Sistemas Operativos
Fisop 2022

El Kernel

La Perspectiva del Usuario

¿Qué vé el usuario?

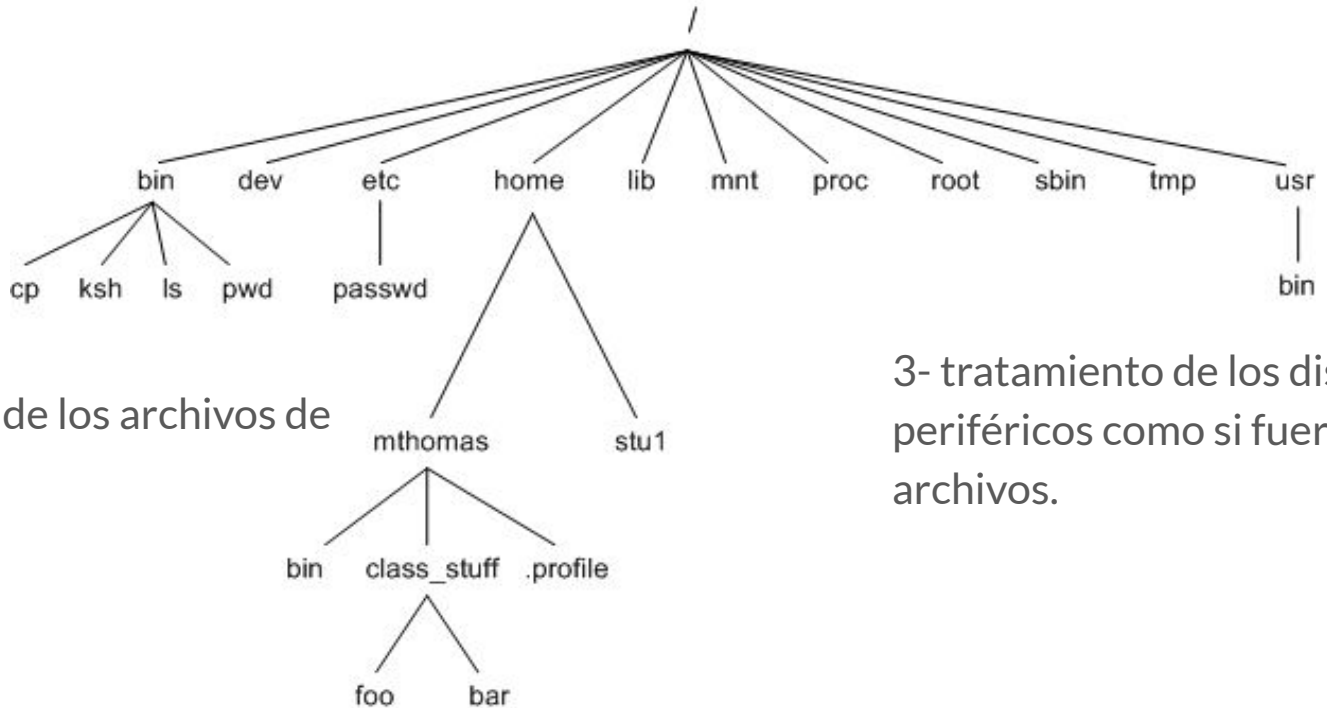
Desde el punto de vista del usuario lo que “se ve” es:

- El **Sistema de Archivos** o File System.
- El **Entorno de Procesamiento**.
- Los **Bloques Primitivos de Construcción**.

1- estructura jerárquica

El Sistema de Archivos

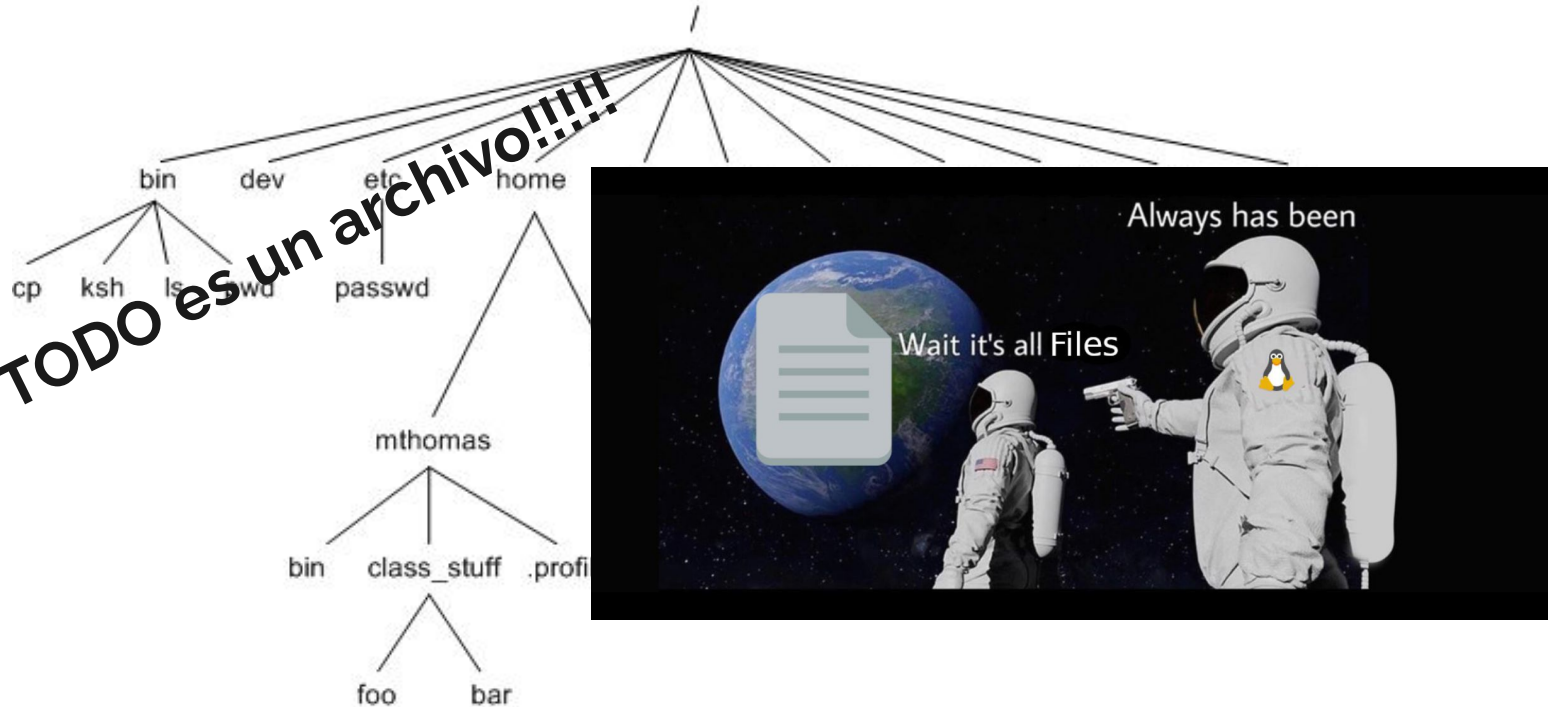
2- habilidad de crear y borrar archivos



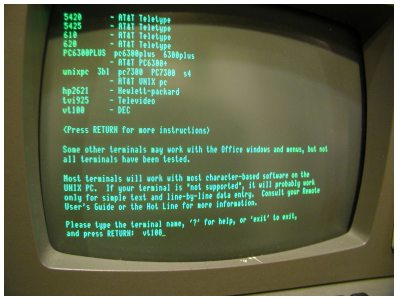
3- tratamiento de los dispositivos periféricos como si fueran archivos.

4- protección de los archivos de datos

El Sistema de Archivos



El Entorno de Procesamiento



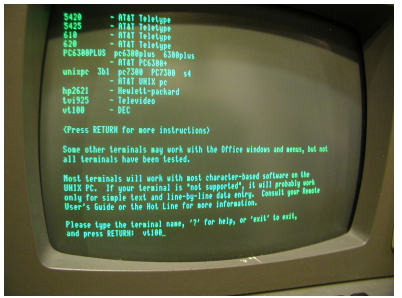
El intérprete de comandos o shell es el primer programa que se ejecuta (en un unix) en modo usuario, este a su vez ejecuta el comando de login.

El intérprete de comandos captura las teclas que son presionadas en el teclado y las traduce a nombres de comandos.

El shell puede interpretar tres tipos de comandos:

- Comandos Ejecutables simples programas , por ejemplo el ls
- Shell scripts
- Estructuras de control del Shell if-then-else-fi

El Entorno de Procesamiento



- El shell ejecuta los comandos buscando en ciertos directorios en una determinada secuencia.
- Dado que el shell es un programa no forma parte del Kernel del sistema operativo

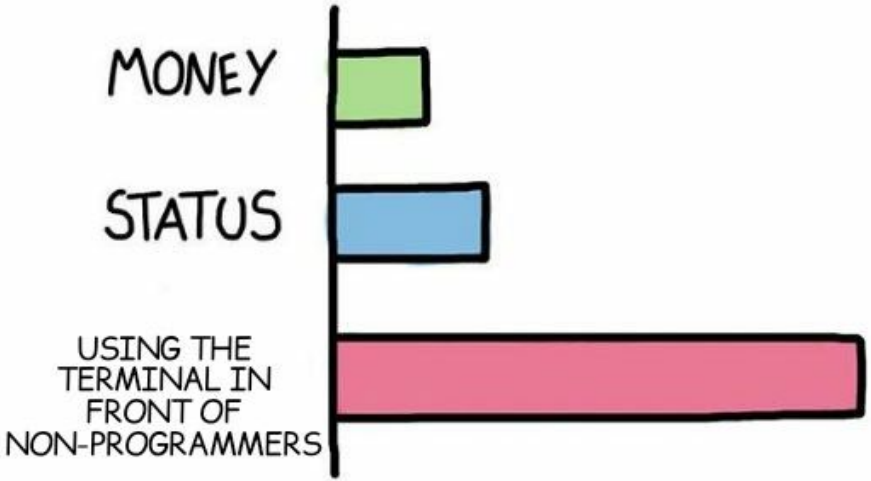
- El sistema operativo provee también unas System Calls para la creación y manipulación de procesos:
  - **fork():** crea un proceso hijo a partir de un proceso padre.
  - **execl(),execve():** creación de procesos.
  - **wait():** sincronización

Bloques Primitivos de Construcción

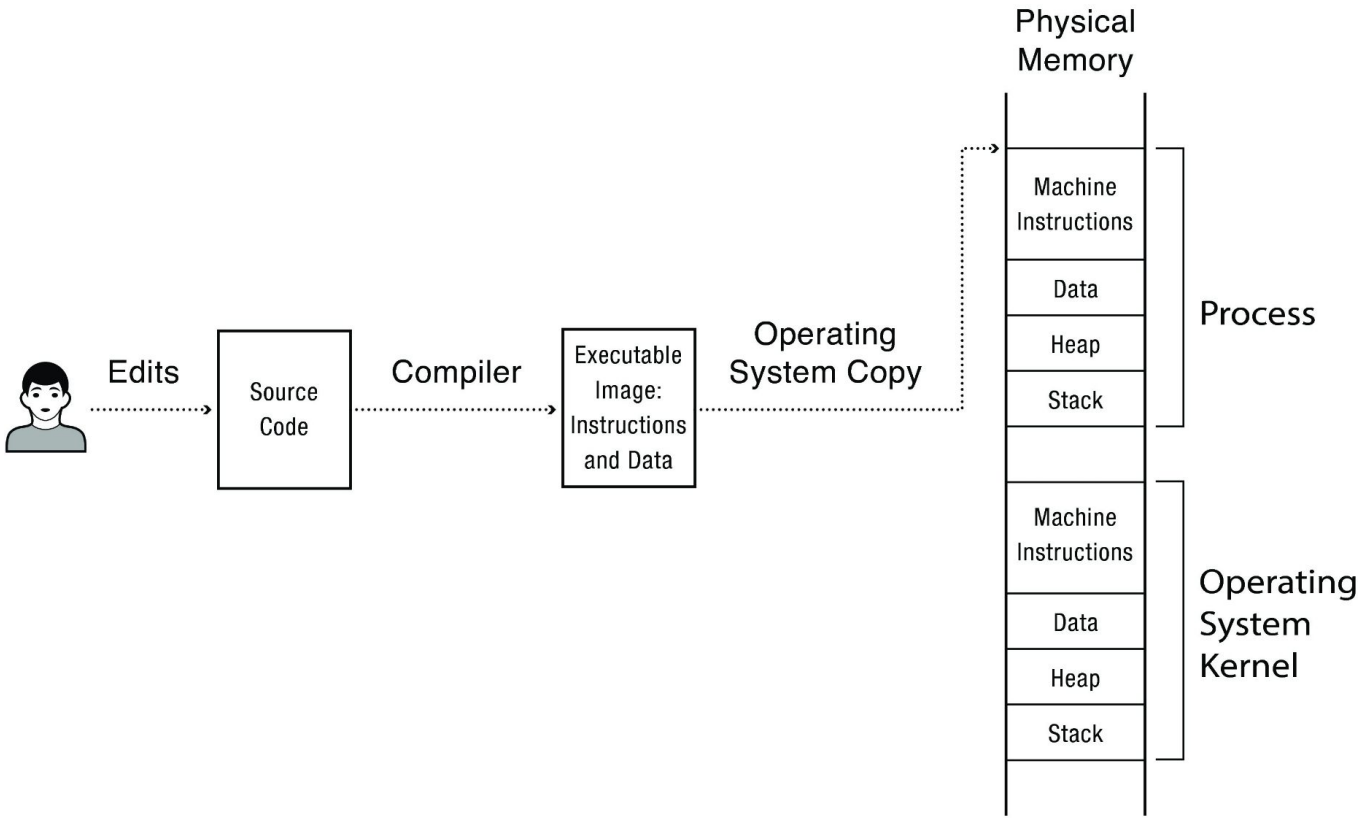
- La idea de unix es la de proveer ciertos mecanismos para que los programadores construyan a partir de pequeños programas otros más complejos.

- El **redireccionamiento de entrada y salida** de datos.
- El operador > **redirecciona el standar output a un archivo**.
- El operador < **redirecciona el standar input tomando los datos de un archivo**.
- Las tuberías o **los pipes**.

WHAT GIVES PEOPLE FEELINGS OF POWER

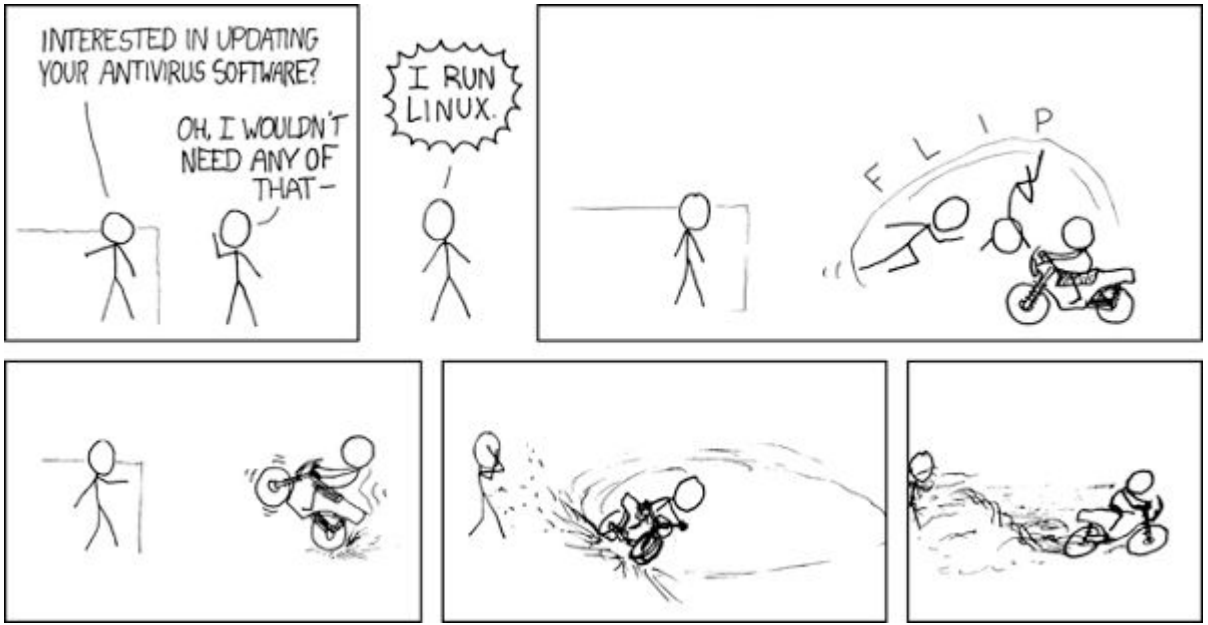


El kernel



De Programa a Proceso

¿Por Qué es Necesario el Kernel?



El rol central de un sistema operativo es la protección

Ejecución Directa

El sistema Operativo

El SO crea un punto de entrada en la lista de Procesos

Reserva memoria para alojar al programa

Carga el Programa en la memoria
Setea el Stack con argv/argc

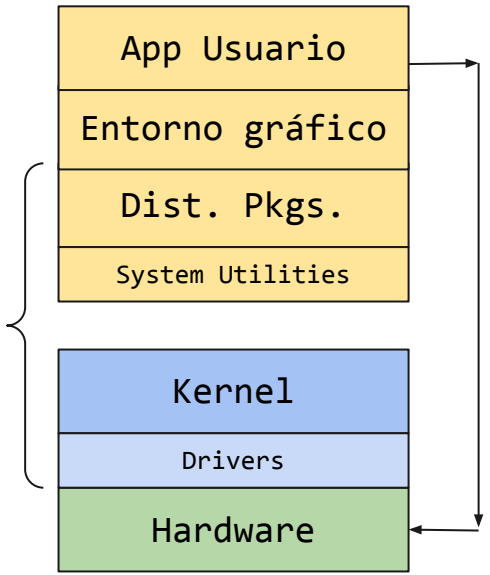
Limpia los registros
Ejecuta la llamada a main()

El Programa

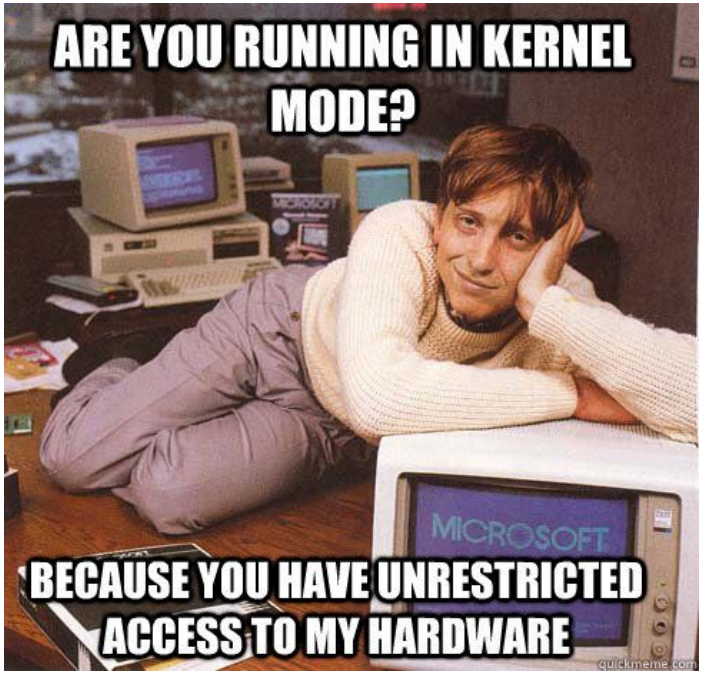
Corre main()
Ejecuta return 0

Libera la memoria reservada

Elimina la entrada en la lista de Procesos



Ejecución Directa



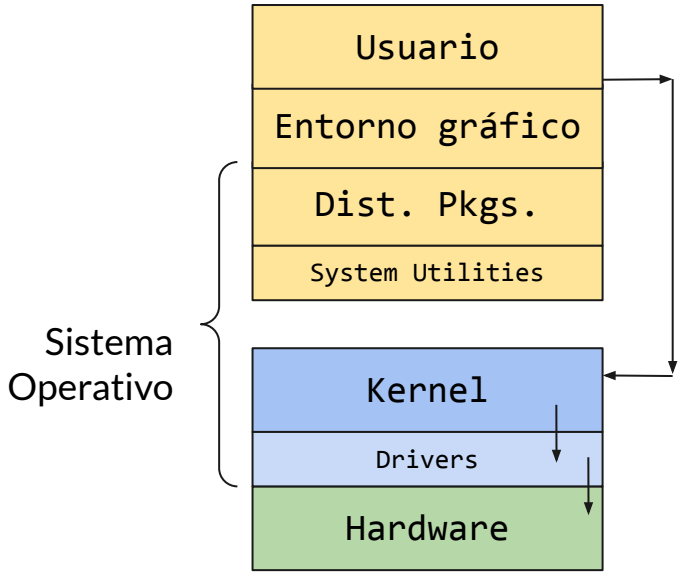
Ejecución Directa

Ejecución Directa

Ejecución Directa

Limitar la Ejecución Directa

La ejecución directa es PELIGROSA!

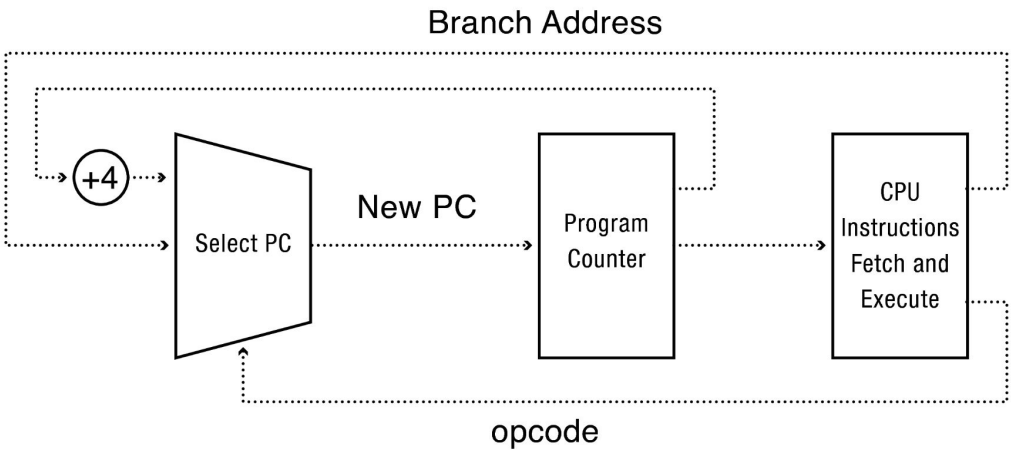


Modo Dual de Operaciones

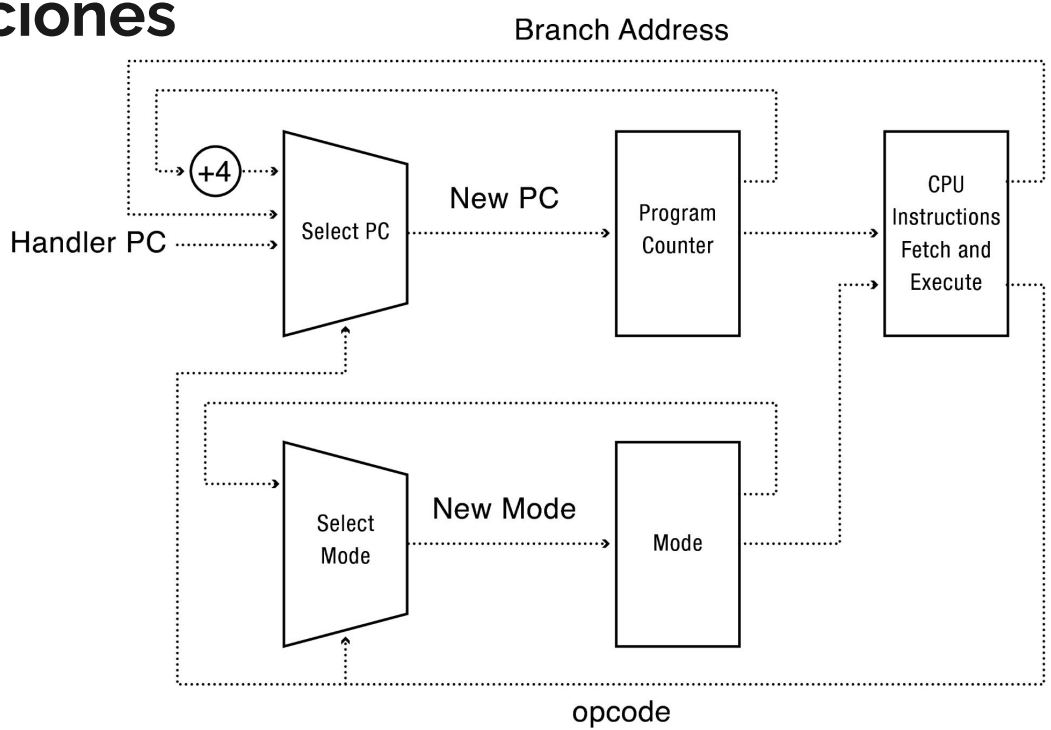
User-Mode

Kernel-Mode

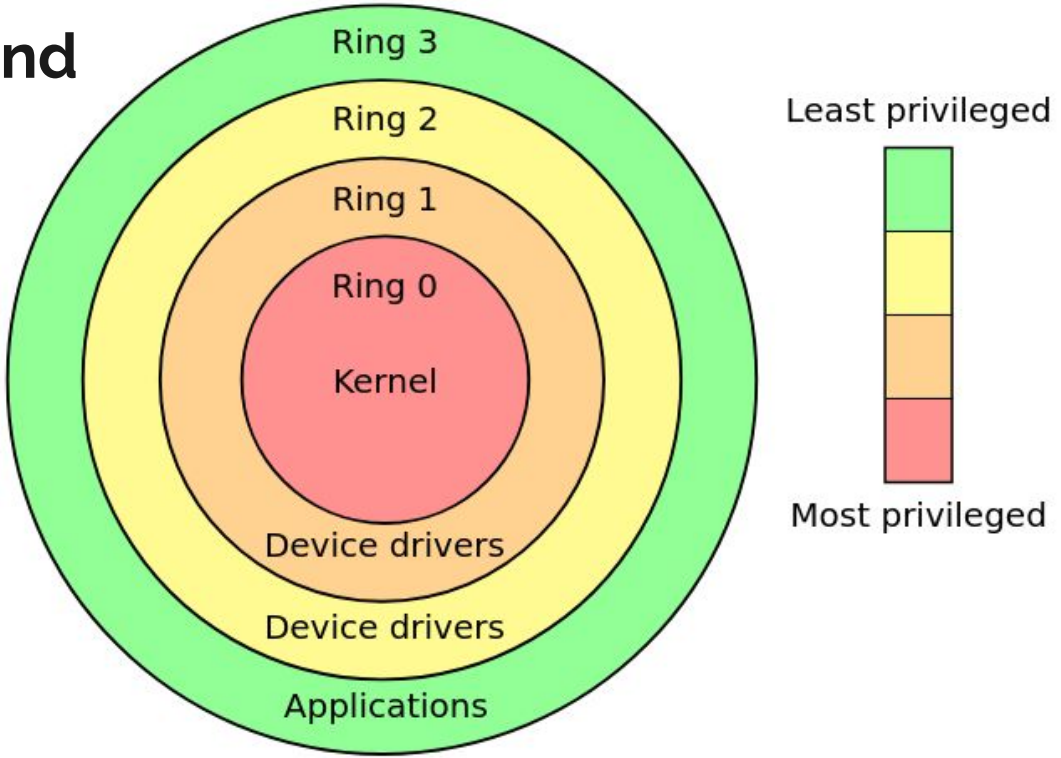
Modo Dual de Operaciones



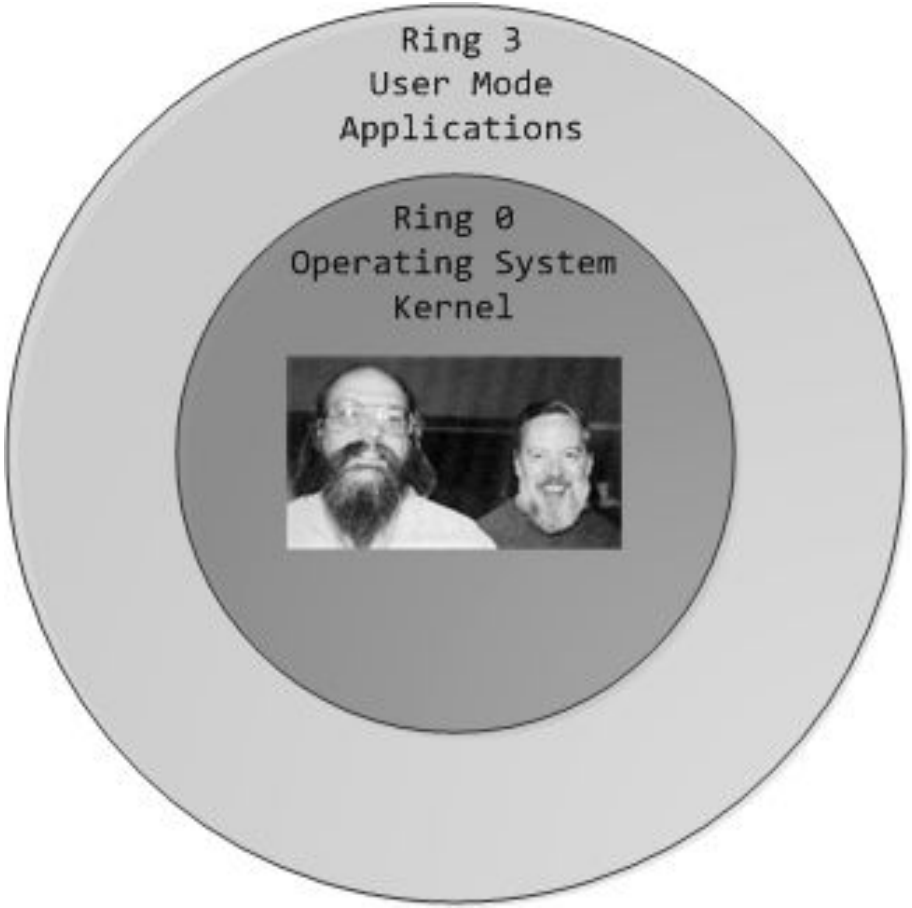
Modo Dual de Operaciones



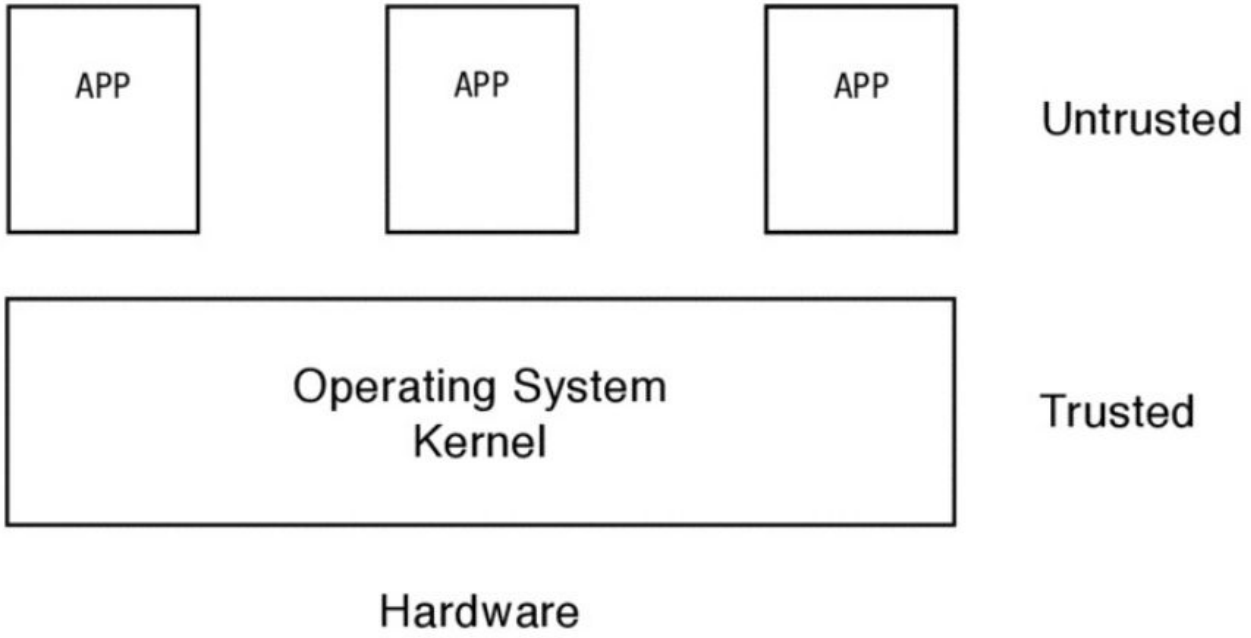
Kernerl Land y User Land



Kernerl Land y User Land



Kernerl Land y User Land



IOPL

I/O Privilege level

Ocupa los bits 12 y 13 en el registro FLAGS.

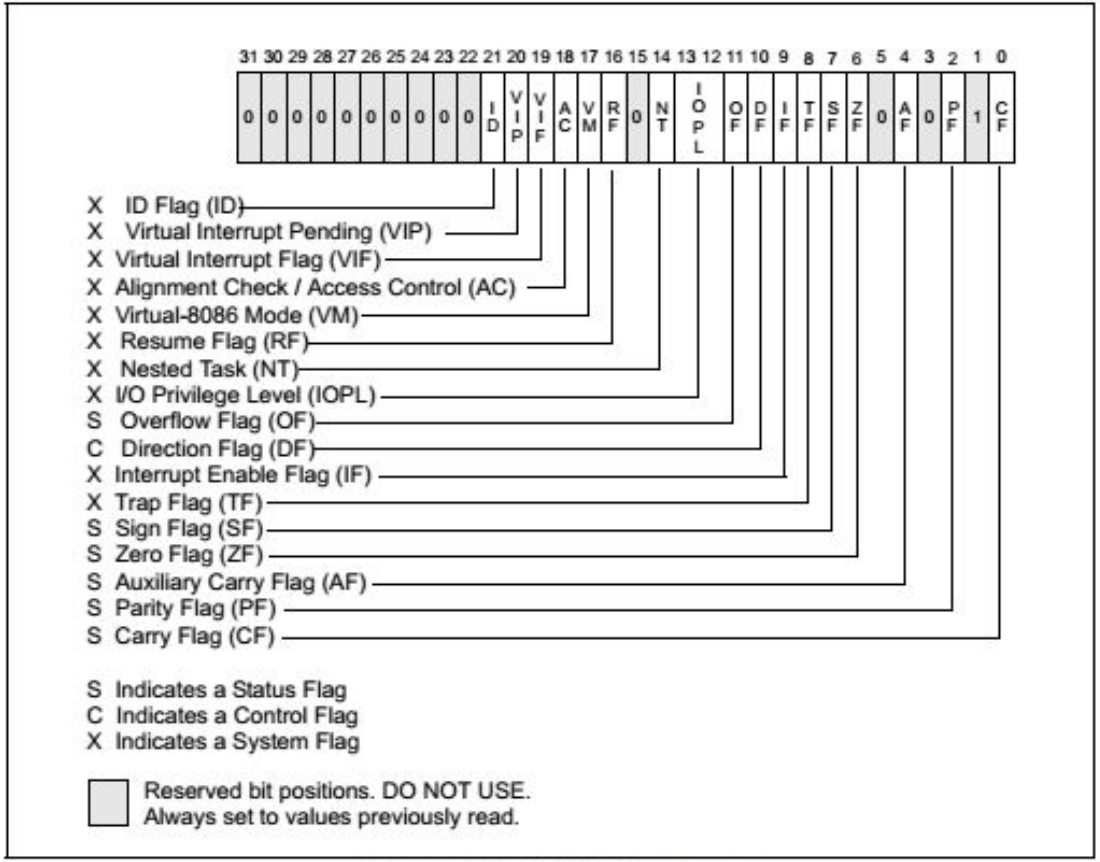


Figure 3-8. EFLAGS Register

Cómo protege el kernel de un SO las aplicaciones y los usuarios

Instrucciones Privilegiadas

Protección de Memoria

Timer Interrupts

Instrucciones Privilegiadas

Distintos set de instrucciones

Instrucciones Privilegiadas

- I/O instructions and Halt instructions
- Turn off all Interrupts
- Set the Timer
- Context Switching
- Clear the Memory or Remove a process from the Memory
- Modify entries in the Device-status table

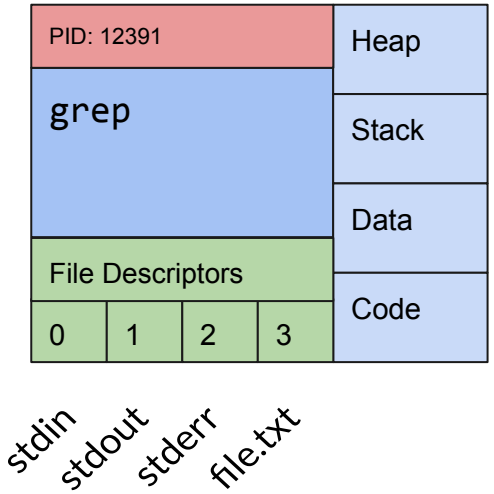
Instrucciones No Privilegiadas

- Reading the status of Processor
- Reading the System Time
- Generate any Trap Instruction
- Sending the final printout of Printer

Protección de Memoria

Dirección Virtual vs Dirección Física

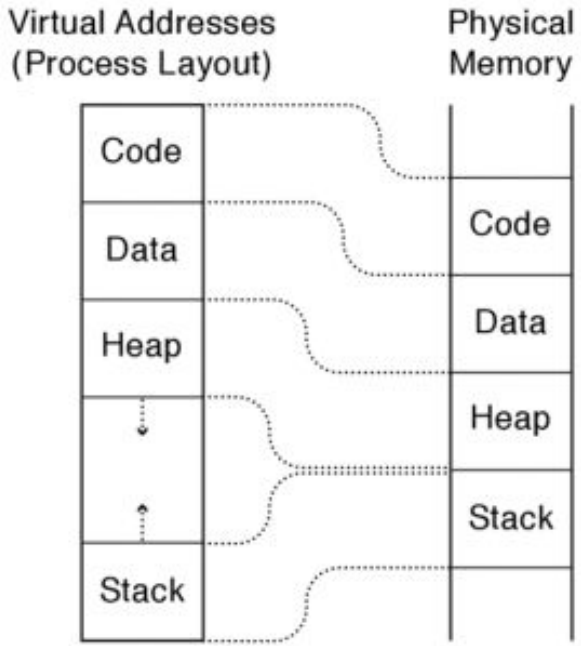
Espacio de direcciones



Protección de Memoria

Dirección Virtual vs Dirección Física

Espacio de direcciones



Timer Interrupts (Interrupciones por temporizador)

¿ Cómo hace el Kernel para volver a tener que un proceso tiene?

Casi todos los procesadores contiene un dispositivo llamado Hardware Timer, cada timer interrumpe a un determinado procesador mediante una interrupción por hardware.

Cuando una interrupción por tiempo se dispara se transfiere el control desde el processo de usuario al Kernel

Timer Interrupts (Interrupciones por temporizador)

Veamoslo !

sudo cat /proc/timer_list

Modos de Transferencia

Modo de Transferencia

Una vez que el kernel pone a un proceso de usuario en un entorno aislado (sandbox), la próxima pregunta es: **¿cómo se transiciona entre un modo y otro**?

Modo de Transferencia

Una vez que el kernel pone a un proceso de usuario en un entorno aislado (sandbox), la próxima pregunta es: **¿cómo se transiciona entre un modo y otro**?

Modo de Transferencia

Kernel Mode to User Mode

User Mode to Kernel Mode

User Mode to Kernel Mode

Interrupciones

Excepciones del Procesador

System Calls

Interrupciones

Una interrupción es una señal asincrónica hacia el procesador avisando que algún evento externo requiere su atención

Interrupciones

```
bob@susel:~> sudo cat /proc/interrupts
           CPU0
  0:         78   IO-APIC-edge      timer
  1:       9012   IO-APIC-edge      i8042
  3:          1   IO-APIC-edge
  4:       6609   IO-APIC-edge
  6:          7   IO-APIC-edge      floppy
  7:          0   IO-APIC-edge      parport0
  8:          1   IO-APIC-edge      rtc0
  9:          0   IO-APIC-fasteoi   acpi
 12:     101705   IO-APIC-edge      i8042
 14:       1374   IO-APIC-edge      ata_piix
 15:      28050   IO-APIC-edge      ata_piix
 16:        893   IO-APIC-fasteoi   Ensoniq AudioPCI
 17:      77088   IO-APIC-fasteoi   ioc0, ehci_hcd:usb1
 18:        112   IO-APIC-fasteoi   uhci_hcd:usb2
 19:       5167   IO-APIC-fasteoi   eth0
 40:          0   PCI-MSI-edge      PCIe PME, pciehp
 41:          0   PCI-MSI-edge      PCIe PME, pciehp
 42:          0   PCI-MSI-edge      PCIe PME, pciehp
 43:          0   PCI-MSI-edge      PCIe PME, pciehp
 44:          0   PCI-MSI-edge      PCIe PME, pciehp
 45:          0   PCI-MSI-edge      PCIe PME, pciehp
 46:          0   PCI-MSI-edge      PCIe PME, pciehp
```

Sudo cat /proc/interrupts

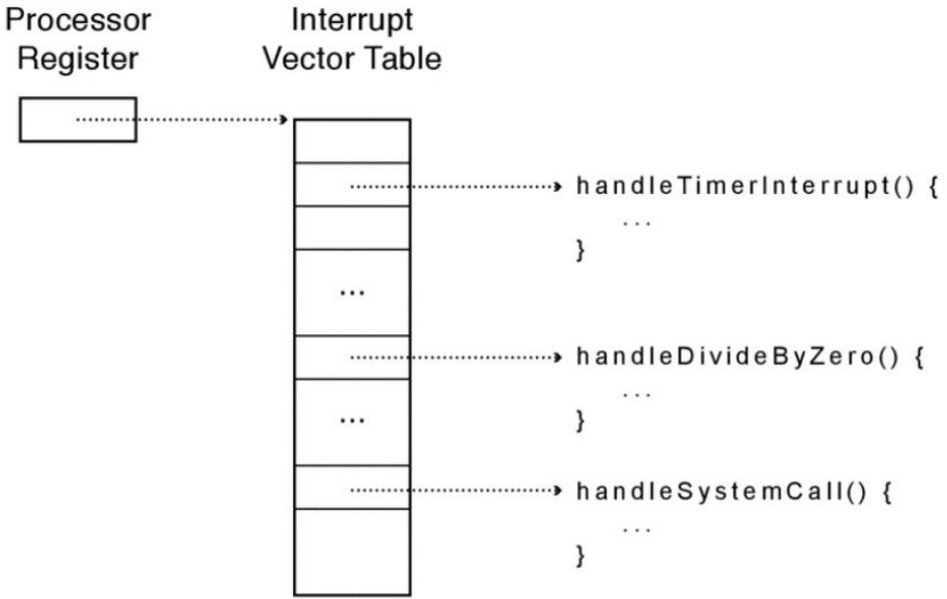
Orden de importancia

- Errores de la Máquina
- Timers
- Discos
- Network devices
- Terminales
- Interrupciones de Software

Interrupciones

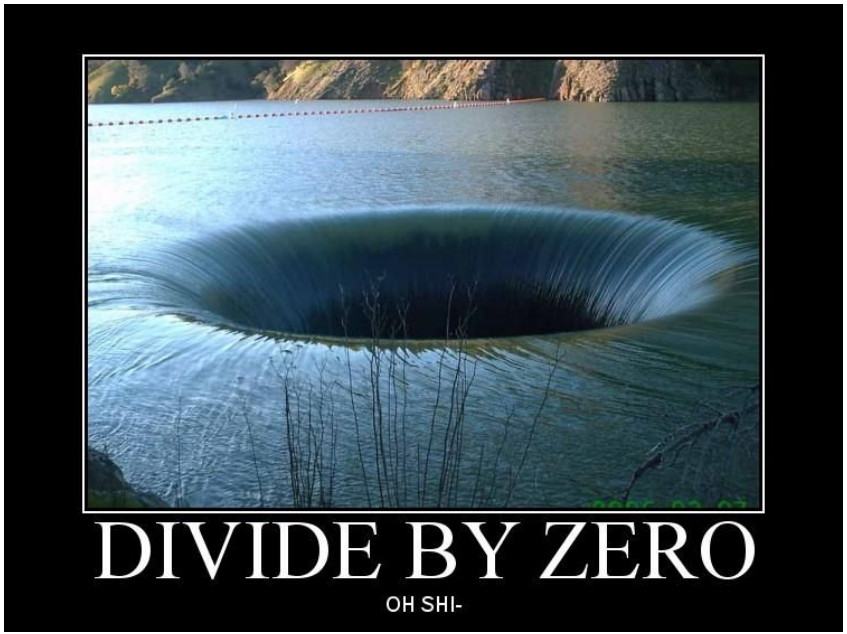
Orden de importancia

1. Errores de la Máquina
2. Timers
3. Discos Network devices
4. Terminales
5. Interrupciones de Software



Excepciones del Procesador

Una excepción es un evento de hardware causado por una aplicación de usuario que causa la transferencia del control al Kernel.



Excepciones del Procesador

Una excepción es un evento de hardware causado por una aplicación de usuario que causa la transferencia del control al Kernel.

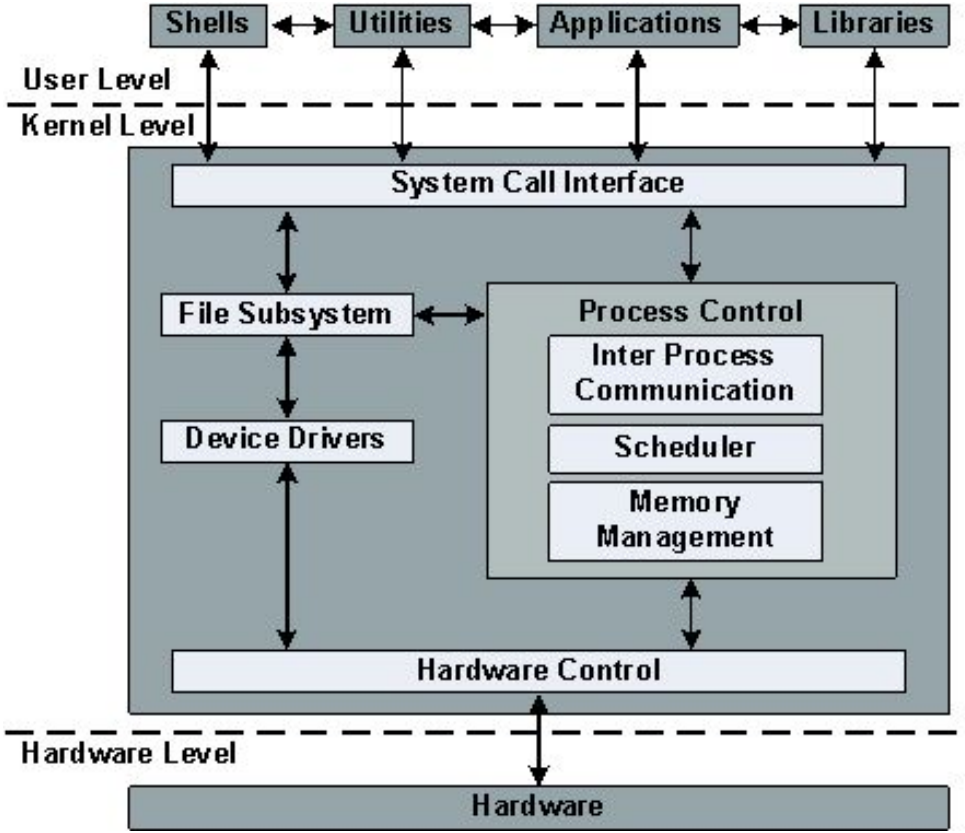
```
set_trap_gate(0,&divide_error);
set_intr_gate(1,&debug);
set_intr_gate(2,&nmi);
set_system_intr_gate(3, &int3); /* int3/4 can
be called from all */
set_system_gate(4,&overflow);
set_trap_gate(5,&bounds);
set_trap_gate(6,&invalid_op);
set_trap_gate(7,&device_not_available);
set_task_gate(8,GDT_ENTRY_DOUBLEFAULT_TSS);
set_trap_gate(9,&coprocessor_segment_overrun)
;
set_trap_gate(10,&invalid_TSS);
set_trap_gate(11,&segment_not_present);
set_trap_gate(12,&stack_segment);
set_trap_gate(13,&general_protection);
set_intr_gate(14,&page_fault);
set_trap_gate(15,&spurious_interrupt_bug);
set_trap_gate(16,&coprocessor_error);
set_trap_gate(17,&alignment_check);
```

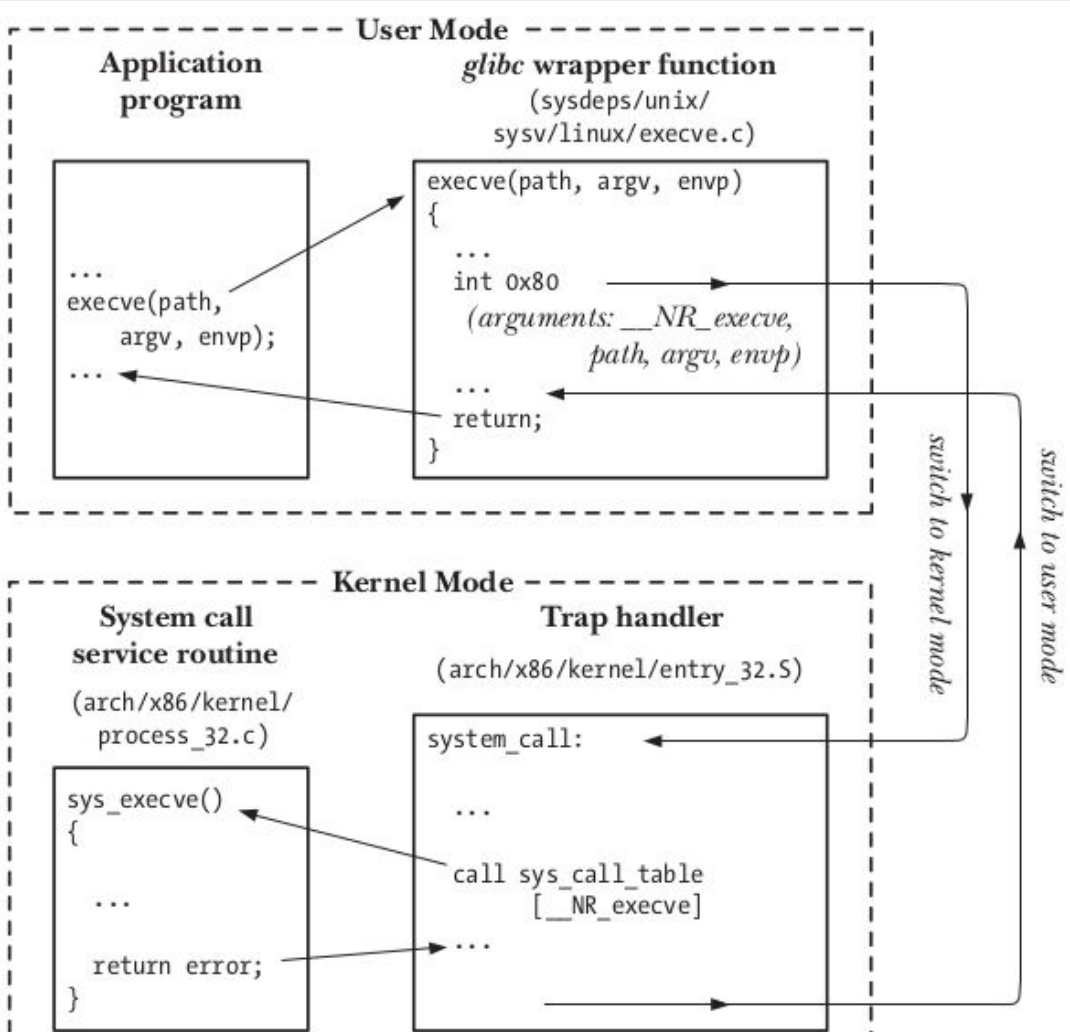
System Calls

Un proceso de usuario puede hacer que la transición de modo sea hecha voluntariamente.

Una System Call es algún procedimiento provisto por el kernel que puede ser llamada desde el user level.

System Calls




User Mode
Application program
glibc wrapper function
(sysdeps/unix/sysv/linux/execve.c)
...
execve(path, argv, envp);
...
execve(path, argv, envp)
{
...
int 0x80
(arguments: __NR_execve, path, argv, envp)
...
return;
}
switch to kernel mode
switch to user mode
Kernel Mode
System call service routine
(arch/x86/kernel/process_32.c)
Trap handler
(arch/x86/kernel/entry_32.S)
sys_execve()
{
...
return error;
}
system_call:
...
call sys_call_table [__NR_execve]
...

System Calls

Desde el punto de vista de un programa llamar a una system call es mas o menos como invocar a una función de C. Por supuesto, detrás de bambalinas muchas cosas suceden:

El programa realiza un llamado a una system call mediante la invocación de una función wrapper (envoltorio) en la biblioteca de C.

Dicha función wrapper tiene que proporcionar todos los argumentos al system call trap_handling. Estos argumentos son pasados al wraper por el stack, peeero el kernel los espera en determinados registros. La función wraper copia estos valores a los registros.

Dado que todas las system calls son accedidas de la misma forma, el kernel tiene que saber identificarlas de alguna forma. Para poder hacer esto, la función wrapper copia el número de la system call a un determinado registro de la CPU (%eax).

La función wrapper ejecuta una instrucción de código maquina llamada trap machine instruction (int 0x80), esta causa que el procesador pase de user mode a kernel mode y ejecute el código apuntado por la dirección 0x80 (128) del vector de traps del sistema.

System Calls

En respuesta al trap de la posición 128, el kernel invoca su propia función llamada syste_call() (arch/i386/entry.s) para manejar esa trap. Este manejador:

graba el valor de los registros en el stack del kernel.

verifica la validez del número de system call.

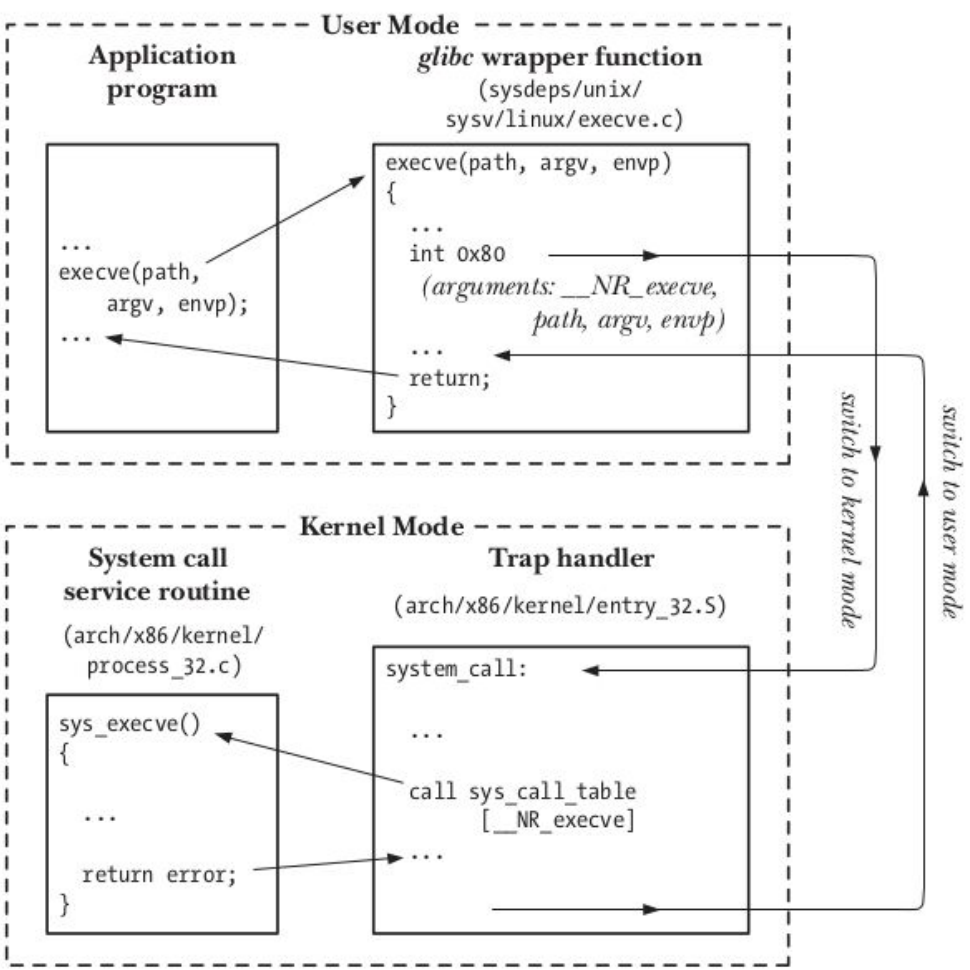
invoca el servicio correspondiente a la system call llamada a través del vector de system calls, el servicio realiza su tarea y finalmente le devuelve un resultado de estado a la rutina system_call().

se restauran los registros almacenado en el stack del kernel y se agrega el valor de retorno en el stack.

se devuelve el control al wraper y simultáneamente se pasa a user mode.

Si el valor de retorno del la rutina de servicio de la system call da error, la función wrapper setea el valor en errno.

System Calls



System Calls

Strace

Ltrace

strace ls testdir/

strace -o trace.log ls testdir/

Kernel Mode to User Mode

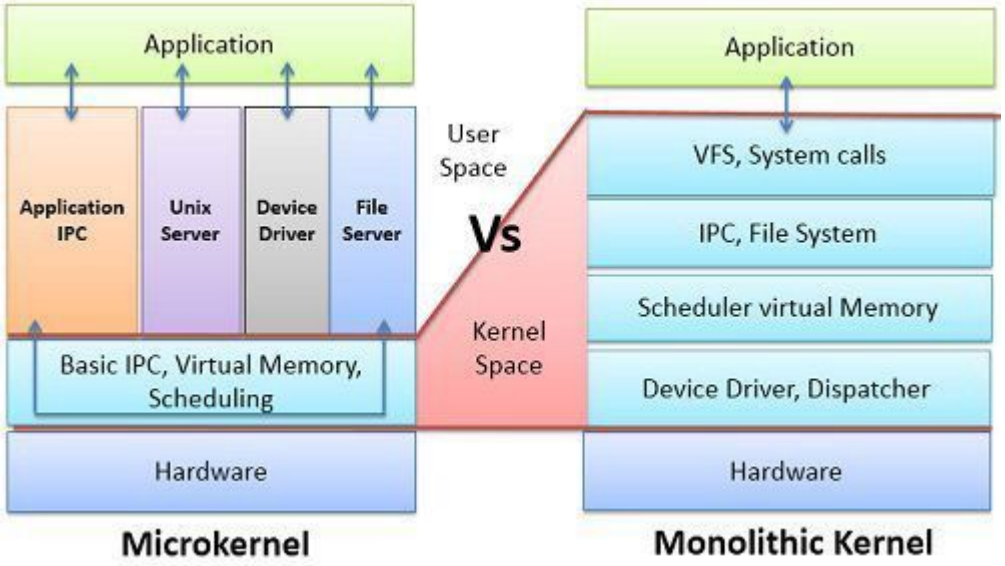
Nuevo Proceso

Continuar luego de una interrupción, excepción o una sys call

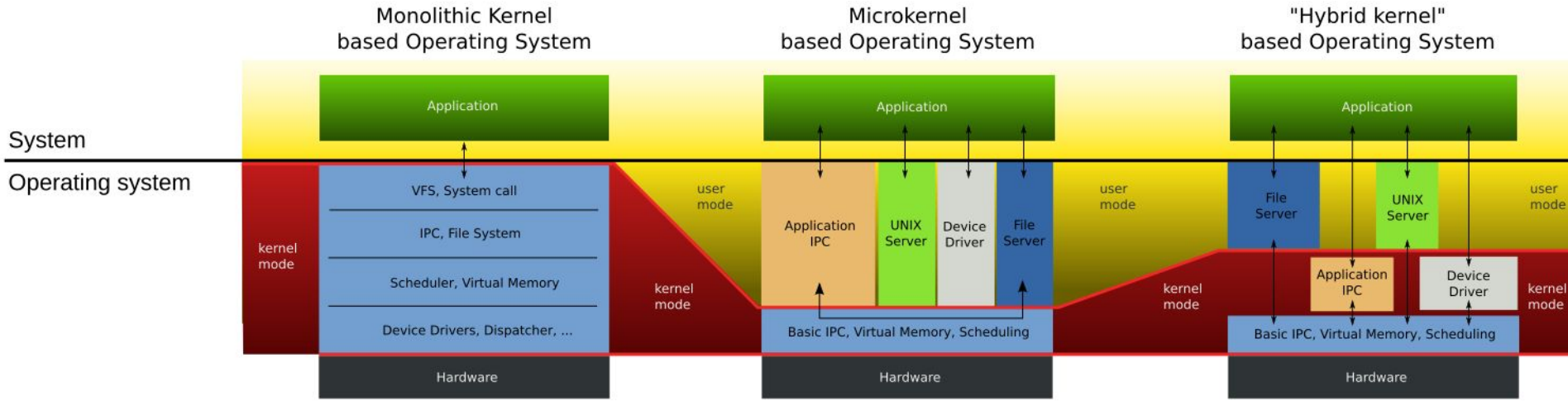
Cambiar entre diferentes procesos

Dahlin Cap 2 hasta 2.4!

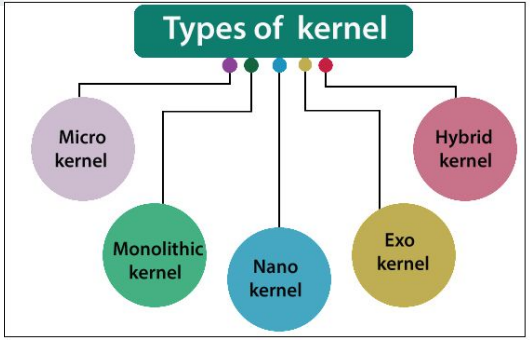
Tipos de Kernel



Tipos de Kernel



Tipos de Kernel



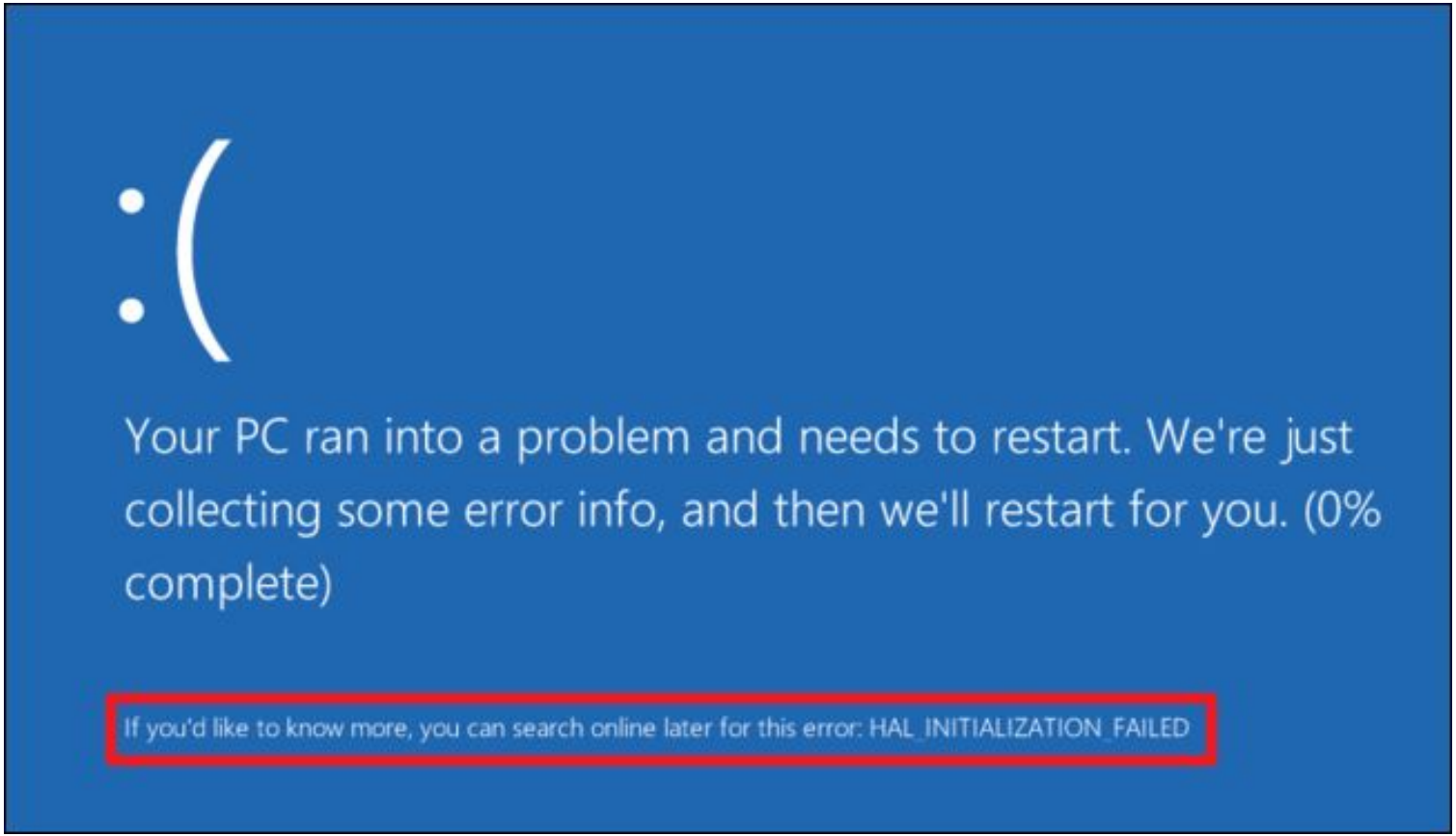
Monolithic Kernel: - In a monolithic kernel, the kernel and operating system, both run in the same memory, and it is mainly used where security is not a major concern. The result of the monolithic kernel is fastly accessible. But in some situations, like if a device driver has a bug, then there may be chances of a whole system crash.

Microkernel: - A Microkernel is the derived version of the monolithic kernel. In microkernel, the kernel itself can do different jobs, and there is no requirement of an additional GUI.

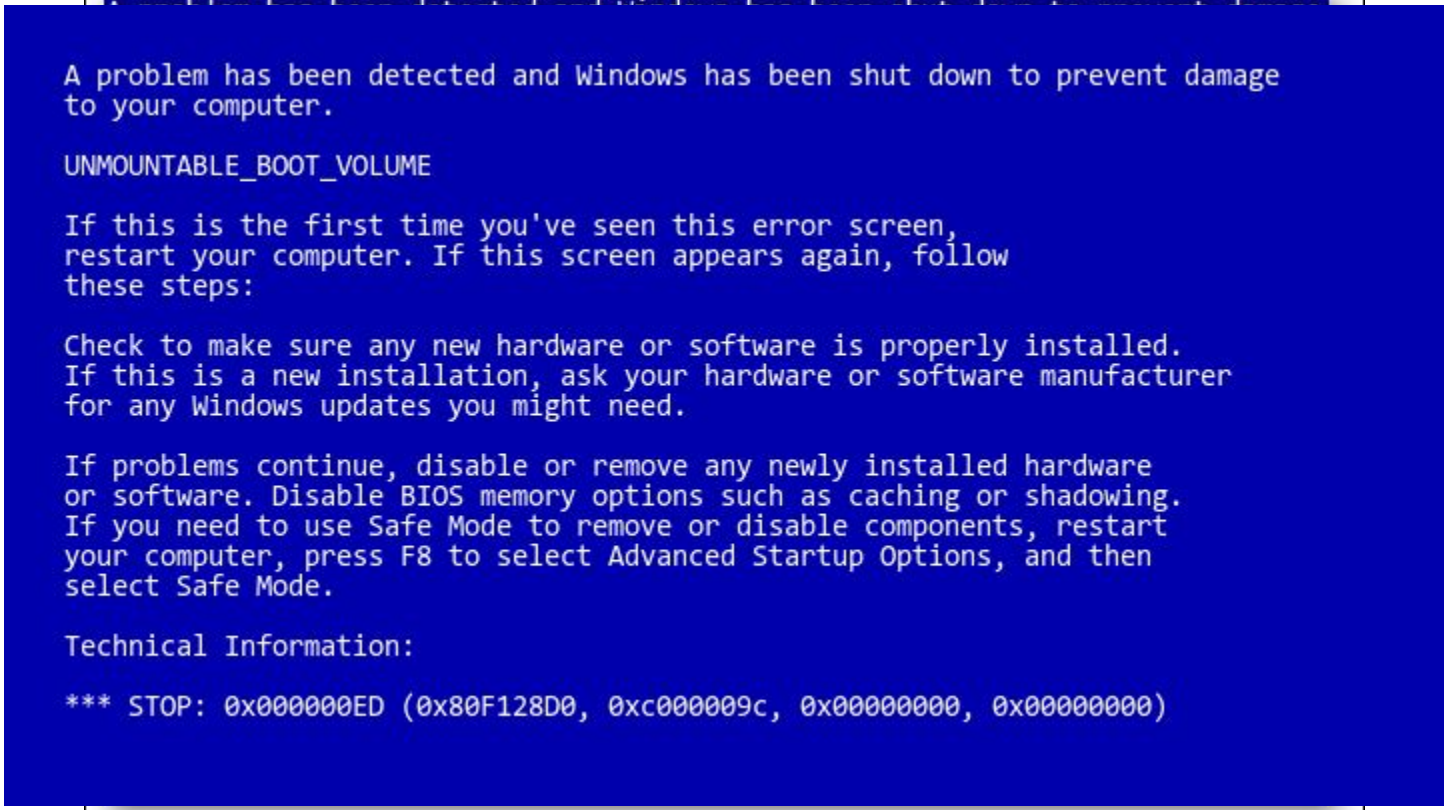
Nano kernel: - Nano kernel is the small type of kernel which is responsible for hardware abstraction, but without system services. Nano kernel is used in those cases where most of the functions are set up outside.

Exo kernel: - Exo kernel is responsible for resource handling and process protection. It is used where you are testing out an inhouse project, and in up-gradation to an efficient kernel type.

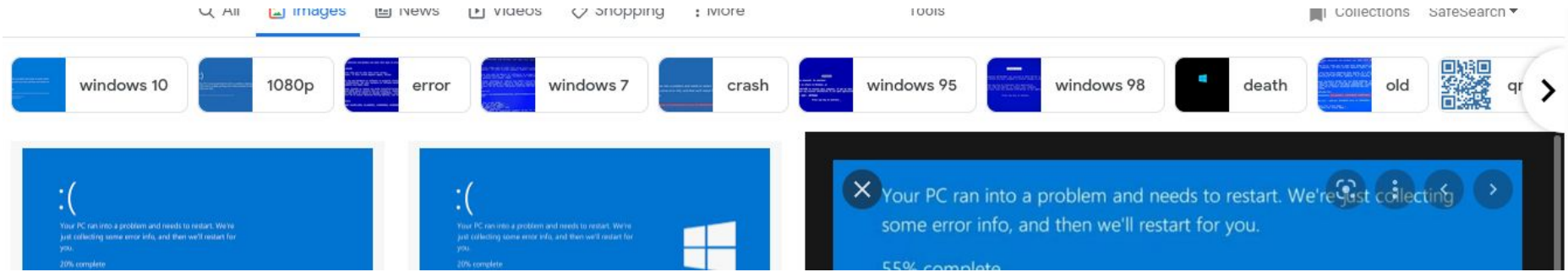
Hybrid kernel: - Hybrid kernel is a mixture of microkernel and monolithic kernel. The Hybrid kernel is mostly used in Windows, Apple's macOS. Hybrid kernel moves out the driver and keeps the services of a system inside the kernel.



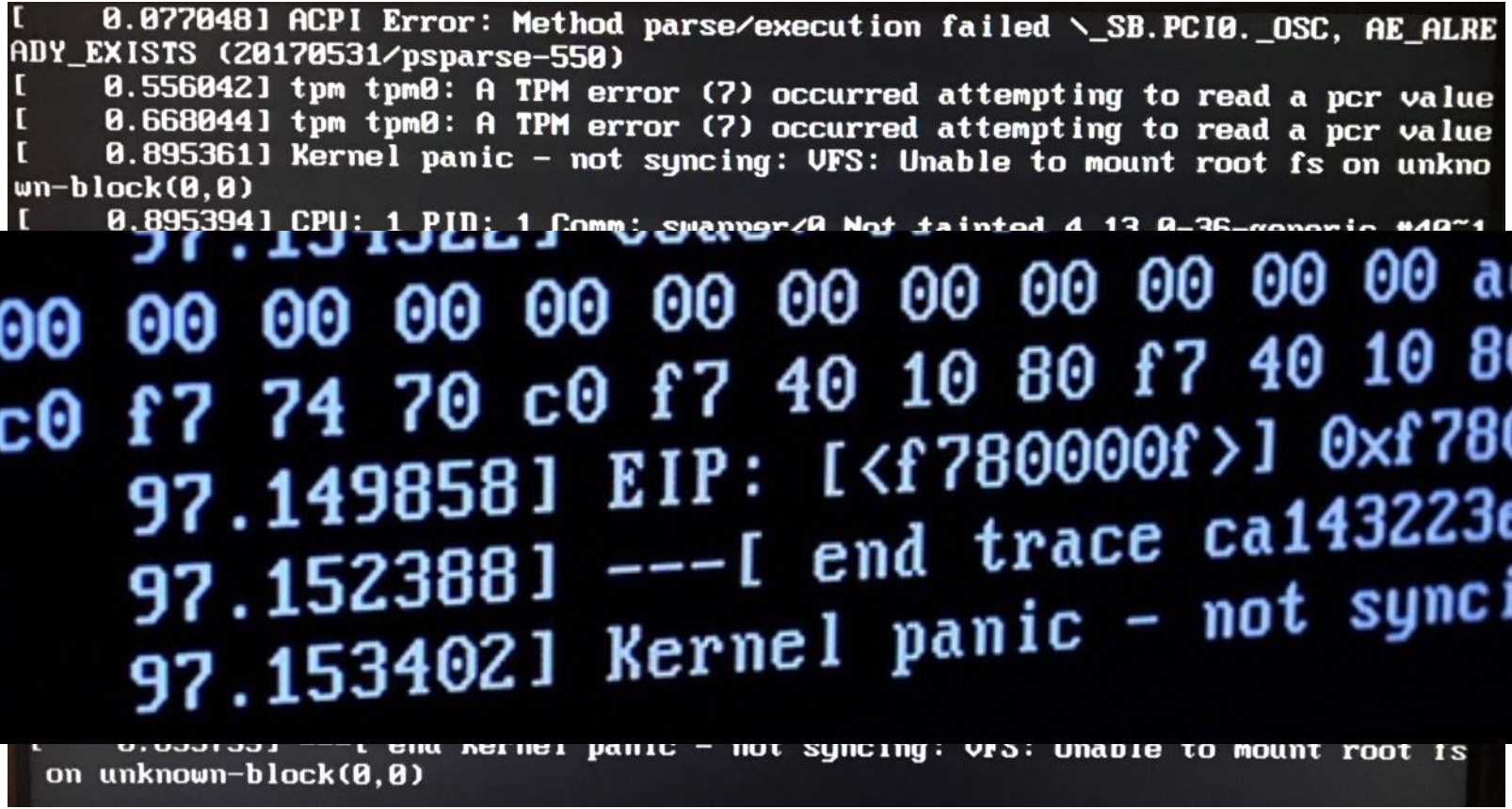
Y si el Kernel Falla ...



Y si el Kernel Falla ...

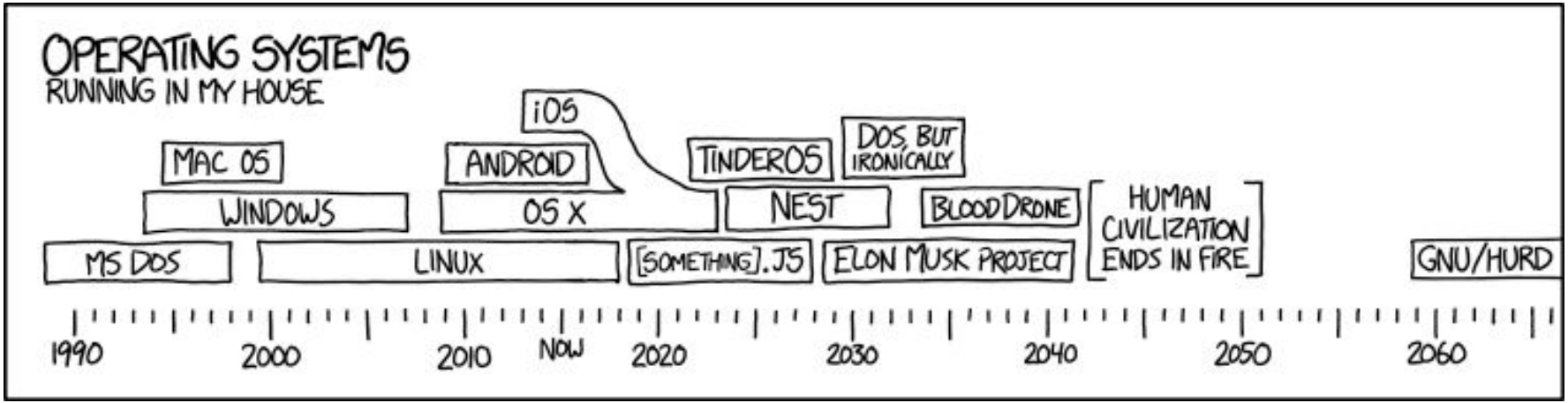


Y si el Kernel Falla ...



Y si el Kernel Falla ...

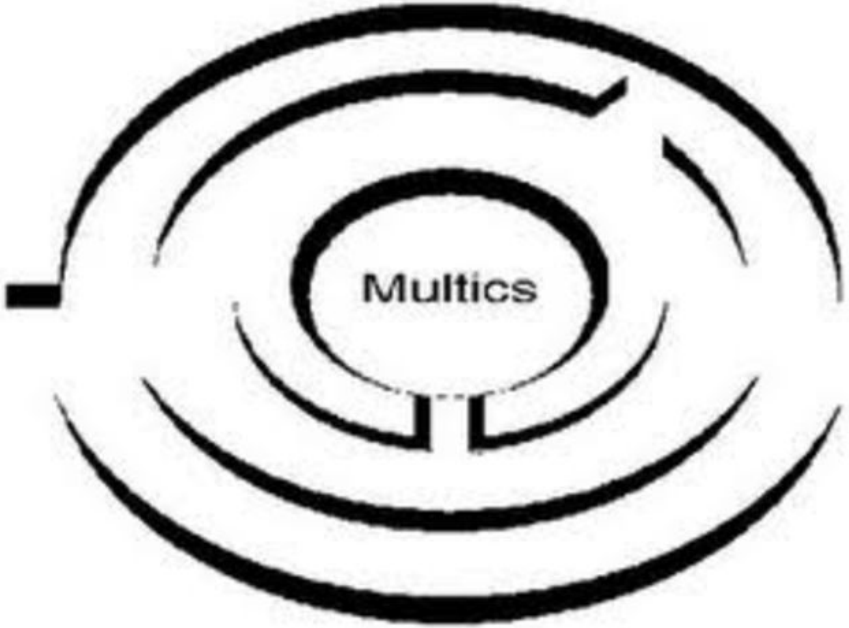
Unix


OPERATING SYSTEMS
RUNNING IN MY HOUSE
MAC OS
WINDOWS
MS DOS
LINUX
iOS
ANDROID
OS X
[SOMETHING].JS
TINDEROS
NEST
ELON MUSK PROJECT
DOS, BUT IRONICALLY
BLOOD DRONE
HUMAN CIVILIZATION ENDS IN FIRE
GNU/HURD
1990
2000
2010
NOW
2020
2030
2040
2050
2060

Multics

Multiplexed Information and Computing Service

1964



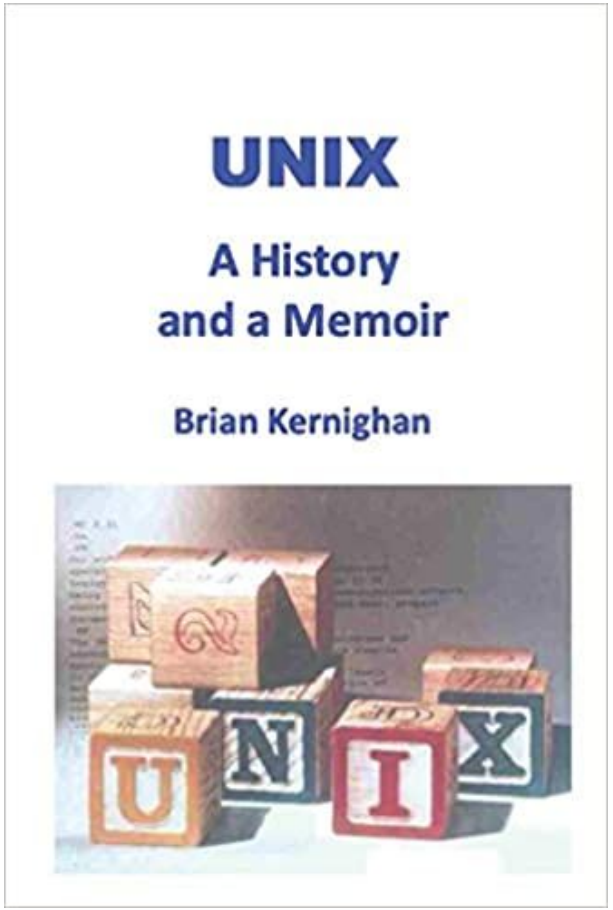
Unix

Hacia 1969 Ken Thompson decidió utilizar una PDP-7 que se encontraba en desuso para implementar una versión mínima del sistema operativo llamado MULTICS. No fue hasta el año 1971 que la primera versión de UNIX fue lanzada como proyecto en los laboratorios Bell, de la mano de **Dennis Ritchie y Ken Thompson**

Unix

Unix

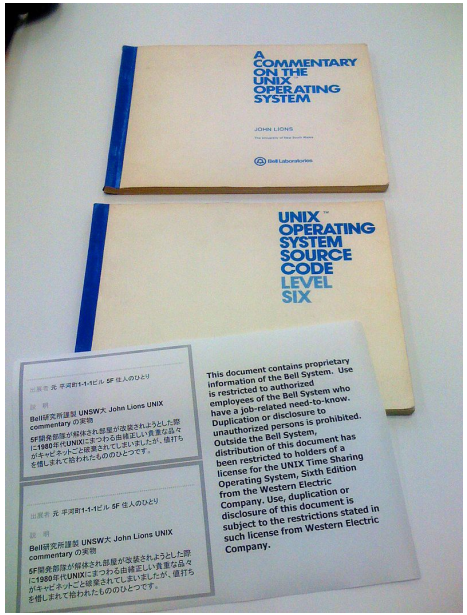
Unix

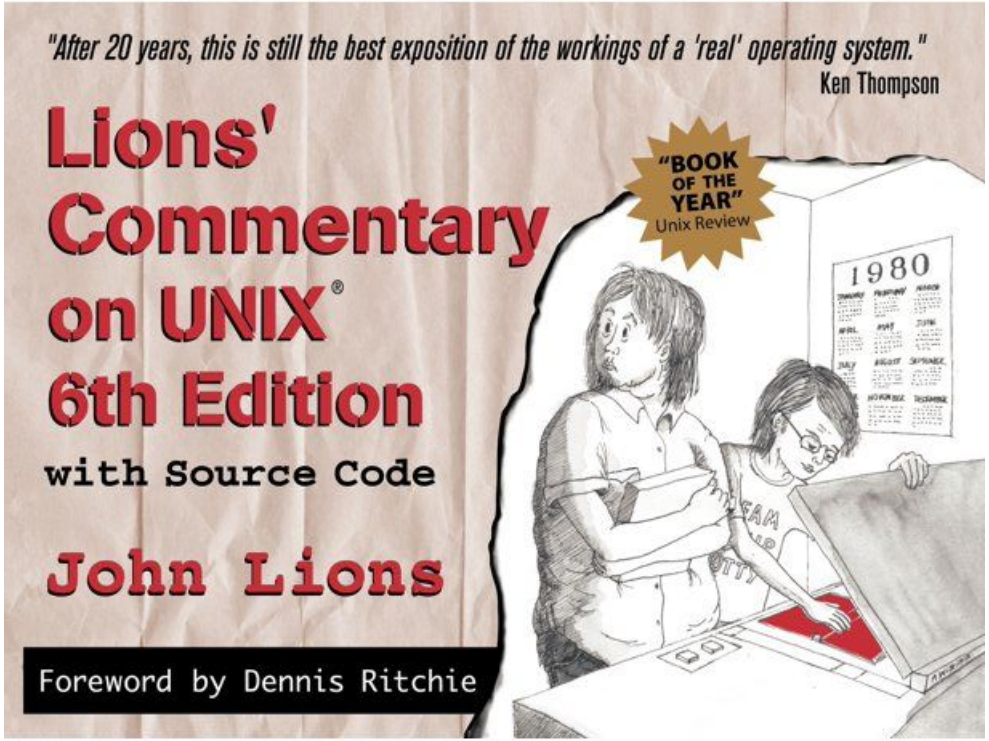


Unix: Las Versiones Academicas

UNIX-v1 (Nov. 3, 1971): En su primera edición el sistema operativo ocupaba 16 Kb, 8Kb para programas de usuario, un disco de 512 Kb y un límite de 64 kb por archivo. Escrito en assembler.
UNIX-v2 (Jun. 12, 1972)
UNIX-v3 (Feb. 1973): Versión que tenía un compilador C.
UNIX-v4 (Nov. 1973): Primera versión escrita totalmente en C.
UNIX-v5 (Jun. 1974)
UNIX-v6 (May 1975): Tal vez la versión más conocida por su licencia gratuita para investigación y enseñanza.
UNIX-v7 (Jan. 1979): Versión padre de todos los sistemas operativos basados en UNIX (excepto Coherent, Minix, and Linux).

Unix: V6





https://warsus.github.io/lions-/

Linux

Linus Benedict Torvalds

Hello everybody out there using minix -

I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones. This has been brewing since april, and is starting to get ready. I'd like any feedback on things people like/dislike in minix, as my OS resembles it somewhat (same physical layout of the file-system (due to practical reasons) among other things).

I've currently ported bash(1.08) and gcc(1.40), and things seem to work. This implies that I'll get something practical within a few months, and I'd like to know what features most people would want. Any suggestions are welcome, but I won't promise I'll implement them :-)

Linus (torv...@kruuna.helsinki.fi)

PS. Yes - it's free of any minix code, and it has a multi-threaded fs. It is NOT protable (uses 386 task switching etc), and it probably never will support anything other than AT-harddisks, as that's all I have :-(.

Iniciar al Sistema Operativo y el Kernel

Inicio de S.O.

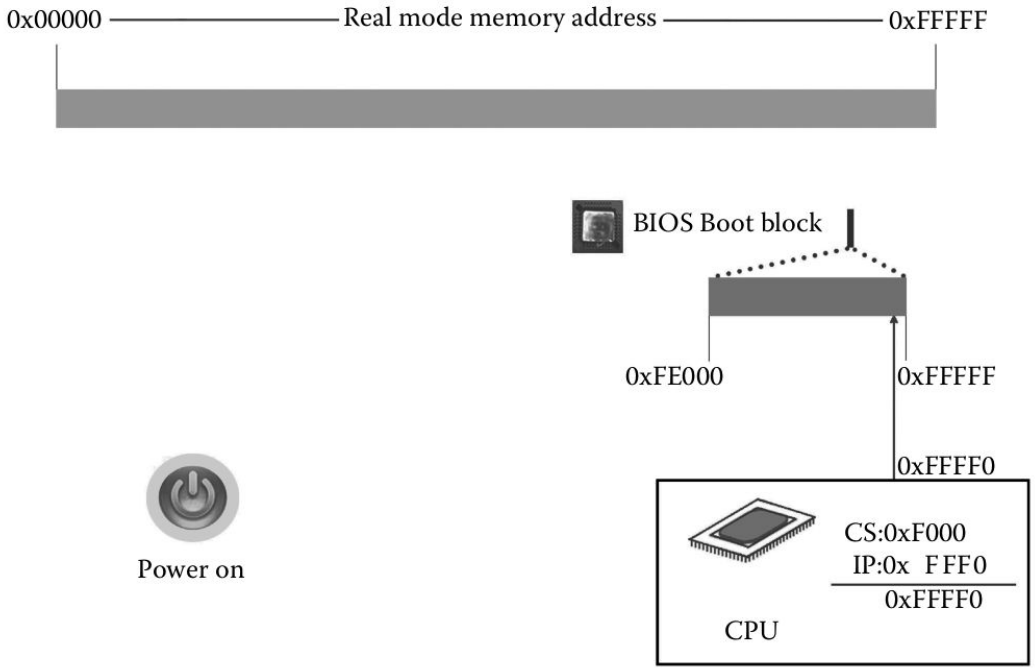
1. Booteo
2. Carga del Kernel
3. Inicio de las Aplicaciones de Usuarios

Booteo

Este proceso es denominado bootstrap, y generalmente depende del hardware de la computadora. En el se realizan los chequeos de hardware y se carga el bootloader, que es el programa encargado de cargar el Kernel del Sistema Operativo. Este proceso consta de 4 partes.

Booteo 1

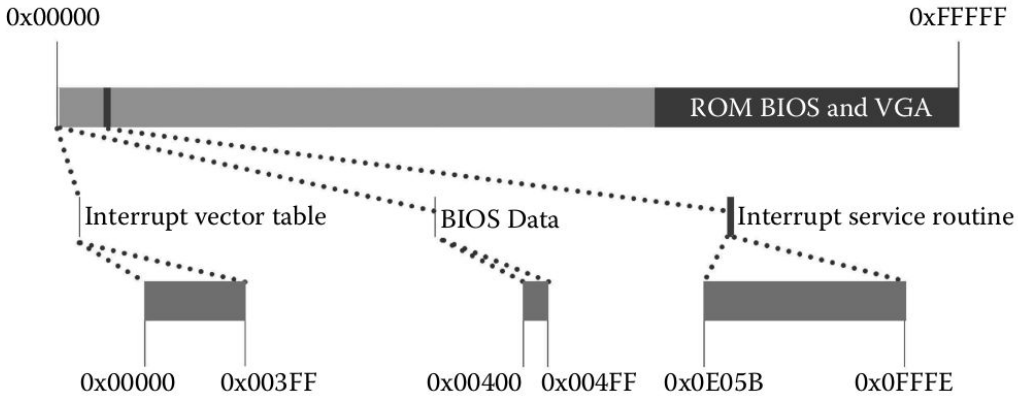
Cargar el **BIOS (Basic Input/Outp System)**: para eso apenas se encien la PC, se carga CS con 0xFFFF y IP c 0x0000; por ende, la dirección CS:IP es 0xFFFF0, justamente dirección de memoria de la BIOS.



Booteo 2

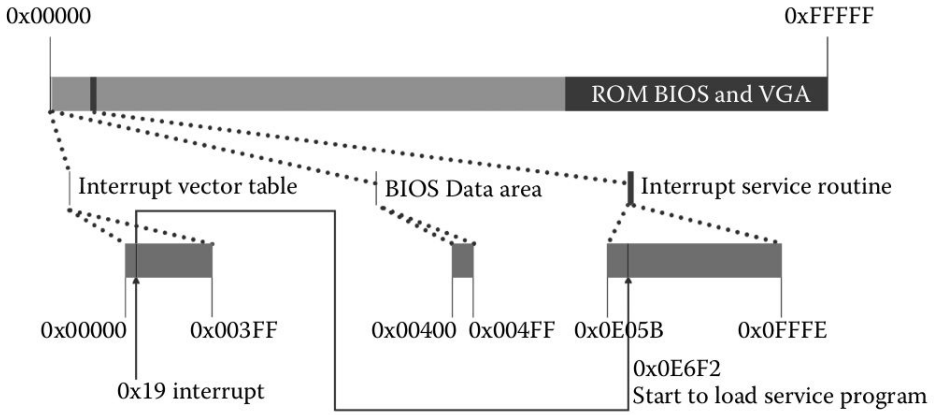
Crear la **Interrupt Vector Table** y cargar las rutinas de manejo de interrupciones en **Modo Real**: el BIOS pone la tabla de interrupciones en el inicio de la memoria 1 KB (0x00000–0x003FF), son 256 entradas de 4 bytes.

El área de datos del BIOS de unos 256 B (0x00400–0x004FF), y el servicio de atención de interrupciones (8 KB), 56 KB, después de la dirección 0x0E05B.



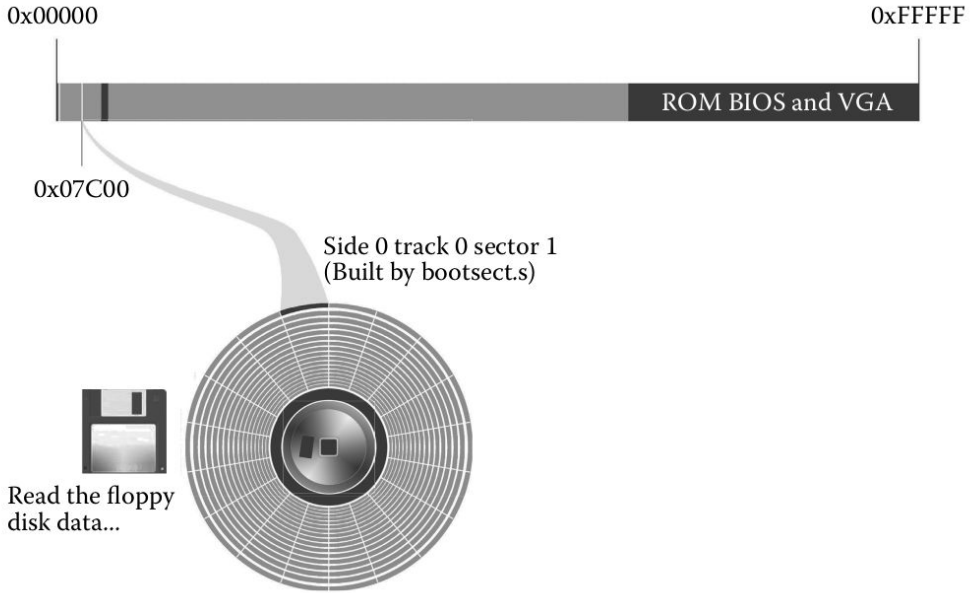
Booteo 3

La BIOS genera una interrupción 19 (INT 19) de la tabla de interrupciones la cual hace apuntar a CS:IP a 0x0E6F2.



Booteo 4

Lo cual hace ejecutar el servicio de interrupciones, el handler de dicha interrupción, que es leer el primer sector de 512 bytes del disco a memoria, y ahí termina.



Fase de Inicio del kernel

La función de arranque para el kernel (también llamado intercambiador o proceso 0) establece la gestión de memoria (tablas de paginación y paginación de memoria), detecta el tipo de CPU y cualquier funcionalidad adicional como capacidades de punto flotante, y después cambia a las funcionalidades del kernel para arquitectura no específicas de Linux, a través de una llamada a la función start_kernel().

Fase de Inicio del kernel

Por lo tanto, el núcleo **inicializa los dispositivos**, **monta el sistema de archivos raíz** especificado por el gestor de arranque como de sólo lectura, **y se ejecuta Init (/sbin/init**), que es designado como el primer proceso ejecutado por el sistema (PID=1). También puede ejecutar opcionalmente initrd para permitir instalar y cargar dispositivos relacionados (disco RAM o similar), para ser manipulados antes de que el sistema de archivos raíz está montado.

En este punto, con las interrupciones habilitadas, el programador puede tomar el control de la gestión general del sistema, para proporcionar multitarea preventiva, e iniciar el proceso para continuar con la carga del entorno de usuario en el espacio de usuario.

El Proceso de Inicio

El trabajo de **Init** es "**conseguir que todo funcione como debe ser**" una vez que el kernel está totalmente en funcionamiento. En esencia, establece y opera todo el espacio de usuario. Esto incluye:

1. la comprobación y montaje de sistemas de archivos
2. la puesta en marcha los servicios de usuario necesarios y, en última instancia, cambiar al entorno de usuario cuando el inicio del sistema se ha completado.