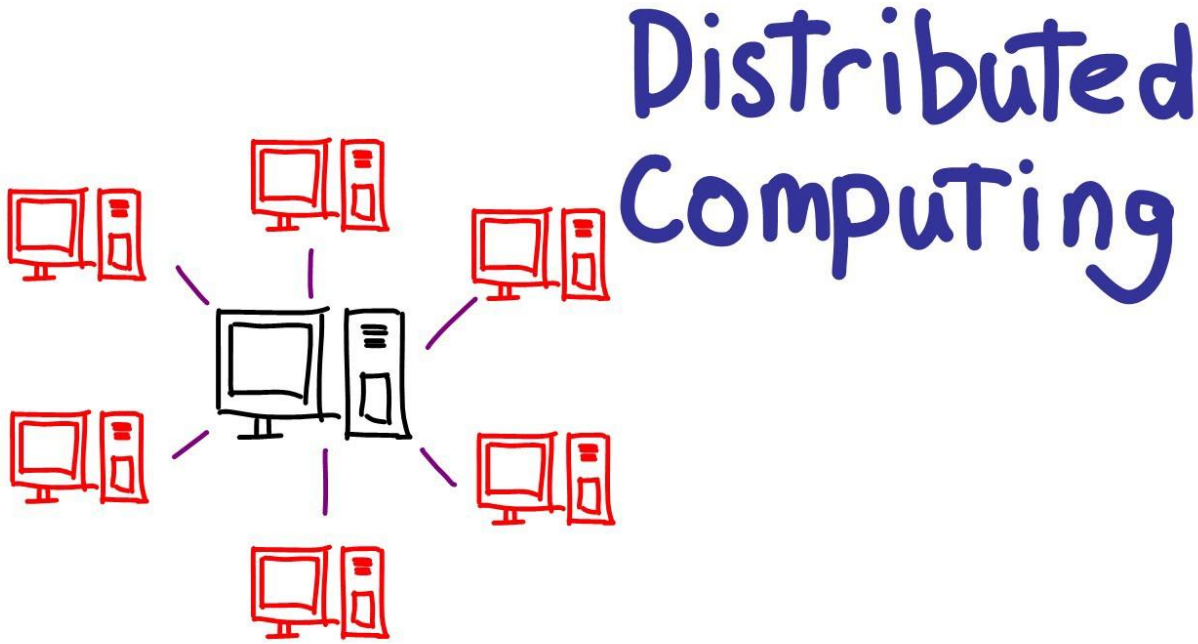
Les systèmes distribués

Compte Rendu pour les TP1,TP2,TP3 et TP4



Réalisé par : Anas BRITAL .

Encadré par : PR Hicham TOUIL

MASTER DE SYSTÈME INTELLIGENTS ET DÉVELOPPEMENT DES SYSTÈMES DÉCISIONNELS (MSIDSD)

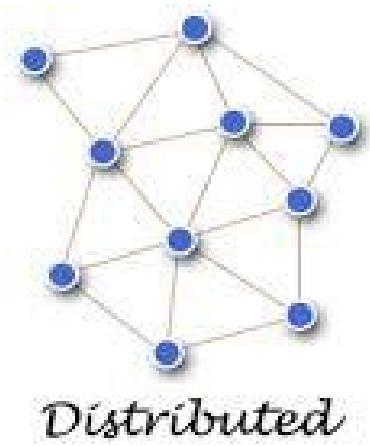
Faculté polydisciplinaire de Larache

Université Abdelmalek Essaadi

année Universitaire : 2020/2021

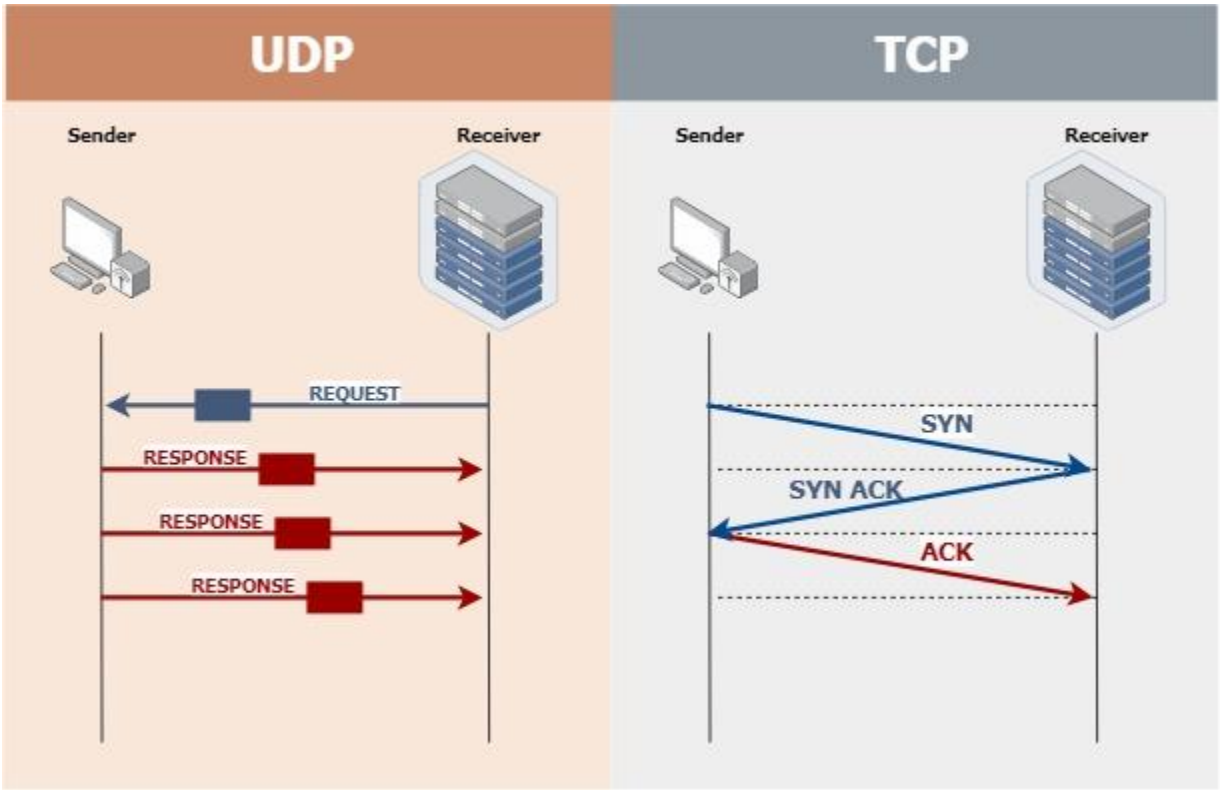
Résumé

Une architecture distribuée fait référence à un système d'information ou à un réseau pour lequel toutes les ressources disponibles ne sont pas localisées au même endroit ou sur la même machine. Ce concept, dont l'un peut être une combinaison de transmissions de type client-serveur, entre en conflit avec le concept d'architecture centralisée .



Pour pouvoir développer un système distribué, nous avons besoin des moyens de communication, et c'est l'objectif du chapitre 1 .

Dans ce rapport, je vous parle du travail que nous avons effectué dans le module des systèmes distribués, nous avons fait 4 TP et vu deux types des Socket AF_INET et AF_UNIX avec les deux mode connecté(TCP) et non connecté (UDP).



TP 1

Travail à faire : Écrivez un programme qui envoie une chaîne de lettres minuscules au serveur, le serveur convertit la chaîne en majuscules puis l'envoie au client.

Client.c

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

int main(int argc, char** argv) {
 if(argc<2)
 {
     fprintf(stderr,"Not Enough Parameters ...\n");
     exit(EXIT_FAILURE);
 }
 struct sockaddr_un SocketAddress;
 int socketDescripteur,rc;

 if ( (socketDescripteur = socket(AF_UNIX, SOCK_STREAM, 0)) == -1) {
     fprintf(stderr,"Failed to create The Socket ...\n");
     exit(EXIT_FAILURE);
 }

 memset(&SocketAddress, 0, sizeof(SocketAddress));
 bcopy(argv[1],SocketAddress.sun_path,sizeof(argv[1]));
 SocketAddress.sun_family = AF_UNIX;

   if   (connect(socketDescripteur,   (struct   sockaddr*)&SocketAddress,
sizeof(SocketAddress)) == -1) {
```

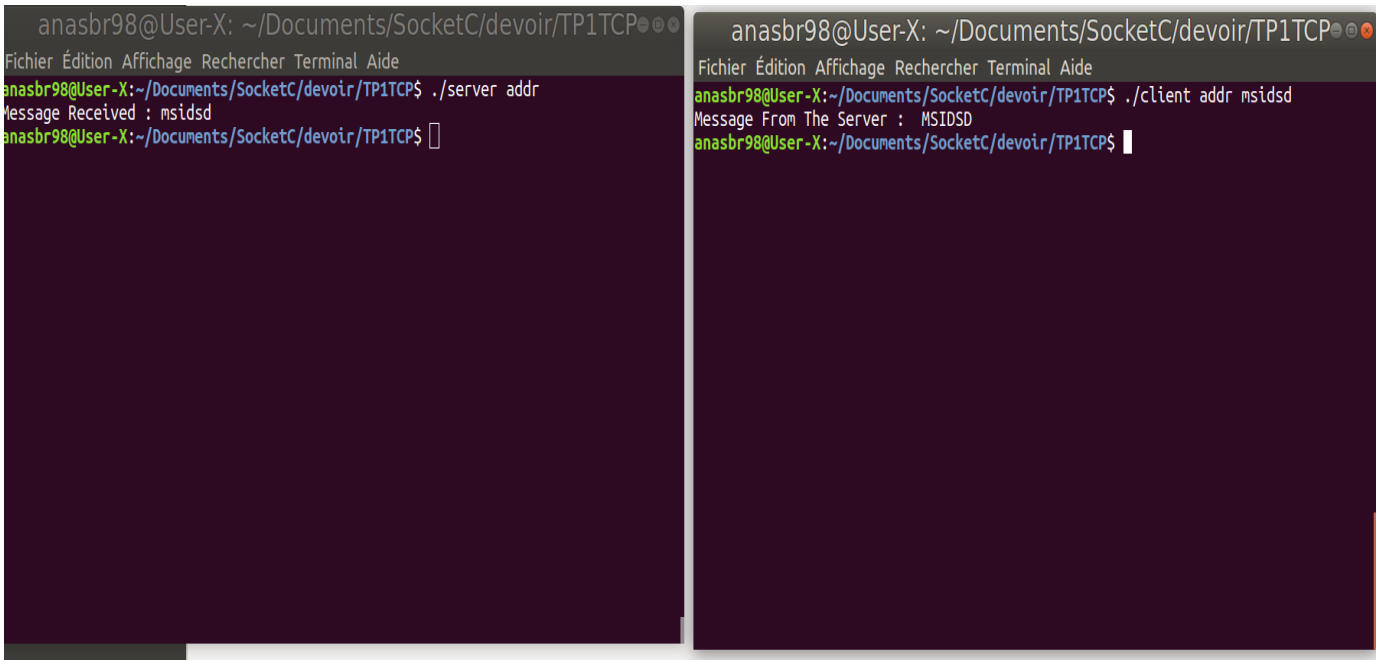
```
    fprintf(stderr,"Failed to Connect To The Server ...\n");
    exit(EXIT_FAILURE);
}

if(send(socketDescripteur,argv[2],sizeof(argv[2]),0) == -1)
{
    fprintf(stderr,"Failed to Send The Message ...\n");
    exit(EXIT_FAILURE);
}

if(recv(socketDescripteur,argv[2],sizeof(argv[2]),0) == -1)
{
    fprintf(stderr,"Failed to Receive The Message ...\n");
    exit(EXIT_FAILURE);
}

 printf("Message From The Server :  %s \n",argv[2]);
 close(socketDescripteur);

return 0;
}
```

Server.c

```
#include <stdio.h>
#include <unistd.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <stdlib.h>
#include<string.h>
void toUpperCase(char* ch)
{
   char *p;
   for(p=ch;*p!='\0';p++)
   {
       if(*p >= 'a' && *p <= 'z')
       {
           *p = *p -32;
       }
   }
}

int main(int argc, char** argv) {
 if(argc < 1)
 {
     fprintf(stderr,"Not Enough Parameters ...\n");
     exit(EXIT_FAILURE);
 }
  struct sockaddr_un SocketAddress;
 int socketDescripteur,clientDescripteur;
 char message[20];

 if ( (socketDescripteur = socket(AF_UNIX, SOCK_STREAM, 0)) == -1) {
   fprintf(stderr,"Failed Create The Socket ...\n");
   exit(EXIT_FAILURE);
 }

 memset(&SocketAddress, 0, sizeof(SocketAddress));
 SocketAddress.sun_family = AF_UNIX;
 bcopy(argv[1],SocketAddress.sun_path,sizeof(argv[1]));
       if    (bind(socketDescripteur,    (struct    sockaddr*)&SocketAddress,
sizeof(SocketAddress)) == -1) {
```

```
  fprintf(stderr,"Failed To Bind ...\n");
  exit(EXIT_FAILURE);
}

if (listen(socketDescripteur, 5) == -1) {
  fprintf(stderr,"Failed To listen ...\n");
  exit(EXIT_FAILURE);
}

if ( (clientDescripteur = accept(socketDescripteur, NULL, NULL)) == -1) {
    fprintf(stderr,"Failed To accept a new Client ...\n");
    exit(EXIT_FAILURE);
  }

if(recv(clientDescripteur,message,sizeof(message),0) == -1)
{
   fprintf(stderr,"Failed To Receive The Message ...\n");
    exit(EXIT_FAILURE);
}

printf("Message Received : %s \n",message);
toUpperCase(message);

if(send(clientDescripteur,message,sizeof(message),0) == -1)
{
  fprintf(stderr,"Failed To Send The Message ...\n");
    exit(EXIT_FAILURE);
}
close(socketDescripteur);
close(clientDescripteur);
return 0;
}
```

Exécution


anasbr98@User-X: ~/Documents/SocketC/devoir/TP1TCP
Fichier Édition Affichage Rechercher Terminal Aide
anasbr98@User-X:~/Documents/SocketC/devoir/TP1TCP$ ./server addr
Message Received : msidsd
anasbr98@User-X:~/Documents/SocketC/devoir/TP1TCP$
anasbr98@User-X: ~/Documents/SocketC/devoir/TP1TCP
Fichier Édition Affichage Rechercher Terminal Aide
anasbr98@User-X:~/Documents/SocketC/devoir/TP1TCP$ ./client addr msidsd
Message From The Server : MSIDSD
anasbr98@User-X:~/Documents/SocketC/devoir/TP1TCP$

TP 2

Travail à faire : Écrivez un programme qui lit le contenu d'un fichier et l'envoie au serveur, et le serveur lit les données et les stocke dans un fichier.

Client.c (TCP)

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<sys/un.h>
#include<string.h>
#include<unistd.h>
#include "structure.h"

void getString(char* result ,Etudiant e)
{
sprintf(result,"%d-%s-%s-%s-%s",e.numero,e.prenom,e.nom,e.cne,e.filiere);
}

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

   int SocketDescripteur;
   Etudiant e;
   FILE *PF;
   char messageToSend[100];

   struct sockaddr_un serverAddress;
   bzero((char*)&serverAddress,sizeof(serverAddress));
   serverAddress.sun_family = AF_UNIX;
   bcopy("unix_socket",serverAddress.sun_path,11);

if((SocketDescripteur  = socket(AF_UNIX,SOCK_STREAM,0)) == -1)
   {
```

```
        perror("Failed to create Socket ...\n");
        exit(EXIT_FAILURE);
    }

    if(connect(SocketDescripteur,(struct
sockaddr*)&serverAddress,sizeof(serverAddress)) == -1)
    {
    perror("Failed to Connect ...\n");
    exit(EXIT_FAILURE);
    }
    printf("The connection has been established successfully \n");

    PF = fopen("file1.txt","r");
    if(PF == NULL)
    {
        perror("Failed to Open The file");
        exit(EXIT_FAILURE);
    }

    while(!feof(PF))
    {

fscanf(PF,"%d\t%s\t%s\t%s\t%s\n",&e.numero,e.prenom,e.nom,e.cne,e.filiere)
;
    getString(messageToSend,e);
    write(SocketDescripteur,messageToSend,sizeof(messageToSend));
    }
    fclose(PF);
    close(SocketDescripteur);

    return EXIT_SUCCESS;
}
```

Server.c (TCP)

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<sys/un.h>
#include<string.h>
#include<unistd.h>
#include "structure.h"

void getEtudiant(char *str,Etudiant *e)
{
   int i=0;
   const char *delimiteur = "-";
   char* token;

   token = strtok(str, delimiteur);
    while (token != 0) {
       switch (i)
   {
   case 0 :
       e->numero = atoi(token);
       break;
   case 1 :
       strcpy(e->prenom,token);
       break;
   case 2 :
       strcpy(e->nom,token);
       break;
   case 3 :
       strcpy(e->cne,token);
       break;
   case 4 :
       strcpy(e->filiere,token);
       break;
   }
   i++;
       token = strtok(0, delimiteur);
   }  }
```

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

   //déclaration des variables
   int ServerDescipteur,ClientDescripteur,longSocketClient;
   char messageReceived[100];
   Etudiant e;
   FILE *PF;
   struct sockaddr_un AddressServer,AddressClient;
   bzero((char*)&AddressServer,sizeof(AddressServer));
   AddressServer.sun_family = AF_UNIX;
   bcopy("unix_socket",AddressServer.sun_path,11);
   if((ServerDescipteur = socket(AF_UNIX,SOCK_STREAM,0)) == -1)
   {
   perror("Failed to Create Socket ... \n");
   exit(EXIT_FAILURE);
   }
   if(bind(ServerDescipteur,(struct
sockaddr*)&AddressServer,sizeof(AddressServer)) == -1)
   {
       perror("Failed to Bind ... \n");
       exit(EXIT_FAILURE);
   }
   if(listen(ServerDescipteur,5) == -1)
   {
   perror("Failed to listen ... \n");
   exit(EXIT_FAILURE);
   }
   longSocketClient = sizeof(AddressClient);
   if((ClientDescripteur = accept(ServerDescipteur,(struct sockaddr
*)&AddressClient,&longSocketClient)) == -1)
   {
       perror("Failed to connect to client ... \n");
       exit(EXIT_FAILURE);
   }
   printf("The connection has been established successfully \n");
   PF = fopen("file2.txt","w");
   if(PF == NULL)
   {
       perror("Failed to Open The file");
       exit(EXIT_FAILURE);}
```

```
while(read(ClientDescripteur ,messageReceived,sizeof(messageReceived))>0)
{
getEtudiant(messageReceived,&e);
fprintf(PF,"%d\t%s\t%s\t%s\t%s\n",e.numero,e.prenom,e.nom,e.cne,e.filiere;
}

   close(ServerDescipteur);
   close(ClientDescripteur);
   fclose(PF);
   unlink("unix_socket");

   return EXIT_SUCCESS;
}
```

Client.c (UDP)

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<sys/un.h>
#include<string.h>
#include<unistd.h>
#include "structure.h"

void getString(char* result ,Etudiant e)
{
sprintf(result,"%d-%s-%s-%s-%s",e.numero,e.prenom,e.nom,e.cne,e.filiere);
}

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

   //Declaration des Variables
   int SocketDescripteur;
   Etudiant e;
   FILE *PF;
   char messageToSend[100];
```

```
   struct sockaddr_un serverAddress;
   bzero((char*)&serverAddress,sizeof(serverAddress));
   serverAddress.sun_family = AF_UNIX;
   bcopy("unix_socket",serverAddress.sun_path,11);
   if((SocketDescripteur  = socket(AF_UNIX,SOCK_DGRAM,0)) == -1)
   {
       perror("Failed to create Socket ...\n");
       exit(EXIT_FAILURE);
   }

   printf("The connection has been established successfully \n");

   PF = fopen("file1.txt","r");
   if(PF == NULL)
   {
       perror("Failed to Open The file");
       exit(EXIT_FAILURE);
   }

   while(!feof(PF))
   {

fscanf(PF,"%d\t%s\t%s\t%s\t%s\n",&e.numero,e.prenom,e.nom,e.cne,e.filiere)
;
   getString(messageToSend,e);
   printf("message to send : %s\n",messageToSend);
   sendto(SocketDescripteur,messageToSend,sizeof(messageToSend),0,(struct
sockaddr*)&serverAddress,sizeof(serverAddress));
   }
   //sleep(100);
   fclose(PF);
   close(SocketDescripteur);
   return EXIT_SUCCESS;
}
```

Server.c (UDP)

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<sys/un.h>
#include<string.h>
#include<unistd.h>
#include "structure.h"

void getEtudiant(char *str,Etudiant *e)
{
   int i=0;
   const char *delimiteur = "-";
   char* token;

   token = strtok(str, delimiteur);
    while (token != 0) {
       switch (i)
   {
   case 0 :
       e->numero = atoi(token);
       break;
   case 1 :
       strcpy(e->prenom,token);
       break;
   case 2 :
       strcpy(e->nom,token);
       break;
   case 3 :
       strcpy(e->cne,token);
       break;
   case 4 :
       strcpy(e->filiere,token);
       break;
   }
   i++;
       token = strtok(0, delimiteur);
   }  }
```

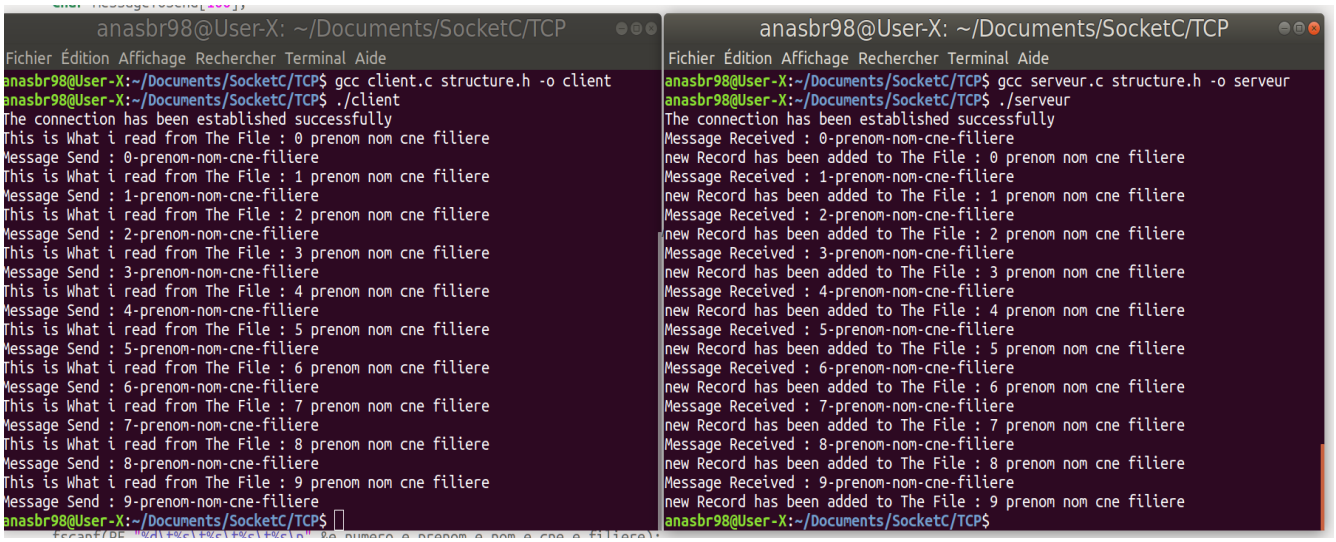
```c
int main(int argc, char **argv){

   //déclaration des variables
   int ServerDescipteur,longSocketClient;
   char messageReceived[1024];
   Etudiant e;
   FILE *PF;
   struct sockaddr_un AddressServer,AddressClient;
   bzero((char*)&AddressServer,sizeof(AddressServer));
   AddressServer.sun_family = AF_UNIX;
   bcopy("unix_socket",AddressServer.sun_path,11);
   if((ServerDescipteur = socket(AF_UNIX,SOCK_DGRAM,0)) == -1)
   {
   perror("Failed to Create Socket ... \n");
   exit(EXIT_FAILURE);
   }
   if(bind(ServerDescipteur,(struct
sockaddr*)&AddressServer,sizeof(AddressServer)) == -1)
   {
       perror("Failed to Bind ... \n");
       exit(EXIT_FAILURE);
   }

   printf("The connection has been established successfully \n");
   PF = fopen("file2.txt","w");
   if(PF == NULL)
   {
       perror("Failed to Open The file");
       exit(EXIT_FAILURE);
   }

   longSocketClient = sizeof(AddressClient);
   int i;
for(i=0;i<10;i++)
{
recvfrom(ServerDescipteur,messageReceived,sizeof(messageReceived),0
,(struct sockaddr *)&AddressClient,&longSocketClient);
getEtudiant(messageReceived,&e);
```

```
fprintf(PF,"%d\t%s\t%s\t%s\t%s\n",e.numero,e.prenom,e.nom,e.cne,e.filiere)
;}
   close(ServerDescipteur);
   fclose(PF);
   unlink("unix_socket");
   return EXIT_SUCCESS;
}
```

Structure Etudiant

```
#ifndef __structure
typedef struct Etudiant
{
   int numero;
   char prenom[10];
   char nom[10];
   char cne[15];
   char filiere[10];
}Etudiant;
#endif
```

Exécution



TP 3

Travail à faire : Écrivez un programme qui envoie une commande au serveur, et le serveur exécutera la commande et enverra le résultat au client.

Client.c

```
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <arpa/inet.h>
int main(void)
{
 int sockdescripteur = 0;
 char receivedMessage[1024],messageToSend[100];
 struct sockaddr_in serv_addr;
 memset(receivedMessage, '0' ,sizeof(receivedMessage));
 memset(messageToSend,'0',sizeof(messageToSend));
 if((sockdescripteur = socket(AF_INET, SOCK_STREAM, 0))< 0)
 {
     perror("Failed to create The Socket ...\n");
     exit(EXIT_FAILURE);
 }
 serv_addr.sin_family = AF_INET;
 serv_addr.sin_port = htons(5000);
 serv_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
 if(connect(sockdescripteur, (struct sockaddr *)&serv_addr,
sizeof(serv_addr))<0)
   {
     perror("Connection Failed ...\n");
```

```
      exit(EXIT_FAILURE);
    }
      printf("Enter The Command => ");
      gets(messageToSend);
      if(send(sockdescripteur,messageToSend,sizeof(messageToSend),0) == -1)
      {
        perror("Failed to send the command ...\n");
        exit(EXIT_FAILURE);
      }

      if(recv(sockdescripteur,receivedMessage,sizeof(receivedMessage),0) ==
-1)
      {
        perror("Failed to receive the result  ...\n");
        exit(EXIT_FAILURE);
      }
      printf("%s",receivedMessage);
 return 0;
}
```

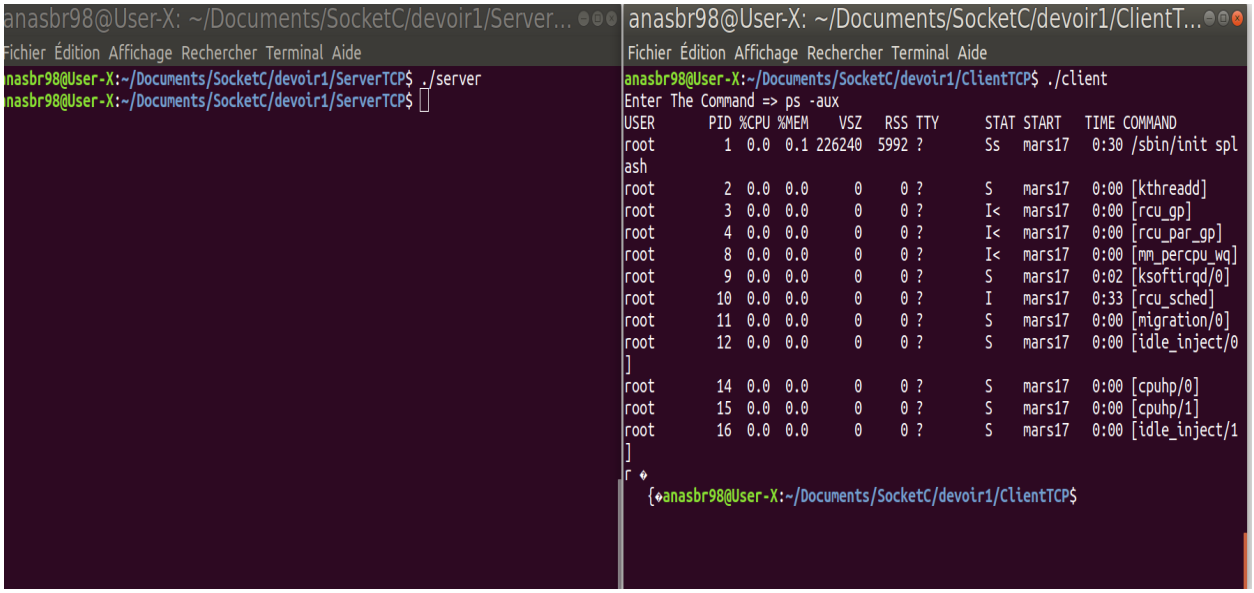
Server.c

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include<wait.h>
   int main(void)
   {
      int SocketDescripteur = 0,ClientSocket = 0;

      struct sockaddr_in ServerAdd;

      char sendBuff[1025],receiveBuff[1025];
```

```
    int numrv;

    SocketDescripteur = socket(AF_INET, SOCK_STREAM, 0);
    memset(&ServerAdd, '0', sizeof(ServerAdd));
    memset(sendBuff, '0', sizeof(sendBuff));
    memset(receiveBuff,'0',sizeof(receiveBuff));
    ServerAdd.sin_family = AF_INET;
    ServerAdd.sin_addr.s_addr = htonl(INADDR_ANY);
    ServerAdd.sin_port = htons(5000);

    bind(SocketDescripteur, (struct
sockaddr*)&ServerAdd,sizeof(ServerAdd));

    if(listen(SocketDescripteur, 10) == -1){
        perror("Failed to listen ... \n");
        exit(EXIT_FAILURE);
    }

    ClientSocket = accept(SocketDescripteur, (struct sockaddr*)NULL
,NULL);
     if(recv(ClientSocket,receiveBuff,sizeof(receiveBuff),0) == -1)
    {
        perror("Failed to recive ...\n");
        exit(EXIT_FAILURE);
    }
    if(fork()==0)
    {
        int count = 0;
    char *ptr = receiveBuff;
    while((ptr = strchr(ptr, ' ')) != NULL) {
        count++;
        ptr++;
    }
    char *argv[count+2];
    const char *delimiter = " ";
    char *token;
    token = strtok(receiveBuff,delimiter);
    int i=0;
    while(token != 0)
    {
```

```
        argv[i] = malloc(sizeof(token)+1);
        strcpy(argv[i],token);
        i++;
        token = strtok(0,receiveBuff);
      }
      argv[i] = NULL;
      dup2(ClientSocket,STDOUT_FILENO);
      execvp(argv[0],argv);
      }
      wait(NULL);
      close(ClientSocket);

 return 0;
}
```

Exécution



TP 4

Travail à faire : Écrivez un programme qui envoie un fichier au serveur, le fichier peut être une image, une vidéo ou un son, et le serveur lit les données et les stocke dans un fichier avec la même extension.

Client.c (UDP)

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<sys/socket.h>
#include<arpa/inet.h>
#include<sys/ioctl.h>
#include<unistd.h>
#include<errno.h>

int main(int argc,char** argv)
{
   if(argc < 3)
   {
       perror("Not enough Parameters ...\n");
       exit(EXIT_FAILURE);
   }
   int socketDes,fileLenght,result,byteSend = 0,SIZEMAX =
65000,PacketNumber=0;
   struct sockaddr_in sockaddr;
   char FileInfos[20],*buffer;
   FILE *file;

   memset(&sockaddr,'0',sizeof(sockaddr));

   socketDes = socket(AF_INET,SOCK_DGRAM,0);
   sockaddr.sin_port = htons(atoi(argv[2]));
   sockaddr.sin_family = AF_INET;
```

```
   sockaddr.sin_addr.s_addr = inet_addr(argv[1]);

   file = fopen(argv[3],"r");
   fseek(file,0,SEEK_END);
   fileLenght = ftell(file);
   fseek(file,0,SEEK_SET);
   buffer = (char*)malloc(SIZEMAX*sizeof(char));
   sprintf(FileInfos,"%s@%d",argv[3],fileLenght);
   if(sendto(socketDes,FileInfos,sizeof(FileInfos),0,(struct
sockaddr*)&sockaddr,sizeof(sockaddr))<0)
   {
       perror("Failed to send the File Infos ...\n");
       exit(EXIT_FAILURE);
   }
   while(byteSend < fileLenght)
   {
       if(fileLenght-byteSend < SIZEMAX)
       {
           SIZEMAX = fileLenght - byteSend;
           buffer = realloc(buffer,SIZEMAX);
       }
       if((result = fread(buffer,1,SIZEMAX,file)) != SIZEMAX)
       {
           perror("Failed to read the file ...\n");
           exit(EXIT_FAILURE);
       }
       printf("Sending %d byte \n",result);
       if((result = sendto(socketDes,buffer,SIZEMAX,0,(struct
sockaddr*)&sockaddr,sizeof(sockaddr))) != SIZEMAX)
       {
           perror("Failed to send the File ...\n");
           exit(EXIT_FAILURE);
       }
       byteSend += result;
       PacketNumber++;
   }

   printf("Bytes Sended : %d , File Lenght : %d \n",byteSend,fileLenght);
   printf("Number of Packet Sended is %d \n",PacketNumber);
   printf("File Sended successfully ...\n");
```

```
   close(socketDes);
   fclose(file);
   return 0;
}
```

Server.c (UDP)

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<sys/socket.h>
#include <sys/stat.h>
#include <sys/types.h>
#include<arpa/inet.h>
#include<unistd.h>
#include<errno.h>

void getFileInfos(char *message,char *fileN,int * fileL)
{
   char * token = strtok(message,"@");
   strcpy(fileN,token);
   token = strtok(0,"@");
   *fileL = atoi(token);
}

int main()
{
   int socketDescripteur,socketlen,fileLenght,result,bytesReceived =
0,SIZEMAX = 65000,PacketNumber=0;
   struct sockaddr_in socketAddress;
   char FileInfos[20],fileName[10],path[20],*buffer;
   FILE *file;

   memset(&socketAddress,'0',sizeof(socketAddress));

   socketDescripteur = socket(AF_INET,SOCK_DGRAM,0);
   socketAddress.sin_family = AF_INET;
   socketAddress.sin_port = htons(5000);
   socketAddress.sin_addr.s_addr = htonl(INADDR_ANY);
```

```
   if(bind(socketDescripteur,(struct
sockaddr*)&socketAddress,sizeof(socketAddress))<0)
   {
       perror("Failed to bind ...");
       exit(EXIT_FAILURE);
   }
   socketlen = sizeof(socketAddress);
   if(recvfrom(socketDescripteur,FileInfos,sizeof(FileInfos),0,(struct
sockaddr*)&socketAddress,&socketlen)<0)
   {
       perror("Failed to receive ...\n");
       exit(EXIT_FAILURE);
   }
   getFileInfos(FileInfos,fileName,&fileLenght);
   printf("Starting to Receive The File With The Name %s and Size %d
...\n",fileName,fileLenght);
   struct stat st;
   memset(&st,'0',sizeof(st));
   if(stat("files",&st) == -1)
   {
       mkdir("files",0700);
   }
   sprintf(path,"files/%s",fileName);
   buffer = (char*)malloc(sizeof(char)*SIZEMAX);
   if((file = fopen(path,"w+")) == NULL)
   {
   printf("Failed to open The File ...\n");
   exit(EXIT_FAILURE);
   }
   while(bytesReceived < fileLenght)
   {
       if(fileLenght-bytesReceived < SIZEMAX)
       {
           SIZEMAX = fileLenght - bytesReceived;
           buffer = realloc(buffer,SIZEMAX);
       }
       if((result = recvfrom(socketDescripteur,buffer,SIZEMAX,0,(struct
sockaddr*)&socketAddress,&socketlen)) != SIZEMAX)
       {
           perror("Failed to receive the file ...\n");
```

```
            exit(EXIT_FAILURE);
        }
        printf("Receiving %d byte \n",result);
        if((result = fwrite(buffer,1,SIZEMAX,file)) != SIZEMAX )
        {
            perror("Failed to write The File ...");
            exit(EXIT_FAILURE);
        }
        bytesReceived += result;
        PacketNumber++;
    }
    printf("Bytes Received : %d , File Lenght : %d
\n",bytesReceived,fileLenght);
    printf("Number of Packet Received is %d \n",PacketNumber);
    printf("File received Successfully ... \n");
    fclose(file);
    close(socketDescripteur);
    return 0;
}
```

Client.c (TCP)

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<arpa/inet.h>
#include<unistd.h>


int main(int argc,char** argv)
{
    if(argc < 3)
    {
        fprintf(stderr,"Not Enough Parameters ...\n");
        exit(EXIT_FAILURE);
```

```
    }

    int
serverSocketDescripteur,structLen,result,BytesSended=0,FileLenght,SIZEMAX
= 65000,PacketNumber=0;
    struct sockaddr_in clientSocket;
    char FileInfos[20],*buffer;
    FILE *file;

    memset(&clientSocket,'0',sizeof(clientSocket));

    clientSocket.sin_family = AF_INET;
    clientSocket.sin_port = htons(atoi(argv[2]));
    clientSocket.sin_addr.s_addr = inet_addr(argv[1]);

    if((serverSocketDescripteur = socket(AF_INET,SOCK_STREAM,0)) == -1)
    {
        fprintf(stderr,"Failed to Create the Socket ...\n");
        exit(EXIT_FAILURE);
    }

    if(connect(serverSocketDescripteur, (struct sockaddr
*)&clientSocket,sizeof(clientSocket)) == -1)
    {
        fprintf(stderr,"Failed to Connect to The Server ...\n");
        exit(EXIT_FAILURE);
    }

    if((file = fopen(argv[3],"r")) == NULL)
    {
        fprintf(stderr,"Failed to Open The File ...\n");
        exit(EXIT_FAILURE);
    }
    fseek(file,0,SEEK_END);
    FileLenght = ftell(file);
    fseek(file,0,SEEK_SET);
    buffer = (char*)malloc(SIZEMAX * sizeof(char));
    sprintf(FileInfos,"%s@%d",argv[3],FileLenght);

    if(send(serverSocketDescripteur,FileInfos,sizeof(FileInfos),0) == 0)
```

```
    {
        fprintf(stderr,"Failed to Send The File Infos To The Server
...\n");
        exit(EXIT_FAILURE);
    }

    while(BytesSended < FileLenght)
    {
        if(FileLenght-BytesSended < SIZEMAX)
        {
            SIZEMAX = FileLenght-BytesSended;
            buffer = realloc(buffer,SIZEMAX);
        }

        if((result = fread(buffer,1,SIZEMAX,file)) == -1)
        {
            fprintf(stderr,"Failed to Read From The File ...\n");
            exit(EXIT_FAILURE);
        }

        if((result = send(serverSocketDescripteur,buffer,SIZEMAX,0)) == -1)
        {
            fprintf(stderr,"Failed to send Data to Server ...\n");
            exit(EXIT_FAILURE);
        }

        printf("Sending %d bytes ...\n",result);
        BytesSended += result;
        PacketNumber++;
    }
    printf("Bytes Sended : %d  ,  File Lenght :  %d
\n",BytesSended,FileLenght);
    printf("Number of Packet Sended is %d ...\n",PacketNumber);
    printf("File Sended Successfully ...\n");
    close(serverSocketDescripteur);
    close(serverSocketDescripteur);
    fclose(file);
    return 0;
}
```

Server.c (TCP)

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<sys/stat.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<arpa/inet.h>
#include<unistd.h>

void getFileInfos(char* message,char* fileN,int *fileL)
{
   char *token = strtok(message,"@");
   strcpy(fileN,token);
   token = strtok(0,"@");
   *fileL = atoi(token);
}


int main(int argc,char** argv)
{
    if(argc < 1)
   {
       fprintf(stderr,"Not Enough Parameters ...\n");
       exit(EXIT_FAILURE);
   }
   int
socketDescripteur,clientSocketDescripteur,result,BytesReceived=0,structlen
,FileLenght,SIZEMAX = 65000,PacketNumber=0;
   struct sockaddr_in serverSocket;
   char FileInfos[20],FileName[10],Path[10],*buffer;
   FILE *file;

   memset(&serverSocket,'0',sizeof(serverSocket));

   serverSocket.sin_family = AF_INET;
   serverSocket.sin_port = htons(atoi(argv[1]));
   serverSocket.sin_addr.s_addr = htonl(INADDR_ANY);
```

```
   if((socketDescripteur = socket(AF_INET,SOCK_STREAM,0)) == -1)
   {
       fprintf(stderr,"Failed to create The Socket ...\n");
       exit(EXIT_FAILURE);
   }

   if(bind(socketDescripteur,(struct
sockaddr*)&serverSocket,sizeof(serverSocket)) == -1)
   {
       fprintf(stderr,"Failed to Bind ...\n");
       exit(EXIT_FAILURE);
   }

   if(listen(socketDescripteur,10) == -1)
   {
       fprintf(stderr,"Failed to listen ...\n");
       exit(EXIT_FAILURE);
   }

   if((clientSocketDescripteur = accept(socketDescripteur,(struct
sockaddr*)NULL,NULL)) == -1)
   {
       fprintf(stderr,"Failed to accept a new Client ...\n");
       exit(EXIT_FAILURE);
   }

  if(recv(clientSocketDescripteur,FileInfos,sizeof(FileInfos),0) == -1)
  {
      fprintf(stderr,"Failed to Receive The File Infos from the Client
...\n");
      exit(EXIT_FAILURE);
  }

   getFileInfos(FileInfos,FileName,&FileLenght);
   sprintf(Path,"files/%s",FileName);

   struct stat st;
   memset(&st,'0',sizeof(st));
   if(stat("files",&st) == -1)
   {
```

```
        mkdir("files",0700);
    }

    buffer = (char*)malloc(SIZEMAX*sizeof(char));

    if((file = fopen(Path,"w")) == NULL)
    {
        fprintf(stderr,"Failed to Open The File ...\n");
        exit(EXIT_FAILURE);
    }

    while (BytesReceived < FileLenght)
    {
        if(FileLenght - BytesReceived < SIZEMAX)
        {
            SIZEMAX = FileLenght - BytesReceived;
            buffer = realloc(buffer,SIZEMAX);
        }

        if((result = recv(clientSocketDescripteur,buffer,SIZEMAX,0)) == -1)
        {
            fprintf(stderr,"Failed to from The Client ...\n");
            exit(EXIT_FAILURE);
        }

        printf("Receiving %d bytes ... \n",result);

        if((result = fwrite(buffer,1,SIZEMAX,file)) == -1)
        {
            fprintf(stderr,"Failed to Write in the File ...\n");
            exit(EXIT_FAILURE);
        }
        BytesReceived += result;
        PacketNumber ++;
    }

    printf("Bytes Received : %d  , FileLenght : %d ...
\n",BytesReceived,FileLenght);
    printf("Number of Packet Received is %d ... \n",PacketNumber);
    printf("File Received Successfully ...\n");
```

```
    close(socketDescripteur);
    close(socketDescripteur);
    fclose(file);
    return 0;
}
```

Exécution