tarea 1 — Rastreador de system calls (rust, linux)

estudiante: Sarah Quesada - 2021046027

curso y periodo: sistemas operativos — II 2025

repositorio: https://github.com/sarah03qc/rastreador-syscalls-rust.git

binario: adjunto en el .zip de classroom

---

1. introducción

En esta asignación se implementa un rastreador de llamadas al sistema (parecido a `strace`) escrito en rust para gnu/linux. El programa ejecuta un binario objetivo (en adelante **prog**) y con `ptrace`, toma cada **entrada** y **salida** de las system calls hechas por ese proceso, se tienen dos modos de operación:

- `-v`: muestra cada system call con la mayor cantidad posible de detalles (nombre o número, argumentos y valor de retorno)
- `-V`: igual que `-v`, pero **para** en cada system call hasta que el usuario presione una tecla

al finalizar la ejecución de **prog**, el rastreador imprime una **tabla** con el conteo total de cada system call observada, el objetivo es poder entender el flujo de ejecución a nivel de kernel y practicar el uso de `ptrace` y también la gestión de procesos en user-space

---

2. ambiente de desarrollo

- **sistema operativo:** ubuntu (virtualbox) x86_64
- **kernel:** Linux sarahpqc-VirtualBox 6.8.0-48-generic #48-Ubuntu SMP PREEMPT_DYNAMIC Fri Sep 27 14:04:52 UTC 2024 x86_64 x86_64 x86_64 GNU/Linux
- **rust:** rustc 1.89.0 (29483883e 2025-08-04)
- **cargo:** cargo 1.89.0 (c24e10642 2025-06-23)
- **dependencias (Cargo.toml):**
  - `nix` (con features activadas, `ptrace`, `process`, `signal`)
  - `libc`
  - `clap` (derive) para la línea de comandos
  - `termios` para la pausa interactiva
  - `anyhow` para manejo simple de errores

---

3. estructuras de datos usadas y funciones

estructuras de datos principales - `Args` (deriva de `clap::Parser`): define la interfaz cli con los campos:

- `verbose: bool` (`-v`)
- `verbose_pause: bool` (`-V`)
- `prog: String` (programa objetivo)
- `prog_args: Vec<String>` (argumentos de **prog**, se toman como "trailing var arg")
- `HashMap<i64, u64>`: acumulador de conteos por número de syscall
- `Vec<CString>`: construcción del `argv` para `execvp`
- `libc::user_regs_struct`: estructura de registros del proceso hijo obtenida con `PTRACE_GETREGS`

funciones más importantes - `main()`: 1. parsea cli con `clap` y arma `argv`

2. `fork()` del proceso
3. en el **hijo**: `ptrace(TRACEME)`, `SIGSTOP`, `execvp(prog, argv...)`
4. en el **padre**: `waitpid` inicial, `ptrace::setoptions` (incluye `PTRACE_O_TRACESYSGOOD`), bucle que alterna **entrada/salida** de los syscalls con `ptrace::syscall`, impresión (`-v`/`-V`) y el conteo, al final imprime la tabla acumulada ordenada

- `syscall_num(regs)`: devuelve el número de syscall (x86_64: `orig_rax`)
- `syscall_ret(regs)`: devuelve el valor de retorno de la syscall (x86_64: `rax`)
- `syscall_args(regs)`: devuelve los seis args según el ABI x86_64 (rdi, rsi, rdx, r10, r8, r9)
- `syscall_name(n)`: mapeo parcial número->nombre para syscalls comunes (si no está mapeada, se ve `sys_<n>` en el resumen)
- `read_string_from_child(pid, addr, max_len)`: trata de leer una cadena terminada en `\0` del espacio del hijo usando `process_vm_readv` (mejora la legibilidad de `execve/open/openat`)
- `wait_keypress()`: configura la terminal en modo no canónico y sin eco para que pare en `-V`
- `read_regs(pid)`: obtiene registros del hijo con `PTRACE_GETREGS`

---

4. instrucciones para ejecutar el programa

compilación (x86_64):

```
cargo build --release
```

sintaxis general:

```
rastreador [opciones-del-rastreador] -- Prog [opciones-de-Prog]
```

> nota: `--` hay que separar las opciones del rastreador de los argumentos de **Prog**, si **Prog** usa flags que empiezan con `-` (como los `ls -l`), poner `--` evita que el parser del rastreador los confunda con sus propias opciones, por eso en mi caso debemos usarlo así

opciones del rastreador: - `-v` -> imprime cada syscall con detalles - `-V` -> como `-v`, pero se para en cada syscall hasta que se presione una tecla

consultas concretas usadas (problema del enunciado y otros casos):

```
# 1) eco simple con detalle (muestra execve, mmap/brk, write=5, exit_group)
./target/release/rastreador -v -- /bin/echo hola

# 2) listar la raíz con formato largo (ocupamos el -- para separar -l)
./target/release/rastreador -v -- /bin/ls -l /

# 3) modo pausado por syscall (inspeccion paso a paso)
./target/release/rastreador -V -- /bin/echo hola

# 4) caso negativo (ruta inexistente) para observar retornos -errno (p. ej. -2 = ENOENT)
./target/release/rastreador -v -- /bin/ls /no-existe

# 5) ejecucion simple sin flags del rastreador
./target/release/rastreador -- /bin/true
```

---

5. actividades realizadas por estudiante (timesheet)

> resumen breve por tarea, fecha y horas

Fecha	Actividad	Horas
2025-08-27	Lectura y análisis del enunciado	0.5
2025-08-27	Investigación, análisis y familiarización con rust	1.0
2025-08-28	Preparación de ambiente (rustup, deps, verificación de versiones)	0.5

Fecha	Actividad	Horas
2025-08-29	Implementación completa del tracer (fork/exec/ptrace, entrada/salida, lectura de registros y cadenas, -v/-V), robustez (opciones ptrace y señales), formato de tabla final, correcciones de build y pruebas	6.0
2025-08-30	Pruebas manuales y revisión final	0.6
2025-08-30	Documentación final en markdown y generación de pdf	2.0
Total		**10.6**

---

6. autoevaluación

estado final: El rastreador ejecuta **Prog** con sus argumentos, tiene los modos `-v` y `-V`, imprime detalles de cada syscall (incluida lectura de algunas cadenas cuando sea posible) y da un resumen acumulado ordenado al finalizar, compila y funciona en ubuntu x86_64

problemas encontrados y soluciones: - faltaban features de `nix` para ptrace/process/signal -> se activaron en `Cargo.toml` - se confundían paradas de syscall con señales -> se configuró `PTRACE_O_TRACESYSGOOD` y se reenvían señales normales - valores `-38 (ENOSYS)` observados inicialmente -> se alterna mas estricto entrada/salida solo en `WaitStatus::PtraceSyscall` - flags de **Prog** (p. ej. `-l` de `ls`) interferían con el parser que usé -> se documenta y usa `--`

limitaciones: - mapeo número->nombre de syscalls parcial, entonces cuando falta se imprime `sys_<n>` - decodificacion de argumentos complejos acotada (solo algunas rutas comunes)

reporte de commits de git: - 870d110 2025-08-30 Documentación markdown agregada

- 99dc299 2025-08-29 rastreador rust con -v/-V y tabla de syscalls

calificación según rubrica: - opcion `-v` (10%): 10/10

- opcion `-V` (20%): 20/20
- ejecución de Prog (20%): 20/20
- analisis de syscalls (30%): 28/30
- documentación (20%): 19/20

7. lecciones aprendidas

- **arrancar simple y validar rapido**: mejor primero hacer que el tracer ejecute `prog` y cuente syscalls, ya despues agregar `-v` y por ultimo `-V`, ir en pasos pequeños para evitar bloquearse
- **distinguir entrada/salida de syscall correctamente**: usar `ptrace::setoptions` con `PTRACE_O_TRACESYSGOOD` y alternar entre `WaitStatus::PtraceSyscall` para leer `orig_rax` (entrada) y `rax` (salida), asi evitamos confundir señales con syscalls y el tipico `-38 (enosys)`.
- **separar bien las opciones del rastreador y de prog**: con `clap` la convencion `--` evita que flags de `prog` (p. ej. `-l` de `ls`) rompan el parser.
- **conocer la abi de mi arquitectura**: en x86_64 los argumentos van en `rdi, rsi, rdx, r10, r8, r9`, saber esto nos puede ahorrar horas de "por que sale basura??"
- **manejo de señales**: reenviar señales normales (`WaitStatus::Stopped(_, sig)`) en lugar de tratarlas como syscalls, asi no atascamos al hijo.
- **documentar mientras programo**: estar haciendo comentarios pequeños de lo que voy haciendo y guardar ejemplos que me ayuden a guiarme baja el tiempo de redaccion final, a la hora de redactar documentación.
- **usar git desde el primer momento**: esta es una que se me olvidó hacer, pero ya lo aprendí efectivamente (usualmente sí lo hago), pero en este caso me emocioné y empecé a desarrollar la solución a la tarea, y cuando me di cuenta, ya lo tenía practicamente listo y no había creado el repositorio, yo sé que es una mala practica, por lo que seré más consciente al respecto en futuras tareas y proyectos.

---

8. bibliografia

man-pages y referencias de linux - ptrace(2) — man7.org

- process_vm_readv(2) — man7.org
- execve(2) — man7.org
- syscalls(2) — indice general — man7.org
- wait(2)/waitpid(2) — man7.org
- signal(7) — man7.org
- strace(1) — man7.org
- x86-64 system v abi psABI (convencion de llamadas y registros)

documentacion de rust y crates usados - the rust programming language (rust book)

- cargo book
- nix crate — docs.rs
- clap crate — docs.rs
- termios crate — docs.rs
- anyhow crate — docs.rs
- std (io, ffi, os::unix) — doc.rust-lang.org

control de versiones y entregables - pandoc — manual (para pasar de markdown a pdf)