Objectifs : Maîtriser les bases de la mise en place de programmes dans cette UE et la mise en place d'échanges entre un programme serveur et un programme client.

Quelques points à retenir et à appliquer :

1. prendre le temps de réflexion avant de programmer.
2. ne pas coder en dur des données exemples (privilégier le passage de paramètres, des saisies au clavier ou des lectures depuis des fichiers). Se poser la question : dois je modifier mon programme et le recompiler pour tester différents exemples, pour l'exécuter sur une autre machine ou pour qu'un autre utilisateur puisse réutiliser mon code ? Si la réponse est oui, vous faites fausse route et devez corriger vos habitudes dès à présent !
3. éviter les variables et instructions inutiles (lire et comprendre ce que vous faites).
4. enrichir un programme avec des traces d'exécution.
5. développer et tester vos programmes étape par étape.
6. si un programme est réutilisé dans un exercice, penser à créer une copie.

1 Compiler et passer des paramètres

Télécharger les fichiers fournis depuis Moodle. Un squelette de base est fourni pour la mise en oeuvre des programmes du TP. Vous y trouverez :

1. un squelette pour un programme client
2. un squelette pour un programme serveur
3. un Makefile pour compiler.

Vous êtes encouragés pour la suite des TP à adopter une structuration semblable à celle du répertoire fourni, en particulier pour des applications impliquant plusieurs programmes / fichiers sources.

Avant de lire le contenu des fichiers

1. compiler avec la commande make
2. lancer le programme serveur et relancer le si nécessaire pour aller plus loin dans l'exécution.
3. lancer le programme client et relancer le si nécessaire pour aller plus loin dans l'exécution.

A ce stade, vous devez comprendre que la première étape qui est réalisée lorsque un code est fourni (c'est exactement la même chose pour un correcteur lorsqu'il évalue un travail pratique), est de compiler et d'exécuter les programmes. Tous les éléments permettant de lancer correctement un programme doivent être fournis. Ici, ces éléments sont les paramètres demandés.

Vous êtes maintenant invités à lire le contenu des fichiers fournis.

2 Votre premier programme client

A présent, un serveur est en cours d'exécution sur une machine de la FdS. Il attend des messages en provenance de clients. La sortie standard du serveur est projetée tout le long de cet exercice et permettra de notifier une réception d'un message de votre client et de relever d'éventuelles erreurs / incohérences dans les échanges.

L'objectif est alors d'écrire un programme client. Ce dernier demande en paramètre, l'adresse (IP + numéro de port) de la socket du serveur (fournie par l'intervenant(e)) et un numéro de port pour la socket de votre client. Les étapes à réaliser dans votre programme sont données en commentaire dans le squelette :

1. Après avoir désigner la socket du serveur, demander à l'utilisateur de saisir une chaîne de caractères (de taille max 100 caractères). Pour la saisie, vous pouvez utiliser la fonction scanf(...) ou fgets(...) (qui permet de saisir des chaînes avec des espaces). Vous pouvez exceptionnellement utiliser les moyens d'entrée/sortie du C++ (cin et cout).
2. envoyer la chaîne saisie à l'aide de la fonction sendto(...). Traiter les valeurs de retour possibles de cette fonction (voir le cours ou, encore mieux, la documentation via la commande man). Tester votre programme.
3. recevoir un entier à l'aide de la fonction recvfrom(...) et l'afficher. Traiter les valeurs de retour possibles de cette fonction (voir le cours ou la documentation). Tester votre programme.

4. afficher l'adresse de la socket de l'expéditeur du message reçu à partir de l'adresse fournie par la fonction recvfrom(...) et s'assurer que cette adresse correspond bien à celle passée en paramètres. Rappel : l'adresse d'une socket inclut l'IP et le numéro de port.

A ce stade du TP, vous devez avoir compris les bases pour un échange de messages. Remarque : lors d'un échange, n'envoyer que des données utiles.

3 Programmer le serveur

A vous maintenant de programmer le serveur communiquant avec votre client. Ce serveur doit :
- attendre un message, sous forme de chaîne de caractères, affiche ce message et l'adresse de la socket du client. Tester votre programme en exécutant le serveur et le client.
- répondre au client en lui envoyant la taille (en nombre d'octets) du message qu'il vient de recevoir. Cette taille est un entier. Tester.

Dans un premier temps, vous pouvez lancer vos programmes sur une même machine. Ensuite, vous pouvez vous habituer à utiliser des machines distantes à l'aide de la commande ssh. A la FdS, vous pouvez vous connecter, via ssh aux machines suivantes : prodpeda-x2go-focal1, prodpeda-x2go-focal2, ..., prodpeda-x2go-focal6.

4 Tests avancés

Les questions suivantes sont indépendantes (sauf indication contraire) :

Première série :

- Modifier votre programme client pour supprimer le nommage de sa socket. Exécuter votre application. Le nommage supprimé était-il nécessaire ? Pourquoi ?
- Modifier le client pour qu'il envoie un tableau d'entiers (que vous pouvez initialiser à votre convenance) à la place de la chaîne de caractères saisie. Que faut-il modifier coté serveur pour que votre application s'exécute correctement ? Le serveur répond toujours en envoyant le nombre d'octets effectivement reçus.
- Faire pareil, mais cette fois, il est question d'envoyer la structure sockaddr_in qui désigne la socket du serveur.

Remarque : vous pouvez tout à fait garder les échanges existants et en ajouter, au lieu de remplacer l'existant.

Deuxième série :
- Que se passe-t-il si le client est lancé avant le serveur ?
- Que se passe-t-il si le client envoie un message juste avant le nommage coté serveur ? Pour réaliser ce test, modifier le serveur pour ajouter une saisie juste avant le nommage, puis lancer le serveur qui sera suspendu en attente de saisie, lancer le client qui doit dépasser l'étape d'envoi d'un message, puis reprendre l'exécution du serveur en faisant une saisie.
- Que se passe-t-il si le client envoie un message après le nommage coté serveur mais avant l'appel à recvfrom(...) par ce serveur ? Pour réaliser ce test, suivre le même principe que la question précédente.
- Que se passe-t-il si le serveur attend un message de taille plus petite que celle du message envoyé par le client ? Pour tester, il est question de modifier le paramètre taille de la fonction recvfrom(...).
- Modifier vos programmes pour que le client envoie un tableau de doubles. La taille de ce tableau (en nombre d'octets) doit dépasser les 70Ko. Tester. Le serveur répond toujours en envoyant le nombre d'octets effectivement reçus.

Troisième série (dans l'ordre) :
- Consulter la documentation de la fonction getsockname(...)
- Modifier le programme serveur pour que le nommage soit fait automatiquement par la couche transport.
- Utiliser la fonction getsockname(...) pour obtenir le numéro de port choisi par la couche transport.
- Exécutez votre application (attention aux paramètres à passer).

Quatrième série :
- Que se passe-t-il si vous lancez un serveur et deux clients ? Il n'est pas question de programmer un nouveau client mais uniquement de créer deux processus à partir du même programme.
- Que se passe-t-il si vous lancez deux serveurs avec le même numéro de port ? Cette question nécessite un choix explicite du numéro de port du serveur (à passer en paramètre).
- Modifier le programme serveur pour qu'il puisse dialoguer avec plusieurs clients ? Tester.