Introducción a los Sistemas Operativos

Introducción – I

I.S.O.

- ☑ Versión: Agosto 2019
- ☑ Palabras Claves: Sistemas Operativos, Harware, Interrupciones, Registros

Los temas vistos en estas diapositivas han sido mayormente extraídos del libro de William Stallings (Sistemas Operativos: Aspectos internos y principios de diseño)

¿Qué es un Sistema Operativo?

¿SO?

Sistema Operativo

- Es software:
  - Necesita procesador y memoria para ejecutarse

- Dos perspectivas
  - De arriba hacia abajo
  - De abajo hacia arriba

Perspectiva de arriba hacia abajo

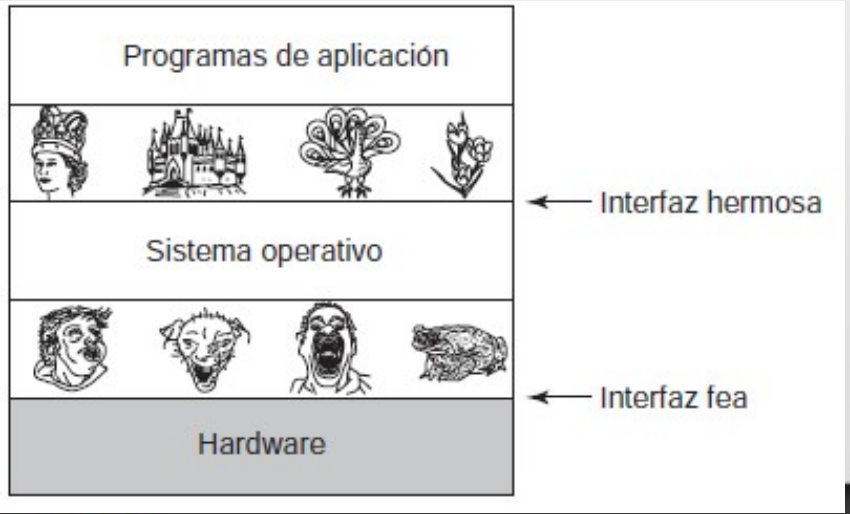
- Abstracción con respecto a la arquitectura
  - Arquitectura: conjunto de instrucciones, organización de memoria, E/S, estructura de bus)
- El SO “oculta” el HW y presenta a los programas abstracciones más simples de manejar.
- Los programas de aplicación son los “clientes” del SO.
- Comparación: uso de escritorio y uso de comandos de texto
- Comodidad, “amigabilidad” (friendliness)

Perspectiva de abajo hacia arriba

- ☑ Visión del SO como un administrador de recursos
- ☑ Administra los recursos de HW de uno o más **procesos**
- ☑ Provee un conjunto de servicios a los usuarios del sistema
- ☑ Maneja la memoria secundaria y dispositivos de I/O.
- ☑ Ejecución simultánea de procesos
- ☑ Multiplexación en tiempo (CPU) y en espacio (memoria)

Sistema Operativo

- ☑ Gestiona el HW
- ☑ Controla la ejecución de los **procesos**
- ☑ Interfaz entre aplicaciones y HW
- ☑ Actúa como intermediario entre un usuario de una computadora y el HW de la misma

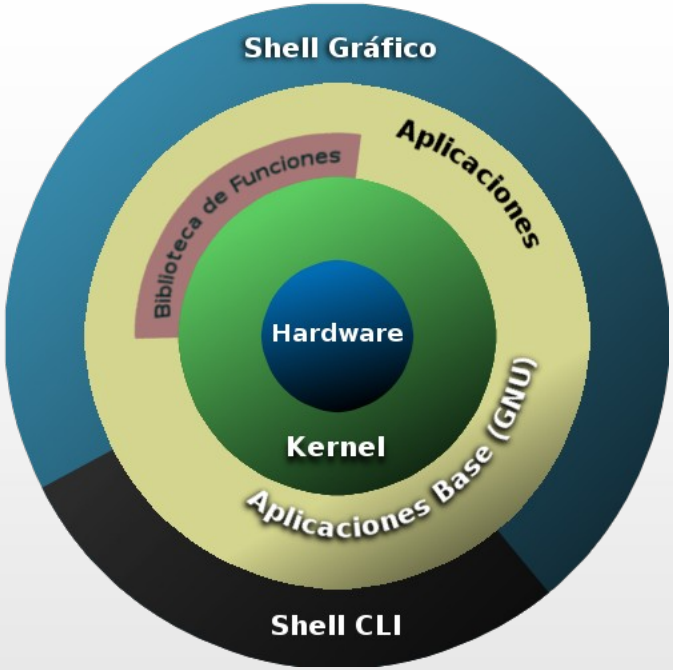


Objetivos de los S.O.

- Comodidad
  - Hacer mas fácil el uso del hardware (PC, servidor, switch, router, controlador específico)
- Eficiencia
  - Hacer un uso más eficiente de los recursos del sistema
- Evolución
  - Permitir la introducción de nuevas funciones al sistema sin interferir con funciones anteriores

Componentes de un SO

- ☑ Kernel
- ☑ Shell
  - GUI / CUI o CLI



- ☑ Herramientas
  - Editores, Compiladores, Librerías, etc.

Kernel (Núcleo)

- “Porción de código”
  - Que se encuentra en memoria principal
  - Que se encarga de la administración de los recursos.
- Implementa servicios esenciales:
  - Manejo de memoria
  - Manejo de la CPU
  - Administración de procesos
  - Comunicación y Concurrencia
  - Gestión de la E/S

Servicios de un SO

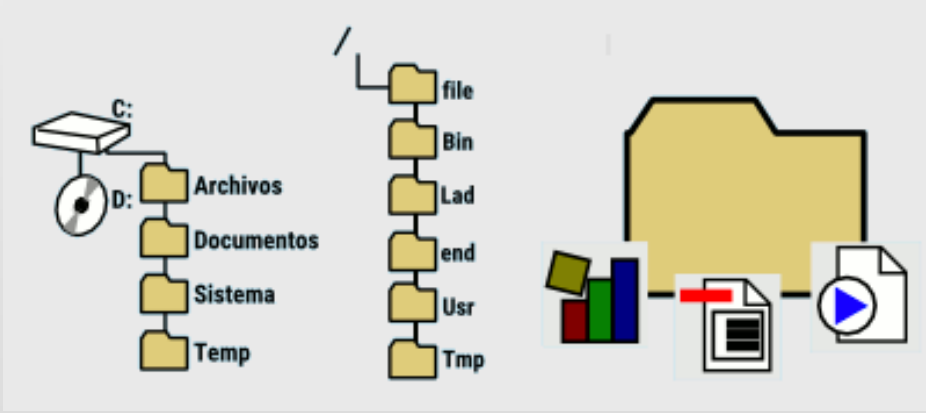
- Administración y planificación del procesador
  - Multiplexacióon de la carga de trabajo
  - Imparcialidad, “justicia” en la ejecución (Fairness)
  - Que no haya bloqueos
  - Manejo de Prioridades

Servicios de un SO

- Administración de Memoria
  - Administración de memoria eficientemente
  - Memoria física vs **memoria virtual**. Jerarquías de memoria
  - Protección de programas que compiten o se ejecutan concurrentemente

Servicios de un SO

- Administración del almacenamiento– Sistema de archivos
  - Acceso a medios de almacenamiento externos
- Administración de dispositivos
  - Ocultamiento de dependencias de HW
  - Administración de accesos simultáneos



Servicios de un SO (cont.)

- Detección de errores y respuestas
  - Errores de HW internos y Externos
    - Errores de Memoria/CPU
    - Errores de Dispositivos
  - Errores de SW
    - Errores Aritméticos
    - Acceso no permitido a direcciones de memoria
  - Incapacidad del SO para conceder una solicitud de una aplicación

Servicios de un SO (cont.)

- Interacción del Usuario (Shell)
- Contabilidad
  - Recoger estadísticas del uso
  - Monitorear parámetros de rendimiento
  - Anticipar necesidades de mejoras futuras
  - Dar elementos si es necesario facturar tiempo de procesamiento

```
# mkdir
Missing argument.
#
# mkdir "Expertos en Linux Windows y Mac Opensys Colombia"
# ls
daemon.cf "Expertos/ en/ Linux/ Windows/ y/ Mac/ Opensys/ Colon
# ls
daemon.cf "Expertos/ en/ Linux/ Windows/ y/ Mac/ Opensys/ Colon
# ls -l
-rwx      root      64 daemon.cf
drwx      root       5 "Expertos/
drwx      root       5 en/
drwx      root       5 Linux/
drwx      root       5 Windows/
drwx      root       5 y/
drwx      root       5 Mac/
drwx      root       5 Opensys/
drwx      root       5 Colombia"/
# _
```

Introducción a los Sistemas Operativos

Introducción – II

I.S.O.

- ☑ Versión: Agosto 2019
- ☑ Palabras Claves: Sistema Operativo, Hardware, System Call, Interacción, Modos de Ejecución, Protección, Kernel

Los temas vistos en estas diapositivas han sido mayormente extraídos del libro de Andrew S. Tanenbaum (Sistemas Operativos Modernos)

Funciones principales de un SO

- ☑ Brindar abstracciones de alto nivel a los procesos de usuario
- ☑ Administrar eficientemente el uso de la CPU
- ☑ Administrar eficientemente el uso de la memoria
- ☑ Brindar asistencia al proceso de E/S por parte de los procesos

Problemas que un SO debe evitar

- ☑ Que un proceso se apropie de la CPU
- ☑ Que un proceso intente ejecutar instrucciones de E/S por ejemplo.
- ☑ Que un proceso intente acceder a una posición de memoria fuera de su espacio declarado.
  - Proteger los espacios de direcciones

Para ello, el SO entre otras debe:

- ☑ Gestionar el uso de la CPU
- ☑ Detectar intentos de ejecución de instrucciones de E/S ilegales
- ☑ Detectar accesos ilegales a memoria
- ☑ Proteger el vector de interrupciones
  - Así como las RAI (Rutinas de atención de interrupciones)

Apoyo del Hardware

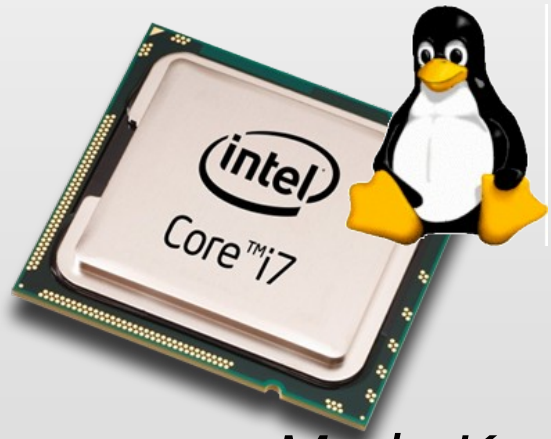
- ☑ **Modos de Ejecución:** Define limitaciones en el conjunto de instrucciones que se puede ejecutar en cada modo
- ☑ **Interrupción de Clock:** Se debe evitar que un proceso se apropie de la CPU
- ☑ **Protección de la Memoria:** Se deben definir límites de memoria a los que puede acceder cada proceso (registros base y límite)

Modos de ejecución

- El bit en la CPU indica el modo actual
- Las instrucciones privilegiadas deben ejecutarse en modo **Supervisor o Kernel**
  - Necesitan acceder a estructuras del kernel, o ejecutar código que no es del proceso
- En modo **Usuario**, el proceso puede acceder sólo a su espacio de direcciones, es decir a las direcciones “propias”.

Modos de ejecución (cont)

- El kernel del SO se ejecuta en modo supervisor
- El resto del SO y los programas de usuario se ejecutan en modo usuario (subconjunto de instrucciones permitidas)



Modo Kernel

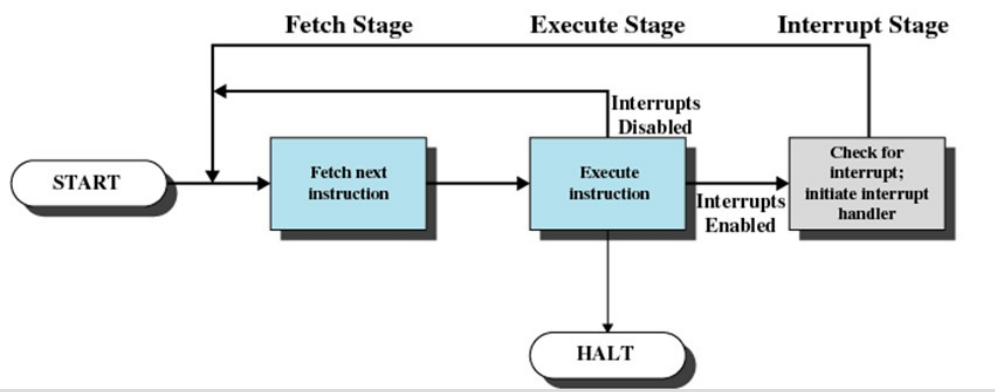
Modo Usuario

Tener en cuenta que...

- ☑ Cuando se arranque el sistema, arranca con el bit en modo supervisor.
- ☑ Cada vez que comienza a ejecutarse un proceso de usuario, este bit se DEBE PONER en modo usuario.
  - Mediante una Instrucción especial.
- ☑ Cuando hay un trap o una interrupción, el bit de modo se pone en modo Kernel.
  - Única forma de pasar a Modo Kernel
  - No es el proceso de usuario quien hace el cambio explicitamente.

Cómo actúa...

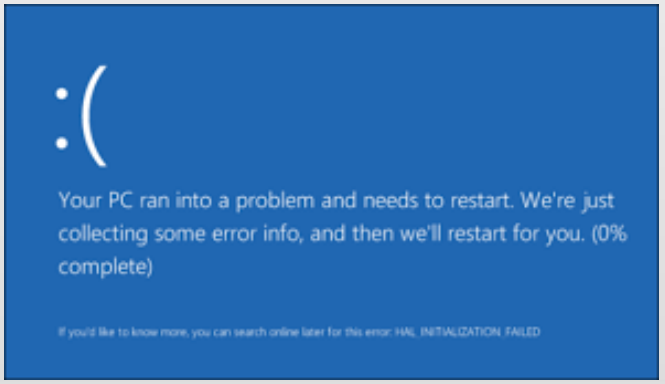
- Cuando el proceso de usuario intenta por sí mismo ejecutar instrucciones que pueden causar problemas (las llamadas instrucciones privilegiadas), el HW lo detecta como una operación ilegal y produce un trap al SO.



En Windows...

- ☑ En WIN2000 el modo núcleo ejecuta los servicios ejecutivos. El modo usuario ejecuta los procesos de usuario.

Cuando un programa se bloquea en modo usuario, a lo sumo se escribe un suceso en el registro de sucesos. Si el bloqueo se produce estando en modo supervisor se genera la BSOD (pantalla azul de la muerte).



Modos de Ejecución

- Procesador Intel 8088 no tenía modo dual de operación ni protección por hardware.
- En MsDos las aplicaciones pueden acceder directamente a las funciones básicas de E/S para escribir directamente en pantalla o en disco.

Resumiendo...

- **Modo kernel:**
  - **Gestión de procesos:** Creación y terminación , planificación, intercambio, sincronización y soporte para la comunicación entre procesos
  - **Gestión de memoria:** Reserva de espacio de direcciones para los procesos, Swapping, Gestión y páginas de segmentos
  - **Gestión E/S:** Gestión de *buffers, r*eserva de canales de E/S y de dispositivos de los procesos
  - **Funciones de soporte:** Gestión de interrupciones, auditoría, monitoreo
- **Modo usuario:**
  - Debug de procesos, definición de protocolos de comunicación gestión de aplicaciones (compilador, editor, aplicaciones de usuario
  - En este modo se llevan a cabo todas las tareas que no requieran accesos privilegiados
  - En este modo no se puede interactuar con el hardware
  - El proceso trabaja en su propio espacio de direcciones

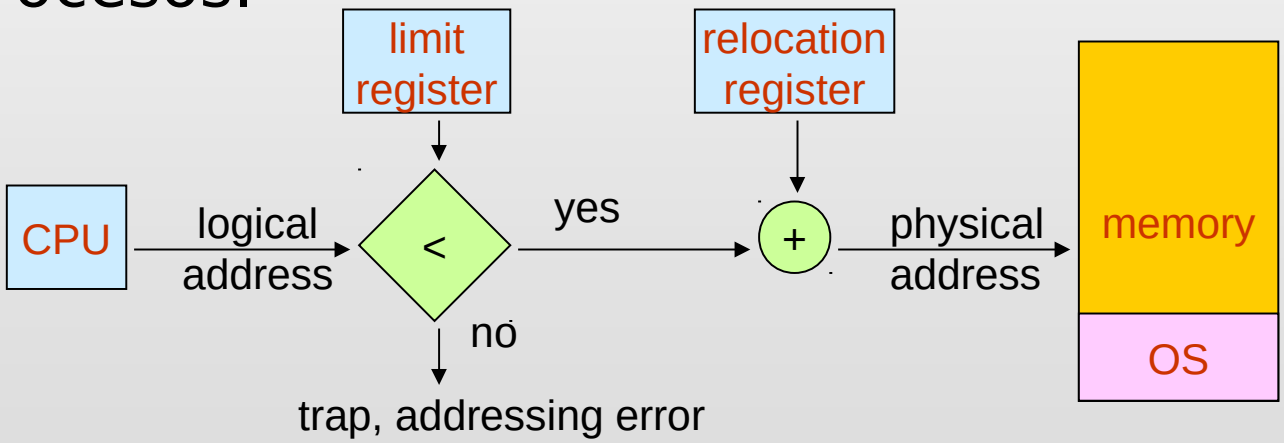
Protección de la memoria

- ☑ Delimitar el espacio de direcciones del proceso
- ☑ Poner limites a las direcciones que puede utilizar un proceso
  - Por ejemplo: Uso de un registro base y un registro límite
  - El kernel carga estos registros por medio de instrucciones privilegiadas. Esta acción sólo puede realizarse en modo Kernel

Protección de la memoria (cont)

La memoria principal aloja al SO y a los procesos de usuario

- ✓ El kernel debe proteger para que los procesos de usuario no puedan acceder donde no les corresponde
- ✓ El kernel debe proteger el espacio de direcciones de un proceso del acceso de otros procesos.



Protección de la E/S

- Las instrucciones de E/S se definen como privilegiadas.
- Deben ejecutarse en Modo Kernel
  - Se deberían gestionar en el kernel del sistema operativo
  - Los procesos de usuario realizan E/S a través de llamadas al SO (es un servicio del SO)

Protección de la CPU

- Uso de interrupción por clock para evitar que un proceso se apropie de la CPU
- Se implementa normalmente a través de un clock y un contador.
- El kernel le da valor al contador que se decrementa con cada tick de reloj y al llegar a cero puede expulsar al proceso para ejecutar otro.

Protección de la CPU (cont.)

- Las instrucciones que modifican el funcionamiento del reloj son privilegiadas.
- Se le asigna al contador el valor que se quiere que se ejecute un proceso.
- Se la usa también para el cálculo de la hora actual, basándose en cantidad de interrupciones ocurridas cada tanto tiempo y desde una fecha y hora determinada.

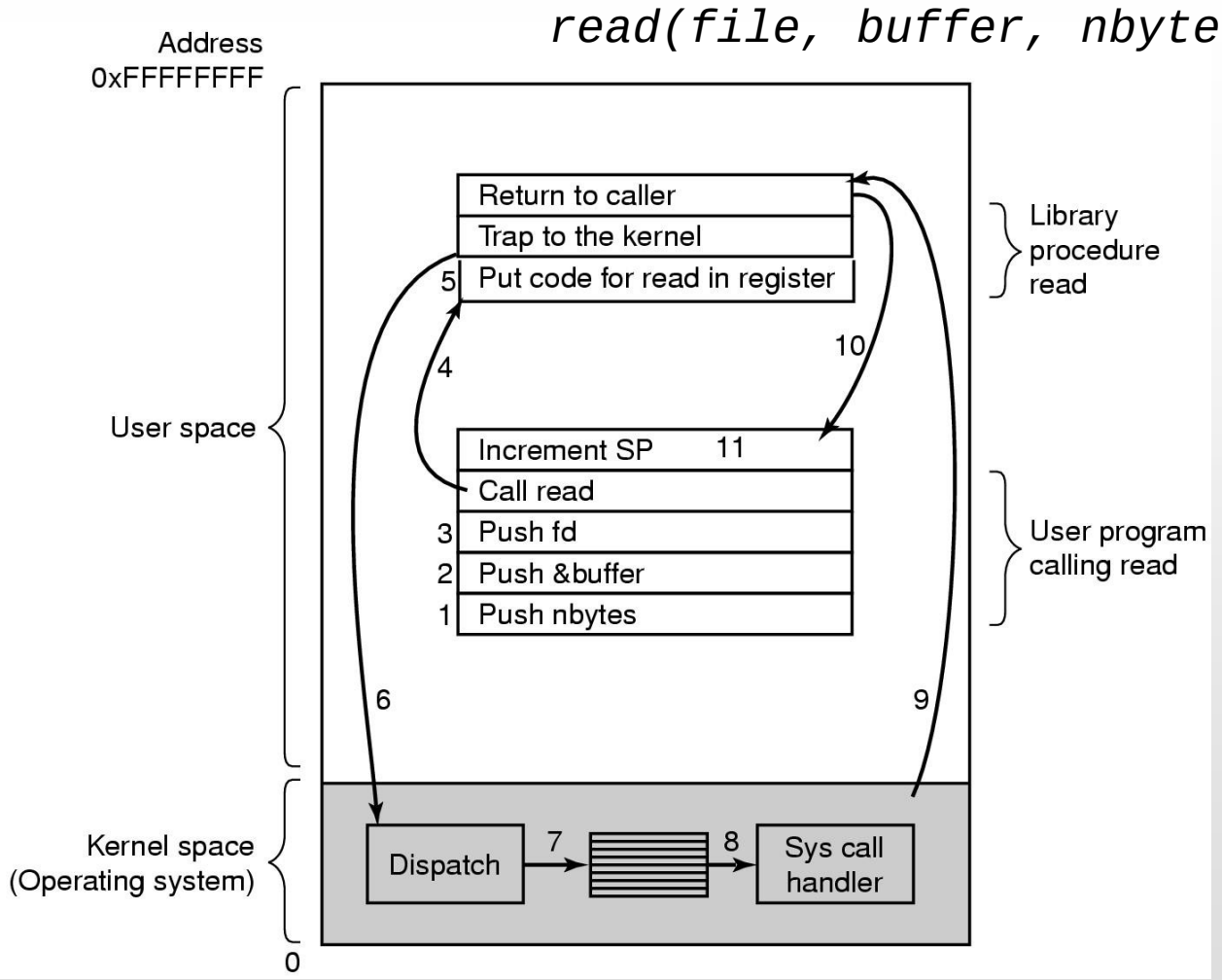
System Calls

- Es la forma en que los programas de usuario acceden a los servicios del SO.
- Los parámetros asociados a las llamadas pueden pasarse de varias maneras: por registros, bloques o tablas en memoria ó la pila.

```
count=read(file, buffer, nbytes);
```

- Se ejecutan en modo kernel o supervisor

System calls (cont.)



System Calls - Categorias

- Categorías de system calls:
  - Control de Procesos
  - Manejo de archivos
  - Manejo de dispositivos
  - Mantenimiento de información del sistema
  - Comunicaciones

System calls - Categorias (cont.)

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

System calls - Categorias (cont.)

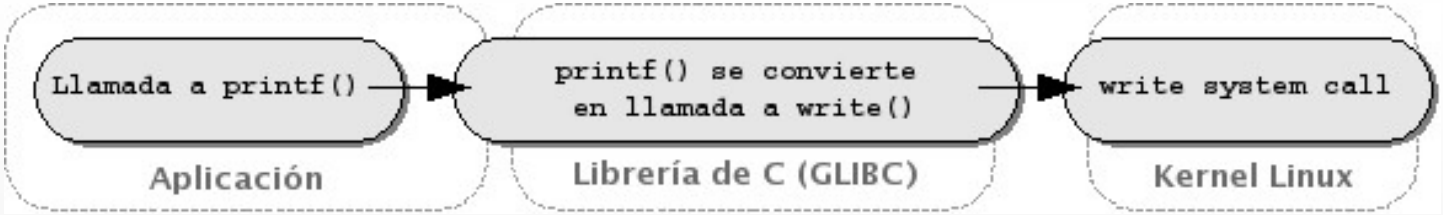
Directory and file system management

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

Ejemplo - System Call Linux

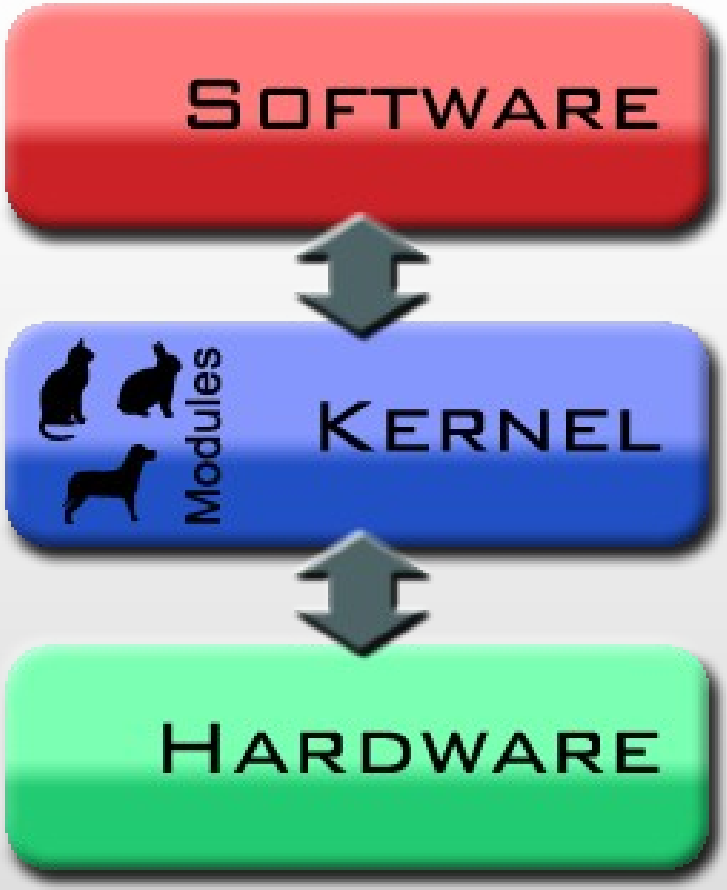


- Para activar iniciar la system call se indica:
  - el número de syscall que se quiere ejecutar
  - los parámetros de esa syscall
- Luego se emite una interrupción para pasar a modo Kernel y gestionar la systemcall
- El manejador de interrupciones (syscall handler) evalúa la system call deseada y la ejecuta

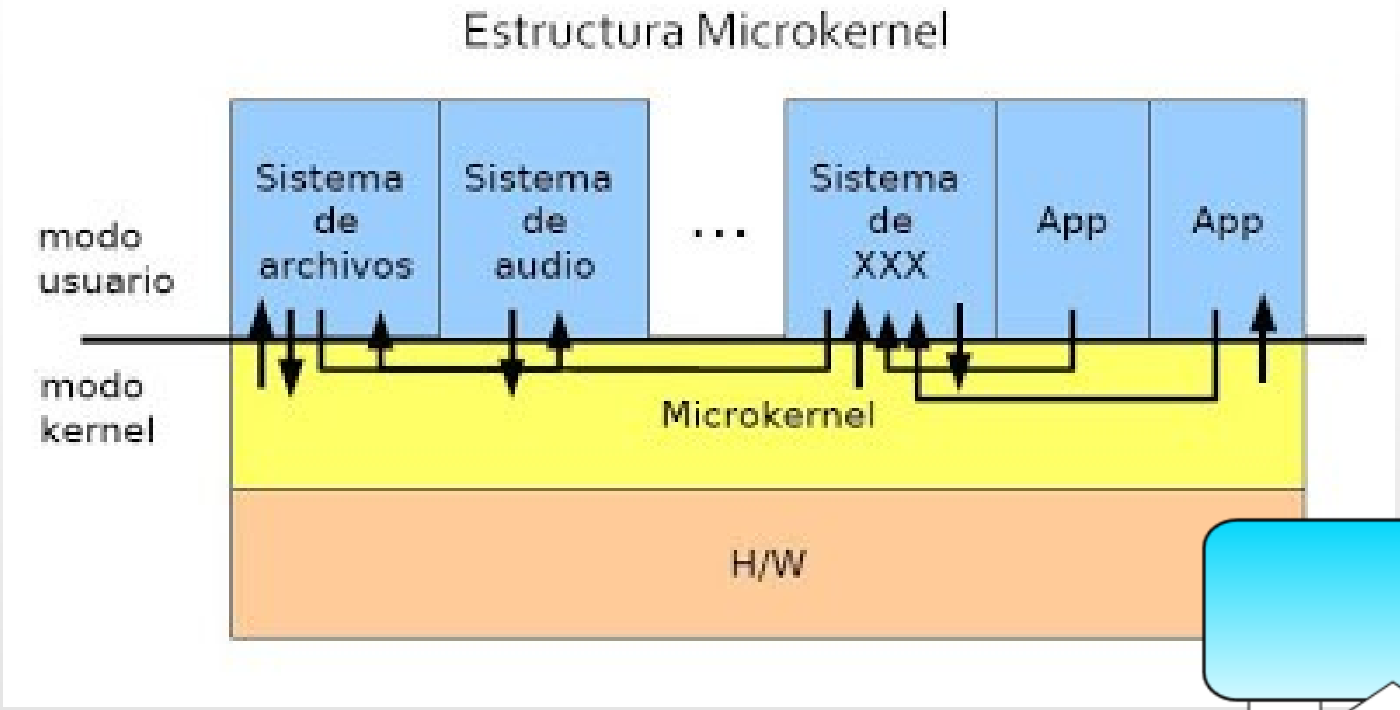
Systems Calls - Ejemplos

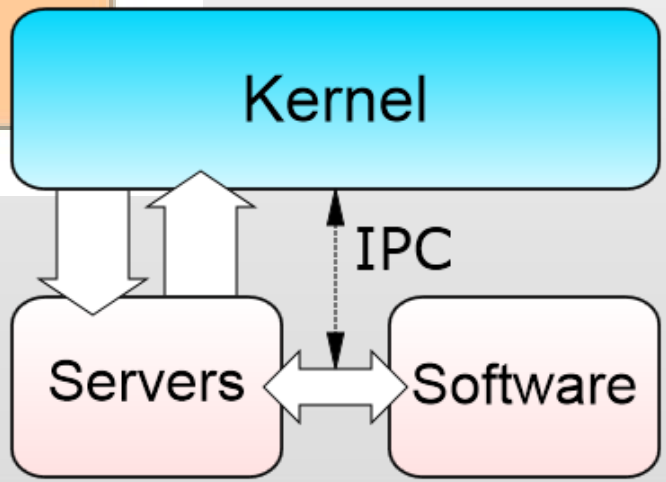
UNIX	Win32	Description
fork	CreateProcess	Create a new process
waitpid	WaitForSingleObject	Can wait for a process to exit
execve	(none)	CreateProcess = fork + execve
exit	ExitProcess	Terminate execution
open	CreateFile	Create a file or open an existing file
close	CloseHandle	Close a file
read	ReadFile	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
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

Tipos de kernel - Monolítico

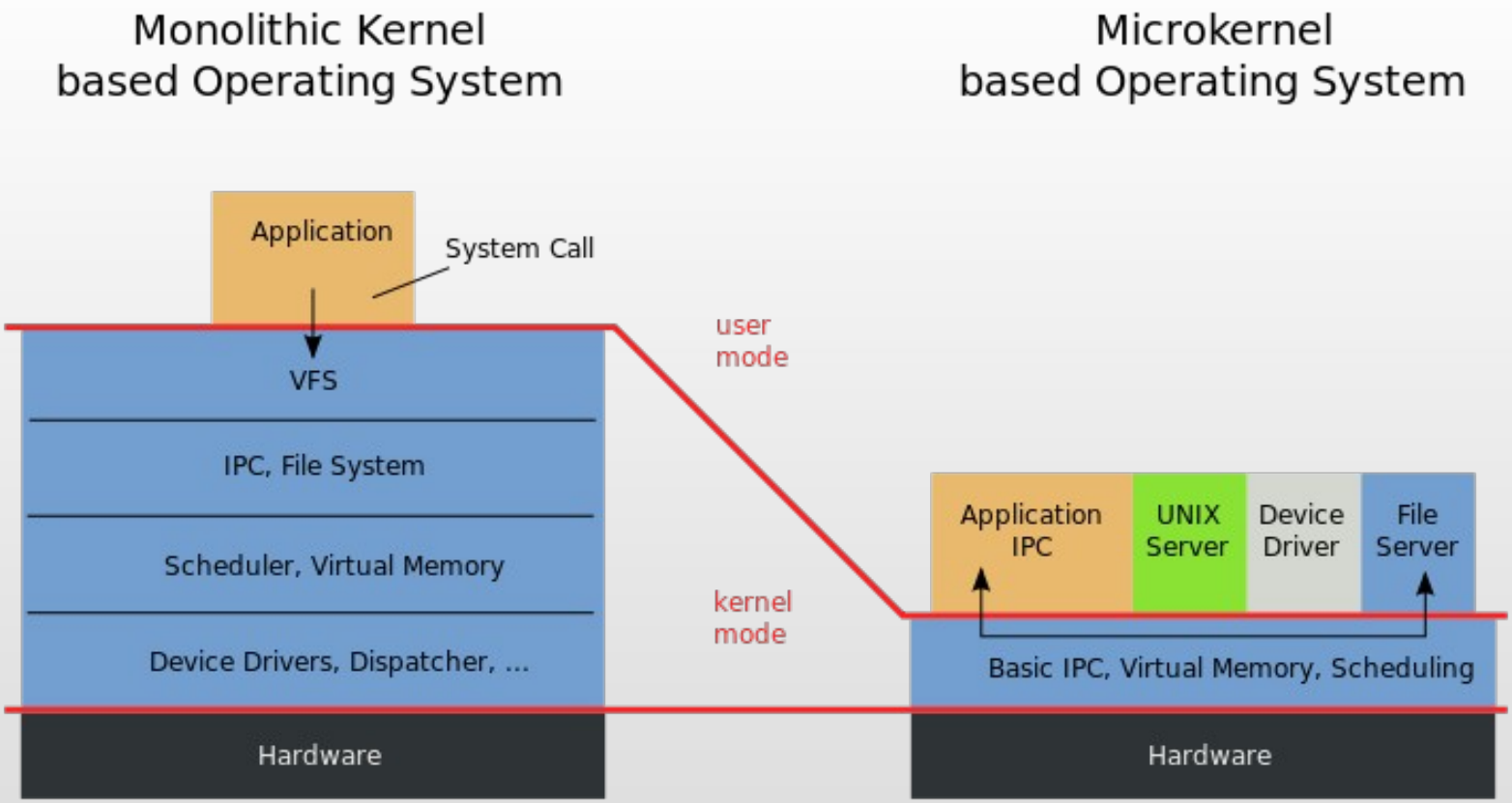


Tipos de kernel - Microkernel





Monolitico Vs. Microkernel



Introducción a los Sistemas Operativos

Anexo – Arquitectura de Comp.

I.S.O.

- ☑ Versión: Agosto 2018
- ☑ Palabras Claves: Sistemas Operativos, Harware, Interrupciones, Registros

Los temas vistos en estas diapositivas han sido mayormente extraídos del libro de William Stallings (Sistemas Operativos: Aspectos internos y principios de diseño)

Elementos Básicos de una computadora

- Procesador
- Memoria Principal
  - Volátil
  - Se refiere como memoria real o primaria
- Componentes de E/S
  - Dispositivos de memoria secundaria
  - Equipamiento de comunicación
  - Monitor / teclado / mouse
- Bus Sistema
  - comunicación entre procesadores, memoria, dispositivos de E/S

Componentes de alto nivel

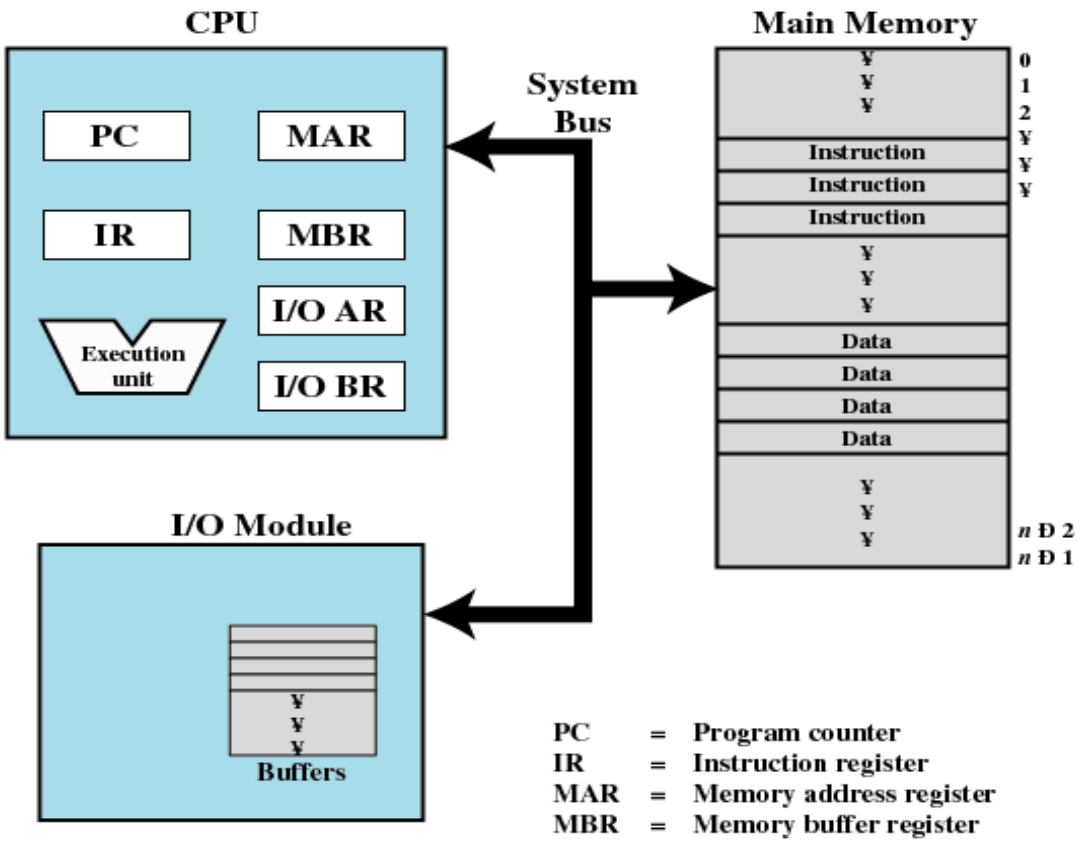


Figure 1.1 Computer Components: Top-Level View

Registros del Procesador

- Visibles por el usuario
  - Registros que pueden ser usados por las aplicaciones
- De Control y estado
  - Para control operativo del procesador
  - Usados por rutinas privilegiadas del SO para controlar la ejecución de procesos

Registros Visibles por el usuario

- Pueden ser referenciados por lenguaje de máquina
- Disponible para programas/aplicaciones
- Tipos de registros
  - Datos
  - Direcciones
    - Index
    - Segment pointer
    - Stack pointer

Registros de Control y Estado

- Program Counter (PC)
  - Contiene la dirección de la proxima instrucción a ser ejecutada
- Instruction Register (IR)
  - Contiene la instrucción a ser ejecutada
- Program Status Word (PSW)
  - Contiene códigos de resultado de operaciones
  - habilita/deshabilita Interrupciones
  - Indica el modo de ejecución (Supervisor/usuario)

Ciclo Ejecución de Instrucción

- Dos pasos
  - Procesador lee la instrucción desde la memoria
  - Procesador ejecuta la instrucción

Ciclo Instrucción

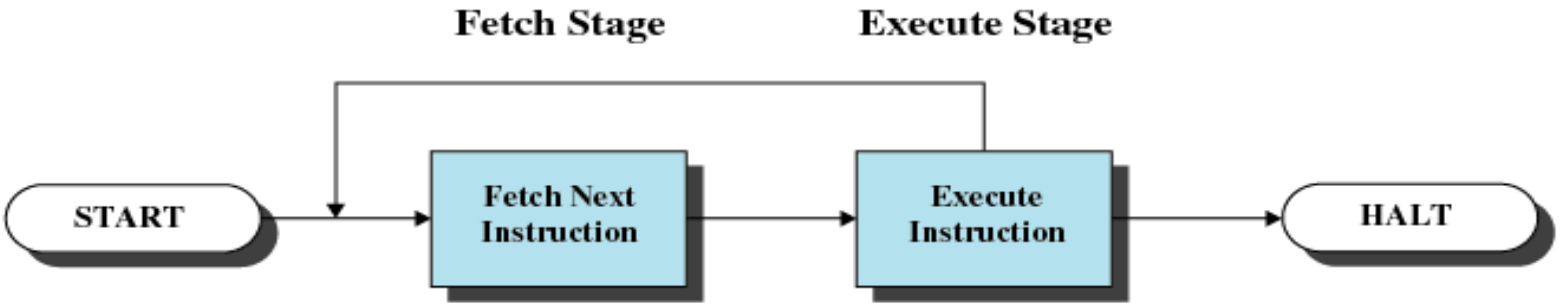


Figure 1.2 Basic Instruction Cycle

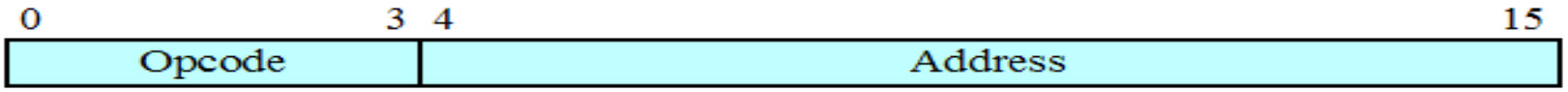
Instrucción: Fetch y Execute

- El procesador busca (fetch) la instrucción en la memoria
  - (PC) → IR
- El PC se incrementa después de cada fetch para apuntar a la próxima instrucción
  - PC = PC + 4

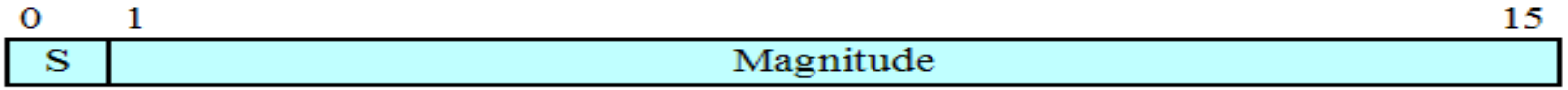
IR - Instruction Register

- La instrucción referenciada por el PC se almacena en el IR y se ejecuta
- Categorías de instrucciones
  - Procesador - Memoria
    - Transfiere datos entre procesador y memoria
  - Procesador - E/S
    - Transfiere datos a/o desde periféricos
  - Procesamiento de Datos
    - Operaciones aritméticas o lógicas sobre datos
  - Control
    - Alterar secuencia de ejecución

Características de una máquina hipotética



(a) Instruction format



(b) Integer format

Program Counter (PC) = Address of instruction
Instruction Register (IR) = Instruction being executed
Accumulator (AC) = Temporary storage

(c) Internal CPU registers

0001 = Load AC from Memory
0010 = Store AC to Memory
0101 = Add to AC from Memory

(d) Partial list of opcodes

Figure 1.3 Characteristics of a Hypothetical Machine

Ej. de una ejecución de programa

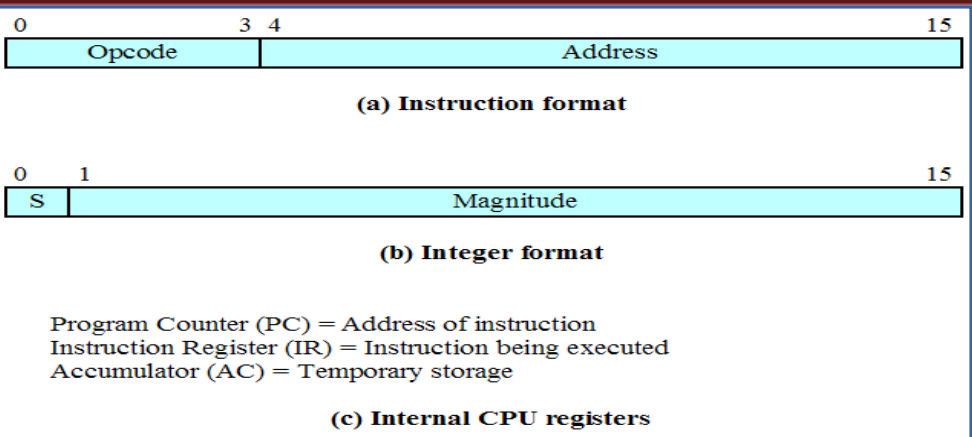


Figure 1.3 Characteristics of

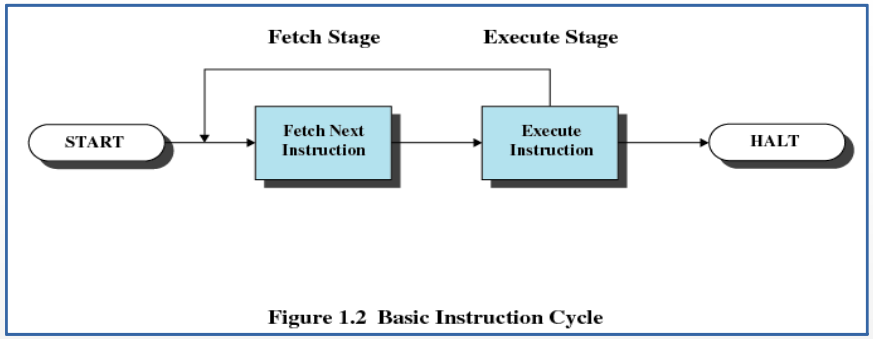
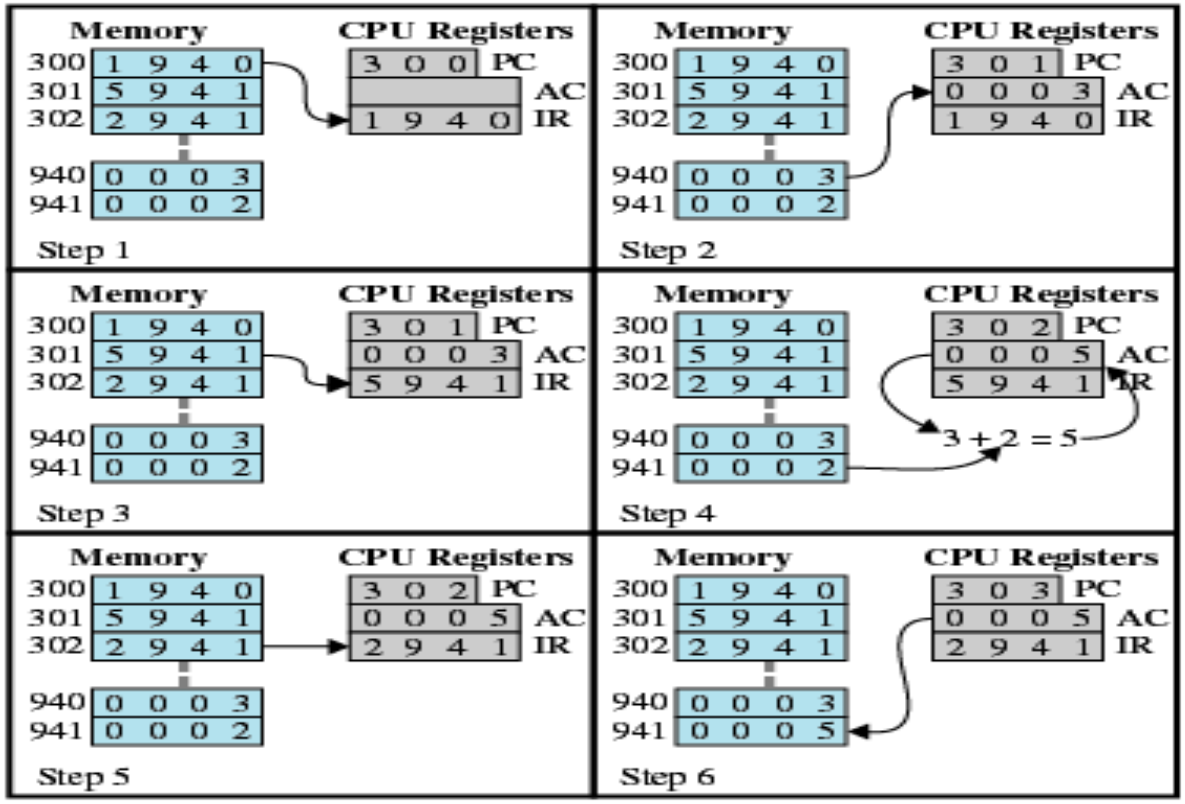


Figure 1.2 Basic Instruction Cycle



Interrupciones

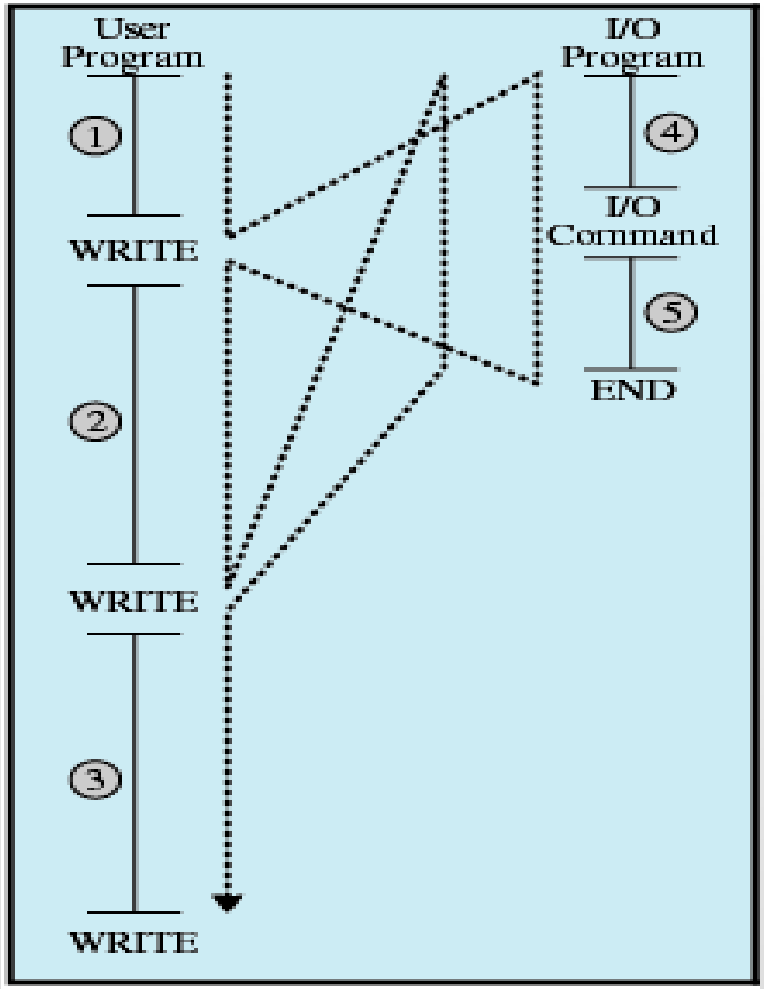
- Interrumpen el secuenciamiento del procesador durante la ejecución de un proceso
- Dispositivos de E/S más lentos que el procesador
  - Procesador debe esperar al dispositivo

Clases de Interrupciónes

Table 1.1 Classes of Interrupts

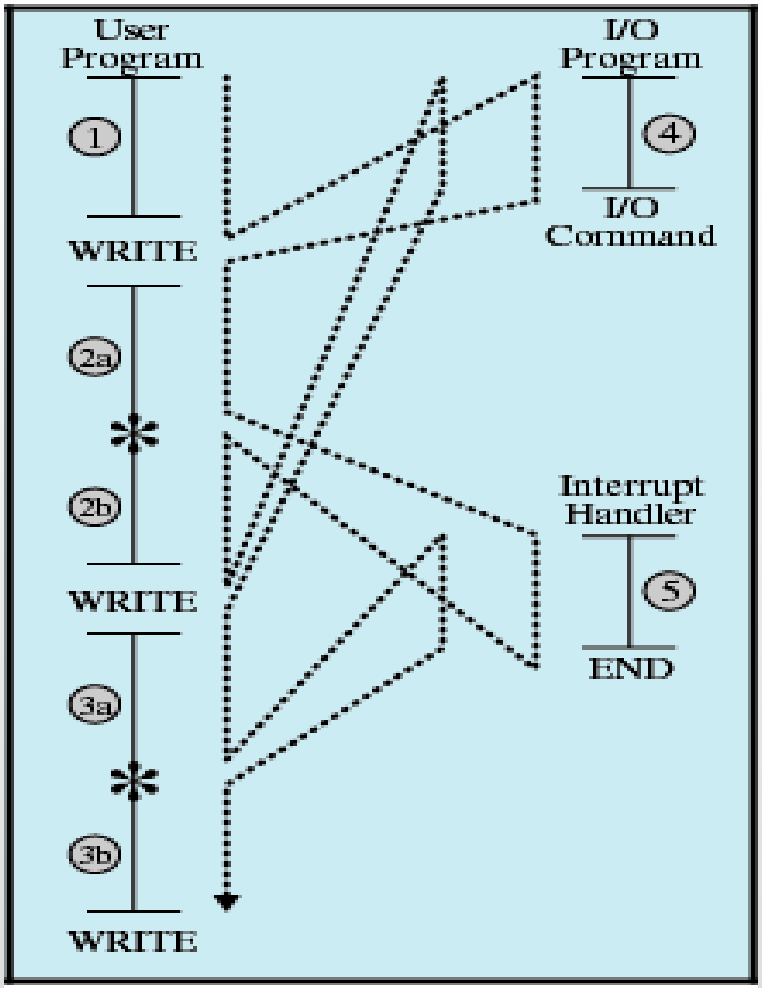
Program	Generated by some condition that occurs as a result of an instruction execution, such as arithmetic overflow, division by zero, attempt to execute an illegal machine instruction, and reference outside a user's allowed memory space.
Timer	Generated by a timer within the processor. This allows the operating system to perform certain functions on a regular basis.
I/O	Generated by an I/O controller, to signal normal completion of an operation or to signal a variety of error conditions.
Hardware failure	Generated by a failure, such as power failure or memory parity error.

Flujo de control SIN interrupciones



(a) No interrupts

Flujo de control CON interrupciones



(b) Interrupts; short I/O wait

Interrupt Handler

- Programa (o rutina) que determina la naturaleza de una interrupción y realiza lo necesario para atenderla
  - Por ejemplo, para un dispositivo particular de E/S
- Generalemente es parte del SO

Interrupciones

- Suspende la secuencia normal de ejecución

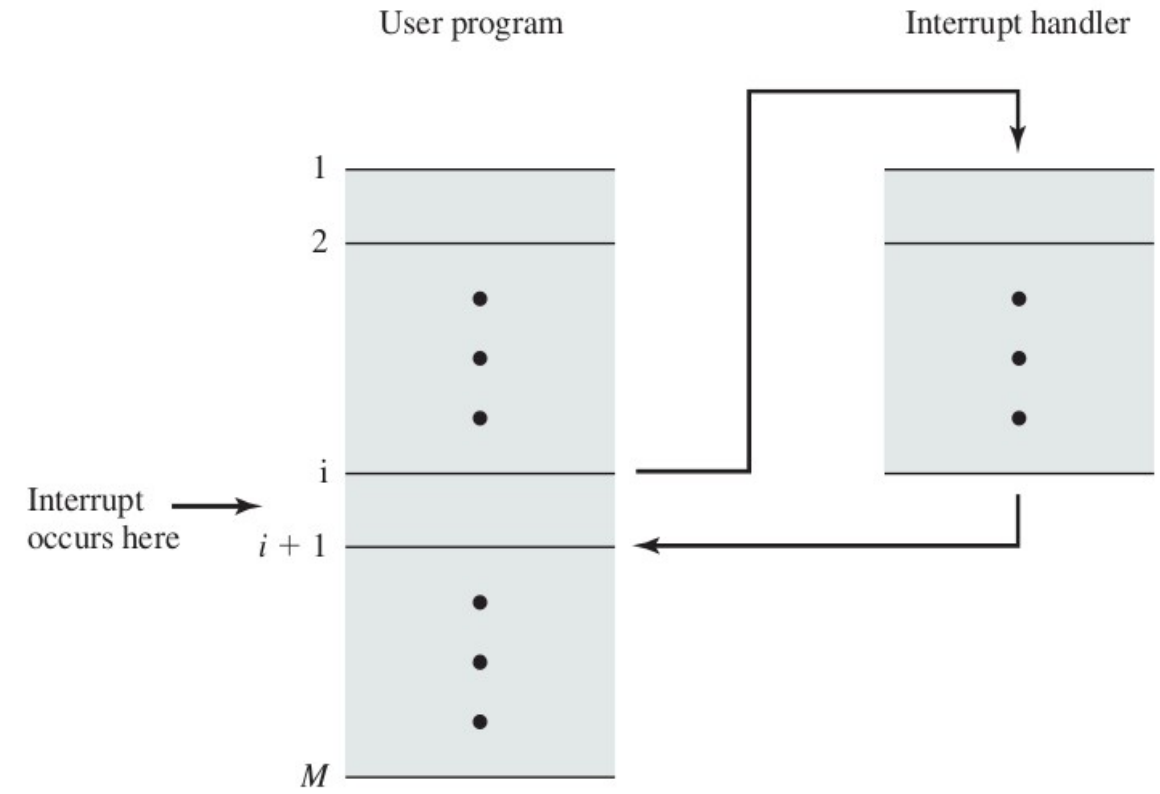


Figure 1.6 Transfer of Control via Interrupts

Ciclo de interrupción

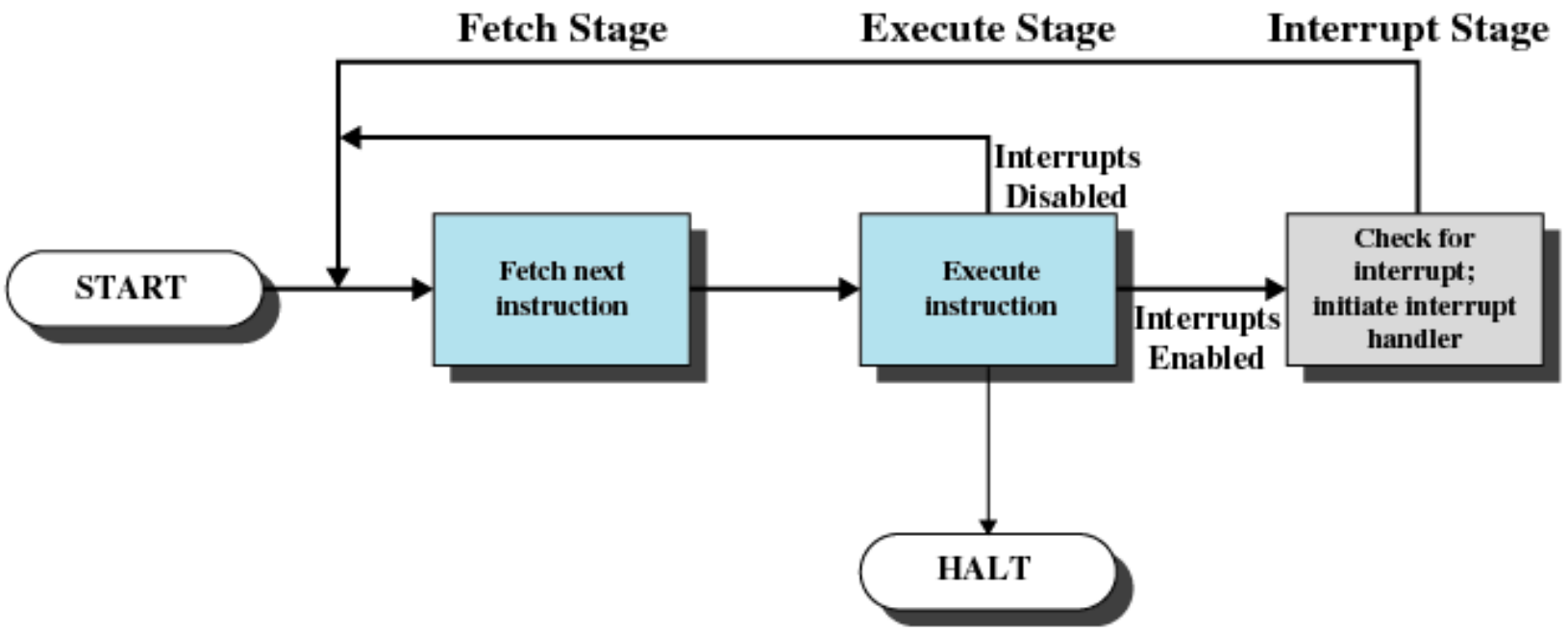


Figure 1.7 Instruction Cycle with Interrupts

Ciclo de interrupción

- El procesador chequea la existencia de interrupciones.
- Si no existen interrupciones, la próxima instrucción del programa es ejecutada
- Si hay pendiente alguna interrupción, se suspende la ejecución del progama actual y se ejecuta la rutina de manejo de interrupciones.

Simple Interrupt Processing

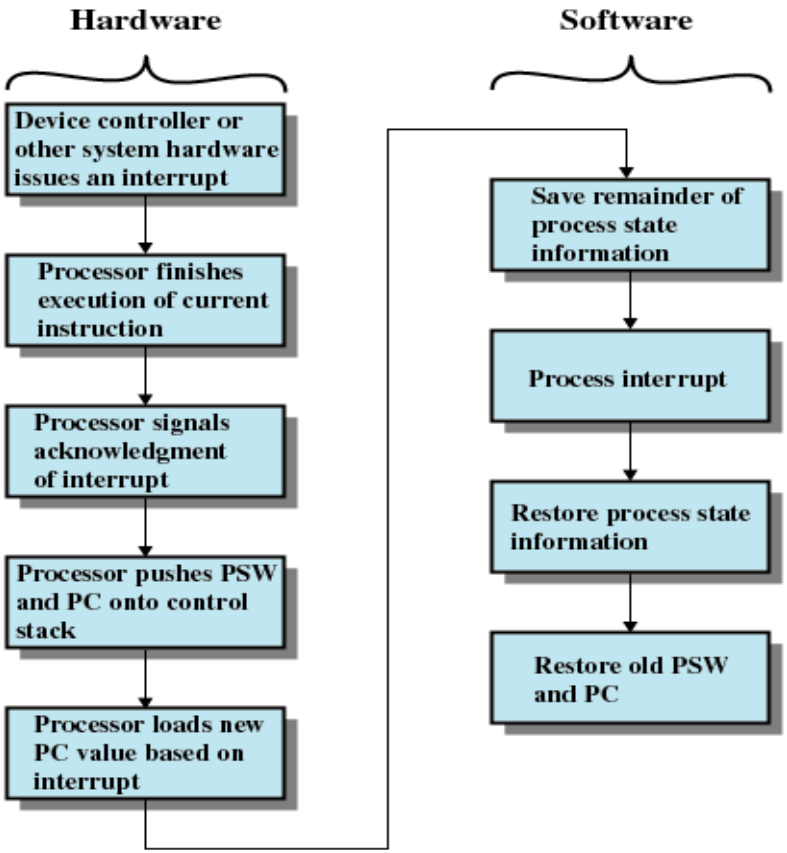
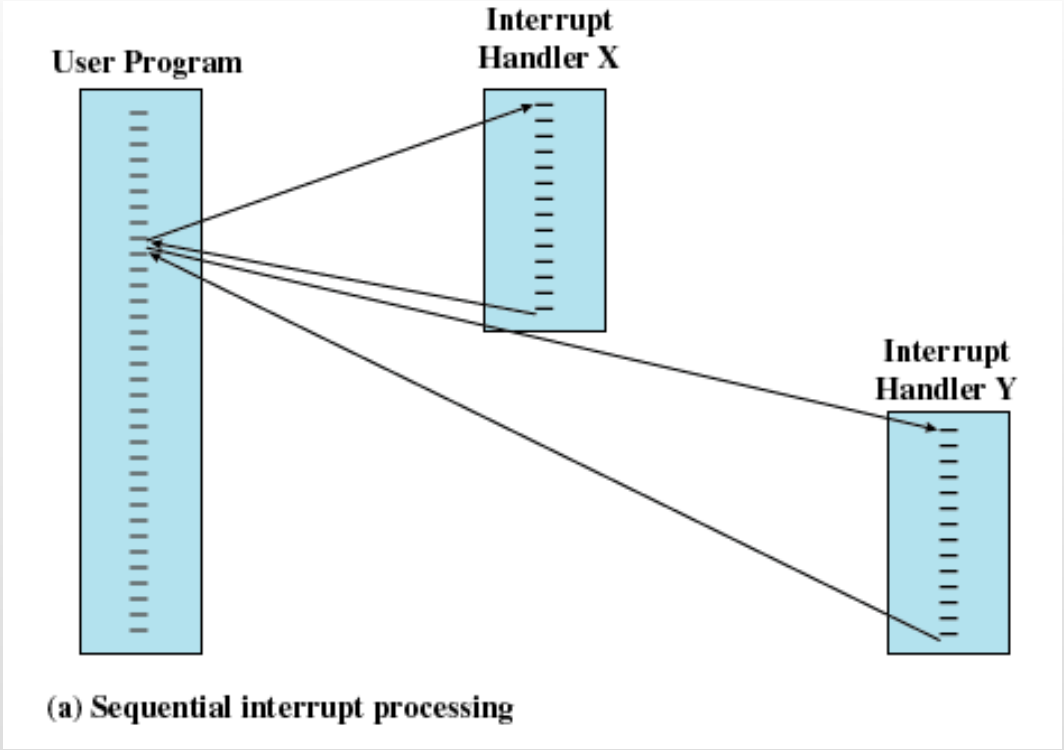


Figure 1.10 Simple Interrupt Processing

Multiples Interrupciones

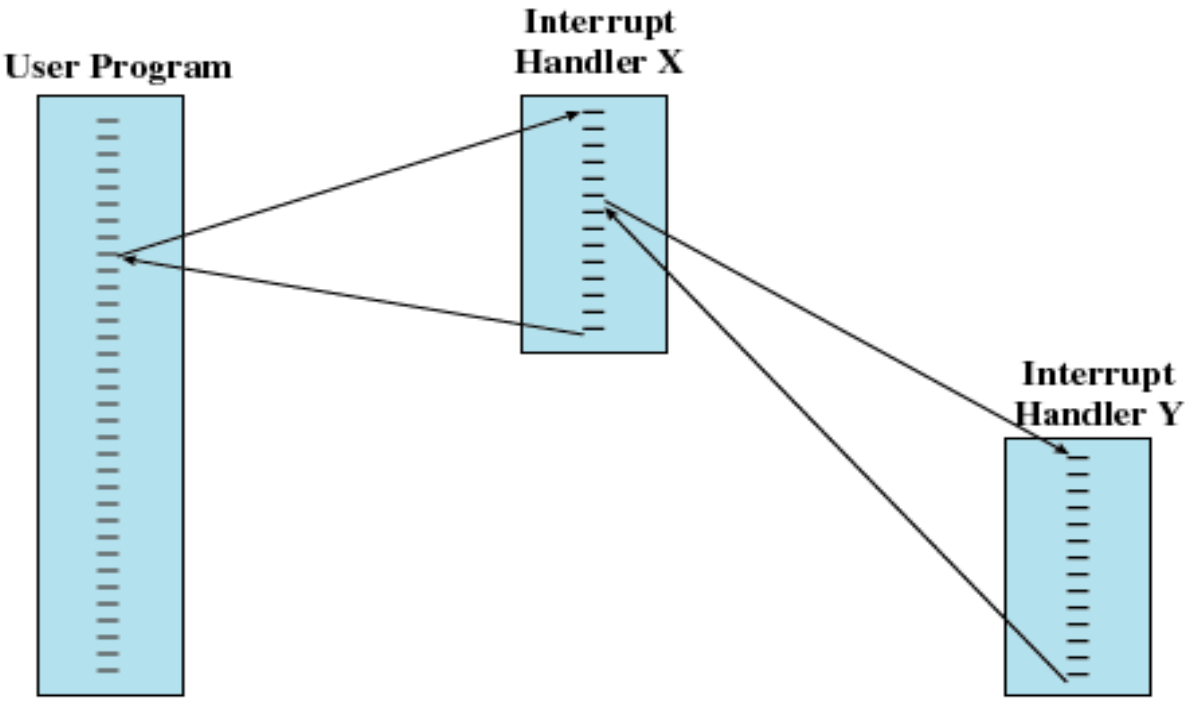
- ☑ Deshabilitar las interrupciones mientras una interrupción está siendo procesada.



(a) Sequential interrupt processing

Multiples Interrupciones

- ☑ Definir prioridades a las interrupciones



(b) Nested interrupt processing

Multiples Interrupciones

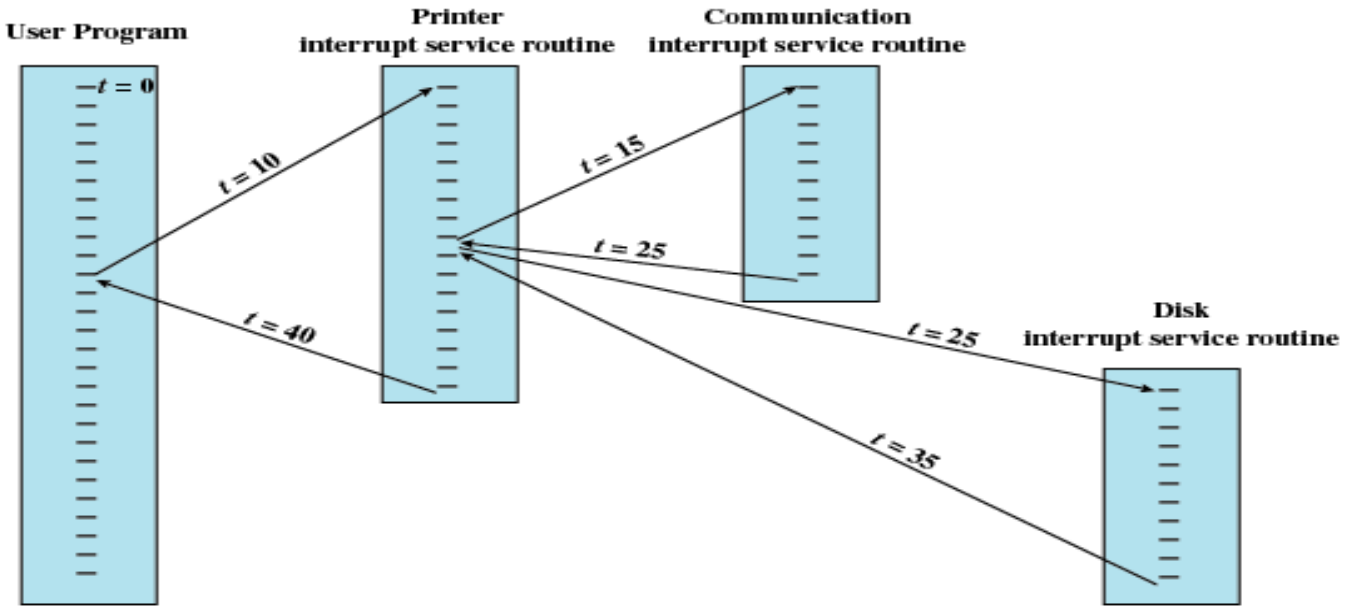


Figure 1.13 Example Time Sequence of Multiple Interrupts

Introducción a los Sistemas Operativos

Anexo – Evolución

I.S.O.

- ☑ Versión: Agosto 2018
- ☑ Palabras Claves: Sistema Operativo, Servicios, Evolución, Batch, Multiprogramación, Timesharing

Los temas vistos en estas diapositivas han sido mayormente extraídos del libro de William Stallings (Sistemas Operativos: Aspectos internos y principios de diseño)

Evolución de un S.O.

Los SO evolucionan con el objeto de:

- Soportar nuevos tipos de HW
- Brindar nuevos Servicios
- Ofrecer mejoras y alternativas a problemas existentes
  - en la planificación
  - en el manejo de la memoria
  - etc

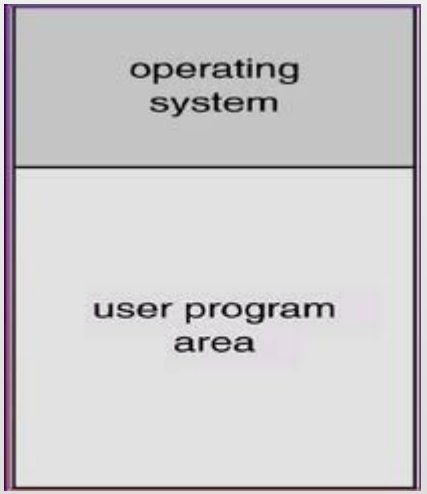
S.O. - Evolución Histórica

- Procesamiento en Serie
  - No existía un SO
  - Máquinas eran utilizadas desde una consola que contenía luces, interruptores, dispositivos de entrada e impresoras.
  - Problemas:
    - Planificación. Alto nivel de especialización. Costos
    - Configuración: Carga del compilador, fuente, salvar el programa compilado, carga y linkeo.

S.O. - Evolución Histórica (cont.)

- Sistemas por Lotes Sencillos (batch)
  - Monitor Residente
    - Software que controla la secuencia de eventos
    - Los trabajos se colocan juntos
    - Los programas vuelven al monitor cuando finaliza la ejecución
    - No hay interacción con el usuario

  mientras se ejecutan los trabajos

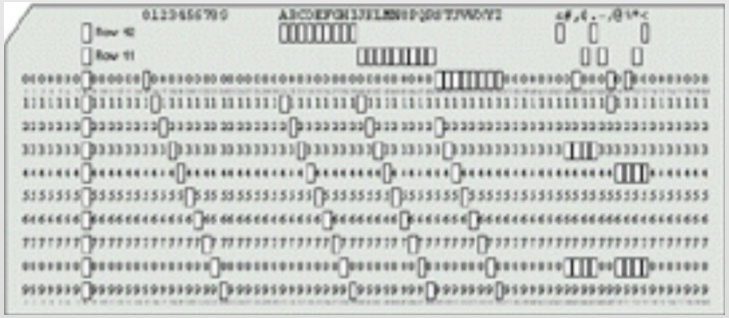


S.O. - Evolución Histórica (cont.)

☑ Batch processing

The elements of the basic IBM 1401 system are the 1401 Processing Unit, 1402 Card Read-Punch, and 1403 Printer.

☑ Punching cards

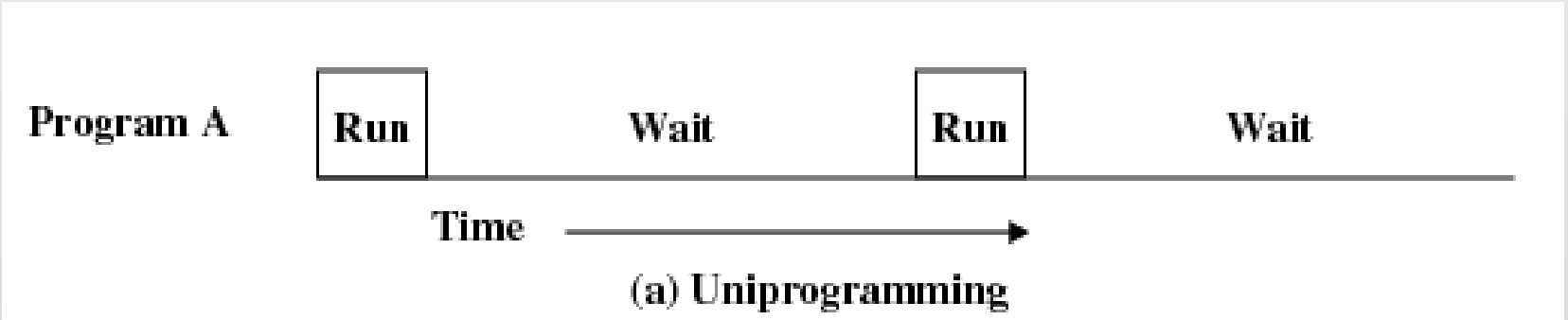


Sistema Batch

Baja utilización de la CPU

Dispositivos de E/S mucho mas lentos con respecto a la CPU

Ante instrucción de E/S, el procesador permanece ocioso. Cuando se completa la E/S, se continua con la ejecución del programa que se estaba ejecutando



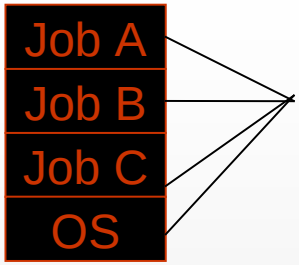
Multiprogramación

- La operación de los sistemas batch se vio beneficiada del spooling de las tareas, al solapar la E/S de una tarea de la ejecución de otra
- Al estar las tareas cargadas en disco, ya no era necesario ejecutarlas en el orden en el que fueron cargadas (job scheduling)
- El SO mantiene varias tareas en memoria al mismo tiempo.

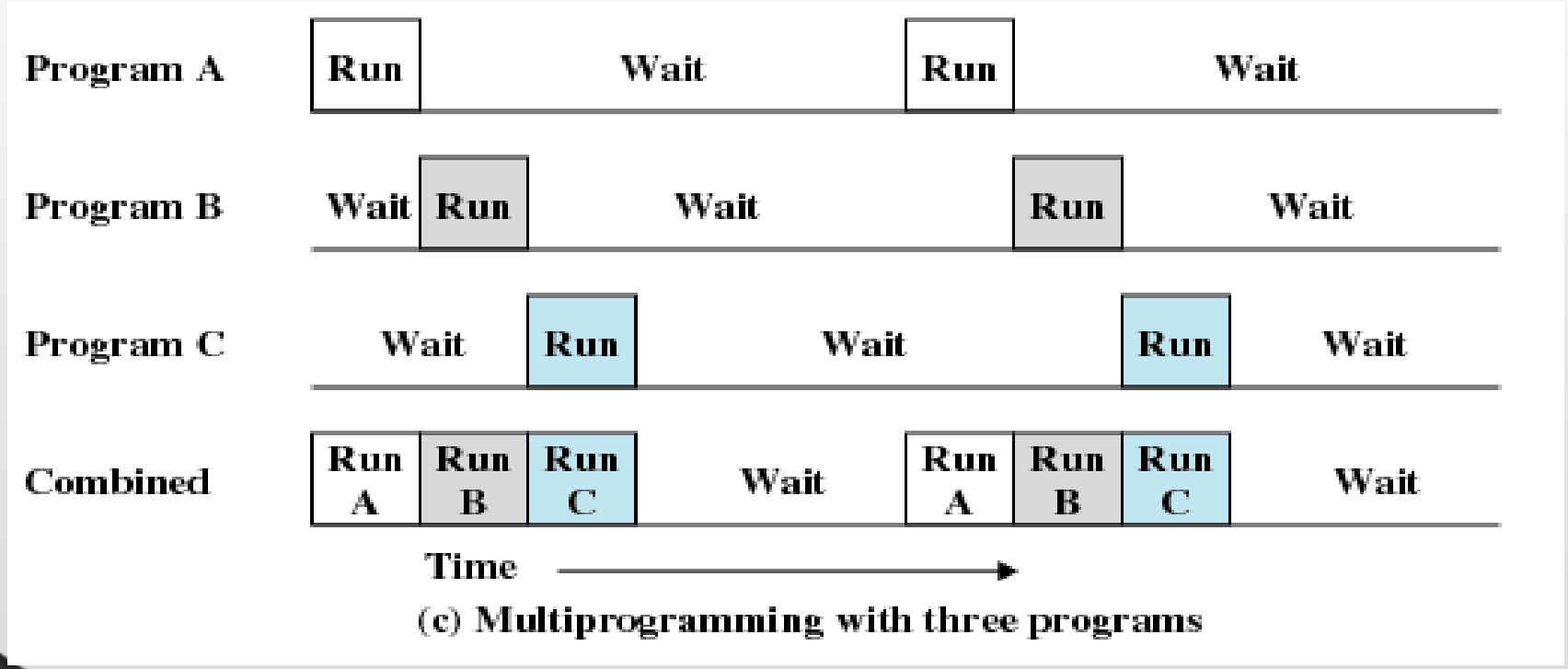
Multiprogramación (cont)

- La secuencia de programas es de acuerdo a prioridad u orden de llegada
- Cuando el proceso necesita realizar una operación de E/S, la CPU en lugar de permanecer ociosa, es utilizada para otro proceso.
- Después que se completa la atención de la interrupción, el control puede o no retornar al programa que se estaba ejecutando al momento de la interrupción

Multiprogramación (cont)



Memory partitions

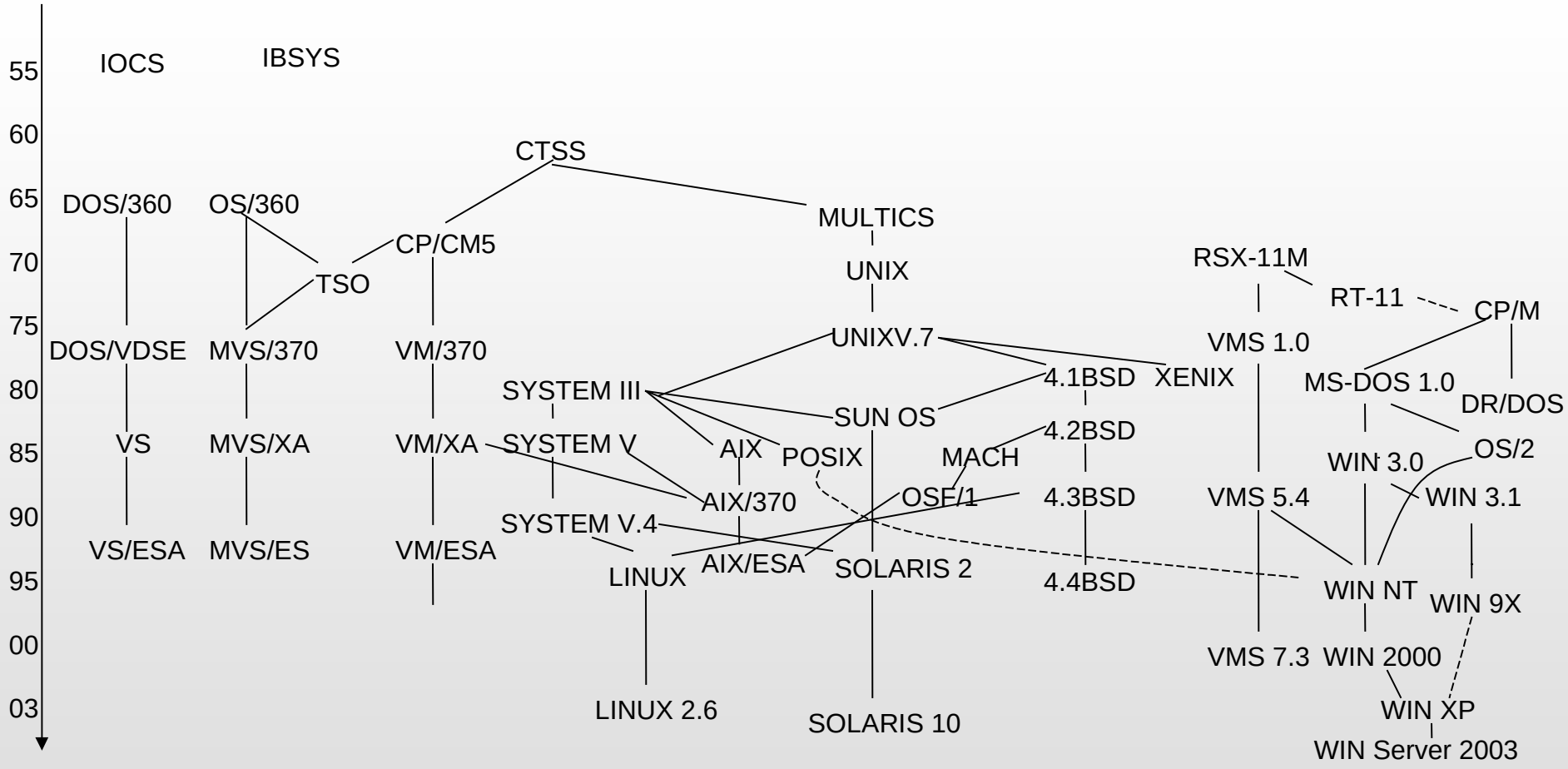


(c) Multiprogramming with three programs

Tiempo Compartido

- Utiliza la multiprogramación para manejar múltiples trabajos interactivos
- El tiempo del procesador es compartido entre múltiples trabajos.
- Múltiples usuarios podrían acceder simultáneamente al sistema utilizando terminales
- Los procesos usan la CPU por un periodo máximo de tiempo, luego del cual se le da la CPU a otro proceso

Operating Systems Evolution



Referencias

- ☑ Historia de los S.O.

http://es.wikipedia.org/wiki/Historia_y_evoluci%C3%B3n_de_los_sistemas_operativos

- ☑ Línea del tiempo

http://en.wikipedia.org/wiki/Operating_systems_timeline

Referencias

- ☑ Historia de la primer Computadora Argentina

http://www.tectv.gob.ar/programacion-series/clementina

CLEMENTINA / EPISODIOS

Capítulo 1
PRESENTACIÓN EN SOCIEDAD

VER

Capítulo 2
TÉ CON AMIGOS

VER

Capítulo 3
UN AMOR LÓGICO

VER

Capítulo 4
EL LEGADO

VER

Introducción a los Sistemas Operativos

Introducción – IV

Anexo llamadas al Sistema

Objetivo

- Programar un llamado a una “System Call” de manera directa. Sin utilizar ninguna librería.
- Considerar distintos aspectos al intentar realizar lo mismo en las siguientes arquitecturas:
  - 32 bits
  - 64 bits

Hello World!!

- Para programar el clasico “hello world” se necesitan mínimo realizar hacer 2 llamadas al sistema:
  - Una para escribir en pantalla un mensaje
    SYSCALL WRITE
  - Otra para terminar la ejecución de un proceso
    SYSCALL EXIT

Hello World!!

- Para obtener información sobre estas SYSCALLs podemos utilizar los manuales del sistema.
- El comando man permite acceder a distintos tipos de documentación, en particular a información referida a systemcalls
  - write (man 2 write)
  - exit (man exit)

Hello World!!!

- Los manuales de las system calls permiten saber cuales son los parámetros

```
NAME
       write - write to a file descriptor

SYNOPSIS
       #include <unistd.h>

       ssize_t write(int fd, const void *buf, size_t count);

DESCRIPTION
       write()  writes  up  to  count bytes from the buffer pointed buf to the
       file referred to by the file descriptor fd.
```

```
NAME
       exit - cause normal process termination

SYNOPSIS
       #include <stdlib.h>

       void exit(int status);

DESCRIPTION
       The  exit() function causes normal process termination and the value of
       status & 0377 is returned to the parent (see wait(2)).
```

Número de syscalls a utilizar

- Para indicarle al sistema operativo lo que queremos hacer (write o exit), es necesario saber cuál es el número asociado que tiene cada una de las syscalls
- Puede ser distinto en distintas arquitucturas

Del github de Linus Torvald

- https://github.com/torvalds/linux/blob/master/arch/x86/entry/syscalls/syscall_32.tbl
- https://github.com/torvalds/linux/blob/master/arch/x86/entry/syscalls/syscall_64.tbl

Hello World en x86 32bit

https://github.com/torvalds/linux/blob/master/arch/x86/entry/syscalls/syscall_32.tbl

```
# 32-bit system call numbers and entry vectors
#
# The format is:
# <number> <abi> <name> <entry point> <compat entry point>
#
# The abi is always "i386" for this file.
#
0       i386    restart_syscall         sys_restart_syscall
1       i386    exit                    sys_exit
2       i386    fork                    sys_fork                        sys_fork
3       i386    read                    sys_read
4       i386    write                   sys_write
5       i386    open                    sys_open                        compat_sys_open
6       i386    close                   sys_close
```

En x86 32bit las sistem calls tienen los siguientes números:

- write → syscall número 4
- exit → syscall número 1

Hello World en x86 64bit

https://github.com/torvalds/linux/blob/master/arch/x86/entry/syscalls/syscall_64.tbl

```
# 64-bit system call numbers and entry vectors
#
# The format is:
# <number> <abi> <name> <entry point>
#
# The abi is "common", "64" or "x32" for this file.
#
0       common  read                    sys_read
1       common  write                   sys_write
2       common  open                    sys_open
3       common  close                   sys_close
```

```
57      common  fork                    sys_fork/ptregs
58      common  vfork                   sys_vfork/ptregs
59      64      execve                  sys_execve/ptregs
60      common  exit                    sys_exit
61      common  wait4                   sys_wait4
62      common  kill                    sys_kill
```

En x86 64bit las sistem calls tienen los siguientes números:

- write → syscall número 1
- exit → syscall número 60

Pasaje de parámetros en x86 32bit

- https://syscalls.kernelgrok.com/
  - EAX lleva el numero de syscall que se desea ejecutar
  - EBX lleva el primer parámetro
  - ECX lleva el segundo parámetro
  - EDX …
  - ESI
  - EDI

Instrucción que inicia la system call: **int 80h**

Pasaje de parámetros en x86 64bit

- http://blog.rchapman.org/posts/Linux_System_Call_Table_for_x86_64/
  - EAX lleva el numero de syscall que se desea ejecutar
  - RDI lleva el primer parámetro
  - RSI lleva el segundo parámetro
  - RDX …
  - R10
  - R8
  - R9

Instrucción que inicia la system call: **syscall**

Hello world en x86 32 bit

```
NAME
       write - write to a file descriptor

SYNOPSIS
       #include <unistd.h>

       ssize_t write(int fd, const void *buf, size_t count);

DESCRIPTION
       write()  writes  up  to  count b
       file referred to by the file des
```

```
# 32-bit system call numbers and entr
#
# The format is:
# <number> <abi> <name> <entry point>
#
# The abi is always "i386" for this f
#
0       i386    restart_syscall
1       i386    exit
2       i386    fork
3       i386    read
4       i386    write
5       i386    open
6       i386    close
```

```
NAME
       exit - cause normal process term

SYNOPSIS
       #include <stdlib.h>

       void exit(int status);

DESCRIPTION
       The  exit() function causes normal process termination and the value of
       status & 0377 is returned to the parent (see wait(2)).
```

```
_start:

    ; sys_write(stdout, message, length)

    mov eax, 4          ; sys_write syscall
    mov ebx, 1          ; stdout
    mov ecx, message    ; message address
    mov edx, 14         ; message string length
    int 80h

    ; sys_exit(return_code)

    mov eax, 1          ; sys_exit syscall
    mov ebx, 0          ; return 0 (success)
    int 80h

section .data
    message: db 'Hello, world!',0x0A     ; message and newline
```

Hello world en x86 64 bit

```
NAME
       write - write to a file descriptor

SYNOPSIS
       #include <unistd.h>

       ssize_t write(int fd, const void *buf, size_t count);

DESCRIPTION
       write()  writes  up  to  count bytes from the buffer pointed buf to the
       file referred to by the file descriptor fd.
```

- write → syscall número 1
- exit → syscall número 60

```
; sys_write(stdout, message, length)

mov rax, 1  ; sys_write
mov rdi, 1  ; stdout
mov rsi, message ; message address
mov rdx, length ; message string length
syscall

; sys_exit(return_code)

mov rax, 60  ; sys_exit
mov rdi, 0  ; return 0 (success)
syscall

section .data
 message: db 'Hello, world!',0x0A ; message and newline
 length: equ 14 ;
```

```
NAME
       exit - cause normal process term

SYNOPSIS
       #include <stdlib.h>

       void exit(int status);

DESCRIPTION
       The  exit() function causes normal process termination and the value of
       status & 0377 is returned to the parent (see wait(2)).
```

Resumen

- Los manuales del sistema indican los parámetros necesarios para activar una system call

```
NAME
       write - write to a file descriptor

SYNOPSIS
       #include <unistd.h>

       ssize_t write(int fd, const void *buf, size_t count);

DESCRIPTION
       write()  writes  up  to  count bytes from the buffer pointed buf to the
       file referred to by the file descriptor fd.
```

- Dependiendo la arquitectura, cambiará:
  - el número de system call utilizado para realizar una función determinada
  - La forma de pasar los parámetros al kenel

Resumen

- Los procesadores 32 bit y 64 bits usan un esquema de registros diferentes.

- Los procesadores 32 bit y 64 bits usan una instrucción distinta para activar las systemcalls:
  - 32 bits: **int 80h**
  - 64 bits: **syscall**

Referencias

Como programar un “hello world” en x86 32bit y 64bit

- http://shmaxgoods.blogspot.com.ar/2013/09/assembly-hello-world-in-linux.html
- https://stackoverflow.com/questions/19743373/linux-x86-64-hello-world-and-register-usage-for-parameters

Mas información sobre formas de pasar parametros a una syscall

- https://github.com/torvalds/linux
- https://github.com/torvalds/linux/blob/master/arch/x86/entry/syscalls/syscall_32.tbl
- https://github.com/torvalds/linux/blob/master/arch/x86/entry/syscalls/syscall_64.tbl
- https://syscalls.kernelgrok.com/
- http://blog.rchapman.org/posts/Linux_System_Call_Table_for_x86_64/
- http://www.int80h.org/bsdasm/#system-calls