République Algérienne Démocratique et Populaire
Ministre de l'Enseignement Supérieur et de la Recherche Scientifique

Université Amar Telidji de Laghouat
Faculté de Technologie
Département de l'Électrotechnique

Polycopié de cours

Systèmes temps réels

Niveau : 2éme Année Master
Option : Automatique et Systèmes

Présenté par :

Dr. BENMOUIZA Khalil
Maître de Conférences -A-

2021/2022

Sommaire

Chapitre 4 : Langage de programmation temps réel (4 semaines)

Résumé :

Le cours des systèmes temps réel est destiné aux étudiants de **Master en Automatique**, option **Automatique et Systèmes**. L'objectif de ce cours est de présenter les notions permettant d'analyser les exigences d'un problème temps-réel, conception de la solution, démonstrations et la correction de la conception proposée, programmation de la solution , validation de la solution et de concevoir la conception des applications sur un système temps réel. Le contenu du cours peut se résumer dans les points suivants :

- Chapitre 1 : Introduction , architecture et fonctionnement des systèmes en temps réel ;
- Chapitre 2 : Techniques de spécifications d'un système TR ;
- Chapitre 3 : Programmation concurrente ;
- Chapitre 4 : Langage de programmation en TR.

Connaissances préalables recommandées :

L'étudiant devra posséder les connaissances suivantes :

- Connaissance des bases du fonctionnement des microprocesseurs.
- Connaissance de la programmation en plusieurs langage informatique (C, C++, Java , Python …etc.).

Introduction générale

Le rythme de la technologie évolue rapidement chaque année, les nouvelles technologies évoluent plus rapidement que jamais. Le coût et le risque liés à une production de meilleure qualité augmentent de jour en jour. L'émergence des systèmes embarqués a réduit non seulement les coûts et les risques, mais a également amélioré la qualité.

Les systèmes embarqués peuvent être appelés systèmes informatiques intégrés dans des systèmes électriques ou mécaniques. Les systèmes embarqués en temps réel sont utiles pour surveiller et contrôler les environnements externes[1-2].

L'environnement externe est relié aux systèmes informatiques à l'aide d'interfaces d'entrée/sortie, d'actionneurs et de capteurs. L'environnement externe peut être constitué d'objets biologiques et physiques comme des animaux ainsi que des humains. Alors que pour comprendre le comportement du monde extérieur, les systèmes informatiques sont configurés pour suivre certains paramètres. C'est ce qu'on appelle comprendre le comportement en temps réel de l'environnement à travers un système embarqué [1-2] .

L'utilisation du système embarqué en temps réel s'est pratiquement améliorée ces dernières années. Cela va des grandes industries comme l'armée, l'aéronautique, la robotique aux industries commerciales comme les télécommunications, l'automobile et les appareils de cuisine.

C'est un fait connu que les systèmes embarqués sont alliés à la technologie logicielle parce que la technologie logicielle détermine le succès et les performances des systèmes embarqués.

Mais les systèmes embarqués ne sont pas exactement une application logicielle, ils possèdent un ensemble de caractéristiques inhérentes qui les distinguent des applications logicielles courantes. Les systèmes embarqués en temps réel sont destinés à répondre à des situations du monde réel, qui suivent strictement les paramètres définis par l'environnement avec lequel ils interagissent. La plus haute importance de la principale caractéristique des systèmes embarqués est qu'ils doivent répondre à un ensemble d'incitations externes dans des contraintes de temps inflexibles et précaires [1-2].

Une application en temps réel comprendra normalement à la fois des composants matériels et logiciels. Un réacteur nucléaire est un exemple très parlant du système en temps réel dur. Il ne détecte pas seulement la panne mais aussi assez rapidement dans les situations pour éviter la fusion. Les composants souples sont un peu moins exigeants en temps et en précision. Sous une forme ou une autre, chaque voiture qui sort de n'importe quel centre de production utilise une technologie embarquée [1-2, 5].

Les systèmes embarqués incluent les systèmes d'exploitation en temps réel. Le système d'exploitation en temps réel cache toutes les fonctionnalités du backend, il présente simplement un écran à l'utilisateur et sa cache tous les calculs.

Le système d'exploitation en temps réel (RTOS) aide l'utilisateur à utiliser un maximum de temps et de ressources pour obtenir la sortie souhaitée. Le système d'exploitation en temps réel prend en charge les processus d'exécution, de surveillance et de contrôle. RTOS est largement utilisé dans les secteurs tels que les unités de fabrication d'avions, de voitures, de robots et de machines. Les RTOS donnent plus de sorties en gardant tous les appareils et ressources actifs. Le RTOS est utilisé dans les jeux vidéo, les appareils numériques et l'industrie robotique [6].

Semestre: 3

Unité d'enseignement: UEM 2.1

Matière: Systèmes temps réel

VHS: 37h30 (Cours: 1h30, TP: 1h00)

Crédits: 3

Coefficient: 2

Chapitre I : Introduction , architecture et fonctionnement des systèmes en temps réel

I.1 Définitions

I.1.1 Système embarqués

Un système embarqué (SE) est un système informatisé spécialisé qui constitue une partie intégrante d'un système plus large ou une machine. Typiquement, c'est un système sur un seul processeur et dont les programmes sont stockés en ROM.

Un système embarqué est une combinaison de logiciel et matériel, avec des capacités fixes ou programmables, qui est spécialement conçu pour un type d'application particulier. Les distributeurs automatiques de boissons, les automobiles, les équipements médicaux, les caméras, les avions, les jouets, les téléphones portables et les PDA sont des exemples de systèmes qui abritent des SE. Les SE programmables sont dotés d'interfaces de programmation et leur programmation est une activité spécialisée [1-2].

I.1.2 Système à temps réel

Un système temps réel est un système informatique devant répondre à des stimuli de l'environnement en tenant compte de **contraintes de temps**. Le comportement correct d'un tel système ne dépend donc pas uniquement des résultats fournis, mais également de **la date** à laquelle ces résultats sont communiqués. Il ne faut pas confondre temps réel et rapidité ; un temps réel veut dire prédictibilité et non rapidité [5].

Table I.1 Ordre des grandeurs de temps.

Temps	**Exemple**
nanosecond - O(ns)	Temps d'accès à une RAM (5-80ns) Durée entre deux ticks d'horloge d'un processeur Pentium (1GHz) Fréquence de 1GHz (10^9Hz)
microseconde - O(μs)	Traitement dans un noyau de système d'exploitation (changement de contexte, interruption matérielle) Systèmes utilisant des radars (navigation, détection de mouvement, etc.) Transmission sur des bus de terrain, transmission radio Fréquence de 1 MHz (10^6 Hz)
milliseconde - O(ms)	Temps d'accès à un disque dur SCSI ou IDE (5-20 ms) La durée d'échantillonnage du son, protocoles de télécommunication Fréquence de 1 KHz (10^3 Hz)
seconde (s) - O(s)	Systèmes de visualisation humain (temps durant lequel l'oeil peut "intégrer" 25 images au plus) Applications multimédia Temps de réponse des applications informatiques (accès DB, compilation, etc.) Fréquence de 1 Hz
L'heure (h) - O(h)	Applications de surveillance de réactions chimiques, surveillance de données météorologiques
Le mois, l'année - O(m, a)	Systèmes de navigation de sonde spatiale

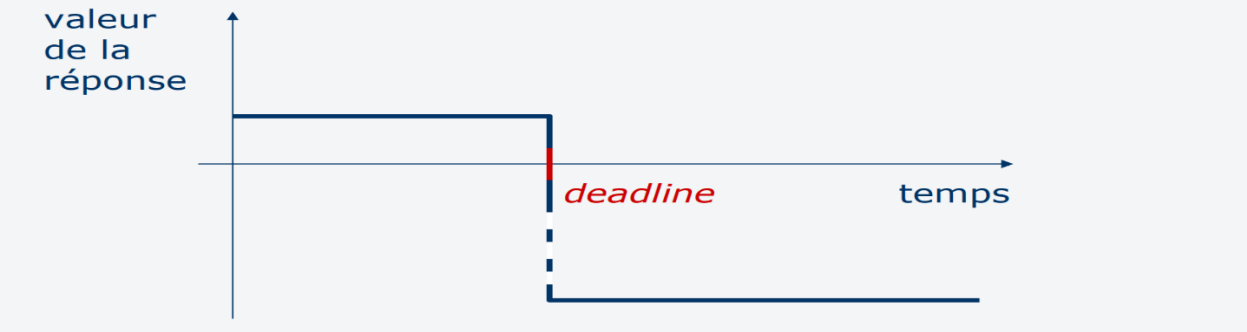
I.2 Classification des systèmes temps réel

Les systèmes temps réel peuvent être classifis en trois catégories [1-2, 5-6] :

- Temps réel **dur** (hard real time) ;
- Temps réel **ferme** (firm real time);
- Temps réel **mou** (soft real time).

A) Temps réel dur

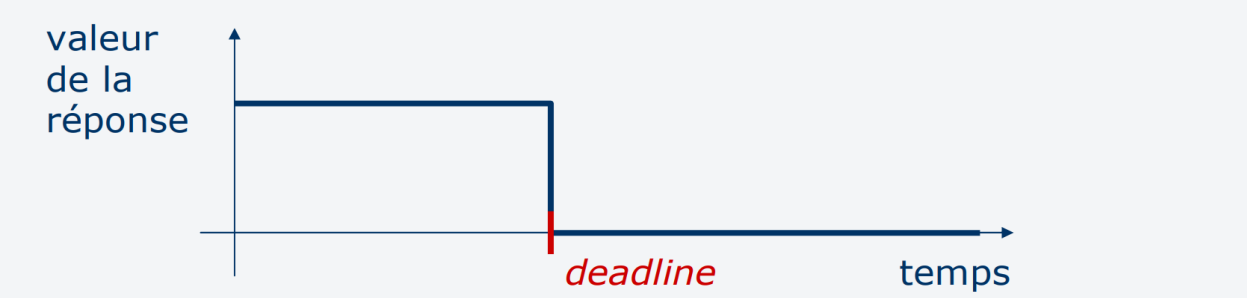
La réponse du système dans les délais est vitale. L'absence de réponse est catastrophique (le non-respect des contraintes temporelles entraîne la faute du système).



Exemples : contrôle aérien, contrôle d'une centrale nucléaire…

B) Temps réel ferme

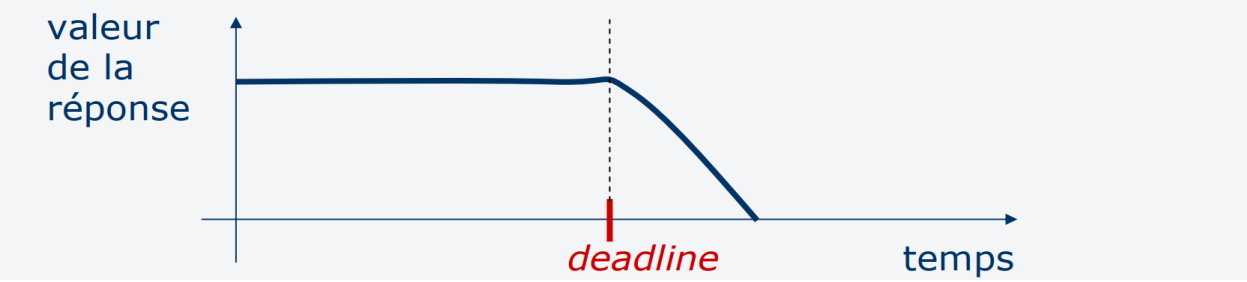
La réponse du système dans les délais est essentielle. Le résultat ne sert plus à rien une fois le deadline passé (définition alternative : la pénalité de non-réponse est dans le même ordre de magnitude que la valeur de la réponse).



Exemples : transactions en bourse… (c'est subjectif : le temps réel ferme de l'un peut être le temps réel dur de l'autre)

C) Temps réel mou

La réponse du système après les délais réduit progressivement sont intérêt. Les pénalités ne sont pas catastrophiques.



Exemples : VoD, logiciel embarqué de votre téléphone, iPod, etc.

I.3 Quelques exemples d'applications

A) Système de transport

Les véhicules de tourisme, utilitaire, poids lourd, etc., ferroviaire, aérien, ou spatial, les systèmes de transport sont caractérisés par une forte criticité et une forte complexité due à l'expansion du nombre de fonctions et la nécessité de tolérance aux fautes. Ils sont typiquement décomposés en sous-systèmes interconnectés par un ensemble de bus de terrains.

B) Drone volant, roulant, navigant, ou sous-marin

Ce type d'applications est de plus en plus répandu, et de plus en plus de l'autonomie est donnée aux drones. Qu'ils soient d'observation ou tactiques, ce type d'application trouve une large place aussi bien dans les applications militaires que civiles (exploration de zone radioactive après un accident, planétaire, sous-marine, cinéma et télévision, etc.).

C) Robot de production

Un robot, réalisant une activité spécifique (peinture, assemblage, tri) sur une chaîne de production, doit effectuer son travail en des temps fixés par la cadence de fabrication. S'il agit trop tôt ou trop tard, l'objet manufacturier traité sera détruit ou endommagé conduisant à des conséquences financières ou humaines graves (oubli d'un ou plusieurs rivets sur un avion).

D) Téléphone mobile

Le système de contrôle-commande doit remplir plusieurs fonctions dont certaines ont des contraintes temporelles fortes pour avoir une bonne qualité de service (QoS : qualité of service). Ainsi, la première fonction est de transmettre et de recevoir les signaux de la parole et l'envoie des messages.

E) Système de vidéoconférence

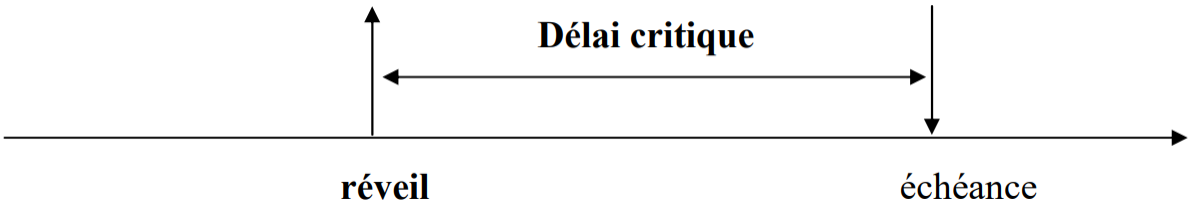
Ce système doit permettre l'émission et la réception d'images numérisées à une cadence de 20 à 25 images par seconde pour avoir une bonne qualité de service. Afin de minimiser le débit du réseau, une compression des images est effectuée. D'autre part la parole doit aussi être transmise.

I.4 Architecture d'un système en temps réel

I.4.1 Tâches

Une tâche est généralement caractérisée par un temps de calcul (C_i), une échéance (D_i) qui est la date à laquelle la tâche doit avoir terminé son exécution, et dans le cas des tâches périodiques, par une période (T_i) qui représente la durée séparant ses instants d'activation. Une exécution de la tâche est appelée une instance.

Chaque tâche possède un délai critique : temps maximal pour s'exécuter depuis sa date de réveil. La date butoir résultante est appelée échéance. Le dépassement d'une échéance est appelé faute temporelle.



A) Tâches périodiques

Elles correspondent aux mesures sur le procédé ; elles se réveillent régulièrement (toutes les P unités de temps) , on peut trouver :

- Périodiques strictes : contraintes temporelles à respecter absolument.
- Périodiques relatives : contraintes temporelles qui peuvent être non respectées de temps à autre.

B) Tâches apériodiques

Elles correspondent aux événements ; elles se réveillent de manière aléatoire

- Apériodiques strictes : contraintes temporelles dures à respecter absolument.
- Apériodiques relatives : contraintes temporelles molles qui peuvent être non respectées de temps à autre (sans échéance).

I.4.2 Thread

Permet de faire fonctionner plusieurs portions de code simultanément dans le même processus => partagent le même espace mémoire. Parmi les exemples de thread [3] :

– Correcteur grammatical d'un traitement de texte ;

– Accélération de calculs matriciels sur un multi-processeur/cœur ;

– Permet de traiter plusieurs connexions client/serveur simultanément.

I.4.3 Architecture des systèmes en temps réel

La structure d'un système en temps réel est donnée par la figure suivante :

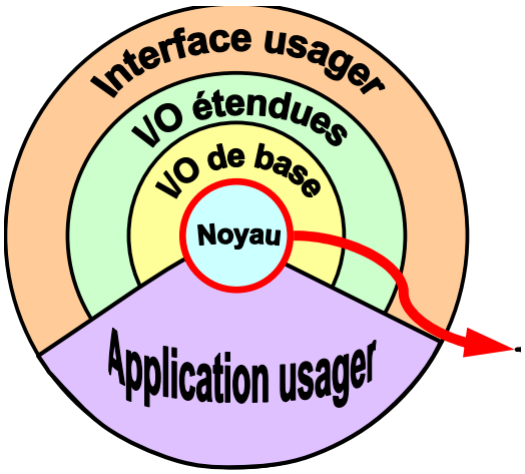


Fig. II.1 : Architecture d'un système en temps réel

- Interface usager :

C'est l'ensemble des mécanismes, matériels ou logiciels, qui permettent à un utilisateur d'interagir avec un système informatique. En d'autres termes, l'interface utilisateur, c'est ce qui fait le lien entre l'Homme et la machine.

- I/O étendues, I/O de base :

Les périphériques sont reliés au reste du système par des circuits appelés ports d'entrées et ports de sortie.

- Application usager :

Sont les applications permettant à l'usager d'interagir avec le milieu ou le logiciel .

- Noyau :

Un noyau de système d'exploitation, ou simplement noyau, ou *kernel* (en l'anglais), est une des parties fondamentales de certains systèmes d'exploitation. Il gère les ressources de l'ordinateur et permet aux différents composants — matériels et logiciels — de communiquer entre eux.

Le noyau fournit 3 fonctionnalités :

- Ordonnanceur (scheduler) : Choisit l'ordre d'exécution des tâches.
- Répartiteur (dispatcher) : Effectue le changement d'une tâche à l'autre.
- Communication inter-tâches : Gère les échanges d'information.

L'objectif du noyau est :

- Donne à l'exécutif l'accès à une machine virtuelle .
- Réside en mémoire centrale .
- Assure : la gestion des tâches , l'allocation du processeur , la gestion de la mémoire , la gestion des interruptions et des trappes , la synchronisation et l'exclusion mutuelle , la communication entre tâches , la communication avec l'extérieur.

Hiérarchie de noyau

Système d'exploitation	**Exécutif + interface usager généralisée, sécurité et gestion du système de fichier.**
Exécutif	**Noyau + privatisation des blocs de mémoire, services d'I/O et autres fonctions complexes.**
Noyau	**Micro-noyau + communication inter-tâches et synchronisation (sémaphores, boîtes à lettres, mutex, …).**
Micro-noyau	**Nano-noyau + ordonnanceur.**
Nano-noyau	**Gestion simple des tâches (threads). Fournit seulement la fonction de répartition.**

I.4.5 Pseudo parallélisme et quasi parallélisme

Partage d'un processeur par plusieurs processus

- Pseudo-parallélisme : à tout moment un processus est en exécution – la commutation de processus se fait à l'insu des processus – c'est le schéma utilisé pour un langage temps réel.
- Quasi-parallélisme : – à tout instant un processus est en exécution – la commutation de processus se fait à la demande du processus actif par une instruction particulière – utilisé dans le contexte de la simulation.

I.4.6 Exceptions et interruptions

a) Exceptions

Dans l'exception, n'importe quel événement qui interrompt l'exécution normale du processeur et l'oblige à exécuter un ensemble d'instructions spéciales dans un état privilégié. Les exceptions peuvent être synchrones ou asynchrones. Les exceptions générées par des événements internes sont des exceptions synchrones, par exemple la division par zéro, non alignement des données. Les exceptions générées par des événements externes sont des exceptions asynchrones, par exemple la pression sur le bouton de reset, arrivée de données sur le module de communication du processeur.

b - Interruptions

Une Interruption est une classe particulière d'exceptions externes (asynchrones) provoquées par un événement déclenché par du matériel extérieur au processeur. Les interruptions se distinguent des exceptions synchrones par la source (le processeur lui-même pour les exceptions synchrones, un matériel externe pour les interruptions). Les exceptions et les interruptions sont utilisées pour communiquer entre le processeur et le matériel extérieur pour les erreurs internes, la gestion de conditions spéciales, l'utilisation concurrente du matériel et la gestion des demandes de service.

Dans le cas d'un programme temps réel, on s'intéresse particulièrement aux évènements asynchrones matérialisés par des signaux électriques. Une procédure d'E/S peut être invoquée en temps réel par un évènement asynchrone matériel. Cet évènement est lié à un signal logique au niveau d'une broche spécifique du CPU, appelée entrée d'interruption.

I.4.6.1 Rôle des interruptions

Une entrée interruption permet d'interrompre le microprocesseur pour exécuter une tâche (appelée routine d'interruption), considérée comme prioritaire. On peut distinguer deux types d'interruption, masquable et non masquable

a) Interruption non masquable

Elle prise en compte à son arrivée (après avoir terminé l'instruction en cours s'exécution). Ce type d'interruption est utilisé plutôt dans les fonctionnalités du système : Reset, horloge temps réel, défaillance d'alimentation… Elle est appelée généralement NMI (No Masquable Interrupt).

b) Interruption masquable

Ces interruptions sont liées à des fonctionnalités de l'application. Le microprocesseur a la possibilité d'honorer ou d'ignorer la demande d'une interruption masquable. En général chaque microprocesseur possède un bit interne, dont sa valeur booléenne détermine l'inhibition (masquage) ou la validation de l'interruption.

I.4.6.2 Gestion des interruptions

Lors de la gestion de l'interruption, le microprocesseur doit effectuer les procédures suivantes :

- Résolution de priorité : le microprocesseur doit déterminer l'interruption la plus prioritaire au cas où plus qu'une interruption se présente.
- Sauvegarde de l'adresse de retour : A cet effet, le microprocesseur doit sauvegarder l'adresse de retour dans une pile gérer par un pointeur de pile avant de se brancher à la routine d'interruption.
- Localisation en mémoire de la routine d'interruption : à chaque interruption est associée une adresse permettant de localiser le point d'entrée de la routine d'interruption. Le mécanisme de création de ces adresses s'appelle vectorisation. La vectorisation est assurée soit par le CPU, soit par l'interface source d'interruption.

Chapitre II : Techniques de spécifications d'un système TR

II.1 Introduction

Dans un système multi-utilisateur à temps partagé, plusieurs processus peuvent être présents en mémoire centrale en attente d'exécution. La limitation des ressources (en particulier du processeur) conduit à bloquer des processus (ils ne peuvent progresser du fait de manque de ressource). Afin de respecter en permanence les échéances, il faut gérer efficacement la pénurie et tenter de favoriser les processus dont l'avancement est le plus urgent . Si plusieurs processus sont prêts, le système d'exploitation doit gérer l'allocation du processeur aux différents processus à exécuter. C'est l'ordonnanceur qui s'acquitte de cette tâche.

II.2 Ordonnancement

Un ordonnancement consiste à définir un ordre sur l'utilisation des ressources du système afin de respecter les échéances temporelles**.** Un ordonnanceur fait face à deux problèmes principaux :

- le choix du processus à exécuter,
- le temps d'allocation du processeur au processus choisi.

II.2.1 Types d'ordonnanceurs

Il est possible de distinguer trois types d'ordonnanceurs :

A) Long terme :

L'ordonnanceur fait la sélection de programmes à admettre dans le système pour leur exécution. Les programmes admis deviennent des processus à l'état prêt. L'admission dépend de la capacité du système (degré de multiprogrammation) et du niveau de performance requis.

B) Moyen terme :

Il fait la sélection de processus déjà admis à débarquer ou rembarquer sur la mémoire. Il effectue ses tâches de gestion en fonction du degré de multiprogrammation du système, et aussi des requêtes d'E/S des périphériques.

C) Court terme :

L'ordonnanceur à court terme a comme tâche la gestion de la file des processus prêts. Il sélectionne , en fonction d'une certaine politique , le prochain processus à exécuter. Il effectue

aussi le changement de contexte des processus. Il peut implanter un ordonnancement préemptif, non préemptif, ou coopératif. L'ordonnanceur est activé par un événement : interruption du temporisateur, interruption d'un périphérique, appel système ou signal.

II.2.2 Catégories d'ordonnancement

a) Système multitâches préemptif

Un système d'exploitation multitâche est préemptif lorsque celui-ci peut arrêter (réquisition) à tout moment n'importe quelle application pour passer la main à la suivante. Dans les systèmes d'exploitation préemptifs on peut lancer plusieurs applications à la fois et passer de l'une à l'autre, voire lancer une application pendant qu'une autre effectue un travail [4].

b) Système multitâches coopératif

Un multitâche coopératif permet à plusieurs applications de fonctionner et d'occuper des plages mémoire, laissant le soin à ces applications de gérer cette occupation, au risque de bloquer tout le système. Par contre, avec un « multi-tâche préemptif », le noyau garde toujours le contrôle (qui fait quoi, quand et comment), et se réserve le droit de fermer les applications qui monopolisent les ressources du système. Ainsi les blocages du système sont inexistants [4].

II.2.3 Objectifs de l'ordonnanceur

- S'assurer que chaque processus en attente d'exécution reçoive sa part de temps processeur.
- Minimiser le temps de réponse.
- Utiliser le processeur à 100%.
- Utilisation équilibrée des ressources.
- Prendre en compte des priorités.
- Être prédictibles.

II.2.4 Les critères d'ordonnancement

La politique d'ordonnancement doit optimiser les performances du système à travers plusieurs critères :

- Rendement d'utilisation du CPU : pourcentage de temps pendant lequel le CPU est actif. Plus ce paramètre est élevé mieux c'est. Ceci permettra d'augmenter le nombre de jobs achevés par unité de temps.

- Utilisation globale des ressources: c'est assurer une occupation maximale des ressources de la machine et minimiser le temps d'attente pour l'allocation d'une ressource à un processeur.
- Débit: nombre de processus qui complètent dans l 'unité de temps
- Equité : capacité de l'ordonnanceur à allouer le CPU d'une façon équitable à tous les processus de même priorité (éviter la famine).
- Temps de rotation (temps de séjour ou de virement) : est l'intervalle de temps entre la soumission du processus et son achèvement (durée moyenne nécessaire pour qu'un processus termine son exécution) .
- Temps d'attente : durée moyenne qu'un processus passe à attendre le CPU.
- Temps de réponse : temps moyen qui s'écoule entre le moment où un utilisateur soumet une requête et celui où il commence à recevoir les réponses.

À maximiser	**À minimiser**
Utilisation globale des ressources	Temps de rotation
Rendement d'utilisation du CPU	Temps d'attente
Débit	Temps de réponse

$$Temps\ de\ rotation = Temps\ fin\ d'exécution - Temps\ d'arrivée$$

$$Temps\ d'attente = Temps\ de\ rotation - Durée\ d'exécution$$

$$Temps\ moyen\ d'attente = \frac{\sum Temps\ attente}{nbre\ de\ processus}$$

$$Rendement = \frac{\sum Temps\ d'exécution}{nbre\ de\ processus}$$

II.3 Les algorithmes d'ordonnancement

Un algorithme d'ordonnancement est une méthode ou stratégie utilisée pour ordonnancer les processus . Un tel algorithme s'appuie sur la connaissance de certaines caractéristiques des processus ou du système.

Cas 1 : Ordonnanceurs non préemptifs

Dans un système à ordonnancement non préemptif ou sans réquisition, le système d'exploitation choisit le prochain processus à exécuter, en général, on a [4] :

- **Premier Arrivé est le Premier Servi PAPS** (ou First-Come First-Served FCFS) :

L'ordonnanceur lui alloue le processeur jusqu'à ce qu'il se termine ou qu'il se bloque (en attente d'un événement). Il n'y a pas de réquisition.

- **Le plus court d'abord** (Short Job First SJF):

L'ordonnanceur choisit, parmi le lot de processus à exécuter, le plus court (plus petit temps d'exécution). Cette stratégie est bien adaptée au traitement par lots de processus dont les temps maximaux d'exécution sont connus ou fixés par les utilisateurs car elle offre un meilleur temps moyen de séjour.

Exemples :

Considérons cinq travaux A, B, C, D et E, dont les temps d'exécution et leurs arrivages respectifs sont donnés dans la table III.1.

Table III.1 Données d'ordonnancement.

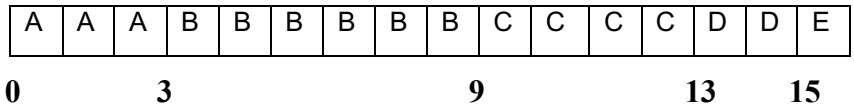
Processus	Temps d'exécution	Temps d'arrivage
A	3	0
B	6	1
C	4	4
D	2	6
E	1	7

1. Faire un schéma qui illustre son exécution (diagramme de Gantt).
2. Calculer le temps de séjour de chaque processus, le temps moyen de séjour, le temps d'attente et le temps moyen d'attente en utilisant : Premier arrivé premier servi (PAPS) et Le plus court d'abord (SJF).

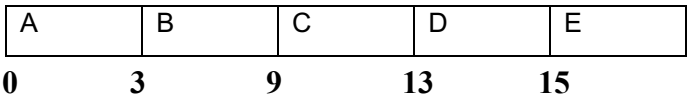
Solution :

1- **Utilisant PAPS**

- **Schéma d'exécution** :



Ou bien



Au temps 0, seulement le processus A est dans le système et il s'exécute. Au temps 1 le processus B arrive mais il doit attendre qu'A termine car il a encore 2 unités de temps. Ensuite B s'exécute pendant 4 unités de temps. Au temps 4, 6, et 7 les processus C, D et E arrivent mais B a encore 2 unités de temps. Une fois que B a terminé, C, D et E entrent au système dans l'ordre.

- **Le temps de rotation (séjour)** pour chaque processus est obtenu soustrayant le temps d'entrée du processus du temps de terminaison. Ainsi :

Processus	Temps de sejour
A	3-0 = 3
B	9-1 = 8
C	13-4 = 9
D	15-6 = 9
E	16-7 = 9

- **Le temps moyen** de séjour est : $\frac{(3+8+9+9+9)}{5} = \frac{38}{5} = 7,6$

- **Le temps d'attente** est calculé soustrayant le temps d'exécution du temps de séjour :

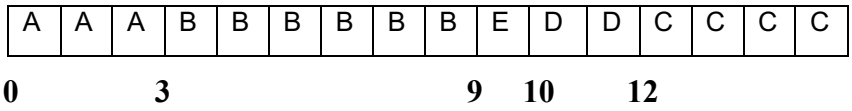
Processus	Temps d'attente
A	3-3 = 0
B	8-6 = 2
C	9-4 = 5
D	9-2 = 7
E	9-1 = 8

- **Le temps moyen d'attente** est : $\frac{(0+2+5+7+8)}{5} = \frac{23}{5} = 4,4$

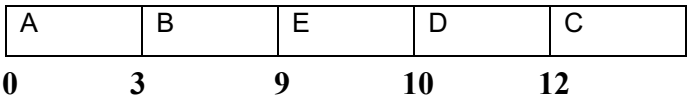
- **Débit** : il y a cinq tâches exécutées dans 16 unités de temps, alors 16/5 = 3.2 **unités de temps par processus**.

2- **Utilisant SJF**

- **Schéma d'exécution** :



Ou bien



Au temps 0, seulement le processus A est dans le système et il s'exécute. Au temps 1 le processus B arrive mais il doit attendre qu'A termine car il a encore 2 unités de temps. Au temps 9 (3+6) , les processus C, D et E arrivent ; on choisit le processus qui a le plus petit temps d'exécution E , D et C.

- **Le temps de rotation (séjour)** pour chaque processus est obtenu soustrayant le temps d'entrée du processus du temps de terminaison. Ainsi :

Processus	Temps de séjour
A	3-0 = 3
B	9-1 = 8
E	10-7 = 3
D	12-6 = 6
C	16-4 = 12

- **Le temps moyen** de séjour est : $\frac{(3+8+3+6+12)}{5} = \frac{32}{5} = 6,4$
- **Le temps d'attente** est calculé soustrayant le temps d'exécution du temps de séjour :

Processus	Temps d'attente
A	3-3 = 0
B	8-6 = 2
E	3-1 = 2
D	6-2 = 4
C	12-4 = 8

- **Le temps moyen d'attente** est : $\frac{(0+2+2+4+8)}{5} = \frac{16}{5} = 3,2$
- **Débit** : il y a cinq tâches exécutées dans 16 unités de temps, alors 16/5 = 3.2 **unités de temps par processus**.

➢ Conclusion

Ordonnancement First Come First Served (FCFS)

- Intérêts : Algorithme facile à comprendre Faible complexité d'implémentation (une seule liste chaînée).
- Inconvénients : Pas de prise en compte de l'importance relative des processus Temps d'attente du processeur généralement important.

Short Job First SJF

- Intérêts : SJF réduit le temps d'attente des processus Utilisation limitée à des environnements et à des applications spécifiques.
- Inconvénients : Pas de prise en compte de l'importance relative des processus Algorithme optimal uniquement dans le cas où tous les processus sont disponibles simultanément.

Les ordonnanceurs non préemptifs ne sont pas intéressants pour les systèmes multi-utilisateurs car les temps de réponse ne sont pas toujours acceptables.

Cas 2 : Ordonnanceurs préemptifs

Dans le cadre préemptif, une tâche peut être momentanément interrompue par l'ordonnanceur afin de laisser une autre tâche s'exécuter. De ce fait, une tâche très longue n'a plus de risque de bloquer des tâches plus courtes trop longtemps. En général, on a [4] :

a) Temps restant le plus court (Shortest Remaining Time , SRT) :

C'est la version préemptive de l'algorithme SJF. Un processus arrive dans la file de processus, l'ordonnanceur compare la valeur espérée pour ce processus contre la valeur du processus actuellement en exécution. Si le temps du nouveau processus est plus petit, il rentre en exécution immédiatement.

b) L'algorithme du tourniquet, circulaire (Round Robin , RR) :

Est un algorithme ancien, simple, fiable et très utilisé. Il mémorise dans une file du type FIFO (First In First Out) la liste des processus prêts, c'est-à-dire en attente d'exécution. Le processeur passe donc d'un processus à un autre en exécutant chaque processus pendant quelques dizaines ou centaines de millisecondes. Le temps d'allocation du processeur au processus est appelé **quantum**. Cette commutation entre processus doit être rapide, c'est-à-dire ,exiger un temps nettement inférieur au quantum.

Choix du processus à exécuter :

Il alloue le processeur au processus en tête de file ,pendant un quantum de temps. Si le processus se bloque ou se termine avant la fin de son quantum, le processeur est immédiatement alloué à un autre processus (celui en tête de file). Si le processus ne se termine pas au bout de son quantum, son exécution est suspendue. Le processeur est alloué à un autre processus (celui en tête de file). Le processus suspendu est inséré en queue de file. Les processus qui arrivent ou qui passent de l'état bloqué à l'état prêt sont insérés en queue de file.

Choix de la valeur du quantum

Un quantum trop petit provoque trop de commutations de processus et abaisse l'efficacité du processeur. Un quantum trop élevé augmente le temps de réponse des courtes commandes en mode interactif. Un quantum entre 20 et 50 ms est souvent un compromis raisonnable.

Exemples :

Soient deux processus A et B prêts tels que A est arrivé en premier suivi de B, 2 unités de temps après. Les temps de UCT nécessaires pour l'exécution des processus A et B sont respectivement 15 et 4 unités de temps .Le temps de commutation est supposé nul. Calculer le temps de séjour de chacun des processus A et B, le temps moyen de séjour, le temps d'attente, le temps moyen d'attente, et le nombre de changements de contexte pour:

- SRT.
- Round robin (quantum =10 unités de temps).
- Round robin (quantum =3 unités de temps.

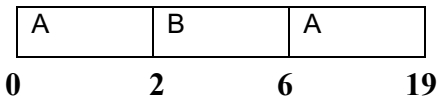
Solution :

Table des données d'ordonnancement.

Processus	Temps d'exécution	Temps d'arrivage
A	15	0
B	4	2

- SRT :

Diagramme de Gantt :



Au temps 0, seulement le processus A est dans le système et il s'exécute. Au temps 2 le processus B arrive. Selon la méthode de SRT , le processus a moins temps d'exécution restant s'exécute . Pour A il reste 13 unités de temps par contre seulement 4 unités de temps pour B. Alors , le processus B s'exécute. Lorsque qu'il finit, le processeur exécute A.

- **Le temps de rotation (séjour)**:

Processus	Temps de séjour
A	19-0 = 19
B	6-2 = 4

- **Le temps moyen** de séjour est : $\frac{19+4}{2} = 11,5$
- **Le temps d'attente** est calculé soustrayant le temps d'exécution du temps de séjour :

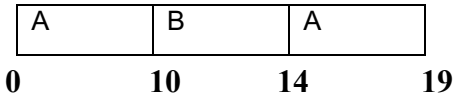
Processus	Temps d'attente
A	19-15 = 4
B	4-4 = 0

- **Le temps moyen d'attente** est : $\frac{4+0}{2} = 2$

Il y'a 3 changements de contexte.

- Round robin (quantum= 10) :

Diagramme de Gantt :



Au temps 0, seulement le processus A est dans le système et il s'exécute. Au temps 2 le processus mais il doit attendre qu'A termine son quantum car il a encore 8 quantum. Après 10 quantum, le processus B s'exécute. Lorsque qu'il finit, le processeur exécute A.

- **Le temps de rotation (séjour)**:

Processus	Temps de séjour
A	19-0 = 19
B	14-2 = 12

- **Le temps moyen** de séjour est : $\frac{19+12}{2} = 15,5$

- **Le temps d'attente** est calculé soustrayant le temps d'exécution du temps de séjour :

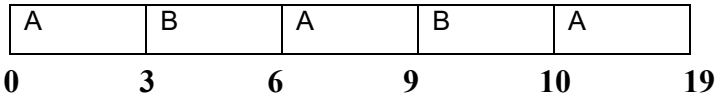
Processus	Temps d'attente
A	19-15 = 4
B	12-4 = 8

- **Le temps moyen d'attente** est : $\frac{4+8}{2} = 6$

Il y'a 3 changements de contexte.

- Round robin (quantum= 3) :

Diagramme de Gantt :



- **Le temps de rotation (séjour)**:

Processus	Temps de séjour
A	19-0 = 19
B	10-2 = 8

- **Le temps moyen** de séjour est : $\frac{19+8}{2} = 13,5$

- **Le temps d'attente** est calculé soustrayant le temps d'exécution du temps de séjour :

Processus	Temps d'attente
A	19-15 = 4
B	8-4 = 4

- **Le temps moyen d'attente** est : $\frac{4+4}{2} = 4$

Il y'a 5 changements de contexte.

c) <u>Ordonnancement avec priorité :</u>

L'algorithme round robin permet une répartition équitable du processeur. Cependant il n'est pas intéressant si certains processus sont plus importants ou urgents que d'autres. L'ordonnanceur à priorité attribue à chaque processus une priorité. Le choix du processus à élire dépend des priorités des processus prêts. Les processus de même priorité sont regroupés dans une file du type FIFO.Il y a autant de files qu'il y a de niveaux de priorité. L'ordonnanceur choisit le processus le plus prioritaire qui se trouve en tête de file. En général, les processus de même priorité sont ordonnancés selon l'algorithme du tourniquet.

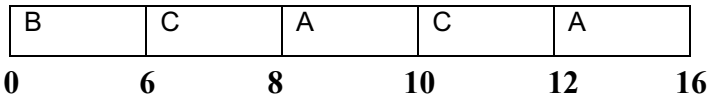
Example :

Soient trois processus A , B et C prêts tels l' arrivé des trois processus est en méme temps avec un quantum de temps égal à 2.

Processus	Temps d'exécution	Priorité	Temps d'arrivage
A	6	1	0
B	6	2	0
C	4	1	0

Nous pouvons noter que la tâche B, la plus prioritaire, est exécutée au départ, puis un tourniquet est effectué pour les deux autres tâches. Cette technique de tourniquet à priorité correspond à la politique standard observée dans les systèmes d'exploitation, car elle permet à toutes les tâches de s'exécuter sans risque de famine.

Diagramme de Gantt :



➢ Conclusion

Temps restant le plus court (Shortest Remaining Time , SRT) :

- Intérêts : SRT minimise le temps d'attente moyen des processus les plus courts Utilisation limitée à des environnements et à des applications spécifiques
- Inconvénients :Pas de prise en compte de l'importance relative des processus Non équité de service : SRT pénalise les processus longs Possibilité de famine pour les processus longs.

L'algorithme du tourniquet, circulaire (Round Robin , RR) :

- Intérêts : équité de l'attribution du processeur entre toutes les tâches Mise en œuvre simple
- Inconvénients : Pas de prise en compte de l'importance relative des tâches Difficulté du choix de la tranche de temps Si **q** est trop grand, Round-Robin devient équivalent à FIFO Si **q** est trop petit, il y'a augmentation du nombre de changements de contexte.

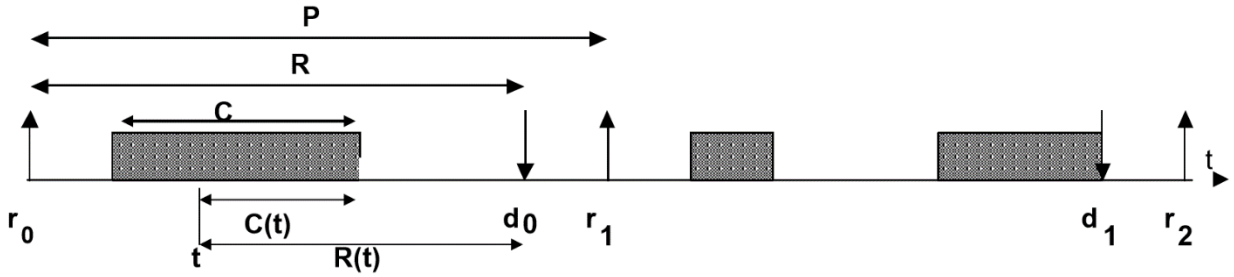
Ordonnancement avec priorité :

- Intérêts : Prise en compte de l'importance relative des processus Mise en œuvre simple.
- Inconvénients : Problèmes de blocages durant des périodes de temps illimitées Problèmes de famines des processus de moindres priorités.

II.4 Algorithmes d'ordonnancement pour les tâches périodiques

Dans de nombreux systèmes temps réel, certaines tâches doivent s'exécuter de manière répétée, à intervalles réguliers. Il pourra par exemple s'agir d'une tâche chargée d'aller vérifier la valeur d'un capteur tous les dixièmes de seconde. Ces tâches sont appelées tâches périodiques, puisqu'elles doivent s'exécuter régulièrement, en suivant une période fixe [4].

Pour la mise au point des algorithmes d'ordonnancement dans le cadre des tâches périodiques, nous utiliserons la notation suivante :



r_0: date de réveil de la tâche

C: durée d'exécution maximale

R : délai critique

P : période d'exécution

r_k: date de réveil de la k^{ième} instance de la tâche $r^k = r_0 + kP$

d_k: échéance $d_k = r_k + R$

C(t): temps d'exécution restant à t

R(t): délai critique dynamique (temps restant à t jusqu'à d) Quand R=P : tâche à échéance sur requête.

II.4.1 Algorithme Rate Monotonic(RM)

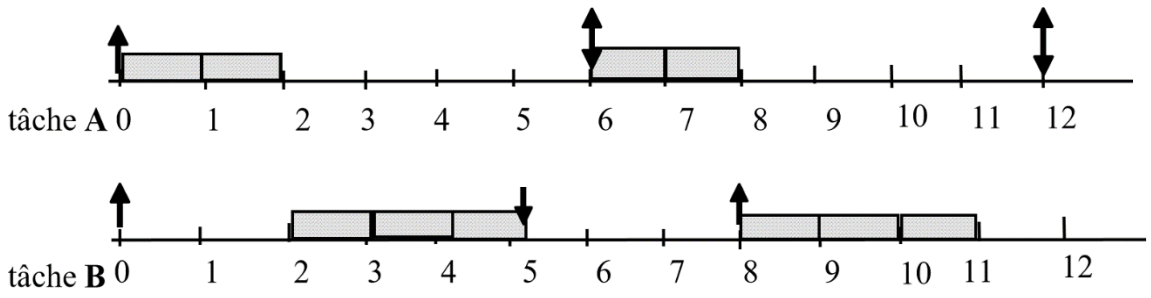
L'algorithme Rate Monotonic est un algorithme statique, applicable à un ensemble de tâches à échéance sur requête, où les priorités des tâches sont fixes et décidée avant le lancement du système. La priorité des tâches est fixée en fonction de leur période d'activation. Plus une tâche a une petite période d'activation, plus sa priorité sera haute. Il s'agit d'un algorithme préemptif,

l'activation d'une tâche de petite période devant permettre la préemption d'une tâche de période plus grande. Cet algorithme est optimal dans le cas des priorités fixes.

- La tâche de plus petite période est la tâche la plus prioritaire.
- Pour un ensemble de n tâches périodiques à échéance sur requête $T_{pi}(r_0, C_i, R_i, P_i)$, un test d'acceptabilité est:

Si condition vérifiée (**condition suffisante**) $\sum_{i=1}^{n} \frac{C_i}{P_i} \leq n\left(2^{\frac{1}{n}} - 1\right)$ alors la configuration est ordonnançable.

Example :



Tâche A (r0=0, C=2, R=6, P=6)

Tâche B (r0=0, C=3, R=5, P=8)

Test d'ordonnançabilité : $(2/6 + 3/8) <= 2(2^{1/2}-1)$, on a :

0.708 <= 0.82 donc les taches sont ordonnançables.

II.4.2 Algorithme d'ordonnancement Inverse Deadline (DM)

L'algorithme Rate Monotonic est basé sur l'hypothèse que les tâches sont à échéance sur requête, c'est-à-dire que l'échéance d'une tâche est égale à sa période. Dans certains systèmes cette hypothèse ne pourra s'avérer exacte, et les échéances pourront être plus petites que les périodes. Dans ce cas, l'algorithme Deadline Monotonic est un algorithme optimal dans le cas des algorithmes à priorité statique avec échéances plus petites que les périodes. Comme RM, il est préemptif, la priorité d'une tâche est fixée par son échéance. Plus son échéance est petite, plus sa priorité est grande.

- Priorité de la tâche est fonction de son délai critique. Priorité constante.
- La tâche de plus petit délai critique est la tâche la plus prioritaire
- Pour un ensemble de n tâches périodiques à échéance sur requête $T_{pi}(r_0, C_i, R_i, P_i)$, un test d'acceptabilité est:

Si condition vérifiée (**condition suffisante**) $\sum_{i=1}^{n} \frac{C_i}{P_i} \leq n\left(2^{\frac{1}{n}} - 1\right)$ alors la configuration est ordonnançable.

II.4.3 Algorithme d'ordonnancement Earliest Deadline (EDF)

L'algorithme Earliest Deadline First (EDF) donne la priorité à la tâche ayant l'échéance la plus proche. A chaque fois qu'une tâche est réveillée, l'ordonnanceur réévalue les tâches prêtes et sélectionne celle ayant l'échéance la plus courte. Cet algorithme est appliqué, ici, à des tâches périodiques à échéance sur requête où la préemption est autorisée.

- Priorité de la tâche est fonction de son délai critique dynamique. Priorité dynamique.
- A l'instant t, la tâche de plus petit délai critique dynamique (de plus proche échéance) est la tâche la plus prioritaire.
- Pour un ensemble de n tâches périodiques à échéance sur requête $T_{pi}(r_0, C_i, R_i, P_i)$, un test d'acceptabilité est:

Si condition vérifiée (**condition nécessaire et suffisante**) $\sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1$ alors la configuration est ordonnançable.

Chapitre III : Programmation concurrente

III.1 Introduction

On appelle programmation concurrente les techniques et notations permettant :

- L'expression et la manipulation (construction, destruction, lancement, arrêt, ...) d'entité concurrentes que l'on appelle tâches ou processus.
- La mise en place de moyens de communication et de synchronisation entre ces entités.
- L'abstraction de l'exécution réelle des processus ou tâches sur une architecture parallèle ou non.

III.2 Définitions

a) La concurrence

La possibilité que plusieurs sous-programmes d'un programme informatique ne soient pas exécutés en séquence, mais progressent séparément les uns des autres, plus ou moins indépendamment. Cela recouvre d'une part les situations dans lesquelles plusieurs éléments d'un programme doivent sembler fonctionner en même temps (par exemple, dans un navigateur Web, les sous-programmes chargés de la communication avec les serveurs, de l'affichage des pages, de l'interaction avec l'utilisateur, etc.) [3] .

b) Parallélisme

Qui désigne le fait que plusieurs opérations soient effectivement exécutées simultanément, avec un gain de performances [3] .

c) Section critique

Une section critique est une partie de code qui doit être traitée indivisiblement. Dès qu'une section critique commence son exécution, elle ne peut pas être interrompue. Pour assurer cela, les interruptions sont désactivées avant de rentrer dans le code de la section critique et réactivées quand le code est terminé [3] .

d) Exclusion mutuelle

Une ressource partagée est une ressource (LCD, port série, variable partagée, etc) qui peut être utilisée par plus d'une tâche. Chaque tâche doit gagner l'accès exclusif à la ressource partagée pour éviter la corruption des données, on appele ce mécanisme l'exclusion mutuelle qui est facilement implémentée grâce au sémaphore [3] .

e) Sémaphore

Un sémaphore est un objet possédant une valeur entière. Sa valeur est positive ou nulle et est uniquement manipulable à l'aide de deux opération wait(s) et signal(s) [3] :

- wait(s) décrémente s si s > 0; sinon, la tâche est suspendue (état bloqué).
- signal(s) si au moins une tâche est suspendue à la suite d'une opération wait(s), rend exécutable une de ces tâches; sinon incrémente s
- L'exécution d'une opération wait(s) ou signal(s) se fait sans intéraction possible (de façon atomique).

Notes :

- Les sémaphores binaires sont des variantes de sémaphores ne pouvant prendre que les valeurs 0 à 1.
- La sélection d'une tâche bloquée sur un sémaphore en vue de la rendre à nouveau exécutable se fait par priorité.

f) Mutex

Un mutex (pour exclusion mutuelle), ou verrou, est un objet dont l'état peut être soit verrouillé (ou pris) par un thread unique (qui est alors le propriétaire du mutex), soit déverrouillé (ou libre). Un thread peut devenir propriétaire d'un mutex grâce à une opération d'acquisition, bloquante, qui consiste à attendre que le mutex soit libre (s'il ne l'est pas déjà) avant de le prendre effectivement. Le thread doit alors relâcher le mutex pour que d'autres puissent le prendre [3] .

g) Interblocages (deadlock)

Un inter-blocage au sens large est une situation dans laquelle plusieurs threads causent le blocage définitif les uns des autres.

On dit d'un thread qu'il est bloqué si et seulement s'il est en attente pour verrouiller un ou plusieurs mutex. On dit qu'un thread progresse ssi. il n'est pas bloqué [3] .

h) Communication entre tâches

Le transfert correct de données d'une tâche à une autre soulève des problèmes plus importants que dans le cas d'un programme unique communiquant avec des routines d'interruption.

- Chacune des deux parties peut ici voir son exécution suspendue après chaque instruction afin de transférer l'usage du processeur à l'autre tâche.
- Les changements de contexte ne peuvent pas toujours être évités par une simple désactivation des interruptions. Cela ne peut se faire qu'en interagissant avec l'ordonnanceur [3] .

i) Spectre

C'est une vulnérabilité qui permet de faire en sorte qu'un programme accède à des emplacements arbitraires de la mémoire vive allouées à celui-ci. Un attaquant peut lire le contenu de la mémoire accédée ainsi, ce qui peut potentiellement lui permettre d'obtenir des données sensibles. Les protections contre les vulnérabilités Spectre se composent de plusieurs parties :

- Introduction d'instructions prohibant l'exécution spéculative dans le code machine produit (typiquement LFENCE pour architectures x86).
- Introduction de retpolines (construction permettant d'empêcher le processeur de spéculer).
- Utilisation de l'IBRS (Indirect Branch Restricted Speculation) des processeurs Intel qui permet d'inhiber la spéculation également [3] .

III.3 Systèmes d'exploitation temps réel

Un système d'exploitation temps réel, en anglais RTOS (real-time operating system) est un système d'exploitation pour lequel le temps maximum entre un stimulus d'entrée et une réponse de sortie est précisément déterminé. Ce système est doté de deux caractéristiques clés : le déterminisme (les tâches répétées sont effectuées dans un délai serré, alors que dans un système d'exploitation à usage général, ce n'est pas nécessairement le cas) et la Prévisibilité (nous savons combien de temps prendra une tâche, et qu'elle produira toujours le même résultat).

Un RTOS facilite la création d'un système temps réel, mais ne garantit pas que le résultat final respecte les contraintes temps réel, ce qui exige le développement correct du logiciel. Un RTOS n'a pas nécessairement pour but d'être performant et rapide, mais un RTOS fournit des services et des primitives qui, si elles sont utilisées correctement, peuvent garantir les délais souhaités [3,5] .

III.3.1 Comparaison entre un système d'exploitation OS et RTOS

En général, un système d'exploitation (OS) est responsable de la gestion des ressources matérielles d'un ordinateur et de l'hébergement des applications qui s'exécutent sur l'ordinateur. Un RTOS effectue ces tâches, mais est également spécialement conçu pour exécuter des applications avec un timing très précis et un haut degré de fiabilité. Cela peut être particulièrement important dans les systèmes de mesure et d'automatisation où les temps d'arrêt sont coûteux ou un retard de programme peut entraîner un risque pour la sécurité.

III.3.2 Catégories des RTOS

Les RTOS sont subdivisés en systèmes temps réel « soft » et systèmes en temps réel « hard ».

- Les systèmes temps réel souples (soft) fonctionnent en quelques centaines de millisecondes, à l'échelle d'une réaction humaine.
- Les systèmes en temps réel dur (hard), cependant, fournissent des réponses qui sont prévisibles en quelques dizaines de millisecondes ou moins.

III.3.3 Caractéristiques des RTOS

- Déterminisme : la répétition d'une entrée entraînera la même sortie.
- Haute performance : les systèmes RTOS sont rapides et réactifs, exécutant souvent des actions en une petite fraction du temps nécessaire à un système d'exploitation général.
- Sûreté et sécurité : les RTOS sont fréquemment utilisés dans les systèmes critiques lorsque les pannes peuvent avoir des conséquences catastrophiques, comme la robotique ou les contrôleurs de vol. Pour protéger ceux qui les entourent, ils doivent avoir des normes de sécurité plus élevées et des dispositifs de sécurité plus fiables.
- Planification basée sur la priorité : la planification prioritaire signifie que les actions affectées d'une priorité élevée sont exécutées en premier, et celles dont la priorité est inférieure viennent après. Cela signifie qu'un RTOS exécutera toujours la tâche la plus importante.
- Faible encombrement : par rapport à leurs homologues de système d'exploitation généraux, les RTOS ne pèsent qu'une fraction de la taille. Par exemple, Windows 10, avec les mises à jour post-installation, occupe environ 20 Go. VxWorks®, d'autre part, est environ 20 000 fois plus petit, mesuré en mégaoctets à un chiffre.

III.3.4 Applications des RTOS

Les RTOS peuvent être trouvés dans d'innombrables produits à travers le monde, VxWorks alimentant à lui seul plus de deux milliards d'appareils. Les systèmes, des moteurs de voiture aux télescopes pour l'espace lointain, en passant par les systèmes de guidage d'hélicoptères et les rovers martiens, utilisent des systèmes embarqués qui exécutent un système d'exploitation en temps réel.

Table III.1 : application des RTOS

Application et Development	**Télécommunication**	**Transport**	**Médicale**	**L'industrie**
- Contrôleur d'affichage de vol - Turbine moteur Drone - Des rovers extraterrestres	- Modem 5G - Modem satellitaire - Station de base	- Systèmes de sécurité fonctionnelle - Systèmes de freinage d'urgence - Systèmes d'avertissement du moteur	- Imagerie par résonance magnétique - Matériel de chirurgie - Ventilateurs	- Systèmes robotiques d'usine - Systèmes de sécurité - Moniteurs de vibrations pour le pétrole et le gaz

III.3.4 Liste des systèmes d'exploitation temps réel

Cette liste présente les systèmes d'exploitation temps réel. Un RTOS est un système d'exploitation pour les applications embarquées et temps réel permettant ainsi de garantir les contraintes et de fournir les services nécessaires au développement des systèmes liés à ces deux domaines [7].

Table III.2 listes des RTOS et leurs applications

Nom	Licence	Code source	Domaine d'utilisation
Ardence RTX	Propriétaire	Fermé	Extension MS Windows
BeRTOS	GNU GPL modifiée	Open source	Système embarqué
ChibiOS/RT	GNU GPL modifiée	Open source	Système embarqué, small footprint
CMX RTOS	Propriétaire	-	Système embarqué
Contiki	BSD	Open source	Système embarqué
DNIX	Propriétaire	Fermé	Usage général
DrRtos	Gratuit ?	Open?	-
eCos	GNU GPL modifiée	Open source	Usage général
eCosPro	GNU GPL modifiée et eCosPro license	Open source avec des parties non libres	Usage général
embOS	Propriétaire	Fermé	Système embarqué
EROS	GPL	Open source	Experimental research use
Femto OS	GPLv3	Open source	Système embarqué
FreeRTOS	GNU GPL modifiée	Open source	Système embarqué
Fusion RTOS	Gratuit	-	Semi-Usage général
Helium	Gratuit	-	-
LynxOS	Propriétaire	-	-
MaRTE OS	-	-	Système embarqué
MicroC/OS-II	Propriétaire	-	Système embarqué
Nano-RK	Mixed	Open source	Système embarqué
Neutrino	Propriétaire	Code source fourni	Microkernel
Nucleus OS	propriétaire	Code source fourni	Système embarqué
NuttX RTOS	BSD	Open source	Système embarqué, small footprint
OSE	Propriétaire	Fermé	Usage général
OS-9	-	-	-
OSEK	n/a	Specification	Système embarqué

Phar Lap ETS	-	-	-
PaulOS	GNU GPL	Open source	Système embarqué
PICOS18	GNU GPL	Open source	Système embarqué
Phoenix-RTOS	-	-	-
PikeOS	Propriétaire	Fermé	Micro-kernel
Prex	BSD	Open source	Micro-kernel
pSOS	Propriétaire	-	-
QNX	Mixed	-	Usage général
RSX-11	Propriétaire	-	-
RT-11	Propriétaire	-	Usage général
RTEMS	GNU GPL modifiée	-	Système embarqué
RTLinux	GNU GPL	Open source	Usage général
SCEPTRE	n/a	Spécification	Système embarqué
SHaRK	GNU GPL	Open source	-
SimpleAVROS	GPLv3	Open Source	Embedded
Symbian OS	-	-	-
Talon DSP RTOS	Propriétaire	-	Système embarqué (DSP)
ThreadX	Propriétaire	disponible en clientèle	-
Trampoline Operating System (OSEK)	GNU LGPL	Open source	Système embarqué
TNKernel	BSD	Open source	Système embarqué
Transaction processing facility	Propriétaire	-	Usage général
TRON Project	Open ?	Specification	Mixe
TUD:OS	GNU GPL	Open source	-
Ubuntu Studio	GNU GPL	Open source	Audio, image et sons
UNIX-RTR	-	-	-
u-velOSity	-	-	Micro-kernel
VRTX	-	-	-
VxWorks	Propriétaire	-	Système embarqué
Windows CE	Propriétaire	Microsoft Shared Source	Système embarqué
Xenomai	GPLv2	Open source	Générale
Erika Enterprise	GPL	Open source	-

De plus , Figure III.1 représente le nombre d'entreprise utilisant les systèmes RTOS dans le monde

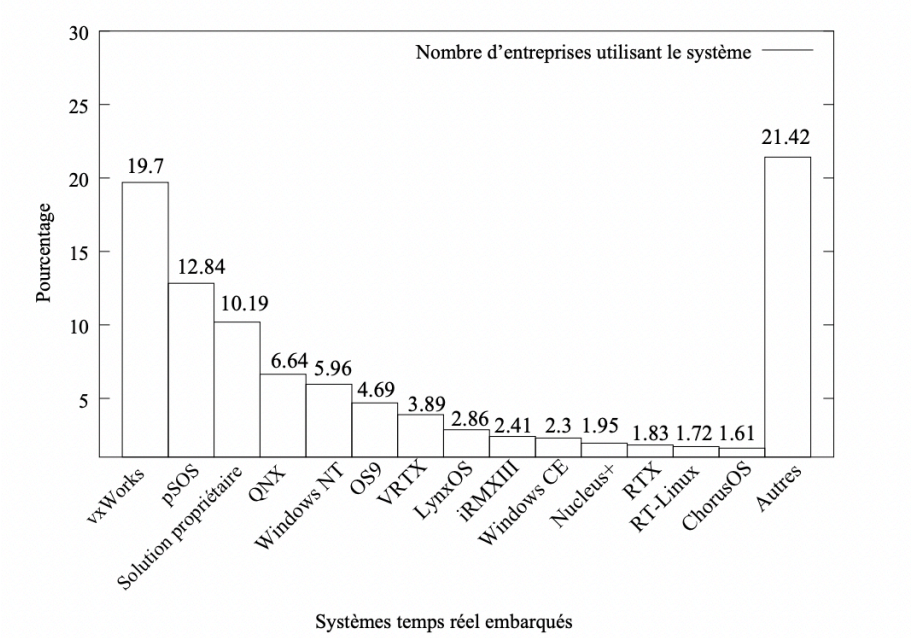


Figure III.1 Pourcentage d'utilisation des RTOS [7]

III.3.5 Exemples des systèmes d'exploitation temps réel

a) VxWorks

VxWorks est un système d'exploitation multitâche temps réel, généralement utilisé dans les systèmes embarqués. Cet OS de la firme Wind River, a été employé par la NASA pour les missions spatiales Mars Pathfinder, Stardust, ainsi que pour les deux rovers martiens Spirit et Opportunity. La sonde martienne Mars Reconnaissance Orbiter l'utilise également. Il est également utilisé dans de nombreux systèmes de communication d'entreprise (IP PBX Mitel ICP de Mitel Networks, Nortel Communications Server 1000).

Ce système d'exploitation est principalement utilisé pour la recherche et l'industrie. Plusieurs cartes mères peuvent être gérées à l'aide de ce système. Chaque carte mère possède à son tour des slots d'extension afin d'y ajouter des interfaces de toutes sortes (sondes de températures, réseau, écran, cartes d'entrées/sorties). Le langage utilisé avec ce système d'exploitation est le C/C++ [7].

b) OS-9

OS-9 est un système d'exploitation temps réel dur né en 1979-1980, et créé par la société Microware Systems Corporation. Le système d'exploitation en temps réel OS-9 fournit les éléments de base d'une gamme complète d'appareils embarqués. Ce système d'exploitation peut être mis sur une carte placée dans un rack VME, connecté par le biais d'une liaison série à un PC ou à une station de travail (RS/6000, etc.) ; il est alors possible via le logiciel Microware Hawk, fonctionnant sous Windows, de communiquer avec la carte en accédant à un interpréteur de commandes.

En 1999, 19 ans après la première version d'OS-9, Apple lance son système d'exploitation Mac OS 9. Cette année-là, Microware poursuivit Apple en raison de la marque déposée.. L'OS-9 sert de programme de contrôle fondamental qui gère les actions critiques des microprocesseurs. OS-9 a une base supérieure éprouvée qui est mature, fiable et sécurisée. OS-9 est robuste et offre pourtant un code plus fini que tout autre RTOS disponible dans le commerce. Cela réduit les coûts et aide à mettre votre produit fini sur le marché en premier. OS-9 est un système d'exploitation en temps réel évolutif, multithread et tolérant aux pannes qui prend en charge toutes les familles de processeurs populaires. Il est disponible pour Windows 95, 98, 2000, Windows NT et Windows ME [7].

c) VRTX

Versatile Real-Time Executive (VRTX) est un système d'exploitation temps réel (RTOS) développé et commercialisé par la société Mentor Graphics. VRTX convient à la fois aux systèmes embarqués traditionnels basés sur carte et aux architectures de système sur puce (SoC). Il a été remplacé par le Nucleus RTOS.

Le système d'exploitation VRTX a commencé comme un produit de Hunter & Ready, une société fondée par James Ready et Colin Hunter en 1980, qui est devenue plus tard Ready Systems. Cette société a ensuite fusionné avec Microtec Research en 1993 et est devenue publique en 1994. Cette société a ensuite été acquise par Mentor Graphics en 1995, et VRTX est devenu un produit Mentor. Depuis les années 1980, le principal rival de VRTX est VxWorks, un produit de Wind River Systems. VxWorks a fait ses débuts au milieu des années 1980 en tant qu'outils de compilateur et de langage d'assemblage pour compléter VRTX, nommé VRTX works ou VxWorks. Plus tard, Wind River a créé sa propre offre de noyau en temps réel similaire à VRTX [7] .

Chapitre IV : Langage de programmation en TR

IV.1 JAVA

IV.1.1 Introduction

Java est un langage de programmation orienté objet, basé sur des classes et à usage général, conçu pour avoir moins de dépendances d'implémentation. C'est une plate-forme informatique pour le développement d'applications. Java est donc rapide, sécurisé et fiable. Il est largement utilisé pour développer des applications Java dans les ordinateurs portables, les centres de données, les consoles de jeux, les superordinateurs scientifiques, les téléphones portables, etc.

La platforme Java est une collection de programmes qui aident les programmeurs à développer et à exécuter efficacement des applications de programmation Java. Il comprend un moteur d'exécution, un compilateur et un ensemble de bibliothèques. Il s'agit d'un ensemble de logiciels informatiques et de spécifications. James Gosling a développé la plate-forme Java chez Sun Microsystems, et Oracle Corporation l'a ensuite acquise [8].

IV.1.2 Histoire du langage de programmation Java

Le langage Java s'appelait initialement OAK. À l'origine, il a été développé pour la manipulation d'appareils portables et de décodeurs, mais OAk a été un échec massif. En 1995, Sun a changé le nom en "Java" et a modifié le langage pour tirer parti de l'activité de développement en plein essor de www (World Wide Web). Plus tard, en 2009, Oracle Corporation a acquis Sun Microsystems et a pris possession de trois actifs logiciels clés de Sun : Java, MySQL et Solaris[8].

IV.1.3 Utilisation de Java

- Développer des applications Android ;
- Vous aide à créer des logiciels d'entreprise ;
- Large gamme d'applications Java mobiles ;
- Applications de calcul scientifique ;
- Utilisation pour Big Data Analytics ;
- Programmation Java des périphériques matériels ;
- Utilisé pour les technologies côté serveur comme Apache, JBoss, GlassFish, etc.

IV.1.4 Fonctionnalités Java

Voici quelques importantes fonctionnalités Java:

- C'est l'un des langages de programmation faciles à apprendre.
- Écrivez le code une seule fois et exécutez-le sur presque toutes les plates-formes informatiques.
- Java est indépendant de la plate-forme. Certains programmes développés sur une machine peuvent être exécutés sur une autre machine.
- Il est conçu pour créer des applications orientées objet.
- C'est un langage multithread avec gestion automatique de la mémoire.
- Il est créé pour l'environnement distribué d'Internet.
- Facilite l'informatique distribuée en tant que centrée sur le réseau.

IV.1.5 Composants du langage de programmation Java

Un programmeur Java écrit un programme dans un langage lisible par l'homme appelé code source. Par conséquent, le CPU ou les puces ne comprennent jamais le code source écrit dans aucun langage de programmation. Ces ordinateurs ou puces ne comprennent qu'une seule chose, appelée langage machine ou code. Ces codes machine s'exécutent au niveau du processeur. Par conséquent, il s'agirait de codes machine différents pour d'autres modèles de CPU. Cependant, vous devez vous soucier du code machine, car la programmation concerne uniquement le code source. La machine comprend ce code source et le traduit en code compréhensible par la machine, qui est un code exécutable. Toutes ces fonctionnalités se produisent à l'intérieur des 3 composants de plateforme Java suivants [8] :

a) Kit de développement Java (JDK)

JDK est un environnement de développement logiciel utilisé pour créer des applets et des applications Java. La forme complète de JDK est Java Development Kit. Les développeurs Java peuvent l'utiliser sous Windows, macOS, Solaris et Linux. JDK les aide à coder et à exécuter des programmes Java. Il est possible d'installer plusieurs versions du JDK sur le même ordinateur. Voici les principales raisons d'utiliser JDK :

- JDK contient les outils nécessaires pour écrire des programmes Java et JRE pour les exécuter.
- Il comprend un compilateur, un lanceur d'applications Java, Appletviewer, etc.
- Le compilateur convertit le code écrit en Java en code binaire.
- Le lanceur d'applications Java ouvre un JRE, charge la classe nécessaire et exécute sa méthode principale.

b) Machine virtuelle Java (JVM)

Java Virtual Machine (JVM) est un moteur qui fournit un environnement d'exécution pour piloter le code Java ou les applications. Il convertit le bytecode Java en langage machine. JVM fait partie de l'environnement d'exécution Java (JRE). Dans d'autres langages de programmation, le compilateur produit du code machine pour un système particulier. Cependant, le compilateur Java produit du code pour une machine virtuelle appelée machine virtuelle Java. Voici les raisons importantes d'utiliser JVM :

- JVM fournit un moyen indépendant de la plate-forme d'exécuter le code source Java.
- Il possède de nombreuses bibliothèques, outils et frameworks.
- Une fois que vous avez exécuté un programme Java, vous pouvez l'exécuter sur n'importe quelle plate-forme et gagner beaucoup de temps.
- JVM est livré avec un compilateur JIT (Just-in-Time) qui convertit le code source Java en langage machine de bas niveau. Par conséquent, il s'exécute plus rapidement qu'une application régulière.

c) Environnement d'exécution Java (JRE)

JRE est un logiciel conçu pour exécuter d'autres logiciels. Il contient les bibliothèques de classes, la classe de chargeur et la JVM. En termes simples, si vous souhaitez exécuter un programme Java, vous avez besoin de JRE. Si vous n'êtes pas programmeur, vous n'avez pas besoin d'installer JDK, mais juste JRE pour exécuter des programmes Java. Voici les principales raisons d'utiliser JRE :

- JRE contient des bibliothèques de classes, JVM et d'autres fichiers de support. Il n'inclut aucun outil de développement Java comme un débogueur, un compilateur, etc.
- Il utilise des classes de packages importantes telles que les bibliothèques math, swing, util, lang, awt et runtime.
- Si vous devez exécuter des applets Java, JRE doit être installé sur votre système.

IV.1.6 La programmation en Java

Un programme Java comprend les sections suivantes :

- Section Documentation ;
- Déclaration de packages ;
- Importer des instructions ;
- Déclaration d'interface ;
- Définition de classe ;
- Classe de méthode principale ;
- Définition de la méthode principale.

Section	La description
Section Documentation	Vous pouvez écrire un commentaire dans cette section. Les commentaires sont bénéfiques pour le programmeur car ils l'aident à comprendre le code. Ceux-ci sont facultatifs, mais nous vous suggérons de les utiliser car ils sont utiles pour comprendre le fonctionnement du programme, vous devez donc écrire des commentaires dans le programme.
Déclaration de packages	Vous pouvez créer un package avec n'importe quel nom. Un package est un groupe de classes définies par un nom. Autrement dit, si vous souhaitez déclarer plusieurs classes dans un élément, vous pouvez le déclarer dans un package. Il s'agit d'une partie facultative du programme, c'est-à-dire que si vous ne souhaitez déclarer aucun paquet, il n'y aura aucun problème et vous n'obtiendrez aucune erreur. Ici, le package est un mot clé qui indique au compilateur que le package a été créé. Il est déclaré comme suit : package package_name;
Importer des instructions	Cette ligne indique que si vous souhaitez utiliser une classe d'un autre package, vous pouvez le faire en l'important directement dans votre programme. Exemple: import calc.add;

Déclaration d'interface	Les interfaces sont comme une classe qui inclut un groupe de déclarations de méthode. C'est une section facultative et peut être utilisée lorsque les programmeurs souhaitent implémenter plusieurs héritages dans un programme.
Définition de classe	Un programme Java peut contenir plusieurs définitions de classe. Les classes sont les éléments principaux et essentiels de tout programme Java.
Classe de méthode principale	Chaque programme autonome Java nécessite la méthode principale comme point de départ du programme. C'est une partie essentielle d'un programme Java. Il peut y avoir plusieurs classes dans un programme Java, et **une seule classe définit la méthode main** . Les méthodes contiennent une déclaration de type de données et des instructions exécutables.

Voici un exemple du programme **Hello Java** pour comprendre la structure et les fonctionnalités des classes. Il y a quelques lignes dans le programme, et la tâche principale du programme est d'imprimer le texte Hello Java à l'écran.

```
// Nom de fichier "Hello.java"

public class Hello
{
   /* Author: K.benmouiza
   Date: 2021-04-28
   Description: Programme Java  */

   public static void main(String[] args)
   {
      System.out.println("Hello Java");
   }
}
```

L'explication de chaque ligne de commande de ce programme est illustrée dans le tableau suivant.

public class Hello	• Cela crée une classe appelée Hello. • Tous les noms de classe doivent commencer par une majuscule. • Le mot public signifie qu'il est accessible depuis n'importe quelle autre classe.
/* Commentaires */	Le compilateur ignore le bloc de commentaires. Le commentaire peut être utilisé n'importe où dans le programme pour ajouter des informations sur le programme ou le bloc de code, ce qui sera utile aux développeurs pour comprendre facilement le code existant à l'avenir.
Accolades	Deux accolades **{...}** sont utilisées pour regrouper toutes les commandes, ainsi on sait que les commandes appartiennent à cette classe ou méthode.

public static void main	• Lorsque la méthode main est déclarée **public** , cela signifie qu'elle peut également être utilisée par du code en dehors de sa classe, grâce à quoi la méthode main est déclarée public. • Le mot **statique** est utilisé lorsque nous voulons accéder à une méthode sans créer son objet, comme nous appelons la méthode principale, avant de créer des objets de classe. • Le mot **void** indique qu'une méthode ne renvoie pas de valeur. main() est déclaré void car il ne renvoie pas de valeur. • **main** est une méthode ; c'est le point de départ d'un programme Java. Vous remarquerez que le code de la méthode principale a été déplacé vers quelques espaces restants. C'est ce qu'on appelle l' **indentation** qui rendait un programme plus facile à lire et à comprendre.
String[] args	C'est un tableau où chaque élément est une chaîne, qui a été nommée "args". Si votre programme Java est exécuté via la console, vous pouvez transmettre le paramètre d'entrée et la méthode main() le prend en entrée.
System.out.println() ;	Cette instruction est utilisée pour imprimer du texte à l'écran en tant que sortie, où **le système** est une classe prédéfinie et **out** est un objet de la classe PrintWriter définie dans le système. La méthode **println** imprime le texte à l'écran avec une nouvelle ligne. Vous pouvez également utiliser la méthode **print()** au lieu de la méthode println(). Toutes les instructions Java se terminent par un point-virgule.

IV.2 Modula

Modula et son successeur Modula-2 sont des langages de programmation impératifs fortement typés développés dans les années 1970 par Niklaus Wirth. MODULA est un langage relativement petit et concis. Le choix des fonctionnalités du langage est fortement biaisé en faveur de constructions qui peuvent être implémentées efficacement et qui nécessitent peu ou pas de support d'exécution. En particulier, il n'y a pas de tableaux dynamiques, le mécanisme de paramétrage est très simple, la structure du processus est monolithique et il n'y a pas d'E/S de haut niveau.

Le langage de programmation MODULA a été conçu par Wirth pour la programmation de petits ordinateurs dédiés utilisés dans l'environnement temps réel. La langue est basée sur PASCAL; ainsi qu'un ensemble simple de fonctionnalités algorithmiques, MODULA contient des fonctionnalités de multiprogrammation et des mécanismes spécifiques à la machine pour référencer le matériel périphérique. Une extension de la structure de bloc est utilisée pour structurer les programmes, permettant un contrôle explicite de la communication des noms entre des parties distinctes (modules) du programme [9].

IV.2.1 La programmation en Modula

Un programme Modula comprend les sections suivantes [9] :

- Module ;
- Processus ;
- Module d'interface ;
- Module de périphérique ;
- Procédure .

Section	La description
Module	Un module est une collection de constantes, de variables, de types et de procédures. La structure du module constitue une "barrière" autour des objets inclus à travers laquelle seuls les noms spécifiquement "importés" dans ou "exportés" depuis le module peuvent passer. Les variables exportées sont en lecture seule en dehors du module. Les instructions d'initialisation, exécutées lors de la saisie de la portée contenant la déclaration du module, sont fournies à la fin du module.
Processus	Syntaxiquement similaire à une procédure, mais exécuté simultanément lorsque le processus est appelé. Les processus ne peuvent être déclarés qu'au niveau le plus externe du programme et ne peuvent être appelés qu'au niveau le plus externe.
Module d'interface	Un seul processus peut être actif dans un module d'interface à la fois, bien que d'autres processus puissent envoyer ou attendre un signal dans ce module. Les variables partagées entre processus peuvent résider dans des modules d'interface.
Module de périphérique	Les constructions de langage spécifiques à la machine pour permettre l'utilisation de matériel périphérique ne peuvent être utilisé dans des modules particuliers qui sont désignés comme modules d'appareil. Il est possible que la syntaxe de ces fonctionnalités soit également spécifique à la machine.
Procédure	Peut avoir à la fois des paramètres constants (en lecture seule) et variables. Tous les paramètres réels des types non fondamentaux peuvent être passés par référence.

IV.2.3 Caractéristique du langage Modula

La syntaxe générale de Modula est celle de Pascal. La différence majeure étant l'usage moins fréquent du mot clé BEGIN, et le remplacement du mot clé PROGRAM par MODULE, IMPLEMENTATION MODULE ou DEFINITION MODULE selon les cas. Comme dans le langage Pascal, les mots réservés et noms des éléments de base (fonctions, types et constantes) sont écrits en majuscule.

Voici un exemple du programme **Hello world** pour comprendre la structure et les fonctionnalités des classes du langage Modula. La tâche principale du programme est d'imprimer le texte Hello world à l'écran.

```
MODULE HPrintHelloWorld

(*This program prints "Hello world!" on the
standard output device*)

FROM InOut IMPORT WriteString, WriteLn;

BEGIN

    WriteString('Hello world!');

    WriteLn;

END PrintHelloWorld.
```

IV.3 ADA

Ada est un langage algorithmique moderne avec les structures de contrôle habituelles et avec la possibilité de définir des types et des sous-programmes. Il répond également au besoin de modularité, grâce auquel les données, les types et les sous-programmes peuvent être regroupés. Il traite également la modularité au sens physique, avec une possibilité de prendre en charge une compilation séparée.

En plus de ces aspects, le langage prend en charge la programmation en temps réel, avec des fonctionnalités permettant de définir l'invocation, la synchronisation et la synchronisation des tâches parallèles. Il prend également en charge la programmation des systèmes, avec des fonctionnalités permettant l'accès aux propriétés dépendantes du système et un contrôle précis de la représentation des données.

IV.3.1 Historique de langage ADA

Dans les années 1970, le Département de la Défense des États-Unis (DOD) a souffert d'une explosion du nombre de langages de programmation, avec différents projets utilisant des dialectes différents et non standard ou des sous-ensembles/sur-ensembles linguistiques. Le DOD a décidé de résoudre ce problème en lançant une demande de propositions pour un langage de programmation commun et moderne. La proposition gagnante a été soumise par Jean Ichbiah de CII Honeywell-Bull.

La première norme Ada a été publiée en 1983; il a ensuite été révisé et amélioré en 1995, 2005 et 2012, chaque révision apportant de nouvelles fonctionnalités utiles.

Ce didacticiel se concentrera sur Ada 2012 dans son ensemble, plutôt que sur l'enseignement de différentes versions du langage.

Aujourd'hui, Ada est largement utilisé dans les systèmes temps réel embarqués, dont beaucoup sont critiques pour la sécurité. Alors qu'Ada est et peut être utilisé comme un langage à usage général, il brillera vraiment dans les applications de bas niveau :

- Systèmes embarqués nécessitant peu de mémoire (pas de ramasse-miettes autorisé).
- Interfaçage direct avec le matériel.
- Systèmes temps réel soft ou hard.
- Programmation de systèmes de bas niveau.

IV.3.2 Domaines d'utilisation du langage ADA

Les domaines spécifiques voyant l'utilisation d'Ada incluent l'aérospatiale et la défense, l'aviation civile, le rail et bien d'autres. Ces applications nécessitent un haut degré de sécurité : un défaut logiciel n'est pas seulement une gêne, mais peut avoir de graves conséquences. Ada fournit des fonctionnalités de sécurité qui détectent les défauts à un stade précoce, généralement au moment de la compilation ou à l'aide d'outils d'analyse statique. Ada peut également être utilisé pour créer des applications dans une variété d'autres domaines, tels que [10]:

- Programmation de jeux vidéo ;
- Audio en temps réel ;
- Modules du noyau.

IV.3.3 Structure du programme ADA

Structure:

procédure xxx est

... // Les déclarations vont ici

begin // L'exécution commence ici

... // Les instructions exécutables vont ici

fin xxx ;

- Une procédure est similaire à une méthode ;
- Le nom de la procédure xxx est choisi par l'utilisateur ;
- begin ... end marque un bloc de programme, comme {...} en Java.

Voici un exemple du programme **Hello world** pour montrer la syntaxe d'un langage ADA. La tâche principale du programme est d'imprimer le texte Hello world à l'écran.

```
with Ada.Text_IO; -- Bibliothèque
-- Déclaration de la procédure "Hello"
procedure Hello is
begin
  -- Imprimer "Hello, world!" à l'écran
  Ada.Text_IO.Put_Line("Hello, world!");
end Hello;
```

Conclusion générale

Les systèmes en temps réel couvrent une grande partie de l'industrie informatique. Jusqu'à présent, la plupart des recherches sur les systèmes en temps réel se sont principalement limitées aux systèmes à nœud unique et principalement à l'ordonnancement des processeurs. Cela doit être étendu pour plusieurs ressources et nœuds distribués.

Les systèmes temps réel s'étendent à plusieurs autres domaines tels que l'industrie automobile et les systèmes embarqués temps réel. En particulier, le mariage d'Internet avec des applications multimédias a ouvert plusieurs nouvelles applications en volume.

Références bibliographiques

[1] Francis Cottet, Emmanuel Grolleau, Sébastien Gérard, Jérome Hugues, Yassine Ouhammou, Systèmes temps réel embarqués : Spécification, conception, implémentation et validation temporelle, - 2e édition, Dunod, 2014.

[2] Francis Cottet, Emmanuel Grolleau, Systèmes temps réel de contrôle-commande : Conception et implémentation Relié, Dunod, 2005.

[3] B. Nichols, D. Buttlar, J. Proulx Farrel, O'Reilly, P threads programming, (1996)

[4] Maryline Chetto, Ordonnancement dans les systèmes temps réel, ISTE, 2014.

[5] Jane W. S. Liu, « Real-time Systems », Prentice Hall, 2000

[6] Christian Bonnet. Isabelle Demeure, Introduction aux systèmes temps réel. Collection pédagogique de télécommunications, Hermès, septembre 1999.

[7] A. Dorseuil and P. Pillot. Le temps réel en milieu industriel. Edition DUNOD, Collection Informatique Industrielle, 1991.

[8] Y. Daniel Liang. Introduction to Java Programming (8th. ed.). Prentice Hall Press, USA. 2010.

[9] W. L. John Ogilvie. MODULA-2 programming. McGraw-Hill, Inc., USA,1985.

[10] Andrew T. Shvets. Beginning Ada Programming (1st. ed.)). Apress Berkeley, CA,USA.2020.