Instituto Tecnológico de Costa Rica
Escuela de Ingeniería en Computación
Centro Académico de Alajuela

Tarea 1: Rastreador de System Calls

Estudiante: Jesús Cordero Díaz

Curso:
IC-6600: Principios de Sistemas Operativos

Grupo: 20

Profesor: Kevin Moraga Garcia

Fecha de entrega: 30 de marzo del 2025

Semestre: I

Contents

1 Introducción

El rastreo de llamadas al sistema (system calls) es un componente esencial para entender el funcionamiento interno de los sistemas operativos. En esta tarea se requiere desarrollar un programa en C para GNU/Linux que ejecute otro programa (denominado *Prog*) y rastree todas las system calls utilizadas durante su ejecución. El objetivo es capturar y mostrar un resumen acumulativo de dichas llamadas, facilitando la comprensión de la interacción entre el software de usuario y el kernel del sistema.

1.1 Objetivos del Proyecto

- Familiarizarse con el entorno POSIX y las llamadas al sistema.
- Desarrollar habilidades en programación en C y en la manipulación de procesos en GNU/Linux.
- Implementar un programa que ejecute otro proceso y capture las system calls invocadas.
- Generar un resumen detallado que muestre la frecuencia de cada system call utilizada.

1.2 Presentación del Problema

El desarrollo de software de bajo nivel en sistemas operativos requiere comprender la interacción entre el código de usuario y el núcleo del sistema. En este proyecto se plantea el desafío de crear un programa en C para GNU/Linux que, al ejecutar otro proceso (denominado *Prog*), capture y rastree todas las llamadas al sistema realizadas durante su ejecución. El problema central consiste en monitorizar de manera eficiente estas llamadas, identificar sus patrones de uso y generar un informe detallado que permita analizar el comportamiento del proceso a nivel del sistema.

2 Ambiente de Desarrollo

Para la implementación de la tarea se utilizaron las siguientes herramientas:

- **Visual Studio Code:** Editor de código ligero y multiplataforma que ofrece un amplio ecosistema de extensiones para facilitar la programación y la depuración.
- **C (GCC versión 13.3.0):** Lenguaje de programación de alto nivel ampliamente utilizado en el desarrollo de sistemas operativos y aplicaciones de bajo nivel, reconocido por su eficiencia y control sobre los recursos del sistema.
- **GitHub:** Plataforma de control de versiones basada en Git que facilita la colaboración y el seguimiento de cambios en el desarrollo del proyecto.
- **Ubuntu 24.04.2 LTS:** Sistema operativo Linux basado en Debian, conocido por su estabilidad y soporte a largo plazo, ideal para entornos de desarrollo.

3 Estructuras de Datos y Funciones

Esta sección describe las principales estructuras de datos y funciones implementadas en el código, las cuales permiten rastrear y contabilizar las llamadas al sistema (syscalls) efectuadas por un proceso.

3.1 Estructuras de Datos

- **SyscallCount:** Se define una estructura `SyscallCount` que almacena la información de cada llamada al sistema registrada.
  - `name`: Cadena de caracteres que representa el nombre de la syscall.
  - `count`: Entero que indica el número de veces que la syscall ha sido invocada.
- **Tabla de Syscalls:** Se utiliza un arreglo dinámico de tipo `SyscallCount` denominado `syscall_table` para almacenar cada registro de syscall.
- **Contador Total:** La variable `total_syscalls` lleva el seguimiento del número de entradas diferentes registradas en la tabla.

3.2 Funciones Principales

- **`get_syscall_name(long syscall_num)`:** Esta función recibe el número de una syscall y retorna su nombre utilizando el arreglo `syscall_names`. Se asegura de que el número de syscall esté dentro de un rango válido, devolviendo `"desconocido"` si no es así.
- **`update_syscall_count(long syscall_num)`:** Esta función actualiza el contador de una syscall en la `syscall_table`.
  - Primero, obtiene el nombre de la syscall llamando a `get_syscall_name`.
  - Luego, busca en la tabla si la syscall ya ha sido registrada.
  - Si la encuentra, incrementa el contador.
  - Si no, redimensiona la tabla, agrega una nueva entrada inicializada en 1, y actualiza el total de syscalls.
- **`cleanup(int sig)`:** Función que se encarga de liberar la memoria dinámica asignada a la tabla de syscalls y a los nombres registrados. Se invoca al recibir la señal SIGINT, permitiendo una salida limpia del programa.
- **`main(int argc, char *argv[])`:** Función principal que:
  - Procesa los argumentos de línea de comandos para habilitar modos de operación (detallado o pausa mediante las opciones `-v` y `-V`).
  - Valida la presencia de un programa a rastrear y, en caso afirmativo, crea un proceso hijo utilizando `fork()`.

- En el proceso hijo, utiliza `ptrace(PTRACE_TRACEME)` para permitir el rastreo y reemplaza su imagen mediante `execvp()`.
- En el proceso padre, se espera la ejecución del hijo, se extrae el número de syscall usando `ptrace(PTRACE_GETREGS)` y se actualiza el contador correspondiente.
- Finalmente, se imprime un reporte acumulativo del conteo de syscalls y se limpia la memoria.

4 Instrucciones para Ejecutar el Programa

La siguiente sección describe cómo ejecutar el programa rastreador, incluyendo la sintaxis de la línea de comandos, las opciones disponibles y ejemplos prácticos.

4.1 Sintaxis y Ejemplos

El programa se ejecuta desde la línea de comandos siguiendo la siguiente sintaxis:

```
rastreador [-v | -V] Prog [opciones de Prog]
```

Donde:

- `-v`: Activa el modo detallado, en el que se muestra información adicional sobre cada llamada al sistema detectada.
- `-V`: Activa el modo detallado y, adicionalmente, pausa la ejecución después de cada syscall, requiriendo que el usuario presione una tecla para continuar.
- `Prog`: Es el nombre o la ruta del programa que se desea ejecutar y rastrear.
- `[opciones de Prog]`: Son los argumentos que se le pasarán al programa `Prog`. El rastreador simplemente los reenvía sin analizarlos.

Ejemplo de uso:

```
./rastreador -v ls -la
```

En este ejemplo, el rastreador ejecuta el programa `ls` con las opciones `-la` y muestra información detallada sobre cada llamada al sistema.

4.2 Opciones de Ejecución

- **Modo Detallado (`-v`):** Al especificar la opción `-v`, el programa imprime en la salida estándar información sobre cada syscall detectada, incluyendo el nombre de la llamada y el identificador del proceso (PID).
- **Modo Detallado con Pausa (`-V`):** La opción `-V` combina el modo detallado con una pausa adicional en cada iteración, permitiendo al usuario presionar una tecla para continuar. Esto resulta útil para depurar o analizar el comportamiento del programa paso a paso.
- **Argumentos del Programa a Rastrear:** Todos los argumentos que siguen a `Prog` se envían directamente al programa que se desea rastrear. El rastreador no los procesa, simplemente los utiliza al ejecutar `execvp()` en el proceso hijo.

Para ejecutar correctamente el rastreador, es importante asegurarse de que se especifique al menos una opción (`-v` o `-V`) y que se incluya el nombre del programa a ejecutar. En caso de no cumplir con estos requisitos, el programa imprimirá un mensaje de error indicando la forma correcta de uso.

5 Actividades Realizadas por el Estudiante

A continuación se presenta un registro de las actividades realizadas durante el desarrollo del proyecto, detallando la fecha, descripción de la actividad y las horas invertidas en cada una.

Fecha	**Actividad**	**Horas**
21/03/2025	Revisión del enunciado de la tarea y planificación inicial del proyecto. Se definieron los objetivos y se elaboró el cronograma de trabajo.	2
22/03/2025	Configuración del ambiente de desarrollo: instalación y configuración de Visual Studio Code, Ubuntu, GitHub y compilador GCC para C.	2
23/03/2025	Análisis de la documentación del sistema y estudio de las llamadas al sistema (syscalls) relevantes para la tarea. Elaboración de un esquema preliminar del código.	3
24/03/2025	Diseño e implementación de la estructura principal del código, incluyendo la definición de la estructura `SyscallCount` y la tabla dinámica de syscalls.	4
25/03/2025	Desarrollo de funciones esenciales, como `get_syscall_name` y `update_syscall_count`, y pruebas iniciales de su funcionamiento.	3
26/03/2025	Implementación del manejo de señales (`cleanup`) y la integración de `ptrace` para rastrear las llamadas al sistema.	3
27/03/2025	Pruebas y validación del programa con diferentes programas a rastrear, revisión y depuración del código.	2
28/03/2025	Redacción de la documentación inicial, incluyendo la estructura y funciones principales, y ajustes finales al código.	2
29/03/2025	Revisión final del proyecto, empaquetado, generación del reporte de actividades y subida del código a GitHub.	1

Total de horas invertidas: 22 horas.

6 Autoevaluación

En esta sección se presenta una reflexión crítica sobre el desarrollo del proyecto, destacando el estado final del programa, los problemas encontrados durante la implementación, las limitaciones que se identificaron y un breve reporte de los commits realizados a lo largo del desarrollo.

6.1 Estado Final del Programa

El rastreador de llamadas al sistema se encuentra en un estado funcional, cumpliendo con los requerimientos planteados. El programa es capaz de:

- Ejecutar un proceso especificado y rastrear en tiempo real las syscalls que se invocan durante su ejecución.
- Actualizar correctamente el conteo de cada llamada al sistema, mostrando un reporte final acumulativo.
- Operar en modo detallado, tanto en el modo `-v` como en el modo `-V`, donde se permite una pausa para revisión manual.

6.2 Problemas Encontrados y Limitaciones

Durante el desarrollo se presentaron algunos desafíos, entre los que se destacan:

- **Gestión de la memoria dinámica:** La correcta administración de la memoria al redimensionar la tabla de syscalls requirió especial atención para evitar pérdidas de memoria y asegurar que cada cadena duplicada mediante `strdup` fuese liberada correctamente.
- **Interacción con `ptrace`:** Integrar las funciones de `ptrace` para capturar los registros del proceso rastreado presentó desafíos en cuanto a la sincronización entre el proceso padre y el hijo. Se realizaron pruebas exhaustivas para garantizar la estabilidad y precisión en la captura de syscalls.
- **Validación de argumentos:** Se implementaron validaciones robustas para asegurarse de que se proporcionen correctamente las opciones y el programa a rastrear, sin embargo, la gestión de casos de error específicos podría optimizarse en futuras versiones.

6.3 Reporte de Commits y Control de Versiones

El desarrollo del proyecto se realizó utilizando GitHub para el control de versiones, lo cual facilitó el seguimiento de los cambios y la resolución de errores. A continuación se presenta una tabla que resume los commits más relevantes, vinculados directamente con las actividades realizadas durante el proyecto:

Fecha	Commit	Descripción
21/03/2025	Commit Inicial	Configuración del entorno de desarrollo y creación del esqueleto del proyecto, estableciendo la base para el rastreador.
22/03/2025	Configuración y Ambiente	Integración de Visual Studio Code y ajustes iniciales en el repositorio, preparando el sistema para el desarrollo.
23/03/2025	Análisis y Diseño Preliminar	Revisión del enunciado y análisis de las syscalls, lo que permitió definir la estructura de datos y diseñar el esquema preliminar del código.
24/03/2025	Estructuras de Datos	Implementación de la estructura `SyscallCount` y la inicialización de la tabla dinámica para registrar las llamadas al sistema.
25/03/2025	Desarrollo de Funciones Básicas	Creación de las funciones `get_syscall_name` y `update_syscall_count` para el rastreo y conteo de syscalls.
26/03/2025	Integración de `ptrace` y Manejo de Señales	Incorporación de la lógica de `ptrace` para rastrear las syscalls y la implementación del manejo de señales mediante la función `cleanup`.
27/03/2025	Pruebas y Depuración	Realización de pruebas con distintos programas a rastrear y corrección de errores en la comunicación entre procesos.
28/03/2025	Ajustes y Redacción de Documentación	Integración de la documentación inicial en el repositorio y ajustes finales al código basados en la retroalimentación obtenida.
29/03/2025	Commit Final	Revisión final, empaquetado del proyecto, generación del reporte de actividades.

El uso de Git permitió revertir cambios problemáticos y mantener una historia clara del desarrollo, lo que facilitó la identificación y solución de errores, así como la incorporación de mejoras durante el proceso de desarrollo.

6.4 Evaluación según la Rúbrica

De acuerdo a la rúbrica de evaluación, se asignaron los siguientes porcentajes a cada componente del proyecto:

- **Opción `-v`:** 10%
- **Opción `-V`:** 20%
- **Ejecución de Prog:** 20%
- **Análisis de Syscalls:** 30%
- **Documentación:** 20%

Considerando los aspectos implementados y la calidad de la documentación, se considera que el proyecto cumple con los criterios de evaluación en forma adecuada.

7 Lecciones Aprendidas

Durante el desarrollo del proyecto se adquirieron valiosas lecciones que no solo fortalecieron la comprensión del funcionamiento de las llamadas al sistema, sino que también mejoraron las habilidades en programación en C y la gestión de procesos en entornos GNU/Linux. A continuación, se destacan las principales lecciones aprendidas:

- **Gestión de la Memoria Dinámica:** La utilización de funciones como `realloc` y `strdup` evidenció la importancia de gestionar correctamente la memoria. Se aprendió a evitar pérdidas de memoria mediante la liberación adecuada de recursos y a prever posibles fallos durante la redimensión de estructuras dinámicas.
- **Uso Eficiente de `ptrace`:** Integrar `ptrace` para rastrear las llamadas al sistema supuso un reto significativo. La experiencia permitió profundizar en el conocimiento de las interacciones entre procesos y la manipulación de registros del procesador, mejorando la capacidad para depurar y analizar el comportamiento de los procesos.
- **Control de Versiones y Colaboración:** El uso de GitHub no solo facilitó el seguimiento de los cambios y la reversión de errores, sino que también demostró ser una herramienta esencial para mantener una historia clara del desarrollo del proyecto. Esto permitió aprender la importancia de documentar cada paso y mejorar la colaboración en proyectos futuros.
- **Manejo de Errores y Validaciones:** La implementación de validaciones robustas en el análisis de argumentos y el manejo de señales mostró la necesidad de prever distintos escenarios de error. Esto llevó a reforzar las técnicas de programación defensiva, asegurando una mayor robustez y estabilidad del programa.
- **Iteración y Mejora Continua:** Las pruebas y la depuración constantes fueron clave para identificar y corregir errores en el programa. Se aprendió que la iteración continua y la revisión periódica del código son fundamentales para alcanzar un producto final de calidad.

8 Bibliografía

References

[1] R. Love, *Linux System Programming*, 1.ª ed., Sebastopol, CA, USA: O'Reilly Media, 2005.

[2] A. Silberschatz, P. B. Galvin y G. Gagne, *Operating System Concepts*, 10.ª ed., Hoboken, NJ, USA: Wilcy, 2020.