2ème année DUT Informatique
IUT Aix-en-Provence

TP1 Programmation répartie
Sockets TCP en C

Safa YAHI

Exercice 1 (Serveur daytime)

Considérons le serveur TCP daytime suivant :

```
#include <arpa/inet.h>
#include <sys/socket.h>
#include <unistd.h>
#include <iostream>
#include <string.h>
#include <stdio.h>
#include <time.h>
#include <stdlib.h>

#define NUM_PORT 50013
#define BACKLOG 50
#define NB_CLIENTS 100

using namespace std;

void exitErreur(const char * msg) {
	perror(msg);
	exit( EXIT_FAILURE);

}

int main() {

	int sock_serveur = socket(AF_INET, SOCK_STREAM, 0);

	struct sockaddr_in sockaddr_serveur;

	sockaddr_serveur.sin_family = AF_INET;
	sockaddr_serveur.sin_port = htons(NUM_PORT);
	sockaddr_serveur.sin_addr.s_addr = htonl(INADDR_ANY);

	int yes = 1;
	if (setsockopt(sock_serveur, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(int))
			== -1)
		exitErreur("setsockopt");

	if (bind(sock_serveur, (struct sockaddr *) &sockaddr_serveur,
			sizeof(sockaddr_in)) == -1)
		exitErreur("bind");

	if (listen(sock_serveur, BACKLOG) == -1)
		exitErreur("listen");

	int sock_client;
```

```
	char * msg;
	time_t date;

	cout << "Serveur DayTime lancé  sur le port " << NUM_PORT << endl;

	for (int i = 1; i <= NB_CLIENTS; i++) {

		sock_client = accept(sock_serveur, NULL, NULL);
		if (sock_client == -1)
			exitErreur("accept");

		date = time(NULL);
		msg = ctime(&date);

		if (write(sock_client, msg, strlen(msg)) == -1)
			exitErreur("write");

		close(sock_client);

	}
	close(sock_serveur);
	return 0;
}
```

1) Quelle est l'adresse de la socket d'écoute de ce serveur (adresse IP et numéro de port d'écoute) ?

2) Lancez-le sur votre machine (reccupérez le fichier .cxx depuis Ametice et ne faites pas copiez coller depuis cet énnoncé). Vérifier qu'il utilise bien l'adresse de socket que vous avez identifiée dans la question précédente. Pour celà, utilisez la commande "**netstat -ltn**" ("l" pour "listening", "t" pour "TCP" et "n" pour "numeric").

3) L'utilitaire Telnet peut être, entre autres, utilisé comme un client TCP. Dans un terminal, il suffit de faire "telnet @IP_serveur port_serveur" et on échange ensuite les messages qu'il faut.
Depuis votre machine, envoyez une requête à votre serveur via telnet en utilisant l'adresse 127.0.0.1 (localhost), ensuite en utilisant l'adresse Ethernet IPv4 de votre machine (cette dernière peut etre obtenue avec /sbin/ifconfig).

4) Envoyez-lui une requête, toujours avec telnet, depuis la machine de votre voisin (en utilisant bien sûr l'adresse Ethernet e non pas loopback).

5) L'utilitaire "netcat" ou "nc" peut être aussi utilisé comme client TCP. Testez-le. Notez que contrairement à Telnet, il peut être utilisé comme client UDP.

6) Lancez une deuxième instance du serveur daytime (sans arrêter la première). Que se passe-t-il ? Repérez l'endroit du code du serveur qui nous a permis d'afficher ce message d'erreur. Ce point sera détaillé dans le cours n°2.

7) Essayez maintenant de le lancer sur le port 13 (ou n'importe quel port libre inférieur à 1024). Quel est le problème ?

Notez l'utilité de vérifier à chaque fois la valeur de retour des fonctions de l'API sockets afin

de bien gérer les éventuelles erreurs !

8) Arrêtez le serveur et modifiez-le pour qu'il écoute par une socket liée uniquement à l'adresse Ethernet IPv4 de votre machine et le port 50013. Vous pouvez vous inspirer du serveur du cours n°1.
   Lancez-le et vérifier la nouvelle adresse de socket d'écoute par “netstat -ltn”. Est-ce qu'il répond aux requêtes de type: telnet localhost 50013 ?
   Annulez la modification précédente (en remettat INADDR_ANY).

9) Modifez le serveur pour qu'il affiche, à chaque connexion d'un client, l'adresse IP et le port de ce dernier. Utilisez pour ce faire le deuxième et troisième paramètre de la fonction accept( ) décrits en cours.

10) Refaite la même question en utilisant cette fois-ci la fonction getpeername(). Référez-vous au manuel associé.

Exercice 2 (Client daytime)

=> Quand on veut réccupérer un message en TCP avec read() (ou recv()), un seul appel ne garantit pas toujours de réccupérer la totalité de ce message. Pourquoi ? Quelle est la solution ?

=> Implémentez le client TCP daytime tout en gérant les éventuelles erreurs. Testez-le en local et en réseau avec le serveur daytime précédent.

Exercice 3 (Client Web)

HTTP (Hypertext Transfer Protocol) est un protocole utilisé pour l'échange de données entre clients et serveurs sur le Web depuis 1990. C'est un protocole de la couche application qui opère au dessus du protocole TCP. Les clients HTTP les plus fréquemment utilisés sont les navigateurs Web. On peut citer aussi les aspirateurs de sites. Pour les serveurs HTTP, il existe plusieurs implémentations telles que : Apache HTTP Server ou IIS. Par défaut, un serveur HTTP utilise le port 80. Il existe plusieurs versions du protocole HTTP : HTTP/0.9, HTTP/1.0 et HTTP/1.1 et HTTP/2. Ces deux derniers sont des standards.

Pour simplifier les choses, nous nous interessons ici uniquement au protocole HTTP/0.9 pour sa simplicité.

HTTP/0.9 est la première version de ce protocole. Dans cette version, nous avons un seul type de requêtes, à savoir la requête **GET** qui permet de demander une ressource (page HTML, image, etc). Le dialogue entre le client et le serveur se veut très simple :

- Tout d'abord, le client se connecte au serveur
- Le client envoie ensuite au serveur une requête GET dont la syntaxe est comme suit :

  GET /chemin_fichier\n

  où /chemin/fichier est le chemin du fichier demandé par le client.
- Le serveur répond au client en lui envoyant le contenu du fichier demandé

- Enfin, le serveur ferme la connexion

Ecrivez un client Web pour le protocole HTTP/0.9 en utilisant les sockets TCP en C. Testez-le avec le serveur Web dont l'adresse et le port d'écoute sont indiqués par votre enseignant(e). Il est de même pour la ressource demandée.

Exercice 4 (Chat à tour de rôle entre client et serveur)

Ecrivez en C un **client** et un **serveur TCP** qui s'échangent, à tour de rôle, des messages lus via le clavier. Plus précisément :

- le client se connecte au serveur,
- lui envoie un premier message lu depuis le clavier,
- le serveur lui répond par un autre message lu depuis le clavier,
- le client lui envoie un nouveau message et ainsi de suite
- jusqu'à ce que le client envoie le message “bye”.

On suppose que le serveur traite les clients séquentiellement et non pas “simultanément”.

Testez cette application pour discuter avec vos voisins.

Exercice 5 (Transfert de fichiers en réseau) [Facultatif]

Le but de cet exercice est d'implémenter une application client/serveur qui permet le transfert de fichiers en réseau du serveur vers le client. Dans cette application, quand le client se connecte au serveur, il peut lui envoyer trois types de requêtes :

- lister les fichiers qui peuvent être téléchargés depuis le serveur
- reccuperer un fichier particulier
- et fermer la connexion.

Le protocole que l'on veut utiliser ici est beaucoup plus simple que le protocle FTP. Par exemple, il y a moins de requêtes, il n'y a pas d'authentifcation et il n'y a pas de distinction entre connexion de données et connexion de contrôle (donc pas de mode passif et actif).

=> Spécifiez plus précisément le protocole de cet exercice (format des messages échangés) et implémentez cette application.