Proyecto Integrador de Redes y Sistemas Operativos: Cliente de NachOS

A. Arévalo Alvarado, A. Badilla Olivas, G. Rivera Hernández, J. P. Chacón González

E.C.C.I., Universidad de Costa Rica.

Contributing authors: ariel.arevalo@ucr.ac.cr; anthonny.badilla@ucr.ac.cr; geancarlo.riverahernandez@ucr.ac.cr; jean.chacongonzalez@ucr.ac.cr;

Resumen

En esta investigación se presenta el desarrollo e implementación de un cliente en `NachOS` que es capaz de solicitar y obtener figuras de *LEGO* específicas desde el servidor previamente construido. El proyecto se desvió del cliente anterior en términos de arquitectura debido a las limitaciones técnicas impuestas por el sistema NachOS. Se describen los resultados de las pruebas realizadas y se concluye con una reflexión sobre la complejidad de desarrollar un programa con funcionalidad de red y las expectativas para las etapas futuras del proyecto.

Keywords: HTTPS, servicio de red, protocolos de comunicación

1. Introducción

Esta investigación presenta el desarrollo e implementación de un cliente que es capaz de solicitar figuras de *LEGO* específicas a un servidor y obtener las piezas necesarias para construir el objeto solicitado. Fundamentalmente, debido a la existencia de trabajo anterior en NachOS y la primera etapa de este mismo proyecto, este trabajo consistió solamente en la elaboración de un programa de usuario `client`, el cual corre en NachOS, así como las modificaciones necesarias al servidor de LEGO para que este emitiera una salida legible al humano para el cliente de NachOS. El documento describe la metodología utilizada para lograr estos objetivos y presenta los resultados obtenidos. El programa en funcionamiento fue desarrollado para entornos Linux y fue probado con éxito según los requisitos del proyecto.

2. Desarrollo

2.1. NachOS

A diferencia de las demás entregas de este proyecto, el cliente de NachOS fue menos estricto en la adherencia a una arquitectura específica, particularmente debido a las limitaciones técnicas del ambiente de NachOS. Aunque en general se podría decir que se sigue un paradigma orientado a objetos, ocasionalmente el código nativo de NachOS, así como el nuestro, presenta violaciones de este paradigma en aras de evitar una explosión de complejidad debido a la naturaleza de la funcionalidad siendo implementada. Como preparación del ambiente de NachOS se desarrollaron los siguientes llamados a sistema:

2.1.1. Read

El llamado `Read` obtiene el contenido de un descriptor de archivo. La implementación utilizada lee los valores de la dirección donde guardar el contenido y la cantidad de bytes a leer, así como el descriptor del cual leer, de los registros de la máquina. Seguido, se confirma si el descriptor representa la entrada estándar, un archivo, o un socket. Si es un socket, se confirma que este sea SSL, seguido lo cual se hace el procedimiento de lectura haciendo uso de las librerías OpenSSL, y se escribe el contenido del socket a memoria. Si el socket no es SSL, se hace uso del llamado `read` de Unix. Se retorna el número de bytes leídos.

2.1.2. Write

El llamado `Write` escribe contenido a un descriptor de archivo. La implementación utilizada lee los valores de la dirección donde tomar el contenido y la cantidad de bytes a escribir, así como el descriptor al cual escribir, de los registros de la máquina. Seguido, se confirma si el descriptor representa la salida estándar, un archivo, o un socket. Si es un socket, se confirma que este sea SSL, seguido lo cual se hace el procedimiento de escritura haciendo uso de las librerías de SSL, y se escribe el contenido de la memoria al socket. Si el socket no es SSL, se hace uso del llamado `write` de Unix. Se retorna el número de bytes escritos.

2.1.3. Socket

El llamado `Socket` genera un socket nuevo. La implementación utilizada es un espejo de aquella en la clase `IPv4SslSocket` de la primera etapa. Se leen los valores de la familia, el tipo, y si será SSL el socket nuevo de los registros de la máquina. Se hace una conversión entre las constantes para familia y tipo de Unix a NachOS y luego se hace uso del llamado `socket` de Unix. Una vez obtenido el descriptor de Unix, este se convierte a uno de NachOS, y en el caso de un socket SSL, se procede a hacer la implantación de este descriptor en un nuevo contexto SSL. Se retorna el descriptor de NachOS del socket.

2.1.4. Connect

El llamado `Connect` negocia una conexión nuevo para un socket. La implementación utilizada lee los valores del descriptor del socket el cual conectar, una ubicación de memoria con la dirección donde conectarse, y el puerto al cual conectarse, de los registros de la máquina. Después de esto, se procede a confirmar que el socket descrito realmente exista en el sistema. Una vez hecho esto, se obtiene el descriptor de Unix para el socket, se lee de memoria la dirección donde conectarse, y se efectúa una conexión simple basado en la familia del socket. Si el socket es SSL, se procede además a hacer la conexión haciendo uso de las librerías OpenSSL para el contexto SSL. Se retorna el resultado de la conexión.

2.1.5. Close

El llamado `Close` cierra acceso a un descriptor de archivo. La implementación utilizada lee el valor del descriptor a cerrar de un registro de la máquina. Seguido, se confirma que este descriptor verdaderamente represente un archivo abierto a nivel de sistema. Una vez hecho esto, y recuperado el descriptor de Unix del archivo, se procede a hacer uso del llamado `close` de Unix. No se retorna nada.

2.2. client.c

La otra parte del desarrollo de la funcionalidad de esta entrega consistió en la elaboración de un programa de usuario que corre en el ambiente de NachOS. Este programa se conecta al servidor previamente construido y muestra el contenido enviado por el mismo. El programa es bastante simple. Primero, se usa `Socket` para generar un nuevo socket a utilizar con los parámetros para un socket TCP IPv4. Después, se llama `Connect` con la dirección del servidor, el puerto 7777, y el descriptor del socket. Seguidamente, se arma el request que se va a enviar al servidor, haciendo uso de los llamados `Write` y `Read` para solicitar del usuario el nombre de la figura a pedir por entrada y salida estándar. Hecho lo anterior, se usa `Write` para enviar el request al socket. Ya con el request enviado, se obtiene la respuesta del socket y se escriba esta respuesta a la salida estándar haciendo uso de los llamados `Read` y `Write`. Finalmente, se llama `Close` para cerrar el socket.

3. Resultados de Pruebas

Salida al enviar al servidor una figura válida:

```
--
horse
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 278
Connection: close

HORSE
5 pieces of brick 2x2 yellow
4 pieces of brick 2x4 white
2 pieces of brick 2x4 yellow
1 pieces of brick 2x6 yellow
1 pieces of brick 2x3 yellow
```

```
2 pieces of brick 1x1 yellow eye
2 pieces of brick 2x1 yellow
1 pieces of brick 2x2 white
Total amount of parts for Horse is: 18
Exited code 0
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 9591, idle 0, system 1010, user 8581
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 0
Paging: faults 0
Network I/O: packets received 0, sent 0

Cleaning up...
```

Salida al enviar al servidor una figura inválida:

```
--
aaaaa
HTTP/1.1 404 Not Found
Content-Type: text/html
Content-Length: 31
Connection: close

Could not find requested figure
Exited code 0
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 9591, idle 0, system 1010, user 8581
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 0
Paging: faults 0
Network I/O: packets received 0, sent 0

Cleaning up...
```

4. Conclusión

Con base en la experiencia de desarrollar un servicio de red utilizando servicios de sistema operativo construidos por los autores directamente, se ha logrado obtener una apreciación aún mayor del nivel de coordinación requerido tanto para la telecomunicación, como para implementar la funcionalidad clave de un sistema operativo. A pesar de que la funcionalidad desarrollado por los autores ultimádamente hizo bastante uso de los servicios regulares de Unix, el indagar en los específicos del diseño y la implementación de esta funcionalidad nos ha dado una perspectiva nueva sobre el esfuerzo que es parte de la programación de sistemas operativos.