Ordonnancement

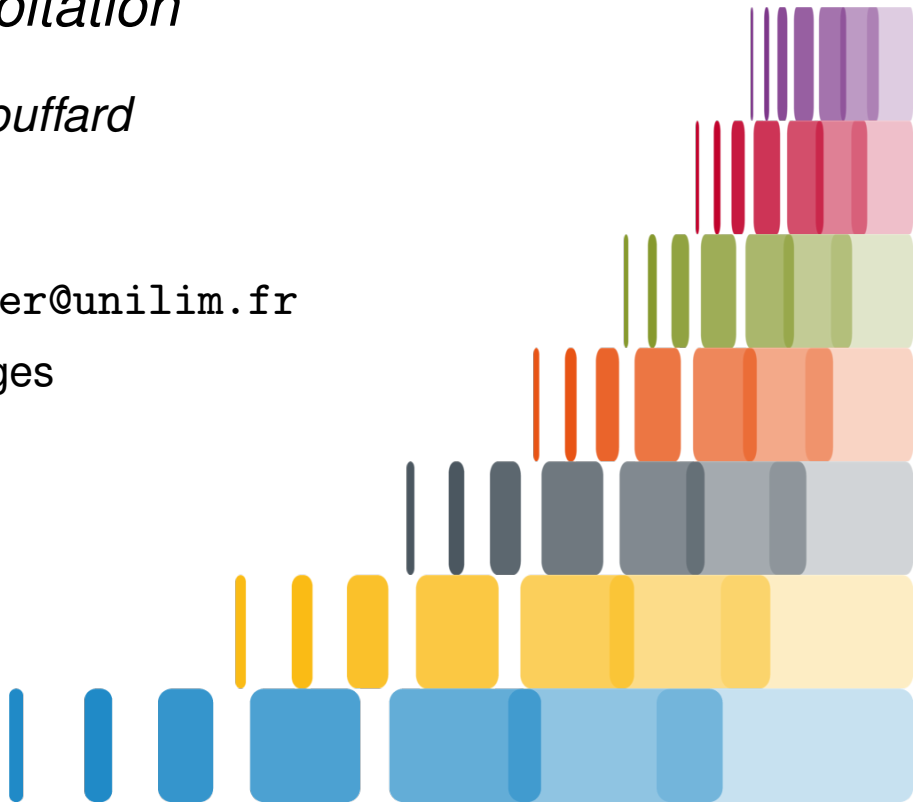
L3 Informatique – Systèmes d'Exploitation

Sources: Jean-Louis Lanet & Guillaume Bouffard

Vincent Neiger `vincent.neiger@unilim.fr`

Département Informatique – Université de Limoges

28 mars 2018



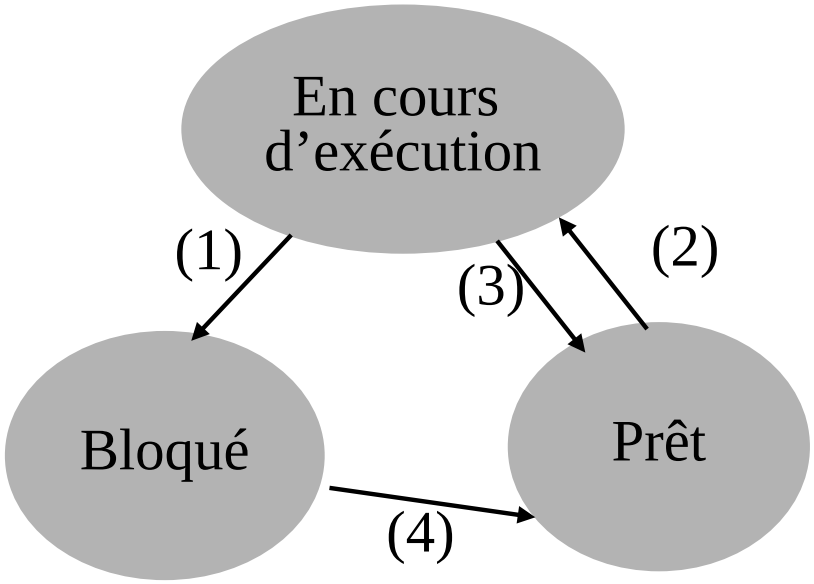
Plan

- Introduction : ordonnancement de processus
- Algorithmes classiques
- Systèmes temps réels

Processus

- Un processus est une activité: programme, entrées, sorties…
- Systèmes monoprocesseurs : *pseudo-parallélisme*
- Multiprogrammation: basculement entre processus
- Implémentation de Processus
  - Processus possède son propre espace d'adressage: programme, données, pile.
  - Le changement de contexte (changement de processus) → Table de processus, avec une entrée/ processus contenant registres, identificateur, ptr vers le segment texte, ptr vers segment de données, ptr vers le segment de pile, état …

États de Processus



- Le processus est bloqué, en attente d'une donnée, événement,
- L'ordonnanceur choisit un autre processus,
- L'ordonnanceur choisit ce processus,
- La donnée, l'évènement devient disponible.

Le problème

- Dans un système à processus :
  - de nombreux processus attendent qu'un événement se produise…
    - ils n'ont pas immédiatement besoin du processeur…
    - … mais doivent pouvoir l'obtenir dès que l'événement attendu se produit
  - certains processus font des calculs de façon intensive, sans attente d'événements
    - ils souhaitent garder le processeur le plus longtemps possible
  - Conflit d'intérêt : ordonnanceur (*scheduler*) = arbitre + chef d'orchestre

Objectifs d’un ordonnanceur

- Rôle d’un algorithme d’ordonnancement :
  - décider de l’allocation d’une ressource aux processus qui l’attendent, pour atteindre certains objectifs
  - dans la suite, « processus » (au sens large) signifie un:
    - processus (au sens Unix, processus « lourd »),
    - thread : fil d’exécution à l’intérieur de la mémoire d’un processus
- Exemple de ressource : le processeur
  - Objectif : aboutir à un partage efficace du temps d’utilisation du processeur
  - Problème : que veut dire efficace ? Et pour qui?

Critères d'efficacité pour le CPU

- Respect de la priorité
  - La plupart des systèmes permettent d'accorder des priorités différentes aux processus…
  - Priorité peut être statique ou dynamique (se modifie au cours du temps) …
- Respect de l'équité
  - Deux processus qui ont le même niveau de priorité doivent pouvoir utiliser le CPU aussi souvent l'un que l'autre

Critère d'optimisation pour l'ordonnancement du CPU

- Utilisation maximale du processeur
  - Maximiser: Taux Utilisation(CPU) = Durée Activité(CPU) / Durée Totale
- Débit processus
  - Maximiser Débit = Nombre Processus Terminés / Unité Temps
- Temps de traitement moyen :
  - doit être minimal pour un traitement batch
- Temps de réponse maximum :
  - doit être minimal pour un traitement interactif ou temps réel

Classification des algorithmes d'ordonnancement

- Dans un monde idéal (statistiquement) :
  - le hasard devrait bien faire les choses : les processus endormis ne devraient pas se réveiller tous en même temps
- Dans la réalité :
  - les activités des processus sont « corrélées » : les processus ne se réveillent pas au hasard …
- Deux familles d'algorithmes :
  - Sans réquisition : c'est aux processus de relâcher volontairement la ressource (non préemptif)
  - Avec réquisition : l'ordonnanceur peut récupérer la ressource détenue par un processus au profit d'un autre (préemptif)

Mécanismes de base nécessaires

- L'activation de l'ordonnanceur est possible
  - À chaque entrée dans le noyau, à chaque appel système,
  - À chaque interruption du matériel : disque, horloge, …
- Chaque appel système peut donc potentiellement activer un autre processus
- Ressources de type CPU :
  - commutation de contexte
    - Pour permettre à un autre processus d'utiliser la ressource,
    - Contexte peut être en partie matériel (registres, état),
- Algorithmes avec réquisition : besoin d'horloge
  - pour contrôler la durée d'utilisation
  - pour percevoir l'écoulement du temps: interruptions périodiques pour mesurer le temps passé et lancer des actions à des dates fixées

Plan

- Introduction : ordonnancement de processus
- Algorithmes classiques
  - Sans préemption
  - Avec Préemption
- Systèmes temps réels

Sans préemption

- Ressource allouée à une entité jusqu'à ce qu'elle n'en ait plus besoin
  - Par nécessité (ex: imprimante)
- Inconvénients :
  - ne peut convenir aux activités « temps réel »
  - convient difficilement aux activités interactives :
    - Obligation de programmer des applications « sociables »
    - Tolérable dans un système faiblement mono utilisateur (Windows 3.x, 95, …)
  - ne correspond pas à de vrais processus indépendants (il s'agit en fait de co-routines)
- Avantages :
  - facile à mettre en œuvre
  - pas besoin de mécanismes matériels spécifiques…(horloges, interruptions)

Mise en oeuvre

- Au moment de la libération de la ressource :
  - L’ex-détenteur de la ressource invoque l’algorithme d’ordonnancement
    - Cette action peut être réalisée à l'insu du programmeur (exemple : win3x, win9x)
  - L’algorithme **choisit** le processus suivant
  - L’algorithme déclenche la commutation de contexte

Politique de choix : FIFO

- Politique « FIFO » (First In First Out)
- Allocation dans l'ordre d'arrivée (premier arrivé = premier servi)
- Inconvénient : défavorise les entités ayant besoin d'utiliser la ressource un court laps de temps
  - Le temps d'attente n'est pas proportionnel au temps d'utilisation
    - pas équitable,
    - temps moyen de traitement élevé

Ordonnancement FIFO

Processus	Tps CPU
P_1	24
P_2	3
P_3	3

- Supposons que les processus arrivent dans l'ordre suivant: P_1 , P_2 , P_3 Le diagramme correspondant est:

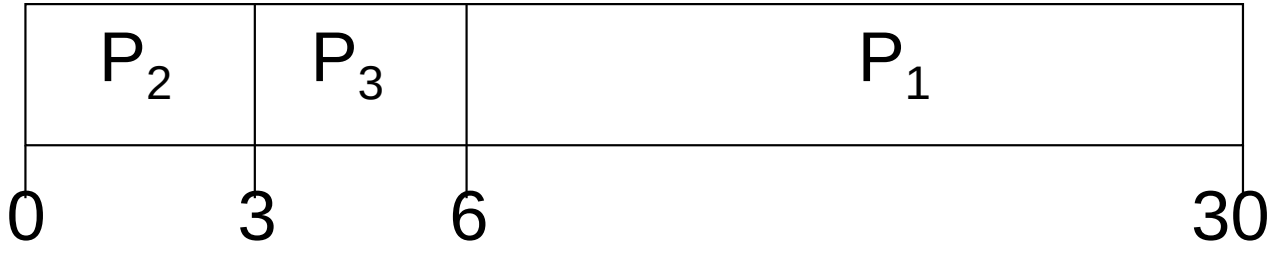
- Temps d'attente de P_1 = 0; P_2 = 24; P_3 = 27
- Temps d'attente moyen: (0 + 24 + 27)/3 = 17

Ordonnancement FIFO

Supposons que les processus arrivent dans l'ordre suivant

P_2 , P_3 , P_1

- Le diagramme de Gantt serait alors:



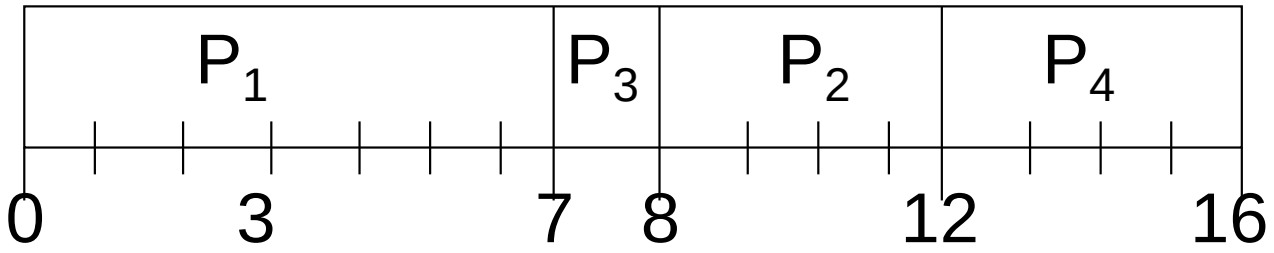
- Temps d'attente de P_1; P_2, P_3.?
- Temps d'attente moyen: ?
- Conclusion ?

Politiques de choix : PCTU

- Politique PCTU (Plus Court Temps d'Utilisation d'abord)
- Allocation selon ordre croissant de durée d'utilisation prévue
- Inconvénients
  - Pas réaliste : exige la connaissance a priori des durées d'utilisation
  - Famine (privation) : les tâches dont la durée d'exécution estimée est longue peuvent attendre leur tour indéfiniment …
- Avantages
  - Temps d'attente faible pour entités à courte durée d'utilisation
  - Temps moyen d'attente minimal
- Il est optimal – donne un temps moyen minimal pour un ensemble de processus donnés

Exemple de pctu

Processus	Tps d'Arrivée	Tps CPU
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4



- Temps moyen d'attente = (0 + 6 + 3 + 7)/4 = 4

Politique de choix : FIFO avec priorité

- Politique FIFO avec priorités
  - Chaque entité a une priorité
  - Une file FIFO par niveau de priorité
  - Ressource allouée à une entité ssi
    - FIFOs de priorités supérieures vides & la ressource est en tête de sa FIFO
  - Inconvénients
    - Tout le monde veut la plus haute priorité…
    - Famine pour entités de faible priorité
  - En pratique
    - utilisation parcimonieuse des priorités élevées
    - modification dynamique des niveaux de priorité

Plan

- Introduction : ordonnancement de processus
- Algorithmes classiques
  - Sans préemption
  - Avec Préemption
- Systèmes temps réels

Description

- Motivations
  - politiques sans réquisition mal adaptées, voire inadaptées, à certaines activités
    - temps réel
    - interactivité
- La réquisition permet:
  - de forcer le partage du temps d'utilisation (modulo les contraintes de priorités)
  - de diminuer le temps de traitement maximum
  - mais cela détériore le temps de traitement moyen (overhead = frais de gestion, temps passé dans le noyau), donc diminue le débit

Mise en oeuvre

- Le détenteur de la ressource peut être interrompu avant d'avoir terminé :
  - lorsqu'un délai maximal expire
  - lorsqu'un processus de priorité plus élevée demande la ressource
- La politique d'ordonnancement choisit le nouveau processus
- Le processus interrompu est mis «en sommeil» (état prêt)
- C'est la politique utilisée dans les systèmes « à temps partagé » (time-sharing) : Unix, NT…

Problème des fonctions non réentrantes

- Un processus peut être interrompu alors qu'il exécute une fonction de l'exécutif
  - Problème des fonctions non ré-entrantes dans la même mémoire :
  - Le nouveau/futur élu peut demander à son tour l'exécution de la même fonction :
    - Réutilisation d'une même variable globale
    - Insertion non terminée dans une liste chaînée
    - …
  - Solution : retarder la commutation jusqu'à ce que l'exécution atteigne un point de commutation, c.-à-d. contrôler les sections critiques du noyau.

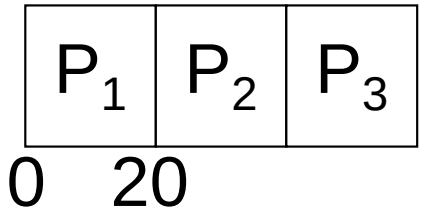
La politique du tourniquet

- Politique du Tourniquet (Round Robin / RR)
  - idée : Fournir à l'une quelconque des *n* entités en attente, *1/n* ème du temps d'utilisation de la ressource
  - pb : *n* varie au cours du temps
  - Solution : le temps est découpé en tranches de taille (durée)
  - identique, appelées quantum de temps
    - Les entités sont placées dans une file
    - La ressource est allouée à l'entité en tête de la file, pour une durée d'au maximum un quantum
    - Lorsque le quantum est épuisé, l'entité est interrompue et replacée à la fin de la file d'attente

Exemple de RR avec Q = 20

Processus	Temps CPU
P_1	53
P_2	17
P_3	68
P_4	24

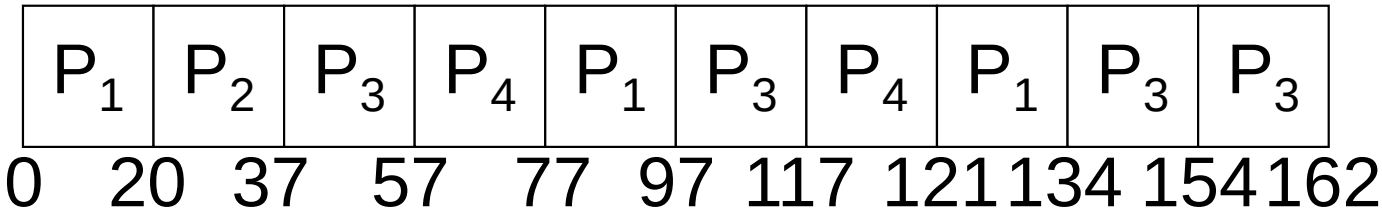
Le diagramme de Gantt est:



Exemple de RR avec Q = 20

Processus	Temps CPU
P_1	53
P_2	17
P_3	68
P_4	24

- Le diagramme de Gantt est:



- Typiquement, une moyenne de temps de rotation plus grande que pctu, mais un meilleur *temps de réponse*

Tourniquet : choix du quantum

- Durée ni trop courte …
  - Lorsque la durée du quantum est écoulée, il faut déclencher un changement de contexte
  - Un changement de contexte prend du temps : Plus le quantum de temps est petit, plus on perd souvent du temps à changer de contexte !
- … ni trop longue :
  - L'illusion d'exécution parallèle s'estompe
  - Lorsque la durée est trop longue, l'interactivité diminue
    - Exemple : quantum = 1s, 3 tâches de longue durée sont présentes et attendent le processeur. Elles comptent utiliser systématiquement tout leur quantum.
    - Une tâche interactive ne peut obtenir le processeur au mieux que toutes les 3 secondes (délai entre frappe clavier et affichage d'au moins 3 secondes …)

Tourniquet : partage du temps CPU

- Horloge programmée
  - Déclenche une interruption à intervalles de temps réguliers
    - interruption appelée « tic d'horloge » : tic, tac, …
  - Le traitant de l'interruption :
    - 1. Décompte du temps d'occupation CPU pour l'entité courante (initialisé à la valeur de quantum)
    - 2. Si temps restant = 0 : lancer algorithme d'ordonnancement pour choisir un nouveau processus
    - 3. Autres actions non liées à l'ordonnancement
    - 4. actions liées à l'ordonnancement effectuées à des tics principaux (versions évoluées du tourniquet)
    - 5. Commutation de contexte vers entité élue

Politique d'ordonnancement préemptif

- En théorie : à tout instant la ressource est détenue par le processus de plus haute priorité
  - il faut donc retirer la ressource au processus qui la possède lorsqu'elle n'est plus la plus prioritaire
- En pratique :
  - il suffit que l'exécutif regarde la priorité d'un processus qui naît ou se réveille
  - si la commutation est retardée (cf. pb ré-entrance), on parle d'inversion de priorité

Préemption : problème de famine

- Problème : famine des entités de faible priorité
  - Solution : ajustement dynamique des priorités
    - Plus une entité attend longtemps, plus sa priorité augmente
    - Lorsqu'une entité obtient (enfin) la ressource, sa priorité redescend au niveau initial
- Conséquence : provoque de nombreux changements dans les files de priorités
  - Car les processus de même niveau de priorité sont placés sur une même file (généralement FIFO)
  - La gestion des files doit être efficace !

Plan

- Introduction : ordonnancement de processus
- Algorithmes classiques
  - Sans préemption
  - Avec Préemption
- Systèmes temps réels
  - Introduction
  - Tâches périodiques
  - Tâches apériodiques

Introduction

- Deux types de systèmes
- Synchrone
  - Existence d'une base de temps commune,
  - Les évènements n'arrivent pas n'importe quand
- Asynchrone
  - Pas d'hypothèse sur les instants où les évènements peuvent se produire
- Le monde synchrone est plus « simple », le monde réel est plutôt asynchrone

Déterminisme

- Pouvoir garantir que le système respectera ses spécifications, notamment temporelles, pendant sa durée de vie
  - Ré-exécution donne des résultats identiques
- Méthodologie
  - Déterminer les cas pires
  - Conditions de faisabilité (CF)
  - Déterminer valeurs numériques CF
  - Vérifier

Algorithme déterministe

- Temps maximum d'exécution garanti
- Indépendant du contexte courant
- Indépendant de la valeur des arguments
- Principe valable pour toutes les fonctions d'un même service
  - allocation / libération pour gestion mémoire
- S'applique aux séquences d'exclusion mutuelle
  - Pas d'effet(s) de bord sur reste du système

Caractéristiques temporelles

- Durée maximum (pire cas) d'un thread TH_i : C_i
  - Thread seul sans interruption
  - Par analyse ou par mesure
- (Pire) temps de réponse d'un thread TH_i : R_i
  - Temps entre demande activation et réponse
  - Prend en compte délai dans exécution induit par les autres threads et l'overhead OS
- $R_i \geq C_i$

Contraintes Temporelles

- Échéance de terminaison au plus tard
- Thread TH_i activé à instant t_i doit être terminé au plus tard à instant $t_i + D_i$
- D_i : échéance relative
- $t_i + D_i$: échéance absolue

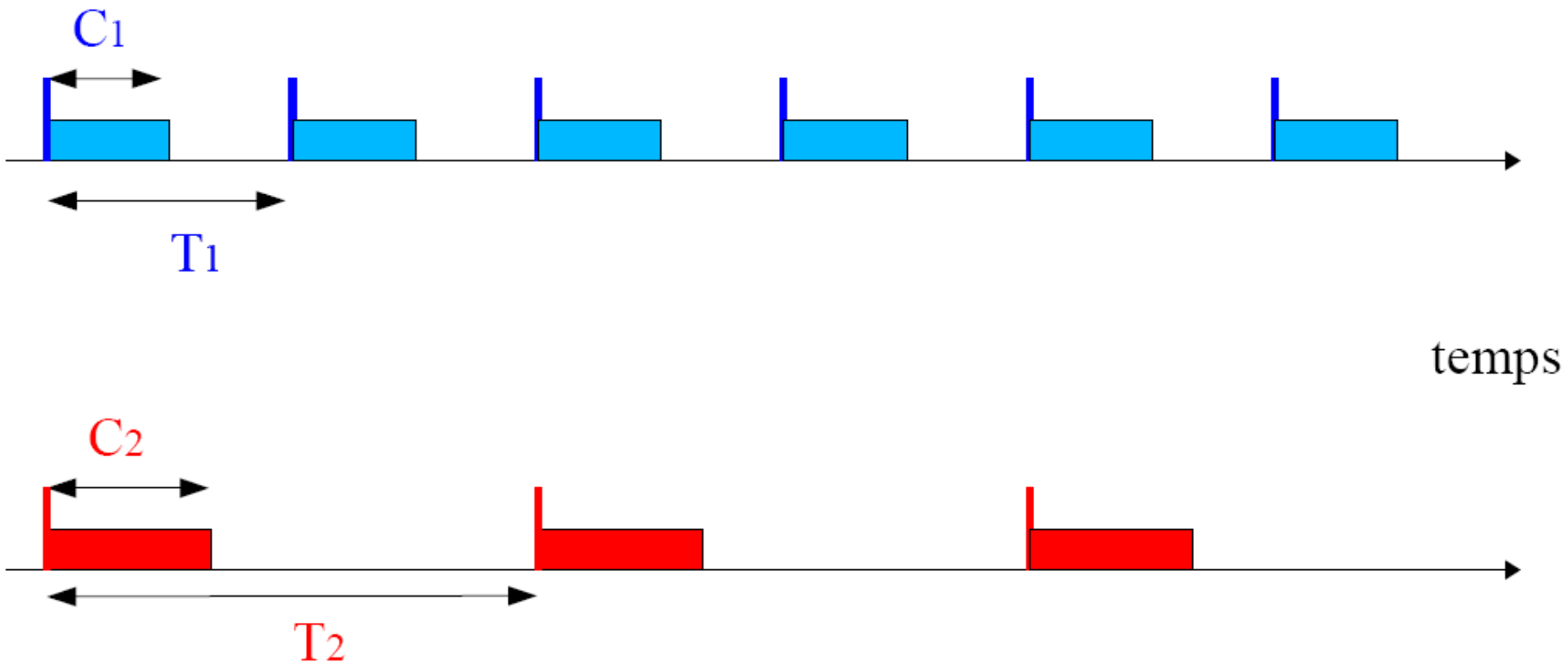
Plan

- Introduction : ordonnancement de processus
- Algorithmes classiques
  - Sans préemption
  - Avec Préemption
- Systèmes temps réels
  - Introduction
  - Tâches périodiques
  - Tâches apériodiques

Systèmes Périodiques

- Chaque thread THi activé périodiquement
  - Période activation Ti
  - Échéance relative Di (en général Ti = Di)
- La kième instance du thread THi
  - Est activée à (k-1) Ti
  - Doit être terminée au plus tard à k Ti
- Hyper période = PPCM(Ti) i = 1,...,n

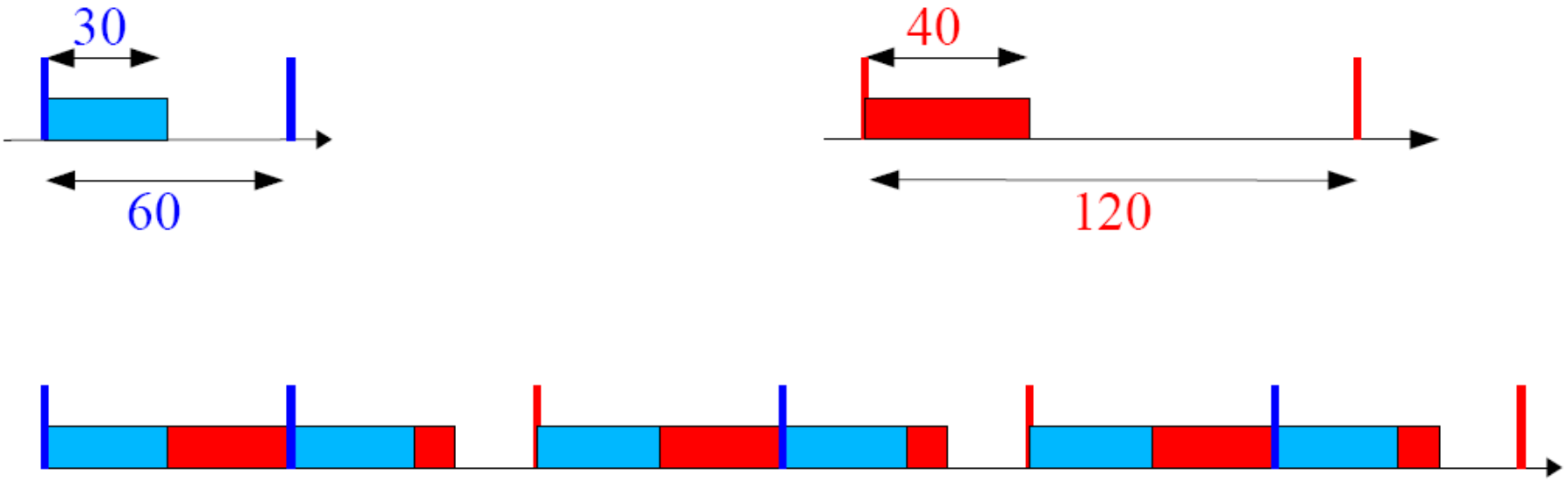
Systèmes Périodiques



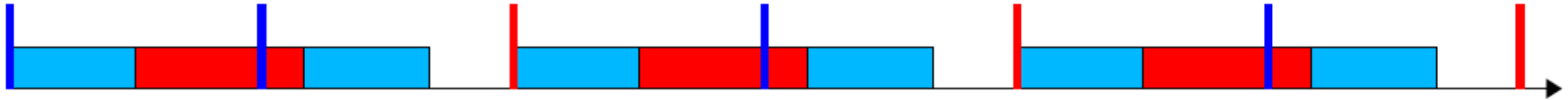
RMA

- Priorité déterminée en fonction de la période
- Plus la période est petite, plus la priorité est élevée
- Optimal
  - Pour systèmes périodiques
  - Avec ordonnancement préemptif

Ordonnancement Rate Monotonic



Ordonnancement "optimal" (minimise Nb. changement de contextes)



Test d'ordonnancement

- *Liu and Layland* ont démontré que lorsque la condition suivante est rencontrée on aura toujours un résultat d'ordonnancement **:**

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

- Quand N -> ∞ le terme de droite tend vers 69.3%

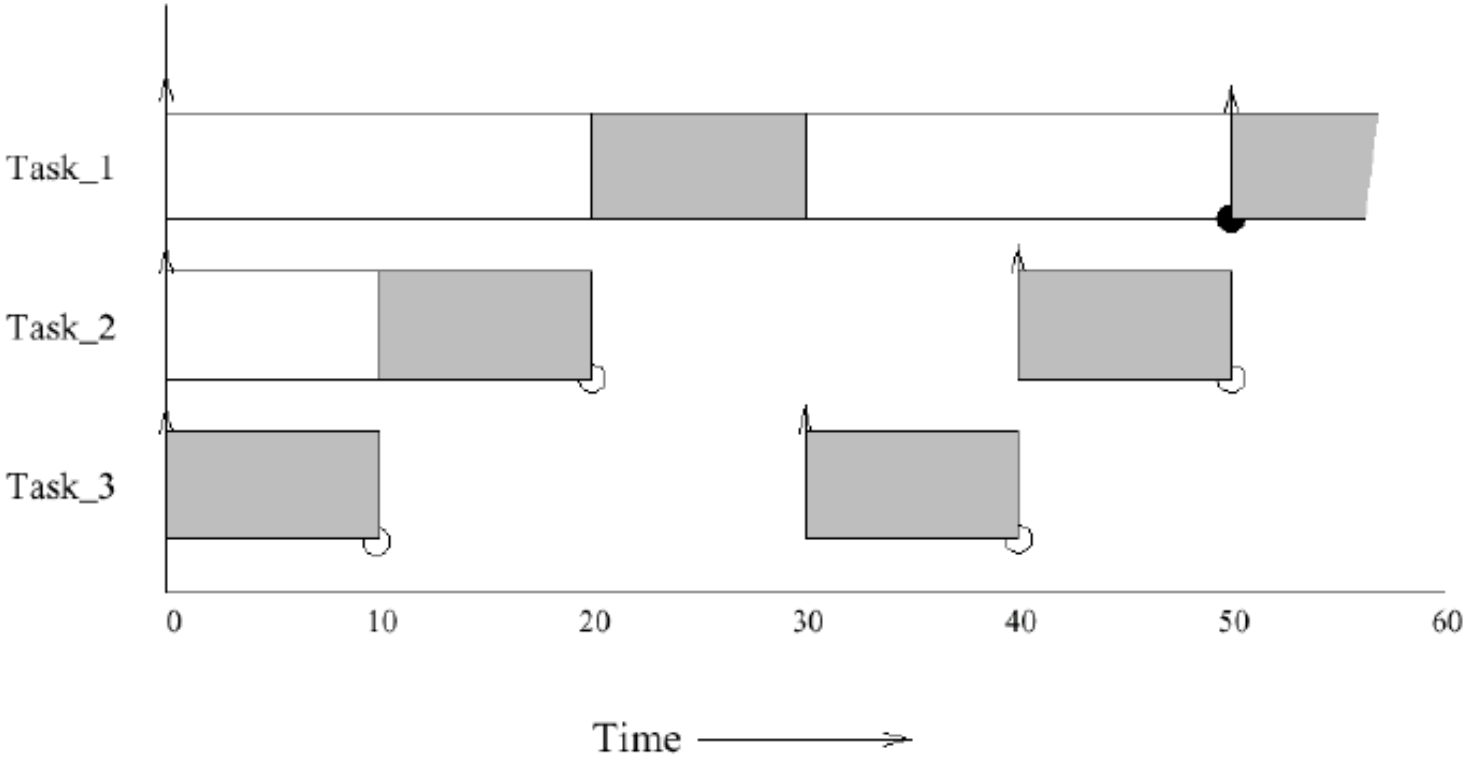
Condition d'ordonnancement

Exemple 1: Soit l'ensemble de tâches suivant à ordonnancer:

	Period T	Computation Time, C	Priority P	Utilization U
Task_1	50	12	1	0.24
Task_2	40	10	2	0.25
Task_3	30	10	3	0.33

Que peut on dire ?

RMA



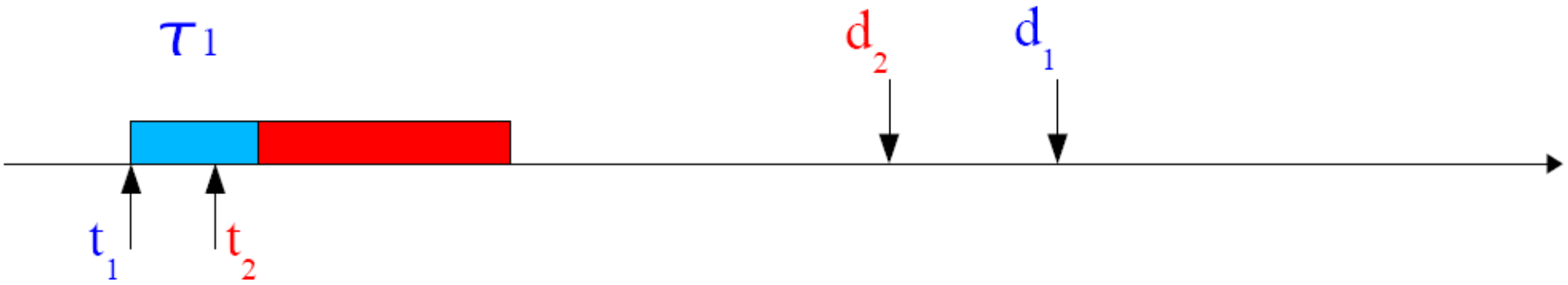
On voit graphiquement que l'ordonnancement est impossible.

Plan

- Introduction : ordonnancement de processus
- Algorithmes classiques
  - Sans préemption
  - Avec Préemption
- Systèmes temps réels
  - Introduction
  - Tâches périodiques
  - Tâches apériodiques

Tâches Sporadiques

- Chaque thread THi
  - Activée à un temps ti
  - Durée maximum d'exécution ci
  - Échéance absolue di = ti + Di
- Ordonnancement possible



Tâches Sporadiques

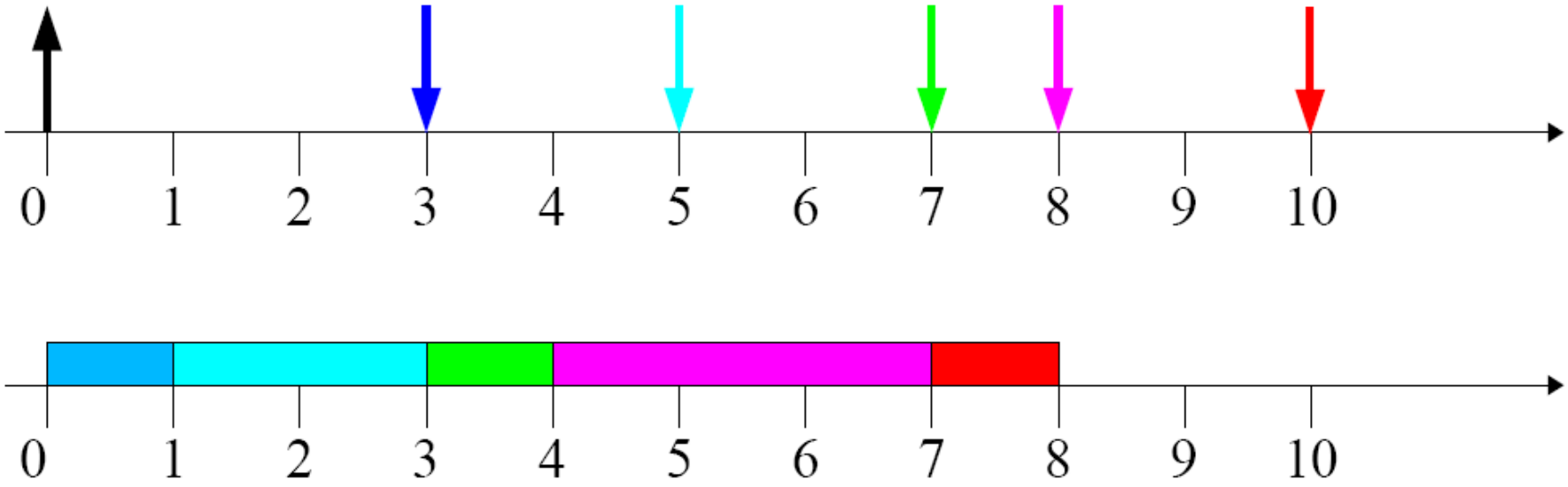
- Ordonnancement statique
  - Ensemble (ti ,ci ,di) i = 1,...,n connu avant exécution
  - Construire un ordonnancement qui respecte les échéances de chaque thread (ordonnancement faisable)
- Ordonnancement dynamique
  - A chaque "moment d'ordonnancement", déterminer la prochaine thread à exécuter
  - Satisfaire les échéances de toutes les threads

Ordonnancement EDF

- **E**arliest **D**eadline **F**irst
- Ordonnanceur dynamique
- Donne priorité à la thread la plus proche de son échéance de terminaison
- Doit trier les threads en fonction de leurs échéances
- Optimal si système non surchargé
- Comportement non-prédictible en cas de surcharge

EDF / même temps d'activation

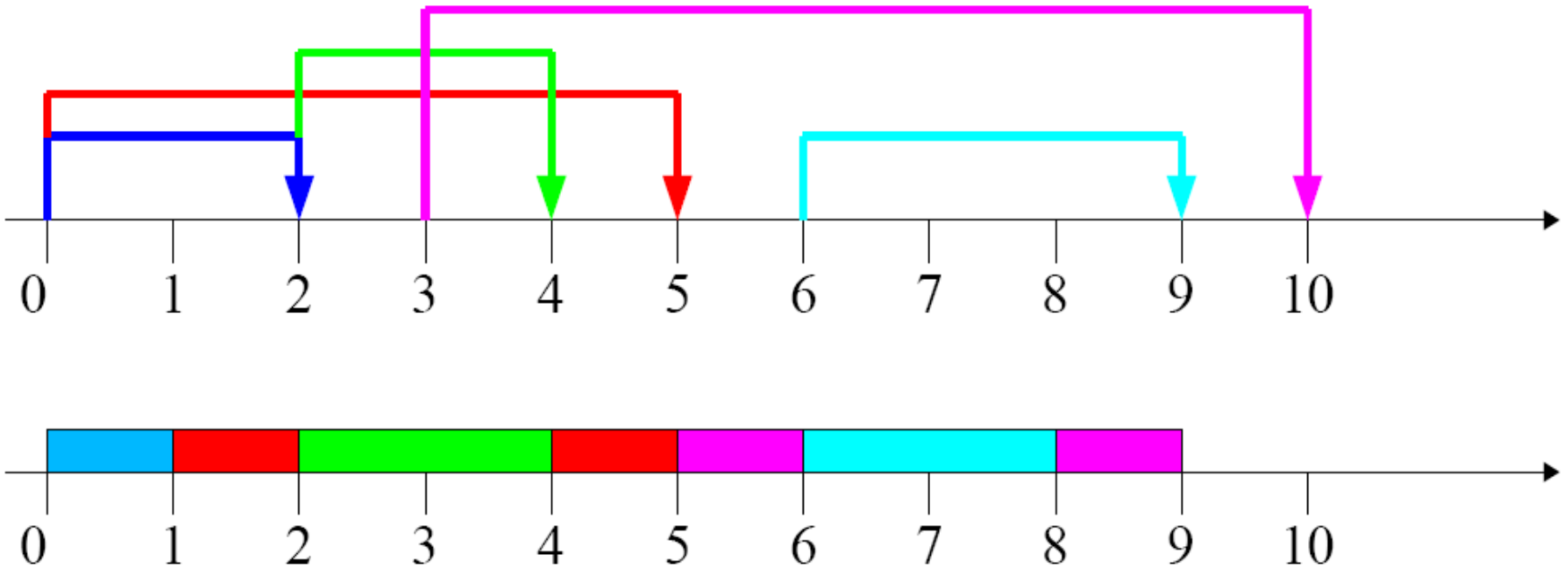
5 threads : (0,1,3), (0,1,10), (0,1,7), (0,3,8), (0,2,5)



préemption pas nécessaire

EDF / temps d'activation différents

5 threads : (0,1,2), (0,2,5), (2,2,4), (3,2,10), (6,2,9)



=> Préemption nécessaire pour satisfaire échéance de (2,2,4)

Inconvénients

- Ordonnancements basés sur le temps
  - Supposent CPU est la seule ressource partagée
- Dépendants
  - Caractéristiques matériel
    - CPU (instructions, fréquence, cache(s), etc...)
    - Performances bus (mémoire, I/O, etc)
  - Compilateurs
- Très sensibles aux évolutions du logiciel
  - Correction de bugs
  - Ajout de tâches

Inconvénients

- Décomposition des activités en blocs d'exécution synchrones :
- thread = (activation, durée maximum, échéance)
  - Figé
  - [relativement] simple
- Décomposition en étapes asynchrones de priorités différentes :
- étape = interruption / thread
  - Plus souple
  - Plus complexe

Any question ?