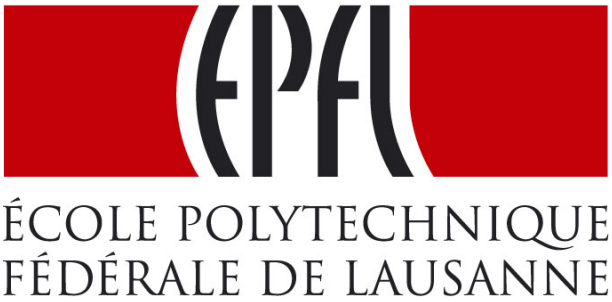
Ecole Polytechnique Fédérale de Lausanne

Notes de Cours en

Informatique du temps réel



Olivier Cloux

Automne 2016

Table des Matières

Les notes sont mélangées entre français et anglais, car les slides sont en anglais mais le discours est énoncé en français

1 Introduction

Un système temps-réel peut être un système transformational ou reactive. Un système transformational a une durée de vie limitée, prend un input et sort un output, et basta. Un système réactif peut être interactif : en dialogue constant avec un ou des utilisateurs. Rien n'est fait sans requête, et le temps de réponse n'est pas trop important. Un système reactive peut être aussi real-time. Il est connecté au temps réel, il doit réagir dans une marge temporelle parfois précise (par exemple airbag). Ces systèmes ne s'arrêtent en général jamais, et ne devraient jamais mettre l'utilisateur ou le processus en danger (par exemple, un robot pour mettre les chocolats dans une boite, doit réagir exactement dans le temps, sinon il rate sa tâche).

Définition : Temps de réponse.
L'intervalle entre l'input change et l'instant auquel l'output est sensiblement modifié

Définition : Système de temps réel.
Un système informatique donc la correctness du calcul

Hard real-time en cas de violation d'une contrainte temporelle, perte complète de la fonctionnalité.

Soft real-time la fonctionnalité n'est pas perdue, mais la qualité en pâti.

2 Ordonnancement (scheduling)

Les contraintes peuvent être douces ou dures. Les opérations peuvent venir en parallèle, il convient alors de déterminer quelle tâche doit être exécutée à quel moment afin de remplir les conditions (exemple du lièvre et de la tortue)

2.1 Parenthèse sur la théorie de la complexité

Comparer la complexité de différents algorithmes. Pour prouver qu'un problème est (trop) compliqué, on prouve qu'il est aussi compliqué qu'un autre, par exemple on va comparer (ou associer) un problème au problème du Knapsack.

Rappel : P : solution peut être trouvée en temps polynomial ; NP : solution peut être démontrée comme correcte en temps polynomial ; NP-Hard : chaque problème de NP en est un cas spécial ; NP-Complete : intersection de NP et NP-Hard

2.2 Taxonomie de la tâche

Périodique : T_i est le temps entre chaque nouvelle tâche (toujours le même entre les tâches). C_i est la somme des temps de calcul des tâches entre chaque T_i. D_i est l'échéance pour la tâche (avec $D_i \leq T_i$). r_i est le release time, ou le décalage à l'origine.
Sporadique : les tâches sont déclenchées par un événement. On peut garantir qu'entre ces événements sont séparés par T_i, un temps minimum (peut être plus grand). Il est possible que le temps minimum (T_i) soit nul. Auquel cas, on parlera de tâches **apériodique**. Attention dans la littérature, il arrive que le terme apériodique soit utilisé pour désigner un système sporadique.
Cyclique : C_i est toujours le temps de calcul de la tâche, et nous créons T_i^{av}, le temps moyen entre chaque tâche, et T_i^{max}, le temps maximum entre chaque tâche.
Permanente : Tous les T_i secondes, on donne $C_i\%$ du processeur

Une tâche est composée d'un flux (infini ou non) de tâches (c'est l'enveloppe) ; chaque tâche a des paramètres (C_i, $D_i, ...$), et chaque paramètre prend aussi des tâches (la deadline absolue de la tâche i : d_i

Mais ces tâches ne sont pas complètement indépendantes. Il y a une dimension de précédence (quelle tâche vient avant moi), la synchronisation mutuelle, l'exclusion, le partage de ressources (et des combinaisons de tout ça).

Avant de commencer l'ordonnancement ou le calcul, il est important de faire une analyse de faisabilité : est-ce que, ayant tous ces paramètres, il m'est

possible d'exécuter la tâche ? On commence par faire une configuration, pour arranger ces tâches en fonction de leurs paramètres. On fait une distribution, préemptive ou non.

Un système peut être déterministe (les séquences et leurs occurrences sont connues d'avance) ou prévisible (on peut prédire que le système se comporte en relation à ses propriétés). C'est ce dernier qui est important, le déterminisme ne sert à rien.

2.3 Algorithmes d'ordonnancement

Les algorithmes d'ordonnancement sont nombreux. Ils ont en général des priorités fixes, et sont directement basés sur les caractéristiques de la tâche. Tout est basé sur la notion de *worst-case execution time* C. Ce temps est une *hypothèse*, difficile à évaluer et souvent beaucoup trop pessimiste.

2.3.1 Rate Monotonic (RM)

On assume que les tâches sont périodiques, que la fin de la deadline marque la fin de la période, les tâches sont préemptives, ne peuvent se bloquer ou se suspendre elles-mêmes, et que le temps d'exécution C_i est connu et fixe.

Dans cette simulation, plus la période est courte (T est petit), plus la priorité est haute.

Dans les transparents : les tâches A,B,C se suivent, jusqu'à 10 : A prend la priorité et interromps C. Une fois A fini on reprend la tâche C, qui est de nouveau interrompue par B, etc.

Pour assurer que ça fonctionne, il faut que le processeur ne soit pas surchargé, donc (NÉCESSAIRE) que

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

Et, suffisamment, que (**SEULEMENT POUR RM À PRIORITÉ FIXE**)

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

Une analyse plus poussée introduit le temps de réponse du pire cas :

$$R_i = C_i + \sum_{\forall j \in hp(i)} \frac{R_i}{T_j} C_j$$

2.3.2 Deadline Monotonic

Ici, la condition nécessaire (charge du proco) est toujours vraie. Mais la seconde est fausse.

Optimalité de RM et DM

Sous certaines conditions (dont l'absence de jitter), RM et DM ont été démontrés optimaux. Par exemple, RM est optimal pour $D = T$, DM l'est pour des échéances constantes et $D < T$, mais pas autrement ! Dans les autres cas, il est prouvé qu'il existe un assignment des priorités qui est optimal.

2.3.3 Earlier Deadline First

En regardant la slide 27, nous voyons en 10 que la seconde instance de A a une échéance courte (10 secondes), mais éloignée (en 20). Alors que l'instance de B interrompue a une échéance plus longue (15) mais elle est interrompue alors que son échéance est en 15 ! Il serait plus logique de la finir d'abord avant de lancer une tâche dont l'échéance est plus courte (absolument) mais plus loin (relativement).

Pour cet algorithme nous avons :

- Les échéances des tâches sont arbitraires (pas forcément constantes).
- Les tâches sont préemptives et indépendantes
- Les tâches ne peuvent se bloquer ou suspendre elles-même
- Le pire cas d'exécution C_i est connu.

Cet algorithme est optimal car, s'il existe un algorithme qui a réussi un ordonnancement, alors EDF peut aussi le faire (bien entendu sous les conditions de EDF). Voir la preuve sur les transparents.

Les conditions nécessaires et suffisantes sont plus intéressantes : la condition nécessaire (1) est toujours valable, mais de plus, si $\forall i D_i \geq T_i$, alors elle devient aussi suffisante

Cet algorithme est bon, mais souffres d'indétermination en cas de surcharge.

2.4 Mutual exclusion

Compléter

Pour éviter un **blocage** (deadlock), une tâche ne doit pas être autorisée à démarrer à moins que les ressources disponibles à cet instant soient suffisantes pour ses besoins
De même, pour éviter les **inversions de priorité**, une tâche n'est pas autorisée à démarrer à moins que les ressources disponibles ne soient suffisantes pour ses besoins et ceux de toutes les tâches qui pourrait la préempter

3 Évaluation des performances