Client / server

Akachar Yassine

3 avril 2020

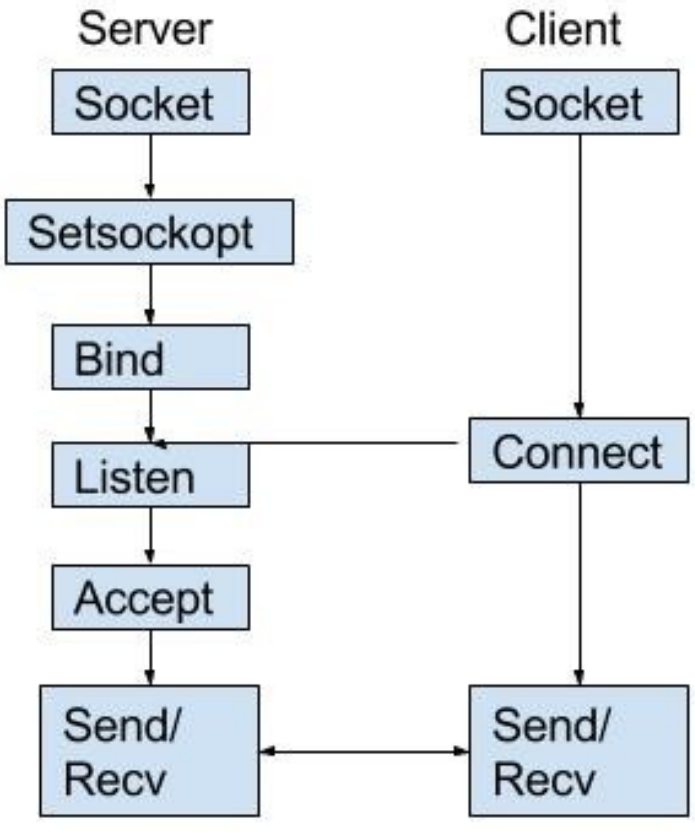
Table des matières

Chapitre 1

Introduction

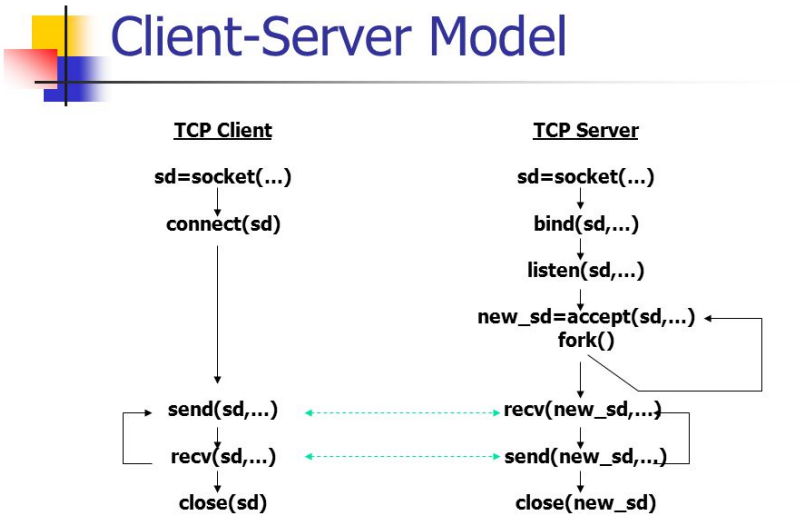
1.1 Librairie socket

L'objectif de ce travail est de créer une application client/serveur afin que plusieurs utilisateurs puissent se connecter et y stocker toutes sortes d'informations. Pour ce faire, il faudra lancer un serveur sur une machine, et au minimum un client sur une autre machine. Pour ce travail, c'est par la librairie socket que l'on passe pour permettre cette connexion.



1.2 Appel system fork

Pour pouvoir assurer la connection avec plusieurs utilisateurs simultanement, on utilise l'appel system fork. De ce fait, nous allons pouvoir dupliquer autant de fois le process qu'il n'y a d'utilisateurs et ainsi leur permettre d'agir "en même temps" tout en ayant un seul et unique serveur.



1.2.1 Comment ca marche ?

L'appel système fork retorune une valeur entière. Pour pouvoir différencier le processus pere du fils il faut regarder la valeur du fork. Si cette valeur est nulle, c'est que nous sommes dans le processus fils, sinon la valeur est égale au pid du fils au quel cas nous sommes dans le processus père.

```
    int main(void)
{
  printf("processus pere, avant le fork\n");
  int pid = fork();
  if (pid == 0)
    printf("processus fils, apres le fork\n");
  else
    printf("processus pere, apres le fork (pid du fils = %d)\n", pid);
  printf("fin de processus\n");
}
```

Ce qui donne :

```
processus pere, avant le fork
processus fils, apres le fork
processus pere, apres le fork (pid du fils = 25532)
fin de processus
fin de processus
```

Chapitre 2

Serveur

2.1 Code source

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <fcntl.h>


int main(int argc, char * argv[]){
// sudo rcSuSEfirewall2 stop
   if(argc != 2){
      printf("Do not forget to enter the addres !\n");
      exit(0);
   }
   int sockfd, ret, newSocket;
   struct sockaddr_in serverAddr;
   struct sockaddr_in newAddr;
   socklen_t addr_size;
   char uip [100];
   pid_t childpid;

   sockfd = socket(PF_INET, SOCK_STREAM, 0);
   if(sockfd < 0){
      printf("[-]Error in connection.\n");
      exit(1);
   }
   printf("[+]Server Socket is created.\n");

   serverAddr.sin_family = AF_INET;
   serverAddr.sin_port = htons(5990);
   if (! inet_aton(argv[1],&serverAddr.sin_addr.s_addr)){
      perror("[-]Error in INET_ATON");
      exit(1);
   }
   ret = bind(sockfd, &serverAddr, sizeof(serverAddr));
   if(ret < 0){
      perror("[-]Error in binding.\n");
```

```
      exit(1);
   }
   printf("[+]Bind to port %d\n", 5990);

   if(listen(sockfd, 10) == 0){
      printf("[+]Listening....\n");
   }else{
      printf("[-]Error in binding.\n");
   }


   while(1){
      newSocket = accept(sockfd, (struct sockaddr*)&newAddr, &addr_size);
      if(newSocket < 0){
         perror("Error on accepting");
         exit(1);
      }
      printf("Connection accepted from %s:%d\n", inet_ntoa(newAddr.sin_addr), ntohs(newAddr.sin_port));

      if((childpid = fork()) == 0){
         close(sockfd);

         while(1){
            int fd;
            int i, count_r, count_w;
            char* bufptr;
            char buffer[1024];
            char buf[1024];
            char filename[1024];
            recv(newSocket, buffer, 1024,0);
            if(strcmp(buffer, ":exit") == 0){
               printf("Disconnected from %s:%d\n", inet_ntoa(newAddr.sin_addr), ntohs(newAddr.sin_port)
               break;
            }else if(strcmp(buffer,":history") == 0){
               memset(buffer, 0, sizeof(buffer));
               sscanf(inet_ntoa(newAddr.sin_addr),"%s",uip);
               strcat(uip,".txt");
               //FILE* f = fopen(uip,"w+");
               //if(f == NULL){  // Créer le fichier local avec le nom indiqué en mode écriture
               // perror("Open");
               //}
               fd = open(uip, O_CREAT |O_WRONLY | O_TRUNC);
               if (fd == -1)
               {
                  perror("File open error");
                  exit(1);
               }
               while((count_r = read(newSocket, buf, 1024))>0)
               {
                  count_w = 0;
                  bufptr = buf;
                  while (count_w < count_r)
                  {
                     count_r -= count_w;
                     bufptr += count_w;
                     count_w = write(fd, bufptr, count_r);
                     if (count_w == -1)
                     {
                        perror("Socket read error");
                        exit(1);
                     }
                  }
                  if(strcmp(bufptr,"#")){
                     break;
                     }
```

```
                }
                close(fd);
                send(newSocket, "Historique enregistré", strlen("Historique enregistrée"), 0);
            }else{
                if(strlen(buffer) < 20){
                   printf("%s: %s\n",inet_ntoa(newAddr.sin_addr),buffer);
                }
                send(newSocket, buffer, strlen(buffer), 0);
                bzero(buffer, sizeof(buffer));
            }

         }
      }

   }

   close(newSocket);


   return 0;
}
```

2.1.1 Variables

— *serverAddr* : représente la structure sockaddr_in contenant les informations du serveur
— *newAddr* : représente la structure sockaddr_in qui contiendra les informations de chaque client (une variable pour chaque client)
— *sockfd* : représente le socket à qui l'on va assigné l'adresse du serveur grâce à la structure sockaddr_in

```
ret = bind(sockfd, &serverAddr, sizeof(serverAddr));
```

— *newSocket* : représentera le socket qui permettra l'échange entre le serveur et le client

```
newSocket = accept(sockfd, (struct sockaddr*)&newAddr, &addr_size);
```

2.1.2 Fork

Concentrons nous sur la partie duplication de process. Grâce au l'appel systeme fork, nous pouvons dupliquer autant de fois notre process ce qui nous permet d'avoir autant de client que nous voulons. Cependant, pour ne pas surcharger le serveur et empêcher que trop de personnes se connectent au serveur et le fasse crasher, nous allons limiter le nombre de client connecté en même temps. Nous allons le spécifier lors de l'appel à la fonction listen. La fonction listen() marque le socket spécifié comme un socket passive, c'est à dire un socket prêt à accepter les demandes de connections (fonction accept citée au dessus).

```
listen(sockfd, 10)
```

Une fois le socket passif, et une connexion acceptée, Il faut pouvoir gérer la connexion de chaque client individuellement tout en maintent le serveur actif. D'où la duplication de process. Comme expliqué précédemment, en vérifiant que le retour de l'appel systeme fork soit égale à 0, on sait que tout ce qui est exécutés entre accolade du if ne sera exécutés que par le processus fils.

```
if((childpid = fork()) == 0){
     ...
}
```

Chapitre 3

Client

3.1 Code source

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <sys/types.h>
#include <arpa/inet.h>

#define PORT 5990

int NbLinesInHistory(){
   FILE *fp;
   int count = 0;  // Line counter (result)
   char c;
   fp = fopen("history.txt", "r");

   // Check if file exists
   if (fp == NULL)
   {
   perror("Open of history");
   return 0;
   }

   // Extract characters from file and store in character c
   for (c = getc(fp); c != EOF; c = getc(fp))
   if (c == '\n') // Increment count if this character is newline
   count = count + 1;

   // Close the file
   fclose(fp);
   return count;
}

int getLastLinesInHistory(){
   char tmp[1024];
   FILE * fp;
   fp = fopen("history.txt", "r");

   while(!feof(fp))
```

```
        fgets(tmp, 1024, fp);

    printf("Fyn A fDP C'EST LAAAAAAAAAAAAAAAAAAAAAAA DERNIÈRE LIGNE\n%s", tmp);
    char * strToken = strtok (tmp," ");
    int * x;
    sscanf(strToken, "%d", &x);
    return x;


}


int main(int argc, char * argv[]){
    if(argc != 2){
        printf("Do not forget to enter the addres !\n");
        exit(0);
    }
    int clientSocket, ret,t,b;
    struct sockaddr_in serverAddr;
    struct in_addr adresse;
    char buffer[1024];
    clientSocket = socket(PF_INET, SOCK_STREAM, 0);
    if(clientSocket < 0){
        perror("[-]Error in connection.\n");
        exit(1);
    }
    printf("[+]Client Socket is created.\n");
    memset(&serverAddr, '\0', sizeof(serverAddr));
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_port = htons(PORT);
    if(!inet_aton(argv[1],&adresse)){
        perror("[-]Error in INET_ATON");
        exit(1);
    }
    serverAddr.sin_addr = adresse;

    ret = connect(clientSocket, &serverAddr, sizeof(serverAddr));
    if(ret == -1){
        perror("[-]Error in connection.\n");
        exit(1);
    }
    printf("[+]Connected to Server.\n");

    while(1){
        printf("\nQue voulez-vous faire?\n");
        printf("\t:history\t pour upload votre historique\n");
        printf("\t:exit\t \t pour se déconnecter \n");
        printf("Client: \t");
        scanf("%s", &buffer);
        send(clientSocket, buffer, strlen(buffer), 0);
        if (strcmp(buffer,":history") == 0){
            //send(clientSocket, buffer, strlen(buffer), 0);
            FILE* f= fopen("history.txt","r+");             // Ouvrir le fichier local en mode lecture
            char v;                     // Variable d'envoi

            do
            {
            v=fgetc(f);
            t=send(clientSocket,(const char*)&v,sizeof(v),0);     // Envoyer le caractère
            }while(v!=EOF);                 // Reboucler si on n'atteint toujours pas la fin du fichier
            close(f);                     // Fermer le fichier
            printf("(Historique envoyé)\n");

            ////////////////////////////////////////////////////////////////////////////////////////////////
```

```
      }
      if(strcmp(buffer, ":exit") == 0){
         close(clientSocket);
         printf("[-]Disconnected from server.\n");
         exit(1);
      }

      if(recv(clientSocket, buffer, 1024, 0) < 0){
         perror("[-]Error in receiving data.\n");
      }else{
         if(strlen(buffer) < 100){
            printf("Server: \t%s\n", buffer);
         }

      }
      memset(buffer, 0, sizeof(buffer));
   }

   return 0;
}
```

3.1.1 Variables

— *serverAddr* : représente la structure sockaddr_in contenant les informations du serveur
— *clientSocket* : représente le socket que l'on va connecter à l'adresse dur serveur et qui nous permettra l'échange avec le serveur

3.1.2 Connection

Une fois le socket créé, il faut le connecter. Cela se fait grâce à la fonction connect.

```
     ret = connect(clientSocket, &serverAddr, sizeof(serverAddr));
     if(ret == -1){
        perror("[-]Error in connection.\n");
      exit(1);
   }
```

Si connect retourne -1, c'est que la connection à échouer. Grâce à perror, on peut en savoir la cause exact.

3.1.3 Commandes

```
Que voulez-vous faire?
        :history        pour upload votre historique
        :exit           pour se déconnecter
Client:
```

1. :history permet d'envoyer son historique
2. :exit permet de se seconnecter

3.1.4 client.sh

C'est un script qui charge les n dernières commandes tapés par le client et lancer le client.

```
#!/bin/bash



echo "
 __           __  _                                _____ _ _                _
 \\ \\         / / | |                              / ____| (_)              | |
  \\ \\  /\\  / /__| | ___ ___  _ __ ___   ___| |    | |_  ___ _ __ | |_
   \\ \\/  \\/ / _ \\ |/ __/ _ \\| '_ \` _ \\ / _ \\ |    | | |/ _ \\ '_ \\| __|
    \\  /\\  /  __/ | (_| (_) | | | | | |  __/ |____| | |  __/ | | | |_
     \\/  \\/ \\___|_|\\___\\___/|_| |_| |_|\\___|\\_____|_|_|\\___|_| |_|\\__|



"


echo "Please enter the number of commands you wanna save ?"

read numberOfCommands

HISTFILE=~/.bash_history  # Set the history file.
set -o history            # Enable the history.
history -a
history $numberOfCommands > history.txt          # Save the history.

gcc -w -Wall -o myclientFork clientFork.c
./myclientFork  192.168.121.128

exit
```

Chapitre 4

Configuration

4.1 Coté serveur

Avant de lancer le serveur, il faut s'assurer que le firewall est désactivé. S'il ne l'est pas , impossible de joindre le serveur puisqu'il ne laissera rien passer.

```
sudo rcSuSEfirewall2 stop // pour executer en tant qu'administrateur
```

Pour le serveur, c'est tout ce qu'il faut faire. Une fois le firewall désactivé, vous pouvez compiler le fichier serverfork.c lancer votre serveur de la sorte

```
gcc -w -Wall -o server serverFork.c
./server XXX.XXX.XXX.XXX   //où XXX.XXX.XXX.XXX est l'adresse du serveur.
```

4.2 Coté client

Avant de lancer le client, il faut s'assurer que l'historique de commande se met à jour à chaque fois qu'une commande est entrée. Il faut copier ses deux lignes dans le fichiers /.bashrc. Ensuite redémarrer pour mettre à jour les modifications.

```
shopt -s histappend                      # active la fonction append pour l'historique
export PROMPT_COMMAND="history -a; history -c; history -r; $PROMPT_COMMAND"
```

Une fois le fichier /.bashrc à jour, vous devez modifier dans le script client.sh et y mettre l'adresse à laquelle vous voulez vous connecter et entrer l'adresse de votre serveur

```
gcc -w -Wall -o myclientFork clientFork.c
./myclientFork  192.168.121.128
```