Fichier serveur_tcp.c pour ipv4

Liste include
<stdio.h><netinet/in.h><netdb.h> <sys/socket.h>,<unistd.h><string.h><stdlib.h>

```
int main(int argc, char *argv[])   {
  int s_ecoute,scom, lg_app,i,j;
  struct sockaddr_in adr;
  struct sockaddr_storage recep;
  char buf[1500], renvoi[1500], host[1024],service[20];

   s_ecoute=socket(AF_INET,SOCK_STREAM,0);
   printf("creation socket\n");

   adr.sin_family=AF_INET;
   adr.sin_port=htons(atoi(argv[1]));
   adr.sin_addr.s_addr=INADDR_ANY;

   if (bind(s_ecoute,(struct sockaddr *)&adr,sizeof(struct sockaddr_in)) !=0) {
     printf("probleme de bind sur v4\n");
     exit(1); }

   if (listen(s_ecoute,5) != 0)      {
       printf("pb ecoute\n"); exit(1);}

   printf("en attente de connexion\n");
   while (1)  {
      scom=accept(s_ecoute,(struct sockaddr *)&recep, (socklen_t *)&lg_app);
      getnameinfo((struct sockaddr *)&recep,sizeof (recep), host, sizeof(host),service,
sizeof(service),0);   //optionnelle pour info
       printf("recu de %s venant du port %s\n",host, service);   //optionnnelle
                 while (1)     {
                    recv(scom,buf,sizeof(buf),0);
                    printf("buf recu %s\n",buf);
                    bzero(renvoi,sizeof(renvoi));
                    for(i=strlen(buf)-1,j=0;i>=0;i--,j++)  renvoi[j]=buf[i];
                    renvoi[j+1]='\0';

                    send(scom,renvoi,strlen(renvoi),0);
                    bzero(buf,sizeof(buf));
                    if (strcmp(renvoi,"NIF") == 0)    break;   }
                close(scom);   }
              close(s_ecoute); }
```

Fichier client_tcp.c pour ipv4

Liste include
<stdio.h><stdlib.h> <netinet/in.h> <netdb.h> <sys/socket.h> <string.h><unistd.h>

```
void main(int argc, char *argv[])
{
  char buffer[200],texte[200];
  int port, rc, sock,i,c;
  struct sockaddr_in addr;
  struct hostent *entree;

  if (argc !=3)  {
     printf("usage : client_tcp nom_machine numero_port\n");
             exit(1);    }

  addr.sin_port=htons(atoi(argv[2]));
  addr.sin_family=AF_INET;
  entree=(struct hostent *)gethostbyname(argv[1]);
  bcopy((char *)entree->h_addr,(char *)&addr.sin_addr,entree->h_length);

  sock= socket(AF_INET,SOCK_STREAM,0);

  if (connect(sock, (struct sockaddr *)&addr,sizeof(struct sockaddr_in)) < 0) {
    printf("probleme connexion\n");
    exit(1); }

  printf("connexion passe\n");

  while (1) {
     bzero(texte,sizeof(texte));
     bzero(buffer,sizeof(buffer));
     i = 0;
     printf("Entrez une ligne de texte : \n");
     while((c=getchar()) != '\n')
             texte[i++]=c;
     send(sock,texte,strlen(texte)+1,0);
     recv(sock,buffer,sizeof(buffer),0);
     printf("recu %s\n",buffer);
     if (strcmp("FIN",texte) == 0)          break;
   }

  close(sock); }
```

Fichier serveur_tcp.c pour ipv6 (automatique)

```
#include <stdio.h>, <stdlib.h>, <netinet/in.h>,<netdb.h>,<sys/socket.h>,<strings.h>,
<string.h>, <ifaddrs.h>

int main(int argc, char *argv[])
{
 int sockx,scom, lg_app, ecode,i,j;
 struct sockaddr_in6 adr6, appelant, *so6;
 struct sockaddr_storage recep;
 char buf[1500], renvoi[1500], host[1024], service[20], serv[10];
 char adresseipv6[INET6_ADDRSTRLEN];
 char adresseipv4[INET_ADDRSTRLEN];
  struct addrinfo *res1,*rres1, hints;
  struct in6_addr ip;
  struct ifaddrs *res = NULL;

  printf("Une facon de faire automatique\n");

  memset(&hints, 0, sizeof(hints));
  hints.ai_flags = AI_PASSIVE;
  hints.ai_socktype= SOCK_STREAM;
  //  hints.ai_family = AF_INET6;  pour avoir que ipV6
  hints.ai_family = PF_UNSPEC;  // pour avoir tout ip

  sprintf(serv,"%d",atoi(argv[1]));
  ecode = getaddrinfo(NULL, serv, &hints, &rres1);

  if(ecode) {
     printf("problem %s\n",gai_strerror(ecode));
     exit(1);     }

  for(res1 = rres1; res1; res1 = res1->ai_next) {
    // en linux, un seul bind possible
    if (res1->ai_family == AF_INET6) {
              sockx = socket(res1->ai_family,res1->ai_socktype,res1->ai_protocol);

              if (bind(sockx, res1->ai_addr, res1->ai_addrlen) < 0) {
                printf("Problerme de bind\n");
                exit(1);
              }
              listen(sockx,5);
             }
    }
   printf("fin de l'initialisation automatique\n");

             while (1)
             {
               scom=accept(sockx,(struct sockaddr *)&recep, (unsigned long *)&lg_app);
               getnameinfo((struct sockaddr *)&recep,sizeof (recep), host, sizeof(host),service,
sizeof(service),0);
                printf("recu de %s\n",host);

                while (1) {
                   recv(scom,buf,sizeof(buf),0);

                   printf("buf recu %s %d\n",buf,strlen(buf));
                   bzero(renvoi,sizeof(renvoi));
                   for(i=strlen(buf)-1,j=0;i>=0;i--,j++)  renvoi[j]=buf[i];
                   renvoi[j+1]='\0';

                   send(scom,renvoi,strlen(renvoi),0);
                   bzero(buf,sizeof(buf));

                   if (strcmp(renvoi,"NIF") == 0)   break;     }
               close(scom); }
             close(sockx);
}
```

Fichier serveur_tcp.c pour ipv6 (manuel)

```
int main(int argc, char *argv[])
{
  int s_ecoute, scom, lg_app,i,j;
  // meme variable…

   /* recuperation et affichage des adresses IP */
  if (getifaddrs(&res))
    printf("pb sur getifaddrs\n");

   printf("Recuperation des adresse IP\n");

   for(; res !=NULL;res = res ->ifa_next)
     {
```

```
   if ( (res->ifa_addr)->sa_family ==AF_INET6) {
            so6 = (struct sockaddr_in6 *)res->ifa_addr;
            bcopy(so6->sin6_addr.s6_addr, &ip,16);
            inet_ntop(AF_INET6,&ip,adresseipv6, sizeof(adresseipv6));
             printf("adresse ipv6 %s\n",adresseipv6);       }
   if ( (res->ifa_addr)->sa_family ==AF_INET) {
            so4 = (struct sockaddr_in *)res->ifa_addr;
            inet_ntop(AF_INET,&(so4->sin_addr),adresseipv4,sizeof(adresseipv4));
            printf("adresse ipv4 %s\n",adresseipv4); }
   }
  printf("debut initialisation manuelle\n");

  s_ecoute=socket(AF_INET6,SOCK_STREAM,0);

  adr.sin6_family=AF_INET6;
  adr.sin6_port=htons(atoi(argv[1]));
  memcpy((void *)&adr.sin6_addr, (void *)&in6addr_any, sizeof in6addr_any);

  if (bind(s_ecoute,(struct sockaddr *)&adr,sizeof(struct sockaddr_in6)) !=0){
     printf("probleme de bind sur v6\n");
     exit(1);      }

   if (listen(s_ecoute,5) != 0) {
     printf("pb ecoute\n"); exit(1);}

  printf("en attente de connexion\n");
 …..
```

Fichier client_tcp.c pour ipv6

```
#include <stdio.h>,<stdlib.h>, <netinet/in.h>,<netdb.h>,<sys/socket.h>,<strings.h>,
<string.h>, <ifaddrs.h>

void main(int argc, char *argv[])
{
  char buffer[200], texte[200];
  int port, rc, sock,c,i;
  struct in6_addr serveraddr;
  struct addrinfo hints, *res=NULL;

  if (argc !=3) {
     printf("usage : clientv6 nom_serveur port\n");
            exit(1);    }

memset(&hints, 0x00,sizeof(hints));
hints.ai_flags = AI_NUMERICSERV;
hints.ai_family = AF_INET6;
hints.ai_socktype = SOCK_STREAM;

port=atoi(argv[2]);

//on teste sil'adresse IP est numeric ou textuel
// si rc = 1 , c'est numeric donc resolution numeric
  rc = inet_pton(AF_INET6, argv[1], &serveraddr);
if (rc ==1)
  hints.ai_flags |= AI_NUMERICHOST;

// recuperation des infos sur le serveur distant
  rc = getaddrinfo(argv[1], argv[2], &hints, &res);
if (rc != 0)
  {  printf("pb %s\n",gai_strerror(rc));exit(1);}

sock= socket(res->ai_family, res->ai_socktype, res->ai_protocol);

if (connect(sock, res->ai_addr, res->ai_addrlen) <0) {
  printf("probleme connexion\n");
  exit(1); }

while (1)
  {
    bzero(texte,sizeof(texte));
    bzero(buffer,sizeof(buffer));
    i = 0;
    printf("Entrez une ligne de texte : \n");
    while((c=getchar()) != '\n')
            texte[i++]=c;
    printf("texte %s\n",texte);
    send(sock,texte,strlen(texte)+1,0);
    recv(sock,buffer,sizeof(buffer),0);
    printf("recu %s\n",buffer);


    if (strcmp("FIN",texte) == 0) break;
  }

    close(sock); }
```

Fichier serveur_udp.c pour ipv4

```
#include <stdio.h>
#include <netinet/in.h>
#include <netdb.h>
#include <sys/socket.h>

int main(int argc, char *argv[]) {
  int s_ecoute, recus;
  char mes[20], host[200],service[10];
  struct sockaddr_in adr;
  struct sockaddr_storage recep;
  int lg_app;

  s_ecoute=socket(AF_INET, SOCK_DGRAM,0);
  printf("la socket est cree\n");

  adr.sin_family=AF_INET;
  adr.sin_port=htons(atoi(argv[1]));
  adr.sin_addr.s_addr=INADDR_ANY;

  if (bind(s_ecoute,(struct sockaddr *) &adr, sizeof (struct sockaddr_in)) !=0)
    {
      printf("Pb de connexion\n");
      exit(1); }

  printf("je suis en ecoute\n");
  lg_app=sizeof(struct sockaddr_storage);
  recus=recvfrom(s_ecoute,mes,sizeof(mes),0,(struct sockaddr *) &recep,&lg_app);
  if (recus <=0)
    printf("bug\n");
  else {
    getnameinfo((struct sockaddr *)&recep,sizeof (recep), host, sizeof(host),service,
sizeof(service),0);
    printf("message : %s venant de %s et du port %s\n",mes,host,service);
    sendto(s_ecoute,"OK",3,0,(struct sockaddr *)&recep, lg_app); }
  close(s_ecoute);
}
```

Fichier client_udp.c pour ipv4

```
#include <stdio.h>
#include <netinet/in.h>
#include <netdb.h>
#include <sys/socket.h>
#include <string.h>


int main(int argc, char *argv[])
{
  int s_com,emis;
  char mes[20];
  struct sockaddr_in adr;
  struct sockaddr_storage recep;
  struct hostent *entree;
  int lg_app;

  if (argc !=3)
    {printf("usage : prog nom_serveur port\n"); exit(1);}

  s_com=socket(AF_INET, SOCK_DGRAM,0);
  printf("la socket est cree\n");

  adr.sin_family=AF_INET;
  adr.sin_port=htons(atoi(argv[2]));

entree= (struct hostent *)gethostbyname(argv[1]);
  bcopy((char *) entree->h_addr, (char *)&adr.sin_addr, entree->h_length);

  printf("entrez un mot\n");
  scanf("%s",mes);
  emis=sendto(s_com,mes,sizeof(mes),0,(struct sockaddr *)&adr,sizeof(adr));

  if (emis <=0)
    printf("gros probleme\n");

  recvfrom(s_com,mes,sizeof(mes),0,(struct sockaddr *)&recep,&lg_app);
  printf("message recu : %s\n",mes);
  close(s_com);
}
```

Fichier serveur_udp.c pour ipv6

```
#include <stdio.h>
#include <netinet/in.h>
#include <netdb.h>
#include <sys/socket.h>
#include <string.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
  int s_ecoute, recus;
  char mes[20], host[200],service[10];
  struct sockaddr_in6 adr;
  struct sockaddr_storage recep;
  int lg_app;

  s_ecoute=socket(AF_INET6, SOCK_DGRAM,0);
  printf("la socket est cree\n");

  adr.sin6_family=AF_INET6;
  adr.sin6_port=htons(atoi(argv[1]));
   memcpy((void *)&adr.sin6_addr, (void *)&in6addr_any, sizeof in6addr_any);

  if (bind(s_ecoute,(struct sockaddr *) &adr, sizeof (struct sockaddr_in6)) !=0)
    {
      printf("Pb de connexion\n");
      exit(1); }

  printf("je suis en ecoute\n");
  lg_app=sizeof(struct sockaddr_storage);
  recus=recvfrom(s_ecoute,mes,sizeof(mes),0,(struct sockaddr *) &recep,&lg_app);
  if (recus <=0)
    printf("bug\n");
  else {
    getnameinfo((struct sockaddr *)&recep,sizeof (recep), host, sizeof(host),service,
sizeof(service),0);
    printf("message : %s venant de %s et du port %s\n",mes,host,service);
    sendto(s_ecoute,"OK",3,0,(struct sockaddr *)&recep, lg_app); }
  close(s_ecoute);
}
```

Fichier client_udp.c pour ipv6

```
#include <stdio.h>, <netinet/in.h>,<netdb.h>, <sys/socket.h>, <string.h>, <stdlib.h>

int main(int argc, char *argv[])
{
  int s_com,emis, port, rc, lg_app;
  char mes[20];
  struct sockaddr_in6 adr;
  struct sockaddr_storage recep;
  struct in6_addr serveraddr;
  struct addrinfo hints, *res=NULL;

  if (argc !=3)
    {printf("usage : udpcliv4 nom_serveur port\n"); exit(1);}

 memset(&hints, 0x00,sizeof(hints));
  hints.ai_flags = AI_NUMERICSERV;
  hints.ai_family = AF_INET6;
  hints.ai_socktype = SOCK_DGRAM;
  port=atoi(argv[2]);

  //on teste sil'adresse IP est numeric ou textuel
      rc = inet_pton(AF_INET6, argv[1], &serveraddr);

  if (rc ==1)    hints.ai_flags |= AI_NUMERICHOST;

   rc = getaddrinfo(argv[1], argv[2], &hints, &res);
  if (rc != 0)
    {  printf("pb %s\n",gai_strerror(rc));exit(1);}

  s_com= socket(res->ai_family, res->ai_socktype, res->ai_protocol);
  printf("la socket est cree\n");

  printf("entrez un mot\n");
  scanf("%s",mes);
  emis=sendto(s_com,mes,sizeof(mes),0,(struct sockaddr *)res->ai_addr,res->ai_addrlen);

  if (emis <=0)     printf("gros probleme\n");

  recvfrom(s_com,mes,sizeof(mes),0,(struct sockaddr *)&recep,&lg_app);
  printf("message recu : %s\n",mes);
  close(s_com);}
```

Fichier client_tcp.c pour ipv4 avec csocket (windows)

```
#include "stdafx.h", <string.h>, <iostream>, <afxsock.h>
using namespace std;
/* Attention : Change project settings:
     - Open the Project Settings window ("Alt+F7")
     - Select the General tab
 - Choose use MFC in a shared DDL in the Microsoft Foundation Classes box */

void DieError(char* errorMessage) {
   cout << errorMessage << endl; exit(0);}

int _tmain(int argc, _TCHAR* argv[]) {
  if (!AfxWinInit(::GetModuleHandle(NULL), NULL, ::GetCommandLine(), 0)) {
    DieError("Fatal Error: MFC initialization failed");}

            cout << "debut du programmme\n";
            int port = 16515;
            CSocket servSocket;

            AfxSocketInit(NULL);
            if (!servSocket.Create(port)) { DieError("servSock.Create() failed");}
            if (!servSocket.Listen(5)) {DieError("servSock.Listen() failed");}

             for(;;) {
               sockaddr_in Addr; // Client address
               int clntLen = sizeof(Addr); //taille du client
               CSocket sock_com;

               if (!servSocket.Accept(sock_com)) {DieError("servSock.Accept() failed");}

              if (!sock_com.GetPeerName((sockaddr *)&Addr, &clntLen)) {
                              DieError("clntSock.GetPeerName() failed");}

               cout << "Handling client " << inet_ntoa(Addr.sin_addr) << endl;

            while (1)
            {           int recvMsgSize, i, j;
                        char Buffer[100], renvoi[100];

                        recvMsgSize = sock_com.Receive(Buffer, 100, 0);
                        if (recvMsgSize < 0) { DieError("clntSock.Receive() failed");}
                        cout << "message recu : " << Buffer << "\n";

                        for(i=strlen(Buffer)-1,j=0;i>=0;i--,j++) renvoi[j]=Buffer[i];
                        renvoi[j+1]='\0';
                        sock_com.Send(renvoi, strlen(renvoi), 0);
                        if (strcmp(renvoi,"NIF")==0) break; }
                        sock_com.Close();
             }
             servSocket.Close();
            getchar();
            return 0;
}
```

Fichier client_tcp.c pour ipv4 en socket (windows)

```
#include "pch.h",<stdio.h>,<iostream>,<ws2tcpip.h>,"winsock2.h"
#pragma comment(lib, "ws2_32.lib")
using namespace std;

int main(int argc, CHAR* argv[])
{
            int s_com, i, c;
            char phrase[50], buf[50];
            WSADATA info;
            struct addrinfo *result = NULL,*ptr = NULL, hints;
            DWORD dwRetval;

            if (WSAStartup(MAKEWORD(2, 0), &info) == SOCKET_ERROR)
            {  cout << "erreur dans l'initialisation " << endl;
                        WSACleanup();exit(1);}

            //--------------------------------
            // Setup the hints address info structure which is passed to the getaddrinfo() function
            ZeroMemory(&hints, sizeof(hints));
            hints.ai_family = AF_INET;
            hints.ai_socktype = SOCK_STREAM;
            hints.ai_protocol = 0;

            dwRetval = getaddrinfo("nom_machine", "numero_port", &hints, &result);
            if (dwRetval != 0) {
                        printf("getaddrinfo failed with error: %d\n", dwRetval);
                        WSACleanup();
                        return 1;  }
```

```
	printf("getaddrinfo returned success\n");
	s_com = socket(AF_INET, SOCK_STREAM, 0);
	cout << " la socket est cree\n";

	if (connect(s_com, result->ai_addr, result->ai_addrlen) < 0)
	{
		cout << "probleme de connexion";
		getchar();exit(1);}
	freeaddrinfo(result);
	while (true)
	{
		printf("Entrez une ligne de texte\n");
		i = 0;
		while ((c = getchar()) != '\n')
			phrase[i++] = c;
		phrase[i] = '\0';
		printf("ligne trouve %s\n", phrase);
		send(s_com, phrase, i + 1, 0);
		for (i = 0; i < sizeof(buf);i++)
			buf[i] = '\0';
		recv(s_com, buf, sizeof(buf), 0);
		printf("message recu %s\n", buf);
		if (strncmp(phrase, "FIN", 3) == 0)
			break;
	}
	WSACleanup();
	return 0;
}
```