UNC | Universidad Nacional

Cátedra de Sistemas Operativos II

Trabajo Práctico Nº I

Federico Perez

29 de Julio del 2020

Introducción

Propósito

El propósito del siguiente trabajo es familiarizarse y aprender sobre conceptos de IPC (inter-process communication), y aplicar otros conceptos aprendidos en asignaturas como Ingenieria de Software y Sistémas Operativos I, de forma práctica, resolviendo un problema de aplicación.

Todo el código y lo necesario para la ejecución de este proyecto se encuentra en el siguiente repositorio Git público:

https://github.com/0xfede7c8/SOII2020/

Problema a resolver

El problema a resolver, es la creación de un servicio cuya función es el grabado de imágenes booteables de algún sistema operativo o utilidad, en un sistema de almacenamiento másivo como puede ser un Pendrive o un disco duro.

Dicho sistema debe estar compuesto por una parte de cliente y otra de servidor, el cual puede ser utilizado remotamente para transferir las imágenes, como así soportar otras funciones.

Los detalles de los requerimientos se encuentran adjuntos a este trabajo.

Diseño de solución

El mecanismo principal de comunicación entre procesos esta dada por sockets (aunque se utilizan FIFOs en una parte del sistema para cumplir con los requerimientos). La mayoría de los mensajes transmitidos dentro del sistema, están tabulados mediante enumerados en un archivo llamado **commands.h**

```
/**
 * Enum que contiene los mensajes de control posibles entre el cliente y servidor
 */
typedef enum Message {
    CLIENT_EXIT = 0u,              /*!< Finalización de la conexión por el cliente */
    AUTHENTICATE_REQUEST,     /*!< Pedido del cliente para autenticarse */
    AUTHENTICATE_PROCEED,    /*!< Pedido del cliente para autenticarse concedido */
    AUTHENTICATE_BLOCKED,    /*!< Autenticación bloqueada por el servidor */
    AUTHENTICATE_FAILED,      /*!< Autenticación fallida */
    AUTHENTICATE_PASSED,     /*!< Autenticación exitosa */
    USER_LIST,                /*!< Pedido de lista de usuario */
    USER_LIST_FINISH,         /*!< Mensaje de terminación de envío de lista de usuarios */
    USER_PASSWORD,            /*!< Pedido de cambio de contraseña */
    FILE_LIST,                /*!< Pedido de listado de archivos */
    FILE_DOWN,                /*!< Pedido de descarga de archivo */
    FILE_DOWN_AUTHORIZED,  /*!< Autorización para descargar */
    FILE_DOWN_REJECTED,      /*!< Descarga rechazada */
    GET_FILE_SERVER_PORT,    /*!< Pedido de información del puerto del file server */
    MESSAGE_SUCCESS,        /*!< Envío o recepción de mensaje exitoso */
    MESSAGE_FAILED,          /*!< Envío o recepción de mensaje fallida */
} Message;
```

De no utilizarse estos mensajes, se utilizan otras funciones que implementan protocolos que realizan el manejo de la información requerida.

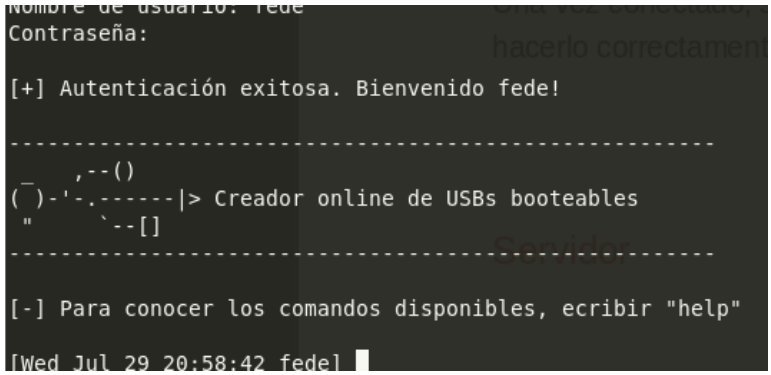
Cliente-uso

Proceso encargado de servir como interfaz de usuario al cliente físico, además de contener la lógica del grabado de imágenes en dispositivo, entre otras cosas.

Ejecución:

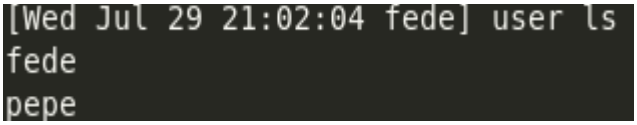
sudo client <ip-server> <puerto-server>

Una vez conectado, se le pide al usuario que ingrese sus credenciales de autenticación, y de hacerlo correctamente, se le ofrece un prompt para la ejecución de comandos:

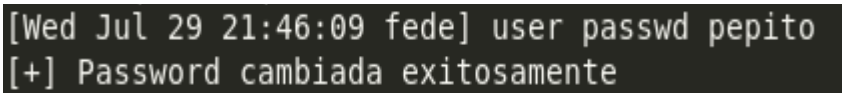


Los comandos disponibles son los siguientes:

user ls → *lista los usuarios almacenados en la base de datos*



user passwd <nueva-passwd> → *cambia la password para el cliente que esta logeado en este momento.*



file ls → *lista los archivos en el directorio de descarga del file server. Muestra el tamaño del archivo y el MD5 checksum.*

```
[Wed Jul 29 21:46:06 fede] file ls
Imagen | Tamaño | MD5
bionicpup32-8.0-uefi.iso | 286261248 | 0eb56b91ef855aa7bec37e3aa63d0b01
run.sh | 1205 | 44c5b45301d50616a73b61bb242e604b
```

file down <nombre-imagen> <dispositivo> → *descarga la imagen <nombre-imagen> (debe ser una de las disponibles al ejecutar **file ls**) y graba en el dispositivo o archivo local **<dispositivo>**. Chequea que la imágen tenga un header MBR válido al inicio para chequear la booteabilidad de la misma. Luego de grabarla muestra informacion sobre el MBR y el MD5 checksum de lo grabado.*

Cliente-implementación

Básicamente hay cinco operaciones del cliente que requieren acceso al servidor remoto (*autenticación, user ls, user passwd, file ls y file down*) y cada una de ellas se implementa mediante un protocolo sincronizado con los mensajes explicados con anterioridad.

Autenticación

El primer paso que realiza el cliente al inicializarlo es autenticarse con el servidor remoto. El servidor remoto tambien espera que lo primero que haga el cliente al conectarse al socket sea autenticarse. Si esto no ocurre, el servidor nunca va a procesar ningún comando recibido por el cliente.

Esto esta implementado con un protocolo que tiene la siguiente forma:

1. El usuario ingresa el usuario y contraseña a probar.
2. Se le envía al servidor un mensaje **AUTHENTICATE_REQUEST**, que le indica al servidor que el cliente se quiere autenticar.
3. El servidor contesta **AUTHENTICATE_PROCEED**, en el caso de estar autorizado
4. El cliente envía las credenciales en una estructura *Credentials* definida en *users_definitions.h.*

```
/**
 * Estructura que representa las credenciales del usuario
 */
typedef struct Credentials {
   char username[CREDENTIALS_SIZE];    /*!< nombre de usuario */
   char password[CREDENTIALS_SIZE];    /*!< contraseña */
} Credentials;
```

4. El servidor contesta con el estado de la autenticación:
    1. **AUTHENTICATE_PASSED:** En el caso que la autenticación haya sido exitosa.
    2. **AUTHENTICATE_FAILED:** En el caso que la autenticación haya sido fallida, por nombre de usuario o contraseña equivocados.
    3. **AUTHENTICATE_BLOCKED:** Luego de probar tres veces la autenticación con un usuario y no ingresar las credenciales correctas, el servidor de autenticación bloquea al usuario. El cliente lee este mensaje y avisa al usuario de este problema. No se pude desbloquear el usuario bloqueado sin acceder al servidor de autenticación.

Luego de estar autenticado el cliente puede realizar la operación que desee.

Como nota, todos los tipos y funciones de paso de mensajes estan definidos en el archivo *message_transmission.h* y *message_transmission.c.*

Listado de usuarios

El algoritmo es el siguiente:

1. El usuario envia al servidor el mensaje **USER_LIST**.
2. El servidor le envía y el cliente recibe la lista de usuarios mediante la función *receiveStrings/sendStrings.*
3. Luego el servicio cliente lista los usuarios por pantallas.

Cambio de contraseña

1. El usuario envía el mensaje **USER_PASSWORD**
2. El usuario luego envía el string conteniendo la nueva contraseña al servidor.
3. El servidor constesta **MESSAGE_SUCCESS** *o* ***MESSAGE_FAILED*** *dependiendo el resultado de la operación.*

Listado de imágenes disponibles

1. El cliente envía el mensaje **FILE_LIST** al servidor.
2. Luego el cliente recibe los datos mediante la función *receiveFilesDB* y la estructura *FileInfoDB* la cual esta compuesta por estructuras *FileInfo* conteniendo toda la información sobre los archivos almacenados. Estas estructuras estan definidas en *files_definition.h.*

```
/**
 * Estructura que representa información sobre
 * un archivo
 */
typedef struct FileInfo {
    char fileName[MAX_FILENAME_LENGTH];
    unsigned char md5Sum[MD5_DIGEST_LENGTH];
    size_t size;
} FileInfo;

/**
 * Estructura que representa las base de datos
 * de archivos disponibles
 */
typedef struct FileInfoDB {
    FileInfo filesInfo[MAX_FILES_AMOUNT];    /*!< Arreglo de información de archivos
*/
    size_t dbSize;                    /*!< Tamaño de la base de datos */
} FileInfoDB;
```

4. Luego el usuario imprime el nombre de cada archivo, su tamaño y su suma MD5, todo provisto por el servidor.

Descarga de imágenes a dispositivo

Este es el algoritmo más complejo del sistema e involucra varias partes.

1. El usuario envía al servidor el mensaje **GET_FILE_SERVER_PORT**, el cuál indica un pedido de los datos de conexión al servidor de archivos. Esto se debe a que el traspaso del archivo es directa con el servidor de archivos, en vez de hacer de proxy con el server principal.
2. El servidor le contesta con el puerto en el cual escucha el servidor de archivos.
3. El cliente se conecta a este servidor y le envía el nombre del archivo a descargar.
4. El servidor de archivos contesta con:
    1. **FILE_DOWN_AUTHORIZED:** En el caso que el nombre del archivo exista.
    2. **FILE_DOWN_REJECTED:** Si el archivo es inexistente.

5. En el caso de estar aprobada la transacción, el cliente empieza a recibir el archivo mediante la función *receiveBootableFileAndStore.*
6. Al recibir los primero 512 bytes, el cliente sabe si se trata de un archivo con MBR, analizando el header y de no ser así, cancela la descarga. De ser un archivo booteable, continua la descarga y grabación del archivo.
7. Al finalizar utiliza las funciones definidas en *mbr.h* y *md5.h* para mostrar información de particiones y el MD5 de lo grabado.

Server principal

El servidor funciona como proxy de los comandos recibidos por el cliente hacia los servicios, y recibe los datos de los mismos y los vuelve a enviar al cliente. En el único caso que no realiza esta tarea es en el caso de la descarga del archivo, en la cual su única tarea es darle al cliente los datos de conexión y de poner al servidor de archivos en modo descarga, escuchando nuevas conexiones. Todas las comunicaciones se hacen mediante sockets, menos para la función de listado de usuarios, para la cual se utilizan FIFOS para la comunicación con el auth server.

El código es bastante auto explicativo para esta parte por lo que no se entrará en detalles de implementación.

Cabe detallar que el diseño del servidor principal se podría haber mejorado, reduciendo el tamaño de código fuente, utilizando otro tipo de diseño mas simple, que solo haga de relay de mensajes entre usuario y servicios, en vez de explícitamente enviar y recibir todos los mensajes.

Servidor de autenticación

El servidor de autenticación o *authserver* es el encargado de mantener registro de toda la información de lo usuarios.

Esto se materializa en una “base de datos” en formato CSV llamado *credentials.csv* ubicado a la par del binario *auth.*

Al correr el proceso, este busca este archivo y lo levanta en una DB en memoria, en la estructura *UserDB:*

```
/**
 * Estructura que representa la cantidad de intentos por nombre de usuario
 */
typedef struct UserInfo {
    Credentials* credentials;
    char* lastLoginTime;
    uint32_t triesCount;
} UserInfo;

/**
 * Estructura que representa las base de datos
 * de los intentos realizados por cada usuario
 */
typedef struct UserDB {
    UserInfo userInfo[CREDENTIAL_LIMIT_AMOUNT];  /*!< DB de intentos */
    size_t dbSize;                               /*!< Tamaño de la base de datos */
    int lastUserIndex;                           /*!< Indice del último usuario en logearse */
} UserDB;
```

Esta estructura luego es utilizada por los diferentes protocolos para realizar las operaciones necesarias como entregar la lista de usuarios y cambiar las contraseñas.

El cambio de contraseñas se realiza sobre esta base de datos en memoria y inmediatamente despues, se dumpea el contenido de la estructura a la base de datos en filesystem (*CSV*).

Files server

El files server es el encargado del manejo y transmisión de archivos y metadata, entre el cliente o servidor principal y el mismo.

Las imágenes se almacenan en un directorio llamado *files* a la par del binario *fileserv*. La dinámica del funcionamiento es similar a la del auth server. Al arranar el proceso, el servidor de archivos lee los contenidos de este directorio y los almacena en una DB en memoria conteniendo la metadata de los mismos.

La DB en memoria se materializa como la siguiente estructura:

```
/**
 * Estructura que representa información sobre
 * un archivo
 */
typedef struct FileInfo {
    char fileName[MAX_FILENAME_LENGTH];
    unsigned char md5Sum[MD5_DIGEST_LENGTH];
    size_t size;
} FileInfo;

/**
 * Estructura que representa las base de datos
 * de archivos disponibles
 */
typedef struct FileInfoDB {
    FileInfo filesInfo[MAX_FILES_AMOUNT];    /*!< Arreglo de información de archivos
*/
    size_t dbSize;                    /*!< Tamaño de la base de datos */
} FileInfoDB;
```

Un punto sobre el file server es que cuando recibe un mensaje **FILE_DOWN**, este se pone en modo descarga de archivos, lo cual implica escuchar nuevamente por nuevas conexiones en el mismo socket por el cual el servidor principal se conectó.

Luego espera que el cliente se conecte directamente y ejecuta el algoritmo que se describió anteriormente.

Compilación y ejecución del servidor

En la raiz del repositorio hay un script llamado *run.sh* el cual contiene la lógica de compilación y ejecución tanto del cliente como del servidor.

Para y ejecutar el servidor con sus correspondientes servicios basta con ejecutar el siguiente comando:

```
./run.sh backend
```

```
Limpiando proyecto...
rm -rf src/client/*.o src/server/*.o src/common/*.o bin/*
rm -rf src/server/auth/*.o src/server/fileserv/*.o
Compilando servidor...
gcc -O3 -ggdb -Wall -Werror -Wno-missing-field-initializers -pedantic -Wextra -Wconversion -std=gnu11 -Isrc/common   -c -o src/server/named_pipe_connection.o src/server/named_pipe_connection.c
gcc -O3 -ggdb -Wall -Werror -Wno-missing-field-initializers -pedantic -Wextra -Wconversion -std=gnu11 -Isrc/common   -c -o src/server/authenticate.o src/server/authenticate.c
gcc -O3 -ggdb -Wall -Werror -Wno-missing-field-initializers -pedantic -Wextra -Wconversion -std=gnu11 -Isrc/common   -c -o src/server/server.o src/server/server.c
gcc -O3 -ggdb -Wall -Werror -Wno-missing-field-initializers -pedantic -Wextra -Wconversion -std=gnu11 -Isrc/common   -c -o src/common/message_transmission.o src/common/message_transmission.c
gcc -O3 -ggdb -Wall -Werror -Wno-missing-field-initializers -pedantic -Wextra -Wconversion -std=gnu11 -Isrc/common   -c -o src/common/tcp_connection.o src/common/tcp_connection.c
gcc -O3 -ggdb -Wall -Werror -Wno-missing-field-initializers -pedantic -Wextra -Wconversion -std=gnu11 -Isrc/common   -c -o src/common/mbr.o src/common/mbr.c
gcc -o bin/server src/server/named_pipe_connection.o src/server/authenticate.o src/server/server.o src/common/message_transmission.o src/common/tcp_connection.o src/common/mbr.o -O3 -ggdb -Wall -Werror -Wno-miss
ing-field-initializers -pedantic -Wextra -Wconversion -std=gnu11 -Isrc/common
gcc -O3 -ggdb -Wall -Werror -Wno-missing-field-initializers -pedantic -Wextra -Wconversion -std=gnu11 -Isrc/common   -c -o src/server/auth/named_pipe_connection.o src/server/auth/named_pipe_connection.c
gcc -O3 -ggdb -Wall -Werror -Wno-missing-field-initializers -pedantic -Wextra -Wconversion -std=gnu11 -Isrc/common   -c -o src/server/auth/auth.o src/server/auth/auth.c
gcc -o bin/auth src/server/auth/named_pipe_connection.o src/server/auth/auth.o src/common/message_transmission.o src/common/tcp_connection.o src/common/mbr.o src/server/auth/csv.h src/server/auth/user_authentica
tion.h -O3 -ggdb -Wall -Werror -Wno-missing-field-initializers -pedantic -Wextra -Wconversion -std=gnu11 -Isrc/common -Isrc/server/auth
cp src/server/auth/credentials.csv bin/
gcc -O3 -ggdb -Wall -Werror -Wno-missing-field-initializers -pedantic -Wextra -Wconversion -std=gnu11 -Isrc/common   -c -o src/server/fileserv/fileserv.o src/server/fileserv/fileserv.c
gcc -o bin/fileserv src/server/fileserv/fileserv.o src/common/message_transmission.o src/common/tcp_connection.o src/common/mbr.o -O3 -ggdb -Wall -Werror -Wno-missing-field-initializers -pedantic -Wextra -Wconve
rsion -std=gnu11 -Isrc/common -lssl -lcrypto
mkdir -p bin/files
cp -r src/server/fileserv/files bin/
Ejecutando backend...
```

Luego, el servicio queda a la espera de nuevas conexiones por parte de un cliente.

Conclusión

Fue un trabajo interesante para aplicar técnicas de diseño de software, como asi también para ejercitar el seguimiento de requerimientos y el uso de mecanismos de comunicación entre procesos para segmentar tareas. Si bien existen otras técnicas de IPC que no fueron utilizadas en el trabajo, existió una interiorización de las mismas debido a la lectura de documentación y análisis de las mejores opciones disponibles para la realización de dicho trabajo.