Rastreador de syscalls

Rastreador de syscalls

> **Curso:** Principios de Sistemas Operativos
>
> **Asignación:** Rastreador de syscalls
>
> **Profesor:** Kevin Moraga Garcia
>
> **Estudiante:** Anthony Josue Rojas Fuentes
>
> **Fecha:** 02/09/2025
>
> **Repositorio:**
>
> https://github.com/ajrojasfuentes/Rastreador_de_systemcalls.git

---

1. Introducción

El objetivo de esta asignación es construir un rastreador de llamadas al sistema (syscalls) en GNU/Linux, similar a una versión simplificada de `strace`, utilizando el mecanismo de depuración `ptrace(2)`. El rastreador debe ejecutar un programa objetivo (`Prog`), interceptar todas sus syscalls, y al finalizar mostrar un resumen acumulado con el conteo por tipo de syscall. Asimismo, deben implementarse dos modos de salida detallada:

- `v`: Muestra **cada evento de syscall** (entrada y salida) con la **máxima cantidad de detalles posible** (nombre, argumentos relevantes, valor de retorno y `errno` si aplica).
- `V`: Igual que `v`, pero **pausando** tras cada evento hasta que el usuario presione una tecla.

Enfoque propuesto

El diseño se basa en el patrón clásico de `ptrace`:

1. El proceso padre (el **trazador**) realiza `fork()`.
2. El proceso hijo ejecuta `ptrace(TRACEME)` y luego `execve(Prog, argv...)`.
3. El padre usa `waitpid()` y `ptrace(PTRACE_SYSCALL)` para **alternar stops** en **entrada** y **salida** de cada syscall, leyendo registros con `PTRACE_GETREGS`.
4. Se decodifican valores (número → nombre de syscall, argumentos y retorno), se lleva un **contador** por llamada, y se imprime un **resumen** al finalizar.

Este enfoque permite una instrumentación **transparente** (no requiere modificar `Prog`), es **portable** dentro de Linux/x86_64, y cumple las restricciones de no requerir privilegios especiales al trazar un **hijo**.

---

2. Ambiente de desarrollo

- **SO:** Ubuntu 24.04 LTS (Linux x86_64)
- **Compilador/Toolchain:** Rust stable (e.g. `rustc 1.89.0`, `cargo 1.89.0`)
- **IDE:** RustRover (JetBrains)
- **Dependencias (crates):**
  - `nix` (ptrace, wait, señales)
  - `libc` (constantes SYS_*, errno, structs)
  - `clap` (CLI)
  - `crossterm` (captura de tecla para `V`)
  - `once_cell`, `anyhow`, `thiserror` (utilitarios)

Compilación (release)

```
cargo clean
cargo build --release
```

Binario resultante: `target/release/rastreador`.

---

3. Estructuras de datos y funciones principales

Estructuras

- **`Opts`** (CLI, con `clap`):
  - `verbose: bool` (activa `v`), `very_verbose: bool` (activa `V` e implica `v`).
  - `prog: String` (programa objetivo), `args: Vec<String>` (argumentos de `Prog`; se pasan tal cual).
- **`ThreadState`**: Estado por TID (hilo/proceso) del trazado.
  - `entering: bool` — alterna entrada/salida de syscall (inicialmente `true`).
  - `last_syscall: u64` — guarda el número de syscall visto en la entrada previa.
- **Contadores**:
  - `HashMap<u64, u64>` — mapa número_syscall → conteo.

Módulos/Funciones (resumen)

- **`child_exec(opts)`**: Hijo llama `ptrace(TRACEME)` y `execvp(Prog, argv)`.
- **`parent_trace(child_pid, opts)`**: Bucle principal del trazador; configura `PTRACE_O_TRACESYSGOOD`, alterna `PTRACE_SYSCALL`, maneja `waitpid` y actualiza contadores.
- **`ptrace_getregs(pid)`**: Envuelve `PTRACE_GETREGS` y retorna `user_regs_struct`.
- **`log_sys_enter(pid, scno, regs)`**: Imprime entrada de la syscall con decodificación especial para `execve` (ruta y `argv`) y `openat` (flags y modo), o genérica para otras.
- **`log_sys_exit(pid, scno, ret)`**: Imprime salida con valor de retorno; si es negativo en `[-4095..-1]`, se mapea a `errno`.
- **`sysdecode::syscall_name(n)`**: Tabla (ampliada) número → nombre de syscall; si no se encuentra, produce `"sys_<n>"`.
- **`sysdecode::{read_c_string, read_ptr}`**: Lectura segura de memoria del hijo con `ptrace::read` (para imprimir cadenas como rutas y `argv`).
- **`sysdecode::{fmt_flags_open, decode_errno}`**: Decodifica flags de `open/openat` y varios `errno`.

- `read_argv_preview(pid, argv_ptr, max_items)`: Muestra una vista limitada de `argv[]` en `execve`.
- `wait_keypress()`: Habilita modo raw con `crossterm` y espera cualquier tecla (para `V`).
- `print_summary(counts, total)`: Imprime la tabla acumulada (nombre, conteo, porcentaje) y el total.

Mapeo de registros x86_64

- **Entrada**: `orig_rax` = número de syscall.
- **Argumentos**: `rdi`, `rsi`, `rdx`, `r10`, `r8`, `r9`.
- **Salida**: `rax` = valor de retorno (negativo con `errno` si falla).

> Nota sobre alternancia: Al primer PtraceSyscall de cada TID, el estado debe considerarse entrada. Se inicializa entering = true para evitar el conteo erroneo de sys_0.

---

4. Instrucciones para ejecutar el programa

Sintaxis general

```
./rastreador [opciones_del_rastreador] Prog [opciones_de_Prog]
```

Opciones del rastreador

- `v`: Modo **verboso** (imprime cada syscall con detalle).
- `V`: Modo **muy verboso** (como `v`, pero pausando por tecla tras cada evento).

> Para evitar ambigüedad con los argumentos de Prog, se recomienda usar -- como separador:
>
> ```
> ./target/release/rastreador -v -- ls -l /
> ```

Ejemplos básicos (enunciado)

```
# Ejecución mínima (solo resumen final)
./target/release/rastreador -- /bin/true

# Verboso: ver cada syscall de ls -l /
./target/release/rastreador -v -- ls -l /

# Muy verboso: pausando en cada evento
./target/release/rastreador -V -- /bin/echo "hola"
```

Casos de prueba adicionales

A) Programa Rust de prueba (`sysplay`) – cubre archivos, memoria, pipes, sockets UNIX, temporización y aleatorio.

Ejecutar:

```
./rastreador/target/release/rastreador -v -- ./prog/target/release/sysplay
```

Esperado: ver `openat`, `write`, `fsync`, `lseek`, `fstat`, `mmap/mprotect/munmap`, `pipe2`, `poll`, `read`, `socketpair`, `getrandom`, `nanosleep`, `readlink`, `unlink`, `close`, etc.

B) Script de Bash (`prog.sh`) – prioriza **builtins** para que el trabajo ocurra dentro del propio bash; también invoca procesos externos para generar `fork/execve/wait4`.

```
#!/usr/bin/env bash
set -Eeuo pipefail
trap 'echo "[trap] SIGINT capturada, seguimos..."' INT
umask 022
orig="$PWD"
cd /tmp

tmp="rt_bash_$$"
exec {out}> "$tmp.txt"
printf 'Linea 1: %s\n' "hola" >&"$out"
printf 'Linea 2: epoch=%(%s)T\n' -1 >&"$out"
exec {out}>&-
```

```
exec {in}< "$tmp.txt"
IFS= read -r -t 0.2 -u "$in" line1 || true; echo "leido-1: $line1"
IFS= read -r -t 0.2 -u "$in" line2 || true; echo "leido-2: $line2"
exec {in}<&-

[[ -e "$tmp.txt" ]] && echo "existe: $tmp.txt"
cd "$orig"; umask 077
kill -s INT $$ || true
ls -l /proc/self/fd >/dev/null 2>&1 || true
: > "/tmp/$tmp.empty"
rm -f "/tmp/$tmp.txt" "/tmp/$tmp.empty" || true
```

Ejecutar:

```
chmod +x ./prog.sh
./rastreador/target/release/rastreador -v -- ./prog.sh
```

Esperado: `chdir`, `umask`, `openat/read/write/close`, `rt_sigaction/rt_sigprocmask`, `kill/tgkill`, `fork/execve/wait4` (por `ls` / `rm`), y `pselect/select` implícito por `read -t`.

---

5. Actividades realizadas por el/los estudiante(s) (timesheet)

> Formato: [Fecha] — [Horas] — [Descripción breve]

Ejemplo (plantilla):

```
29/08/2025 — 2.0h — Lectura: lectura del enunciado y definición de alcance.
29/08/2025 — 3.5h — Prototipo ptrace: fork/TRACEME/exec y bucle SYSCAL
L.
30/08/2025 — 2.0h — Decodificación básica: nombres de syscalls + tabla ini
cial.
31/08/2025 — 2.5h — Modo -v: impresión de entrada/salida.
01/09/2025 — 1.5h — Lectura de cadenas (PEEKDATA), openat/execve detalla
```

dos.
01/09/2025 — 1.0h — Modo -V: pausa con crossterm en raw mode.
02/09/2025 — 1.0h — Resumen/tabla y ordenamiento por frecuencia.
02/09/2025 — 1.5h — Pruebas con programas de estrés (Rust + Bash).
02/09/2025 — 1.0h — Documentación y empaquetado.

Total: 16.0 h

---

6. Autoevaluación

Estado final del programa:

- Implementa ejecución de `Prog` y seguimiento de sus syscalls.
- `v` imprime entrada/salida con argumentos clave y `errno`.
- `V` añade pausa por tecla (modo raw) tras cada evento.
- Resumen final con conteo y porcentaje por syscall.

Problemas encontrados y limitaciones:

- **Alternancia entrada/salida**: Requiere inicializar `entering = true` para cada TID nuevo; de lo contrario aparece `sys_0` espuria (corregido).
- **Cobertura**: La decodificación detallada se centró en `execve` y `openat`. Se puede ampliar (`read/write`, `mmap` flags, `clone/exec` argv completo, `sockaddr` en `connect/accept`).
- **Seguimiento de hijos/hilos**: Esta versión **no** sigue procesos hijos/hilos creados por `fork/clone`. Puede ampliarse con `PTRACE_O_TRACEFORK|VFORK|CLONE`.
- **Buffers grandes**: Al leer cadenas del hijo se impone un **límite** (p. ej., 4096 bytes) para seguridad.

Reporte de commits:

035c2fd 2025-09-02 Anthony Josue Rojas Fuentes Initial commit
8d93c59 2025-09-02 ajrojasfuentes rastreador-v.0.1
8189dfb 2025-09-02 ajrojasfuentes add prog and bugfixes

Calificación propuesta:

- Opción `v` (10 %): **10/10** — Entradas y salidas formateadas, argumentos clave, errno.
- Opción `V` (20 %): **20/20** — Pausa confiable por tecla.
- Ejecución de Prog (20 %): **20/20** — Manejo de `execvp` y mensajes claros en fallo.
- Análisis de Syscalls (30 %): **26/30** — Tabla final correcta; buena decodificación en `execve/openat`; (espacio para agregar detalles como descripciones de la syscall).
- Documentación (20 %): **20/20** — Estructura completa y clara.

---

7. Lecciones aprendidas

- `ptrace` **es un "ping-pong"**: Cada syscall genera dos stops (entrada/salida). La **alternancia** por TID es crucial.
- `TRACESYSGOOD` **evita falsas detecciones**: Diferenciar stops de syscalls de otras señales simplifica la lógica.
- **Leer memoria del hijo con límites**: `PTRACE_PEEKDATA` permite recuperar rutas y `argv`, pero hay que acotar y manejar errores.
- **Señales**: Reinyectar señales no relacionadas con syscall para no alterar el comportamiento de `Prog`.
- **Separación de argumentos con** `-`: Evita conflictos entre flags del rastreador y de `Prog`.
- **Salida**: Usar `stderr` para el trazo y `stdout` para el resumen final ayuda a redirigir y analizar.

---

8. Bibliografía

- **Páginas de manual (Linux):** `man 2 ptrace`, `man 2 waitpid`, `man 2 execve`, `man 2 openat`, `man 2 mmap`, `man 2 read`, `man 2 write`, `man 7 signal`, `man 2 getrandom`.

- **ABI x86_64 (llamadas y registros):** *System V Application Binary Interface, x86-64 Architecture Processor Supplement*.
- **Documentación de crates:** `nix` y `libc` (docs.rs).
