Temps Réel

Jérôme Pouiller <j.pouiller@sysmic.org>

Quatrième partie IV

Ordonnancement

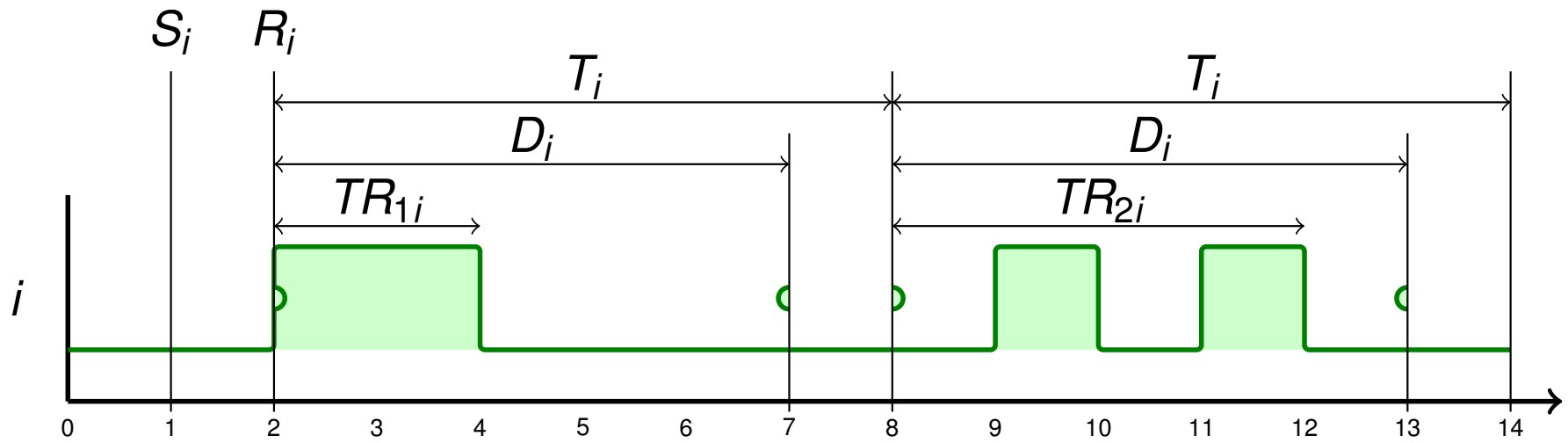
sysmic

Modèle de tâches

Paramètres définissant une tâche i :

- Date d'arrivée de la tâche dans le système : S_i
- Première date d'activation : R_i
- Période d'activation : T_i
- Délai critique ou *deadline* (Délai maximum acceptable pour son exécution) : D_i
- Capacité (Temps CPU nécessaire à l'exécution de la tâche) : C_i

Modèle de tâches



Tâche	Arrivée	Période	Capacité	Délai
i	2	6	2	5

Modèle de tâches

Par conséquent :

- Tâche périodique définie par : $(S_i, R_i, T_i, D_i, C_i)$
- Tâche apériodique définie par : $(S_i, R_i, 0, D_i, C_i)$

Très souvent :

- Les tâches sont connues au début de l'ordonnancement : $S_i = 0$
- Les tâches périodiques sont actives dès le début de l'ordonnancement : $R_i = 0$
- Le délai critique des tâches périodiques est leur période : $D_i = T_i$

Paramètres statiques

Paramètres statiques :

- Date de réveil sur une période k : r_{ik}
- Échéance sur une période k : $d_{ik} = r_{ik} + D_i$
- Facteur d'utilisation du processeur : $U_i = C_i / T_i$
- Facteur de charge du processeur : $CH_i = C_i / D_i$ ($CH_i = U_i$ si délai critique sur périodes)
- Laxité (ou *Slack*) de la tâche (retard maximum acceptable pour l'exécution de la tâche) : $L_i = D_i - C_i$

Paramètres dynamiques

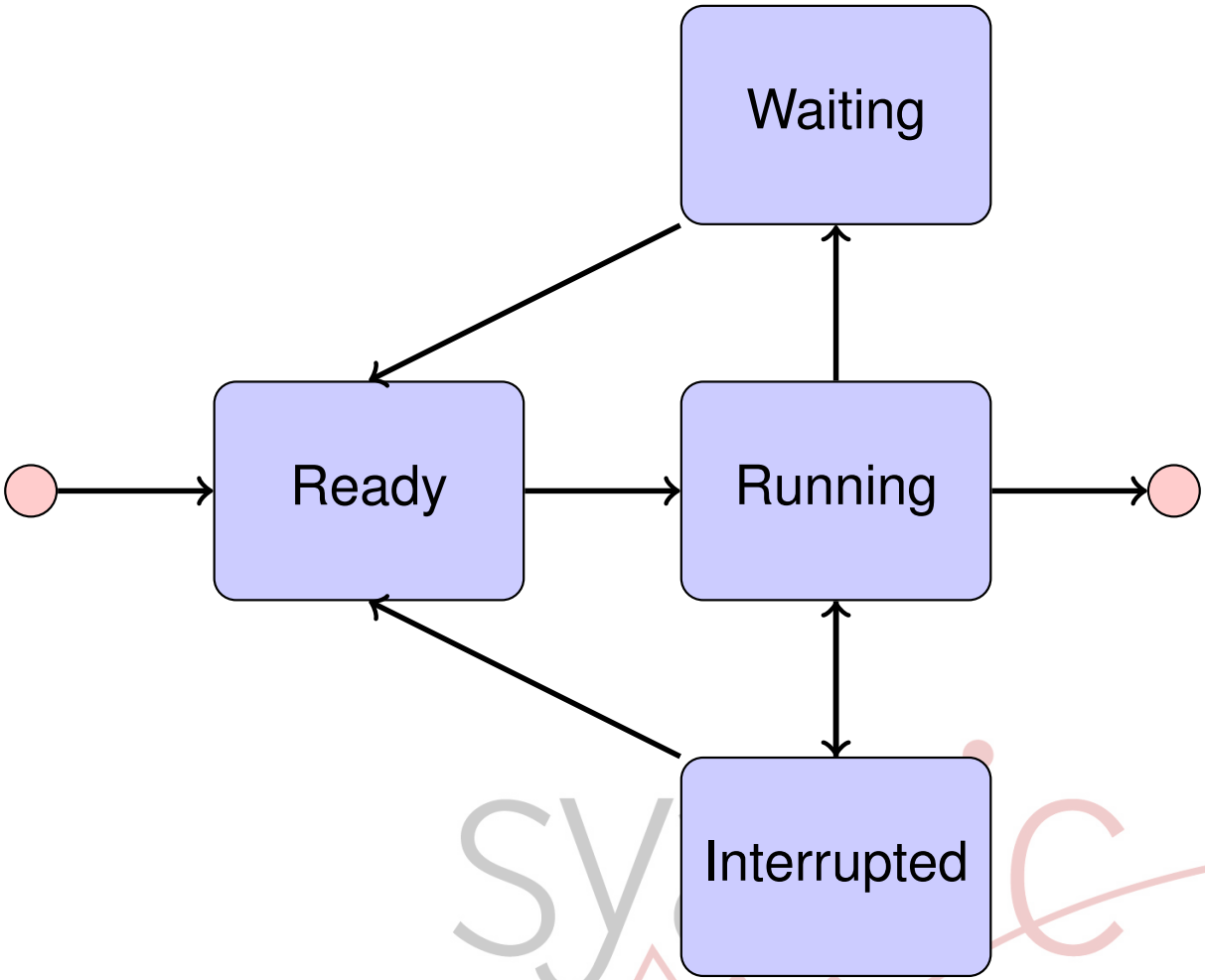
Paramètres dynamiques (dépendant de l'ordonnancement) :

- Priorité (peut être dynamique ou statique) : P_i
- Date du début de l'exécution d'une période k : s_{ik}
- Date de la fin de l'exécution d'une période k :e_{ik}
- Temps de réponse de la tâche $TR_{ik} = e_{ik} - r_{ik}$ (Nous verrons comment le calculer)
- Durée d'exécution résiduelle à la date t : $C_i(t)$ $(0 \leq C_i(t) \leq C_i)$
- Délai critique résiduel à la date t : $D_i(t) = d_{ik} - t$ $(0 \leq D_i(t) \leq D_i)$
- Charge résiduelle à la date t : $CH_i(t) = C_i(t)/D_i(t)$ $(0 \leq CH_i(t) \leq CH_i)$
- Laxité résiduelle à la date t : $L_i(t) = D_i(t) - C_i(t)$ $(0 \leq L_i(t) \leq L_i)$
- Laxité conditionnelle à la date t (somme sur les tâches déclenchées à la date t et qui sont devant i du point de vue de l'ordonnancement) : $LC_i(t)$ (calcul complexe)

Paramètres du système

- Configuration : ensemble des tâches mises en jeu par l'application
- Taux d'utilisation du processeur : $U = \sum_i U_i$
- Taux de charge du processeur : $CH = \sum_i CH_i$
- Intervalle d'étude : intervalle de temps minimum pour prouver l'ordonnancabilité d'une configuration
  - Dans le cas de tâches périodiques : $ppcm_i(T_i)$
  - Dans le cas de tâches apériodiques : $[\min_i\{R_i\}, \max_i\{R_i + D_i\} + 2 \times ppcm_i(T_i)]$
- Laxité du processeur : intervalle de temps pendant lequel le processeur peut rester inactif tout en respectant les échéances : $LP(t) = \min_i\{LC_i(t)\}$

États des tâches



Typologie des algorithmes

On distingue diverses typologies d'algorithmes :

- *on line* ou *off line* : Choix dynamique ou prédéfini à la conception
- à priorité *statique* ou *dynamique* : La priorité d'une tâche est-elle fixe ou une variable dépendante d'autres paramètres
- *préemptif* ou *non préemptif* : Une tâche peut-elle perdre le processeur (au profit d'une tâche plus prioritaire) ou non

Théorème de l'instant critique

Si toutes les tâches arrivent initialement dans le système en même temps et si elles respectent leur première échéance, alors toutes les échéances seront respectées par la suite, quel que soit l'instant d'arrivée des tâches.

- C'est une condition nécessaire et suffisante si toutes les tâches du système sont initialement prêtes au même instant.
- Dans le cas contraire, c'est une condition nécessaire

Temps de réponse

Délai entre l'activation d'une tâche et sa terminaison.

$$TR_i = C_i + \sum_{P_j > P_i} \left\lceil \frac{TR_i}{T_j} \right\rceil C_j$$

Calcul du temps de réponse

Technique de calcul : on évalue de façon itérative

$$w_i^{n+1} = C_i + \sum_{P_j > P_i} \left\lceil \frac{w_i^n}{T_j} \right\rceil C_j$$

- On démarre avec $w_i^0 = C_i$
- Échec si $w_i^n > T_i$
- Réussite si $w_i^{n+1} = w_i^n$

Calcul du temps de réponse

Exemple :

Tâche	Arrivée	Période	Capacité	Délai	Priorité
A	0	7	3	Fin de période	1
B	0	12	2	Fin de période	2
C	0	20	5	Fin de période	3

Calcul du temps de réponse

Exemple :

Tâche	Arrivée	Période	Capacité	Délai	Priorité
A	0	7	3	Fin de période	1
B	0	12	2	Fin de période	2
C	0	20	5	Fin de période	3

$$w_1^0 = C_1 = 3$$

$$w_2^0 = C_2 = 2$$

$$w_2^1 = 2 + 3\left\lceil \frac{2}{7} \right\rceil = 5$$

$$w_2^2 = 2 + 3\left\lceil \frac{5}{7} \right\rceil = 5$$

$$w_3^0 = C_3 = 5$$

$$w_3^1 = 5 + 3\left\lceil \frac{5}{7} \right\rceil + 2\left\lceil \frac{5}{12} \right\rceil = 10$$

$$w_3^2 = 5 + 3\left\lceil \frac{10}{7} \right\rceil + 2\left\lceil \frac{10}{12} \right\rceil = 13$$

$$w_3^3 = 5 + 3\left\lceil \frac{13}{7} \right\rceil + 2\left\lceil \frac{13}{12} \right\rceil = 15$$

$$w_3^4 = 5 + 3\left\lceil \frac{15}{7} \right\rceil + 2\left\lceil \frac{15}{12} \right\rceil = 18$$

$$w_3^5 = 5 + 3\left\lceil \frac{18}{7} \right\rceil + 2\left\lceil \frac{18}{12} \right\rceil = 18$$

Rate Monotonique

Aussi appelé *RMA* (*Rate Monotonic Algorithm*)

Ordonnancement à priorité statique où les priorités sont inversement proportionnelles aux périodes des tâches.

Fonctionne en version préemptive. La version non-préemptive n'est pas garantie.

Liu et Layland ont démontré qu'un système est ordonnancable si le taux d'occupation du processeur U vérifie la condition suffisante (non nécessaire) suivante :

$$U = \sum_{i}^{n} \frac{C_i}{T_i} \leq n\left(2^{\frac{1}{n}} - 1\right)$$

n	limite d'occupation
1	100.0%
2	82.8%
3	78.0%
4	75.7%
5	74.3%
10	71.9%
∞	69.3%

Exemple 1

Tâche	Arrivée	Période	Capacité	Délai
A	0	20	8	Fin de période
B	0	10	1	Fin de période
C	0	4	1	Fin de période

Exemple 1

Tâche	Arrivée	Période	Capacité	Délai
A	0	20	8	Fin de période
B	0	10	1	Fin de période
C	0	4	1	Fin de période

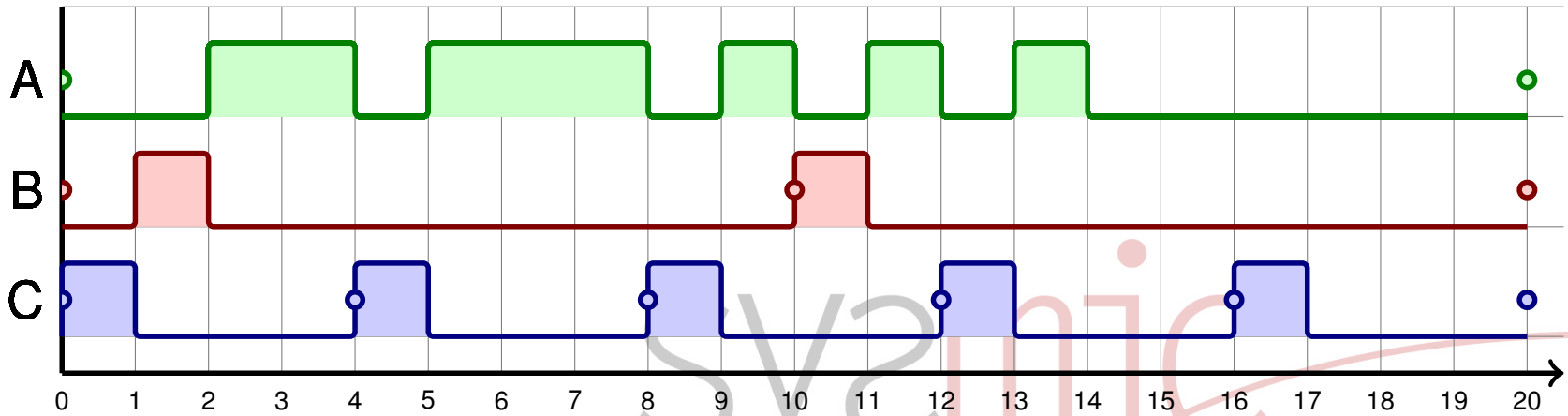
Charge du CPU :

$$\frac{8}{20} + \frac{1}{10} + \frac{1}{4} = 0.75$$

Exemple 1

Tâche	Arrivée	Période	Capacité	Délai
A	0	20	8	Fin de période
B	0	10	1	Fin de période
C	0	4	1	Fin de période

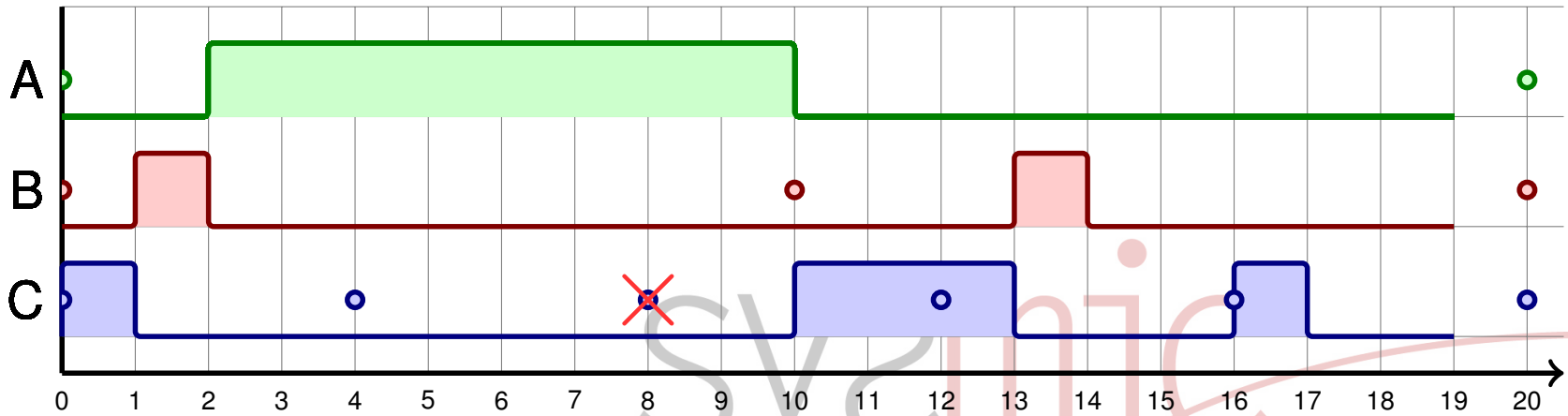
Mode préemptif ($ppcm(4, 10, 20) = 20$) :



Exemple 1

Tâche	Arrivée	Période	Capacité	Délai
A	0	20	8	Fin de période
B	0	10	1	Fin de période
C	0	4	1	Fin de période

Mode non-préemptif :



Exemple 2

Tâche	Arrivée	Période	Capacité	Délai
A	0	16	8	Fin de période
B	0	8	2	Fin de période
C	0	4	1	Fin de période

Exemple 2

Tâche	Arrivée	Période	Capacité	Délai
A	0	16	8	Fin de période
B	0	8	2	Fin de période
C	0	4	1	Fin de période

Charge du CPU :

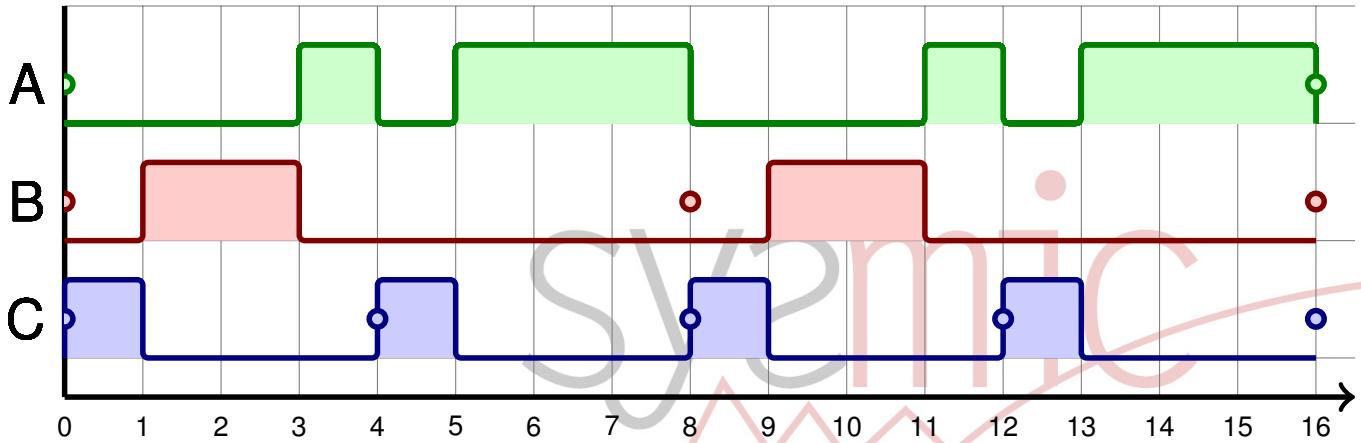
$$\frac{8}{16} + \frac{2}{8} + \frac{1}{4} = 1$$

Exemple 2

Tâche	Arrivée	Période	Capacité	Délai
A	0	16	8	Fin de période
B	0	8	2	Fin de période
C	0	4	1	Fin de période

Charge du CPU :

$$\frac{8}{16} + \frac{2}{8} + \frac{1}{4} = 1$$



Deadline Monotonic (DM)

- Aussi appelé *DMA* (*Deadline Monotonic Algorithm*
- Algorithme à priorité statique
- Généralisation de Rate Monotonic pour les tâche avec $D_i < T_i$
- Basé sur le délai critique :
  - La tâche de plus petit délai critique est la plus prioritaire
- Test d'acceptabilité (suffisant mais pas nécessaire) :

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

- Équivalent à Rate Monotonic dans le cas des tâches à échéance sur requête, meilleur dans les autres cas

Exemple

Tâche	Arrivée	Période	Capacité	Délai
A	0	20	3	7
B	0	5	2	4
C	0	10	2	9

Exemple

Tâche	Arrivée	Période	Capacité	Délai
A	0	20	3	7
B	0	5	2	4
C	0	10	2	9

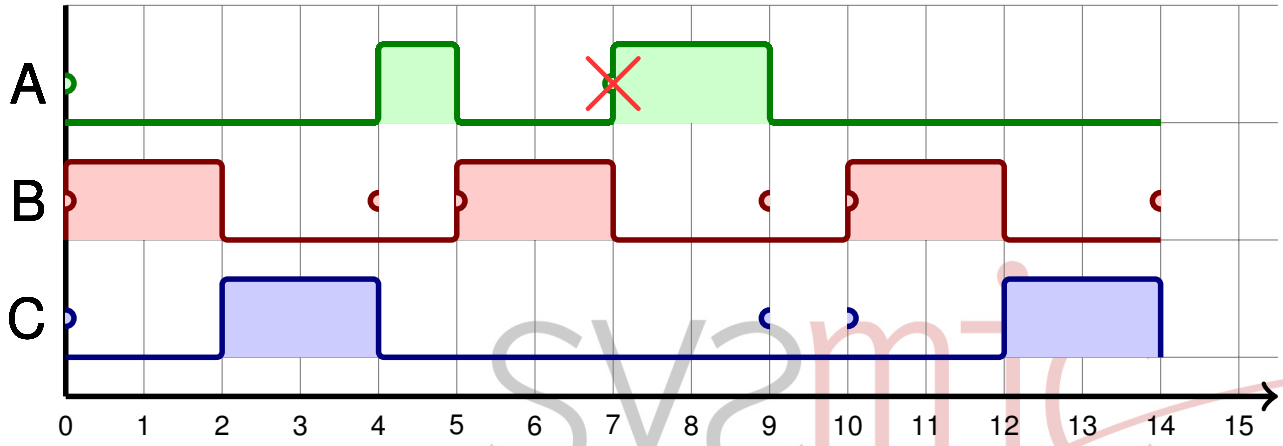
Charge CPU :

$$\frac{3}{20} + \frac{2}{5} + \frac{2}{10} = 0.75$$

Exemple

Tâche	Arrivée	Période	Capacité	Délai
A	0	20	3	7
B	0	5	2	4
C	0	10	2	9

Par RM :



Exemple

Tâche	Arrivée	Période	Capacité	Délai
A	0	20	3	7
B	0	5	2	4
C	0	10	2	9

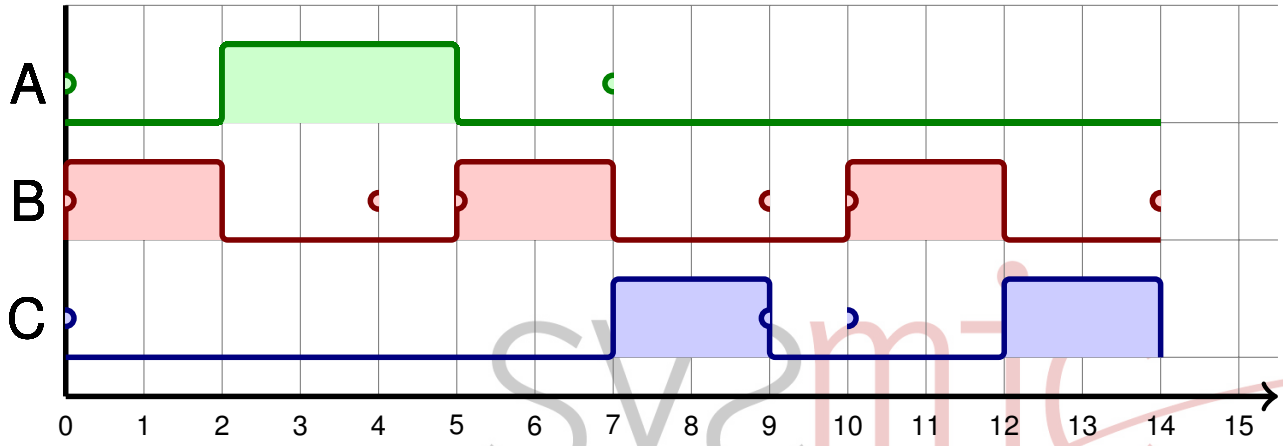
Test d'acceptabilité :

$$\frac{3}{7} + \frac{2}{4} + \frac{2}{9} = 1.15$$

Exemple

Tâche	Arrivée	Période	Capacité	Délai
A	0	20	3	7
B	0	5	2	4
C	0	10	2	9

Par DM :



Algorithme EDF

- Algorithme à priorité dynamique
- Basé sur l'échéance :
  - A chaque instant (i.e à chaque réveil de tâche), la priorité maximale est donnée à la tâche dont l'échéance est la plus proche
- Test d'acceptabilité :
  - Condition nécessaire :

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

  - Condition suffisante :

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

  - Dans le cas de tâche à échéance sur requête, les deux conditions sont égales

Exemple

Tâche	Arrivée	Période	Capacité	Délai
A	0	20	3	7
B	0	5	2	4
C	0	10	2	8

Exemple

Tâche	Arrivée	Période	Capacité	Délai
A	0	20	3	7
B	0	5	2	4
C	0	10	2	8

Condition nécessaire :

$$\frac{3}{20} + \frac{2}{5} + \frac{2}{10} = 0.75 \leq 1$$

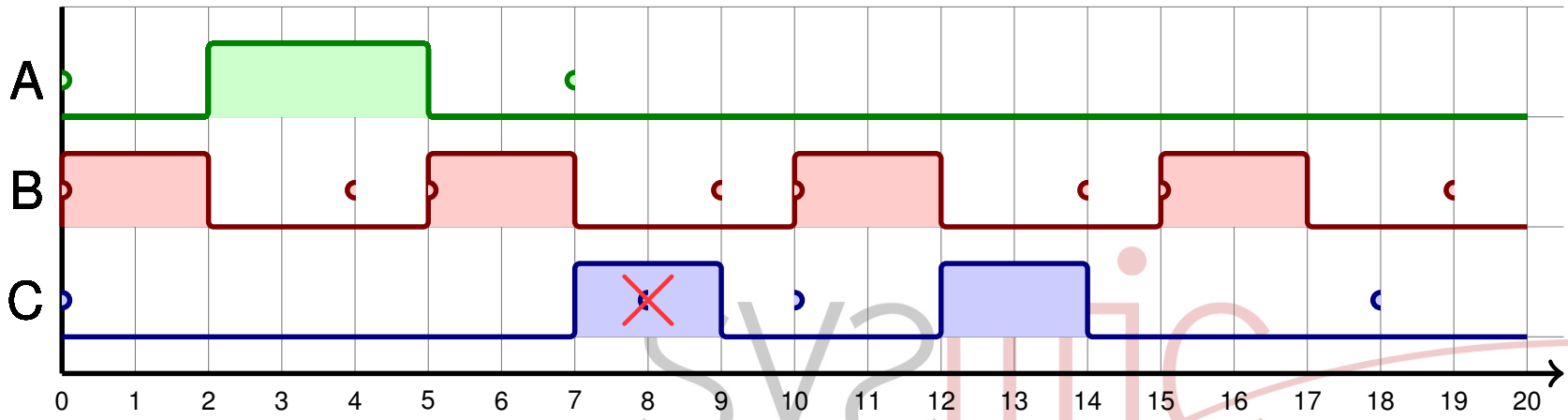
Condition suffisante :

$$\frac{3}{7} + \frac{2}{4} + \frac{2}{9} = 1.15 > 1$$

Exemple

Tâche	Arrivée	Période	Capacité	Délai
A	0	20	3	7
B	0	5	2	4
C	0	10	2	8

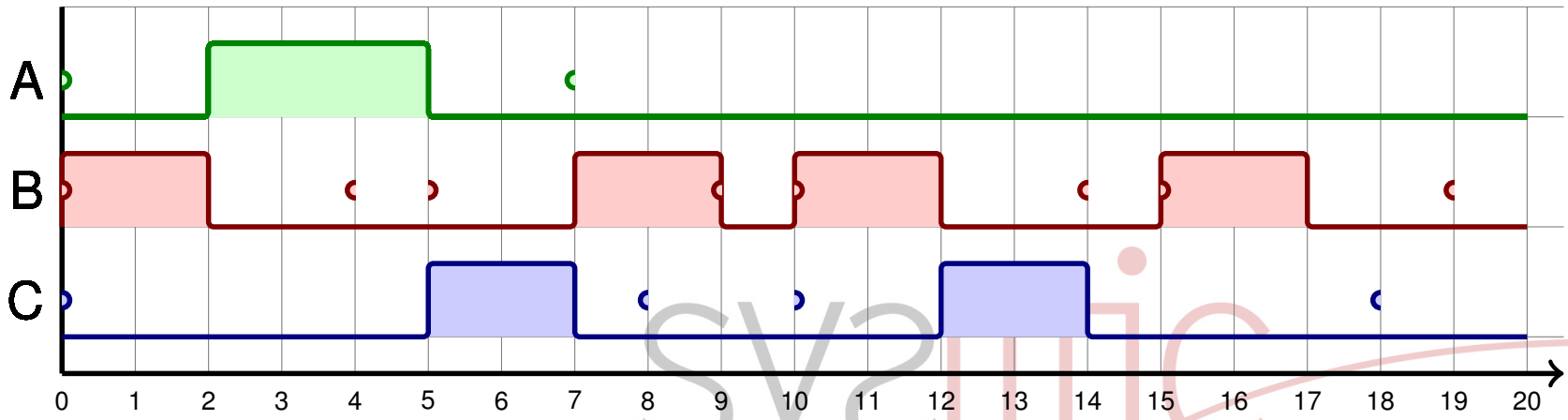
Par DM :



Exemple

Tâche	Arrivée	Période	Capacité	Délai
A	0	20	3	7
B	0	5	2	4
C	0	10	2	8

Par EDF :



Least Slack Time (LST)

- Algorithme à priorité dynamique
- Aussi appelé *Least Laxity First (LLF)*
- Basé sur la laxité résiduelle :
  - La priorité maximale est donnée à la tâche qui a la plus petite laxité résiduelle : $L(t) = D(t) - C(t)$
- Équivalent à EDF si on ne calcule la laxité qu'au réveil des tâches
- Optimum à trouver entre la granularité du calcul et le nombre de changements de contexte provoqués
- Permet d'être parfois plus résistant aux erreurs
- Demande une connaissance de la capacité des tâches
- Gain discutable
- Peu utilisé dans la pratique

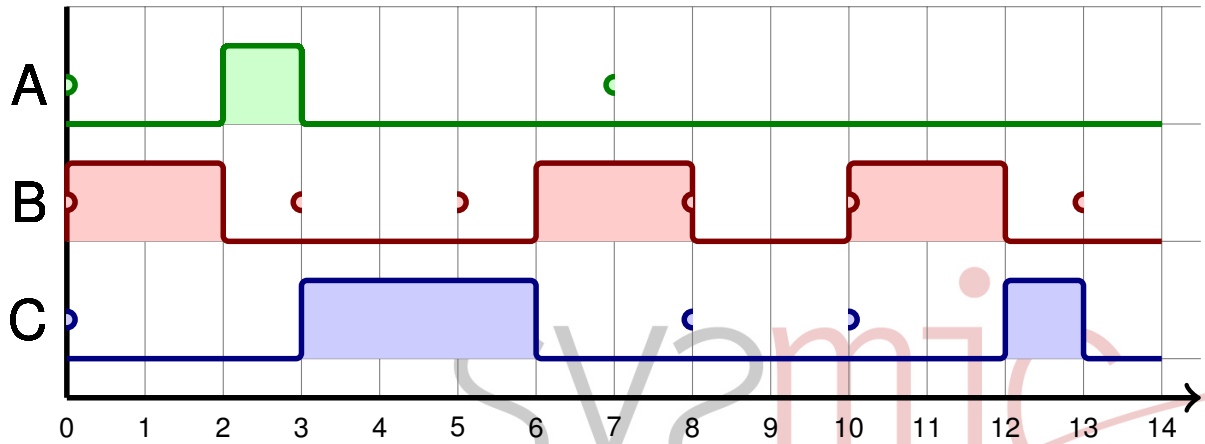
Exemple

Tâche	Arrivée	Période	Capacité	Délai
A	0	20	1	7
B	0	5	2	3
C	0	10	3	8

Exemple

Tâche	Arrivée	Période	Capacité	Délai
A	0	20	1	7
B	0	5	2	3
C	0	10	3	8

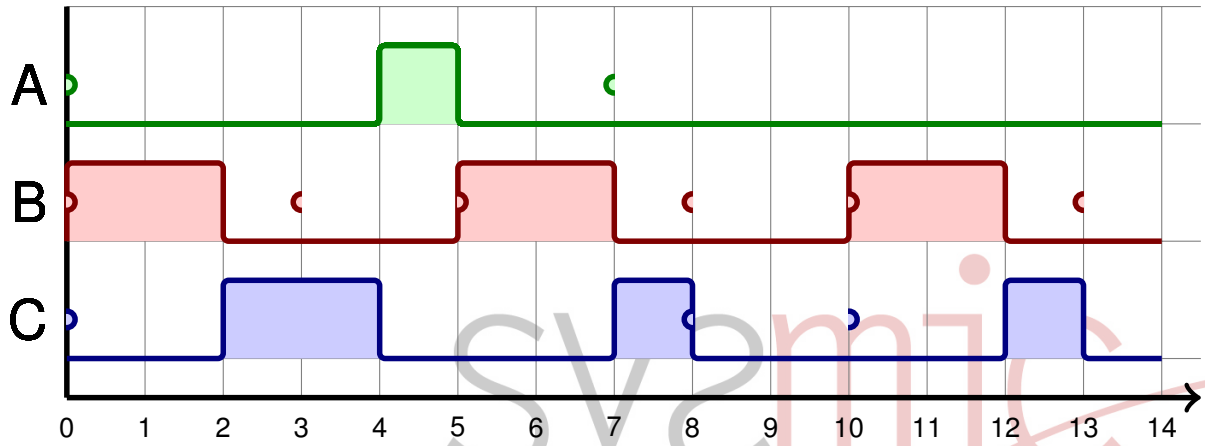
Par EDF :



Exemple

Tâche	Arrivée	Période	Capacité	Délai
A	0	20	1	7
B	0	5	2	3
C	0	10	3	8

Par LST, en recalculant l'algorithme à chaque période :



Exemple - 2

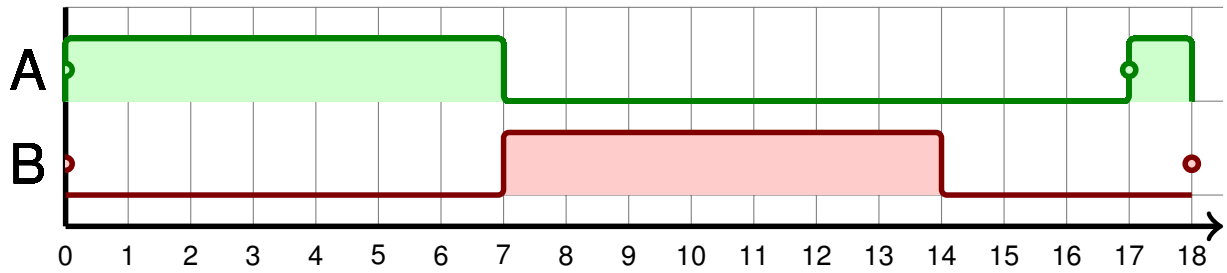
Tâche	Arrivée	Période	Capacité	Délai
A	0	17	7	Fin de période
B	0	18	7	Fin de période

Par EDF :

Exemple - 2

Tâche	Arrivée	Période	Capacité	Délai
A	0	17	7	Fin de période
B	0	18	7	Fin de période

Par EDF :

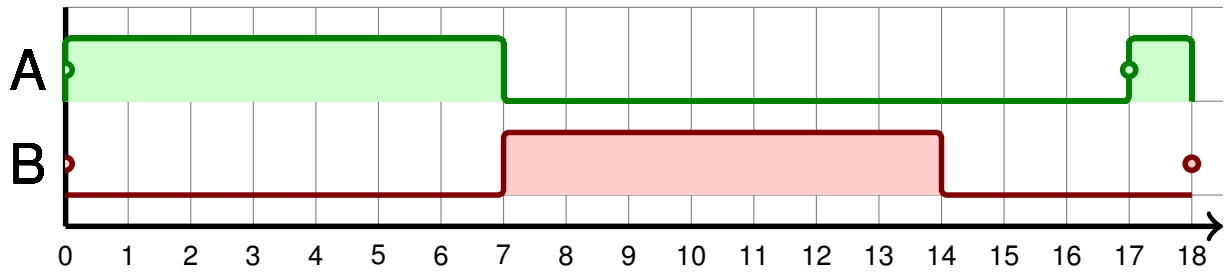


Par LST, en recalculant l'algorithme à chaque période :

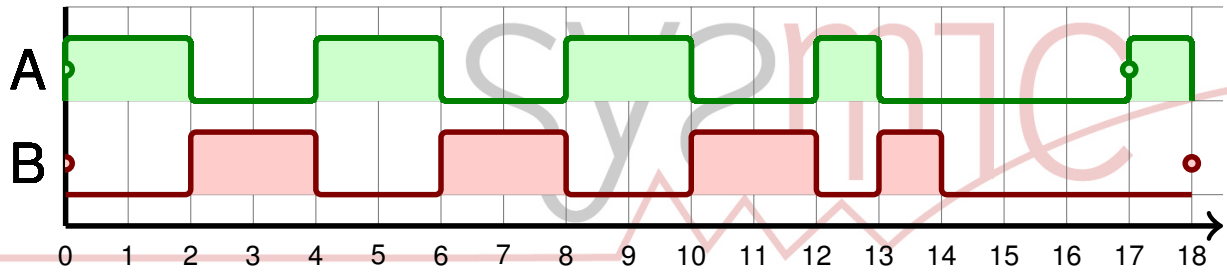
Exemple - 2

Tâche	Arrivée	Période	Capacité	Délai
A	0	17	7	Fin de période
B	0	18	7	Fin de période

Par EDF :



Par LST, en recalculant l'algorithme à chaque période :



Tâches apériodiques

- Tâches prises en compte dans une configuration comprenant déjà des tâches périodiques
- A priori, on ne connaît pas l'instant d'arrivée de la requête de réveil de la tâche apériodique
- Contraintes temporelles strictes ou relatives
- Buts à atteindre :
  - Maintenir les garanties du temps réelles sur les tâches déjà présentes dans l'ordonnanceur
  - Si contraintes relatives : minimiser le temps de réponse
  - Si contraintes strictes : maximiser le nombre de tâches acceptées en respectant leurs contraintes

Ordonnancement conjoint

- Algorithme fonctionnant avec DMA, EDF ou LST et des tâches apériodiques avec des délais critiques
- On ordonnance les tâches apériodiques de la même manière que les tâches périodiques
- Avant d'accepter la tâche, il faut vérifier le critère d'acceptabilité (ce qui correspond à évaluer l'algorithme hors ligne)

Traitement en arrière-plan

- Aussi appelé *backgound* ou sur *temps creux*
- Tâches apériodiques ordonnancées quand le processeur est oisif
  - Les tâches périodiques restent les plus prioritaires
- Ordonnancement relatif des tâches apériodiques en mode FIFO
- Traitement le plus simple, mais le moins performant
- Pas de marge de manoeuvre pour améliorer le temps de réponse des tâches apériodique. Potentiellement, les tâches apériodique peuvent avoir des temps de réponse long.

Traitement en arrière-plan

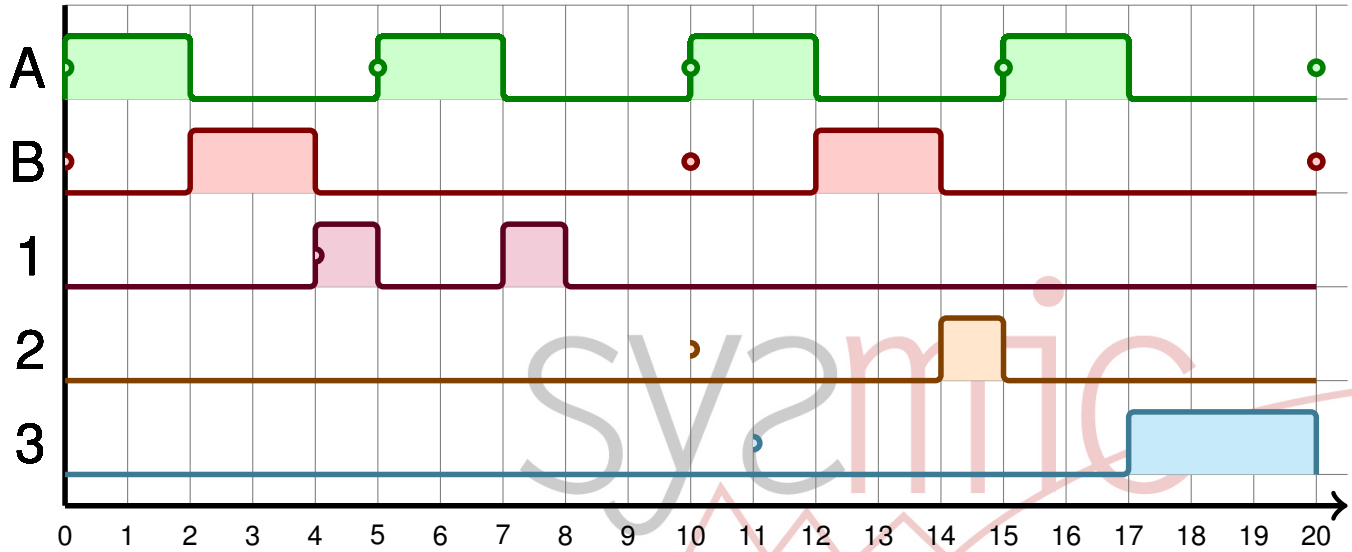
Avec ordonnancement RM des tâches périodiques :

Tâche	Arrivée	Période	Capacité	Délai
A	0	5	2	Fin de période
B	0	10	2	Fin de période
1	4	-	2	Aucune
2	10	-	1	Aucune
3	11	-	3	Aucune

Traitement en arrière-plan

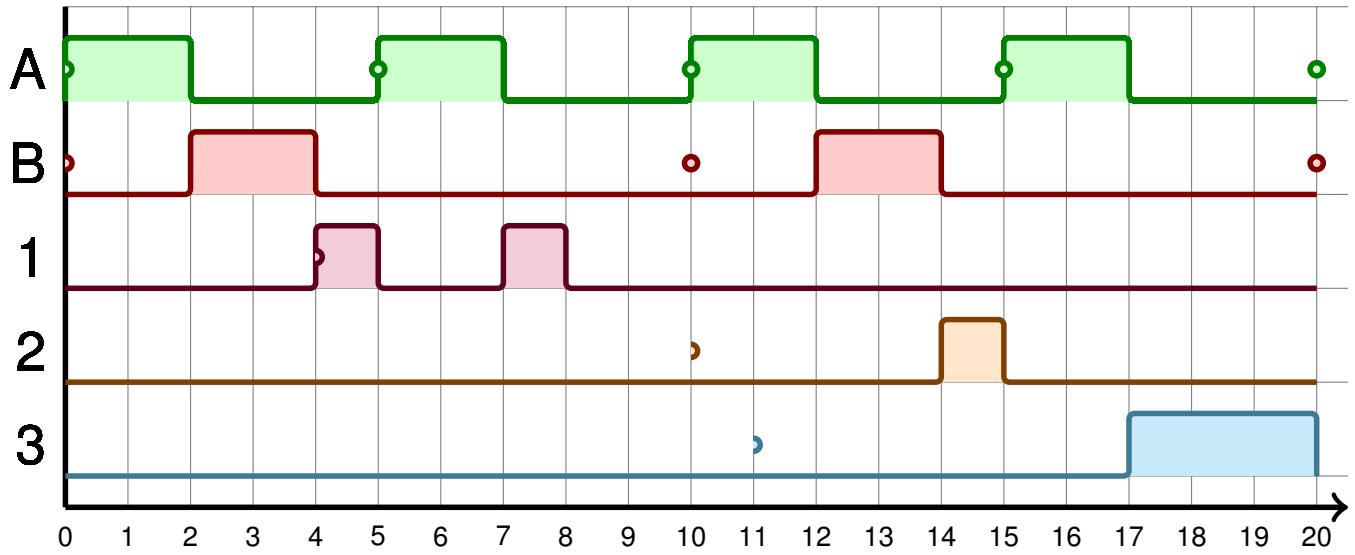
Avec ordonnancement RM des tâches périodiques :

Tâche	Arrivée	Période	Capacité	Délai
A	0	5	2	Fin de période
B	0	10	2	Fin de période
1	4	-	2	Aucune
2	10	-	1	Aucune
3	11	-	3	Aucune



Traitement en arrière-plan

Avec ordonnancement RM des tâches périodiques :



$$TR_1 = 4$$
$$TR_2 = 5$$
$$TR_3 = 9$$

Traitement par serveur

- Un serveur est une tâche périodique créée spécialement pour prendre en compte les tâches apériodiques
- Un serveur est caractérisé par :
  - Sa période
  - Son temps d'exécution : capacité du serveur
- Un serveur est généralement ordonnancé suivant le même algorithme que les autres tâches périodiques
- Très souvent, afin de diminuer le temps de réponses des tâches apériodiques, la priorité du serveur est haute (sinon, ses performances sont identiques au traitement sur temps creux)
- Une fois actif, le serveur sert les tâches apériodiques dans la limite de sa capacité.
- l'ordre de traitement des tâches apériodiques ne dépend pas de l'algorithme général
- Il est possible de le combiner le traitement avec un serveur avec un traitement en background (Temps de réponse+, prédictabilité-)

Serveur par scrutation

- Aussi appelé *polling*
- A chaque activation, traitement des tâches en suspens jusqu'à épuisement de la capacité ou jusqu'à ce qu'il n'y ait plus de tâches en attente
- Si aucune tâche n'est en attente (à l'activation ou parce que la dernière tâche a été traitée) , le serveur se suspend immédiatement et perd sa capacité qui peut être réutilisée par les tâches périodiques
- Quand une instance (un événement) de tâche apériodique arrive, elle attend jusqu'à ce que la capacité du serveur soit disponible.
- Il est possible de rendre la main au CPU si aucunes taches apériodiques n'est en attente (TR des taches périodiques +, prédictabilité -)
- Dans le cas ou le serveur à la plus petite priorité, l'algorithme équivaut à peu près au traitement en background

Exemple

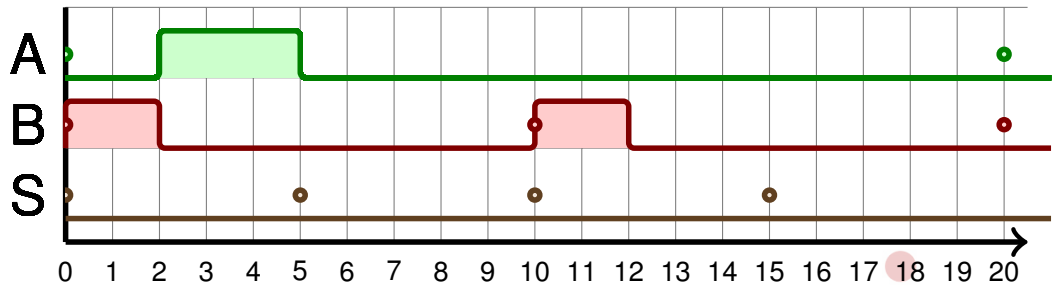
A vide, avec ordonnancement RM des tâches périodiques :

Tâche	Arrivée	Période	Capacité	Délai
A	0	20	3	Fin de période
B	0	10	2	Fin de période
S	0	5	2	Fin de période

Exemple

A vide, avec ordonnancement RM des tâches périodiques :

Tâche	Arrivée	Période	Capacité	Délai
A	0	20	3	Fin de période
B	0	10	2	Fin de période
S	0	5	2	Fin de période



Exemple

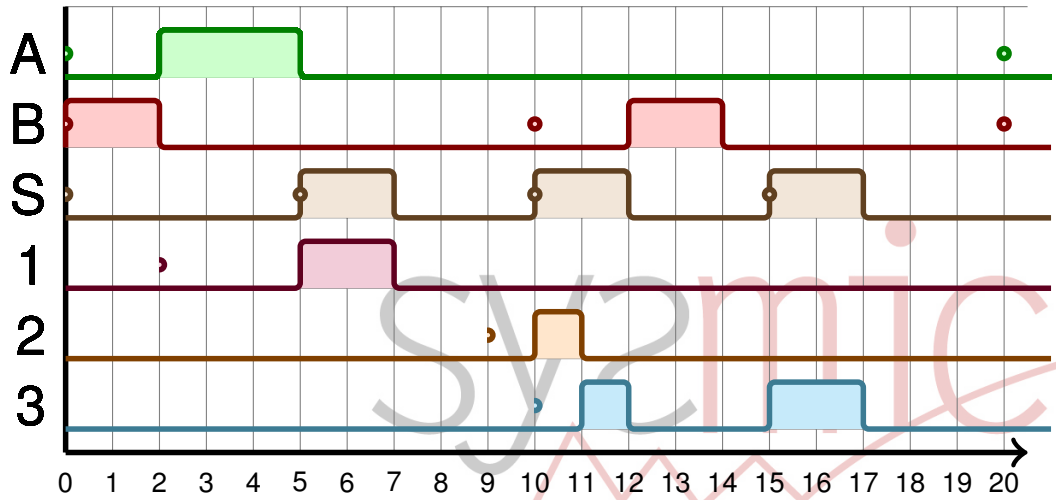
Idem avec les 3 tâches apériodiques :

Tâche	Arrivée	Période	Capacité	Délai
A	0	20	3	Fin de période
B	0	10	2	Fin de période
S	0	5	2	Fin de période
1	2	-	2	Aucune
2	9	-	1	Aucune
3	10	-	3	Aucune

Exemple

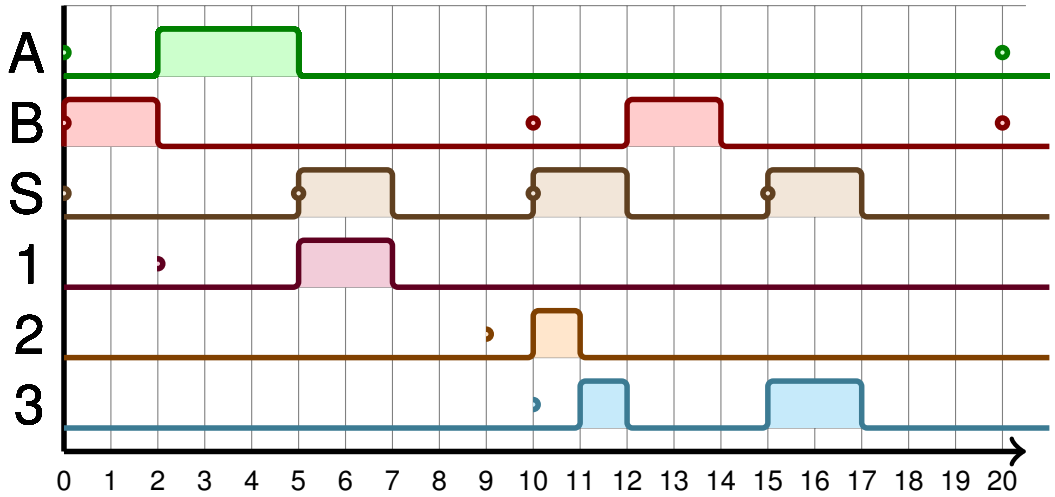
Idem avec les 3 tâches apériodiques :

Tâche	Arrivée	Période	Capacité	Délai
A	0	20	3	Fin de période
B	0	10	2	Fin de période
S	0	5	2	Fin de période
1	2	-	2	Aucune
2	9	-	1	Aucune
3	10	-	3	Aucune



Exemple

Idem avec les 3 tâches apériodiques :



$$TR_1 = 5$$

$$TR_2 = 2$$

$$TR_3 = 7$$

Exemple

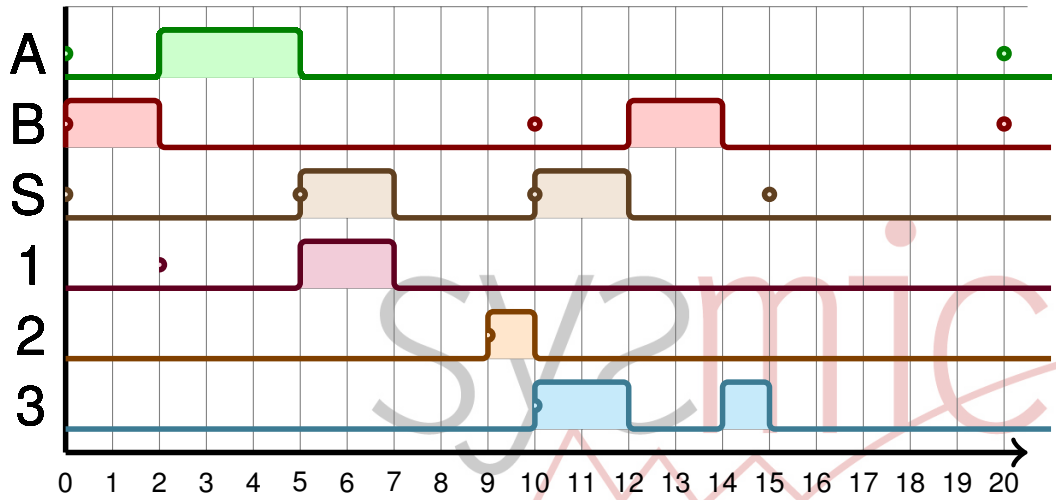
Idem, en utilisant les temps creux en plus :

Tâche	Arrivée	Période	Capacité	Délai
A	0	20	3	Fin de période
B	0	10	2	Fin de période
S	0	5	2	Fin de période
1	2	-	2	Aucune
2	9	-	1	Aucune
3	10	-	3	Aucune

Exemple

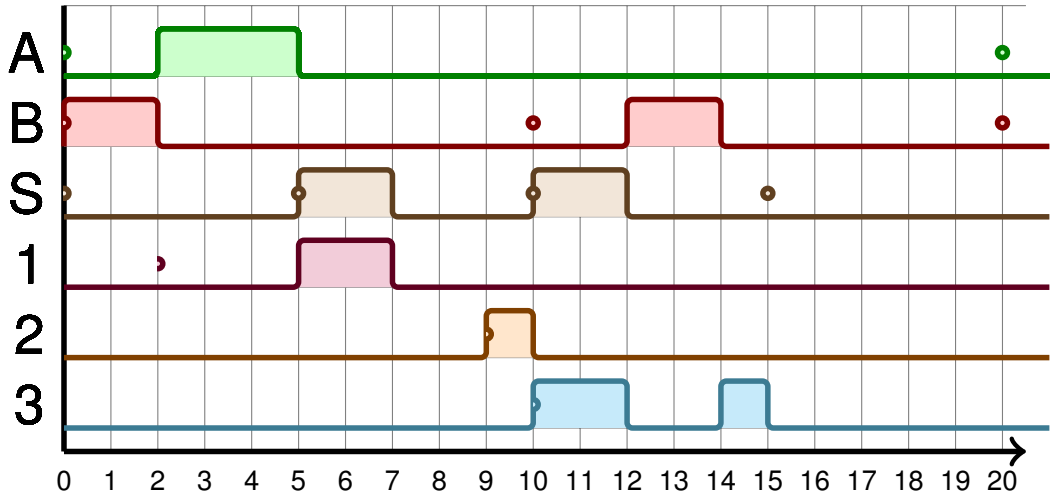
Idem, en utilisant les temps creux en plus :

Tâche	Arrivée	Période	Capacité	Délai
A	0	20	3	Fin de période
B	0	10	2	Fin de période
S	0	5	2	Fin de période
1	2	-	2	Aucune
2	9	-	1	Aucune
3	10	-	3	Aucune



Exemple

Idem, en utilisant les temps creux en plus :



$$TR_1 = 5$$

$$TR_2 = 1$$

$$TR_3 = 5$$

Limitation

Limitations du serveur par scrutation

- perte de la capacité si aucune tâche apériodique en attente
- si occurrence d'une tâche apériodique alors que le serveur est suspendu, il faut attendre la requête suivante

Serveur différé

- Aussi appelé serveur ajournable
- La fausse bonne idée
- Mal géré, il provoque des erreurs d'ordonnancement
- Cas d'exécutions *dos à dos* du serveur

Exemple

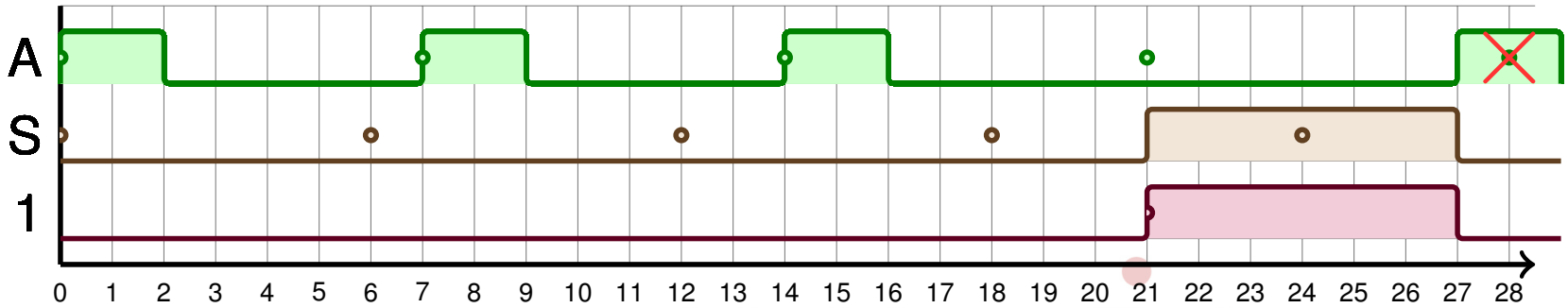
Avec ordonnancement RM :

Tâche	Arrivée	Période	Capacité	Délai
A	0	7	2	Fin de période
S	0	6	3	Fin de période
1	21	-	6	Aucune

Exemple

Avec ordonnancement RM :

Tâche	Arrivée	Période	Capacité	Délai
A	0	7	2	Fin de période
S	0	6	3	Fin de période
1	21	-	6	Aucune



Serveur différé

Tout de même possible de l'utiliser si :

$$\sum_{i \in TP} \frac{C_i}{T_i} \leq \ln \frac{U_S + 2}{2U_S + 1}$$

Grosso modo, on exige que le système soit ordonnançable avec une capacité du serveur deux fois plus importante.

Serveur différé

Tout de même possible de l'utiliser si :

$$\sum_{i \in TP} \frac{C_i}{T_i} \leq \ln \frac{U_S + 2}{2U_S + 1}$$

Grosso modo, on exige que le système soit ordonnançable avec une capacité du serveur deux fois plus importante.
Dans le cas précédant :

$$\frac{2}{7} \approx 0.286$$

$$\ln \frac{0.5+2}{2 \times 0.5 + 1} = \ln 1.25$$

$$\approx 0.223$$

$$\frac{2}{7} \not\leq \ln \frac{0.5+2}{2 \times 0.5 + 1}$$

Serveur sporadique

- Améliore le temps de réponse des tâches apériodiques sans diminuer le taux d'utilisation du processeur pour les tâches périodiques
- Très utilisé pour les IHM car permet une meilleure expérience utilisateur
- Comme le serveur différé mais
  - Ne retrouve pas sa capacité à période fixe, mais à un instant de réinitialisation égal à la date courante additionnée de la période de réinitialisation
  - La capacité retrouvée est égale à la capacité consommée

Exemple

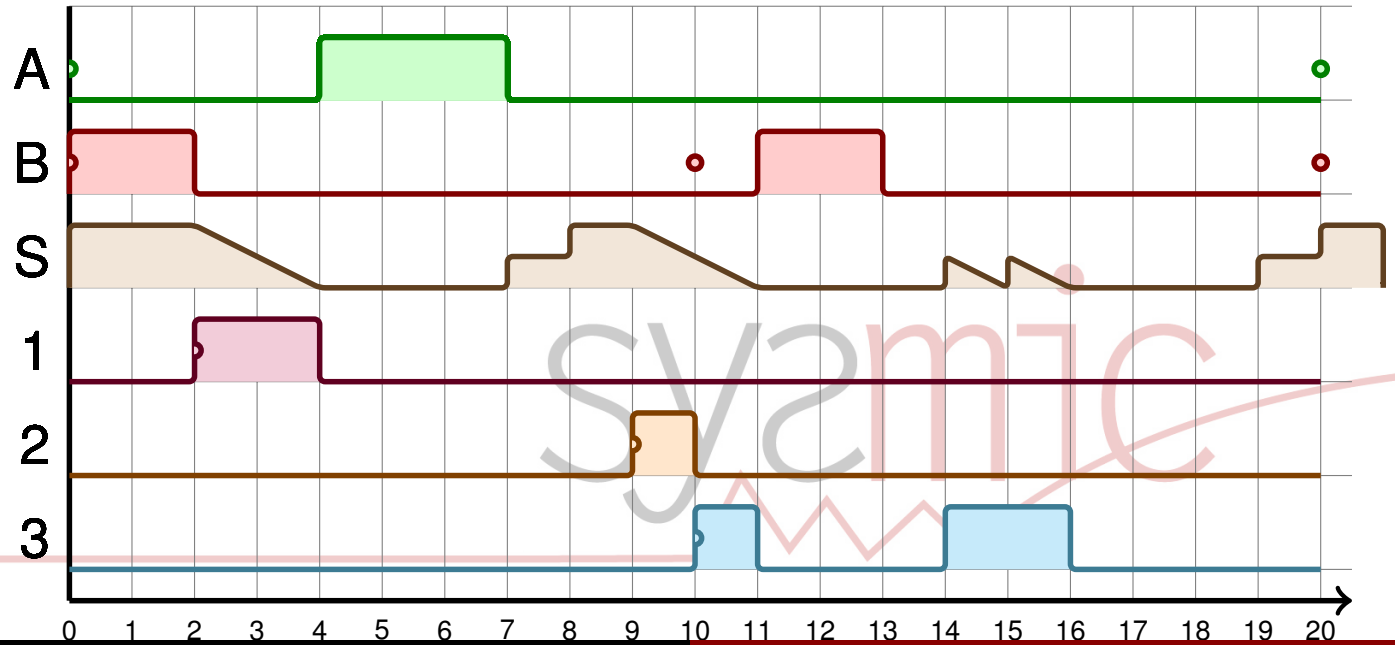
Avec ordonnancement RM :

Tâche	Arrivée	Période	Capacité	Délai
A	0	20	3	Fin de période
B	0	10	2	Fin de période
S	0	5	2	Fin de période
1	2	-	2	Aucune
2	9	-	1	Aucune
3	10	-	3	Aucune

Exemple

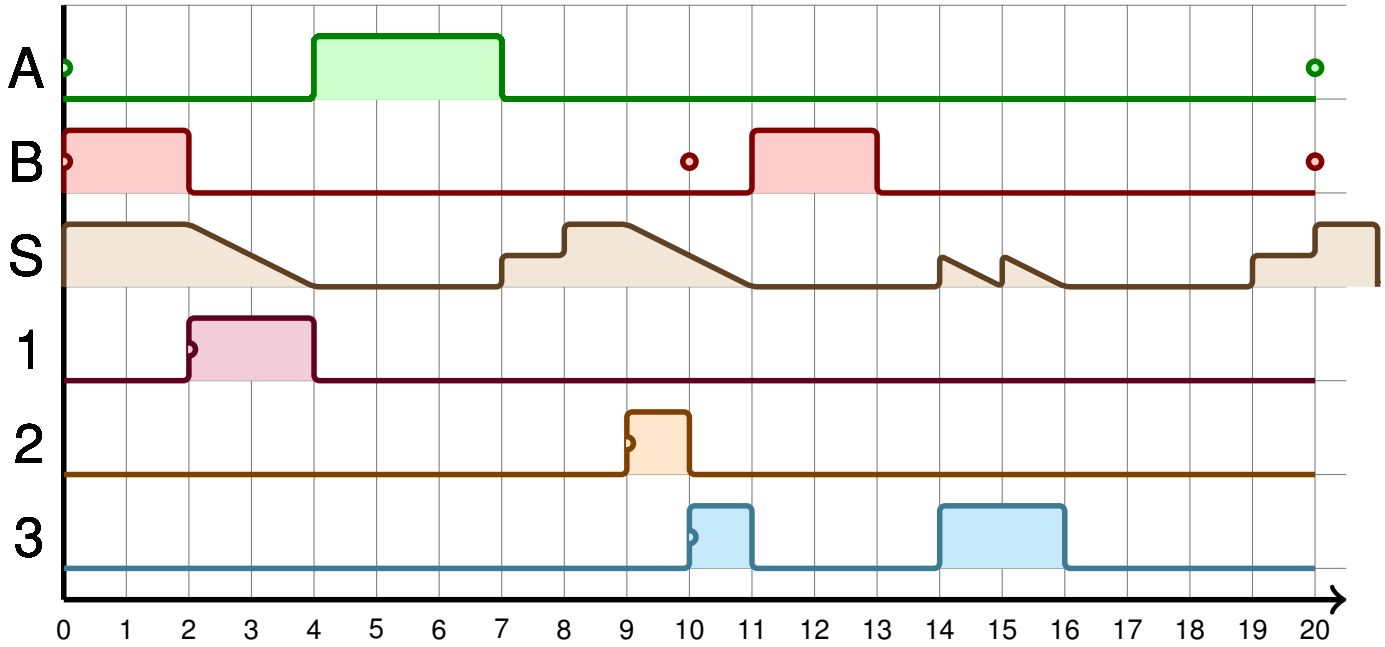
Avec ordonnancement RM :

Tâche	Arrivée	Période	Capacité	Délai
A	0	20	3	Fin de période
B	0	10	2	Fin de période
S	0	5	2	Fin de période
1	2	-	2	Aucune
2	9	-	1	Aucune
3	10	-	3	Aucune



Exemple

Avec ordonnancement RM :



$TR_1 = 2$

$TR_2 = 1$

$TR_3 = 5$

Slack Stealer

- On alloue $min_i LC_i(t)$ périodes pour l'exécution des tâches apériodiques
- On recalcule l'algorithme à chaque activation d'une tâche (périodique ou apériodique) dans le système
- Temps de réponses des tâches apériodiques optimal
- $LC_i(t)$ se calcule d'une manière similaire au temps de réponse
- Garantir le temps de réponse du calcul de $LC_i(t)$ est difficile
- Du coup, c'est le serpent qui se mange la queue : On connaît un algorithme optimal, mais il est incompatible avec les problématique du temps réel
- Principalement universitaire, peu utilisé dans l'industrie