UNIVERSIDAD NACIONAL JORGE BASADRE GROHMANN
FACULTAD DE INGENIERÍA
ESCUELA PROFESIONAL DE INGENIERÍA EN INFORMÁTICA Y SISTEMAS

XV6

Curso: Sistemas operativos

Docente: Hugo Manuel Barraza Vizcarra

Integrantes

Jhon Alfredo Mamani Tacora	2023-119036
Jhon Sebastian Mamani Ancco	2022-119022

Tacna - Perú
2025

Introducción

El proyecto se centra en el estudio de los Sistemas Operativos (SO), específicamente en el análisis y extensión del sistema XV6, una implementación moderna y simplificada de Unix de Sexta Edición (v6) desarrollada por el MIT con fines académicos.

Objetivos

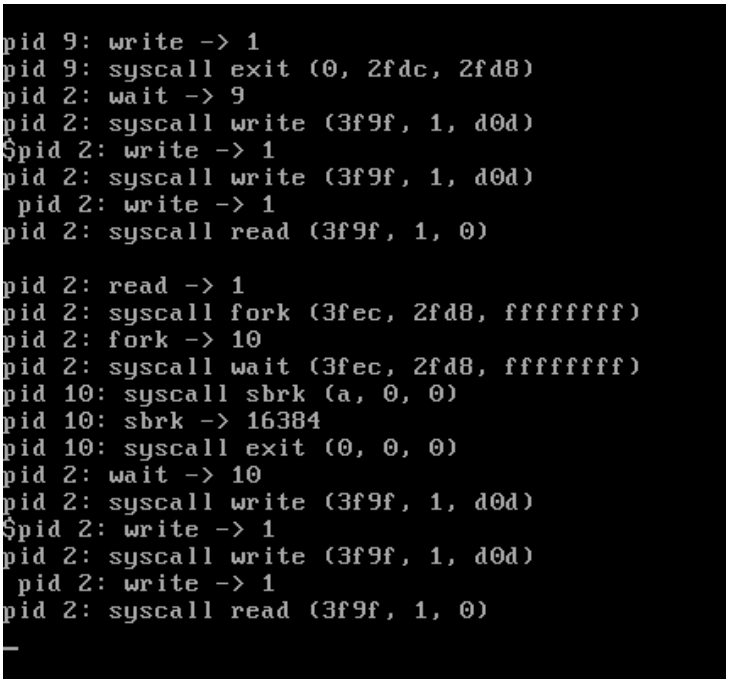
- Objetivo General: Extender las capacidades del núcleo de XV6 para mejorar la observabilidad del sistema y la gestión de recursos.

- **Objetivos Específicos:** Implementar un mecanismo de auditoría de *syscalls*, desarrollar herramientas de monitoreo de procesos en tiempo real y cuantificar el uso de la interfaz del sistema.

Descripción de las modificaciones realizadas

Instrumentación de llamadas al Sistema (Entregable 1)

1. Sycall trace echo hello



2. notrace: deshabilita el tracing

```
pid 2: syscall read (3f9f, 1, 0)
pid 2: read -> 1
pid 2: syscall read (3f9f, 1, 0)
pid 2: read -> 1
pid 2: syscall read (3f9f, 1, 0)
pid 2: read -> 1
pid 2: syscall read (3f9f, 1, 0)
pid 2: read -> 1
pid 2: syscall read (3f9f, 1, 0)
pid 2: read -> 1
pid 2: syscall read (3f9f, 1, 0)
pid 2: read -> 1
pid 2: syscall read (3f9f, 1, 0)
pid 2: read -> 1
pid 2: syscall fork (3fec, 2fd8, ffffffff)
pid 2: fork -> 11
pid 2: syscall wait (3fec, 2fd8, ffffffff)
pid 11: syscall sbrk (6e, 0, 0)
pid 11: sbrk -> 16384
pid 11: syscall exec (bfac, 19a0, 19a0)
pid 11: exec -> 0
pid 11: syscall trace (0, 0, 0)
System call tracing disabled
$
```

3. Tracing global con trace ls: todas las sycalls de ls se muestran

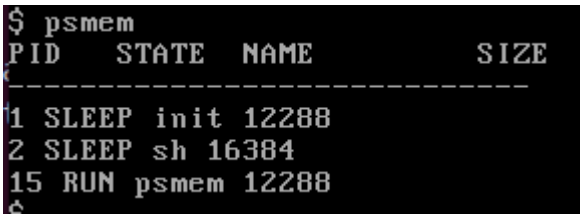
```
pid 12: syscall read (2d6c, 10, 0)
pid 12: read -> 16
pid 12: syscall read (2d6c, 10, 0)
pid 12: read -> 16
pid 12: syscall read (2d6c, 10, 0)
pid 12: read -> 16
pid 12: syscall read (2d6c, 10, 0)
pid 12: read -> 16
pid 12: syscall read (2d6c, 10, 0)
pid 12: read -> 16
pid 12: syscall read (2d6c, 10, 0)
pid 12: read -> 16
pid 12: syscall read (2d6c, 10, 0)
pid 12: read -> 16
pid 12: syscall read (2d6c, 10, 0)
pid 12: read -> 0
pid 12: syscall close (2d6c, 10, 0)
pid 12: close -> 0
pid 12: syscall exit (0, 0, 0)
pid 2: wait -> 12
pid 2: syscall write (3f9f, 1, d0d)
$pid 2: write -> 1
pid 2: syscall write (3f9f, 1, d0d)
 pid 2: write -> 1
```

Nuevos Comandos (Entregable 2):

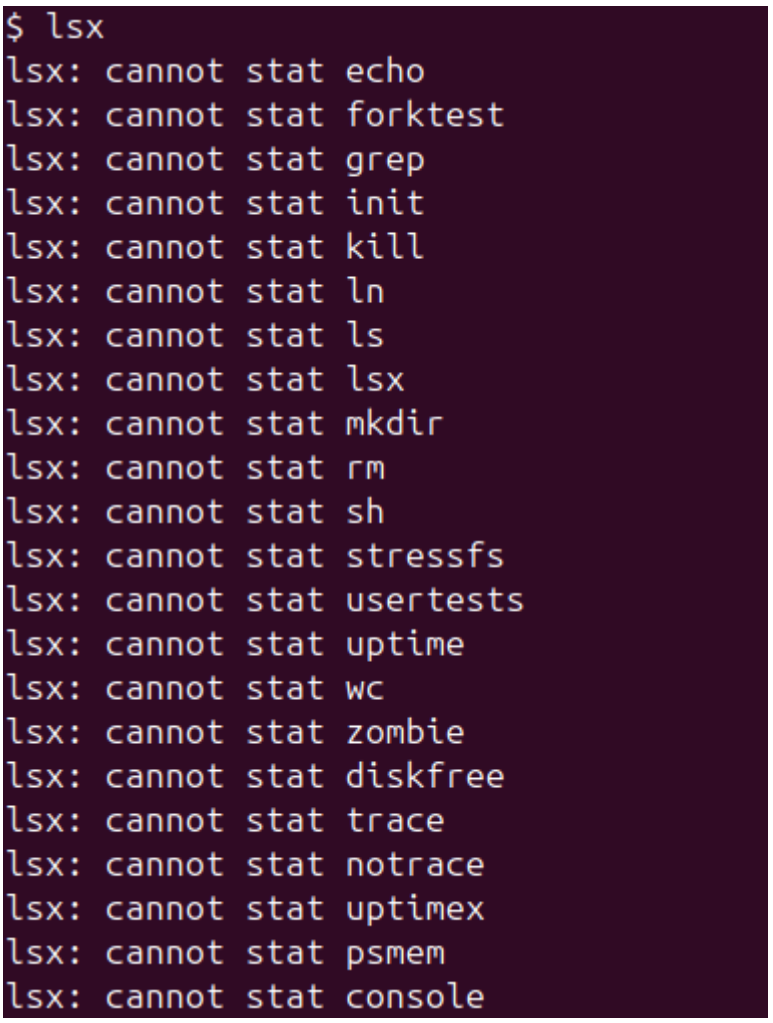
1. Uptimex

```
$ uptimex
1 SLEEP init 12288
2 SLEEP sh 16384
14 RUN uptimex 12288
up 32015 ticks, 3 processes
$
```

2. psmem



3. lsx



Contador de Syscalls (Entregable 3):

Es cuantificar el uso de la interfaz del núcleo. Para ello, se diseñó una estructura de datos persistente en el **espacio del kernel** que registra la frecuencia de cada syscall.

Modificación realizadas

1. **syscall.c (modificar)**

- Se implementó un arreglo de enteros indexado por el número de la llamada al sistema. El tamaño está definido por la constante NSYSCALLS.

```
int syscall_counts[NSYSCALLS];
```

- Se modificó la función central syscall(void) para que, antes de ejecutar cualquier función del kernel, incremente la posición correspondiente en el arreglo.

```
void syscall(void) {
        struct proc *p = myproc();
        int num = p->tf->eax;
        if(num > 0 && num < NSYSCALLS && syscalls[num]) {
                syscall_cou   nts[num]++;  < —--       AGREGADO
                p->tf->eax = syscalls[num]();
       } else {
                ….
       }
}
```

2. sysproc.c (modificar)

- Se creó la función `sys_getcount` para exponer los datos del kernel al espacio de usuario de forma segura.

```
int sys_getcount(void) {
      int id;
      if(argint(0, &id) < 0)
            return -1;
      if(id > 0 && id < NSYSCALLS) {
           return syscall_counts[id];
       }
       return -1;
       }
```

3. **sysstat.c (crear)**
   - Este programa es la herramienta que el usuario utiliza en Shell para visualizar los datos. Se diseñó para manejar dos modos de operación mediante la evaluación de `argc`

4. MAKEFILE (modificar)

- Se agregó la regla _sysstat a la variable UPROGS del Makefile con el objetivo de integrar el programa de usuario en la imagen del sistema de archivos (fs.img).

```
UPROGS=\
      _cat\
```

```
_echo\
_forktest\
…
_sysstat\          < - - -
```

Fragmentos relevantes de código comentado.

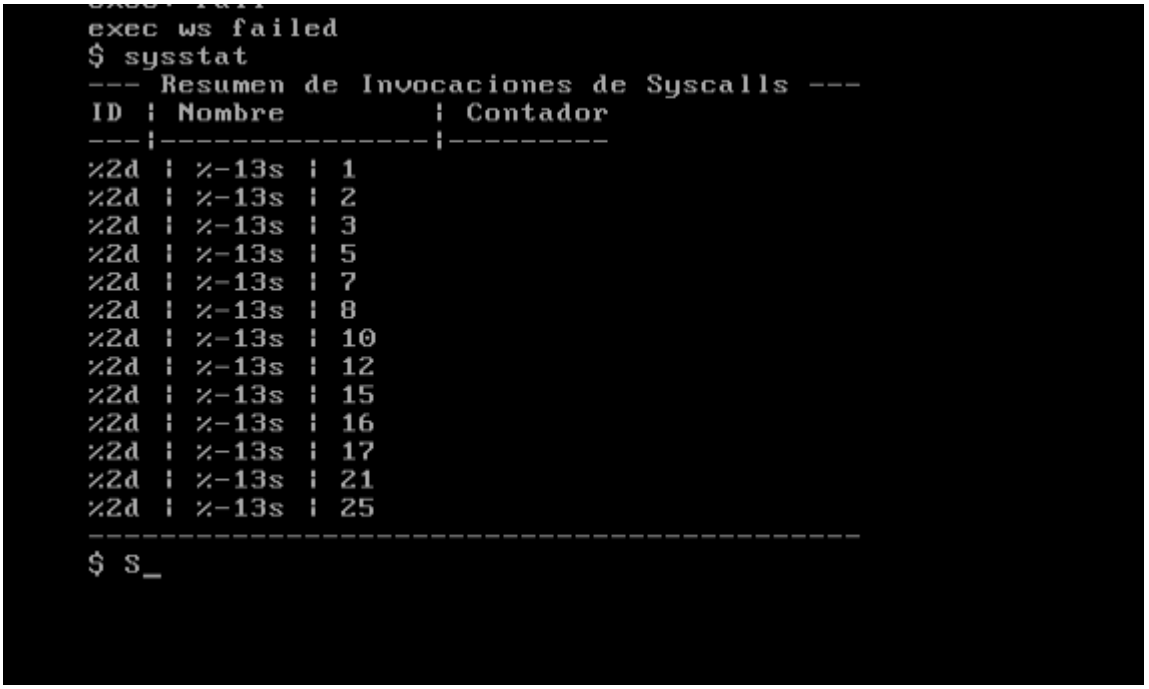
sysstat.c (Entregable 3)

```
#include "types.h"
#include "stat.h"
#include "user.h"
#include "syscall.h"

// Mapeo de IDs numéricos a nombres legibles para la impresión en pantalla
static char *syscall_names[] = {
  [SYS_fork]    "fork",
  [SYS_exit]    "exit",
  [SYS_wait]    "wait",
  [SYS_pipe]    "pipe",
  [SYS_read]    "read",
  [SYS_kill]    "kill",
  [SYS_exec]    "exec",
  [SYS_fstat]   "fstat",
  [SYS_chdir]   "chdir",
  [SYS_dup]     "dup",
  [SYS_getpid]  "getpid",
  [SYS_sbrk]    "sbrk",
  [SYS_sleep]   "sleep",
  [SYS_uptime]  "uptime",
  [SYS_open]    "open",
  [SYS_write]   "write",
  [SYS_mknod]   "mknod",
  [SYS_unlink]  "unlink",
  [SYS_link]    "link",
  [SYS_mkdir]   "mkdir",
  [SYS_close]   "close",
  [SYS_trace]   "trace",
  [SYS_psinfo]  "psinfo",
  [SYS_fsinfo]  "fsinfo",
  [SYS_getcount] "getcount"
};

#define MAX_SYSCALL_ID 25
```

```
int
main(int argc, char *argv[])
{
  // CASO 1: Consulta individual por ID (Ejemplo: sysstat 5)
  if (argc == 2) {
    int id = atoi(argv[1]); // Convierte el argumento a entero
    int count;

    if (id <= 0 || id > MAX_SYSCALL_ID) {
      printf(2, "ID no válido. Rango: 1-%d\n", MAX_SYSCALL_ID);
      exit();
    }

    // Invocación a la nueva syscall implementada en el kernel
    count = getcount(id);

    if (count < 0) {
      printf(2, "Error al consultar ID %d.\n", id);
    } else {
      char *name = (id <= MAX_SYSCALL_ID && syscall_names[id]) ?
syscall_names[id] : "unknown";
      printf(1, "Syscall %d (%s) invocada %d veces.\n", id, name, count);
    }
  }

  // CASO 2: Resumen global (Ejemplo: sysstat sin argumentos)
  else if (argc == 1) {
    printf(1, "--- Resumen de Invocaciones de Syscalls ---\n");
    printf(1, "ID | Nombre        | Contador\n");
    printf(1, "---|---------------|---------\n");

    // Itera sobre todas las syscalls para obtener sus contadores
    for (int id = 1; id <= MAX_SYSCALL_ID; id++) {
      int count = getcount(id);

      // Muestra solo las syscalls que han sido usadas al menos una vez
      if (count > 0) {
        char *name = (id <= MAX_SYSCALL_ID && syscall_names[id]) ?
syscall_names[id] : "unknown";
        printf(1, "%2d | %-13s | %d\n", id, name, count);
      }
    }
    printf(1, "-----------------------------------------\n");
  }

  // Manejo de error en sintaxis de comando
  else {
    printf(2, "Uso: sysstat [ID opcional]\n");
```

```
  }

  exit();
}
```

Resultados de pruebas

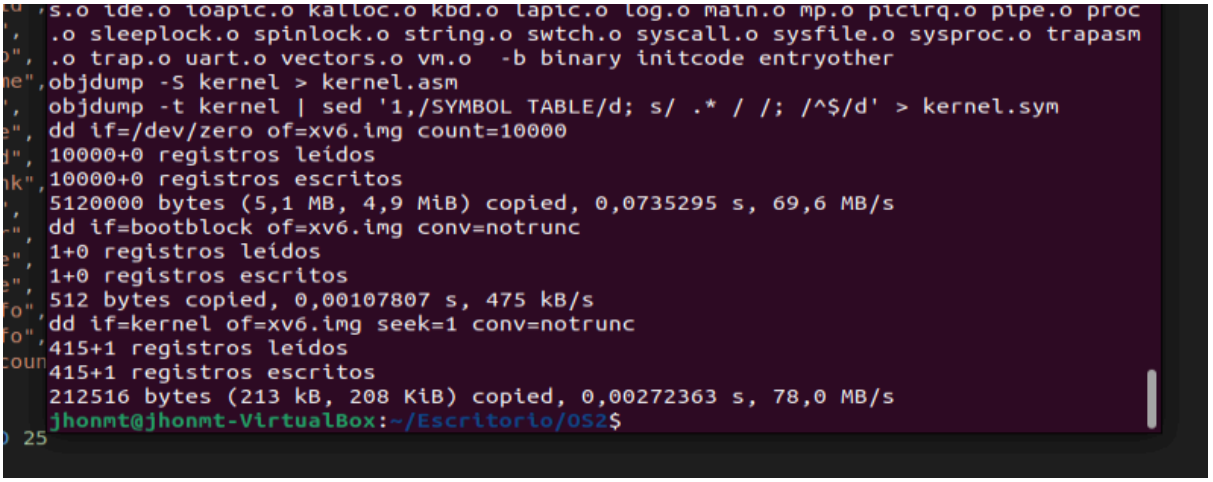


Conclusión

La implementación del contador de llamadas al sistema permitió validar la interacción entre la gestión de procesos y la administración de E/S. Al monitorear syscalls como read, write y fork, se observó experimentalmente cómo el sistema operativo coordina la planificación de procesos y el acceso a recursos de hardware.

El proyecto demostró que el uso de memoria virtual es fundamental para la seguridad del kernel, ya que la syscall getcount debió emplear mecanismos específicos para leer datos desde el espacio de direccionamiento del usuario sin comprometer la integridad de la memoria del sistema. Finalmente, esta práctica refuerza el concepto de que el kernel actúa como un gestor de recursos centralizado, controlando desde la planificación de disco hasta la comunicación entre procesos, sentando las bases para entender sistemas más complejos como los sistemas distribuidos.

ANEXO


s.o ide.o ioapic.o kalloc.o kbd.o lapic.o log.o main.o mp.o picirq.o pipe.o proc
.o sleeplock.o spinlock.o string.o swtch.o syscall.o sysfile.o sysproc.o trapasm
.o trap.o uart.o vectors.o vm.o -b binary initcode entryother
objdump -S kernel > kernel.asm
objdump -t kernel | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > kernel.sym
dd if=/dev/zero of=xv6.img count=10000
10000+0 registros leídos
10000+0 registros escritos
5120000 bytes (5,1 MB, 4,9 MiB) copied, 0,0735295 s, 69,6 MB/s
dd if=bootblock of=xv6.img conv=notrunc
1+0 registros leídos
1+0 registros escritos
512 bytes copied, 0,00107807 s, 475 kB/s
dd if=kernel of=xv6.img seek=1 conv=notrunc
415+1 registros leídos
415+1 registros escritos
212516 bytes (213 kB, 208 KiB) copied, 0,00272363 s, 78,0 MB/s
jhonmt@jhonmt-VirtualBox:~/Escritorio/OS2$