El Sistema Operativo XV6 - Capitulo 2 - Parte 1

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

Introducción

Es fundamental que un sistema operativo **admita múltiples actividades a la vez**, el SO debe compartir los recursos disponibles entre todos los procesos, garantizando que todos tengan la oportunidad de ejecutarse.

El SO también debe brindar **aislamiento entre los procesos**, de tal manera que si un proceso falla o no funciona correctamente, no afecte a los procesos que no dependen de este proceso con errores. El aislamiento completo de procesos es demasiado *fuerte*, ya que se debe permitir que los procesos interactúen intencionadamente. Una solución a este tipo de situaciones es la ***pipe*** o tubería.

Requisitos del Sistema Operativo

El SO debe brindar: multiplexación, aislamiento e interacción. [1]

Objetivos del capítulo

El capítulo 2 proporciona una descripción general sobre la organización que implementan muchos sistemas operativos basados en Unix, se centrara principalmente en los diseños convencionales en un **kernel monolítico**.

Este capítulo también proporcionará una descripción más general de un proceso en XV6 y la creación del primer proceso cuando arranca el SO.

Entorno

XV6 se ejecuta en un microprocesador multinúcleo RISC-V [1], y parte significativa de sus funcionalidades de bajo nivel es específica de RISC-V.

RISC-V es una CPU de 64 bits, y XV6 está escrito en “LP64”C, esto significa que los punteros largos **(LP)** en el lenguaje de programación C, son de 64 bits, pero *int* es de 32 bits.

Arquitectura

El libro [1], asumirá que el lector ha realizado programación a nivel de máquina en **cualquier** arquitectura, así que se presentaran las ideas propias y específicas de RISC-V a medida que sea necesario.

UNIVERSIDAD NACIONAL DE COLOMBIA

Referencias para RISC-V

En el libro [1] se brindan algunas referencias útiles para RISC-V como:

- The RISC-V Reader: An Open Architecture Atlas. [2]
- El ISA a nivel de usuario. [3]
- El ISA de arquitectura privilegiada. [4]

Hardware simulado

La CPU de una computadora completa está rodeada de hardware de soporte, una gran parte en forma de interfaces de E/S [1].

XV6 está escrito para el hardware de soporte simulado, este es brindado por **QEMU**, dentro del hardware de soporte simulado se incluye:

- RAM.
- ROM la cual contiene código de inicio.
- Conexión en serie al teclado y la pantalla del usuario.
- Un disco para almacenamiento.

UNIVERSIDAD NACIONAL DE COLOMBIA

¿Por qué tener un SO?

¿Realmente es necesario contar con un sistema operativo? Esto es un cuestionamiento válido, dado que se pueden implementar las llamadas al sistema como una librería que emplearían las aplicaciones. Esto permitiría incluso que cada aplicación tenga su propia librería adaptada a sus necesidades, además de permitir emplear de mejor manera los recursos disponibles. **(Incluso algunos sistemas operativos para sistemas embebidos o sistemas en tiempo real, se organizan de esta manera)**

El enfoque anterior tiene un problema, y es que su funcionamiento se basa en la confianza. Los procesos deberán cooperar entre sí **(respetar los recursos compartidos, ceder recursos para que sean aprovechados por otros procesos)** y confiar en que no se presenten errores.

UNIVERSIDAD NACIONAL DE COLOMBIA

El enfoque anterior tiene un problema, y es que su funcionamiento se basa en la confianza. Los procesos deberán cooperar entre sí **(respetar los recursos compartidos, ceder recursos para que sean aprovechados por otros procesos)** y confiar en que no se presenten errores.

Esto no es común, al contrario, lo más típico es que las aplicaciones no confíen entre sí y tengan errores, por lo que generalmente se requiere de un aislamiento de procesos más fuertes que el propuesto en el sistema cooperativo.

UNIVERSIDAD NACIONAL DE COLOMBIA

Aislamiento

Una forma de proporcionar un aislamiento fuerte, es prohibir a que las aplicaciones accedan directamente a recursos de hardware confidenciales [1], esto mediante la abstracción de recursos.

Ya hemos visto esta estrategia, por ejemplo, el brindar acceso a diferentes archivos únicamente mediante operaciones de: apertura, lectura, escritura y cierre, en lugar de acceder directamente al almacenamiento en disco. Por lo que la administración del disco es una responsabilidad cedida al SO, siendo incluso más conveniente para las aplicaciones.

De igual manera, en sistemas basados en Unix, se evidencia la distribución de la CPU entre los procesos, guardando y restaurando el estado del proceso según sea necesario, esto elimina la necesidad de que las aplicaciones tengan que tener una administración de recursos enfocada al tiempo compartido.

Esto permite que el SO comparta CPU incluso con procesos que están en bucles infinitos.

Conmutación por error

Parte importante del aislamiento fuerte es definir un límite estricto entre las aplicaciones y el sistema operativo, si una aplicación falla, no es conveniente que esto afecte al SO u otras aplicaciones. El SO debe poder limpiar la aplicación fallida y continuar ejecutando otras aplicaciones.

De igual manera, el SO debe garantizar que ninguna aplicación tenga la capacidad de leer o modificar las estructuras de datos e instrucciones del SO y tampoco acceder a memoria de otros procesos.

Aislamiento soportado por hardware

RISC-V presenta tres modos en los que la CPU puede ejecutar instrucciones:

- **Modo máquina:** Las instrucciones ejecutadas en este modo tienen **todos los privilegios**, principalmente se emplea en la configuración de la computadora.
- **Modo supervisor:** En este modo, la CPU puede ejecutar instrucciones privilegiadas.
  Habilitar/Deshabilitar interrupciones, leer y escribir el registro que contiene la dirección de una tabla de paginas, etc.
- **Modo usuario:** Este es el nivel con menos privilegios, donde las instrucciones son básicas.
  Sumar números.

UNIVERSIDAD NACIONAL DE COLOMBIA

La CPU inicia en modo máquina, aquí XV6 realiza algunas configuraciones para el arranque de la computadora **(por ej. configuración de la memoria virtual)** y luego cambia a modo supervisor.

El modo supervisor es también considerado **espacio del kernel**, aquí se permite la ejecución de instrucciones privilegiadas. En caso de que un programa por fuera del espacio del kernel intente ejecutar instrucciones privilegiadas, **la CPU no ejecutara la instrucción**, cambiara a modo supervisor, y desde este modo finalizara la aplicación, dado que intento hacer algo que no tiene permitido.

UNIVERSIDAD NACIONAL DE COLOMBIA

Espacio de kernel - Espacio de usuario

Si una aplicación quiere emplear una función del kernel, debe hacer **una transición al kernel**; dado que no puede invocar dicha función directamente. Para esto, la CPU proporciona una **instrucción especial** que cambia del modo usuario al modo supervisor, ingresando al espacio de kernel en un punto definido por el kernel.
En el caso de RISC-V, es la instrucción **ecall**.

Una vez la CPU se encuentra en modo supervisor, el kernel puede validar los argumentos de la llamada al sistema **(por ej. verificar si la dirección de memoria pasada a la syscall hace parte de la memoria de la aplicación)**, decidir si la llamada solicitada puede ser ejecutada por la aplicación **(por ej. verificar si la aplicación tiene el permiso para escribir el archivo especificado)**, y con base en estas validaciones llevar a cabo o negar la ejecución.

Seguridad respecto al punto de entrada al kernel

Es importante que el kernel controle el punto de entrada al modo supervisor; si no fuese así, una aplicación malintencionada podría decidir el punto de entrada, permitiéndole, por ejemplo, ingresar al kernel en un punto donde se omite la validación de argumentos.

UNIVERSIDAD NACIONAL DE COLOMBIA

Diseño

Como se menciona anteriormente, una decisión clave en la implementación del sistema operativo, es que este se ejecute en el modo supervisor de la CPU.

Pero existen otros factores a considerar, por ejemplo, la opción de que todo el sistema operativo resida en el kernel, esto permite que todas las llamadas al sistema se ejecuten en el modo supervisor. Esta organización se llama **kernel monolítico**.

Kernel monolítico

En el diseño de kernel monolítico, todo el sistema operativo se ejecuta con todos los privilegios de hardware, es conveniente porque el diseñador del SO no tiene que decidir que partes del SO no requieren de privilegios completos sobre el hardware, adicionalmente, facilita la cooperación entre distintas partes del SO.

Un sistema operativo puede tener un caché de búfer que puede ser compartido tanto por el sistema de archivos como por el sistema de memoria virtual.

Desventajas

Una desventaja de la organización monolítica es que las interfaces entre las diferentes partes del sistema operativo suelen ser complejas, por lo que aumenta la probabilidad de cometer un error.

En esta metodología de diseño, un error es fatal, dado que un error en el modo supervisor a menudo ocasionará una falla del kernel, causando una falla en la computadora y las aplicaciones en ejecución, por lo que es necesario reiniciar.

UNIVERSIDAD NACIONAL DE COLOMBIA

Microkernel

Con el fin de reducir estos errores en el kernel, los diseñadores pueden minimizar la cantidad de código del sistema operativo que se ejecuta en modo supervisor, ejecutando la mayor parte del sistema en modo usuario. **Esta organización se llama microkernel.**

Figura: Estructura de un microkernel y método de comunicación

Fuente: Elaboración propia. Basado en xv6: a simple, Unix-like teaching operating system [1]

En la figura 1 se ilustra como el **sistema de archivos** se ejecuta como un proceso a nivel de usuario, estos servicios del sistema operativo que se ejecutan como procesos se denominan **servidores**.

Para que las aplicaciones interactúen con los servidores del sistema operativo, el kernel brinda mecanismos de comunicación entre procesos para enviar mensajes de un proceso en espacio de usuario a otro.

Si una aplicación como el Shell desea leer o escribir un archivo, envía un mensaje al servidor de archivos y espera una respuesta [1].

En un microkernel, la interfaz de llamadas al sistema es simple, consta de algunas funciones de bajo nivel para iniciar las aplicaciones, realizar el envío de mensajes entre procesos, acceder a hardware de dispositivo, etc. Esto permite que el kernel sea simple, ya que la mayor parte reside en los servidores que se encuentran a nivel de usuario.

En el mundo real, ambos diseños **(kernel monolítico - microkernel)**, son populares, incluso pueden combinarse hasta cierto punto.

Por ejemplo, tomemos el caso de Linux, este cuenta con una peculiaridad, y es que algunas funciones del SO se ejecutan en servidores a nivel de usuario, por ejemplo **el sistema de ventanas**.

Por otro lado, existen ciertos microkernels que ejecutan algunos de los servicios de nivel de usuario en el espacio del kernel con el fin de obtener mejor rendimiento.

UNIVERSIDAD NACIONAL DE COLOMBIA

Actualmente, existe un gran debate sobre qué implementación es mejor; lamentablemente, no existe evidencia concluyente que demuestre la superioridad de un modelo respecto al otro. Adicionalmente, este abre una nueva discusión sobre la definición de que significa "mejor": mayor velocidad de rendimiento, tamaño de código más pequeño, fiabilidad del kernel, fiabilidad del sistema operativo completo, etc.

Algunos sistemas operativos modernos tienen kernels monolíticos porque fueron construidos de esta manera, y existen pocos incentivos para pasarlos a un diseño de un microkernel puro, por lo que desarrollar nuevas características se torna más importante que la reconstrucción de un sistema operativo completo.

XV6

XV6 tiene una implementación monolítica, como gran parte de los sistemas basados en Unix. La interfaz del kernel corresponde a la interfaz del sistema operativo, y el kernel implementa el sistema operativo completo [1]. Por el tamaño reducido de XV6, su kernel es más pequeño que algunos microkernels, pero a nivel conceptual es monolítico.

En el material se abordarán las ideas centrales entre ambos diseños.

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] David Patterson y Andrew Waterman. **The RISC-V Reader: an open architecture Atlas**. Strawberry Canyon, 2017. ISBN: 099924910X, 9780999249109.

[3] Andrew Waterman y Krste Asanovic, eds. **The RISC-V instruction set manual Volume I: unprivileged ISA**. https://github.com/riscv/riscv-isa-manual/releases/download/Ratified-IMAFDQC/riscv-spec-20191213.pdf. 2019.

[4] Andrew Waterman, Krste Asanovic y John Hauser, eds. **The RISC-V instruction set manual Volume II: privileged architecture**. https://github.com/riscv/riscv-isa-manual/releases/download/Priv-v1.12/riscv-privileged-20211203.pdf. 2021.

UNIVERSIDAD NACIONAL DE COLOMBIA

Gracias por la atención

Contacto:

csandovalc@unal.edu.co

UNIVERSIDAD NACIONAL DE COLOMBIA