Tarea 2: MySystemCall

Esteban-David Campos-Granados
Jose-Antonio Ortega-González
email:este0111@hotmail.com, ortega.josant@gmail.com
Área Académica de Ingeniería en Computadores
Instituto Tecnológico de Costa Rica

I. Introducción

Este proyecto se basa en el desarrollo de una llamada al sistema que realice un rastreo de todas las llamadas al sistema que son utilizadas durante la ejecución de un proceso desde que inicia hasta que finaliza. El programa realizado debe ser lo suficientemente robusto para evitar errores debido a la comunicación fuerte que mantiene el sistema operativo con el hardware.

Las llamadas al sistema es la interfaz entre los programas ejecutados y son rutinas especializadas que pueden ser solicitar por los procesos para que se ejecuten en modo kernel. Son proporcionadas por el sistema operativo, y su finalidad es proporcionar servicios del sistema operativo que los procesos puedan utilizar, sin darles permiso a estos de que se ejecuten en modo privilegiado o kernel. El sistema que implementamos precisamente pretende rastrear todas las veces que un proceso determinado solicite cierta rutina de estas.

Dos ejemplos de llamadas al sistema, y cuyas funciones son muy útiles para este proyecto, son **fork** y **ptrace**. El primero crea un proceso hijo con la misma imagen en memoria que el proceso que lo creó; mientras que la segunda le permite a un proceso observar y controlar la ejecución de otro proceso, incluso examinar y cambiar su memoria y registros.

El documento está dividido en ocho secciones, en la sección III se detallan todos los aspectos relacionados con el diseño y desarrollo de la aplicación implementada, además de su funcionamiento. En la sección IV se incluyen las instrucciones para poder ejecutar el proyecto en su propia computadora.

También se mencionan todas las especificaciones y condiciones del ambiente de desarrollo en la sección II, en la V la tabla de actividades que contiene todas las horas invertidas por cada miembro del grupo desarrollador y la coevaluación de cada uno en la sección VI. Finalmente, se dan las conclusiones en VII y las sugerencias, recomendaciones e ideas para trabajos futuros en VIII. La sección VIII es para las referencias.

II. Ambiente de desarrollo

El proyecto se realizó en el sistema operativo Linux, con la distribución Ubuntu 20.04 LTS (Focal Fossa), en el lenguaje de programación C sin el uso de ninguna librería externa. Se utilizó Visual Studio Code como IDE. El kernel de linux utilizado fué el 5.8.1.

La solución hace uso de un archivo del propio sistema operativo para realizar el mapeo de los nombres de las llamadas al sistema, esto ya que el método implementado solo rastrea las direcciones de los punteros en cuestión. Debido a esto, la arquitectura de la máquina debería ser de x86 (precisamente de 64 bits). Esto es una consideración importante para saber en qué registros o posiciones de memoria se encuentra la información de las llamadas al sistema en ejecución [1].

III. Detalles del diseño de la solución

La solución implementada puede ser organizada en 0 distintas secciones:

A. Llamada al sistema

La aplicación implementada no se desarrolló como una llamada al sistema, sino como un programa en C normal; esto debido a que se llegó a un acuerdo para poder utilizar métodos ya existentes como strace en Linux para rastrear las llamadas al sistema, que para utilizarlos se tenía que implementar como un programa totalmente externo al kernel.

B. Rastreo de llamadas

Para esta funcionalidad, se basó en el tutorial escrito por Nelson Elhage en su propio blog [2].

Para realizar el rastreo de las llamadas se utilizó un funcionamiento similar al método strace. Este método rastrea las llamadas al sistema que realiza un proceso y las imprime en pantalla. Hace uso de una llamada al sistema llamada **ptrace** para el rastreo. Es precisamente esta llamada **ptrace** la que se utilizó para el rastreo de nuestra solución.

Inicialmente se utiliza la llamada **fork** para iniciar el proceso hijo que será el que ejecutará el proceso que se le indica, mientras que el proceso padre esperará y realizará el rastreo de las llamadas. Además, se tiene que realizar una configuración del método **ptrace** para que solo notifique al proceso padre cuando el hijo termine su ejecución o cuando interactúe con llamas al sistema (ya sea cuando las solicita o cuando estas terminan).

El proceso padre estará en un ciclo esperando a cuando **ptrace** notifique el acceso o el retorno de una llamada al sistema solicitada por el proceso hijo, y saldrá del ciclo cuando el hijo termine su ejecución. De esta manera, se rastrean todas las llamadas utilizadas por el proceso hijo.

Para el uso de ambas llamadas al sistema se realizó un estudio de ellas en la documentación de llamadas de Linux que nos propone Michael Kerrisk [3].

C. Modos de operación

El programa rastrea todas las llamadas solicitadas por el proceso hijo, pero la información que se va a mostrar al usuario es diferente dependiendo el modo en el que se ejecute el programa:

- Modo interactivo: La aplicación imprimirá en pantalla la información de las llamadas al sistema solicitadas en el momento justo en el que son detectadas. Además imprimirá el valor de retorno de las llamadas en cuestión.
- Modo automático: La aplicación espera a que el proceso hijo termine su ejecución. En ese momento, imprime toda las llamadas realizadas y la cantidad de veces que fueron solicitadas.

Para el modo automático, se tiene un array dinámico (a base de punteros) de structs, los cuales guardan el ID de la llamada, su nombre y la cantidad de veces que se solicitó.

D. Obtención del nombre de las llamadas

Con **ptrace**, podemos hacer seguimiento de la dirección del puntero de una llamada al sistema en particular, más los argumentos de esta y su valor de retorno; no el nombre es esta.

Para realizar el tracking de este nombre se hace uso de un header que forma parte de los sistemas Linux con arquitecturas x86 de 64 bits. Este archivo es el siguiente:

/usr/include/x86_64-linux-gnu/asm/unistd_64.h

En este header se definen o nombran todas las llamadas al sistema junto con la dirección de su puntero. Por lo que se desarrolló un módulo que se encargará de encontrar el nombre de las llamadas con las direcciones que **ptrace** nos brinda. Para extraer el nombre darle formato a las llamadas de sistema, como parte del código implementado se adaptó el que se encuentra en [4].

E. Configuración

Para facilitar la compilación y el uso de la aplicación a los usuarios, se desarrollaron los debidos Makefiles del código fuente.

En la figura 1 se muestra un diagrama del flujo base del programa. En el diagrama se omite el modo de operación.

IV. Instrucciones de uso

Primero, debe descargar el archivo con la aplicación y descomprimirlo en el directorio que desee. Una vez hecho esto, se debe de ejecutar dentro de este directorio el siguiente comando:

```
make tracer
```

Este comando se encargará de ejecutar el Makefile que nos compilará el proyecto.

Para poder utilizar la aplicación hay que primero entender como pasarle el modo de operación y el proceso para realizarle el rastreo. El orden de los argumentos es el siguiente:

```
./tracer \{proceso\} \{modo\}
```

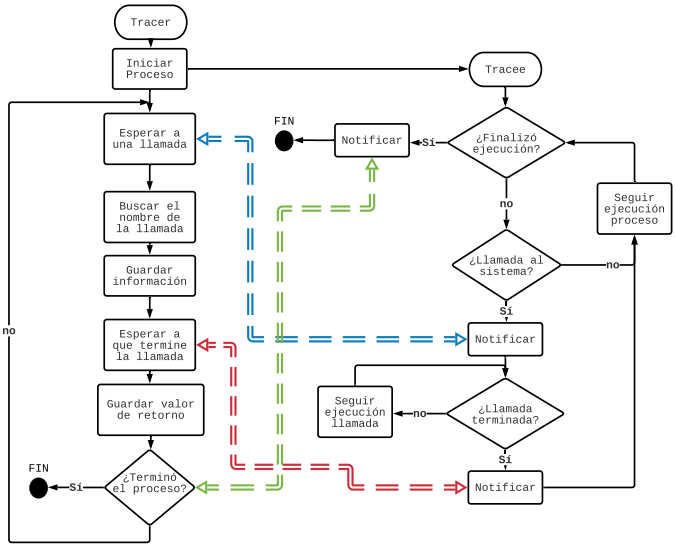

Fig. 1: Flujo principal de la comunicación entre el tracer y el proceso hijo (tracee). Las flechas de colores indican la ”comunicación” entre ambos mediante el **ptrace**.

Esto nos quiere decir que después de primero escribiremos la solicitud de ejecutar **tracer** (nuestro programa), luego pondremos el proceso a ejecutar y finalmente el modo. El programa no admite menos ni más argumentos.

Finalmente, existen dos modos de operación como mencionamos anteriormente, donde 0 corresponde al modo interactivo y 1 al modo automático.

Con todo esto, si queremos rastrear las llamadas al sistema que realiza un cierto programa *test* que se encuentra en la misma carpeta que el programa, y esto en el modo automático, tendremos que ejecutar lo siguiente:

```
./tracer ./test 1
```

V. Tabla de actividades

TABLE I: Horas dedicadas al proyecto por Jose Ortega

Día	Tema	Horas
4/10/2020	Investigación de syscalls y su obtención	4
5/10/2020	Investigación y desarrollo de algoritmo	5
6/10/2020	Continuación de algoritmo de detección de nombres	4
Total		13

TABLE II: Horas dedicadas al proyecto por Esteban Campos

Día	Tema	Horas
3/10/2020	Investigación sobre posibles métodos para implementar el proyecto. Estudio de las llamadas **ptrace** y **fork**.	3
4/10/2020	Implementación del rastreo de las llamadas al sistema que solicita un proceso usando **ptrace**.	5
5/10/2020	Implementación de los dos modos de operación del programa.	2.5
6/10/2020	Elaboración de la documentación externa.	4
Total		14.5

VI. Coevaluación

Se detallan los aspectos en cuanto al rendimiento y el aporte de los integrantes del equipo.

- Evaluación para Jose Ortega:
  - Comunicación: Jose siempre se mantuvo comunicado desde el momento en que el equipo empezó el trabajo y siempre reportaba lo que había realizado al final del día.
  - Adaptación: Se aseguró de entender todo el código que implementó además del funcionamiento de todo el sistema. Ademas, hizo la investigación necesaria para poder implementar funcionalidades considerando múltiples opciones posibles.
  - Responsabilidad: Cumplió con las tareas asignadas en los días que se le habían asignado.
  - Aporte: Todo lo que implementó en el proyecto fué funcional y suficientemente robusto para evitar futuros errores.
- Evaluación para Esteban Campos:
  - Comunicación: La coordinación es buena, se disponen tiempos para llamadas en línea utilizando la aplicación Google Meets y los avances son comunicados por escrito y a modo de audios por medio de la aplicación WhatsApp.
  - Adaptación: Si tener mucho conocimiento del funcionamiento interno, rápidamente se dispone para investigar y obtener los conocimientos necesarios para iniciar el diseño y la implementación del diseño planteado.
  - Responsabilidad: Se cumplen los objetivos y si se presentan problemas, se comunica con el debido tiempo para lograr llegar a la solución en equipo.
  - Aporte: El aporte es apropiado y la comprensión es buena, por lo que facilita la integración del código.

VII. Conclusiones

Al utilizar **ptrace** para realizar el rastreo de las llamadas del sistema, se observa lo estrechamente relacionado que es está el hardware con el sistema operativo. Para obtener los argumentos, memoria y demás recursos de una llamada al sistema se tuvo que tener en consideración la arquitectura de nuestras computadoras para considerar en que registros se encontrarían nuestros datos de interés y como realizar el mapeo de los nombres de las llamadas con la dirección de su puntero.

Otro aspecto muy importante fué la decisión de realizar el programa de C externo al kernel. Para poder implementar el programa como llamada al sistema hay que tomar en cuenta que solo se pueden usar los headers que la propia interfaz de programación del kernel nos brinda, por lo que el entorno puede ser mucho menos flexible para poderla implementar.

VIII. Sugerencias y recomendaciones

Como trabajo futuro pensamos en la elaboración de nuestra aplicación como llamada al sistema y no como un simple código de C, para poder observar las consideraciones del kernel que debemos considerar para su implementación.

Para la implementación como llamada al sistema se recomienda el estudio de la utilidad strace de Linux, ya que tiene una funcionalidad muy similar de la cual nos basamos para poder desarrollar el presente trabajo.

Referencias

[1] Ryan A. Chapman. Linux system call table for x86 64. http://blog.rchapman.org/posts/Linux_System_Call_Table_for_x86_64/, 2012. [En línea; publicado el 29 de noviembre].

[2] Nelson Elhage. Write yourself an strace in 70 lines of code. https://blog.nelhage.com/2010/08/write-yourself-an-strace-in-70-lines-of-code/, 2010. [En línea; publicado el 29 de agosto].

[3] Michael Kerrisk. Linux syscalls documentation - man7.org. https://man7.org/linux/man-pages/index.html, 2020. [En línea; consultado el 3 de octubre].

[4] Martin. Trim a string in c. http://www.martinbroadhurst.com/trim-a-string-in-c.html#:~:text=To%20left%20trim%20a%20string,move%20the%20terminating%20nul%20character., 2016.