Réseaux 2008-2009

Construire un protocole au – dessus de udp

Notes de cours provisoires.

Liste complète des notes

Chapitre	
0	Terminologie et modèles de référence.
1	Les signaux, câblages et interfaces
2	les connexions point à point, les réseaux locaux.
2T	Les Réseaux locaux de type jeton
3	La couche réseau
4	Les protocoles de transport
4PN	Programmer un protocole
5	Applications et administration réseaux

Toute critique constructive peut être adressée à : herman.vanstapel@skynet.be

Construire un protocole au – dessus de udp

Ces notes sont disponibles gratuitement pour les étudiants.

Vous ne pouvez modifier le contenu sans autorisation de l'auteur.

Structure de ce syllabus

Introduction

Ce syllabus vous invite à élaborer votre propre protocole à l'aide de librairies de base 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.

Structure du document

La programmation illustrée

Ensemble d’exemples qui vous permettront de programmer votre application au laboratoire.

Ecrire des protocoles fiables.

Illustration par des graphiques des problèmes qui peuvent survenir quand on programme des applications.

Les librairies

Explications sur les paramètres des différentes fonctions

La programmation illustrée

La liste des exemples fournis

Quatre exemples de combinaisons de programmes vous seront fournis.
Ces exemples comportent toujours un client, un serveur et un makefile.

Voici les titres des exemples.

1) Un client envoyant des bytes à un serveur
2) Un client envoyant une structure à un serveur
3) Un serveur gérant plusieurs clients
4) Un serveur gérant plusieurs connexions réseaux distinctes (différents ports)
5) Un serveur gérant plusieurs clients et une interaction avec le clavier
6) Un serveur gérant plusieurs clients. Les clients gèrent un timer en cas de non réponse du serveur
7) Un serveur gérant plusieurs clients avec une machine d'état. Les clients gèrent un timer en cas de non réponse du serveur
8) Fermer une connexion
9) Calcul de la taille maximum de paquet envoyé sur réseau

Obtenir les sources des exemples et librairies

Aller dans mon **Centre des ressources**
Aller dans le répertoire librairies

Si vous travaillez sur linux, charger le fichier pour knoppix ou suse. Si vous travaillez pour SUN, prenez le fichier librairies SUN . Attention dans ce cas, des adaptations sont à effecter dans les makefiles et fichier sources des exemples. Pour cela lire le document correspondant en Annexe (qui se trouve aussi sur Ecole Virtuelle).

Après décompression, il suffit de placer ce répertoire dans un session d'utilisateur sur une machine ***unix compaq*** (école ou testdrive) ou encore sur une ***machine linux*** (distribution mandrake ou suze ou ***knoppix***).

Les problèmes avec de transfert ou de compilation

Malheureusement l'utilitaire ftp a la facheuse tendance à convertir les noms de fichier en majuscules. Si cela se produit, il existe la procédure shell ***UPPER*** qui convertit les noms de fichiers de majuscules en minuscules.

Après avoir porté le répertoire d''une machine compaq vers linux ou l'inverse, toujours executer la procédure ***clean*** qui supprimera les executables et les fichiers o. Le fait de compiler un exmeple sous compaq avec des .o créés sous linux entrainera la création d'un fichier non executable.

N'oubliez pas que les noms des machines changent, il faut donc modifer dans le fichier makefile et remplacer le nom par le nom correct.ou mieux par ***127.0.0.1***.

Attention, a partir du moment ou vous travaillez sur la même machine, il est important ***d'utiliser vos propres ports***. Pour les connaître, il faut vous adresser à votre professeur de laboratoire.

Conventions sur la gestion des erreurs

Quand un programme est lancé il dispose d'office des trois fichiers suivants.

Type de canal	Valeur handle (open)	FILE * (fop
Unité d'entrée (par défaut , clavier)	0	***stdin***
Unité de sortie (par défaut , ecran)	1	***stdout***
Unité d'affichage des messages d'erreurs (par défaut , ecran)	2	***stderr***

La convention veut que les messages d'erreurs soient écrits sur le canal d'erreur. Les messages de trace sont aussi envoyés sur ce canal.

Pour faciliter la recherche d'erreur et l'analyse après le fonctionnement du programme, il est préférable de rediriger ces messages vers le canal d'erreur. En sh voici la procédure à suivre.

```
./client1 2>err.log
cat err.log
```

.

1) Un client envoyant des bytes à un serveur

repertoire

aller dans le répertoire ***cphex***.

Objectif

Le client envoie des bytes au serveur qui les affiche.

Exécution du programme

127.0.0.1 est l'ip de la machine sur laquelle vous exécutez le programme. Et si 1100 et 1200 sont les ports qui vous sont attribués.

Le lancement du serveur (fernetre 1)

```
vanstap@linux:~/lib2006/cphex> make
echo "Compilation de serveur"
Compilation de serveur
cc -o ser  ser.c physlib.o tcplib.o
vanstap@linux:~/lib2006/cphex> ./ser
Ceci est le serveur
ser ser port cli port
vanstap@linux:~/lib2006/cphex> ./ser 127.0.0.1 1200 127.0.0.1 1100
Ceci est le serveur
IP du client:127.0.0.1
IP du serveur:127.0.0.1
port 1200
Liaison Cree
bytes:17:Ceci est un test
vanstap@linux:~/lib2006/cphex>
```

Le lancement du client (fenetre 2)
Un client envoyant des bytes à un serveur

```
vanstap@linux:~/lib2006/cphex> ./cli 127.0.0.1 1100 127.0.0.1 1200
IP du client:127.0.0.1
IP du serveur:127.0.0.1
port 1100
Liaison Cree
Envoi de 17 bytes
vanstap@linux:~/lib2006/cphex>
```

POUR Les explications des fonctions voir annexe Les librairies

Le client

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

-----------------------------------------------------------------*/
#include <stdio.h>
#include <string.h>
#include "../physlib/physlib.h"

int main(int argc, char *argv[])
{
 int rc ;
 char *message ;
 struct Physique L ;

 bzero(&L,sizeof(struct Physique)) ;
 if (argc!=5)

{
    printf("cli client portc serveur ports\n") ;
    exit(1) ;
 }
 rc = CreerConnexion(&L,argv[1],atoi(argv[2]),argv[3],atoi(argv[4]))
;
 if ( rc == -1 )
    perror("CreerLiaison:") ;
 else
    fprintf(stderr,"Liaison Cree \n") ;
 message = "Ceci est un test" ;
 rc = VersCouchePhysique(&L,message,strlen(message)+1 ) ;
 if ( rc == -1 )
    perror("VersCouchePhysique") ;
 else
   fprintf(stderr,"Envoi de %d bytes\n",rc ) ;
}
```

Le serveur

```
/*-------------------------------------
  cphex\ser.c
---------------------------------------*/

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

int main(int argc,char *argv[])
{
 int rc ;
 struct Physique L ;
 char message[100] ;
 int tm ;

 bzero(&L,sizeof(struct Physique)) ;
 printf("Ceci est le serveur\n") ;
 if ( argc!=5)
 {
  printf("ser ser port cli port\n") ;
  exit(1) ;
 }
 rc = CreerConnexion(&L,argv[1],atoi(argv[2]),argv[3],atoi(argv[4]))
;
 if ( rc == -1 )
    perror("CreerLiaison:") ;
 else
    printf("Liaison Cree \n") ;
 tm = sizeof(message) ;
 rc = OrigineCouchePhysique(&L,message,&tm ) ;
 if ( rc == -1 )
    perror("OrigineCouchePhysique") ;
 else
   fprintf(stderr,"bytes:%d:%s\n",rc,message ) ;
}
```

Le fichier makefile

```
# cphex\makefile

all:  cli   ser   physlib.o   tcplib.o

physlib.o:  ../physlib/physlib.h    ../physlib/physlib.c
      echo "Compilation de physlib.o"
      cc -c ../physlib/physlib.c

tcplib.o:   ../tcplib/tcplib.h      ../tcplib/tcplib.c
      echo "Compilation de tcplib.o"
      cc -c ../tcplib/tcplib.c

cli:  cli.c physlib.o   tcplib.o
      echo "Compilation de client"
      cc  -o cli  cli.c physlib.o tcplib.o

ser:  ser.c physlib.o   tcplib.o
      echo "Compilation de serveur"
      cc -o ser  ser.c physlib.o tcplib.o
```

2) Un client envoyant une structure à un serveur

repertoire

aller dans le répertoire ***cphexS***.

Objectif

Le client envoie une structure de donnée au serveur qui les affiche.

Exécution du programme

127.0.0.1 est l'ip de la machine sur laquelle vous exécutez le programme. Et si 1100 et 1200 sont les ports qui vous sont attribués.

Le lancement du serveur (fenetre 1)

```
vanstap@linux:~/lib20063/cphexS> ./ser 127.0.0.1 1200 127.0.0.1 1100
Ceci est le serveur
IP du client:127.0.0.1
IP du serveur:127.0.0.1
port 1200
Liaison Cree
bytes:44:1:Ceci est un test
UnData.Valeur= 1
UnData.message= Ceci est un test
```

Le lancement du client (fenetre 2)
Un client envoyant une structure à un serveur

```
vanstap@linux:~/lib20063/cphexS> ./cli 127.0.0.1  1100 127.0.0.1 1200
IP du client:127.0.0.1
IP du serveur:127.0.0.1
port 1100
Liaison Cree
Envoi de 44 bytes
```

La structure de donnée utilisée est dans le fichier ***structure.h.*** Il est vivement conseillé de placer la structure de donnée dans un fichier commun.

```
struct Data
    {
     int Valeur ;
     char message[40] ;
    } ;
```

Le client

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

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

int main(int argc, char *argv[])
{
 int rc ;
 struct Physique L ;
 struct Data UnData ;

 bzero(&L,sizeof(struct Physique)) ;
 if (argc!=5)

{
    printf("cli client portc serveur ports\n") ;
    exit(1) ;
 }
 rc = CreerConnexion(&L,argv[1],atoi(argv[2]),argv[3],atoi(argv[4])) ;
 if ( rc == -1 )
    perror("CreerLiaison:") ;
 else
    fprintf(stderr,"Liaison Cree \n") ;

 UnData.Valeur = 1 ;
 strncpy(UnData.message,"Ceci est un test",sizeof(UnData.message)) ;
 rc = VersCouchePhysique(&L,&UnData,sizeof(struct Data)) ;
 if ( rc == -1 )
    perror("VersCouchePhysique") ;
 else
   fprintf(stderr,"Envoi de %d bytes\n",rc ) ;
}
```

Le serveur

```
/*--------------------------------------------------------------------------
  cphexS\ser.c
--------------------------------------------------------------------------*/

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

int main(int argc,char *argv[])
{
 int rc ;
 struct Physique L ;
 struct Data UnData ;
 int tm ;

 bzero(&L,sizeof(struct Physique)) ;
 printf("Ceci est le serveur\n") ;
 if ( argc!=5)
 {
  printf("ser ser port cli port\n") ;
  exit(1) ;
 }
 rc = CreerConnexion(&L,argv[1],atoi(argv[2]),argv[3],atoi(argv[4])) ;
 if ( rc == -1 )
    perror("CreerLiaison:") ;
 else
    printf("Liaison Cree \n") ;
 tm = sizeof(struct Data) ;
 rc = OrigineCouchePhysique(&L,&UnData,&tm ) ;
 if ( rc == -1 )
    perror("OrigineCouchePhysique") ;
 else
   fprintf(stderr,"bytes:%d:%d:%s\n",rc,UnData.Valeur,UnData.message ) ;

  printf("UnData.Valeur= %d\n", UnData.Valeur) ;
  printf("UnData.message= %s\n",UnData.message) ;
}
```

Le makefile

```
# cphexS\makefile

all:	cli	ser	physlib.o	tcplib.o

physlib.o:	../physlib/physlib.h	../physlib/physlib.c
	echo "Compilation de physlib.o"
	cc -c ../physlib/physlib.c

tcplib.o:../tcplib/tcplib.h  ../tcplib/tcplib.c
	echo "Compilation de tcplib.o"
	cc -c ../tcplib/tcplib.c

cli:	cli.c	physlib.o	tcplib.o	structure.h
	echo "Compilation de client"
	cc  -o cli  cli.c physlib.o tcplib.o

ser:	ser.c	physlib.o	tcplib.o	structure.h
	echo "Compilation de serveur"
	cc -o ser  ser.c physlib.o tcplib.o
```

3) Un serveur gérant plusieurs clients

Répetoire

aller dans le répertoire ***cphexser2***.

Objectif

On lance plusieurs clients qui envoient des bytes au client. Le serveur répond à chaque client individuellement.

Exécution du programme

Lancement du serveur (fenêtre 1)

```
vanstap@linux:~/lib20063/cphexser2> ./ser 127.0.0.1 1300
Ceci est le serveur
IP du client:127.0.0.1
port 1300
Liaison Cree
Event:3
bytes lus:17:Ceci est un test
bytes □ecrits :15:Bonjour Madame
Event:3
bytes lus:17:Ceci est un test
bytes □ecrits :15:Bonjour Madame
Event:-1
Event:-1
```

Lancement du client1 (fenêtre 2)

```
vanstap@linux:~/lib20063/cphexser2> ./cli 127.0.0.1 1100 127.0.0.1 1300
IP du client:127.0.0.1
IP du serveur:127.0.0.1
port 1100
Liaison Cree
Envoi de 17 bytes
Reception de 15 bytes Bonjour Madame
```

Lancement du client2 (fenêtre 3)

```
vanstap@linux:~/lib20063/cphexser2> ./cli 127.0.0.1  1200 127.0.0.1 1300
IP du client:127.0.0.1
IP du serveur:127.0.0.1
port 1200
Liaison Cree
Envoi de 17 bytes
Reception de 15 bytes Bonjour Madame
```

Le client

```
/*-----------------------------------------------------------------
   Vanstapel Herman
   cphexser2\cli.c

-----------------------------------------------------------------*/
#include <stdio.h>
#include "../physlib/physlib.h"

int main(int argc, char *argv[])
{
 int rc ;
 char *message ;
 struct Physique L ;
 char   Buffer[500] ;
 int  tm  ;

 bzero(&L,sizeof(struct Physique)) ;
 if (argc!=5)
 {
    printf("cli client portc serveur ports\n") ;
    exit(1) ;
 }
 rc = CreerConnexion(&L,argv[1],atoi(argv[2]),argv[3],atoi(argv[4]))
;
 if ( rc == -1 )
    perror("CreerLiaison:") ;
 else
    fprintf(stderr,"Liaison Cree\n") ;
 message = "Ceci est un test" ;
 rc = VersCouchePhysique(&L,message,strlen(message)+1 ) ;
 if ( rc == -1 )
    perror("VersCouchePhysique") ;
 else
   fprintf(stderr,"Envoi de %d bytes\n",rc ) ;
 tm = sizeof(Buffer) ;
 rc = OrigineCouchePhysique(&L,Buffer,&tm) ;
 if ( rc == -1 )
    perror("OrigineCouchePhysique") ;
 else
   fprintf(stderr,"Reception de %d bytes %s\n",rc,Buffer ) ;
}
```

Le serveur

```
/*-----------------------------------------------------------------
  cphexser2\ser.c
-----------------------------------------------------------------*/

#include <stdio.h>

#include "../evlib/evlib.h"
#include "../physlib/physlib.h"

int main(int argc,char *argv[])
{
 int rc ;
 struct Physique L ;
 char message[100] ;
 int tm ;
 char *Chaine ;
 static int HLs[100] ;
 static int HEs[100] ;
 int HL,HE ;
 int descphys ;
 static struct gEvenement gEv ;

 bzero(&L,sizeof(struct Physique)) ;

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

 descphys = CreerConnexion(&L,argv[1],atoi(argv[2]),NULL,0) ;
 if ( descphys == -1 )
    perror("CreerLiaison:") ;
 else
    fprintf(stderr,"Liaison Cree\n") ;

 HLs[descphys]=1 ;
 HLs[0] = 1 ;

 rc = PrepareEvenement(&gEv,HLs,HEs,100,300,0 ) ;
 if ( rc== -1 )
    perror("PrepareEvenement:") ;

 while(1)
 {
  rc = AttendreEvenement(&gEv,&HL,&HE) ;
  if ( rc == -1 )
     perror("AttendreEvenement:") ;
  else
     fprintf(stderr,"Event:%d\n",HL) ;
  if ( HL == 0 )
     {
      char Buffer[100] ;
      fgets(Buffer,sizeof Buffer,stdin) ;
      Buffer[strlen(Buffer)]=0 ;
      printf("La touche enfonc est %s \n",Buffer) ;
```

```
     }
  if ( HL == descphys )
  {
   tm = sizeof(message) ;
   rc = OrigineCouchePhysique(&L,message,&tm ) ;
   if ( rc == -1 )
      {
       perror("OrigineCouchePhysique") ;
       return(-1) ;
      }
   else
      fprintf(stderr,"bytes lus:%d:%s\n",rc,message ) ;

   Chaine = "Bonjour Madame" ;
   rc = Repondre(&L,Chaine,strlen(Chaine)+1) ;
   if ( rc == -1 )
      {
       perror("OrigineCouchePhysique") ;
       return(-1) ;
      }
    else
       fprintf(stderr,"bytes □ecrits :%d:%s\n",rc,Chaine ) ;
   }
 }
}
```

Le fichier makefile

```
# Cphexser2\makefile

all:  cli   ser   physlib.o   tcplib.o

physlib.o:  ../physlib/physlib.h    ../physlib/physlib.c
      echo "Compilation de physlib.o"
      cc -c ../physlib/physlib.c

tcplib.o:   ../tcplib/tcplib.h      ../tcplib/tcplib.c
      echo "Compilation de tcplib.o"
      cc -c ../tcplib/tcplib.c

evlib.o:    ../evlib/evlib.c  ../evlib/evlib.h
      echo "Compilation de evlib.c"
      cc -c ../evlib/evlib.c

cli:  cli.c physlib.o   tcplib.o
      echo "Compilation de client"
      cc  -o cli  cli.c physlib.o tcplib.o

ser:  ser.c physlib.o   tcplib.o    evlib.o
      echo "Compilation de serveur"
      cc -o ser  ser.c physlib.o tcplib.o evlib.o
```

4) Un serveur gérant plusieurs connexions réseaux distinctes (différents ports)

Répetoire

aller dans le répertoire ***cphexser2b***.

Objectif

Un lance un serveur qui écoute ***sur deux ports***.
On lance deux clients. Un client envoie un message sur un port, Le serveur ne répond pas immédiatement. Quand le deuxième client envoieune requête, le serveur répond à la fois au premier client et au second.

Exécution du programme

Lancement du serveur (fenêtre 1)

```
vanstap@linux:~/lib20063/cphexser2b> ./ser 127.0.0.1 1300 1400
Ceci est le serveur
IP du client:127.0.0.1
port 1300
Liaison Cree
IP du client:127.0.0.1
port 1400
Liaison Cree
Event:3
bytes:17:Ceci est un test
Je vous repond quand il y'aura reception sur le second port
Event:4
bytes:17:Ceci est un test
Reponse 2 bytes ecrits :17:Bonjour Madame 2
Reponse 1 bytes ecrits :17:Bonjour Madame 1
```

Lancement du client1 (fenêtre 2)

```
vanstap@linux:~/lib20063/cphexser2b> ./cli 127.0.0.1 1100 127.0.0.1 1300
IP du client:127.0.0.1
IP du serveur:127.0.0.1
port 1100
Liaison Cree
Envoi de 17 bytes
Reception de 17 bytes Bonjour Madame 1
```

Lancement du client2 (fenêtre 3)

```
vanstap@linux:~/lib20063/cphexser2b> ./cli 127.0.0.1 1200 127.0.0.1 1400
IP du client:127.0.0.1
IP du serveur:127.0.0.1
port 1200
Liaison Cree
Envoi de 17 bytes
Reception de 17 bytes Bonjour Madame 2
```

[2]*Le client*

```
/*-------------------------------------------------------------
   Vanstapel Herman
   cphexser2b\cli.c

-------------------------------------------------------------*/
#include <stdio.h>
#include "../physlib/physlib.h"

void main(int argc, char *argv[])
{
 int rc ;
 char *message ;
 struct Physique L ;
 char   Buffer[500] ;
 int  tm  ;

 bzero(&L,sizeof(struct Physique)) ;
 if (argc!=5)
 {
    printf("cli client portc serveur ports\n") ;
    exit(1) ;
 }
 rc = CreerConnexion(&L,argv[1],atoi(argv[2]),argv[3],atoi(argv[4]))
;
 if ( rc == -1 )
    perror("CreerLiaison:") ;
 else
    fprintf(stderr,"Liaison Cree \n") ;
 message = "Ceci est un test" ;
 rc = VersCouchePhysique(&L,message,strlen(message)+1 ) ;
 if ( rc == -1 )
    perror("VersCouchePhysique") ;
 else
   fprintf(stderr,"Envoi de %d bytes\n",rc ) ;
 tm = sizeof(Buffer) ;
 rc = OrigineCouchePhysique(&L,Buffer,&tm) ;
 if ( rc == -1 )
    perror("OrigineCouchePhysique") ;
 else
   fprintf(stderr,"Reception de %d bytes %s\n",rc,Buffer ) ;
}
```

Le serveur

```
/*-------------------------------------------------------------
  cphexser2b\ser.c
-------------------------------------------------------------*/

#include <stdio.h>

#include "../evlib/evlib.h"
#include "../physlib/physlib.h"

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

```
struct Physique L1,L2 ;
char message[100] ;
int tm ;
char *Chaine ;
static int HLs[100] ;
static int HEs[100] ;
int HL,HE ;
int descphys,descphys2 ;
static struct gEvenement gEv ;

bzero(&L1,sizeof(struct Physique)) ;
bzero(&L2,sizeof(struct Physique)) ;

printf("Ceci est le serveur\n") ;
if ( argc!=4)
{
 printf("ser ipser port port \n") ;
 exit(1) ;
}

descphys = CreerConnexion(&L1,argv[1],atoi(argv[2]),NULL,0) ;
if ( descphys == -1 )
   perror("CreerLiaison:1") ;
else
   fprintf(stderr,"Liaison Cree \n") ;

descphys2 = CreerConnexion(&L2,argv[1],atoi(argv[3]),NULL,0) ;
if ( descphys == -1 )
   perror("CreerLiaison:2") ;
else
   fprintf(stderr,"Liaison Cree \n") ;

HLs[descphys]  = 1 ;
HLs[descphys2] = 1 ;
HLs[0] = 1 ;

rc = PrepareEvenement(&gEv,HLs,HEs,100,300,0 ) ;
if ( rc== -1 )
   perror("PrepareEvenement:") ;

while(1)
{
 rc = AttendreEvenement(&gEv,&HL,&HE) ;
 if ( rc == -1 )
    perror("AttendreEvenement:") ;
 else
    fprintf(stderr,"Event:%d\n",HL) ;
 if ( HL == 0 )
    {
     char Buffer[100] ;
     fgets(Buffer,sizeof Buffer,stdin) ;
     Buffer[strlen(Buffer)]=0 ;
     fprintf(stderr,"La touche enfonc est %s \n",Buffer) ;
    }
 if ( HL == descphys )
 {
  tm = sizeof(message) ;
  rc = OrigineCouchePhysique(&L1,message,&tm ) ;
  if ( rc == -1 )
     {
      perror("OrigineCouchePhysique") ;
```

```
        return(-1) ;
       }
    else
       fprintf(stderr,"bytes:%d:%s\n",rc,message ) ;
       printf("Je vous repond quand il y'aura reception sur le second
port\n") ;

    }
   if ( HL == descphys2 )
   {
    tm = sizeof(message) ;
    rc = OrigineCouchePhysique(&L2,message,&tm ) ;
    if ( rc == -1 )
       {
        perror("OrigineCouchePhysique") ;
        return(-1) ;
       }
    else
       fprintf(stderr,"bytes:%d:%s\n",rc,message ) ;

    Chaine = "Bonjour Madame 2" ;
    rc = Repondre(&L2,Chaine,strlen(Chaine)+1) ;
    if ( rc == -1 )
       {
        perror("OrigineCouchePhysique") ;
        return(-1) ;
       }
     else
        fprintf(stderr,"Reponse 2 bytes ecrits :%d:%s\n",rc,Chaine ) ;

    Chaine = "Bonjour Madame 1" ;
    rc = Repondre(&L1,Chaine,strlen(Chaine)+1) ;
    if ( rc == -1 )
       {
        perror("OrigineCouchePhysique") ;
        return(-1) ;
       }
     else
        fprintf(stderr,"Reponse 1 bytes ecrits :%d:%s\n",rc,Chaine ) ;
    }

 }
}
```

Le makefile

```
# cphexser2b\makefile

all:  cli   ser   physlib.o   tcplib.o

physlib.o:  ../physlib/physlib.h    ../physlib/physlib.c
      echo "Compilation de physlib.o"
      cc -c ../physlib/physlib.c

tcplib.o:   ../tcplib/tcplib.h      ../tcplib/tcplib.c
      echo "Compilation de tcplib.o"
      cc -c ../tcplib/tcplib.c

evlib.o:    ../evlib/evlib.c  ../evlib/evlib.h
      echo "Compilation de evlib.c"
      cc -c ../evlib/evlib.c

cli:  cli.c physlib.o   tcplib.o
      echo "Compilation de client"
      cc  -o cli  cli.c physlib.o tcplib.o

ser:  ser.c physlib.o   tcplib.o    evlib.o
      echo "Compilation de serveur"
      cc -o ser  ser.c physlib.o tcplib.o evlib.o
```

5) Un serveur gérant plusieurs clients et une interaction avec le clavier

Répetoire

aller dans le répertoire ***cphexser3***.

Objectif

On lance plusieurs clients qui envoient des bytes au client. Le serveur répond à chaque client individuellement. Il gère en plus le clavier.

La gestion du clavier est aussi intégrée au client pour des améliorations futures.

Exécution du programme

Lancement du serveur (fenêtre 1)

```
vanstap@vanstap:~/Documents/lib2008G/cphexser3> ./ser 127.0.0.1 1300
Ceci est le serveur version 2
IP du client:127.0.0.1
port 1300
Liaison Cree
Event:-1
Event:3
bytes:17:Ceci est un test
bytes ecrits :15:Bonjour Madame
j
Event:0
La touche enfoncee est j
```

Lancement du client1 (fenêtre 2)

```
anstap@vanstap:~/Documents/lib2008G/cphexser3> ./cli 127.0.0.1 1200 127.0.0.1 1300
IP du client:127.0.0.1
IP du serveur:127.0.0.1
port 1200
Liaison Cree
 Pressez la touche 1 (enter ) pour envoyer msg
1
Evenement 0
Envoi de 17 bytes
 Pressez la touche 1 (enter ) pour envoyer msg
Evenement 3
Reception de 15 bytes Bonjour Madame
vanstap@vanstap:~/Documents/lib2008G/cphexser3>
```

Le client

```
/*-----------------------------------------------------------------
   Vanstapel Herman
   cphexser3\cli.c

-----------------------------------------------------------------*/
#include <stdio.h>

#include "../evlib/evlib.h"
#include "../physlib/physlib.h"
#include "../evtmlib/evtmlib.h"
#include <string.h>

static struct gEvenement gEv ;

int main(int argc, char *argv[])
{
 int rc ;
 char *message ;
 struct Physique L ;
 struct gEvenement gEv ;

 char   Buffer[500] ;
 int  tm  ;
 static int HLs[100] ;
 static int HEs[100] ;
 int evenement ;
 int HE ;
 int Connexion ;

 bzero(&L,sizeof( struct Physique )) ;
 bzero(&gEv,sizeof( struct gEvenement )) ;

 if (argc!=5)
 {
    printf("cli client portc serveur ports\n") ;
    exit(1) ;
 }
 Connexion =
CreerConnexion(&L,argv[1],atoi(argv[2]),argv[3],atoi(argv[4])) ;
 if ( Connexion == -1 )
    perror("CreerLiaison:") ;
 else
    fprintf(stderr,"Liaison Cree\n") ;

 HLs[Connexion]=1 ;
 HLs[0] = 1 ;

 rc = PrepareEvenement(&gEv,HLs,HEs,100,300,0) ;
 if ( rc==-1)
    perror("PrepareEvenement") ;



 while(1)
 {
  printf (" Pressez la touche 1 (enter ) pour envoyer msg \n" ) ;
```

```
   rc = AttendreEvenement(&gEv,&evenement,&HE) ;
   if ( rc == 0 )
      printf("timeout !!\n") ;
   if ( rc == -1 )
      printf("Erreur d'attendreEvenement\n") ;
   printf("Evenement %d \n",evenement) ;
   if ( evenement == 0 )
      {
       char Buffer[100] ;
       fgets(Buffer,sizeof Buffer,stdin) ;
       if ( Buffer[0] == '1' )
          {
           message = "Ceci est un test" ;
           rc = VersCouchePhysique(&L,message,strlen(message)+1 ) ;
          if ( rc == -1 )
              perror("VersCouchePhysique") ;
          else
             fprintf(stderr,"Envoi de %d bytes\n",rc ) ;
          }
       sleep(2) ;
      }
   if ( evenement == Connexion  )
   {
   tm = sizeof(Buffer ) ;
   rc = OrigineCouchePhysique(&L,Buffer,&tm) ;
   if ( rc == -1 )
     perror("OrigineCouchePhysique") ;
   else
   {
     fprintf(stderr,"Reception de %d bytes %s\n",rc,Buffer ) ;
     exit(0) ;
   }
   }
  }
}
```

Le serveur

```
/*-----------------------------------------------------------------
  cphexser3\ser.c
-----------------------------------------------------------------*/

#include <stdio.h>

#include "../evlib/evlib.h"
#include "../physlib/physlib.h"
#include <string.h>

int main(int argc,char *argv[])
{
 int rc ;
 char message[100] ;
 int tm ;
 char *Chaine ;
 static int HLs[100] ;
 static int HEs[100] ;
 int HL,HE ;
 int descphys ;
 struct Physique L ;
 struct gEvenement gEv ;

 bzero(&L,sizeof( struct Physique )) ;
 bzero(&gEv,sizeof( struct gEvenement)) ;
 printf("Ceci est le serveur\n") ;
 if ( argc!=3)
 {
  printf("ser ser port \n") ;
  exit(1) ;
 }

 descphys = CreerConnexion(&L,argv[1],atoi(argv[2]),NULL,0) ;
 if ( descphys == -1 )
    perror("CreerLiaison:") ;
 else
    fprintf(stderr,"Liaison Cree\n") ;

 HLs[descphys]=1 ;
 HLs[0] = 1 ;

 rc = PrepareEvenement(&gEv,HLs,HEs,100,300,0 ) ;
 if ( rc== -1 )
    perror("PrepareEvenement:") ;

 while(1)
 {
  rc = AttendreEvenement(&gEv,&HL,&HE) ;
  if ( rc == -1 )
     perror("AttendreEvenement:") ;
  else
     fprintf(stderr,"Event:%d\n",HL) ;
  if ( HL == 0 )
     {
      char Buffer[100] ;
      fgets(Buffer,sizeof Buffer,stdin) ;
      Buffer[strlen(Buffer)]=0 ;
      printf("La touche enfoncee est %s \n",Buffer) ;
```

```
      sleep(10) ;
     }
  if ( HL == descphys )
  {
   tm = sizeof(message) ;
   rc = OrigineCouchePhysique(&L,message,&tm ) ;
   if ( rc == -1 )
      {
       perror("OrigineCouchePhysique") ;
       return(-1) ;
      }
   else
      fprintf(stderr,"bytes:%d:%s\n",rc,message ) ;

   Chaine = "Bonjour Madame" ;
   rc = Repondre(&L,Chaine,strlen(Chaine)+1) ;
   if ( rc == -1 )
      {
       perror("OrigineCouchePhysique") ;
       return(-1) ;
      }
    else
       fprintf(stderr,"bytes ecrits :%d:%s\n",rc,Chaine ) ;
   }
 }
}
```

Le fichier makefile

```
# cphexser3\makefile

all:  cli   ser   physlib.o   tcplib.o

physlib.o:  ../physlib/physlib.h    ../physlib/physlib.c
      echo "Compilation de physlib.o"
      cc -c ../physlib/physlib.c

tcplib.o:   ../tcplib/tcplib.h      ../tcplib/tcplib.c
      echo "Compilation de tcplib.o"
      cc -c ../tcplib/tcplib.c

evlib.o:    ../evlib/evlib.c  ../evlib/evlib.h
      echo "Compilation de evlib.c"
      cc -c ../evlib/evlib.c

timerlib.o: ../timerlib/timerlib.c  ../timerlib/timerlib.h
      echo "Compilation de timerlib.o"
      cc -c ../timerlib/timerlib.c

evtmlib.o:  ../evtmlib/evtmlib.c    ../evtmlib/evtmlib.h
      ../timerlib/timerlib.o
      echo "Compilation de evtmlib.o"
      cc -c ../evtmlib/evtmlib.c

cli:  cli.c physlib.o   tcplib.o    timerlib.o  evlib.o
      evtmlib.o
      echo "Compilation de client"
      cc  -o cli  cli.c physlib.o tcplib.o timerlib.o evlib.o
evtmlib.o

ser:  ser.c physlib.o   tcplib.o    evlib.o
      echo "Compilation de serveur"
      cc -o ser  ser.c physlib.o tcplib.o evlib.o
```

L'implémentation des timers

La première idée qui vient à l'esprit, quand on souhaite implémenter un timer, est d'utiliser la fonction ***alarm***. Cette fonction reçoit comme paramètre un certain nombre de secondes et après expiration déclenche un signal ***SIGALARM***.
Malheureusement cette fonction ne gère qu'une alarme et faut pouvoir gérer plusieurs timers !. Autre problème important , la fonction ne travaille qu'en secondes. Or une seconde dans le domaine des réseaux, c'est une éternité.
L'idée est de consulter le temps courant qui est fournit par la fonction ***times*** qui retourne le nombre de clockticks écoulés depuis le 1 janvier 1970. Le nombre de clockticks par seconde varie selon les systèmes. L'appel suivant donne cette information :

sysconf(_SC_CLK_TCK) ;

Le Unix de Compaq(Digital) retourne la valeur 60.

La fonction ***StartTimer*** reçoit comme paramètre un numéro d'alarme et le délais au bout duquel l'alarme doit se déclencher est exprimé en millisecondes. On consulte le temps courant via la fonction times et on y ajoute le délai convertit en clockticks via la fonction ***millitoclktck.***

Cette valeur est stockée ensuite avec le numéro de timer associé dans une liste linéaire de type ***struct ListeTimer***. Le timer devant se déclencher en premier, se trouve en premier dans la liste.

On remarquera que la fonction ***select*** qui et utilisée pour implémenter ***AttendreEvenenement***, peut générer un timeout qui s'exprime en micro-secondes (***10 –6*** sec). Quand ce timeout se déclenche la fonction select retourne la valeur zéro. On va en profiter pour consulter la liste des timers.
On consulte ensuite la liste des timers et si la première valeur excède le temps courant. On considère que le timer s'est déclenché et on retire ce timer de la liste. C'est la fonction ***TestTimer*** qui réalise cette opération.

La fonction ***StopTimer*** permet d'arrêter un timer avant expiration.

Pour faciliter la gestion des timers, la fonction AttendreEvenement a reçu une variante qui intègre la modification des timers. C'est la fonction ***AttendreEvenementtm***. Elle génère un événement TIMEOUT via le paramètre ***HL***.

TIMEOUT + 0 correspond au timer 0.

TIMEOUT + 1 correspond au timer 1.

Et ainsi de suite

6) Un serveur gérant plusieurs clients. Les clients gérent un timer en cas de non réponse du serveur

Répetoire

aller dans le répertoire ***cphexser4***.

Objectif

On lance plusieurs clients qui envoient des bytes au client. Le serveur répond à chaque client individuellement. Il gère en plus le clavier. Le serveur affiche aussi ***port*** et l'***IP*** du client.

Le client déclenche un timer qui expire en cas de non-réponse du serveur

Exécution du programme

Lancement du serveur (fenêtre 1)

```
vanstap@vanstap:~/Documents/lib2008G/cphexser4> ./ser 127.0.0.1 1300
Ceci est le serveur
IP du client:127.0.0.1
port 1300
Liaison Cree
Event:3
bytes:17:Ceci est un test
 IP client 127.0.0.1 Port client 1200
bytes ecrits :15:Bonjour Madame
( CTRL C )
vanstap@vanstap:~/Documents/lib2008G/cphexser4>
```

On notera que l'on fait ctrl C sur le serveur pour déclencher les timers au niveau du client.

Lancement du client (fenêtre 2)

```
anstap@vanstap:~/Documents/lib2008G/cphexser4> ./cli 127.0.0.1 1200 127.0.0.1 1300
IP du client:127.0.0.1
IP du serveur:127.0.0.1
port 1200
Liaison Cree
 Pressez la touche 1 (enter ) pour envoyer msg
1
Envoi de 17 bytes
( 1 :Cl:1718984356),.
 Pressez la touche 1 (enter ) pour envoyer msg
Reception de 15 bytes Bonjour Madame



vanstap@vanstap:~/Documents/lib2008G/cphexser4> ./cli 127.0.0.1 1200 127.0.0.1 1300
IP du client:127.0.0.1
IP du serveur:127.0.0.1
port 1200
Liaison Cree
 Pressez la touche 1 (enter ) pour envoyer msg
1
Envoi de 17 bytes
( 1 :Cl:1718987150),.
 Pressez la touche 1 (enter ) pour envoyer msg
TIMEOUT !!!!!
TIMEOUT:Envoi de 17 bytes
 Pressez la touche 1 (enter ) pour envoyer msg
```

Pour la première exécution du programme , le serveur est actif.

(1 :Cl:1718984356),. Est l'affichage de la liste des timers.

Pour la seconde exécution, le serveur n'est pas actif et le timer se déclenche.

Client

```
/*-----------------------------------------------------------------
   Vanstapel Herman
   cphexser4\cli.c

-----------------------------------------------------------------*/
#include <stdio.h>

#include "../evlib/evlib.h"
#include "../physlib/physlib.h"
#include "../evtmlib/evtmlib.h"


void main(int argc, char *argv[])
{
 int rc ;
 char *message ;
 char   Buffer[500] ;
 int  tm  ;
 int HLs[100] ;
 int HEs[100] ;
 struct ListeTimer *plt=NULL ;
 int evenement ;
 int HE ;
 int Connexion ;
 struct Physique L ;
 struct gEvenement gEv ;

 bzero(&L,sizeof(struct Physique))  ;
 bzero(&gEv,sizeof(struct gEvenement)) ;

 bzero(HLs,sizeof(HLs) );
 bzero(HEs,sizeof(HEs) );

 if (argc!=5)
 {
    printf("cli client portc serveur ports\n") ;
    exit(1) ;
 }
 Connexion =
CreerConnexion(&L,argv[1],atoi(argv[2]),argv[3],atoi(argv[4])) ;
 if ( rc == -1 )
    perror("CreerLiaison:") ;
 else
    fprintf(stderr,"Liaison Cree\n") ;

 HLs[Connexion]=1 ;
 HLs[0] = 1 ;

 rc = PrepareEvenement(&gEv,HLs,HEs,100,0,200000) ;
 if ( rc==-1)
    perror("PrepareEvenement") ;

 while(1)
 {
  printf (" Pressez la touche 1 (enter ) pour envoyer msg \n" ) ;
  rc = AttendreEvenementtm(&gEv,&evenement,&HE,&plt) ;
  if ( rc == -1 )
     fprintf(stderr,"Erreur d'attendreEvenement\n") ;
```

```
  if ( evenement == 0 )
     {
      char Buffer[100] ;
      fgets(Buffer,sizeof Buffer,stdin) ;
      if ( Buffer[0] == '1' )
         {
          message = "Ceci est un test" ;
          rc = VersCouchePhysique(&L,message,strlen(message)+1 ) ;
         if ( rc == -1 )
             perror("VersCouchePhysique") ;
         else
           {
            fprintf(stderr,"Envoi de %d bytes\n",rc ) ;
            plt = StopTimer(1,plt) ;
            plt = StartTimer(1,4000,plt) ;
            AfficheTimer(plt) ;
           }
         }
      sleep(2) ;
     }
  if ( evenement == Connexion  )
  {
  plt = StopTimer(1,plt ) ;
  tm =  sizeof(Buffer) ;
  rc = OrigineCouchePhysique(&L,Buffer,&tm) ;
  if ( rc == -1 )
    perror("OrigineCouchePhysique") ;
  else
  {
    printf("Reception de %d bytes %s\n",rc,Buffer ) ;
    exit(0) ;
  }
  }
  if ( evenement >= TIMEOUT )
  {
   printf("TIMEOUT !!!!!  \n") ;
   sleep(2) ;
   /*   Reecrire la trame        */

   message = "Ceci est un test" ;
   rc = VersCouchePhysique(&L,message,strlen(message)+1 ) ;
   if ( rc == -1 )
      perror("VersCouchePhysique") ;
   else
      fprintf(stderr,"TIMEOUT:Envoi de %d bytes\n",rc ) ;

   plt = StopTimer (1,plt) ;
   plt = StartTimer(1,4000,plt) ;
  }
 }
}
```

Serveur

```
/*----------------------------------------------------------------
  cphexser4\ser.c
----------------------------------------------------------------*/

#include <stdio.h>
#include "../evlib/evlib.h"
#include "../physlib/physlib.h"

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

 char message[100] ;
 int tm ;
 char *Chaine ;
 int HL,HE ;
 int descphys ;

  int HLs[100] ;
  int HEs[100] ;
  struct Physique L ;
  struct gEvenement gEv ;
  int PortD ;
  int IPD ;
  char ips[17] ;

  bzero(HLs,sizeof HLs ) ;
  bzero(HEs,sizeof HEs ) ;
  bzero(&L, sizeof ( struct Physique))  ; /* pour eviter un bug
curieux !!!!!!!*/
  bzero(&gEv,sizeof(struct gEvenement )) ;

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

 descphys = CreerConnexion(&L,argv[1],atoi(argv[2]),NULL,0) ;
 if ( descphys == -1 )
    perror("CreerLiaison:") ;
 else
    fprintf(stderr,"Liaison Cree\n") ;

 HLs[descphys]=1 ;
 HLs[0] = 1 ;

 rc = PrepareEvenement(&gEv,HLs,HEs,100,300,0 ) ;
 if ( rc== -1 )
    perror("PrepareEvenement:") ;

 while(1)
 {
  rc = AttendreEvenement(&gEv,&HL,&HE) ;
  if ( rc == -1 )
```

```
     perror("AttendreEvenement:") ;
  else
     fprintf(stderr,"Event:%d\n",HL) ;
  if ( HL == 0 )
     {
      char Buffer[100] ;
      fgets(Buffer,sizeof Buffer,stdin) ;
      Buffer[strlen(Buffer)]=0 ;
      fprintf(stderr,"La touche enfoncee est %s \n",Buffer) ;
      sleep(10) ;
     }
  if ( HL == descphys )
  {
   tm = sizeof(message) ;
   rc = OrigineCouchePhysique(&L,message,&tm ) ;
   if ( rc == -1 )
      {
       perror("OrigineCouchePhysique") ;
       return(-1) ;
      }
   else
      fprintf(stderr,"bytes:%d:%s\n",rc,message ) ;

   IPD =  IPDistante(&L)  ;
   PortD= PortDistant(&L) ;
   Ipv4ToS(IPD,ips)       ;
   printf(" IP client %s Port client %d \n",ips,PortD) ;

   Chaine = "Bonjour Madame" ;
   rc = Repondre(&L,Chaine,strlen(Chaine)+1) ;
   if ( rc == -1 )
      {
       perror("OrigineCouchePhysique") ;
       return(-1) ;
      }
    else
       fprintf(stderr,"bytes ecrits :%d:%s\n",rc,Chaine ) ;
   }
 }
}
```

Makefile

```
# cphexser4\makefile

all:  cli   ser   physlib.o   tcplib.o

physlib.o:  ../physlib/physlib.h    ../physlib/physlib.c
      echo "Compilation de physlib.o"
      cc -c ../physlib/physlib.c

tcplib.o:   ../tcplib/tcplib.h      ../tcplib/tcplib.c
      echo "Compilation de tcplib.o"
      cc -c ../tcplib/tcplib.c

evlib.o:    ../evlib/evlib.c  ../evlib/evlib.h
      echo "Compilation de evlib.c"
      cc -c ../evlib/evlib.c

timerlib.o: ../timerlib/timerlib.c  ../timerlib/timerlib.h
      echo "Compilation de timerlib.o"
      cc -c ../timerlib/timerlib.c

evtmlib.o:  ../evtmlib/evtmlib.c    ../evtmlib/evtmlib.h
      ../timerlib/timerlib.o
      echo "Compilation de evtmlib.o"
      cc -c ../evtmlib/evtmlib.c

cli:  cli.c physlib.o   tcplib.o    timerlib.o  evlib.o
      evtmlib.o
      echo "Compilation de client"
      cc  -o cli  cli.c physlib.o tcplib.o timerlib.o evlib.o
evtmlib.o

ser:  ser.c physlib.o   tcplib.o    evlib.o
      echo "Compilation de serveur"
      cc -o ser  ser.c physlib.o tcplib.o evlib.o
```

7) Un serveur gérant plusieurs clients avec une machine d'état. Les clients gérent un timer en cas de non réponse du serveur

Répetoire

Aller dans le répertoire ***cphexser3me***.

Objectif

On lance plusieurs clients qui envoient des bytes au client. Le serveur répond à chaque client individuellement. Il gère en plus le clavier.

Le client déclenche un timer qui expire en cas de non-réponse du serveur

La nouvauté est que le client gère une machine d'état. Cela permet au client de traiter les cas ou le client reçoit des messages alors qu'il n en n'a pas encore envoyé.

Le client et serveur peuvent afficher l'ip et le port correspondant en utilisant les routines correspondantes. (*`IPDistante`* `et` *`PortDistant`*)

Exécution du programme

La structure de donnee

```
struct Message
{
  int Entier1 ;
  int Entier2 ;
  int Resultat ;
  char Commentaire[200] ;
};
```

Lancement du serveur (fenêtre 1)

```
vanstap@linux:~/lib20063/cphexser4me> ./ser 127.0.0.1 1300
Ceci est le serveur
IP du client:127.0.0.1
port 1300
Liaison Cree
Event:3
bytes:212:12 + 22 : AAAAAAAAAAAAAAAAAAAA

IP distante 127.0.0.1 Port Distant 1400
bytes ecrits :212
```

Lancement du client (fenêtre 2)

```
vanstap@linux:~/lib20063/cphexser4me> ./cli 127.0.0.1 1400 127.0.0.1 1300
IP du client:127.0.0.1
IP du serveur:127.0.0.1
port 1400
Liaison Cree
1 Calcul d'une somme
----------------------
1                                   < Tapez 1 puis enter
Touche pressee !!!!!
Entrez entier1:12
Entrez entier2:22
Entrez un commentaire:AAAAAAAAAAAAAAAAAAAA
Envoi de 212 bytes
( 1 :Cl:1718573192),.
Reception de 212 bytes
                    de 127.0.0.1 : 1300
Le resultat de 12 + 22 est 34
```

Le Client

```
/*-------------------------------------------------------------------
   Vanstapel Herman 2004
   Cpheser4me\cli.c

-------------------------------------------------------------------*/
#include <stdio.h>

#include "../evlib/evlib.h"
#include "../reslib/reslib.h"
#include "../physlib/physlib.h"
#include "../evtmlib/evtmlib.h"
#include "message.h"


struct Global {
 struct Physique *L ;
 struct ListeTimer *PLT ;
 struct Message Msg ;
 int Etat ;
};

#define RIEN  1
#define ENVOI 2

int Send(struct Global *pG,void* Message,int TM,int NumTimer,int Delais)
{
 int rc ;

 rc = VersCouchePhysique(pG->L,Message,TM) ;
 if ( rc == -1 )
    {
     perror("VersCouchePhysique") ;
     return(rc ) ;
    }
 else
   printf("Envoi de %d bytes\n",rc ) ;

 pG->PLT = StopTimer(NumTimer,pG->PLT) ;
 pG->PLT = StartTimer(NumTimer,Delais,pG->PLT) ;
 AfficheTimer(pG->PLT) ;
 return(rc) ;
}

int Receive(struct Global *pG,void* Message,int *TM)
{
 int rc ;

  pG->PLT = StopTimer(1,pG->PLT ) ;
  rc = OrigineCouchePhysique(pG->L,Message,TM) ;
  if ( rc == -1 )
     perror("OrigineCouchePhysique") ;
  else
 {
     int IPR,PORTD   ;
     char Buffer[30] ;
     IPR   = IPDistante (pG->L) ;
     PORTD = PortDistant(pG->L) ;
     Ipv4ToS(IPR, Buffer ) ;
```

```
     printf("Reception de %d bytes %s de %s : %d\n",rc,Message,Buffer,PORTD) ;
 }
 return(rc) ;
}

void AfficheMenu()
{
 printf("1 Calcul d'une somme  \n") ;
 printf("----------------------\n") ;
}

int SaiSieMessage(struct Message *msg)
{
 char Buffer[100] ;
 printf("Entrez entier1:") ;
 fgets(Buffer,sizeof Buffer,stdin) ;
 Buffer[strlen(Buffer)] = 0  ;
 msg->Entier1 = atoi(Buffer) ;
 printf("Entrez entier2:") ;
 fgets(Buffer,sizeof Buffer,stdin) ;
 Buffer[strlen(Buffer)] = 0 ;
 msg->Entier2 = atoi(Buffer) ;
 printf("Entrez un commentaire:") ;
 fgets(msg->Commentaire,sizeof (msg->Commentaire),stdin) ;
}

int AfficheMessage(struct Message *msg)
{
 printf("Le resultat de %d + %d est %d \n",msg->Entier1,msg->Entier2,msg->Resultat ) ;
}

void TraitementTouche(char touche,char*Message,struct Global *pG )
{
 int rc ;
 if ( touche == '1')
    switch(pG->Etat)
    {
    case RIEN :
         SaiSieMessage(&(pG->Msg)) ;
         rc =Send(pG,&(pG->Msg),sizeof(struct Message),1,4000)  ;
              pG->Etat = ENVOI ;

              break ;
    case ENVOI :
        printf("transaction enn cours \n") ;
            break ;
    default:
        printf("Etat inconnu :\n") ;
            exit(0) ;
            break ;
    }
 else
    AfficheMenu() ;
};


void TraitementReception(struct Global *pG )
{
 int rc ;
 struct Message UnMsg ;
```

```
 int       tm  ;

 tm = sizeof(struct Message) ;
 switch (pG->Etat )
 {
  int Entier1 ;
  int Entier2 ;
  int Resultat ;
  char Commentaire[200] ;

  case ENVOI:
     rc = Receive(pG,&UnMsg,&tm) ;
     AfficheMessage(&UnMsg) ;
     pG->Etat = RIEN ;
     break ;
  case RIEN:
     printf("Je n'ai rien demande\n");
     /* Purger neanmoins le Buffer */
     rc = Receive(pG,&UnMsg,&tm) ;
     break ;
  default:
      printf("Etat inconnu :\n") ;
      exit(0) ;
      break ;
 }
}


int main(int argc, char *argv[])
{
 struct Physique Phys  ;
 struct gEvenement gEv ;
 int HLs[100] ;
 int HEs[100] ;
 int evenement ;
 int Connexion ;
 int rc ;
 char   Buffunctionfer[500] ;
 int       tm  ;
 int HE ;
 struct Global G;
 char *message ;

 /*Initialisation des structures globales */

 bzero(&Phys, sizeof ( struct Physique )) ;
 bzero(&gEv,  sizeof ( struct gEvenement )) ;
 bzero(HLs, sizeof HLs ) ;
 bzero(HEs, sizeof HEs ) ;

 G.L    = &Phys ;
 G.PLT  = NULL ;
 G.Etat = RIEN ;

 if (argc!=5)
 {
    printf("cli client portc serveur ports\n") ;
    exit(1) ;
 }
 Connexion = CreerConnexion(G.L,argv[1],atoi(argv[2]),argv[3],atoi(argv[4])) ;
```

```
 if ( rc == -1 )
    perror("CreerLiaison:") ;
 else
    printf("Liaison Cree \n") ;

 HLs[Connexion]=1 ;
 HLs[0] = 1 ;

 rc = PrepareEvenement(&gEv,HLs,HEs,100,0,200000) ;
 if ( rc==-1)
    perror("PrepareEvenement") ;


 message = "Ceci est un test" ;

 AffricheMenu() ;
 while(1)
 {
  rc = AttendreEvenementtm(&gEv,&evenement,&HE,&(G.PLT)) ;
  if ( rc == -1 )
     printf("Erreur d'attendreEvenement\n") ;
  if ( evenement == 0 )
     {
      char Key ;
      char Buffer[100] ;
      fgets(Buffer,sizeof Buffer,stdin) ;
      printf("Touche pressee !!!!!\n")  ;

      Key = Buffer[0] ;
      TraitementTouche(Key,message,&G ) ;
      sleep(2) ;
     }
  if ( evenement == Connexion  )
      TraitementReception(&G);
  if ( evenement >= TIMEOUT )
  {
     rc =Send(&G,&G.Msg,sizeof(struct Message),1,4000) ;
     G.Etat = ENVOI ;
  }
 }
}
```

Le serveur

```
/*-------------------------------------------------------------
  Herman Vanstapel 2004
   Cpheser4me\ser.c
--------------------------------------------------------------*/

#include <stdio.h>

#include "../evlib/evlib.h"
#include "../physlib/physlib.h"
#include "message.h"

/*
struct Message
{
  int Entier1 ;
```

```
  int Entier2 ;
  int Resultat ;
  char Commentaire[200] ;
};
*/

int main(int argc,char *argv[])
{
 int rc ;
 struct Physique L ; /* pour eviter un bug curieux !!!!!!!*/
 int tm ;
 int HLs[100] ;
 int HEs[100] ;
 char Buffer[50] ;
 int HL,HE ;
 int descphys ;
 static struct gEvenement gEv ;
 int IPR,PORTD ;


 bzero(&L,sizeof (struct Physique)) ;
 bzero(HLs, sizeof(HLs )) ;
 bzero(HEs, sizeof(HEs)) ;

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

 descphys = CreerConnexion(&L,argv[1],atoi(argv[2]),NULL,0) ;
 if ( descphys == -1 )
    perror("CreerLiaison:") ;
 else
    printf("Liaison Cree \n") ;

 HLs[descphys]=1 ;
 HLs[0] = 1 ;

 rc = PrepareEvenement(&gEv,HLs,HEs,100,300,0 ) ;
 if ( rc== -1 )
    perror("PrepareEvenement:") ;

 while(1)
 {
  rc = AttendreEvenement(&gEv,&HL,&HE) ;
  if ( rc == -1 )
     perror("AttendreEvenement:") ;
  else
     printf("Event:%d\n",HL) ;
  if ( HL == 0 )
     {
      char Buffer[100] ;
      fgets(Buffer,sizeof Buffer,stdin) ;
      Buffer[strlen(Buffer)]=0 ;
      printf("La touche enfoncee est %s \n",Buffer) ;
      sleep(10) ;
     }
  if ( HL == descphys )
```

```
  {
   struct Message Msg ;
   tm = sizeof(struct Message) ;
   rc = OrigineCouchePhysique(&L,&Msg,&tm ) ;
   if ( rc == -1 )
     {
      perror("OrigineCouchePhysique") ;
      return(-1) ;
     }
   else
     printf("bytes:%d:%d + %d : %s\n",rc,Msg.Entier1,Msg.Entier2,Msg.Commentaire ) ;

   IPR=IPDistante(&L)    ;
   PORTD=PortDistant(&L) ;
   Ipv4ToS(IPR, Buffer) ;
   printf("IP distante %s Port Distant %d \n",Buffer,PORTD ) ;

   /* calcul de la r□onse */
   Msg.Resultat = Msg.Entier1 + Msg.Entier2 ;
   rc = Repondre(&L,&Msg,sizeof(struct Message)) ;
   if ( rc == -1 )
     {
      perror("OrigineCouchePhysique") ;
      return(-1) ;
     }
    else
      printf("bytes ecrits :%d\n",rc ) ;
   }
 }
}
```

Le makefile

```
# Cpheser4me\makefile

all:    cli     ser     physlib.o       tcplib.o

physlib.o:      ../physlib/physlib.h    ../physlib/physlib.c
        echo "Compilation de physlib.o"
        cc -c ../physlib/physlib.c

tcplib.o:../tcplib/tcplib.h  ../tcplib/tcplib.c
        echo "Compilation de tcplib.o"
        cc -c ../tcplib/tcplib.c

evlib.o: ../evlib/evlib.c    ../evlib/evlib.h
        echo "Compilation de evlib.c"
        cc -c ../evlib/evlib.c

timerlib.o:     ../timerlib/timerlib.c  ../timerlib/timerlib.h
        echo "Compilation de timerlib.o"
        cc -c ../timerlib/timerlib.c

evtmlib.o:      ../evtmlib/evtmlib.c    ../evtmlib/evtmlib.h    ../timerlib/timerlib.o
        echo "Compilation de evtmlib.o"
        cc -c ../evtmlib/evtmlib.c
```

```
cli:    cli.c    physlib.o    tcplib.o timerlib.o    evlib.o    evtmlib.o
        echo "Compilation de client"
        cc  -o cli  cli.c physlib.o tcplib.o timerlib.o evlib.o evtmlib.o

ser:    ser.c    physlib.o    tcplib.o evlib.o
        echo "Compilation de serveur"
        cc -o ser  ser.c physlib.o tcplib.o evlib.o
```

8 Fermer une connexion existante

Si vous utilisez la commande

rc = CreerConnexion(&L,argv[1],atoi(argv[2]),argv[3],atoi(argv[4])) ;

dans une boucle ou une fonction, le deuxième appel sera refusé car il y'a déjà une connexion existante qui n'a pas été fermée.

Pour remédier à ce problème , il suffit de rajoute la fonction ***FermetureConnexion*** dans la librairie physlib qui se trouve dans le répertoire de même nom.

dans le fichier physlib.c , rajouter la fonction

```
int FermetureConnexion(struct Physique *pL  )
{
 close( pL->desc ) ;
}
```

dans le fichier physlib.h rajouter la déclaration

```
int CreerConnexion(struct Physique *pL,char * NCli,int pcli,char *NSer,int pser) ;
```

Voici un exemple d'utilisation de la nouvelle fonction

```
int rc ;
 char Message[80] ;
 struct Physique L ;
 char   Buffer[500] ;
 int       tm,i ;
...
while(i<10)
 {
 rc = CreerConnexion(&L,argv[1],atoi(argv[2]),argv[3],atoi(argv[4])) ;
 if ( rc == -1 )
    perror("CreerLiaison:") ;
 else
    fprintf(stderr,"Liaison Cree\n") ;
 sprintf(Message,"Ceci est un test %d \n",i) ;
 rc = VersCouchePhysique(&L,Message,strlen(Message)+1 ) ;
 if ( rc == -1 )
    perror("VersCouchePhysique") ;
 else
   fprintf(stderr,"Envoi de %d bytes\n",rc ) ;
 tm = sizeof(Buffer) ;
 rc = OrigineCouchePhysique(&L,Buffer,&tm) ;
 if ( rc == -1 )
    perror("OrigineCouchePhysique") ;
 else
   fprintf(stderr,"Reception de %d bytes %s\n",rc,Buffer ) ;
  FermetureConnexion(&L) ;
  i++ ;
 }
}
```

9 Calcul de la taille maximum de paquet envoyé sur réseau

Les deux programmes suivant permettent de tester quelle est la taille maximum autorisée de paquet par udp.

On envoit

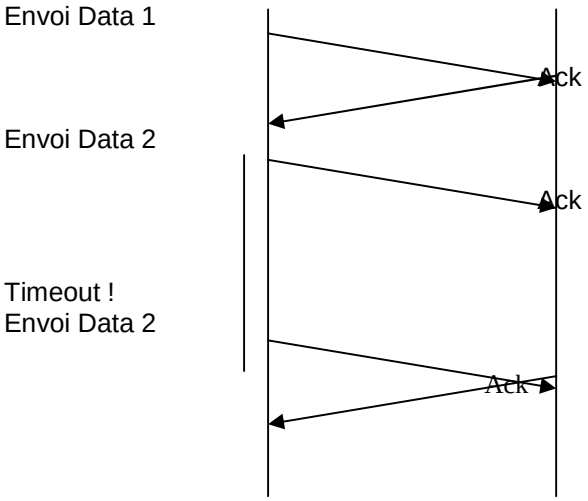
L'émetteur double la taille systématiquement jusqu'à ce que le paquet ne passe plus

```
/*------------------------------------------------------------------
   Vanstapel Herman
   C05\cli.c

  Test de la taille  : l'émetteur
------------------------------------------------------------------*/
#include <stdio.h>
#include "../physlib/physlib.h"

void main(int argc, char *argv[])
{
 int rc ;
 char *message ;
 struct Physique L ;
 char Tampon[65000] ;
 int  i ;

 if (argc!=5)
 {
    printf("cli client portc serveur ports\n") ;
    exit(1) ;
 }
 rc = CreerConnexion(&L,argv[1],atoi(argv[2]),argv[3],atoi(argv[4]))
;
 if ( rc == -1 )
    perror("CreerLiaison:") ;
 else
    printf("Liaison Créé \n") ;
 message = "Ceci est un test" ;
 i = 1 ;
 memset(Tampon,'X',65000) ;
 while(i < 65000 )
 {
 rc = VersCouchePhysique(&L,message, i ) ;
 if ( rc == -1 )
    perror("VersCouchePhysique") ;
 else
   printf("Envoi de %d bytes\n",rc ) ;
  i = i * 2 ;
 }
}
```

```
/*--------------------------------------
  C05\ser.c
----------------------------------------*/

#include <stdio.h>
#include "../physlib/physlib.h"

void main(int argc,char *argv[])
{
 int rc ;
 static  struct Physique L ; /* pour eviter un bug curieux !!!!!!!*/
 char message[100] ;
 int tm ;
 char Tampon[65000] ;

 printf("Ceci est le serveur\n") ;
 if ( argc!=5)
 {
  printf("ser ser port cli port\n") ;
  exit(1) ;
 }
 rc = CreerConnexion(&L,argv[1],atoi(argv[2]),argv[3],atoi(argv[4]))
;
 if ( rc == -1 )
    perror("CreerLiaison:") ;
 else
    printf("Liaison Créé \n") ;
 while(1 )
 {
  tm = 65000 ;
  rc = OrigineCouchePhysique(&L,Tampon,&tm ) ;
  if ( rc == -1 )
    perror("OrigineCouchePhysique") ;
  else
    printf("bytes:%d reçus  \n",rc ) ;
 }
}
```

Ecrire des protocoles fiables.

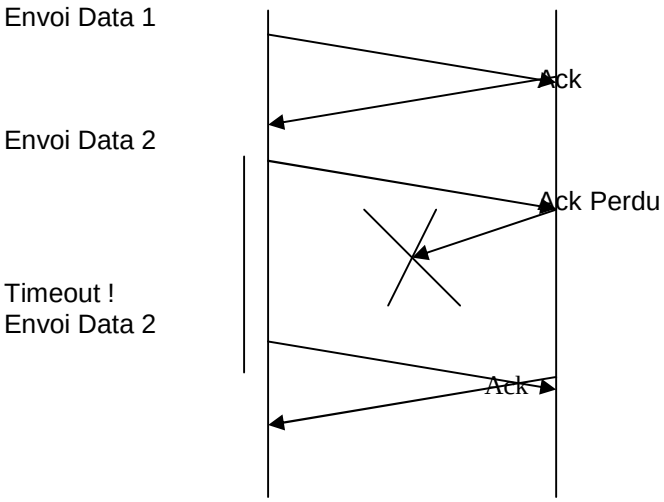
Le système avec acquittement



Remarques générales

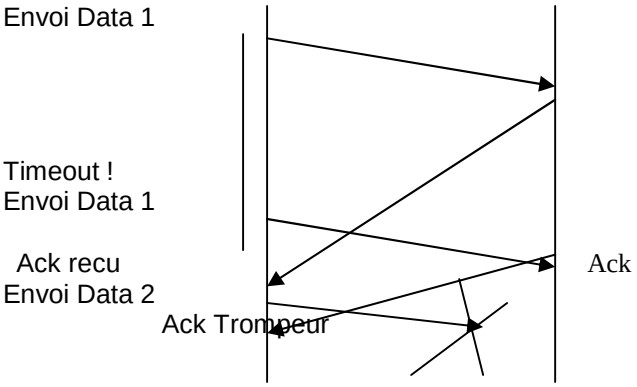
- On suppose que le canal de communication n'est pas parfait.
- Des trames peuvent être erronées ou perdues.
- La solution consiste en l'ajout d'un temporisateur. L'émetteur émet une trame et le récepteur n'émet une trame d'acquittement que si les données sont correctement reçues. Après expiration du timer, l'émetteur réémet la trame.
- Ce protocole présente une faille.

Le problème rencontré



- La couche réseau de A délivre un paquet numéro 1 à sa couche liaison de données. Le paquet est correctement reçu par B et transmis à la couche réseau de B. B envoie une trame d'acquittement à A.
- L'acquittement est perdu !!!!.
- Le temporisateur, armé par la couche liaison de données de A lors de l'envoi de la trame, expire ➔ Réexpédition de la trame.
- La trame dupliquée arrive également à la couche liaison de données de B d'une manière correcte et est transmise à la couche réseau.
- ***La solution au problème est de distinguer une trame émise une première fois d'une trame retransmise en utilisant simplement un numéro de séquence. Un champ numéro doit être ajouté à chaque trame.***

Le bug de l'émetteur



- La durée des temporisateurs doit être suffisamment longue pour permettre l'arrivée des acquittements.
- Si le temporisateur expire avant que l'acquittement arrive, l'émetteur envoie une trame dupliquée.
- L'acquittement arrive enfin et l'émetteur pense que c'est pour la trame qu'il vient d'envoyer. La trame suivante est envoyée et est perdue. L'acquittement pour la trame retransmise arrive enfin et le protocole croit à tord que c'est pour la trame qu'il vient de transmettre.
- ***La solution consiste à numéroter les acquitements***.

Annexe :Les librairies

Les librairies disponibles

Pour faciliter le développement du protocole réseau. Une série de librairies a été développée

Voici leur fonction

Nom	Description
Tcplib	Librairie permettant de créer une ***socket*** de type ***UDP*** ou ***TCP***. De transformer une adresse IP sous forme de chaîne vers un entier et vice versa et de l'afficher.
Physlib	Libraire permettant de créer une connexion de type udp. D'écrire et de lire des bytes sur cette connexion. Une fonction répondre est également disponible pour les serveurs qui reçoivent des données en provenance de différents clients.
Evlib	Cette libraire permet de gérer plusieurs périphériques d'entrée simultanément dont le clavier, les sockets réseau, les fichiers. L'objectif est d'éviter d'être bloqué sur un seul handle.
Timerlib	Cette librairie permet de gérer plusieurs ***timers*** simultanément. Les timers travaillent au niveau de la milliseconde.
Evtmlib	Cette libraire permet de gérer les ***déclenchements des timers*** et de tester plusieurs périphériques d'entrée simultanément dont le clavier, les ***sockets réseau***, les fichiers. L'objectif est d'éviter d'être bloqué sur un seul ***handle***.

Tcplib.h

Objectif

Librairie permettant de créer une ***socket*** de type UDP ou TCP. De transformer une adresse IP sous forme de chaîne vers un entier et vice versa et de l'afficher

Le fichier .h

```
/*---------------------------------------------------------------------
 ====================================================================
  Vanstapel Herman EPL

 Fonctions de base TCP/IP
 ====================================================================
 --------------------------------------------------------------------*/

#ifndef TCPLIB
#define TCPLIB
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <ctype.h>

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) ;
#endif TCPLIB
```

physlib.h

Objectif

Libraire permettant de créer une connexion de type udp. D'écrire et de lire des bytes sur cette connexion. Une fonction répondre est également disponible pour les serveurs qui reçoivent des données en provenance de différents clients.

Librairie(s) nécessaire

Tcplib

Le fichier .h

```
/*-----------------------------------------------------------------

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  Vanstapel Herman EPL
 physlib\physlib.h
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 -----------------------------------------------------------------*/

#ifndef PHYSLIB
#define PHYSLIB
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <netinet/in.h>
#include "../tcplib/tcplib.h"
#include <errno.h>

struct Physique
{
 struct sockaddr_in ai ; /* adresse du client */
 struct ip4 ip4cl,ip4se ;
 struct sockaddr_in sain ;       /* adresse du serveur */
 struct sockaddr_in oain ;    /* origine du client */
 int desc  ;
 char Type ;
} ;

extern int errno ;

int CreerConnexion(struct Physique *pL,char * NCli,int pcli,char
*NSer,int pser) ;
int VersCouchePhysique(struct Physique *pL,void *message,int tm ) ;
int OrigineCouchePhysique(struct Physique *pL, void *message,int *tm
) ;
int Repondre(struct Physique *pL,void *message,int tm ) ;
int IPDistante(struct Physique *pL) ;
int PortDistant(struct Physique *pL) ;
#endif
```

Explications

int ***CreerConnexion***(struct Physique *__pL__,char * __NCli__,int __pcli__,char *__NSer__,int __pser__) ;

Cree une connexion Réseau port pour ce programme.Cette connexion vous permettra d'envoyer et de recevoir des données.

__pL__ désigne les paramètres de la connexion seront stockés dans la structure Physique poinée par pL par CreerConnexion.
__Ncli__ est le nom ou l'ip de la machine surlaquelle votre programme tourne.
__pcli__ est le port du utilisé par ce programme.
__Nser__ est le nom de la machine vers laquelle on établit la connexion.
__Pser__ est le port du programme vers lequel on établit la connexion.

En cas de succès, un handle désignant la socket de connexion est retourné.
Sinon –1 est retourné.

__Remarque 1:__ Un programme serveur a toujours les paramètres __Nser__ et __pser__ à __Null__.
__Remarque 2:__ Les paramètres Ncli et Nser peuvent être identique si les deux programmes qui tournent sur la même machine mais les numéros de ports doivent toujours être différent.

int ***VersCouchePhysique***(struct Physique *__pL__,void *__message__,int __tm__) ;

Cette fonction permet d'envoyer des données par la connexion réseau.

__pL__ désigne la connexion que l'on a créé via __creerconnexion__
__message__ la structure que l'on souhaite envoyer au destinataire.
__Tm__ la taille de la structure message envoyée.

En cas de succès le nombre de bytes écrits est retourné.
Sinon –1 est retourné.

int ***OrigineCouchePhysique***(struct Physique *__pL__, void *__message__,int *__tm__) ;

Permet de lire des données en provenance de la connexion réseau.

__pL__ désigne la connexion que l'on a créé via __creerconnexion__
Message contient le buffer qui recevra les données.
__Tm__ est initialisé avec la taille du buffer.

En cas de succès le nombre de bytes reçus est retourné. Tm conient aussi le nombre de bytes lus.
Sinon –1 est retourné.

int ***Repondre***(struct Physique *__pL__,void *__message__,int __tm__) ;

Cette fonction est utilisée par les serveurs après un appel à OrigineCouchephysique. Cette fonction permet de répondre au programme client qui a envoyé les données.

__pL__ désigne la connexion que l'on a créé via __creerconnexion__
__message__ la structure que l'on souhaite envoyer au destinataire.
__Tm__ la taille de la structure message envoyée.

En cas de succès le nombre de bytes écrits est retourné.
Sinon –1 est retourné.

int ***IPDistante***(struct Physique ****pL***) ;

Cette fonction retourne l'IP du client qui a envoyé le dernier message

pL désigne la connexion que l'on a créé via ***creerconnexion***

En cas de succès l'IP est retourné.

int ***PortDistant***(struct Physique ****pL***) ;

Cette fonction retourne le port du client qui a envoyé le dernier message

pL désigne la connexion que l'on a créé via ***creerconnexion***

En cas de succès le port est retourné.

evlib.h

Objectif

Cette libraire permet de gérer plusieurs périphériques d'entrée simultanément dont le clavier, les sockets réseau, les fichiers. L'objectif est d'éviter d'être bloqué sur un seul handle.

Le fichier .h

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

  evlib\evlib.h
  Ceci  est la librairie evlib.h
  Utilisation d'evenements
-------------------------------------------------------------------*/

#ifndef EVLIB
#define EVLIB

#include <sys/types.h>
#include <sys/time.h>

struct gEvenement{
       fd_set rfds ;
         fd_set wfds ;
       struct timeval tv ;
       int maxdesc ;
};

int PrepareEvenement( struct gEvenement *gEv,int HLs[],int HEs[],int nbr,
int ts,int tms );
int AttendreEvenement(struct gEvenement *gEv,int *HL,int *HE) ;
void ActiverCouche(struct gEvenement *gEv,int num ) ;
void DesactiverCouche(struct gEvenement *gEv,int num ) ;
#endif
```

Explications

```
int PrepareEvenement( struct gEvenement *gEv,int HLs[],int HEs[],int nbr,
int ts,int tms );

Cette fonction permet de préparer la fonction AttendreEvevement qui
encapsule le select de unix.

gEv contiendra les parametres à passer à AttendreEvenement.
HLs contient les descripteurs à tester en lecture. Chaque entrée du tableau
en lecture contient 1 si le numéro de descripteur correspondant doit être
testé.
Hes contient les descripteurs à tester en écriture
Nbr contient le numéro du descripteur le plus haut à tester.
Ts Précise en secondes le temps d'attente en seconde d'Attendre
Evenement
Tms exprime le temps en micro-secondes ( 10-6 secondes )

Remarque : pour tester le clavier il faut placer l'erntrée zéro du tableau
HL à 1.
```

int ***AttendreEvenement***(struct gEvenement *gEv,int *HL,int *HE) ;

Cette fonction permet d'attendre si des descripteurs sont disponibles en lecture.

gEv Contient la liste des descripteurs à tester.
HL Contient le descripteur en lecture qui est disponible en lecture. Il vaudra 0 si c'est le clavier ou le descripteur retourné par ***creerconnexion***.
HE Le descripteur en écriture disponible.

En cas de succès,
Elle retourne 0 si c'est un timeout
Sinon elle retourne le nombre maximum de descripteurs sur lesquels une opération est possible
Sinon
la fonction retourne -1 si il y'a un problème

Timerlib.h

Objectif

Cette librairie permet de gérer plusieurs ***timers*** simultanément. Les timers travaillent au niveau de la milliseconde.

Le fichier .h

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

   Prototype pour la librairie timerlib.h

-----------------------------------------------------------------*/

#ifndef TIMERLIB
#define TIMERLIB
#include <unistd.h>
#include <sys/times.h>

struct ListeTimer
{
 int num    ;
 long clktck ;
 struct ListeTimer *psuiv ;
};

int microtoclktck( int micro ) ;
int millitoclktck( int milli ) ;
struct ListeTimer * StartTimer( int num, int to, struct ListeTimer *pl) ;
struct ListeTimer *TestTimer(int *num , struct ListeTimer *pl) ;
struct ListeTimer *StopTimer(int num , struct ListeTimer *pl ) ;
void   AfficheTimer(struct ListeTimer *pl) ;
#endif
```

Explications

```
struct ListeTimer * StartTimer( int num, int to, struct ListeTimer *pl) ;

Ajoute un timer à la liste des timers.

Ajoute le timer num qui expirera dans to millisecondes à la liste pl de
timer.

La valeur retournée est toujours la nouvelle liste avec le timer que nous
venons d'ajouter.

Remarque 1: Si la liste des timers est vide, elle doit être intialisée à
NULL.
Remarque 2: La fonction ne vérifie pas l'ajout de doublons. On considère
comme doublons, deux timers qui portent le même numéro.
```

struct ListeTimer ***TestTimer***(int ****num*** , struct ListeTimer ****pl***) ;

Cette commande permet de tester si timer s'est déclenché (délai expiré).
Si c'est le cas, Le Timer concerné est retiré de la liste.

Num contiendra toujours le numéro de timer qui a expiré. Sinon il prendra la valeur ***–1,*** si rien ne s'est déclenché.
pl contient la liste des timers.

La valeur retournée est toujours la nouvelle liste avec le timer que nous venons d'ajouter.

Remarque : cette commande est utilise par le code d'attendreevenementtm. Il ne faut paz l'utiliser ***TestTimer*** dans vos programmes.

struct ListeTimer ***StopTimer***(int num , struct ListeTimer *pl) ;

Cette commande retire un timer de la liste.

Num contient le numéro du timer que l'on souhaite retirer.
Pl contient la liste des timers.

La valeur retournée est toujours la nouvelle liste avec le timer que nous venons d'ajouter.

Remarque : Toujours utiliser cette commande avant d'utiliser StartTimer.
Ceci vous garantit la suppression de doublons.

evtmlib.h

Objectif

Cette libraire permet de gérer les ***déclenchements des timers*** et de tester plusieurs périphériques d'entrée simultanément dont le clavier, les ***sockets réseau***, les fichiers. L'objectif est d'éviter d'être bloqué sur un seul ***handle***.

Librairie(s) nécessaire

Evlib, Timerlib.

Le fichier .H

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

  Ceci  est la librairie evtmlib.c
  Utilisation d'evenements

  evtmlib.h
-----------------------------------------------------------------*/

#ifndef EVTMLIB
#define EVTMLIB

#include "../evlib/evlib.h"
#include "../timerlib/timerlib.h"

/*       attendre un timeout          */
#define TIMEOUT 5000

int AttendreEvenementtm(struct gEvenement *gEv,int *HL,int *HE,struct
ListeTimer **pl) ;
#endif
```

Explications

```
int AttendreEvenementtm(struct gEvenement *gEv,int *HL,int *HE, ListeTimer
**pl) ;

Cette fonction permet d'attendre si des descripteurs sont disponibles en
lecture.

gEv    Contient la liste des descripteurs à tester.
HL     Contient le descripteur en lecture qui est disponible en lecture ou la
valeur TIMEOUT. Il vaudra 0 si c'est le clavier ou le descripteur retourné
par creerconnexion.
HE     Le descripteur en écriture disponible.
Pl     pointeur vers la liste des timers à tester.

En cas de succès,
Elle retourne 0 si c'est un timeout
Sinon elle retourne le nombre maximum de descripteurs sur lesquels une
opération est possible
Sinon
la fonction retourne -1 si il y'a un problème
```

Annexe : Les opérations sur les fichiers

Tous ces fichiers se trouvent dans *le répertoire Fichiers* des librairies

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) ;
  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 ;
}
```

Suite du listing page suivante

```
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) ;
}
```

Annexe :Les directives de compilation SUN

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

Dans le source C pour les instructions suivantes rajouter le header

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

Il faut ensuite editer le fichier ***makefile*** et ajouter aux lignes du style

```
cc -o ser  ser.c physlib.o tcplib.o
cc  -o cli  cli.c physlib.o tcplib.o

Vous ajoutez les paramètres suivants en gras italique.

cc -o ser  ser.c physlib.o tcplib.o –lsocket –lnsl
cc  -o cli  cli.c physlib.o tcplib.o –lsocket –lnsl
, :x
```

Je remercie monsieur Mercenier pour sa collaboration.

Bibliographie

Ouvrages

- Réseaux : Andrew Tanenbaum , InterEditions.
- TCP/IP : Karanjit S. Siyan, S&SM.
- Pratique des réseaux d'entreprise : Jean-Luc Montagnier, Eyrolles.
- Cisco TCP/IP : Chris Lewis, Mc Graw Hill
- Réseaux, Tony Bastiannelli, ASBL DEFI.
- Les réseaux, Pujolle, Eyrolles.
- TCP/IP, Addison-Wesley, K. Washburn & J.T Evans.
- TCP/IP, Michel Beaulen, ASBL DEFI.
- TCP/IP MCSE : Emmett Dulaney, Sherwood Lawrence…, S&SM
- Guide pratique des réseaux Ethernet, Charles E. Spurgeon, vuibert

Encyclopédies

- Encyclopédie Universalis.

Revues

- PC Expert.
- Linux magazine.
- Linux pratique.

Remerciements

Je remerçie les étudiants suivants pour leurs critiques constructives.

Gouverneur Thomas Informatique et systèmes Réseaux et Télécommunications
Equeter Thomas Informatique et systèmes Réseaux et Télécommunications

.