

Conceptos de arquitectura de computadores

Adaptación de múltiples referencias bibliográficas

Juan Felipe Muñoz Fernández

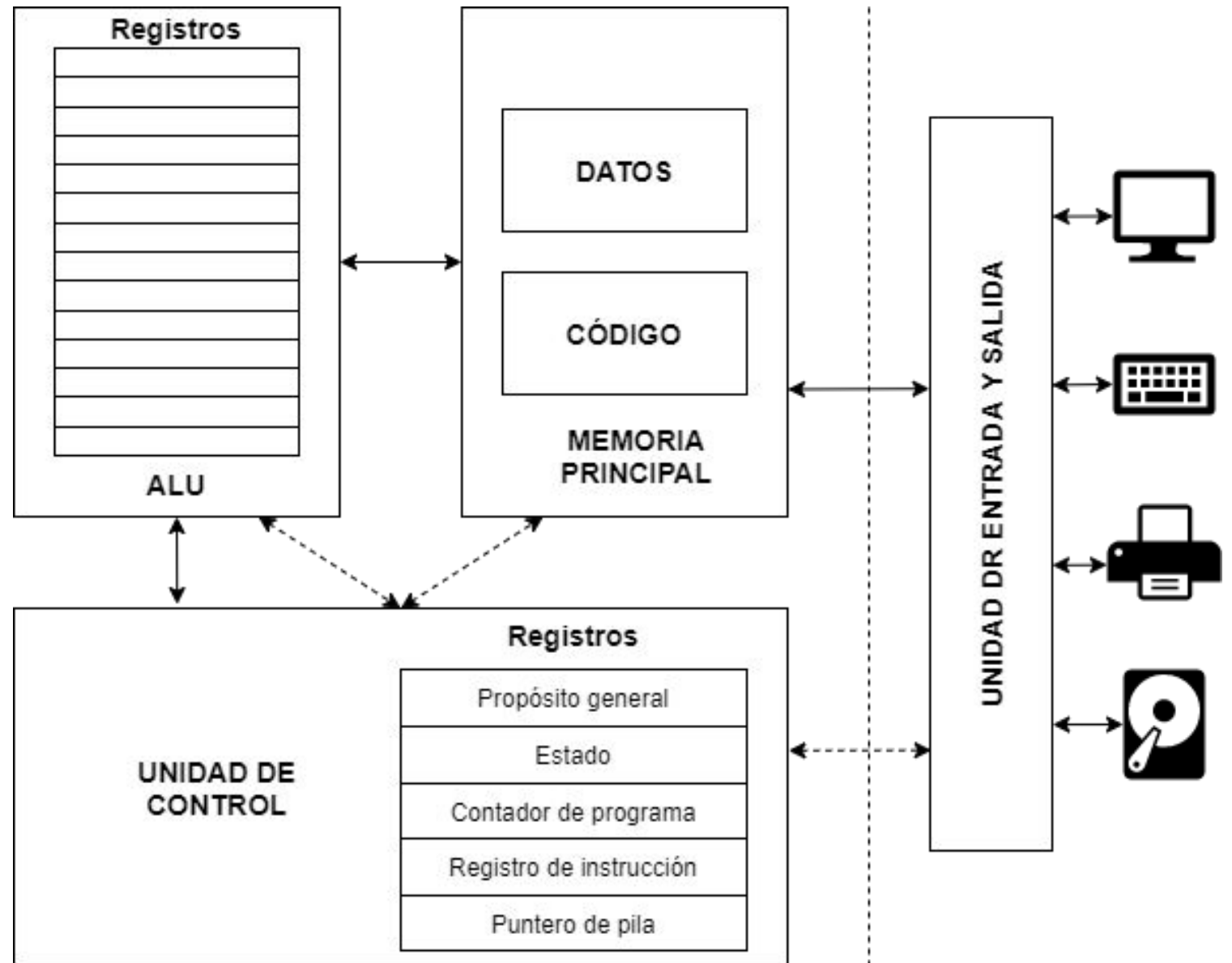
Arquitectura John von Neumman



John von Neumman
Fuente: Wikipedia

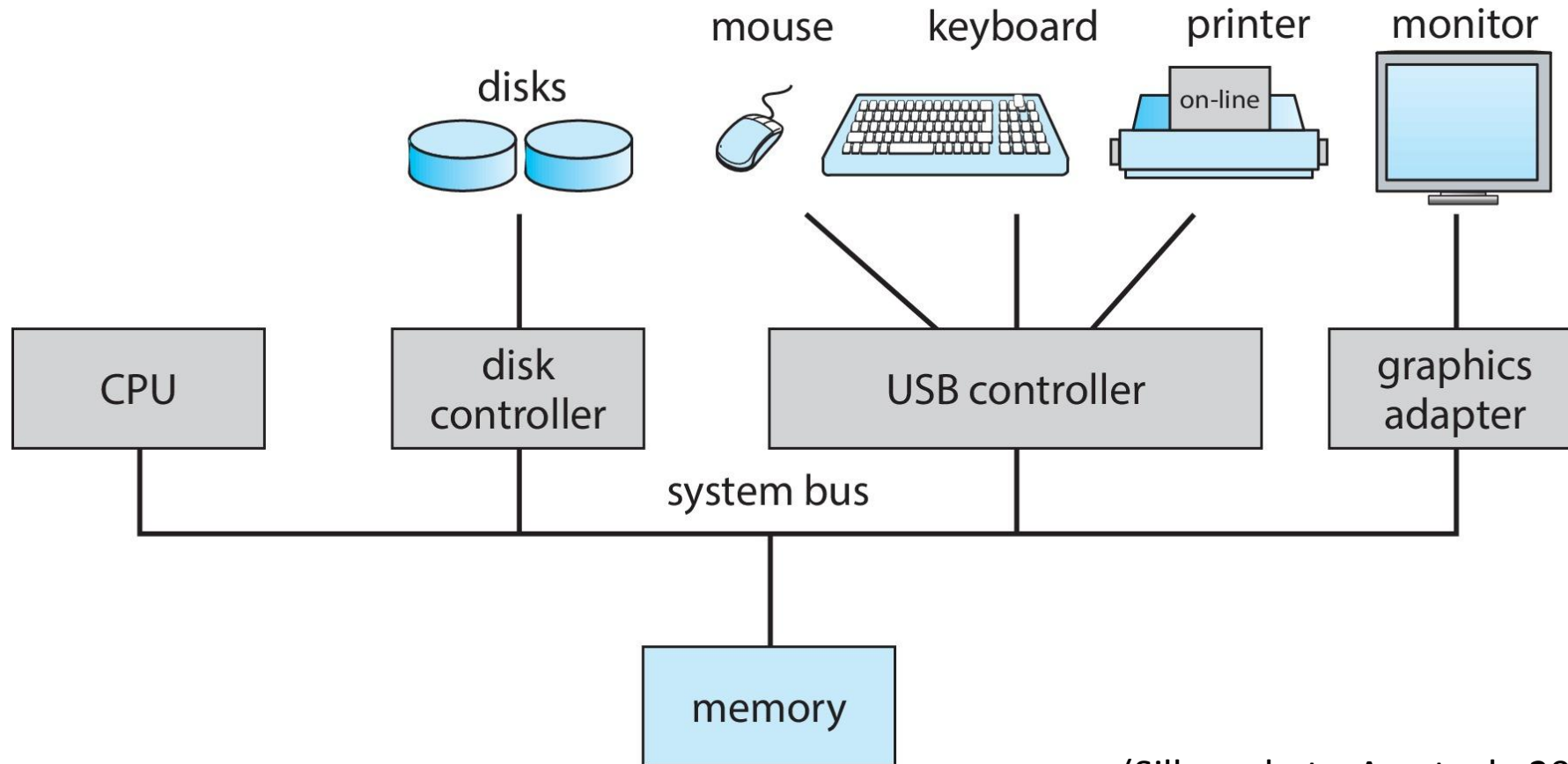
- Un computador se compone de:
 - Unidad de control
 - Memoria principal
 - Los dispositivos de E/S o periféricos
 - Unidad Aritmético – Lógica
- Todos los elementos conectados a través de un bus común
 - Bus del sistema o *System bus*
 - Bus de direcciones
 - Bus de datos
 - Bus de control

Arquitectura John von Neumman

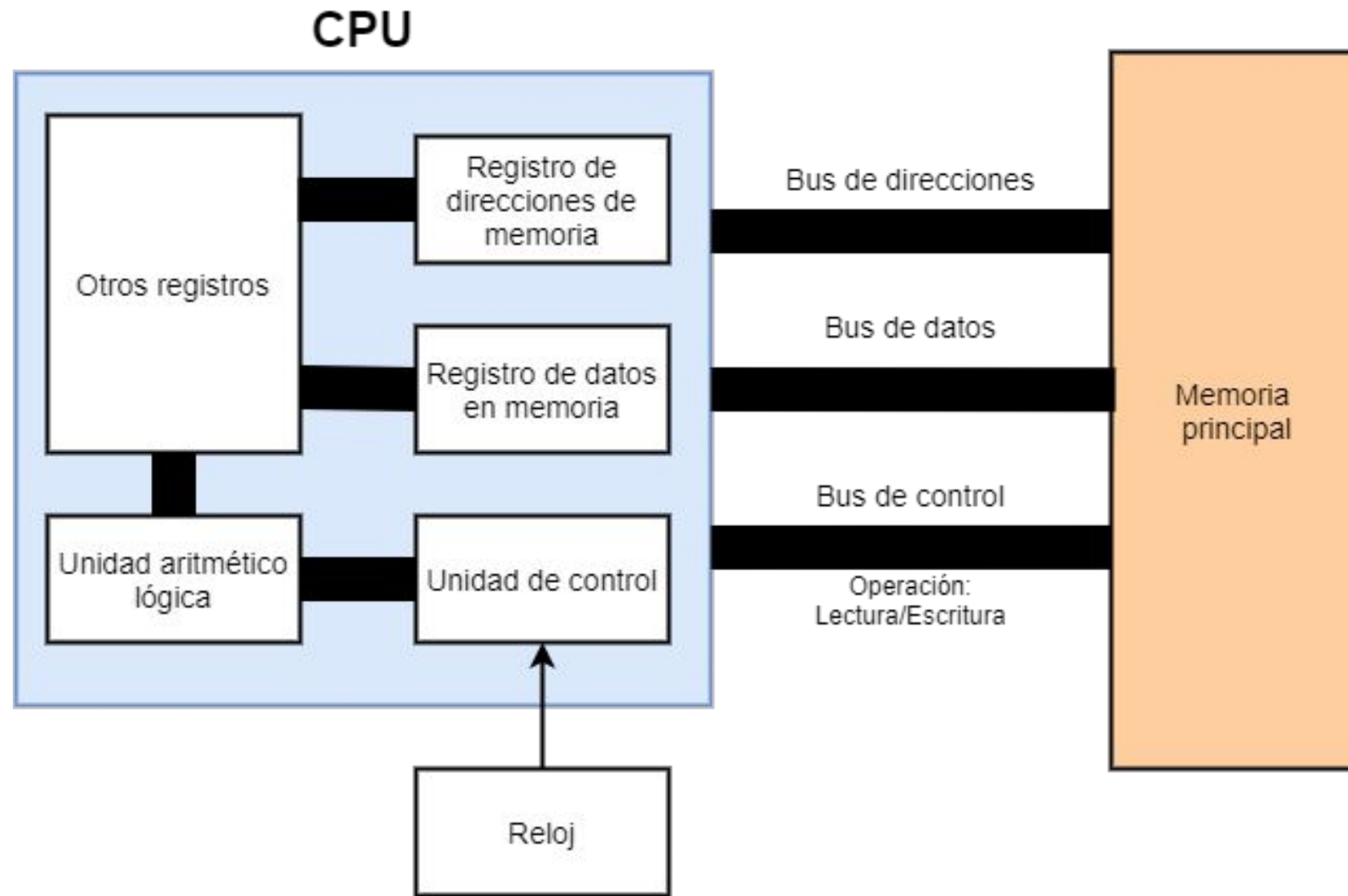


Con modificaciones tomada de:
(Carretero Pérez, J. et. al., 2001)

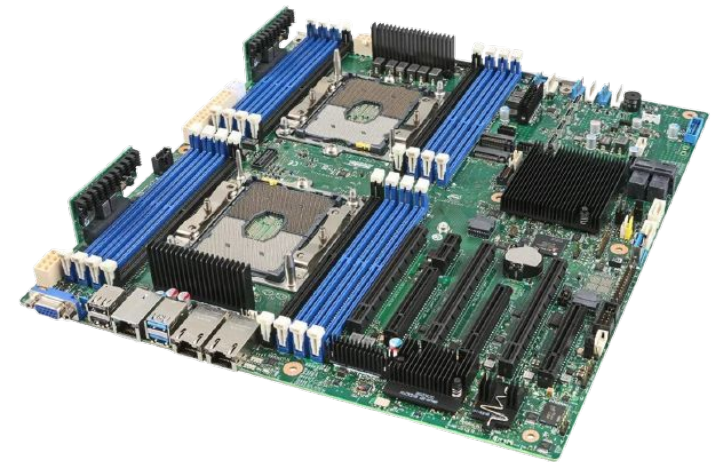
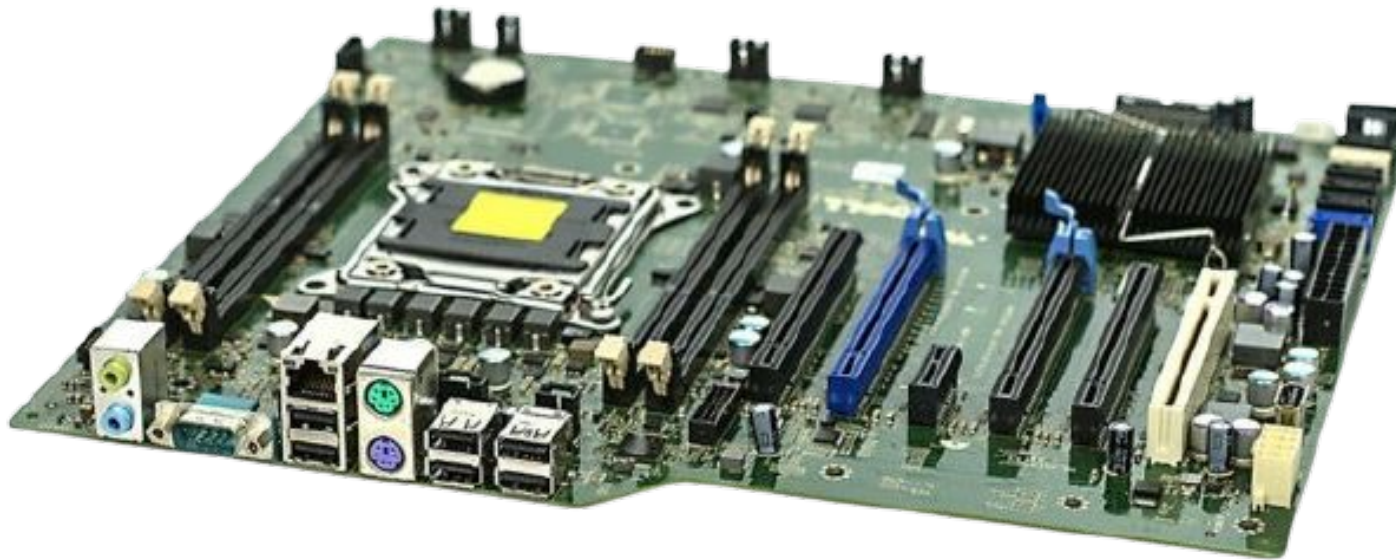
Arquitectura John von Neumman



(Silberschatz, A., et. al., 2018)



Arquitectura Jhon von Neumman

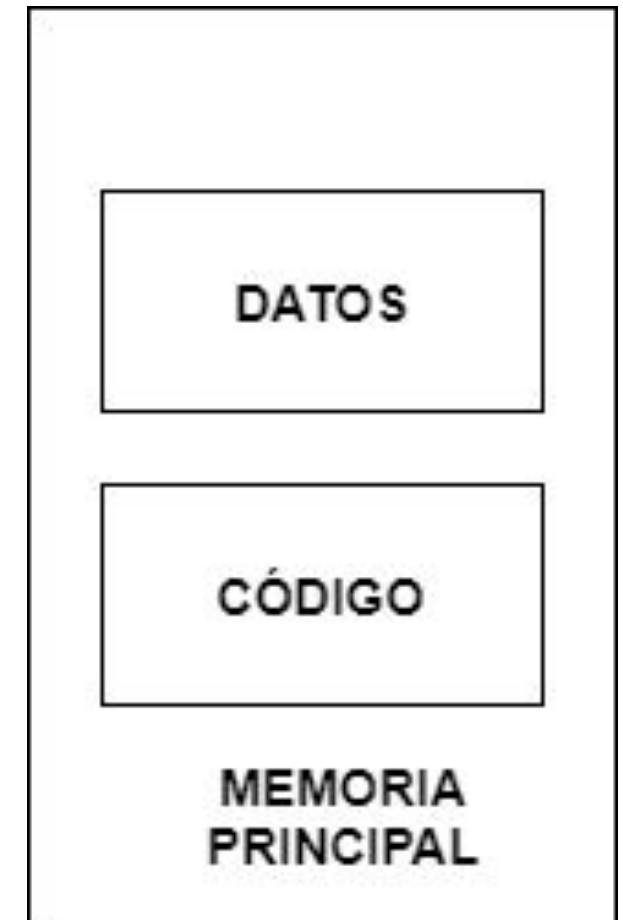


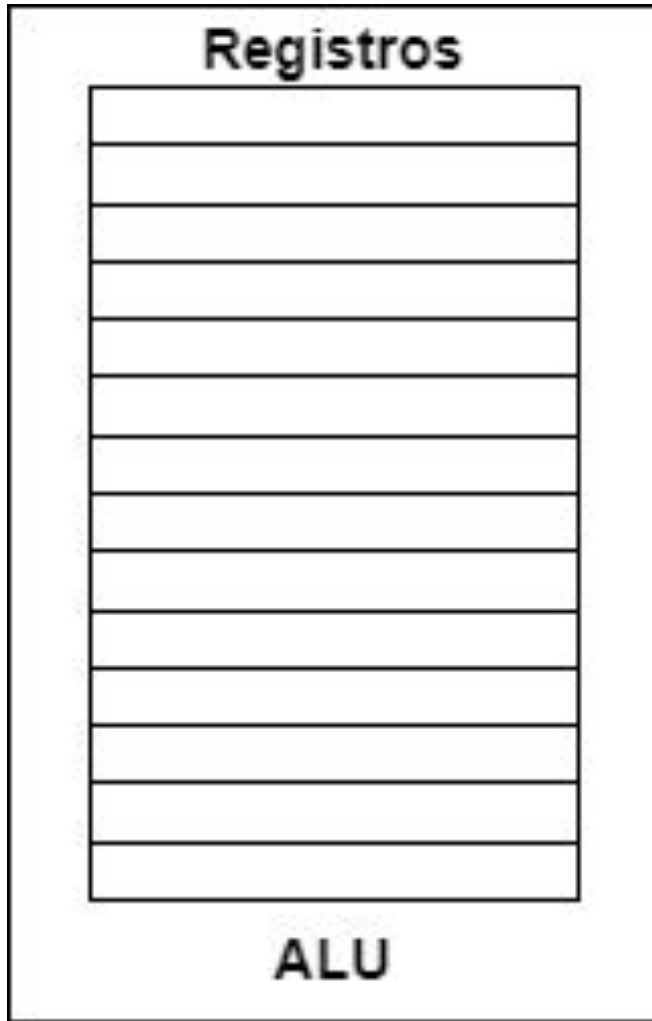
Memoria principal

Demo x64dbg
Demo Visual
Studio

```
#include <stdio.h>
int main()
{
    printf("Hello World!");
    return 0;
}
```

- Compuesta por
 - RAM (*Random Access Memory*)
 - ROM (*Read Only Memory*)
- Contiene
 - Datos de los programas
 - Código de los programas
 - Resultados de las operaciones
- Se accede a través de las direcciones
 - Estructura de “celdas”
 - Cada celda (byte) tiene una dirección
 - Usualmente palabras de 4 u 8 bytes
 - Un arreglo de muchos bytes





Unidad Aritmético – Lógica

- Realiza operaciones aritméticas y lógicas
 - Uno o dos operandos
 - Operandos almacenados en registros o en memoria principal
 - Resultados se almacenan en registros o en memoria principal

Demo x64dbg
Demo Visual
Studio

```
#include <stdio.h>
int main()
{
    short a = 1;
    short r = 0;
    r = a + 3;
    printf("%d", r);
    return 0;
}
```


Unidad de control

- Funciones

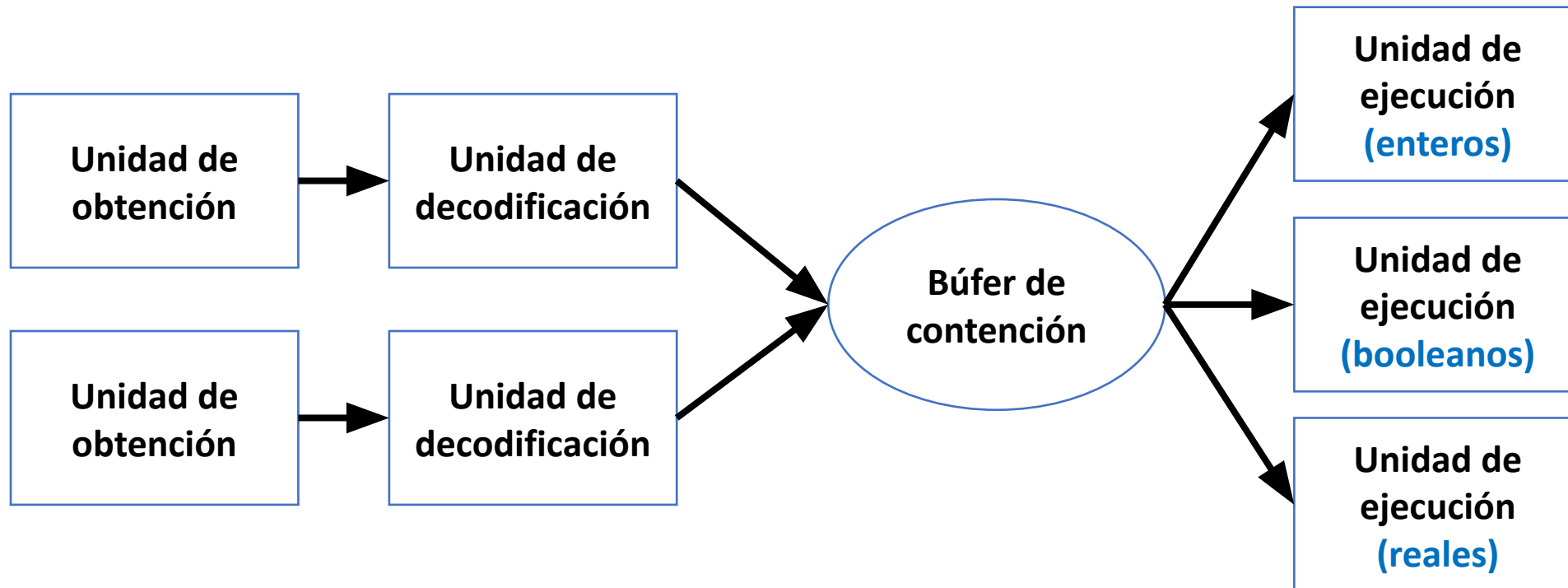
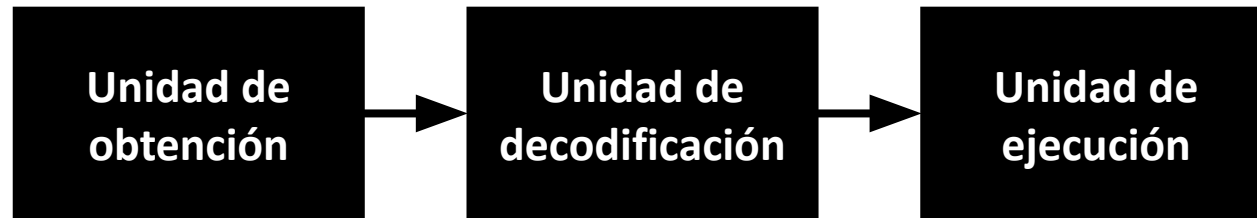
- Lee de memoria instrucciones que forman el programa
- Interpreta instrucción leída
- Lee datos referenciados por cada instrucción
- Ejecuta cada instrucción
- Almacena el resultado de operaciones
- Sincroniza todo el sistema

- Se mantiene en una secuencia lineal (instrucciones consecutivas) de alta velocidad

1. Lectura memoria principal de la instrucción apuntada por *Instruction Pointer*
2. Incremento del *Instruction Pointer*
3. Ejecución de la instrucción



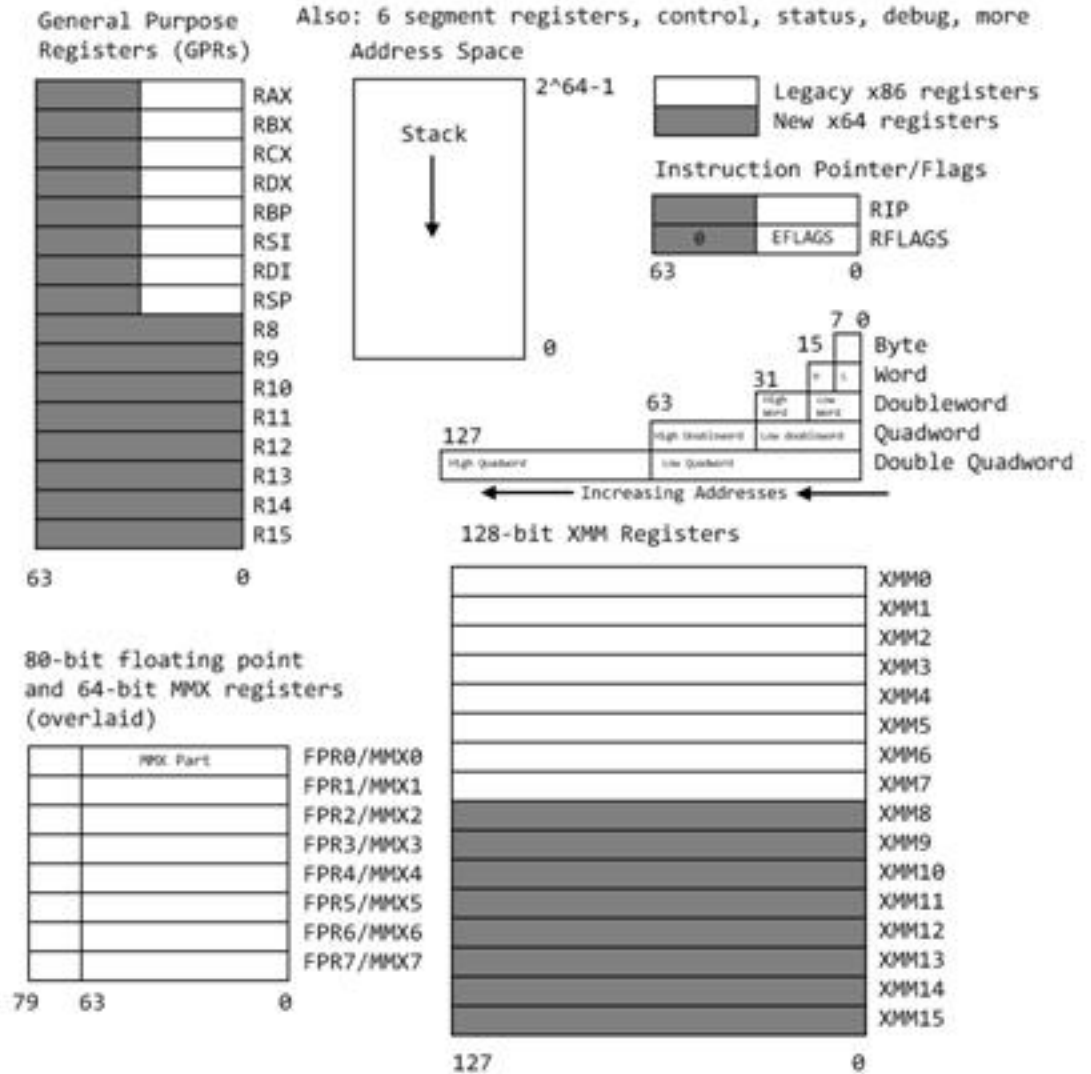
Ciclo obtención, decodificación, ejecución



Unidad de control

```
#include <stdio.h>
int main()
{
    short a;
    short b = 2;
    a = 1;
    if (a > b)
        printf("True");
    else
        printf("False");
}
```

Demo x64dbg
Demo Visual
Studio



Registros en CPU

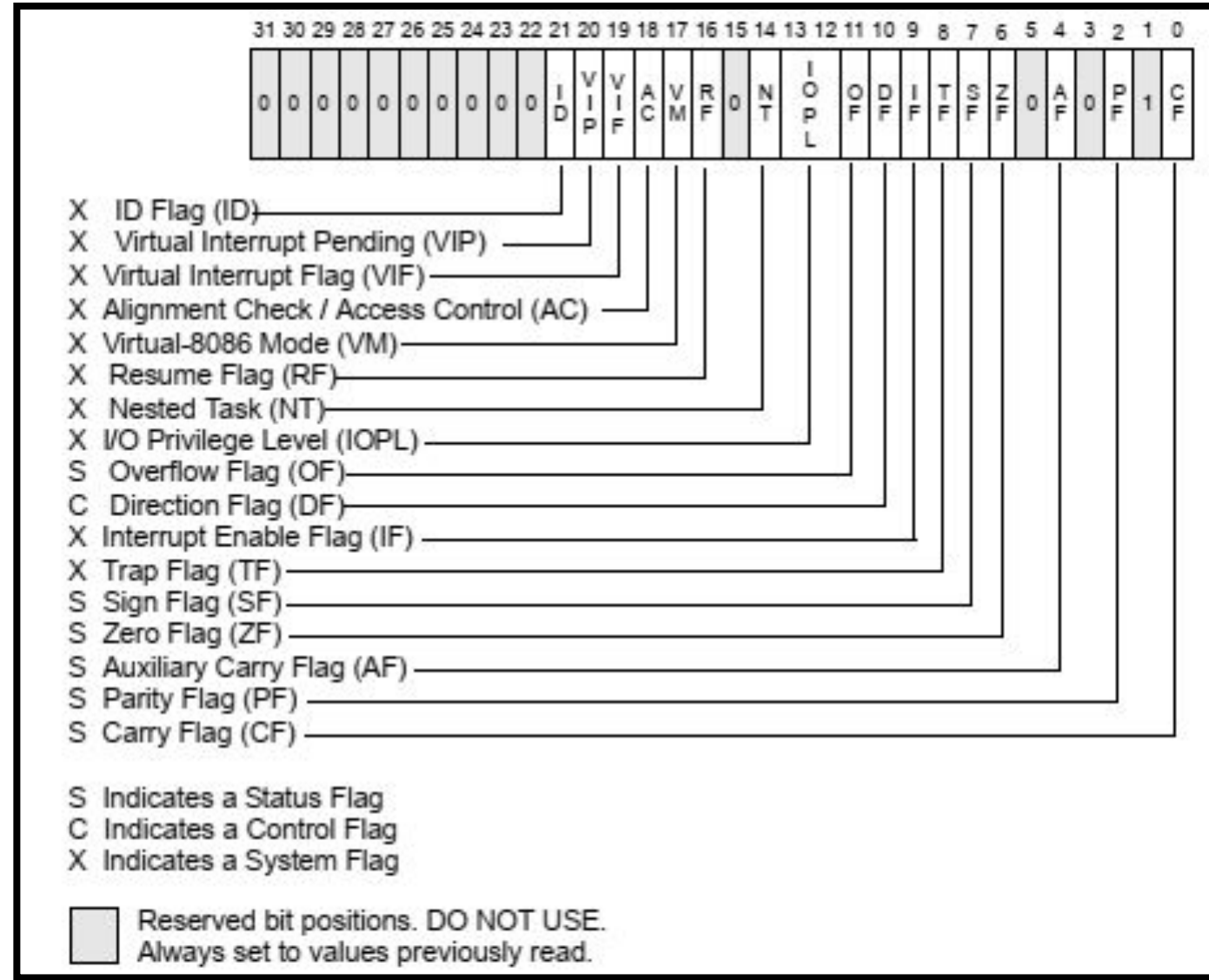
- Registros de direcciones de memoria
 - *MAR (Memory Address Registers)*
 - Mantienen la dirección de una ubicación en memoria
- Registros de datos en memoria
 - *MDR (Memory Data Registers)*
 - Mantienen los datos que se acaban de leer o escribir a la memoria
- Contador del programa
 - Mantiene la dirección de la siguiente instrucción a ejecutar
- Registro de instrucción
 - Mantiene la dirección de la instrucción que está siendo ejecutada actualmente
- Registros de uso general
 - Usados por los programadores

Registros en CPU x64

- Registros de propósito general
 - RAX, RBX, RCX, RDX
 - +8 adicionales: R8 a R15
- Registros apuntadores
 - RBP, RSP
- Registros índice
 - RSI, RDI
- Registros de segmentos
 - CS, DS, ES, SS, FS, GS
- Apuntador de instrucciones
 - RIP
 - CS:RIP □ Próxima instrucción a ejecutar
- Registro de estado (R/EFLAGS)
 - Mantiene el registro del estado de la ejecución de instrucciones

Registro de estado

- No todos los bits son modificables por el programador.
- Algunos requieren **ejecución privilegiada de instrucciones**.
- Mantiene información del estado del sistema



Registros CPU x64

64-bit register	Lower 32 bits	Lower 16 bits	Lower 8 bits
RAX	EAX	AX	AL
RBX	EBX	BX	BL
RCX	ECX	CX	CL
RDY	EDX	DX	DL
RSI	ESI	SI	SIL
RDI	EDI	DI	DIL
RBP	EBP	BP	BPL
RSP	ESP	SP	SPL

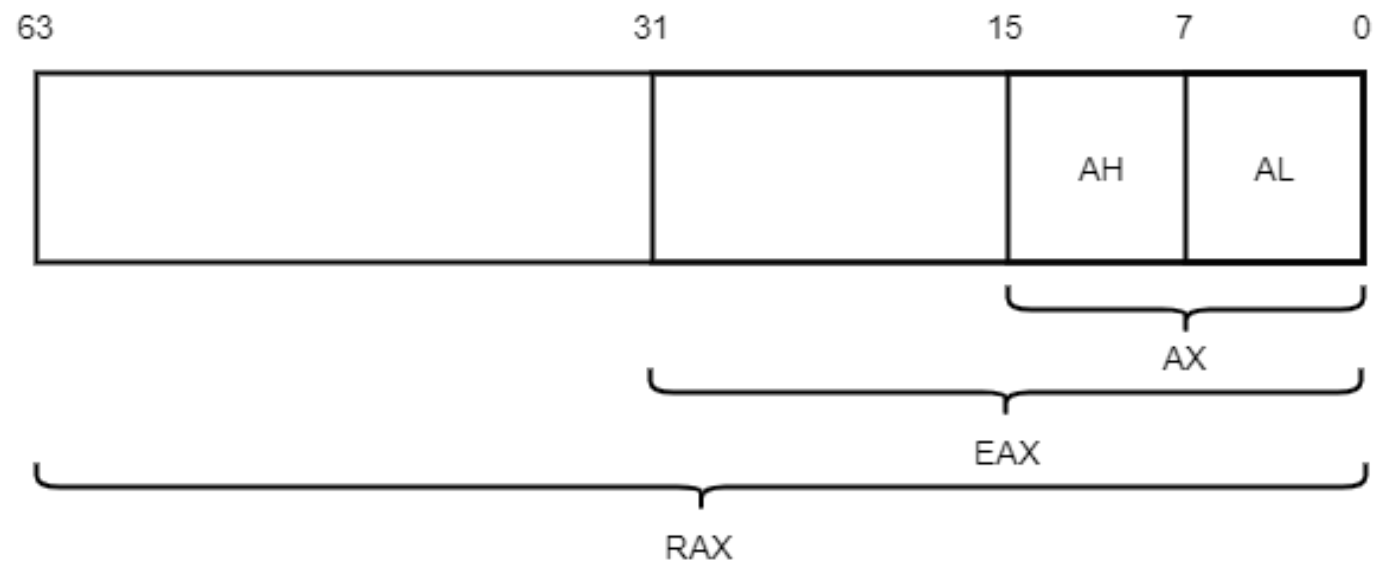
Fuente: <https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/x64-architecture>

Registros CPU x64

64-bit register	Lower 32 bits	Lower 16 bits	Lower 8 bits
R8	R8D	R8W	R8B
R9	R9D	R9W	R9B
R10	R10D	R10W	R10B
R11	R11D	R11W	R11B
R12	R12D	R12W	R12B
R13	R13D	R13W	R13B
R14	R14D	R14W	R14B
R15	R15D	R15W	R15B

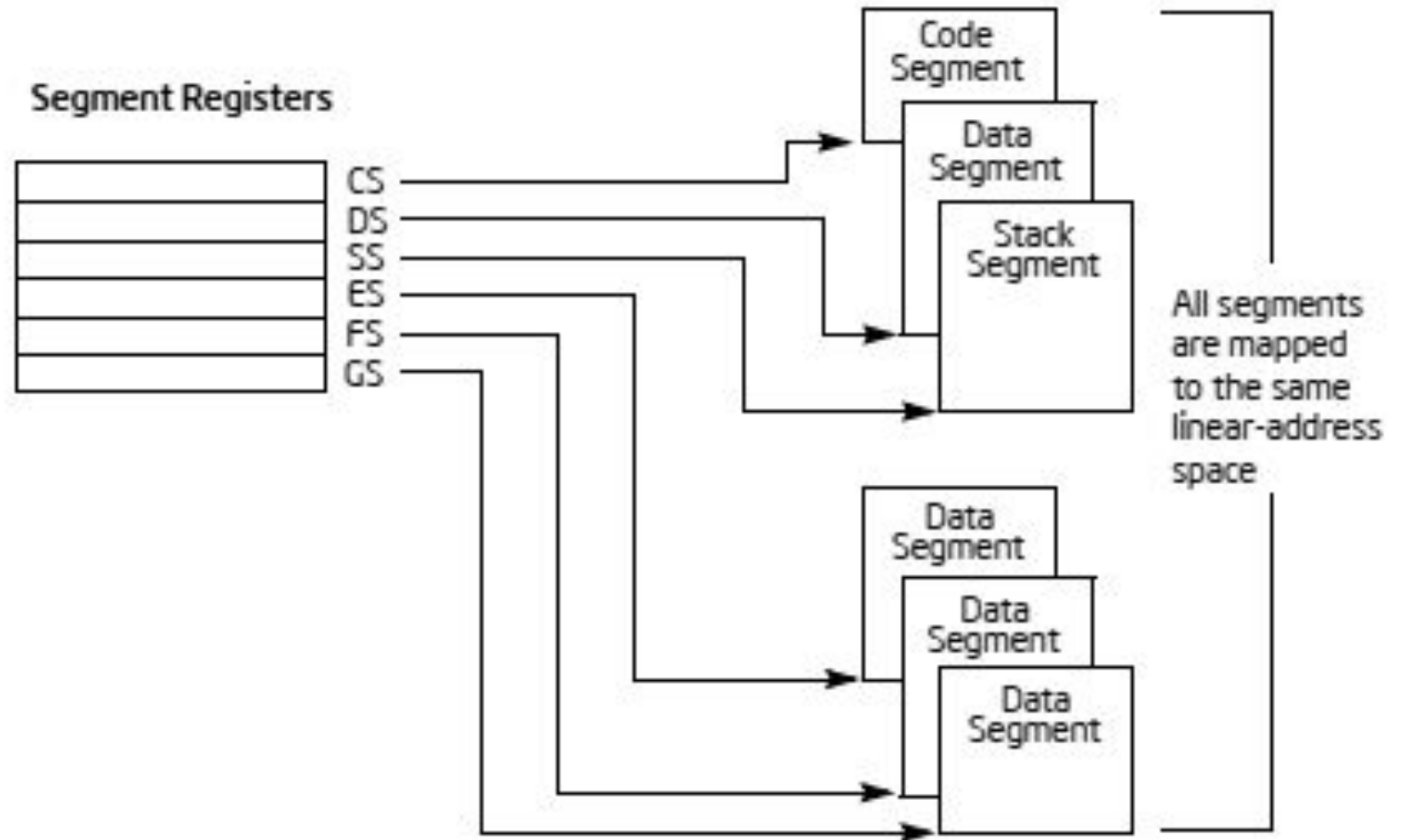
Fuente: <https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/x64-architecture>

Registros CPU x64



Registros de segmentos

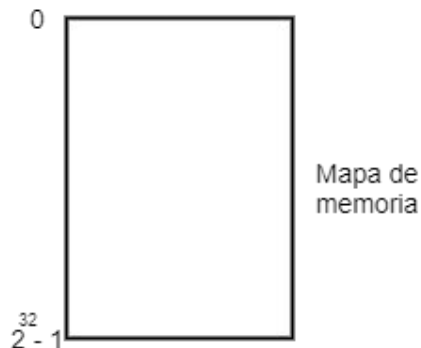
Fuente: Intel® 64 and IA-32
Architectures. Software Developer's
Manual. Volume 1: Basic Architecture



Unidad de entrada/salida

- Realiza la transferencia de información entre:
 - Memoria principal y periféricos
 - Registros y periféricos
 - Acceso por interrupciones de E/S
- También se puede acceder a través de la CPU a los dispositivos
 - E/S programada
 - Procesador ejecuta un programa de E/S
 - No hay concurrencia entre procesador y E/S
- También se puede hacer de forma independiente (DMA)
 - *Direct Memory Access*
 - Puede existir concurrencia de procesos
 - Responsable el controlador del dispositivo

Modelo de programación



Con modificaciones tomada de:
(Carretero Pérez, J. et. al., 2001)

Modelo de programación

- Almacenamiento
 - **Visibles a las instrucciones**
 - Registros generales (RAX, EAX, RBX, EBX,...,etc.)
 - *Instruction Pointer* (CS: IP)
 - Puntero de pila (SS)
 - Estado (R/EFLAGS)
 - Memoria principal
 - Mapa de E/S
- Secuencia de funcionamiento
 - Define cómo se ejecutan las instrucciones

Modelo de programación

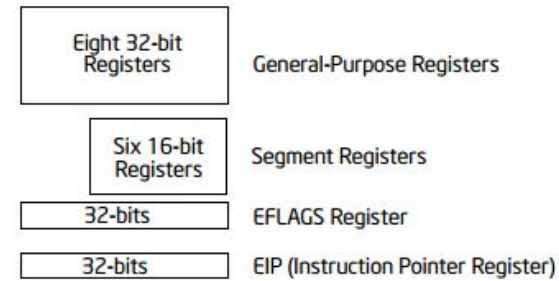
- Juego de instrucciones
 - Operaciones que es capaz de hacer la CPU
 - Modos de direccionamiento
 - Destino, fuente: `MOV AX, BX`
 - Modalidad inmediata: `MOV AX, 5`
 - Modalidad directa: `MOV AX, [200H]`
 - Modalidad indirecta: `MOV AX, SI:[200H]. MOV AX, DI:[200H]`
 - Modalidad base relativa: `MOV AX, [BP]. MOV AX, [BP+SI+5]`

Modelo de programación Intel x86

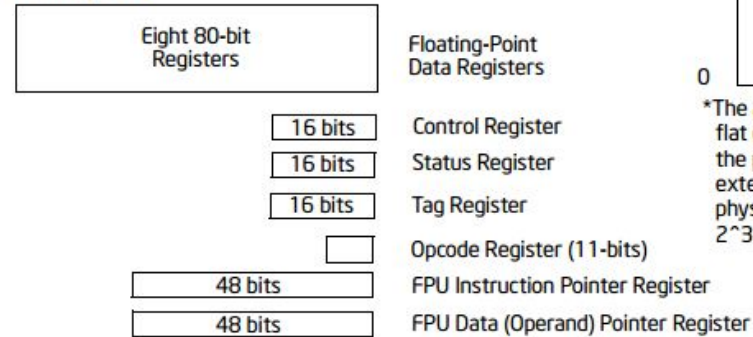
Fuente: Intel® 64 and IA-32 Architectures. Software Developer's Manual. Volume 1: Basic Architecture

Operaciones SIMD
Una instrucción múltiples datos
Paralelismo a nivel de datos

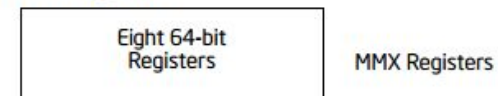
Basic Program Execution Registers



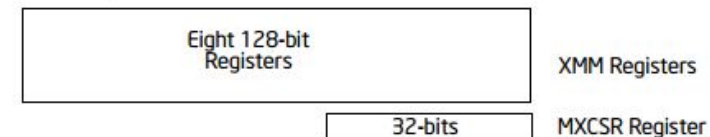
FPU Registers



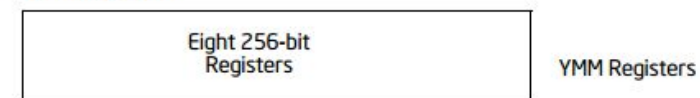
MMX Registers



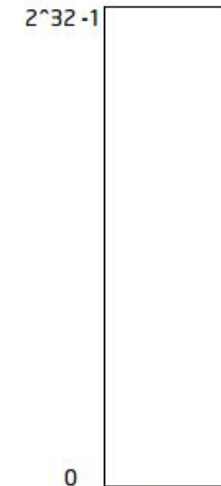
XMM Registers



YMM Registers

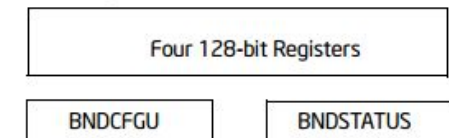


Address Space*



*The address space can be flat or segmented. Using the physical address extension mechanism, a physical address space of $2^{36} - 1$ can be addressed.

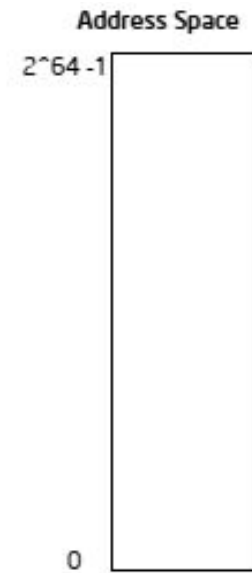
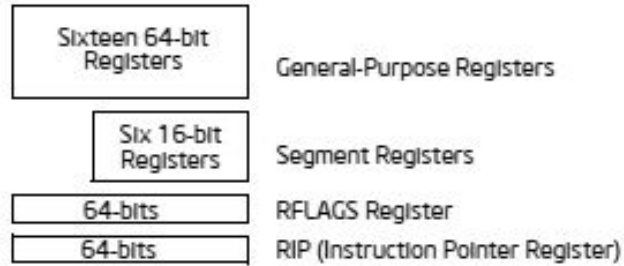
Bounds Registers



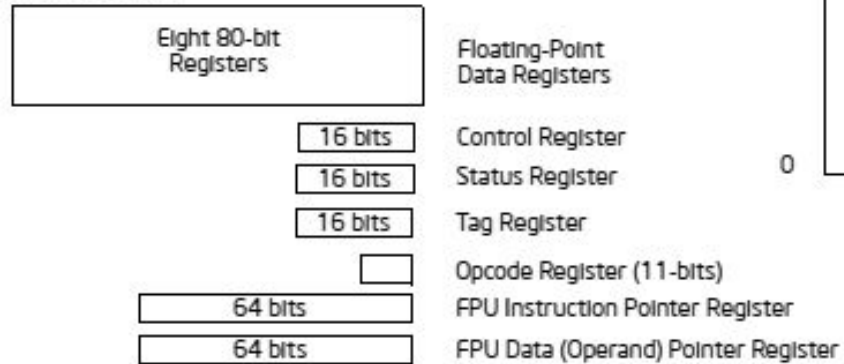
Modelo de programación Intel x64

Fuente: Intel® 64 and IA-32 Architectures. Software Developer's Manual. Volume 1: Basic Architecture

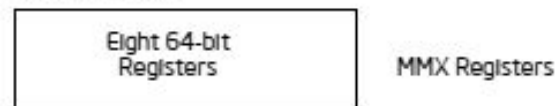
Basic Program Execution Registers



FPU Registers



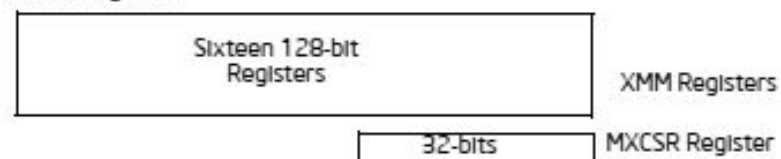
MMX Registers



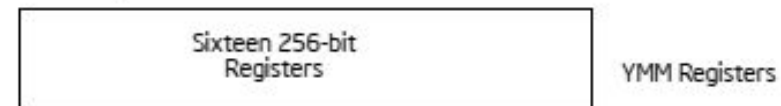
Bounds Registers



XMM Registers



YMM Registers



Modelo de programación

- Lecturas sugeridas
 - Intel® 64 and IA-32 Architectures. Software Developer's Manual. Volume 1: Basic Architecture
- Sitio interesante con documentos sobre arquitectura de procesadores Intel
 - <https://software.intel.com/content/www/us/en/develop/articles/intel-sdm.html>

Niveles de ejecución

- **Nivel de usuario (ring 3)**

- Disponible un subconjunto de todo el juego de instrucciones
- Demás instrucciones prohibidas
- Prohibido el acceso a ciertos registros
- Prohibido al acceso a determinadas zonas del mapa de memoria y del mapa de E/S.

- **Nivel de núcleo (ring 0)**

- No hay restricciones
- Kernel del sistema operativo
- Controladores de algunos dispositivos
- Depuradores tipo **SoftICE** y **WinDbg**

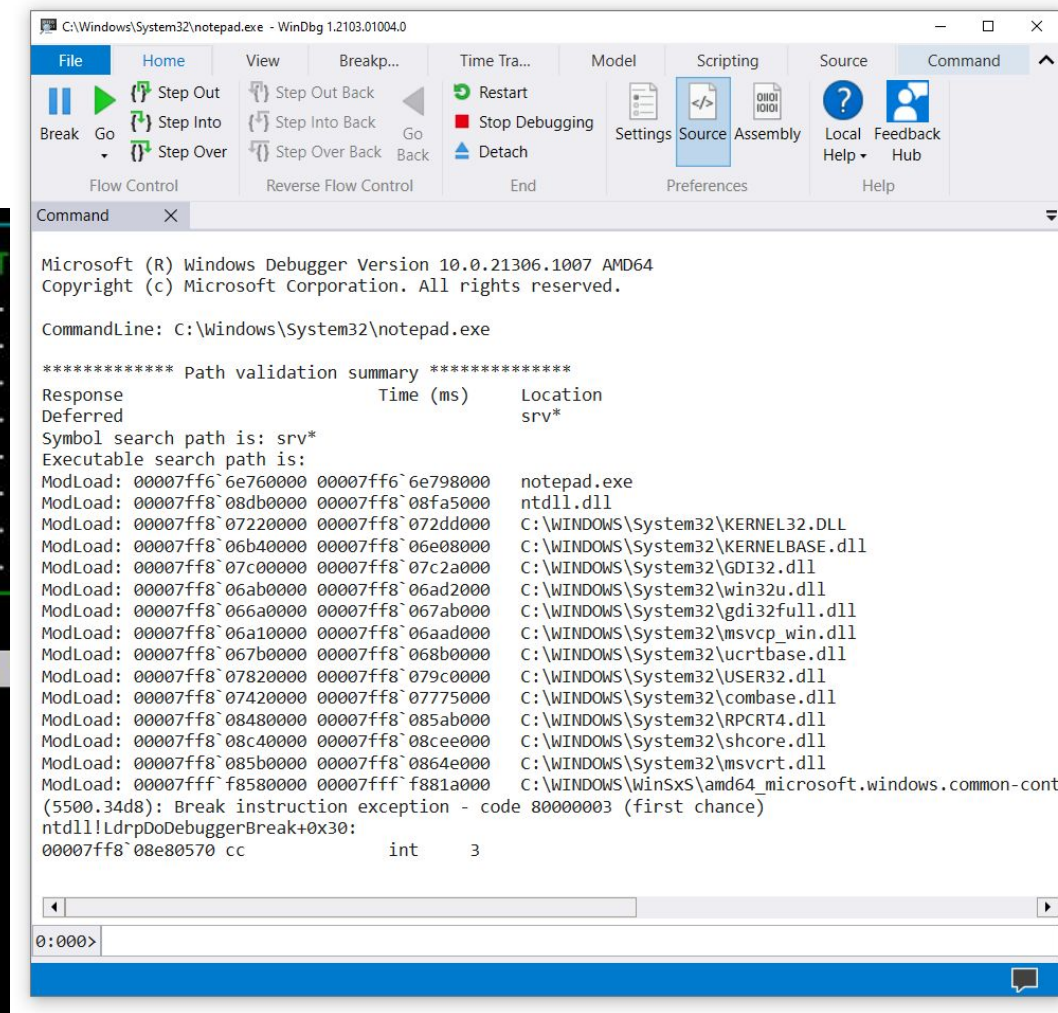
SoftICE y WinDbg

```
0010:00000000 ?? ?? ?? ?? ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ??
0010:00000010 ?? ?? ?? ?? ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ??
0010:00000020 ?? ?? ?? ?? ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ??
0010:00000030 ?? ?? ?? ?? ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ??
0010:00000040 ?? ?? ?? ?? ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ??
0010:00000050 ?? ?? ?? ?? ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ??
0010:00000060 ?? ?? ?? ?? ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ??
0010:00000070 ?? ?? ?? ?? ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ??

0008:B1F0A293 STI
==> B1F0A294 JMP ↑B1F0A294 (JUMP
0008:B1F0A296 INT
0008:B1F0A298 IRE
0008:B1F0A299 PUS
0008:B1F0A29A PUS
0008:B1F0A29B MOV B60
0008:B1F0A2A0 MOV [EBX],203D5344
0008:B1F0A2A6 ADD
0008:B1F0A2A9 MOV
0008:B1F0A2AC CAL
0008:B1F0A2B1 MOV
0008:B1F0A2B7 ADD
(PASSIVE)-KTEB(81B
: WD 8
:SOFTICE1.FREE.FR
Invalid command
```

- Copy
- Paste
- Copy&Paste
- Display
- Un-Assemble
- What
- Prev
- Reip
- Add Watch
- Break On Text
- Name

B60
[EBX],203D5344
28]
[EBX+04],0020
NTice!.text+2F13

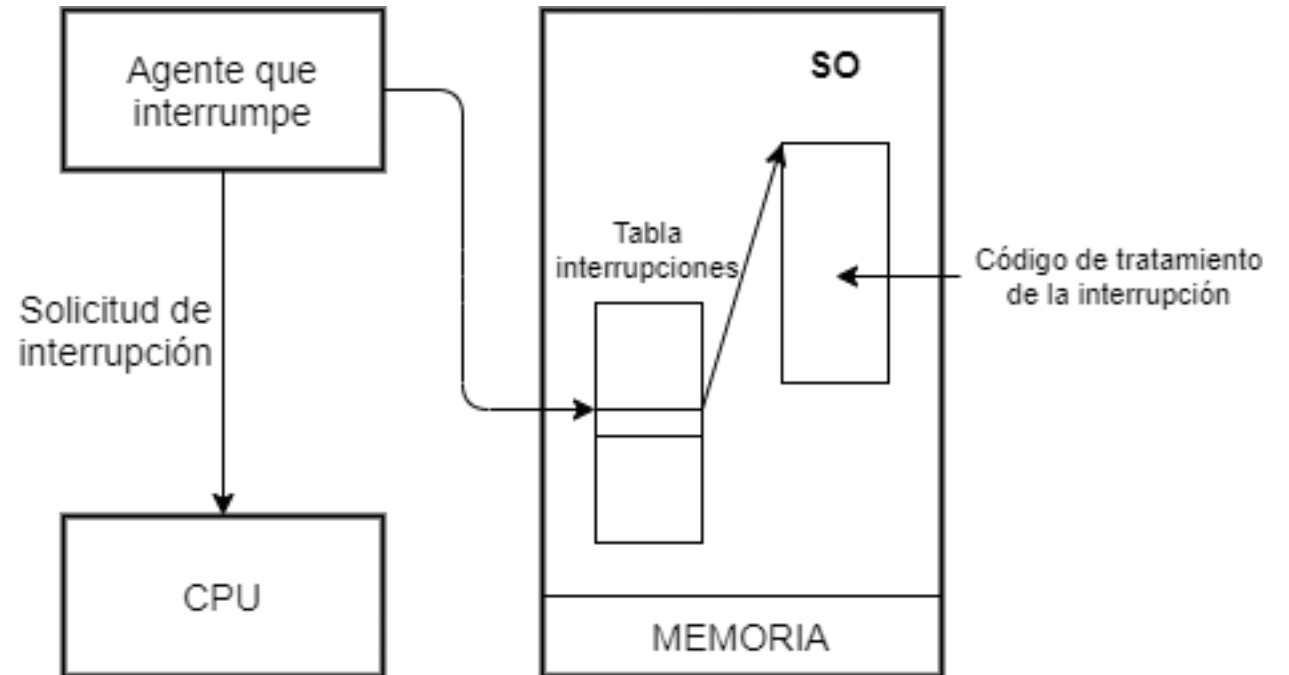


Interrupciones

- Se envía una señal a la CPU solicitando atención, cuando se atiende
 - Guardar registros: estado, generales, *Instruction Pointer* (IP)
 - Elevar nivel de ejecución a nivel de núcleo.
 - Carga en IP instrucción de tratamiento de interrupción
- El agente que interrumpe
 - Suministra vector de interrupción
 - Dirección de comienzo del programa que atiende (tratamiento) de la interrupción
- Mecanismo que permite al S.O manejar eventos asíncronos

Interrupciones

- Por seguridad en SO
 - Tabla de interrupciones
 - Código de tratamiento de la interrupción
 - Evitar que programas de otros usuarios hagan daños.



Con modificaciones tomada de:
(Carretero Pérez, J. et. al., 2001)

Interrupciones

- Causas

- Excepciones del programa
 - Desbordamiento
 - División por cero
 - Operaciones inválidas
- Interrupciones de reloj
- Interrupciones de E/S
 - Controladores de dispositivos
- Interrupciones de hardware
 - Solicitud de atención a un dispositivo

- Causas

- Excepciones de hardware
 - Errores de paridad
 - Corte de suministro eléctrico
- **Instrucciones TRAP para solicitar servicios al SO**
 - Pasa ejecución a modo Kernel.

INT 21 en DOS (interrupción de software)

```
org 0x100
```

```
mov dx, msg
```

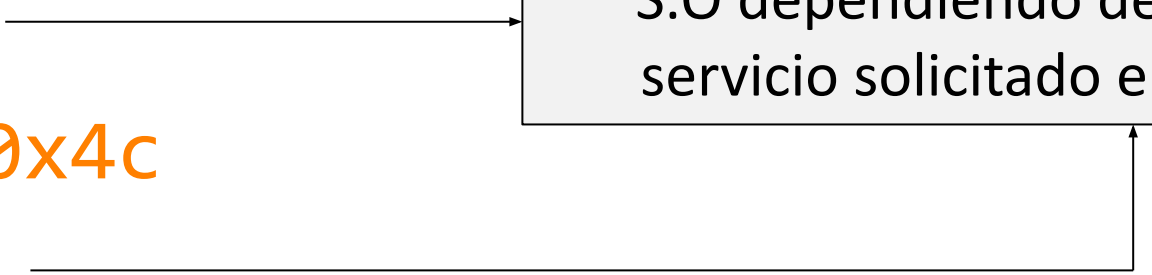
```
mov ah, 9
```

```
int 0x21
```

```
mov ah, 0x4c
```

```
int 0x21
```

```
msg db 'Hello, World!', 0x0d, 0x0a, '$'
```



Mismo número de interrupción.
S.O dependiendo de parámetros ubica
servicio solicitado en el código del S.O

Reloj

- Circuito oscilador con tres funciones básicas
 - Señal que gobierna el ritmo de ejecución de las instrucciones
 - Generador de interrupciones periódicas
 - Contador de fecha y hora
- Por ejemplo 1 Hz
 - El procesador se activa cada segundo
 - Activación completa
 - Realiza todas las tareas de las que es capaz cada segundo
- Por ejemplo 1 GHz
 - El procesador se activa 1.000.000.000 (mil millones) veces por segundo
- Por ejemplo Intel(R) Core(TM) i5-6200U CPU @ 2.30GHz
 - El procesador se activa 2.300 millones de veces por segundo

Reloj

- Generador de interrupciones periódicas
 - Interrupciones de reloj o ticks
 - El S.O entra ejecutar de forma sistemática cada cierto intervalo de tiempo
 - Evitar que un programa monopolice el uso de la CPU.
 - Permite que múltiples programas puedan ocupar la CPU en diferentes intervalos de tiempo.
- Contador de fecha y hora
 - El contador se incrementa en cada tick de reloj
 - Se toma de referencia una fecha para incrementar el contador
 - En computadores actuales se mantiene mediante circuito dedicado que requiere batería
 - En computadores más viejos había que indicar fecha y hora

Periféricos

- Registros del controlador incluidos en el mapa de E/S
 - Se acceden mediante instrucciones de E/S
- Registro de datos
 - Intercambio de datos
 - Carga de datos leídos desde el dispositivo
 - Extracción de datos para escritura en el dispositivo
- Registro de estado
 - Operación de lectura: registro de datos con nuevo valor
 - Operación de escritura: necesita datos en el registro de datos
 - Indicar errores
- Registro de control: indicar operaciones ha realizar

Periféricos



Con modificaciones tomada de:
(Carretero Pérez, J. et. al., 2001)

E/S y concurrencia

- E/S programada
 - Procesador ejecuta un programa de E/S
 - No hay concurrencia entre el procesador y la E/S
- E/S por interrupciones y DMA
 - El procesador no tiene que estar atendiendo la E/S
 - El procesador puede estar ejecutando otro programa mientras espera los datos
- Fases de una operación de E/S
 1. Envío de orden al periférico
 2. Lectura /escritura de datos
 3. Fin de operación

E/S y concurrencia

- Envío de orden al periférico
 - Escribir la orden en los registros del controlador del periférico
 - Se usan instrucciones de E/S
 - Las instrucciones se pueden ejecutar a la velocidad del procesador
- Transferencia de datos
 - Interviene el periférico (más lento que la CPU)
 - Espera activa (E/S programada)
 - En E/S programada esperar a que se lean/escriban TODOS los datos
 - Espera pasiva (E/S por interrupción)
 - El controlador genera una interrupción cuando tenga datos

```
n = 0
while n < m bytes
    read registro_control
    if (registro_control ==
dato_disponible)
        read registro_datos
        store datos en memoria principal
        n = n + 1
    endif
endwhile
```

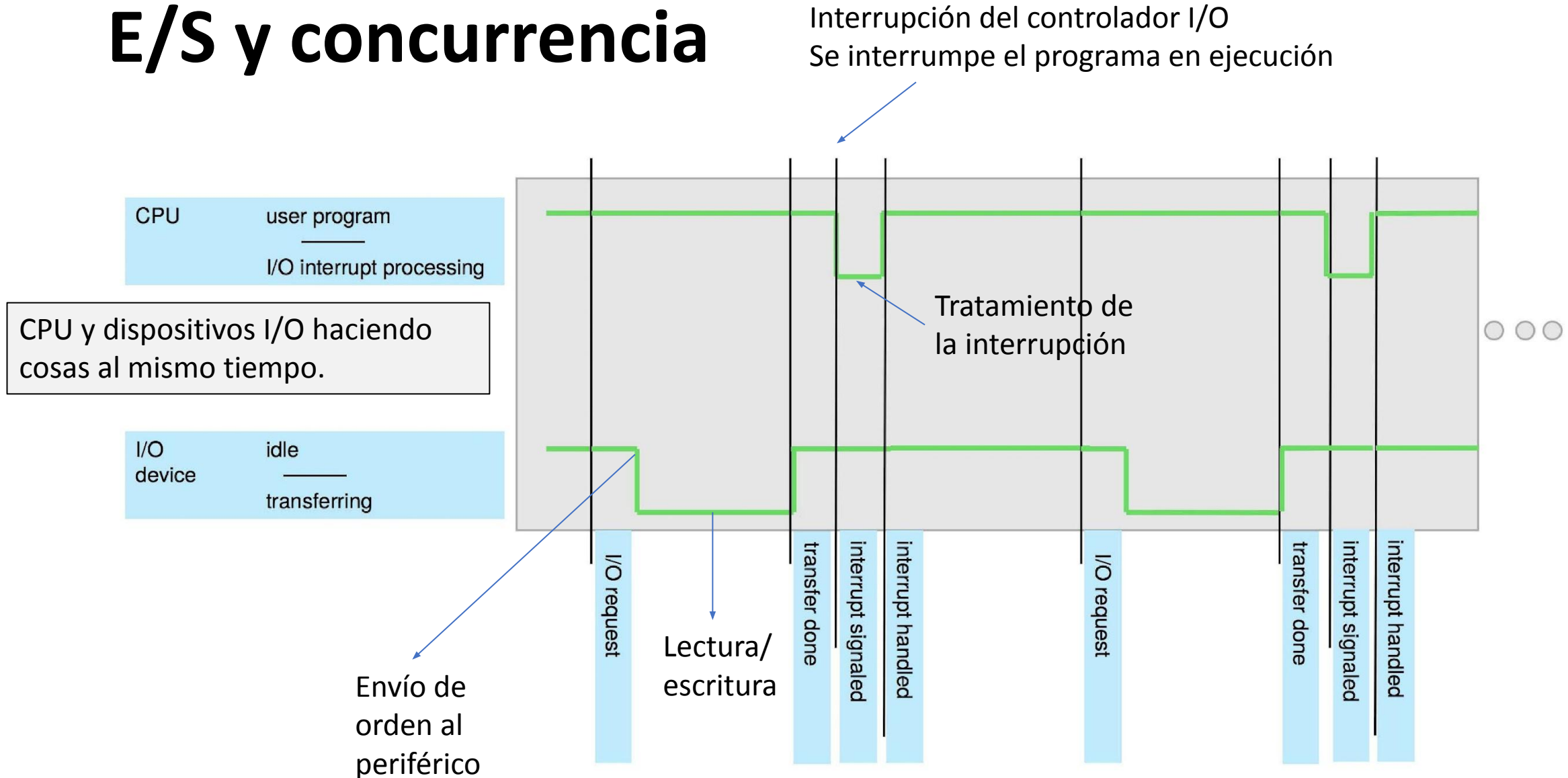
E/S y concurrencia

- Transferencia de datos
 - E/S por DMA
 - El controlador transfiere directamente los datos a la memoria principal
 - No se usa la CPU para la transferencia de datos
 - Cuando se termina la transferencia de los datos se genera una interrupción
 - Se avisa al procesador que se tienen los datos listos en memoria
 - Se requiere mayor inteligencia por parte del controlador
- Es función del S.O explotar la concurrencia de la E/S y el procesador
 - La idea es que el procesador esté ocupado haciendo cosas útiles
 - Se requieren mecanismos (diseños) eficientes para atender a dispositivos de E/S

E/S y concurrencia

	Envío de orden	Espera de dato	Transferencia de dato	Fin operación
E/S programada	Procesador	Procesador	Procesador	Procesador
E/S por interrupciones	Procesador	Controlador	Procesador	Procesador
E/S por DMA	Procesador	Controlador	Controlador	Procesador

E/S y concurrencia



(Silberschatz, A., et. al., 2018)

E/S y concurrencia

- **Momento 1:** Dispositivo I/O recibe una solicitud de I/O.
 - La solicitud se recibe a través del controlador del dispositivo.
 - Dispositivo I/O inicialmente está en estado *IDLE*
- **Momento 2:** Dispositivo I/O empieza a transferir datos
 - Se cambia de *IDLE* a *transferring*
- **Momento 3:** Dispositivo I/O termina de transferir datos y genera una interrupción
- **Momento 4:** CPU se interrumpe y atiende interrupción
- **Momento 5:** CPU continua su estado de ejecución del programa de usuario
- Se asume concurrencia. P. Ej.: E/S por interrupción o E/S por DMA.

E/S y concurrencia

- Hardware puede generar una interrupción en cualquier momento.
 - Señaliza a la CPU: ¡Hey! necesito que me atiendas
 - Señal que usualmente viaja por el bus del sistema
 - Línea física solicitud-interrupción.
- CPU lee línea de solicitud-interrupción después de cada instrucción.
 - Se lee el número de interrupción (índice del vector de interrupciones)
 - Se “salta” a la dirección de memoria indicada por el vector de interrupciones.
 - Se guarda el estado de ejecución antes de la interrupción: p. ej.: PUSH registros.
 - Se atiende interrupción: ejecución del programa de interrupción
 - Se restaura estado
 - Se retorna de la interrupción

Software usado para las demostraciones

- Visual Studio 2019 Community Edition
 - <https://visualstudio.microsoft.com/es/vs/community/>
- x64dbg
 - <https://x64dbg.com/>
- Visual Studio Code
 - <https://code.visualstudio.com/>
- Compilador C/C++ GNU para Windows
 - MinGW: <https://sourceforge.net/projects/mingw/>

Referencias

- Carretero Pérez, J., de Miguel Anasagasti, P., García Carballeira, F., & Pérez Costoya, F. (2001). *Sistemas Operativos. Una visión aplicada*. McGraw-Hill/Interamericana de España.
- Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating system concepts*. Wiley.

Introducción a los sistemas operativos

Adaptación de múltiples referencias bibliográficas

Juan Felipe Muñoz Fernández

Do you pine for the days
when men were men and
wrote their own device
drivers?

Torvalds, Linus (Oct 5, 1991)



Programar un controlador de Floppy Disk: PD765 (en palabras)

- Comandos
 - Leer datos
 - Escribir datos
 - Desplazar brazo del disco
 - Formatear pistas
 - Detectar dispositivo y unidades
 - Inicializar dispositivo y unidades
 - Recalibrar dispositivo y unidades
 - Otros
- En total 16 comandos para programar



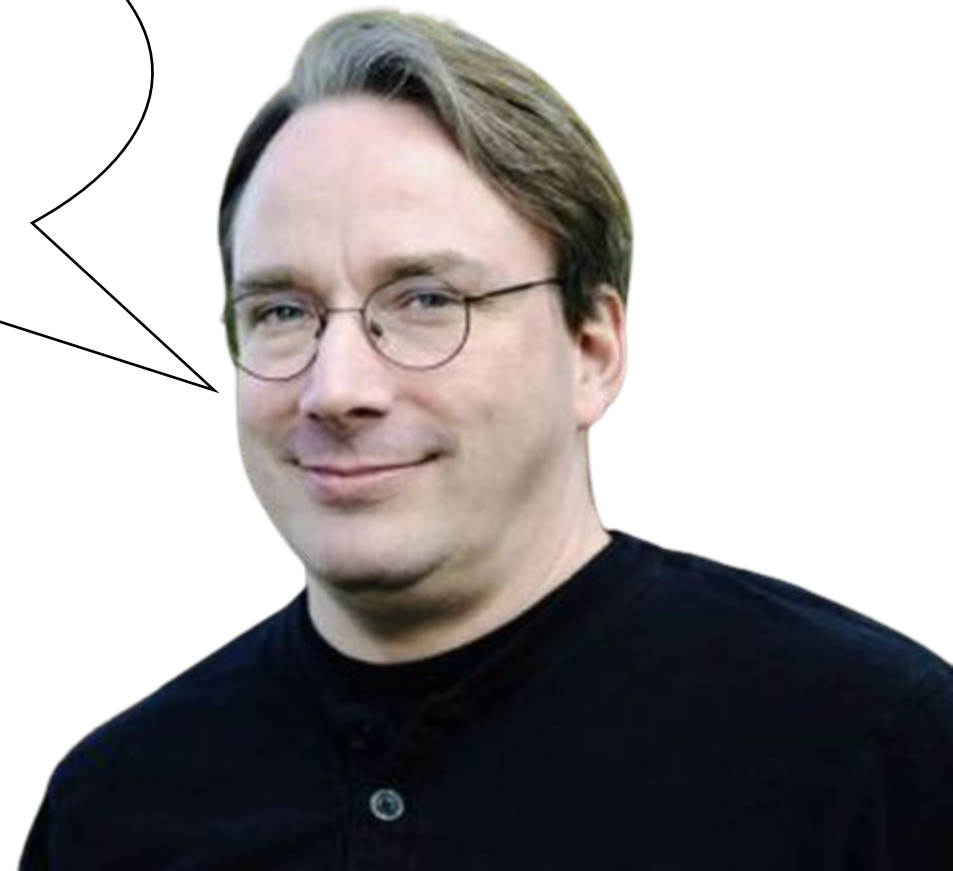
Programar un controlador de Floppy Disk: PD765 (en palabras)

- 16 Comandos
 - Cada comando carga de 1 a 9 bytes en el registro del dispositivo.
- Comando básicos: **read** y **write**
 - Cada uno requiere 13 parámetros
 - Los 13 parámetros deben ir empaquetados en 9 bytes.
- ¿Qué especifican **read** y **write**?
 - Dirección de bloque a leer
 - Número de sectores por pistas
 - Modo de grabación
 - Espacio separación entre sectores
 - Decisión con sectores disponibles (datos eliminados)

Programar un controlador de Floppy Disk: PD765 (en palabras)

- Operación completada chip controlador de dispositivo devuelve:
 - 23 campos de estado y error
 - Empaquetados en 7 bytes
- Otros para programar
 - ¿Motor encendido o apagado?
 - Encender motor
 - Apagar motor para evitar desgaste y daños en los medios
 - Y otros...
- ¿Sería usted capaz de programar un controlador de *floppy disk*?

Talk is cheap.
Show me the
code.



Dos funcione básicas (S.O)

1. Proporcionar a los programadores un conjunto **abstracto** de recursos simples
2. Administrar los recursos de **hardware**

Dos funciones básicas

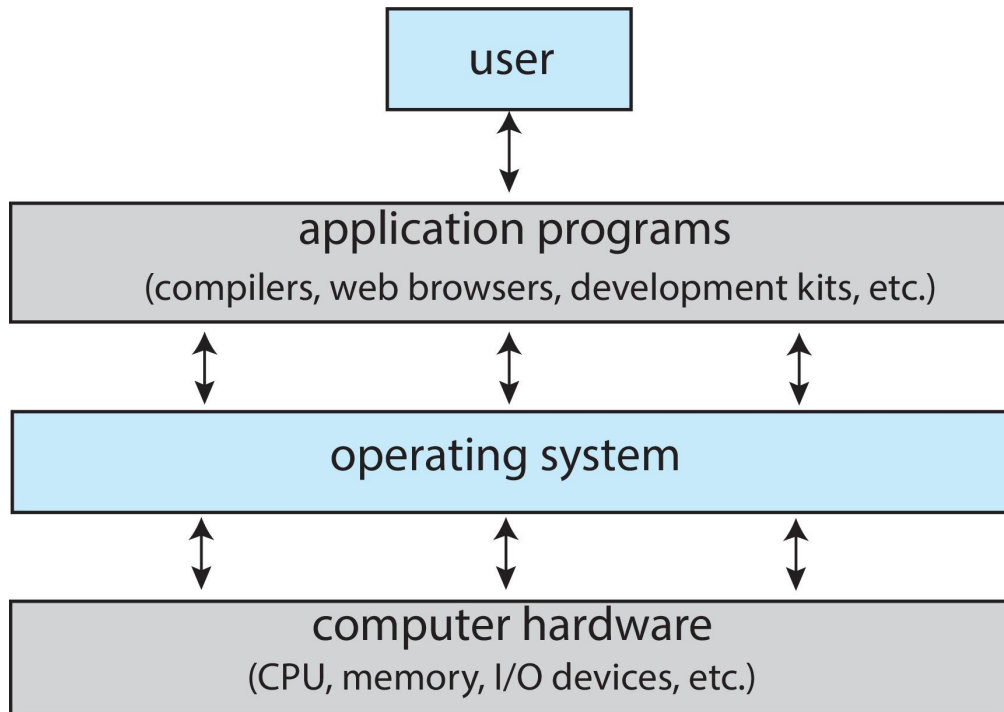
1. Proporcionar a los programadores un conjunto **abstracto** de recursos simples
 - **Abstracción:** ocultar la complejidad y los detalles de hardware
 - Abstracción **simple** de lidiar con los detalles de leer y escribir en disco
 - P. Ej.: ver el **diskette** como una colección de archivos con nombre
 - Abrir archivo, leer/escribir, cerrar archivo

Dos funciones básicas

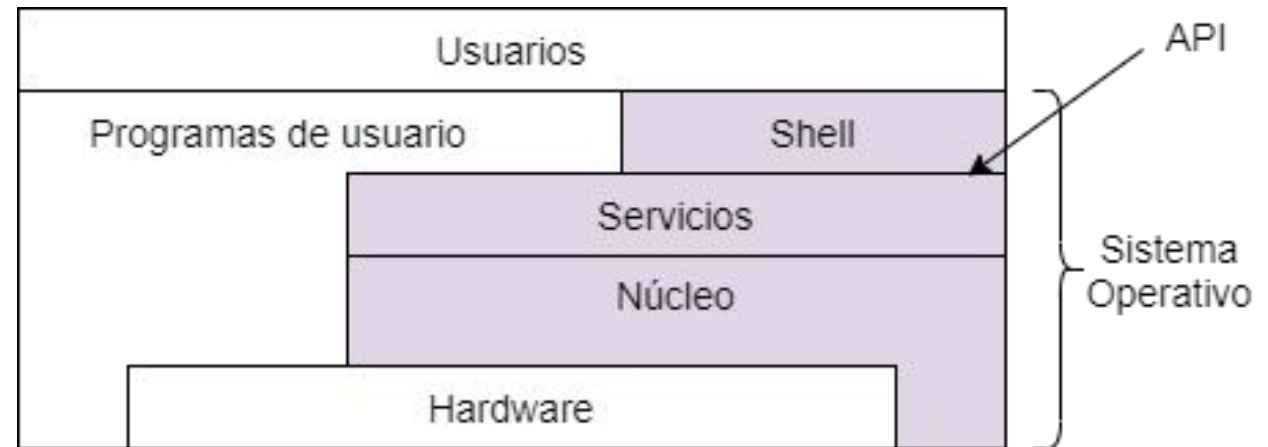
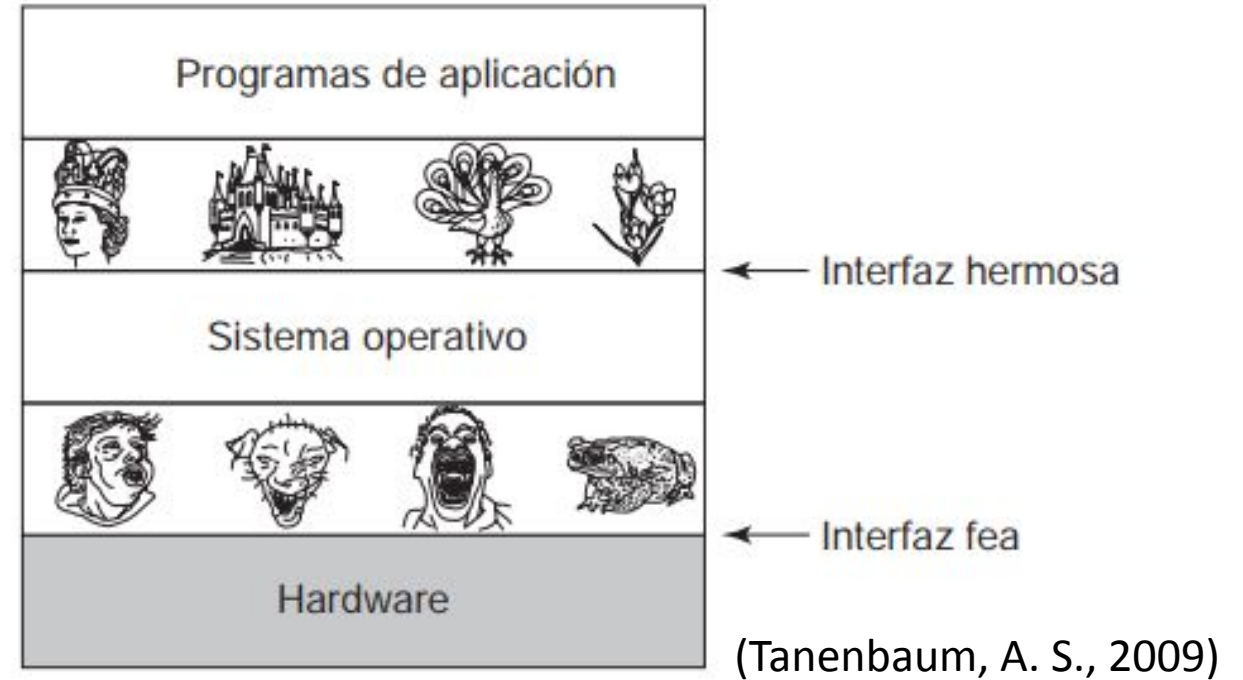
2. Administrar los recursos de **hardware**

- Asignación ordenada y controlada de recursos
- Programas compiten por los recursos
- P. Ej.: tres programas imprimiendo de manera simultánea.
- S.O impone **orden** al uso de los recursos
 - Orden de llegada. P. Ej.: la cola de impresión.
 - Uso de búferes para que los programas pongan allí lo que vana imprimir
- **Multiplexar** (compartir) los recursos
 - **En el tiempo** (por turnos): CPU, impresora
 - **En espacio** (una parte del recurso): Disco duro, RAM
- ¿Cómo se multiplexa? ¿Quién sigue? ¿Cuánto espacio? ¿Cuánto tiempo? Etc.

Definición



(Silberschatz, A., et. al., 2018)



Con modificaciones tomada de:
(Carretero Pérez, J. et. al., 2001)

Definición

- ¿Qué es un SO?
 - Pieza de software (programa de computadora)
- ¿Por qué se necesita?
 - La CPU solo está ejecutando repetidamente un bucle infinito de tres pasos.
 - Complejidad del hardware

Si no hay nada para ejecutar, ¿qué sentido tiene tener una CPU?



1. Lectura instrucción IP
2. Incremento del IP
3. Ejecución instrucción

- ¿Cuál es su objetivo?
 - Simplificar el manejo y la utilización del computador
 - De manera eficiente
 - De manera segura
- ¿Cuáles son sus funciones?
 1. Gestión de los recursos (hardware)
 2. Ejecución de servicios
 3. Ejecución de órdenes de los usuarios

Componentes básicos

- **Núcleo o Kernel**

- Gestiona los recursos de hardware
- Ofrece funcionalidad básica

- **Shell y/o GUIs***

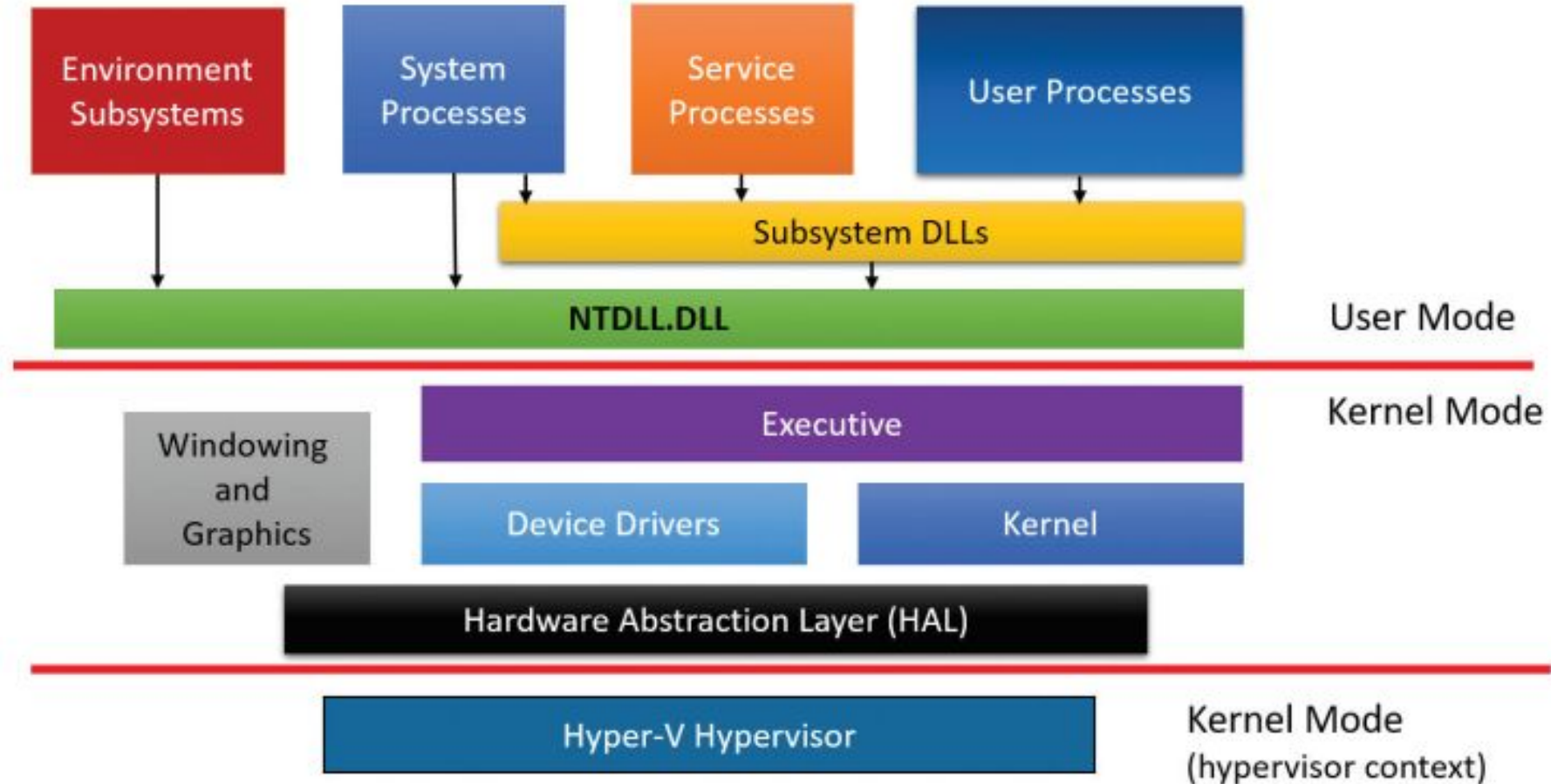
- Línea de comandos
- Intérprete de comandos
- Interfaz de texto o gráfica con la que el usuario dialoga con el SO

- **Servicios**

- API (Application Programming Interface)
- Proporciona servicios a los programas de usuario
- Windows API (Windows)
- System Calls (Linux)
- **Punto de entrada a los programadores**
 - Abstracciones simples del hardware

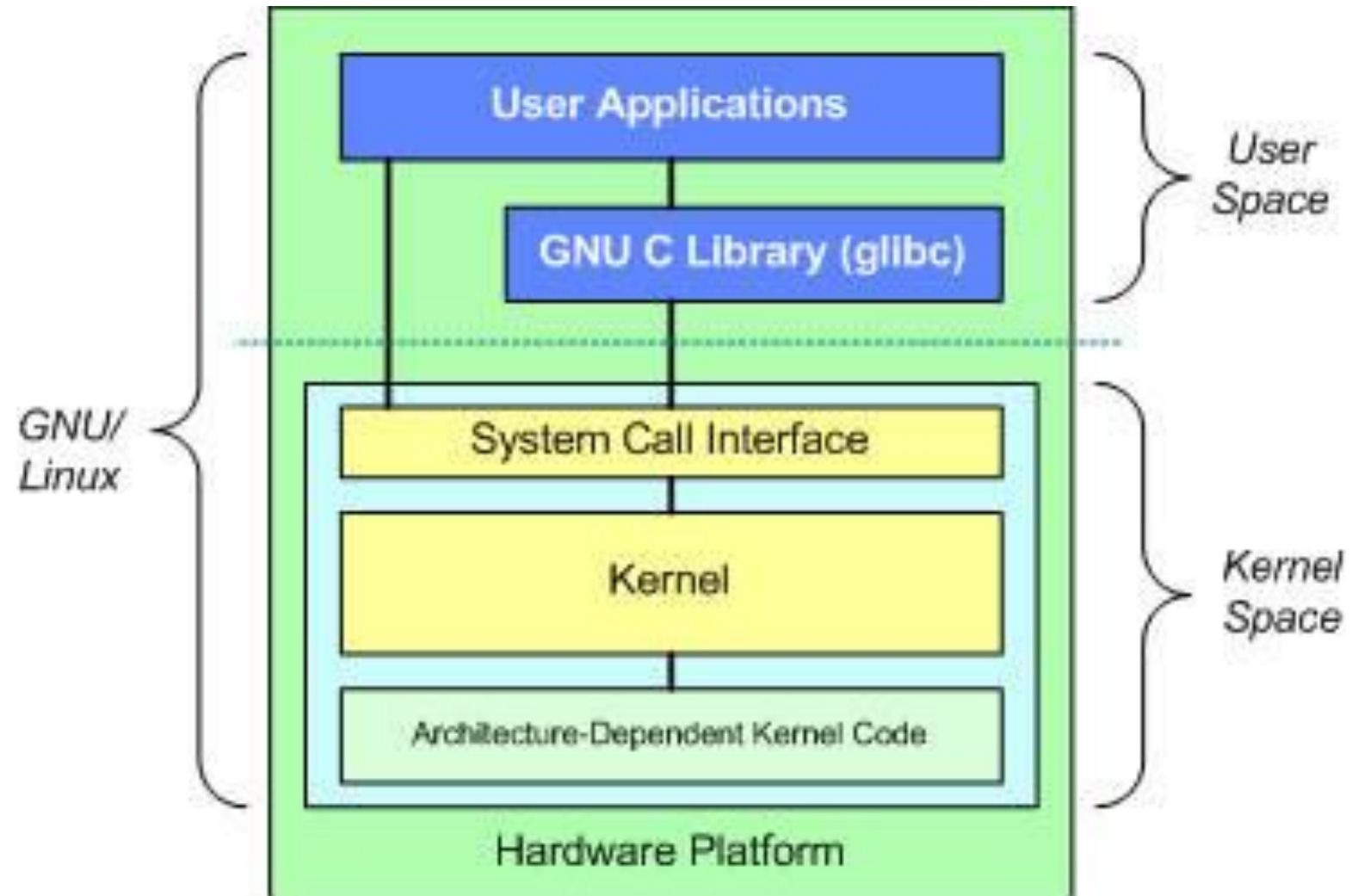
*Algunos consideran que no hace parte el S.O

Arquitecturas de diseño: Windows



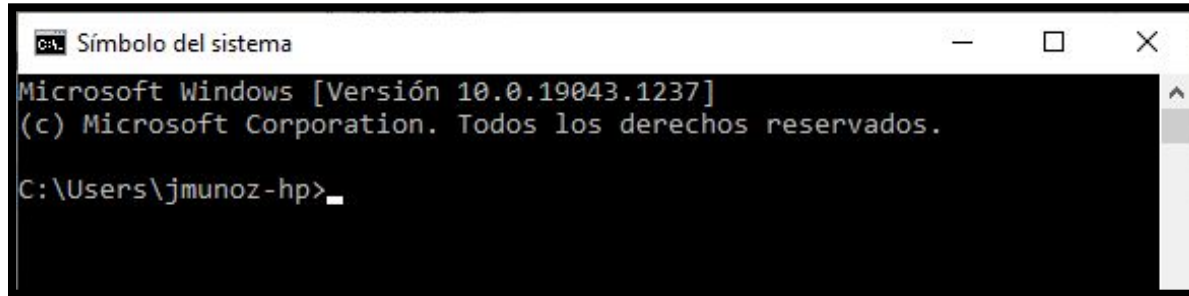
(Yosifovich, P., et. al., 2017)

Arquitecturas de diseño: Linux

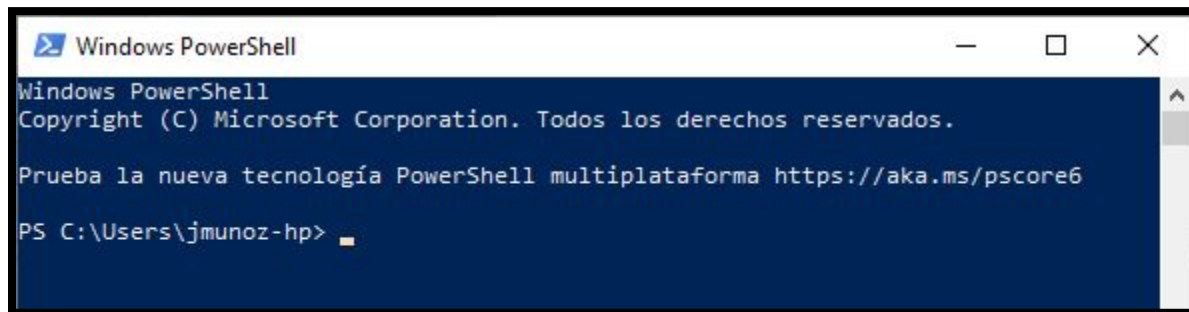


Fuente: <https://developer.ibm.com/articles/l-linux-kernel/>

Shells e intérpretes de comandos



```
Símbolo del sistema
Microsoft Windows [Versión 10.0.19043.1237]
(c) Microsoft Corporation. Todos los derechos reservados.
C:\Users\jmunoz-hp>
```



```
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Prueba la nueva tecnología PowerShell multiplataforma https://aka.ms/pscore6
PS C:\Users\jmunoz-hp>
```



```
root : bash
File Edit View Scrollback Bookmarks Settings Help
[root@correo ~]#
```

Windows API

- Documentación de Windows API
 - <https://docs.microsoft.com/en-us/windows/win32/apiindex/windows-api-list>
- Programa que usa las funciones `CreateFile()` y `CloseHandle()` definidas en el API de Windows.
- Programa crea un archivo en blanco llamado `archivo.txt`

```
#include <windows.h>
#include <stdio.h>

int main() {
    LPCWSTR filename = L"archivo.txt";

    HANDLE hFile;
    hFile = CreateFile(filename,
        GENERIC_WRITE,
        0,
        NULL,
        CREATE_ALWAYS,
        FILE_ATTRIBUTE_NORMAL,
        NULL);

    if (hFile == INVALID_HANDLE_VALUE)
    {
        return 2;
    }
    CloseHandle(hFile);
}
```

Linux System Calls

- Programa que usa las funciones `open()` y `close()` definidas en las System Calls de Linux
- Programa crea un archivo en blanco llamado `archivo.txt`
- Documentación de estas funciones
 - <https://man7.org/linux/man-pages/man2/open.2.html>

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int main()
{
    int hFile;
    const char filename[] = "archivo.txt";
    hFile = open(filename, O_CREAT);
    if (hFile < 0)
    {
        return 2;
    }
    close(hFile);
}
```

Solo librería estándar de lenguaje C

- El mismo código funciona para ambos sistemas operativos.
- ¿**Cuándo** usar entonces la capa de servicios?
 - Funcionalidades **específicas** del S.O
 - P. Ej.: Registro de Windows (Windows)
 - P. Ej.: Colas de mensajes System V (Linux)
 - P. Ej.: Manipulación de usuarios y grupos (cada S.O)
 - Drivers

```
#include <stdio.h>

int main()
{
    FILE* hFile = NULL;
    const char filename[] = "archivo.txt";

    hFile = fopen(filename, "a");
    if (hFile == NULL)
    {
        return 2;
    }

    fclose(hFile);
}
```

El mismo código para Windows y Linux

Funciones del SO

1. El S.O como gestor de recursos
2. El S.O como maquina extendida
3. El S.O como interfaz de usuario

El S.O como gestor de recursos

- En un computador se ejecutan varios programas al tiempo
 - Programas compiten por recursos
 - CPU, Memoria, E/S
1. **Asignación** □ Disponibilidad, prioridad, resolución de conflictos, recuperación luego del uso.
 2. **Protección** □ Unos programas no puedan usar los recursos de otros, protección de la información.
 3. **Contabilidad** □ Medir cantidad de recursos usados, monitoreo.

El S.O como máquina extendida

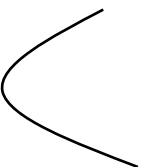
Extiende el modelo de programación de la computadora.

1. **Ejecución de programas (procesos)** □ lanzar ejecución, modificar condiciones ejecución, comunicar, sincronizar, administrar procesos
 - `exec()` □ Linux System Call
 - `CreateProcess()` Windows API
2. **Ordenes de E/S** □ operaciones de lectura/escritura, estado periféricos, dependiente del hardware
3. **Operaciones sobre archivos** □ creación, borrado, renombrado, apertura, escritura, lectura de archivos
4. **Tratamiento de errores** □ detección y tratamiento de errores
 - Operaciones inválidas

El S.O como interfaz de usuario

- **A través del shell □ línea de comandos**

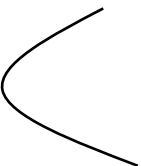
- Bucle infinito

- 
- Espera la orden del usuario
 - Interpreta las órdenes escritas del usuario
 - Ejecuta la orden

- Archivos de scripting (programación)

- **GUI (*Graphical User Interface*)**

- Bucle infinito (P. ej. En Windows Message Loop, Events Loops)

- 
- Lee los eventos producidos por el mouse, teclado
 - Interpreta los eventos
 - Ejecuta las acciones que indican los eventos

- Desarrollo de aplicaciones gráficas

Ejemplo Message Loop en Windows

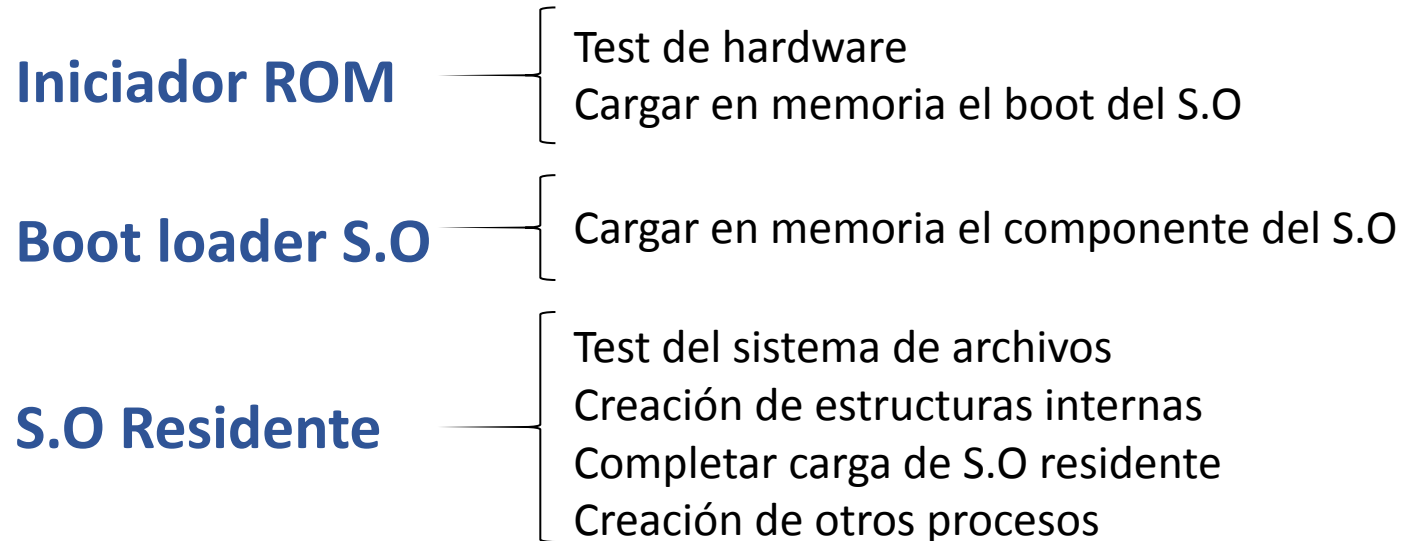
```
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow)
{
    MSG msg;
    BOOL bRet;

    while (1)
    {
        bRet = GetMessage(&msg, NULL, 0, 0);

        if (bRet > 0)  // (bRet > 0 indicates a message that must be processed.)
        {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
        else if (bRet < 0)  // (bRet == -1 indicates an error.)
        {
            // Handle or log the error; possibly exit.
            // ...
        }
        else  // (bRet == 0 indicates "exit program".)
        {
            break;
        }
    }
    return msg.wParam;
}
```

Arranque del computador

- Dos fases
 - Arranque del hardware
 - Arranque del S.O
- Actividades y responsabilidades

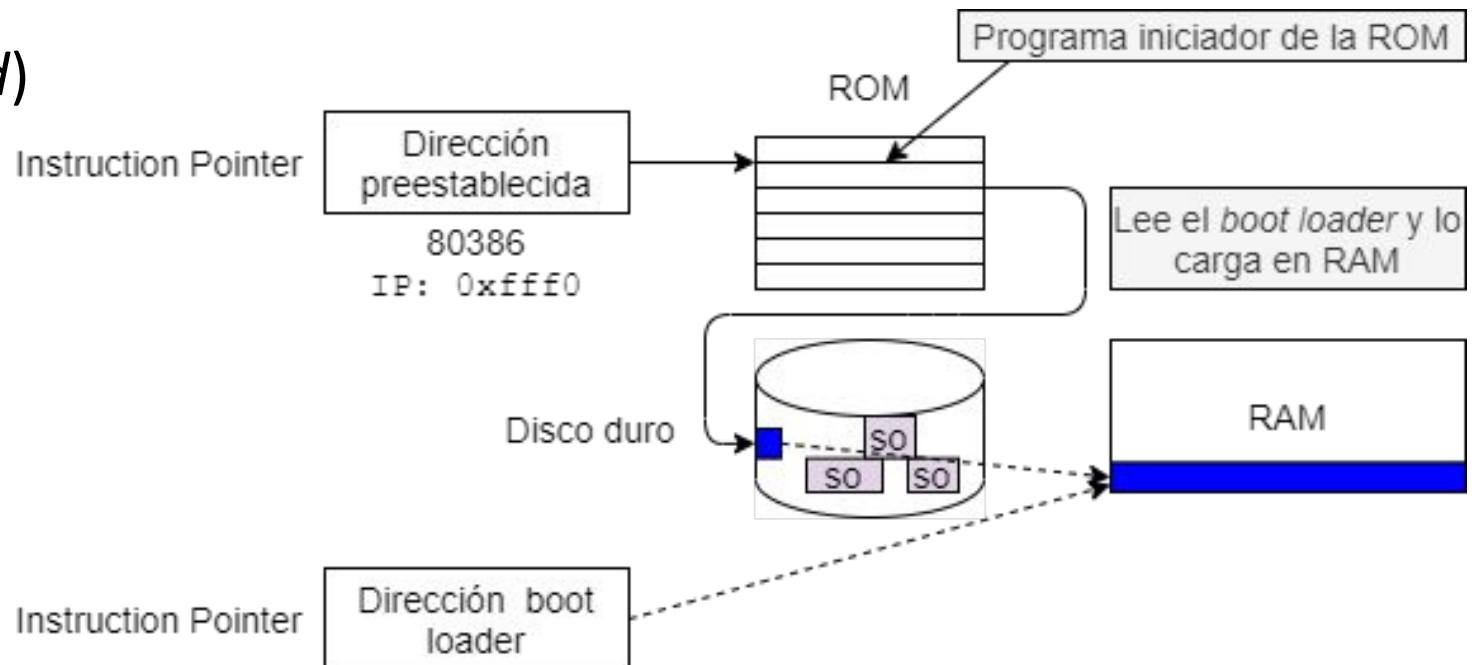


Arranque del computador

- Arranque del hardware
 - Encendido o RESET
 - Cargar valores **predefinidos** en los registros
 - Registro IP con la dirección de comienzo del iniciador ROM (programa)
 - Iniciador ROM
 - Comprobación del sistema
 - Lectura y carga del *boot loader* del sistema operativo en memoria
 - Entregar control al *boot loader*: IP apunta a la primera instrucción del *boot loader*

Arranque del computador

- Ubicación del sistema operativo
 - *Boot loaders* están siempre almacenados en zonas predefinidas del disco
 - P. Ej.: los cuatro primeros sectores del disco.
 - Tamaño fijo
 - MBR (*Master Boot Record*)



Arranque del computador

- Arranque del sistema operativo
 - Boot loader lleva a memoria algunos componentes del S.O
 - Se inician operaciones tales como
 - Comprobación del sistema: hardware, sistema de archivos
 - Establecimiento de estructuras propias del S.O: procesos, memoria, E/S
 - Cargar en memoria S.O residente (siempre está en memoria)
 - Iniciar otros programas: login, demonios, procesos auxiliares
- Lectura recomendada: Kernel booting process. Part 1
 - <https://0xax.gitbooks.io/linux-insides/content/Booting/linux-bootstrap-1.html>

Referencias

- Carretero Pérez, J., García Carballeira, F., De Miguel Anasagasti, P., & Pérez Costoya, F. (2001). Introducción a los sistemas operativos. In *Sistemas operativos. Una Visión Aplicada* (pp. 33–74). McGraw Hill.
- Tanenbaum, A. S. (2009). Introducción. In *Sistemas Operativos Modernos* (3rd ed., pp. 3–75). Pearson Educación.
- Yosifovich, P., Ionescu, A., Russinovich, M. E., & Solomon, D. A. (2017). *Windows Internals Part 1. System architecture, processes, threads, memory management, and more* (7th ed., Ser. Windows Internals). Microsoft Press.

Procesos

Adaptación

Juan Felipe Muñoz Fernández

Preguntas

- ¿Qué es un proceso?
- ¿Por qué es importante el concepto de proceso en los sistemas operativos?

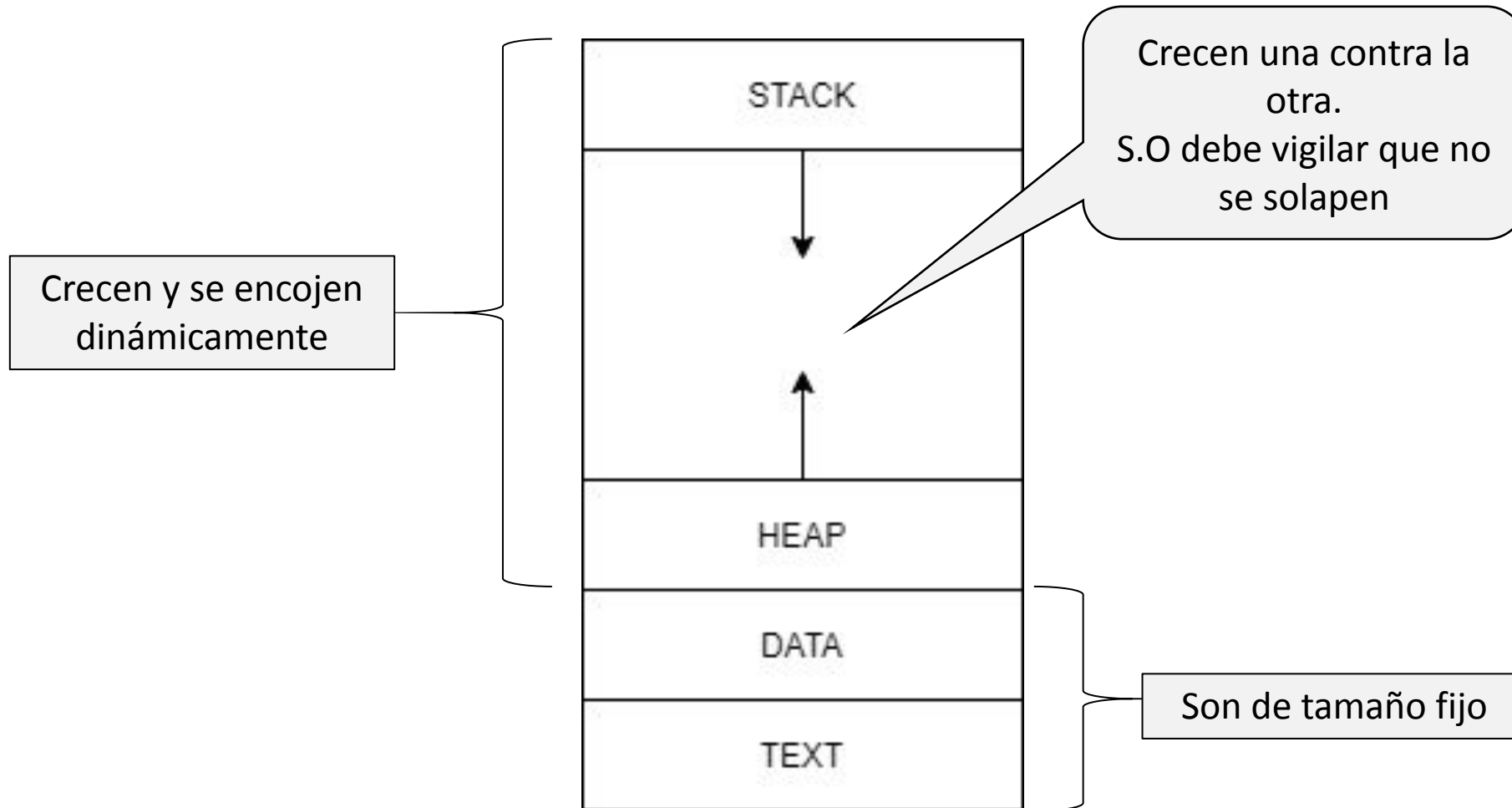
Definición

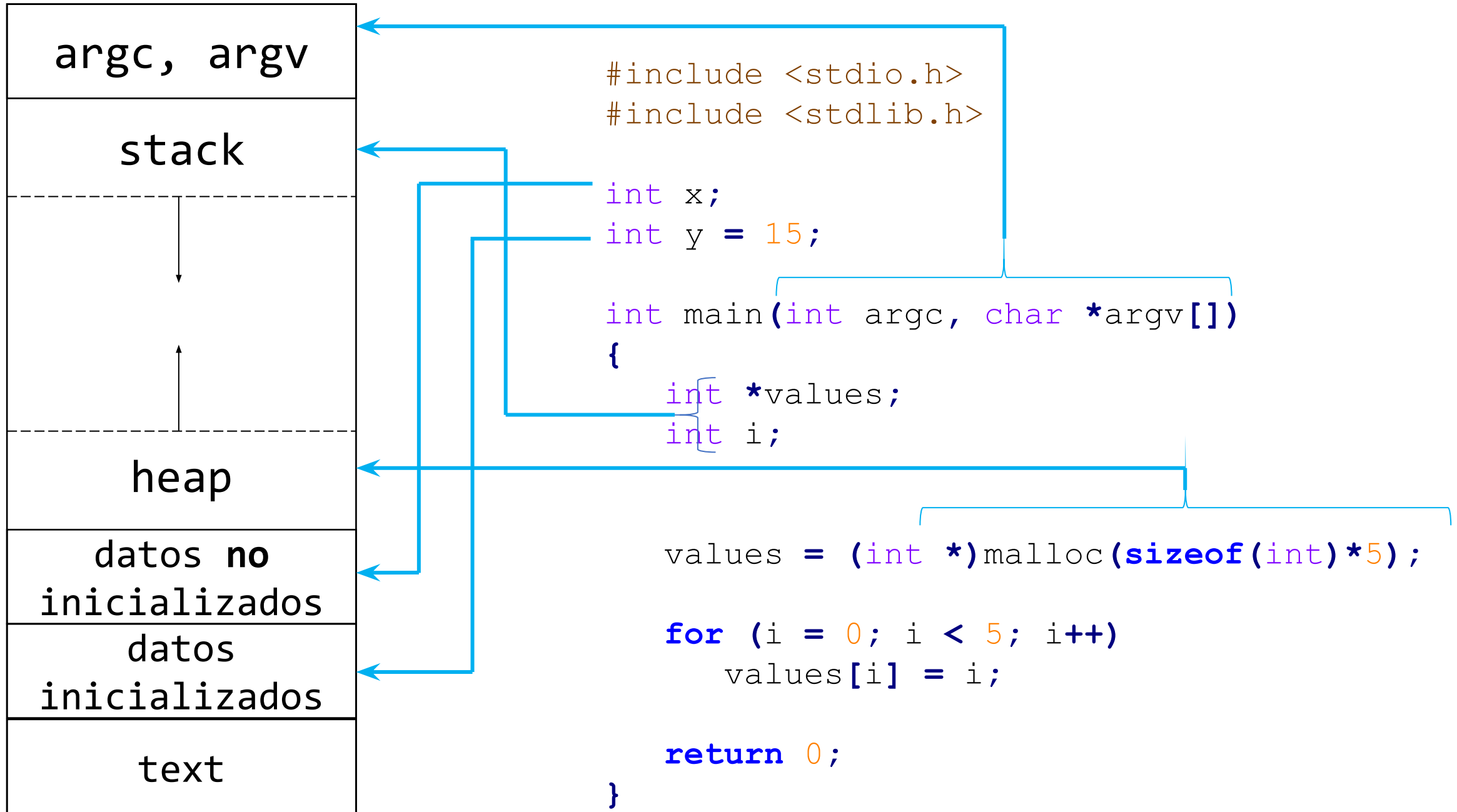
- Programa en ejecución
 - Programa en disco es una entidad estática
 - Programa cargado en memoria es una entidad dinámica
- Es la unidad de trabajo del sistema operativo
 - El S.O también tiene sus propios procesos
- Un sistema computacional moderno consiste de una colección de procesos
 - Ejecutando código de usuario (aplicaciones de usuario: ring 3)
 - Ejecutando código del núcleo del S.O (procesos del S.O: ring 0)
- Todos los procesos podrían ejecutarse concurrentemente en CPU
 - Tiempo compartido (multiplexión en el tiempo)

Definición

- Un programa se vuelve proceso cuando:
 - El archivo ejecutable es cargado en memoria
 - Todos los programas deben estar cargados en memoria para su ejecución
- ¿Cómo se carga un archivo ejecutable en memoria?
 - Usuario da la orden: doble clic, enter, orden en línea de comandos
 - El S.O lo carga automáticamente: servicios, demonios, procesos propios
- Mapa de memoria de un proceso
 - Sección **text**: código ejecutable
 - Sección **data**: variables globales
 - Sección **heap**: memoria dinámica (p. ej.: listas)
 - Sección **stack**: datos temporales, parámetros funciones, variables locales

Mapa de memoria de un proceso



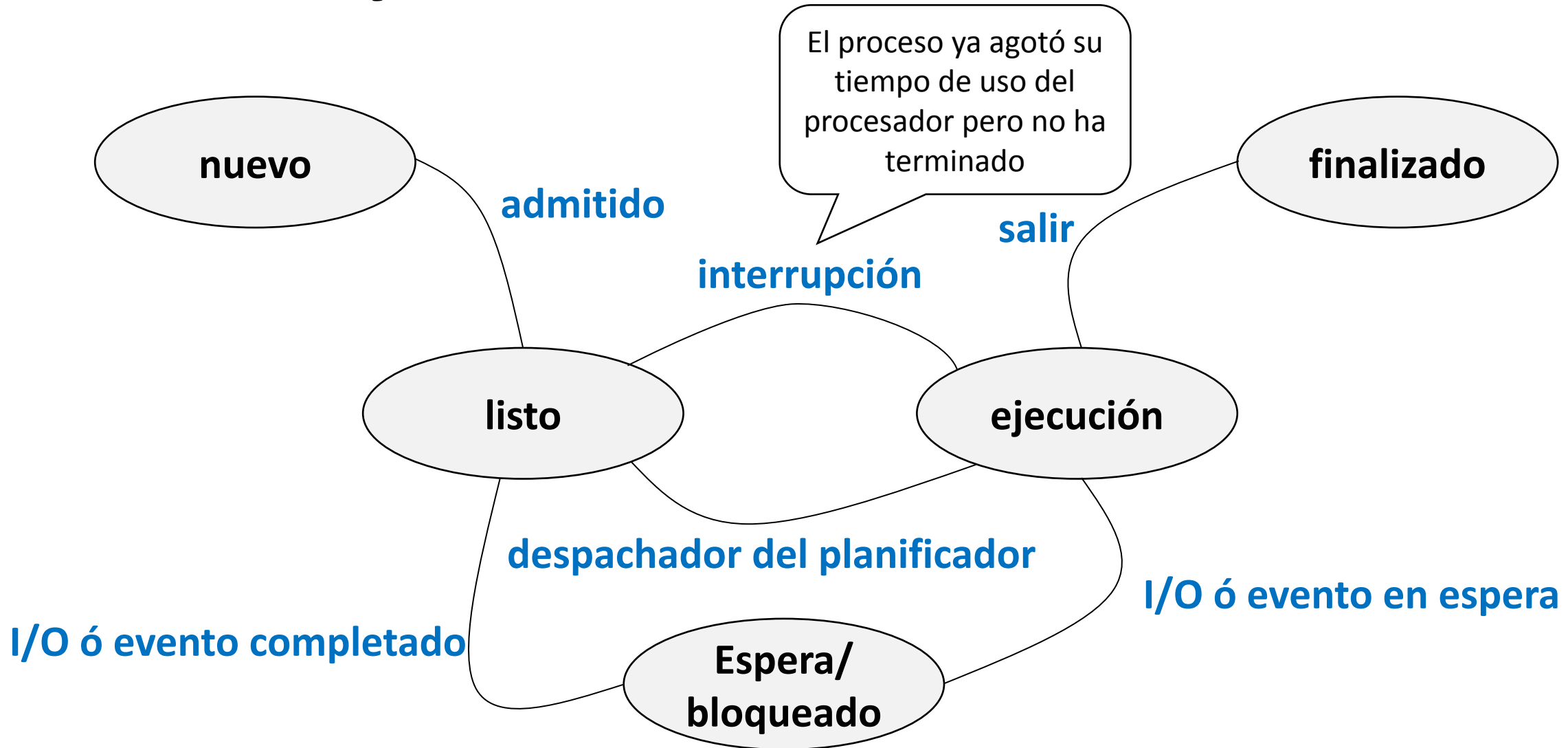


Formación de un proceso

- Completar toda la información que lo constituye
 - Asignar espacio de memoria (virtual) constituido por varios segmentos
 - Seleccionar un PCB libre en tabla de procesos
 - Rellenar el BCP con la información del proceso
 - Cargar en el segmento de texto el código + rutinas del sistema
 - Cargar en el segmento de datos los datos inicializados
 - Crear el segmento de pila con los parámetros que se pasan al programa

Estados y transiciones

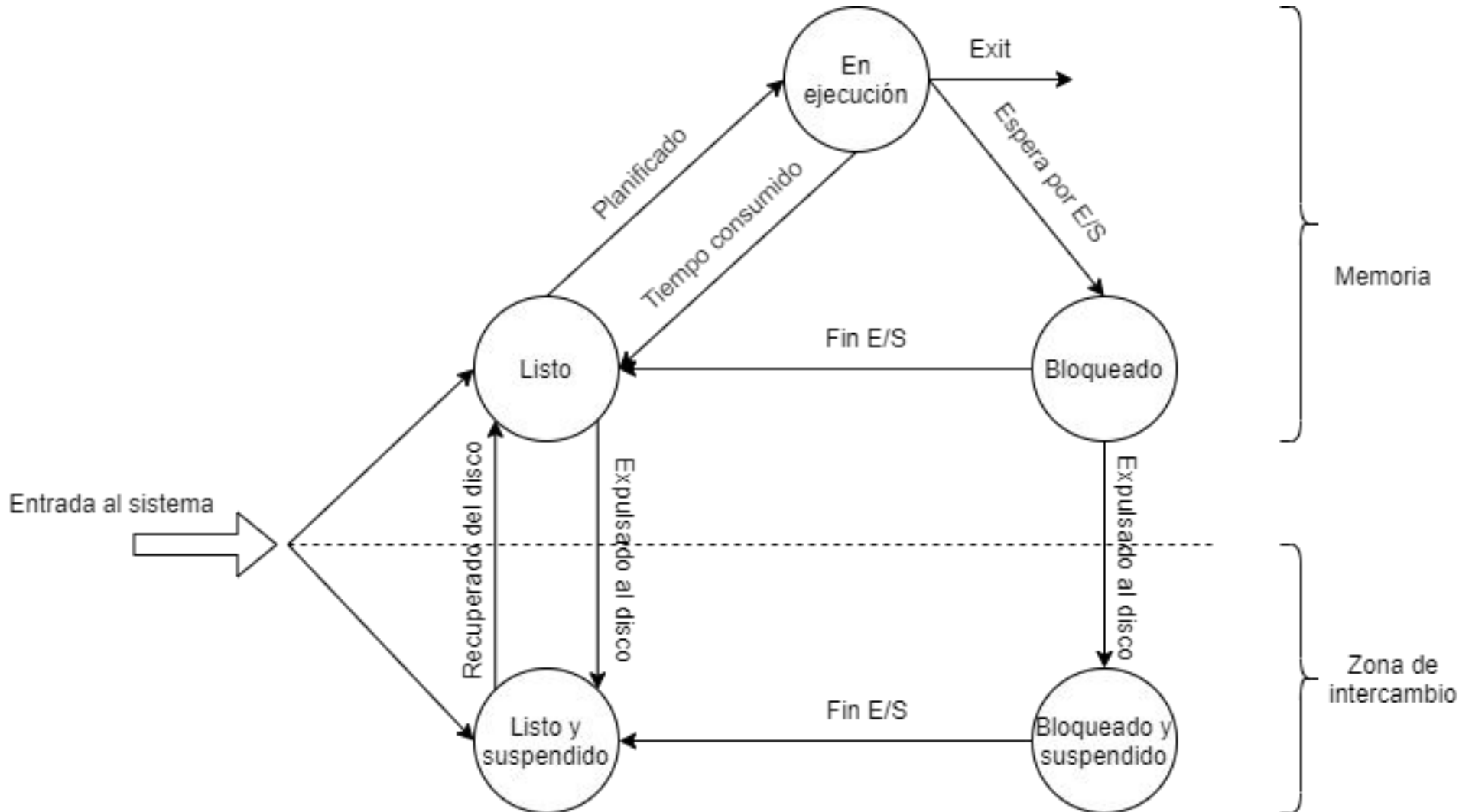
Solo un proceso puede estar en ejecución en cualquier núcleo de procesador a la vez.



Estados y transiciones

- **Nuevo**
 - Proceso creado (cargado en memoria)
- **Ejecución**
 - Instrucciones siendo ejecutadas
- **Espera/bloqueado**
 - El proceso está a la espera de que ocurra algo
 - Que se complete una solicitud de I/O
 - Que se reciba una señal
- **Listo**
 - El proceso está a la espera de que se le asigne tiempo de procesador
- **Finalizado**
 - El proceso completó su ejecución

Estados y transiciones



- Estado de suspensión: retira los marcos de página del proceso y los envía al área de *swapping*.
- Liberar memoria para los procesos no suspendidos

Bloque de control de proceso (BCP/PCB)

- Se requiere una estructura de control y representación de la información de un proceso
 - Control de ejecución
 - Tiempos asignados
 - Recursos usados
 - Tiempos de espera
- El despachador del planificador requiere cierta información del proceso
 - Asignar nuevamente tiempo de procesador
 - Hacer transiciones hacia otros estados

Bloque de control de proceso

- Registra la «vida» de un proceso en el sistema

Estado del proceso
Número de proceso (PID)
Contador de programa (IP: Instruction pointer)
Registros de CPU
Límites de memoria
Archivos abiertos
...

Bloque de control de proceso

- Estado del proceso
 - Nuevo, listo, en ejecución, en espera, terminado, etc.
- Contador del programa
 - Dirección de la siguiente instrucción a ejecutarse
- Registros de CPU
 - Todos los registros que se requieran para interrumpir y ejecutar nuevamente el proceso.
- Información del planificador
 - Prioridad, apuntadores a las colas de planificación, entre otros

Bloque de control de proceso

- Información de administración de memoria
 - Valores de límites de memoria, registros base, tablas de páginas, etc.
- Información de contabilidad
 - Cantidad de tiempo en CPU, límites de CPU, etc.
- Información de estado de I/O
 - Dispositivos de I/O asignados al proceso
 - Archivos abiertos
- Otra información
 - El PCB es dependiente de las consideraciones de diseño de cada S.O
 - No es una estructura estándar.

PCB en xv6 [1/2]

```
// Registros del proceso
struct context {
    int eip;
    int esp;
    int ebx;
    int ecx;
    int edx;
    int esi;
    int edi;
    int ebp;
};
```

```
// Estado del proceso
enum proc_state {
    UNUSED,
    EMBRYO,
    SLEEPING,
    RUNNABLE,
    RUNNING,
    ZOMBIE
};
```

PCB en xv6 [2/2]

```
struct proc {
    char *mem;           // Dónde inicial el proceso en memoria
    uint sz;             // Tamaño del proceso en memoria
    char *kstack;        // Parte inferior pila Kernel para proceso
    enum proc_state state; // Estado del proceso
    int pid;             // ID del proceso
    struct proc *parent; // Proceso padre
    void *chan;          // Si !=0, proceso durmiendo
    int killed;          // Si !=0 proceso killed
    struct file
*ofile[NOFILE];         // Archivos abiertos
    struct inode *cwd;   // Directorio actual
    struct context context; // Contexto del proceso
    struct trapframe *tf; // Para manejo de interrupciones
};
```

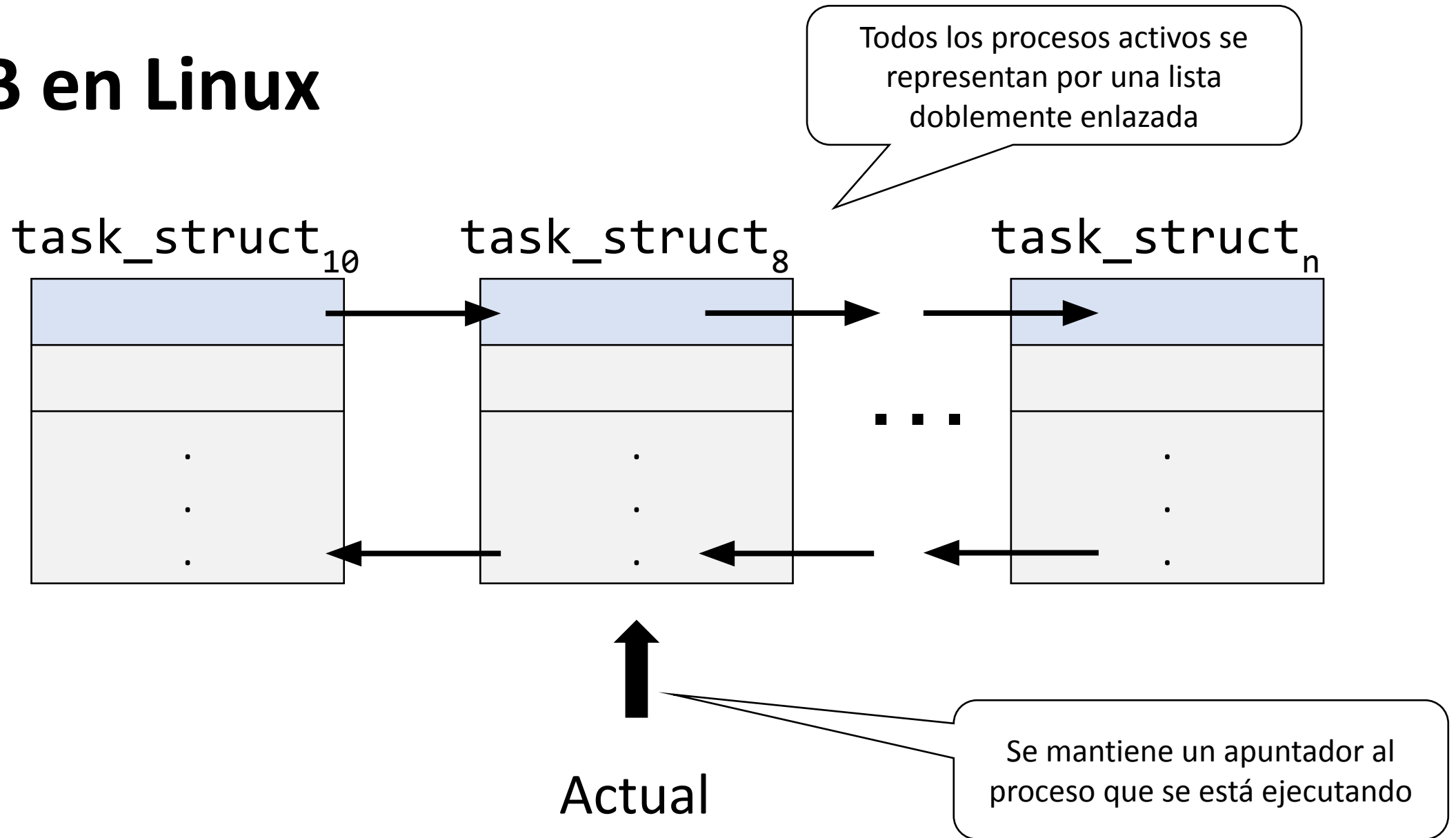

PCB en Linux

```
long state;
struct sched_entity se;
struct task_struct *parent;
struct list_head children;
struct files_struct *files;
struct mm_struct *mm;

// Estado del proceso
// Información del planificador
// Proceso padre de este proceso
// Hijos de este proceso
// Lista de archivos abiertos
// Espacio de direcciones en RAM
```

- El PCB en Linux está representado por la estructura **task_struct**.
- **Algunos** miembros de esta estructura son los indicados en el código anterior.
- La estructura **task_struct** está definida en `<include/linux/sched.h>`

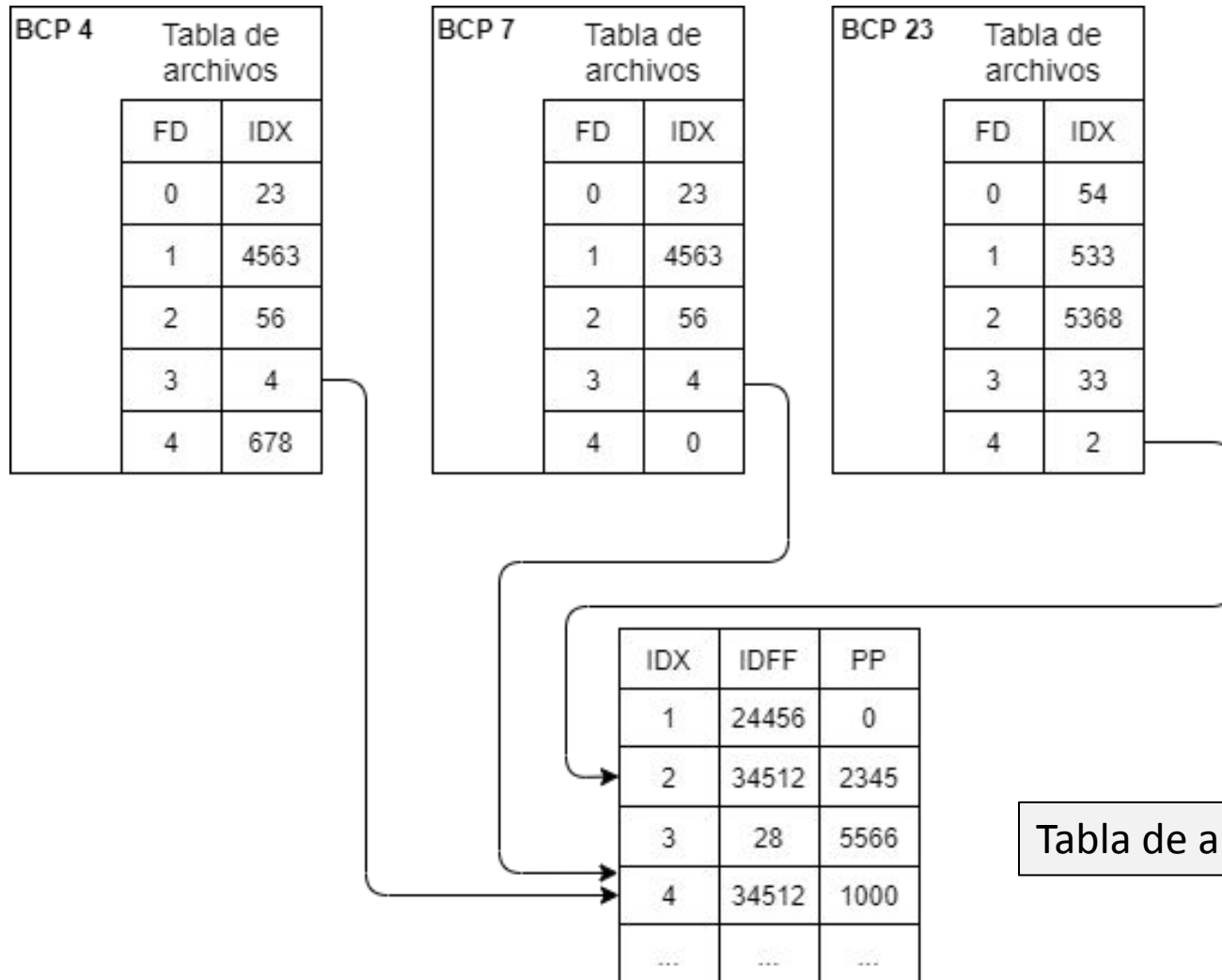
PCB en Linux



PCB e información compartida

- La información compartida por procesos no puede residir en el PCB
 - El PCB es único a cada proceso.
 - Información restringida de cada proceso.
- El PCB debe incluir un apuntador a la información compartida
- Considere el caso de dos procesos que abrieron el mismo archivo
 - Archivo se heredó de proceso padre.
 - Archivo se abrió de manera independiente por los dos procesos
 - Se debe compartir el puntero de posición
 - Puntero de posición no debe estar en el BCP

PCB/BCP e información compartida



- Cada apertura del archivo genera una nueva entrada en la tabla de archivos externa.
- Cada proceso tiene su PP puntero de posición en el archivo.
- BCP 7 es hijo de BCP 4, están compartiendo el mismo PP.
- El PP no puede estar en el BCP: considere casos de BCP 4 y BCP 7

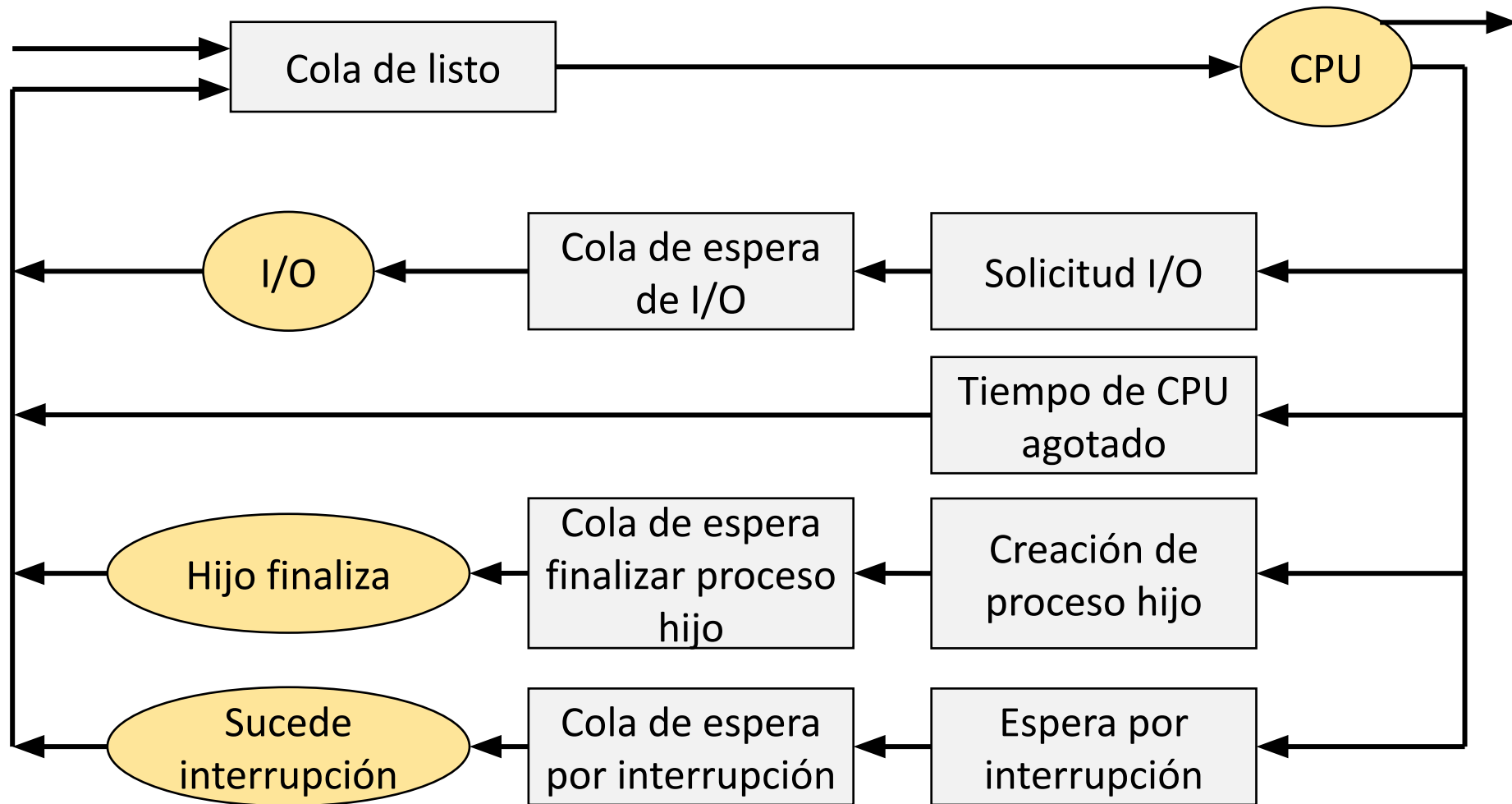
Planificación de procesos

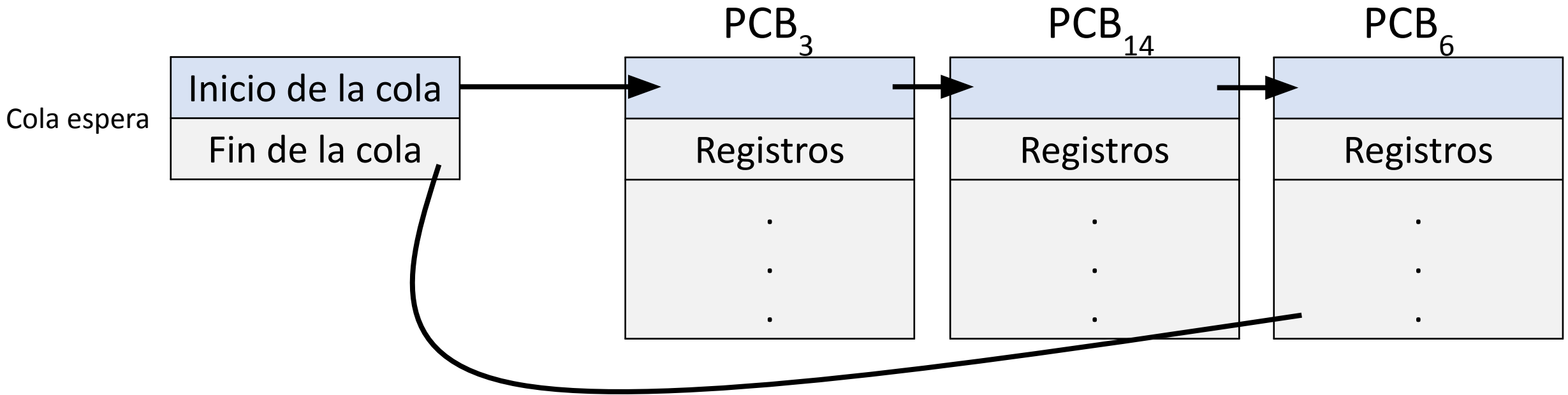
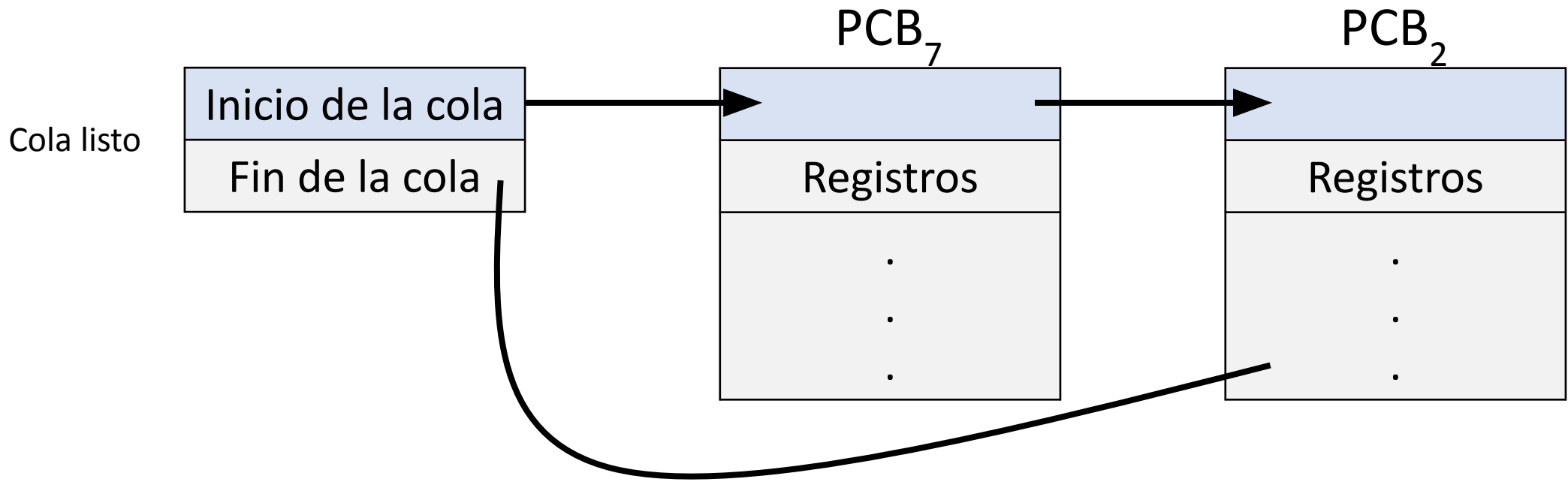
- Objetivos
 - Maximizar el uso del procesador
 - Compartir en el tiempo el uso del procesador por todos los procesos que lo requieran
 - De la mejor manera posible
 - Ningún proceso debe monopolizar el tiempo de procesador
- Cada núcleo de procesador puede ejecutar un proceso a la vez
 - Sistemas multinúcleo pueden ejecutar más de un proceso a la vez
- Grado de multi programación
 - Número de procesos actualmente en memoria

Colas de planificación de procesos

- Estado de **listo** es una cola
 - Proceso queda a la espera de que se le asigne tiempo de procesador
 - Usualmente se implementa como una lista enlazada (ligada)
 - Encabezado de la cola apunta al primer PCB en la lista
 - Cada PCB incluye un apuntador al siguiente PCB en la lista
- El S.O implementa varias colas para manejar los diferentes estados de un proceso

Colas: una de listo y tres de espera





Colas: una de listo y tres de espera

- Los círculos/óvalos indican los recursos que sirven a las colas
- Flechas indican el flujo de un proceso en el sistema
- Eventos que pueden suceder cuando se ejecuta un proceso
 - Hace una solicitud de I/O. P. Ej.: la lectura de un archivo □ Cola de I/O.
 - Crea un proceso hijo. □ Cola hasta que hijo termine.
 - Sale de CPU por tiempo agotado o por alguna interrupción del proceso.
- Tres colas de espera para representar el estado de espera o bloqueado de un proceso

Planificación de CPU

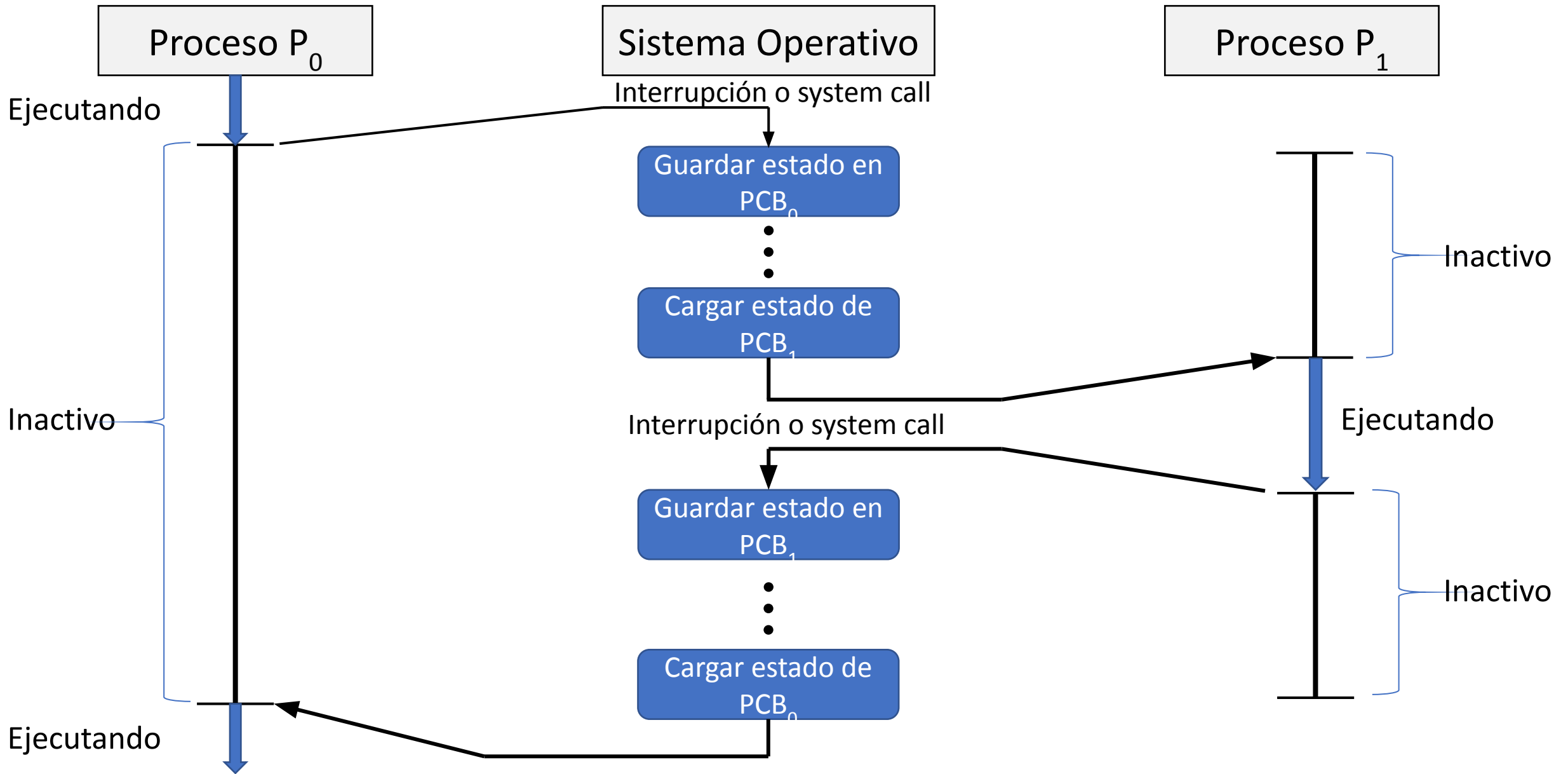
- Objetivos del planificador
 - Seleccionar un proceso (entre varios) para ejecutarse
 - De la mejor manera posible
- Procesos intensivos de I/O
 - Se ejecutan unos milisegundos antes de quedar en espera por I/O
- Procesos intensivos de CPU
 - Requieren más tiempo de uso de CPU
- Ningún proceso puede monopolizar el tiempo de CPU

Cambios de contexto

- Cuando ocurre una interrupción (p. ej.: I/O) el S.O debe sacar de CPU al proceso en ejecución
 - Para atender la interrupción
 - El S.O ejecuta código que atiende la interrupción
 - Debe ejecutar otro proceso distinto en la CPU: Un proceso propio del S.O
- Se debe guardar el contexto del proceso interrumpido
 - Proceso que se estaba ejecutando antes de la interrupción
 - Para seguir ejecutándolo cuando se le asigne nuevamente tiempo de CPU

Cambios de contexto

- El contexto de un proceso está representado en el PCB
- El cambio de contexto implica al núcleo del S.O
 - Guardar el contexto del proceso interrumpido en el PCB
 - Cargar el contexto del proceso que será ejecutado
 - Recordar que cada vez que se crea un proceso se crea el PCB del proceso
- El cambio de contexto es un gasto necesario (*overhead*).
 - El sistema no está haciendo nada útil mientras realiza el cambio de contexto
 - CPU no se usa en este cambio de contexto
 - Unos cuantos nano/**micro**/mili/segundos
 - Velocidad de la memoria RAM, número de registros, instrucciones especiales



Llamadas al sistema y cambios de contexto

- Consideremos la siguiente llamada al sistema

```
#include <unistd.h>
ssize_t read(int fd, void *buf, size_t count);
```

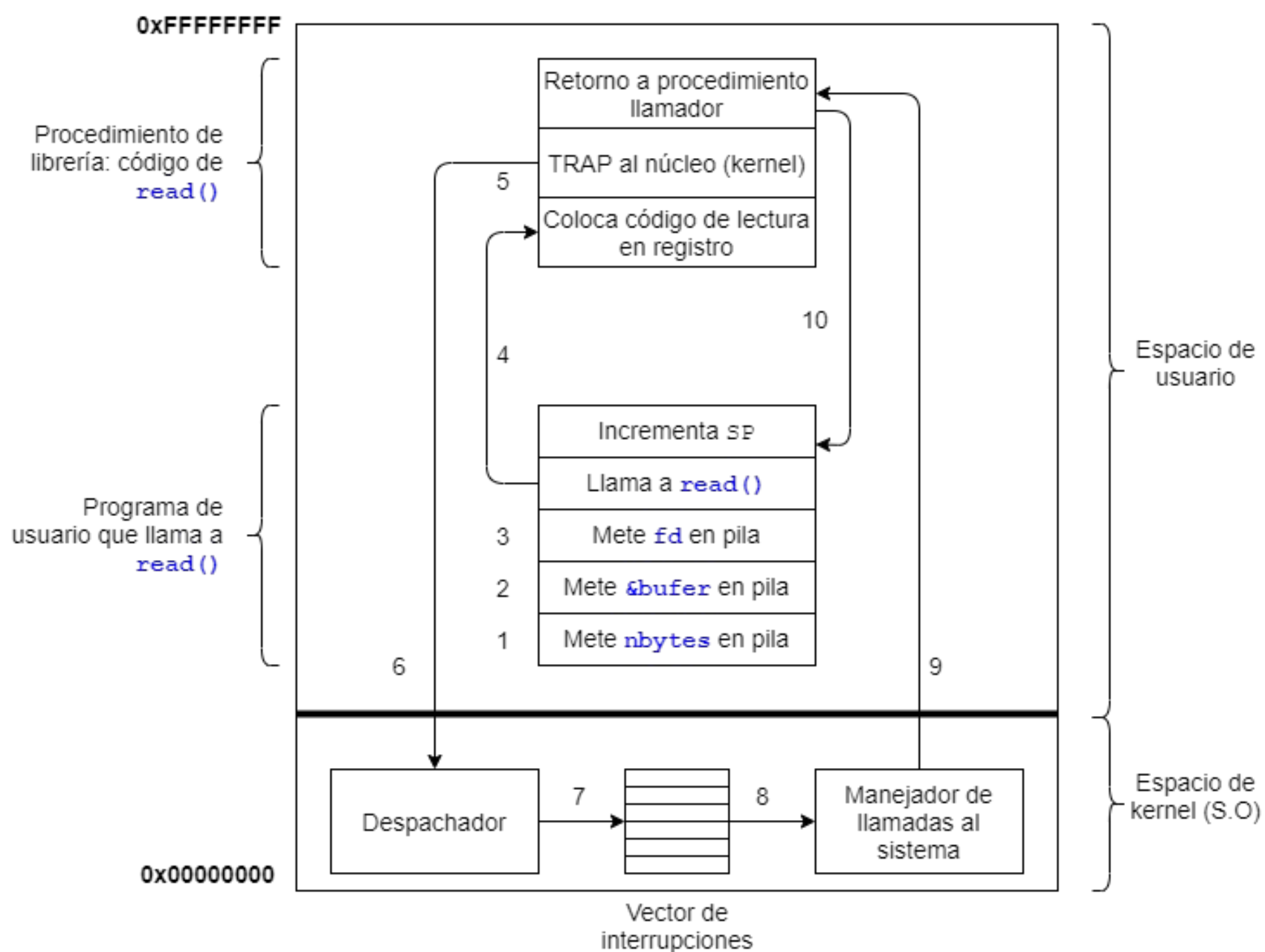
- Donde,
 - `<unistd.h>` archivo de cabecera que debe incluirse para hacer la llamada
 - `ssize_t` tipo de dato que retorna: entero con signo (POSIX.1.)
 - `read` nombre de la llamada
 - `int fd` manejador (*handle*) de un archivo abierto
 - `void *buf` apuntador al búfer para recibir los datos leídos
 - `size_t count` número de bytes a leer.
- La llamada `read()` retorna el número de bytes leídos

Llamadas al sistema y cambios de contexto

- Programa llamador mete parámetros en pila en orden inverso.
 - Primer y tercer parámetro por valor
 - Segundo parámetro por referencia
- Se hace llamada a `read()`
- Código de `read()` coloca en registro código de la llamada al sistema
 - El S.O espera que el código esté en ese registro
- Código de `read()` ejecuta instrucción TRAP
 - Interrupción por software
- Se ejecuta código del núcleo del S.O que examina número de llamada

Llamadas al sistema y cambios de contexto

- Pasa control a manejador de llamadas al sistema
 - En el núcleo del S.O.
 - Atiende llamada
- Se devuelve control al código de `read()` posterior a la instrucción TRAP
- Se devuelve control al programa llamador a la instrucción posterior a `read()`
- Programa llamador limpia la pila
 - Incrementa el apuntador de pila



Llamadas al sistema y cambios de contexto

- ¿Cuántos cambios de contexto hubo en el caso anterior?
 - Una llamada al sistema no siempre produce un cambio de contexto a menos que la llamada sea bloqueante.
 - Si la llamada es bloqueante se aprovecha el tiempo para darle tiempo de procesador a otro proceso.

Vector de interrupciones

- Cada entrada del vector contiene
 - La dirección de la rutina de tratamiento de esa interrupción
 - El vector está indexado por el número de interrupción
- Se transfiere el control (se ejecuta) las instrucciones a las que apunta la entrada en el vector

Sistemas operativos monotarea

- **Monotarea o monoproceso**

- Solo existe un proceso a cada instante en el procesador
- El segundo proceso no puede ejecutarse hasta que el primero halla finalizado completamente
- La memoria RAM la ocupa el S.O y el proceso en ejecución.
- MS-DOS

- **Multitarea o multiproceso**

- Permite la coexistencia de varios procesos activos a la vez
- El S.O (planificador y despachador) selecciona un proceso para ejecutarse
- Se requiere **cambio de contexto** entre proceso y proceso.

Referencias

- Carretero Pérez, J., García Carballeira, F., de Miguel Anasagasti, P., & Pérez Costoya, F. (2001). Procesos. In *Sistemas operativos. Una Visión Aplicada* (pp. 77–160). McGraw Hill.
- Silberschatz, A., Baer Galvin, P., & Gagne, G. (2018). Process Management. In *Operating Systems Concepts* (10th ed., pp. 105–115). John Wiley & Sons, Inc.
- Tanenbaum, A. S. (2009). Introducción. In *Sistemas Operativos Modernos* (3rd ed., pp. 3–75). Pearson Educación.