SEA

Ordonnancement Temps-Réel

Pablo de Oliveira

Introduction

Plan du cours

Systèmes temps-réel

- Présentation des systèmes temps-réel
- Tâches périodiques et sporadiques / priorités
- Ordonnançement
- Problèmes: Inversion de priorité, Famine, Race Conditions
- Polling actif ou évenements

Système Temps-Réel

> *In real-time computing the correctness of the system depends not only on the logical result of the computation but also on the time at which results are produced. J.A. Stankovic, IEEE Computer 1988.*

Exemples de systèmes temps-réel:

- Centrale nucléaire
- Contrôle vitesse d'un train
- Robotique
- Avionique

Objectifs

- Déterminisme logique: les sorties sont déterminées par les entrées et éventuellement l'état logique interne.
- Déterminisme temporel: le système respecte les échéances temporelles
- Un système temps-réel n'est pas forcément un système rapide, c'est un système qui garantit des contraintes temporelles.

Deux types de systèmes Temps-Réel

- Temps-réel mou: un résultat en retard diminue la qualité de service, mais même en retard le résultat est toujours utile.
  - Exemple: un distributeur de billets devrait répondre à la pression d'une touche en moins de $250ms$, un retard diminue l'ergonomie, mais n'est pas dramatique.
- Temps-réel dur: un résultat en retard est inutile
  - Exemple: déclenchement de l'airbag dans une voiture

Types de contraintes

- Précision: il faut effectuer une action à un moment précis dans le temps (horloge électronique).
- Temps de réponse: effectuer une action avant son échéance (freinage pour un système de contrôle de vitesse) ou avec un temps moyen fixé (décompresseur video, 24 images par secondes environ).
- Rendement: nombre de requêtes traitées par unité de temps (robot de production dans une usine)

Types de tâches

- Tâches périodiques, la tâche est activée à intervalles réguliers avec une durée C_i et une période T_i.
- Tâches sporadiques, la tâche s'active à un instant quelconque, néanmoins il existe un temps minimal entre deux activations
- Tâches apériodiques, on ne peut faire aucune hypothèse sur l'instant d'activation des tâches

Ordonnancement

Architecture mono-coeur ou multi-coeur

- Ordonnancement sur un processeur:
  - Plusieurs tâches s'exécutent sur un seul processeur. Un ordonnanceur, passe d'une tâche à l'autre pour simuler une execution concurrente.
- Ordonnancement sur multi-processeur:
  - Plusieurs tâches s'exécutent sur plusieurs processeurs, en parallèle.

Priorités

- Les tâches peuvent être ordonnées par niveau de priorité
- Les priorités peuvent être statiques ou changer au cours du temps (dynamiques).
- En général dans un système temps-réel:
  - Une tâche n'est jamais bloqué par une tâche de moindre priorité (sauf problème d'inversion de priorité)
  - On minimise les changements de contexte entre tâches de même priorité

Ordonnancement

- Comment répartir le temps processeur parmi un groupe de tâches ?
  - Peut-on garantir qu'aucune tâche ne ratera une échéance ?
- Algorithmes statiques: table d'exécution ou analyse rate monotonic
- Algorithmes dynamiques: EDF, LLF, ...

Table d'exécution

- L'exécution des tâches est complétement statique.
- L'ordre d'exécution des tâches est pilotée par une table donnée par l'utilisateur.
- Peut-être vérifé pour des tâches périodiques en considérant le PPCM de toutes les périodes.
- Avantages:
  - Ordre d'exécution connu et déterministe
  - Algorithme simple
- Désavantage:
  - Comment gérer des tâches sporadiques ?
  - Les caractéristiques du systèmes doivent êtres connues en avance

Rate monotonic analysis

- Hypothèses:
  - Pas d'intéraction entre taches (Pas de partage de ressources)
  - Taches périodiques de période T_i et de durée C_i.
  - L'échéance des tâches D_i est supposée égale à T_i. Çàd, une tâche doit être complétée avant sa prochaine activation.
  - Les tâches sont préemptibles.
  - On suppose le côut du changement de contexte négligeable.
  - Un seul processeur

Utilisation du processeur

- Taux d'utilisation: quantité de temps de traitement par les tâches par rapport au temps total.
  - L'utilisation du processeur est définie comme $U = \sum_{i=1}^{n} \frac{C_i}{T_i}$

Rate monotonic analysis

- La priorité d'une tâche i est égale à $P_i = \frac{1}{T_i}$.
- Plus la fréquence est forte, plus la tâche est prioritaire.
- C'est toujours la tâche la plus prioritaire qui est exécutée d'abord.
- Si toutes les périodes sont multiples entre elles (système harmonique)
  - le système admet un ordonnancement *ssi* $U \leq 1$.

Rate monotonic analysis

- Si le système n'est pas harmonique
  - [Liu & Layland '73] Si $U \leq n.(2^{\frac{1}{n}} - 1)$ alors il existe un ordonnancement satisfaisant toutes les échéances.
  - C'est une condition *suffisante*, mais *pas* une condition *nécessaire*.
  - On montre $\lim_{n\to\infty} n(2^{1/n} - 1) = ln(2) \simeq 0.69$
  - Si la charge des tâches représente moins que 69% du temps processeur, le système admet un ordonnancement RM.

Rate monotonic analysis

- [Leung & Whitehead '82] Le système RM est optimal parmi les ordonnancements à priorité statique. Çàd si n'importe quel ordonnancement *à priorité statique* peut ordonanncer un système de tâches, alors RM le peut aussi.

Théorème de la zone critique

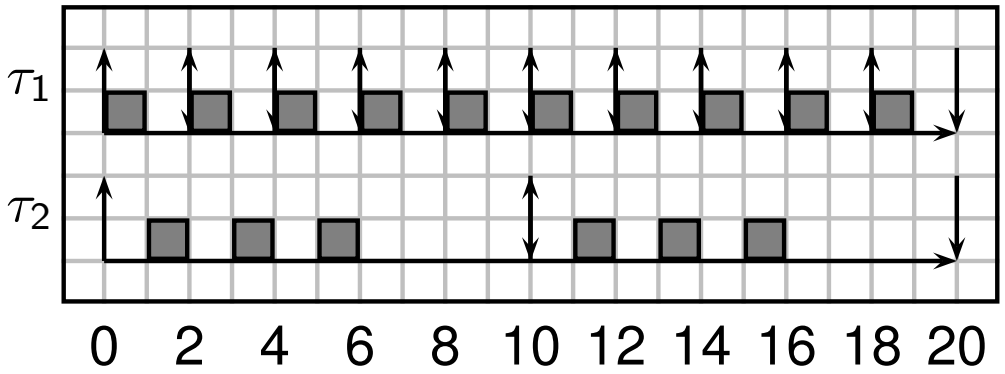
Si l'ensemble des tâches arrivent dans le système simultanément et si elles respectent leur première échéance en RM, alors toutes les tâches respecteront leurs échéances futures. [Liu & Layland '73]

Exemple 1

- Système de deux tâches $T_1 = 2s, C_1 = 1s$ et $T_2 = 10s, C_2 = 3s$.
- Le système est-il ordonnançable avec la méthode RM ?

Exemple 1

- Système de deux tâches $T_1 = 2s, C_1 = 1s$ et $T_2 = 10s, C_2 = 3s$.
- Utilisation $1/2 + 3/10 = 80\%$
- Or $80\% \leq 2(2^{1/2} - 1) \simeq 83\%$, ordonnançable !
- On aurait pu aussi utiliser $U \leq 1$ car le système est harmonique.

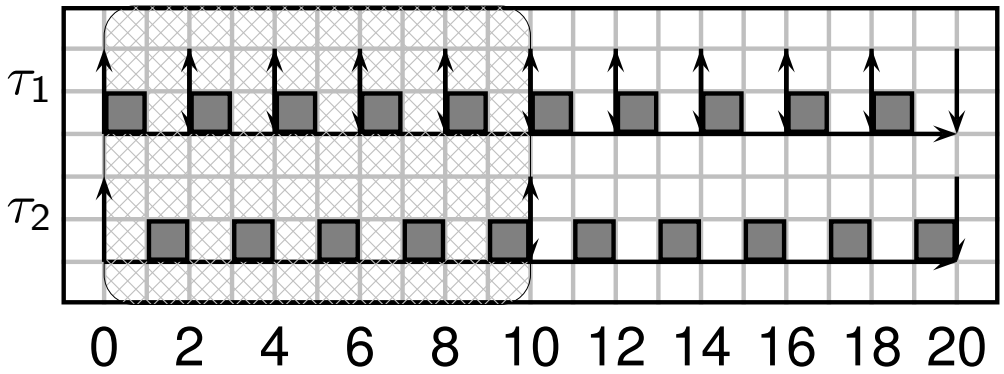


Exemple 2

- Système de deux tâches $T_1 = 2s, C_1 = 1s$ et $T_2 = 10s, C_2 = 5s$.
- Le système est-il ordonnançable ?

Exemple 2

- Système de deux tâches $T_1 = 2s, C_1 = 1s$ et $T_2 = 10s, C_2 = 5s$.
- Utilisation $1/2 + 5/10 = 100\%$
- Le système est harmonique, comme $U \leq 1$ il est ordonnançable.
- On aurait pu aussi utiliser le théorème de la zone critique:



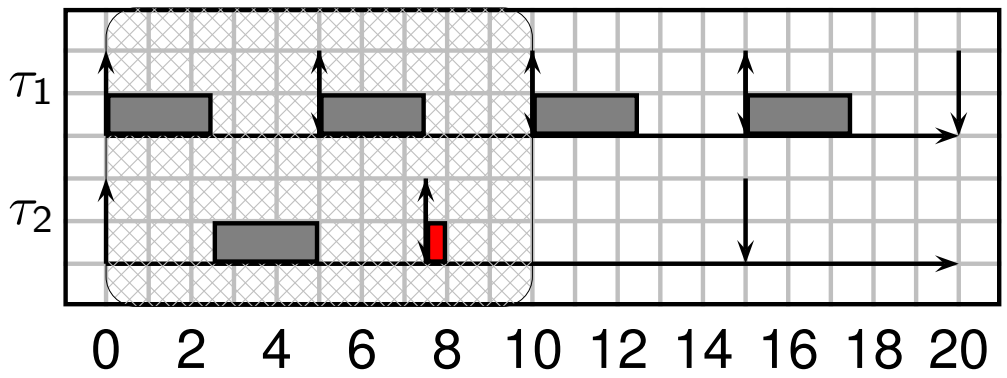
- Mais système ordonnonçable, théorème de la zone critique.

Exemple 3

- Système de deux tâches $T_1 = 5s, C_1 = 2.5s$ et $T_2 = 7.5s, C_2 = 3s$.
- Le système est-il ordonnançable ?

Exemple 3

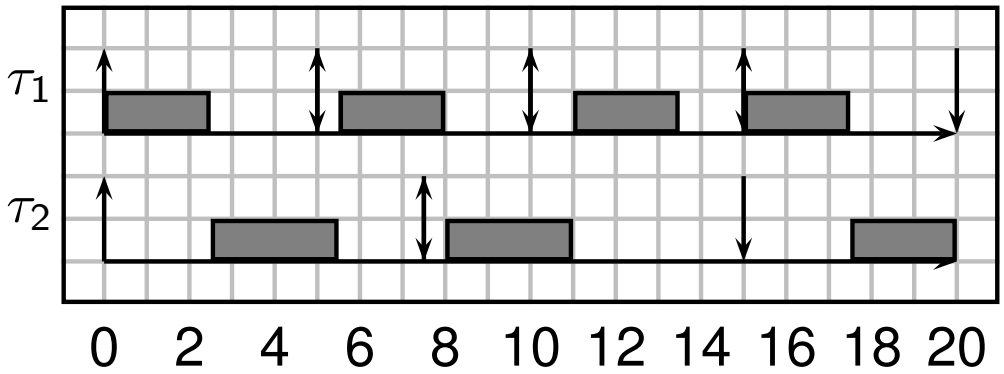
- Système de deux tâches $T_1 = 5s, C_1 = 2.5s$ et $T_2 = 7.5s, C_2 = 3s$.
- Utilisation $2.5/5 + 3/7.5 = 90\%$
- Or $90\% > 2(2^{1/2} - 1) \simeq 83\%$, on ne peut rien conclure...



- Théorème de la zone critique: le système n'est pas ordonnançable en RM.

Exemple 3

- Mais si on s'afffranchit de RM ...



- ... le système est ordonnançable.

Rate monotonic sur multi-processeur ?

- Problème ouvert.
- Travaux de recherche en cours.
- Voir par exemple, Min-Allah et al. J. Parallel Distib. Computing 2011.

Test d'ordonnancement pour tâches irrégulières

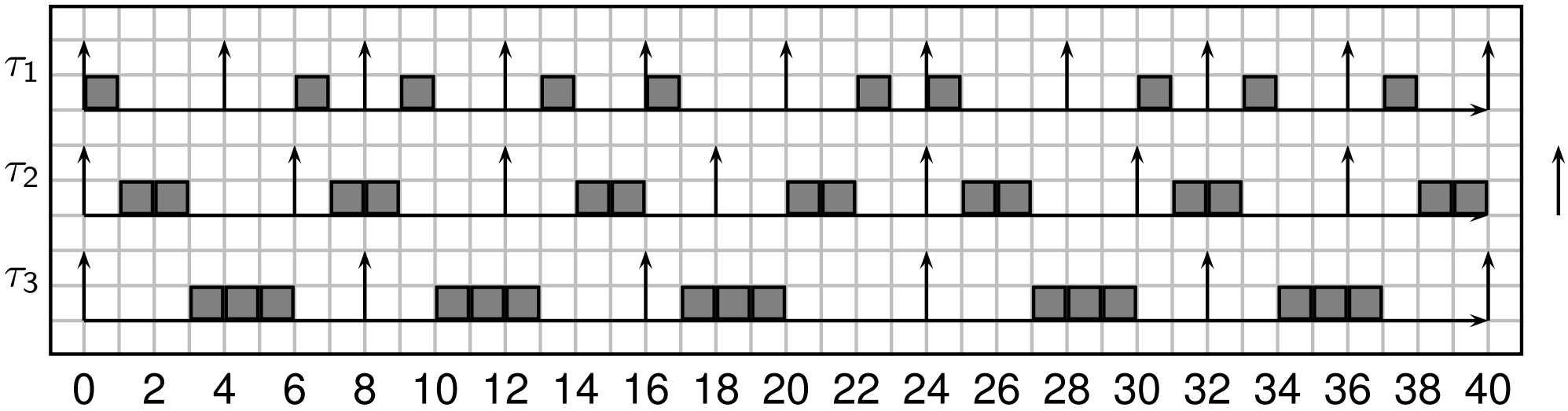
- Problème ouvert.

Ordonnancement dynamique (EDF)

- EDF: Earliest Deadline First
- À tout instant t, le travail le plus prioritaire est celui dont l'échéance est la plus courte.
- Condition d'ordonnançabilité avec EDF (pour des tâches périodiques et $D_i = T_i$) : $U \leq 1$.
- EDF est optimal: si un ensemble de tâches est ordonnançable, alors il est ordonnançable avec EDF.

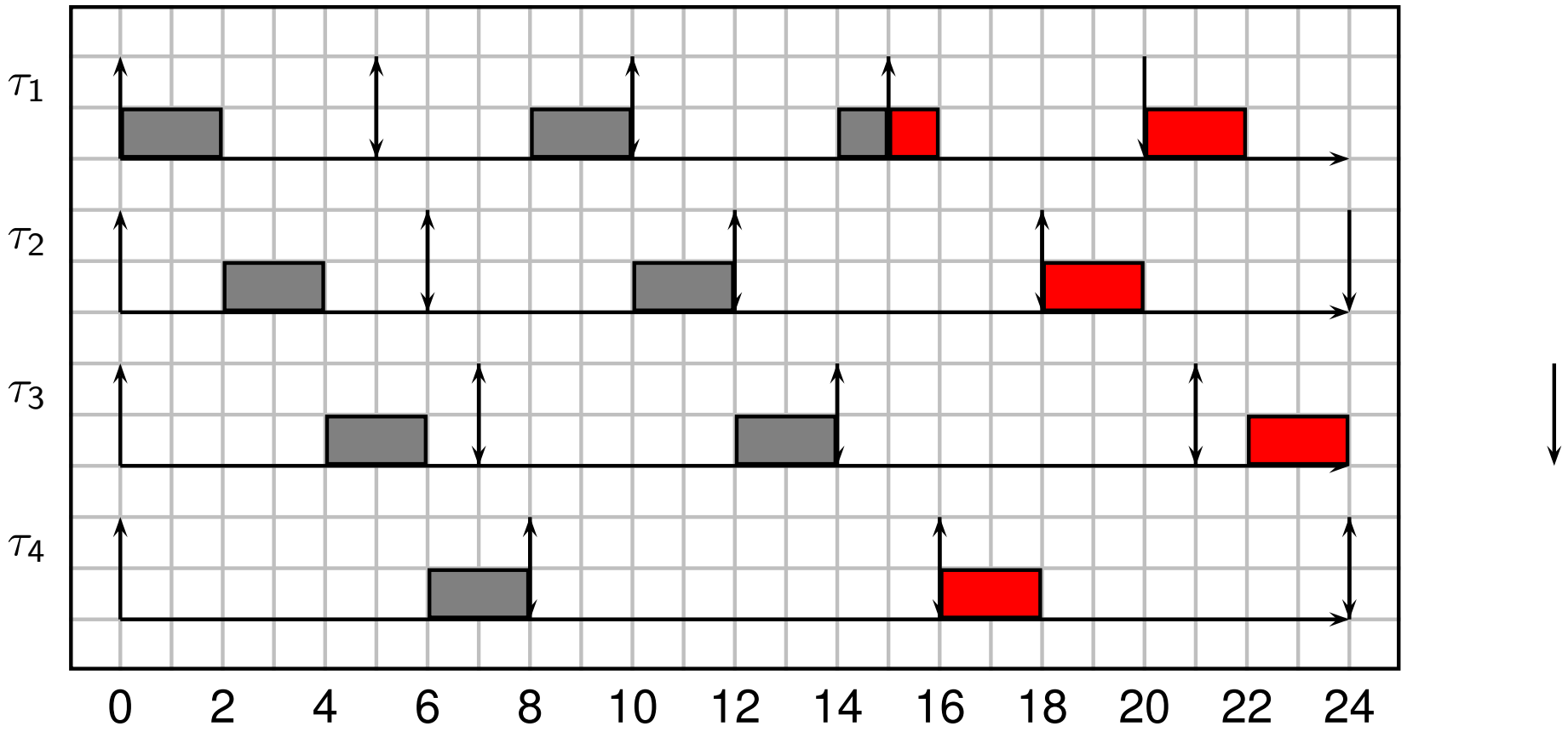
EDF

- Système de trois tâches:
  - $C_1 = 1s,\ T_1 = 4s$
  - $C_2 = 2s,\ T_2 = 6s$
  - $C_3 = 3s,\ T_3 = 8s$



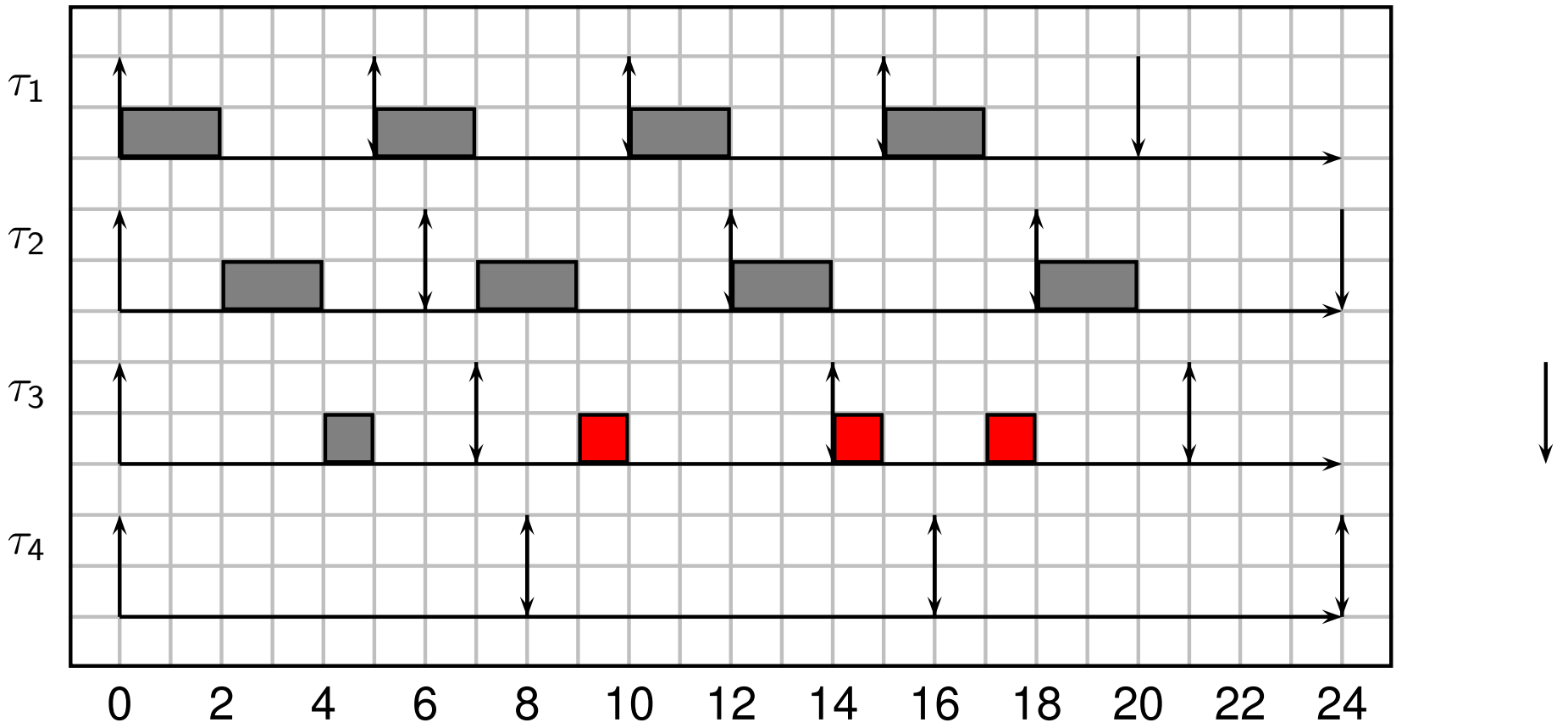
Effet Domino

- Comportement très mauvais en cas de surcharge.
- Avalanche d'échéance manquées



Effet Domino

- Rate Monotonic à un meilleur comportement, les tâches de priorité faible sont affectées les premières.
- Mais certaines tâches peuvent ne jamais être exécutées.



Comparaison RM et EDF

- RM avantages:
  - Très simple à implémenter
  - Prédictible
  - Priorités statiques
- RM inconvénients:
  - Modèle restrictif
  - Utilisation du processeur sous-optimale
- EDF avantages:
  - Simple à implémenter
  - Bonne utilisation du processeur
  - Réactif pour les tâches à échéances courtes
- EDF inconvénients:
  - Mauvais comportement en cas de surcharge (effet domino)
  - Moins prédictible

Problèmes liés à l'exécution concurrente

- Partage de ressources:
  - Partage de temps processeur
  - Accès concurrent à la mémoire
  - Accès concurrent aux périphériques
- Problèmes:
  - Cohérence mémoire
  - Dead-lock
  - Race-condition
  - Famine
  - Inversion de priorité

Sémaphore

- Pour protéger l'accès aux ressources partagées on utilise des sémaphores
- Un sémaphore est composé de:
  - Un compteur S_n initialisé lors de la création du sémaphore
  - Une liste de tâches en attente, initialement vide
- Il comporte deux opérations:
  - $P(S)$: prise du sémaphore (on demande une ressource)
  - $V(S)$: relachement du sémaphore (on libère une ressource).

Sémaphore

- $P(S)$
  - On décrémente S_n
  - Si $S_n < 0$ on met la tâche en attente et on la bloque. La tâche ne sera relachée que lorsque S_n deviendra positif.
- $V(S)$
  - On incrémente S_n
  - On débloque la première tâche en attente.

Vérrou

- Un vérrou est un sémaphore avec S_n initialisé à 1.
- Une seule tâche à la fois peut accéder à la ressource:

```
P(S)
accès ressource
V(S)
```

Implémentation d'un Vérrou

- Pour implémenter un vérrou il faut utiliser des opérations atomiques.
- Par exemple sur ARM on peut utiliser SWP qui effectue une lecture suivie d'une écriture de manière atomique:

```
#define LOCK 0
MOV r0, #LOCK
lock:
 SWP r0,r0, [verrou] @ échange atomique r0 et verrou
 CMP r0, #LOCK       @ verrou déjà pris ?
 BEQ lock            @ alors on essaye à nouveau
```

(Les ARMv6 proposent une manière plus efficace en utilisant les instructions LDREX et STREX.)

Deadlock

- Un Deadlock (interblocage) se produit lorsqu'au moins deux processus s'attendent mutuellement.
- Exemple

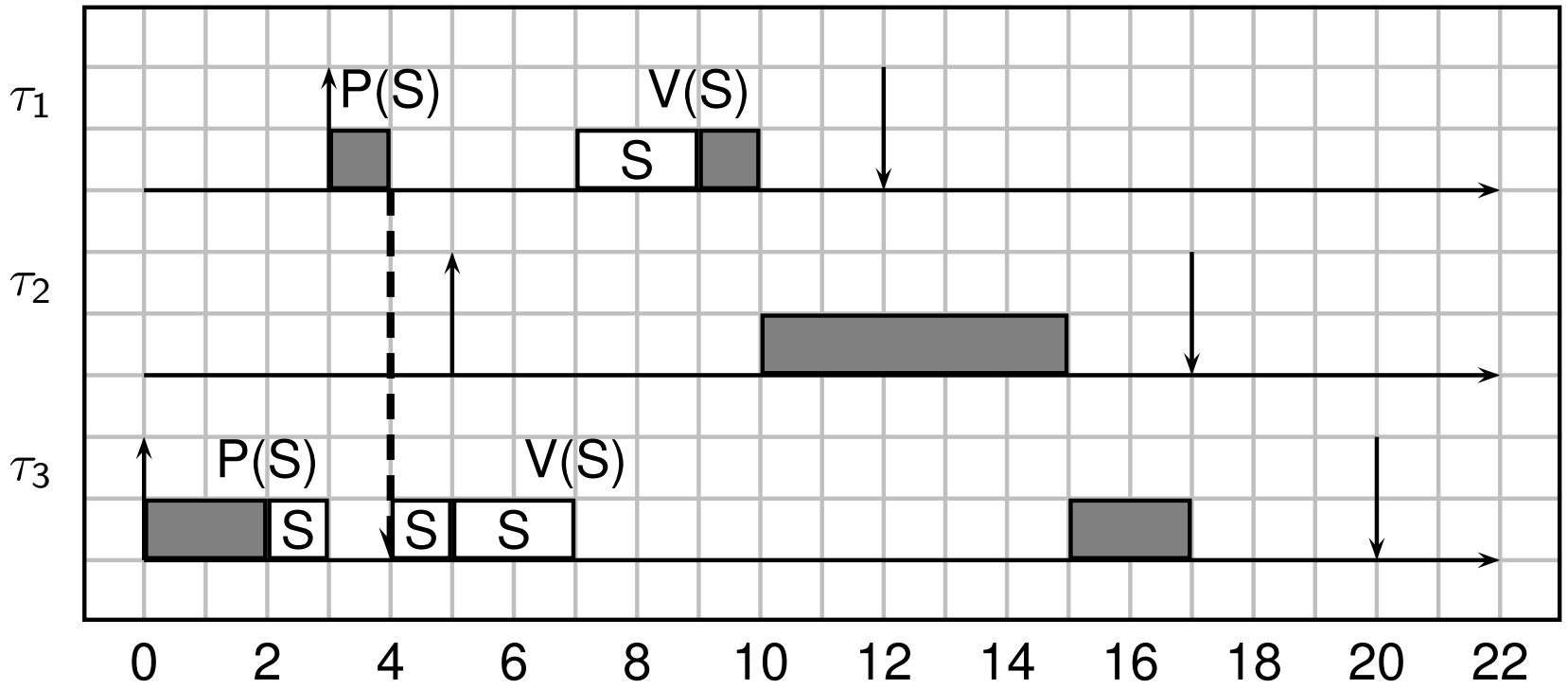
```
Tâche A          Tâche B
  P(1)             P(2)
  P(2)             P(1)
    action A         action B
  V(2)             V(1)
  V(1)             V(2)

Exemple d'inter-blocage:

 A obtient 1
 B obtient 2
 A attend 2 et B attend 1, tout le monde est bloqué !
```

Inversion de priorité

- Une inversion de priorité ce produit quand une tâche T_1 de priorité haute attend un verrou qui est détenu par une tâche T_2 de priorité faible.
- On peut éviter le problème par *héritage de priorité*. On augmente temporairement la priorité de T_2 à celle de T_1 pour qu'elle libère au plus vite la ressource.
- Bug du Mars Pathfinder en 1997 alors que le robot était déjà sur Mars. Reproduit sur simulateur et corrigé à distance, en activant l'héritage de priorité sur les sémaphores.

Exemple: Inversion de priorité

- La tâche T_3 possède un verrou requis par T_1
- La priorité de T_3 est augmentée temporairement, la tâche T_2 devra attendre mais le verrou sera libéré rapidement.



Gestion des événements sporadiques

- Jusqu'à maintenant on a considéré surtout des tâches périodiques
- Comment gérer des événements sporadiques, comme la lecture d'une touche depuis le clavier ?
- Deux méthodes:
  - Par polling (ou scrutation)
  - Par interruption

Gestion par polling

- Une tâche va périodiquement vérifier si une touche à été appuyée.
- Avantages:
  - Simple à mettre en place
- Inconvénients
  - Pour être réactif, la période de scrutation doit être faible
  - Énergivore

Gestion par interruption

- On demande au processeur de lever une intérruption lorsqu'une touche est appuyée
- L'interruption intérromp les tâches en cours, et permet l'exécution de la tâche sporadique.
- Avantages:
  - Économe en énergie, on n'exécute la tâche que quand un évenement est reçu
  - Réactif
- Inconvénients:
  - Moins prédictible et plus complèxe à implémenter
  - Pour ne pas bloquer le système le traitement d'interruption doit être très court, il faut découpler l'enregistrement de l'évenement et le traîtement associé.

Bibliographie

- Cours Systèmes Embarqués Temps Réel, Samuel Tardieu, ENST
- Lecture from Giuseppe Lipari, Scuola Superiore Sant'Anna
- Ordonnancement Temps Réel, présentation de Pierre-Yves Duval du CPPM, à l'École d'informatique temps réel 2002.
- Cours de Benoît Miramond, ENSEA
- What Really Happened on Mars by Glenn Reeves of the JPL Pathfinder team