**1.1**

**INTRODUCCIÓN SO**

**Concepto y definición de un sistema operativo**

**Sistema operativo**

Los sistemas operativos realizan dos funciones básicas que no están relacionadas:

* Proporcionar a los programadores de aplicaciones un conjunto abstracto de recursos

simples, en vez de los complejos conjuntos de hardware.

* Administrar estos recursos de hardware.
* El trabajo del sistema operativo es crear buenas abstracciones para después implementar y administrar los objetos abstractos entonces creados

El programa con el que interactúan los usuarios, generalmente llamado **shell** cuando está basado en texto y el **GUI** (Interfaz gráfica de usuario), cuando utiliza iconos, en realidad no forma parte del sistema operativo, aunque utiliza el sistema operativo.

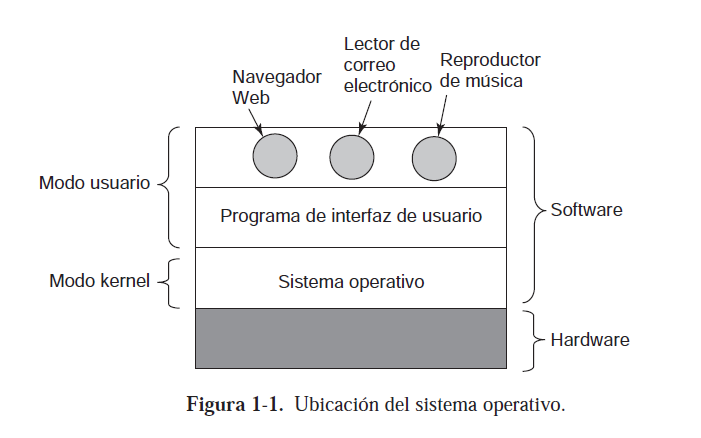
La mayoría de las computadoras tienen dos modos de operación: **modo kernel** y **modo usuario**.

**Modo Kernel**

El sistema operativo, la pieza de software más fundamental, se ejecuta en modo kernel (también llamado modo supervisor). En este modo tiene En este modo tiene acceso completo a todo el hardware y puede ejecutar cualquier instrucción que la máquina esté capaz de ejecutar.

**Modo usuario**

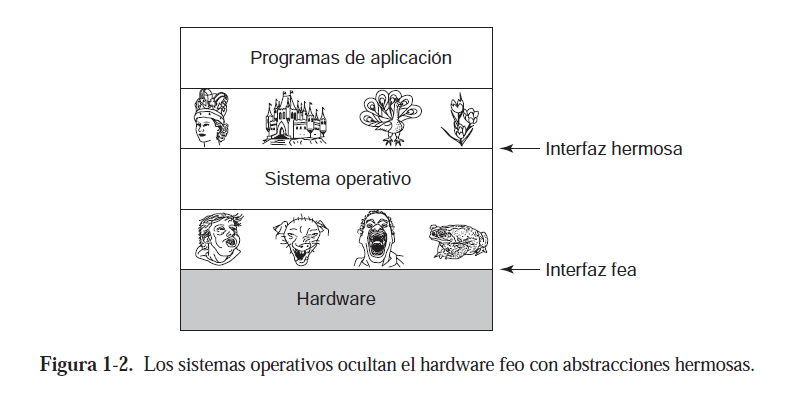
El cual sólo un subconjunto de las instrucciones de máquina es permitido. En particular, las instrucciones que afectan el control de la máquina o que se encargan de la E/S (entrada/salida) están prohibidas para los programas en modo usuario.



* **shell o GUI**, es el nivel más bajo del software en modo usuario y permite la ejecución de otros programas, como un navegador Web, lector de correo electrónico o reproductor de música. Estos programas también utilizan en forma intensiva el sistema operativo.

**El sistema operativo como una maquina extendida**

El trabajo del sistema operativo es crear buenas abstracciones para luego implementar y administrar los objetos creados. Una tarea principal del S.O. es ocultar el hardware y presentar a los programadores abstracciones agradables y simples.



**El sistema operativo como administrador de recursos**

el sistema operativo está presente para administrar todas las piezas

de un sistema complejo.

El trabajo del sistema operativo es proporcionar una asignación ordenada

y controlada de los procesadores, memorias y dispositivos de E/S, entre los diversos programas que compiten por estos recursos.

Los sistemas operativos modernos permiten la ejecución simultánea de varios programas.

Cuando una computadora (o red) tiene varios usuarios, la necesidad de administrar y proteger

la memoria, los dispositivos de E/S y otros recursos es cada vez mayor; de lo contrario, los usuarios

podrían interferir unos con otros.

**Esta visión del sistema operativo sostiene que su tarea principal es llevar un registro de qué**

**programa está utilizando qué recursos, de otorgar las peticiones de recursos, de contabilizar su uso**

**y de mediar las peticiones en conflicto provenientes de distintos programas y usuarios.**

La administración de recursos incluye el **multiplexaje** (compartir) de recursos en dos formas

distintas: en el tiempo y en el espacio.

* **En el tiempo**: los distintos programas o usuarios toman turnos para utilizarlo: uno de ellos obtiene acceso al recurso, después otro, y así en lo sucesivo.

La tarea de determinar cómo se multiplexa el recurso en el tiempo (quién sigue y durante cuánto tiempo) es responsabilidad del sistema operativo.

* **En el espacio**: En vez de que los clientes tomen turnos, cada uno obtiene una parte del recurso. (la memoria principal se divide entre varios

programas en ejecución)

**HISTORIA DE LOS SISTEMAS OPERATIVOS**

La primera computadora digital verdadera fue diseñada por el matemático inglés Charles Babbage

(de 1792 a 1871).

* **La primera generación (1945 a 1955): tubos al vacío.**

Primera computadora digital funcional, utilizaba 300 tubos de vacío (bulbos).

Toda la programación se realizaba exclusivamente en lenguaje máquina o, peor aún, creando circuitos eléctricos mediante la conexión de miles de cables a tableros de conexiones.

* **La segunda generación (1955 a 1965): transistores y sistemas de procesamiento por lotes.**

Por primera vez había una clara separación entre los diseñadores, constructores, operadores, programadores y el personal de mantenimiento.

Estas máquinas, ahora conocidas como **mainframes**, estaban encerradas en cuartos especiales

con aire acondicionado y grupos de operadores profesionales para manejarlas.

Para ejecutar un **trabajo** el programador primero escribía el programa en papel y después lo pasaba a tarjetas perforadas.

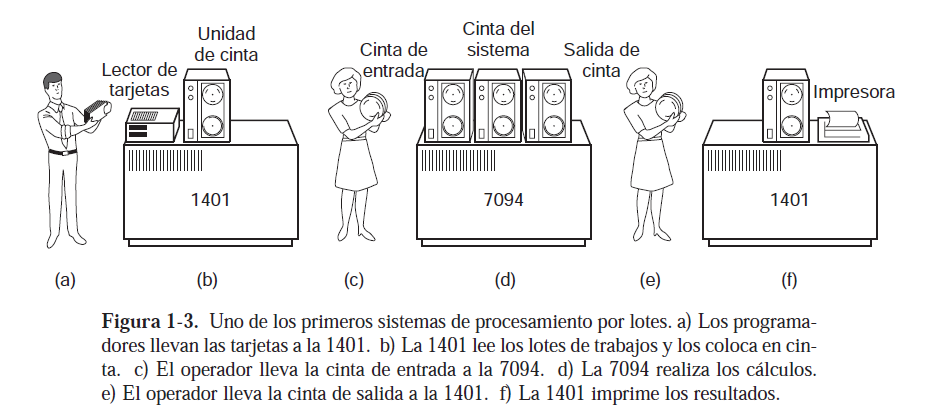
Cuando la computadora terminaba el trabajo que estaba ejecutando en un momento dado, un

operador iba a la impresora y arrancaba las hojas de resultados para llevarlas al cuarto de salida de

datos, para que el programador pudiera recogerlas posteriormente.

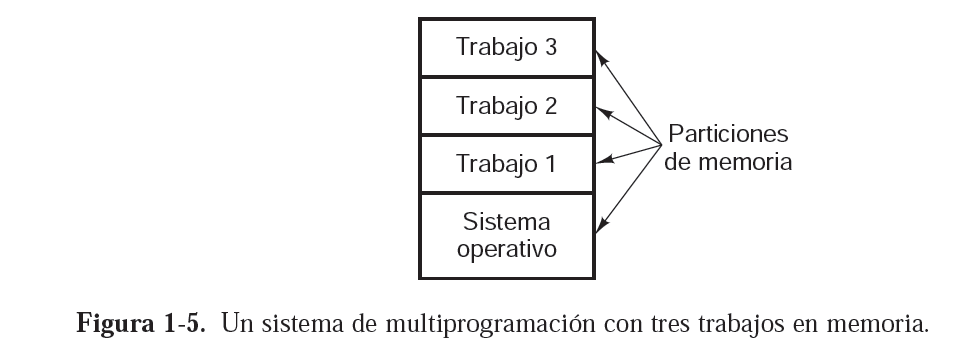
Dado el alto costo del equipo, no es sorprendente que las personas buscaran rápidamente formas

de reducir el tiempo desperdiciado. La solución que se adoptó en forma general fue el **sistema de procesamiento por lotes**.



* **La tercera generación (1965 a 1980): circuitos integrados y multiprogramación**

En la década de 1960, los fabricantes de computadoras tenían líneas de productos incompatibles para aplicaciones científicas y comerciales. IBM abordó este problema con la línea de computadoras System/360, que ofrecía máquinas compatibles con software de diferentes tamaños y capacidades. A pesar de la eficiencia inicial, la idea de una sola familia de computadoras resultó en un sistema operativo OS/360 enormemente complejo y propenso a errores. Aunque el OS/360 y sistemas operativos similares satisfacían a la mayoría de los clientes, su complejidad llevó a desafíos significativos. La introducción de técnicas clave, como la **multiprogramación**, ayudó a mejorar la eficiencia al permitir que la CPU estuviera ocupada casi todo el tiempo. A pesar de sus problemas, estos sistemas operativos de tercera generación dejaron razonablemente satisfechos a los usuarios y popularizaron importantes técnicas informáticas.



Durante la tercera generación de computadoras, los sistemas operativos introdujeron características como el spooling para la entrada/salida (E/S) y la capacidad de tiempo compartido (timesharing).

El **spooling** permitía cargar trabajos en disco mientras otros estaban en ejecución, eliminando la necesidad de máquinas como la 1401. Aunque estos sistemas eran eficientes para cálculos científicos y procesamiento masivo de datos comerciales, surgieron desafíos, y los programadores anhelaban tiempos de respuesta más rápidos. Esto condujo al desarrollo del **tiempo compartido**, donde múltiples usuarios podían interactuar con la CPU simultáneamente a través de terminales en línea.

El proyecto **MULTICS**, una "utilería para computadora", buscaba proporcionar poder de cómputo a cientos de usuarios simultáneos. Aunque no tuvo un éxito comercial significativo, influyó en sistemas operativos posteriores. El concepto de tiempo compartido se materializó con sistemas como CTSS y eventualmente llevó al desarrollo de **UNIX**, un sistema operativo modular y flexible que se hizo popular en ámbitos académicos y comerciales.

La evolución de UNIX y su influencia en sistemas operativos actuales, junto con la aparición de sistemas como Linux, reflejan la continuidad de las ideas y conceptos introducidos durante la tercera generación de computadoras.

* **La cuarta generación (1980 a la fecha) las computadoras personales**

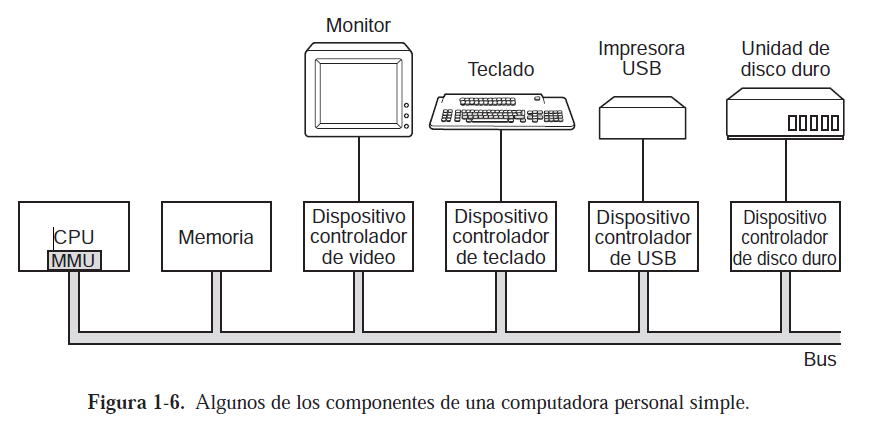
En resumen, con el surgimiento de los **circuitos LSI** (permite integrar en un solo chip una gran cantidad de componentes electrónicos, como transistores, resistencias, condensadores, iodos, entre otros), se inició la era de las computadoras personales.

El microprocesador 8080 de Intel fue crucial, y Gary Kildall desarrolló el sistema operativo CP/M para él.

La introducción de **GUI**, inspirada por Xerox PARC y adoptada por Apple, influyó en Microsoft, llevando al desarrollo de Windows. Aunque UNIX es fuerte en servidores, Linux se convierte en una alternativa popular en computadoras personales. Se destacan los sistemas operativos distribuidos y en red, y se menciona el desarrollo de FreeBSD y el uso de UNIX en estaciones de trabajo RISC.

**REVISIÓN DEL HARDWARE DE COMPUTADORA**

La CPU, la memoria y los dispositivos de E/S están conectados mediante un bus del sistema y se comunican entre sí a través de este bus.



* **Procesadores**

El “cerebro” de la computadora es la CPU, que obtiene las instrucciones de la memoria y las ejecuta.

El ciclo básico de toda CPU es obtener la primera instrucción de memoria, decodificarla para

determinar su tipo y operandos, ejecutarla y después obtener, decodificar y ejecutar las instrucciones

subsiguientes. El ciclo se repite hasta que el programa termina. De esta forma se ejecutan los

programas.

Los registros generales utilizados para contener variables y resultados temporales, la mayoría de las computadoras tienen varios registros especiales que están visibles para el programador:

1. **Contador de Programa (Program Counter):**

* **Función**: Contiene la dirección de memoria de la siguiente instrucción a ejecutar.
* **Actualización**: Se actualiza después de obtener la instrucción actual, apuntando así a la siguiente instrucción en secuencia.
* **Responsabilidad**: Controla el flujo de ejecución del programa, indicando qué instrucción se ejecutará a continuación.

1. **Apuntador de Pila (Stack Pointer):**

* **Función**: Apunta a la parte superior de la pila en la memoria.
* **Contenido de la Pila**: Almacena conjuntos de valores para cada procedimiento activo, incluyendo parámetros de entrada, variables locales y temporales.
* **Uso**: Facilita la gestión de la ejecución de procedimientos y la preservación de datos locales durante las llamadas a funciones.

1. **Palabra de Estado del Programa (PSW - Program Status Word):**

* **Función**: Contiene información sobre el estado del programa y bits de control.
* **Contenido**: Incluye bits de código de condición, prioridad de la CPU, modo (usuario o kernel) y otros bits de control.
* **Uso**: Desempeña un papel crucial en las llamadas al sistema y operaciones de Entrada/Salida (E/S). Los programas de usuario pueden leer el PSW y, en algunos campos, escribir en él.

En resumen, el Contador de Programa maneja la secuencia de ejecución de instrucciones, el Apuntador de Pila gestiona la memoria temporal para procedimientos, y el PSW proporciona información sobre el estado del programa y controla aspectos como el modo de ejecución y las operaciones del sistema. Cada uno cumple una función específica en el control y la gestión del flujo de un programa.

Cada vez que detiene un programa en ejecución, el sistema operativo debe guardar todos los registros para poder restaurarlos cuando el programa continúe su ejecución.

Muchas CPUs modernas cuentan con medios para ejecutar más de una instrucción al mismo tiempo.

* **canalización** (*pipeline*)

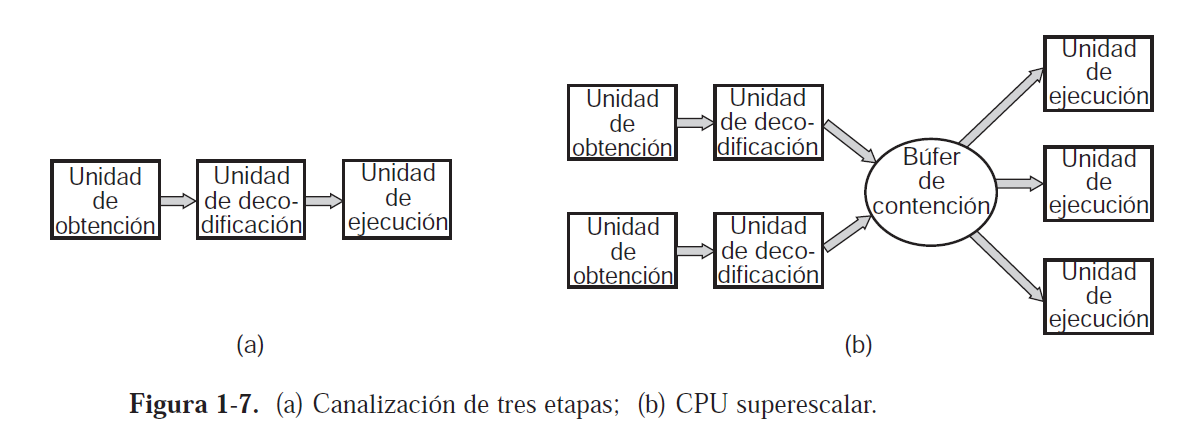
Una CPU podría tener unidades separadas de obtención, decodificación y ejecución, de manera que mientras se encuentra ejecutando la instrucción n, también podría estar decodificando la instrucción n +1 y obteniendo la instrucción n + 2.

Una vez que se ha obtenido una instrucción y se coloca en la canalización, se debe ejecutar aún si la instrucción anterior era una bifurcación condicional que se tomó.

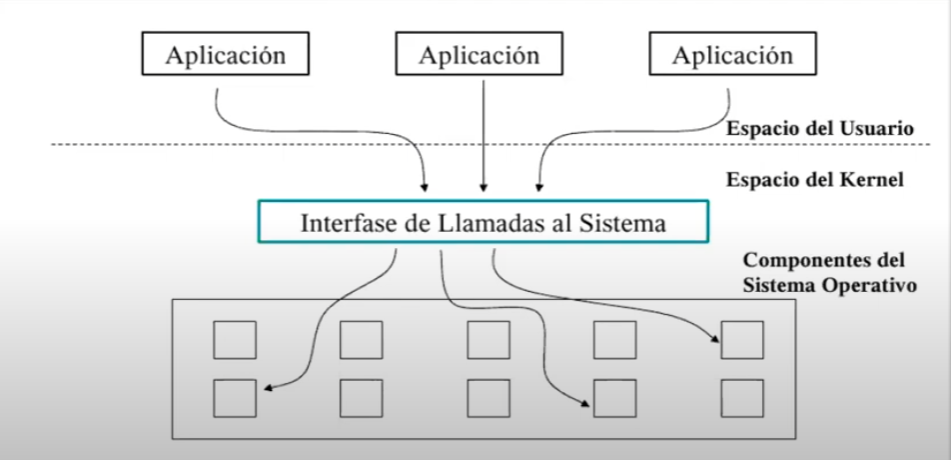
* **CPU** **superescalar**

Tipo de procesador que tiene la capacidad de ejecutar múltiples instrucciones en paralelo, superando la limitación de las CPUs escalares tradicionales que ejecutan instrucciones de una en una.

Dos o más instrucciones se obtienen a la vez, se decodifican y se vacían en un búfer de contención hasta que puedan ejecutarse. Tan pronto como una unidad de ejecución se encuentre libre, busca en el búfer de contención para ver si hay una instrucción que pueda manejar; de ser así, saca la instrucción del búfer y la ejecuta.



**Llamada al sistema (*system call*)**

****

Para obtener servicios del sistema operativo, un programa usuario debe lanzar una llamada al

Sistema la cual se atrapa en el kernel e invoca al sistema operativo. Cuando se ha completado

el trabajo, el control se devuelve al programa de usuario en la instrucción que va después de

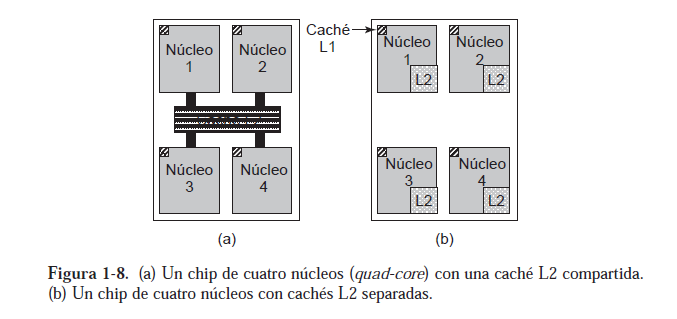
la llamada al sistema.

Chips con multihilamiento y multinúcleo

Las arquitecturas superescalares, con múltiples unidades funcionales, fueron una solución inicial. Sin embargo, la abundancia de transistores lleva a explorar opciones adicionales, como el multihilamiento o hiperhilamiento, que permite a la CPU contener el estado de dos hilos de ejecución y alternar entre ellos rápidamente.

El **multihilamiento** no ofrece un verdadero paralelismo, pero puede **mejorar la eficiencia** al cambiar entre hilos durante operaciones que requieren tiempo de espera. Esto afecta al sistema operativo, ya que cada hilo se presenta como una CPU separada.

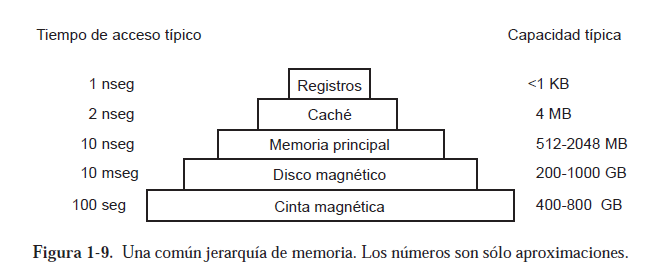
Más allá del multihilamiento, la evolución incluye chips de CPU con múltiples núcleos (cores) independientes en su interior. Los chips multinúcleo contienen varios minichips, cada uno con su propia CPU. La gestión eficiente de estos núcleos requiere sistemas operativos multiprocesador.

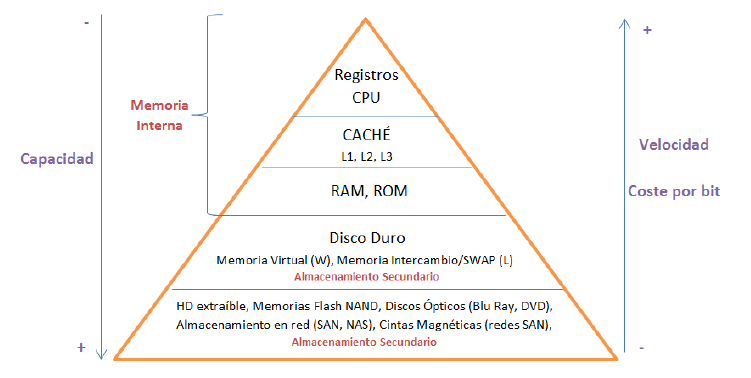


* **Memoria**

El sistema de memoria está construido como una jerarquía de capas.

Las capas superiores tienen mayor velocidad, menor capacidad y mayor costo por bit que las capas inferiores.





* **Registros**: Los programas deben administrar los registros (es decir, decidir qué deben guardar en ellos) por su cuenta, en el software. La capacidad de almacenamiento disponible en estos registros es generalmente de 32 \_ 32 bits en una CPU de 32 bits y de 64 \_ 64 bits en una CPU de 64 bits. Menos de 1 KB en ambos casos.
* **memoria caché**: el hardware la controla de manera parcial. La memoria principal se divide en líneas de caché, que por lo general son de 64 bytes, con direcciones de 0 a 63 en la línea de caché 0, direcciones de 64 a 127 en la línea de caché 1 y así sucesivamente. Cuando el programa necesita leer una palabra de memoria, el hardware de la caché comprueba si la línea que se requiere se encuentra en la caché. Si es así (a lo cual se le conoce como acierto de caché), la petición de la caché se cumple y no se envía una petición de memoria a través del bus hacia la memoria principal. Los aciertos de caché por lo general requieren un tiempo aproximado de dos ciclos de reloj. Los fallos de caché tienen que ir a memoria, con un castigo considerable de tiempo.

Por último, cuando se vuelve a escribir una línea de caché en la memoria principal (si se ha modificado desde la última vez que se puso en caché), la posición en memoria donde se debe volver a escribir se determina únicamente por la dirección en cuestión.

Las cachés son una idea tan útil que las CPUs modernas tienen dos de ellas. La **caché L1** o de

primer nivel está siempre dentro de la CPU, y por lo general alimenta las instrucciones decodificadas al motor de ejecución de la CPU.

Además, a menudo hay una segunda caché, conocida como **caché L2**, que contiene varios megabytes de palabras de memoria utilizadas recientemente. La diferencia entre las cachés L1 y L2 está en la velocidad. El acceso a la caché L1 se realiza sin ningún retraso, mientras que el acceso a la caché L2 requiere un retraso de uno o dos ciclos de reloj.

* **Memoria principal**: comúnmente conocida como RAM (Memoria de Acceso Aleatorio), es esencial en la jerarquía de memoria. Su tamaño ha aumentado significativamente, y todas las solicitudes de la CPU que no se pueden satisfacer desde la caché pasan a la memoria principal.

Además de la memoria principal, algunas computadoras tienen memoria no volátil, como la ROM (Memoria de sólo lectura), que es programada en la fábrica y no se puede modificar. También existen memorias no volátiles como EEPROM y memoria flash, que permiten la escritura y borrado de datos, utilizadas para almacenamiento en dispositivos portátiles.

La **memoria flash** se emplea comúnmente en dispositivos electrónicos portátiles y tiene velocidades intermedias entre la RAM y el disco. Aunque más duradera que la memoria en disco, puede desgastarse con el tiempo.

La **memoria CMOS**, aunque volátil, se utiliza para almacenar datos como fecha y hora actuales, así como parámetros de configuración. Alimentada por una pequeña batería, la memoria CMOS consume poca energía y puede durar varios años, pero su deterioro puede afectar la retención de datos y configuraciones.

* **Discos:** Los discos duros son la siguiente capa en la jerarquía de memoria después de la RAM. Aunque son más económicos y ofrecen más capacidad que la RAM, su acceso aleatorio es más lento debido a la naturaleza mecánica. Los discos constan de platos giratorios y un brazo mecánico que se mueve sobre ellos. La información se organiza en pistas y cilindros concéntricos.

El tiempo para acceder aleatoriamente a los datos en un disco implica movimientos mecánicos y puede variar entre 5 y 10 milisegundos. Los discos también desempeñan un papel en esquemas de memoria virtual, donde se utilizan para almacenar partes menos utilizadas de programas.

En **sistemas de multiprogramación**, la presencia de caché y la **unidad de administración de memoria** (MMU) puede afectar el rendimiento, especialmente durante cambios de contexto. Los programadores deben tener en cuenta estrategias para minimizar el impacto en el rendimiento al gestionar la caché y las operaciones de la MMU.

* **Cintas:** La cinta magnética representa la última capa en la jerarquía de memoria. Principalmente utilizada como respaldo para el almacenamiento en disco y para conjuntos de datos extensos, su acceso puede llevar varios minutos. Aunque es económica y removible, la cinta se utiliza en situaciones donde la velocidad no es crítica. La jerarquía de memoria sigue siendo fundamental, ofreciendo diversas capas con diferentes tiempos de acceso, capacidades y costos por bit.
* **Dispositivos de E/S**

Los dispositivos de E/S también interactúan mucho con el sistema operativo.

Los dispositivos de E/S generalmente constan de dos partes: un dispositivo controlador y el dispositivo en sí. El dispositivo controlador es un chip o conjunto de chips que controla físicamente el dispositivo.

La otra pieza es el dispositivo en sí. Los dispositivos tienen interfaces bastante simples, debido

a que no pueden hacer mucho y también para estandarizarlas.

El software que se comunica con un dispositivo controlador, que le proporciona comandos y acepta respuestas, se conoce como **driver** (controlador).

Para utilizar el driver, se tiene que colocar en el sistema operativo de manera que pueda ejecutarse en modo kernel.

Todo dispositivo controlador tiene un número pequeño de registros que sirven para comunicarse con él.

La colección de todos los registros del dispositivo forma el espacio de puertos de E/S.

En otras computadoras, los registros de dispositivo se colocan en un espacio de puertos de E/S especial, donde cada registro tiene una dirección de puerto. En estas máquinas hay instrucciones IN y OUT especiales disponibles en modo kernel que permiten a los drivers leer y escribir en los registros.

**Los métodos para realizar operaciones de entrada/salida (E/S) incluyen:**

1. **Espera Ocupada**:

* El programa emite una llamada al sistema.
* El driver inicia la E/S y sondea continuamente al dispositivo.
* La CPU queda ocupada hasta que el dispositivo termina.

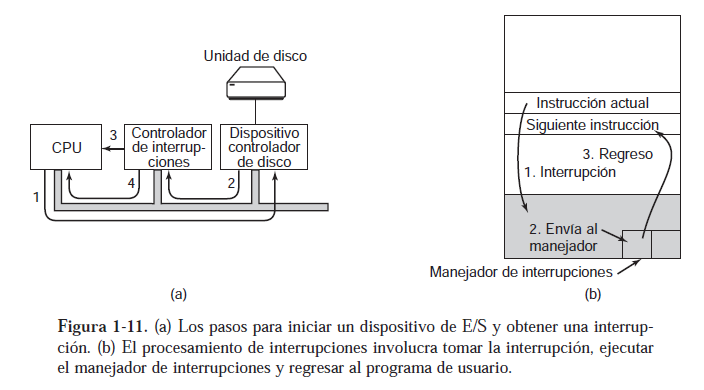
1. **Interrupciones:**

* El driver inicia el dispositivo y solicita una interrupción al finalizar.
* El sistema operativo puede bloquear el programa llamador y realizar otras tareas.
* Cuando el dispositivo termina, genera una interrupción.
* La CPU responde a la interrupción, ejecutando el manejador correspondiente del driver.
* Se utilizan vectores de interrupción para identificar el dispositivo y manejar la interrupción.

1. **Acceso Directo a Memoria (DMA):**

* Se utiliza un chip DMA para controlar la transferencia de datos entre memoria y dispositivo sin intervención constante de la CPU.
* La CPU configura el chip DMA y le indica los detalles de la transferencia.
* El chip DMA genera una interrupción al finalizar la operación.

Además, la CPU puede deshabilitar y habilitar las interrupciones según sea necesario para evitar interrupciones inoportunas. Cuando múltiples dispositivos terminan, el controlador de interrupciones prioriza y decide cuál permitir primero.



* **Buses**

El sistema tiene ocho buses (caché, local, memoria, PCI, SCSI, USB, IDE e ISA), cada uno con una velocidad de transferencia y función distintas. El sistema operativo debe estar al tanto de todos estos buses para su configuración y administración.

**LOS TIPOS DE SISTEMAS OPERATIVOS**

Los sistemas operativos se clasifican en diversas categorías según su aplicación y características. A continuación, se presenta un resumen de cada tipo:

1. **Sistemas operativos de Mainframe:**

- Diseñados para mainframes, grandes computadoras en centros de datos corporativos.

- Orientados a procesar múltiples trabajos simultáneamente, con énfasis en E/S.

- Ejemplos: OS/390, variantes de UNIX.

1. **Sistemas operativos de Servidores:**

- Ejecutan en servidores, dando servicio a múltiples usuarios a través de una red.

- Soportan servicios como impresión, archivos y Web.

- Ejemplos: Solaris, FreeBSD, Linux, Windows Server.

1. **Sistemas operativos de Multiprocesadores:**

- Utilizados en sistemas con múltiples CPUs conectadas.

- Variantes de sistemas operativos de servidores con características para la comunicación y consistencia.

- Ejemplos: Windows, Linux.

1. **Sistemas operativos de Computadoras Personales:**

- Soportan multiprogramación, proporcionando buen soporte para un solo usuario.

- Ampliamente utilizados para procesamiento de texto, hojas de cálculo e Internet.

- Ejemplos: Linux, Windows, macOS.

1. **Sistemas operativos de Computadoras de Bolsillo:**

- Diseñados para dispositivos de bolsillo y PDAs.

- Ejecutan en CPUs de 32 bits, con sistemas operativos como Symbian OS y Palm OS.

1. **Sistemas operativos Integrados:**

- Utilizados en dispositivos embebidos como hornos de microondas, televisores, etc.

- Ejecutan software confiable almacenado en ROM.

- Ejemplos: QNX, VxWorks.

1. **Sistemas operativos de Nodos Sensores:**

- Operan en redes de nodos sensores para aplicaciones como monitoreo y detección.

- Utilizan sistemas como TinyOS en nodos con recursos limitados.

1. **Sistemas operativos en Tiempo Real:**

- Enfocados en cumplir tiempos estrictos para acciones críticas.

- Diferencian entre sistemas en tiempo real duro (acciones críticas sin excepción) y suave (tolerancia ocasional de fallos en el tiempo).

- Ejemplos: e-Cos, sistemas para control de procesos industriales.

1. **Sistemas operativos de Tarjetas Inteligentes:**

- Operan en tarjetas del tamaño de una tarjeta de crédito con chips de CPU.

- Pueden ser propietarios o basados en Java, ejecutando applets de Java interpretados por la Máquina Virtual de Java (JVM).

- A menudo realizan funciones específicas como pagos electrónicos.

Estos sistemas operativos abarcan una amplia gama de aplicaciones, desde entornos empresariales hasta dispositivos embebidos y tarjetas inteligentes.

**CONCEPTOS DE LOS SISTEMAS OPERATIVOS**

* **Procesos**

Un **proceso** es en esencia un programa en ejecución. Cada proceso tiene asociado un **espacio de direcciones**, una lista de ubicaciones de memoria que va desde algún mínimo (generalmente 0) hasta cierto valor máximo, donde el proceso puede leer y escribir información.

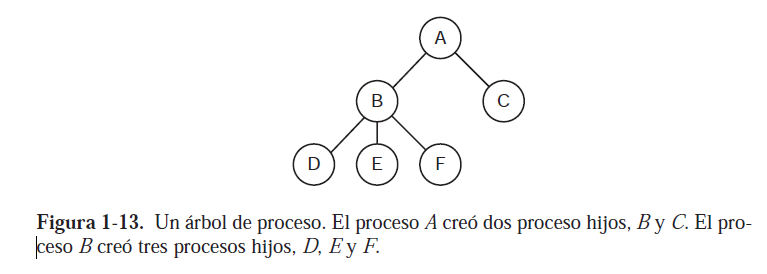
El espacio de direcciones contiene el programa ejecutable, los datos del programa y su pila.

En esencia, un proceso es un recipiente que guarda toda la información necesaria para ejecutar un programa.

Cuando un proceso se suspende en forma temporal, debe reiniciarse después exactamente en el mismo estado que tenía cuando se detuvo. Esto significa que toda la información acerca del proceso debe guardarse en forma explícita en alguna parte durante la suspensión.

Toda la información acerca de cada proceso se almacena en una tabla del sistema operativo, conocida como la **tabla de procesos**, la cual es un arreglo (o lista enlazada) de estructuras, una para cada proceso que se encuentre actualmente en existencia.

Si un proceso puede crear uno o más procesos aparte (conocidos como **procesos hijos**) y estos procesos a su vez pueden crear procesos hijos, llegamos rápidamente la estructura de árbol de procesos. A esta comunicación se le conoce como **comunicación entre procesos**.



En algunas ocasiones se tiene la necesidad de transmitir información a un proceso en ejecución que no está esperando esta información.

Cada persona autorizada para utilizar un sistema recibe una **UID (User Identification, Identificación de usuario)** que el administrador del sistema le asigna. Cada proceso iniciado tiene el **UID** de la persona que lo inició. **Un proceso hijo tiene el mismo UID que su padre**. Los usuarios pueden ser miembros de grupos, cada uno de los cuales tiene una GID (Group Identification, Identificación de grupo).

Una UID conocida como **superusuario** (superuser en UNIX) tiene poder especial y puede violar muchas de las reglas de protección.

* **Espacios de direcciones**

Por lo general, cada proceso tiene cierto conjunto de direcciones que puede utilizar, que generalmente van desde 0 hasta cierto valor máximo. En el caso más simple, la máxima cantidad de espacio de direcciones que tiene un proceso es menor que la memoria principal. De esta forma, un proceso puede llenar su espacio de direcciones y aún así habrá suficiente espacio en la memoria principal para contener todo lo necesario.

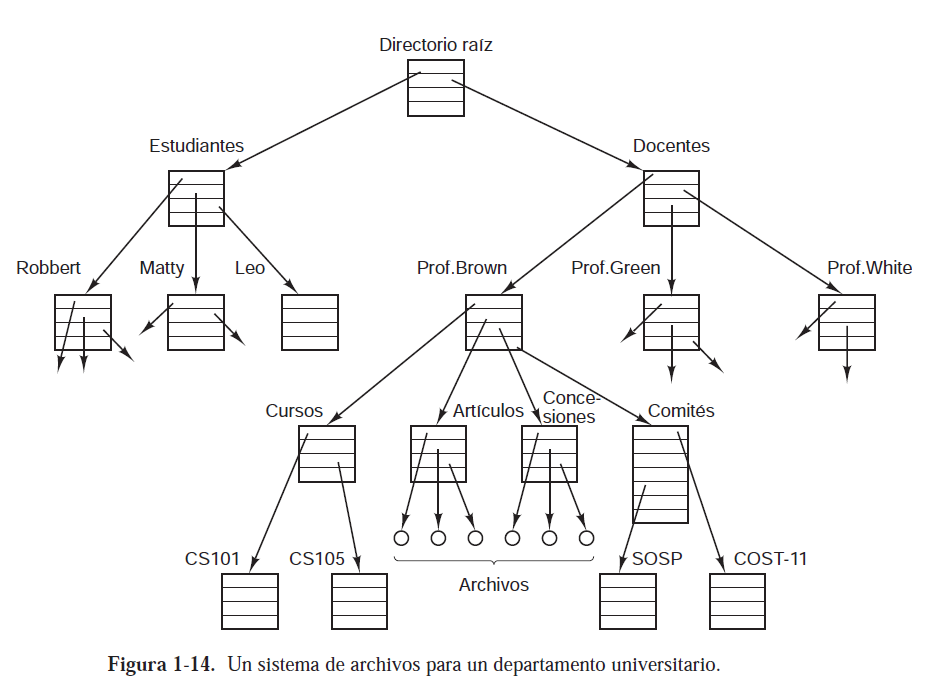
Muchas computadoras las direcciones son de 32 o 64 bits, con lo cual se obtiene un espacio de direcciones de o bytes, respectivamente. ¿Qué ocurre si un proceso tiene más espacio de direcciones que la memoria principal de la computadora, y desea usarlo todo?

Existe una técnica llamada memoria virtual, como se mencionó antes, en la cual el sistema operativo mantiene una parte del espacio de direcciones en memoria principal y otra parte en el disco, moviendo pedazos de un lugar a otro según sea necesario.

* **Archivos**

Se requieren las llamadas al sistema para crear los archivos, eliminarlos, leer y escribir en ellos.

Para proveer un lugar en donde se puedan mantener los archivos, la mayoría de los sistemas operativos tienen el concepto de un **directorio** como una manera de agrupar archivos.

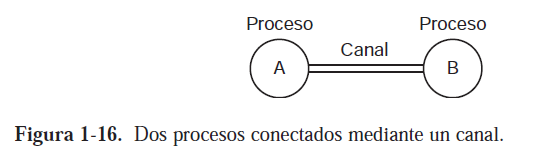


UNIX utiliza nombres de ruta absolutos para especificar la ubicación de archivos en su jerarquía de directorios. Estos nombres inician desde el directorio raíz y utilizan barras diagonales (/) como separadores. Los procesos tienen un directorio de trabajo actual que afecta la resolución de nombres de ruta relativos.

Antes de leer o escribir en un archivo, es necesario abrirlo y verificar los permisos. Si se permite el acceso, se devuelve un descriptor de archivo; de lo contrario, se devuelve un código de error.

En cuanto a los procesos, UNIX introduce la idea de **canales** (**pipes**), pseudoarchivos que conectan dos procesos para la comunicación de datos. La comunicación entre procesos en UNIX se asemeja a las operaciones comunes de lectura y escritura en archivos.

* **Entrada/salida**

****

Cada SO tiene un subsistema de E/S para administrar los dispositivos E/S. Parte del software de E/S es independiente de los dispositivos, se aplica a la mayoría por igual.

* **Protección**

Para administrar la seguridad del sistema el código de protección consta de 3 campos de 3 bits cada uno, uno para el propietario, otro para el grupo del mismo y otros para los demás usuarios. Cada campo tiene un bit para lectura, escritura y ejecución (bits 𝑟𝑤𝑥).

* **Shell**

Intérprete de comandos. No forma parte del SO, pero utiliza muchas características del mismo. Es la interfaz entre usuario y SO.

**LLAMADAS AL SISTEMA (system calls)**

Son la interfaz entre los programas y el usuario con el SO. **Se encuentra en el espacio kernel.** **Siempre se deben verificar los resultados de una llamada al sistema.**

Para hacer más entendible el mecanismo de llamadas al sistema, vamos a dar un vistazo rápido a la llamada al sistema read. Como dijimos antes, tiene tres parámetros: el primero especifica el archivo, el segundo apunta al búfer y el tercero proporciona el número de bytes a leer.

read(fd, buffer, nbytes)

Cualquier computadora con una sola CPU puede ejecutar sólo una instrucción a la vez.

Si un proceso está ejecutando un programa de usuario en modo usuario y necesita un servicio del sistema, como leer datos de un archivo, tiene que ejecutar una instrucción de trap para transferir el control al sistema operativo. Después, el sistema operativo averigua qué es lo que quiere el proceso llamador, para lo cual inspecciona los parámetros. Luego lleva a cabo la llamada al sistema y devuelve el control a la instrucción que va después de la llamada al sistema. En cierto sentido, realizar una llamada al sistema es como realizar un tipo especial de llamada a un procedimiento, sólo que las llamadas al sistema entran al kernel y las llamadas a procedimientos no.

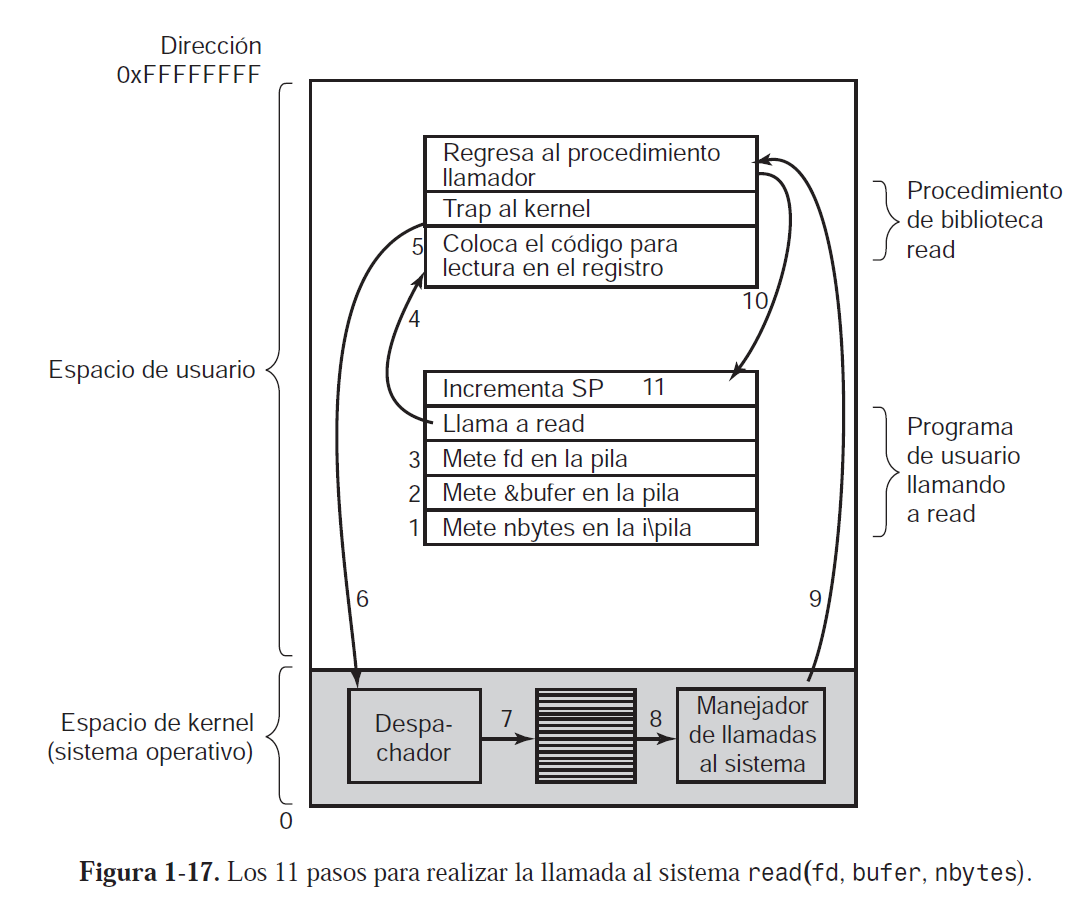
EJEMPLO

cuenta=read(fd, bufer, nbytes);

La llamada al sistema (y el procedimiento de biblioteca) devuelve el número de bytes que se leen en cuenta. Por lo general este valor es el mismo que nbytes pero puede ser más pequeño si, por ejemplo, se encuentra el fin de archivo al estar leyendo.

Si la llamada al sistema no se puede llevar a cabo, ya sea debido a un parámetro inválido o a un error del disco, cuenta se establece a \_1 y el número de error se coloca en una variable global llamada errno.

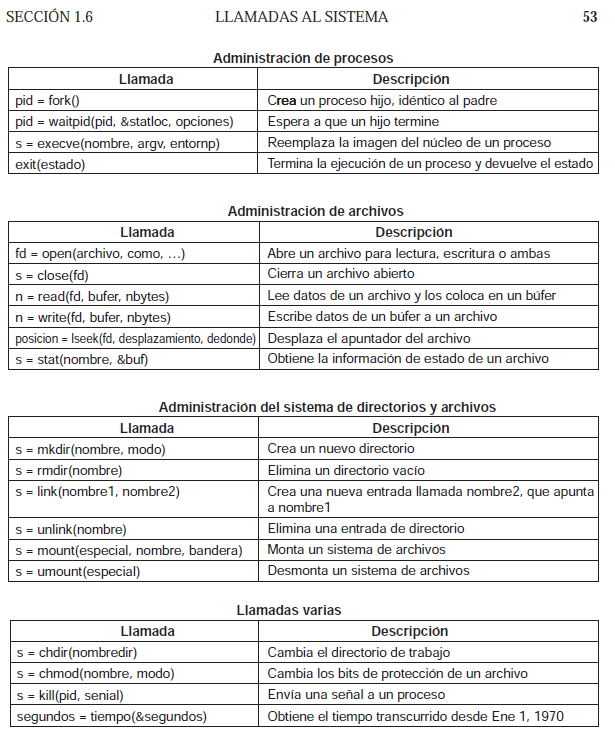
**Los programas siempre deben comprobar los resultados de una llamada al sistema para ver si ocurrió un error.**

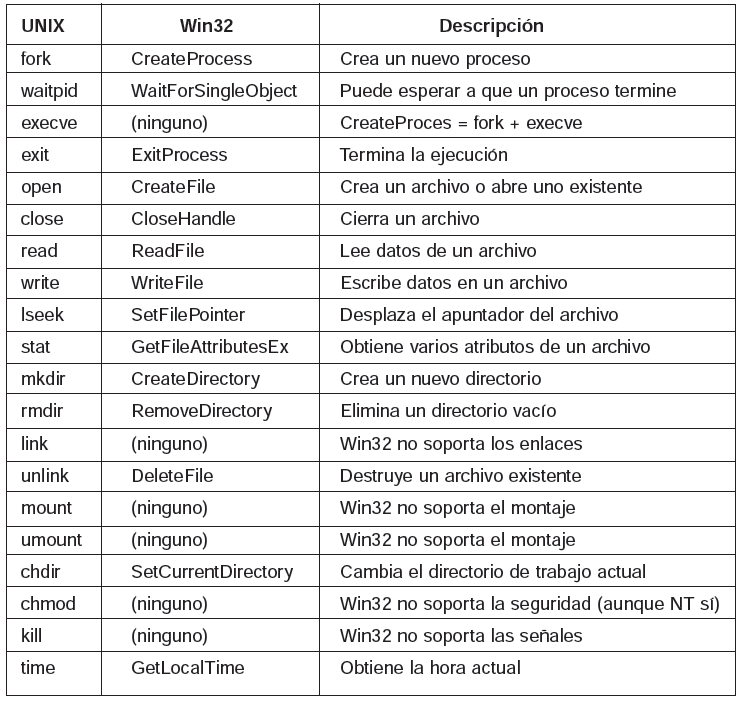


Pasos**:**

1. El programa llamador primero mete los parámetros en la pila.
2. Los parámetros primero y tercero se pasan por valor
3. El segundo parámetro se pasa por referencia, lo cual significa que se pasa la dirección del búfer (lo cual se indica mediante &), no el contenido del mismo.
4. Viene la llamada al procedimiento de biblioteca, es la instrucción de llamada a procedimiento normal utilizada para llamar a todos los procedimientos.
5. El procedimiento de biblioteca (probablemente escrito en lenguaje ensamblador) coloca por lo general el número de la llamada al sistema en un lugar en el que el sistema operativo lo espera, como en un registro
6. Ejecuta una instrucción TRAP para cambiar del modo usuario al modo kernel y empezar la ejecución en una dirección fija dentro del núcleo. TRAP en realidad es muy similar a la instrucción de llamada a procedimiento en el sentido en que la instrucción que le sigue se toma de una ubicación distante y la dirección de retorno se guarda en la pila para un uso posterior. En primer lugar, como efecto secundario, cambia a modo kernel. En segundo lugar, salta a una ubicación fija, hay un campo de 8 bits en la instrucción que proporciona el índice a una tabla en memoria que contiene direcciones de salto, o su equivalente.
7. El código de kernel que empieza después de la instrucción TRAP examina el número de llamada al sistema y después la pasa al manejador correspondiente de llamadas al sistema, por lo general a través de una tabla de apuntadores a manejadores de llamadas al sistema, indexados en base al número de llamada al sistema.
8. En ese momento se ejecuta el manejador de llamadas al sistema.
9. Una vez que el manejador ha terminado su trabajo, el control se puede regresar al procedimiento de biblioteca que está en espacio de usuario (La llamada al sistema puede bloquear al procedimiento llamador, evitando que continúe), en la instrucción que va después de la instrucción TRAP.
10. Luego este procedimiento regresa al programa de usuario en la forma usual en que regresan las llamadas a procedimientos.
11. Para terminar el trabajo, el programa de usuario tiene que limpiar la pila, como lo hace después de cualquier llamada a un procedimiento.

**Llamadas al sistema para la administración de procesos**

****

****

**FORK**

**fork** es la única manera de crear un nuevo proceso. Crea un duplicado exacto del proceso original, incluyendo todos los descriptores de archivos, registros y todo lo demás. Después de **fork**, el proceso original y la copia (el padre y el hijo) se van por caminos separados. Todas las variables tienen valores idénticos al momento de la llamada a **fork**, pero como los datos del padre se copian para crear al hijo, los posteriores cambios en uno de ellos no afectarán al otro.

La llamada a fork devuelve un valor, que es cero en el hijo e igual al identificador del proceso (PID) hijo en el padre. Mediante el uso del PID devuelto, los dos procesos pueden ver cuál es el proceso padre y cuál es el proceso hijo.

**ESTRUCTURA DE UN SISTEMA OPERATIVO**

**Sistemas monolíticos**

Se ejecuta como un solo programa en modo kernel. El sistema operativo se escribe como una colección de procedimientos, enlazados entre sí en un solo programa binario ejecutable extenso. Cada procedimiento en el sistema tiene la libertad de llamar a cualquier otro, si éste proporciona cierto cómputo útil que el primero necesita.

Al tener miles de procedimientos que se pueden llamar entre sí sin restricción, con frecuencia se produce un sistema poco manejable y difícil de comprender.

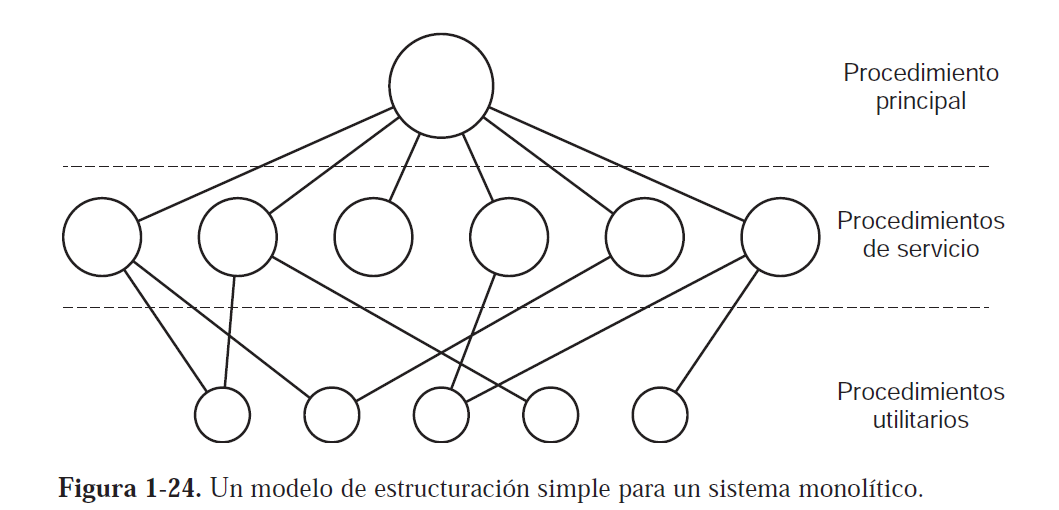
Para construir el programa objeto actual del sistema operativo cuando se utiliza este diseño, primero se compilan todos los procedimientos individuales y luego se vinculan en conjunto para formar un solo archivo ejecutable, usando el enlazador del sistema.

Estructura básica para el sistema operativo:

1. Un programa principal que invoca el procedimiento de servicio solicitado.

2. Un conjunto de procedimientos de servicio que llevan a cabo las llamadas al sistema.

3. Un conjunto de procedimientos utilitarios que ayudan a los procedimientos de servicio.



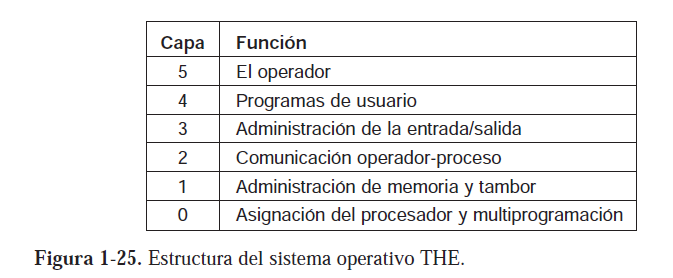
**Sistemas de capas**

El sistema constaba de seis capas, donde el nivel 0 se encargaba de la asignación del procesador y la multiprogramación básica de la CPU.

Las capas superiores se ocupaban de tareas específicas:

* **Capa 1**: Administración de memoria, asignando espacio en la memoria principal y un tambor para partes de procesos.
* **Capa 2**: Gestión de archivos y almacenamiento, permitiendo a los procesos interactuar con archivos sin preocuparse de su ubicación física.
* **Capa 3**: Administración de dispositivos de entrada/salida (E/S) y gestión de búferes para flujos de información.
* **Capa 4**: Ejecución de programas de usuario, liberando a los programas de preocupaciones relacionadas con procesos, memoria, consola y E/S.
* **Capa 5**: Operador del sistema, responsable de la supervisión general.

Este enfoque permitía a los procesos trabajar de manera independiente en niveles superiores, delegando las complejidades operativas a capas inferiores.



**Microkernels**

El diseño de capas en sistemas operativos permite separar el kernel y el usuario para mejorar la confiabilidad. Los microkernels, como MINIX 3, reducen el kernel a funciones mínimas, ejecutando la mayoría de los componentes como procesos de usuario para prevenir fallos catastróficos.

MINIX 3 tiene un microkernel de aproximadamente 3200 líneas de código C y 800 líneas de ensamblador. Los drivers y servidores se ejecutan en modo usuario, comunicándose mediante mensajes cortos. El sistema cuenta con restricciones de acceso para limitar el poder de cada proceso, garantizando que un error en un componente no afecte todo el sistema. Además, presenta un servidor de reencarnación para reemplazar automáticamente componentes defectuosos, asegurando una alta confiabilidad y autocorrección.

La idea de tener un kernel mínimo implica colocar solo el mecanismo en el kernel y dejar la directiva fuera. Por ejemplo, en la planificación de procesos, el algoritmo simple está en el kernel, mientras que la asignación de prioridades puede ser realizada por procesos en modo usuario, permitiendo desacoplar la directiva y el mecanismo para reducir el tamaño del kernel.

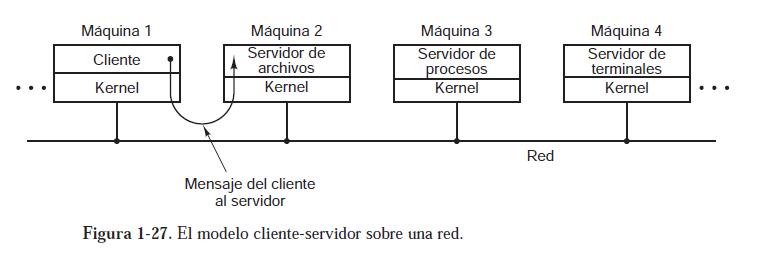
**Modelo cliente-servidor**

El modelo cliente-servidor es una variante del concepto de microkernel, donde se distinguen dos clases de procesos: los servidores, que proporcionan servicios, y los clientes, que utilizan estos servicios. La comunicación entre clientes y servidores se realiza mediante el paso de mensajes, donde un cliente construye un mensaje indicando su solicitud y lo envía al servidor correspondiente, que realiza el trabajo y envía la respuesta de vuelta.

Este modelo puede aplicarse tanto a un solo equipo, donde los clientes y servidores se ejecutan localmente, como a una red de computadoras conectadas, donde la comunicación se realiza a través de una red de área local o amplia. En este enfoque, los clientes no necesitan saber si la comunicación se gestiona localmente o a través de una red.

El modelo cliente-servidor se utiliza ampliamente en sistemas que involucran usuarios en PCs domésticas como clientes y equipos más grandes en otros lugares como servidores. Por ejemplo, en la Web, una PC envía una petición de una página Web al servidor, y la página se devuelve como respuesta. Este modelo es una abstracción versátil aplicable a una variedad de configuraciones de sistemas.

Las funciones centrales son controladas por el kernel, y la interfaz del usuario la controla el Shell. Se encarga de la planificación de hilos, no de procesos, eso hace posible el multitasking. Manipula interrupciones y excepciones. El objetivo es hacer el microkernel lo más liviano posible y cada vez que necesite gestionar memoria lo pido como un servicio a un proceso en modo usuario.

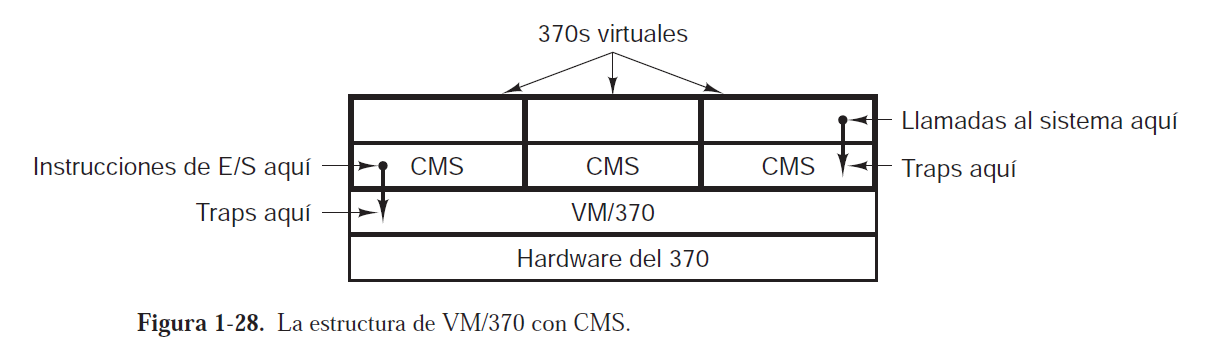


**Máquinas virtuales**

Este sistema, que se llamó VM/370 estaba basado en una astuta observación: un sistema de tiempo compartido proporciona (1) **multiprogramación** y (2) una **máquina extendida** con una interfaz más conveniente que el hardware por sí solo.

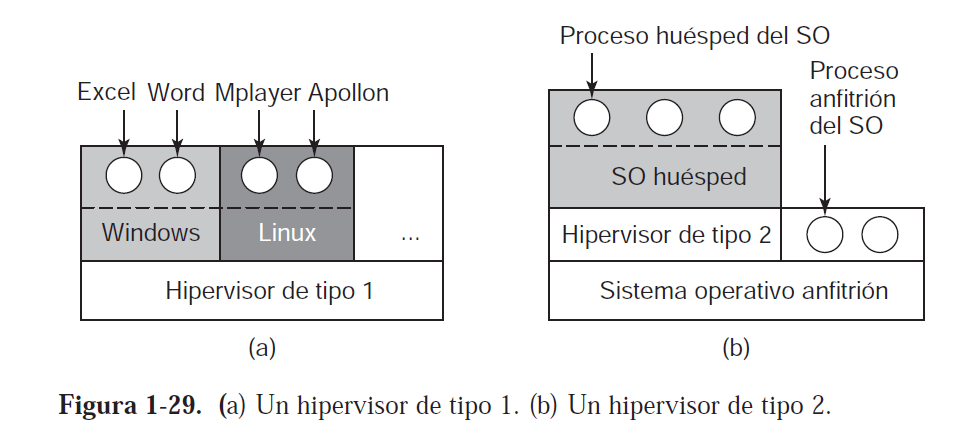
El corazón del sistema, que se conoce como **monitor de máquina virtual**, se ejecuta en el hardware solamente y realiza la multiprogramación, proporcionando no una, sino varias máquinas virtuales a la siguiente capa hacia arriba Las Máquinas virtuales son copias exactas del hardware, incluyendo el modo kernel/ usuario, la E/S, las interrupciones y todo lo demás que tiene la máquina real.

Como cada máquina virtual es idéntica al verdadero hardware, cada una puede ejecutar cualquier sistema operativo que se ejecute directamente sólo en el hardware.



Los clientes que rentan una máquina virtual pueden ejecutar cualesquier sistema operativo y software que deseen, pero a una fracción del costo de un servidor dedicado (debido a que la misma máquina física soporta muchas máquinas virtuales al mismo tiempo).

El término “monitor de máquinas virtuales” ha cambiado su nombre por el de hipervisor de tipo 1 en años recientes.



Para poder ejecutar software de máquina virtual en una computadora, su CPU debe ser virtualizable.

Los hipervisores de tipo 2 se ejecutan como programas de aplicación encima de Windows, Linux o algún otro sistema operativo, conocido como sistema operativo anfitrión. Una vez que se inicia un hipervisor de tipo 2, lee el CD-ROM de instalación para el sistema operativo huésped elegido y lo instala en un disco virtual, que es tan sólo un gran archivo en el sistema de archivos del sistema operativo anfitrión.

Cuando un sistema operativo que opera en una máquina virtual (en modo usuario) ejecuta una instrucción privilegiada, tal como para modificar el PSW o realizar una operación de E/S, es esencial que el hardware la atrape para el monitor de la máquina virtual, de manera que la instrucción se pueda emular en el software.

**Multitarea**: Un sistema operativo multitarea divide el tiempo de procesador disponible entre los procesos o subprocesos que lo necesitan. El sistema está diseñado para la multitarea preferente; asigna un segmento de tiempo de procesador a cada subproceso que ejecuta.

**Multiprogramación**: La multiprogramación es una técnica sistemas operativos donde varios programas se cargan en la memoria simultáneamente. El CPU cambia rápidamente entre los programas, permitiendo la ejecución simultánea.

**Abstracciones**:  elemento del [sistema operativo](https://es.wikipedia.org/wiki/Sistema_operativo) que funciona como una [interfaz](https://es.wikipedia.org/wiki/Interfaz) entre el [software](https://es.wikipedia.org/wiki/Software) y el [hardware](https://es.wikipedia.org/wiki/Hardware) del sistema, proveyendo una plataforma de hardware consistente sobre la cual corren las aplicaciones.

**1.2**

**PROCESOS E HILOS**

**Proceso:**  puede entenderse informalmente como un [**programa**](https://es.wikipedia.org/wiki/Programa_inform%C3%A1tico)**en ejecución**.

* Formalmente un proceso es "**Una unidad de actividad que se caracteriza por la ejecución de una secuencia de instrucciones, un estado actual, y un conjunto de recursos del sistema asociados"**.
* Un proceso no es más que una instancia de un programa en ejecución, incluyendo los valores actuales del contador de programa, los registros y las variables.

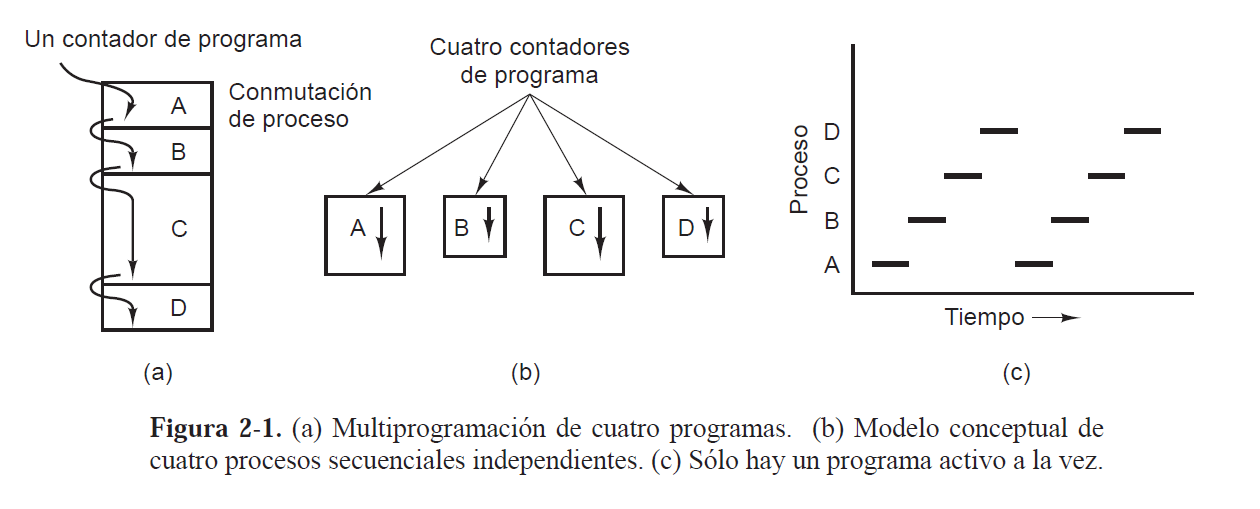
En cualquier sistema de multiprogramación, la CPU conmuta de un proceso a otro con rapidez, ejecutando cada uno durante décimas o centésimas de milisegundos: hablando en sentido estricto, en cualquier instante la CPU está ejecutando sólo un proceso, y en el transcurso de 1 segundo podría trabajar en varios de ellos.

**El modelo del proceso**

Todo el software ejecutable en la computadora, que algunas veces incluye al sistema operativo, se organiza en varios **procesos secuenciales**.

Un proceso no es más que una instancia de un programa en ejecución, incluyendo los valores actuales del contador de programa, los registros y las variables.

Para entender el sistema es mucho más fácil pensar en una colección de procesos que se ejecutan en (pseudo) paralelo, en vez de tratar de llevar la cuenta de cómo la CPU conmuta de programa en programa. Esta conmutación rápida de un proceso a otro se conoce como **multiprogramación**.



Sólo hay un contador de programa físico, por lo que cuando se ejecuta cada proceso, se carga su contador de programa lógico en el contador de programa real. Cuando termina (por el tiempo que tenga asignado), el contador de programa físico se guarda en el contador de programa lógico almacenado, del proceso en memoria.

La velocidad a la que un proceso ejecuta sus cálculos no es uniforme y tal vez ni siquiera sea reproducible si se ejecutan los mismos procesos de nuevo.

La diferencia entre un proceso y un programa es sutil pero crucial.

EJEMPLO

Un científico computacional hornea un pastel

En esta analogía, la receta es el programa (es decir, un algoritmo expresado en cierta notación adecuada), el científico computacional es el procesador (CPU) y los ingredientes del pastel son los datos de entrada. El proceso es la actividad que consiste en que nuestro cocinero vaya leyendo la receta, obteniendo los ingredientes y horneando el pastel.

Ahora imagine que el hijo del científico entra corriendo y gritando que una abeja acaba de picarlo. El científico computacional registra el punto de la receta en el que estaba (el estado del proceso en curso se guarda), saca un libro de primeros auxilios y empieza a seguir las instrucciones que contiene. Aquí el procesador conmuta de un proceso (hornear el pastel) a uno de mayor prioridad (administrar cuidados médicos), cada uno de los cuales tiene un programa distinto (la receta y el libro de primeros auxilios).

La idea clave es que un proceso es una actividad de cierto tipo: tiene un programa, una entrada, una salida y un estado.

Si un programa se está ejecutando por duplicado cuenta como dos procesos.

**Creación de un proceso**

Hay cuatro eventos principales que provocan la creación de procesos:

1. El arranque del sistema.
2. La ejecución, desde un proceso, de una llamada al sistema para creación de procesos.
3. Una petición de usuario para crear un proceso.
4. El inicio de un trabajo por lotes.

Los procesos que permanecen en segundo plano para manejar ciertas actividades se conocen como demonios (daemons).

En UNIX podemos utilizar el programa ps para listar los procesos en ejecución.

A menudo, un proceso en ejecución emitirá llamadas al sistema para crear uno o más procesos nuevos, para que le ayuden a realizar su trabajo.

En UNIX sólo hay una llamada al sistema para crear un proceso: **fork.** Esta llamada crea un clon exacto del proceso que hizo la llamada.

Por lo general, el proceso hijo ejecuta después a **execve** o una llamada al sistema similar para cambiar su imagen de memoria y ejecutar un nuevo programa.

Tanto en UNIX como en Windows, una vez que se crea un proceso, el padre y el hijo tienen sus propios espacios de direcciones distintos.

**Terminación de procesos**

Tarde o temprano el nuevo proceso terminará, por lo general debido a una de las siguientes condiciones:

1. Salida normal (voluntaria).
2. Salida por error (voluntaria).
3. Error fatal (involuntaria).
4. Eliminado por otro proceso (involuntaria). (cpmandop kill)

La mayoría de los procesos terminan debido a que han concluido su trabajo.

Cuando un compilador ha compilado el programa que recibe, ejecuta una llamada al sistema para indicar al sistema operativo que ha terminado. Esta llamada es **exit** en UNIX.

La cuarta razón por la que un proceso podría terminar es que ejecute una llamada al sistema que indique al sistema operativo que elimine otros procesos. En UNIX esta llamada es **kill**.

cuando un proceso termina (ya sea en forma voluntaria o forzosa) todos los procesos que creó se eliminan de inmediato también.

**Jerarquías de procesos**

El proceso hijo puede crear por sí mismo más procesos, formando una jerarquía de procesos.

En UNIX, un proceso y todos sus hijos, junto con sus posteriores descendientes, forman un grupo de procesos.

Los procesos se identifican con un ID de proceso, denominado PID. Al crear un proceso, este obtiene copias de la pila, datos, heap y segmento text de su padre. Padre e hijo se diferencian en el resultado de la llamada a fork(), al padre le devuelve el PID del hijo y al hijo 0.

**Estados de un proceso**

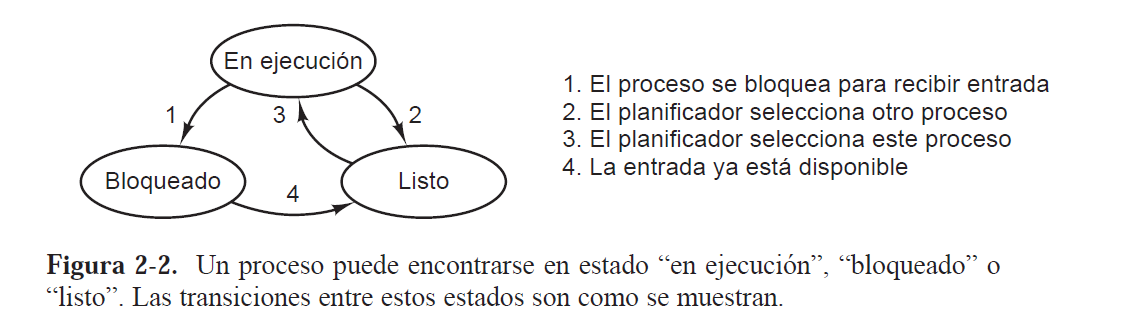
A menudo los procesos necesitan interactuar con otros.Un proceso puede generar cierta salida que otro proceso utiliza como entrada.

Cuando un proceso se bloquea, lo hace debido a que por lógica no puede continuar, comúnmente porque está esperando una entrada que todavía no está disponible.

También es posible que un proceso, que esté listo en concepto y pueda ejecutarse, se detenga debido a que el sistema operativo ha decidido asignar la CPU a otro proceso por cierto tiempo.

Tres estados en los que se puede encontrar un proceso:

1. En ejecución (en realidad está usando la CPU en ese instante).
2. Listo (ejecutable; se detuvo temporalmente para dejar que se ejecute otro proceso).
3. Bloqueado (no puede ejecutarse sino hasta que ocurra cierto evento externo).



La transición 1 ocurre cuando el sistema operativo descubre que un proceso no puede continuar justo en ese momento.

Las transiciones 2 y 3 son producidas por el planificador de procesos, una parte del sistema operativo, sin que el proceso sepa siquiera acerca de ellas.

La transición 2 ocurre cuando el planificador decide que el proceso en ejecución se ha ejecutado el tiempo suficiente y es momento de dejar que otro proceso tenga una parte del tiempo de la CPU. La transición 3 ocurre cuando todos los demás procesos han tenido su parte del tiempo de la CPU y es momento de que el primer proceso obtenga la CPU para ejecutarse de nuevo.

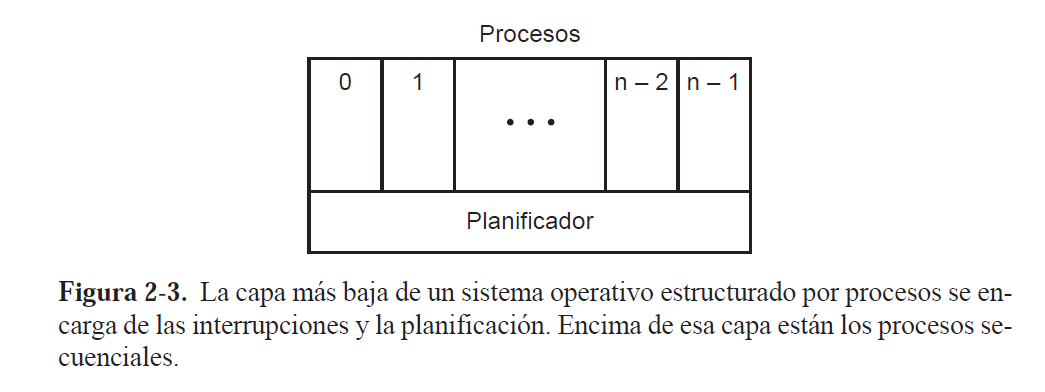
La transición 4 ocurre cuando se produce el evento externo por el que un proceso estaba esperando (como la llegada de ciertos datos de entrada). Si no hay otro proceso en ejecución en ese instante, se activa la transición 3 y el proceso empieza a ejecutarse.

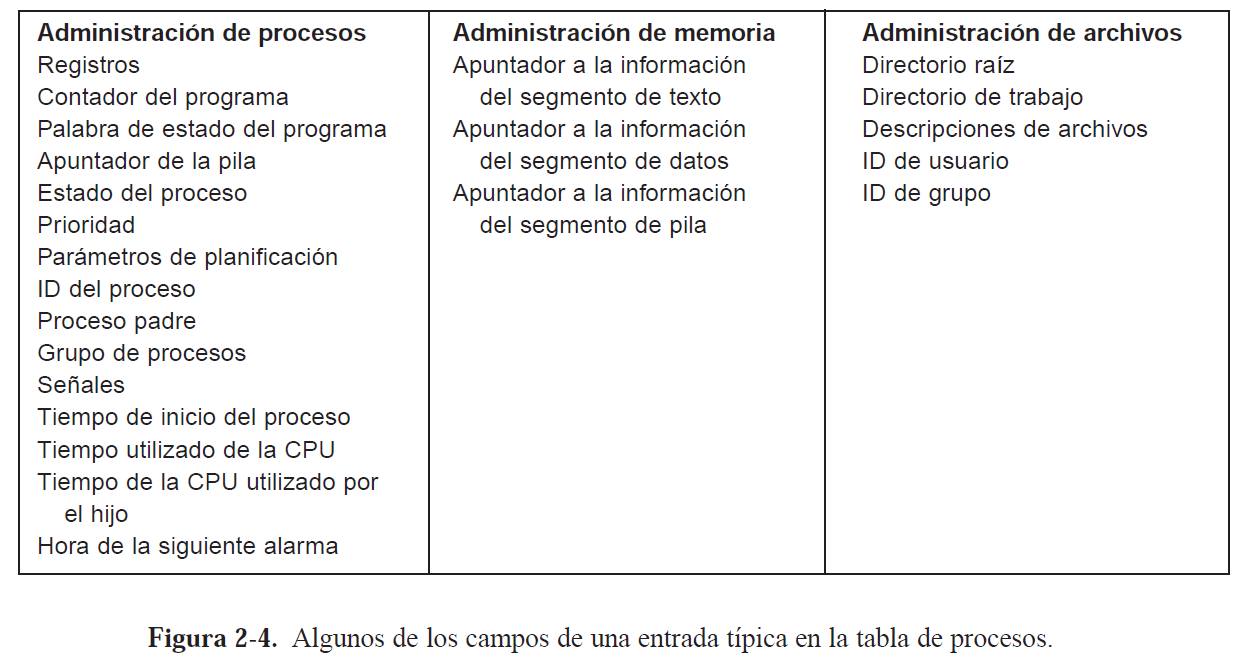
Todo el manejo de las interrupciones y los detalles relacionados con iniciar y detener los procesos se ocultan en lo que aquí se denomina **planificador**, que en realidad no es mucho código.

**Implementación de los procesos**

Para implementar el modelo de procesos, el sistema operativo mantiene una tabla (un arreglo de estructuras) llamada **tabla de procesos**, con sólo una entrada por cada proceso.

Esta entrada contiene información importante acerca del estado del proceso, incluyendo su contador de programa, apuntador de pila, asignación de memoria, estado de sus archivos abiertos, información de contabilidad y planificación, y todo lo demás que debe guardarse acerca del proceso cuando éste cambia del estado **en ejecución a listo o bloqueado**, de manera que se pueda reiniciar posteriormente como si nunca se hubiera detenido.

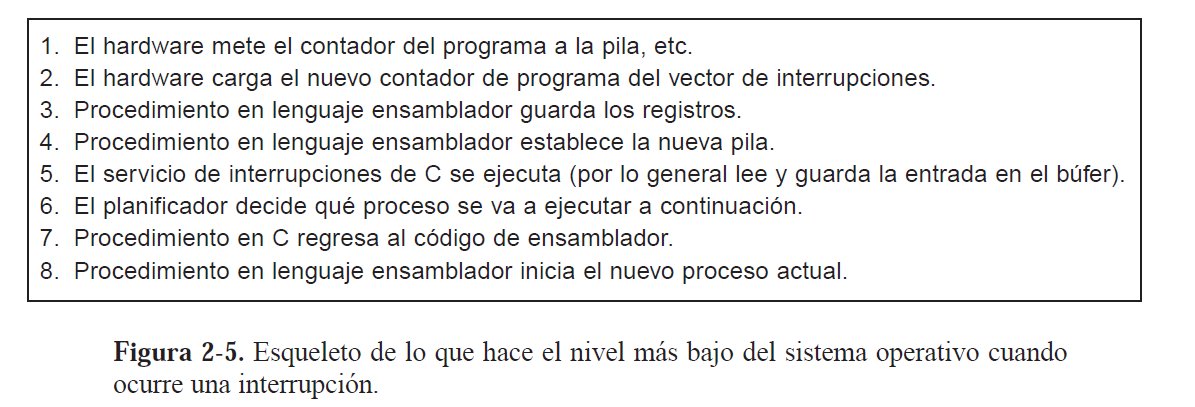




Con cada clase de E/S hay una ubicación asociada (por lo general, en una ubicación cerca de la parte final de la memoria), a la cual se le llama vector de interrupción. Esta ubicación contiene la dirección del procedimiento del servicio de interrupciones.

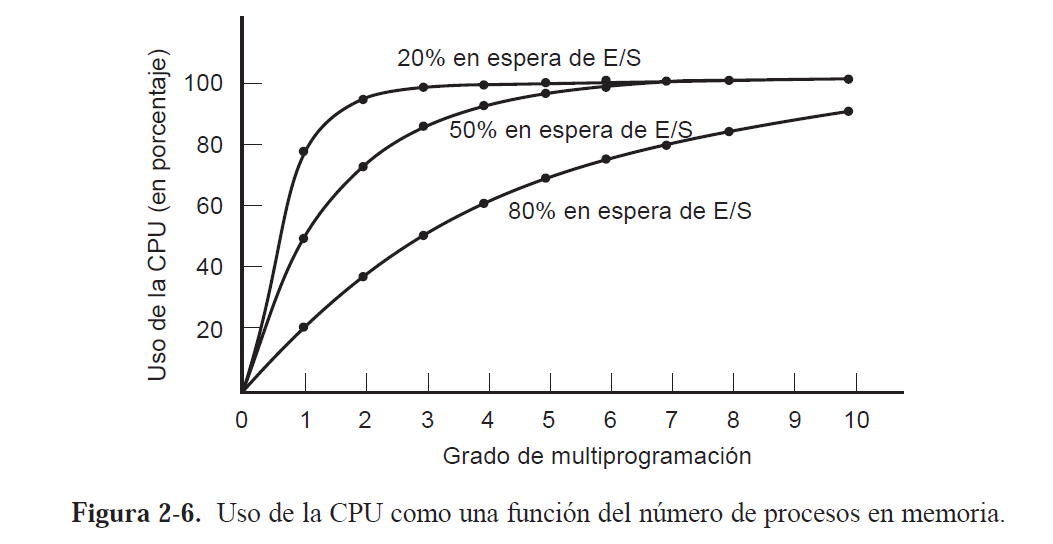
Todas las interrupciones empiezan por guardar los registros, a menudo en la entrada de la tabla de procesos para el proceso actual. Después, se quita la información que la interrupción metió en la pila y el apuntador de pila se establece para que apunte a una pila temporal utilizada por el manejador de procesos.

Cuando termina esta rutina, llama a un procedimiento en C para realizar el resto del trabajo para este tipo de interrupción específico. Cuando ha terminado su trabajo y tal vez ocasionando que algún otro proceso esté entonces listo, el planificador es llamado para ver qué proceso se debe ejecutar a continuación. Después de eso, el control se pasa de vuelta al código en lenguaje ensamblador para cargar los registros y el mapa de memoria para el proceso que entonces es el actual y se empieza a ejecutar.



**Modelación de la multiprogramación**

Un mejor modelo es analizar el uso de la CPU desde un punto de vista probabilístico.

****

Un mejor modelo es analizar el uso de la CPU desde un punto de vista probabilístico. Suponga que un proceso gasta una fracción p de su tiempo esperando a que se complete una operación de E/S. Con n procesos en memoria a la vez, la probabilidad de que todos los n procesos estén esperando la E/S (en cuyo caso, la CPU estará inactiva) es pn. Entonces, el uso de la CPU se obtiene mediante la fórmula

Uso de la CPU = 1-

La figura deja claro que, si los procesos gastan 80 por ciento de su tiempo esperando las operaciones de E/S, por lo menos debe haber 10 procesos en memoria a la vez para que el desperdicio de la CPU esté por debajo de 10%.

**Implementación de Procesos**

El SO mantiene una tabla de procesos con una entrada para cada uno, con información de su estado, contador de programa, puntero de pila, asignación de memoria, estado de archivos abiertos y todo lo que se necesite para que cuando el proceso cambie de estado se pueda reanudar.

**Procesos huérfanos y zombies**

Un proceso se convierte en huérfano cuando su proceso padre termina antes que él. Al huérfano se le asigna como nuevo padre un proceso superior o INIT (PID = 1). Un proceso se convierte en zombie cuando termina antes que su proceso padre. Se borra de memoria el proceso, pero no su PID ni su valor de retorno (status). Un zombie no puede ser matado por kill, cuando el padre hace un wait, el zombie es sacado completamente de memoria.

**1.3**

**HILOS**

**Hilo**: es la unidad más pequeña de ejecución dentro de un proceso en un sistema operativo. Dentro de un proceso, se pueden tener múltiples hilos de ejecución concurrentes que comparten los mismos recursos, incluido el espacio de memoria del proceso.

La principal razón de tener hilos es que en muchas aplicaciones se desarrollan varias actividades a la vez. Algunas de ésas se pueden bloquear de vez en cuando. Al descomponer una aplicación en varios hilos secuenciales que se ejecutan en cuasi-paralelo, el modelo de programación se simplifica.

**VENTAJAS DE LOS HILOS**:

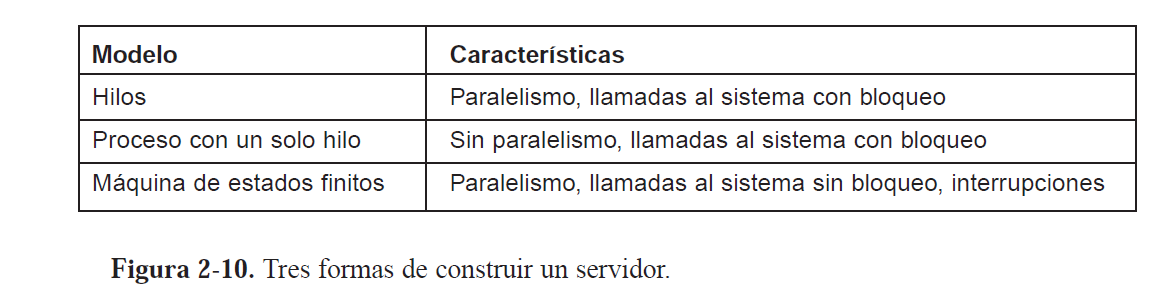
* La habilidad de las entidades en paralelo de compartir un espacio de direcciones y todos sus datos entre ellas.
* Un segundo argumento para tener hilos es que, como son más ligeros que los procesos, son más fáciles de crear (es decir, rápidos) y destruir.
* Una tercera razón de tener hilos es también un argumento relacionado con el rendimiento. Los hilos no producen un aumento en el rendimiento cuando todos ellos están ligados a la CPU, pero cuando hay una cantidad considerable de cálculos y operaciones de E/S, al tener hilos estas actividades se pueden traslapar, con lo cual se agiliza la velocidad de la aplicación.

Los hilos posibilitan el concepto de procesos secuenciales que realizan llamadas al sistema con bloqueo (por ejemplo, para la E/S de disco) y de todas formas logran un paralelismo.

Las llamadas al sistema con bloqueo facilitan el proceso de programación y el paralelismo mejora el rendimiento.

El servidor de un solo hilo retiene la simpleza de las llamadas al sistema con bloqueo, pero pierde rendimiento.

El tercer método logra un alto rendimiento a través del paralelismo, pero utiliza llamadas e interrupciones sin bloqueo y por ende, es difícil de programar.



Un tercer ejemplo en donde los hilos son de utilidad es en las aplicaciones que deben procesar cantidades muy grandes de datos.

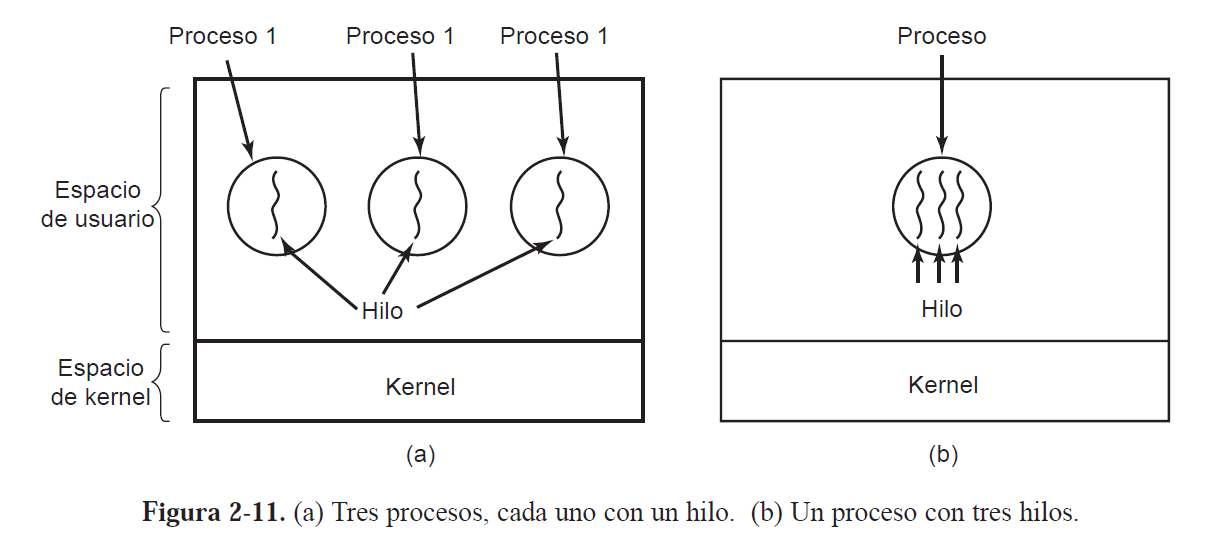
**El modelo clásico de hilo**

El modelo de procesos se basa en dos conceptos independientes: agrupamiento de recursos y ejecución.

El hilo tiene un contador de programa que lleva el registro de cuál instrucción se va a ejecutar a continuación. Tiene registros que contienen sus variables de trabajo actuales. Tiene una pila, que contiene el historial de ejecución, con un conjunto de valores para cada procedimiento al que se haya llamado.

Tener varios procesos ejecutándose en paralelo en un proceso es algo similar a tener varios procesos ejecutándose en paralelo en una computadora.

Como los hilos tienen algunas de las propiedades de los procesos, algunas veces se les llama **procesos ligeros**. El término **multihilamiento** también se utiliza para describir la situación de permitir varios hilos en el mismo proceso.

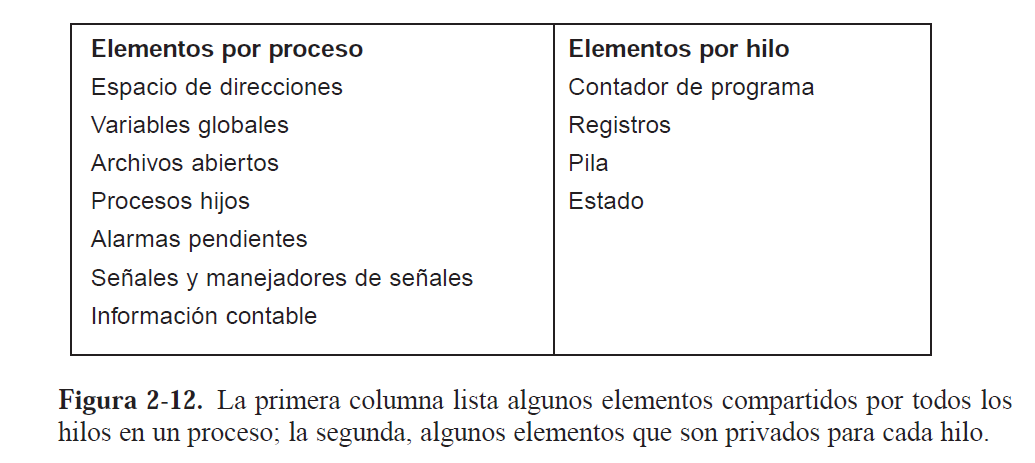


Aunque en ambos casos tenemos tres hilos, en la figura 2-11(a) cada uno de ellos opera en un espacio de direcciones distinto, mientras que en la figura 2-11(b) los tres comparten el mismo espacio de direcciones.

Cuando se ejecuta un proceso con multihilamiento en un sistema con una CPU, los hilos toman turnos para ejecutarse.

El multihilamiento funciona de la misma manera. La CPU conmuta rápidamente entre un hilo y otro, dando la ilusión de que los hilos se ejecutan en paralelo

Como cada hilo puede acceder a cada dirección de memoria dentro del espacio de direcciones del proceso, un hilo puede leer, escribir o incluso borrar la pila de otro hilo.



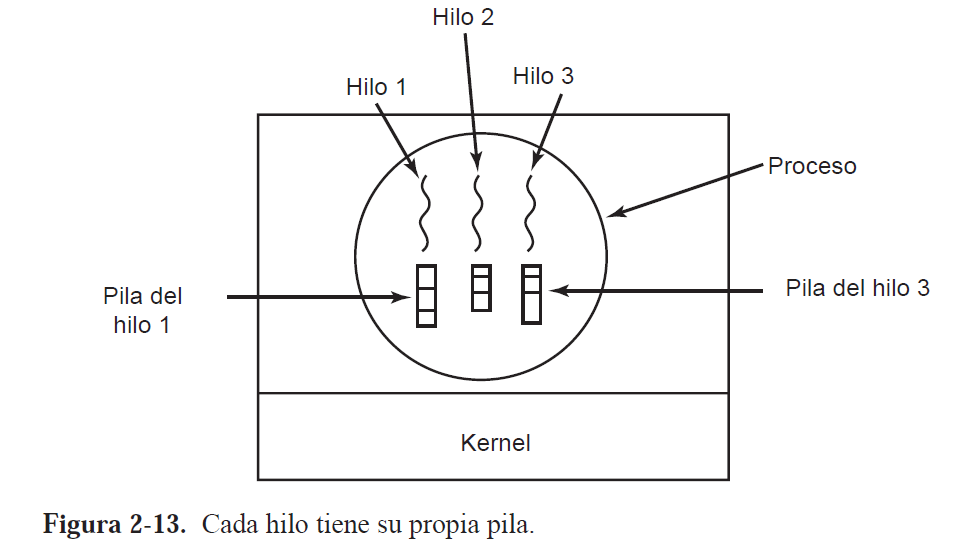
Al igual que un proceso tradicional (es decir, un proceso con sólo un hilo), un hilo puede estar en uno de varios estados:

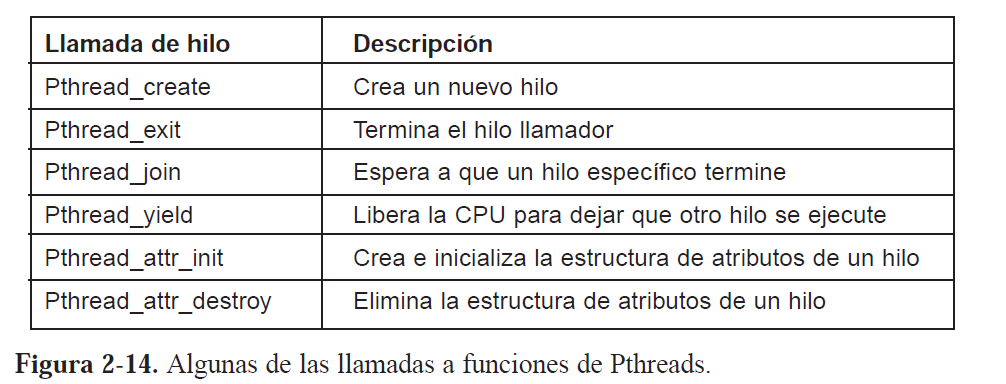
Ejecución (iene la

* CPU en un momento dado y está activo)
* Bloqueado (esperando a que cierto evento lo desbloquee)
* listo (se programa para ejecutarse y lo hará tan pronto como sea su turno)
* terminado

La pila de cada hilo contiene un conjunto de valores para cada procedimiento llamado, pero del que todavía no se ha regresado. Este conjunto de valores contiene las variables locales del procedimiento y la dirección de retorno que se debe utilizar cuando haya terminado la llamada al procedimiento.

Por lo general, cada hilo llama a distintos procedimientos y por ende, tiene un historial de ejecución diferente. Esta es la razón por la cual cada hilo necesita su **propia pila**.





**Implementación de hilos en el espacio de usuario**

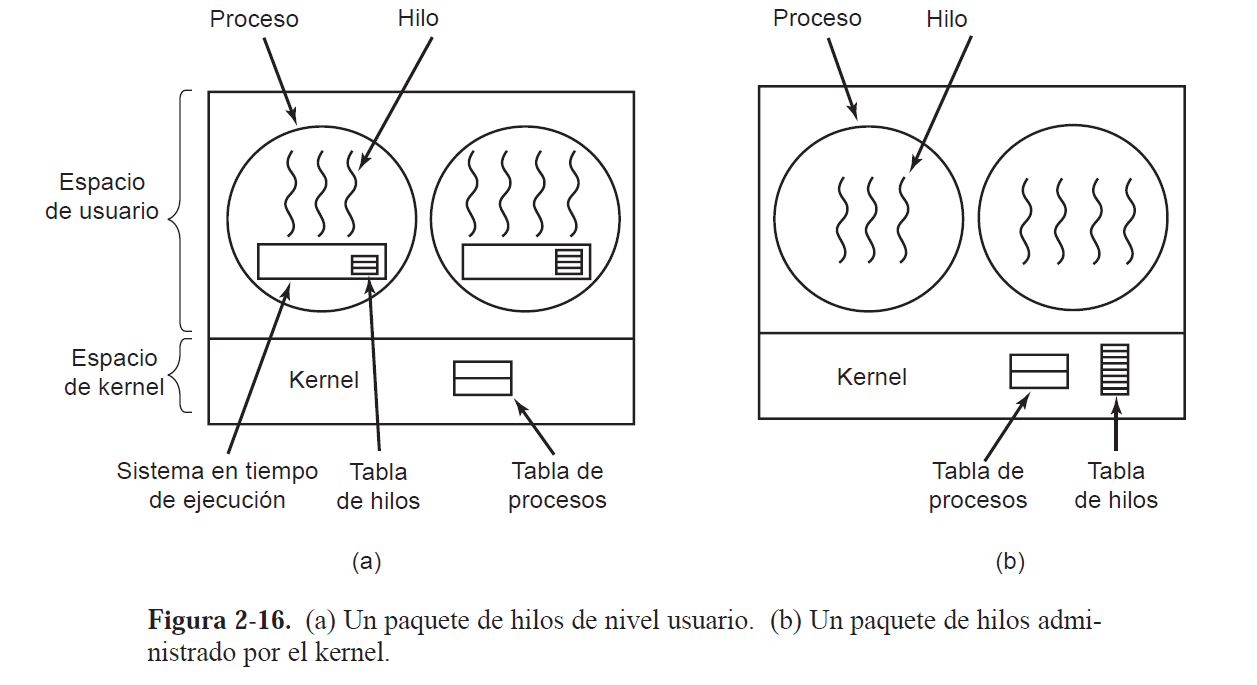
Hay dos formas principales de implementar un paquete de hilos:

* Espacio de usuario
* Espacio de kernel
* **Espacio de usuario**

El primer método es colocar el paquete de hilos completamente en espacio de usuario. El kernel no sabe nada acerca de ellos. En lo que al kernel concierne, está administrando procesos ordinarios con un solo hilo. La primera ventaja, la más obvia, es que un paquete de hilos de nivel usuario puede implementarse en un sistema operativo que no acepte hilos. Con este método, los hilos se implementan mediante una biblioteca.

Cada proceso necesita su propia **tabla de hilos** privada para llevar la cuenta de los hilos en ese proceso. La tabla de hilos es administrada por el **sistema en tiempo de ejecución**.

La tabla de hilos es administrada por el sistema en tiempo de ejecución.



El fragmento de texto proporciona información sobre los hilos de nivel usuario, destacando algunas ventajas y desventajas de este enfoque en comparación con los hilos de nivel kernel.

**Ventajas de los hilos de nivel usuario:**

1. **Eficiencia en la planificación de hilos**: Al evitar las llamadas al kernel, la planificación de hilos puede ser más rápida y eficiente.
2. **Personalización de algoritmos de planificación**: Permite a cada proceso tener su propio algoritmo de planificación personalizado.
3. **Escalabilidad**: Los hilos de nivel usuario escalan mejor en términos de recursos que los hilos del kernel, especialmente en entornos con muchos hilos.

D**esventajas de los hilos de nivel usuario:**

1. **Problemas con las llamadas al sistema de bloqueo**: La implementación de llamadas al sistema de bloqueo puede ser problemática, ya que un hilo bloqueado puede afectar a otros hilos en el mismo proceso.
2. **Dificultades con los fallos de página**: La gestión de fallos de página puede ser complicada, ya que un fallo de página en un hilo puede bloquear todo el proceso.
3. **Problemas de planificación en un solo proceso**: Dentro de un solo proceso, los hilos de nivel usuario pueden no permitir una planificación adecuada de los hilos, ya que no hay interrupciones de reloj.

**Críticas generales:**

* Se plantea la pregunta sobre la utilidad de los hilos de nivel usuario en aplicaciones donde los hilos se bloquean con frecuencia, como en servidores web multihilados.
* Se destaca que para aplicaciones que están limitadas a la CPU y raramente se bloquean, la introducción de hilos puede no ofrecer beneficios significativos.

En resumen, aunque los hilos de nivel usuario pueden proporcionar ventajas de rendimiento, también enfrentan desafíos en situaciones donde se requiere una gestión más robusta de llamadas al sistema, fallos de página y planificación dentro de un solo proceso.

* **Espacio en el kernel**

**Gestión de Hilos a Nivel de Kernel**:

1. No se necesita un sistema en tiempo de ejecución para la creación o destrucción de hilos. El kernel administra una tabla global de hilos.
2. La tabla de hilos del kernel almacena registros, estados y otra información sobre cada hilo.
3. Las llamadas que podrían bloquear un hilo se implementan como llamadas al sistema, con un mayor costo en comparación con las llamadas al sistema de un sistema en tiempo de ejecución.
4. Cuando un hilo se bloquea, el kernel puede decidir ejecutar otro hilo del mismo proceso o de un proceso diferente.
5. Al destruir un hilo, algunos sistemas optan por marcarlo como no ejecutable y reciclarlo más tarde, lo que ahorra cierta sobrecarga.
6. Los hilos de kernel pueden abordar problemas como los fallos de página, pero el costo de las llamadas al sistema es significativamente mayor.

**Desafíos y Consideraciones**:

1. La sobrecarga de las llamadas al sistema es mayor en comparación con los hilos de nivel usuario.
2. Los hilos de kernel no resuelven todos los problemas, como la gestión de procesos que utilizan la llamada a fork o el manejo de señales.
3. La elección entre hilos y procesos depende de la planificación futura del proceso.
4. La gestión de señales presenta desafíos cuando varios hilos se registran para la misma señal.

En resumen, aunque los hilos de kernel ofrecen soluciones a ciertos problemas, también introducen nuevas complejidades y deben considerarse cuidadosamente en función de los requisitos específicos del sistema y la aplicación.

* **Implementaciones híbridas**

En un intento de combinar las ventajas de los hilos de nivel usuario y de los hilos de nivel kernel, se ha explorado la opción de utilizar hilos de nivel kernel y multiplexar los hilos de nivel usuario sobre ellos. Aquí hay un resumen de este enfoque:

**Modelo de Multiplexación de Hilos:**

1. Se utilizan hilos de nivel kernel que son gestionados y planificados por el kernel del sistema operativo.
2. Sobre algunos de estos hilos de nivel kernel, se multiplexan hilos de nivel usuario.
3. El programador tiene la flexibilidad de determinar cuántos hilos de nivel kernel usar y cuántos hilos de nivel usuario multiplexar en cada uno.

**Funcionamiento del Modelo**:

1. El kernel solo es consciente de los hilos de nivel kernel y se encarga de su planificación.
2. Algunos hilos de nivel kernel pueden tener varios hilos de nivel usuario multiplexados sobre ellos.
3. Los hilos de nivel usuario se crean, destruyen y planifican de manera similar a como lo harían en un sistema operativo sin capacidad de multihilamiento.
4. Cada hilo de nivel kernel tiene un conjunto de hilos de nivel usuario que toman turnos para utilizarlo.

**Ventajas y Flexibilidad:**

1. Este modelo proporciona flexibilidad al permitir la combinación de hilos de nivel usuario y kernel según las necesidades del programador.
2. Los beneficios de eficiencia de los hilos de nivel usuario pueden combinarse con la capacidad de planificación y gestión del kernel.

**Consideraciones:**

1. Aunque se logra flexibilidad, la implementación de este modelo puede ser más compleja y requerir una cuidadosa gestión de recursos.
2. El rendimiento dependerá de la eficiencia en la multiplexación de hilos de nivel usuario sobre los hilos de nivel kernel.

En resumen, la multiplexación de hilos proporciona una solución intermedia que busca aprovechar las ventajas de ambos enfoques, permitiendo una planificación eficiente a nivel kernel y la flexibilidad de hilos de nivel usuario en la aplicación.