

Modificaciones al sistema operativo XV6

Integrantes:

- | | |
|--------------------------------|-------------|
| • Vladimir Roger Ticona Mamani | 2023-119063 |
| • Jorge Enrique Obando Huallpa | 2017-130045 |
| • Edu Rubinho Puma Ccama | 2023-119053 |

Objetivos

- **Objetivo general**

- Extender la funcionalidad del sistema operativo XV6 mediante la implementación de mecanismos de instrumentación, monitoreo y análisis para comprender su funcionamiento interno.

- **Objetivos específicos:**

- Instrumentación de Syscalls
- Comandos de Monitoreo
- Contador de Invocaciones

Introducción

- **¿Qué es XV6?**
 - Sistema operativo educativo desarrollado por el MIT
- **¿Por qué XV6?**
 - El tamaño de código es manejable, es de código abierto y educativo para enseñanza
- **Conceptos Fundamentales:**
 - System Calls(Syscalls)
 - Modo Usuario vs Modo Kernel
 - Gestión de Procesos

ENTREGABLE 1

- Activación/desactivación mediante comando `trace`
- Muestra nombre de la syscall
- Indica el PID del proceso
- Se ejecuta de forma transparente

Captura 1:

Se prueba el comando antes de ver los resultados

```
ENSEÑO
$ echo hola
hola
```

Captura 2:

Se prueba el comando para ver los resultados

```
$ trace 1
Syscall tracing ACTIVADO
[TRACE] PID 15: exit
[TRACE] PID 2: write
$[TRACE] PID 2: write
[TRACE] PID 2: read
echo hola
[TRACE] PID 2: read
[TRACE] PID 2: fork
[TRACE] PID 2: wait
[TRACE] PID 16: sbrk
[TRACE] PID 16: exec
[TRACE] PID 16: write
h[TRACE] PID 16: write
o[TRACE] PID 16: write
l[TRACE] PID 16: write
a[TRACE] PID 16: write
```

ENTREGABLE 2

• Comandos para ver la informacion del sistema

Agregamos dos nuevas syscalls en el kernel:

- numprocs: que cuenta cuantos procesos estan activos
- getmem: que te dice cuanta memoria tiene asignada el proceso actual

Luego creamos dos programas de usuario:

- uptime.c: Este programa te muestra cuanto tiempo lleva el sistema corriendo. Obtiene los ticks de reloj del sistema (XV6 cuenta muy rapido, unos 100 ticks por segundo), los convierte a segundos y minutos, y los muestra en pantalla junto con el numero de procesos activos.
- psmem.c: Este programa muestra informacion del proceso que lo ejecuta. Te dice el PID, cuanta memoria tiene asignada (en bytes y kilobytes), y cuanto tiempo lleva el sistema funcionando

```
$ uptime
== INFORMACION DEL SISTEMA XV6 ==
Tiempo de ejecucion: 1057 ticks
Tiempo formateado: 0 minutos 10 segundos
Numero de procesos activos: 3
=====
```

```
$ psmem
===== INFORMACION DEL PROCESO ACTUAL =====
PID del proceso: 5
Tiempo de sistema: 1567 ticks
Tiempo formateado: 0 minutos 15 segundos
```

ENTREGABLE 3

• CONTAR CUANTAS VECES SE USA CADA SYSCALL

- Agregamos un arreglo en el kernel llamado `syscall_count` con 26 espacios (uno para cada syscall). Lo inicializamos en ceros.
- Implementamos una nueva syscall llamada "syscount" que te permitia consultar estos contadores desde un programa de usuario.
- Finalmente, creamos un programa llamado `syscountcmd.c` que te mostraba estos contadores. Si lo ejecutabas sin argumentos, veias una tabla con todas las syscalls y sus contadores. Si escribias "syscountcmd 5", te mostraba solo el contador de la syscall numero 5 (que es `read`).

Cuando ejecutabas "ls", veias que `fork` subia en 2, `exec` subia en 2, `read` subia en muchos, `write` subia mucho (porque tiene que mostrar el resultado en pantalla), y `close` subia varios. Esto nos ayudo a visualizar exactamente que hace XV6 cuando ejecutas un comando.

RESUMEN DE INVOCACIONES DE SYSCALLS		
ID	Nombre	Invocaciones
1	fork	6
2	exit	0
3	wait	4
4	pipe	0
5	read	82
6	kill	0
7	exec	7
8	fstat	24
9	chdir	0
10	dup	2
11	getpid	0
12	sbrk	5
13	sleep	0
14	uptime	0
15	open	27
16	write	2073
17	mknod	1
18	unlink	0
19	link	0
20	mkdir	0
21	close	25
22	trace	0
23	numprocs	0
24	getmem	0
25	syscount	74


```
$ syscountcmd 1
```

==== CONTADOR DE INVOCACIONES ====
Syscall: fork (ID: 1)
Invocaciones: 7

Demostracion de los comandos

CONCLUSIONES

El proyecto nos permitió comprender de manera práctica cómo funcionan los sistemas operativos: vimos que las syscalls son la puerta entre el modo usuario y el kernel, que cada comando se ejecuta en un proceso separado mediante fork, exec y wait, y que la memoria de cada proceso está completamente aislada. Observamos además la gran cantidad de actividad que ocurre incluso con comandos simples y entendimos la importancia de validar la seguridad y el correcto funcionamiento de estas operaciones. La experiencia nos mostró que los sistemas operativos no son magia, sino código que, al comprender sus conceptos básicos, podemos modificar y extender, consolidando así nuestra transición de la teoría a la práctica.

XV6

**xv6: A Simple,
Unix-like Teaching
Operating System**

*Russ Cox, Frans Kaashoek,
Robert Morris*

**¡Muchas
gracias!**

