Programmation réseau dans toute sa splendeur Windows Socket Api

Par Phoneus [Thierry Brémard]

AVANT PROPOS

Dans La programmation réseau dans toute sa splendeur je tente d'expliquer les concepts qui doivent être maîtrisés par tout programmeur qui veut se lancer dans le développement réseau.
Les nombreuses fonctions écrites en C sont là pour favoriser et accélérer votre compréhension. Vous découvrirez dans cet article les plus grands secrets de l'Internet: vous allez notamment apprendre le File Transfert Protocol (création d'un client et d'un serveur).
Si vous avez déjà commencé à programmer des applications réseau et que vous avez laissé tomber en vous rendant compte qu'il y a trop de subtilités à connaître, c'est que ce document est aussi pour vous.
Avec une étude progressive vous serez, après avoir lu ce document, à la hauteur de développer tout type d' application utilisant le réseau.
Tous les listings présentés dans ce document ont été testés avec succès.
Cet article requiert la connaissance du langage C et suppose que vous avez quelques notions des protocoles TCP IP.
A travers tout ce document je construirai avec vous des fonctions qui forment une librairie. L'utilisation de cette librairie vous permettra de réaliser des applications rapidement, avec plus de simplicité, tout en restant au même niveau que les API. Un récapitulatif de toutes ces fonctions est disponible à la fin.

J'ai pris beaucoup de temps pour rédiger ce document seul un feed back de votre part me permettra de l'améliorer. Mon mail se situe tout à la fin.
Voici une petite idée des différentes choses que j'arrive à réaliser aujourd'hui et que vous pourrez aussi implémenter grâce à ce document :
***Client ftp, client mail (pop et smtp)**
***Serveur sur ma machine qui permet sur demande d'exécuter n'importe qu'elle commande (aussi bien listing d'un repertoir que lecture d'un fichier mp3).**
***Serveur Chat**

Ce document à un but didactique et n'a pas la prétention d'être une référence parmi les milliers d'autres articles traitant du réseau, c'est un document parmi tant d'autes avec ses points positifs et négatifs.

J'ai besoin de vos commentaires pour amélirorer ce document, n'hésitez pas ! bremardt@esiee.fr

Table des matières.

Last modification: 14/10/03

I Gestion de winsock.dll

Avant d'avoir recours aux fonctions réseau de Windows, il faut lancer la dll(dynamic link Library) winsock.dll. C'est dans cette dll que sont les fonctions que nous allons utiliser.

1) Initialisation de winsock

La fonction WSAStartup() initialise winsock.dll pour un processus

```
int WSAStartup (
   WORD wVersionRequested,
   LPWSADATA lpWSAData
  );
```

Les paramètres sont les suivants :

WVersionRequested contient les numéros de la version de Winsock que le programme doit utiliser. L'octet de poids fort indique le numéro de la version, l'octet de poids faible le numéro de la révision. (Pour version 2.0, MSB=2 et LSB=0).

LpWSAData est un paramètre de sortie. C'est un pointeur vers une structure de données : DATA qui va recevoir les détails de l'implémentation Winsock.

Définition da la structure WSAData : *(contenue dans le header winsock.h)*

```
typedef struct WSAData {
        WORD        wVersion;
        WORD        wHighVersion;
        char        szDescription[WSADESCRIPTION_LEN+1];
        char        szSystemStatus[WSASYS_STATUS_LEN+1];
       unsigned short        iMaxSockets;
       unsigned short        iMaxUdpDg;
       char FAR *            lpVendorInfo;
} WSADATA;
typedef WSADATA FAR *LPWSADATA;
```

La macro MAKEWORD crée un unsigned int de 16 bits en concaténant deux valeurs d'unsigned char, données en paramètre.

```
WORD MAKEWORD(
   BYTE bLow,   // Octet de poids faible
   BYTE bHigh   // Octet de poids fort
  );
```

Pour info, la macro MAKEWORD est définie de la manière suivante :

```
#define MAKEWORD(a, b)     ((WORD) (((BYTE) (a)) | ((WORD) ((BYTE) (b))) << 8))
```

Pour appeler WSAStartup, nous ferons :

```
  WORD wVersionRequested;
   WSADATA wsaData;
   int err;

   wVersionRequested = MAKEWORD( 2, 0 );            //On veut utiliser la version 2.0 de winsock
  err = WSAStartup( wVersionRequested, &wsaData );
```

WSAStartup() retourne 0 si elle réussit à initialiser winsock.dll

Etant donné que cette fonction se charge de lancer winsock.dll ce devra être la première fonction qui devra être appelée dans un programme utilisant les fonctions winsock.

Pour initialiser winsock.dll nous utiliserons à présent la fonction initwinsock() que voici:

```
/*******************************************************\
```

```
|                    INITWINSOCK
|cette fonction charge winsock.dll. cette opération est
|absolument nécessaire pour utiliser les API Winsock
|(on n'oubliera pas de fermer Winsock avec WSACleanup() )
|Retourne :1 si succès ou SOCKET_ERROR si échec
\*********************************************************/

int initwinsock(void)
{
   WORD wVersionRequested;
   WSADATA wsaData;
   int err;

   wVersionRequested = MAKEWORD( 2, 0 );
   err = WSAStartup( wVersionRequested, &wsaData );
   if ( err != 0 )
   {
      return SOCKET_ERROR;                //impossible de charger winsock.dll
   }

//-- Ceci permet de vérifier que Winsock supporte bien la version 2.0 --//
   if ( LOBYTE( wsaData.wVersion ) != 2 || HIBYTE( wsaData.wVersion ) != 0 )
   {
      WSACleanup();
      return SOCKET_ERROR;                //impossible de charger winsock.dll
   }
   else
   {
      return 1;                     //Winsock.dll succefully initialised
   }
}
```

2) Fermeture de winsock.dll

A la fin de tout programme utilisant les API Winsock, on devra fermer proprement notre dll en lançant la fonction WSACleanup().

Nous venons de présenter les procédures indispensables qui sont à intégrer dans tout programme tournant sur le réseau. Nous allons voir maintenant quelles sont les fonctions mises à notre disposition pour communiquer à travers le réseau.

II Les fonctions principales d'un programme réseau : connexion, réception, envoi, conversion…

Vous avez sans doute entendu parlé du terme client serveur :
Un client est un programme qui se connecte et demande des services au serveur.
Un serveur est un programme qui se lance sur une machine et qui attend la connexion des clients pour leur fournir les données dont ils ont besoin.
Nous allons étudier indépendamment la structure d'un client puis celle d'un serveur afin de mettre en évidence les fonctions principales utilisées par chacun de ces deux types de programmes.

1) Le client

Le client est un programme qui permet à une machine de se connecter à une autre machine sur laquelle est lancé un serveur.
Lors de son exécution, un programme client réalise 5 étapes :

- Il prend l'adresse et le numéro de port du serveur auquel il doit se connecter.
- Il crée une socket.(une socket est une variable qui est utilisée dans les opérations sur le réseau ; c'est assimilable a un fichier texte sur lequel on peut réaliser des opérations de lecture et d'écriture)
- Il réalise une connexion de sa socket sur la machine distante .
- Il dialogue ensuite avec le programme serveur de la machine distante.
- Il ferme la connexion et sa socket quand il n'a plus besoin du serveur.

a)Adresse et numéro de port du serveur

L'adresse du serveur, c'est une série de chiffres séparés par des points. On appelle ce genre d'adresse l'adresse ip ; elle est du type : 123.156.189.147
Cette adresse ne désigne qu'une seule machine connectée : chaque machine a une ip unique.
La plupart du temps, on ne fait pas attention à cette adresse car il existe des protocoles de résolution de nom , qui à patir d'un nom de serveur font correspondre une adresse ip. Nous verrons les protocoles de résolution de noms dans la prochaine partie.

Le numéro de port est un chiffre qui permet de situer la position du serveur sur la machine distante ; c'est en quelque sorte le complément de l'adresse ip : grâce à l'adresse ip, on accède à une machine bien déterminée et grâce au numéro de port,on accède à un endroit spécifique sur la machine dite serveur .
Par exemple une machine dont l'adresse ip est 123.156.189.147 peut héberger plusieurs serveurs : un serveur FTP(File Transfert Protocole) qui sera lancé sur le port 21
Et un serveur http (HyperText Transfert Protocol) sur le port 80.
Ayant la même adresse, on différenciera ces serveurs par leurs numéros de port.

Pour réaliser la connexion de notre client , nous devrons donc connaître une adresse ip et un n° de port valides.

Utilisation de l'adresse ip et du numéro de port dans un programme client :
Nous devons, pour réaliser la connexion à notre serveur, copier dans une structure l'ip et le n° de port. Cette structure est struct sockaddr_in

```
struct sockaddr_in
{
        short   sin_family;         //le type d'adresse
        u_short sin_port;           //le numéro du port sur la machine distante(avec les bits ordonnés pour
                                    //l'utilisation réseau)
        struct  in_addr sin_addr;   //l'adresse ip du serveur
        char    sin_zero[8];        //données inutilisées à mettre à 0
};
```

Pour remplir la structure de destination nous faisons:

```
  struct sockaddr_in sin ;          //déclaration de la variable sin de type struct sockaddr_in

   /* R empli la structure de destination*/
   sin.sin_family  = AF_INET;
   sin.sin_port    =  htons(port);
   sin.sin_addr.s_addr=inet_addr(adresse);
   sin.sin_zero    = 0;
```

Vous pouvez constater que nous utilisons les fonctions inet_addr() et htons() ,que nous n'avons pas présentées. Ces fonctions transforment les données d'adressage au format binaire utilisé par les fonctions qui ont recours à la structure sockaddr_in.
Le type de famille est AF_INET.

b) Création de la socket

Nous devons créer un fichier qui nous permettra, en lecture (recv()) de lire ce que le serveur nous envoie, et en écriture (send()) d'envoyer des messages au serveur.
Malgré mes allusions à la gestion des fichiers, nous n'utiliserons pas une variable FILE pour gérer nos entrées nos entrées sorties, mais une variable de type SOCKET.
Pour les curieux, vous trouverez la définition type socket dans le header winsock.h de la manière suivante :
typedef UINT_PTR SOCKET;
Une variable SOCKET est donc un pointeur int. Ceci pourra vous aider à comprendre comment certaines données sont passées en paramètres dans des fonctions qui à première vue font un passage par valeur.
(Note : sous linux, une socket sera une variable de type int. Les types int et SOCKET ont la même taille, ils pourraient être confondus, néanmoins je n'utiliserai que le type SOCKET dans le reste du document).

Déclarons une variable socket :

```
SOCKET sock ;          //sock est notre socket qui va nous permettre de nous connecter a un serveur
```

Ensuite nous devons rendre cette socket utilisable (comme on rend un descripteur de fichier opérationnel avec fopen) grâce à la fonction socket() :

```
SOCKET socket (
        int af,         //la famille d'adresse
        int type,       //le type d'adresse
        int protocol    //le type de protocole
);
```

Le paramètre af peut prendre la valeur: AF_INET ou PF_INET.Nous utiliserons toujours AF_INET.
Le type de la socket est soit: SOCK_STREAM pour utiliser le Transmission control Protocol (TCP), soit SOCK_DGRAM pour utiliser le protocole UDP (User Datagram Protocol) ou encore SOCK_RAW ,mais SOCK_RAW n'est pas compatible avec tous les compilateurs, paraît il.
Le paramètre protocol reste nul pour SOCK_STREAM et SOCK_DGRAM.
La fonction socket retourne une socket valide dans le cas ou elle réussie , sinon elle retourne la valeur INVALID_SOCKET
Ex: sock=socket(AF_INET,SOCK_STREAM,0);
Nous avons désormais créé une socket. Il ne nous reste plus qu'à la connecter à la machine distante.

c) Réalisation de la connection avec la machine distante

Pour réaliser cette opération nous utilisons la fonction connect() :

```
int connect (
SOCKET s,                       //socket qui vient d'être créé(non connectée)
const struct sockaddr FAR*  name, //adresse de la structure sockaddr_in
int namelen                     //taille de la structure sockaddr_in
);
```

Il est important de noter que le premier paramètre est une socket qui n'est pas connectée : on ne peut pas réaliser plusieurs connections avec la même socket. Pour reconnecter une socket ,il faudra préalablement la fermer avec closesocket()
connect() retourne SOCKET_ERROR en cas d'échec sinon , la connexion a été établie et l'on peut commencer le dialogue entre le client et le serveur.
Ex:

```
err=connect(sock, (struct sockaddr *)&sin,sizeof(struct sockaddr))
if(err==SOCKET_ERROR)
        perror("connect()");
```

d) Dialogue entre le serveur et le client

Après s'être connecté , le seveur envoie toujours ,en principe un message de bienvenue(il me semble que les serveurs http font autrement….)
Nous mettrons donc notre client en écoute tout de suite après la connexion. Pour attendre un message, nous utilisons la fonction recv() :

```
int recv (
SOCKET s,        //la socket connectée au serveur
char FAR* buf,   //l'adresse d'un buffer qui servira pour copier les données recues
int len,         //la taille du buffer de reception
int flags        //flag qui est toujours mis à 0
);
```
recv() retourne SOCKET_ERROR si il y a un problème.

Exemple d'utilisation :
```
err=recv(sock,(char *)buffer,taille,0);
```
(Note Sur linux on n'utilisera pas la fonction recv(), mais la fonction read())
Par défaut recv est bloquante, c'est-à-dire que tant qu'il n'y a pas eut de réception de données, le programme restera à la fonction recv et n'exécutera rien à la suite.

Une fois la réception effectuée, nous allons envoyer des données. Là c'est send() que l'on utilise :
```
int send (
SOCKET s,        //socket connectée à travers laquelle on va envoyer nos données
const char FAR * buf, //l'adresse du buffer que l'on va envoyer
int len,         //longueur en caractères du buffer
int flags        //flag toujours mis à 0
);
```
Comme recv() :send renvoie SOCKET_ERROR en cas d'échec
Exemple d'utililsation :
```
  err=send(sock,buffer,strlen(buffer),0);  //envoi de données
```
(Note Sur Linux, on n'utilisera pas la fonction send() pour envoyer des données ,mais la fonction write)
Note (Très importante): les données reçues par le serveur doivent être terminées par la succession de <Retour Chariot> et <Saut de ligne> ; c'est à dire que chaque chaîne de caractère envoyée sur le réseau doit être de la forme :
" abcdef….mnop\r\n ".
Si vous envoyez une chaîne qui n'est pas terminée par "\r\n", le serveur ne réagira pas : il attend la succession \r\n pour interpréter une commande. Cette règle est vraie aussi bien dans le sens client serveur que dans le sens serveur client.

e) Fermeture de la connexion

La fermeture d'une connexion se fait en deux étapes.
Génaralement on peut dans un premier temps informer le serveur que l'on va quitter grâce à l'envoi de commandes comme 'quit'.
La fermeture de la connexion, dans tous les cas se réalise de la manière suivante :
shutdown() puis closesocket()
```
int shutdown (
SOCKET s,        //la socket à déconnecter
int how          // la procédure de déconnexion
);
```
how peut prendre les valeurs suivantes :
SD_RECEIVE , SD_SEND , SD_BOTH

SD_RECEIVE indique à l'autre coté de la connexion que la socket ne peut plus recevoir des données.
SD_SEND indique que la socket ne peut plus envoyer des donnéses.
SD_BOTH indique que la socket ne peut ni envoyer ni recevoir.
L'appel à shutdown() n'est pas indispensable et n'est pas souvent éffectué. On passe souvent directement à closesocket()

```
int closesocket (
SOCKET s         //la socket à fermer
);
```

Sous windows, pour manipuler les socket nous devrons inclure le header :winsock.h ou winsock2.h.Dans la plupart des listing, il faudra aussi inclure malloc.h, stdio.h, et string.h
Le code d'un client est présenté, ce client se connecte à un serveur attend un message de la part du serveur, puis tourne en boucle en demandant à l'utilisateur de saisir une commande, envoi cette commande au serveur et attend une réponse.

<Code d'un Client>

```c
#include <windows.h>
#include <winsock.h>
#include <stdio.h>
#include <conio.h>

int main (void)
{
    SOCKET sock;            //notre socket
    int err;                //pour choper les valeurs de retour
    char *buffer;           //string contennant les données à envoyer
    struct sockaddr_in sin; //la fameuse structure dans laquelle on met les coordonées du serveur
    int port;

    initwinsock();
    sock=socket(AF_INET,SOCK_STREAM,0);
    if(sock==INVALID_SOCKET)
    {
        puts("Impossible de créer une socket\fin du programme");
        return 0;
    }

    buffer=(char*)malloc(1024);
    printf("connection sur la machine locale\r\n veuillez specifier un numero de port:\r\n");
    scanf("%d",&port);
    fflush(stdin);

    /* rempli la structure sockaddr_in  */
    sin.sin_addr.s_addr=inet_addr("127.0.0.1");
    sin.sin_port=htons(port);
    sin.sin_family=AF_INET;


    err=connect(sock,(const struct sockaddr FAR* )&sin,sizeof(struct sockaddr_in));
    if(err==SOCKET_ERROR)
    {
         perror("Impossible de réaliser la connection");
        return 0;
    }
    puts("Connection reussie");

    while(TRUE)    //boucle infinie
    {
        puts("Attente d'un message de la part du serveur...");
        memset(buffer,0,1024);
        err = recv(sock,buffer,1024,0);
        if(err == SOCKET_ERROR)
        {
            perror("Erreur lors de la reception");
            return 0;
        }
        puts(buffer);
        puts("\nTapez une commande a envoyer au serveur");
        gets(buffer);
        strcat(buffer,"\r\n");
        err=send(sock,buffer,strlen(buffer),0);
        if(err==SOCKET_ERROR)
        {
            perror("Erreur lors de l'envoi de la commande");
            getch();
            exit(0);
        }

        if(!strcmp(buffer,"quit\r\n"))
            break; //quand l'utilisateur saisit la commande quit, on sort de la boucle while
    }

    puts("Fermeture de la socket");
    shutdown(sock,SD_BOTH );
    closesocket(sock);
    WSACleanup();

    return 0;
}
```

<Fin du code client>

2) Le serveur

Le serveur est un programme qui , lancé sur une machine ayant accès à un réseau, attend la connexion de clients et leur fourni des informations dont ils ont besoin.
Un programme serveur réalise , comme le client cinq étapes :

- On lui spécifie le numéro du port sur lequel on veut qu'il s'installe
- Il crée une socket et la fait correspondre avec le numéro du port
- Il attend la connexion d'un client
- Il dialogue avec le client connecté
- Il ferme la connexion

a)Le numéro de port

Nous venons de voir que le client a besoin d'une adresse ip et d'un numéro de port pour se connecter à un serveur. L'adresse ip est spécifique à une machine ; il n'est donc pas envisageable de spécifier une ip au serveur différente de celle de la machine sur laquelle il est lancé ; par contre on doit donner un numéro de port bien précis (compris entre 0 et 2000).
Comme pour le client on rempli une structure sockaddr_in :

```
SOCKADDR_IN sin;    //déclaration de la variable sin de type struct sockaddr_in

sin.sin_family=PF_INET;                 //famille de l'adresse
sin.sin_port=htons(port);               //n°de port
sin.sin_addr.s_addr=0;          // on ne choisit pas notre ip => on met 0
```

b) Création de la socket et liaison de celle-ci avec le n° de port

Comme pour le client, le serveur doit se créer une socket :

```
sock=socket(AF_INET,SOCK_STREAM,0);
```

une fois la socket créé,nous devons faire une étape qui n'intervient pas chez le client : la liaison de la socket avec un n°de port. Pour cela nous utilisons la fonction bind :

```
int bind (
SOCKET s,                               //socket qui vient d'être créé
const struct sockaddr FAR*  name,       //adresse de la structure sockaddr_in
int namelen                             //taille de la structure sockaddr_in
);
```

Exemple d'utilisastion :

```
ret = bind(sock,(LPSOCKADDR)&sin,sizeof(struct sockaddr));
```

bind retourne SOCKET_ERROR si une erreur est arrivée

c) Attente de la connexion d'un client

Un serveur passe la plupart de son temps à attendre la connexion d'un client .Pour cela nous devons utiliser deux fonctins : listen() qui ,comme son nom l'indique , se charge d'écouter.
Cette fonction retourne immédiatement.
Puis nous avons la fonction accept() qui permet d'enregistrer les coordonnées du client afin d'établir le dialogue.

```
int listen (
SOCKET s,       // la socket liéée
int backlog     // nombre maximum de connections  Pour un maximum nous
                // mettrons SOMAXCONN (constante définie dans le header winsock.h
);
```

listen retourne SOCKET_ERROR dans le cas d'un échec.

```
SOCKET accept (
SOCKET s,                   // la socket crée par le serveur
struct sockaddr FAR* addr, // l'adresse optionnelle d'une structure sockaddr_in qui sera remplie avec les
                            // coordonnées du client
int FAR* addrlen            // taille de la struct sockaddr_in
);
```

La fonction accept retourne une socket qui permet de dialoguer avec le client. Cette fonction est bloquante.

Exemple d'utilisation :

```
ret = listen(sock,5);
sock2=accept(sock,(LPSOCKADDR)&sin2,&taille);        // on récupère la socket du client en retour de la fonction
                                                     //accept et on a les infos du client ds la struct sin2
```

d) Dialogue entre client et serveur

le dialogue se réalise de la même manière chez le serveur et chez le client : le serveur envoie des données avec la fonction send() et recoit des données avec recv()

e) Fermeture de la connexion

La fermeture de la conection chez le serveur se réalise de la même manière que chez le client : shutdown() closesocket() sont successivement appelées .

<Code d'un serveur>

```
#include <windows.h>
#include <winsock.h>
#include <stdio.h>
#include <conio.h>


int  main(void)
{
SOCKET sock,s_client;
struct sockaddr_in sin,sin2;
int err;
int port=85;
char *buffer;
char *reponse;
struct sockaddr *sockadd;

    initwinsock();

    sock=socket(AF_INET,SOCK_STREAM,0);
    if (sock==INVALID_SOCKET)
    {
        puts("Impossible de creer une socket");
        return 0;
    }

    sin.sin_family=AF_INET;
    sin.sin_port=htons(port);
    sin.sin_addr.s_addr=0;


    err=bind(sock,(const struct sockaddr FAR* )&sin,sizeof(SOCKADDR_IN));
    if(err==SOCKET_ERROR)
    {
        perror("Erreur de bind()");
        return 0;
    }

    listen(sock,1);
    printf("Attente d'une connexion d'un client sur le port %d\n",port);

    err=sizeof(struct sockaddr_in);
    s_client = accept(sock,(struct sockaddr * )&sin2,(int *)&err);//retourne quand il y a connexion

    puts("Un client est connecte :");
    printf("Addresse IP: %s Port du client:%d\n",inet_ntoa(sin2.sin_addr),ntohs(sin2.sin_port));
    closesocket(sock);      // c'est désormais avec la socket retournée par accept que nous
                            // dialoguerons

    buffer=(char*)malloc(1024);
    strcpy(buffer,"Welcome to our server\r\n");
    send(s_client,buffer,strlen(buffer),0);
    reponse = (char * )malloc(128);

    while(1)
    {
        memset(buffer,0,1024);
        err=recv(s_client,buffer,1024,0);
        if(err==SOCKET_ERROR)
```

```
        {
            perror("Erreur dans la reception");
            return 0;
        }
        printf("Le client a envoye :\n%s\n",buffer);

        if(!strcmp(buffer,"quit\r\n"))
            break;

        snprintf(reponse,128,"Time: %d\r\n",GetTickCount());

        err=send(s_client,reponse,strlen(reponse),0);
        if(err==SOCKET_ERROR)
        {
            puts("Erreur dans l'envoi de la reponse");
            return 0;
        }
    }

    shutdown(s_client,SD_BOTH);
    closesocket(s_client);
    WSACleanup();
    free(buffer);
    free(reponse);

    puts("Programme serveur termine");
    return 0;
}
```

<Fin du code serveur>

Screen shot of client and serveur programs :

```
G:\Prog\Articles\WSA\client>client
connection sur la machine locale
 veuillez specifier un numero de port:
85
Connection reussie
Attente d'un message de la part du serveur...
Welcome to our server


Tapez une commande a envoyer au serveur
bonjour
Attente d'un message de la part du serveur...
Time: 1005671


Tapez une commande a envoyer au serveur
comment allez vous ?
Attente d'un message de la part du serveur...
Time: 1018953


Tapez une commande a envoyer au serveur
ces chiffres que vous m'envoyez sont ils en
relation ???
Attente d'un message de la part du serveur...
Time: 1043234


Tapez une commande a envoyer au serveur
quit
Fermeture de la socket
```

```
G:\Prog\Articles\WSA\serveur>serveur
Attente d'une connexion d'un client sur le port 85
Un client est connecte :
Addresse IP: 127.0.0.1 Port du client:1032
Le client a envoye :
bonjour

Le client a envoye :
comment allez vous ?

Le client a envoye :
ces chiffres que vous m'envoyez sont ils en relation ???

Le client a envoye :
quit

Programme serveur termine
```

3) Fonctions de conversion

a) L'utilité d'une opération de conversion

Nous avons dans les exemples précédents utilisé des fonctions de convers ion sans les présenter…Nous allons , dans cette section voir les différentes fonctions de conversion qui peuvent être utilisées dans un programme réseau et expliquer leurs utilités.

Les fonctions de conversion sont utilisées pour mettre des données au format correspondant à un type d'argument d'une fonction; en d'autres termes, il se peut qu'une fonction retourne une valeur, et que vous souhaitiez utiliser cette valeur de

retour dans une autre fonction. Si cette valeur de retour n'a pas le bon format, vous devrez avant appeler la fonction qui doit utiliser le résultat, utiliser une fonction de conversion.
Les fonctions de conversion sont utilisées pour faire passer des données du format ASCII au format réseau (utilisé dans les headers TCP/IP) et vice versa. D'autres fonctions sont là pour changer la disposition en mémoire des octets dans un short ou dans un int.

b) Convertir le numéro de port

Nous devons convertir le numéro de port quand nous voulons remplir une structure sockaddr_in.: le champ sin_port doit être rempli par le numéro de port dont les bits doivent être ordonnés pour le réseau.(Network By Order)
Pour réaliser la conversion [Nombre décimal au format normal]=>[Nombre en Network By Order], nous avons la fonction htons() dont voici le prototype:

```
u_short htons (             //Host TO Network Short
   u_short hostshort        // Nombre short à un format normal
  );
```

Dans le même type de fonction , nous avons:

```
u_long htonl (              //Host TO Network Long
   u_long hostlong          //nombre long à un format normal
     );
```

Ces deux fonctions ont la même finalité: convertir des données du format local au format réseau. La seule différence entre ces deux fonctions réside dans le fait qu'elles acceptent des nombres plus ou moins long. Le type de nombre est désigné par la dernière lettre de chaque fonction: htons(): S comme short et htonl(): L comme Long.

Si la structure sockaddr_in a le numéro de port mis en ordre réseau, il faut savoir convertir se numéro au format normal: quand un client se connecte à un serveur, le serveur a la possibilité de compléter une structure sockaddr_in avec les caractéristiques du client.
Pour retirer un numéro normal, nous utiliserons les fonctions: ntohs() ou ntohl().

```
u_short ntohs (             //Network TO Host Short
   u_short netshort         //Un nombre dont les bits sont ordonnés pour le réseau.
  );
```

Même fonction , mais pour les Long (non utilisée dans tout cet article)

```
u_long ntohl (
   u_long netlong
  );
```

Pour nous exercer à la conversion nous pouvons réaliser un petit programme:

```
void main(void)
{
u_short port;  //port est du type unsigned short ( on trouve dans winsock.h : typedef unsigned short  u_short;)
struct sockaddr_in sin;

   puts("Entrez un numero de port");
   scanf("%d",&port);
   fflush(stdin);

   sin.sin_port=htons(port);
   printf("Le chiffre %0x converti donne : %0x\n",port,sin.sin_port);
   printf("\n\nLe numero du port dans la struc sockaddr_in est: %0x, il devient, une fois converti: %0x\n",sin.sin_port,ntohs(sin.sin_port));
}
```

Les nombres sont affichés sous format hexadécimal, ceci permet de comprendre comment la traduction marche:
Pour faire passer un nombre du format local au format réseau, il faut mettre ce nombre sous notation hexadécimal ,puis on inverse les deux octets:
Soit le nombre x, en supposant que x s'écrive :0x7f4b, le nombre traduit aura pour écriture hexadécimale: 0X4B7F

A partir de maintenant quand une fonction prend en paramètre un unsigned short en Network by Order, il faudra convertir tout nombre grâce à la fonction htons() de même quand une fonction retourne un nombre en Network By Order et que nous voulons utiliser ce nombre, on appellera ntohs()

c) Convertir l'adresse IP

L'adresse IP doit aussi être convertie avant d'être utilisée sur le réseau. Pour convertir une adresse IP, nous avons les deux fonctions: inet_addr() et inet_ntoa()

inet_addr() converti une adresse IP du format décimal pointé au format réseau.

```
unsigned long inet_addr (
   const char FAR * cp      //chaîne de caractères désignant une adresse IP au format "a.b.c.d"
  );
```

inet_addr() retourne un unsigned long. L'unsigned long retourné correspond à l'adresse IP mise au format réseau

La fonction qui permet de traduire une adresse IP du format réseau au format décimal pointé est inet_ntoa() ; il s'agit en quelque sorte de l'inverse de la fonction inet_addr():

```
char FAR * inet_ntoa (  //Network TO Ascii
   struct in_addr in
  );
```

Cette fonction retourne un pointeur vers une adresse IP de la forme "a.b.c.d" , c'est-à-dire au format décimal pointé.

La fonction inet_ntoa() est un peu difficile à utiliser car elle prend comme paramètre une structure in_addr (et non un unsigned long comme on pourrai le penser). On peut trouver la déclaration de la structure in_addr dans le header winsock.h :

```
struct in_addr {
        union {
                struct { u_char s_b1,s_b2,s_b3,s_b4; } S_un_b;
                struct { u_short s_w1,s_w2; } S_un_w;
                u_long S_addr;
        } S_un;
#define s_addr  S_un.S_addr
                        /* can be used for most tcp & ip code */
#define s_host  S_un.S_un_b.s_b2
                        /* host on imp */
#define s_net   S_un.S_un_b.s_b1
                        /* network */
#define s_imp   S_un.S_un_w.s_w2
                        /* imp */
#define s_impno S_un.S_un_b.s_b4
                        /* imp # */
#define s_lh    S_un.S_un_b.s_b3
                        /* logical host */
};
```

Ce qui est équivalent à la déclaration suivante:

```
struct in_addr
{
        u_long s_addr,
        u_char s_host,
        u_char s_net,
        u_short s_im p,
        u_char s_impno,
        u_char s_lh
};
```

Pour utiliser la fonction inet_ntoa(), nous devrons donc faire les étapes suivantes:
Déclaration d'une variable de type struct in_addr,
Copie dans le champ s_addr de l'unsigned long correspondant à l'adresse IP que l'on veut connaître.
Appel de la fonction inet_addr()

Nous devrions théoriquement faire ceci :

```
struct in_addr in;
      memcpy((unsigned long *)&in.s_addr,(unsigned long *)&ip_number,sizeof(u_long));
      printf("L'ip est : %s\n", inet_ntoa(in.in_addr));
```

Mais nous ne ferons pas exactement ceci dans notre programme, simplement parceque la structure in_addr n'est plus utilisée aujourd'hui....et oui ("This is Evolution")
Nous avons dans le header winsock.h une autre structure que nous allons utiliser à la place de in_addr, il s'agit de la structure sockaddr_in que voici :

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

On constate que l'on retrouve dans le troisième champ une structure in_addr. Parfait , on sait maintenant ce qu'il nous reste à faire.

```
char *trad(unsigned long u)
{
struct sockaddr_in sin;

    memcpy((unsigned long *)&sin.sin_addr.s_addr,(unsigned long *)&u,sizeof(u));
    return inet_ntoa(sin.sin_addr);
}

void main(void)
{
    char *buffer;
    unsigned long u;

    buffer=(char*)malloc(256);
    memset(buffer,0,256);
    strcpy(buffer,"207.228.238.101");
    u=inet_addr(buffer);
    printf("l'adresse ip: %s convertie devient: %0x\n",buffer,u);
    printf("La conversion de %0x grace a la fonction inet_ntoa() donne : %s",u,trad(u));
    getch();
    exit(0);
}
```

Nous obtenons le résultat :
L'adresse ip: 207.228.238.101 convertie devient: 65eee4cf
La conversion de 65eee4cf grâce a la fonction inet_ntoa() donne : 207.228.238.101

La notation en hexadécimal nous permet de faire une remarque analogue à la remarque que nous avions faite dans la section de traduction des numéros de port:
Le format réseau s'obtient de la manière suivante :
On isole les nombres présents entre chaque points. Chaque section d'une adresse IP ne pouvant excéder 256, chaque nombre peut être mis sur un octet.
Le nombre retourné par la fonction inet_ntoa() est donc constitué de 4 octets, chaque octet correspondant à une partie de l'adresse ip sous format décimal pointé. Les octets sont rangés dans l'ordre inverse : le premier correspondant à la forme décimale pointée se situe à la fin de l'unsigned long

Exemple : 207 en hexa vaut :0xcf, 228=0xe4 , 238=0xee , 101=0x65
On a donc les octets successifs : 0x cf e4 ee 65
En inversant l'ordre, on a bien le même résultat que inet_addr() : 0x65eee4cf (Ce nombre sous forme décimal vaut : 1710154959 ,ce qui nous indique strictement rien ; il est ici flagrant que la notation hexadécimale est très utile en programmation)

D'une manière générale, je vous recommande, même si ce n'est pas évident au début, de travailler avec les formes réseau des ip, ainsi, vous travaillerez avec des variables de 4 octets, les chaines de caractères, c'est cool pour l'affichage, mais c'est tout.

ON RETIENDRA QU'IL FAUT TOUJOURS SE SOUCIER DU FORMAT DU RETOUR ET DES PARAMETRES DES FONCTIONS WINSOCK:
VOUS DEVEZ TOUJOURS SAVOIR COMMENT PASSER DU FORMAT RESEAU AU FORMAT LOCAL ET INVERSEMENT.

III Résolution des noms

La résolution des noms est une partie importante de la programmation réseau : les adresses ip sont très difficile à retenir , aussi nous désignons la plupart du temps un serveur grâce à un nom. Exemple : pour google nous saisissons le nom « google » dans Internet explorer à la place de son adresse ip.
Pour connaître l'adresse ip d'un serveur nous devons faire une requête dns : Domain Name Server .Des serveurs DNS sont disponibles partout, en les contactant avec une adresse sous forme de chaine de carctère, ils retournent l'adresse ip correspondante.

1) Requête DNS

Notre problème ici consiste à retrouver une adresse IP à partir d'un nom de domaine. L'API utilisé est : gethosbyname() :

```
struct hostent FAR * gethostbyname (
const char FAR * name      //nom du domaine dont on cherche l'adresse ip
);
```

Cette fonction prend pour unique paramètre un pointeur char pointant sur le nom de domaine dont on cherche à retirer l'adresse IP.
Gethostbyname() retourne NULL dans le cas d'un échec ou un pointeur vers une structure hostent si succès:

```
struct hostent {
   char FAR *      h_name;          //nom officiel du serveur
   char FAR * FAR * h_aliases;      //tableau terminé par NULL pointant vers les noms alternatifs
   short           h_addrtype;      //le type de l'adresse retournée
   short           h_length;        //la longueur en octets de chaque adresse
   char FAR * FAR * h_addr_list; // liste terminée par NULL des différentes adresses IP. Les adresses sont retournées
                                    //en network byte order
};
```

Ainsi le listing suivant permet de retourner les différentes adresses ip d'un serveur à partir du nom officiel de celui-ci.

code résolution dns

```
#include <winsock.h>
#include <stdio.h>


int main(int argc,char **argv)
{
    char *buffer;
    struct hostent *host;

    buffer=(char*)malloc(256);
    memset(buffer,0,256);
    if(argc!=2)
    {
        puts("Entrez le nom du domaine dont vous voulez connaitre l'ip");
        gets(buffer);
    }
    else
        strcpy(buffer,argv[1]);

    initwinsock();


    host=gethostbyname(buffer);
    if(!host)
        printf("Can't resolve host %s\n",buffer);
    else
    {
        puts(host->h_name);
        while(host->h_addr_list[0] != NULL)
        {
            printf("%s\n",ltoip(* ((u_long *)host->h_addr_list[0])));
            host->h_addr_list++;
        }

    }
```

```
    WSACleanup();
    return 0;
}
```

Ce programme à l'exécution donne :
Entrez le nom du domaine dont vous voulez connaître l'ip
google.com
google.com
216.239.51.100
216.239.35.100

Vous pourrez remarquer que certains noms de domaines sont associés à différentes adresses IP, ceci permet aux serveurs d'accueillir plus de clients.
On obtient de même pour le nom de domaine microsoft.com 5 adresses IP différentes.

207.46.197.102	207.46.230.220
207.46.230.218	207.46.197.100
207.46.230.219	

Note : Nous avons ici créé la fonction display_hostent() que nous réutiliserons dans la section suivante

2) Retirer un nom de domaine à partir d'une adresse IP :

Lors d'un scanning, on peut trouver de nombreuses adresses IP intéressantes… il est parfois utile de connaître le nom de la machine associée à une adresse IP. Pour cela nous pouvons utiliser la fonction gethostbyaddr() :

```
struct hostent FAR * gethostbyaddr (
  const char FAR * addr,  //pointeur vers l'adresse ip du serveur en network byte order
  int len,                //longueur en octets de l'adresse
  int type                //le type de l'adresse
 );
```

De manière analogue à la fonction gethosbyname(), la fonction gethostbyaddr() retourne un pointeur vers une structure hostent
Le listing suivant nous permet de retrouver le nom d'un serveur grace a son adresse ip uniquement (On notera qu'il faut toujours initialiser winsock.dll et la libérer à la fin de chaque programme utilisant les WSA)

Listing de la fonction d'un programme utilisant la fonction gethostbyaddr()

```
#include <stdio.h>
#include <windows.h>

void main(int argc,char **argv)
{
    char *buffer;
    struct hostent *host;
    unsigned long addr_nbo;


    if(argc!=2)
    {
        fprintf(stderr,"usage: %s <ip>\r\n",argv[0]);
        exit(0);
    }

    buffer=(char*)argv[1];
    initwinsock();

    addr_nbo=inet_addr(buffer);
    host=gethostbyaddr((char*)&addr_nbo,NULL,AF_INET);

    if(!host)
        puts("Can't resolve host");
    else
    {
        puts(host->h_name);
        while(host->h_addr_list[0] != NULL)
        {
            printf("%s\n",ltoip(* ((u_long *)host->h_addr_list[0])));
            host->h_addr_list++;
        }
    }

    WSACleanup();
```

```
    exit(0);
}
```

Nous pouvons dès lors créer une fonction de résolution qui nous facilitera notre travail lors des prochaines productions :

```
char *reverse_dns(char *ip)
{
    unsigned long addr_nbo;
    struct hostent *host;

    addr_nbo=inet_addr(ip);
    host=gethostbyaddr(( const char FAR *)&addr_nbo,NULL,AF_INET);
    return host->h_name;
}
```

Cette fonction pourra être utilisée de la manière suivante :

```
printf("le serveur correspondant a l'adresse 216.239.39.101 est: %s\n",reverse_dns("216.239.39.101"));
```

ATTENTION !!
Dans l'exemple ci-dessus, le nom de domaine est pointé par `host->h_name` c'est winsock qui a réservé de la place pour le nom dans une zone spéciale qui n'est pas gérable par l'utilisateur, c'est-à-dire que vous ne savez pas si la zone pointée par le champ h_name sera utilisée par une autre donnée, ainsi il se peut que si vous faites plusieurs résolutions de noms à la suite, vous vous retrouviez avec différents pointeurs sur la même zone mémoire. Je vous conseille donc de faire un appel à strdup pour avoir une chaine dans votre popre espace mémoire suite à chaque appel de gethostbyaddr.

3) Retirer le nom d'un client connecté

Lorsqu'un serveur recoit plusieurs connections, il peut arriver que l'on veuille savoir les noms des différents clients. Pour cela nous avons la fonction getpeername() :

```
int getpeername (
   SOCKET s,                    //socket sur laquelle le client ,dont on veut connaître le nom, est connecté
   struct sockaddr FAR*  name,  //pointeur ves une structure sockaddr vide que la fonction getpeername() remplie
   int FAR*  namelen            //taille en octets réservée pour le pointeur
  );
```

La fonction getpeername met le nom du client connecté à la socket s dans la structure pointée par name. La longueur du nom est donnée en octets par namelen.

La structure sockaddr est déclarée comme suit dans winsock.h :

```
struct sockaddr {
      u_short sa_family;          /* famille de l'adresse */
      char    sa_data[14];        /* up to 14 bytes of direct address */
};
```

à partir de ce nom,il est facile de retrouver l'ip du client grâce à la méthode de résolution de nom expliquée en 1)
Note ; Si l'on veut uniquement l'adresse ip d'un client on pourra utiliser plus simplement le paramètre facultatif de la fonction accept : un pointeur vers une structure sockaddr_in ,dans laquelle est directement mise l'adresse ip

(cette fonction est à ma connaissance, pratiquement jamais utilisée)

4) Retirer le nom de la machine sur laquelle on lance notre programme

Un serveur peut donner le nom de domaine sur lequel il est lancé. Pour effectuer cela il utilise l'API : gethostname()

```
int gethostname (
   char FAR * name,        //pointeur à un emplacement vers de la mémoire libre pour inscrire le nom du domaine
   int namelen             // nombre d'octets disponibles pour le nom
  );
```

Nous pouvons à partir du nom de l'host retirer plus d'informations, comme l'adresse ip de la machine sur laquelle notre programme tourne (pratique pour un serveur) grâce au protocole de résolution de nom, expliqué en 1)

Exemple pour retirer son nom de domaine et son adresse ip

```
char *buffer;
buffer=(char*)malloc(1024) ;
gethostname(buffer,1024);
```

printf(“ Le nom de l’host est : %s \n”,buffer);

Une manière de retirer son ip est de choper le nom de la machine et de faire un appel à gethostbyname.

5) Retirer des informations sur des protocoles ou des serveurs

Pour retirer des information sur un protocole , on doit soit connaître le numéro , soit le nom du protocole
Nous avons deux fonctions à notre disposition : getprotobynumber() et getprotobyname()

```
struct protoent FAR * getprotobyname (
   const char FAR * name
  );

struct protoent FAR * getprotobynumber (
   int number
  );
```

Ces deux fonctions retournent un pointeur vers une structure protoent :

```
struct protoent {
   char FAR *       p_name;          //nom officiel du protocole
   char FAR * FAR *  p_aliases;      //un tableau terminé par NULL des autre noms
   short            p_proto;         //le numéro du protocole en host byte order.
};
```

Nous avons deux fonctions similaires pour retirer des informations sur les serveurs : getserverbyname() et getserverbyport()

```
struct servent FAR * getservbyname (
   const char FAR * name,
   const char FAR * proto
  );

struct servent FAR * getservbyport (
   int port,
   const char FAR* proto
  );
```

Ces deux fonctions retournent un pointeur vers une strucutre servent que voici :

```
struct servent {
   char FAR *   s_name;             //nom official de la machine
   char FAR * FAR *  s_aliases;     //tableau terminé par NULL des autres nom
   short            s_port;         //le n° port sur lequel le service peut être contacté , en  Network Byte Order
   char FAR *    s_proto;           //le nom du protocole à utiliser  quand on contacte le service
};
```

Nous allons créer deux fonctions qui permettent, d’une part de trouver le nom d’un type de serveur grâce à un numéro de port, et d’autre part de trouver le numéro d’un port correspondant à un nom de type de serveur.

```
/*************************\
|serv_by_port()
|permet de retirer le nom d'un
|service qui tourne sur le numéro de
|port passé en paramètre
|
\*************************/
char *serv_by_port(int port)
{
struct servent *serv;

   if((serv=getservbyport((int)htons(port),"tcp")))
   {
       return serv->s_name;
   }

return NULL;
}
```

Nous pouvons établir facilement une liste de correspondance avec une petite boucle for faisant appel à notre fonction :

```
    for(port=0;port<5000;port++)
    {
        if(buffer=serv_by_port(port))
            printf("%d \t %s \n",port,buffer);
    }
```

Nous obtenons ainsi la liste suivante:

7	echo	80	http	170	print-srv	540	uucp
9	discard	88	kerberos	179	bgp	543	klogin
11	systat	101	hostname	194	irc	544	kshell
13	daytime	102	iso-tsap	389	ldap	556	remotefs
17	qotd	107	rtelnet	443	https	636	ldaps
19	chargen	109	pop2	445	microsoft-ds	666	doom
20	ftp-data	110	pop3	464	kpasswd	749	kerberos-adm
21	ftp	111	sunrpc	512	exec	1109	kpop
23	telnet	113	auth	513	login	1433	ms-sql-s
25	smtp	117	uucp-path	514	cmd	1434	ms-sql-m
37	time	119	nntp	515	printer	1512	wins
42	nameserver	135	epmap	520	efs	1524	ingreslock
43	nicname	137	netbios-ns	526	tempo	1723	pptp
53	domain	139	netbios-ssn	530	courier	2053	knetd
70	gopher	143	imap	531	conference		
79	finger	158	pcmail-srv	532	netnews		

Il existe des documents qui recensent beaucoup plus de port que çà.

Exemples

```
#include <stdio.h>
#include <winsock.h>

void main (void)
{
struct protoent *proto;
char *buffer;

    buffer=(char*)malloc(128);
    puts("Entrez le nom du protocole dont vous voulez retirer des informations");
    gets(buffer);

    initwinsock();
    proto=getprotobyname(buffer);

    if(proto)display_proto(proto);


    WSACleanup();
    free(buffer);
}
```

```
G:\Prog\Articles\club_prog\2_dns\getproto>name
Entrez le nom du protocole dont vous voulez retirer des informations
icmp
Nom du protocole: icmp
Aliases du protocole:
         ICMP
Numero du protocole: 1
```

```
#include <stdio.h>
#include <winsock.h>

void display_proto(struct protoent *proto)
{

    printf("Nom du protocole: %s\n",proto->p_name);
```

```
    puts("Aliases du protocole:");

    while(*proto->p_aliases)
    {
        printf("\t %s \n",*proto->p_aliases);
        proto->p_aliases++;
    }

    printf("Numero du protocole: %d\n",proto->p_proto);
}

void main (void)
{
struct protoent *proto;
int num;


    initwinsock();
    for(num=0;num<2000;num++)
    {
        proto=getprotobynumber(num);

        if(proto)
        {
            display_proto(proto);
            puts("_____________________________");
        }
    }

    WSACleanup();
}
```

```
G:\Prog\Articles\club_prog\2_dns\getproto>number
Nom du protocole: ip
Aliases du protocole:
         IP
Numero du protocole: 0
_____________________________
Nom du protocole: icmp
Aliases du protocole:
         ICMP
Numero du protocole: 1
_____________________________
Nom du protocole: ggp
Aliases du protocole:
         GGP
Numero du protocole: 3
_____________________________
Nom du protocole: tcp
Aliases du protocole:
         TCP
Numero du protocole: 6
_____________________________
Nom du protocole: egp
Aliases du protocole:
         EGP
Numero du protocole: 8
_____________________________
Nom du protocole: pup
Aliases du protocole:
         PUP
Numero du protocole: 12
_____________________________
Nom du protocole: udp
Aliases du protocole:
         UDP
Numero du protocole: 17
_____________________________
Nom du protocole: hmp
Aliases du protocole:
         HMP
Numero du protocole: 20
_____________________________
Nom du protocole: xns-idp
Aliases du protocole:
```

```
        XNS-IDP
Numero du protocole: 22
_____________________________
Nom du protocole: rdp
Aliases du protocole:
        RDP
Numero du protocole: 27
_____________________________
Nom du protocole: rvd
Aliases du protocole:
        RVD
Numero du protocole: 66
_____________________________
```

serv_name.c

```
#include <stdio.h>
#include <winsock.h>

void display_servent(struct servent *serv)
{
char ** name;

    printf("Nom officiel de la machine: %s\n",(char*)serv->s_name);
    puts("Aliases de la machine: ");
    name=serv->s_aliases;
    while(*name)
    {
        printf("\t %s \n",*name);
        name++;
    }

    printf("Port utilise: %d\n",ntohs(serv->s_port));
    printf("Nom du protocole: %s\n",serv->s_proto);
}

/*************************\
|serv_by_port()
|permet de retirer le nom d'un
|service qui tourne sur le numéro de
|port passé en paramètre
|
\*************************/
char *serv_by_port(int port)
{
struct servent *serv;

   if((serv=getservbyport((int)htons(port),"tcp")) !=0) //penser à convertir le numéro en N B Order
   {
       return serv->s_name;
   }

return NULL;
}



void main(void)
{
int err;
struct servent *serv;
int port;
char *buffer;


    initwinsock();

    for(port=0;port<5000;port++)
    {
        if((buffer=serv_by_port(port)) !=0)
            printf("%d \t %s \n",port,buffer);
    }
    serv=getservbyport(htons(port),"tcp");
    if(!serv)
    {
```

```
        perror("cant retrieve information");
       // erreur(WSAGetLastError());
    }
    else
        display_servent(serv);


    WSACleanup();
}
```

G:\Prog\Articles\club_prog\2_dns\getserv>name
Nom officiel : smtp
Aliases :
mail
Port utilise: 25
Nom du protocole: tcp

Serv_port.c

```
#include <stdio.h>
#include <winsock.h>


void display_servent(struct servent *serv)
{
char ** name;

    printf("Nom officiel : %s\n",(char*)serv->s_name);
    puts("Aliases : ");
    name=serv->s_aliases;
    while(*name)
    {
        printf("\t %s \n",*name);
        name++;
    }

    printf("Port utilise: %d\n",ntohs(serv->s_port));
    printf("Nom du protocole: %s\n",serv->s_proto);
}

void main(void)
{
int err;
struct servent *serv;
char *buffer="smtp";

    initwinsock();
    serv=getservbyname(buffer,"tcp");
    if(!serv)
    {
        perror("cant retrieve information");
    }
    else
        display_servent(serv);

    WSACleanup();
}
```

IV Gestion des fonctions non bloquantes avec select()

1) Rendre une socket bloquante ou non bloquante

Nous avons jusqu'à présent utilisé des fonctions bloquantes ,c'est à dire que nos fonctions ne retournaient une valeur qu'une fois que leur taches étaient réalisées. Il peut néanmoins arriver que le programmeur désire rendre ces fonctions non bloquantes : en effet quand une fonction bloquante est lancée , si elle n'est pas terminée sur le champ elle met toutes les instructions du programme en attente puisque le programme attend une valeur de retour pour continuer son exécution. Il est possible de rendre une fonction bloquante (ie en non-bloking-mode).Ceci est particulièrement utile pour gérer une connexion avec un timeout. Nous ne rendons pas exactement une fonction non bloquante , mais une socket en « non blocking mode » .Ainsi si on met une socket en non bloking mode, toutes les fonctions utilisant cette socket retournerons immédiatement une valeur.
La fonction qui sert à gérer l'état bloquant d'une socket est ioctlsocket() dont voici le prototype :

```
int ioctlsocket (
   SOCKET s,              //la socket dont l'état va changer
   long cmd,              //la commande que l'on veut exécuter sur la socket
   u_long FAR* argp       //un pointeur vers le paramètre de la commande
  );
```

Pour rendre une socket bloquante ou non bloquante nous devons utiliser la commande : FIONBIO
Les autre commandes sont : FIONREAD et SIOCATMARK pour l'instant je ne sais pas à quoi ces deux dernières commandes servent …

Argp doit pointer vers une variable de type u_long notons val cette variable. Si val vaut 1 on passe la socket en non-bloking mode, si val vaut 0 on passe la socket en blocking mode
Attention , il faudra bien utiliser la fonction ioctlsocket() : il est parfois utile de rendre la socket non bloquante pour réaliser une connection avec un timeout, mais il faut penser à la remettre en mode bloquant pour les opérations de discussions avec recv()
Une fois que notre est socket est mise en non blocking mode, toutes les fonctions retournent immédiatement, la plupart du temps elles retournent SOCKET_ERROR , mais ne n'est pas forcément une mauvaise chose, on pourra regarder l'erreur exacte avec WSAGetlastError()
Nous avons précédemment évoqué la notion de timeout, c'est en effet en gérant un délai que nous allons voir si notre socket est dans l'état espéré.Pour gérer ce timeout(=temps de délai), nous utilisons la fonction select(). Cette fonction est assez complexe, il faut créer des variables spécialement pour cette fonction : des set de socket : des « File Descriptor set ».Voici le prototype de la fonction select() :

2) Utilisation de la fonction select()

```
int select (
   int nfds,                        //Cet argument est ignore il n'est là que pour des histoires de compatibilité
   fd_set FAR * readfds,            //Jeu (SET) de socket qui doivent être lisible
   fd_set FAR * writefds,           //Jeu de socket sur lesquelles on peut réaliser des opérations d'écriture
   fd_set FAR * exceptfds,          //jeu de socket sur lesquelles on a des erreurs
   const struct timeval FAR * timeout        // pointeur vers une structure timeval
  );
```

Voyons ce qu'est un " set de socket " nous présenterons ensuite la structure timeval
FD_CLR(s, *set) enlève le descripteur s (la socket) du jeu de descripteur
FD_ISSET(s, *set) cette fonction retourne true si la socket s fait parti du set ,sinon la fonction retourne 0
FD_SET(s, *set) ajoute la socket s au jeu de socket
FD_ZERO(*set) initialise le jeu de socket à 0: après l'instruction: FD_ZERO (*set) set ne contient plus aucune socket, il faudra le recharger avec des FD_SET(x,*set)

Exemple d'utilisation: nous voulons ajouter la socket sock dans le set f_set:

```
FD_ZERO(f_set);                 //initialisation du set f_set
FD_SET(sock,f_set);             // met notre socket dans le set
….[Opérations sur la socket réalisées en non bloking mode.. Puis appel de select]
..
```

Le cinquième paramètre de la fonction select() est un pointeur vers une structure timeval. Une structure timeval est une structure dans laquelle on met le temps maximum que la fonction doit attendre avant de retourner une valeur .Ce temps est donné à la microseconde près.
Voici la structure timeval:

```
struct timeval {
      long    tv_sec;        /* secondes */
      long    tv_usec;       /* microsecondes (et pas milli !)*/
};
```

Aini pour spécifier un timeout de 3.10 secondes, nous ferons :

```
timeval time ;
time.tv_sec         = 3 ;
time.tv_tv_usec   = 100 000;
```

Attention: il se peut que select() change le contenu de la structure timeout(on lui passe une adresse) dans le cas où on appelle plusieurs fois select(), il faudra à chaque fois reremplir, par précaution la structure timeout.
Le RETOUR DE SELECT() :
Select() retourne SOCKET_ERROR dans le cas d'un échec. Sinon:
Si au bout du temps écoulé, aucune socket n'a changé d'état: readable ou writable select() renvoie 0
Si avant que le temps soit écoulé au moins une socket devient readable ou writable select() renvoie le nombre de socket qui ont changé d'état.

Si nous avons mis plusieurs descripteurs de fichier dans nos jeux de socket, la fonction select() retourne dès qu'au moins une socket change d'état.C'est pour cette raison que la fonction select() retourne généralement 1, elle retournera X (X>1) sil il y a eut X socket qui ont changé d'état en MEME temps.

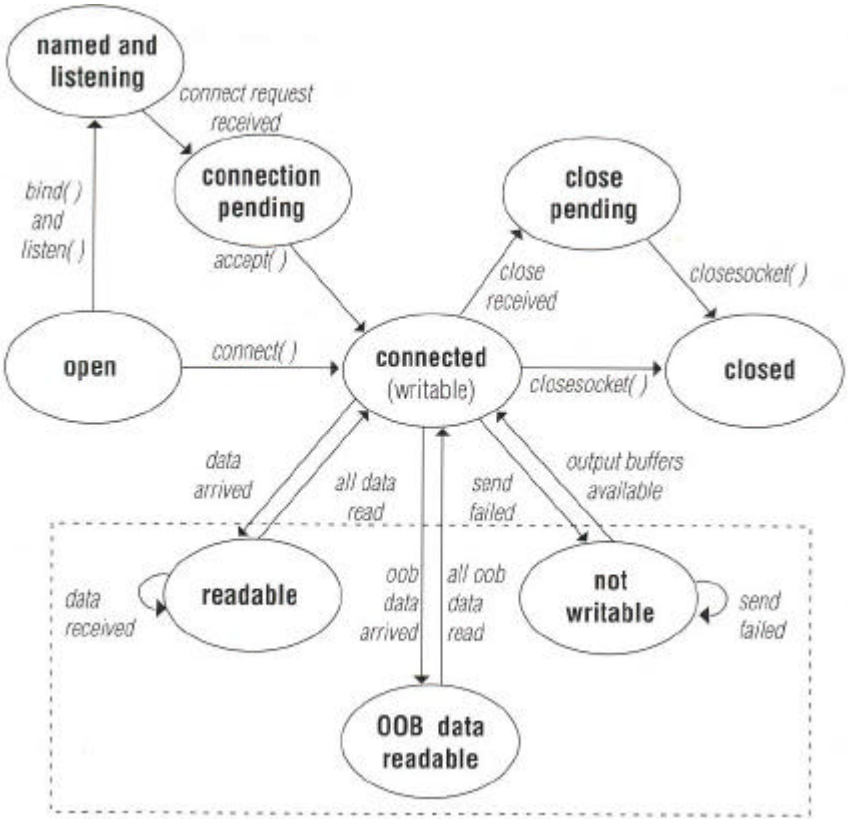

Diagramme indiquant l'état des socket selon les fonctions utililisées.

Image tirée de :Windows Socket Network Programming de Bob Quinn & Dave Shute

Voila à partir de ce que l'on vient de dire nous pouvons construire notre fonction de connexion gérant les timeout:

```
/*************************************\
|                 CONNEXION
| Réalise une connexion tcp sur une machine distante
|
| SOCKET connexion(
|                 ,char *adresse  // ip de l'hote
|                 ,int port          // port de l'hote
|                 ,int time          // timeout en millisecondes
|                 ,int threads);     // nombre de tentatives
|
| retourne NULL si échec ou une socket connectée si succès
|
\*************************************/

SOCKET connexion(char *adresse,int port,int time,int thread)
{
    struct sockaddr_in sin;
    int err;                                  // pr choper les valeurs de retour
    FD_SET rset;                        //on crée un descripteur pour select()
    struct timeval timeout;             // pr le timeout de select
    unsigned long ttr;
    SOCKET sock;


    /*On rempli la struct de destination*/
    sin.sin_port = htons(port);
    sin.sin_family = AF_INET;
    sin.sin_addr.s_addr=inet_addr(adresse);


    //-- Boucle de threads --//
    do
    {

        //-- On converti les milisecondes en microsecondes et--//
        //--  secondes pr remplir la structure timeout  --//
        timeout.tv_sec=time/1000;
        timeout.tv_usec=(time%1000)*1000;

        closesocket(sock);   // on doit à chaque fois refermer le socket pr le recréer
                             // afin de se reconnecter

        if ((sock=socket(AF_INET,SOCK_STREAM,0))==INVALID_SOCKET)
        {
            MessageBox(NULL,"Can't Create the socket","Merde !",MB_OK);
            return NULL;
        }


        FD_ZERO(&rset);      // on initialise le descripteur à 0
        FD_SET(sock,&rset);  // on ajoute notre socket dans la liste du descripteur

        ttr=TRUE;
        err=ioctlsocket(sock,FIONBIO,&ttr);
        if( err== SOCKET_ERROR)        // on rend le socket non blocant
        {
            erreur();
            return NULL;
        }


        err=connect(sock, (struct sockaddr *)&sin,sizeof(struct sockaddr));
        if(err== SOCKET_ERROR)
        {
            err=WSAGetLastError();
            if((err!=WSAEALREADY)&&(err!=WSAEWOULDBLOCK))
            {
                erreur();
                return NULL;
            }
        }

        printf(".");       // affiche la progression
```

```
        /* apelle select pour verifier la lecture apres le timeout */
        err = select(-1,NULL,&rset,NULL,&timeout); // Regarde si le socket est connecté en
                                                  // vérifiant qu'il soit writable
        if(err==SOCKET_ERROR)
        {
            erreur();
            return NULL;

        }
        else if(err!=0) //si il n'y a pas d'erreurs
        {

            ttr=FALSE;
            err=ioctlsocket(sock,FIONBIO,&ttr);  //on le remet en bloking mode
            if (err==SOCKET_ERROR)
            {
                 perror("ioctlsocket back  dans connextion");
                 return NULL;
            }

            return sock;
        }

        thread--;    // décrémente la variable thread

    }while(thread>0); //tant  qu'il reste des threads

    return NULL;

}
```

3) Réception avec timeout.

Si vous avez compris comment marche la fonction connexion, ca va être du gateau. La différence ici vient du fait qu'on appelle select avant d'appeler recv, alors que précédemment on a du d'abord appeler connect puis select. C'est logique : une socket présente différents états, select permet de tester ces états. Une fonction non connectée ne peut pas être lue. Il faut la rendre lisible en la connectant à une autre socket. Une fois que nous savons que nous sommes connecté, on sait que la socket sera de nouveau 'readable' si il y a eut un envoi de données de l'autre coté. On teste donc avec select si la socket est readable, ensuite selon le résultat on fait appel à recv, qui est toujours une fonction bloquante, mais comme on sait qu'il y a des choses à lire, recv va retourner immédiatement.

```
// Like recv, but support timeout in milisec.
int recvEx(SOCKET sock,void * data,unsigned long *len,unsigned int milisec_TimeOut)
{
    long read;
    fd_set  set;
    struct timeval timeout;
    int ret;


    timeout.tv_usec = milisec_TimeOut*1000;//ms = 10^3 us
    timeout.tv_sec = 0;

    FD_ZERO(&set);
    FD_SET(sock, &set);
    ret = select(0,&set,NULL,NULL,&timeout);
    if(!ret || ret == SOCKET_ERROR)
    {
         *len = 0;
         return ret;
    }
    if (FD_ISSET(sock, &set))
    {
        read = recv(sock,data,*len,0);
        if(read == SOCKET_ERROR)
        {
             *len = 0;
            return read;
        }
        else
        {
             *len = read;
            return read;
        }

    }

    *len = 0;
    return SOCKET_ERROR;


}
```

V Présentation des protocoles les plus connus

Nous avons vu jusqu'à présent comment se réalisaient les opérations primaires de discussion entre clients et serveurs à travers le réseau.
Il existe de nombreux protocoles standardisés, c'est à dire qu'il existe pour différents types de serveurs un mode de discussion bien défini. Nous allons, dans cette partie présenter plusieurs protocoles qui sont omniprésents sur le réseau, nous verrons comment faire des applications s'adaptant à ces protocoles.
(Pour voir d'autres informations sur les protocoles allez jeter un coup d'œil sur http://www.commentcamarche.net/internet/smtp.php3)

1) Le Simple Mail Transfert Protocol (SMTP)

Ce protocole est utilisé pour envoyer des mails: quand je veux envoyer un mail à mon ami, j'édite un mail sur un logiciel de messagerie, puis ce logiciel se connecte à un serveur SMTP et lui envoie des instuctions spécifiques quand je commande l'envoi du mail.
Le Simple Mail Transfert Protocol est un protocole qui permet uniquement d'envoyer des messages mails. Pour retirer des mails de sa boîte aux lettres, on utilise le protocole POP3, qui sera présenté dans la section suivante.
Lors de la discussion entre client / serveur Mail, il faut respecter un ordre bien préci pour arriver à envoyer un message.
Les lignes suivantes montrent comment se réalise un envoi de message:
<Le Serveur Mail est lancé sur une machine distante>
<Un client se connecte sur le serveur Mail. Le port officiel du SMTP est le port numéro 25 >
Le serveur envoie au client un message de bienvenue: 220 Welcome on the Phoneusis.Smtp.com server
Le client envoie alors un message qui va lui permettre de commencer une cession sur le server. Pour cela , il utilise la commande HELO qu'il complète avec le nom du server sur lequel il est connecté:
HELO phoneusis.smtp.com
Important : Je rappelle que chaque commande envoyée par le client doit être terminée par la séquence <Cariage Return> < Line Feed> c'est à dire par : "\r\n"(La séquence \r\n est automatiquement envoyée par telnet quand on frappe ENTER)
Le serveur , répond alors qu'il est prêt : il commence une cession pour nous et attends d'autres commandes de la part du client.
250 OK proceed
Le client commence par envoyer l'e-mail de l'expéditeur: il le joint à la commande MAIL FROM:
MAIL FROM: phoneus@wanadoo.com
Attention: certains serveurs mail n'acceptent que les e-mails ayant le même nom de domaine qu'eux.Par exemple: smtp.wanadoo.fr ne peut accepter que les expéditeurs dont les emails sont du type fucker@wanadoo.fr. Ceci est vrai aussi bien pour l'email de l'auteur que pour celui de destination.
Le serveur répond au client qu'il accepte l'email du messager et attend les instructions complémentaires.
Le client continue en envoyant ,via la commande RCPT TO: , l'adresse du destinataire.
RCPT TO: mon_pote@wanadoo.com
Il se peut que le serveur refuse de transmettre le mail pour les raisons que je viens d'évoquer précédemment.dans ce cas il répondra :" Relaying denied" et vous devrez soit changer les e-mails, ce qui n'est pas toujours faisable , soit changer de serveur…hu hu..
Toutefois , si le serveur accepte les adresses , il vous répondra:
250 OK
A ce stade, il ne reste plus qu'à envoyer le contenu du mail. Pour faire comprendre au serveur qu'on est sur le point de balancer notre texte, on lui envoi la commande DATA
Le server doit alors renvoyer une phrase du type : 354 go ahead
Envoyez votre message, il doit être fini par la séquence <CR LF>.<CR LF>
(ce qui veut dire que le serveur va mettre dans le message tout ce que vous allez lui envoyer jusqu'au moment où vous lui enverrez : "\r\n.\r\n"
bla bla bla bla
ça va bien ?
cool et toi ? bla bla bla
bla bla je pense souvent à toi… bla bla bla
à bientôt mon meilleur pote bla bla
.

A la fin du message, le serveur nous envoie une note qui témoigne du succès de l'opération.
A ce moment le client peut quitter le serveur quand il veut en envoyant la commande QUIT
Note: si le client veut envoyer plusieurs mails d'un coup, il n'a pas besoin de se déconnecter, puis de se reconnecter: il retourne à l'étape MAIL FROM: à la place d'envoyer QUIT

Format du message

Un message est composé de 2 éléments : l'en-tête et le corps.

Description des principaux champs de l'en-tête :
Après avoir envoyé la commande data au serveur et une fois sa réponse recue, nous envoyons :
To : adresse(s) du ou des destinataire(s) (séparées par une virgule)
Cc : adresse(s) en copie
From : adresse de l'auteur du message
Reply-To : adresse(s) de réponse
Date : date et heure de l'expédition du message
Subject : sujet du message
Received : trace ajoutée à chaque étape du routage du message
Return-Path : indique l'adresse de retour à l'expéditeur
Message-Id : identification unique du message
Et maintenant le teste que l'on veut envyer sans mettre de mot clef spécifique tel que « body : »
.

Je remets le dialogue entre le client et le serveur, sans les commentaires:
Le texte en bleu correspond au texte envoyé par le client , celui en vert corespond à la réponse du serveur.
220 smtp.noos.fr ESMTP
MAIL FROM: phoneus@noos.fr
250 ok
RCPT TO: phoneus@noos.fr
250 ok
DATA
354 go ahead
subject : Coucou mon ami
to : Caroline
from : Bernard
Message ID: 321321645.ti.com
Salut Caro la clef privée Pgp de Cécile est la suivante : é »4'6(46 »é'46(6 »é4cv 46' »é4(6g4é64c4f(' »-'è(è-_654'-64b6('4z64r6é »v(
En espérant que tu en feras bon usage
.
250 ok
QUIT
221 smtp.noos.fr

Note: chaque réponse du serveur commence par une série de trois chiffres.ces trois chiffres permettent d'automatiser une discussion: si le client ne recoit pas la bonne séquence de chiffre, il y a un problème.Les chiffres sont utilisés lors d'une discussion avec des serveurs SMTP et FTP

Le principe est relativement simple à comprendre. Codons une fonction qui se charge d'envoyer des mails . send_mail() envoie un mail et spécifie différents champs tels que :
le sujet,
le Message Id qui est obligatoire sur certains serveurs
Le nom ordinaire de l'expéditeur (en plus de son e mail)
Le nom ordinaire du destinataire

Send_mail() fait appel à une fonction de winsock.dll qui n'a pas été présentée, il s'agit de TransmitFile, qui permet d'envoyer un fichier entier sur une socket, TransmitFile est déclarée dans le header mswsock.h.
J'ai créé cette fonction pour créer une fonction send_mail à titre d'exemple, je n'ai pas travaillé dessus très en détail.

```
#include <mswsock.h>  // pour TransmitFile()
 /*************************************************\
|               SEND MAIL
```

```
|Envoi un mail
| int send_mail(char *smtp,  //adresse d'un serveur SMTP
|               char *from,  //l'adresse mail de l'émetteur
|               char *rcpt,  //l'adresse e mail du recepteur
|               char *mess,  //un pointeur vers le texte a envoyer
|               char *obj,   //l'objet du message
|               char *nom1,  //nom civil de l'expéditeur
|               char *nom2,  //nom civil du destinataire
|               short n_mail);  //nombre de mails à envoyer
|
| Retourne: NULL si le mail n'as pas pu etre envoyé sinon 1
\************************************************/

int send_mail(char *smtp,char *from,char *rcpt,char *file_message,char *obj,char *nom1,char *nom2,short n_mail)
{
int err;
char *buffer,*answer;
SOCKET sock;
short i,etap=0;
static char *ip;
HANDLE h_file;



 // avant toute chose , on vérifie que le fichier que l'on va envoyer soit bien accesible
 h_file=CreateFile(file_message,GENERIC_READ     ,FILE_SHARE_READ,NULL,OPEN_EXISTING,FILE_ATTRIBUTE_NORMAL,NULL);
 if(!h_file)
 {
    fprintf(stderr,"error, CreateFile()");
    return 0;
 }

 ip=dns(smtp);                                          //on chope l'ip du SMTP

 buffer=(char*)malloc(1030);
 memset(buffer,0,1024);
 answer=(char*)malloc(1030);



 sock=connexion(ip,25,3000,4);              // et on se connecte sur le port 25 avec 4 threads de 3 sec
 if(err==SOCKET_ERROR) {printf("\nImpossible de réaliser la connection sur %s port 25\n",buffer);return 0;}
 memset(answer,0,1024);
 err=recv(sock,answer,1024,0);
 if(err==SOCKET_ERROR) perror("Erreur de reception");

 sprintf(buffer,"HELO %s\r\n",smtp);      //tout d'abord nous saluons notre hote
 err=send(sock,buffer,strlen(buffer),0);
 if(err==SOCKET_ERROR)perror("SEND ()");
 memset(answer,0,1024);
 err=recv(sock,answer,1024,0);
 if(err==SOCKET_ERROR) perror("Erreur de reception");
 if(traitement(answer,&err)!=SERVER_OK)
 {
    return send_mail_error(buffer,answer,etap);
 }


 for(i=0;i<n_mail;i++)
 {
    etap=1;
    printf("\nMail %d/%d ",i+1,n_mail);

    sprintf(buffer,"MAIL FROM: %s\r\n",from);   //ensuite on indique notre adresse
    send(sock,buffer,strlen(buffer),0);
    memset(answer,0,1024);
    err=recv(sock,answer,1024,0);
    if(err==SOCKET_ERROR) perror("Erreur de reception");
    if(traitement(answer,&err)!=SERVER_OK)
    {
       return send_mail_error(buffer,answer,etap);
    }
   etap++;   //2
    fputc('.',stdout);

    sprintf(buffer,"RCPT TO: %s\r\n",rcpt);   //puis on donne l'adresse de destination
```

```
    send(sock,buffer,strlen(buffer),0);
    memset(answer,0,1024);
    err=recv(sock,answer,1024,0);
    if(err==SOCKET_ERROR) perror("Erreur de reception");
    if(traitement(answer,&err)!=SERVER_OK)
    {
       return send_mail_error(buffer,answer,etap);
    }
   etap++;   //3

    sprintf(buffer,"DATA\r\n");   //on l'informe que nous sommes sur le point de lui donner le contenu du mail
    send(sock,buffer,strlen(buffer),0);
    memset(answer,0,1024);
    err=recv(sock,answer,1024,0);
    if(err==SOCKET_ERROR) perror("Erreur de reception");
    if(traitement(answer,&err)!=START_MAIL)
    {
       return send_mail_error(buffer,answer,etap);
    }
   etap ++;  //4

    // a partir de ce moment là, on ne recoit plus rien de la part du serveur tant que l'on n'envoie pas "\r\n.\r\n"

 //-- En tête du mail!!! on a toujours tendance à oublier l'entête!! c'est important !! -//

    sprintf(buffer,"From: %s\r\n",nom1);  //Le nom que verra le destinataire Ex"De La Part de ton pire ennemi"
    send(sock,buffer,strlen(buffer),0);

    sprintf(buffer,"To: %s\r\n",nom2);      // le nom du destinataire Ex: "Gros connar"
    send(sock,buffer,strlen(buffer),0);

    sprintf(buffer,"Subject: %s\r\n",obj);    // L'objet du mail
    send(sock,buffer,strlen(buffer),0);

    sprintf(buffer,"Date: Fri,30 Dec 1980 00:00:00 -0000\r\n");
    send(sock,buffer,strlen(buffer),0);



    sprintf(buffer,"X-Mailer: Internet Mail Service\r\n");
    send(sock,buffer,strlen(buffer),0);


    sprintf(buffer,"X-Priority: 3\r\nIMPORTANCE: Normal\r\nMessage-Id: <%d%d.mailed_tracker>\r\n",GetTickCount(),GetTickCount()/);
    send(sock,buffer,strlen(buffer),0);

    sprintf(buffer,"Content-Type: text/plain; charset=\"iso-8859-1\"\r\nContent-Transfer-Encoding: quoted-printable\r\n\r\n\r\n");
    send(sock,buffer,strlen(buffer),0);

    fputc('.',stdout);

 //-- fin de l'entête, nous balancons maintenant le message -//

    if(!TransmitFile(sock,h_file,0,2056,NULL,NULL,TF_WRITE_BEHIND))
    {
       fprintf(stderr,"erreur de la fonction TransmitFile");
       return 0;
    }


    sprintf(buffer,"\r\n.\r\n");                              //et on lui dit que notre lettre est terminée
    send(sock,buffer,strlen(buffer),0);

    memset(answer,0,1024);
    err=recv(sock,answer,1024,0);
    if(err==SOCKET_ERROR) perror("Erreur de reception");
    if(traitement(answer,&err)!=SERVER_OK)
    {
       return send_mail_error(buffer,answer,etap);
    }
    etap ++;
    fputc('.',stdout);

 }              //fin de la boucle de Bourrinage ;}

 sprintf(buffer,"QUIT\r\n");                                  //On s'ARRACHE ! !! !
```

```
    send(sock,buffer,strlen(buffer),0);
    memset(answer,0,1024);
    err=recv(sock,answer,1024,0);
    if(err==SOCKET_ERROR) perror("Erreur de reception");
    if(traitement(answer,&err)!=GOOD_BYE)
    {
       return send_mail_error(buffer,answer,etap);
    }

    CloseHandle(h_file);
    free(buffer);
    closesocket(sock);
    return 1;
}
```

Il est maintenant facile théoriquement d'envoyer des mails à tout le monde sans avoir besoin d'utiliser le fameux Outlook (pour ceux qui ne connaissent pas, essayer FoxMail : ca évite d'être victime des éventuels exploits). Malheuresement, la plupart des serveurs SMTP n'acceptent pas toutes les adresse mails. Une bonne chose est d'utiliser le serveur mail de son fournisseur d'accès avec le pseudo correspondant. (tant que nous ne voulons pas faire de mails anonymes)

2)Le Post Office Protocole(POP3)

Le protocole POP3 est le protocole utilisé pour retirer les messages qui sont arrivés dans une boite aux lettres.
Les serveurs POP3 sont tous lancés sur le port 110.Comme pour le SMTP, je vais vous présenter un dialogue entre client et serveur:
Le client se connecte au serveur.
Le serveur envoie d'emblée un message: pop3 server ready
Puis le client s'identifie: il balance son pseudo avec la commande USER, et à l'invite du serveur il rentre son mot de passe: PASS xxxxx
L'identification avec le serveur POP3 se fait de la même manière qu'avec un serveur FTP (voir section suivante).Il faut faire attention à respecter la casse lors de la saisie des commandes
Dans le cas d'une mauvaise identification, le serveur nous envoie chier , sinon , il nous dit:
Mailbox open, X messages (ou X représente le nombre de messages présents dans la boîte aux lettres)

Pour avoir une liste des messages, le client doit envoyer la commande LIST
Le serveur renvoie alors une liste des différents messages avec un numéro correspondant à chaque message.
Pour lire un message, le client envoie la commande RETR <Numéro du message> (retr comme retreive)
Le serveur envoie alors une magnifique entêtre pleine d'informations interéssantes, puis le corps du message
Voilà voilà tout.
Je dirait juste une autre chose pour finir: quand vous êtes connecté sur votre compte, personne d'autre peut se connecter en même temps sur votre compte: quand vous vous connectez, un fichier lock est créé. Ce fichier est effacé quand vous vous déconnectez

Les serveurs POP3 ne commencent pas leurs réponses par une séquence de 3 chiffres comme le font les serveurs SMTP et FTP : ils répondent en envoyant au début de chaque envoi soit par : +OK pour indiquer le succès de la commande envoyée ou –ERR pour signaler une erreur

La commande DELE <numéro du message> efface le messsage correspondant

```
+OK Hello there.
USER phoneus
+OK Password required.
PASS abcdfuckefgh
+OK logged in.
list
+OK POP3 clients that break here, they violate STD53.
1 420
.
rete
-ERR Invalid command.
retr 1
+OK 420 octets follow.
Return-Path: <phoneus@noos.fr>
```

```
Delivered-To: phoneus@noos.fr
Received: (qmail 46481204 invoked by uid 0); 21 Nov 2002 15:08:40 -0000
Received: from unknown ([212.198.47.61]) (envelope-sender <phoneus@noos.
        by 212.198.2.75 (qmail-ldap-1.03) with SMTP
        for <phoneus@noos.fr>; 21 Nov 2002 15:08:40 -0000
DATA
subject: Fuck
messag
message id : 3213131321
exit
cls
close
quit$
quit
.
list
+OK POP3 clients that break here, they violate STD53.
1 420
.
retr 5
-ERR Invalid message number.
retr 3
-ERR Invalid message number.
retr 2
-ERR Invalid message number.
retr 1
+OK 420 octets follow.
Return-Path: <phoneus@noos.fr>
Delivered-To: phoneus@noos.fr
Received: (qmail 46481204 invoked by uid 0); 21 Nov 2002 15:08:40 -0000
Received: from unknown ([212.198.47.61]) (envelope-sender <phoneus@noos.
        by 212.198.2.75 (qmail-ldap-1.03) with SMTP
        for <phoneus@esiee.fr>; 21 Nov 2002 15:08:40 -0000
DATA
subject: Dates de Rave parties
messag
message id : 3213131321
exit
cls
close
quit$
quit
.
```

3) Le File Transfert Protocol (FTP)

Après avoir vu comment les emails étaient transmis sur Internet, intéressons nous à un protocole plus ancien, le File Transfert Protocole. C'est à mes yeux de loin le protocole le plus important à comprendre. Ce protocole permet de transférer des fichiers. A la naissance d'Internet, on n'utilisait que le FTP. C'est seulement une fois qu'on été créées les

pages web interactives que le FTP a laissé place au http. Ceci dit, le FTP a toujours aujourd'hui une place très importante. C'est par ce protocole que la plupart des sites web sont installés et entretenus et que les plus gros fichiers sont téléchargés .

a) Liste des fonctions du FTP

Le FTP permet une gestion avancée des fichiers; on peut naviguer à travers les répertoires présents sur le serveur. Voici une liste des différentes fonctions supportées par la majorité des serveurs FTP.

LES COMMANDES FTP			
COMMANDE		FONCTION	EXEMPLE D'UTILISATION
Identification	USER	Commence une identification aaa est le login du compte	USER phoneus
	PASS	Termine l'identification xxx est le mot de passe correspondant au login qui vient d'être donné	PASS 012345
	REIN	Reste connecté au serveur , mais retourne à l'étape de l'identification: permet de changer de compte rapidement	REIN
Gestion des répertoires	RMD	Efface un répertoire (**R**e**M**ove **D**irectory)	RMD old_files
	MKD	Crée un répertoire dont le nom est nom_rep (**M**a**K**e **D**irectory)	MKD new_files
	XMKD	Idem que MKD	XMKDR rep
	CWD	Change le répertoire courant(**C**hange **W**orking **D**irectory) nous sommes désormais dans le répertoire nom_dir	CWD .\archive
	CDUP	CDUP Passe au répertoire parent	CDUP
Configuration Du Transfert	PORT	Le client informe le serveur qu'il a ouvert un port pour le transfert	PORT 127,0,0,1,23,12
	PASV	Demande au serveur d'ouvrir un port pour le canal de transfert	PASV
	TYPE X	Change le mode de transfert: A pour le transfert ASCII ou I pour un transfert en binaire	TYPE A
Gestion des fichiers	RETR	Demande au serveur de nous envoyer le fichier file_name	RETR info.txt
	STOR	Envoi d'un fichier du client vers le serveur	STOR upgrade.pak
	REST	Commence un téléchargement à une certaine position	
	APPE	Comme STOR sauf que les données sont mises à la suite	
	ABOR	Annule le transfert	ABOR
Environnement	STAT	Le serveur affiche le statut de client connecté: il indique le nombre de fichiers uploadés et downloadés, ainsi que leur équivalence en Ko	STAT
	SYST	Retire le nom du système d'exploitation sur lequel est lancé le serveur	SYST
	NOOP	Ne fait aucune opération.(**NO OP**eration) Cette commande permet de s'assurer que nous sommes toujours connecté au serveur. Le serveur répond: 200 command ok	NOOP
	HELP	Le serveur affiche les commandes qu'il reconnaît	

Après avoir envoyé une requête sur le serveur, le client doit attendre une réponse de la part du serveur avant d'envoyer une autre requête.

b) Créer une fonction pour s'identifier

Avant de réaliser des opérations sur les fichiers ,et une fois que nous sommes connectés sur le serveur FTP, nous devons, comme pour le POP3 nous identifier.Pour cela, on passe d'abord un login , puis un mot de passe.Le dialogue du début d'une cession ressemble au suivant:
<Connection du client >
Le serveur envoie un message de bienvenue au client et l'invite à s'identifier:
Le client commence par donner son nom d'utilisateur (login): il envoie au serveur
USER phoneus
Le serveur enregistre le nom d'utilisateur et demande le mot de passe correspondant
Le client envoie alors :
PASS aaaaaa
Si le nom de l'utilisateur et le mot de passe sont corrects , le serveur informe le client qu'il peut gérer les fichiers.
Si le mot de passe OU le nom de l'utilisateur sont mauvais, le serveur renvoi INVALID PASSWORD, et le client devra recommencer l'identification
Il est facile de créer une fonction qui se chargera de l'identification sur un FTP:

```
/**************************
| int recv_line(sock,buffer)
|
|réception d'une seule ligne terminée par \n
| retourne -1 si échec dans la reception
| si non le nombre de caractères récupérés dans le buffer
\**************************/
```

```
int recv_line(SOCKET sock,char *buffer,int len)
{
char recup;
int i=0;
int err;

   while((recup!='\n')&&(i<len))
```

```
    {
      err=recv(sock,(char*)&recup,1,0);
      if(err==SOCKET_ERROR)
      {
        if(WSAGetLastError()==WSAECONNABORTED)
          return NEED_TO_RECONNECT;
        else return err;
      }
      else if(err==NULL)
        return -1;
      buffer[i]=recup;
      i++;
    }

    if(i>0) i--;   //pour se mettre à l'index de '\n'

    if(buffer[i-1]=='\r')
    {
      i--;
      buffer[i]=0;
    }

    else
      buffer[i]=0;   //écrase le saut de ligne avec la fin de la chaine
    return i;
}



/*****************************\
|  remove_back()
| enlève les backspace d'un buffer
| pour les serveurs qui acceptent des clients telnet
|
| ceci évite d'imposer aux utilisateurs de ne pas se tromper dans une saisie et les
| autorise à utiliser la touche BACKSPACE
| (retour caractère ='\b')
|
\*****************************/
void remove_back_space(char *buffer,int len)
{

int index,back;
int decal_dest=0, decal_src=0;

    for(index=0;index+decal_src<len;index++)
    {
      back=0;
      while(buffer[index+decal_src+back]=='\b')
      {
        back++;
      }

      decal_src+=back;// on décale vers la droite de la source du nombre de backspace

      while(back>0)  //on retourne en arrière pour écraser les caractères de la destination
      {
        decal_dest--;
        back--;
        if(index+decal_dest==0)   //pour rester dans la zone de mémoire de la chaine(decal_dest est négatif)
          break;
      }

      buffer[index+decal_dest]=buffer[index+decal_src];
    }

    buffer[index+decal_dest]=0;
}



 /*****************************\
|  send_user()
|envoie sur la socket le login passé en paramètre
| La commmande est construite toute seule
\*****************************/
int send_user(SOCKET sock,char *user)
{
    char *buffer;
    int err;

    buffer=(char*)malloc(512);
    ZeroMemory(buffer,512);

    sprintf(buffer,"USER %s\r\n",user);     // on construit la commande
    err=send(sock,buffer,strlen(buffer),0); // et on l'envoi

    if(err==SOCKET_ERROR)
    {
      perror("send_user()");
      free(buffer);
      getchar();
      return SOCKET_ERROR;
    }

    free(buffer);
    return 1;
}



 /*****************************\
|  send_pass()
|envoie sur la socket mot de passe passé en paramètre
| La commmande est construite toute seule
\*****************************/
int send_pass(SOCKET sock,char *pass)
{
    char *buffer;
    int err;

    buffer=(char*)malloc(512);
    ZeroMemory(buffer,512);

    sprintf(buffer,"PASS %s\r\n",pass);       //on construit la commande
    err=send(sock,buffer,strlen(buffer),0);   // on envoie la commande au serveur

    free(buffer);
    if(err==SOCKET_ERROR)
    {
      perror("send pass" );
      return SOCKET_ERROR;
    }

    return 1;
}



 /*****************************************\
|                LOGING
|
| Pour s'identifier au ftp
| Utilise send_user() et send_pass()
```

```
| retourne 1 si succés, 0 sinon,ou 2 si on est viré
\***************************************/

int ftp_login(SOCKET *sock, char *user, char *pass)
{
   int err;
   char *buffer;

   buffer=(char*)malloc(512);
   ZeroMemory(buffer,512);

   send_user(*sock, user);
   recv_line(*sock,buffer);   //on recoit la réponse du serveur

   err=traitement(buffer);        // on identifie la réponse du serveur
   if(err==USER_OK_PASS)          // si on est invité a saisir le pass on envoi le pass
   {
      send_pass(*sock,pass);
      recv_line(*sock,buffer);    // on recoit la réponse du serveur

      err=traitement(buffer);     // on regarde ce que le serveur nous dit
      if(err==USER_LOGGED_IN)     // si c'est gagné  ! !
      {
         free(buffer);
         return 1;
      }

      else if(err==NO_LOGGED_IN)   //si le pass n'est pas accepté
      {
         free(buffer);
         return 0;
      }

   }


   // si on arrive là c'est que l'on ne s'est pas connecté
   free(buffer);
   return NEED_TO_RECONNECT;   // on est déconnecté
}
```

c) Ouverture de ports supplémentaires le serveur

Pour transmettre des fichiers, tout comme pour retirer le listing d'un repertoire, le FTP crée de nouveaux ports :
A tout moment le client peut choisir d'ouvrir un port sur sa machine et indiquer au serveur qu'il peut se connecter sur ce port. Ou bien, le client peut demander au serveur d'ouvrir un nouveau port.
Quand le client veut échanger des données avec le serveur, il a donc le choix, pour transférer un fichier, entre ouvrir un port ou demander au serveur d'ouvrir un port.
En d'autres termes, lors d'une cession FTP, nous avons deux canaux:

- Un canal de contrôle, à travers lequel on envoie les commandes, comme pour l'identification ou pour la création de dossiers.
- Un canal de données, où circulent les contenus des fichiers qui sont en train d'être transférés.

Quand le client veut envoyer des données au serveur, ou en recevoir, il lui demande d'ouvrir un port. Cette requête est réalisée grâce à la commande PASV:
Le client envoie PASV
Le serveur répond: 227 Entering Passive Mode (127,0,0,1,4,19)
Là les chiffres entre les parenthèse vont nous être bien utiles: les quatre premiers chiffres constituent l'adresse IP du serveur à connecter pour transférer les données. Généralement ils désignent la même adresse que celle sur laquelle est lancé le serveur. Les deux derniers chiffres servent à désigner le port ouvert par le serveur. Pour obtenir le numéro de port mis en service, il faut multiplier l'avant dernier chiffre par 256 et additionner au produit le dernier chiffre.
Le serveur a donc ouvert le port numéro: 4*256 +19 =1043

Send_pasv() est chargée d'envoyer une requête PASV au serveur. Une fois que la requête est envoyée elle attend la réponse du serveur et copie sur ip l'adresse à laquelle il va falloir se connecter et met à l'adresse port le port ouvert par le serveur. Cette fonction pourra être réutilisée pour retirer une liste des fichiers et pour le transfert de fichiers

```
/**********************************
|  send_pasv()
| cette fonction demande au serveur ftp de nous
| ouvrir un port pour le canal de transfert
| le serveur renvoi une IP et un numéro de port
| dont les valeurs sont mises dans les pointeurs ip et port
|
\**********************************/
int send_pasv(SOCKET sock,char *ip,int *port)
{
   char *buffer,*litle;
   int a,b,c,d;   //Les différents octets de l'adresse IP
   int part1,part2;                // les deux nombres qui donnent le numéro de port
   int index=0;

   buffer=(char*)malloc(512);
   ZeroMemory(buffer,512);

   send(sock,"PASV\r\n",6,0);
   receive(sock,buffer,512);
   printf("Server: %s\n",buffer);
   if(traitement(buffer)!=ENTERING_PASSING_MODE) return 0;

   while(buffer[index]!='(')
      index++;

   index++;   //pour virer la première parenthèse
   litle=(char*)&buffer[index];
    //ici litle est de la forme "a,b,c,d,part1,part2) "
   index=0;
   while(litle[index])
   {
      index++;
      if(litle[index]==',')
         litle[index]=' ';   //on remplace les virgules par des espaces

      if(litle[index]==')')
         litle[index]=0;     //on écrase la dernière parenthèse en mettant le symbol de fin de chaine
   }

   sscanf(litle,"%d %d %d %d %d %d",&a,&b,&c,&d,&part1,&part2);

   sprintf(ip,"%d.%d.%d.%d",a,b,c,d);

   *port=part1*256+part2;
   free(buffer);
   return 1;

}
```

Faisons une routine qui permet à un serveur de répondre à la requête PASV:

```
   else if(!strncmp(buffer,"PASV",4)) //buffer est la commande que vient d'envoyer le client
   {
      pasv_sock=manage_port((int*)&prt);  //manage_port() est décrite juste après
      listen(pasv_sock,1);
      strcpy(argue,local_ip());
      i=0;
      while(argue[i])
      {
         if(argue[i]=='.') argue[i]=',';  //remplace les . de l'ip par des virgules
         i++;
      }
      sprintf(buffer,"227 Entering passing mode (%s,%d,%d)\r\n",argue,prt/256,prt%256);
      SENDE;
      return 1;
   }
```

```
SOCKET open_port(int port)
{
struct sockaddr_in sin;
SOCKET sock;

   if((sock=socket(AF_INET,SOCK_STREAM,0))==SOCKET_ERROR) return -1;
   sin.sin_port=htons(port);
   sin.sin_addr.s_addr=0;
   sin.sin_family=AF_INET;
   if(bind(sock,(const struct sockaddr FAR*)&sin,sizeof(SOCKADDR_IN))==SOCKET_ERROR) return -1;

   return sock;
}
SOCKET manage_port(int *pprt)
{
static int port=500;
SOCKET sock;

   while((sock=open_port(port))==-1)
      port++;

   *pprt=port;
   return sock;
}
```

d) Retirer une liste des fichiers da ns un répertoire du serveur

Une fois que le client s'est identifié, il va toujours chercher à avoir une idée de l'arborescence des fichiers sur le serveur.Pour cela, il va demander au serveur de lui envoyer une liste des différents fichiers présents dans le répertoire courant.
La liste est obtenue en envoyant la commande LIST au serveur. La réception de la liste des fichiers n'est pas aussi simple que l'on ne pourrai le penser: le serveur n'envoie pas la liste dans sa réponse au client: il l'envoie par le canal de données. Il faudra donc, pour recevoir la liste des fichiers, que:
Le client envoie la commande PASV, puis que le client se connecte sur le port annoncé par le serveur .Une fois que le canal de transfert est établi, le client peut envoyer sa commande LIST: le Serveur lui envoie via le canal de transfert les noms des différents fichiers, ainsi que leurs attributs , et leur date de création.

Pour retirer une liste je peux faire donc :
Je me connecte et je m'identifie sur un serveur FTP avec telnet sur le port 21
J'envoie au serveur la commande PASV pour que celui ci ouvre un canal de transmission; le serveur me répond en me donnant un numéro de port qu'il a ouvert.
Je lance un autre telnet et je me connecte à l'adresse du serveur sur le port qu'il vient de me donner.
Quand mon deuxième telnet est aussi connecté, je lance LIST à partir du premier telnet. Et là , miracle le deuxième telnet reçoit le contenu du répertoire racine. La connexion avec le deuxième telnet est perdue aussi tôt: il faudra un autre PASV pour se connecter à un nouveau port.

Bon maintenant que nous avons compris comment retirer la liste des fichiers , ça serait bien de se faire une petite fonction qui se chargera de nous afficher gentiment la liste.
La difficulté réside dans le fa it qu' il faut isoler le numéro du port de la réponse du serveur et qu'il faut se créer une deuxiuème socket.

La première fonction : disp_list() est celle qui doit être appelée par le programme client. Elle se charge d'afficher sur la console le contenu du dossier courant. La liste est mise dans un buffer, ce qui nous permet d'utilliser quand on veut cette liste dans sa globalité.
Cette fonction fait appel à la fonction send_pasv() dont le listing est exposé juste après.

```
/*************************\
|disp_list()
| affiche la liste des fichiers
| dans le répertoir courant sur un serveur FTP
|disp_list(SOCKET sock) socket connectée au ftp
|
\*************************/
SOCKET disp_list(SOCKET sock)
{
char *buffer;
int *port;
SOCKET sock_pasv;
int err;

   buffer=(char*)malloc(1024);
   ZeroMemory(buffer,1024);
```

```
    port=(int*)malloc(sizeof(int));

    if(!send_pasv(sock,buffer,port)) return sock;
    printf("Le serveur dont l'ip est %s a ouvert le port
numéro:%d\n",buffer,*port);

    connection((SOCKET*)&sock_pasv,buffer,*port,500,3);

    strcpy(buffer,"LIST \r\n");
    if(send(sock,buffer,strlen(buffer),0)==SOCKET_ERROR) erreur();
    recv_line(sock,buffer);
    puts(buffer);
```

```
    recv_ftp(sock_pasv,buffer,1023);
    closesocket(sock_pasv);
    printf("La liste des fichiers est : \n%s\n",buffer);

    receive(sock,buffer,512);  //il doit nous envoyer transfere complete
    puts(buffer);

    free(buffer);
    free(port);
    return sock;
}
```

```
/*****************************
|recv_ftp(SOCKET sock,char *buffer,int taille);
|recoit la liste d'un repertoir
|attend que le serveur ferme sa socket pour retourner
\*****************************/
recv_ftp( SOCKET sock,char *buffer,int len)
{
char *recup;
short len_r=1;
short total_len=0;

    recup=(char*)malloc(len);
    memset(buffer,0,len);
```

```
    memset(recup,0,len);

    while((len_r>0)  && (total_len<len)  )
    {
       strcat(buffer,recup);
       memset(recup,0,len);
       len_r=recv(sock,recup,len,0);
       total_len+=len_r;
    }
    free(recup);
    return 1;
}
```

e)Retirer et envoyer un fichier

Une fois que nous pouvons naviguer dans les dossiers, et que l'on sait ce qu'il y a dans les différents répertoires, il devient intéressant de savoir comment échanger des fichiers. Nous venons de voir comment retirer la liste des fichiers présents dans un répertoire, le principe est le même pour downloader ou uploader des fichiers sur le serveur:
Le client envoie une requête pasv (cool on vient de créer une fonction qui fait ça).
Le client se connecte sur le port que le serveur vient de créer.
Le client envoie une requête de téléchargement ou d'upload de fichier
Pour le téléchargement: le client crée un fichier et y copie toutes les données que le serveur lui envoie à travers le canal de trasfert de données.
Pour l'upload (ou envoi de fichiers sur le serveur) le client ouvre le fichier qu'il veut envoyer, il envoie les données contenues dans le fichier dans le canal de transfert. Le serveur enregistre tout ce qui arrive sur son port.

Dans tous les cas le repère qui marque la fin d'un transfert de fichier est la fermeture de la socket de l'expéditeur.
Les deux fontions suivantes montrent comment un client télécharge ou envoie un fichier sur un serveur.

```
/************************
| download_file()
| recoit un fichier d'un serveur ftp
| download_file(SOCKET sock, //socket du canal de dialogue
|        char *file_name); //le nom du fichier à downloader
|
\************************/

int download_file( SOCKET sock, char *file_name)
{
char ip[16]={0};
int *port;
char *buffer;
SOCKET sock_transfert;
int err,taille=0,compteur=0;
FILE *fd;
int ko=0;
int time;


    port=(int*)malloc(sizeof(int));
    if(!send_pasv(sock,(char *)&ip,port)) return 0;
    connection(&sock_transfert,(char*)&ip,*port,500,3);

    buffer=(char*)malloc(1024);
    ZeroMemory(buffer,1024);

    sprintf(buffer,"RETR %s\r\n",file_name);
```

```
    send(sock,buffer,strlen(buffer),0);
    recv_line(sock,buffer);      //le serveur nous annonce qu'il envoie le fichier
    puts(buffer);
    if(traitement(buffer)==PERMISSION_DENIED)
        return 0;


    fd=xfopen(file_name,"w+b");
    printf("Debut de la reception du fichier %s\n",file_name);
    time=GetTickCount();
    while((err=recv(sock_transfert,buffer,1024,0))>0)
    {
        fwrite(buffer,err,1,fd);
                memset(buffer,0,1024);
        taille+=err;
        compteur++;
        ko++;
        if(compteur==20)
        {
            printf("\r %d Ko recus",ko);
            compteur=0;
        }
    }
    time=GetTickCount()-time;
    time/1000;  // pour avoir des sec et non des millisec
    fclose(fd);
    printf("\n");
    printf("%d octets transférés en %d secondes\n",taille,time);
    receive(sock,buffer,256);   //le serveur nous annonce que le transfert est terminé
    puts(buffer);
    return 1;

}




/************************
| upload_file()
| envoi un fichier sur un serveur ftp
| upload_file(SOCKET sock, //socket du canal de dialogue
|         char *file_name); //le nom du fichier à downloader
|
\************************/

int upload_file(SOCKET sock, char *file_name)
{
char ip[16]={0};
int *port;
char *buffer;
SOCKET sock_transfert;
int err,taille=0;
FILE *fd;
int ko;                             //la taille en ko envoyés
int time;           //le temps qu'a duré l'upload
int len;            //la taille en octet de chaque bloc envoyés
int c;
int i;

     //-- établissement du canal de transfert de données --//
    port=(int*)malloc(sizeof(int));
    if(!send_pasv(sock,(char *)&ip,port)) return 0;
    connection(&sock_transfert,(char*)&ip,*port,500,3);

    buffer=(char*)malloc(1024);
    ZeroMemory(buffer,1024);

    sprintf(buffer,"STOR %s\r\n",file_name);                    //annonce de l'envoi
    send(sock,buffer,strlen(buffer),0);
    receive(sock,buffer,1024);     //le serveur nous annonce qu'il est prêt a recevoir le fichier
    puts(buffer);
    if(traitement(buffer)==PERMISSION_DENIED)
        return 0;
```

```
   fd=xfopen(file_name,"r+b");
   rewind(fd); //au cas ou...
   printf("Debut de l'envoi du fichier %s\n",file_name);
   time=GetTickCount();
   i=0;
   ko=0;
   while((c=fgetc(fd))!=EOF)                    //on fait une lecture caractère par caractère
   {
                  buffer[i]=c;
                  if(i==1024)
                  {
                     send(sock_transfert,buffer,1025,0);
                     i=0;
                     ko++;
          printf("\r %d Ko envoyes",ko);
          taille+=1025;
          memset(buffer,0,1024);
                     continue;
                  }
                  i++;
   }
   send(sock_transfert,buffer,i,0);

   taille+=i;
   closesocket(sock_transfert);

   // fclose(fd);   <= la fermeture du fichier engendre une erreur... WhY?
   time=GetTickCount()-time;
   time/1000;  // pour avoir des sec et non des millisec

   printf("\n %d octets envoyes en %d secondes\n",taille,time);
   receive(sock,buffer,256);   //le serveur nous annonce que le transfert s'est terminé
   puts(buffer);
   return 1;

}
```

Voicile code qui permet au serveur de transférer des fichiers:

```
<Debut du code du serveur >
<buffer contient la ligne de commande du client>
…
  else if(!strncmp(buffer,"RETR",4))  //Le Client veut retirer un fichier
   {
      strcpy(argue,(char*)&buffer[5]);
      transfert=xfopen(argue,"r+b");
      printf("Le client download le fichier %s\n",argue);
      sprintf(buffer,"150 Sending file: %s\r\n",argue);
      SENDE;

                  s_transfert=accept(pasv_sock,NULL,NULL);
      closesocket(pasv_sock);
      memset(buffer,0,2048);
      i=0;
      taille=0;
      while((c=fgetc(transfert))!=EOF)
      {
         buffer[i]=c;
         if(i==2047)
         {
            taille+=2048;
                  printf("\r%d Ko",taille/024);
                  if(send(s_transfert,buffer,2048,0)==-1){perror("send dans
envoi de fichier");erreur();}
                  i=0;
                  memset(buffer,0,2048);
                  continue;
               }
               i++;
      }
      send(s_transfert,buffer,i,0);
      printf("\n%d octets envoyes\n",taille+i);
      closesocket(s_transfert);
      fclose(transfert);
      sprintf(buffer,"226 Transfert de %s termine\r\n",argue);
      SENDE;
      return 1;
   }



   else if(!strncmp(buffer,"STOR",4)) //Le Client veut envoyer un fichier
   {
      strcpy(argue,(char*)&buffer[5]);
      transfert=xfopen(argue,"w+b");
      printf("Le client upload le fichier %s\n",argue);
      sprintf(buffer,"150 Opening data canal to receive %s\r\n",argue);
      SENDE;
                  s_transfert=accept(pasv_sock,NULL,NULL);
      closesocket(pasv_sock);
      memset(buffer,0,2048);
      taille=0;
      while((err=recv(s_transfert,buffer,2048,0))>0)
      {
```

```
    fwrite(buffer,err,1,transfert);
    taille+=err;
    err=0;
    printf("\r%d Ko recus",taille/024);
}
fclose(transfert);
```

```
    closesocket(s_transfert);
    printf("\n%d octets copies dans le fichier %s\n",taille,argue);
    sprintf(buffer,"226 Transfert de %s termine\r\n",argue);
    SENDE;
    return 1;
}
```

4) Le Hyper Text Transfert Protocol (http)

Le http fait encore parti des protocoles les plus utilisés sur Internet: c'est ce protocole qui permet de parcourir les pages Web.
Pour retirer une page web, le client envoie: GET /nom_de_la_page_html http/1.0\r\n\r\n
Le serveur répond en envoyant le contenu de la page correspondant au nom.Si la page n'existe pas, le serveur envoie un message d'erreur.

VI

```
/***********************************\
|           ERREUR
|Cette procédure donne les explications
|des erreurs qui peuvent arriver
| les explications proviennent de la doc windows
|
\***********************************/
void erreur(void)
{
int i;
i=WSAGetLastError();
    switch(i)
    {
        case WSANO_DATA :    puts("ERROR : Valid name, no data record of requested type");
                             break;
        case WSANO_RECOVERY:          puts("ERROR : Non recoverable errors, FORMERR, REFUSED,
NOTIMP");
                             break;
        case WSATRY_AGAIN :  puts("ERROR : Non-Authoritative: Host not found, or SERVERFAIL");
                             break;
        case WSAHOST_NOT_FOUND:       puts("ERROR : Authoritative Answer: Host not found");
                             break;
        case WSANOTINITIALISED:       puts("ERROR : A successful WSAStartup must occur before
using this FUNCTION.");
                                      break;
        case WSAENETDOWN :   puts("ERROR : The network subsystem has failed.");
                                      break;
        case  WSAEFAULT      :        puts("ERROR : The addrlen argument is too small or addr
is not a valid part of the user address space");
                                   break;
        case  WSAEINTR:               puts("ERROR : The (blocking) call was canceled through
WSACancelBlockingCall.");
                                   break;
        case  WSAEINPROGRESS:         puts("ERROR : A blocking Windows Sockets 1.1 call is in
progress, or the service provider is still processing a callback function.");
                                   break;
        case  WSAEINVAL      :        puts("ERROR : listen was not invoked prior to accept.");
                                   break;
        case  WSAEMFILE      :        puts("ERROR : The queue is nonempty upon entry to accept
and there are no descriptors available.");
                                   break;
        case  WSAENOBUFS:    puts("ERROR : No buffer space is available.");
                                   break;
        case  WSAENOTSOCK:   puts("ERROR : The descriptor is not a socket.");
                                   break;
        case  WSAEOPNOTSUPP: puts("ERROR : The referenced socket is not a type that supports
connection-oriented service.");
                                   break;
        case WSAEWOULDBLOCK: puts("ERROR : The socket is marked as nonblocking and no
connections are present to be accepted.");
                                   break;
        case  WSAEADDRINUSE :        puts("ERROR : (WSAEADDRINUSE)The specified address is
already in use.");
                                   break;
        case  WSAEALREADY :  puts("ERROR : (WSAEALREADY)A nonblocking connect call is in
progress on the specified socket.Note  In order to preserve backward compatibility, this error
is reported as WSAEINVAL to Windows Sockets 1.1 applications that link to either WINSOCK.DLL
or WSOCK32.DLL.");
                                   break;
        case  WSAEADDRNOTAVAIL:       puts("ERROR : (WSAEADDRNOTAVAIL)The specified address is
not available from the local machine.");
                                   break;
        case  WSAEAFNOSUPPORT:        puts("ERROR : (WSAEAFNOSUPPORT)Addresses in the
specified family cannot be used with this socket.");
                                   break;
        case  WSAECONNREFUSED:        puts("ERROR : (WSAECONNREFUSED) The attempt to connect
was forcefully rejected.");
                                   break;
        case WSAENOTCONN :   puts("ERROR : (WSAENOTCONN) Socket is not connected. ");
                                   break;

        default: printf("ERROR :  Error number %d, WSABASEERR+%d\n",i,i-10000);

    }
```

```
}


/*****************************************************\
|         INITWINSOCK
|
|cette fonction charge winsock.dll. cette opération est
|absolument nécessaire pour utiliser les API Winsock
|(on n'oubliera pas de fermer winsock avec WSACleanup() )
|Retourne :1 si succès ou NULL si échec
\*****************************************************/

int initwinsock(void)
{

    WORD wVersionRequested;
    WSADATA wsaData;
    int err;

    wVersionRequested = MAKEWORD( 2, 0 );
    err = WSAStartup( wVersionRequested, &wsaData );

    if ( err != 0 )
    {
        return NULL;         //impossible de charger winsock.dll
    }

    if ( LOBYTE( wsaData.wVersion ) != 2 || HIBYTE( wsaData.wVersion ) != 0 )
    {
        WSACleanup();
        return NULL;         //impossible de charger winsock.dll
    }

    else
    {
        return 1;            //Winsock.dll succefully initialised
    }
}


/*************************
| int recv_line(sock,buffer)
|
|réception d'une seule ligne terminée par \n
| retourne -1 si échec dans la reception
| si non le nombre de caractères récupérés dans le buffer
\*************************/
int recv_line(SOCKET sock,char *buffer,int len)
{
char recup;
int i=0;
int err;

    while((recup!='\n')&&(i<len))
    {
       err=recv(sock,(char*)&recup,1,0);
       if(err==SOCKET_ERROR)
       {
          if(WSAGetLastError()==WSAECONNABORTED)
              return NEED_TO_RECONNECT;
          else return err;
       }
       else if(err==NULL)
           return -1;
       buffer[i]=recup;
       i++;
    }

    if(i>0) i--;  //pour se mettre à l'index de '\n'

    if(buffer[i-1]=='\r')
    {
        i--;
        buffer[i]=0;
    }
```

```
    else
        buffer[i]=0;  //écrase le saut de ligne avec la fin de la chaine
    return i;
}



// Like recv, but support timeout in milisec.
int recvEx(SOCKET sock,void * data,unsigned long *len,unsigned int milisec_TimeOut)
{
    long read;
    fd_set  set;
    struct timeval timeout;
    int ret;


    timeout.tv_usec = milisec_TimeOut*1000;//ms = 10^3 us
    timeout.tv_sec = 0;

    FD_ZERO(&set);
    FD_SET(sock, &set);
    ret = select(0,&set,NULL,NULL,&timeout);
    if(!ret || ret == SOCKET_ERROR)
    {
         *len = 0;
         return ret;
    }
    if (FD_ISSET(sock, &set))
    {
        read = recv(sock,data,*len,0);
        if(read == SOCKET_ERROR)
        {
             *len = 0;
            return read;
        }
        else
        {
             *len =  read;
            return read;
        }

    }

    *len = 0;
    return SOCKET_ERROR;


}

/***************************\
|  remove_back()
| enlève les backspace d'un buffer
| pour les serveurs qui acceptent des clients telnet
|
| ceci évite d'imposer aux utilisateurs de ne pas se tromper dans une saisie et les
| autorise à utiliser la touche BACKSPACE
| (retour caractère ='\b')
|
\***************************/
void remove_back_space(char *buffer,int len)
{

int index,back;
int decal_dest=0, decal_src=0;

    for(index=0;index+decal_src<len;index++)
    {
        back=0;
        while(buffer[index+decal_src+back]=='\b')
        {
            back++;
        }

        decal_src+=back;// on décale vers la droite de la source du nombre de backspace
```

```
        while(back>0) //on retourne en arrière pour écraser les caractères de la destination
        {
            decal_dest--;
            back--;
            if(index+decal_dest==0)  //pour rester dans la zone de mémoire de la
chaine(decal_dest est négatif)
                break;
        }

        buffer[index+decal_dest]=buffer[index+decal_src];
    }

    buffer[index+decal_dest]=0;
}


/*************************************\
|                  CONNEXION
| Réalise une connexion tcp sur une machine distante
|
| SOCKET connexion(
|                  ,char *adresse  // ip de l'hote
|                  ,int port          // port de l'hote
|                  ,int time          // timeout en millisecondes
|                  ,int threads);     // nombre de tentatives
|
| retourne NULL si échec ou une socket connectée si succès
|
\*************************************/

SOCKET connexion(char *adresse,int port,int time,int thread)
{
    struct sockaddr_in sin;
    int err;                                // pr choper les valeurs de retour
    FD_SET rset;                        //on crée un descripteur pour select()
    struct timeval timeout;             // pr le timeout de select
    unsigned long ttr;
    SOCKET sock;


    /*On rempli la struct de destination*/
    sin.sin_port = htons(port);
    sin.sin_family = AF_INET;
    sin.sin_addr.s_addr=inet_addr(adresse);


    //-- Boucle de threads --//
    d
    {

        //-- On converti les milisecondes en microsecondes et--//
        //--  secondes pr remplir la structure timeout  --//
        timeout.tv_sec=time/1000;
        timeout.tv_usec=(time%1000)*1000;

        closesocket(sock);      // on doit à chaque fois refermer le socket pr le recréer afin
de se reconnecter

        if ((sock=socket(AF_INET,SOCK_STREAM,0))==INVALID_SOCKET)
        {
            MessageBox(NULL,"Can't Create the socket","Merde !",MB_OK);
            return NULL;
        }


        FD_ZERO(&rset);               // on initialise le descripteur à 0
        FD_SET(sock,&rset);  // on ajoute notre socket dans la liste du descripteur

        ttr=TRUE;
        err=ioctlsocket(sock,FIONBIO,&ttr);
        if( err== SOCKET_ERROR)        // on rend le socket non blocant
        {
            erreur();
            return NULL;
        }
```

```
        err=connect(sock, (struct sockaddr *)&sin,sizeof(struct sockaddr));
        if(err== SOCKET_ERROR)
        {
            err=WSAGetLastError();
            if((err!=WSAEALREADY)&&(err!=WSAEWOULDBLOCK))
            {
                erreur();
                return NULL;
            }
        }

        printf(".");       // affiche la progression

        /* apelle select pour verifier la lecture apres le timeout */
        err=select(-1,NULL,&rset,NULL,&timeout);         //on regarde si le socket est
connecté en vérifiant qu'il soit writable
        if(err==SOCKET_ERROR)
        {
            erreur();
            return NULL;

        }
        else if(err!=0) //si il n'y a pas d'erreurs
        {

            ttr=FALSE;
            err=ioctlsocket(sock,FIONBIO,&ttr);  //on le remet en bloking mode
            if (err==SOCKET_ERROR)
            {
                 perror("ioctlsocket back  dans connextion");
                 return NULL;
            }

            return sock;
        }

        thread--;           // on dé crémente la variable thread

    }while(thread>0);       //tant  qu'il reste des threads

    return NULL;

}

/*********************\
| ouvre un port sur la machine locale
| retourne la socket correspondant au port
| si succès, sinon NULL
|
\*********************/
SOCKET OpenPort(int port)
{
struct sockaddr_in sin;
SOCKET sock;
unsigned char val=1;

    if((sock=socket(AF_INET,SOCK_STREAM,0))==SOCKET_ERROR) return NULL;
    sin.sin_port=htons(port);
    sin.sin_addr.s_addr=0;
    sin.sin_family=AF_INET;
    //Allow the socket to be bound to an address which is already in use
    if(setsockopt(sock,SOL_SOCKET,SO_REUSEADDR,&val,sizeof(val))==SOCKET_ERROR) return NULL;
// (By default, a socket cannot be bound  to a local address which is already in use.)

    if(bind(sock,(const struct sockaddr FAR*)&sin,sizeof(SOCKADDR_IN))==SOCKET_ERROR) return
NULL;

    if(listen(sock,SOMAXCONN)==SOCKET_ERROR) return NULL;

    return sock;
}
```

```
/**********************\
| cherche à ouvrir un port
| appelle la fonction précédente
\**********************/
SOCKET manage_port(int *pprt)
{
static int port=500;
SOCKET sock;

    while((sock=open_port(port))==-1)
        port++;

    *pprt=port;
    return sock;
}
```

Ip.c

```
// few more functions for my socket library
// here I developp functions to calculate / generate ip for a scanner

//Calculates number of ip between aip and bip
// ret= aip-bip
u_int subip(u_long aip,u_long bip)
{
u_int sum=0;
int i;
u_char a,b;
    for(i=0;i<4;i++)
    {
         a=(0x0ff&(aip>>(24-i*8)));
         b=(0x0ff&(bip>>(24-i*8)));
         if(b>a)
             sum+= ((255-b)+a)*(u_int)pow(255,i);
         else
             sum+= (a-b)*(u_int)pow(255,i);
    }

    return sum;
}



//increments ip (from the right to the left in the decimal pointed format)
u_long incip(u_long ip)
{
    u_long ret=0;
    int i;
    char carry=1;
    u_long a;


    for(i=0;i<4;i++)
    {
         a=(0x0ff&(ip>>(24-i*8)));
         if(a==0xff && carry)
         {
             a=0;
             carry=1;
         }
         else if(carry)
         {
             a++;
             carry=0;
         }
         ret<<=8;
         ret|=a;
    }
    return ret;
}
```

name.c

```
/*////////////////// ********************** ////////////////
                    resolution de noms
/////////////////// ********************** ////////////////////*/


#include <windows.h>
#include <stdio.h>

/*
char* dns(char* hote);
char *reverse_dns(char *ip);
char *local_ip(void);
char *serv_by_port(int port);

*/


//long to ip
//translate a u_long (from network representation)
// to decimal pointed ip
// the return MUST be released with free()  ! !!
char *ltoip(u_long addr)
{
    SOCKADDR_IN sin;
    sin.sin_addr.s_addr=addr;
    return strdup(inet_ntoa(sin.sin_addr));
}

/*********************************\
|          DNS
|retire l'adresse ip correspondant
|au nom de l'hote
|
|Retourne un pointeur vers l'adresse ip de l'hote si succes ou NULL si échec
\*********************************/
char* dns(char* hote)
{
HOSTENT *host;
struct sockaddr_in sin;
    if ((inet_addr(hote))==INADDR_NONE) //si hote n'est pas une adresse ip, on fait une
requete
    {
        host=gethostbyname(hote);   // <==  il y a un bug ici quand on appelle plusieurs fois
de suite  dns()  ??PrKoi ?
        if (!host)   // la requete a échouée
        {
            return NULL;
        }
        memcpy((unsigned long *)&sin.sin_addr.s_addr,(unsigned long *)host-
>h_addr,sizeof(host->h_addr));
    }

     else
         sin.sin_addr.s_addr=inet_addr(hote);
     return strdup(inet_ntoa(sin.sin_addr));

}



/*****************************************\
|         REVERSE DNS
|Donne le nom du domaine a correspondant à l'adresse IP
|Retourne un pointeur char versle nom du domaine si succès sinon NULL
|
\*****************************************/

char *rdns(char *ip)
{
    unsigned long addr_nbo;
    struct hostent *host;
    char *name;

    addr_nbo=inet_addr(ip);
```

```
    host=gethostbyaddr(( const char FAR *)&addr_nbo,NULL,AF_INET);
    if(!host)
        return NULL;

    name=(char*)malloc(strlen(host->h_name)+1);
    strcpy(name,host->h_name);
    return name;
}

/*****************************\
| local_ip()
|Retourne un pointeur char vers
| l'adresse ip de la machine locale
\*****************************/
char *local_ip(void)
{
char *buffer;
    buffer=(char*)malloc(256);
    memset(buffer,0,256);
    gethostname(buffer,255);
    buffer=dns(buffer);
    return buffer;
}



/************************\
|serv_by_port()
|permet de retirer le nom d'un
|service qui tourne sur le numéro de
|port passé en paramètre
|
\************************/
char *serv_by_port(int port)
{
struct servent *serv;
    serv=getservbyport((int)htons(port),"tcp");
    if(serv) //penser à convertir le numéro en N B Order
    {
        return serv->s_name;
    }

    return NULL;
}


/*****************************\
|
| procédure affichant le contenu d'une structure protoent
| sur la console
|
\*****************************/
void display_proto(struct protoent *proto)
{

    printf("Nom du protocole: %s\n",proto->p_name);
    puts("Aliases du protocole:");

    while(*proto->p_aliases)
    {
        printf("\t %s \n",*proto->p_aliases);
        proto->p_aliases++;
    }

    printf("Numero du protocole: %d\n",proto->p_proto);
}


/************************\
|
|  proto_by_number
|
| renvoie un pointeur char vers le nom du protocol
| correspondant au chiffre passé en paramètre
|
\************************/
char *proto_by_number(int number )
```

```
{
struct protoent *proto;

    proto=getprotobynumber(number);
    if(!proto)
        return NULL;
    return proto->p_name;
}
```

Windows Socket Api .

Fiche Technique *par Phoneus*

accept() :
Accepte une connection sur un socket
SOCKET accept (SOCKET s , struct sockaddr FAR* addr , nt FAR* addrlen);
Ex: sock2=accept(sock,(struct sockaddr_in*)&sin2,sizeof(SOCKADDR_IN));
Retourne : SOCKET_ERROR siéchec sinnon : un socket valide.

bind()
Relie un socket à une adresse ip , un port correspondant
int bind (SOCKET s , const struct sockaddr FAR* name , int namelen);
Ex : err=bind(sock , (LPSOCKADDR)&sin,sizeof(SOCKADDR_IN));
Retourne : SOCKET_ERROR si échec .

closesocket()
Ferme un socket un fois que nous n'en avons plus besoin.
int closesocket (SOCKET s);
Ex: err=closesocket(sock);
Retoutne : 0 si succès ou : SOCKET_ERROR si échec.

connect()
Pour réaliser une connection à un machine distante.
int connect (SOCKET s, const struct sockaddr FAR* name, int namelen);
Ex :err=connect(sock,(LPSOCKADDR)&sin ,sizeof(SOCKADDR));
Retourne SOCKET_ERROR si échec sinon retoure un socket valide .

Connection()
Comme connect() mais prend gére le timeout
int connection(int *sock,char *adresse,int port,int time,int thread)
Ex :connection(&sock,buffer,port,500,3) ;
Retourne : 1 si succès sinon SOCKET_ERROR

gethostname()
Retourne le nom de domaine standard.
int gethostname (char FAR * name , int namelen);
Ex : err=gethostname(buffer,1024);
Retourne: 0 si succès sinon :SOCKET_ERROR

gethostbyname()
Retourne des informations sur l'host à qui correspond le hostname.
struct hostent FAR * gethostbyname (const char FAR * name);
Ex :Cet exemple montre comment retirer son adresse IP
Hst=gethosbyname(buffer);
memcpy(&sin.sin_addr,Hst->h_addr,sizeof(Hst->h_addr)); // car inet_ntoa() prend pour paramètre un sin_addr
printf("Adreses IP locale : %s\n",inet_ntoa(sin.sin_addr));
printf("Nom de domaine : %s"\n,Hst->h_name) ;
Retourne : 0 si échec , sinon un pointeur vers une structure hostent

htons()
Convertit un unsigned short en format TCP/IP network by order .
u_short htons (u_short hostshort);
Ex: sin.sin_port=htons(21);
Retourne : Une valeur au format TCP/IP

inet_ntoa()
Convertit une adresse ip du format sin_addr (network by order) au format decimal pointé sous forme de string.
char FAR * inet_ntoa (struct in_addr in);
Ex : printf("Ip du client : %s\n",inet_ntoa(sin2.sin_addr)) ;
Retourne NULL si échec

inet_addr()
Convertit une adresse ip du format string au format sin_addr pour une structure sockaddr_in.
unsigned long inet_addr (const char FAR * cp);
Ex : sin.sin_addr=inet_addr(125.123.56.85);
Retourne INADDR_NONE si l'adresse ip est invalide.

initwinsock()
Initialise winsock.dll , ce qui est impératif pour utiliser les fonctions WSA..(Fonction non standard).
Int initwinsock(void) ;
Ex : err= initwinsock() ;
Retoutne SOCKET_ERROR si échec.

ioctlsocket()
Contrôle le mode d'un socket.
int ioctlsocket (SOCKET s , long cmd , u_long FAR* argp);
long cmd : FIONBIO ou FIONREAD ou SIOCATMARK
Ex : err=ioctlsocket(sock,FIONBIO,&block);
Retourne 0 si success ou SOCKET_ERROR si échec.

listen()
Attends une connection sur un socket.
int listen (SOCKET s , int backlog);
err=listen(sock,5);
Retourne : SOCKET_ERROR si échec .

recv():
Reception de données sur un socket.
int recv (SOCKET s , char FAR* buf , int len , int flags);
Ex : err=recv(sock,buffer,strlen(buffer),0);
Retourne SOCKET_ERROR si échec.

receive():
Reception d'un buffer sur un socket. (Fonction non standard)
Int receive(int *sock, char *buffer_de_reception);
Ex: err=receive(&sock,buffer);
Retoutne SOCKET_ERROR si échec.

socket()
Crée un socket pour un usage spécifique.
SOCKET socket (int af , int type , int protocol);
Ex : sock=socket(AF_INET,SOCK_STREAM,0) ;
Retoutne SOCKET_ERROR si échec. Si non un socket valide

select()
Détetermine l' état d'un ou de plusieurs socket en attente.
int select (int nfds , fd_set FAR * readfds , fd_set FAR * writefds , fd_set FAR * exceptfds
, const struct timeval FAR * timeout);
FD_CLR(s, *set) Removes the descriptor s from set.
FD_ISSET(s, *set) Nonzero if s is a member of the set. Otherwise, zero.
FD_SET(s, *set) Adds descriptor s to set.
FD_ZERO(*set) Initializes the set to the NULL set.
Ex: FD_SET rset;
FD_ZERO(&rset);
FD_SET(sock,&rset);
ret=select(NULL,&rset,NULL,NULL,&timestruct);
Retourne : Le nombre de descripteurs prêts ou SOCKET_ERROR si échec.

send()
Envoie des données en utilisant un socket.
int send (SOCKET s , const char FAR * buf , int len , int flags);
Ex : err=send(sock,buffer,strlen(buffer),0) ;
Retoutne SOCKET_ERROR si échec.

struct hostent
Structure retournée par gethostbyname() et gethostbyaddr()

```
struct  hostent {
        char    FAR * h_name;           /* official name of host */
        char    FAR * FAR * h_aliases;  /* alias list */
        short   h_addrtype;             /* host address type */
        short   h_length;               /* length of address */
        char    FAR * FAR * h_addr_list; /* list of addresses */
#define h_addr  h_addr_list[0]          /* address, for backward compat */
};
typedef struct hostent HOSTENT;
typedef struct hostent *PHOSTENT;
typedef struct hostent FAR *LPHOSTENT;
```

Ex : Cet exemple affiche le nom de domaine complet de l'host.

```
 LPHOSTENT Hst;   ou struct hostent *Hst;
Hst=gethostbyname(buffer);
printf("Host name  : %s\n", Hst->h_name);
```

struct sockaddr_in
Structure fondamentale dans laquelle on met et on récupère l'adresse ip , le port

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

```
/* Microsoft Windows Extended data types */
typedef struct sockaddr_in SOCKADDR_IN;
typedef struct sockaddr_in *PSOCKADDR_IN;
typedef struct sockaddr_in FAR *LPSOCKADDR_IN;
```

Ex : struct sockaddr_in sin ;
Sin.sin_port=htons(21);
Sin.sin_family=AF_INET;
Sin.sin_addr=inet_addr("123.123.153.42"); ou :sin.sin_addr.sin_zero=0;
memset(sin.sin_zero,0,8);

struct in_addr.
Structure utilisée dans la structure sockaddr_in

```
struct in_addr {
      union {
            struct { u_char s_b1,s_b2,s_b3,s_b4; } S_un_b;
            struct { u_short s_w1,s_w2; } S_un_w;
            u_long S_addr;
            } S_un;
#define s_addr  S_un.S_addr                 /* <=  can be used for most tcp & ip code */
#define s_host  S_un.S_un_b.s_b2            /* host on imp */
#define s_net   S_un.S_un_b.s_b1            /* network */
#define s_imp   S_un.S_un_w.s_w2            /* imp */
#define s_impno S_un.S_un_b.s_b4            /* imp # */
#define s_lh    S_un.S_un_b.s_b3            /* logical host */
};
```

Ex : sin.sin_addr.s_addr=0; à la place de : sin.sin_addr.S_un.S_addr=0;

struct sockaddr
Structure utilisée par les functions bind,accept

```
struct sockaddr {
      u_short sa_family;            /* address family */
      char    sa_data[14];          /* up to 14 bytes of direct address */
};
/* Microsoft Windows Extended data types */
typedef struct sockaddr SOCKADDR;
typedef struct sockaddr *PSOCKADDR;
typedef struct sockaddr FAR *LPSOCKADDR;
```

struct timeval
Structure utilisée par la function select() pour determiner un timeout

```
struct timeval {
      long    tv_sec;         /* secondes */
      long    tv_usec;        /*  microsecondes */
};
```

Ex: timeval timestruct;
timestruct.tv_sec=2;
timestruct.tv_usec=0;

WSACleanup()
Ferme proprement winsock.dll quand on a fini de l'utiliser.
int WSACleanup (void);
Ex : WSACleanup();
Retoutne 0 si succès ou : SOCKET_ERROR si échec.

Bibliographie:

Une introduction à TCP/IP J.Davidson
Windows Socket Network Programming Bob Quinn & Dave Shute
Articles de Blacksun.box.sk
Team Tipiak (Un grand Merci à **Obscurer**)
Linux Magazine (pour la mise en application de select)

Si vous êtes un professeur , ou responsable d'une université d'informatique et que vous trouvez que cet article correspond à un de ceux que vous attendez, sachez que je me ferai un plaisir de réaliser une conférence présentant tout ce qui est expliqué ici.

Pour toute suggestion, critique ou remerciement : bremardt@esiee.fr