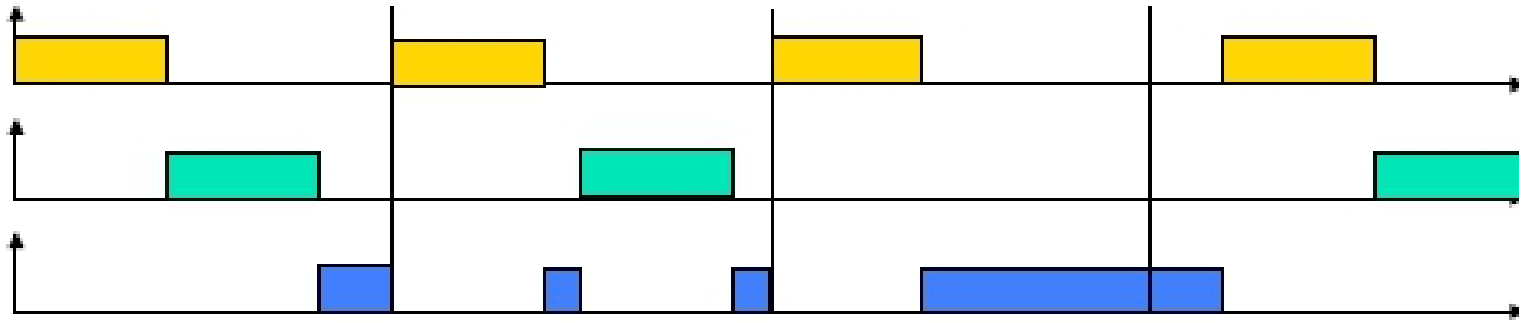
- Une activation de tâche réveille l'ordonnanceur
- la tâche qui a l'échéance la plus proche occupe le CPU
- Objectif : obtenir une meilleure occupation du processeur

Test d'ordonnançabilité

- Condition nécessaire et suffisante : $U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1$

Ordonnancement par priorité dynamique
Earliest Deadline First

	période	calcul	utilisation
τ_1	10	4	0.400
τ_2	15	4	0.267
τ_3	36	12	0.333



Ordonnancement par priorité dynamique
Earliest Deadline First

Avantages

- Utilisation possible de 100% du processeur
- Optimal pour les ordonnancements à priorité dynamique

Inconvénients

- Légère complexité de la mise en oeuvre
- Moins répandu dans les exécutifs que RMS

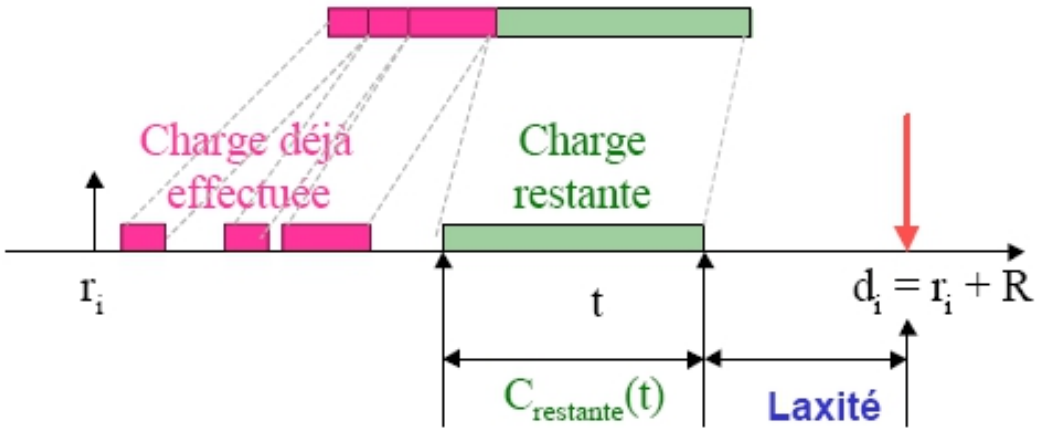
Ordonnancement par priorité dynamique
Least Laxity First

Hypothèses

- Similaire à celles d'EDF

Principe

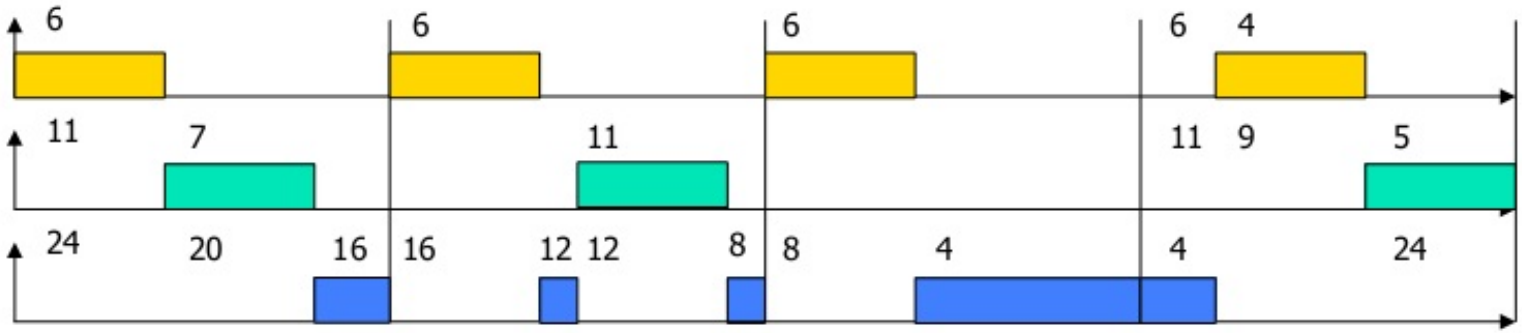
- Une activation de tâche réveille l'ordonnanceur
- choisir la tâche éligible de moindre marge
- marge = échéance - temps de calcul restant - temps courant



- Condition nécessaire et suffisante : $U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1$

Ordonnancement par priorité dynamique
Least Laxity First - Exemple

	période	calcul	utilisation
τ_1	10	4	0.400
τ_2	15	4	0.267
τ_3	36	12	0.333



Ordonnancement par priorité dynamique
Least Laxity First

Avantages

- Meilleur qu'EDF dans le cas de multi-processeur

Inconvénients

- Forte complexité de la mise en oeuvre (Difficulté du calcul du temps de calcul restant)
- Nombre de préemptions engendrées supérieur

Ordonnancement de tâches apériodiques

Définitions

- Les tâches apériodiques sont activées à des instants aléatoires
- Les tâches sporadiques se caractérisent par un délai minimum entre deux activations

Principes

- Des tâches apériodiques doivent être intégrées dans un ordonnancement de tâches périodiques
- Les tâches périodiques doivent respectées leurs échéances

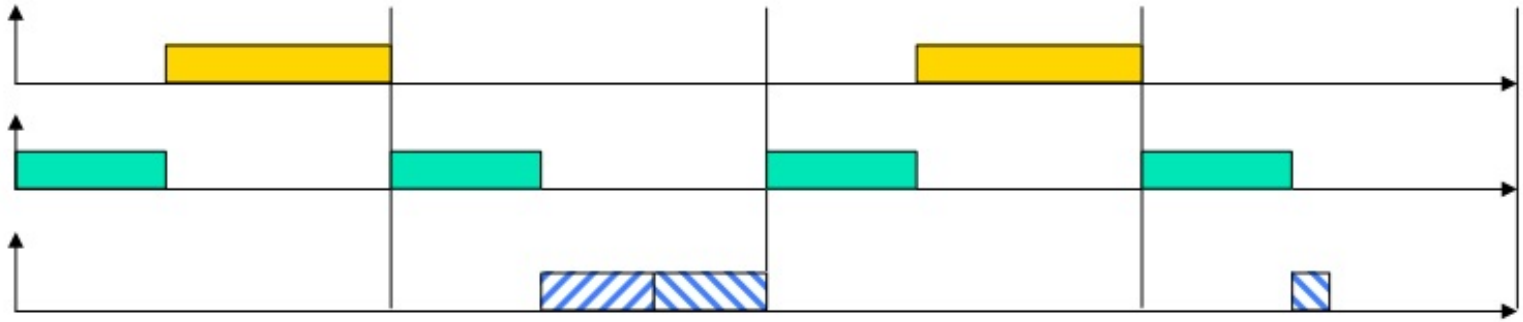
Ordonnancement de tâches apériodiques
Serveur en arrière plan

Principes

- Les tâches apériodiques sont traitées séquentiellement par un serveur de faible priorité
- Le serveur n'a pas de temps de calcul associé ou de capacité (faible priorité)
- L'absence de capacité vient du fait que le serveur remplit les trous dans l'ordonnancement

Ordonnancement de tâches apériodiques
Serveur en arrière plan - Exemple

	période/activation	calcul	priorité	utilisation/réponse
événement ε_1	7	3		17
événement ε_2	11	4		35
tâche τ_1	20	6	3	0,300
tâche τ_2	10	4	2	0,400
serveur en arrière plan	*	*	10	*



Ordonnancement de tâches apériodiques
Serveur en arrière plan

Avantages

- Simplicité de mise en oeuvre

Inconvénients

- On ne peut prédire les échéances de traitement des tâches apériodiques
- Les tâches apériodiques peuvent être critiques

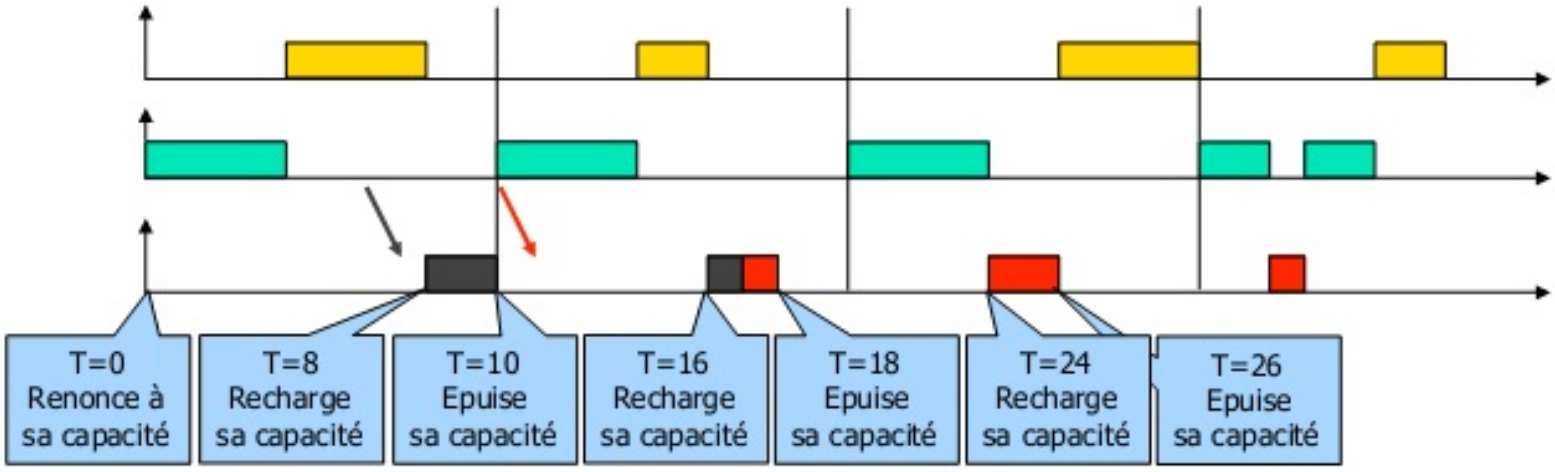
Ordonnancement de tâches apériodiques
Serveur à scrutation

Principes

- Les tâches apériodiques sont traitées séquentiellement par un serveur de forte priorité
- Le serveur dispose d'une capacité et d'une période
- La capacité est réallouée toutes les périodes
- Le temps consommé à traiter une tâche apériodique est débité sur la capacité
- Le serveur devenu actif traite toutes les tâches apériodiques dans la limite de sa capacité
- Le serveur devenu inactif (absence de tâche) perd sa capacité jusqu'à la prochaine ré-allocation

Ordonnancement de tâches apériodiques
Serveur à scrutation - Exemple

	période/activation	calcul	priorité	utilisation/réponse
événement ε_1	7	3		17
événement ε_2	11	4		33
tâche τ_1	20	6	3	0,300
tâche τ_2	10	4	2	0,400
serveur à scrutation	8	2	1	0,250



Ordonnancement de tâches apériodiques
Serveur à scrutation

Avantages

- Simplicité de mise en oeuvre

Inconvénients

- En relâchant sa capacité, le serveur épuise le temps alloué pour des tâches à venir
- Mauvais temps de réponse (Le traitement peut s'étendre sur plusieurs périodes même pour une tâche de courte durée)

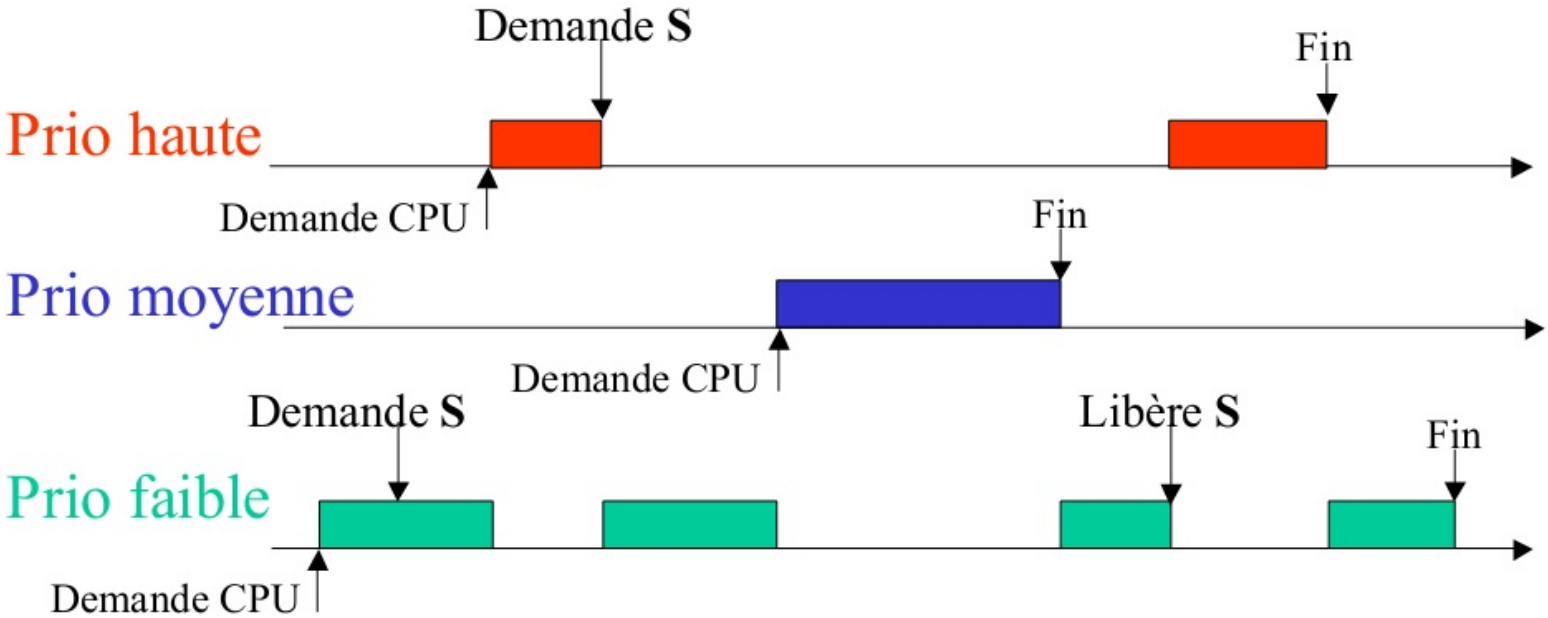
Ordonnancement de tâches apériodiques
Serveur différé

Principes

- Les tâches apériodiques sont traitées séquentiellement par un serveur de forte priorité
- Le serveur dispose d'une capacité et d'une période
- La capacité est réallouée toutes les périodes
- Le temps consommé à traiter une tâche apériodique est débité sur la capacité
- Le serveur devient actif lorsqu'une tâche apériodique est à traiter et que sa capacité n'est pas épuisée

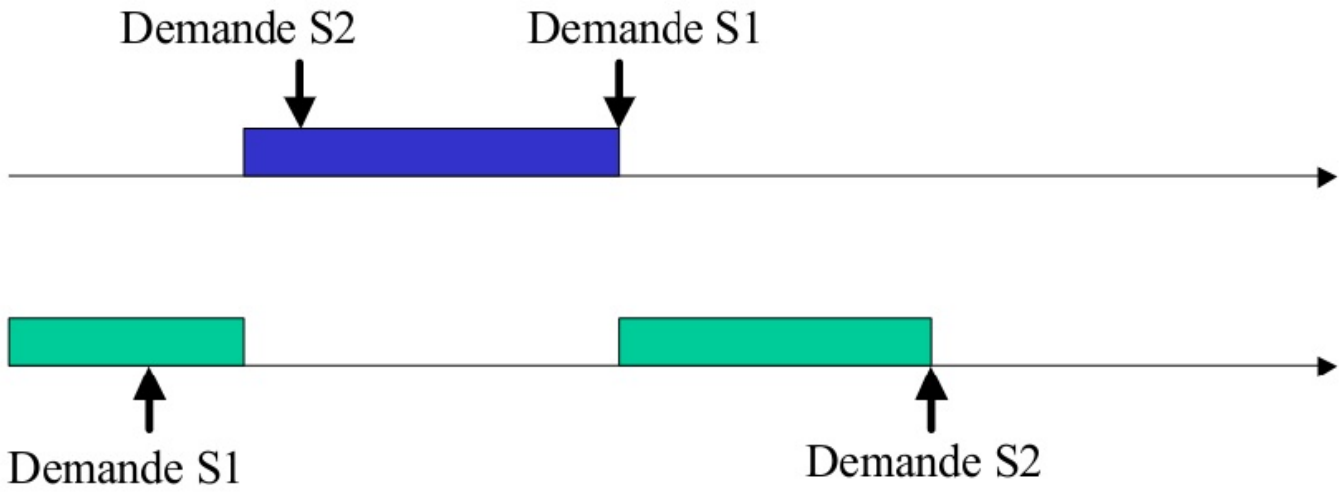
Partage de ressources (contrainte de dépendance) Blocage et Ordonnancement

- Accéder à une ressource c'est éventuellement devoir attendre qu'elle se libère = borne sur le temps de blocage
- Problème de l'inversion de priorité : S'il y a exclusion mutuelle, l'approche préemptive basée sur les priorités conduit à des situations où une tâche plus prioritaire ne peut s'exécuter à cause d'une tâche moins prioritaire



Partage de ressources (contrainte de dépendance) Blocage et Ordonnancement

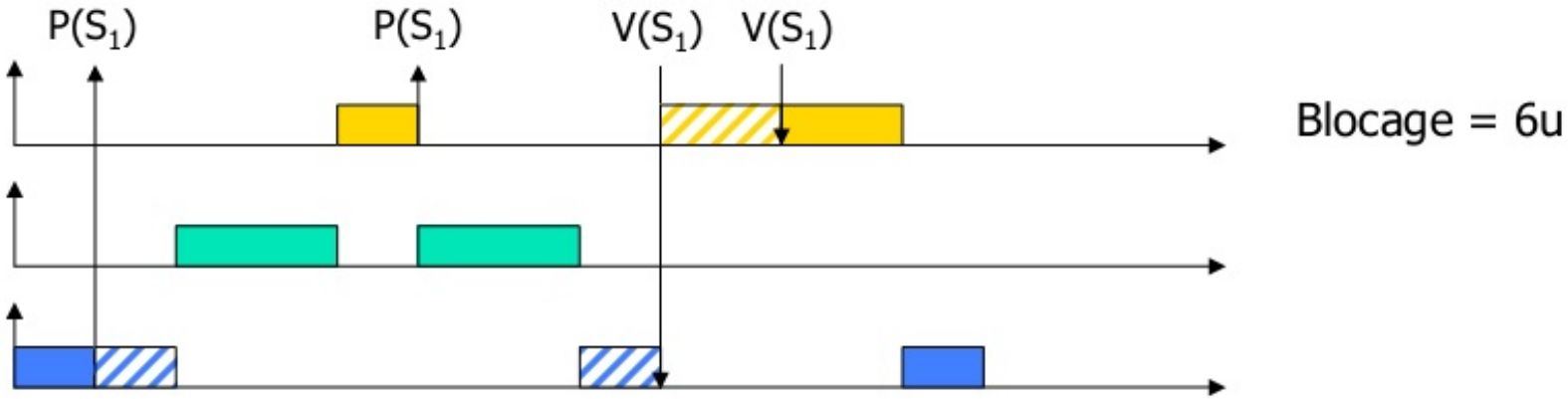
- Interblocage (deadlock)

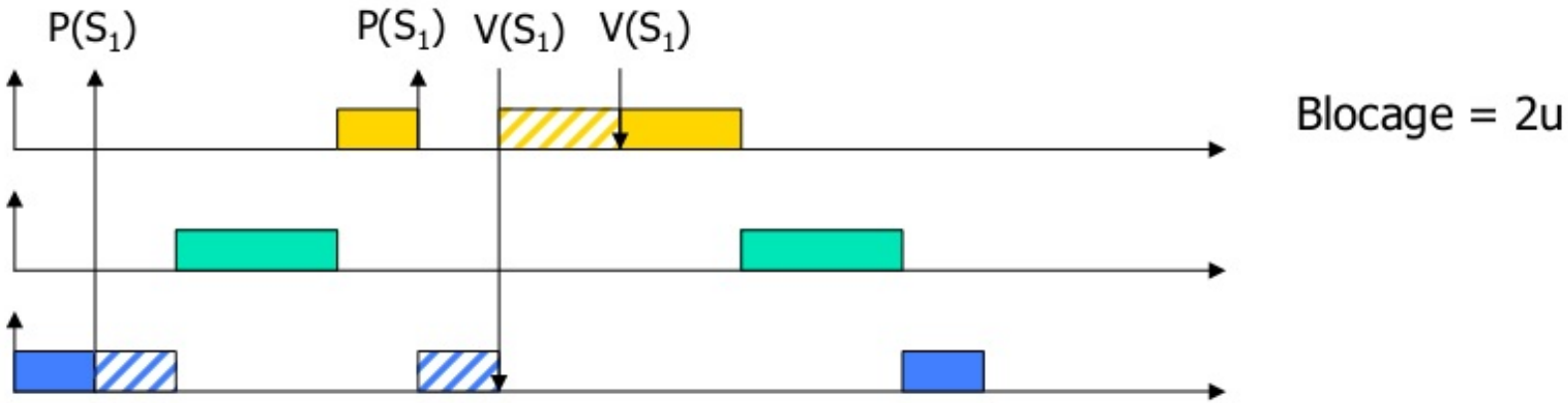


- L'objectif consiste alors à éviter les deadlocks et de réduire le blocage en instaurant des politiques de partage de ressources adéquates

Partage de ressources
Héritage de priorité simple

- Principe : rehausser la priorité du processus bloquant à celle du processus qu'il bloque





Partage de ressources
Héritage de priorité simple

Avantages

- Le temps de blocage dû à la situation d'inversion de priorité se limite à l'utilisation de sémaphore par la tâche de faible priorité

Inconvénients

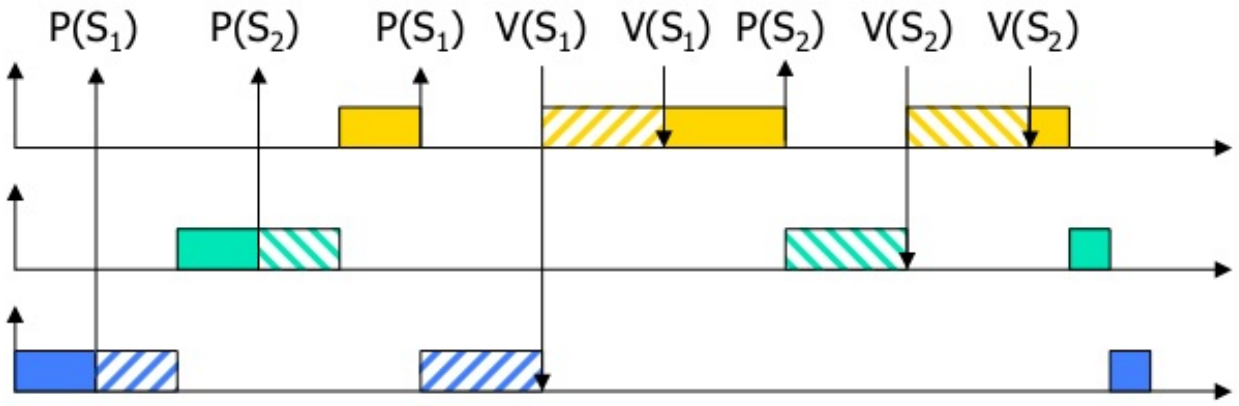
- En cas d'accès à plusieurs sémaphores, la tâche de forte priorité va enchaîner les temps de blocage
- Dans le cas d'accès à un sémaphore par plusieurs tâches, les élévations de priorité vont s'enchaîner
- Les interblocages restent possibles

Partage de ressources
Protocole avec priorité plafonnées

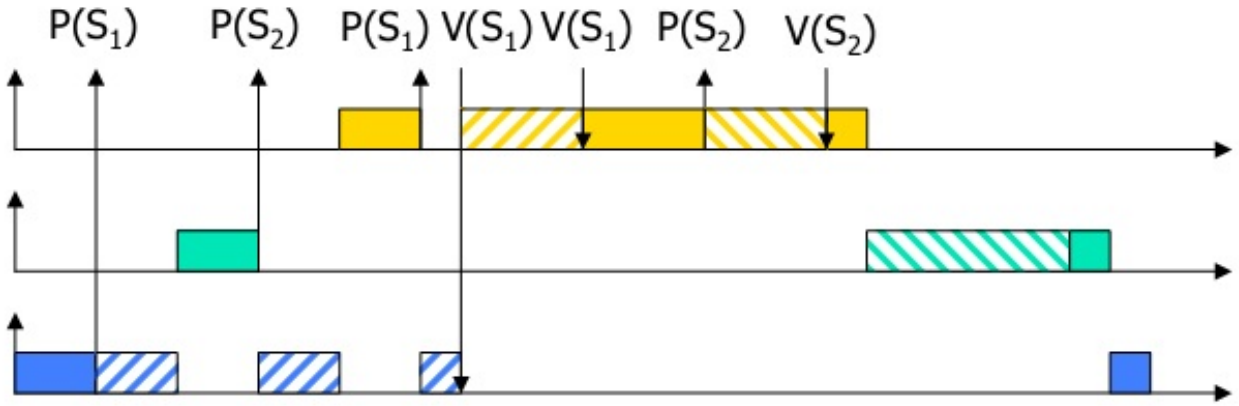
Principes :

- On définit une valeur statique associée à un sémaphore qui est la valeur de priorité max des tâches qui l'utilisent
- Une tâche n'entre en section critique (obtient le sémaphore) que si sa priorité courante est supérieure à toutes les priorités des sémaphores en cours d'utilisation lors de sa demande
- En section critique la tâche hérite de la prioritéde la tâche la plus prioritaire qu'elle bloque

Partage de ressources
Protocole avec priorité plafonnées

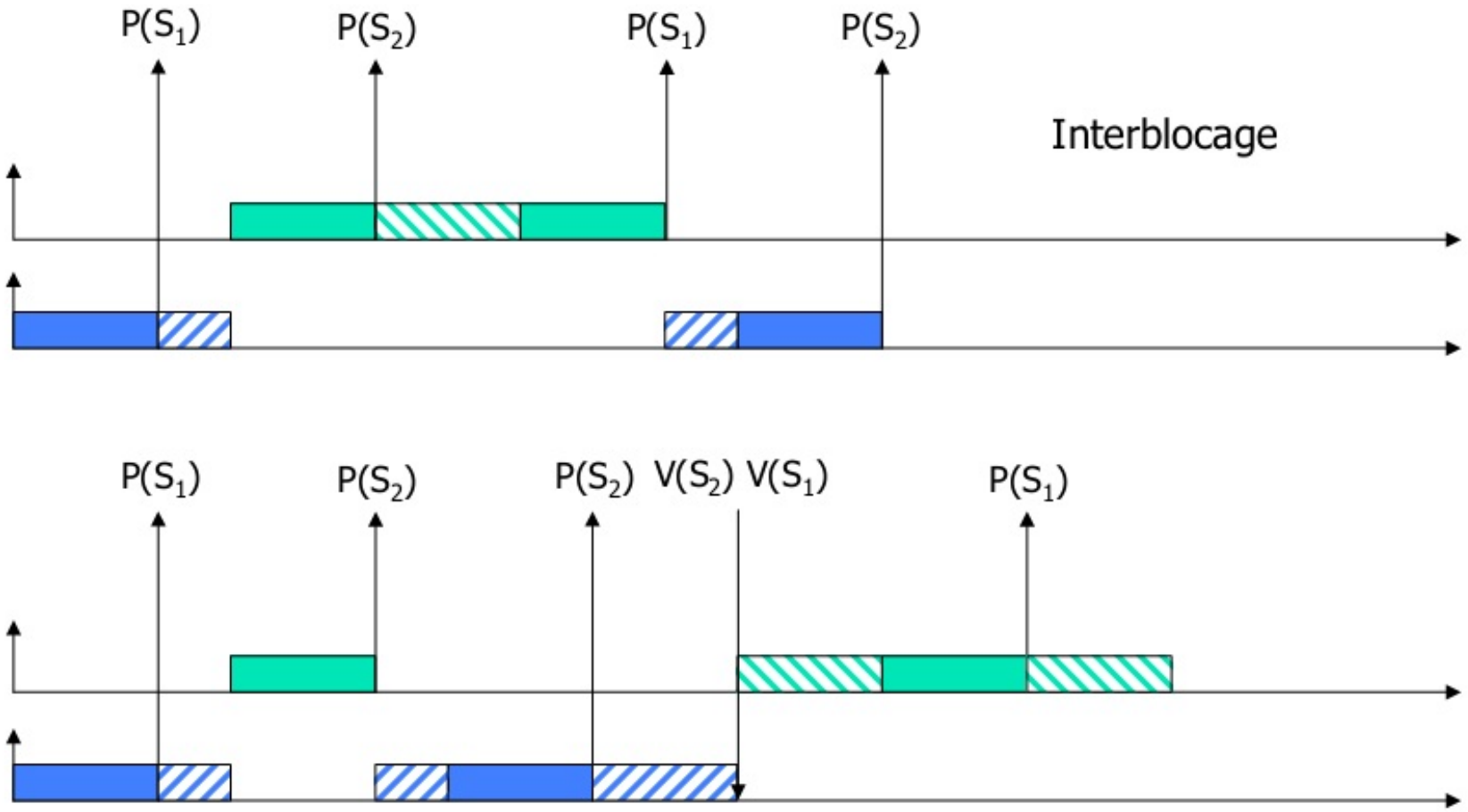


Blocage = 6u



Blocage = 1u

Partage de ressources
Protocole avec priorité plafonnées



Partage de ressources
Protocole avec priorité plafonnées

Avantages

- Pas d'enchaînement de temps de blocage
- Pas d'enchaînement d'élévations de priorité
- Pas de risques d'interblocages

Inconvénients

- Forte complexité de la mise en oeuvre