**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.

|  |  |
| --- | --- |
| #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;  }    /\*\*\*\*\*\*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 | #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  suite1  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\*\*/    **/\*\*\*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 |























