Tarea #1

Instituto Tecnológico de Costa Rica Carrera: Bachillerato en Ingeniería en Computación **Curso:** Principios de Sistemas Operativos **Profesor:** Kevin Moraga García **Alumno:** Jonathan Quesada Salas **Carnet:** 2020023583 **Tarea Corta 1:** Rastreador de System Calls

---

Introducción

De desarrollará un rastreador de syscalls por medio del lenguaje de programación Rust, la cual tendrá 3 posibles parámetros en el formato de *rastreador [opciones rastreador] Prog [opciones de Prog]* :

1. En cuanto respecta a **opciones rastreador** serán dos opciones -v y -V, las cuales tendrán en común la acción de desplegar un mensaje por cada Syscalls Calls detectado, pero con la opción -V será diferente con el hecho que se deberá presionar cualquier tecla para poder continuar la ejecución de Prog hasta que el punto que se pueda recorrer la totalidad de los syscalls junto son su resumen de cuantas veces fue utilizado cada syscall en un resumen.
2. La opción **Prog** se puede remplazar por cualquier syscall que pueda existir, como por ejemplo de Progs, pueden ser: ls, rm y move.
3. En las opciones de prog puede ser consideradas como pasadas para la inicialización de Prog en cuanto respecta a su ejecución.

Ambiente de desarrollo

Con lo que respecta al ambiente de desarrollo se usará el sistema operativo Ubuntu, específicamente la versión:

```
Ubuntu 20.04.2 LTS
```

En cuanto al IDLE se usará el **Visual Studio Code** para el desarrollo de la tarea.

Adicionalmente se usará un repositorio de Github para poder almacenar los avances significativos de la asignación de la primera tarea de principios de sistemas operativos llamado tarea1.

Estructuras de datos usadas y funciones

Con lo que respecta a la estructura de datos usada se pueden describir los siguientes módulos:

1. argument
2. argument_parser
3. arguments_summary
4. runner
5. tracer

Y adicionalmente a los módulos existiran tres archivos importantes para la ejecución de la asignación:

1. Cargo.toml
2. main.rs

Empezando por el módulo ***argument*** se pueden destacar cuatro archivos de extensión *.rs*, los cuales son los siguientes y adicionalmente se comentará la función de cada uno:

1. mod.rs: Se encargará de definir los módulos con visibilidad pública. Los cuales son:
    - t_argument
    - program_argument
    - tracer_option_argument
2. **program_argument.rs:** Este archivo tendrá 3 funciones:
    - **new:** Para poder generar espacio en memoria para los argumentos que tenga el comando ingresado
    - **get_text:** Para obtener el texto de la linea de comando
    - **summarize_argument:** Se declara el resumen de los argumentos de la linea de comando
3. **t_argument.rs:** Este archivo es una es una interfaz que solo define 2 métodos
    - get_text
    - summarize_argument
4. **tracer_option_argument.rs:** Este archivo contendrá una estructura de datos y 3 funciones que son las siguientes:
    - La **estructura de datos** se define como TracerOptionArgument es almacenar la información de los argumentos que tenga la linea de comando en caso que exista -v y -V.
    - Y las funciones serían:
        - **new:** Crea una implementacion de TracerOptionArgument para almacenar la información de los argumentos
        - **get_text:** Se obtiene el texto del texto ya confirmado en caso que tenga las sentencias -v y -V.
        - **summarize_argument:** En dado caso que se haya encontrado un syscall se levanta una bandera para indicar que se encontró.

Por otra parte analizando el módulo ***argument_parser*** se pueden apreciar tres archivos de extensión *.rs*, los cuales son los siguientes y adicionalmente se comentará la función de cada uno:

1. **mod.rs:** Este archivo declara 2 módulos públicos.
    - t_parser
    - parser
2. **parser.rs:** El cual solo tendrá una función llamada ***parse*** el cual su función es parsear el mensaje que haya obtenido del usuario en la terminal por medio del comando ingresado para luego poder utilizarlo en próximas funciones y módulos
3. **t_parser.rs:** Este archivo es una interfaz que se define un método el cual es **parse**

De igual manera se tiene otro módulo llamado **arguments_summary** en este dicho módulo se puede apreciar solo un archivo, el cual es el siguiente adicionalmente se comentará la función o funciones del archivo:

1. **mod.rs:** Este archivo contendrá una estructura de datos y una función:
    - En cuanto a la estructura de datos llamada ***ArgumentsSummary*** se puede apreciar la siguiente:
        - **print_syscall_found:** Esta parte nos ayudará para -v y -V ya que es en caso que se encuentre el syscall-

        - **pause_when_syscall_found:** Esta parte de la estructura nos servirá para determinar cuando usar el -V-
        - **program_command:** El cual consiste en el strig que uno ingrese en la terminal para la ejecución del programa.
    - Ahora bien, con lo que respecta a la función se puede apreciar ***summarize*** la cual consiste en:
        - Tomar los argumentos de agument_parser y summary, para recibir una lista de argumentos y resumirlos en un objeto llamado ArgumentsSummary, para que el tracer no se preocupe en los argumentos y solo llame a ArgumentsSummary de la información que necesita.

Con lo que respecta al módulo ***runner*** se pueden encontrar 2 archivos, los cuales explicados a continuación:

1. **mod.rs:** Se crea un modulo publico t_runner el cual servirá en la ejecución del programa
2. **t_runner.rs:** Es una interfaz que define runner para la ejecución

Y con lo que se puede apreciar con el último módulo llamado ***tracer*** se aprecian 2 archivos de extensión *.rs*, los cuales se explicarán a continuación y adicionalmente explicando las funciones de cada uno:

1. **mod.rs:** En este archivo se pueden contemplar 7 funciones y una estructura de datos:
    - Con lo que respecta a la estructura de datos llamada Tracer, la cual tiene dos atributos:
        - **arguments_summary:** los argumentos datos en el módulo arguments_summary se contienen en esta estructura.
        - **sys_calls_summary:** Será un hashmap para el resumen de todos los syscalls.
    - Ahora adentrando en las funciones se pueden apreciar las siguientes y adicionalmente una explicación de su función.
        - **new:** Se crean nuevas instancia para almacenar toda información aportada por parte de arguments_summary y sys_calls_summary explicadas anteriormente.
        - **add_sys_call:** Esta función se encarga de ir añadiendo cada system call en una lista para luego, esa misma lista es pasada a las funciones de print para que los impriman continuamente.
        - **print_sys_call_summary:** Esta función se encarga de imprimir el resumen de los syscalls diciendo la cantidad de veces que fueron usados
        - **print_sys_call:** Esta función se encarga de imprimir los system calls en su totalidad, a partir del módulo de system_call_names.rs que este mismo contiene la totalidad de los system calls
        - **run_tracer:** La función run_tracer consistira en ejecuta las instrucciones dependiendo que sea la instrucción -v y -V ya que estos mismos parámetros puede variar su funcionalidad.
        - **run_tracee:** Funcion que analiza el comando con los parametros que estan después del -v y -V-ç.
        - **execute:** Se declara execute el cual se encargará de ejecurar los hilos de ejecución padre e hijo.

2. **system_call_names.rs:** Este archivo es una variable estática pública la cual contendrá todos los syscall names almancenados en la variable **SYSTEM_CALL_NAMES**

En cuanto respecta a los archivos adicionales que no se encuentran en algún módulo se pueden apreciar los siguientes:

1. **main.rs:** Este archivo tendrá una función llamada *main* la cual se encargará de dar el procedimiento lógico al problema definido anteriormente.
2. **Cargo.toml:** Este viene predeterminado con Rust, solo que para la realización de esta asignación se puede contemplar que se añadió las siguientes dependencias:

```
[dependencies]
execute = "0.2.8"
nix = "0.25.0"
linux-personality = "1.0"
termion = "1.5.5"
libc = "0.2"
```

Instrucciones para ejecutar el programa

1. Primeramente se debe de cargar el programa en un ambiente virtual, en este caso será Visual Studio Code.
2. Ya habiendo cargado dicho folder en el IDLE se procederá a abrir la terminal en la dirección de dicho proyecto.
3. Abriendo dicho proyecto en la dirección específica de donde estará se procederá en poner en terminal el siguiente comando para compilar el proyecto:

   ```
   cargo build
   ```

4. Ya habiendo compilado el proyecto se procederá a ingresar el siguiente comando en terminal según la preferencia del usuario a la hora de usar -v o -V:

   ```
   ./target/debug/sys_call_tracer -V ls
   ```

   Por poner un ejemplo, otro comando posible sería:

   ```
   ./target/debug/sys_call_tracer -V rm
   ```

Actividades realizadas por estudiante

Fecha	Inicio	Fin	Avance Realizado
9/08/2002	3:00 pm	5:00 pm	Leer documentación de Rust
9/08/2002	7:00 pm	10:00 pm	Ver vídeos tutoriales de Rust
10/08/2002	8:00 pm	11:00 pm	Crear ambiente virtual y leer documentación de Rust
11/08/2002	9:00 pm	10:00 pm	Instalar IDLE para Rust

13/08/2002	4:00 pm	7:00 pm	Investigar e implementar el funcionamiento de fork y avanzar la documentación
14/08/2002	9:00 am	12:00 md	Investigar de ptrace, syscalls y avanzar la documentación
15/08/2002	8:00 am	2:00 pm	Implementación de los módulos argument, argument_parser, arguments_sumary y runner
15/08/2002	5:00 pm	9:00 pm	Redactar la documentación con el avance realizado y crear el módulo tracer

---

Autoevaluación

Estado final

Con lo que respecta al estado final del programa considero personalmente en lo que se refiere a la funcionalidad de -v y -V y el rastreador en cuanto al análisis de los syscalls, que tiene como objetivo mostrar la cantidad respectiva del uso de cada syscall muestran una completitud consistente, sin embargo en mi visón personal considero que la ejecución de **prog** podría fallar con algunas opciones, como por ejemplo mv, ya que dicha funcionalidad lo llegué a probar con opciones reducidas para una mejor agilización para el desarrollo de la tarea 1.

Problemas y limitaciones

1. Primeramente una de las limitaciones que me encontré con la tarea fue tener que aprender el lenguaje de programación Rust, ya que era un lenguaje totalmente nuevo, el cual su sintaxis es difícl de comprender a primera vista.
2. El hecho de syscall el concepto lo manejaba, pero llegar a implementar algo que hiciera uso de eso era totalmente diferente.
3. El aprendizaje ptrace y de la manera que se deben de ver los syscalls en terminal para poder comprender como se debe de ver la finalización del programa.
4. El aprendizaje de fork para poder crear los hilos de ejecución hijo y padre, ya que tambien fue un problema en la realización de la tarea.
5. Para poder contar las veces que se usa un syscall con un determinado prog, por ejemplo con "rm".
6. Ahora del análisis y del diseño de la tarea añadí muchas interfaces que no llegaron a ser útiles en la tarea.
7. A la hora de crear la estructura de tracer se me presentó un problema ya que quería crear dos atributos de la estructura, los cuales eran arguments_summary y sys_calls_summary, pero de manera global, pero se me presentaron varios errores de compilación, entonces dicha estructura fue agregada a la funcion que estaba necesitando desde un principio.
8. Un problema muy latente en el transcurso de la realiación de la tarea fue poder encontrar los métodos que llegaba a necesitar en el momento sin saber que existian, como por ejemplo stdout(), traceme(), flush() etc...
9. En cuanto al manejo de *HashMap* se me presentó el problema que en las llamadas en las impresiones de los syscalls no llegaban a imprimirse correctamente ya que cometía el error de almacenarlo en una estructura externa a la función o simplemente no seguía la sintaxis correcta del HashMap.

Reporte de commits

```
git log - en terminal
```

Calificación

Rubro	Porcentaje
Opción -v	10%
Opción -V	20%
Ejecución de Prog	15%
Análisis de Syscalls	30%
Documentación	20%
***	

Lecciones Aprendidas

En cuanto a las lecciones aprendidas con lo que respecta a esta tarea pueden ir de lo más simple a algo más complejo, pero son las siguientes:

1. Primeramente la lección más grande que puedo destacar es el tener que estudiar y comprender un lenguaje de programación e implementar un programa funcional en dicho programa en un periodo corto de tiempo, ya que el lenguaje de programación Rust nunca lo había escuchado ni visto su sintaxis.
2. Acorde con la enseñanza 1, se puede hilar también es que esta tarea me obligó a tener que leer más exhaustivamente y con un mayor detenimiento los manuales que pudiera encontrar en internet ya que la información que se lograba encontrar no era tan abundante como en otros lenguajes de programación.
3. Ya más específicamente adentrando un poco más en el objetivo de la tarea fue el hecho de hacer un programa rastreador de syscalls, ya que esto mismo lo había leído en el libro, pero no se me ocurría una forma de programar algo parecido.
4. Algo adicional a tomar en cuenta es que en la resolución de la tarea fue indispensable investigar e implementar ptrace ya que este mismo es la llamada al sistema ptrace() proporciona un medio por el cual un proceso (el "rastreador") puede observar o controlar la ejecución de otro proceso (el "rastreador"), y examinar y cambiar la memoria y los registros del rastreador. Se utiliza principalmente para implementar la depuración de puntos de interrupción y el seguimiento de llamadas del sistema.
5. Tener que investigar de una manera profunda para poder encontrar módulos útiles en la realización de la tarea como puede ser el caso de **libc**, **termion**, **nix**, **execute** y **linux_personality**. Cada una se explicará a continuación su funcionamiento:
   - **libc:** Enlaces FFI sin formato a las bibliotecas del sistema de las plataformas
   - **termion:** Termion es una biblioteca sin enlaces de Rust puro para el manejo, la manipulación y la lectura de información de bajo nivel sobre terminales. Esto proporciona una alternativa completa a Termbox.
   - **nix:** Nix utiliza características de Cargo para habilitar la funcionalidad opcional. Se pueden habilitar en cualquier combinación.
   - **execute:** Esta biblioteca se usa para extender Command para ejecutar programas más fácilmente.

    - **linux_personality:** Esta caja es un envoltorio seguro de tipo alrededor de la función de personalidad de Linux.

6. Algo que personalmente me llegó a dar bastantes problemas fue utilizar Unsafe en las funciones y cuando tener que usarlas. Con lo que respecta a Unsafe es en que Unsafe en Rust existe porque, por naturaleza, el análisis estático es conservador. Cuando el compilador intenta determinar si el código mantiene o no las garantías, es mejor que rechace algunos programas válidos que aceptar algunos programas no válidos; y una de las aclaraciones que me llegó a servir fue que aunque el código puede estar bien, si el compilador de Rust no tiene suficiente información para estar seguro, rechazará el código.
7. Algo importante que logré aprender es que inicial pensaba que Rust era un lenguaje muy estático para el manejo del mismo, pero en realidad se puede llegar a comparar con java, no en cuanto a la sintaxis, sino a la creación de interfaces e implementaciones que necesitan las funciones para poder cumplir con el "contrato" de una interfaz, entonces apreciar esa similitud, es bastante apreciada por mi persona.
8. Adicionalmente fue el aprendizaje que se necesitaba de fork() para la realización de la tarea, ya que este mismo va a consistir en realizar dos hilos de ejecución los cuales será el padre y el hijo, estos mismos hilos de ejecución serán iguales. Básicamente se usa para crear un nuevo proceso duplicando el proceso de llamada. El nuevo proceso se denomina proceso hijo. El proceso de llamada se denomina proceso padre. Tanto el proceso secundario como el principal tienen su id de proceso único.
9. Por último una de las enseñanzas de esta tarea que hizo que pudiera completar la mayoría de ella fue el uso del ptrace en Rust, ya que esto se usa para las opciones de ptrace y el rastreador puede configurar el rastreo para que se detenga en ciertos eventos. Esta enumeración se utiliza para definir esos eventos tal como se definen

---

Bibliografía

[1] Tech With Tim. Rust Tutorial #1 - Introduction To Rust Programming. (9 de mayo de 2022). Accedido el 9 de agosto de 2022. [Video en línea]. Disponible: https://www.youtube.com/watch?v=T_KrYLW4jw8

[2] Tech With Tim. Rust Tutorial #2 - Using Rust Tools (cargo, rustfmt). (11 de mayo de 2022). Accedido el 9 de agosto de 2022. [Video en línea]. Disponible: https://www.youtube.com/watch?v=gvgBUY8iNO4

[3] Tech With Tim. Rust Tutorial #3 - Variables, Constants and Shadowing. (15 de mayo de 2022). Accedido el 9 de agosto de 2022. [Video en línea]. Disponible: https://www.youtube.com/watch?v=xYgfW8cIbMA

[4] Tech With Tim. Rust Tutorial #4 - Data Types. (16 de mayo de 2022). Accedido el 9 de agosto de 2022. [Video en línea]. Disponible: https://www.youtube.com/watch?v=t047Hseyj_k

[5] Tech With Tim. Rust Tutorial #5 - Console Input. (17 de mayo de 2022). Accedido el 9 de agosto de 2022. [Video en línea]. Disponible: https://www.youtube.com/watch?v=PQBX-ev5q2k

[6] Tech With Tim. Rust Tutorial #6 - Arithmetic and Type Casting. (20 de mayo de 2022). Accedido el 9 de agosto de 2022. [Video en línea]. Disponible: https://www.youtube.com/watch?v=Uwa3P9dBHdA

[7] Tech With Tim. Rust Tutorial #7 - Conditions and Control Flow (if/else if/else). (1 de junio de 2022). Accedido el 9 de agosto de 2022. [Video en línea]. Disponible: https://www.youtube.com/watch?v=MOa7ulhNYc0

[8] Tech With Tim. Rust Tutorial #8 - Functions, Expressions & Statements. (4 de junio de 2022). Accedido el 9 de agosto de 2022. [Video en línea]. Disponible: https://www.youtube.com/watch?v=APrANyLHCtQ

[9] Steve Klabnik and Carol Nichols, The Rust Programming Language. Covers Rust 2018, 2019. Accedido el 9 de agosto de 2022. [En línea]. Disponible: https://kolegite.com/EE_library/books_and_lectures/Програмиране/Rust/The%20Rust%20programming%20language.pdf

[10] "std::result - Rust". Learn Rust - Rust Programming Language. https://doc.rust-lang.org/std/result/ (accedido el 14 de agosto de 2022).

[11] "Install Rust". Rust Programming Language. https://www.rust-lang.org/tools/install (accedido el 9 de agosto de 2022).

[12] "The Rust Programming Language - The Rust Programming Language". Learn Rust - Rust Programming Language. https://doc.rust-lang.org/book/ (accedido el 9 de agosto de 2022).

[13] "Introduction - Rust By Example". Learn Rust - Rust Programming Language. https://doc.rust-lang.org/rust-by-example/ (accedido el 10 de agosto de 2022).

[14] "nix::sys::ptrace::traceme - Rust". Docs.rs. https://docs.rs/nix/0.18.0/nix/sys/ptrace/fn.traceme.html (accedido el 10 de agosto de 2022).

[15] "Unsafe Operations - Rust By Example". Learn Rust - Rust Programming Language. https://doc.rust-lang.org/rust-by-example/unsafe.html (accedido el 10 de agosto de 2022).

[16] "Dependencies - Rust By Example". Learn Rust - Rust Programming Language. https://doc.rust-lang.org/rust-by-example/cargo/deps.html (accedido el 11 de agosto de 2022

[17] "stdout in std::io - Rust". Learn Rust - Rust Programming Language. https://doc.rust-lang.org/stable/std/io/fn.stdout.html (accedido el 15 de agosto de 2022).

[18] "linux_personality - Rust". Docs.rs. https://docs.rs/linux-personality/latest/linux_personality/ (accedido el 15 de agosto de 2022).

[19] "nix::sys::ptrace::traceme - Rust". Docs.rs. https://docs.rs/nix/0.18.0/nix/sys/ptrace/fn.traceme.html (accedido el 15 de agosto de 2022).

[20] "stdout in std::io - Rust". Learn Rust - Rust Programming Language. https://doc.rust-lang.org/stable/std/io/fn.stdout.html (accedido el 15 de agosto de 2022).

[21] "Implementing strace in Rust". Jakob Waibel. https://jakobwaibel.com/2022/06/06/ptrace/ (accedido el 13 de agosto de 2022).

[22] "Linux System Call `fork()` in Rust". Knoldus Blogs. https://blog.knoldus.com/linux-system-call-fork-in-rust/ (accedido el 13 de agosto de

2022).

[23] "Fork in fork - Rust". Docs.rs. https://docs.rs/fork/latest/fork/enum.Fork.html (accedido el 13 de agosto de 2022).

[24] "execute - Rust". Docs.rs. https://docs.rs/execute/latest/execute/ (accedido el 15 de agosto de 2022).

[25] "nix - Rust". Docs.rs. https://docs.rs/nix/latest/nix/ (accedido el 15 de agosto de 2022).

[26] "termion - Rust". Docs.rs. https://docs.rs/termion/latest/termion/ (accedido el 15 de agosto de 2022).

[27] "libc - Rust". Docs.rs. https://docs.rs/libc/latest/libc/ (accedido el 15 de agosto de 2022).