Contenido 1: Introducción Sistemas Operativos y llamadas al sistema

Lesly Reyes
lareyes2@uc.cl

Álvaro Quintana
alvaroq@uc.cl

Departamento de Ciencias de la Computación
Pontificia Universidad Católica Chile

August 12, 2024

Contenido

Introducción
- Estructura de un Sistema Operativo

¿Qué hace un sistema operativo?
- Un sistema operativo tiene múltiples roles
- ¿Qué incluye un sistema operativo?
- Estructura de un sistema operativo
- ¿Cómo se invocan los servicios del sistema operativo?
- Historia Sistemas operativos
- Arquitecturas

¿Cómo funciona un sistema Operativo?

Syscalls: Llamadas al sistema

Material Adicional

Introduction

¿Por qué existen los sistemas operativos?

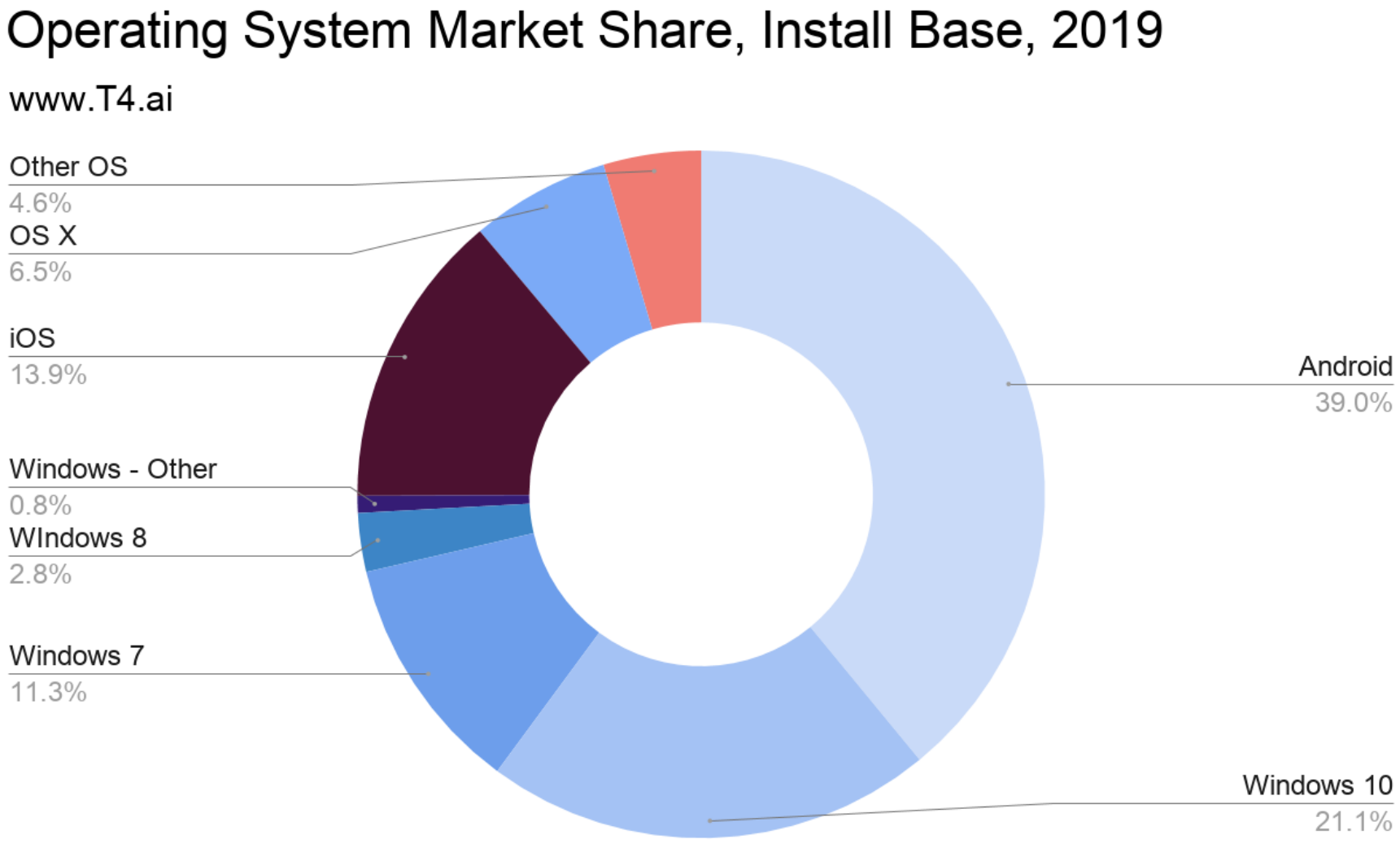
Los computadores de hoy tienen uno o más procesadores, algo de memoria principal, discos, un teclado, un trackpad, una pantalla, interfaces de red y varios otros dispositivos de entrada/salida. En definitiva, es un sistema complejo.
Si cada programador tuviera que entender cómo funcionan todas estos componentes en detalle, y administrarlos nunca programaría. Gestionar todos estos componentes y utilizarlos de forma óptima es un trabajo sumamente desafiante. Por esta razón, las computadoras están equipadas con un capa de software llamada sistema operativo.

Un sistema operativo tiene como trabajo proporcionar a los programas de usuario un modelo de computadora mejor, más simple y más limpio, y manejar la administración de todos los recursos que se acaban de mencionar.

Introducción

Ejemplos de sistemas operativos

Windows, Linux, FreeBSD, or OS X



¿Qué hace un sistema operativo?

Un sistema operativo tiene múltiples roles

- **Para el usuario**:

  Le permite utilizar los recursos del computador
  Ejecutar programas, leer archivos, efectuar cálculos ...

- **Para el sistema computacional**:

  Para un administrador:

  Le permite seleccionar y proveer los recursos necesarios. Y permite que se usen de manera eficiente.

  Para un programa de control que evita el uso indebido

  No "botar" otros programas en ejecución.
  No sobreescribir por error el SO
  Evitar que un usuario tome control del sistema

¿Qué incluye un sistema operativo?

Componentes

Sistema operativo

Kernel + Programas del sistema

kernel

Es un programa que **siempre** se esta ejecutando. Provee funcionalidad mínima, como acceso a CPU, Memoria, dispositivos.

Programa del sistema

Extienden las funciones del **kernel**.

Estructura de un sistema operativo

El sistema operativo habla con el hardware. El SO es software.
El sistema operativo:

- Nos abstrae del hardware
- Nos permite compartir recursos
- Nos ofrece servicios para hacer más cómodo el uso del hardware
- Puede correr en distintas arquitecturas, single core...

Estructura de un sistema operativo

¿Dónde vive el SO

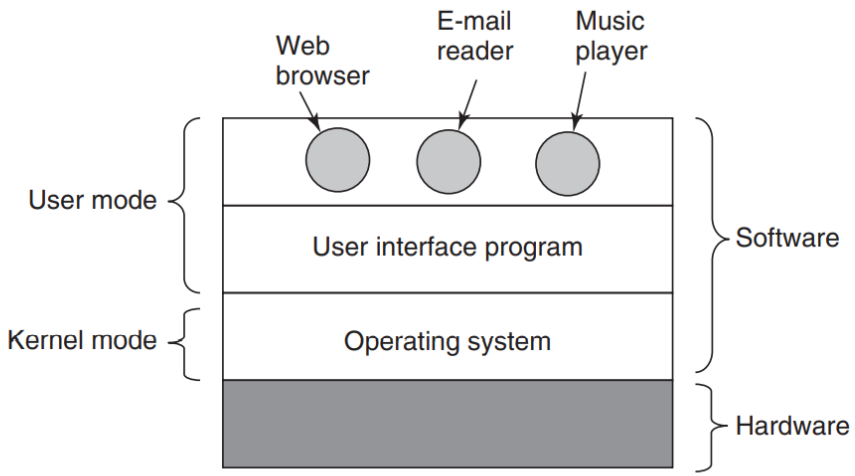
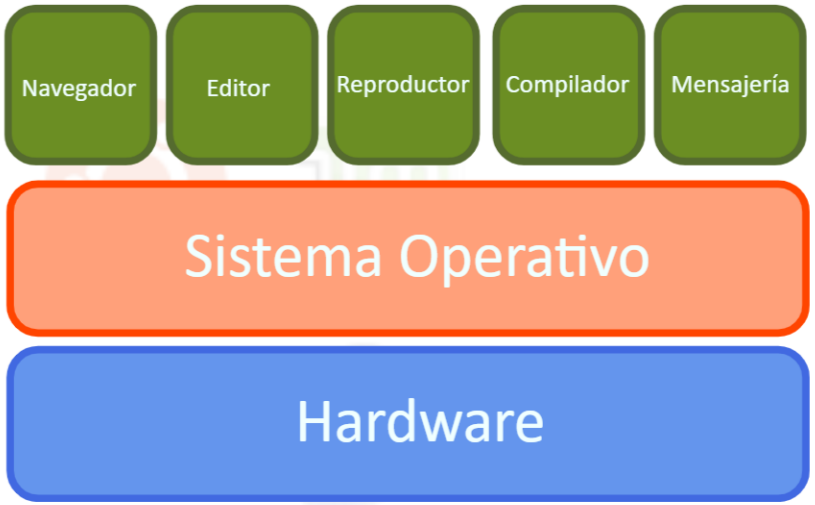


Figure 1-1. Where the operating system fits in.

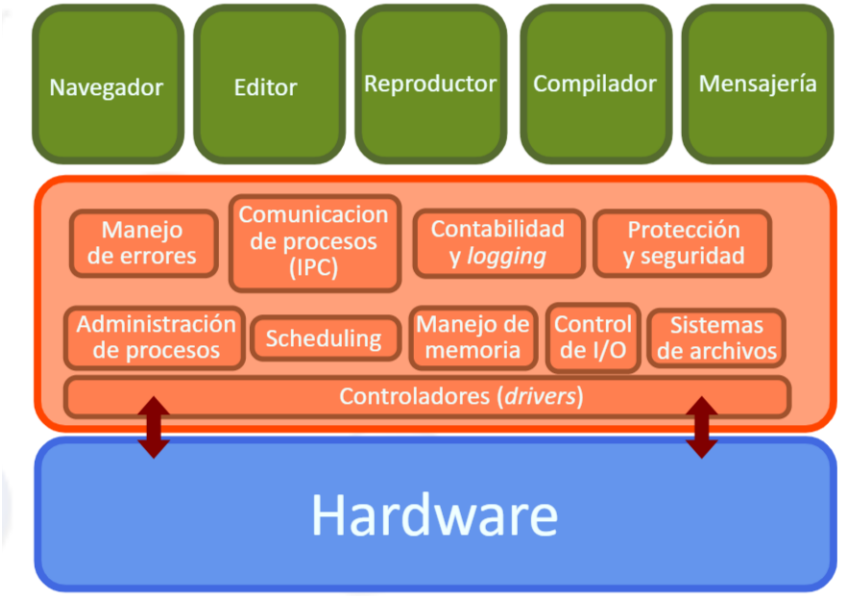
Estructura de un sistema operativo

¿Dónde vive el SO?



Estructura de un sistema operativo

¿Qué hay dentro de un sistema operativo?



Estructura de un sistema operativo

¿Qué hay dentro de un sistema operativo?

- Administración de procesos
- Manejo de memoria
- Sistemas de archivos
- Protección y seguridad
- Control de I/O
- Manejo de errores
- Comunicación inter-procesos
- Contabilidad / logging
- Controladores (drivers)
- etc . . .

¿Cómo se invocan los servicios del sistema operativo?

Syscalls

Sistema operativo provee una interfaz de llamadas al sistema (syscalls)

- Cada vez que un programa desea solicitar algo al sistema operativo, debe invocar una syscall
- "Por favor, sistema operativo, haz esto por mi"
- Cada sistema operativo puede proveer un conjunto distinto de syscalls
- Sistemas operativos proveen bibliotecas (libraries) en algún lenguaje de programación (C) para invocar syscalls.

Sistema operativo provee interfaces de usuario (user interface) que enmascara syscalls

¿Cómo se invocan los servicios del sistema operativo?

GUI, CLI, BATCH

Sistema operativo provee interfaces de usuario (user interface) que enmascara syscalls

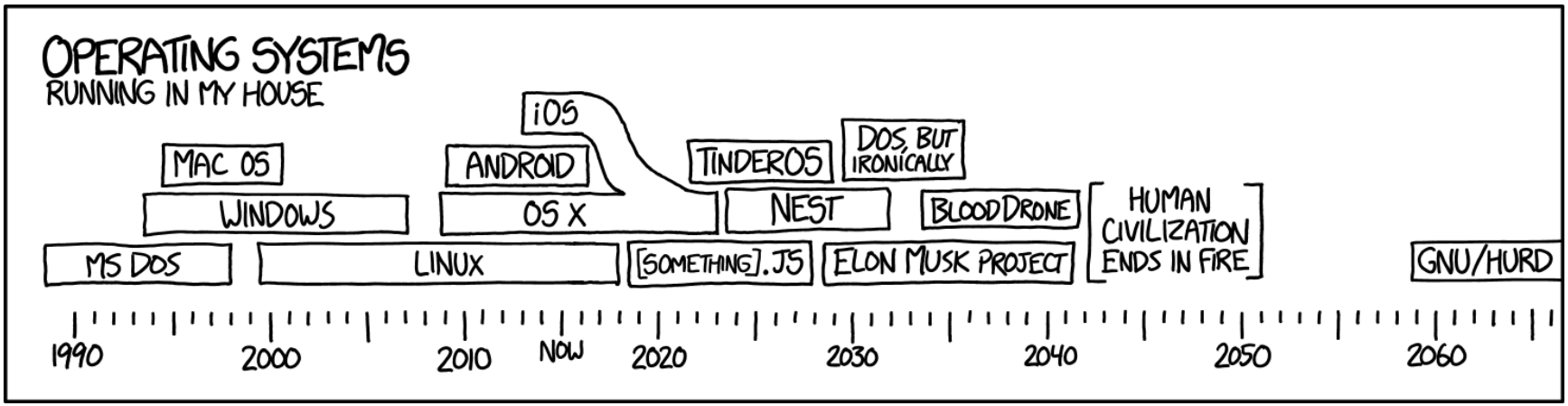
- Ofrecen un entorno más "amigable" para usar el computador

Distintas maneras de interactuar con el sistema operativo

- Interfaces gráficas: GUI (Graphical User Interface)
- Intérprete de comandos (Command Line)
- secuencias de comandos no-interactivas

Historia Sistemas operativos

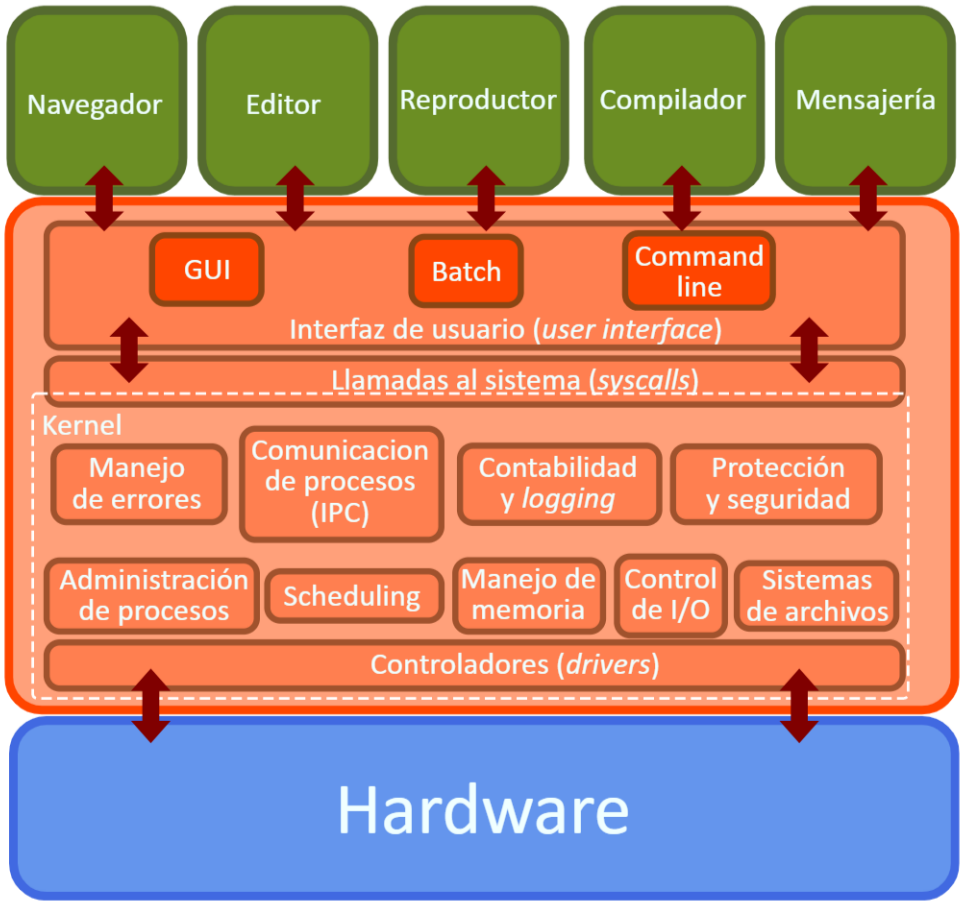
A través del tiempo



Arquitecturas

Monolítico, Microkernel y Híbrido

- Monolítico
- Microkernel
- Híbrido



Arquitecturas

¿Cómo se organiza el kernel?

Muchos componentes a integrar

- Primeros sistemas operativos empezaron pequeños y luego crecieron
- No tenían mucha estructura

MS-DOS (1981)

- Sistema monolítico (un gran kernel estático) Primeros sistemas operativos empezaron pequeños y luego crecieron
- Programas de usuario podían acceder directamente al hardware
- Hardware (Intel 8088) no proveía modos de protección

Arquitecturas

Kernel monolítico

Todo dentro en un único programa

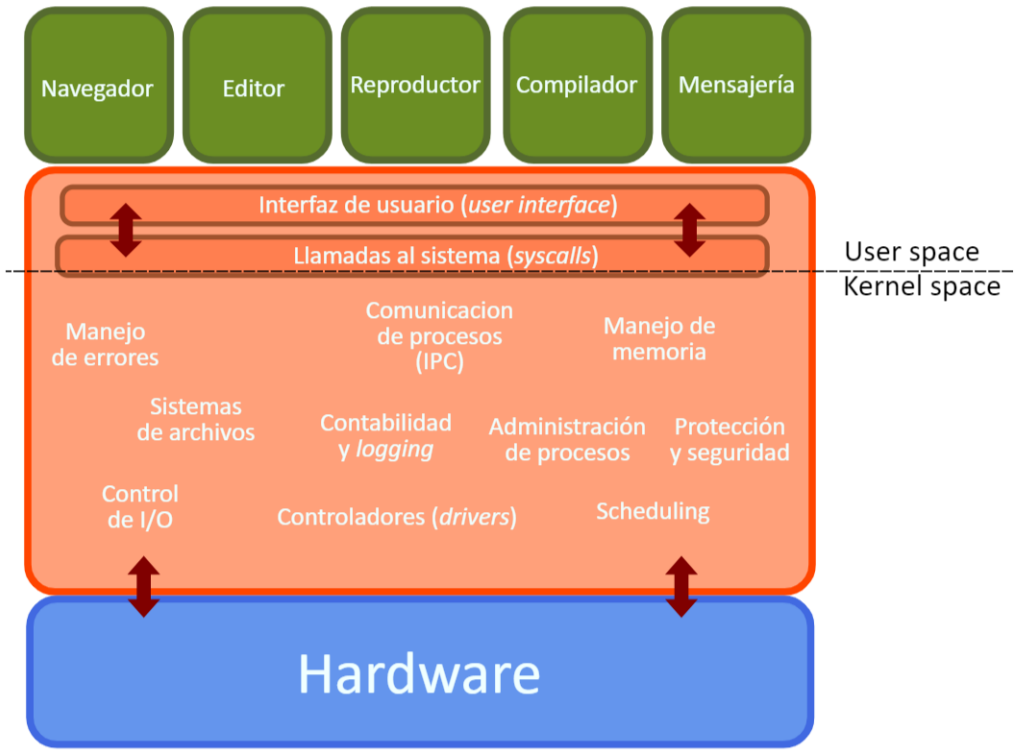
- Servicios para el sistema y para el usuario en el mismo espacio
- Todos los servicios se ejecutan en modo kernel
- La falla de un servicio compromete al kernel
- Kernel complejo y de mayor tamaño

Ejemplos

- Unix (BSD, System V)
- Linux
- MS-DOS, Windows 95/98/Me

Arquitecturas

Kernel monolítico



Arquitecturas

Kernel monolítico

Todo en un único programa extensible

- Componentes pueden ser incluidos al momento de compilar, o bien durante la ejecución como módulos
- Módulos extienden al kernel
- Módulos se ejecutan en el espacio del kernel
- Ayuda a controlar el tamaño del kernel (footprint)

Arquitecturas

Desventajas Kernel monolítico

Como todos los archivos se ejecutan en una sola imagen, si falla algún servicio, falla todo el sistema. Si el usuario necesita agregar cualquier servicio nuevo, entonces el usuario necesita modificar todo el sistema. No es fácil de mantener y depurar.

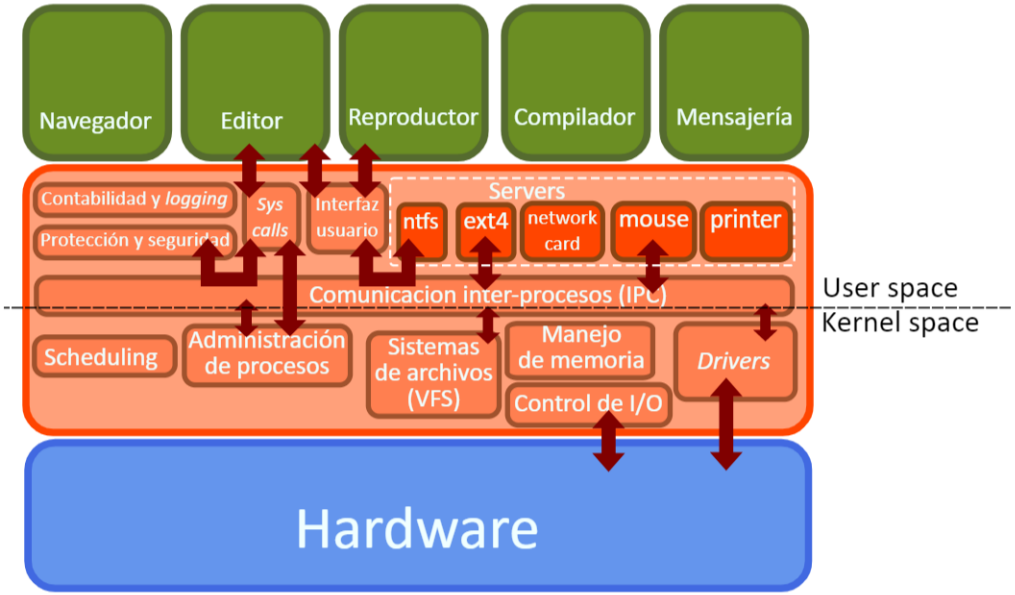
Arquitecturas

Se siguen usando Kernel monolítico

Como no hay espacio de usuario y espacio de kernel separados, la ejecución del proceso será más rápida en los kernels monolíticos.
Ventajas: proporciona programación de CPU, programación de memoria, administración de archivos a través de llamadas al sistema.
Linux, BSD (OpenBSD, NetBSD, FreeBSD), Solaris, DOS, OpenVMS, etc

Arquitecturas

Microkernel lo más pequeño posible



Arquitecturas

Microkernel lo más pequeño posible

Mantener el kernel del menor tamaño posible

- Solo servicios básicos en el kernel
- Funcionalidades (servers) se ejecutan en user space
- Errores en los servicios no afectan al kernel
- Kernel más sencillo, pequeño y fácil de portar
- Ejecución con overhead por comunicaciónAyuda a controlar el tamaño del kernel (footprint)
- Procesos se comunican (IPC) mediante paso de mensajes

Ejemplos

- Mach, implementación de UNIX con microkernel (CMU)
- GNU/Hurd, inspirado en Mach
- L4
- MINIX 3, desarrollado por Andrew S. Tanenbaum

Arquitecturas

Desventajas Microkernel

La ejecución en microkernel es lenta ya que la comunicación entre la aplicación y el hardware del sistema se establece a través del paso de mensajes.

Aislamiento de fallas

- el diseño del micronúcleo ayuda a aislar las fallas y evitar que afecten a todo el sistema. Si un servidor u otro componente falla, se puede reiniciar o reemplazar sin causar interrupciones al resto del sistema.

Modularidad

- debido a que el núcleo y los servidores se pueden desarrollar y mantener de forma independiente, el diseño del micronúcleo permite una mayor modularidad. Esto puede facilitar la adición y eliminación de funciones y servicios del sistema.

Arquitecturas

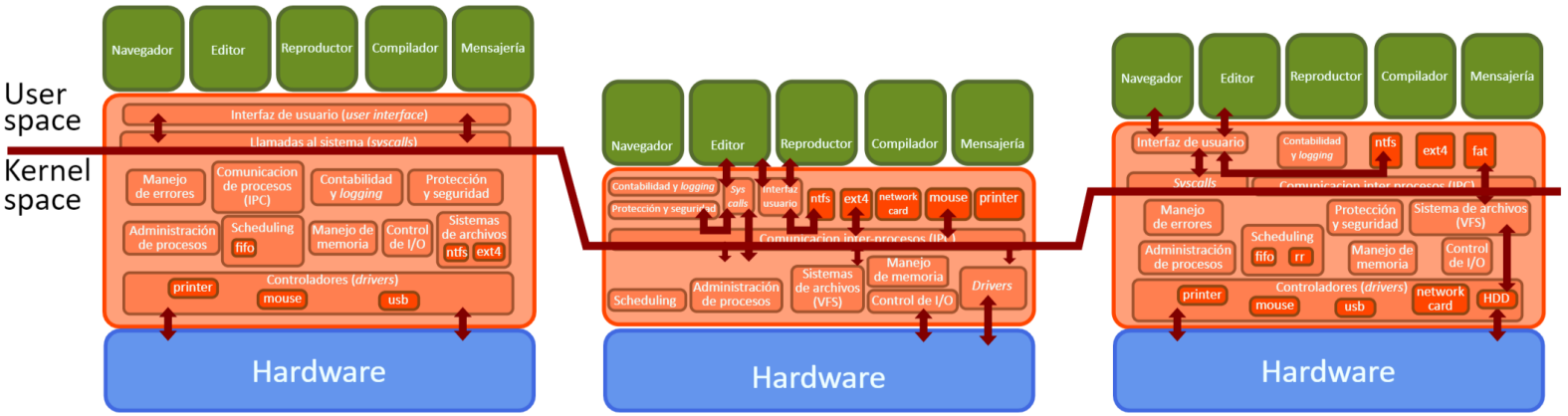
Monolíto vs Microkernel

Monolítico

- *Kernel* de mayor tamaño y complejo
- Todos los servicios en *kernel space*
- Falla en un servicio afecta a todo el *kernel*
- Ejecución más rápida
- Difícil de extender, pero puede usar módulos

Microkernel

- *Kernel* pequeño y simple
- Servicios mínimos en *kernel space*. Mayoría en *user space*
- Falla en un servicio queda aislada
- Ejecución con *overhead* por comunicación
- Fácil de extender



¿Cómo funciona un sistema Operativo?

Modos de protección

Al iniciar el computador se ejecuta código de la tarjeta madre

- **BIOS**: Basic Input/Output System, o bien
- **UEFI**: Unified Extensible Firmware Interface

Este código tiene dos misiones fundamentales

- Inicializar/identificar el hardware
- Ubicar y arrancar mediante un bootloader el kernel en modo privilegiado

El kernel es el único software que funciona en modo privilegiado

¿Cómo funciona un sistema Operativo?

Modos de protección

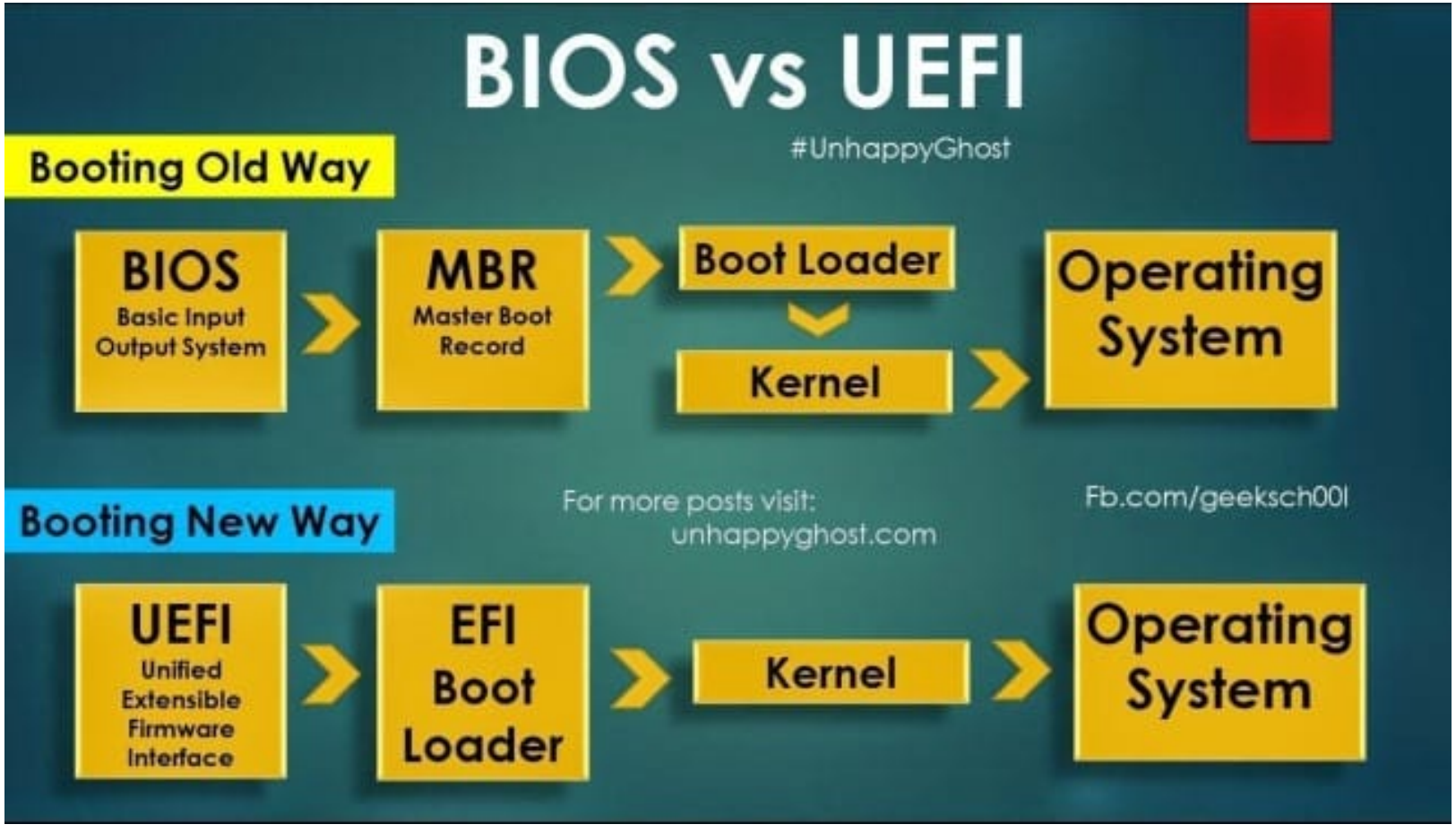


Figure: Difference Bios vs UEFI

¿Cómo funciona un sistema Operativo?

Modos de protección

El hardware posee dos modos de protección

- **Kernel Mode**: acceso completo al hardware
- **User Mode**: acceso a un set restringido de instrucciones

Si se está en user mode el hardware genera una interrupción (protection fault), o bien ignora la instrucción

¿Cómo funciona un sistema Operativo?

Modos de protección

Instrucciones privilegiadas (ring 0)

- Modificar vector de interrupciones (CLI, IRET)
- Acceder a dispositivos de I/O
- Modificar timer del computador
- Detener el computador (halt, HLT)
- Cambiar el modo de protección (user a kernel, o viceversa)
- Modificar tablas de acceso a memoria (pagetable, INVLPG)
- Almacenar estado de la máquina (SMSW)

Estas instrucciones son un ejemplo de las instrucciones privilegiadas del set de instrucciones Intel 64.

Hardware moderno posee 4 o más modos de protección

- Se les llama rings (anillos) de protección
- Permite implementar virtualización
- Intel 64 posee 4 niveles y una lista de instrucciones privilegiadas

¿Cómo funciona un sistema Operativo?

Modos de protección

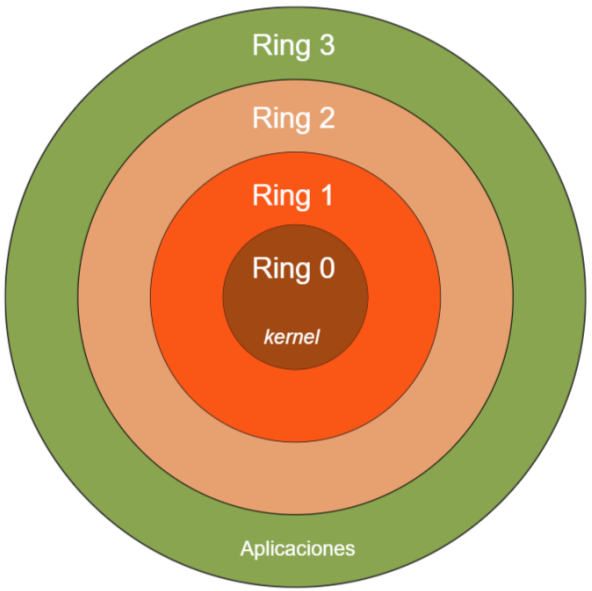


Figure: Anillo de privilegios

¿Cómo funciona un sistema Operativo?

Modos de protección

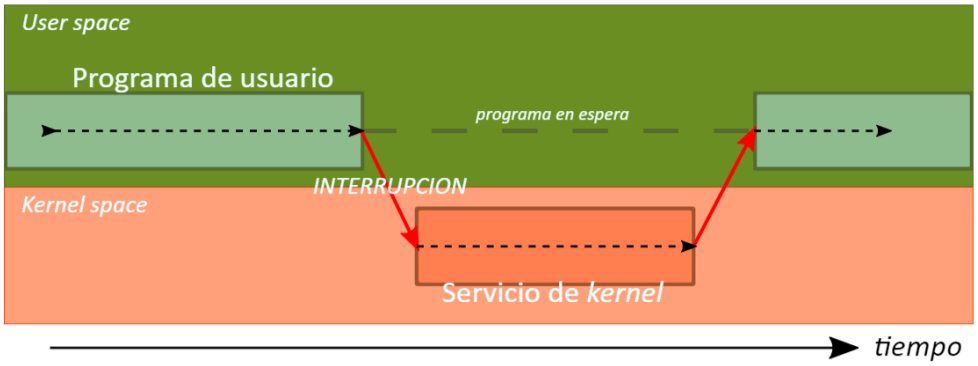
Mientras nadie lo llame, el sistema operativo no hace nada.
Como es el flujo entre el usuario que intenta ejecutar algo.

- Primero: El programa de usuario genera un **trap**: interrupción generada por software.
- Segundo: Se pasará el control al sistema operativo: kernel space
- Tercero: Sistema operativo ejecuta servicio solicitado por el usuario
- Por útlimo: Una vez terminado el servicio, sistema operativo regresa el control al programa de usuario.

¿Cómo funciona un sistema Operativo?

Modos de protección

Como es el flujo entre el usuario que intenta ejecutar algo.



En la imagen se puede ver que cuando el programa queda en espera el sistema operativo ejecutará el servicio solicitad

Syscalls: Llamadas al sistema

Interfaz para solicitar servicios al sistema operativo

- Las syscalls son la manera precisa de llamar al sistema operativo.
- Sistemas proveen bibliotecas que implementan syscalls.

Syscalls: Llamadas al sistema

¿Qué pasa cuando copio un archivo?

¿Qué pasa cuando copio un archivo?

copy source.txt dest.txt

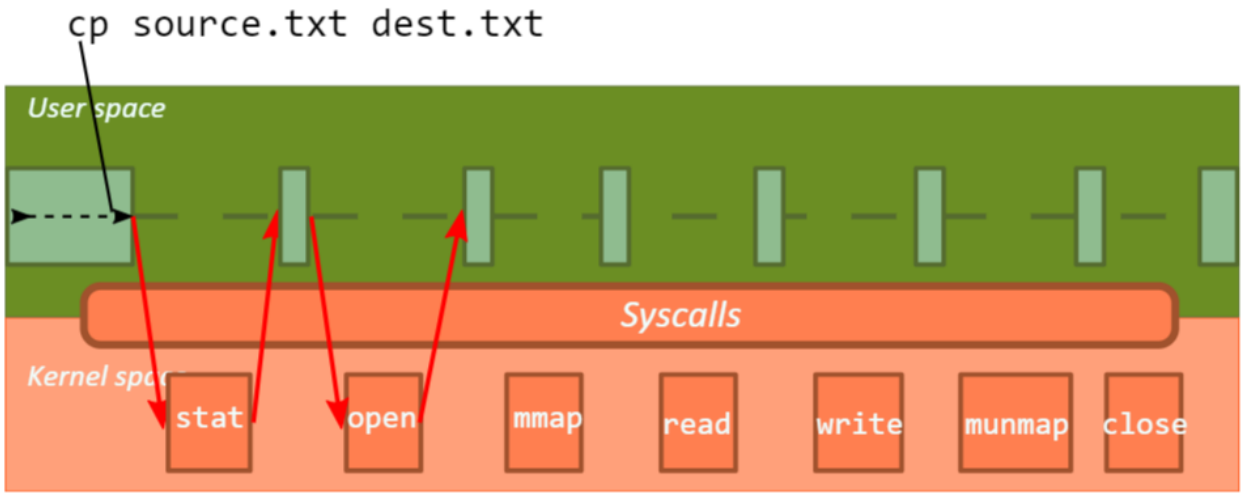
- Primero abre source.txt (stat, open)
- El SO va chequear: ¿si existe?, ¿tiene permiso de lectura?, . . .
- Luego va a leer source.txt hacia alguna región de memoria (mmap, read)
- Después va escribir desde la memoria hacia dest.txt (write)
- Por último liberará la memoria y guardar los archivos (munmap, close)

Todas estas acciones requieren llamadas al sistema

Pruebe con: strace cp source.txt dest.txt

Syscalls: Llamadas al sistema

¿Qué pasa cuando copio un archivo?



Syscalls: Llamadas al sistema

Definir una interfaz estándar ayuda a la portabilidad

- POSIX, interfaz definida en IEEE1003.1-2017
- SCI, System Call Interface. Interfaz de syscalls de Linux
- Linux kernel API: interfaz SCI + implementación en glibc
- Windows NT Native API, interfaz para Windows NT. Implementada en ntdll.dll
- Windows API (Win32), en kernel32.dll, advapi32.dll, user32.dll

Syscalls: Llamadas al sistema

¿Qué es POSIX?

POSIX no es en realidad una cosa. Describe una cosa, como una etiqueta. Imagine una caja etiquetada: POSIX, y dentro de la caja hay un estándar. Un estándar consta de conjuntos de reglas e instrucciones de las que se ocupa POSIX. POSIX es la abreviatura de interfaz de sistema operativo portátil. Es un estándar IEEE1003.1 que define la interfaz de idioma entre los programas de aplicación (junto con los shells de línea de comandos y las interfaces de utilidades) y el sistema operativo UNIX.

Syscalls: Llamadas al sistema

¿Qué es POSIX?

El cumplimiento del estándar garantiza la compatibilidad cuando los programas UNIX se mueven de una plataforma UNIX a otra. El enfoque de POSIX está principalmente en las funciones de System V UNIX y BSD UNIX de AT&T.

Syscalls: Llamadas al sistema

¿Cómo se ejecuta una syscall?

Procedimiento es dependiente del hardware

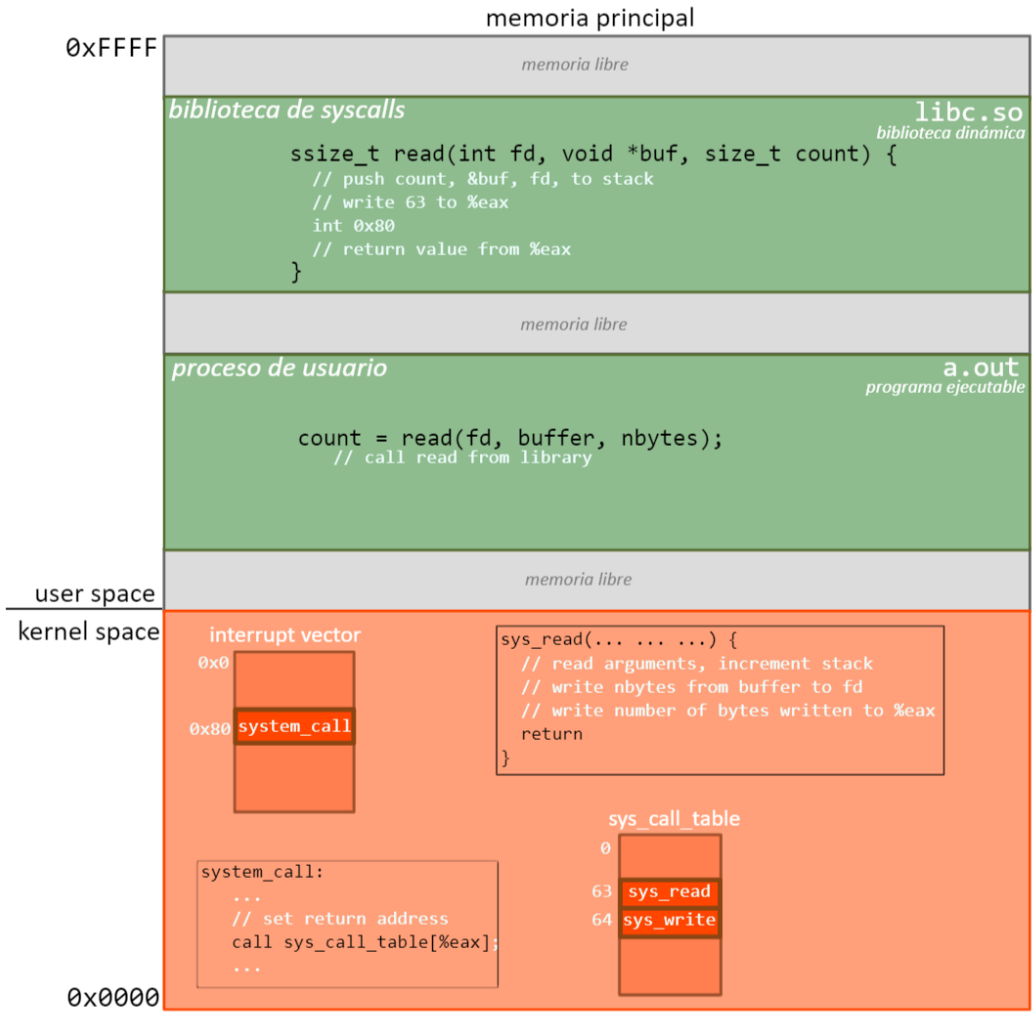
Participan:

- Proceso ejecutable de usuario
- Biblioteca (estática o dinámica) que invoca la syscall
- Vector de interrupciones del hardware, configurado por el sistema operativo
- Código de syscalls del sistema operativo, que utilizan el hardware

¿Qué hace read? RTFM

Syscalls: Llamadas al sistema

¿Cómo se ejecuta una syscall?



Syscalls: Llamadas al sistema

Algunas syscalls de POSIX.

Process management

Call	Description
pid = fork( )	Create a child process identical to the parent
pid = waitpid(pid, &statloc, options)	Wait for a child to terminate
s = execve(name, argv, environp)	Replace a process' core image
exit(status)	Terminate process execution and return status

File management

Call	Description
fd = open(file, how, ...)	Open a file for reading, writing, or both
s = close(fd)	Close an open file
n = read(fd, buffer, nbytes)	Read data from a file into a buffer
n = write(fd, buffer, nbytes)	Write data from a buffer into a file
position = lseek(fd, offset, whence)	Move the file pointer
s = stat(name, &buf)	Get a file's status information

Directory- and file-system management

Call	Description
s = mkdir(name, mode)	Create a new directory
s = rmdir(name)	Remove an empty directory
s = link(name1, name2)	Create a new entry, name2, pointing to name1
s = unlink(name)	Remove a directory entry
s = mount(special, name, flag)	Mount a file system
s = umount(special)	Unmount a file system

Miscellaneous

Call	Description
s = chdir(dirname)	Change the working directory
s = chmod(name, mode)	Change a file's protection bits
s = kill(pid, signal)	Send a signal to a process
seconds = time(&seconds)	Get the elapsed time since Jan. 1, 1970

Syscalls: Llamadas al sistema

Algunas syscalls de POSIX y su equivalencia en Win32

- Es natural que no todas las syscall tenga una equivalencia.
- Cada sistema provee su propio conjunto de instrucciones para interactuar con el usuario.

UNIX	Win32	Description
fork	CreateProcess	Create a new process
waitpid	WaitForSingleObject	Can wait for a process to exit
execve	(none)	CreateProcess = fork + execve
exit	ExitProcess	Terminate execution
open	CreateFile	Create a file or open an existing file
close	CloseHandle	Close a file
read	ReadFile	Read data from a file
write	WriteFile	Write data to a file
lseek	SetFilePointer	Move the file pointer
stat	GetFileAttributesEx	Get various file attributes
mkdir	CreateDirectory	Create a new directory
rmdir	RemoveDirectory	Remove an empty directory
link	(none)	Win32 does not support links
unlink	DeleteFile	Destroy an existing file
mount	(none)	Win32 does not support mount
umount	(none)	Win32 does not support mount, so no umount
chdir	SetCurrentDirectory	Change the current working directory
chmod	(none)	Win32 does not support security (although NT does)
kill	(none)	Win32 does not support signals
time	GetLocalTime	Get the current time

Material Adicional

- Video explicativo Arquitecturas de Kernel: Link Video
- Video explicativo Sistemas operativos: Link Video
- Video explicativo de syscalls: Link Video
- Tutorial para crear tu propia syscalls: Link

Próximo contenido a revisar serán Procesos