REPUBLIQUE TUNISIENNE

MINISTERE DE L'ENSEIGNEMENT SUPERIEURE, RECHERCHE SCIENTIFIQUE ET TECHNOLOGIQUE

****

DIRECTION GENERALE D'ETUDES TECHNOLOGIQUES

***** *****

INSTITUT SUPERIEURE D'INFORMATIQUE

RAPPORT DU MINI PROJET PROGRAMMATION SYSTEME ET RESEAU SOUS UNIX

Sujet :

APPLICATION CLIENT/SERVEUR

Elaboré par :

KAROUI Sirine 2ING02

Professeur :

Mr ZAGROUBA Ezzeddine

Année universitaire : 2022/2023

Contexte Générale du projet

Partie I :

Nous souhaitons réaliser un système de communication **client/serveur**. L'architecture de ce système correspond au schéma général suivant : Un serveur attend des questions de clients dans un tube nommé **fifo1**, une question correspond à la demande d'envoi de n nombres tirés au sort par le serveur (n est un nombre aléatoire compris entre 1 et **NMAX** tiré au sort par le client).

Dans sa question, le client envoi également son numéro de telle sorte que le serveur peut le réveiller par l'intermédiaire du signal **SIGUSR1** quand il a écrit la réponse. En effet plusieurs clients pouvant être en attente de réponse dans le même tube nommé **fifo2**. Il est nécessaire de définir un protocole assurant que chaque client lit les réponses qui lui sont destinées. Le client avertit par ce même signal le serveur quand il a lu les réponses.

Partie II :

- Développerment d'une application client/serveur à l'aide des sockets (au choix UDP ou TCP) qui reprend l'énoncé de la partie I du mini projet (faite avec les tubes nommés). Le serveur est multi-client et recurrent (parallèle).
- Développement d' une interface graphique permettant d'integrer les deux parties.

Chapitre 1 :

Environnement de développement

Introduction

Dans cette partie, nous allons présenter l'environnement matérielle et logicielle relatif à la réalisation de notre projet.

Environnement matériel

Durant la réalisation de notre projet, nous avons utilisé un PC ayant les caractéristiques suivantes :

Modèle	ASUS
Système d'exploitation	Windows 10
Type du système	64 bits
Processeur	Intel® Core™ i7-7500U
Fréquence processeur	2.29 GHz
Mémoire installée (RAM)	8.00 Go

Tableau 1 – Propriétés du PC

Environnement Logiciel

Dans cette partie, nous allons présenter les différents outils, langage de développement et frameworks que nous avons utilisé pour la réalisation de notre projet :

Langage C

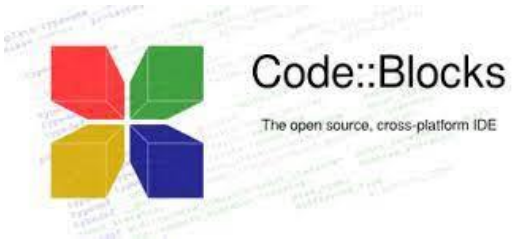
C est un **langage de programmation procédural et généraliste**. Il est qualifié de langage de bas niveau dans le sens où chaque instruction du langage est conçue pour être compilée en un nombre d'instructions machine assez prévisible en termes d'occupation mémoire et de charge de calcul.

VMware Workstation Player

VMware Workstation Player est un **outil qui permet de créer des machines virtuelles afin d'y installer un système d'exploitation différent de celui de la machine hôte**. Il prend en charge plus de 200 systèmes d'exploitation (Linux, anciennes versions de Windows, BSD).

CodeBlocks

Code::Blocks est un **environnement de développement intégré libre et multiplate-forme**. Il est écrit en C++ et utilise la bibliothèque wxWidgets.



Glade

Glade est un outil interactif de conception d'interface graphique GTK. Il prend en charge toute la partie de gestion/génération de l'interface pour permettre au développeur de se concentrer sur le code « utile ».

Glade enregistre les interfaces graphiques en générant des fichiers XML.

Conclusion

Dans ce chapitre on a définit les outils nécessaire pour la mise en œuvre de notre solution. Dans le chapitre suivant nous allons étudier la phase d'analyse et de spécification des besoins.

Chapitre 2 :

Spécifications des exigences

Introduction

Dans ce chapitre nous allons analyser et spécifiés les besoins de notre système : besoins fonctionnels qui expriment les fonctionnalités concrètes de notre application et les besoins non fonctionnels, besoins/contraintes liés à l'implémentation.

Les besoins Fonctionnels

Il s'agit des fonctionnalités du système suivant :

Pour les Tubes nommées : un point de rendez-vous

- Création des tubes nommés
- Ouverture des tubes nommés
- Le serveur en attente de question de client
- Envoi de la question par le client au serveur
- Réception de la question du client chez le serveur
- Client en attente de la réponse du serveur
- Traitement de la question par le serveur
- Le serveur envoi de signal de réveil SIGUSR1 vers le client
- Envoi de la réponse vers le client destiné
- Le client reçoit la réponse du serveur
- Le client envoi le signal SIGUSR1 au serveur pour l'avertir qu'il a lu la réponse

Pour les sockets : communication à une adresse unique

- création des adresses client et serveur
- Création de socket
- faire associer l'adresse du serveur à un port
- connexion au serveur
- serveur en attente de question de client sur un port précis
- ouverture et connexion au socket chez le client
- Envoi de la question au serveur
- Traitement de question et envoi de la réponse au client
- Réception de la réponse su serveur par le client

Les besoins non Fonctionnels

Il s'agit des besoins qui caractérisent le système, notre application devra assurer ces besoins :

- Ergonomie : Le système doit être facile à utilisé, les interfaces doivent être simples et claires.
- Performance : L'application doit être performante et optimisé à travers les fonctionnalités et répond à tous les exigences des usagers d'une manière optimale.

Conclusion

Au cours de ce chapitre, on a défini les besoins fonctionnels et non fonctionnels de notre système. Dans le chapitre suivant nous allons modéliser les différentes parties du cycle de vie.

Chapitre 3 :

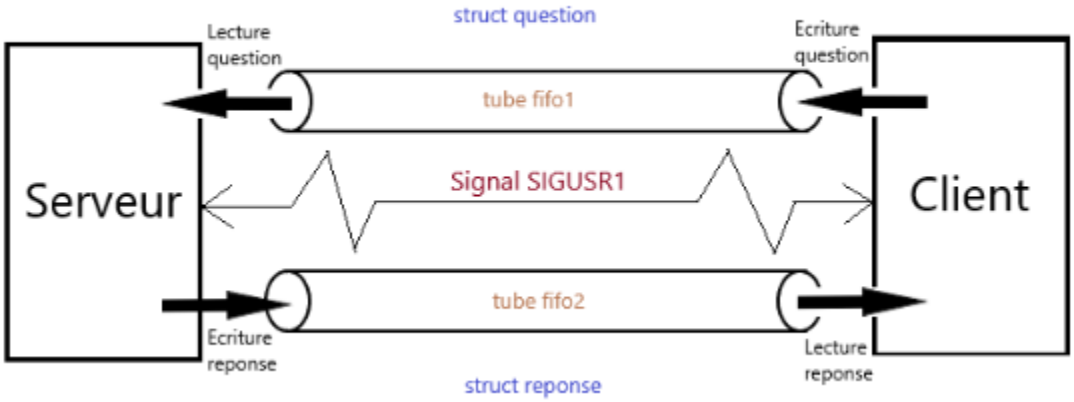
Conception

Introduction

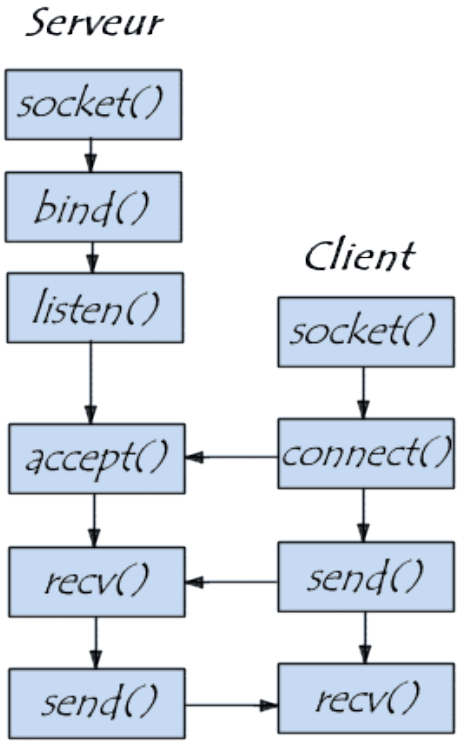
Maintenant qu'on a déterminé les différentes exigences du projet, au cours de ce chapitre on illustre le mécanismes introduits dans le système.

Présentation du mécanisme :

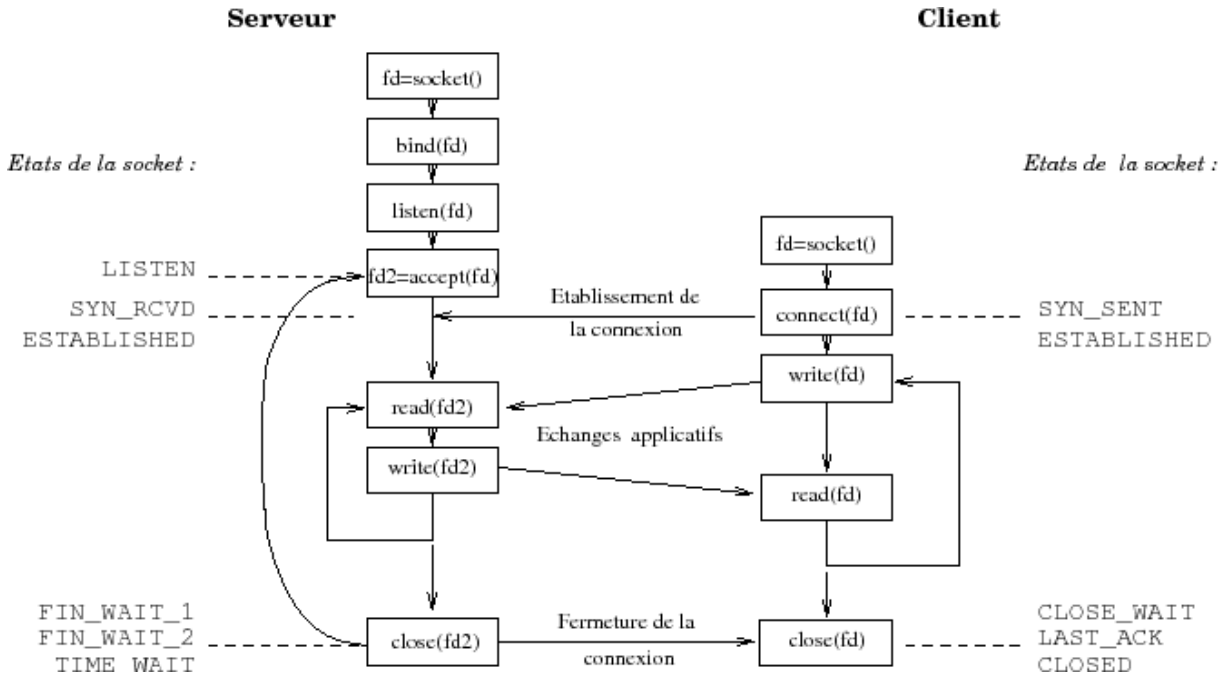
Mécanisme des Tubes Nommées (FIFO) :



Mécanisme des Sockets :



Socket Client/Serveur TCP :



Conclusion

Dans ce chapitre on a présenter les différentes mécanismes introduits dans notre projet et comment ils fonctionnent.

Chapitre 4 :

Réalisation

Introduction

Après avoir terminé la partie conceptuelle du projet, ce dernier chapitre consiste à présenter des extrais du code de programmation et des captures d'écran des interfaces de l'application réalisé.

Les interfaces de notre système

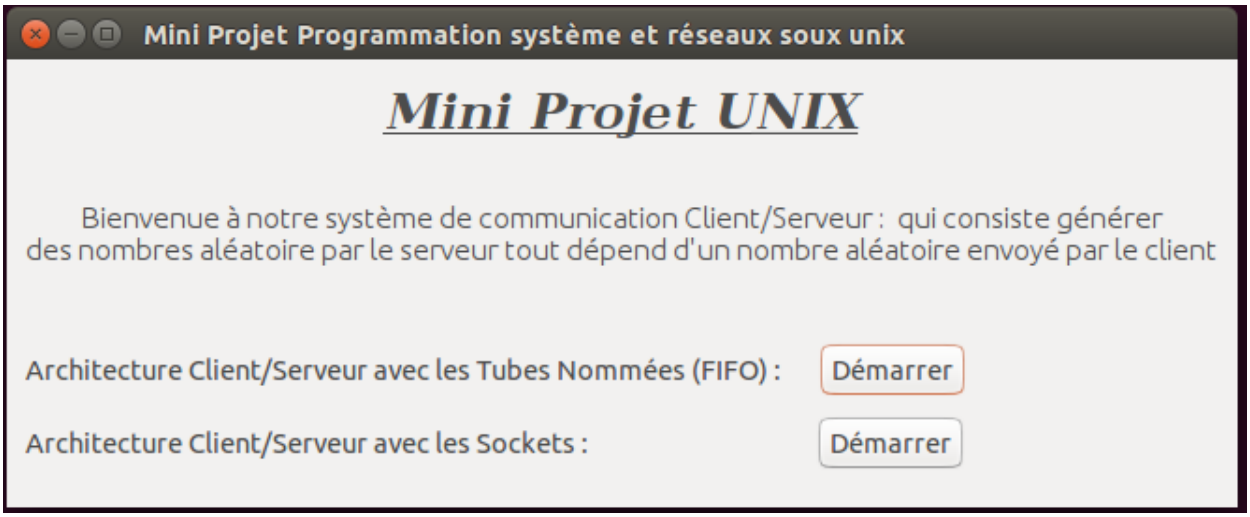


Figure : Interface de MENU

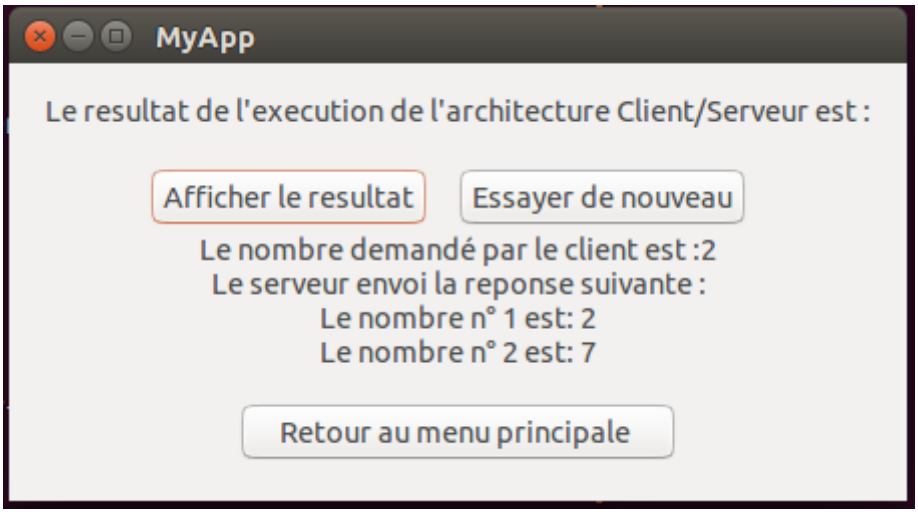


Figure : Interface résultat exécution par Sockets

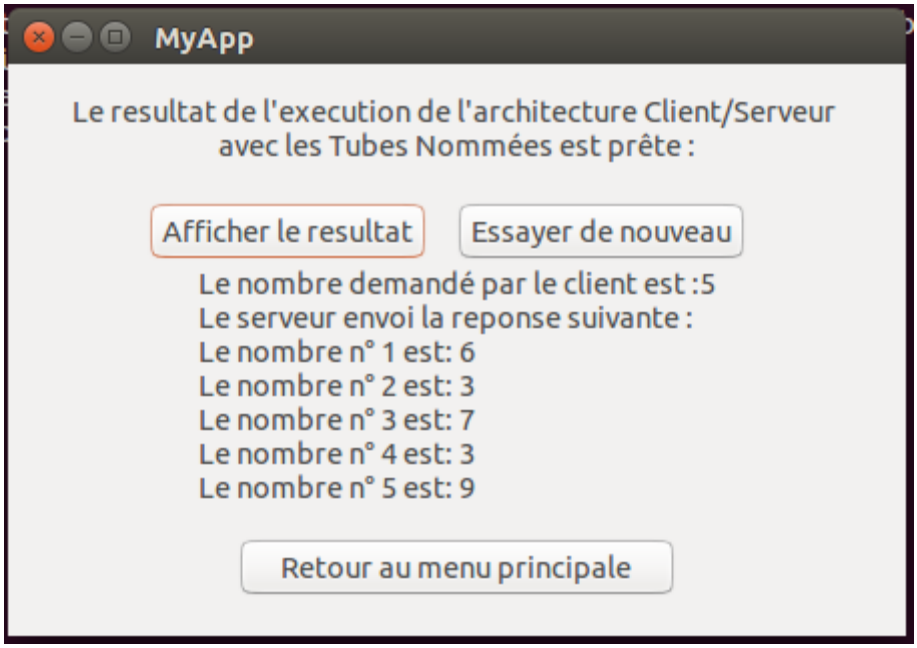


Figure : Interface résultat exécution par tubes nommés

Fichiers du système et leurs rôles

Pour établir ce projet, on a générer les fichiers suivantes :

Tubes Nommées	Sockets	Glade
client_fifo.c : code principale du processus client fifo	**client.c** : code principale du processus client socket	**Linux_App.glade** : fichier de gestion de l'interface graphique
server_fifo.c : code principale du processus serveur fifo	**server.c** : code principale du processus serveur socket	**myapp.c** : contient le main de l'interface graphique
handlers_serv.h		**myapp.h**
serv_cli_fifo.h		
fifo.txt : fichier contient le résultat de la communication client/serveur avec tube nommés	**client.txt** : fichier contient le résultat de la communication client/serveur avec socket	**makefile.txt** : fichier qui automatise l'exécution de notre système contient les commandes d'exécution des différentes fichiers système

Tableau 2 – Fichiers du système

Extraits et Explication du code

Tubes Nommées :

- Le serveur crée les tubes nommées fifo1 et fifo2 et les ouvres et se met en attente jusqu'à la réception d'une question de la part d'un client
- Le client ouvre les tube fifo1, fifo2, il crée un nombre aléatoire et l'écrit dans tube fifo1 et attendre la réponse
- Le serveur lit la question et génère des nombres aléatoire selon le nombre reçu et envoi la réponse au client concerné en le réveillant par l'intermédiaire du signal SIGUSR1
- Le client reçoit le signal, il se réveil et lire dans le tube fifo2, il envoi le signal SIGUSR1 au serveur pour l'avertir de sa lecture de la réponse
- Le serveur s'atteint lors du réception d'un signal quelconque ou du signal du ctrl+c

handlers_serv.h

- Les importations des bibliothèques nécessaires pour l'utilisation des différentes fonctions prédéfinis.
- **void hand_reveil (int sig) :** Réveil suite à la reception du signal SIGUSR1
- **void fin_serveur (int sig) :** Fin du serveur suite à la réception d'un signal quelconque

```
#ifndef handlers_serv
#define handlers_serv

 # include <stdio.h>
 # include <stdlib.h>
 # include <unistd.h>
 # include <sys/types.h>
 # include <sys/stat.h>
 # include <sys/fcntl.h>
 # include <signal.h>

void hand_reveil ( int sig )
 {
 printf("SIGNAL DE REVEIL \n");
 return ;
 }

  void fin_serveur ( int sig )
{
 unlink ( QUESTION );
 unlink ( REPONSE );

 exit (2);
 }

 # endif
```

serv_cli_fifo.h

- contient la définition :
  - des constantes et macros communes aux clients et aux serveur (NMAX, fifo1, fifo2 ..)
  - Structures de données pour représenter une question et une réponse
  - Entêtes des fichiers systèmes nécessaires

```c
#ifndef serv_cli_fifo
#define serv_cli_fifo

 # include <stdio.h>
 # include <stdlib.h>
 # include <unistd.h>
 # include <sys/types.h>
 # include <sys/stat.h>
 # include <sys/fcntl.h>
 # include <signal.h>

 # define NMAX 20
 # define QUESTION " fifo1 "
 # define REPONSE " fifo2 "

 struct question {
 int pid_client ;
 int question ;
 };
 struct reponse {
 int pid_serveur ;
 int reponse [ NMAX ];
 };
 # endif
```

server_fifo.c

Contient le programme serveur avec les grandes fonctionnalité : création et ouverture des tubes nommées, attente et lecture des questions , écriture des réponses.

```c
# include "serv_cli_fifo.h"
#include "handlers_serv.h"


 int main ( void )
 {
  /* Declarations */
 int d_question , d_reponse ; /* Descripteurs sur les tubes*/
 int ind , sig ;
 struct question question ;
 struct reponse reponse ;
 FILE * fp;
 fp = fopen ("fifo.txt", "w+");
 mode_t mode = S_IRUSR | S_IRGRP | S_IROTH | S_IWUSR | S_IWGRP | S_IWOTH;

printf("SERVER -> je suis le serveur %d\n",getpid());
 /* Creation des tubes nommées */

 if( mkfifo ( QUESTION , mode ) == -1 || mkfifo ( REPONSE , mode ) == -1)
{
 perror(" Creation des tubes impossible \n ");
 exit (2);
 };

/*initialisation du générateur de nombres aléatoires */

 srand ( getpid ());
 reponse . pid_serveur = getpid ();
```

```c
 /* ouverture des tubes nommées */
    printf("SERVER -> Ouverture des tubes nomées \n");
 d_question = open ( QUESTION , O_RDONLY );
 d_reponse = open ( REPONSE , O_WRONLY );

  /* installation des handlers */

 for ( sig =1; sig < NSIG ; sig ++)
 signal (sig , fin_serveur );
 //le serveur est reveillé par le signal SIGUSR1
signal(SIGUSR1,hand_reveil);

 while (1){
 /* lecture d'une question */
printf("SERVER -> lecture de la question : le client %d envoi le nombre %d\n",question.pid_client,question.question);
 if( read ( d_question ,& question , sizeof ( struct question )) <= 0 )
 {
 close ( d_question );
 d_question = open ( QUESTION , O_RDONLY );
 continue ;
 }
 /* construction de la reponse */
fprintf (fp, "Le nombre demandé par le client est :%d\nLe serveur envoi la reponse suivante :\n",question.question);
 for ( ind =0; ind < question . question ; ind ++)
 { reponse . reponse [ ind ]= rand ()%10;
  printf ("%d ", reponse . reponse [ ind ]);
  fprintf (fp, "Le nombre n° %d est: %d\n",ind+1,reponse . reponse [ ind ]);

 }
```



```c
/*envoi de la reponse */
printf("SERVER -> envoi de la reponse vers client %d\n", question.pid_client);
 if( write ( d_reponse ,& reponse , sizeof ( struct reponse )) == -1)
 {
 perror ("SERVER -> erreur d'ecriture de reponse ");
 fin_serveur ( SIGUSR2 );
 }
 /* envoi du signal SIGUSR1 au client concerné */
 kill ( question . pid_client , SIGUSR1 );
 //attente de signal

 exit(0);

 }
 }
```

client_fifo.c

contient le programme client à exécuté avec 3 grandes fonctionnalité : ouverture des tubes nommées, écriture d'une question et lecture d'une réponse

```c
# include "serv_cli_fifo.h"
#include "handlers_serv.h"

 int main ( void ){
 /* Declarations */
 int d_question , d_reponse ;
 int ind ,sig;
 struct question question ;
 struct reponse reponse ;
printf("CLIENT -> Je suis le client %d\n",getpid());
/* ouverture des tubes nommées */
printf("CLIENT -> Ouverture des tubes par le client \n ");
 d_question = open ( QUESTION , O_WRONLY );
 d_reponse = open ( REPONSE , O_RDONLY );
 if( d_reponse == -1 || d_question == -1)
 {
 perror ("CLIENT -> Ouverture des tubes impossible \n ");
 exit (2);
 };

 /* installation des handlers */
for ( sig =1; sig < NSIG ; sig ++)
 signal (sig , fin_serveur );
```

```c
/*construction et envoi d'une question */
 srand ( getpid ());
 question . pid_client = getpid ();
 question . question = 1 + rand ()% NMAX ;
printf("CLIENT -> ecriture de la question : le client %d envoi le nombre %d\n",question.pid_client,question.question);
 if( write ( d_question ,& question , sizeof ( struct question )) == -1){
 perror ("CLIENT -> erreur d'ecriture de la question ");
 exit (2);
 }
printf("CLIENT -> question envoyé\n");
 /*attente de la reponse */

 signal(SIGUSR1,hand_reveil);
close(d_question);
close(d_reponse);
system("./MyApp 2 2");
 /*lecture de la reponse */

 if( read ( d_reponse ,& reponse , sizeof ( struct reponse ) ) <= 0 )
 {
  //perror (" Probleme de lecture \n ");
  printf("CLIENT -> lecture du reponse par le client %d\n", getpid());
 exit (2);
 };
 /* envoi du signal SIGUSR1 au serveur */

 kill ( reponse . pid_serveur , SIGUSR1 );
 printf("CLIENT -> envoi signal vers le serveur %d\n",reponse.pid_serveur);
```



```c
/* traitement locale de la reponse */
 printf ("CLIENT -> Reponse reçu de %d nombres aléatoires  :\n", question . question );
 for ( ind =0; ind < question . question ; ind ++)
 printf ("%d ", reponse . reponse [ ind ]);
 printf ("\n ");

 exit (0);
 }
```

Sockets

client.c

- int nombre() : calcule du nombre aléatoire

```c
#include <sys/socket.h>
#include <sys/un.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdbool.h>
#include "myapp.h"
# define NMAX 20

//char *socket_path = "./socket";
char *socket_path = "\0hidden";
FILE * fp;
int c;


int nombre()
{
        int nb;
        srand(time(NULL));
        nb=1 + rand()% NMAX;
        printf("CLIENT -> Voici le nombre:%d\n",nb);
        return nb;
}
```

- main du programme contient :
  - ouverture de la fichier client.txt pour enregistrer le resultat quand il est écrite
  - recupération de l'adresse du socket à partir de la ligne de commande
  - recupération du descripteur du socket

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

  struct sockaddr_un addr;
  char buf[100], server_reply[100];
  int fd,rc,ind;
    int res[NMAX];
    int client_actif = true;
    char ok[5];
   fp = fopen ("client.txt", "w+");
  if (argc > 1){

    socket_path='\0' + argv[1];
  } else {
    printf("CLIENT -> Veuillez spécifier un nom de socket valide en argument.\n");
    exit(1);
  }

  if ( (fd = socket(AF_LOCAL, SOCK_STREAM, 0)) == -1) {
    perror("CLIENT -> socket error");
    exit(-1);
  }
  memset(&addr, 0, sizeof(addr));
  addr.sun_family = AF_LOCAL;
  if (*socket_path == '\0') {
    *addr.sun_path = '\0';
    strncpy(addr.sun_path+1, socket_path+1, sizeof(addr.sun_path)-2);
  } else {
    strncpy(addr.sun_path, socket_path, sizeof(addr.sun_path));
  }
```

- connection de la socket à l'adresse stocké dans la structure **sockaddr** avec la fonction **connect()**
- génération de la question du client avec la fonction **nombre()**
- envoi de la quqestion au serveur avec la fonction **write()**
- lecture de la reponse sur la socket
- ecriture de la reponse sur le fichier **client.txt**
- fermeture du descripteur de la socket avec **close(fd)**

```c
  if (connect(fd, (struct sockaddr*)&addr, sizeof(addr)) == -1) {
    perror("CLIENT -> connect error");
    exit(-1);
   bzero ((char *) buf, 100);
                int question = nombre();
                buf[0] = question;
                printf("CLIENT -> Le nombre aléatoire demandé est %d, \n",question);

     printf("CLIENT -> Patientez pendant que le serveur traite votre message ... \n");
                if( write(fd , buf , strlen(buf)) < 0)
                {
                        puts("CLIENT -> Send failed");
                        exit(-1);
                }

     if( read(fd , res , 100 ) < 0)
       {
         puts("CLIENT -> recv failed");

       }
      printf("CLIENT -> Le serveur repond par %d nombre aléatoires : \n",question);
        fprintf (fp, "Le nombre demandé par le client est :%d\nLe serveur envoi la reponse suivante :\n",question);
     for( ind = 0; ind < question ; ind ++)
     {
     printf(" CLIENT -> Nombre aléatoire n°%d : %d \n",ind+1, res[ind]);
   fprintf (fp, "Le nombre n° %d est: %d\n",ind+1,res[ind]);
     }
  close(fd);
  printf("Fin\n");
  return 0;
}
```


server.c

- Importation des bibliothèques nécessaires pour l'utilisation des fonctions des sockets et signaux
- Declaration des variables :
  - les descripteurs des sockets : sockfd et newsockfb
  - Socket_path : l'adresse du socket
  - pid : l'identifiant du processus appelant
  - reponse et rep : variable de sauvegarde de la reponse à envoyé eu client

```c
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <sys/un.h>
#include <signal.h>
#include <sys/select.h>
# define NMAX 20

int client_number = 0;
int sockfd, newsockfd;
char *socket_path = "\0hidden";
char tampon [100];
pid_t pid;
int reponse [ NMAX ];
char rep[NMAX];
int ind,i;
char port[5];

//Fermer puis spprimer la socket créee, et indiquer l'arrêt du programme
void quit(){
  printf("\n SERVER ->  Signal intercepté ...\n");
  close(newsockfd);
  remove(socket_path);
  printf("SERVER ->  Arret du serveur ...\n");
  exit(0);
}

//Lire le message reçu par un client, simuler le traitement de sa demande et lui renvoyer un message
int gererClient(struct sockaddr_un cli_addr, socklen_t clilen){

   bzero ((char *) tampon, 100);
   //lire les données provient du client
   read (newsockfd, tampon, 100);
if(tampon[0] != 0){


     printf ("SERVER ->  Client n° %i a envoyé le nombre aléatoire: %d\n", client_number, tampon[0]);
   fprintf (fp, "Question :%d\n", tampon[0]);
   printf("SERVER ->  Traitement des données ...\n");

  sleep(1); //simuler une action bloquante du serveur. Les autres clients ne devraient pas
             //être bloqués grâce au fork() et les processus lourds créés.

    for ( ind =0; ind < tampon[0] ; ind ++)
        { reponse [ ind ]= rand ()%10;
        }
   printf("SERVER ->  Fin du traitement pour le client %i\n",client_number);
   //envoi du resultat au client
    printf("VEUILLEZ PATIENTEZ SVP \n");
   write (newsockfd, reponse, 100);

   sleep(2);//simuler une action bloquante du serveur pour l'ouverture de
               //l'interface conçu à l'affichage
//appel d ela commande system pour l'ouverture de l'executable de l'interface
//graphique
system("./MyApp 2");

}

}
```

```c
int main (int argc, char** argv){

 signal(SIGINT,quit);

 socklen_t clilen, servlen;
 //creation des adresses client et serveur
 struct sockaddr_un cli_addr;
 struct sockaddr_un serv_addr;

 fd_set active_fd_set, read_fd_set;

//creation et test sur le socket
 if ( (sockfd = socket(AF_LOCAL, SOCK_STREAM, 0)) < 0)
 {
   printf ("SERVER ->  Erreur de creation de socket\n"); exit (1);
 }

 bzero((char *)&serv_addr, sizeof(serv_addr));
 serv_addr.sun_family = AF_LOCAL;
 //on associe l'adresse du serveur à un port
 strcpy(serv_addr.sun_path, argv[0]);

 servlen = strlen(serv_addr.sun_path) + sizeof(serv_addr.sun_family);

//connexion au serveur
 if ( bind (sockfd, (struct sockaddr *) &serv_addr, servlen) < 0)
 {
   printf ("SERVER ->  Erreur de bind\n");
    exit (1);
 }

 listen(sockfd, 5);
 FD_ZERO(&active_fd_set);
 FD_SET(sockfd,&active_fd_set);

 int select_status;

 clilen = sizeof(cli_addr);
 read_fd_set = active_fd_set;
 while(1)
 { printf ("SERVER ->  serveur: En attente sur %s\n" ,argv[0]);
   newsockfd = accept (sockfd, (struct sockaddr *) &cli_addr, &clilen);

   pid = fork();
   client_number ++;

   switch (pid)
      {
          case -1 : printf ("SERVER ->  Erreur dans la creation du processus fils.\n");
          perror ("Erreur : ");
          break;
          case 0 : printf("SERVER ->  Arrivée d'un nouveau client (client n° %i) \n",client_number);

          gererClient(cli_addr,clilen);
          break;
          default:
          break;
      }
    }
    return 0;
}
```

Glade

myapp.c

```
//My application
// gcc -o MyApp myapp.c -Wall `pkg-config --cflags --libs gtk+-3.0` -export-dynamic
//Libraries
#include <gtk/gtk.h> // GTK Library
#include <stdio.h> // c io library
#include <sys/wait.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include "myapp.h"

//variables
GtkWidget *window, *window_socket,*window_res_socket,*window_res_fifo;

GtkLabel *socket_res,*tube_res;
FILE *fp;
int i =0;
```

```
int main(int argc, char *argv[])
{
        GtkBuilder *builder;
        gtk_init(&argc,&argv);
        builder = gtk_builder_new();
        gtk_builder_add_from_file(builder, "Linux_App.glade",NULL);

        window = GTK_WIDGET(gtk_builder_get_object(builder,"MyWindow"));
        window_socket = GTK_WIDGET(gtk_builder_get_object(builder,"window_socket"));
        window_res_socket = GTK_WIDGET(gtk_builder_get_object(builder,"window_res_socket"));
        window_res_fifo = GTK_WIDGET(gtk_builder_get_object(builder,"window_res_fifo"));
        socket_res = GTK_LABEL(gtk_builder_get_object(builder,"socket_res"));
        tube_res = GTK_LABEL(gtk_builder_get_object(builder,"tube_res"));


        gtk_builder_connect_signals(builder,NULL);
        g_object_unref(builder);

        if(argc == 1) {

        gtk_widget_show_all(window);
        }else if(argc == 2){
        gtk_widget_show_all(window_res_socket);
        }else{
        gtk_widget_show_all(window_res_fifo);
        }
        gtk_main();
        return 0;

}
```



```
void retour_menu(){
        gtk_widget_destroy(window_res_socket);
        gtk_widget_destroy(window_res_fifo);
        gtk_widget_show_all(window);
}
void exit_app()
{
        printf("Exit app \n");
        gtk_main_quit();
}


void afficher_socket(){
char string[100];
char s[500];

    FILE *fp;
    fp=fopen("client.txt","r");

        while(fgets(string,100,fp) != NULL){
    printf(" %s",string);
strcat(s,string);
}
gtk_label_set_text(socket_res,s);
//creation de bouton

 fclose(fp);
}
```

makefile.txt

```
client.o: client.c
	gcc client.c -o client
server.o: server.c
	gcc server.c -o server
client_fifo.o: client_fifo.c
	gcc client_fifo.c -o client_fifo
server_fifo.o:server_fifo.c
	gcc server_fifo.c -o server_fifo
MyApp.o: myapp.c
	gcc -o MyApp myapp.c -Wall `pkg-config --cflags --libs gtk+-3.0` -export-dynamic
```

Conclusion

Au cours de ce dernier chapitre on a présenter des captures des différentes interfaces à affichés durant l'exécution de notre système, ainsi les fichiers de développement du système et leurs rôles et enfin des captures du code avec commentaires et des brève explications des principales fonctions prédéfinies des librairies implémenté utilisés dans le développement du code.