Tarea 1

- Estudiante: Pablo Alberto Muñoz Hidalgo
- Profesor: Kevin Moraga García
- Curso y Universidad: ITCR Sistemas Operativos
- Año: 2022

Introducción

El problema a resolver en esta ocasión es realizar un "syscall tracker" o rastreador de llamadas al sistema, este deberá rastrear las llamadas al sistema que realice un programa, tomando en cuenta parámetros que le introduzca el usuario, este rastreador tendrá dos modos ("-V" y "-v") los cuales tienen funciones diferentes. Una vez rastreados debe imprimir en terminal los resultados encontrados.

Ambiente de desarrollo

Se estará utilizando Ubuntu 20.04.4 LTS y como IDE se utilizará Visual Studio Code. Además de un repositorio en github.

Estructuras de datos usadas y funciones:

Estructuras:

- **Vectores:** Se utilizan vectores para almacenar la línea de mensaje separada en partes, también para alamacenar en tupla todos los syscalls encontrados y la cantidad de cada uno de estos.
- **Integers:** Se usan integers de diferentes tamaños para almacenar variables numericas
- **Strings o str:** Se usan strings o str para almacenar variables textuales, por ejemplo el nombre de un syscall.

Funciones:

- **fn comprobador (mut argumentos : Vec):** Función que recibe como entrada un vector de argumentos y sirve para comprobar que la entrada del usuario cumple con los requerimientos mínimos del sistema y se puede proceder con la ejecución, después de comprobar el mensaje llama a la función respectiva.
- **fn argumentos_comando (argumentos : Vec) -> Vec:** Función que recibe como entrada un vector de argumentos y devuelve un vector de argumentos, parece redudante, pero sirve para separar la primer parte del mensaje ya procesado de "opciones Prog" para luego pasarlo como argumentos de Prog.
- **fn rastreador(prog: &String, argumentos_prog: Vec):** Esta función recibe un string que va a ser el programa a ejecutar junto con un vector de argumentos de prog, esto llamará a "ejecutar_programa" para empezar a ejecutar el programa y de esta manera empezar a rastrearlo, después llama a "rastrear" para empezar a rastrear los syscalls.
- **fn rastreador_con_pausa (prog: &String, argumentos_prog: Vec):** Esta función recibe un string que va a ser el programa a ejecutar junto con un vector de argumentos de prog, esto llamará a "ejecutar_programa" para empezar a ejecutar el programa y de esta manera empezar a rastrearlo, después llama a "rastrear_con_pausa" para empezar a rastrear los syscalls agregando una pausa después de cada impresión de syscall.
- **fn rastrear(id_hijo: Pid) -> Vec<(String, i128)>:** Esta función recibe el id del hijo del proceso al que se le hizo fork en la función "rastreador" y lo empieza a rastrear para imprimir sus datos por cada syscall que se encuentre, además registra llama a la función "contador_syscalls" para ir registrando los syscalls que se usan e irlos contando, al final retorna este vector de syscalls.
- **fn rastrear_con_pausa(id_hijo: Pid) -> Vec<(String, i128)>:** Esta función recibe el id del hijo del proceso al que se le hizo fork en la función "rastreador_con_pausa" y lo empieza a rastrear para imprimir sus datos por cada syscall que se encuentre, además registra llama a la función "contador_syscalls" para ir registrando los syscalls que se usan e irlos contando, al final retorna este vector de syscalls. Por último llama a la función "pausa" para esperar un input del usuario y poder continuar con la ejecución.
- **fn contador_syscalls(syscall: &str, vector_syscalls: &mut Vec<(String, i128)>) -> Vec<(String, i128)>:** Esta función recibe el nombre de un syscall y un vector con syscalls ya existentes o bien un vector vacío, comprueba si el nombre que se introdujo está en el vector y si no lo está lo agrega, en caso de ser existente solo le suma 1 a la cantidad de ese syscall que se encontró. Retorna el vector con syscalls actualizado.
- **fn pausa():** Esta función no recibe nada ni devuelve nada, es solo el proceso de espera y continuación cuando se necesite que el usuario ingrese una tecla cualquiera.
- **fn main():** Esta función será la primera que se ejecute y comenzará preguntando al usuario por la línea de comandos, la

separa en un vector y llama a "comprobador" para continuar con la ejecución.

Instrucciones para ejecutar el programa:

Paso 1: Ejecutar el programa

Paso 2: Ingresar el texto "rastreador <*opción del rastreador*> <_programa a ejecutar_> <_argumentos para el programa a ejecutar (opcional)_>"

Paso 3: Presionar la tecla Enter *En caso de que se haya seleccionado la opcion de rastreador "-V" presionar la tecla enter despues de cada impresion en pantalla*

Ejemplo de linea a ejecutar "rastreador -v ls /home/user/Desktop/"

Actividades realizadas por estudiante:

Fecha	Hora de Inicio	Hora de Finalización	Actividad realizada
9/08/2022	7:00 PM	8:00 PM	Creación del git y descargar add-ons de Visual Studio Code
10/08/2022	7:00 PM	9:30 PM	Investigar sobre syscalls y su comportamiento
11/08/2022	6:00 PM	9:00 PM	Investigación sobre ptrace, strace y lurk, además como adaptarlo a Rust
14/08/2022	6:30 PM	9:30 PM	Primer commit, creación de la documentación y avance en syscalls, además de investigación en "split()"
14/08/2022	9:30 PM	11:00 PM	Hacer comprobaciones necesarias para el corrido del programa, el programa funciona sin argumentos en el "Prog"
15/08/2022	8:00 AM	10:00 AM	Se logró pasar argumentos al comando y se trabajó en las opciones de rastreador
15/08/2022	2:00 PM	5:00 PM	Se implementa el continuar presionando cualquier botón y se cuentan los syscalls de cada tipo, se suman y se imprimen en pantalla
15/08/2022	5:00 PM	8:30 PM	Release final con todo funcionando como lo solicita la especificacion
16/08/2022	9:00 AM	11:30 AM	Refinar detalles, comentar el codigo y terminar la documentación

Horas totales: 21 horas

Autoevaluación:

Estado del programa

El programa es encuentra en un perfecto estado, no tiene warnings ni errores, corre perfectamente y como lo solicita la especificación. El programa funciona si se siguen las intrucciones, en caso de que no se sigan este responderá con errores comprobados.

Problemas encontrados y limitaciones adicionales

Encontré un problema a la hora de realizar la pausa para continuar con cualquier tecla ya que muchas de la soluciones que se me venían a la cabeza implicaban que es usara la tecla "enter" para poder continuar, sin embargo encontré la biblioteca "termios" que me ayudó mucho en esta fase. También un problema que surjió fue a la hora de contar los syscalls, a pesar de que la solución fue sencilla al final el dolor de cabeza que me produjo fue bastante. Por último el sistema presentó errores a la hora de leer los syscalls, sin embargo con "linux personality" y "nix" se solucionó este problema. Considero que el programa no cuenta con limitaciones adicionales.

Evaluación

Opción -v	Opción -V	Ejecución de Prog	Análisis de syscalls	Documentación
10/10	20/20	20/20	30/30	20/20

Reporte de commits:

commit 7c402a74515e9358e8005966bb0b8f2f361b68e8 (HEAD -> main, origin/main, origin/HEAD) Author: Pablo Munoz Hidalgo 53487847+Litecore50@users.noreply.github.com Date: Tue Aug 16 11:17:11 2022 -0600

```
Octavo Avance v1.3 Final :)
```

commit 2de455702ccf97f5bf972b95f23708dac3487c46 Author: Pablo Munoz Hidalgo 53487847+Litecore50@users.noreply.github.com Date: Tue Aug 16 10:56:29 2022 -0600

```
Septimo avance 1.2 detalles y nombres de variables significativos
```

commit 1fc00db22c88cc6d398e97728765d022e4f4d273 Author: Pablo Munoz Hidalgo 53487847+Litecore50@users.noreply.github.com Date: Mon Aug 15 20:17:02 2022 -0600

```
Sexto avance v1.1 mejor presentacion y documentación
```

commit bfc5931d72a0f0d6123c966c2a12d116af39e1d4 Author: Pablo Munoz Hidalgo 53487847+Litecore50@users.noreply.github.com Date: Mon Aug 15 17:26:48 2022 -0600

```
Quinto avance v1.0, todas las funciones completas :)
```

commit 9e4670227730efaa8205a87b4adbf857b524c202 Author: Pablo Munoz Hidalgo 53487847+Litecore50@users.noreply.github.com Date: Mon Aug 15 16:38:04 2022 -0600

```
Cuarto avance v0.9, cuenta syscalls individuales y continua presionando cualquier boton
```

commit a8679014b9a1cdf070080e5c14c03e33791044d1 Author: Pablo Munoz Hidalgo 53487847+Litecore50@users.noreply.github.com Date: Mon Aug 15 09:49:21 2022 -0600

```
Código con comentarios
```

commit e9d89e6ba251a2c8d392eac47231b7c98a167b65 Author: Pablo Munoz Hidalgo

53487847+Litecore50@users.noreply.github.com Date: Mon Aug 15 09:32:36 2022 -0600

```
Tercer Avance funcional con opciones de rastreador
```

commit 7d5fb26e98dd54b53910945a6a0ba70fdee20f34 Author: Pablo Munoz Hidalgo
53487847+Litecore50@users.noreply.github.com Date: Sun Aug 14 22:52:22 2022 -0600

```
Segundo avance funcional
```

commit 713023acc729bf349d62c6bd5142a9c1b3835dbf Author: Pablo Munoz Hidalgo
53487847+Litecore50@users.noreply.github.com Date: Sun Aug 14 20:56:21 2022 -0600

```
Primer avance
```

commit c069d102ccacb3ff2addccd4af723d08c05b1202 Author: Pablo Munoz Hidalgo
53487847+Litecore50@users.noreply.github.com Date: Sun Aug 14 20:54:57 2022 -0600

```
Bibliografia
```

commit fa892870e3663cebb76575419b674fca85c9441d Author: Pablo Munoz Hidalgo
53487847+Litecore50@users.noreply.github.com Date: Sun Aug 14 20:50:44 2022 -0600

```
Create README.md
```

commit 9a0b1f93d3bb0821e6b1959e21aac68f110edd5a Author: Pablo Munoz Hidalgo
53487847+Litecore50@users.noreply.github.com Date: Sun Aug 14 19:36:53 2022 -0600

```
Primer Commit de prueba
```

Lecciones Aprendidas:

En esta tarea se aprendió el funcionamiento de los "syscalls" y como se comportan estos a la hora de ejecutar un programa, además de aprender lo básico sobre estos también se sacó provecho del lenguaje Rust, un lenguaje útil e intuitivo teniendo en cuenta experiencias pasadas con C y Java. Por último se aprecia mucho la aplicación de los conocimientos adquiridos en las lecturas, ya que muchas veces esto se queda en teoría pero con esta tarea todo pasó a la parte práctica y es muy satisfactorio haberla finalizado exitosamente.

Bibliografía:

[1] "lib.rs.html -- source". Docs.rs. https://docs.rs/linux/0.0.1/src/linux/lib.rs.html#1-21 (accedido el 16 de agosto de 2022).

[2] "Communicating with the OS - The Node Experiment - Exploring Async Basics with Rust". Site not found · GitHub Pages. https://cfsamson.github.io/book-exploring-async-basics/3_1_communicating_with_the_os.html (accedido el 16 de agosto de 2022).

[3] "strace(1): trace system calls/signals - Linux man page". Linux Documentation. https://linux.die.net/man/1/strace (accedido el 16 de agosto de 2022).

[4] "linux::syscall - Rust". Docs.rs. https://docs.rs/linux/0.0.1/linux/syscall/index.html (accedido el 16 de agosto de 2022).

[5] "System programming in Rust, take 2". 128nops and counting. https://carstein.github.io/2022/05/29/rust-system-programming-2.html (accedido el 16 de agosto de 2022).

[6] "GitHub - JakWai01/lurk: A pretty (simple) alternative to strace". GitHub. https://github.com/JakWai01/lurk (accedido el 16 de agosto de 2022).

[7] "Implementing strace in Rust". Jakob Waibel. https://jakobwaibel.com/2022/06/06/ptrace/ (accedido el 16 de agosto de

2022).

[8] "How to Split a String in Rust? (Explained with Examples)". Become A Better Programmer - Trust The Process. https://www.becomebetterprogrammer.com/split-string-rust/ (accedido el 16 de agosto de 2022).

[9] "What are some good ways to implement a read line or a sleep timer in Rust". Stack Overflow. https://stackoverflow.com/questions/66823720/what-are-some-good-ways-to-implement-a-read-line-or-a-sleep-timer-in-rust (accedido el 16 de agosto de 2022).

[10] "How can I read one character from stdin without having to hit enter?" Stack Overflow. https://stackoverflow.com/questions/26321592/how-can-i-read-one-character-from-stdin-without-having-to-hit-enter (accedido el 16 de agosto de 2022).

[11] "Storing Lists of Values with Vectors - The Rust Programming Language". Learn Rust - Rust Programming Language. https://doc.rust-lang.org/book/ch08-01-vectors.html (accedido el 16 de agosto de 2022).