UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL CÓRDOBA - EXTENSIÓN ÁULICA
BARILOCHE
INGENIERÍA EN SISTEMAS DE INFORMACIÓN
AÑO LECTIVO 2025

Sistemas Operativos

Resumen para el primer parcial

Unidades 1, 2 y 3 de la cátedra

Profesor: Eduardo Tapia
Ayudante:
Fecha: 24/05/2025

Alumno: Ricardo Nicolás Freccero
Número de legajo: 415753

Índice

1. Unidad 1 - Concepto de Sistemas Operativos

1.1. ¿Qué es el sistema operativo?

Un sistema operativo es un programa que controla la ejecución de aplicaciones y programas. Es quien se encarga de "conectar" o "comunicar" las apliaciones con el hardeware de la computadora. Se puede considerar que un sistema operativo tiene tres objetivos:

- **Facilidad de uso.** Un sistema operativo facilita el uso de un computador.
- **Eficiencia.** Un sistema operativo permite que los recursos de un sistema de computación se puedan utilizar de una manera eficiente.
- **Capacidad para evolucionar.** Un sistema operativo se debe construir de tal forma que se puedan desarrollar, probar e introducir nuevas funciones en el sistema sin interferir con su servicio.

1.1.1. El sistema operativo como una interaz de usuario/computador

El hardware y software que le permiten al usuario acceder y utilizar las aplicaciones y programas de una computadora se pueden ver de forma jerárquica como muestra la Figura 1. Al usuario no le suelen interesar los detalles del hardware de la computadora y vé al sistema de computación como un conjunto de aplicaciones. Por otro lado, cada aplicación se puede expresar en un lenguaje de programación y normalmente es desarrollada por un programador. Al programador sí le importan los detalles del hardware, pero no se comunica casi nunca directamente con él ya que suele ser una tarea extremadamente compleja. Para eso existe un *conjunto de programas de sistema* que le permiten al programador comunicarse de una manera mas eficiente con el hardware. El programa de sistema mas importante es el **sistema operativo** que proporciona una interfáz entre el software y el hardware, actuando como mediador y facilitando el acceso del programador a las utilidades y servicios del sistema.

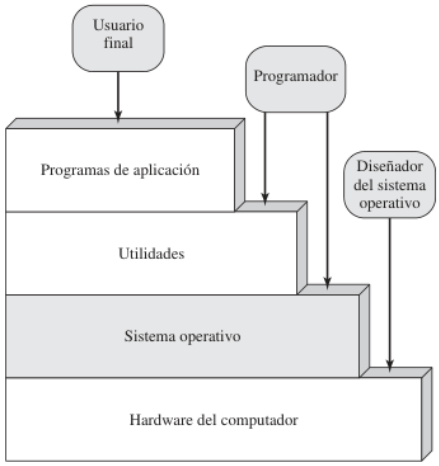

Figura 2.1. Capas y vistas de un sistema de computación.

Figura 1: Imagen sacada de (Stallings, 2005). Jerarquía de un sistema de computación.

El sistema operativo proporciona normalmente servicios en las siguientes áreas:

- **Desarrollo de programas.** Ofrece editores y depuradores para asistir al programador en la creación de programas y aplicaciones.
- **Ejecución de programas.** Se encarga de realizar todos los pasos necesarios para la ejecución de programas en nnombre del usuario.
- **Acceso a dispositivos de E/S.** Facilita el acceso de los programadores a estos dispositivos.
- **Acceso al sistema.** Protege los recursos y los datos,e vitando el uso no autorizado de los usu-sarios y resolviendo conflictos en el caso de conflico de recursos.
- **Detección y respuesta a errores.** Debe proporcionar una respuesta a cualquier error que pueda surgir de manera que se elimine la condición de error, suponiendo el menor impacto en las aplicaciones que están en ejecución.
- **Contabilidad.** Recoge estadísticas de uso de los diferentes recursos y monitoriza parámetros de rendimiento. Esta información es útil para anticipar las necesidades de mejoras futuras y para optimizar el sistema a fin de mejorar su rendimiento.

1.1.2. El sistema operativo como gestor de recursos

El sistema operativo dirige al procesador en el uso de los recursos de la computadora y en el tiempo que debe tomarse para la ejecución de otros programas. Para eso, el sistema operativo cede el control para que el procesador realice sus tareas y luego retoma el control para decirle qué sigue.

El sistema operativo decide cuándo un programa en ejecución puede utilizar un dispositivo de E/S y controla el acceso y uso de los ficheros, además decide cuánto tiempo debe tomarse cada procesador para la ejecución de un programa.

1.1.3. Facilidad de evolución de un sistema operativo

Un sistema operativo debe evolucionar en el tiempo por las siguientes razones:

- Actualizacioes de hardware y nuevos tipos de hardware.
- Nuevos servicios. (Por ejemplo, si es dificil mantener un buen rendimiento con las herramientas existentes, se pueden añadir al sistema operativo nuevas herramientas de medida y control.)
- Resolución de fallos.

1.2. Evolución de los sistemas operativos

1.2.1. Procesamiento en serie

El programador interactuaba directamente con el hardware de la computadora, que contaba con una consola, luces, interruptores, algún dispositivo de entrada y una impresora; *no existía ningún sistema operativo*. Los programas en código máquina se cargaban a través del dispositivo de entrada y si un error provocaba la parada del programa, las luces indicaban la condición de error. El programador podía examinar los registros del procesador y memoria principal para determinarl la causa del error. Si el programa terminaba de forma normal, la salida se imprimía.

1.2.2. Sistemas en lotes sencillos

Se empieza a usar un software denominado **monitor**, que permite que el usuario no tenga que acceder directamente a la máquina. En su lugar, el la computadora recibe una secuencia de trabajos que va a usar el monitor. El monitor de encarga de leer uno a uno cada trabajo y decirle al procesador que los vaya realizando en ese orden.

1.2.3. Sistemas en lotes multiprogramados

Aún usando un sistema de lotes sencillos, el procesador está haciendo nada la mayoría del tiempo ya que este es mucho mas rápido que los dispositivos de E/S. La computadora pasa mas tiempo buscando la información que tiene que procesar el procesador que procesando esa información. Lo que se puede hacer entonces es que mientras se está esperando por la E/S, se le puede asignar al procesador otro trabajo que no esté esperando por una operación de E/S.

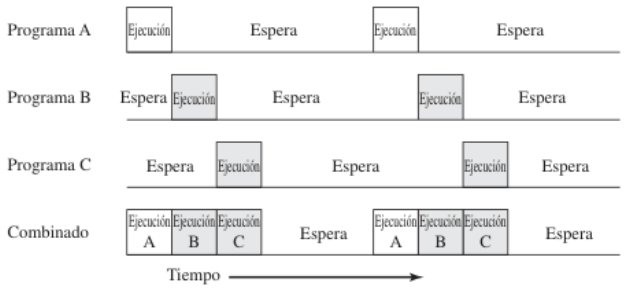

Figura 2: Imagen sacada de (Stallings, 2005). Ejemplo de multiprogramación con tres programas.

1.2.4. Sistemas de tiempo compartido

Son sistemas que comparten el tiempo de programación entre múltiples usuarios.

1.3. Desarrollos que llevaron a los sistemas operativos modernos

1.3.1. Arquitectura micronúcleo o microkernel

Antes, la mayoría de los sistemas operativos estaban formados por **núcleos monolíticos**. Estos núcleos proporcinaban la mayoría de las funcionalidades consideradas propias del sistema operativo, incluyendo planificación, los sistemas de ficheros, las redes, los controladores de dispositivos, la gestióno de memoria, etc. Una **arquitectura microkernel** asigna sólo unas pocas funciones esenciales al kernel, incluyendo los espacios de almacenamiento, comunicación entre procesos, y planificación básica.

1.3.2. Multithreading

Es una técnica en la cual un proceso, ejecutando una aplicación, se divide en una serie de *hilos* o *threads*.

- **Thread o hilo.** Es una unidad de trabajo. Incluye el contexto del procesador (que contiene el contador del programa y el puntero de pila) y su propia área de datos para una pila (para posibilitar el salto a subrutinas). Un hilo se ejecuta secuencialmente y se puede interrumpir para dar paso a otro hilo.
- **Proceso.** Es una colección de uno o más hilos y sus recursos de sistema asociados. Es un programa en ejecución.

Esta técnica es útil para aplicaciones que llevan a cabo tareas que no necesitan correrse en serie, es decir, que se pueden ejecutar en simultáneo.

1.3.3. Multiprocesamiento simétrico (SMP)

Se refiere a la arquitectura del hardware de la computadora y al comportamiento del sistema operativo que explota dicha arquitectura. Tiene las siguientes características:

- Tiene múltiples procesadores.
- Los procosadores comparten las mismas utilidades de momeira principal y de E/S.
- Todos los procesadores pueden realizar las mismas funciones.

El sistema operativo de un SMP planifica procesos o hilos a través de todos los procesadores.

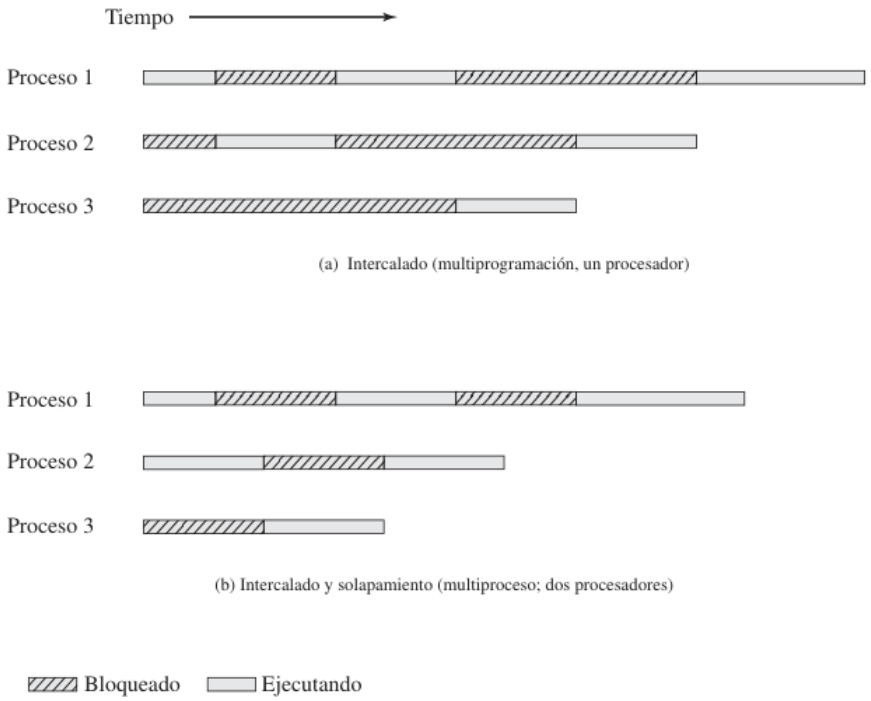

Figura 3: Imagen sacada de (Stallings, 2005). Multiprogramación y multiproceso.

1.3.4. Sistema operativo distribuído

Da la ilusión de tener un solo espacio de memoria principal y un solo espacio de memoria secundario, cuando tenemos un “cluster” de computadoras (varias computadoras que operan como si fuese una sola).

1.3.5. Diseño orientado a objetos

Introduce una disciplina al proceso de añadir extensiones modulares a un pequeño núcleo. Permite a los programadores personalizar un sistema operativo sin eliminar la integridad del sistema.

1.4. Sistemas UNIX tradicionales

UNIX es un sistema operativo que se desarrollo inicialmente en los laboratorios de Bell y se hizo operacional en 1970. En la figura 4 se puede ver la arquitectura general de UNIX.

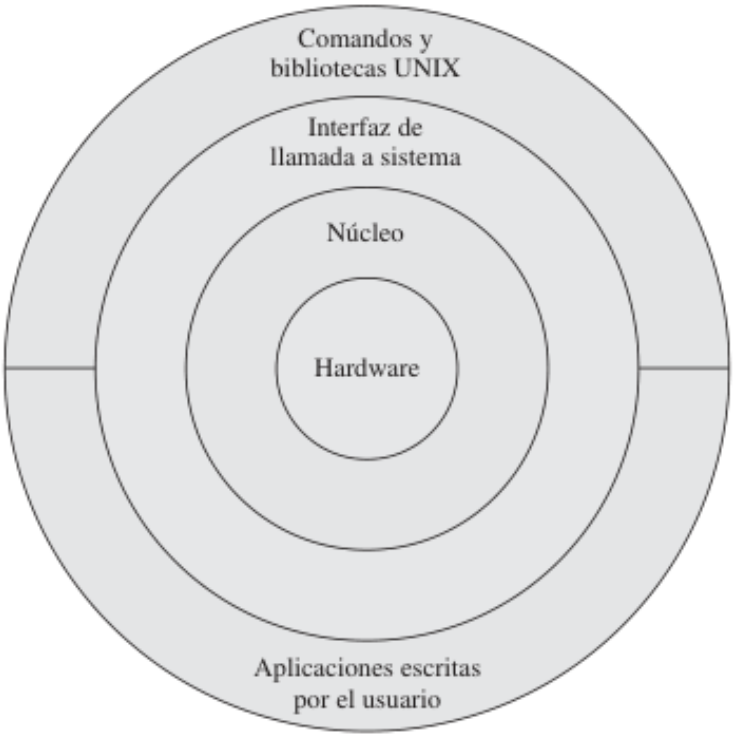

Figura 4: Imagen sacad de (Stallings, 2005). Arquitectura general de UNIX.

1.5. Linux

Linux comenzó como una variante UNIX. Linus Torvalds, un estudiante finlandés de informática, escribió la versión inicial. Linux es un sistema UNIX completo de software libre (cualquier persona puede ver y modificar el código fuente).

La mayoría de los núcleos de Linux son monolíticos. Aunque no usa la técnica de microkernel, logra muchas de las ventajas potenciales de esta por medio de su arquitectura modular particular. Linux está estructurado como una colección de módulos, algunos de los cuales pueden cargarse y descargarse automáticamente bajo demanda. Estos bloques se denominan **módulos cargables**.

Los módulos cargables de Linux tienen dos características importantes:

- **Enlace dinámico**. Un módulo de núcleo puede cargarse y enlazarase al núcleo mientras el núcleo está en memoria y ejecutándose. Un módulo también puede desenlazarase y eliminarse de la memoria en cualquier momento.

- **Módulos apilables**. Los módulos se gestionan como una jerarquía. Cuando un módulo superior en la jerarquía referencia a un módulo inferior, el primero actúa como módulo cliente y el segundo como biblioteca.

2. Unidad 2 - Administración y Gestión de Archivos

Los **archivos** son unidades logicas de informacion que pueden ser leidas o creadas por los procesos. La informacion que se almacena en los archivos debe ser persistente. Un archivo debe desaparecer solo cuando su propietario lo remueve de manera explicita.

Los archivos son administrados por el sistema operativo. La parte del sistema operativo que trata con los archivos se conoce como **sistema de archivos**.

2.1. Archivos

Los archivos proporcionan una manera de almacenar informacion en el disco y leerla despues. Cuando un proceso crea un archivo le proporciona un nombre. Cuando el proceso termina, el archivo sigue existiendo y puede ser utilizado por otros procesos mediante su nombre.

Algunos sistemas de archivos, como el de UNIX, diferencian las letras mayusculas de las minusculas, mientras que otros no.

Existen varios sistemas de archivos que vamos a analizar mas adelante. Por ahora solo vamos a decir que Windows 95 y 98 usan el sistema de archivos **FAT-16**. Windows 98 extendio FAT-16 e introdujo **FAT-32** pero estos dos sistemas son bastante similares. Las versiones mas nuevas de Windows admiten ambos sistemas FAT, que en realidad ya son obsoletos, pero tienen un sistema de archivos nativo que se conoce como **NTFS**.

Muchos sistemas operativos aceptan nombres de archivos en dos partes, separadas por un punto. La parte que va despues del punto se conoce como la **extension del archivo** y suele indicar algo acerca de la naturaleza de ese archivo.

En algunos sistemas (como UNIX) las extensiones de archivo son solo convenciones y no son impuestas por el sistema operativo. Funcionan mas como un recordatorio para el propietario que como un medio para transportar informacion a la computadora. Sin embargo, en otros sistemas (como Windows) éste es consciente de las extensiones y les asigna significado. Los usuarios pueden registrar extensiones y asignar programas a cada una de manera que cuando se le hace doble click al nombre del archivo, el programa asignado a su extension se inicia con ese archivo como parametro.

2.1.1. Estructura de archivos

Los archivos se pueden estructurar de varias formas:

- **Secuencias de byte sin estructura.** El sistema operativo no sabe, ni le importa que hay en el archivo. Esto provee la maxima flexibilidad ya que los programas de usuario pueden colocar cualquier cosa que quieran en sus archivos y denominarlos de cualquier manera conveniente.
- **Secuencia de registros.** Un archivo es una secuencia de registros de longitud fija, cada uno con cierta estructura interna. Estos sistemas se usaban cuando todavia se utilizaban las tarjetas perforadas de 80 columnas, de manera que cada registro consistia de 80 caracteres, simulando la tarjeta.

- **Arbol.** Un archivo consiste en un arbol de registros, donde no todos son necesariamente de la misma longitud; cada uno de llos contiene un campo llave en una posicion fija dentro del registro. El arbol se ordena con base en el campo llave para permitir una busqueda rapida por una llave especifica.

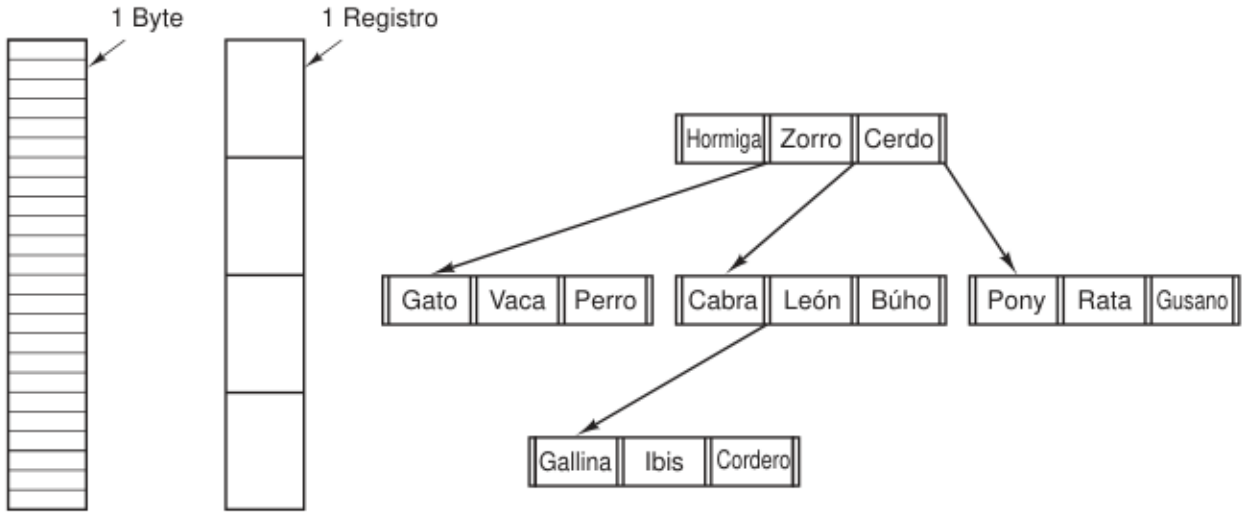

Figura 5: Imagen sacada de (Tanenbaum, 2009). Tres tipos de archivos.

2.1.2. Tipos de archivos

Muchos sistemas operativos soportan varios tipos de archivos. Por ejemplo, UNIX y Windows tienen archivos y directorios regulares. UNIX también tiene archivos especiales de caracteres y de bloques. Los **archivos regulares** son los que contienen información del usuario. Los **directorios** son sistemas de archivos para mantener la estructura del sistema de archivos. Los **archivos especiales de caracteres** se relacionan con la E/S y se utilizan para modelar dispositivos de E/S en serie. Los **archivos especiales de bloques** se utilizan para modelar discos.

Por lo general, los **archivos regulares** son archivos ASCII o binarios. Los archivos ASCII consisten en líneas de texto, se pueden mostrar e imprimir como están, y se pueden editar con cualquier editor de texto. Los archivos binarios tienen código binario y una cierta estructura interna conocida para los programas que los utilizan. La Figura 6 muestra dos archivos binarios de las primeras versiones de UNIX. El primero es un archivo binario ejecutable. El segundo es una colección de procedimientos (módulos) de biblioteca compilados, pero no enlazados.

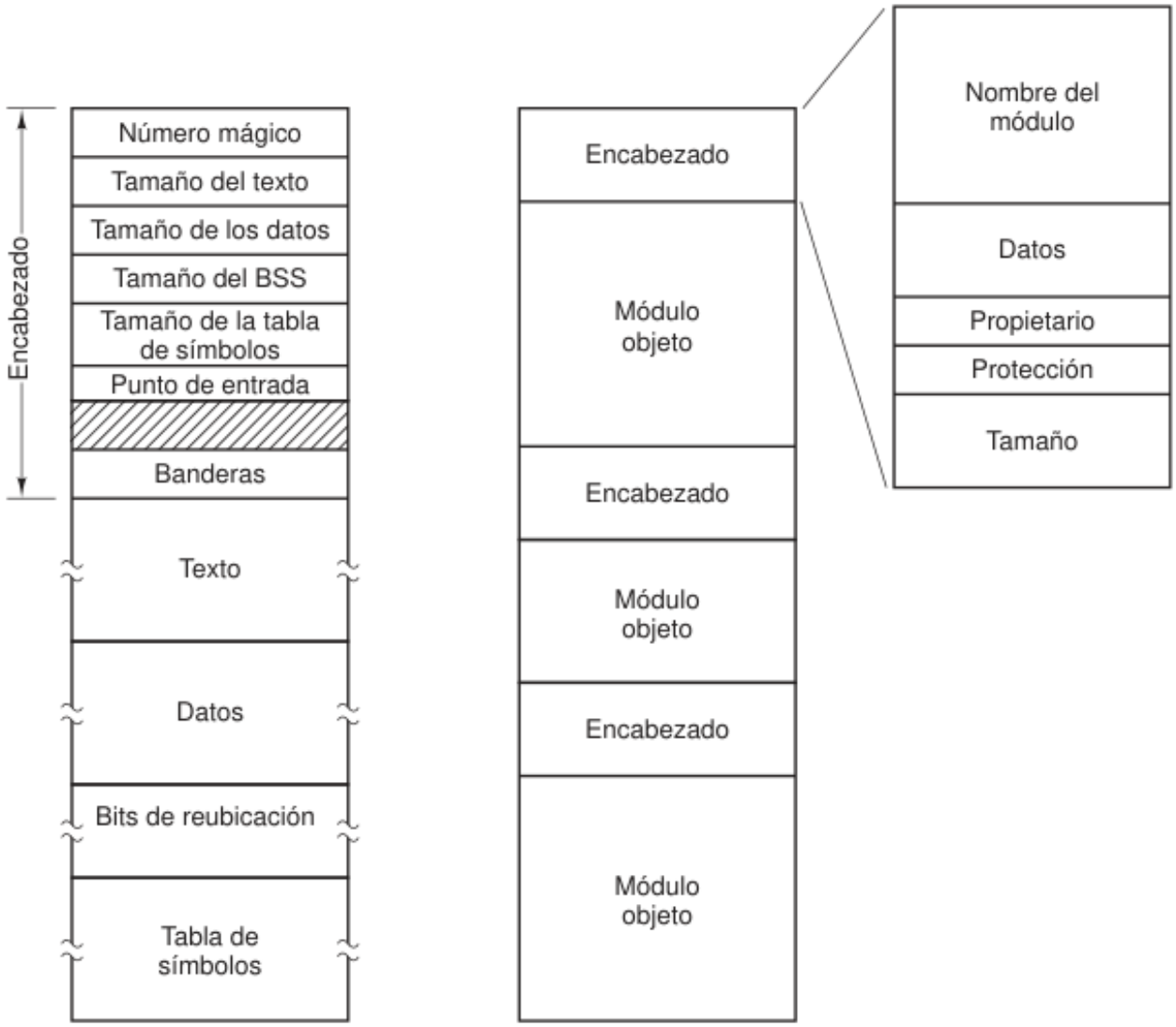

Figura 6: Imagen sacada de (Tanenbaum, 2009). Dos archivos binarios distintos.

2.1.3. Acceso a archivos

Los primeros sistemas operativos proporcionaban sólo un tipo de acceso: **acceso secuencial**. En estos sistemas, sólo se podían leer los bytes en orden secuencial, pero no se podía saltar entre bytes.

Mas adelante se crearon los **archivos de acceso aleatorio** en los que se pueden leer sus bytes en cualquier orden. En estos se pueden usar los métodos `seek` (para establecer la posición desde la que se quiere comenzar a leer) y `read` (para comenzar a leer de manera secuencial desde la posición actual).

2.1.4. Atributos de archivos

Todo archivo tiene un nombre y sus datos. Además, todos los sistemas operativos asoian otra información con cada archivo. A estos elementos adicionales se los conoce como **atributos**. A continuación se muestra una tabla con varios atributos, aunque no son todos, que *pueden* tener los archivos

en un sistema operativo.

Atributo	Significado
Protección	Quién puede acceso al archivo y en qué forma
Contraseña	Contraseña necesaria para acceder al archivo
Creador	ID de la persona que creó el archivo
Propietario	El propietario actual
Bandera de sólo lectura	0 para lectura/escritura; 1 para sólo lectura
Bandera oculto	0 para normal; 1 para que no aparezca en los listados
Bandera del sistema	0 para archivos normales; 1 para archivo del sistema
Bandera de archivo	0 si ha sido respaldado; 1 si necesita respaldarse
Bandera ASCII/binario	0 para archivo ASCII; 1 para archivo binario
Bandera de acceso aleatorio	0 para sólo acceso secuencial; 1 para acceso aleatorio
Bandera temporal	0 para normal; 1 para eliminar archivo al salir del proceso
Banderas de bloqueo	0 para desbloqueado; distinto de cero para bloqueado
Longitud de registro	Número de bytes en un registro
Posición de la llave	Desplazamiento de la llave dentro de cada registro
Longitud de la llave	Número de bytes en el campo llave
Hora de creación	Fecha y hora en que se creó el archivo
Hora del último acceso	Fecha y hora en que se accedió al archivo por última vez
Hora de la última modificación	Fecha y hora en que se modificó por última vez el archivo
Tamaño actual	Número de bytes en el archivo
Tamaño máximo	Número de bytes hasta donde puede crecer el archivo

Figura 7: Imagen sacada de (Tanenbaum, 2009). Algunos atributos de archivos.

2.1.5. Operaciones de archivos

Las llamadas al sistema (system calls) mas comunes relacionadas con los archivos son las siguientes:

1. `Create`. El archivo se crea sin datos.
2. `Delete`. Se elimina el archivo, liberando el espacio en disco.
3. `Open`. Se llevan los atributos y la lista de direcciones de disco a memoria principal para tener acceso rápido al archivo.
4. `Close`. Se libera el espacio que ocupaba el archivo en la memoria principal.
5. `Read`. Se leen los datos del archivo desde la posición actual.
6. `Write`. Se escriben datos en el archivo desde la posición actual.
7. `Append`. Se escriben datos al final del archivo.
8. `Seek`. Se modifica la posición actual del puntero del archivo.

9. `Get attributes`. Se obtienen los atributos de un archivo.
10. `Set attributes`. Algunos de los atributos los puede establecer el usuario y se pueden modificar después de haber creado el archivo.
11. `Rename`. Se cambia el nombre del archivo.

2.2. Directorios

Para llevar el registro de los archivos, los sistemas de archivos por lo general tiene **directorios**, que en muchos sistemas también son archivos.

2.2.1. Sistemas de direcorios de un solo nivel

La forma más simple de un sistema de directorios es tener un directorio que contenga todos los archivos. Como si en nuestra computadora solo tuviesemos una sola carpeta, y en ella existan todos los archivos que necesitamos.

2.2.2. Sistemas de directorios jerárquicos

El sistema anterior es útil para aplicaciones simples, pero nos podemos imaginar que sería un caos si fuese el caso de nuestra computadora. Lo que se hace ahora es tener un **árbol de directorios**. De esta manera, existe un directorio raíz (que en UNIX suele ser "/") del que se desprenden tantos directorios como quiera el usuario para agrupar los archivos de la forma mas conveniente.

2.2.3. Nombres de rutas

Al tener el File System organizado como un árbol de directorios, se necesita cierta forma de especificar los nombres de los archivos y existen dos métodos para lograr esto.

- **Nombre de ruta absoluto.** Consiste en usar el camino desde el directorio raíz hasta el archivo. Un ejemplo en Linux es `/home/ricman/uni/apuntes/sop/parcial1/resumen.pdf`, que es la dirección en donde se encuentra este documento.
- **Nombre de ruta relativa.** Consiste en nombrar al archivo siguiendo el camino desde el **directorio actual**, que es el directorio de trabajo, en el que está parado el usuario, hasta el archivo deseado. Por ejemplo, si nuestro directorio actual es `/home/ricman/uni/`, y quiero referenciar este archivo, su nombre de ruta relativa sería `apuntes/sop/parcial1/resumen.pdf`.

La mayoría de sistemas operativos que proporcionan un sistema de directorios jerárquico tienen dos entradas especiales en cada directorio "."(punto) y ".."(puntopunto). *Punto* se refiere al directorio actual; *puntopunto* se refiere al directorioi padre del directorio actual. Esto es útil cuando queremos acceder a archivos que se encuentran en el directorio padre del directorio actual, pero no queremos usar una ruta absoluta para acceder a él.

El directorio *punto* suele ser utilizado cuando, por ejemplo, queremos copiar algún archivo de un directorio externo al directorio actual. Para eso podemos usar el comando `cp` (copy) y pasarle primero el nombre del archivo que queremos copiar y luego el directorio actual usando *punto*.

```
cp /home/ricman/uni/apuntes/hola.txt .
```

Lo anterior es lo mismo que decir

```
cp /home/ricman/uni/apuntes/hola.txt /home/ricman/uni/apuntes/sop/parcial1/
```

2.2.4. Operaciones de directorios

Al igual que con los archivos, podemos realizar varias operaciones con los directorios. En el libro (Tanenbaum, 2009) se nombran unas operaciones que me parece que son viejisimas y no las vamos a usar en la cátedra. Pongo a continuación las que yo considero que son equivalentes y sí vamos a usar en la cátedra:

1. `mkdir` (MaKe DIRectory). Crea un directorio vacío, excepto por *punto* y *puntopunto* que el sistema coloca de manera automática.

2. `rm -r` (RemoVe). En realidad `rm` es un comando que se utiliza para borrar archivos, pero al pasarle el argumento `-r` es capaz de borrar directorios.

3. `cd` (Change Directory). Este comando los usamos para movernos entre los directorios del File System.

4. `mv` (MoVe). Al igual que `rm`, el comando `mv` se usa para "mover" archivos o directorios de un lugar a otro dentro del File System. Esto lo hace tomando como primer parámetro el archivo o directorio que deseamos mover, y como segundo el lugar a donde lo queremos mover, entonces crea una copia del archivo en la dirección especificada y borra el original. Esto lo podemos usar para *renombrar* directorios si le decimos que lo mueva desde un directorio hacia el mismo lugar pero con otro nombre. Por ejemplo `mv /home/ricman/uni/ /home/ricman/versi` renombra el directorio "uni" como "versi".

5. `ln` (LiNk). Crea un **vínculo** desde un archivo existente hasta el nombre especificado por la ruta. De esta forma, el mismo archivo puede aparecer en varios directorios.

Una variante sobre la idea de vincular archivos es el **vínculo simbólico**. En vez de tener dos nombres que apunten a la misma estructura de datos interna que representa un archivo (como hace un vínculo), se puede crear un nombre que apunte a un archivo que nombre a otro.

2.3. Implementación de File Systems

Ya vimos qué son los archivos y cuáles son sus propiedades, y qué son los directorios y cuáes son sus propiedades. Ahora vamos a ver cómo se almacenan, cómo se administra el espacio en el disco y cómo hacer que todo funcione con eficiencia y confiabilidad.

2.3.1. Distribución del sistema de archivos

Los sistemas de archivos se alamacenan en discos. La mayoría de discos se pueden dividir en una o más particiones, con sistemas de archivos independientes en cada partición. El sector 0 del disco se conoce como el **MBR** (*Master Boot Record*) y se utiliza para arrancar la computadora. El final del MBR contiene la tabla de particiones, la cual proporciona las direcciones de inicio y fin de cada partición. Una de las particiones en la tabla se marca como activa. Cuando se arranca la computadora, el BIOS lee y ejecuta el MBR. Lo primero que hace el programa MBR es localizar la partición activa, leer su primero bloque, conocido como el **bloque de arranque**, y ejecutarlo. El programa en el bloque de arranque carga el sistema operativo contenido en esa partición.

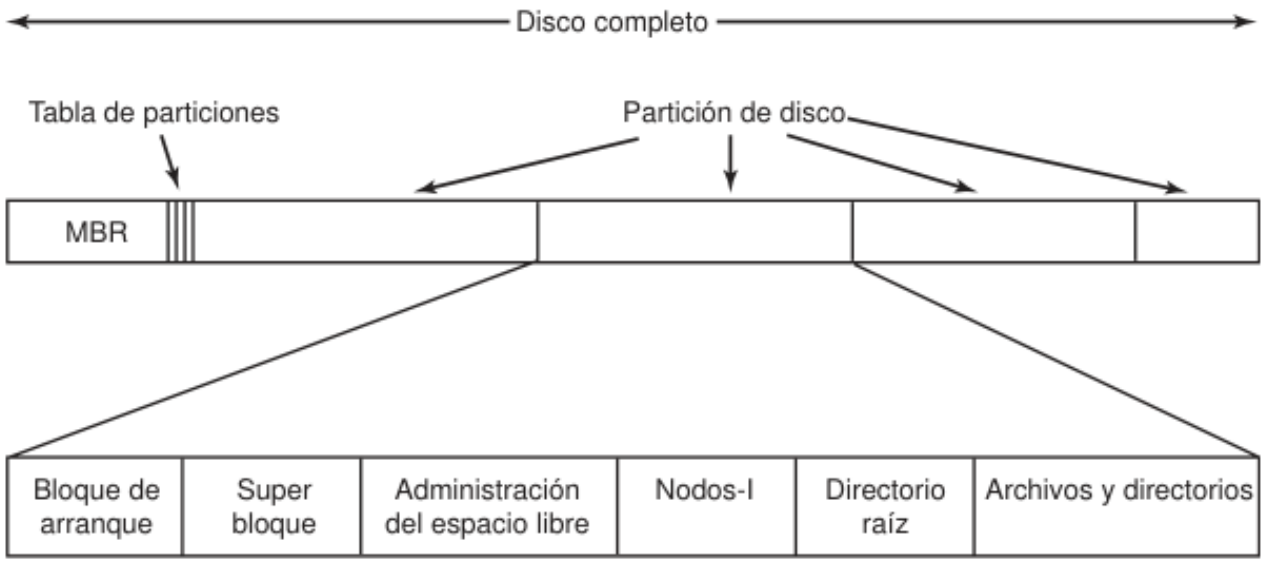

Figura 8: Imagen sacada de (Tanenbaum, 2009). Una posible distribución del sistema de archivos.

2.3.2. Implementación de archivos

Existen varias formas para implementar el almacenamiento de archivos.

Asignación contigua

El esquema de asignación más simple es almacenar cada archivo como una serie contigua de bloques de disco. Así, en un disco con bloques de 1 KB, a un archivo de 50 KB se le asignarían 50 bloques consecutivos.

Este sistema es facil de implementar ya que solo hay que saber dónde empieza y dónde termina cada archivo, y el rendimiento de lectura es excelente ya que se lee secuencialmente. Sin embargo, tiene una gran desventaja debido a que con el transcurso del tiempo, los discos se fragmentan. Podemos ver en la siguiente figura que a medida que vamos eliminando o modificando los archivos, empiezan a aparecer espacios vacíos entre archivos. Y si después queremos reutilizar ese espacio tendríamos que estar viendo que el archivo que queramos meter ahí entre justo en el hueco para no desaprovechar

espacio. Otra opción sería compactar todo el espacio, moviendo todos los archivos para volver a dejar el espacio libre al fondo del disco. Ambas opciones son extremadamente costosas y no termina siendo "worth".

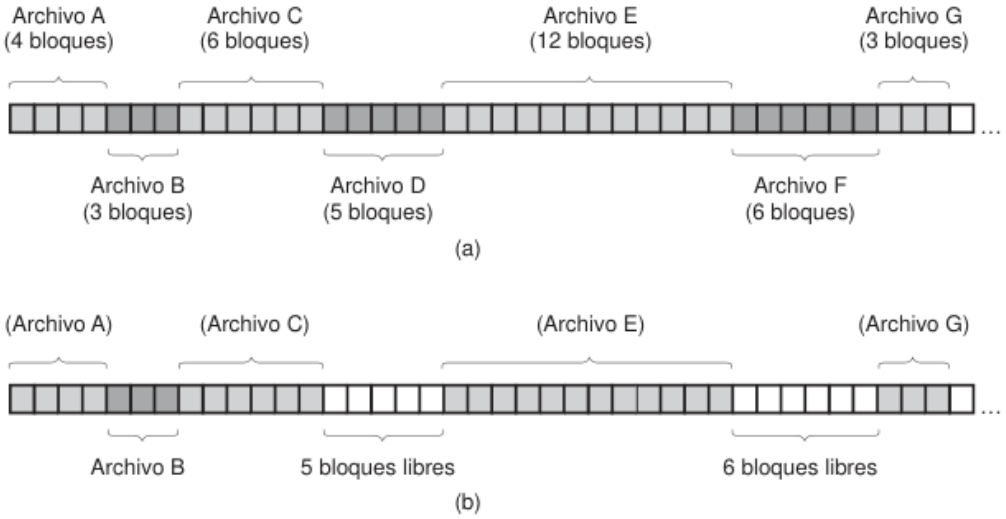

Figura 9: Imagen sacada de (Tanenbaum, 2009). (a) Asignación contigua de espacio de disco para siete archivos. (b) El estado del disco después de haber removido los archivos D y F.

Asignación de lista enlzada (linked list)

Otro método es mantener cada archivo como una lista enlazada de bloques de disco. La primera palabra de cada bloque se utiliza como apuntador al siguiente. El resto del bloque es para los datos.

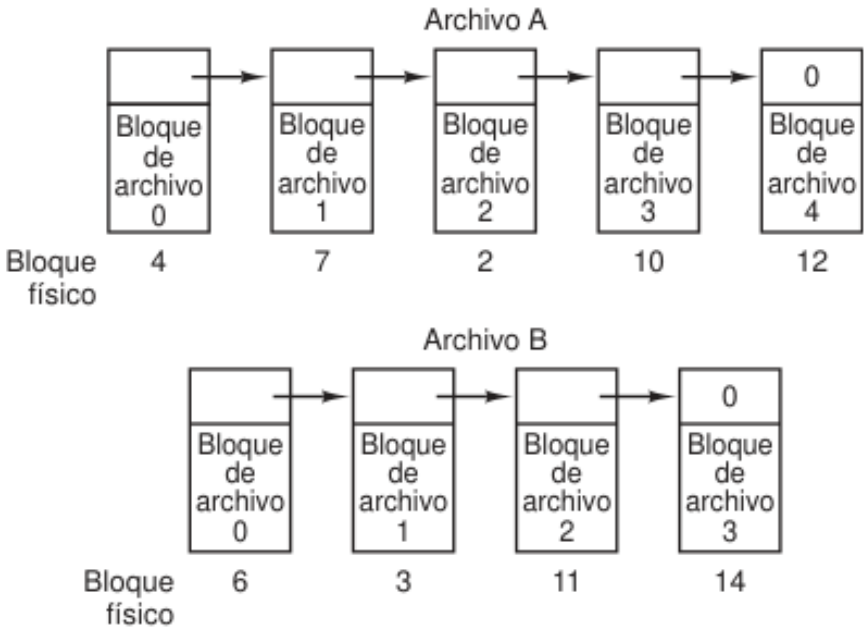

Figura 10: Imagen sacada de (Tanenbaum, 2009). Almacenamiento de archivos como lista enlazada.

Con este método no se pierde espacio debido a la fragmentación (salvo por la fragmentación

interna del último bloque), y para acceder al archivo sólo hace falta saber la ubicación en disco del primer bloque. Sin embargo, el acceso aleatorio es super lento y, al tener que usar una parte del espacio de cada bloque para referenciar al siguiente, perdemos espacio de almacenamiento del archivo.

Asignación de lista enlazada utilizando una tabla en memoria

Las dos desventajas de la asignación de lista enlzada se pueden eliminar si tomamos la palabra del puntero de cada bloque de disco y la ponemos en una tabla en memoria. En la Figura 11 vemos un archivo A que utiliza los bloques de disco 4, 7, 2, 10 y 12, en ese orden y el archivo B que utiliza los bloques de disco 6, 3, 11 y 14, en ese orden.

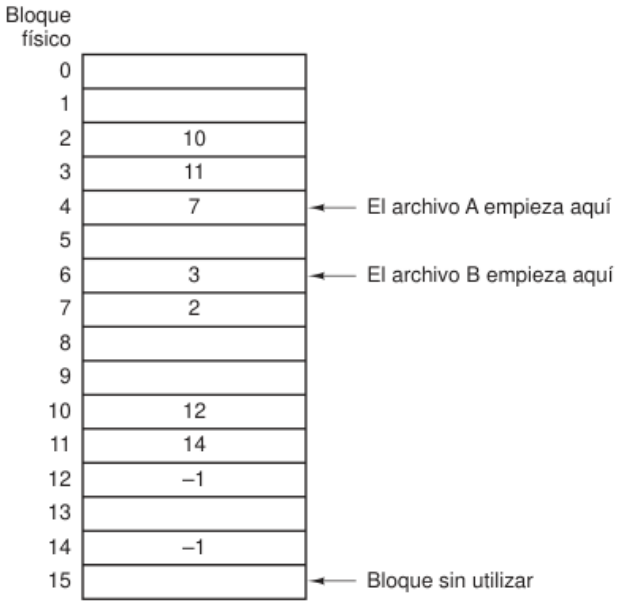

Figura 11: Imagen sacada de (Tanenbaum, 2009).

La principal desventaja de este método es que toda la tabla debe estar en memoria principal todo el tiempo para que funcione y esto es muy poco práctico.

i-nodos

El último método que vamos a ver es el de asociar con cada archivo una estructura de datos conocida como **i-nodo**(**nodo-indice**), que lista los atributos y las direcciones de disco de los bloques del archivo. La gran ventaja de esto es que el i-nodo necesita estar en memoria sólo cuando está abierto el archivo correspondiente. Si cada i-nodo ocoupa n bytes, y puede haber un máximo de k archivos abiertos a la vez, la memoria total ocupada por el arreglo que contiene los i-nodos para los archivos abiertos es kn bytes.

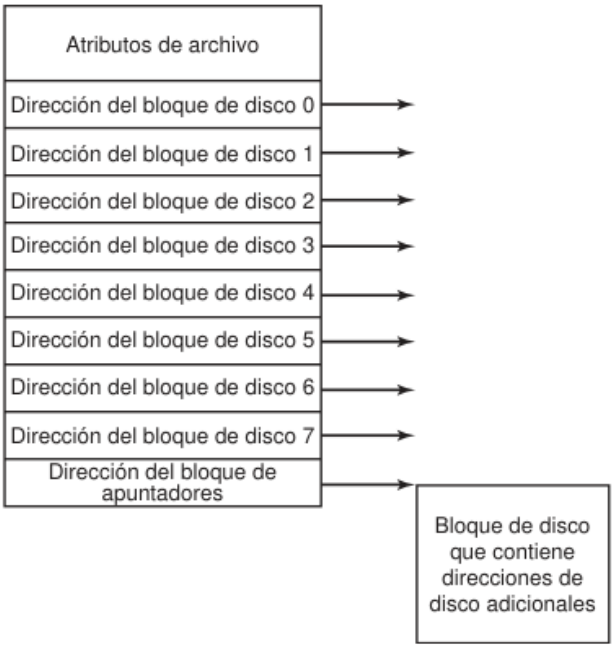

Figura 12: Imanen sacada de (Tanenbaum, 2009). Un i-nodo de ejemplo.

2.3.3. Implementación de directorios

Cuando vamos a guardar un arhivo dentro de un directorio, tenemos dos posibilidades principales para almacenar los atributos de los mimsmo. Podemos almacenarlos directamente en la entrada del directorio, como se muestra en la Figura 13 (a). Otra forma es almacenar los atributos en los i-nodos, en vez de hacerlo en la entrada de los directorios, como se muestra en la Figura 7.

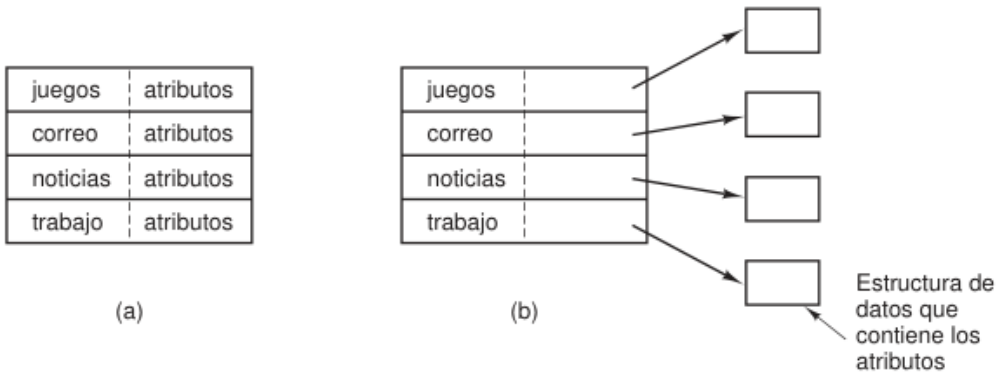

Figura 13: Imagen sacada de (Tanenbaum, 2009). (a) Un directorio simple en Windows. (b) Un directorio en UNIX.

Estas formas de almacenar los archivos son válidas pero traen problemas si queremos almacenar nombres de archivos mas largos, ya que la logitud de cada bloque es fija, por lo que los archivos que tengan nombres cortos estarían desperdiciando todo el espacio que les sobra.

Una alternativa es que la estructura este dada en tres partes: una primera parte para indicar la longitud de la entrada, una segunda parte para almacenar los atributos, y la tercer parte continene el nombre del archivo, siendo este tan largo o corto como quiera el usuario. Esto se ve en la Figura 14 (a). Sin embargo esto trae problemas de fragmentación, aunque ahora es mas factible compactar el directorio ya que se encuentra en memoria principal.

Otra manera de manejar los nombres de longitud variable es hacer que las entradas de directorio sean de longitud fija y mantener los nombres de los archivos juntos en un heap al final del directorio, como se muestra en la Figura 14 (b).

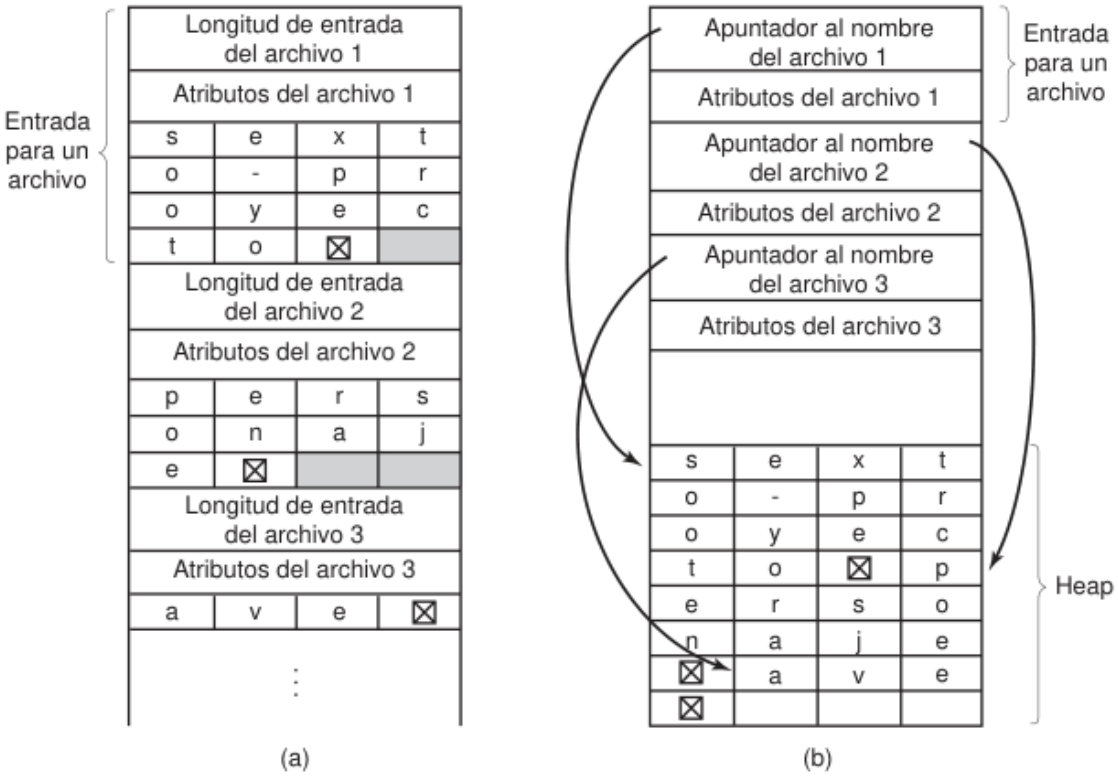

Figura 14: Imagen sacada de (Tanenbaum, 2009). Dos maneras de manejar nombres de archivos de longitud variable. (a) En línea. (b) En un heap.

2.3.4. Archivos compartidos

Cuando hay varios usuarios trabajando en conjunto en un proyecto, a menudo necesitan compartir archivos. De esta manera, resulta conveniente que un mismo archivo aparezca en dos directorios distintos (uno para cada usuario), como se vé en la Figura 15. La conexión entre el directorio *B* y el archivo compartido del directorio *C* se conoce como **vínculo**. El sistema de archivos en sí ahora pasa a ser un **Grafo acíclico dirigido** (*Directed Acyclic Graph*, **DAG**) en vez de un árbol.

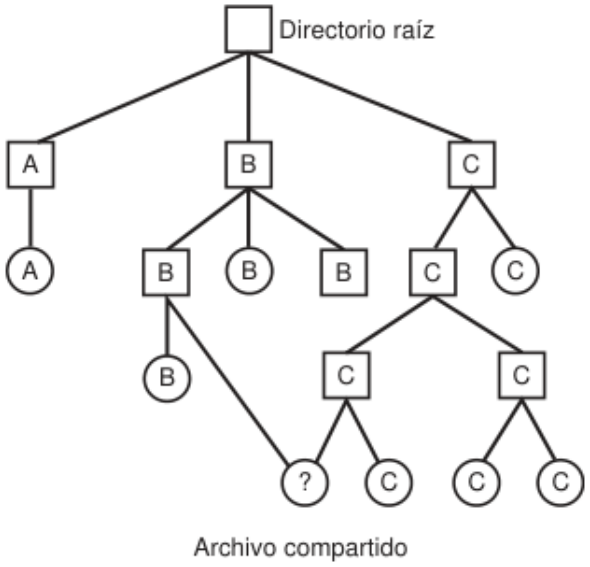

Figura 15: Imagen sacada de (Tanenbaum, 2009). Sistema que contiene un archivo compartido.

También existe lo que se conoce como **link simbólico** o **vínculo simbólico** en el que, al hacer el vínculo del directorio *B* a uno de los archivos de *C*, el nuevo archivo (el que se crea como link en *B*) contiene sólo el nombre de la ruta del archivo al cual está vinculado.

Ambos métodos tiene sus desventajas. En el caso de los *hard-links* (los vínculos normales), supongamos el caso de la figura 15. Si en algún momento el directorio *C* trata de eliminar el archivo que está vinculado con *B*, el archivo va a seguir existiendo pero *B* será el único usuario con acceso al mismo, aunque el propietario sigue siendo *C* ya que no se puede eliminar el i-nodo del archivo en el que se encuentran los atributos (tampoco se puede modificar quién es el propietario). Entonces todos los cambios que haga *B* sobre el archivo se le van a acreditar a *C*, muy probablemente quitándole espacio de almacenamiento por un archivo al que no puede acceder.

Con los vínculos simbólicos no sucede esto ya que, como el archivo en *B* guarda la ruta del archivo en *C*, si *C* elimina el archivo, la ruta que hay en *B* ya no es válida así que esta vez el archivo sí desaparece sin problemas. Sin embargo, la desventaja en este caso está en que cuando *B* quiere modificar el archivo se requiere de un mayor nivel de procesamiento ya que hay que leer la ruta del archivo a la que apunta, verificar si existe, ir hasta la ruta, acceder a su espacio en disco y recién ahí realizar los cambios requeridos.

Existen muchos otros sistemas de archivos entre los que se encuentran: **Sistemas de archivos estructurados por registro**, **Sistemas de archivos por bitácora**, **Sistemas de archivos virtuales**. Estos últimos son sistemas de archivos que utiliza UNIX principalmente para poder integrar varios sistemas de archivos distintos en uno solo.

La mayoría de los sistemas UNIX utilizan el concepto de **VFS** (*Virtual File System*) en el que todas las llamadas al sistema relacionadas con archivos se dirigen al sistema de archivos virtual para su procesamiento inicial. EStas llamadas, que provienen de procesos de usuarios, son las llamadas de POSIX estándar tales como `open`, `read`, `write`, etc.

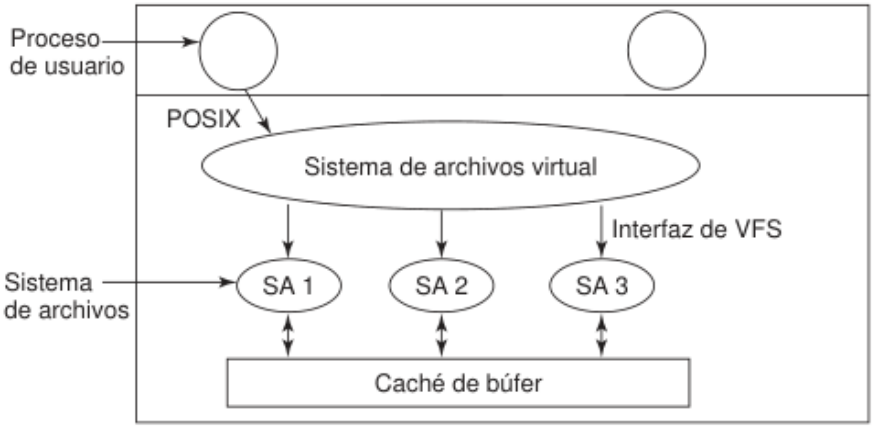

Figura 16: Imanen sacada de (Tanenbaum, 2009). Posición del sistema de archivos virtual.

2.4. Administracion y optimizacion de sistemas de archivos

Siempre la idea de crear un sistema de archivos es hacerlo de manera eficiente. Por lo general los archivos se almacenan en disco, asi que la administracion del espacio en disco es una cuestion importante. Los puntos a tener en cuenta son:

- **Tamano de bloque.** Lo mas eficiente es almacenar los archivos en bloques de tamano fijo para evitar la fragmentacion del espacio. Es importante decidir el tamano que debe tener cada bloque ya que si se usan bloques de tamano muy grande, archivos de tamano pequeno desperdician el espacio. Por el contrario, si usamos bloques demasiado pequenos se pierde tiempo al leer los archivos.

- **Registro de bloques libres.** El siguiente paso es determinar c'omo llevar el regisro de los bloques libres. Se puede hacer de dos formas tambien: con una lista enlazada de bloques de disco, donde cada bloque contiene tantos numeros de bloques de disco libres como pueda. Con un bloque de 1 KB y un numero de disco de 32 bits, cada bloque en la lista de bloques libres contiene los numeros de 255 bloques libres (para un disco de 500 GB, se requieren aproximadamente 1.9 millones de bloques). La otra opcion es crear un mapa de bits, donde cada bit referencia a un bloque de disco. Si el bit es 0, el bloque esta libre, si el bit es 1 quiere decir que el bloque no esta vacio (para un disco de 500 GB ahora se necesitan aproximadamente 60.000 bloques).

- **Cuotas de disco.** Para evitar que los usuarios ocupen demasiado espacio en disco, los sistemas operativos multiusuario proporcionan un mecanismo para imponer las cuotas de disco.

3. Unidad 3 - Administracion de Procesos

3.1. ¿Qué es un proceso?

Podemos definir un proceso como un programa en ejecución. Los procesos se caracterizan por tener una serie de elementos, incluyendo los siguientes:

- **Identificador (PID)**. Un identificador único asociado a cada proceso, para distingirlo del resto.
- **Estado**. Si está en ejecución, detenido, o finalizado.
- **Prioridad**. Nivel de prioridad relativo al resto de procesos.
- **Contador de programa**. La dirección de la siguiente instrucción del programa que se ejecutará.
- **Punteros a memoria**. Punteros al código de programa y los datos asociados a dicho proceso, además de cualquier bloque de memoria compartido con otros procesos.
- **Datos de contexto**. Los datos que están presentes en los registros del procesador cuando es proceso está corriendo.
- **Información de estado de E/S**. Las peticiones de E/S pendientes, dispositivos de E/S asignados a dicho proceso, una lista de ficheros en uso por el mismo, etc.
- **Información de auditoría**. Puede incluir la cantidad de tiempo de procesador y de tiempo de reloj utilizados, así como los límites de tiempo, registros contables, etc.

Toda esta información se almacena en una estructura de datos, que se suele llamar **Bloque de Control de Procesos (PCB)**, que es creada y gestionada por el sistema operativo. El PCB es una herramienta clave que permite al sistema operativo dar soporte a múltiples procesos y proporcionar multiprogramación.

“Cuando un proceso se interrumpe, los valores actuales del contador de programa y los registros de procesador (datos de contexto) se guardan en los campos correspondientes del PCB y el estado del proceso se cambia a cualquier otro valor. El sistema operativo ahora es libre de poner otro proceso en estado de ejecución, para después recuperar el PC y los datos de contexto para poder volver a ejecutar el procesos donde como si nada hubiese pasado.”

3.2. Estados de los procesos

Dijimos que un proceso es un programa en ejecución. Sabemos que el que se encarga de ejecutar los programas es el procesador de la computadora. Este va ejecutando las instrucciones en una secuencia dada por el cambio del PC (Program Counter) [Recordar materia arquitectura]. Lo que vamos a ver ahora es que a lo largo del tiempo el PC puede apuntar al código de diferentes programas que son parte de diferentes procesos.

En la Figura 17 podemos ver un ejemplo de la disposición de 3 procesos en la memoria principal. Ademas, hay un programa adicional llamado **activador** que es el que se encarga de decirle al procesador cuál proceso debe ejecutar. Ademas, en la Figura 18, podemos ver las **trazas** de los procesos, que no es mas que las direcciones en memoria de cada una de las instrucciones que tiene cada proceso.

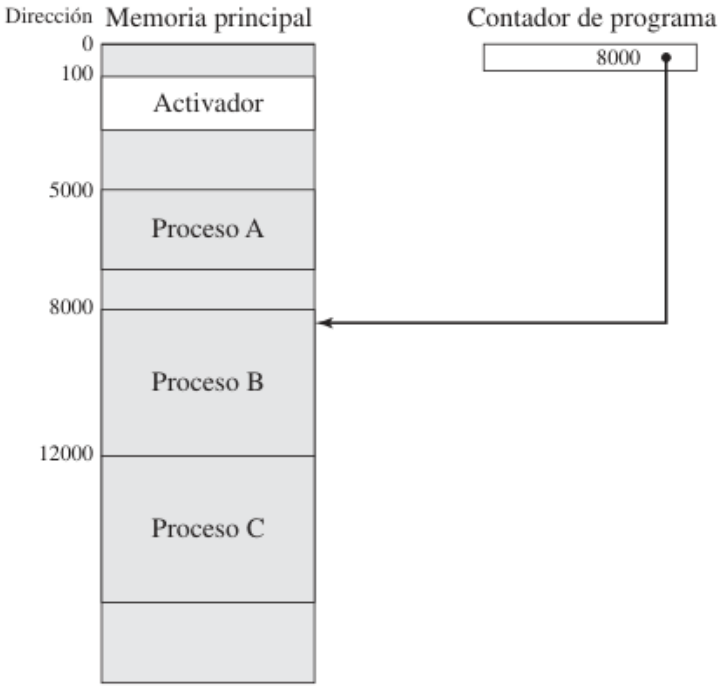

Figura 17: Imagen sacada de (Stallings, 2005). Ejemplo de la ejecución de procesos.

(a) Traza del Proceso A	(b) Traza del Proceso B	(c) Traza del Proceso C
5000	8000	12000
5001	8001	12001
5002	8002	12002
5003	8003	12003
5004		12004
5005		12005
5006		12006
5007		12007
5008		12008
5009		12009
5010		12010
5011		12011

5000 = Dirección de comienzo del programa del Proceso A.
8000 = Dirección de comienzo del programa del Proceso B.
12000 = Dirección de comienzo del programa del Proceso C.

Figura 18: Imagen sacada de (Stallings, 2005). Trazas de los procesos A, B y C.

Ahora, en la siguiente figura podemos ver cuáles son los valores que va tomando el PC durante los

primeros 52 ciclos de ejecución. En este ejemplo estamos suponiendo que el sistema operativo sólo deja que un proceso se ejecute durante seis cíclos de instrucción, para después interrumpirse y darle paso a la ejecución de otro proceso. De esta manera se evita que un solo proceso monopolice el uso del tiempo del procesador. Vemos además que entre la ejecución de los procesos A, B y C, se ejecuta el programa que habíamos llamado activador, ya que es este el que se encarga de modificar el valor del PC una vez pasados los 6 cíclos de reloj para que apunte a otro proceso.

1	5000	
2	5001	
3	5002	
4	5003	
5	5004	
6	5005	
		Temporización
7	100	
8	101	
9	102	
10	103	
11	104	
12	105	
13	8000	
14	8001	
15	8002	
16	8003	
		Petición de E/S
17	100	
18	101	
19	102	
20	103	
21	104	
22	105	
23	12000	
24	12001	
25	12002	
26	12003	
27	12004	
28	12005	
		Temporización
29	100	
30	101	
31	102	
32	103	
33	104	
34	105	
35	5006	
36	5007	
37	5008	
38	5009	
39	5010	
40	5011	
		Temporización
41	100	
42	101	
43	102	
44	103	
45	104	
46	105	
47	12006	
48	12007	
49	12008	
50	12009	
51	12010	
52	12011	
		Temporización

100 = Dirección de comienzo del programa activador.
Las zonas sombreadas indican la ejecución del proceso de activación;
la primera y la tercera columna cuentan ciclos de instrucciones;
la segunda y la cuarta columna las direcciones de las instrucciones que se ejecutan

Figura 19: Imagen sacada de (Stallings, 2005). Ejemplo de ejecución de tres procesos.

3.2.1. Un modelo de procesos de dos estados

Para que el sistema operativo pueda controlar la ejecución de procesos como acabamos de ver, tiene que saber sí o sí en qué estado se encuentra cada proceso. Necesita saber si el proceso A se está ejecutando para poder decidir si debe pararlo o no, y necesita saber lo mismo de los procesos B y C para poder decidir si debe ejecutarlos o no.

El modelo mas simple posible para controlar la ejecución de procesos es el modelo de dos estados. En este modelo un procesos puede tener solo uno de dos estados: Ejecutando o No Ejecutando. De este análisis super sencillo ya podemos deducir que en el sistema operativo, los procesos van a tener que representarse de alguna manera que indiquen si se están ejecutando o no. Los que no se estén ejecutando podrían estar en alguna especie de cola, esperando a que llegue su turno de ejecución (ver Figura 20).

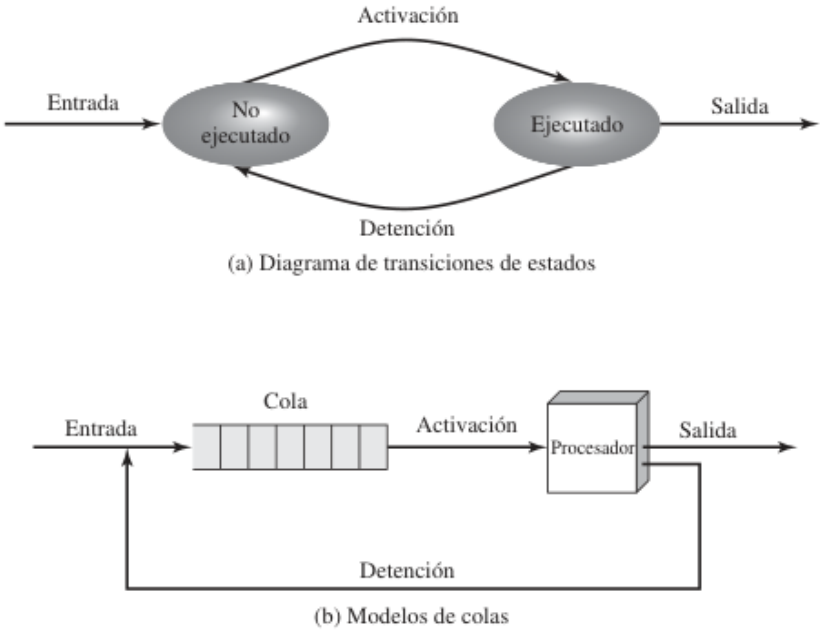

Figura 20: Imagen sacada de (Stallings, 2005). Modelo de procesos de dos estados.

Cuando los procesos son creados, el sistema operativo debe encargarse de generar la estrucutra de datos que se usa para manejar el proceso (PCB) y reservar el espacio de direcciones en memoria principal.

Las Figuras 21 y 22 describen las razones por las cuales un proceso puede crearse o terminarse.

Nuevo proceso de lotes	El sistema operativo dispone de un flujo de control de lotes de trabajos, habitualmente una cinta un disco. Cuando el sistema operativo está listo para procesar un nuevo trabajo, leerá la siguiente secuencia de mandatos de control de trabajos.
Sesión interactiva	Un usuario desde un terminal entra en el sistema.
Creado por el sistema operativo para proporcionar un servicio	El sistema operativo puede crear un proceso para realizar una función en representación de un programa de usuario, sin que el usuario tenga que esperar (por ejemplo, un proceso para controlar la impresión).
Creado por un proceso existente	Por motivos de modularidad o para explotar el paralelismo, un programa de usuario puede ordenar la creación de un número de procesos.

Figura 21: Imagen sacada de (Stallings, 2005). Razones para la creación de un proceso.

Finalización normal	El proceso ejecuta una llamada al sistema operativo para indicar que ha completado su ejecución
Límite de tiempo excedido	El proceso ha ejecutado más tiempo del especificado en un límite máximo. Existen varias posibilidades para medir dicho tiempo. Estas incluyen el tiempo total utilizado, el tiempo utilizado únicamente en ejecución, y, en el caso de procesos interactivos, la cantidad de tiempo desde que el usuario realizó la última entrada.
Memoria no disponible	El proceso requiere más memoria de la que el sistema puede proporcionar.
Violaciones de frontera	El proceso trata de acceder a una posición de memoria a la cual no tiene acceso permitido.
Error de protección	El proceso trata de usar un recurso, por ejemplo un fichero, al que no tiene permitido acceder, o trata de utilizarlo de una forma no apropia, por ejemplo, escribiendo en un fichero de sólo lectura.
Error aritmético	El proceso trata de realizar una operación de cálculo no permitida, tal como una división por 0, o trata de almacenar números mayores de los que la representación hardware puede codificar.
Límite de tiempo	El proceso ha esperado más tiempo que el especificado en un valor máximo para que se cumpla un determinado evento.
Fallo de E/S	Se ha producido un error durante una operación de entrada o salida, por ejemplo la imposibilidad de encontrar un fichero, fallo en la lectura o escritura después de un límite máximo de intentos (cuando, por ejemplo, se encuentra un área defectuosa en una cinta), o una operación inválida (la lectura de una impresora en línea).
Instrucción no válida	El proceso intenta ejecutar una instrucción inexistente (habitualmente el resultado de un salto a un área de datos y el intento de ejecutar dichos datos).
Instrucción privilegiada	El proceso intenta utilizar una instrucción reservada al sistema operativo.
Uso inapropiado de datos	Una porción de datos es de tipo erróneo o no se encuentra inicializada.
Intervención del operador por el sistema operativo	Por alguna razón, el operador o el sistema operativo ha finalizado el proceso (por ejemplo, se ha dado una condición de interbloqueo).
Terminación del proceso padre	Cuando un proceso padre termina, el sistema operativo puede automáticamente finalizar todos los procesos hijos descendientes de dicho padre.
Solicitud del proceso padre	Un proceso padre habitualmente tiene autoridad para finalizar sus propios procesos descendientes.

Figura 22: Imagen sacada de (Stallings, 2005). Razones para la terminación de un proceso.

3.2.2. Modelo de procesos de cinco estados

Este modelo que describimos recién sirve para darnos una idea de cómo manejar la ejecución de programas, pero la verdad que no sirve en la realidad. Cuando tenemos solo dos estados de los procesos, y todos comparten una misma cola de ejecución, puede surgir el problema de que hayan procesos que están próximos a ser ejecutados pero que todavía no están listos porque necesitan que se complete alguna operación de E/S. Y van a haber programas que sí están listos para ejecutarse ya, pero como no es su turno en la cola tienen que esperar a que todos los de mas adelante se ejecuten primero.

Una forma de solucionar este tipo de problemas es agregar mas estados:

- **Ejecutando**. El procesos está actualmente en ejecución.
- **Listo**. Un proceso que está esperando su turno para ejecutar y que ya está listo para ejecutarse.
- **Bloqueado**. Un proceso que está esperando su turno para ejecutar pero que todavía no está listo ya que debe esperar que se cumpla un evento determinado o que se complete alguna operación de E/S.
- **Nuevo**. Un proceso que se acaba de crear. Este proceso ya tiene su PCB pero todavía no está cargado en la memoria principal.
- **Saliente**. Un proceso que ya no está mas en la lista de procesos ejecutables por cualquier razón.

Figura 23: Imanen sacada de (Stallings, 2005). Modelo de procesos de cinco estados.

En la Figura 23 podemos ver cómo sería la gestión de este modelo de procesos. Las posibles transiciones son las siguientes:

- **Null** → **Nuevo**. Se crea un nuevo proceso.

- **Nuevo** → **Listo**. Cuando el sistema operativo está listo, admite al proceso nuevo. La mayoría de sistemas operativos fijan un límite basado en el número de procesos existentes.
- **Listo** → **Ejecutando**. El sistema operativo selecciona uno de los procesos que se encuentre en estado Listo y lo ejecuta.
- **Ejecutando** → **Saliente**. El sistema operativo finaliza el proceso que estaba en ejecución.
- **Ejecutando** → **Listo**. Lo mas normal es que el procesos en ejecución alcanzó su cuota máxima de tiempo posible de ejecución.
- **Ejecutando** → **Bloqueado**. Generalmente porque el proceso solicita algo por lo que tiene que esperar.
- **Bloqueado** → **Listo**. Cuando sucede el evento por el cual el proceso estaba esperando, se mueve a Listo.
- **Listo** → **Saliente**. Puede ser que el proceso haya sido creado por un proceso padre, y este fue terminado, por lo que el proceso hijo que estaba listo para ejecutar, ahora es un proceso saliente.
- **Bloqueado** → **Saliente**. Lo mismo que el caso anterior.

Para este tipo de sistemas lo mas conveniente es usar un sistema de dos colas, en vez de una sola: una cola para los procesos Listos, y otra para los Bloqueados. Y mejor aún, se pueden usar varias colas para los procesos bloqueados: una para cada evento que está siendo esperado. De esta manera, cuando existen muchos procesos esperando distintos eventos, no hay que recorrer un cola extremadamente larga hasta encontrar al proceso que esperaba al evento que acaba de ocurrir, sino que solo ingresa a la cola de Listos el siguiente proceso de la cola de dicho evento. Quizás se puede ver un poco mejor en la Figura 24.

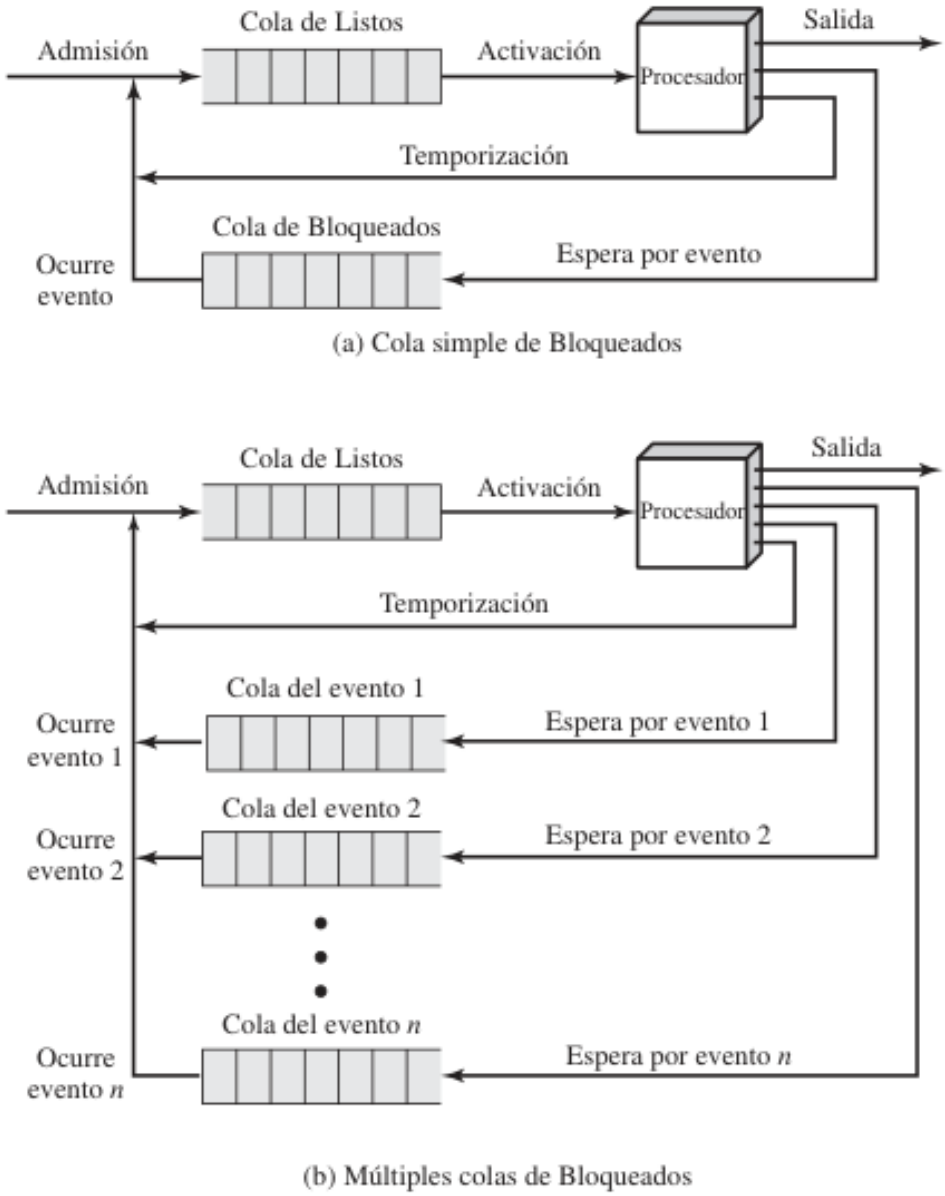

Figura 24: Imagen sacada de (Stallings, 2005). Modelo de colas para cada evento.

3.2.3. Procesos suspendidos

Los tres principales estados que describimos recién (Listo, Ejecutando, Bloqueado) sirven para modelar y diseñar un sistema operativo. Sin embargo, todavía se pueden agregar más estados. Recordemos que todo lo que estamos haciendo es intentar aprovechar al máximo el tiempo de procesamiento del procesador, ya que la velocidad de las operaciones de E/S son muy lentas en comparación. La diferencia entre estas velocidades es tal que, aún si usamos este método de cinco estados en un sistema multiprogramado, el procesador puede estar ocioso la mayor parte del tiempo.

Esto quiere decir que al procesador todavía le dá para procesar muchos mas procesos de los que somos capaces de gestionar en la memoria principal, y adquirir mayores cantidades de memoria no resulta factible. Lo que sí se puede hacer es usar una técnica que se conoce como **swapping**. Esta

técnica consiste en agarrar los estados que se encuentran en estado de Bloqueados, y quitarlos de la memoria principal para mandarlos al disco, dejándolos en un nuevo estado que se conoce como **Suspendido**. Esta técnica es de mucha utilidad ya que le permite a la computadora gestionar una mucho mayor cantidad de procesos al mismo tiempo, sin tener que usar tanta memoria principal.

Aunque, en realidad no es tan sencillo como agregar un solo estado llamado Suspendido, porque podemos tener un proceso que estaba Bloqueado en memoria principal y fue trasladado al disco, pero aún después de haber transcurrido un cierto tiempo puede que todavía no haya ocurrido el evento que lo pondría en estado de Listo. Por lo que si este proceso es llevado de vuelta a la memoria principal, seguiría estando bloqueado.

Teniendo en cuenta esto, lo que conviene hacer ahora es agregar no uno, sino dos nuevos estados además de Listo y Bloqueado:

- **Listo**. El proceso está en memoria principal disponible para su ejecución.
- **Bloqueado**. El proceso está en memoria principal y esperando un evento.
- **Bloqueado/Suspendido**. El proceso está en almacenamiento secundario (disco) Y esperando un evento.
- **Listo/Suspendido**. El proceso está en almacenamiento secundario pero está disponible para su ejecución tan pronto como sea cargado en memoria principal.

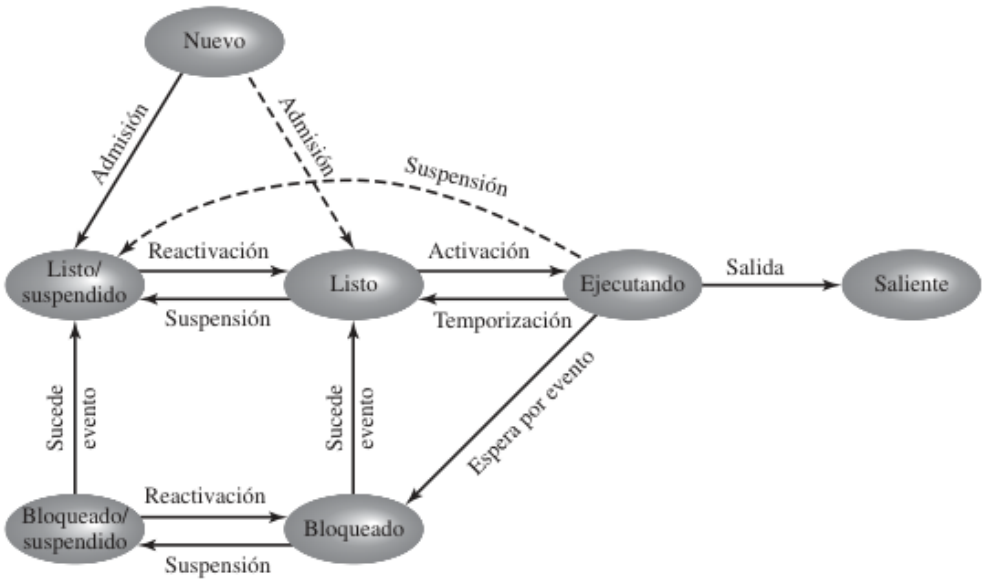

Figura 25: Imagen sacada de (Stallings, 2005). Procesos con dos estados suspendidos.

En la siguiente Figura se pueden ver las razones por las cuales un proceso podría suspenderse.

Swapping	El sistema operativo necesita liberar suficiente memoria principal para traer un proceso en estado Listo de ejecución.
Otras razones del sistema operativo	El sistema operativo puede suspender un proceso en segundo plano o de utilidad o un proceso que se sospecha puede causar algún problema.
Solicitud interactiva del usuario	Un usuario puede desear suspender la ejecución de un programa con motivo de su depuración o porque está utilizando un recurso.
Temporización	Un proceso puede ejecutarse periódicamente (por ejemplo, un proceso monitor de estadísticas sobre el sistema) y puede suspenderse mientras espera el siguiente intervalo de ejecución.
Solicitud del proceso padre	Un proceso padre puede querer suspender la ejecución de un descendiente para examinar o modificar dicho proceso suspendido, o para coordinar la actividad de varios procesos descendientes.

Figura 26: Imagen sacada de (Stallings, 2005). Razones para la suspensión de un proceso.

3.2.4. Estructuras de control de procesos

El sistema operativo se encarga de la gestión de procesos y recursos, y para esto construye y mantiene tablas de información sobre cada entidad que gestiona. Los detalles pueden variar de un sistema operativo a otro pero, fundamentalmente, todos los sistemas operativos mantienen información sobre: memoria, E/S, archivos y procesos.

Para el sistema operativo poder manejar los procesos, necesita conocer dónde están localizados los procesos, y cuáles son sus atributos.

Localización de los procesos

Para localizar un proceso su utiliza el concepto de **imagen del proceso**. Una imagen de un proceso es un conjunto de elementos que nos permiten identificarlo. En ella están el conjunto de programas a ejecutar, las pocisiones de memoria que necesita y usa, la pila que está asociada a él y su PCB.

Atributos de proceso

La información sobre todos los atributos del proceso se encuentra en el PCB. La tabla completa real con toda la información que contiene el PCB para que el sistema operativo use es muy larga y no viene al caso. Lo que sí tenemos que saber es que en el PCB se almacena información en tres categorías principales:

- **Identificación del proceso**. PID, identificadores de usuario, identificadores del proceso padre.
- **Información de estado del procesador**. Indica los contenidos de los registros del procesador.
- **Información de control de proceso**. Información de estado y planificación del proceso, privilegios del proceso, gestión de memoria, comunicación con otros procesos, etc.

El bloque de control de procesos (PCB) es la estructura de datos mas importante del sistema operativo. Cada bloque de control de proceso contiene toda la información sobre un proceso que necesita el sistema operativo y todo funciona en base a ello. Se puede decir que el conjunto de llos bloques de control de proceso definen el estado del sistema operativo.

3.3. Control de procesos

3.3.1. Modos de ejecución

La mayoría de los procesadores proporcionan dos modos de ejecución: **modo usuario**, que es el modo menor privilegiado; y **modo kernel**.

El motivo de la existencia de estos dos modos es la protección del sistema operativo y las tablas de información, de la interferencia con programas de usuario. Basicamente, es para que los usuarios no se puedan mandar cagadas y borren datos que son críticos para el funcionamiento de la computadora.

La forma en que el sistema operativo le hace saber al procesador en qué modo debe ejecutar cada programa es a través de un bit en la palabra de estado de programa (*Program State Word* PSW) que indica el modo de ejecución.

3.3.2. Creación de procesos

Cuando el sistema operativo decide crear un proceso, por cualquier razón, procede de la siguiente manera:

1. **Asignar un identificador de proceso único al proceso**. Se añade una nueva entrada a la tabla de procesos, que contiene una entrada por proceso.
2. **Reservar espacio para proceso**. Se debe reservar espacio suficiente para todos los elementos de la imagen del proceso.
3. **Inicialización del bloque de control de proceso**.
4. **Establecer enlaces apropiados**. Por ejemplo, el sistema operativo debe situar el proceso en la cola de Listos o Listos/Suspendidos.
5. **Creación o expansión de otras estructuras de datos**. Por ejemplo, el sistema operativo puede mantener un registro de auditoría por cada proceso que se puede utilizar posteriormente a efectos de facturación y/o análisis de rendiemiento del sistema.

3.3.3. Cambio de proceso

Un cambio de proceso es cuando se está ejecutando un proceso A, y este es interrumpido para que se pueda ejecutar el proceso B.

Los cambios de proceso puden ocurrir en cualquier instante en el que el sistema operativo obtiene el control sobre el proceso que está actualmente en ejecución. En general, puede deberse a tres causas:

- **System Call**. Puede ocurrir que el programa necesita abrir un archivo y realiza una llama al sistema, dándole control al sistema operativo para gestione la operación de abrir el archivo solicitado.
- **Interrupción**. Un ejemplo puede ser que el sistema operativo haya determinado que el proceso en ejecución ha excedido su tiempo de ejecución y lo interrumpe. En las interrupciones el control se transfiere inicialmente al manejador de interrupción, que realiza determinadas tareas internas y que posteriormente salta a una rutina del sistema operativo.
- **Traps**. Son interrupciones asociadas a una condición de error o algún acceso no permitido a un archivo.

3.3.4. Cambio de modo

Generalmente, el cambio de modo ocurre cuando sucede una interrupción de un proceso en ejecución. En este caso el sistema operativo actúa de la siguiente manera:

1. Coloca el contador de programa en la dirección de comienzo de la rutina del programa manejador de la interrupción.
2. Cambia de modo usuario a modo núcleo de forma que el código de tratamiento de la interrupción pueda incluir instrucciones privilegiadas.

3.3.5. Cambio de estado del proceso

Un cambio de estado del proceso y un cambio de modo son dos cosas distintas. Un cambio de estado de un proceso puede ocurrir sin que haya un cambio de modo y viceversa. Cuando un proceso va a cambiar de estado Ejecutando a cualquier otro estado, el sistema operativo tiene que realizar cambios importantes en su entorno. Los pasos que realiza son los siguientes:

1. Guardar el estado del procesador (PC y otros registros).
2. Actualizar el PCB del proceso que está actualmente en ejecución.
3. Mover el PCB a la cola correspondiente al nuevo estado.
4. Seleccionar el nuevo proceso a ejecutar.
5. Actualizar el PCB del proceso elegido pasándolo al estado Ejecutando.
6. Actualizar las estructuras de datos de gestión de memoria.
7. Si el proceso no es nuevo, restaurar el estado del procesador al que tenía el proceso antes de salir del estado Ejecutando la última vez.

3.4. Hilos

Acá viene un cambio re copado sobre lo que veníamos viendo. Hasta ahora dijimos que básicamente los procesos tenían dos características:

- **Propiedad de recursos**. Cada proceso cuenta con un espacio de direcciones de memoria para el manejo de la imagen del proceso; la imágen era un conjunto de programa, datos, pila y el PCB.
- **Planificación/Ejecución**. La ejecución de un proceso sigue una ruta de ejecución (traza) a través de uno o más progamas. Esta ejecución puede estar intercalada con ese u otros procesos. De esta manera, un proceso tiene un estado de ejecución y una prioridad de activación, y ésta es la entidad que se planifica y activa por el sistema operativo.

Lo que pasa ahora es que hoy en día, la mayoría de los sistemas operativos tratan a estas dos características como independientes una de otra. Para distinguirlas, a la característica que se activa (Planificación/Ejecución) se la denomina **hilo** o **thread**, mientras que a la parte de Propiedad de recursos se la denomina **proceso**.

3.4.1. Multihilo

Lo que nos permite hacer esta diferenciación es que, ahora, un sistema operativo puede ser capaz de dar soporte a *múltiples hilos* de ejecución dentro de *un solo proceso*. Hasta ahora veníamos estudiando los procesos que admitían un solo hilo a la vez, pero vamos a ver ahora que tener múltiples hilos por proceso es mucho mas eficiente.

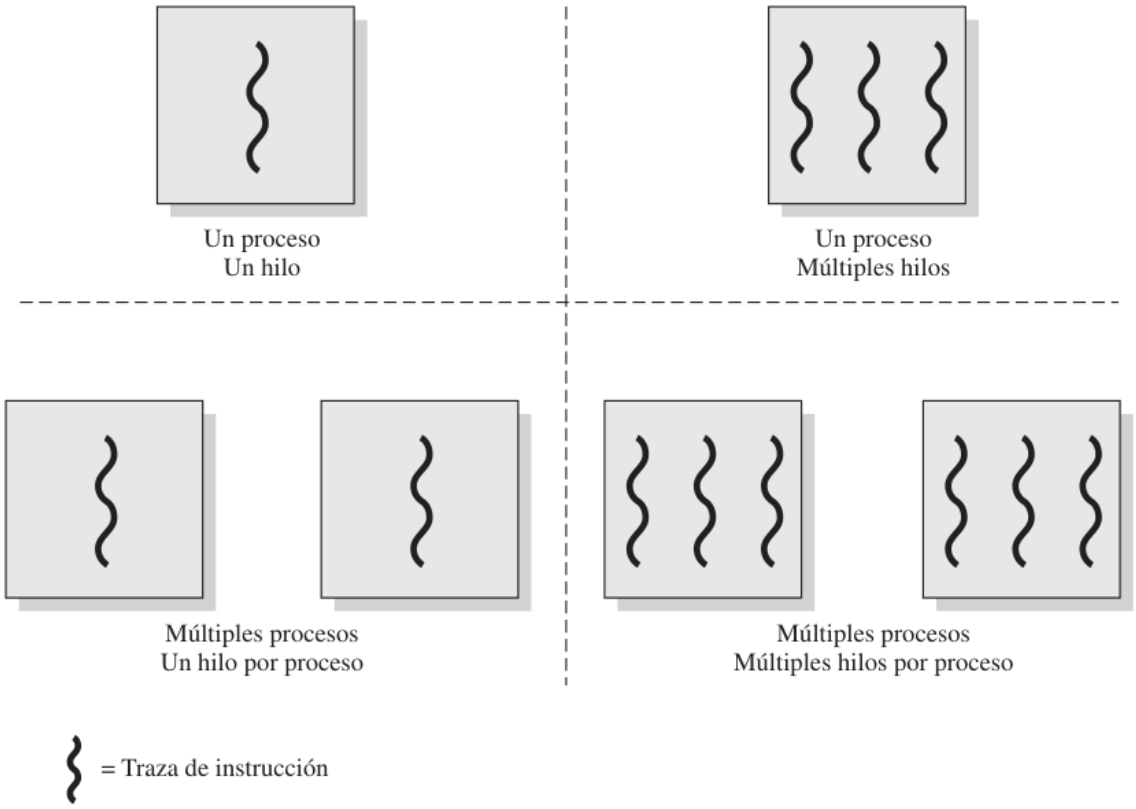

Figura 27: Imagen sacada de (Stallings, 2005). Hilos y procesos.

En un entorno multihilo, un proceso se define como la unidad de asignación de recursos y una unidad de protección. Un proceso tiene:

- Un espacio de direcciones de memoria que soporta la imagen del proceso.
- Acceso protegido a procesadores, otros procesos, archivos y recursos de E/S.

Dentro de un proceso puede haber uno o más hilos, cada uno con:

- Un estado de ejecución por hilo (Ejecutando, Listo, etc).
- Un contexto de hilo que se almacena cuando no está en ejecución. Serían como los datos necesarios para poder retomar el hilo donde había quedado cuando se vuelva a ejecutar.
- Una pila de ejecución.
- Por cada hilo, espacio de almacenamiento para variables locales.
- Acceso a la memoria y recursos de su proceso, compartido con todos los hilos de su mismo proceso.

En un entorno multihilo, sigue habiendo un único bloque de control del proceso y un espacio de direcciones de usuario asociado al proceso, pero ahora hay varias pilas separadas para cada hilo, así

como un bloque de control para cada hilo que contiene los valores de los registros, la prioridad, etc. Lo hilos de un mismo proceso comparten el estado y los recursos de ese proceso, reciben el mismo espacio de direcciones y tienen acceso a los mismo datos.

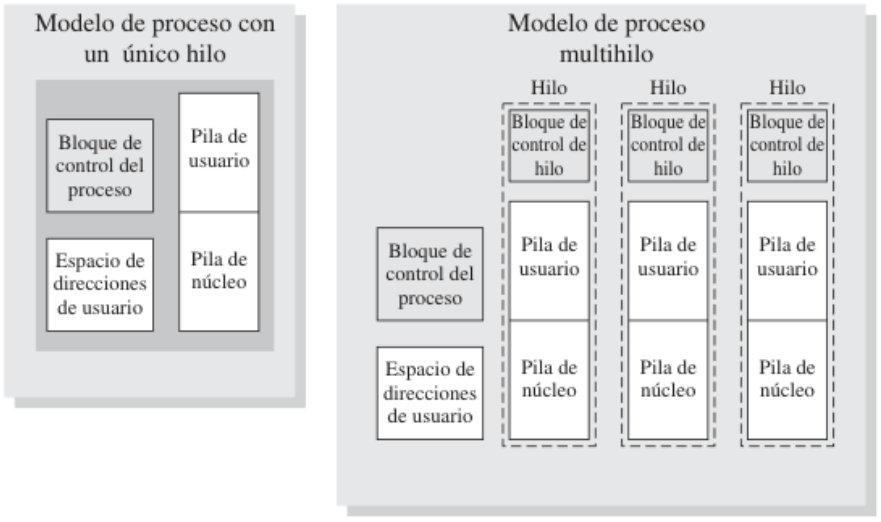

Figura 28: Imagen sacada de (Stallings, 2005). Modelos de proceso con único hilo y multihilo.

Los beneficios de los hilos son:

1. Lleva mucho menos tiempo (10 veces menos) crear un nuevo hilo en un proceso existente que crear un proceso totalmente nuevo.
2. Lleva menos tiempo finalizar un hilo que un proceso.
3. Lleva menos tiempo cambiar entre dos hilos dentro del mismo proceso.
4. los hilos mejoran la eficiencia de la comunicación entre diferentes programas que están ejecutando.

Ejemplos de usos de hilos:

- **Trabajo en primer plano y segundo plano**. Un hilo puede estar mostrando un menú en un excel mientras otro hilo está actualizando la hoja.
- **Procesamiento asíncrono**. Se puede crear un hilo que cada minuto escriba el *buffer* de memoria RAM de un archivo en el disco (guarda los cambios del archivo cada un minuto).
- **Velocidad de ejecución**. Un hilo puede estar esperando a la E/S mientras otro sigue procesando datos.

3.4.2. Funcionalidades de los hilos

Los hilos, al igual que los procesos, tienen estados de ejecución y se pueden sincronizar entre ellos. Los principales estados de los hilos son: Ejecutando, Listo y Bloqueado. No tiene sentido hablar de

hilos Suspendidos ya que eso es un concepto del proceso. Los hilos tienen que poder sincronizarse para que no interfieran entre ellos o corrompan la estructura de datos. No puede pasar que dos hilos intenten hacer cambios distintos al mismo tiempo en un mismo archivo.

En la siguiente figura vemos cómo mejora la eficiencia de un programa que realiza una llamada a procedimiento remoto (RPC) a dos máquinas diferentes utilizando un único procesador.

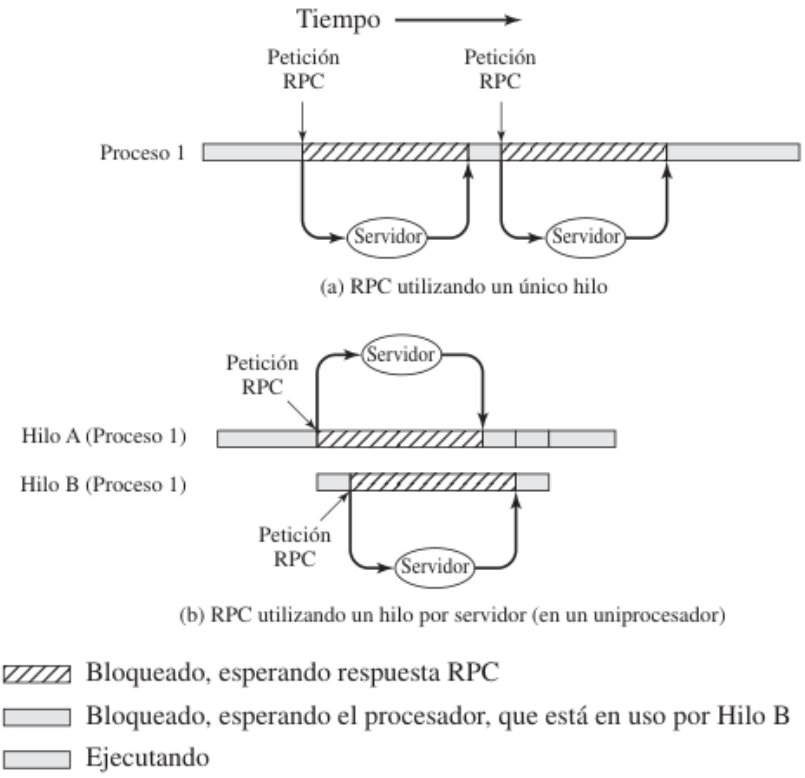

Figura 29: Imagen sacada de (Stallings, 2005). RPC utilizando hilos.

3.4.3. Hilos de nivel de usuario y de nivel de núcleo

Existen dos tipos de hilos: los de nivel de usuario (*user-level threads*, **ULT**) y los de nivel de núcleo (*kernel-level threads*, **KLT**).

Hilos de nivel de usuario

En un entorno ULT puro, la aplicación gestiona todo el trabajo de los hilos y el núcleo no es consciente de la existencia de los mismos. Cualquier aplicación puede programarse para ser multihilo a través del uso de una biblioteca de hilos. Esta biblioteca contiene código para la creación y destrucción de hilos, para paso de mensajes y datos entre los hilos, para planificar la ejecución de los hilos, y para guardar y resetaurar el contexto de los hilos. Esto se hace en el espacio de usuario y dentro de un solo proceso; el núcleo continúa planificando el proceso como una unidad y le asigna un único estado.

Ventajas del uso de ULT en vez de KLT:

- El cambio de hilo no requiere privilegios de modo núcleo.

- La planificación puede especificarse como parte de la aplicación. Una aplicación podría tener un algorítmo de planificación cíclico, mientras otra podría planificar basado en prioridades.
- Los ULT puede ejecutar en cualquier sistema operativo.

Desventajas:

- Cuando UTL realiza una system call que sea bloqueante, no solo se bloquea el hilo que hizo la llamada sino que todo el proceso entero se bloquea con él.
- Si el entorno es ULT puro, la aplicación no puede usar multiproceso ya que el núcleo le asigna un solo proceso a cada aplicación.

Hilos de nivel de núcleo

En un entorno KLT puro, el núcleo gestiona todo el trabajo de gestión de h ilos. No hay código de gestión de hilos en la aplicación, solamente una interfaz de programación de aplicación (API) para acceder a las utilidades de hilos del núcleo. Windows es un ejemplo de esto.

Este tipo de entornos resuelve las dos desventajas que aprecen en entornos ULT puros ya que el núcleo puede planificar simultáneamente múltiples hilos de un solo proceso en múltiples procesadores, y si se bloquea un hilo de un proceso, el núcleo puede planificar otro hilo del mismo proceso.

La principal desventaja del enfoque KLT es que la transferencia de control de un hilo a otro del mismo proceso requiere un cambio de modo al núcleo y esto significa un aumento muy significativo en el tiempo requierido para realizar dicha acción. Por ejemplo, crear un hilo a nivel usuario puede tardar al rededor de $34\mu S$, mientras que hacerlo a nivel de núcleo demora unos $948\mu S$.

Enfoques combinados

Existen sistemas operativos que usan ULT y KLT al mismo tiempo. Si está bien implementado, se pueden aprovechar las ventajas de los dos enfoques minimizando las desventajas.

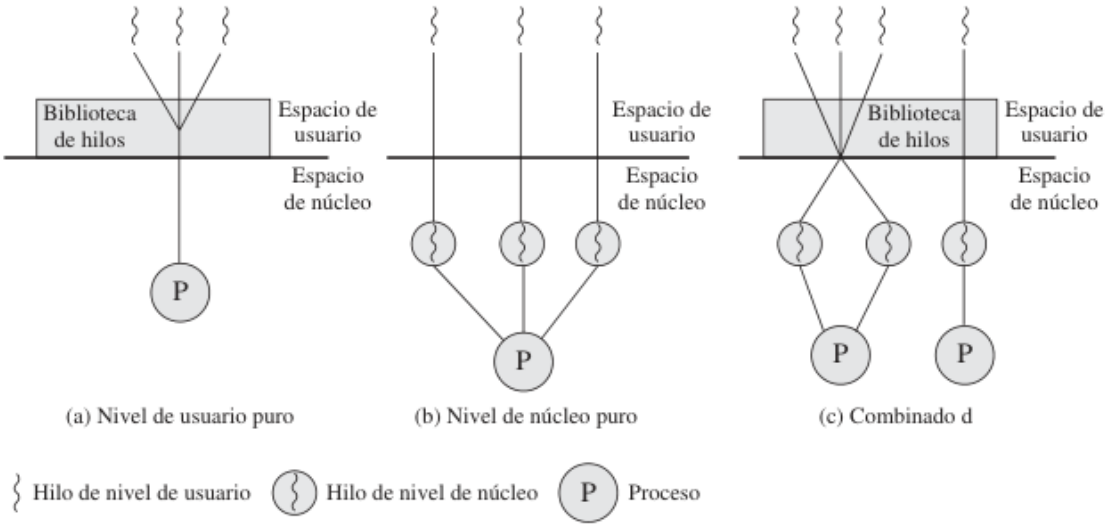

Figura 30: Imagen sacada de (Stallings, 2005). Hilos de nivel usuario y de nivel de núcleo.

3.5. Multiprocesamiento simétrico (SMP)

El SMP es un enfoque que se usa para proporcionar paralelismo entre procesos. La arquitectura de un sistema SMP está diseñada para que cada procesador pueda ejecutar el núcleo del sistema operativo, y normalemte cada procesador realiza su propia planificación del conjunto disponible de procesos e hilos. El núcleo a su vez puede construirse como múltiples procesos o hilos para permitir la ejecución de partes del mismo en paralelo.

En un sistema SMP existen múltiples procesadores, cada uno con acceso a una memoria principal compartida y dispositivos de E/S a través de algún mecanismo de interconexión. Los procesadores se puede comunicar entre sí a través de la memoria

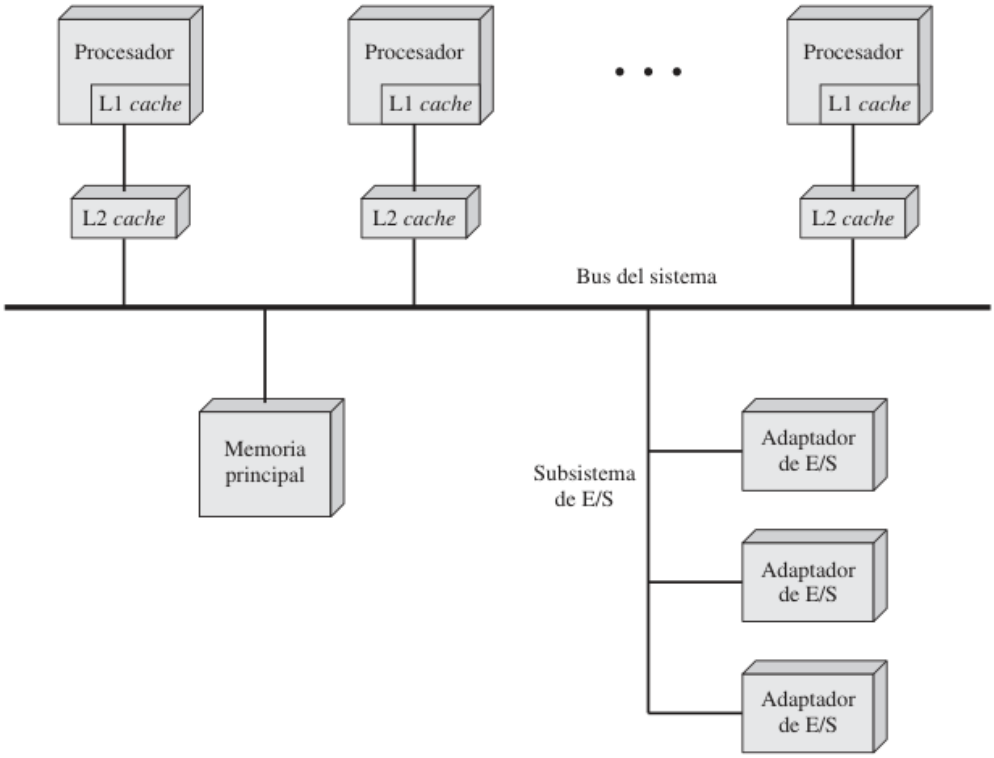

Figura 31: Imagen sacada de (Stallings, 2005). Organización de los SMP.

3.6. Micronúcleos

Los primeros sistemas operativos que se desarrollaron eran **sistemas monolíticos**, donde prácticamente desde cualquier procedimiento se podía llamar a cualquier otro. No había estructura en la implementación del código y esto se hizo insostenible a medida que los sistemas operativos fueron creciendo.

Luego aparecieron los sistemas operativos por capas, que estaban mas organizados pero igual seguían habiendo problemas. En estos, cada capa tenía distintas funcionalidades y se conectaba con las capas adyacentes. El problema es que si se realizaba un cambio en una capa, había que modificar las adyacentes también y los programas eran de millones de líneas de código.

Entonces apareció la idea de el **micronúcleo**. La idea era que solo las funciones absolutamente esenciales del sistema operativo estén en el núcleo, y el resto de servicios y aplicaciones se construyen sobre el micronúcleo y se ejecutan en modo usuario. De esta forma, el micronúcleo funciona como un intercambiador de mensajes: valida mensajes, los pasa entre los componentes, y concede el acceso al hardware.

3.6.1. Beneficios de una organización micronúcleo

- **Interfaz uniforme**. No hace falta diferenciar entre servicios a nivel de núcleo o usuario porque todos se proporcionan a través de paso de mensajes.
- **Extensibilidad**. Si se quiere añadir o modificar algún servicio, no hace falta rediseñar todo el núcleo, solo el módulo deseado.
- **Felxibilidad**. Se pueden añadir nuevas características o quitar existentes para una implementación mas pequeña y eficiente.
- **Portabilidad**. Es mas facil adaptar el núcleo a diferentes procesadores ya que todo el sistema operativo está en el micronúcleo.
- **Fiabilidad**. Un micronúcleo se puede verificar de forma rigurosa al no ser un código tan extenso.
- **Soporte de sistemas distribuidos**.
- **Soporte de sistemas operativos orientados a objetos**.

3.6.2. Rendimiento del micronúcleo

Referencias

Stallings, W. (2005). *Sistemas Operativos* (5.ª ed.). Pearson Educación, S.A.

Tanenbaum, A. S. (2009). *Sistemas operativos modernos.* (3.ª ed.). Pearson Educación.