El Sistema Operativo XV6 - Capítulo 2 - Parte 2

Carlos Santiago Sandoval Casallas

csandovalc@unal.edu.co

Departamento de Ingeniería de Sistemas e Industrial
Universidad Nacional de Colombia

7 de mayo de 2025

UNIVERSIDAD NACIONAL DE COLOMBIA

Agenda

UNIVERSIDAD NACIONAL DE COLOMBIA

Organización del kernel

El código fuente del kernel se encuentra en el subdirectorio *“kernel”*, este se divide en archivos, buscando una noción de modularidad [1].

En la figura 2 se enumeran los archivos y su descripción. Las interfaces entre módulos se definen en ***kernel/defs.h***

Figura: Estructura del kernel XV6

System call	Description
bio.c	Disk block cache for the file system.
console.c	Connect to the user keyboard and screen.
entry.S	Very first boot instructions.
exec.c	exec() system call.
file.c	File descriptor support.
fs.c	File system.
kalloc.c	Physical page allocator.
kernelvec.S	Handle traps from kernel, and timer interrupts.
log.c	File system logging and crash recovery.
main.c	Control initialization of other modules during boot.
pipe.c	Pipes.
plic.c	RISC-V interrupt controller.
printf.c	Formatted output to the console.
proc.c	Processes and scheduling.
sleeplock.c	Locks that yield the CPU.
spinlock.c	Locks that don't yield the CPU.
start.c	Early machine-mode boot code.
string.c	C string and byte-array library.
swtch.S	Thread switching.
syscall.c	Dispatch system calls to handling function.
sysfile.c	File-related system calls.
sysproc.c	Process-related system calls.
trampoline.S	Assembly code to switch between user and kernel.
trap.c	C code to handle and return from traps and interrupts.
uart.c	Serial-port console device driver.
virtio_disk.c	Disk device driver.
vm.c	Manage page tables and address spaces.

UNIVERSIDAD NACIONAL DE COLOMBIA

Fuente: xv6: a simple, Unix-like teaching operating system [1]

Procesos

La unidad de aislamiento en XV6 es un proceso. Esta abstracción permite tener un mayor control sobre operaciones sobre la memoria, la CPU, los descriptores de archivos, etc. Esto evita que un proceso pueda manipular, espiar o dañar otro proceso o incluso el kernel mismo.

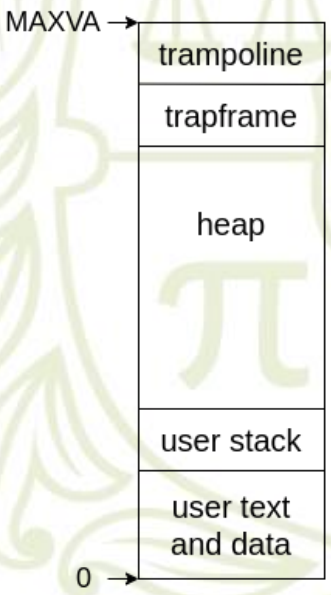
El kernel debe tener una implementación segura del proceso, dado que si se presentan errores, estos pueden ser empleados para engañar el kernel o al hardware, siendo una falla de seguridad. Los mecanismos empleados por el kernel para implementar procesos se basan en: indicadores de modo usuario/supervisor, espacios de direcciones y división de tiempo de subprocesos. [1]

Aislamiento

Para facilitar el proceso de aislamiento se proporciona la ilusión a un programa de que tiene la totalidad de la máquina para su ejecución. Para lograr la ilusión, al programa se le proporciona un sistema de memoria privada (espacio de direcciones), que no puede ser leída o escrita por algún otro proceso. Adicionalmente, se brinda lo que parece ser su propia CPU para ejecutar las instrucciones del programa.

XV6 implementa un sistema de direcciones basado en tablas de páginas, estas son implementadas por hardware, para administrar un espacio de direcciones virtuales. La tabla de páginas que implementa RISC-V mapea una dirección virtual **(la dirección que manipulan las instrucciones de RISC-V)** a una dirección física **(una dirección que el chip de la CPU envía a la memoria principal)**.[1]

Figura: Diseño del espacio de direcciones virtuales de un proceso



Fuente: xv6: a simple, Unix-like teaching operating system [1]

Tabla de páginas

XV6 brinda una tabla de páginas **para cada proceso**, esta definirá el espacio de direcciones de ese proceso, como se aprecia en la siguiente figura 2, este espacio de direcciones **incluye la memoria de usuario del proceso**, la cual inicia en la dirección virtual cero. Su estructura en orden creciente de las direcciones es:

- Las instrucciones que ejecutara el proceso
- Las variables globales
- El stack o pila
- El heap o montículo

Límites físicos

El heap puede ser empleado para expandir el espacio de direcciones según sea necesario, pero este crecimiento está limitado, los punteros en RISC-V tienen un tamaño de 64 bits, pero el hardware, solo usa los **39 bits menos significativos** cuando mapea direcciones virtuales en las tablas de páginas, donde XV6 **solo usa 38 de estos 39 bits**.

Por lo que **la dirección máxima es** $2^{38} - 1$, esta se define en la macro ***MAXVA***, esta se encuentra en ***(kernel/riscv.h:363)***. [2]

Trampolín y Trapframe

Hacia las direcciones más altas del espacio de direcciones de memoria, encontramos el espacio de XV6, aquí, se reserva una página para el ***trampoline*** y una página que mapea el ***trapframe*** del proceso. Estas páginas son empleadas por XV6 para hacer la transición entre el kernel y el proceso.

Estas páginas se abordarán con más profundidad en próximos capítulos, pero en resumen, la página del trampolín contiene el código para entrar y salir del kernel, mientras el trapframe es usado para almacenar el estado del proceso. [3]

Estado del proceso

El kernel de XV6 almacena en una estructura el estado de cada proceso, esta estructura se define en ***(kernel/proc.h:85)*** [2]. Las partes más importantes del núcleo de un proceso son:

- El estado de ejecución ***kernel/proc.h:89***
- La pila del kernel ***kernel/proc.h:99***
- La tabla de páginas ***kernel/proc.h:101***

Hilos

Cada proceso tiene un ***hilo o subproceso*** de ejecución, este es el encargado de ejecutar una serie de instrucciones del proceso, estos hilos pueden ser suspendidos para posteriormente reanudar su ejecución.

Para ejecutar los cambios entre procesos, el kernel **suspende** el hilo que está en ejecución y **reanuda** el hilo relacionado con otro proceso.

El sistema presenta un ***cambio de contexto,*** a raíz de cambio de estado de un subproceso, para que el sistema lleve a cabo esta operación es necesario que: **variables locales, direcciones de retorno de llamadas a funciones**, sean almacenadas en la pila del hilo.

UNIVERSIDAD NACIONAL DE COLOMBIA

Cada proceso tiene dos ***pilas o stacks***: una pila de usuario y una pila del kernel *kstack*.

- Cuando el proceso ejecuta instrucciones de usuario, solo emplea la pila de usuario, y su pila de kernel está vacía.
- Cuando el proceso ingresa al kernel (ya sea por una llamada al sistema o una interrupción), el código de manejo, del kernel se ejecutará en la pila del kernel del proceso.
- Mientras un proceso este en el kernel, su pila de usuario **aun contiene** datos almacenados, aunque estos no se usen activamente.

Pila del kernel

La pila del kernel está separada (y protegida del código de usuario), para que el kernel pueda ejecutarse aun cuando un proceso haya estropeado su pila de usuario. [1]

UNIVERSIDAD NACIONAL DE COLOMBIA

Un proceso puede realizar una llamada al sistema empleando la instrucción ***ecall*** de RISC-V [1]. Esta instrucción **eleva** el nivel de privilegio del hardware y **cambia** el ***contador del programa*** al punto de entrada al kernel definido [1].

El código de manejo, cambia a la pila del kernel y ejecuta las instrucciones para resolver la petición. Cuando finaliza la llamada al sistema, el kernel vuelve a la pila de usuario mediante la instrucción ***sret***, esta instrucción, reducirá el nivel de privilegio del hardware y reanuda la ejecución de las instrucciones del usuario, justo después de la instrucción de la llamada al sistema [1]

Interrupciones

El hilo de un proceso puede ser suspendido por el kernel mientras espera a que se solucione la petición de E/S, y reanudar donde se quedó.

UNIVERSIDAD NACIONAL DE COLOMBIA

Un vistazo al código

En el código fuente de XV6, “p->state” indica si el proceso está asignado, listo para ejecutarse, en ejecución, esperando E/S o finalizando. [1]

Mientras que “p->pagetable”, contiene la tabla de páginas de un proceso, en el formato que maneja el hardware de RISC-V. XV6 se encarga que el hardware de paginación emplee la tabla de páginas de un proceso cuando se ejecuta ese proceso en el espacio de usuario. **Esta tabla de páginas actúa como registro de las direcciones físicas de las páginas asignadas a un proceso.** [1]

UNIVERSIDAD NACIONAL DE COLOMBIA

Resumen

Un proceso agrupa unas ideas fundamentales de diseño: un espacio de direcciones para dar al proceso la ilusión de su propia memoria, y un hilo, para dar la ilusión de su propia CPU. [1]

En sistemas operativos más robustos, un proceso puede tener múltiples hilos para aprovechar varias CPU.

Arranque

Cuando la computadora se enciende, se ejecuta el cargador de arranque almacenado en la memoria ***ROM*** (Read-Only Memory), el gestor de arranque **carga el kernel de XV6** en la memoria.

Cada CPU arranca en modo máquina, el emulador QEMU se encarga de que cada hilo de ejecución del que dispone el hardware vaya al punto de entrada definido en código ensamblador de RISC-V, este punto de entrada se define a través de la etiqueta "_entry" ***kernel/entry.S:7***.

Memoria

En esta etapa, el hardware de paginación está deshabilitado, por lo que las direcciones virtuales se asignan directamente a las direcciones físicas.

UNIVERSIDAD NACIONAL DE COLOMBIA

Habilitando la ejecución de código

Cuando el kernel es cargado a la memoria, será cargado en la dirección física **0x80000000**, esta dirección es empleada dado que las direcciones desde **0x0:0x80000000** son empleados por QEMU para los dispositivos de E/S. [1]

Las instrucciones en _entry configuran el stack para que XV6 pueda ejecutar código en C. XV6 declara el espacio para la pila inicial, **stack0**, en el archivo start.c ***(kernel/start.c:11)*** [1].

El código en _entry carga la dirección del **stack+4096** bytes al puntero de pila, esta operación se realiza por cada CPU. En el puntero de la pila se carga la dirección más alta, dado que en RISC-V **la pila crece hacia direcciones más bajas**.

Una vez está asignado la pila del kernel, _entry ejecuta la función de inicio ***(kernel/start.c:21)***. [1]

UNIVERSIDAD NACIONAL DE COLOMBIA

Configurando el sistema

Después de ser invocada por _entry, la función de inicio se encarga de realizar las configuraciones para iniciar el sistema, algunas de estas operaciones solo se permiten en el modo máquina, después de finalizar las operaciones que lo requieren, XV6 reducirá su privilegio sobre el hardware, al modo supervisor.

RISC-V proporciona la instrucción “mret”, esta instrucción se suele emplear para regresar a un nivel de privilegios anterior, por ejemplo: si ocurre una excepción en el modo supervisor, la CPU cambiará al modo máquina para ejecutar las directrices de manejo de la excepción, una vez finalice este proceso, se empleará mret para retornar al modo supervisor.

UNIVERSIDAD NACIONAL DE COLOMBIA

Configuraciones en modo máquina

Respecto al manejo de mret, se presenta un problema, la función start no regresa de una llamada de este tipo, por lo que se configura el sistema como si lo hiciera, estas configuraciones son:

- Se define el “modo anterior”, a modo supervisor, esto mediante el cambio en el registro **mstatus**.
- Establece la **dirección de retorno en el método main**, escribiendo la dirección de dicho método en el registro **mepc**.
- Deshabilita la paginación en modo supervisor, esto se realiza escribiendo un 0 en el registro **satp** de la tabla de páginas.
- Delega todas las interrupciones y excepciones al modo supervisor.

UNIVERSIDAD NACIONAL DE COLOMBIA

Continuando con las operaciones de configuración:

- Concede al modo supervisor el acceso **toda** la memoria física, mediante la configuración de la protección de memoria física.
- Programa el chip del reloj para recibir interrupciones del temporizador.
- Almacena el identificador de cada CPU en el registro de hilo **(thread pointer o tp)**

Una vez se finaliza esta configuración, se ejecuta la instrucción de retorno, esto para “regresar” al modo supervisor y cambiando el puntero del programa **(stack pointer o sp)** a la función main, esta función se define en ***(kernel/main.c:11)***. [1] [2]

UNIVERSIDAD NACIONAL DE COLOMBIA

Iniciando el sistema

La función main, primero reconoce la CPU con ID 0, y este se encargará de inicializar y configurar diversos sistemas, algunas de las tareas que realiza son:

- Inicia la consola.
- Habilita la salida estándar (printf).
- Inicia el asignador de páginas físicas.
- Crea la tabla de páginas del kernel.
- Habilita la paginación.
- Inicia la tabla de procesos

Entre otras muchas configuraciones, pero el primer proceso de usuario que ejecuta es userinit, el cual se define en ***kernel/proc.c:233*** [2].

UNIVERSIDAD NACIONAL DE COLOMBIA

Primer proceso de usuario

El primer proceso de usuario ejecuta un código ensamblador, este está definido en ***user/initcode.S***, este código prepara el uso de la syscall **exec**.

Primero, carga la dirección del método **init** en el registro **a0**, luego la dirección del arreglo con los argumentos de la llamada **(**argv) al registro **a1**, despues carga el número que identifica la llamada al sistema de SYS_exec ***(kernel/syscall.h:8)*** en el registro **a7**, y finalmente llama a ecall para ingresar de nuevo al kernel. [1]

Preparando el espacio de usuario

El kernel emplea el número de la llamada al sistema almacenada en el registro a7 para ejecutar la llamada indicada ***(kernel/syscall.c:132)***. Estos identificadores son asignados en ***kernel/syscall.c:107***, donde se relaciona a la constante SYS_exec con el método sys_exec, que invoca el kernel. [1]

Como se trató en el capítulo 1, exec reemplaza la memoria y los registros del proceso actual con una nueva imagen ejecutable **(en este caso /init)**.

Iniciando el espacio de usuario

Una vez el kernel finaliza estas operaciones, “regresa” al espacio del usuario en el proceso init ***(user/init.c:15)***. Este proceso inicia la consola, si es necesario crea un nuevo archivo de dispositivo que referencia la consola, luego lo abre como descriptores de archivo con los valores estándar (0, 1 y 2). Inicia un Shell en la consola.

El sistema está activo!!!

UNIVERSIDAD NACIONAL DE COLOMBIA

Seguridad en el sistema operativo

El enfoque respecto al diseño que se debe implementar es, asumir que **todo** código de nivel de usuario de un proceso buscara de **todas las formas posibles** arruinar el kernel u otros procesos.

- Referenciar puntos fuera de su espacio de direcciones permitido.
- Ejecutar instrucciones de la CPU aunque estas no estén permitidas al usuario.
- Intentar leer o escribir sobre registros de control de RISC-V.
- Intentar acceder directamente al hardware del dispositivo.
- Pasar argumentos a las llamadas al sistema que busquen entorpecer o bloquear el kernel.

UNIVERSIDAD NACIONAL DE COLOMBIA

El objetivo del kernel es restringir los procesos de cada usuario para que solo pueda leer/escribir/ejecutar su espacio de direcciones asignado, emplear los 32 registros de RISC-V de propósito general, emplear llamadas al sistema que modifiquen otros proceso o el kernel de manera segura. Es un requisito absoluto que el **kernel evite cualquier otra acción que el usuario desee ejecutar**. [1]

Comunidad y enfoque

Se asume que el código del kernel está escrito por programadores cuidadosos y sin malas intenciones, se espera que el código del kernel este libre de errores y no contenga nada malicioso. [1] **Algunas funciones como el spinlock puede generar múltiples problemas si se usa de manera incorrecta**

Esto debe estar presente al momento de leer el código, se debe suponer que el código está escrito correctamente, también que este sigue todas las reglas sobre el uso de funciones y estructuras de datos propias del kernel [1]. Igualmente, se supone que el hardware funciona como se define en la documentación, y que no se presentan errores.

UNIVERSIDAD NACIONAL DE COLOMBIA

Claramente, estos son escenarios idealistas, no es tan simple evitar que el código ejecutado en espacio de usuario ocasione que el sistema quede inutilizable o que esté entre en pánico, principalmente porque el código malicioso intenta consumir recursos protegidos por el kernel, como: espacio en disco, tiempo de CPU, entradas de la tabla de procesos, etc. [1]

Generalmente, es imposible escribir código sin errores o diseñar de hardware libre de errores, por lo que si desarrolladores maliciosos están al tanto de los errores del kernel o el hardware, los explotarán. [1]

Incluso en kernels maduros y de alto uso, como Linux, se descubren nuevas vulnerabilidades continuamente; por lo que, vale la pena diseñar sistemas de protección en caso de que el kernel contenga errores, algunas de las estrategias de protección son:

- ***Aserciones***
- Verificación de tipos
- Páginas de protección de la pila
- Entre otros

UNIVERSIDAD NACIONAL DE COLOMBIA

Mundo real

La distinción entre el código del usuario y del kernel suele ser distorsionada. Algunos procesos de nivel de usuario pueden brindar servicios esenciales y ser parte del sistema operativo de manera efectiva, y en algunos sistemas operativos, el código de usuario puede insertar nuevo código en el kernel (como con los módulos del kernel de Linux). [1]

UNIVERSIDAD NACIONAL DE COLOMBIA

Implementaciones modernas

La mayoría de los sistemas operativos modernos emplean el concepto de proceso, siendo similares a la implementación de XV6; sin embargo, los sistemas modernos brindan soporte de subprocesos dentro de un proceso, esto permite que un solo proceso emplee varias CPU.

XV6 no proporciona el soporte para múltiples subprocesos.

UNIVERSIDAD NACIONAL DE COLOMBIA

Referencias I

[1] Cox, Kaashoek, Morris. **xv6: a simple, Unix-like teaching operating system.** 2022. URL: https://github.com/mit-pdos/xv6-riscv-book.

[2] MIT PDOS. **xv6-riscv**. 25 de ago. de 2022. URL: https://github.com/mit-pdos/xv6-riscv.

[3] Alex Hoppus. **What is trap frame? And what is difference between trap frame and task_struct?** https://stackoverflow.com/questions/47851969/what-is-trap-frame-and-what-is-difference-between-trap-frame-and-task-struct. [Online; accessed 3-July-2023]. 2017.

UNIVERSIDAD NACIONAL DE COLOMBIA

Gracias por la atención

Contacto:

csandovalc@unal.edu.co

UNIVERSIDAD NACIONAL DE COLOMBIA