Objectifs : Apprendre à réaliser des échanges de base entre un programme serveur et un ou plusieurs programmes clients et comprendre le fonctionnement de TCP.

Il sera important de suivre les étapes suivantes :

1. effectuer une réflexion avant réalisation pour :
   — comprendre la structure de l'application (définir les programmes qui la composent et le rôle de chacun).
   — définir un protocole d'application / d'échange.
2. implémenter les programmes de l'application
3. exécuter les programmes sur différentes machines (impératif)
4. réaliser des tests mettant en évidence quelques propriétés de TCP.

Notations et rappel :
Le protocole de transport **TCP** permet de réaliser des communications en mode connecté. Un message envoyé en **TCP** est transféré/acheminé sous forme de flux d'octets. Enfin, **TCP** gère la duplication et la remise dans l'ordre des paquets à leur réception.

1 Premiers échanges

L'objectif de cet exercice est de mettre en oeuvre un simple échange de données entre un programme client et un programme serveur. Il s'agit d'une base avant de traiter des scénarios mettant en évidence le comportement de TCP via les exercices suivants.
L'idée est d'écrire :

— un programme serveur qui attend un message, sous forme de chaîne de caractères, envoyé par un client, affiche ce message, renvoie au client la taille du message reçu (en nombre d'octets), termine le traitement de ce client et termine (un seul client est traité par le serveur).
— un programme client qui demande à l'utilisateur de saisir une chaîne de caractères au clavier (taille max 200 octets), envoie cette chaîne au serveur, reçoit un entier de la part du serveur et compare la valeur de cet entier avec la taille du message envoyé. L'objectif est d'obtenir des valeurs identiques pour toute exécution de votre application.

Pour vous aider, votre point de départ sera le code source du précédent TP.

Une fois les deux programmes écrits entièrement, réaliser plusieurs tests d'exécution et s'assurer de l'affichage correct des chaînes de caractères coté serveur. Il est aussi attendu de minimiser le nombre d'octets échangés (réduits à des données utiles).

Observer et analyser ce qui se produit à l'exécution si :

— vous lancez le serveur, suspendez son exécution avant l'appel à accept(...) et lancez le client. Pour suspendre l'exécution, demandez une saisie d'un caractère ou autre au clavier avant l'appel accept(..). Poursuivre l'exécution du serveur une fois avoir analysé le comportement du client.
— vous faites la même chose mais en lançant 7 clients avant de reprendre la suite de l'exécution du serveur. Que se passe t-il pour l'ensemble des clients avant et après la reprise de l'exécution du serveur.

2 Comprendre le format de transmission en TCP

L'objectif de cet exercice est de mettre en évidence des propriétés du protocole TCP vues en cours et de mettre en oeuvre des communications correctes.

Etape 1

Ecrire deux programmes :

— un programme client qui demande à un utilisateur de saisir une chaîne de caractères, envoie successivement deux fois cette chaîne (en effectuant deux appels de la fonction send(...) et sans inclure le caractère de fin de chaîne '\0') et affiche le nombre total d'octets effectivement envoyés. La taille de la chaîne de caractères saisie au clavier ne dépassera pas 1500 caractères.
— un programme serveur qui reçoit une suite d'octets de taille maximum 4000 caractères (un seul appel à la fonction recv(...), ajoute le caractère '\0' à la fin du message reçu, affiche le message reçu et le nombre d'octets effectivement reçus. Le serveur ne fait donc qu'une seule réception.

Exécuter vos programmes (dans la mesure du possible, sur deux machines différentes) en s'assurant que les deux envois ont été exécutés avant que le serveur ne soit en réception (ajouter une saisie au clavier avant la réception). Qu'observez vous ? En particulier, le serveur a t-il extrait un, deux ou partie des messages envoyés par le client ? Discutez du résultat avec votre chargé(e) de TD et si nécessaire, corrigez votre programme.

Etape 2

- Modifier le programme client pour ajouter le caractère '\0' lors de chaque envoi.
- Modifier le protocole d'échange entre votre client et votre serveur pour que le serveur ait connaissance de la taille d'un message à recevoir.
- Tester à nouveau vos programmes et observer la différence avec la première étape.

A ce stade, lorsque le serveur effectue un appel à recv(...) pour recevoir une chaîne de caractères, il recevra la chaîne en entier ou en partie (expliquer pourquoi). Discutez du comportement obtenu avec votre chargé(e) de TD.

Etape 3

Nous allons maintenant observer le comportement avec plusieurs envois et réceptions successifs. Coté serveur, il n'est plus nécessaire d'afficher les messages reçus ni de stocker tous les messages reçus : une nouvelle réception peut écraser un précédent message reçu.

- Modifier le programme client pour que la chaîne de caractères soit envoyée en boucle. Le nombre d'itérations de la boucle sera un paramètre de votre programme. A la sortie de la boucle, le client doit afficher le nombre total d'octets envoyés depuis la première émission (peut importe le type des messages). Il doit afficher aussi le nombre total d'octets supposés être envoyés et le nombre total d'appels de la fonction send(...).
- Modifier le programme serveur pour qu'il puisse recevoir les chaînes de caractères envoyées par le client (à l'aide d'une boucle infinie). Le serveur affichera à chaque réception (après chaque appel à recv(...) le nombre total d'octets reçus par le client depuis la première réception (peu importe le type des messages) et le nombre total d'appels effectués de recv(...).

Exécuter les deux programmes (sur deux machines différentes) en faisant varier la taille de la chaînes de caractères saisie et le nombre de messages à envoyer (nombre d'itérations) de quelques uns à plusieurs centaines.

Qu'observez vous ? En particulier :

- Le nombre total d'octets effectivement envoyés est-il toujours égal au nombre d'octets effectivement reçus ? Si ce n'est pas le cas, expliquer les raisons de ce résultat et les conséquences avant d'en discuter avec votre chargé(e) de TD. Il est rappelé qu'il n'y a pas de perte de paquets en TCP ni de duplication à la réception par la couche application.
- Le nombre total d'appels à la fonction send(...) est-il toujours égal au nombre total d'appels à la fonction recv(...) ?
- Le nombre d'octets à envoyer est-il toujours égal aux nombre d'octets effectivement envoyés ? Si ce n'est pas le cas, quelles sont les conséquences ? Discuter avec votre chargé(e) de TD.

Etape 4

Modifier vos programmes de manière à avoir une exécution permettant de remplir le buffer de réception du serveur et le buffer d'envoi du client. Que se passe-t-il lorsque les deux buffers sont pleins ?

3 Envoyer et recevoir un message en TCP : Une bonne méthode

Pour gérer les situations vues dans les exercices précédents, il est souvent (voir toujours) indispensable, pour échanger UN message, de mettre en place les fameuses boucles d'envoi et de réception discutées en cours.

- Ecrire une fonction **int sendTCP(int sock, char * msg, int sizeMsg)** qui dépose les sizeMsg octets, se trouvant à l'adresse msg, dans le buffer d'envoi de la socket ayant le descripteur de fichier sock. La fonction retourne 1 si le dépôt des sizeMsg octets a réussi, 0 si la socket a été fermée par la couche transport, -1 en cas d'erreur. Nous supposons un comportement par défaut de la fonction send(..) à utiliser (option à 0).

— Ecrire une fonction **int recvTCP(int sock, char * msg, int sizeMsg)** qui extrait sizeMsg octets du buffer de réception de la socket ayant le descripteur de fichier sock pour les stocker à l'adresse msg. La fonction retourne 1 si la lecture des sizeMsg octets a réussi, 0 si la socket a été fermée par la couche transport, -1 en cas d'erreur. Nous supposons un comportement par défaut de la fonction recv(..) à utiliser (option à 0).

Modifier les programmes clients et serveurs obtenus à l'étape 3 de l'exercice 2 pour utiliser ces nouvelles fonctions à la place des fonctions send(..) et recv(..). Ces nouvelles fonctions sont à utiliser peut importe le type des messages à échanger.

Exécutez vos programmes et observez le comportement (mêmes questions qu'à l'étape 3 de l'exercice 2).

4 Traiter plusieurs clients

— Exécuter à nouveau l'application de l'exercice précédent, sauf qu'avant de saisir la chaîne de caractères coté client, lancer un second client. Que se passe t-il ? Poursuivre l'exécution et expliquer à nouveau ce qui se produit.

— Modifier le programme serveur pour qu'il soit itératif. Lancer l'application avec plusieurs clients. Ce type de serveur est -il approprié ?

— Modifier à nouveau le programme serveur pour qu'il soit concurrent. Relancer l'application avec plusieurs clients. Que pensez vous de cette nouvelle version en comparaison avec un serveur itératif ?