TD6: La programmation client/serveur avec sockets (le reste est que du C)

Exo17: Définition de protocole pour le serveur de calcul. Q17.1 reprendre le serveur de calcul et définir un protocole/interface à la manière du cours. Il faut penser à définir correctement les structures des requêtes, les structures des réponses, les codes des requêtes et les codes d'erreur.Ici le protocole est particulièrement simple puisque toutes les requêtes envoyées du client vers le serveur contiennent la même chose : l'opérateur et deux données. Il n'est donc pas nécessaire de définir plusieurs types de requêtes et leur traitement.

```c
#include <stdio.h>  #include <stdlib.h>   // Client
#include <string.h>  #include <unistd.h>
#include "fonctionsSocket.h"
 // Protocole includes   #include "protocole.h"
int main(int argc, char **argv)
{   int sock, err;
    char boucle = 'o';
    char operateur;
   // Structures de requete/reponse
  TRequete req;
  TReply rep;
 if ( argc != 3 ) {printf(" nom_machine no_port\n"); exit(1); }
  //Creation d'une socket, domaine AF_INET, protocole TCP
   sock = socketClient_EAD( argv[1], atoi( argv[2]) );
 if ( sock < 0 ) {printf( "client : erreur socketClient\n"); exit(2);  }
  do {   /** Saisie de l'operation */
   scanf(" %c", &operateur );
   switch( operateur ) {
   case '+' : req.operateur = PLUS; break;
   case '-' : req.operateur = MOINS; break;
   case '*' : req.operateur = FOIS; break;
   case '/' : req.operateur = DIV; break;
   default: printf("Client unknown operator\n");
   }
  printf("\t donner l'operande 1 : ");
   scanf(" %d", &req.firstOp );
   printf("\t donner l'operande 2 : ");
   scanf(" %d", &req.secondOp );
 printf("client : envoi de - %d %c %d - \n",
    req.firstOp, operateur, req.secondOp );
   /** Envoi de l'operation en une seule structure */
   err = send( sock, (void*) &req, sizeof( req ), 0);
  if ( err!= sizeof( req ) ) { perror(" err send de la requete");
    shutdown(sock, 2);  exit(3);   }
   /*  * Reception du resultat */
   err = recv ( sock, (void *) &rep, sizeof( rep ), 0 );
  if ( err == -1 ) {perror(" err a la reception");
    shutdown(sock, 2); exit(6);  }
   // Traitement de la reponse
   switch( rep.errorCode ) {
   case ERR_OK : printf("client : resultat recu : %d\n", rep.resultat);
    break;
  case ERR_DIV_ZERO : printf("Server error : division by zero\n");
    break;
  case ERR_UNKNOWN_OP : printf("Server err : \n");
    break;
  default :   printf("Client error : unknown error code\n");
  }
    scanf(" %c", &boucle );
 } while ( boucle == 'o' );
 shutdown(sock, 2);
 close(sock);
 return 0;
}
```

```c
#include <string.h> #include <stdio.h>// Serveur
#include <stdlib.h> #include <unistd.h>
#include <errno.h> #include <sys/socket.h>
#include <netinet/in.h> #include "protocole.h"
/* taille du buffeur de reception */
#define TAIL_BUF 100
// Fonction de traitement de la requete utilisee par
// le processus fils
void traitReq( int sockTrans )
{
  TRequete  req;        /* Requete de calcul */
 TReply       rep;        /* Reponse au client */
  int          encore;  // test de boucle pour les envois
 pid_t         myPid;       /* Identif du process */
 int           err;         /* code d'erreur */
 encore = 1;
  myPid = getpid();

 while ( encore == 1 ) {
      /** Reception  affichage de l'operation en provenance
du client * si ce dernier a coupe la connexion, on sort sans
rien recevoir */
   err = recv( sockTrans, &req, sizeof( req ), 0);
   if (err < 0) { perror(" err dans la reception d'operateur");
    shutdown(sockTrans, 2); exit(4);   }
   if ( err == 0 ) {
    printf("serveur %d : fin de la connexion client\n", myPid);
     encore = 0;
    } else {
        char operateur = '#';
    switch( req.operateur ) {
          case PLUS : operateur = '+'; break;
          case MOINS : operateur = '-' ; break;
          case FOIS : operateur = '*'; break;
          case DIV : operateur = '/'; break;
          default: printf("Client unknown operator\n");
   }
    printf("serveur %d : voila l'operation recue : %d
     %c %d\n", myPid, req.firstOp, operateur, req.secondOp );

 switch ( req.operateur ) {
    case PLUS :  rep.errorCode = ERR_OK ;
  rep.resultat = req.firstOp + req.secondOp;
  break;
    case MOINS :   rep.errorCode = ERR_OK ;
  rep.resultat = req.firstOp - req.secondOp;
  break;
    case FOIS : rep.errorCode = ERR_OK ;
  rep.resultat = req.firstOp * req.secondOp;
  break;
    case DIV :
```

suite1

```
/******PROTOCOLE.H***/

#include "fonctionsSocket.h"

// Types enumeres : request type and error type
typedef enum { PLUS, MOINS, FOIS, DIV } TOperator;
typedef enum { ERR_OK, ERR_DIV_ZERO ,
                    ERR_UNKNOWN_OP } TErrorCode;

// Structure de requete
 typedef struct {
   TOperator operateur;
    int firstOp;
    int secondOp;
 } TRequete;

//  Structure de reponse
 typedef struct {
   TErrorCode errorCode;
    int resultat;
 } TReply;
```

//dans le protocole on a que ce qui concerne la communication
//donne les structures des données utilisées par la communication avec le serveur de calcul

FonctionsSocket.h

int **socketServeur_EAD(**unsigned short port);
int **socketClient_EAD**(const char *nom_machine, unsigned short port);

int **socketUDP_EAD**(unsigned short port);
struct **sockaddr** *socketAddr_EAD(const char *nom_machine, unsigned short port)
socklen_t tailleAddr_EAD(void);

socketServeur_EAD
Socket-initialise struct sockaddr_in nom avec sin_family, port,.s-addr-bind-listn ...

socketClient_EAD
Struct hostent *host, sockaddr_in addr; - socket-memset , sin_family, sin_port, gethostbyname(addr)-addr.sin_addr.s_addr = ((struct in_addr *) (host->h_addr_list[0]))->s_addr;
Connect

socketUDP_EAD(
Socket-init @socket - bind


```
  if ( req.secondOp == 0 ) {
    rep.errorCode = ERR_DIV_ZERO;
   } else {
    rep.errorCode = ERR_OK ;
   rep.resultat = req.firstOp / req.secondOp;
  }
  break;
   default :
  printf("serveur %d : erreur, operateur inconnu\n", myPid);
  rep.errorCode = ERR_UNKNOWN_OP;
  rep.resultat = 0;
   }

   err = send( sockTrans, (void *) &rep , sizeof( rep ), 0 );
   if ( err != sizeof( rep ) ) {perror(" err dans l'envoi du resultat");
   shutdown(sockTrans, 2); exit(7);   }
  }
 }
}
 // Fonction principale
int main(int argc, char** argv)
{
   int   sock_cont, sock_trans;
    struct sockaddr_in nom_transmis;
    socklen_t      size_addr_trans;
   int   pid;            /* PID du processus fils */

   if ( argc != 2 ) { printf ( "usage : serveur no_port\n" ); exit( 1 );   }

   size_addr_trans = sizeof(struct sockaddr_in);
    /** Creation de la socket, protocole TCP  */
    sock_cont = socketServeur_EAD( atoi( argv[1] ) );
   if ( sock_cont < 0 ) { printf( " err socketServeur\n" );
    exit( 2 );   }
     /* * Boucle du serveur */
   for (;;) {
     /*  * Attente de connexion   */
    sock_trans = accept(sock_cont,
          (struct sockaddr *)&nom_transmis,
          &size_addr_trans);
    if (sock_trans < 0) { perror(" erreur sur accept"); exit(3);    }
     pid = fork();
     switch ( pid ) {
    case 0 :  // Processus fils, appel de la fonction
        // de traitement des requetes
        close( sock_cont );
         traitReq( sock_trans );
    /* * arret de la connexion et fermeture */
          shutdown(sock_trans, 2);
           close(sock_trans);
     // fin du fils   exit(0);
                    break;
    case -1 : // erreur
    perror("Creation processus fils");
     // Fermeture des sockets et fin
   close( sock_cont );
   shutdown(sock_trans, 2);
   close(sock_trans);   exit(3);
    default : // processus pere
    // Ferme la socket connectee et attend la suite
   close(sock_trans);
    }
   }
  return 0;
}
```

Exo18: Serveur de numéros de port

Le fichier /etc/services n'étant pas forcément le même sur toutes les machines, il serait utile de disposer d'un serveur centralisant toutes ces informations (manière du DNS pour les noms et IP).

Le serveur doit être capable de répondre aux requêtes de création et destruction d'un nouveau service (avec ou sans numéro de port fixé), de recherche du numéro de port d'un service ou de recherche d'un nom de service.

Q18.1 Proposer un protocole d'accès à ce serveur, donner le code du serveur et le code d'un client.

Nous ne nous intéressons pas ici à la gestion de la table des services à l'intérieur du serveur, aussi nous n'avons pas géré de ré-allocation dans cette table, ce problème étant plutôt de l'ordre de l'algorithmique que de la communication en distribué. Lorsqu'un service est supprimé, nous nous contentons de rendre l'entrée dans la table indisponible. Remarquer ici la définition d'un fichier serveur.h où sont déclarés l'ensemble des structures de données propres au serveur mais qui ne concerne pas le client. Ceci est le cas pour la structure service entry de la table des services qui regroupe l'ensemble des informations liées à un service.

Dans ce cas, la structure de donnée utilisée pour les requêtes est plus complexe car les données échangées ne sont pas toujours les mêmes. Les requêtes possibles sont :

— une requête de déclaration qui prend en entrée le numéro de port, le nom du service et du protocole. Le retour de cette requête donne un code d'erreur.

— une requête de demande d'attribution de port qui prend en entrée les noms du service et du protocole. Le retour de cette requête donne un code d'erreur et le numéro de port attribué.

— des requêtes de consultation à partir du nom ou du numéro de port qui retournent un code d'erreur et les infos manquantes.

— une requête de suppression qui prend en entrée le nom du service. Le retour de cette requête donne un code d'erreur.

— une requête de fin de connexion qui est un autre mode que celui utilisé dans le serveur de calcul.

Pour chacune des requêtes il est nécessaire de déterminer quelles sont les données en appel (entrée) ou en retour (sortie). Les structures des requêtes sont constituées sur cette base.

La structure du serveur n'est plus une structure de serveur concurrent dans la mesure où les clients doivent accéder à la même table de données partagée. Il n'est pas simple d'avoir un partage entre processus. Cela peut se faire par mémoire partagée, par communication, en ayant des threads, etc. Vous verrez ces aspects dans le cours d'algorithmique concurrente. Nous nous sommes donc contentés d'un select. Noter tout de même la structure du select avec clients multiples dont vous n'aviez pas encore eu de correction.

Il faut noter que la corrections donnée n'est qu'une implantation possible du protocole. D'autres possibilités existent et peuvent être aussi valides si vous avez des arguments pour les justifier. En particulier, le choix d'une structure unique peut être contesté au profit de plusieurs structures ayant pour première donnée un code de requête. De la même manière, il est possible de s'interroger sur la nécessité de définir différentes structures de données dans la mesure où il serait possible d'utiliser toujours à la même sans incidence sur les performances de communication dans la mesure où le fait de définir une union nous conduit à définir une structure de données qui recouvre l'ensemble des définitions. Dans le cas présent, les choix sont justifiés par l'amélioration de la lisibilité du code.

/**serveur.h**/

```
#ifndef SERVEUR_H
#define SERVEUR_H
/* Nombre max de connexions */
#define MAX_CONNECT 32
typedef enum { OCCUPED, FREE } TState;
// Service table entry
typedef struct {
  TState state;
  char   nom[NAME_SIZE];
  char   proto[PROTO_SIZE];
  short  port;
} TServiceTableEntry;
// Service Table size
#define SERVICE_TABLE_SIZE 10
// code d'erreur non trouve dans la table
#define NOT_FOUND -1
#endif
```

/*PROTOCOLE QUE DE LA COMMUNICATION***/**

```
 #include "fonctionsSocket.h"
// Constantes specifiques au protocole
#define NAME_SIZE 50
#define PROTO_SIZE 4
 // Types enumeres : request type and error type
// codes des requetes
typedef enum {
        AJOUT,
        ATTRIB,
        CONSULT_NOM,
        CONSULT_PORT,
        SUPPRESS,
        FIN
 } TCodeRequest;

// codes d'erreur
typedef enum {
        ERR_OK,
        ERR_UNKNOWN_REQUEST,
        ERR_NO_MORE_ENTRY,
        ERR_NAME_ALREADY_EXIST,
        ERR_UNKNOWN_NAME,
        ERR_UNKNOWN_PORT
} TErrorCode;

// Structures des requetes**********
//**** Partie specifique**********

// Requete d'ajout
typedef struct {
    char nom[NAME_SIZE];
     char proto[PROTO_SIZE];
     short port;
 } TAjoutReq;

// Requete d'attribution
typedef struct {
   char nom[NAME_SIZE];
  char proto[PROTO_SIZE];
} TAttribReq;
```

```
 // Requete de consultation par nom
typedef struct {
  char nom[NAME_SIZE];
 } TConsultNomReq;

// Requete de consultation par port
typedef struct {
  short port;
 } TConsultPortReq;

// Requete de suppression
typedef struct {
  char nom[NAME_SIZE];
} TSuppressReq;

// Type de requete generique
 typedef struct {
  TCodeRequest codeReq;

  union {
     TAjoutReq      ajout;
     TAttribReq     attrib;
     TConsultNomReq  consultNom;
     TConsultPortReq consultPort;
     TSuppressReq    suppress;
  } specific ;

} TRequest;

//  Structure des reponses***
// Partie specifique
// Reponse d'ajout n'a pas de partie specifique
// Reponse d'attribution
typedef struct {
  short port;
 } TAttribRep;

// Reponse de consultation par nom
typedef struct {
  char proto[PROTO_SIZE];
  short port;
 } TConsultNomRep;

// Reponse de consultation par port
typedef struct {
  char nom[NAME_SIZE];
 char proto[PROTO_SIZE];
 } TConsultPortRep;

// Reponse de suppression n'a pas de parti//specifique
 // Type de reponse generique

typedef struct {
  TErrorCode codeRep;
  union {
            TAttribRep          attrib;
            TConsultNomRep  consultNom;
            TConsultPortRep    consultPort;
         } specific;
} TReply;       //fin protocole.h
```

```c
// Standard includes
#include <stdio.h> #include <stdlib.h>
#include <string.h> #include <unistd.h>
#include "fonctionsSocket.h"
// Protocole includes
#include "protocole.h"
// Fonction d'affichage du resultat
void traitRep( TCodeRequest errorCode)
{// Traitement de la reponse
  switch( errorCode ) {
       case ERR_NAME_ALREADY_EXIST:
        printf("Server error : Name already used \n");
        break;
      case ERR_UNKNOWN_NAME :
        printf("Server error : Name not found \n");
        break;
      case ERR_UNKNOWN_PORT :
        printf("Server error : Port not found \n");
        break;
      default :
        printf("Client error : unknown error code\n");
  }
}
//*****Principal******
int main(int argc, char **argv)
{
  int sock,err;
  // Structures de requete/reponse
  TRequest req;
  TReply rep;

  if (argc != 3) {
    printf("usage : client nom_machine no_port\n");
    exit(1);}
/* Creation d'une socket, domaine AF_INET, protocole TCP */
  sock = socketClient_EAD( argv[1], atoi( argv[2]) );
  if (sock < 0){printf( "client : erreur socketClient\n");
    exit(2);}
   // Creation d'une requete d'ajout
  req.codeReq = AJOUT;
  strcpy( req.specific.ajout.nom, "ead" );
  strcpy( req.specific.ajout.proto, "tcp" );
  req.specific.ajout.port = 2609;
  // envoi de la requete
  err = send( sock, (void*) &req, sizeof( req ), 0);
  if ( err!= sizeof( req ) ) {
    perror("client : erreur sur le send de la requete");
    shutdown(sock, 2);
    exit(3);
  }
  printf("client : envoi ajout realise\n");
```

```c
// Creation d'une requete de consultation
req.codeReq = CONSULT_NOM;
strcpy( req.specific.consultNom.nom, "ead" );
// envoi de la requete
err = send( sock, (void*) &req, sizeof( req ), 0);
if ( err!= sizeof( req ) ) {
  perror("client : erreur sur le send de la requete");
  shutdown(sock, 2);
  exit(3);
}
printf("client : envoi consult realise\n");

// Reception du resultat
err = recv ( sock, (void *) &rep, sizeof( rep ), 0 );
if ( err == -1 ) {
  perror("client : erreur a la reception");
  shutdown(sock, 2);
  exit(6);
}

if ( rep.codeRep != ERR_OK ) {
  traitRep( rep.codeRep );
} else {
  printf("Client : nom = %s proto = %s port = %d \n",
     req.specific.consultNom.nom,
     rep.specific.consultNom.proto,
     rep.specific.consultNom.port );
}
    // Creation d'une requete de suppression
    req.codeReq = SUPPRESS;
    strcpy( req.specific.suppress.nom, "ead" );
    // envoi de la requete
    err = send( sock, (void*) &req, sizeof( req ), 0);
    if ( err!= sizeof( req ) ) {
      perror("client : erreur sur le send de la requete");
      shutdown(sock, 2);
      exit(3);
    }
    printf("client : envoi suppress realise\n");
    // Reception du resultat
    err = recv ( sock, (void *) &rep, sizeof( rep ), 0 );
    if ( err == -1 ) {
      perror("client : erreur a la reception");
      shutdown(sock, 2);
      exit(6);
    }
    if ( rep.codeRep != ERR_OK ) {
      traitRep( rep.codeRep );
    } else {
      printf("Client : request correctly executed \n");
    }
    // Creation d'une requete de fin
    req.codeReq = FIN;
    // envoi de la requete
    err = send( sock, (void*) &req, sizeof( req ), 0);
    if ( err!= sizeof( req ) ) {
      perror("client : erreur sur le send de la requete");
      shutdown(sock, 2);
      exit(3);
    }
    printf("client : envoi suppress realise\n");
      /*Fermeture de la connexion et de la socket*/
    shutdown(sock, 2);
    close(sock);
    return 0;
}
```

```c
// Standard includes
#include <string.h>     #include <stdio.h>
#include <stdlib.h>     #include <unistd.h>
#include <errno.h>      #include <sys/types.h>
#include <sys/time.h>   #include <sys/socket.h>
#include <netinet/in.h> #include "serveur.h"
// Protocol includes
#include "protocole.h"
// Service Table
TServiceTableEntry serviceTable[ SERVICE_TABLE_SIZE ];

// ----------------------------------------
// Fonction de traitement de la table
// ----------------------------------------

// Initialisation de la table
void initServiceTable()
{  int i;
  for ( i = 0 ; i < SERVICE_TABLE_SIZE ; i++ ) {
    serviceTable[ i ].state = FREE;
  }
}

// recherche dans la table par nom
// retourne -1 si nom trouve
int getIndiceByName(char* nom)
{ int i = 0;
  int retour;
  int found = 0;
  while (( found == 0) && ( i < SERVICE_TABLE_SIZE )) {
    if ( strcmp( nom, serviceTable[ i ].nom ) == 0 ) {
      found = 1;
    } else {
      i++;
    }
  }
  if ( i == SERVICE_TABLE_SIZE ) retour = NOT_FOUND; else retour = i;
  return retour;
}
```

```c
// ----------------------------------------------------
// Fonction de traitement d'une requete de consultation par nom
// ----------------------------------------------------
TErrorCode consultNom( TConsultNomReq consultNom , char* proto, short* port)
{
  int i;
  TErrorCode retour;

  i = getIndiceByName( consultNom.nom );
  if ( i == NOT_FOUND ) { // Name doesn't exist

    retour = ERR_UNKNOWN_NAME;

  } else {

    // recopie des infos
    strcpy( proto, serviceTable[ i ].proto );
    (*port) = serviceTable[ i ].port ;

    retour = ERR_OK;
  }

  return retour;
}
```

```c
// recherche dans la table par port
int getIndiceByPort(short port)
{ int i = 0;
  int retour;
  int found = 0;
  while (( found == 0) && ( i < SERVICE_TABLE_SIZE )) {
    if ( port == serviceTable[ i ].port ) {
      found = 1;
    } else {
      i++;
    }
  }
  if ( i == SERVICE_TABLE_SIZE ) retour = NOT_FOUND; else retour = i;
  return retour;
}
// recherche le prochain libre de la table
int getNextFree()
{
  int i = 0;
  int retour;
  int found = 0;

  while (( found == 0) && ( i < SERVICE_TABLE_SIZE )) {

    if ( serviceTable[ i ].state == FREE ) {

      found = 1;

    } else {

      i++;
    }
  }

  if ( i == SERVICE_TABLE_SIZE ) retour = NOT_FOUND; else retour = i;
  return retour;
}
  // ---------------------------------------------
  // Fonction de traitement d'une requete d'ajout
  // ---------------------------------------------
  TErrorCode ajoutTable( TAjoutReq ajout )
  {
    int i;
    TErrorCode retour;

    i = getIndiceByName( ajout.nom );
    if ( i != NOT_FOUND ) { // Name exist

      retour = ERR_NAME_ALREADY_EXIST;

    } else {

      i = getNextFree();
      if ( i == NOT_FOUND ) { // no more entry

        retour = ERR_NO_MORE_ENTRY;

      } else {

        // ajout service
        serviceTable[ i ].state = OCCUPED;
        strcpy( serviceTable[ i ].nom, ajout.nom );
        strcpy( serviceTable[ i ].proto, ajout.proto );
        serviceTable[ i ].port = ajout.port;

        retour = ERR_OK;
      }
    }

    return retour;
  }
```

```c
// ----------------------------------------------------
// Fonction de traitement general de la requete
void traitReq( TRequest req, TReply* rep )
{  // parametres des fonctions specifiques
  short port;
  char nom[NAME_SIZE];
  char proto[PROTO_SIZE];
  // appel des fonctions specifiques aux requetes
   switch ( req.codeReq ) {
      case AJOUT : printf("Serveur : ajout\n");
        (*rep).codeRep = ajoutTable( req.specific.ajout );
        break;
     case ATTRIB :  printf("Serveur : attrib\n");
        (*rep).codeRep = attribPort( req.specific.attrib, &port );
        if ( (*rep).codeRep == ERR_OK ) {
          (*rep).specific.attrib.port = port;}
        break;
     case CONSULT_NOM : printf("Serveur : consult_nom\n");
        (*rep).codeRep = consultNom( req.specific.consultNom, proto, &port );
        if ( (*rep).codeRep == ERR_OK ) {
          strcpy( (*rep).specific.consultNom.proto, proto );
          (*rep).specific.consultNom.port = port;}
        break;
     case CONSULT_PORT : printf("Serveur : consult_port\n");
        (*rep).codeRep = consultPort( req.specific.consultPort, nom, proto );
        if ( (*rep).codeRep == ERR_OK ) {
          strcpy( (*rep).specific.consultPort.nom, nom );
          strcpy( (*rep).specific.consultPort.proto, proto );}
        break;
     case SUPPRESS : printf("Serveur : suppress\n");
        (*rep).codeRep = suppressTable( req.specific.suppress );
        break;
     case FIN : printf("Serveur : fin \n");
        break;
     default :
        printf("serveur : erreur, operateur inconnu\n");
        (*rep).codeRep = ERR_UNKNOWN_REQUEST; }
}
```

```c
// ----------------------------------------------------
// Fonction de traitement d'une requete de consultation par port
// ----------------------------------------------------
TErrorCode consultPort( TConsultPortReq consultPort , char* nom, char* proto)
{
  int i;
  TErrorCode retour;

  i = getIndiceByPort( consultPort.port );
  if ( i == NOT_FOUND ) { // Port doesn't exist

    retour = ERR_UNKNOWN_PORT;

  } else {

    // recopie des infos
    strcpy( nom, serviceTable[ i ].nom );
    strcpy( proto, serviceTable[ i ].proto );


    retour = ERR_OK;
  }

  return retour;
}
```

```c
    // ----------------------------------------------------
    // Fonction de traitement d'une requete de suppression
    // ----------------------------------------------------
    TErrorCode suppressTable( TSuppressReq suppress )
    {
      int i;
      TErrorCode retour;

      i = getIndiceByName( suppress.nom );
      if ( i == NOT_FOUND ) { // Name doesn't exist

        retour = ERR_UNKNOWN_NAME;

      } else {

        // suppression service
        serviceTable[ i ].state = FREE;

        // Inutile... mais plus sur
        strcpy( serviceTable[ i ].nom, " " );
        strcpy( serviceTable[ i ].proto, " " );
        serviceTable[ i ].port = 0;

        retour = ERR_OK;
      }

      return retour;
    }
```

```c
  // Gestion des requetes
  for ( i = 0 ; i < nbConnect ; i++ ) {
      if ( FD_ISSET( sockTrans[ i ], &readSet )) {
  /* Reception du message */
  err=recv(sockTrans[ i ], (void *) &req, sizeof( req) , 0);
  if ( err < 0 ) {
     printf("serveur: Erreur %d dans le recv pour %d\n", errno, i);
    shutdown(sockTrans[ i ], 2);
    close(sockTrans[ i ]);

  }
  traitReq( req, &rep );
  // Pour la fin on ne retourne pas de reponse
  if ( req.codeReq == FIN ) {

    printf("Serveur : fin client %d \n", i);
    shutdown(sockTrans[ i ], 2);
    close(sockTrans[ i ]);

    nbConnect--;

  } else {

    err = send( sockTrans[ i ], (void *) &rep , sizeof( rep ), 0 );
    if ( err != sizeof( rep ) ) {

      perror("serveur : erreur dans l'envoi du resultat");
      shutdown(sockTrans[ i ], 2);
      close(sockTrans[ i ]);
    }
  }
    }//premier if du for
  } // fin descripteurs de transmission
} // fin boucle for(;;)
return 0;
}
```

```c
// Fonction principale
// ------------------------------------------------
int main( int argc, char** argv, char** envp )
{  int sock_cont,                 /* descipteur de socket de connexion */
      sockTrans[MAX_CONNECT],     /* descipteurs des sockets locales */
      nbConnect,                  /* nombre de connexions en cours */
      err;                        /* code d'erreur */
  struct sockaddr_in nomTrans;    /* adresse de la socket de transmission */
  TRequest req;                   /* Requete recue */
  TReply   rep;                   /* Reponse envoyee */
  socklen_t   sizeAddrTrans;     /* taille de l'adresse d'une socket */
   /* Variables pour le select */
  fd_set readSet;
  /* Variables d'iteration */
  int i;
    /* Initialisations */
  nbConnect = 0;
  initServiceTable();

  if ( argc != 2 ) {printf ( "usage : serveur no_port\n" );exit( 1 );}

  /** Creation de la socket, protocole TCP */
  sock_cont = socketServeur_EAD( atoi( argv[1] ) );
  if ( sock_cont < 0 ) {
     printf( "serveur : erreur socketServeur\n" );
    exit( 2 ); }


  /* Boucle principale */
  for (;;) {

    /* Preparation du fd_set */
    FD_ZERO( &readSet );

    FD_SET( sock_cont, &readSet );
    /* Positionne l'ensemble des descripteurs */
    for ( i = 0 ; i < nbConnect ; i++ ) {
      FD_SET( sockTrans[ i ], &readSet );
    }

    err = select ( FD_SETSIZE, &readSet, NULL, NULL, NULL );
    if (err<0) {
      perror("serveur : Error in select");
    }

    /* Test du descripteur de connexion */
    if ( FD_ISSET( sock_cont, &readSet )) {
        // Gestion des nouvelles demandes de connexion
      if ( nbConnect < MAX_CONNECT ) {
    /* Acceptation d'une demande de connexion */
    sizeAddrTrans = sizeof( struct sockaddr_in );
    sockTrans[ nbConnect ] = accept (sock_cont,
                     (struct sockaddr *) &nomTrans,
                     &sizeAddrTrans);
         if ( sockTrans[ nbConnect ] < 0 ) {
           printf("serveur: Erreur %d sur le accept1\n", errno);
         } else {
            printf("serveur : connexion de %d \n", nbConnect );
           nbConnect++;
          }
      } else {printf("serveur : plus de connexion disponibles \n");}

    } /* Fin demandes de connexion */
```

```
christophe@christophe-asus:~/CSD/exo/ex18/qu1$ ls
client  client.c  Makefile  protocole.h  README  serveur  serveur.c  serveur.h
christophe@christophe-asus:~/CSD/exo/ex18/qu1$ ./serveur
usage : serveur no_port
christophe@christophe-asus:~/CSD/exo/ex18/qu1$ ./serveur  5000
serveur : creation de la socket sur 5000
serveur : connexion de 0
Serveur : ajout
Serveur : attrib
Serveur : consult_nom
Serveur : suppress
Serveur : fin
Serveur : fin client 0
```

```
christophe@christophe-asus: ~/CSD/exo/ex18/qu1
File  Edit  View  Search  Terminal  Help
christophe@christophe-asus:~/CSD/exo/ex18/qu1$ ./client localhost 5000
client : connect to localhost, 5000
client : envoi ajout realise
Client : request correctly executed
client : envoi attrib realise
Client : port attribue 2609
client : envoi consult realise
Client : nom = ead proto = tcp port = 2609
client : envoi suppress realise
Client : request correctly executed
client : envoi suppress realise
christophe@christophe-asus:~/CSD/exo/ex18/qu1$
```