Universidad Carlos III
Curso Sistemas Distribuidos 2023-24
Ejercicio evaluable 3

Rubén de Arriba Viejo (100471975)
Óscar Hontoria Herrador (100471920)

Índice

1. Introducción

Este documento presenta los razonamientos y pasos seguidos para implementar el diseño y la implementación del servicio distribuido para el almacenamiento de tuplas. En este sistema, uno o más clientes mediante el uso de RPC, podrán acceder a las tuplas almacenadas en el servidor, pudiendo realizar distintas operaciones sobre ellas. Las tuplas se componen de <clave-valor1-valor2>.

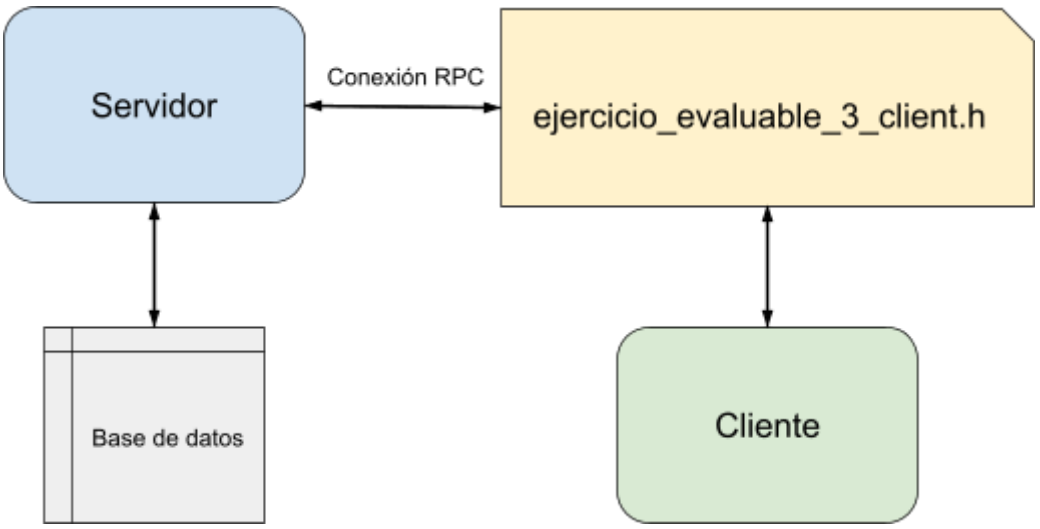
Para hacerlo se ha implementado un servidor para la gestión de las claves almacenadas, un cliente para solicitar operaciones y una biblioteca dinámica que define las operaciones.

2. Componentes

Como el requisito de esta práctica era utilizar RPC, hemos generado el fichero ejercicio_evaluable_3.x, que contiene el código XDR que genera las funciones correspondientes y la estructura que se utiliza para el paso de mensajes. Tras ejecutar este fichero se generan varios ficheros: Makefile.ejercicio_evaluable_3 (Fichero de compila el proyecto), ejercicio_evaluable_3_xdr.c (Fichero que se encarga de la comunicación RPC), ejercicio_evaluable_3_client.c (Fichero que actúa como claves.c), ejercicio_evaluable_3_clnt.c (Fichero que se encarga de llamar a las comunicaciones RPC de ejercicio_evaluable_3_client.c), ejercicio_evaluable_3_server.c (Fichero que es el servidor propiamente dicho), ejercicio_evaluable_3_svc.c (Fichero que se encarga de llamar a las comunicaciones RPC de ejercicio_evaluable_3_server.c) y el fichero ejercicio_evaluable_3.h (Fichero que tiene definidas todas las funciones de comunicación y estructuras definidas en el fichero ejercicio_evaluable_3.x para que el resto de ficheros las utilicen).

Adicionalmente, hemos añadido dos ficheros: client.c (Fichero que es el cliente propiamente dicho) y ejercicio_evaluable_3_client.h (Fichero idéntico a claves.h de los anteriores ejercicios).

Con estos ficheros, y haciendo modificaciones que se especifican en el siguiente punto de la memoria, el sistema estaría listo. Se ha decidido que la comunicación sea de tipo TCP ya que, al estar almacenando y solicitando información a un servidor, la conexión debe ser segura.



3. Compilación y ejecución

Sin haber ejecutado el fichero ejercicio_evaluable_3.x, el directorio únicamente tiene los ficheros ejercicio_evaluable_3.x, ejercicio_evaluable_3_client.h y cliente.c.

Al ejecutar el fichero ejercicio_evaluable_3.x (rpcgen -Ma ejercicio_evaluable_3.x), se generan todos los ficheros para que funcionen las conexiones RPC. La M en la compilación indica la creación de hilos para que el servidor sea concurrente, pero dependiendo del sistema operativo que se utilice, el servidor no será concurrente.

Para que el programa se pueda ejecutar correctamente, se deben modificar los ficheros generados:

- **Fichero ejercicio_evaluable_3.x:**

  En este fichero se debe eliminar la definición de la variable *register int32_t *buf*, y el fichero ya estaría listo

- **Fichero ejercicio_evaluable_3_client.c:**

  En este fichero se deben poner las funciones dentro del fichero ejercicio_evaluable_3_client.h y poner en ellas cómo se pasa de valores recibidos por la función a la estructura que se desea enviar. El fichero quedaría de la siguiente manera:

```
#include "ejercicio_evaluable_3.h"
int init()
{
   CLIENT *clnt;
   enum clnt_stat retval_1;
   int result_1;
   char *innit_1_arg;
#ifndef    DEBUG
          char *host = getenv("IP_TUPLAS");
          if (host == NULL) {
          fprintf(stderr, "Error: La variable de entorno 'IP_TUPLAS' no está definida.\n");
          exit(1);
          }
   clnt = clnt_create (host, INNIT_PROG, INNIT_VERSION, "tcp");
   if (clnt == NULL) {
           clnt_pcreateerror (host);
           exit (1);
   }
#endif    /* DEBUG */
   retval_1 = innit_1((void*)&innit_1_arg, &result_1, clnt);
   if (retval_1 != RPC_SUCCESS) {
           clnt_perror (clnt, "call failed");
          clnt_destroy (clnt);
   }
```

```
        return result_1;
#ifndef    DEBUG
#endif     /* DEBUG */
}
int set_value(int key, char* val_1, int n_elem, double* vector)
{
   CLIENT *clnt;
   enum clnt_stat retval_1;
   int result_1;
   struct mensaje  set_value_2_arg;
#ifndef    DEBUG
        char *host = getenv("IP_TUPLAS");
        if (host == NULL) {
        fprintf(stderr, "Error: La variable de entorno 'IP_TUPLAS' no está
definida.\n");
        exit(1);
        }
   clnt = clnt_create (host, SET_VALUE_PROG, SET_VALUE_VERSION, "tcp");
   if (clnt == NULL) {
         clnt_pcreateerror (host);
         exit (1);
   }
#endif    /* DEBUG */
        set_value_2_arg.key = key;
        set_value_2_arg.val_1 = (char*)malloc(sizeof (val_1)* sizeof(char));
        strcpy(set_value_2_arg.val_1, val_1);
        set_value_2_arg.vector.vector_val      =      (double*)malloc(n_elem      *
sizeof(double));
        set_value_2_arg.vector.vector_len = n_elem;
        for (int i = 0; i < set_value_2_arg.vector.vector_len; i++) {
        set_value_2_arg.vector.vector_val[i] = vector[i];
        }
        retval_1 = set_value_2(&set_value_2_arg, &result_1, clnt);
        if (retval_1 != RPC_SUCCESS) {
        clnt_perror (clnt, "call failed");
        clnt_destroy (clnt);
        }
        return result_1;
#ifndef    DEBUG
#endif     /* DEBUG */
}
int get_value(int key, char *value1, int *N_value2, double *V_value2)
{
   CLIENT *clnt;
   enum clnt_stat retval_1;
   struct mensaje result_1;
   int  get_value_3_arg;
#ifndef    DEBUG
        char *host = getenv("IP_TUPLAS");
        if (host == NULL) {
```

```
        fprintf(stderr, "Error: La variable de entorno 'IP_TUPLAS' no está definida.\n");
        exit(1);
        }
   clnt = clnt_create (host, GET_VALUE_PROG, GET_VALUE_VERSION, "tcp");
   if (clnt == NULL) {
        clnt_pcreateerror (host);
        exit (1);
   }
#endif    /* DEBUG */
        get_value_3_arg = key;
        result_1.val_1 = (char *)malloc(256 * sizeof(char));
        result_1.vector.vector_val = (double *)malloc(32 * sizeof(double));
        retval_1 = get_value_3(&get_value_3_arg, &result_1, clnt);
        if (retval_1 != RPC_SUCCESS) {
        clnt_perror (clnt, "call failed");
        clnt_destroy (clnt);
        }
        if(result_1.key != -1){
        key =  get_value_3_arg;
        strcpy(value1, result_1.val_1);
        N_value2 = (int*)&result_1.vector.vector_len;
        for(int i = 0; i < result_1.vector.vector_len; i++){
        V_value2[i] = result_1.vector.vector_val[i];
        }
        return 0;
        }
        return -1;
#ifndef    DEBUG
#endif     /* DEBUG */
}
int modify_value(int key, char* val_1, int n_elem, double* vector)
{
   CLIENT *clnt;
   enum clnt_stat retval_1;
   int result_1;
   struct mensaje  modify_value_4_arg;
#ifndef    DEBUG
        char *host = getenv("IP_TUPLAS");
        if (host == NULL) {
        fprintf(stderr, "Error: La variable de entorno 'IP_TUPLAS' no está definida.\n");
        exit(1);
        }
                clnt = clnt_create (host, MODIFY_VALUE_PROG, MODIFY_VALUE_VERSION, "tcp");
   if (clnt == NULL) {
        clnt_pcreateerror (host);
        exit (1);
   }
```

```
#endif    /* DEBUG */
        modify_value_4_arg.key = key;
        modify_value_4_arg.val_1 = (char*)malloc(sizeof (val_1)* sizeof(char));
        strcpy(modify_value_4_arg.val_1, val_1);
        modify_value_4_arg.vector.vector_val = (double*)malloc(n_elem *
sizeof(double));
        modify_value_4_arg.vector.vector_len = n_elem;
        for (int i = 0; i < modify_value_4_arg.vector.vector_len; i++) {
        modify_value_4_arg.vector.vector_val[i] = vector[i];
        }
        retval_1 = modify_value_4(&modify_value_4_arg, &result_1, clnt);
        if (retval_1 != RPC_SUCCESS) {
        clnt_perror (clnt, "call failed");
        return -1;
        clnt_destroy (clnt);
        }
        return result_1;
#ifndef    DEBUG
#endif     /* DEBUG */
}
int delete_key(int key)
{
   CLIENT *clnt;
   enum clnt_stat retval_1;
   int result_1;
   int  delete_key_5_arg;
#ifndef    DEBUG
        char *host = getenv("IP_TUPLAS");
        if (host == NULL) {
        fprintf(stderr, "Error: La variable de entorno 'IP_TUPLAS' no está
definida.\n");
        exit(1);
        }
      clnt = clnt_create (host, DELETE_KEY_PROG, DELETE_KEY_VERSION,
"tcp");
   if (clnt == NULL) {
         clnt_pcreateerror (host);
         exit (1);
   }
#endif    /* DEBUG */
        delete_key_5_arg = key;
   retval_1 = delete_key_5(&delete_key_5_arg, &result_1, clnt);
   if (retval_1 != RPC_SUCCESS) {
         clnt_perror (clnt, "call failed");
        clnt_destroy (clnt);
   }
        return result_1;
#ifndef    DEBUG

#endif     /* DEBUG */
```

```
}
int exist(int key)
{
   CLIENT *clnt;
   enum clnt_stat retval_1;
   int result_1;
   int  exist_6_arg;
#ifndef    DEBUG
          char *host = getenv("IP_TUPLAS");
          if (host == NULL) {
          fprintf(stderr, "Error: La variable de entorno 'IP_TUPLAS' no está
definida.\n");
          exit(1);
          }
   clnt = clnt_create (host, EXIST_PROG, EXIST_VERSION, "tcp");
   if (clnt == NULL) {
           clnt_pcreateerror (host);
           exit (1);
   }
#endif    /* DEBUG */
          exist_6_arg = key;
   retval_1 = exist_6(&exist_6_arg, &result_1, clnt);
   if (retval_1 != RPC_SUCCESS) {
           clnt_perror (clnt, "call failed");
          clnt_destroy (clnt);
   }
          return result_1;
#ifndef    DEBUG
#endif     /* DEBUG */
}
```

- **Fichero ejercicio_evaluable_server.c:**

Este fichero tiene las funciones que se realizan para el almacenamiento de las tuplas. No es necesario que tenga main porque ya existe en ejercicio_evaluable_3_svc.c.
El fichero quedaría de la siguiente manera:

```
#include "ejercicio_evaluable_3.h"
int *keys;
char **valores_1;
int* num_elements;
double **vectores;
int num_data = 0; // Numero de elementos almacenados
bool_t
innit_1_svc(void *argp, int *result, struct svc_req *rqstp)
{
          bool_t retval;
          *result = -1;
          free(valores_1);
          free(keys);
          free(vectores);
```

```
        free(num_elements);
        *result = 0;
        printf("Operación init realizada\n");
        retval = TRUE;
        return retval;
}
int
innit_prog_1_freeresult (SVCXPRT *transp, xdrproc_t xdr_result, caddr_t result)
{
   xdr_free (xdr_result, result);
   /*
    * Insert additional freeing code here, if needed
    */
   return 1;
}
bool_t
set_value_2_svc(struct mensaje *argp, int *result, struct svc_req *rqstp)
{
        bool_t retval;
        retval = TRUE;
        *result = 0;
        for(int i = 0; i < num_data; i++) {
        if (keys[i] == argp->key) {
        *result = -1;
        break;
        }
        }
        if ((argp->vector.vector_len > 32) | (argp->vector.vector_len < 1)){
        *result = -1;
        }
        if(*result != -1){
        num_data++;
        int *temp_keys = realloc(keys, num_data * sizeof(int));
        int *temp_num_elements = realloc(num_elements, num_data * sizeof(int));
        char **temp_valores_1 = NULL;
        temp_valores_1 = realloc(valores_1, num_data * sizeof(char*));
        double **tempo_vectores = realloc(vectores, num_data * sizeof(double*));
        if (temp_keys == NULL) {
        printf("Memory allocation failed\n");
        *result = -1;
        retval = FALSE;
        }
        if (temp_num_elements == NULL) {
        printf("Memory allocation failed\n");
        *result = -1;
        retval = FALSE;
        }
        if (temp_valores_1 == NULL) {
        printf("Memory allocation failed\n");
        *result = -1;
```

```
        retval = FALSE;
        }
        if (tempo_vectores == NULL) {
        printf("Memory allocation failed\n");
        *result = -1;
        retval = FALSE;
        }
        // Hacemos la capacidad de la base de datos más grande
        keys = temp_keys;
        valores_1 = temp_valores_1;
        num_elements = temp_num_elements;
        vectores = tempo_vectores;
        // Asignamos los valores al nuevo elemento de la base de datos
        keys[num_data - 1] = argp->key;
        num_elements[num_data - 1] = argp->vector.vector_len;
        valores_1[num_data - 1] = (char *)malloc((sizeof(argp->val_1) + 1) *
sizeof(char));
        strcpy(valores_1[num_data - 1], argp->val_1);
        vectores[num_data - 1] = (double *)malloc((argp->vector.vector_len) *
sizeof(double));
        for (int i = 0; i < argp->vector.vector_len; i++){
        vectores[num_data - 1][i] = argp->vector.vector_val[i];
        }
        }
        printf("Operación set_value realizada, %d\n", *result);
        return retval;
}
int
set_value_prog_2_freeresult (SVCXPRT *transp, xdrproc_t xdr_result, caddr_t result)
{
   xdr_free (xdr_result, result);
   /*
    * Insert additional freeing code here, if needed
    */
   return 1;
}
bool_t
get_value_3_svc(int *argp, struct mensaje *result, struct svc_req *rqstp)
{
        bool_t retval;
        result->key = -1;
        result->val_1 = (char *)malloc((256) * sizeof(char));
        result->vector.vector_val = (double *)malloc((32) * sizeof(double));
        for(int i = 0; i < num_data; i++) {
        if (keys[i] == *argp) {
        // Si se encuentra la clave, copiar los datos al mensaje de respuesta
        result->key = 0; // Indicador de éxito
        result->vector.vector_len = num_elements[i];
        strcpy(result->val_1, valores_1[i]);
        for(int j = 0; j < num_elements[i]; j++) {
```

```
                result->vector.vector_val[j] = vectores[i][j];
        }
        break;
        }
        }
        printf("Result (svr) = %d\n", result->key);
        printf("Operación get_value realizada\n");
        retval = TRUE;
        return retval;
}
int
get_value_prog_3_freeresult (SVCXPRT *transp, xdrproc_t xdr_result, caddr_t
result)
{
   xdr_free (xdr_result, result);
   /*
    * Insert additional freeing code here, if needed
    */
   return 1;
}
bool_t
modify_value_4_svc(struct mensaje *argp, int *result, struct svc_req *rqstp)
{
        bool_t retval;
        *result = -1;
        for (int i = 0; i < num_data; i++){
        if (argp->key == keys[i]){
        valores_1[i] = realloc(valores_1[i], sizeof(argp->val_1) * sizeof(char));
        strcpy(valores_1[i], argp->val_1);
        if (argp->vector.vector_len != num_elements[i]){
                num_elements[i] = argp->vector.vector_len;
                vectores[i] = realloc(vectores[i], sizeof(argp->vector.vector_len) *
sizeof(double));
        }
        for (int j = 0; j < argp->vector.vector_len; j++){
                vectores[i][j] = argp->vector.vector_val[j];
        }
        *result = 0;
        break;
        }
        }
        printf("Operación modify_value realizada\n");
        retval = TRUE;
        return retval;
}
int
modify_value_prog_4_freeresult (SVCXPRT *transp, xdrproc_t xdr_result, caddr_t
result)
{
   xdr_free (xdr_result, result);
```

```
    /*
     * Insert additional freeing code here, if needed
     */
    return 1;
}
bool_t
delete_key_5_svc(int *argp, int *result, struct svc_req *rqstp) {
        bool_t retval;
        int index = -1;
        retval = TRUE;

        for (int i = 0; i < num_data; i++) {
        if (keys[i] == *argp) {
        index = i;
        }
        }
        if (index == -1) {
        *result = -1;
        retval = TRUE;
        }
        else {
        free(valores_1[index]);
        free(vectores[index]);
        for (int i = index; i < num_data - 1; i++) {
        keys[i] = keys[i + 1];
        valores_1[i] = valores_1[i + 1];
        num_elements[i] = num_elements[i + 1];
        vectores[i] = vectores[i + 1];
        }
        num_data--;
        keys = realloc(keys, num_data * sizeof(int));
        valores_1 = realloc(valores_1, num_data * sizeof(char *));
        num_elements = realloc(num_elements, num_data * sizeof(int));
        vectores = realloc(vectores, num_data * sizeof(double *));
        *result = 0;
        }
        printf("Operación delete_key realizada\n");
    return retval;
}
int
delete_key_prog_5_freeresult (SVCXPRT *transp, xdrproc_t xdr_result, caddr_t
result)
{
    xdr_free (xdr_result, result);

    /*
     * Insert additional freeing code here, if needed
     */

    return 1;
```

```
}
bool_t
exist_6_svc(int *argp, int *result, struct svc_req *rqstp)
{
   bool_t retval;
         *result = 0;
         for (int i = 0; i < num_data; i++){
         if (keys[i] == *argp){
         *result = 1;
         break;
         }
         }
         retval = TRUE;
   return retval;
}
int
exist_prog_6_freeresult (SVCXPRT *transp, xdrproc_t xdr_result, caddr_t result)
{
   xdr_free (xdr_result, result);
   /*
    * Insert additional freeing code here, if needed
    */
   return 1;
}
```

- **Fichero Makefile.ejercicio_evaluable_3:**

En este fichero debemos añadir la compilación del fichero cliente.c y de la librería dinámica.
El fichero quedaría de la siguiente manera:

```
CC = gcc
CFLAGS = -Wall -g -I/usr/include/tirpc
LDLIBS += -lnsl -lpthread -ldl -ltirpc
LIB_LOCATION = ./
# Lista de archivos fuente
SERVER_SRCS = ejercicio_evaluable_3_server.c ejercicio_evaluable_3_svc.c ejercicio_evaluable_3_xdr.c
CLIENT_SRCS = ejercicio_evaluable_3_client.c ejercicio_evaluable_3_client.h ejercicio_evaluable_3_clnt.c cliente.c
# Lista de objetos a generar
SERVER_OBJS = $(SERVER_SRCS:.c=.o)
CLIENT_OBJS = $(CLIENT_SRCS:.c=.o)
# Nombre del ejecutable del servidor
SERVER_EXEC = servidor
# Nombre de la librería dinámica del cliente
CLIENT_LIB = libejercicio_evaluable_3_client.so
# Nombre del ejecutable del cliente
CLIENT_EXEC = cliente
all: $(SERVER_EXEC) $(CLIENT_LIB) $(CLIENT_EXEC)
# Compilar el ejecutable del servidor
```

```
$(SERVER_EXEC): $(SERVER_OBJS)
    $(CC) $(CFLAGS) -o $@ $^ $(LDLIBS)
# Compilar el archivo del cliente
ejercicio_evaluable_3_client.o: ejercicio_evaluable_3_client.c
ejercicio_evaluable_3_client.h ejercicio_evaluable_3_clnt.c
    $(CC) $(CFLAGS) -fPIC -c -o $@ $<
# Regla genérica para compilar archivos fuente a objetos
%.o: %.c
    $(CC) $(CFLAGS) -fPIC -c -o $@ $<
# Compilar la librería dinámica del cliente
$(CLIENT_LIB): $(CLIENT_OBJS)
    $(CC) $(CFLAGS) -shared -o $@ $^ $(LDLIBS)
# Compilar el ejecutable del cliente
$(CLIENT_EXEC): cliente.o ejercicio_evaluable_3_xdr.o
    $(CC) $(CFLAGS) -o $@ $^ -L$(LIB_LOCATION) -lejercicio_evaluable_3_client
$(LDLIBS) -Wl,-rpath=$(LIB_LOCATION)
clean:
    rm -f $(SERVER_EXEC) $(CLIENT_LIB) $(CLIENT_EXEC) $(SERVER_OBJS)
$(CLIENT_OBJS)
```

Tras modificar los ficheros de la manera explicada, se ejecutaría el makefile de la forma make -f Makefile.ejercicio_evaluable.c y el sistema estaría listo para ser ejecutado de la forma que se especifica en el enunciado de la práctica. Es posible que al copiar y pegar el código, algunas tabulaciones del Makefile se tomen como espacios, por lo que se debe verificar que se ha copiado correctamente antes de ejecutar y, si no se ha copiado correctamente, se deben cambiar los espacios por tabulaciones.

Adicionalmente, si se encuentra algún fallo a la hora de ejecutar, se pueden encontrar los ficheros con los que se ha hecho el proyecto en el siguiente enlace: https://github.com/100471920/sistemas_distribuidos/tree/main/ejercicio_evaluable_3

4. Pruebas

Para probar el programa se han ido introduciendo las operaciones seguidas de los datos en el cliente y se ha ido verificando que el resultado fuese el correcto, debido a la implementación utilizada, donde el cliente va introduciendo por terminal las diferentes operaciones, con sus datos respectivos, se invita al usuario a probar creando el cliente y enviado varias operaciones. Ejemplos:

```
$ env IP_TUPLAS=localhost ./cliente
Indique la operación a realizar: set_value
Indique la clave sobre la que se desea hacer set_value(key, valor_1, num_elements, vector): key = 1
Indique el valor 1 para set_value(key, valor_1, num_elements, vector): valor_1 = 1
Indique el numero de elementos de valor 2 para set_value(key, valor_1, num_elements, vector): num_elements = 1
Indique el elemento para set_value(key, valor_1, num_elements, vector): vector[0] = 1
Resultado = 0
Indique la operación a realizar: get_value
Get value
Indique la clave de la tupla que desea obtener: key = 1
Tupla encontrada:
Clave: 1
Valor1: 1
Valor2 longitud: 1
Valor2 vector:
[1.00 ]
Resultado = 0
Indique la operación a realizar: modify_value
Indique la clave sobre la que se desea hacer modify_value(key, valor_1, num_elements, vector): key = 1
Indique el valor 1 para modify_value(key, valor_1, num_elements, vector): valor_1 = hola
Indique el numero de elementos de valor 2 para modify_value(key, valor_1, num_elements, vector): num_elements = 2
Indique el elemento para hacer modify_value(key, valor_1, num_elements, vector): vector[0] = 3
Indique el elemento para hacer modify_value(key, valor_1, num_elements, vector): vector[1] = 4
Resultado = 0
Indique la operación a realizar: get_value
Get value
Indique la clave de la tupla que desea obtener: key = 1
Tupla encontrada:
Clave: 1
Valor1: hola
Valor2 longitud: 2
Valor2 vector:
[3.00 , 4.00 ]
Resultado = 0
Indique la operación a realizar: exist
Indique la clave sobre la que se desea hacer exist(key): key = 1
Resultado = 1
Indique la operación a realizar: delete_key
Indique la clave sobre la que se desea hacer delete_key(key): key = 1
Resultado = 0
Indique la operación a realizar: exist
Indique la clave sobre la que se desea hacer exist(key): key = 1
Resultado = 0
```

Este es un ejemplo donde se hacen las operaciones: set_value, get_value, modify_value, get_value, exist, delete_key y exist. En la imagen se ve un ejemplo de funcionamiento cuando las operaciones se realizan con las pre condiciones adecuadas: la clave usada en set_values no tenía una tupla anteriormente.

Partiendo de que se ha creado una tupla para la clave 1, se van a intentar hacer operaciones que no deberían funcionar.

```
$ env IP_TUPLAS=localhost ./cliente
Indique la operación a realizar: set_value
Indique la clave sobre la que se desea hacer set_value(key, valor_1, num_elements, vector): key = 1
Indique el valor 1 para set_value(key, valor_1, num_elements, vector): valor_1 = 1
Indique el numero de elementos de valor 2 para set_value(key, valor_1, num_elements, vector): num_elements = 1
Indique el elemento para set_value(key, valor_1, num_elements, vector): vector[0] = 1
Resultado = 0
Indique la operación a realizar: set_value
Indique la clave sobre la que se desea hacer set_value(key, valor_1, num_elements, vector): key = 1
Indique el valor 1 para set_value(key, valor_1, num_elements, vector): valor_1 = 1
Indique el numero de elementos de valor 2 para set_value(key, valor_1, num_elements, vector): num_elements = 1
Indique el elemento para set_value(key, valor_1, num_elements, vector): vector[0] = 1
Resultado = -1
Indique la operación a realizar: modify_value
Indique la clave sobre la que se desea hacer modify_value(key, valor_1, num_elements, vector): key = 2
Indique el valor 1 para modify_value(key, valor_1, num_elements, vector): valor_1 = 1
Indique el numero de elementos de valor 2 para modify_value(key, valor_1, num_elements, vector): num_elements = 1
Indique el elemento para hacer modify_value(key, valor_1, num_elements, vector): vector[0] = 1
Resultado = -1
Indique la operación a realizar: get_value
Get value
Indique la clave de la tupla que desea obtener: key = 2
call failed: RPC: Can't decode result
[Error] No se pudo obtener la tupla para la clave 2 ,compruebe que existe
Resultado = -1
Indique la operación a realizar: delete_key
Indique la clave sobre la que se desea hacer delete_key(key): key = 2
Resultado = -1
```

Vemos como se intenta volver hacer set_value sobre la clave que ya tiene una tupla y se intenta usar las operaciones modify_value, get_value y delete_key sombre una clave que no tiene una tupla.

Ahora vamos a mostrar otros errores posibles, desde parámetros de la funciones erróneos, ip errónea.

```
$ env IP_TUPLAS=localhost ./cliente
Indique la operación a realizar: set_value
Indique la clave sobre la que se desea hacer set_value(key, valor_1, num_elements, vector): key = w
[ERROR] El valor de key debe ser un int
: Success
Resultado = -1
Indique la operación a realizar: get_key
[ERROR] Operación no reconocida
Resultado = -1
Indique la operación a realizar: set_value
Indique la clave sobre la que se desea hacer set_value(key, valor_1, num_elements, vector): key = 1
Indique el valor 1 para set_value(key, valor_1, num_elements, vector): valor_1 = h
Indique el numero de elementos de valor 2 para set_value(key, valor_1, num_elements, vector): num_elements = 0
Resultado = -1
```

```
$ env IP_TUPLAS=w ./cliente
Indique la operación a realizar: set_value
Indique la clave sobre la que se desea hacer set_value(key, valor_1, num_elements, vector): key = 1
Indique el valor 1 para set_value(key, valor_1, num_elements, vector): valor_1 = 1
Indique el numero de elementos de valor 2 para set_value(key, valor_1, num_elements, vector): num_elements = 1
Indique el elemento para set_value(key, valor_1, num_elements, vector): vector[0] = 1
w: RPC: Unknown host
```

El primer ejemplo es uno donde se intenta llamar a una operación inexistente o dar un parámetro no válido. El segundo uno donde la ip es incorrecta.