Resumen

Autor	Juan Pablo Frascino
Reviewed	☐

Introducción a los sistemas operativos

¿Qué es un Sistema Operativo y cuál es su función?

Un sistema operativo es un programa que administra el hardware de una computadora. También proporciona las bases para los programas de aplicación y actúa como un intermediario entre el usuario y el hardware de la computadora.

Existen distintos tipos de sistemas operativos, algunos son para computadoras personales los cuales están diseñados para proporcionar un entorno en el cual el usuario pueda interactuar fácilmente con la computadora y ejecutar programas, otros son hechos para mainframes y están diseñados para optimizar el uso de hardware. Por lo tanto algunos se diseñan para ser prácticos, otros para ser eficientes y otros para ser ambas cosas.

Objetivos de los sistemas operativos

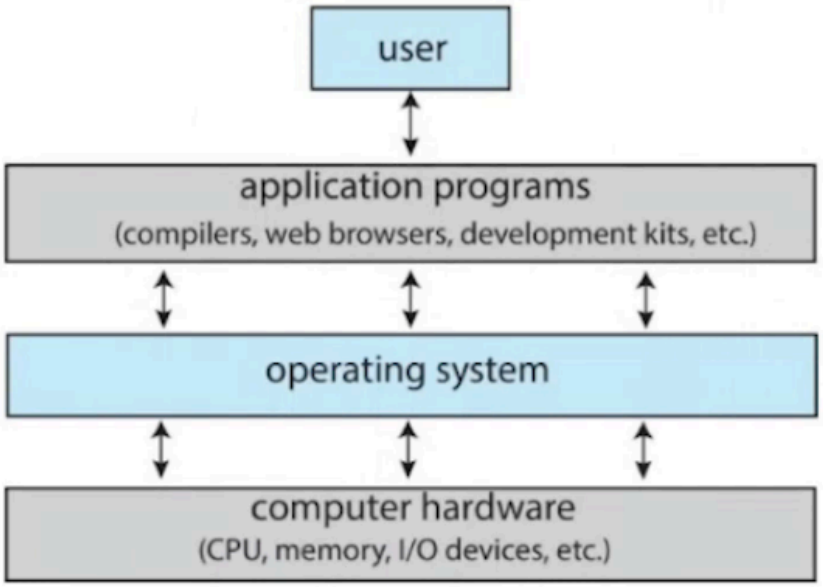
- Ejecutar programas y facilitar la resolución de problemas de los usuarios
- Hacer que el sistema sea cómodo de usar
- Aprovechar los recursos de la computadora

Organización de un sistema de computación

Cuando hablamos con un sistema de computación nos encontramos los siguientes grupos:

1. **El hardware:** está compuesto por los componentes y periféricos como el mouse, teclado etc. A esto se lo conoce como máquina desnuda o naked machine
2. **Sistema operativo**: Capa de software que se agrega sobre la máquina desnuda convirtiéndola en la máquina extendida.
3. **Programas de aplicación:** Sobre este sistema operativo ejecutan algunas aplicaciones, algunas ya vienen integradas con el S.O como el explorador de archivos o los kits de desarrollo.

4. **Usuario**: Quien hace uso de del sistema, pueden ser usuarios o también otras computadoras.



Sistemas de un procesador

En un sistema de un solo procesador hay una CPU principal capaz de ejecutar un conjunto de instrucciones de propósito general, incluyendo instrucciones de los procesos de usuario.

Casi todos los sistemas tienen un procesador de propósito general, como procesadores de E/S que transfieren rápidamente datos entre los componentes del sistema.

Sistemas de multiprocesador (sistemas paralelos)

Estos sistemas disponen de dos o más procesadores que se comunican entre sí, compartiendo el bus de la computadora y, en ocasiones, el reloj, la memoria y los dispositivos periféricos.

Ventajas de los sistemas multiprocesadores:

1. **Mayor rendimiento:** Al tener más procesadores, se realiza más trabajo en menos tiempo.
2. **Economía a escala:** Resultan más baratos que muchos sistemas de un solo procesador, ya que pueden compartir periféricos, almacenamiento masivo y fuentes de alimentación.
3. **Mayor fiabilidad**: Al tener las funciones repartidas, un fallo no hará que deje de funcionar el sistema.

Operación en modo dual

Para asegurar la correcta ejecución del sistema operativo, tenemos que poder distinguir entre la ejecución del código del sistema operativo y del código definido por el usuario. El método que usan la mayoría de los sistema informáticos consiste en proporcionar soporte **hardware** que nos permita diferenciar entre varios modos de ejecución.

Como mínimo, necesitamos dos modos diferentes de operación: **modo usuario y modo kernel**. Un bit de modo, se añade al hardware de la computadora para indicar el modo actual, gracias a este bit podemos diferenciar entre una tarea que se ejecuta en nombre del sistema operativo y otra que se ejecute en nombre del usuario. Cuando el sistema informático está ejecutando una aplicación de usuario, el sistema se encuentra en modo usuario. Sin embargo, cuando una aplicación de usuario solicita un servicio del sistema operativo (a través de una llamada al sistema), debe pasar del modo de usuario al modo kernel para satisfacer la solicitud.

Cuando se arranca el sistema, el hardware se inicia en el modo kernel. El sistema operativo se carga y se inician las aplicaciones de usuario en el modo usuario. Cuando se produce una excepción o interrupción, el hardware conmuta del modo de usuario al modo kernel (cambiando el bit de modo), en consecuencia, el S.O obtiene el control de la computadora.

El modo dual de operación dual nos proporciona los medios para proteger el sistema operativo de los usuarios que puedan causar errores, también para proteger a los usuarios de los errores de otros usuarios. Esta protección se consigue designando algunas de las instrucciones de máquina que pueden causar daño como **instrucciones privilegiadas**. El hardware hace que las instrucciones privilegiadas sólo se ejecuten en el modo kernel. Si se hace un intento de ejecutar una instrucción privilegiada en modo de usuario, el hardware no ejecuta la instrucción sino que la trata como ilegal y envía una excepción al sistema operativo.

La instrucción para conmutar al modo usuario es un ejemplo de instrucción privilegiada. Entre otros ejemplos se incluyen el control de E/S, la gestión del temporizador y la gestión de interrupciones.

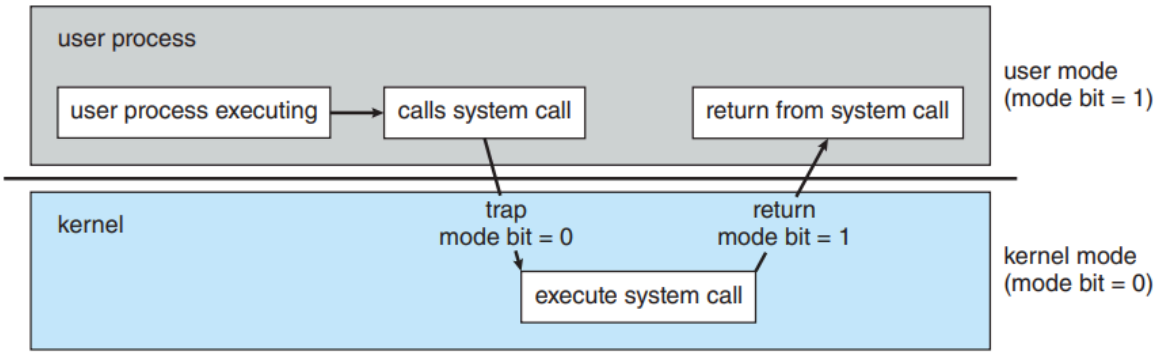


Figure 1.13 Transition from user to kernel mode.

Podemos ver el ciclo de vida de la ejecución de una instrucción en un sistema informático:

1. El control se encuentra inicialmente en manos del sistema operativo, donde las instrucciones se ejecutan en el modo kernel.
2. Cuando se da el control a una aplicación de usuario, se pasa a modo usuario.
3. Finalmente, el control se devuelve al sistema operativo a través de una interrupción, una excepción o una llamada al sistema.

Las **llamadas al sistema** proporcionan los medios para que un programa de usuario pida al sistema operativo que realice tareas reservadas del sistema operativo en nombre del programa de usuario, en otras palabras, son un método usado por un proceso para solicitar la actuación del sistema operativo. Cuando se ejecuta una llamada al sistema, el hardware la trata como una interrupción software.

Gestión de procesos

Sobre los procesos

Un programa en ejecución es un proceso, un programa de usuario de tiempo compartido (ej: un compilador) también es un proceso, un procesador de textos que ejecute un usuario en su pc también es un proceso. Un tarea del sistema, como enviar datos de salida a una impresora, también puede ser un proceso(o parte de él).

Un proceso necesita para llevar a cabo su tarea ciertos recursos, entre los que incluyen tiempo de CPU, memoria, archivos y dispositivos de E/S. Estos recurso se proporcionan al proceso en el momento de crearlo o se le asignan mientras se está ejecutando. Además se le puede enviar ciertos datos de inicialización(entradas).

Un programa por sí solo no es un proceso; un programa es una entidad *pasiva*, tal como los contenidos de un archivo almacenado en disco, mientras que un proceso es una entidad *activa*. Un proceso de un solo hilo tiene un contador de programa que especifica la siguiente instrucción que hay que ejecutar y su ejecución es secuencial. Por otra parte, están los procesos multihilos que tienen múltiples contadores de programa donde cada uno de ellos apunta a la siguiente instrucción que cada hilo ejecutara.

Tareas de gestión de procesos

El S.O es responsable de las siguientes actividad en cuanto a la gestión de procesos:

1. Crear y borrar los proceso de usuario y del sistema.
2. Suspender y reanudar los procesos.
3. Proporcionar mecanismos para la sincronización de procesos.
4. Proporcionar mecanismos para la comunicación entre procesos.
5. Proporcionar mecanismos para el tratamiento de los interbloqueos.

Gestión de memoria

La memoria es fundamental en la operación de un sistema informático moderno. La memoria principal es una matriz de palabras o bytes cuyo tamaño se encuentra en el rango de cientos de miles a miles de millones de posiciones distintas. Cada palabra o byte tiene su propia dirección.

La memoria principal es un repositorio de datos rápidamente accesibles, compartida por la CPU y los dispositivos de E/S.

Para que un programa puede ejecutarse, debe estar asignado a direcciones absolutas y cargado en memoria. Mientras el programa se está ejecutando, accede a las instrucciones y a los datos de la memoria generando dichas direcciones absolutas. Finalmente, el programa termina, su espacio de memoria se libera y otro programa puede ser cargado y ejecutado.

Para mejorar la utilización de la CPU como la velocidad de respuesta, las computadoras de propósito general pueden mantener varios programas en memoria. Se utilizan distintos esquemas de gestión de memoria dependiendo de distintos factores, especialmente relativos al hardware.

Tareas de gestión de memoria

El S.O es responsable de las siguientes actividades en cuanto a la gestión de memoria:

1. Controlar qué partes de la memoria están actualmente en uso y por parte de quién.
2. Decidir qué datos y procesos (o parte de procesos) añadir o extraer de la memoria.
3. Asignar y liberar la asignación de espacio de memoria según sea necesario.

ESTRUCTURAS DEL SISTEMA OPERATIVO

Un sistema operativo, en esencia, es una capa de software que se sitúa entre el usuario y el hardware, facilitando el uso y la eficiencia del sistema de computación.

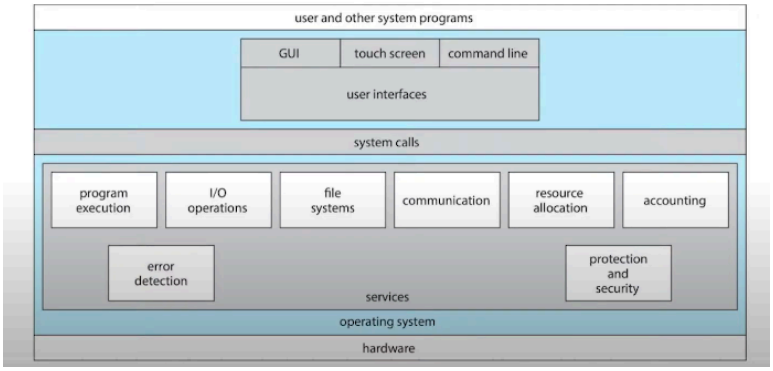
Imaginemos una computadora sin sistema operativo, una "máquina desnuda" (naked machine). Sería extremadamente difícil de manejar y poco eficiente. El sistema operativo aporta la organización y las herramientas para que la interacción con la máquina sea fluida y productiva.

Componentes Claves del Sistema Operativo

El sistema operativo no es una entidad única, sino un conjunto de **módulos internos** que trabajan de forma coordinada para ofrecer diversos servicios. Entre los servicios más importantes se encuentran:

- **Ejecución de programas:** El sistema operativo se encarga de cargar los programas en memoria y administrar su ejecución. En la jerga de los sistemas operativos, un programa en ejecución recibe el nombre de **proceso**.
- **Operaciones de entrada/salida (E/S):** Este módulo se encarga de gestionar la comunicación con los dispositivos de hardware, como el teclado, el ratón, la pantalla o las unidades de almacenamiento.
- **Sistema de archivos (File System):** Este módulo es el responsable de organizar y administrar los archivos en los dispositivos de almacenamiento. Define la estructura del sistema de archivos, gestiona el espacio en disco, indica qué espacios están ocupados y libres, y a qué archivos pertenecen.
- **Comunicaciones:** Facilita la comunicación entre diferentes procesos, tanto dentro de una misma máquina (comunicación local) como entre diferentes máquinas (comunicación remota). Estas facilidades de comunicación entre procesos se denominan **IPC (Inter Process Communication)**.

- **Asignación de recursos:** El sistema operativo arbitra el uso de los recursos del sistema, como la CPU, la memoria, las impresoras o las unidades de cinta. Decide qué proceso o usuario tiene acceso a un recurso en un momento dado, y si ese acceso es exclusivo o compartido.
- **Detección de errores:** Este módulo se encarga de identificar y gestionar los errores que puedan surgir durante el funcionamiento del sistema.
- **Protección y seguridad:** El sistema operativo se encarga de proteger el sistema de accesos no autorizados y de garantizar la seguridad de los datos. Gestiona la creación de usuarios, la asignación de permisos, la autenticación y la autorización de los usuarios para acceder a diferentes recursos.



Interacción con el Sistema Operativo

Para acceder a los servicios del sistema operativo, las aplicaciones utilizan **llamadas al sistema (system calls)**. Se trata de un conjunto de instrucciones especiales que permiten a los programas solicitar al sistema operativo que realice una tarea específica.

Los usuarios interactúan con el sistema operativo a través de una **interfaz**, que puede ser de dos tipos:

- **Interfaz de línea de comandos (CLI):** El usuario introduce comandos de texto para interactuar con el sistema. En el mundo UNIX, el intérprete de comandos se llama **shell(BASH, BOURNE AGAIN SHELL)**.
- **Interfaz gráfica de usuario (GUI):** El usuario interactúa con el sistema mediante elementos gráficos como iconos, ventanas y menús.

El Kernel: El Corazón del Sistema Operativo

El kernel es el núcleo del sistema operativo. Es la parte del sistema operativo que se ejecuta en modo privilegiado y tiene acceso directo a los recursos de hardware. Es responsable de

gestionar la memoria, la CPU, los dispositivos de E/S y la comunicación entre procesos.

POSIX: Hacia la Portabilidad de los Sistemas

POSIX (Portable Operating System Interface) es un conjunto de estándares que define las **APIs (Application Programming Interfaces)** que un sistema operativo debe soportar para ser considerado compatible con POSIX. La estandarización de las APIs facilita la portabilidad de las aplicaciones entre diferentes variantes de UNIX.

La Evolución de los Sistemas Operativos: Estructuras y Enfoques

A lo largo de la historia, los sistemas operativos han evolucionado en su diseño y estructura.

Estructuras de Sistemas Operativos:

- **Monolítico:** El kernel es una sola unidad grande que contiene todos los componentes del sistema operativo.
- **Modular:** El kernel se divide en módulos independientes que pueden cargarse y descargarse dinámicamente.
- **En capas:** El sistema operativo se organiza en capas, donde cada capa proporciona un conjunto de servicios a la capa superior.
- **Microkernel:** El kernel se reduce a su mínima expresión, delegando la mayor parte de las funciones del sistema operativo a procesos que se ejecutan en modo usuario.

Windows NT y la Aproximación al Microkernel

Windows NT, desarrollado por Microsoft, fue diseñado con una estructura de microkernel. Sin embargo, el "kernel" de Windows NT (llamado **ejecutivo**) no es tan pequeño como un microkernel puro. Muchas funciones del sistema operativo se implementan dentro del ejecutivo, y los subsistemas (como POSIX, Win32 y OS/2) se ejecutan como procesos en modo usuario.

PROCESOS

Se puede considerar al procesador como el recurso más importante de la computadora, este ejecuta las instrucciones que solucionan los problemas que tienen los usuarios. Cuando hablamos de procesador sabemos que este ejecuta instrucciones que forman parte de un programa y se dice que los programas puestos en ejecución son procesos.

Concepto de proceso

Un proceso es un programa en ejecución. La ejecución del proceso debe avanzar en forma secuencial, no hay una ejecución en paralelo de instrucciones dentro de un único proceso, es decir, que tenemos un programa en disco el cual abrimos mediante un doble click o escribiendo su nombre y presionando enter en caso de utilizar una shell, el S.O busca ese programa y lo carga en memoria para ejecutarlo, podemos considerar esto como transformarlo en proceso, el proceso deja de identificarse por el nombre y pasa a identificarse por un *process ID* que es una identificación única del proceso y el programa comienza a ejecutarse.

Un programa es una entidad pasiva en disco, este no hace nada en principio, para que haga algo tiene que transformarse en proceso.

Hilo de ejecución (Thread)

La ejecución consiste en realizar instrucciones de manera secuencial, también puede saltar de un punto a otro si se encuentra una bifurcación. Si pudiéramos ir marcando las instrucciones que va recorriendo el proceso, esa línea me describe un hilo de ejecución en inglés llamado **Thread.**

Cuando los proceso aparecieron conceptualmente, estos tenían un único hilo de ejecución o thread. Por lo tanto podíamos conocer en un momento dado que instrucción estaba realizando un proceso en particular, debido a que no existía la posibilidad de tener dos procesos ejecutando a la vez. Si mi proceso tiene un único hilo de ejecución, no hay posibilidad de tener ejecución paralela dentro de un proceso.

Estructura de un proceso en memoria

Si tomamos ese programa que el sistema operativo lo carga en memoria y transforma en proceso, yo podría pensarlo como que ese proceso ocupa un espacio de direcciones en memoria , que supongamos que comienza en una posición 0 (no física sino relativa a su inicio), en donde tengo un lugar donde está el programa (en la bibliografía le dicen texto en lugar de código) donde están las instrucciones.

Entonces a un proceso se le asignan 4 áreas fundamentales de memorias las cuales son:

1. Text section

En esta área se guarda el código del proceso, en general el código se encuentra separado de los datos. Los procesadores tienen varios registros en el procesador para procesar información donde tienen un registro que actúa como puntero a la próxima instrucción (PC) este registro

tiene direcciones de memoria que hacen referencia a esta sección donde se encuentra el código.

En la bibliografía se refieren al código como texto.

2. Data section

En el espacio llamado **data** se guardan los datos o variables globales.

Muchas veces los procesadores para hacer referencia a un dato claramente no usan el PC para referirse a un dato porque este apunta a direcciones, para apuntar a datos se usan uno o más registros que referencian al área de datos.

En la arquitectura Intel, la sección que apunta al área de datos se suele llama segmento de datos, y hay un registro llamada ds (data segment) qeu es una especie de puntero a donde comienzan el área de datos, este trabaja junto a otros registros que indican el desplazamiento con respecto al origen de la sección.

3. Stack

Cuando se hace una llamada a una subrutina, El hardware guarda la dirección de retorno de esa subrutina en la pila (**stack**), bifurca y comienza a ejecutar la subrutina que se llamó, cuando se termina y llega al retorno de la subrutina, el hardware saca del stack la dirección de retorno guardada y la pone en el registro de próxima instrucción para que automáticamente se siga ejecutando en el orden que venía.

Cuando llamamos a una rutina o función, puede que tengan variables locales, estas variables no se almacenan en el área de datos porque no son globales, las variables locales se almacenan en el Stack. Cuando la rutina que invoqué, que tiene cierta cantidad de variables locales, va a retornar, y como resultado de ese retorno va a desapilar a las variables locales (las descarta) y a su vez va a desapilar la dirección de retorno, es por eso que dicen que las variables locales viven solamente durante el tiempo que vive la función, al momento del retorno de la función las variables locales dejan de existir.

Otra cosa que almacena el Stack son los parámetros que se le pasa a una función, si yo llamo a una función que recibe dos parámetros, el código que invoca a la función lo primero que hace es apilar en el Stack el parámetro 1, luego el parámetro 2, luego llama a la rutina con lo cual apila la dirección de retorno, después cuando entra a la función apila las variables locales, al terminar la función se desapilan las variables locales, la dirección de retorno, se vuelve a donde se invocó la función y las siguientes instrucciones de la invocación a la instrucción es desapilar el espacio que se le asignó a los parámetros que se le pasaron a la función y después la función invocadora continúa con su ejecución de instrucciones.

4. Heap

Mi programa puede que requiere espacio de memoria gestionado dinámicamente, algunos lenguajes de programación utilizan esto como por ejemplo en la POO cuando se crea un objeto, hay que asignarle memoria en el momento que se crea el objeto. O podría ser que se esté programando en un lenguaje de alto nivel donde se requiere memoria en forma dinámica como en C llamando a la función m alloc . Esa gestión de **memoria dinámica** se obtiene de otra área de memoria preasignada al proceso con el nombre de área **Heap** (o montículo en español). Tanto el área de Heap como la de Stack no tienen un límite fijo sino que uno crece en sentido y la otra en otro, la idea es que no se choquen.

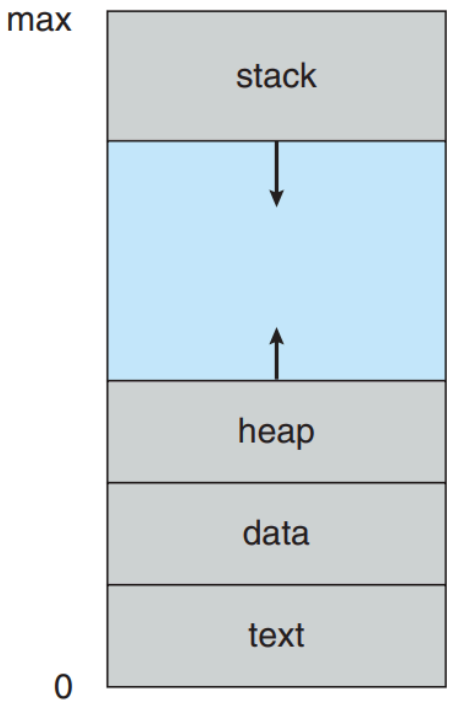


Figure 3.1 Layout of a process in memory.

En resúmen:

Stack: Contiene las direcciones de retorno de las subrutinas.T También contienen las variables locales y los parámetros que se le pasan a una función.

Data: Contiene los datos o variables globales.

Heap: Gestión de memoria dinámica.

Estados de los procesos

Cuando el sistema operativo pone un programa en ejecución, dando comienzo a un proceso, este tiene un ciclo de vida en el que pasa por distintos estados.

Una introducción a estos procesos:

1. New: El proceso está siendo creado.
2. Ready: El proceso está a la espera de que se le asigne el procesador.
3. Running: Las instrucciones se están ejecutando.
4. Waiting: El proceso esta esperando que un evento ocurra o termine.
5. Terminated: El proceso has finished execution.

Diagrama de transición de estados de un proceso

En el siguiente diagrama de transición de estados podemos apreciar todos los estados que puede atravesar un proceso a lo largo de su vida.

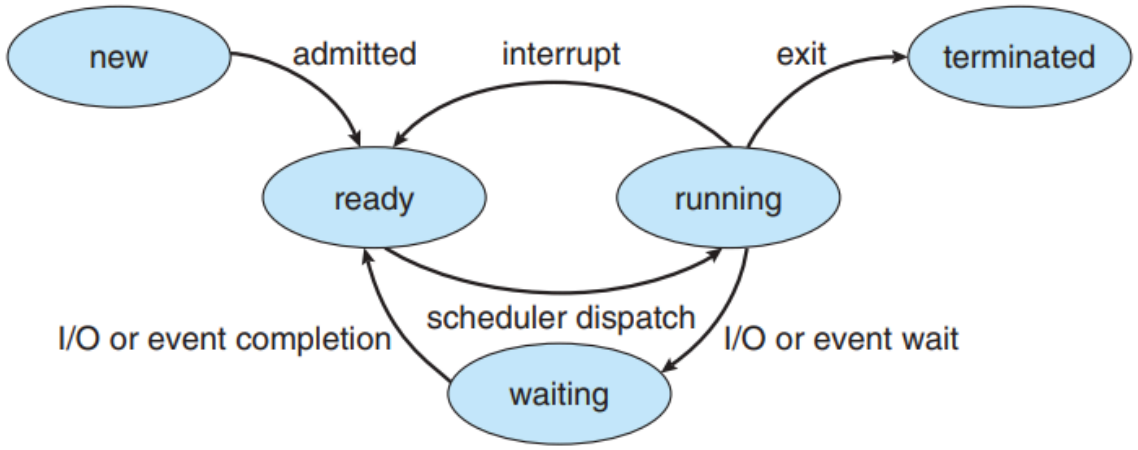


Figure 3.2 Diagram of process state.

Un proceso comienza en el estado **New** , el sistema operativo guarda información sobre él, le asigna un número que lo identifica unívocamente y lo pasa al estado **Ready**, que significa que está listo para ponerse en ejecución. El S.O puede tener muchos procesos en estado de ready al mismo tiempo.

Cuando un proceso es seleccionado y comienzan a ejecutarse sus instrucciones su nuevo estado es **Running.** En el caso de un S.O que corre sobre un hardware de un solo procesador, puede tener muchos en estado Ready pero uno solo en estado Running.

Si por algún motivo el proceso que se estaba ejecutando deja de hacerlo, el sistema operativo se encarga de mirar entre todos los procesos que están en estado ready y a través de un algoritmo de selección toma uno de esos procesos y lo pone en ejecución. El módulo del sistema operativo que cumple con la función de realizar esto es el **Scheduler** (planificador). Inmediatamente después de que el Scheduler seleccione un proceso se invoca a otro módulo del sistema operativo llamado **Dispatcher,** este se encarga de poner en ejecución al módulo seleccionado por el Scheduler, cambiando así el estado de un proceso que estaba en Ready al estado Running.

Un proceso en estado Running tiene dos posibles destinos, uno es detenerse y el otro terminar. Cuando un proceso en estado Running le solicita un servicio al sistema operativo, como por ejemplo, una operación de entrada y salida, el sistema operativo lo cambia al estado **Waiting** y como consecuencia de esto, se le quita el recurso del procesador y deja de ejecutarse. Mientras tanto el S.O realiza la operación solicitada y para aprovechar ese lapso de tiempo invoca al Scheduler y al Dispatcher para tomar un nuevo proceso del estado Ready y ponerlo en ejecución. Cuando la operación de entrada y salida solicitada por el primer proceso termine, la controladora asociada genera una interrupción, esta interrupción hace que el procesador deje de ejecutar el segundo proceso que entró en ejecución, y el proceso que quedó la espera del dato/escritura vuelva al estado Ready para luego volver al estado Running y terminar su ejecución. Una vez que este primer proceso termine, el segundo proceso que fue interrumpido puede pasar a terminar su ejecución. Otro ejemplo donde sucede el mismo cambio de Running a Waiting, es cuando un proceso está ejecutando como padre y ejecuta el system call wait(), el S.O mueve ese proceso de Running a Waiting porque está a la espera de un evento(que su proceso hijo muera) y cuando ese evento suceda va a pasar de Waiting a Ready.

La otra posibilidad que tenía un proceso en estado Running es pasar de vuelta al estado Ready a través de una interrupción, el Dispatcher tiene un timer seteado (un temporizador) en un cierto valor antes de que el proceso comience a ejecutarse, mientras el proceso se está ejecutando el timer se va decrementando hasta llegar a 0 y ahí es donde genera una interrupción, la cual es capturada por el S.O y cuando este recibe una interrupción por timer sabe que el proceso en ejecución ya consumió el tiempo (quantum) que se le asignó y saca ese proceso del estado Running y lo regresa al estado Ready.

Cuando un proceso llega a su final entonces le avisa al sistema operativo a través de una llamad al sistema que terminó y cambia al estado **Terminated**. Este es un estado terminal en el que el proceso cargado en memoria ya no tiene razón para seguir estándolo por lo tanto se le quitan todos los recursos (considerando a la memoria como recurso).

Distintas muertes de un proceso

Cuando un proceso termina su última instrucción de forma satisfactoria y pasa al estado terminado se dice que el proceso termino o tuvo una **muerte natural o normal**.

La otra posibilidad es que le proceso esté ejecutando y por algún motivo cancele/aborte su ejecución, al ser cancelado por el S.O como que este lo haya matado al proceso, o también puede que un usuario con los permisos necesarios lo haga, por eso se dice que el **proceso murió asesinado** o tuvo una **muerte anormal**. De hecho la instrucción en Linux para terminar un proceso es kill (matar/asesinar).

Generalmente cuando un proceso termina regresa un código de retorno el cual puede ser consultado por el proceso que lo invocó.

Bloque de control de procesos (PCB) 📄

El sistema operativo es responsable de administrar los recursos, y por lo tanto, en estructuras de datos propias, tiene que reflejar el estado de los recursos. Por ejemplo, si tenemos una unidad de cinta conectada, toda la información de control el kernel la debe tener, tanto el código para comunicarse con la cinta (leer, escribir), como también tener las estructuras de datos que reflejan que tenemos esa cinta, saber está siendo ocupada y cuál proceso la está utilizando.

Así como tiene una tabla de recursos físicos externos, como la unidad de cinta mencionada, también tiene una tabla interna en donde guarda información esta abstracción que llamamos proceso. Básicamente tiene información sobre los procesos y esta se guarda en un registro en memoria por cada proceso, que recibe el nombre de **Process Control Block** o bloque de control de procesos.

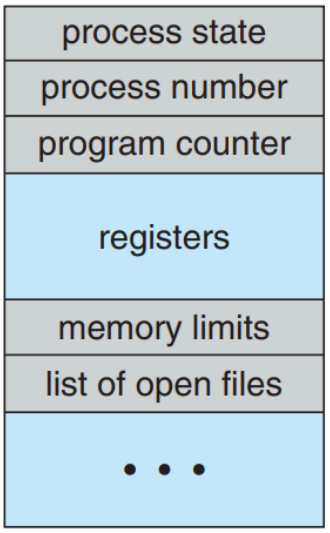


Figure 3.3 Process control block (PCB).

Contenido del PCB

La información que se guarda en el bloque de control de proceso es:

1. **Número de proceso:** Número que identifica unívocamente al proceso. Al sistema operativo le sirve mucho más tratar a los procesos por un número y no por nombre ya que un mismo programa se puede ejecutar varias veces, teniendo así un solo programa pero muchos procesos.
2. **Estado del proceso:** El estado que tiene un estado en un momento dado.
3. **Registro de próxima instrucción** (Program Counter): Registro del procesador con la siguiente instrucción a ejecutar. Útil cuando el proceso que estaba corriendo en estado Running pasa a Waiting y más tarde ese proceso retoma su ejecución, sigue ejecutando desde donde dejó gracias a este registro.
4. **Registros de la CPU:** Los datos guardados en los registros del procesador. A medida que se van ejecutando instrucciones, estos pueden ir modificándose.
5. **Información de planificación de la CPU:** El S.O implementa alguna estrategia de planificación de procesos (scheduling), por lo tanto va a necesitar guardar cierta información para hacer la planificación del procesador, por ejemplo la prioridad del proceso.
6. **Información de gestión de memoria:** Incluye información acerca de dónde está cargado el proceso y cuanto espacio ocupa, la forma de describir como está distribuido el código, los datos, el stack y el heap, depende de cómo administra la memoria el S.O.

7. **Información sobre las entradas y salidas**: Cuales son los dispositivos E/S asignados al proceso, cual es la lista de archivos abierta por el proceso, etc.
8. **Información de contabilidad**: Guarda información sobre la cantidad de CPU y tiempo utilizados. También sobre aquellas operaciones de E/S.

Introducción a los Hilos

Si tomamos un proceso y vamos haciendo un seguimiento de cada instrucción que se va ejecutando una por una, vamos a ir formando una especie de línea o hilo (thread en inglés), entonces tengo un **hilo de ejecución**.

Sistemas monohilos y multihilos

Al principio los sistemas tenían procesos y estos tenían un único thread de ejecución. Este único hilo de control permite que el proceso realice solo una tarea a la vez. Por ejemplo, cuando un proceso esta corriendo un procesador de texto, una instrucción de un único hilo se esta ejecutando ,esta permite controlar una única tarea al a vez, el usuario no podría tipear caracteres mientras utiliza el corrector de ortografía.

La mayoría de los sistemas operativos modernos han ampliado el concepto de proceso para permitir que un proceso tenga múltiples hilos de ejecución y, por lo tanto, realizar más de una tarea a la vez. Esta característica es especialmente beneficiosa en sistemas multinúcleos , donde varios hilos pueden ejecutarse en paralelo. Un procesador multihilos podría, por ejemplo, asignar un hilo para administrar la entrada del usuario mientras
otro hilo ejecuta el corrector ortográfico. En los sistemas que admiten hilos, el PCB se expande para incluir información para cada hilo.

Representación de procesos en Linux

El bloque de control de procesos se representa en C como una estructura o como un registro de Pascal. Esta estructura tiene varios componentes o campos, entre ellos estan:

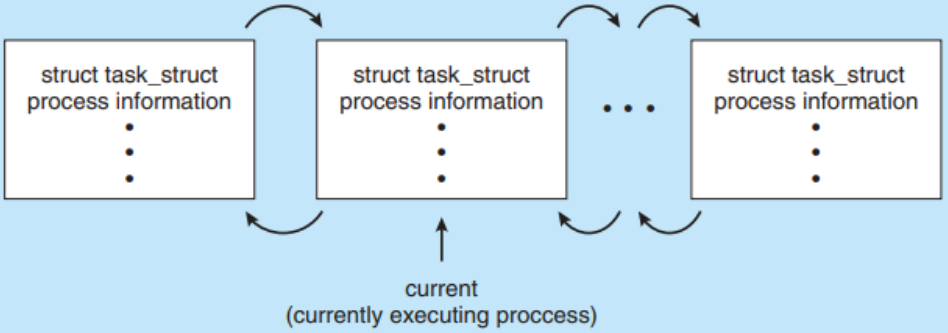
- Pid: El identificador de proceso (Process ID)
- time_slice: El tiempo asignado al proceso (quantum)
- Otras estructuras
- Puntero a otras estructuras

Se suele hablar de tablas pero en la implementación, esto se almacena en una multi-lista doblemente encadenada donde cada nodo describe a un proceso.

▼ Gráfico

```
long state;                     /* state of the process */
struct sched_entity se;         /* scheduling information */
struct task_struct *parent;     /* this process's parent */
struct list_head children;      /* this process's children */
struct files_struct *files;     /* list of open files */
struct mm_struct *mm;           /* address space */
```

For example, the state of a process is represented by the field `long state` in this structure. Within the Linux kernel, all active processes are represented using a doubly linked list of `task_struct`. The kernel maintains a pointer–`current`–to the process currently executing on the system, as shown below:



As an illustration of how the kernel might manipulate one of the fields in the `task_struct` for a specified process, let's assume the system would like to change the state of the process currently running to the value `new_state`. If `current` is a pointer to the process currently executing, its state is changed with the following:

```
current->state = new_state;
```

Planificación de procesos

El Scheduler selecciona entre los distintos procesos que están disponibles el siguiente proceso a ejecutarse sobre un determinado núcleo. Su objetivo es maximizar el uso de la CPU haciendo un cambio entre ellos lo más rápido posible. Ese lapso de tiempo entre que el Scheduler selecciona entre los distintos procesos, es una tarea puramente administrativa, se busca que se realice lo más rápido posible ya que lo verdaderamente importante es la ejecución del proceso y no las tareas administrativas entre elegir y sacar alguno.

Queue's de estados

Por otro lado, el Scheduler debe tener sus estructuras de datos para reflejar el estado en el cual están los procesos, esto se maneja a través de queue's de planificación. Hay una queue

donde se encolan los procesos que están listos, otra donde están los de espera.

Las flechas que pasan de un estado a otro, lo que realmente reflejan es la migración de un proceso de un estado a otro se traduce en la migración de un proceso de una cola a otra.

En la siguiente imagen podemos apreciar las distintas colas(muestra solo 2 en realidad pero existen las demás) con los bloques de control de los procesos. Si un proceso está en ready y pasa a Running, entonces el PCB correspondiente a dicho proceso se encola en la cola de Running.

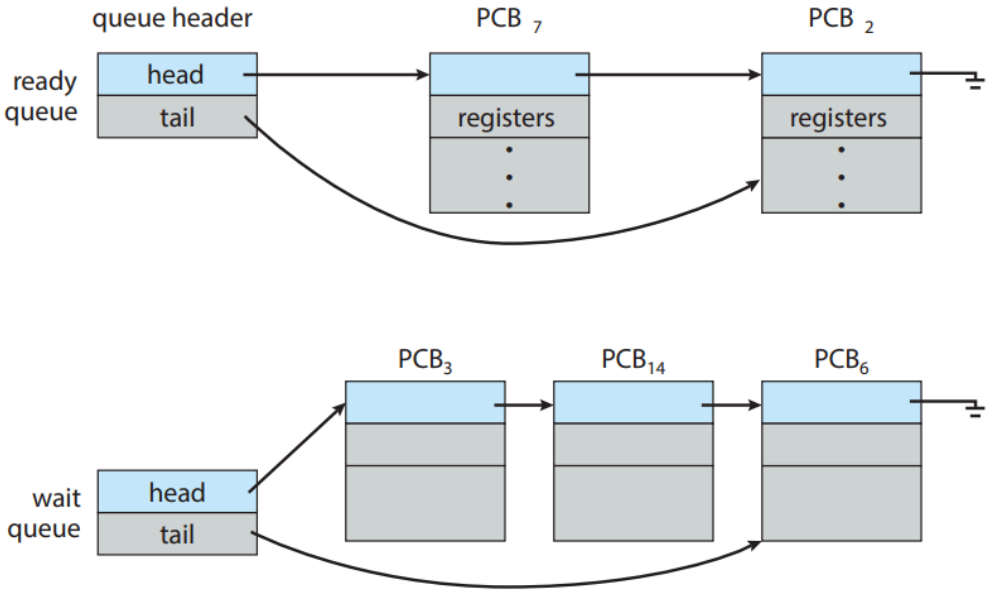


Figure 3.4 The ready queue and wait queues.

Diagrama de colas para la planificación de procesos

En este otro gráfico podemos ver que un proceso cuando nace entra en la cola de listos, cuando el Scheduler selecciona pasa a hacer uso de la CPU.

Si se solicita una operación de E/S pasa a la cola de espera (El S.O le gestiona la operación de E/S), cuando termina esa operación de E/S, vuelve a la cola de listos.

Otro caso es cuando se le termina el tiempo, el time slice del tiempo asignado expiró entonces se lo saca de Running vuelve a la cola de listo.

Los otros casos se profundizan más adelante pero básicamente si un proceso crea otro proceso, el proceso padre espera al que el proceso hijo muera y cuando este termina vuelve al

estado listo.

El último caso es cuando se espera por una interrupción y esta finalmente se produce, vuelve a la cola de listo

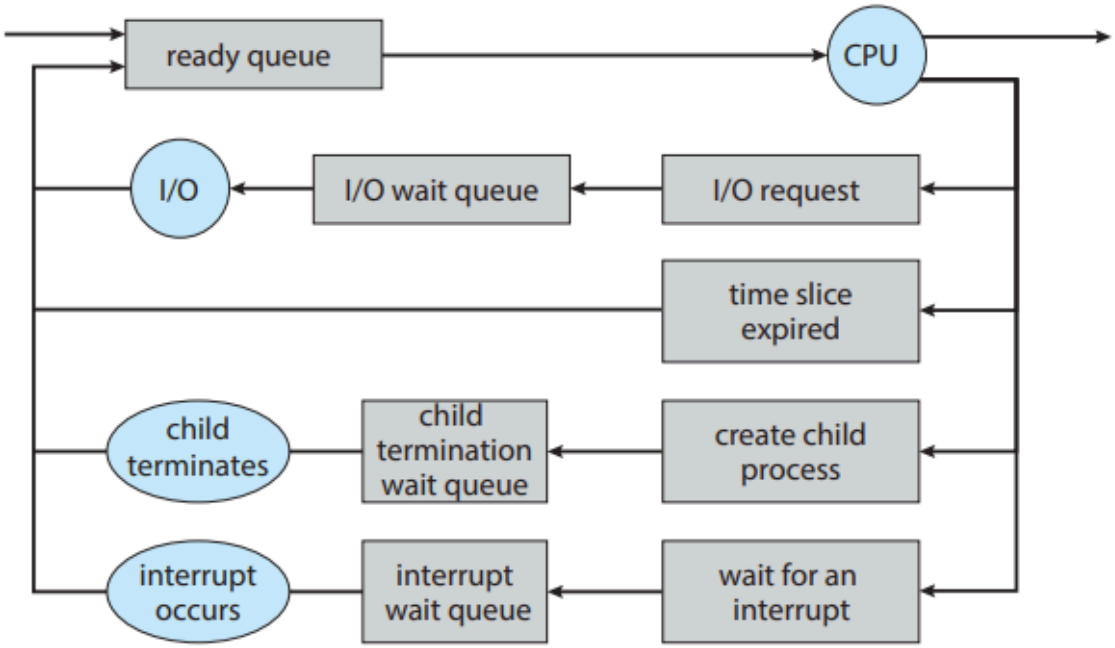


Figure 3.5 Queueing-diagram representation of process scheduling.

Cambio de contexto

Concepto de cambio de contexto

Se denomina cambio de contexto a la operación de la CPU para cambiar de un proceso a otro proceso. El sistema se encarga de sacar un proceso de ejecución, preservar su estado, cargar el estado de un nuevo proceso y ponerlo en ejecución. Es una operación súper crítica porque se realiza una infinidad de veces en muy corto tiempo.

El contexto de un proceso se guarda en el bloque de control de proceso (PCB) de tal manera que cuando se lo quiera volver a ejecutar se lo vuelve a cargar. El tiempo que se tarda entre cambio de proceso es un overhead, solo se realiza ese cambio.

Ejemplo de contexto

Un cambio de contexto ocurre cuando el procesador cambia de un proceso a otro.

En el gráfico podemos apreciar dos ejes donde tenemos la ejecución de dos procesos, P0 y P1. Se va realizando un cambio de contexto entre los dos procesos.

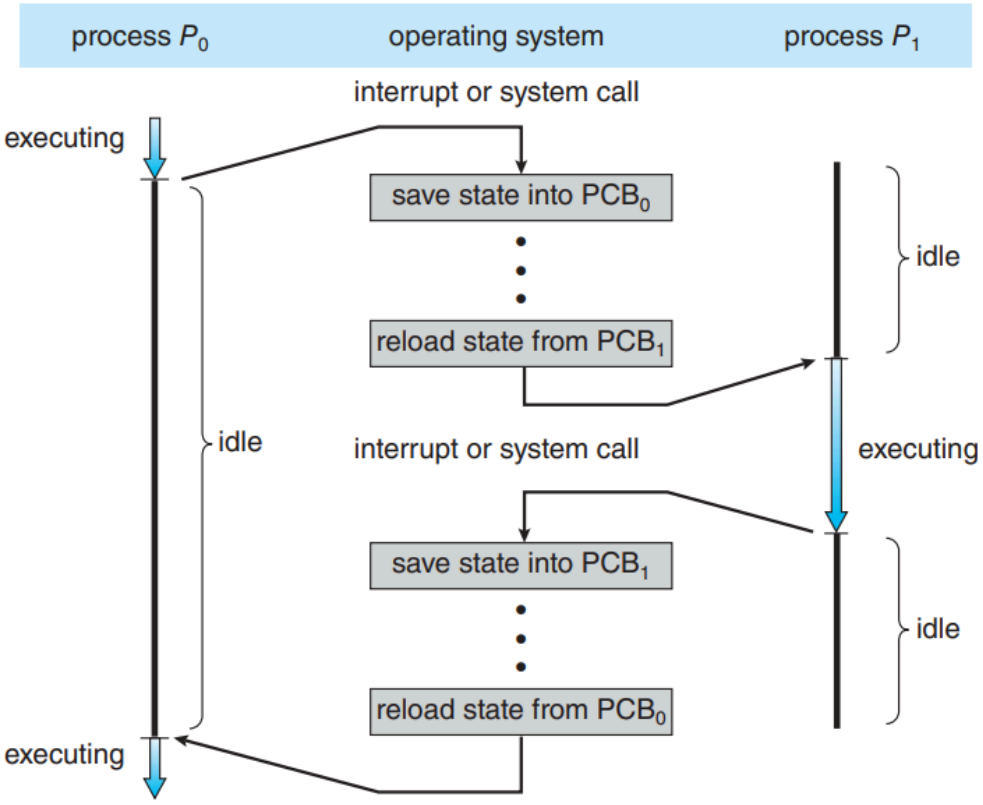


Figure 3.6 Diagram showing context switch from process to process.

Suponiendo que el Scheduler selecciona P0 para ejecutarlo, puede que este en algún momento solicite una operación E/S entonces realiza una llamada al sistema a través de una interrupción por software. Como P0 tiene que continuar después de que el S.O resuelva la llamada al sistema, el S.O toma control y preserva el estado del proceso guardándolo dentro del PCB de P0. Después mientras se realiza la operación de E/S, el S.O hace una recarga de PCB de P1 (lo pone en ejecución ya que estaba en estado ready).

El P1 se ejecuta hasta recibir una interrupción o llamada al sistema, entonces se guarda el estado en el PCB de P1, luego se recarga el estado del PCB de P0 y se retoma la ejecución de P0.

Operaciones sobre los procesos

En la mayoría de los sistemas, los procesos pueden ejecutarse de forma concurrente y pueden crearse y eliminarse dinámicamente. Por tanto, estos sistemas deben proporcionar un

mecanismo para la creación y terminación de procesos.

1. Creación de procesos

Durante el transcurso de una ejecución, un proceso puede crear otros procesos nuevos mientras se ejecuta; para ello se utiliza una llamada al sistema específica para la creación de procesos(fork()). El proceso creador se denomina **proceso padre** y los nuevos procesos son los **hijos** de dicho proceso. Cada uno de estos procesos nuevos puede a su vez crear otros proceso, dando lugar a un **árbol de procesos**.

La mayoría de los sistemas operativos identifican a los procesos mediante un identificador de proceso unívoco o pid (process identifier), que normalmente es un número entero. El pId nos da un valor único para cada proceso en el sistema, el cual puede ser utilizado como un índice para acceder a varios atributos del sistema dentro del kernel.

Ejemplo de árbol de procesos en Linux (jerarquía de procesos)

Luego de que el kernel se inicializa arranca el primer proceso del sistema (proceso de pid=1, llamado init y ahora systemd), este proceso no tiene padre ya fue generado automáticamente por el kernel. Este proceso crea otros proceso que son sus hijos y esos procesos hijos pueden seguir creando otros procesos, generando así una estructura de árbol.

La siguiente figura nos muestra un típico árbol de procesos en Linux, nos muestra el nombre de cada proceso junto a su pid (Linux se suele referir a los procesos como tareas o task).

El proceso *systemd* (que siempre tiene el pid 1) sirve como el proceso raíz o padre para todos los procesos de usuario, y es el primer proceso de usuario creado cuando el sistema arranca.

Una vez que el sistema ya arrancó, el proceso systemd crea nuevos procesos los cuales proveen servicios adicionales como un servidor web o de impresión(g u a r d a con mi traducción).

Muchos procesos en el mundo UNIX/Linux que corren en el background y brindan servicios, habitualmente conocidos como servicios, antes se llamaban demonios, son procesos que corren solos y desatendidos, sin interactuar con el usuario pero brindando un servicio. En la bibliografía se refieren a ellos como demons, de ahí vienen la d. también podemos identificarlos por los nombres que terminan en d

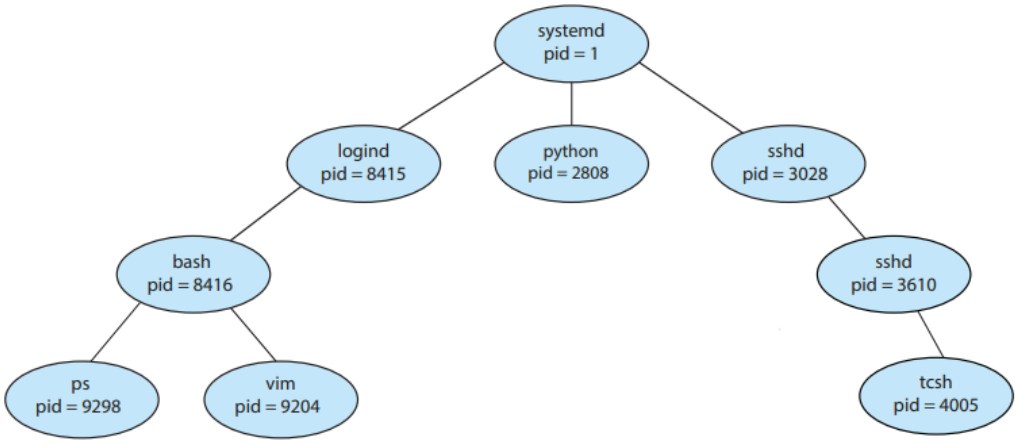


Figure 3.7 A tree of processes on a typical Linux system.

Los sistemas UNIX tradiciones tenían al proceso *init* como la raíz de todos los procesos, este tiene asignado un pid 1 y es el primer proceso creado cuando el sistema arranca.

En los sistemas UNIX y Linux, podemos obtener un listado de los proceso usando el comando `ps` . por ejemplo el comando: `ps -el` mostrará una lista completa de información de todos los procesos activos actualmente.

En general , cuando un proceso crea un proceso hijo, ese proceso hijo necesitará ciertos recursos (Tiempo de CPU, memoria, archivos, dispositivos E/S) para realizar su tarea. El proceso hijo puede obtener sus recursos directamente del sistema operativo, o puede estar restringido a un subconjunto recursos del proceso padre. El proceso padre puede tener que dividir sus recursos entre sus hijos, o puede compartir algunos recursos (como memoria o archivos) entre varios de sus hijos. Restringir a un proceso hijo a un subconjunto de los recursos del proceso padre previene que haya una sobrecarga del sistema a causa de crear demasiados procesos hijos.

Además de suministrar varios recurso físicos y lógicos, el proceso padre puede pasar datos de inicialización(inputs) al proceso hijo.

Cuando un nuevo proceso se crea, existen **dos posibilidades de ejecución**:

1. El proceso padre continúa su ejecución en simultaneó con sus hijos.
2. El proceso padre espera a que alguno o todos de sus hijos hayan terminado.

También hay **dos posibilidades de espacio de direcciones para el nuevo proceso**:

1. El proceso hijo es un duplicado del proceso padre (tiene el mismo programa y datos que el padre).
2. El proceso hijo tiene un nuevo programa cargado.

Proceso de creación utilizando fork() 🤯

El hijo no es más que un duplicado del padre y después el hijo, ejecuta una instrucción especial que es una llamada al sistema donde le pide al sistema operativo que busque un programa y lo cargue encima de su espacio de memoria. Entonces el proceso padre para crear un hijo ejecuta el system call **fork()** y esto crea un nuevo proceso. Notar en el siguiente gráfico que el proceso padre viene ejecutando hasta que encuentra una instrucción de tipo fork() y el resultado de esta función se le asigna a la variable pid (variable numérica que representa el número de proceso). Después de ejecutar el fork podemos ver como continua ejecutando el proceso padre que tiene un pid mayor a 0 y que además la función fork le devolvió la identificación del proceso hijo. Como resultado del fork al proceso padre le queda en la variable pid la identificación del proceso hijo.

Con respecto al proceso hijo, la función fork en el proceso hijo que tiene su propio espacio de direcciones, la función fork le devuelve el valor 0, por lo tanto, pid tiene el valor 0 dentro del proceso hijo, esta variable toma valores distintos según estemos ejecutando código del padre o código del hijo, entonces como pid es = 0, el proceso hijo ejecuta la función exec() la cual es un system call que reemplaza el espacio de memoria del proceso con un nuevo programa y a partir de ahí tanto proceso padre como hijo corren concurrentemente. Cuando el proceso hijo termina ejecuta un system call llamado exit()..

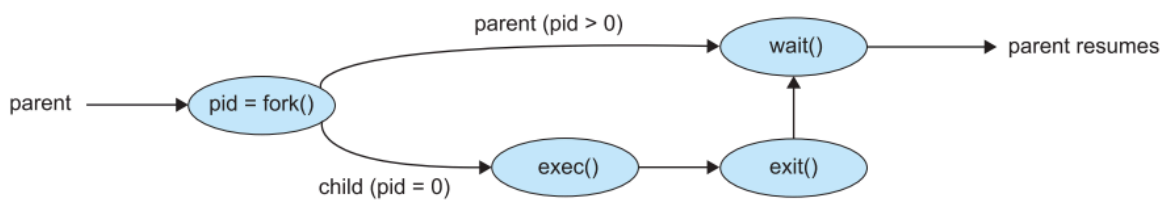


Figure 3.9 Process creation using the `fork()` system call.

Sobre el proceso de arranque

Cuando prendemos el equipo este hace un power on self test, básicamente sería un auto testeo para ver cómo están las componentes, y luego accede al disco de booteo, carga un programa a memoria en memoria al cual transfiere el control y ese programa es quien se encarga de cargar al sistema operativo.

El kernel no es un proceso, es un conjunto de instrucciones y estructuras de datos que se cargan en la parte baja de memoria y después de inicializarse internamente lanza procesos, particularmente esos procesos tienen un número que los identifica, entonces lanza un proceso con el proceso con el nombre de proceso número 1.

2. Terminación de procesos

Un proceso le puede avisar al sistema operativo que terminó, entonces ejecuta el system call exit().

Es posible que un proceso se cancele a sí mismo utilizando la llamada al sistema abort(), algunas razones por las que puede suceder esto son:

1. El proceso hijo excedió la cantidad de recursos asignados.
2. Las tareas asignadas al hijo no son requeridas.
3. El proceso padre se está yendo y el sistema operativo no permite a los procesos hijos continuar cuando su padres terminaron.

THREADS Y CONCURRENCIA

Introducción

El modelo de procesos mencionado anteriormente asume que un proceso era un programa en ejecución con un único hilo de control. Sin embargo, virtualmente todos los sistemas operativos brindan características que permiten que un proceso contenga múltiples hilos de control. Identificar oportunidades de paralelismo mediante el uso de subprocesos se está volviendo cada vez más importante para los sistemas multinúcleo modernos que proporcionan varias CPU’s.

En este capítulo, se introducen varios conceptos asociados a los sistemas de computación multihilos.

Hilos 🧵

Un hilo es una unidad básica de utilización de CPU, este contiene un Identificador de hilo, un contador de programa (PC), un conjunto de registro, y un Stack. Comparte con otros hilos pertenecientes al mismo proceso su sección de código, la sección de data, y otros recursos del sistema operativo, como archivos abiertos y señales.

> 💡 Si tomamos un proceso y vamos haciendo un seguimiento de cada instrucción que se va ejecutando una por una, vamos a ir formando una especie de línea o hilo (thread en inglés), entonces tengo un **hilo de ejecución**.

Un proceso tradicional tiene un único hilo de control. Si un proceso tiene múltiples hilos de control, puede realizar más de una tarea a la vez.

El siguiente gráfico muestra la diferencia entre un proceso de un único hilo y otro multihilo, notar que el proceso multihilo comparte la sección de code,data y files, pero cada uno tiene sus propios registros, stack y el PC.

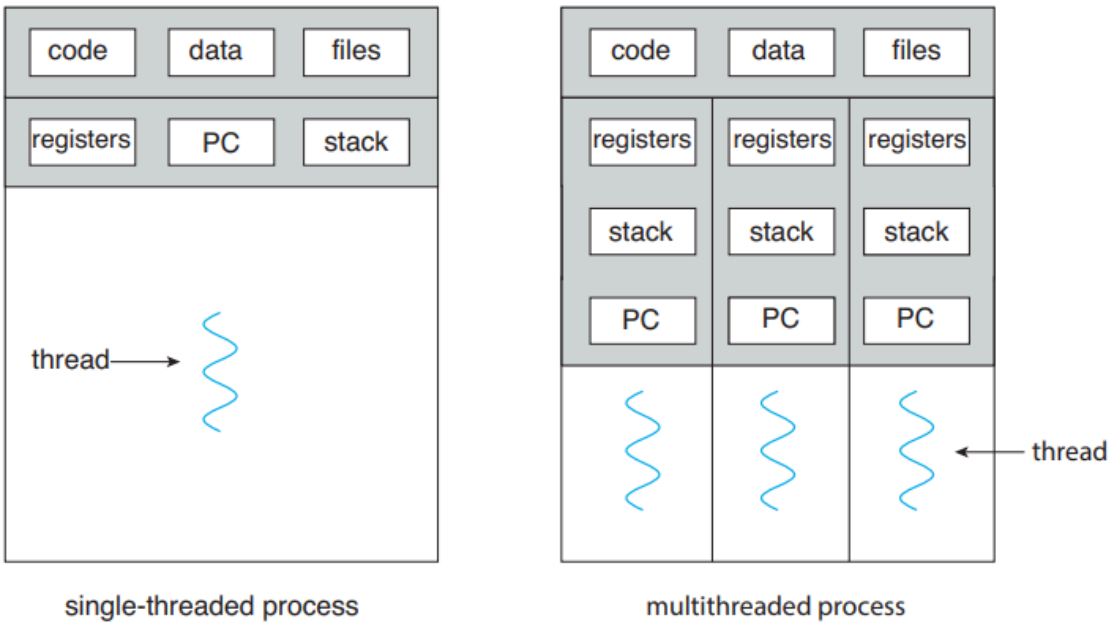


Figure 4.1 Single-threaded and multithreaded processes.

Muchas aplicaciones también pueden aprovechar el uso de múltiples hilos, incluidos algoritmos básicos de ordenamiento, árboles y gráficos. Además, los programadores que deben resolver problemas contemporáneos de uso intensivo de CPU en minería de datos, gráficos e inteligencia artificial pueden aprovechar el poder de los sistemas multinúcleo modernos al diseñar soluciones que se ejecutan en paralelo.

Sistemas mono hilos vs multihilos

Al principio los sistemas tenían procesos y estos tenían un único thread de ejecución. Este único hilo de control permite que el proceso realice solo una tarea a la vez. Por ejemplo, cuando un proceso está corriendo un procesador de texto, una instrucción de un único hilo se esta ejecutando ,esta permite controlar una única tarea al a vez, el usuario no podría tipear caracteres mientras utiliza el corrector de ortografía.

La mayoría de los sistemas operativos modernos han ampliado el concepto de proceso para permitir que
un proceso tenga múltiples hilos de ejecución y, por lo tanto, realizar
más de una tarea a la vez
. Esta característica es especialmente beneficiosa en sistemas multinúcleos , donde varios hilos pueden ejecutarse en paralelo. Un procesador multihilos podría, por ejemplo, asignar un hilo para administrar la entrada del usuario mientras
otro hilo ejecuta el corrector ortográfico. En los sistemas que admiten hilos, el PCB
se expande para incluir información para cada hilo.

Ventajas

Los beneficios de la programación multihilo se pueden mencionar en cuatro puntos:

1. **Capacidad de respuesta:** El uso de múltiples hilo en una aplicación interactiva permite que un programa continúe ejecutándose incluso aunque parte de él esté bloqueado si está corriendo una aplicación larga. Lo que aumenta la capacidad de respuesta al usuario. Por ejemplo un navegador web permite la interacción del usuario a través de un hilo mientras que en otro hilo se carga una imagen.
2. **Compartición de recursos:** Los hilos comparten memoria y los recursos del proceso al que pertenecen. La ventaja de compartir el código y los datos es que permite que una aplicación tenga varios hilos de actividad diferente del mismo espacio de direcciones.
3. **Economía:** La asignación de memoria y recursos para la creación de procesos es costosa. Dado que los hilos comparten recursos del proceso al que pertenecen, es más económico crear y realizar cambios de contexto entre unos y otros hilos. Puede ser difícil determinar

empíricamente la diferencia en la carga adicional de trabajo administrativo pero, en general, se consume mucho más tiempo en crear y gestionar los procesos que los hilos.

4. **Escalabilidad:** Las ventajas pueden verse incrementadas significativamente en una arquitectura multiprocesador, donde los hilos pueden ejecutarse en paralelo en los diferentes procesadores. Un proceso monohilo sólo se puede ejecutar en una CPU, independientemente de cuántas haya disponibles. Los mecanismos multihilos en una máquina con varias CPUs incrementan el grado de concurrencia.

Arquitectura cliente servidor

La arquitectura cliente servidor es una de las arquitecturas más utilizadas, donde se ejecuta una aplicación cliente que se conecta con un servidor y le esta le pide servicios.

Un ejemplo típico es un navegador como cliente y lo que hace al momento de que nosotros nos conectamos a internet y buscamos acceder a un sitio web, es establecer la conexión con un servidor web, el cliente es un web browser y el servidor es un web server. El servidor está esperando requerimiento del cliente, y ante cada requerimiento del cliente devuelve una respuesta.

Este modelo de trabajo se llama Request Response, el cliente es quien inicia la conversación y el servidor le envía un archivo HTML (en el caso de trabajar con páginas estáticas) la cual el cliente sabe cómo recibirla y componer dinámicamente la página porque sabe interpretar el código que la compone, la renderiza.

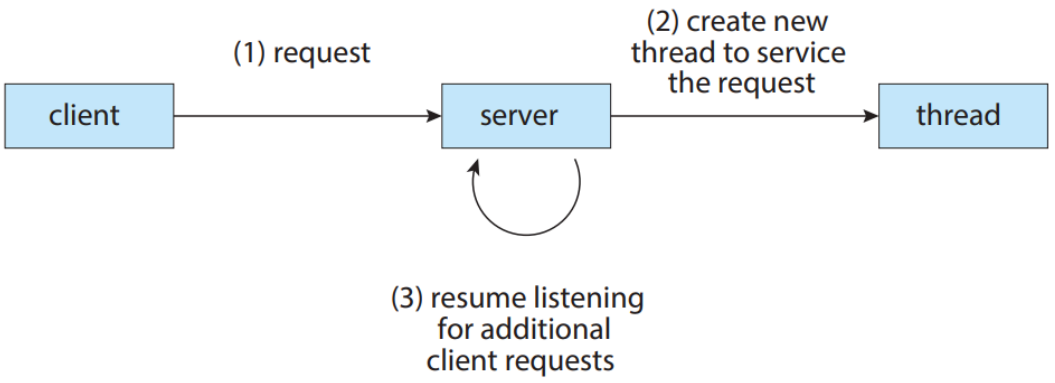


Figure 4.2 Multithreaded server architecture.

Ejemplo de multithreading con la arq cliente-servidor

Si en lugar de tener un único cliente tuvieramos 2, 3 o más, el cliente le pediría al servidor tal página, el servidor accede al archivo que contiene la página y se lo envía al cliente. Esto se denomina un file transfer, es decir una transferencia de archivo.

Si tengo otro cliente que hace un pedido de otra página, tendrá que esperar que el servidor le entregue la página al cliente 1 para atender al cliente 2, a nivel de comunicaciones, en caso de que esto fuera TCP/IP, se forma una cola de pedidos a la entrada del servidor, y el servidor tiene una forma de analizar estos pedidos que consiste en atenderlos de forma secuencial a medida que llegan. Cada hilo sería un ciclo, al finalizar la petición de un cliente, se atiende el siguiente. La atención es estrictamente secuencial porque manejamos un único hilo.

Suponiendo que el servidor es un gran servidor, que contiene mucha memoria, muchos CPUs, pero al mismo tiempo una cantidad muy grande de clientes. No existiría ningún tipo de ganancia ya que la lógica de atención de solicitudes no va a ser la mejor, vamos a tener una cola larga de peticiones y por más rápido que sea el servidor los requerimientos van a quedar encolados, Incluso puede pasar que la cantidad de requerimientos a encolar sea mayor al tamaño de la cola, esto se llama el backlog del servidor. En conclusión,como yo tengo procesos con un único thread no voy a aprovechar en nada que tenga muchos procesadores, porque lo único que necesita mi proceso por tener un único hilo es tener un solo procesador o un solo core.

Luego cuando se incorporó el trabajo multihilo, una vez que se recibía una petición, una función se encarga de resolver la petición, pero en lugar de llamar a la rutina con un call, le pide al SO que cree un nuevo hilo y el SO inicia la ejecución no por main sino por donde comienza la función ,para que la función pueda emitir la respuesta, y el hilo principal vuelve a revisar si hay más solicitudes, en caso de haberlas, puede crear otro nuevo hilo para atenderla. Esta operación es mucho menos costosa que crear nuevos procesos, todos los hilos comparten el mismo código, los mismos datos globales, pero cada uno tiene su propio stack para el retorno de subrutinas, o para guardar variables locales, lo único que se agrega por cada hilo es el área de stack.

De este modo, si tuviéramos muchos procesadores o cores, la planificación del procesador se trabaja por hilos en un entorno multihilos, y si por ejemplo tenemos 48 peticiones simultaneas atendidas por 48 hilos, cada hilo puede ejecutar por cada core, somo un procesador individual,se puede aprovechar la potencia de los múltiples procesadores, porque el proceso tiene múltiples hilos. Si tenemos N hilos de ejecución y N o más cores en ese procesador, esos N hilos de ejecución pueden ejecutarse en forma simultánea.

Programación multicore

En la programación multicore, al tener varios núcleos (cores) cuando tengo que ejecutar muchos procesos podemos asignarle un núcleo a cada uno de los procesos, si tengo 4 núcleos puedo tener 4 procesos en estado de running. Esto me ayuda a poder ejecutar más procesos por unidad de tiempo.

No ayuda en programas críticos y lentos tengan mejor performance, porque aunque tenga muchos procesadores mi único proceso no puede aprovechar todos los procesadores.

La programación multicore pone en presión a los programadores ya que ahora tiene el desafíos nuevos como:

- Dividir las actividades para que se puedan ejecutar en simultáneo.
- Balance: Balanceo a nivel de carga entre las distintas actividades
- Data splitting: Se dividen los datos para que las distintas subrutinas manejen sus datos
- Data dependency: Se analiza la dependencia de los datos, como por ejemplo, si un dato tiene que ser procesado antes que otro.
- Testing and debugging: El testeo y el debugging son tareas más difíciles en la programación multicore porque hay que seguir varios hilos que a su vez pueden estar en distintas etapas.

Paralelismo y concurrencia

Si yo tengo múltiples procesadores puedo tener paralelismo, el **paralelismo** implica que un sistema puede ejecutar más de una tarea de manera simultánea. Tengo 2,3,4 procesadores o cores que pueden estar ejecutando tantas actividades como cores tenga, esto se realiza en paralelo, si en un instante determinado detengo el tiempo, voy a ver que los distintos procesadores están ejecutando distintas instrucciones simultáneamente.

Ahora si tengo un solo procesador, yo podría tener la ilusión de que hay varias tareas ejecutando a la vez, pero sin embargo, cuando analizo en detalle, en un instante determinado ese procesador está ejecutando una única instrucción, pero como se da tan rápido, por ejemplo en un sistema interactivo con Round Robin, va a cambiar y ejecutar otro proceso ágilmente, yo como usuario final tengo la sensación de que está ejecutando en paralelo pero no lo está haciendo en realidad. En este caso se dice que está ejecutando de manera **concurrente**.

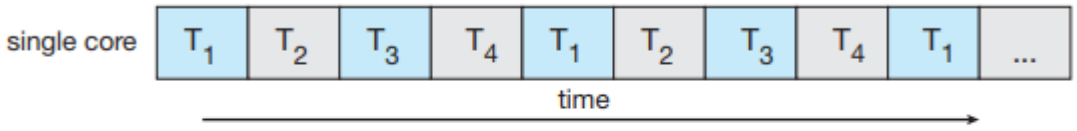


Figure 4.3 Concurrent execution on a single-core system.

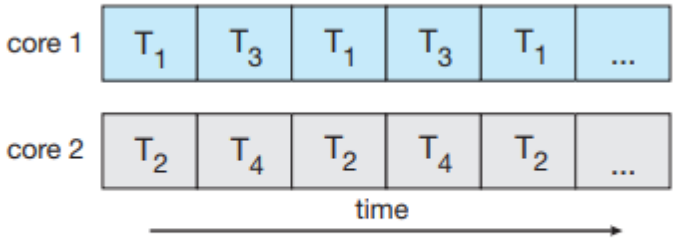


Figure 4.4 Parallel execution on a multicore system.

Paralelismo a nivel de datos y de tareas

Existen dos maneras de paralelismo:

1. Paralelismo a nivel de tarea (Task parallelism): Tengo muchas tareas o hilos para ejecutar y estos hilos ejecutan en distintos Cores, donde cada hilo realiza su tarea.
2. Paralelismo a nivel de datos (Data parallelism): Tomar esos datos, distribuir esos datos en distintos cores y hacer una operación.

Hardware threads

Cada core de una CPU, puede además de soportar hilos a nivel de software, soportar hilos a nivel de hardware, esto significa que cada procesador tiene una N cantidad de cores y a su vez cada core soporta hilos de hardware (hardware threads), entonces por ejemplo, un procesador con 8 cores y 8 hilos por hardware. Esto desde el punto de vista del sistema operativos se vería como si tuviera 64 procesadores, desde el punto del hardware tendría 8 cores y cada core soporta 8 hilos

---

Modelos multi hilos

Desde el punto de vista práctico, el soporte para hilos puede proporcionarse en el nivel de usuario (para los hilos de usuario) o por parte del kernel (para los hilos del kernel). El soporte para los hilos de usuario se proporciona por encima del kernel y los hilos se gestionan sin soporte del mismo, mientras que el SO soporta y gestiona directamente los hilos del kernel.

Los hilos se pueden implementar a dos niveles:

1. Threads de usuario
2. Threads a nivel de kernel

y también existen varias maneras de implementar los hilos:

1. Modelo Muchos a uno

Muchos threads a nivel se usuario que se transforman a un solo thread a nivel de kernel.

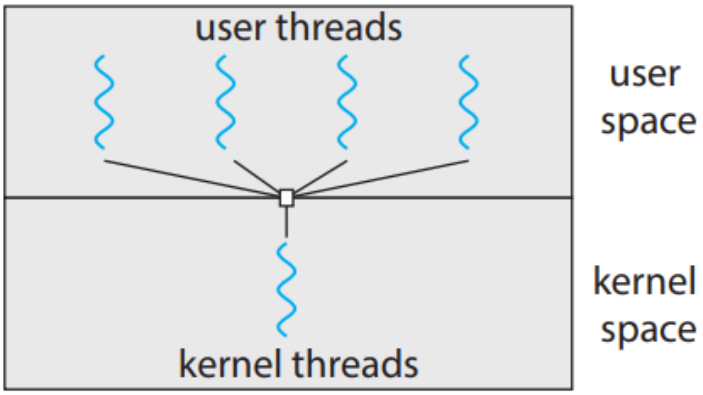


Figure 4.7 Many-to-one model.

2. Modelo Uno a uno

Un thread a nivel de usuario que cuando piden un servicio al SO se transforman en un solo thread a nivel de kernel.

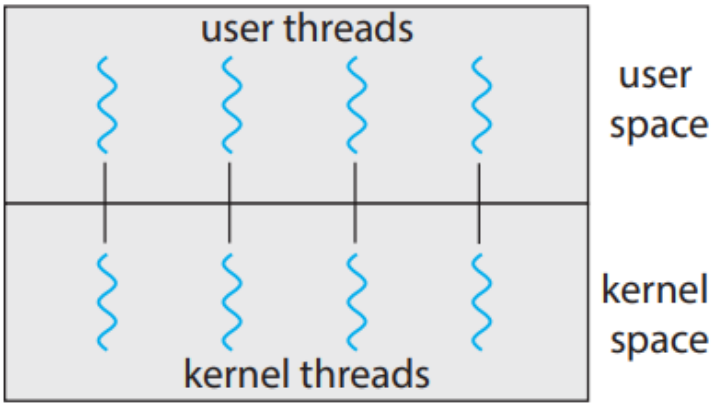


Figure 4.8 One-to-one model.

3. Modelo muchos a muchos

Cualquier thread a nivel de usuario cuando pide un servicio al SO, ese servicio ejecuta en un thread especifico.

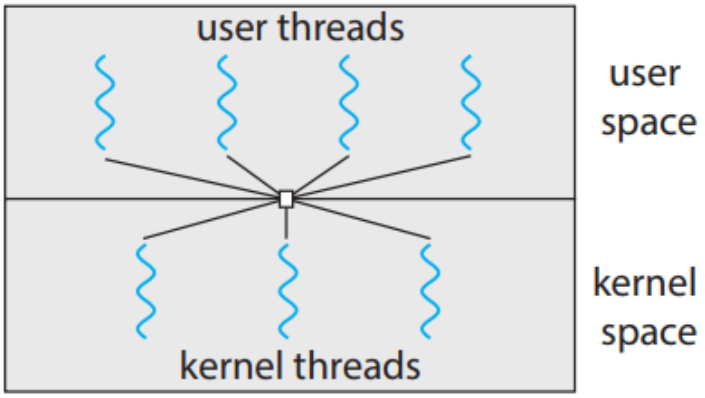


Figure 4.9 Many-to-many model.

Biblioteca de hilos

Una biblioteca de hilos le otorga al programador una API para crear y manejar hilos. Hay dos formas primarias de implementar una biblioteca de hilos.

El primer método consiste en proporcionar una biblioteca enteramente en el espacio de usuario, sin ningún soporte del kernel. Todas las estructuras de datos y el código de la biblioteca se encuentra en el espacio de usuario. Esto significa que invocar una función de la biblioteca es como realizar una llamada a una función local en el espacio de usuario y no una llamada al sistema.

El segundo método consiste en implementar una biblioteca en el nivel del kernel, soportada directamente por el sistema operativo. En este caso, el código y las estructuras de datos de la biblioteca se encuentra en el espacio del kernel. Invocar una función en la API de la biblioteca normalmente da lugar a que se produzca una llamada al sistema dirigida al kernel.

POSIX threads (Pthreads)

POSIX significa portable operating system y la IX viene de UNIX. Es un estándar desarrollado por la IEEE. Cuando hablando de POSIX threads nos referimos a una implementación de los hilos para el POSIX, un estándar para el desarrollo de threads básicamente.

Los **Phtreads** se basan en el estándar POSIX que define una API para la creación y sincronización de los hilos. Se trata de una especificación para el comportamiento de los hilos, no de una implementación.

PLANIFICACION DE LA CPU

Conceptos básicos

La planificación del procesador es la base de los sistemas operativos. Cuando cambiamos el CPU entre los procesos, el sistema operativo puede hacer más productiva a la computadora. En este capítulo se introduce los conceptos básicos de planificación del CPU y se presentan varios algoritmos de planificación del CPU.

Mono programación vs multiprogramación

Al principio de la computación se usaba una técnica secuencial para ejecutar programas llamada **mono programación**, en donde se iban ejecutando los procesos a medida que terminaban.

Teniendo en cuenta que en un sistema de un procesador un solo núcleo, un solo proceso puede ejecutarse a la vez. Los otros procesos deben esperar hasta que el núcleo de la CPU esté libre y pueda ser reasignado. Desde el punto de vista de la CPU, esta era utilizada durante la ejecución y cuando había una operación de E/S la CPU se queda a la espera, y teniendo en

cuenta que las operaciones de entrada y salida son más largas que las de cálculo, la CPU se pasaba la mayoría del tiempo inactiva. Todo ese tiempo de espera es desperdiciado.

Luego aparece la **multiprogramación**, esta técnica tiene el objetivo de aprovechar los tiempos de espera de la CPU que se desperdician en la mono programación. Para lograr esto varios procesos son guardado en memoria de una vez. Cuando un proceso tiene que esperar, el sistema operativo le quita la CPU al proceso que tiene que esperar y se lo da a otro proceso. Este patrón continúa. Cada vez que un proceso tiene que esperar, otro proceso puede hacer uso de la CPU. En un sistema multinúcleo (multicore), este concepto de mantener a la CPU ocupada se extiende a todos los núcleos de procesamiento en el sistema.

Podemos decir que la multiprogramación consiste en tener varios programas “arrancados” ya en ejecución, no ejecutando en forma simultánea si es que tenemos un solo procesador porque no sería posible, pero desde el punto de vista del usuario estos programas están ejecutando a la vez. Esa ejecución aparente de simultaneidad se llamada **concurrencia**.

Planificar esto es una parte fundamental de las funciones del sistema operativo. Casi todos los recursos de la computadora son planificados antes de utilizarse. El CPU es el principal y por eso su planificación es fundamental para el diseño de un sistema operativo.

Ciclos de ráfagas

Un proceso ejecuta instrucciones y en algún momento puede solicitar una operación de E/S, después de terminar esa operación se le devuelve el procesador y continua su ejecución. Cuando ejecuta instrucciones, está utilizando el procesador y se dice que está ejecutando una ráfaga de CPU (**CPU Burst**), cuando solicita una operación e/s estaría ejecutando una rafaga de entrada y salida (**I/O Burst**). Los proceso alternan entre los estados de ráfagas de CPU y ráfagas de E/S, hasta alcanzar el final donde termina la ejecución.

esta se grafica con una línea punteada indicando que se maneja por la controladora y no participa la CPU.

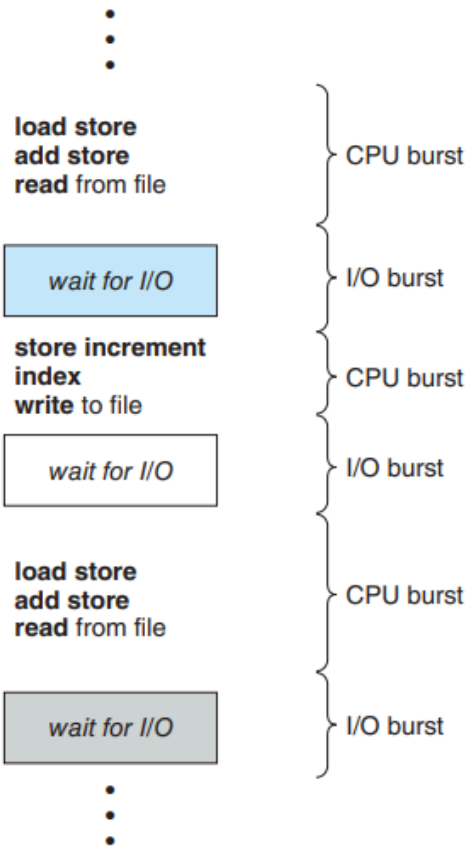


Figure 5.1 Alternating sequence of CPU and I/O bursts.

Planificador de la CPU (Scheduler)

La planificación del procesador es lo que me permite manejar la manera en que se asigna la CPU entre los distintos procesos ya que nos vamos a encontrar en un entorno de multiprogramación en donde mientras algunos procesos están esperando que termine la operación e/s, yo podría adelantar y empezar a ejecutar otros. Gracias a la planificación del procesador logramos ocuparnos de esto con el fin de alcanzar la máxima utilización de la CPU mediante la multiprogramación.

Para esto el sistema operativo tiene un módulo llamado **CPU Scheduler** (planificador de la CPU), este se encarga de que cuando la CPU se encuentra inactiva, seleccionar uno de los procesos que se encuentra en la cola de procesos listos para ejecutar.

Es un módulo que se ejecuta muy frecuentemente, pensar que cada vez que el SO decide analizar la cola de procesos listos y poner a uno en ejecución, el Scheduler es llamado para encargarse.

💡 La cola de procesos listos no necesariamente es una implementación FIFO, puede también ser una cola de prioridad, un árbol, o una linkedlist desordenada. El contenido en ellas generalmente son los PCBs o bloques de control de proceso.

Planificación con y sin desalojo

Las decisiones del planificación de procesos toman lugar e invocan al Scheduler cuando un proceso:

1. Cambia del estado Running a Waiting. (Solicita E/S)
2. Cambiada del estado Running a Ready. (Por el timer)
3. Cambian del estado Waiting a Ready. (Termina la operación E/S)
4. Termina.

Para los casos 1 y 4 no hay elección en términos de planificación. Un nuevo proceso se debe seleccionar para que se ejecute. En cambio, para las situaciones 2 y 3 si puede haber una elección.

Cuando la planificación toma lugar bajo las circunstancias 1 y 4, decimos que el esquema de planificación es non-preemptive (sin desalojo) . Sino, es preemptive (Con desalojo).

La planificación con desalojo (preemptive) significa que se puede desalojar o quitar de ejecución a un proceso para poner en ejecución a otro proceso que esté listo que tenga mayor prioridad.

La planificación sin desalojo (non-preemptive) la única forma que ese proceso deje el procesador es porque lo hice de manera voluntaria, terminando o solicitando una operación de entrada salida. No lo desalojaron o echaron.

Módulo Dispatcher

Otro componente involucrado en la planificación de la CPU es el Dispatcher. El **Dispatcher** es un módulo del sistema operativo que le da el control del núcleo de la CPU al proceso seleccionado por el Dispatcher. En otras palabras, lo pone en ejecución.

Las funciones que realiza son:

1. Cambiar el contexto de un proceso a otro.
2. Cambiar el modo de usuario.
3. Saltar a la ubicación adecuada en el programa de usuario para continuar la ejecución de dicho programa.

El Dispatcher tiene que ser tan rápido como sea posible, ya que es invocado durante cada cambio de contexto. El tiempo que le toma al Dispatcher parar el proceso y empezar otra ejecución es conocido como **dispatch latency** (latencia del Dispatcher).

Como esta es una tarea administrativa implementada por el S.O, debería ser lo más eficiente posible.

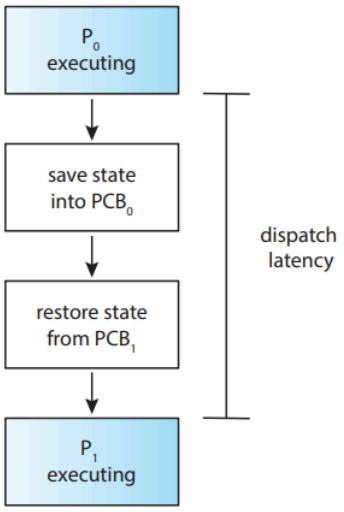


Figure 5.3 The role of the dispatcher.

La latencia de despacho es el tiempo que le lleva entre detener a un proceso salvando su estado y arrancar el otro proceso cargando su estado.

Criterio del Scheduler

Quien programa el Scheduler tiene que optimizar los siguientes puntos:

- **Utilización de la CPU:** Se busca mantener al CPU tan ocupado como sea posible. Se mide con porcentajes.
- El **Throughput** es una medida que indica la cantidad de procesos que completan su ejecución por unidad de tiempo. Es como una medida de qué tan rápido se procesan los

procesos. Se desea que esta sea tan alta como sea posible.

- Otra medida importante es el **Turnaround time** o tiempo de retorno, indica el tiempo que pasa desde que entra en ejecución un programa hasta que ese proceso termina.
- Otro tiempo interesante para evaluar es el **Waiting time** o tiempo de espera, este es la cantidad de tiempo que un proceso pasa en la cola de listo, en el estado de Ready.
- Por último está el **Response time** o tiempo de respuesta, este es el tiempo que pasa desde que el proceso hace un requerimiento al sistema y este responde. (Corta con la primera respuesta, no cuando termina toda la respuesta).

Por ende, los criterios a considerarse son:

1. Maximizar el uso de la CPU.
2. Maximizar el Throughput
3. Minimizar el Turnaround time
4. Minimizar el Waiting time
5. Minimizar el response time

Algoritmos de planificación

Para lograr aquellos criterios propuestos por el Scheduler existen distintos algoritmos, los cuales se fueron apareciendo en el mundo de la computación a medida que esta fue evolucionando.

1. FCFS (First-Come, First-Served) Scheduling

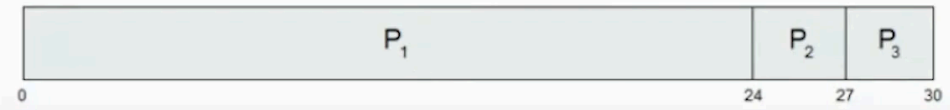
El primero que llega es el primero que se sirve, a veces se lo menciona como FIFO.

Vamos a tener una cola de procesos listos, esta tiene un orden cronológico de llegada. Este algoritmo toma al primer algoritmo que llegó y a este le asigna el procesador. Cuando termina el proceso 1 pone en ejecución al proceso 2 y así sucesivamente.

Analizando el tiempo de espera se puede determinar que tan bueno es el algoritmo, se busca minimizarlo lo más posible, como todos los procesos tienen un tiempo de espera distinto se realiza un promedio.

Process	Burst Time
P_1	24
P_2	3
P_3	3

- Suppose that the processes arrive in the order: P_1 , P_2 , P_3
  The Gantt Chart for the schedule is:



- Waiting time for P_1 = 0; P_2 = 24; P_3 = 27
- Average waiting time: (0 + 24 + 27)/3 = 17

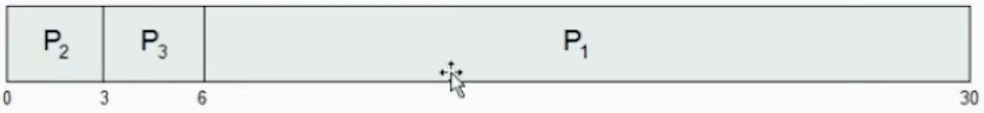
Ese tiempo de espera es muy dependiente de el orden de llegada de los procesos, a pesar de que lleguen los mismos procesos, dependiendo de como lleguen se puede obtener un promedio más alto o más bajo.

Notar que en el siguiente ejemplo, cambia el orden de llegada de los procesos y el tiempo promedio pasa de ser 17 a ser 3.

Suppose that the processes arrive in the order:

P_2 , P_3 , P_1

- The Gantt chart for the schedule is:



- Waiting time for P_1 = 6; P_2 = 0, P_3 = 3
- Average waiting time: (6 + 0 + 3)/3 = 3
- Much better than previous case
- **Convoy effect** - short process behind long process
  - Consider one CPU-bound and many I/O-bound processes

Efecto convoy

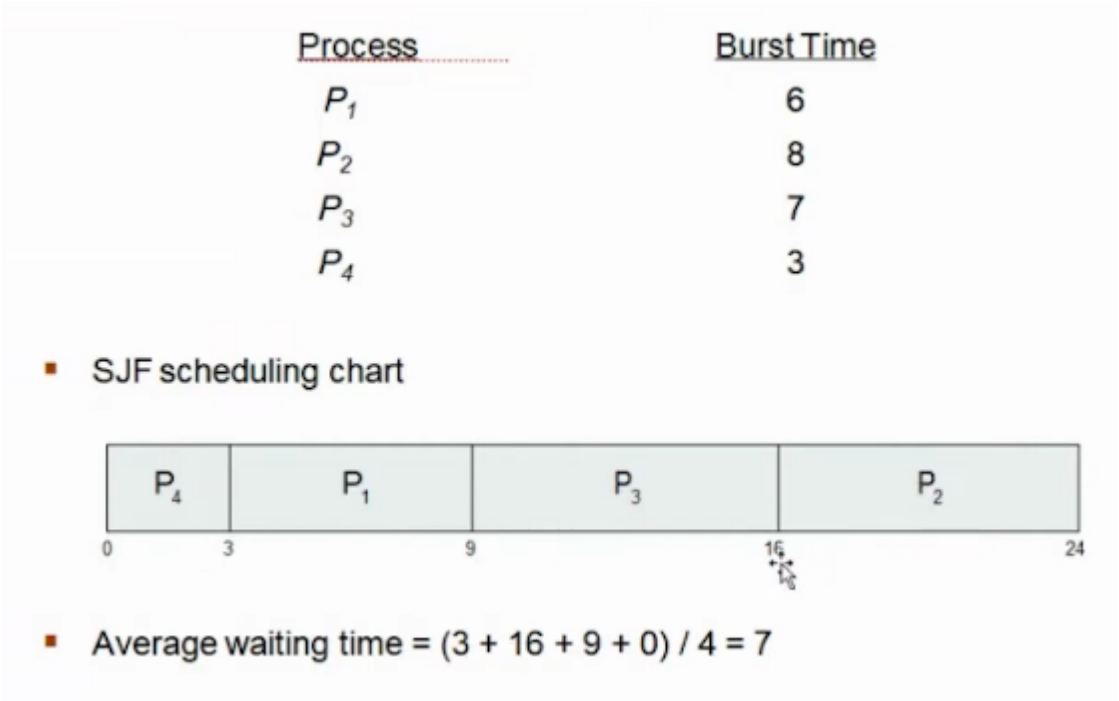
El efecto convoy hace referencia a cuando los procesos cortos llegan detrás de los procesos largos, van a tener que esperar al finalización de los procesos largos. En cambio si entran los procesos más largos al final, esperan un menor cantidad de tiempo y luego pueden ejecutarse sin la presión de tener un proceso de poco tiempo a la espera.

El algoritmo de primero el más corto es una optimización del algoritmo FCFS, donde primero se van a ejecutar los procesos más cortos, dejando al final lo más largos.

Una analogía podría ser una fila de supermercado donde un carrito repleto está por delante de alguien que solo lleva un producto.

2. SJS (Shortest Job First) Scheduling

El Scheduler para seleccionar el próximo proceso a ejecutar lo que hace es revisar la información de Scheduling, la cual en el algoritmo FCFS solo era la fecha y hora de entrada, ahora en el SJS lo que se evalúa es la duración de la ráfaga de ejecución.



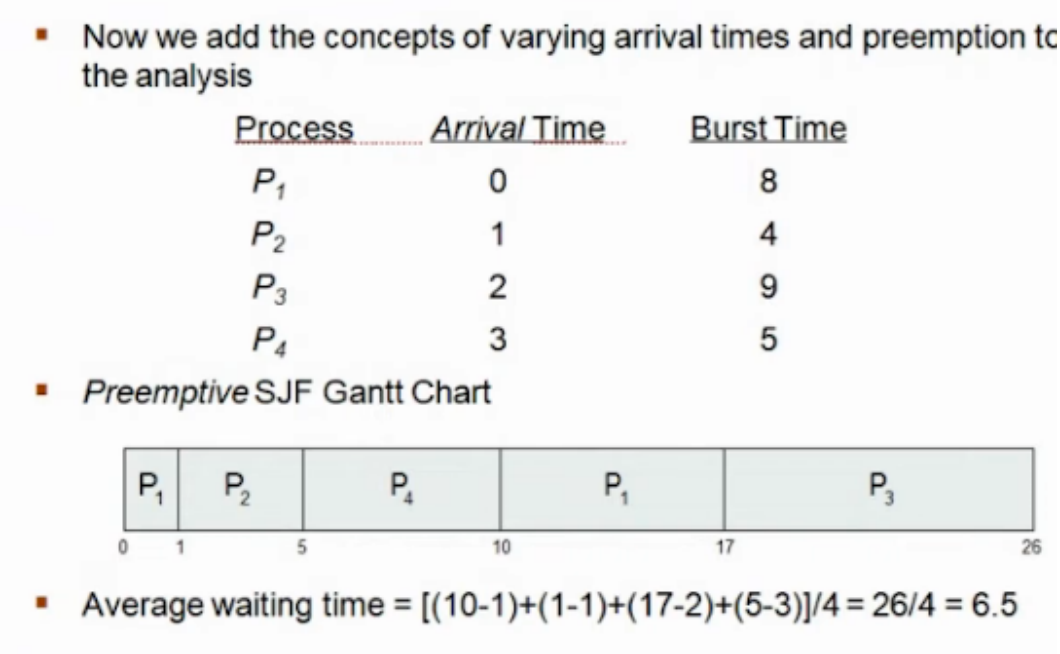
El sistema operativo no tiene manera de saber cuánto va a tardar un proceso, es imposible saber que va a pasar a futuro, el sistema operativo lo que hace es analizar el valor de las ráfagas anterior y estimar el valor de la próxima. Más precisamente, el cálculo se realiza haciendo un promedio exponencial, este es hacer un promedio de todas las ráfagas ponderando el valor según su antigüedad, las más recientes son más importantes.

3. STR (Shortest Remaining time first)

El algoritmo STR, el que menor tiempo restante corre primero, es una variante del SJS.

Este algoritmo tiene en cuenta en tiempo restante de los procesos, por ejemplo, si un proceso de 100 ejecutó 90, ahora tiene un tiempo restante de 10 y no debería tratarlo como si todavía

tiene 100.



Computación interactiva

Los algoritmos mencionados anteriormente eran los más utilizados al comienzo de la computación, los sistemas operativos soportaban multiprogramación pero la computación era totalmente Bash, la ejecución se hacía desde un lugar central con un operador, el cual ponía a ejecutar programas que el solicitaban.

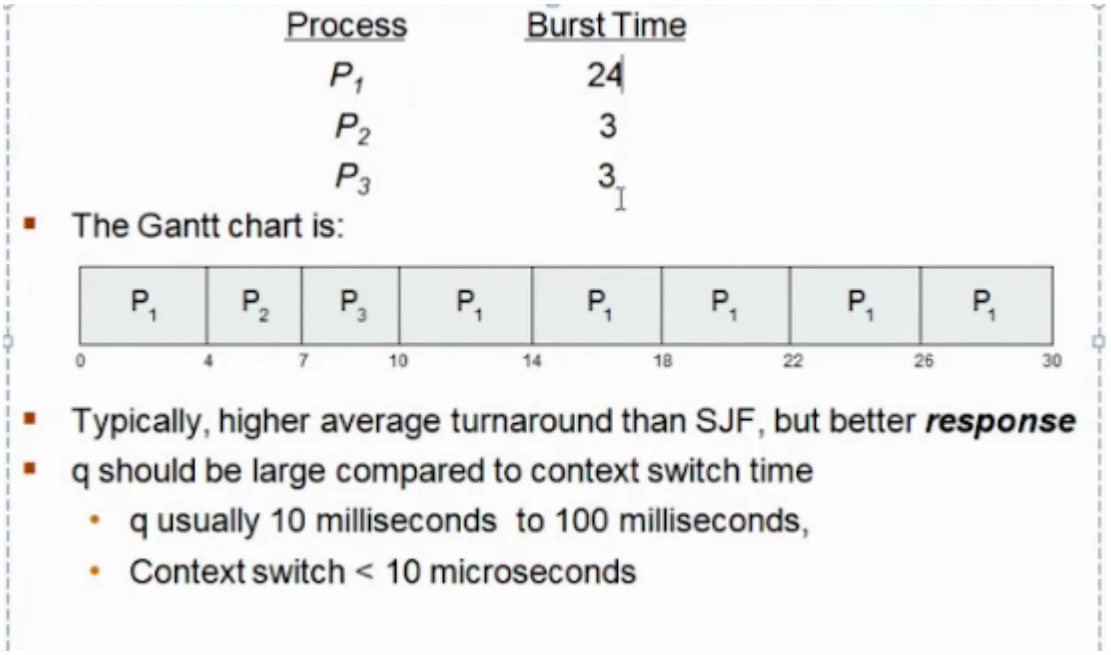
Más adelante, aparecieron las terminales pero fuera de los centros de cómputo, estaban físicamente en la computadora de los usuarios, las cuales luego de loguearse podían ejecutar comandos, así nació la computadora interactiva, y los algoritmos vistos anteriormente dejan de funcionar. Al usuario que está frente a la computadora hay que crearle la ilusión de que está ejecutando en todo momento la información, por eso hay que darle tiempo de procesador cada tanto de manera regular al proceso que corre en su terminal.

4. Round Robin (RR)

El algoritmo Round Robin busca distribuir el tiempo de procesamiento de manera uniforme entre todos los procesos, entonces cada proceso obtiene una pequeña cantidad de tiempo de CPU llamada time **quantum** y es un valor que oscila entre los 10-100 milisegundos.

El Scheduler selecciona un proceso y el Dispatcher lo pone en ejecución, pero el Dispatcher al momento de ponerlo en ejecución activa un temporizador (timer) y una vez finalizado ese tiempo de ejecución, se genera una interrupción y el sistema operativo toma el control, luego desaloja al proceso, lo lleva a la cola de listos y selecciona el siguiente proceso para ejecutar.

💡 El algoritmo RR es similar al FCFS pero se añade desalojo (preemption) para permitir que el sistema cambie entre los procesos.



El siguiente proceso a ejecutar es simplemente el siguiente que se encuentre en la cola de listos.

5. Algoritmo de planificación basado en prioridades

El algoritmo basado en prioridades trabaja asignándole un número entero a cada proceso, entonces en el bloque de control de procesos (PCB) vamos a encontrar este número que indica la prioridad del proceso. En general, cuando ese número es más chico la prioridad es mayor.

Este algoritmo se puede implementar con o sin desalojo. Si tuviera desalojo entonces puede suceder que un proceso con mayor prioridad se añada a la cola de listos y entonces se desaloja al que está en ejecución para poner al de mayor prioridad.

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

- Priority scheduling Gantt Chart

- Average waiting time = 8.2

Starvation y Aging

Este algoritmo presenta un problema, también aparece en otros pero acá con mayor fuerza, que se llama **Starvation** (hambruna). Cuando un proceso tiene baja prioridad no se le va asignar procesador y puede suceder que pase su vida en la cola de listo, puede que de casualidad entre en ejecución porque todos los procesos de mayor prioridad entraron en espera pero apenas entre otro proceso se lo vuelve a desalojar. Entonces se dice que el proceso muere de hambre porque nunca llega a consumir el procesador.

La solución implementada por el sistema operativo es una técnica llamada **Aging** o envejecimiento, que consiste básicamente en determinar cuales procesos están hace mucho tiempo en la cola de listos y aumentarle su prioridad.

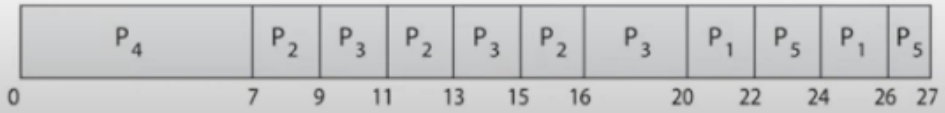
5. Algoritmo de prioridades con RR

Esta es una variante del algoritmo de prioridades en donde tenemos procesos con la misma prioridad pero esos procesos se ejecutan con Round Robin.

Notar en el gráfico que los proceso que tienen la misma prioridad, utilizan RR.

Process	Burst Time	Priority
P_1	4	3
P_2	5	2
P_3	8	2
P_4	7	1
P_5	3	3

- Run the process with the highest priority. Processes with the same priority run round-robin
- Gantt Chart with time quantum = 2



6. Multilevel Queue (Colas multinivel)

Este algoritmo en lugar de trabajar con una única cola de procesos listos, traba con N colas de listos. Entonces esas colas manejan una prioridad distinta y el Scheduling se hace partiendo de la cola que tiene mejor prioridad.

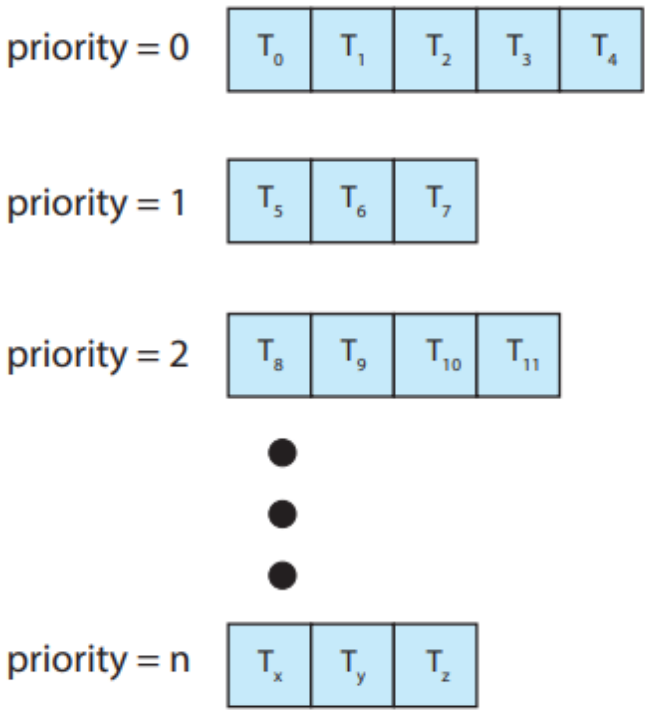


Figure 5.7 Separate queues for each priority.

7. Multilevel Feeback Queue (Colas multinivel con feedback)

Este algoritmo de planificación es una variante del anterior, se implementan varias colas con distintos niveles de prioridad y utilizando otro algoritmo al mismo tiempo.

Una implementación puede ser la siguiente:

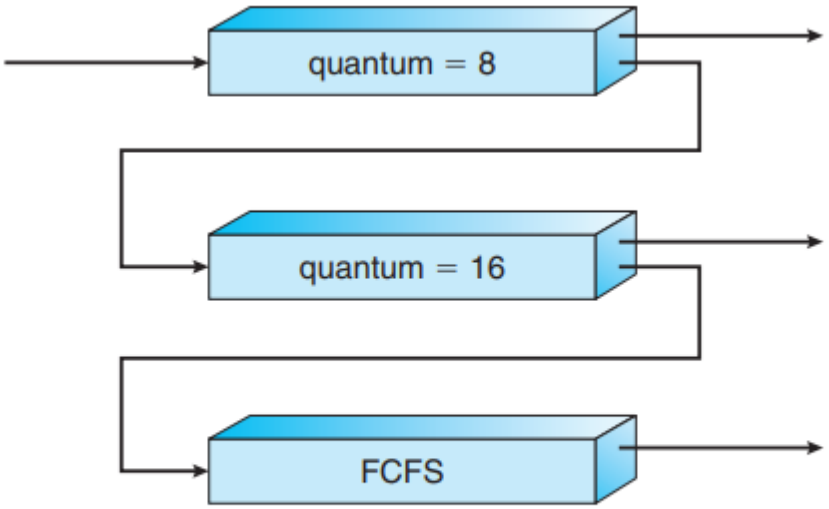


Figure 5.9 Multilevel feedback queues.

Estas colas pueden manejar RR con un quantum en distintos valores por cada cola y por último una cola con el algoritmo FCFS.

Este algoritmo también tiene que ir determinando cuales procesos son interactivos y cuales no para tratarlos con Round Robin.

También acá se puede implementar la técnica de Aging para lograr que un proceso pueda asegurarse su ejecución.

MEMORIA VIRTUAL

1. Monoprogramación:

En los albores de la computación, los sistemas operativos empleaban la monoprogramación, un modelo simple donde la memoria se dividía en dos secciones: una reservada para el sistema operativo y la otra para el único proceso en ejecución. Este esquema, si bien sencillo, presentaba una desventaja crucial: la **ineficiencia en el uso de la CPU**. Al depender de un

solo proceso, la CPU permanecía inactiva durante las operaciones de entrada/salida, desperdiciando valiosos ciclos de procesamiento.

2. Multiprogramación y Particiones de Memoria:

Para optimizar el uso de recursos, especialmente el tiempo de CPU, se implementó la **multiprogramación**, permitiendo la ejecución concurrente de múltiples procesos. Esta innovación requirió una estrategia más elaborada de administración de memoria: la **partición**. La memoria se dividía en múltiples secciones, llamadas particiones, cada una destinada a alojar un proceso.

2.1. Particiones Fijas:

En este esquema, el tamaño de las particiones se establecía durante la instalación del sistema operativo, basándose en las necesidades previstas de los programas a ejecutar. Esta rigidez en el tamaño de las particiones generaba **fragmentación interna**, un problema que implicaba el desperdicio de memoria dentro de las particiones. Un proceso que no ocupaba completamente la partición asignada dejaba un espacio inutilizable para otros procesos.

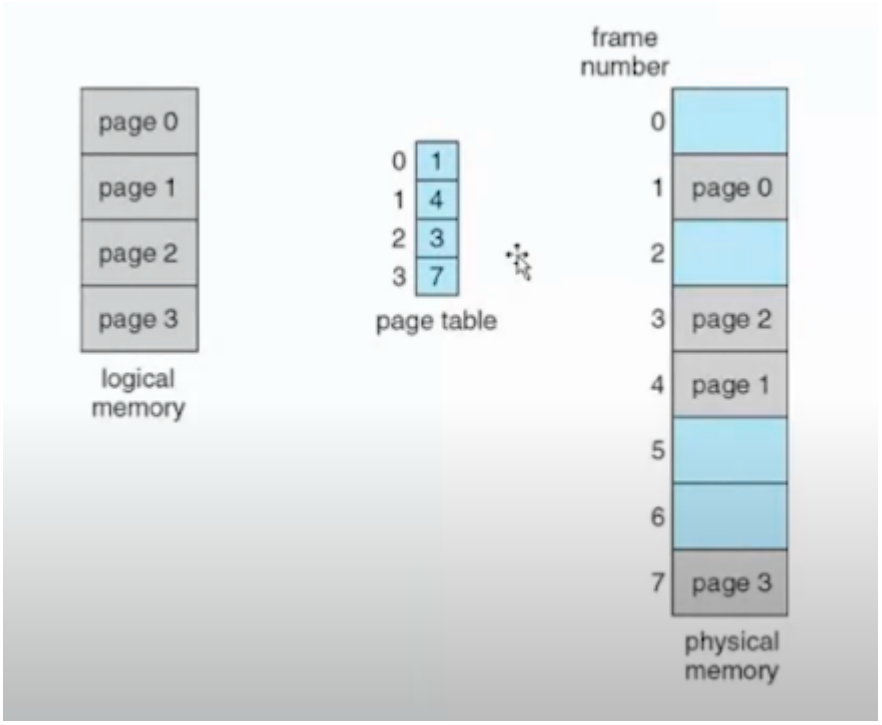
2.2. Particiones Variables:

Para mitigar la fragmentación interna, se introdujeron las particiones variables, donde el tamaño de la partición se ajustaba dinámicamente al tamaño del proceso en el momento de su carga. Si bien eliminaba la fragmentación interna, este esquema podía dar lugar a **fragmentación externa**, un problema que surge cuando el espacio libre disponible en memoria no es contiguo.

2.2.1. Soluciones para la Fragmentación Externa:

- **Compactación:** Este proceso reubica los procesos en memoria, moviéndolos a posiciones contiguas para generar un espacio libre unificado. La compactación, si bien efectiva, podía ser una tarea costosa en términos de tiempo de procesamiento.
- **DMA y Reubicación Dinámica:** Para agilizar la compactación, se delegó el movimiento de datos a las controladoras de entrada/salida (DMA). El DMA, con acceso directo a la memoria, podía transferir los procesos a disco y luego reubicarlos en memoria de manera más eficiente, liberando a la CPU de esta tarea.

3. Paginación: Un Esquema Moderno



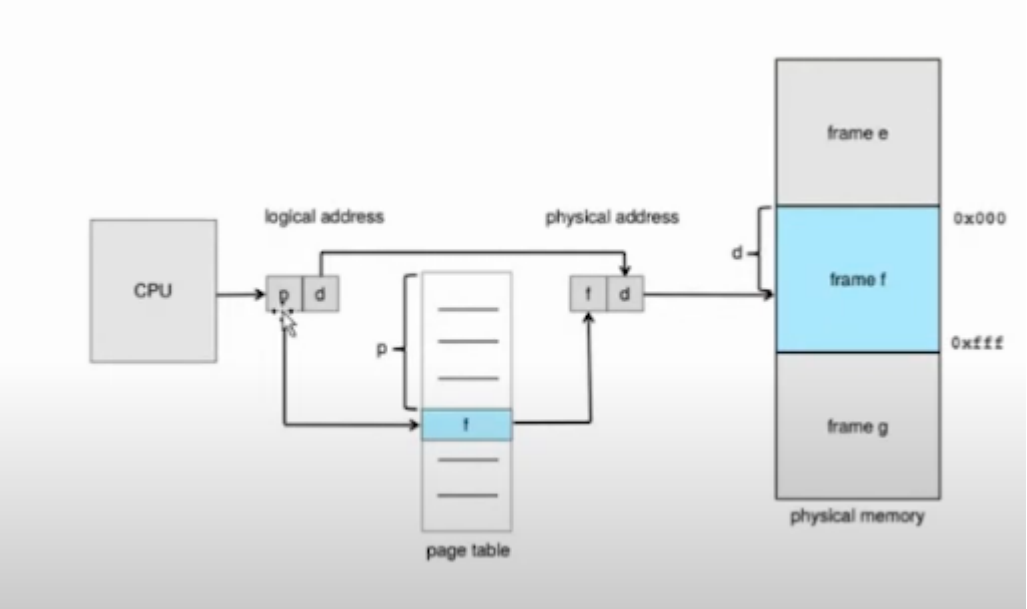
La paginación, una técnica más avanzada(necesita que el hardware la permita), elimina la necesidad de que los procesos se carguen en memoria de forma contigua. La memoria física se divide en marcos de página(frames) de tamaño fijo, y los procesos se dividen en páginas del mismo tamaño.

3.1. Tabla de Páginas:

Para mapear las páginas del proceso a los marcos(Frame) de página de la memoria física, se utiliza una tabla de páginas. Cada entrada en la tabla de páginas indica la ubicación física (el marco de página) de una página del proceso.

3.2. Traducción de Direcciones:

Cuando la CPU necesita acceder a una instrucción o a un dato, genera una dirección lógica, que se divide en dos partes: el número de página y el desplazamiento dentro de la página. Utilizando la tabla de páginas, el número de página se traduce al número de marco de página correspondiente, obteniendo la dirección física.(mejora el problema de contiguidad pero me relentiza los accessos a memoria ya que me agrega dos accesos)



3.3. TLB (Translation Lookaside Buffer): Acelerando la Traducción

Para optimizar el proceso de traducción, se implementa la TLB, una memoria caché que almacena las traducciones de página más recientes. Al acceder a una dirección lógica, primero se consulta la TLB. Si la traducción se encuentra en la TLB (hit), se obtiene la dirección física rápidamente. Si no se encuentra (miss), se accede a la tabla de páginas en memoria, lo que implica un tiempo de acceso mayor.

3.4. Tasa de Hits y Tiempo de Acceso Efectivo

La eficiencia de la TLB se mide por su tasa de hits, que es el porcentaje de veces que la traducción se encuentra en la TLB. Cuanto mayor sea la tasa de hits, menor será el tiempo de acceso efectivo a la memoria.

3.5. Fragmentación Interna en la Paginación

La paginación también puede generar fragmentación interna, ya que el tamaño de las páginas es fijo y un proceso puede no ocupar completamente la última página asignada. Sin embargo, la fragmentación interna en la paginación suele ser menor que en los esquemas de particionamiento.

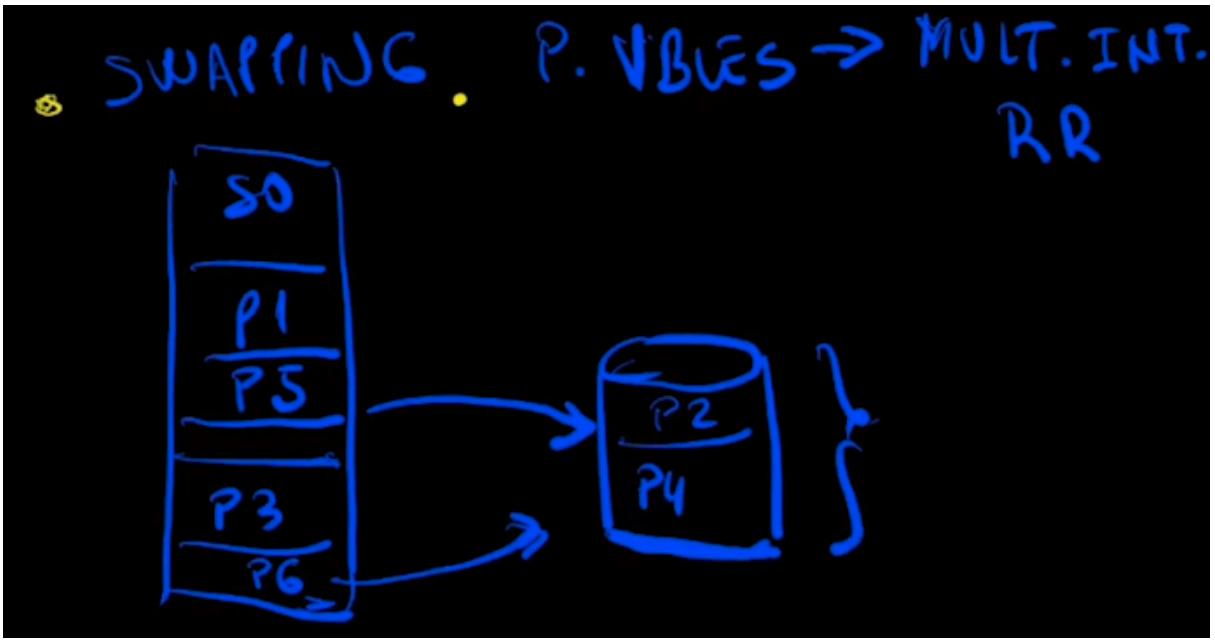
MEMORIA VIRTUAL

Antes de la memoria virtual, los programas debían cargarse completamente en memoria para ejecutarse. Esto limitaba el tamaño de los programas y la cantidad de procesos que podían ejecutarse concurrentemente.

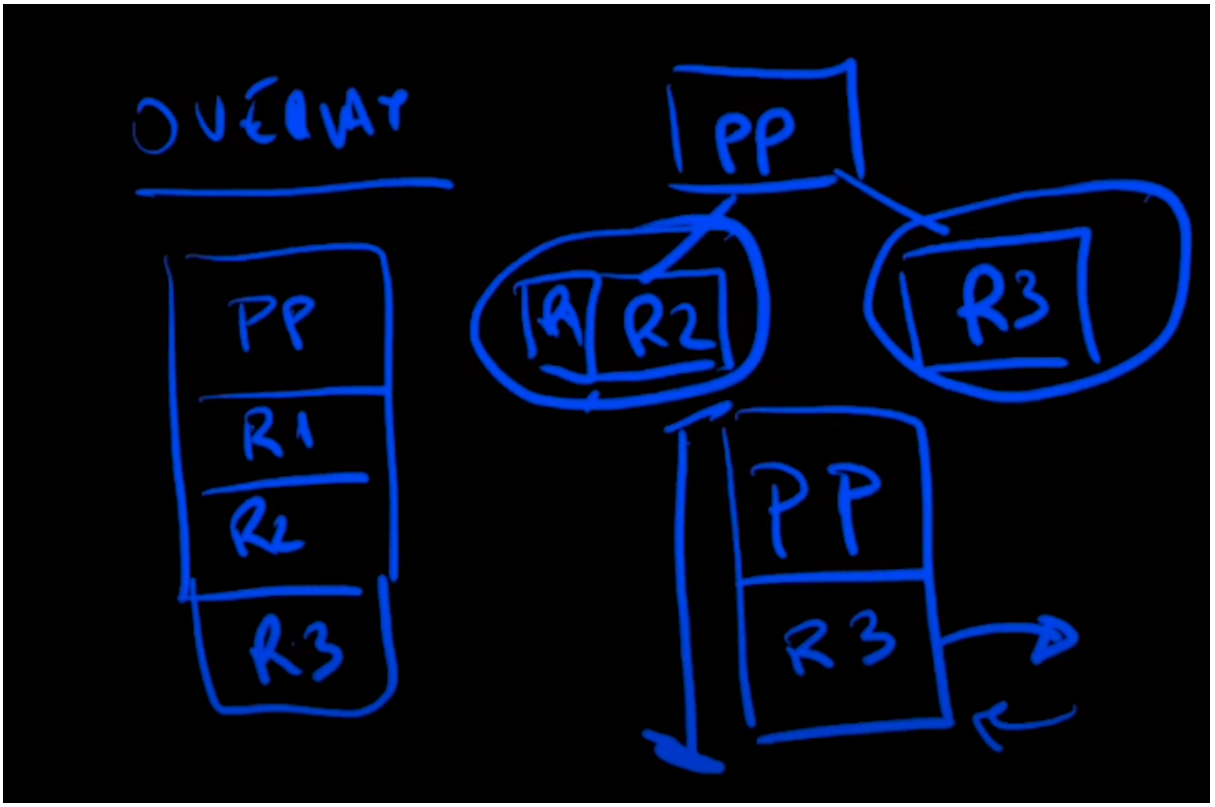
La memoria virtual utiliza el disco como una extensión de la memoria RAM, creando la ilusión de que hay más memoria disponible. Solo las partes del programa que se necesitan en un momento dado se cargan en memoria RAM, mientras que el resto reside en el disco.

Técnicas Precursoras de la Memoria Virtual:

- **Swapping:** Esta técnica consiste en mover procesos completos entre la memoria RAM y el disco. Si un proceso no se está utilizando activamente, se puede "sacar" (swap out) al disco para liberar espacio en memoria RAM. Cuando se necesita de nuevo, se "trae" (swap in) de vuelta a la memoria RAM.



- **Overlay:** Esta técnica se aplica a programas individuales. El programa se divide en módulos, y solo el módulo principal y los módulos necesarios en un momento dado se cargan en memoria. Cuando se necesita un módulo diferente, se carga desde el disco, reemplazando a un módulo que ya no se necesita.



Paginación por Demanda:

La implementación más común de la memoria virtual es la paginación por demanda(tambien existe por segmentacion). Se basa en la paginación, donde la memoria se divide en marcos de página de tamaño fijo, y los procesos se dividen en páginas del mismo tamaño.

En la paginación por demanda, solo se cargan en memoria las páginas que el proceso necesita en un momento dado. Cuando el proceso intenta acceder a una página(en el proceso de traduccion) que no está en memoria, se produce una **falla de página** (page fault).

Existe un tipo que es la demanda pura(siempre se genera un page fault), una instruccion puede acceder a multiples paginas y el hardwaare debe poder soportar este tipo de paginacion.

Pasos para Manejar una Falla de Página:

1. El hardware genera una interrupción.
2. El sistema operativo toma el control.
3. Se verifica que la referencia a la página sea válida.
4. Se busca la página en el disco (en el área de swap o backing store).
5. Se busca un marco de página libre en memoria RAM.
6. Si no hay marcos libres, se elige una página víctima para ser "sacada" (page out) al disco.

7. Se carga la página solicitada en el marco libre.
8. Se actualiza la tabla de páginas para reflejar la nueva ubicación de la página.
9. Se reinicia la instrucción que causó la falla de página.

Beneficios de la Memoria Virtual:

- Permite ejecutar programas más grandes que la memoria física disponible.
- Aumenta el grado de multiprogramación, permitiendo que se ejecuten más procesos concurrentemente.
- Reduce las operaciones de entrada/salida, ya que solo se cargan en memoria las partes del programa que se necesitan.
- Acelera la creación de procesos, ya que no es necesario cargar el programa completo en memoria al inicio.

Aspectos Adicionales:

- **Seguridad:** Los sistemas con un nivel de seguridad C2 exigen que la memoria se "limpie" (se escriban ceros en todos los bytes) antes de ser reasignada a un nuevo proceso. Esto evita que un proceso pueda acceder a información residual de otro proceso.
- **Rendimiento:** La paginación por demanda puede generar un mayor número de operaciones de entrada/salida, lo que puede afectar el rendimiento. Se utilizan diversas técnicas para minimizar este impacto, como el uso de la TLB y algoritmos inteligentes para la selección de páginas víctimas.

Copy-On-Write

El mecanismo de **Copy-On-Write (COW)**, central en la gestión eficiente de memoria en sistemas operativos como Unix/Linux, se activa especialmente durante la creación de un nuevo proceso a partir de uno existente, como al usar la llamada al sistema `fork()`. COW busca **maximizar la eficiencia al retrasar la copia de memoria hasta que sea estrictamente necesario**.

Compartiendo Páginas: Eficiencia en la Creación de Procesos

Cuando un proceso padre usa `fork()` para crear un proceso hijo, la técnica tradicional implica copiar toda la memoria del padre al hijo. COW, en cambio, **permite que padre e hijo compartan inicialmente las mismas páginas de memoria física**, evitando una costosa copia inicial.

Tablas de Páginas: Gestionando el Acceso Compartido

Para administrar este acceso compartido, **cada proceso (padre e hijo) mantiene su propia tabla de páginas**. Las entradas correspondientes a las páginas compartidas se marcan como **"solo lectura"** en ambas tablas, garantizando que ningún proceso modifique accidentalmente la memoria compartida sin que el sistema operativo lo detecte.

Modificando Datos: El Momento de la Copia

Si un proceso (padre o hijo) necesita modificar una página compartida, se dispara una **interrupción**. El sistema operativo, al capturar esta interrupción, reconoce la necesidad de una copia independiente de la página para el proceso que busca la modificación. Se genera una **copia de la página**, se asigna al proceso, y la tabla de páginas se actualiza para reflejar este cambio. La marca de "solo lectura" se elimina, permitiendo la escritura en la nueva copia privada.

Beneficios de COW: Ahorro de Memoria y Mayor Velocidad

COW ofrece dos beneficios principales:

- **Ahorro de Memoria:** Al evitar copias innecesarias, COW optimiza el uso de la memoria, especialmente cuando los procesos hijos heredan gran parte del estado del padre y no realizan modificaciones extensas.
- **Ahorro de Tiempo en `fork()`:** La creación de procesos hijos se vuelve mucho más rápida al eliminar la necesidad de copiar grandes volúmenes de memoria durante la llamada a `fork()`.

Gestión de Páginas de Código y Datos: Diferencias Claves

Las páginas de código, que almacenan instrucciones, no se modifican durante la ejecución normal, por lo que pueden ser compartidas sin problemas. Las páginas de datos, en cambio, sí son susceptibles a modificaciones, lo que justifica la implementación de COW para evitar inconsistencias entre procesos.

Algoritmos de Reemplazo de Páginas: Gestionando la Escasez de Memoria Estos algoritmos entran en juego cuando la memoria física disponible se agota y es necesario liberar espacio para nuevas páginas. Se exploran tres algoritmos principales:

- **FIFO (First-In, First-Out):** El algoritmo más simple, que selecciona como víctima la página que lleva más tiempo en memoria.
- **Óptimo:** Un algoritmo teórico que selecciona la página que no se usará durante el mayor tiempo posible. Aunque ideal en teoría, en la práctica es imposible de implementar ya que requiere conocimiento del futuro.
- **LRU (Least Recently Used):** Un algoritmo popular que selecciona la página que lleva más tiempo sin ser usada. Tambien existe el de segunda chance, que les permite

mantenerse un tiempo mas en memoria a algunas paginas que iban a ser reemplazadas por primera vez, pero ya en la segunda las reemplaza.

Asignación de Frames: Distribuyendo la Memoria entre Procesos

La asignación de frames, es decir, cómo se distribuye la memoria física disponible entre los distintos procesos, también se aborda en las fuentes. Se mencionan dos esquemas principales:

- **Asignación Fija:** Se asigna una cantidad fija de frames a cada proceso, independientemente de su tamaño o necesidades.
- **Asignación Proporcional:** Se asigna una cantidad de frames proporcional al tamaño del proceso.

Reemplazo Global vs. Local: De Dónde Sacar la Página Víctima

Cuando es necesario liberar un frame, el sistema operativo puede recurrir al reemplazo global o local:

- **Reemplazo Global:** Se puede seleccionar cualquier frame de cualquier proceso como víctima.
- **Reemplazo Local:** La página víctima se selecciona únicamente entre los frames asignados al proceso que necesita liberar espacio.

"Thrashing" o Hiperpaginación: Un Enemigo del Rendimiento

El "thrashing" o hiperpaginación es un problema grave que puede afectar al rendimiento del sistema. Se produce cuando el sistema operativo se ve obligado a realizar una cantidad excesiva de paginación (intercambio de páginas entre memoria principal y disco), dedicando la mayor parte de su tiempo a esta tarea en lugar de ejecutar procesos. Las causas del "thrashing" pueden ser diversas, como una cantidad insuficiente de memoria física, una mala configuración del sistema o un algoritmo de reemplazo de páginas ineficiente.

El Principio de Localidad: Clave para Evitar el "Thrashing"

El principio de localidad, que establece que los programas tienden a acceder a un conjunto reducido de páginas (su "localidad") durante un período de tiempo, es fundamental para comprender y mitigar el "thrashing". Si la suma de los tamaños de las localidades de todos los procesos en ejecución es mayor que la memoria física disponible, es muy probable que se produzca "thrashing".

El Conjunto de Trabajo ("Working Set"): Mapeando la Localidad

El concepto de "working set" o conjunto de trabajo se utiliza para representar la localidad de un proceso en un momento dado. El "working set" de un proceso se define como el conjunto de páginas a las que ha accedido el proceso en un período de tiempo reciente. Un "working

set" demasiado grande, en relación con la memoria física disponible, puede ser un indicio de que el sistema está en riesgo de sufrir "thrashing".

ALMACENAMIENTO MASIVO

Introducción al almacenamiento masivo

En este capítulo, discutimos cómo está estructurado el sistema de almacenamiento masivo, el sistema de almacenamiento no volátil de una computadora. El principal sistema de almacenamiento masivo en las computadoras modernas es el almacenamiento secundario, que generalmente se proporciona a través de discos duros (HDD) y dispositivos de memoria no volátil (NVM). Algunos sistemas también tienen almacenamiento terciario más lento y más grande, generalmente compuesto por cintas magnéticas, discos ópticos o incluso almacenamiento en la nube.
Dado que los dispositivos de almacenamiento más comunes e importantes en los sistemas informáticos modernos son los HDD y los dispositivos NVM, la mayor parte de este capítulo se dedica a discutir estos dos tipos de almacenamiento. Primero describimos su estructura física. Luego consideramos los algoritmos de programación, que programan el orden de E / S para maximizar el rendimiento. A continuación, discutimos el formateo del dispositivo y la administración de bloques de arranque, bloques dañados y espacio de intercambio. Finalmente, examinamos la estructura de los sistemas RAID.

Jerarquía de memoria

En cuanto a la jerarquía de memorias hay 3 puntos sobre la memoria a tener en cuenta

1. **Capacidad:** El tamaño es siempre un tema abierto, cuánta más capacidad tenga se desarrollarán aplicaciones que la utilicen
2. **Rapidez:** Cuando el procesador ejecuta instrucciones, no es deseable que tenga que detenerse al a espera de instrucciones o de operandos. (Tiempo de acceso
3. **Coste:** Debe ser razonable con relación a los otros componentes

Estas 3 características están relacionadas entre si de la siguiente manera:

- A menos tiempo de acceso, mayor coste por bit.

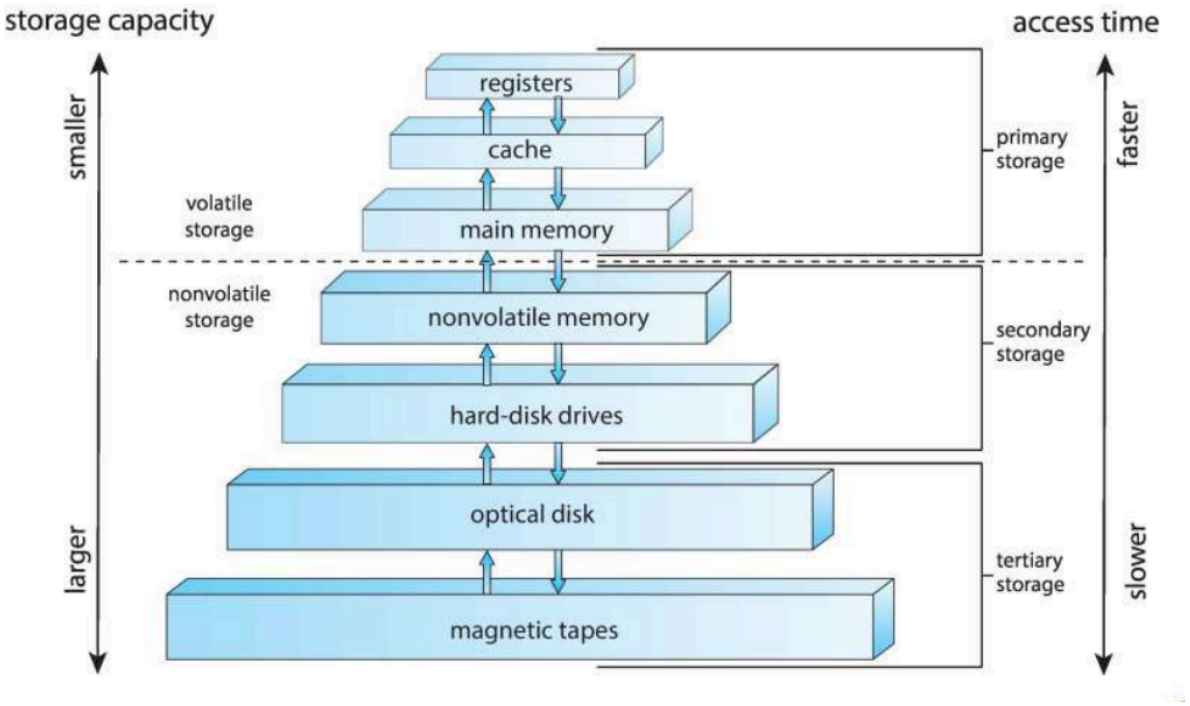
- A mayor capacidad, menor coste por bit
- A mayor capacidad, mayor tiempo de acceso.

Al intentar ganar en algún aspecto perdemos en otro.

Pirámide de la jerarquía de memorias

Podemos ver representado esta jerarquía en una pirámide donde en la punta tenemos al procesador, lo que más rápido anda y menor cantidad de almacenamiento tiene.

Y por el contrario, cuando nos alejamos nos encontramos con mayor capacidad pero la velocidad de acceso disminuye, así también el precio.



Estructura de almacenamiento masivos

La mayor parte del almacenamiento secundario para computadoras modernas se proporciona a través de discos duros (HDD) y dispositivos de memoria no volátil (NVM). En esta sección, describimos los mecanismos básicos de estos dispositivos y explicamos cómo los sistemas operativos traducen sus propiedades físicas a almacenamiento lógico a través de la asignación de direcciones.

1. Hard Disk Drives (HDD)

Conceptualmente, los discos duros son relativamente simples. Cada disco de plato tiene una forma circular plana, como un CD. Las dos superficies de un plato están cubiertas con un material magnético. Almacenamos información grabándola magnéticamente en los platos y leemos información detectando el patrón magnético en los platos.

- Esos discos pueden girar a distintas velocidades dependiendo de su tecnología.
- El **tiempo de posicionamiento** es el tiempo que se tarda en mover el brazo del disco al disco seleccionado(tiempo de búsqueda) y el tiempo para que el sector deseado rote debajo de la cabeza del disco(latencia rotacional).
- La **tasa de transferencia** (transfer rate) nos indica cuanta información se puede transportar por unidad de tiempo.
- Cuando hablamos de la **performance o rendimiento de los discos magnéticos**, hace referencia a la **latencia de acceso** , la cual se define como el tiempo promedio de posicionamiento de las cabezas de lectura/escritura + el tiempo promedio de latencia rotacional.
- Existe el tiempo de selección electrónica de la cabeza de lectura/escritura (despreciable)

En el disco duro tenemos un conjunto de discos rotando sobre una cabeza, en cada disco tenemos pistas concéntricas, y a su vez, cada una de esas pistas está dividida en sectores, que son aquellos que son leídos cuando se pide leer determinado dato. Estos discos están apoyados en los brazos del disco.

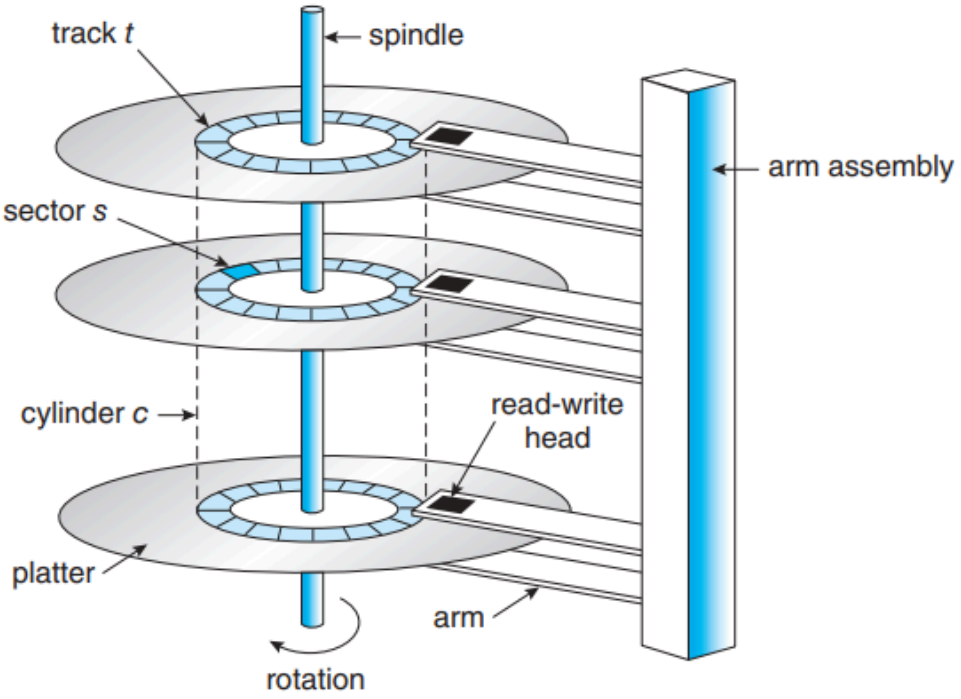
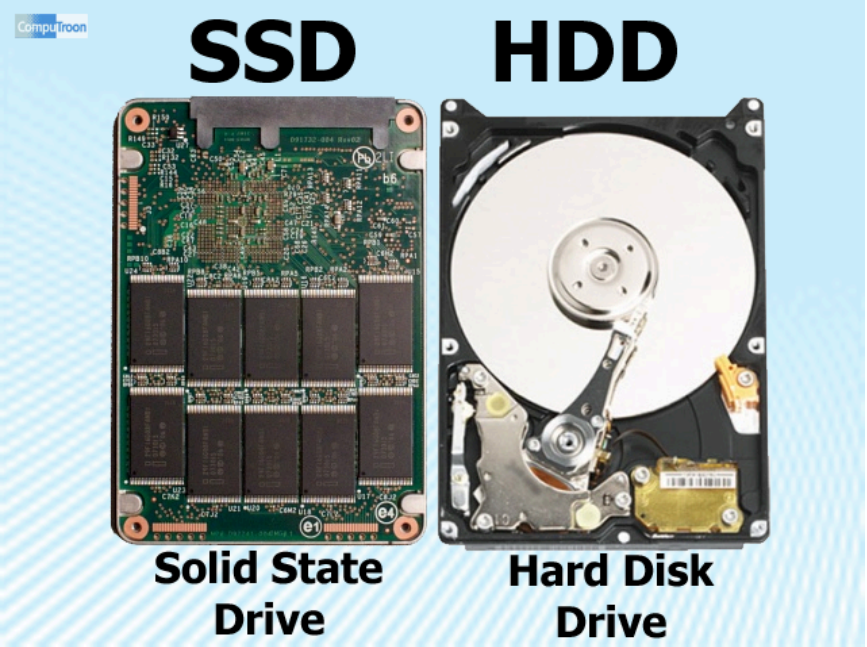


Figure 11.1 HDD moving-head disk mechanism.

2. Solid State Disks (SSD)

Los dispositivos de memoria no volátil (NVM). Simplemente descrito, los dispositivos NVM son eléctricos en lugar de mecánicos. Más comúnmente, tal dispositivo está compuesto por un controlador y chips semiconductores de flash NAND, que se utilizan para almacenar datos. Si es como un disco de unidad, entonces se llama **discos de estado sólido**.

- Son más confiables porque no son tan frágiles como los discos duros
- Son más caro por MB pero también son mucho más rápidos
- No tiene partes móviles, por ende, no hay tiempo de rotación ni de posicionamiento.



3. Magnetic Tapes

Las cintas magnéticas son una forma de almacenamiento de datos de gran capacidad que, a pesar de su reducido tamaño, ofrece una relación capacidad-costo excepcional. Esta característica las hace ideales para la creación de copias de seguridad, siendo la solución ideal para el almacenamiento de grandes cantidades de información sin incurrir en grandes costos. Se trata, por tanto, de una tecnología muy útil que se ha convertido en una herramienta indispensable para la protección de los datos.

Disk Attachment

Hay distintas formas de conectar un disco a una computadora, a estos se accede a través de puertos de entrada/salida que se comunican por medio de buses.

Existen distintas tecnologías para la conexión de los discos:

- ATA (Advanced technolgy attachment)
- SATA (Serial Advanced technolgy attachment)
- eSATA
- SCSI
- USB
- FC (Fibra optica)

Para los discos de estado solido se diseño una nueva interfaz llamada NVM express conectado directamente al PCI bus.

Se puede acceder al almacenamiento de tres maneras

1. **Host -attached:** Discos dentro del servidor
2. **Network-attached:** Conectados a través de una red (NAS), un ejemplo típico es tener 2 PCs y una pc me comparte en la red una carpeta.
3. **Cloud:** El almacenamiento en la nube permite a los usuarios acceder a sus archivos desde cualquier lugar y en cualquier momento, siempre y cuando tengan acceso a Internet

Algoritmos de planificación del brazo

Estos son algoritmos de planificación de la entrada y salida sobre los discos duros, también llamados planificación del brazo del disco. Esto significa tener un tiempo de acceso rápido y tener un ancho de banda importante para los discos.

Para poder acceder de manera rápida a los discos magnéticos hay que ocuparse de la tarea que más tiempo lleva que es el tiempo de posicionamiento de la cabeza de lectura/escritura para

alcanzar el cilindro que se quiere leer o escribir. Otro punto que se intenta lograr es minimizar el tiempo de seek o búsqueda.

1. Algoritmo FCFS

El algoritmo más simple de programación de disco es, por supuesto, el algoritmo de primero en llegar, primero en servir (FCFS) (o FIFO). Este algoritmo es intrínsecamente justo, pero generalmente no ofrece el mejor servicio.

Se dibujan en un eje todos los cilindros de un disco y marcamos con un punto donde está la cabeza de lectura y escritura, si va a resolver los pedidos en el orden que llega de manera secuencial, haciendo una especie de zic zac

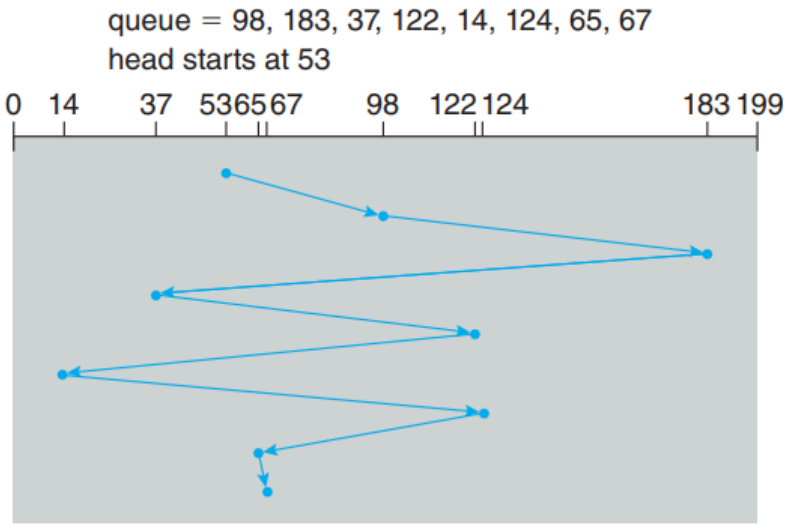


Figure 11.6 FCFS disk scheduling.

2. Algoritmo SCAN (Ascensor)

En el algoritmo SCAN, el brazo del disco comienza en un extremo del disco y se mueve hacia el otro extremo, atendiendo las solicitudes a medida que alcanza cada cilindro, hasta llegar al otro extremo del disco. En el otro extremo, la dirección del movimiento de la cabeza se invierte y el servicio continúa. La cabeza escanea continuamente de un lado a otro del disco.

El algoritmo SCAN a veces se llama algoritmo de ascensor, ya que el brazo del disco se comporta como un ascensor en un edificio, primero atendiendo todas las solicitudes hacia arriba y luego invirtiendo para atender las solicitudes del otro lado.

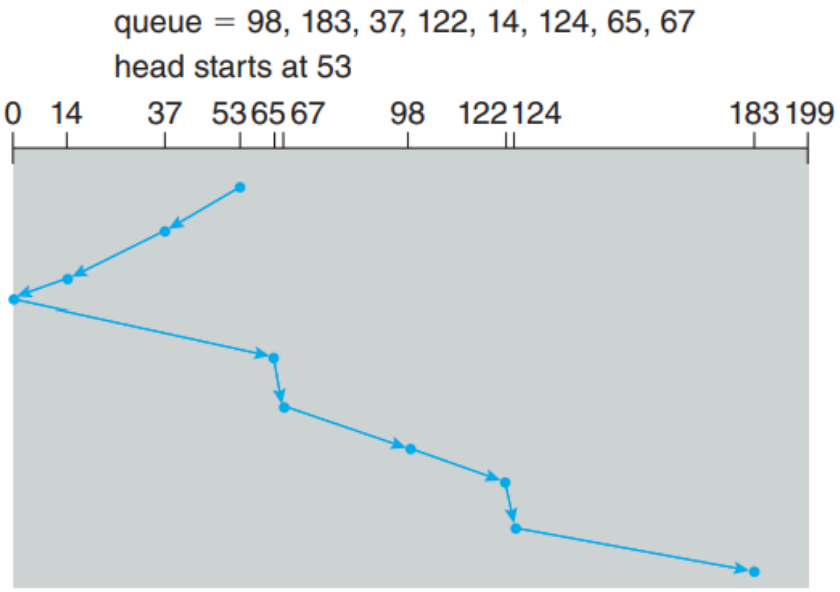


Figure 11.7 SCAN disk scheduling.

3. Algoritmo C-SCAN

El algoritmo C-SCAN (Circular SCAN) es una variante de SCAN diseñada para proporcionar un tiempo de espera más uniforme. Al igual que SCAN, C-SCAN mueve la cabeza de un extremo del disco al otro, prestando servicios en el camino. Sin embargo, cuando la cabeza llega al otro extremo, regresa inmediatamente al comienzo del disco sin prestar servicios en el viaje de regreso.

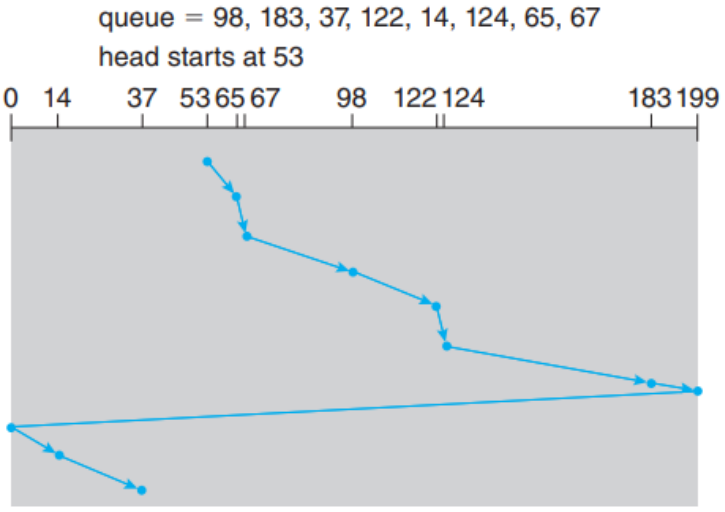


Figure 11.8 C-SCAN disk scheduling.

RAID

RAID es un acrónimo que significa Redundant Array of Inexpensive Disks (Matriz Redundante de Discos Baratos). Es una tecnología de almacenamiento de datos que combina varios discos duros en un solo sistema lógico para mejorar el rendimiento, la disponibilidad y la tolerancia a fallos.

Aquí hay una lista de los distintos tipos de RAID que existen:

1. RAID 0: Este tipo de RAID divide los datos en fragmentos y los distribuye entre varios discos duros. Este tipo de RAID no proporciona redundancia, lo que significa que si un disco falla, todos los datos se pierden.
2. RAID 1: Este tipo de RAID es conocido como espejo y consiste en replicar los datos en dos discos duros. Si uno de los discos falla, el otro disco toma su lugar sin interrupción.
3. RAID 5: Este tipo de RAID requiere al menos tres discos y proporciona redundancia utilizando paridad. La paridad se utiliza para reconstruir los datos en caso de fallo de un disco.
4. RAID 6: Similar al RAID 5, pero utiliza dos bloques de paridad en lugar de uno. Esto proporciona una mayor tolerancia a fallos ya que se pueden tolerar dos fallos de discos.
5. RAID 10: Este tipo de RAID combina RAID 1 y RAID 0. Los datos se dividen en fragmentos y se replican en parejas de discos.
6. RAID 50: Este tipo de RAID combina RAID 5 y RAID 0. Los datos se dividen en fragmentos y se protegen mediante paridad en una configuración RAID 5 y luego se combinan en una configuración RAID 0.
7. RAID 60: Similar al RAID 50, pero utiliza dos bloques de paridad en lugar de uno. Esto proporciona una mayor tolerancia a fallos.

Estos son solo algunos de los tipos de RAID más comunes. Hay muchas otras configuraciones que combinan diferentes tipos de RAID para lograr un equilibrio entre rendimiento, disponibilidad y tolerancia a fallos. Es importante elegir el tipo correcto de RAID para cada aplicación en función de sus requisitos de rendimiento, disponibilidad y tolerancia a fallo

SISTEMA DE ARCHIVOS

El Archivo: Una Abstracción Clave

Desde la perspectiva del sistema operativo, un archivo se presenta como una secuencia lógica y continua de bytes, sin importar la naturaleza de la información que contiene. Esta abstracción permite al sistema operativo trabajar con cualquier tipo de archivo de manera uniforme, sin necesidad de conocer la estructura interna específica de cada uno. La responsabilidad de interpretar el contenido de un archivo recae en la aplicación que lo crea o manipula. Por ejemplo, un procesador de texto como Word es el encargado de interpretar la estructura interna de un archivo .doc, mientras que una hoja de cálculo como Excel se encarga de interpretar la estructura de un archivo .xls.

Atributos del Archivo: Una Identidad Completa

Para identificar y gestionar cada archivo, el sistema operativo mantiene una serie de atributos asociados a él. Estos atributos forman parte de la metadata que describe el archivo, permitiendo al sistema operativo y a los usuarios comprender su naturaleza y características. Los atributos más importantes son:

- **Nombre:** Es la representación legible por el usuario que identifica al archivo dentro del sistema.
- **Identificador único (número de inodo):** Es un número que identifica unívocamente al archivo dentro del sistema de archivos. Este número es utilizado internamente por el sistema operativo para referenciar al archivo.
- **Tipo:** Se refiere al formato del archivo, generalmente indicado por la extensión del nombre del archivo (por ejemplo, .txt, .doc, .exe, etc.). Esta información permite al sistema operativo asociar el archivo con la aplicación adecuada para su manipulación.
- **Ubicación:** Se refiere a la posición física del archivo en el disco. El sistema operativo utiliza punteros que hacen referencia a los bloques de disco que contienen los datos del archivo.
- **Tamaño:** Indica la cantidad de bytes que ocupa el archivo en el disco.
- **Protección:** Define los permisos de acceso al archivo, especificando quién puede leer, escribir o ejecutar el archivo. Los permisos se establecen para el dueño del archivo, para el grupo al que pertenece el dueño y para el resto de los usuarios del sistema.
- **Fechas y horas:** Registran la fecha y hora de creación del archivo, la última vez que se accedió a él (lectura) y la última vez que se modificó. Esta información es útil para el seguimiento y la gestión de archivos.
- **Identificación del usuario que lo creó:** Permite identificar al usuario responsable de la creación del archivo.

Operaciones con Archivos: Manipulando la Información

El sistema de archivos proporciona un conjunto de operaciones para manipular los archivos, permitiendo a las aplicaciones crear, acceder, modificar y eliminar información. Las operaciones más comunes son:

- **Crear:** Permite la creación de nuevos archivos. La forma de crear un archivo varía según el programa que lo genera. Por ejemplo, un editor de texto crea archivos de texto, un compilador genera archivos ejecutables y una hoja de cálculo produce archivos de datos.
- **Abrir (Open):** Es el primer paso para acceder a un archivo. El sistema operativo busca el archivo en la estructura de directorios, carga su metadata (incluyendo el inodo) en memoria y verifica los permisos de acceso del usuario. Si la operación es exitosa, devuelve un descriptor de archivo (file descriptor), un número entero que se utiliza para referenciar al archivo en las operaciones posteriores.
- **Leer (Read):** Permite obtener datos del archivo. La lectura se realiza de forma secuencial, a partir de la posición actual del puntero lógico. El puntero lógico es una marca que indica la posición actual dentro del archivo desde la cual se leerá la siguiente información. Al finalizar la lectura, el puntero lógico avanza para apuntar a la siguiente posición.
- **Escribir (Write):** Permite guardar datos en el archivo. La escritura también se realiza de forma secuencial, comenzando en la posición actual del puntero lógico. Al finalizar la escritura, el puntero lógico se actualiza para reflejar la nueva posición dentro del archivo.
- **Reposicionar (Seek):** Permite mover el puntero lógico a una posición específica dentro del archivo, sin necesidad de leer o escribir datos. Esta operación es útil para acceder a diferentes partes del archivo de forma rápida y eficiente.
- **Cerrar (Close):** Finaliza el acceso a un archivo. El sistema operativo libera los recursos asociados al archivo, como la memoria utilizada para almacenar la metadata, y fuerza la escritura de los bloques modificados (bloques sucios) al disco, asegurando que los cambios realizados en el archivo se guarden de forma permanente.
- **Borrar (Delete):** Elimina un archivo del sistema de archivos. Al borrar un archivo, se elimina tanto su contenido como su metadata (inodo).
- **Truncar (Truncate):** Elimina el contenido de un archivo, pero conserva sus atributos, como el nombre y los permisos. El archivo sigue existiendo en el sistema de archivos, pero su tamaño se reduce a cero.

El Intrincado Baile de la Gestión del Espacio en Disco

Para gestionar eficientemente el espacio en disco, el sistema de archivos lo divide en bloques, unidades mínimas de almacenamiento. La asignación y liberación de bloques se realiza de forma transparente para el usuario, pero es un proceso crucial para el correcto funcionamiento del sistema de archivos.

El **superbloque** juega un papel fundamental en la gestión del espacio en disco. Contiene información general del sistema de archivos, incluyendo:

- **Listas de bloques libres:** Indican qué bloques del disco están disponibles para almacenar nuevos datos.
- **Listas de nodos índice (inodos) libres:** Indican qué inodos están disponibles para describir nuevos archivos.

Los **inodos**, como se ha explicado anteriormente, describen los archivos físicos, almacenando sus atributos y los punteros a los bloques de datos que los componen.

Directorios: Organizando el Caos

Para facilitar la organización y el acceso a los archivos, el sistema de archivos utiliza una estructura jerárquica de directorios, similar a la estructura de un árbol. El **directorio raíz (/)** es el punto de partida de la jerarquía, y a partir de él se ramifican los demás directorios. Cada directorio puede contener archivos y otros directorios, formando una estructura arborescente.

Los directorios son, en esencia, archivos especiales que contienen información sobre los archivos y subdirectorios que se encuentran dentro de ellos. Cada entrada en un directorio asocia el nombre de un archivo o subdirectorio con su correspondiente número de inodo.

Para navegar por la jerarquía de directorios, se utilizan dos entradas especiales presentes en cada directorio:

- **"." (punto):** Hace referencia al propio directorio.
- **".." (dos puntos):** Hace referencia al directorio padre.

Montaje de Sistemas de Archivos: Expandiendo el Horizonte

Un disco físico puede dividirse en varias particiones, y cada partición puede formatearse con un sistema de archivos diferente. El comando `mount` permite conectar un sistema de archivos a la jerarquía principal del sistema en un punto de montaje específico. De esta manera, un sistema de archivos que reside en una partición separada se integra como si fuera parte de la estructura de directorios principal.

El comando `umount` se utiliza para desconectar un sistema de archivos. Al desmontar un sistema de archivos, se lo separa de la jerarquía principal, dejándolo inaccesible hasta que se vuelva a montar.

Cruzando Fronteras: Acceso a Archivos Remoto

En entornos de red, es común la necesidad de acceder a archivos almacenados en otros equipos. Para facilitar el acceso remoto a archivos, se utilizan protocolos como **NFS (Network File System)** y **Samba**.

NFS es un protocolo ampliamente utilizado en entornos Unix/Linux para compartir archivos a través de la red. Permite a los clientes NFS montar sistemas de archivos remotos como si fueran parte de su propio sistema de archivos local, brindando acceso transparente a los archivos compartidos.

Samba es un conjunto de software que permite compartir archivos e impresoras entre equipos Windows y Unix/Linux. Implementa los protocolos SMB/CIFS utilizados por Windows para compartir archivos en red, permitiendo a los equipos Linux integrarse en entornos Windows y viceversa.

Consistencia del Sistema de Archivos: Un Escudo contra la Corrupción

La consistencia del sistema de archivos es fundamental para garantizar la integridad de los datos y el correcto funcionamiento del sistema operativo. La **corrupción del sistema de archivos** puede ocurrir por diversos factores, como fallos de hardware, errores de software o interrupciones abruptas del suministro eléctrico.

Para prevenir la corrupción y asegurar la consistencia, los sistemas de archivos modernos utilizan técnicas como el **journaling**, un mecanismo que registra las transacciones realizadas en el sistema de archivos antes de que se apliquen de forma permanente.

En caso de un fallo, el sistema de archivos puede utilizar el **journal** para reconstruir su estado consistente, deshaciendo las transacciones incompletas y asegurando que los datos no se pierdan ni se corrompan.

El comando `fsck` (filesystem check) es una herramienta que permite verificar y reparar inconsistencias en el sistema de archivos. Al ejecutar `fsck`, se analiza la estructura del sistema de archivos en busca de errores, como bloques perdidos, inodos inconsistentes o referencias incorrectas entre directorios y archivos. Si se encuentran errores, `fsck` intenta repararlos para restaurar la consistencia del sistema de archivos.

Tipos de Archivos: Más Allá de la Apariencia

Aunque el sistema operativo trata a todos los archivos como secuencias de bytes, los archivos se clasifican en diferentes tipos según su formato y su uso.

El **tipo de archivo** se suele identificar por la extensión del nombre del archivo. Algunos tipos comunes de archivos son:

- **Archivos ejecutables:** Contienen código de programa que puede ser ejecutado por el sistema operativo. Suelen tener extensiones como .exe, .bin o .com.
- **Archivos de código fuente:** Contienen código escrito en un lenguaje de programación. Su extensión depende del lenguaje utilizado (por ejemplo, .c para C, .java para Java, .py para Python, etc.).
- **Archivos de texto:** Contienen texto plano, sin formato. Suelen tener extensiones como .txt, .log o .cfg.
- **Archivos de datos:** Contienen información organizada en un formato específico, como bases de datos, hojas de cálculo, imágenes, audio, vídeo, etc. Su extensión depende del formato de datos (por ejemplo, .db, .xls, .jpg, .mp3, .avi, etc.).

Enlaces: Tejiendo Redes entre Archivos

Los enlaces son mecanismos que permiten crear múltiples referencias a un mismo archivo. Existen dos tipos principales de enlaces:

- **Enlace duro (hard link):** Crea un nuevo nombre para un archivo existente, ambos nombres referencian al mismo inodo y, por lo tanto, al mismo contenido físico en el disco. Al crear un hard link, se incrementa el contador de referencias al inodo. El archivo original solo se elimina del sistema de archivos cuando el contador de referencias llega a cero.
- **Enlace simbólico (symbolic link o soft link):** Crea un archivo especial que contiene la ruta al archivo original. Al acceder al enlace simbólico, el sistema operativo redirige la solicitud al archivo original, utilizando la ruta especificada en el enlace simbólico. Si se elimina el archivo original, el enlace simbólico queda "roto", apuntando a una ubicación inexistente.

Control de Acceso: Protegiendo la Información

Los permisos de acceso a los archivos son una parte fundamental de la seguridad del sistema operativo. Permiten controlar quién puede acceder a cada archivo y qué acciones pueden realizar sobre él (leer, escribir, ejecutar).

El comando `chmod` permite modificar los permisos de un archivo, especificando los permisos para el dueño, el grupo y el resto de los usuarios.

El comando `chown` permite cambiar el propietario de un archivo. Solo el administrador del sistema (root) tiene la autoridad para cambiar el propietario de un archivo.

Bloqueos: Previniendo la Interferecia

En sistemas multiusuario o multiproceso, es posible que varios procesos intenten acceder a un mismo archivo de forma simultánea. Esto puede dar lugar a problemas de **interferencia**, donde las acciones de un proceso pueden afectar negativamente a las acciones de otro proceso sobre el mismo archivo.

Para evitar la interferencia, se utilizan **bloqueos**, mecanismos que permiten a un proceso obtener acceso exclusivo a un archivo o a una parte de él.

Existen dos tipos principales de bloqueos:

- **Bloqueo exclusivo:** Permite acceso exclusivo a un archivo o a una parte de él. Ningún otro proceso puede acceder al archivo bloqueado hasta que se libere el bloqueo.
- **Bloqueo compartido:** Permite a múltiples procesos leer un archivo de forma simultánea, pero impide que cualquier proceso lo modifique hasta que se liberen todos los bloqueos compartidos.