**UNIVERSITE D'ANTANANARIVO**

**ECOLE SUPERIEURE POLYTECHNIQUE**

**Département :** ELECTRONIQUE

**MEMOIRE DE FIN D'ETUDES**

**EN VUE DE L'OBTENTION DU DIPLÔME D'INGENIEUR**

Option : **Electronique Automatique**

# ETUDE DE LA PROGRAMMATION RESEAU SOUS WINDOWS

Présenté par : **M^me^ NIRINASOA Viviane**

Soutenu le 07 juillet 2009

**N° 02 /EN/EA/08**

Année universitaire: 2007 - 2008

**UNIVERSITE D'ANTANANARIVO**

**ECOLE SUPERIEURE POLYTECHNIQUE**

**Département :** ELECTRONIQUE

**MEMOIRE DE FIN D'ETUDES**

**EN VUE DE L'OBTENTION DU DIPLÔME D'INGENIEUR**

**Options: Electronique Automatique**

**N° 02/EN/EA/08**

# ETUDE DE LA PROGRAMMATION RESEAU SOUS WINDOWS

**Présenté par :** Madame NIRINASOA Viviane

**Président de jury :** Monsieur RASTEFANO Elisée

**Examinateurs :** Madame RABEHERIMANANA Lyliane Irène

Monsieur RABESANDRATANA ANDRIAMIHAJA Mamisoa

Monsieur RATSIMBAZAFY Guy Predon

**Rapporteur :** Monsieur RAKOTONDRASOA Justin

Soutenu le 07 juillet 2009

**Année universitaire** : 2007-2008

« Tsy ny tenanay akory no
ampy hahahevitra zavatra toy ny
avy amin'ny tenanay ;
Andriamanitra ihany no
ihavian'ny fahampianay »

*II Korintiana 3, 5*

# *REMERCIEMENTS*

*D'abord, je loue mon Dieu tout puissant, de m'avoir donné la force, la santé et le courage…pour que je puisse mener à terme ce travail.*

*J'adresse me vifs remerciements à :*

*Monsieur RASTEFANO Elisée, chef du département Electronique qui m'a formé et m'a guidé tout au long de cinq années d'études.*

*Monsieur RASTEFANO Elisée soit également remercié d'avoir bien voulu présider la soutenance de ce présent mémoire.*

*Monsieur RAKOTONDRASOA Justin enseignant chercheur à l'E.S.P.A, qui n'est autre que le rapporteur du présent mémoire. Vous avez consacré une grande partie de vos précieux temps pour ma guider tout au long de l'élaboration et de la réalisation de ce mémoire.*

*Aux membres du jury en personnes de :*

*Monsieur RABESANDRATANA ANDRIAMIHAJA Mamisoa*

*Monsieur RATSIMBAZAFY Guy Predon*

*Madame RABEHERIMANANA Lyliane Irène*

*qui ont jugé la forme et le fond de cet ouvrage malgré leurs occupations socioprofessionnelles.*

*A tous les responsables des centres de documentations (bibliothèque de Vontovorona, bibliothèque de l'AFITEL de Vontovorona, bibliothèque d'Ankatso), et les responsables du laboratoire de notre département.*

*Enfin, j'ai une pensée pleine de gratitude à mes parents et à tous les membres de me famille, ainsi qu'à mes amis et collègues qui m'ont toujours encouragés dans mes études.*

*QUE DIEU VOUS BENISSE !*

*Viviane*

# Résumé

Cet ouvrage présente l'étude de base de la programmation sous Windows et plus particulièrement la programmation réseau. L'API  Windows étant encore un sujet presque inconnu par le grand public et même ignoré par certains développeurs, l'objectif de ce mémoire permet de montrer son importance dans le développement d'applications par le biais de Winsock. Ce dernier est une API spécialisée dans la communication réseau utilisant les sockets.

Au terme de cette étude, on va développer cet API en utilisant la bibliothèque Winsock

# Table des matières

# Liste des tableaux

# Liste des figures

# Liste des abréviations

| | |
|---|---|
| **ACK** | Acknowledgment flag |
| **ADO** | ActiveX Data Object |
| **API** | Application Programming Interface |
| **CLI** | Common Language Infrastructure |
| **CLR** | Common Language Runtime |
| **DLL** | Dynamic-Link Libraries |
| **DOS** | Denial Of Service |
| **FCL** | Framework Class Library |
| **FIN** | Finish flag |
| **FTP** | File Transfer Protocol |
| **GDI** | Graphics Device Interface |
| **HTTP** | Hyper Text Transfer Protocol |
| **ICMP** | Internet Control Message Protocol |
| **IHL** | Internet Header Lenght |
| **IP** | Internet Protocol |
| **IPv4** | Internet Protocol version4 |
| **IrDA** | Infrared Data Association |
| **ISO** | International Organization for Standardisation |
| **J2EE** | Java 2 Enterprise Edition |
| **JIT** | Just In Time |
| **MCI** | Media Control Interface |
| **MS** | Microsoft |
| **MSIL** | Microsoft Intermediate Language |
| **LAN** | Local Area Network |
| **.NET** | DotNet |
| **ODBC** | Open Database Connectivity |
| **OLE BD** | Object Linking and Embedded Database |
| **OSI** | Open Systems Interconnection |

| | |
|---|---|
| **PE** | Portable Executable |
| **Pf** | Protocol family |
| **PSH** | Push flags |
| **RST** | Reset flag |
| **SDK** | Software Development Kit |
| **SE** | Second Edition |
| **SGBD** | Système de Gestion de Base de Données |
| **SMTP** | Simple Mail Transport Protocol |
| **SQL** | Structure Query Language |
| **SYN** | Synchronize flag |
| **TCP** | Transmission Control Protocol |
| **TSE** | Terminal Server Emulation |
| **TTL** | Time To Live |
| **UDP** | User Datagram Protocol |
| **URG** | Urgent flag |
| **USB** | Universal Serial Bus |
| **VFW** | Video for Windows |
| **WAN** | Wide Area Network |
| **Windows Me** | Millennium |
| **Windows NT** | New Technology |
| **WinInet** | Windows Internet API |
| **WinSock** | Windows Sockets API |

# Introduction

Actuellement, Microsoft Windows est le système d'exploitation le plus répandu sur micro-ordinateur, que ce soit pour utilisation en entreprise ou personnelle. Il est également de plus en plus présent sur d'amples architectures comme les appareils mobiles, les consoles de jeu et autres systèmes embarqués.

Windows est un système d'exploitation composé d'un noyau extrêmement portable et d'une interface graphique agréable. Le fait qu'il soit presque entièrement écrit en C (avec quelques parties du noyau spécifiques à chaque machine seulement écrites en assembleur et une partie de l'interface graphique en C++) est la principale raison de cette portabilité. Windows fournit aux applications une interface de programmation communément appelée « Win32API » ou « API Windows » (*Application Programming Interface).* L'API Windows permet aux programmeurs de contrôler et d'interagir avec Windows en utilisant un langage de programmation [8] [9].

Cependant, Windows est moins apprécié dans le domaine du réseau informatique comme les serveurs ou les hébergeurs de site Web. Pourtant, Windows dispose de ressources suffisantes comme les API [14]. Le problème vient du fait qu'il n'intègre pas directement les services et les applications réseau comme les systèmes Unix.

Ce mémoire est intitulé « **Etude de la programmation réseau sous Windows**», a pour but de réaliser des applications d'administrations réseau en utilisant Winsock afin de montrer que Windows est aussi capable de gérer ces services en exploitant les API [10].

Pour atteindre cet objectif, une bonne connaissance de la programmation sous Windows est nécessaire.

Ce présent manuscrit est organisé de la manière suivante : le premier chapitre est consacré à la généralité du système d'exploitation Windows, le second chapitre présente la base de la programmation réseau sous Windows et le troisième chapitre traite la programmation réseau avec Winsock.

# CHAPITRE I.
# VUE GENERALE DU SYSTEME D'EXPLOITATION WINDOWS

## I.1. Historique

Windows est le système d'exploitation commercialisé par la société Microsoft, dont le siège est implanté à Seattle. La société Microsoft, initialement baptisé «Traf-O-Data» en 1972 a été rebaptisée «Micro-soft» en novembre 1975, puis «Microsoft» le 26 novembre 1976.

La première version de Microsoft Windows (Microsoft Windows 1.0) est apparue en novembre 1985. Il s'agissait d'une interface graphique, inspirée de l'interface des ordinateurs Apple de l'époque. Windows 1.0 n'a pas eu de succès auprès du public, pas plus que Microsoft Windows 2.0, lancé le 9 décembre 1987.

C'est le 22 mai 1990 que le succès de Microsoft Windows a débuté avec Windows 3.0, puis Windows 3.1 en 1992 et enfin Microsoft Windows for Workgroups, baptisé par la suite Windows 3.11, comprenant des fonctionnalités réseau. Windows 3.1 ne peut pas être considéré comme un système d'exploitation à part entière car il s'agit d'une interface graphique fonctionnant au-dessus du système MS-DOS.

Le 24 août 1995, Microsoft lance le système d'exploitation Microsoft Windows 95. Windows 95 marque la volonté de Microsoft de transférer des fonctionnalités de MS-DOS dans Windows, mais cette version s'appuie encore largement sur le système DOS 16-bits et garde notamment les limitations des systèmes de fichiers FAT16.

Microsoft commercialise le 25 juin 1998 la version suivante de Windows : Windows 98. Windows 98 intègre nativement d'autres fonctionnalités de MS-DOS mais s'appuie toujours sur ce dernier. D'autre part Windows 98 souffre d'une mauvaise gestion du partage de la mémoire entre processus, pouvant provoquer des dysfonctionnements du système. Une seconde édition de Windows 98 paraît, le 17 février 2000, elle se nomme Windows 98 SE (Second Edition).

Le 14 septembre 2000, Microsoft commercialise Windows Me (*Millenium Edition*), également appelé Windows Millenium. Windows Millenium s'appuie largement sur Windows 98, mais apporte des fonctionnalités multimédia et réseau supplémentaires. D'autre part, Windows Millenium intègre un mécanisme de restauration du système permettant de revenir à un état précédent en cas de plantage. Parallèlement, Microsoft a lancé dès octobre 1992 un

système d'exploitation entièrement 32 bits (indépendant du  système MS-DOS) pour un usage professionnel, à une époque où les entreprises utilisaient essentiellement des mainframes. Il s'agit de Windows NT (*New Technology*). Windows NT n'est donc pas une version ou une évolution de Windows 95, mais un système d'exploitation à part entière.

Le 24 mai 1993 la première version de Windows NT est commercialisée. Il s'agit de Windows NT 3.1, puis Windows NT 3.5 sort en septembre 1994 et Windows 3.51 en juin1995. C'est avec Windows NT 4.0, lancé sur le marché le 24 août 1996, que Windows NT va enfin connaître un réel succès.

En juillet 1998, Microsoft commercialise Windows NT 4.0 TSE (*Terminal Server Emulation*), le premier système Windows à permettre la possibilité de brancher des terminaux sur un serveur, c'est-à-dire d'utiliser des clients légers pour accéder à une session ouverte sur le serveur.

Le 17 février 2000, la version suivante de Windows NT 4.0 est baptisée Windows 2000 afin de montrer la convergence des systèmes NT. Windows 2000 est un système entièrement 32-bits possédant les caractéristiques de Windows NT, ainsi qu'une gestion améliorée des processus et une prise en charge complète des périphériques USB (*Universal serial Bus*) et Firewire.

Puis, le 25 octobre 2001, Windows XP (XP pour « *new user experience* ») fait son apparition. Il s'agit de la convergence des systèmes précédents.

Enfin le 24 avril 2003, un système d'exploitation dédié pour les serveurs est commercialisé par Microsoft : Windows Server 2003.

La figure 1.1 donne un bref aperçu de toutes les versions de Windows depuis sa création.

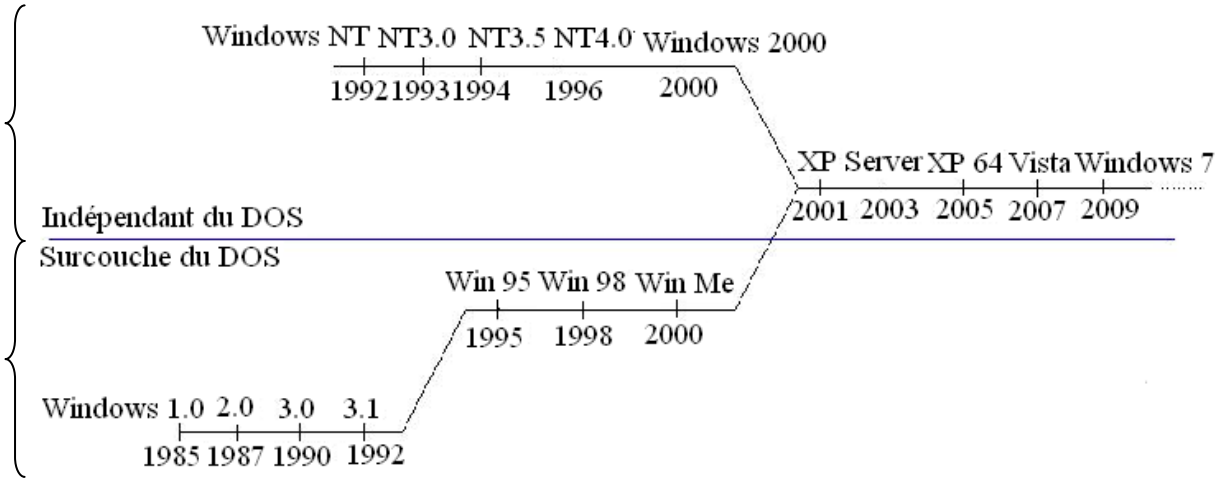


***Figure 1.1:*** Les versions de Windows

## I.2. L'API Windows :

### a. Qu'est-ce qu'une API ?

Une API (*Application Programming Interface*) est un ensemble de fonctions fourni par le constructeur d'un système ou tout simplement d'un logiciel permettant à une application de communiquer avec ces derniers. Ces fonctions sont regroupées dans une ou plusieurs bibliothèques et qui sont accessibles pour tous les développeurs via une documentation fournie par le concepteur lui-même [8].

### b. Qu'est-ce que l'API Windows ?

L'API Windows permet aux développeurs de créer des applications pouvant être utilisées et exploités par toutes les fonctionnalités de Windows. Elle offre donc un contrôle de Windows non plus depuis l'interface utilisateur mais depuis un langage de programmation [8]. Il n'est cependant pas forcément nécessaire de connaître cette API pour développer des applications Windows et d'ailleurs, de nos jours, bon nombre de programmeurs ne l'utilisent pas, et ne l'ont jamais utilisé ou même ignorent tout simplement son existence.

La figure 1.2 donne un aperçu de cette relation entre l'API Windows et le système d'exploitation Windows.

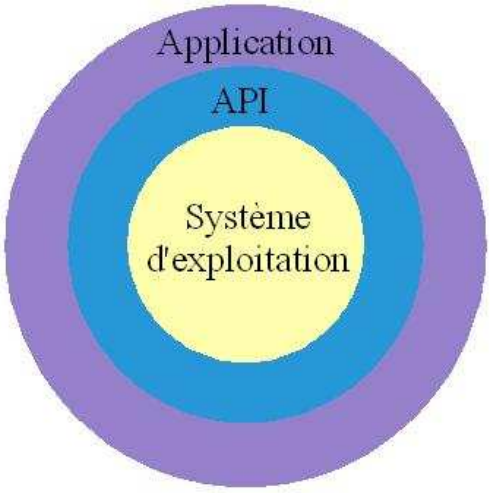


***Figure 1.2:*** L'API Windows

## I.3. « .NET » (*DotNet*):

### a. Introduction :

« .NET » est un standard proposé par la société Microsoft, pour le développement d'applications d'entreprises multi-niveaux, basées sur des composants. Microsoft « .NET » constitue ainsi la réponse de Microsoft à la plate-forme J2EE de Sun [14] qui est une machine virtuelle. « .NET » s'appuie sur la norme CLI (*Common Language Infrastructure*) qui est indépendante du langage de programmation utilisé. Ainsi tous les langages compatibles respectant la norme CLI ont accès à toutes les bibliothèques installées dans l'environnement d'exécution.

### b. Le « framework .NET »

Le « framework .NET » est un sous ensemble de la technologie Microsoft « .NET » [14]. Il a pour but de faciliter la tâche des développeurs en proposant une approche unifiée à la conception d'applications Windows ou Web, tout en introduisant des facilités pour le développement, le déploiement et la maintenance d'applications.

Le « framework » gère tous les aspects de l'exécution d'une application dans un environnement d'exécution dit « managé» :

- Il alloue la mémoire pour le stockage des données et des instructions du programme.
- Il autorise ou refuse des droits à l'application.

- Il démarre et gère l'exécution.
- Il gère la ré-allocation de la mémoire pour les ressources qui ne sont plus utilisées.

Le « framework **.**NET **»** comprend notamment :

**(i) Un environnement d'exécution qui est composé :**

▪ d'un moteur d'exécution, appelé CLR (*Common Language Runtime*), permettant de compiler le code source de l'application en un langage intermédiaire, baptisé MSIL (*Microsoft Intermediate Language*) et agissant telle la machine virtuelle Java. Lors de la première exécution de l'application, le code MSIL est à son tour compilé à la volée en code spécifique au système grâce à un compilateur JIT (*Just In Time*).

▪ d'un environnement d'exécution d'applications et de services web, appelé ASP .NET .

▪ d'un environnement d'exécution d'applications lourdes, appelé « WinForms ».

**(ii) Des services**

Ils sont sous la forme d'un ensemble hiérarchisé de classes appelé FCL (*Framework Class Library*). La FCL est une librairie orientée objet, fournissant des fonctionnalités pour les principaux besoins actuels des développeurs. Le SDK (*Software Development Kit*) fournit une implémentation de ces classes.

La figure 2.3 montre les relations du CLR et de la bibliothèque de classes avec les applications et avec l'ensemble du système :

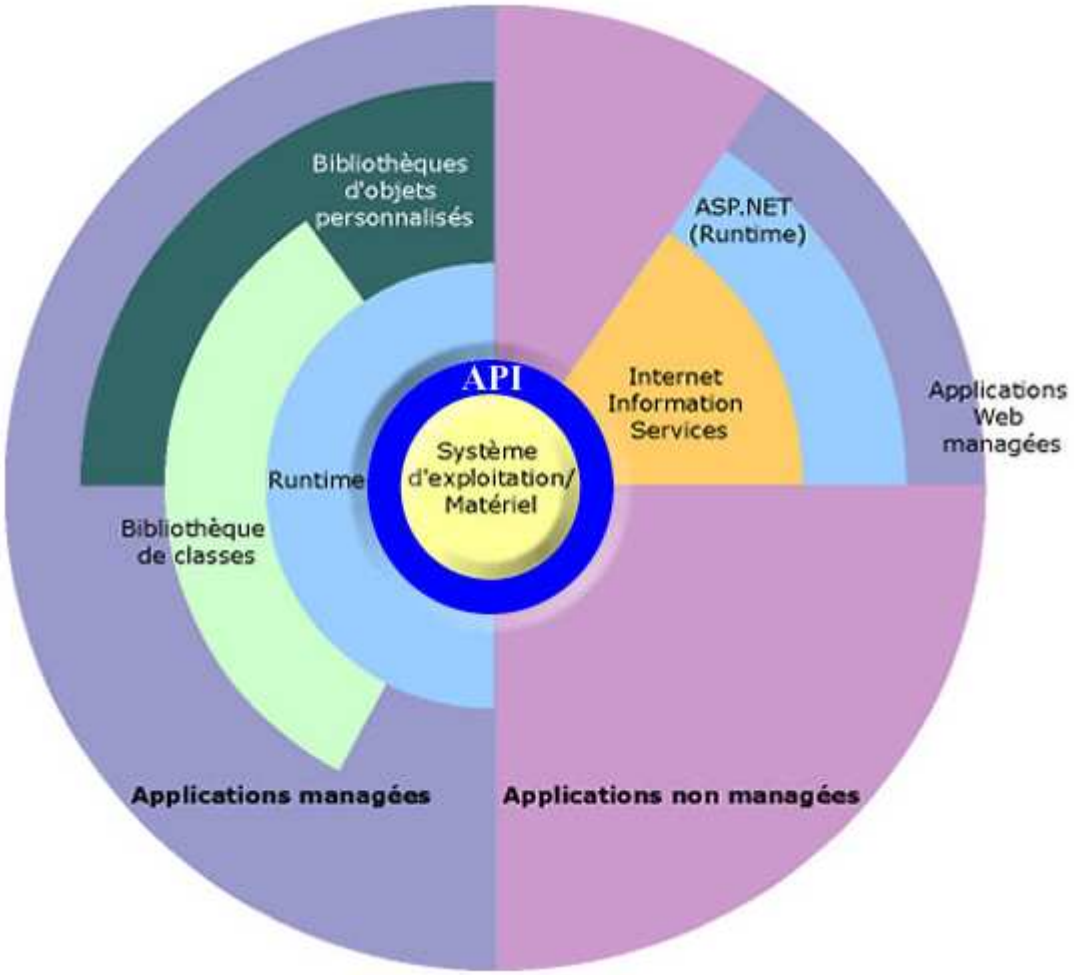


***Figure 1.3:*** Le « framework » .NET

## I.4. Format des fichiers exécutables :

Sous Windows, le terme fichier exécutable désigne un fichier contenant du code exécutable. Il en existe 3 types :

- Les applications
- Les bibliothèques
- Les pilotes

Ces fichiers sont organisés selon un format unique appelé PE (*Portable Executable*).

### a. Les applications

Les applications sont les programmes qui peuvent être lancés et arrêtés par un utilisateur. Elles portent généralement l'extension « .exe ». D'autres applications portent

toutefois une extension différente. Par exemple, les écrans de veille qui portent l'extension « .scr ».

### b. Les bibliothèques

Les bibliothèques, ou plus précisément bibliothèques dynamiques *(Dynamic-Link Libraries*), sont des fichiers qui peuvent contenir des fonctions, des variables globales ou des ressources (données ou fichiers) [9]. Elles servent à partager du code ou des données entre plusieurs applications. Les bibliothèques dynamiques portent généralement l'extension « .dll » mais certaines bibliothèques spéciales portent parfois une extension différente [8]. Par exemple, les extensions du panneau de configuration qui portent l'extension « .cpl ».

Un aperçu de l'utilisation d'un DLL est donné sur la fig.1.4 :

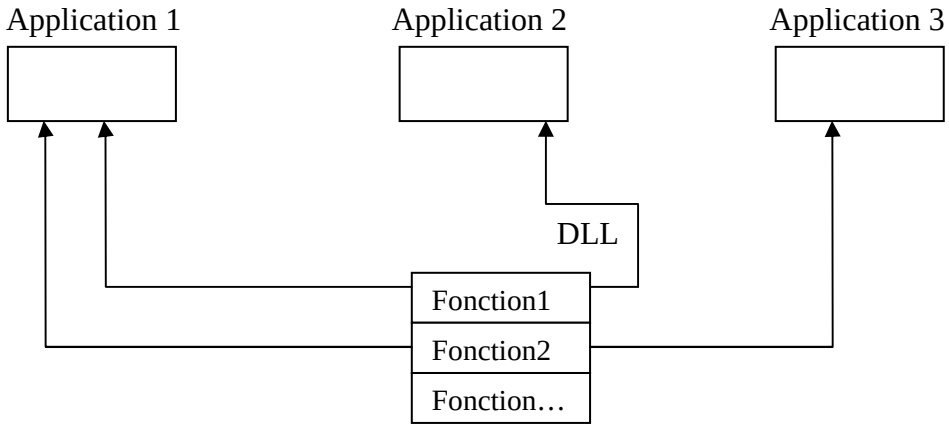


***Figure 1.4:*** Utilisation d'un DLL

### c. Les pilotes

Les pilotes sont des programmes qui gèrent l'interface entre un périphérique et le système. En effet, chaque matériel peut avoir ses spécificités donc chaque constructeur de matériel doit développer une API se conformant aux spécifications du système afin qu'il puisse être exploité par ce dernier. Les pilotes portent l'extension « .drv », « .vxd » ou « .sys ».

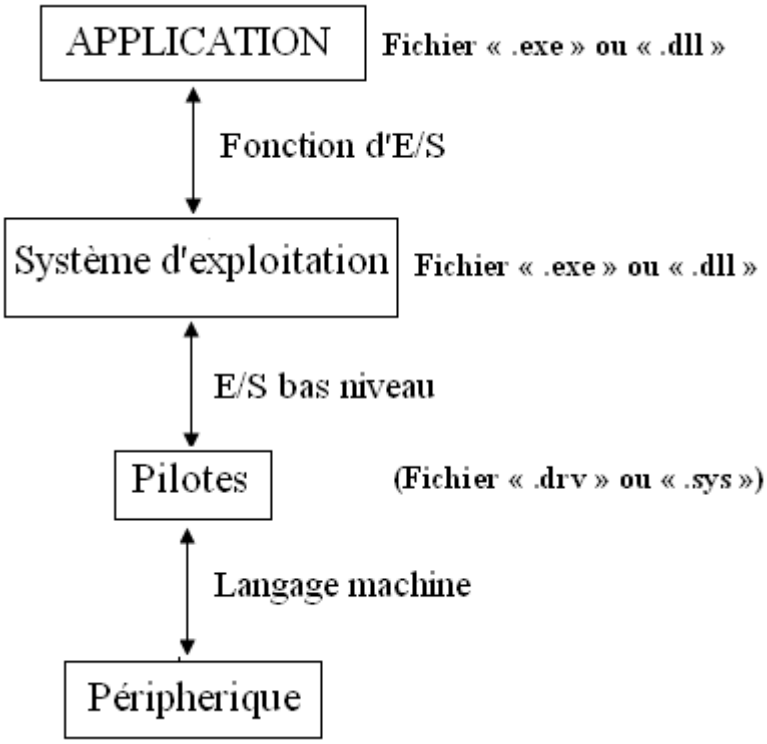


***Figure 1.5:*** Interconnexion entre application et périphérique

## I.5. Les caractéristiques du système d'exploitation Windows

Windows est un système d'exploitation graphique, multitâche, multi-threadé et multiutilisateur, fonctionnant en mode protégé [5].

### a. Graphique

Windows est un système à « Shell » graphique, c'est-à-dire que les échanges entre l'utilisateur et le système se font principalement à travers une interface graphique. Parmi les composantes responsables du graphisme sous Windows, l'une des plus importantes est celle qu'on connaît sous le nom de GDI (*Graphics Device Interface*) [8]. Il s'agit d'une API permettant de dessiner sur n'importe quel périphérique graphique (écran, imprimante, etc.) de manière standard et largement utilisée par le système lui-même pour rendre du graphisme.

### b. Multi-tâche

Comme tout système d'exploitation moderne, Windows permet de faire exécuter plusieurs programmes en même temps. Les programmes sont ordonnancés à l'aide d'une file

à priorité. Un processus à priorité élevé aura donc plus de temps processeur qu'un processus à priorité moins élevé [5].

**c. Multi-threadé**

Chaque processus peut exécuter plusieurs tâches en même temps, ces tâches, appelées également « threads », sont elles aussi ordonnancées à l'aide d'une file à priorité appartenant au processus

**d. Multiutilisateur**

Windows permet à plusieurs utilisateurs de travailler simultanément sur une même machine. Un utilisateur est caractérisé par ses identifiants (login, mot de passe, etc.), ses droits et ses préférences (dossier personnel, thème, etc.).

**e. Le mode protégé**

Le microprocesseur d'un micro-ordinateur possède trois modes de fonctionnement :

- **Le mode réel** est le mode par défaut du microprocesseur, et il a les mêmes fonctionnalités que le 8086. Cela a plusieurs conséquences dont l'une est qu'il n'est possible d'adresser plus d'1 Mo de mémoire.
- **Le mode protégé** permet d'exploiter les plus hautes performances du microprocesseur et d'adresser 4Go de mémoire vive. Il fournit un support pour l'implémentation de systèmes multitâches et multiutilisateurs.
- **Le mode virtuel** est un mode qui permet à des programmes réalisés pour le 8086 de tourner sur un système multiutilisateur.

Le mode protégé désigne un mode de fonctionnement qui a été introduit avec les processeurs 80286 du fabricant Intel. En mode protégé chaque programme se voit allouer un espace en mémoire qui lui est spécifique et auquel les autres programmes ne peuvent accéder : leur bon fonctionnement est ainsi garanti au contraire du mode réel.

Le mode protégé permet d'exploiter davantage de mémoire vive que le mode réel et donc de concevoir des programmes plus complexes, il offre aussi le support de la mémoire virtuelle et d'un environnement multitâches.

La mise en œuvre du mode protégé d'un processeur est assurée par le système d'exploitation. Le DOS fonctionne ainsi en mode réel, Windows fonctionne pour sa part en mode protégé.

Le mode protégé implémente 4 niveaux de protections. Le mode le plus privilégié est le « ring 0 » ou « *kernel mode* », c'est le mode utilisé par le noyau du système d'exploitation. Le mode qui possède la priorité la plus basse est le « ring 3 », il est utilisé par les applications utilisateurs.

## I.6. Autres fonctionnalités intégrées :

### a. Multimédia

Windows dispose de nombreuses bibliothèques permettant de développer des applications multimédia dont la plus connue est sans doute DirectX. En voici quelques unes :

#### (i) DirectX

DirectX est le nom d'une collection de bibliothèques développées par Microsoft pour développer des applications multimédia performantes et de haute qualité. DirectX permet de gérer le graphisme (2D/3D), le son et les entrées de l'utilisateur (clavier, souris, joystick, etc.). DirectX est actuellement un standard en matière de programmation multimédia et bénéficie de l'accélération matérielle de la part de nombreux constructeurs [14].

#### (ii) OpenGL

OpenGL est une interface de programmation portable pour la 2D et 3D et est largement admis comme étant le principal concurrent de DirectX. OpenGL ne gère cependant que le calcul graphique et non l'affichage, ce qui signifie que les programmes qui utilisent OpenGL doivent recourir à une bibliothèque tierce pour créer leurs fenêtres [14].
Tout comme DirectX, OpenGL bénéficie elle aussi de l'accélération matérielle de la part des constructeurs de cartes graphiques.

### (iii) Windows Multimedia

Windows Multimedia est le nom donné à l'ensemble des bibliothèques de l'API Windows permettant d'exploiter les fonctionnalités multimédia du système [14]. Elle est divisée en 4 grandes parties :

- Multimedia Audio, comprenant les fonctions de manipulation de son et des périphériques audio
- Multimedia Input, comprenant les fonctions de manipulation des périphériques d'entrée (clavier, souris, joysticks, etc.).
- VFW (*Video for Windows*), comprenant les fonctions de manipulation des images vidéo et des périphériques vidéo
- MCI (*Media Control Interface*), une API de haut niveau permettant de gérer aussi bien l'audio que la vidéo et les périphériques qui leurs sont associés.

## b. Bases de données

Windows supporte de nombreuses technologies permettant d'accéder aux données et aux fonctionnalités de n'importe quel SGBD parmi lesquelles ODBC, OLE DB et ADO.

### (i) **ODBC** (Open Database Connectivity)

ODBC est une interface de programmation permettant aux applications d'accéder aux données provenant d'un SGBD (système de gestion de bases de données). Il s'agit d'une implémentation Microsoft des spécifications X/Open et ISO/IEC utilisant le SQL comme langage d'accès aux données. ODBC offre une portabilité maximale aux applications car elle est indépendante de tout système d'exploitation et de tout SGBD [14].

### (ii) **OLE DB** (Object Linking and Embedded Database)

OLE DB est une API générique, indépendante du SGBD, elle a été conçue pour remplacer ODBC et uniformiser les accès aux bases de données [14]. En fait, OLE DB va bien au delà et permet d'accéder à toute source de données. Il peut s'agir d'un SGBD, mais également d'un fichier Excel. C'est une API bas niveau, conçue pour donner des performances optimales. En fait, c'est la couche OLE DB qui effectue les accès au SGBD.

Elle exécute des requêtes SQL sur le serveur et reçoit les données en retour. On pourrait donc utiliser directement OLE DB pour accéder au SGBD puisqu'il s'agit d'une API à part entière.

Cependant elle est très complexe à mettre en œuvre. Aussi, Microsoft a défini ADO par dessus OLEDB afin de simplifier son usage.

**(iii) ADO** (ActiveX Data Object)

ADO est une librairie d'objets COM représentant des objets de données. Il s'agit d'une API de haut niveau fournissant un niveau d'abstraction élevé pour accéder à une base de données [14]. Il est essentiellement basé sur OLE DB. Il a été bâti par-dessus OLE DB pour simplifier son utilisation et la rendre plus accessible au commun des mortels. Son principe de fonctionnement est de fournir aux développeurs un modèle objet de haut niveau masquant la complexité des accès bas niveaux. Il s'agit d'une couche d'adaptation.

### c. Réseau

Windows supporte une grande variété de protocoles réseau dont TCP/IP, IrDA (*Infrared Data Association*) et Bluetooth [14] et il fournit une kyrielle de fonctions permettant de travailler avec ces systèmes via un certain nombre d'API dont :

**(i) WinSock** (Windows Sockets API)

WinSock permet de développer des applications qui communiquent à travers un réseau quel que soit le protocole utilisé (TCP/IP, IrDA, Bluetooth, etc.) [3].

**(ii) WinInet** (Windows Internet API)

WinInet permet de développer facilement des applications clientes utilisant les protocoles standards de l'Internet comme HTTP ou FTP [14].

**(iii) IP Helper API**

IP Helper API permet d'accéder à la configuration réseau (adresse IP, masque, table de routage, etc.) de la machine locale [10] [14].

**(iv) IrDA** (Infrared Data Association API)

L'Infrared Data Association API permet de gérer les périphériques et services utilisant l'infrarouge [14].

**(v) Bluetooth API**

Cette API permet de gérer les périphériques et services Bluetooth [14].

# CHAPITRE II.
# LA BASE DE LA PROGRAMMATION SOUS WINDOWS

Windows est un système d'exploitation proposant une interface graphique. L'interface de programmation Win32 (*en anglais Win32 Application Programming Interface ou Win32API*) ou communément appelée API Windows offre aux programmeurs la possibilité d'interagir avec le système d'exploitation Windows. Ces API ne sont en fait que des fonctions qui sont contenues dans des fichiers DLL (*Dynamic-Link Library*), tels "user32.dll" ou "kernel32.dll" et elles présentent des possibilités presque infinies, et dépassent de très loin les possibilités apportées par les environnements de développement comme Visual Basic ou Windev. Par exemple, elles permettent de contrôler une application, d'accéder au registre de Windows, de jouer des sons ou de manipuler directement les périphériques.

## II.1.Création et compilation d'un programme Windows avec Micrsoft Visual C++ 2005

Microsoft Visual Studio est un ensemble complet d'outils de développement pour établir des applications Web, des applications de bureau et des applications mobiles. Visual C++, Visual Basic, Visual C# et Visual J#, utilisent toute le même environnement intégré de développement (IDE), qui leur permet de partager des outils et de facilité la création de solution commune entre ces langages.

### a. Pourquoi utiliser Microsoft Visual C++ 2005 ?

Microsoft Visual C++ 2005 procure un environnement puissant et flexible de développement pour créer des applications Windows de base. Il peut être utilisé comme un outil de développement intégré pour Windows ou simplement pour créer des applications personnelles.

Et de plus, C++ est le langage le plus utilisé pour la programmation au niveau des systèmes d'exploitation et de ce fait plus proche du système elle-même.

### b. Avantage du compilateur Visual C++ 2005

Le compilateur Visual C++ 2005 est un excellent éditeur de code, avec le code complétion qui offre des suggestions intéressantes au niveau des fonctions et de constantes et

qui corrige automatiquement les erreurs de frappes, rendant ainsi l'écriture des codes plus rapides et plus aisés. Il est aussi un excellent outil de débogage.

### c. Création d'un projet avec Visual C++ 2005

1. Lancez Visual C++ 2005.
2. Allez dans le menu **File** > **New** puis cliquez sur **Project**


***Figure 2.1:*** Menu « **File** »

3. Une fenêtre apparaît, cliquez sur **Win32** puis sur **Win32 Project** et après entrez le nom du projet par exemple « test » et cliquez sur le bouton **OK**

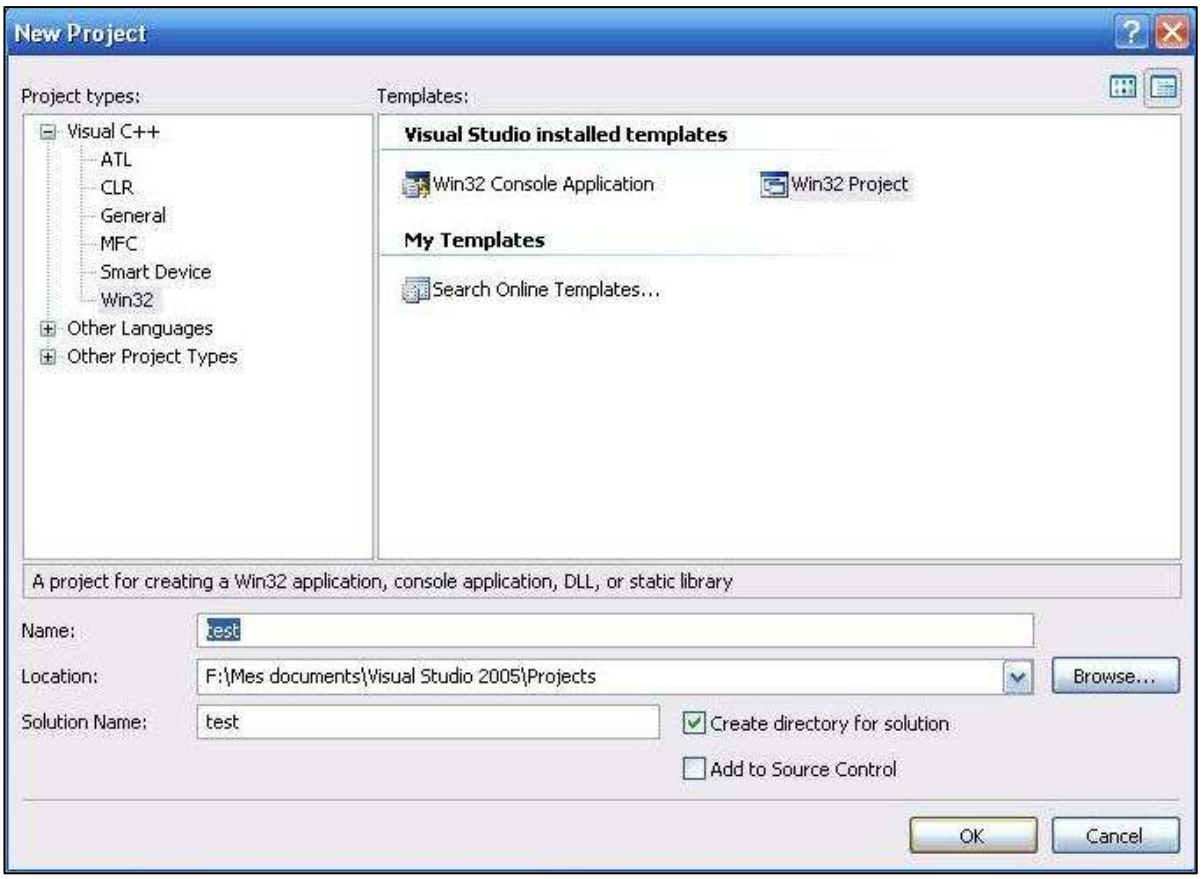


***Figure 2.2:*** Projet Win32

4. Une autre fenêtre apparaît, cliquez sur **Application Settings,** pour le type d'application choisissez « **Windows Application** » puis cochez la case « **Empty project** » et cliquez sur le bouton **Finish.**

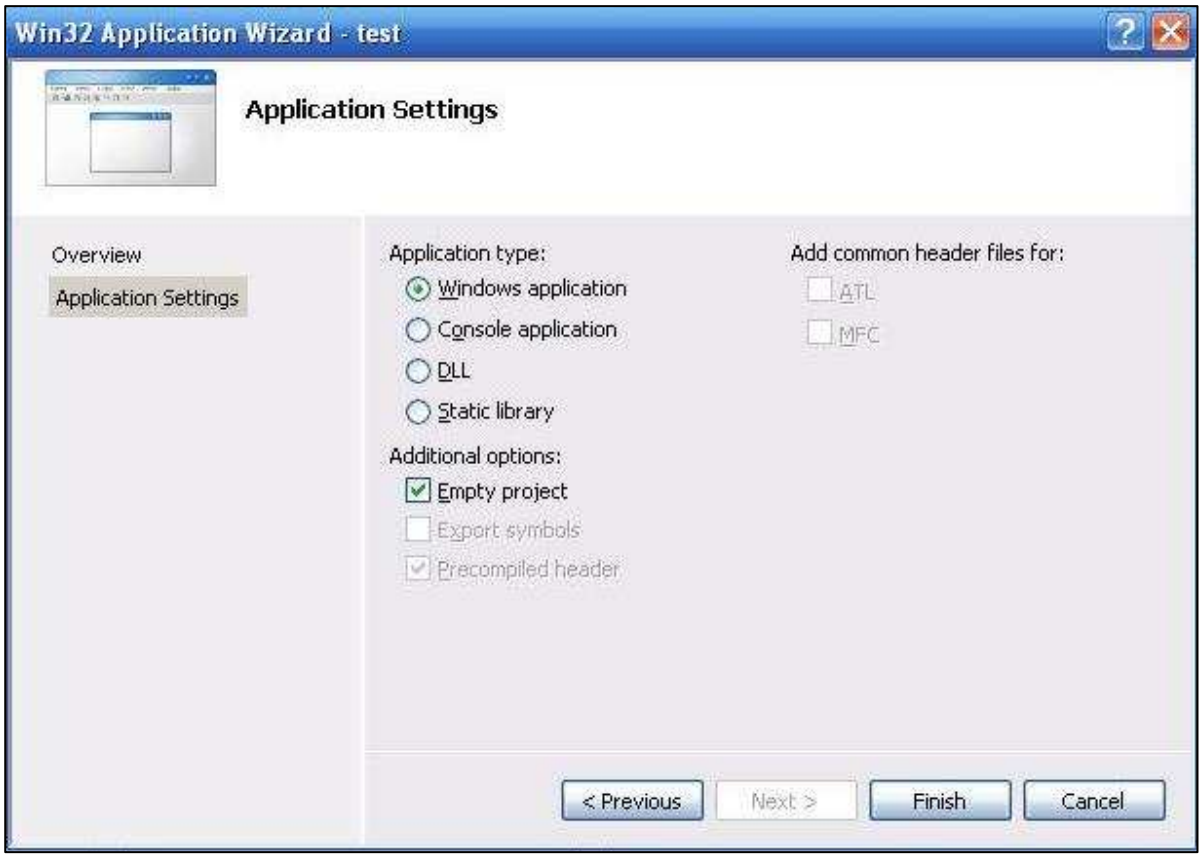


***Figure 2.3:*** Création d'une application Windows

5. Maintenant que le nouveau projet a été crée, il faut ajouter un nouveau fichier source à ce projet. Dans l'explorateur de projet Clique droite sur le filtre **Source File > Add** puis cliquez sur **New Item.**


***Figure 2.4:*** Ajout d'un fichier source

6. Une fenêtre apparaît, cliquez sur **Code** puis sur **C++ File(.cpp)** et ensuite entrez le nom du fichier, par exemple « main.cpp » et cliquez sur le bouton **Add**.

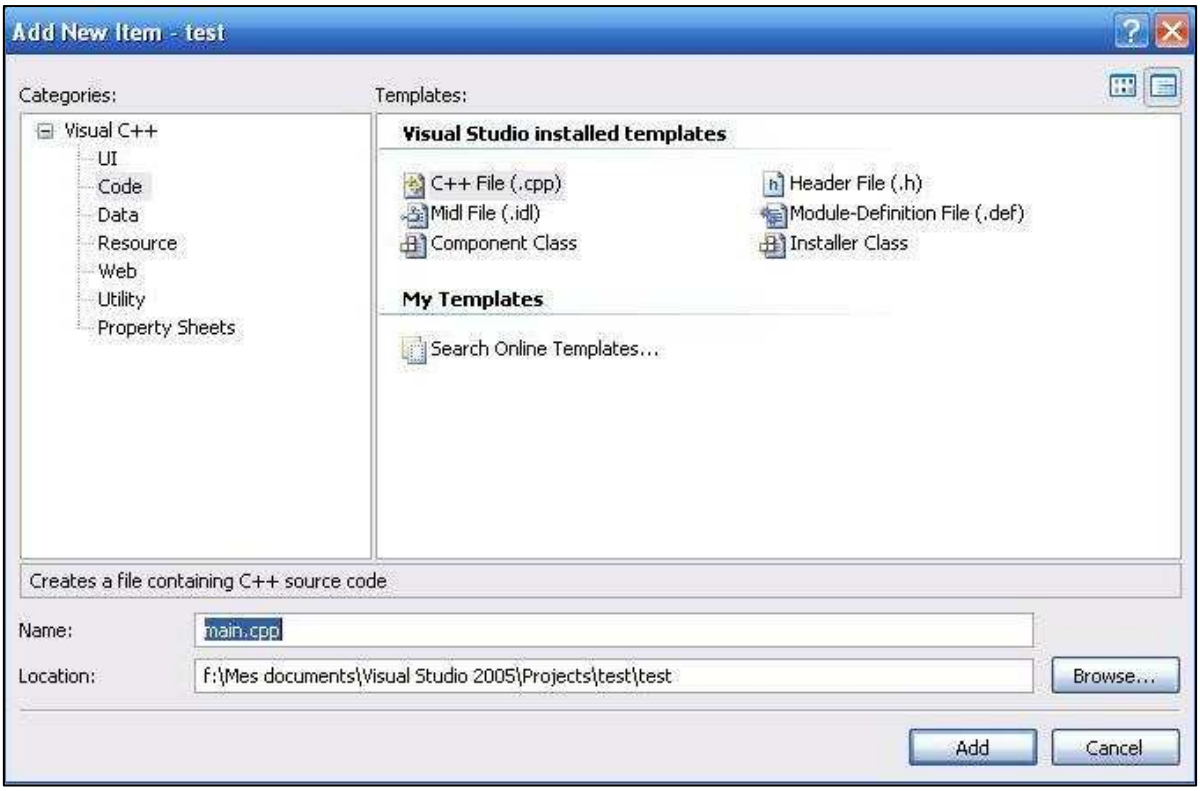


***Figure 2.5:*** Le fichier « **main.c** »

7. Enfin, il faut inclure le fichier d'en-tête « windows.h » au fichier source qu'on vient de créer car toutes les fonctions et les constantes qu'on va utiliser pour la programmation windows se trouvent dans ce fichier.

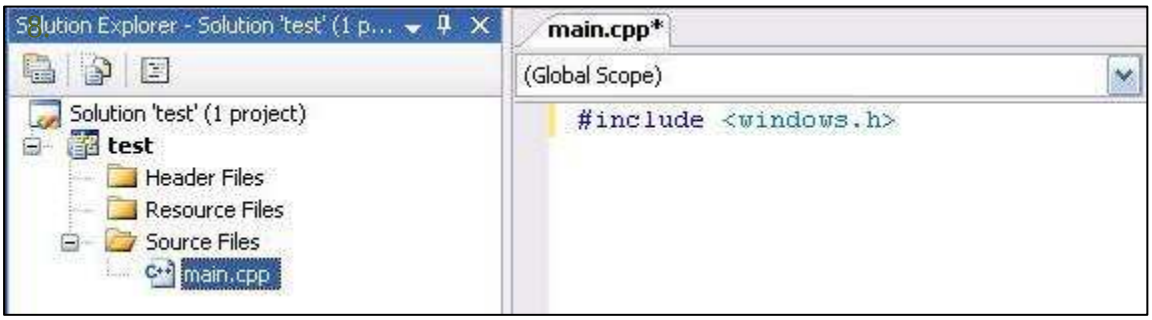


***Figure 2.6:*** L'en-tête « windows.h »

### d. Edition des liens et compilation

Certains projets utilisent des bibliothèques spécifiques comme **Winsock 2** pour la programmation des sockets par exemple. Ainsi pour qu'on puisse appeler les fonctions de cette bibliothèque, il faut que l'exécutable soit lié au fichier « ws2_32.lib » qui est un relais vers le fichier « ws2_32.dll » qui contient réellement le code des fonctions qu'on va utiliser.

Voici la marche à suivre pour lier un projet à une bibliothèque spécifique :

1. Cliquez sur le projet qu'on vient de créer dans l'explorateur de projet, puis allez dans le menu **Project** et cliquez sur **Properties (Alt + F7).**

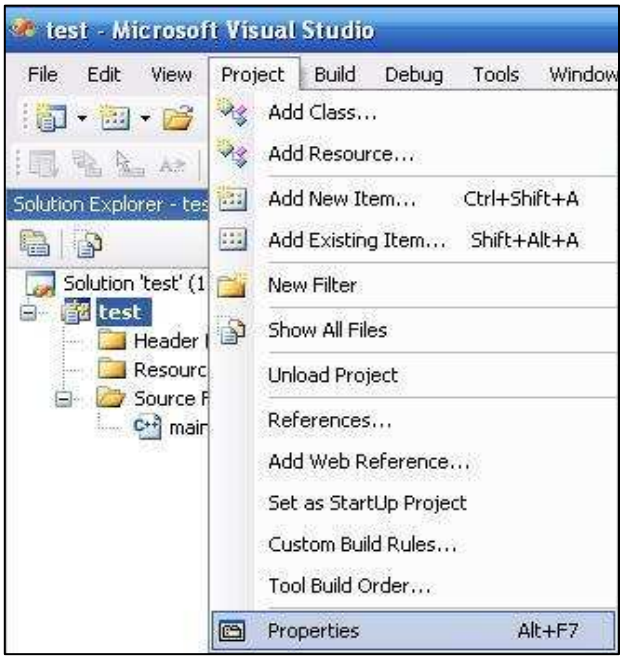


***Figure 2.7:*** Création d'un projet

2. Une nouvelle fenêtre apparaît, Cliquez sur **Configuration Properties > Linker > Input**, puis sur la droite de la fenêtre cliquez sur **Additional Dependencies** et entrez le nom

de la librairie, dans l'exemple précèdent, c'est le fichier « ws2_32.lib » et ensuite cliquez sur le bouton **OK.**

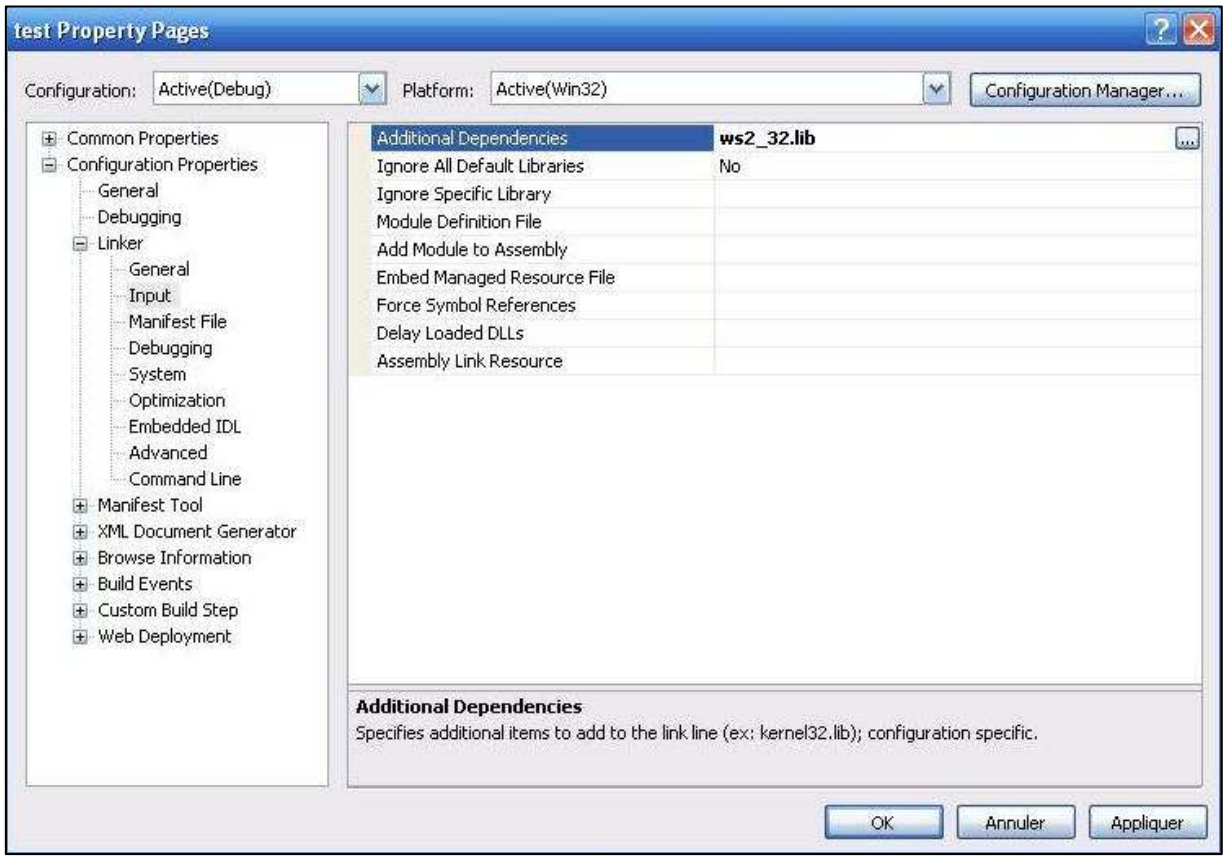


***Figure 2.8:*** Projet « ws2_32.lib »

3. Maintenant que le projet est lié à la bibliothèque, il ne reste plus qu'à inclure le fichier d'en-tête « winsock2.h » de la bibliothèque dans le fichier source. Certains fichiers d'en-tête comme « windows.h » sont déjà inclus par « winsock2.h », ainsi en incluant ce dernier, on n'a plus besoin de les inclure.

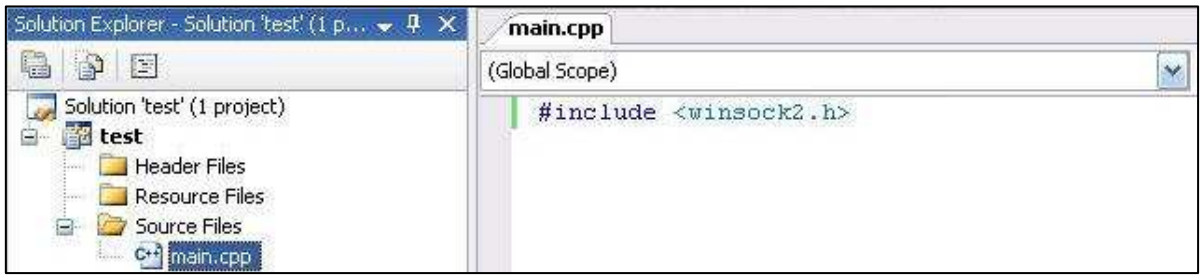


***Figure 2.9:*** L'en-tête « winsock2.h »

4. Une fois le code du programme écrit, il ne reste plus qu'à le compiler. Pour la compilation, allez dans le menu **Build**, puis cliquez sur **Build Solution(F7)**.

5. Un fichier exécutable qui porte le nom du projet est alors crée, on peut le retrouver dans le répertoire du projet, dans le dossier **Debug**.


***Figure 2.10:*** Compilation d'un projet

## II.2. La notation hongroise

La notation hongroise est la pratique de préfixer chaque identificateur pour coder des informations qui sont bons à savoir à son sujet. Cette pratique est très utilisée par les programmeurs Windows car elle permet de repérer et reconnaître facilement le type d'une variable. Par exemple le préfixe « p » ou « lp » sert à indiquer qu'il s'agit d'un pointeur.

Le tableau 2.1 montre les préfixes les plus utilisés :

**Tableau 2.1** : Préfixes

| Préfixe | Type de la variable |
|---|---|
| **h** | handle |
| **p, lp** | pointeur |
| **sz, lpsz** | chaîne de caractère terminée par 0 |
| **i, n** | int (entier) |
| **u** | UINT (unsigned int) |
| **ul** | ULONG (unsigned long) |
| **dw** | DWORD (unsigned long) |
| **c** | effectif (count) |
| **cb** | taille en octets (count of bytes) |
| **b** | BOOL (int) |

Exemple :

```
char lpszFileName [256];          /*au lieu de char FileName [256] ;*/
```

FileName est ici une variable contenant une chaine de caractère terminée par 0, ainsi on le reconnaît tout de suite avec le préfixe « lpsz ».

### II.3. Le jeu de caractères Unicode

« Unicode » est un jeu de caractères dérivé de « l'ASCII » utilisant 16 bits pour représenter chaque caractère. Il est utilisé en interne par Windows mais les applications peuvent également utiliser le jeu de caractères « ANSI » (8 bits) pour communiquer avec le système grâce à un jeu de conversions totalement opaques pour l'utilisateur. Par contre lorsqu'on développe des programmes qui doivent dialoguer directement avec le noyau du système (comme les pilotes de périphériques par exemple), on est obligé d'utiliser des chaînes en « Unicode » uniquement.

Le tableau 2.2 illustre la différence entre l'ASCII et l'Unicode en format binaire et hexadécimale :

**Tableau 2.2** : Code ASCII

| az | ASCII | Unicode |
|---|---|---|
| a | 01100001 (61h) | 00000000 01100001 (0061h) |
| b | 01100010 (62h) | 00000000 01100010 (0062h) |
| … | … | … |
| z | 01111010 (7Ah) | 00000000 01111010 (007Ah) |

Par défaut, Microsoft Visual C++ 2005 utilise « l'Unicode ». Pour pouvoir utiliser « l'ANSI », il faut ajouter au début du code la syntaxe suivante :

```
#undef  UNICODE
```

## II.4. Structure de base d'un programme utilisant l'API Windows

Malgré l'énorme potentiel de l'API Windows, les programmes issus directement de cette interface ont une structure assez simple.

Une application Windows est divisée en deux grandes parties :

- Le point d'entrée
- La procédure de fenêtre

### a. Le point d'entrée

Généralement une application doit avoir un point d'entrée. Pour les applications Windows le point d'entrée est la fonction WinMain().

```
int WINAPI  WinMain(HINSTANCE  hInstance,  HINSTANCE  hPrevInstance,
LPSTR  lpCmdLine, int  nCmdShow);
```

Ces quatre paramètres sont définis comme suit:

- hInstance: handle de l'instance de l'application. Un *handle* est un numéro qui identifie de manière unique un objet quelconque. Ici, *hInstance* identifie donc de manière unique l'application.

- hPrevInstance : handle de l'instance précédente, il vaut toujours NULL. Ce paramètre fut utilisé dans les premières versions de Windows et est uniquement gardé pour des raisons de compatibilité.

- lpCmdLine : pointeur des arguments de la ligne de commande de l'application. Pour obtenir la ligne de commande dans son intégralité, il faut utiliser la fonction *GetCommandLine()*. Cette fonction ne nécessite aucun argument et retourne un pointeur sur la ligne de commande toute entière.

- nCmdShow : indique comment l'utilisateur désire t-il que la fenêtre principale soit affichée : avec sa taille normale, agrandie ou réduite ? Rien n'oblige cependant le programme à considérer ce choix.

La fonction *WinMain()* retourne un entier appelé code d'erreur de l'application. Pour toutes les applications Windows de base, le point d'entrée contient trois étapes fondamentales :

- Enregistrement d'une classe de fenêtre
- Création d'une fenêtre
- Interception des messages

### *(i) Enregistrement d'une classe de fenêtre*

Chaque fenêtre doit avoir une classe de fenêtre. Une classe de fenêtre définit les attributs d'une fenêtre, tels que le modèle de son icône, de son curseur, du nom du menu et du nom de la procédure de fenêtre.

1. Créer une structure WNDCLASS que l'on nommera wc.

```
WNDCLASS  wc;
```

2. Compléter la structure wc

```
wc . cbClsExtra          =  0;
wc . cbWndExtra          =  0;
wc . hbrBackground       =  (HBRUSH) (COLOR_WINDOW + 1);
wc . hCursor             =  LoadCursor (NULL, IDC_ARROW);
wc . hIcon               =  LoadIcon(NULL, IDI_APPLICATION);
wc . hInstance           =  hInstance ;
wc . lpfnWndProc         =  <Adresse d'une procédure de fenêtre>;
wc . lpszClassName       =  "Nom de la Classe" ;
wc . lpszMenuName        =  NULL ;
wc . style               =  CS_HREDRAW | CS_VREDRAW;
```

3. Enregistrer la classe de fenêtre wc avec la fonction *RegisterClass()*.

```
RegisterClass(&wc);
```

### *(ii) Création d'une fenêtre*

Après avoir enregistré une classe de fenêtre, on peut désormais créer une fenêtre. La fenêtre créée sera la fenêtre principale.

1. Créer un objet HWND que l'on nommera hWnd.

```
HWND  hWnd;
```

2. Appelé la fonction *CreateWindows()* pour créer la fenêtre.

```
hWnd = CreateWindow("Nom de la Classe",     /* Classe de la fenêtre */
          "Nom de la fenêtre",              /* Titre de la fenêtre */
          WS_OVERLAPPEDWINDOW,              /* Style de la fenêtre */
          100 ,                             /* Abscisse du coin supérieur gauche */
          100 ,                             /* Ordonnée du coin supérieur gauche */
          600 ,                             /* Largeur de la fenêtre */
          300 ,                             /* Hauteur de la fenêtre */
          NULL ,                            /* Fenêtre parent */
          NULL ,                            /* Pas de menu */
          hInstance ,                       /* Instance de l'application*/
          NULL ) ;                          /* Paramètres additionnels */
```

3. Après sa création, il faut afficher la fenêtre avec la fonction *ShowWindow()*.

```
ShowWindow(hWnd, <ShowCmd>);
```

Où <ShowCmd> est un entier qui indique la disposition d'affichage de la fenêtre qu'on va souhaiter. On peut utiliser par exemple les constantes SW_SHOW, SW_HIDE, SW_MINIMIZE, SZ_MAXIMIZE. Généralement, on utilise nCmdShow de la fonction *WinMain()* afin que la fenêtre s'affiche comme l'utilisateur le désir.

Le paramètre « style » de la fonction *CreateWindows()* est la plus importante, il peut être une ou plusieurs combinaisons des constantes suivantes:

**Tableau 2.3** : Les constantes

| Constante | Description |
|---|---|
| **WS_POPUP** | Fenêtre pop-up (fenêtre "nue") |
| **WS_BORDER** | Fenêtre comportant une bordure |
| **WS_CAPTION** | Fenêtre avec barre de titre |
| **WS_MINIMIZEBOX** | Fenêtre avec un bouton Réduire |
| **WS_MAXIMIZEBOX** | Fenêtre avec un bouton Agrandir |
| **WS_SYSMENU** | Fenêtre avec menu système (+ bouton Fermer) |
| **WS_SIZEBOX (ou WS_THICKFRAME)** | Fenêtre redimensionnable |
| **WS_OVERLAPPED (ou WS_TILED)** | Fenêtre recouvrable |
| **WS_OVERLAPPEDWINDOW** | Combine tous les styles ci-dessus. |
| **WS_CHILD** | Fenêtre enfant (fenêtre dans une fenêtre) |
| **WS_VISIBLE** | Fenêtre initialement visible |

### *(iii) Interception des messages*

Une fois que la fenêtre principale est créée et affichée, la fonction *WinMain()* peut commencer sa tâche primaire, qui est de lire les messages de la file d'attente de l'application et de les expédier à la fenêtre appropriée.

Le système n'envoie pas directement le message à l'application. Mais tout d'abord, il place les entrées souris et clavier de l'utilisateur dans une file d'attente de message, avec des messages signalés par le système et par d'autres applications. L'application doit lire la file d'attente de message, rechercher les messages, et les expédier de sorte que la procédure de fenêtre puisse les traiter.

L'application doit employer la boucle de message suivante pour l'interception:

```
MSG msg;

while (GetMessage(&msg,  NULL,  0,  0) )
{
   TranslateMessage(&msg);
   DispatchMessage(&msg);
}
```

La fonction *GetMessage()* retourne TRUE tant qu'elle n'a pas reçue le message WM_QUIT. Une application doit donc envoyer ce message pour quitter la boucle. Une fois qu'on a quitté la boucle, on termine le programme.

### b. La procédure de fenêtre

#### *(i) Rôle de la procédure de fenêtre*

Chaque fenêtre doit avoir une procédure. Le vrai travail se trouve dans la procédure de fenêtre. En effet, c'est la procédure de fenêtre qui va déterminer que le programme va afficher sur la fenêtre cliente et c'est elle aussi qui va gérer tous les évènements se produisant dans l'application comme les entrées souris et clavier de l'utilisateur.

Une procédure de fenêtre est toujours définie comme suit:

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);
```

Ces quatre paramètres sont identiques aux quatre premiers champs de la structure de MSG:

- hWnd est le handle de la fenêtre recevant le message. Si le programme contient des fenêtres multiples basées sur la même classe de fenêtre, alors hWnd identifie la fenêtre particulière recevant le message.
- Le deuxième paramètre est un nombre qui identifie le message.
- Les deux derniers paramètres wParam et lParam sont des paramètres de 32 bits qui fournissent toutes les informations au sujet du message. Ils sont différents pour chaque type de message.

Généralement le programme n'appelle pas la procédure de fenêtre directement. La procédure de fenêtre est presque toujours appelée par la fenêtre lui-même.

#### *(ii) Traitement des messages*

La procédure de fenêtre reçoit les messages envoyés par le système. Chaque message reçu est identifié par un nombre, c'est le paramètre **message** de la procédure de fenêtre. Tous les messages commencent par le préfixe WM *(Window Message)*.

D'une manière générale, on utilise le control de choix multiple *switch* et *case* pour déterminer quel message la procédure de fenêtre reçoit et comment le traiter en conséquence ? Tous les messages, qu'une procédure de fenêtre choisit de ne pas traiter, doivent être passés vers une fonction appelée *DefWindowProc()* et ainsi c'est le système qui va s'en charger.

Une procédure de fenêtre est structurée comme suit :

```
switch( message )
{
        case WM_CREATE:
            …
            break;

        case WM_PAINT:
            ...
            break;

        case WM_DESTROY:
            PostQuitMessage(0);
          break;

        default:
            return ( DefWindowProc( hWnd, message, wParam, lParam) );
}
```

Etant donné qu'il existe à peu près une centaine types de messages, on ne peut pas tous les énumérer mais on va seulement en citer quelque unes :

- WM_CREATE : Ce message est reçu lorsqu'une fenêtre a été créée avec la fonction *CreateWindow()* ou *CreateWindowEx().*
- WM_PAINT : Ce message indique qu'on doit redessiner une partie ou tout l'ensemble de la fenêtre de l'application.
- WM_DESTROY: Ce message est envoyé par Windows lorsque la fenêtre est sur le point d'être détruite (après que l'utilisateur à cliquer sur le bouton fermer par exemple). A ce moment, on doit alors poster le message WM_QUIT avec la fonction *PostQuitMessage().* Le 0 passé en paramètre de cette fonction indique une fin normale.

# CHAPITRE III.
# LA PROGRAMMATION RESEAU AVEC WINSOCK

La création d'applications communicantes à travers un réseau en utilisant l'API Windows est assez aisée. Toutefois, les applications sur un réseau qui tournent sous Windows peuvent également communiquer avec des applications tournant sous Mac OS X ou sous UNIX et peu importe le langage dans lequel elles ont été écrites, la seule chose qui compte c'est qu'elles doivent utiliser le même protocole. En effet le protocole utilisé au niveau de la couche transport est le plus important car c'est ce protocole qui définit le mode de communication utilisé par l'application. On distingue généralement deux modes de communications : les communications en mode **connecté** avec le protocole TCP et les communications en mode **non connecté** avec le protocole UDP.

Pour plus d'information de ces deux protocoles, voir l'annexe 1.

La programmation réseau sous Windows est basée sur la bibliothèque Winsock (*Windows Sockets API*), qui est actuellement à sa version 2 et est connue sous le nom de **Winsock 2**. Pour utiliser **Winsock 2**, il faut lier le projet avec la bibliothèque «**ws2_32.lib»** et inclure le fichier d'en-tête «**winsock2.h»** et initialiser la DLL «**ws2_32.dll»** à l'intérieur du programme.

## III.1. Création d'une application Winsock de base :

Voici donc la marche à suivre:

**1.** Créer un nouveau projet «**Win32 Project**» vide.

**2.** Ajouter un nouveau fichier source « **C++** » vide au projet.

**3**. Lier le projet à la bibliothèque « **ws2_32.lib** » de Winsock.

**4**. Commencer le programme en incluant le fichier d'en-tête « **winsock2.h** »

```
#include "winsock2.h"

int main()
{
	return 0;
}
```

### a. Définition

Winsock est une interface de programmation réseau (*API*) et non un protocole, elle permet de créer des applications intranet ou internet avancées et d'autres applications réseau capables de transmettre des données.

Winsock 2 contrairement à la version 1.1, a la fierté de posséder de nombreuses fonctions capables de manipuler plusieurs protocoles, elle supporte ainsi une plus grande variété de protocoles. La manipulation de ces protocoles se fait via les sockets.

### b. Initialisation de Winsock

Toutes les applications utilisant Winsock doivent être initialisées pour s'assurer que les sockets sont soutenus par le système. Pour cette initialisation on procède comme suit:

**1.** Créer un objet **WSADATA** que l'on nommera **wsaData**.

```
WSADATA wsaData;
```

**2.** Appeler la fonction *WSAStartup ()* pour lancer l'utilisation de la DLL « **ws2_32.dll** ».

**3.** Terminer l'utilisation de la DLL « **ws2_32.dll** » avec la fonction *WSACleanup()* qui ne nécessite aucun argument.

```
if  (WSAStartup( MAKEWORD(2, 0),  &wsaData) != 0)
   /* La fonction WSAStartup a échoué */
else
{
   /* ... On peut continuer ... */
   WSACleanup() ;       // appel de WSACleanup à la fin du programme pour libérer la DLL
}
```

**NOTE :**

- **WSADATA** est une structure contenant les informations sur l'exécution de Winsock.
- La fonction *WSAStartup()* fait une demande de la version de Winsock sur le système.
- La macro **MAKEWORD (2,0)** crée une valeur **WORD** en enchaînant les valeurs indiquées et spécifie que l'on va utiliser la version 2 de Winsock.
- La fonction *WSAStartup()* retourne **0** si l'appel a réussi sinon elle retourne une valeur quelconque.

## III.2. Les sockets

### a. Définition

Un socket est une API permettant la communication entre deux processus distants (client et serveur en général). Ce socket est assigné à un numéro de port et à une adresse IP, donc quand on veut se connecter au port d'un serveur distant, on doit se connecter au socket distant correspondant au port voulu. Ces ports sont aux nombres de 65535, les 1024 premiers représentent les ports les plus connus, correspondants à des services tels que FTP (21), Telnet (23), SMTP (25),… . Donc, un client désirant se connecter à un serveur, va se connecter à un de ses ports ouverts en utilisant un des ports de sa machine (port source).

Voir annexe 1 pour les informations des adresses IP.

### b. Création d'un socket

La création d'un socket n'est pas très différente de l'initialisation de Winsock :

**1.** Créer un objet **SOCKET** que l'on nommera **sock**.

```
SOCKET sock;
```

**2.** Appeler la fonction *socket()* qui retourne sa valeur à la variable **sock**:

```
SOCKET socket(int pf, int type, int protocol);
```

**3.** Libérer le socket **sock** avec la fonction *closesocket()* dont le seul argument est le socket crée précédemment.

**NOTE :**

- L'argument **pf** (protocol family) définit le domaine d'adressage du socket, qui va sélectionner la famille de protocoles à utiliser, ces domaines peuvent être :
  - AF_INET ou PF_INET: Ce domaine d'adressage est le plus courant.
  - AF_UNIX (*Unix internal protocols*): Ce domaine est utilisé essentiellement pour la communication inter-processus.
  - AF_ISO (*Iso protocols)*
  - AF_NS (*Xeros Network System protocols*)
- L'argument **type** spécifie le type de socket. Le tableau 3.1 montre les valeurs possibles pour l'argument **type:**

**Tableau 3.1** : L'argument types

| Type | Description |
|---|---|
| **SOCK_STREAM** | Pour la famille d'adresse TCP. |
| **SOCK_DGRAM** | Pour la famille d'adresse UDP. |
| **SOCK_RAW** | Les raw sockets permettent à une application d'envoyer et recevoir des paquets avec les en-têtes crées par l'utilisateur lui-même. |

- L'argument **protocol** spécifie le protocole à utiliser avec le socket. La liste suivante montre les valeurs possibles :

  - IPPROTO_TCP : protocole TCP
  - IPPROTO_UDP : protocole UDP
  - IPPROTO_RAW : pour un protocole personnalisé.
  - 0 : la fonction va choisir elle même le protocole qui va le mieux avec le socket vu les valeurs qu'on a spécifié dans les deux premiers arguments.

- La fonction ***socket()*** retourne **INVALID_SOCKET** si l'appel a échoué sinon elle retourne un numéro identifiant le socket crée.

### III.3. Les sockets en mode connecté ou SOCK_STREAM

La communication en mode connecté est comparable à la communication téléphonique c'est-à-dire qu'un programme demande à un autre d'établir une connexion et si ce dernier accepte, la connexion est effectivement établie et la communication entre les deux programmes peut se faire tant que la connexion reste active. Pour établir une connexion avec une machine sur le réseau, il faut spécifier son adresse IP puis le numéro du port sur lequel on veut se connecter.

#### a. Réalisation d'un serveur

Il y a cinq étapes à suivre pour créer un serveur :

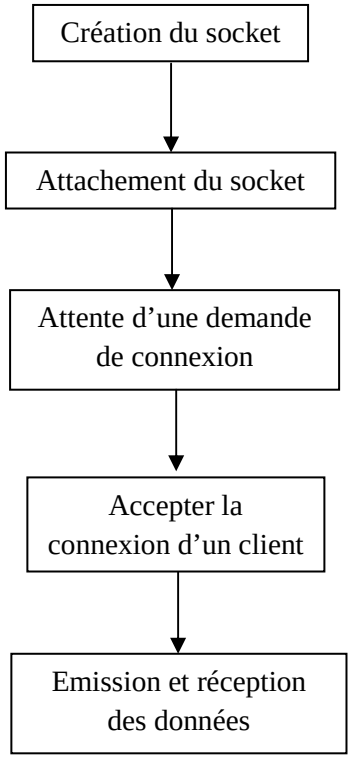


***Figure 3.1:*** Réalisation d'un serveur

**Etape I** : ***Création du socket***

En mode connecté, le protocole employé est le protocole TCP. De ce fait, le socket crée doit supporter la famille de protocole PF_INET et il doit être de type SOCK_STREAM.

```
SOCKET  sock_serveur;

sock_serveur  =  socket(PF_INET,  SOCK_STREAM,  IPPROTO_TCP);

if (sock_serveur  = =  INVALID_SOCKET)
   /* La fonction socket a échoué */ ;
else
{
   /* ... On continue ... */
   closesocket(sock_serveur);   // libération du socket sock a la fin du programme
}
```

**NOTE :**

- La fonction ***socket()*** retourne **INVALID_SOCKET** si l'appel a échoué sinon elle retourne un numéro identifiant le socket créé.

**Etapes II:** ***Attachement du socket***

Pour qu'un serveur accepte la connexion d'un client, il doit être lié à une adresse IP dans le réseau. Ainsi après sa création on doit relier le socket à une adresse IP et à un numéro

de port, afin que chaque client puisse se connecter au serveur par l'intermédiaire de cette adresse IP et de ce port.

La structure SOCKADDR contient les informations concernant l'adresse IP et le numéro du port. La structure SOCKADDR_IN est un sous-ensemble de SOCKADDR et est employé pour les applications utilisant l'IPV4.

**1**. Créer une structure SOCKADDR_IN que l'on nommera serveur.

```
SOCKADDR_IN   serveur;
```

**2.** Remplir cette structure.

```
serveur . sin_family                = PF_INET;
serveur . sin_addr.s_addr           =  inet_addr("127.0.0.1");
serveur . sin_port                  =  htons(5050);
```

**3.** Appeler la fonction *bind(),* en passant le socket sock_serveur et la structure SOCKADDR_IN serveur comme paramètres afin de les reliés ensembles.

```
if  ( bind(sock_serveur,  (SOCKADDR *)&serveur,  sizeof(serveur) ) = = SOCKET_ERROR)
   /* La fonction bind a échoué */
else
{
   /* Connexion reussie! */
}
```

NOTE :

- Le paramètre client.sin_family spécifie la famille d'adresse à utiliser ; ici c'est la famille d'adresse PF_INET.

- Les paramètres client.sin_addr.s_addr et client.sin_port spécifient respectivement l'adresse IP et le numéro du port auquel le socket sock_serveur sera lié.

- La fonction *inet_addr()* convertie une chaine de caractère contenant une adresse IP en un entier représentant cette adresse.

- La fonction *htons()* permet de réarranger l'ordre des octets de l'entier passé en argument de façon à ce qu'il corresponde au format requis sur le réseau.

- L'opérateur *sizeof()* donne la quantité de stockage en octets pour stocker un objet du type de l'opérande et permettant ainsi d'éviter et d'indiquer la tailles des données dans le programme.

**Etapes III:** ***Attente d'une demande de connexion d'un client***

Après que le socket soit lié à une adresse IP et à un port sur le système, le serveur doit écouter sur cette adresse IP et ce port, toutes demandes de connexion entrantes.

Pour mettre le socket en état d'écoute de toutes les demandes de connexion entrantes, il faut appeler la fonction *listen()* dont voici le prototype **:**

```
int listen(SOCKET sock_serveur, int backlog);
```

**NOTE :**

- La fonction *listen()* retourne 0 si l'appel a réussi sinon elle retourne SOCKET_ERROR.

- **backlog** est la longueur maximale de file d'attente des demandes des connexions. Si on le met à 1, uniquement un seul pourra se connecter au serveur.

**Etapes IV:** ***Accepter une demande de connexion d'un client***

Une fois que le socket est en état d'écoute, il ne reste plus qu'à patienter qu'un client envoi une demande de connexion. Ce client a besoin d'un socket pour s'identifier, donc en premier lieu, on doit lui en créer un.

**1**. Créer temporairement un objet SOCKET pour le client que l'on nommera **sock_client**.

```
SOCKET sock_client;
```

**2.** Créer une boucle continue pour vérifier toutes les demandes de connexions entrantes. Si une demande de connexion arrive, appeler la fonction *accept()* pour traiter la demande et une fois la demande acceptée, transférer le socket provisoire **sock_client** au socket initial **sock_serveur** et enfin sortir la boucle avec la commande **break**.

```
while (1)
 {
   sock_client = SOCKET_ERROR;

   while (sock_client = = SOCKET_ERROR )
  {
     sock_client = accept( sock_serveur, NULL, NULL );
   }
   sock_serveur = sock_client;
   break;
}
```

**Etapes V:** ***Emission et réception des données***

L'émission et la réception des données sont deux étapes importantes dans une application Winsock. En effet, les fonctions utilisées sont différentes selon le protocole de transport employé. En mode connecté, on a la fonction *send()* pour l'émission et la fonction *recv()* pour la réception des données.

```
int bytesSent, bytesRecv;
char recvbuf[32] , sendbuf[32] = "Données a envoyés.";

bytesSent = send(sock, sendbuf, strlen(sendbuf), 0);

bytesRecv = recv( sock, recvbuf, 32, 0)
```

**NOTE:**

- Les entiers **bytesSent** et **bytesRecv** reçoivent respectivement le nombre d'octets envoyé et le nombre d'octets reçu.

- **sendbuf** et **recvbuf** sont des tampons contenant respectivement les données à envoyer et les données reçues.

- La fonction *strlen()* permet d'obtenir la longueur de la chaine de caractère.

- Le dernier paramètre est rarement utilisé car la plupart du temps on le met tout simplement **0** mais en fait son rôle est de spécifier la manière de réaliser l'opération en cours. Par exemple, dans le cas d'une réception de données, s'il est égal à **MSG_PEEK**, les données ne seront pas retirées du tampon de réception **recvbuf** après leur réception et dans le cas d'une émission de données, s'il est égal à **MSG_DONTROUTE** les données ne seront pas routées.

**b. Réalisation d'un Client**

La création d'un client est plus aisée parce qu'il n'y a que quatre étapes à suivre :

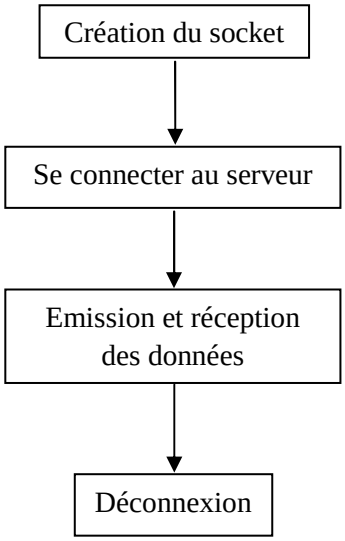


***Figure 3.2:*** Réalisation d'un client

Etant donné qu'on est toujours en mode connecté.

**Etape I** : ***Création du socket***

Cette étape est la même que pour la création du serveur.

**Etape II:** ***Se connecté au serveur***

Pour qu'un client puisse communiquer sur le réseau, il doit se connecter à un serveur. La création de cette connexion est presque la même que l'attachement du socket de coté serveur.

**1.** Créer une structure SOCKADDR_IN que l'on nommera **client**.

```
SOCKADDR_IN   client;
```

**2.** Remplir cette structure.

```
client. sin_family                = AF_INET;
client . sin_addr.s_addr          =  inet_addr("127.0.0.1");
client . sin_port                 =  htons(5050);
```

**3.** Appeler la fonction *connect(),* en passant le socket **sock_client** et la structure SOCKADDR_IN **client** comme paramètres.

```
if  ( connect(sock_client,  (SOCKADDR *)&client,  sizeof(client) ) = = SOCKET_ERROR)
   /* La fonction connect a échoué */
else
{
   /* Connexion réussie! */
}
```

**NOTE :**

- Le paramètre **client.**sin_family est défini pour mettre la famille d'adresse à utiliser

- Le paramètre **client.**sin_addr.s_addr est défini pour mettre l'adresse IP du serveur auquel le socket client **sock_client** va se connecter ; ici c'est l'ordinateur local.

- Le paramètre **client.**sin_port sert à spécifier le numéro du port du serveur auquel le socket client **sock_client** va se connecter ; ici c'est le port 5050.

**Etape III** : c'est aussi la même que l'étape III du coté serveur

**Etapes IV: *Déconnexion***

Une fois la connexion terminée, on doit fermer le socket client et libérer toutes les ressources liées à ce socket. La méthode est la même pour le mode connecté et le mode non connecté mais les conséquences sont différentes.

1. Refuser l'émission et/ou la réception des données avec la fonction *shutdown()*.

```
int shutdown(SOCKET sock_ client, int how);
```

2. Appeler la fonction *closesocket ()* pour libérer les ressources liées au socket **sock_client** et pour fermer le socket.

```
int closesocket(SOCKET sock_ client);
```

**NOTE :**

- Le paramètre **how** spécifie le type d'opération qui ne sera plus autorisé, il peut avoir trois valeurs possibles:

  - SD_RECEIVE : pour rejeter tous les appels à la fonction *recv()* sur le socket.
  - SD_SEND : pour rejeter tous les appels à la fonction *send()* sur le socket.
  - SD_BOTH : pour rejeter tous les appels aux fonctions *recv()* et *send()* sur le socket.

### III.4. Les sockets en mode non connecté ou SOCK_DGRAM

En mode non connecté, les programmes se communiquent entre eux sans établir une connexion. Ce type de communication est comparable à la communication par courrier. En effet à chaque envoi de données, il faut spécifier l'adresse du destinataire et le numéro du port sur lequel on veut que les données arrivent à la réception, il faut toujours vérifier d'où

viennent les données. Ainsi, il ne s'agit plus de client-serveur mais plutôt d'expéditeur-récepteur.

La programmation des sockets en mode non connecté utilise le protocole UDP et elle est presque similaire à celle des sockets en mode connecté. La principale différence c'est qu'on n'a plus besoin d'appeler la fonction *connect()* du coté expéditeur puisque l'adresse du destinataire est spécifiée à chaque envoi , et plus besoin aussi d'appeler les fonctions *listen()* et *accept()* du côté du récepteur puisque chaque données entrantes peuvent venir de n'importe où. Par contre il est toujours nécessaire d'attacher le socket **sock_expediteur** à un port, c'est-à-dire faire appel à la fonction *bind()* et en outre, les fonctions *send()* et *recv()* seront remplacées par les fonctions *sendto()* et *recvfrom()* pour l'émission et la réception des données.

### a. Réalisation du récepteur

La réalisation d'un récepteur à la différence d'un serveur ne comprend que trois étapes :

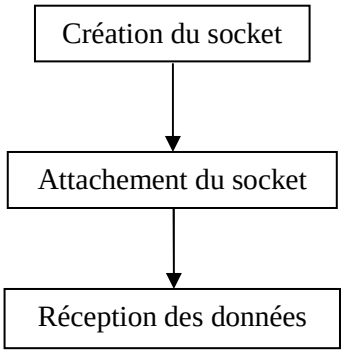


***Figure 3.3:*** Réalisation du récepteur

**Etapes I: *Création du socket***

En mode non connecté, le protocole employé est le protocole UDP. De ce fait, le socket crée doit supporter la famille de protocole PF_INET et il doit être de type SOCK_DGRAM.

```
SOCKET  sock_recepteur;

sock_recepteur  =  socket(PF_INET,  SOCK_DGRAM,  IPPROTO_UDP);

if (sock_recepteur  = =  INVALID_SOCKET)
   /* La fonction socket a échoué */ ;
else
{
   /* ... On continue ... */
   closesocket(sock_recepteur);   // libération du socket sock a la fin du programme
}
```

**NOTE :**

- La famille de protocole utilisé est la même qu'en mode connecté parce que les protocoles TCP et UDP appartiennent à la même famille de protocoles.

**Etapes II:** ***Attachement du socket***

Tout comme en mode connecté, le socket récepteur **sock_recepteur** doit être relié à un numéro de port, mais étant donné qu'un récepteur peut recevoir des données venant de n'importe quel expéditeur, il ne sera pas lié à une adresse IP spécifique.

**1**. Créer une structure SOCKADDR_IN que l'on nommera **recepteur**.

```
SOCKADDR_IN   recepteur;
```

**2.** Remplir cette structure.

```
recepteur . sin_family            = PF_INET;
recepteur . sin_addr.s_addr       =  htonl(INADDR_ANY);
recepteur . sin_port              =  htons(5050);
```

**3.** Appeler la fonction bind(), en passant le socket **sock_recepteur** et la structure SOCKADDR_IN **recepteur** comme paramètres afin de les reliés ensembles.

```
if  ( bind(sock_recepteur,  (SOCKADDR *)&recepteur,  sizeof(recepteur) ) = = SOCKET_ERROR)
   /* La fonction bind a échoué */
else
{
   /* Connexion reussie! */
}
```

**NOTE :**

- Le paramètre client.sin_family spécifie la famille d'adresse à utiliser ; ici c'est la famille d'adresse PF_INET.

- La valeur INADDR_ANY du paramètre client. sin_addr.s_addr indique que le socket n'est lié à aucune adresse IP spécifique.

- Le paramètre client.sin_port spécifie le numéro du port auquel le socket sock_recepteur sera lié ; ici c'est le port 5050.

- Les fonctions *htons()* et *htonl()* permettent de réarranger l'ordre des octets de l'entier passé en argument de façon à ce qu'il correspond au format requis sur le réseau.

- L'opérateur *sizeof()* donne la quantité de stockage en octets pour stocker un objet du type de l'opérande et permettant ainsi d'éviter et d'indiquer la tailles des données dans le programme.

**Etapes III:** ***Réception des données***

Pour la réception des données, on a besoin d'une autre structure SOCKADDR_IN pour recevoir toutes les informations concernant l'expéditeur.

**1.** Créer une structure SOCKADDR_IN que l'on nommera **info_expéditeur**.

```
SOCKADDR_IN   info_expéditeur;
```

**2.** Appeler la fonction *recvfrom()* qui retourne le nombre de bit reçu.

```
int   bytesRecv;
char   recvbuf[100];

bytesRecv = recvfrom( sock_recepteur, recvbuf, sizeof(recvbuf), 0, (SOCKADDR *) &info_expéditeur,
sizeof(info_expéditeur) );
```

**NOTE:**

- L'entier **bytesRecv** reçoit le nombre d'octets recueilli.
- **recvbuf** est le tampon contenant les données reçues.
- Le quatrième paramètre est rarement utilisé car la plupart du temps on le met tout simplement 0.
- Toutes les informations concernant l'expéditeur comme l'adresse IP ou le port qu'il a utilisé sont stockées dans la structure **info_expéditeur.**

### b. Réalisation d'un expéditeur

Il n'y a que deux étapes à suivre pour l'envoi de données en mode non connecté :

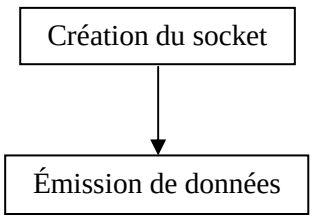


**Figure 2.4** : Réalisation d'un expéditeur

**Etape I** : étant donné qu'on est toujours en mode non connecté, la première étape est la même que celle du récepteur.

**Etapes II:** ***Émission de données***

Pour l'émission de données, on a aussi besoin d'une autre structure SOCKADDR_IN pour mettre toutes les informations concernant le destinataire des données afin que ce dernier puisse vérifier que les données lui sont bien destinées.

**1.** Créer une structure SOCKADDR_IN que l'on nommera **info_destinataire**.

```
SOCKADDR_IN   info_destinataire;
```

**2.** Remplir cette structure avec la famille d'adresse, l'adresse IP et le numéro de port utilisés par le destinataire.

```
info_destinataire . sin_family              =  PF_INET;
info_destinataire . sin_addr.s_addr         =  inet_addr("127.0.0.1");
info_destinataire . sin_port                =  htons(5050);
```

**3**. Appeler la fonction *sendto()* qui retourne le nombre de bit envoyé.

```
int   bytesSent;
char   sendbuf[100];

bytesSent  =  sendto ( sock_expediteur,  sendbuf,  sizeof(sendbuf),  0,  (SOCKADDR *) &info_destinataire,
                sizeof(info_destinataire) );
```

**NOTE:**

- L'entier **bytesSent** reçoit le nombre d'octets envoyé.
- **recvbuf** est le tampon contenant les données à émettre.
- Le quatrième paramètre est rarement utilisé car la plupart du temps on le met tout simplement 0.
- Toutes les informations concernant le destinataire comme l'adresse IP ou le port qu'il va utiliser pour recevoir les données, sont stockées dans la structure **info_destinataire.**

# CONCLUSION

Ce mémoire a permis de montrer une partie des possibilités de Windows dans les applications réseau et aussi d'approfondir la programmation sous Windows. Cela, afin de mettre en évidence les fonctions API plus ou moins ignorées par les développeurs et le grand public surtout dans le domaine du réseau. Lors de sa réalisation, plusieurs obstacles ont été surmontés, parmi ceux-ci :

- L'acquisition de machines pour faire les recherches (au moins deux ordinateurs de type Pentium IV équipés d'une carte réseau Ethernet chacune).
- La recherche de la documentation concernant les fonctions API utilisées pour le réseau surtout pour les sockets bruts.

Les applications réalisées dans cette étude permettent de résoudre à distance les problèmes de maintenances et d'administrations dans le domaine du réseau informatique. Cependant, elles pourront être aussi utilisées dans d'autres domaines comme le télé-enseignement par exemple.

Pourtant, Winsock n'est seulement qu'une infime partie de l'API Windows. En effet les API telles : WinInet, IrdA API, Bluetooth API, DirectX, OpenGL, … ; pourront être chacune un nouveau thème de mémoire voire même plus à l'avenir.

Ce mémoire peut aussi servir de référence pour les développeurs qui souhaitent s'aventurer dans le domaine du réseau pour leur expérience professionnelle, et peuvent tirer profit des applications mentionnées dans ce livre

Windows restera encore le système d'exploitation le plus utilisé par le grand public notamment avec la prochaine sortie de Windows seven prévue pour le mois de juillet de cette année 2009 et aussi avec Midori qui est la nouvelle version de Windows dont la sortie est annoncée pour l'année 2013.

# *ANNEXE*

## ANNEXE A1

NOTION GENERALE SUR LES RESEAUX

### A1.1 Définition

Un réseau est un système dans lequel des ordinateurs et des périphériques peuvent partager des données, des périphériques, des logiciels…C'est aussi un ensemble des ordinateurs et périphériques connectés les uns aux autres. Les réseaux sont classifiés selon la distance, la topologie et l'interconnexion.

On distingue deux types de réseau, d'une part le réseau local ou LAN qui relie les ordinateurs proches les uns des autres et d'autre part, le réseau étendu ou WAN qui couvre une grande distance géographique de l'ordre de la taille d'un pays ou d'un continent.

### A1.2 Protocoles

Un protocole est un ensemble de règles qu'il faut respecter pour émettre et recevoir des données sur un réseau. Il en existe plusieurs selon ce que l'on attend de la communication. Certains protocoles seront par exemple spécialisés dans l'échange des fichiers, d'autres pourront servir à gérer simplement l'état de transmission et des erreurs pour le cas de protocole ICMP mais les plus importants dans cette étude sont le protocole TCP, le protocole IP, le protocole UDP et le protocole ICMP.

#### 1. Le protocole TCP

Le protocole TCP met en œuvre dans la couche transport du modèle OSI, c'est un service de transport fiable. Il permet de gérer les données à destination de la couche inférieure du protocole IP.

TCP est un protocole opérant un contrôle de transmission des données pendant une communication établie entre deux machines.

##### *(i) Rôle*

- Le protocole TCP permet de remettre en ordre les datagrammes en provenance du protocole IP.
- Il permet de vérifier le flot de données afin d'éviter une saturation du réseau.
- Il permet de formater les données en segments de longueur variable afin de les remettre au protocole IP.
- Il permet de faire circuler simultanément les informations provenant de sources distinctes sur une même ligne.
- Enfin, TCP permet l'initialisation et la fin d'une communication de manière courtoise.

### *(ii) Segmentation*

L'unité de protocole de TCP est appelée un segment. Ces segments sont échangés pour établir la connexion, pour transférer des données, pour les acquittements, pour modifier la taille de la fenêtre et enfin pour fermer une connexion. Les informations de contrôle de flux peuvent être transportées dans le flot de données inverses. Chaque segment est composé de deux parties : l'en-tête suivi des données.

Le tableau 1 montre le format d'un segment TCP

***Tableau A1.1:*** Format d'un segment TCP

<table>
<tr><td colspan="4">Port Source<br>(16 bits)</td><td colspan="4">Port destination<br>(16 bits)</td></tr>
<tr><td colspan="8">Numéro de séquence<br>(32 bits)</td></tr>
<tr><td colspan="8">Numéro d'accusé de réception<br>(32 bits)</td></tr>
<tr><td>Décalage donnée<br>(4 bits)</td><td>Réservé<br>(6 bits)</td><td colspan="2">Fenêtre<br>(16 bits)</td><td>URG</td><td>ACK</td><td>PSH</td><td>RST</td><td>SYN</td><td>FIN</td></tr>
<tr><td colspan="4">Somme de contrôle de l'en-tête (checksum)<br>(16 bits)</td><td colspan="4">Pointeur d'urgence<br>(16 bits)</td></tr>
<tr><td colspan="4">Options</td><td colspan="4">Remplissage</td></tr>
<tr><td colspan="8">Données</td></tr>
</table>

Voici la signification de ce format du segment :

- *Port source*: ce champ contient l'adresse du port d'entrée. Associée avec l'adresse IP, cette valeur donne un identificateur unique appelé socket.

- *Port destination*: même chose que le précédent mais pour l'adresse destination.

- *Numéro d'accusé de réception*: ce champ indique le numéro du premier octet porté par le segment.

- *Décalage des données*: c'est la valeur de la longueur de l'en-tête par un multiple de 32 bits. En effet, si cette valeur est 8, la longueur totale de l'en-tête est de 8 x 32 bits. Cette valeur est nécessaire parce que la zone d'option est de longueur variable.

- *Réservé*: c'est la zone réservée pour une utilisation ultérieure. Ce champ doit être rempli de 0.

- *Drapeaux (flags) (6x1 bit):* les drapeaux représentent des informations supplémentaires:

  - **URG (*Urgent*):** si ce flag est à 1 le paquet doit être traité de façon urgente.

- **ACK (Acknowledgment):** si ce drapeau est à 1 le paquet est un accusé de réception.
- **PSH (*Push*):** si ce drapeau est à 1, le paquet fonctionne suivant la méthode PUSH.
- **RST (*Reset*):** si ce drapeau est à 1, la connexion est réinitialisée.
- **SYN (*Synchronize*):** Indique une demande d'établissement de connexion.
- **FIN (*Finish*):** si ce drapeau est à 1 la connexion s'interrompt.

- *Fenêtre:* champ permettant de connaître le nombre d'octets que le récepteur souhaite recevoir sans accusé de réception.
- *Somme de contrôle* : les deux octets permettent de détecter les erreurs dans l'en-tête et le corps du segment.
- *Pointeur d'urgence* : ce champ spécifie le dernier octet d'un message urgent.
- *Options* : cette zone contient les différentes options du protocole TCP. On y trouve principalement des options de routage.
- *Remplissage* : on remplit l'espace restant après les options des zéros pour avoir une longueur multiple de 32 bits.
- Le segment se termine par les données transportées.

### 2. Le protocole IP

Le protocole IP qui est mis en œuvre dans la couche réseau, est l'un des protocoles les plus importants d'Internet car il assure l'élaboration et le transport des paquets de données de bout en bout. En réalité, le protocole IP traite les paquets de données indépendamment les uns des autres en définissant leur représentation, leur routage et leur expédition.

Le protocole IP a pour rôle d'assurer le routage, de fragmenter les messages.

#### *(i) Les datagrammes*

Un datagramme c'est un paquet de données c'est-à-dire l'information à acheminer est découpée en paquet.

Le datagramme IP est l'unité de transfert de base qui est constitué d'un en-tête et d'un champ de données. Le format d'un datagramme IP est illustré dans le tableau 2 ci-dessous :

***Tableau A1.2:*** Format d'un datagramme IP

32 bits

| Version (4 bits) | Longueur d'en-tête (4 bits) | Type de service (8 bits) | Longueur totale (16 bits) | |
|---|---|---|---|---|
| Identification (16 bits) | | | Drapeau (3 bits) | Décalage fragment (13 bits) |
| Durée de vie (8 bits) | | Protocole (8 bits) | Somme de contrôle de l'en-tête (checksum) (16 bits) | |
| Adresse IP source (32 bits) | | | | |
| Adresse IP destination (32 bits) | | | | |
| Données | | | | |

La signification de ce format est la suivante :

- La longueur de l'en- tête en mots est de 32 bits.
- *Version*: vaut 4 dans la version actuelle et 6 dans la prochaine version. C'est pour cette raison que ces protocoles sont désignés par IPv4 et IPv6. Il s'agit de vérifier la validité du datagramme.
- *Longueur d'en-tête, ou IHL pour Internet Header Length* : il s'agit du nombre de mots de 32 bits constituant l'en-tête (la valeur minimale est 5). Ce champ est codé sur 4 bits.
- *Type de service*: indique comment le datagramme est géré (délai de transit, la sécurité, la priorité…..).
- *Longueur totale:* c'est la longueur totale de datagramme (en-tête+données).
- *Identification* : c'est l'entier qui identifie le datagramme initial de manière unique (utilisé pour la reconstitution à partir de fragments qui ont tous la même valeur).
- *Drapeau* : utilisé pour la fragmentation.
- *Durée de vie* : ce champ indique en secondes, c'est la durée maximale de transit du datagramme sur l'internet.
- *Protocole*: ce champ identifie le protocole de niveau supérieur dont le message est véhiculé dans le champ donné du datagramme.
- *Somme de contrôle de l'en-tête* : ce champ permet de détecter les erreurs survenant dans l'en-tête.
- *Adresse IP source*: ce champ représente l'adresse IP de la machine émettrice, il permet au destinataire de répondre.
- *Adresse IP destination:* c'est l'adresse IP du destinataire du message.

### *(ii) Les adresses IP*

Une adresse IP est une adresse numérique codée sur 32 bits ou 4 octets entre 0 et 255 et notée sous forme décimal pointé comme suit : xxx.xxx.xxx.xxx, par exemple : 192.168.1.0. Elle est constituée d'une paire qui identifie le réseau et la machine de ce réseau. Chaque ordinateur possède une adresse IP.

### *(iii) Les classes IP*

Les adresses IP sont désignées par gammes appelées ***classe***. On définit cinq classe d'adresse notées de A à E mais les trois premiers sont les plus courants, c'est-à-dire les classes A, B et C. La classe D définit les adresses de groupe et la classe E est réservée pour un usage futur.

La classe A, commençant par la séquence binaire 0, réserve 7 bits pour les numéros de réseaux et 24 bits pour les équipements. Puis la classe B commence par la séquence binaire 10, réserve 14 bits pour les numéros de réseaux et 16 bits pour les équipements. Et enfin, la classe C, commençant par 110, réserve 21 bits pour les réseaux et 8 bits pour les équipements. On va récapituler les classes IP dans le tableau 3 suivant :

***Tableau A1.3:*** Les classes IP

| | | | |
|---|---|---|---|
| Classe A | 0 | Réseaux (7 bits) | Equipements (24 bits) |
| Classe B | 10 | Réseaux (14 bits) | Equipements (16 bits) |
| Classe C | 110 | Réseaux (21 bits) | Equipements (8 bits) |

### *(iv) Le routage IP*

- Le routeur :

Un routeur est un équipement d'interconnexion de réseaux informatiques permettant d'assurer le routage de paquets entre deux réseaux et de déterminer le chemin qu'un paquet de données va emprunter. Les routeurs étaient des ordinateurs ayant deux ou plusieurs cartes réseaux dont chacune est reliée à un réseau différent.

Un routeur a pour rôle de filtrer et de transmettre les paquets entrants en se basant sur l'adresse de destination du paquet et de sa table de routage, il peut sélectionner les meilleurs chemins et assurer un contrôle de flux.

- Routage :

Le routage c'est l'acheminement des datagrammes. Afin d'acheminer les datagrammes ou les paquets entre les stations, le routeur constitue la table de routage.

Le routage est effectué à partir du numéro de réseau de l'adresse IP de l'hôte de destination. La table contient, pour chaque numéro de réseau à atteindre, l'adresse IP du

routeur auquel il faut envoyer le datagramme. Elle peut également contenir une adresse de routeur par défaut et l'indication de routage direct. La difficulté du routage provient de l'initialisation et de la mise à jour des tables de routage.

- La table de routage :

La table de routage c'est une table de correspondance entre l'adresse de la machine visée et le nœud suivant auquel le routeur doit délivrer le message. Pour choisir le meilleur chemin et acheminer les datagrammes, les algorithmes de routage renseignent des tables de routage au niveau de chaque routeur.

### 3. Le protocole UDP (*User Datagram Protocol*)

#### *(i) Définition*

Le protocole UDP est un service de transport non fiable. Il permet aux applications d'échanger les datagrammes.

Ce protocole utilise la notion de port qui permet de distinguer les différentes applications qui s'exécutent sur une machine. En plus du datagramme et de ses données, un message UDP contient, à la fois, un numéro de port source et un numéro de port destination.

Le protocole UDP est donc fourni un service de transport le plus simple possible pour les applications.

#### *(ii) Segmentation*

Voici l'en-tête du segment UDP qui montre dans le tableau 4:

***Tableau A1.4:*** Format d'un segment UDP

| | |
|---|---|
| Port Source<br>(16 bits) | Port destination<br>(16 bits) |
| Longueur<br>(16 bits) | Somme de contrôle de l'en-tête<br>(checksum)<br>(16 bits) |
| Données | |

La signification de ces différents champs est la suivante:

- *Port Source*: il s'agit du numéro de port correspondant à l'application émettrice du segment UDP. Ce champ représente une adresse de réponse pour le destinataire. Ainsi, ce champ est optionnel, cela signifie que si l'on ne précise pas le port source, les 16 bits de ce champ seront mis à zéro, auquel cas le destinataire ne pourra pas répondre (cela n'est pas forcément nécessaire), notamment pour des messages unidirectionnels.
- *Port Destination*: ce champ contient le port correspondant à l'application de la machine destinataire à laquelle on s'adresse.

▪ *Longueur*: ce champ indique la longueur totale du segment, en-tête comprise, or l'en-tête a une longueur de 4 x 16 bits (soient 8 x 8 bits) donc le champ longueur est nécessairement supérieur ou égal à 8 octets.

▪ *Somme de contrôle de l'en-tête* : il s'agit d'une somme de contrôle réalisée de telle façon à pouvoir contrôler l'intégrité du segment.

## 4. Le protocole ICMP (*I*nternet *C*ontrol *M*essage *P*rotocol)

### *(i) Définition*

Le protocole ICMP (Internet Control Message Protocol) est un protocole qui permet de gérer les informations relatives aux erreurs aux machines connectées.

### *(ii) Principe*

Dans le système en mode non connecté d'Internet, chaque passerelle et chaque machine fonctionnent de façon autonome. Le routage et l'envoi des datagrammes se font sans coordination avec l'émetteur.

Ce système fonctionne bien, tant que toutes les machines n'ont pas de problème et que le routage est correct, mais cela n'est pas toujours le cas. En dehors des pannes du réseau et des équipements terminaux, les problèmes arrivent lorsqu'une machine est temporairement, ou de façon permanente déconnectée du réseau, ou lorsque la durée de vie du datagramme expire, ou enfin, lorsque la congestion d'une passerelle est trop importante.

Pour permettre aux machines de rendre compte de ces anomalies de fonctionnement, on a ajouté à Internet un protocole d'envoi de messages de contrôle, c'est le protocole ICMP.

Le destinataire d'un message ICMP n'est pas un processus application, mais le logiciel Internet de la machine concernée. Quand un message est reçu, IP traite le problème.

### *(iii) Messages ICMP*

Les messages ICMP ne sont pas uniquement transmis à partir des passerelles. En effet, n'importe quelle machine du réseau peut envoyer des messages à n'importe quelle autre. Cela permet d'avoir un protocole unique pour tous les messages de contrôle et d'information.

Chaque message a son propre format et permet de rendre compte de l'erreur jusqu'à l'émetteur du message. Les messages ICMP sont transportés dans la partie « Données » des datagrammes IP. Comme n'importe quel autre datagramme, ils peuvent être perdus.

En cas d'erreur d'un datagramme contenant un message de contrôle, aucun message de rapport de l'erreur ne sera transmis, et cela, pour éviter les avalanches.

Le tableau 5 suivant montre le format de messages ICMP :

***Tableau A1.5:*** Format d'un message ICMP

| Type (8 bits) | Code (8 bits) | Somme de contrôle de l'en-tête (checksum) (16 bits) |
|---|---|---|
| Données | | |

ANNEXE A2

SYNTAXES ET PARAMETRES DES FONCTIONS DE WINSOCK

## A2.1. La fonction WSAStartup()

Syntaxe:

```
WSAStartup(MAKEWORD(2, 0),  &wsaData);
```

Paramètres:

- **wsaData** est une structure contenant les informations sur l'exécution de Winsock.
- La fonction ***WSAStartup()*** fait une demande de la version de Winsock sur le système.
- La macro **MAKEWORD (2,0)** crée une valeur **WORD** en enchaînant les valeurs indiquées et spécifie que l'on va utiliser la version 2 de Winsock.
- La fonction ***WSAStartup()*** retourne **0** si l'appel a réussi sinon elle retourne une valeur quelconque.

## A2.2. *Les fonctions* bind() *et* connect()

Syntaxes:

```
SOCKADDR_IN   sockad;

sockad.sin_family                   = PF_INET;
sockad.sin_addr.s_addr              = inet_addr("127.0.0.1");
sockad.sin_port                     = htons(5050);

bind(socket, (SOCKADDR *)&sockad, sizeof(sockad));

connect(socket, (SOCKADDR *)&sockad, sizeof(sockad));
```

Paramètres:

- Le paramètre **sockad.sin_family** spécifie la famille d'adresse à utiliser ; ici c'est la famille d'adresse PF_INET.
- Les paramètres **sockad.sin_addr.s_addr** et **sockad.sin_port** spécifient respectivement l'adresse IP et le numéro du port auquel le socket sera lié.

- La fonction ***inet_addr()*** convertit une chaine de caractère contenant une adresse IP en un entier représentant cette adresse.

- La fonction ***htons()*** permet de réarranger l'ordre des octets de l'entier passé en argument de façon à ce qu'il corresponde au format requis sur le réseau.

- L'opérateur ***sizeof()*** donne la quantité de stockage en octets pour stocker un objet du type de l'opérande et permettant ainsi d'éviter d'indiquer la tailles des données dans le programme.

### A2.3. *La function* listen()

Syntaxe:

```
int listen(SOCKET socket, int backlog);
```

Paramètres:

- La fonction ***listen()*** retourne 0 si l'appel a réussi sinon elle retourne SOCKET_ERROR.

- **backlog** est la longueur maximale de file d'attente des demandes des connexions. Si on le met à 1, un seul client pourra se connecter au serveur.

### A2.4. *Les fonctions* send() *et* recv()

Syntaxes:

```
int bytesSent, bytesRecv;
char recvbuf[32], sendbuf[32] = "Données a envoyés.";

bytesSent = send(socket, sendbuf, strlen(sendbuf), 0);

bytesRecv = recv(socket, recvbuf, 32, 0);
```

Paramètres:

- Les entiers **bytesSent** et **bytesRecv** reçoivent respectivement le nombre d'octets envoyé et le nombre d'octets reçu.

- **sendbuf** et **recvbuf** sont des tampons contenant respectivement les données à envoyer et les données reçues.

- La fonction ***strlen()*** permet d'obtenir la longueur de la chaine de caractère.
- Le dernier paramètre est rarement utilisé car la plupart du temps on le met tout simplement 0 mais en fait son rôle est de spécifier la manière de réaliser l'opération en cours. Par exemple, dans le cas d'une réception de données, s'il est égal à **MSG_PEEK**, les données ne seront pas retirées du tampon de réception **recvbuf** après leur réception et dans le cas d'une émission de données, s'il est égal à **MSG_DONTROUTE** les données ne seront pas routées.

### A2.5. *Les fonctions* shutdown() *et* closesocket()

Syntaxes:

```
int shutdown(SOCKET sockect_ client, int how);
int closesocket(SOCKET sockect_ client);
```

Paramètres:

Le paramètre **how** spécifie le type d'opération qui ne sera plus autorisé, il peut avoir trois valeurs possibles:

- SD_RECEIVE : pour rejeter tous les appels à la fonction ***recv()*** sur le socket.
- SD_SEND : pour rejeter tous les appels à la fonction ***send()*** sur le socket.
- SD_BOTH : pour rejeter tous les appels aux fonctions ***recv()*** et ***send()*** sur le socket.

ANNEXE A3

LES STRUCTURES CORRESPONDANTS AUX EN-TETES ET LES STRUCTURES PERSONNALISEES

### A3.1. La structure correspondant à l'en-tête d'un datagramme IP.

```
typedef struct  iphdr
{
unsigned char  verlen;        /* version + la longueur de l'en tète */
unsigned char  tos;           /* type de service */
unsigned short  tot_len;      /* longueur totale du datagramme */
unsigned short  id;           /* identification */
unsigned short  offset;       /* décalage */
unsigned char  ttl;           /* durée de vie du paquet */
unsigned char  protocol;      /* protocole */
unsigned short  checksum;     /* somme de contrôle */
unsigned int  saddr;          /* adresse IP source */
unsigned int  daddr;          /* adresse IP destinataire */
} IP_HDR;
```

### A3.2. La structure correspondant à l'en-tête d'un message ICMP

```
typedef struct icmphdr
{
    unsigned char type;             /* type du message ICMP */
    unsigned char code;             /* code*/
    unsigned short checksum;        /* checksum */

} ICMP_HDR;
```

### A3.3. La structure correspondant à l'en-tête d'un segment TCP

```
typedef struct tcphdr
{
    unsigned short sport;           /* port source */
    unsigned short dport;           /* port de destination */
    unsigned int seqnum;            /* numéro de séquence */
    unsigned int acknum;            /* accusé de réception */
    unsigned char dataoffset;       /* décalage des données (data offset) */
    unsigned char flags;            /* flags */
    unsigned short windows;         /* fenêtre */
    unsigned short checksum;        /* Somme de contrôle */
    unsigned short urgpointer;      /* pointeur de données urgentes */

} TCP_HDR;
```

### A3.4. La structure correspondant à l'en-tête d'un segment UDP

```
typedef struct udphdr
{
    unsigned short sport;          /* port source */
    unsigned short dport;          /* port de destination */
    unsigned short  tot_len;       /* longueur totale du segment */
    unsigned short checksum;       /* Somme de contrôle */
} UDP_HDR;
```

### A3.5. Remplissage de la structure IP pour le forgeur de paquets:

```
unsigned short packet_size, ip_version, ip_len;
struct iphdr * ip;      /* la structure IP */

/* Calcul de la taille du paquet */
packet_size = sizeof(struct iphdr) + sizeof(struct icmphdr);
ip = (struct iphdr *) malloc (sizeof(struct iphdr));
memset(ip, 0x0, sizeof(struct iphdr));
ip_len = sizeof(struct icmphdr) / sizeof(unsigned long);
ip_version = 4;

/* Remplissage la structure IP */
ip->verlen = (ip_version << 4) | ip_len;
ip->tos = 0;
ip->tot_len = htons(sizeof(struct iphdr) + sizeof(struct icmphdr));
ip->id = 1;
ip->offset = 0;
ip->ttl = 255;
ip->protocol = IPPROTO_ICMP;  /*Protocole de l'en-tete suivant*/
ip->saddr = ip_source;        /* adresse IP source */
ip->daddr = ip_destination;   /* adresse IP de destination */
ip->checksum = 0;

/* Calcul du checksum avec la fonction */
ip->checksum = checksum(&ip, sizeof(struct iphdr));
```

### A3.6. Remplissage de la structure ICMP pour le forgeur de paquets ICMP:

```
struct icmphdr * icmp;        /* la structure ICMP */

icmp = (struct icmphdr *) malloc(sizeof(struct icmphdr));
memset(icmp, 0x0, sizeof(struct icmphdr));

/* Remplissage de la structure */
icmp->type = 8;   /* echo request */
icmp->code = 0;
icmp->checksum = 0;

/* Calcul du checksum avec la fonction */
icmp->checksum = checksum(&icmp, sizeof(struct icmphdr));
```

### A3.7. La fonction checksum()

La fonction ***checksum()*** permet de calculer la valeur du champ « somme de contrôle » de l'en-tête. Ce champ contient une valeur codée sur 16 bits qui permet de contrôler l'intégrité de l'en-tête afin de déterminer si celui-ci n'a pas été altéré pendant la transmission. La somme de contrôle est le complément à un de tous les mots de 16 bits de l'en-tête (champ somme de contrôle exclu). Celle-ci est en fait telle que lorsque l'on fait la somme des champs de l'en-tête (somme de contrôle incluse), on obtient un nombre avec tous les bits positionnés à 1.

Voici le code de la fonction ***checksum().***

```
USHORT checksum(void * p, int size)
{
    unsigned long cksum = 0;
    USHORT * buffer = p;

    while (size >1)
    {
        cksum + = *buffer++;
        size - = sizeof(USHORT);
    }
    if (size)
    {
        cksum + = *(UCHAR*)buffer;
    }
    cksum = (cksum >> 16) + (cksum & 0xffff);
    cksum + = (cksum >>16);

    return (USHORT)(~cksum);
}
```

### A3.8. La structure MYAPP

```
typedef struct _MYAPP {
      HWND hWnd;
      SOCKET sock, client;
      UINT_PTR uTimer;
} MYAPP;
```

### A3.9. La structure SCREEN

```
typedef struct _SCREEN {
      HDC hDC, hMemDC;
      HBITMAP hBitmap;
      BITMAP Bitmap;
      LONG bmBufSize;
} SCREEN;
```

# REFERENCES


[1] : IA551, «Cours Téléinformatique », 5ème année, Département Electronique, ESPA, 2008

[2] : E510, « Cours Réseau local », 5ème année, Département Electronique, ESPA, 2008

[3] : Roux Benjamin, « Les sockets en C », Janvier 2008

[4] : Bob et CGi, « Tutorial d'initiation à la programmation avec l'API Windows », novembre 2003

[5] : Patrick Smacchia, « Architecture Windows NT/2000/XP », Année 2002

[6] : David J. Kruglinski, George Shepherd et Scott Wingo, « Atelier Microsoft Visual C++ 6.0 », Edition 2001.

[7] : David A. Schmitt, « International Programming for Microsoft Windows», Année 2000

[8] : Charles Petzold, « Programming Windows », 5ième édition, Année 1998

[9] : Jeffrey Richter, «Programming Applications for Microsoft Windows », 4ième édition, Année 1999

[10] : Anthony Jones et Jim Ohlund, « Network Programming for Microsoft Windows », Année 1999

[11] : David TILLOY, « Introduction aux réseaux TCP/IP », Département Informatique,

[12] : Chin Shiuh Shieh, « Internet Programming in Microsoft WinSock API »

[13] : Dr Dimitri Konstantas, « Infrastructure de Communication »

[14] : http://msdn.microsoft.com

[15] : http://www.commentcamarche.net

**Auteur:**

M^me^ NIRINASOA Viviane

**Titre: « ETUDE DE LA PROGRAMMATION RESEAU SOUS WINDOWS »**

**Nombre de pages** : 68

**Nombre de figures** : 19

**Nombre de tableaux** : 09

---

**Résumé :**

Cet ouvrage présente l'étude de base de la programmation sous Windows et plus particulièrement la programmation réseau. L'API Windows étant encore un sujet presque inconnu par le grand public et même ignoré par certains développeurs, l'objectif de ce mémoire permet de montrer son importance dans le développement d'applications par le biais de Winsock. Ce dernier est une API spécialisée dans la communication réseau utilisant les sockets.

Au terme de cette étude, on va développer cet API en utilisant la bibliothèque Winsock.

---

**Mots clés**: API, Windows, Sockets, Serveur, Winsock, Visual Studio, RAW.

**Rapporteur** : Monsieur RAKOTONDRASOA Justin

**Adresse de l'auteur** :

Lot 04-2966 Ambohimandamina

MAJUNGA(401)

**Téléphone**: 0330723433