1. TD Temps Réel 2ème année 2006-2007

Ordonnancement

1. Ordonnancement de tâches apériodiques dans une configuration de tâches périodiques

On considère une configuration comprenant 3 tâches périodiques :

- Tp_1 : ($r_0 = 0$, C = 5, D = 25, P = 30)
- Tp_2 : ($r_0 = 0$, C = 10, D = 40, P = 50)
- Tp_3 : ($r_0 = 0$, C = 20, D = 55, P = 75)

1. quelle est la période d'étude de cette configuration ?
2. la configuration est elle ordonnançable par l'algorithme EDF ? Dessinez le chronogramme et les temps creux.
3. Soient les tâches apériodiques :
   - Tap_1 : ($r_0 = 40$, C = 10, D = 15)
   - Tap_2 : ($r_0 = 70$, C = 15, D = 35)
   - Tap_3 : ($r_0 = 100$, C = 20, D = 40)
   - Tap_4 : ($r_0 = 105$, C = 5, D = 25)
   - Tap_5 : ($r_0 = 120$, C = 5, D = 15)

Chacune de ces tâches peut-elle être acceptée au sein de la configuration en utilisant la méthode des temps creux ?

Corrigé

1. La période d'étude est le PPCM des différentes périodes des tâches périodiques, soit T = 150
2. Une condition suffisante d'ordonnancement EDF des tâches périodiques est $\sum_{i=1}^{n} \frac{C_i}{D_i} \leq 1$

5/25 + 10/40 + 20/55 = 0,2 + 0,25 + 0,36 = 0,81

La condition suffisante est donc bien vérifiée est les tâches sont ordonnançables suivant l'algorithme EDF

Les temps creux sont [40, 50], [65, 75] [110, 120] et [125, 150]

3. Tâche Tap_1 ($r_0 = 40$, C = 10, D = 15) peut être acceptée et exécutée dans le temps creux [40, 50]

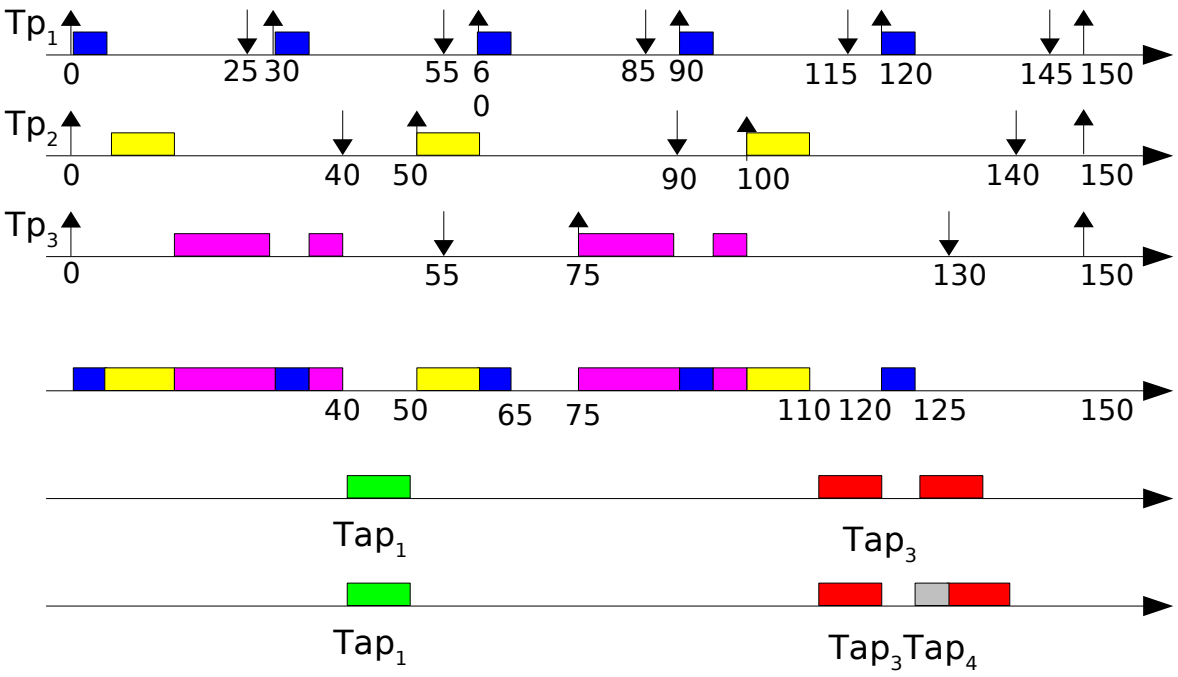
   Tâche Tap_2 ($r_0 = 70$, C = 15, D = 35) n'a pas de temps creux suffisant pour son exécution entre 70 et 105. Elle est refusée

   Tâche Tap_3 ($r_0 = 100$, C = 20, D = 40) peut être exécutée dans les temps creux [110, 120] et [125, 150], et ainsi se terminer à 135,

avant son échéance à 140

Tâche Tap_4 (r_0 = 105, C = 5, D = 25) est acceptée car son exécution dans le temps creux [125, 150] ne remet pas en cause l'exécution de Tap_3 qui peut être reculée de 5 et se terminer juste à 140

Tâche Tap_5 (r_0 = 120, C = 5, D = 15) pourrait etre exécutée dans le temps creux [125, 150], mais son acceptation n'est pas compatible avec le respect de l'échéance de Tap_3

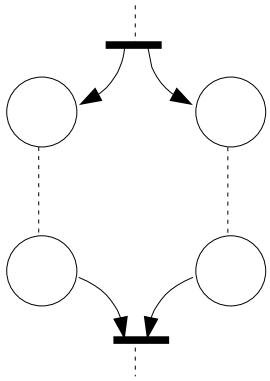


Réseaux de Petri

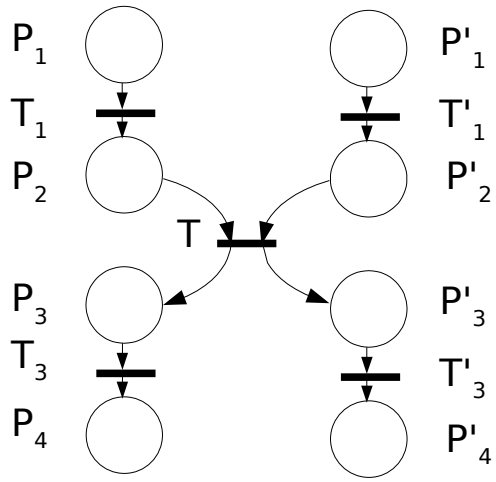
2. Visualisation de comportements simples

1. Utiliser des réseaux de Petri pour visualiser l'exécution en parallèle de deux tâches différentes (par exemple, après un *fork*)

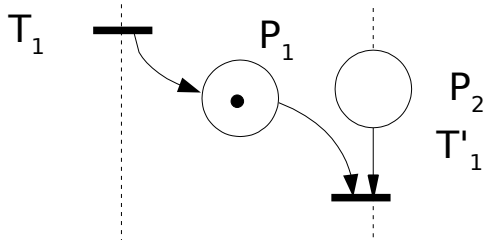
1. TD Temps Réel 2ème année 2006-2007



2. Rendez-vous entre 2 tâches :
   Modifier le RdP ci-contre pour que les deux tâches se synchronisent sur la place P_2 pour la première et P'_2 pour la deuxième.



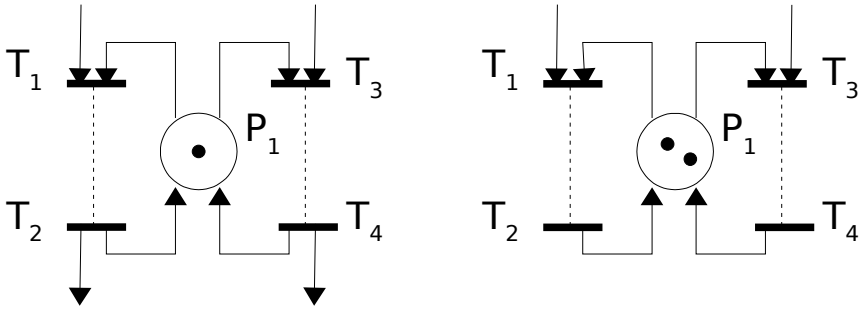
Synchronisation par un sémaphore :



3. Partage de ressource :

   Le franchissement d'une transition T_1 pour une première tâche et une transition T'_1 pour une seconde tâche nécessite le partage d'une ressource commune. Modéliser cette situation avec un RdP,

quand une seule tâche peut avoir accès à la fois à la ressource commune. Modifier le RdP pour que 2 tâches puissent accédere simultanément à la ressource.

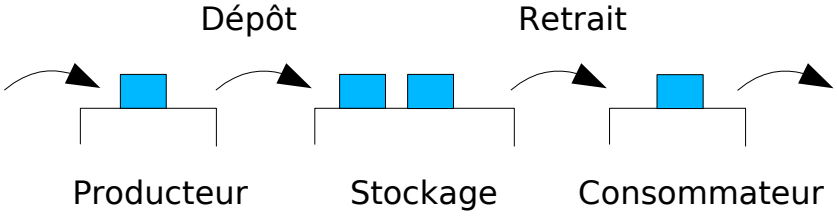


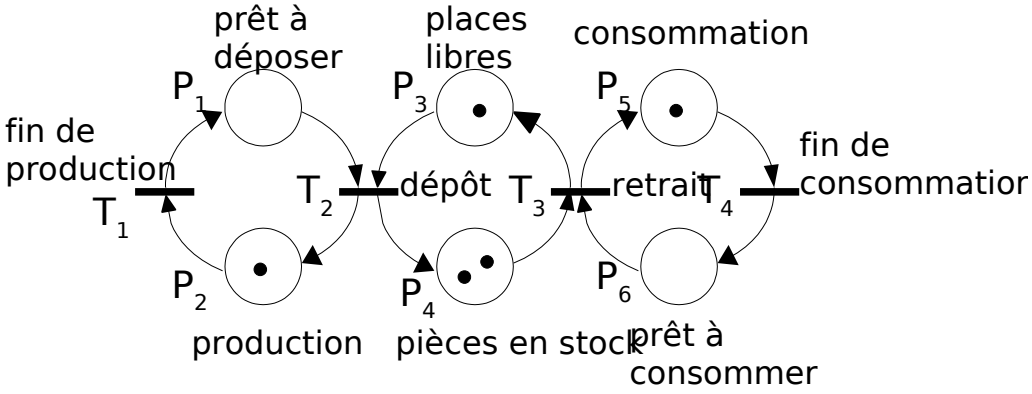
3. Relation producteur-consommateur

Un producteur produit une seule pièce à la fois. Quand la pièce est prête, il la dépose

dans un stockage qui a une capacité de 3 places. Il reprend immédiatement la production d'une nouvelle pièce. Le consommateur n'est capable de consommer qu'une seule pièce à la fois, qu'il prélève du stockage quand celui-ci n'est pas vide

1. représenter ce système par un réseau de Petri avec un marquage initial correspondant à la figure





1. TD Temps Réel 2ème année 2006-2007

2. quels sont les invariants de marquage de ce système ? Quelle est leur signification.
   (on appelle invariant de marquage des sommes de marquages d'un sous-ensemble de places qui ne varie pas durant la vie du RdP)

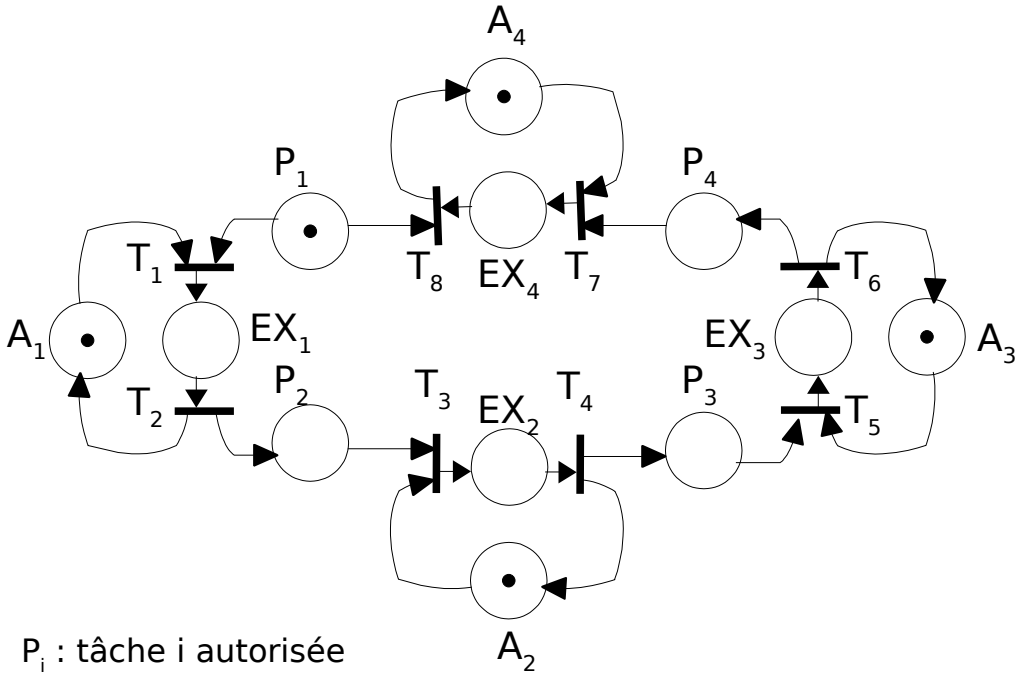
Invariants de marquage :

- $m_1 + m_2 = 1$ signifiant que le producteur n'a que 2 états possibles : "en production" et "prêt à déposer"
- $m_5 + m_6 = 1$ 1 signifiant que le consommateur n'a que 2 états possibles : "en consommation" et "prêt à consommer"
- $m3 + m_4 = 3$ correspondant à la capacité du stockage

4. Modélisation d'un ordonnancement Round Robin

Plusieurs tâches se partagent la CPU du système. Elles peuvent s'exécuter à tour de rôle (elles ont la même priorité), chacune exécutant alors une instruction, puis doivent relâcher la CPU pour la laisser à la tâche suivante.

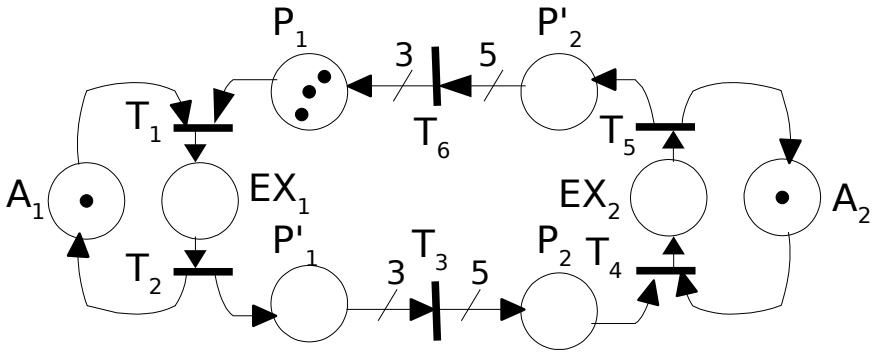
1. Modéliser par un RdP l'ordonnancement de 4 tâches (que l'on suppose de durée infinie)



P_i : tâche i autorisée
A_i : tâche i en attente
EX_i : tâche i en cours d'exécution

Pour le marquage initial, seule la transition T_1 est validée. Quand on la franchit, on dépose une marque dans la place EX_1 et la tâche s'exécute. Quand T_2 est franchie (à la fin de l'exécution de la tâche 1), une marque est déposée dans P_2 et T_3 est validée et son franchissement permet l'exécution de la tâche 2, et ainsi de suite...

2. Modéliser par un RdP généralisé l'ordonnancement de 2 tâches en supposant que l'on exécute 3 instructions de la tâche 1 quand on en exécute 5 pour la tâche 2



Avec le marquage initial, on doit exécuter trois fois la tâche 1 (franchissement de T_1 et T_2 avant d'avoir déposé dans P'_1 les trois marques qui permettront de valider T_3 et de déposer 5 marques dans P_2 .