Cátedra de Sistemas Operativos II

Trabajo Práctico N° II

Piñero, Tomás Santiago

14 de Mayo de 2020

Índice

1. Introducción

Objetivo

El objetivo del trabajo es diseñar una solución que utilice el paradigma de memoria compartida mediante *OpenMP* para realizar el procesamiento de una imagen *BMP*.

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.
- **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*

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

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*.[1]
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.

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

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

Código 1: *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 **??**, 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 2: *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 **??**.

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`.

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`