Cátedra de Sistemas Operativos II

Trabajo Práctico N° I

Piñero, Tomás Santiago

XX de Abril de 2020

Índice

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.

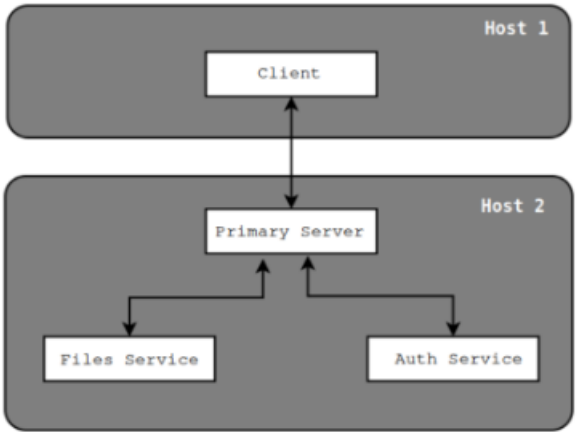

Figura 1: Arquitectura del sistema.

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.
- **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*

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.

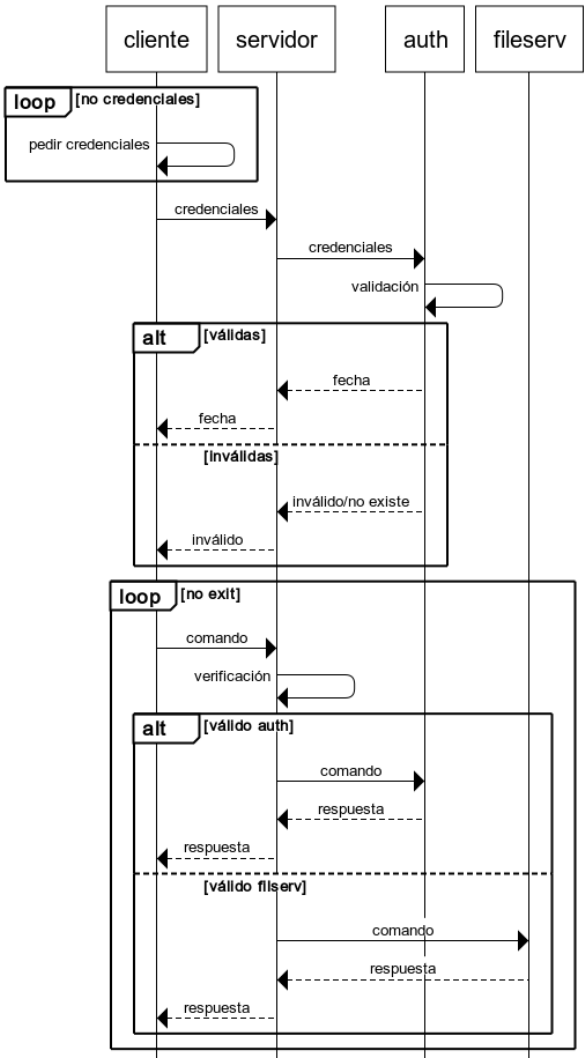

Figura 2: Diagrama de secuencia de la comunicación cliente-servidor.

[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.

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

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

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

```c
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 */

#define SV_IP "127.0.0.1" /** Direccion IP del servidor. */

/** 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(uint16_t port);

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

5. Conclusiones

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`