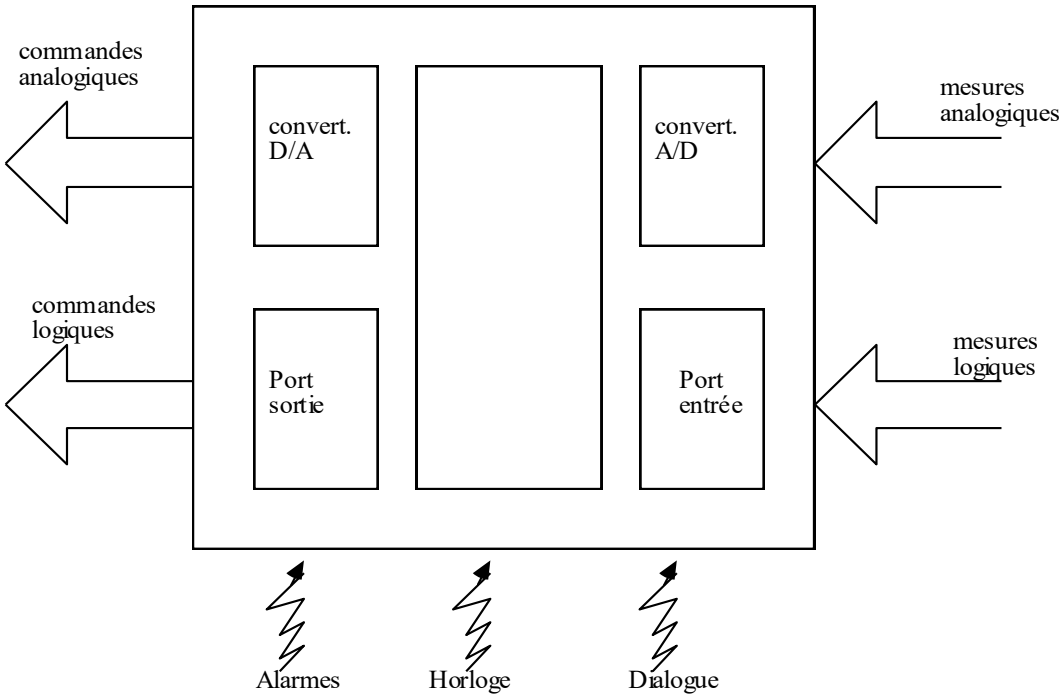
Ordonnancement dans les STR

Boris Vidolov

UMR CNRS 7253 HeuDiaSyC
Département Génie Informatique

bvidolov@utc.fr

Tâches périodiques et apériodiques



- **contrôle** (observation) - Traitement des signaux provenant du procédé par l'intermédiaire des capteurs. Les informations peuvent arriver n'importe quand, elles doivent être traitées en priorité; les contraintes de temps sont serrées.

- **commande** - Informations à envoyer au procédé, de façon périodique ou à des instants bien précis. Les commandes doivent être impérativement envoyées avant la fin du temps limite.

- **gestion du dialogue homme-machine** - Dialogue peut important donc peu prioritaire; le rôle de l'homme est un rôle de supervision. Il y a peut ou pas de contraintes de temps.

- **traitement des données** - Mise à jour des fichiers. Il s'agit de conserver l'historique de la régulation, la trace dans le temps des alarmes, et d'éditer un journal.

Ordonnancement préemptif des processus apériodiques

- garantir les délais pour les processus à contraintes strictes
- fournir un temps de réponse moyen satisfaisant pour les processus apériodiques à contraintes relatives.

Problème difficile

Solution : considère désormais que l'on a une configuration de processus périodiques et éventuellement des processus apériodiques. On choisit de garantir en priorité les processus périodiques, au détriment des éventuels processus apériodiques

Tâches apériodiques à contraintes relatives

Traitement d'arrière plan – (background processing)

- d'abord les tâches périodiques et s'il reste du temps les tâches apériodiques – gestion de FIFO
- valable pour tout algorithme d'ordonnancement
- simple et peu « couteuse »
- mauvais temps de réponse pour les Tap

Utilisation d'un serveur périodique de processus apériodiques

Idée : créer un serveur périodique pour les processus apériodiques.

- il faut que la capacité de traitement du processeur soit suffisante pour garantir tous les processus périodiques.

- les processus périodiques sont supposés être ordonnancés selon l'algorithme Rate Monotonic.

Scrutation des processus

- serveur périodique pour les processus apériodiques = tâche de priorité élevée

- s'il n'y a pas de Tap, le serveur se suspend lui-même jusqu'à sa prochaine période d'activité – perte de la capacité de traitement

- les processus apériodiques ont la priorité de leur serveur.

- cette technique garantit une périodicité dans le traitement des processus apériodiques, mais avec un rendement faible.

Serveur différé

- tâche périodique de haute priorité – Cap, Tap
- le serveur garde sa capacité de traitement
- Cap renouvelée à chaque activation du serveur
- meilleur Tr pour les tâches apériodiques
- risque de manque d'échéance pour des tâches périodiques

$$CS: \quad U_p \leq \ln\left(\frac{U_s + 2}{2U_s + 1}\right)$$

Serveur sporadique

- processus périodique de haute priorité dont le but est de servir les processus apériodiques

- la capacité de service n'est pas réinitialisé périodiquement, mais seulement après que tout ou une partie a été consommée

- date de réinitialisation = instant de démarrage du serveur + sa période

Serveur à échange de priorités

Idée : un processus périodique dont la capacité de service est préservée tout au long de sa période

- elle est réinitialisée à sa valeur maximale, au début de chaque période

- on attribue au serveur une priorité supérieure à la plus grande des priorités des autres processus périodiques

$$CS: \quad U_p \leq \ln(2/U_s + 1)$$

Ordonnancement conjoint

- pas de serveur périodique pour les tâches apériodiques

- ordonnancement des tâches apériodiques au plus tôt

- à l'activation d'une tâche apériodique les instants de déclenchement des tâches périodiques sont reculés au maximum – laxité des tâches

- petit temps de réponse pour les Tap

- variantes suivant l'algorithme d'ordonnancement RM, EDF

Tâches apériodiques à contraintes strictes

- attribution aux Tap des « pseudo périodes » - intervalle de temps entre deux apparitions – ordonnancement selon RM.

  - comment définir cette période
  - surdimensionnement du système

- à chaque apparition d'une Tap une « routine de garantie » est appelée

Acceptation dans le temps creux

- ordonnancement des tâches apériodiques dans les temps creux suivant la méthode EDF + routine de garantie :

  - temps creux suffisant avant l'échéance de la Tap

  - l'acceptation de la nouvelle Tap ne compromet pas les échéances des Tap déjà acceptées.

Acceptation des tâches apériodiques et ordonnancement conjoint

- à chaque apparition d'une Tap une nouvelle séquence d'exécution des tâches (périodiques et apériodiques) suivant la méthode EDF

- prise en compte des laxités des tâches

Ordonnancement et Relations entre tâches

•les tâches s'exécutent selon un ordre fixé - des contraintes de précédence

•les tâches partagent une (ou plusieurs) ressource(s) - contraintes de ressources

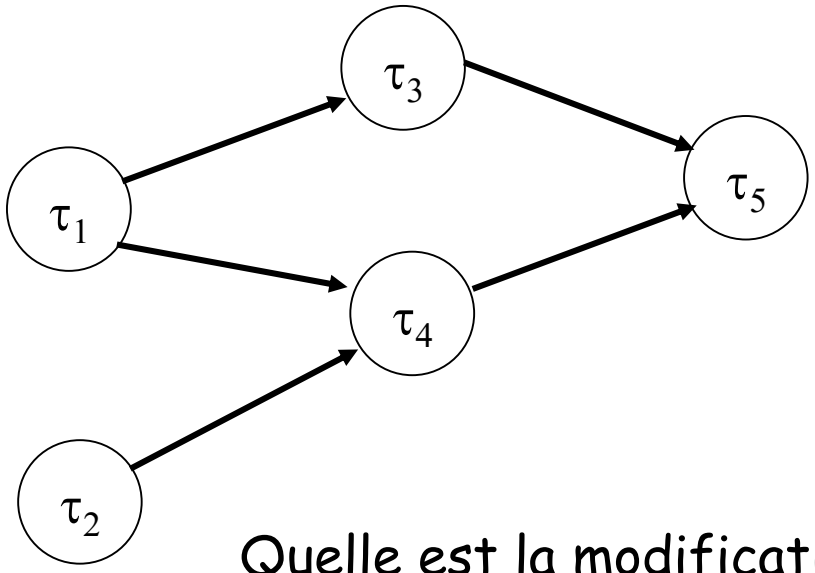
Algorithmes d'ordonnancement intégrant les relations entre tâches :

Relations de précédence
Contraintes de ressources

$\Rightarrow$ problèmes de : blocage, inversion de priorité …

Ordonnancement de tâches dépendantes

Relations de précédence



$\tau_i \rightarrow \tau_j$: τ_j doit attendre la fin de τ_i pour commencer sa propre exécution

Quelle est la modification des paramètres des tâches qui permettra une exécution dans le respect des échéances ?

- $r_j \geq r_i$
- $\text{Prio}_i \geq \text{Prio}_j$: fonction de la politique d'ordonnancement

Transformation hors ligne

en un ensemble de tâches indépendantes

Relations de précédence et RM

$$r_i^* = Max\left(r_i, r_j^*\right) : \tau_j \rightarrow \tau_i$$

Relations de précédence et DM

$$r_i^* = Max\left(r_i, r_j^*\right) : \tau_j \rightarrow \tau_i$$

$$R_i^* = Max\left(R_i, R_j^*\right)$$

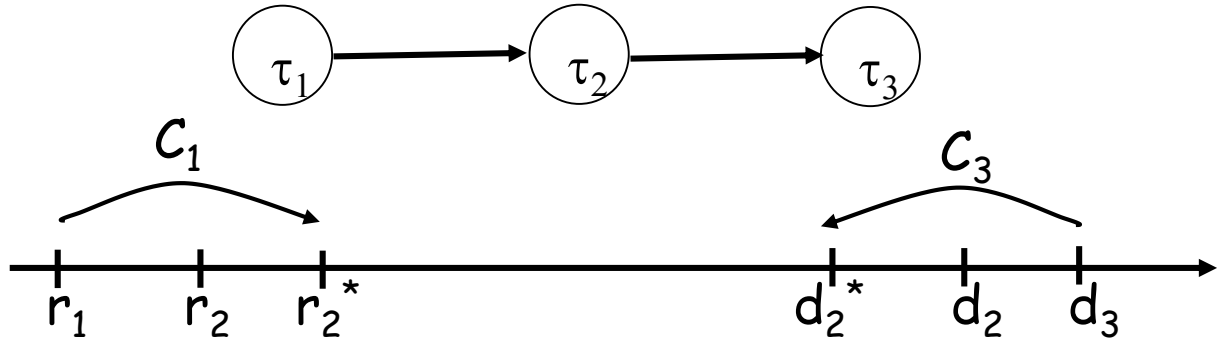
Variante de EDF basée sur la notion d'échéance modifiée

- Une tâche n'est pas activable que si tous ses prédécesseurs ont terminé => dates de disponibilité

$$r_i^* = Max\left(r_i, Max\left(r_j^* + C_j\right)\right) : \tau_j \rightarrow \tau_i$$

- L'échéance d'une tâche doit être inférieure à toutes les échéances des ses successeurs

$$d_i^* = min\left(d_i, min\left(d_j^* - C_j\right)\right)$$



Ordonnancement de tâches dépendantes
Contraintes de ressources

Resource critique à un point d'accès => exclusion mutuelle

- ordonnancement non préemptif – pas de problème
- ordonnancement préemptif – **temps de réponse**

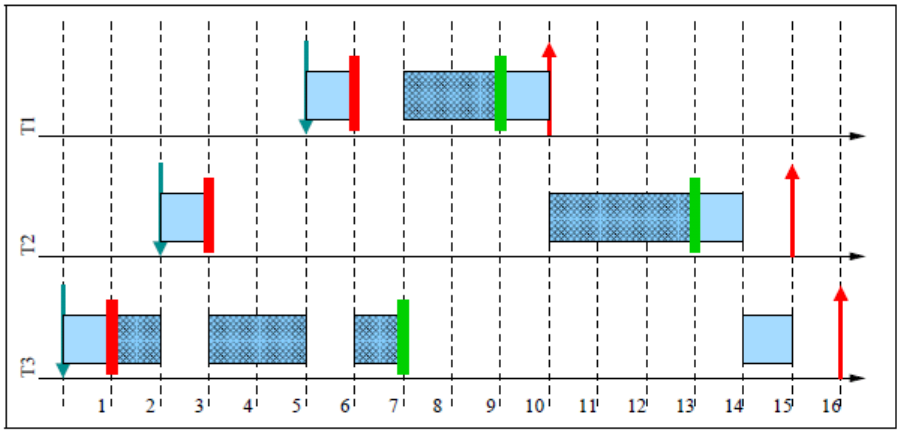
Inter blocage

- Classes ordonnées – demande de ressources suivant le même ordre pour toutes les tâches

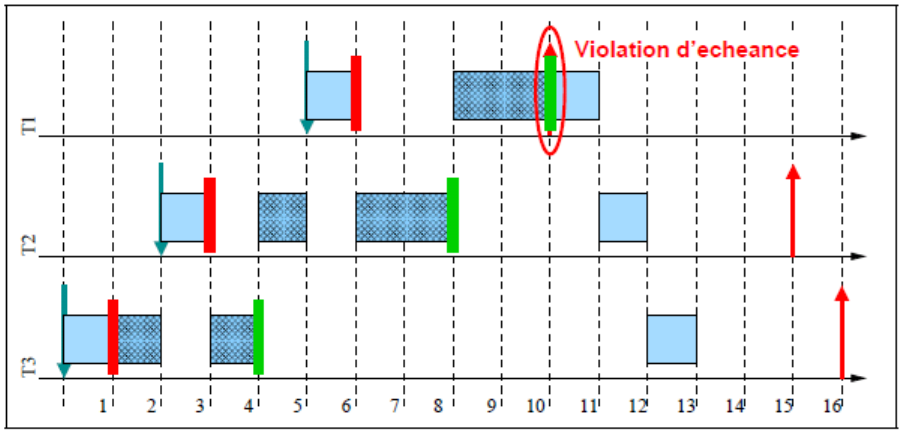
- Section critique non interruptible – **problèmes si longues**

- Méthodes formelles pour l'allocation dynamique de ressources + durée max de blocage

Ordonnancement de tâches dépendantes
Contraintes de ressources



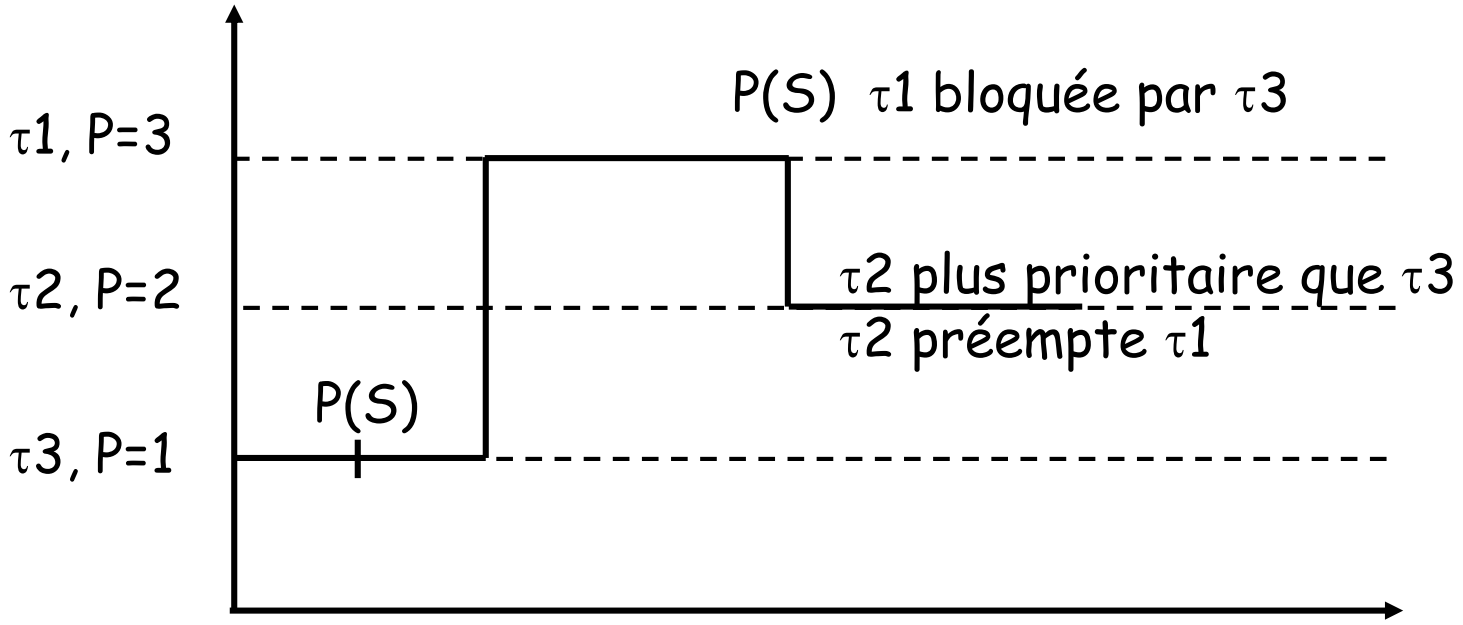
Section critique longue



Section critique courte

Ordonnancement de tâches dépendantes
Contraintes de ressources

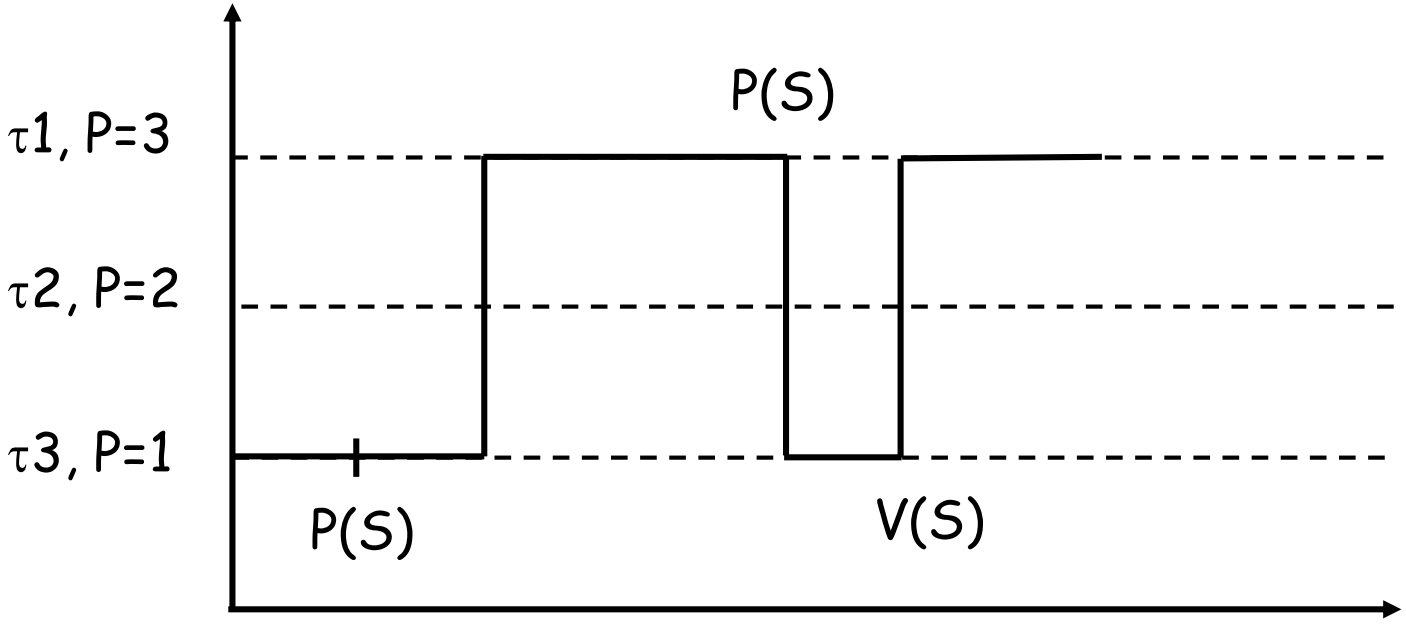
Inversion de priorité



Ordonnancement de tâches dépendantes
Contraintes de ressources

Héritage de priorité simple

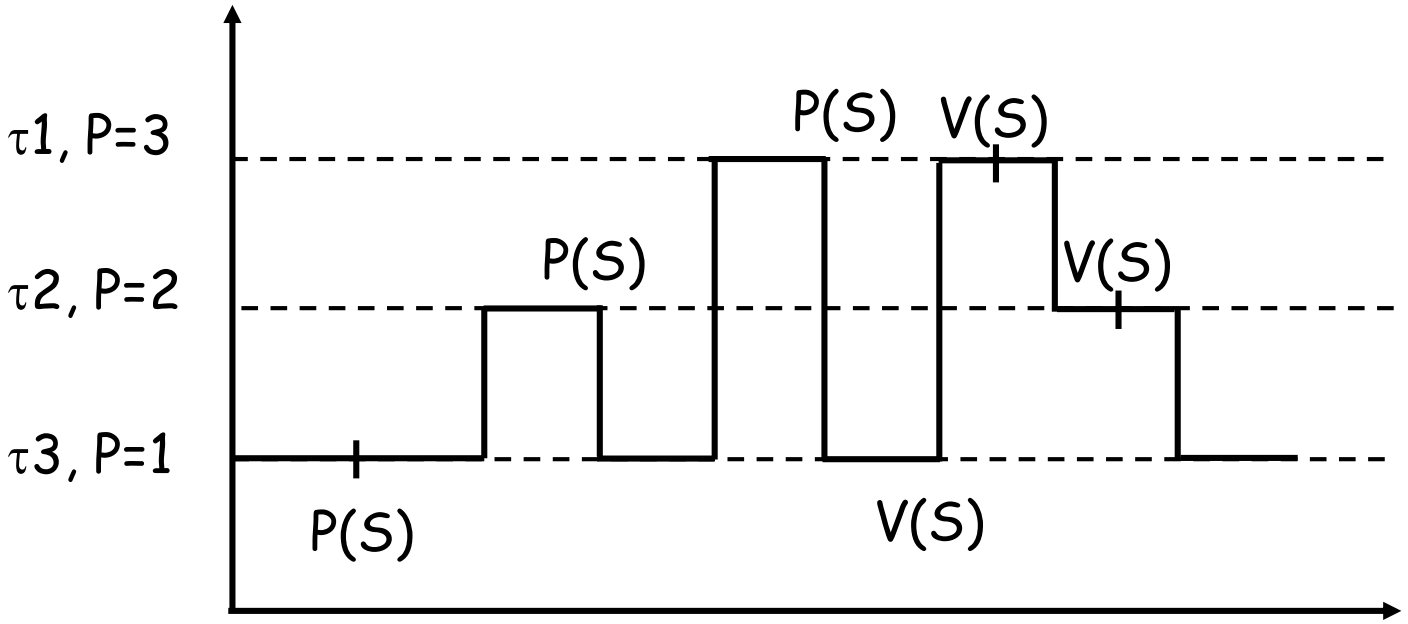
La tâche possédant la ressource qui bloque des tâches plus prioritaires hérite de la priorité de la tâche la plus prioritaire



Ordonnancement de tâches dépendantes
Contraintes de ressources

Héritage de priorité simple

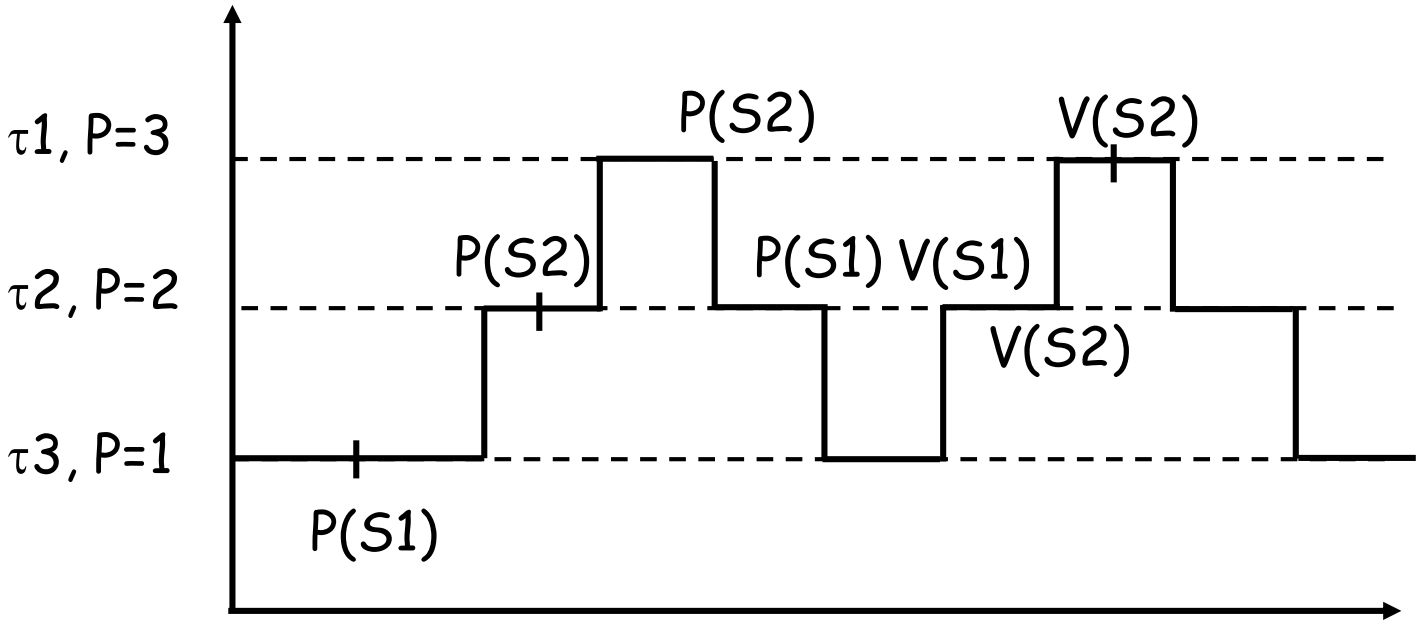
L'héritage est dynamique est peut se produire plusieurs fois



Ordonnancement de tâches dépendantes
Contraintes de ressources

Héritage de priorité simple

L'héritage est transitif



Ordonnancement de tâches dépendantes
Contraintes de ressources

Héritage de priorité simple

-Une tâche est retardée au plus par la somme des durées des sections critiques partagées avec des tâches de plus faible priorité

- Ne prévient pas l'interblocage

Ordonnancement de tâches dépendantes
Contraintes de ressources

Héritage avec plafond de priorité

Le plafond de priorité d'un sémaphore est la priorité de la tâche la plus prioritaire accédant à la ressource

- On accède à une ressource libre si la tâche en cours a une priorité supérieure au plafond des ressources en-cours

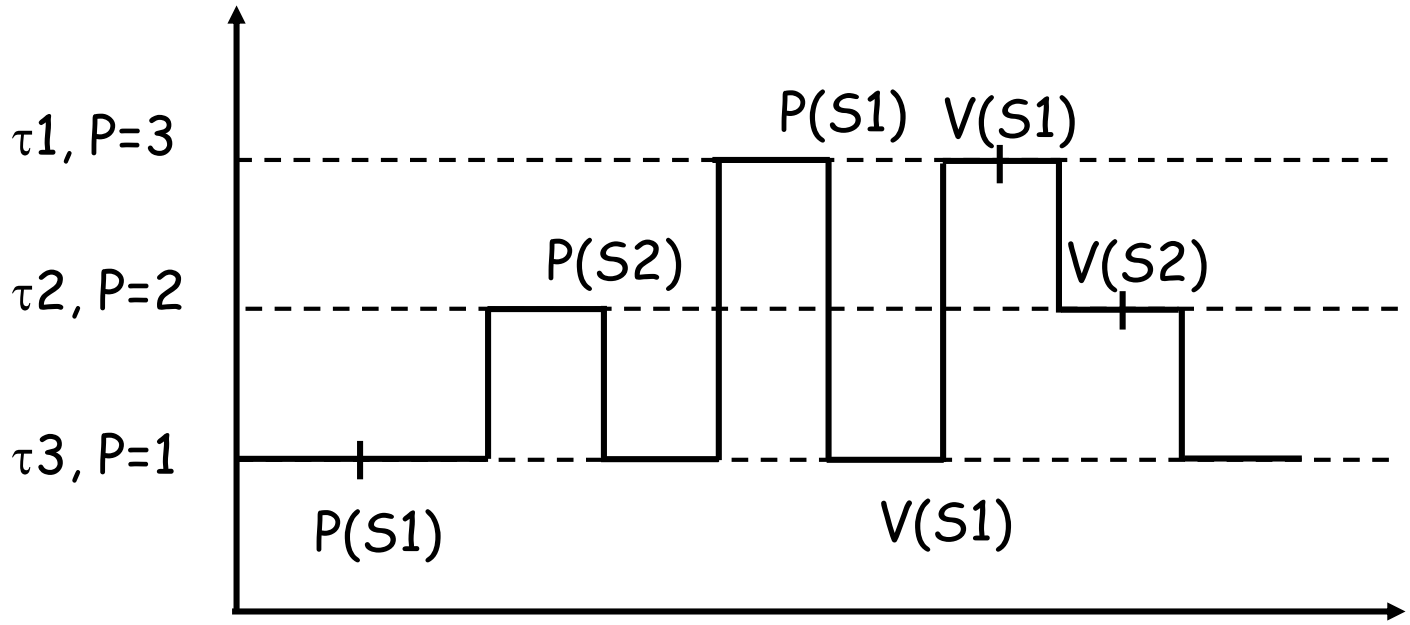
- Sinon, on se bloque et on laisse exécuter la tâche τ_i possédant une ressource de plafond supérieur ou égal à la priorité de la tâche en cours
  τ_i hérite de la priorité de la tâche en cours

- Si la ressource n'est pas libre on se bloque et la tâche possédant la ressource hérite de la priorité de la tâche en cours

Ordonnancement de tâches dépendantes
Contraintes de ressources

Héritage avec plafond de priorité

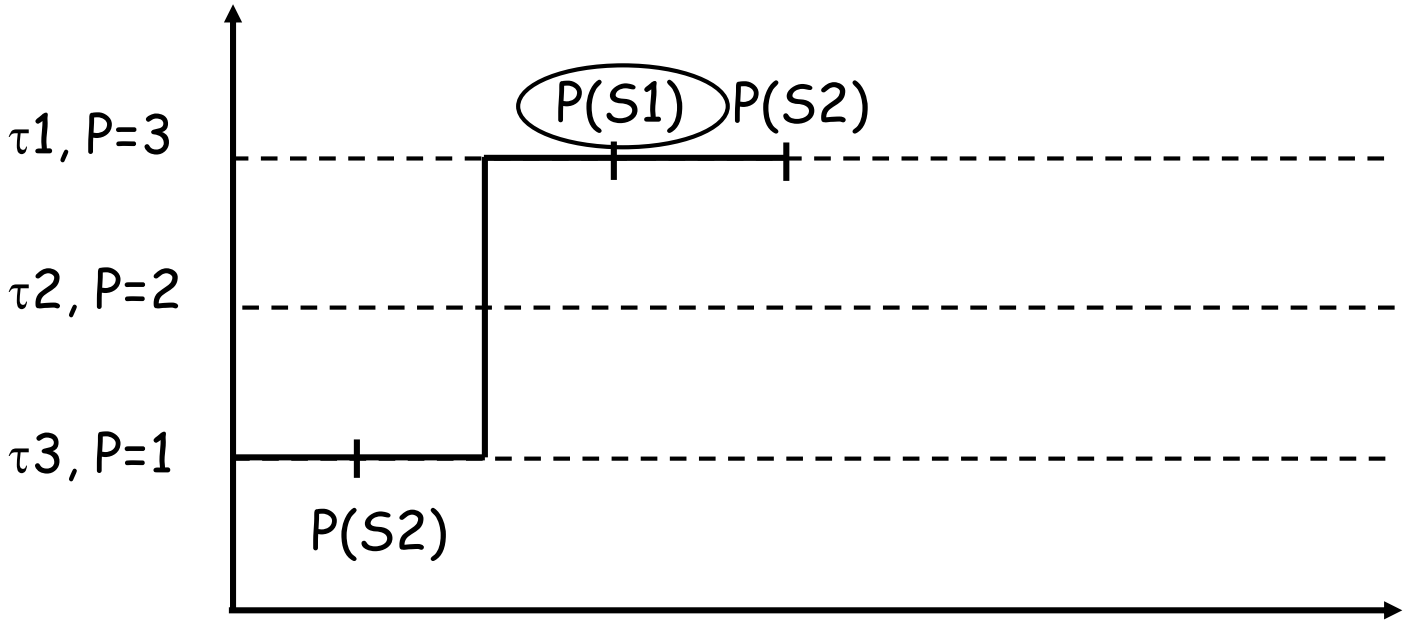
Plafonds : S1 = 3; S2 = 2



Ordonnancement de tâches dépendantes
Contraintes de ressources

Héritage avec plafond de priorité

Les interblocages ne peuvent pas arriver



Ordonnancement de tâches dépendantes
Conditions suffisantes

Ordonnancement à priorité statique avec B_i la borne sur le temps de blocage

$$\forall i \in [1 \ldots n], \sum_{i=1}^{n} \frac{Ci}{\min(T_i, D_i)} + \frac{B_i}{\min(T_i, D_i)} \leq n\left(2^{1/n} - 1\right)$$

Ordonnancement à priorité dynamique avec B_i la borne sur le temps de blocage

$$\forall i \in [1 \ldots n], \sum_{i=1}^{n} \frac{Ci}{\min(T_i, D_i)} + \frac{B_i}{\min(T_i, D_i)} \leq 1$$

Ordonnancement en situation de surcharge

Surcharge => impossibilité de respecter les échéances des tâches

- transmission défectueuse, réception tardive de signaux
- ressources partagées
- réveils de tâches apériodiques

Conséquences

- imprévisibilité de non respect des échéances des tâches
- cascade de fautes temporelles
- système sans prise en compte des situations de surcharge ???

Ordonnancement en situation de surcharge

Redondance matérielle –

⇒ Plusieurs calculateurs réalisent le même travail
⇒ répartition de la charges => algorithmes lourds

Solutions logiciels -

⇒ Algorithmes d'ordonnancement tolérants aux fautes
⇒ Modèles de tâches modifiés

Tolérance aux surcharges

Tâches périodiques

Mécanisme à échéance

Deux versions pour chaque tâche –

- *primaire* – bon résultat, mais dans un temps plus long
- *secondaire* – résultat acceptable dans un temps court

Pour toute tâche l'un des deux programmes doit être exécuté

Algorithme de la première chance

Les secondaires sont exécutés avec une priorité supérieure aux primaires. Les primaires sont exécutés dans les temps creux

Algorithme de la seconde chance

Le primaire est exécuté d'abord et l'ordonnanceur calcule une date t_i au plus tard pour lé début du secondaire. Si le primaire n'est pas terminé à t_i il est préempté au profil du secondaire.

Calcul approché

La partie mandataire doit être obligatoirement exécuter dans le respect de son échéance.

La *partie optionnelle*, affine le résultat, s'exécute s'il reste assez de temps

Avantage - pas de multiple versions des tâches, ni de coordination entre les versions

Tolérance aux surcharges

Tâches périodiques et apériodiques

⇒ Définition d'un nouveau paramètre – **importance**

⇒ A chaque éveil de tâche – un test de garantie évalue si la nouvelle tâche peut être ordonnancée sans provoquer de faute temporelle

$$LC_i(t) = d_i - \sum_j C_j(t) - t : d_j \leq d_i$$

$$C_j(t) = C_j - t$$

$$LC_i(t) < 0 : \text{surcharge}$$

⇒ Prise en compte du critère d'importance

Tolérance aux surcharges

Tâches périodiques et apériodiques

Amélioration : Pas de suppression de tâches, mais plusieurs modes de fonctionnement

Tâche T1

Mode_Normal (C = 10, Imp)

```
Début
      Acquérir (capteur);
      Lire (capteur, temp);
      Restituer (capteur);
      Rez := Calcul (temp);
      Envoyer (Rez, T2);
Fin
```

Mode_Liberation (C = 3, Imp) – **suppression lors de l'exécution**

```
Début
      Restituer (capteur);
      Envoyer (Rez, T2);
Fin
```

Mode_suppression (C = 1, Imp) - **suppression avant l'exécution**

```
Début
      Rez := Rez(k-1)*facteur_de_correction;
      Envoyer (Rez, T2);
Fin
```