

TP1 Programmation répartie Sockets TCP en C

Safa YAHY

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 sa socket d'écoute (adresse IP et numéro de port) ?
- 2) Lancez-le sur votre machine. Vérifier qu'il utilise bien l'adresse de socket que vous avez identifiée dans la question précédente, et ce en utilisant la commande “**netstat -ltn**” (“l” pour “listening”, “t” pour “TCP” et “n” pour “numeric”).
- 3) Envoyez-lui une requête depuis votre machine via un client telnet (ou nc) en utilisant l'adresse 127.0.0.1, ensuite en utilisant l'adresse Ethernet IPv4 de votre machine.
- 4) Envoyez-lui une requête, toujours avec telnet, depuis la machine de votre voisin. Est-ce que vous pouvez utiliser l'adresse loopback dans ce cas ?
- 5) Lancez une deuxième instance du serveur daytime (sans arrêter la première). Que déduisez-vous ? Repérer l'endroit du code du serveur qui nous a permis d'afficher ce message d'erreur.
- 6) 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.

- 7) 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. Lancez-le et vérifiez la nouvelle adresse de socket d'écoute par “**netstat -ltn**”. Est-ce qu'il répond aux requêtes de type: telnet localhost 50013 ?

Exercice 2 (Client daytime)

Implémentez le client TCP daytime tout en gérant les éventuelles erreurs de connexion. Testez-le en

réseau avec le serveur daytime précédent.

Notez que la fonction `read()` n'alloue pas de l'espace mémoire pour le buffer qui contiendra le message reçu. Le pointeur qui est donné comme paramètre doit pointer vers un espace déjà alloué (d'une manière statique ou dynamique). Remplacer `char msg[TAILLE_BUFFER]` par `char * msg`. Quel est le message d'erreur obtenu ? Annulez cette modification.

Rappelons que le port (local) de la socket du client est choisi dynamiquement par le SE. D'une exécution à une autre, il change de valeur. Vérifiez cela en appelant la fonction `getsockname` juste après la connexion au serveur (regardez le man de cette fonction).

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 qui constitue un standard.

Pour simplifier les choses, nous nous intéressons ici uniquement au protocole HTTP/0.9.

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 vous est indiquée par votre enseignant(e) sur le port 80 afin d'avoir la page “/index.html”.

Exercice 4 (Discussion 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”. En plus, on suppose que le serveur traite les clients séquentiellement et non pas “simultanément”. Testez-le pour discuter avec vos voisins.

Exercice 5 (Refuser un client par rapport à son adresse IP)

- Modifiez le serveur de discussion à tour de rôle pour qu'il affiche (sur sa console), 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.
- Refaite la même question en utilisant cette fois-ci la fonction `getpeername()`. Reférez-vous au manuel associé.
- Modifiez le serveur de telle sorte qu'il ne réponde pas à une adresse IP de votre choix.

Exercice 6 (Transfert de fichiers en réseau)

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
- récupérer un fichier particulier
- et ferme la connexion.

Le protocole que l'on veut utiliser ici est beaucoup plus simple que le protocole FTP. Par exemple, il y a moins de requêtes, il n'y a pas d'authentification 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.