

Advanced Systems

Ordonnancement/ Ordonnancement temps-réel 2/2

Grégoire Pichon

University of Lyon/ LIP

MIF18 - M1. Février-Juin 2022

Plan

La problématique du temps réel

Ordonnancement Temps-réel

Ordonnancement de tâches périodiques, avec priorité
Ressources partagées, inversion de priorité

Conclusion

Credits

Intro honteusement pompée sur les transparents

<http://beru.univ-brest.fr/~singhoff/supports.html>

Définition

“En informatique temps réel, le comportement correct d'un système dépend, non seulement des résultats logiques des traitements, mais aussi du temps auquel les résultats sont produits.”

Différentes notions de déterminisme

Le **déterminisme** est une notion-clef :

- Déterminisme logique : les mêmes entrées appliquées au système produisent les mêmes résultats.
- Déterminisme temporel : respect des contraintes temporelles (échéances, rythme, ...).

Les contraintes de temps

Important Un système temps réel n'est pas un système "qui va vite" mais un système qui satisfait à des contraintes temporelles.

Quelques ordres de grandeur :

- La milliseconde pour les systèmes radar.
- La seconde pour les systèmes de visualisation humain.
- qq heures : production chimique
- ...

Des contraintes plus ou moins dures

Le besoin en **garantie de service** (niveau de respect des contraintes) peut être différent :

- Systèmes temps réel dur ou critique.
- Systèmes temps réel souple.

Exemple 1 : domaine de l'avionique

Système temps réel critique :

- Contraintes temporelles : temps de réponse, échéance, date d'exécution au plus tôt, cadence, etc.
- Dimensionnement au pire cas et réservation des ressources.
- Utilisation de redondance matérielle et logicielle.
- Matériel et logiciel dédiés. Système fermé, validé a priori.
- Système réparti synchrone : commandes de vol, radars, moteurs, etc.

Exemple 2 : multimédia sur le Web

Système temps réel souple :

- Contraintes temporelles : gigue, délais de bout en bout, temps de réponse. Synchronisations intra et inter-flux.
- Plate-forme généraliste. Non déterminisme temporel à cause du matériel et du logiciel (ex : PC + windows).
- Application interactive.
- Nombre de flots inconnu.
- Débits variables et difficiles à estimer hors ligne.

Autres exemples / domaines d'activité

- Transports (métro, aérospatiale, SIG : systèmes d'info géographique et systèmes de régulation automobile).
- Médias (décodeurs numériques openTV).
- Services téléphoniques (téléphone mobile, auto-commutateur).
- Supervision médicale, écologique.
- Système de production industriel : centrale nucléaire, chaîne de montage, usine chimique.
- Robotique (ex : PathFinder)

Solutions pour les systèmes critiques

- OS minimal, accès direct aux ressources, temps “garantis” .
- Approche langages plus ou moins haut niveau (non abordée cette année)

Plan

La problématique du temps réel

Ordonnement Temps-réel

Ordonnement de tâches périodiques, avec priorité
Ressources partagées, inversion de priorité

Conclusion

Ordonnancement sous Linux

Les soucis des ordonnanceurs généralistes :

- Pas de prise en compte de l'urgence ou de contrainte temporelle.
- Politique souvent opaque.
- **Temps de réponse inconnue.**

Problématique

Respect des contraintes TR : échéances, périodicité, ...

Approches **“bare metal”** :

- Pas d'OS : produire un code décrivant la boucle de réaction.
 - Penser en pire cas.
 - On vérifie que l'implémentation est suffisamment rapide par “Analyse du temps d'exécution pire-cas” .
 - Pour les applis les + critiques.
- ▶ Tend à **sur-dimensionner** dans le cas de systèmes non critiques. Difficile dans le cas où l'on veut du **parallélisme** (ex : moteur).
- ▶ Dans la suite : approches construisant un OS temps-réel.



OS temps réel, problématique

Un OS temps réel disposera d'un ordonnanceur de tâches :

- il gère les priorités, traite le temps de manière ad-hoc en donnant des garanties de temps de traitement des interruptions.
- avec des garanties.

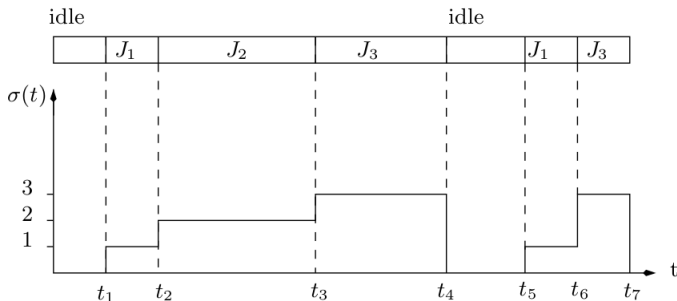
Des garanties de temps réel à condition que soient connus statiquement :

- le nombre de tâches
- les durées/coûts de chaque tâche
- les priorités entre tâches.

Ordonnement temps réel sur monoprocesseur

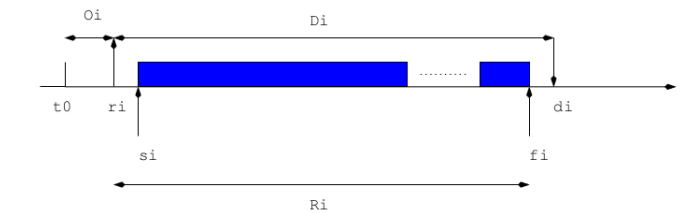
Soit un ensemble de tâches $J = \{J_1, \dots, J_n\}$. Un **ordonnement** est une fonction $\sigma : \mathbb{R}^+ \rightarrow \mathbb{N}$ qui assigne des dates à des tâches :

$$\forall t \in \mathbb{R}^+, \exists t_1, t_2 \text{ tq } t \in [t_1, t_2[\text{ et } \forall t' \in [t_1, t_2[, \sigma(t) = \sigma(t')$$



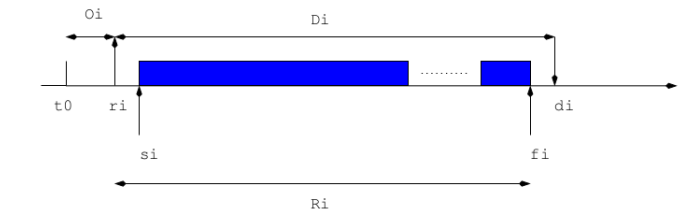
Ordo TR : terminologie, notations 1/3

- Date de début d'exécution :
 - r_i instant où tâche prête à être exécutée
 - O_i décalage par rapport au lancement.
 - (tâche synchrone si $r_i = t_0$).
- Échéance (deadline) :
 - D_i durée à ne pas dépasser pour une exécution
 - C_i borne sup estimée du pire temps d'exécution. (WCET, ou capacité)
 - $d_i = r_i + D_i$ date avant laquelle la tâche doit être terminée (échéance absolue).



Ordo TR : terminologie, notations 2/3

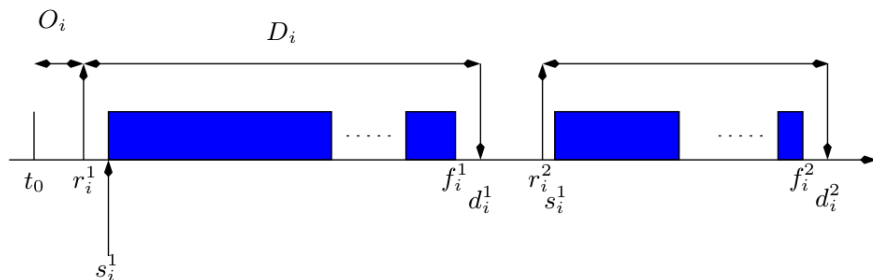
- Dates de début et de fin d'exec : s_i, f_i
- Temps de réponse $R_i = f_i - r_i$.



Ordo TR : terminologie, notations 3/3

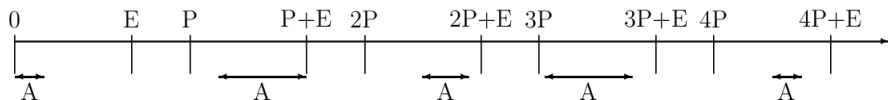
La k -ième instance d'une tâche i est notée t_i^k .

- tâche périodique : l'intervalle entre deux activations est constant, de période T_i , donc $r_i^k = r_i^1 + (k - 1)T_i$.
- tâche sporadique : l'intervalle entre deux activations est sup à une certaine valeur, ie $r_i^k - r_i^{k-1} \geq T$.
- tâche apériodique : aucune contrainte sur les dates d'activation.



Notations : exemple

La tâche A respecte période et échéance :



Rque on peut rajouter certaines contraintes temporelles : bornes sur la durée max d'exec, écart max entre deux événements (synchro image/son), taux de production (flux vidéo)...

Algorithmes en-ligne et hors-ligne

Il y a deux classes d'algorithmes d'ordonnancement :

- **Statique** (off-line) L'ordonnancement est pré-calculé statiquement et peut être stocké dans une table qui détermine qui est activé et quand. Cela s'applique seulement lorsque :
 - Le nombre de tâches est connu statiquement.
 - Les priorités sont fixes et connues statiquement.

▶ Absence de flexibilité.
- **Dynamique** (on-line) L'ordonnancement est calculé dynamiquement : meilleure utilisation (charge) du processeur, et on peut prendre en compte des événements sporadiques et aperiodiques.

Plan

La problématique du temps réel

Ordonnancement Temps-réel

Ordonnancement de tâches périodiques, avec priorité

Ressources partagées, inversion de priorité

Conclusion

Le sous-problème traité

Ordonnancement de tâches **périodiques** :

- r_i date de réveil
- C_i capacité (WCET)
- D_i échéance (deadline)
- T_i période

On se place dans le cas particulier $r_i = 0$ et $D_i = T_i$.

+ monoprocesseur + préemption autorisée

Remarque (test simple) La charge du processeur est $U = \sum_i \frac{C_i}{T_i}$. Donc, si $U > 1$ il n'existe aucun ordonnancement monoprocesseur, préemptif ou non.

Ordonnancement en ligne avec priorités

Durant l'exécution, l'ordonnanceur choisit la tâche à activer de plus haute priorité (choix arbitraire si deux égales, ou alors celui qui minimise les commutations) :

- RM : priorités statiquement calculées.
- EDF : priorités calculées dynamiquement.

RM et EDF : algo

Les deux algorithmes d'ordonnancement les plus connus (et utilisés).

- **Rate Monotonic** (RM) On choisit la tâche de plus forte priorité statique (période minimale). (fixed priority scheduling)

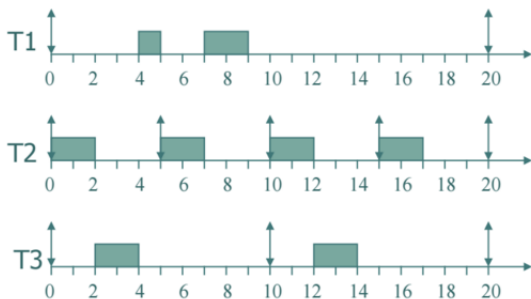
$$select(\ell) = \text{choose } j \in \ell \text{ such that } T(j) = \min_{k \in \ell}(T(k))$$

- **Earliest Deadline First** (EDF) On choisit la tâche dont la deadline est la plus proche (dynamic priority scheduling). Marche aussi pour tâches non périodiques.

$$select(\ell) = \text{choose } j \in \ell \text{ such that } cpt(j) = \min_{k \in \ell}(cpt(k))$$

Rate Monotonic, exemple

Tâche	Période	Échéance	Capacité
T1	20	20	3
T2	5	5	2
T3	10	10	2



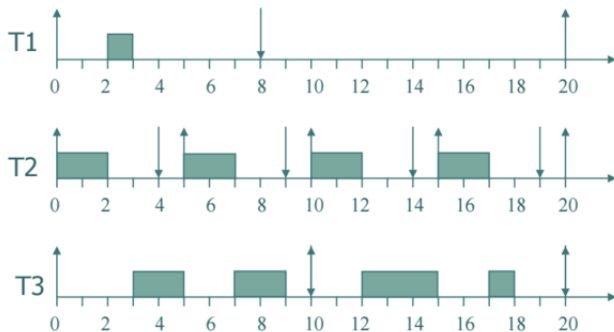
$$Prio(T2) > Prio(T3) > Prio(T1)$$

Exécution cyclique (ppcm des périodes).

Earliest Deadline First, exemple

Tâche	Période	Échéance	Capacité
T1	20	8	1
T2	5	4	2
T3	10	10	4

2 préemptions à $t = 5$ et 15



Notion de faisabilité d'un (algo) d'ordo.

Il existe des tests simples permettant de savoir si un ensemble de tâches est ordonnançable (pour un algo donné)

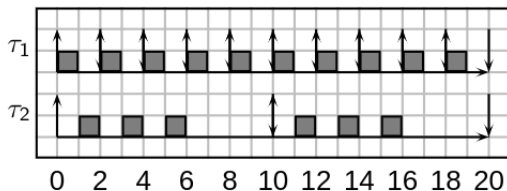
► test d'ordonnançabilité.

Critères suffisants (pour le cas préemptif) :

- Pour RM (Liu and Layland, 1973) n tâches indep :
 $U \leq n(2^{1/n} - 1)$. La limite de cette quantité décroissante est .69, donc si la charge est inférieure à 69%, le système admet un ordo RM (quel que soit n).
- pour RM + tâches harmoniques : $U \leq 1$ est CNS.
- Pour EDF : $U \leq 1$.

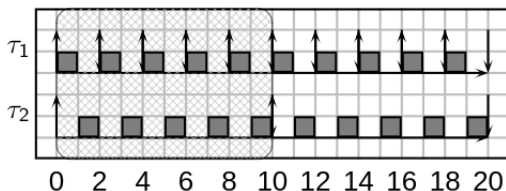
Analyses et exemples pour RM 1/3

- 2 tâches $T_1 = 2s$, $C_1 = 1s$, $T_2 = 10s$, $C_2 = 3s$.
- Utilisation : $1/2 + 3/10 = 80\% < 2 * (2^{1/2} - 1) = 83\%$ donc ordonnançable !
- Système harmonique donc on aurait pu utiliser $U \leq 1$.



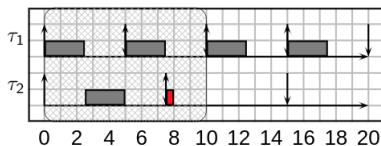
Analyses et exemples pour RM 2/3

- 2 tâches $T_1 = 2s$, $C_1 = 1s$, $T_2 = 10s$, $C_2 = 5s$.
- Utilisation : 100% et harmonique, donc OK !

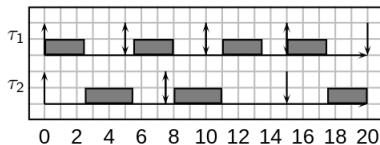


Analyses et exemples pour RM 3/3

- 2 tâches $T_1 = 5s$, $C_1 = 2,5s$, $T_2 = 7,5s$, $C_2 = 3s$.
- Utilisation : 90% on ne peut pas conclure, ici NOK.



Mais si on s'affranchit de RM :

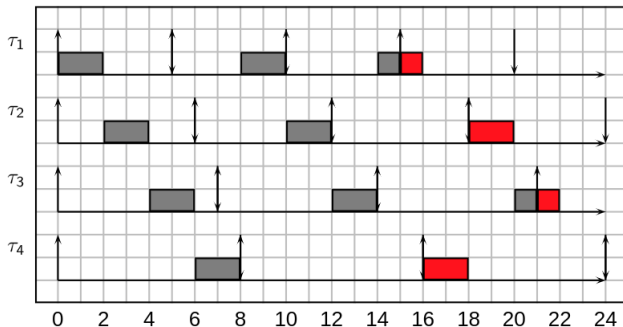


Conditions, encore

- CNS Pour un algorithme donné faire une simulation pire-cas sur une hyper-période $T = ppcm(T_i)$.
- CNS pour EDF $U \leq 1$.
- Plein d'autres dans la littérature.

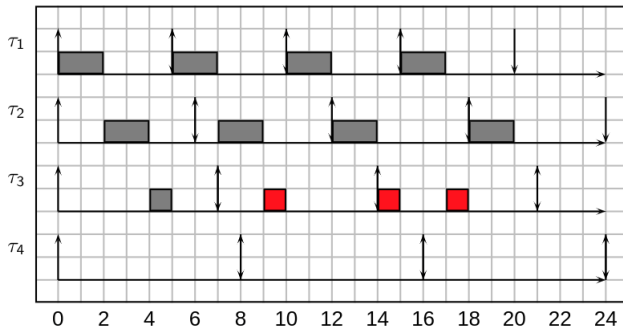
Comparaison RM/EDF 1/3 - effet domino

EDF : à tout moment, le travail prioritaire est celui dont l'échéance est la plus courte. Mais comportement mauvais en cas de surcharge, cela peut provoquer une avalanche d'échéances manquées :



RM / EDF 2/2

RM a un comportement meilleur, le souci affecte les tâches les moins prioritaires, mais certaines tâches peuvent ne jamais être exécutées :



Plan

La problématique du temps réel

Ordonnement Temps-réel

Ordonnement de tâches périodiques, avec priorité
Ressources partagées, inversion de priorité

Conclusion

Le partage des ressources, le début des ennuis

On a considéré le cas de tâches sans synchronisation entre elles et donc sans accès à une ressource partagée nécessitant un mécanisme de verrou.

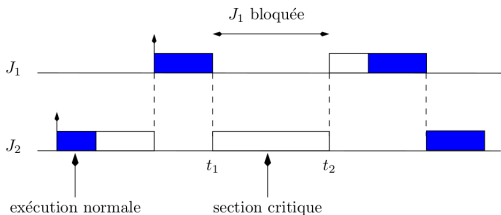
Considérons deux tâches J_1 et J_2 accédant à une ressource partagée R_k devant être accédée en exclusion mutuelle en utilisant un sémaphore S_k :

- $wait(S_k)$: demande d'accès à la ressource partagée
- $signal(S_k)$: libération de la ressource

$J_1 = \dots$	$J_2 = \dots$
$wait(S_k);$	$wait(S_k);$
\dots	\dots
$R_k;$	$R_k;$
$signal(S_k);$	$signal(S_k);$
\dots	\dots

Inversion de priorité

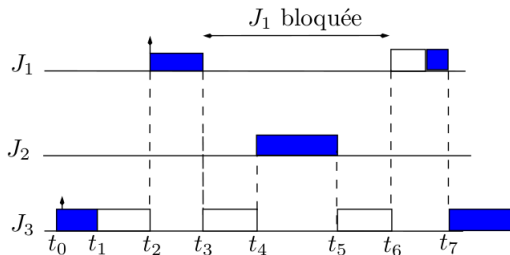
- J_1 a une priorité plus forte que J_2 .
- J_2 est activée puis entre en section critique (bloque le sémaphore).
- J_1 arrive. Parce que sa priorité est plus forte, préempte J_2 .
- A $t = t_1$, J_1 est bloquée et donc J_2 reprend.
- J_1 doit attendre jusqu'à $t = t_2$, lorsque J_2 quitte la section critique.



► Le temps d'attente maximum pour J_1 est égale à la durée de la section critique de J_2 .

Cas plus grave : Mars Pathfinder, 1997¹

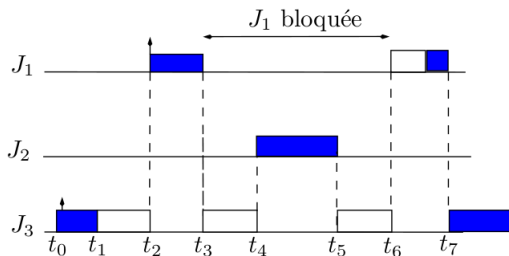
Le temps d'attente ne peut pas toujours être borné par la durée de la section critique exécutée par la tâche de priorité la plus faible.



- J_1 arrive au temps t_2 et préempte J_3 durant sa section critique.
- A l'instant t_3 , J_1 tente d'utiliser la ressource mais est bloquée sur une sémaphore S .
- Donc J_3 continue son exécution en section critique.
- Si J_2 arrive à $t = t_4$, il la préempte J_3 (car il a une priorité plus forte).
- Cela augmente la durée pendant laquelle J_1 est bloquée.

1. http://research.microsoft.com/en-us/um/people/mbj/mars_pathfinder/mars_pathfinder.html

Inversion 3 : toujours Pathfinder



- Le temps de blocage maximum pour J_1 ne dépend plus seulement du temps de la section critique de J_3 ; il dépend du temps d'exécution maximum de J_2 .
 - Une inversion de priorité a lieu dans l'intervalle $[t_3, t_6]$
- Cette durée n'est pas bornée statiquement puisque n'importe quelle tâche de priorité intermédiaire préemptant J_3 bloque indirectement J_1 .

Solutions à l'inversion de priorité

Plusieurs possibilités :

- Interdire la préemption durant l'exécution d'une section critique : réaliste à condition que celles-ci soient courtes.
- **Héritage de priorité** : modifier la priorité d'une tâche qui cause un blocage. Quand une tâche J_i bloque une ou plusieurs tâches de plus forte priorité, elle hérite temporairement de la priorité la plus forte de la tâche bloquée.

Héritage de priorité, mise en œuvre

- Lorsque J_i essaie d'entrer en section critique $z_{i,j}$ et que la ressource $R_{i,j}$ est utilisée par une tâche de priorité plus faible, J_i reste bloqué. Sinon, il entre en section critique.
- Lorsque J_i est bloqué sur un sémaphore, il transmet sa priorité à la tâche J_k qui tient le sémaphore. Donc, J_k continue et exécute la suite de sa section critique avec la priorité $p_k = p_i$. J_k hérite de la priorité de J_i .
- Lorsque J_k sort de sa section critique, il débloquent le sémaphore et la tâche de plus forte priorité bloquée sur le sémaphore est libérée. La nouvelle priorité de J_k est modifiée : si aucune autre tâche n'est bloquée par J_k , p_k est réinitialisé à sa priorité initiale, sinon, il hérite de la priorité la plus haute des tâches bloquées par J_k .
- L'héritage de priorité est transitif : si une tâche J_3 bloque une tâche J_2 et que J_2 bloque J_1 , alors J_3 hérite de la priorité J_1 .

Plan

La problématique du temps réel

Ordonnement Temps-réel

Ordonnement de tâches périodiques, avec priorité
Ressources partagées, inversion de priorité

Conclusion

Take out message

- Des algorithmes spécifiques au temps-réel.
- Des algorithmes classiques à connaître et à savoir mettre en oeuvre. (cf “services POSIX pour l’ordonnancement”)

Le cas multiprocesseur, est largement plus complexe.