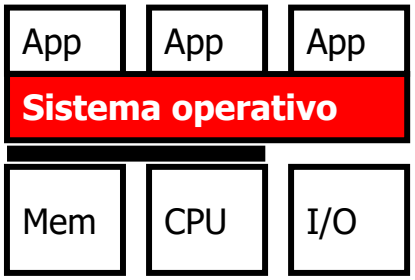
Tema 4. El sistema operativo

Fundamentos de Computadores

Grado de Ciencia e Ingeniería de Datos

Curso 2022/2023

Recap: Sistema operativo



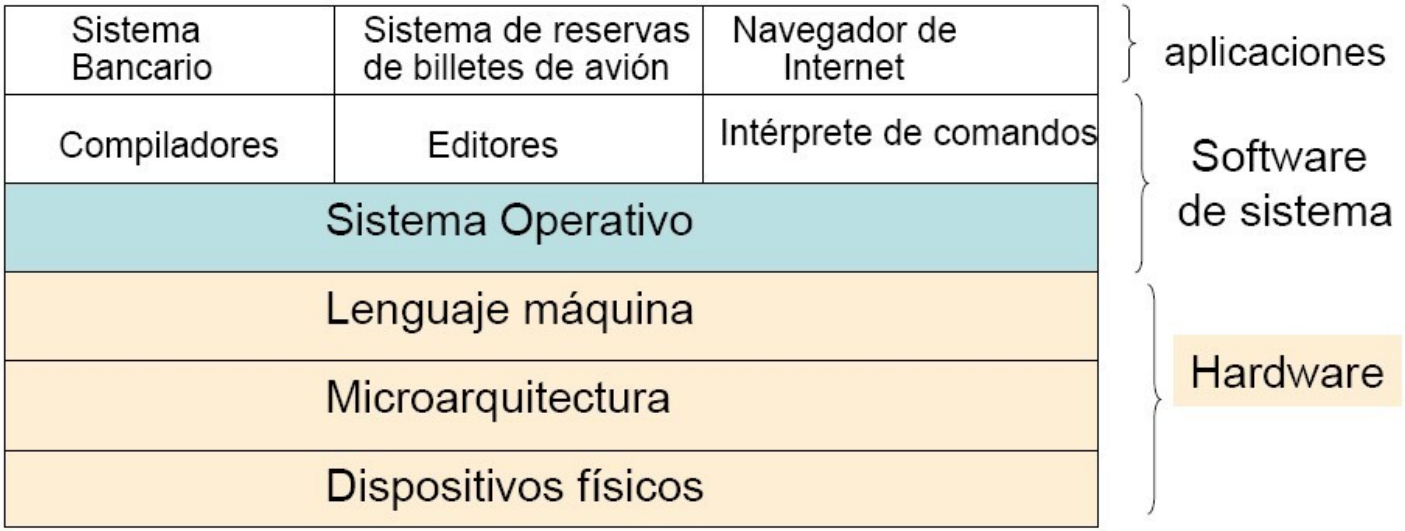
- Las aplicaciones acceden a la E/S (y otros servicios) por medio del **sistema operativo**:
  - Software con privilegios para acceder a todo el hardware
  - Abstrae la complejidad y particularidades de la diversidad de dispositivos periféricos de E/S
  - Virtualiza el hardware: aísla a unos programas de otros
  - Cada aplicación ignora la presencia de otras
  - Simplifica la programación, aporta robustez y seguridad
  - Ejemplos: Windows, Linux, MacOS, Android, UNIX...

Índice

Índice

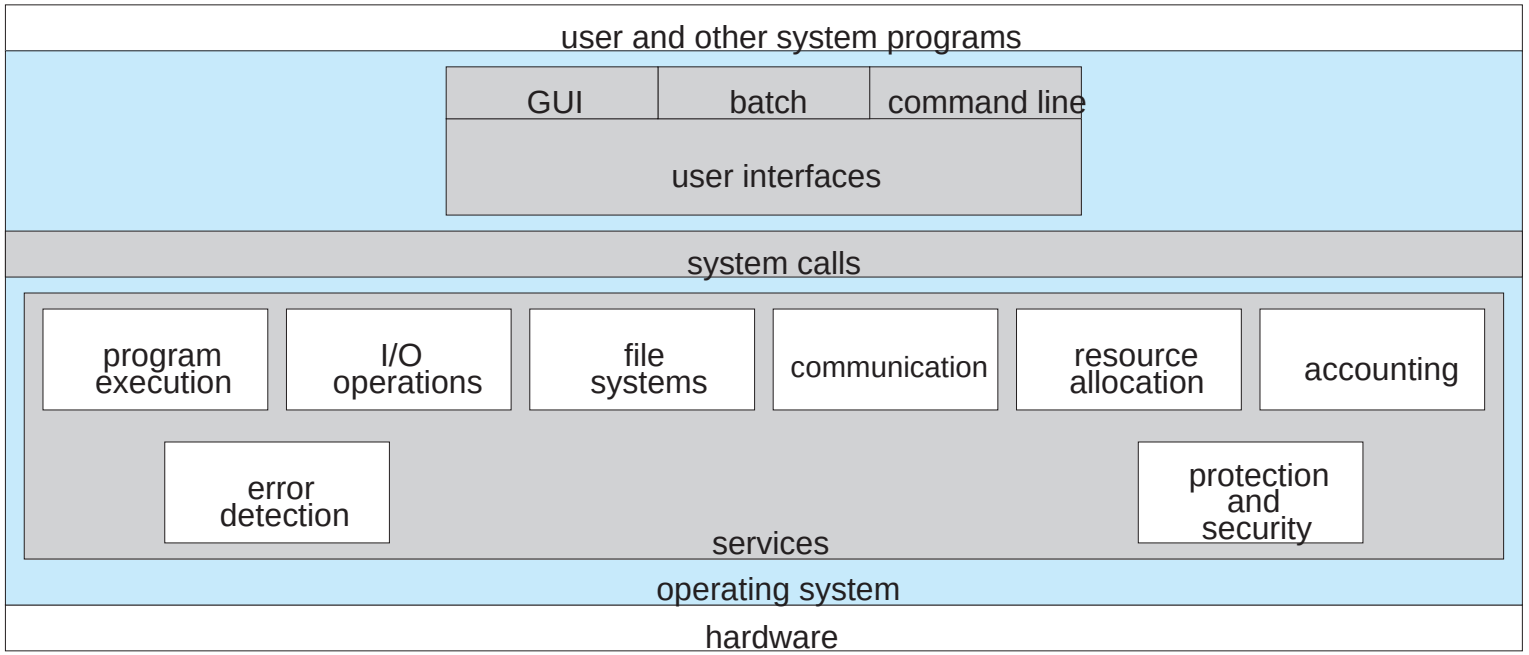
Ubicación del sistema operativo

- El sistema operativo (SO) se sitúa entre el hardware “desnudo” y las aplicaciones de usuario:
  - Ofrece servicios que permiten un uso más sencillo del computador:
    - Para usuarios convencionales:
      - Interfaces gráficos, lanzamiento de programas, manipulación del sistema de ficheros, E/S, etc.).
    - Para programadores: Llamadas al sistema, abstracciones (proceso, VM, ficheros)
  - Administra los recursos hardware (CPU, memoria y E/S):
    - Gestiona, controla y coordina que todos los recursos se utilicen de forma eficiente en beneficio de las aplicaciones



Servicios que el SO ofrece al usuario

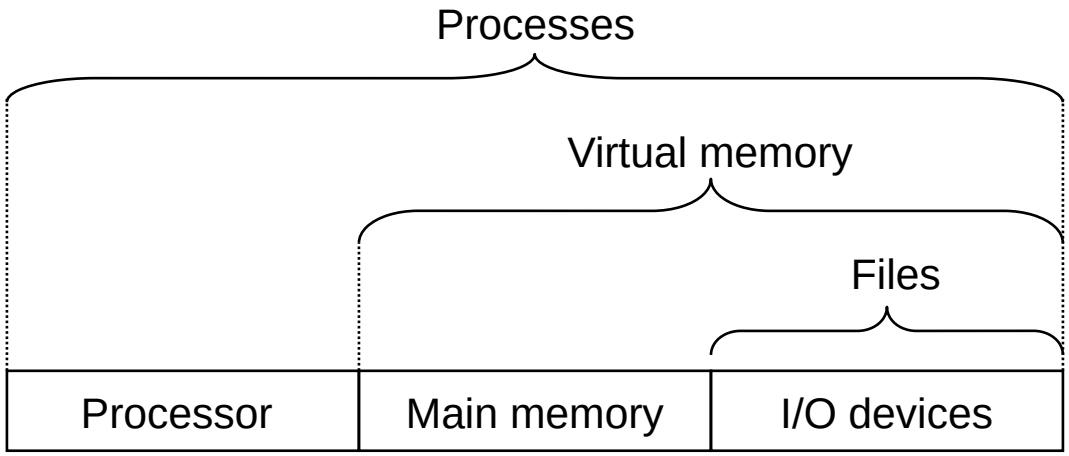
- Interfaces de usuario
- Ejecución de programas
- Operaciones de E/S
- Sistemas de ficheros
- Comunicación
- Asignación de recursos
- Contabilidad
- Detección de errores
- Protección y seguridad



Silberschatz, Galvin and Gagne. Operating System Concepts. 9th Edition

Abstracciones que ofrece el SO

- **Ficheros**
  - Datos y… abstracción de los dispositivos de E/S
- **Memoria virtual:**
  - Abstracción de la memoria principal y los dispositivos de E/S de almacenamiento (discos)
- **Procesos**:
  - Abstracción de procesador + memoria + E/S para un programa



Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Procesos

- Abstracción del SO para un **programa en ejecución**.
  - Procesador + memoria + E/S
- El SO proporciona al programa la ilusión de:
  - Ser el único programa en ejecución en el sistema
  - Tener el uso exclusivo del hardware
    - La CPU y toda la memoria para sí mismo, así como los dispositivos E/S
  - Ejecutar su código sin interrupciones
    - El procesador ejecuta sus instrucciones una tras otra
  - Su código y datos del programa parecen ser lo único que hay en la memoria del sistema.

Procesos

- Los procesos solicitan acciones sobre el hardware a través de las **llamadas al sistema** que ofrece el SO
- Puede haber varios procesos ejecutándose de forma **concurrente**
- La CPU se puede compartir entre procesos concurrentes mediante **cambios de contexto**
  - El **contexto del proceso** incluye su contador de programa (PC), los datos en los registros y los datos en memoria
  - Al cambiar de proceso en ejecución, se guarda el contexto actual y se carga el contexto del nuevo proceso
- Un proceso puede tener varios flujos de ejecución: **hilos**
  - Los hilos comparten el mismo código y datos globales del proceso
  - Permiten la ejecución más rápida de aplicaciones cuando hay un soporte nativo de la CPU (múltiples procesadores o múltiples hilos)

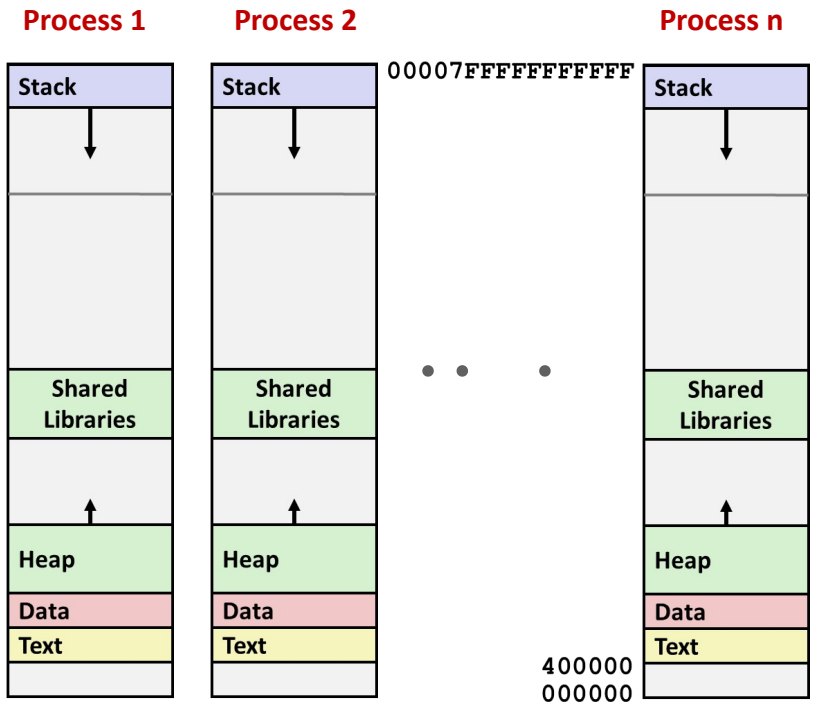
Hilos

- Normalmente pensamos en los procesos como si cada uno tuviera un único flujo de ejecución
  - A lo largo de este tema asumimos que un proceso ejecuta en cada instante instrucciones de un único punto del código del programa
- Realidad: un proceso puede tener varios flujos de ejecución → **hilos** (*threads*)
  - Es posible ejecutar diferentes partes del mismo programa simultáneamente
    - O la mismas instrucciones del programa, pero operando con datos diferentes
  - Todos los hilos se ejecutan en el contexto del proceso
  - Comparten el mismo código y los datos globales
  - Cada hilo tiene sus propios datos locales (hay una zona de *pila* por hilo)
    - Variables locales de los procedimientos en ejecución, parámetros, etc.
- Modelo programación multi-hilo: cada vez más extendido
  - Resulta más fácil y eficiente compartir datos entre hilos que entre procesos
    - Aunque es necesario *sincronizar* el acceso a los datos compartidos para la corrección
  - Hacer que un programa se ejecute más rápido cuando hay múltiples procesadores disponibles (p.ej. en un procesador multinúcleo) o soporte *multi-threading*

Memoria Virtual

- Abstracción sobre la memoria principal para dar **a cada proceso** la *ilusión* de que tiene para sí **toda la memoria en exclusiva**

- Proporciona a cada proceso un **espacio de direcciones virtual**
  - Grande, uniforme, privado. Mucho más grande que el tamaño de la memoria principal
  - Simplemente un array de bytes que el programa puede subdividir en diferentes unidades de almacenamiento
  - Funciona gracias a una sofisticada interacción entre el hardware y el software del SO
    - Incluyendo la traducción en hardware de cada dirección generada por el procesador
- Idea básica:
  - Almacenar el contenido de la memoria virtual de cada proceso en disco
  - Usar memoria principal como caché de disco



Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Ficheros

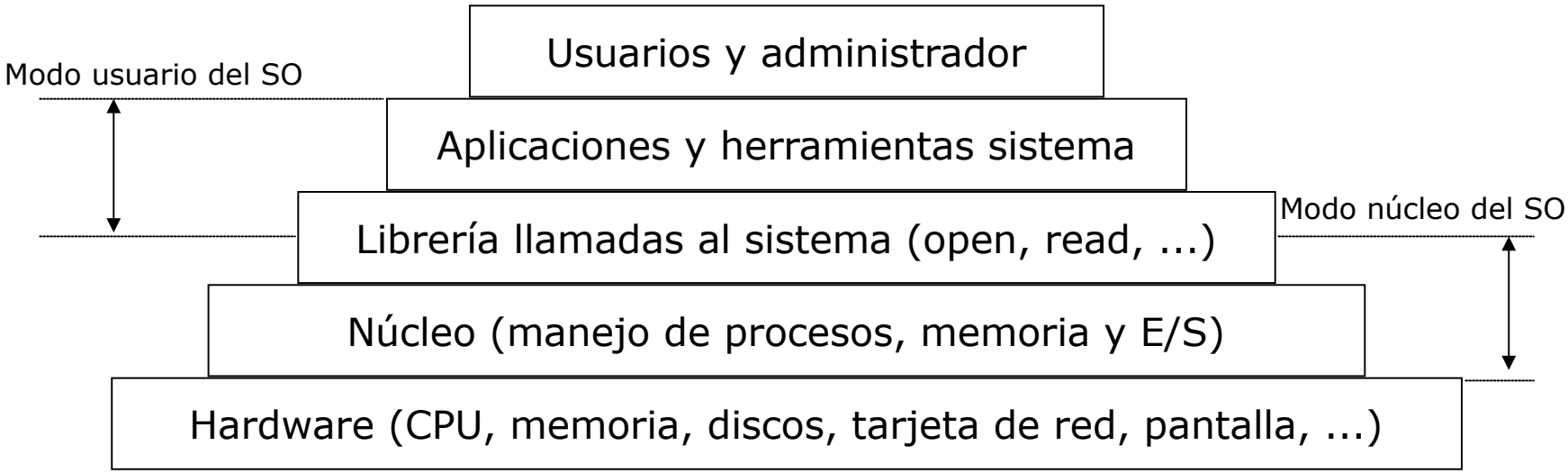
- Secuencia de bytes (nada más y nada menos)
  - Unidad lógica de almacenamiento para datos que necesitan persistencia (discos)
    - Identificado por su ruta
      - Nombre del fichero y ubicación en el árbol de directorios
  - Contiene datos arbitrarios relacionados, organizados por su creador
  - Válido para cualquier clase de datos: texto, enteros, reales, imágenes, sonido, vídeo, librerías, instrucciones máquina...
- En los sistemas UNIX todos los dispositivos de E/S se modelan como ficheros:
  - Discos, teclados, pantallas, interfaces de red, etc.
  - Proporciona una visión uniforme de los dispositivos de E/S
  - Gracias a ello la programación es ***independiente del dispositivo***
    - Ejemplo: Los programadores crean aplicaciones que manipulan el contenido de los ficheros en disco, sin prestar atención a la tecnología concreta de almacenamiento (magnético, óptico, estado sólido, etc.). El mismo programa funcionará en diferentes sistemas que utilizan diferentes tecnologías de disco.

Interfaces de usuario: GUI vs. CLI

- Interfaz de usuario: Permite al usuario dar órdenes al sistema. Dos tipos:
  - **Gráfico** (**GUI**, *Graphical User Interface*): usa ratón, ventanas, iconos, menús, paneles, lanzadores, atajos de teclado, etc.
    - Presentan a los usuarios una visión sencilla e intuitiva del sistema.
    - Basados en un gestor de ventanas que permite arrancar y terminar aplicaciones, trabajar simultáneamente con varias actividades, manipular ficheros y directorios, configurar el sistema, etc.
  - De **línea de comandos** (**CLI**, *command line interface*): usa un terminal con órdenes tecleadas, con diversos parámetros y opciones:
    - Los intérpretes de comandos pueden ejecutar
      - Comandos internos: reconocidas y ejecutadas por el propio intérprete
        - Ejemplo: *cd*, *exit*, *alias*
      - Comandos externos: programas almacenados en su propio fichero ejecutable, y que el programa intérprete simplemente busca en disco para ejecutarlo cuando se invocan
        - La gran mayoría: *ls*, *cp*, *mv*, *find*, *firefox*, *libreoffice*, etc.

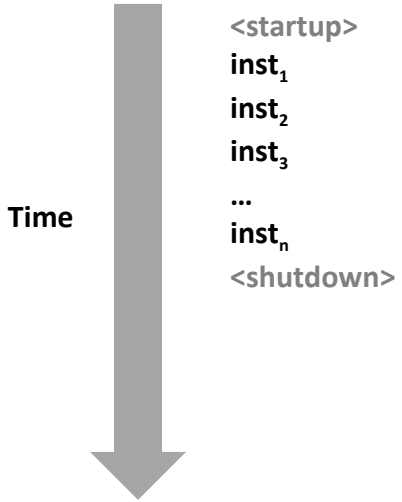
Visiones de un SO

- Dos visiones complementarias de un SO:
  - Como máquina extendida:
    - "Enmascara" la complejidad del hardware.
    - Ofrece un uso más sencillo del computador:
      - Para usuarios: Programas de sistema (interfaces gráficos, intérpretes de comandos, editores, navegadores, sistemas de ficheros).
      - Para programadores: Llamadas al sistema (subrutinas de fácil uso para manejar memoria, procesos, ficheros, etc.).
  - Como administrador de recursos:
    - Controla y coordina que todos los recursos hardware se manejen de forma eficiente:
      - CPU, Memoria, Dispositivos.



Flujo de control regular y excepcional

- Cada núcleo del procesador simplemente lee y ejecuta una **secuencia de instrucciones**, **de una en una**
  - Desde un punto de vista arquitectural (lo que se observa a nivel externo), las instrucciones se ejecutan de una en una, pero a nivel interno (microarquitectura), la CPU puede usar ejecución paralela, fuera de orden
  - Esta secuencia de instrucciones es **el flujo de control de la CPU**
- Mecanismos para cambiar el flujo de control (secuencial) en un programa...
  - Instrucciones de salto condicional e incondicional
  - Instrucciones para llamar a procedimientos y retornar tras llamada
- ...permiten reaccionar a cambios en el estado del programa
  - Cambios en los valores de las variables
- ¿Cómo reaccionar ante los cambios en el estado del sistema?
  - Datos que llegan a un adaptador de red, o procedentes del disco
  - Instrucción que divide por cero
  - El usuario teclea Ctrl-C en el tecleado para interrumpir al proceso
  - El temporizador del sistema se agota
- El sistema necesita mecanismos para el **flujo de control excepcional**
  - Mediante combinación de soporte hardware y el **software del SO**



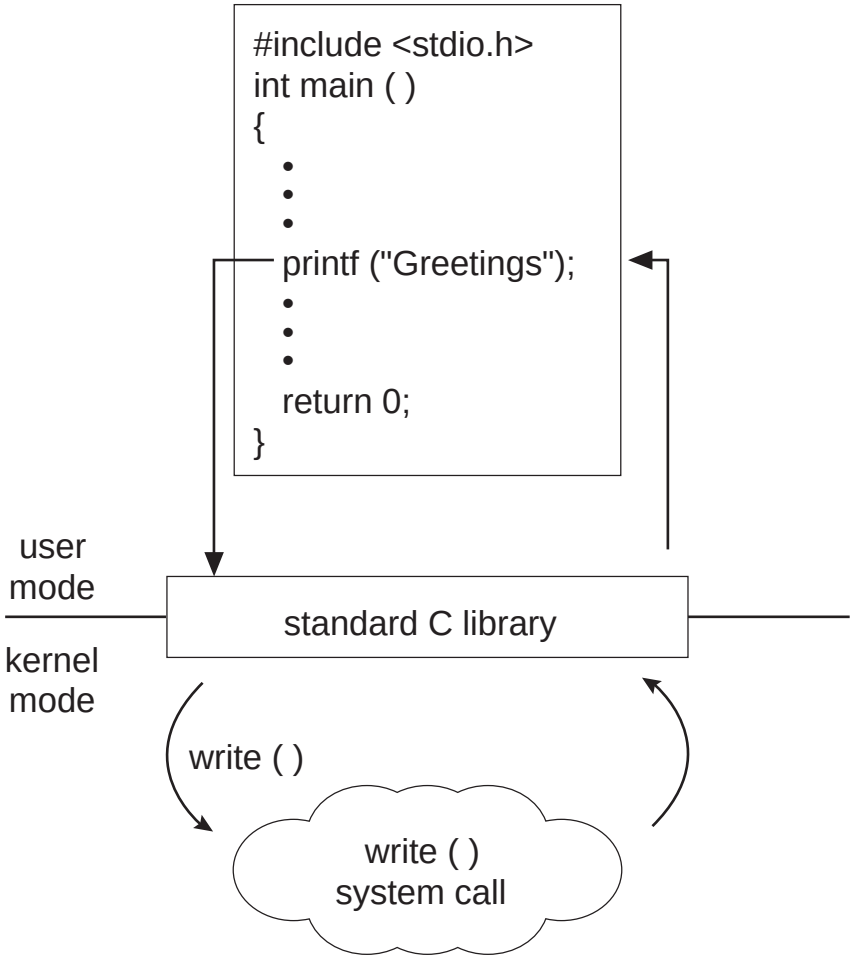
¿Cuándo se ejecuta el código del SO?

- **Excepción**: transferencia del flujo de control al núcleo del SO en respuesta a algún evento. Dos tipos: asíncronas y síncronas
  - Por ejemplo, un cambio en el estado del procesador
- Excepciones **asíncronas**: causadas por **eventos externos** al procesador
  - También llamadas **interrupciones hardware**:
  - Generadas al activar el pin de interrupción del procesador
    - Ejemplo: Interrupción del temporizador (cada pocos ms, un chip temporizador externo dispara una interrupción, usado por el kernel para recuperar el control de los programas de usuario
    - Ejemplo: Interrupción de dispositivo de E/S (pulsar Ctrl-C, paquete que llega a la interfaz de red...)
- Excepciones **síncronas**: causadas por eventos que ocurren como resultado de la ejecución de una instrucción del programa
  - **Llamadas al sistema**: Intencionada, usada por el programa para solicitar a los servicios del núcleo del SO
    - A través de la biblioteca del sistema (p.ej. Biblioteca estándar de C)
  - **Fallos**: involuntarios pero posiblemente recuperables. Ejemplos: fallos de página, fallos de protección, excepciones de PF
  - **Abortos**: involuntario e irrecuperable. Ejemplo: Instrucción ilegal, error de paridad, etc.

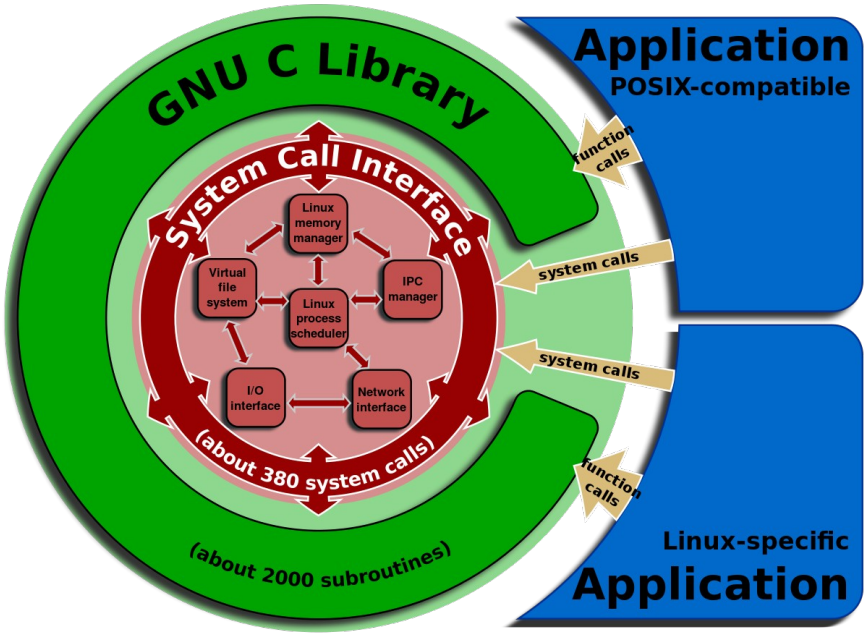
Llamadas al sistema: Interfaz programas y SO

- Llamadas al sistema definen el interfaz entre programas de usuario y el SO
  - Parámetros, valores devueltos, comportamiento deseado, etc.
  - Existen estándares para facilitar la portabilidad entre ciertos sistemas: POSIX, librería estándar de C, etc.
- **Tipos** de llamadas al sistema:
  - Procesos: creación de nuevos procesos, terminación de procesos (ídem para hilos), etc.
  - Acceso a dispositivos: Apertura y cierre de ficheros, lectura y escritura (ídem para acceso a la red, teclado, pantalla, etc.)
  - Gestión de la memoria: Solicitud dinámica de espacio, liberación de espacio, etc.
  - Manipulación de sistema de ficheros: creación y borrado de directorios, movimiento por directorios, manipulación de permisos, acceso a metadatos (dueño, fecha, hora, tamaño de ficheros, ...)
  - Otros: sincronización y comunicación entre procesos, etc.

Llamadas al sistema y biblioteca del sistema



Silberschatz, Galvin and Gagne. Operating System Concepts. 9th Edition



Shmuel Csaba Otto Traian, https://upload.wikimedia.org/wikipedia/commons/4/45/Linux_kernel_System_Call_Interface_and_glibc.svg

Llamadas al sistema. Ejemplo (I)

- Programa en C que abre un fichero, lee una cadena del mismo y la imprime por pantalla

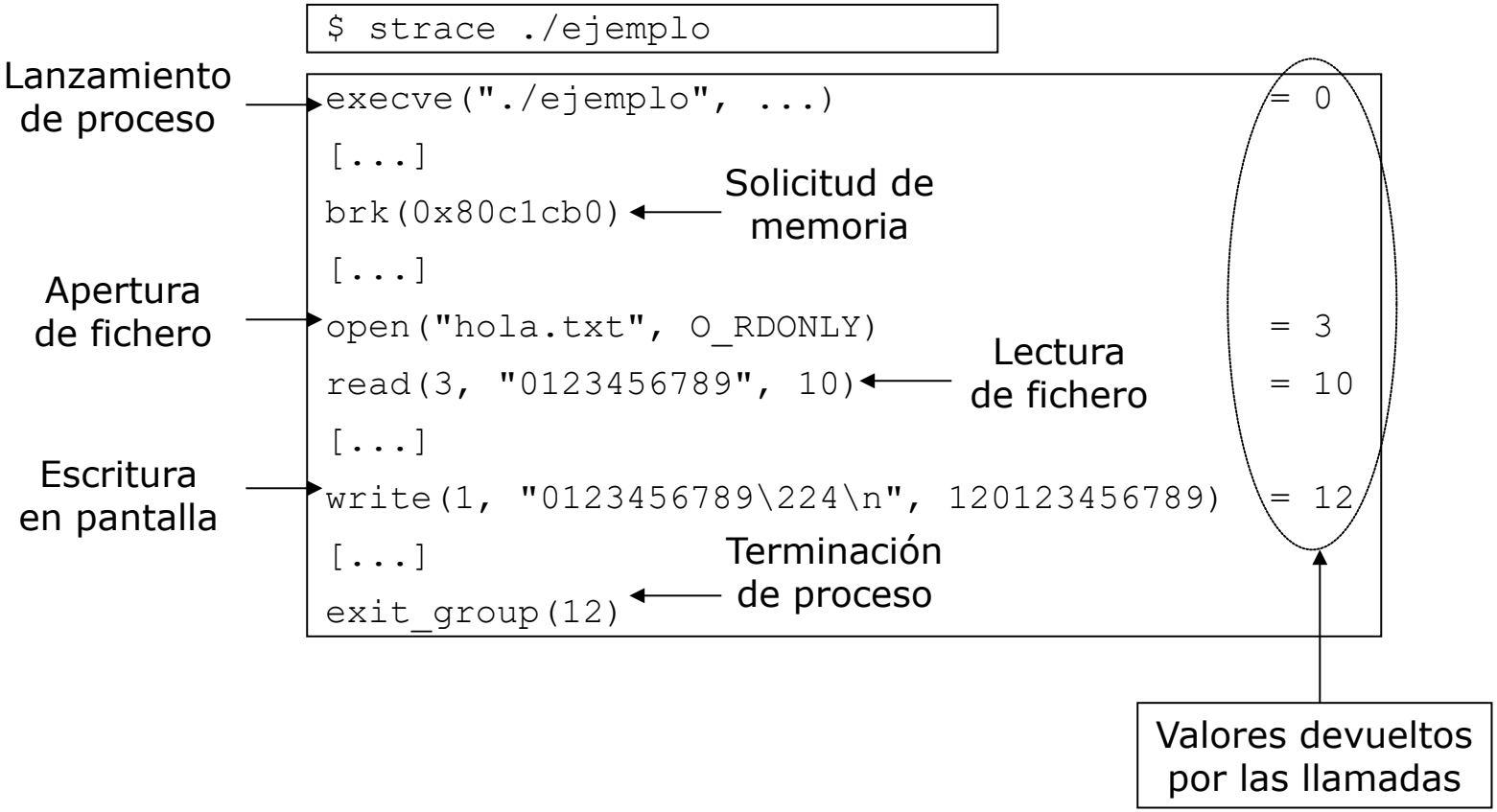
```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int main()
{
    char cadena[100];
    int fd;

    fd = open("hola.txt", O_RDONLY);
    read(fd,cadena,10);
    cadena[11] = 0;
    printf("%s\n",cadena);
}
```

  - Compilamos para generar programa ejecutable:

```
$ gcc ejemplo.c -static -o ejemplo
```

Llamadas al sistema. Ejemplo (II)

- Ejemplo:
  - `strace` permite "tracear" las llamadas al sistema

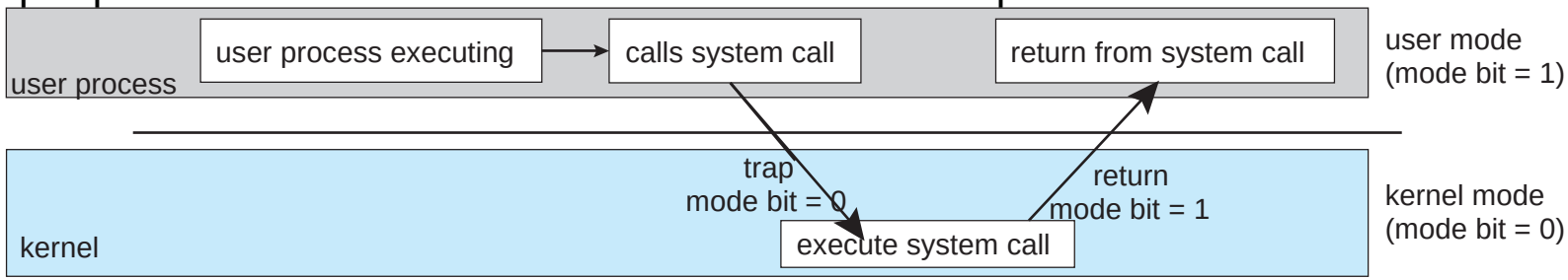


Modos de ejecución: usuario vs núcleo

- Es necesario soporte hardware para garantizar la seguridad del sistema:
  - Dos modos de funcionamiento de la CPU: privilegiado vs no privilegiado
  - Objetivo: evitar que código de usuario ejecute determinadas instrucciones ("privilegiadas")
- **Modo usuario** (*no privilegiado*):
  - Permite sólo instrucciones "normales" de la CPU (procesamiento, acceso a registros/memoria, saltos a subrutinas, bucles, etc.).
  - Prohíbe completamente cualquier tipo de acceso directo al hardware:
    - No se permite ningún tipo de instrucción de E/S
    - No se permite acceso a direcciones de memoria física
  - Es el modo usado normalmente por los procesos de usuario.
- **Modo núcleo** (*privilegiado*):
  - Permite ejecutar todo el repertorio de instrucciones de la CPU, incluyendo el acceso al hw.
  - Sólo el núcleo (SO) tiene permiso para ejecutarse en este modo

Transición modo usuario ←→ modo núcleo

- Modo usuario → modo núcleo:
  - **Llamadas al sistema**: El proceso llama explícitamente a un servicio del SO
    - A veces llamadas "interrupciones software": el programa se interrumpe a sí mismo de forma *síncrona*
    - Similar a una llamada a una subrutina (pero en este caso del SO)
  - **Interrupciones hardware**: Algún evento reclama la atención inmediata del SO
  - Un evento externo *asíncrono* importante señalado por un dispositivo de E/S → RSI *(rutina servicio interrupción)*
    - Reloj marca final de quantum del proceso, termina una operación de disco que se lanzó anteriormente, etc.)
  - **Excepciones, fallos**: Un error en el proceso en ejecución que le impide continuar
    - Provocado por una instrucción errónea, error numérico, acceso a memoria indebido, etc.
- Modo núcleo → modo usuario:
  - Regreso de una llamada al sistema o tras finalizar una RSI
  - Siempre precedido de la **restauración del contexto** del proceso a reanudar



Silberschatz, Galvin and Gagne. Operating System Concepts. 9th Edition

Arranque del sistema operativo

- Al encender un ordenador, la CPU ejecuta inmediatamente un programa en ROM (*iniciador ROM*), que:
  - Realiza un rápido autodiagnóstico del *hw* (memoria, tarjeta gráfica, discos, teclado, etc.) y comprueba que todo esté correcto.
  - Lee del disco y ejecuta el programa llamado *cargador* (que permite seleccionar, en su caso, entre varios SO presentes), y que se encarga de cargar y dejar residente el núcleo básico del SO (*kernel*) en memoria.
  - El *kernel* toma el control y establece sus estructuras internas básicas (tabla de procesos, áreas de memoria, E/S, etc.).
  - Una vez inicializado el núcleo, se lanza un proceso llamado `init`. Éste empieza a lanzar procesos auxiliares y *demonios* (impresión, red, servicios de conexión remota, etc.) según esté configurado
  - Finalmente, lanza uno (o varios) procesos de login, que permiten a un usuario autenticarse y empezar a trabajar.

Núcleo del sistema operativo

- Núcleo (kernel) del SO:
  - La parte más interna del SO, la más importante, y la primera en arrancar
  - La parte del SO que reside en memoria continuamente
  - No es un proceso separado, sino que se ejecuta como parte de un proceso existente
  - En Linux, el espacio del kernel está siempre presente en memoria y mapea las mismas direcciones físicas para todos los procesos
  - El código y los datos del kernel están siempre direccionables, listos para manejar interrupciones y llamadas al sistema en cualquier momento

Linux: Características

- **Clon de Unix**, iniciado por **Linus Torvalds** (1991)
- **Open source**: Código fuente disponible, licencia GPL
  - Puede usarse, modificarse y distribuirse libremente
  - Torvalds dirige la evolución de la rama principal.
- Multiplataforma:
  - Corre en CPUs Intel, AMD, PowerPC, ARM, etc., de 32/64 bits.
  - Muy portable por estar casi íntegramente **escrito en C**.
- Multiusuario/multitarea
  - También aprovecha arquitecturas SMP y multicore
- Protección máxima entre procesos
- Soporte para múltiples sistemas de ficheros:
  - Ext4 (nativo), XFS, FAT, NTFS, ISO9660 (CD), UDF (DVD), ...
- Soporte para múltiples protocolos de red: TCP, UDP, IPv4, IPv6, etc.
- Soporte para infinidad de dispositivos de E/S
- Múltiples **distribuciones**: Ubuntu, Fedora, Debian, SUSe, …
- Distribución = núcleo de Linux + software variado + sistema configuración
- Presente en todos los segmentos. Algunas estadísticas: https://truelist.co/blog/linux-statistics/
  - 47% programadores, 39.2% websites, 85% smartphones, 2% usuarios, 100% supercomputadores en Top500

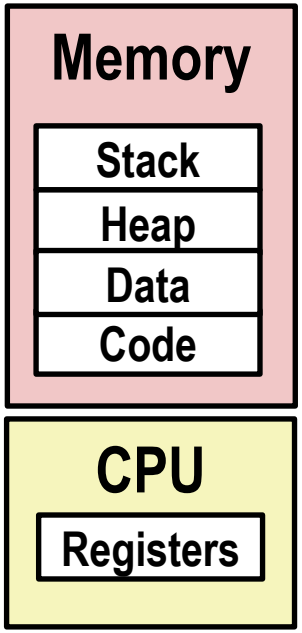
Índice

Programa *vs* Proceso

- **Programa**: Código ejecutable almacenado en disco
  - Una entidad pasiva: fichero que contiene instrucciones guardadas en disco (fichero ejecutable)
  - Puede ser cargado en memoria para ser ejecutado. Concepto estático (sin evolución ni estado).
- **Proceso**: Programa en ejecución, con recursos asignados (CPU, memoria, ficheros...)
  - Es un concepto dinámico, con estado cambiante: Contador de programa (PC), estado actual de los datos en memoria, el contenido de los registros del procesador, etc.
  - Entidad activa, con un PC (siguiente instrucción a ejecutar) y un conjunto de recursos asociados
  - Es la **unidad de trabajo** del SO, quien se encarga de crear/eliminar/suspender/reanudar/... procesos
  - Mucho más que simplemente el código del programa (sección de código):
    - Sección de datos donde se guardan las variables globales del programa
    - Pila (*stack*):, datos temporales (parámetros a procedimientos, direcciones de retorno, variables locales...)
    - Montón (*heap*): sección de la memoria reservada dinámicamente durante la ejecución del proceso
- Programa → proceso: cuando un fichero ejecutable se carga en memoria y ejecuta
  - Doble click en el fichero ejecutable, escribiendo el nombre en la línea de comandos, etc.
- Aunque dos procesos pueden estar asociados al mismo programa, se consideran dos secuencias separadas de ejecución
  - Ejemplo: Múltiples instancias del navegador, editor o visor de PDF → cada una es un proceso distinto, aunque todas tienen secciones de código equivalentes, los datos, la pila, etc. de cada una son distintos

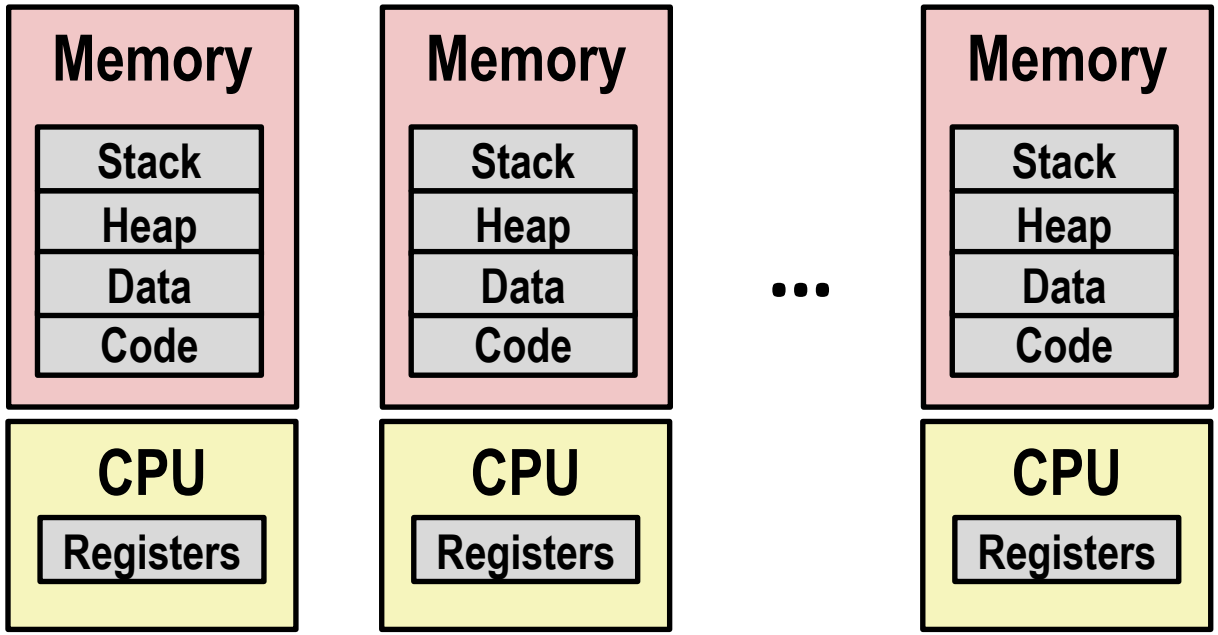
Abstracciones ofrecidas por el proceso

- El concepto de proceso proporciona a cada programa dos **abstracciones** clave:
  - Flujo de control lógico
    - Cada programa parece tener uso exclusivo de la CPU
    - Ofrecido gracias a un mecanismo del SO llamado cambio de contexto (*context switching*)
  - Espacio de direcciones privado
    - Cada programa parece tener uso exclusivo de la memoria principal
    - Ofrecido gracias un mecanismo del SO llamado memoria virtual



Multiprogramación: la ilusión

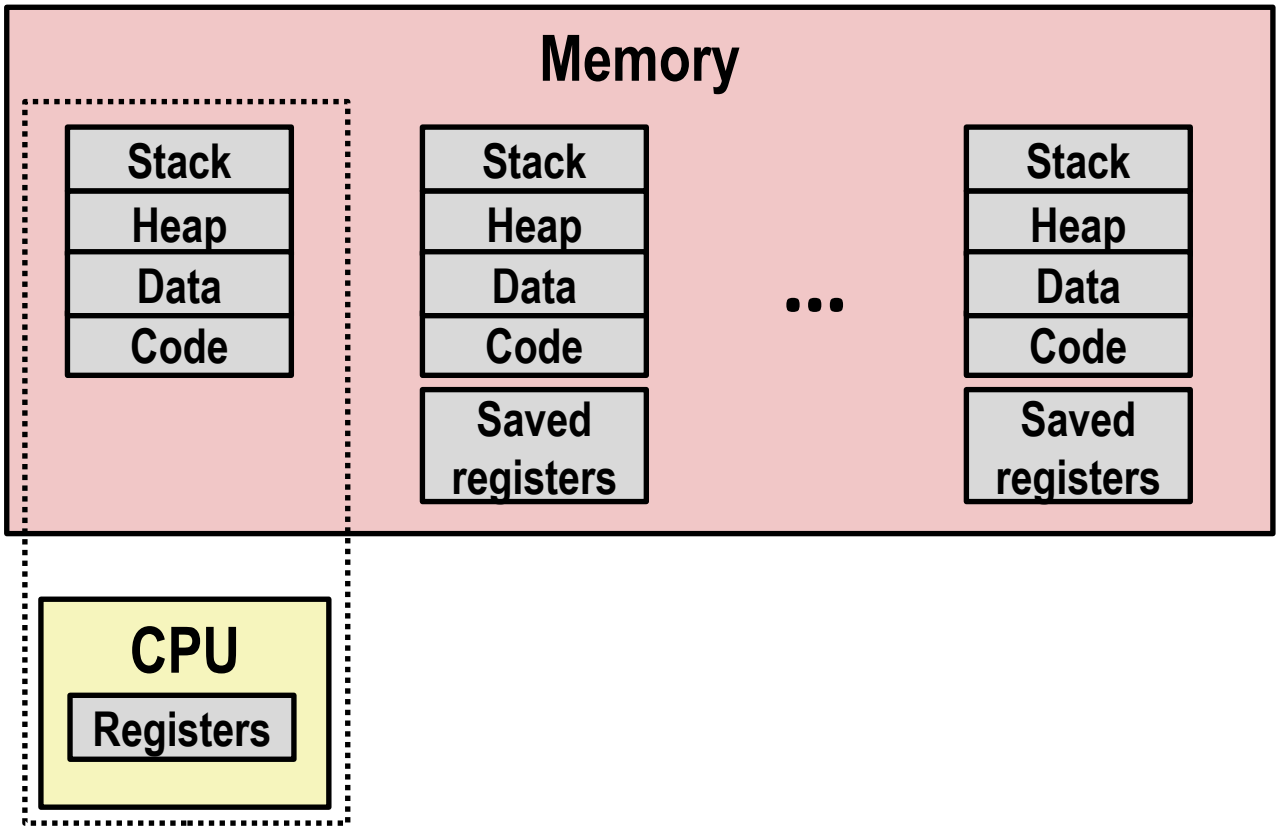
- El computador ejecuta muchos procesos simultáneamente
  - Aplicaciones de uno o varios usuarios
    - Navegador, cliente de correo, editores de texto, ...
  - Tareas en segundo plano
  - Procesos de monitorización de los dispositivos de E/S (red, etc.)



Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

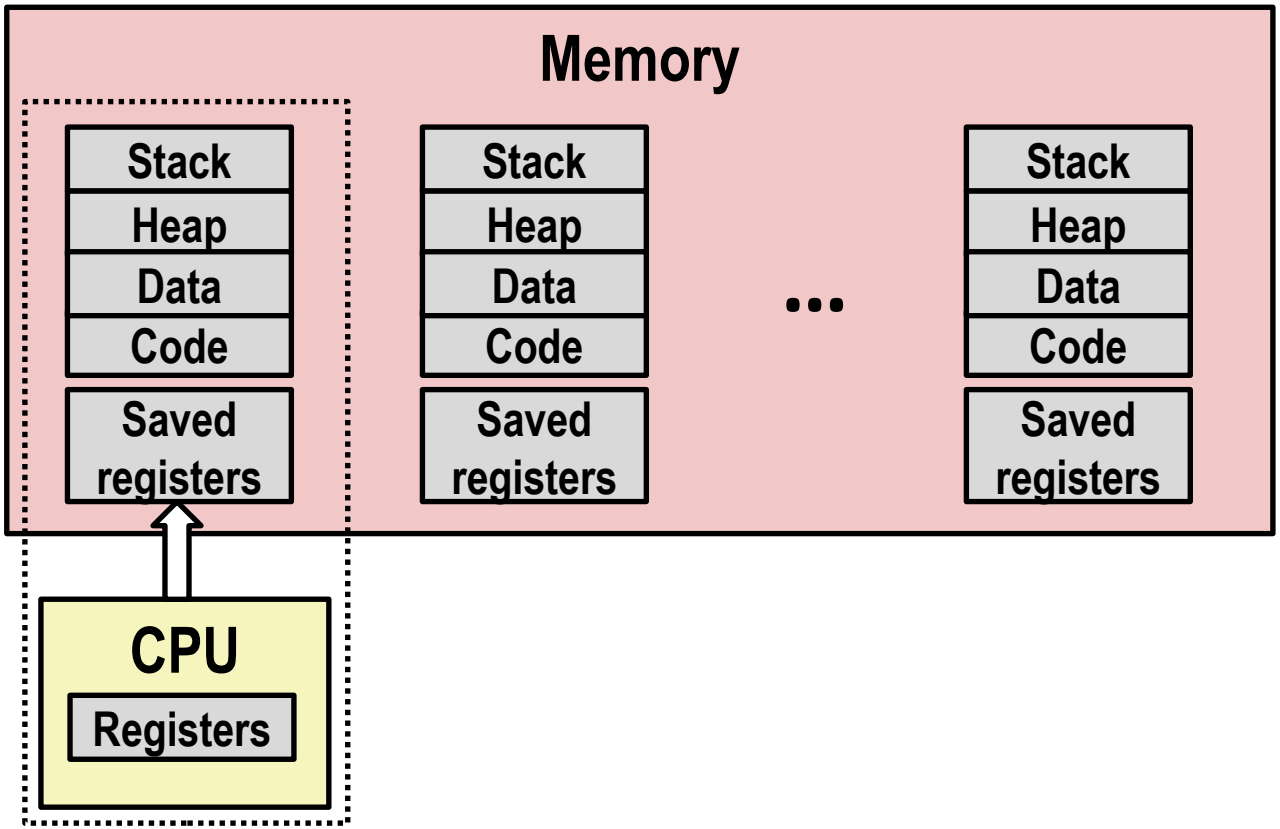
Multiprogramación: la realidad

- Un único procesador ejecuta múltiples procesos concurrentemente
  - Las ejecuciones de los procesos se entrelazan (multitarea), alternando el uso de la CPU
  - El sistema de memoria virtual gestiona los espacios de direcciones de cada proceso
  - Los valores de los registros para los procesos que no están en ejecución se guardan en memoria



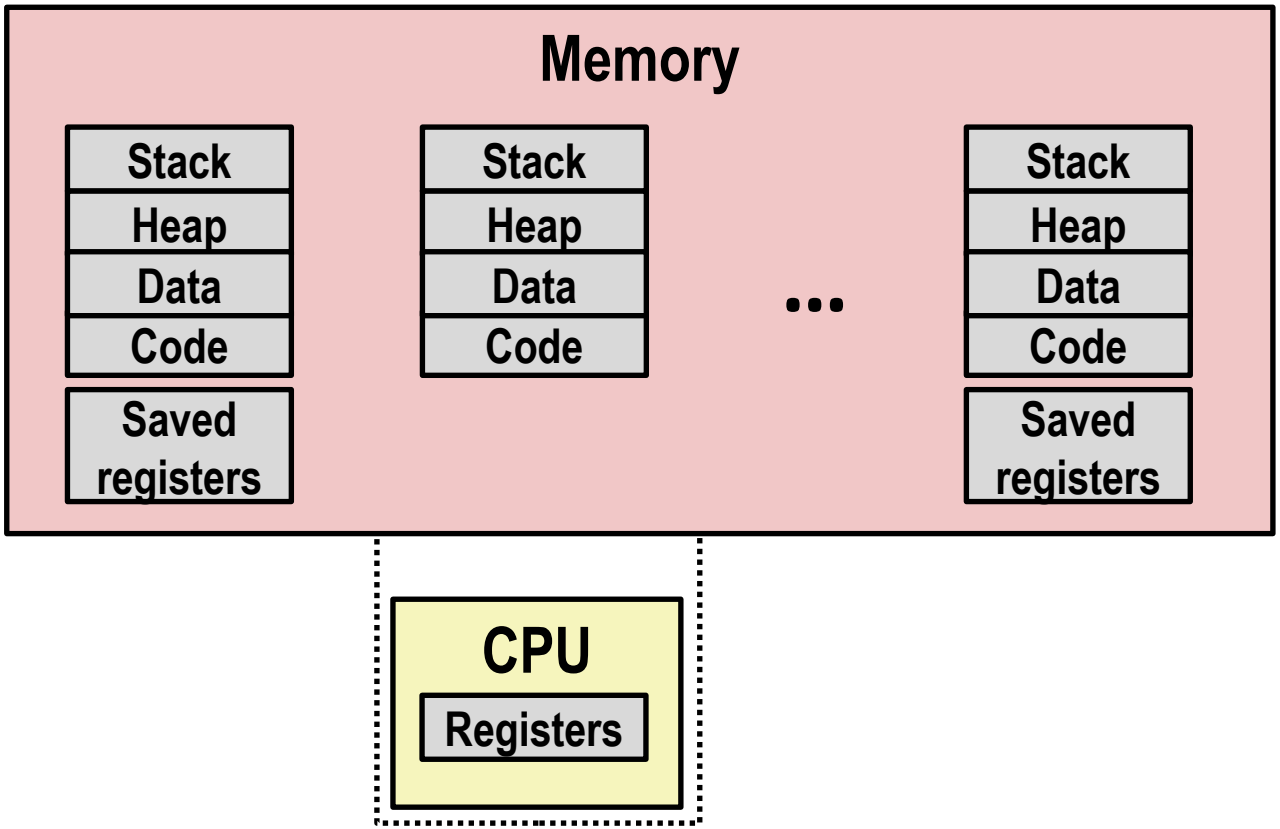
Multiprogramación: la realidad

- Al interrumpir un proceso que está en ejecución (usando la CPU), el SO debe primero guardar los valores de los registros de la CPU en ese instante
  - Para poder reanudarlo posteriormente en el mismo punto donde su ejecución se interrumpió
  - El kernel del SO mantiene en su memoria los registros guardados de todos los procesos que no se están ejecutando



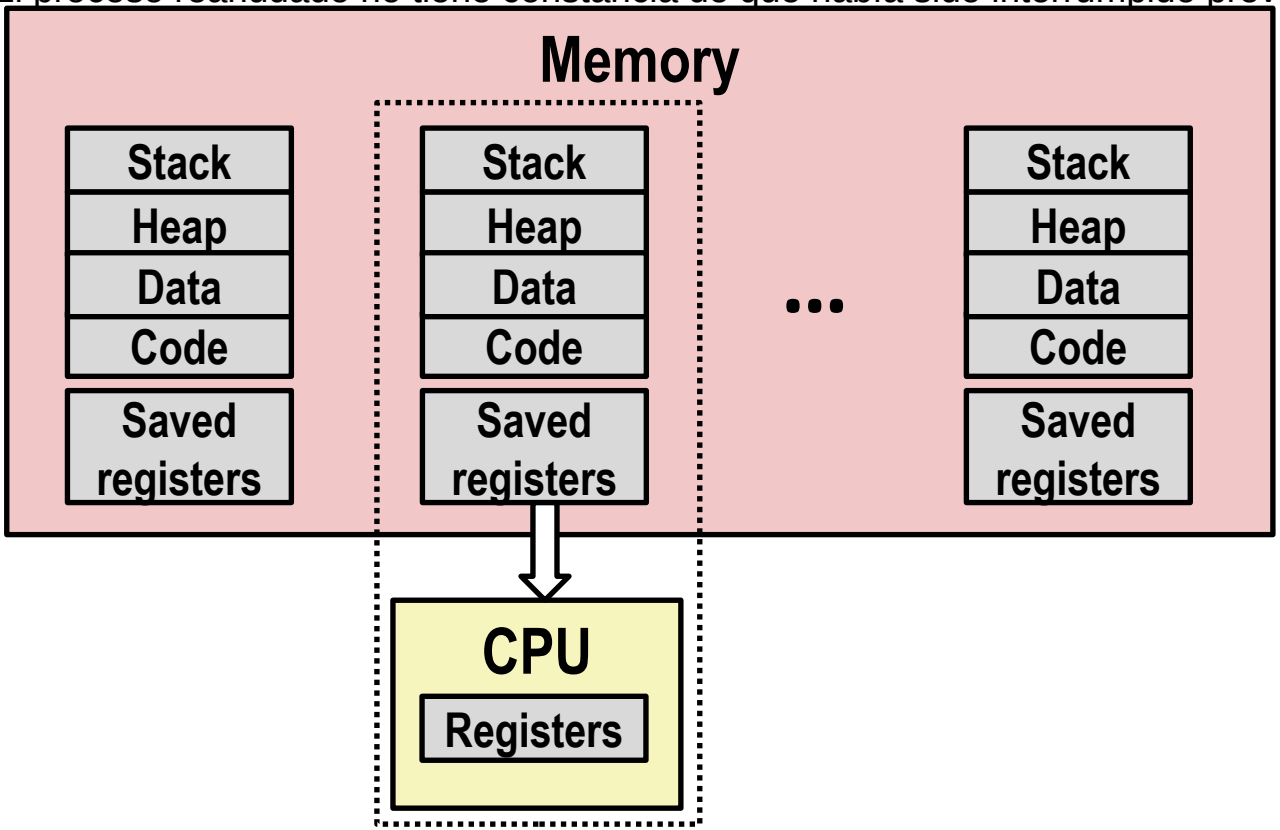
Multiprogramación: la realidad

- El SO decide mediante el planificador de procesos (*scheduler*) cuál es el siguiente proceso que debe pasar a ejecutarse
  - Elige de entre todos los procesos que están listos para ejecutarse
    - Unos procesos están bloqueados en espera de alguna operación de E/S que han solicitado
    - Otros simplemente perdieron la CPU para dejar que otros procesos se ejecuten también
  - El SO asigna y gestiona la prioridad de cada proceso para conseguir un reparto equitativo



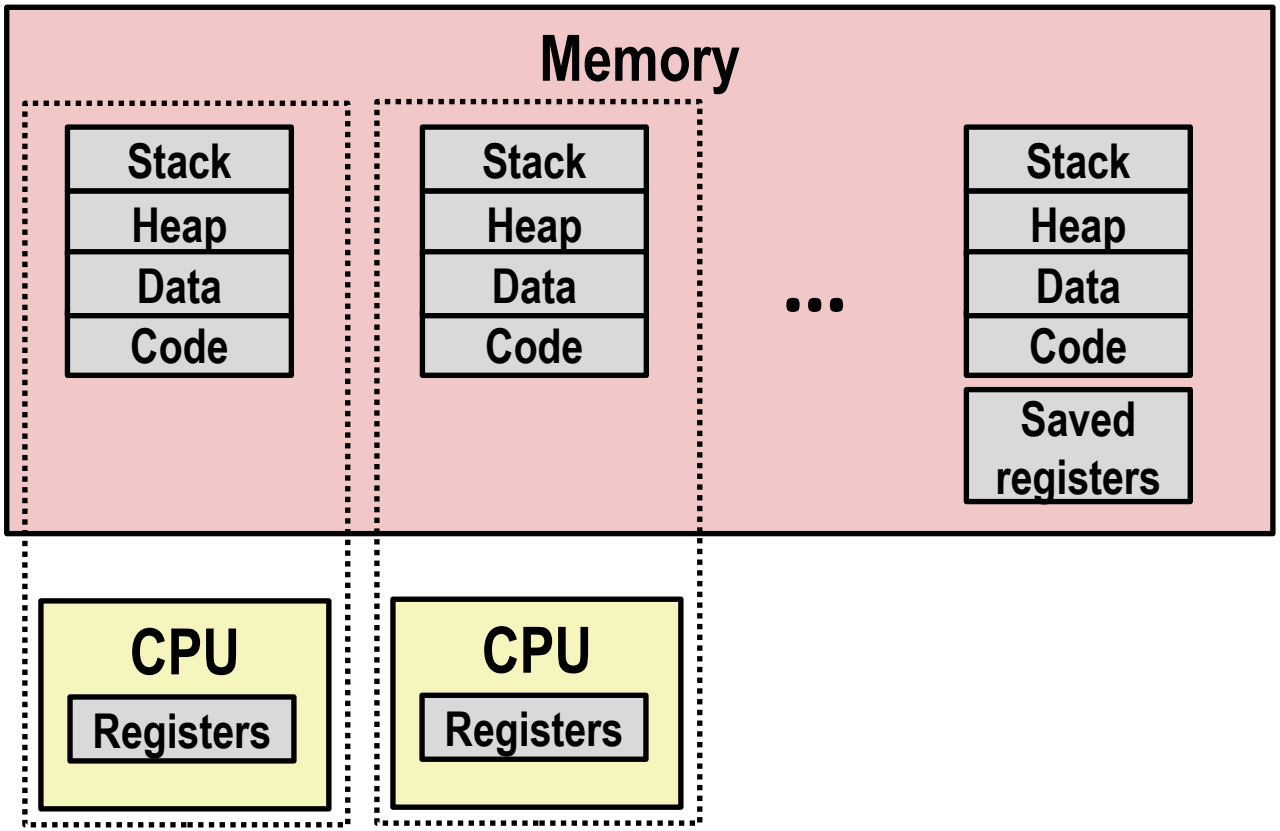
Multiprogramación: la realidad

- El SO restaura los registros guardados e intercambia el espacio de direcciones (cambio de contexto)
  - Entre los registros restaurados está el contador de programa, que es el último que se establece
    - Se establece inmediatamente antes de salir del código del SO mediante una instrucción especial, que hace que la CPU pase de modo núcleo (privilegiado) a modo usuario (no privilegiado)
  - El proceso reanudado no tiene constancia de que había sido interrumpido previamente



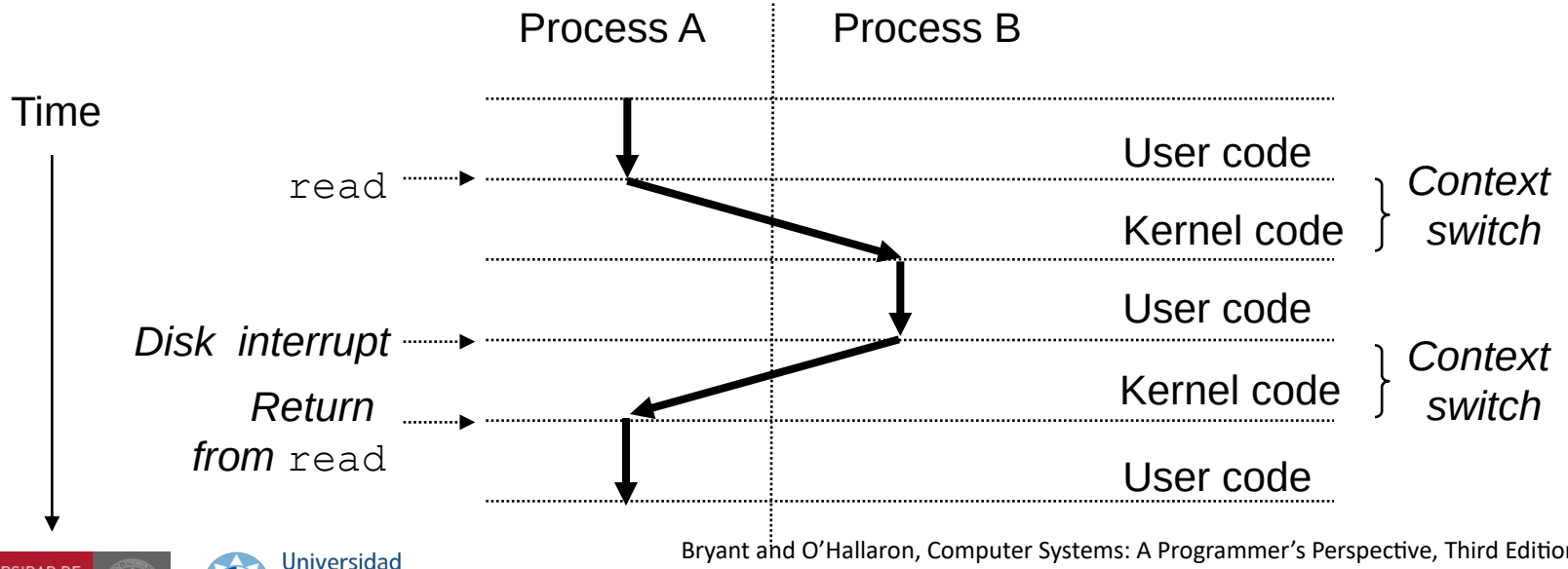
Multiprogramación: la realidad (en la actualidad)

- Procesadores multinúcleo (*multicore*)
  - Varias CPUs integradas en el mismo chip
  - Todas comparten la memoria principal (y algunas cachés)
  - Cada núcleo de procesamiento (core) puede ejecutar un proceso separado
    - El kernel del SO se encarga de la planificación de procesos para asignarlos a cada uno de los cores del sistema



Cambio de contexto entre procesos

- El flujo de control pasa de un proceso a otro mediante un cambio de contexto
  - Ejemplo:
    - La CPU está ejecutando instrucciones del proceso A
    - A solicita una operación de lectura de fichero
    - Puesto que leer del fichero tardará un tiempo, el SO cambia el contexto
      - El SO guarda todo el estado del proceso A (su **contexto**) para poder reanudarlo posteriormente
    - El SO restaura el contexto (previamente guardado) del proceso B
    - La CPU pasa a ejecutar instrucciones del proceso B, en el punto donde se quedó
    - El disco envía una señal de interrupción indicando que los datos del fichero están listos
    - El SO transfiere los datos del fichero a la memoria del proceso A y restaura su contexto



Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Estado de Proceso

- Los procesos cambian de estado conforme se ejecutan
- Estado de un proceso:
  - Definido en parte por la actividad actual de dicho proceso
- Estados:
  - Nuevo
    - Cuando está siendo creado.
  - Listo
    - Esperando a ser asignado a una CPU.
  - En ejecución
    - Está ejecutando instrucciones en la CPU
  - Esperando
    - Esperando a que ocurra algún evento, como p.ej., la terminación de una operación de E/S o la recepción de una señal
  - Terminado
    - Ha terminado su ejecución.

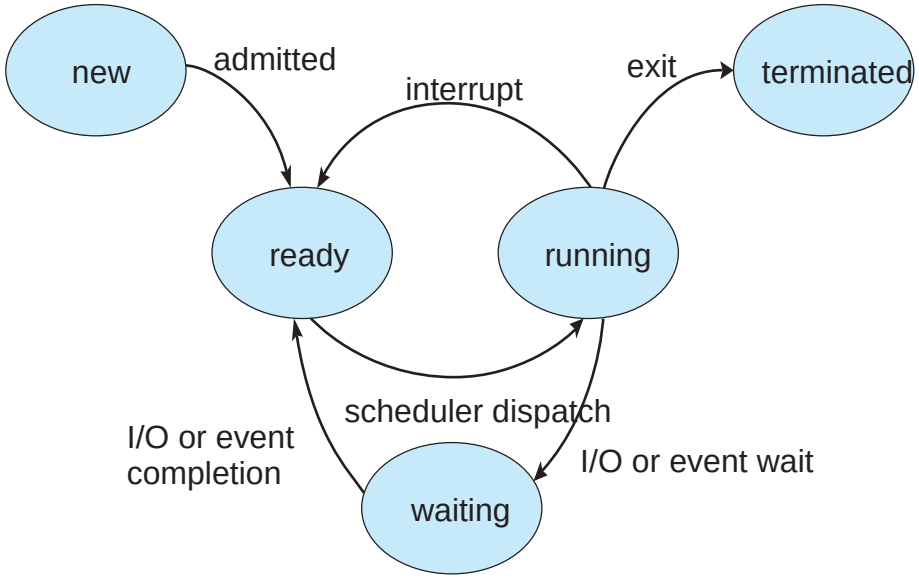
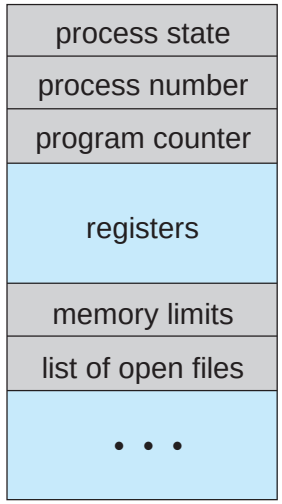


Diagram of process state

Silberschatz, Galvin and Gagne. Operating System Concepts. 9th Edition

Bloque de control de un proceso (PCB)

- Cada proceso se representa en el SO mediante un **bloque de control de proceso** (PCB)
- PCB. Contiene mucha información relativa al proceso:
  - Identificador de proceso (**pid**).
    - Generalmente un número entero, único en el sistema, usado como índice para acceder a los diferentes atributos de un proceso dentro del núcleo
  - Estado:
    - Nuevo, listo, en ejecución, esperando, terminado
  - Contador de programa
    - Indica la dirección de la siguiente instrucción a ser ejecutada por este proceso
  - Registros de la CPU
    - Varían en número y tipo, dependiendo de la arquitectura del procesador
  - Información sobre planificación
    - Incluye la prioridad del proceso, punteros a las colas de planificación, etc.
  - Información sobre gestión de la memoria
    - Incluye la tabla de páginas (estructura de datos clave en memoria virtual)
  - Información sobre E/S
    - Lista de dispositivos asignados al proceso, lista de ficheros abiertos, etc.
  - Información sobre contabilidad. Tiempo de uso de CPU, tiempo real, etc.



Process control block (PCB).

Silberschatz, Galvin and Gagne. Operating System Concepts. 9th Edition

Cambio de contexto entre procesos

- El SO permite la multiprogramación:
  - Uso compartido de la(s) CPU(s) entre distintos procesos
  - Vía para permitir múltiples usuarios
- Planificador (*scheduler*): es parte del núcleo del SO, responsable de optimizar el uso de la(s) CPU(s)
  - Los procesos bloqueados en una E/S (p.ej., lectura de disco) "ceden" automáticamente la CPU a otro proceso
  - El planificador siempre intenta optimizar el uso de este recurso.
- Todos los procesos van avanzando con sensación de simultaneidad
  - Puede no haber paralelismo real
    - Si hay sólo una CPU

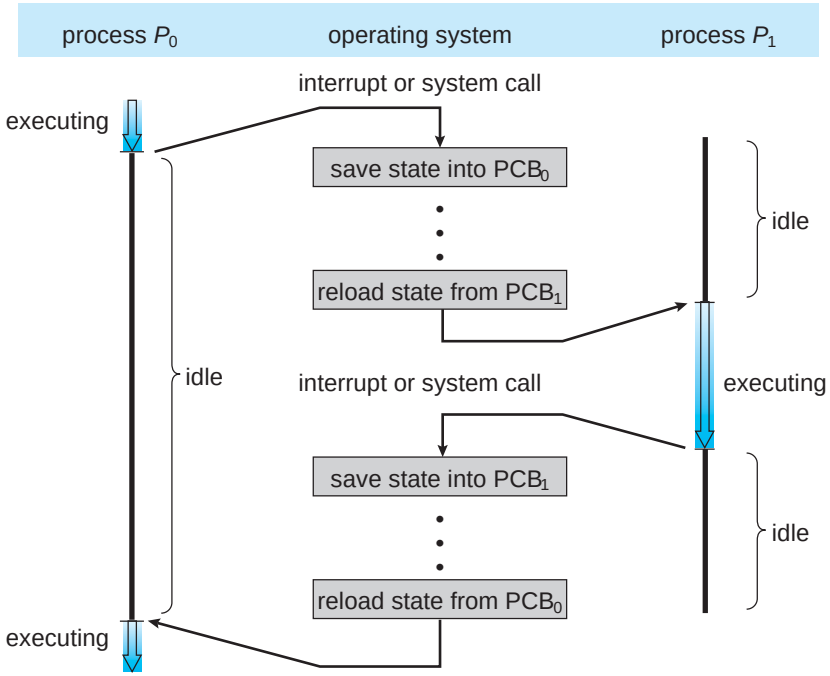
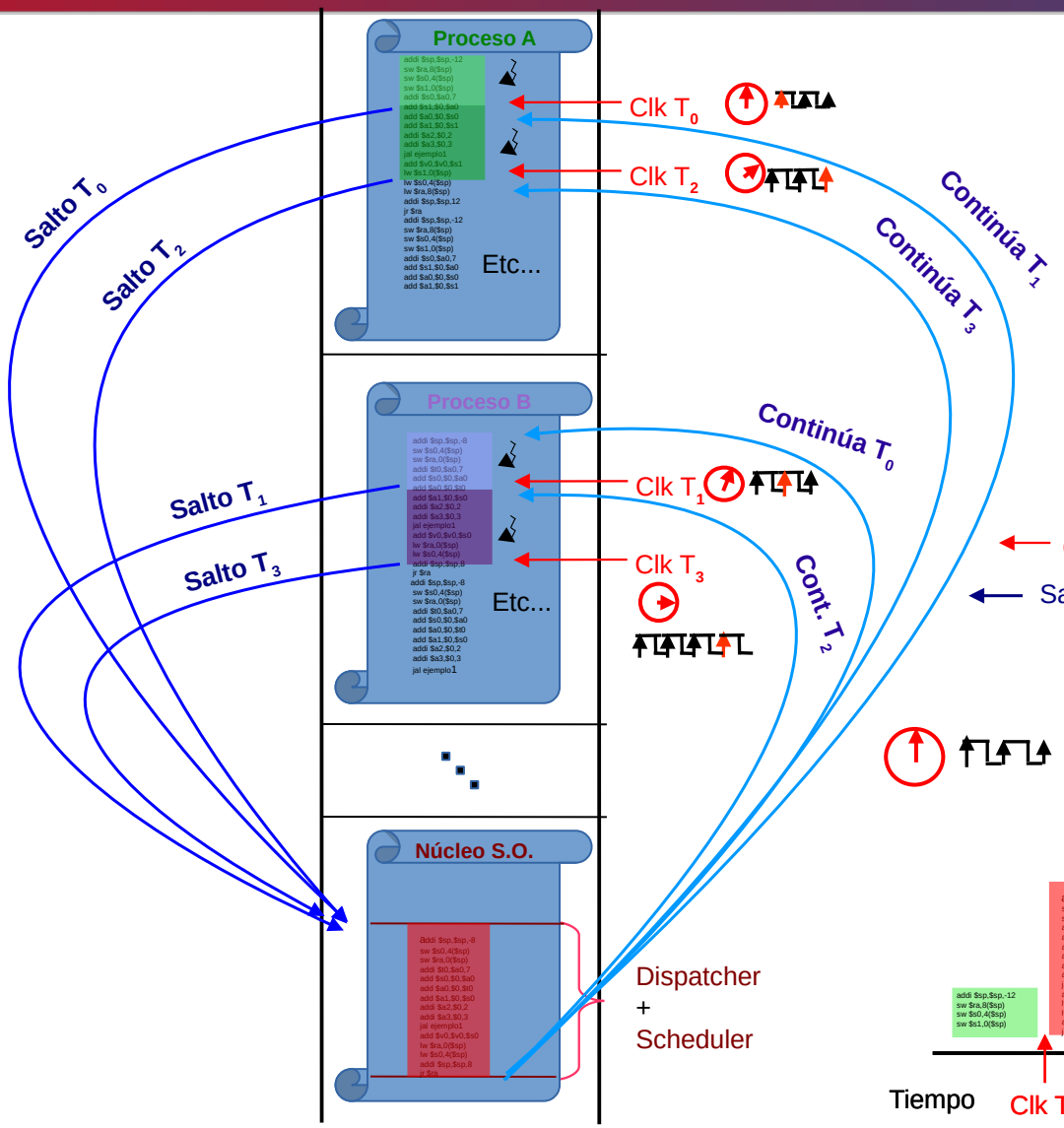


Diagram showing CPU switch from process to process

Silberschatz, Galvin and Gagne. Operating System Concepts. 9th Edition

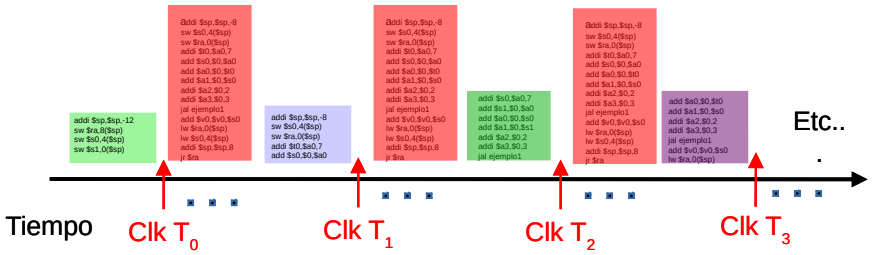
Gestión de procesos: Multiprogramación



Clk T_i = Interrupción del Reloj

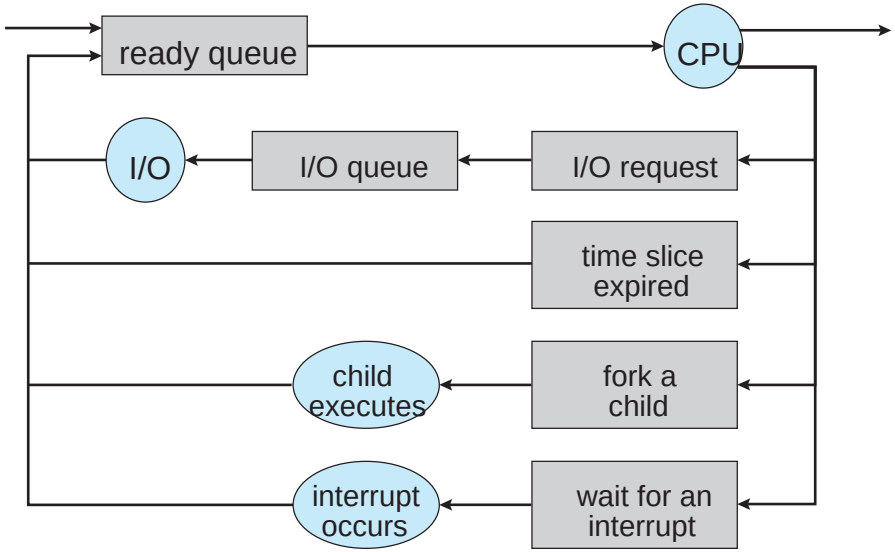
Salto T_i = Salto a la R.S.I. que provoca la Clk T_i

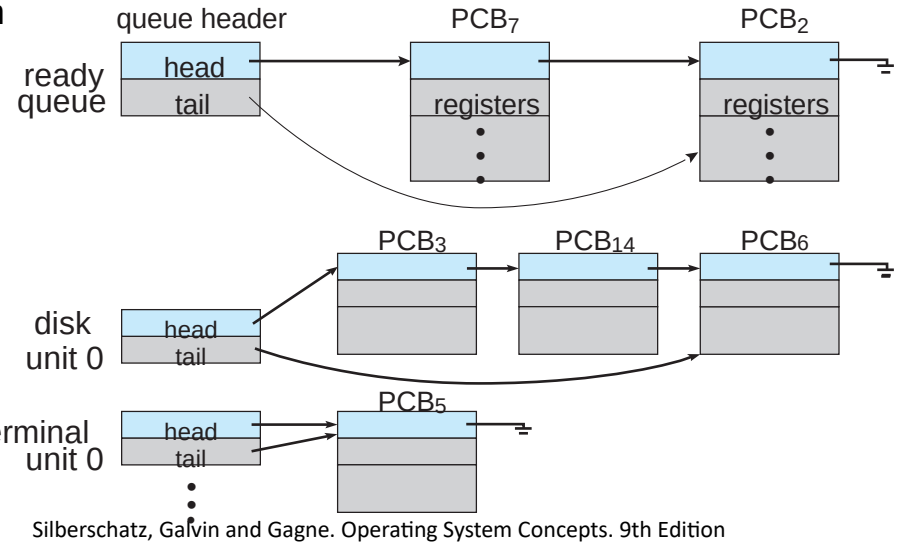
A la CPU le llega un pulso de reloj cada *x* milisegundos, cediéndose el control de ejecución al SO, que guarda datos del proceso actual para poder devolvérselos posteriormente.



Planificación de procesos (*scheduling*)

- Objetivo de la multiprogramación:
  - Tener algún proceso en ejecución en todo momento → maximizar la utilización de la CPU .
  - Permitir a los usuarios interactuar con los procesos mientras se ejecutan → cambiar la CPU con una frecuencia muy elevada
- Para cumplir estos objetivos, el **planificador de procesos** selecciona un proceso disponible para su ejecución
- **Cola de procesos listos** (*ready queue*): Procesos en memoria principal y listos para ejecutarse
- **Cola de dispositivo** (*device queue*): Procesos bloqueados en espera de un dispositivo de E/S particular
  - Una cola por cada dispositivo de E/S
- Los procesos se colocan en la *ready queue* al crearse, en espera de ser despachados (se les asigna la CPU)
- Una vez el proceso está ejecutándose en la CPU:
  - a) Si el proceso emite una petición de E/S → se le pone en la cola de E/S del dispositivo correspondiente
  - b) Si el proceso crea un nuevo proceso hijo y espera a que este termine → se le pone en la cola de procesos bloqueados
  - c) Si el proceso es sacado a la fuerza de la CPU, como resultado de una interrupción → se le vuelve a poner en la *ready queue*
  - En los casos a) y b): eventualmente el proceso cambiará de estado (de "esperando" a "listo") → se le vuelve a poner en la cola de procesos listos

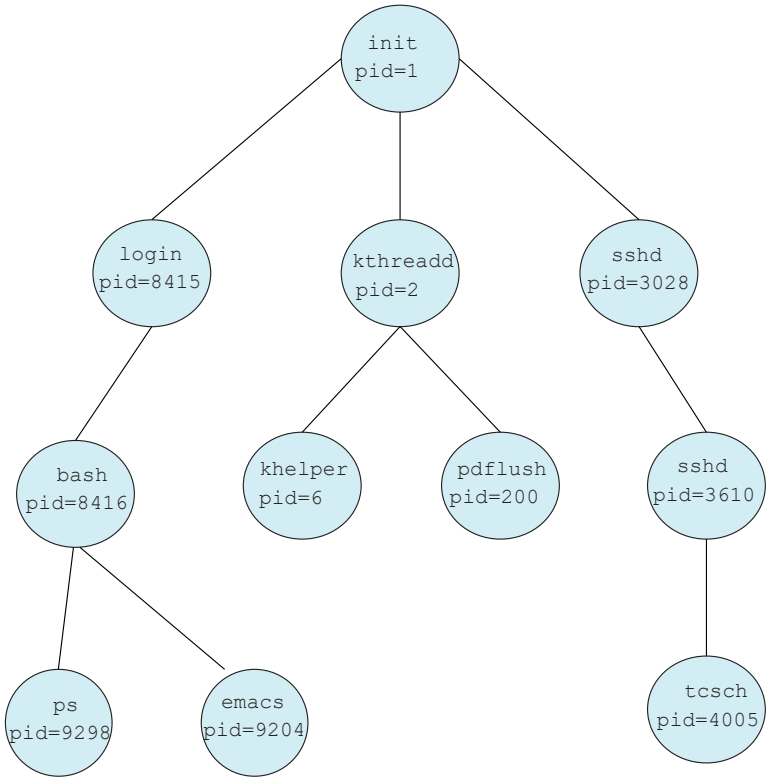


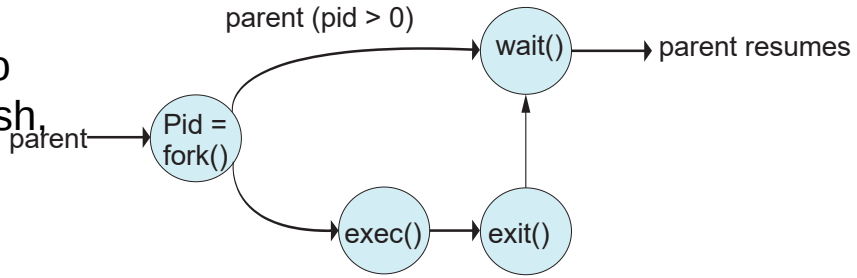


Silberschatz, Galvin and Gagne. Operating System Concepts. 9th Edition

Creación de procesos. Árbol de procesos

- Como parte de su ejecución, un proceso puede crear nuevos procesos (llamada al sistema `fork()`)
  - El proceso creador se llama proceso padre, y los nuevos procesos son hijos
  - Inicialmente el hijo es un clon del proceso padre (idéntico)
- El proceso hijo puede optar por ejecutar un programa diferente al del padre (llamada al sistema `exec()`)
  - El proceso padre puede optar por esperar a que termine el proceso hijo, o bien continuar su ejecución (en "paralelo")
- Cada uno de estos procesos hijos, a su vez, pueden crear otros procesos, formando un árbol
- El proceso `init` sirve como proceso raíz para todos los procesos de usuario
  - Siempre tiene pid 1
- Una vez que el sistema ha arrancado, el proceso init puede crear otros procesos: login, servidor ssh, etc.
  - Ejemplo: usuario conectado a un terminal (vía login) que ha lanzado desde el shell (bash) el editor emacs y el programa ps





Silberschatz, Galvin and Gagne. Operating System Concepts. 9th Edition

Índice

Gestión de la memoria

- Cada proceso comparte la CPU y la memoria principal con otros procesos
- Gestión de la CPU: conforme sube su demanda, los procesos se ralentizan paulatinamente
  - El SO se encarga de que los procesos usen la CPU por turnos (*time-sharing*)
    - Turnos más cortos o menos frecuentes → ejecución se hace poco a poco más lenta
- Gestión de la memoria: recurso crítico, si unos procesos demandan más memoria de la disponible, pueden hacer que otros no puedan ejecutarse
  - El SO hace lo posible por que cada proceso tenga toda la memoria que necesite
    - Al tiempo que trata de evitar que unos procesos impidan la ejecución de otros
  - La memoria es vulnerable a la corrupción, evitando que un proceso escriba en la memoria utilizada por otro proceso, modificando su lógica de ejecución
- **Memoria Virtual** (VM): Abstracción ofrecida por el SO sobre la memoria principal para dar a cada proceso la ilusión de que tiene **toda la memoria** para sí mismo

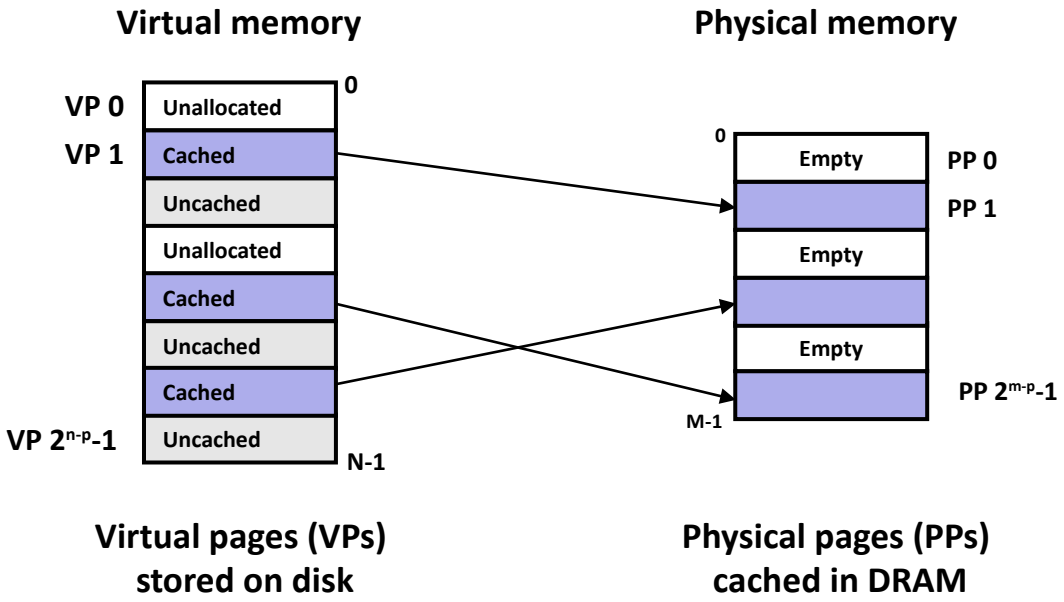
Memoria virtual: Capacidades

- **Facilita la utilización eficiente de la memoria principal**
  - Trata la memoria principal (RAM) como una **caché** para un espacio de direcciones almacenado en el disco
  - Mantiene en memoria principal sólo las áreas activas del espacio de direcciones virtual de cada proceso
  - Transfiere datos entre disco y memoria principal a demanda
- **Simplifica la gestión de la memoria**
  - Proporciona a cada proceso un espacio de direcciones uniforme
  - Idéntico para todos los procesos
  - Facilita la compartición de código/datos entre procesos
- **Protege el espacio de direcciones de cada proceso**
  - Aísla la memoria de cada proceso, impide acceso por parte de otros procesos
- MV: solución elegante → cooperación hardware/software
  - Hardware: MMU: *Memory Management Unit* (parte de la CPU para MV)
  - Software: El SO gestiona la MMU, mantiene estructuras de datos para MV…

Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Caching mediante MV

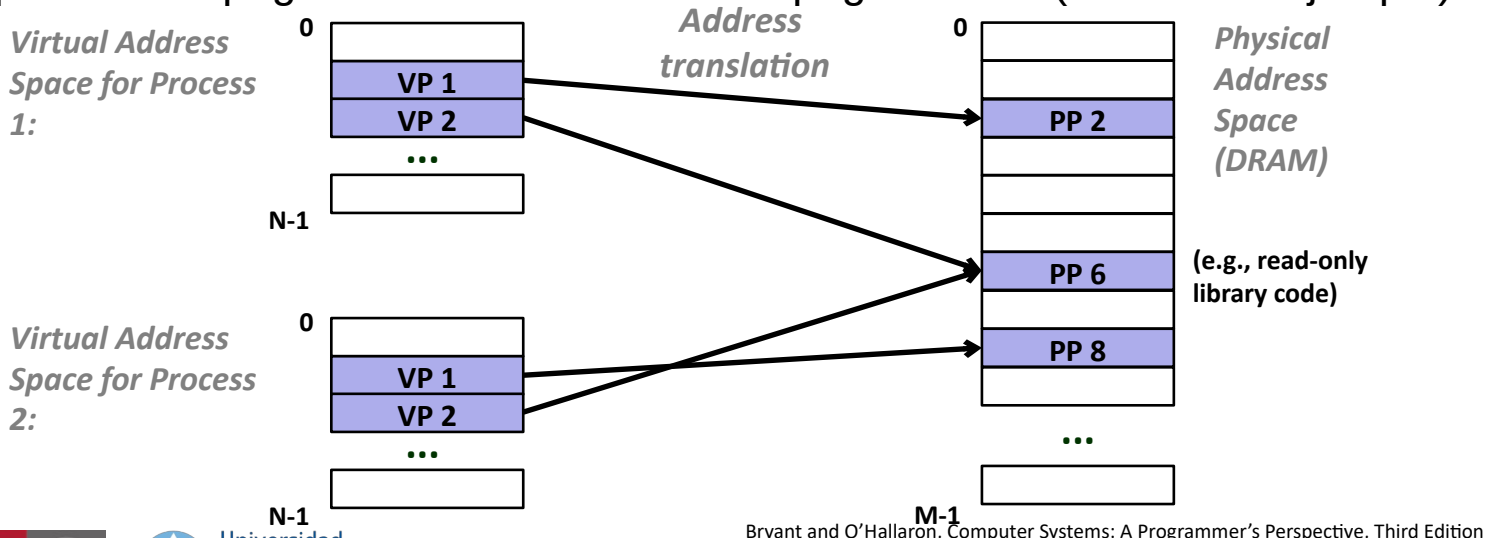
- Conceptualmente: la memoria virtual se puede ver como un array de bytes contiguos almacenados en disco
- El contenido del array en disco "se cachea" en memoria física
  - La memoria principal (DRAM) actúa como caché del disco
    - Mantiene las páginas recientemente accedidas por los procesos. Principio de localidad
- Página: unidad de transferencia de datos disco ↔ memoria física



Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Gestión de memoria mediante MV

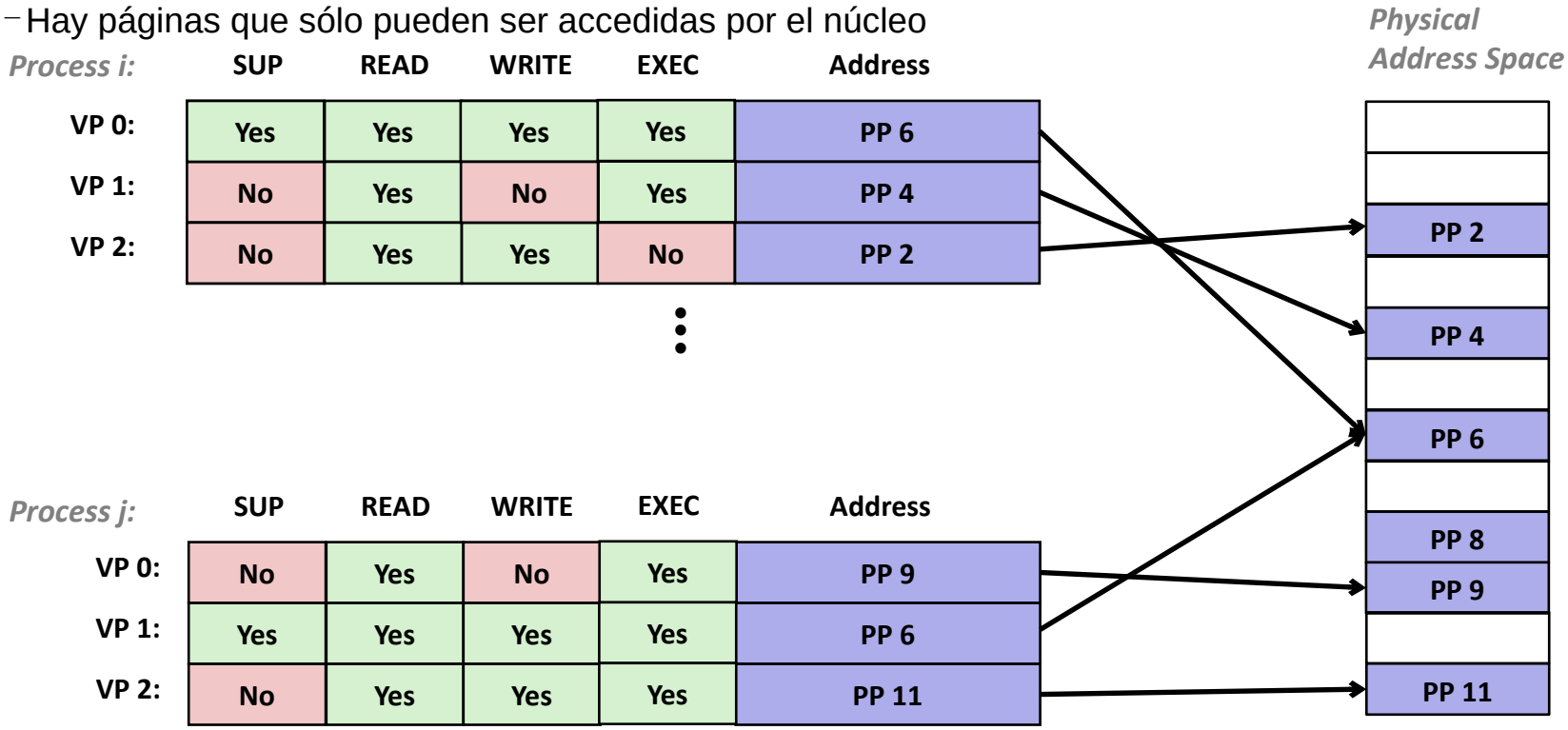
- Cada proceso tiene su espacio virtual de direcciones privado
  - Puede ver la memoria como un simple array lineal
  - Una función de mapeo distribuye las direcciones virtuales en memoria física
- Simplifica la reserva de memoria
  - Cada página virtual puede estar mapeada a cualquier página física
  - En instantes distintos una misma página virtual (VP) puede estar almacenada en diferentes páginas físicas (PP)
- Facilita la compartición de código y datos entre procesos
  - Mapeando las páginas virtuales a la misma página física (PP 6 en el ejemplo)



Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

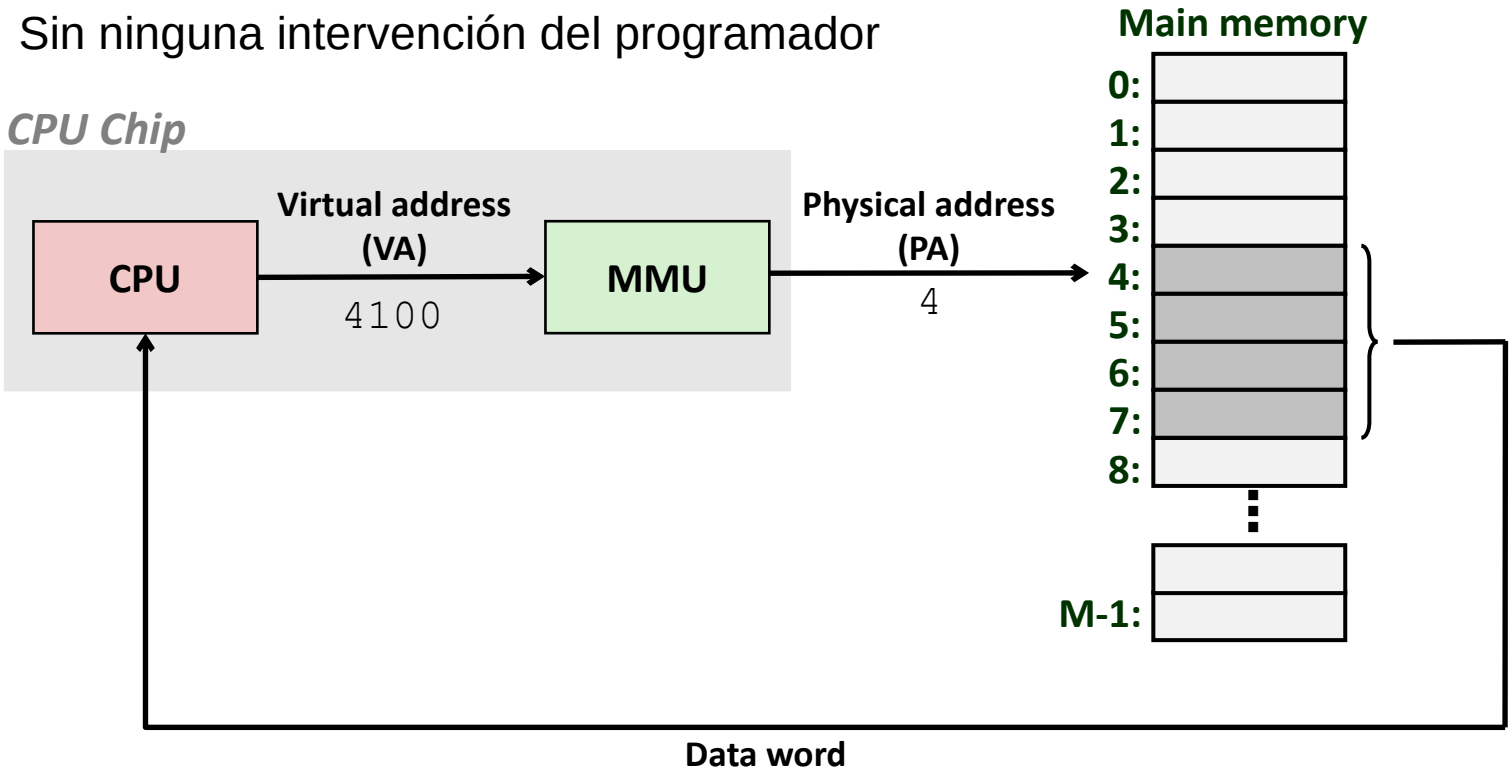
Protección de memoria mediante MV

- El sistema operativo mantiene una **tabla de páginas** (PT) que facilita la gestión del espacio de memoria de cada proceso
- Las entradas de la tabla de páginas (PTEs) contienen bits con los permisos sobre la página
  - Lectura, escritura, ejecución, modo supervisor (SUP)
- La MMU comprueba esos bits en cada acceso
  - Lanza una excepción (fallo de protección) en su caso
  - Hay páginas que sólo pueden ser accedidas por el núcleo



Direccionamiento virtual

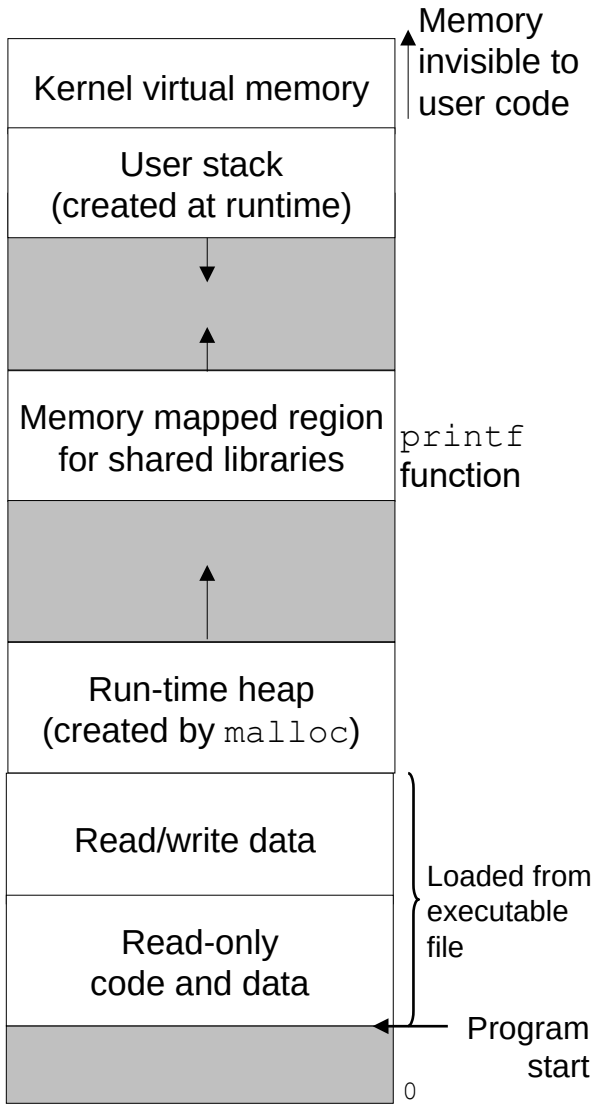
- Usado en cualquier computador moderno
  - Servidores, portátiles, smartphones, etc.
- Una de las mejores ideas en la historia de los computadores
  - Éxito: funciona de forma silenciosa y automática, transparente
    - Sin ninguna intervención del programador



Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Espacio de direcciones virtual de un proceso

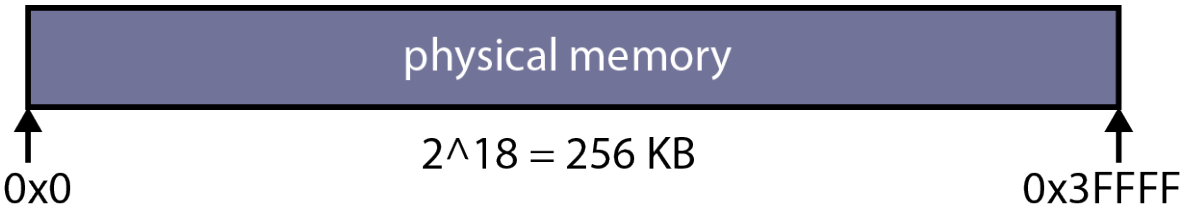
- El SO proporciona a **cada proceso** una visión **uniforme** de la memoria: su espacio de direcciones virtual
  - Lineal, desde 0 hasta 2^V-1, direcciones de V bits
  - Grande, uniforme, privado. Mucho más grande que el tamaño de la memoria principal
  - Los procesos de usuario sólo "ven" direcciones virtuales: código (PC), datos, etc.
- Espacio de direcciones virtual dividido en áreas:
  - **Código y datos** del programa. Se carga directamente del fichero ejecutable del programa en disco.
  - Memoria montón (***heap***). Espacio de memoria para reserva dinámica, en tiempo de ejecución.
  - **Librerías compartidas**. Librerías dinámicas cargadas necesarias para el funcionamiento del programa.
  - **Pila**. Espacio de memoria usado para soportar llamadas a procedimientos (pasar/devolver valores en las llamadas, variables locales, preservar registros, etc.)
  - Memoria virtual del kernel del SO: código y estructuras de datos del sistema operativo



Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

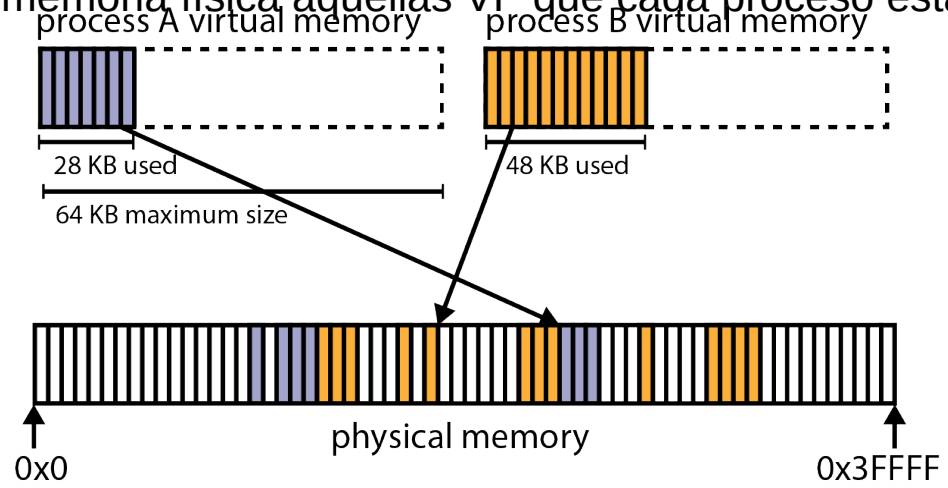
Memoria física vs Memoria virtual

- **Direcciones de memoria física** de tamaño P bits
  - Máxima cantidad de memoria principal utilizable en una máquina concreta: 2^P bytes
  - Puede variar entre CPUs, aunque sean de la misma familia (distinta microarquitectura)
    - Ejemplo: Intel(R) Core(TM) i7-8700 (P=39 bits, 512GB), AMD EPYC 7282 (P=43 bits, 8TB)
  - La cantidad de memoria física instalada en una máquina puede (y suele) ser menor que la cantidad máxima que puede ser accedida
- **Direcciones de memoria virtual** de tamaño V bits. Por lo general, V>=P
  - Determina el tamaño del espacio de direcciones virtual de cada proceso: 2^V bytes
  - Igual para todas las CPUs que implementan una misma arquitectura (ISA)
    - Ejemplo: ISA Intel x86-64: V=64 bits (16 exabytes)
- Ejemplo: con P=18, como máximo 256KB de memoria física
  - Desde 0 hasta 2^{18}-1 = 11 1111 1111 1111 1111 = 0x3FFFF



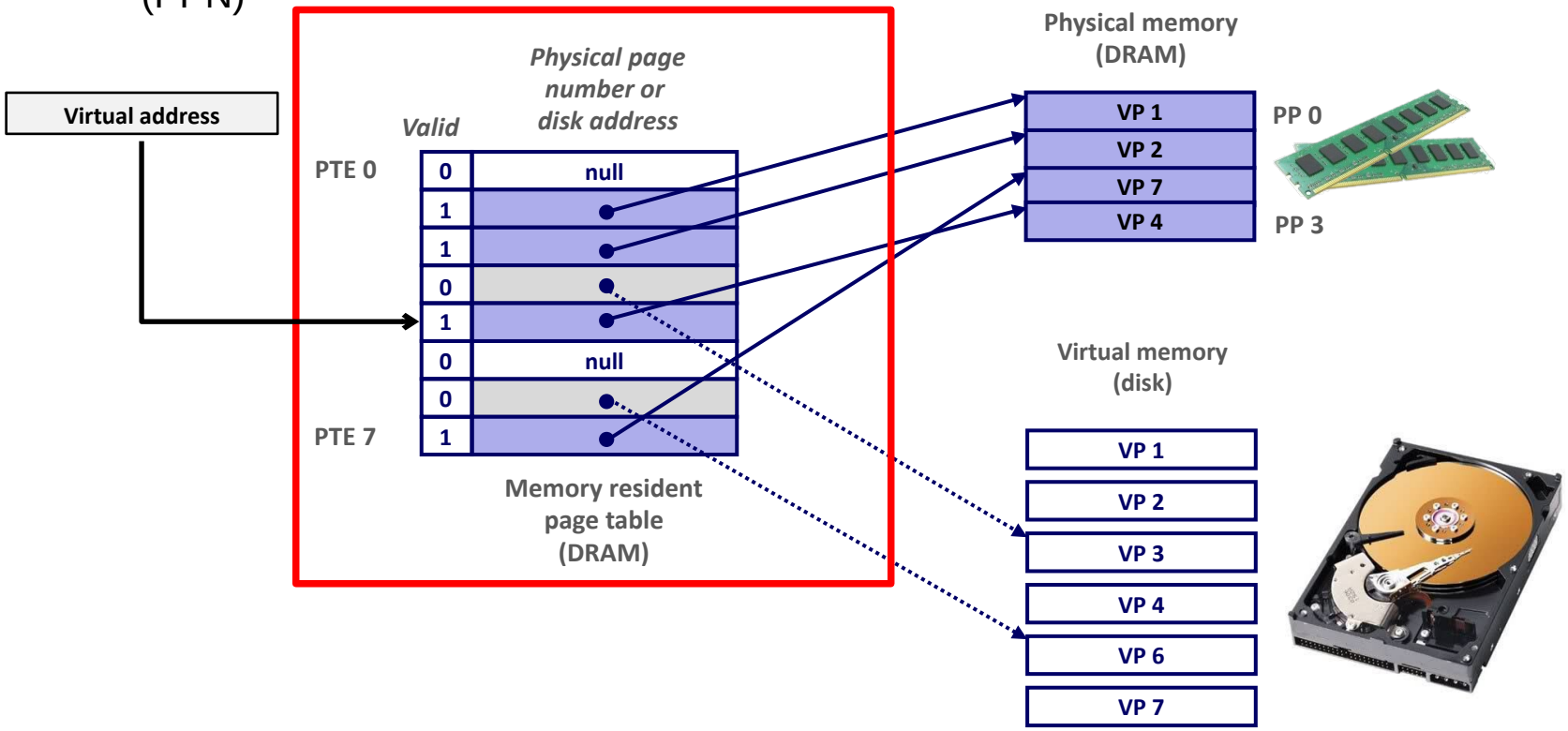
Paginación. Traducción de direcciones

- Para facilitar su gestión, el SO divide la memoria (tanto virtual como física) en trozos de igual tamaño llamados **páginas**:
  - En la memoria virtual, se llaman simplemente **páginas virtuales (VP)**
  - En la memoria física, se llaman **páginas físicas (PP)** o también **marcos** (*frames*)
- Ejemplo: si el tamaño de página es 4KB (2^{12} bytes)...
  - Con P=18, en total tendremos $2^{18} / 2^{12} = 2^6 = 64$ marcos
  - Con V=32, cada proceso tendrá hasta 4 GB de memoria virtual, distribuidas en $2^{32} / 2^{12} = 2^{20}$ páginas como mucho
- El SO mantiene la **correspondencia entre páginas virtuales y páginas físicas**.
  - No todas las VP de un proceso tienen por qué estar en memoria física a la vez
    - Sólo se cargan en memoria física aquellas VP que cada proceso está usando



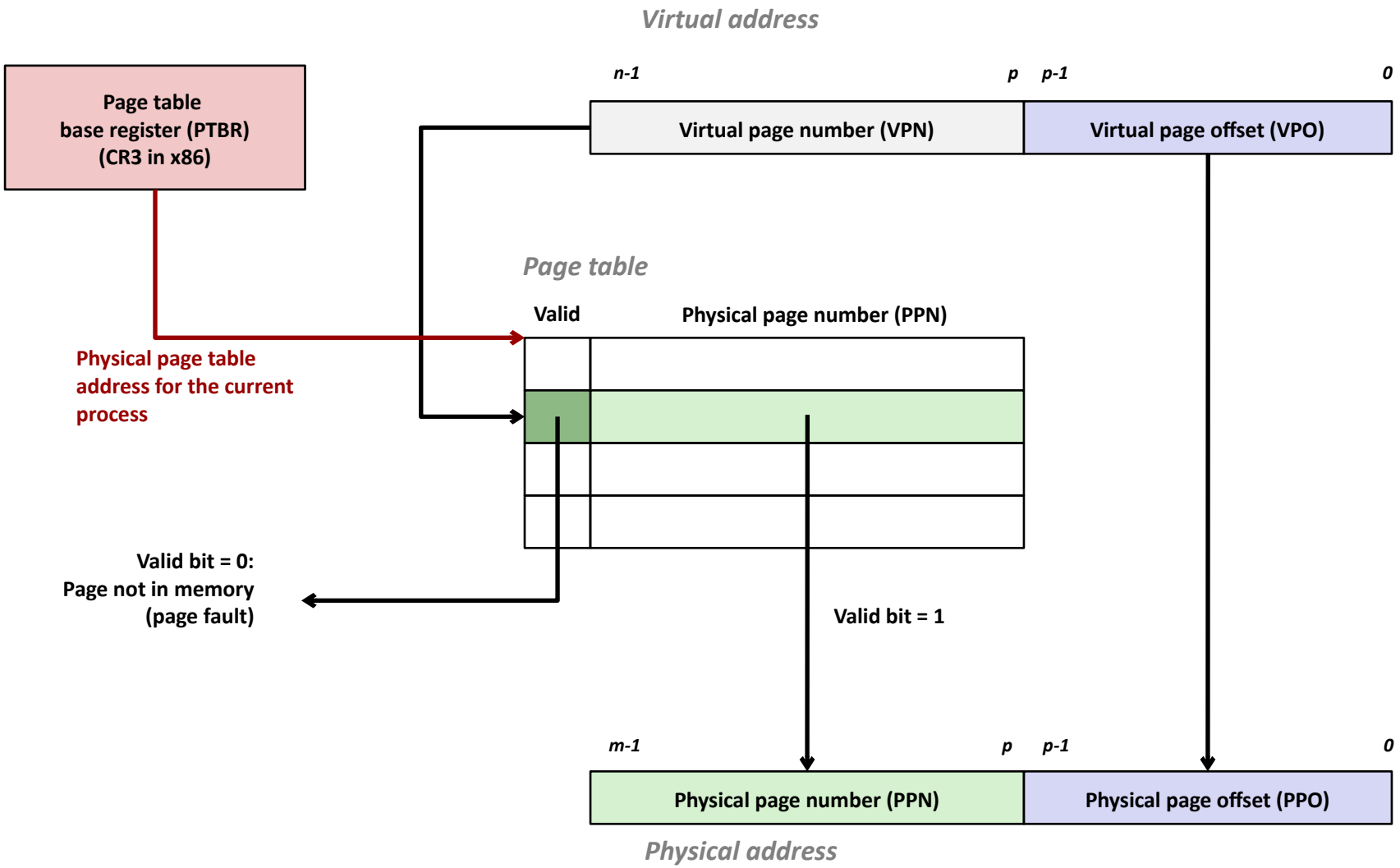
Traducción de direcciones: Tabla de páginas

- **Tabla de páginas** (PT): mantiene la correspondencia entre páginas virtuales y páginas físicas.
  - De la dirección virtual (VA) se extrae el número de página (VPN) → indexar PT
  - Cada entrada (PTE) indica si esa página virtual está en memoria física, y **dónde**
    - Si ese VPN está presente en memoria física, la PTE contiene el número de página física (PPN)



Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

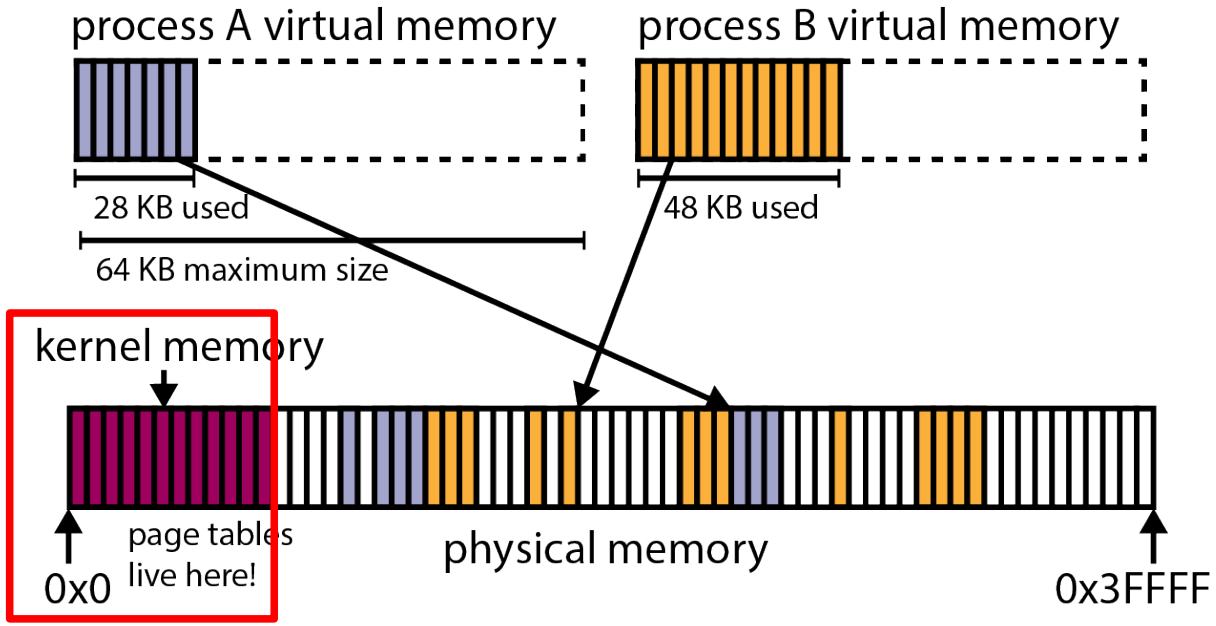
Traducción de direcciones: Tabla de páginas



Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

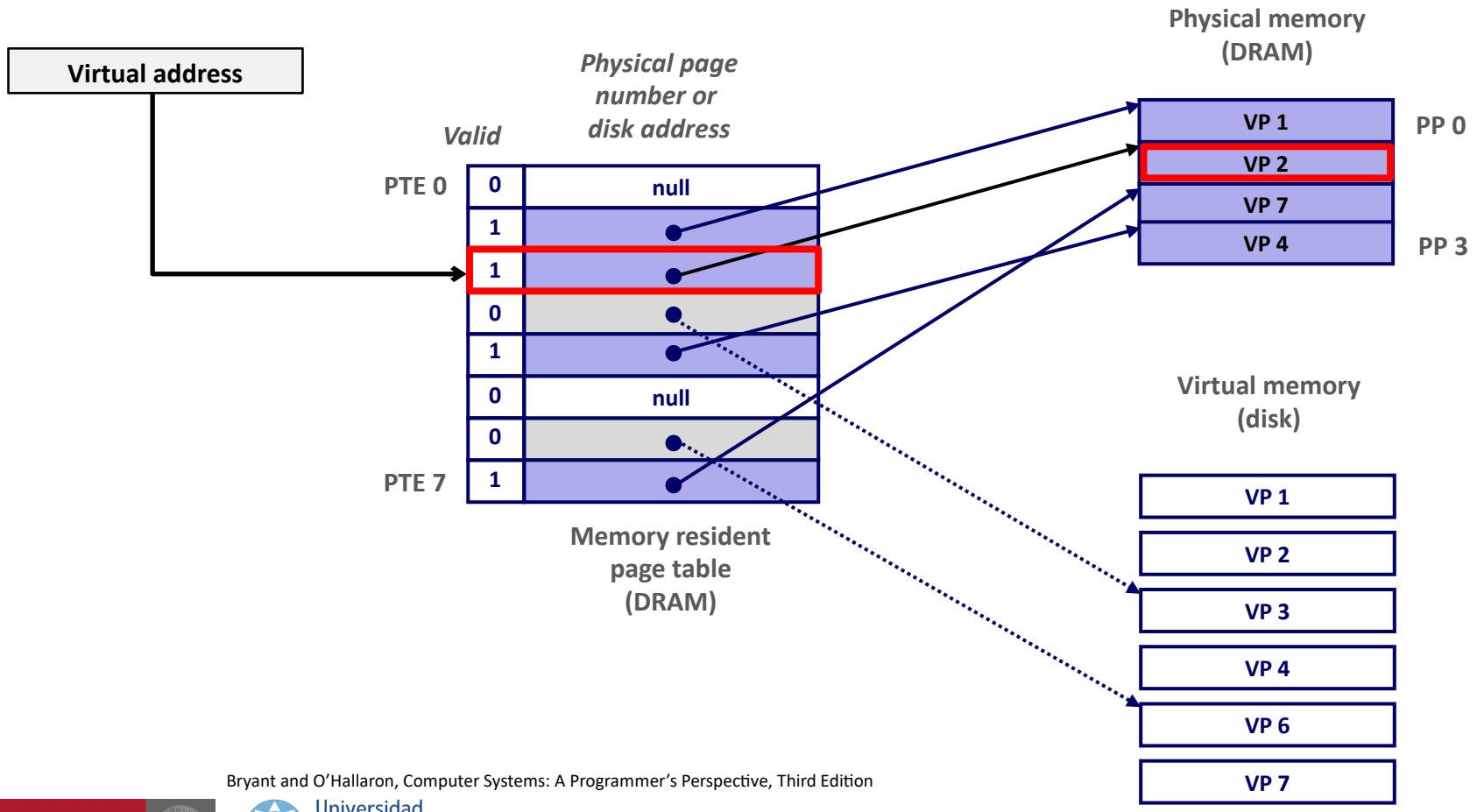
Tablas de páginas en la memoria del *núcleo*

- El SO mantiene una tabla de páginas **por cada proceso**
  - Es una de las estructuras de datos clave en la gestión de memoria
    - Se mantiene en software, dado su gran tamaño (potencialmente)
    - Soporte hardware en la CPU para cachear mapeos VPN ← → PPN (MMU & TLB)
  - Parte de ella reside en memoria física, en la memoria del núcleo del SO
    - Fuera del espacio de direcciones virtuales del proceso
      - Únicamente accesible por el núcleo



Acierto de página

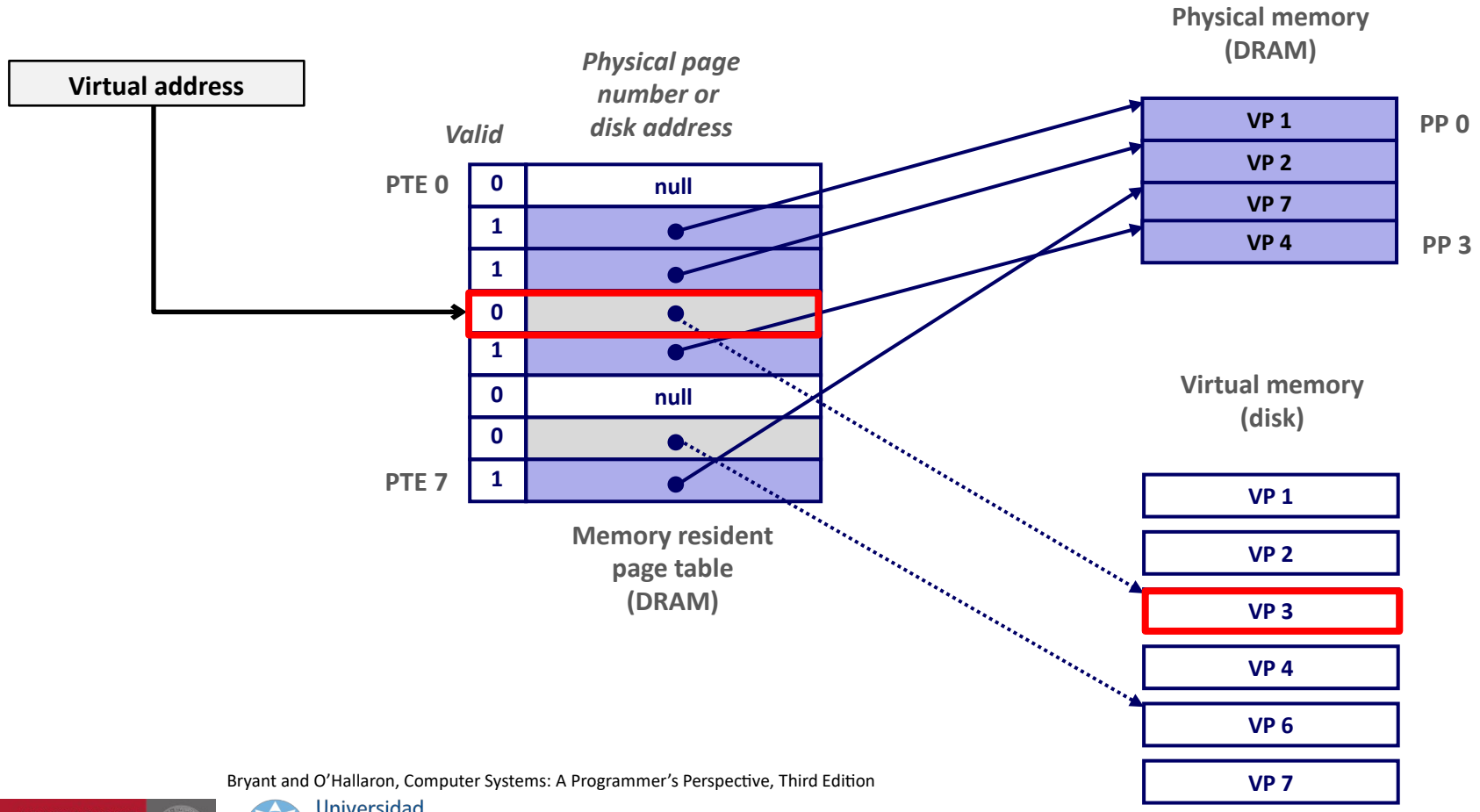
- Referencia a una palabra en MV que está en memoria física
  - Acierto en la “caché” DRAM



Bryant and O’Hallaron, Computer Systems: A Programmer’s Perspective, Third Edition

Fallo de página

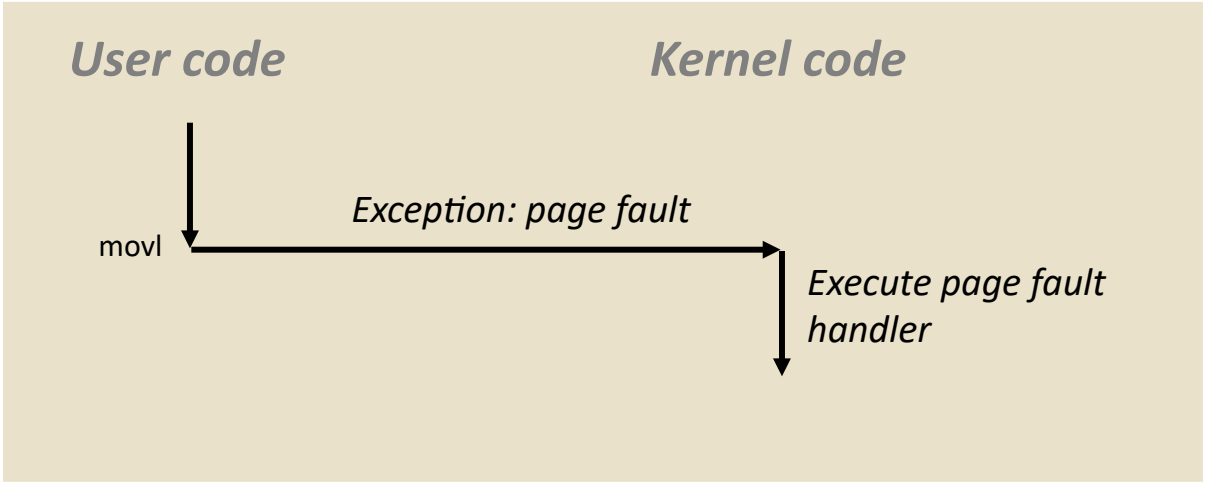
- Referencia a una palabra en VM que **NO** está en memoria física
  - "Fallo" en la caché DRAM



Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

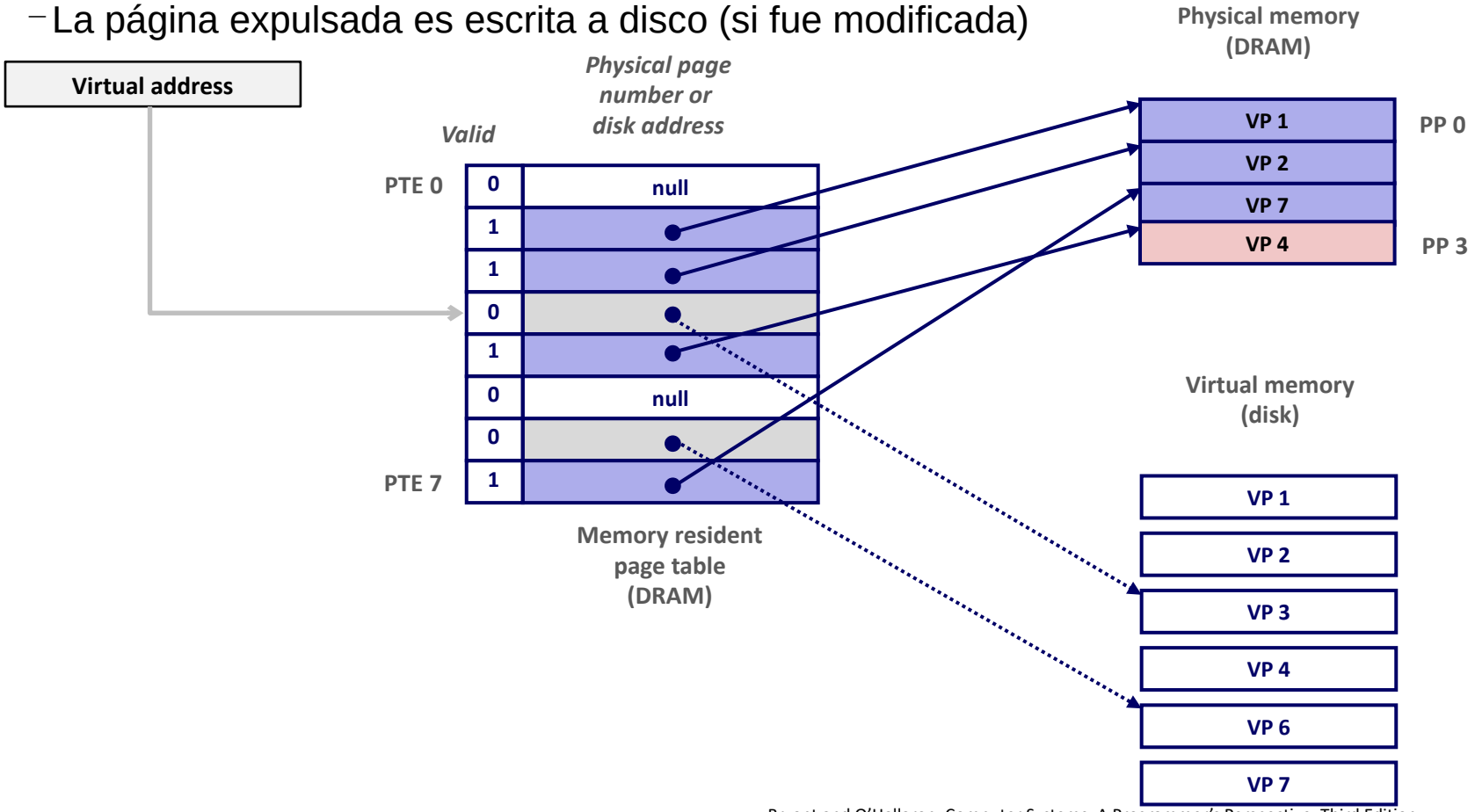
Lanzamiento de un fallo de página

- El programa de usuario escribe en una dirección de memoria
- Esa página de la memoria del usuario está en ese instante en disco
- La MMU lanza una excepción por fallo de página
  - Más detalles después
  - Eleva el nivel de privilegio de la CPU de modo usuario a modo núcleo
  - Causa una llamada a un procedimiento del SO que maneja el fallo de página
    - Rutina software del SO, siempre presente en memoria



Manejo de un fallo de página

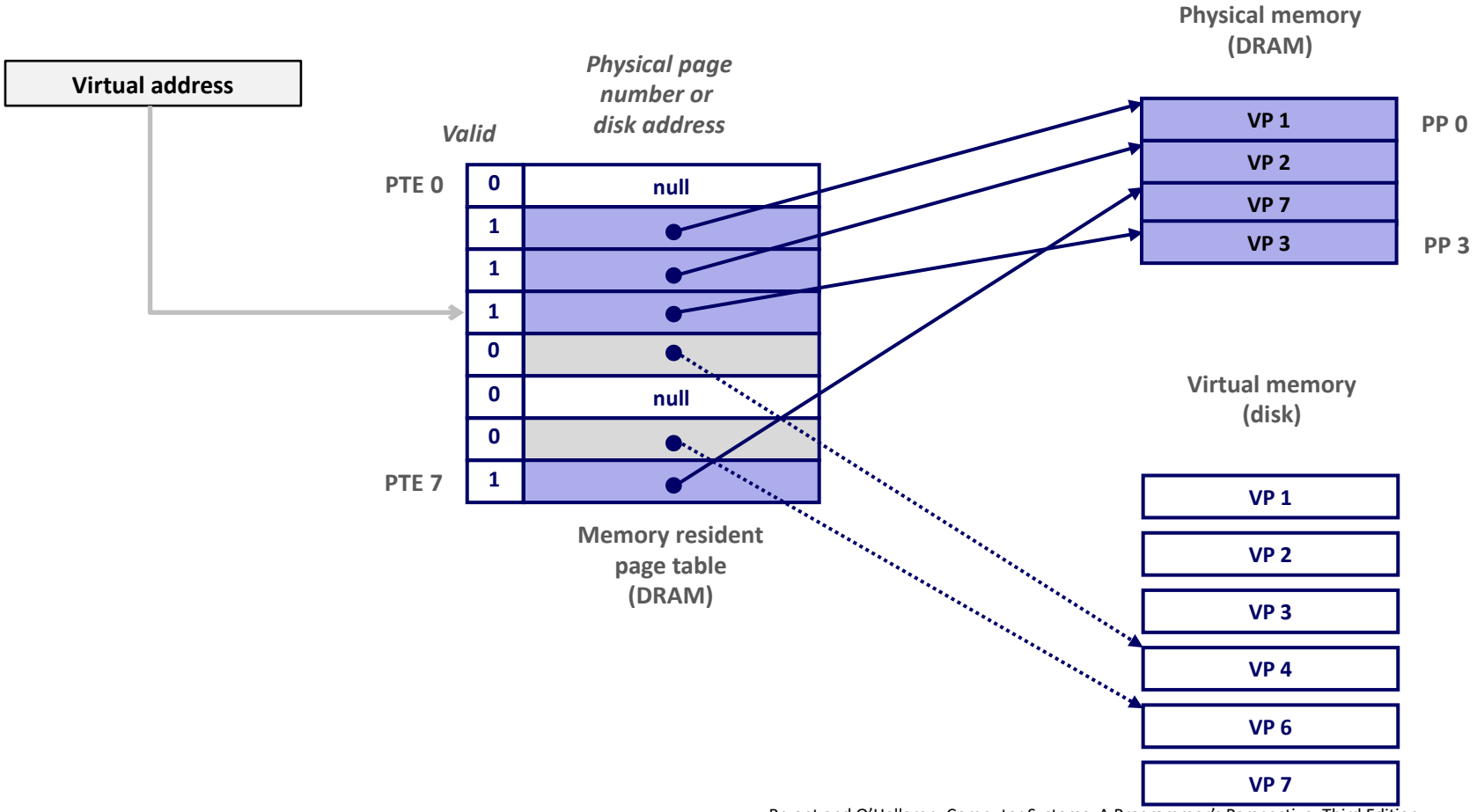
- El acceso a la tabla de páginas no encuentra PP → fallo de página
- El manejador de fallo de página (SO) elige una VP en PP como víctima
  - Será expulsada de la memoria física para hacer espacio (VP 4 en el ejemplo)
  - La página expulsada es escrita a disco (si fue modificada)



Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Manejo de un fallo de página

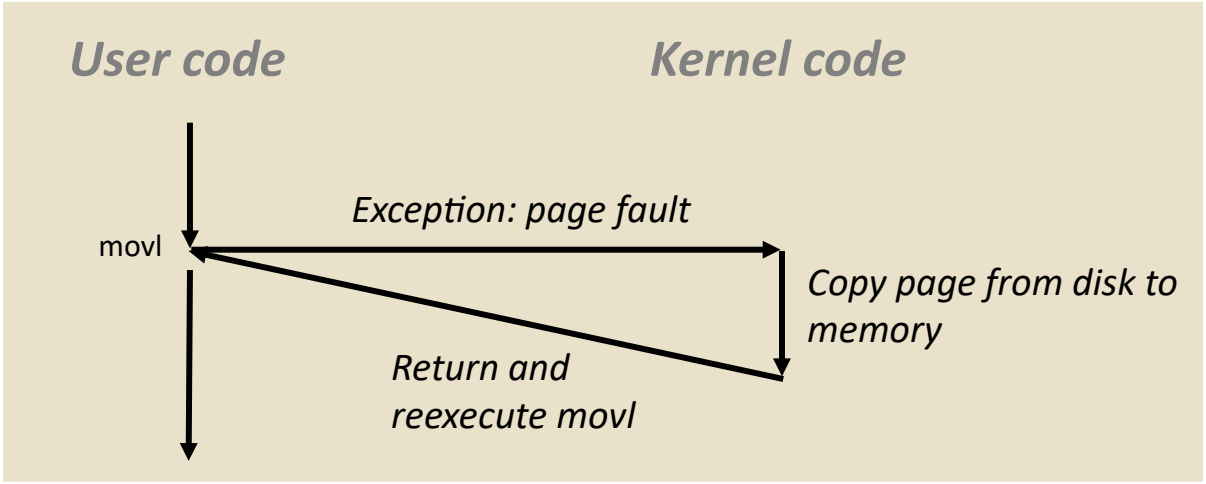
- La instrucción que causa el fallo se reinicia → Acierto de página
- Clave: esperar a que se produzca un fallo para copiar la página de disco a DRAM se conoce como **paginación bajo demanda** (*demand paging*)



Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Reanudación tras fallo de página

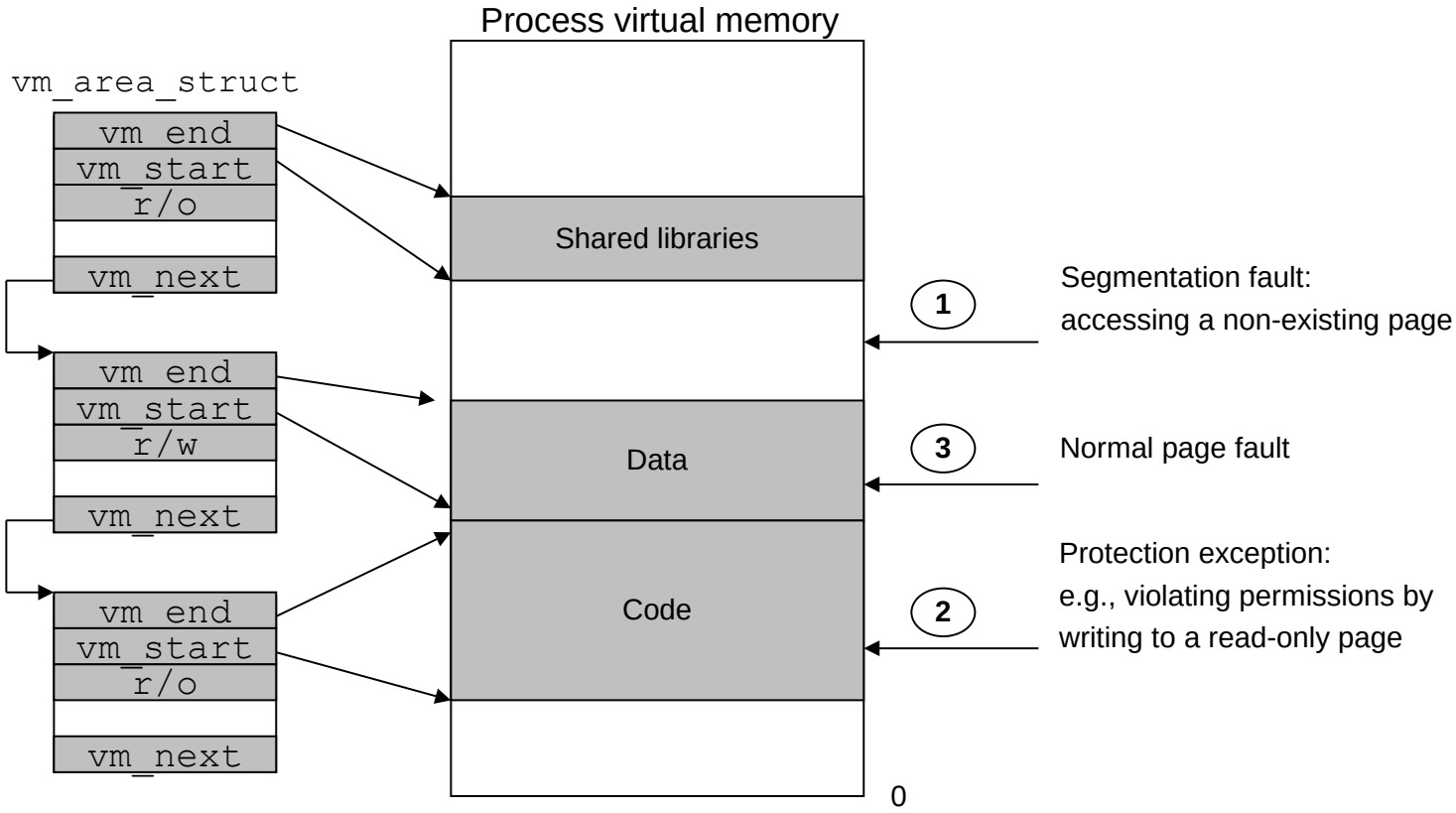
- El manejador de fallo de página ejecuta una instrucción de retorno de interrupción (`iret` en x86)
  - Similar a un retorno de un procedimiento (`ret` en x86), pero también restaura el nivel de privilegio (modo núcleo → modo usuario)
  - Regresa al código de usuario en el PC que causó el fallo de página
  - La instrucción se re-ejecuta sin causar fallo de página



Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Manejo de excepciones por fallo de página

1) Referencia a una dirección virtual ilegal
2) Excepción por protección
3) Fallo de página convencional



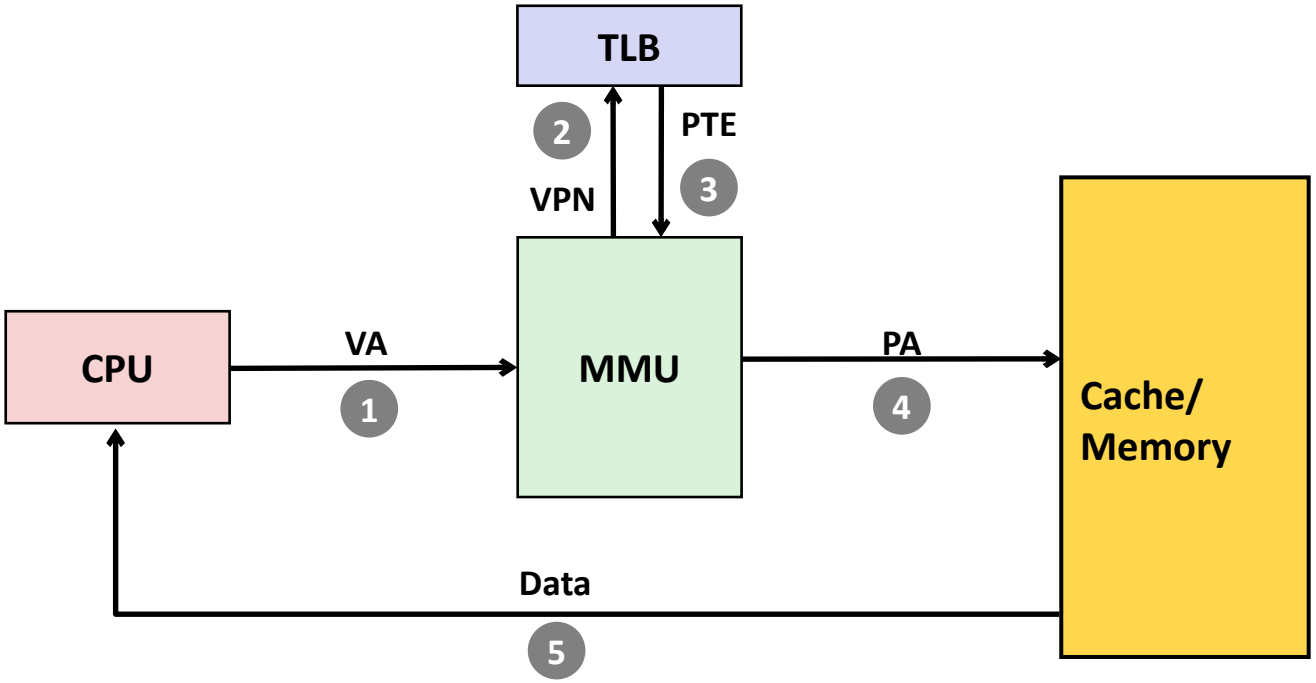
Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Memoria virtual y localidad

- La MV parece muy ineficiente, pero funciona gracias a la **localidad**
- En un instante de tiempo dado, los programas tienden a acceder a un conjunto de páginas virtuales activas → **Conjunto de trabajo** (*working set*)
  - Los programas con mejor localidad temporal tendrán conjuntos de trabajo de menor tamaño
- Si el tamaño del conjunto trabajo < tamaño memoria principal:
  - El proceso exhibirá buen rendimiento (tras la fase de calentamiento)
- Si el tamaño del working set > tamaño memoria principal:
  - ***Thrashing***: El rendimiento cae en picado cuando las páginas se intercambian (copian) entre memoria y disco continuamente: *paginación*
  - Si hay varios procesos concurrentes, el *thrashing* ocurre si la suma de sus conjuntos de trabajo excede el tamaño de la memoria principal

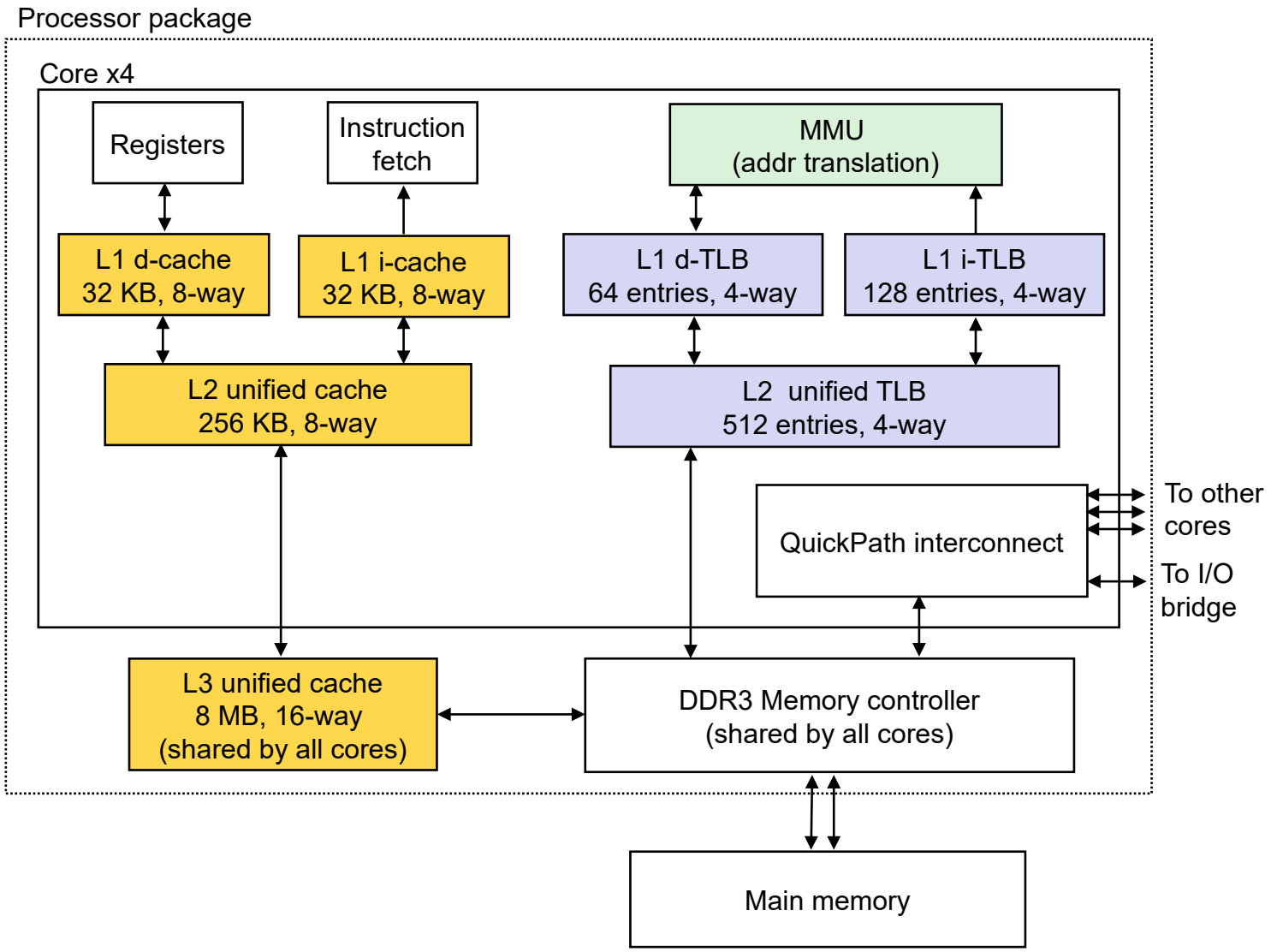
Hardware para acelerar la traducción V→P

- **TLB:** *Translation Lookaside Buffer*
  - Pequeña caché hardware asociativa que es parte de la MMU (en la CPU)
  - Mantiene correspondencias VPN←→PPN
  - Contiene entradas de la tabla de páginas (PTEs) para un pequeño número de páginas



Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Recap: Sistema de memoria Intel Core



Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Memoria virtual. Resumen

- **Visión del programador** sobre la memoria virtual:
  - Cada proceso tiene su propio espacio de direcciones lineal
  - Este espacio de direcciones está aislado de otros procesos
- **Visión del sistema** sobre la memoria virtual:
  - Uso eficiente de la memoria mediante el cacheo de páginas
    - Eficiente gracias a la localidad de referencia
  - Simplifica la gestión de la memoria y la programación
  - Simplifica la protección, al proporcionar un mecanismo de interposición ideal para comprobar permisos
- MV implementada **combinando hardware y software**:
  - Hardware en el procesador: MMU, TLB
  - SO: Maneja fallos de página, gestiona TLB, tablas de páginas, etc.

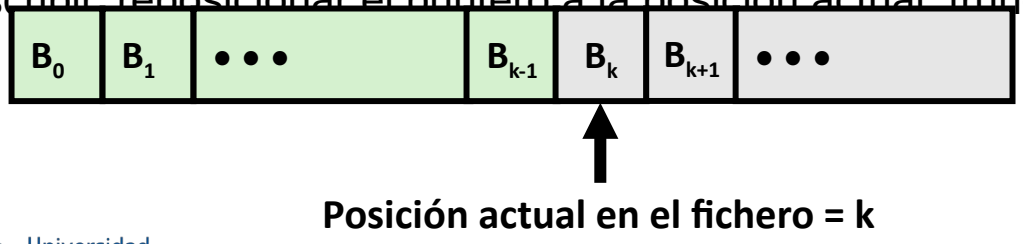
Índice

Sistemas de ficheros

- Componente del SO **encargado de gestionar los ficheros**
  - Ofrece acceso eficiente y práctico a los discos al permitir que los datos puedan ser almacenados, localizados y recuperados fácilmente
    - A los usuarios: Proporciona una visión jerárquica, uniforme, sobre los datos.
  - A los programadores: Proporciona un API mediante la que manipular ficheros
  - Interactúa con el subsistema de E/S del kernel (de más bajo nivel)
    - Es el subsistema de E/S el que realmente accede a los datos en los dispositivos de E/S
- El sistema de ficheros está compuesto por **ficheros y directorios**:
  - Colección de ficheros: almacenan los datos propiamente dichos
  - Estructura de directorios: organizan y proporcionan información sobre los ficheros
    - Los directorios son listas de ficheros y otros directorios
  - Tanto los ficheros como los directorios se identifican por su ruta
  - Una ruta es una secuencia de cadenas de caracteres separadas por la barra (‘/’ en UNIX, ‘\’ en Windows)
- El SF reside permanentemente **en el almacenamiento secundario**
  - Los datos almacenados en los ficheros pueden estar repartidos en varias áreas del disco, o incluso distribuidos en múltiples discos, o incluso en distintas máquinas.
- El SF es la parte más visible del SO para la mayoría de usuarios

Tipos de ficheros. Atributos. Operaciones

- **Fichero**: colección de información relacionada definida por su creador, identificada por un nombre, que se guarda en almacenamiento secundario
  - Para el usuario: unidad lógica mínima de almacenamiento secundario
    - No es posible guardar datos en el almacenamiento secundario a menos que estén en un fichero
- Cada fichero tiene un **tipo** que indica su rol en el sistema. Tipos en Linux:
  - Fichero regular: contiene datos arbitrarios
  - Directorio: Índice para agrupar ficheros relacionados (en Linux los directorios son un tipo de fichero)
  - Enlace simbólico: Puntero indirecto a otro fichero/directorio (~ acceso directo en Windows)
  - Otros: (siguientes transparencias)
- Cada fichero tiene una serie de **atributos** (metadatos), entre los que se incluyen:
  - Nombre, identificador, tipo, ubicación, tamaño, protección, fecha y hora...
- **Operaciones** sobre ficheros:
  - Crear, leer, escribir, reposicionar el puntero a la posición actual, truncar, borrar



Ficheros regulares. Ficheros de texto

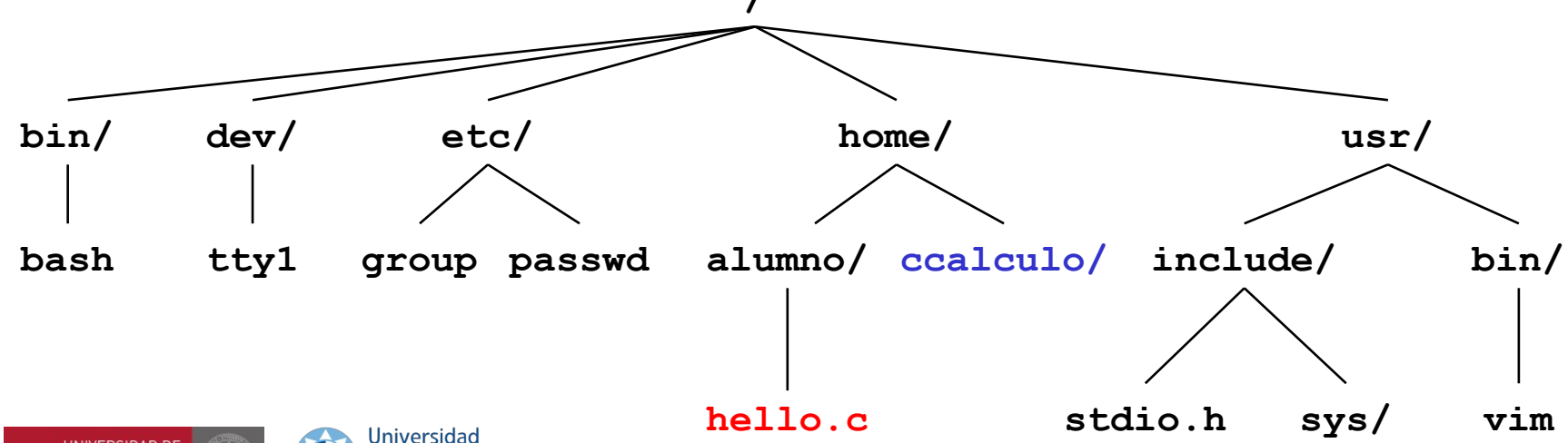
- Un fichero regular contiene datos arbitrarios, organizados por sus usuarios
  - El contenido de un fichero no es (necesariamente) relevante al SO, que no sabe cómo está organizado ni por qué cierta colección de bytes se ha organizado conjuntamente
- Las aplicaciones a menudo distinguen entre **ficheros de texto** y **ficheros binarios**
  - Los ficheros de texto son ficheros regulares que contienen únicamente caracteres
    - Caracteres ASCII o UNICODE
  - Los ficheros binarios son todos los demás
    - Ejemplo: Ficheros de código objeto, imágenes JPEG, documentos PDF, ODT, DOCX, etc.
  - El núcleo del sistema operativo **no distingue** entre ficheros de texto y binarios
- Un **fichero de texto** es una secuencia de líneas de texto
  - Una línea de texto es una secuencia de caracteres terminada por un carácter de nueva línea (‘`\n`’)
  - Indicador de fin de línea (EOL, *end of line*):
    - Linux y Mac OS: ‘\n’ (0x0A)
      - Alimentación de línea (LF)
    - Windows y protocolos de Internet : ‘\r\n’ (0x0D 0x0A)
      - Retorno de carro (CR) seguido de alimentación de línea (LF)

Fichero y dispositivos de E/S. Unix I/O

- Otros tipos de fichero en Unix/Linux:
  - **Sockets:** comunicación entre procesos en hosts distintos
  - **Tuberías con nombre** (FIFOs): comunicación entre procesos en el mismo host
  - **Dispositivos de bloques/caracteres**
    - /dev/sda2 → /usr, partición de disco (*block device*)
    - /dev/tty2 → terminal (*character device*)
- Todos los **dispositivos de E/S se representan como ficheros**
  - Gracias a la elegante correspondencia de ficheros a dispositivos E/S → SO ofrece una interfaz muy simple: **Unix I/O**
    - Abrir, leer, escribir y cerrar ficheros
      - open(), read(), write(), close()
    - Cambiar posición actual en el fichero: Indica siguiente desplazamiento (*offset*) a leer/escribir
      - lseek()

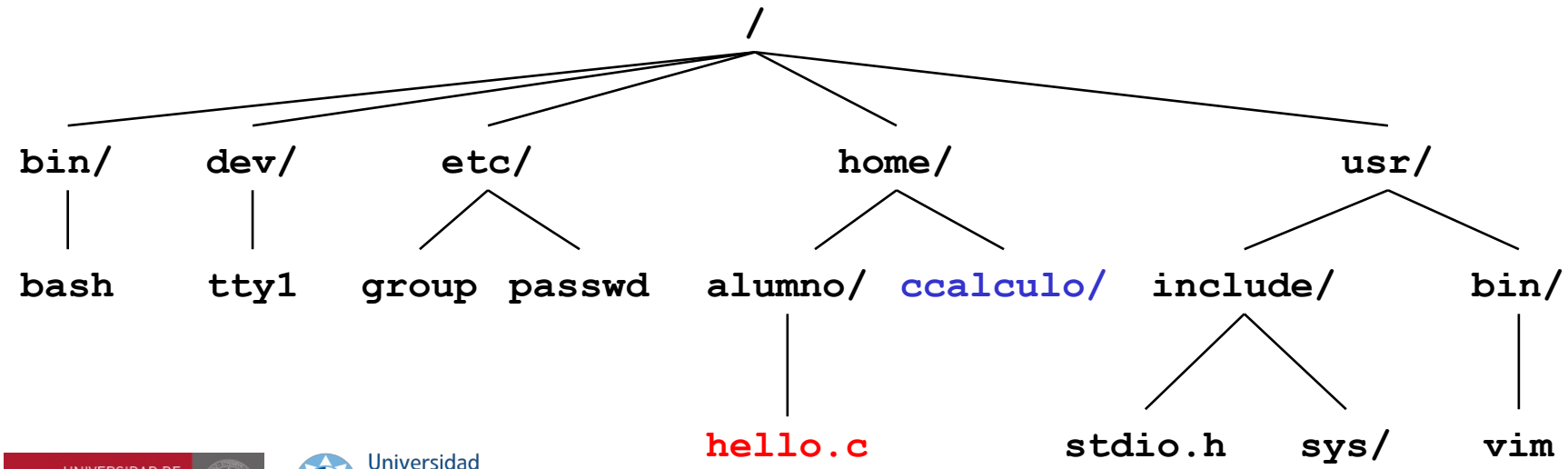
Directorios. Jerarquía de directorios

- Un directorio consiste en una **lista de enlaces**
  - Cada enlace mapea un nombre de fichero a un fichero (su contenido)
    - Es posible tener diferentes enlaces (nombres de ficheros distintos) que apunten al mismo contenido
  - El tamaño del directorio viene dado por el número de entradas (longitud de la lista de enlaces)
    - No por el contenido de los ficheros a los que apunta
- Cada directorio contiene al menos dos entradas de directorio
  - `.` (punto): es un enlace a sí mismo
  - `..` (punto punto): es un enlace al directorio padre en la jerarquía de directorios
- Todos los ficheros se organizan como una jerarquía anclada por el **directorio raíz**, llamado / (barra)
  - El kernel mantiene para cada proceso su directorio de trabajo actual



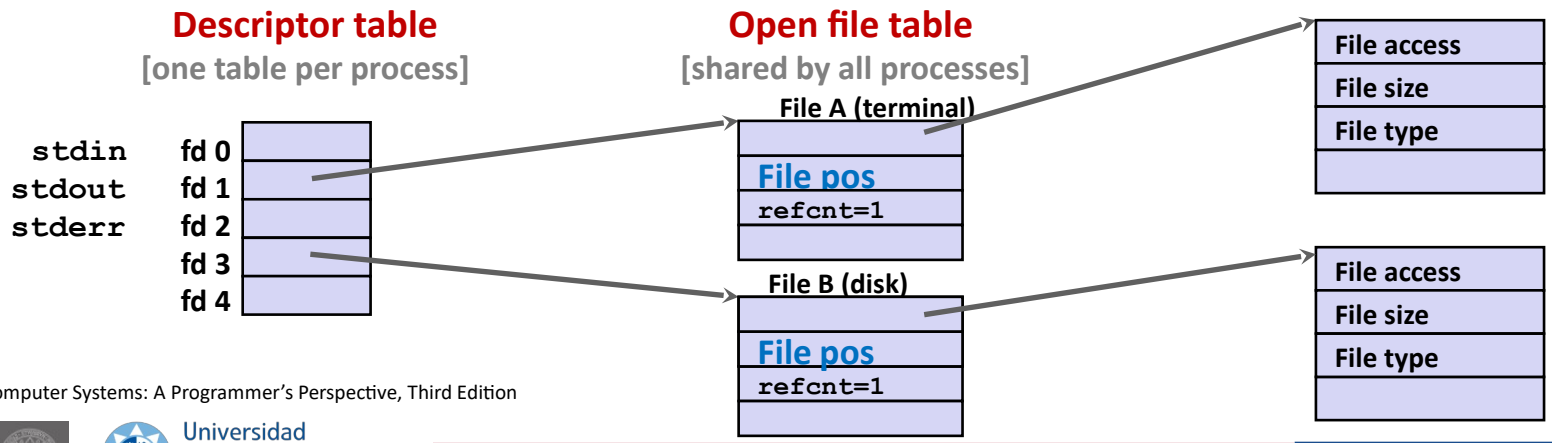
Rutas a ficheros

- La ubicación de los ficheros en la jerarquía de directorios se denota por su **ruta** (*pathname*)
  - Rutas absolutas:
    - Empiezan por '/' e indican la ruta desde el directorio raíz
      - Ejemplo: **`/home/alumno/hello.c`**
  - Rutas relativas
    - Denota la ruta desde el directorio de trabajo actual
      - Ejemplo: **`../alumno/hello.c`** siendo el directorio de trabajo actual **`/home/ccalculo`**



Descriptores de fichero. Ficheros abiertos

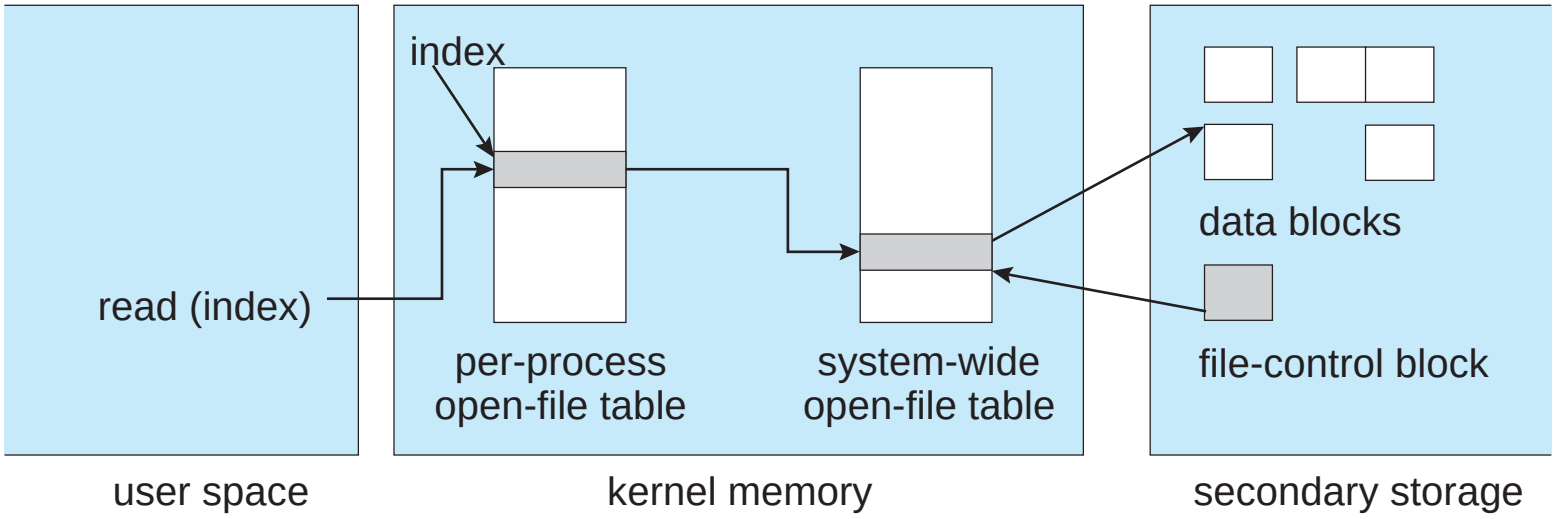
- Operaciones con ficheros → OS debe buscar en el directorio la entrada correspondiente al fichero (identificado por su nombre)
  - Para evitar búsquedas recurrentes, el SO requiere *abrir* un fichero antes de una operación
    → La apertura (vía `open()`) devuelve un **descriptor de fichero**
  - El SO lo utiliza como índice en la tabla de descriptores de fichero (una por proceso)
  - Cada proceso creado por el shell comienza con tres ficheros abiertos (asociados al terminal)
    - 0: standard input (`stdin`); 1: standard output (`stdout`); 2: standard error (`stderr`)
- Ejemplo: Dos descriptores de fichero, dos ficheros abiertos (distintos)
  - El descriptor 1 (*stdout*) apunta al terminal; el descriptor 3 apunta a un fichero en disco
  - La **posición en el fichero** se mantiene por cada fichero abierto



Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Sistema de ficheros: estructuras del kernel

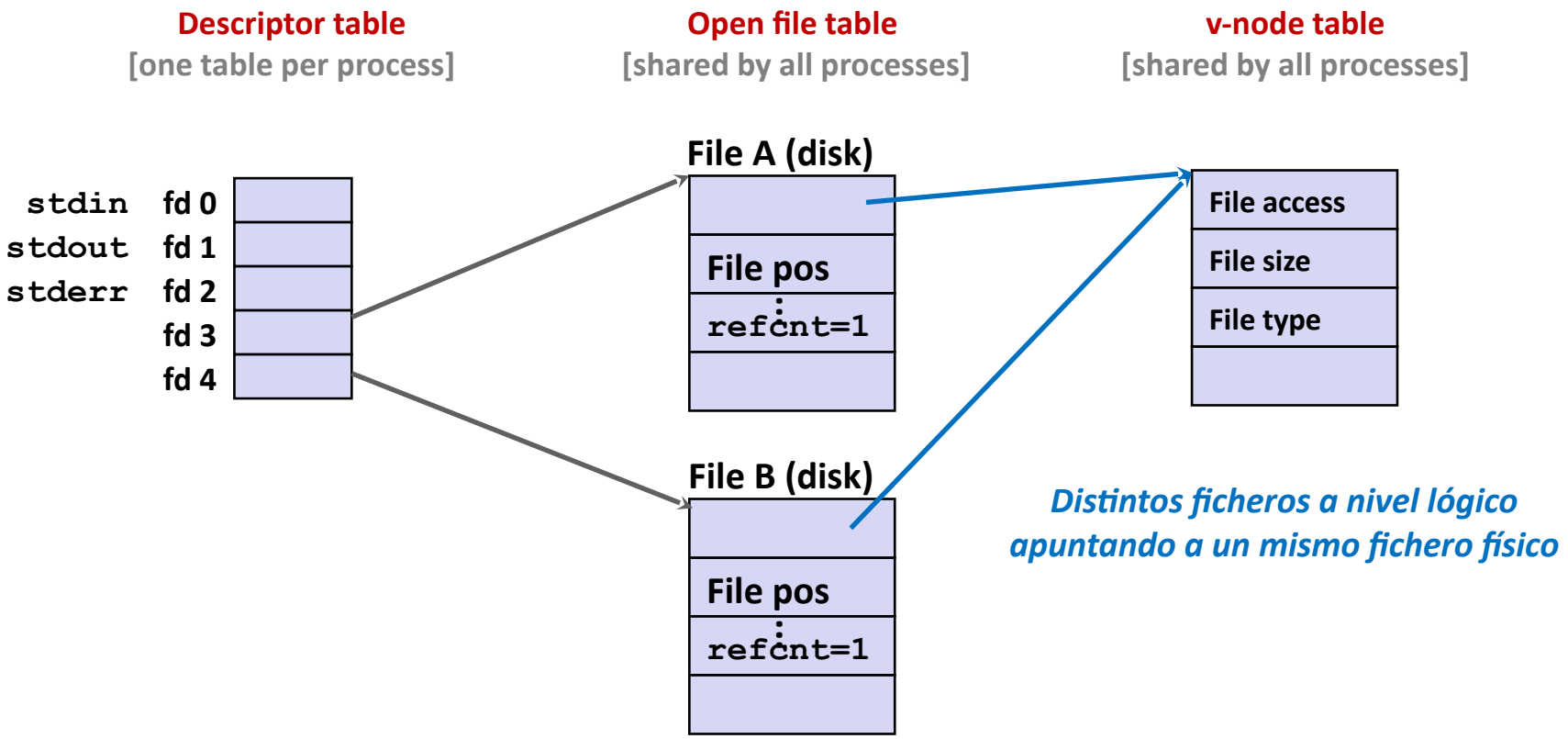
- Los procesos solicitan operaciones sobre ficheros (*`read`*, *`write`*, etc.) indicando el descriptor del fichero devuelto por `open`
- El núcleo del SO mantiene una **tabla de descriptores** por cada proceso y una **tabla de ficheros abiertos** (global)
  - Utiliza el descriptor de fichero para indexar la tabla de descriptores de fichero del proceso
  - La tabla de descriptores de fichero, a su vez, indexa la tabla de ficheros abiertos
    - Contiene información sobre los ficheros en uso por los diferentes procesos.
    - Ejemplo: la posición actual en el fichero
  - La tabla de ficheros abiertos global apunta al **bloque de control de fichero** (FCB)



Silberschatz, Galvin and Gagne. Operating System Concepts. 9th Edition

Compartición de ficheros

- Dos descriptores de fichero pueden compartir el mismo fichero en disco a través de dos entradas distintas en la tabla de ficheros abiertos
  - Ejemplo: Llamar a `open` dos veces con el mismo nombre de fichero como argumento



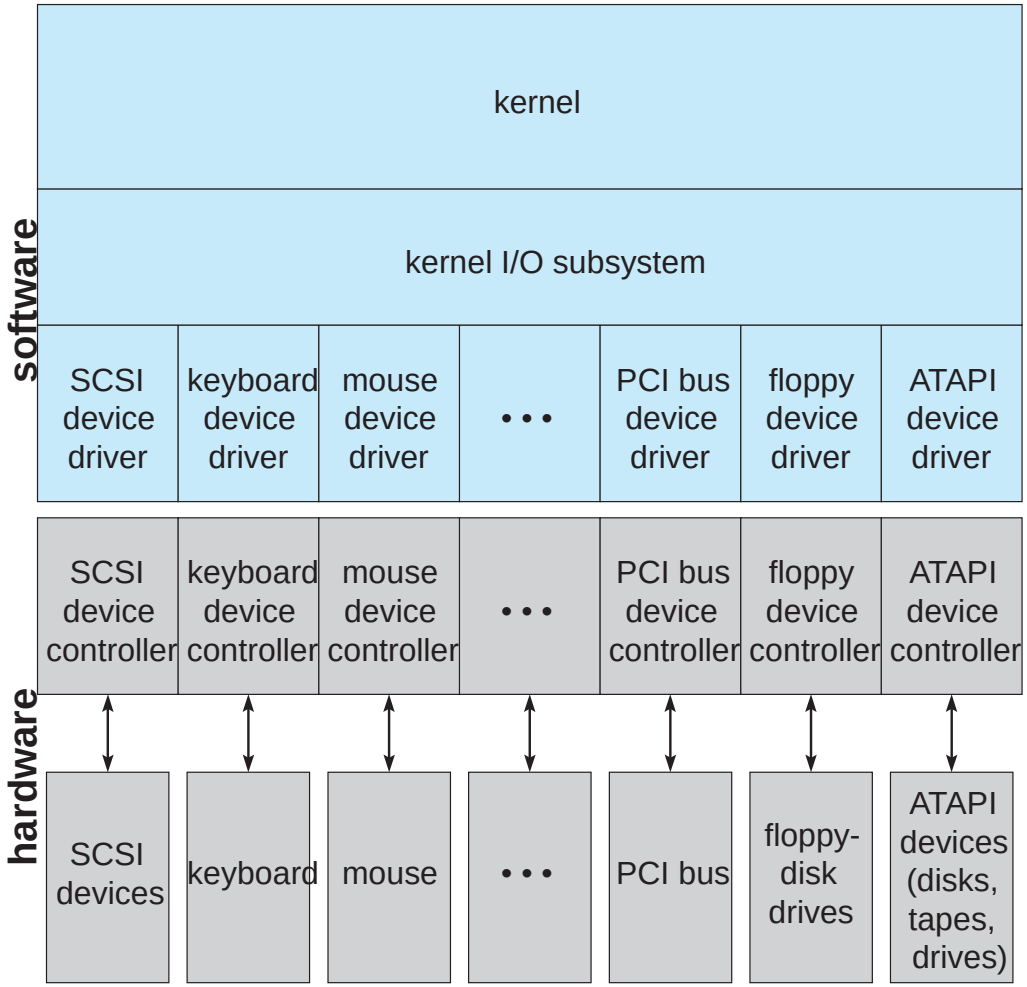
Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Gestión de la E/S: Ficheros abiertos

- El SO mantiene cierta información por cada fichero abierto
  - **Posición actual** en el fichero (*file pointer*)
    - Este puntero es diferente para cada proceso operando en el fichero y, por tanto, debe mantenerse separado de los atributos del fichero (que se guardan en disco)
  - **Contador de aperturas**
    - Conforme se cierran los ficheros, el SO reutiliza las entradas de la tabla de ficheros abiertos para evitar quedarse sin espacio en la tabla. En el caso de que múltiples procesos hayan abierto un mismo fichero, el SO debe esperar hasta que el último proceso lo cierre
  - **Ubicación en disco** del fichero
    - La mayoría de operaciones requieren la modificación de los datos en el fichero. La información necesaria para localizar el fichero en disco se mantiene en memoria para evitar leerla de disco en cada operación de lectura, escritura, etc.
  - **Permisos de acceso**
    - Cada proceso abre un fichero en un determinado modo de acceso. Esta información se mantiene en la tabla de ficheros de cada proceso y se utiliza para permitir o denegar peticiones de E/S subsiguientes

Estructura de E/S del núcleo. *Drivers*

- OS: encapsula las particularidades de los distintos dispositivos de E/S: **manejadores de dispositivo** (*device drivers*)
  - Ocultan al subsistema de E/S del núcleo las diferencias entre las controladoras de dispositivo, presentan una interfaz de acceso a dispositivos uniforme: **independencia de dispositivo**
- Se identifican unas pocas clases generales de dispositivos de E/S.
  - Cada clase se accede a través de un conjunto de funciones estandarizado
- Manejadores de dispositivo (**software**):
  - Módulos del kernel, se ejecutan en modo núcleo
  - Internamente están hechos a medida del dispositivo específico
  - Exportan al kernel una de las interfaces estándar
  - Envía comandos escribiendo patrones de bits en la memoria de la controladora de E/S, para indicarle las acciones que debe emprender.
- Controladoras de dispositivo (**hardware**)
  - Chip que controla y opera la electrónica del periférico concreto, interfaz con el *driver*



Silberschatz, Galvin and Gagne. Operating System Concepts. 9th Edition

Acceso a la E/S

- ¿Cómo usa el programador los dispositivos de E/S?

1 Solicita el servicio al SO, mediante una llamada al sistema.
   - Ejemplo: transfiere 500 bytes del fichero “fich.txt” a dirección de memoria 0x804060.

2 Llamada al sistema (`syscall` en x86): paso a modo núcleo y transfiere el control al SO.

3 El SO organiza y coordina las peticiones de E/S de los distintos procesos, comprueba los permisos, y abstrae la complejidad de los distintos dispositivos

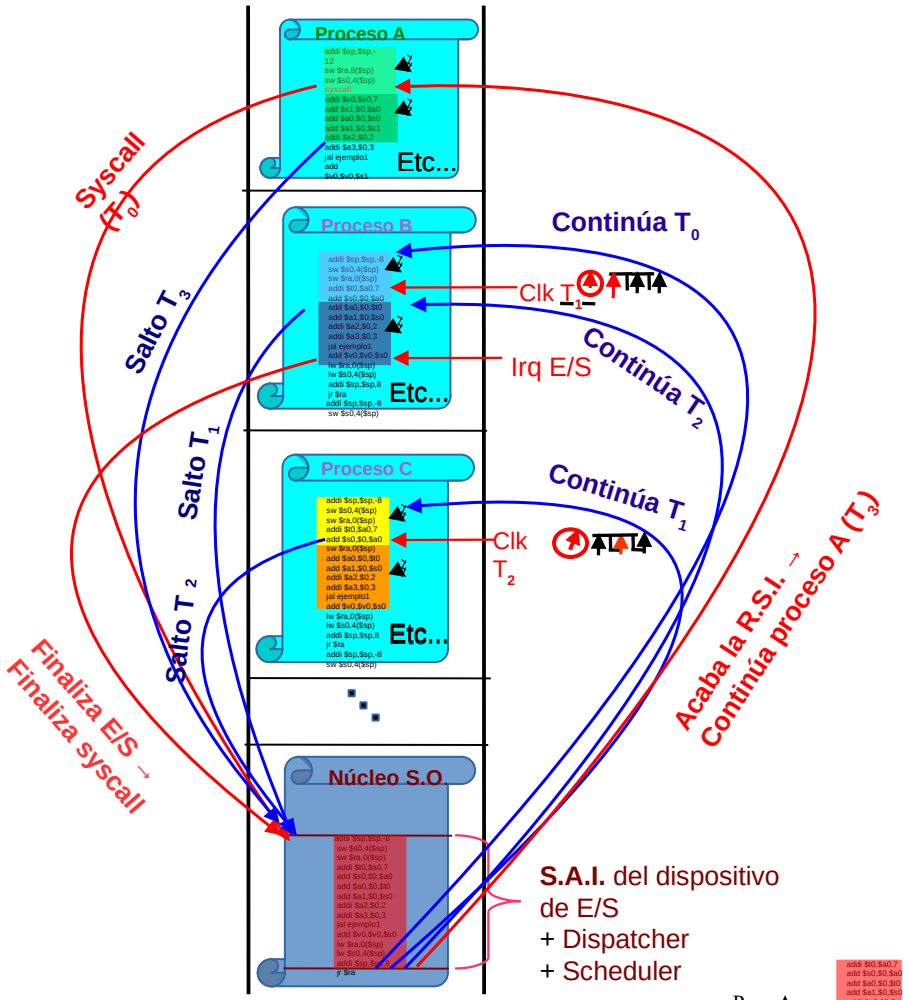
4 El SO usa el manejador de dispositivo adecuado para enviar comandos a la controladora de dispositivo: el *driver* actúa como una especie de “traductor”
   - Recibe como entrada comandos de alto nivel (p.ej., “leer el bloque 123”);
   - Produce como salida instrucciones de bajo nivel, específicas para un hardware concreto (dependientes de la interfaz presentada por el chip de la controladora del dispositivo concreto)
     - Ejemplo: Traduce “Lee el bloque de disco en la dirección disco 1, cilindro 73, pista 2, sector 10” a una ristra de bits que escribirá en la memoria del chip de la controladora

5 El manejador de interrupción asociado al dispositivo se encarga de transferir la información entre el dispositivo y los búferes del driver en memoria principal

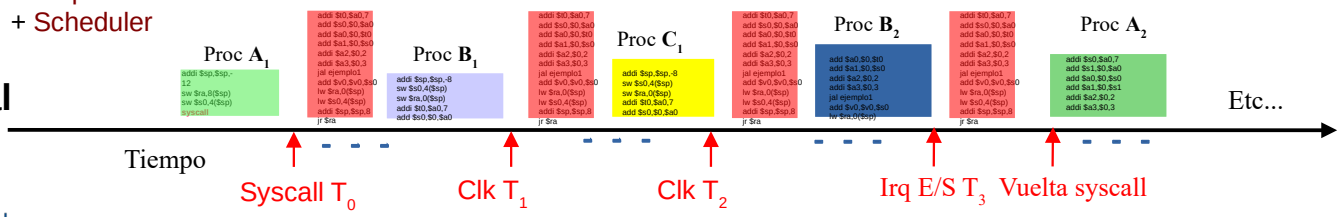
6 Finalmente, si es una lectura, el kernel del SO copia los datos a la memoria del proceso

7 Al terminar de servir la petición: regreso al código de usuario en el programa llamador
   - Una llamada puede fallar (si no se cumple algún requisito, o se proporcionan parámetros erróneos)
   - El SO maneja el fallo y protege al resto de procesos, informa del fallo al proceso invocador

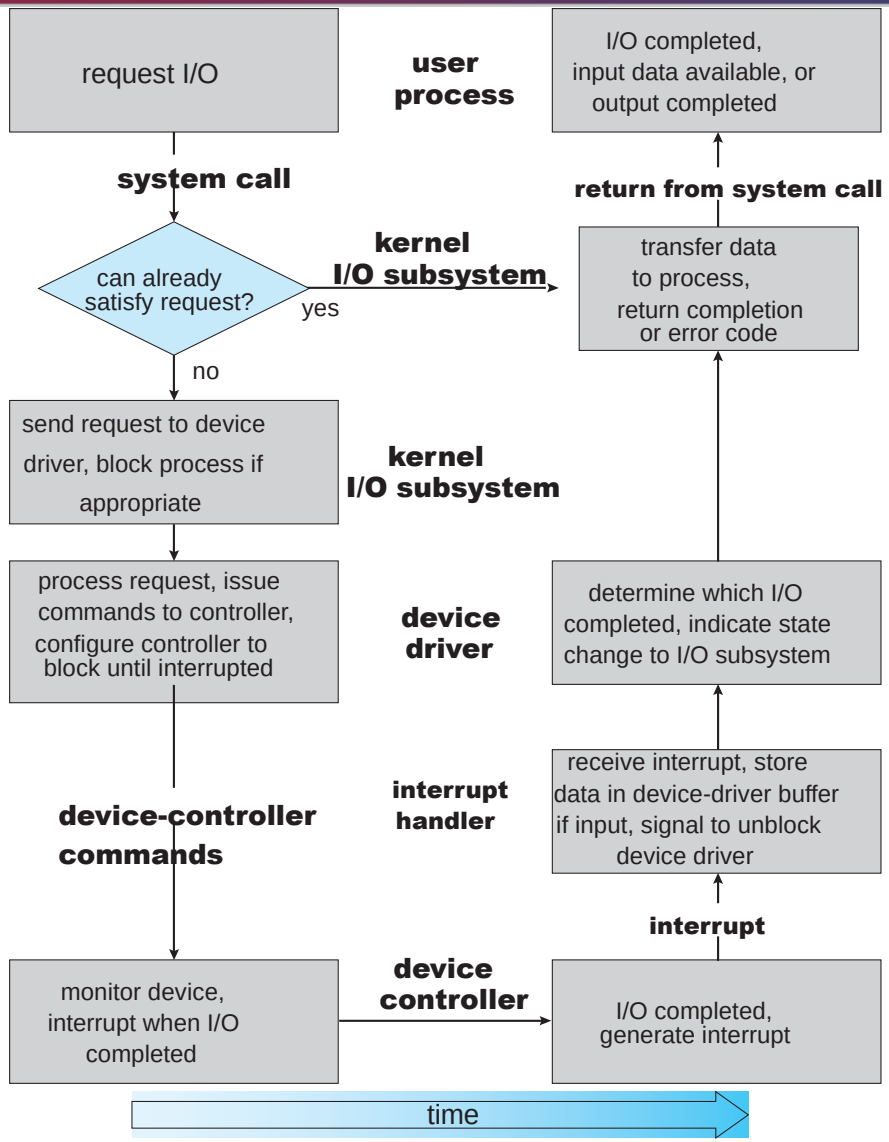
Gestión de procesos y acceso a la E/S



- Esquema de tratamiento de una llamada al sistema:
  1. Proceso A realiza llamada al S.O. (syscall).
  2. Núcleo del S.O. lanza orden de E/S al dispositivo y, como la respuesta va a llevar un tiempo, pasa el control a otros procesos (B, C) mientras el dispositivo acaba su tarea.
  3. Dichos procesos (B, C) pueden ir siendo a su vez interrumpidos por el planificador...
  4. … hasta que el dispositivo de E/S termina su tarea y lanza interrupción (irq).
  5. Núcleo entonces vuelve a tomar el control, y despierta al proceso bloqueado (A).



Ciclo de vida de una petición de E/S



Silberschatz, Galvin and Gagne. Operating System Concepts. 9th Edition