DESARROLLO DE SISTEMAS DISTRIBUIDOS

Servidor interactivo de Archivos

Elaborado por: Ukranio Coronilla

Una gran cantidad de aplicaciones distribuidas requieren la transferencia de información entre los nodos que la componen. En ocasiones la cantidad de información es muy grande y para ello se requiere programar servidores de archivos con su correspondiente cliente.

Para tener flexibilidad en la transferencia de información entre nodos, es recomendable utilizar un formato de mensaje que pueda ser modificado con facilidad en un solo archivo de encabezado, el cual se puede compilar con cualquier aplicación que haga uso del mismo. A continuación se muestra el formato del mensaje que utilizaremos y que deberá estar contenido en el archivo de encabezado *header.h* :

```
/* Definiciones necesarias para los clientes y el servidor de archivos */
#define MAX_PATH     255   //Longitud máxima en el nombre de un archivo
#define BUF_SIZE     1024  //Cantidad de bytes que se transfieren por paquete

/* Definición de las operaciones permitidas */
#define CREATE 1  //Crea un nuevo archivo
#define READ   2  //Lee una parte del archivo
#define WRITE  3  //Escribe una parte del archivo
#define DELETE 4  //Elimina un archivo existente

/*Códigos de error */
#define OK           0     //Operación correcta
#define E_BAD_OPCODE -1    //Operación desconocida
#define E_BAD_PARAM  -2    //Error en un parámetro
#define E_IO         -3    //Error en disco u otro error de E/S

/* Formato del mensaje que viaja del cliente hacia el servidor. El tipo de dato numérico
uint32_t está definido en inttypes.h y es consistente entre computadoras distintas e
independiente del sistema operativo */
struct messageCS{
   uint32_t opcode;            //Código de la operación solicitada por el cliente
   uint32_t count;             //Numero de bytes útiles (leídos o por escribir), devuelto
por el servidor en READ y enviado por el cliente en WRITE
   uint32_t offset;            //Lugar del archivo donde se debe iniciar la lectura o
escritura, enviado por el cliente
   char name[MAX_PATH];    //Nombre del archivo sobre el cual se opera
};

/* Formato del mensaje que viaja del servidor hacia el cliente */
struct messageSC{
   uint32_t count;             //Numero de bytes útiles, devuelto por el servidor en READ y
enviado por el cliente en WRITE
   uint32_t result;            //Código de error devuelto por el servidor
   char data[BUF_SIZE];    //Datos del archivo
```

```
};
```

Lea con atención los comentarios del código header.h para que pueda entender de manera general el protocolo de transferencia de archivos, cualquier duda consúltela con el profesor.
El profesor ya tiene programado el servidor de archivos interactivos cuyo seudocódigo es el siguiente:

```
#include "header.h"

void main(void)
{
   struct message m;

   while(1){
      recvfrom(&m)
      switch(m1.opcode){
         case CREATE:         r=crear(&m);          break;
         case READ:           r=leer(&m);           break;
         case WRITE:          r=escribir(&m);       break;
         case DELETE:         r=borrar(&m);         break;
         default:
      }
      m.result = r;
      sendto(&m);
}
```

Haciendo uso de las clases `PaqueteDatagrama` y `SocketDatagrama` así como del capítulo 8 del manual "Programación de sistemas LINUX" de Ukranio Coronilla, se debe programar un ***cliente*** en lenguaje C++, que reciba como parámetros la IP, el puerto y el nombre del archivo almacenado en el servidor. El cliente solo tendrá implementada la opción de obtener un archivo que se encuentra ubicado en el servidor.

Como ejemplo si el programa cliente se ejecuta como sigue:

```
./cliente 192.168.1.100 7000 hola.mp3
```

Entonces lee el archivo hola.mp3 ubicado en el servidor y lo escribe en la computadora local y con el mismo nombre.

Comience transfiriendo el archivo, el cual cabe en un solo paquete UDP:

psetup.xpm 687 bytes

Posteriormente el archivo:

umax.jpg 1615 bytes

Dado que este archivo no cabe en un paquete UDP, es importante utilizar el campo offset para indicarle al servidor a partir de qué posición se desea realizar la lectura del archivo.

Finalmente el archivo:

Linux.mp4 9389959 bytes

En este último caso y muy probablemente existirán pérdidas de paquetes, por lo tanto será imprescindible tener programada la situación de reenvío de mensajes con ayuda de la función setsockopt para cambiar la opción del socket:

```
struct timeval tiempoFuera;

tiempoFuera.tv_sec = 0;
tiempoFuera.tv_usec = 500000;


setsockopt(s, SOL_SOCKET, SO_RCVTIMEO, (char *)&tiempoFuera, sizeof(tiempoFuera));
```

Observe que en este caso la función recvfrom() devolverá el valor -1 si el mensaje no llega en medio segundo.