Compte rendu du TP2

Exercice 1

Dans ce schéma, qui est le client et qui est le serveur ? Le serveur est le père, il donne des nombres aléatoires, le fils est le client qui calcul et affiche pi.

Code source

```c
#include <sys/socket.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <time.h>

#define BUFFER_SIZE 1024
#define MAX_TIRAGE 1000000

int main(){

    int nbLus = 0;
    double x;
    double y;
    double pi;
    int n = 0;
    int m = 0; //le nombre tirage tq x^2+y^2 <= 1
    char c = '\0';

    /* Création de la socketpair */
    int sockfd[2]; //Le tableau contenant des descripteurs de la socket
pair
    int spair = socketpair(AF_UNIX, SOCK_STREAM, 0, sockfd);
    if(spair == -1){
        perror("Erreur lors de la creation de la socket");
        return 1;
    }
    /* Le fils */
    if(fork() == 0){
        close(sockfd[0]);
        while (1){
            n++;
            write(sockfd[1], &c, sizeof(char));
            nbLus = read(sockfd[1], &x, sizeof(double));
            if(nbLus == -1){
                perror("Erreur de read X");
                return 1;
            }
            if(nbLus == 0){
                break;
```

```
            }

            write(sockfd[1], &c, sizeof(char));
            nbLus = read(sockfd[1], &y, sizeof(double));

            if(nbLus == -1){
                perror("Erreur de read Y");
                return 1;
            }
            if(nbLus == 0){
                break;
            }

            if(x*x + y*y <= 1){
                m++;
            }

            if(n%10000 == 0){
                pi = 4.0*m/n;
                printf("Valeur de pi %g \n", pi);
            }

            if(n >= MAX_TIRAGE){
                pi = 4.0*m/n;
                printf("Valeur finale de pi %g \n", pi);
                break;
            }
        }
        shutdown(sockfd[1], SHUT_RDWR);
        close(sockfd[1]);
        exit(0);
    }
    /* Le père - effectue les tirages*/
    close(sockfd[1]);

    srand(time(NULL));
    double z = (double) random()/RAND_MAX;

    while(1){
        nbLus = read(sockfd[0], &c, sizeof(char));
        if(nbLus == -1){
            perror("Erreur de read demande");
            return 1;
        }
        if(nbLus == 0){
            break;
        }
        z = (double) random()/RAND_MAX;
        write(sockfd[0], &z, sizeof(double));
    }

    shutdown(sockfd[0], SHUT_RDWR);
    close(sockfd[0]);
}
```

Exercice 2

Pourquoi les sockets stream (par exemple, via TCP) sont mieux adaptées à ce type d'application ?

En mode socket stream, je suis connecté,il y une fiabilité de la connexion, si le message n'est pas reçu, le message sera redemandé. Cela permet également de savoir combien de clients sont connectés pour leur envoyer les messages. On peut également gérer les déconnexions.

Testez l'application. Quel port utilise cette application pour communiquer (utilisez la commande ss et sa page de man pour le découvrir) ? Quelle option de ss avez-vous utilisé ?

Avec la commende `ss -pant` pour afficher tout les ports en écoute. Le serveur écoute toutes les adresses sur le port 5015.

State	Recv-Q	Send-Q	Local Address:Port	Peer Address:Port	Port
LISTEN	0	128	0.0.0.0:5015	0.0.0.0:*	users: (("chat_serveur",pid=6110,fd=5), ("chat_serveur",pid=5531,fd=5))
ESTAB	0	0	127.0.0.1:5015	127.0.0.1:54050	users: (("chat_serveur",pid=6110,fd=6), ("chat_serveur",pid=5531,fd=6))
ESTAB	0	0	127.0.0.1:54050	127.0.0.1:5015	users: (("chat_client",pid=6109,fd=3), ("chat_client",pid=6108,fd=3))

De combien de processus l'application serveur est-elle composée pour gérer 1, 2, n clients (utilisez la commande ps et sa page de man pour le découvrir) ?

L'application serveur est toujours composée d'un unique processus, le nombre de processus ne dépends pas du nombre de client.

Partie client

Code source

```
#include <stdio.h>                  /* fichiers d'en-tête classiques */
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <signal.h>
#include <string.h>
#include <fcntl.h>
#include <errno.h>

#include <sys/socket.h>             /* fichiers d'en-tête "réseau" */
```

```c
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>

#define PORT_SERVEUR 5015           /* Numéro de port pour le serveur */
#define BUFFER_SIZE  1024           /* Taille maximum des messages */

int main(int argc, char *argv[]){

    int nbLus;
    char message[BUFFER_SIZE];

    char pseudo[strlen(argv[2])+1];
    strcpy(pseudo, argv[2]);

    char messageToSend[BUFFER_SIZE];


    /* 1. Création de la socket */

    int sclient = socket(AF_INET, SOCK_STREAM, 0);
    if(sclient == -1){
        perror("Erreur de création de la socket");
        return 1;
    }

    /* 2. Création de la structure d'adresse */

    struct sockaddr_in saddr = {0};
    saddr.sin_family = AF_INET;
    saddr.sin_port = htons(PORT_SERVEUR);
    saddr.sin_addr.s_addr = inet_addr(argv[1]);

    /* 3. Connexion au serveur */

    while(connect(sclient, (struct sockaddr *) &saddr, sizeof(saddr)) ==
-1);

    /* Lecture des messages envoyés par le serveur */
    if(fork() == 0){
        while(1){
            int nbLus = read(sclient, message, BUFFER_SIZE-1);
            if (nbLus == -1){
                perror("Erreur lors du read depuis le serveur");
            }
            if (nbLus == 0){
                break;
            }

            //Ecriture sur la sortie
            write(1,"\n", strlen("\n"));
            write(1,message, nbLus);
            write(1,"> ", strlen("> "));
        }
```

```
            shutdown(sclient, SHUT_RDWR);
            close(sclient);
            exit(0);
        }

        /* Envoie de message*/
        while(1){
            write(1, "> ", strlen("> "));
            nbLus = read(0, message, BUFFER_SIZE);

            if(nbLus == -1){
                perror("Erreur lors du read");
                return 1;
            }

            if(nbLus == 0){
                break;
            }

            message[nbLus] = '\0';
            sprintf(messageToSend, "%s : %s", pseudo, message);

            write(sclient, messageToSend, strlen(messageToSend));

        }
        close(sclient);
        shutdown(sclient, SHUT_RDWR);
        return 0;
    }
```

Partie serveur

Expliquez pourquoi l'architecture traditionnelle de serveur qui consiste à générer un processus fils par connexion est difficile à réaliser dans le cas d'une application de messagerie instantanée.

Générer un processus fils pour chaque connexion client dans une application de messagerie instantanée peut être inefficace et difficile à gérer à grande échelle pour plusieurs raisons :

- Consommation des ressources : Chaque processus fils consomme des ressources (mémoire, CPU), et à mesure que le nombre de clients augmente, cela peut rapidement devenir non scalable.
- Synchronisation des messages : Dans un système de chat, tous les clients doivent recevoir les messages envoyés par chaque client. Gérer cette diffusion avec des processus séparés implique une complexité supplémentaire pour partager les messages entre les processus fils.

Pourquoi le processus serveur serait-il tué en écrivant sur une socket fermée ? Proposez une solution pour que cela n'arrive pas

Si un serveur tente d'écrire sur une socket fermée par un client, il peut recevoir un signal *SIGPIPE*, ce qui provoque la terminaison du processus serveur. Ce problème survient lorsqu'une tentative d'écriture est faite sur une connexion qui n'existe plus.

On peut ignoré le signal SIGPIPE. Ou et c'est ce qui est fait dans le programme suivant, retirer des la fin de la connexion, le client déconnecté de l'ensemble de descripteur.

Code source

```c
#include <stdio.h>                  /* fichiers d'en-tête classiques */
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <signal.h>
#include <string.h>
#include <fcntl.h>
#include <errno.h>

#include <sys/socket.h>             /* fichiers d'en-tête "réseau" */
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>

#define PORT_SERVEUR 5015           /* Numéro de port pour le serveur */
#define MAX_CLIENTS   128           /* Nombre maximum de clients */
#define BUFFER_SIZE  1024           /* Taille maximum des messages */
#define ADRESSE "127.0.0.1"

int main(int argc, char *argv[]) {

    signal(SIGCHLD, SIG_IGN);

    int nbLus;
    char message[BUFFER_SIZE];

    /* 1. Création de la socket d'écoute */

    int secoute = socket(AF_INET, SOCK_STREAM, 0);
    if(secoute == -1){
        perror("Erreur de création de la socket");
    }

    int sservice;

    /* 2. Création de la structure d'adresse */

    struct sockaddr_in saddr = {0};
    saddr.sin_family = AF_INET;
    saddr.sin_port = htons(PORT_SERVEUR);
    saddr.sin_addr.s_addr = inet_addr(ADRESSE);

    struct sockaddr_in caddr = {0};
    unsigned int caddrlen;

    /* 3. On attache l'adresse et on ouvre l'écoute */
```

```
    if(bind(secoute, (struct sockaddr *) &saddr, sizeof(saddr)) == -1){
        perror("Erreur lors du bind");
        return 1;
    }

    if(listen(secoute, MAX_CLIENTS) == -1){
        perror("Erreur lors du listen");
        return 1;
    }

    /* 4. Création du set de descripteur */

    fd_set ensemble, temp;
    FD_ZERO(&ensemble);
    FD_SET(secoute, &ensemble);

    int max = secoute; //le descripteur max
    printf("Serveur en attente de nouveaux clients ou messages. \n");
    while(1){

        temp = ensemble;
        if(select(max+1, &temp, NULL, NULL, NULL) == -1){
            printf("Signal reçu \n");
        }

        //* 5. Vérification de l'arrivée de client */

        if(FD_ISSET(secoute, &temp)){
            caddrlen = sizeof(caddr);
            sservice = accept(secoute, (struct sockaddr *) &caddr,
&caddrlen);
            if(sservice > max){
                max = sservice;
            }
            FD_SET(sservice, &ensemble);
            printf("Un nouveau client est connecte \n");
        }

        /* 6. Gestion des clients */
        for (int fd = 0; fd < max+1; fd++){

            /* On ne lit pas sur la socket d'écoute, si la socket fd est
inactive, il n'y a rien à lire donc on saute*/
            if(!FD_ISSET(fd, &temp) || fd == secoute){
                continue;
            }

            nbLus = read(fd, message, BUFFER_SIZE-1);

            if(nbLus == -1){
                perror("Erreur de read");
                FD_CLR(fd, &ensemble); //Retire le descripteur du set
                shutdown(fd, SHUT_RDWR);
                close(fd);
```

```
                continue;
            }
            if(nbLus == 0){
                printf("Déconnexion d'un client \n");
                FD_CLR(fd, &ensemble); //Retire le descripteur du set
                shutdown(fd, SHUT_RDWR);
                close(fd);
                continue;
            }

            /* Affichage du message sur la sortie standard */
            if(nbLus > 0){
                write(1, "Message reçu de ", strlen("Message reçu de "));
                write(1, message, nbLus);
            }

            /* Affichage du message reçu chez tout les clients*/
            for (int fdi = 3; fdi <max+1; fdi++){
                if(fdi == fd){
                    continue;
                }
                write(fdi, message, nbLus);
            }
        }
    }
    shutdown(secoute, SHUT_RDWR);
    close(secoute);
    return 0;

}
```