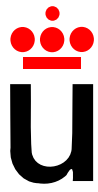
Introducción y estructura del sistema

Sistemas Operativos

Enrique Soriano, Gorka Guardiola

GSYC

11 de septiembre de 2024



Universidad
Rey Juan Carlos

Este trabajo se entrega bajo la licencia "Atribución-CompartirIgual 4.0 Internacional"[1] (cc-by-sa).

Usted es libre de:

- **Compartir**: *Copiar y redistribuir el material en cualquier medio o formato.*
- **Adaptar**: *Remezclar, transformar y construir a partir del material para cualquier propósito, incluso comercialmente.*
- **Se pueden dispensar estas restricciones si se obtiene el permiso de los autores.**
- *Las imágenes de terceros mantienen sus derechos originales.*

Historia

distintos transistores

Historia

primeros ordenadores se utilizan para descifrar código de los alemanes

- 1945-1955: todo se hacía desde cero, se usan relevadores electromecánicos y después los tubos de vacío.
- 1955-1965: aparecen lenguajes de programación (FORTRAN) y sistemas operativos básicos (bibliotecas). Aparecen los circuitos integrados, sistemas por lotes.
- 1965-1980: aparecen lenguajes de programación de alto nivel (C, COBOL), minicomputadoras, multiprogramación y tiempo compartido. Los sistemas operativos evolucionan: MULTICS y después Unix.
- 1980: aparecen los ordenadores personales, redes de ordenadores, alto nivel de integración, distintos paradigmas de lenguajes de alto nivel, sistemas operativos modernos.

programación bajo nivel mas dificil de comprender que el nivel de alto nivel (más fácil de entender), cuanto más alto el nivel de abstarcción más expresivo y más lento

¿Qué es Unix?

Unix fue un sistema operativo creado por Ken Thompson y Dennis Ritchie, empezaron en 1969:

"...the number of Unix installations has grown to 10, with more expected..."
- Dennis Ritchie and Ken Thompson, June 1972

¿Qué es Unix?

Hay muchos sistemas derivados y reimplementaciones, se llaman sistemas *Unix-like*[1]:

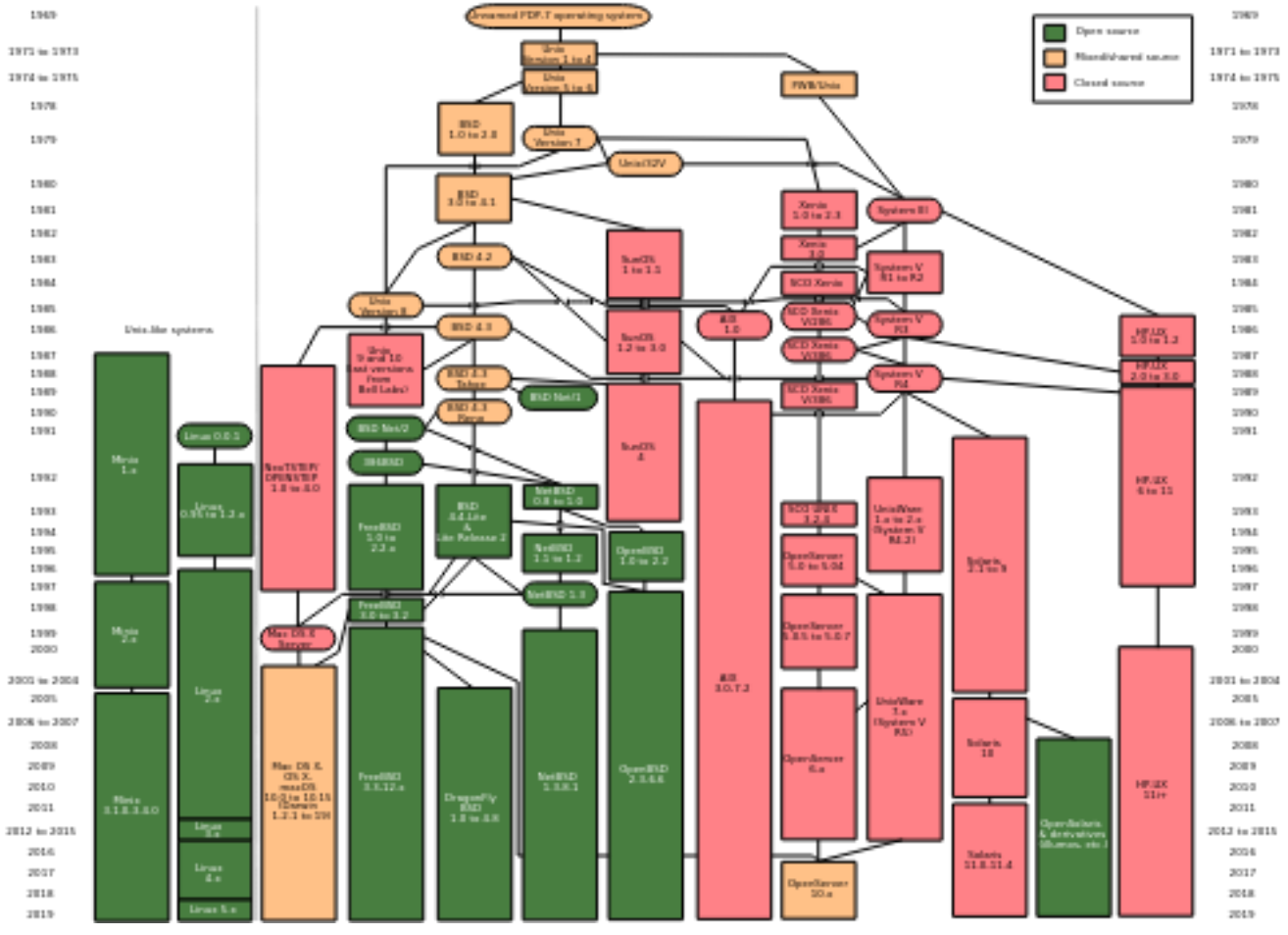
1BSD, 2BSD, 3BSD, 4BSD, 4.4BSD Lite 1, 4.4BSD Lite 2, 386 BSD, Acorn RISC iX, Acorn RISC Unix, AIX, AIX PS/2, AIX/370, AIX/6000, AIX/ESA, AIX/RT, AMiX, Android , AOS Lite, AOS Reno, AppleTV, ArchBSD, ASV, Atari Unix, A/UX, BBX, BOS, BRL Unix, BSD Net/1, BSD Net/2, BSD/386, BSD/OS, CB Unix, Chorus, Chorus/MiX, Coherent, CTIX, CXOs, Darwin, Debian GNU/Hurd, DEC OSF/1 ACP, Dell Unix, DesktopBSD, Digital Unix, DragonFly BSD, Dynix, Dynix/ptx, ekkoBSD, Eunice, FireFly BSD, FreeBSD, FreeDarwin, GNU, GNU-Darwin, Gnuppix GNU/Hurd-L4, HPBSD, HP-UX, HP-UX BLS, IBM AOS, IBM IX/370, Inferno, Interactive 386/ix, Interactive IS,iOS, iPhone OS, iPod OS, IRIS GL2, IRIX, Junos OS, Lites, LSX, macOS (Mac OS X), Mach, MERT, MicroBSD, MidnightBSD, Mini Unix, Minix, Minix-VMD, MIPS OS RISC/os, MirBSD, Mk Linux, Monterey, more/BSD, mt Xinu, MVS/ESA OpenEdition, NetBSD, NeXTSTEP, NonStop-UX, Open Desktop, Open Unix, OpenBSD, OpenDarwin, OpenIndiana, OpenServer, OpenSolaris, OPENSTEP, OS/390 OpenEdition, OS/390 Unix, OSF/1, OS X, PC-BSD, PC/IX, Plan 9, Plurix, PureDarwin, PWB, PWB/Unix, QNX, QNX RTOS, QNX/Neutrino, QUnix, ReliantUnix, Rhapsody, RISC iX, RT, SCO Unix, SCO UnixWare, SCO Xenix, SCO Xenix System V/386, Security-Enhanced Linux, Silver OS, Sinix, ReliantUnix,Solaris, SPIX, SunOS, Triance OS, Tru64 Unix, Trusted IRIX/B, Trusted Solaris, Trusted Xenix, TS, Tunis, UCLA Locus, UCLA Secure Unix, Ultrix, Ultrix 32M, Ultrix-11, Unicos, Unicos/mk, Unicos/mp, Unicox-max, UNICS, UniSoft UniPlus, Unix 32V, Unix Interactive, Unix System III, Unix System IV, Unix System V, Unix System V Release 2, Unix System V Release 3, Unix System V Release 4, Unix System V/286, Unix System V/386, Unix Time-Sharing System, UnixWare, UNSW, USG, Venix, Xenix OS, Xinu, xMach, z/OS Unix System Services, ...

bien programado

... nosotros nos centraremos en uno llamado **GNU/Linux**

[1] Ver https://www.levenez.com/unix/

¿Qué es Unix?



¿Qué es un sistema operativo?

definición laxa: programas básicos que me permiten usar el ordenador

el programa que arranca cuando enciendo el ordenador es el kernel

- Def.- Programas que te dejan usar la máquina, es subjetivo.
- Ventajas:

en cierto sentido, una vez que programas algo, lo puedes utilizar reiteradas veces (drivers)

  - Similar a una biblioteca $\rightarrow$ reutilización.
  - Abstrae de la máquina: no necesitas conocer los detalles para usarla.

a los diferentes procesos que hay ejecutando

  - Gestiona y reparte la máquina: no necesitas preocuparte de gestionar el tiempo que ejecuta un programa, organizarle la memoria, etc.

¿Qué es un sistema operativo?

- Es una **máquina abstracta**: el sistema operativo proporciona una máquina que realmente no existe, es una máquina ficticia que nos ofrece dispositivos virtuales: ficheros, directorios, procesos, conexiones de red, ventanas...
- Hoy en día tenemos distintos tipos de software de sistemas: sistema operativo, hipervisores, contenedores, etc.

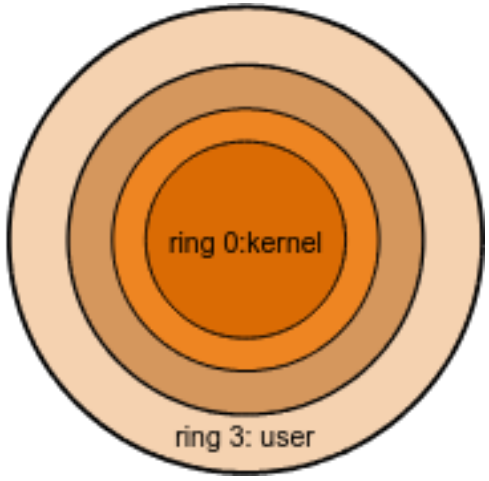
¿Qué es un sistema operativo?

Es un gestor de recursos:

- Multiplexa en tiempo: p. ej. procesador, red.
  - ¿Tienes que preocuparte de soltar el procesador en tu aplicación?
  - Ejemplo: abstracción llamada **proceso**.
- Multiplexa en espacio: p. ej. memoria, disco.
  - ¿Tienes que preocuparte de no pisar la memoria de otra aplicación?
  - Ejemplo: abstracción llamada **fichero**.

Niveles de privilegio de la CPU

Visión clásica:

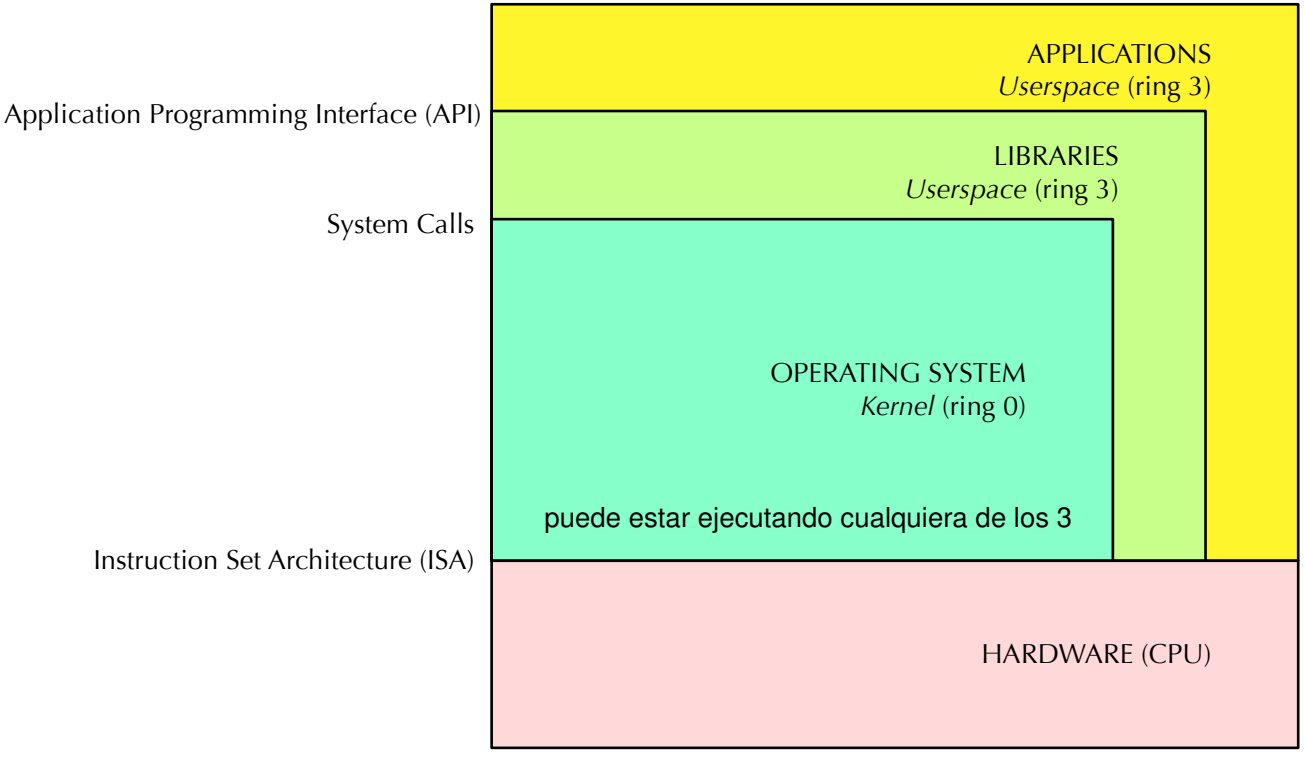


Las máquinas modernas tienen otros niveles *negativos*, el sistema operativo no es consciente de ellos:

- Ring -1: Intel VT-x, AMD-V, soporte para virtualización de sistema operativo.
- Ring -2: System Management Mode (SMM), ejecuta el firmware de la máquina para gestionar la energía, manejar errores del hardware, etc. casi todo corre en el ring -2
- Ring -3: Intel ME, sistema que que ejecuta en una CPU separada y activo en todo momento, con acceso a toda la memoria física, a las interfaces de red, etc. Este sistema se activa antes de arrancar la CPU principal.

Estructura del sistema

Importante entender bien esto

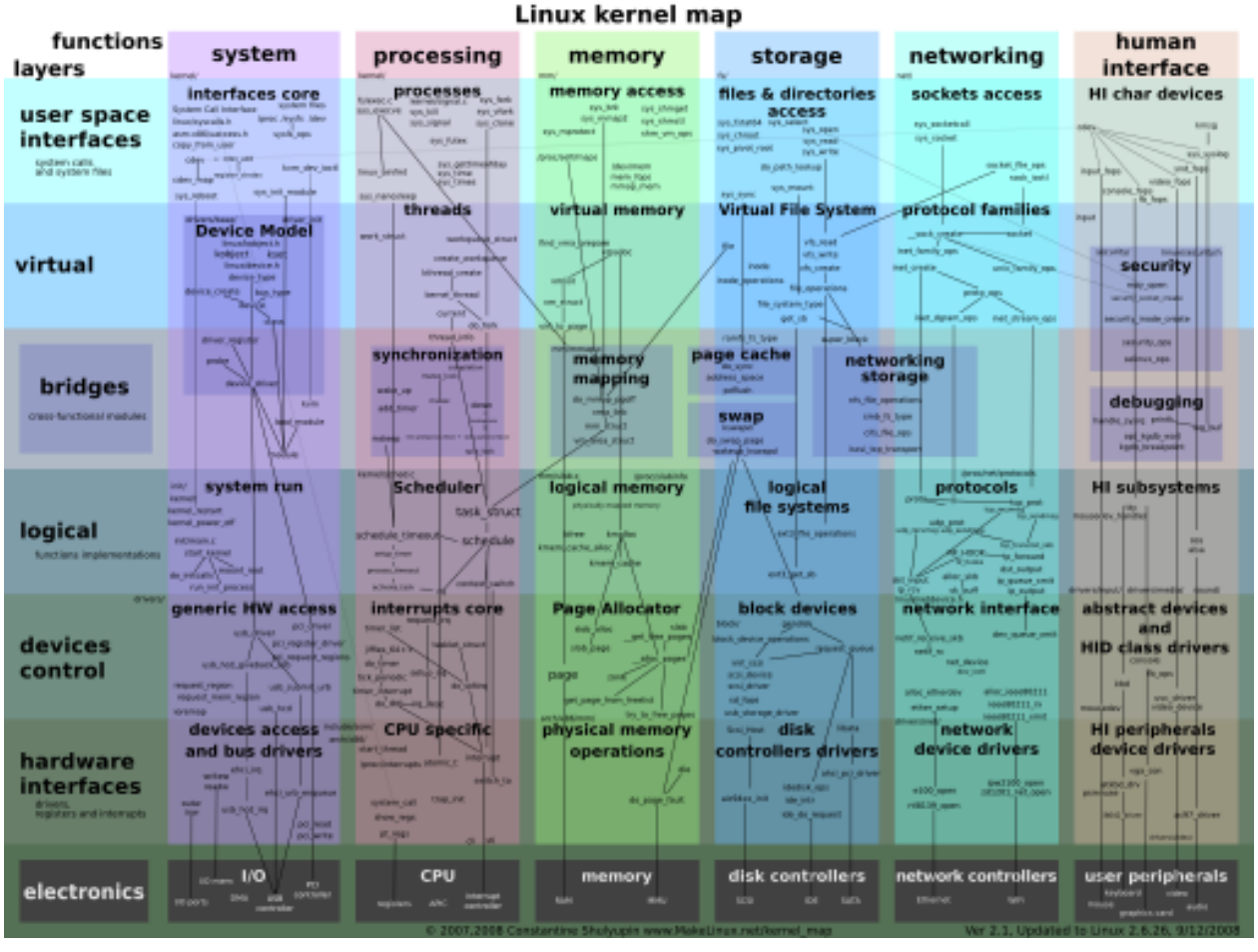


Núcleo (*kernel*)

programa principal que arranca cuando encendemos el ordenador

- Ejecución en **modo privilegiado** (ring 0): se pueden ejecutar instrucciones especiales (acceder a ciertos registros de la CPU, invalidar caches, etc.).
- Multiplexa la máquina (espacio y tiempo): implementa las **políticas y mecanismos** para repartir la CPU, memoria, disco, red,...
- Maneja el hardware: **drivers**.
- Proporciona **abstracciones**:
  - **Proceso**: programa en ejecución.
  - **Fichero**: datos agrupados bajo un nombre.
  - ...
- Da servicio al resto de programas en ejecución, que no ejecutan en modo privilegiado. Si el kernel es **reentrante**, puede dar servicio a múltiples simultáneamente. Todos los kernels de tipo Unix lo son.

Núcleo (*kernel*)



Área de usuario (*userspace, userland*)

Algoritmo: conjunto de intrucciones o pasos que me permiten hacer algo

Algoritmo -> implementación -> Programa -> compilador
(c, python...) -> intérprete

- Así ejecutan los programas del usuario (aplicaciones, herramientas, GUI, etc.).
- Se ejecutan en **modo no privilegiado** (ring 3): no se pueden ejecutar instrucciones peligrosas.
- Piden servicio al kernel realizando **llamadas al sistema**.

Procesos y programas

- Programa: conjunto de datos e instrucciones que implementan un algoritmo.
- Proceso: programa que está en ejecución, un programa vivo que tiene su propio **flujo de control** y es independiente de los otros procesos.

Procesos

Elementos fundamentales para un flujo de control:

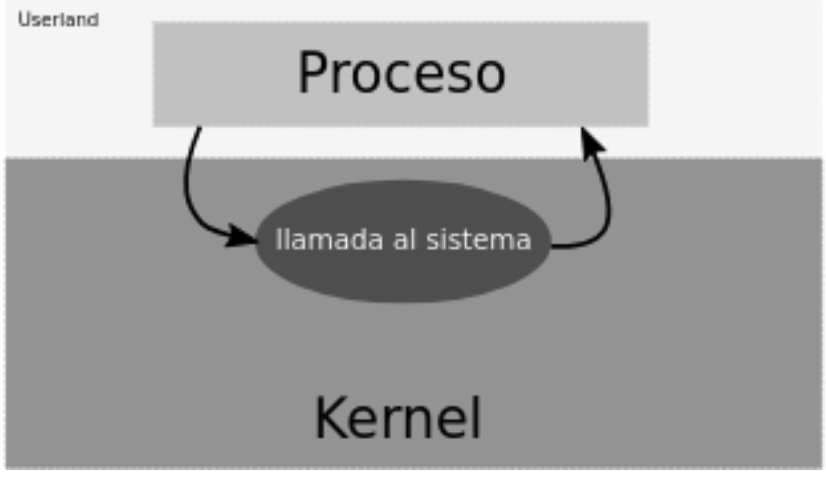
- **Contador de programa**: un registro de la CPU (PC) apunta a la instrucción por la que va ejecutando el programa.
- **Pila**: un registro de la CPU (SP) apunta a zona de la memoria donde se guardan los **registros de activación** (o marcos de pila) donde se guardan los datos necesarios para realizar llamadas a procedimiento (parámetros, variables locales, dirección del programa para retornar, etc.).

Procesos

- Procesos concurrentes: varios procesos que están ejecutando al mismo tiempo.
- El sistema operativo crea la ilusión de que cada uno tiene su propia CPU. hay dos tipos:
- Ejecución paralela vs. ejecución pseudo-paralela $\rightarrow$ para el programador es lo mismo.

Llamadas al sistema

- Es cuando el proceso *entra al kernel* por iniciativa propia, p. ej. porque necesita que el kernel realice operaciones privilegiadas para él.
- El proceso deja de ejecutar el código de la aplicación y pasa a ejecutar, en modo privilegiado, el código del kernel: se ejecuta el código del kernel *en el contexto del proceso*.
- El proceso tiene en realidad dos pilas: pila de usuario y pila kernel.

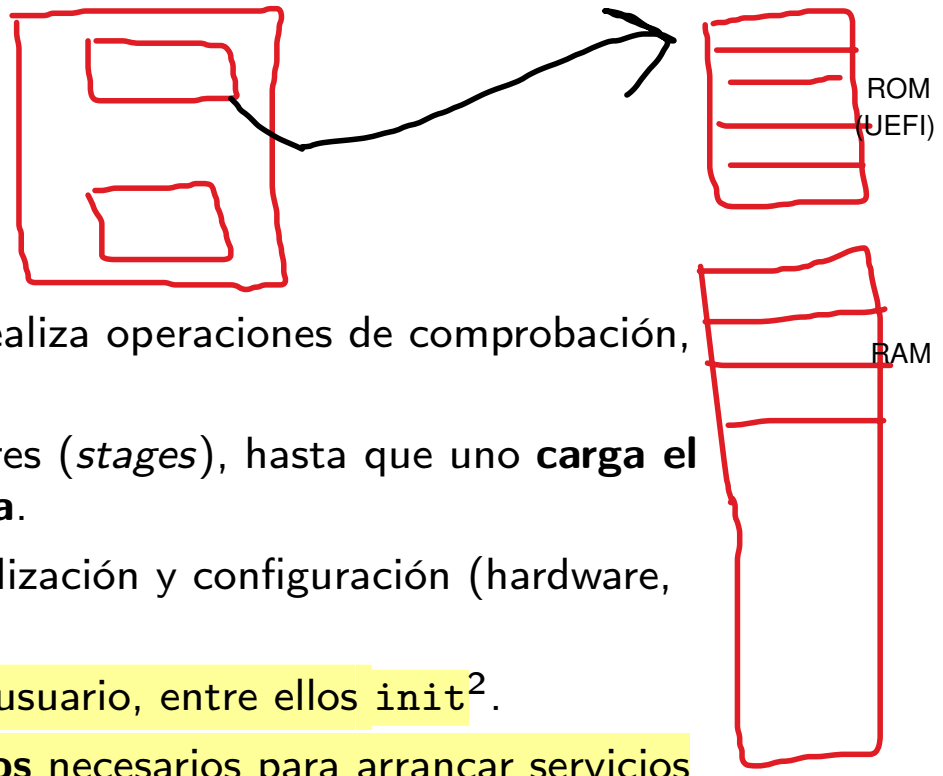


¿Cómo funciona? Ejemplo, Plan 9 para AMD64:

1. Coloca los argumentos en la **pila del proceso**.
2. Carga el **número de llamada** al sistema en un **registro**.
3. Ejecuta la instrucción `SYSCALL`, que pasa a ring 0 y salta al punto de entrada de las llamadas al sistema en el kernel (apuntado por el registro `Lstar`, configurado en tiempo de arranque) que:
   3.1 **Cambia el Puntero de Pila** (SP) para usar **pila de kernel** del proceso.
   3.2 **Guarda el contexto** del proceso en área de usuario en la **pila de kernel**.
4. **Copia los argumentos** de la llamada al sistema a la **estructura** que representa al **proceso**.
5. **Indexa la tabla** de llamadas al sistema con el número de llamada al sistema, y **llama a la función** que la implementa.

Arranque de la máquina

importante, como arranca un pc



El arranque común es el siguiente:

1. Se ejecuta el **firmware** (p. ej. **UEFI**): realiza operaciones de comprobación, carga un **cargador** (p. ej.**GRUB**).
2. El cargador puede cargar otros cargadores (*stages*), hasta que uno **carga el kernel** y se salta a su **punto de entrada**.
3. El kernel ejecuta sus funciones de inicialización y configuración (hardware, estructuras, etc.).
4. El kernel crea los primeros procesos de usuario, entre ellos `init`[2].
5. El proceso `init` **crea todos los procesos** necesarios para arrancar servicios (demonios), shells, interfaz gráfica de usuario, etc. (dependiendo de la configuración del sistema).
6. Esos procesos van creando otros procesos, siguiendo una relación padre-hijo.

[2] Actualmente, en la mayoría de las distribuciones de GNU/Linux, se arranca `systemd`.

Estructura del OS: Tipos de kernel

este es el kernel que vemos en la asignatura

Kernel **monolítico**:

linux es uno

- El kernel es un único programa.
- Pros: simplicidad, rendimiento.
- Contras: protección de los datos, puede haber falta de estructura (si se programa mal).
- Ejemplo: Linux, FreeBSD.

Pros más beneficiarios que los contras

Estructura del OS: tipos de kernel

Microkernel:

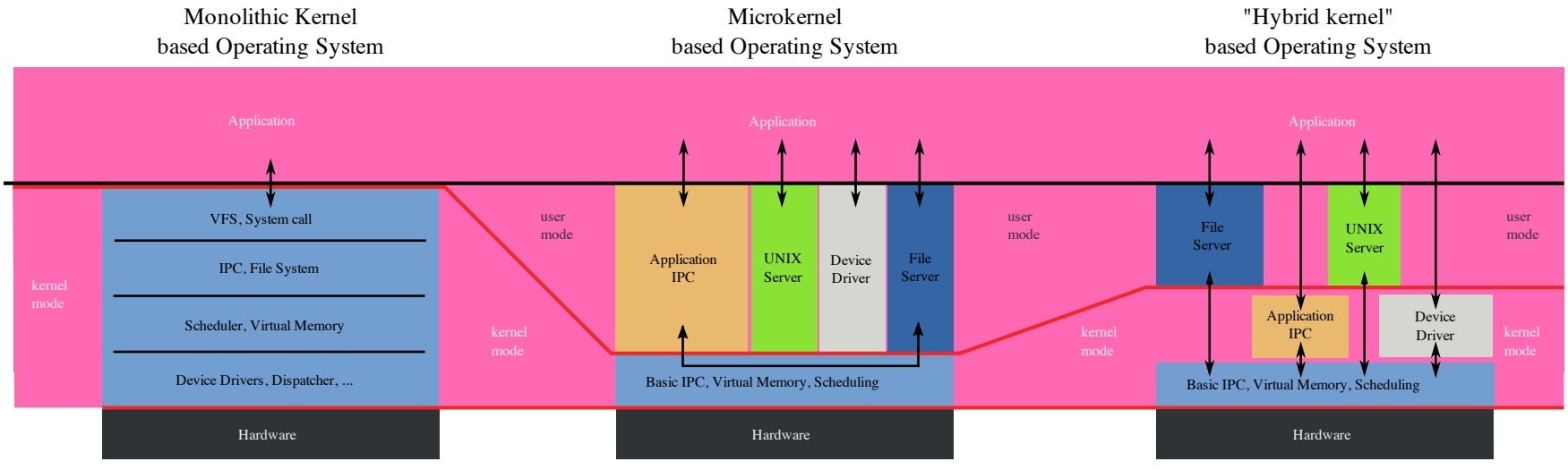
- El kernel queda reducido a lo mínimo: abstracción del HW (HAL), flujos de control, comunicación y gestión de memoria.
- El resto (*OS personality*: gestión de procesos, red, sistemas de ficheros...) se implementa en servidores independientes.
- Idea: **políticas** en espacio de usuario, **mecanismos** en espacio de kernel.
- Pros: modularidad, tolera fallos en los servidores, distribución.
- Contras: pero rendimiento en general, más complejo.
- Las nuevas generaciones tienen mejor rendimiento.
- Ejemplo: Mach, L4 y derivados.

Estructura del OS: tipos de kernel

Kernel **híbrido**:

- Compromiso entre microkernel y monolítico.
- Algunos incluyen ciertos componentes en espacio de kernel: device drivers, gestión de procesos, etc. Ejemplos: Minix, QNX.
- Otros simplemente sólo siguen un diseño de microkernel, pero todos los servidores están en espacio de kernel. Ejemplos: XNU (OSX), Windows NT.

Estructura del OS: tipos de kernel



Kernel **modular**

La mayoría de los kernels actuales permiten la **carga dinámica** de módulos para ampliar/reducir su funcionalidad sin la necesidad de rearrancar el sistema.

- Ventaja: sólo se cargan los drivers necesarios $\rightarrow$ ahorro de memoria.
- Desventaja: seguridad.
- Ejemplos: Linux (.ko), Mac OSX (.kext), FreeBSD (.kld), Windows (.sys).

Módulos en GNU/Linux

Comandos:

- `lsmod` escribe en su salida la lista de módulos cargados.
- `modprobe` carga un módulo.
- `rmmod` descarga un módulo.

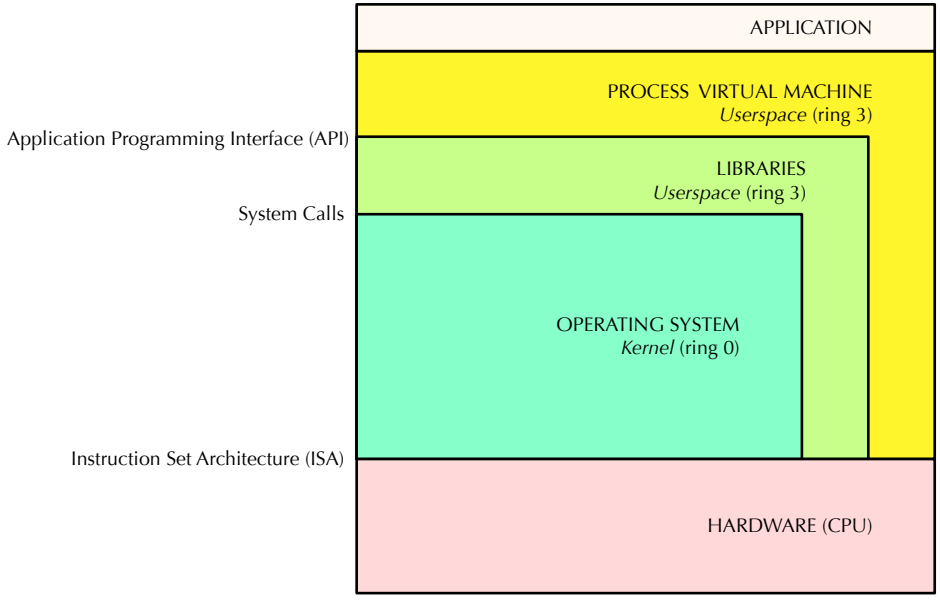
Los módulos están en `/lib/modules/`*`versión-de-kernel`*`/`

- `uname -r` escribe en su salida la versión del kernel que estamos ejecutando.

Virtualización

- La virtualización es muy común hoy en día (cloud computing, etc.).
- Existen distintos tipos de *máquinas virtuales*:
  - **Máquina virtual de proceso:** tiene como objetivo proporcionar una plataforma para ejecutar un único programa: emuladores de otra ISA (p. ej. OSX Rosetta), optimizadores, ISA virtuales (p. ej. Java VM, .NET).
  - **Máquina virtual de sistema:** un hipervisor (VMM o VM Monitor) proporciona un entorno completo y persistente para ejecutar un sistema operativo completo.

Máquinas Virtuales de Proceso

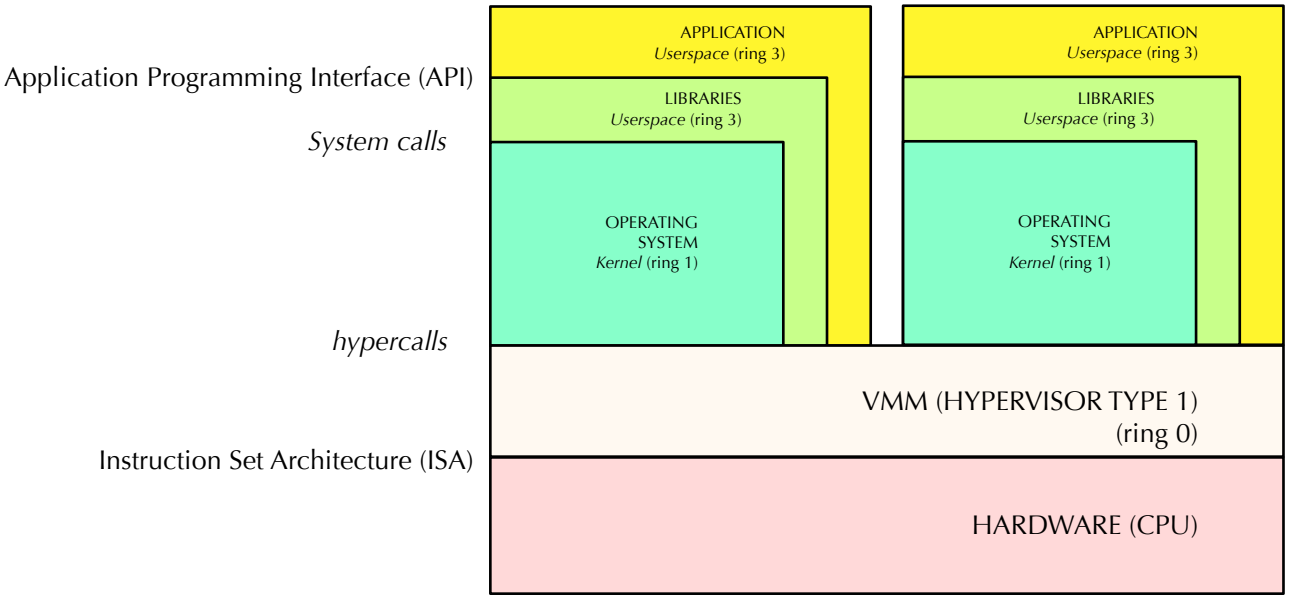


Máquinas Virtuales de Sistema

- VM clásica (hypervisor type 1 a.k.a. *bare metal* a.k.a. *unhosted*).
  - Paravirtualización.
  - Virtualización completa asistida por HW.
- VM alojada (hypervisor type 2).

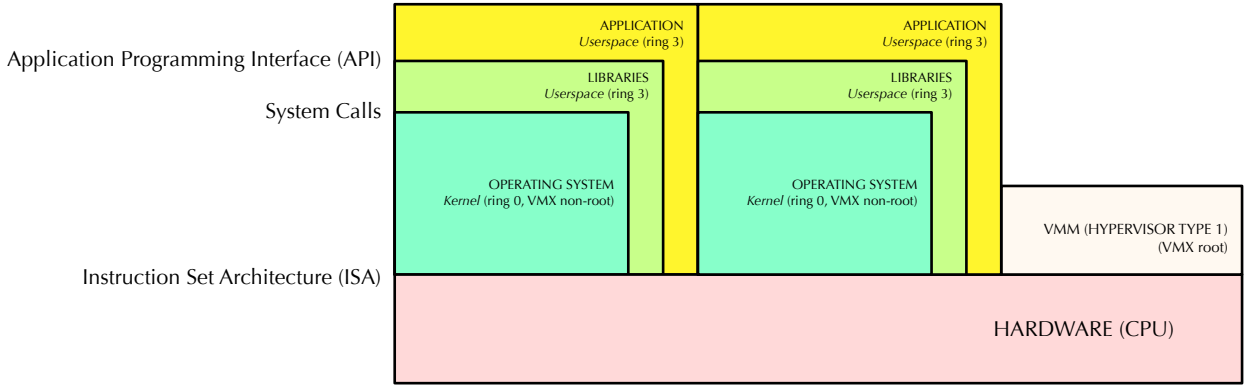
Máquinas Virtuales de Sistema: paravirtualización

- El OS huésped está modificado para ejecutar sobre el VMM.
- El OS huésped realiza *hypercalls* para gestionar la tabla de páginas, configurar el HW, etc.
- Ejemplos: Xen, KVM.



Máquinas Virtuales de Sistema: asistidas por HW

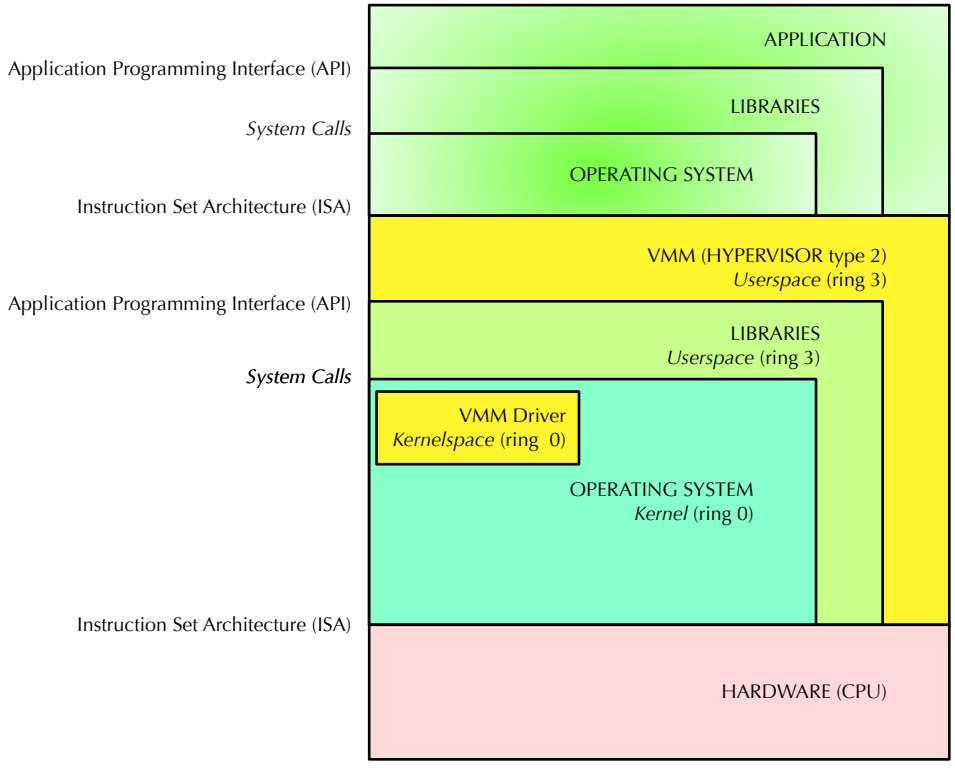
- Se basa instrucciones especiales de la CPU para virtualización. P. ej. Intel VT-x, AMD-V.
- Instrucciones VMX: activar el modo VMX root, lanzar una VM, pasar el control al VMM, retomar una VM, etc.
- Además de ring 0-3, hay un modo especial en el que ejecuta el VMM: VMX root.
- El OS huésped no necesita modificaciones.
- Ejemplo: VMware vSphere.



Máquinas Virtuales de Sistema: alojada

- La VM se aloja sobre otro OS.
- El VMM puede instalar drivers en el OS anfitrión para mejorar el rendimiento. P. ej. VMWare Fusion, Virtual Box.
- *Whole-system* VM: la ISA de la VM no es la misma que la del HW y necesita **emulación**. P. ej. Virtual PC.

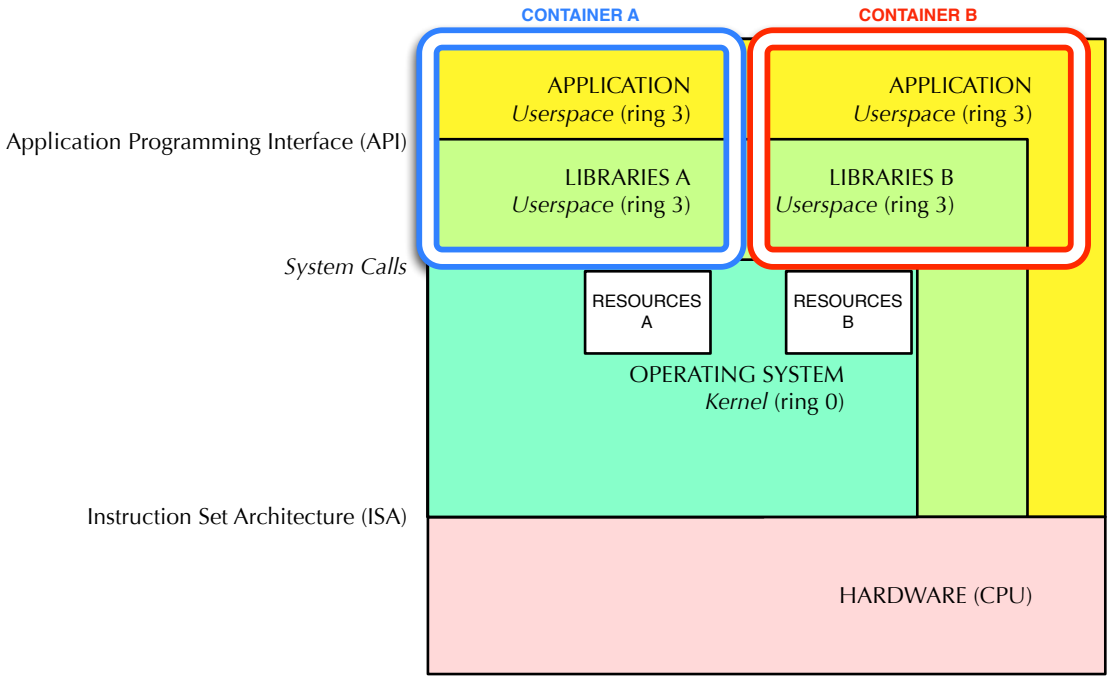
Máquinas Virtuales de Sistema: alojada



Virtualización a nivel del SO: contenedores

- Una VM aisla distintas **imágenes completas** de distintos sistemas operativos ejecutando. Si lo que queremos aislar es un servicio, pagamos cierto coste ejecutando un OS completo para él (**tiempo en arrancar y parar la VM**, rendimiento, etc.).
- Contenedor: dentro del mismo sistema operativo (kernel) se pueden crear distintos entornos aislados, cada uno con sus propias abstracciones y recursos (espacio de procesos, sistema de ficheros raíz, CPU, recursos de red, usuarios, etc.).
- Pros: arranque rápido, más ligeros en general, no necesitas una imagen entera del sistema alojado.
- Contras: menos aislamiento, menos seguridad.
- Ejemplos: Docker, Linux Containers (LXC), OpenVZ, FreeBSD Jails, Solaris Zones, etc.

Virtualización a nivel del SO: contenedores



En este curso nos centraremos en el el **sistema operativo**, sus partes fundamentales y su uso efectivo.