RAPPORT

DE

PROJET

LEBOCEY PIERRE

NANOASCIONE GERALD

Sommaire

Introduction p 4

1ère partie : *Liaison entre deux PC* p 5

1°) Présentation générale
2°) Notion Serveur/Client
3°) Liaison Serveur/Client sous TCP
4°) Liaison Serveur/Client sous TCP multitâche
5°) UDP/IP, TCP/IP Quelles sont les différences ?
6°) Choix

2ème partie : *Le client de la communication* p 19

1°) Qu'est ce que FESTO ?
2°) Les programmes FST
3°) Notre protocole
4°) Composition du programme
5°) Avantages et Inconvénients

3ème partie : *Partie logicielle* p 27

1°) Présentation DELPHI
2°) Composants Piette
3°) Partie DELPHI du Projet
4°) Forme principale
5°) Particularité
6°) Télécommande

Conclusion p 41

Annexes p 42

INTRODUCTION

Dans le cadre du projet de deuxième année, nous avons à réaliser la connexion entre un PC via de la programmation Delphi et un automate programmable Festo, et tout cela par l'intermédiaire de sockets TCP/IP.

Ce projet se place dans un ensemble de projets : la supervision d'un automate par Internet. Un groupe s'est occupé de la construction de l'automate, mise en place des capteurs et des moteurs et programmation. Un binôme a programmé un superviseur en trois dimensions avec le logiciel delphi et nous, nous avons donc fait la liaison entre les deux applications
Le travail s'est d'abord fait séparément chaque binôme travaillant de son coté et nous avons rassemblé le travail durant les dernières séances de projet.

Pour faire la liaison automate PC, nous avons premièrement étudié différents protocoles Internet : TCP/IP, UDP/IP entre deux PCs ensuite chacun a travaillé sur une partie : un a fait le coté automate et l'autre a fait la partie PC.

I Liaison entre deux PCs

1°) présentations générales

Dans la première partie de notre projet, nous avons étudié, quelques protocoles d'Internet : TCP/IP et UDP/IP. Nous avons analysé ces protocoles parce qu'ils sont faciles à mettre en œuvre sur delphi et sur les IPC Festo.

➢ protocole IP

Un protocole est une méthode standard qui permet la communication entre deux machines, c'est-à-dire un ensemble de règles et de procédures à respecter pour émettre et recevoir des données sur un réseau.
IP (Internet Protocol) est un protocole très répandu permettant de faire communiquer des machines de différents constructeurs
Si à strictement parler IP est un protocole de transport de l'information, sa popularité est liée à tout un ensemble de produits comme FTP (transfert de fichiers), Telnet (session sur une machine distante), SMTP (courrier électronique)...

➢ adresse IP

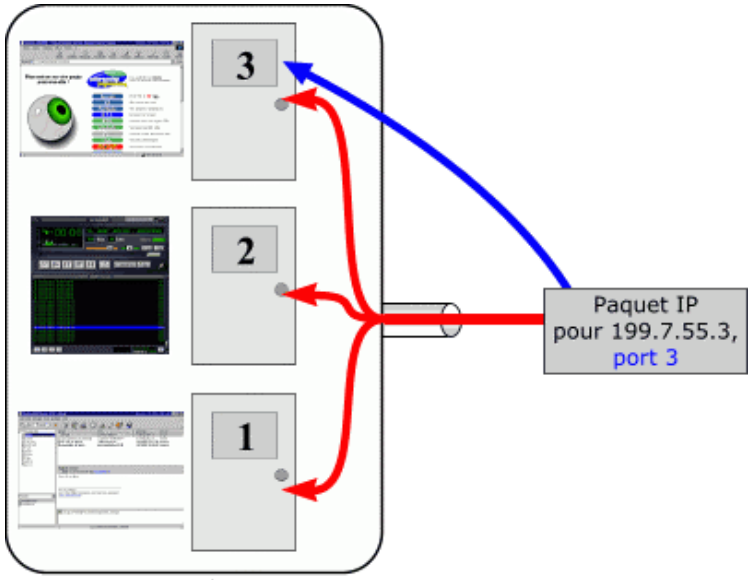
Dans le protocole IP chaque système informatique (ordinateur, automate programmable) a une adresse unique. Ces adresses sont gérer mondialement et chaque constructeur reçoit un lot d'adresse qu'il peut allouer à ses machines. Toutes les adresses sont faites de quatre nombres compris entre 0 et 256 et sont séparé par des points. Par exemple : 193.50.179.217
Les premiers bits indiquent la classe du réseau (A, B, C) et l'adresse du réseau et les derniers bits donnent l'adresse de la machine dans le réseau. Dans le cas de l'adresse précédente :

193.50.179.217 ⇨ adresse du pc dans le réseau
⇩ ⇩ ⇩

@du réseau

- numéro de port

De nombreux programmes utilisant un protocole IP peuvent être exécutés simultanément sur Internet (Vous pouvez par exemple ouvrir plusieurs navigateurs simultanément ou bien naviguer sur des pages HTML tout en téléchargeant un fichier par FTP). Chacun de ces programmes travaille avec son propre protocole toutefois l'ordinateur doit pouvoir distinguer les différentes sources de données.
Ainsi, pour faciliter ce processus, chacune de ces applications se voit attribuer une adresse unique sur la machine, codée sur 16 bits : le numéro de port. Lorsque l'ordinateur reçoit une requête sur un port, les données sont envoyées vers l'application correspondante



Attention toute une série de port est déjà réservé, on ne peut pas les utiliser :

Voici certaines de ces assignations par défaut :

Port	Service ou Application
21	FTP
23	Telnet
25	SMTP**Erreur! Signet non défini.**
53	Domain Name Server
63	Whois
70	Gopher
79	Finger
80	HTTP
110	POP3
119	NNTP

Les ports 0 à 1023 sont les ports reconnus ou réservés, les ports 1024 à 49151 sont appelés ports enregistrés, les ports 49152 à 65535 sont les ports dynamiques ou privés
Du côté du client, le port est choisi aléatoirement parmi ceux disponibles par le système d'exploitation. Ainsi, les ports du client ne seront jamais compris entre 0 et 1023 car cet intervalle de valeurs représente les ports connus.

- le socket

La combinaison adresse IP + port est alors une adresse unique au monde, elle est appelée socket.

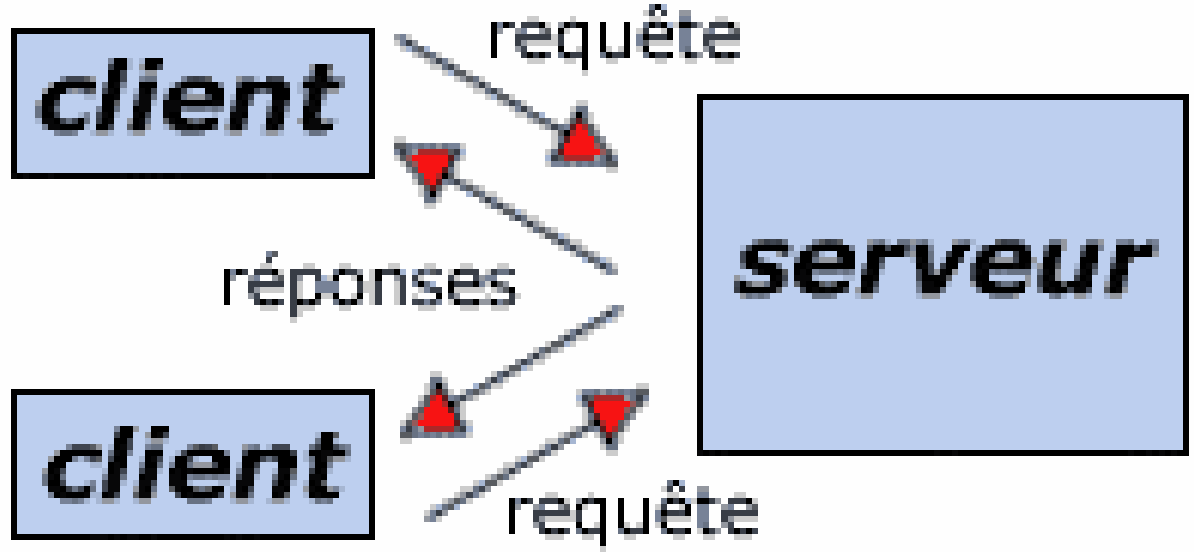
- client /serveur

La liaison client / serveur est un type de liaison particulier dans lequel un programme demande a communiquer avec un deuxième programme : c'est le client de la communication.

Le second programme, celui à qui le client demande une communication, va traiter la demande et y répondre : ce programme est appelé serveur.

Il peut y avoir plusieurs clients dans la liaison. Tous les clients voient le serveur à la même adresse et au même port. Il est donc nécessaire que le serveur libère le plus vite possible le port où les clients l'appellent. Cependant les clients ne se voient pas entre eux.

Un système client / serveur fonctionne selon le schéma suivant :



Dans notre projet, le client est l'automate car c'est lui qui va demander la communication. Le serveur est donc le programme Delphi.

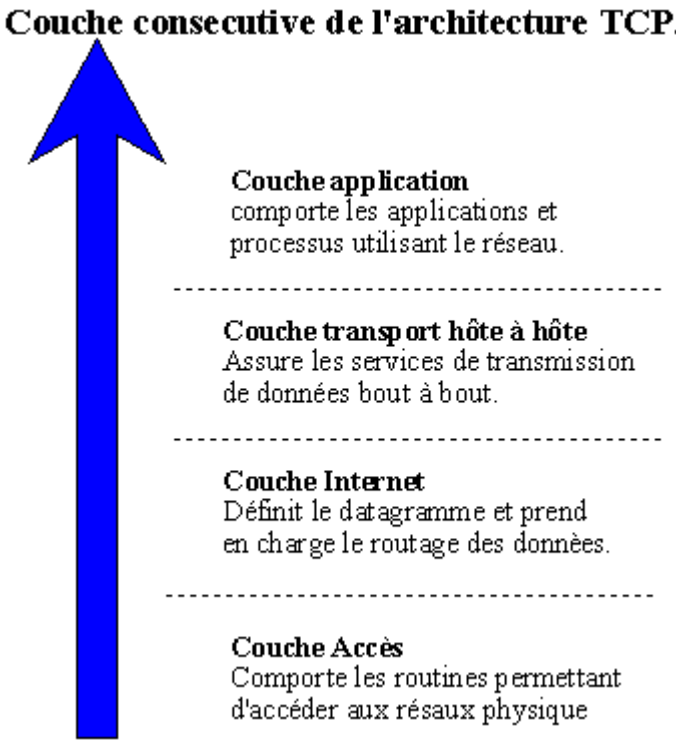
3°) liaison serveur client sous TCP/IP

➢ TCP/IP

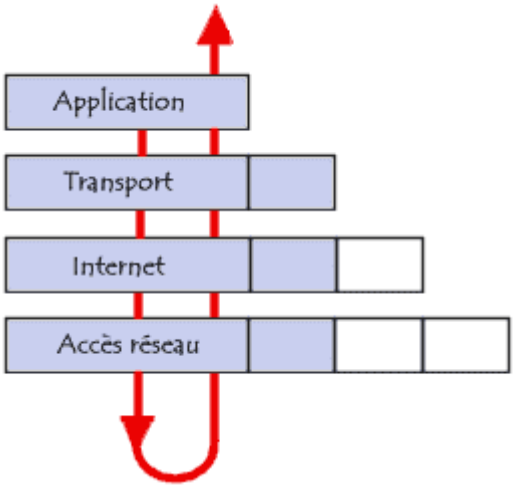
TCP/IP est un ensemble de protocoles : TCP (Transmission Control Protocol) et IP. Selon le model OSI, TCP/ IP est un protocole architecturé en 4 couches :

- Couche accès réseau : spécifie la forme sous laquelle les données doivent être acheminées quels que soit le type de réseau utilisé.
- Couche Internet : (IP) elle est chargée de fournir le paquet de données (datagramme).
- Couche Transport : (TCP) elle assure l'acheminement des données, ainsi que la gestion des erreurs.

➨ Couche Application : elle englobe les applications standard du réseau.



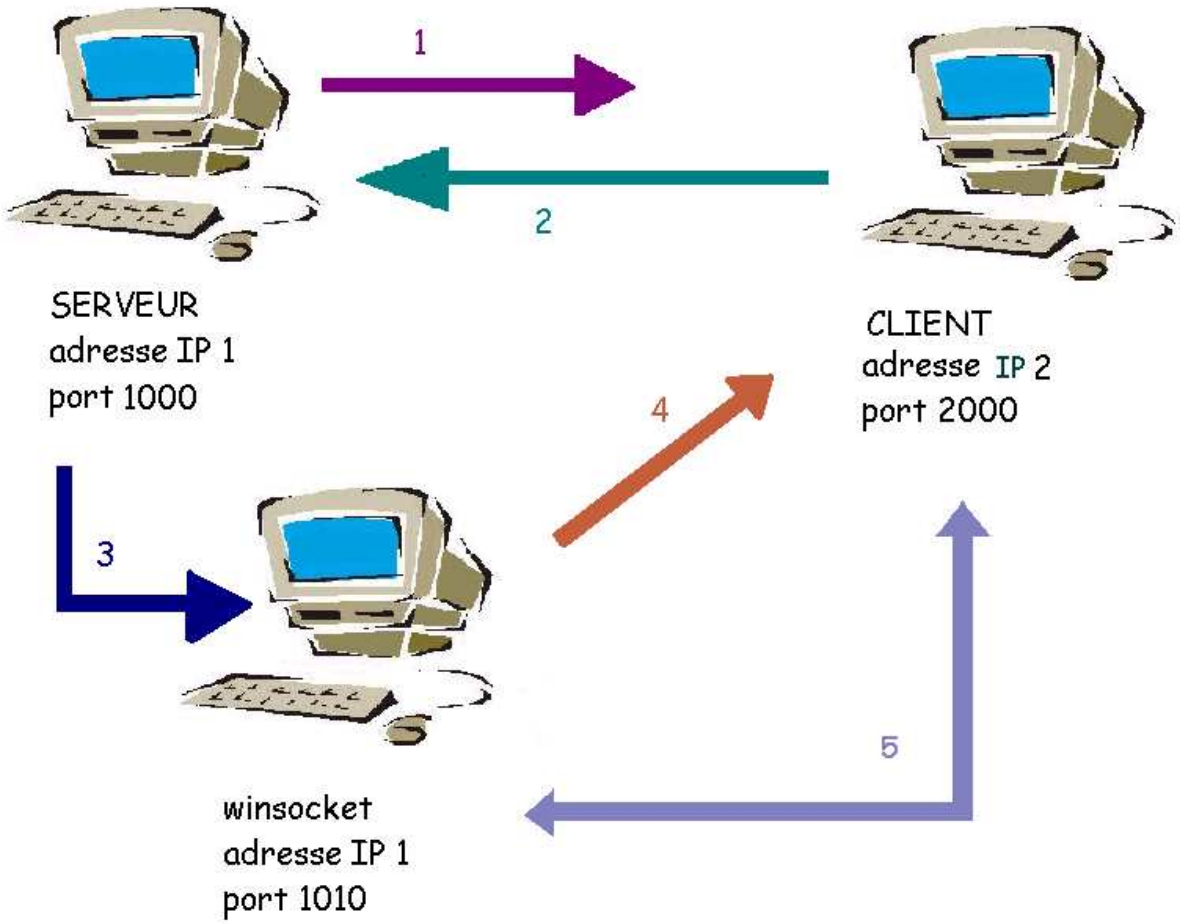
Ces couches sont encapsulées, c'est à dire que lors d'une transmission la couche émettrice demande à une couche inférieur, par l'intermédiaire d'une primitive de service, d'envoyer ses données en y ajoutant son en-tête. A la réception, chaque couche enlève son en-tête et fait passer les données à la couche supérieur :



- liaison entre un serveur et un client par TCP/IP

➔ mise en communication

Au début de notre projet, nous avons étudié la liaison serveur / client en programmant des sockets TCP/IP sur delphi. Les sockets client sont différents et beaucoup plus simples que les sockets serveurs. Voici comment se déroule la mise en communication : Il y a cinq étapes.



Etape 1 :

Le serveur doit se mettre en écoute et ouvre ses ports pour être prêt à recevoir les demandes des clients. Puis le serveur devient passif est attends En delphi : ServerSocket1.open;

Etape 2 :

Le client fait une demande de connexion en envoyant un paquet avec l'adresse du serveur et le port avec lequel il veut communiquer.

Il envoi aussi sa propre adresse et son propre port (celui-ci est choisit par l'OS) En delphi, il faut avoir configurer le socket et on code : ClientSocket1.open ;

Etape 3 :

Le serveur crée automatiquement un autre socket (WinSocket sous Delphi) qui a bien évidemment la même adresse mais qui change de port (c'est le système d'exploitation qui gère ce changement)
A ce moment avec delphi, il y a une procédure d'acceptation de la communication et on peut créer une nouvelle forme pour le client :

```
procedure TForm1.ServerSocket1Accept
      Begin
        Form2:=Tform2.Create(self);
         End ;
```

Etape 4 :
Ce nouveau socket du serveur renvoi un message au client. Dans ce message, il lui dit de changer de port d'envoi (port remontant) et de communiquer avec lui sur ce nouveau port.

Etape 5 :
La communication est établie, le client est le serveur correspondent en s'envoyant des paquets mutuellement.
De plus le premier port du serveur est libre donc un autre client peut arriver et demander une liaison avec le serveur.

➔communication établie

Nous avons fait des essais de cette communication en développant un chat.
Du coté client, on peut surveiller l'arrivée de données et les afficher :

```
procedure TForm1.ClientSocket1Read
```

```
Begin
    edit2.Text:=clientsocket1.Socket.ReceiveText;
end;
```

L'envoi de données est aussi simple que la réception :

```
procedure TForm1.Button1Click
      Begin
            clientsocket1.Socket.SendText(edit1.Text);
      end;
```

On peut même récupérer les données de la liaison :

```
procedure TForm1.ClientSocket1Connect
  Begin
    label1.Caption:='Connecté avec '+Socket.RemoteHost;
    with Socket do
      Begin
        Form1.label3.caption:=IntToStr(RemotePort); ⟹ port du
serveur
        Form1.label5.caption:=remotehost; ⟹ host du serveur
        edit1.Text:=inttostr(localport); ⟹ port du client
      end;
end;
```

L'envoi et la réception de données se fait de même du coté serveur.

➔ fin de la communication

Le client peut demander à mettre fin à une connexion au même titre que le serveur.
La fin de la connexion se fait de la manière suivante:
Du coté serveur : serversocket1.Socket.Close;
Du coté client : Clientsocket1.Close;

4°)TCP multitâche ou INDY

Pour ce protocole nous n'avons fait que le serveur et nous avons utilisé un client en TCP monotâche. Ce type de communication est plus simple a utiliser car tout le code est dissimulé au yeux de l'utilisateur. Cependant il est plus compliqué à appréhender.

Le TCP multitâche a, au début, un fonctionnement identique au TCP monotâche.

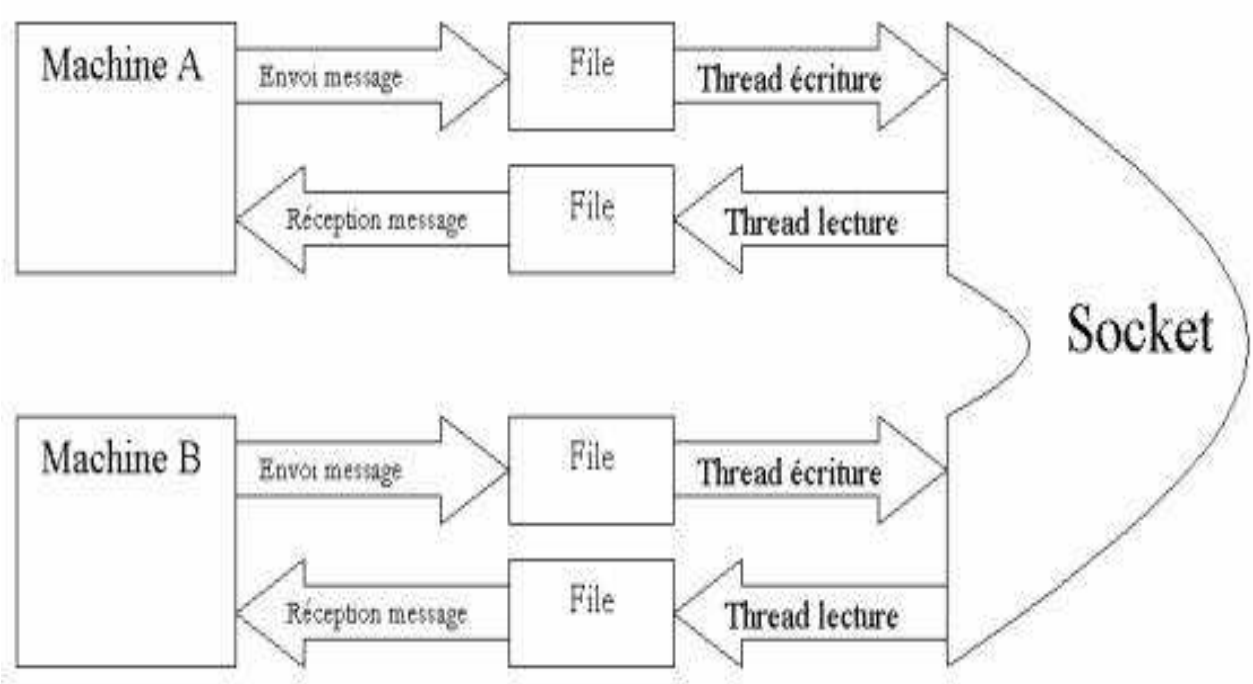
→Tout d'abord, le serveur se met sur écoute.

→Dès qu'un client demande une communication, les différences avec le TCP monotâche apparaissent. Le serveur va lancer l'exécution de tâches.

Une tache est un programme qui ne s'occupe que d'une seule action.

Par exemple dans la communication, le serveur va lancer une tâche qui ne va s'occuper que d'écouter si un client appelle. Il va aussi lancer une seconde tâche qui, elle, ne servira qu'à envoyer des trames.

Chaque tâche est spécialisée pour faire une seule et unique chose. Grâce à l'exécution de plusieurs tâches en simultané on peut recevoir et envoyer des données en même temps.

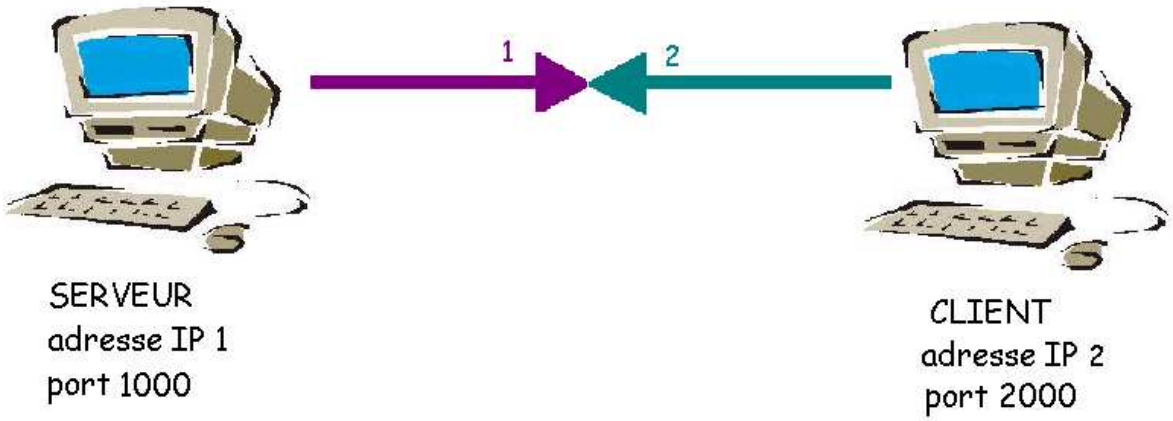


5°) UDP/IP, TCP/IP : quelles sont les différences ?

le protocole TCP, opère un contrôle de transmission des données pendant que la communication est établie entre deux machines. dans un tel schéma, la machine réceptrice envoie des accusés de réception lors de la communication, ainsi la machine émettrice est garante de la validité des données qu'elle envoie. Les données sont ainsi envoyées sous forme de flot. TCP est donc un protocole orienté connexion

Quand le user datagram protocol (UDP), il s'agit d'un mode de communication dans lequel la machine émettrice envoie des données sans prévenir la machine réceptrice, et la machine réceptrice reçoit les données sans envoyer d'avis de réception à la première. Les données sont ainsi envoyées sous forme de blocs (datagrammes). UDP est donc un protocole non orienté connexion

Voyons comment marche UDP plus précisément :



Etape 1 :

Le serveur doit être mis en écoute, c'est à dire qu'il a ouvert son port de réception.

Etape 2 :

Le client demande une requête au serveur en déposant un datagramme chez le serveur. Dans l'entête du datagramme il aura mis les données correspondant à son socket. Mais il ne va pas vérifié si le serveur est déjà connecté.

Le serveur va répondre de la même façon.

6°) CHOIX

Parmi les protocoles que nous venons de présenter, nous avons choisi le protocole UDP/IP.

Notre choix a été guidé tout d'abord par la vitesse d'envoi et de réception des données : le protocole UDP/IP est un protocole sans contrôle donc plus rapide.

Nous avons donc préféré la vitesse de transmission à la fiabilité des données. Travaillant avec des données non critiques (le bras robot n'est pas un bras un outil de chirurgie !) cela ne prête pas à de conséquences graves. Il faut ajouter que l'on envoi souvent des données donc une erreur parmi plusieurs envoi devient négligeable.

De plus, lors du contrôle du bras robot avec la télécommande(dans le cas où le bras devient serveur : il reçoit les données) il faut programmer en FESTO comment traiter les données. Cela devient extrêmement compliqué si il faut en plus gérer les erreurs de transmission.

II) FESTO : le client de la liaison Automate Pc

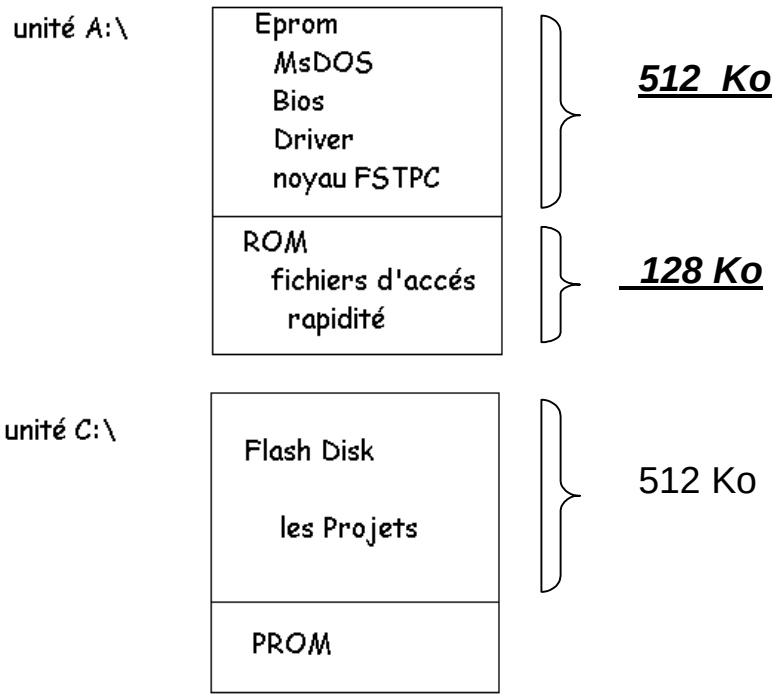
1°) Qu'est que FESTO

Les IPC (industrial PC) FESTO, sont des automates programmables donc ils correspondent à un microprocesseur avec de la mémoire mais aux quels il faut rajouter la possibilité d'ajouter des modules comme :

- module d'entrées sorties TOR
- module d'entrées sorties analogiques
- compteurs
- temporisation
- module de communication

Le système d'exploitation qui est installer sur les IPC est MsDOS 6.0. Mais celui-ci n'est pas suffisant pour faire fonctionner des automates programmables industriels : Ce n'est pas du temps réel. C'est pourquoi les automates sont équipés d'un noyau temps réel : FSTPCR22.

Voici comment est faite la mémoire des IPC :



Dans ces IPC, nous avons un accès un tableau de données avec dedans :

- des entées (0 à 255) : les IW
- des sorties (0 à 255) : les OW
- des flags (0 à 9999) : les FW
- des compteurs (0 à 255)
- des timmers (0 à 255)
- des registres (0 à 255)

Dans notre projet, nous avons ajouté un module : une carte Ethernet ce qui nous à permis de pouvoir être le client d'une communication UDP/IP.

Pour programmer ces IPC, nous avons un logiciel de développement : FST 4. Il est relié à l'automate par une liaison série.

2°) Les programmes FST

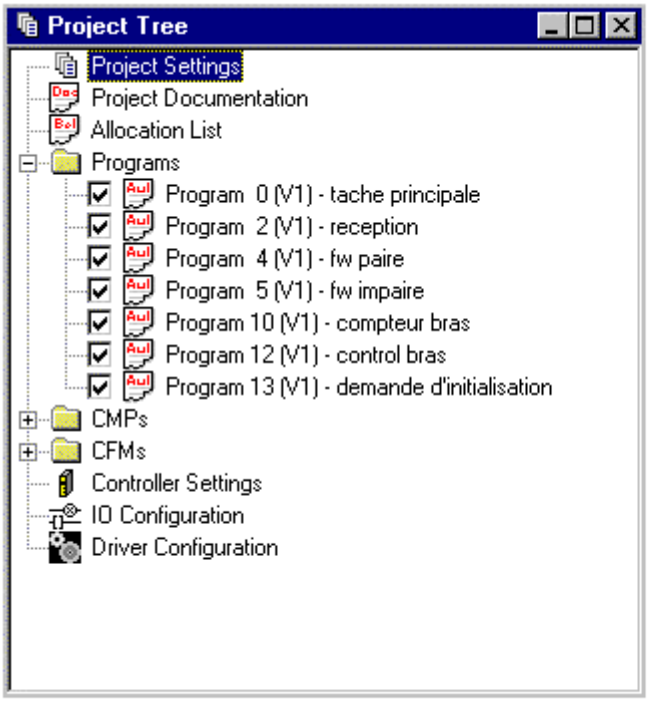
Les programmes se déroulent en pas. Chaque "step" commence par une condition (attention, si on veut mettre une valeur en décimale il faut mettre un V devant : V0), elle suivit d'une action. Si la condition n'est pas réalisée, l'action n'est pas exécutée et on passe à la condition du pas suivante. Pour changer de pas, il faut que la dernière condition soit vraie, sinon on reste bloqué dans le step.
Par exemple

STEP *"nom du step1"*
IF *"condition1"*
THEN *"action1"*

IF "condition2"
THEN "action2"

STEP "nom du step2"

Les IPC peuvent être programmés en multitâche. A partir du programme P0 on peut lancer d'autres tâches (jusqu'à 64 au maximum), des CMP (call module program) ou des CFM (call function module) ces deux dernières étant des tâches appelées dans un programme avec des paramètres, exécutées puis elles reviennent dans le programme. On peut les comparer aux procédures de la programmation en pascal.
On peut voir le multitâche dans le project tree



Quelques mots clé utiles :

nop: (not operand) sert quand on veut q'une condition soit toujours vraie (par exemple la dernière condition d'un pas) : IF NOP
Ou sert quand on ne veut rien faire et juste attendre une condition : then nop

jmp to : pour aller directement au début d'un pas : JMP TO fin

reset et set : servent à mettre à 1 ou à 0 des bits ou des tâches.

with: mot utilisé pour passer les paramètres des CMP et CFM.

Load : sert à chargées des données

3°) Notre protocole :

Nous avons mis au point un protocole de communication. Ce protocole est nécessaire pour savoir ce que l'on reçoit dans les paquets UDP. Notre protocole est constitué de quatre octets :

- 1° octet : le type d'entrée sortie que nous envoyons, symbolisé par le code ASCII d'une lettre (I ou O)
- 2° octet : le numéro de l'entrée sortie envoyée
- 3° octet : le poids faible des données correspondant à l'I/O
- 4° octet : le poids fort des données

Nous avons décidé d'inverser les poids forts et faibles des données pour éviter un traitement des données à la transmission car la fonction UDP_send de festo envoi les données octet par octet.
Il faut donc faire attention, quand on utilise le protocole, on ne doit pas programmer de la même façon si on est sous delphi ou sous festo.

Notre structure de données

Notre projet en lui-même n'utilise pas d'entrées sorties, nous devons juste les surveiller. Par contre nous utilisons beaucoup de flags word pour recopier toutes les I/O et le protocole correspondant à chaque I/O. Il y a 512 entrées sorties donc nous utilisons 1024 flags word : de FW5000 à FW 6023.
Nous utilisons de les premiers flags (de FW0 à FW11) pour la réception des paquets UDP.
Nous utilisons des registres que nous utilisons comme variables internes (de R100 à R114).

Un programme festo composé de 4 tâches :

- **UNE TACHE INITIALE :** elle prépare une table IP (c'est un CMP) avec l'adresse du serveur Delphi. Pour cela elle a besoin de plusieurs paramètres :
  - V1 : on écrit dans la table
  - V1 : numéro de l'index dans la table
  - V193, V50, V179, V217 : les quatre nombres de l'adresse IP.

  A partir de ce moment, le serveur correspond à un index dans cette table : cela simplifie la programmation à venir : on n'a pas besoin de rentrer les 4 numéros de l'adresse à chaque fois, mais juste l'index correspondant.
  A la fin, ce programme lance les autres tâches

- **UNE TACHE DE RECEPTION :** elle installe le handler pour recevoir les paquets UDP/IP(aussi un CMP) le handler prépare donc une place dans les flags word pour pouvoir recevoir les données : 10 flags pour l'entête, pour nous il y avait ensuite 2 flags de données (protocole de 4 octets). le CMP à donc besoin de
  -V2000 numéro du port local
  -V1 numéro du premier flag de réception

  Il faut faire attention au port local, celui-ci doit être assez grand sinon la réception ne se fait pas.
  Ensuite elle surveille l'arrivée des données et à chaque réception, pour cela elle regarde le flag 0 (dans l'entête, c'est celui qui correspond au nombre de communication effectué depuis le début de la "liaison" donc c'est le seul dont on est sûr qu'il est modifié à chaque fois).
  Dès qu'un paquet arrive, il est partagé en trois parties (lettre, numéro, données). On vérifie que la lettre est bien un 'O'.

Ensuite cette tâche s'occupe de rangement des données dans la table d'entrée sorties par l'intermédiaire d'un module : output (que nous avons nous même fabriqué). Ce module recherche quel est le

numéro et modifie la sortie correspondant au numéro. De plus chaque entrée sortie est recopiée dans son flag word image pour que l'on puisse voir s'il est modifié par la suite et pour ne pas envoyer une modification en trop car on surveille sans arrêts.

- **UNE TACHE D'INITIALISATION DES FLAGS WORD :** elle prépare les flags word qui vont être envoyé au serveur. On y met dans le premier octet le numéro de l'entrée sortie et dans le deuxième, on y inscrit un 'I' pour input ou un 'O' pour output. Pour cela on utilise une CFM : WINDEXMW. Cette fonction nous permet de faire une boucle et d'incrémenter un registre : on gagne beaucoup de ligne de code par rapport à un accès au I/O. cette fonction nécessite du numéro de flags et de sa valeur. On peut remarquer que l'on à inverser le numéro et la lettre parce que L'IPC envoi les données octet par octet donc les poids forts et les poids faibles sont inversé.
  Entre chaque mot, on a laissé la place pour les données : à une entrée sortie correspond donc deux flags word.

- **UNE TACHE DE RECOPIE DES ENTREES SORTIES :** elle regarde sans arrêt les entrées sorties que nous utilisons (de 0 à 15). Vu que ce n'est pas possible d'accéder aux entrées sorties avec un index comme pour les flags, nous sommes obligés de faire autant de morceaux de code qu'il y a de mots à surveiller. Si on enregistre une modification, le programme la recopie dans le flag word correspondant à l'I/O changé avant de les envoyer au serveur. Comme précédemment, nous avons recopié le code pour les 16 entrées et les 16 sorties.

Pour envoyé les données, nous avons utilisé un autre call module programme : UDP_send : celui-ci requière plusieurs information pour envoyer ses données :

- numéro du port local

- adresse IP remontante (donné par l'index dans la table IP)

- le port du serveur

- le nombre d'octets à envoyer (dans notre cas toujours 4)

- le numéro du premier flag word à envoyer (sur festo on ne peut envoyer des données en UDP qu'à partir de flag word.

Il faut faire attention de bien renvoyé le "FU32" à la fin du module, parce qu'on appel, à l'intérieur du module, un autre module donc on modifie la variable FU32.

Ce programme de recopie des I/O est bouclé sur lui-même et recommence indéfiniment.

5°) Avantages et Inconvénients

AVANTAGES :

le protocoles est très court : vitesse importante.

Il est adaptable aux IPC et aux PC

C'est le client qui envoie des données quand il y a des changements et non pas le serveur qui fait des demandes tout le temps : pas de transfert de paquets inutiles.

INCONVENIENTS :

Il n'y a pas de champs prévu pour la récupération de donnés dans l'ordre : il faudrait l'ajouter car ni TCP ni UDP ne l'on prévu.

UDP n'est pas le mieux adapté en cas de problèmes on n'est pas sur qu'une demande d'arrêt d'urgence (par exemple) soit arrivée.

III Partie logicielle

1°) Présentation DELPHI

Une partie de notre projet correspond à la récupération de données en provenance de l'automate et à l'envoi d'ordres.
Cette partie ce fait via un ordinateur. C'est en programmation Delphi qu'ont été fait les exécutables.

Delphi est un langage de programmation dit orienté objet. Dans DELPHI, le programmateur utilise des objets (c'est à dire un ensemble de méthodes et de fonctions) qui ont déjà été programmés. DELPHI est un langage de programmation inspiré du langage PASCAL. Contrairement à ce dernier, il est fondé sur des notions d'évènements : le clic à un endroit précis…. Il permet de créer simplement de belles interfaces graphiques tout en disposant d'un puissant langage de programmation.

En Delphi, on construit des projets. Un projet est constitué d'unités qui ressemblent à des programmes Pascal et de fiches qui définissent l'interface graphique. Dans un projet, une unité est associée à chaque fiche. Chaque unité et chaque fiche seront stockées dans des fichiers différents (extension PAS pour les unités, DFM pour les fiches ; le projet ayant l'extension DPR).

Pour notre projet nous avons utilisé Internet mais les composants fournis initialement par le logiciel DELPHI6.0 ne nous convenait pas. Nous avons donc décidé d'utilisé des composants programmés par F.PIETTE.

2°) Composants PIETTE

Ce sont des objets programmés que l'ont peut utiliser librement. Dans ces composants il se trouve des sockets de type Winsock(Wsocket pouvant être utilisé en TCP, UDP, DNS…), des composant FTP, des composants HTTP… En plus des composants, nous trouvons divers exemples qui fonctionnent, avec de nombreuses explications(exemple de chat…).

Les composants sont nommés ICS(Internet Component Suite) avec le code source complet pour DELPHI. Dans notre projet, nous avons utilisé le composant Wsocket.

3°)Partie DELPHI du projet

Comme indiqué dans l'introduction, notre projet consiste à établir un lien entre un ordinateur et un automate. Toute la partie gérée par l'ordinateur à été programmée en DELPHI.

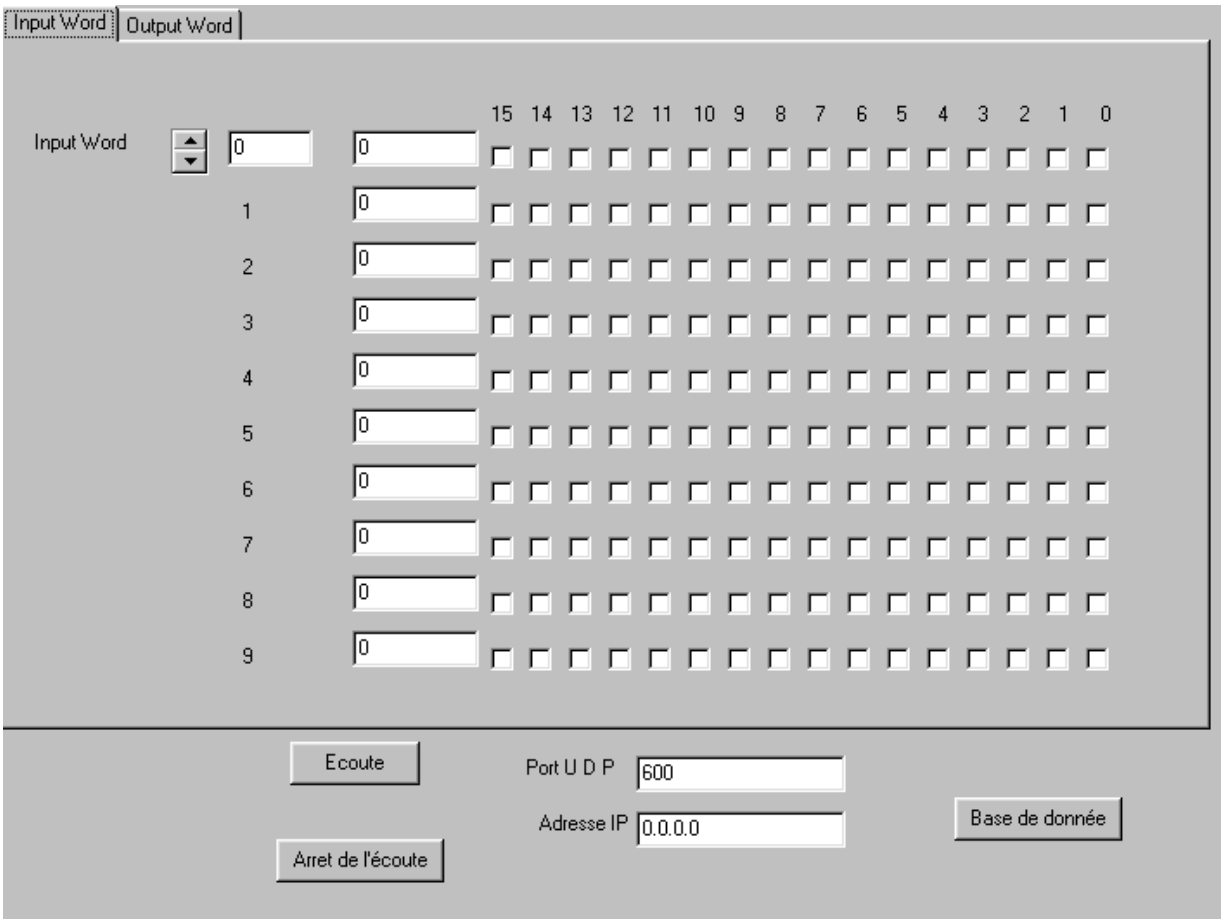
Nous vous présenterons la forme principale grâce à laquelle nous pouvons recevoir et envoyer des données, puis nous vous présenterons une "télécommande " qui permet de contrôler le bras robot.

La forme dite " principale " est théoriquement invisible à l'utilisateur non initié (comme des visiteurs lors des Journées Portes Ouvertes de l'IUT) car nous y trouvons de nombreuses informations techniques et la forme visuelle peut ne rien signifier à une personne novice.

La télécommande, elle, a été conçue et prévue pour être vue et utilisée par n'importe qui (simple d'utilisation et aucune information technique).

Il y a cependant de nombreuses correspondances entre ces deux programmes, notamment l'utilisation de composants sockets identiques (issus des composants de F.PIETTE : les Wsocket) et des objets plus " classiques " comme les objets boutons.

4°) Forme Principale



Cette forme peut se diviser en deux partie : la partie pour la communication(en bas) ainsi la partie affichage et modification.

Dans la partie de communication se trouve trois boutons ainsi que deux champs à remplir.
De plus, invisible à l'utilisateur, se trouve le composant socket qui permet la réception des données

a°) Fonctionnement du socket de réception(ou serveur)

Le serveur est initialement inactivé et ne regarde pas si quelqu'un souhaite communiquer. Pour pouvoir lui parler, deux informations sont primordiales :
→ tout d'abord connaître son adresse IP
→ enfin connaître le port sur lequel le serveur écoute (il n'écoute que sur un port, voir dans le 1)

Pour activer le serveur il suffit d'utiliser la commande :

Wsocket.Listen ;

Celle-ci ordonne au serveur d'écouter toutes les communications qui arrivent sur le port précisé.
Cette ligne de commande s'exécute lors du clic sur le bouton : " Ecoute "

Une fois que le serveur est mis en écoute, il vérifie l'arrivée de données.
S'il en arrive, le socket déclenche l'événement
" WsocketDataAvailable ".

C'est dans cet événement que nous recevons et traitons les paquets de données reçues.

Dans la ligne de code :

long:=Wsocket1.ReceiveFrom(@tab,sizeof(tab),adresse,longadresse) ;

Nous mettons dans la variable long toutes les données qui ont été envoyées. Celles-ci sont mise sous la forme d'un entier.

La boucle qui suit permet de vérifier que l'ont à bien reçu des données. Si tel n'est pas le cas, on ne fait rien. Lorsque long n'est pas vide, on effectue le " pelage " des données reçues.

A chaque envoi de donnés, nous envoyons quatre paquets de huit bits. Nous avons utilisé un certain protocole :
→ les huit premiers bits correspondent au code ASCII du o ou du i majuscule
→ les huit suivants correspondent au numéro de l'entrée ou de la sortie à modifier
→ enfin les seize derniers sont les données.

Revenons en à notre " pelage ". Les données transitent sous la forme du code ASCII, c'est à dire qu'un I est codé sous la forme binaire de 73. Pour retrouver la forme voulue, nous utilisons la fonction TAB qui du code ASCII permet de repasser au code normal(il le transforme en caractère).

Si les huit premiers bits reçus ne correspondent pas à I ou O majuscule, alors ont fait apparaître un message d'erreur. La valeur 007 apparaît sur la première ligne d'affichage. Cela que nous regardions les entrées (Inputs) ou les sorties (Outputs). De plus sur un label se trouvant sur la partie Output, on fait apparaître les vingt-quatre derniers bits reçus précédés de la lettre e (cela permet de vérifier si seul le premier caractère était erroné.)

Après avoir fait cette première vérification on met dans un entier(mot) la valeur de l'entrée ou de la sortie à modifier.
Pour récupérer la bonne valeur de modification la ligne de code est :

valeur :=ord(tab[3])*256 + ord(tab[2]) ;

valeur est un entier. ORD est une fonction qui transforme un caractère en valeur entière.

Dans le protocole que nous avons utilisé le poids fort et le poids faible sont inversés. Donc pour récupérer la bonne valeur de modification, il suffit de multiplier les huit bits de poids fort par 256. Cela a pour effet de les décaler :
→ avant multiplication par 256 : 0000 0000 1111 1111
→ après multiplication par 256 : 1111 1111 0000 0000

Il ne manque plus qu'à y ajouter la valeur du poids faible, c'est à dire : Ord(tab[2]).

Nous venons de récupérer toutes les données qui nous ont été transmises, nous les avons traduites de l'ASCII en caractères utilisables, nous les avons découpées et mise dans des variables.

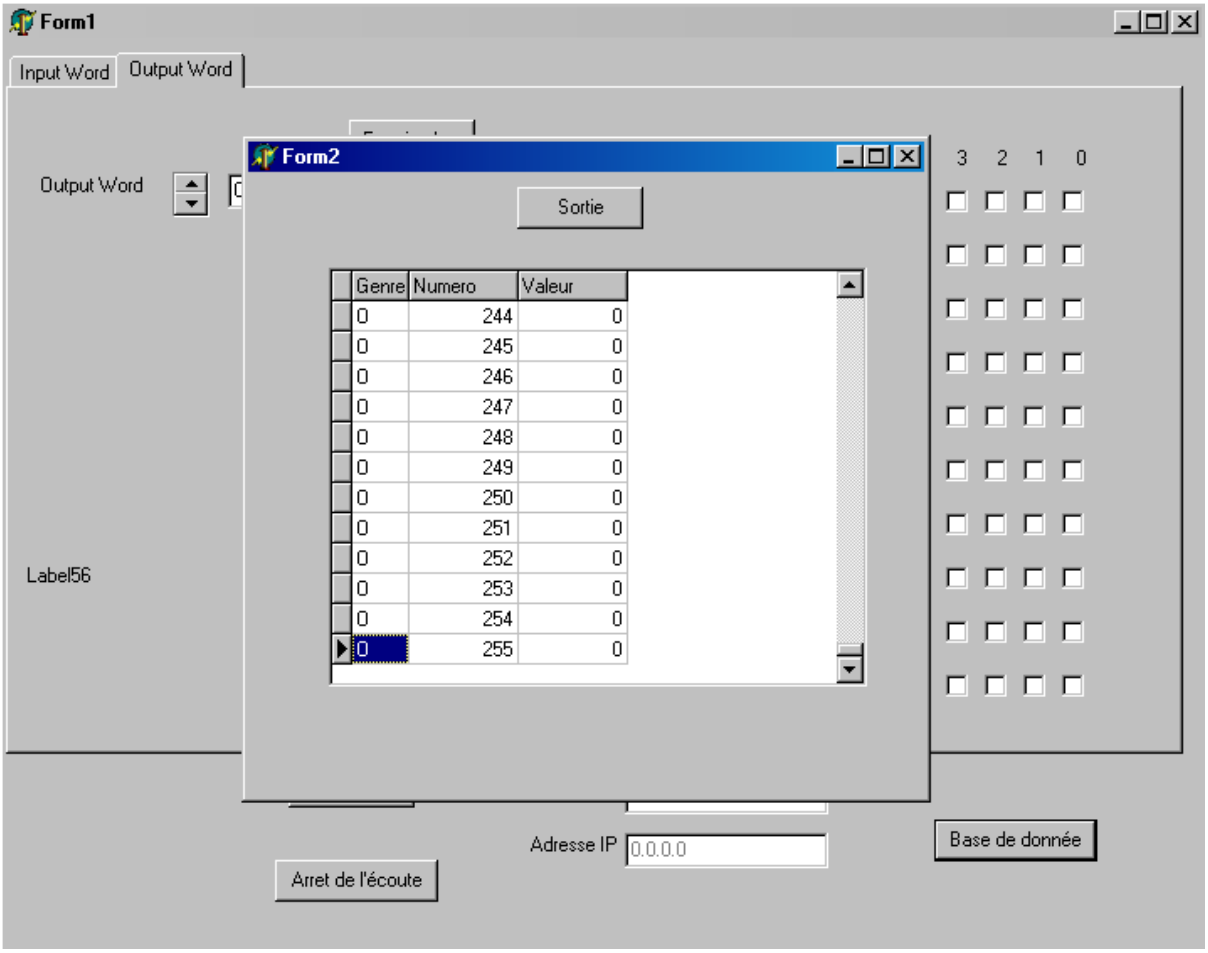
Pour faciliter l'affichage et la gestion des données, nous avons créer deux tableaux d'entier :
→dans Tablo_Iw se trouvent toutes les données relatives aux entrées
→dans Tablo_Ow se trouvent toutes les données relatives aux sorties.

En fonction du premier caractère reçu nous allons modifier soit la table des entrées, soit la table des sorties.

Celle-ci sont utilisées de la manière suivante. On met la valeur reçue et transformée en entier dans la case comportant le même numéro que celui mis dans la variable MOT.

Dans l'application finale, nous avons gardé le même protocole. Les huit premiers bits correspondent aux entrées ou aux sorties. Les huit suivants se rapportent à un moteur du bras robot car chacun possède un numéro différent. Les seize derniers bits sont les données c'est à dire les modifications à apporter à l'affichage.

Nous avons crée une base de donnée grâce au composant DELPHI nommé TTable. De plus la gestion de l'affichage se fait grâce au composant DBGrid. Nous l'avons placée sur une seconde forme qui permet de voir ce qui se trouve dans cette base de données.



Toujours en fonction du premier caractère reçu, nous utiliserons deux procédures effectuant un travail plus ou moins identique :

→BaseDonneeIW(mot,valeur)

→BaseDonneeOW(mot,valeur)

Pour ouvrir la base de données et donc pouvoir écrire à l'intérieur il faut utiliser la commande :

Table1.open ;

Pour pouvoir écrire la commande *Table1.Append* est nécessaire.

Puis pour modifier un des champs de la base de données, on utilise la fonction *Fields* et un numéro qui correspond aux champs à modifier.

Une fois que la modification de la base de donnée a été faite, il nous faut nous occuper de l'affichage. La partie supérieure de la forme finale est en faite constituée de deux onglets l'un sur lequel on affiche les Inputs et un second sur lequel on affiche les Outputs. Il nous faut donc ne modifier que ce qui est nécessaire et laisser l'affichage sur l'autre onglet tel quel.

Sur chacun des onglets l'affichage s'effectue sur 10 lignes ou se trouve un edit par ligne et 16 checkbox.

Dans l'edit se trouve une valeur en décimal, et les checkbox sont cochés de manière à faire apparaître en binaire la même valeur que celle en décimale.

La procédure RafraichirIW effectue ses modifications sur l'onglet des Inputs alors que RafraichirOW les effectue sur l'onglet des Outputs. De même pour les procédures CheckboxIW en Inputs et CheckboxOW en Outputs.

Les procédures RafraichirIW et RafraichirOW servent à mettre dans les 10 edit, respectivement en Input et en Output, les valeurs se trouvant dans le tableau des données.

Après avoir modifié l'affichage en décimal, on le modifie en binaire grâce aux fonctions CheckboxIW et CheckboxOW.

b°)Envoi de données

S'il y a un socket qui s'occupe de l'écoute et de la réception des données, un second lui va s'occuper d'envoyer les modifications à effectuer. Tous les changements ne peuvent se faire que sur des Outputs, c'est donc sur l'onglet correspondant que se trouve le socket d'envoi.

Il va donc être le client de la liaison. Pour envoyer les données, comme à tous client, il a besoin de quelques informations : l'adresse IP et le port.

Le changement à faire peut être écrit dans un des Edit (en décimal) ou en cochant les checkbox (les modifications seront reportées dans les Edit avant l'envoi).

Une fois les modifications effectuées il suffit de cliquer sur le bouton ENVOI.

Dès que l'envoi est ordonné, on va comparer les Edit avec la table des sorties. On va ainsi repérer les données et les numéros qui ont changés et n'envoyer que ceux-là.

On défini au socket :

→ le protocole utilisé, pour nous UDP (Wsocket2.Proto)
→ l'adresse à laquelle on écrit (Wsocket2.Addr)
→ le port sur lequel on écrit (Wsocket2.Port)

Après on ouvre le socket (Wsocket2.coonnect), puis on envoi les données (Wsocket.sendstr).

Pour l'envoi de ces données, il faut savoir que le composant envoi en premier la valeur la plus proche de la parenthèse ouvrante.

La fonction CHR est toujours utilisée pour éviter le code ASCII.

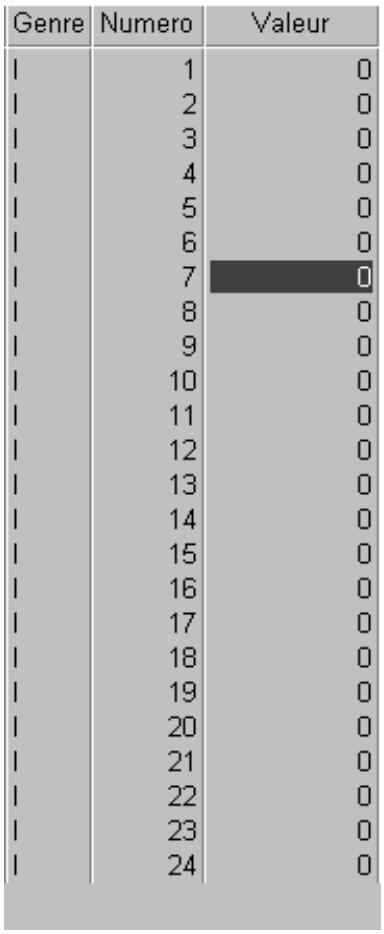
Une fois l'envoi fait, il suffit de refermer le serveur en utilisant la commande CLOSE.

Il ne faut pas oublier de faire les modification dans la table des sorties ET dans la base de donnée.

5°) Particularités

La base de donnée est crée à chaque ouverture de la forme et est vidée à chaque fermeture grâce à la commande (Base.Empty).

C'est lors du click sur le bouton ECOUTE qu'est générée une base de données de 256 entrées et de 256 sorties, le nombre 256 étant une variable globale. Cette base de donnée est initialement à 0 partout pour le champ Valeur.



Genre	Numero	Valeur
I	1	0
I	2	0
I	3	0
I	4	0
I	5	0
I	6	0
I	7	0
I	8	0
I	9	0
I	10	0
I	11	0
I	12	0
I	13	0
I	14	0
I	15	0
I	16	0
I	17	0
I	18	0
I	19	0
I	20	0
I	21	0
I	22	0
I	23	0
I	24	0

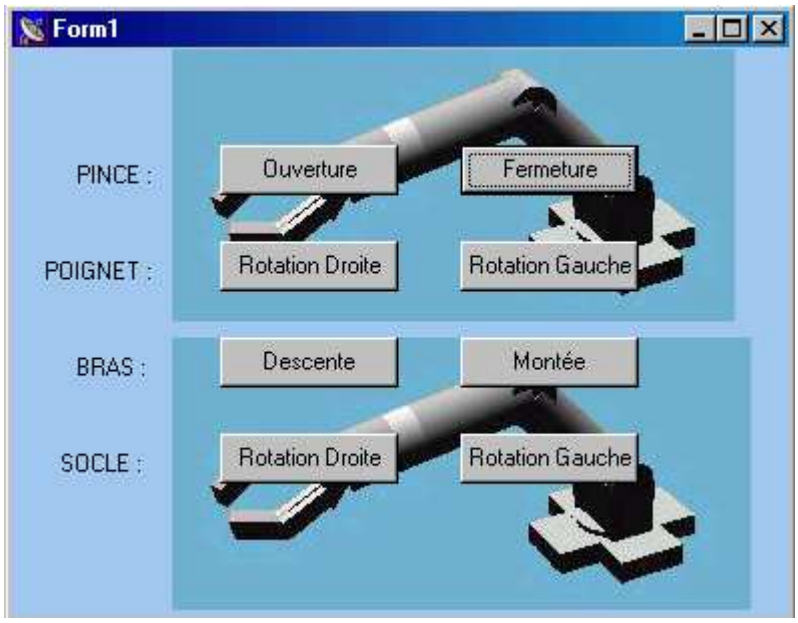
C'est lors du click sur le bouton ARRET ECOUTE qu'est vidée la base de données. Pour éviter des conflits, cette commande est désactivée lors du démarrage. Imaginons que l'on clique sur ARRET ECOUTE avant d'avoir appuyé sur ECOUTE : on va vouloir vider une base qui n'existe pas !

En ce qui concerne l'affichage, celui-ci va de 0 à 255 pour les Inputs et pour les Outputs (chiffres modifiable dans le code). On peut se déplacer d'un en un en utilisant le bouton flèche ou bien taper la valeur sur laquelle on veut aller.

Si le chiffre tapé est supérieur à la valeur maximale d'affichage(ici 255) alors la ligne d'affichage la plus basse prendra , ici, la valeur 255.

De même, si on tape un nombre négatif (ou inférieur à la valeur minimale d'affichage), alors la ligne d'affichage la plus haute prendra la valeur 0 (ou la valeur minimale d'affichage)

6°) Télécommande



Cette télécommande utilise le même socket que celui qui permet l'envoi de données dans la forme principale. Pour lui faire envoyer un ordre, il suffit de rester cliqué sur un des huit boutons de la forme.

A chaque fois que l'on appuie sur l'un d'entre eux, il se déclenche l'événement *BoutonCliqué.Onmousedown* . Lorsqu'il est relâché, c'est l'événement *BoutonCliqué.Onmouseup* qui a lieu.

Pour l'envoi des données le même protocole que précédemment à été suivi (O ou I, numéro, données).

Cependant ces données sont particulières : elles ne servent qu'a activer un bit précis.

Exemple de données : 0000 0100

Tous les moteur du bras à commander sur trouvent définis sur un même mot de sortie. Pour allumer un des moteur il suffit de

mettre le bit correspondant à 1. Si on reste cliqué le bit qui correspond au moteur est à 1 donc le moteur tourne.

Dès que l'on " déclique ", on effectue à nouveau un envoi composé uniquement de 0

Exemple : 0000 0000

Cela va éteindre le moteur qui était actif.

Dans cette forme a été insérée deux images en 3dimensions du bras robot à diriger pour rendre la visualisation plus agréable.

CONCLUSION

Notre projet a été mené à bien. Après le regroupement des trois binômes travaillant sur le projet du bras robot, nous avons pu présenter, lors des JPO, une application qui fonctionnait.

Le bras robot et le logiciel de supervision étaient prêt, et après quelques petits réglages tout marchait correctement. Il reste cependant quelques améliorations et quelques corrections à effectuer.

Nous avons donc appris à travailler avec des contraintes de temps, de fonctionnement, mais aussi à travailler en équipe pour réussir à créer de A à Z une application que nous n'aurions jamais eut le temps de faire seuls.

Après avoir réussi à faire fonctionner ce projet on peut imaginer diverses possibilités de globalisations, notamment faire l'affichage en temps réel du bras robot sur plusieurs ordinateurs en même temps. On peut aussi penser à multiplier les bras robots…

Ce projet est destiné à devenir un TP de la section RLI dans un avenir proche.