Trabajo práctico N 1: IPC - Sockets UNIX

Trabajo práctico Nº 1

Sistemas Operativos II
DC - FCEFyN - UNC

Marzo de 2020

1. Introducción

Los sockets son una abstracción de comunicación entre procesos (IPC) que, en un sistema tipo UNIX, se implementan en un descriptor de archivo, sobre el cual se envía o recibe información, al igual que como se lee o escribe un archivo (1). Son una herramienta muy importante, uno de los pilares de la comunicación entre procesos, y sumamente utilizada en la mayoría de las aplicaciones de red.

2. Objetivo

El objetivo del presente trabajo práctico es que el estudiante sea capaz de diseñar e implementar un software que haga uso de las API de IPC del Sistema Operativo, implementando lo visto en el teórico, práctico y haciendo uso todos los conocimientos adquiridos en Ingeniería de Software y Sistemas Operativos I.

3. Desarrollo

Se pide que diseñe, implemente y testee un software (desarrollado en lenguaje C), siguiendo el diagrama de la Figura 1.

El cliente (*Host 1*) debe conectarse al servidor (*Host 2*) de manera segura [1] a un puerto fijo. El acceso debe ser mediante validación de usuario y contraseña, que, en caso de poder conectarse, el servidor le proporciona un *prompt*.

Las credenciales se le solicitan al usuario cuando este inicia el programa cliente. De ingresar credenciales erróneas, el sistema debe notificar al usuario y pedirle que las ingrese nuevamente. Al Tercer intento incorrecto, se debe bloquear al usuario para impedir ataques de fuerza bruta.

[1] De manera segura nos referimos a confiable, no a autenticación, Integridad y confidencialidad

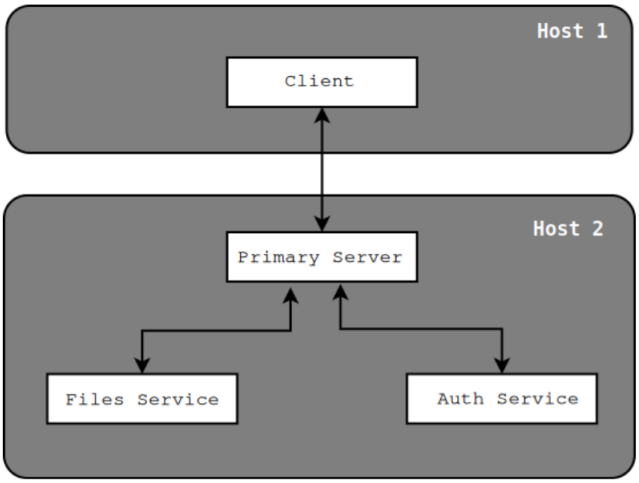

Figura 1: Arquitectura del sistema

3.1. Desde el punto de vista del cliente

Una vez autenticado, se podrá enviar una serie de comandos al servidor a través del *prompt*.

Los comandos a implementar son:

1. **exit**: Envía al servidor un aviso de finalización de sesión y cierra la conexión local. El servidor vuelve a escuchar nuevas conexiones

2. **user ls**: Lista los usuarios, si estos están bloqueados y su última vez de conexión exitosa.

3. **user passwd <*new_pass*>**: Cambia el password del usuario actual por *new_pass*

4. **file ls**: Lista las imágenes disponibles en el servidor para su descarga, junto con su tamaño y su *hash MD5* (2).

   Las imágenes están almacenadas en una carpeta del servidor y se corresponden a imágenes de instalación de sistemas operativos *booteables* por USB. Por ejemplo la imagen de instalación de *Arch Linux*(3) o textitCentOS(4).

5. **file down <*image_name*>*[...opt1...] [.....] [...optn...]***: Este comando permite la descarga directa de la imagen *image_name* del servidor al dispositivo USB local, de tal manera que a la finalización de la ejecución del comando, el dispositivo USB quedará en condiciones de utilizarse para la instalación de un sistema operativo. Una vez finalizada la descarga y escritura en el USB, el software cliente deberá obtener localmente y mostrar en pantalla la siguiente información:

a) *Hash MD5* de lo escrito en el dispositivo

b) Información de la tabla de particiones, (Sólo MBR y particiones primarias) de la imagen, como su tamaño, tipo, sector de inicio y si es *booteable* (5).

Los argumentos ***[...opt1...] [.....] [...optn...]*** son argumentos opcionales, que el programador decidirá si los necesita agregar, sus valores y su funcionalidad. Por ejemplo: el programador podría utilizar, si así lo desea, un argumento para indicar el dispositivo USB en concreto a grabar (6).

3.2. Desde el punto de vista del servidor

Se divide en tres procesos diferentes, cada uno de ellos con funciones específicas.

1. **Auth Service**: Encargado de proveer toda la funcionalidad asociada al manejo de la base de datos de usuarios, esto es, validar usuarios, bloquear usuario, listar usuarios y cambiar la contraseña de los mismos.

2. **File Service**: Encargado de proveer funcionalidad relacionada a las imágenes disponibles para descarga, cómo obtener nombres, tamaño y sus *hash MD5*. También es el encargado establecer la transferencia del archivo a través de un nuevo socket.

3. **Primary server**: Es el *middleware* encargado de establecer conexiones y recibir comandos de conexiones válidas. Hace uso de los otros 2 procesos para cumplir los requerimientos del software.

4. Restricciones

4.1. Del servidor

1. No se pueden crear procesos adicionales, sólo los tres definidos. Esto implica que **no** pueden utilizarse llamadas al sistema que creen procesos como por ejemplo un *popen* o *system(ls ./)*.

2. Los procesos deberán ser tres binarios distintos.

3. Para la transferencia del archivo no debe utilizarse el mismo socket que para la comunicación de comandos.

4. Sólo puede atender de a un cliente por vez.

4.2. Lado del cliente

El comando *CTRL-C* debe enviar el comando *exit* al servidor antes de salir del programa.

4.3. Del sistema

1. Para la comunicación entre procesos, en su conjunto, se deberá utilizar al menos dos mecanismos de IPC diferentes. (Socket cuentan como uno)
2. Debe incluirse un mecanismo de control y manejo de errores en todo el sistema.
3. Se exige el uso de *cppcheck* y documentación con *doxygen*.
4. La compilación debe hacerse con al menos las banderas de *-Wall -Werror -pedantic -Wextra -Wconversion* y se debe estar bajo el estándar *gnu11*(7).
5. Deberá utilizar dos plataformas de hardware diferentes, una para el servidor y otra para el cliente. Recomendamos para simplificar trabajo utilizar una Raspberry o similar.

5. Recomendaciones

- Hacer pruebas localmente cliente/servidor a través de la interfaz de loopback simplifica mucho el trabajo.
- **Tenga en cuenta que un código mal diseñado o mal parametrizado puede funcionar correctamente en una computadora con un disco, pero en una con dos o más discos puede llevar a una pérdida de información y perdida de tabla de particiones del segundo disco. Para segmentar el trabajo podría primero probar copiar localmente a un archivo y no directamente sobre el USB. Usar el comando *md5sum* sirve para realizar *checks md5* sobre archivos.**
- Utilizar tipos de datos independientes de la plataforma a través de *¡stdint.h¿*. Esto permite evitar errores debido a la diferencia de arquitecturas y compiladores cruzados. Por ejemplo, en vez de utilizar *int*, utilizar *int32_t* para asegurarse de que el entero ocupa 32 bits.
- Utilizar estructuras para representación de datos y como argumentos de las funciones permite extender la funcionalidad sin modificar las interfaces.
- No escriba dos veces el mismo código, si dos funciones son muy similares debería dividirlas en funciones más cortas y compartir funciones para compartir funcionalidad, mantener consistencia y detectar más fácil errores.
- Si va a utilizar fragmentos de código para *debugging*, relevamiento de estadísticas o diferentes comportamiento del código en desarrollo utilicen las directivas de precompilación *ifdef* y *endif*.
- No hardcode números en el código, utilice defines.

- Defina el protocolo de comunicación antes de empezar a codificar (Formato y secuencia de mensaje)
- Definir tipos de datos en *headers* comunes a varios procesos permite mantener consistencia (invariantes)
- Utilice el *man page* de Linux.
- Lea mucho, escriba simple, si se complica relea el código y corrija.

6. Entrega

Se deberá proveer los archivos fuente, ası como cualquier otro archivo asociados a la compilación, archivos de proyecto "Makefile" y el código correctamente documentado. No debe contener ningún archivo asociado a proyectos en IDE y se debe asumir que el proyecto podrá ser compilado y corrido por una *tty* en una distribución de Linux con las herramientas típicas de desarrollo. También se deberá entregar un informe con el formato provisto.

7. Evaluación

El presente trabajo práctico es **individual** y deberá entregarse antes de las 23:50ART del día 23 de Abril de 2020 mediante el LEV. Será corregido y luego se coordinará una fecha para la defensa oral del mismo.

Referencias

[1] M. Kerrisk, *The Linux Programming Interface.* Adison-Wesley, 2010.

[2] S. M. Ulrich Drepper and D. Madore. md5sum(1) the linux manual page. [Online]. Available: man7.org/linux/man-pages/man1/md5sum.1.html

[3] A. Linux. Downloads. [Online]. Available: https://www.archlinux.org/download/

[4] CentOS. Downloads. [Online]. Available: https://www.centos.org/download/

[5] D. B. Sedory. Mbr/ebr partition tables. [Online]. Available: https://thestarman.pcministry.com/asm/mbr/PartTables.htm

[6] S. S. Lars Wirzenius, Joanna Oja. The /dev directory. [Online]. Available: hhttps://www.tldp.org/LDP/sag/html/dev-fs.html

[7] *Making The Best Use of C.* [Online]. Available: https://gcc.gnu.org/onlinedocs/gcc/Standards.html