UNIVERSITE SAAD DAHLEB DE BLIDA

Faculté des Sciences de l'Ingénieur
Département d'Electronique

MEMOIRE DE MAGISTER

Spécialité : Electronique option Contrôle

ETUDE ET IMPLEMENTATION D'UN SYSTEME DE CONTROLE EN TEMPS REEL D'UN PROCEDE INDUSTRIEL VIA UNE ARCHITECTURE CLIENT/SERVEUR

Par

TOUIZA Maamar

Devant le jury composé de **:**

Mr. A. GUESSOUM	Professeur, USD, BLIDA	Président
Mr. H. SALHI	Maître de conférences, USD, BLIDA	Examinateur
Mr. A. BELKHIR	Professeur, USTHB, ALGER	Examinateur
Mr. R. TAYEB CHERIF	Chargé de cours, USD, BLIDA	Examinateur
Mr. M. OULD ZMIRLI	Maître de conférences, CU, MEDEA	Promoteur

Soutenu à Blida, le 09 Novembre 2005

Abstract

The use of real-time data processing in the implementation of automated systems took a considerable rise in the industrial world. The correct operation of such system does not depend only on a correct logic of calculations, but also of instants when these results are produced.

In this work, we propose a real-time control model of an industrial process based on client/server architecture. The server is implemented under the real-time operating system RTLinux. The role of this OS is to control our process by respecting its specific time constraints. The client in our model is equipped with functionalities necessary to observe and act on the distant process. The two units, client and server, are connected through a point-to-point network, using the programming interface socket TCP/IP.

A concrete application of this model is implemented. It is a control system of an irradiation device for silicon doping by neutron transmutation in a nuclear reactor. This control takes into account all aspects related to irradiation conditions in order to meet requirements needed by this technique of doping.

Résumé

L'utilisation de l'informatique temps réel dans la mise en œuvre des systèmes automatisés a pris un essor considérable dans le monde industriel. Le fonctionnement correct d'un tel système ne dépend pas seulement d'une logique correcte des calculs, mais également du temps auquel ils sont produits.

Dans ce travail, nous proposons un modèle de contrôle en temps réel d'un procédé industriel basé sur une architecture client/serveur. Le serveur est implémenté sous une plate-forme temps réel, Il s'agit du système d'exploitation RTLinux. Ce système d'exploitation a pour mission de contrôler notre procédé en respectant des contraintes de temps spécifiques et doit élaborer, à partir des informations acquises, des informations de commande ou de contrôle dans un temps cohérent avec l'évolution du procédé contrôlé. Le client dans notre modèle a été équipé de fonctionnalités nécessaires pour observer et agir sur le procédé distant. Les deux unités, client et serveur, sont liées à travers une liaison réseau point à point en utilisant l'interface de programmation socket TCP/IP.

Une application concrète de ce modèle a été mise en œuvre. Il s'agit d'un système de contrôle d'un dispositif d'irradiation pour le dopage de silicium par transmutation neutronique dans un réacteur nucléaire. Ce contrôle tient compte des aspects liés aux conditions d'irradiation et à la production pour satisfaire toutes les exigences d'irradiation requises par cette technique de dopage.

ملخص

لقد أخذ استعمال المعلومـ ـ ـاتية في الوقت الحقيقي في تشغيل الأنظمة الآلية تطورا معتبرا في المجال الصناعي. إن الاشتغال الصحيح لهذا النظام لايخضع فقط للمنطق الصحيح للحسابات وإنما يتعلق كذلك بالوقت الذي تنجز فيه .

نقترح في هذا العمل نموذج مراقبة في الوقت الحقيقي لنمط صناعي مبني على شكل زبون وخادم، يعمل الخادم في هذا النموذج تحت نظام استغلال بوقت حقيقي RTLINUX مهمة هذا الأخير هي مراقبة نمطنا الصناعي باحترام التقيدات الوقتية الخاصة وإعداد معلومات التحكم انطلاقا من المعلومات الواردة في وقت متسق مع سيران النمط المراقب. جهز الزبون في هذا النموذج بالوظائف الضرورية للمراقبة والتحكم عن بعد للنمط الصناعي وتم وصل الوحدتين (الخادم والزبون) بواسطة شبكة معلوماتية من طرف لطرف باستعمال الوسيط البرمجي TCP/IP SOCKET

تم تطبيق هذا النموذج على نظام مراقبة لجهاز تشعيع نووي لتنشيط السيلسوم عن طريق الاستحالة النيترونية في مفاعل نووي. يأخذ بعين الاعتبار هذا النظام للمراقبة في الوقت الحقيقي الجوانب المتعلقة بشروط التشعيع والإنتاج للاستجابة لكل متطلبات التشييع اللازمة لتقنية التنشيط.

A mes parents

A mes frères et sœurs

A ma femme et à mes filles

A la mémoire du Cher ami BENAOUDA Zerrouk

REMERCIEMENTS

Ce travail a été réalisé au sein du centre de recherche nucléaire de Birine (C.R.N.B.) dans le cadre de mes activités de recherche et de développement autour des installations nucléaires du Centre.

Je tiens à assurer de ma sincère et totale gratitude et de ma profonde reconnaissance Monsieur Mohamed OULD ZMIRLI du centre universitaire de Médéa, qui a proposé ce travail et qui, par sa confiance, ses conseils amicaux, ses explications pertinentes, a contribué de manière essentielle à la réalisation de ce travail.

C'est un honneur pour moi que Monsieur A.GUESSOUM ait accepté de participer à ce jury et d'en assumer la tâche de président. Je l'en remercie très sincèrement et très profondément car en acceptant de participer à cette commission, Monsieur GUESSOUM témoigne de son intérêt pour mes travaux.

Je remercie Monsieur H.SALHI, Monsieur M. TAYEB CHERIF et Monsieur A.BELKHIR qui m'ont fait l'honneur et le grand plaisir de participer à ce jury, qu'ils soient assurés de toute ma reconnaissance, pour l'intérêt qu'ils ont montré pour mon travail.

Je remercie aussi l'équipe de la division techniques et applications nucléaires représentée par Mr M. ABBACI et notamment le groupe du Département d'irradiation Mrs M. SALHI, M. MOUGHARI et S. LAOUAR pour leur aide appréciable et leur soutien technique durant la réalisation de ce mémoire. Qu'ils trouvent ici ma sincère gratitude.

Mes remerciements vont aussi à tous mes collègues du centre de recherche nucléaire de Birine et en particulier Mrs B.MOHAMMEDI, D.KHELFI, A.BOUZIDI, A.BOURENANE et S.SAADI pour leur aide, leur soutien moral et pour l'ambiance sympathique qu'ils ont su créer.

Mes très vifs remerciements s’adressent à l’équipe de la division étude et développement de l’instrumentation nucléaire à leur tête Mr I.ABDELANI et en particuliers à Mrs A.REHAL et K.HALBAOUI pour leur vive collaboration.

Que soient remerciés tous ceux qui m’ont aidé de prés ou de loin, que ce soit par leur amitié, leur conseil ou leur soutien moral, trouvent dans ces quelques lignes l’expression de ma profonde gratitude.

TABLE DES MATIERES

LISTE DES ILLUSTRATIONS, GRAPHIQUES ET TABLEAUX

1. LISTE DES FIGURES :

2. LISTE DES TABLEAUX :

INTRODUCTION

L'utilisation de l'informatique temps réel dans la mise en œuvre des systèmes automatisés a pris un essor considérable dans le monde industriel. L'ordinateur, sous des formes très variées s'installe dans de très nombreuses applications, où sa capacité d'adaptation fait merveille et contribue progressivement à introduire de nouvelles fonctionnalités. Le prix de l'informatique temps réel y est considéré comme de peu d'importance, relativement aux coûts qui seraient engendrés par la défaillance du procédé piloté. On trouve aujourd'hui les systèmes informatiques temps réel dans des domaines aussi divers que l'aéronautique, le contrôle de procédés industriels, la supervision de centrales nucléaires ou la gestion de réseaux de télécommunication **[1].**

Un système temps réel se caractérise par un environnement avec lequel il maintient une interaction constante de manière à agir en continu sur son environnement en apportant les corrections nécessaires pour son bon fonctionnement. De plus, un système temps réel possède un ensemble de contraintes temporelles dont le respect fait l'objet d'une attention particulière. En effet, un résultat juste mais hors délai est un résultat faux, ce qui implique que l'exactitude des réponses dépend non seulement des calculs des résultats mais également du temps auquel ils sont produits. Une défaillance dans un système de ce type peut souvent occasionner des pertes en vie humaine, ce qui exige une qualité irréprochable dans son développement **[1]**, **[2]**.

Dans ce travail, nous proposons un modèle de contrôle en temps réel d'un procédé industriel basé sur une architecture client/serveur. Ce modèle de contrôle permet à un opérateur distant de contrôler et de surveiller à distance un processus réel. Le serveur reçoit les commandes en provenance du client et les transmet au procédé. De même, il envoie au client les mesures, et les différents états du système contrôlé. Le serveur est implémenté sous une plate-forme temps réel, Il s'agit du système d'exploitation RTLinux. Ce système d'exploitation a pour mission de contrôler notre procédé en respectant des

contraintes de temps spécifiques et doit élaborer, à partir des informations acquises, des informations de commande ou de contrôle dans un temps cohérent avec l'évolution du procédé contrôlé. Le client permet une manipulation à distance de notre procédé et offre à l'opérateur éloigné une interface utilisateur de supervision. Les deux unités, client et serveur, sont liées à travers une liaison réseau point à point en utilisant le mécanisme socket TCP/IP.

Dans ce document, nous commençons au premier chapitre, par exposer la notion de temps réel et les différentes caractéristiques d'un système informatique temps réel. Ensuite, nous décrivons les différentes méthodes d'ordonnancement qui permettent de définir l'ordre d'exécution des tâches dans une application temps réel. Nous présentons en fin de ce chapitre le système d'exploitation RTLinux.

Le deuxième chapitre présente l'architecture client/serveur comme méthode incontournable pour la communication point à point au niveau applicatif. Nous décrivons ensuite le modèle client serveur et le concept de **socket** comme interface de programmation permettant aux programmes d'échanger des données à travers des connexions réseau.

Nous étudions au troisième chapitre deux contrôleurs bien connus dans le domaine de contrôle, à savoir le contrôleur proportionnel intégral dérivé (PID) et le contrôleur prédictif généralisé (GPC). Des informations plus ou moins détaillées sur les aspects théoriques des deux algorithmes ainsi que les techniques de leurs implémentations concrètes seront présentés.

Dans les quatrième et cinquième chapitres, nous présentons l'implémentation réelle de notre modèle de contrôle étudié par l'exposition des aspects de programmation des différents composants du système. Nous montrons aussi une procédure d'évaluation des paramètres temporels des tâches du système pour assurer leur ordonnançabilité. En fin nous faisons une synthèse des deux correcteurs PID et GPC implémentés pour la tâche temps réel de contrôle.

Dans le dernier chapitre, nous proposons une application concrète de notre modèle de contrôle. Il s'agit d'un système de contrôle d'un dispositif d'irradiation pour le dopage de silicium par transmutation neutronique au niveau du réacteur Es-salam de Birine. Ce contrôle tient compte des aspects liés aux conditions d'irradiation et à la production pour satisfaire toutes les exigences d'irradiation requises par cette technique de dopage.

CHAPITRE 1

SYSTEME TEMPS REEL

1.1 – Définition du temps réel :

On qualifie de temps réel, tout système qui doit réagir en ligne et dans le respect de contraintes temporelles, à des sollicitations émises par un environnement extérieur. Plus précisément, on parle d'application temps réel informatisée lorsqu'on requière les services d'un outil informatique pour élaborer à partir d'informations "d'entrée" acquises d'un phénomène extérieur, des informations de "sortie" et ce, dans un temps cohérent avec l'évolution de ce phénomène **[1]**. La terminologie "temps réel" est donc directement liée à l'aptitude du système informatique à réagir face à l'occurrence d'événements externes, avec la contrainte pour les temps de réponse (appelés aussi temps de réaction) d'être suffisamment harmonieux vis à vis de la vitesse d'évolution de ces événements.

On peut aussi définir un système temps réel comme étant un système d'information dont les corrections ne dépendent pas uniquement du résultat logique des algorithmes mais aussi de l'instant où ces résultats ont été produits." **[1]**, **[2]**, **[3]**

J.P. Elloy **[4]** dégage bien la relation système /environnement et qualifie l'application temps réel celle qui met en œuvre un système informatique dont le comportement est conditionné par l'évolution dynamique de l'état du procédé qui lui est connecté. Ce système informatique est alors chargé de suivre ou de piloter ce procédé en respectant des contraintes temporelles définies dans le cahier des charges de l'application.

Parmi l'ensemble des systèmes temps réels existants, on peut distinguer deux grandes familles :

- *les systèmes à contraintes dures (ou strictes):* dans de tels systèmes, un résultat correct qui arrive trop tard est considéré comme une défaillance du système.

- *les systèmes à contraintes souples :* des systèmes dont l'exécution des tâches est aussi assujettie au respect de dates critiques; cependant, on ne permet qu'occasionnellement, certaines de ces dates ne soient pas respectées avec comme seule conséquence, un fonctionnement en mode dégradé et non une panne complète du système.

Un système temps réel est donc une association logiciel/matériel où le logiciel permet, entre autre, une gestion adéquate des ressources matérielles en vue de remplir certaines tâches ou fonctions dans des limites temporelles bien précises. La partie du logiciel qui réalise cette gestion est le système d'exploitation ou noyau temps réel. Ce noyau temps réel va offrir des services au(x) logiciel(s) d'application; ces services seront basés sur les ressources disponibles au niveau du matériel **[1]**.

1.2 - Différence entre un système classique et un système temps réel :

Dans les systèmes informatiques qualifiés de classiques, l'ordre d'exécution des programmes est totalement indépendant du facteur temps mais est déterminé de façon à optimiser l'exploitation de ressources coûteuses et présentes en nombre limité. Dans ce type de système, le temps n’est pas géré d’une façon optimale. L'équité du service est assurée par la mise en oeuvre d'une politique de "temps partagé" qui signifie que le système doit simplement offrir à l'utilisateur un certain confort et assurer que les tâches soient toutes exécutées. Le temps de réponse à un instant donné ne dépend que de la charge infligée au calculateur à cet instant et une valeur jugée excessive de celui-ci ne pourra être considérée comme une erreur de fonctionnement.

Le comportement d’un système informatique est qualifié de « temps réel » lorsqu’il est assujetti à l’évolution d’un procédé qui lui est connecté et qu’il doit piloter ou suivre en réagissant à tous ses changements d’états.

Outre la correction algorithmique, le temps intervient dans la validité d'un programme temps réel de telle sorte que :

- le temps de réaction doit être adapté aux événements externes,
- le programme doit pouvoir fonctionner en continu en maintenant sa capacité à traiter le flux de données d'entrée
- les temps de calculs sont connus (estimés) et peuvent être utilisés dans une analyse de réactivité

Un système d'exploitation temps réel doit s'affranchir donc des incertitudes sur le temps et possède la caractéristique d'être déterministe **[2]**

1.3 - Concepts de base d'un système temps réel :

1.3.1 - Notion d'une tâche temps réel : **[3]**

Une tâche temps réel est un processus élémentaire exécuté par le processeur dont l'exécution doit respecter des contraintes de temps qui expriment des délais critiques ou des dates d'exécution au plus tard (échéances). Elle est constituée de trois zones :

- *le code :* contient les instructions du programme,
- *les données :* contient les données globales du programme,
- *la pile :* contient la pile du programme pour les données temporaires.

Chaque tâche est caractérisée par un contexte. Il contient le nom du processus, son état et sa priorité. Les états possibles d'une tâche sont :

- *active :* la tâche possède le processeur et s'exécute,
- *prête :* la tâche ne possède pas le processeur mais on peut le lui attribuer à tout instant,
- *suspendue :* cet état peut résulter de deux actions :
  - la tâche vient d'être créée,
  - la tâche vient d'être suspendue, elle pourra être réactivée par un autre processus,
- *endormie :* la tâche se met en sommeil pour une certaine durée (le temps qu'un traitement soit fait par une autre tâche).

Une tâche peut être activée à intervalles réguliers (tâche périodique) ou de manière aléatoire (tâche apériodique). Chaque tâche Ti est caractérisée par les paramètres temporels suivants :

- *date de réveil Si :* elle représente l'instant d'arrivée de la tâche dans le système,
- *la capacité Ci :* c'est le temps d'exécution maximale d'une activation de la tâche,
- *la période Pi :* elle définie la périodicité d'exécution de la tâche,
- *l'échéance Di :* c'est l'intervalle maximum admissible entre l'instant d'arrivée d'une requête d'exécution d'un travail de la tâche Ti et l'accomplissement de ce travail.

1.3.2 - Ordonnancement des tâches : **[5]**, **[6]**, **[7]**

La présence de plusieurs tâches doit être gérée par le système d'exploitation. La primitive réalisant ce travail est l'Ordonnanceur (scheduler). Il permet d'allouer le processeur à chaque tâche. Dés lors qu'un ensemble de tâches nécessite l'accès à des ressources en même temps, il se pose un problème d'ordonnancement. Celui-ci consiste alors à trouver un algorithme appelé algorithme d'ordonnancement pour la planification dans le temps de l'accès des tâches à ces ressources tout en respectant leurs contraintes temporelles.

1.3.2.1 - Différentes lois d'ordonnancement classiques :

a - Premier Arrivé, Premier Servi (FIFO) :

Cet algorithme est basé sur l'ordre de création des tâches. Lorsqu'elle est créée, la tâche se voit mise à la fin d'une file d'attente. Lorsque l'ordonnanceur décide de donner la main à une tâche il choisit celle se trouvant en tête de la file d'attente. Il choisit donc la tâche la plus ancienne. Cet algorithme assure que chaque tâche sera exécutée.

Supposons trois tâches, T1, T2 et T3 à exécuter. Leurs durées de traitement valent respectivement 6, 3 et 4 UT. On obtient alors le planning d'exécution suivant :

T1	T1	T1	T1	T1	T1	T2	T2	T2	T3	T3	T3	T3
1	2	3	4	5	6	7	8	9	10	11	12	13

Figure 1.1 : Exemple d'ordonnancement FIFO

Cet algorithme n'est absolument pas efficace en temps réel. Toute tâche créée doit attendre que les précédentes soient terminées. Le début de son exécution dépend donc de la durée des tâches précédentes.

b - Tour de rôle ou tourniquet (round robin) :

Les tâches sont mises dans une file d'attente et sont activées périodiquement. Cette période se nomme « quantum de temps » . A la fin de ce temps, la tâche qui était exécutée est mise à la fin de la file et une autre tâche prend le relais pendant un nouveau quantum.

En reprenant le même exemple que l'ordonnancement FIFO, on obtient l'exécution suivante :

T1	T2	T3	T1	T2	T3	T1	T2	T3	T1	T3	T1	T1
1	2	3	4	5	6	7	8	9	10	11	12	13

Figure 1.2 : Exemple d'ordonnancement Round Robin

Avec cette méthode, les processus sont tous exécutés. Cependant, aucune notion d'importance n'est prise en compte. De plus, le temps d'exécution des processus est proportionnel au nombre de processus. Les processus sont considérés comme ayant tous la même priorité.

1.3.2.2 - Les algorithmes d'ordonnancement temps réel :

Les systèmes classiques utilisent souvent un ordonnanceur basé uniquement sur les quantum de temps. Ces systèmes permettent d'assurer que toutes les tâches seront effectuées et offrent en outre un certain confort d'utilisation pour l'utilisateur. Toutes les tâches sont effectuées à tour de rôle.

Cependant, ces systèmes ne garantissent pas toujours les contraintes de temps. Afin de corriger ces problèmes, les systèmes temps réel mettent en oeuvre des algorithmes d'ordonnancement basés sur les temps d'exécution des tâches.

a – L'algorithme Rate Monotonic (RM):

Pour utiliser cet algorithme, il faut commencer par déterminer la priorité de chaque tâche en affectant, hors ligne, une priorité statique aux tâches.

Les Critères retenus pour les tâches sont :

- les tâches doivent être périodiques : la priorité de chaque tâche est inversement proportionnelle à sa période.
  - les tâches sont indépendantes les unes des autres
  - la capacité ou le temps de traitement des tâches est connue.

Prenant deux tâches indépendantes T1 et T2 ayant comme paramètres temporels (capacité Ci, période Pi) respectivement : (2,5) et (5,15).

Suivant l'algorithme RM, les deux tâches seront exécutés comme suit :

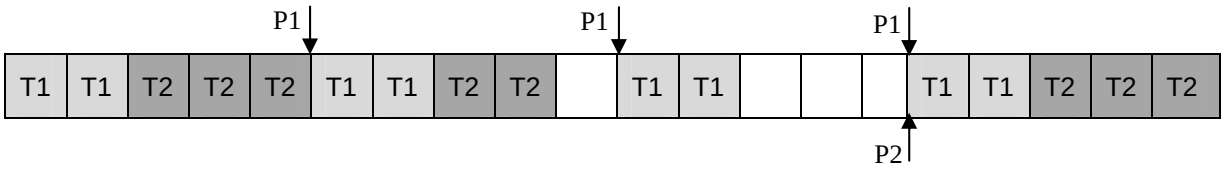


Figure 1.3 : Exemple d'ordonnancement Rate Monotonic

La tâche T1 est toujours exécutée au début de la période, car cette tâche est toujours la tâche la plus prioritaire (tâche de plus petite période). La tâche moins prioritaire T2 est toujours préemptée pour permettre l'exécution de la tâche plus prioritaire T1.

- Test d'ordonnançabilité :

L'ordonnancement RM d'un ensemble de n tâches périodiques (T1…Ti…Tn) est faisable si le facteur d'utilisation :

$$U = \sum_{i=1}^{n} \frac{C_i}{P_i} \qquad \text{vérifie la relation} \qquad U \leq U_0 = n \cdot \left(\sqrt[n]{2} - 1\right) \qquad \textbf{[6]}$$

Le système des deux tâches T1 et T2 de l'exemple précédent est ordonnançable puisque le facteur d'utilisation U=(2/5)+(5/15)=0.73 est inférieur à $U_0 = 2 \cdot \left(\sqrt{2} - 1\right)$ = 0.82

b – L'algorithme Earliest Deadline First (EDF):

C'est un algorithme qui est basé sur des priorités dynamiques (à tout instant, la priorité est inversement proportionnelle à son échéance). Dans cet algorithme on sélectionne la tâche prête qui a l'échéance la plus proche. Lorsque plusieurs tâches ont la même priorité, EDF laisse la tâche courante s'exécuter. L'algorithme EDF permet d'accepter un plus grand nombre de configurations de tâches que l'algorithme RM.

- Test d'ordonnançabilité :

L'ordonnancement EDF d'un ensemble de n tâches périodiques (T1,…Ti…,Tn) est faisable si le facteur d'utilisation :

$$U = \sum_{i=1}^{n} \frac{C_i}{P_i} \qquad \text{vérifie la relation} \quad U < 1 \qquad \textbf{[6]}$$

Considérons un exemple d'un ensemble de deux tâches périodiques (Ci, Pi)= {(2.5), (3.6)}.

Pour cet exemple, la condition d'ordonnançabilité (U=0.9 < 1) est satisfaite. La séquence temporelle d'exécution des tâches selon l'ordonnancement EDF est représentée dans la figure suivante :

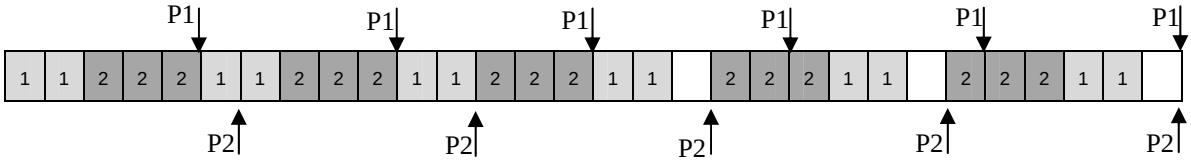


Figure 1.4 : Exemple d'ordonnancement EDF

Dans ce cas, et comme l'algorithme EDF est un algorithme à priorité dynamique, la tâche T1 n'est pas toujours la plus prioritaire, donc ses travaux ne sont plus systématiquement exécutés au début de la période. A chaque instant, le travail en attente à exécuter sera celui dont l'échéance (Fin de la période) est la plus proche.

1.4 - Caractéristiques d'un système temps réel : **[5]**, **[6]**, **[7]**

1.4.1 - Gestion de temps :

Un système temps réel est essentiellement caractérisé par le temps; il doit être capable de répondre en des temps bien définis. Il est donc impératif de contrôler efficacement le temps d'exécution de chaque tâche du système.

Pour manipuler le temps dans un système temps réel, on utilise l'horloge temps réel du système. Ce composant matériel a pour objet de fournir une indication sur le temps qui s'écoule. La forme la plus simple est un générateur d'impulsions périodiques. La fréquence de ces impulsions est généralement divisée par un composant dédié *(timer)* et génère des interruptions périodiques. Cette période est utilisée comme base de temps par le système temps réel. Si cette période est faible, on a une bonne précision sur le temps, mais on surcharge le CPU par les interruptions d'horloge. Le choix de cette période est intiment lié à l'application (constante de temps de procédé à contrôler) et au support d'exécution.

1.4.2 - Gestion de mémoire :

Lorsqu'un programme est exécuté, la mémoire dont il a besoin est souvent plus importante que la mémoire physique (RAM) disponible. Pour résoudre ce problème on a recourt à une mémoire dite virtuelle. Ce procédé consiste à laisser au programme la liberté de pouvoir accéder à la quantité de mémoire dont il a besoin.

Le problème d'un tel mécanisme réside dans les transferts entre la mémoire virtuelle et la mémoire physique qui occupe du temps machine. De plus, ce temps est augmenté quand toute la mémoire physique est occupée. Si le programme demande l'accès à une adresse mémoire non chargée en mémoire physique, il faut libérer une place. Ceci double donc le temps machine utilisé.

On associe souvent à la mémoire virtuelle le mécanisme de pagination qui consiste à découper la mémoire en plusieurs parties appelées pages. Pour savoir quelles sont les pages à charger/décharger, le système gère une table des pages.

Elle permet de savoir à tout instant si une page est présente en mémoire ou non. Lorsqu'une tâche demande l'accès à la mémoire, le système d'exploitation détermine si la page correspondante est chargée ou non.

En conclusion, on constate que l'utilisation du mécanisme de mémoire virtuelle paginée peut entraîner des fluctuations importantes dans les temps d'exécution des tâches ce qui ramène les exécutifs temps réel à travailler directement en mémoire physique. Ce choix engendre une dégradation dans les performances du système, cependant on garantie un temps moyen d'exécution plus stable.

1.4.3 - Gestion des interruptions :

Une interruption est générée pour signaler au processeur qu'un périphérique est prêt (interruptions matérielles) ou qu'une tâche souhaite faire appel á une primitive du noyau (interruptions logicielles). Lorsqu'une interruption survient, diverses actions sont nécessaires afin de maintenir l'intégrité du système. Il faut d'abord sauvegarder le contexte de la tâche pour pouvoir y revenir après le traitement de l'interruption. On détermine ensuite la source de l'interruption et on exécute le traitement correspondant. Pour connaître le temps du traitement d'une interruption (le temps qui sépare l'abandon d'une tâche jusqu'à son retour) il faut prendre en compte plusieurs temps :

- *temps de commutation* : temps moyen pris par le système pour commuter entre deux tâches (sauvegarde et restitution de contextes) ;
- *temps de sélection* : temps moyen pour déterminer l'identité de la prochaine tâche courante. Cette tâche est déterminée à partir de la source de l'interruption, notamment lors d'interruptions matérielles ;
- *temps de latence* : intervalle de temps situé entre le moment où l'interruption est reçue, et le moment où elle commence à être traitée. Ce temps ne dépend pas seulement de l'efficacité de l'exécutif mais aussi de l'architecture du processeur.

Le temps de réponse à une interruption correspond à la somme de ces trois temps.

Les systèmes classiques n'offrent pas de garanties concernant le temps de latence des interruptions. Pour accéder à un niveau aussi proche du matériel, le système utilise les drivers de périphériques. Ils exécutent leurs instructions dans un mode de protection réservé au système d'exploitation lui-même.

Dans un système temps réel, seul le strict minimum est assuré par le driver. Cette durée peut alors être évaluée et constitue un paramètre fixe du système. En plus, le mécanisme de gestion d'interruption doit être intégré avec le système d'ordonnancement de l'application, de sorte que le driver puisse être programmé en tant que n'importe quelle autre tâche dans le système.

1.4.4 - Synchronisation et communication :

La présence de plusieurs tâches dans un système unique oblige à avoir une certaine cohérence entre chacune d'elles. Elles ne doivent pas être mises en conflit. Ainsi, lorsque deux tâches souhaitent accéder à une même ressource, elles ne doivent pas le faire en même temps. Elles doivent donc être capable de communiquer et de se synchroniser entre elles. Pour ce faire, divers moyens peuvent être mis en œuvre. La solution la plus connue à ce problème est l'utilisation d'un sémaphore binaire accompagné de ses deux primitives P et V. De par leur simplicité, elles peuvent être implantées de façon matérielle et utilisées pour réaliser des mécanismes de synchronisation plus évolués.

1.5 - Présentation du système d'exploitation temps réel, RTLinux :

RTLinux est une extension de Linux apportant une dimension temps réel au noyau standard. Cette extension a été imaginé en 1997 par Victor Yodaiken et Michael Barabanov du "Department of Computer Science of the Institute for Mining and Technology of New Mexico". **[8]**, **[9]**

RTLinux est un système d'exploitation dans lequel cohabitent un micro-noyau temps réel et le noyau Linux. L'intention des concepteurs est de continuer d'utiliser les services de haut niveau du système à temps partagé Linux, tout en permettant de fournir un comportement déterministe.

En standard, le noyau Linux dispose de certaines fonctionnalités utiles pour le temps réel. Il supporte notamment la norme POSIX qui définit une interface standard pour la gestion des tâches concurrentes. Mais en aucun cas, le noyau standard n'est prédictif. Il faut pour cela avoir recours à une variante temps réel de Linux capable de garantir l'exécution d'une tâche dans un délai donné. La figure 1.5 montre une vue d'ensemble du noyau Linux.

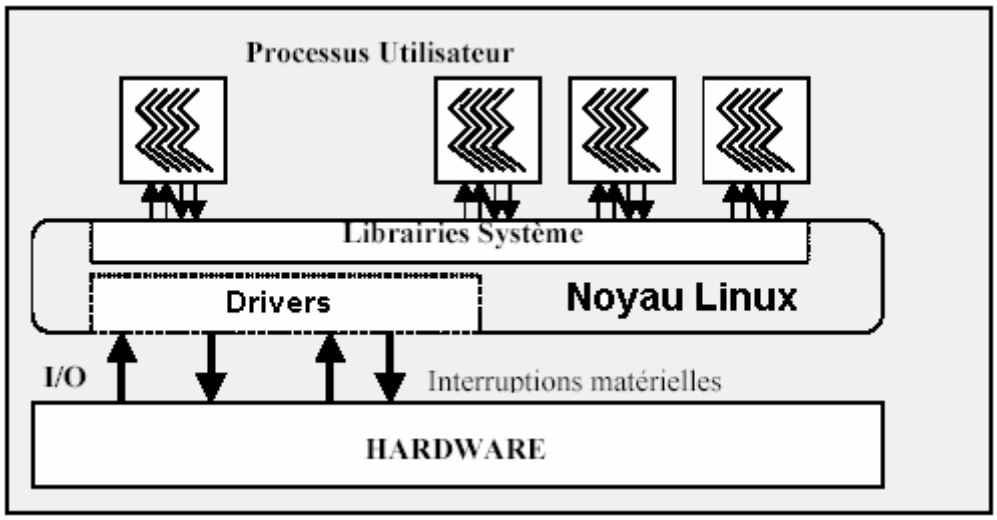


Figure 1.5 : Présentation du noyau Linux

En effet, les systèmes temps réel ont souvent besoin d'un support réseau, d'une interface graphique, d'un système de fichier et éventuellement d'un accès à une base de données. Ceci peut être fourni par un système d'exploitation Linux. La technique adoptée par RTLinux est celle qui permet à ce micro-noyau d'exécuter le véritable noyau Linux comme sa tâche de plus faible priorité. Dans cet environnement, toutes les interruptions sont initialement prises en compte par le noyau temps réel et sont passées à Linux seulement s'il n'y a pas de tâche temps réel à exécuter. (Voir figure 1.6).

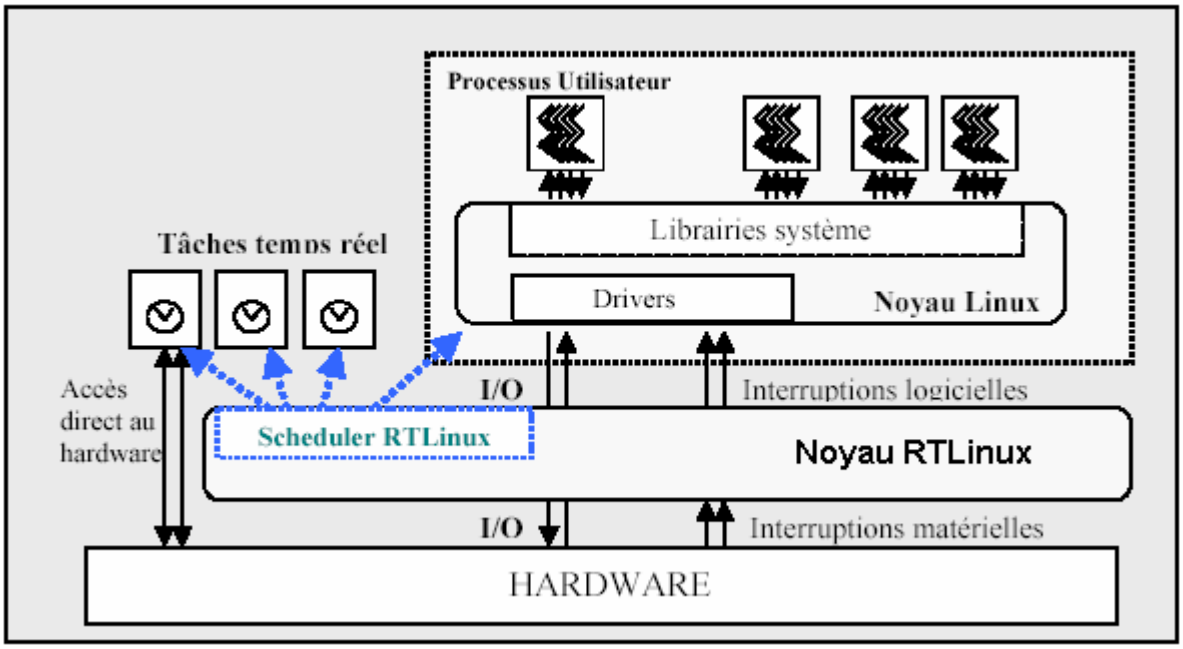


Figure 1.6 : Présentation du système d'exploitation RTLinux

Les tâches temps réel s'exécutent en mode noyau et peuvent donc accéder au matériel sans problème (interruptions, registre d'E/S, DMA, ...). Par contre la programmation en mode noyau interdit l'allocation dynamique de mémoire et ne permet pas d'accéder aux données dans l'espace utilisateur des processus ou aux appels systèmes Linux directement. Pour cela RTLinux propose un moyen de communication entre les tâches temps réel et les processus utilisateurs basé sur les FIFOs. Ces fichiers spéciaux sont vus et utilisés, coté processus, à l'aide des appels systèmes classiques de gestion des fichiers sous Linux. Coté RTLinux, une bibliothèque de fonctions est proposée pour permettre l'envoi et la réception de données par ce biais.

RTLinux version 3.1, qu'on a utilisé, est structuré en un module principal et plusieurs modules optionnels :

- Le module principal implémente la gestion des interruptions et le dialogue avec le noyau linux.
- Le module d'ordonnancement (rtl_sched) : ce module implémente une API de gestion des tâches à la mode POSIX ainsi qu'une bibliothèque de compatibilité avec les versions antérieures.

- Le module de gestion d'horloge (rtl_time) : contrôle les horloges systèmes et offre un jeu de fonctions pour les utiliser.
- Le module d'E/S (rtl_posixio) : fournit un interface de type POSIX pour les drivers.
- Le module de gestion des FIFOs (rl_fifo) : permet de créer des canaux de communication entre les tâches temps réel et les processus utilisateur.
- un module de gestion de sémaphores.
- un module de gestion de la mémoire partagé

CHAPITRE 2

L'ARCHITECTURE CLIENT/SERVEUR

2.1 - Introduction :

La communication entre microordinateurs pour le pilotage à distance des processus industriels s'appuie aujourd'hui essentiellement sur le support réseau vu son évolution technologique et son utilisation croissante. L'architecture logicielle la plus utilisée repose sur le modèle client/serveur qui assure une communication entre les deux entités grâce à une connexion par un canal d'échange appelé « Socket ».

Cette architecture est très souple et permet au client et au serveur de se trouver sur deux postes connectés en réseau d'une manière transparente.

Dans ce chapitre, on présente en premier lieu la structure d'un réseau informatique et les différents protocoles de communications utilisés ,puis on définit l'architecture client serveur et on donne enfin une description complète des différentes méthodes de connexion par socket.

2.2 - Structure d'un réseau informatique : **[10]**

Un réseau informatique présente deux aspects :

- des applications réseau qui se servent pour dialoguer,
- le sous-système de communication réseau qui véhicule les données entre applications.

Entre 1977 et 1984 les professionnels ont développé un modèle de conception appelé « Modèle de Référence de Connexion entre Systèmes Ouverts » (Reference Model of Open Systems Interconnection - OSI). Le modèle ISO/OSI utilise une représentation sous forme de « couches » de manière à organiser un réseau en modules fonctionnels bien définis.

Chaque couche (layer) propose une fonctionnalité ou un service spécifique aux couches qui lui sont adjacentes et avec lesquelles elle communique. Ce modèle, fournit sept couches réseau définies dans le tableau 2.1

Tableau 2.1 : description des couches réseau

Numéro	Nom de la couche	Fonctionnalité
7	Couche Application	Elle concerne les applications utilisant le réseau
6	Couche présentation	Elle permet la traduction des données transmises dans un format, commun à toutes les applications, défini par la norme afin de permettre à des ordinateurs ayant différents formats de communiquer entre eux.
5	Couche session	Cette couche effectue la mise en relation de deux applications et fournit les protocoles nécessaires à la synchronisation des .changes, leur arrêt et leur reprise. Elle permet d'attribuer un numéro d'identification à l'échange d'une suite de messages entre applications.
4	Couche transport	La fonction de base de cette couche est d'accepter des données de la couche session, de les découper, le cas échéant, en plus petites unités, et de s'assurer que tous les morceaux arrivent correctement de l'autre coté.
3	Couche réseau	Elle détermine le chemin que devra prendre des données et s'occupe du trafic réseau et de sa congestion.
2	Couche liaison	Pour qu'une succession de bits prenne sens, cette couche définie des conventions pour délimiter les caractères ou des groupes de caractères. L'unité de traitement à ce niveau est appelée trame de données (de quelques centaines d'octets).
1	Couche physique	Elle représente la carte réseau qui offre les services de l'interface entre l'équipement de traitement informatique et le support physique de transmission.

Chaque couche réseau utilise un protocole particulier pour discuter avec les couches qui lui sont directement adjacentes. Un protocole est un ensemble de règles et de conventions reconnues pour la communication. Les protocoles qui nous intéressent se nomment communément TCP/IP.

2.2.1 - Les protocoles TCP/IP :

C'est un ensemble de protocoles qui permettent de communiquer via Internet.

Tableau 2.2 : Description des protocoles Internet

IP	*Internet Protocol*	Protocole de couche réseau
TCP	*Transport Control Protocol*	Protocole de couche transport
UDP	*User Datagram Protocol*	Un autre protocole de couche transport (moins complexe mais moins fiable que TCP)
ICMP	*Internet Message Control Protocol*	Protocole qui transporte les messages d'erreurs réseau entre autres
IGMP	*Internet Group Management Protocol*	Protocole de gestion des groupes

La disposition de ces protocoles est schématisée sur la figure 2.1. Nous remarquons dans cette figure que la couche physique est identique à celle du modèle ISO/OSI. Elle inclut le support de transmission. La couche liaison inclut une interface matérielle et deux modules de protocole : le protocole de résolution d'adresses (address resolution protocol *ARP)* qui transforme les adresses de la couche réseau en adresses de la couche liaison et le protocole de résolution d'adresses inversé (Reverse Adress Resolution Protocol - *RARP)* qui effectue la transformation inverse.

D'après la figure 2.1, notre application couvre trois couches simultanées : application, présentation et session. Il faut alors choisir entre deux protocoles possibles pour la couche de transport : *UDP* ou *TCP*. Dans notre étude, nous n'avons pas besoin des services offerts par les protocoles *IGMP* et *ICMP*.

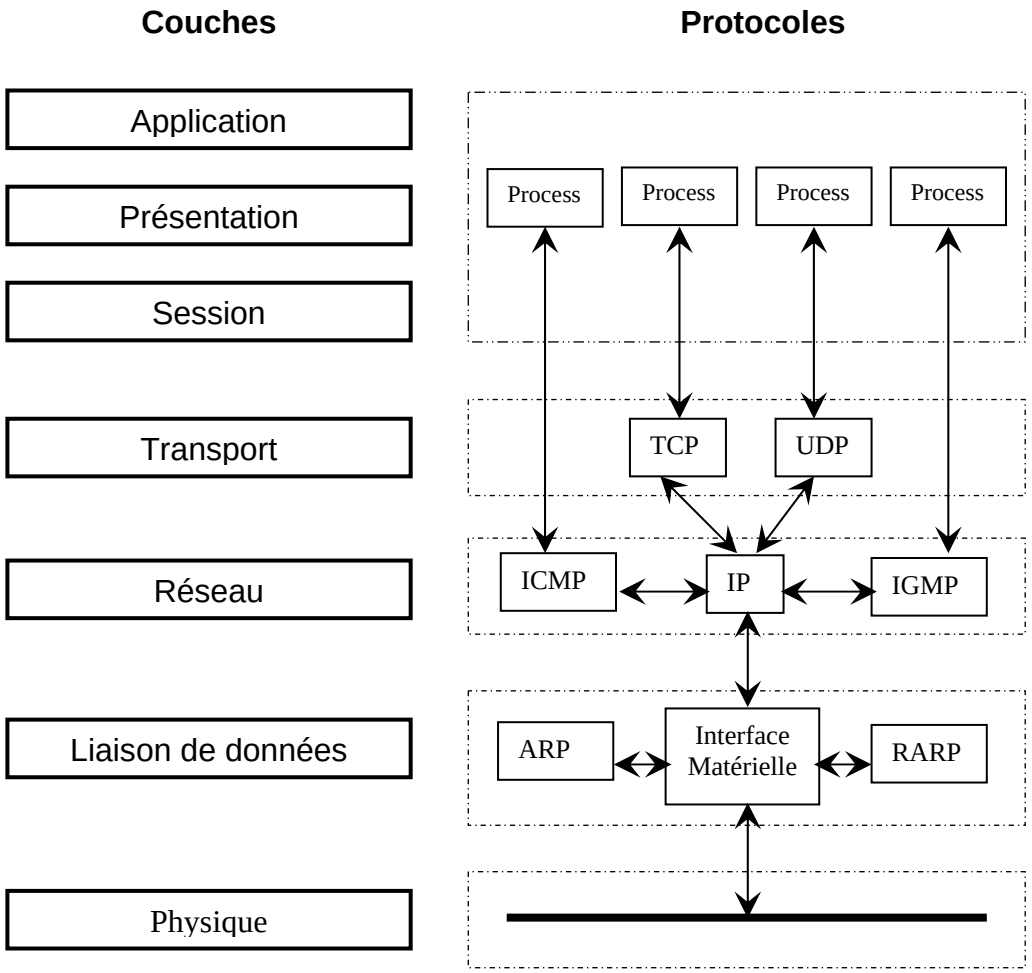


Figure 2.1 : Correspondance entre les couches du modèle ISO et les protocoles Internet

2.2.2 - TCP et UDP

TCP est un protocole orienté connexion par flots d'octets. C'est à dire que lorsque deux applications veulent dialoguer, elles doivent d'abord établir une connexion à l'image du téléphone où il faut d'abord établir la ligne avant de dialoguer avec son correspondant. D'autre part, toutes les données émises, quelque soit leur longueur, sont transmises dans le même ordre.

UDP est un protocole non orienté connexion par datagrammes. C'est à dire que les deux applications n'ont pas besoin de créer de connexion ; elles s'envoient des données sous la forme de datagrammes (des séquences d'octets indépendantes les unes des autres).

C'est un protocole non fiable car on n'est pas certain que tous les datagrammes émis aient été réceptionnés par le destinataire et il n'y a pas de moyen de le savoir. D'autre part, on n'est pas sûr de l'ordre de réception des datagrammes. L'intérêt d'un tel protocole (UDP) est qu'il est nettement moins complexe à programmer ; les aspects de fiabilité sont alors à programmer dans la couche application selon ses propres besoins.

Les protocoles de communication de la famille TCP/IP permettent la communication point à point entre deux applications s'exécutant éventuellement sur deux machines distinctes.

2.2.3 - Le Protocole IP :

La description du protocole IP le présente comme étant responsable de l'acheminement des données et de leur livraison dans un réseau. Cette fonction porte le nom de routage. Le routage au niveau IP est entièrement basé sur les numéros d'adresse.

Chaque système possède une table de numéros d'adresses. On identifie chaque hôte sur le réseau par une adresse Internet de 32 bits. La méthode de décrire cette adresse 32 bits est la notation IP pointillé décimale qui divise l'adresse en 4 octets séparés de points décimaux (ex : 193.120.10.2).

Au niveau application, nous pouvons affecter à chaque adresse internet un nom représentatif formé d'une chaîne de caractère pour faciliter la reconnaissance des hôtes. Ces désignations doivent être mentionnées dans le fichier HOST du système.

2.3 - Modèle Client/Serveur : **[11]**

L'architecture client/serveur est devenue la méthode incontournable pour la communication point à point au niveau application, quand le protocole utilisé est de la famille TCP/IP.

C'est un modèle utilisé par la plupart des programmeurs. Les communications entre deux applications exécutées sur des ordinateurs

différents nécessitent une connexion réseau. L'une des applications doit attendre l'initiative de la communication de la part de l'autre application.

Le détail du transfert effectif des données entre les deux applications est spécifié par le protocole de communication, mais le moment et la façon dont les applications interagissent entre elles sont laissés à la charge du programmeur.

Les applications peuvent être classées en deux catégories:

- les clients : applications qui prennent l'initiative du lancement de la communication, c'est à dire demande l'ouverture de connexion, l'envoi des requêtes et l'attente des réponses des requêtes.
- les serveurs : applications qui attendent la communication, c'est-à-dire l'attente d'une demande d'ouverture de connexion, la réception des requêtes et l'envoi des réponses.

Dans une connexion, chaque extrémité s'appelle un *socket*. Le socket client demande une connexion à un socket serveur puis dialogue avec lui. Le socket serveur attend une demande de connexion puis dialogue avec le ou les sockets clients dont la requête est acceptée.

2.4 - Description d'un socket : **[11]**

2.4.1 - Définition :

Un socket est un mécanisme d'interface de programmation qui permet aux programmes d'échanger des données à travers des connexions réseau.

Les sockets vous permettent de créer une application pouvant communiquer avec d'autres systèmes par TCP/IP et ses protocoles associés. A l'aide des sockets, vous pouvez lire et écrire sur des connexions à d'autres machines sans vous soucier des détails concernant le logiciel réseau.

L'utilisation des sockets vous permet d'écrire des applications serveur ou client qui lisent et écrivent sur des systèmes distants. Les connexions par socket peuvent être scindées en deux groupes principaux indiquant la façon

dont la connexion a été ouverte et le type de connexion : connexions client et connexions serveur

Lorsque la connexion au socket client est effective, la connexion serveur est identique à une connexion client. Les deux extrémités ont les mêmes possibilités et reçoivent des événements de même type.

2.4.2 - Concept de base d'un socket :

Le concept de socket est assez difficile à appréhender. Il a été introduit pour accomplir les communications inter-processus. Cela veut dire qu'un socket est utilisé pour permettre aux processus de communiquer entre eux de la même manière que le téléphone nous permet de communiquer entre nous.

Il existe toute une bibliothèque de fonctions qui permettent de manipuler les sockets et ses paramètres associés. Ces fonctions sont définies sur toutes les plates-formes et systèmes d'exploitation comme étant des commandes système dédiées au réseau et exploitables par la majorité des langages de programmation.

L'analogie entre le concept de socket et le téléphone est assez proche, et sera utilisée pour éclaircir la notion de socket.

Pour recevoir des coups de téléphone, vous devez d'abord installer le téléphone chez vous. De la même façon vous devez commencer par créer un socket avant d'attendre des demandes de communications. La commande socket() est alors utilisée pour créer un nouveau socket.

Seulement il faut créer un socket avec les bonnes options. Vous devez spécifier le type d'adressage du socket. Les deux types d'adressage les plus répandus sont AF_UNIX (famille d'adresse UNIX utilisé pour les communications locales sous Unix et Linux) et AF_INET (famille d'adresses Internet utilisé par tous les systèmes d'exploitation).

AF_INET utilise les adresses Internet qui sont du format suivant xx.xx.xx.xx (ex: 178.33.174.34). En plus des adresses Internet, on a besoin

aussi d'un numéro de port sur la machine pour pouvoir utiliser plusieurs sockets simultanément.

Une autre option à spécifier lors de la création d'un socket est son type. Les deux types les plus répandus sont SOCK_STREAM et SOCK_DGRAM. SOCK_STREAM sont spécifiques au mode connecté alors que SOCK_DGRAM sont spécifiques au mode déconnecté.

De la même façon qu'on vous attribue un numéro de téléphone pour recevoir vos appels, on doit spécifier au socket une adresse à laquelle il doit recevoir les messages qui lui sont destinés. Ceci est réalisé par la fonction bind() qui associe un numéro au socket.

Les sockets de type SOCK_STREAM ont la possibilité de mettre les requêtes de communication dans une file d'attente, de la même façon que vous pouvez recevoir un appel pendant une conversation téléphonique. C'est la fonction listen() qui permet de définir la capacité de la file d'attente (jusqu'à 5) . Il n est pas indispensable d'utiliser cette fonction mais c'est plutôt une bonne habitude que de ne pas l'oublier.

L'étape suivante est d'attendre les demandes de communication. C'est le rôle de la fonction accept(). Cette fonction retourne un nouveau socket qui est connecté à l'appelant. Le socket initial peut alors se remettre à attendre les demandes de communication. C'est pour cette raison qu'on exécute un fork() à chaque demande de connexion.

On sait maintenant comment créer un socket qui reçoit des demandes de communication, mais comment l'appeler ? Pour le téléphone vous devez d'abord avoir le numéro avant de pouvoir appeler. Le rôle de la fonction connect() est de connecter un socket a un autre socket qui est en attente de demande de communication. Maintenant qu'une connexion est établie entre deux sockets, la conversation peut commencer. C'est le rôle des fonction read() et write(). A la fin de la communication on doit raccrocher le téléphone ou fermer le socket qui a servi à la communication. C'est le rôle de la fonction close().

2.4.3 - Adresses des sockets :

2.4.3.1 - les familles d'adresses :

Il existe plusieurs familles d'adresses, chacune correspondant à un protocole particulier. Les familles les plus répandues sont:

AF_UNIX Protocoles internes de UNIX et LINUX
AF_INET Protocoles Internet

2.4.3.2 - les structures d'adresse :

Plusieurs appels systèmes réseaux sous Linux nécessitent un pointeur sur une structure d'adresse de socket. Sous le langage C et C++, La définition de cette structure se trouve dans :

```
<sys/socket.h>
struct sockaddr
 {
        u_short         sa_family ;
        char            sa_data[14] ;
 };
```

sa_family : adresse de famille et peut prendre la valeur AF_xxx
sa_data : peut contenir jusqu'à 14 octets de protocole spécifique d'adresse Interprété selon le type d'adresse.

Pour la famille Internet les structures suivantes sont définies dans le fichier

```
<netinet/in.h>.
struct in_addr
  {
    u_long      s_addr ;
  } ;
```

s_addr : 32 bits constituants l'identificateur du réseau et de la machine hôte ordonnés selon l'ordre réseau.

```
struct sockaddr_in
{
        short           sin_family ;
        u_short         sin_port ;
        struct  in_addr     sin_addr ;
        char            sin_zero[8] ;
} ;
```

sin_family : AF_INET ;

sin_port Pour effectuer une communication inter-processus il faut identifier de façon unique les processus. Un numéro de port de 16 bits est associé à chaque processus.

sin_addr : 32 bits constituants l'identificateur du réseau et de la machine hôte ordonnés selon l'ordre réseau.

sin_zero : inutilisé ;

2.5 - Les appels système :

2.5.1 - L'appel système socket :

```
#include <sys/types.h>
#include <sys/socket.h>
int socket (int family, int type, int protocole) ;
```

La variable *family* peut prendre 2 valeurs dont les préfixes commencent par AF comme Famille d'Adresse :

AF_UNIX : Protocoles internes de UNIX ,
AF_INET : Protocoles Internet.

La variable type prend principalement 2 valeurs :

SOCK_STREAM **:** utilisé en mode connecté au dessus de TCP.
SOCK_DGRAM : utilisé en mode déconnecté avec des datagrammes au dessus de UDP.

L'argument protocole dans l'appel système socket est généralement mis à 0. Dans certaines applications spécifiques, il faut cependant spécifier la valeur du protocole. Les combinaisons valides pour la famille AF_INET sont les suivantes :

TYPE	PROTOCOLE	PROTOCOLE ACTUEL
SOCK_DGRAM	IPPROTO_UDP	UDP
SOCK_STREAM	IPPROTO_TCP	TCP

Les constantes IPPROTO_xxx sont définies dans le fichier <netinet/in.h>.

L'appel système socket retourne un entier dont la fonction est similaire à celle d'un descripteur de fichier. Nous appellerons cet entier descripteur de sockets (sockfd).

2.5.2 - L'appel système bind :

#include <sys/types.h>
#include <sys/socket.h>
*int bind (int sockfd, struct sockaddr *myaddr , int addrlen) ;*

Le premier argument est le descripteur de socket retourné par l'appel socket. Le second argument est un pointeur sur une adresse de protocole spécifique et le troisième est la taille de cette structure d'adresse.

Il y a 3 utilisations possibles de bind :

1 - Le serveur enregistre sa propre adresse auprès du système. Il indique au système que tout message reçu pour cette adresse doit lui être fourni. Que la liaison soit avec ou sans connexion l'appel de bind est nécessaire avant l'acceptation d'une requête d'un client.

2 - Un client peut enregistrer une adresse spécifique pour lui même.

3 - Un client sans connexion doit s'assurer que le système lui a affecté une unique adresse que ses correspondants utiliseront afin de lui envoyer des messages.

L'appel de bind rempli l'adresse locale et celle du process associés au socket.

2.5.3 - L'appel système connect :

Un socket est initialement créé dans l'état non connecté, ce qui signifie qu'il n'est associé à aucune destination éloignée. L'appel système connect associe de façon permanente un socket à une destination éloignée et la place dans l'état connecté.

Un programme d'application doit invoquer connect pour établir une connexion avant de pouvoir transférer les données via un socket de transfert fiable en mode connecté.

Les sockets utilisés avec les services de transfert en mode datagramme n'ont pas besoin d'établir une connexion avant d'être utilisés, mais procéder à interdir de transférer des données sans mentionner à chaque fois, l'adresse de destination.

#include <sys/types.h>
#include <sys/socket.h>
*int connect (int sockfd, struct sockaddr *servaddr , int addrlen) ;*

Avec :
sockfd : est le descripteur de socket retourné par l'appel socket.
Servaddr : est un pointeur sur une structure d'adresse de socket qui indique l'adresse de destination avec laquelle le socket doit se connecter.
addrlen : est la taille de la structure d'adresse.

2.5.4 - L'appel système listen :

#include <sys/types.h>
#include <sys/socket.h>
int listen (int sockfd, int backlog) ;

Cet appel est généralement utilisé après les appels socket et bind et juste avant l'appel accept. L'argument backlog spécifie le nombre de connexions à établir dans une file d'attente par le système lorsque le serveur exécute l'appel accept. Cet argument est généralement mis à 5 qui est la valeur maximale utilisée.

2.5.5 - L'appel système accept :

#include <sys/types.h>
#include <sys/socket.h>
*int accept (int sockfd, struct sockaddr *peer , int *addrlen) ;*

L'argument sockfd désigne le descripteur de socket sur lequel seront attendues les connexions. Peer est un pointeur vers une structure d'adresse de socket. Lorsqu'une requête arrive, le système enregistre l'adresse du client dans la stucture *peer et la longueur de l'adresse dans *addrlen. Il crée alors un nouveau socket connecté avec la destination spécifiée par le client, et renvoi à l'appelant un descripteur de socket. Les échanges futurs avec ce client se feront donc par l'intermédiaire de ce socket.

Le socket initial sockfd n'a donc pas de destination et reste ouvert pour accepter de futures demandes. Tant qu'il n'y a pas de connexions le serveur se bloque sur cet appel. Lorsqu'une demande de connexion arrive, l'appel système accept se termine. Le serveur peut gérer les demandes itérativement ou simultanément :

Dans l'approche itérative, le serveur traite lui même la requête, ferme le nouveau socket puis invoque de nouveau accept pour obtenir la demande suivante.

Dans l'approche parallèle, lorsque l'appel système accept se termine, le serveur crée un serveur fils chargé de traiter la demande (appel de fork et exec). Lorsque le fils a terminé, il ferme le socket et meurt. Le serveur maître ferme quand à lui la copie du nouveau socket après avoir exécuté le fork. Il appelle ensuite de nouveau accept pour obtenir la demande suivante.

2.5.6 - Emission d'information sur un socket :

Une fois que le programme d'application dispose d'un socket, il peut l'utiliser afin de transférer des données. Cinq appels système sont utilisables : send, sendto, sendmsg, write et writev. Send, write et writev ne sont utilisables qu'avec des sockets en mode connecté car ils ne permettent pas d'indiquer d'adresse de destination. Les différences entre ces trois appels sont mineures :

```
#include <sys/types.h>
#include <sys/socket.h>
write ( int sockfd, char *buff, int nbytes ) ;
writev ( int sockfd, iovec *vect_E/S, int lgr_vect_E/S ) ;
int send (int sockfd, char *buff, int nbytes, int flags ) ;
```

sockfd	contient le descripteur de socket .
buff	est un pointeur sur un tampon où sont stockées les données à envoyer.
nbytes	est le nombre d'octets ou de caractères que l'on désire envoyer
vect_E/S	est un pointeur vers un tableau de pointeurs sur des blocs qui constituent le message à envoyer.
flags	drapeau de contrôle de la transmission :

Pour le mode non connecté, on a deux appels sendto et sendmsg qui imposent d'indiquer l'adresse de destination :

```
#include <sys/types.h>
#include <sys/socket.h>
int sendto (int sockfd, char *buff, int nbytes, int flags, struct sockaddr *to, int addrlen) ;
```

Les quatre premiers arguments sont les mêmes que pour send, les deux derniers sont l'adresse de destination et la taille de cette adresse.

Pour les cas où on utiliserait fréquemment l'appel sendto qui nécessite beaucoup d'arguments et qui serait donc d'une utilisation trop lourde on définit la structure suivante:

```
struct struct_mesg
{       int *sockaddr ;
        int   sockaddr_len ;
        iovec *vecteur_E/S
        int    vecteur_E/S_len
        int   *droit_d'acces
        int    droit_d'acces_len
 }
```

Cette structure sera utilisée par l'appel sendmsg :

```
int sendmsg ( int sockfd, struct struct_mesg, int flags ) ;
```

2.5.7 - Réception d'information sur un socket :

On distingue 5 appels système de réception d'information qui sont symétriques aux appels d'envoi. Pour le mode connecté on a les appelles read, readv et recv et pour le mode sans connexion on a les appels recvfrom et recvmsg.

*int read (int sockfd, char *buff, int nbytes) ;*
*int readv (int sockfd, iovec *vect_E/S, int lgr_vect_E/S) ;*
*int recv (int sockfd, char *buff, int nbytes, int flags) ;*

sockfd est le descripteur sur lequel les données seront lues .
buff est un pointeur sur un buffer où seront stockées les données lues
nbytes est le nombre maximal d'octets ou de caractères qui seront lus.
readv permet de mettre les données lues dans des cases mémoire non contiguës. Ces cases mémoires sont pointées par un tableau de pointeurs qui lui même est pointé par vect_E/S.
lgr_vect_E/S est la longueur de ce tableau.

Pour le mode sans connexion, il faut préciser les adresses des correspondants desquels on attend des données.

*int recvfrom (int sockfd, char *buff, int nbytes, int flags, struct sockaddr *from int addrlen) ;*

Pour les mêmes raisons que pour sendto et sendmsg, et pour des raison de symétrie on a défini l'appel recvmsg qui utilise la même structure que sendmsg.

int recvmsg (int sockfd, struct struct_mesg, int flags) ;

2.5.8 - L'appel système de multiplexage select() :

L'appel système qui permet le multiplexage des sockets est select(). select() examine les descripteurs qui lui sont passés en paramètre et teste si certains sont prêts à lire, écrire ou ont une condition d'exception. Les données hors bande sont les seules conditions d'exception que peut recevoir un socket.

La description de l'appel système select() est :

*int select (int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout)*

On ne s'attardera pas sur les structures passées en paramètre, seuls les fonctions qui permettront leur manipulation seront détaillées. Les arguments 2, 3 et 4 de select() sont des pointeurs sur des structures qui contiendront les descripteurs de sockets à tester en lecture, écriture et en réception d'exception par le select.

Le dernier argument timeout, s'il est non nul , spécifie l'intervalle de temps maximal à attendre avant de sortir de select. Si ce pointeur est nulle select() se bloquera indéfiniment.

Pour réaliser un polling sur select() (i.e. select ne rendra la main que lorsqu'au moins un descripteur aura été sélectionné) le pointeur timeout doit être non nul mais pointant sur une structure timeval nulle.

A la fin de select les structures passées en arguments contiennent les descripteurs sélectionnés par la fonction select.

Avant d'appeler select() , il faut initialiser ses arguments. On commencera toujours par les initialiser à une structure nulle par la fonction *FD_ZERO (fd_set &fdset*).

void FD_SET (int fd, fd_set &fdset) ajoute le descripteur fd à fdset.
void FD_CLR (int fd, fd_set &fdset) retire le descripteur fd a fdset.

A la fin de select, la fonction *int FD_ISSET (int fd, fd_set &fdset)* nous permet de savoir si le descripteur fd est sélectionné dans fdset ou non. Cette fonction est à appeler après select().

2..6 - Enchaînement des appels système :

2.6.1 - mode connecté :

Deux approches de programmation de serveurs se présentent: l'approche itérative et l'approche parallèle. Rappelons que dans l'approche itérative, le serveur traite lui même la requête, ferme le nouveau socket puis invoque de nouveau accept() pour obtenir la demande suivante.

L'enchaînement dans le temps des appels système pour un serveur itératif se résume par la figure 2.2

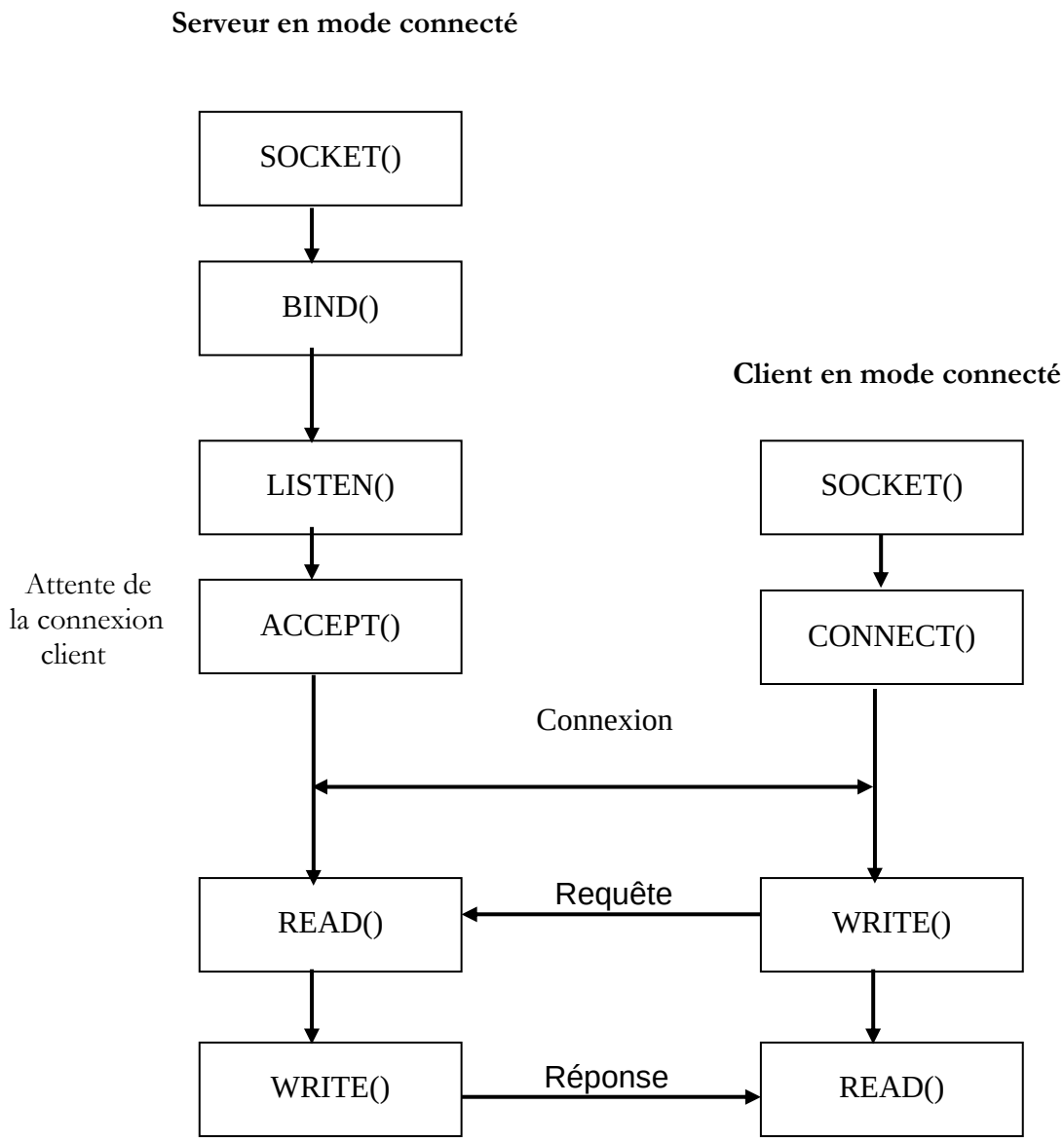


Figure 2.2: L'enchaînement des appels système en mode connecté pour un serveur itératif

Dans l'approche parallèle, lorsque l'appel système accept se termine le serveur crée un serveur fils chargé de traiter la demande (appel de fork). Lorsque le fils a terminé, il ferme le socket et meurt. Le serveur maître ferme quand à lui la copie du nouveau socket après avoir exécuté le fork. Il appelle ensuite de nouveau accept pour obtenir la demande suivante.

L'enchaînement des appels systèmes pour un serveur parallèle est présenté dans la figure 2.3

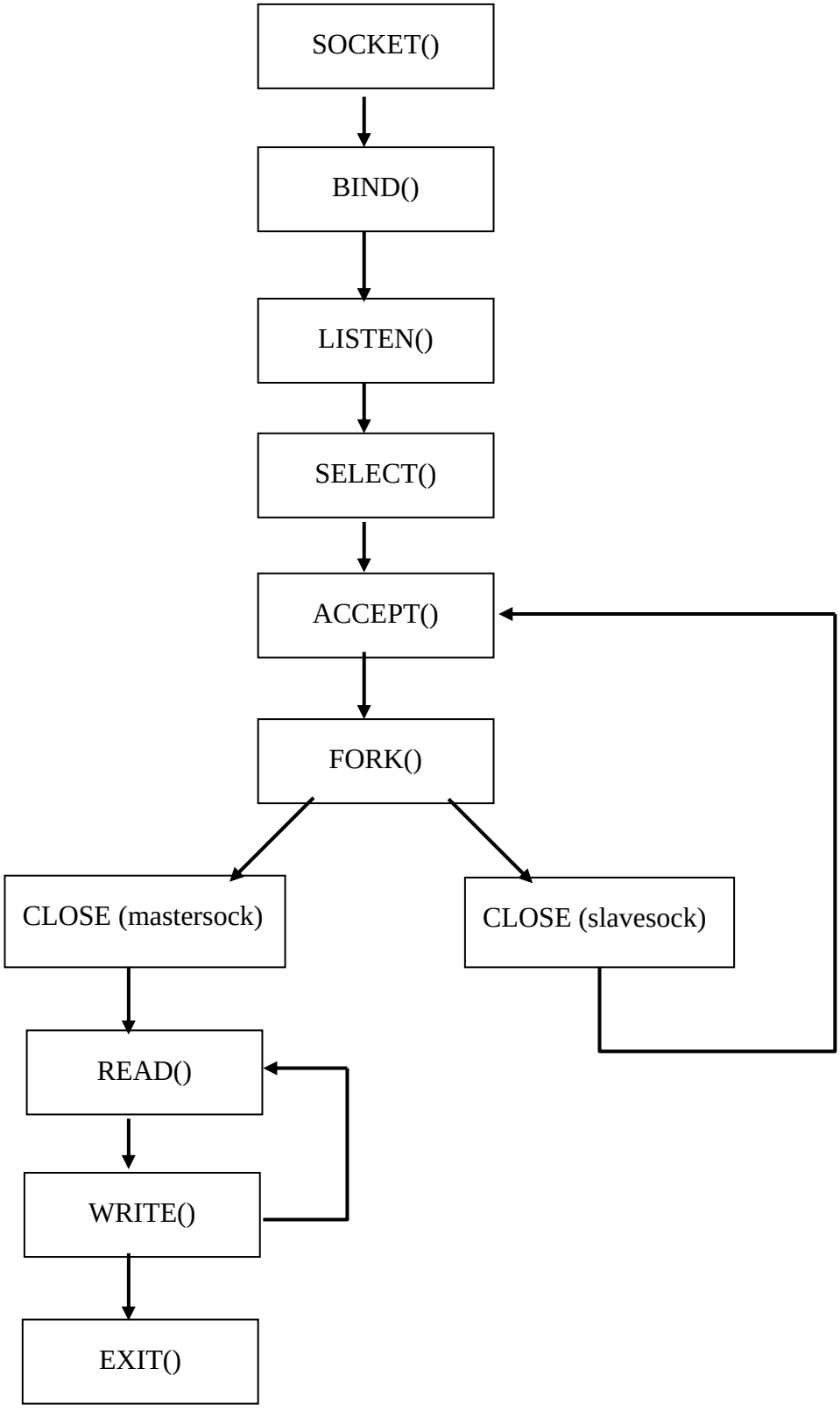


Figure 2.3 : L'enchaînement des appels système en mode connecté pour un serveur parallèle

2.6.2 - mode non connecté :

Dans le cas du protocole sans connexion UDP, les appels système sont différents. Le client n'établit pas de connexion avec le serveur. Le client envoi un datagramme au serveur en utilisant l'appel système sendto(). Symétriquement le serveur n'accepte pas de connexion d'un client, mais attend un datagramme d'un client avec l'appel système recvfrom(). Ce dernier revoie l'adresse réseau du client, avec le datagramme, pour que le serveur puisse répondre à la bonne adresse.

L'enchaînement dans le temps des appels systèmes pour une communication en mode non connecté et pour un serveur itératif est donné par la figure suivante:

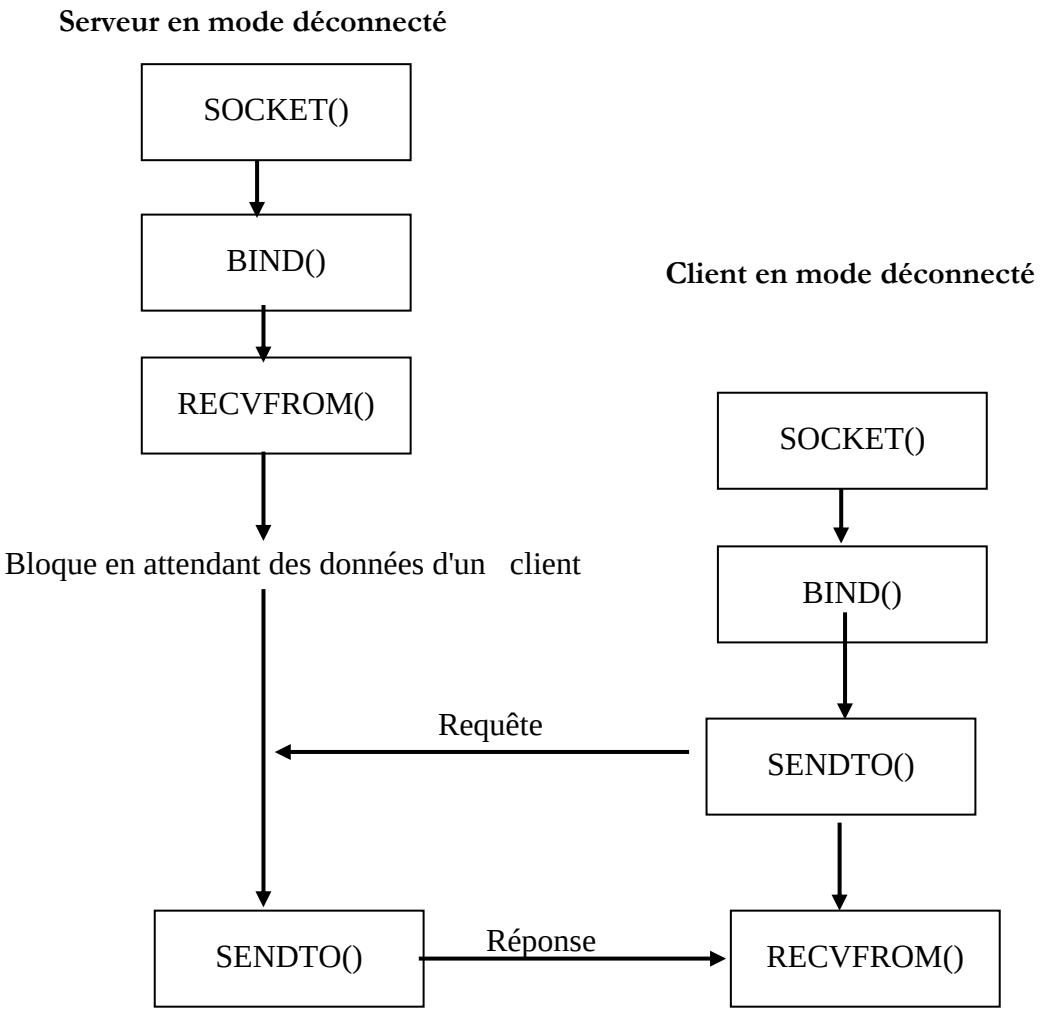


Figure 2.4: L'enchaînement des appels système en mode non connecté pour un serveur itératif

L’enchaînement dans le temps des appels systèmes pour une communication en mode non connecté et pour un serveur parallèle se résume par la figure suivante:

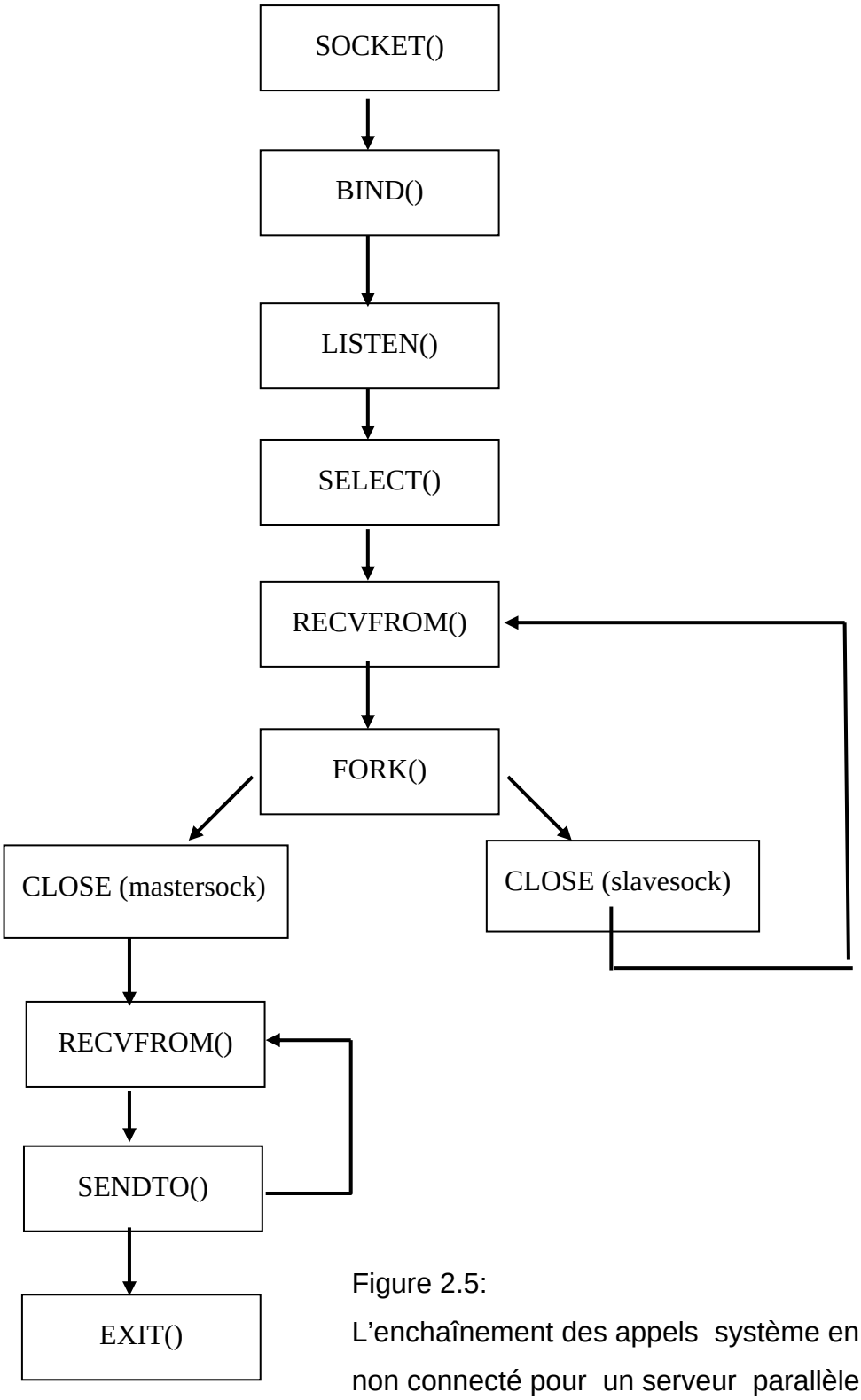


Figure 2.5:
L’enchaînement des appels système en mode non connecté pour un serveur parallèle

CHAPITRE 3

PRESENTATION DES DEUX CONTROLEURS PID ET GPC

3.1 - Introduction :

Un système de contrôle en temps réel d'un procédé industriel par ordinateur est un système qui doit répondre aux événements externes dans un temps spécifique. Le fonctionnement correct du système ne dépend pas seulement d'une logique correcte des calculs, mais aussi d'un respect des restrictions temporelles concernant son exécution sans quoi il y aurait une erreur. L'aspect temporel du contrôleur est présent à tous les niveaux fonctionnels. Par exemple, les routines d'asservissement doivent s'effectuer périodiquement en respectant des fréquences d'échantillonnage nécessaires pour éviter la discontinuité dans les commandes et assurer un bon comportement fonctionnel du contrôleur. **[5]**

Ce chapitre va décrire deux contrôleurs bien connus dans le domaine du contrôle à savoir le PID et le GPC. Des informations plus ou moins détaillées sur les aspects théoriques des deux algorithmes ainsi que les techniques de leurs implémentations numériques seront présentés.

3.2 - Le correcteur PID :

L'algorithme PID (proportionnel intégral dérivé) est le contrôleur le plus utilisé comme moyen d'asservissement dans le domaine industriel. Il a été employé avec succès pour plus de 50 ans. C'est un robuste algorithme facilement implémentable et fournit d'excellentes performances de contrôle pour une grande variété de procédés industriels.

Le schéma de commande d'un procédé industriel utilisant le correcteur PID prend généralement la forme présentée dans la figure 3.1 :

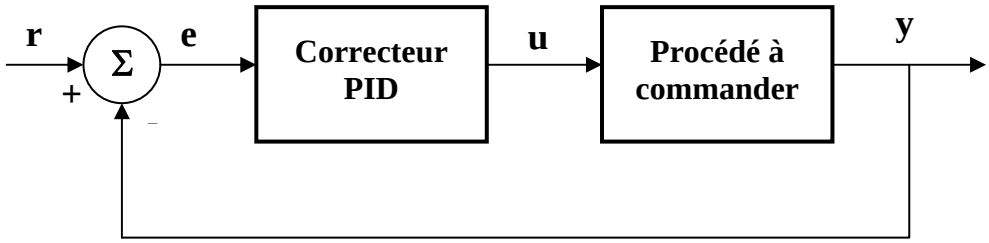


Figure 3.1 : Schéma de commande avec un correcteur PID

Avec :

- r : la consigne désirée
- y : le signal de sortie à commander ;
- e : l'erreur entre la consigne et le signal de sortie;
- u : le signal de commande du procédé.

3.2.1 - Description du correcteur :

Comme son nom l'indique, l'algorithme PID se caractérise par trois actions de base, l'action proportionnelle, l'action intégrale et l'action dérivée. Sous la forme continue, ce correcteur s'écrit :

$$u(t) = K_c e(t) + \frac{K_c}{T_i}\int_0^t e(t)dt + K_c T_d \frac{de(t)}{dt} \qquad (3.1)$$

L'action proportionnelle est essentielle au fonctionnement du PID. Elle permet essentiellement de donner de la puissance au signal de commande en agissant sur le coefficient K_c.

Plus K_c est grand, plus le système converge vite vers sa valeur finale. Mais en contrepartie, pour des valeurs de K_c trop grandes, le système oscille et génère des dépassements.

Le problème majeur avec l'action proportionnelle est que le signal de sortie n'atteint jamais la consigne désirée. C'est ce que l'on appelle l'erreur statique, elle

correspond à la différence entre le signal généré et le signal désiré en régime permanent. Pour compenser cette erreur statique, on rajoute le terme intégral.

Le correcteur intégral sert principalement à supprimer l'erreur statique. L'idée principale est de "charger" ou intégrer l'erreur depuis le début et d'ajouter cette erreur à la consigne jusqu'à ce qu'elle devienne nulle. Lorsque cette erreur est nulle, le terme intégral se stabilise et compense parfaitement l'erreur entre la consigne et le signal de sortie.

Bien que l'erreur statique s'est éliminé avec un correcteur PI, le système oscille encore et les dépassements sont plus grands. Le terme dérivé est préconisé pour permettre de diminuer considérablement ces dépassements. **[2], [12]**

3.2.2 - Implémentation du correcteur :

Pour une écriture discrète de notre correcteur présenté par l'équation (3.1), des approximations doivent être réalisées pour les termes intégral et dérivé sous une forme facilement implémentable sur un ordinateur.

On peut écrire :

$$\frac{de(t)}{dt} \approx \frac{e(t)-e(t-1)}{T_s} \tag{3.2}$$

et $$\int_0^t e(t)dt \approx T_s \sum_0^t e(i) \tag{3.3}$$

T_s est la période d'échantillonnage de l'algorithme qui doit être fixe pour avoir une meilleure régulation du procédé.

L'algorithme PID, sous sa forme discrète, s'écrit alors :

$$u(t) = K_c e(t) + \frac{K_c T_s}{T_i} \sum_{i=0}^{t} e(i) + \frac{K_c T_d \,(e(t)-e(t-1))}{T_s} \tag{3.4}$$

Par un décalage en arrière d'une période d'échantillonnage, l'équation (3.4) s'écrira :

$$u(t-1) = k_c e(t-1) + \frac{k_c T_s}{T_i} \sum_{i=0}^{t-1} e(i) + \frac{k_c T_d \,(e(t-1)-e(t-2))}{T_s} \tag{3.5}$$

La soustraction de l'équation (3.5) de (3.4) nous donne :

$$u(t) = u(t-1) + K_c\left[e(t) - e(t-1)\right] + \frac{k_c T_s}{T_t} e(t) + \frac{k_c T_d}{T_s}\left[e(t) - 2e(t-1) + e(t-2)\right] \qquad (3.6)$$

Une forme finale facile à programmer est donnée par :

$$u(t) = u(t-1) + \left[K_c + \frac{K_c T_s}{T_i} + \frac{K_c T_d}{T_s}\right] e(t) - \left[K_c + \frac{2K_c T_d}{T_s}\right] e(t-1) + \frac{K_c T_d}{T_s} e(t-2) \qquad (3.7)$$

3.2.3 - Réglage des paramètres du correcteur : **[12]**

Le réglage d'un PID implique l'ajustement des paramètres K_c, T_i et T_d pour réaliser une réponse optimale du système. Bien qu'un ajustement par tâtonnement soit généralement le plus utilisé pour un choix optimal de ces coefficients, il existe plusieurs méthodes d'estimation. La plus populaire est la méthode dite Ziegler Nichols.

Cette méthode consiste en premier lieu à considérer le correcteur en mode proportionnel seul. On ajuste par la suite le coefficient K_c en commençant par de petites valeurs et on choisit une petite valeur de consigne. On augmente à chaque fois K_c par un facteur de deux jusqu'à ce que la réponse du système présente des oscillations. Finalement, on ajuste K_c jusqu'à produire une réponse avec des oscillations continues. Dans ces conditions, on relève la valeur de $K_c = K_0$ et on mesure la période des oscillations T_0.

La méthode de Ziegler Nichols suggère un choix des paramètres du correcteur selon la loi suivante :

Tableau 3.1 : Réglage des paramètres du correcteur PID par la méthode Ziegler Nichols.

	K_c	**T_i**	**T_d**
P	$K_0/2$		
PI	$K_0/2.2$	$T_0/1.2$	
PID	$K_0/1.7$	$T_0/2$	$T_0/8$

3.3 - Le correcteur GPC :

La Commande Prédictive Généralisée (GPC), introduite par D.W. Clarke, est une méthode relativement récente qui, par sa simplicité, suscite un réel intérêt dans le domaine industriel. Cette approche consiste à prendre en compte le comportement futur d'un système grâce à un modèle numérique de prédiction. Elle permet la minimisation d'une fonction de coût dépendant de l'intégrale de l'erreur au carré et de l'énergie de la commande. Cette fonction est minimisée sur un horizon fini appelé horizon de prédiction. **[13]**

3.3.1 - Mise en équation du système : **[14]**

Soit un système linéaire à une entrée et une sortie dont le comportement est modélisé par l'équation récursive suivante:

$$A\left(q^{-1}\right)y(t) = B(q^{-1})u(t-Te) + \frac{C(q^{-1})}{D(q^{-1})}\xi(t) \qquad (3.8)$$

où:

- $u(t)$ est l'entrée du système,
- $y(t)$ est la sortie du système,
- $\{\xi(t)\}$ est une séquence de variables aléatoires indépendantes telles que $E\{\xi(t) = 0\}$
- T_e est la période d'échantillonnage,
- q^{-1} est un opérateur de retard d'une période d'échantillonnage,
- A, B, C, D sont des polynômes en q^{-1} .

Les polynômes A, B, C, D sont respectivement d'ordre n_a, n_b, n_c, n_d:

$$\begin{aligned} A(q^{-1}) &= 1+a_1q^{-1}+...+a_{na}q^{-na} \\ B(q^{-1}) &= b_0q^{-d}+b_1q^{-d-1}+...+b_{nb}q^{-nb-d} \\ C(q^{-1}) &= 1+c_1q^{-1}+...+c_{nc}q^{-nc} \\ D(q^{-1}) &= 1+d_1q^{-1}+...+d_{nd}q^{-nd} \end{aligned} \qquad (3.9)$$

où d est un entier positif et dT_e est le retard pur du système.

Dans le cas du correcteur GPC, on fixe $D(q^{-1})= 1- q^{-1} = \Delta$. L'équation (3.8) peut donc être réécrite sous la forme suivante:

$$A\left(q^{-1}\right)\Delta y(t) = B(q^{-1})\Delta u(t - Te) + C(q^{-1})\xi(t) \qquad (3.10)$$

Avec:

$$\Delta y(t) = (1 - q^{-1})\, y(t) = y(t) - y(t - Te)$$
$$\Delta u(t) = (1 - q^{-1})u(t) = u(t) - u(t - Te)$$

L'équation (3.10) décrit donc l'évolution de la variation de la sortie Δy en fonction de la variation de l'entrée Δu entre deux instants d'échantillonnage successifs.

3.3.2 - La fonction de coût :

Le correcteur GPC est un correcteur prédictif. Son objectif est de minimiser l'erreur quadratique entre les prédictions de la sortie et les consignes futures. Cela signifie qu'il doit réaliser une estimation des prédictions des valeurs de la sortie *y* aux instants d'échantillonnage futurs en fonction des valeurs futures de l'entrée *u*. Soit *r* la grandeur de référence de l'asservissement dont la valeur est supposée connue non seulement à l'instant *t* présent mais également aux N_2 instants d'échantillonnage futurs. Le correcteur GPC génère, à chaque instant d'échantillonnage, N_u commandes Δu de manière à minimiser la fonction de coût suivante:

$$J(u,t) = E\left\{ \sum_{j=N_1}^{N_2} \left[y(t + jTe) - r(t + jTe)\right]^2 + \lambda \sum_{j=1}^{Nu} \left[\Delta u(t + (j-1)Te)\right]^2 \right\} \qquad (3.11)$$

avec : $Nu<N_2$ et $\Delta u(t+jTe)=0 \quad \forall j \geq Nu$

où N_1, N_2 et N_u sont des entiers strictement positifs et λ un scalaire positif. Ces termes sont définis comme suit:

- N_1 est l'horizon d'initialisation,
- N_2 est l'horizon de prédiction,
- N_u est l'horizon de commande,
- λ est la pondération de la commande.

On peut remarquer que la fonction $J(u,t)$ comprend deux termes. Le premier,

$$\sum_{j=N_1}^{N_2}\left[y(t+jTe)-r(t+jTe)\right]^2$$

est la somme des erreurs au carré entre les sorties et les signaux de référence futurs. Le second,

$$\sum_{j=1}^{Nu}\left[\Delta u(t+(j-1)Te)\right]^2$$

est un terme proportionnel à " l'énergie " fournie par la commande.

Ce dernier terme est pondéré parλ. Ainsi, plus λ est faible, et moins " l'énergie " de la commande est pénalisée dans la fonction de coût, et par conséquent, plus le correcteur GPC synthétisé est "énergique", et donc sa réponse est rapide. Ce terme permet d'éviter les signaux de commande trop importants pouvant saturer le système.

3.3.3 - Calcul des prédictions de la sortie :

Dans cette section, nous développerons le calcul de $\hat{y}(t+jTe)$, la prédiction à l'instant t de y à j pas d'échantillonnage en avance. Ce calcul nécessite la résolution de 2 équations diophantiennes.

Afin d'alléger les notations, un polynôme $X(q^{-1})$ sera simplement noté X dans cette section.

Soient E_j et F_j les solutions de l'équation Diophantienne suivante:

$$C = A\Delta E_j + q^{-j}F_j \qquad (3.12)$$

avec:

$$E_j = 1+e_1^{(j)}q^{-1}+\ldots+e_{j-1}^{(j)}q^{-j+1} \qquad (3.13)$$

$$F_j = f_0^{(j)}+f_1^{(j)}q^{-1}+\ldots+f_{nf}^{(j)}q^{-nf}$$

et $nf(j)=\max(na,nc-j)$. La résolution des équations (3.12) de manière récursive est décrite en annexe B.

En réécrivant (3.10) pour le *j*-ème pas d'échantillonnage, on obtient:

$$A\Delta y(t+jTe) = B\Delta u(t+(j-1)Te) + C\xi(t+jTe) \qquad (3.14)$$

En introduisant (3.12) dans cette équation, nous obtenons:

$$A\Delta y(t+jTe) = B\Delta u(t+(j-1)Te) + A\Delta E_j\xi(t+jTe) + F_j\xi(t) \qquad (3.15)$$

En isolant $\xi(t)$ dans (3.10) et en utilisant cette relation dans (3.15) on obtient après simplification par $A\Delta$:

$$y(t+jTe) = \frac{F_j}{C}y(t) + E_j\xi(t+jTe) + \frac{BEj}{C}\left(\frac{C}{AE_j}u(t+(j-1)Te) - \frac{F_j}{AE_j}u(t-Te)\right) \qquad (3.16)$$

Grâce à l'équation Diophantienne (3.12), nous pouvons écrire que:

$$\frac{C}{AE_j}u(t+(j-1)Te) - \frac{F_j}{AE_j}u(t-Te) \quad =$$

$$\Delta u(t+(j-1)Te) + q^{-j}\frac{F_j}{AE_j}u(t+(j-1)Te) - \frac{F_j}{AE_j}u(t-Te) \;=$$

$$= \Delta u(t+(j-1)Te) \qquad (3.17)$$

En combinant (3.16) et (3.17) nous obtenons finalement:

$$y(t+jTe) = \frac{F_j}{C}y(t) + \frac{E_JB}{C}\Delta u(t+(j-1)Te) + E_j\xi(t+jTe) \qquad (3.18)$$

Dans cette dernière équation, toutes les références à des valeurs de la perturbation pour des instants d'échantillonnage passés et présents ont été supprimées. Il ne subsiste donc plus que la combinaison linéaire $E_j\xi(t+jTe)$ de valeurs futures de la perturbation ou du bruit. Or ces valeurs sont par définition indépendantes de signaux mesurables à l'instant *t*. Il est donc clair que la prédiction optimale à l'instant *t* de $y(t+jT_e)$ notée $\hat{y}(t+jTe)$ est donnée par:

$$\hat{y}(t+jTe) = \frac{F_j}{C}y(t) + \frac{E_jB}{C}\Delta u(t+(j-1)Te) \qquad (3.19)$$

Dans cette équation (3.19), le terme $E_j B\Delta u(t+(j-1)Te)$ est une combinaison linéaire de valeurs de Δu à des instants d'échantillonnage compris entre les instants t-(nb+d)T_e et t+(j-1-d)T_e. Mais la fonction de coût (3.11), implique seulement les valeurs futures et la valeur présente de Δu. Aussi allons-nous chercher à séparer ces valeurs des valeurs passées dans l'équation (3.19).

Cette séparation peut être obtenue grâce à la résolution d'une seconde équation Diophantienne en G_j et H_j

$$E_jB=G_jC+q^{-j}H_j \qquad (3.20)$$

avec:

$$G_j = g_0^{(j)}+g_1^{(j)}q^{-1}+...+g_{j-1}^{(j)}q^{-j+1} \qquad (3.21)$$

$$H_j = h_0^{(j)}+h_1^{(j)}q^{-1}+...+h_{nh}^{(j)}q^{-nh}$$

et $nh=max(nc,nb+d-1)$. Les équations Diophantiennes (3.20) peuvent être résolues récursivement. Ce calcul est proposé en annexe B.

L'utilisation de (3.20) dans (3.19) conduit à:

$$\hat{y}(t+jTe)=\frac{F_j}{C}y(t)+G_j\Delta u(t+(j-1)Te)+\frac{H_j}{C}\Delta u(t-Te) \qquad (3.22)$$

3.3.4 - Calcul de la solution optimale :

Soit $f_p \in \Re^{N_2-N_1+1}$, un vecteur constitué de la prédiction de la réponse libre du système pour les instants d'échantillonnage $\geq t+N_1Te$ (horizon d'initialisation) et $\leq t+N_2Te$ (horizon de prédiction):

$$f_p = \begin{bmatrix} \frac{H_{N_1}}{C}\Delta u(t-Te) + \frac{F_{N_1}}{C}y(t) \\ \frac{H_{N_1+1}}{C}\Delta u(t-Te) + \frac{F_{N_1+1}}{C}y(t) \\ \vdots \\ \frac{H_{N_2}}{C}\Delta u(t-Te) + \frac{F_{N_2}}{C}y(t) \end{bmatrix} \qquad (3.23)$$

Soit $\tilde{u} \in \Re^{N_u}$, le vecteur contenant les incréments de *u* sachant que $\Delta u(t + jTe) = 0$ $\forall j \geq N_u$

$$\tilde{u} = \left[\Delta u(t), \Delta u(t+Te), \ldots, \Delta u(t+(N_u-1)Te)\right]^T \qquad (3.24)$$

Soit $\hat{Y} \in \Re^{N_2-N_1+1}$, le vecteur contenant les prédictions des sorties à partir de l'horizon d'initialisation:

$$\hat{Y} = \left[\hat{y}(t+N_1Te), \hat{y}(t+(N_1+1)Te), \ldots, \hat{y}(t+N_2Te)\right] \qquad (3.25)$$

On peut donc réécrire l'équation (3.22) pour $N_1 \leq j \leq N_2$ sous forme matricielle:

$$\hat{Y} = G\tilde{u} + f_p \qquad (3.26)$$

avec $G \in \Re^{(N_2-N_1+1)XN_u}$ défini comme suit:

$$G = \begin{bmatrix} g_{N_1-1} & \cdots & g_0 & \cdots & \cdots & 0 \\ g_{N_1} & g_{N_1-1} & \cdots & g_0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ g_{N_u-1} & g_{Nu-2} & g_{N_u-3} & \cdots & \cdots & g_0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ g_{N_2-1} & g_{N2-2} & g_{N_2-3} & \cdots & g_{N_1-N_u+1} & g_{N_2-N_u} \end{bmatrix} \qquad (3.27)$$

Les termes g_k avec $0 \leq k \leq N_2 - 1$ sont les coefficients du polynôme G_j (voir équation (3.21), solution de l'équation Diophantienne (3.20).

La fonction de coût quadratique (3.11) devient donc:

$$J = (\hat{Y} - R)^T (\hat{Y} - R) + \lambda \tilde{u}^T \tilde{u} \qquad (3.28)$$

où $R \in \Re^{(N_2 - N_1 + 1)}$ est un vecteur contenant les signaux de référence futurs à partir de l'horizon d'initialisation:

$$R=[r(t+N_1T_e),...,r(t+N_2T_e)]^T \qquad (3.29)$$

La valeur minimale de J au sens des moindres carrés est donc obtenue pour $\tilde{u} = \tilde{u}^*$ avec:

$$\tilde{u}^* = (G^T G + \lambda I)^{-1} G^T (R - f_p) = K(R - f_p) \qquad (3.30)$$

La matrice $K \in \Re^{N_u x (N_2 - N_1 + 1)}$ est la matrice de gain optimal du correcteur GPC.

3.3.5 - Implémentation du correcteur GPC : **[14]**, **[15]**

L'équation (3.30) permet d'obtenir la valeur présente et les valeurs futures optimales pour les incréments de la commande jusqu'à l'horizon de commande. Mais seule la première valeur $\tilde{u}_1^* = \Delta u(t)^*$ de $\tilde{u}^*$ est utilisée. A l'instant d'échantillonnage suivant $t+T_e$, la commande optimale $\tilde{u}^*$ est recalculée. La commande $u(t)$ est donc générée de la manière suivante:

$$u(t) = u(t - Te) + \Delta u(t)^* \qquad (3.31)$$

Le schéma de commande avec un correcteur GPC est donc le suivant :

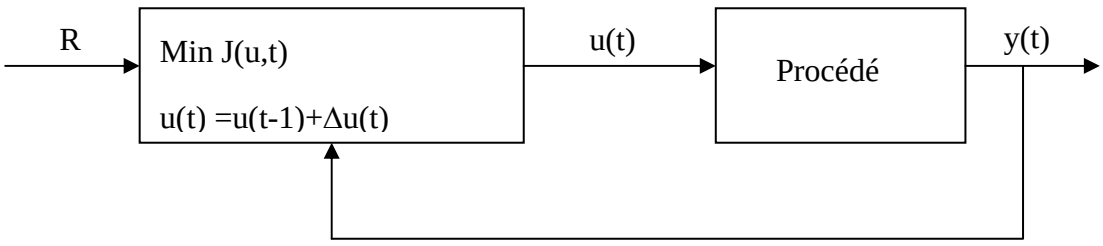


Figure 3.2 : Schéma de commande d'un procédé par un correcteur GPC

Avec:

$$\begin{aligned}\tilde{u}_1^* = \Delta u(t)^* = K_{11}\left[r(t+N_1Te) - (H_{N_1}\Delta u_c(t-Te) + F_{N_1}y_c(t))\right] \\ + K_{12}\left[r(t+(N_1+1)Te) - (H_{N_1+1}\Delta u_c(t-Te) + F_{N_1+1}y_c(t))\right] \\ \vdots \\ + K_{1(N_2-N_1+1)}\left[r(t+N_2Te) - (H_{N_2}\Delta u_c(t-Te) + F_{N_2}y_c(t))\right]\end{aligned} \quad (3.32)$$

tels que $C\Delta u_c(t-Te) = \Delta u(t-Te)$ et $Cy_c(t)=y(t)$. Ces termes peuvent être calculés grâce aux équations récursives suivantes:

$$\Delta u_c(t-Te) = u(t-Te) + (1-C)\Delta u_c(t-Te) \quad (3.33)$$

$$y_c(t) = y(t) - (1-C)y_c(t)$$

Les k_{1j} sont les éléments de la première ligne de la matrice de gain optimal K.

On peut remarquer que si le modèle est invariant, un seul calcul préalable du gain K est nécessaire. Ce calcul peut être fait au moment de l'initialisation de l'asservissement. Le calcul de $u(t)$ est donc une simple équation récursive dont les variables sont les mesures et les incréments de commande passés filtrés par $\frac{1}{C}$ ainsi que les consignes futures.

Le GPC est également adapté aux systèmes dont le modèle varie avec le temps. Dans ce cas, le gain K doit être réactualisé (à chaque pas d'échantillonnage ou quand cela s'avère nécessaire). Le calcul de K comprend, entre autres, une inversion d'une matrice N_u x N_u (voir équation (3.30)). Si on envisage une implémentation temps réel d'un tel calcul, il est donc nécessaire de limiter l'horizon de commande à une valeur raisonnable.

3.3.6 - Réglage du correcteur GPC : **[15]**

La synthèse du correcteur GPC dépend de 4 paramètres de réglage:

- l'horizon d'initialisation N_1,
- l'horizon de prédiction N_2,
- l'horizon de commande N_u,

- et la pondération de la commande λ .

a- Influence de N_1 :

La modification de l'horizon d'initialisation permet d'ajuster la fenêtre d'optimisation à une zone précise de la réponse. On peut ainsi choisir de ne pas faire intervenir les premiers échantillons dans l'optimisation lorsque, par exemple, le système contient plusieurs retards purs. En pratique, on prendra généralement $N_1=d+1$.

b- Influence de N_u :

On choisit en général un horizon de commande N_u relativement réduit. Ce choix est entièrement justifié si le signal de référence de la boucle ne varie pas. Dans ce cas, seules les perturbations, qui par nature sont aléatoires, sont susceptibles de modifier la sortie du système. Il n'est donc pas nécessaire de calculer un nombre important de commandes à l'avance étant donné que seule la première est effectivement appliquée au système et qu'aucune prédiction ne peut être faite sur les perturbations.

Par contre, si le signal de référence n'est pas constant, le fait de ne faire intervenir dans l'optimisation qu'un nombre limité de commandes par rapport à l'horizon de prédiction N_2 a des effets indésirables sur la réponse. Ainsi, lorsque on applique un échelon sur le système et que cet échelon commence à entrer dans la zone supérieure de la fenêtre d'optimisation (c'est à dire que l'instant d'occurrence de la prédiction de l'échelon est $t+N_2T_e$) , si $N_u<<N_2$, il est impossible d'atteindre la référence à l'instant $t+N_2T_e$ en seulement N_u variations de la consigne puisque $\Delta u(t) = 0$ si $t \geq N_u Te$.

L'erreur quadratique $\sum_{j=N_1}^{N_2} \left[y(t + jTe) - r(t + jTe) \right]^2$ sera donc d'autant plus grande que l'instant prévu de l'échantillon est éloigné de l'instant $t+N_uTe$ de la dernière variation de la commande.

Il faut donc conserver un horizon de commande N_u en rapport avec l'horizon de prédiction N_2. Par contre, pour le rejet de perturbation, l'augmentation

de l'horizon de prédiction par rapport à l'horizon de commande n'a pas d'influence.

c- Influence de N_2 :

On choisi en général N_2 tel que N_2T_e soit de l'ordre de grandeur de la constante de temps dominante du système en boucle fermée. Ainsi, la fenêtre d'optimisation contient toute la réponse du système.

d- Influence de λ :

De tous les paramètres de réglage du GPC, λ est celui dont l'influence est la plus évidente. En effet, il permet de pondérer l'influence des commandes dans l'optimisation et ainsi permet de générer un correcteur plus ou moins "énergique" donc plus ou moins rapide.

CHAPITRE 4

MODELE D'UN SYSTEME DE CONTROLE EN TEMPS REEL PILOTE À DISTANCE

4.1 – Introduction :

Dans ce chapitre nous proposons un modèle de contrôle en temps réel d'un procédé industriel. Il s'agit d'un système basé sur une architecture client/serveur qui permet à un opérateur distant de contrôler et surveiller à distance un processus réel en activant des tâches temps réel de contrôle et d'acquisition de données.

Le serveur est implémenté sous une plate-forme temps réel, Il s'agit du système d'exploitation RTLinux, décrit au chapitre 1. Ce système d'exploitation a pour mission de contrôler notre procédé en respectant des contraintes de temps spécifiques et doit élaborer, à partir des informations acquises, des informations de commande ou de contrôle dans un temps cohérent avec l'évolution de notre procédé contrôlé. Le serveur, dans notre système de contrôle, dirige toutes les tâches temps réel du procédé et communique avec le client via un canal de communication sous réseau.

Le client permet une manipulation à distance de notre procédé et offre à l'opérateur une interface utilisateur de supervision, Ce client doit fournir à l'utilisateur plusieurs fonctions tels que la gestion de connexion et le transfert de messages avec le serveur, la sélection et l'envoi des commandes et des paramètres de contrôle des tâches temps réel, la réception des informations de ces tâches et la visualisation des paramètres principaux et enfin la sauvegarde des paramètres jugés utiles.

Les deux unités, client et serveur, sont liées à travers une liaison réseau point à point en utilisant le mécanisme socket TCP/IP. Cette liaison permet un transfert rapide de données et une procédure de communication très souple et offre la possibilité de partager les données du procédé à contrôler sur tout un réseau local.

La figure 4.1 suivante donne une schématisation générale de notre modèle de contrôle.

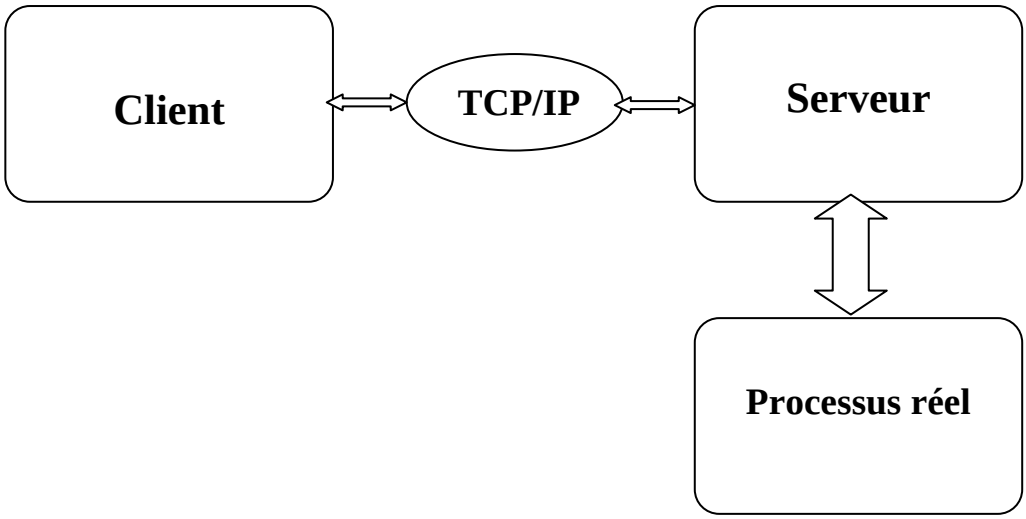


Figure 4.1 : Structure générale du modèle de contrôle proposé.

4.2 - La structure du serveur :

Le serveur, implémenté sous RTLinux, est structuré en deux modules principaux comme décrit dans la figure suivante :

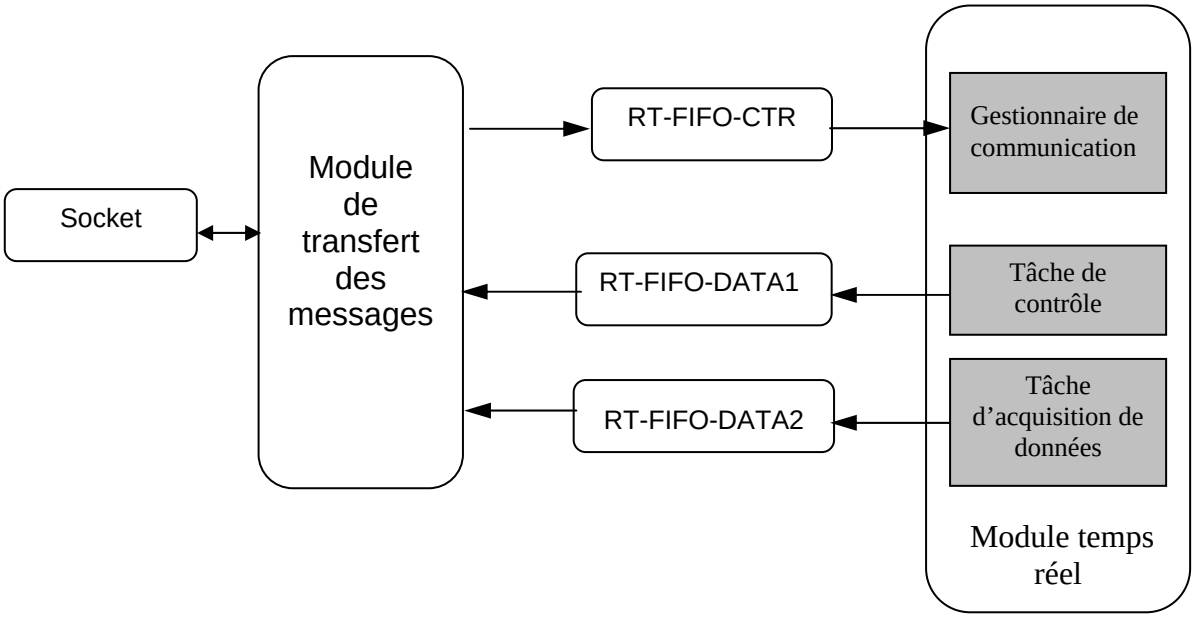


Figure 4.2 : Structure d'implémentation du serveur

***Module de transfert de messages* :** c'est un processus linux non RT conçu pour mettre le client distant en communication avec le module temps réel. Ce processus communique avec le module temps réel à travers des canaux de communications appelés RT_Fifos. La communication de ce module avec le client distant se fait via une liaison réseau TCP/IP en utilisant le mécanisme socket.

***Module temps réel* :** il gère l'exécution des tâches temps réel de contrôle et d'acquisition de données à travers un algorithme d'ordonnancement à priorités fixes. Ce Module bénéficie de tous les privilèges fournis par le système d'exploitation en temps réel.

Etant donné que les tâches temps réel dans notre modèle sont périodiques, l'algorithme d'ordonnancement que nous avons implémenté est à priorités fixes, basé sur l'algorithme Rate Monotonic (RM). Ainsi, cet ordonnanceur octroie le processeur à la tâche de plus petite période. Quant aux tâches non RT, l'ordonnanceur utilisé par défaut par le système est de type Round Robin (RR). Pour ce dernier ordonnanceur, l'exécution de ces tâches s'effectue à tours de rôle.

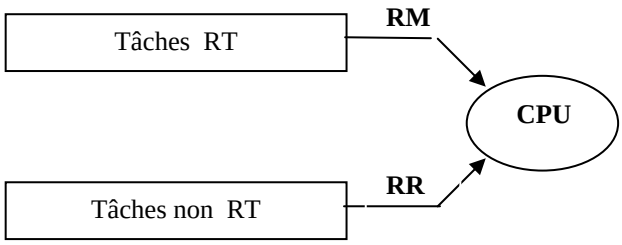


Figure 4.3 : Les type d'ordonnancement associés aux tâches du serveur.

Les tâches non RT sont vues par l'ordonnanceur temps réel comme des processus de la plus basse priorité relativement aux tâches RT. Dans notre situation, il est nécessaire de garantir une marge temporelle suffisante à l'exécution du module de transfert de messages, qui est une tâche non RT, pour permettre ainsi au client, d'entrer en communication avec le module RT.

4.2.1 - Module de transfert de messages :

Le module de transfert de messages, qui est un processus non RT, est le module du serveur en charge de la communication avec le client. Il permet à l'opérateur éloigné d'effectuer une communication bidirectionnelle avec le module

temps réel de contrôle et d'acquisition de données par l'envoi et la réception de messages.

Une fois la connexion établie entre le serveur et le client, ce module s'apparente alors à un pont entre deux canaux de communication : le mécanisme socket TCP/IP qui communique avec le client distant, et les 3 piles RT_fifos qui communiquent avec le module temps réel.

En pratique, quand un message parvient à l'un des canaux de communication, et après les vérifications nécessaires, il est immédiatement transmis vers l'autre canal pour permettre le maintien de l'interactivité entre tous les modules de l'application.

Notre module utilise deux processus dans son exécution (processus Père et processus Fils). Quand une demande de connexion par un client distant est acceptée, le processus père ferme le socket principal et crée un autre processus fils avec un nouveau socket ; ainsi, cette procédure coupe le chemin de connexion vers le serveur à d'autres clients. Le socket principal sera réouvert une fois que le processus fils termine sa session.

Quand on veut terminer une session de connexion, et avant de mettre fin au processus fils, la commande *QUIT* doit être envoyée à toutes les taches RT du module temps réel pour les suspendre. Le processus père se réapproprie alors sa fonction d'écoute ; il reste ainsi en attente d'une nouvelle demande de connexion par le client.

La communication avec le module RT s'effectue à travers trois canaux unidirectionnels (*RT_fifo-ctr, RT_fifo_data1* et *RT_fifo_data2*). Ces fichiers spéciaux sont vus et utilisés, coté module de transfert de messages, à l'aide des appels systèmes classiques de gestion des fichiers sous Linux. Du coté module temps réel, une bibliothèque de fonctions est proposée pour permettre l'envoi et la réception de données par ce biais.

- Le *RT_fifo_ctr* : ce fichier est utilisé, une fois établie la connexion entre le client et le serveur, pour acheminer vers le module temps réel le message de contrôle transmis par le client distant. Ce message comprend les ordres de sélection, d'activation et d'arrêts des tâches RT, outre l'envoi des

paramètres de configuration des tâches (période d'exécution, paramètres de régulation, ...).

- Les deux *RT_fifo_data :* chacun de ces deux fichiers reçoit les informations transmises par la tâche RT qui lui est associée. Ces informations seront récupérées par le module de transfert de messages et envoyées, via le mécanisme socket, vers le client.

La figure 4.4 donne une schématisation générale des principales opérations effectuées par le module de transfert de messages :

4.2.2 - Module temps réel :

Le module temps réel est la partie du serveur s'exécutant sous un environnement temps réel. Il dirige nos tâches RT pour le contrôle et l'acquisition de données en respectant rigoureusement les contraintes temporelles de notre application, principalement le respect des périodes d'exécution des tâches.

Ce module est composé :

- Des corps des tâches temps réel ;
- d'un manipulateur d'attente sur les FIFO ;
- des routines d'initialisation et d'arrêt du module temps réel ;

4.2.2.1 - Les tâches temps réel :

Dans notre application, on dispose de deux tâches temps réel indépendantes. Ces tâches doivent être exécutées de manière périodique, voire strictement périodique, et ce, afin d'avoir une bonne transmission de l'information entre le système numérique et le procédé physique.

La première tâche est chargée d'exécuter un algorithme de commande en boucle fermée pour le contrôle de la vitesse d'un moteur à courant continu. Les algorithmes de commande implémentés pour notre application sont le correcteur PID classique et le correcteur prédictif généralisé (GPC).

La deuxième tâche est un processus d'acquisition de données. On attribue à cette tâche la priorité maximale pour permettre sa préemption sur toutes les autres tâches.

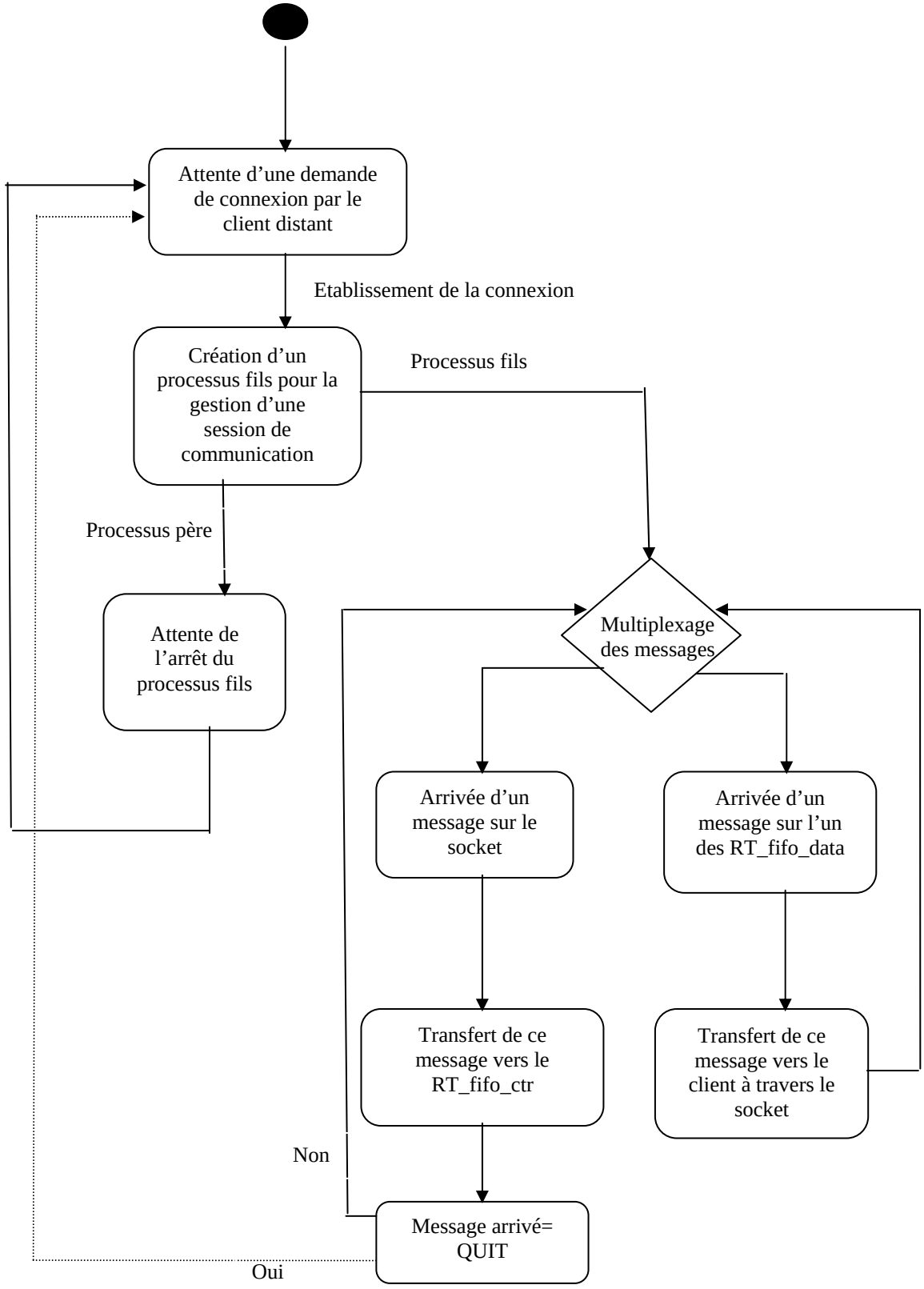


Figure 4.4 : Les principales opérations effectuées par le module de transfert des messages.

4.2.2.2 - Le manipulateur d'attente sur les FIFO :

Quand un message de contrôle, envoyé par le client distant, est reçu par le module de transfert de messages à travers le socket TCP/IP, il sera transmis vers le module temps réel à travers le fichier de communication RT_fifo_ctr. L'écriture sur ce FIFO entraîne l'exécution de la fonction d'écoute au niveau du module temps réel. Les différentes opérations réalisées par cette fonction sont présentées dans la figure suivante :

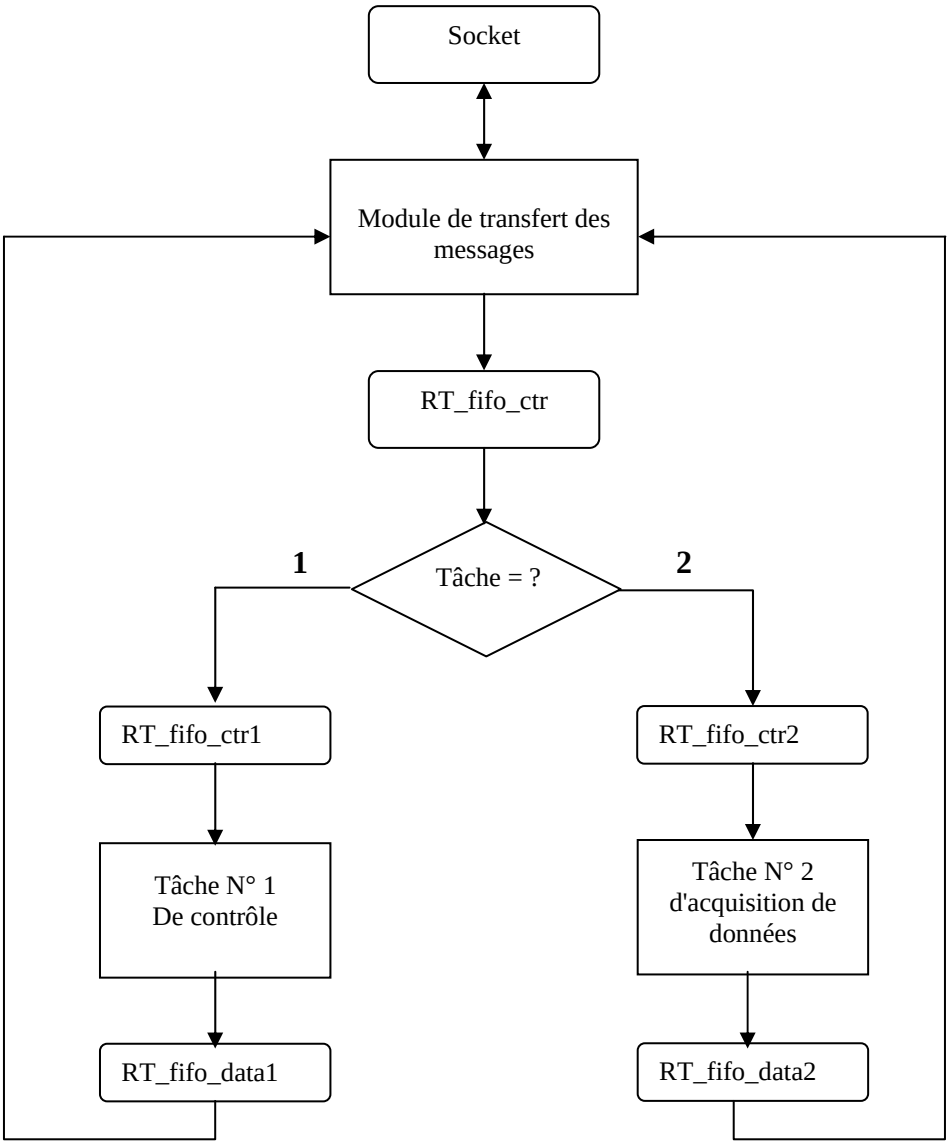


Figure 4.5 : Schéma d'acheminement des messages dans le serveur

Une fois ce message de contrôle détecté, il sera orienté, vers la tâche ciblée par le client à travers l'un des deux RT_fifos de contrôle (RT_fifo_ctr1 et RT_fifo_ctr2 associés aux deux tâches RT).

Quant aux messages de données qui comprennent les informations que désire envoyer les tâches temps réel vers le client distant, ils seront transmis directement à travers les deux RT_fifo_data vers le module de transfert de messages. Ce dernier les envoie vers le client à travers le réseau à l'aide du socket TCP/IP.

4.2.2.3 - Routines d'initialisation et d'arrêt de module temps réel :

Le module temps réel possède deux fonctions standards qui organisent son exécution. La première fonction d'initialisation, appelée lors du chargement du module, contient les ordres de création des tâches temps réel avec les attributs nécessaires. Elle contient aussi les ordres de création des différents canaux de communications RT-FIFOs utilisés pour l'acheminement des données entre le module temps réel et le module de transfert de message. Cette fonction contient encore les déclarations des différentes variables globales utilisées par l'ensemble des tâches temps réel.

La fonction d'arrêt du module temps réel est la dernière fonction exécutée par ce module. Elle permet de libérer toutes les ressources système qui étaient occupés par les tâches temps réel.

4.3 - La structure du client :

Le client représente, comme nous l'avons mentionné, l'interface utilisateur qui permet à un opérateur d'effectuer un contrôle-commande à distance de notre système de contrôle. Cette interface doit fournir une certaine convivialité de présentation et une souplesse dans son élaboration.

Ce client est chargé de :

- l'établissement de la connexion avec le serveur;
- la sélection des tâches temps réel ;

- l'envoi des commandes d’activation et d’arrêt des tâches temps réel du système éloigné ;
- l'envoi des paramètres de configuration des tâches ;
- la réception des informations des tâches temps réel ;
- la visualisation des paramètres principaux;
- l'enregistrement des paramètres du procédé;
- la fermeture de la connexion avec le serveur.

CHAPITRE 5

IMPLEMENTATION DU MODELE DE CONTROLE

5.1 – Introduction :

Afin de mettre en évidence notre modèle de contrôle en temps réel, décrit au chapitre précédent, nous présentons dans cette partie une implémentation réelle de ce modèle par l'exposition des aspects de programmation des différents composants du système.

Pour une meilleure démonstration du modèle, nous avons utilisé, comme moyen expérimental, un banc expérimental composé d'une part d'un moteur à courant continu avec toutes les unités associées (module de commande, carte d'interface avec le PC, tachymètre…) et d'autre part une carte d'acquisition universel avec un générateur de signaux programmés (Voir figure 5.1)

Dans ce chapitre nous montrerons aussi une procédure d'évaluation des paramètres temporels des tâches du système pour assurer leur ordonnançabilité. Et enfin nous ferons une synthèse des deux correcteurs PID et GPC implémentés pour la tâche temps réel de régulation.

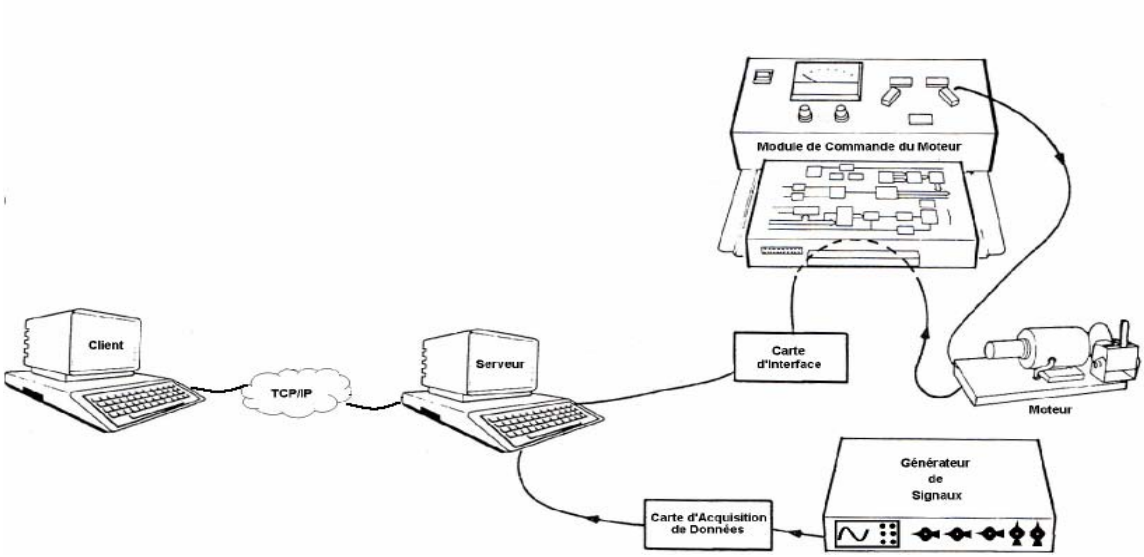


Figure 5.1 : Présentation du banc expérimental

5.2 - Implémentation du serveur :

Le système de fichiers sous RTLinux, dédié à notre serveur, est composé de deux répertoires, *Mod-mes* et *Mod-rt* (figure 5.1). Ces répertoires reflètent l'organisation structurelle du serveur qui est basé sur deux principaux modules : le premier est chargé de la gestion de la communication entre le serveur et le client. L'autre module dirige les tâches temps réel pour le contrôle et l'acquisition de données.

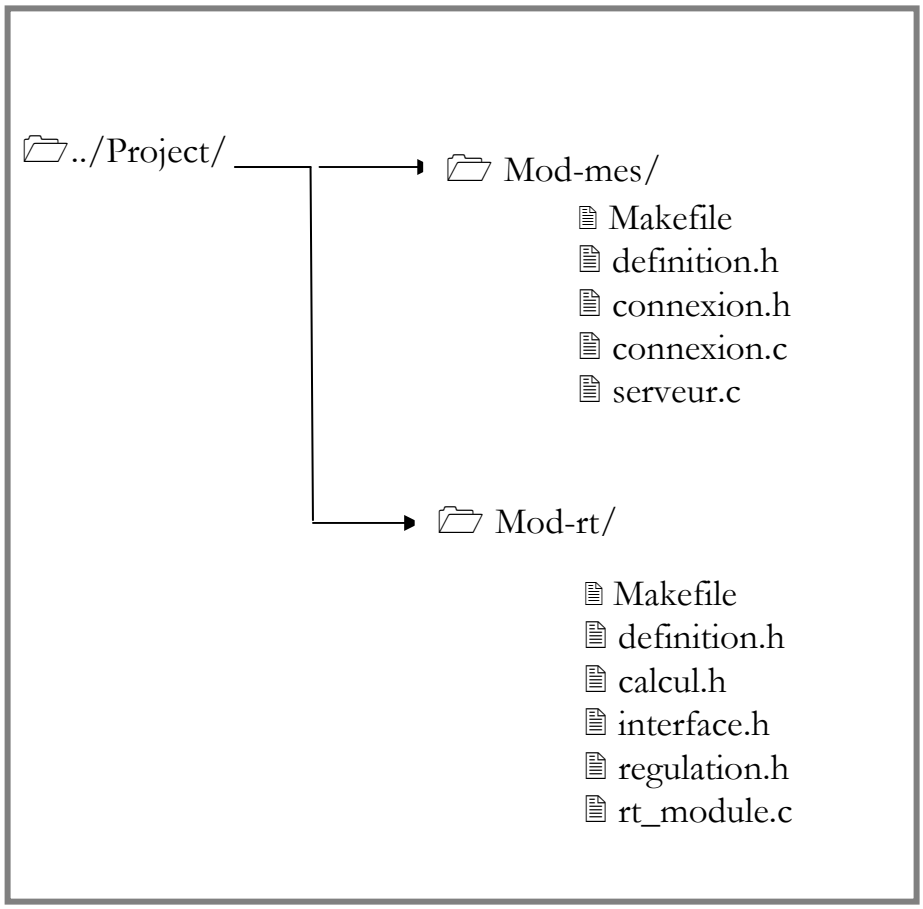


Figure 5.2 : Description du système de fichiers de l'application

Le fichier *definition.h*, commun pour les deux modules, contient les définitions des constantes et les structures relatives au protocole de communication. Il contient en particulier la structure de message de contrôle servant à la réception des requêtes provenant du client et la structure des données permettant l'encapsulation et l'envoi des données des tâches RT vers le client.

5.2.1 - Le répertoire Mod-mes :

Le répertoire *Mod-mes* contient les fichiers d'implémentation du module de gestion de connexion et de transfert de messages.

- *Connexion*.c : contient les appels système d'initialisation de la communication sous réseau.
- *Serveur*.c : ce fichier constitue le pivot de notre application. Il contient le code complet pour la création du canal de communication par le mécanisme socket et pour l'établissement de la connexion avec le client distant

Quand le serveur reçoit une demande de connexion du client, et une fois acceptée, il crée un processus fils qui a pour mission l'écoute et la direction des messages de communication entre le socket et les RT-Fifos.

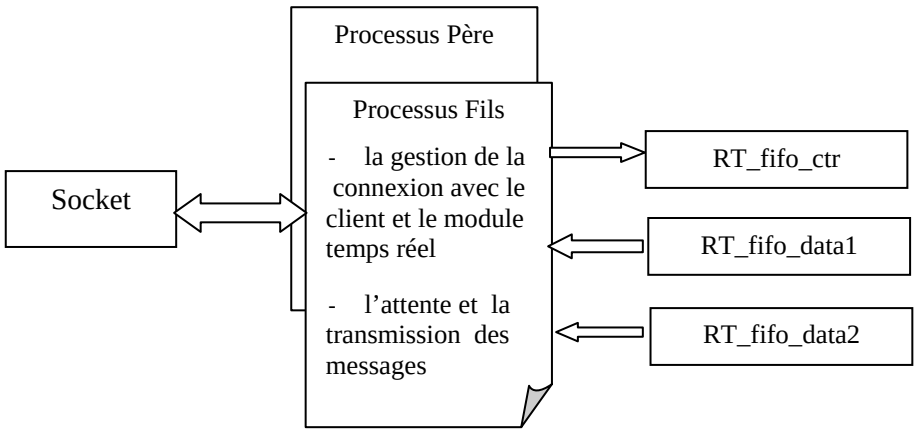


Figure 5.3 : Procédure d'écoute sur les canaux de transmission

L'attente des messages sur tous les canaux de transmission est réalisée par l'appel système *select()* comme décrit dans le code de la figure 5.4 :

```
retval = select(FD_SETSIZE, &rfds, NULL, NULL, &tv);
  if (retval > 0)
    {
      if (FD_ISSET(socket , &rfds))
        {
         recv(socket, &mesrx, sizeof(mesrx)) ;        /*  Réception du message de contrôle à partir du  */
         write(rt_fifo_ctr, &mesrx, sizeof(mesrx));   /*  socket et son envoi vers la pile RT_fifo_ctr    */
         }

      if (FD_ISSET(rt_fifo_data1 , &rfds))            /*    La  même chose sera pour rt_fifo_data2   */
        {
         read(rt_fifo_data1, &mestx, sizeof(mestx)) ;  /*   Réception du message de données à partir du  */
         write(socket, &mestx, sizeof(mestx)) ;         /*   RT_fifo_data1 (ou RT_fifo_data2 )  et  son     */
         }                                              /*    envoi vers le socket      */
     }
```

Figure 5.4 : l'appel select pour l'écoute et la transmission des messages.

5.2.2 - Le répertoire mod-rt :

Les principaux fichiers composant le module temps réel sont :

- *interface.h* : contient les fonctions de pilotage des deux cartes électroniques : Une carte d'acquisition type ISA comportant 16 entrées analogiques avec gain programmable et la carte d'interface ISA/XT pour la commande du banc d'essai du moteur à courant continu.

- *regulation.h* : contient l'implémentation numérique des algorithmes de commande en boucle fermée, à savoir : le correcteur PID et le correcteur prédictif généralisé GPC.

- *rt_module.c* : ce fichier contient les corps des tâches temps réel et les fonctions d'initialisation et d'arrêt de notre module temps réel.

5.3 - Compilation des programmes :

Afin de faciliter la compilation des modules de l'application, des fichiers appelés *Makefile* sont utilisés pour regrouper toutes les lignes des commandes et les options servant à la compilation des codes de programmes écrits. Notons que

le compilateur C utilisé par linux est la commande système *gcc*. La construction des modules se fait en exécutant la commande *Make.*

Le fichier *Makefile* contenu dans le répertoire *Mod-mes* et associé au module de transfert de messages est le suivant :

```
all : connexion.o serveur

include  rtl.mk
 FLAGS = -O2 -Wall

serveur : serveur.o connexion.o
        gcc  ${INCLUDE} ${FLAGS}   serveur.c
connexion.o -o serveur

serveur.o : serveur.c connexion.h
        gcc -c ${INCLUDE} ${FLAGS} serveur.c

connexion.o : connexion.c connexion.h
        gcc -c ${INCLUDE} ${FLAGS} connexion.c
```

Figure 5.5 : Le contenu du fichier Makefile associé au module de transfert de messages

Après l'exécution de ce Makefile, on obtient le fichier exécutable *serveur* basé sur les deux modules compilés *connexion.o* et *serveur.o.* Ce fichier *serveur* va permettre l'établissement de la connexion avec le client et la direction des messages.

Quant au fichier *Makefile* contenu dans le répertoire *Mod-rt*, il comporte deux parties : la première se charge de la compilation et la construction du module temps réel *rt_module.o* servant à la gestion et le pilotage des tâches temps réel. La deuxième partie contient toutes les lignes d'exécution de nos programmes d'application. Le fichier *Makefile* est représenté dans la figure 5.4 et son exécution est réalisée par la ligne de commande *Make project*

Après création du module rt_module.o, la première action réalisée par le processus rmrtl se charge de l'élimination de tous éventuels modules temps réel chargés auparavant. Par la suite, et à l'aide de processus insrtl, on procède à

l'installation des modules de base du noyau temps réel. Une fois cette installation réalisée, la ligne de commande insmod rt_module.o est exécutée pour permettre le chargement du module temps réel de l'application. La dernière action accomplie dans le fichier Makefile est l'exécution du processus serveur qui met en marche notre partie serveur.

```
all: rt_module.o

include rtl.mk

rt_module.o: rt_module.c
  gcc ${INCLUDE} ${CFLAGS}  -O2 -c rt_module.c -o rt_module.o


project : all
        -rmmod rt_module
        (cd /usr/src/rtlinux-2.3; ./rmrtl)
        @echo "Insertion des modules de gestion des FIFO "
        @echo " et de l'ordonnanceur"
        (cd /usr/src/rtlinux-2.3 ; ./insrtl)
        @echo " installation du module temps réel "
        @insmod rt_module.o
        @echo "Lancement de l'application"
        (cd /project/msg;./serveur )

clean : -rmmod rt_module
```

Figure 5.6 : Le contenu du fichier Makefile associé au module temps réel

5.4 - Structure des messages :

Etant donnée que la communication se fait entre deux systèmes d'exploitation hétérogènes (Linux & RTLinux pour le serveur et Windows pour le client), une attention particulière doit être donnée lors de la formulation des messages pour assurer une compatibilité dans la manipulation des données échangées. Pour garantir cet aspect de généralité, tous les paramètres inclus dans les messages sont regroupés dans une séquence de caractères au format ASCCI adopté par toutes les plateformes.

```
/*********************************************************/
/*          structure de message de contrôle             */
/*********************************************************/
struct msgrx
  {
        char   action ;
        char   tache[4];
        struct regulation
        {
                char   periode[4];
                char   p1[8] ;
                char   p2[8] ;
                char   p3[8] ;
                char   p4[8] ;
         } regul ;

        struct dataacq
         {
                char   periode[4];
                char   canal[4];
                char   gain[4];
         }data_acq;
  } ;

/*********************************************************/
/*              structure de message de données          */
/*********************************************************/

struct msgtx
   {
      char   tache[4];
      char   nbre[4];
      char   data1[8];
      char   data2[8];
      char   data3[8];
   };
```

Figure 5.7 : structure des messages de contrôle et de données

Le message de contrôle émanant du client distant se regroupe dans la structure *msgrx.* Le premier paramètre *action,* inclus dans le message, indique le type d'action à appliquer à la tâche temps réel. Les différents états que peut prendre ce paramètre sont définis dans le fichier *definition.h* :

```
#define    Start_task        'S'
#define    Stop_task         's'
#define    Quit              'q'
```

Le deuxième élément de message est un entier formaté sur 4 caractères. Il indique le numéro de la tâche temps réel ciblée par l'action. Il prend la valeur « 1 » pour la tâche de régulation et la valeur « 2 » pour la tâche d'acquisition de données.

Les deux sous structures incluses dans le message de contrôle contiennent tous les paramètres propres aux deux tâches temps réel disponibles dans l'application.

La première sous structure, nommée *regulation,* reçoit les paramètres de l'un des deux algorithmes de correction que nous avons étudiés (le régulateur PID ou le correcteur GPC).
En plus de l'élément *periode* qui indique la périodicité d'exécution de la tâche de régulation, les 4 autres paramètres Pi de cette sous structure correspondent aux coefficients du régulateur choisi (la consigne et les 3 facteurs P, Td, Ti pour le correcteur PID ou bien la consigne, N1, N2 et λ pour le correcteur GPC).

La seconde sous structure appelée *dataacq* est destinée à la deuxième tâche temps réel pour l'acquisition de données. Les 3 facteurs inclus représentent respectivement la période d'exécution de la tâche, le numéro du canal d'entrée du signal analogique au niveau de la carte d'acquisition utilisée et enfin la valeur de gain d'amplification choisi pour ce signal d'entrée.

Le message de données, qui correspond aux informations émises par les deux tâches temps réel vers le client distant, est regroupé dans la structure *msgtx.* Il est constitué de 5 paramètres qui représentent respectivement le numéro de la tâche désirante d'envoyer des données, le nombre des messages envoyés et les 3 autres paramètres *Data* propres à la tâche émettrice de message.

5.5 - Fonctions d'initialisation et d'arrêt du module RT :

Les deux fonctions d'initialisation et d'arrêt du module temps réel, fournies par le système d'exploitation RTLinux, sont respectivement init_module() et cleanup_module(). Ces deux fonctions permettent au programmeur d'inclure les commandes d'initialisation des paramètres et la création des tâches temps réel et

leur destruction à la fin d'exécution. Pour notre application, elles sont présentées dans le code suivant :

```
/*   Fonction d'initialisation du module temps réel  */
int init_module(void)
{
     int c[5];
     int ret,i;
     pthread_attr_t attr;
     struct sched_param sched_param;

      for(i=0 ;i<5 ;i++)
      rtf_destroy(i);
     c[0] = rtf_create(0, 1000);
     c[1] = rtf_create(1, 1000);
     c[2] = rtf_create(2, 1000);
     c[3] = rtf_create(3, 4000);
     c[4] = rtf_create(4, 4000);
     for(i=0 ;i<5 ;i++)
     printk("Fifo return fifo[%d] = %d \n ",i,c[i]) ;
     pthread_attr_init (&attr);
     sched_param.sched_priority = 2;
     pthread_attr_setschedparam (&attr, &sched_param);
     ret= pthread_create (&tasks[0], &attr, task_code0, (void *)1);
     pthread_attr_init (&attr);
     sched_param.sched_priority = 1;
     pthread_attr_setschedparam (&attr, &sched_param);
     ret= pthread_create (&tasks[1], &attr, task_code1, (void *)2);
     rtf_create_handler(2, &my_handler);
     printk("Le module temps réel est installé \n");
     return 0;
}

/*  Fonction d'arrêt du module temps réel */

void cleanup_module(void)
   {
      int i ;
      for(i=0 ;i<5 ;i++)
      rtf_destroy(i);
      pthread_delete_np (tasks[0]);
      pthread_delete_np (tasks[1]);
   }
```

Figure 5.8 : Le corps des fonctions d'initialisation et d'arrêt du module RT

Dans la première fonction init_module(), on procède premièrement à la création des piles Fifo servant à établir les canaux de communication entre le module temps réel et le module de transfert de messages. A l'aide de la fonction rtf_create(), les cinq RT_Fifos de l'application sont créés avec pour argument la taille de mémoire allouée à chaque Fifo. Par la suite, les tâches temps réel proprement dites seront créées par le biais de la fonction système pthread_create() qui possède pour arguments le nom de la tâche, la structure *attr* contenant les attributs de la tâche, le nom du programme principal de la tâche et un identificateur associé à l'appel de ce programme.

La structure *attr* est utilisée pour attribuer la priorité d'exécution à la tâche associée. Ainsi, on assigne la priorité « 1 », qui est la plus haute priorité, à la tâche d'acquisition de données nommée task[1] et la priorité « 2 » pour la tâche de régulation nommée task[0].

La dernière action entreprise dans la procédure d'initialisation est l'installation d'une fonction d'écoute sur le Fifo numéroté 2 qu'on a appelé RT_Fifo_ctr. La fonction est exécutée à chaque fois qu'il y a écriture sur ce Fifo. Cette installation est assurée par la fonction rtf_create_handler() ayant pour arguments le numéro de Fifo et le nom de la fonction d'écoute.

Dans la fonction d'arrêt du module temps réel, nommée cleanup_module() et qui est la dernière fonction appelée dans le module temps réel, il sera procédé à la fermeture des tâches temps réel et la destruction de toutes les piles Fifos. Ces deux actions sont assurées respectivement par les deux fonctions pthread_delete_np() et rtf_destroy().

5.6 - Fonction d'attente sur le FIFO :

Comme il a été mentionné au chapitre précédent, le module temps réel dispose d'un manipulateur d'attente faisant office d'organe d'écoute des messages de contrôle qui émaneraient du client distant. Ce manipulateur nous renseigne donc de l'arrivée de l'éventuel message qui sera dispatché vers la tâche temps réel concernée.

Ce manipulateur est créé au niveau de la fonction d'initialisation init_module() à l'aide de la fonction rtf_create_handler(). Cette opération d'écoute concerne la pile RT_Fifo_ctr et est assurée par la fonction qu'on a appelé my_handler(). Cette dernière fonction est décrite par le code suivant :

```
int my_handler(void)
{
    struct  msgrx mesrx;
    int err;

    while ((err = rtf_get(RT_fifo_ctr,&mesrx,sizeof(mesrx)))==
            sizeof(mesrx))
    {
     printk("l'action temps réel est .%c. \n",mesrx.action);

      If(mesrx.action == quit)
            {
              pthread_suspend_np(tasks[0]);
               pthread_suspend_np(tasks[1]);
            }

     for(ii=0;ii<4;ii++)
        ibuf[3-ii]= mesrx.tache[ii];
        taskno =  *(int *)&ibuf;

     select (taskno)
       {
        case 0  : rtf_put(RT_fifo_ctr1,&mesrx,sizeof(mesrx));
                  pthread_wakeup_np (tasks[0]);
                  break ;

        case 1  : rtf_put(RT_fifo_ctr2,&mesrx,sizeof(mesrx));
                  pthread_wakeup_np (tasks[1]);
                  break ;

        default : printk("tâche non disponible");
       }

       if (err != 0)
                {    return -EINVAL;    }
        return 0;
}
```

Figure 5.9 : La fonction d'écoute sur la pile RT_Fifo_ctr

Quand un message de contrôle arrive au niveau de la pile RT_Fifo_ctr, la fonction rtf_get() est appelée pour récupérer ce message à partir de la structure mesrx.

Comme première étape, le paramètre *action* du message est examiné. Si ce paramètre vaut *Quit*, les deux tâches temps réel seront suspendues à l'aide de la fonction pthread_suspend_np(). Si ce n'est pas le cas et suivant la valeur du paramètre *tache*, on appelle la fonction de réveil pthread_wakeup_np() qui permet de mettre en exécution la tâche concernée par le message. Par la suite, on envoie ce message de contrôle vers le Fifo de contrôle associé à cette tâche (RT_Fifo_ctr1 pour la tâche de régulation et RT_Fifo_ctr2 pour la tâche d'acquisition de données) et ceci par le biais de la fonction rtf_put().

5.7 - Description des tâches temps réel :

Comme déjà décrit dans la procédure d'initialisation du module temps réel, Les fonctions pthread_create() ont permis de créer les tâches temps réel mais ne les ont pas encore exécutées. Les programmes principaux de nos deux tâches récemment créées sont task_code0() pour la tâche de régulation et task_code1() pour la tâche d'acquisition de données.

La structure générale qu'on a adoptée pour l'écriture de ces deux programmes prend la forme de la figure 5.10.

La tâche temps réel (la fonction task_code()) est une boucle infinie qui exécute un programme propre à la fonction de la tâche. Ce programme appelle à chaque fin de boucle la fonction rt_task_wait(). Cette dernière est une fonction qui suspend l'exécution de la tâche qui l'a invoquée jusqu'à la prochaine activation ; moment où l'exécution va continuer à partir de l'instruction suivant immédiatement l'appel à la fonction rt_task_wait().

Cette activation est produite par deux processus : soit que cette tâche est réveillée par l'appel de la fonction pthread_wakeup_np(), soit qu'elle est définie comme étant une tâche périodique à l'aide de la fonction pthread_make_periodic_np(). Cette dernière fonction prend deux arguments de temps : le premier argument est l'instant exprimé en temps absolu, où la tâche est activée pour la première fois, et le second argument est la période séparant les activations successives après la première activation.

```
void *task_code(void *t)
{
       while (1)
        {
       int ret;
       int err;
       ret = pthread_wait_np();
       if ((err=rtf_get(RT_Fifo_ctri,&mesrx,sizeof(mesrx)))==sizeof(mesrx))
          {
           rtl_printf("Execution de la commande .%c. \n",mesrx.action);
           switch(mesrx.action)
             {
             case start_task:

                  …
            Extraction des paramètres de la tâche à partir de message mesrx
                  …
               pthread_make_periodic_np(tasks[i],gethrtime(),periode(i));
               break;

             case stop_task:
               pthread_suspend_np(tasks[i]);
               break;

             default :
               rtl_printf("RTL task: mauvaise action \n");
               return 0;
             }
           }
                 …
                  Corps principal de la  tâche
                 …


      rtf_put(RT_Fifo_datai,&mestx,sizeof(mestx));
   }
}
```

Figure 5.10 : La forme générale attribuée au programme principal de la tâche temps réel.

Dans le cas des activations périodiques, la tâche suspend elle-même son exécution (après avoir terminé son travail) et attend la prochaine activation dès l'occurrence de la prochaine période.

Cette activation est produite par deux processus : soit que cette tâche est réveillée par l'appel de la fonction pthread_wakeup_np(), soit qu'elle est définie comme étant une tâche périodique à l'aide de la fonction

pthread_make_periodic_np(). Cette dernière fonction prend deux arguments de temps : le premier argument est l'instant exprimé en temps absolu, où la tâche est activée pour la première fois, et le second argument est la période séparant les activations successives après la première activation.

Dans le cas des activations périodiques, la tâche suspend elle-même son exécution (après avoir terminé son travail) et attend la prochaine activation dès l'occurrence de la prochaine période.

Après l'envoi d'un message de contrôle par le client distant, la tâche temps réel le reçoit à travers le manipulateur d'écoute. Les paramètres inclus dans le message, propre à la tâche, seront extraits.

Concernant la tâche de régulation, un choix préalable d'un algorithme de régulation est effectué pour la commande de la vitesse du moteur à courant continu. Si le correcteur PID est choisi, les paramètres reçus par la tâche à partir de la pile RT_fifo_ctr1, sont la période d'exécution et les quatre paramètres de l'algorithme ; à savoir la valeur de consigne de la vitesse et les trois coefficients de correction K, Td et Ti.

Si l'action incluse dans le message est *Start_task*, la tâche sera activée périodiquement par la fonction pthread_make_periodic_np() avec la période mentionnée dans le message. Périodiquement donc, on appelle la fonction *reg_pid()* qu'on a défini dans le fichier *regulation.h*. Cette fonction met en service le correcteur suivant le schéma d'asservissement décrit dans la figure 3.1 du chapitre 3.

Si on choisit de travailler avec le correcteur GPC, la tâche de régulation suit le même déroulement que dans le cas du correcteur PID, sauf que pour ce correcteur, les paramètres reçus dans le message de contrôle sont, en plus de la période d'exécution, les facteurs N1, N2,λ et la consigne de vitesse.

La fonction de service du correcteur GPC, que nous avons écrite, est *reg_gpc()* définie toujours dans le fichier *regulation.h* . Cette correction suit le schéma d'asservissement présenté par la figure 3.2.

A la fin de chaque activation périodique de la tâche de régulation, les paramètres principaux sont regroupés dans la structure mestx et sont envoyés vers le client distant à travers la pile RT_fifo_data1.

La deuxième tâche d'acquisition de données du module temps réel s'occupe d'acquérir périodiquement un signal analogique (ou plusieurs) au moyen d'une carte d'acquisition de données. Les paramètres reçus à travers le message de contrôle émanant du client, sont la période d'exécution de la tâche (qui représente la période d'échantillonnage pour la lecture du signal), le numéro du canal d'entrée du signal dans la carte et le gain d'amplification.

A chaque activation périodique, la fonction chargée de l'acquisition, que nous avons appelé *daq(),* est appelée. Suite à cet appel, toutes les informations jugées utiles de cette tâche seront envoyées à travers la pile RT_fifo_data2 vers le client distant.

Il est à signaler que toutes les fonctions de gestion des deux cartes d'acquisition et de commande sont regroupées dans le fichier *interface.h*, Ce fichier joue le rôle de pilote de la partie matérielle associée à notre serveur.

5.8 - Vue générale sur le client :

Le client dans notre modèle a été équipé de fonctionnalités nécessaires pour observer et agir sur le procédé distant. Cette observation a été réalisée essentiellement par des graphes qui permettent de suivre l'évolution des paramètres contrôlés. De même ils ont été superposés des informations additionnelles pour enrichir la perception du système distant. La figure 5.9 donne une vue de l'interface graphique réalisée pour le Client.

Les programmes de notre client sont écrits en Delphi 5 .Cet outil de programmation fournit de grandes performances surtout dans la gestion des communications sous réseau en utilisant le composant socket défini dans l'environnement Windows.

Les structures des deux messages de réception et d'envoi, pour la communication entre le client et le serveur, ont été élaborées pour le client avec les mêmes formats que celles adoptées par le serveur sous l'environnement RTLinux. Cela garantit une concordance entre les deux applications client et serveur.

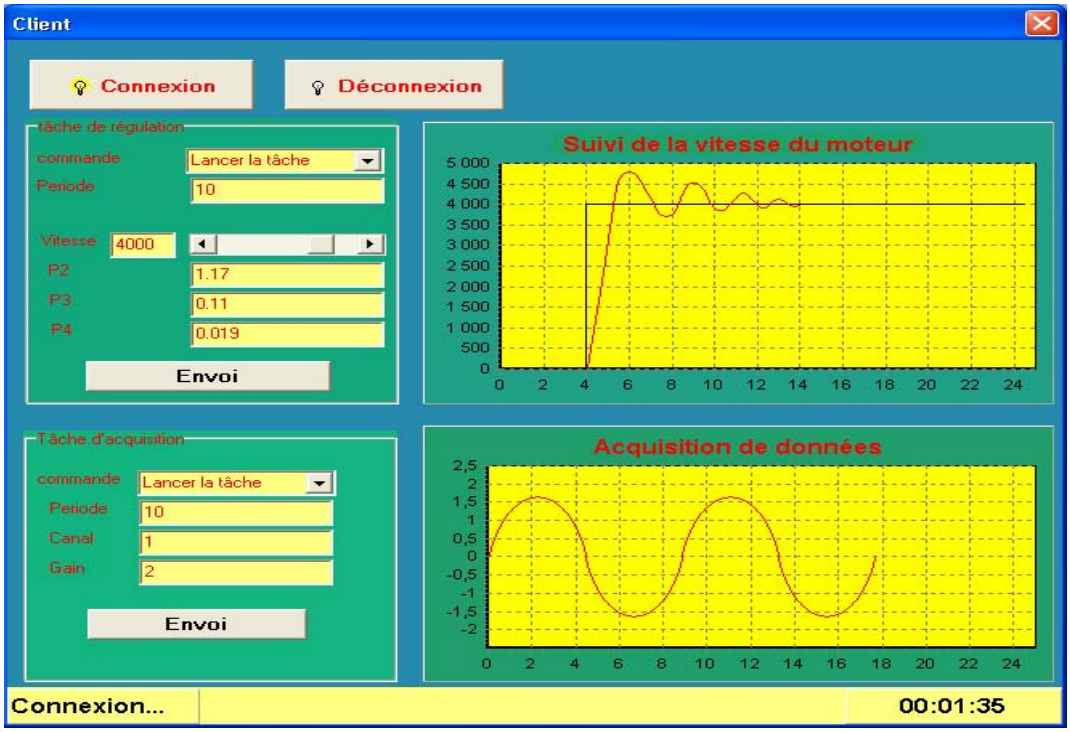


Figure 5.11 Vue de l'interface graphique du Client.

5.9 - Test d'ordonnançabilité des tâches temps réel :

L'élément central d'un système temps réel strict est un algorithme d'ordonnancement qui est chargé de déterminer l'ordre dans lequel doivent être exécutées les tâches. Un algorithme d'ordonnancement repose sur un modèle de tâches qui définit l'ensemble des restrictions auxquelles doivent se conformer les tâches, et sur un test d'ordonnançabilité qui détermine si oui ou non les échéances des tâches temps réel strict seront toujours respectées.

Un test d'ordonnançabilité doit prendre en compte la charge de travail maximale à laquelle le système peut être soumis, et les temps d'exécution au pire cas des tâches et du système d'exploitation.

Pour les tâches de notre module temps réel qui sont des tâches périodiques, leurs temps d'exécution au pire cas ne sont que leurs périodes d'exécution Ti. Pour déterminer la charge de travail maximale Ci de chaque tache temps réel, nous avons écrit une procédure, qui permet de l'estimer en prélevant le moment de commencement et le moment de fin de la boucle pour chaque activation périodique. La charge de travail de la tâche est donc la différence entre ces deux moments.

```
  …
T_fin=gethrtime();
charge = T_fin –T_debut;
pthread_wait_np();
T_debut =gethrtime();
  …
```

Les instants T_fin et T_debut sont prélevés avant et après l'appel de la fonction pthread_wait_np(). Ceci permet d'estimer juste le temps de travail effectif en excluant le temps pendant lequel la tâche est en attente de la prochaine activation périodique.

Suivant l'algorithme d'ordonnancement Rate Monotonic (RM), Le système est ordonnançable si le facteur d'utilisation :

$$U = \sum_{i=1}^{n} \frac{C_i}{P_i} \qquad \text{Vérifie la relation} \qquad U \leq U_0 = n \cdot \left(\sqrt[n]{2} - 1\right)$$

Avec *Ci* étant la charge maximale de travail de la tâche *Ti, Pi* sa période et *n* le nombre de tâches existantes

Dans ce test, même les tâches non RT sont introduites avec pour périodes équivalents à la plus petite des périodes des tâches temps réel.

Les charges maximales de travail des tâches que nous avons mesurées pour notre serveur, qui est un Pentium I 166Mhz de 32Mo de RAM, sont présentées dans le tableau suivant :

Tableau 5.1: Les charges maximales de travail mesurées pour les tâches temps réel

Tâches	**Charge de travail maximale (μs)**
La Tâche temps réel d'acquisition de données	247
La tâche RT de régulation (cas du correcteur PID)	195
La tâche RT de régulation (cas du correcteur GPC)	392
La tâche non RT correspondant au module de transfert de message	320

Dans le cas d'utilisation du correcteur GPC, si on fixe la période d'exécution de la tâche de régulation à 10 ms, la période minimale d'exécution que l'on peut appliquer à la tâche d'acquisition de données pour satisfaire la condition d'ordonnancement est :

$$T_{daq}(\min imale) = \approx \underline{877\ \mu s}$$

Cette période est équivalente à une fréquence maximale d'échantillonnage d'environ 1.14 KHz qui est la fréquence limite autorisée. Réellement, la fréquence d'échantillonnage doit être choisie inférieure à cette valeur limite pour s'éloigner de l'état de saturation du système.

Dans cet exemple, si on veut avoir une fréquence d'échantillonnage plus grande, on a deux solutions : soit qu'on augmente la période d'exécution de la tâche de régulation, soit qu'on remplace carrément le PC par un autre plus puissant.

Ce test d'ordonnançabilité est hors ligne, c'est à dire qu'il est effectué avant l'exécution de l'application entière. Il est nécessaire donc de faire cette vérification à chaque éventuel changement ou ajout dans les codes des tâches.

5.10 - Mise en œuvre des correcteurs PID et GPC :

La sortie du procédé que l'on commande doit évoluer pour suivre la consigne demandée. Il faut donc périodiquement appliquer, à l'entrée puissance du procédé, la commande appropriée à travers un correcteur adéquat. Le point

essentiel dans la mise en œuvre de ce correcteur est l'ajustement de ces paramètres jusqu'à l'aboutissement à un certain niveau de performance dans la commande du procédé.

Le travail réalisé dans cette partie consiste à déterminer les coefficients optimaux des deux correcteurs PID et GPC que nous avons implémenté pour la régulation de la vitesse du moteur à courant continu.

5.10.1 - Cas du correcteur PID :

La méthode adopté pour le réglage des 3 paramètres du correcteur K,Ti et Td est la méthode dite de Ziegler-Nichols décrite au chapitre 3. Cette méthode consiste à implémenter l'algorithme avec l'action proportionnelle seule. Par la suite, on ajuste en boucle fermée le gain proportionnel K de telle manière qu'il y ait une oscillation entretenue dans la boucle. On prélève dans ces conditions :

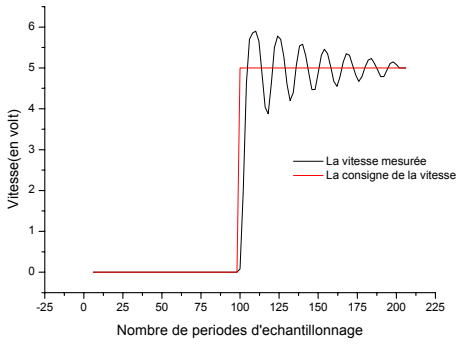
K_0 : le gain proportionnel qui fait ossiller le système,
T_0 : la période de ces oscillations à la sortie du système.

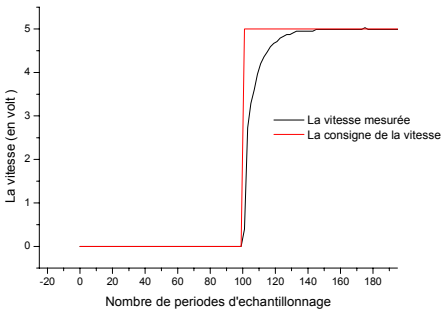
Pour l'estimation de ces deux paramètres, nous avons utilisé comme consigne d'entrée un échelon unité passant d'une vitesse de 0 tr/mn (qui correspond à 0 volt dans le module de commande du moteur) à 500tr/mn (correspond à 5 Volt). La période d'échantillonnage de l'algorithme est de 10 ms.

Par augmentation du paramètre K avec de petites fractions, on a abouti à la réponse avec des oscillations continues présentées dans la figure 5.12-a. Dans cet état, on a relevé la valeur de K_0 =2.18 et T_0 = 0,176 secondes. Les valeurs adoptées pour les 3 paramètres K, T_i et T_d du correcteur PID d'après la méthode de Ziegler-Nichols (voir tableau 3.1) sont :

$K=K_0/1.7 = 1.282$; $T_i = T_0/2 = 0.088$ s ; $T_d = T_0/8 = 0.022$ s

On représente dans la figure 5.12-b la réponse du système avec les valeurs optimales des trois paramètres du PID évalués par la méthode de Ziegler-Nichols.





(a) (b)

Figure 5.12 : Réglage des paramètres du PID par la méthode de Ziegler-Nichols

5.10.2 - Cas du correcteur GPC :

Le concept clé de la commande prédictive généralisée réside dans la création d'un effet anticipatif. On exploite pour cela les connaissances explicites sur l'évolution de la trajectoire à suivre dans le futur (connaissances nécessaires requises au moins sur un horizon de quelques points au delà de l'instant présent). La mise en œuvre de ce concept nécessite la définition d'un modèle numérique du système permettant de réaliser la prédiction du comportement futur du système. Ce modèle peut être obtenu par une discrétisation de la fonction de transfert continue du modèle.

Comme indiqué dans le manuel d'utilisation du banc d'essai que nous avons utilisé, le moteur à courant continu a été modélisé par une fonction de transfert du second ordre donnée dans le domaine continu par la relation :

$$F(s) = \frac{\Omega}{U_a} = \frac{7757{,}47}{s^2 + 59{,}56s + 13{,}18} \qquad (5.1)$$

Avec Ω : Vitesse du moteur (en tr/mn) ; U_a : la tension de commande (en volt)

Pour l'implémentation numérique de ce modèle, la fonction de transfert doit être écrite sous la forme récursive adopté par l'algorithme GPC comme décrit au chapitre 3 à savoir :

$$A(q^{-1})\Omega(t) = B(q^{-1})u_a(t - Te) + C(q^{-1})\xi(t) \qquad (5..2)$$

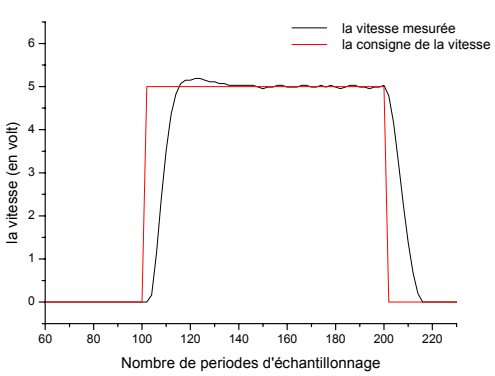
Par le remplacement de s par $\frac{1-q^{-1}}{Te}$ dans l'équation (5.1), q^{-1} étant l'opérateur retard et *Te* la période d'échantillonnage fixée à 10 ms, les polynômes A, B et C en q^{-1} seront :

$$A(q^{-1}) = 1 + 0.19q^{-1} + 0.07q^{-2}$$
$$B(q^{-1}) = 0.056$$
$$C(q^{-1}) = 0$$

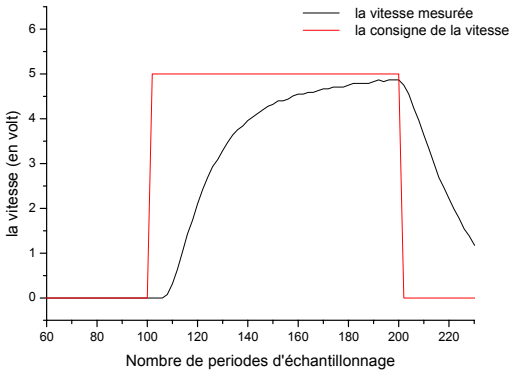
Comme décrit au chapitre 3, la synthèse du correcteur GPC dépend de 4 paramètres de réglage N_1, *N2, Nu*, et λ. L'horizon d'initialisation N_1 est choisi de telle sorte que N_1 *Te* soit égal au retard pur du système. Ainsi, pour un système ne présentant pas de retard ou un retard mal connu ou variable, N_1 est choisi égal à 1. L'horizon de prédiction de sortie N_2 est choisi de sorte que le produit N_2 *Te* soit limité par la valeur du temps de réponse souhaité. La valeur de L'horizon de commande N_u=1 est très souvent suffisante pour beaucoup d'applications relativement simples. Dans ce dernier cas, le calcul de la séquence de commandes futures se réduit au simple calcul de la première commande qui est effectivement appliquée au système. Le facteur de la pondération de la commande λ est celui dont l'influence est la plus évidente. En effet, il permet de pondérer l'influence des commandes dans l'optimisation et ainsi permet de générer un correcteur plus ou moins "énergique" donc plus ou moins rapide.

Pour résumer, le choix des paramètres se limite donc très souvent à une recherche bidimensionnelle (N_2 et λ) aboutissant à une sélection caractéristique d'un bon réglage.

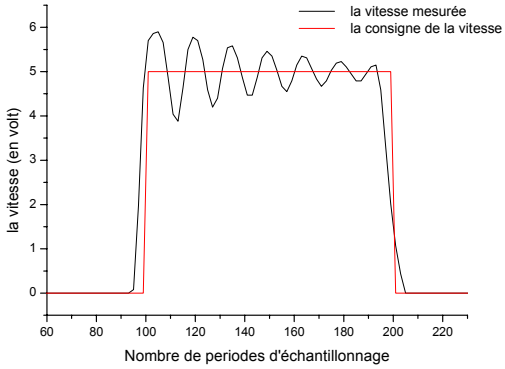
Nous présentons dans ce qui suit quelque résultats obtenues pour la commande de la vitesse de notre moteur en fixant N_1=1 et N_u = 1 avec comme consigne de commande un signal carrée de largeur de 100 périodes d'échantillonnage passant de 0 à 5 Volt.



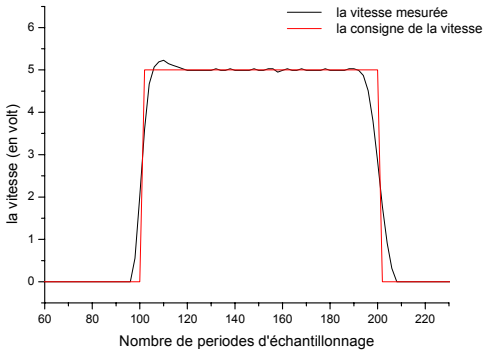
N_2=1, λ=0.1



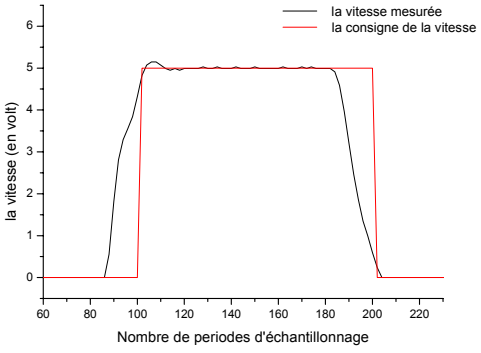
N_2=1, λ=0.2



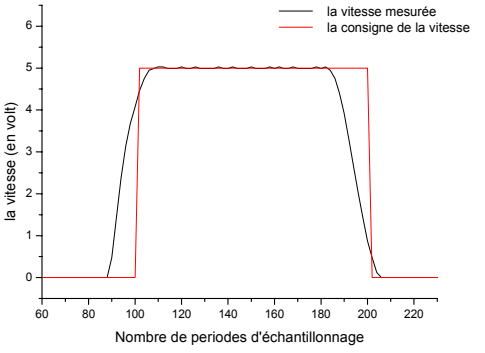
N_2=10, λ=0.15



N_2=10, λ=0.3



N_2=20, λ=0,3.



N_2=20, λ=0.5

Figure 5.13 : résultats de la commande GPC par variation du couple (N_2 et λ)

On constate d'après les réponses du système de la figure 5.13 que les deux paramètres N_2 et λ sont fortement liés. Leur ajustement ne se fait pas donc indépendamment l'un de l'autre. Le choix optimal de ces deux paramètres que nous avons pu réaliser pour produire la meilleure réponse est donné pour N_2= 23 et λ=0.4 (voir figure 5.14).

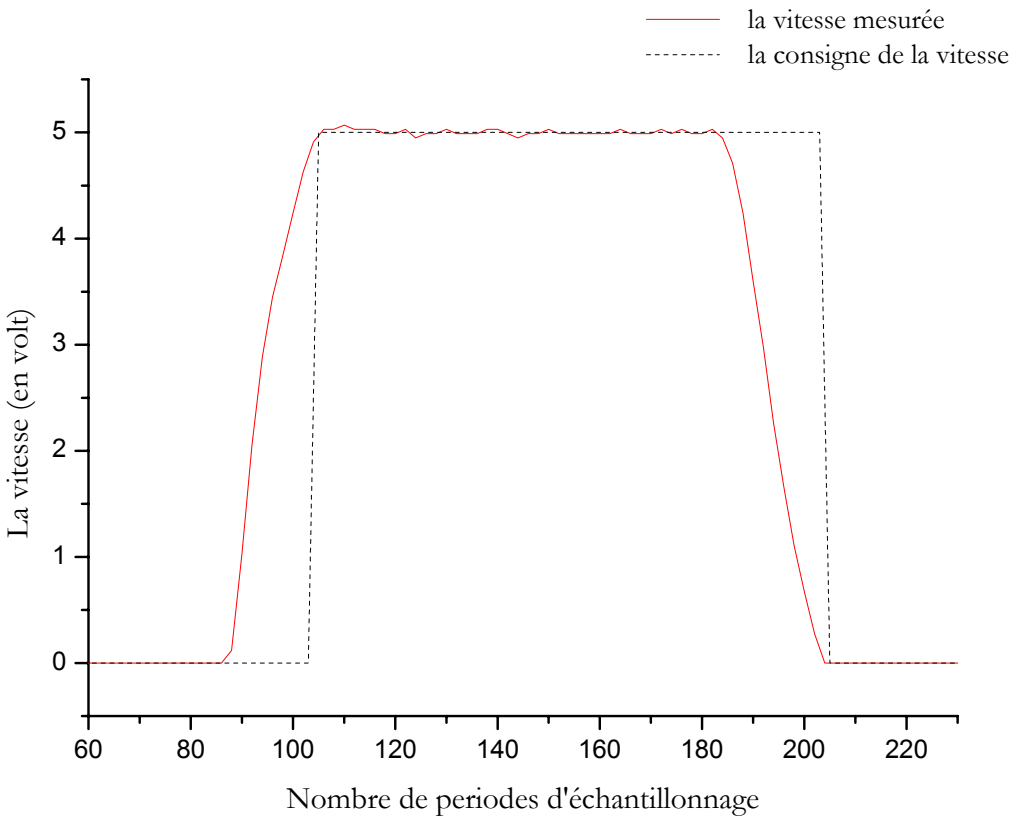


Figure 5.14 : résultat de la commande GPC pour N_2 = 23 et λ = 0.4

CHAPITRE 6

APPLICATION REELLE DU SYSTEME DE CONTROLE ETUDIE

6.1 - Introduction :

Au niveau de l'installation nucléaire de Birine, et dans le cadre des activités autour du réacteur El-Salam, on a procédé à une implémentation concrète de notre modèle de contrôle étudié. Il s'agit d'un système de contrôle commande d'un dispositif d'irradiation pour le dopage de silicium par transmutation neutronique. Cette technique consiste à irradier du silicium naturel dans un dispositif placé sur l'un des canaux expérimentaux du réacteur. La réaction nucléaire souhaitée est :

$$^{30}Si\ (n,\gamma)\ ^{31}Si \longrightarrow\ ^{31}P + \beta^{-}$$

Le but de cette technique est d'augmenter le nombre du phosphore (^{31}P) dans la cible du silicium pour réduire sa résistivité. Le silicium transformé trouve son application dans l'industrie de la microélectronique.

Pour satisfaire les exigences d'irradiation. Le système de contrôle conçu est constitué de :

- une chaîne de mesure des flux neutroniques instantanés au niveau des lingots de silicium pour le calcul des flux intégrés reçus durant toute l'irradiation;
- Un mécanisme de rotation du dispositif pour assurer une homogénéité radiale de flux neutronique autour des lingots ;
- Un circuit de refroidissement par air comprimé du silicium irradié et la mesure du débit de l'air et les températures à l'entrée et à la sortie du dispositif;

Le choix de l'architecture client/serveur dans notre application, basé sur une communication à distance à travers un réseau TCP/IP, est justifié principalement par le fait que le procédé contrôlé se situe dans une zone radioactive.

6.2 - Description du système de contrôle du dispositif d'irradiation :

6.2.1 - La chaîne de mesure du flux neutronique :

La mesure de flux neutronique est assurée par un détecteur appelé SPND (self powered neutron detector). C'est un détecteur permettant de suivre en continu les variations de flux neutronique dans le réacteur. Sa caractéristique principale est la non nécessité d'une alimentation électrique en plus ses petites dimensions le rendent tout indiqué comme moniteur lors des expériences d'irradiation. Cette mesure de flux est nécessaire afin de satisfaire d'une manière précise toutes les conditions liées à l'irradiation et la production.

La mesure de flux neutronique est basée sur la réaction nucléaire de capture neutronique $A(n,\gamma)B$. Ce détecteur fonctionne suivant le principe d'une diode. Une des électrodes est génératrice d'électrons ou particules β produites par la désintégration du corps B, proportionnellement aux captures neutroniques, c'est l'émetteur. L'autre électrode recueille ces charges, c'est le collecteur. Les deux électrodes sont séparées par un isolant (voir figure 6.1). Si on ferme le circuit (émetteur et collecteur) à travers un appareil de mesure extérieur, des charges électriques y circulent pour s'équilibrer. Ainsi le dispositif fonctionnera en générateur de courant proportionnel au flux neutronique incident.

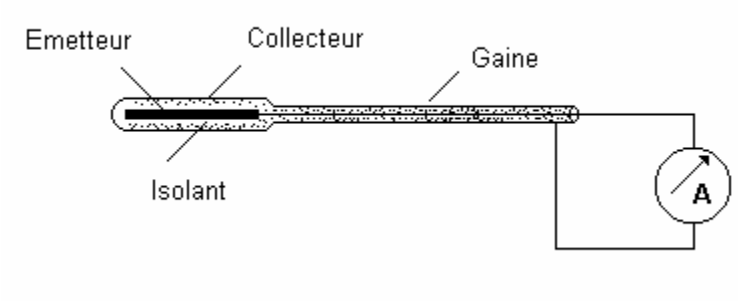


Figure 6.1 : Principe de fonctionnement du SPND

Le flux neutronique sera donc : $\Phi = S.I_d$

Avec S : la sensibilité du détecteur ;

I_d : le courant correspond au flux d'électrons de désintégration.

La chaîne de mesure proposée (Figure 6.2) est constitué de :

- deux détecteurs SPND placés dans le dispositif d'irradiation au niveau des lingots de silicium à irradier;
- un module d'amplification à quatre voies SPND acquis dans le cadre du projet ;
- un PC muni d'une carte d'acquisition qui sera décrite par la suite.

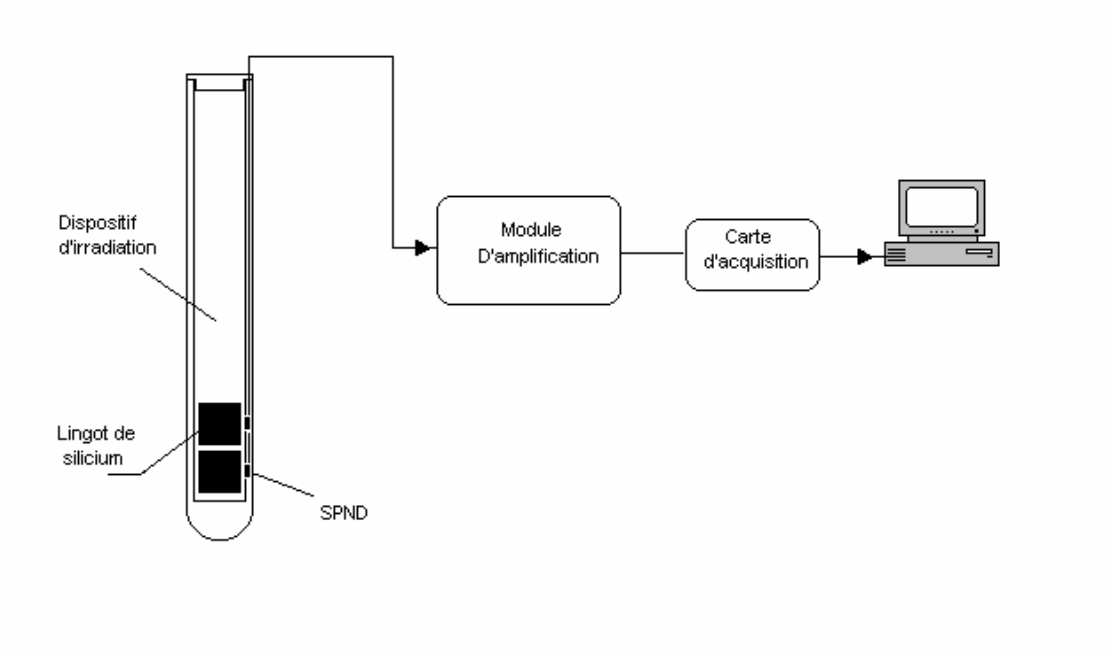


Figure 6.2 : La chaîne de mesure de flux neutronique

6.2.2 - Mécanisme de rotation du dispositif d'irradiation :

Afin d'avoir une homogénéisation radiale du flux neutronique incident, nous avons muni le dispositif d'un mécanisme de rotation composé de deux parties. La première partie mécanique est constituée d'un moteur asynchrone triphasé, un réducteur de vitesse et un transmetteur de mouvement par le biais d'une courroie entre l'axe du réducteur et le dispositif d'irradiation. La deuxième partie électrique est constituée d'un variateur de vitesse et un tachymètre pour la mesure de la vitesse de rotation du dispositif et un circuit de commande du moteur (Figure 6.3).

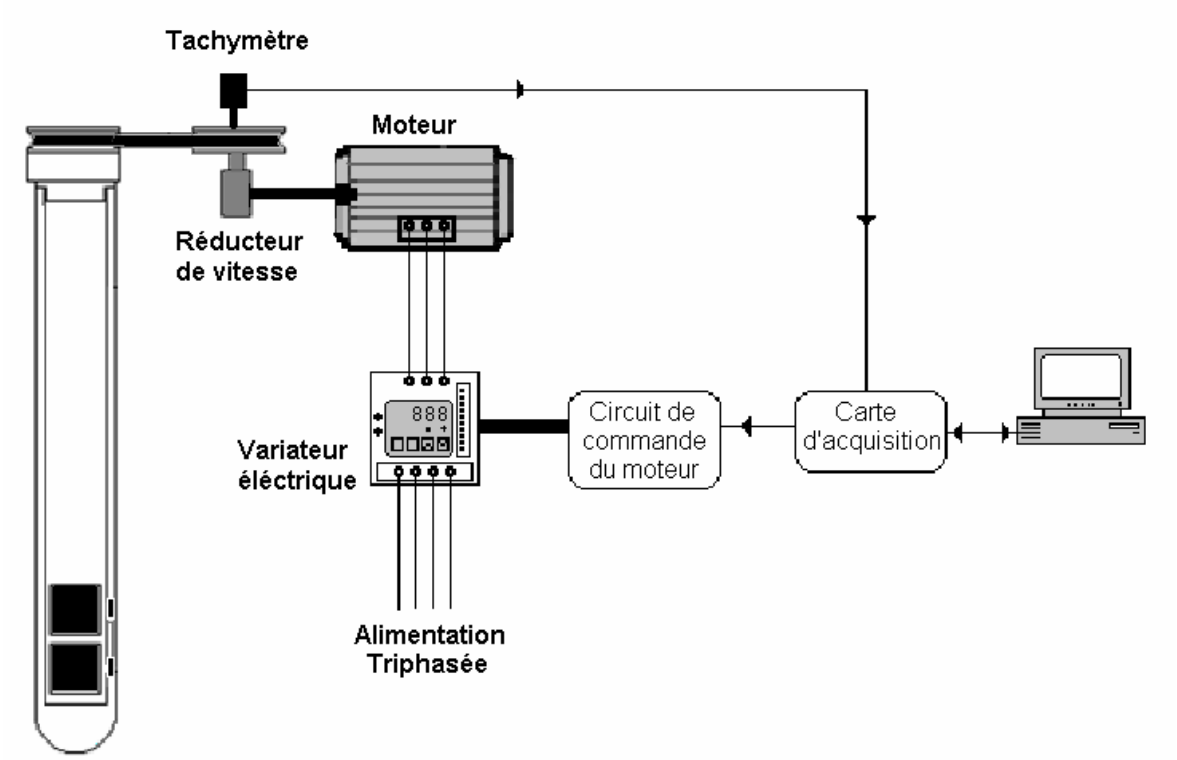


Figure 6.3 : Schéma de commande du mécanisme de rotation du dispositif.

La commande de mécanisme se fait par PC à travers la carte d'acquisition et de commande et consiste donc en la régulation de la vitesse de rotation dudit dispositif. La commande de la vitesse du moteur se fait par changement de la fréquence du secteur au niveau du variateur électrique.

6.2.3 - Circuit de refroidissement du dispositif d'irradiation :

Afin d'extraire la chaleur produite dans le dispositif généré par les rayonnements gamma dans le silicium irradié et les matériaux de structure dudit dispositif, nous avons doté notre procédé d'un circuit de refroidissement à air comprimé. Les paramètres contrôlés sont le débit de l'air de refroidissement et les deux températures à l'entrée et à la sortie du dispositif (Figure 6.4).

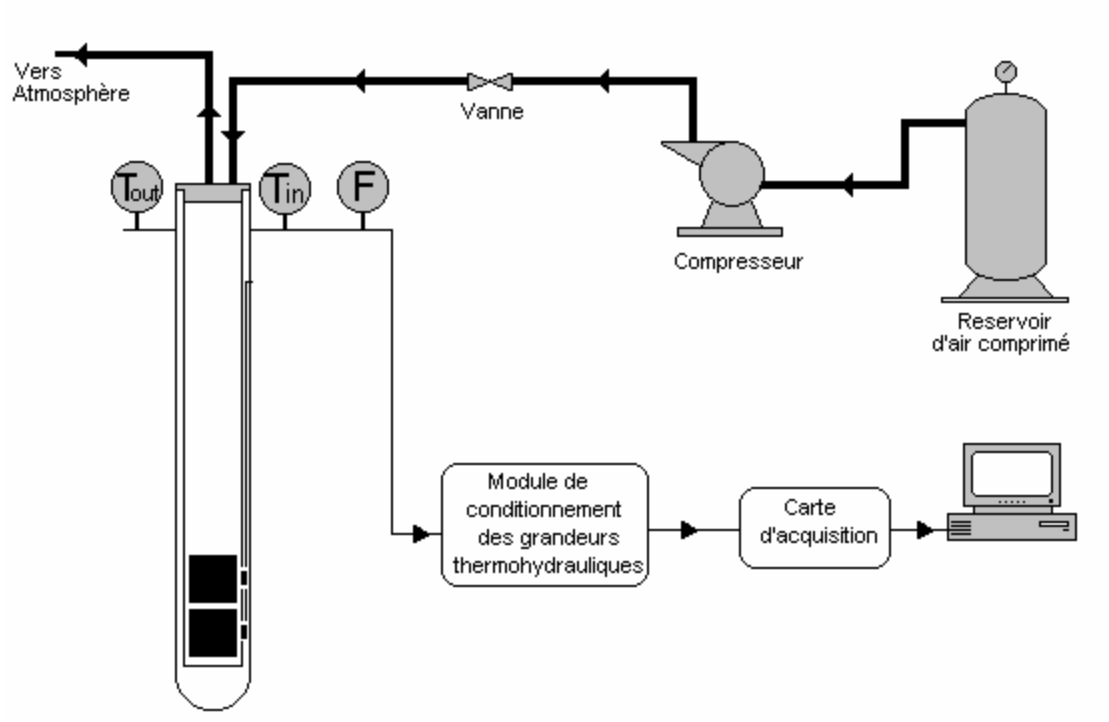


Figure 6.4 : Circuit de refroidissement du dispositif d'irradiation

La mesure des deux températures Tin et Tout est assurée par deux thermorésistantes de type PT100 intégrés dans des ponts de Wheatstone au niveau de module de conditionnement pour permettre une conversion résistance/tension 0-10V. La mesure de débit est réalisée par un débitmètre à tourbillon de type Vortex qui génère des oscillations de fréquence proportionnelle au débit de l'air de refroidissement. Le module de conditionnement assure la conversion fréquence/ tension 0-10V.

6.2.4 - La carte d'acquisition et de commande :

Notre carte d'acquisition, fournie dans le cadre du projet, est de type PCI-ADC de Blue-Chips Technology. Cette carte couvre tous nos besoins en terme de nombre de paramètres à acquérir et les performances demandées. Elle fournit 16 entrées analogiques 12 bits de gain programmable, quatre sorties analogiques bipolaires 12 bits. Il y a également 24 E/S numériques programmables compatibles TTL, trois timers/ compteurs programmables et une ligne d'interruption PCI contrôlée de manière sélective par n'importe laquelle des huit sources d'interruption sur la carte.

Vu le manque du pilote de la carte sous la plateforme linux, nous avons réalisé une bibliothèque de fonctions écrites sous le langage C. Ces fonctions permettent d'exploiter toutes les opérations d'entrées-sorties assurées par la carte en tenant compte de tous les modes de fonctionnement fournis.

6.3 - Description générale du Système de contrôle commande du dispositif :

La figure 6.5 présente d'une manière générale notre système de contrôle conçu qui s'articule sur une architecture client/serveur.

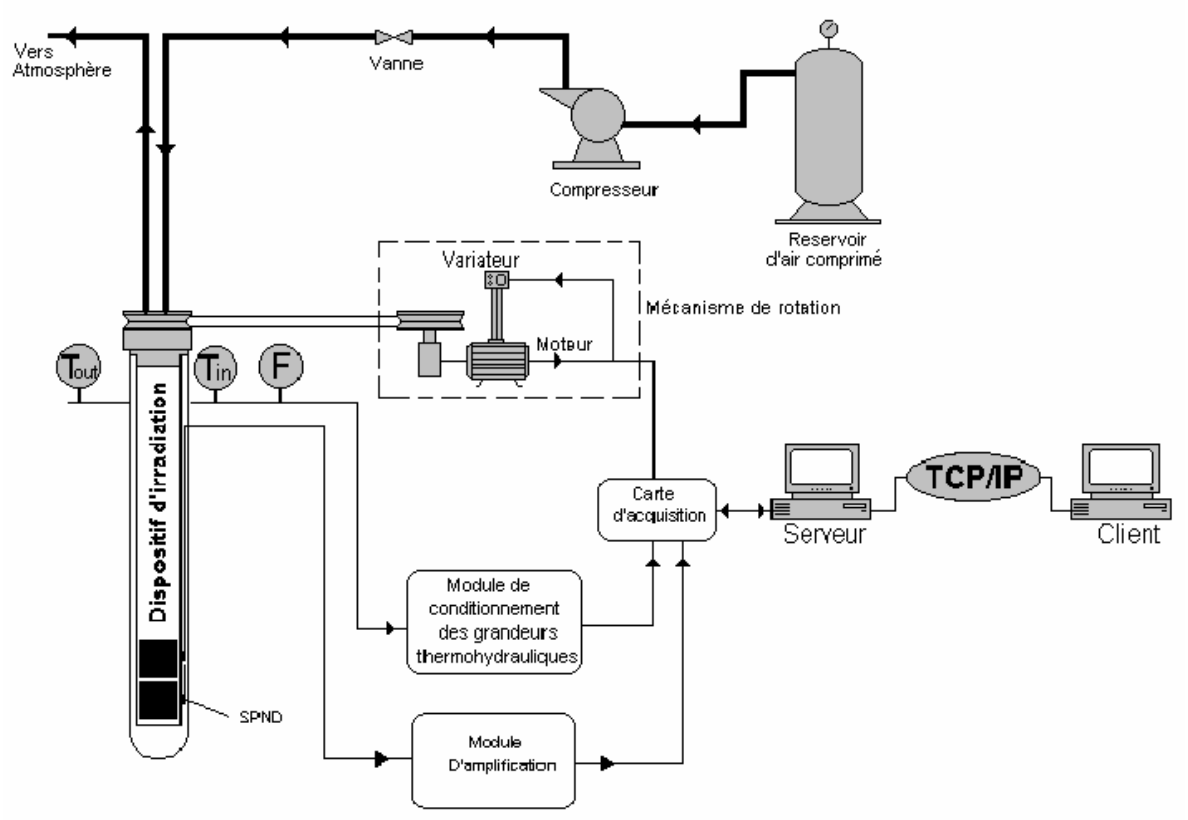


Figure 6.5 : Description générale du Système de contrôle commande du dispositif

Le serveur est l'unité liée directement à notre procédé à travers la carte d'acquisition et de commande, il est placé au niveau du hall du réacteur proche du dispositif. Ce serveur est assuré par un PC doté, comme nous savons, du système d'exploitation temps réel RTLINUX-3.1, qui possède la capacité à répondre aux contraintes temporelles qui se résument surtout dans notre cas à la

haute précision exigée sur les périodes d'échantillonnage pour les mesures précises des flux neutroniques instantanés au niveau des lingots de silicium. A partir de ces mesures, les flux intégrés seront calculés à partir d'une intégration numérique.

Ce serveur permet à travers la carte d'acquisition et de commande :

- L'acquisition des signaux issus des deux SPND pour la mesure des flux neutroniques reçus par les lingots de silicium ;
- la génération de la commande de la vitesse du moteur par le changement de la fréquence du secteur au niveau du variateur électrique et l'acquisition de signal correspond à cette vitesse de rotation fournie par le tachymètre électrique ;
- La commande du circuit de refroidissement et l'acquisition des trois paramètres thermohydrauliques associés (le débit de l'air et les deux températures de l'air à l'entrée et à la sortie du dispositif) ;

Au niveau logiciel, le programme conçu pour le serveur est composé de trois tâches temps réel et un processus pour l'établissement de la connexion avec le client :

- La première tâche du serveur est une tâche périodique temps réel. Elle a pour mission :
  - L'acquisition des deux paramètres de flux neutronique instantanée avec une période d'échantillonnage appropriée ;
  - Le calcul des flux intégrés à partir des mesures des flux instantanés en utilisant la méthode d'intégration numérique dite trapèze ;
  - L'envoi des paramètres mesurés et calculés vers le client à travers le canal de communication (RT_Fifo) associé.

- La deuxième tâche temps réel s'occupe de la commande numérique de la vitesse du moteur. L'algorithme de commande utilisé est de type PID (Proportionnel intégral dérivé) classique qui permet de maintenir la vitesse de rotation constante en l'occurrence des perturbations extérieures (la variation de la fréquence du secteur électrique, variation de la charge du moteur...). Les

paramètres appropriés du régulateur (les paramètres K, Td et Ti) ont été choisis d'une manière expérimentale.

■ La troisième tâche temps réel dans le serveur s'occupe de la commande on/off du compresseur d'air et l'acquisition des trois paramètres thermohydrauliques et l'envoi de ces données au client.

■ Le processus non RT dans le serveur est chargé d'établir la connexion entre le serveur et le client et d'assurer les moyens de transmission des données entres eux. Cette transmission se fait à travers quatre structures de messages; l'un de contrôle et les trois autres de données.
La structure de contrôle reçoit les messages émanant du client. Elle comprend les fonctions suivantes :

- Les ordres d'exécution et de l'arrêt des tâches temps réel ;
- Les paramètres de fonctionnement du système (les périodes d'exécution des trois tâches. La valeur de consigne de la vitesse de rotation du dispositif et les paramètres de régulateur PID et la commande d'action du compresseur).

La première structure de message de données permet l'envoi vers le client des données concernant la première tâche temps réel à savoir :

- Les deux mesures de flux neutronique instantané ;
- Les deux valeurs calculées des flux intégrés ;

La deuxième structure de données concernant la deuxième tâche temps réel s'occupe de l'envoi de la valeur mesurée de la vitesse de rotation du dispositif. La troisième structure de données se charge de l'envoi des trois paramètres thermohydrauliques de la troisième tâche temps réel.

Ce processus se sert du mécanisme de programmation *socket* pour l'établissement de la communication réseau entre le serveur et le client. Ce concept, décrit au chapitre 2, permet la communication point à point entre deux applications s'exécutant éventuellement sur deux machines différentes.

Le client, dans notre système, est un PC placé dans une salle de contrôle loin du procédé et de toute zone radioactive. Il est relié, comme mentionné auparavant, au serveur à travers une liaison réseau et fonctionne sous Windows. Il offre à l'utilisateur une interface graphique et facilite la gestion des données de procédé. Les programmes sont écrits dans le langage de programmation Delphi version 5.

Le client est chargé des fonctions suivantes :

- L'établissement de la liaison réseau avec le serveur ;
- L'envoi des commandes d'activation et d'arrêt des tâches temps réel du serveur ;
- L'envoi des paramètres de fonctionnement des tâches temps réel ;
- La réception de toutes les données mesurées ou calculées par les tâches temps réel ;
- La présentation graphique des paramètres de procédé;
- L'enregistrement des données sur fichiers.

CONCLUSION

Le travail présenté dans ce mémoire porte sur l'étude d'un modèle de contrôle en temps réel d'un procédé industriel. Ce modèle est basé sur une architecture client/serveur avec un serveur temps réel qui commande le procédé et diffuse ses informations vers un superviseur client par le biais d'un réseau informatique classique avec un protocole TCP/IP.

Le serveur a été construit autour de deux modules : le premier module temps réel (RT) à été réalisé pour interfacer le serveur avec le monde extérieur en créant deux tâches temps réel, l'une de contrôle et l'autre d'acquisition de données à travers des cartes d'acquisition et de commande. Le deuxième module de serveur est le module de communication; ce dernier s'apparente à un pont entre Le client distant et le module temps réel. Le client dans notre modèle a été équipé de fonctionnalités nécessaires pour observer et agir sur le procédé distant.

A travers l'implémentation de ce modèle de contrôle proposé, Nous pensons que nous avons pu acquérir une certaine expertise dans L'utilisation de l'informatique temps réel dans la mise en œuvre des systèmes automatisés. Ce travail nous a permis entre autre de :

- ✓ Ressortir l'intérêt du temps réel dans la commande des processus industriels par rapport aux systèmes classiques;
- ✓ La mise en oeuvre du système d'exploitation Linux comme étant un système multitâches et multi-utilisateurs. Son implantation nous a permis d'explorer ses grandes performances et sa transparence puisque les sources de son noyau système peuvent être obtenues, compilées et installées et même modifiées pour satisfaire les exigences du programmeur.

- ✓ L'installation, la mise en service et l'utilisation du noyau temps réel RTLinux. Ce dernier, qui est une extension du système d'exploitation Linux, convertit celui-ci en un environnement temps réel matériel. RTLinux a montré qu'il est bien capable de gérer toutes les contraintes possibles connues dans la manipulation des tâches temps réel. Il a aussi prouvé sa capacité de répondre à des exigences temps réel telles que : les échéances de temps, la gestion de périodicité, la gestion de priorité et bien d'autres contraintes temporelles de gestion des tâches temps réel.
- ✓ L'écriture sous l'environnement RTLinux de plusieurs bibliothèques de fonctions pour l'exploitation des cartes d'acquisition et de commandes utilisées dans notre travail.
- ✓ Acquérir une expertise dans la planification dans le temps des tâches temps réel notamment dans l'établissement des bilans temporels de ces tâches en vue de leur ordonnancement.
- ✓ La mise en œuvre de l'architecture client/serveur. Cette architecture permet au client et au serveur de se trouver en réseau, connectés d'une manière transparente. En outre, elle offre la possibilité de partager les données de procédé sur tout un réseau local.
- ✓ L'implémentation numérique des deux algorithmes de contrôle, le PID et le GPC, pour l'asservissement en vitesse d'un moteur à courant continu. Un travail de synthèse a été réalisé dans cette partie pour déterminer les coefficients optimaux de ces deux correcteurs pour obtenir le réglage optimal de ce procédé et ceci sans entrer dans les aspects avancées de contrôle (stabilité, contraintes, identification …etc.).

Une première application concrète de notre modèle de contrôle en temps réel a été mise en œuvre. Il s'agit d'un système de contrôle d'un dispositif d'irradiation pour le dopage de silicium par transmutation neutronique au niveau du réacteur Es Salam de Birine. Le système de contrôle réalisé a prouvé sa capacité pour une meilleure prise en charge des paramètres d'irradiation. De ce fait elle ouvre la voie vers son application pour d'autres procédés existants au niveau du réacteur Es-Salam de Birine.

Une deuxième application envisagée pour notre modèle de contrôle est de réaliser un laboratoire à distance à des fins pédagogiques. Ce concept permet de s'affranchir non seulement de la contrainte de lieu, mais aussi des horaires souvent rigides associés aux travaux pratiques. Cette nouvelle possibilité de réaliser des expériences de laboratoire à distance permet un apprentissage plus flexible et étend également les moyens didactiques de présentation. En effet, un enseignant peut réaliser par ce biais des démonstrations réelles en salle de cours, sans avoir à déplacer l'expérience à présenter. Cette application est en phase de concrétisation et elle sera enrichie par des outils de visualisation offrant le pouvoir de *sentir* l'expérience de la même manière que si l'utilisateur se trouvait à côté d'elle.

REFERENCES

1: J.A. Biancolin « *Spécification et conception des systèmes temps réel*. » Synthèses informatiques – CNAM. Hermès, Paris, 1995.

2 : D. Tschirhart. « *Commande en temps réel*. » Dunod, Paris, 1990.

3: S. Bennett. « Real-*Time Computer Control – An introduction*. » Prentice Hall, UK, 1994.

4 : J-P. Elloy. « *Les contraintes du temps réel dans les systèmes industriels répartis.* » *RGE*, , Février 1991.

5 : J-J. Montois. « *Gestion des processus industriels temps réel.* » Ellipses, 1999.

6 : W. Blachier. « *Le temps réel* ». Publications de l'Institut National Polytechnique de Grenoble, 2000

7 : D. Decotigny. « *Une infrastructure de simulation modulaire pour l'évaluation des performances de systèmes temps-réel ».* Thèse de doctorat en informatique, Université de Rennes 1, Avril 2003

8 : L.Bernard *« Présentation de RTLinux »* Publications de ATRID Systèmes ,1999 , www.atrid.fr

9 : M. Barabanov et V.Yodaiken. *« Real-time Linux »* Linux journal, Février 1997.

10 : L.Toutain *« Réseau locaux et Internet »* Hèrmes, Paris. 1997

11 : S. Rouveyrol *«Programmation réseau sur TCP/IP, l'interface des sockets »* Rapport ENSIMAG, Paris 1993

12 : H. Richard « *Asservissements, notes du cours »*. École Polytechnique de Montréal, septembre 2000.

13 : P. Boucher et D. Dumur *« La commande prédictive »* Technip Editions, Paris 1996

14 : G. Ramond *« Commande prédictive généralisée adaptative directe et Applications »* Thèse de doctorat en Sciences, Université de Paris XI, Septembre 2001

15 : N. Minthtri *« Commande adaptative multivariable avec contraintes »* Thèse de doctorat en Automatique, Institut National Polytechnique de Grenoble, 1989

Documents fournis sur le site officiel de RTLinux : www.fmslabs.com

16: M. Barabanov: *"A Linux-based Real-Time Operating System"*

17: V. Yodaiken: *"The RTLinux Manifesto"*

18: V. Yodaiken et M. Barabanov: *"A Real-Time Linux"*

19: J. Esplin: *"Linux as an Embedded Operating System"*

20: FSMLabs: *"RTLinux FAQ"*

21: V. Yodaiken: *"Cheap operating systems research and teaching with Linux"*

22: F.M. Proctor: *"Using Shared Memory in Real-Time Linux"*

23: H. Bruyninckx: *"Real Time and Embedded HOWTO"* , K.U. Leuven

24: C. Schroeter: "*Programming PCI-Devices under Linux"*

25: P.Kadionik: *"Drivers de la carte CANPCI sous MacOS et sous MSDOS"*

26: R. Baruch & C. Schroeter: *"Writing Character Device Driver for Linux"*