Ex11 Communication structurée

Un serveur qui réalise les 4 opérations arithmétique (+,-,,*,/) pour le compte d'un client.
Le serveur ne traite qu'un client qui envoie des opérations élémentaires.
Le client doit envoyer plusieurs données au serveur et donc généré un message structuré pour que le serveur comprenne.
1ier solution requête sous forme textuelle 4+5 mais lourd à gérer sur le serveur on doit faire une analyse syntaxique du message reçu.
2ième solution envoyé les données les unes après les autres et les recevoir sous cette forme.et on peut faire un envoi en 1 seule fois dans une structure et le serveur doit les recevoir de la même manière.

Envoyer les données à la suite c'est à dire char operateur et int opérande1, opérande2. On demande une saisie de ces données et on envoie l'opérateur &operateur puis &operande1 ensuite &operande2 tous avec send .
Le serveur char operateur, int operande1, int operande2 on réceptionne &opérateur ensuite &operande1 et &operande2 sur chaque recv ensuite dans un switch(operateur) +, -,*,/
on place dans resultat = operand1+operand2 et en envoie au client &resultat avec send et le Client reçoit le résultat.

Pour des opération 3+5 mais pour (3*5)+(34/2)? On peut envoyer plusieurs opérations à réaliser
Soit en faisant autant de connexion que de demande d'opération. Soit en maintenant la connexion
Mais comment permettre à un client d'envoyer un nombre quelquonque d'opération à effectuer et de terminer sans affecter le serveur ?
Il faut maintenir la connexion tant que le client envoie des opérations donc il faut gérer un code fin qui peut être réaliser avec un code de fin de connexion.
[cette fin peut être réalisée avec un code d'opération spécifique retour à 0 de la fonction recv]
Le client a une boucle do{...voulez vous continuer=0} while (boucle == '0') donc le client ferme la connexion.
Cotés serveur une variable encore=1 avec une boucle do while (encore ==1) et au premier recv &operateur comme on a fermer la connexion recv=err=0 on a un if(err=0) encore=0 else on continu dans la boucle. Donc on sort de la boucle et on finie le serveur s'arrête
Explication forum?:

Le **serveur traite plusieurs clients**, adapter le serveur pour qu'il soit accessible après la fin de l'exécution du client et serve le second client: on ajoute une boucle infinie:
Ajouter une boucle infinie for(;;;)juste avant l'attente de connexion sock_trans = accept(sock_conn, NULL,NULL); et on fini apres le close(sock_trans)et avant le return 0 du main.

Ici le client1 exécute les calculs mais pour que le client2 a une réponse il faut que le client1 se déconnecte et ensuite le client2 reçoit sa réponse: puisque le serveur recv bloque et le client1 attend une réponse à chaque fois et que le client2 doit attendre la fin du client1

Pour ses opérations on a des cas d'erreur comme la division par 0 on peut ajouter l'envoi d'un entier avant le résultat pour indiquer que l'opération c'est bien passé ou qu'il y a une erreur.
Côté serveur on a int ok=0; si (operateur == / && operande2==0) alors ok=1 sinon on fait l'opération.
Et on envoi send &ok au client.
Côté client au niveau de la réception du résultat on recv d'abord &ok et si(ok==0) on reçoit le résultat sinon on traite ok dans un switch case 1 division par zero donc une erreur …

Q11.7 le serveur traite plusieurs client de sorte qu'il puisse (adaptation) servir d'autres clients avant que le premier ne soit terminé

Sockets non-bloquantes et avec la fonction select

```c
#include <stdio.h> #include <string.h>
#include <stdlib.h> #include <unistd.h>
 #include "fonctionsSocket.h"  CLIENT

int main(int argc, char *argv[]) {
  int sock, err ;
  char boucle = 'o';
  char operateur;
  int operande1, operande2;
  int ok;
  int resultat;
   if (argc != 3) {printf("client nom_machine no_port\n"); exit(1);}
  sock = socketClient_EAD(argv[1], atoi(argv[2]));
 if (sock < 0) { printf( "erreur socketClient_ead\n");  exit(2);  }
  do {
   printf("\t donner un operateur : ");
   scanf(" %c", &operateur);
    err = send(sock, &operateur, sizeof(operateur), 0);
  if (err != sizeof(operateur)) { perror("err sur le send op");
    shutdown(sock, 2);  exit(3);   }
     scanf(" %d", &operande1);
    err = send(sock, &operande1, sizeof(operande1), 0);
  if (err != sizeof(operande1)) {perror("err sur le send opd1");
    shutdown(sock, 2);   exit(4);   }
   scanf(" %d", &operande2);
    err = send(sock, &operande2, sizeof(operande2), 0);
  if (err != sizeof(operande2)) {perror("err sur le send opd2");
    shutdown(sock, 2);   exit(5);   }
   printf("client: envoi realise\n");
   /* * Reception du résultat    */
  err = recv(sock, &ok, sizeof(ok), 0);
  if (err < 0) { perror("err a la reception");
 shutdown(sock, 2);   exit(6);   }

  if (ok == 0) {
     err = recv(sock, &resultat, sizeof(resultat), 0);
    if (err < 0) {   perror("client: erreur a la reception");
     shutdown(sock, 2);    exit(7);    }
    printf("client: resultat recu : %d\n", resultat);
   } else {
    switch (ok) {
      case 1:printf("client: operation illegale : division 0\n");
        break;
      case 2: printf("client: operation illegale : operatio\n");
        break;
      default: printf("client: operation illegale : erreur\n");
        break;
      }
    }   /** On continue ?  */
  printf("client: on continue = o : ");
   // Attention, l'espace avant le %c permet de vider le
buffer
  // du caractere de nouvelle ligne de la saisie precedente
  scanf(" %c", &boucle);
  printf("\n");

 } while (boucle == 'o');
  /*   * Fermeture de la connexion et de la socket   */
 shutdown(sock, 2);
 close(sock);
  return 0;
```

```c
#include <string.h> #include <stdio.h>
#include <stdlib.h> #include <unistd.h>
#include <errno.h> #include <sys/select.h>
#include <sys/socket.h> #include <netinet/in.h>
 #include "fonctionsSocket.h"  #define MAX_CLIENTS 10

int main(int argc, char *argv[]) {
 int sock_cont, sock_trans[MAX_CLIENTS] , err ;
 int nfds;
 int i;
 int etape[MAX_CLIENTS];
 char operateur[MAX_CLIENTS];
 int operande1[MAX_CLIENTS], operande2[MAX_CLIENTS];
 int resultat;
  fd_set set;
  if (argc != 2) {printf ("serveur no_port\n");exit(1);  }
  sock_cont = socketServeur_EAD(atoi(argv[1]));
 if (sock_cont < 0) {printf("err socketServeur_EAD\n"); exit( 2 ); }

 for (i = 0; i < MAX_CLIENTS; ++i) {
   etape[i] = -1; //initialise tableau à -1
 }
for (;;;){/** Boucle du serveur */Reconstitution ensemble
   FD_ZERO(&set);
   FD_SET(sock_cont, &set);
   nfds = sock_cont;

      for (i = 0; i < MAX_CLIENTS; ++i) {
              if (etape[i] == -1) { continue;  }
           FD_SET(sock_trans[i], &set);
            if (sock_trans[i] > nfds) { nfds = sock_trans[i]; }
       }       /** Attente de connexion    */
  err = select(nfds + 1, &set, NULL, NULL, NULL);

  if (FD_ISSET(sock_cont, &set)) {
      printf("serveur: nouvelle connexion\n");
       /* Recherche d'un slot libre */
      for (i = 0; i < MAX_CLIENTS; ++i) {
          if (etape[i] == -1) {break;}
       }
      if (i == MAX_CLIENTS) { /* Aucun slot libre */
         fprintf(stderr, "serveur: aucun slot libre, arret!\n");
        exit(9);
       }
      sock_trans[i] = accept(sock_cont, NULL, NULL);
       if (sock_trans[i] < 0) {perror(" erreur sur accept");
          exit(3);
        }
      printf("serveur: attribution du slot %i\n", i);
      etape[i] = 0;
  } else {
    for (i = 0; i < MAX_CLIENTS; ++i) {
       if (etape[i] == -1) {   continue;    }

     if (!FD_ISSET(sock_trans[i], &set)) {
       continue;
     }
```

```
}
```

```
root@christophe-asus:/home/christophe/CSD/exo/ex11# ./serveur5 5000
serveur: creation de la socket sur 5000
serveur: nouvelle connexion
serveur: attribution du slot 0
serveur: nouvelle connexion
serveur: attribution du slot 1
serveur: reception sur la connexion 1 (etape 0)
serveur: reception sur la connexion 0 (etape 0)
serveur: reception sur la connexion 0 (etape 1)
serveur: reception sur la connexion 1 (etape 1)
serveur: reception sur la connexion 1 (etape 2)
serveur: voila l'operation recue par 1 : 4 + 4
serveur: reception sur la connexion 0 (etape 2)
serveur: voila l'operation recue par 0 : 3 - 2
```

```
root@christophe-asus: /home/christophe/CSD/exo/ex11
File  Edit  View  Search  Terminal  Help
oot@christophe-asus:/home/christophe/CSD/exo/ex11# ./client5 localhost 5000
lient:
        donner un operateur : -
        donner l'operande 1 : 3
        donner l'operande 2 : 2
lient: envoi de - 3 - 2 -
lient: envoi realise
lient: resultat recu : 1
lient: on continue = o :
```

```
root@christophe-asus: /home/christophe/CSD/exo/ex11
File  Edit  View  Search  Terminal  Help
oot@christophe-asus:/home/christophe/CSD/exo/ex11# ./client5 localhost 5000
lient:
        donner un operateur : +
        donner l'operande 1 : 4
        donner l'operande 2 : 4
lient: envoi de - 4 + 4 -
lient: envoi realise
lient: resultat recu : 8
lient: on continue = o :
```

```
printf("serveur: reception sur la connexion %i (etape %i)\n", i,
                                                        etape[i]);
      /* La i-ème connexion est active */
     if (etape[i] == 0) {
       /* réception de l'opérateur */
       err = recv(sock_trans[i], &operateur[i],
                                  sizeof(operateur[i]), 0);
       if (err < 0) {perror("err dans la reception d'operateur");
       shutdown(sock_trans[i], 2);       exit(4);       }

       if (err == 0) {
        close(sock_trans[i]);
         etape[i] = -1;
        printf("serveur: fermeture de la connexion %i\n", i);
         break;
       }

       etape[i] = 1;
     } else
     if (etape[i] == 1) {
       /* réception du premier opérande */
       err = recv(sock_trans[i], &operande1[i],
                                  sizeof(operande1[i]), 0);
       if (err < 0) {perror("err dans la reception d'operande1");
       shutdown(sock_trans[i], 2);     exit(5);     }
       etape[i] = 2;
     } else
     if (etape[i] == 2) {
          /* réception second opérande */
          err = recv(sock_trans[i], &operande2[i],
                                  sizeof(operande2[i]), 0);
       if (err < 0) {perror("err dans la reception d'operande2");
       shutdown(sock_trans[i], 2);       exit(6);       }
       /* et envoi du résultat */
        int ok = 0;
       if (operateur[i] == '/' && operande2[i] == 0) {
             ok = 1;
         } else {
        printf("serveur: voila l'operation recue par %i : %d %c
            %d\n", i, operande1[i], operateur[i], operande2[i]);
       switch (operateur[i]) {
       case '+' : resultat = operande1[i] + operande2[i]; break;
       case '-' : resultat = operande1[i] - operande2[i]; break;
       case '*' : resultat = operande1[i] * operande2[i]; break;
       case '/' : resultat = operande1[i] / operande2[i]; break;
    default : printf("serveur: erreur, operateur inconnu\n");ok = 2;
         }
       }
       err = send(sock_trans[i], &ok, sizeof(ok), 0);
       if (err != sizeof(ok)) { perror("err envoi de legalite de
                                                  l'operation");
       shutdown(sock_trans[i], 2);          exit(7);        }
     err = send(sock_trans[i], &resultat, sizeof(resultat), 0);
       if (err != sizeof(resultat)) { perror("err dans l'envoi du
                                                      resultat");
       shutdown(sock_trans[i], 2);     exit(8);        }

       etape[i] = 0;
     }
    } //for
   } //else for
  } //for( ;;;)
 return 0;
}
```