الجمهوريـة الجزائريـة الديمقراطيـة الشعبيـة

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT
SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE
Université Mustapha Stambouli de Mascara

Faculté des Sciences exacts
Département Informatique

وزارة التعليم
العالي و البحث العلمي
جامعة مصطفى اسطنبولي معسكر

كلية العلوم الدقيقة
قسم الإعلام الآلي

Support de Cours

Les Systèmes temps réel

Yahyaoui Khadidja

[Les systems temps réel],
[Université Mustapha Stambouli Mascara]. 2022.

Avant-propos

Ce cours présente les principales notions sur les systèmes temps rèel. Il s'adresse aux étudiants de filière informatique aussi bien de spécialité ingénierie des systèmes informatiques qu'au spécialité réseaux et systèmes distribués. Son contenu couvre les différents aspects des systèmes temps réel notamment l'aspect algorithmique pour la conception des applications temps réel.

Il leur permet d'analyser les exigences d'un problème temps-réel, conception de la solution, démonstrations de la correction de la conception proposée, programmation de la solution, validation de la solution, et de concevoir des applications sur un système temps réel.

Nous commençons ce cours par le premier chapitre sur des généralités sur les systèmes temps réel tout en mettant l'accent sur le concept temps réel dans les systèmes d'exploitation. Le second chapitre est dédié à la définition de l'application temps réel, le modèle de tâches périodiques et apériodiques, description approfondie sur les contraintes temporelles ainsi que les différentes approaches d'analyse de l'ordonnançabilité. Les différents algorithmes d'ordonnancement des tâches temps réel périodiques et apériodiques à base de priorité statique et dynamique ont pris un large éventail dans le chapitre trois. Des exemples sur l'ordonnancement des tâches temps réel sont élaborés sous forme des diagrammes de Gantt (graphes temporels) pour chaque algorithme ainsi que des analyses d'ordonnançabilités sont éffectuées sur différents modèles de tâches temps réel.

Le quatriéme chapitre traite le cas du modèle de tâches plus complexe. Des tâches présentent des liens de dépendance par le partage de ressources critiques et des lien de précédence. Le partage de ressource induit le problème d'inversion de priorités des tâches. Des protocoles (algorithmes) sont évoqués pour la résolution de cette problèmatique. Afin d'assimiler ces notions, des exemples sont élaborés pour chaque protocole. Des techniques d'ordonnancement sont adaptées pour ordonnancer un système de tâches avec des liens de précédence. Le support de cours présente un ensemble d'exércices résolus et propose des exercices à résoudre à travers les quels l'étudiant va développer un aspect de réflexion pour la résolution des problèmes complexes en étendant des solutions efficaces et présentant une certaine logique dans leur justesse.

Le cinquiéme chapitre se focalise sur la conception d'application temps réel. Comme outil de conception UML a été adoptée avec des diagrammes dynaqmiques et statiques.

Le support s'achéve par des sujets des examens dans les systèmes temps réel.

Pré-requis

Ce cours nécessite une certaine maitrise de principes fondamentaux pour l'écriture des algorithmiques et les notions de base sur les systèmes d'exploitation I et II, tels que :

- Les composants matériels et logiciels des systèmes d'exploitation
- Les processus et les threads
- L'ordonnancement des tâches et ses différents algorithmes
- La gestion des ressources partagées notamment les ressources à accès exclusif
- Le concept de synchronisation des processus et les outils de synchronisation à savoir les sémaphores d'exclusion mutuelle.
- La conception d'application temps réel.

3. Algorithmes d'ordonnancement des tâches temps réel périodiques et apériodiques

4. Modèle de tâches temps réel dépendantes

5. Conception d'application temps réel

1

CONCEPTS DES SYSTEMES TEMPS REEL

1.1 Introduction

L'objectif de ce chapitre est de décrire les concepts de base des systèmes temps réel avec leur aspect matériel et logiciel. Le chapitre montre La différence entre l'application temps réel et le noyau temps réel .Une classification des systèmes temps réel est présenté.

1.2 Definitions des Systèmes Temps Reel

Les systèmes temps réel sont des systèmes numériques qui permettent l'implantation d'applications où le respect des contraintes temporelles est la principale contrainte à satisfaire. Plusieurs définitions sont dégagées pour définir un système temps réel, nous citons :

- On désigne par le temps réel, toute application mettant en œuvre un système informatique dont le comportement (fonctionnement) est conditionné par l'évolution dynamique de l'état d'un environnement (appelé procédé) qui lui est connecté et dont il doit contrôler le comportement. Le rôle du système informatique est alors de suivre ou de piloter ce procédé en respectant des contraintes temporelles définies dans le cahier des charges de l'application.
- Un Système temps réel est un système dont la correction dépend non seulement de la justesse des calculs (exactitude logique) mais du temps auquel est fournie la réponse (exactitude temporelle).
- Temps réel signifie l'aptitude d'un système d'exploitation de fournir le niveau de service requis au bout d'un temps de réponse borne.

La figure 1.1 représente un système temps réel agissant sur son environnement par son système de contrôle à travers des actionneurs, et récupère des informations sur l'état de l'environnement grâce à des capteurs.

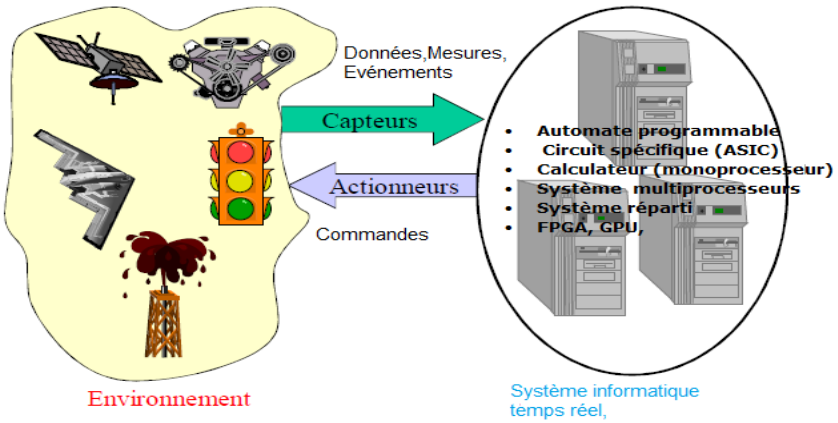


Figure 1.1 *Structure générale d'un système temps réel.*

1.3 Architecture des systèmes temps reel

L'architecture d'un système temps réel tourne autour de deux aspects : l'aspect matériel qui désigne l'ensemble des ressources matérielles (ou physiques) qui peuvent être nécessaire à l'exécution de la couche logicielle : cela inclut donc les processeurs, la mémoire, les réseaux, les dispositifs d'entrées / sorties (qui sont reliées, par exemple, aux capteurs et actionneurs), support de stockage. L'aspect logiciel d'un système temps réel se décompose typiquement d'un exécutif temps réel, qui est une partie de bas niveau faisant le lien avec la couche matérielle et un programme applicatif, qui est une partie de haut niveau, correspondant aux fonctions permettant de contrôler le système temps réel.

1.3.1 *L'exécutif temps réel*

L'exécutif temps réel (micro noyau) est composé :

- d'un noyau temps réel, qui est le cœur de l'exécutif temps réel, pour pouvoir être qualifié de temps réel, il doit répondre à certaines exigences, notamment en fournissant des services de base : L'ordonnancement, les routines de gestion de ressources et les primitives de communication.
- **Des modules ou bibliothèques** venant compléter le noyau temps réel et facilitant la conception d'une application à travers l'apport de routines de gestion de fichiers, de gestion de timers, de gestion de réseaux, etc..

Exécutif temps réel : c'est le système d'exploitation temps réel

- fournir à l'utilisateur un environnement lui permettant de mettre en œuvre facilement son application temps réel,
- constitue d'un ensemble de primitives chargées de fournir cette fonctionnalité
- possédant des fonctionnalités spécifiques pour gérer les contraintes temporelles :
  - ✓ le comportement du système doit être prédictible en termes de temps de réponse
  - ✓ Il doit fournir les outils pour aider à respecter les échéances

1.3.2 Le programme applicatif

Le programme applicatif correspond à la partie logicielle du système qui va exécuter les différentes fonctions nécessaires au contrôle du système, et plus particulièrement du procédé. Le programme applicatif est divisé en entités distinctes appelées tâches qui ont un rôle clé dans les systèmes temps réel et ayant chacune un rôle qui lui est propre figure, comme par exemple : réaliser un calcul, être associé à une alarme, traiter des entrées / sorties, etc

Remarques

1. Un système temps réel peut être mis en place via un système d'exploitation ou non. Sans système d'exploitation, il est possible de disposer d'un exécutif (ou moniteur) temps réel, qui est une sorte de micro-noyau. Cet exécutif est responsable de l'ordonnancement des tâches (ordre dans lequel elles s'exécutent), les tâches s'occupant de leur traitement propre. Cette solution peut être adaptée à des systèmes embarqués dont la fonctionnalité est très spécifique et peu soumise à modification.

2. Dans des systèmes plus importants, il est agréable de disposer des facilités proposées par un système d'exploitation, tels qu'une interface utilisateur ou la gestion de fichiers, par exemple.

3. Un système d'exploitation doit par contre être spécifiquement conçu pour le temps réel, capable d'exécuter plusieurs tâches de manière concurrente, alternant entre les différentes tâches.

4. Il est en effet impossible de faire fonctionner une application temps réel strict sur un système d'exploitation standard. Ceci est notamment en partie dû aux facteurs d'imprédictibilité qui caractérisent les systèmes temps réel.

La figure 1.2 illustre la partie logicielle d'un système temps réel

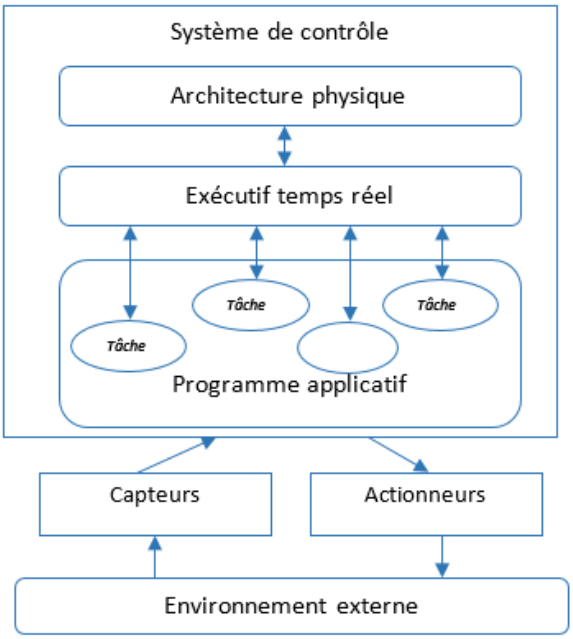


Figure 1.2 *Structure architecturale d'un système temps réel*

Un schéma blocs présentée par la figure 1.3 illustre l'architecture des exécutifs temps réel avec deux interfaces séparées, l'une spécialisée dans le temps réel et l'autre générique où Les applications temps réel font alors appel à la partie temps réel du noyau.

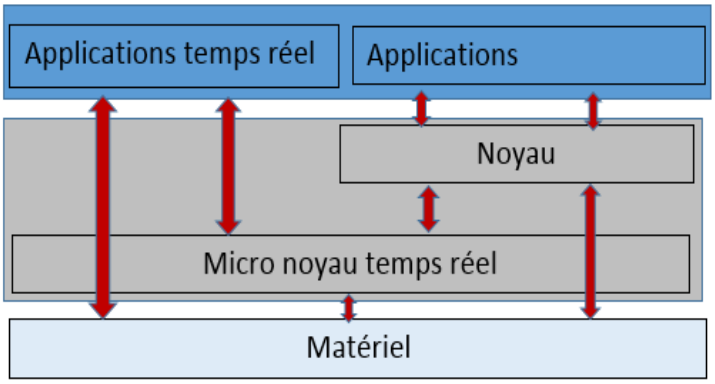


Figure 1.3 *Architecture des exécutifs temps réel spécialisée dans le temps réel et générique*

1.4 Exemples des systèmes temps reel

Les exemples sont issuss du site suivant:
http://en.wikipedia.org/wiki/List_of_real-time_operating_systems

Comme exemples des systems généralisés nous citons: VxWorks, QNX, ChorusOS (VirtualLogix) pSOS+, VRTX, WINDOWS CE, XENOMAI.

Comme exemples des Systèmes d'exploitation temps réel spécialisés nous citons :
OSEK/VDX , IPhone, Android, Windows
Le tableau 1.1 résume ces exemples avec leeurs motivations

	Exemples	Motivations
Systèmes d'exploitation temps réel généralisés	VxWorks, QNX, ChorusOS (VirtualLogix) pSOS+, VRTX, WINDOWS CE, XENOMAI.	- la plupart des applications ont besoin de services "généraux" : accès à des fichiers, accès au réseau. - Minimisation des phases de développement : minimiser les coûts et réutilisation de composants - La portabilité des applications, indépendance vis-à-vis de la plateforme d'exécution.
Systèmes d'exploitation temps réel spécialisé	OSEK/VDX , IPHONE, ANDROID, WINDOWS,.	- Performances : adéquation au matériel utilisé - Utilisations très spécifiques - Marché captif : automobile, avionique et télécommunication

Tableau 1.1 ***Exemples de systèmes temps réel généralisés et spécialisés***

1.5 Services du système d'exploitation temps réel

Les fonctionnalités résumées dans le tableau 2.2 se présentent comme des services offerts par les systèmes temps reel.

Services	Fonctionnalités des systèmes temps réel
Gestion des threads	Implémentent généralement l'unité d'activité Découpage en tâches ou en processus concurrents
	Peuvent exister sous forme de ✓ threads périodiques ✓ threads apériodiques (sporadiques) ✓ threads serveurs (pour implémenter des serveurs de tâches apériodiques)
	états : ✓ dormant (en attente de la requête de réveil) ✓ prêt (en attente de la CPU) ✓ en cours ✓suspendu (en attente d'une ressource) ✓ terminé
le noyau	Exécute au bénéfice de l'utilisateur les opérations qui nécessitent des privilèges spéciaux ✓ appels systèmes ✓ ordonnanceur (+ gestion du temps) ✓ gestion des interruptions matérielles et logicielles
	souvent implémenté sous la forme d'un « micro-noyau »
Gestion du temps	Horloges, Timers : il faut pouvoir manipuler **le temps concret (horloge)** de plusieurs façons: soit en définissant la date à laquelle une action doit être commencée soit en définissant la date à laquelle une action doit être finie.
Communication et synchronisation	files de messages, mémoire partagée sémaphores, mutex, variables conditionnelles
Notifications Asynchrones	signaux, événements
Gestion de la mémoire	mémoire virtuellle , verrouillage de la mémoire protection de la mémoire
Entrées-sorties, Gestion du réseau	

Tableau 1.2 *Fonctionnalités des systèmes temps réel*

1.6. Classification des systèmes temps réel

Dans le contexte des systèmes temps réel, les données ont une validité définie à la fois par le domaine de valeurs admises et par la durée de validité (temporelle) de celles-ci, qui dépend naturellement de l'échéance. Les données ont donc une durée d'existence limitée. Une classification répandue des systèmes temps réel en fonction des conséquences que peut avoir le non respect d'une échéance est donnée comme suit :

1.6.1. Systèmes temps réel a contraintes strictes (TRCS)

Ces systèmes sont composés de tâches soumises à des contraintes temporelles strictes. Chaque occurrence de tâche se voit associer une échéance qu'il serait inadmissible de ne pas respecter.Le respect des dates de fin d'exécution au plus tard est obligatoire. Une faute temporelle pourrait être coûteuse humainement et/ou financièrement. Nous pouvons les rencontrer dans des systèmes du domaine de l'embarqué comme exemples : Contrôle aérien, l'avionique, Contrôle d'une centrale nucléaire, l'automobile avec les systèmes ABS : Anti blocking system (ABS) , Transactions en bourses, etc..

1.6.2 Systèmes temps réel a contraintes relatives ou souples (TRCRS)

Les Systèmes Temps Réel à Contraintes Souples dont la mesure d'efficacité en fonction de la qualité de service qu'ils offrent, en termes de probabilité d'erreur. Une analyse des systèmes temps réel s'opère généralement sur des temps de réponse moyen. Le respect des dates de fin d'exécution au plus tard est souhaitable. Comme exemples de ces systèmes :les applications de type multimédia telles que la téléphonie mobile ou la vidéo (Vidéo conférence). Si quelques images ne sont pas affichées, cela ne met en péril le fonctionnement correct de l'ensemble du système.

1.6.3 Systèmes temps réel à contraintes mixtes (TRCMS)

Cette classe de systèmes regroupant la plupart des systèmes temps réel puisque les systèmes TRCM sont composés de tâches TRCS et de tâches TRCR. La problématique dans ce contexte est d'éliminer toute possibilité de faute temporelle pour les tâches TRCS tout en minimisant les fautes temporelles de tâches TRCM. Les Systèmes Temps Réel à Contraintes Mixtes regroupent des applications temps réel composées de tâches dont un sous ensemble doit impérativement respecter des contraintes temporelles, à l'inverse des autres tâches dont le critère d'évaluation cherchera à minimiser les fautes temporelles. Ces systèmes regroupent la plupart des systèmes temps réel actuels.

Une classification des systèmes temps réel est présentée par la figure 1.4.

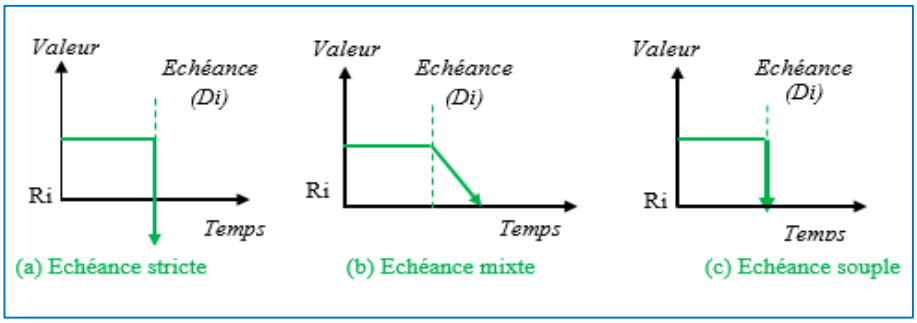


Figure I.4. *Classification des systèmes temps réel*

En Consequences
=> Dans le cas des systèmes temps réel durs, on cherchera à être **prévisible**, **déterministe** et **fiable**
=> Utilisation de techniques mathématiques (ordonnancement, évaluation des pires cas…)
=> Dans le cas des systèmes temps réel souples, on cherchera à minimiser la probabilité de rater une échéance plusieurs fois de suite...

Prédictibilité (predictability) :
Les performances de l'application doivent être définies dans tous les cas possibles de façon à assurer le respect des contraintes de temps. On parle de pire cas.

Déterminisme (determinism) :
Il n'y a aucune incertitude sur le comportement du système : pour un contexte donne le comportement est toujours le même.

Fiabilité (reliability):
Capacité d'un système à réaliser et maintenir ses fonctionnalités dans des conditions normales d'utilisation. En temps réel, la fiabilité concerne. le respect des contraintes temps réel. On peut également vouloir que le système reste fiable même si certaines pannes sont apparues, on parle alors de tolérance aux fautes.

1.7 Présentation des systèmes temps réel embarqué

Un système embarqué (Embedded system) ou système enfoui est un système informatique dans lequel le processeur/calculateur est englobé dans un système plus large et où le logiciel est entièrement dédié à une application donnée. Exemple : sonde

spatiale, terminal GSM, carte à puce. Tout ce qui n'est pas perçu comme un ordinateur : équipements industriels ou scientifiques,

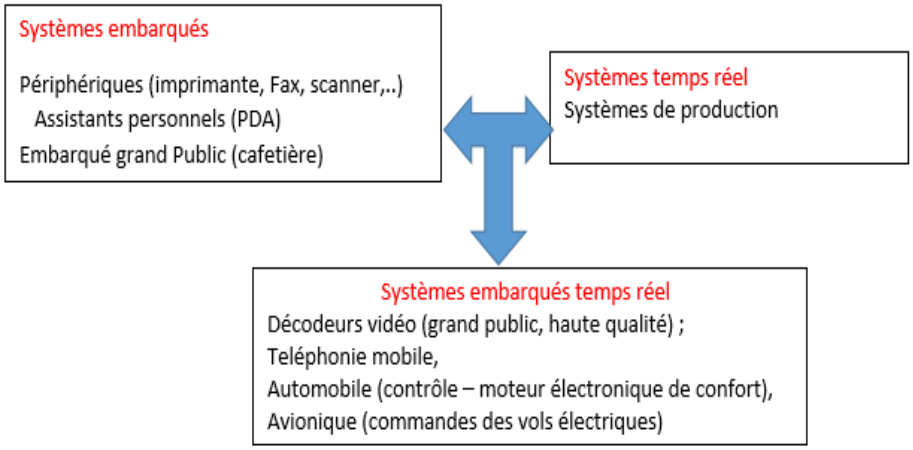


Figure 1.5. *Systèmes temps réel et embarqués*

1.7.1 Spécificités des systèmes temps réel embarqués

En plus de ceux fournis par les Systèmes temps réel, les systèmes temps réel embarqués devront offrir les services suivants :

Gestion de l'énergie	**A**ssurer une faible consommation : les algorithmes utilisés par les Systèmes temps réel ne pourront pas toujours être implantés par les systèmes temps embarqués.
Gestion de la mémoire	**Au niveau temporel :** allocation des blocs mémoire en temps prédictible, **Au niveau espace d'adressage :** prise en compte des contraintes de faible encombrement.
Gestion des pannes	Des mécanismes de reprise en cas d'erreur, des possibilités de fonctionnement en mode dégradé,
Adaptabilité	Adaptabilité à la variation de configuration logicielle (HAL ou BSP)

Tableau 1.3 *Services des systems temps réel embarqués*

1.7.2 Exemple d'applications temps réel embarquées :

Exemple d'application temps réel et embarquée présentant des tâches critiques et non critiques : Automobile

Tâches critiques	Tâches non critiques
Contrôle airbag,	Contrôle climatisation
contrôle ABS Contrôle pneumatique	Radio

1.8 Caractéristiques des applications temps réel

Les applications temps réel sont caractérisées par :

- Exemples de grandeurs des contraintes temporelles
  - **La milliseconde** pour les systèmes de radar
  - **La seconde** pour les systèmes de visualisation humaine
  - **Quelques heures** pour le contrôle de production impliquant des réactions chimiques 24 heures pour les prévisions météo.
  - **Plusieurs mois ou années** pour les systèmes de navigation de sonde spatiale.

- **Larges, complexes et fiables** : Un système temps réel interagit avec un environnement extérieur souvent complexe et en évolution, il doit pouvoir interagir avec différents types d'éléments matériels, il doit respecter des échéances temporelle en garantissant une fiabilité importante.

1.9 Domaines des applications temps réel

Les applications temps réel couvrent un très grand domaine :

- Contrôle de procédés : suivi en temps réel des signaux d'un laminoir accessibles dans une mémoire commune, durant une fenêtre temporelle, pendant laquelle les signaux peuvent être lus.

- Systèmes de conduite, fixe ou mobile (embarqué) de véhicules : bus, métro, voiture, train, avion et navette.

- Transactions bancaires et boursières avec des dates de valeurs définies et prises en compte.

- Multimédia, Vidéo conférence : Sur un réseau local, Le système numérise l'image, puis la séquence. Il faut qu'il traite 30 images/sec pour avoir une scène vidéo acceptable. Plusieurs opérations vont s'enchaîner dans le temps : numérisation, compression, transmission. La durée de ces opérations est définie comme un temps de latence du système. La voix doit également être numérisée, compressée et transmise. Il faut qu'il y ait synchronisation

son/image. Un retard léger est acceptable. S'il est trop long, la scène devient incompréhensible (temps réel mou, ou échéances souples, relatives).

- **Contrôle des systèmes automatisés de production** : La production automatisée est assurée par la bonne conduite, le réglage et le contrôle en temps réel ainsi que la maintenance des premiers niveaux par l'alerte des services compétents. La dimension du système est de comprendre les procédés physiques qui sont automatisés, d'imaginer des nouvelles solutions tant au niveau supervision que contrôle-commande ou communication **Application robotique»** Un robot doit prendre des objets qui défilent sur un tapis Le robot dispose d'une "fenêtre temporelle" pour agir. S'il agit trop tard, il manquera l'objet, s'il agit trop tôt, il va bloquer l'objet (et les objets suivants).

- **Systèmes de surveillance** : traçabilité des événements et recherche en ligne des relations de causalité.

- **Radiotéléphonie GSM** : transmission d'un flux régulier des échantillons de parole (577 µs de parole toutes les 4,6 ms) à des abonnés mobiles. La mobilité modifie les distances et les chemins de communication.

1.9. Exemples d'une application temps réel : Etude de cas :

Cas 1 : Nous considérons un système de contrôle de vol simplifié sur la figure 1.6 ci-dessous. Ce système contrôle :

– l'attitude
– la trajectoire
– la vitesse d'un avion

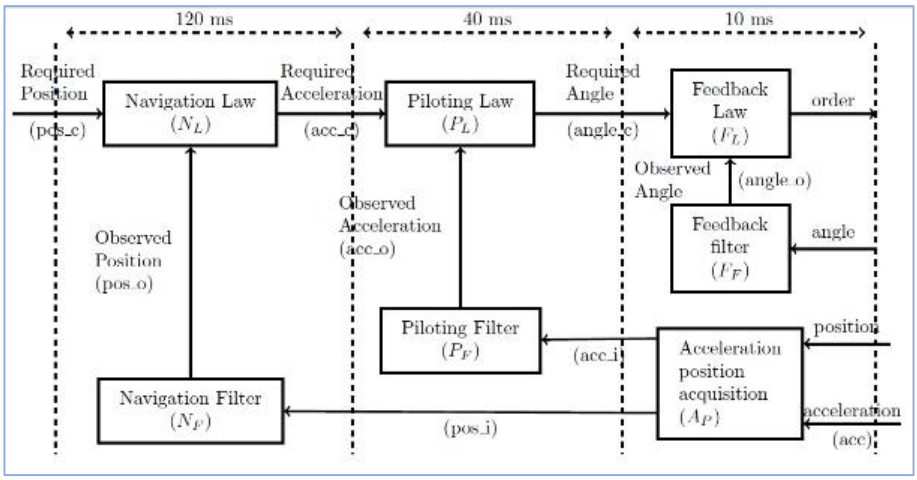


Figure 1.6 *Exemple d'application temps réel*
(Système de contrôle de vol aérien)

Le système de contrôle se compose de 7 tâches qui s'exécutent à plusieurs reprises à un rythme périodique. Elles Ces tâches présentent les contraintes temporelles suivantes :

T : Période d'exécution
C : Durée d'exécution maximale
R : Date de réveil
D : Délai critique

- Le sous-système est le plus rapide et s'exécute à10ms, il acquiert l'état du système (angles, la position, l'accélération) et calcule la loi de retour du système. La commande est alors envoyée aux de commande de vol
- Le sous-système intermédiaire est la boucle de pilotage, il s'exécute à 40 ms et détermine l'accélération à appliquer.
- Le sous-système le plus lent est la boucle de navigation, il s'exécute à 120ms et détermine la position à atteindre.
- La position souhaitée de l'avion est acquis via le pilote

De cette description, nous aurons le modèle de tâches détaillée comme suit :

Tâches	T (ms)	C(ms)	r(ms)	D(ms)
NL	120	20	0	120
NF	120	10	0	120
PL	40	5	0	40
PF	40	5	0	40
FL	10	2	0	10
FF	10	1	0	10
AP	10	1	0	10

Table 2.3 *Valeurs des paramètres temporels*

Cas2 :

Comme exemple de système temps réel, nous avons un système de détection d'intrusion (IDS) temps réel dont le fonctionnement se résume en plusieurs tâches :

- Collection des données brutes via des capteurs (senseur) et les mettre en forme des évènements à l'analyseur.
- Analyse du trafic Internet ;
- Identification des IPs impliquées dans l'attaque ;
- Reconfiguration d'équipement (firewall) ;
- Notification visuelle de l'alerte : affichage de l'alerte dans une ou plusieurs consoles de management.

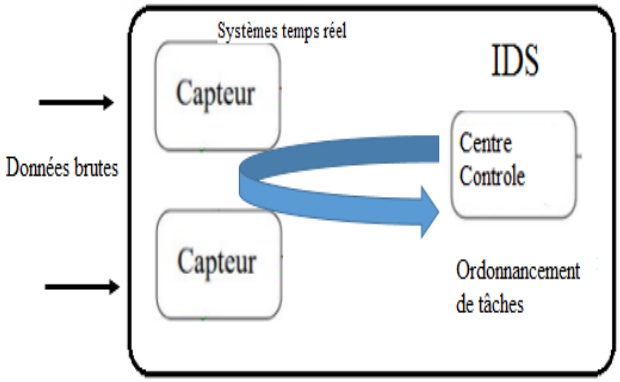


Figure 1.7 *Exemple système temps réel : Système de détection d'intrusion*

La fréquence d'utilisation d'un système de détection d'intrusion peut s'effectuer :

- Périodiquement : l'analyse se fait périodiquement à la recherche de nouvelles violation ;
- En temps réel : détection de l'attaque au moment où elle se produit, l'analyse se fait de manière continue. Limitation des dégâts en temps réel en appliquant des contremesures.

Chaque tâche du système présente des caractéristiques temporelles telles que période, temps de calcul, date de réveil, date de fin, deadline.

Cas 2 : Conduite d'un réacteur chimique.

Dans une usine de produits chimiques, la synthèse d'un produit C à partir de 2 produits A et B s'effectue dans un réacteur.

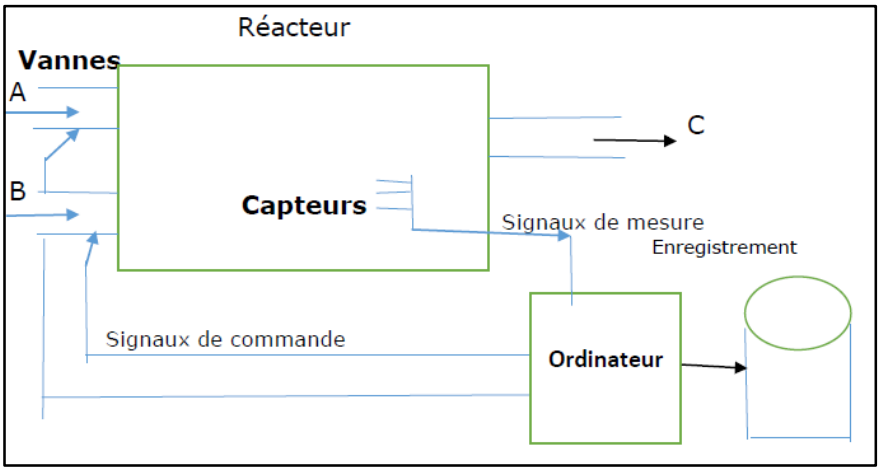


Figure 1.8 *Exemple d'un système temps réel : Conduite d'un réacteur chimique.*

Tâches principales de l'application temps réel :

a. Action sur les organes externes : Lecture des capteurs et commande des vannes ; (tâche périodique)
b. Prise en compte du temps physique : déclenchement périodique du cycle de traitement ; (tâche périodique)
   Réaction aux événements extérieurs : arrêt d'urgence ; (tâche apériodique)
c. Gestion d'information : conservation et entretien du fichier journal ; (tâche périodique)

2

Modèle de tâches temps réel

[Systèmes temps rée].
[Université Mustapha Stambouli-Mascara-].2022.

2.1 Introduction

Après avoir définir le concept de temps réel dans les systèmes, donner une classification des systèmes temps réel. Ce cours, est consacré aux modèles de tâches temps réel. Ces tâches sont de type périodique et apériodique. Les applications temps réel se présentent comme un ensemble des tâches indépendantes, leurs exécutions s'appuient sur des algorithmes d'ordonnancement avec des tests d'acceptabilité et d'optimalité.

2.2 Définitions application temps réel

- Une application temps-réel effectue des fonctions de contrôle et de pilotage. Chaque fonctionnalité est assumée par une tâche temps-réel.

- Application temps réel = Ensemble de tâches qui coopèrent en accédant à des ressources communes, en s'échangeant des messages tout en respectant différentes contraintes : Cohérence des ressources partagées, Tolérance aux fautes, Sécurité et Temps de réponse.

> Système temps réel vs Application temps réel : Les deux notions sont souvent confondues

La figure 2.1 illustre une application temps réel comme suit :

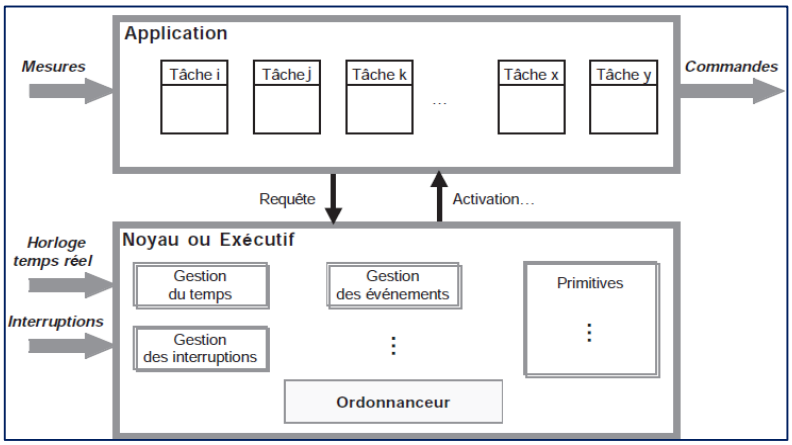


Figure 2.1 *Architecture de l'application : tâches et noyau temps réel*.

2.3 Exemple d'application temps réel

Soit un système de visualisation intégré dans un véhicule automobile. Cette application temps réel (système) est constituée d'un ensemble de tâches. Ces tâches permettent :

- la lecture de la vitesse du véhicule pendant une durée de 2 millisecondes toutes les 10 millisecondes, de la température dans l'habitacle du véhicule pendant une durée d'1 millisecondes toute les 10 millisecondes et la position du GPS du véhicule pendant une durée de 4 millisecondes toutes les 40 millisecondes. Ces différentes lectures sont effectuées respectivement par 3 capteurs : Capteur_1, Capteur_2 et Capteur_3.
- L'affichage pendant une durée de 2 millisecondes toutes les 12 millisecondes qui produit un résumé des informations produites des tâches des différentes lectures ci-dessus. Un autre affichage de la carte routière est réalisé pendant une durée 2 millisecondes selon la demande de l'utilisateur toutes les 6 millisecondes. Après une multitude de mesures des programmes implantés par les tâches sur la cible, nous aurons les valeurs des paramètres de l'application (système) temps réel comme suit :

Tâches	Charge (millisecondes)	Période (millisecondes)
La tâche CAPTEUR_1	2	10
La tâche CAPTEUR_2	1	10
La tâche CAPTEUR_3	4	40
La tâche AFFICHAGE_1	2	12
La tâche AFFICHAGE_2	2	6

Tableau 2.1 : *Contraintes temporelles et valeurs*

2.4 Tâche temps réel

Les tâches temps réel se présentent comme :

- Tâche / activité / processus : Une unité d'exécution ou bien un ensemble d'instructions destinées être exécutées sur un processeur.
- Les tâches sont des entités génériques qui ont des contraintes temporelles et des relations de synchronisation et de communication.
- Certaines fonctions sont critiques et ne peuvent pas être retardées, c.-à-d. que les tâches correspondantes doivent impérativement avoir terminé leur traitement sous un délai donné, à partir de leur date de réveil.
- tâches temps réel modélise le fonctionnement des applications temps réel.

- La tâche peut être activée suite à un évènement d'horloge, une interruption externe ou bien générée par d'autres tâches.

La figure 2.1 est une representation schématique d'une tâche temporelle

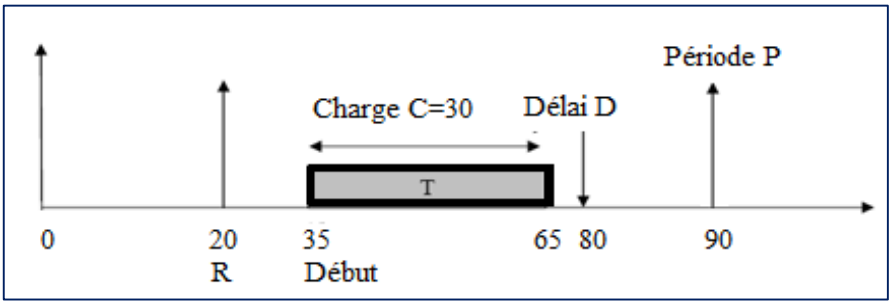


Figure 2.1 *Propriétés temporelles d'une tâche temps réel.*

Chaque tâche a un temps d'arrivée R dans lequel la tâche devient ordonnançable. Après un certain temps la tâche peut commencer l'exécution et se terminer après l'occupation de C unités de temps processeur. Un délai critique est associé à la tâche, donc chaque tâche doit terminer l'exécution avant ce délai D.

2.4.1 Contraintes d'une tâche temps réel

La tâche temps réel est caractérisée par des paramètres temporels. Ces paramètres sont présentés dans la figure 2.2.

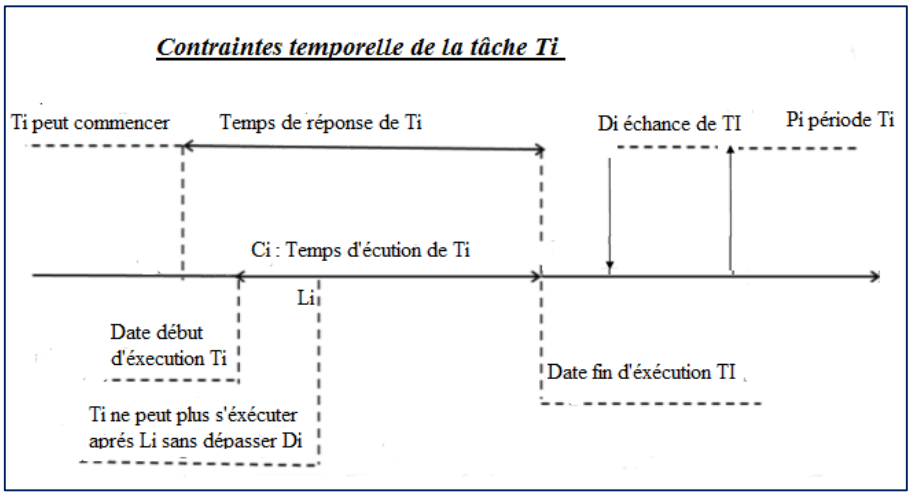


Figure 2.2 *Les principaux paramètres temporels de la tâche Ti*

R_i (Ready time ou Release time)	C'est la date à laquelle la tâche Ti peut commencer son exécution appelé aussi la date au plus tôt.
C_i (computing time)	C'est la durée d'exécution de la tâche Ti. Ce paramètre est considéré dans la majorité des travaux sur l'ordonnancement temps réel comme **pire temps d'exécution de tâche (WCET : pour Worst Case Execution Time)** sur le processeur auquel elle est affectée.
D_i (deadline)	C'est son échéance appelée aussi la date au plus tard ; elle représente l'instant auquel l'exécution de la tâche doit être terminée dont le dépassement entraîne une faute temporelle.
L_i (laxity)	*C*'est laxité d'une tâche Ti, qui représente le temps restant avant l'occurrence de sa date de démarrage ou de reprise au plus tard.
Pr_i(Priorité)	Une tâche peut avoir ou non une priorité (qui définit son poids ou son importance par rapport aux autres tâches du système), dont la valeur dépend de l'algorithme d'ordonnancement sous-jacent.
U_i (Taux d'occupation)	Il s'agit de l'utilisation du processeur. Elle est déduite à partir du temps de computation (Ci) et de la période (Pi) de la tâche Ti. $$U_i = \frac{Ci}{Pi}$$
L_i (laxité)	Laxité représente le temps restant avant l'occurrence de la date de démarrage de la tâche Ti ou de reprise au plus tard.

2.4.2 Etats d'une tâche

Chaque tâche va correspondre aux exécutions sur le processeur d'une séquence d'opérations données elle fournit une partie des services de l'application. La figure II.3 illustre les différentes transitions entre les états d'une tâche.

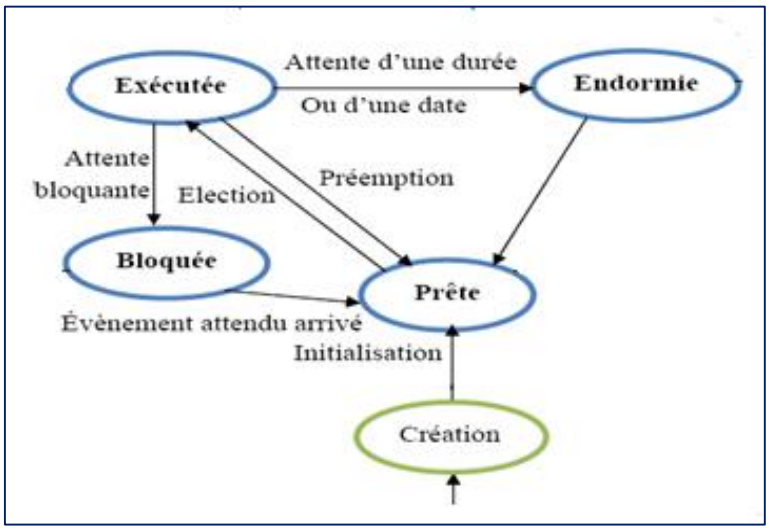


Figure 2.3 *Différents états d'une tâche*.

Exécutée	l'instance de tâche est en cours d'exécution sur une unité de calcul qui lui est allouée.
Bloquée	l'instance de tâche est en attente d'une ressource, un message ou un signal de synchronisation.
Prête	l'instance d'une tâche attend d'être élue pour exécution.
En dormie	l'instance de tâche n'a pas de requêtes en cours, elle est stoppée.

2.4.3 Types des tâches

A. Tâches périodiques

Les tâches périodiques sont caractérisées par :

- L'activation de la tâche survient à des intervalles **des temps réguliers** P_i.
- Chacune de ces exécutions est appelée instance.
- Constitue la majorité des applications temps réel : par exemple de mesure de la température toutes les secondes.

Comme exemple de tâche périodique : lecture de capteurs. La figure 2.4 présente la périodicité de la tâche.

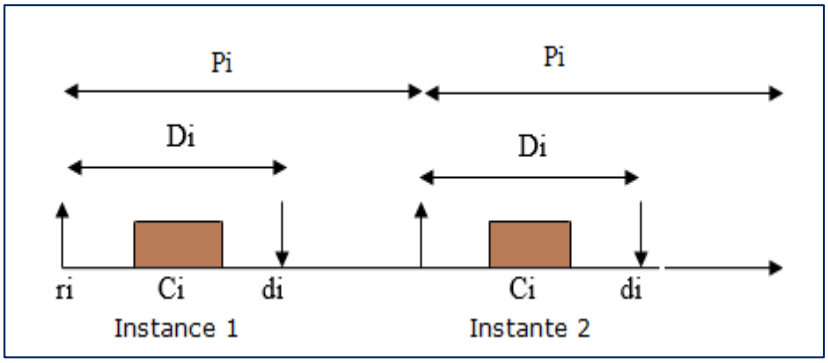


Figure 2.4 *Tâche temps réel périodique (avec deux instances)*

Tâche dite bien formée : si $0 \leq C_i \leq D_i \leq P_i$.
Tâche à échéance sur requête si *Di = Pi.*
Tâches synchrones : des tâches périodiques avec des dates d'activation des premières instances identiques, ou à départ simultané.

B. Tâche apériodique

Les tâches apériodiques sont des tâches en temps réel qui sont activées de **manière irrégulière** à des taux éventuellement illimités. La contrainte de la tâche temps réel est généralement l'échéance (Di). Comme exemple de tâche apériodique : alarme Figure 2.5 illustre la tâche apériodique

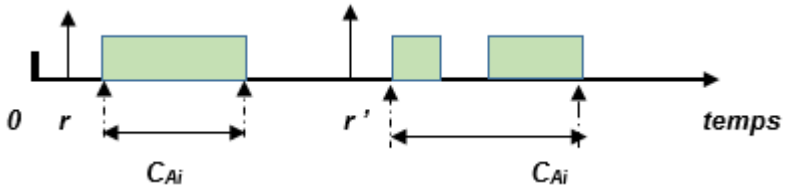


Figure 2.5 *Représentation de l'exécution d'une tâche apériodique (2 instances de la tâche) (r, r': dates de réveil, Cai : temps de computation de la charge)*

C. Tâche sporadique

Les tâches sporadiques sont un cas particulier des tâches périodiques. Deux caractéristiques démarquent les sporadiques des périodiques :

- On ne connaît d'une part pas a priori leur date de première activation, simplement parce qu'il est rare de pouvoir prévoir des dates de déclenchement des alarmes avant la mise en marche d'un système.
- Pour les tâches périodiques, Pi représente une durée fixe séparant deux activations successives ; pour les sporadiques, c'est une durée minimale séparant deux occurrences successives de la tâche.

2.5 Priorité des tâches

Dans un système temps réel, L'ordonnanceur assigne des priorités aux différentes tâches ou aux travaux de ces tâches. Il existe donc deux types d'assignations des priorités :priorité fixe et priorité dynamique Une tâche peut avoir ou non une priorité (qui définit son poids ou son importance par rapport aux autres tâches du système), dont la valeur dépend de l'algorithme d'ordonnancement sous-jacent. Nous distinguons deux types de priorités :

Priorité fixe où une tâche est affectée d'une priorité. Toutes les instances de cette tâche hériteront de cette priorité. C'est-à-dire que la priorité de chaque tâche est fixée lors de la conception du système et demeure invariable tout au long de sa vie.
***Priorité dynamique*.** On parle de ce type de priorité, quand la priorité n'est plus affectée à une tâche mais à ses instances, c'est-à-dire que la priorité d'une instance peut évoluer au cours du temps.

2.6 Préemption des tâches

Une tâche préemptible peut être temporairement suspendue et reprise ultérieurement. Si une tâche est non préemptible est interrompue, son exécution doit être reprise de nouveau depuis le début. Une tâche i interrompue par la tâche j, soit lorsque cette dernière a besoin d'une ressource, telle que le processeur, qui utilisait la tâche i et la tâche j plus prioritaire que la tâche i, soit la tâche i ne pouvait pas continuer son exécution par manque de ressources et que la tâche j le pouvait. Dans tous les cas, déterminer qu'une tâche est préemptible, est principalement lié à la nature de l'application. La figure 2.6 met en évidence les tâches avec préemption.

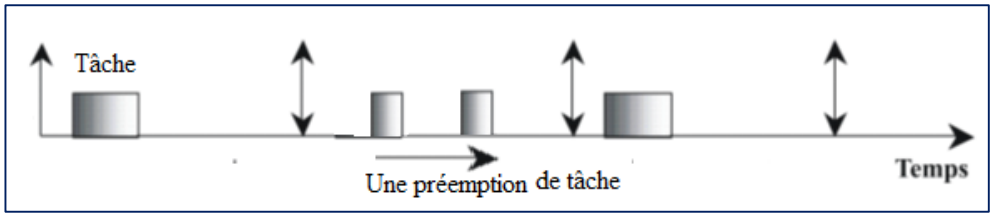


Figure 2.6 *Préemption de tâche(Tâche a subi une préemption (interruption) à sa deuxième instance)*

2.7 Ordonnancement des tâches dans les systèmes temps réel

L'ordonnancement définit la manière dont est partagé le (ou les) processeur(s) disponible(s) pour l'exécution des processus. Des mécanismes particuliers sont nécessaires pour réaliser cet ordonnancement. Il s'agit des files de processus, gestion de priorités, des préemptions, attente passives (le blocage et le déblocage). L'ordonnancement peut être réalisé entièrement par le noyau ou par une tâche spécialisée (scheduler).

Les problèmes d'ordonnancement de recherche opérationnelle sont différents de ceux de temps réel, le but étant pour la recherche opérationnelle de minimiser (hors ligne) le temps de réponse et pour le Temps réel de satisfaire (en ligne) des échéances.

Nous énumérons trois types d'ordonnancements :

1. **Ordonnancements de tâches périodiques :** comme Ordonnancement à base de table, Ordonnancements préemptif à priorité fixe et Ordonnancements préemptif à priorité dynamique.

2. **Ordonnancements de tâches apériodiques** : comme Serveur de tâches apériodique.
3. **Ordonnancement sous la synchronisation des tâches :** comme héritage de priorité et Priorité plafonnée.

Pour les trois types de modèles de tâches (périodiques, apériodiques et avec dépendance (syhronisation) , nous détaillons leurs algorithmes d'ordonnancement dans les chapitres 3et 4.

2.7.1 Objectif de l'ordonnancement temps réel

Comme un but principal de l'ordonnancement : la planification des requêtes (tâches) sous le respect des contraintes de temps associées à l'application et aux tâches**.** Chaque tâche est caractérisée par les contraintes temporelles (fonctionnement nominal), il faut qu'elle respecte ces contraintes par l'utilisation efficace des ressources disponibles (Ressource CPU, ressource accès ou données et ressources de communications en sont des exemples). Le non respect de ces contraintes (fonctionnement anormal), engage parfois des conséquences sévères, et le plus souvent les conflits survenant entre les tâches lors de leur exécution .Ces contraintes peuvent être des conflits sur:

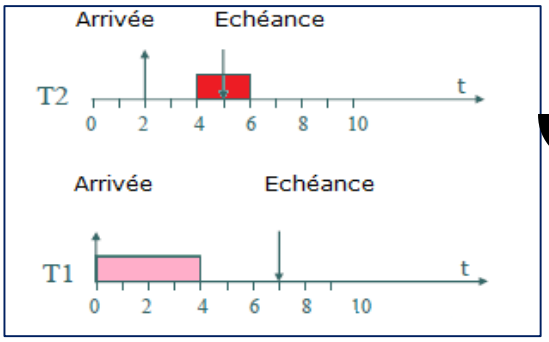
- Les processeurs (plusieurs tâches peuvent accéder simultanément au même processeur),
- Les autres ressources (plusieurs tâches peuvent demander l'accès d'une ressource ou donnée non partagée telle que une unité d'entrée/ sortie).

Ces conflits peuvent être traités en faisant appel à ***l'ordonnanceur*** ce dernier permet l'exécution de l'ensemble des tâches de l'application sur un ensemble des ressources, en tenant compte des contraintes bien spécifiées.

Exemple 1

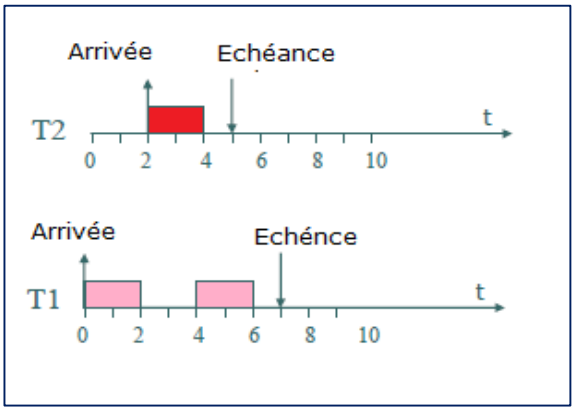
Tâche T1 : arrivée en 0, durée 4, échéance 7
Tâche T2 : arrivée en 2, durée 2, échéance 5

Ordonnancement _1 est réalisé selon la politique : premier arrivé, premier servi.



Ordonnancement_1 :
Non valide : T2 dépasse son échéance

Ordonnancement _2 est réalisé selon la politique : de priorité, T2 plus prioritaire que T1.



Ordonnancement_2: valide (respect des contraintes temporelles)

2.7.2 Comment définir un problème d'ordonnancement dans un système temps rèel ?

Il est important de connaître le **modèle du système**, **la nature de tâche** et **l'objectif de l'algorithme de l'ordonnancement** afin de définir un problème d'ordonnancement dans un système temps réel.

- **Le modèle de système :** qui décrit l'architecture matérielle de système proposée (machine, réseaux ….etc.) et les contraintes qui lui sont associées (nombre de processeurs dans chaque machine mono ou multiprocesseurs), la mémoire disponible, les contraintes lies aux réseaux (architecture, accès médium, etc.).
- **la nature des tâches à ordonnancer :** on décrit à ce niveau l'architecture fonctionnelle de l'application et les contraintes associées qui représentent les besoins. Ces contraintes peuvent être des contraintes temporelles. Les contraintes les plus utilisés sont présentées dans la figure 2.2.

- **Objectif d'exécution**
  Un algorithme hors-ligne (off-line scheduling) : dans ce cas, une séquence ordonnancement valide (à répéter à l'infini) est générée par l'algorithme choisi. Cette séquence est implémentée dans une table qui sera consultée par un ordonnanceur simplifié que l'on appel séquenceur. Cette approche conduit à un surcoût minimal à l'exécution mais nécessite que la majorité des caractéristiques des tâches dans le système soient connues à l'avance. Elle est inflexible et ne s'adopte par aux changements de l'environnement ou à l'environnement dans le comportement n'est pas complètement prévisible. Par contre, cette approche présente un grand avantage essentiellement lorsqu'il s'agit de systèmes dits temps critique où la prédiction de comportement du système est l'objectif les plus importants à atteindre. L'ordonnancement hors ligne est souvent le seul moyen d'offrir une prédiction dans le système complexe (s'avoir à l'avance si le système va respecter ou non les contraintes imposées). Figure 2.7 illustre ce type de séquencement.

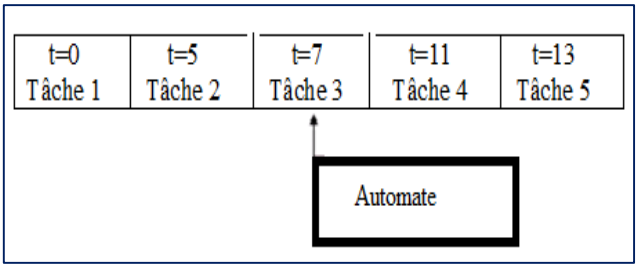


Figure 2.7 *Séquence construite hors ligne.*

Un algorithme en-ligne (on-line scheduling) : l'ordonnancement des tâches est calculé progressivement au fur et à mesure que les tâches arrivent dans le système. Ainsi l'ordonnanceur est capable, à tout instant de l'exécution de l'application, de choisir la prochaine tâche à ordonnancer et utiliser comme information les paramètres temporelles des tâches déclenchées à cet instant, ce qui rend flexible et capable de s'adapter facilement aux changements qui surviennent au cours de la vie du système. Ainsi généralement au moment où une nouvelle tâche arrive, ses caractéristiques deviennent connues et une décision doit être prise sur l'acceptation ou non de cette tâche. Figure II.8 illustre cet ordonnancement en ligne.

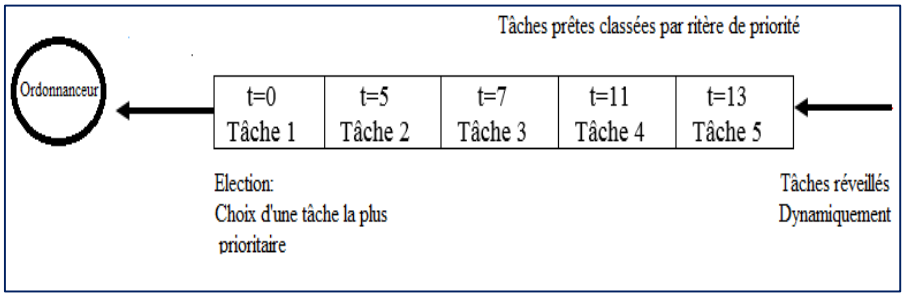


Figure 2.8 *Ordonnanceur en ligne.*

2.8. Etude d'ordonnançabilité

Dans un ordonnanceur temps réel, si le système arrive à ordonnancer toutes ses tâches en respectant leurs échéances, nous dirons de celui-ci qu'il est ordonnançable**.**
Ordonnancement faisable : On dit d'un ordonnancement qu'il est faisable si chaque tâche se termine avant son échéance dans cet ordonnancement. Un système est ordonnançable s'il existe un ordonnancement faisable pour ce système.
Ordonnancement optimale : Une règle d'assignation de priorité R désigne la façon dont la priorité de sélectionner un travail par rapport à un autre est définie. On dit que R est optimale si, lorsqu'un système est faisable, il est ordonnançable étant donné la règle R.

2.8.1 Approches d'analyse de l'ordonnancabilité

Pour avoir une certaine prédictibilité et pouvoir s'assurer du respect des contraintes temporelles, en particulier quand il s'agit d'un système temps réel strict, une analyse de l'ordonnancement doit être effectuée.
En considérant les tâches, leurs contraintes, les processeurs dont dispose l'architecture ainsi qu'un algorithme d'ordonnancement, le but est de déterminer s'il existe un ordonnancement, qu'on obtient avec cet algorithme, respectant toutes les contraintes. Trois principales techniques d'analyse sont employées dans l'analyse de

l'ordonnançabilité. Approche analytique, approche par simulation et approche par modèle checking.

A. Approche analytique :

Les principales techniques d'analyse sont :
A.1 Analyse de l'utilisation processeur (Processor Utilization Analysis)
A.2. Analyse de la demande processeur (Processor Demand Analysis)
A.3. Analyse des temps de réponse (Reponse Time Analysis)

A.1 Analyse de l'utilisation processeur (Processor Utilization Analysis)

Le facteur d'utilisation U est la fraction de temps que le processeur passe à exécuter n tâches. Il se définit par :

$$U = \sum_{i=1}^{i=n} Ci / Pi$$

Avec *Ci :* la charge de la tâche i, *Pi :* période de la tâche i

Dans certains cas particuliers, le facteur d'utilisation du processeur permet de conclure l'ordonnançabilité d'une configuration de tâches après vérification de certaines conditions de faisabilité.

A.2 Analyse de la demande processeur (Processor Demand Analysis)

Cette technique repose sur le calcul de la demande cumulée des exécutions des tâches réveillées et terminées dans un intervalle de temps (Demand Bound Function). Cette approche générale permet de construire des tests d'ordonnancabilité en limitant l'étude d'ordonnançabilité à quelques intervalles d'une période d'activité du processeur.

A.3.Analyse des temps de réponse (Reponse Time Analysis)

Cette technique consiste à calculer les pires temps de réponse des tâches et à les comparer avec leurs délais critiques. Une tâche est ordonnançable si, et seulement si, son pire temps de réponse est inférieur ou égal à son délai critique. Dans le cadre de l'ordonnancement à priorité fixe, cette méthode est souvent désignée sous le nom d'analyse RTA (Response Time Analysis) .Ce concept sera détaillé dans le chapitre 3)

B. Model-checking

Le model-checking est une méthode permettant l'exploration exhaustive de l'espace d'états du système. Ces méthodes permettent de vérifier de nombreuses propriétés sur les systèmes autres que l'ordonnancement. Le formalisme utilisé est souvent celui des systèmes de transitions étendus temporellement : automates, réseaux de Petri. Il existe des extensions spécifiques aux systèmes temps réel comme les réseaux de Petri étendus à l'ordonnancement. L'inconvénient majeur, comme on peut le prévoir, est que l'espace d'états

que les méthodes de model-cheking doivent prendre en compte a une taille exponentielle, par conséquent, ceci limite la taille des modèles analysables.

C. Simulation

La simulation est couramment utilisée pour l'analyse des performances, le dimensionnement ou la mise au point des systèmes temps réel et l'analyse d'ordonnancement. Elle consiste à modéliser le comportement du système à l'aide d'un formalisme adéquat puis à l'exécuter avec un outil de simulation sur des scénarios définis par l'utilisateur. Comme il est difficile de réaliser une étude exhaustive de tous les scénarios possibles, elle peut montrer qu'un système n'est pas ordonnançable (en montrant un scénario non ordonnançable), mais elle ne peut prouver qu'un système est ordonnançable (même si tous les scénarios testés le sont). Des outils de simulations existent et sont utilisés dans l'industrie, parmi lesquels on peut citer Cheddar.

2.9. Présentation de de l'outil de simulation Cheddar

Cheddar est un framework d'ordonnancement des tâches en temps réel gratuit. Ce simulateur est conçu pour vérifier les contraintes temporelles des tâches d'une application/d'un système en temps réel. Cela peut également nous aider pour le prototypage rapide des ordonnanceurs en temps réel. Enfin, il peut être utilisé à des fins pédagogiques. Cheddar est un logiciel libre. Il est élaboré par l'équipe EA 2215, Université de Brest. Les principales fonctionnalités de l'outil sont :
Calcul d'ordonnancement pour la plupart des algorithmes "standards" :

- Rate Monotonic (ou RM)
- Earliest Deadline First (ou EDF)
- Deadline Monotonic (ou DM, Inverse Deadline).
- Least Laxity First (ou LLF).

L'environnement de simulation Cheddar présente les résultats sous deux aspects :
Aspect graphique : Les différents ordonnancements de tâches issus de l'application des algorithmes sont présentés à l'aide de diagrammes de Gantt, avec des légendes. Ces légendes sont des informations sur les contraintes temporelles des tâches, les différentes préemptions que subissent les tâches, les temps morts où le processeur est inactif ainsi que le contexte d'ordonnancement.
Aspect numérique : A partir d'un ordonnancement, l'environnement de simulation permet de calculer numériquement :

- Les temps de blocage sur ressources partagées (bornes maximales, minimales,délais moyens)
- Les temps de réponse des tâches (bornes maximales, minimales, délais moyens).
- Recherche d'interblocage, d'inversion de priorité.

- Echéances manquées.

Toutes les versions du simulateur Cheddar se trouvent dans le lien suivant:

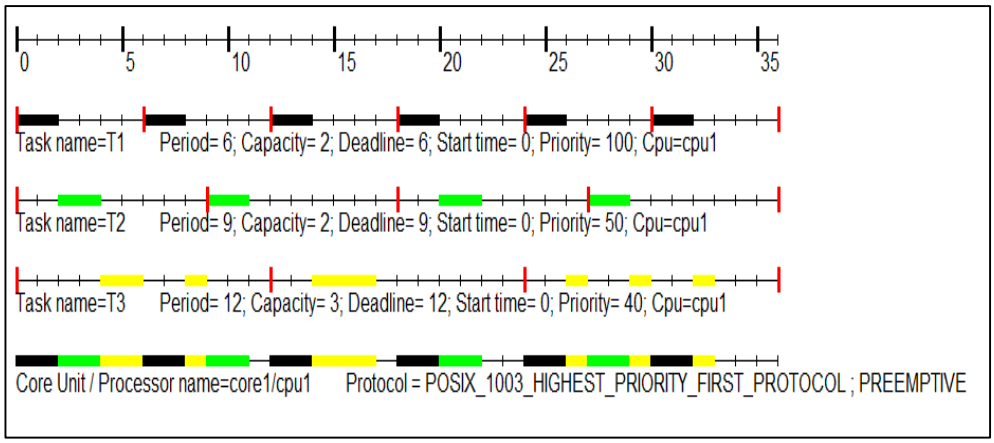
http://beru.univ-brest.fr/svn/CHEDDAR/trunk/releases.

2.10. Exercices

Exercice 1

Faite une analyse d'ordonnançabilité sur la simulation en utilisant l'outil Cheddar.
On veut simuler une configuration donné de notre système de tâches préemptibles et une priorité selon RM :
T1(0,2,6,6), T2 (0,2,9,9),T3 (0,3,12,12)
Comme résultat de simulation, nous avons:

1. L' ordonnancement des tâches donnés graphiquement

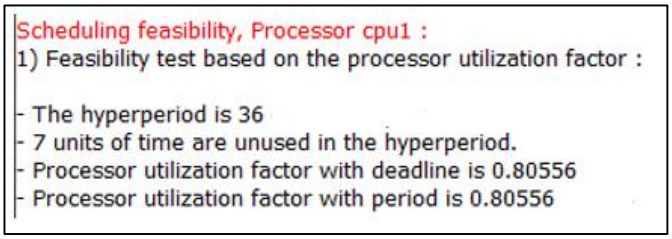


L'analyse par outil de simulation porte sur le temps de réponse des tâches. Cette analyse est présentée ci-dessous.

```
Scheduling simulation, Processor cpu1 :
- Number of context switches :  15
- Number of preemptions :  3

- Task response time computed from simulation :
   T1 => 2/worst
   T2 => 4/worst
   T3 => 9/worst
- No deadline missed in the computed scheduling : the task set is schedulable if you computed the scheduling on the feasibility interval.
```

L'analyse analytique de l'ordonnançabilité basée sur le calcul du facteur d'utilisation



Exercice 2

Nous nous intéressons à la description de notre système de contrôle de vol présenté au chapitre 1.Nous voulons un ordonnancement EDF et préemptif des tâches. On suppose qu'on va lui associer un seul processeur et aux tâches (NL;NF ; PL; PF ; FL; FF ;AP) les contraintes temporelles suivantes :

Tâches	T (ms)	C(ms)	r(ms)	D(ms)
NL	120	20	0	120
NF	120	10	0	120
PL	40	5	0	40
PF	40	5	0	40
FL	10	2	0	10
FF	10	1	0	10
AP	10	1	0	10

1. Donner une simulation par l'outil cheddar du système décrit ci-dessus,
2. Faite une analyse basée sur le calcul du temps de réponse,
3. Faite une analyse analytique de l'ordonnancabilité basée sur le calcul du facteur d'utilisation.
4. Refaire les questions ci-dessus selon la description suivante :

Tâches	T (ms)	C(ms)	r(ms)	D(ms)	Processeur
NL	120	20	0	120	1
NF	120	10	0	120	2
PL	40	5	0	40	1
PF	40	5	0	40	2
FL	10	2	0	10	2
FF	10	1	0	10	2
AP	10	1	0	10	1

3

Algorithmes d'ordonnancement des tâches temps réel périodiques et apériodiques

[Systèmes temps rée].
[Université Mustapha Stambouli-Mascara-].2022.

3.1 Introduction

L'application temps réel est une exécution d'algorithmes des tâches temps réel. Ces tâches se présentent dans un contexte bien spécifié ou un modèle de tâches bien défini. Il s'agit des tâches périodiques ou apériodiques, période fixe ou dynamique avec préemption ou sans préemption. L'objectif de ce chapitre est de détailler ces algorithmes en vérifiant l'ordonnancabilité des tâches. Chaque algorithme est présenté avec des exemples simples et complexes. Les plans des ordonnancements des applications temps réel sont présentés numériquement et graphiquement sous des diagrammes temporels (diagramme de Gantt). Les algorithmes concernent le calcul de pire cas d'éxecution des tâches ainsi que les algorithmes d'ordonnancement des tâches périodiques et apériodiques modélisant l'application temps réel

3.2 Algorithmes d'ordonnancement des taches périodiques dans les systèmes temps réel

Nous nous intéressons dans cette section aux algorithmes d'ordonnancement dans les systèmes temps réel monoprocesseur. Ces algorithmes sont basés sur les contextes suivants :

- Configuration : ensemble de N taches qui vont s'exécuter concurremment pour réaliser l'application.
- Accès concurrent au CPU (monoprocesseur) régi par un système de priorités.
- Ordonnancement préemptif : à chaque instant, c'est à la tâche de plus haute priorité que l'on attribue la CPU.
- Algorithme d'ordonnancement : mécanisme qui va attribuer à chaque tâche une priorité de façon à ce que l'ensemble des tâches de la configuration soit exécutée en respectant les contraintes temporelles et en vérifiant un critère d'ordonnancabilité.
- L'algorithme d'ordonnancement consistera à trouver une séquence de tâches caractérisée par une périodicité de longueur L qui a pour valeur le PPCM des périodes de toutes les tâches :
  - ✓ **L = [0, PPCM (Périodes des tâches du système)]** (tâches synchrones)
  - ✓ La période de longueur **L** est appelée **période d'étude, hyper période** ou **interval de travail**. Ce qui signifie que si la première occurrence de chacune des tâches est respectée, alors toutes les autres occurrences le seront.

- ✓ La séquence ainsi déterminée peut être reproduite à l'infini.
- ✓ La période d'étude peut s'avérer très longue selon les relations entre périodes.

Parmi les algorithmes nous avons :

3.2.1. Algorithme Rate Monotonic (Rate Monotonic Algorithm) (RMA)

Principe :

- L'algorithme Rate Monotonic (RMA) a été introduit par **Lui et Layland en 1973**.
- Algorithme à priorité constante ou fixe,
- La priorité des tâches est basée sur la période (taches à échéance sur requête) : La tâche de plus petite période est la plus prioritaire c'est-à-dire que :

$$\text{Priorité de } T_i = \frac{1}{\text{Période de } T_i}$$

- Les tâches ne communiquent pas.
- La tâche peut être préemptée à n'importe quel instant de l'exécution.
- Tâches échéance sur requête ($D_i = P_i$) avec D_i et P_i sont respectivement l'échéance et la période de la tâche i)
- Algorithme Rate monotonic est optimal (pour le modèle donnée) au sens de l'ordonnançabilité, s'il existe un algorithme capable de trouver une solution faisable pour l'ensemble des tâches étudiées, alors l'algorithme Rate Monotonic est aussi.
- Test d'ordonnancabilité (condition suffisante) comme suit :

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

Avec c_i : Temps de computation de la tâche i

P_i : Période de la tâche i

n: Nombre de tâches

Cette équation exprime que lorsque $n \rightarrow \infty$, l'utilisation du processeur doit rester inférieur à 69.3%.

Exemple 1

Un exemple de tâches échéances sur requêtes ordonnancées par l'algorithme *Rate Monotonic* est présenté dans la figure 2.9, l'algorithme est appliqué à trois tâches synchrones : tâche T1 ($c_1 = 3, p1 = D_1 = 20$), tâche T2 $(c_2 = 2\,; p_2 = D_2 = 5)$ tâche T3($c_3 = 2, P_3 = D_3 =$

10), avec c_i la durée d'exécution de la tâche T1 , P_i sa période et D_i son échéance. Ainsi, la tâche la plus prioritaire est la tâche T2 et la tâche la moins prioritaire est la tâche

T1. La séquence d'ordonnancement sur la période d'étude ou l'intervalle de travail est [0, 20]. Avec la solution trouvée, les trois tâches respectent leurs contraintes temporelles.

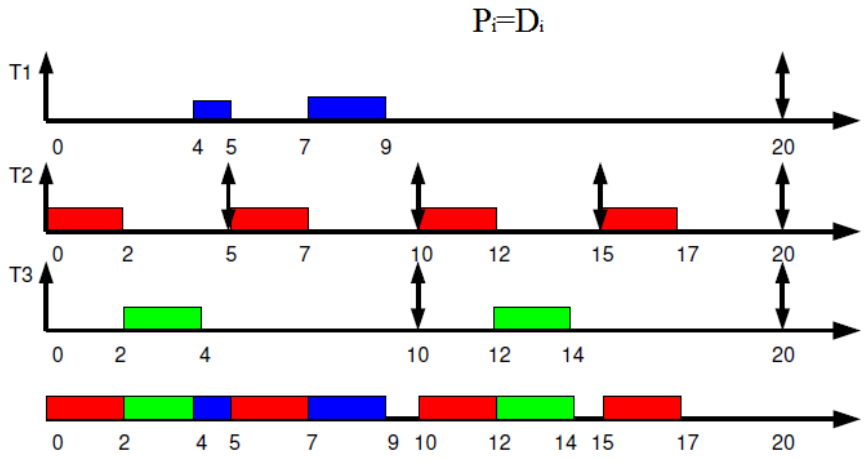


Figure 3.1 *Application de l'algorithme Rate Monotonic à un exemple de trois tâches périodiques.*

3.2.2. Algorithme de calcul du temps de réponse de tâche temps réel –WCET-

Le WCET (Worst Case Execution Time) constitue une borne supérieure sur le temps consommée (la tâche peut se terminer plus tôt). La valeur de ce paramètre doit être sécurisée, jamais dépassée la valeur du deadline, pour être efficace.

Ce temps ce calcule ainsi : $R_i = C_i + I_i$

- C_i est la charge de la tâche dans le processeur.
- I_i est le temps pendant lequel la tâche i est suspendue par des tâches plus prioritaires.
- Durant R_i, une tâche $j > i$ est activée $\left\lceil \frac{R_i}{P_j} \right\rceil$ fois
- l'interférence par la tâche j est $\left\lceil \frac{R_i}{P_j} \right\rceil \times C_j$
- l'interférence totale est $I_i = \sum_{j \in hp(i)} \left\lceil \frac{R_i}{P_j} \right\rceil \times C_j$

Avec hp(i) l'ensemble des tâches de plus forte priorité que *i*.
P_j est la période de la tâche j la plus prioritaire.

- Le temps de réponse de la tâche i est $R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{P_j} \right\rceil \times C_j$

L'algorithme de calcul du temps de réponse est donné par [Joseph et Pandya, 1986] comme suit :

1.	Initialisation	On démarre avec $\omega_i^0 = C_i$
2.	Traitement	Pour le calcul, on utilise une technique itérative : On évalue $R_i$de façon itérative avec ω_i^n $\omega_i^{n+1} = C_+ + \sum_{j \in hp_{(i)}} \left\lceil \frac{\omega_i^n}{P_j} \right\rceil * C_j$ P_j : Période de la tâche j
3.	Test d'arrêt	On s'arrête : lorsque $\omega_i^{n+1} = \omega_i^n$ → cas réussite lorsque $\omega_i^{n+1} > P_i$ → Cas Echec

Exemple 2

Soit le système de tâches de trois taches T1, T2 et T3.

T 1 (C1=3, P1=10) ; Tâche 2 (C2=4, P2=15) ; tâche 3 (C3=2, P3=20)

1/P1>1/P2>1/P3 → Priorité T1= 1/P1 > Priorité T2=1/P2 > priorité T3= 1/P3

Solution

$R_{T1} = ?$

$hp_{(T1)} = \emptyset$; $\omega_{T1}^0 = 3$ $R_{T1} = 3$

$R_{T2} = ?$

$hp_{(T2)} = \{T_1\}$

$\omega_{T2}^0 = 4$

Iteration 1: $\omega_{T2}^1 = 4 + \left\lceil \frac{4}{10} \right\rceil \times 3 = 7$

Iteration 2: $\omega_{T2}^2 = 4 + \left\lceil \frac{7}{10} \right\rceil \times 3 = 7$ $\omega_{T2}^1 = \omega_{T2}^2 = 7$ → $R_{T2} = 7$

$R_{T3} = ?$

$hp_{(T3)} = \{T_1, T_2\}$

$\omega_{T3}^0 = 2$

Iteration 1: $\omega_{T3}^{1} = 2 + \left\lceil \frac{2}{10} \right\rceil \times 3 + \left\lceil \frac{2}{15} \right\rceil \times 4 = 2 + 3 + 4 = 9$

Iteration 2: $\omega_{T2}^{1} = 2 + \left\lceil \frac{9}{10} \right\rceil \times 3 + \left\lceil \frac{9}{15} \right\rceil \times 4 = 2 + 3 + 4 = 9 \omega_{T2}^{1} = \omega_{T2}^{2} = 7 \rightarrow R_{T2} = 9$

3.2.3 *Algorithme Deadline Monotonic (Deadline Monotonic Algorithm) (DM)*

Principe

- L'algorithme Deadline Monotonic a été introduit par Leung en 1982 [LW92] pour des tâches respectant la caractéristique $P_i \geq D_i \geq C_i$, avec $P_i , D_i \; et \; C_i$ respectivement la période, l'échéance et la charge de la tâche T_i .
- Basé sur le délai critique : la tâche de plus petit délai critique D_i est la plus prioritaire, (La priorité d'une tâche est inversement proportionnelle à son échéance)
- Equivalent à RM dans le cas des tâches à échéance sur requête $(P_i = D_i)$, meilleur dans les autres cas.
- Test d'ordonnancabilité est une condition suffisante de l'ordonnançabilité donnée comme suit :

$$\sum_{i=1}^{n} \frac{c_i}{D_i} \leq n(2^{\frac{1}{n}} - 1)$$

Avec c_i : Temps de computation de la tâche i

D_i : Période de la tâche i

n : Nombre de tâches

Exemple 3

Un exemple de tâches échéances sur requêtes ordonnancées par l'algorithme *deadline Monotonic* est présenté dans la figure 2.8, l'algorithme est appliqué à trois tâches synchrones : tâche T1 ($c_1 = 3, p1 = 20, D_i = 7$) , tâche T2 ($c_2 = 2; P_2 = 5 , D_2 = 4$) tâche T3($c_3 = 2, P_3 = 10, D_3 = 9$), avec c_i la durée d'exécution de la tâche T1 , P_i sa période et D_i son échéance. Ainsi, la tâche la plus prioritaire est la tâche 2 et la tâche la moins prioritaire est la tâche 1, la séquence d'écrite sur la période d'étude ou l'intervalle de travail est [0, 20], avec la solution trouvée, les trois tâches respectent leurs contraintes temporelles.

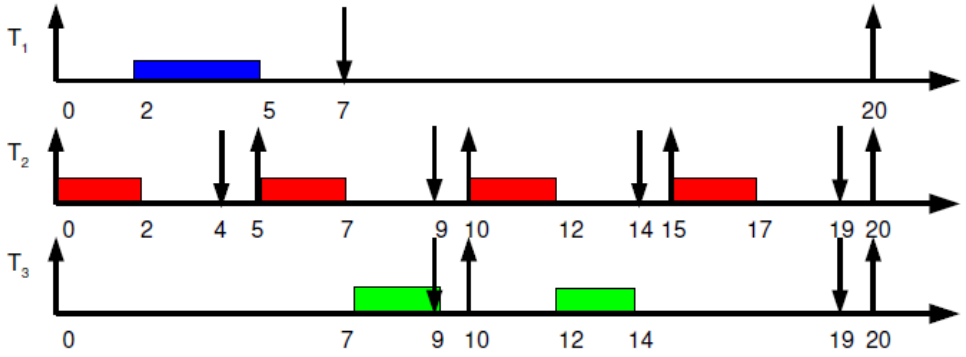


Figure 3.2 *Exemple d'ordonnacement de 3 tâches par l'algorithme Deadline Monotonic*

3.2.4. Algorithme Earliest deadline first (EDF)

Principe

- Obtenir une meilleure occupation du processeur.
- A tout moment, la tâche qui a l'échéance la plus proche occupe le CPU.
- Toutes les instances d'une tâche n'ont pas la même priorité ➔ priorité dynamique
- Préemption : quand une tâche arrive, si elle a l'échéance plus proche que la tâche en cours
- Un ensemble de n tâches est ordonnançable par EDF si l'utilisation du processeur est inférieure à 100% :

Si di = pi alors la Condition nécessaire et suffisante :

$$\sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1$$

Si di<pi alors la condition est suffisante mais pas nécessaire :

$$\sum_{i=1}^{n} \frac{C_i}{D_i} \leq 1$$

Exemple 4

Soit le système de tâches suivant.

T1 (r0 = 0, C=3, D=7, P=20), T2 (r0 = 0, C=2, D=4, P=5),
T3 (r0 = 0, C=2, D=8, P=10)

Vérifier son ordonnancabilité par EDF $\sum_{i=0}^{n} \frac{C_i}{P_i} = \frac{3}{20} + \frac{2}{5} + \frac{2}{10} = 0.75 < 1$

La figure 2.9 présente un diagramme d'ordonnancement des tâches T1, T2 et T3 selon EDF

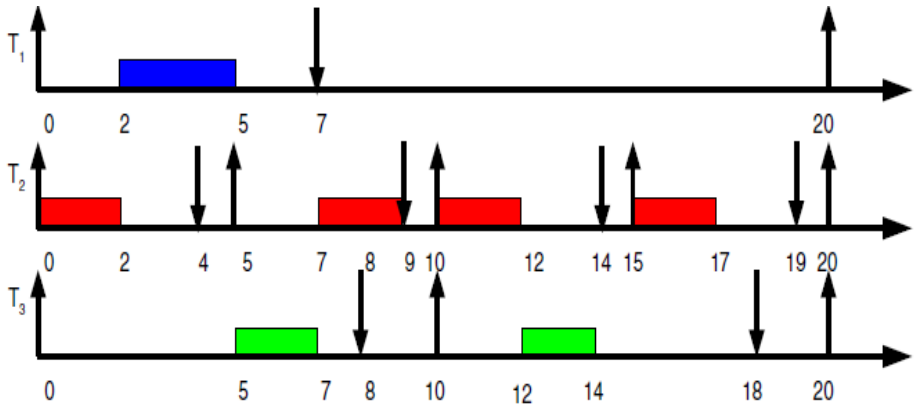


Figure 3.3 *Exemple d'ordonnancement de 3 trois tâches avec l'algorithme EDF à un exemple de trois tâches périodiques*

3.2.5 Algorithme least laxity first (LLFA)

Principe

- l'affectation des propriétés selon la laxité des tâches à chaque unité de temps.
- La tâche dont la laxité est la plus petite aura la plus grande priorité.
- Cet algorithme présente l'inconvénient du calcul de priorité à chaque unité de temps, ce qui implique une charge de processeur plus élevée.

- Laxité d'une tâche $L_i(t) = D_i - reste(t)$

  Où: Di: Echéance absolue.

  t: temps courant

  Reste (t): le restant de capacité à exécuter pour l'activation courante.

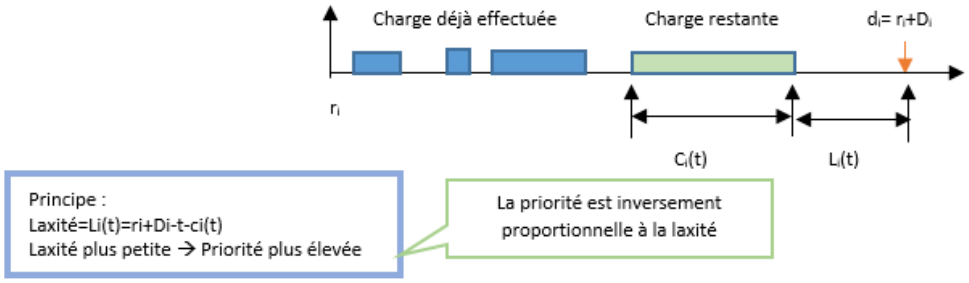


Figure 2.10. *Laxité d'une tâche temps réel en fonction des contraintes temporelles*

Exemple 5

Soit le système de tâche suivant :

tâche T1 ($r_0 = 0, c_1 = 3,\ p1 = 20, D_i = 7$), tâche T2 $(r_0 = 0, c_2 = 2\,; P_2 = 5\,, D_{\ 2} = 4)$ tâche T3 $(r_0 = 0, c_3 = 2, P_3 = 10, D_3 = 8)$, avec r_0 est la première date de réveil de la tâche, c_i sa durée d'exécution de la tâche $P_{i,}$ sa période et D_i son échéance. $L_1 = 7 - 4 = 3, L_2 = 4 - 2 = 2, L_3 = 8 - 2 = 6$). La figure 3.4 présente un graphique d'ordonnancement de tâches par LLFA

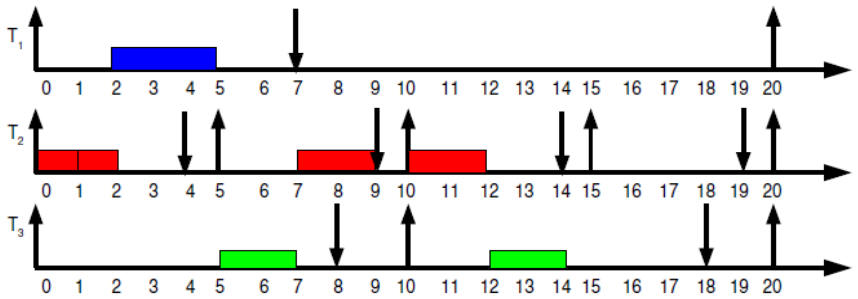


Figure 3.4 *Exemple d'ordonnancement de 3 trois tâches avec l'algorithme LLFA*

3.3 Algorithmes d'ordonnancement des tâches apériodiques

3.3.1 Présentation des tâches apériodiques

Dans cette section, nous nous intéressons aux modèles de tâches apériodiques et à leurs algorithmes d'ordonnancement. Une définition des tâches apériodiques a été donnée dans le chapitre 2 de ce manuscrit. Les tâches apériodiques présentent les caractéristiques suivantes :

- Les tâches apériodiques, elles sont caractérisées par le fait qu'elles peuvent demander à être exécutées à n'importe quel moment, sans avoir de temps minimal d'invocation.
- Le seul paramètre connu est la durée d'exécution, **Ca,** de la tâche. La date de réveil est aléatoire étant donné que l'instant de l'exécution de la tâche dépend du contexte d'évolution du procédé et ne peut pas être connu à priori.
- Le temps entre deux instances d'une tâche apériodique est inconnu ou connu en moyenne.
- Leurs exécutions sont prises en compte dans un travail de fond, c'est à dire dans les temps laissés libres par les tâches à contraintes strictes (tâches périodiques), soit prises par des serveurs d'apériodiques.
- Les tâches apériodiques peuvent être préemptées au cours de leurs exécutions et ainsi exécuter en plusieurs fois lors d'une instance.

La figure 3.4 représente graphiquement l'exécution d'une tâche apériodique Ai. Les deux dates de réveil (r et r') sont choisies aléatoirement et une charge de C_{Ai}.

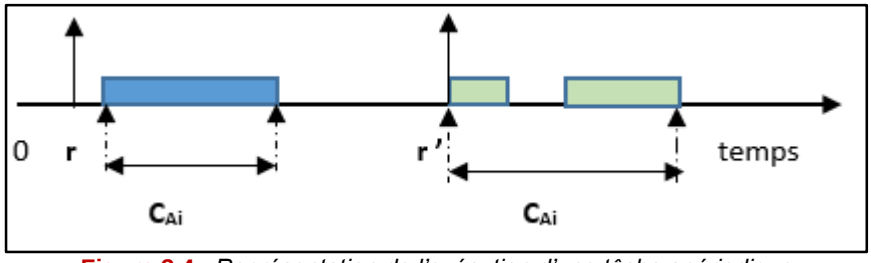


Figure 3.4 *Représentation de l'exécution d'une tâche apériodique*

3.3.2. Ordonnancement de tâches apériodiques.

Deux approches permettant l'ordonnancement des tâches apériodiques. La première approche est le traitement d'arrière-plan (background processing , la seconde est l'approche qui utilise le serveur de tâches.

A. Approche traitement d'arriere-plan

Principe

La méthode la plus simple pour prendre en compte les tâches apériodiques est de les ordonnancer lorsqu'aucune tâche périodique/sporadique n'est prête et lorsqu'il n'y a plus de tâches apériodiques acceptées précédemment à ordonnancer (le processeur est oisif). Les

tâches apériodiques sont prises en compte dans les temps creux des tâches périodiques. Il s'agit des tâches non critiques.

Exemple 6

Soit le système de tâches temps réel suivant :

Tp1 (r0=0, C=2, P=5, D=5), Tp2 (r0=0, C=2, P=10,D=10)

Ta3 (r=3, C=2), Ta4 (r=10, C=1), Ta5 (r=11, C=2)

Les tâches périodiques sont Ordonnancées selon la stratégie RMA.

La Figure 3.5 présente les plans d'ordonnancement des différentes tâches du système temps réel à échéance sur requête selon l'approche arrière-plan.

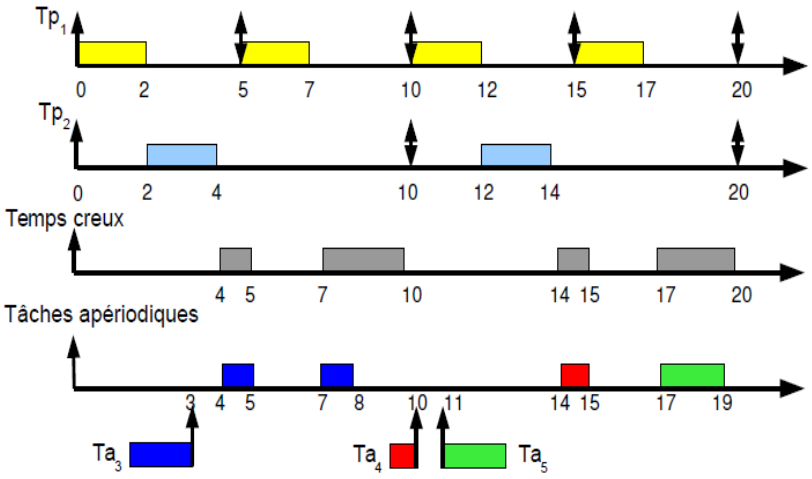


Figure 3.5 *Exemple d'ordonnancement de tâches apériodiques (stratégie arrière-plan)*

B. Approches serveurs de tâches

Principe

Des tâches nommées serveurs permettant de traiter dans le temps les tâches apériodiques. Le serveur est caractérisé par :

- Sa période.
- Son temps d'exécution : capacité du serveur.
- Serveur généralement ordonnancé suivant le même algorithme que les autres tâches périodiques.

- Une fois actif, le serveur sert les tâches apériodiques dans la limite de sa capacité.
- l'ordre de traitement des tâches apériodiques ne dépend pas de l'algorithme des tâches périodiques.

Deux types de serveurs se présentent : Serveur par scrutation (polling) et serveur sporadique.

B.1. Serveur par scrutation (polling)

Principe

- A chaque activation, le traitement des tâches en suspens jusqu'a épuisement de la capacité ou jusqu'à ce qu'il n'y ait plus de tâches en attente.
- Si aucune tâche n'est en attente (à l'activation ou parce que la dernière tâche a été traitée), le serveur se suspend immédiatement et perd sa capacité qui peut être réutilisée par les tâches périodiques (amélioration du temps de réponse).

Exemple 7

Soit le système temps réel suivant:

Tâches périodiques :

Tp1 (r0=0, C=3, P=20), Tp2(r0=0, C=2, P=10),Tps(r0=0, C=2, P=5),

Tâches apériodiques :

Ta3(r0=4, C=2),Ta4(r0=10, C=1),Ta5(r0=11, C=2).

Ordonnancer ce système par RM. La condition d'ordonnancabilité est vérifiée:

$$\sum_{i=1}^{n} \frac{C_i}{P_i} + \frac{C_s}{P_s} \leq (n+1)\left(2^{\frac{1}{n+1}} - 1\right)$$

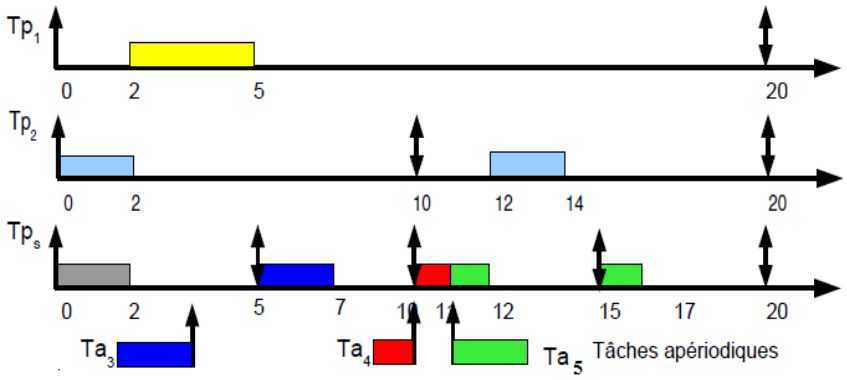


Figure 3.6 *Ordonnancement avec serveur par scrutation (polling)*

Limitations du serveur par scrutation :

- Perte de la capacité si aucune tâche apériodique en attente.
- Le serveur de tâches est suspendu si aucune des requêtes de tâches apériodiques n'est arrivée .Il faut attendre la requête suivante.

B.2 Serveur sporadique

Principe

- Si le serveur sporadique est activé par le biais de présence des tâches apériodiques, ce dernier exécute les tâches apériodiques sur le long de son capacité par son période (avec la possibilité de suspendus le serveur si les tâches apériodiques épuisent leurs capacités avant l'écoulement de la capacité du serveur).
- Le serveur sporadique améliore le temps de réponse des tâches apériodiques sans diminuer le taux d'utilisation du processeur pour les tâches périodiques.
- le serveur sporadique peut être considéré comme une tâche périodique « normale » du point de vue des critères d'ordonnancement.

Exemple 8

Soit le système temps réel suivant :

• Tâches périodiques :

Tp1 (r0=0, C=3, P=20), Tp2 (r0=0, C=2, P=10) et Tps (r0=0,C=2, P=5).

• Tâches apériodiques :

Ta1 (r0=4, C=2), Ta2 (r0=10, C=1) et Ta3 (r0=11, C=2).

Un ordonnancement des tâches par la stratégie « serveur sporadique » est illustré par la figure 3.7

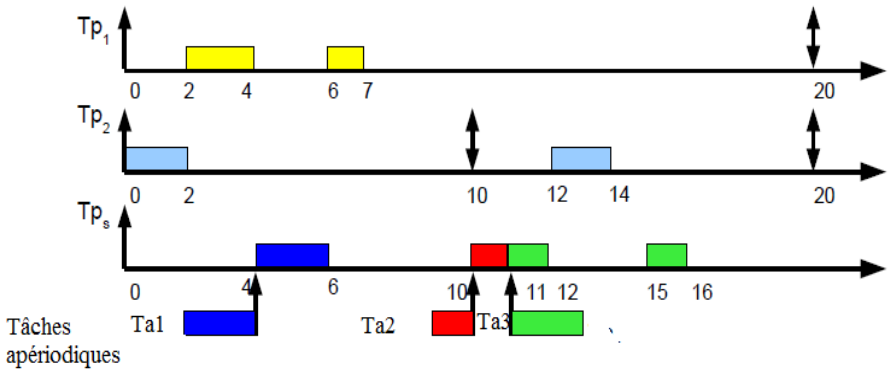


Figure 3.7 *Exemple d'Ordonnancement de tâches avec RM avec serveur sporadique*

3.3.4. Comparaison entre les algorithmes d'ordonnancement des tâches apériodiques

Une comparaison entre les algorithmes d'ordonnancement des tâches apériodique est résumée comme suit :

Serveur à scrutation de tâches

- Facile à implémenter
- Ordonnançable avec RM
- Le serveur est une tâche périodique à priorité fixe.
- Gaspillage du temps CPU
- La capacité non utilisée est perdue.
- Fonctionne bien si peu de tâches apériodiques (la capacité peu être déterminée de manière optimale).

Serveur sporadique

- Meilleure utilisation du CPU
- Bien adapté si beaucoup de tâches apériodiques
- Plus difficile à implémenter.

3.4. Classes de problèmes d'ordonnancement des tâches temps réel

Le tableau 3.1 résume les classes des problèmes d'ordonnancement des tâches temps réel :

Classes des problèmes d'ordonnancement des tâches temps réel	Descriptions
Monoprocesseur	➢ L'architecture ne dispose que d'un seul processeur
Multiprocesseurs	➢ Plusieurs processeurs
Préemptif	➢ La préemption se traduit par la suspension temporaire d'une l'exécution tâche au profit d'une autre tâche
Non préemptif	➢ Les tâches des systèmes ne peuvent être pas préemptées
En-ligne	➢ Il se fait au fur et à mesure que les tâches arrivent dans le système. ➢ l'ordonnancement consiste à déterminer la séquence d'exécution durant la vie du système temps-réel ➢ réévaluation en ligne de l'ordonnancement en fonction de la modification des critères de choix : algorithme dynamique
Hors-ligne	➢ la séquence d'ordonnancement est établie avant le lancement de l'application pour être répétée à l'infini. ➢ Une tâche n'est qu'un concept de conception ➢ les décisions d'ordonnancement sont prises avant l'exécution du système : algorithme statique.
Non oisif	➢ un processeur exécute la tâche la plus prioritaire dès qu'elle est prête à être exécutée et qu'il ne peut pas la retarder s'il n'a rien d'autre à faire.
Oisif	➢ un système peut n'être ordonnançable qu'en n'exécutant aucune tâche pendant un certain temps.
Statique	➢ basé uniquement sur les paramètres des tâches au départ (avant l'activation).

Dynamique	➢ un ordonnancement dynamique (réactif) est basé sur les paramètres des tâches qui varient au cours de l'exécution.
Optimal	➢ s'il permet de trouver un ordonnancement qui respecte toutes les contraintes lorsqu'un tel ordonnancement existe.
Non optimal	➢ un algorithme d'ordonnancement non-optimal vise à trouver des solutions approchées
Ordonnancement conduit par le temps	➢ Ordonnancement est conduit par le temps (tick-driven) et consiste à appeler l'ordonnanceur périodiquement. L'ordonnanceur prend une décision d'ordonnancement qui sera alors appliquée par le système d'exploitation.
Ordonnancement conduit ou les évènements	➢ Ordonnancement est conduit par les évènements (*event-driven*) et consiste à réagir aux évènements tels que des activations ou terminaisons de tâche

Tableau 3.1 *Classes des problèmes d'ordonnancement des tâches temps reel*

3.5. Exercices

Exercice 1

Soient trois tâches périodiques T1, T2 et T3 définies par les paramètres suivants : S1 = S2 = S3 = 0, P1 = 29, C1 = 7, P2 = 5,C2 = 1, P3 = 10,C3 = 2. On suppose un ordonnancement à priorité fixe avec une affection Rate Monotonic des priorités, et échéance sur requête.

1. Etudier l'ordonnançabilité de ce système de tâches.
2. Sur l'hyperpériode de ce système, donner graphiquement l'ordonnancement préemptif généré par RM puis l'ordonnancement non préemptif. Que remarquez vous ?
3. En se basant sur L'algorithme de Joseph et Pandya, calculer le temps de réponse de chaque tâche. Que remarquez vous.

Exercice 2

soit le système de tâches A(9,75) ; B (20,35) ;C(5,20)

1. Le système de tâches est-il oronnançable par RMS
2. Calculer le temps de réponse de A,B et C.

Exercice 3

Soit une configuration avec 2 tâches à échéance sur requête :

T1 : (r0 = 0, C1 = 2, P1 = 5)

T2 : (r0 = 0, C2 = 4, P2 = 7)

1. Donner l'intervalle d'étude.
2. la configuration est-elle ordonnançable avec RMA.

Justifiez votre réponse.

3. même question avec EDF

4. tracer les chronogrammes en indiquant les éventuelles fautes temporelles

5. Déterminer le nombre d'unité de temps libre sur l'interval de travail.

Exercice 4

Soit le système comportant 4 tâches avec les caractéristiques suivantes :
A(100,20,100) ;B(50,12,50),C(35,10,35),D(25,5,15)

1. Quelle politique utilise-t-on ?
2. Quel ordre de priorité ? Est-ce ordonnançable

Exercice 5

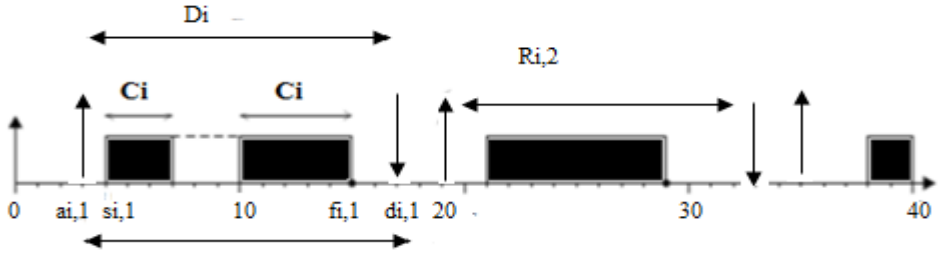
Soient deux tâches T1 et T2, périodiques, synchrones et à échéances sur requêtes, définies par les paramètres suivants :

Tâche T1 : Charge = 4, Deadline = Période = 8, temps de réveil = 0
Tâche T2 : Charge = 5, Deadline = Période = 10, temps de réveil = 0
Priorités des tâches : appliquer l'algorithme Rate Monotonic.

1. Elaborer un ordonnancement sur l'intervalle de faisabilité pour ce jeu de tâches, s'exécutant sur un processeur et utilisant un algorithme à priorité fixe préemptif
2. Calculer les pires temps de réponse (WCET) pour chaque tâche. Que remarquez vous ?
3. Utiliser un algorithme EDF préemptif pour ordonnancer ce système de tâches, puis refaire la questions 2.
4. Comparez les résultats obtenus. Que constatez-vous ?

Exercice 6

1. Décrire le shéma suivant :
2. Ddonner un titre à ce graphique.



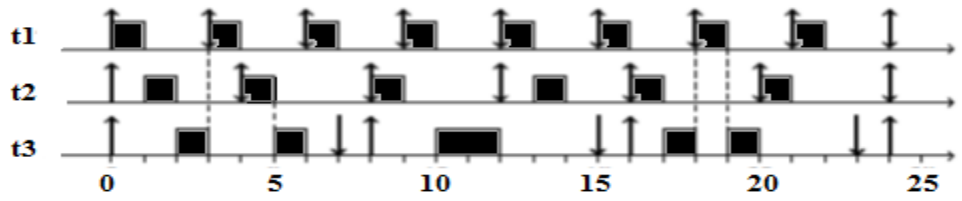
Exercice 6 : Solution

1.

- Ce graphique est un exemple d'exécution d'une tâche périodique à échéance contrainte (Di < Pi).
- La période de la tâche i est Ti=1, échéance (Di) =14.
- Le premier travail de la tâche i s'active à t=3 ($a_{i,1} = O_i$), mais ne commence à s'exécuter qu'à t=4 ($s_{i,1}$).
- Ce travail est suspendu en t=7 et reprend son exécution en t=10 pour finalement se terminer en t=15 ($f_{i,1}$).
- L'échéance est bien respectée puisque $f_{i,1} < d_{i,1} = a_{i,1}+D_i$.
- Le temps de réponse ($R_{i,1}$) correspond à la durée entre l'activation du travail et la fin de son exécution : $R_{i,1}=f_{i,1}-a_{i,1}=12$.

2 .Schéma représentatif d'une tâche i temps réel, périodique à échéance contrainte (Di<Pi)

Exercice 7 Soit le graphique suivant :



1. A partir du graphe, décrit le modèle de tâches avec ses contraintes temporelles.
2. De quel exécutif temps réel s'agit-il ?
3. Donner l'intervalle de travail.
4. Donner un titre au graphique.

Exercice 7 : Solution

1. Soit tâches t1,t2 et t3 avec les contraintes temporelles suivantes :

A : Date d'arrivée, T :Période, C :Temps de computation et D :Deadline

	A	T	C	D
t1	0	3	1	3
t2	0	4	1	4
t3	0	8	2	7

2. L'exécutif temps réel est : Rate Monotonic (RM) en mode préemptif. L'attribution des priorités selon RM. La tâche t1ayant la plus petite période, a la plus haute priorité. La tâche t3 possède la plus basse priorité. La seule tâche préemptée est la tâche t3 aux instants t=3 par la tâche t1, t2 et t=18par la tâche t1. La 2éme instante de la tâche t3, à t=10, peut s'exécuter sans interruption.
3. Intervalle de travail est de 24 unités de temps.
4. Graphe d'un ordonnancement RM de systèmes de tâches temps réel.

Exercice 8

Soit le système de tâches avec un temps d'arrivé a_i , un temps de computation C_i et un deadline d_i

	J_1	J_2	J_3
a_i	1	2	2
C_i	2	2	3
d_i	6	5	11

1. Déterminer la laxité de chaque job J_i $avec$ $i = 1,2$ 3.
2. Trouver un ordonnancement faisable avec et sans préemption.

3. Déterminer le temps de réponse, lateness de chaque job dans l'ordonnancement de chaque job.

Exercice 9

Considérons trois tâches périodiques ordonnancées par la stratégie RM basé sur la période comme critère :

T1 (r1=0, C1=3, P1=7) ; T2 (r2=0, C2=2, P2=12) ; T3 (r3=0, C3=5, P3=20) ;

En appliquant l'algorithme de Joseph et Pandya, Montrer que WCET de T1=3, WCET de T2=5 et WCET de T3=18.

Exercice 10

Nous voulons faire un ordonnancement préemptif Rate Monotonic (RM) avec l'algorithme de Lui et Layland.

Cet Ordonnancement concerne le système de tâches suivant périodiques et indépendantes à échéance sur requête. Les tâches sont définies par les paramètres temporels suivants : - T1 (r0= 0, C=3, D=10, P=10) - T2 (r0= 0, C=4, D=15, P=15) - T3 (r0= 0, C=2, D=20, P=20)

1. Etudier l'ordonnançabilité du système de ce tâches.
2. Calculer l'hyperpériode de ce système de tâches. Justifier votre réponse.
3. Tracer le diagramme temporel (graphe de Gantt) des tâches ci dessus et de la séquence d'exécution correspondante.
4. Calculer les temps creux du CPU.

En plus des tâches périodiques précédentes, le système en temps réel doit traiter des tâches apériodiques.

On considère deux cas d'arrivée d'une tâche apériodique (Tap):

Cas A : échéance à 6 : Tap (r= 12, C=2, D=14)

Cas B : échéance à 10 : Tap (r= 12, C=2, D=16)

Nous voulons faire un ordonnancement de tâches avec la stratégie : ordonnancement avec un serveur par scrustation. Cette tâche « serveur : Ts» se présente avec les caractéristiques suivantes

Ts (r0=0, C=1, D=10, P=10).

- Donner le facteur d'utilisation U et la période d'étude de cette nouvelle configuration.
- Faite une analyse analytique sur l'ordonnançabilité par RM du nouveau système de tâches.
- Tracer le diagramme temporel et la séquence d'exécution correspondante. Est-il possible de respecter l'échéance de la tâche apériodique Tap (considérer les deux (2) cas)?
- Donner l'avantage d'ordonnancer avec le serveur par scrutation.

4

Modède de tâches temps réel dépendantes

4.1 Introduction

En plus des contraintes temporelles, d'autres contraintes peuvent s'ajouter à la description des tâches d'une application temps réel telle que les contraintes de ressources et les contraintes de précédence. L'objectif de ce chapitre est de mettre l'accent sur les caractéristiques de ce modèle de tâches temps réel et les différents algorithmes et protocoles d''ordonnancement de ce système de tâches.

4.2. Tâches temps réel dépendantes

Dans une application temps réel ou système temps réel, deux types de dépendances entre tâches :

- dépendance liée au partage des ressources entre les tâches : Les tâches se partagent des ressources communes de manière exclusive (si une seule tâche à la fois peut l'utiliser). Comme exemples de ressources, nous pouvons citer, une donnée accédée en écriture, des capteurs ou actionneurs, etc.
- dépendances liée aux Contraintes de précédence entre les tâches : Une tâche doit absolument s'exécuter avant une autre

> Le partage de ressources dans les systems temps réel peut engendrer des problèmes d'inversion de priorité ou d'interblocage.

4.3 Tâches dépendantes par partage des ressources

Dans les systèmes temps réel, le partage de ressources en exclusion mutuelle cause des problèmes. Avant d'analyser cette problématique et donner des solutions, des notions essentielles sur les ressources partagées et leur gestion doivent être évoquées.

4.3.1 *Types de ressources*

Ressource : C'est le besoin de l'avancement de l'exécution d'une tâche.
Ressource partagée : Ressource utilisée par plusieurs tâches.
Ressource en exclusion mutuelle : Ressource partagée et utilisée par une seule tâche pendant un certain temps délimitée. A tout instant du système, il y a au maximum un processus qui peut entrer dans une section critique.

4.3.2 Notions de séction critique

Portion de code où la tâche (processus) utilise la ressource critique sous la contrainte d'exclusion mutuelle. L'exécution de la section critique est une assurance pour une tâche que l'exécution d'une séquence d'instructions se fait sans interférence. La figure 4.1 illustre la notion de section critique et des processus en section critique.

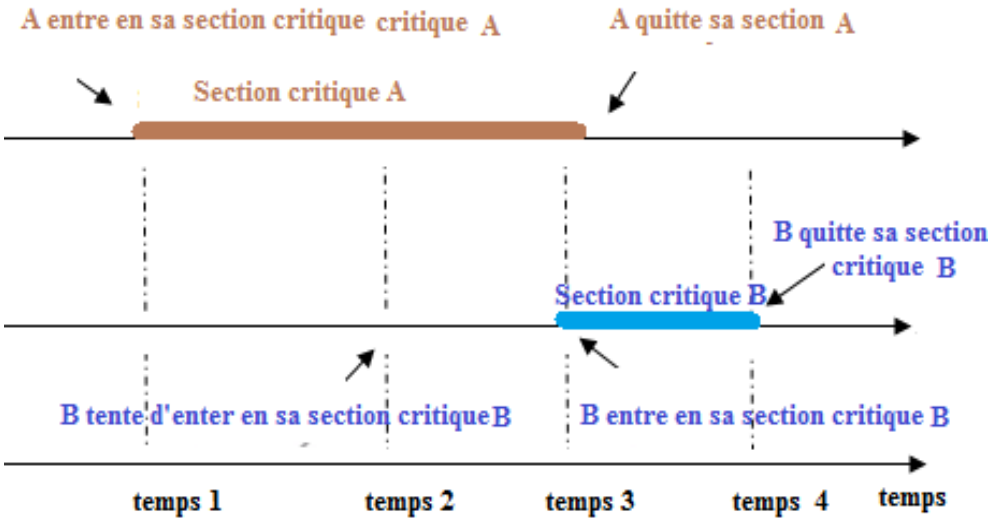


Figure 4.1 *Processus A et processus B sont en concurrence pour l'exécution de leurs sections critiques*
< section critique A, section critique B>

L'exécution de la section critique doit vérifier quatre conditions :

1. A tout instant un processus au plus peut se trouver en section critique.
2. Si plusieurs processus sont bloqués en attente de la ressource critique, alors qu'aucun processus ne se trouve en section critique. L'un d'eux doit pouvoir y entrer au bout d'un temps fini (éviter qu'un blocage mutuel des processus puisse durer indéfiniment)
3. Si un processus est bloqué hors d'une section critique, ce blocage ne doit pas empêcher l'entrée un autre processus en sa section critique.
4. La solution doit être la même pour tous les processus c'est-à-dire qu'aucun processus ne doit jouer de rôle privilégié.

Algorithme d'exécution de la section critique :

1. Début
2. Protocole d'entrée < > ;
3. Exécution section critique ;
4. Protocole de sortie < > ;
5. Fin.

Exemple 1 : Partage de ressource critique (solutions logicielle)
Résoudre le problème de partage de variables.

Variable *Occupé*
Processus pi

1. Début
2. **Section non critique**
3. Tant que (occupé) faire attente ftq //attente active//
4. Occupé:= vrai;
5. **Execution section critique** ;
6. Occupé:= faux;
7. **Section non critique**
8. Fin

4.3.3 *Géstion des ressources critiques*

Des outils matériels en systèmes temps réel permettant de gérer l'accès aux ressources critiques tout en assurant une sychronisation entre les processus concurrents Parmi ces outils : les sémaphores. Les appels aux sémaphores se font dans le protocole d'entrée et de sortie de la section critique.

Sémaphores
Un sémaphore s est constitué d'une variable entière e(s) et d'une file d'attente f(s). La variable e(s) peut prendre des valeurs entières positives, négatives ou nulles.Initialement le sémaphore a une valeur non nulle et non négative e_0 (s) et sa file f(s) est vide.
On peut agir sur un sémaphore s par les deux primitives indivisibles P(s) et V(s).

Algorithme de P(s) :

1. Début
2. e(s) :=e(s)-1 ;
3. Si e(s) < 0 alors

4. Début
5. Etat (r) :=bloqué // Cette primitive est exécutée par le processus r//
6. f(s) ← r // mettre dans le processus r dans la file f(s)//
7. Fin
8. Fin

Algorithme de V(s) :

1. Début
2. e(s) :=e(s)+1 ;
3. Si e(s) <= 0 alors
4. Début
5. q← f(s) // sortir le processus q de la file f(s) //
6. Etat (q) :=Actif// Cette primitive est exécutée par le processus q //
7. Fin
8. Fin

4.3.3. Exclusion mutuelle

A tout instant un processus au plus peut se trouver en section critique. Le protocole d'exclusion mutuelle est décrit comme suit :

6. Début
7. Protocole d'entrée < > ; // appels de la primitive P (sémaphore)
8. Exécution section critique ;
9. Protocole de sortie < > ; // appels de la primitive V (sémaphore)
10. Fin.

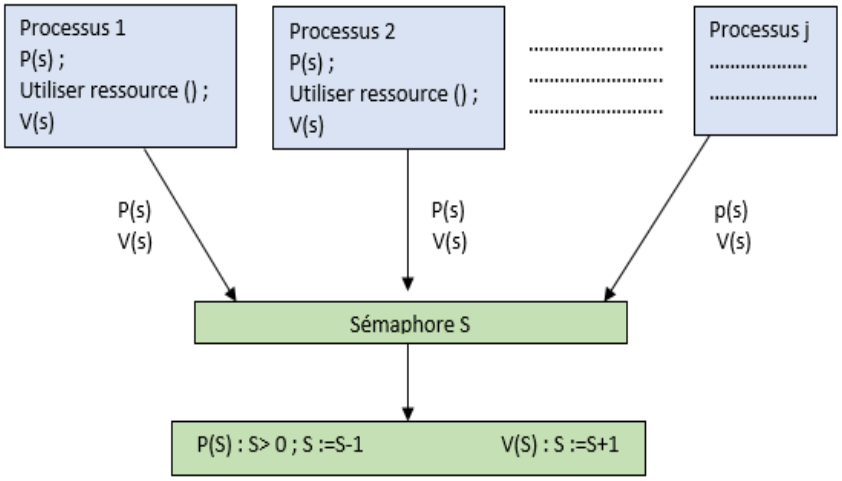


Figure 4.2 *Processus en synchronisation: utilisation de l'outil sémaphore*

4.3.4. Sémaphore d'exclusion mutuelle

L'exclusion mutuelle se résout comme suit : on introduit unsémaphore Mutex (Abréviation Mutual Exclusion). Initialisé à 1. Et chaque processus voulant accéder à une ressource critique à un seul exemplaire suit l'algorithme suivant :

Algorithme processus en exclusion mutuelle

1. Début
2. P (Mutex) ; // demande d'autorisation d'entrée en section critique//
3. Exécution section critique < > ;
4. V (Mutex) ;// sortie de sa section critique//
5. Suite des instructions ;
6. Fin

4.4. Problème d'inversion de priorité

Dans un système temps réel, une tâche prioritaire est bloquée alors qu'une tâche moins prioritaire qu'elle, s'exécute. Autrement dit, une tâche de faible priorité bloquant une tâche de plus forte priorité pendant une durée inconnue. La non gestion de l'inversion de priorité peut avoir des effets désastreux.

Exemple 3 : Une ordre d'arrêt d'urgence d'une centrale nucléaire qui serait bloqué par un autre ordre de mondre priorité.

Présentation du problème

Lorsque le processus de priorité basse est en section critique, le processus de priorité haute ne peut bien sûr pas entrer en section critique, il est donc bloqué : cela est inévitable. Cependant, le phénomène que l'on observe sur la figure 4.4 est que le processus de priorité intermédiaire, s'il est prêt au moment où le processus de faible priorité est en section critique, obtient le processeur alors même qu'un processus de priorité haute est en attente. Ce phénomène s'appelle **une inversion de priorité** et est à proscrire dans les systèmes pour lesquels le temps de réponse est important. La plupart des noyaux temps réel ou langages de programmation pour le temps réel proposent une solution à ce problème.

Illustration

- A de forte priorité
- B de priorité moyenne
- C de faible priorité

Scénario :

- C bloque une ressource
- Q demande (et bloque) sur la ressource R
- B arrive
- B prend 100% CPU ; car il a la priorité sur C
- C ne relâche pas R
- A ne peut pas être exécuté
- B bloque A (par l'intermédiare de C)

Solutions

- Ne peut pas trop prioriser les processus de priorité supérieure. (pas de variable en temps réel)

- Aléatoirement, donner du temps CPU à des tâches de priorité basse
- Donner une priorité à un mutex. Si un processus bloque le mutex ; il hérite ()

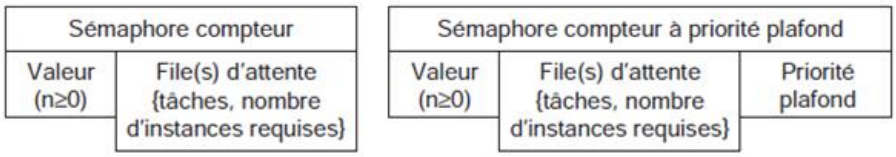


Figure 4.3. Comportement d'un sémaphore compteur :
(a) *Sans protocole de gestion de ressource / avec protocole héritée*
(b) *: Avec protocole à priorité plafond.*

4.4.1. Exemple de problème d'inversion de priorité avec un temps de blocage limité

Le schéma présenté par la figure 4.3 illustre le problème d'inversion de priorité comme suit :
Soit deux tâches T1 et T2 dont la priorité de T1 est plus grande que celle de T2. A un instant, la tâche T1 veut accéder à une ressource critique détenue par T1 et protégée par le sémaphore d'exclusion mutuelle Mutex.

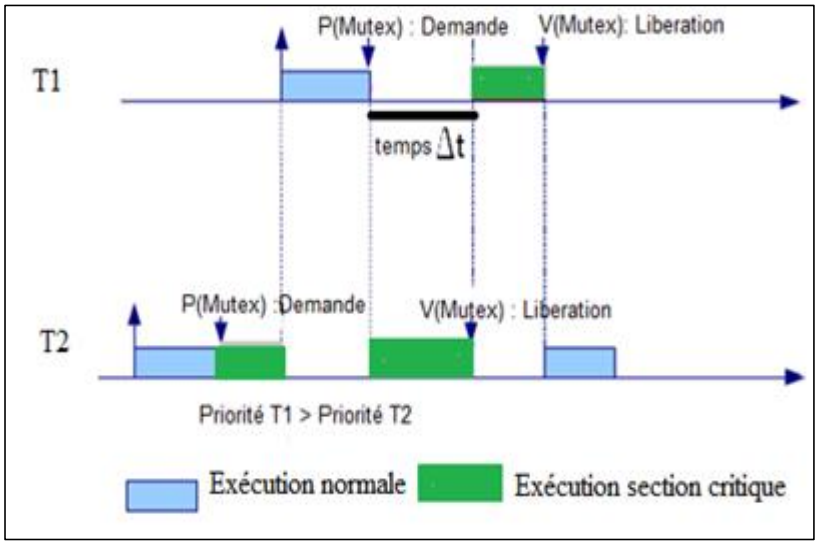


Figure 4.4. *Synchronisation de 2 tâches : Phénomème d'inversion de priorité : (Le délais imposé par T2 à T1 est normal)*

4.4.2. Exemple de problème d'inversion de priorité avec un temps de blocage illimité

Le temps de blocage peut être illimité. La tâche avec haute priorité est bloquée par une tâche de faible priorité pendant un intervalle de temps illimitée. Le graphe présenté par la

figure 4.4 illustre ce phénomène.

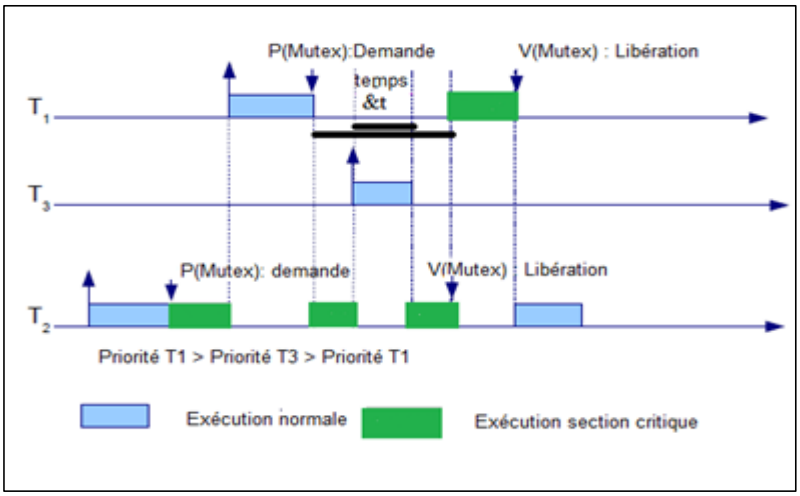


Figure 4.5 *Synchronisation de 3 tâches : Phénomème d'iversion de priorité(le délai supplémentaire apporte à T1 par T3 est dû au Phénomène d' ' inversion de priorité)*

La tâche T3 de priorité intermédiaire qui n'utilise pas la ressource R1 retarde la tâche T2 et donc la tâche T1. Pour pouvoir borner l'attente de T1 et inclure cette borne au test d'acceptabilité de la configuration à ordonnancer, le test d'acceptabilité de la configuration des tâches périodiques est :

Test classique + facteur de blocage B

4.5. Protocoles pour problème d'inversion de priorité

Comme solutions pour soulever le problème d'inversion de priorité, introduire un protocole de contrôle de la concurrence pour accéder aux sections critiques. Les protocoles d'accès aux ressources sont :

- Protocole Non Premption : Non Preemptive Protocol (NPP)
- Protocole d'héritage de priorité : Priority Inheritance Protocol (PIP)
- Priorité Plafonnée :Priority Ceiling Protocol (PCP)

4.5.1. Protocole de non préemption : Non préemptive protocol (NPP)

A. Principe

La préemption est interdite dans les sections critiques (NPP). Lorsqu'une tâche entre dans une section critique sa priorité est augmentée à la valeur maximale.

B. Exemple d'ordonnancement avec NPP

La figure 4.5 présente un ordonnancement de tâches avec NPP comme solution du problème d'inversion de priorité.

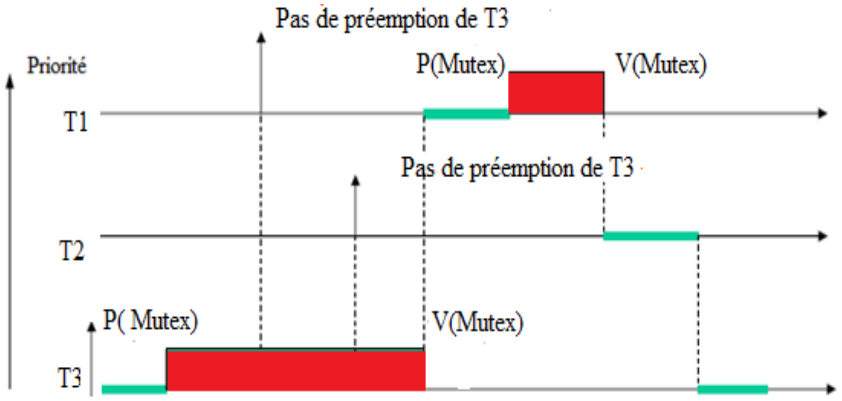


Figure 4.6 *Ordonnancement avec Le protocole Non Préemption (NPP)*

C. Problèmes de protocole NPP

- Difficile dans le cas d'une section critique longue.
- Les tâches hautement prioritaires qui n'utilisent pas la section critique se bloquent inutilement. Voir la figure 4.6

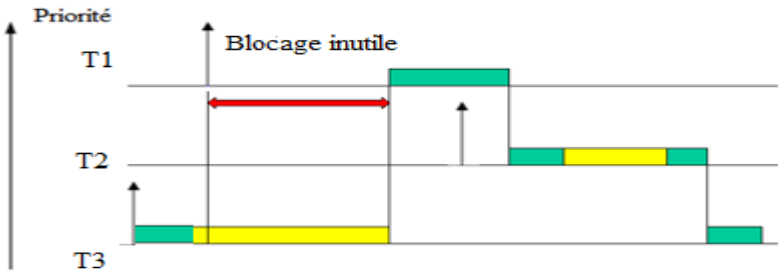


Figure 4.7 *La tâche T1 ne peut pas faire la préemption bien qu'elle peut le faire (blocage inutile)*

4.5.2. Héritage de priorite : Priority inheritance protocol (PIP)

A. Principe

Lorsqu'il y a blocage d'une tâche T2 (de haute priorité) lors de l'accès à la ressource R, la tâche bloquante T2 (de basse priorité) hérite de la priorité de T2. Utilisation par T2 de la priorité héritée jusqu'à libération de R, où elle retrouve la priorité qu'elle avait lors de son acquisition.

$$P_{SC} = \text{Max } \{P_k / T_k \text{ bloquées pour accès section critique}\}$$

Psc : Priorité de la tâche de faible priorité en section critique

P_k : Priorités des tâches T_k de plus hautes priorités.

B. Exemple d'ordonnancement avec le protocole PIP

La figure 4.7 présente un ordonnnacement de tâches en utilisant le protole PIP comme solution du problème d'inversion de priorité.

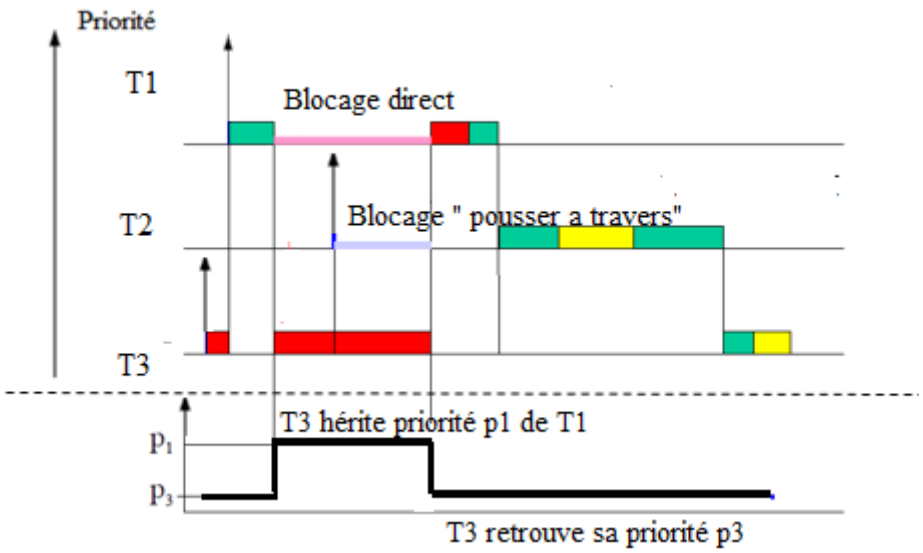


Figure 4.8 Ordonnancement avec le protocole héritage de priorité

C. Limites du temps de blocage par le protocole PIP

Le blocage est un retard causé par une tâche de priorité inférieure. Dans ce protocole, les blocages pouvant s'identifier comme :

- **Blocage direct :** Une tâche se bloque sur un sémaphore verrouillé
- **Blocage « pousser à travers » :** une tâche se bloque parce qu'une tâche de priorité inférieure a hérité d'une priorité supérieure.

  On montre que la borne B (temps de blocage maximal pour une tâche sur l'attente d'une ressource) est égale au plus à la somme des durées des sections critiques partagées avec des tâches de plus faible priorité. Si nous prenons comme exemple d'ordonnancement de n tâches périodiques sous l'algorithme Rate Monotonic, le test d'acceptabilité devient :

$$\sum_{j=1}^{j=n} {}^{C_j}/_{P_j} + {}^{B_n}/_{P_n} \leq n(2^{\frac{1}{n}} - 1) \qquad \text{Pour j allant de 1 à n tâches}$$

D. Exemple de blocage de tâches avec PIP

Soit trois tâches : T1, T2 et T3. Priorité de T1> Priorité de T2> Priorité de T3, la figure 4.8 illustre graphiquement le blocage des tâches resultant de l'ordonnancement par le protocole PIP.

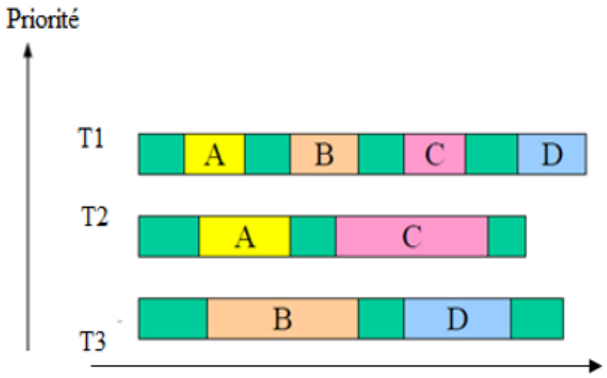


Figure 4.9 ***Blocage des tâches par le protocole PIP***

- T1 peut être bloquée une fois par T2 en (A2 ou C2) et une fois par T3 en (B3 ou D3)
- T2 peut être bloquée une fois par fois par T3 en (B3 ou D3)
- T3 ne peut être bloquée.

Avantages

- C'est transparent pour le programmeur
- Le protocole héritage de priorité (PIP) borne l'inversion de priorité

Problèmes

Le protocole PIP n'évite pas les blocages et les blocages enchainés

E. Exemple de blocage enchainé

Soit un système de tâches : T1, T2, T3 et T4. Ces tâches présentent des priorités comme suit : Priorité T1 > Priorité T2 > Priorité T3 > Priorité T4. A chaque demande de T1 (la tâche de plus haute priorité) pour accès à une ressource critique détenue par une tâche de priorité inférieure, La tâche T 1 peut être bloquée une fois au plus fois par chaque tâche de priorité inférieure que la priorité de Ti. Dans cet exemple la tâche T1 a subi trois blocage B1, B2 et B3 respectivement des tâches de priorité inférieure T4, T3 et T2.

Ceci est illustré par le graphique présenté par la figure 4.9.

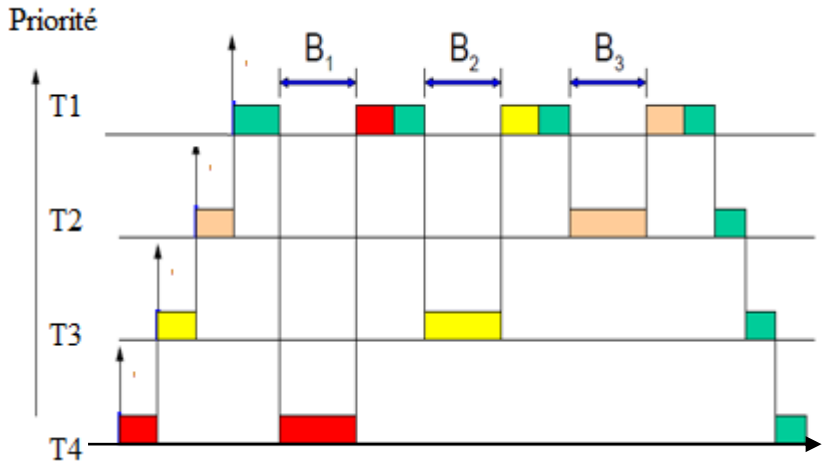


Figure 4.9 *Blocages chainés avec PIP*

4.5.3. Protocole priorité plafonnée : priority ceiling protocol (PCP)

A. Principe

Le protocole de plafonnement de priorité (PCP) consiste à associer une priorité (plafond) à chaque ressource. La priorité de la ressource est la plus haute parmi celles des tâches qui peuvent l'utiliser. L'ordonnanceur transfère cette priorité à chaque tâche qui

accède à cette ressource. Lorsqu'une tâche fini d'utiliser la ressource, sa priorité revient à la valeur d'origine.

Plafonds de ressources

A chaque sémaphore sk est attribué un plafond :

$$C(S_k) = \max \{P_j : T_j \text{ utilise } S_k\}$$

La tâche Ti peut entrer en section critique seulement si P_i > max {C(S_K) : S_K verrouillées par tâches autres que T_i}

B. Exemple d'ordonnancement des tâches avec le protocole PCP

Soit trois tâches T1, T2 et T3 avec priorité (P1) T1> priorité T2 (P2) > priorité T3 (P3). Le graphe présenté par la figure 4.10 présente un ordonnancement des tâches avec le protocole de priorité plafonnée. R1 est partagée entre T1 et T3, R2 est partagée entre T1 et T2.

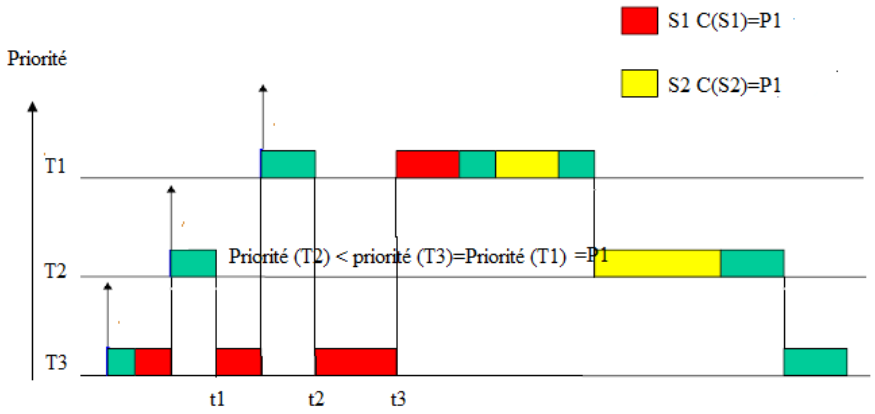


Figure 4.10 *Ordonnancement des tâches avec le protocole de priorité plafonnée*

- A t1, T2 est bloquée puisque la priorité de T2 (P2) < C(S1)
  - (T2 ne peut pas prendre R2 alors quelle est libre puisque la priorité de T2 est inférieure à la priorité plafond de la resource utilisée R1) donc T3 herite de la priorité de T2.
- A t2 : T3 hérite de priorité (S1) = priorité (T1)=P1.
- A t 3 : T3 reprend sa priorité initiale (la plus faible)=P3.

- A t4: T1 a demandé la resource R1 demande accordée car T1 est plus prioritaire et R2 partagée entre T1 et T2 est libre.

Avantages

- Une tâche ne peut être bloquée que pendant la durée d'une section critique d'une tâche de plus faible priorité.
- Protocole permet la prévention des interblocages.
- Le temps de blocage dû à la situation d'inversion de priorité se limite à l'utilisation de semaphore par la tâche de faible priorité

Problèmes

- Le protocole priorité plafonnée : Priority CeilingProtocol (PCP) n'est pas transparent au programmeur
- Semaphores ont besoin des plafonds
- En cas d'accés à plusieurs semaphores, la tâche de forte priorité va entrainer les temps de blocage
- Dans le cas d'accés à un semaphore par plusieurs tâches; les elevations de priorité vont s'enchainer
- Il ya blocages de tâche intermédiares qui ne partagent pas le sémapohore en question.

Autres definitions Protocole de priorité plafonnée

Notons que ce protocole se différencie du protocole à héritage de priorité simpleuniquement s'il y a plus de **2** ressources critiques partagées.

Exemple :

Prenons l'exemple de deux tâches **t1** et **t2** qui utilisent les deux ressources critiques **R1** et **R2**.

- La tâche **t1**ayant la priorité la plus forte, les deux ressources possèdent la même priorité plafond (**PP1 PP2 Prio1**), c'est-à-dire la priorité de **t1**. La figure 4.11 montre le principe de fonctionnement de ce protocole.

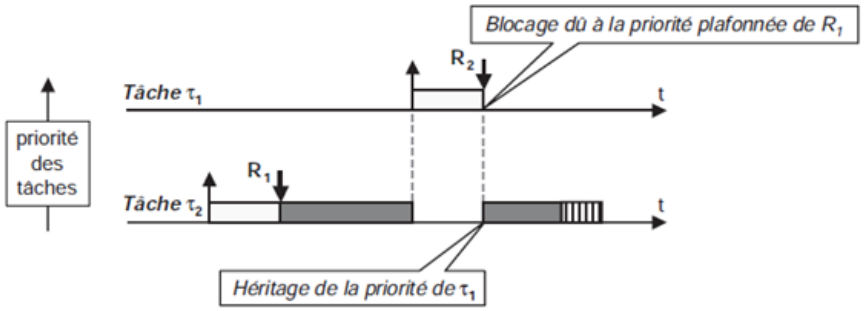


Figure 4.*11 Principe du protocole à priorité plafond*
Dans un ordonnacement en presence de ressources critiques.

C. Prévention des interblocages avec le protocole PCP

Deux tâches T1 et T2, pour exécuter leurs sections critiques, elles ont besoin aux ressources R1 et R2. La tâche T1 détient R1 et elle a besoin de R2, la tâche T2 détient R2 et demande R1. Ce cas de situation, met le système en situation d'interblocage. Une programmation concurrente à l'aide des sémaphores modélise le problème d'interblocage comme suit :

Var : R1, R2 : Sémaphore ; R1=R2=1

Tâche T1	**Tâche T2**
Début	Début
……. ;	……. ;
P(R1) ;	P(R2) ;
P(R2) ;	P(R1) ;
…….. ;	…… ;
Fin	Fin

Le problème de l'interblocage est présenté par la figure 4.11.

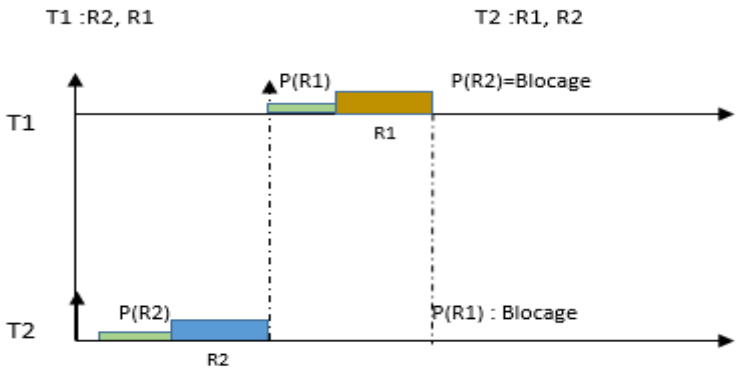


Figure 4.11 *Exemple de la situation d'interblocage causé par PCP.*

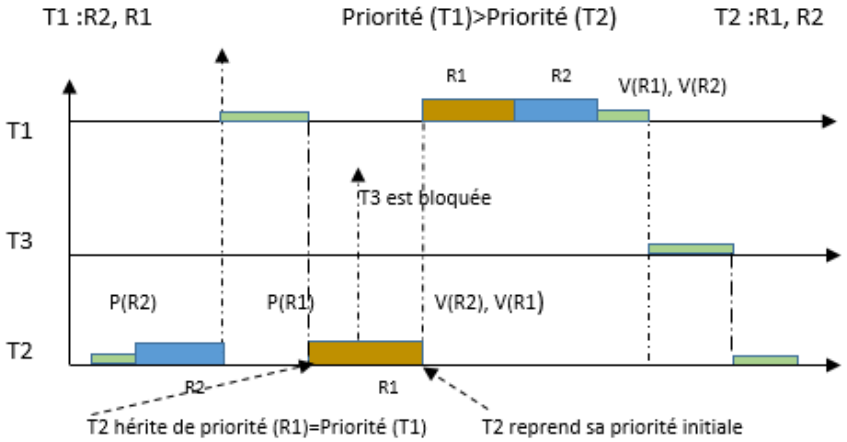


Figure 4.12 *Prévention des interblocages par le protocole de pririoté plafonée.*

Avec: c(r1)= c(r2)=priorité de T1=p1

Contraintes sur les ressources

- Choix de l'algorithme d'ordonnancement et le protocole d'accès aux ressources.

- Calcul des temps de blocages maximum (Bi) pour chaque tache.
- Nous effectuons le test de garantie, y compris le blocage termes.

4.6. Tâches dépendantes temps réel sous contraintes de précédence

Les contraintes de précédence entre tâches sont généralement décrites par un graphe orienté (acyclique) G présentée par la figure 4.13

- A < B indique que la tâche A est un prédécesseur de B

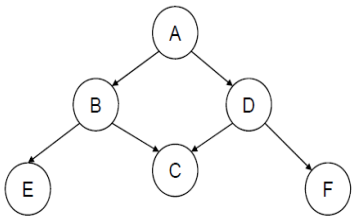


Figure 4.13 *Graphe de précédence*

4.6.1. Principe d'ordonnancement des tâches sous contraintes de précédence

Ce principe rend les tâches indépendantes en modifiant leurs paramètres selon Blazewicz et Chetto et al. La modification des paramètres des tâches dépendantes afin de rendre les tâches indépendantes

1. Les modifications des paramètres des tâches s'effectuent sur la date de réveil, sur les délais et sur les priorités.
2. validation de l'ordonnancabilité selon des critères utilisés pour des tâches indépendantes

4.6.2. Algorithmes d'ordonnancement avancés

A. Avec l'algorithme Rate Monotonic

$$\forall i,j \textbf{ Si } j < i \textbf{ Alors } r_i^* = Max\{r_i,\ r_j^*\} \forall i,j \textbf{ Si } i < j \textbf{ Alors } Priorité_i > Priorité_j$$

B. Avec l'algorithme Deadline Monotonic (DM)

$$\forall \mathbf{i,j}\ \textbf{Si } j < i \textbf{ Alors } r_i^* = \begin{cases} Max\{r_i,\ r_j^*\} \\ Max\{d_i,\ d_j^*\} \end{cases}$$

$\forall \boldsymbol{i,j}\ \textbf{Si } i < j \textbf{ Alors } Priorité_i > Priorité_j$ Selon L'algorithme DM

C. Avec l'algorithme Earliast Deadline First (EDF)

1. Modification des échéances de façon à ce qu'une tâche ait toujours un di inférieur à celui de ses successeurs
2. Calcul de r et de d selon les règles suivantes :

$$\forall i,j \textbf{ Si } j < i \textbf{ Alors } r_i^* = Max\{r_i\ , Max\ \{r_j^* + C_j\}$$
$$\forall i,j \textbf{ Si } i < j \textbf{ Alors } d_i^* = Min\{d_i\ , Min\{d_j^* - C_j\}$$

3. On commence les calculs par les tâches qui n'ont pas de prédécesseurs pour le calcul des r et par les tâches qui n'ont pas de successeur pour le calcul des d.

4.6.3. Exemple d'ordonnancement des tâches avec contraintes de précédence:

Soit le graphe de 5 tâches suivant avec les paramètres des tâches.

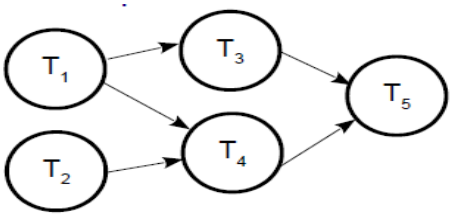


Figure 4.14 *Exemple de graphe de tâches avec lien de précédence*

	Contraintes temporelles		
Tâche	ri	ci	di
T1	0	1	5
T2	5	2	7
T3	0	2	5
T4	0	1	10
T5	0	3	12

Tableau 4.1 *Valeurs des contraintes temporelles des tâches temps réel dépendantes*

Appliquer le principe d'ordonnancement des tâches avec EDF sous contraintes de précédence pour le système de tâches ci-dessous.

Processus de résolution :

- T1 n'a pas de prédécesseur : r*1 = 0
- T2 n'a pas de prédécesseur : r*2 = 5
- T3 a T1 pour prédécesseur : r*3 = Max(r3, r*1+C1) = 1
- T4 a T1 et T2 pour prédécesseurs r*4 = Max(r4, Max(r*1+C1,r*2+C2))= 7
- T5 a T3 et T4 pour prédécesseurs r*5 = Max(r5, Max(r*3+C3,r*4+C4)) = 8
- T5 n'a pas de successeur d*5 = 12
- T4 a T5 pour successeur d*4 = Min (d4, Min (d*5-C5)) = 9
- T3 a T5 pour successeur d*3 = Min (d3,Min (d*5-C5))= 5
- T2 a T4 pour successeur d*2 = Min (d2,Min (d*4-C4))= 5
- T1 a T3 et T4 pour successeurs d*2 = Min (d2, Min (d*3-C3, d*4-C4))= 3

	Paramètres des tâches				
Tâche	**ri**	**ci**	**di**	r_i^*	d_i^*
T1	0	1	5	0	3
T2	5	2	7	5	7
T3	0	2	5	1	5
T4	0	1	10	7	9
T5	0	3	12	8	12

Tableau 4.2 Nouvelles paramètres temporelles de tâches dépendantes

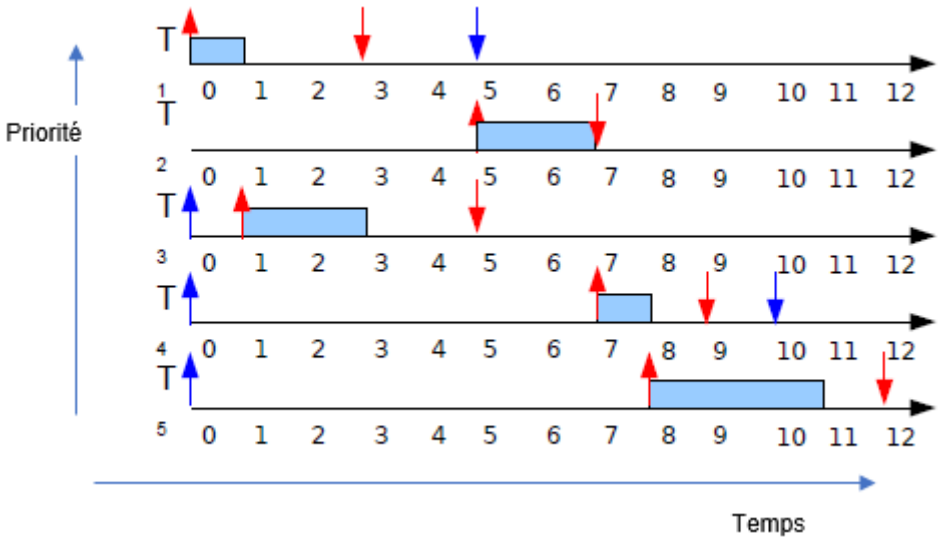
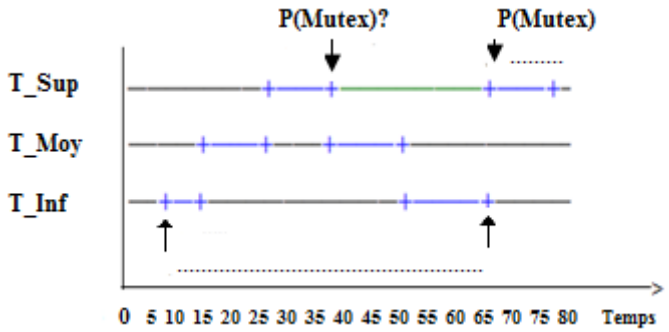


Figure 4.14 *Ordonnancement EDF des tâches sous contraintes de précédence (En couleur rouge les valeurs initiales des contraintes temporellesn en couleur bleu:les nouvelles valeurs)*

4.7. EXERCICES

Exercice 1 :

Considérons un système avec 3 tâches, ces tâches sont appelées T_Inf, T_Moy et T_Sup et elles ont respectivement les priorités inferieure, moyenne et supérieure.



1. Commenter le graphe ci-dessus en donnant un scénario d'exécution
2. Décrit le problème illustré dans ce graphisme.
3. Donner des solutions.

Exercice 1 :

solution

1. Le scénario d'exécution est La séquence suivante :
    1. T_Inf est activée, elle s'exécute et accède à la ressource Mutex.
    2. T_moy est activée et préempte T_Inf qui est en train d'utiliser Mutex. T_moy prend 100% CPU car il a la priorité sur T_Inf.
    3. T_Sup est activée et préempte T_Moy.
    4. T_Sup demande l'accès à Mutex.

Toutefois, T_Sup ne peut accéder à Mutex car T_Inf en a l'accès exclusif. Donc T_Sup est suspendu. La tâche restante de plus haute priorité est T_Moy et T_Moy continue donc à s'exécuter. En conséquence, T_Sup doit attendre que T_Moy termine avant de pouvoir s'éxécuter à nouveau.

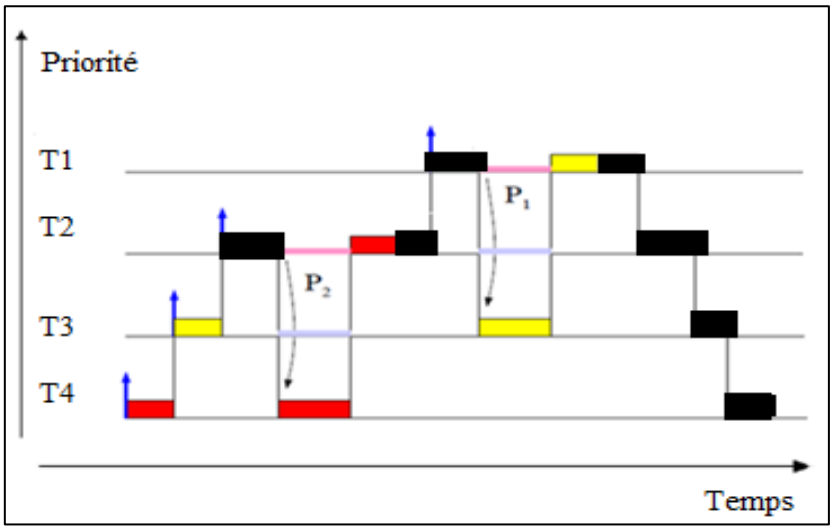
2. Il s'agit du phénomène s'appelant l'inversion de priorité.

- ne pas trop prioriser les threads de priorité supérieure. (Pas valable en temps réel)
- aléatoirement, donner du temps CPU à des taches de priorité basse.

- (priority ceiling) donner une priorité à un mutex. Si un thread bloque le mutex, il hérite (temporairement) de la priorité du mutex (si elle est plus haute)
- (priority inherance) si un thread T_Inf bloque le mutex, et un thread T_Sup de plus haute priorité demande le mutex, T_Inf hérite (temporairement) de la priorité de T_Sup.

Exercice 2

Soit le graphe présenté comme suit :



1. Commenter le graphe suivant présentant les tâches T1, T2, T3 et T4 partagent des ressources. P1, P2, P3 et P4 sont les priorités respectives de T1, T2, T3 et T4 sachant que P1>P2>3>T4.
2. Donner un titre à ce graphe.

Exercice3

Soit le graphe de tâches suivant modèlisant une application multimedia (la réference de cette application est dans la bibliographie)

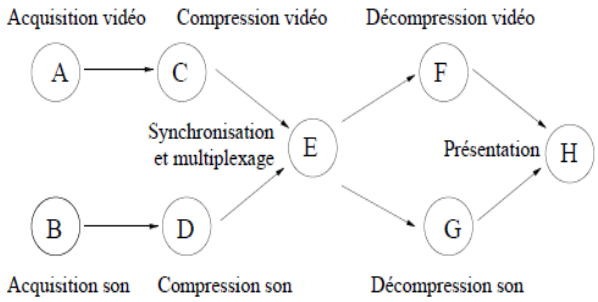


Figure 4.15 *Application multimedia*

les traitements de cette application sont présentés par la figure 4.15. Les paramètres temporels de cette application sont donnés ci-dessous (exprimés en milli-secondes) :

Tâches	Si	Ci	Di
A	0	3	12
B	0	1	10
C	0	3	20
D	0	1	15
E	0	5	25
F	0	4	37
G	0	1	20
H	0	1	40

1. Peut-on utiliser algorithme EDF pour ordonnancer ce système de tâches ? justifier votre réponse.
2. En se basant sur le principe de la méthode de Blazewicz/Chetto,
   - Etudier l'ordonnançabilité de ce système de tâches par EDF préemptif sachant que Le graphe de tâches est activé périodiquement toutes les 40ms.
   - Générer son ordonnancement.

.
.

5

Conception d’application temps réel

5.1 Introduction

Dans ce chapitre, nous sommes interessés à la conception d'application temps réel. L'application est un algorithme d'ordonnancement des tâches temps réel. Ces algorithmes (applications) concernent les algorithmes d'ordonnancement des tâches monoprocesseurs et les algorithmes d'ordonnancement multiprocesseurs. Les tâches sont des tâches périodiques, apériodiques, dépendantes par le partage des ressources et par des liens de précédence. UML (Unified modeling Langage) est l'un des outils de modélisation adoptés pour les applications temps reel.

5.2 UML et son rôle dans la modélisation et la conception orientées objet

Les langages orientés objet dominent le monde de la programmation parce qu'ils modélisent des objets du monde réel. L'UML combine plusieurs notations orientées objet : Object- Oriented Design (conception orientée objet), Object Modeling Technique (technique de modélisation objet) et Object-Oriented Software Engineering (génie logiciel orienté objet). D'une façon général, UML est une représentation standardisée d'un système orienté objet. UML se base sur une notation graphique normalisée pour modéliser des systèmes objets. Cette notation graphique est regroupée dans trois familles de diagrammes:

- les diagrammes statiques (diagrammes de classes, d'objet et de cas d'utilisation)
- les diagrammes dynamiques (diagrammes d'activité, de collaboration, de séquence, d'état-transitions et de cas d'utilisation)
- les diagrammes d'architecture : (diagrammes de composants et de déploiements

5.3 Concepts orientés objet dans le langage UML

Les objets dans UML sont des entités du monde réel qui existent autour de nous. Dans le développement de logiciels, les objets peuvent être utilisés pour décrire ou modéliser le système en cours de création sous un angle pertinent. Les objets permettent également la décomposition des systèmes complexes en éléments compréhensibles qui permettent de construire les pièces une par une.
Voici quelques concepts fondamentaux d'un monde orienté objet :

- **Objet** représente une entité et le module de base.
- **Classe** représente le plan d'un objet

- **Abstraction** est le comportement d'une entité du monde réel
- **Encapsulation** est un mécanisme consiste à relier les données et à les cacher du monde extérieur
- **Héritage** permet de créer de nouvelles classes à partir d'une classe existante .
- **Polymorphisme** représente la faculté d'une méthode à pouvoir s'appliquer à des objets de classes différentes.
- **Interface** est une spécification des opérations visibles d'une classe ou de toutes autres éléments, tels que des paquetages Une interface ne spécifie souvent qu'une partie limitée du comportement de la classe

5.4 Modélisation avec UML

Les diagrammes de paquetage et diagramme de classes logique déployés sont présentés dans les sections suivantes:

5.4.1 Diagramme de paquetage déployé

Déployer des packages pour faciliter la recherche ou la localisation des classes et des interfaces, éviter les conflits des noms de classe et pour réutiliser les classes contenues dans les packages d'autres programmes.

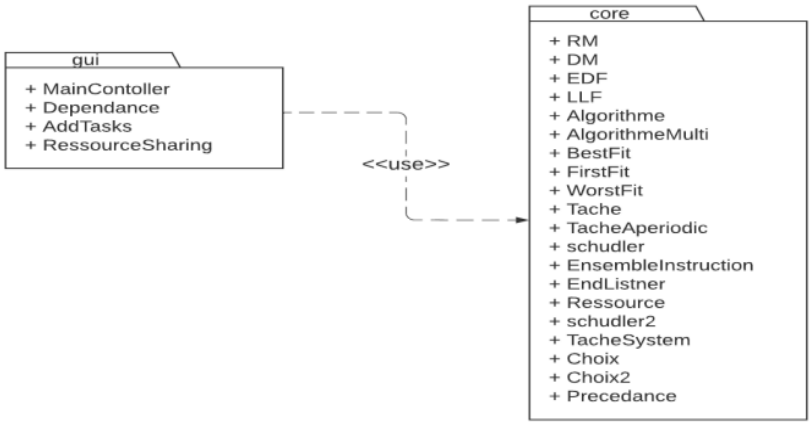


Figure 5.1: *Diagramme de paquetage déployé*

5.4.2 Diagrammes de classes logiques déployés

Classe: Letableau 5.1 présente les classes et leurs caractéristiques déployés

Classes	Attribues	Operations
RM	name	
DM	name	
EDF	name	
LLF	name	
BestFit	name	
WorstFit	name	
FirstFit	name	
Schudler	algorithme, taches,dure,TA, endListener, group,rcrs, msgpremption,Type,tacheS, analysePrecedance,Mode,	Run,start tachePrioriteur TestExistTacheA
EnssembleInstruction	wait, isRun, terminal.	Run,resume,arrete
Tache	idTask, name, p, c, r, d, restC, wait, temp, occupe, msgviolet, instanceviolet, ensembleInstuction, chronogramme ,w preced,apres,srcTime,src	execute isWait violet
TacheAperiodic	name,c,chronogramme restC, occupe,etat, ensembleInstuction,r	execute
TacheSystem	name,p,c,chronogrammer, restC, wait, temp, occupe, msgviolet, instanceviolet, ensembleInstuction, d	execute isWait getPriority,
Choix	source,sink	
Choix2	time,tache	
schudler2	algorithme ,taches,hp,dure, endListener , group,type, msgpremption , rcrs,T,s,	run,start tachePrioriteur Sharing,searche
Precedance		reclaculeR,recalculeD recalculeP,searche
Ressource	nom , tache,chronogramme	

Tableau 5.1:Algorithmes d'ordonnancement des tâches dans les systèmes temps réel multi processeur

Interface: Les interfaces utilisées sont presents dans le tableau 5.1:

Interface	Attribue	Operations
Algorithme		isOrdonnensable
AlgorithmeMulti		toGroupes
EndListener		OnEnd

Tableau 5.2: ***Liste des interfaces***

5.4.3 Relations de réalisation d'interface

Dans les diagrammes UML, une relation de réalisation d'interface est un type spécialisé de relation d'implémentation entre un discriminant et une interface fournie. La relation de réalisation d'interface spécifie que le discriminant réalisant doit se conformer au contrat spécifié par l'interface fournie. Comme le montre la figure suivante, une relation de réalisation d'interface est affichée dans l'éditeur de diagramme comme une ligne tiretée avec une pointe de flèche vide. La réalisation d'interface est dirigée du discriminant vers l'interface fournie.

Interface Algorithme

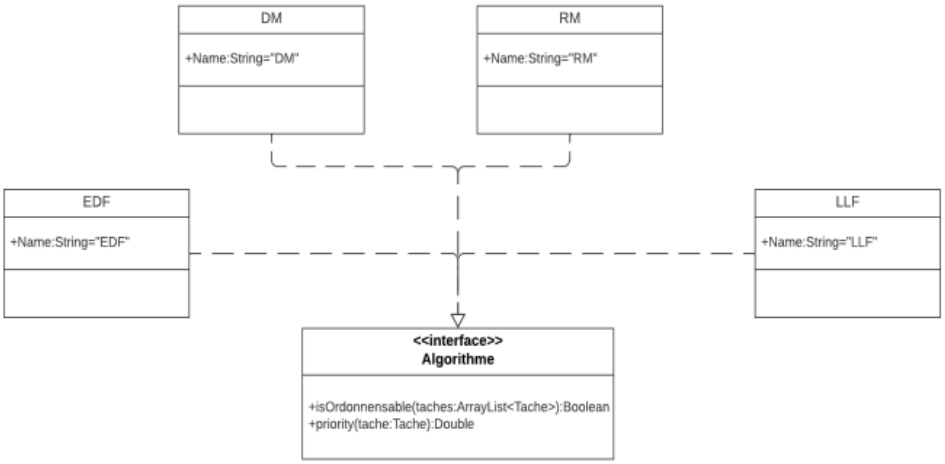


Figure 5.3: Relations de réalisation de l'interface Algorithme

Interface AlgorithmeMulti

Interface pour le cas d'ordonnancement dans les systèmes multiprocesseurs

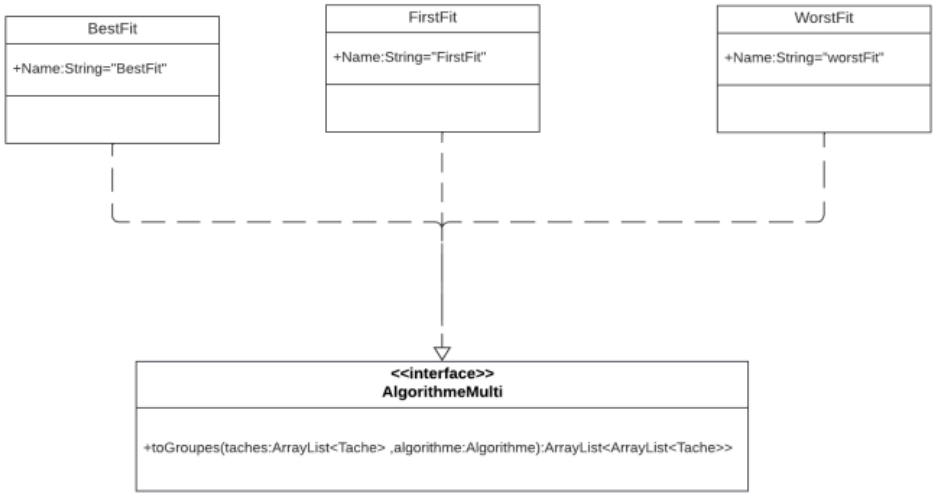


Figure 5.4: *Relations de réalisation de l'interface AlgorithmeMulti*

Interface Runnable

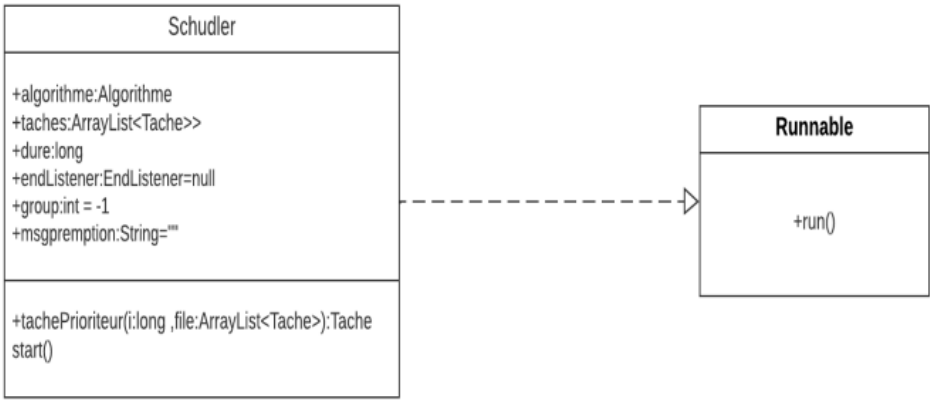


Figure 5.5: *Relations de réalisation de l'interface Runnable*

5.4.4Diagramme de classes déployé :

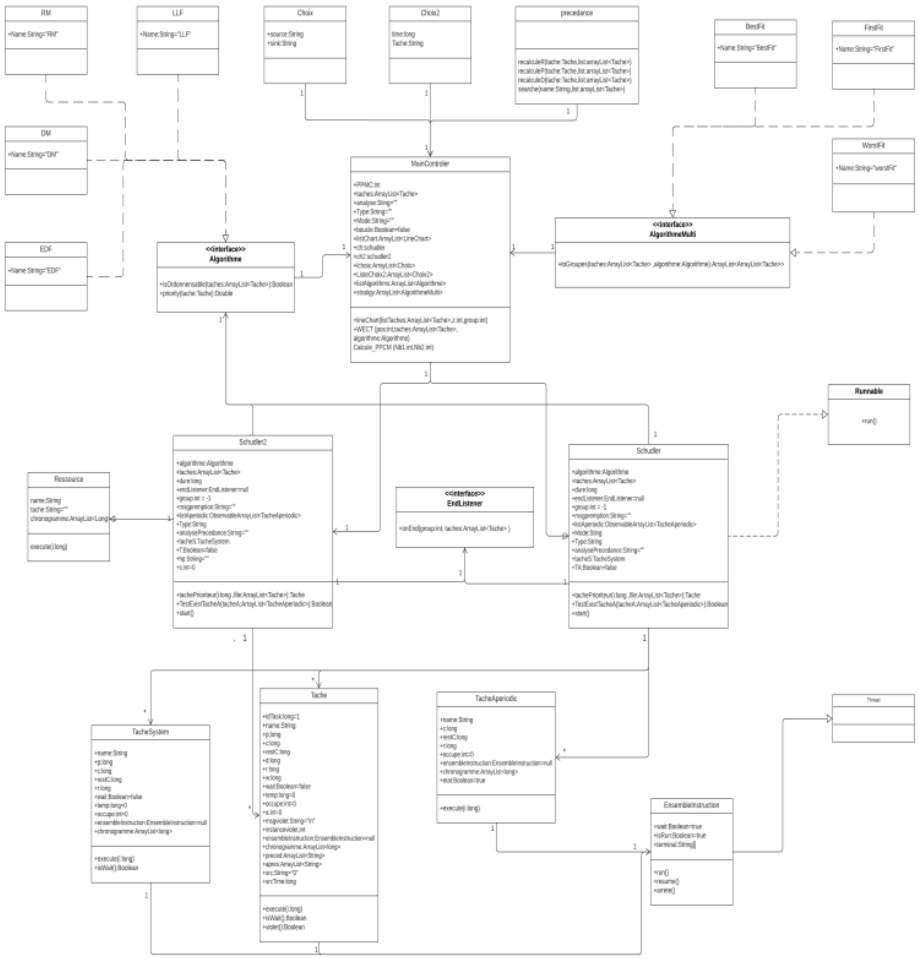


Figure 5.6 : *Diagramme de classes de l'application : ordonnancement des tâches dans les systèmes temps réels*

5.5 Mise en oeuvre des algorithmes

- **Ordonnacement des taches apériodiques selon la stratégie Arriere-plan**

  Exemple:

Title	R	P	C	D
affiche	0	5	2	5
calcule	0	12	2	10
ecrire	3	0	2	0
Lire	7	0	1	0
grouper	14	0	2	0

FIgure 5.7: *Exemple de modèle de tâches temps reel (R: temps d'arrivée:R;C: charge;Ddeadline;P: période)*

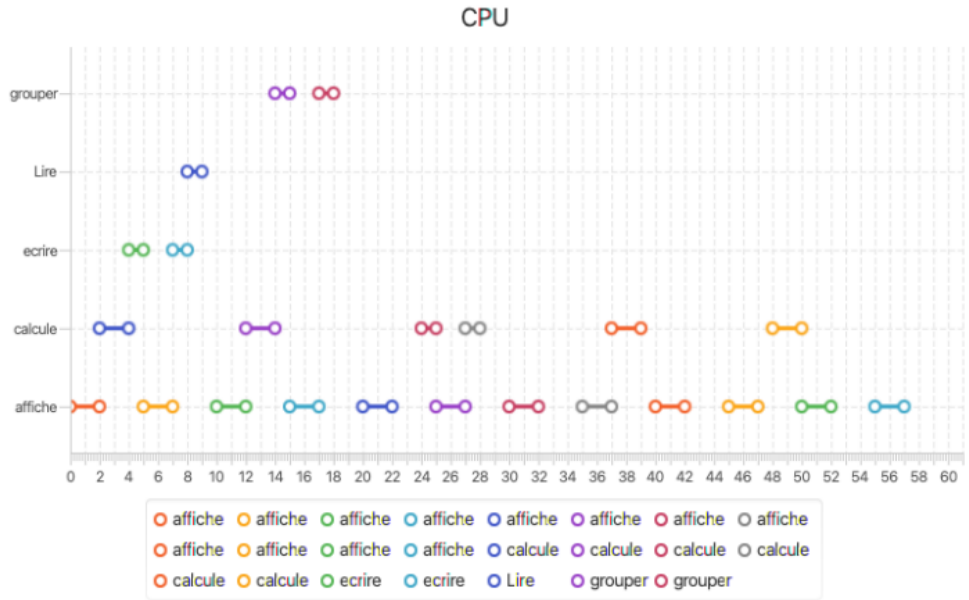


Figure 5.8: ***Ordonnancement de tâches apériodiques par arrière plan***

Analyse
Temps de repense de chaque tache
W(calcule) = 2
W(affiche) = 4

L'execution des taches :
affiche, affiche, calcule, calcule, ecrire, affiche, affiche, ecrire, Lire, affiche, affiche, calcule,
calcule, grouper, affiche, affiche, grouper, affiche, affiche, calcule, affiche, affiche, calcule, affiche,
affiche, affiche, affiche, calcule, calcule, affiche, affiche, affiche, affiche, calcule, calcule, affiche,
affiche, affiche, affiche,

La langueur d'ordonnacement PPMC :60
Nombre de creux : 21
Taux d'occupation de chaque taches periodic:
affiche = 24 , calcule = 10 ,
Taux d'occupation de chaque taches aperiodic:
ecrire = 2 , Lire = 1 , grouper = 2 ,
la premption :
affiche est plus prioritaire causer une premption sur ecrire dans l'instance 6
affiche est plus prioritaire causer une premption sur Lire dans l'instance 11
affiche est plus prioritaire causer une premption sur grouper dans l'instance 16
affiche est plus prioritaire causer une premption sur grouper dans l'instance 21

- **Ordonnacement des taches apériodiques selon la stratégie scrutation (Polling)**

Exemple:

Title	R	P	C	D
TacheAperiodique	1	0	3	0
TachePeriodique	0	8	2	5
TachePeriodique2	0	10	1	5
TacheSysteme	0	8	3	0
TacheAperiodique2	1	0	4	0

Figure 5.7: *Exemple de modèle de tâches temps reel*
(R: temps d'arrivée:R;C: charge;Ddeadline;P: période)

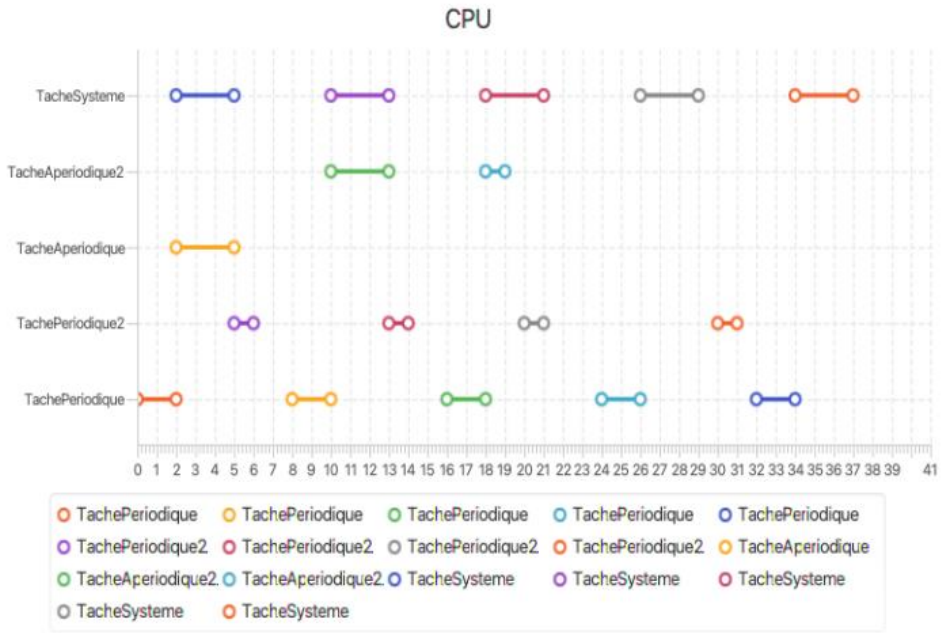


Figure 5.10: ***Ordonnacement des tâches apériodiquespar stratégie polling***

Analyse
Temps de repense de chaque tache
W(TachePeriodique2) = 1
W(TachePeriodique) = 3
W(T22) = 6

L'execution des taches :
TachePeriodique, TachePeriodique, TacheAperiodique, TacheSysteme, TacheAperiodique, TacheSysteme, TacheAperiodique, TacheSysteme, TachePeriodique2, TachePeriodique, TachePeriodique, TacheAperiodique2, TacheSysteme, TacheAperiodique2, TacheSysteme, TacheAperiodique2, TacheSysteme, TachePeriodique2, TachePeriodique, TachePeriodique, TacheAperiodique2, TacheSysteme, TacheSysteme, TachePeriodique2, TacheSysteme, TachePeriodique, TachePeriodique, TacheSysteme, TacheSysteme, TacheSysteme, TachePeriodique2, TachePeriodique, TachePeriodique, TacheSysteme, TacheSysteme, TacheSysteme,

La langueur d'ordonnacement PPMC :40
Nombre de creux : 19
Taux d'occupation de chaque taches periodic:
TachePeriodique = 10 , TachePeriodique2 = 4 ,
Taux d'occupation de chaque taches aperiodic:
TacheAperiodique = 3 , TacheAperiodique2 = 4 ,

- **Ordonnacement des taches apériodiques selon la stratégie sporadique**
  Exemple :

Title	R	P	C	D
TacheAperiodique	1	0	3	0
TachePeriodique	0	8	2	5
TachePeriodique2	0	10	1	5
TacheSysteme	0	8	3	0
TacheAperiodique2	1	0	4	0

***Figure 5.12**: Exemple de modèle de tâches temps réel (R: temps d'arrivée:R;C: charge;Ddeadline;P: période)*

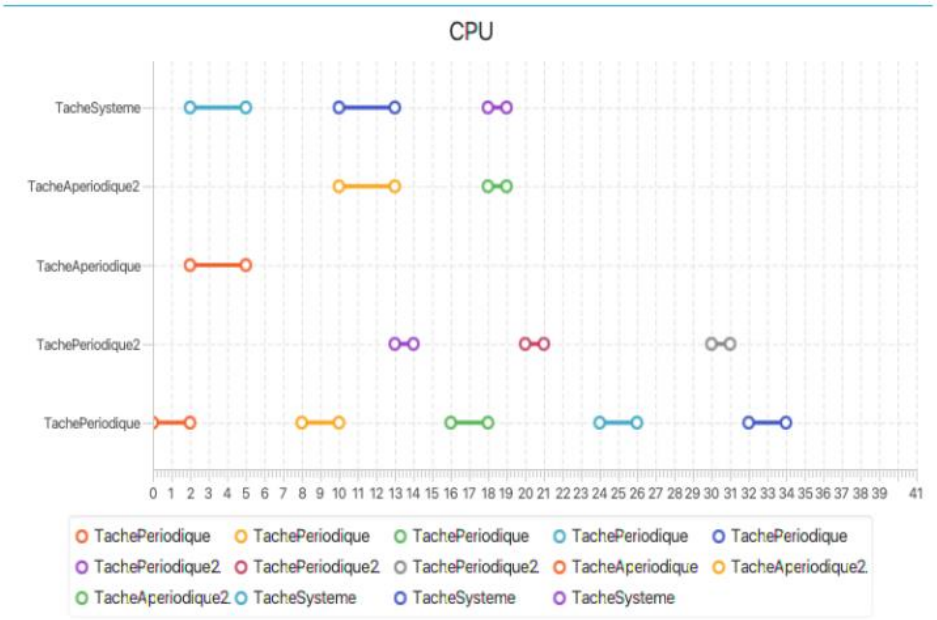


***Figure 5.13**: Ordonnacement des taches apériodiques selon la stratégie sporadique*

Analyse
Temps de repense de chaque tache
W(TachePeriodique2) = 1
W(TachePeriodique) = 3
W(T3) = 6

L'execution des taches :
TachePeriodique, TachePeriodique, TacheAperiodique, TacheSysteme, TacheAperiodique,
TacheSysteme, TacheAperiodique, TacheSysteme, TachePeriodique, TachePeriodique,
TacheAperiodique2, TacheSysteme, TacheAperiodique2, TacheSysteme, TacheAperiodique2,
TacheSysteme, TachePeriodique2, TachePeriodique, TachePeriodique, TacheAperiodique2,
TacheSysteme, TachePeriodique2, TachePeriodique, TachePeriodique, TachePeriodique2,
TachePeriodique, TachePeriodique,

La langueur d'ordonnacement PPMC :40
Nombre de creux : 20
Taux d'occupation de chaque taches periodic:
TachePeriodique = 10 , TachePeriodique2 = 3 ,
Taux d'occupation de chaque taches aperiodic:
TacheAperiodique = 3 , TacheAperiodique2 = 4 ,
Les instances violets des tache periodic :
TachePeriodique2 instance violet dans i = 5 le reste de la charge est 1

la premption :
TacheSysteme est plus prioritaire causer une premption sur TachePeriodique dans l'instance 18
TachePeriodique est plus prioritaire causer une premption sur TacheSysteme dans l'instance 25

- **Ordonnacement des taches dépendantes aved liens de précedence**

Exemple:

Title	R	P	C	D
T1	0	6	1	5
T2	5	8	2	7
T3	0	6	2	5
T4	0	12	1	10
T5	0	12	3	12

Figure 5.15: *Exemple de modèle de tâches temps reel (R: temps d'arrivée:R;C: charge;Ddeadline;P: période)*

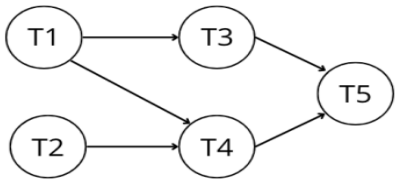


Figure 5.16: *Tâches dépendantes : Graphe de précedence*

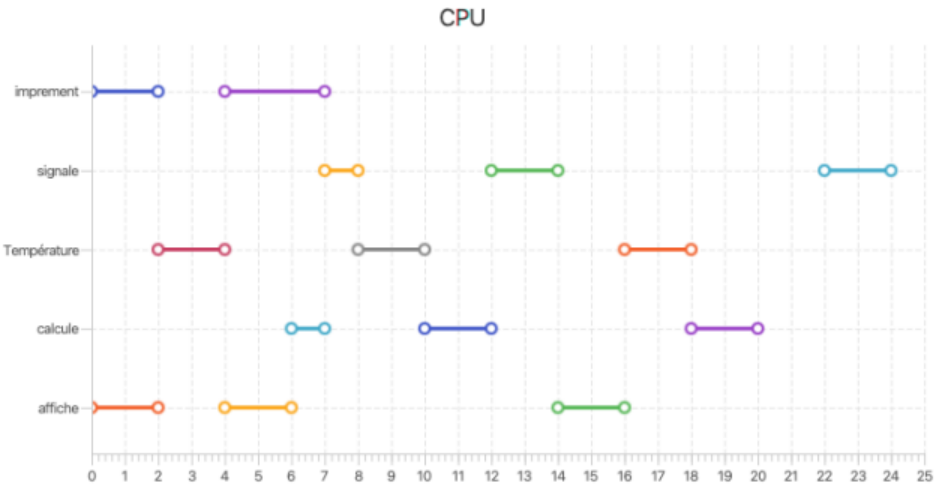


Figure 5.16: *– Ordonnacement des tâches avec liens de précedence*

Analyse
Temps de repense de chaque tache
W(T4) = 1
W(T5) = 4
W(T2) = 6
W(T1) = 7
W(T3) = 15
Precedance
Recalculation des nouveaux contraintes :
T1
D = 5
R = 0
T2
D = 7
R = 0
T3
D = 5
R = 0
T4
D = 10
R = 0
T5
D = 12
R = 0
L'execution des taches :
T1, T3, T3, T2, T2, T4, T1, T3, T3, T2, T2, T5, T1, T3, T3, T4, T2, T2, T1, T3, T3, T5, T5, T5,

La langueur d'ordonnacement PPMC :24
Nombre de creux : 0
Taux d'occupation de chaque taches periodic:
T1 = 4 , T2 = 6 , T3 = 8 , T4 = 2 , T5 = 4 ,
Taux d'occupation de chaque taches aperiodic:

Les instances violets des tache periodic :
T5 instance violet dans i = 12 le reste de la charge est 2

Ordonnacement des taches dépondantes avec e partage de ressources (héritage de priorité)

Exemple:

Title	R	P	C	D
affiche	0	12	4	8
calcule	1	8	2	6
Température	2	6	2	6
signale	3	8	2	8

Figure 4.31 : Exemple de modèle de tâches temps reel
(R: temps d'arrivée:R;C: charge;Ddeadline;P: période)

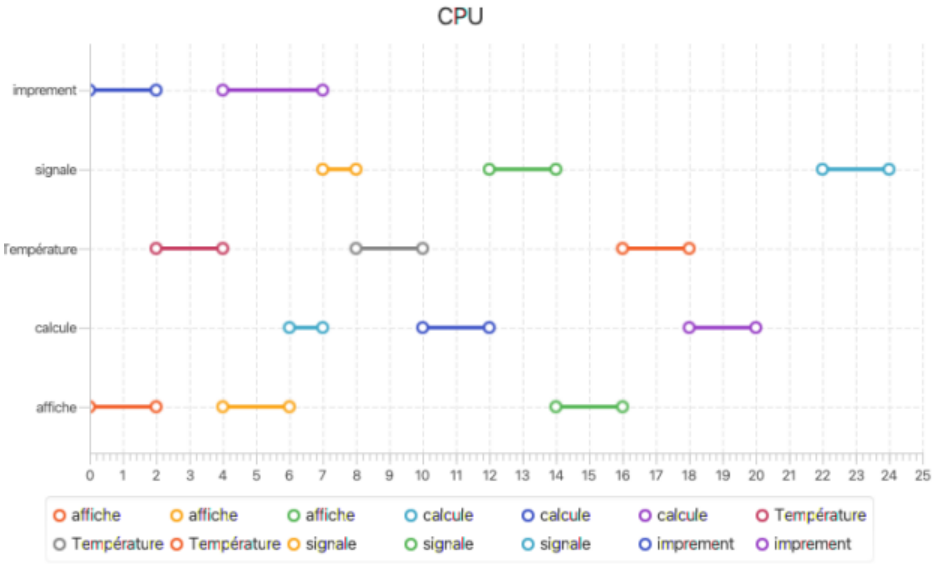


Figure 5.17: *Ordonnancement des tâches avec lieu de péecedence*

Analyse
Temps de repense de chaque tache
W(affiche) = 4
W(calcule) = 6
W(signale) = 8
W(Température) = 22
L'execution des taches :
affiche, affiche, Température, Température, affiche, affiche, calcule, signale, Température, Température, calcule, calcule, signale, signale, affiche, affiche, Température, Température, calcule, calcule, signale, signale,

La langueur d'ordonnacement PPMC :24
Nombre de creux : 2
Taux d'occupation de chaque taches periodic:
affiche = 6 , calcule = 5 , Température = 6 , signale = 5 ,
Taux d'occupation de chaque taches aperiodic:

Les instances violets des tache periodic :
affiche instance violet dans i = 20 le reste de la charge est 2
calcule instance violet dans i = 7 le reste de la charge est 1
signale instance violet dans i = 11 le reste de la charge est 1

la premption :
Température est plus prioritaire causer une premption sur affiche dans l'instance 2

Héritage de priorité :
heritage de priorité entre la tache calcule et la tache affiche dans i=1

Exemples sur les algorithmes d'ordonnancement des tâches environement : Monoprocesseur
Exemple

Title	R	P	C	D
affiche	0	12	4	8
calcule	1	8	2	6
Température	2	6	2	6
signale	3	8	2	8

Table 5.8 : Exemple d'ordonnancement selon RM

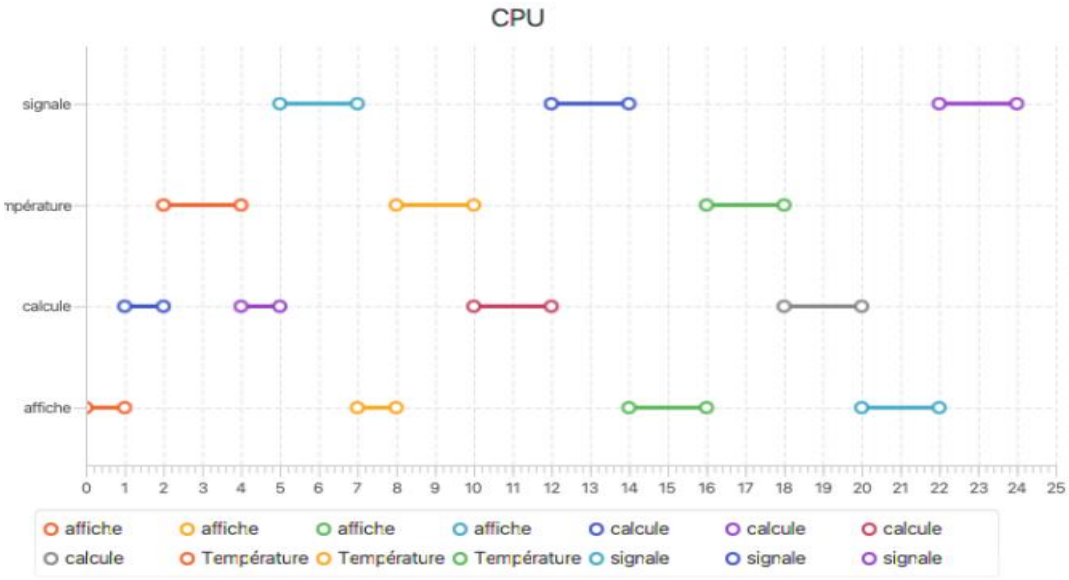


***Figure 5.19**: Résultat d'ordonnancement avec RM*

Analyse d'ordonnancement :

Analyse
Temps de repense de chaque tache
W(affiche) = 4
W(calcule) = 6
W(signale) = 8
W(Température) = 22

L'execution des taches :
affiche, calcule, Température, Température, calcule, signale, signale, affiche, Température, Température, calcule, calcule, signale, signale, affiche, affiche, Température, Température, calcule, calcule, affiche, affiche, signale, signale,

Analyse
Temps de repense de chaque tache
W(affiche) = 4
W(calcule) = 6
W(signale) = 8
W(Température) = 22

L'execution des taches :
affiche, calcule, Température, Température, calcule, signale, signale, affiche, Température, Température, calcule, calcule, signale, signale, affiche, affiche, Température, Température, calcule, calcule, affiche, affiche, signale, signale,

La langueur d'ordonnacement PPMC :24
Nombre de creux : 0
Taux d'occupation de chaque taches periodic:
affiche = 6 , calcule = 6 , Température = 6 , signale = 6 ,
Taux d'occupation de chaque taches aperiodic:

Les instances violets des tache periodic :
affiche instance violet dans i = 8 le reste de la charge est 2

la premption :
calcule est plus prioritaire causer une premption sur affiche dans l'instance 1
Température est plus prioritaire causer une premption sur calcule dans l'instance 2

Figure 5.20: Analyse de l'exemple RM

Ordonnancement par la statégie Deadline monotonic (DM)

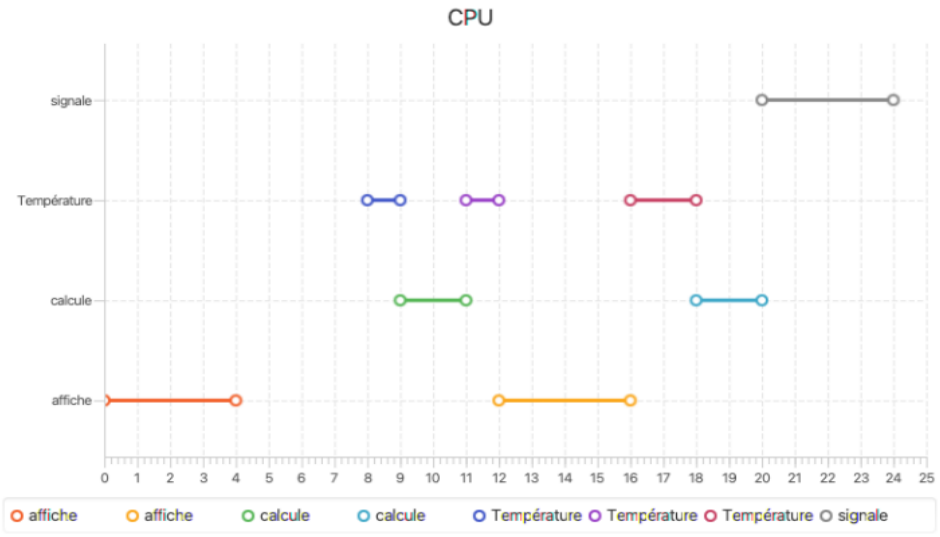


Figure 5.21: ***Ordonnacement des tâches selon l'algorithme Deadline Monotic (DM)***

Analyse
Temps de repense de chaque tache
W(affiche) = 4
W(signale) = 6
W(Température) = 8
W(calcule) = 10

L'execution des taches :
affiche, affiche, affiche, affiche, Température, calcule, Température, affiche, affiche, affiche,
affiche, Température, Température, calcule, calcule, signale, signale, signale, signale,

La langueur d'ordonnacement PPMC :24
Nombre de creux : 4
Taux d'occupation de chaque taches periodic:
affiche = 8 , calcule = 4 , Température = 4 , signale = 4 ,
Taux d'occupation de chaque taches aperiodic:

Les instances violets des tache periodic :
Température instance violet dans i = 8 le reste de la charge est 2
signale instance violet dans i = 11 le reste de la charge est 2

Ordonnancement Par Earliest deadline First (EDF)

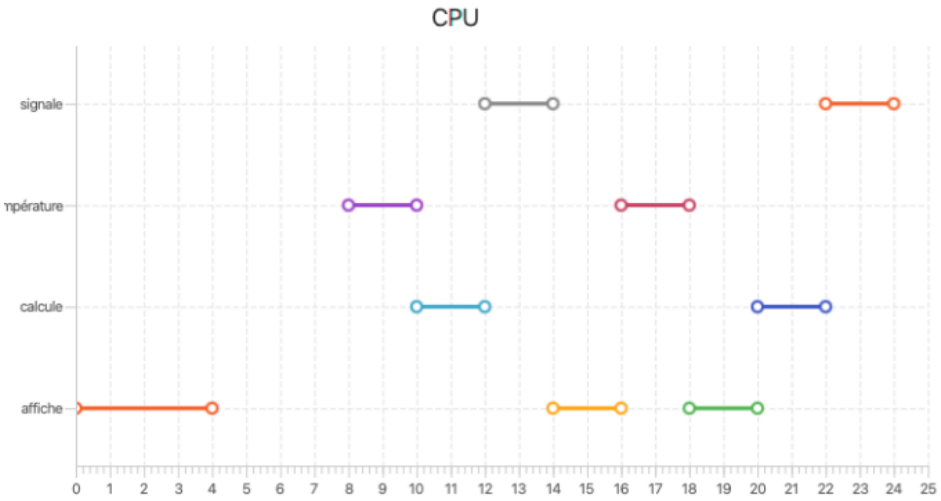


Figure 5.22: l'algorithme Earliest deadline First (EDF)

Analyse
Temps de repense de chaque tache
W(calcule) = 2
W(affiche) = 4
W(Température) = 2
W(signale) = 4

L'execution des taches :
affiche, affiche, affiche, affiche, Température, Température, calcule, calcule, signale, signale,
affiche, affiche, Température, Température, affiche, affiche, calcule, calcule, signale, signale,

La langueur d'ordonnacement PPMC :24
Nombre de creux : 4
Taux d'occupation de chaque taches periodic:
affiche = 8 , calcule = 4 , Température = 4 , signale = 4 ,
Taux d'occupation de chaque taches aperiodic:

Les instances violets des tache periodic :
Température instance violet dans i = 8 le reste de la charge est 2
signale instance violet dans i = 11 le reste de la charge est 2

la premption :
Température est plus prioritaire causer une premption sur affiche dans l'instance 16

Ordonnancement Par Earliest deadline First (EDF)

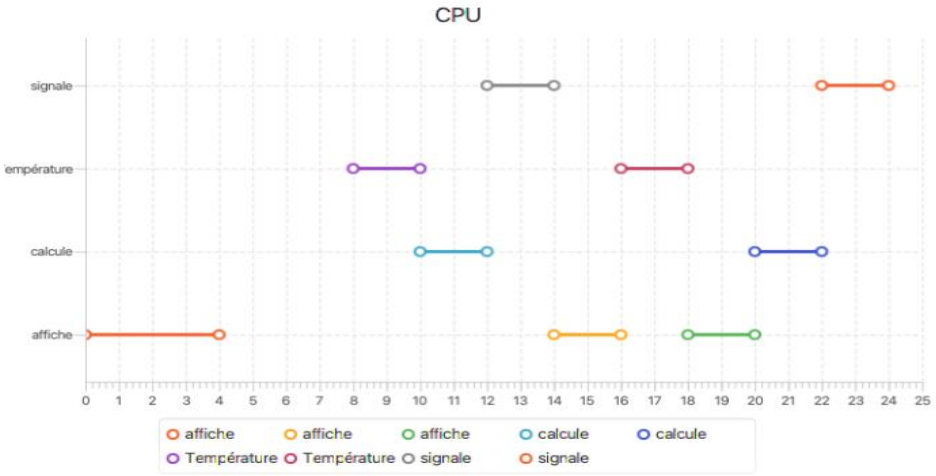


***Figure 5.24:** Ordonnacement des tâches selon l'algorithme Least Laxity First (LLF)*

Analyse
Temps de repense de chaque tache
W(calcule) = 2
W(affiche) = 4
W(Température) = 2
W(signale) = 4

L'execution des taches :
affiche, affiche, affiche, affiche, Température, Température, calcule, calcule, signale, signale,
affiche, affiche, Température, Température, affiche, affiche, calcule, calcule, signale, signale,

La langueur d'ordonnacement PPMC :24
Nombre de creux : 4
Taux d'occupation de chaque taches periodic:
affiche = 8 , calcule = 4 , Température = 4 , signale = 4 ,
Taux d'occupation de chaque taches aperiodic:

Les instances violets des tache periodic :
Température instance violet dans i = 8 le reste de la charge est 2
signale instance violet dans i = 11 le reste de la charge est 2

la premption :
Température est plus prioritaire causer une premption sur affiche dans l'instance 16

3

Sujets d’éxamens

[Systèmes temps rée].
[Université Mustapha Stambouli-Mascara-].2022.

Examen Master I (spécialité : ISI) Nom :……

2018-2019 Prénom :……

---

1. Définir la notion de la validation logique
   ……
   ……
2. Définir la notion de la validation temporelle
   ……
   ……
   ……
3. Citer les algorithmes d'ordonnancement temps réel multiprocesseur ainsi que leur compléxité. ……
   ……
   ……
   ……
   ……
   ……
4. Soit la fonction suivante :

$$\omega_i^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{\omega_i^n}{Pe_j} \right\rceil * C_j$$

3.1 De quelle fonction s'agit-il ?

:……
……

3.2 Définir chaque terme de cette fonction

……
……
……
……
……

ω_i^{n+1}

3.3 Donner l'algorithme du processus itératif

……
……
……
……
……
……
……

3.3 Dans quel cas : ……

$\omega_i^{n+1} = C_i$

5. Soit le système de tâches suivant :

Tâche	C_i	Pe_i
A	3	10
B	4	15
C	2	20

4.1 Calculer les temps de réponse de chaque tâche sachant que le système de tâches est preemptible et ordonnançable avec RM

……
……
……
……
……
……
……
……
……
……

6. Remplir le tableau comparatif suivant :

Algorithme	RM	EDF
- Définir l'algorithme	……	……
- Calcul de priorité	……	……
- Election de tâche	……	……
- Condition d'ordonançabilité	……	……
- Interval de travail	……	……

En considère un ensemble de 6 tâches {A,B,C,D,E, F}.La tâche A doit précéder les tâches B ,C ,D. Les tâches B et C doivent précéder la tâche E. Les tâches D et E doivent précéder la tâche F. 2

Proposer un programme pour ordonnancer ce modèle de tâches : .

……
……
……
……
……
……
……
……
……

Département d'informatique université de Mascara

Examen Master I (spécialité : ISI) /20-21 Nom & Prénom :……

1. Citre les approches de vérification de l'ordonnançabilité.

:……

……

……

……..

2. Commenter les digrammes suivants:

Diagramme 1

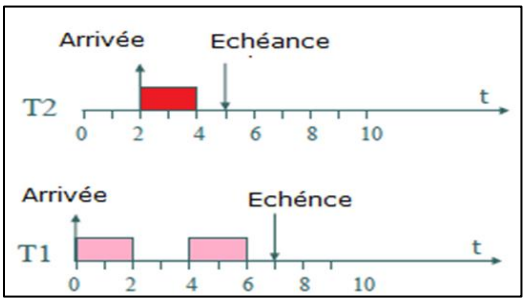


Diagramme 2

……
……
……
……
……
……
……
……

Soit le système de tâches temps réel à échéance sur requêtes suivants :

T1 : (C=1, D=3) T2 : (C=1, D=4) T3 : (C=2, D=6)

2.1 Calculer l'intervalle de travail

……
……

2.2 Calculer le temps d'occupation du processeur par le système de tâches TR

……
……

2.3 Donner l'ordonnançabilité par RM :

……
……
……

2.4 Calculer les WCET des tâches temps réel

……
……
……
……
……
……
…… …….. ……
……

……
……
……
……
……
……

2.5 donner le nombre de switch pour chaque tâche

……
……

3. Compléter le tableau suivant :

Fonctions	Objectif	Paramètres	Contexte du système de tâches temps réel
$\sum_{j \in hp(i)} \left\lceil \frac{Ri}{Pj} \right\rceil$			
$Ii = \sum_{j \in hp(i)} \left\lceil \frac{Ri}{Pj} \right\rceil Cj$			
$Ri = Ci + \sum_{j \in hp(i)} \left\lceil \frac{Ri}{Pj} \right\rceil Cj$			
$\frac{1}{n} \sum_{i=1}^{n} (f_i - a_i)$			
$max_i(f_i) - min_i(a_i)$			
$max_i(f_i - d_i)$			
$P(T) = \sum_{i=1}^{l} P(\tau_{S,i})$			
$R(T) = \sum_{i=1}^{k} R(\tau_{F,i})$			

1. Décrire les deux figures : les légendes de la figure ci-dessous et donner son titre.

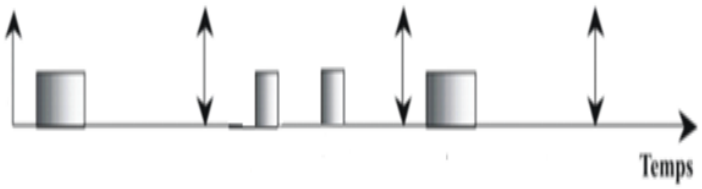


Figure 1 :

……
……
……
……

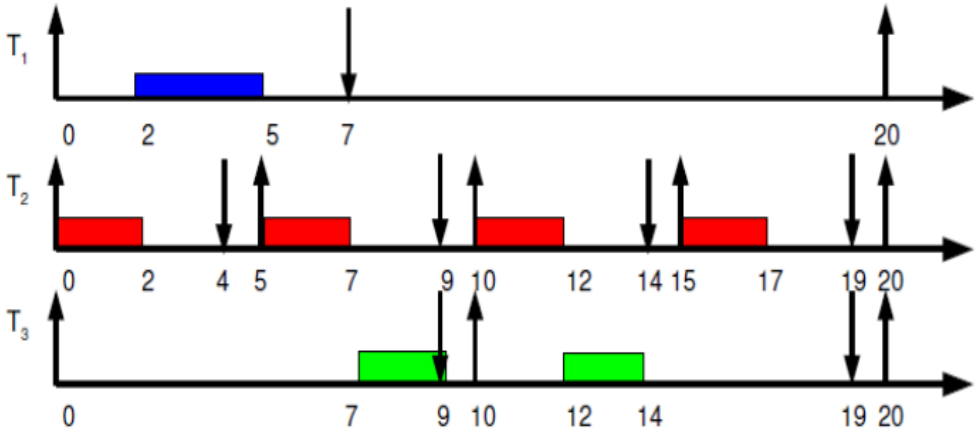


Figure2 :

……
……
……
……
……
……

2. Soit le système de tâches temps réel suivants à ordonnancer avec RM
T1(0,2,6,6), T2 (0,2,9,9),T3 (0,3,12,12)

3.1 Calculer l'intervalle de travail

……
……

3.2 Le système de tâches est il l'ordonnançable par RM ? :

……
……
……
……
……
……

3.3 Calculer les WCET des tâches temps réel

……
……
……
……
……
……
…… …… ……
……
……
……
……
……

4 Compléter le tableau suivant :

Fonctions	Objectif	Paramètres
$Ri=Ci+\sum_{j\in hp(i)}\left\lceil \frac{Ri}{Pj}\right\rceil Cj$		
$\sum_{i=1}^{n}\frac{C_i}{P_i}+\frac{C_s}{P_s}\leq (n+1)\left(2^{\frac{1}{n+1}}-1\right)$		
$\sum_{j=1}^{j=n} C_j/P_j + B_n/P_n \leq n(2^{\frac{1}{n}}-1)$		
$\forall i,j$ **Si** $j<i$ **Alors** $r_i^* = Max\{r_i, Max\{r_j^* + C_j\}$ $\forall i,j$ **Si** $i<j$ **Alors** $d_i^* = Min\{d_i, Min\{d_j^* - C_j\}$		

Examen : Systèmes temps réel Année : 2016-2017

A.Dans une application temps réel embarquée :

1. Définir la notion de la validation logique

   Exactitude des résultats

   Définir la notion de la validation temporelle

   Résultats au temps adéquat sans dépasser le délais critique (deadline)

B. Soit un système de visualisation intégré dans un véhicule automobile. Nous devons étudier le respect des contraintes temporelles des tâches et les différentes synchronisations entre les tâches grâce à un outil spécifié par exemple réseau de pétri. Ce système est constitué d'un ensemble de tâches. Ces tâches permettent :

- la lecture de la vitesse du véhicule pendant une durée de 2 millisecondes toutes les 10 millisecondes, de la température dans l'habitacle du véhicule pendant une durée d'1 millisecondes toute les 10 millisecondes et la position du GPS du véhicule pendant une durée de 4 millisecondes toutes les 40 millisecondes et
- L'affichage pendant une durée de 2 millisecondes toutes les 12 millisecondes qui produit un résumé des informations produites des tâches des différentes lectures ci-dessus. Un autre affichage de la carte routière est réalisé pendant une durée d'1 milliseconde selon la demande de l'utilisateur toutes les 6 millisecondes .

1 De quel type d'application s'agit-il ?

Application temps réel embarquée

2 Citer les différentes tâches considérées avec leurs paramètres

1. ***C1 : La tâche CAPTEUR_1 qui lit toutes les 10 millisecondes la vitesse du véhicule. (periode1, capacité1)***
2. ***C2 : La tâche CAPTEUR_2 qui lit toutes les 10 millisecondes la température dans l'habitacle du véhicule. (periode2, capacité2)***
3. ***C3 : La tâche CAPTEUR_3 qui lit toutes les 40 millisecondes la position GPS du véhicule. (periode3, capacité3)***
4. ***C4 : La tâche AFFICHAGE_1 qui produit toutes les 12 millisecondes, un résumé des informations produites par les tâches CAPTEUR_1, CAPTEUR_2 et CAPTEUR_3 sur un écran LCD. (periode4, capacité4)***
5. ***C5 : La tâche AFFICHAGE_2 qui affiche à la demande de l'utilisateur une carte routière. L'affichage doit être réactualisé toutes les 6 millisecondes. (periode6, capacité6)***

3 Donner les différentes paramètres des tâches citées dans B.2 avec leurs valeurs.

Tâches	*Période*	*Priorité*	*Capacité*
C1	***10***	***1***	***1***
C2	***10***	***2***	***2***
C3	***40***	***4***	***4***
A1	***12***	***3***	***2***
A2	***6***	***0***	***1***

4. Quel exécutif temps réel peut-on utiliser pour garantir le contrôle cette application ? Justifier

exprimer la contrainte de précédence entre C1, C2, C3 et A1 : dans ce cas, on affecte une priorité plus forte à C1, C2 et C3 afin de garantir que ces tâches soient exécutées avant A1. - privilégier l'urgence des tâches : dans ce cas on affecte les priorités selon Rate Monotonic (cad selon les périodes). Avec cette solution, on peut vérifier que les tâches respectent leurs contraintes temporelles avec le test sur le taux d'utilisation et le test sur le temps de réponse.
- Les priorités des tâches sont affectées selon Rate Monotonic.
- Les capacités des tâches étant des bornes sur leur temps d'exécution, la capacité de A2 est de 1 ms.
- C1 et C2 ayant la même période, on choisit, de façon arbitraire, d'affecter une priorité plus forte à C1 et moins forte à C2 ... le choix inverse est aussi possible.

5. Calculer le du pire temps de réponse du système considéré (algorithme RM)

RC1 = 1 + 1/6*1 = 2
= 1 + 2/6*1 = 2

RC2 = 2 + 2/6*1 + 2/10*1 = 4
= 2 + 4/6*1 + 4/10*1 = 4

RC3 = 4 + 4/6*1 + 4/10*2 + 4/10*1 + 4/12*2 = 10
= 4 + 10/6*1 + 10/10*2 + 10/10*1 + 10/12*2 = 11
= 4 + 11/6*1 + 11/10*2 + 11/10*1 + 11/12*2 = 14
= 4 + 14/6*1 + 14/10*2 + 14/10*1 + 14/12*2 = 17
= 4 + 17/6*1 + 17/10*2 + 17/10*1 + 17/12*2 = 17

RA1=6

RA2=1

C. En considère un ensemble de 6 tâches {A,B,C,D,E, F}.La tâche A doit précéder les tâches B ,C ,D. Les tâches B et C doivent précéder la tâche E. Les tâches D et E doivent précéder la tâche F. Donner les différents ordonnancements pour réaliser la synchronisation de ces tâches en utilisant les sémaphores.

Variable :sema S1=S2=S3=S4=S5=0

A' : Exe A ;V(S1) ; V(S1) ;V(S1) .
B' : P(S1) ;Exe B ;V(S2) .
C': P(S1) ;Exe C ; V(S3) .
D' : P(S1) ;Exe D ;V(S4).
E' :P(S2) ;P(S3) ;Exe E ;V(S5).
F' :P(S4) ;P(S5) ;Exe F.

Références Bibliographiques

A. Burns and A. Wellings	Real-time Systems and Programming Languages. Addison Wesley, 1997.
F. Cottet, J. Delacroix, C.Kaiser, and Z. Mammeri.	Ordonnancement temps réel. Hermès, 2000.
C. L. Liu and J. W. Layland	« Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environnment ». *Journal of the Association for Computing Machinery*, 20(1):46–61, January 1973.
M. Joseph and P. Pandya.	Finding Response Time in a Real-Time System. Computer Journal, 29(5):390--395, 1986.
J. Blazewicz	« Scheduling Dependant Tasks with Different Arrival Times to Meet Deadlines ». In. Gelende. H. Beilner (eds), Modeling and Performance Evaluation ofComputer Systems, Amsterdam, Noth-Holland, 1976.
H. Chetto, M. Silly, and T. Bouchentouf	« Dynamic Scheduling of Real-time Tasks Under Precedence Constraints ». Real Time Systems, The International Journal of Time-Critical Computing Systems, 2(3):181–194, September 1990.
Crocus	Systèmes d'explotation des ordinateurs, 1ére edition 1975, Maison d'édition DUNOD

https://docplayer.fr/170480595-Un-processeur-est-defini-par-les-champs-suivants-voir-figure-1-1.html (application contrôle aérien)

http://beru.univ-brest.fr/svn/CHEDDAR/trunk/releases.// frank singhoff

https ://www.lucidchart.com/pages/fr/langage-uml. le 11 mai 2022.
https ://docwiki.embarcadero.com/radstudio/sydney/fr/definition diagrammes de classes uml 1,5. le 11 mai 2022.
https ://docwiki.embarcadero.com/radstudio/sydney/fr/espaces de nommage et packages. le 11 mai 2022.