Cátedra de Sistemas Operativos II

Trabajo Práctico N° I

Piñero, Tomás Santiago
23 de Abril de 2020

Índice

Índice **1**

1. Introducción **2**
Objetivo . . . 2
Arquitectura del Sistema . . . 2
Definiciones, Acrónimos y Abreviaturas . . . 3

2. Descripción general **3**
Restricciones . . . 3
Esquema del proyecto . . . 3
Requisitos futuros . . . 4

3. Diseño de solución **4**
Comunicación entre los procesos . . . 4
Procesos del servidor . . . 5
Server . . . 5
Auth . . . 6
Fileserv . . . 6
Makefile . . . 6

4. Implementación y resultados **6**
Sockets . . . 6
Mensajes . . . 7
Resultados . . . 8
Inicio de sesión . . . 8
Listado de usuarios . . . 8
Listado de imágenes . . . 9
Descarga de imagen . . . 9

5. Conclusiones **10**

Referencias **10**

1. Introducción

Objetivo

El objetivo del trabajo es diseñar e implementar un software que haga uso de las API de IPC del Sistema Operativo, implementando los conocimientos adquiridos en las clases.

Arquitectura del Sistema

El sistema está formado por dos *Hosts*. El *Host 1* es el cliente, que se conecta al servidor (*Host 2*) a un puerto fijo.

El servidor está conformado por tres procesos, cada uno con funciones específicas.

1. ***Auth service***: encargado de proveer las funcionalidades relacionadas al manejo de la base de dato de los usuarios:
   - Validar de usuario.
   - Listar usuarios.
   - Cambiar contraseña del usuario.
2. ***File service***: encargado de proveer las funcionalidades relacionadas con las imágenes desponibles para descarga:
   - Mostrar información de las imágenes disponibles.
   - Establecer la transferencia de la imagen a través de un nuevo *socket*.
3. ***Primary server***: encargado de establecer las conexiones y recibir los comandos del cliente. Hace uso de los dos procesos anteriores.

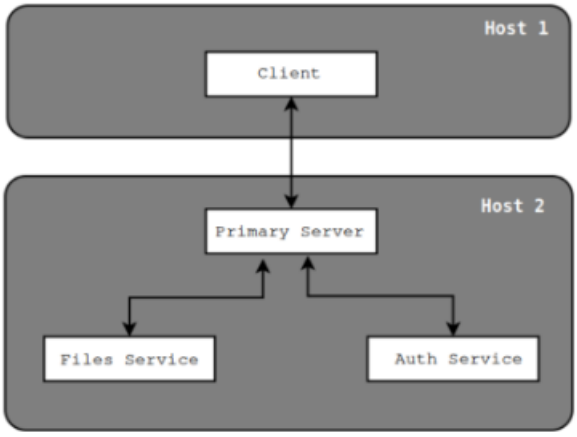

Figura 1: Arquitectura del sistema.

Definiciones, Acrónimos y Abreviaturas

- *API*: *Application Programming Interface*. Es un conjunto de funciones ofrecidas para ser utilizadas por otro software.
- *IPC*: *Inter-Process Communication*. Mecanismo del Sistema Operativo que permite a los procesos comunicarse y sincronizarse entre sí.
- *TCP/IP*: conjunto de protocolos que posibilitan las comunicaciones de Internet.

2. Descripción general

Restricciones

A las restricciones dadas (ver `Enunciado.pdf`) se le agregaron las siguientes:

1. El usuario y contraseña no pueden ser los mismos.
2. El comando para descargar la imagen debe tener el siguiente formato:

   ```
   file down ‘nombre_imagen’ ‘directorio_usb’
   ```

3. Las imágenes para descargar deben estar ubicadas en la carpeta `imgs`.

Esquema del proyecto

El proyecto está dividido en las siguientes carpetas:

- **bin**: contiene los archivos ejecutables.
- **inc**: contiene los *headers* utilizados por los códigos fuente:
  - *messages.h*: *header* con los datos para el envío y recepción de mensajes entre los procesos que conforman el servidor.
  - *sockets.h*: *header* con los datos y funciones para la utilización de los *sockets*: su creación y su lectura/escritura.
  - *utilities.h*: *header* con las utilidades comunes a los procesos.
- **imgs**: contiene las imágenes disponibles para descargar.
- **res**: contiene la base de datos de los usuarios.
- **src**: contiene los códigos fuente.
  - *client.c*
  - *primary_server.c*
  - *auth_service.c*
  - *files_service.c*
  - *sockets.c*
  - *utilities.c*

Requisitos futuros

- Utilizar *CMake* para lograr la compilación curzada.
- Hacer uso de la *API* de *MySQL* para la base de datos.
- Exigir una longitud determinada de caracteres para usuario y contraseña.
- Permitir el registro de un nuevo usuario en la base de datos.

3. Diseño de solución

Comunicación entre los procesos

Además de la implementación de *sockets*, se debe elegir otro mecanismo de los IPC existentes:

- *PIPEs* y *FIFOs*.
- Semáforos.
- *Signals*.
- Memoria compartida.
- Cola de mensajes.

Para realizar la comunicación entre los procesos del servidor se decidió utilzar el sistema de cola de mensajes *SystemV*, ya que múltiples procesos pueden acceder a una misma cola de mensajes sin problemas de concurrencia.

La cola de mensajes es *única* para los tres procesos que conforman el servidor y es creada por el proceso *server*. Los estructura de mensaje utilizada para la cola es la siguiente:

Código 1: Estructura de mensaje.

```
1 struct msg             /* Estructura para mensajes generales. */
2 {
3   long mtype;          /* Identificador del mensaje. */
4   char msg[STR_LEN];   /* Contenido del mensaje. */
5 };
```

El identificador de mensaje que se muestra en el Código 1 puede tomar los siguientes valores:

- **to_prim**: los mensajes dirigidos al proceso *server*.
- **to_auth**: los mensajes dirigidos al proceso *auth*.
- **to_file**: los mensajes dirigidos al proceso *fileserv*.

Procesos del servidor

A continuación se explica brevemente el diseño para cada uno de los procesos involucrados en el sistema[1].

Server

Es el intermediario entre el cliente y los otros dos procesos que conforman el servidor (ver Figura 1).

Este proceso tiene como hijos al proceso *auth* y *fileserv*. La creación de estos procesos se realiza una vez creados los medios de comunicación necesarios (*socket* y cola de mensajes). Se decidió por esta implementación debido a que de esta forma los procesos hijos podrán acceder a la cola de mensajes sin problemas.

El protocolo de envío de mensajes entre ellos se muestra en el siguiente diagrama de secuencia.

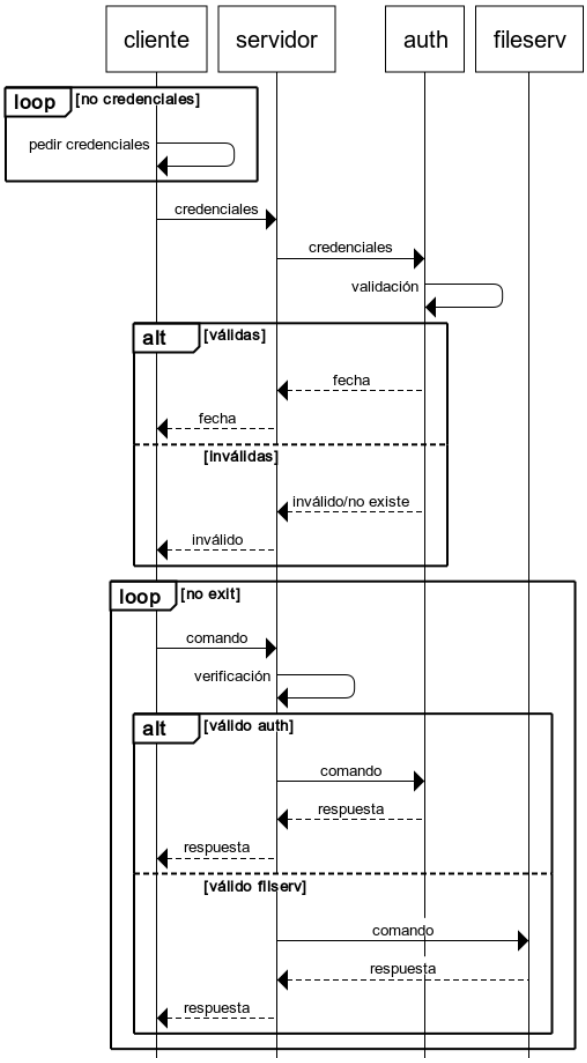

Figura 2: Diagrama de secuencia de la comunicación cliente-servidor.

[1]Para información más detallada sobre el funcionamiento de cada proceso se recomienda ver la documentación del mismo

Auth

Proceso encargado del manejo de la base de datos de los usuarios y de la validación de las credenciales enviadas por el usuario.

Se permiten tres intentos de inicio de sesión, al superarse este límite, el usuario se bloquea y se terminan todos los procesos del sistema. Si el inicio de sesión es exitoso, se modifica la fecha y hora del último inicio de sesión a la fecha actual.

Base de datos

Para simular la base de datos se utiliza un archivo `.txt` con valores separados con coma. La estructura sus contenidos es la siguiente:

```
ID,usuario,contraseña,estado,última_conexión
```

Fileserv

Este proceso se encarga del manejo de las imágenes disponibles para descargar, las cuales se encuentran en la carpeta `imgs` y también de enviar la imagen seleccionada por el usuario mediante un nuevo socket.

Makefile

Las recetas disponibles en el *Makefile* son las siguientes:

1. *all*: genera los ejecutables del sistema.
2. *check*: corre *cppcheck* sobre el directorio `src`.
3. *doc*: genera la documentación del código en HTML utilizando *doxygen* en la carpeta llamada `Doc` y la abre en el navegador *Firefox*.[2]
4. *client*: crea el directorio `bin` y genera el ejecutable correspondiente al cliente.
5. *server*: general los ejecutables correspondientes al servidor.
6. *clean*: elimina los directorios `bin` y `Doc`.

4. Implementación y resultados

Sockets

El método de comunicación de *sockets* es utilizado por tres procesos: *client*, *server* y *fileserv*, por lo que se decidió realizar un *header* con las funciones que se utilizan para la creación de los *sockets* y su lectura/ escritura.

[2]Si no se encuentra instalado, se sugiere cambiar el navegador en la receta.

A continuación se muestra brevemente el contenido del *header*.

Código 2: *Header* para el uso de *sockets*.

```
enum ports        /** Puertos para la conexion de los sockets. */
{
  port_ps = 4444, /** Puerto para 'primary_server'. */
  port_fi = 5555  /** Puerto para 'files_service'. */
};

#define STR_LEN 1024  /** Largo de los strings */

/** Escribe en el socket deseado un mensaje. */
ssize_t send_cmd(int sockfd, void *cmd);

/** Lee el mensaje del socket deseado. */
ssize_t recv_cmd(int sockfd, void *cmd);

/** Crea el socket en el puerto pedido. */
int create_svsocket(char *ip, uint16_t port);

/** Conecta al cliente en el puerto del servidor. */
int create_clsocket(char *ip, uint16_t port);
```

Mensajes

Como se explicó en la subsección 3, se utiliza una cola de mensajes entre los tres procesos que conforman el servidor, por lo que dichos procesos utilizan una librería en común: *messages.h*.

Código 3: *Header* para el uso de la cola de mensajes.

```
#define STR_LEN 1024 /**< Largo de los strings */

/* Enum con con los destinos de los mensajes a enviar en la cola de
   mensajes. */
enum msg_ID
{
  to_auth = 1, /* Destinado a 'auth_service'. */
  to_file = 2, /* Destinado a 'files_service'. */
  to_prim = 3  /* Destinado a 'primary_server'. */
};

/* Enum con los estados posibles del usuario. */
enum status
{
  blocked = 1, /* El usuario esta bloqueado. */
  active  = 2, /* El usuario esta activo. */
  invalid = 3, /* Credenciales invalidas. */
  not_reg = 4  /* Es usuario no existe. */
};

#define QU_PATH  "./server" /* Archivo para crear la cola de mensajes. */
```

En este *header* también se encuentra incluida la estructura de mensaje mostrada en el Código 1.

Resultados

A continuación se muestran los resultados obtenidos al correr el programa con los comandos requeridos. Las figuras que muestran el listado de usuarios e imágenes disponibles fueron sacadas conectado cliente y servidor a la dirección *loopback* (`127.0.0.1`), mientras que la descarga de imágenes son de un cliente desde una *Raspberry Pi* con dirección IP `192.168.0.177`.

Inicio de sesión

```
mrgreen@Archimedes:~/Documents/Materias/SOII/TP1/bin$ sudo ./client 127.0.0.1
Ingrese usuario: tomas
Ingrese contraseña:

Last login: 22/04/2020 01:29:15
#>
```

Figura 3: Inicio de sesión exitoso.

```
mrgreen@Archimedes:~/Documents/Materias/SOII/TP1/bin$ sudo ./client 127.0.0.1
Ingrese usuario: hola
Ingrese contraseña:
El usuario no existe.

Ingrese usuario: tomas
Ingrese contraseña:
Credenciales inválidas.

Ingrese usuario:
```

Figura 4: Inicio de sesión fallido,

Listado de usuarios

```
Last login: 22/04/2020 22:19:05
#> user ls
=================================================
Usuario         Estado          Último login
=================================================
tomas           activo          22/04/2020 22:41:30
doomslayer      activo          06/06/2006 06:06:06
1337            bloqueado       15/02/2019 09:53:08
beastmaster64   bloqueado       15/03/2020 20:53:26
tame            activo          19/04/2020 21:29:38

#>
```

Figura 5: Comando *user ls*.

Listado de imágenes

El comando demora en mostrarse debido a que el cálculo del *hash* MD5 se realiza en el momento de ejecución.

```
#> file ls
================================================================================
Imagen                          Tamaño [MB]    MD5
================================================================================
bionicpup32-8.0-uefi.iso        286            9610a1bc7d30e12979d2fd0dbd985859
install66.fs                    471            a3f0f07d33b339bb9989f440c15a64b9

#>
```

Figura 6: Comando *file ls*.

Descarga de imagen

```
pi@raspberrypi:~/TP1/src $ sudo ./cl 192.168.0.135
Ingrese usuario: tomas
Ingrese contraseña:

Last login: 22/04/2020 22:01:28
#> file down bionicpup32-8.0-uefi.iso /dev/sda
Writing...
Done writing.
Escritura exitosa.

================================
 - Booteable: Si.
 - Tipo de partición: 00
 - Sector de inicio: 0
 - Tamaño de la partición: 286 MB

#>
```

Figura 7: Comando *file down 'imagen' 'usb'*.

```
pi@raspberrypi:~ $ sudo dd if=/dev/sda bs=1 count=510 | tail -c 64 | hexdump -C
510+0 records in
510+0 records out
510 bytes copied, 0.00478987 s, 106 kB/s
00000000  80 00 01 00 00 3f 60 10  00 00 00 00 00 88 08 00  |.....?`.........|
00000010  00 fe ff ff ef fe ff ff  78 00 00 00 00 30 00 00  |........x....0..|
00000020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
```

Figura 8: Comando *hexdump* en USB.

```
mrgreen@Archimedes:~/Documents/Materias/SOII/TP1/imgs$ dd if=./bionicpup32-8.0-u
efi.iso bs=1 count=510 | tail -c 64 | hexdump -C
510+0 records in
510+0 records out
510 bytes copied, 0,00407534 s, 125 kB/s
00000000  80 00 01 00 00 3f 60 10  00 00 00 00 00 88 08 00  |.....?`.........|
00000010  00 fe ff ff ef fe ff ff  78 00 00 00 00 30 00 00  |........x....0..|
00000020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
```

Figura 9: Comando *hexdump* en imagen.

5. Conclusiones

El diseño general del sistema fue claro desde el comienzo. Durante el momento de implementación se tuvieron problemas causados tanto por la poca expeciencia en el lenguaje C como en el manejo de los errores. La implementación de la cola de mensajes y los *sockets* fueron más sencillas de lo esperado.

A pesar de las dificultades, los requerimientos pedidos por la cátedra fueron cumplidos.

Referencias

[1] *Beej's Guide to Unix IPC, Message Queues*,
`Link to Message Queues`

[2] *Codeforwin: Replace text in a line*,
`Link to Replace text`

[3] *Stackoverflow: How to get date and time*,
`Lonk to Date-time`

[4] *Web Sequence Diagram*,
`Web sequence diagrams`

[5] *Stackoverflow: How to list files in a directory*,
`Link to list files`

[6] *Stackoverflow: How to calculate MD5*,
`Link to calculate MD5`

[7] *Stackoverflow: How to show MD5 as a string*,
`Link to MD5 to string`