Programmation Temps Réel

Ordonnancement de tâches apériodiques

Yann Thoma

Reconfigurable and Embedded Digital Systems Institute
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

Septembre 2017

Période d'étude

- Périodicité des tâches $\Rightarrow$ Ordonnancement cyclique
- L'algorithme d'ordonnancement consistera à trouver une séquence de tâches caractérisée par une périodicité de longueur L
- La période de longueur L est appelée période d'étude ou période de base.
- Ce qui signifie que si l'échéance de chacune des tâches est respectée dans la période d'étude, toutes les autres occurrences le seront. La séquence ainsi déterminée peut être reproduite à l'infini.
- Evaluation
  - Valide quel que soit l'algorithme d'ordonnancement
  - La période d'étude peut s'avérer très longue selon les relations entre périodes.
  - Adapté aux ordonnancements hors-ligne (simulation)

Période d'étude

- Si les tâches sont synchrones (débutent au même instant)
  - $L = [0, PPCM(P_i)]$
- Si les tâches sont asynchrones (ne débutent pas au même instant)
  - $L = [\min\{r_{i,0}\}, 2 \times PPCM(P_i) + \max\{r_{i,0}, r_j + D_j\}]$
  - Où $r_{i,0}$ est la date d'activation de la première occurence de la tâche périodique T_i
  - Et r_j est la date d'activation de la tâche apériodique T_j

Introduction

- Tâches périodiques
  - Facile de prévoir un ordonnancement
  - Possible de garantir l'ordonnançabilité
- Tâches apériodiques
  - Peuvent arriver n'importe quand
  - Difficile de prévoir un ordonnancement
  - Impossible de garantir l'ordonnançabilité
- Tâches sporadiques
  - Temps minimal entre 2 événements borné
  - Difficile de prévoir un ordonnancement
  - Possible de garantir l'ordonnançabilité

Introduction

- Tâches sporadiques
  - L'ordonnancement peut être garanti off-line
  - Pour des tâches à échéance stricte
- Tâches apériodiques
  - L'ordonnancement peut être garanti on-line
  - Pour des tâches à échéance ferme
    - A l'arrivée d'une tâche, elle est acceptée ou rejetée
  - Impossible pour des tâches à échéance stricte

Hypothèses

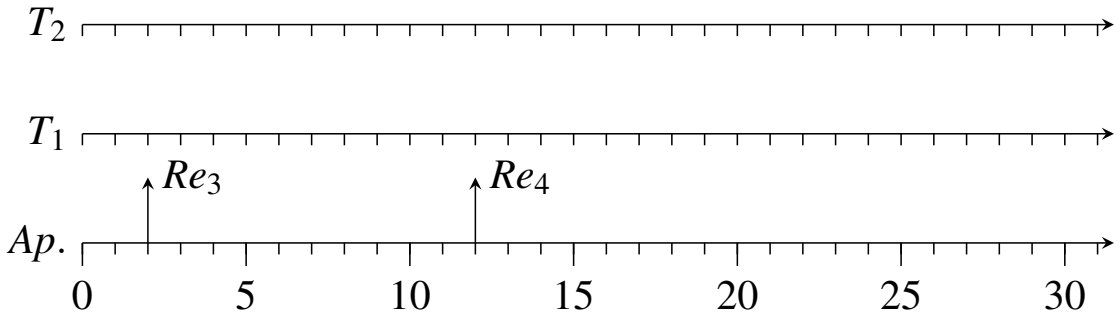
- Les tâches périodiques sont:
  - Agencées par RM
  - A échéance sur requête
  - Lancées au temps $t = 0$ (déphasage nul)
- Les temps d'arrivée des tâches apériodiques ne sont pas connus
- Le temps minimal d'activation d'une tâche sporadique est égal à son échéance
- Préemption
- single CPU

Traitement en arrière plan

- En anglais: *background scheduling* ou *background processing*
- Concept: Les tâches apériodiques sont exécutées lorsque le processeur est oisif
- Le temps de réponse peut être long
- Possible si les contraintes de temps ne sont pas strictes
- Et si les tâches périodiques ne chargent pas trop le processeur
- Avantage: grande simplicité d'implémentation

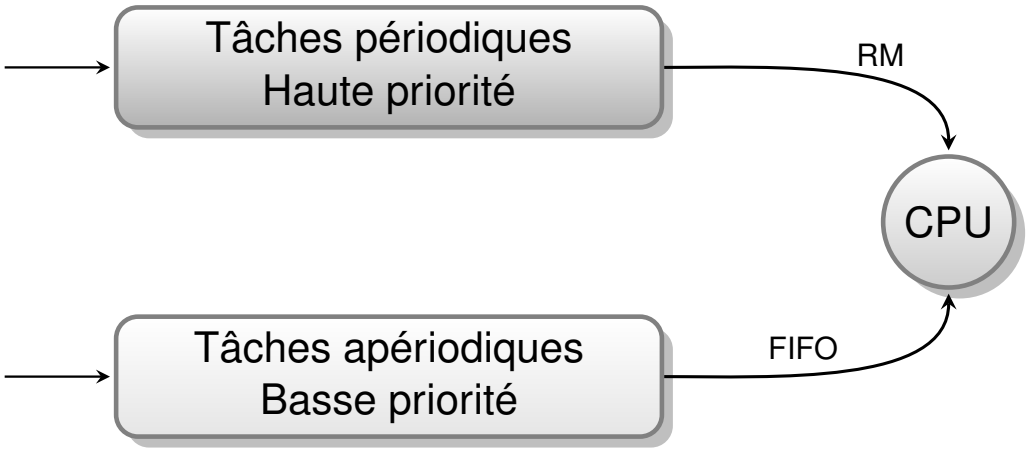
Traitement en arrière plan: exemple

Tâche périodique	Coût	Période
T_1	2	6
T_2	4	10



Files d'ordonnancement

- Implémentation grâce à deux files d'attente
- Les tâches de basse priorité sont servies si aucune tâche de haute priorité n'est prête



Traitement vs. serveur

- Un serveur est une tâche périodique
- Rôle:
  - servir les tâches apériodiques
- Implémentations:
  - Serveur à scrutation
  - Serveur ajournable
  - Serveur à échanges de priorité
  - Serveur sporadique
  - Serveur "Slack stealer"

Serveur à scrutation

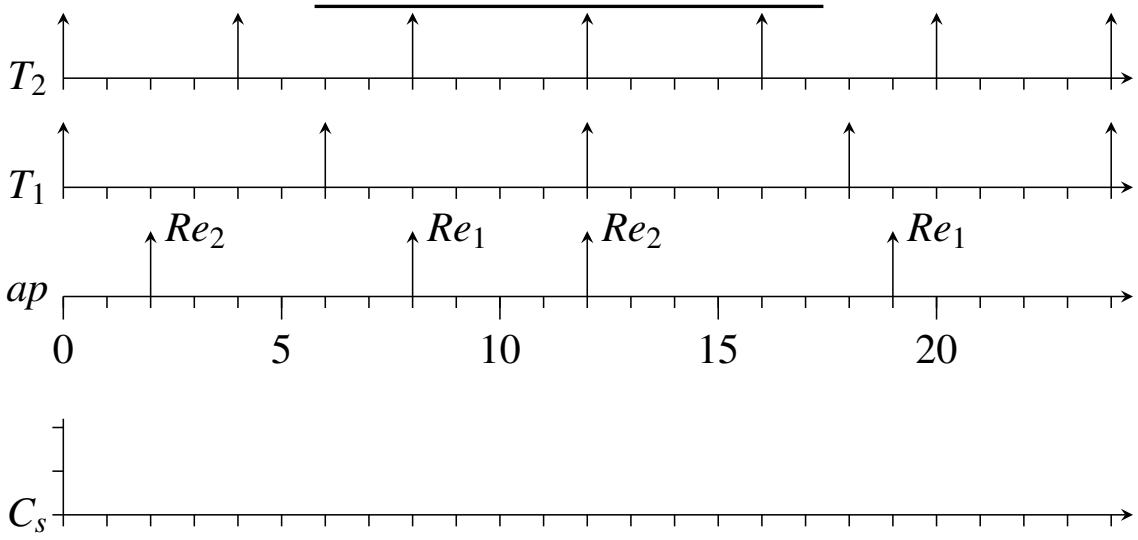
- Le serveur est une tâche périodique
- Il est caractérisé par:
  - Une période P_s
  - Un temps d'exécution C_s
    - Appelé *capacité du serveur*
- Ordonnancé selon le même algorithme que les tâches périodiques
- L'ordonnancement des tâches apériodiques peut être géré différemment

Gestion des tâches apériodiques

- A chaque activation du serveur, il vérifie s'il y a des tâches apériodiques prêtes
  - Si non, il s'endort
  - Si oui, il les sert
- Si une tâche apériodique arrive pendant le temps réservé au serveur, elle est servie au tour suivant

Serveur à scrutation: exemple

Tâche	Coût	Période
T_1	2	6
T_2	1	4
T_s	2	5



Serveur à scrutation: ordonnançabilité

- Le serveur est une tâche périodique, donc nous pouvons appliquer la règle de RM

Condition suffisante d'ordonnançabilité

$$\sum_{i=1}^{n} \frac{C_i}{P_i} + \frac{C_s}{P_s} \leq U_{lub_{RM}}(n+1) = (n+1)(2^{\frac{1}{n+1}} - 1) \quad (1)$$

- Il est possible d'avoir plusieurs serveurs, avec chacun une priorité différente
- Pour m serveurs, nous avons:

Condition suffisante d'ordonnançabilité

$$\sum_{i=1}^{n} \frac{C_i}{P_i} + \sum_{i=1}^{m} \frac{C_{s_i}}{P_{s_i}} \leq U_{lub_{RM}}(n+m) = (n+m)(2^{\frac{1}{n+m}} - 1) \quad (2)$$

Serveur à scrutation: garanties apériodiques

- L'échéance d'une tâche apériodique peut-elle être garantie?
  - Si $C_a \leq C_s$ alors:
    - La tâche doit attendre au plus une période pour commencer
    - La condition est donc:

$$2P_s \leq D_a \tag{3}$$

  - Dans le cas général, la condition est:

$$P_s + \left\lceil \frac{C_a}{C_s} \right\rceil P_s \leq D_a \tag{4}$$

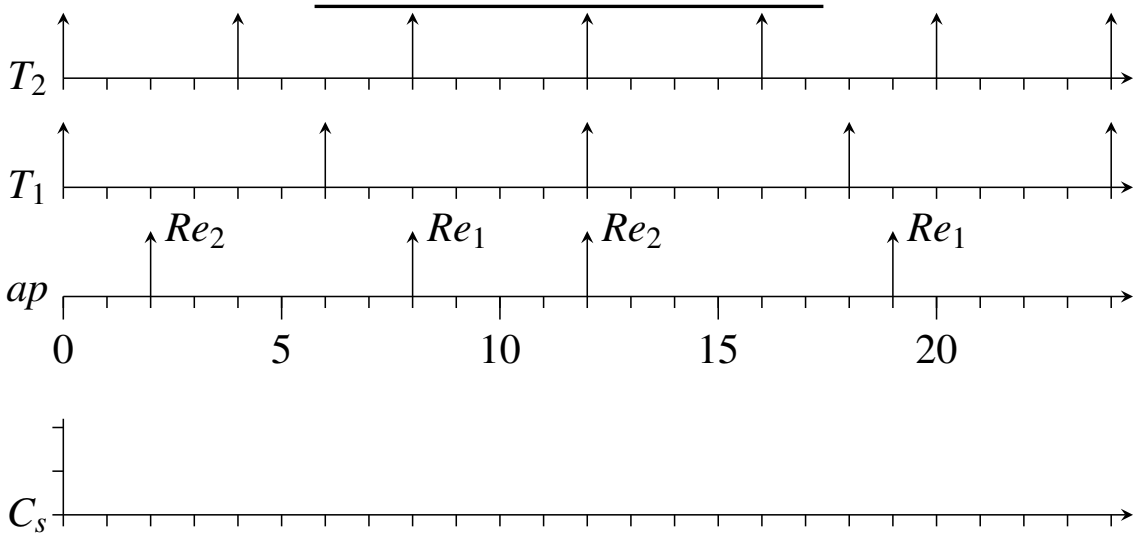
  - Ceci n'est valable que pour une tâche unique (calcul plus délicat pour des tâches multiples)

Serveur ajournable

- Problèmes du serveur à scrutation:
  - S'il n'y a pas de tâche apériodique à traiter, le temps du serveur est perdu
  - Si une tâche apériodique arrive pendant la période de traitement, elle n'est pas directement exécutée
- Serveur ajournable (*deferrable server*)
  - La capacité est sauvegardée si non utilisée
  - Permet un meilleur temps de réponse

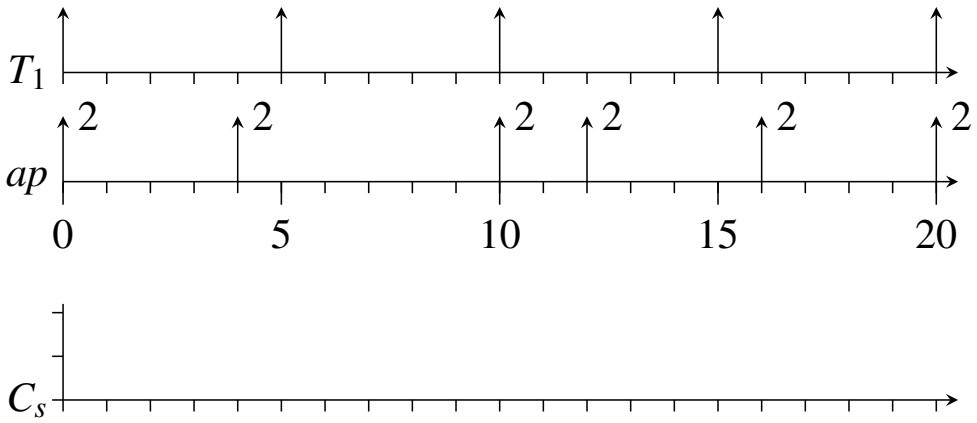
Serveur ajournable: exemple

Tâche	Coût	Période
T_1	2	6
T_2	1	4
T_s	2	5



Serveur ajournable: problème

Tâche	Coût	Période
T_1	2	5
T_s	2	4



Serveur ajournable: ordonnançabilité

- Problème: l'ajournement du traitement peut compromettre l'exécution d'une tâche périodique
- La condition d'ordonnançabilité devient donc:

Condition suffisante d'ordonnançabilité

$$\sum_{i=1}^{n} \frac{C_i}{P_i} \leq \ln\left(\frac{U_s + 2}{2U_s + 1}\right) \tag{5}$$

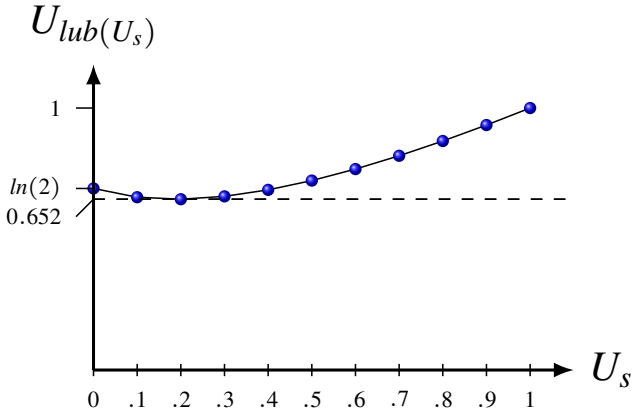
Et:

$$U_{lub} \simeq 0.652$$

Pour $U = U_p + U_s$, et obtenu avec:

$$U_s = \frac{\sqrt{33} - 5}{4} \simeq 0.186$$

Least Upper Bound



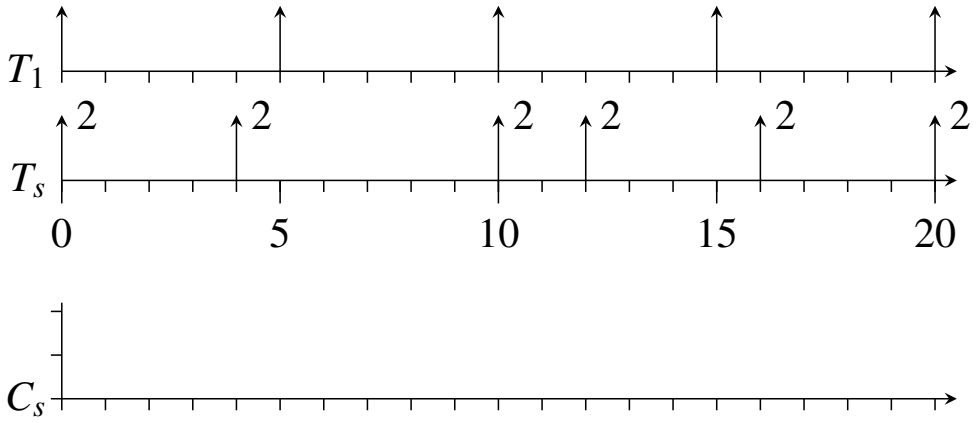
- Valable pour

$$U = \sum_{i=1}^{n} \frac{C_i}{P_i} + U_s < U_{lub}$$

Serveur ajournable: autre exemple

Tâche	Coût	Période
T_1	1	5
T_s	2	4

Ordonnançable?



Serveur sporadique

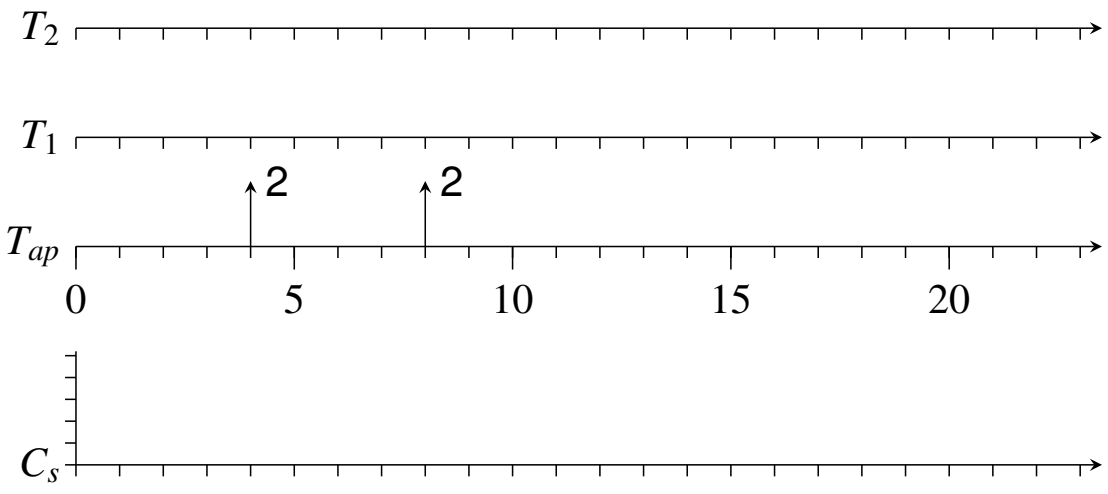
- But: améliorer le temps de réponse des tâches apériodiques
- La politique de recharge du serveur et différente des deux versions précédentes
- Il possède une capacité C_s et une période P_s
- Le serveur consiste en une tâche qui s'exécute lorsqu'il y a conjointement:
  - Des tâches apériodiques à servir
  - Sa capacité courante est non nulle
  - Sa priorité est éligible

Serveur sporadique

- Soit:
  - P_{exe} la priorité de la tâche actuellement en cours d'exécution
  - P_s la priorité du serveur sporadique
- Alors:
  - Le serveur est dit *actif* lorsque $P_{exe} \geq P_s$
  - Le serveur est dit *au repos (idle)* lorsque $P_{exe} < P_s$
- Lorsqu'il s'exécute, sa capacité diminue
- Sa capacité est restituée un temps P_s après le début de son activation
- La quantité restituée est égale à la capacité consommée entre son activation et la fin de sa période d'activation

Serveur sporadique: exemple

Tâche	Coût	Période
T_1	1	5
T_2	4	15
T_s	5	10



Comparaison: scrutation vs. sporadique

- Serveur à scrutation de tâches
  - Facile à implémenter
  - Ordonnançable avec RM
  - Le serveur est une tâche périodique à priorité fixe.
  - Gaspillage du temps CPU
  - La capacité non utilisée est perdue.
  - Fonctionne bien si peu de tâches apériodiques (la capacité peu être déterminée de manière optimale).
- Serveur sporadique
  - Meilleure utilisation du CPU
  - Bien adapté si beaucoup de tâches apériodiques
  - Plus difficile à implémenter

Comparaison (2)

Algorithme	**Perf.**	**Complex. comput.**	**Mémoire**	**Complex. d'impl.**
Background	☹	🙂	🙂	🙂
Scrutation	☹	🙂	🙂	🙂
Ajournable	😐	🙂	🙂	🙂
Sporadique	😐	😐	😐	😐
Priority exchange	😐	😐	😐	😐
Slack stealer	🙂	☹	☹	☹