COMPTE RENDU TRAVAUX PRATIQUES DE LANGAGE C

TP 4: Communication réseaux - Protocole TCP

Etudiant : Assane Thiao

Formation : Télécommunications et Réseaux

Niveau : ING1

Année : 2024/2025

Objectif

L'objectif de ce TP était de se familiariser avec le protocole TCP en écrivant un client et un serveur TCP, capables d'envoyer et de recevoir des messages. Le but était aussi d'explorer la gestion de multiples connexions clients via le protocole TCP.

1. Client TCP

a) Connexion au serveur

Le premier objectif était d'écrire un client TCP qui se connecte au serveur local sur le port 8080 et envoie un message ASCII au serveur. Le client TCP suit plusieurs étapes pour établir la connexion et envoyer un message :

1. **Création du socket** : Utilisation de la fonction `socket()` pour créer un socket TCP.
2. **Configuration du socket** : Le socket est configuré avec une adresse et un port à l'aide de la structure `SOCKADDR_IN`.
3. **Connexion au serveur** : Utilisation de la fonction `connect()` pour établir la connexion au serveur.
4. **Envoi d'un message** : Utilisation de la fonction `send()` pour envoyer le message au serveur.

Après avoir exécuté le client, j'ai vérifié que la connexion était bien acceptée par le serveur et que le message envoyé était correctement reçu et affiché par celui-ci.

b) Réception du message du serveur

J'ai modifié le code pour que le client puisse recevoir une réponse du serveur après l'envoi du message. La fonction `recv()` a été utilisée pour récupérer le message envoyé par le serveur et l'afficher sur la console du client.

Le serveur a répondu avec un message de confirmation ou une simple réponse comme "Message reçu".

c) Choix dynamique du port

J'ai ajouté la possibilité de spécifier un port d'envoi sur la ligne de commande (par exemple, `client.exe 8080`). Si aucun port n'est spécifié, le port par défaut (8085) est utilisé. J'ai testé ce mécanisme en choisissant des ports valides et invalides. Si le port était incorrect, le client ne pouvait pas se connecter et le message n'arrivait pas au serveur.

d) Code du client

```c
/* Assane Thiao 14/03/2025 */

#include "network.h"  /* Inclure le fichier d'en-tête personnalisé */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define BUFFER_SIZE 1024

int main(int argc, char* argv[]) {
    SOCKET sock;
    struct sockaddr_in server_addr;
    char buffer[BUFFER_SIZE];

    int port = 8085;

    if (argc == 2) {
        port = atoi(argv[1]);
    }

    initNetwork();

    sock = socket(AF_INET, SOCK_STREAM, 0);
    if (sock == INVALID_SOCKET) {
        printf("Erreur lors de la creation du socket\n");
        cleanNetwork();
        return EXIT_FAILURE;
    }

    memset(&server_addr, 0, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(port);
    server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");  /* Adresse localhost */

    if (connect(sock, (struct sockaddr*)&server_addr, sizeof(server_addr)) == SOCKET_ERROR) {
        printf("Erreur lors de la connexion au serveur\n");
        closesocket(sock);
        cleanNetwork();
        return EXIT_FAILURE;
    }

    printf("Connecte au serveur sur le port %d\n", port);
```

```c
    printf("Connecte au serveur sur le port %d\n", port);

    while (1) {
        printf("Entrez un message a envoyer au serveur : ");
        fgets(buffer, sizeof(buffer), stdin);
        buffer[strcspn(buffer, "\n")] = 0;  /* Enlever le '\n' à la fin du message */

        if (send(sock, buffer, strlen(buffer), 0) == SOCKET_ERROR) {
            printf("Erreur lors de l'envoi du message\n");
            closesocket(sock);
            cleanNetwork();
            return EXIT_FAILURE;
        }

        printf("Message envoye au serveur : %s\n", buffer);

        if (strcmp(buffer, "CLIENT:QUIT") == 0) {
            break;
        }

        int recv_len = recv(sock, buffer, sizeof(buffer) - 1, 0);
        if (recv_len > 0) {
            buffer[recv_len] = '\0';  /* Ajouter le caractère de fin de chaîne */
            printf("Reponse du serveur : %s\n", buffer);
        }
        else {
            printf("Erreur ou fermeture de la connexion du serveur\n");
            break;
        }
    }

    closesocket(sock);
    cleanNetwork();
    return 0;
}
```

2. Serveur TCP

a) Réception du message

J'ai écrit le serveur en utilisant une boucle infinie pour accepter plusieurs connexions clients. Le serveur crée un socket, configure la structure `SOCKADDR_IN`, puis lie ce socket au port 8080 avec la fonction `bind()`. Ensuite, il écoute les connexions avec la fonction `listen()` et accepte les connexions entrantes avec `accept()`.

Lorsqu'un message est reçu d'un client, le serveur utilise la fonction `recv()` pour récupérer et afficher ce message.

b) Envoi de la réponse au client

Après avoir reçu le message du client, le serveur répond avec un message. J'ai utilisé la fonction `send()` pour envoyer un message de retour au client, similaire au comportement du programme `serverTCP.exe`.

c) Fermeture du serveur sur "SERVER:QUIT"

J'ai modifié le serveur pour qu'il ferme la connexion lorsqu'il reçoit le message `"SERVER:QUIT"`. Cette fonctionnalité a été testée avec le client, et le serveur s'est bien arrêté après avoir reçu cette commande.

d) Code du serveur

```c
/* Assane Thiao 14/03/2025 */

#include "network.h"  /* Inclure le fichier d'en-tête personnalisé */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define PORT 8085
#define BUFFER_SIZE 1024
#define MAX_CLIENTS 2

int main() {
    SOCKET sock, client1 = -1, client2 = -1;
    struct sockaddr_in server_addr, client_addr;
    fd_set readfds;
    int addr_len = sizeof(client_addr);
    char buffer[BUFFER_SIZE];

    /* Initialisation du réseau */
    initNetwork();

    /* Création du socket */
    sock = socket(AF_INET, SOCK_STREAM, 0);
    if (sock == INVALID_SOCKET) {
        printf("Erreur lors de la creation du socket\n");
        cleanNetwork();
        return EXIT_FAILURE;
    }

    /* Configuration de l'adresse du serveur */
    memset(&server_addr, 0, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(PORT);
    server_addr.sin_addr.s_addr = INADDR_ANY;

    /* Liaison du socket */
    if (bind(sock, (struct sockaddr*)&server_addr, sizeof(server_addr)) == SOCKET_ERROR) {
        printf("Erreur lors de la liaison du socket\n");
        closesocket(sock);
        cleanNetwork();
        return EXIT_FAILURE;
    }
```

```c
    /* Mise en écoute du socket */
    if (listen(sock, MAX_CLIENTS) == SOCKET_ERROR) {
        printf("Erreur lors de la mise en ecoute\n");
        closesocket(sock);
        cleanNetwork();
        return EXIT_FAILURE;
    }
    printf("\n");
    printf(" ********************************************************************************\n");
    printf("|                                                                              |\n");
    printf("|  Port par defaut c'etait 8080, mais ce port est utilise par httpd.exe,       |\n");
    printf("|  et sur la machine de l'ecole on a pas les droits pour stopper ce processus. |\n");
    printf("|                                                                              |\n");
    printf(" ********************************************************************************\n");
    printf("\n");
    printf("Serveur en ecoute sur le port %d...\n", PORT);

    while (1) {
        FD_ZERO(&readfds);
        FD_SET(sock, &readfds);
        if (client1 != -1) FD_SET(client1, &readfds);
        if (client2 != -1) FD_SET(client2, &readfds);

        int max_sock = sock;
        if (client1 > max_sock) max_sock = client1;
        if (client2 > max_sock) max_sock = client2;

        /* Utilisation de select pour gérer plusieurs clients simultanément */
        if (select(max_sock + 1, &readfds, NULL, NULL, NULL) < 0) {
            printf("Erreur lors de l'utilisation de select\n");
            break;
        }

        /* Gestion des connexions entrantes */
        if (FD_ISSET(sock, &readfds)) {
            SOCKET new_client = accept(sock, (struct sockaddr*)&client_addr, &addr_len);
            if (new_client == INVALID_SOCKET) {
                printf("Erreur lors de l'acceptation du client\n");
                continue;
            }
```

```
            if (client1 == -1) {
                client1 = new_client;
                printf("Client 1 connecte\n");
            }
            else if (client2 == -1) {
                client2 = new_client;
                printf("Client 2 connecte\n");
            }
            else {
                printf("Nombre maximal de clients atteint\n");
                closesocket(new_client);
            }
        }

        /* Gestion des données reçues des clients */
        SOCKET clients[] = { client1, client2 };
        int i;
        for (i = 0; i < MAX_CLIENTS; i++) {
            SOCKET client = clients[i];
            if (client != -1 && FD_ISSET(client, &readfds)) {
                int recv_len = recv(client, buffer, BUFFER_SIZE - 1, 0);
                if (recv_len <= 0) {
                    printf("Client deconnecte\n");
                    closesocket(client);
                    if (client == client1) client1 = -1;
                    if (client == client2) client2 = -1;
                    continue;
                }

                buffer[recv_len] = '\0';
                printf("Message recu : %s\n", buffer);

                /* Si le message reçu est "SERVER:QUIT", on ferme tous les sockets et on arrête le serveur */
                if (strcmp(buffer, "SERVER:QUIT") == 0) {
                    printf("Fermeture du serveur\n");
                    closesocket(client1);
                    closesocket(client2);
                    closesocket(sock);
                    cleanNetwork();
                    return 0;
                }
                /* Si le message reçu est "CLIENT:QUIT", on ferme la connexion du client correspondant */
```

```
                /* Si le message reçu est "CLIENT:QUIT", on ferme la connexion du client correspondant */
                else if (strcmp(buffer, "CLIENT:QUIT") == 0) {
                    printf("Fermeture de la connexion client\n");
                    closesocket(client);
                    if (client == client1) client1 = -1;
                    if (client == client2) client2 = -1;
                }
                else {
                    /* Renvoi du même message reçu en réponse */
                    send(client, buffer, strlen(buffer), 0);
                }
            }
        }
    }

    closesocket(sock);
    cleanNetwork();
    return 0;
}
```

3. Gestion de plusieurs clients avec select()

a) Utilisation de select() pour gérer plusieurs clients

La question facultative 3 consistait à gérer plusieurs clients simultanément en utilisant la fonction `select()`. Pour ce faire, trois sockets ont été utilisés :

- `sock` pour le serveur.
- `client1` et `client2` pour les clients connectés.

Le comportement attendu était le suivant :

1. **Attente de connexion** : Le serveur attend les connexions des clients via le socket `sock` en utilisant la fonction `select()`.
2. **Gestion des connexions** : Lorsqu'un client se connecte, il est affecté à un socket client disponible. Si les deux sockets clients sont déjà connectés, le serveur envoie un message d'avertissement et ferme la connexion.
3. **Traitement des messages** : Lorsqu'un client envoie un message, le serveur le traite et envoie une réponse. Si la commande `"CLIENT:QUIT"` est reçue, le serveur ferme le socket du client correspondant. Si la commande `"SERVER:QUIT"` est reçue, tous les sockets sont fermés et le serveur s'arrête.

b) Comportement observé

J'ai testé le serveur avec plusieurs clients simultanés et observé le comportement attendu : chaque client était connecté et traité correctement, et le serveur a pu gérer plusieurs clients en même temps.

4) Test et résultat

Coté serveur:

```
P:\TPlangageC\source\repos\repos4\TraveauxPratiques\Communication réseau - Protocole TCP>gcc serveur.c -o serveur.exe
2_32

P:\TPlangageC\source\repos\repos4\TraveauxPratiques\Communication réseau - Protocole TCP>.\serveur.exe

 ********************************************************************************
|                                                                                |
|  Port par defaut c'etait 8080, mais ce port est utilise par httpd.exe,         |
|  et sur la machine de l'ecole on a pas les droits pour stopper ce processus.   |
|                                                                                |
 ********************************************************************************

Serveur en ecoute sur le port 8085...
Client 1 connecte
Client 2 connecte
Nombre maximal de clients atteint
Message recu : client1
Message recu : Client2
Message recu : je peux envoyer autant de message
Message recu : jenvoie CLIENT:QUIT pour me deconnecter
Message recu : jenvoie SERVER:QUIT pour fermer le serveur et me deconnecter
Message recu : CLIENT:QUIT
Fermeture de la connexion client
Message recu : allons y
Message recu : SERVER:QUIT
Fermeture du serveur

P:\TPlangageC\source\repos\repos4\TraveauxPratiques\Communication réseau - Protocole TCP>
```

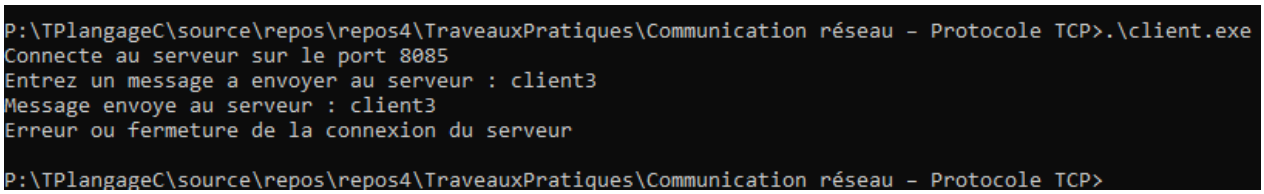
Coté client 1:

```
P:\TPlangageC\source\repos\repos4\TraveauxPratiques\Communication réseau - Protocole TCP>.\client.exe
Connecte au serveur sur le port 8085
Entrez un message a envoyer au serveur : client1
Message envoye au serveur : client1
Reponse du serveur : client1
Entrez un message a envoyer au serveur : jenvoie SERVER:QUIT pour fermer le serveur et me deconnecter
Message envoye au serveur : jenvoie SERVER:QUIT pour fermer le serveur et me deconnecter
Reponse du serveur : jenvoie SERVER:QUIT pour fermer le serveur et me deconnecter
Entrez un message a envoyer au serveur : allons y
Message envoye au serveur : allons y
Reponse du serveur : allons y
Entrez un message a envoyer au serveur : SERVER:QUIT
Message envoye au serveur : SERVER:QUIT
Erreur ou fermeture de la connexion du serveur

P:\TPlangageC\source\repos\repos4\TraveauxPratiques\Communication réseau - Protocole TCP>
```

Coté client 2:

```
P:\TPlangageC\source\repos\repos4\TraveauxPratiques\Communication réseau - Protocole TCP>.\client.exe
Connecte au serveur sur le port 8085
Entrez un message a envoyer au serveur : Client2
Message envoye au serveur : Client2
Reponse du serveur : Client2
Entrez un message a envoyer au serveur : je peux envoyer autant de message
Message envoye au serveur : je peux envoyer autant de message
Reponse du serveur : je peux envoyer autant de message
Entrez un message a envoyer au serveur : jenvoie CLIENT:QUIT pour me deconnecter
Message envoye au serveur : jenvoie CLIENT:QUIT pour me deconnecter
Reponse du serveur : jenvoie CLIENT:QUIT pour me deconnecter
Entrez un message a envoyer au serveur : CLIENT:QUIT
Message envoye au serveur : CLIENT:QUIT

P:\TPlangageC\source\repos\repos4\TraveauxPratiques\Communication réseau - Protocole TCP>
```

Coté client 3:



Conclusion

Ce TP a permis de comprendre les principes de la communication réseau en utilisant le protocole TCP. Nous avons appris à :

- Créer un client et un serveur TCP.
- Gérer l'envoi et la réception de messages.
- Ajouter des fonctionnalités telles que la gestion des ports et la fermeture propre du serveur.
- Implémenter la gestion simultanée de plusieurs clients à l'aide de la fonction `select()`.