



UNIVERSIDAD NACIONAL JORGE BASADRE GROHMANN

# Extensiones al Kernel de XV6: Un Kit de Herramientas para Diagnóstico y Monitoreo del Sistema

Proyecto Final de Sistemas Operativos

---

Arratia Paz, Russell Jhean Paul (2023-119004)

Carita Pinchi, Nelly Adriana (2023-119007)

Asignatura: Sistemas Operativos

Docente: MsC. Hugo Manuel Barraza Vizcarra

---

Tacna - Perú, 2025

# El Objetivo: Transformando XV6 de una Caja Negra a un Kernel Transparente



XV6 Caja Negra  
IBM Plex Sans



Kernel Transparente  
IBM Plex Sans

El sistema operativo XV6 es una plataforma educativa excepcional. Sin embargo, para comprender a fondo su comportamiento, es necesario observar sus operaciones internas en tiempo real. Este proyecto se centró en construir un conjunto de herramientas integradas al kernel para instrumentar, consultar y auditar su funcionamiento.



**Instrumentar el Kernel:**  
Monitorear y registrar las llamadas al sistema (syscalls) para obtener visibilidad del comportamiento dinámico del sistema.

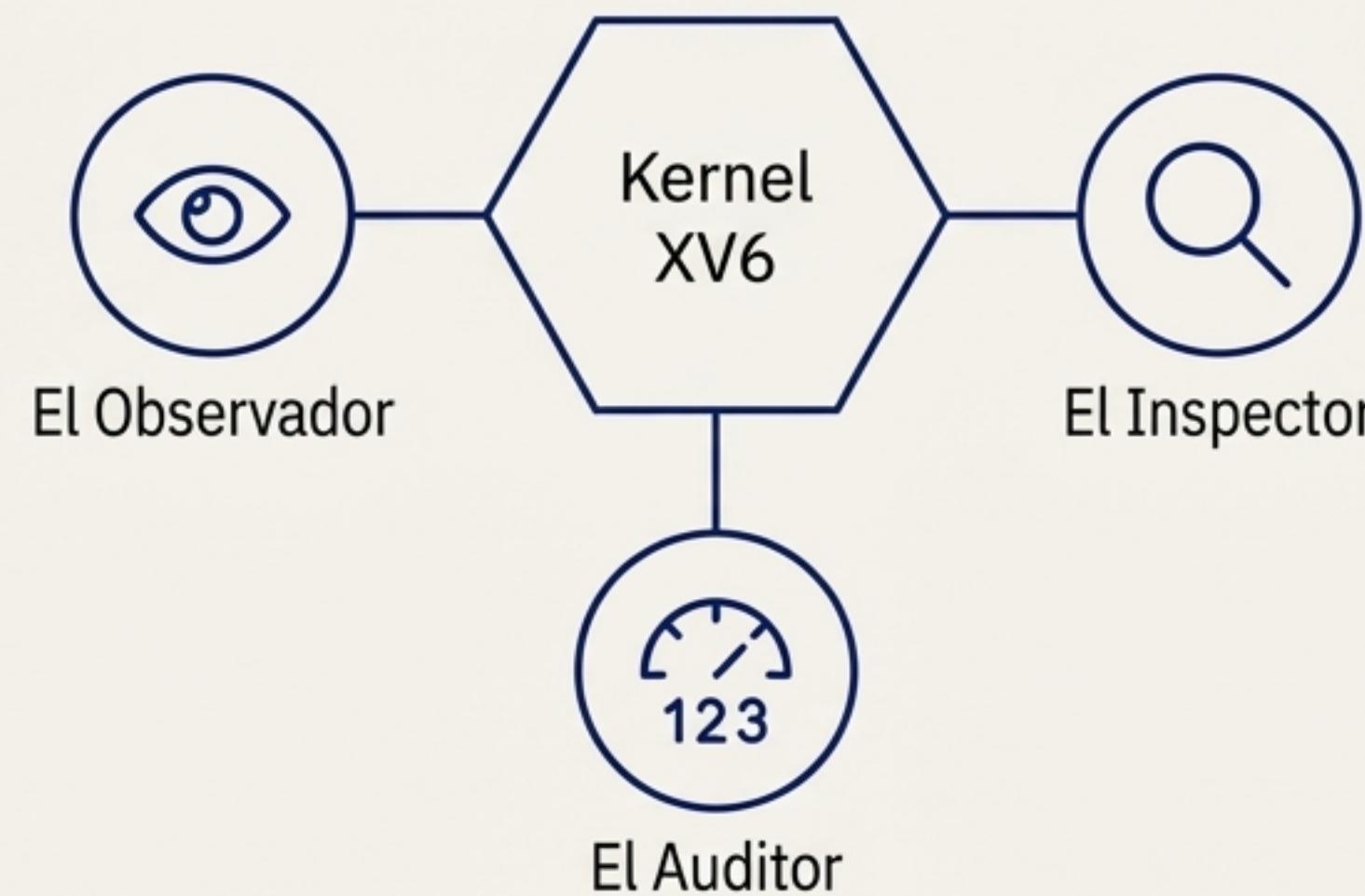


**Extender la Funcionalidad:**  
Crear nuevas syscalls que permitan a los programas de usuario acceder de forma segura a información interna del kernel.



**Desarrollar Comandos de Usuario:** Construir herramientas de línea de comandos que utilicen las nuevas capacidades del kernel para presentar información útil y auditabile.

# Nuestro Kit de Herramientas para el Kernel de XV6



1. **El Observador ('trace')**: Una herramienta para monitorear en tiempo real cada llamada al sistema, mostrando su nombre y parámetros. Permite ver la interacción directa entre los programas y el kernel.
2. **El Inspector ('schedinfo', 'uptime')**: Un conjunto de comandos para consultar el estado actual del sistema, incluyendo el listado de procesos activos, sus estados (RUNNING, SLEEPING, etc.) y el tiempo de actividad del sistema.
3. **El Auditor ('count')**: Un mecanismo para contabilizar la frecuencia de cada llamada al sistema desde el arranque, proporcionando una vista agregada del uso de recursos del kernel.

# Herramienta 1: El Observador ('trace')

## Arquitectura de Cambios

- **Objetivo:** Implementar el trazado de syscalls activable desde el espacio de usuario.
- **Componentes Modificados:**
  - **syscall.c:** Se añadió la variable global `syscall_tracing` y el arreglo `syscall_names[]`. Se modificó la función `syscall()` para imprimir la traza si está activa.
  - **sysproc.c:** Se implementó la nueva syscall `sys_trace()` para modificar la variable `syscall_tracing`.
  - **syscall.h, user.h, usys.S:** Se realizaron las declaraciones y enlaces necesarios para la nueva syscall (`SYS_trace`).
  - **trace.c:** Se creó el nuevo comando de usuario que invoca `trace(1)` para 'on' y `trace(0)` para 'off'.

## Demostración en Acción

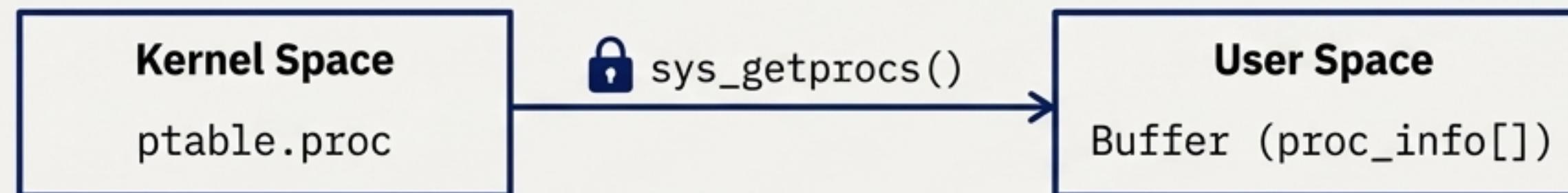
Activación, ejecución de un comando ('ls') y desactivación del trazado.

```
init: starting sh
$ trace on
syscall: write(1, 2f8f, 1)
Tsyscall: write(1, 2f8f, 1)
rsyscall: write(1, 2f8f, 1)
ls
syscall: read(0, 3f9f, 1)
syscall: read(0, 3f9f, 1)
syscall: fork
syscall: wait
syscall: sbrk(32768)
syscall: exec("ls", bfac)
syscall: open(".", 0)
syscall: fstat(3, 2d7c)
syscall: read(3, 2d6c, 16)
syscall: open("./", 0)
trace off
syscall: read(0, 3f9f, 1)
syscall: fork
syscall: wait
syscall: sbrk(32768)
syscall: exec("trace", bfac)
syscall: trace(0)
Tracing desactivado
```

# Herramienta 2: El Inspector (Arquitectura)

## Creando la Syscall `getprocs` para Exponer Información de Procesos

Para permitir que comandos de usuario consulten el estado de los procesos, implementamos una nueva syscall (`sys_getprocs`) que transfiere datos de manera segura desde el espacio del kernel al espacio de usuario.



- **Estructura de Datos (`user.h`)**
  - Se definió `struct proc_info` para estandarizar la información transferida: `pid`, `name[16]`, `state`.
- **Implementación en Kernel (`sysproc.c`)**
  - La función `sys_getprocs()` recibe un puntero a un buffer y un tamaño máximo.
  - **Seguridad:** Valida el puntero del buffer del usuario con `argptr()` para prevenir accesos a memoria inválida.
  - **Funcionalidad:** Itera sobre la tabla global de procesos (`ptable.proc`), copia la información (PID, nombre, estado) de los procesos activos (`state != UNUSED`) al buffer del usuario.
  - **Retorno:** Devuelve el número total de procesos copiados.

# Herramienta 2: El Inspector (Demostración)

Comandos de Usuario para Consultar el Estado del Sistema

## ‘schedinfo’ - Listado de Procesos Activos

Este comando utiliza la syscall `getprocs()` para obtener la lista de todos los procesos activos y la presenta en una tabla formateada, mostrando PID, nombre y estado.

```
$ schedinfo
PID NAME      STATE
--- ----
 1 init      SLEEPING
 2 sh        SLEEPING
 4 schedinfo  RUNNING
Total de procesos: 3
$ sh &
$ schedinfo
PID NAME      STATE
--- ----
 1 init      SLEEPING
 2 sh        SLEEPING
 4 sh        SLEEPING
 6 sh        SLEEPING
 7 schedinfo  RUNNING
Total de procesos: 5
```

## ‘uptime’ - Tiempo de Actividad del Sistema

Un comando simple que invoca la syscall existente `uptime()` y convierte los ‘ticks’ del sistema a segundos, mostrando el tiempo transcurrido desde el arranque.

```
$ uptime grep
Uptime: 2982 ticks (29 seconds)
$
```

# Herramienta 3: El Auditor (`count`)

## Arquitectura de Cambios

**Objetivo:** Implementar un contador global para cada syscall y exponerlo al usuario.

### Componentes Modificados:

- **syscall.c:** Se añadió el arreglo global `syscall_counts[NSYSCALLS]`. La función `syscall()` ahora incrementa `syscall_counts[num]` después de cada ejecución exitosa.
- **sysproc.c:** Se implementó la nueva syscall `sys_getcounts()`, que utiliza `copyout()` para copiar de forma segura el arreglo de contadores al espacio de usuario.
- **syscall.h, user.h, usys.S:** Se añadieron las definiciones y enlaces para `SYS_getcounts`.
- **count.c:** Se creó el comando de usuario que invoca a `getcounts()` y muestra los resultados. Permite consultar todos los contadores o uno específico por nombre o número.

## Demostración en Acción

Consultando los contadores de forma general y específica por ID de syscall.

```
init: starting sh
xv6> count
Contadores de syscalls:
fork: 2
read: 6
exec: 3
dup: 2
sbrk: 1
open: 3
write: 23
mknod: 1
close: 1
xv6> _
```

```
xv6> count 1
fork: 7
xv6> count 16
write: 908
xv6> █
```

# El Kit de Herramientas en Acción: Analizando la Actividad del Sistema

Al ejecutar una serie de comandos, podemos utilizar nuestro kit de herramientas para analizar el impacto en el kernel. Primero, observamos los contadores iniciales. Luego, ejecutamos varios comandos y volvemos a consultar los contadores para ver el cambio.

## 1. Estado Inicial

```
init: starting sh
xv6> count
Contadores de syscalls:
fork: 2
read: 6
exec: 3
dup: 2
sbrk: 1
open: 3
write: 23
mknod: 1
close: 1
xv6> _
```

## 2. Actividad del Sistema

```
$ ls
$ cat README
$ sh &
```



Se observa un aumento de 6 a 60 en `read` y de 23 a 756 en `write`, reflejando la intensa actividad de I/O.

## 3. Análisis Posterior

```
xv6> countt
exec: fail
exec countt failed
xv6> count
Contadores de syscalls:
fork: 6
wait: 4
read: 60
exec: 7
fstat: 24
dup: 2
sbrk: 5
open: 27
write: 756
mknod: 1
close: 25
getcounts: 1
xv6> _
```

# Conclusiones Técnicas: Capacidades Obtenidas

El proyecto culminó exitosamente con la integración de un robusto kit de herramientas de diagnóstico en el kernel de XV6. Las modificaciones demuestran una comprensión profunda de la interacción entre el espacio de usuario y el kernel.



## Capacidad de Rastreo (`trace`)

Se implementó una herramienta funcional para la depuración y comprensión del comportamiento del sistema a nivel de syscalls, permitiendo la observación en tiempo real.



## Capacidad de Consulta (`schedinfo`, `uptime`)

Se desarrollaron comandos de usuario efectivos que, a través de nuevas syscalls seguras, exponen información crítica del kernel como el estado y la lista de procesos.



## Capacidad de Auditoría (`count`)

Se añadió un contador global que permite **auditar la frecuencia** de uso de las **funcionalidades** del kernel, ofreciendo una métrica clave para el análisis de rendimiento y comportamiento del sistema.

# Explore el Código Fuente

Todo el código desarrollado para este proyecto, incluyendo los nuevos comandos de usuario y las modificaciones al kernel, está disponible públicamente en nuestro repositorio de GitHub.

<https://github.com/LARMD-2/Proyecto-U2-S0.git>



---

## \*\*Referencias Clave\*\*:

- Código fuente de XV6 – Repositorio oficial de MIT
- Libro de XV6 – “Xv6, a simple Unix-like teaching operating system” (MIT 6.828)