Systèmes d'exploitation embarqués et temps-réel

Chapitre 2 : Ordonnancement temps-réel

Aimen Bouchhima

2020-2021

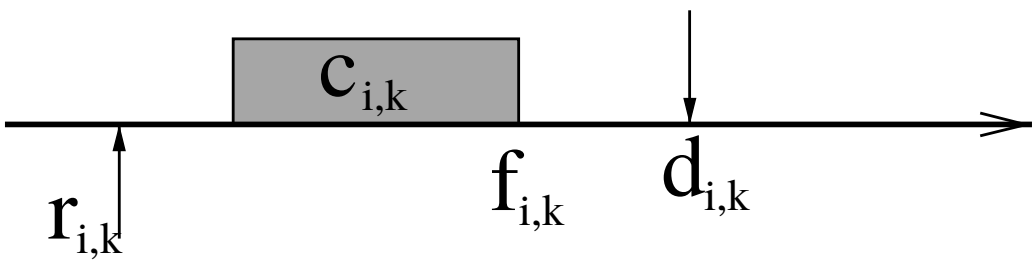
Plan

Processus/Thread/Tâche

- Un processus est un programme en exécution (instance de programme)
- Un Thread (processus léger) est un chemin d'exécution (flow of execution) dans un processus
- Une tâche : Processus ou Thread

Modèle de tâche temps-réel

- Une tâche temps-réel τ_i est une séquences d'actions $J_{i,k}$
- Une action $J_{i,k} = (r_{i,k}, c_{i,k}, d_{i,k})$:
  - arrive à l'instant $r_{i,k}$ (instant d'activation)
  - execute pendant une durée $c_{i,k}$
  - se termine à l'instant $f_{i,k}$
  - doit se terminer avant l'échéance absolue $d_{i,k}$ (deadline)



- Temps de réponse d'une activité $J_{i,k}$: $\rho_{i,k} = f_{i,k} - r_{i,k}$

Tâche périodique

- Une tâche périodique $\tau_i = (C_i, D_i, T_i)$ est une séquences d'actions $J_{i,k}$ avec
  - $r_{i,k+1} = r_{i,k} + T_i$
  - $d_{i,k+1} = r_{i,k} + D_i$
  - $C_i = max(c_{i,k})$
- T_i est la période de la tâche et D_i son échéance relative
- C_i est le temps d'exécution au pire cas (WCET)
- R_i : temps de réponse au pire cas :
  $R_i = max(\rho_{i,k}) = max(f_{i,k} - r_{i,k})$
  - Une tâche qui respecte son échéance doit vérifier : $R_i \leqslant D_i$

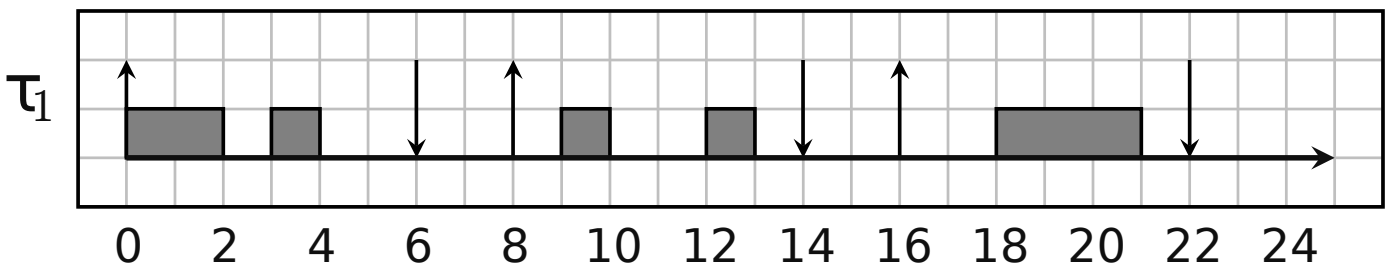
Tâche périodique : exemple

- Une tâche périodique à une structure régulière
  - s'active périodiquement (chaque T_i)
  - effctue un traitement
  - se bloque en attendant la prochaine activation

```
void * PeriodicTask(void * arg)
{
  <initialization>;
  <start periodic timer, period = T>;
  while (1) {
    <read sensors>;
    <update outputs>;
    <update state variables>;
    <wait next activation>;
  }
}
```

Tâche périodique : représentation graphique

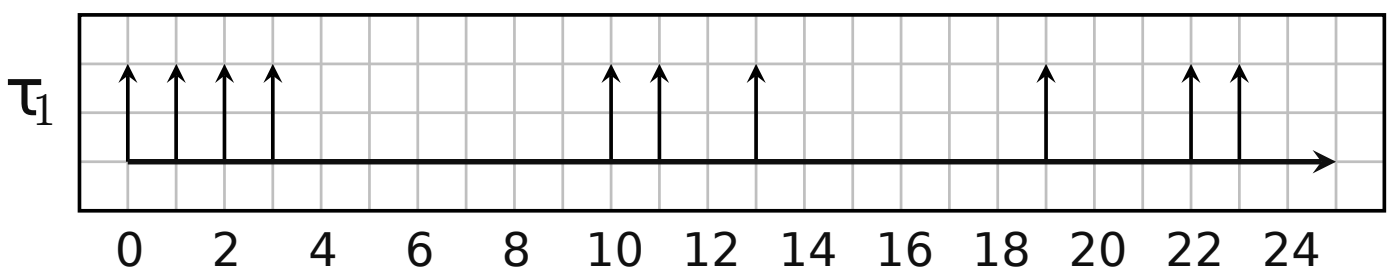
- Les tâches sont représentées graphiquement par un diagramme d'ordonnancement
- Exemple : tâche périodique $\tau_1 = (3, 6, 8)$
  - $C_i = 3, D_i = 6, T_i = 8$



- Remarque : les actions $J_{1,1}$ et $J_{1,3}$ exécutent pendant 3 unités de temps (WCET), alors que $J_{1,2}$ consomme 2 unités de temps

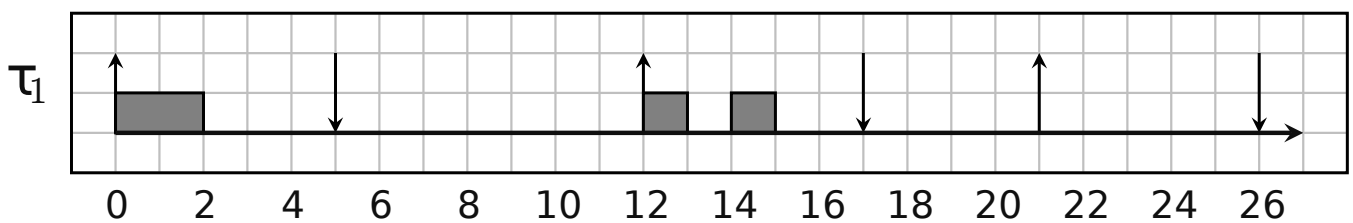
Tâche apériodique

- Une tâche apériodique n'est pas caractérisée par une période
  - Le temps entre deux activations successives n'est pas connu à l'avance
  - On ne peut même pas garantir un temps minimal entre deux activations successives
- Les tâches apériodiques peuvent modéliser
  - La réponse à des évènements rares et imprévisibles qui proviennet de l'environnement
  - La réponse à des évènements irréguliers (exemple des paquets venants du réseau)



Tâche sporadique

- Tâche apériodique avec un temps minimal entre deux activations successives
- Une tâche sporadique est modélisée comme une tâche périodique $\tau_i = (C_i, D_i, T_i)$
  - T_i : le temps minimal inter-arrivés (minimum interarrival time)
- Exemple $\tau_1 = (2, 5, 9)$



Criticité d'une tâche

- Une échéance est ferme (hard), si le dépassement de cette échéance provoque une défaillance critique du système
  - Une tâche est dite temps-réel ferme (ou dûr) si tous ses échéances sont fermes ($\forall j, \rho_{i,j} \leq D_{i,j} \Rightarrow R_i \leq D_i$)
- Une échéance est souple (soft), si le dépassement de cette échéance provoque une dégradation de la qualité de service (QoS), mais pas la défaillance du système
  - Une tâche est dite temps-réel souple si elle a une échéance souple

Exemples

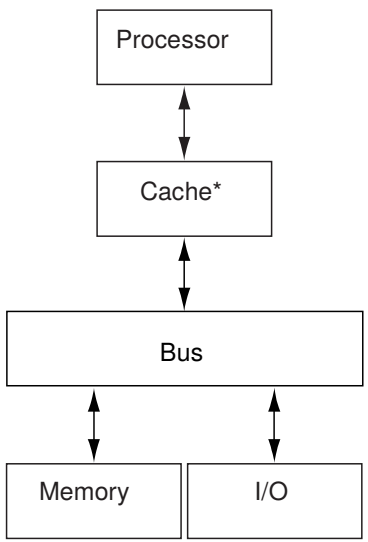
- Temps-réel ferme : un contrôleur de robot mobile doit détecter les obstacles et répondre dans un délai qui dépend de sa vitesse, autrement, le robot heurtera les obstacles
- Temps-réel souple : Lecteur vidéo
  - $fps = 25 \Rightarrow$ délai pour traiter une image (frame) $= 40$ms
  - Si une image est affichée un peu en retard, l'utilisateur ne s'apercevra même pas de la dégradation de la qualité de service
  - Mais si le taux d'images affichées avec retard ou supprimées dépasse un certain seuil, ça deviendra gênant pour l'utilisateur
  - Dans tous les cas, temps-réel souple ne signifie pas absence d'échéance.

Définition d'un ordonnanceur

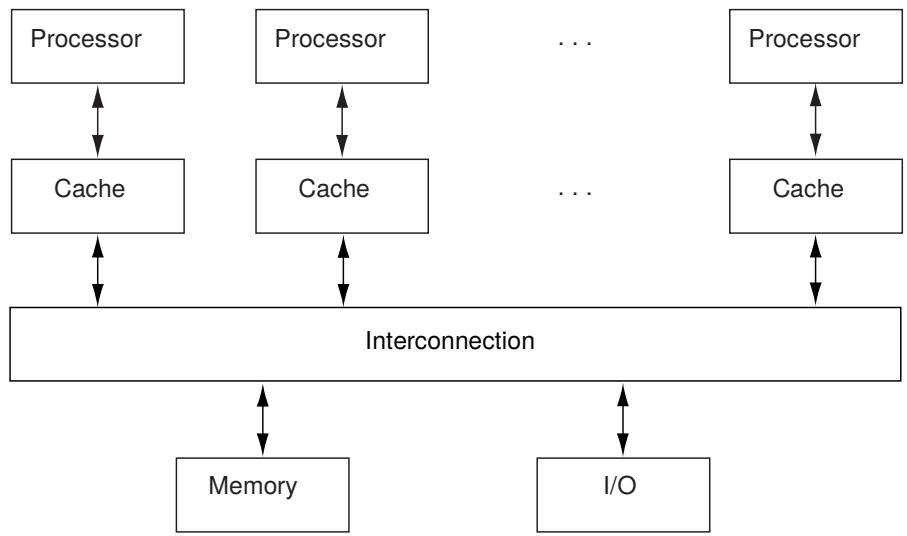
- Un ordonnanceur est un composant du système d'exploitation qui implémente un algorithme permettant de dispatcher l'ensemble des tâches $\boldsymbol{\tau}$ sur l'ensemble des processeurs physiquement disponibles dans la machine
- Formellement, un ordonnanceur pour une architecture contenant m processeurs implémente $\sigma(t)$, une fonction qui à chaque instant t associe un vecteur de tâches
  $\sigma : t \rightarrow (\boldsymbol{\tau} \cup \tau_{idle})^m$
  - τ_{idle} : tâche particulière *idle* qui, lorsqu'elle "s'exécute" sur un processeur, celui-ci devient en mode "idle"
  - Cas $m = 1$ (Uni-processeur) : $\sigma : t \rightarrow \boldsymbol{\tau} \cup \tau_{idle}$

Architecture matérielle

- Dans la suite du cours, on distingue le cas uni-processeur (UP) et le cas multi-processeur (Symetrical MultiProcessors SMP)
- L'architecture SMP est un cas particulier de MIMD (Multiple Instruction Multiple Data) dans laquelle la mémoire est accessible par tous les processeurs d'une façon uniforme



Architecture UP



Architecture SMP

Ordonnanceur Préemptif Vs Coopératif

Un ordonnanceur est dit préemptif s'il a le pouvoir d'interrompre une tâche lorsque celle-ci est en cours d'exécution. Dans le cas contraire, l'ordonnaceur est dit non-préemptif ou coopératif

- En pratique, l'implémentation de la préemption se fait via l'interruption matérielle
  - Suite à une interruption matérielle, le processeurs interrompt l'exécution de la tâche (mais termine l'instruction en cours comme même) pour se brancher à une adresse fixe dans la mémoire instruction (routine de gestion d'interruption ou ISR)
  - L'ISR fait partie du code du système d'exploitation
  - A partir de là, l'ordonnanceur récupère la main ...
- Un ordonnaceur coopératif doit attendre que la tâche rend la main explicitement (soit elle se termine soit elle fait un appel système qui peut être simplement "passer la main" ou yield)

Ordonnanceur Statique Vs Dynamique

- Ordonnancement statique appelé aussi ordonnacement hors ligne (offline)
  - Le choix des tâches à exécuter se fait, à l'avance, par le concepteur du système.
  - Non préemptif : chaque tâche est exécutée jusqu'à terminaison
  - Ordonnaceur simple : il consulte, à des points précis sur l'axe temps, une table (planning) qui contient la tâche à exécuter à cet instant précis
- Ordonnancement dynamique appelé aussi ordonnacement en ligne (online)
  - Pas de planification en avance des tâches.
  - L'ordonnaceur se base sur l'information disponible à un instant donné (ensemble des tâches actives plus éventuellement des méta-données associées comme les priorités des tâches) pour prendre la décision adéquate (politique d'ordonnancement).

Ordonnaceur Cyclique

- Cas particulier d'ordonnancement statique pour tâches périodiques
- Idée :
  - Calculer le pgcd (Cycle mineur Δ) et le ppcm (Cycle majeur T) de toutes le tâches
  - L'axe de temps est divisé en créneaux égaux au cycle mineur
  - Les créneaux sont statiquement alloués aux tâches (via le planning) sur toute la période d'un cycle majeur
  - Un timer matériel interrompt le système périodiquement à chaque créneau

Ordonnaceur Cyclique : exemple

On considère trois tâches périodiques τ_1, τ_2 et τ_3 à échéance sur requête ($D_i = T_i$)

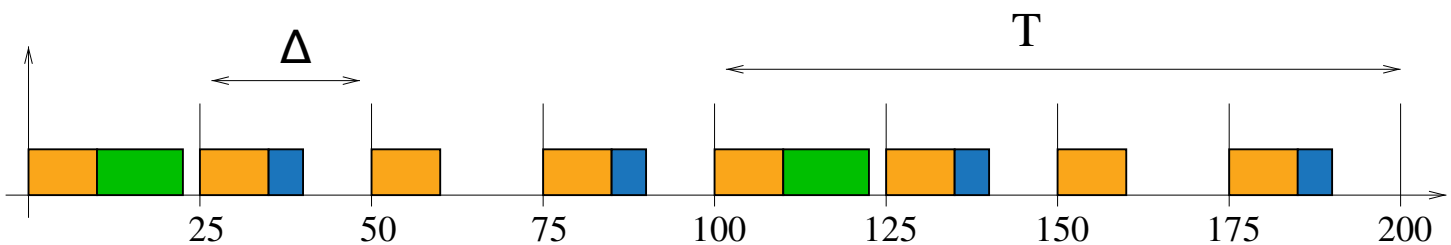
Tâche	C_i(ms)	T_i(ms)
τ_1	10	25
τ_2	5	50
τ_3	13	100

$\Rightarrow$

Créneau	Tâches
1	$\{\tau_1, \tau_3\}$
2	$\{\tau_1, \tau_2\}$
3	$\{\tau_1\}$
4	$\{\tau_1, \tau_2\}$

$\Delta = 25,\ T = 100$

Plannning



Ordonnaceur Cyclique : Implémentation

- Exercice (TP) : implémenter l'exemple précédent pour une cible micro-contôleur (STM32)
  - Chaque tâche effectue la séquence suivante : mettre un pin GPIO à 1, faire un traitement puis remet le pin à 0

```
// Declare global variable schedule

void * SysTick_ISR()
{
}

int main()
{
  //(1) Enable clock for
  //GPIOA and configure pins
  //(2) start SysTick Timer
  while (1) {
    // Equivalent to Idle task
  }
}
```

```
void * Task1(void * arg)
{
  GPIO_ResetBits(GPIOA, GPIO_Pin_0);
  for (i=0; i<N1; i++) {
    // consume some cycles to
    // simulate computation time
    // of task 1 (adjust N1
    // accordingly)
  }
  GPIO_SetBits(GPIOA, GPIO_Pin_0);
}
```

Ordonnaceur Cyclique : Avantages et Inconvénient

- Avantages
  - Implémentation simple
    - Simple appels de fonction
    - Système d'exploitation non nécessaire
    - Pas besoin de protéger les resources partagées par des mutex, sémaphores,...
  - Surcoût minimal lors de l'exécution (taille mémoire et latences)
  - Plus facile à tester et valider (utilisé dans les systèmes critiques)
- Inconvénients
  - Implémentation simple
    - Le temps d'exécution des tâche doit être connu avec précision
    - Que se passe si une tâche n'arrive pas à terminer son traîtement avant l'interruption TIMER ?
    - Difficulté de gérer les tâches apériodiques et sporadiques

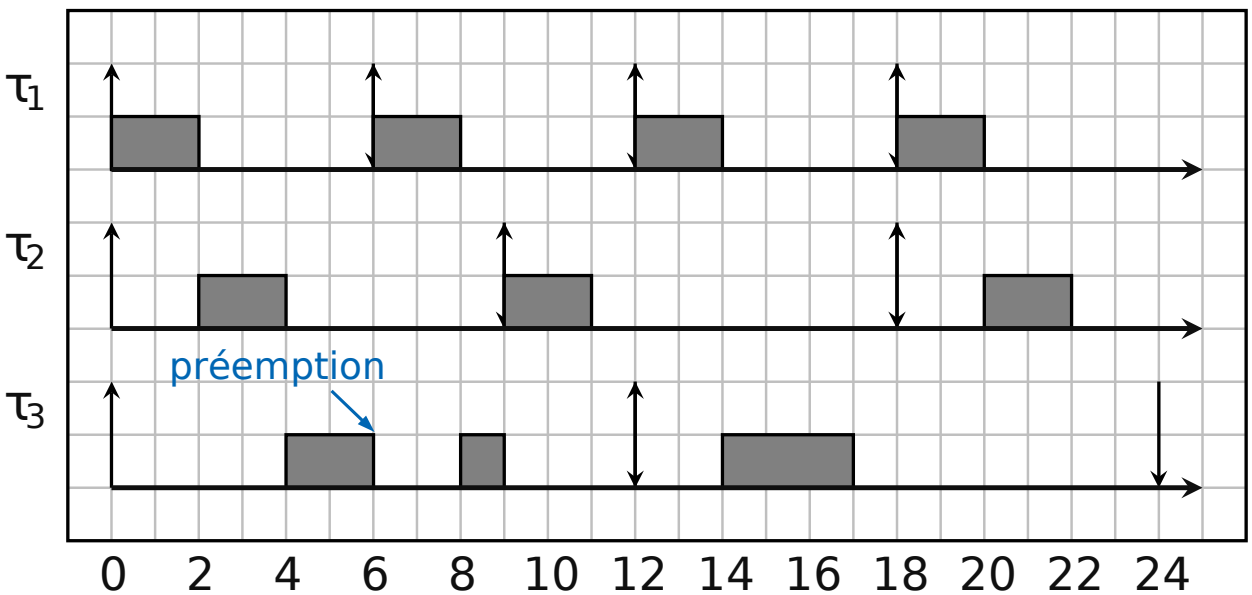
Ordonnaceur à priorité fixe

- Cas particulier d'ordonnancement dynamique
- Ordonnancement préemptif
- Principe simple :
  - Chaque tâche τ_i possède une priorité fixe p_i
  - A tout moment, La tâche active ayant la priorité la plus élevé est exécutée
- Dans les deux exemples suivant, on considère des tâches périodiques à temps d'exécution constant et à échéance sur requête

Exemples d'ordonnancement

- Exemple 1

Tâche	C_i	T_i	p_i
τ_1	2	6	3
τ_2	2	9	2
τ_3	3	12	1

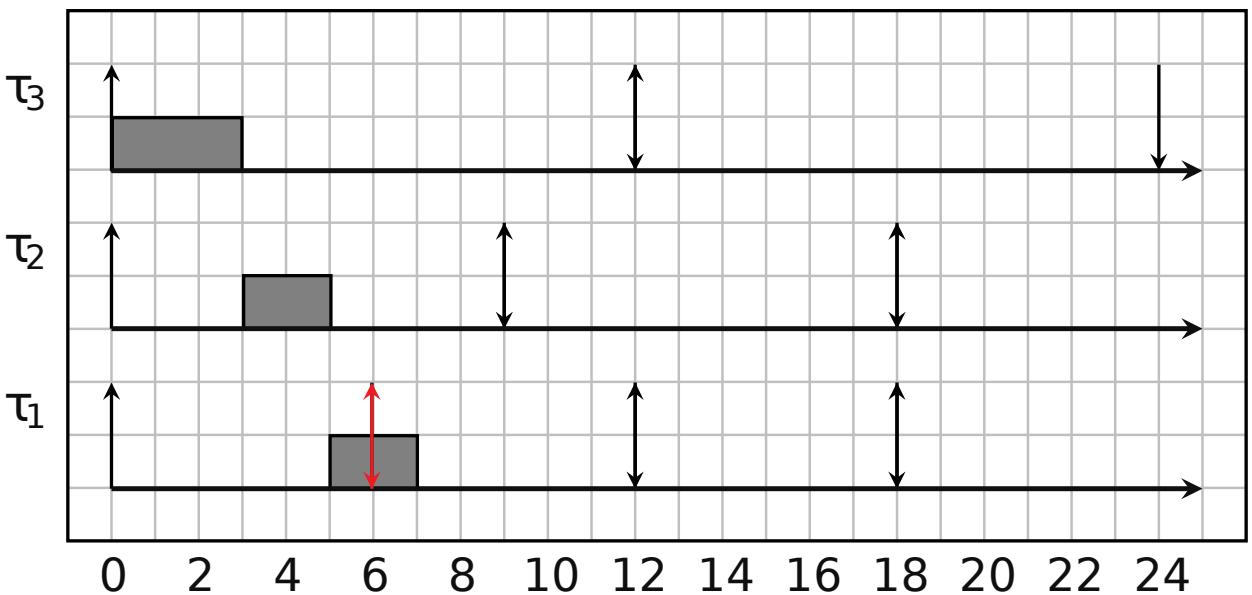


- Pas de dépassement d'échéance $\Rightarrow$ ordonnançable

Exemples d'ordonnancement

- Exemple 2 (non ordonnançable)

Tâche	C_i	T_i	p_i
τ_1	2	6	1
τ_2	2	9	2
τ_3	3	12	3



- La tâche τ_1 a dépassé son échéance $\Rightarrow$ non ordonnançable

Comment fixer les priorités ?

- L'affectation des priorités au tâches peut influencer l'ordonnançabilité des ces tâches
- Problème : comment affecter les priorités pour que les tâches soient ordonnaçables ?
- Objectif : Trouver une affectation optimale *Opt* des priorités pour un ensemble de tâches τ telque:
  - Si τ est ordonnançable par une autre affectation de priorité, alors il est ordonnançable par *Opt*
  - Si τ n'est ordonnançable par *Opt*, alors il ne l'ai pas pour toutes autres affectations de priorité

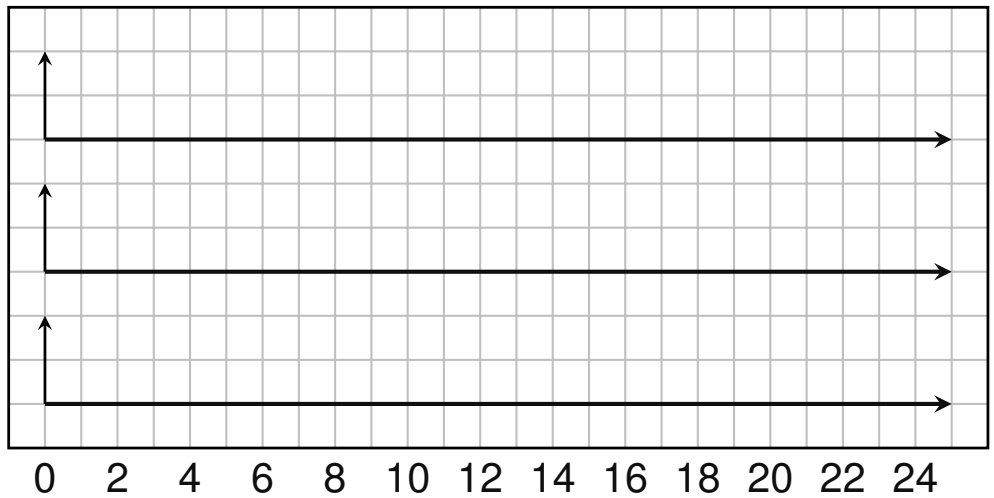
Affectation optimale des priorités

- Etant donné un ensemble de tâches périodiques à échéances sur requêtes ($D_i = T_i$) et avec un décalage égale à zéro ($r_{i,0} = 0$)
  - L'affectation optimale des priorités est celle appelé "Rate Monotonic" (RM)
  - Plus la période est petite $\rightarrow$ plus la priorité est élevée
- Etant donné un ensemble de tâches périodiques avec un décalage égale à zéro ($r_{i,0} = 0$)
  - L'affectation optimale des priorités est celle appelé "Deadline Monotonic" (DM)
  - Plus l'échéance relative est petite $\rightarrow$ plus la priorité est élevée
- RM est un cas particulier de DM

Exercice

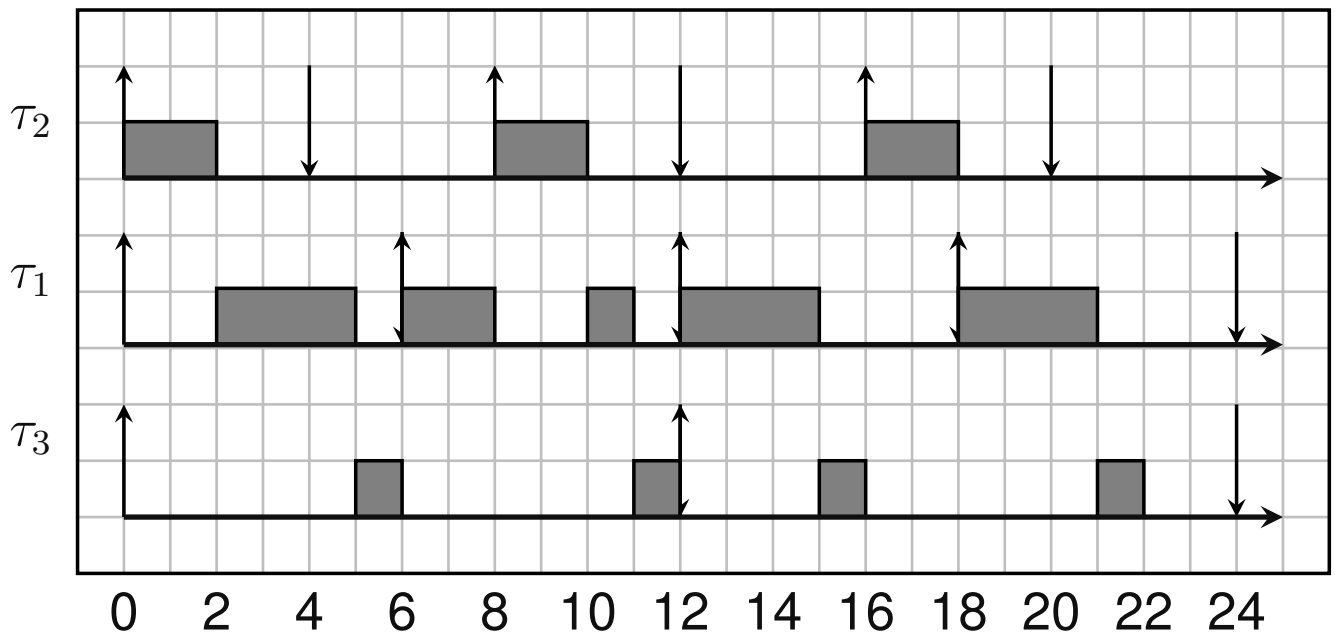
- On considère l'ensemble des tâches:

	C_i	D_i	T_i
τ_1	3	6	6
τ_2	2	4	8
τ_3	2	12	12



- Proposer un ordonnancement optimal avec priorités fixes
- Est ce que le système est ordonnançable ? discuter.

Exercice (correction)



Test d'ordonnançabilité

- Comment affirmer qu'un ensemble de tâches ordonnancées selon RM/DM est ordonnançable ou non ?
- Méthode 1 : Simuler l'exécution des tâches à la main sur une période de test $T_{analyse}$ suffisante (voir exercice précédent)
  - Tâches sans décalage ($r_{i,0} = 0$) :
    $T_{analyse} = ppcm(T_i) = CycleMajeur = T$
  - Tâches avec décalage ($r_{i,0} \neq 0$) :
    $T_{analyse} = 2T + max(r_{i,0})$
- Méthode 2 : Test basé sur le seuil limite d'utilisation du processeur
  - Test suffisant mais pas nécessaire
- Méthode 3 : Test basé le temps de réponse
  - Test nécessaire et suffisant qui peut être automatisé par un algorithme

Méthode 2 : Test basé sur l'utilisation du processeur

- On se place dans le cas monoprocesseur (UP)
- Chaque tâche utilise le processeur pendant une fraction U_i
  $U_i = \frac{C_i}{T_i}$
- l'utilisation totale du processeur U (charge du processeur)
  $U = \sum U_i = \sum \frac{C_i}{T_i}$
- Si $U > 1$, alors sûrement l'ensemble de tâches n'est pas ordonnançable
- Si $U \leq 1$, l'ensemble de tâches peut être ordonnançable ou non
- Etant donnée un algorithme d'ordonnancement, il existe un seuil minimal limite d'utilisation du processeur U_{lub} (Least Upper Bound) au dessous duquel on peut affirmer l'ordonnançabilité
  $U \leq U_{lub} \Rightarrow$ Ordonnançable

Méthode 2 : Test basé sur l'utilisation du processeur

- Etant données n tâches périodiques ordonnancées par RM :
- $U_{lub} = n(2^{\frac{1}{n}} - 1)$
  - U_{lub} est une fonction décroissante de n

n	U_{lub}
2	0.828
3	0.779
4	0.756
5	0.743
6	0.734
...	...
∞	0.69

Méthode 2 : Test basé sur l'utilisation du processeur

- Ainsi, pour RM, le test rapide d'ordonnançabilité consiste à :
  - Calculer $U = \sum \frac{C_i}{T_i}$
  - Si $U \leq U_{lub}$, alors les tâches sont ordonnançables
  - Si $U > 1$, alors les tâches ne sont pas ordonnançables
  - Si $U_{lub} < U \leq 1$, alors il faut pousser l'analyse avec la méthode 1 ou la méthode 3
- Pour DM, la même démarche s'applique, mais en prenant $U = \sum \frac{C_i}{D_i}$
  - Pour le voir, il suffit de raisonner sur le même ensemble de tâche τ' mais avec $T'_i = D_i$
  - le nouveau système τ' peut être vu comme un "pire cas" du système original
  - Si τ' est ordonnançable (en utilisant RM) $\Rightarrow$ Le système original est ordonnaçable

Méthode 3 : analyse des temps de réponses

- Rappel : Une tâche qui respecte son échéance doit vérifier : $R_i \leqslant D_i$, avec R_i le temps de réponse de la tâche
- Le temps de réponse d'une tâche dépond :
  - de son propre temps d'exécution
  - des temps d'exécution de toutes les tâches de plus fortes priorités
- le temps de réponse d'une tâche :

$$R_i = C_i + \sum_{j>i} \left\lceil \frac{R_i}{T_j} \right\rceil C_j = F(R_i)$$

- R_i est le plus petit point fixe de la fonction monotone F

Méthode 3 : analyse des temps de réponses

- Solution algorithmique:
  - $W_0 = C_i$
  - $W_1 = F(W_0)$
  - $W_2 = F(W_1)$
  - ...
  - $W_{k+1} = F(W_k)$
  - On s'arrête quand
    - $W_{k+1} = W_k$ et $W_k \leq D_i \Rightarrow$ ordonnaçable
    - ou $W_k > D_i \Rightarrow$ non ordonnançable

Ordonnaceur à priorité variable

- Earliest deadline first (EDF)
- Objectif : obtenir une meilleure occupation du processeur
- Principe : A tout moment, la tâche qui a l'échéance (absolue) la plus proche occupe le processeur
- Théorème : Un système est ordonnançable avec EDF si est seulement si $U = \sum \frac{C_i}{T_i} \leq 1$
  - Avantage : ordonnancement optimal
  - Inconvénient : plus difficile à implémenter que RM