Moser Loïc

TP 5 - Serveur TCP

ATI01 - Algorithmie avancée

B- Questions de Compréhension des Programmes fournis

1. La classe d'objet **Serveur_TCP** contient un constructeur, **Serveur_TCP**, un destructeur, les variables et les fonctions qu'on utilisera dans le c++.
2. L'objet créé à partir de la classe **Serveur_TCP** est **Serveur_TCP**
3. La classe d'objet **Serveur_TCP** contient la vérification de version de winsocket et indique au client si la version n'est pas la bonne.

C- Partie algorithmique et programmation

1.
```
Serveur_TCP::Serveur_TCP()
{
    int erreur;
    VersionRequise = MAKEWORD(2,2);
    ErreurdeVersion = 2;

    //C1
    erreur=WSAStartup(VersionRequise, &InformationsdeVersion);
    //C1

    if (erreur!=0)
      {
        cout<<"Serveur : La version 2.2 de winsocket n'est pas installee sur cette machine"<<endl;
        ErreurdeVersion = 1;
        WSACleanup();
        system("PAUSE");
      }
    else
      {
        cout<<"Serveur : Winsock.dll trouve !!"<<endl;
        //C1
        cout<<"Serveur : Etat serveur TCP/IP : "<<InformationsdeVersion.szSystemStatus<<endl;
        //C1
        ErreurdeVersion = 0;
          }
}
int Serveur_TCP::NumerodeVersion()
{
        //C1
         return ErreurdeVersion;
         //C1
}
```

2.
```
int Serveur_TCP::CreationdunSocket()
{
    SocketTCP = INVALID_SOCKET;
    FamilledAdresse = AF_INET;
    TypedeCommunication = SOCK_STREAM;
    Protocole = IPPROTO_TCP;
    dwflag = WSA_FLAG_OVERLAPPED;

    SocketTCP = WSASocket(FamilledAdresse, TypedeCommunication, Protocole, NULL, 0, dwflag);

    if (SocketTCP == INVALID_SOCKET)
    {
        cout<<"Serveur : Le Canal de Communication avec les clients n'a pas pu etre etabli pour la raison suivante : "<<endl;
        cout<<WSAGetLastError()<<endl;
        WSACleanup();
        return 1;
    }
    else
    {
       // cout<<"Serveur : Le Canal de Communication avec les Clients est etabli."<<endl;
        return 0;
    }
}
```

3.
```
int Serveur_TCP::ConnexionauSocket()
{
    int reponse = SOCKET_ERROR;

    Adresse.sin_family = AF_INET;
    Adresse.sin_addr.S_un.S_addr = inet_addr("127.0.0.1");
    Adresse.sin_port = htons(23);

    reponse = bind(SocketTCP, (struct sockaddr*)&Adresse, sizeof(Adresse));

    if (reponse == SOCKET_ERROR)
    {
        cout<<"Serveur : La Connexion du Serveur n'a pas pu etre etablie pour la raison suivante : "<<endl;
        cout<<WSAGetLastError()<<endl;
        closesocket(SocketTCP);
        WSACleanup();
        return 1;
    }
    else
    {
        //cout<<"Serveur : La Connexion du Serveur est etablie."<<endl;
        return 0;
    }
}
```

4.
```
int Serveur_TCP::SocketenEcoute()
{
    int reponse;
    reponse = listen(SocketTCP, SOMAXCONN);

    if (reponse == SOCKET_ERROR)
    {
        cout<<"Serveur : La mise en ecoute du Serveur a echoue a cause de l'erreur "<<endl;
        cout<<WSAGetLastError()<<endl;
        closesocket(SocketTCP);
        WSACleanup();
        return 1;
    }
    else
    {
        //cout<<"Serveur : La mise en ecoute du Serveur est etablie"<<endl;
        return 0;
    }
}
```

5.
```
int Serveur_TCP::AcceptationdunClient()
{
    ClientTCP = SOCKET_ERROR;

    int AdresseTaille = sizeof(Adresse);
    ClientTCP = WSAAccept(SocketTCP, (struct sockaddr*)&Adresse, &AdresseTaille, NULL, 0);

    if (ClientTCP == SOCKET_ERROR)
    {
        cout<<"Serveur : Connexion refusee"<<endl;
        cout<<WSAGetLastError()<<endl;
        closesocket(SocketTCP);
        WSACleanup();
        return 1;
    }
    else
    {
        //cout<<"Serveur : Connexion acceptee."<<endl;
        return 0;
    }
}
```

6.
```
int Serveur_TCP::ReceptiondunMessage()
{
    int reponse;
    WSABUF Reception;
    DWORD OctetsduMessage, Flag = 0;
    char  buffer[T_bloc];

    Reception.len = T_bloc;
    Reception.buf = buffer;
    // Réception du message du client
    reponse = WSARecv(ClientTCP, &Reception, 1, &OctetsduMessage, &Flag, NULL, NULL);


    if (reponse == SOCKET_ERROR) // Remplacer les ?
      {
        cout<<"Serveur : Le client s'est deconnecter "<< endl;
        closesocket(ClientTCP);
        ClientTCP = 0;
        return 1;
      }
      else
      {
        cout<<"Message recu (Octets : "<<OctetsduMessage<<") "<<Reception.buf<<endl;
         return 0;
      }
    return -1;
}
```

7.
```
int Serveur_TCP::EnvoiAccusedeReception()
{
    int reponse;
    DWORD OctetsduMessage;
    WSABUF AccusedeReception;
    char ContenuduMessage[1024] = "Serveur : J'ai bien recu votre message\0";

    AccusedeReception.len = 1024;
    AccusedeReception.buf = ContenuduMessage;
    reponse = WSASend(ClientTCP, &AccusedeReception, 1, &OctetsduMessage, 0, NULL, NULL);

    if (reponse == SOCKET_ERROR)
    {
        cout<<"Serveur : L'accuse de reception n'a pas pu etre envoye au serveur client pour la raison suivante : "<<endl;
        cout<<WSAGetLastError()<<endl;
        closesocket(SocketTCP);
        WSACleanup();
        return 1;
    }
    else
    {
        cout<<"Serveur : L'accuse de reception a ete envoye au client."<<endl;
        return 0;
    }
}
```

8.
```
// Création du canal internet de communication avec les clients
ObjetServeurTCP.CreationdunSocket();

// Le socket est créé, nous pouvons mettre en place le processus de communication
cout<<"Serveur : Le Socket est cree !!!"<<endl;


// Création de l'adresse physique de la liaison de communication
// Le serveur est physiquement en place.
ObjetServeurTCP.ConnexionauSocket();


// Mise en écoute du serveur
ObjetServeurTCP.SocketenEcoute();

cout<<"Serveur : le serveur est en ecoute, il attend la connexion d'un client .... "<<endl;


// Attente d'une demande de connexion du client
ObjetServeurTCP.AcceptationdunClient();

cout<<"Serveur : Le serveur a accepte la demande de connexion du client."<<endl;
cout<<"Serveur : Attente de reception d'un message"<<endl;


// Réception du message provenant du client
ObjetServeurTCP.ReceptiondunMessage();


// Envoi de l'accusé de réception au client
ObjetServeurTCP.EnvoiAccusedeReception();


// Déconnexion
ObjetServeurTCP.FindelaConnexion();
```

9.
```
do{
// Réception du message provenant du client
ObjetServeurTCP.ReceptiondunMessage();


// Envoi de l'accusé de réception au client
}while(ObjetServeurTCP.EnvoiAccusedeReception() == 0);

// Déconnexion
ObjetServeurTCP.FindelaConnexion();
```