Université de Versailles St-Quentin-En-Yvelines

Rapport du projet AISE

Réaliser par:
BEGRICHE Massiva
BELKHIRNadine

Année universitaire 2020-2021

Contents

Introduction:

L'objectif de ces notes est de présenter la communication inter processus sur un réseau en utilisant les outils socket et thread.

Les sockets servent à communiquer entre deux hôtes appelés Client / Serveur à l'aide d'une adresse IP et d'un port ; ces sockets permettront de gérer des flux entrant et sortant afin d'assurer une communication entre les deux (le client et le serveur), soit de manière fiable à l'aide du protocole TCP/IP, soit non fiable mais plus rapide avec le protocole UDP.

Un socket est donc un point de terminaison d'une communication bidirectionnelle, c'est-à-dire entre un client et un serveur en cours d'exécution sur un réseau donné. Les deux sont liés par un même numéro de port TCP de sorte que la couche puisse identifier la demande de partage de données.

Ensuite, un thread est créé chaque fois qu'un client arrive. Pour qu'il puisse gérer au mieux le client dont il a la charge, chaque thread doit connaître le descripteur de la socket avec laquelle il va pouvoir échanger des données avec le client.

Nous allons étudier comment on a implémenté l'architecture client/serveur avec le mode TCP/IP a l'aide des sockets et thread.

1 Structure de travail:

Notre mini-projet consiste a développer un outil de monitoring permettant de récupérer toutes les métriques de performances de plusieurs machines.

On a fait dans un premier lieu une partie 1 qui fait la communication entre un client et un serveur en utilisant les socket comme vu en cour , Puis dans une deuxième partie ou on a implémenter l'architecture du multi-threading en utilisant les socket ainsi que les thread.

Nous allons détailler dans ce qui suit la différente partie de notre implémentation.

1.1 Partie 1: réaliser un socket TCP entre un client et un serveur:

la première partie de notre travail consiste a réaliser un socket TCP/IP permettant au client de récupérer un fichier a partir d'un serveur contenant les informations sur le système.

La communication entre le serveur/client se fait de la manière suivante:

- Un serveur fonctionne sur une machine bien définie et est lié à un numéro de port spécifique. Le serveur se met simplement à l'écoute d'un client, qui demande une connexion.

- -Quant au client, celui-ci connaît le nom de la machine sur laquelle le serveur est en exécution et le numéro de port sur lequel il écoute. Le client va demander une connexion au serveur en s'identifiant avec son adresse IP ainsi que le numéro de port qui lui est lié.

- Une fois la connexion établie et les sockets en possession, il est possible de récupérer le flux d'entrée et de sortie de la connexion TCP vers le serveur.

Dans notre cas nous avons utiliser le protocole connecté TCP/IP dont le principe est suivant:

- Le serveur lie une socket d'écoute sur un certain port bien précis et appelle un service d'attente de connexion de la part d'un client.

- Le client appelle un service pour ouvrir une connexion avec le serveur , et il récupère une socket (associée à un port quelconque par le système).

- Du coté du serveur, le service d'attente de connexion retourne une socket de service (associée à un port quelconque).
  C'est la socket qui permet de dialoguer avec ce client.

- Enfin le client et le serveur communiquent en envoyant et recevant des données via leur socket.

1.1.1 Partie pratique:

Nous allons commencer par inclure les bibliothèques nécessaire du coté client ainsi que serveur:

```
#include <sys/socket.h>

#include <arpa/inet.h>

#include <sys/types.h>

#include <netinet/in.h>

#include <unistd.h>
```

Puis nous allons déclarer un socket de type int comme suite :
int socket(int domain, int type, int protocol);

Dans notre cas nous allons utiliser le protocole TCP/IP la socket sera comme suite:
int socket(AF_INET,SOCK_STREAM,0);

Après avoir déclarer et créé un socket nous allons la paramétrer pour cela nous allons déclarer une structure de type SOCKADDR_IN qui va nous permettre de configurer la connexion, cette structure est définie comme suite:

```
struct sockaddr_in
{
    short      sin_family;
    unsigned short   sin_port;
    struct   in_addr   sin_addr;
    char   sin_zero[8];
};
```

sin_addr.s_addr = htonl(INADDR_ANY); retourne l'adresse IP du serveur afin de pourvoir communiquer avec lui.

sin_port = htons(PORT);le port utiliser pour la communications.

et enfin la famille. AF_INET spécifie que l'on utilise le protocole IP

Établissement de connexion:

1. Socket serveur:

   Une fois la structure remplie, nous allons établir la connexion du serveur:

   • int bind(int sockfd, struct sockaddr *my_addr, socklen_t addrlen); lie un socket avec une structure sockaddr. Maintenant que toute les informations son données, la socket se mettra en mode écoute.

   • int listen(int s, int backlog); le serveur se met en mode écoute en attendant une connexion.

   • int accept(int sock, struct sockaddr *adresse, socklen_t *longueur); Cette fonction accepte la connexion d'un socket pour débuter la communication.

   • Int close(int sock); une fois la communication terminer on fermera la socket.

2. Socket client:

   commençons tout d'abord par créer la socket:

   • int socket(AF_INET,SOCK_STREAM,0);

   puis le client établie une connexion:

   •int connect(int sockfd, const struct *addr, socklen_t addrlen);
   une fois que le serveur accepte la connexion du client, ce dernier pourra envoyé ses requêtes vers le serveur.

   une fois la communication terminer on fermera la socket.

   • Int close(int sock);

1.2 Partie 2:

1.2.1 Client avancé:

Dans cette partie on a réalisé un chat entre un serveur et plusieurs clients.
Le serveur reçois les messages des clients, ce dernier leurs répond.

1.2.2 Gestion de plusieurs clients(multi-clients):

Communication avec plusieurs clients pour le serveur, c'est ce qu'on appelle le multithreading.

Le déroulement de cette partie est le même que la précédente sauf que la communication se fera entre plusieurs clients et un serveur qui répondra a leurs requêtes. Le but est donc de permettre au programme de réaliser plusieurs actions au même moment.

Un même processus peut se décomposer en plusieurs parties, qui vont s'exécuter simultanément en partageant les mêmes données en mémoire. Ces parties se nomment threads.

Dans cette partie nous allons voir comment utiliser les threads avec la bibliothèque pthread.

- **inclusion de la bibliothèque :**
  Nous allons d'abords inclure pthread comme ceci :
  • # include < *pthread.h* >

  Les sources utilisant les threads nécessitent une édition de lien avec la librairie pthread :
  • gcc nomSource.c -lpthread -o nomExecutable

- **Déclarer un thread :**
  Pour pouvoir utiliser notre thread, nous allons tout d'abord déclarer une variable de type pthread_t comme il suit:
  pthread_t thread;

- **Créer un thread**
  Une fois que notre thread est déclaré, il va falloir le lier à une fonction de notre choix, la fonction désignée se déroulera ensuite en parallèle avec le reste de l'application. Pour réaliser cela nous allons utiliser la fonction pthread_create dont le prototype est donné ci-dessous.

  int pthread_create(pthread_t* thread, pthread_attr_t* attr, void*(*start_routine)(void*), void* arg);

  •L'argument thread correspond au thread qui va exécuter la fonction.
  •L'argument attr indique les attributs du thread, ce paramètre ne nous intéresse pas, nous mettrons donc celui-ci à NULL pour que les attributs par défaut soient utilisés.
  •L'argument start_routine correspond à la fonction à exécuter.
  •L'argument arg est un pointeur sur void qui sera passé à la fonction à exécuter. Si vous n'avez aucun paramètre à passer, mettez ce paramètre à NULL.

- **Terminer un thread:**
  Une fois que notre thread est exécuté, il se peut que nous ayons besoin de savoir quand il se termine. La fonction pthread_join va permettre d'attendre la fin du thread c'est à dire la fin de l'exécution de la fonction exécutée par celui-ci.
  Voici le prototype:

  int pthread_join(pthread_t thread, void **thread_return);

  • L'argument thread correspond au thread à attendre.
  • L'argument thread_return est un pointeur sur la valeur de retour du thread.

2 Conclusion:

Dans notre mini-projet on a pu implémenter la communication entre un client et un serveur en utilisant les sockets comme vu en cour, ainsi dans une deuxième partie pour améliorer notre travail on a pu faire la communication entre un serveur et plusieurs clients (le multi threading).Ainsi que l'échange de messages entre eux .