Unix programming project report

Project idea:

Program a client/server interprocess communication application (Front and Back end) with 2 types of communication modes:

- Named pipes
- TCP/IP sockets

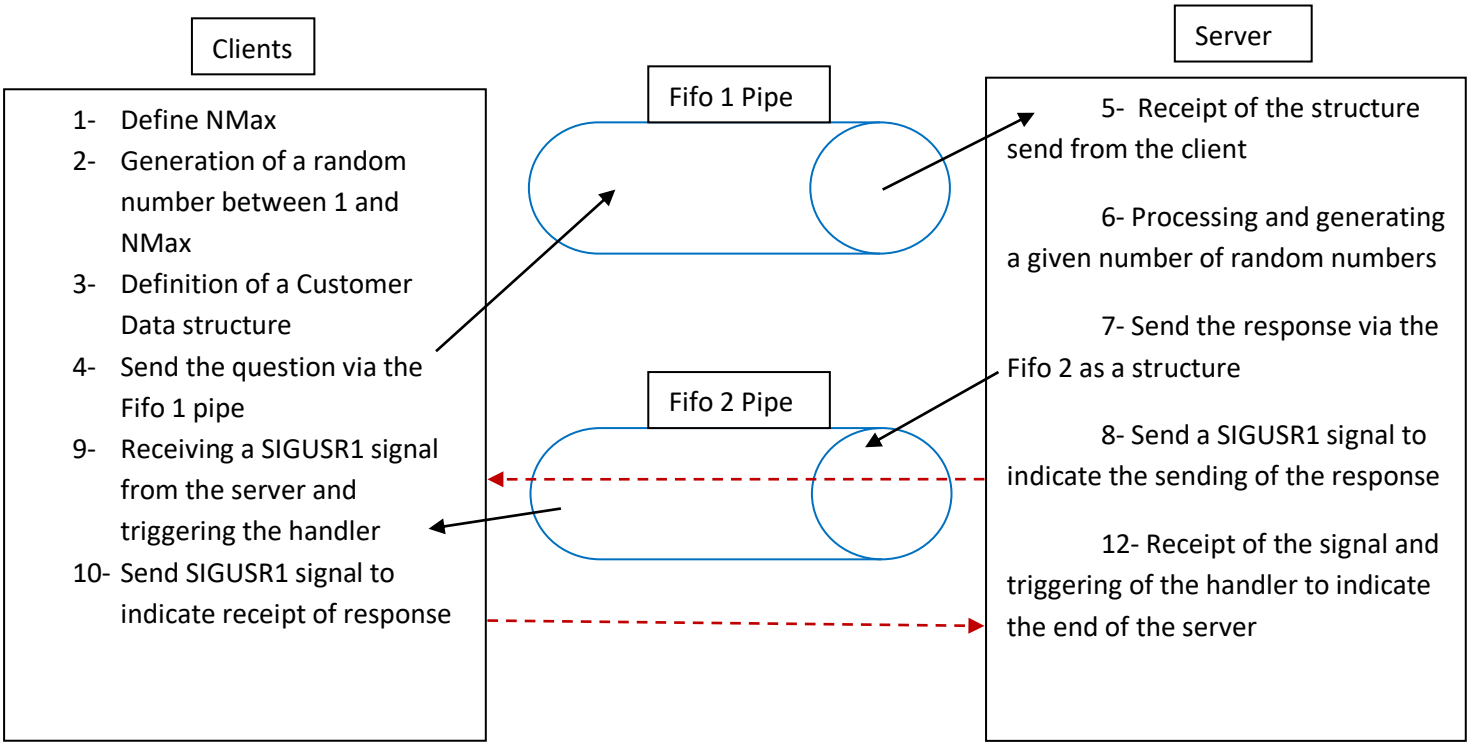
The client generates a random number x between 0 and NMax , and sends it to the server , which sends back to the client x other random numbers between 0 and NMax .

Technologies used:

Back End		Front End

Back end :

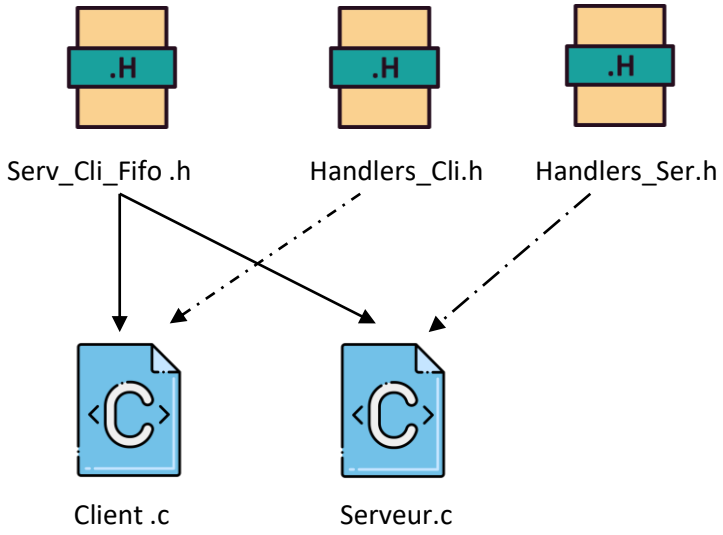
Methode 1 : Named Pipe



The named pipe Backend is represented by 5 files:

Header Files :

C files :



The code of the Serv_Cli_Fifo.h file: will be included in Client.c and Server.c

The libraries needed for Client.c and Server.c files

```
1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<unistd.h>
4 #include<fcntl.h>
5 #include<sys/stat.h>
6 #include<signal.h>
7 char nomTube[]="Fifo1";
8 char nomTube2[]="Fifo2";
9 struct ClientData {
10   int mypid;
11   int monentier;
12   int nmax;
13 };
14 struct ServerData {
15   int mypid;
16   int monentier;
17 };
18 int NMax=10;
```

Declaration of FIFO1 and FIFO 2 pipes

Structure Data Client which will be sent via FIFO1 to the server which contains the pid of the client (to be able to send a signal from the client to the server), an integer which is the number of random numbers requested from the server, and Nmax

Server Data structure which will be sent via FIFO2 to the client which contains the pid of the server (to be able to send a signal from the server to the client), an integer which is one of the random numbers requested from the server.

Definition of NMax

The code of the Handlers_Cli.h file: which will be included in Client.c:

```
void hand_reveil(int sig)
{
printf("je me suis reveillé et  recu la réponse\n");
}
```

Function to perform when receiving a USR1 signal from the server

The code of the Handlers_Serv.h file: which will be included in Server.c:

```
void fin_serveur(int sig)
{
printf("Le client m'a avertit qu'il a recu le  st , c est la fin de mon travail\n");
}
```

Function to be performed when receiving a USR1 signal from the client that it has received the response

The code of **Client.c** file:

client.c

Include files

Structure initialization

Retrieval of the client pid and assignment in mypid of the client structure

Creating the pipe and opening the file descriptor

When receiving SIGUSR1 from the server, execution of the wake-up handler

Generating the random number and filling the structure

Reception of the message from the server and reading of the Fifo2 after the creation of a file descriptor

Send message receipt confirmation signal to server

```
#include"serv_cli_fifo.h"
#include"Handlers_Cli.h"
int main()
{
/*Déclarations*/
struct ClientData CD;
struct ServerData SD;
CD.mypid=getpid();
/*Ouverture des tubes nomées*/
mkfifo(nomTube,0644);
int desc=open(nomTube,O_WRONLY);
/*Installation des handlers*/
signal(SIGUSR1,hand_reveil);
/*Construction et envoie d'une question*/
printf("mon pid est %d \n",CD.mypid);
CD.nmax=NMax;
srand(getpid());
CD.monentier= 1 +rand()%(CD.nmax-1+1);
write(desc,&CD,sizeof(CD));
/*Lecture de la réponse*/
for (int compteur = 0 ; compteur < CD.monentier ; compteur++)
{
int desc2=open(nomTube2,O_RDONLY);
read(desc2,&SD,sizeof(SD));
/*Traitement locale de la réponse*/
printf("le serveur %d a renvoyer le nombre:%d \n",SD.mypid,SD.monentier);
}
/*Envoie du signal SIGUSR1au serveur*/
kill(SD.mypid,SIGUSR1);
return 0;
}
```

The code of **Serveur.c** file :

Include files

Structure initialization

While (1) , to let the server listen to all clients

Recovery of the server pid and assignment in mypid of the server structure

When receiving the SIGUSR1 from the client, execution of the end handler of the

Reading the fifo2 client structure and closing the descriptor

Generation of random numbers, creation of tube 2, and file descriptor and writing in fifo 2 the response.

Sends a signal to the client concerned (identified by its pid) to wake it up

```
#include"serv_cli_fifo.h"
#include"Handlers_Serv.h"
int main()

{
/*Declaration*/
struct ClientData CD;
struct ServerData SD;
while(1){
/*Initialisation du generateur aleatoirede nombres*/
srand(getpid());
SD.mypid=getpid();
/*Ouverture des tubes nommes*/
int desc=open(nomTube,O_RDONLY);
/*Installation des handlers*/
signal(SIGUSR1,fin_serveur);
/*Lecture d'une question*/
read(desc,&CD,sizeof(CD));
close(desc);
/*Envoie de la reponse*/
printf("ecrivain %d a demandé du serveur: %d entiers au hasard\n",CD.mypid,CD.monentier);
for (int compteur = 0 ; compteur < CD.monentier ; compteur++)
{
SD.monentier = 1 +rand()%(CD.nmax-1+1);
printf("l'entier au hasard numero %d est %d \n",compteur,SD.monentier);
mkfifo(nomTube2,0644);
int desc2=open(nomTube2,O_WRONLY);
write(desc2,&SD,sizeof(SD));
}
/*Envoie du signal SIGUSR1 au client*/
kill(CD.mypid,SIGUSR1);
}
return 0;
}
```

Rq: In the previous code, we explained the code of the business side of the project which concerns method 1 (The named pipes), and in the main code attached to this report you will find lines of code containing the creation and the opening client and server output text files to facilitate communication with the Front End side.

Client Side :

```
fh= fopen("outputclient1.txt","a");
if(fh==NULL)
{
printf("error opening file");
exit(1);
}
char serveurpidtotext[100] ;
sprintf(serveurpidtotext, "%d", SD.mypid);
char text2[100] ="le serveur  ";
strcat(text2, serveurpidtotext);
char text3[100] =" a renvoyer le nombre:  ";
strcat(text2,text3);
char monentiertotext[100] ;
sprintf(monentiertotext, "%d", SD.monentier)
strcat(text2,monentiertotext);
char text4[100]="\n";
strcat(text2,text4);
fputs (text2,fh);
fclose(fh);
```

```
fh= fopen("outputserver1.txt","w");
if(fh==NULL)
{
printf("error opening file");
exit(1);
}
printf("ecrivain %d a demandé du serveur: %d entiers au hasard\n",CD.mypid,CD.monentie
char mypidtotext[100] ;
sprintf(mypidtotext, "%d", CD.mypid);
char text1[100] ="ecrivain ";
strcat(text1, mypidtotext);
char text2[100] =" a demandé du serveur: ";
strcat(text1, text2);
char monentiertotext[100] ;
sprintf(monentiertotext, "%d", CD.monentier);
strcat(text1, monentiertotext);
char text3[100] =" entiers au hasard\n";
strcat(text1, text3);
fputs (text1,fh);
fclose(fh);
```

Server side :

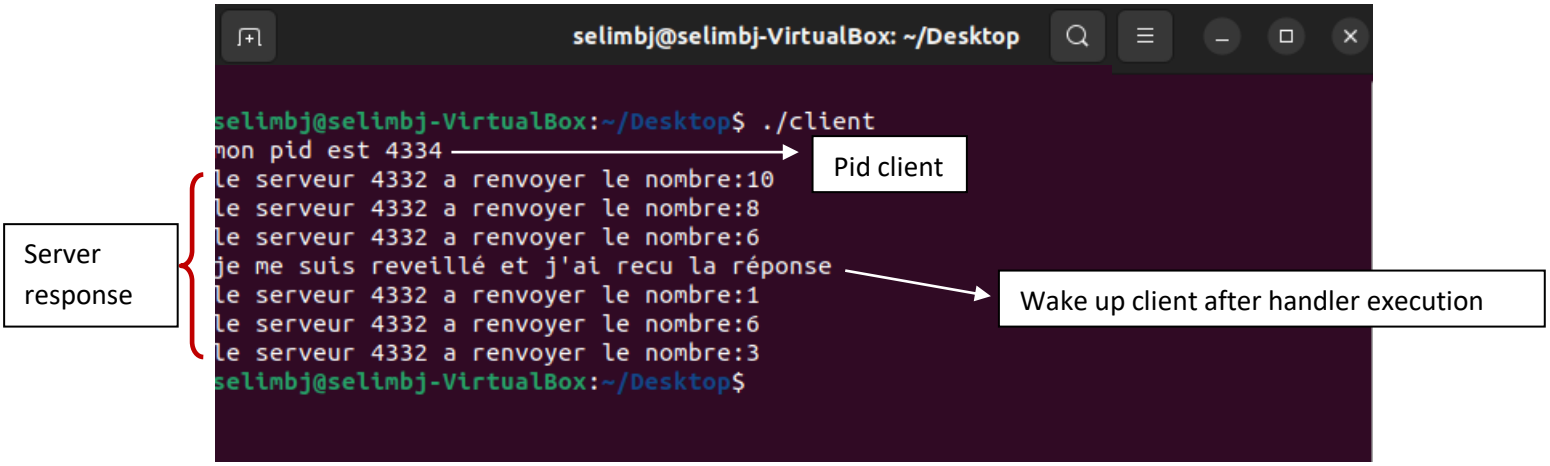
The Make File: use to automatically compile client and server files:

```
all :
                gcc client.c -o client
                gcc serveur.c -o serveur
```

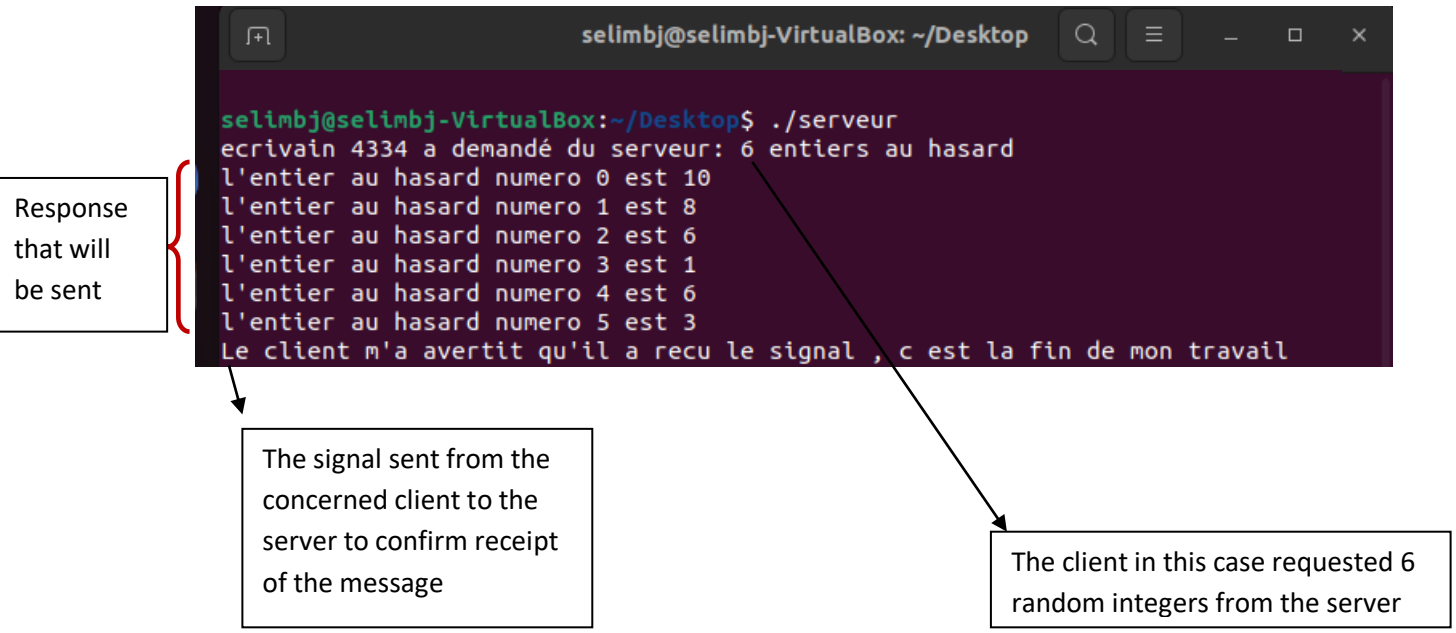
Executing the make file :

```
selimbj@selimbj-VirtualBox:~/Desktop$ make
gcc client.c -o client
gcc serveur.c -o serveur
```

Execution of the **Client** :



Execution of the **Server** :



RQ: The server is still listening to serve other clients who may request random numbers.

Methode 2 : TCP sockets

Clients

1- Set NMax

2-Generation of a random number between 1 and NMax

3-Definition of a Customer Data structure

4- Creation of the Client socket

5- Association and relationship of @ips and ports

9- Connection to the server

11- Data transfer to server

15- Receive data from server

16- Closing the socket and disconnecting from the server

SOCKET

Server

6- Creation of the server socket

7- Association and relationship of @ips and ports

8- Listen to the server

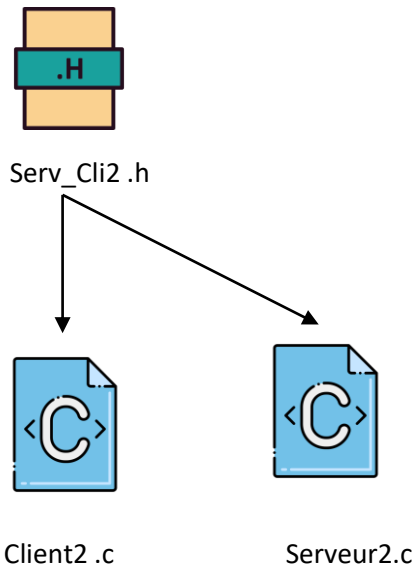
10- Accept the connection request, creation of the listening socket and fork the server

12- Receipt of the structure send from the client

13- Processing and generating a given number of random numbers

14- Send responses to client and close sockets

The Socket Backend is formed by 3 files:



The code of the Serv_Cli2.h file: will be included in Client2.c and Server2.c

The libraries needed for Client.c and Server.c files

```
1 #include <stdio.h> //input et output
2 #include<stdlib.h> //exlt
3 #include<string.h> //bezero rensent
4 #include<unistd.h>//read revceive
5 #include<sys/types.h>
6 #include<sys/socket.h>//creation de socket
7 #include<netinet/in.h>//domaine internet
8 #include<arpa/inet.h>//@ip
9 #include<signal.h>
10
11 struct ClientData {
12   int mypid;
13   int monentier;
14   int nmax;
15 };
16 struct ServerData {
17   int mypid;
18   int monentier;
19 };
20 int NMax=10;
21 FILE *fh;
```

Structure Data Client which will be sent via the socket to the server which contains the pid of the client, an integer which is the number of random numbers requested from the server, and Nmax

Server data structure that will be sent to the client that contains the server's pid, an integer that is one of the random numbers requested from the server.

Client2.c file code :

```c
#include"serv_cli2.h"
int main()
{
int client_sock;
struct sockaddr_in addr;
/*Déclarations*/
struct ClientData CD;
struct ServerData SD;
CD.mypid=getpid();


//creation socket
client_sock = socket(AF_INET,SOCK_STREAM,0);
printf("client socket created succefully\n");
//connection au serveur
memset(&addr,'\n',sizeof(addr));
addr.sin_family = AF_INET;
addr.sin_port = htons(5000);
addr.sin_addr.s_addr=inet_addr("127.0.0.3");
bind(client_sock,(struct sockaddr*)&addr,sizeof(addr));
connect(client_sock,(struct sockaddr*)&addr,sizeof(addr));
printf("connected to server\n");
//transfert des données
/*Construction et envoie d'une question*/
printf("mon pid est %d \n",CD.mypid);
CD.nmax=NMax;
srand(getpid());
CD.monentier= 1 +rand()%(CD.nmax-1+1);
send(client_sock,&CD,sizeof(CD),0);


for (int compteur = 0 ; compteur < CD.monentier ; compteur++)
{
read(client_sock,&SD,sizeof(SD));
/*Traitement locale de la réponse*/
printf("le serveur %d a renvoyer le nombre:%d \n",SD.mypid,SD.monentier);
}


//fermerture socket
close(client_sock);
printf("Disconnected from the server\n");
return 0;
}
```

Include File

Declaration of necessary structures

Customer socket

Initialization @ and port

Binding and cnx to server

Generation of the random number and storage in the structure

Send structure to server

Reading server responses

Closing the socket and disconnecting from the server

Serveur2.c file code :

```c
#include"serv_cli2.h"
int main()
{/*Declaration*/
struct ClientData CD;
struct ServerData SD;
int server_sock,new_sock;
struct sockaddr_in server_addr , new_addr;
/*Initialisation du generateur aleatoirede nombres*/
srand(getpid());
SD.mypid=getpid();
//creation socket
server_sock = socket(AF_INET,SOCK_STREAM,0);
printf("server socket created succefully\n");


//association @ip+n°port
memset(&server_addr,'\n',sizeof(server_addr));
server_addr.sin_family = AF_INET;
server_addr.sin_port = htons(5000);
server_addr.sin_addr.s_addr=inet_addr("127.0.0.3");
int b=bind(server_sock,(struct sockaddr*)&server_addr,sizeof(server_addr));
if(b<0)
{perror("Bind error\n");
exit(1);}
printf("Bind to port number\n");
//ecoute des cnx
listen(server_sock,5);
printf("listening ...\n");
//accept la cnx
while(1){socklen_t addr_size =sizeof(new_addr);
new_sock = accept(server_sock,(struct sockaddr*)&new_addr,&addr_size);
printf("client connected\n");
//transfert des données
/*Lecture d'une question*/
recv(new_sock,&CD,sizeof(CD),0);
/*Envoie de la reponse*/
printf("ecrivain %d a demandé du serveur: %d entiers au hasard\n",CD.mypid,CD.monentier);
for (int compteur = 0 ; compteur < CD.monentier ; compteur++)
{SD.monentier = 1 +rand()%(CD.nmax-1+1);
printf("l'entier au hasard numero %d est %d \n",compteur,SD.monentier);
write(new_sock,&SD,sizeof(SD));}
//fermerture socket
close(new_sock);
close(server_sock);
printf("client disconnected \n");
exit(0);}
return 0;
```

Include File

Declaration of necessary structures

Initialization of the pid and the random number generator

Creation of the server socket

Initialization @ and port

Binding

Make the server listen to client requests

Acceptance of customer requests and socket creation

Reception of customer requests

Generate random numbers and send responses to the customer concerned

Closing sockets

Rq: In the previous code, we explained the code of the business side of the project which concerns method 2 (The sockets), and in the main code attached to this report you will find lines of code containing the creation and the opening of client and server output text files to facilitate communication with the Front End side.

```c
//creation socket
server_sock = socket(AF_INET,SOCK_STREAM,0);
fh= fopen("outputserver2.txt","w");
if(fh==NULL)
{
printf("error opening file");
exit(1);
}
fputs ("server socket created succefully\n",
fclose(fh);
printf("server socket created succefully\n")
```

Serveur2 Side:

```c
//creation socket
client_sock = socket(AF_INET,SOCK_STREAM,0);
fh= fopen("outputclient2.txt","w");
if(fh==NULL)
{
printf("error opening file");
exit(1);
}
fputs ("client socket created succefully\n",fh
fclose(fh);
printf("client socket created succefully\n");
```

Client2 Side:

The Make File: use to automatically compile client and server files:

```
all :
                gcc client.c -o client
                gcc serveur.c -o serveur
                gcc client2.c -o client2
                gcc serveur2.c -o serveur2
```

Execution of the make file :

```
selimbj@selimbj-VirtualBox:~/Desktop$ make
gcc client.c -o client
gcc serveur.c -o serveur
gcc client2.c -o client2
gcc serveur2.c -o serveur2
selimbj@selimbj-VirtualBox:~/Desktop$
```

Execution of **Client2** :

```
selimbj@selimbj-VirtualBox:~/Desktop$ ./client2
client socket created succefully
connected to server
mon pid est 5913
le serveur 5885 a renvoyer le nombre:4
le serveur 5885 a renvoyer le nombre:4
le serveur 5885 a renvoyer le nombre:7
le serveur 5885 a renvoyer le nombre:9
Disconnected from the server
selimbj@selimbj-VirtualBox:~/Desktop$
```

Cnx to the server

Pid client

Response from server

Disconnect from the server

Execution of **Serveur2** :

```
selimbj@selimbj-VirtualBox:~/Desktop$ ./serveur2
server socket created succefully
Bind to port number
listening ...
client connected
ecrivain 5913 a demandé du serveur: 4 entiers au hasard
l'entier au hasard numero 0 est 4
l'entier au hasard numero 1 est 4
l'entier au hasard numero 2 est 7
l'entier au hasard numero 3 est 9
client disconnected
```

Connected client

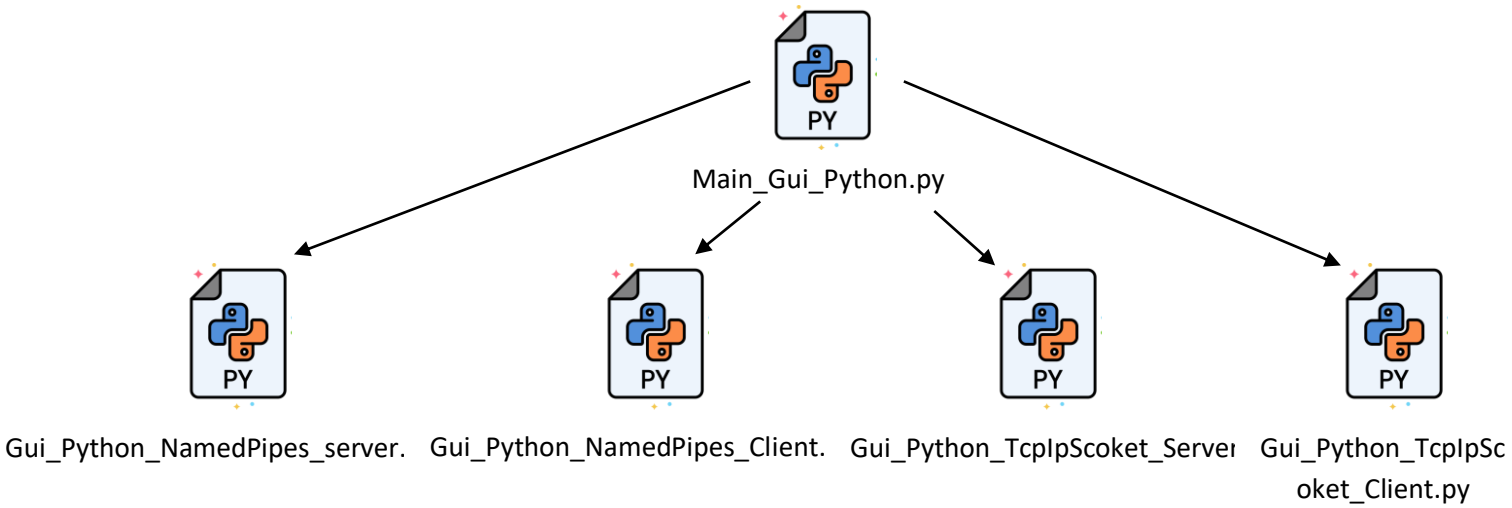
Response from server

Disconnect from client

RQ: The server remains listening to serve other clients who may request random numbers

Front end :

The user will manipulate the application using a graphical interface written in python code distributed as follows:



The code of **mainguipython.py** file :

```
1 from tkinter import *
2 from subprocess import call
3 import time
4
5 def methode1buttons():
6         btn1 = Button(window, text='client', width=10,height=0, bd='1',command=openmethode1clientfile)
7         btn1.place(x=150, y=200)
8         btn2 = Button(window, text='serveur', width=10,height=0, bd='1',command=openmethode1serveurfile)
9         btn2.place(x=630, y=200)
10 def methode2buttons():
11         btn1 = Button(window, text='client', width=10,height=0, bd='1',command=openmethode2clientfile)
12         btn1.place(x=1150, y=200)
13         btn2 = Button(window, text='serveur', width=10,height=0, bd='1',command=openmethode2serveurfile)
14         btn2.place(x=1570, y=200)
15 def openmethode1clientfile():
16         call(["python3","./guipythonmethode1client.py"])
17         time.sleep(1)
18         with open('outputserver1.txt') as f:
19                 contentsserver = f.read()
20                 f.close()
21         Output = Text(window, height = 20,width = 60, bg = "light cyan",wrap="none")
22         Output.insert(END, contentsserver)
23         Output.config(state=DISABLED)
24         Output.place(x=430,y=260)
25         with open('outputclient1.txt') as f:
26                 contentsclient = f.read()
27                 f.close()
28         Output = Text(window, height = 20,width = 45, bg = "light cyan",wrap="none")
29         Output.insert(END, contentsclient)
30         Output.config(state=DISABLED)
31         Output.place(x=20,y=260)
32         # Display image
33         canvas1.create_image( 170, 200, image =imgtubes, anchor = "nw")
34 def openmethode1serveurfile():
35         call(["python3","./guipythonmethode1serveur.py"])
36 def openmethode2clientfile():
37         call(["python3","./guipythonmethode2client.py"])
38         time.sleep(0.4)
39         with open('outputserver2.txt') as f:
40                 contentsserver2 = f.read()
41                 f.close()
42         Output = Text(window, height = 20,width = 55, bg = "light cyan",wrap="none")
43         Output.insert(END, contentsserver2)
44         Output.config(state=DISABLED)
```

Opening files containing run results

Opening files

```
        Output.place(x=1380,y=260)
        with open('outputclient2.txt') as f:
                contentsclient2 = f.read()
                f.close()
        Output = Text(window, height = 20,width = 45, bg = "light cyan",wrap="none")
        Output.insert(END, contentsclient2)
        Output.config(state=DISABLED)
        Output.place(x=1000,y=260)
        # Display image
        canvas1.create_image( 1100, 100, image =imgsocket, anchor = "nw")
def openmethode2serveurfile():
        call(["python3","./guipythonmethode2serveur.py"])
window= Tk()
# Create Canvas
canvas1 = Canvas( window, width = 400,
                 height = 400)

canvas1.pack(fill = "both", expand = True)
window.title("Projet Unix 22/23")
window.geometry("1080x720")
btn1 = Button(window, text='Communication Tubes Nommés', width=30,height=0, bd='1',command=methode1buttons
btn1.place(x=310, y=130)
btn2 = Button(window, text='Communication Socket TCP', width=30,height=0, bd='1',command=methode2buttons)
btn2.place(x=1250, y=130)
# Add Text
canvas1.create_text( 1000, 40,fill="#811d36",font="Arial 20 ", text = "Bienvenue dans le projet unix 22/23
# Add image file
imgsocket = PhotoImage(file = "socket.png")
# Create Canvas
canvas1 = Canvas(window ,width = 1000,
                 height = 1000)
canvas1.pack(fill = "both", expand = True)
# Add image file
imgtubes = PhotoImage(file = "tubes.png")
window.mainloop()
```

Main interface

The code of the Gui_Python_NamedPipes_server.py file: Execution of the back end server .c file:

```
#!/usr/bin/python
import subprocess
Commande="\" ./serveur; exec bash\""
terminal1 = subprocess.Popen(f"gnome-terminal --tab -- bash -c {Commande}",shell=True)
```

The code of the Gui_Python_NamedPipes_Client.py file: Execution of the .c back end client file:

```
#!/usr/bin/python
import subprocess
Commande="\" ./client; exec bash\""
terminal1 = subprocess.Popen(f"gnome-terminal --tab -- bash -c {Commande}",shell=True)
```

The code of the Gui_Python_TcpIpScoket_Server.py file: Execution of the .c backend_server2_file:

```
#!/usr/bin/python
import subprocess
Commande="\" ./serveur2; exec bash\""
terminal1 = subprocess.Popen(f"gnome-terminal --tab -- bash -c {Commande}",shell=True)
```

The code of the Gui_Python_TcpIpScoket_Client.py file: Execution of the .c back end client2 file:

```
#!/usr/bin/python
import subprocess
Commande="\" ./client2; exec bash\""
terminal1 = subprocess.Popen(f"gnome-terminal --tab -- bash -c {Commande}",shell=True)
```

Running the GUI: