Chapitre 1. Introduction

I. Généralités :

1/ L'ordinateur « déconnecté » (hors ligne - off line)

Dans une application classique de calcul, le programmeur écrit son code (programme) qui est ensuite chargé et exécuté jusqu'à complétion. A ce moment, l'utilisateur a à sa disposition le résultat du traitement. Durant l'exécution du programme le calculateur est « autosuffisant ». Les instructions sont exécutées séquentiellement. C'est le résultat final qui compte. Le calculateur peut être considéré comme outil d'aide à la décision.

2/ L'ordinateur « connecté » (en ligne - on line)

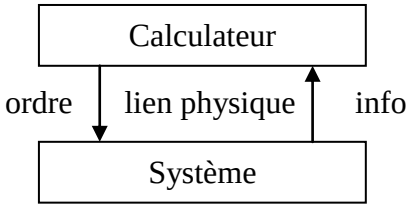
Dans d'autres cas, le calculateur n'est plus un outil de calcul ou d'aide à la décision, mais c'est lui même qui décide, dirige, gère un système extérieur. Pour cela, il faut qu'il puisse envoyer des ordres et recevoir des informations de l'extérieur d'une façon continue et en temps voulu.

Le programmeur écrit un code où il y a des instructions qui permettent l'échange des informations, la communication avec l'extérieur. Ainsi, par exemple, il y a des instructions de temporisation, des instructions qui prennent en charge des informations venues de l'extérieur.

On dira que le calculateur fonctionne en temps réel en particulier si les contraintes temporelles existent. Dans ce cas, si le calculateur ne fait pas un calcul, n'envoie pas un ordre dans un temps déterminé **T**, ce calcul, cet ordre n'ont plus de valeurs.

De cette remarque découle la définition classique d'un calculateur temps réel.

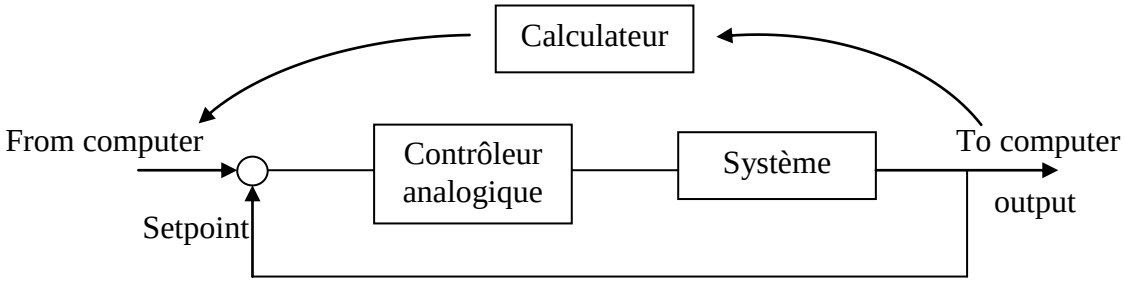
Un ordinateur temps réel doit répondre en un temps limité, fixé à l'avance à un signal extérieur, passé ce temps limite, la réponse n'a plus de valeur. De plus, il existe un lien physique entre le calculateur et le système pour assurer la communication.



3/ Historique

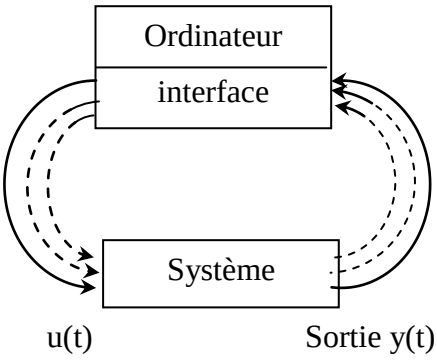
Historiquement, l'ordinateur était un moyen de calcul. Les premières applications qui peuvent être considérées temps réel ont été réalisées dans l'aviation américaine vers la fin 40', étant protégé par le secret militaire, elles ne sont pas bien documentées. Par contre, la première application de l'ordinateur à temps réel dans l'industrie civile est très bien documentée. Elle a été réalisée dans la pétrochimie dans l'usine TEXACO à Port-Arthur.

Les travaux ont commencé en 1956 pour se terminer en 1959. Dans cette application, le rôle de l'ordinateur consistait à calculer les consignes (points de fonctionnement) ; à superviser les grandeurs et a enregistrer les données. Dans ce cas, l'ordinateur n'intervient pas dans la boucle de commande (il ne calcule pas la commande).



Ce n'est que vers le début des années 60' qu'a été réalisée la première application temps réel proprement dit où l'ordinateur intervient directement dans la boucle de commande (il délivre le signal de commande) Direct Digital Control « DDC ». Ceci a été réalisé grâce aux travaux théoriques de Regazzini et son équipe Jury, Zadeh, Kalman en USA, et de Tsyprin (URSS). Cette application a été réalisée en Angleterre dans une usine de ICI (Imperial Chemical Industries). L'ordinateur dans ce cas intervient directement dans une chaîne de boucle de commande, en plus des attributs de l'application de TEXACO.

A partir des années 70' et le boom des micro-processeurs, les applications, se sont généralisées et sont devenues plus complexes. Un système temps réel est devenu un ensemble complexe et cohérent de matériel et de logiciel (Embedded System).



II. Les tâches de l'ordinateur temps réel :

Aujourd'hui, dans l'industrie, les calculateurs effectuent plusieurs taches à la fois, ceci n'implique pas automatiquement que le même calculateur effectue toutes les tâches. Cependant, on peut classifier ces tâches en quatre groupes :

1/ La supervision

Dans ce groupe, on regroupe toutes les tâches où le calculateur n'intervient pas directement sur le système, entre autre :

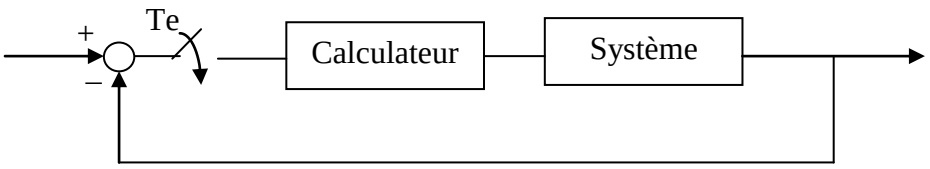
- Accumulation et enregistrement des données ;
- Calcul des consignes ;
- Génération des alarmes.

Les premières applications ont été réalisées en supervision, ces tâches sont du point de vus sécurité parfaitement fiables. Du point de vue SOFT et HARD les exigences ne sont pas très sévères.

2/ Commande en boucle fermée :

On rassemble ici les tâches où le calculateur génère le signal de commande. Il joue donc le rôle de contrôleur en exécutant un algorithme de commande. Ainsi, le calculateur entre dans la boucle fermée et sera donc inclus dans le calcul des caractéristiques de la boucle : stabilité ; précision ; …etc.

Ici, la présence du calculateur va influer directement sur le système et va nécessiter des précautions particulières ; les exigences SOFT-HARD sont plus sévères.



3/ Les automatismes à séquences :

Un automatisme à séquence est l'automatisation d'une suite d'actions coordonnées et s'effectuent les unes après les autres. Cette automatisation peut être réalisée à l'aide d'un câblage « Structure figée ».

On peut aussi considérer une réalisation SOFT par programmation. Ce programme peut être intégré parmi les tâches du calculateur.

Il existe plusieurs exemples d'actions enchaînées dans l'industrie par exemple le démarrage des grands ensembles industriels.

4/ Interface homme – machine :

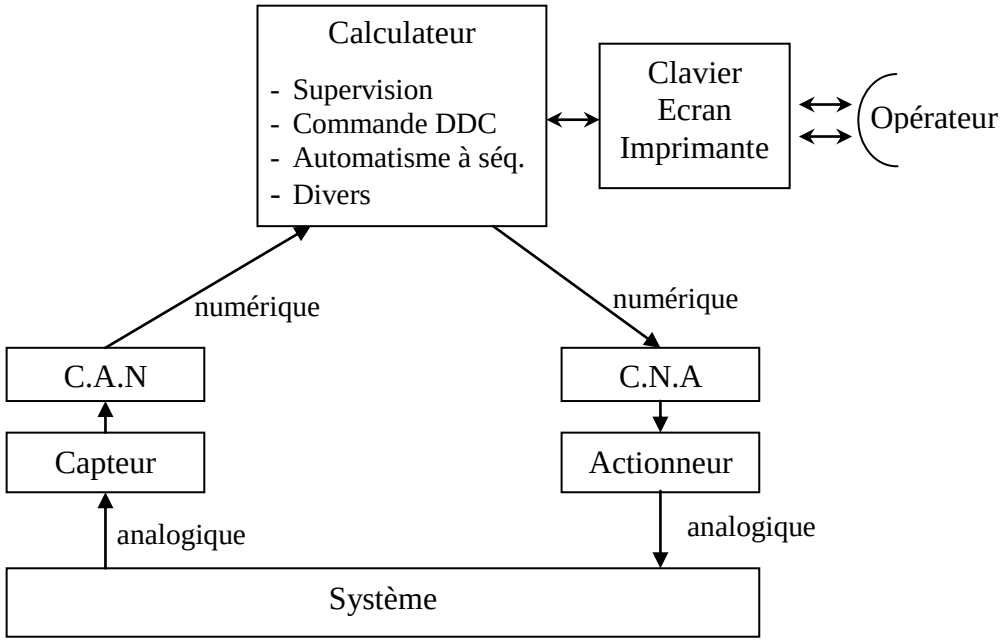
Pour assurer la communication entre l'opérateur et le système, on dispose de l'écran, du clavier et de l'imprimante.

Il s'agit de permettre à l'opérateur d'analyser des données et d'agir sur le système : gérer les alarmes, changer les consignes, changer les paramètres des algorithmes de commande.

III. Structure d'une application temps réel :

Fondamentalement, une application temps réel est caractérisée par les moyens de communication entre le calculateur et le système qu'il gère.

Du point de vue HARD, le moyen de communication est caractérisé par une interface physique constitué principalement des convertisseurs A/N et N/A.



Du point de vue SOFT, l'interface virtuelle constituée d'un programme qui fait fonctionner l'ensemble programme temps réel.

Par rapport à un programme séquentiel, un PTR est caractérisé par :

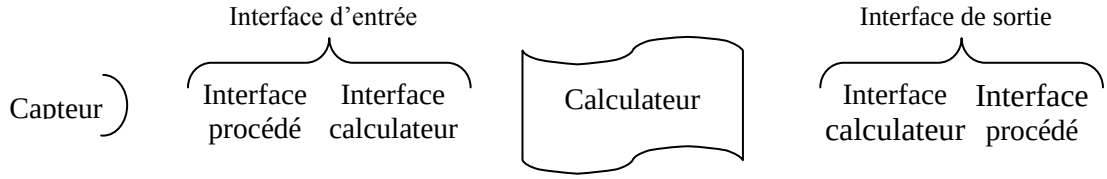
- Sa capacité à réagir à des signaux extérieurs en un temps bien défini. Il faut donc tenir compte du type de matériel utilisé.
- Sa division en tâches qui communiquent entre elles, il faut donc assurer la synchronisation entre ces tâches.
- Il faut assurer la gestion des données communes que les tâches partagent.

Chapitre 2. Les techniques d'interface

I. Introduction :

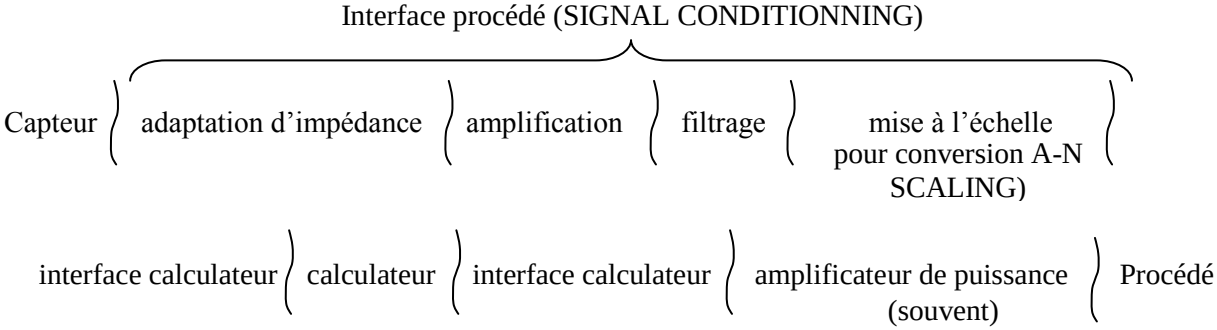
Dans ce chapitre, nous allons introduire les différents traitements que subit le signal (l'information) délivré par le capteur avant son interprétation par le calculateur. Ces différents traitements sont réalisés dans l'interface entre capteur et calculateur.

De plus, on s'intéresse aussi au traitement que subit l'information à sa sortie du calculateur avant son utilisation.



II. Interface procédé

Sous le groupe interface procédé, on rassemble les traitements purement analogiques.



Avant de développer ces différents traitements, nous allons rappeler brièvement les caractéristiques des capteurs et des actionneurs.

1/ Caractéristiques des capteurs (SENSOR) :

En générale, les capteurs sont classifiés en fonction du type du signal délivré à la sortie.

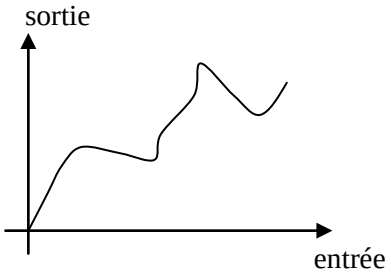
- Les capteurs analogiques : délivre un signal électrique analogique.
- Les capteurs binaires : délivre une information binaire.
- Les capteurs On-Off : Tout ou Rien.

Les capteurs analogiques sont les plus fréquents.

Un capteur est caractérisé par son comportement statique et dynamique.

a. Caractéristique statique

Nous donne la relation entrée-sortie statique c à d en régime permanent pour plusieurs valeurs de l'entrée. La caractéristique statique peut être linéaire ou non linéaire, mais généralement non linéaire globalement, elle est donnée par le constructeur sous forme de tableau ou sous forme de courbe, sinon, l'utilisateur doit relever la caractéristique lui-même.



De plus, un capteur est caractérisé par :

- Sa précision
- Sa résolution
- La dérive
- Domaine de fonctionnement.

b. Caractéristique dynamique

Elle permet d'analyser le régime transitoire du capteur relativement ou régime transitoire de l'ensemble du système. Donc, il s'agit de comparer les paramètres de la réponse unitaire (temps de montée, temps de réponse, 1er dépassement) du capteur et du système ; faut il ou non négliger les paramètres de la réponse du capteur.

2/ Actionneurs

Les actionneurs permettent d'agir sur les systèmes; recevant une information du calculateur, ils la transforment en action. Dans le cas du tout électrique, les actionneurs sont des moteurs. On utilise quatre types de moteurs :

Moteurs pas à pas : à chaque impulsion, il tourne par un angle θ donné par le constructeur. Il ne nécessite pas de boucle d'asservissement. Il ne fonctionne que pour les faibles puissances (ou très faible) (table traçants, petit robot).

Les moteurs à courant continue : boucle d'asservissement, de position et de vitesse facile à réaliser. Les MCC (DCM) sont de construction complexe et donc peu robuste ; ils sont réservé pour les faibles et moyen puissances.

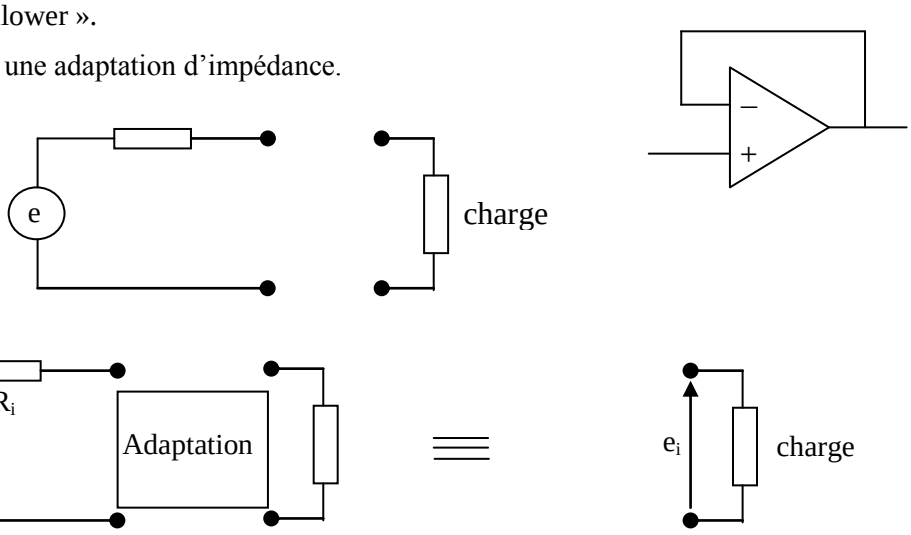
Moteurs à courant alternatif : asynchrone ou synchrone : de construction simple, principe de fonctionnement simple. Asservissement difficile à réaliser, cependant, ils tendent à remplacer les DC. Moyens et fortes puissances.

3/ Conditionnement des signaux :

Le conditionnement du signal (Signal conditioning) regroupe tous les traitements analogiques que subit le signal avant sa conversion A/N (et après conversion N/A).

Le signal issu du capteur doit être isolé de la chaîne de conditionnement pour éviter les effets de charge et protéger la chaîne de mesure. L'isolation peut être réalisée à l'aide d'un ampli opérationnel en « Voltage follower ».

On réalise aussi une adaptation d'impédance.



Ce montage permet d'éliminer les effets de charge. Si en plus, on désire une isolation plus efficace contre des tensions élevées, on peut utiliser un couplage optoélectronique.

Une fois l'isolation réalisée, les traitements les plus souvent réalisées consistent en :

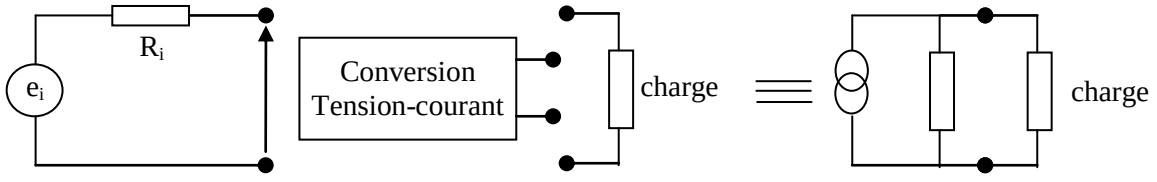
- Filtrage analogique.
- Amplification, atténuation.
- Conversion courant-tension ou tension-courant.
- Linéarisation.

a. Conversion tension-courant et courant-tension :

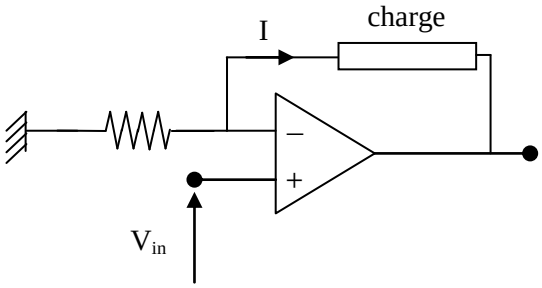
Le signal issu du capteur (après son amplification) est transmis à la chaîne de mesure (carte d'acquisition située en salle de contrôle). Le support de transmission peut être la tension ou le courant.

Lorsqu'on utilise la tension, le signal est directement envoyé sur câble. Cependant, les chutes de tension le long des conducteurs affaiblissent le signal.

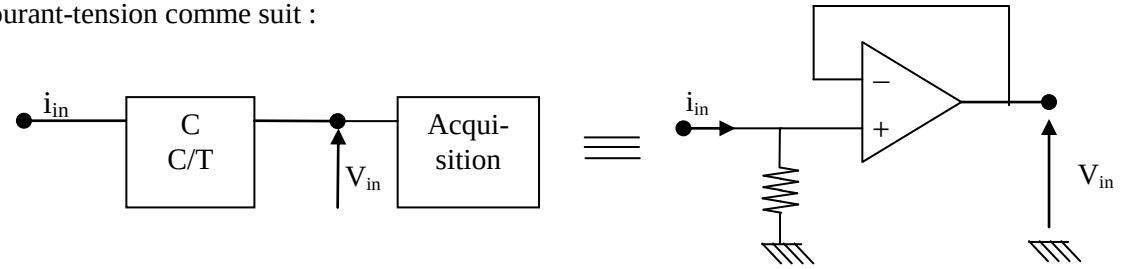
Pour éviter ces affaiblissements, on utilise une transmission en courant. Pour cela, il faut réaliser les conversions nécessaires comme suit :



capteur



Au niveau de la réception, (entrée d'acquisition) il faut réaliser l'opération inverse : conversion courant-tension comme suit :



Les valeurs des signaux de transmission sont standardisées

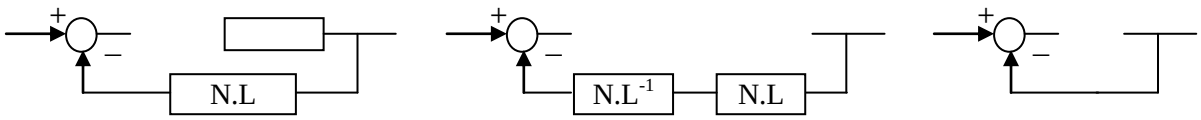
Tension : (0 , 5V) (– 10V , +10V) (0V , +10V) (1V , 5V)

Courant : 4mA , 20mA

b. Linéarisation :

Parfois, la caractéristique statique su capteur est fortement non linéaire et cela influe négativement sur le comportement du système. En particulier, si on a une boucle fermée, les effets d'une non linéarité dans le feedback peuvent provoquer une détérioration des performances (jusqu'à l'instabilité).

Pour cela, il faut compenser la non linéarité.



Cette compensation peut être matérielle ou logicielle.

c. Filtrage

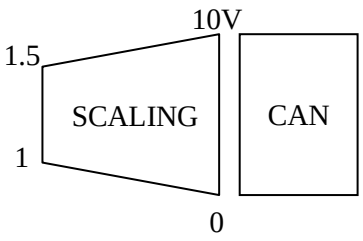
Les signaux qui parviennent à la salle de contrôle sont bruités, il faut donc éliminer le bruit qui s'ajoute au signal utile à l'aide d'un filtre passe-bas. Il faut donc calculer la fréquence de coupure. Ainsi, dans le cas d'une régulation, le signal issu des capteurs varie très peu, le signal de bruit le plus bas et le signal du secteur (50 Hz), on peut alors choisir f_c = 40 Hz.

d. Mise à l'échelle :

La mise à l'échelle consiste à adapter la tension à convertir au convertisseur A-N. cela revient à utiliser pleinement les potentialités de CAN.

Par exemple : CAN : 0 – 10 v - 16 bits.

Tension issue du capteur : 1 v – 1.5 v



III. L'interface calculateur :

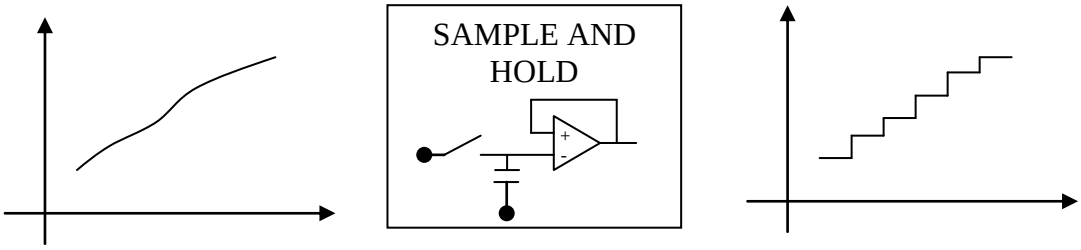
Nous avons vu dans le paragraphe précédent les traitements du signal analogique que nous avons intitulé 'interface procédé'. Dans cette section, nous allons considérer le traitement de conversion du signal ainsi que l'analyse de ce traitement. De plus, nous nous intéresserons au traitement à l'intérieur du calculateur avant son utilisation finale.

a. Echantillonnage :

Pour réaliser la conversion A/N, il faut discrétiser le signal analogique. Ce processus s'appelle « échantillonnage ».

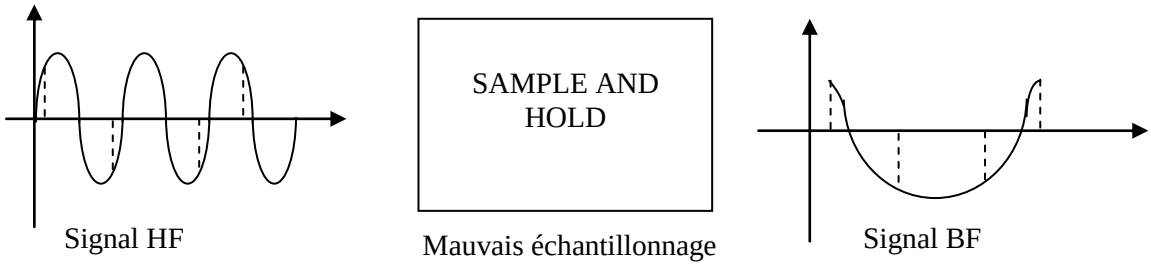
Nous savons que la période d'échantillonnage doit être supérieure au temps de conversion du convertisseur A-N et satisfait le théorème de Shanon.

Pendant la durée de conversion, le signal est maintenu constant à l'entrée du CAN, cette opération s'appelle blocage du signal ou blocage d'ordre zéro.



En général, la plupart des CAN intègre l'opération d'échantillonnage blocage (Sample and Hold).

Le processus d'échantillonnage peut produire un phénomène d'aliaisage si la période d'échantillonnage est mal choisie.



Signal HF — Mauvais échantillonnage — Signal BF

Pour éliminer le phénomène d'aliaisage, on détermine d'abord la fréquence utile la plus élevée f_c, ensuite, on calcule la fréquence d'échantillonnage $f_e > 2\ f_c$. Enfin, on place un filtre passe bas de fréquence de coupure $f_N = f_e/2$ (f_N : fréquence de Nyquist) filtre anti-aliaisage.

a. La conversion A/N :

La conversion analogique numérique consiste à transformer un signal analogique en un signal numérique en base 2. L'entrée analogique du convertisseur est à travers d'une broche, il y a autant de sortie que de bits représentatifs.

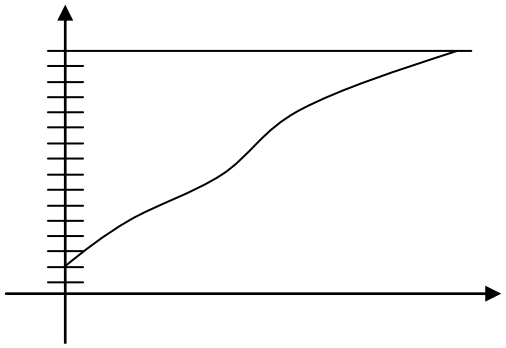
Le convertisseur à N bits quantifie (discrétise) un signal analogique en 2^N valeurs.

Un CAN est caractérisé par sa précision et son temps de conversion. Les convertisseurs sont classés suivant le mode de fonctionnement.

La conversion peut être interprétée comme un processus de quantification qui produit des erreurs dont les effets peuvent être non négligeables.

Pour illustrer ces effets, considérons, par exemple un convertisseur 4 bits et un signal analogique variant entre 0 et 4 volts. Ce signal sera quantifie en 16 valeurs de pas 4/16 = 0.25.

Notons δ le pas de quantification, alors, l'erreur de quantification varie entre $[-\delta/2, +\delta/2]$.



Pour éliminer erreurs de quantification, on peut utiliser une approche déterministe non linéaire qui consiste à interpréter le processus de quantification comme un élément non linéaire dans la boucle et de compenser la non linéarité ou une approche stochastique qui consiste à considérer les erreurs de quantification comme des sources de bruit blanc de moyenne nulle.

c. Traitement à l'intérieur du calculateur :

Une fois le signal est converti, il est introduit dans le calculateur où il subit un certain nombre de traitements préalables entre autre :

Filtrage numérique.

Validation de la mesure.

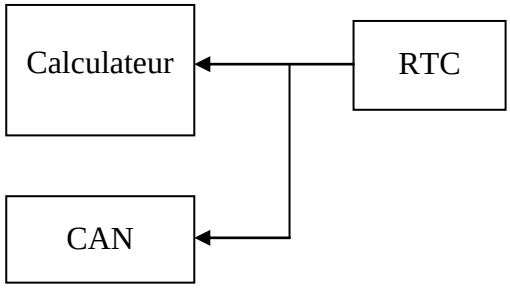
Transformation de l'information numérique en données physiques (engineering units).

Calcul de la moyenne.

d. L'horloge temps réel (Real Time Clock RTC)

La RTC est un générateur d'impulsions qui cadence une application temps réel. Elle est extérieure aux moyens de calcul.

Les impulsions sont prises en compte par des interruptions.



Chapitre 3. Les techniques de spécification des systèmes temps réel

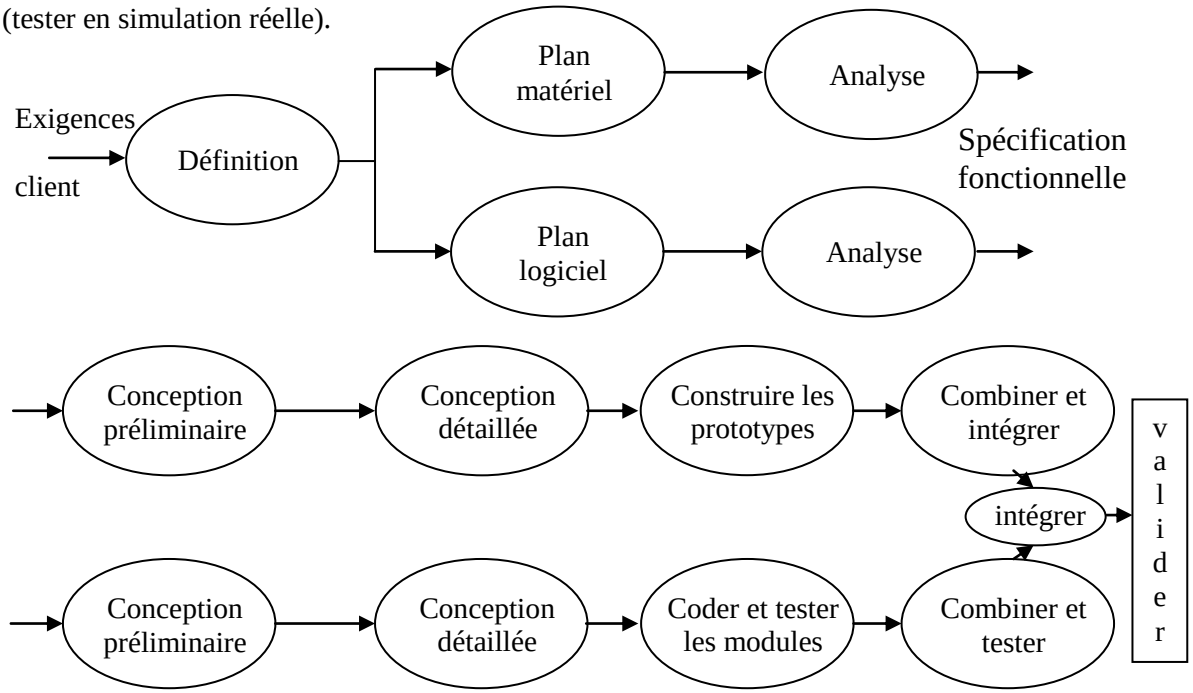
I. Introduction :

Le développement des systèmes temps réel est une tache complexe, impliquant des choix matériels (ou construction matérielle) et développement et choix logiciels. Ceci nécessite une approche disciplinée qui permet de minimiser les erreurs, les retards et obtenir une bonne fiabilité en fonctionnement.

D'une façon générale, une approche disciplinée comporte deux phases :

Phase de planification : à partir des exigences clients (les cahiers des charges). Etablir un plan des travaux à réaliser.

Phase de développement : développer les produits matériels et logiciels, les intégrer et les valider (tester en simulation réelle).



II. Approches pour la conception des logiciels temps réel :

Nous avons vu que la conception des systèmes temps réel passe par deux phases ; la phase de planification ensuite le développement. Nous avons étudié la phase de planification sur un exemple.

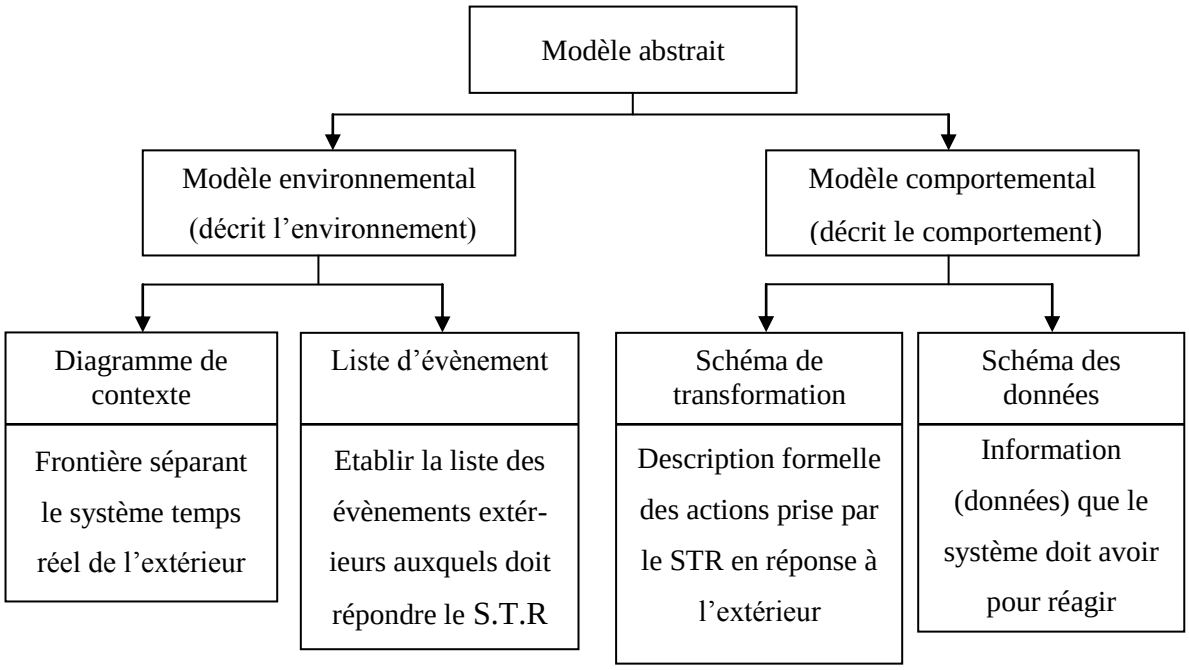
La phase de développement suit cette première phase, son but est de produire un logiciel intégré.

Il y a plusieurs approches pour réaliser le développement :

- L'approche basée sur le langage courant, c'est une approche non formelle. On décrit le développement en langage courant ou au moins la phase préliminaire.
- L'approche semi-formelle : le développement du logiciel se fait en utilisant une discipline selon un formalisme particulier. Dans ces méthodes, le formalisme concerne la phase préliminaire. Il y a plusieurs approches semi-formelles ; parmi les quelles on peut citer :
  - Méthode baser sur les réseaux de Pétri.
  - MASCOT (Modular Approach for Software Construction Operation and Test).
  - SADT : Structured Analysis and Design Technique.
  - SA/SD : Structured Analysis / Structured Design (Ward and Mellor 1986).

1/ SA/SD de Ward et Mellor

Le but de la méthode est de construire une modèle abstrait du logiciel qui permet de discipliner le développement, de faciliter la communication entre les développeurs et d'unifier les modes de représentation des informations.



a. La formulation de la méthode de Ward et Mellor :

Les représentaions graphiques sont des abstractions d'objets informatiques, instructions, fichiers, pile, signal,...

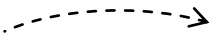
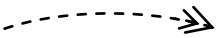
i) Notation

Transformation : 2 types de transformations sont définies :

- La transformation de données : représentée par 
  - Transforme des données d'entrées variables en données de sortie variables. ″abstraction d'un traitement continue de données″
  - Modifie les données stockées. « abstraction d'instructions »

- La transformation de contrôle : représentée par 
  - Commande la transformation de donnée (déclenche)

Les transformations sont reliées entre elles, avec l'extérieur et la mémoire à l'aide de *flot de données* et de *flot d'évènements*.

- Les flots de données : représentés par des arcs pleins, on distingue deux types :
  - Flot de données continus (arrivent continûment) 
  - Flot de données discrets (arrivent occasionnellement) 
- Les flots d'évènements : représentés par des arcs en pointillé, on distingue deux types :
  - Le signal 
    - Le signal montre l'intention de l'expéditeur de signal que quelque chose est arrivée.
  - Activation/Désactivation (Enable/Disable) 
    - Enable : l'expéditeur montre son intention pour que le récepteur produise une sortie
    - Disable : ″ ″ ″ ″ ″ ″ ″ ″ une entrée.

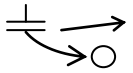
Le réservoir de données : représentés par  deux traits parallèle. Abstraction de fichier.

Buffer :  abstraction d'une pile.

ii) Les règles :

Entrée/Sortie : activation d'une transformation de données : c'est un flot de données discret issu d'une autre transformation, de l'extérieur, d'un buffer, ou d'un signal.

- **Règle 1** : Une transformation de données ne peut avoir qu'une entrée active au plus et aucune ou plusieurs sorties actives.
- **Règle 2** : Les entrées/sorties d'une transformation de contrôle ne peut être que des flots d'évènements (signal ; Enable/Disable).
- **Règle 3** : sur les flots

- Les flots doivent être connectés au moins à 1 bout (T. données, T contrôle, réservoir, buffer) ; s'il y'a un bout non connecté : information dirigée vers l'environnement (action, écran, imprimante,…)
- Flots de données discrets : obligatoirement connecté à T. donnée, l'autre bout peut venir de l'extérieur, réservoir, buffer, T. D.
- Réservoir de données :  fichier de donnée, il donne un flot discret
- Buffer est relié à T. D.

Exercice :

On désire réaliser une commande de température T dans un réacteur chimique. On manipule à vanne A pour que la température reste constante. La constante de temps du procédé est supposée égale à 2 secondes dans le domaine d'utilisation. La consigne peut varier entre 4 et 7. L'algorithme de commande est un PID. L'affichage de T, des paramètres PID, de la consigne et de l'erreur est désiré.
L'affichage de l'état et des vannes serait intéressant (mais pas nécessaire). L'opérateur peut introduire à tout moment : consigne + paramètres PID.
Construire une application temps réel pour ce système :

a) Proposer une structure matériel.
b) Etablir un modèle abstrait du logiciel.

Solution

_ Dimensionner et choisir les vannes.
_ Choisir le capteur de T : Déterminer sa tension de sortie et calculer V = *f* (T)
_ Déterminer la période d'échantillonnage T_e. exp (T_e= 0.1s)
_ Déterminer la fréquence de RTC : $T_{RTC} < T_e$. T_{RTC} = 2s.
_ C.A.N 8 bits ; temps de conversion lent suffit (ms)
_ Nombre de voies analogiques 1 (T)
_ et choisir 1 seule carte à 1 voies.
_ Isolation : adaptation d'impédance : ampli opérationnel.
_ Calculer la tension d'offset et la tension de mise à l'échelle pour connexion au CAN.
_ Déterminer les étages de puissances.

<u>Moyen de calculs</u>

_ Microprocesseur > 8 bits et Fq > 3 MHZ.
_ Taille mémoire RAM et ROM.
_ Avec coprocesseur.

<u>Structure</u>

_ 1 microcontrôleur.

_ 1 micro-ordinateur.

_ 1 back plane bus (bus de fond de panier)

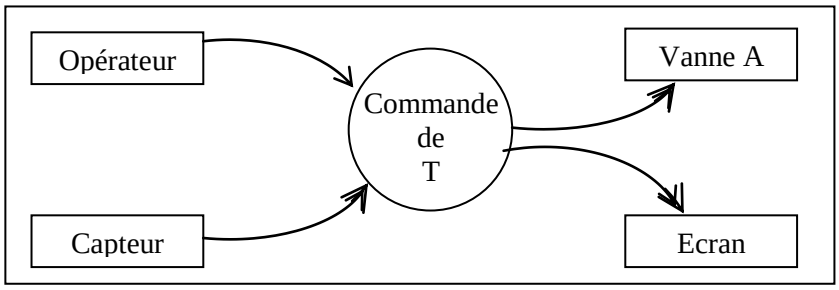
Partie logicielle

Modèle abstrait :

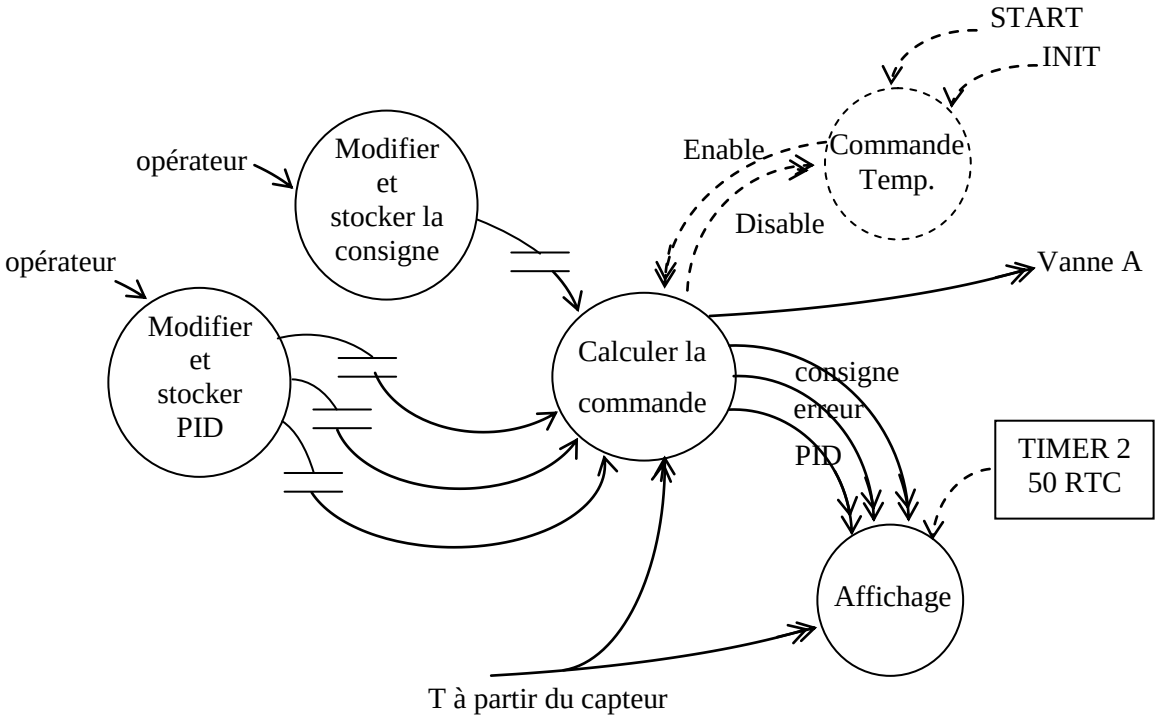
_ Liste des évènements

- RTC
- Signal vanne V_A.
- Signal vanne V_B.
- Signal milieu.

_ Diagramme de contexte



_ Diagramme de comportement schéma de transformation



Chapitre 4. La Programmation Concurrente

I. Introduction

La programmation concurrente (ou parallèle) est une discipline de l'informatique dont le but est de proposer des solutions aux problèmes multitâches. C'est-à-dire aux problèmes qui se posent lorsque plusieurs tâches se concurrencent pour une ou plusieurs ressources communes**.** Elle propose des objets informatiques qui peuvent être réalisés ou implantés dans des systèmes d'exploitation ou des langages de programmation**.**

Trois problèmes fondamentaux ont été identifiés dans la programmation concurrente :

_ L'exclusion mutuelle.

_ La synchronisation.

_ La communication.

Plusieurs objets informatiques théoriques ont été proposés pour la résolution de tels problèmes.

Dans ce chapitre, nous allons adopter une approche classique qui consiste à étudier ces problèmes l'un après l'autre et voir les solutions proposées.

II. L'exclusion mutuelle :

Le problème de l'exclusion mutuelle se pose lorsqu'une ou plusieurs tâches désirent utiliser la même ressource, qu'on appellera « ressource critique ».

On appelle « section critique » d'une tâche la partie de programme où la tâche utilise la ressource critique.

La section critique doit satisfaire les conditions suivantes :

1/ une seule tâche et une seul peut entrer dans la section critique (à un moment donné).

2/ Quand une tâche veut entrer dans la section critique, elle doit le faire au bout d'un temps fini.

3/ Une tâche doit entrer dans la section critique pour un temps fini.

Tout objet informatique qui se propose de résoudre l'exclusion mutuelle doit donc satisfaire ces trois conditions.

La structure de base d'un tel objet est la suivante :

```
Begin
        Préprotocole
        Section critique
        Postprotocole
End
```

Le préprotocole est l'entrée de la section critique. Le postprotocole est la sortie.

L'exclusion mutuelle peut en théorie être résolue en utilisant :

_ Des variables logiques.

_ Les sémaphores.

a) Exemple d'utilisation des variables logiques :

On définit une variable logique qui indique l'état de la ressource

Ressource_occupée : logical.

Ressource_occupée = False.

```
Begin
Test    While Ressource_occupée
        End                                  } Préprotocole
Set     Ressource_occupée = True

        Section critique

        Ressource_occupée = False            } Postprotocole
End
```

Vérification des conditions :

Considérons le cas où une interruption hard peut arrêter l'exécution de l'objet informatique proposé, alors, dans le cas d'un ordonnancement préemptif, la tâche qui exécute le préprotocole peut être abandonnée et une autre tâche prend la main, qui peut vouloir utiliser la ressource critique et qui va entrer dans la section critique car Ressource_occupée = False.

Une solution pour éviter ce problème est de masquer les interruptions ou de grouper ensemble les opérations de Test et Set (Sans interruptions).

b) Les sémaphores :

Le sémaphore est un objet informatique composé d'une variable entière s et d'une liste d'attente q(s) : le sémaphore est utilisé à sa première invocation (appelle) .

On associe à un sémaphore deux primitives : Secure (s) et Release (s) définit comme suit :

```
Secure (s)
        Begin
        s = s – 1
        If s < 0 then
          Tâche exécutante bloquée
                    et placée dans q(s)
        End
        End
```

```
Release (s)
        Begin
        s = s + 1
        If s ≤ 0 then
          Prend une tâche de q(s) et l'exécute.

        End
        End
```

Ici, s est un entier initialisé à 1 pour l'exclusion mutuelle.

Structure pour l'exclusion mutuelle :

Secure (s) } préprotocole.

Use ressource } section critique.

Release (s) } postprotocole.

III. La synchronisation :

La synchronisation exprime une relation d'ordre entre différentes tâches cette situation est très courante dans les systèmes temps réel. Une tâche ne peut être exécuté qu'après exécution d'une ou plusieurs autres tâches.

Le cas le plus simple de synchronisation est la relation d'ordre qui existe entre la tâche « d 'échantillonnage « et la tâche calcul de la commande.

Plusieurs objets informatiques sont été développés pour assurer la synchronisation, les plus utilisés sont :

_ L'évènement.

_ Sémaphore.

a) **l'évènement** est un objet informatique composé d'une variable binaire e = (0, 1) et d'une liste d'attente au quel en associe deux primitives signal (e) ; wait (e) définis comme suit :

signal (e) : e = 1

wait (e) si e = 0 la tâche invoquant est placé en q(e).

si e = 1 " " " état prêt.

Remarque :

Les primitives ne manipulent pas le signal.

Des problèmes de perte de signal peuvent apparaître en particulier si signal (e) est exécuté avant wait (e) lorsque la durée de e = 1 est très courte (signal furtif).

Pour éviter cette situation, deux solutions peuvent être envisagées

_ événement mémoriser ; la valeur de e est placé en mémoire.

_ événement retardé (solution hard) : le signal est maintenu pendant un certain moment Δt fixe ou user defined (signal (e, Δt))

b) **Le sémaphore "privé"** : On peut utiliser le sémaphore pour assurer la synchronisation, on définit alors le sémaphore « privé » associe à une tâche A qui elle seul peut exécuter Secure (s), l'autre tâche exécute Release (s).

Exemple :

On a la relation d'ordre T_1 T_2, comment satisfaire cette relation avec un sémaphore à 0 :

T_1 Release (s)

T_2 Secure (s)

IV. Communication :

Nous considérons le cas où des tâches doivent se communiquer des données. Deux objets informatiques peuvent être utilisés :

_ Les boites aux lettres.

_ Les rendez-vous.

Les boites aux lettres : ce sont des fichiers qui disparaissent à la mise hors tension. Deux primitives sont associées :

Put-mail-box (N°, message) (le numéro indique la zone dans la mémoire).

Get-mail-box (N°, message).

Le rendez-vous : le rendez-vous est une méthode de communication entre tâche : pour chaque tâche, on définit un point de rendez-vous, lorsqu'une tâche atteint ce point, elle doit attendre que toute les tâches aient atteint ce point.

Exemple : Réalisation ADA :

task A task B

B exporte (y) accept exporte(y) do (condition d'attendre)

Nous avons considéré l'exclusion mutuelle, la synchronisation et la communication, les problèmes sont relativement similaires.

Ainsi, un objet informatique qui permet de résoudre un problème peut être réarrangé pour résoudre un autre problème.

Les objets informatiques sont des entités théoriques. La question reste à savoir s'ils sont implantés dans des langages ou des systèmes d'exploitation.

Exemple :

- ADA : Rendez-vous.
- Quelques versions de concurrent pascal : SEMAPHORE, EVENEMENT.
- VAX/VMS : Mail Box.
- Certaine version d'UNIX : SEMAPHORE.

Exercice : Producteur-Consommateur

On dispose d'un espace mémoire sous forme de Buffer circulaire de N positions.

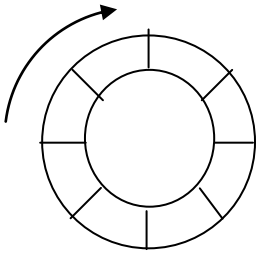
Un programme producteur place les données dans le Buffer.

Un programme consommateur retire les données du Buffer.

Buffer vide : le consommateur ne peut pas retirer.

Buffer plein : le producteur ne peut pas placer.

Les deux programmes ne peuvent pas accéder en même temps au Buffer.



Ecrire les programmes : producteur et consommateur.

On définie Nmax : capacité du Buffer

N : position déjà occupée.

Place (x)

Consomme (x)

On a un problème d'exclusion mutuelle (accès à un Buffer)

Un problème de communication/synchronisation.

<u>Exclusion mutuelle :</u>

Sémaphore : Accès Buffer.

<u>Communication/synchronisation :</u>

Evènement : Etat change.

Programme

Accès Buffer : semaphore
Etat change : evenement
Nmax, N: integer
N = 0

Initialisation

Producer
Begin
Secure (accès Buffer)
If (N = Nmax) then
Release (accès Buffer)
Wait (état change)
Secure (accès Buffer)
End

Place (x)

N = N+1
Signal (Etat change)
Release (accès Buffer)
End.

Consumer
Begin
Secure (accès Buffer)
If (N = 0) then
Release (accès Buffer)
Wait (accès change)
Secure (accès Buffer)
End

Consomme (x)

N = N – 1
Signal (état change)
Release (accès Buffer)
End.