UNIVERSIDAD
Panamericana

Materia: Cómputo Distribuido

Facilitador: Dr. Juan Carlos López Pimentel

Alumnos:

Rosita Aguirre Plasencia

Eduardo Sebastián González Ramírez

Celia Lucía Castañeda Arizaga

Práctica: Comunicación con sockets TCP

Fecha: 20/02/2023

Código del Servidor:

```
#include <stdio.h>
/* The following headers was required in old or some compilers*/
//#include <sys/types.h>
//#include <sys/socket.h>
//#include <netinet/in.h>
#include <netdb.h>
#include <signal.h>   // it is required to call signal handler functions
#include <unistd.h>  // it is required to close the socket descriptor
#include "calculator.h"

#define  DIRSIZE   2048      /* longitud maxima parametro entrada/salida */
#define  PUERTO   15000           /* numero puerto arbitrario */

int                  sd, sd_actual;  /* descriptores de sockets */
int                  addrlen;        /* longitud direcciones */
struct sockaddr_in   sind, pin;      /* direcciones sockets cliente u servidor */


/*  procedimiento de aborte del servidor, si llega una senal SIGINT */
/* ( <ctrl> <c> ) se cierra el socket y se aborta el programa       */
void aborta_handler(int sig){
   printf("....abortando el proceso servidor %d\n",sig);
   close(sd);
   close(sd_actual);
   exit(1);
}


int main(){

       char  dir[DIRSIZE];        /* parametro entrada y salida */

       /*
       When the user presses <Ctrl + C>, the aborta_handler function will be called,
       and such a message will be printed.
       Note that the signal function returns SIG_ERR if it is unable to set the
       signal handler, executing line 54.
       */
   if(signal(SIGINT, aborta_handler) == SIG_ERR){
       perror("Could not set signal handler");
      return 1;
   }
       //signal(SIGINT, aborta);      /* activando la senal SIGINT */

/* obtencion de un socket tipo internet */
       if ((sd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
              perror("socket");
              exit(1);
       }

/* asignar direcciones en la estructura de direcciones */
       sind.sin_family = AF_INET;
       sind.sin_addr.s_addr = INADDR_ANY;   /* INADDR_ANY=0x000000 = yo mismo */
```

```
        sind.sin_port = htons(PUERTO);       /*  convirtiendo a formato red */

/* asociando el socket al numero de puerto */
        if (bind(sd, (struct sockaddr *)&sind, sizeof(sind)) == -1) {
                perror("bind");
                exit(1);
        }

        printf("Conexion abierta\n");

/* ponerse a escuchar a traves del socket */
        if (listen(sd, 5) == -1) {
                perror("listen");
                exit(1);
        }

/* esperando que un cliente solicite un servicio */
        if ((sd_actual = accept(sd, (struct sockaddr *)&pin, &addrlen)) == -1) {
                perror("accept");
                exit(1);
        }

/* tomar un mensaje del cliente */
        if (recv(sd_actual, dir, sizeof(dir), 0) == -1) {
                perror("recv");
                exit(1);
        }

/* leyendo el directorio */
        calculate(dir);

/* enviando la respuesta del servicio */
        if ( send(sd_actual, dir, strlen(dir), 0) == -1) {
                perror("send");
                exit(1);
        }

/* cerrar los dos sockets */
        close(sd_actual);
        close(sd);
        printf("Conexion cerrada\n");
        return 0;
}
```

Código del Cliente:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <unistd.h>  // it is required to close the socket descriptor
#include <netdb.h>

#define  DIRSIZE    2048   /* longitud maxima parametro entrada/salida */
#define  PUERTO    15000   /* numero puerto arbitrario */
#define  MSGSIZE    2048   /* longitud de los mensajes */

int main(argc, argv)
   int    argc;
   char  *argv[];
{
       char                dir[DIRSIZE];  /* parametro entrada y salida */
       int                 sd;                /* descriptores de sockets    */
       struct hostent           *hp;          /* estructura del host        */
       struct sockaddr_in sin, pin;    /* direcciones socket        */
    int                *status;        /* regreso llamada sistema */
    char               *host;          /* nombre del host */

/* verificando el paso de parametros */

        if ( argc != 3) {
           fprintf(stderr,"Error uso: %s <host> <archivo> \n",argv[0]);
           exit(1);
        }
        host = argv[1];

/* encontrando todo lo referente acerca de la maquina host */

       if ( (hp = gethostbyname(host)) == 0) {
              perror("gethosbyname");
              exit(1);
       }

/* llenar la estructura de direcciones con la informacion del host */
       pin.sin_family = AF_INET;
       pin.sin_addr.s_addr = ((struct in_addr *) (hp->h_addr))->s_addr;
       pin.sin_port = htons(PUERTO);

/* obtencion de un socket tipo internet */
       if ( (sd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
              perror("socket");
              exit(1);
       }

/* conectandose al PUERTO en el HOST  */
       if ( connect(sd, (struct sockaddr *)&pin, sizeof(pin)) == -1) {
              perror("connect");
```

```
                exit(1);
        }

/* enviar mensaje al PUERTO del  servidor en la maquina HOST */
        FILE *file_mng = fopen(argv[2], "r"); // "r" for read
    fgets(dir , DIRSIZE , file_mng);
        fclose(file_mng);
        if ( send(sd, dir, sizeof(dir), 0) == -1 ) {
                perror("send");
                exit(1);
        }

/* esperar por la respuesta */
        if ( recv(sd, dir, sizeof(dir), 0) == -1 ) {
                perror("recv");
                exit(1);
        }

/* imprimimos el resultado y cerramos la conexion del socket */
        printf("El resultado de la operacion mandada es: %s \n", dir);
        close(sd);
        return 0;
}
```

Función Calculadora:

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>

#define SIZE 21
char *inOP[SIZE] = {0};
int k = 0;

typedef struct stack
{
       int data;
       struct stack *next;
} stack;

typedef struct exp {
    char op;
    char *term;
    struct exp *left;
    struct exp *right;
} Exp;

Exp *make_exp2(char *str){
    if(!str || !*str) return NULL;//*str == '\0' is format error.
    char *mul = strrchr(str, '*');
    char *div = strrchr(str, '/');
    Exp *node = malloc(sizeof(*node));
    if(mul == NULL && div == NULL){
        node->op = '\0';
        node->term = str;
        node->left = node->right = NULL;
        return node;
    }
    char *op;
    op = mul < div ? div : mul;
    node->op = *op;
    *op = '\0';
    node->left  = make_exp2(str );
    node->right = make_exp2(op+1);
    return node;
}

Exp *make_exp(char *str){
    if(!str || !*str) return NULL;//*str == '\0' is format error.
    char *minus = strrchr(str, '-');
    char *plus  = strrchr(str, '+');
    if(minus == NULL && plus == NULL)
        return make_exp2(str);
    char *op;
    Exp *node = malloc(sizeof(*node));
    op = minus < plus ? plus : minus;
    node->op = *op;
    *op = '\0';
    node->left  = make_exp(str );
```

```
    node->right = make_exp(op+1);
    return node;
}


void print(Exp *exp, int level){
    int i;
    if(exp->op){
        inOP[k] = &exp->op;
        k = k+1;
        print(exp->right, level+1);
        print(exp->left, level+1);
    } else {
        inOP[k] = exp->term;
        k = k+1;
    }
}


int calculate2(int a, char *bC, char op){
  int b = (int)strtol(bC, (char **)NULL, 10);
  switch(op){
    case '+':
      return a+b;
    case '-':
      return a-b;
    case '*':
      return a*b;
    case '/':
      return a/b;
  }
  return 0;
}

void push( stack **head, int value )
{
       stack* node = malloc( sizeof(stack) );

       if( node == NULL ) {
              fputs( "Error: Out of memory\n", stderr );
              exit( 1 );
       } else {
              node->data = value;
              node->next = *head;
              *head = node;
       }
}

int pop( stack **head )
{
       if( *head == NULL ) {
              fputs( "Error: bottom of stack!\n", stderr );
              exit( 1 );
       } else {
              stack* top = *head;
              int value = top->data;
              *head = top->next;
```

```
                free( top );
                return value;
        }
}

int eval( char op, stack** head )
{
        int temp;
        switch( op ) {
                case '+': return pop(head) + pop(head);
                case '*': return pop(head) * pop(head);
                case '-': temp = pop(head); return pop(head) - temp;
                case '/': temp = pop(head); return pop(head) / temp;
        }
}

int need( char op )
{
        switch( op ) {
                case '+':
                case '*':
                case '-':
                case '/':
                        return 2;
                default:
                        fputs( "Invalid operand!", stderr );
                        exit( 1 );
        }
}

int checknr( char* number )
{
        for( ; *number; number++ )
                if( *number < '0' || *number > '9' )
                        return 0;

        return 1;
}

//Se creo la funcion itoa() porque esta funcion no viene incorporada dentro de las
funciones que
//maneja linux, nos sirve para convertir un entero a una cadena.
char *itoa(int n)
{
  static char   buf[32];
  sprintf(buf,"%d ",n);
  return  buf;
}

void calculate(char *e){
  int i, temp, stacksize = 0;
        stack* head = NULL;

  //char str[] = "3+1-4*6-7";
  char str[strlen(e) + 1];
  strcpy(str, e);
```

```
  int length = strlen(str);
  Exp *exp = make_exp(str);
  print(exp, 0);

       for( i = k-1; i >= 0; i--) {
              char* token = inOP[i];
              char* endptr;
              char op;

              if( checknr( token ) ) {
                     /* We have a valid number. */
                     temp = atoi( token );
                     push( &head, temp );
                     ++stacksize;
              } else {
                     /* We have an operand (hopefully) */
                     if( strlen( token ) != 1 ) {
                            fprintf( stderr, "Error: Token '%s' too large.\n", token );
                            exit( 1 );
                     }

                     op = token[0];

                     if( stacksize < need( op ) ) {
                            fputs( "Too few arguments on stack.\n", stderr );
                            exit( 1 );
                     }

                     push( &head, eval( op, &head ) );
                     stacksize -= need( op ) - 1;
              }
       }

       if( stacksize != 1 ) {
              fputs( "Too many arguments on stack.\n", stderr );
              exit( 1 );
       }

       printf( "Result: %i\n", head->data );
    strcpy(e,itoa(head->data));
       //return head->data;
}
```

Ejecución del Código:

- Servidor:

```
PS D:\rfmrm\Documentos\ISGC\8 Semestre\Computo Distribuido> docker exec -it ubnt-cont-1 bash
root@cd5a14d3bac6:/# cd ./distributed_computing/P2_Servidor_TCP/codes
root@cd5a14d3bac6:/distributed_computing/P2_Servidor_TCP/codes# cc tcpserver.c -lnsl -o tcpserver
root@cd5a14d3bac6:/distributed_computing/P2_Servidor_TCP/codes# ./tcpserver
Conexion abierta
Result: 5
Conexion cerrada
```

- Cliente:

```
PS D:\rfmrm\Documentos\ISGC\8 Semestre\Computo Distribuido> docker exec -it ubnt-cont-1 bash
root@cd5a14d3bac6:/# cd ./distributed_computing/P2_Servidor_TCP/codes
root@cd5a14d3bac6:/distributed_computing/P2_Servidor_TCP/codes# cc tcpclient.c -lnsl -o tcpclient
root@cd5a14d3bac6:/distributed_computing/P2_Servidor_TCP/codes# ./tcpclient 0 ./operations.txt
El resultado de la operacion mandada es: 5
```

- Operations.txt: (Archivo de lectura de operaciones)

```
1  3*5-10*1
```