Systèmes temps réel

Fabien Calcado

Email: fabien.calcado@gmail.com

Sommaire

- **Ordonnancement monoprocesseur**
  - Définitions et stratégies d'ordonnancement pour certains modèles de tâches
- **Ordonnancement multiprocesseur**
  - A partir d'un graphe de dépendance de tâches

Ordonnancement

- **Introduction**
  - L'application est un ensemble de n tâches qu'on appelle système de tâches
    - Départ simultané (même 1er date de réveil) ou échelonné
  - Terminologie (rappel)
    - L'**ordonnancement** est l'organisation de l'exécution des tâches sur le(s) CPU(s) du système
      - » *Séquencement , entrelacement…*
    - La **politique d'ordonnancement** est la règle d'organisation de l'exécution des tâches sur les CPU(s) du système
      - » *Citations de… ? :*
        - » "All we have to decide is what to do with the time that is given to us."
        - » "… n'arrive jamais en retard , ni en avance d'ailleurs. Il arrive au moment précis où on l'attend."

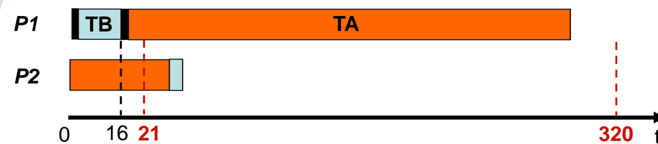
Ordonnancement

- **Introduction**
  - L'application est un ensemble de n tâches qu'on appelle système de tâches
    - Départ simultané (même 1er date de réveil) ou échelonné
  - Terminologie (rappel)
    - L'**ordonnancement** est l'organisation de l'exécution des tâches sur le(s) CPU(s) du système
      - » *Séquencement , entrelacement…*
    - La **politique d'ordonnancement** est la règle d'organisation de l'exécution des tâches sur les CPU(s) du système
      - » *Citations de… Gandalf :*
        - » "All we have to decide is what to do with the time that is given to us."
        - » "Un magicien n'arrive jamais en retard , ni en avance d'ailleurs. Il arrive au moment précis où on l'attend."

Programmation multitâche et ordonnancement – EFREI 2013

- **Introduction**
  - Deux travaux à exécuter
    - Travail A : Durée = 270 , Échéance = 320
    - Travail B : Durée = 15 , Échéance = 21
  - Deux ressources différentes
    - P1 : vitesse = 1 , durée commutation = 1 , priorité = à l'échéance
    - P2 : vitesse = 10 , durée commutation = 0 , priorité = au premier

TA arrive avant TB à t = 0

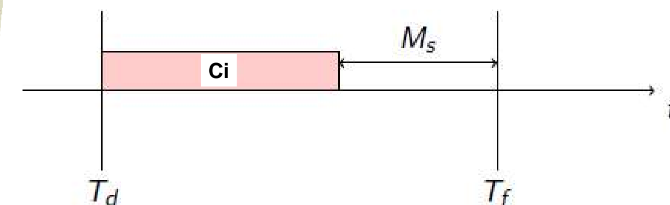


- **Durée d'exécution (noté C)**
  - Temps nécessaire à un processeur pour exécuter le code d'un travail sans interruption
    - La durée d'exécution dépend de la vitesse du processeur
    - La durée d'exécution est théorique (non constante en pratique)
      - *Durée maximale d'exécution (worst case execution time ou **WCET**)*
      - *Durée minimale d'exécution (best case execution time)*

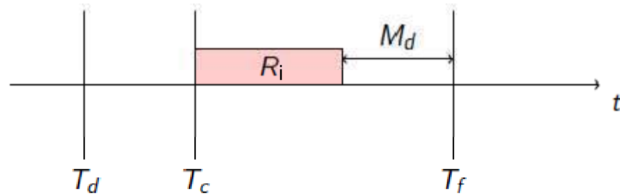
  ➔ En général, le temps d'exécution d'un travail correspond au WCET
  - Mode d'évaluation de la durée d'exécution
    - Mesure en-ligne, analyse hors-ligne
  - Difficultés
    - Complexité et multitude des chemins d'exécution
    - Complexité des processeurs

- **Attributs temporels d'un travail (job)**
  - Date de début au plus tôt (demande / activation) : Td
    - Instant où un travail devient prêt (arrival time ou release time)
  - Date de fin au plus tard (**échéance**) : Tf
    - Date avant laquelle l'exécution d'un travail doit être terminé (deadline)
  - Durée d'exécution d'un travail « i » (besoin) : Ci
- **Paramètres dérivés**
  - Délai critique (fenêtre temporelle)
    - Délai maximum acceptable pour l'exécution d'une tâche
  - Date de début au plus tard
  - Durée jusqu'à la date de début au plus tard (laxité)
  - Date de fin au plus tôt
  - Date de début et de fin d'exécution du travail
    - Préemptions possibles

- **Cas non-préemptif**
  - Marge statique : Ms = (Tf - Td) - Ci
    - Ms ≥ 0
    - Si Ms = 0, pas de choix



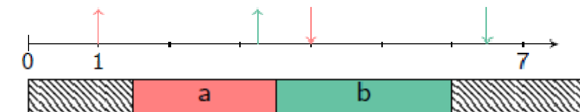
- **Cas préemptif**
  - Marge dynamique ou laxité : Md = (Tf – Tc) – Ri
    - Tc = instant courant
    - Ri = durée d'exécution restante du travail i
    - Md n'évolue pas pour le travail actif
    - Md évolue dynamiquement pour les travaux non actifs



- **Notion de tâche**
  - Une tâche est une entité qui crée des travaux (job release)
    - La tâche s'active et doit exécuter un travail (job)
  - Trois classes majeures
    - Périodique : Crée un travail toutes les périodes T (> 0)
      - *Échéance implicite : Échéance = période*
      - *Échéance contrainte : Échéance ≤ période*
    - Sporadiques : créations de travaux séparées par un intervalle de temps minimal (connue)
    - Apériodique : Modèle le plus général, les créations de travaux doivent quand même être connues

- **Période d'étude (cas périodique)**
  - L'exécution dure indéfiniment mais le comportement de la configuration est périodique
  - **Période d'étude** est l'intervalle [0 , PPCM (Pi)]
    - Pi = période de toutes les tâches
    - Uniquement pour le cas périodique à départ simultané

- **Plan d'ordonnancement**
  - C'est une méthode de prévision pour l'allocation des ressources (phase de conception)
  - Un ordonnancement est dit **optimal** si toutes les contraintes temporelles sont respectées



Ordonnancement

- **Plan d'ordonnancement**
  - Il y a **surcharge** lorsque le volume des tâches à ordonnancer est tel que tous les ordonnancements conduisent au non respect d'au moins une tâche
    - Il n'existe pas d'ordonnancement optimal
  - Un **algorithme** d'ordonnancement est dit optimal, si pour une classe de problème donnée, il produit des ordonnancements optimaux

Ordonnancement

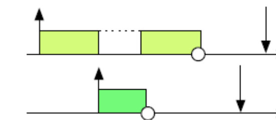
- **Types d'algorithmes d'ordonnancement**
  - Statique
    - Le plan d'ordonnancement est décidé avant exécution (hors-ligne)
      - » *La séquence d'ordonnancement est pré-calculée sur la base de toutes les caractéristiques temporelles des tâches*
  - Dynamique
    - Le plan d'ordonnancement varie pendant l'exécution (en-ligne)
      - » *Les décisions d'ordonnancement sont prises au fil de l'exécution de l'application par l'ordonnanceur*
    - Non préemptifs ou préemptifs
    - Priorité fixe (statique ou dynamique)

Ordonnancement

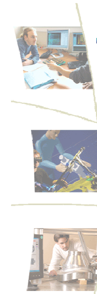
- **Hors ligne : avantages / inconvénients**
  - Nécessite la connaissance a priori du système et de ses caractéristiques temporelles (dates de réveil figés)
    - Manque de flexibilité mais grande prédictibilité (flot d'instruction unique)
  - Adapté à un modèle périodique
    - Calcul de l'ordonnancement sur un cycle (ppcm des périodes des tâches) ordonnanceur cyclique (programmation en boucle)
  - Simplicité de l'ordonnanceur
    - faibles overheads d'exécution (surcoût d'exécution)
  - Efficacité calculatoire non demandée pour le calcul de la séquence d'ordonnancement
    - Mise en œuvre d'algorithmes optimaux + des contraintes additionnelles peuvent être prises en compte (précédence entre tâches, synchronisation entre tâches, réveils à des dates arbitraires, etc.)
  - Régularité d'exécution
    - Rigide (ne peut s'adapter à l'environnement)

Ordonnancement

- **En ligne : avantages / inconvénients**
  - Flexible
    - Adapté aux systèmes dynamiques et évolutifs (capable de prendre une décision à un instant t)
  - Efficacité calculatoire requise
    - Politiques d'ordonnancement simples
  - Difficulté à prendre en compte des contraintes variées
    - Analyse délicate et souvent pessimiste
    - A priori moins sûr → besoin d'une preuve
  - Non oisif
    - Le processeur est toujours utilisé s'il y a au moins une tâche prête

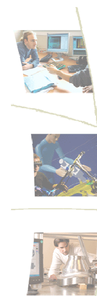


- **Exemple hors ligne : programmation en boucle**
  - T1 : C=10 , Période=30
  - T2 : C=8 , Période=30
  - T3 : C=6 , Période=60
  - T4 : C=10 , Période=60
  - T5 : C=4 , Période=120



- **Modèles – hypothèses**
  - Sur les tâches
    - Types : séquentielles, parallèles
    - Relations : mutuellement dépendantes, indépendantes
    - Valeurs : identiques / différentes, constantes / dépendantes de l'instant de terminaison
    - Abandon : si violation de contraintes exigée, autorisée, interdite
  - Sur les ressources
    - Multiplicité : une ou plusieurs (équivalentes ou non)
    - Mode d'accès : centralisé ou distribué (ressources mémoires)
    - Réquisition (préemption)
      - » *Toujours possible (avec ou sans perte)*
      - » *Possible par moment*
      - » *Impossible*

- **Modèles – hypothèses**
  - Sur les lois évènementielles
    - Connues totalement ou partiellement (non temps-réel)
    - Prédéterminées ou statistiques
  - Sur les lois d'activations des tâches
    - Instants d'activations des tâches
    - Instants d'accès aux ressources
    - Instants délimitant les fenêtres de réquisitions possibles / impossibles



- **Modèles – hypothèses**
  - Un énoncé de problème = un choix parmi toutes ces hypothèses
    - Présentation de diverses classes de problème avec des solutions connues, souvent optimales (cadre hypothèse / contraintes)
    - Dans la suite :
      - » *H1 = hypothèses issues de l'énoncé du problème : identifie la classe du problème*
      - » *H2 = condition sur les données quantitatives du problème*

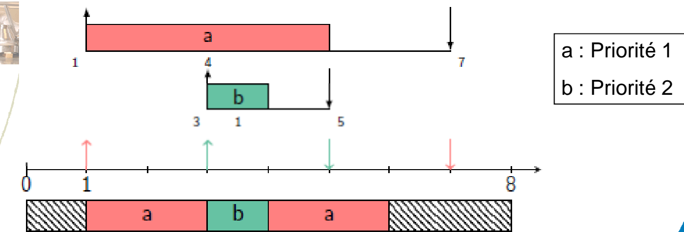
        ➔ Hypothèses de faisabilité à vérifier

Algorithmes d'ordonnancement

- **Algorithmes connus**
  - Algorithmes de bases
    - Les autres sont pour la plupart une combinaison d'entre eux
  - Statiques
    - FP (priorité fixe)
    - RMS
  - Dynamiques
    - FCFS (Premier arrivé / premier Servi)
    - RR (tourniquet)
    - EDF (priorité à l'échéance la plus proche)
    - LLF (priorité à la plus petite laxité)

Algorithmes d'ordonnancement statique

- **Ordonnancement préemptif à priorité fixe (FP)**
  - Les tâches ont une priorité fixe
  - La tâche de plus haute priorité prend la main
    - Doit être en mesure de s'exécuter
    - Préemption possible
  - Exemple



Algorithmes d'ordonnancement statique

- **Ordonnancement préemptif à priorité fixe (FP)**
  - Très proche du HW (E/S asynchrones)
    - Ordonnancement peut être fait au niveau matériel
  - Ponctualité pour une tâche
    - Pour les autres ?
  - Peu de sûreté
  - Peu de vivacité
    - Si beaucoup de tâches de haute priorité risque de famine
    - Amélioration des défauts
      - » *Changement possible des priorités en ligne (Unix / Windows)*
  - Les analyses restent compliquées pour exercer des contrôles sur les temps de traitements
    - => système classique car pas de notion d'urgence

Algorithmes d'ordonnancement statique

- **RMS (Rate Monotonic Scheduling)**
  - Inventé par Liu & Layland
  - Hypothèse du modèle (H1)
    - Algorithme à priorité fixe (constante au cours du temps)
    - Préemption possible
    - Chaque tâche est périodique
    - Pas de dépendances entre les tâches
    - L'échéance est la période
    - La priorité est l'inverse de la période
  - Hypothèse de faisabilité (H2)
    - Existe une condition suffisante (CS)
      - » *Borne théorique correspondant au pire des cas*
    - Ordonnancement sûr si critère vérifié

Algorithmes d'ordonnancement statique

- **RMS (Rate Monotonic Scheduling)**
  - Critère théorique
    - n tâches
    - Ci = durée d'exécution
    - Ti = période = échéance de chaque travail
    - Analyse du taux d'occupation du CPU (W) : $\sum_i \frac{C_i}{T_i}$
    - Si W ≤ U(n) , RMS est optimal

$$\begin{aligned} U(n) &= n * (2^{1/n} - 1) \\ U(1) &= 1 \\ U(2) &= 0.83 \\ U(3) &= 0.78 \\ U(\infty) &= 0.69 = \log 2 \end{aligned}$$

Algorithmes d'ordonnancement statique

- **RMS (Rate Monotonic Scheduling)**

$$W = \sum C_i / T_i$$

$$U(n) = n * (2^{1/n} - 1)$$

  - Condition nécessaire : W ≤ 1 (sinon surcharge)
  - Condition suffisante : W ≤ U(n)
  - Pour W entre CS et CN : impossible de savoir s'il y a une solution ou non
    - Il faut réaliser un plan d'ordonnancement sur une période d'étude « à la main »

Algorithmes d'ordonnancement statique

- **Conclusion RMS**
  - Avantages
    - Peut être étendu à une tâche apériodique
    - Peut être étendu à la surcharge
      - » *Les tâches de priorité haute ne sont pas impactées par les tâches de priorité basse*
    - Simple à implémenter
      - » *Très proche de la programmation en boucle*
  - Inconvénients
    - Hypothèses très simples (rarement rencontrées)
    - Famine si bug dans une tâche de haute priorité
      - » *Vérifier le temps pris par chaque tâche*

Algorithmes d'ordonnancement statique

- **Priorité et synchronisation**
  - Possibilité d'inter-blocage
    - Si une tâche moins prioritaire prend un mutex et qu'une tâche plus prioritaire le demande ensuite
  - Solution : héritage de priorité
    - Le détenteur du verrou hérite de la priorité du demandeur jusqu'à ce qu'il libère le verrou
    - Dans un système important : toutes les tâches finissent souvent à la même priorité
    - Complexifie l'analyse d'ordonnancement

Algorithmes d'ordonnancement dynamique

- **Premier arrivé / premier Servi (First Come/First Served)**
  - Avantages
    - Très simple et très rudimentaire
      - » *Non préemptif*
    - Vivacité (si les tâches se terminent bien sûr…)
  - Inconvénients
    - Ponctualité (=> système classique)
      - » *peut pénaliser les tâches courtes si une tâche longue les précède*
    - Sécurité
  - Utile pour garder un ordre de traitement implicite (pour les E/S)
    - Spooler d'impression
  - FCFS est souvent la stratégie « par défaut » pour beaucoup de ressources (mémoire, ports TCP/IP…)

Algorithmes d'ordonnancement dynamique

- **Plus court d'abord**
  - Avantages
    - Très simple et très rudimentaire (Non préemptif)
    - Vivacité (si les tâches se terminent bien sûr…)
  - Inconvénients
    - Ponctualité (=> système classique)
      - » *Les tâches longues sont pénalisées*
    - Sécurité
    - Impose de connaître la durée des tâches
      - » *Ce qu'on ne connait pas forcément (système classique)*
  - Temps le plus court d'abord
    - Version du « plus court d'abord » avec préemption
    - Préemption lorsqu'une tâche de temps d'exécution inférieur devient prête

Algorithmes d'ordonnancement dynamique

- **Round Robin (tourniquet)**
  - On partage le temps de façon « équitable » entre toutes les tâches en mesure de s'exécuter
    - On donne la main à une tâche pour (au plus) « K » unités de temps (quantum)
    - Après consommation de son quantum de temps, la tâche est placée en fin de la file d'attente des tâches prêtes
  - Avantages
    - Vivacité
    - Paralléliser les I/O et les traitements

Algorithmes d'ordonnancement dynamique

- **Round Robin (tourniquet)**
  - Inconvénients
    - Ponctualité (=> système classique)
    - Les performances dépendent de la taille du quantum de temps
      - » *Trop grand* ➔ *une tâche peut attendre longtemps pour avoir le processeur (temps de réponse)*
      - » *Trop petit* ➔ multiplie les commutations de contexte jusqu'à les rendre non négligeables
  - En pratique, RR est couplé avec priorité fixe

- **EDF (Earliest Deadline First)**
  - Hypothèse de modèle
    - Dynamique
    - Tâche apériodique
      - » *Inclus les problèmes périodiques*
    - Pas de dépendance entre tâche
    - Échéances connues
    - Durées inconnues
    - La priorité est l'inverse de l'échéance relative
    - Préemption
  - Hypothèse de faisabilité
    - Optimal si pas de surcharge (W ≤ 1)

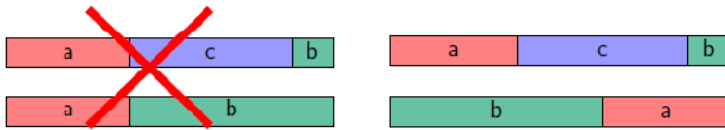
- **LLF (Least Laxity First)**
  - Hypothèse de modèle
    - Dynamique
    - Tâche apériodique
      - » *Inclus les problèmes périodiques*
    - Pas de dépendance entre tâche
    - Échéances connues
    - Durées d'exécution connues
    - La priorité est l'inverse de la laxité
      - » *En ligne, l'ordonnanceur calcule la laxité et exécute celle avec la plus petite*
    - Préemption
  - Hypothèse de faisabilité
    - Optimal si pas de surcharge (W ≤ 1)

- **Différence entre EDF et LLF**
  - si les valeurs de laxité utilisées sont celles qui sont calculées aux dates de réveil des tâches
    - Ordonnancement équivalent
  - si les valeurs de laxité sont calculées à chaque instant
    - LLF entraine beaucoup plus de changement de contexte
      - » *Laxité de la tâche qui s'exécute demeure constante alors que celles des tâches prêtes diminuent*

- **Différence entre théorie et pratique**
  - Hypothèses précédentes ne sont malheureusement presque jamais vérifiées
    - Préemption
      - » *Prends un certain temps*
    - Intégration de tâches critiques / non critiques
      - » *Souvent surcharge*
    - L'indépendance des traitements
      - » *Partage de ressources dont l'utilisation n'est possible qu'en exclusion mutuelle : ressources critiques*
      - » *Contraintes de précédence qui expriment les relations de synchronisation et de communication entre les tâches*
        - ➔ *graphes de dépendances*

Ordonnancement multiprocesseur

- **Ordonnancement sur multiprocesseur**
  - A tout moment t
    - Un travail est exécuté par au plus un processeur
    - Un processeur exécute au plus un travail
  - Ordonnancement non-accumulatif
    - Cela complexifie énormément le problème par rapport au monoprocesseur

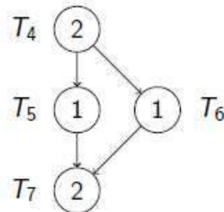


Ordonnancement multiprocesseur

- **Ordonnancement sur multiprocesseur**
  - Les latences ne sont plus négligeables
    - Communication, migrations
  - En monoprocesseur
    - Algorithmes optimaux « en-ligne » (EDF, LLF)
  - En multiprocesseur
    - Nécessité de « connaître le futur » pour ordonnancements optimaux… (Global EDF non optimal)
    - Différentes stratégies
      - » *Partitionnement (migrations interdites)*
      - » *Algorithmes globaux*

Ordonnancement multiprocesseur

- **Relations entre les travaux**
  - Travaux peuvent être reliés par un graphe de dépendance (ou graphe de précédence)
    - Exprime précédence et concurrence
    - Accès concurrents aux ressources, sérialisations



Ordonnancement multiprocesseur

- **Relations entre les travaux**
  - Date de réveil de la tâche est supérieur à toutes les dates de réveil de ses prédécesseurs immédiats augmentées de leur durée d'exécution
    - Une tâche ne sera donc activable que si tous ses prédécesseurs se sont terminés
  - L'échéance d'une tâche doit être inférieure à toutes les échéances de ses successeurs immédiats diminuées de leur temps d'exécution
  - Exemple pour le graphe (T1) ➔ (T2) ➔ (T3)
    - Td(T1) + C(T1) = Td(T2), date de début au plus tôt
    - Tf(T3) - C(T3) = Tf(T2), date de fin au plus tard

- **Hypothèse de cas concret**
  - Graphe de dépendance
  - Deux ressources sont disponibles (deux processeurs)
  - Pas de réquisition (pas de préemption)
  - En 6 unités de temps ?