Réseaux et programmation réseaux

T4P : Programmation réseau en UDP 2016

Introduction

Ce syllabus vous invite à élaborer votre propre protocole à l'aide de librairies de base, **LIB2016** qui vous seront fournies. Le protocole est construit au-dessus de UDP.

Le choix du protocole UDP

Notre protocole
UDP
IP
Ethernet
Physique

Le protocole UDP encapsule très légèrement le protocole IP.

Il ne fait que rajouter la notion de port qui permet le multiplexage.

Il reprend tout les défauts principaux d'IP.

- Pas de garantie de délivrance de l'information.
- Ordre non garanti.
- Pas de mécanisme d'acquittement
- Pas de mécanisme de contrôle de flux.

Il convient donc parfaitement à toute personne qui souhaite réaliser son propre protocole.

UDP de par ses caractéristiques, permet de simuler une couche physique et permet aussi de comprendre les contraintes de l'élaboration d'un protocole de niveau 2.

Cette démarche n'est pas non plus purement pédagogique. Il est très courant dans la réalité que l'on reconstruise un nouveau protocole par dessus un existant. Le VPN utilise cette démarche.

Installer les librairies

Télécharger LIB2017Etud.tar à partir mon centre de ressources (Réseau et programmation Réseaux \ LIB2017

```
vanstap@vanstap2:~/lib2017Etud$
```

Pour exécuter un exemple.

```
vanstap@vanstap2:~/lib2017Etud$ cd ex01
vanstap@vanstap2:~/lib2017Etud/ex01$ make
make: Nothing to be done for `all'.
```

Pour créer une étape du dossier de labo

```
vanstap@vanstap2:~/lib2017Etud$ mkdir Step1
vanstap@vanstap2:~/lib2017Etud$ cd Step1
vanstap@vanstap2:~/lib2017Etud/Step1$
```

Plateformes supportées

Ces notes ont été testées par défaut sous ubuntu 14.04 que vous pouvez installer vous vmware player (voir document sur mon centre de ressource) ou sur une clé usB.Pour compiler sous sun , lire les annexes.

Cela compile aussi sous Sun, mais

La librairie fonctionne sur sun à condition de faire les adaptations suivantes.

1) il faut dans un premier temps effacer les **cli, ser, .o** dans chaque dossier ex0X (x = 1,2,3,5,7)

2) Il faut editer le fichier ***makefile*** et Modifier la variable LIBS

```
LIBS=-lsocket -lnsl
```

3) Dans les makefile remplacer **cc** par **gcc**.

EXO1 : Un Serveur et un Client qui échangent une chaine de caractère

Makefile

```
# cphex\makefile

LIBS=
all:	cli	ser	udplib.o

udplib.o:	../udplib/udplib.h	../udplib/udplib.c
	echo "Compilation de udplib.o"
	gcc-c ../udplib/udplib.c

cli:	cli.c	udplib.o	lib.c
	echo "Compilation de client"
	gcc -o cli  cli.c udplib.o  $(LIBS)

ser:	ser.c	udplib.o	lib.c
	echo "Compilation de serveur"
	gcc-o ser  ser.c udplib.o $(LIBS)
```

Si vous compilez sous SUN, n'oubliez pas de completer la variable LIBS, voir annexe

Compilation

```
vanstap@vanstap2:~/lib2016/ex01$ rm *.o
vanstap@vanstap2:~/lib2016/ex01$ make
```

Note le rm *.o permet d'effacer les .o car ils pourraient être ne pas compatible avec votre OS. Sun n'est pas compatible avec les fichiers d'ubuntu.

Exécution du serveur

```
vanstap@vanstap2:~/lib2016/ex01$ ./ser 127.0.0.1 1300
Ceci est le serveur
port 1300
CreateSockets 3
bytes reçus:9:Bonjour
bytes envoyés:17
```

Note : 127.0.0.1 est l'adresse IP sur laquelle le serveur écoute et 1300 est le port sur leque le serveur écoute

Exécution du Client

```
vanstap@vanstap2:~/lib2016/ex01$ ./cli 127.0.0.1 1400 127.0.0.1 1300
port 1400
CreateSockets 3
Envoi de 9 bytes
bytes reçus:17:bonjour client 1
```

Source Serveur

```
/*---------------------------------------
  cphex\ser.c
  ex01 un seveur recevant des des chaines de caractères
----------------------------------------*/

#include <stdio.h>
#include <string.h>
#include "../udplib/udplib.h"

void die(char *s)
{
   perror(s);
   exit(1);
}

int main(int argc,char *argv[])
{
 int rc ;
 int Desc ;

 struct sockaddr_in sthis ; /* this ce programme */
 struct sockaddr_in sos ; /* s = serveur */
 struct sockaddr_in sor ; /* r = remote */

 u_long  IpSocket ;
 u_short PortSocket ;

 char message[100] ;
 char *reponse ;

 int tm ;

 memset(&sthis,0,sizeof(struct sockaddr_in)) ;
 memset(&sos,0,sizeof(struct sockaddr_in)) ;
 memset(&sor,0,sizeof(struct sockaddr_in)) ;

 printf("Ceci est le serveur\n") ;
 if ( argc!=3)
 {
  printf("ser ser port cli\n") ;
  exit(1) ;
 }
```

```
/* Récupération IP & port   */
IpSocket= inet_addr(argv[1]);
PortSocket = atoi(argv[2]);
// Desc = CreateSockets(&psoo,&psos,,atoi(argv[2]),argv[3],atoi(argv[4])) ;
Desc=creer_socket(SOCK_DGRAM,&IpSocket,PortSocket,&sthis);

if ( Desc == -1 )
  die("CreateSockets:") ;

else
  fprintf(stderr,"CreateSockets %d\n",Desc) ;


tm = sizeof(message) ;
rc = ReceiveDatagram( Desc, message,tm, &sor ) ;
if ( rc == -1 )
  die("ReceiveDatagram") ;
else
 fprintf(stderr,"bytes reçus:%d:%s\n",rc,message ) ;

/* reponse avec psos */
reponse = "bonjour client 1" ;
rc = SendDatagram(Desc,reponse,strlen(reponse)+1 ,&sor ) ;
if ( rc == -1 )
  die("SendDatagram") ;
else
 fprintf(stderr,"bytes envoyés:%d\n",rc ) ;

close(Desc) ;
}
```

argc ,argv

```
int argc,char *argv[])
```

```
argc contient le nombre d'arguments du programme. Le nom du programme compte aussi
pour un argument
argv[0] : contient le nom du programme
argv[1] : contient le premier argument du programme.
```

La notion de socket

```
struct sockaddr_in sthis ; /* this ce programme */
 struct sockaddr_in sos ; /* s = serveur */
 struct sockaddr_in sor ; /* r = remote */
```

Avant de programmer toute connexion réseau, vous devez déclarer des sockaddr_in. Une sockaddr_in contient en fait le type de connexion (udp,tcp),En seconde ce sera toujours udp, l'ip et le port.

sthis	Contient l'ip et le port sur de l'interface réseau de votre machine lequel notre serveur / client va se connecter
sos	Va contenir l'ip et le port du programme auquel notre programme veut envoyer des données Udp est symétrique. Chaque programme est client et serveur
sor	Chaque fois que le serveur recevra des données via la fonction ***ReceiveDatagram***, cette structure est initialisée avec l'ip et le port du client

```
memset(&sthis,0,sizeof(struct sockaddr_in)) ;;
```

il est impératif que chaque sockaddr_in soit initialisée entièrement à zéro avant emploi.

```
/* Récupération IP & port  */
 IpSocket= inet_addr(argv[1]);
 PortSocket = atoi(argv[2]);
```

La fonction inet_addr convertit une chaine de caractère en une ip.

La fonction CreateSockets

```
Desc=creer_socket(SOCK_DGRAM, &IpSocket,PortSocket,&sthis);
```

Cree une connexion udp pour les paramètre in ipSocket et PortSocket, retourne un descripetur de Socket dans Desc et initialise La sockaddr_in sthis

SOCK_DGRAM	int	Crée une socket de Type udp
IpSocket	*Int	Ip à la laquelle notre programme va se connecter
PortSocket	Int	Le port auquel votre programme va se connecter
sthis	struct sockaddr_in*	Contient l'ip de la carte réseau et le port auquel notre programme se connecte
Desc	Int	Descripteur pointant sur la socket udp

La fonction ReceiveDatagram avec une chaine

tm = sizeof(message) ; rc = ReceiveDatagram(Desc, message,tm, &sor) ;		
La fonction ReceiveDatagram permet de recevoir des données sur une connexion udp		
Desc	int	Descripteur retourné par la fonction CreateSockets
message	Void *	Adresse de la chaine de caractère qui va stocker le message reçu.
tm	int	Taille de la donnée à recevoir. Attention cette taille ne doit pas excéder la taille de message et est donc typiquement initialisée à la taille de cette dernière.
sor	struct sockaddr_in*	Contient l'adresse ip et le port du programmme émetteur du message
rc	int	Nombre de bytes effectivement reçus

La fonction SendDatagram avec une chaine

reponse = "bonjour client 1" ; rc = SendDatagram(Desc,reponse,strlen(reponse)+1 ,&sor) ;		
La fonction SendDatagram permet denvoyer des données sur une connexion udp		
Desc	int	Descripteur retourné par la fonction CreateSockets
reponse	Void *	Adresse de la chaine de caractère contenant le message à envoyer
Tm	int	Taille du message à envoyer. Typiquement strlen(reponse)+1
sor	struct sockaddr_in*	Contient l'adresse ip et le port du programmme destinataire.Ce paramètre est initialisé par ReceiveDatagram
rc	int	Nombre de bytes effectivement envoyés

Fermer une connexion udp avec close

```
close(Desc) ;
```

Source Client

```
/*-------------------------------------------------------------
  Vanstapel Herman
  ex01\cli.c

 ex01 Le client dit bonjour et
    le serveur fait de même
--------------------------------------------------------------*/
#include <stdio.h>
```

```
#include <string.h>
#include "../udplib/udplib.h"

void die(char *s)
{
   perror(s);
   exit(1);
}

int main(int argc, char *argv[])
{
 int rc ;
 int Desc ;
 char *message ;
 char reponse[100 ] ;
 int tm ;
 u_long  IpSocket , IpServer;
 u_short PortSocket, PortServer ;

 struct sockaddr_in sthis ; /* this ce programme */
 struct sockaddr_in sos ; /* s = serveur */
 struct sockaddr_in sor ; /* r = remote */

 memset(&sthis,0,sizeof(struct sockaddr_in)) ;
 memset(&sos,0,sizeof(struct sockaddr_in)) ;
 memset(&sor,0,sizeof(struct sockaddr_in)) ;

 if (argc!=5)

{
   printf("cli client portc serveur ports\n") ;
   exit(1) ;
 }

 /* Récupération IP & port   */
 IpSocket= inet_addr(argv[1]);
 PortSocket = atoi(argv[2]);

 IpServer = inet_addr(argv[3]) ;
 PortServer = atoi(argv[4]);

 // Desc = CreateSockets(&psoo,&psos,,atoi(argv[2]),argv[3],atoi(argv[4])) ;
 Desc=creer_socket(SOCK_DGRAM,&IpSocket,PortSocket,&sthis);

 if ( Desc == -1 )
   die("CreateSockets:") ;
 else
   fprintf(stderr,"CreateSockets %d\n",Desc) ;
```

```
 message = "Bonjour " ;

 /*******************************************************/

  sos.sin_family = AF_INET ;
  sos.sin_addr.s_addr= IpServer ;
  sos.sin_port = htons(PortServer) ;

  /*******************************************************/

 rc = SendDatagram(Desc,message,strlen(message)+1 ,&sos ) ;

 if ( rc == -1 )
    die("SendDatagram") ;
 else
   fprintf(stderr,"Envoi de %d bytes\n",rc ) ;

  memset(reponse,0,sizeof(reponse)) ;
 tm = sizeof(reponse) ;
 rc = ReceiveDatagram( Desc, reponse,tm, &sor ) ;
 if ( rc == -1 )
    die("ReceiveDatagram") ;
 else
   fprintf(stderr,"bytes reçus:%d:%s\n",rc,reponse ) ;

 close(Desc) ;
}
```

Le client récupère l'adresse du serveur passée en paramètre et initalise la sockaddr_in sos qui sera utilisée par la primitive SendDatagram.

```
 sos.sin_family = AF_INET ;
 sos.sin_addr.s_addr= IpServer ;
 sos.sin_port = htons(PortServer) ;
```

Le paramètre sos est passé comme paramètre à SendDatagram

```
rc = SendDatagram(Desc,message,strlen(message)+1 ,&sos ) ;
```

EXO2 : Un Serveur et un Client qui échangent une structure de donnée

Makefile

```
LIBS=
all:	cli	ser	udplib.o

udplib.o:	../udplib/udplib.h	../udplib/udplib.c
	echo "Compilation de udplib.o"
	gcc-c ../udplib/udplib.c

cli:	cli.c	udplib.o
	echo "Compilation de client"
	gcc -o cli  cli.c udplib.o   $(LIBS)

ser:	ser.c	udplib.o
	echo "Compilation de serveur"
	gcc-o ser  ser.c udplib.o $(LIBS)
```

Si vous compilez sous SUN, n'oubliez pas de completer la variable LIBS, voir annexe

Compilation

```
vanstap@vanstap2:~/lib2016/ex01$ rm *.o
vanstap@vanstap2:~/lib2016/ex01$ make
```

Note le rm *.o permet d'effacer les .o car ils pourraient être ne pas compatible avec votre OS. Sun n'est pas compatible avec les fichiers d'ubuntu.

Exécution du serveur

```
vanstap@vanstap2:~/lib2016/ex02$ ./ser 127.0.0.1 1300
Ceci est le serveur
port 1300
CreateSockets 3
bytes recus:44:Avec une structure: Bonjour
Type recu 1
bytes envoyes:44
```

Exécution du Client

```
vanstap@vanstap2:~/lib2016/ex02$ ./cli 127.0.0.1 1400 127.0.0.1 1300
port 1400
CreateSockets 3
Envoi de 44 bytes
bytes recus:44:Avec une structure: Bonjour Client
```

Structure.h

```
enum TypeRequete {
      Question = 1 ,
      Reponse  = 2
    } ;

struct Requete
{
  enum TypeRequete Type ;
  char Message[40] ;
} ;
```

Note : A partir du moment ou le client et le serveur vont échanger une structure de donnée, celle-ci doit être définie dans un .h commun aux deux programmes. La structure Requete sera la structure échangée sur Réseau.

un enum permet de définir des constantes ici Question et Reponse qui seront stockée en mémoire respectivement avec les valeurs 1 et 2.

Source Serveur

```
/*--------------------------------------
  Herman Vanstapel

  ex02\ser.c

  Un serveur recevant une structure
----------------------------------------*/

#include <stdio.h>
#include <string.h>
#include "../udplib/udplib.h"
#include "structure.h"


void die(char *s)
{
   perror(s);
   exit(1);
}

int main(int argc,char *argv[])
{
 int rc ;
 int Desc ;
```

```
struct sockaddr_in sthis ; /* this ce programme */
struct sockaddr_in sos ; /* s = serveur */
struct sockaddr_in sor ; /* r = remote */

u_long  IpSocket ;
u_short PortSocket ;

int tm ;
struct Requete UneRequete ;

memset(&sthis,0,sizeof(struct sockaddr_in)) ;
memset(&sos,0,sizeof(struct sockaddr_in)) ;
memset(&sor,0,sizeof(struct sockaddr_in)) ;

printf("Ceci est le serveur\n") ;
if ( argc!=3)
{
 printf("ser ser port cli\n") ;
 exit(1) ;
}

/* Récupération IP & port   */
IpSocket= inet_addr(argv[1]);
PortSocket = atoi(argv[2]);
// Desc = CreateSockets(&psoo,&psos,,atoi(argv[2]),argv[3],atoi(argv[4])) ;
Desc=creer_socket(SOCK_DGRAM,&IpSocket,PortSocket,&sthis);

if ( Desc == -1 )
   die("CreateSockets:") ;
else
   fprintf(stderr,"CreateSockets %d\n",Desc) ;

tm = sizeof(struct Requete) ;
rc = ReceiveDatagram( Desc,&UneRequete ,tm, &sor ) ;
if ( rc == -1 )
   die("ReceiveDatagram") ;
else
  fprintf(stderr,"bytes recus:%d:%s\n",rc,UneRequete.Message ) ;

printf("Type recu %d\n", UneRequete.Type) ;
/* attention l'enum peut être codé en short */
/* reponse avec psos */

UneRequete.Type = Reponse ;
strcat(UneRequete.Message," Client") ;
rc = SendDatagram(Desc,&UneRequete,sizeof(struct Requete) ,&sor ) ;
if ( rc == -1 )
   die("SendDatagram:") ;
```

```
 else
   fprintf(stderr,"bytes envoyes:%d\n",rc ) ;

 close(Desc) ;
}
```

argc ,argv

```
int argc,char *argv[])
```

argc contient le nombre d'arguments du programme. Le nom du programme compte aussi pour un argument
argv[0] : contient le nom du programme
argv[1] : contient le premier argument du programme.

La notion de socket

```
struct sockaddr_in sthis ; /* this ce programme */
 struct sockaddr_in sos ; /* s = serveur */
 struct sockaddr_in sor ; /* r = remote */
```

Avant de programmer toute connexion réseau, vous devez déclarer des sockaddr_in. Une sockaddr_in contient en fait le type de connexion (udp,tcp),En seconde ce sera toujours udp, l'ip et le port.

sthis	Contient l'ip et le port sur de l'interface réseau de votre machine lequel notre serveur / client va se connecter
sos	Va contenir l'ip et le port du programme auquel notre programme veut envoyer des données Udp est symétrique. Chaque programme est client et serveur
sor	Chaque fois que le serveur recevra des données via la fonction ***ReceiveDatagram***, cette structure est initialisée avec l'ip et le port du client

```
memset(&sthis,0,sizeof(struct sockaddr_in)) ;;
```

il est impératif que chaque sockaddr_in soit initialisée entièrement à zéro avant emploi.

```
/* Récupération IP & port  */
 IpSocket= inet_addr(argv[1]);
 PortSocket = atoi(argv[2]);
```

La fonction inet_addr convertit une chaine de caractère en une ip.

La fonction CreateSockets

Desc=creer_socket(SOCK_DGRAM, &IpSocket,PortSocket,&sthis);		
Cree une connexion udp pour les paramètre in ipSocket et PortSocket, retourne un descripetur de Socket dans Desc et initialise La sockaddr_in sthis		
SOCK_DGRAM	int	Crée une socket de Type udp
IpSocket	*Int	Ip à la laquelle notre programme va se connecter
PortSocket	Int	Le port auquel votre programme va se connecter
sthis	struct sockaddr_in*	Contient l'ip de la carte réseau et le port auquel notre programme se connecte
Desc	Int	Descripteur pointant sur la socket udp

Avec les structures de données, on notera que le paramètre tm utilise l'instruction sizeof à la place de strlen

```
tm = sizeof(struct Requete) ;
 rc = ReceiveDatagram( Desc,&UneRequete ,tm, &sor ) ;
```

Pour renvoyer la réponse au client

```
UneRequete.Type = Reponse ;
 strcat(UneRequete.Message," Client") ;
 rc = SendDatagram(Desc,&UneRequete,sizeof(struct Requete) ,&sor ) ;
```

Source Client

```
/*---------------------------------------------------------------
  Vanstapel Herman
  ex02\cli.c

 Le client dit bonjour en utilisant un structure  et
 le serveur fait de même
---------------------------------------------------------------*/
#include <stdio.h>
#include <string.h>
#include "../udplib/udplib.h"
#include "structure.h"

void die(char *s)
{
   perror(s);
   exit(1);
}

int main(int argc, char *argv[])
{
 int rc ;
 int Desc ;
 int tm ;

 u_long  IpSocket , IpServer;
 u_short PortSocket, PortServer ;

 struct sockaddr_in sthis ; /* this ce programme */
 struct sockaddr_in sos ; /* s = serveur */
 struct sockaddr_in sor ; /* r = remote */
 struct Requete UneRequete ;

 memset(&sthis,0,sizeof(struct sockaddr_in)) ;
 memset(&sos,0,sizeof(struct sockaddr_in)) ;
 memset(&sor,0,sizeof(struct sockaddr_in)) ;

 if (argc!=5)

{
   printf("cli client portc serveur ports\n") ;
   exit(1) ;
 }
```

```
/* Récupération IP & port   */
IpSocket= inet_addr(argv[1]);
PortSocket = atoi(argv[2]);

IpServer = inet_addr(argv[3]) ;
PortServer = atoi(argv[4]);

// Desc = CreateSockets(&psoo,&psos,,atoi(argv[2]),argv[3],atoi(argv[4])) ;
Desc=creer_socket(SOCK_DGRAM,&IpSocket,PortSocket,&sthis);

if ( Desc == -1 )
  die("CreateSockets:") ;
else
  fprintf(stderr,"CreateSockets %d\n",Desc) ;

 sos.sin_family = AF_INET ;
 sos.sin_addr.s_addr= IpServer ;
 sos.sin_port = htons(PortServer) ;


UneRequete.Type = Question ;
strncpy(UneRequete.Message , "Avec une structure: Bonjour" , sizeof(UneRequete.Message)) ;

rc = SendDatagram(Desc,&UneRequete,sizeof(struct Requete) ,&sos ) ;

if ( rc == -1 )
  die("SendDatagram") ;
else
  fprintf(stderr,"Envoi de %d bytes\n",rc ) ;

memset(&UneRequete,0,sizeof(struct Requete)) ;
tm = sizeof(struct Requete) ;

 rc = ReceiveDatagram( Desc, &UneRequete,tm, &sor ) ;
if ( rc == -1 )
  die("ReceiveDatagram") ;
else
  fprintf(stderr,"bytes recus:%d:%s\n",rc,UneRequete.Message ) ;

close(Desc) ;
}
```

Le client récupère l'adresse du serveur passée en paramètre et initalise la sockaddr_in sos qui sera utilisée par la primitive SendDatagram.

```
 sos.sin_family = AF_INET ;
 sos.sin_addr.s_addr= IpServer ;
 sos.sin_port = htons(PortServer) ;
```

Initialisation de la chaine de caractère de la structure

```
strncpy(UneRequete.Message , "Avec une structure: Bonjour" , sizeof(UneRequete.Message)) ;
```

Envoi de la structure au client

```
rc = SendDatagram(Desc,&UneRequete,sizeof(struct Requete) ,&sos ) ;
```

EXO3 : un serveur multiclient

Exécution du serveur

```
vanstap@vanstap2:~/lib2016/ex03$ ./ser 127.0.0.1 1300
Ceci est le serveur
port 1300
CreateSockets 3
bytes recus:44:Avec une structure: Bonjour
Received packet from 127.0.0.1:1400
Type recu 1
bytes envoyes:44
bytes recus:44:Avec une structure: Bonjour
Received packet from 127.0.0.1:1500
Type recu 1
bytes envoyes:44
```

Exécution des Clients

```
vanstap@vanstap2:~/lib2016/ex03$ ./cli 127.0.0.1 1400 127.0.0.1 1300
port 1400
CreateSockets 3
Envoi de 44 bytes
bytes recus:44:Avec une structure: Bonjour Client
```

```
vanstap@vanstap2:~/lib2016/ex03$ ./cli 127.0.0.1 1500 127.0.0.1 1300
port 1500
CreateSockets 3
Envoi de 44 bytes
bytes recus:44:Avec une structure: Bonjour Client
```

Makefile

```
LIBS=
all:	cli	ser	udplib.o

udplib.o:	../udplib/udplib.h	../udplib/udplib.c
	echo "Compilation de udplib.o"
	gcc-c ../udplib/udplib.c

cli:	cli.c	udplib.o
	echo "Compilation de client"
	gcc -o cli  cli.c udplib.o   $(LIBS)

ser:	ser.c	udplib.o
	echo "Compilation de serveur"
	gcc-o ser  ser.c udplib.o $(LIBS)
```

Source Serveur

```
/*---------------------------------------
  Herman Vanstapel

  ex03\ser.c

  Un serveur recevant une structure
---------------------------------------*/

#include <stdio.h>
#include <string.h>
#include "../udplib/udplib.h"
#include "structure.h"


void die(char *s)
{
   perror(s);
   exit(1);
}

int main(int argc,char *argv[])
{
 int rc ;
 int Desc ;
 struct sockaddr_in sthis ; /* this ce programme */
 struct sockaddr_in sos ; /* s = serveur */
 struct sockaddr_in sor ; /* r = remote */
```

```
u_long  IpSocket ;
u_short PortSocket ;

int tm ;
struct Requete UneRequete ;

memset(&sthis,0,sizeof(struct sockaddr_in)) ;
memset(&sos,0,sizeof(struct sockaddr_in)) ;
memset(&sor,0,sizeof(struct sockaddr_in)) ;

printf("Ceci est le serveur\n") ;
if ( argc!=3)
{
 printf("ser ser port cli\n") ;
 exit(1) ;
}

/* Récupération IP & port   */
IpSocket= inet_addr(argv[1]);
PortSocket = atoi(argv[2]);
// Desc = CreateSockets(&psoo,&psos,,atoi(argv[2]),argv[3],atoi(argv[4])) ;
Desc=creer_socket(SOCK_DGRAM,&IpSocket,PortSocket,&sthis);

if ( Desc == -1 )
   die("CreateSockets:") ;
else
   fprintf(stderr,"CreateSockets %d\n",Desc) ;

while(1)
{
tm = sizeof(struct Requete) ;
rc = ReceiveDatagram( Desc,&UneRequete ,tm, &sor ) ;
if ( rc == -1 )
   die("ReceiveDatagram") ;
else
  fprintf(stderr,"bytes recus:%d:%s\n",rc,UneRequete.Message ) ;

 printf("Received packet from %s:%d\n", inet_ntoa(sor.sin_addr), ntohs(sor.sin_port));
 printf("Type recu %d\n", UneRequete.Type) ;
/* attention l'enum peut être codé en short */
/* reponse avec psos */

UneRequete.Type = Reponse ;
strcat(UneRequete.Message," Client") ;
rc = SendDatagram(Desc,&UneRequete,sizeof(struct Requete) ,&sor ) ;
if ( rc == -1 )
   die("SendDatagram:") ;
```

```
 else
   fprintf(stderr,"bytes envoyes:%d\n",rc ) ;
 }
 close(Desc) ;
}
```

Au niveau du code du serveur, rien ne change avec le multiclient sauf qu'on affiche l'ip et le port du client qui envoie le datagramme. L'ip et le port du client sont contenue dans la sockaddr_in sor qui est initialisée par le ReceiveDatagram.

inet_ntoa fair la conversion de l'ip vers une chaine de caractères représentant l'ip au format dotted decimal.
ntohs fait la convertion du port au format intel

```
printf("Received packet from %s:%d\n", inet_ntoa(sor.sin_addr), ntohs(sor.sin_port));
```

Source Client

```
/*------------------------------------------------------------
   Vanstapel Herman
   ex03\cli.c

 Le client dit bonjour en utilisant un structure  et
 le serveur fait de même
------------------------------------------------------------*/
#include <stdio.h>
#include <string.h>
#include "../udplib/udplib.h"
#include "structure.h"

void die(char *s)
{
   perror(s);
   exit(1);
}

int main(int argc, char *argv[])
{
 int rc ;
 int Desc ;
 int tm ;

 u_long  IpSocket , IpServer;
 u_short PortSocket, PortServer ;

 struct sockaddr_in sthis ; /* this ce programme */
 struct sockaddr_in sos ; /* s = serveur */
```

```
 struct sockaddr_in sor ; /* r = remote */
 struct Requete UneRequete ;

 memset(&sthis,0,sizeof(struct sockaddr_in)) ;
 memset(&sos,0,sizeof(struct sockaddr_in)) ;
 memset(&sor,0,sizeof(struct sockaddr_in)) ;

 if (argc!=5)

{
   printf("cli client portc serveur ports\n") ;
   exit(1) ;
 }

 /* Récupération IP & port   */
 IpSocket= inet_addr(argv[1]);
 PortSocket = atoi(argv[2]);

 IpServer = inet_addr(argv[3]) ;
 PortServer = atoi(argv[4]);

 // Desc = CreateSockets(&psoo,&psos,,atoi(argv[2]),argv[3],atoi(argv[4])) ;
 Desc=creer_socket(SOCK_DGRAM,&IpSocket,PortSocket,&sthis);

 if ( Desc == -1 )
   die("CreateSockets:") ;
 else
   fprintf(stderr,"CreateSockets %d\n",Desc) ;

  sos.sin_family = AF_INET ;
  sos.sin_addr.s_addr= IpServer ;
  sos.sin_port = htons(PortServer) ;


 UneRequete.Type = Question ;
 strncpy(UneRequete.Message , "Avec une structure: Bonjour" , sizeof(UneRequete.Message)) ;

 rc = SendDatagram(Desc,&UneRequete,sizeof(struct Requete) ,&sos ) ;

 if ( rc == -1 )
   die("SendDatagram") ;
 else
  fprintf(stderr,"Envoi de %d bytes\n",rc ) ;

 memset(&UneRequete,0,sizeof(struct Requete)) ;
 tm = sizeof(struct Requete) ;

  rc = ReceiveDatagram( Desc, &UneRequete,tm, &sor ) ;
```

```
 if ( rc == -1 )
    die("ReceiveDatagram") ;
 else
   fprintf(stderr,"bytes recus:%d:%s\n",rc,UneRequete.Message ) ;

 close(Desc) ;
}
```

EXO5 : serveur Multiport , multiclient

Exécution du serveur

```
vanstap@vanstap2:~/lib2016/ex05$ ./ser 127.0.0.1 1300 127.0.0.1 1350
Ceci est le serveur
port 1300
 CreateSockets 1 : 3
port 1350
 CreateSockets 2 : 4
Traitement requete sur 3
bytes:84:Avec une structure: Bonjour:type 1
Received packet from 127.0.0.1:1400
bytes:84
Traitement requete sur 4
bytes:84:Avec une structure: Bonjour:type 1
Received packet from 127.0.0.1:1500
bytes:84
```

Exécution des Clients

```
vanstap@vanstap2:~/lib2016/ex05$ ./cli 127.0.0.1 1400 127.0.0.1 1300
port 1400
CreateSockets 3
Envoi de 84 bytes
bytes reçus:84:Avec une structure: Bonjour Client
vanstap@vanstap2:~/lib2016/ex05$
```

```
vanstap@vanstap2:~/lib2016/ex05$ ./cli 127.0.0.1 1500 127.0.0.1 1350
port 1500
CreateSockets 3
Envoi de 84 bytes
bytes reçus:84:Avec une structure: Bonjour Client
```

Makefile

```
LIBS=
all:	cli	ser	udplib.o

udplib.o:	../udplib/udplib.h	../udplib/udplib.c
	echo "Compilation de udplib.o"
	gcc-c ../udplib/udplib.c

cli:	cli.c	udplib.o
	echo "Compilation de client"
```

```
	gcc -o cli  cli.c udplib.o   $(LIBS)

ser:	ser.c	udplib.o
	echo "Compilation de serveur"
	gcc-o ser  ser.c udplib.o $(LIBS)
```

Source Serveur

```
/*---------------------------------------
  Herman Vanstapel
  ex05\ser.c
  Un serveur multiport multiclients
----------------------------------------*/

#include <stdio.h>
#include <string.h>
/* include pour le select */
#include <sys/types.h>
#include <sys/time.h>

#include "../udplib/udplib.h"
#include "structure.h"


void die(char *s)
{
   perror(s);
   exit(1);
}

void DelNewLine(char *Chaine)
{
 Chaine[strlen(Chaine)-1] = 0 ;
}

char ReadChar()
{
 char Tampon[80] ;
 fgets(Tampon,sizeof Tampon,stdin ) ;
 return Tampon[0] ;
}

void TraitementRequete(int Desc )
{
  struct sockaddr_in sor ; /* r = remote */

  int tm ;
```

```
 struct Requete UneRequete ;
 char Tampon[40] ;
 int Ip, Port ;
 int rc ;

 printf("Traitement requete sur %d \n",Desc ) ;
 memset(&sor,0,sizeof(struct sockaddr_in)) ;
 tm = sizeof(struct Requete) ;
  rc = ReceiveDatagram( Desc,&UneRequete ,tm, &sor ) ;
  if ( rc == -1 )
    perror("ReceiveDatagram") ;
  else
     fprintf(stderr,"bytes:%d:%s:type %d\n",rc,UneRequete.Message,UneRequete.Type ) ;

  printf("Received packet from %s:%d\n", inet_ntoa(sor.sin_addr), ntohs(sor.sin_port));
  /* reponse avec sor qui contient toujours l'adresse du dernier client */

  UneRequete.Type = Reponse ;
  strcat(UneRequete.Message," Client") ;
  rc = SendDatagram(Desc,&UneRequete,sizeof(struct Requete) ,&sor ) ;
  if ( rc == -1 )
    perror("SendDatagram:") ;
  else
    fprintf(stderr,"bytes:%d\n",rc ) ;
}


int main(int argc,char *argv[])
{
 int rc ;
 int Desc1, Desc2 ;
 char car ;

 struct sockaddr_in sthis1 ; /* this ce programme */
 struct sockaddr_in sthis2 ; /* this ce programme */
 fd_set readfs ;
 struct timeval tv ;
 u_long  IpSocket ;
 u_short PortSocket ;

 memset(&sthis1,0,sizeof(struct sockaddr_in)) ;
 memset(&sthis2,0,sizeof(struct sockaddr_in)) ;

 printf("Ceci est le serveur\n") ;
 if ( argc!=5)
 {
  printf("ser ser port ser port \n") ;
  exit(1) ;
```

```
}

/* Récupération IP & port   */
IpSocket= inet_addr(argv[1]);
PortSocket = atoi(argv[2]);
// Desc = CreateSockets(&psoo,&psos,,atoi(argv[2]),argv[3],atoi(argv[4])) ;
Desc1=creer_socket(SOCK_DGRAM,&IpSocket,PortSocket,&sthis1);
if ( Desc1 == -1 )
  die("CreateSockets") ;
else
  printf(" CreateSockets 1 : %d \n",Desc1) ;

IpSocket   = inet_addr(argv[3]);
PortSocket = atoi(argv[4]);
Desc2 = creer_socket(SOCK_DGRAM,&IpSocket,PortSocket,&sthis2);
if ( Desc2 == -1 )
  perror("CreateSockets") ;
else
  printf(" CreateSockets 2 : %d \n",Desc2) ;

while(1)
{

 FD_ZERO(&readfs) ;
 FD_SET(0,    &readfs); /* on teste le clavier */
 FD_SET(Desc1, &readfs);
 FD_SET(Desc2, &readfs);

 tv.tv_sec = 30 ; /* 30 secondes de timeout */
 tv.tv_usec = 100000 ; /* temps en micro secondes */

 // FD_SETSIZE Taille Maximum du set , Conseil de Doms Etienne */
 if((rc = select( FD_SETSIZE, &readfs, NULL, NULL, &tv)) < 0)
   die("select()");
 else
 if ( rc == 0 )
    printf ("Timeout !!!!!!! \n" ) ;
 else
 if(FD_ISSET( 0, &readfs))
  {
    car =  ReadChar() ;  /* fonction bibliothèque */
    printf("La touche pressee est %c \n", car  ) ;
    if (( car == 'q') || ( car == 'Q'))
            exit(0) ;
  }
 else
 if(FD_ISSET(Desc1, &readfs))
 {
```

```
    TraitementRequete(Desc1 ) ;
   }
   else
   if(FD_ISSET(Desc2, &readfs))
   {
    TraitementRequete(Desc2 ) ;
   }
 }
}
```

Mise en oeuvre de du select

L'instruction select permet d'écouter simultanément sur plusieurs ip & ports via des descripteur et indique sur quel descripteur on peut lire les données.

Un fd_set est un ensemble de bits qui va indiquer sur quel descripteurs il faut écouter. struct timeval tv est une structure qui permet d'indiquer à select combien de temps il faut écouter.

```
fd_set readfs ;
 struct timeval tv ;
```

On crée deux sockets distinctes afin de gérer nos deux ports . Cela nous donne deux descripteurs qui seront contenus respectivement dans **Desc1** et **Desc2**.

```
IpSocket= inet_addr(argv[1]);
 PortSocket = atoi(argv[2]);
 // Desc = CreateSockets(&psoo,&psos,,atoi(argv[2]),argv[3],atoi(argv[4])) ;
 Desc1=creer_socket(SOCK_DGRAM,&IpSocket,PortSocket,&sthis1);
 if ( Desc1 == -1 )
   die("CreateSockets") ;
 else
   printf(" CreateSockets 1 : %d \n",Desc1) ;

 IpSocket   = inet_addr(argv[3]);
 PortSocket = atoi(argv[4]);
 Desc2 = creer_socket(SOCK_DGRAM,&IpSocket,PortSocket,&sthis2);
 if ( Desc2 == -1 )
   perror("CreateSockets") ;
 else
   printf(" CreateSockets 2 : %d \n",Desc2) ;
```

FD_ZERO initialise le tableau de flags à zéro. Quand ce tableau est à zéro, aucun descripteur ne sera testé par le select. FD_SET permet d'indiquer quel descripteur il faut tester. Ici nous allons tester trois descripteurs 0,Desc1, Desc2 . 0 est celui du clavier. Attention par dédaut toute saisie clavier doit se terminer par enter.

```
   FD_ZERO(&readfs) ;
```

```
   FD_SET(0,     &readfs); /* on teste le clavier */
   FD_SET(Desc1, &readfs);
   FD_SET(Desc2, &readfs);
```

La structure tv indique le temps pendent lequel le select va écouter les descripteurs à savoir 30 secondes et 10000 micro secondes.

```
   tv.tv_sec = 30 ; /* 30 secondes de timeout */
   tv.tv_usec = 100000 ; /* temps en micro secondes */
```

Appel du select qui bloque jusqu'à ce que un descripteur soit disponible en lecture ou qu'un timeout se déclenche. En cas de timeout , le select retourne 0.

```
if((rc = select( FD_SETSIZE, &readfs, NULL, NULL, &tv)) < 0)
```

Tester si il y'a des données en provenance du clavier à lire.

```
if(FD_ISSET( 0, &readfs))
```

Permet de lire un caractère. Supprime automatiquement le newline vu qu'un enter est nécessaire pour que les données de clavier soient prises en compte.

```
       car =  ReadChar() ;
```

Permet de savoir si le descripteur Desc1 est disponible en lecture. Si oui, on peut lire des données.

```
    if(FD_ISSET(Desc1, &readfs))
```

Source Client

```
/*-------------------------------------------------------------
  Vanstapel Herman
  ex05\cli.c

 Le client dit bonjour en utilisant un structure  et
 le serveur fait de même
-------------------------------------------------------------*/
#include <stdio.h>
#include <string.h>
#include "../udplib/udplib.h"
#include "structure.h"

void die(char *s)
{
   perror(s);
   exit(1);
}

int main(int argc, char *argv[])
{
 int rc ;
 int Desc ;
 int tm ;

 u_long  IpSocket , IpServer;
 u_short PortSocket, PortServer ;

 struct sockaddr_in sthis ; /* this ce programme */
 struct sockaddr_in sos ; /* s = serveur */
 struct sockaddr_in sor ; /* r = remote */
 struct Requete UneRequete ;

 memset(&sthis,0,sizeof(struct sockaddr_in)) ;
 memset(&sos,0,sizeof(struct sockaddr_in)) ;
 memset(&sor,0,sizeof(struct sockaddr_in)) ;

 if (argc!=5)

{
   printf("cli client portc serveur ports\n") ;
   exit(1) ;
 }

 /* Récupération IP & port   */
```

```
IpSocket= inet_addr(argv[1]);
PortSocket = atoi(argv[2]);

IpServer = inet_addr(argv[3]) ;
PortServer = atoi(argv[4]);

// Desc = CreateSockets(&psoo,&psos,,atoi(argv[2]),argv[3],atoi(argv[4])) ;
Desc=creer_socket(SOCK_DGRAM,&IpSocket,PortSocket,&sthis);

if ( Desc == -1 )
  die("CreateSockets:") ;
else
  fprintf(stderr,"CreateSockets %d\n",Desc) ;

 sos.sin_family = AF_INET ;
 sos.sin_addr.s_addr= IpServer ;
 sos.sin_port = htons(PortServer) ;


UneRequete.Type = Question ;
strncpy(UneRequete.Message , "Avec une structure: Bonjour" , sizeof(UneRequete.Message)) ;

rc = SendDatagram(Desc,&UneRequete,sizeof(struct Requete) ,&sos ) ;

if ( rc == -1 )
  die("SendDatagram") ;
else
  fprintf(stderr,"Envoi de %d bytes\n",rc ) ;

memset(&UneRequete,0,sizeof(struct Requete)) ;
tm = sizeof(struct Requete) ;

 rc = ReceiveDatagram( Desc, &UneRequete,tm, &sor ) ;
if ( rc == -1 )
  die("ReceiveDatagram") ;
else
  fprintf(stderr,"bytes recus:%d:%s\n",rc,UneRequete.Message ) ;

close(Desc) ;
}
```

EXO7 : Client avec un timeout

Exécution du Client

```
vanstap@vanstap2:~/lib2016/ex07$ ./cli 127.0.0.1 1400 127.0.0.1 1300
port 1400
CreateSockets 3
Envoi du message 0 avec 88 bytes
bytes:88:Compteur 0:Hello Multiclient Client
Envoi du message 1 avec 88 bytes
bytes:88:Compteur 1:Hello Multiclient Client
Envoi du message 2 avec 88 bytes
bytes:88:Compteur 2:Hello Multiclient Client
Envoi du message 3 avec 88 bytes
bytes:88:Compteur 3:Hello Multiclient Client
Envoi du message 4 avec 88 bytes
bytes:88:Compteur 4:Hello Multiclient Client
Envoi du message 5 avec 88 bytes
error sur receive:: Interrupted system call
rc -1 errno:4
Envoi du message 5 avec 88 bytes
error sur receive:: Interrupted system call
rc -1 errno:4
Envoi du message 5 avec 88 bytes
error sur receive:: Interrupted system call
rc -1 errno:4
Envoi du message 5 avec 88 bytes
bytes:88:Compteur 5:Hello Multiclient Client
Envoi du message 6 avec 88 bytes
Doublon 5 !!!!!
doublon 5!!!!!
doublon 5 !!!!!
bytes:88:Compteur 6:Hello Multiclient Client
```

Tout les messages sont numérotés maintenant. On commence à zero.

J'envoie le message 0 , je déclenche le timer

```
Envoi du message 0 avec 88 bytes
```

Je reçoit le message 0 du serveur , j'arrète le timer

```
bytes:88:Compteur 0:Hello Multiclient Client
```

….

J'envoie le message 5

```
Envoi du message 5 avec 88 bytes
```

Je ne reçois pas le message car le serveur est en sleep et le timer qui expire me fait interrompre le ReceiveDatagram.

```
error sur receive:: Interrupted system call
```

Si je ne reçoit pas de réponse du serveur, je suppose qu'il n'a pas reçu mon message 5. Je renvoie donc une copie du message 5. Au total je renvoie trois fois le message car le serveur n'a pas répondu.

```
Envoi du message 5 avec 88 bytes
```

Je reçoit enfin le message 5 du serveur, je ne lis que le premier, les trois autres sont stocké par le stack udp.

```
bytes:88:Compteur 5:Hello Multiclient Client
```

Je passe au message suivant et envoie le message 6

```
Envoi du message 6 avec 88 bytes
```

Quand je me place en lecture, je lis en premier les messages 5 qui sont restés dans le stack, message 6 avant de lire le message 6

```
Doublon 5 !!!!!
doublon 5!!!!!
doublon 5 !!!!!
bytes:88:Compteur 6:Hello Multiclient Client
```

Exécution du serveur

```
vanstap@vanstap2:~/lib2016/ex07$ ./ser 127.0.0.1 1300
Ceci est le serveur
port 1300
 CreateSockets : 3
.bytes:88:Hello Multiclient:0
Received packet from 127.0.0.1:1400
bytes:88
bytes:88:Hello Multiclient:1
Received packet from 127.0.0.1:1400
bytes:88
bytes:88:Hello Multiclient:2
Received packet from 127.0.0.1:1400
bytes:88
bytes:88:Hello Multiclient:3
Received packet from 127.0.0.1:1400
bytes:88
bytes:88:Hello Multiclient:4
Received packet from 127.0.0.1:1400
```

```
bytes:88
^Zlongjumped from interrupt CTRL Z  20
Demarrage du sleep
Fin du sleep
bytes:88:Hello Multiclient:5
Received packet from 127.0.0.1:1400
bytes:88
bytes:88:Hello Multiclient:5
Received packet from 127.0.0.1:1400
bytes:88
bytes:88:Hello Multiclient:5
Received packet from 127.0.0.1:1400
bytes:88
bytes:88:Hello Multiclient:5
Received packet from 127.0.0.1:1400
bytes:88
bytes:88:Hello Multiclient:6
Received packet from 127.0.0.1:1400
bytes:88
```

Le serveir reçoit le premier message du client , affiche son ip et son port puis répond

```
.bytes:88:Hello Multiclient:0
Received packet from 127.0.0.1:1400
bytes:88
```

Je fait ctrl z pour mettre le serveur en pause pendant 30 secondes

```
^Zlongjumped from interrupt CTRL Z  20
Demarrage du sleep
Fin du sleep
```

Le serveur répond à tout les messages reçus, ils se sont accumulés dans le stack udp. Le serveur répond donc quatre fois. Une fois au message original puis au trois doublons.

```
bytes:88:Hello Multiclient:5
Received packet from 127.0.0.1:1400
bytes:88
bytes:88:Hello Multiclient:5
Received packet from 127.0.0.1:1400
bytes:88
bytes:88:Hello Multiclient:5
Received packet from 127.0.0.1:1400
bytes:88
bytes:88:Hello Multiclient:5
Received packet from 127.0.0.1:1400
bytes:88
```

Makefile

```
LIBS=
all:	cli	ser	udplib.o

udplib.o:	../udplib/udplib.h	../udplib/udplib.c
	echo "Compilation de udplib.o"
	gcc-c ../udplib/udplib.c

cli:	cli.c	udplib.o
	echo "Compilation de client"
	gcc -o cli  cli.c udplib.o   $(LIBS)

ser:	ser.c	udplib.o
	echo "Compilation de serveur"
	gcc-o ser  ser.c udplib.o $(LIBS)
```

Source Serveur

```
/*-------------------------------------------------------------------------------------------
  Herman vanstapel

 ex07\ser.c :  Un serveur avec plusieurs clients qui affiche chaque fois l'ip et le port du client connecté
-------------------------------------------------------------------------------------------*/

#include <stdio.h>
#include <string.h>
#include "../udplib/udplib.h"
#include "structure.h"
#include <setjmp.h>
#include <signal.h>
#include <sys/types.h>

static void signal_handler(int sig)
{
  // siglongjmp(env,sig);
  switch (sig)      {
            case SIGINT:
                printf("longjumped from interrupt CRTL C  %d\n",SIGINT);
                exit(0) ;
           /* On fermerait les fichiers */
                break;
            case SIGTSTP:
           printf("longjumped from interrupt CTRL Z  %d\n",SIGTSTP);
           printf("Demarrage du sleep \n") ;
           sleep(30) ;
           printf("Fin du sleep \n") ;
           break ;
     } ;
}

void die(char *s)
{
   perror(s);
   exit(1);
}

int main(int argc,char *argv[])
{
 int rc ;
 int Desc ;

 struct sockaddr_in sthis ; /* this ce programme */
 struct sockaddr_in sos ; /* s = serveur */
```

```
struct sockaddr_in sor ; /* r = remote */

u_long  IpSocket ;
u_short PortSocket ;

int tm ;
struct Requete UneRequete ;
char Tampon[40] ;
int Ip, Port ;

memset(&sthis,0,sizeof(struct sockaddr_in)) ;
memset(&sor,0,sizeof(struct sockaddr_in)) ;

printf("Ceci est le serveur\n") ;
if ( argc!=3)
{
 printf("ser ser port \n") ;
 exit(1) ;
}

/* Récupération IP & port   */
IpSocket= inet_addr(argv[1]);
PortSocket = atoi(argv[2]);
// Desc = CreateSockets(&psoo,&psos,,atoi(argv[2]),argv[3],atoi(argv[4])) ;
Desc=creer_socket(SOCK_DGRAM,&IpSocket,PortSocket,&sthis);

if ( Desc == -1 )
  die("CreateSockets") ;
else
  printf(" CreateSockets : %d \n",Desc) ;

while(1)
{

 signal(SIGINT, signal_handler);
 signal(SIGTSTP, signal_handler) ;

 memset(&UneRequete,0,sizeof(UneRequete)) ;
 tm = sizeof(struct Requete) ;
 rc = ReceiveDatagram( Desc,&UneRequete ,tm, &sor ) ;
 if ( rc == -1 )
   die("ReceiveDatagram") ;
 else
    fprintf(stderr,"bytes:%d:%s:%d\n",rc,UneRequete.Message, UneRequete.Compteur );

  printf("Received packet from %s:%d\n", inet_ntoa(sor.sin_addr), ntohs(sor.sin_port));
 /* reponse avec psor qui contient toujours l'adresse du dernier client */
```

```
 UneRequete.Type = Reponse ;
 strcat(UneRequete.Message," Client") ;
 rc = SendDatagram(Desc,&UneRequete,sizeof(struct Requete) ,&sor ) ;
 if ( rc == -1 )
  die("SendDatagram:") ;
 else
  fprintf(stderr,"bytes:%d\n",rc ) ;
}
}
```

La fonction signal_handler est installée pour traiter le signal SIGSTP qui sera généré par l'appel à la combinaison de touche ctrl Z. Dans ce cas on fait un sleep de 30 secondes , ce qui déclenchera des timeout du coté client.

```
static void signal_handler(int sig)
{
 // siglongjmp(env,sig);
 switch (sig)      {
          case SIGINT:
              printf("longjumped from interrupt CRTL C  %d\n",SIGINT);
              exit(0) ;
         /* On fermerait les fichiers */
              break;
          case SIGTSTP:
         printf("longjumped from interrupt CTRL Z  %d\n",SIGTSTP);
         printf("Demarrage du sleep \n") ;
         sleep(30) ;
```

On installe les handlers avec la primitive sigaction. La différence entre Signal et sigaction , signal relance toujours l'appel système interrompu contrairement à sigaction qui l'interrompt définitivement

```
 signal(SIGINT, signal_handler);
 signal(SIGTSTP, signal_handler) ;
```

La primitive receiveDatagram est automatiquement redémarrée après interruption par le handler.

```
 rc = ReceiveDatagram( Desc,&UneRequete ,tm, &sor ) ;
```

Source Client

```
/*------------------------------------------------------------
  Vanstapel Herman
  Ex07\cli.c
```

```
 Le client dit bonjour et
 le serveur fait de même
-----------------------------------------------------------------*/
#include <stdio.h>
#include <unistd.h>
#include <signal.h>
#include <sys/types.h>  /* ces deux include pour le getpid */
#include <string.h>
#include <errno.h>
#include "../udplib/udplib.h"
#include "structure.h"


int NbrAlarmes=0 ;

static void signal_handler(int sig)
{

}

void die(char *s)
{
   perror(s);
   exit(1);
}


int main(int argc, char *argv[])
{
 int rc ;
 int Desc ;
 int tm ;
 int Port, Ip ;
 char Tampon[80] ;
 unsigned int time_interval  ;
 int ret ;
 int Compteur = 0 ;
 struct sigaction act;

 struct sockaddr_in sthis ; /* this ce programme */
 struct sockaddr_in sos ; /* s = serveur */
 struct sockaddr_in sor ; /* r = remote */

 u_long  IpSocket , IpServer;
 u_short PortSocket, PortServer ;

 struct Requete UneRequeteE ;
 struct Requete UneRequeteR ;
```

```
 memset(&sthis,0,sizeof(struct sockaddr_in)) ;
memset(&sos,0,sizeof(struct sockaddr_in)) ;
memset(&sor,0,sizeof(struct sockaddr_in)) ;

if (argc!=5)

{
   printf("cli client portc serveur ports\n") ;
   exit(1) ;
}


/* Récupération IP & port   */
IpSocket= inet_addr(argv[1]);
PortSocket = atoi(argv[2]);

IpServer = inet_addr(argv[3]) ;
PortServer = atoi(argv[4]);

// Desc = CreateSockets(&psoo,&psos,,atoi(argv[2]),argv[3],atoi(argv[4])) ;
Desc=creer_socket(SOCK_DGRAM,&IpSocket,PortSocket,&sthis);

if ( Desc == -1 )
   die("CreateSockets:") ;
else
   fprintf(stderr,"CreateSockets %d\n",Desc) ;

sos.sin_family = AF_INET ;
 sos.sin_addr.s_addr= IpServer ;
 sos.sin_port = htons(PortServer) ;

Compteur = 0 ;
while(1)
{
  struct sigaction act;

 act.sa_handler = signal_handler;
 act.sa_flags = 0 ;
 sigemptyset (&act.sa_mask);

 rc = sigaction (SIGALRM, &act, NULL);


  UneRequeteE.Compteur = Compteur ;
  UneRequeteE.Type = Question ;
  strncpy(UneRequeteE.Message , "Hello Multiclient" , sizeof(UneRequeteE.Message)) ;
  redo:
```

```
  time_interval = 9 ;
  ret = alarm(time_interval);
  rc = SendDatagram(Desc,&UneRequeteE,sizeof(struct Requete) ,&sos ) ;

  if ( rc == -1 )
    die("SendDatagram") ;
  else
    fprintf(stderr,"Envoi du message %d avec %d bytes\n",UneRequeteE.Compteur, rc ) ;

  while(1) /* on boucle tant que l'on ne reçoit pas le bon message */
  {
   memset(&UneRequeteR,0,sizeof(struct Requete)) ;
   tm = sizeof(struct Requete) ;
   rc = ReceiveDatagram( Desc, &UneRequeteR,tm, &sor ) ;
   if ( rc<=0)
     {
      perror("error sur receive:") ;
      fprintf(stderr,"rc %d errno:%d \n",rc,errno) ;
      goto redo ;
     }
   if ( UneRequeteR.Compteur != Compteur )
     printf("doublon !!!!!\n") ;
   else
     break ;
  }
  /* fin de l'alarme */
  ret = alarm(0);
  fprintf(stderr,"bytes:%d:Compteur %d:%s\n",rc,UneRequeteR.Compteur,UneRequeteR.Message ) ;

  Compteur++ ;
  sleep(5) ;
 }

 close(Desc) ;
}
```

Handler de traitement de la fonction alarm. Il ne contient pas du code. Il sera appellé par le signal alarm et aura pour effet d'interrompre la fonction réseau ReceiveDatagram(

```
static void signal_handler(int sig)
{

}
```

Le code suivant associe la fonction signal_handler au signal alarm par l'intermédiaire de la primitive sigaction.

```
struct sigaction act;

 act.sa_handler = signal_handler;
 act.sa_flags = 0 ;
 sigemptyset (&act.sa_mask);

 rc = sigaction (SIGALRM, &act, NULL);
```

Insistons sur le fait que tout les messages sont maintenant numérotés ce qui est nécessaire pour détecter les doublons.

```
  UneRequeteE.Compteur = Compteur ;
```

Quand on voudra retransmettre un message suite à l'expiration de l'alarme ou timeour , on fera un branchement ici

```
redo:
```

On fixe un délai de 9 secondes pour le timeout et l'appel de la fonction alarm provoquera ce timeout dans neuf secondes.

```
  time_interval = 9 ;
  ret = alarm(time_interval);
  rc = SendDatagram(Desc,&UneRequeteE,sizeof(struct Requete) ,&sos ) ;
```

Je me mets en attente de la réception de la réponse à mon message envoyé. Si je ne reçoit pas de réponse, Un déclenchement du timout provoquera un arret de l'appel système Receidatagram avec le code d'erreur -1 , message Interrupted system call. Dans ce cas on fait un branchement à redo

```
   if ( rc<=0)
     {
      perror("error sur receive:") ;
      fprintf(stderr,"rc %d errno:%d \n",rc,errno) ;
      goto redo ;
```

Si le timer n'expire, j'ai alors reçu une réponse mais il faut voir que la réponse envoyée par le serveur correspond au message envoyé via le numéro. Si ce n'est pas le cas on à affaire à un doublon , message en double reçut suite à des timeouts avec le précédent message

```
   if ( UneRequeteR.Compteur != Compteur )
     printf("doublon !!!!!\n") ;  // je dois boucler à nouveau, j'ai pas le bon message
   else
     break ; // J'ai reçu la bonne réponse, je sort de la boucle pour passer au message suivant
```

EXO7TI : Client avec un timeout udp intégré

Introduction

Ici au lieu d'utiliser la fonction alarm, on va utiliser une des options des socket qui permet de lancer un timer chaque fois qu'un message est envoyé. Cette option est malheureusement considérée comme non fiable dans le sens qu'elle ne fonctionne pas avec toutes les variantes de unix. Elle fonctionne avec ubuntu **mais pas avec sunos**.

Exécution du serveur

```
vanstap@vanstap2:~/lib2016/ex07TI$ ./ser 127.0.0.1 1300
Ceci est le serveur
port 1300
 CreateSockets : 3
bytes:88:Hello Multiclient:0
Received packet from 127.0.0.1:1500
bytes:88
bytes:88:Hello Multiclient:1
Received packet from 127.0.0.1:1500
bytes:88
^Zlongjumped from interrupt CTRL Z  20
Demarrage du sleep
Fin du sleep
bytes:88:Hello Multiclient:2
Received packet from 127.0.0.1:1500
bytes:88
bytes:88:Hello Multiclient:2
Received packet from 127.0.0.1:1500
bytes:88
bytes:88:Hello Multiclient:2
Received packet from 127.0.0.1:1500
bytes:88
bytes:88:Hello Multiclient:3
Received packet from 127.0.0.1:1500
bytes:88
^Clongjumped from interrupt CRTL C  2
```

Exécution du Client

```
vanstap@vanstap2:~/lib2016/ex07TI$ ./cli 127.0.0.1 1500 127.0.0.1 1300
port 1500
CreateSockets 3
Envoi du message 0 avec 88 bytes
Reçu bytes:88:Compteur 0:Hello Multiclient Client
Envoi du message 1 avec 88 bytes
Reçu bytes:88:Compteur 1:Hello Multiclient Client
Envoi du message 2 avec 88 bytes
rc -1 errno:11
receive:: Resource temporarily unavailable
```

```
Envoi du message 2 avec 88 bytes
rc -1 errno:11
receive:: Resource temporarily unavailable
Envoi du message 2 avec 88 bytes
Reçu bytes:88:Compteur 2:Hello Multiclient Client
Envoi du message 3 avec 88 bytes
Message 2 doublon !!!!!
Message 2 doublon !!!!!
Reçu bytes:88:Compteur 3:Hello Multiclient Client
^C
```

Makefile

```
LIBS=
all:	cli	ser	udplib.o

udplib.o:	../udplib/udplib.h	../udplib/udplib.c
	echo "Compilation de udplib.o"
	gcc-c ../udplib/udplib.c

cli:	cli.c	udplib.o
	echo "Compilation de client"
	gcc -o cli  cli.c udplib.o   $(LIBS)

ser:	ser.c	udplib.o
	echo "Compilation de serveur"
	gcc-o ser  ser.c udplib.o $(LIBS)
```

Source Serveur

```
/*------------------------------------------------------------------------------------------
  Herman vanstapel

 ex07ti\ser.c :  Un serveur avec plusieurs clients qui affiche chaque fois l'ip et le port du client
connecté
------------------------------------------------------------------------------------------*/

#include <stdio.h>
#include <string.h>
#include "../udplib/udplib.h"
#include "structure.h"
#include <setjmp.h>
#include <signal.h>
#include <sys/types.h>

static void signal_handler(int sig)
{
```

```
 // siglongjmp(env,sig);
 switch (sig)      {
          case SIGINT:
              printf("longjumped from interrupt CRTL C  %d\n",SIGINT);
              exit(0) ;
          /* On fermerait les fichiers */
              break;
          case SIGTSTP:
          printf("longjumped from interrupt CTRL Z  %d\n",SIGTSTP);
          printf("Demarrage du sleep \n") ;
          sleep(30) ;
          printf("Fin du sleep \n") ;
          break ;
    } ;
}

void die(char *s)
{
   perror(s);
   exit(1);
}

int main(int argc,char *argv[])
{
 int rc ;
 int Desc ;

 struct sockaddr_in sthis ; /* this ce programme */
 struct sockaddr_in sos ; /* s = serveur */
 struct sockaddr_in sor ; /* r = remote */

 u_long  IpSocket ;
 u_short PortSocket ;

 int tm ;
 struct Requete UneRequete ;
 char Tampon[40] ;
 int Ip, Port ;

 memset(&sthis,0,sizeof(struct sockaddr_in)) ;
 memset(&sor,0,sizeof(struct sockaddr_in)) ;

 printf("Ceci est le serveur\n") ;
 if ( argc!=3)
 {
  printf("ser ser port \n") ;
  exit(1) ;
 }
```

```
/* Récupération IP & port   */
IpSocket= inet_addr(argv[1]);
PortSocket = atoi(argv[2]);
// Desc = CreateSockets(&psoo,&psos,,atoi(argv[2]),argv[3],atoi(argv[4])) ;
Desc=creer_socket(SOCK_DGRAM,&IpSocket,PortSocket,&sthis);

if ( Desc == -1 )
  die("CreateSockets") ;
else
  printf(" CreateSockets : %d \n",Desc) ;

while(1)
{

 signal(SIGINT, signal_handler);
 signal(SIGTSTP, signal_handler) ;

 memset(&UneRequete,0,sizeof(UneRequete)) ;
 tm = sizeof(struct Requete) ;
 rc = ReceiveDatagram( Desc,&UneRequete ,tm, &sor ) ;
 if ( rc == -1 )
  die("ReceiveDatagram") ;
 else
   fprintf(stderr,"bytes:%d:%s:%d\n",rc,UneRequete.Message,ntohl(UneRequete.Compteur));

  printf("Received packet from %s:%d\n", inet_ntoa(sor.sin_addr), ntohs(sor.sin_port));
 /* reponse avec psor qui contient toujours l'adresse du dernier client */

 UneRequete.Type = htonl(Reponse) ;
 strcat(UneRequete.Message," Client") ;
 rc = SendDatagram(Desc,&UneRequete,sizeof(struct Requete) ,&sor ) ;
 if ( rc == -1 )
  die("SendDatagram:") ;
 else
  fprintf(stderr,"bytes:%d\n",rc ) ;
}
}
```

Source Client

```
/*-------------------------------------------------------------
  Vanstapel Herman
  ex07ti\cli.c

 Le client dit bonjour et
```

```
 le serveur fait de même
-----------------------------------------------------------------*/
#include <stdio.h>
#include <setjmp.h>
#include <signal.h>
#include <sys/types.h>  /* ces deux include pour le getpid */
#include <string.h>
#include <errno.h>
#include "../udplib/udplib.h"
#include "structure.h"

void die(char *s)
{
   perror(s);
   exit(1);
}

int main(int argc, char *argv[])
{
 int rc ;
 int Desc ;
 int tm ;
 int Port, Ip ;
 char Tampon[80] ;
 int returned_from_longjump ;
 unsigned int time_interval  ;
 int ret ;
 int Compteur = 0 ;
 struct timeval tv;

 struct sockaddr_in sthis ; /* this ce programme */
 struct sockaddr_in sos ; /* s = serveur */
 struct sockaddr_in sor ; /* r = remote */
 u_long  IpSocket , IpServer;
 u_short PortSocket, PortServer ;

 struct Requete UneRequeteE ;
 struct Requete UneRequeteR ;

 memset(&sthis,0,sizeof(struct sockaddr_in)) ;
 memset(&sos,0,sizeof(struct sockaddr_in)) ;
 memset(&sor,0,sizeof(struct sockaddr_in)) ;

 if (argc!=5)

{
   printf("cli client portc serveur ports\n") ;
   exit(1) ;
```

```
}

/* Récupération IP & port   */
IpSocket= inet_addr(argv[1]);
PortSocket = atoi(argv[2]);

IpServer = inet_addr(argv[3]) ;
PortServer = atoi(argv[4]);

// Desc = CreateSockets(&psoo,&psos,,atoi(argv[2]),argv[3],atoi(argv[4])) ;
Desc=creer_socket(SOCK_DGRAM,&IpSocket,PortSocket,&sthis);

if ( Desc == -1 )
  die("CreateSockets:") ;
else
  fprintf(stderr,"CreateSockets %d\n",Desc) ;
 tv.tv_sec = 10;
   tv.tv_usec = 0;
   if (setsockopt( Desc, SOL_SOCKET, SO_RCVTIMEO,&tv,sizeof(tv)) < 0) {
                    perror("Error");
          }

Compteur = 0 ;
while(1)
{
 sos.sin_family = AF_INET ;
 sos.sin_addr.s_addr= IpServer ;
 sos.sin_port = htons(PortServer) ;


 UneRequeteE.Compteur = htonl(Compteur) ;
 UneRequeteE.Type = htonl(Question) ;
 strncpy(UneRequeteE.Message , "Hello Multiclient" , sizeof(UneRequeteE.Message)) ;

 redo:
 rc = SendDatagram(Desc,&UneRequeteE,sizeof(struct Requete) ,&sos ) ;

 if ( rc == -1 )
   perror("SendDatagram") ;
 else
   fprintf(stderr,"Envoi du message %d avec %d bytes\n",ntohl(UneRequeteE.Compteur), rc ) ;

 while(1) /* on boucle tant que l'on ne reçoit pas le bon message */
 {
  memset(&UneRequeteR,0,sizeof(struct Requete)) ;
  tm = sizeof(struct Requete) ;
  rc = ReceiveDatagram( Desc, &UneRequeteR,tm, &sor ) ;
  if (rc<0)
```

```
   {
     fprintf(stderr,"rc %d errno:%d \n",rc,errno) ;
     perror("receive:") ;
         if ( errno==EAGAIN )
           goto redo ;
         else
           exit(0) ;
   }
   else
    if (ntohl(UneRequeteR.Compteur) != Compteur )
      printf("Message %d doublon !!!!!\n",ntohl(UneRequeteR.Compteur)) ;
   else
     break ;
  }
  fprintf(stderr,"Reçu bytes:%d:Compteur
%d:%s\n",rc,ntohl(UneRequeteR.Compteur),UneRequeteR.Message ) ;
  Compteur++ ;
  sleep(5) ;
 }

 close(Desc) ;
}
```

Les lignes suivantes fixent le timout à 10 secondes pour chaque envoi de données

```
  tv.tv_sec = 10;
     tv.tv_usec = 0;
     if (setsockopt( Desc, SOL_SOCKET, SO_RCVTIMEO,&tv,sizeof(tv)) <
0) {
                perror("Error");
```

Quand un timeout expire

```
rc = ReceiveDatagram( Desc, &UneRequeteR,tm, &psor ) ;
     if (rc<0)
     {
        fprintf(stderr,"rc %d errno:%d \n",rc,errno) ;
        perror("receive:") ;
      if ( errno==EAGAIN )
```

```
        goto redo ;
```

ReceiveDatagram est interrompu et le code erreur vaut EAGAIN dans ce cas.

ExO8 la conversion avec ntoh, hton

Excécution du serveur

```
vanstap@vanstap2:~/lib2016/ex08$ ./ser 127.0.0.1 1300
Ceci est le serveur
port 1300
CreateSockets 3
bytes reçus:44:Avec une structure: Bonjour
Type reçu 1
bytes envoyés:44
```

Exécution du client

```
vanstap@vanstap2:~/lib2016/ex08$ ./cli 127.0.0.1 1400 127.0.0.1 1300
port 1400
CreateSockets 3
Envoi de 44 bytes
bytes recus:44:Avec une structure: Bonjour Client
Type recu : 2
```

Source du client

```
/*------------------------------------------------------------
  Vanstapel Herman
  ex08\cli.c

 Le client dit bonjour en utilisant un structure  et
 le serveur fait de même
------------------------------------------------------------*/
#include <stdio.h>
#include <string.h>
#include "../udplib/udplib.h"
#include "structure.h"

void die(char *s)
{
   perror(s);
   exit(1);
}
```

```
int main(int argc, char *argv[])
{
 int rc ;
 int Desc ;
 int tm ;

 u_long  IpSocket , IpServer;
 u_short PortSocket, PortServer ;

 struct sockaddr_in sthis ; /* this ce programme */
 struct sockaddr_in sos ; /* s = serveur */
 struct sockaddr_in sor ; /* r = remote */
 struct Requete UneRequete ;

 memset(&sthis,0,sizeof(struct sockaddr_in)) ;
 memset(&sos,0,sizeof(struct sockaddr_in)) ;
 memset(&sor,0,sizeof(struct sockaddr_in)) ;

 if (argc!=5)

{
   printf("cli client portc serveur ports\n") ;
   exit(1) ;
 }

 /* Récupération IP & port   */
 IpSocket= inet_addr(argv[1]);
 PortSocket = atoi(argv[2]);

 IpServer = inet_addr(argv[3]) ;
 PortServer = atoi(argv[4]);

 // Desc = CreateSockets(&psoo,&psos,,atoi(argv[2]),argv[3],atoi(argv[4])) ;
 Desc=creer_socket(SOCK_DGRAM,&IpSocket,PortSocket,&sthis);

 if ( Desc == -1 )
   die("CreateSockets:") ;
 else
   fprintf(stderr,"CreateSockets %d\n",Desc) ;

  sos.sin_family = AF_INET ;
  sos.sin_addr.s_addr= IpServer ;
  sos.sin_port = htons(PortServer) ;


 UneRequete.Type = htonl (Question) ;
 strncpy(UneRequete.Message , "Avec une structure: Bonjour" , sizeof(UneRequete.Message)) ;
```

```
 rc = SendDatagram(Desc,&UneRequete,sizeof(struct Requete) ,&sos ) ;

 if ( rc == -1 )
   die("SendDatagram") ;
 else
  fprintf(stderr,"Envoi de %d bytes\n",rc ) ;

 memset(&UneRequete,0,sizeof(struct Requete)) ;
 tm = sizeof(struct Requete) ;

  rc = ReceiveDatagram( Desc, &UneRequete,tm, &sor ) ;
 if ( rc == -1 )
   die("ReceiveDatagram") ;
 else
  {
  fprintf(stderr,"bytes recus:%d:%s\n",rc,UneRequete.Message ) ;
  fprintf(stderr,"Type recu : %d \n", ntohl(UneRequete.Type)) ;
  }
 close(Desc) ;
}
```

Tout entier ou constante contenu dans une structure de donnée doit être convertit avant la transmisson sur internet car les processeurs intel utilisent la réprésentation little endian et Internet Big Endian. Avant d'envoyer les données sur le réseau, il faut les convertir avec htonl

```
UneRequete.Type = htonl (Question) ;
```

Pour les données reçues du réseau Il faut donc convertir les entiers avec ntohl, littéralement network to host long 32 bits étant considéré comme long.

```
  fprintf(stderr,"Type recu : %d \n", ntohl(UneRequete.Type)) ;
```

Source du serveur

```
/*-------------------------------------
  Herman Vanstapel

  ex08\ser.c

  Un serveur recevant une structure
---------------------------------------*/

#include <stdio.h>
#include <string.h>
#include "../udplib/udplib.h"
#include "structure.h"
```

```
void die(char *s)
{
   perror(s);
   exit(1);
}

int main(int argc,char *argv[])
{
 int rc ;
 int Desc ;
 struct sockaddr_in sthis ; /* this ce programme */
 struct sockaddr_in sos ; /* s = serveur */
 struct sockaddr_in sor ; /* r = remote */

 u_long  IpSocket ;
 u_short PortSocket ;

 int tm ;
 struct Requete UneRequete ;

 memset(&sthis,0,sizeof(struct sockaddr_in)) ;
 memset(&sos,0,sizeof(struct sockaddr_in)) ;
 memset(&sor,0,sizeof(struct sockaddr_in)) ;

 printf("Ceci est le serveur\n") ;
 if ( argc!=3)
 {
  printf("ser ser port cli\n") ;
  exit(1) ;
 }

 /* Récupération IP & port   */
 IpSocket= inet_addr(argv[1]);
 PortSocket = atoi(argv[2]);
 // Desc = CreateSockets(&psoo,&psos,,atoi(argv[2]),argv[3],atoi(argv[4])) ;
 Desc=creer_socket(SOCK_DGRAM,&IpSocket,PortSocket,&sthis);

 if ( Desc == -1 )
   die("CreateSockets:") ;
 else
   fprintf(stderr,"CreateSockets %d\n",Desc) ;

 tm = sizeof(struct Requete) ;
 rc = ReceiveDatagram( Desc,&UneRequete ,tm, &sor ) ;
 if ( rc == -1 )
   die("ReceiveDatagram") ;
 else
```

```
  fprintf(stderr,"bytes reçus:%d:%s\n",rc,UneRequete.Message ) ;

 printf("Type reçu %d\n", ntohl(UneRequete.Type)) ;
 /* attention l'enum peut être codé en short */
 /* reponse avec psos */

 UneRequete.Type = htonl(Reponse) ;
 strcat(UneRequete.Message," Client") ;
 rc = SendDatagram(Desc,&UneRequete,sizeof(struct Requete) ,&sor ) ;
 if ( rc == -1 )
   die("SendDatagram:") ;
 else
  fprintf(stderr,"bytes envoyés:%d\n",rc ) ;

 close(Desc) ;
}
```

Annexe

Little Endian Big Endian

Stocker des mots en mémoire

Nos avons un mot de 32 birs. Ceci est équivalent à 4 Bytes. Les entiers 32 bits , les nombres flotant simple précisions, une IP sont tous de 32 bits de long. Comment pouvons nous stocker ces valeurs en mémoire ? Après tout, chaque adresse mémoire peut stocker un seul byte et pas quatre bytes.

La réponse est simple. Nous découpons le mot de 32 bits en 4 bytes. Par exemple, supposons que nous avons une quantité de 32 bits qui vaut 90AB12CD16, Ce qui est de l' hexadecimal. Comme chaque code hexa représente 4 bits, nous avons besoin de 8 code hexa pour représenter la valeur de 32 bits.

Ainsi, les 4 bytes sont: 90, AB, 12, CD qui chacun représent 2 hex digits.
Il apparait qu'il y'a deux moyens de stocker ces bytes en mémoire.

Big Endian

En big endian, vous stocker le Byte le plus significatif dans l'adresse mémoire la plus petite Cela donne ceci:

Addresse	Valeur
1000	90
1001	AB
1002	12
1003	CD

C'est la réprésentation adoptée par internet pour ses ips.

Little Endian

En little endian, vous stocker le byte le moins significatif dans l'adresse mémoire la plus petite. Cela donne ceci :

Addresse	Valeur
1000	CD
1001	12
1002	AB
1003	90

C'est la réprésentation adoptée par les processeur intels.
On notera que c'est exactement l'ordre inverse comparé au BIG ENDIAN. Pour se souvenir qui est quoi, rappeler vous que quand le byte le moins significatif est stocké en premier on parle de little endian ou quand le Byte le plus significatif est stocké en premier on parle de Big endian

Source : http://www.cs.umd.edu/class/sum2003/cmsc311/Notes/Data/endian.html

Les opérations sur les fichiers

Tous ces fichiers se trouvent dans *le répertoire Fichiers* des librairies sauf AccesDirect.c

Le fichier structure

```
struct Record {
         int Numero ;
       int Valeur ;
       char Memo[80] ;
      } ;
```

Le fichier ecriture

```
/****************************************
  Herman Vanstapel
  2007
****************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "structure.h"

void DelNewLine(char *Chaine)
{
 Chaine[strlen(Chaine)-1] = 0 ;
}

char ReadChar()
{
 char Tampon[80] ;
 fgets(Tampon,sizeof Tampon,stdin ) ;
 return Tampon[0] ;
}

void SaiSieRecord(struct Record *UnRecord )
{
 char Tampon[80] ;
 printf("Saisie numero :") ;
 fgets(Tampon,sizeof Tampon,stdin ) ;
 UnRecord -> Numero = atoi(Tampon) ;
 printf("Saisie valeur :") ;
 fgets(Tampon,sizeof Tampon,stdin ) ;
 UnRecord -> Valeur = atoi(Tampon) ;
 printf("Saisie Tampon :") ;
 fgets(UnRecord->Memo,sizeof UnRecord->Memo,stdin ) ;
 DelNewLine(UnRecord->Memo) ;
 printf("%d\n",UnRecord->Numero ) ;
 printf("%d\n",UnRecord->Valeur ) ;
 printf("%s\n",UnRecord->Memo) ;
 printf("----------------------\n") ;
 return ;
}
```

```
main()
{
 struct Record UnRecord ;
 FILE *sortie ;
 char NomFichier[80] ;
 char Redo ;

 printf("Le nom de fichier à creer :)") ;
 fgets(NomFichier,sizeof NomFichier,stdin) ;
 DelNewLine(NomFichier) ;
 sortie = fopen(NomFichier,"a") ; /* Si le fichier existe, on le cree sinon
on ajoute */
 if ( sortie == NULL )
    {
     fprintf(stderr,"Echec Ouverture\n") ;
     exit(0) ;
    }
 else
    fprintf(stderr,"Ouverture reussie \n") ;
 setvbuf(sortie, (char *)NULL,  IOLBF, 0) ; /* ceci supprime la
bufferisation */
 Redo='y' ;
 while ( Redo=='Y' || Redo=='y')
 {
  int nbr ;
  printf("Position actuelle dans le fichier %d\n",ftell(sortie)) ;
  SaiSieRecord(&UnRecord ) ;
  nbr = fwrite(&UnRecord,sizeof(UnRecord),1,sortie) ;
  fflush(sortie) ;
  fprintf(stderr,"%d Bytes écrits\n",nbr) ; /* affiche le nombre de record
et pas de bytes */
  printf("Encoder un autre (Y/N) ?)") ;
  Redo=ReadChar() ;
 }
 fclose(sortie) ;
}
```

Le fichier Lecture

```
/****************************************
  Herman Vanstapel
  2007
****************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "structure.h"

void DelNewLine(char *Chaine)
{
 Chaine[strlen(Chaine)-1] = 0 ;
}

char ReadChar()
{
 char Tampon[80] ;
 fgets(Tampon,sizeof Tampon,stdin ) ;
 return Tampon[0] ;
}


void AfficheRecord(struct Record *UnRecord)
{
printf("%d\n",UnRecord->Numero ) ;
 printf("%d\n",UnRecord->Valeur ) ;
 printf("%s\n",UnRecord->Memo) ;
 printf("----------------------\n") ;
}

void SaiSieRecord(struct Record *UnRecord )
{
 char Tampon[80] ;
 printf("Saisie numero :") ;
 fgets(Tampon,sizeof Tampon,stdin ) ;
 UnRecord -> Numero = atoi(Tampon) ;
 printf("Saisie valeur :") ;
 fgets(Tampon,sizeof Tampon,stdin ) ;
 UnRecord -> Valeur = atoi(Tampon) ;
 printf("Saisie Tampon :") ;
 fgets(UnRecord->Memo,sizeof UnRecord->Memo,stdin ) ;
 DelNewLine(UnRecord->Memo) ;
 AfficheRecord(UnRecord) ;
 return ;
}

main()
{
 struct Record UnRecord ;
 FILE *sortie ;
 char NomFichier[80] ;
 char Tampon[80] ;
 int  Numero ;
 int  nbr ;
```

```
 printf("Le nom de fichier à Lire :)") ;
 fgets(NomFichier,sizeof NomFichier,stdin) ;
 DelNewLine(NomFichier) ;
 sortie = fopen(NomFichier,"r") ; /* Si le fichier existe, on le cree sinon
on ajoute */
 if ( sortie == NULL )
    {
     fprintf(stderr,"Echec Ouverture\n") ;
     exit(0) ;
    }
 else
    fprintf(stderr,"Ouverture reussie \n") ;

 printf("Saisie numero :") ;
 fgets(Tampon,sizeof Tampon,stdin ) ;
 Numero = atoi(Tampon) ;
 nbr = fread(&UnRecord,sizeof(UnRecord),1,sortie) ;

 while ( !feof(sortie) && UnRecord.Numero != Numero )
 {
  int nbr ;
  fprintf(stderr,"Record lu %d et Position actuelle dans le fichier
%d\n",nbr,ftell(sortie)) ;
  nbr = fread(&UnRecord,sizeof(UnRecord),1,sortie) ;
 }
 if ( !feof(sortie) )
      AfficheRecord(&UnRecord) ;
 fclose(sortie) ;
}
```

Le fichier AccesDirect

```
/*****************************************
  Herman Vanstapel
  2007
*****************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "structure.h"

void DelNewLine(char *Chaine)
{
 Chaine[strlen(Chaine)-1] = 0 ;
}

char ReadChar()
{
 char Tampon[80] ;
 fgets(Tampon,sizeof Tampon,stdin ) ;
 return Tampon[0] ;
}


void AfficheRecord(struct Record *UnRecord)
{
printf("%d\n",UnRecord->Numero ) ;
 printf("%d\n",UnRecord->Valeur ) ;
 printf("%s\n",UnRecord->Memo) ;
 printf("----------------------\n") ;
}

void SaiSieRecord(struct Record *UnRecord )
{
 char Tampon[80] ;
 printf("Saisie numero :") ;
 fgets(Tampon,sizeof Tampon,stdin ) ;
 UnRecord -> Numero = atoi(Tampon) ;
 printf("Saisie valeur :") ;
 fgets(Tampon,sizeof Tampon,stdin ) ;
 UnRecord -> Valeur = atoi(Tampon) ;
 printf("Saisie Tampon :") ;
 fgets(UnRecord->Memo,sizeof UnRecord->Memo,stdin ) ;
 DelNewLine(UnRecord->Memo) ;
 AfficheRecord(UnRecord) ;
 return ;
}

main()
```

```
{
 struct Record UnRecord ;
 FILE *Fichier ;
 char NomFichier[80] ;
 char Tampon[80] ;
 int  Numero ;
 int  nbr,nbr2 ;
 int Trouve ;

 printf("Le nom de fichier Ã  Lire :)") ;
 fgets(NomFichier,sizeof NomFichier,stdin) ;
 DelNewLine(NomFichier) ;
 Fichier = fopen(NomFichier,"r+") ; /* Si le fichier existe, on le
cree sinon on ajoute */
 if ( Fichier == NULL )
    {
     fprintf(stderr,"Echec Ouverture\n") ;
     exit(0) ;
    }
 else
    fprintf(stderr,"Ouverture reussie \n") ;
 printf("Saisie numero :") ;
 fgets(Tampon,sizeof Tampon,stdin ) ;
 Numero = atoi(Tampon) ;
 /* pour montrer les possibilités de l accés direct on commence par
la fin */
 Trouve = 0 ;
 nbr = fseek(Fichier,0,SEEK_END) ;
 /* fseek ne retourne pas la nouvelle position mais ftell */
 nbr = ftell(Fichier) ;
 printf("Fin de Fichier %d\n",nbr) ;

 while ( (nbr != 0) && (!Trouve) )
 {
  /* on recule pour lire le suivant */
  nbr = fseek(Fichier,-sizeof(UnRecord),SEEK_CUR) ;
  nbr = ftell(Fichier) ;
  nbr2 = fread(&UnRecord,sizeof(UnRecord),1,Fichier) ;
  printf("Premier Record  Position %d Bytes Lus %d Numero
%d\n",nbr,nbr2,UnRecord.Numero) ;
  /* On se replace au debut du record */
  nbr = fseek(Fichier,-sizeof(UnRecord),SEEK_CUR) ;
  nbr = ftell(Fichier) ;
  if (UnRecord.Numero == Numero )
     Trouve = 1 ;
 }

 if ( Trouve  )
     AfficheRecord(&UnRecord) ;
 fclose(Fichier) ;
}
```

Trouver un serveur en arrière plan

Un programme serveur tourne en arrière-plan. Pour savoir le pid d'un processus , on supose que son nom est ***ser***, et lui envoyer un signal

```
ps  axu | grep "ser
vanstap   3452  0.0  0.0   1836   496 pts/1    S+   17:52   0:00 ./ser 127.0.0.1 1500
kill -SIGINT 3452
kill -9 3452
```

Les fonctions de la librairie udplib

```
struct ip4 {
     u_char b1 ;
       u_char b2 ;
     u_char b3 ;
     u_char b4 ;
     };

int   Ipv4ToInt(char *s,int *ip) ;
void  Ipv4ToS(int ip, char *s) ;
void    afficher_adresse( struct ip4 *adresse ) ;
int     creer_socket(int,u_long *ai, u_short port,struct sockaddr_in
*pin) ;
int SendDatagram(int desc,void *message,int tm, struct sockaddr_in
*psos ) ;
int ReceiveDatagram(int desc, void *message,int tm, struct
sockaddr_in *psor ) ;
int     generer_masque(int NbrBits ) ;
```

Utiliser la liste chainée

http://nicolasj.developpez.com/articles/listesimple

Directives pour SUN

La librairie fonctionne sur sun à condition de faire les adaptations suivantes.

Il faut editer le fichier ***makefile*** et Modifier la variable LIBS

```
LIBS=-lsocket -lnsl
```

Attention à ne pas oublier –l

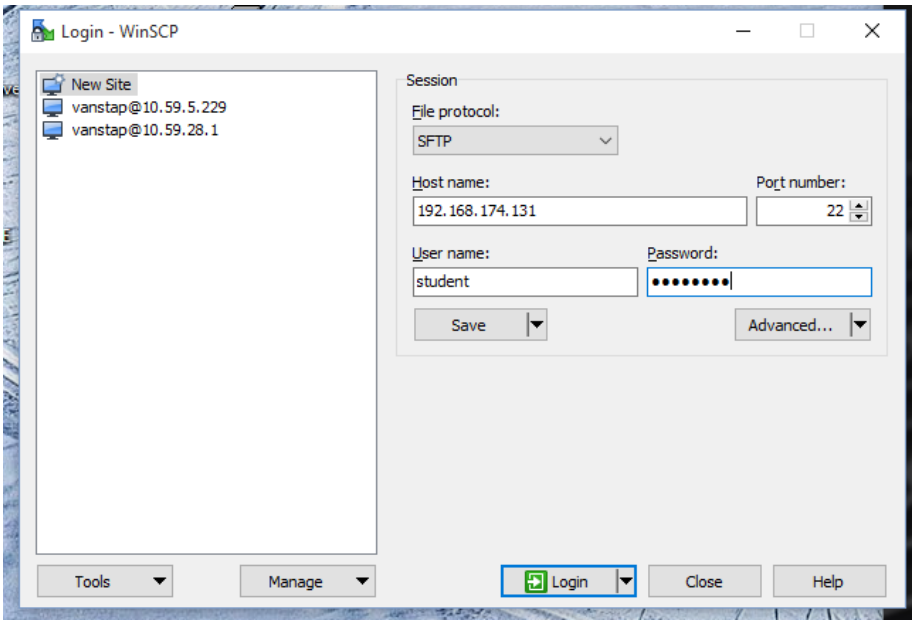
Instructions dont le header change avec sun

Dans le source C pour les instructions suivantes rajouter le header

Instruction	
bzero	<strings.h>
sleep	<unistd.h>

Transfert vers machine virtuelle Sun

Utiliser L'utilitaire WinSCP



Login : student , Passwd : student1 su : root11

Sources

La communication sous Unix : Applications réparties, Jean-Marie Rifflet, EdiScience international.

Unix Network programming, W. Richard Stevens, Prentice Hall Software Séries.

Programmation système en C sous Linux : Signaux;processus,threads, IPC et sockets, Christophe Blaess, Eyrolles

Remerciements

Denys Mercenier pour avoir fournit les directives de compilation sous sun.

Jean-Marc Wagner pour son aide précieuse dans le portage des signaux sous sun.

Daniel Maxime pour les timers intégrés à UDP.