Utilisation des sockets pair

Nous allons construire un programme client serveur qui va faire une approximation de pi avec la méthode de Monte Carlo.

La méthode de Monte-Carlo consiste à effectuer des tirages aléatoires de nombres réels x et y tels qu'ils soient compris entre 0 et 1. N est le nombre de tirages de couples x et y, et M le nombre de tirages tels que x^2 +y^2 <= 1, ainsi, une approximation de pi est π ≃ (4M)/N

Nous allons utiliser **les sockets pairs** pour construire une communication client/serveur **locale**. Le serveurs sera le père, qui fournir des nombres aléatoire et le client sera le fils, qui recevra les nombres et estimera pi.

On commence par définir les constantes :

```
#define BUFFER_SIZE 1024
#define MAX_TIRAGE 1000000
```

Et on construit notre architecture :

```
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';
    /* 1. 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]);
}
```

1. **On crée la socket pair.**

   Pour créer une socketpair, on utilise la primitive `socketpair()` qui prends 4 paramètres :

   - Le domaine : AF_UNIX ou AF_LOCAL
   - le type : SOCK_STREAM ou SOCK_DGRAM
   - le protocol : 0 par défaut
   - un tableau de deux entiers qui va contenir les descripteurs de fichier

   Elle renvoie 0 en cas de succès et -1 en cas d'échec.

   Attention : Les sockets créées sont nécessairement locales ; ainsi, domaine doit être AF_UNIX (ou AF_LOCAL)

   Le tableau d'entier `int sockfd[2]` va donc contenir les descripteurs utilisé par le père et le fils. Le fils utilisera sockfd[1] et le père sockfd[0].

2. **On implémente le fils**

   La première chose à faire est de fermer la socket du père, on ne l'utilisera pas dans le fils. **Attention, il ne faut pas faire shutdown pour autant, il ne faut pas couper la communication**.

   On ouvre une boucle while pour la communication.

   On commence par faire une demande de valeur au père lui envoyant un caractère sans importance. Pour cela on utilise la primitive `write()`

   ```
   write(sockfd[1], &c, sizeof(char));
   ```

   Il va récuperer des valeurs aléatoire, ces valeurs, il les stockera dans les variables `x` et `y`. Pour les récuperer on va utiliser la primitive read(), on stockera sa valeur de retour pour faire les tests communs. (-1 erreur, 0 connexion terminée);

   ```
   nbLus = read(sockfd[1], &x, sizeof(double));
   if(nbLus == -1){
       perror("Erreur de read X");
       return 1;
   }
   if(nbLus == 0){
       break;
   }
   ```

   On remarque que l'on donne l'adresse de x (`&x`), read écrira dans la variable à l'adresse fournie. On fait la même chose avec y.

Et viens l'heure du calcul, on incremente n, on fait la vérification x^2 + y^2 <= 1, et si la condition est vérifié on incremente m. Toutes les 10000 répétitions, on afficher la valeur actuelle de pi. Quand le compteur est terminé, on affiche la valeur finale et on termine.

Pour terminer on utilise la primitive `exit()`.

Et on ferme la communication et la connexion.

3. **On implémente le père.**

   Cette fois, pour le père c'est la socket correspondante au fils que l'on ferme.

   Ensuite on entre dans la boucle while de communication. Dans cette boucle, le père commence par attendre une demande de la part du fils, avec la primitive read(), on n'oublie pas que **la primitive read() mets le processus en attente.**

   Une fois que la demande est reçue, il génère un nombre aléatoirement et l'envoie au fils :

   ```
   z = (double) random()/RAND_MAX;
   write(sockfd[0], &z, sizeof(double));
   ```

   Quand la connexion est interrompue, on sort de la boucle et on ferme la communication et la socket.

close() et shutdown().

close()

La primitive système `close()` ferme et libère la socket sockfd dans le processus appelant. Toute opération ultérieure sur sockfd échouera,avant que ce numéro ne soit pris pour un nouveau descripteur par socket(), accept(), open(), pipe() ou autre primitive.

Cette primitive prend en paramètre un **descripteur de fichier**.

shutdown()

La primitive `shutdown()` ferme la connexion établie par la socket (spécifiée dans le premier paramètre) dans le sens défini par le paramètre sens (2eme paramètre). Si sens vaut SHUT_RD, la connexion est fermée en lecture, si sens vaut SHUT_WR, elle sera fermée en écriture, et elle sera enfin fermée dans les deux sens pour SHUT_RDWR.

Elle prend donc 2 paramètre :

- le descripteur d'une socket
- le sens de fermeture = SHUT_RD : lecture, SHUT_WR : écriture, SHUT_RDWR : les deux