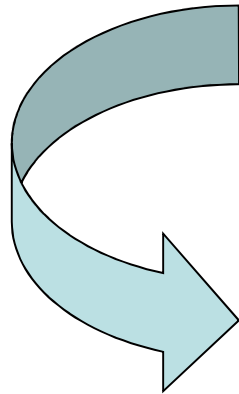


# ***Communications classiques entre processus sous LINUX***

**Communication entre processus d'une même application**

**Communication entre processus indépendants**

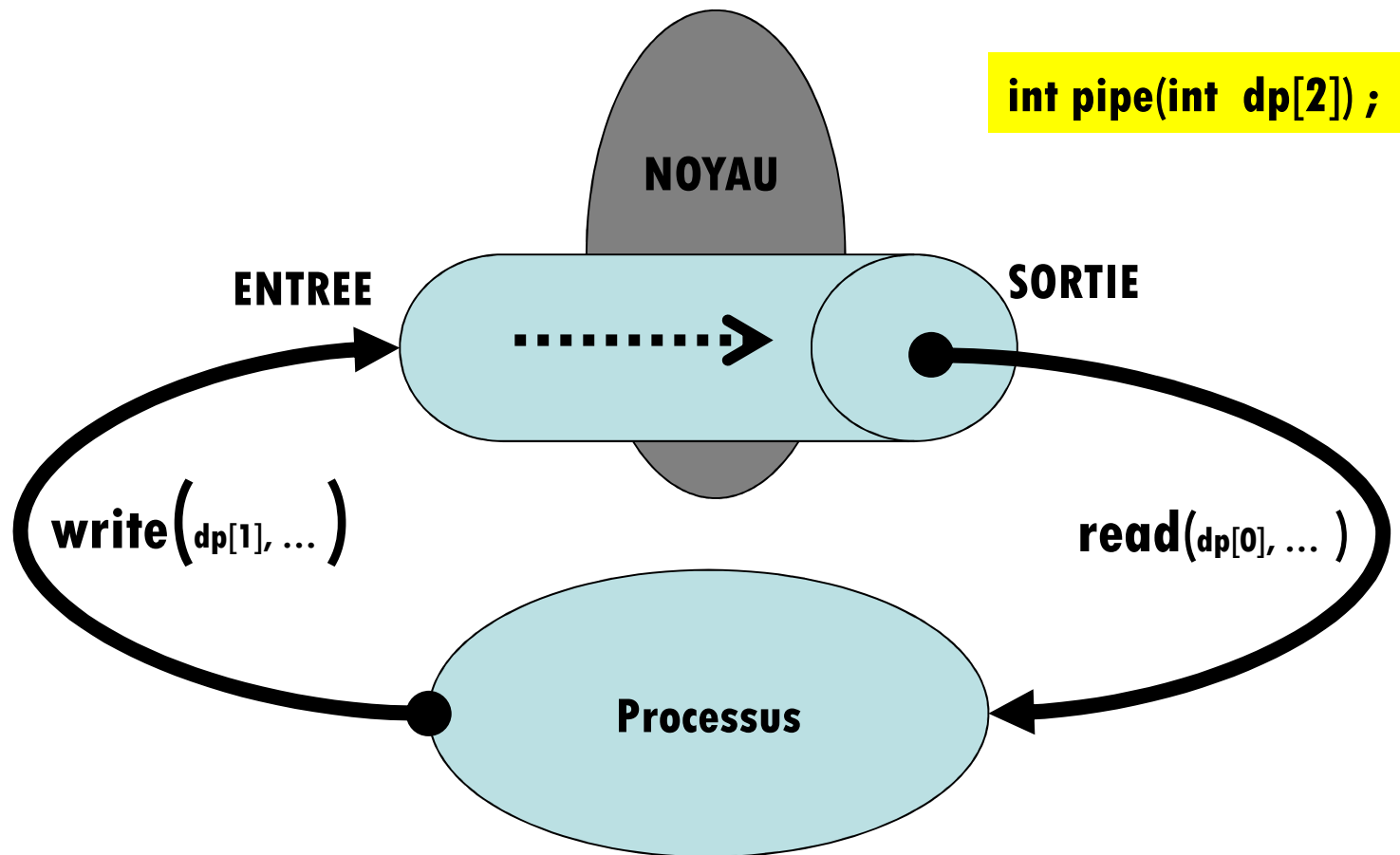
**Communication entre logiciels se trouvant sur des machines différentes**



**Les tubes anonymes**

**Les tubes nommés**

# Les tubes anonymes

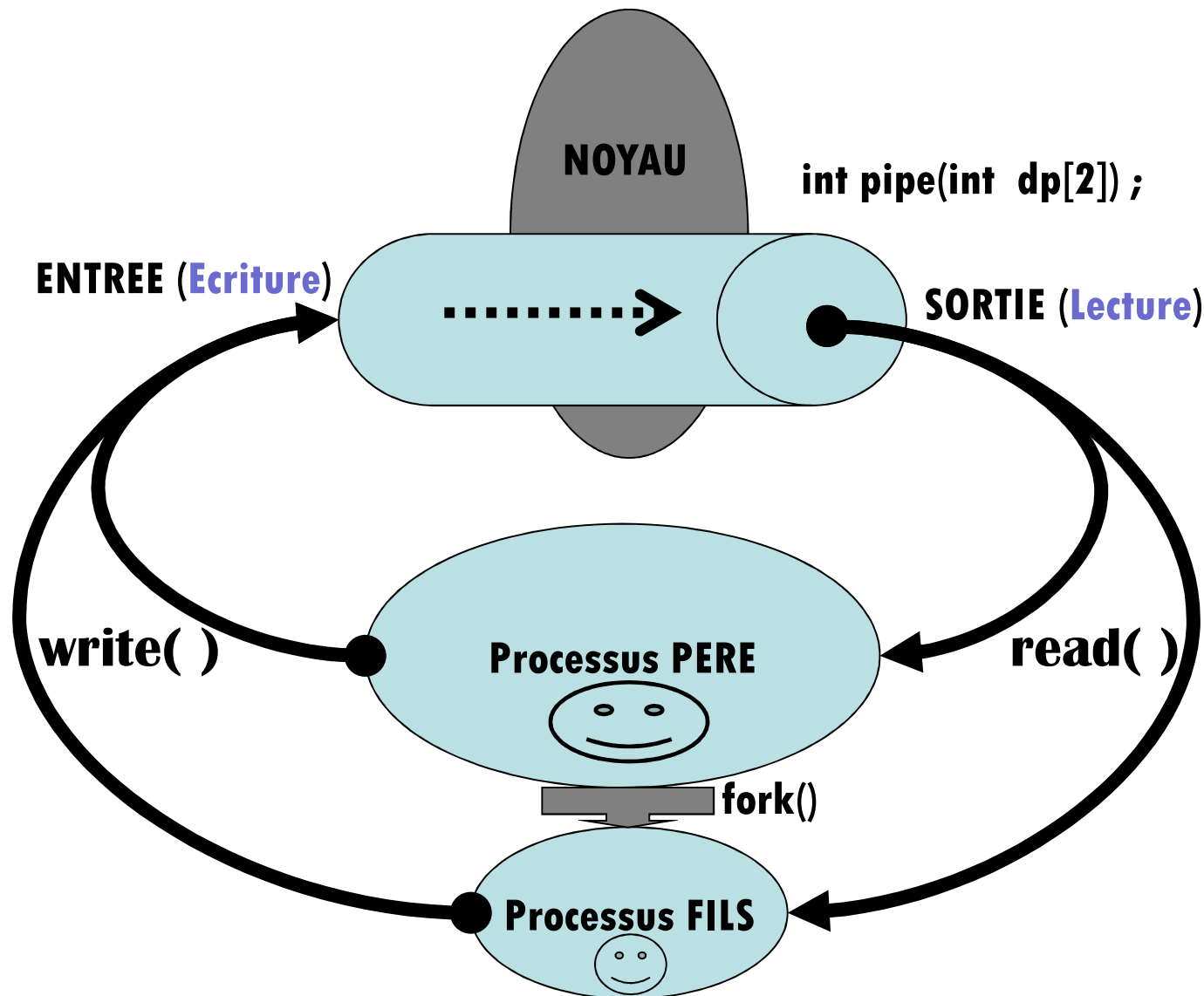


**Tube de communication**

## Les tubes anonymes - Exemple

```
int main() {  
    int tube[2] ;  
    unsigned char buffer[256] ;  
    int i ;  
    printf("Création du tube\n") ;  
    if (pipe(tube) != 0) {  
        perror("Problème – création du tube\n") ;  
        exit (1) ;  
    }  
    for(i=0 ; i<256 ; i++) buffer[i] = i ;  
    printf("Ecriture dans le tube\n") ;  
    if (write(tube[1], buffer, 256) != 256) {  
        perror("Problème – Ecriture dans le tube\n") ;  
        exit (1) ;  
    }  
    printf("Lecture depuis le tube\n") ;  
    if (read(tube[0], buffer, 256) != 256) {  
        perror("Problème – Lecture depuis le tube\n") ;  
        exit (1) ;  
    }  
    return 0 ;  
}
```

## Tube anonyme du PÈRE vers le FILS



## Tube anonyme du PÈRE vers le FILS - Exemple

```
int main(){  
    int fils1, fils2, n,m,tube[2];  
    char buffer[256];  
    if (pipe[tube] != 0) { perror("Problème – création du tube\n"); exit (1); }  
    fils1 = fork();  
    if (fils1 == 0) {  
        close(tube[0]);  
        printf("Je suis le fils producteur\n");  
        printf("Je dépose 5 caractères dans le pipe\n");  
        write(tube[1] , "abcde" , 5) ;  
        close(tube[1]) ; exit (3);  
    }  
    else {  
        fils2 = fork();  
        if (fils2 == 0) {  
            printf("Je suis le fils consommateur");  
            close(tube[1]) ; read(tube[0] , buffer, 5);  
            printf("Voici les caractères lus : %s\n" , buffer);  
            exit(3);  
        }  
        else {  
            printf("Le processus père se termine");  
            close(tube[0]);  
        }  
    }  
    return 0;  
}
```

## Exercice 1 (4 points)

On dispose d'une fonction `f()` dont on ne peut pas modifier le code source.

Cette fonction écrit une certaine quantité de données sur la sortie standard (descripteur 1 = écran par défaut), que l'on aimerait récupérer dans une chaîne de caractères supposée assez grande, pour un traitement ultérieur.

```
{  
char ch[TAILLE_SUFFISANTE];  
...  
f();  
...  
utiliser_resultat(ch);  
}
```

**(1.5 points)** Insérez du code en amont et en aval de l'appel à `f()` pour que tous les caractères émis par `f()` sur la sortie standard soient récupérées dans le tableau `ch[]`.

Vous pouvez utiliser des variables et appels système supplémentaires, mais vous ne pouvez pas utiliser de fichiers ni de processus supplémentaires.

**(1 point)** Que peut-il se passer si la taille des données redirigées est trop importante ?

**(1.5 points)** Proposez une deuxième version corrigeant ce problème. Pour ce faire, vous pouvez utiliser un processus supplémentaire.

## REDIRECTION DES ENTREES / SORTIES - Tube anonyme

### Implémentation de la commande ls | wc

```
#include <stdio.h>
int main( ) {
    int pfd[2]; int pid;
    pipe(pfd); // création d'un tube
    pid=fork();
    if (pid == 0) { // fils
        close(pfd[1]); // ferme l'entrée du tube.
        dup2(pfd[0] , 0); // copie la sortie du tube vers l'entrée standard.
        close(pfd[0]); // ferme le descripteur de la sortie du tube.
        execlp("wc", "wc",NULL); // recouvre avec wc
    } else { // pere
        close(pfd[0]); // ferme la sortie du tube.
        dup2(pfd[1] , 1); // copie l'entrée du tube vers la sortie standard
        close(pfd[1]); // ferme le descripteur de l'entrée du tube.
        execlp("ls", "ls",NULL); // recouvre avec ls
    }
    return 0 ;
}
```

# Les tubes nommés

## Avantage

connecter des processus quelconques **sans lien de parenté** particulier

## Utilité

un processus **serveur** permanent qui offre des services à des **processus clients**

## Fonctionnement

Ils fonctionnent comme les autres tubes pour ce qui concerne la lecture/écriture (**fifo**) , sauf qu'ils **existent** réellement dans le **système de fichier UNIX**, et apparaissent comme les fichiers permanents.

```
$ls -l  
prw-r--r-- ..... TubeNomme
```



## Création d'un tube nommé

```
#include <sys/types.h>  
int mkfifo(char *nom, mode_t mode);
```

**nom** indique le nom du tube à créer

Le paramètre **mode**, ses droits d'accès, généralement **0644**

```
mkfifo("notreTube", 0644);
```

## Ouverture d'un tube nommé

```
#include <fcntl>  
int fd;  
...  
fd = open("TubeNomme", O_WRONLY);
```

# Accès d'un tube nommé

**read() et write()**

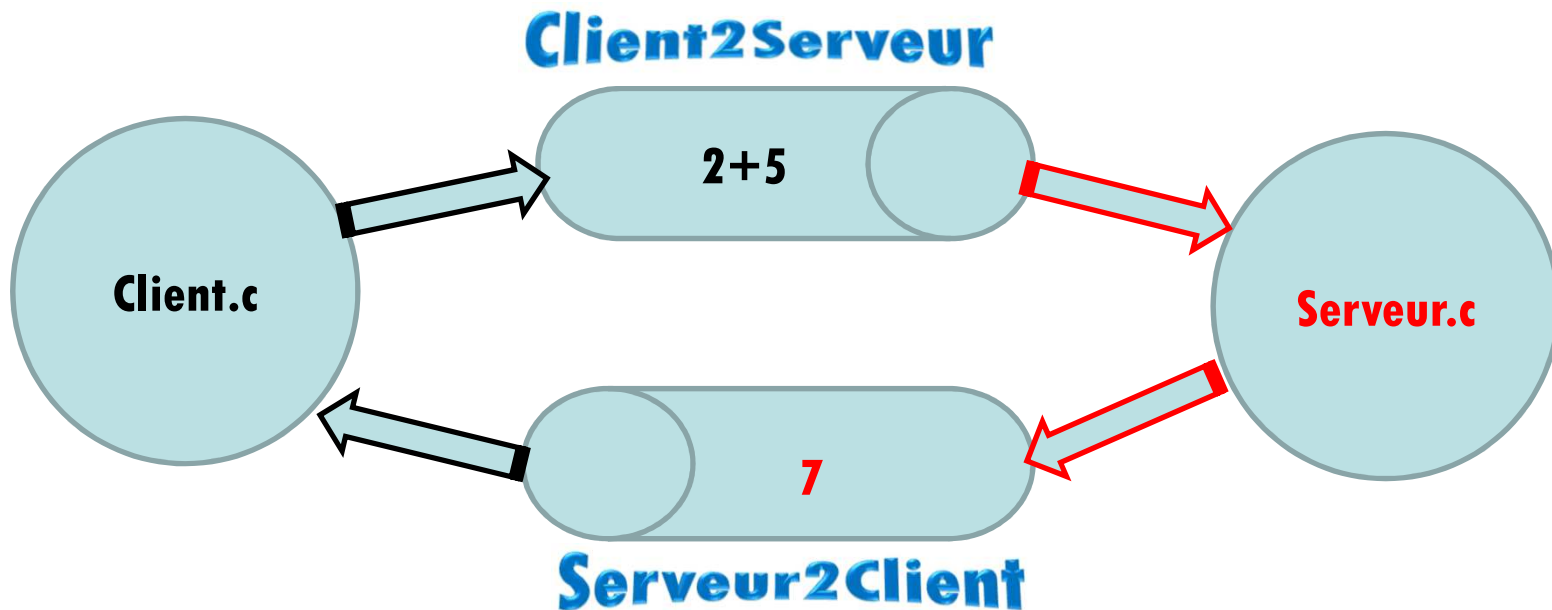
Les **entrées/sorties** sur un tube nommé,  
se passent comme pour les tubes anonymes.

**read() et write()** sont bloquantes ou non  
selon le mode d'ouverture effectuée.

En général,  
un processus **serveur** choisira une ouverture en **écriture non bloquante**  
pour pouvoir écrire des messages sans attendre.

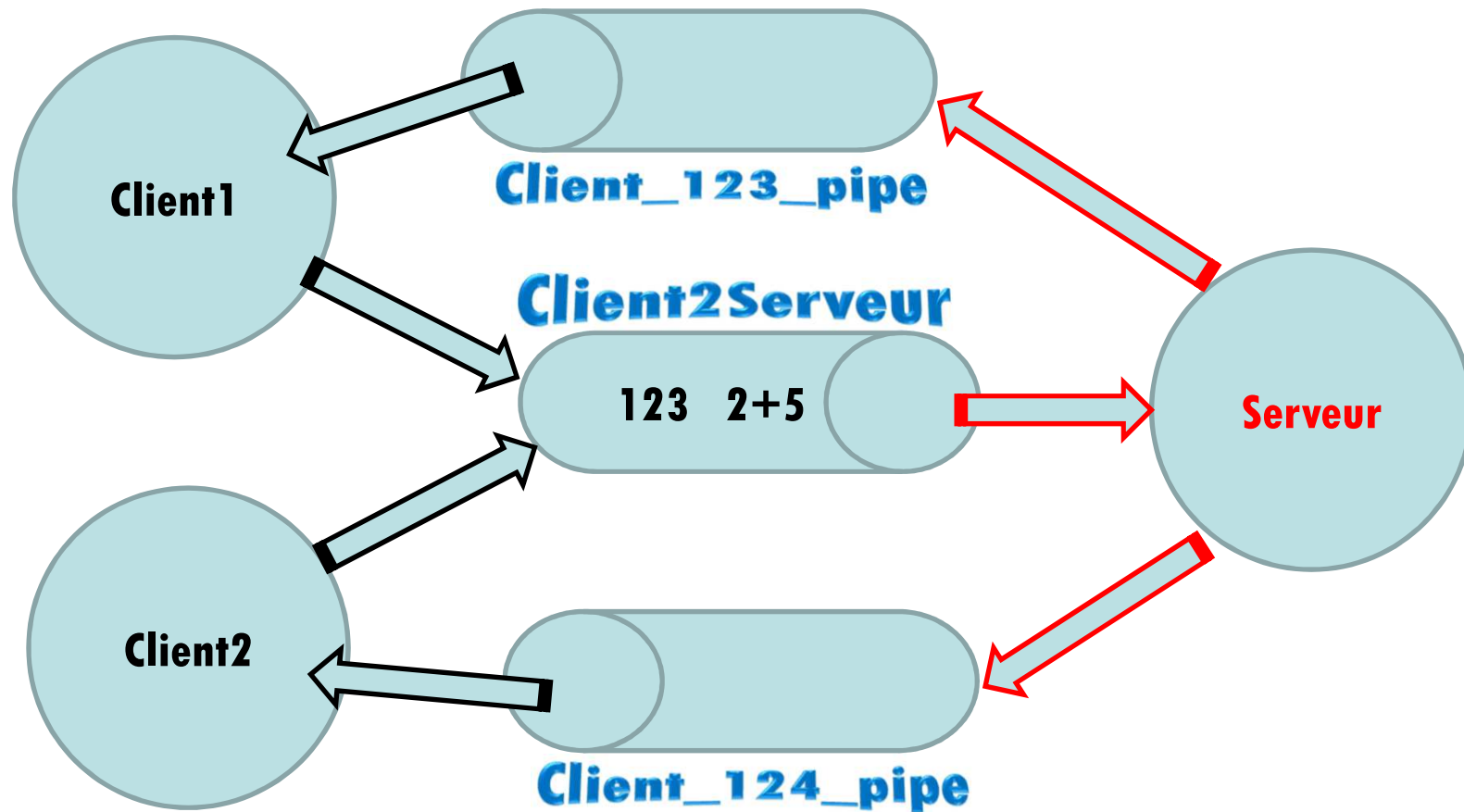
Un processus **client**, ouvrira un pipe nommé en **lecture bloquante**  
pour attendre de lire un message.

## Exemple de programme



**Client.c** envoie une requête de type **a + b** dans le tube **Client2Serveur**,  
**Serveur.c** lui retourne la réponse dans le tube **Serveur2Client**.

# Travaux Pratiques



## Exemple de programme

### Programme Serveur.c

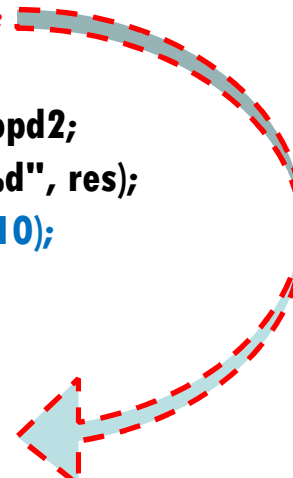
```
#include <sys/types.h>
#include <fcntl.h>
#include <string.h>
#include <stdio.h>
#define QUESTION "Client2serveur"
#define REPONSE "Serveur2Client"

main(){
    int fdq, fdr;
    unlink(QUESTION);
    unlink(REPONSE);
    if ( mkfifo(QUESTION, 0644) == -1 ||
         mkfifo(REPONSE, 0644) == -1) {
        perror("Impossible creation fifos");
        exit(2);
    }

    fdq = open(QUESTION, O_RDONLY);
    fdr = open(REPONSE, O_WRONLY);
    trait(fdr, fdq);
    close(fdq); close(fdr);
    unlink(QUESTION); unlink(REPONSE);
    exit(0);
}
```

```
void trait(int fdr, int fdq)
{
    int opd1, opd2, res;
    char opr; quest[11]; rep[11];

    while (1) {
        read(fdq, quest, 10);
        sscanf(quest, "%d%ls%d", &opd1, &opr, &opd2);
        if (strcmp(quest, "Ciao") == 0) {
            strcpy(rep, "Bye");
            write(fdr, rep, 10);
            break;
        }
        res = opd1 + opd2;
        sprintf(rep, "%d", res);
        write(fdr, rep, 10);
    }
}
```




## Exemple de programme

### programme **Client.c**

```
#include <sys/types.h>
#include <fcntl.h>
#include <string.h>
#include <stdio.h>
#define QUESTION "Client2Serveur"
#define REPONSE "Serveur2Client"
int main() {
    int fdq, fdr;
    fdq = open(QUESTION, O_WRONLY);
    if (fdq == -1) {
        fprintf(stderr, "Pb ouvrir fifo %s\n", QUESTION);
        fprintf(stderr, "Lancer serveur d'\nabord?\n");
        exit(2);
    }
    fdr = open(REPONSE, O_RDONLY);
    if (fdr == -1) {
        fprintf(stderr, "Pb ouvrir fifo %s\n", REPONSE);
        fprintf(stderr, "Lancer serveur d'\nabord?\n");
        exit(2);
    }
    trait(fdr, fdq);
    close(fdq); close(fdr); return 0;
}
```

```
void trait(int fdr, int fdq)
{
    char rep[11]; quest[10];
    while (1) {
        if (gets(quest) == NULL) exit(2);
        write(fdq, quest, 10);
        printf("Client -> %s \n", quest);
        read(fdr, rep, 10);
        printf("Serveur -> %s \n", rep);
        if (strcmp(rep, "Bye") == 0) break;
    }
}
```



## Exemple de programme

### Script d'exécution

**\$ ls Serveur2Client Client2Serveur**

**cli\_serv not found  
serv\_cli not found**

**\$ ./Client**

**Pb ouvrir fifo cli\_serv  
Lancer serveur d'abord?**

**\$ ./ serveur &  
[1] 21653**

**\$ ./Client**

**2 +4**

**Client -> 2 +4**

**Serveur -> 6**

**3 + 7**

**Client -> 3 + 7**

**Serveur -> 10**

**Ciao**

**Client -> Ciao**

**Serveur -> Bye**

**[1] + Done serveur**