3. Parte 2: Utilizando llamadas a procedimientos remotos.

Se trata de implementar el servidor FTP usando RPC. La implementación constará como en el caso anterior de un servidor y un cliente, pero en este caso el servidor implementará una serie de funciones accesibles desde el cliente vía RPC.

3.1. Servidor RPC

La interfaz del servidor RPC se describe en el fichero `srvftp.x`, el cual se reproduce a continuación:

Especificación de la interfaz del servidor RPC `srvftp.x`

```
const LENMSG = 1024;      /* Longitud de cadenas largas */
const LENNOMFILE = 128;   /* Longitud de cadenas cortas */
typedef string cadenaL<LENMSG>;     /* Cadena Long */
typedef string cadenaS<LENNOMFILE>; /* Cadena Short */

/* Especificación de parámetros para las funciones */
struct param1 {
  cadenaS filename;
  int linea;
};

struct param2 {
  cadenaS filename;
  int lineaini;
  int lineafin;
};

/* Alguna función puede retornar una lista de cadenas */
struct listalin  {
  cadenaL linea;
  listalin * sgte;
};

/* Especificación rel resultado devuelto por las funciones */
union Resultado switch (int caso) {
  case 0: bool b;
  case 1: int val;
  case 2: cadenaL linea;
  case 3: listalin * lista;
  case 4: cadenaL err;
};

program SRVFTP {
  version PRIMERA {
    Resultado isfile(cadenaS)=1;
    Resultado isreadable(cadenaS)=2;
    Resultado getnumlines(cadenaS)=3;
    Resultado getline(param1)=4;
    Resultado getlines(param2)=5;
  }=1;
}=0x2023f001;
```

El servidor RPC implementa cinco procedimientos remotos, dentro del fichero `servicios.c`:

- `isfile(cadenaS)`: Recibe como argumento una cadena que representa el nombre de un fichero. Devuelve `CIERTO` si existe un fichero con ese nombre debajo de la carpeta *document root* o `FALSO` en caso contrario.

- `isreadable(cadenaS)`: Recibe como argumento una cadena que representa el nombre de un fichero. Devuelve `CIERTO` si existe el fichero tiene permiso de lectura o `FALSO` en caso contrario.
- `getnumlines(cadenaS)`: Recibe como argumento una cadena que representa el nombre de un fichero. Devuelve el número de líneas que tiene el fichero o un error en caso de que el fichero no pueda abrirse.
- `getline(param1)`: Recibe como argumento un puntero a una variable de estructura `param1`. Esta estructura tiene dos campos, el primero una cadena que representa el nombre de un fichero y el segundo campo un entero que representa un número de línea. Devuelve la línea pedida o bien un error en caso de no existir en el fichero dicho número de línea.
- `getlines(param2)`: Recibe como argumento un puntero a una variable de estructura `param2`. Esta estructura tiene tres campos, el primero una cadena que representa el nombre de un fichero, el segundo campo un entero que representa un número de línea de inicio y el tercer campo un número de línea de fin. Devuelve una lista de líneas dentro del fichero entre número de línea de inicio y número de línea de fin inclusive.

El programa servidor rpc recibe los siguientes argumentos por línea de comando:

Forma de uso de `servidor`

```
$ servidor ruta_absoluta_document_root
Parámetros:
  - ruta_absoluta_document_root: ruta absoluta al directorio raíz de los ficheros a servir
```


Un detalle **muy importante**. Este servidor necesita acceder a la línea de comandos para recoger el parámetro `ruta_absoluta_document_root`, y además necesita almacenar el valor de ese parámetro en una variable global, para que los servicios implementados en `servicios.c` puedan acceder a ella.

Esto significa que la función `main()` generada automáticamente por `rpcgen` no nos sirve, pues no implementa esta funcionalidad específica para nuestro caso. Lo que hay que hacer es, una vez ejecutado `rpcgen`, **modificar** el fichero `servidor_svc.c` resultante para incorporarle el código adicional que haga lo que necesitamos.

Para simplificar y automatizar esto, se proporciona un script Python llamado `apply_srvftp_scv_patches.py`, y el `Makefile` suministrado se ocupa de lanzar este script cada vez que `rpcgen` sea ejecutado desde el `Makefile`. Examina el `Makefile` y el script para asegurarte de comprender lo que hacen.

La variable global declarada e inicializada desde `main()` se declara como externa dentro del fichero `servicios.c`:

Declaración dentro de `servicios.c`

```
extern char *documentroot;
```

3.2. Cliente RPC

El cliente RPC es un programa multihilo que simula a varios clientes que realizan peticiones al servidor RPC anterior. La línea de comandos del cliente es la siguiente:

```
$ cliente numero_clientes ip_serv_srvftp fich_descargas
Parámetros:
  - numero_clientes: número de clientes a simular (hilos a lanzar)
  - ip_serv_srvftp: dirección IP del servidor RPC
  - fich_descargas: fichero de descargas (ver más adelante)
```

3.2.1. Función `main()` del cliente

La función `main()` del cliente comprueba los parámetros pasados al programa desde la línea de comandos. A continuación se inicializan dos mútex:

- `mpet` permite serializar el acceso al fichero de peticiones (evitar que dos hilos estén leyendo de ese fichero a la vez)
- `m` permite evitar condiciones de carrera en el acceso a los resultados devueltos por las invocaciones remotas (evitar que dos hiloshagan una RPC a la vez, pues el resultado retornado por la RPC reside en realidad en una variable estática interna que podría estar compartida entre hilos)

Finalmente se lanza el número de hilos necesario, todos ellos ejecutando la función `procesarDescargas()`. A cada hilo se le pasa un parámetro de tipo `datos_hilo` el cual tiene dos campos, el ordinal del hilo y un puntero a `FILE` al fichero de peticiones (previamente abierto desde `main()`)

3.2.2. Hilo de descargas

La función `procesarDescargas()` implementa un algoritmo que consiste en un bucle en el que, mientras no se detecte el final del fichero de peticiones, se lee (bajo la protección del mútex `mpet`) la siguiente línea no leída. A continuación se tokeniza la línea y se extraen sus dos elementos (nombre de fichero y protocolo de descarga). En función del protocolo de descarga se establece la conexión con el servidor RPC inicializando adecuadamente una estructura `CLIENT` y, bajo la protección del mútex `m`, se realizan una serie de invocaciones remotas para comprobar si el fichero existe en el repositorio del servidor RPC, comprobar si es legible (tiene permiso de lectura) y obtener el número de líneas. Según el número de líneas sea menor o mayor que la constante predefinida `SLICE`, se hace una de las siguientes opciones:

- En caso de que el número de líneas sea mayor que 0 y menor o igual a `SLICE`, el fichero se descargará línea a línea invocando al procedimiento remoto `getline()`.
- En caso de que el número de líneas sea mayor que `SLICE` , el fichero se descargará en bloques de líneas de hasta `SLICE` líneas indicando para cada bloque el ordinal de la primera y última línea a descargar en el fichero.

A medida que se descargan las líneas estas se van grabando en un fichero de salida con el mismo nombre del fichero cuyas líneas estamos descargando. Los errores que se puedan producir se muestran por la salida de error. Cuando al ir a leer la siguiente línea del fichero de peticiones el hilo que lo hace detecta fin de fichero, ese hilo termina. Una vez que todos los hilos de descarga terminan, termina el programa cliente.

3.3. Pruebas

Al igual que en la primera parte, el servidor debe tener una serie de ficheros preparados en una carpeta y el cliente debe tener un fichero de peticiones con nombres de ficheros que existan y otros que no existan en el servidor.

1. Lanzar el servidor pasándole como argumento la ruta absoluta a la carpeta donde están los ficheros a servir.
2. Lanzar el cliente
3. Comprobar que en el lado de cliente aparecen copias de los ficheros descargados y en la salida de error aparecen mensajes de depuración que muestran lo que el cliente va haciendo.
4. Comprobar que en el lado servidor aparecen mensajes de depuración que muestran qué funciones RPC se están ejecutando.