Sistema Operativo

El **Sistema Operativo** es software. Necesita procesador y memoria para ejecutarse.

Desde una perspectiva de **arriba hacia abajo**:

- Es un intermediario entre el usuario y el HW. Oculta y administra detalles con respecto al HW, y presenta programas más simples y amigables de manejar. Hay diferentes SO, y cada uno responde a diferentes objetivos.

Desde una perspectiva de **abajo hacia arriba**:

- Es un administrador de los recursos de HW, E/S, puede ejecutar varios procesos a la vez y es responsable de controlarlos. Compartir la CPU y memoria a lo largo del tiempo.

(Archivo: Abstracción que nos permite almacenar datos en un dispositivo de almacenamientos.)

Objetivos de un SO

- **Comodidad**: Hacer más fácil el uso del HW.
- **Eficiencia**: Hacer un uso más eficiente de los recursos del sistema.
- **Evolución**: Permitir introducir nuevas funciones al sistema sin interferir con las funciones anteriores.

Componentes de un Sistema Operativo

- **Kernel**: El núcleo del SO. encargado de que el software y el hardware puedan trabajar juntos, pues ejecuta programas y gestiona los recursos del sistema y dispositivos de hardware. Sus funciones más importantes son la administración de la memoria, del tiempo de CPU que los programas y procesos utilizan, y de que como usuarios podamos acceder a los dispositivos de E/S de forma cómoda.

- **Shell**: El Shell, o intérprete de comandos, es el modo de comunicación entre el usuario y el SO. Gracias a él, podemos dar órdenes para que nuestro SO realice las tareas que necesitamos. No debemos confundirlo con el SO, pues este es solo un programa que hace de mediador entre el usuario y el Kernel del sistema. El Shell puede ser tanto gráfico (Windows) como de texto (Bash). Cada usuario puede tener una shell, se pueden personalizar y son programables.

- **File System**: Los Sistemas de Archivos estructuran la información guardada en una unidad de almacenamiento (normalmente un disco duro) de una computadora, que luego será representada ya sea textual o gráficamente por medio de un gestor de archivos. La mayoría de SO tienen su propio File System.

Un SO debe evitar que un proceso se apropie de la CPU, que un proceso intente ejecutar instrucciones de E/S, que un proceso intente acceder a una posición de memoria fuera de su espacio declarado...

Para esto, el SO gestiona el uso de la CPU, detecta intentos de ejecución de instrucciones de E/S ilegales, detecta accesos ilegales a memoria, protege el vector de interrupciones...

El HW brinda apoyo para esto a través de los **modos de ejecución**.

Modos de Ejecución

La CPU tiene un bit en la PSW (Program Status Word) que representa el modo en que la CPU se está ejecutando (es decir, indica lo que la CPU puede hacer en función del modo en que está).

- **Modo Kernel/Supervisor**: Toda instrucción que se intente ejecutar en este modo es posible de ejecutar. Es como un modo superpoderoso. Se puede gestionar procesos, memoria, E/S.

- **Modo Usuario**: Las instrucciones que puedo ejecutar en este modo están limitadas. El proceso solo puede acceder a su propio espacio de direcciones. No se puede interactuar con el HW.

El kernel del SO se ejecuta en **Modo Kernel**, y el resto del SO y programas en **Modo Usuario**.

La CPU puede alternar entre un modo y otro, cambiando el valor del bit de modo a través de una interrupción (es la única manera de hacerlo).

Cuando arranca el sistema, arranca en **Modo Kernel**. Cada vez que comienza a ejecutarse un proceso de usuario, se pasa a **Modo Usuario**. Luego, cuando hay una interrupción o trap, se pasa a **Modo Kernel**.

- **Protección de la CPU**: Para evitar que los procesos se apropien de la CPU, hay un tipo de interrupción que ocurre constantemente, conocida como **interrupción por clock**.

  El kernel le da valor al contador que decrementa con cada tick de reloj, y al llegar a cero puede expulsar al proceso actual para ejecutar otro (se pasa a **Modo Kernel** para hacerlo).

- **Protección de la memoria**: Cada proceso que se ejecuta tiene límites que determinan el espacio de direcciones dentro de la memoria en donde puede trabajar, y no puede salirse de esos límites. Esos límites son el registro base (inicio) y el registro límite (fin).

System Calls

System Calls son funciones que los procesos pueden invocar para pedirle un servicio al SO.

Se pasa a **Modo Kernel** para atender un System Call. Un programa que funciona en Windows y no en Linux, se debe a que diferentes SO implementan distintas System Calls.

Kernel

Es un conjunto de módulos de SW, y se ejecuta en la CPU como cualquier otro proceso, por lo que necesita memoria y CPU. Pero no es un proceso. Tiene código, datos, PCB, pero no es un proceso. Hay varios enfoques:

- Kernel como una **entidad independiente** en Modo Kernel que se ejecuta a la par de todo proceso, con su propia región de memoria y con su propio stack.

- Kernel **dentro del proceso**, cuyo código se encuentra dentro del espacio de direcciones de cada proceso, y se ejecuta en el mismo contexto que cada proceso de usuario, pero en Modo Kernel.

Tipos de Kernel

- **Kernel Monolítico**: Toda funcionalidad que debe implementar el SO se ejecuta en Modo Kernel. Linux es monolítico.

- **Microkernel**: Se trata de que el kernel sea lo más chico posible. Para lograr esto, se trata de reducir al máximo lo que si o si necesita estar en Modo Kernel para su ejecución, y todo lo que pueda ser ejecutado en Modo Usuario se ejecutará en ese modo.

Procesos

Un **proceso** tiene un ciclo de vida que va desde que se inicia hasta que termina. No es lo mismo que un programa, pues un programa existe en memoria desde que se edita hasta que se borra.

Un **proceso** es una entidad que se define en un SO para abstraer la ejecución de un programa. Solo un proceso se encontrará activo en cualquier instante (con una sola CPU). Para poder ejecutarse necesita código, datos y stack (datos temporales, ej: Pila).

Atributos de un proceso

- ID del proceso (nro) y de su proceso padre.
- ID del usuario que lo disparó.
- Grupo que lo disparó.
- En ambientes multiusuario, desde qué terminal y quién lo disparó.
- Etc...

Todos esos atributos y muchos más se almacenan en una estructura de datos llamada **PCB (Process Control Block)**.

PCB (Process Control Block)

La **PCB** es una estructura en donde se guardan los atributos y punteros a memoria que indican dónde está el área de código, el área de datos, info del stack, etc..

Cada proceso tiene su propia y única PCB, y es lo **primero** que el SO crea para empezar a inicializar el proceso, y es lo **último** que se tiene que eliminar cuando el proceso termina.

Espacio de direcciones de un proceso

Es el conjunto de direcciones de memoria que ocupa un proceso y que este puede utilizar para mantener su información (código, datos, stack). No incluye su PCB.

- En **Modo Usuario**, un proceso puede acceder solo a su espacio de direcciones.
- En **Modo Kernel**, un proceso puede acceder a estructuras internas o espacios de direcciones de otros procesos.

Contexto de un proceso

Toda la información que el SO necesita mantener para administrar un proceso, y la CPU para ejecutarlo correctamente (registros de la CPU, registro PC, prioridad del proceso, si tiene E/S pendiente, etc.).

Un proceso podría ser sacado de la CPU por algún motivo para meter otro proceso. Eso se llama **Cambio de Contexto**. Cuando esto pasa, se debe resguardar el contexto del proceso saliente, que pasará a espera y retorna después a la CPU. Luego, se carga el contexto del nuevo proceso, y se comienza desde la instrucción siguiente a la última ejecutada en dicho contexto. Es tiempo no productivo de la CPU, y ese tiempo depende del HW.

Estados de un proceso

- **Nuevo**: Se empieza a inicializar todo lo que necesita el proceso para ejecutarse.

- **Listo**: Cuando tiene todo lo necesario para ejecutarse, queda a la espera de que se le asigne CPU.

- **Ejecución**: Se le asignó CPU al proceso. La tendrá hasta que se termine el periodo de tiempo asignado, hasta que termine, o hasta que necesite realizar alguna tarea de E/S.

- **En Espera**: El proceso no requiere CPU porque está esperando a que se complete una tarea que mandó a hacer (E/S, señal por parte de otro proceso, etc). Sigue en memoria, pero no tiene la CPU. Cuando se complete, volverá a estado Listo.

- **Terminado**: El proceso termina, destruyendo todo lo que preparó durante su inicialización.

Colas en la planificación

Las **PCB** se enlazan en **colas** siguiendo un orden determinado, en función del estado de cada proceso. Hay una o varias colas para cada estado de un proceso.

En general, el SO mantiene una cola de **todos** los procesos, más allá del estado de cada uno.

Las colas enlazan las PCB de los procesos. Las PCB no van paseando por la memoria en función de su estado.

Módulos de la planificación

Partes del kernel que realizan distintas tareas y que están relacionadas a la planificación de los procesos en función de los estados. Se ejecutan ante determinados eventos (Creación/Terminación de un proceso, Eventos de E/S, etc).

Los principales módulos son:

- **Planificador de largo plazo (Long Term Scheduler)**: Admite procesos desde el estado Nuevo al estado de **Listo**. El loader es la parte de la planificación que se encarga de cargar el proceso elegido en la memoria, para que pueda pasar a competir por la CPU.

- **Planificador de corto plazo (Short Term Scheduler)**: Selecciona cual de los procesos en estado de **Listo** va a tomar la CPU. El **dispatcher** es la parte de la planificación que realiza la tarea de cambio de contexto de la CPU.

- **Planificador de mediano plazo (Medium Term Scheduler)**: Baja procesos de memoria al **área de Swap** para bajar el grado de multiprogramación, y además se encarga de seleccionar qué proceso puede volver a subir a memoria.

Grado de multiprogramacion

Cantidad de procesos en memoria listos para ejecutarse. No siempre es bueno tener un alto grado, a veces es necesario bajarlo para evitar problemas con la administración de memoria, por ejemplo.

Planificación de la CPU

Permite determinar cuál de todos los procesos en estado **Listo** es el que va a ser seleccionado para tomar la CPU. Para esto, existen **algoritmos de planificación**. Hay 2 tipos:

- **Apropiativos (Preemtive)**: Proceso puede ser expulsado de la CPU en caso de ser necesario.

- **No Apropiativos (Non-Preemtive)**: Proceso se ejecuta en CPU hasta que él mismo abandone la CPU por su cuenta.

Hay diferentes algoritmos de planificación que se pueden emplear, dependiendo de la meta:

- **Procesos Batch**: No existen usuarios que esperan una respuesta en una terminal. El proceso arranca, se ejecuta, termina, y a lo sumo deja algún registro en algún lugar sobre el resultado de la ejecución. Pueden utilizarse algoritmos no apropiativos, pues va a arrancar y terminar el solo.

Busca maximizar el rendimiento y minimizar el tiempo de retorno, por lo que va a tratar de mantener a la CPU ocupada la mayor cantidad de tiempo posible. Ejemplos:

- *FCFS: First Come First Served*
- *SJF: Shortest Job First*

- **Procesos Interactivos**: Hay una interacción con el usuario, con servidores, etc. Son necesarios algoritmos apropiativos para evitar que un proceso acapare la CPU.

  Busca lograr un rápido tiempo de respuesta para cumplir con las expectativas del usuario. Ejemplos:

  - *RR: Round Robin*
  - *Prioridad: Ordenar los procesos por algún criterio, para poder darle la CPU a los procesos que están esperando hace más tiempo para no generar inanición.*
  - *Colas Multinivel. Los ordena en colas por prioridad, y dentro de cada cola, aplica un algoritmo RR.*
  - *SRTF: Shortest Remaining Time First*

Creación de Procesos

Un proceso es creado por otro proceso. Un proceso padre tiene uno o más procesos hijos, formando un árbol de procesos. Cuando se crea un proceso:

- Se crea la PCB
- Se le asigna un PID (Process ID)

- Se le asigna memoria para regiones de text (código), datos (variables) y stack (pila).
- Se crean estructuras de datos asociadas (fork)

Creación de procesos:

- **En Unix**:

    system call fork() // Crea un nuevo proceso.
    system call execve() // Carga un nuevo programa en el espacio de direcciones. Por lo general, se usa después del fork().

- **En Windows**:

    system call CreateProcess() // Crea un nuevo proceso y carga el programa para ejecución.

Terminación de procesos:

- exit // Se retorna el control al SO.

  *Nota: El proceso padre puede esperar recibir un código de retorno por medio del comando **"wait"**. **"wait"** bloquea al proceso, dejándolo a la espera de un valor de salida de un proceso hijo (exit 0, exit 1, exit 2, etc) antes de poder continuar ejecutando el resto de su código.*

- kill // Proceso padre puede terminar la ejecución de sus hijos con este comando.

- execve() // Cuando se ejecuta este comando, se borra el contenido del espacio de direcciones de ese proceso, es

decir, el programa entero que contiene ese proceso. Se cambia por otro programa que se envía como parámetro al “execve()”. El proceso que lo ejecuto no cambia, sigue teniendo el mismo PID.

Relacion Padre-Hijo

Con respecto a la ejecución, podemos tener 2 modelos diferentes:

- El padre puede continuar ejecutándose concurrentemente con su hijo, compitiendo ambos por la CPU.
- El padre puede esperar a que el proceso hijo termine para continuar la ejecución.

Con respecto al espacio de direcciones:

- En Unix, el hijo es un duplicado del proceso padre (ocupa la misma cantidad de memoria lógica que el padre cuando se crea). Los datos son propios de cada proceso, pues la protección entre procesos sigue existiendo. No pueden acceder a los espacios de direcciones del otro. Tienen PID diferente. Contienen el mismo código.
- En Windows, se crea el proceso y se le carga dentro el programa.

Procesos Cooperativos vs Independientes

- **Independientes**: El proceso no afecta ni puede ser afectado por la ejecución de otros procesos. No comparte ningún tipo de dato.

- **Cooperativo**: El proceso afecta y es afectado por la ejecución de otros procesos en el sistema.

Administración de Memoria

Los programas y datos deben estar en la memoria principal para poderlos ejecutar y ser referenciados directamente. El kernel es el encargado de organizar y administrar los espacios de direcciones y cómo se van ubicando los programas y datos en la memoria principal para alcanzar un alto grado de multiprogramación.

Se espera siempre un uso eficiente de la memoria con el fin de alojar el mayor número de procesos. El SO debe:

- Llevar un registro de las partes de memoria que se están utilizando y aquellas que no.
- Asignar espacio en memoria principal a los procesos cuando la necesitan.
- Liberar espacio de memoria asignada a procesos que han terminado.
- Lograr que el programador se abstraiga de la alocación de los programas.
- Brindar seguridad entre los procesos para que unos no accedan a secciones privadas de otros.

- Para procesos que usan los mismos datos, librerías o código, el SO debe brindar la posibilidad de acceso a estos compartiendo determinadas secciones de la memoria.
- Garantizar la performance del sistema.

La memoria física se divide de forma lógica para permitir alojar múltiples procesos. Esta división depende del mecanismo provisto por el HW.

Requisitos de HW para Administración de Memoria

- **Reubicación**: Los procesos son independientes de la ubicación de su espacio de direcciones de memoria. Mientras un proceso se ejecuta, puede ser sacado y traído a la memoria (Swap), y posiblemente sea colocado en una dirección diferente que antes.

  Por esto, las referencias a memoria se deben traducir según la ubicación actual del proceso, para poder encontrarlo esté donde esté en cualquier momento.

- **Protección**: Los procesos NO deben referenciar ni acceder a direcciones de memoria de otros procesos, a no ser que tengan permiso. Esto se chequea durante la ejecución.

- **Compartición**: Se debe permitir que varios procesos accedan a la misma porción de memoria si es que ambos la necesitan para trabajar (datos, librerías, rutinas en común, etc). Evitando copias innecesarias y repetidas se aprovecha mejor el espacio de memoria.

Espacio de Direcciones

Rango de todas las posibles direcciones que un proceso puede direccionar durante su ejecución. El tamaño de ese rango depende de la arquitectura del procesador.
Ej: CPU 32bits = [0..(2^32) - 1], CPU 64bits = [0..(2^64) - 1]

El SO está en todos los espacios de direcciones, es decir, ocupa parte de cada uno de los espacios de direcciones de los procesos.

Tipos de Direcciones

- **Lógicas**: Representan una dirección dentro de un espacio de direcciones, sin importar
donde se encuentra en la memoria principal.

- **Físicas**: Representan una dirección real y física dentro de la memoria principal.

Hay 2 tipos, por eso siempre que se necesite referenciar una dirección dentro de un espacio de direcciones, es necesario hacer una conversión. Para esto tenemos dos registros auxiliares:

- **Registro Base**: Dirección de comienzo del espacio de direcciones del proceso en la
memoria principal.

- **Registro Límite**: Dirección final del espacio de direcciones del proceso en la memoria principal.

Ambos valores se fijan cuando se carga el espacio de direcciones del proceso en memoria.

MMU (Memory Managment Unit)

Es HW. Encargada de realizar la conversión. Cada vez que hay un Cambio de Contexto en la CPU, el SO le re-programa en la MMU los registros base y límite para que esta pueda realizar la conversión. Solo se puede hacer en Modo Kernel.

Si la CPU trabaja con direcciones lógicas, para acceder a la memoria principal se deben transformar en direcciones físicas (address-binding).

Particiones

La memoria principal se va particionando lógicamente para alojar procesos. Tipos:

- **Fijas**: La memoria se divide en particiones de tamaño fijo (de igual o distinto tamaño), capaces de alojar un proceso solamente.

- **Dinámicas**: Las particiones varían en tamaño y número. Cada una se genera en forma dinámica y del tamaño justo que necesita el proceso. Son capaces de alojar un proceso solamente.

Fragmentación

Se produce cuando una localidad de memoria no puede ser utilizada por no encontrarse de forma contigua. Tipos:

- **Interna**: Es la porción de la partición que queda sin utilizar. Se produce en el esquema de particiones fijas.

- **Externa**: Son huecos que van quedando en la memoria a medida que los procesos terminan. Al no encontrarse de forma contigua, puede darse el caso de que tengamos memoria libre para alocar un proceso, pero que no la podamos usar. Se produce en el esquema de particiones dinámicas.

Para solucionar estos problemas de fragmentación generados por el esquema de Registro Base + Registro Límite, llegan la Paginación y la Segmentación.

Paginación

- La **memoria lógica** (espacio de direcciones) se divide en trozos de igual tamaño, llamados **Páginas**.
- La **memoria física** se divide lógicamente en pequeños trozos de igual tamaño, llamados **Marcos**.

Cualquiera de las páginas de un espacio de direcciones puede ir a parar a cualquiera de los marcos. De esta forma ya no necesitamos tener un espacio de direcciones contiguo.

Es necesario que el SO mantenga una **Tabla de Páginas** por cada proceso, donde cada entrada contiene (entre otras) el Marco en el que se coloca cada página. De esta forma se podrá mantener un orden lógico interno para cada espacio de direcciones, y se podrá acceder a los mismos sin importar en qué Marcos de la memoria principal se encuentren distribuidas sus páginas.

Bits de Control:

Cada entrada en la **Tabla de Páginas** tiene bits de control (entre otros):

- Bit V (Validez): Indica si la página está en memoria.
- Bit M (Modificado): Indica si la página fue modificada. Si se modificó en algún momento se deberán reflejar los cambios en memoria secundaria.

Segmentación

- El programa se divide en partes/secciones.
- Un programa es una colección de segmentos. Un segmento es una unidad lógica, es decir, representa una parte del programa (Programa Principal, Procedimientos, Funciones, Variables, etc).

Cada uno de esos segmentos puede ir a parar a cualquier parte de la memoria principal, y pueden ser de distinto tamaño.

Por eso, además de una **Tabla de Segmentos**, será necesario guardar las direcciones Base y Límite de donde se cargó cada segmento.

Vuelve el problema de la fragmentación, pues los segmentos tienen tamaños distintos y se van cargando y descargando de memoria principal de forma dinámica.

Segmentación Paginada

Combinación de ambas técnicas. Cada **segmento** es dividido en **páginas** de tamaño fijo. Tendremos **segmentos dentro de páginas** del mismo tamaño. De esta forma se puede aprovechar lo mejor de ambas técnicas.

Sobre Tabla de Páginas:

Cada proceso tiene su propia Tabla de Páginas. Su tamaño dependerá del espacio de direcciones del proceso.
Existen varias formas de organizarla:

- Tablas de Nivel 1 (Tabla Única Lineal)
- Tabla de Nivel 2 (o más, Multinivel)
- Tabla Invertida (Hashing)

Existen varias formas, porque para direcciones de muchos bits sería imposible guardarla en la RAM, pues ocuparía demasiada memoria. Existe un esquema de direccionamientos indirectos en las Tablas Multinivel, donde se divide la Tabla Lineal en múltiples Tablas de Páginas, cada una con el mismo tamaño pero con un menor número de páginas por tabla.

La Tabla Invertida es la versión inversa a la que veníamos viendo. Hay una sola tabla para todo el sistema, con una entrada por cada marco de página en la memoria real. El

espacio de direcciones de la tabla se refiere al espacio físico de la RAM, en vez del espacio de direcciones virtuales de un proceso. Mantiene un registro de las páginas que están en memoria física.

Tamaño de Página – Características según tamaño:

PEQUEÑO	**GRANDE**
Menor Fragmentación Interna.	Mayor Fragmentación Interna.
Más páginas requeridas por proceso, resultando en Tablas de Página más grandes.	Memoria secundaria puede transferir grandes bloques de datos eficientemente, permitiendo mover páginas a memoria principal más rápido.
Más páginas pueden residir en la memoria.	Menos páginas pueden residir en la memoria.

Memoria Virtual:

No es necesario que un proceso esté en su totalidad siempre en la memoria principal. El SO puede traer a memoria las partes de un proceso a medida que se necesita. La parte del espacio de direcciones de un proceso que está en la memoria es conocida como **“Conjunto Residente”** o **“Working Set”**. Con el apoyo del HW, se detecta cuando se necesita una parte que no está en el Working Set.

Para Memoria Virtual necesito:

- Que el HW soporte Paginación por Demanda (que se puedan meter páginas a medida que se necesiten).
- Un dispositivo o memoria secundaria que dé el apoyo para almacenar las secciones del proceso que no están en memoria principal (SWAP).
- El SO debe ser capaz de manejar el movimiento de páginas entre memoria principal y secundaria.

Ventajas del uso de Working Set:

- Más procesos pueden mantenerse en la memoria principal, y será más probable que existan más procesos “Ready”, mejorando el rendimiento.
- Un proceso puede ser más grande que la memoria principal, pues solo son cargadas algunas secciones de cada proceso.

Fallo de Página (PF):

Ocurre cuando el proceso intenta usar una dirección que está en una página que no se encuentra en la memoria principal en ese momento (Bit V= 0), es decir, la página no está en el Working Set. El proceso de atención de un Fallo de Página es el siguiente:

1. El HW detecta la situación y genera un trap al SO. El SO podrá colocar en “Espera” al proceso mientras gestiona que la página que se necesite se cargue.

2. Busca un Marco libre en la memoria y genera una operación de E/S al disco para copiar la página que se necesita en dicho Marco.
3. Cuando la operación de E/S termina, el SO actualiza la Tabla de Páginas del proceso, colocando el Bit V en 1 para la página en cuestión, y colocando la Dirección Base del Marco donde fue ubicada.
4. Finalmente, el proceso que generó el Fallo de Página vuelve a estado de “Ready”.

Nota: Si los Fallos de Página son excesivos, el rendimiento del sistema decae.

Translation Lookaside Buffer (TLB):

Es una memoria caché pequeña en la CPU, ligada a la MMU. Contiene las entradas de la tabla de páginas que fueron usadas más recientemente. Funcionamiento:

- Dada una dirección virtual, la CPU examina la TLB y:
    - Si la entrada de la tabla de páginas se encuentra en la TLB, se obtiene el Marco y se arma la dirección física.
    - Si no se encuentra en la TLB; el número de página es usado como índice en la tabla de página del proceso. Luego, se controla si la página está en la memoria, y la TLB se actualiza para incluir la nueva entrada. Si la página no está en la memoria se genera un PF.

Nota: Los cambios de contexto generan la invalidación de las entradas de la TLB.

Políticas en el Manejo de Memoria Virtual:

- Asignación de Marcos: La cantidad de páginas de un proceso que se pueden encontrar en memoria es igual al tamaño del conjunto residente.

    - Asignación Dinámica: El número de Marcos para cada proceso varía.
    - Asignación Fija: El número de Marcos para cada proceso es fijo.
    - Equitativa: Se dividen los Marcos por la cantidad de procesos.
    - Proporcional: Se asigna conforme al tamaño del proceso.

- Reemplazo de Páginas: Si ocurre un PF y todos los Marcos están ocupados, se debe seleccionar una página víctima. El reemplazo óptimo sería remover una página que no será referenciada en un futuro próximo. Para predecir esto, se basa en el comportamiento pasado.

    Alcance del Reemplazo:

    - Reemplazo Global: El PF de un proceso puede reemplazar una página de cualquier proceso. Así, puede tomar Marcos de otros procesos y aumentar la cantidad de Marcos asignados a él. No se controla la tasa de PF de cada proceso.

- Reemplazo Local: El PF de un proceso solo puede reemplazar una página propia. No cambia la cantidad de Marcos asignados. Favorece el control de tasa de PF de un proceso.

Algorítmos de Reemplazo:

- ÓPTIMO: Es solo teórico.
- FIFO: Se reemplaza la página que lleva más tiempo en memoria.
- LRU: Se reemplaza la página que hace más tiempo que no son referenciadas.
- 2DA CHANCE: FIFO que beneficia a las páginas más referenciadas. Si el Bit de Referencia está en 1, en lugar de salir, vuelve a la cola con el Bit de Referencia en 0.

Demonio de Paginación:

Proceso creado por el SO durante el arranque. Apoya la administración de la memoria y se ejecuta cuando el sistema tiene una baja utilización o cuando algún parámetro de la memoria lo índica. Tareas:

- Limpiar páginas modificadas, sincronizándolas con la SWAP.
- Mantener un cierto número de marcos libres en el sistema.
- Demorar la liberación de una página hasta que haga falta realmente.

Memoria Compartida:

Gracias al uso de la Tabla de Páginas, varios procesos pueden compartir un marco en memoria. Ese Maro debe estar asociado a una página en la Tabla de Páginas de cada proceso. Los procesos pueden compartir un copia del código (editores de texto, compiladores, etc), pero los datos son privados a cada proceso y se encuentran páginas no compartidas.

Mapeo de una Archivo en Memoria:

Técnica que permite a un proceso asociar el contenido de un archivo a una región de su espacio de direcciones virtuales. Cuando se genera un PF, el contenido de la página es obtenido directamente desde el archivo asociado. Cuando el proceso termina o el archivo se libera, las páginas modificadas son escritas en el archivo asociado.
Permite realizar E/S de una manera alternativa a usar operaciones directamente sobre el Sistema de Archivos.

Área de Intercambio (SWAP):

Área en la memoria secundaria para intercambiar la memoria física con la virtual.

Dos técnicas diferentes para la administración:

1. Cada vez que se crea un proceso, se reserva una zona de SWAP igual al tamaño de imagen del proceso, y se le asigna la dirección en disco de la zona a ese proceso.

2. No se asigna nada inicialmente. A cada página se le asigna su espacio en disco cuando se va a SWAPear, y el espacio se libera cuando la página vuelve a memoria. El problema que trae es que se debe llevar un control en memoria de la localización de las páginas en disco.

Hiperpaginación (Thrashing):

Un sistema está en Thrashing cuando pasa más tiempo paginando que ejecutando procesos, causando una baja importante de rendimiento en el sistema.

Ciclo de la Hiperpaginación:

1. El Kernel monitorea la utilización de la CPU.
2. El kernel detecta que la utilización de la CPU es baja, aumenta el grado de multiprogramación.
3. Aparecen nuevos procesos que necesitan Marcos para cargar sus páginas. Si el algoritmo de reemplazo es global, estos nuevos procesos podrían sacarle Marcos a otros.
4. Procesos que necesitan páginas ya no las tienen. Generan PF.
5. Comienza un ciclo constante de PF y robo de Marcos.
6. Impacta a la E/S: Las colas de espera de los dispositivos empiezan a saturarse de peticiones de lectura de páginas, bajando el uso de la CPU.
7. Vuelve al paso 1.

Soluciones para controlar la Hiperpaginación:

- Usar algoritmos de reemplazo local, así si un proceso entra en Hiperpaginación no roba Marcos a otros procesos. Perjudica el rendimiento del sistema, pero es controlable.
- El SO debería tratar que, en todo momento, un proceso tenga todas las páginas que necesita en memoria, así no habría Hiperpaginación, pues se reducen los PF. Para esto hay dos técnicas:

    - Técnica de Working Set: Se basa en el modelo de localidad: un programa se compone de varias localidades, las cuales están formadas por instrucciones y datos que, cuando una de ellos sea referenciada, probablemente las demás también serán referenciadas.
      Esta técnica trata de mantener las localidades de un proceso en memoria que han sido referenciadas más recientemente, mirando una ventana (Δ “delta”) que indica cuales fueron las referencias de memoria más recientes.

      Problemas:

        - Valor de Δ:
            - Δ Chico: Puede que no cubra lo que realmente es la localidad del proceso.
            - Δ Grande: Puede que queden referencias que ya no forman parte de la localidad en mi Working Set.

- Técnica de Page Fault Frequency (PFF):
  En lugar de calcular el WS de los procesos, utiliza la tasa de PF para estimar si el proceso tiene un conjunto residente que representa adecuadamente al WS.

    - Si la PFF es alta, quiere decir que se necesitan más Marcos.
    - Si la PFF es baja, quiere decir que los procesos tienen muchos Marcos asignados.

Subsistema de E/S:

Metas, Objetivos y Servicios:

- Generalidad: Es deseable manejar todos los dispositivos de E/S de manera estandarizada y uniforme.
- Eficiencia: Los dispositivos de E/S son más lentos que la memoria y CPU, pero haciendo uso de la multiprogramación un proceso puede quedar esperando la finalización de su E/S mientras otro proceso se ejecuta.
- Planificación: El SO es capaz de organizar los requerimientos a un dispositivo para que se use de manera eficiente.
- Buffering: Almacenamiento de los datos en memoria para solucionar problemas de velocidades entre dispositivos.
- Caching: Mantener en memoria una copia de los datos de reciente acceso para mejorar el rendimiento.
- Spooling: Administrar la cola de requerimientos de un dispositivo para coordinar el acceso concurrente al mismo (Ej: Impresora).

- Manejo de Errores: El SO debe administrar errores que puedan suceder y tomar las medidas necesarias para seguir adelante.
- Formas de hacer E/S:
    - Bloqueante: El proceso se suspende hasta que el requerimiento de E/S se complete. Fácil de usar y entender, pero no es suficiente bajo algunas necesidades.
    - No Bloqueante: El requerimiento de E/S retorna en cuanto es posible.

Diseño del SW del SO:

- Modo Usuario:
    - SW de Capa de Usuario: Los SO suelen dejar un conjunto de librerías y procesos de apoyo para que los procesos las utilicen sin requerir Modo Kernel.

- Modo Kernel:
    - SW Independiente del dispositivo: Brinda los principales servicios de E/S a los procesos de usuario. El kernel mantiene información de cada dispositivo.
    - Controladores de Dispositivos: Contienen el código dependiente del dispositivo, y manejan un tipo de dispositivo. Sabe interpretar, manejar y responder ante las interrupciones generadas por los dispositivos.
    - Gestor de Interrupciones: Atiende todas las interrupciones del HW y deriva al driver correspondiente la atención de cada interrupción.

- Rendimiento: E/S es uno de los factores que más afectan al rendimiento del sistema, pues utiliza mucho la CPU para ejecutar los drivers y provoca Context Switches ante las interrupciones y bloqueos de procesos. Para mejorar el rendimiento:
    - Reducir el número de Context Switches, tratando de resolver la E/S en el mismo proceso que está en ese momento en ejecución, sin importar de quién corresponda la E/S.
    - Reducir la cantidad de copias de los datos durante la atención de un requerimiento de E/S.
    - Reducir la frecuencia de interrupciones, por medio de transferencias de gran cantidad de datos o el uso de controladores más inteligentes.
    - Utilizar DMA, para quitarle la tarea a la CPU de ver si terminó la E/S constantemente, y así esta puede seguir realizando trabajo útil.

Sistema de Archivos

Archivo:

Entidad abstracta con nombre. Es un espacio lógico continuo y direccionable que permite a los programas guardar datos y a la vez provee a los programas de datos.

Necesitamos archivos para almacenar grandes cantidades de datos, a largo plazo, y para permitir a distintos procesos acceder al mismo conjunto de información.

Puntos de Vista:

DEL USUARIO	DEL DISEÑO
Qué operaciones se pueden llevar a cabo.	Implementar archivos.
Cómo nombrar un archivo.	Implementar directorios.
Cómo asegurar la protección.	Manejo del espacio en disco.
Cómo compartir archivos.	Manejo del espacio libre.
No tratar con aspectos físicos.	Eficiencia y mantenimiento.

Sistema de Archivos:

Conjunto de unidades de SW que proveen los servicios necesarios para la utilización de archivos. Facilita el acceso a los archivos por parte de las aplicaciones, y permite al programador abstraerse en cuanto al acceso de bajo nivel.

Objetivos del SO en cuanto a archivos:

- Cumplir con la gestión de datos.
- Cumplir con las solicitudes del usuario.
- Buscar minimizar y eliminar la posibilidad de perder o destruir datos, garantizando la integridad del contenido de los archivos.
- Dar soporte de E/S a distintos dispositivos.
- Brindar un conjunto de interfaces de E/S para tratamiento de archivos.

Tipos de Archivos:

- Archivos Regulares: Texto Plano o Binarios.

- Directorios: Archivos que mantienen una estructura y un orden en el Sistema de Archivos. Contiene info de archivos y directorios dentro de él. Es, en sí mismo, un archivo.

Atributos de Archivos:

- Nombre.
- Identificador: Número o algo que lo identifica unívocamente dentro del Sistema de Archivos.
- Tipo.
- Localización: Información necesaria para ubicar el contenido del archivo en el medio de almacenamiento.
- Tamaño.
- Protección, Seguridad y Monitoreo: Owner, permisos, password.
- Fecha de Creación, Último Acceso, Modificación.

Operaciones de Directorios:

- Buscar un archivo.
- Crear un archivo.
- Borrar un archivo.
- Renombrar archivos.
- Listar el contenido.
- Etc..

Ventajas del Uso de Directorios:

- Eficiencia: Localización rápida de archivos.
- Uso del Mismo Nombre: Diferentes usuarios pueden tener el mismo nombre de archivo.
- Agrupación Lógica por Propiedades o Funciones (Juegos, Librerías, Imágenes, Aplicaciones Java, etc..).

Estructura de Directorios:

Los archivos pueden ubicarse siguiendo un “path” desde el directorio raíz, seguido por todos los directorios que aparecen en el medio. De esta forma, distintos archivos pueden tener el mismo nombre, pero el “path” es único.

Al directorio actual se lo conoce como el Directorio de Trabajo (Working Directory). Dentro de este los archivos se pueden referenciar por su:

- Path Absoluto: Ruta completa desde la raíz.
  Ej: “/var/www/index.html”
- Path Relativo: Ruta del archivo desde el Directorio de Trabajo. Ej: Si estoy en “/var/spool/mail”, entonces es: “../../www/index.html”

Compartir Archivos:

En un ambiente multiusuario, se necesita que varios usuarios puedan compartir archivos. Debe ser realizado bajo un esquema de protección.

Protección y Derechos de Acceso:

El administrador debe ser capaz de controlar qué se puede hacer, quién lo puede hacer, y los derechos de acceso de directorios. Tiene todos los derechos y puede darlos a otros usuarios.

Permisos:

- Lectura: El usuario puede ejecutar.
- Añadir (Appending): El usuario puede agregar datos, pero no modificar el contenido o borrarlo.
- Ejecución: El usuario puede ejecutar.
- Actualizar (Updating): El usuario puede modificar, borrar o agregar datos. Incluye creación de archivos y sobrescritura.
- Cambiar Protección: El usuario puede modificar los derechos de acceso.
- Borrado: El usuario puede borrar el archivo.

Algunos conceptos:

- Sector: Unidad de almacenamiento usada en discos rígidos.
- Cluster: Conjunto de sectores consecutivos.
- File System: Define la forma en que los datos son almacenados.
- FAT (File Allocation Table): Diferente de FAT32. Contiene información sobre en qué lugar están alocados los distintos archivos.

Formas de Asignación:

- Continuo: Los sectores o clusters que representan el contenido de un archivo se almacenan de forma continua en el medio de almacenamiento. La FAT es sencilla en cuanto a la cantidad de información que mantiene, pues contiene el número del primer cluster o sector y la longitud para cada archivo. Puede ocurrir Fragmentación Externa, pues pueden quedar sectores continuos sin usar. Se debe conocer el tamaño del archivo durante su creación (Preasignación).

- Encadenada: Los clusters o sectores que representan el contenido de un archivo mantienen un enlace entre ellos. Cada entrada de la FAT contiene el número del primer cluster o sector y la longitud. Cada cluster o sector tiene un puntero al siguiente. No hay Fragmentación Externa, pues no hay continuidad en los sectores en los que se guarda el archivo. El tamaño del archivo puede crecer bajo demanda.

- Indexada: Hay un bloque que contiene punteros a los índices de los sectores o clusters que representan el contenido de un archivo. La FAT contiene la información sobre dónde se encuentra ese bloque de índices. No hay Fragmentación Externa, pues no hay continuidad en los sectores en los que se guarda el archivo. El acceso aleatorio es eficiente. Hay dos variantes:

    - Asignación Indexada por Secciones: A cada entrada de la FAT del bloque índice se le agrega el campo longitud. Así, el índice apuntará al primer sector o

cluster de un conjunto almacenado de forma continua.

- Asignación Indexada con Niveles de Indirección: (I-Nodos) Existen bloques directos de datos, y bloques que son considerados bloques índices que apuntan a varios bloques de datos. Pueden haber varios niveles de indirección.

Gestión de Espacio Libre:

El Sistema de Archivos lleva el control sobre cuáles de los bloques de disco están disponibles.

- Tabla de bits: Tabla con 1 bit por cada bloque de disco que representa su estado (0: libre, 1: en uso). Es fácil encontrar un sector o cluster libre, pero el tamaño de la tabla resulta inconveniente a la hora de cargarla en memoria para leer y guardarla en disco para escribir constantemente.

- Bloques Encadenados: Se tiene un puntero al primer bloque libre, y cada bloque libre tendrá un puntero al siguiente. Si se pierde un enlace, puede generar inconsistencias. Se necesita de varias operaciones de E/S para obtener un grupo de bloques libres, por lo que es ineficiente para la búsqueda de bloques libres, y más difícil aún encontrar bloques libres consecutivos.

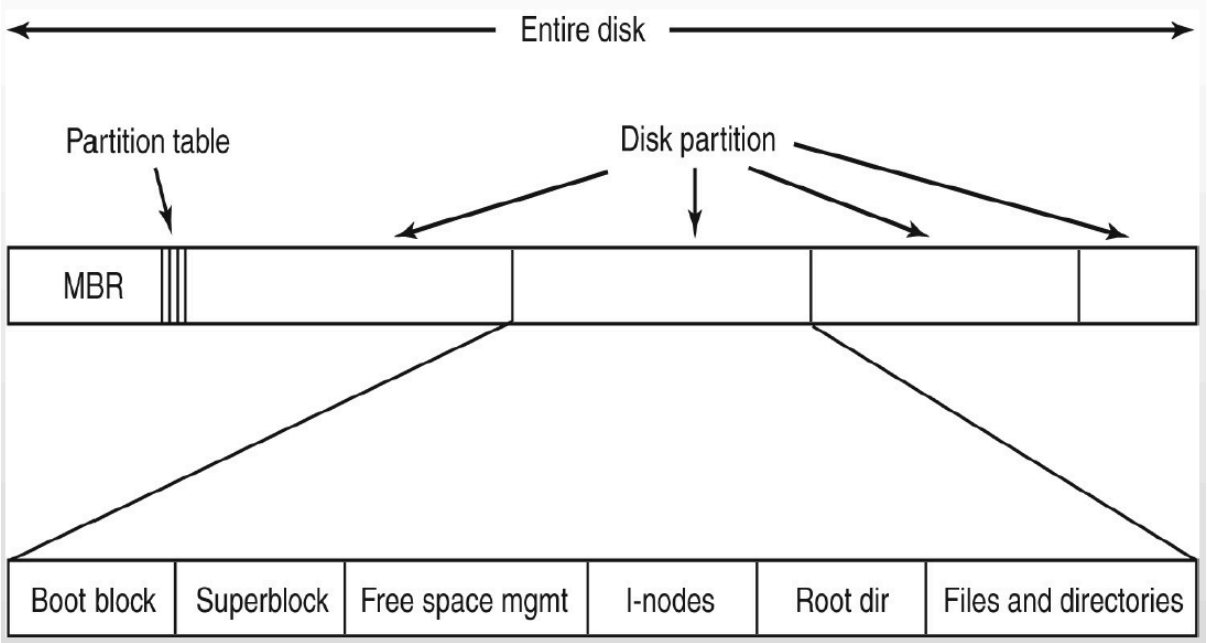
- Indexación: Variante de Bloques Encadenados. El primer bloque libre tiene las direcciones de N bloques libres. Las N-1 primeras direcciones son bloques libres, y la N-ésima

dirección referencia otro bloque con N direcciones de bloque libres.

Sistema de Archivos en Unix:

Cada disco físico puede ser dividido en una o más particiones. Cada partición tiene un SIstema de Archivos. Cada Sistema de Archivos tiene:

- Boot Block: Código para bootear el SO.
- Superblock: Atributos sobre el Sistema de Archivos (Bloques/Clusters libres)
- Tabla de I-Nodos: Tabla que contiene todos los I-Nodos, los cuales son estructuras de control que tienen la información clave de un archivo.
- Data Blocks: Bloques de datos de los archivos.



I-Nodos:

Estructuras de datos que mantienen los atributos de los archivos. No mantiene el nombre. Contiene:

- Información de los permisos.
- Tamaño.
- Fecha de Creación, Modificación, etc..
- Propietario.
- Direcciones a los datos del archivo.
- Cantidad de Enlaces al I-Nodo.

Al formatear, la tabla de I-Nodos se crea entera. Los I-Nodos no aparecen y desaparecen dinámicamente. La tabla de I-Nodos representa la cantidad máxima de archivos que puedo tener. La tabla se crea entera al formatear, pero las estructuras de direcciones e indirecciones que guarda para cada archivo se van creando a medida que ese archivo crece.

Si la cantidad de enlaces de un archivo es cero, quiere decir que no tengo forma de llegar, es decir, el I-Nodo que correspondería a ese archivo está libre. Es posible llegar al mismo archivo a través de distintas rutas siempre y cuando haya más de un enlace al I-Nodo.

Caché de Disco:

Conjunto de buffers en memoria principal que implementa el SO para almacenamiento temporario de bloques de disco, con el objetivo de reducir la frecuencia de acceso al disco.

Cuando un proceso quiere acceder a un bloque de la caché, hay dos alternativas:

- Copiar el bloque al espacio de direcciones del usuario. Pero, si otro proceso también lo quisiera leer, no podría porque el bloque está en el espacio de direcciones de otro proceso.
- Trabajarla como memoria compartida, teniendo todos los bloques en un espacio compartido, permitiendo así el acceso a varios procesos.

Algoritmo de Reemplazo:

Cuando se necesita un buffer para cargar un nuevo bloque, se elige el que hace más tiempo no fue referenciado.

Caché de Disco en Unix System V:

Un buffer cache tiene dos partes:

- Header: Contiene información del bloque, número, estado, relación con otros bloques, etc..
- El buffer en sí: El lugar donde se almacena el bloque de disco traído a memoria.

Cuando un bloque y/o header se mueve a memoria, quedan ahí. No cambian según el estado del bloque, sino que lo que cambian son los enlaces entre esas estructuras.

El Header tiene:

- Información del número de bloque almacenado en ese buffer, e información del dispositivo que corresponde a ese bloque de datos.
- Punteros: 2 para la Hash Queue, 2 para la Free List, 1 para el bloque en memoria.
- Estado:
    - Disponible: Puede que esté vacío porque todavía no se usó, o que esté disponible porque un bloque lo usó y lo liberó, pero el bloque sigue estando en memoria.
    - No Disponible: Es un bloque que aún está usando un proceso. Aún no lo liberó.
    - Escribiendo o Leyendo del Disco: Indica que el bloque que está en ese buffer se está leyendo o escribiendo, para no tomarlo para hacer otra cosa durante la operación.
    - Delayed Write: Indica que hubo una modificación en el bloque en memoria, pero los cambios aún no han sido reflejados en el bloque original en disco.

Los punteros:

- Free List: Enlaza los Headers que están en estado Disponible. Organiza los buffers disponibles para ser utilizados para cargar nuevos bloques de disco. Se ordena por LRU. Cuando se necesita un buffer, se usa el que está primero en la Free List.
- Hash Queue: Son colas para optimizar la búsqueda de un buffer en particular. Se utilizan para determinar si un bloque que se necesita está o no en un buffer. Al número de bloque se le aplica una función de Hash que permite

agrupar los buffers cuyo resultado dio igual para hacer que las búsquedas sean más eficientes.

Funcionamiento del Caché de Disco:

1. Cuando un proceso quiere acceder a un archivo, usa su I-Nodo para localizar los bloques de datos donde se encuentra éste.
2. El requerimiento llega al buffer caché, quien evalúa si puede satisfacer el requerimiento o si debe realizar la E/S.
3. Se pueden dar 5 escenarios:
    - El Kernel encuentra el bloque en la Hash Queue y el buffer está libre en la Free List.
    - El Kernel no encuentra el bloque en la Hash Queue y usa un buffer libre.
    - Idem 2, pero el bloque está marcado como Delayed Write.
    - El Kernel no encuentra el bloque en la Hash Queue y la Free List está vacía (no hay buffers libres).
    - El Kernel encuentra el bloque en la Hash Queue, pero está No Disponible.