Module Master M1

Systèmes temps réel et Informatique Industrielle
Chapitre IX : Temps réel industriel

Projet n°06 d'exposé des étudiants de Master M1 Informatique

Présenté par : Prof. Kholladi Mohamed-Khireddine
Département d'Informatique
Facultés des Sciences Exactes
Université Echahid Hamma Lakhdar d'El Oued
Tél. 0770314924
Email. kholladi@univ-eloued.dz et kholladi@yahoo.fr
Site Web. www.univ-eloued.dz
http://kholladi.doomby.com/ et http://kholladi.e-monsite.com/

IX – Temps réel industriel

IX.1 – Introduction

En informatique temps réel, on distingue habituellement deux domaines l'informatique de gestion, au sens large et l'informatique industrielle.

IX.2 – Informatique de gestion au sens large

Ici, on va énumérer les quatre les fonctions d'informatique de gestion au sens large suivant : la gestion de bases de données, l'élaboration de courbes et de tableaux, les calculs longs et fastidieux, les affichages, et les graphiques.

Exemple d'informatique de gestion en temps réel :

- Gestion de stock dans un magasin :
  - Le réceptionniste ajoute des marchandises dans le stock quand on les lui apporte,
  - Le vendeur consulte le stock quand un client lui demande la disponibilité d'un article,
  - Le livreur enlève des marchandises du stock quand il les amène chez un client.
- Ce sont des transactions qui s'exécutent en temps réel pour l'échelle humaine (de l'ordre de la seconde).

L'ordre de grandeur du temps de réponse selon les domaines de gestion.

Domaine	Ordre de grandeur du temps de réponse
Gestion de stock de magasin en temps réel	la seconde, la minute
L'information boursière en temps réel	la minute, l'heure
La météo en temps réel	l'heure, le jour
Informations sur les séismes en temps réel	l'heure, le jour, le mois

IX.3 – Informatique industrielle

En informatique industrielle, le temps réel signifie le traitement déterministe d'événements survenant sur des procédés généralement rapides.

- Temps déterministe : temps de traitement borné et reproductible.
- Evénement : situation gérable par une variable binaire Tout Ou Rien ("T.O.R."), comme par exemple : le dépassement d'un seuil, le débordement d'un compteur et l'arrivée d'un message.

Il n'y a pas à proprement parler de données comme en informatique de gestion.

Exemples d'informatique en temps réel industriel :

En surveillance incendie,

- l'information température est importante,
- l'information "feu" est prioritaire.

Domaine	Ordre de grandeur du temps de réponse
Compression de donnés en temps réel	la nanoseconde
Cartes d'acquisition en temps réel pour PC	la microseconde ou moins

IX.4 – Système d’exploitation

Le temps réel est fortement lié au système d'exploitation, qui le rend possible ou non. Il y a différentes familles de systèmes d'exploitation. Dans ses systèmes, seront mis en relief les mécanismes temps réel et les contraintes qui en découlent. Il existe un grand nombre d'implémentations de systèmes d'exploitation temps réel, qui se ressemblent beaucoup et il est illusoire de vouloir toutes les étudier. Le système d'exploitation iRMX développé par INTEL et repris par TenAsys est intéressant, car il présente tous les aspects des systèmes informatiques industriels temps réel. Les différences d'implémentation avec d'autres systèmes d'exploitation peuvent être mises en évidence : VMS, UNIX, et surtout Windows NT.

On va citer les différents types systèmes de machines : les fonctions d'un système d'exploitation, la conduite de procédé industriel, l'ordinateur personnel, le système transactionnel, le système en temps partagé et le système parallèle.

a - Représentation en couches logicielles

Le tableau suivant illustre la représentation en couches logicielles.

Logiciel d'Application
Système d'exploitation
Machine Physique

Cette représentation en couches logicielles suppose que la couche inférieure présente à la couche supérieure une interface pour utiliser les services qu'elle propose. Ceci permet la portabilité des logiciels applicatifs : si tous les accès au matériel sont effectués par l'intermédiaire du système d'exploitation, le changement de machine physique devient un problème de système d'exploitation, et pas d'application. Par contre, si l'application fait des appels en direct au matériel (c'est à dire sans passer par le système d'exploitation), le changement de machine physique implique des modifications logicielles de l'application.

b - La couche système d'exploitation

La couche système d'exploitation est l'interface entre l'application et la machine physique, le système d'exploitation définit une nouvelle machine virtuelle (**DOS, iRMX, UNIX, VMS, Windows**, etc.). La frontière entre logiciel de base et logiciel d'application n'est pas fixe :

- Un logiciel d'application peut très bien être intégré au logiciel de base s'il se révèle être un outil d'usage courant pour une large classe d'usagers.
- De même, un programme peut être réalisé par un microprogramme ou un circuit si l'économie ou le gain en performances le justifient.

c - Fonctions d'un système d'exploitation

Il existe plusieurs fonctions d'un système d'exploitation telles que la gestion de l'information, le développement et la mise au point de programmes, l'exploitation de

programmes et les offres de plusieurs machines virtuelles tournant sur une même machine physique.

d - Conduite de procédés industriels

d .1 – Fonctions

Il existe plusieurs fonctions de conduite de procédés industriels telles que la régulation, le suivi de production, de la qualité, l'enregistrement, le rejet, l'apprentissage, la boîte noire, la sécurité et la conservation des fonctions essentielles.

d.2 – Contraintes

Il existe deux types de contraintes ; temporelles et sécuritaires.

- Temporelles :
  - t = temps de réponse du système,
  - T = période d'échantillonnage des mesures,
  - Le système ne fonctionne que si t est inférieur à T.
- Sécuritaires :
  - La conduite de procédés industriels nécessite une sûreté de fonctionnement
  - proche du parfait, ce qui amène des contraintes de type redondance des systèmes, qui n'existent pas dans d'autres domaines informatiques.

e - Systèmes transactionnels

e.1 –Caractéristiques

On cite quelques types de caractéristiques tels que la gestion d'informations de taille importante, l'exécution de transactions souvent interactives et il y a beaucoup de transactions simultanées.

e.2 – Exemples

On peut avoir plusieurs exemples : les réservations de train ou d'avion, la gestion de compte bancaire, la consultation de documents, la gestion de stock de magasin, etc.

f - Systèmes en temps partagé

f.1 - Services fournis

Les services équivalents à une machine individuelle et les services liés à l'existence d'une communauté d'usagers : le partage de l'information et la communication entre usagers.

f.2 – Caractéristiques

On cite quelques types de caractéristiques : la définition de la machine virtuelle offerte à chaque usager, le partage et l'allocation des ressources physiques communes et la gestion de l'information partagée (fichiers et communications).

g - Systèmes parallèles

On distingue deux architectures. Les architectures parallèles fortement couplées, fonctionnant par partage de ressources (mémoire, interruptions, etc.), par exemple : Le Multibus I, le bus PC EISA et le Le bus VME, ainsi que les architectures parallèles faiblement couplées, fonctionnant par échange de messages, par exemple : le Multibus II, le bus PCI et les Transputers.

IX.5 - Mécanismes temps réel

On va citer quelques mécanismes temps réel : les multitâches, l'ordonnanceur de tâches, scheduler, la réentrance et récursivité, la synchronisation (sémaphores, régions), les échanges entre tâches (mémoire partagé, boîtes aux lettres, mailbox), les mécanismes d'empilement et de dépilement, les interruptions matérielles (hardware), les interruptions logicielles (software), le traitement des comptes-rendus système et les contraintes.

a - Multitâches

Si un programme ne traite qu'un seul événement, et que cet événement arrive à des instants prévisibles, ce programme peut être mono tâche, en attente de cet événement. Sinon, le seul moyen est de décomposer le traitement en au moins deux phases :

- Une première phase pour prendre en compte l'événement, avec un temps d'exécution le plus court possible;
- Une ou plusieurs phases suivantes pour effectuer les travaux qui prennent du temps et qui ne participent pas à la prise en compte des événements.

Dans la majorité des cas de temps réel industriel, les événements surviennent à des instants non prévisibles, mais rarement en flot continu. Deux cas particuliers sont à retenir, les avalanches d'événements dus :

- À la perte simultanée d'un grand nombre de capteurs, suite à une coupure du courant électrique alimentant ces capteurs, par exemple;
- À la perte de liaisons avec ces capteurs, retour simultané d'un grand nombre de capteurs, suite à la mise sous tension initiale, au retour du courant électrique alimentant ces capteurs, ou au retour des liaisons avec ces capteurs.

b - Ordonnanceur de tâches (scheduler)

Définition d'une tâche : une tâche est une activité, c'est à dire du point de vue du microprocesseur, un objet qui consomme des ressources de la machine informatique. Les ressources sont de la mémoire (une tâche a besoin de RAM pour tourner), et du temps CPU (le temps qu'une tâche s'exécute, les autres n'ont pas de temps CPU). A tout instant, une tâche a un état. On dit que la tâche est :

- Active (running) : c'est le cas de la tâche qui est en train de s'exécuter. Quand il n'y a qu'un processeur, une seule tâche est active à un moment donné, pas forcément la plus prioritaire;
- Prête (ready) : c'est le cas des tâches qui sont en attente de pouvoir s'exécuter. La priorité de ces tâches est inférieure ou égale à la priorité de la tâche active,
- Endormie (sleep) : c'est le cas des tâches qui sont en attente d'événements qui provoqueront leur réveil (interruption, réception de message ou fin d'un décompte de timer). Une tâche plus prioritaire que la tâche active peut être dans cet état.

L'ordonnanceur de tâches bascule le traitement d'une activité sur une autre en passant la tâche active dans l'état prête ou endormie ou en passant une tâche de l'état prête ou endormie à l'état active. Le temps de basculement de contexte (latency time) est un facteur de qualité du scheduler. Plus ce temps est faible, meilleur est le scheduler, car ce temps est considéré comme du temps perdu, où il ne se passe rien d'efficace.

b.1 - Modes de fonctionnement d'un ordonnanceur de tâches

Le fonctionnement en temps partagé (time sharing) : partage du temps CPU entre les différentes activités. L'inconvénient est que plus il y a de tâches, moins ces tâches ont de temps pour tourner. Le temps d'allocation du temps CPU diminue en fonction du nombre

d'utilisateurs. En général, ce type de fonctionnement correspond à un environnement de développement ou à des applications de gestion.

Le fonctionnement basé sur la priorité (priority based): utilise la préemptivité. Un traitement plus prioritaire préempte le moins prioritaire en cours, pour prendre en compte un événement le plus rapidement possible. Ce type de fonctionnement correspond au temps réel industriel.

Le fonctionnement en tourniquet (round robbin) : association des deux modes précédents. A partir d'un seuil de priorité, on bascule d'un mode de fonctionnement vers l'autre. Ce type de fonctionnement est particulièrement adapté à des machines utilisées à la fois pour des besoins temps réel, et pour des besoins de développement ou de gestion.

b.2 - Réentrance et récursivité

La récursivité est la possibilité pour une procédure de s'appeler elle même. Cette récursivité est employée pour un traitement séquentiel.

Un exemple de récursivité est le calcul de la factorielle d'un nombre :

n (n! = n * (n-1) * (n-2) *... 1, avec 0! = 1)

```
int Factorielle ( int n )
{
  int rep ;
  if ( n> 0 )
     rep = n * Factorielle ( n-1 ) ;
  else
     rep = 1 ;
  return rep ;
}
```

La réentrance, qui est une notion proche de la récursivité, est utilisée en multitâches pour permettre d'interrompre un traitement en cours par un traitement plus prioritaire qui reprend le traitement interrompu. Un exemple de réentrance est le cas d'un logiciel à bord d'une carte d'acquisition qui gère plusieurs lignes identiques. Le traitement de l'information est le même quelle que soit la ligne (par exemple sur une carte 8 lignes vers 8 automates). Il n'y a pas de raison de privilégier une ligne plutôt qu'une autre. Aussi le logiciel multitâche sera constitué d'une procédure réentrante unique de gestion d'une ligne, et de 8 tâches définies sur cette procédure.

Chaque tâche reconnaît la ligne qu'elle gère à l'aide de l'indice de ligne de 0 à 7 :

```
void Gestion_Ligne ( )
{
  // Gestion informatique de la ligne ;
}

for ( i=0 ; i < 8 ; i ++ )
{
  // Creation des taches, depend de l'OS
  Reference_Tache ( i ) = Creation_Tache (
@ Gestion_Ligne ) ;
}
```

c - Synchronisation

c.1 - Nécessité d'une synchronisation

La nécessité d'une synchronisation découle du fait que sur toute machine physique, il y a des ressources critiques. Ces ressources peuvent être : la mémoire principale, la mémoire secondaire (disque dur, disquette, lecteur de bandes magnétiques), une imprimante, un écran, etc. Par exemple, si deux tâches peuvent accéder à une imprimante et qu'aucune précaution n'est prise, il y a risque de collision :

- Supposons la tâche A qui imprime "BONJOUR CR LF", la tâche B qui imprime "BONSOIR CR LF".
- Le scheduler fonctionne en temps partagé, ou basé sur la priorité, les deux tâches ayant la même priorité.
- Le temps d'allocation est celui de la préparation d'envoi d'un caractère plus l'envoi d'un caractère.
- Le résultat de l'impression sera : BBOONNJSOOUIRR CR LF
- (ligne blanche : action du deuxième CR LF)
- (ligne disponible pour une nouvelle impression).
- C'est à dire un listing incompréhensible résultat des deux impressions des deux tâches.

c.2 - Moyens de synchronisation

Le Flag (drapeau ou variable binaire à TRUE ou FALSE) : moyen simple indépendant du système d'exploitation qui permet d'exclure un traitement par rapport à un autre. Le seul problème est le partage du flag entre plusieurs tâches ou processus. C'est justement à cela que sert un système d'exploitation : permettre l'utilisation de ressources critiques par plusieurs

applications, sans que ces applications aient de lien entre elles, si ce n'est le système d'exploitation. Le Sémaphore : un sémaphore fonctionne comme un distributeur de jetons ou de tickets (unités). S'il reste un jeton, on passe. Sinon, on attend que quelqu'un ramène un jeton. Le Sémaphore d'exclusion mutuelle : c'est un cas particulier du sémaphore à un seul jeton, (également appelé MUTEX, pour MUTual EXclusion dans certains systèmes d'exploitation, Windows NT, par exemple). La Région a un fonctionnement identique au sémaphore d'exclusion mutuelle, mais avec une nuance qui permet de régler le problème de l'inversion de priorité. On a un exemple de sémaphore suivant :

- **S** : sémaphore d'imprimante;
- **P(S)** prise du sémaphore;
- **V(S)** libération du sémaphore.

Tâche A	**Tâche B**	**Etat du Sémaphore**
		Libre
P(S)		Occupé
Impression	P(S) échoue	Occupé
V(S)		Libre
	Réveil	Occupé
	Impression	Occupé
	V(S)	Libre

On voit dans notre exemple que la tâche **A** demande la ressource critique (prend le sémaphore). Elle imprime son message sous protection de ce sémaphore. Si la tâche **B** demande la ressource pendant que **A** imprime, **B** est mis en attente par le système d'exploitation. Dès que **A** libère le sémaphore, **B** est réveillé et peut à son tour utiliser l'imprimante, sans qu'il y ait conflit entre les deux impressions.

c.3 - Inter blocage

L'inter blocage signifie que deux utilisateurs se sont bloqués eux-mêmes et qu'il n'y a aucune possibilité de les débloquer (dead lock), si ce n'est de modifier le code d'un, voire des deux utilisateurs : ceux ci ont été mis en attente par le système et chacun attend un jeton que

l'autre a pris. Par exemple, deux sémaphores, deux tâches et deux ressources critiques, par exemple un disque dur et une imprimante. Une tâche prend le sémaphore du disque pour lire un fichier, puis le sémaphore de l'imprimante pour imprimer ce fichier. L'autre tâche prend le sémaphore de l'imprimante puis le sémaphore du disque pour écrire le compte-rendu d'impression. Dans la plupart des cas, le fonctionnement sera correct. Toutefois, il va arriver des cas où les deux tâches vont s'attendre mutuellement : une tâche a pris l'imprimante et attend le disque, pendant que l'autre tâche a pris le disque et attend l'imprimante. Il y a croisement des prises de ressources critiques. Le seul moyen pour débloquer la situation est de réinitialiser le système, etc. Dans ce cas simple, il suffit que les deux tâches prennent les ressources critiques dans le même ordre pour qu'il n'y ait plus de problème d'inter blocage.

c.4 - Inversion de priorité

L'inversion de priorité peut se produire quand on a trois tâches A, B et C qui ont des priorités relatives basse pour A, moyenne pour B et haute pour C. On suppose que les tâches A et C accèdent à une ressource critique partagée protégée par un sémaphore d'exclusion mutuelle. Si la tâche A reçoit une unité du sémaphore, la tâche C sera bloquée si elle essaie d'accéder à la ressource. Cette situation est normale : la tâche C ne peut accéder à la ressource prise par la tâche A, tant que celle-ci l'utilise et ne renvoie pas une unité dans le sémaphore d'exclusion mutuelle. L'inversion de priorité survient si la tâche B (de priorité moyenne) devient prête pendant que la tâche A (de priorité basse) est active, et que la tâche C (de priorité haute) est bloquée. La tâche B va préempter la tâche A, et la tâche C va maintenant attendre que la tâche B se déroule, alors que celle-ci n'a rien à faire avec le fait que la tâche C soit bloquée en attente de ressource. Une région peut être utilisée à la place d'un sémaphore d'exclusion mutuelle pour résoudre ce problème. La priorité d'une tâche qui contrôle l'accès à une ressource protégée par une région est augmenté dynamiquement pour égaler la priorité d'une tâche de priorité supérieure qui demande l'accès à la ressource.

c.5 - Échanges entre tâches

Les échanges interprocessus ou entre tâches sont à la base du traitement temps réel de l'information sous forme d'événement en utilisant la mémoire (section globale ou mémoire commune entre tâches) ou en utilisant les mécanismes de messages et de boîtes aux lettres. Dans ce type de mécanisme d'échange d'information, il y a deux façons de faire. Une première qui consiste à passer dans la boîte aux lettres le contenu du message (d'où un problème de taille du message). La seconde qui consiste à passer la référence (en quelque sorte l'adresse)

du message. La deuxième façon de faire est meilleure car elle ne présume pas de la taille des messages véhiculés par la boîte aux lettres.

c.6 - Mécanismes d'empilement –dépilement

Lorsque des événements arrivent sans pouvoir être traités instantanément, il faut toujours un certain temps non nul pour traiter un événement, et il est donc nécessaire de les empiler pour absorber une charge à un instant donné (cas des avalanches, par exemple). Pour ce faire, il y a plusieurs modes de fonctionnement :

- LIFO (Last In Fisrt Out) ou DEPS (Dernier Entré Premier Sorti),
- FIFO (First In First Out) ou PEPS (Premier Entré Premier Sorti), ce qui implique une inversion de la chronologie du traitement des messages,
- Basé sur la priorité.

c.7 - Interruptions matérielles (hardware)

Les interruptions matérielles et leur traitement sont l'essence même des programmes temps réel. Elles permettent d'interrompre le traitement en cours d'une machine informatique pour faire face à un événement extérieur qu'il peut être urgent de traiter en priorité. L'interruption est traitée par un handler d'interruption. Il y a deux possibilités :

1. Le traitement de l'interruption est court et ne nécessite pas d'appel système. Le handler d'interruption peut gérer seul l'interruption.
2. Autrement, et c'est le cas général, le handler d'interruption peut appeler une tâche d'interruption.

Après que l'interruption ait été traitée soit par le handler, soit par le handler plus la tâche d'interruption, le traitement est redonné à la tâche prête la plus prioritaire.

c.7 - Interruptions logicielles (software)

Le fonctionnement est identique à celui des interruptions matérielles. Par contre l'instant d'arrivée de cette interruption est maîtrisé: il s'agit d'un code opératoire spécial du microprocesseur qui déroute le traitement en cours, volontairement placé par le programmeur ou le système d'exploitation. Ce peut être :

1. L'appel système sous DOS ou CPM;
2. Le traitement des exceptions sous iRMX (traitement d'un code erreur lors d'un appel système);

3. Le fonctionnement d'un debugger symbolique (Soft Scope ou SSCOPE par exemple).

c.8 - Traitement des comptes-rendus système

Les comptes-rendus système, ou exceptions, sont des informations retournées par le système d'exploitation à chaque appel d'une primitive système. Par exemple, l'appel système d'allocation dynamique de mémoire ptr = malloc (100) ; // allocation de 100 octets retourne un compte-rendu pour dire si les 100 octets ont pu être alloués. En langage C, le compte rendu se trouve dans la valeur de pointeur ptr retourné : si ptr est différent de zéro, alors les 100 octets ont pu être alloués, et ptr pointe sur cette zone et si ptr est égal à zéro, alors un problème système a provoqué que les 100 octets demandés n'ont pu être alloués (plus de mémoire principale par exemple). Dans ces conditions, l'utilisation de cette mémoire non allouée peut être catastrophique pour la suite du programme. Pour traiter ces comptes-rendus, il y a trois méthodes :

1. Ne pas les tester, ni les traiter. Ceci conduit à un fonctionnement où il faut de temps en temps relancer le système sans savoir pourquoi.
2. Si le système d'exploitation le permet, utiliser un "handler d'exceptions". Ceci conduit à un fonctionnement où il faut de temps en temps relancer le système, en ayant la possibilité de comprendre pourquoi, et éventuellement corriger le problème.
3. Tester les comptes-rendus système en local. Ceci conduit à un fonctionnement où il n'y a pas de relance fréquente du système. (en contre partie, la taille de code d'une application est facilement doublée par rapport à une application qui ne teste pas les comptes-rendus).

Le traitement ou non des comptes-rendus systèmes peut être illustré par l'histoire suivante : un avion arrive près d'un aéroport, et le pilote annonce à la tour de contrôle qu'il ne peut se poser. Pour le contrôleur aérien, il y a au moins deux possibilités : l'avion a effectivement un problème qui fait qu'il ne peut se poser ou le pilote est devenu fou, et ne veut pas se poser. Dans cette histoire, l'avion représente le matériel, le pilote, le système d'exploitation, et le contrôleur aérien, celui qui veut utiliser la machine, c'est à dire une application. Du point de vue d'une application, les informations obtenues du système d'exploitation (informations du pilote de l'avion) doivent permettre de déterminer si c'est le matériel et/ou le système d'exploitation qui sont en cause, suite à un problème de fonctionnement (l'avion qui ne peut pas se poser, ou le pilote qui ne veut pas se poser). Ne pas tester les comptes-rendus système revient à faire une confiance aveugle au système

d'exploitation, même si celui-ci est devenu "fou". Par exemple, un système d'exploitation peut très bien annoncer un périphérique hors service, alors que celui-ci n'a rien. Le test des comptes-rendus système doit permettre de lever ce genre de doute.

c.9 - Contraintes

Les contraintes imposées par un fonctionnement temps réel sont :

1. Les temps de réponse, qui doivent être déterministes et reproductibles;
2. La disponibilité, avec une gestion des priorités tenant compte des impératifs de temps de réponse;
3. La compacité et la rapidité d'exécution du code, pour garantir ces temps de réponse;
4. La portabilité, pour faire face aux évolutions de plus en plus rapides des technologies;

Les contraintes imposées par un fonctionnement temps réel sont :

1. La datation, pour permettre de comprendre le fonctionnement interne;
2. Le diagnostic, pour la mise au point et la recherche de problèmes de fonctionnement;
3. La gestion dynamique de la mémoire, pour pouvoir s'adapter aux différents contextes;
4. La conception, qui doit prendre en compte les contraintes évoquées ici;
5. Les marches dégradées, pour répondre aux exigences sécuritaires.

IX.6 – Implémentation iRMX

Le système d'exploitation iRMX (iNTEL Real time Multitasking eXecutive) est un système d'exploitation temps réel préemptif multitâches qui tourne sur des machines physiques construites autour de processeurs de la famille xx86. On trouve des applications bâties autour de ce système d'exploitation dans des enregistreurs magnétiques dans les satellites, des centraux de commutation téléphoniques, la distribution électrique et le nucléaire et les péages d'autoroutes et de parking. iRMX est organisé en couches logicielles : le Nucleus ou noyau : contrôle toutes les ressources du système. C'est lui qui procure les fonctions temps réel : le multitâches, la gestion du temps, des priorités et des interruptions et la gestion de la mémoire et des objets (tâches, sémaphores, boîtes aux lettres, etc.). Le noyau d'iRMX possède un sous-ensemble appelé iRMX. Ce sous-ensemble iRMX du noyau d'iRMX est intéressant car on le retrouve à la base d'extensions temps réel proposées pour Windows NT. Le BIOS (Basic Input Output System) : il offre des primitives systèmes pour lire et écrire les périphériques. Ces primitives supportent le synchrone et l'asynchrone. L'intérêt de l'asynchrone (on n'attend pas que la requête système soit terminée pour passer à une autre

activité) par rapport au synchrone (on attend que la requête système soit terminée pour passer à une autre activité) est que ce mode de fonctionnement permet de profiter des temps morts dus à la mécanique comparés aux temps mis en jeu par l'informatique ou l'électronique :

Mécanique	Informatique, Electronique
Disque dur, disquette, etc.	Microprocesseur
CD, imprimante, etc.	Microcontrôleur
la seconde, la milli seconde, etc.	la micro ou la nano seconde, etc.

Le BIOS fonctionne avec des pilotes de périphériques (device driver). L'utilisateur peut ajouter ses propres drivers. L'EIOS (Extended Input Output System) : il offre des primitives systèmes similaires au BIOS pour lire et écrire les périphériques, mais plus simples d'emploi. L'adressage est logique plutôt que physique. L'EIOS ne supporte que le synchrone. L'Application Loader : il permet de charger des exécutables à partir du disque, sous contrôle opérateur ou non.

L'UDI (Universal Developement Interface) : c'est une interface universelle de développement. Elle offre une interface avec l'OS simple et standard, ce qui permet de rendre les applications portables sur des machines supportant l'UDI. Cette couche a été développée également pour DOS, VAX/VMS et UNIX system V.

Le HI (Human Interface) : c'est l'interface utilisateur. C'est le langage de commandes prévu pour un ou plusieurs opérateurs simultanés pour dialoguer avec la machine virtuelle iRMX. Il est constitué d'un ensemble d'appels système, de commandes et d'un CLI, ou Command Line Interpreter.

IX.7 – Références

1. Principes des Systèmes d'exploitation des ordinateurs. S. Krakowiak. Editions Dunod Informatique.
2. Approche du temps réel industriel. J.M. De Geeter. Editions Ellipses.
3. Forth pour micros. J.M. De Geeter. Editions Eyrolles.
4. Les bases de l'informatique industrielle. J. Leray. Editions Dunod.
5. Principes des Systèmes d'exploitation. A. Siberschatz, P.B. Galvin. Editions Addison Wesley France.
6. Les réseaux locaux industriels. F. Lepage. Editions Hermés.

7. UNIX programmation avancée. M.J. Rochkind. Editions Masson.
8. Real Time and Systems Programming for PCs. C. Vickery. Editions Mc Graw Hill.
9. C A Reference Manual. S. P. Harbisson, G. L. Steele. Editions Prentice Hall. Tartan Laboratories.
10. VMS System Services Reference Manual. Digital Equipment Corporation.
11. iRMX System Call Reference. Intel/Radisys.
12. iRMX Programming Techniques and examples. Intel/Radisys.
13. Windows NT 4, la base de registres. R. Tidrow. Simon & Schuster Macmillan.
14. Développer sous Windows 95 et Windows NT 4.0. J. Richter. Microsoft Press.
15. Bible du PC.
16. INtime Software RadiSys Corporation : user's guide.
17. Microsoft Embedded Review.