MEMORIA DE LA PRÁCTICA 4 DE DISEÑO DE SISTEMAS OPERATIVOS

José Antonio Guerrero Avilés

Vamos a construir un servidor de archivos sin estado que implementa sólo algunas funciones para manipular archivos y directorios. Estas son: creat, write, read, rename para archivos, y mkdir y rmdir para directorios.

Para ello, primero creamos nuestro fichero ".x":

```
struct dobleString {
   string primero <64>;
   string segundo <2048>;
};
program SERVIDORARCHIVOS {
        version SERVIDORARCHIVOSVERS {

                int CREAR(string) = 1;
                string ESCRIBIR(dobleString) = 2;
                string LEER(string) = 3;
                int RENOMBRAR(dobleString) = 4;
                int CREARDIR(string) = 5;
                int BORRARDIR(string) = 6;

        } = 1;
} = 0x20000001;
```

Lo guardamos con el nombre, por ejemplo, "cs.x" y ya tenemos nuestro fuchero de especificación RPC.

Compilamos el fichero con:

rpcgen -C cs.x

Y se crearán los fichero cs.h, cs_clnt.c y cs_svc.c

Además, se creará también el archivo cs.xdr generado automáticamente por RPC. El contenid de este archivo es el siguiente:

```
#include "cs.h"

bool_t
xdr_dobleString (XDR *xdrs, dobleString *objp)
{
        register int32_t *buf;

         if (!xdr_string (xdrs, &objp->primero, 64))
                return FALSE;
         if (!xdr_string (xdrs, &objp->segundo, 2048))
                return FALSE;
        return TRUE;
}
```

Ahora debemos crear la implementación de las diferentes funciones (archivos cliente.c y servidor.c), que serán las interfaces del programa.

Archivo cliente.c:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "cs.h"

void main(int argc, char *argv[]) {
   CLIENT *cliente;
   int *return_value;
   char *servidor;
   // Revisamos el buen uso de los argumentos para nuestro programa.
   if (argc <= 2){
        fprintf(stderr, "error: mal uso de los parámetros.\n");
        exit (1);
   }

   servidor = argv[1];
   // Generamos el handle cliente para llamar al servidor.
   if ((cliente=clnt_create(servidor, SERVIDORARCHIVOS, SERVIDORARCHIVOSVERS, "tcp"))
== (CLIENT *)NULL) {
        clnt_pcreateerror(servidor);
        exit(2);
   }
```

```
    // Elegimos una opción mediante parámetro.
    if (strcmp(argv[2], "crear_archivo") == 0) { // 'Crear Archivo'.
            return_value = crear_1(&argv[3], cliente);
            if (! *return_value) {
                    fprintf(stderr, "error: abriendo el archivo %s.\n", argv[3]);
                    exit(1);
            }
            exit(0);

    } else if (strcmp(argv[2], "escribir_archivo") == 0) { // 'Escribir Archivo'.
            char **buffer;
            dobleString ficheros;
            ficheros.primero = argv[3];
            ficheros.segundo = argv[4];
            buffer = escribir_1(&ficheros, cliente);
            if(strcmp(buffer[0], "") != 0) { // Ha habido algún error.
                    printf("%s\n", buffer[0]);
                    exit(1);
            }
            exit(0);

    }

else if (strcmp(argv[2], "leer_archivo") == 0) { // 'Leer Archivo'.

            char **buffer;
            buffer = leer_1(&argv[3], cliente);
            printf("%s\n", buffer[0]);
            exit(0);

    } else if (strcmp(argv[2], "renombrar_archivo") == 0) { // 'Renombrar Archivo'.

            dobleString fichero;
            fichero.primero = argv[3];
            fichero.segundo = argv[4];
            return_value = renombrar_1(&fichero, cliente);
            if (*return_value) {
                    fprintf(stderr, "error: renombrando el archivo %s.\n", argv[3]);
                    exit(1);
            }
            exit(0);
    } else if (strcmp(argv[2], "crear_directorio") == 0) { // 'Crear Directorio'.

            return_value = creardir_1(&argv[3], cliente);
            if (*return_value) {
                    fprintf(stderr, "error: creando el directorio %s.\n", argv[3]);
                    exit(1);
            }
            exit(0);

    }
```

```
else if (strcmp(argv[2], "borrar_directorio") == 0){ // 'Borrar Directorio'.

        return_value = borrardir_1(&argv[3], cliente);
        if (*return_value){
            fprintf(stderr, "error: borrando el directorio %s.\n", argv[3]);
            exit(1);
        }
        exit(0);

   } else {

        printf("*** INFO: Ejemplos de ejecución ***\n");
        printf("./cliente <server> crear_archivo <nombre_archivo>\n");
        printf("./cliente <server> escribir_archivo <nombre_archivo> <texto a escribir..señala los
espacios con barra lateral >\n");
        printf("./cliente <server> leer_archivo <nombre_archivo>\n");
        printf("./cliente <server> renombrar_archivo <nombre_archivo>
<nombre_nuevo_archivo>\n");
        printf("./cliente <server> crear_directorio <nombre_archivo>\n");
        printf("./cliente <server> borrar_directorio <nombre_archivo>\n");
        printf("***//INFO***\n");
        exit(0);

   }
}
```

Archivo servidor.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/types.h>
#include "cs.h"

int *crear_1_svc (char **nombre_fichero, struct svc_req *req) {
   static int resultado = 1;
   FILE *fichero = fopen(*nombre_fichero, "w");
   if (fichero == NULL) {
        resultado = 0;
   }
   fclose(fichero);
   return (&resultado);
}
char **escribir_1_svc (dobleString *ficheros, struct svc_req *req) {
   char *buffer[1024];
   buffer[0] = "";
```

```
    // Abrimos el fichero.
    FILE *fichero = fopen (ficheros->primero, "w");
    if (fichero == NULL) {
        buffer[0] = "error: abriendo el archivo.";
        return (&buffer);
    }

    // Escribimos en el fichero.
    size_t len = strlen(ficheros->segundo);
    int resultado = fwrite (ficheros->segundo, 1, len, fichero);
    if (resultado != len) {
        buffer[0] = "error: escribiendo el archivo.";
        return (&buffer);
    }

    // Cerramos el fichero.
    fclose (fichero);
    return (&buffer);
}

char **leer_1_svc (char **nombre_fichero, struct svc_req *req) {
    FILE *fichero;
    long lSize;
    size_t result;
    char *buffer[1024];

    // Abrimos el fichero.
    fichero = fopen(*nombre_fichero, "r");
    if (fichero == NULL) {
        buffer[0] = "error: abriendo el archivo.";
        return (&buffer);
    }

    // Obtenemos el tamaño del fichero.
    fseek (fichero , 0 , SEEK_END);
    lSize = ftell (fichero);
    rewind (fichero);

    // Asignamos la memoria necesaria a nuestro buffer.
    buffer[0] = (char*) malloc (sizeof(char)*lSize);
    if (buffer[0] == NULL) {
        buffer[0] = "error: asignando memoria.";
        return (&buffer);
    }

    // Volcamos el fichero en nuestro buffer.
    result = fread (buffer[0], 1, lSize, fichero);
    if (result != lSize) {
        buffer[0] = "error: leyendo el fichero.";
        return (&buffer);
    }
```

```
    // Cerramos el archivo.
    fclose(fichero);
    return (&buffer);
}
int *renombrar_1_svc (dobleString *ficheros, struct svc_req *req) {
    static int resultado;
    resultado = rename(ficheros->primero, ficheros->segundo);
    return (&resultado);
}
int *creardir_1_svc (char **nombre_fichero, struct svc_req *req) {
    static int resultado;
    resultado = mkdir(*nombre_fichero, 0644);
    printf("%i", resultado);
    return (&resultado);
}
int *borrardir_1_svc (char **nombre_fichero, struct svc_req *req) {
    static int resultado;
    resultado = rmdir(*nombre_fichero);
    printf("%i", resultado);
    return (&resultado);
}
```

Ahora ya podemos compilar y ejecutar nuestro cliente y nuestro servidor y podremos probar las diferentes opciones que hemos implementado.

Antes de compilar deberemos reiniciar el servicio rpcbind, para lo cual hacemos:

```
sudo -i service portmap stop
sudo -i rpcbind -i -w
sudo -i service portmap start
```

Ahora compilamos nuestro cliente.c y nuestro servidor.c:

```
gcc cliente.c cs_clnt.c -o cliente -lnsl
gcc servidor.c cs_svc.c -o servidor -lnsl
```

Se crearan los ejecutables cliente y servidor, que ya se pueden probar, como está indicado en el código del programa a modo de ejemplos concretos de la siguente manera:

-Lanzamos el servidor:

```
./servidor
```

- Lanzamos una ejecución del cliente en otra terminal (por ejemplo):

```
./cliente <server> crear_archivo <nombre_archivo>
./cliente <server> leer_archivo <nombre_archivo>
```