SISTEMAS OPERATIVOS

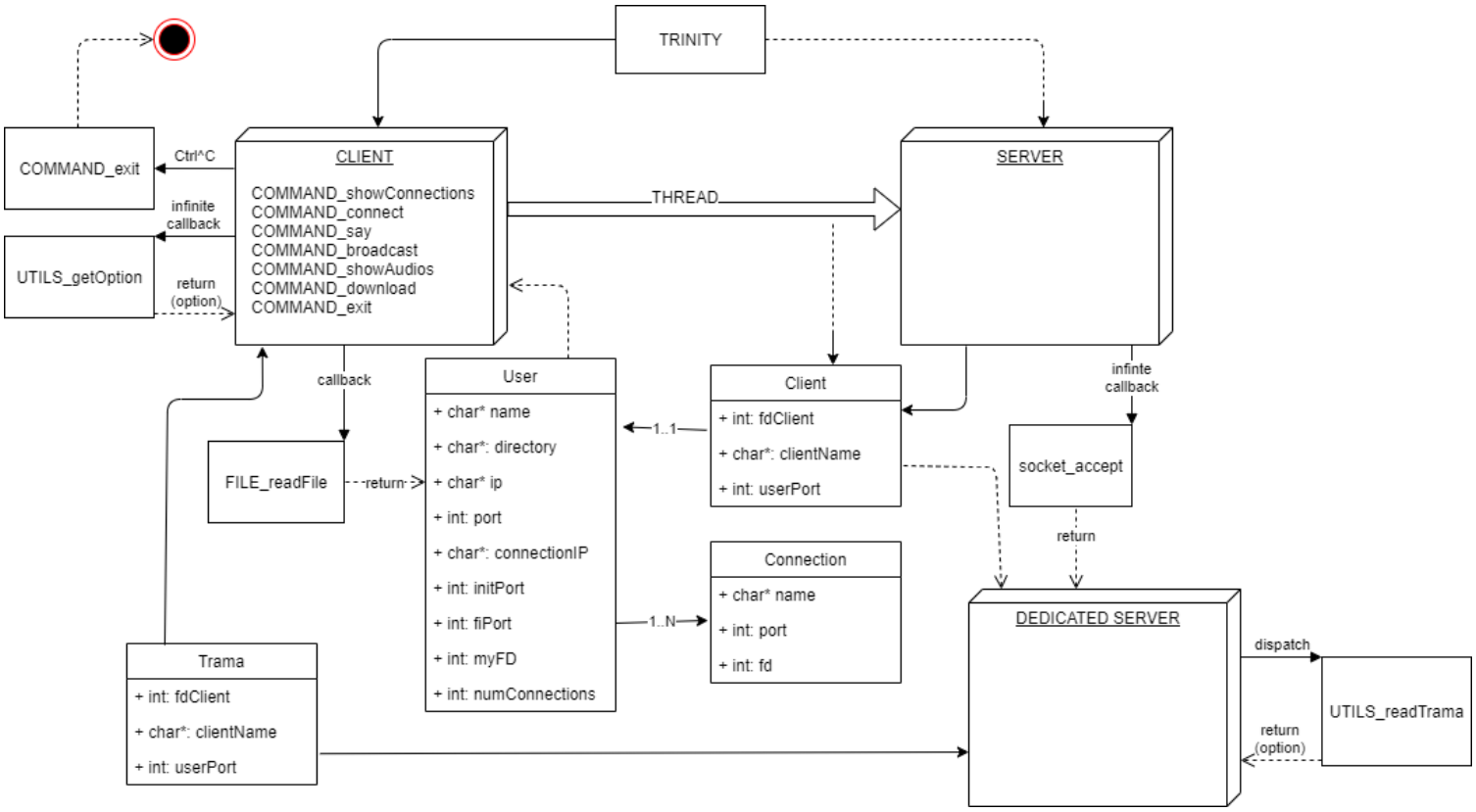
Práctica curso 2019-2020

Javier Gaig Mellado
Paula Rodriguez Delgado

Índice

1. Diseño general

A. De un único Trinity

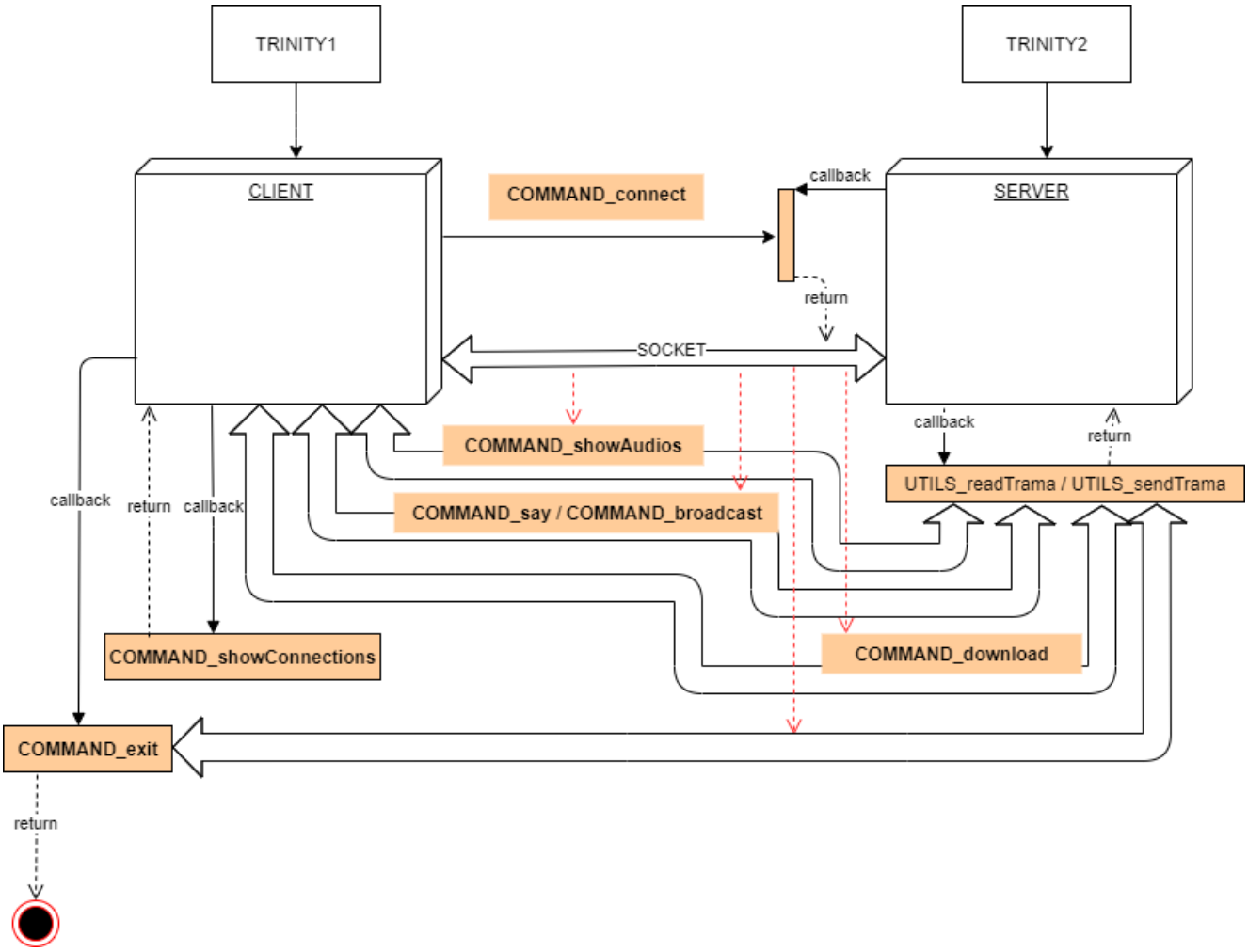


Solo empezar el programa nos creamos el cliente Trinity que, mediante la función FILE_readFile, se crea su usuario e inicializa sus conexiones. Acto seguido el programa realiza un thread que crea la parte servidor de Trinity. A ésta se le pasa nuestro usuario, es decir, su estructura llena que forma la parte de cliente de Trinity, ya que hay aspectos que necesitará en un futuro. Una vez creado nuestro Trinity según el fichero recibido al ejecutar el programa, tenemos dos partes diferenciadas y que realizan diferentes funciones:

- **Trinity client:** la parte del cliente está en un bucle constante leyendo lo que el usuario escribe, decodifica la instrucción que ha introducido y, dependiendo de lo que se quiera hacer, llamará a una función u otra (UTILS_getOption). Estará en este bucle hasta que el comando escrito sea “exit” o se presione Ctrl^C.

- **Trinity server:** el servidor está en un bucle infinito bloqueado en el socket_accept, es decir, esperando a que haya algún cliente que se le quiera conectar. Cuando se recibe una solicitud de conexión, se crea el canal, se llena la información de éste nuevo cliente y se le crea un servidor dedicado. Éste estará en un bucle constante leyendo tramas que recibe y ejecutando diferentes funciones según lo que reciba hasta que el cliente se desconecte.

B. De la comunicación entre Trinity's



Como hemos dicho anteriormente, la parte de servidor del Trinity está constantemente a la escucha de que haya alguien que se quiera conectar. Y es cuando un cliente hace un "connect" al puerto del servidor que está escuchando, que se crea el vínculo y el canal de conexión mediante un socket. Este canal es usado por casi todo el resto de los comandos para pasarse un conjunto de tramas:

- **Say / Broadcast:** se envía una trama (a un usuario en concreto /a todos los usuarios con los que se haya realizado un connect) al servidor con el mensaje que se le quiera decir. Éste responde con otra trama con un MSGOK o MSGKO dependiendo de si se ha recibido y procesado correctamente la trama.

- **Exit:** trama para informar a todos los clientes con los que estoy conectado que procedo a desconectarme, por lo tanto, que la parte cliente de Trinity dejará de funcionar. El servidor responde con un CONOK o CONKO dependiendo de si se ha podido realizar la desconexión correctamente o no.

- **Donwload:** se le envía una trama a un usuario en específico con le nombre de un audio que se quiera descargar. El servidor va enviándole trozos del audio hasta que le envía un EOF, informando de que se ha acabado de enviar todo, y con el correspondiente MD5SUM. Finalmente, el cliente le envía una última trama conforme el MD5SUM es correcto o no (MD5OK/MD5KO)

- **Show audios:** el cliente le envía una trama con cabecera SHOW_AUDIOS a un servido en específico para que éste le envíe los títulos de todos los audios que tiene disponibles mediante una trama con cabecera LIST_AUDIOS

C. Recursos del sistema utilizados

Los únicos recursos utilizados han sido:

- **Pipes y fork:** se ha utilizado para el show connections. Ya que si se ejecutaba el .sh directamente se mostraba por pantalla en un formato diferente la información deseada. Por lo que se ejecutaba el show_connections.sh en el hijo y se le pasaba el file descriptor correspondiente con la salida del comando al padre, que lo parseaba y mostraba la información de forma adecuada.
- **Signals:** se ha utilizado un signal ya que el usuario, en cualquier momento, puede utilizar Ctrl^C para desconectarse.

2. Problemas observados y como se han solucionado

El primer problema al que nos enfrentamos con la realización de la práctica fue con la realización de la funcionalidad de *Show Connections.* En esta, no sabíamos porqué no funcionaba, ya que creíamos que realizábamos las cosas de forma correcta. Tras darle muchas vueltas, más adelante caímos en la cuenta de que no se pasaban de forma correcta los argumentos de la función *execv.*

El segundo mayor problema al que nos enfrentamos con la realización de la práctica, fue la funcionalidad de *Download.* Nada más comenzar a realizar esta funcionalidad, se probó la descarga con ficheros de texto. Funcionaban bien, pero cuando llegábamos a la descarga de un audio, no se guardaba en su totalidad la información que se enviaba. Con ayuda de los becarios, vimos que, al enviar una trama, antes del envío de esta, los datos pasaban primero por la función *asprintf.* Esta función, en cuanto encuentra un '\0', deja de unir información porque ya indica el final de la cadena. Lo que pasa con los ficheros de audios, es que, al ser ficheros binarios, puede contener algún '\0'. Quitando esta función, finalmente se consiguió el envío satisfactorio del audio.

3. Estimación temporal

Diseño

El diseño de la práctica que se realizó al principio ha tenido ciertas modificaciones con el que se ha terminado utilizando. En total, con los cambios añadidos aproximadamente se habrá tardado unas 5 horas.

Implementación

La implementación, ha sido el apartado al que más tiempo se le ha dedicado. Al ir mezclando la implementación con el *testing*, podemos decir que aproximadamente se habrán tardado entre unas 30-35 horas, incluso un poco más.

Testing

Como se ha dicho antes, no podemos diferenciar mucho el tiempo entre el tiempo de implementación con el de *testing*, ya que cada vez que se implementaba algo se iba probando que realmente aquello funcionaba. Si añadimos a este tiempo de *testing* el uso de la herramienta *valgrind,* podríamos decir que aproximadamente se han invertido unas 10 horas.

Memoria

El apartado de la memoria se ha tardado unas 4 horas en realizarla.

4. Conclusiones y propuestas de mejora

Esta práctica ha sido muy interesante no tan solo a nivel técnico, sino también por el tema. Se ha aprendido y profundizado conceptos y recursos muy útiles del lenguaje. Nos ha gustado la temática y la forma de desarrollarla mediante los diferentes checkpoints que ayudan a saber si el desarrollo actual ayuda y facilita la siguiente funcionalidad o si lo dificulta y se ha de cambiar algún aspecto no visto previamente del proyecto. Una cosa que hemos aprendido y encontramos de gran utilidad es el buen uso, aprovechamiento y liberación de la memoria, que de soporte a un código óptimo y menos propenso a errores.

Una propuesta de mejora que encontramos es que se deje libertad para los protocolos de comunicación, ya que puede ser que alguna estructura se haya adaptado a este protocolo o que, mediante otra forma de comunicación, se consiga una optimización del programa.

5. Bibliografía utilizada

Material Proporcionado por la universidad

LlibreUNIX_14-15

La resolución de los laboratorios propuestos cada semana

Páginas web

www.github.com

www.stackoverflow.com

https://linux.die.net/man/