Systèmes avancés

Ordonnancement

Grégoire Pichon

Lyon 1 / Laboratoire de l'Informatique du Parallélisme

MIF18 - M1. 2022/2023

Plan

Lyon 1

Un ordonnanceur, pourquoi ?

C'est l'ordonnanceur qui gère l'allocation et la suspension des tâches. L'ordonnanceur peut être :

- "hors-ligne" ou "en ligne".
- Préemptif ou non-préemptif.

Ordonnanceurs classiques :

- PAPS (FIFO) Premier arrivé, premier servi.
- Tourniquet (Round Robin) à tour de rôle avec un *quantum* de temps.
- Priorité : plus prioritaire d'abord.

Les priorités Unix

Priorités $\neq$ respect des échéances :

- Les priorités UNIX, laissent la main aux tâches moins prioritaires afin qu'elles avancent un peu.
- Si les tâches faiblement prioritaires n'avançaient jamais cela pourrait introduire des *dead-lock*.

▶ Mais les priorités ne permettent pas forcément le respect des échéances.

Algo d'ordo PAPS (FIFO)

Principe : premier arrivé premier servi.

Algo PAPS, durée : A=4, B=6, C=2														
Temps	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Qui ?	A	A	A	A	B	B	B	B	B	B	C	C		D
Démarrage	A	B	C											D

Algo d'ordo Tourniquet (Round Robin)

Principe : donner à manger à tout le monde "en tournant".

Algo tourniquet, durée : A=4, B=6, C=2														
Temps	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Qui ?	A	B	C	A	B	C	A	B	A	B	B	B		D
Démarrage	A	B	C											D

Algo d'ordo PAPS + prio

Principe : idem PAPS, mais en mettant des priorités (choix du processus à ordonnancer si deux sont ordonnançables).

PAPS + Priorités : A=+, B=++, C=++														
Temps	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Qui ?	A	B	B	B	B	B	B	C	C	A	A	A		D
Démarrage	A	B	C											D

Algo d'ordo Tourniquet + prio

Principe : idem tourniquet + priorités.

Tourniquet + Priorités : A=+, B=++, C=++														
Temps	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Qui ?	A	B	C	B	C	B	B	B	B	A	A	A		D
Démarrage	A	B	C											D

Inconvénients des ordonnanceurs classiques

- Le processeur est toujours utilisé et ne peut se mettre en veille. Le passage à l'état de veille du processeur est souvent long.
- Cela fait perdre du temps CPU. (très gênant quand on fait tourner des machines virtuelles.)
- Le *quantum* de temps doit plutôt être choisi en fonction de la charge de la machine.

Solution : au lieu d'être lancé périodiquement, l'ordonnanceur indique dans combien de temps il doit être réveillé. Évidemment, il peut être réveillé par une tâche prioritaire qui devient prête.

Et si on veut plus de garanties ?

Les systèmes généralistes ont quelques soucis :

- Ils ne sont pas déterministes : matériel / logiciel.
- L'ordonnanceur temps partagé n'offre aucune garantie.
- (linux) le noyau (les processus noyaux) est non préemptif (donc si un autre processus plus prioritaire a besoin du processeur ça foire.)

▶POSIX 1003.1b sous Linux

Plan

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
- ...

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)

Plan

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 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.

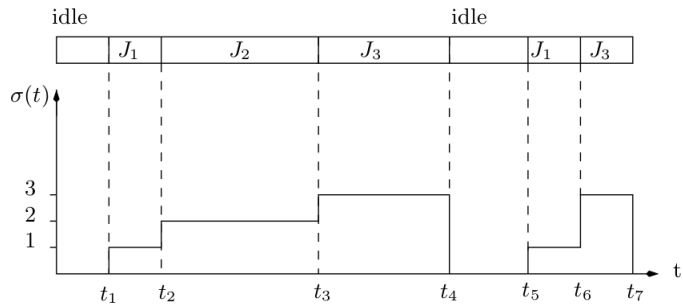
Des garanties de temps réel à condition que soient connus statiquement :

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

Ordonnancement temps réel sur monoprocesseur

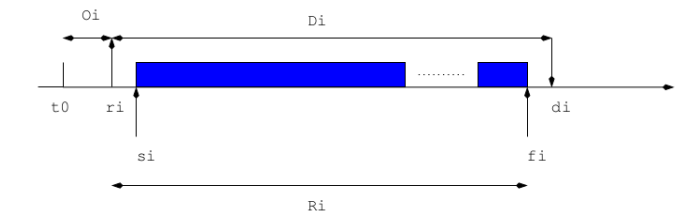
Soit un ensemble de tâches $J = \{J_1, \dots J_n\}$. Un **ordonnancement** est une fonction $\sigma : \mathbb{R}^+ \to \mathbb{N}$ qui assigne des dates à des tâches :

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



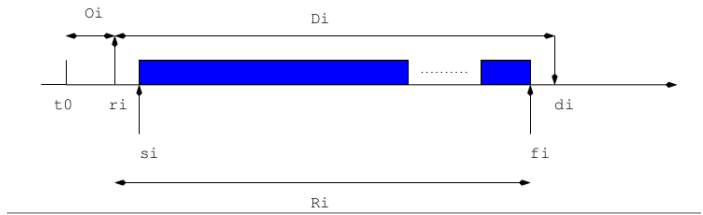
Ordo TR : terminologie, notations 1/2

- 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).



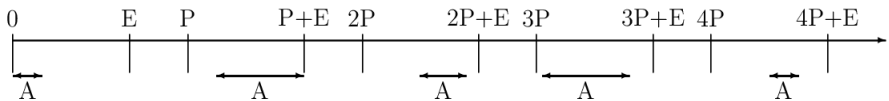
Ordo TR : terminologie, notations 2/2

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



Notations : exemple

La tâche A respecte période et échéance :



Remarque 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 prioriétés 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 apériodiques.

Plan

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)

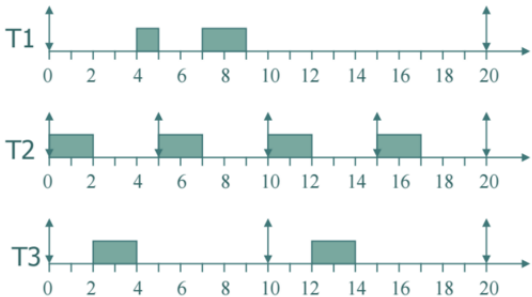
$$select(\ell) = \text{choose } j \in \ell \text{ such that } T(j) = min_{k \in \ell}(T(k))$$

- **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



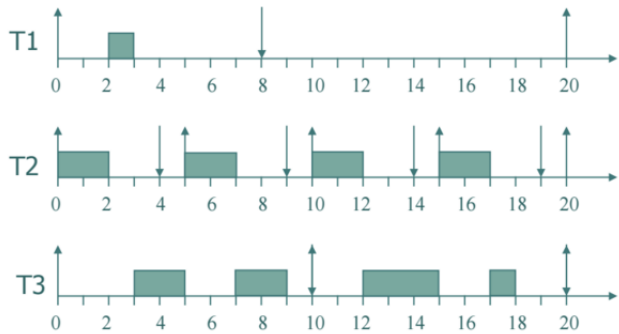
$$Prio(T2) > Prio(T3) > Prio(T1)$$

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.

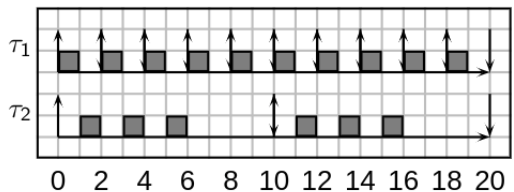
Il existe des tests simples permettant de savoir si un ensemble de tâches est ordonnançable (pour un algo donné)
▶**test d'ordonnançabilité**.

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 (les grandes périodes sont le produit des petites périodes) : $U \leq 1$ est CNS.
- Pour EDF : $U \leq 1$.

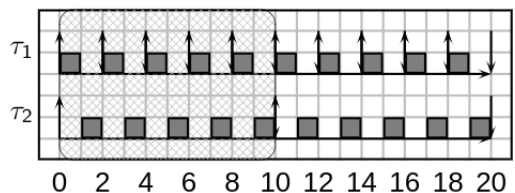
Analyses et exemples pour RM 1/3

- 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 $U \leq 1$.



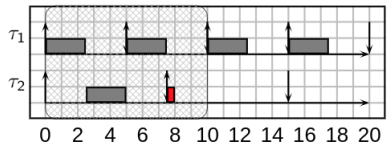
Analyses et exemples pour RM 2/3

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

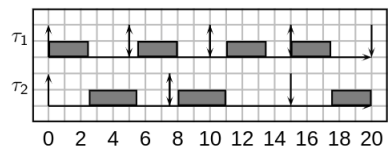


Analyses et exemples pour RM 3/3

- 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 :

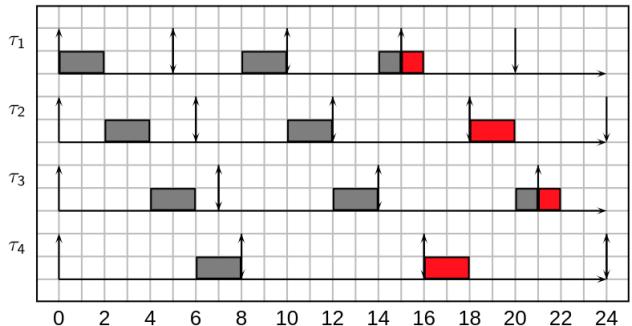


Conditions, encore

- 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.

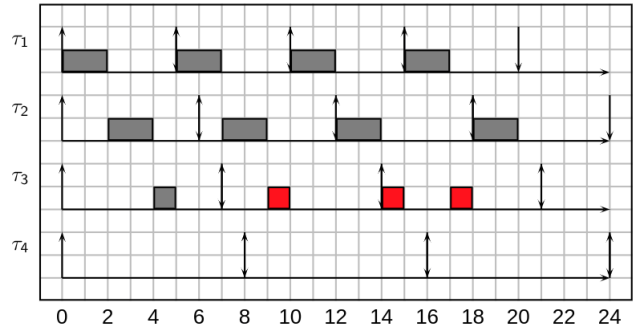
Comparaison RM/EDF 1/2 - effet domino

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

Le partage des ressources, le début des ennuis

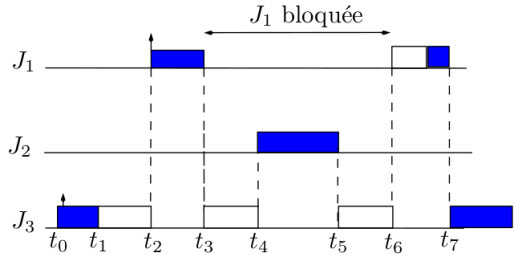
On a considéré le cas de tâches sans synchronisation entre elles et donc sans accès à une ressource partagée nécessitant un mécanisme de verrou.
Considérons deux tâches J_1 et J_2 accédant à une ressource partagée R_k devant être accédée en exclusion mutuelle en utilisant un sémaphore S_k :

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

$$
\begin{array}{lll}
J_1 = & \ldots & J_2 = \quad \ldots \\
& wait(S_k); & \quad wait(S_k); \\
& \ldots & \quad \ldots \\
& R_k; & \quad R_k; \\
& signal(S_k); & \quad signal(S_k); \\
& \ldots & \quad \ldots
\end{array}
$$

Cas problématique : Mars Pathfinder, 1997 [1]

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 resource 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 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`

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 bloquage. 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.

Plan

Conclusion

A retenir

- Les techniques d'ordonnancement pour les systèmes généralistes
- La problématique du temps-réel
- Les différents algorithmes pour l'ordonnancement temps-réél (RM/EDF) et les CN, CS