

Módulo 1 - Introducción a los sistemas operativos

Sistema Operativo

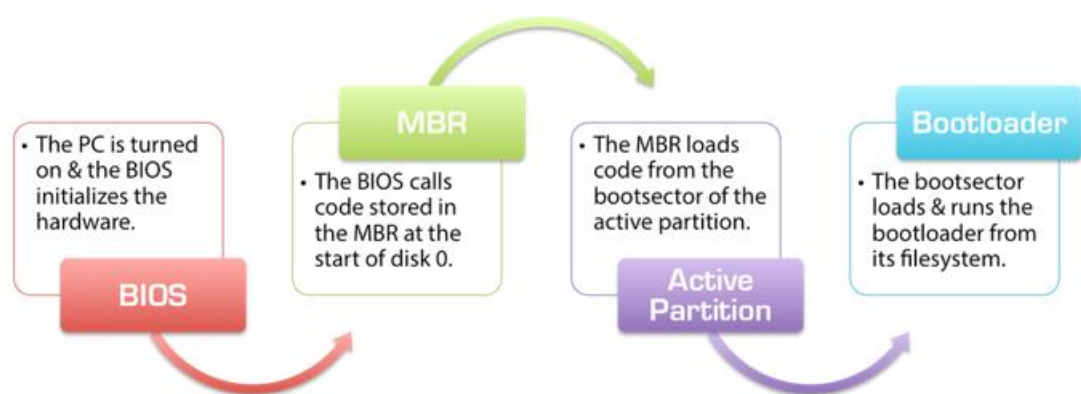
Es un SW que administra el HW, administra recursos físicos, y recursos lógicos. El mismo tiene que estar instalado en una computadora y en funcionamiento. Permite abstraer al usuario de la complejidad del HW.

Es el principal programa que se ejecuta en toda computadora de propósito general.

- Conjunto de módulos o funciones (SW), que, instalados en las computadoras, se ocupan de controlar y administrar la ejecución de los programas sobre los recursos que brinda el equipo (HW), tales como: memoria, procesador, periféricos, etc.
- Conjunto de programas que ordenadamente relacionados entre sí, contribuyen a que la computadora lleve a cabo correctamente su trabajo para nosotros en un ambiente dado.

Es el único programa que interactúa directamente con el hardware de la computadora. Sus funciones primarias son:

- **Inicialización:** Este proceso es desde que el HW se enciende hasta que se llega al login. Prepara la maquina y la lleva a un estado tal que pueda ejecutar el primer trabajo. Existen el cold boot (es la que se usa) y el warm boot; difieren en las tareas de control realizadas previa carga del SO. Prácticamente en la actualidad no hay diferencia. Menos del 1 a 3% de las tareas del SO se destinan a esta función. A mediados de los 90s surge UEFI como sucesor de la BIOS.



La **BIOS** (Basic Input Output System) es un **firmware** que se ejecuta apenas se enciende la computadora. Su función inicial es **verificar el estado del hardware** (como la memoria disponible, el funcionamiento del procesador, etc.) y luego **delegar el control al sistema de booteo**.

Cuando la computadora se enciende, el **registro de próxima instrucción** no contiene dirección alguna, por lo que apunta automáticamente al **sector 0 del disco**, donde se encuentra la rutina de inicialización de la BIOS. Por lo tanto, el sistema carga la BIOS en la CPU y comienza a ejecutarla.

Una vez finalizadas las verificaciones de hardware, la BIOS carga el programa que se encuentra en el **bootloader**, el cual accede al **MBR (Master Boot Record)**, generalmente ubicado en un sector especial del disco primario de la computadora. Desde allí se carga el **sistema operativo (SO)**.

El proceso de carga del sistema operativo se divide en dos etapas:

1. La **etapa monousuario**, que corresponde a la carga base del sistema.
2. La **etapa multiusuario o multitarea**, si el sistema operativo lo soporta.

Una vez completadas estas etapas, se considera que el sistema operativo está **activo y en control** de la computadora.

El **MBR** contiene una **tabla de particiones**, donde se definen las divisiones del disco. La **partición activa** es aquella que se utiliza por defecto para iniciar el sistema. Es importante notar que:

- No es obligatorio tener una partición para tener un sistema operativo.
- Tampoco es obligatorio que haya un solo sistema operativo por partición.
- **Abstracción o maquina extendida (interface hombre maquina):** Los programas no deben tener que preocuparse de los detalles de acceso a hardware, o de la configuración particular de una computadora. El SO se encarga de proporcionar estas abstracciones (Ej: la información organizada en archivos o directorios en uno o muchos dispositivos de almacenamiento). Facilita la comunicación con el usuario y acepta entradas de nuevos trabajos (ejecución de los nuevos programas). Típicamente se clasifican en: GUI, CLI, NUI (natural user interface). El SO no solo debe proveer las abstracciones necesarias, sino que también, ninguno de sus usuarios puede evadir dichas abstracciones. (3 a 10% del código del SO)
- **Administración de recursos (memoria, espacio de almacenamiento, tiempo de procesamiento, etc.):** Es la función principal de cualquier SO. La administración de recursos significa ser capaz de verificar que cada uno de los módulos, HW que administra el sistema operativo, este funcionando correctamente. Si bien los recursos generalmente están relacionados con un HW, no siempre es el caso. Puede pasar que sea una función de un HW más una política de uso. Por ejemplo, los semáforos (recurso que administra el SO y no depende de un HW).

El SO tiene a su disposición una gran cantidad de recursos y los diferentes procesos compiten por ellos. El SO puede implementar políticas que los asignen de forma efectiva y acorde a las necesidades establecidas para dicho sistema. Más del 80% del código está dedicado a esta función. Si el SO estableció determinada política de asignación de recursos, debe evitar que el usuario exceda las asignaciones aceptables, sea durante el curso de su uso normal, o incluso ante patrones de uso oportunista. (80% o más)

- Aislamiento (seguridad): No nos referimos a la seguridad intrínseca de cada módulo, como los mecanismos internos de protección de memoria que evitan que un usuario acceda a datos que no le corresponden. En cambio, hablamos de un nivel superior de seguridad: prevenir que un atacante tome control del sistema, como ocurre con ransomware, virus, exploits, etc.

En un sistema multiusuario y multitarea, cada usuario y proceso debe poder operar de forma independiente, sin verse afectado por otros usuarios que estén utilizando el mismo sistema. Para lograr ese nivel de aislamiento, el sistema operativo recurre a hardware específico de protección.

El objetivo principal es garantizar la integridad de los recursos y de los procesos, así como también validar la identidad de los usuarios que acceden al sistema.

Este aislamiento evita que un proceso malicioso acceda o interfiera con la información de otro usuario o del propio sistema.

En la actualidad, muchos sistemas operativos incluyen herramientas adicionales de defensa para mitigar ataques, tales como:

- Actualizaciones automáticas del sistema.
- Software antimalware preinstalado.
- Firewalls de aplicaciones (application firewalls).
- Listas de control de acceso (ACLs) y permisos granulares.

Si el sistema operativo implementa correctamente la separación de datos, procesos y recursos entre distintos usuarios, ninguno de ellos debería poder acceder a información que otro haya marcado como privada.

Requerimientos en el HW

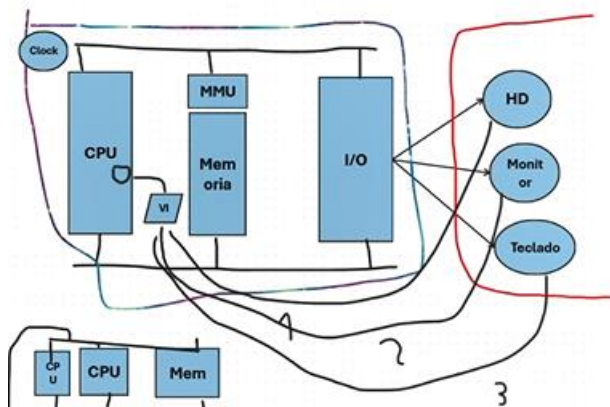
Modo Dual de operaciones / Modo kernel – modo usuario

Originalmente, la seguridad se refería únicamente a la integridad del propio sistema operativo y la protección entre usuarios. Los mecanismos fundamentales para lograr esto son: Modo dual de ejecución de procesador (modo usuario y modo kernel), el cual cambia según el estado de un bit que indica el modo de operación. El modo de ejecución kernel da acceso a un juego de instrucciones diferenciadas del Procesador

Interrupciones

Interrupción de HW externo

En general el procesador después de preservar los contenidos de todos los registros y cierta información acerca del estado del proceso reemplaza el contenido del contador de programa (Program counter), por la dirección de donde se encuentra la RAI.



PSW (program status Word): es un área de la memoria o registro que contiene información sobre el estado de un programa utilizado por el sistema operativo.

FLIH (first level interruption handler): este bit le avisa al procesador que hay una interrupción.

Los distintos dispositivos mediante su IRQ asignado envían la solicitud de interrupción al manejador de interrupciones, el preprocesador, el cual se encarga de manejar las interrupciones. Mediante el INT (interruption) envía la señal de interrupción al procesador, en ese momento en el PSW, el cual tiene un bit llamado FLIH cambia el valor de este bit a 1, avisándole al procesador que hay una interrupción. El procesador a su vez, que está ejecutando una instrucción (las mismas son atómicas), la finaliza y lo primero que hace es verificar el FLIH. Detecta la interrupción, resguarda el contexto de lo que se está ejecutando hasta ese momento y le responde al manejador de interrupciones mediante el INTA (interruption acknowledge), que se ejecutara la RAI. El manejador de interrupciones entrega la RAI. El SO ejecuta la RAI en el procesador, cuando se comienza a ejecutar la RAI el FLIH vuelve a ponerse en cero. Cuando se finaliza la RAI, el SO vuelve a traer el contexto del proceso que se había sacado y lo sigue ejecutando.

Clasificación según prioridad

Las interrupciones pueden organizarse por prioridades, de modo que una interrupción de menor jerarquía no interrumpa a una más importante. Dado que las interrupciones muchas veces indican que hay datos disponibles en algún buffer, el no atenderlas a tiempo podría llevar a la pérdida de datos. Hay un número limitado de interrupciones definidas para cada arquitectura. Las interrupciones son generadas por el controlador de canal en que son producidas. El SO puede elegir ignorar (enmascarar) ciertas interrupciones, pero hay algunas que son no enmascarables. El enmascarado o no enmascarado de las interrupciones, depende de lo que estoy haciendo. Por ejemplo, si estoy ejecutando un proceso del usuario, todas las interrupciones de HW son no enmascarables porque cualquier interrupción de HW es más importante que atender al usuario. **El vector de interrupciones nos da el orden de atención de las interrupciones.**

No enmascarables: Son las de mayor prioridad del sistema. Cuando se detecta una interrupción de este tipo, el SO detiene lo que está haciendo y se ocupa de atender la interrupción. *Este tipo de interrupción puede detener hasta la ejecución de rutinas modo kernel.* En general se utilizan para la notificación de errores irre recuperables por parte de algún HW.

Enmascarables: Son interrupciones de menor prioridad dentro de las interrupciones, pero siempre tienen mayor prioridad a cualquier proceso de usuario. Este tipo de interrupciones puede ser interrumpida por una interrupción de mayor prioridad.

0	1	2	3	4	5	6	7								
			8	9	10	11	12	13	14	15					

Prioridad del vector: 0, 1, 9, 10, 11, 12, 13, 14, 15, 3, 5, 6, 7 (el 2 comunica con el 8 y no se ocupan)

Clasificación según su origen

- **Software:** se denominan así a las llamadas del sistema (syscalls) que hacen los procesos. Si bien no son eventos que el procesador debe reconocer, se categorizan así porque en definitiva se interrumpe el código de usuario para dar lugar a la atención de dicha llamada al SO. **Detienen solo al proceso que ejecutó la llamada al sistema.** Son las interrupciones de menor prioridad.
- **Hardware:** Interrupciones generadas por algún componente físico del sistema. Se subdividen en:
 - **Internas (Traps):** se producen dentro del procesador, en un componente del procesador (división por cero, direccionamiento incorrecto). **Detienen al recurso más importante, al CPU. No necesito el vector de interrupciones ni el FLHI porque el CPU ya conoce la excepción.** Solo queda comunicarle al programa que ejecuto la instrucción que generó el trap.
 - **Externas:** fuera del entorno del procesador (finalización de I/O, falta de papel, error de dispositivos). **Detienen al dispositivo y al proceso.**

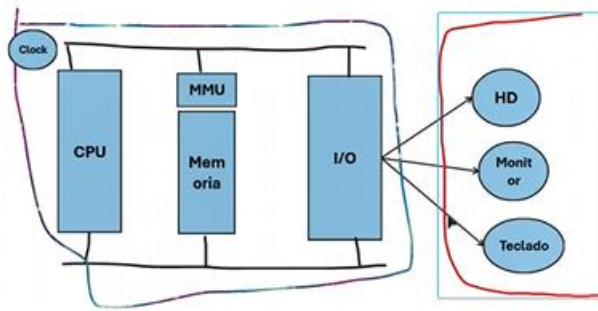
Bloques UMA / NUMA

Cuando tengo gran poder de CPU, nuestro gran cuello de botella pasa a ser los buses.

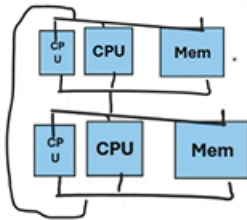
La comunicación y la interacción entre la CPU, el módulo E/S y la memoria principal / MMU se da de manera sincrónica (mundo sincrónico) siguiendo los pulsos del clock (HW externos que emite pulsos), todos estos dispositivos trabajan en el mismo orden de velocidad. Los dispositivos externos conectados al módulo de E/S están en el mundo asincrónico.

Multiprocesamiento simétrico

- Uniform Memory Access (UMA): Varios procesadores compartiendo la memoria con igualdad de condiciones. Utilizan un BUS compartido, por lo que se debe administrar el uso. Cuando una CPU usa el bus la otra espera o hace un ciclo interno.



- Non-Uniform Memory Access (NUMA): cada procesador tiene afinidad con bancos específicos de memoria. Pueden acceder a su memoria local más rápido que a la memoria de otros procesadores o compartida. El objetivo es mejorar la performance. El acceso a la memoria se realiza a través de los buses, por lo que algunos de estos buses serán más rápidos para ir a una parte establecida de la memoria, y otros buses a otras partes de la memoria.



Notar que cada CPU tiene buses que las comunican con la memoria afín, más fácilmente. Se pueden comunicar a otra memoria, pero tardará más y además le deberá pedir el servicio al conector. No sólo puedo planificar por CPU sino también por bloque NUMA.

Cuando el SO detecta que un bloque NUMA tiene demasiada carga, baja los procesos que se están ejecutando en este bloque y los traslada a otro bloque. Esto, genera overhead y trata de evitarse, por eso el SO, cuando crea los procesos y los va subiendo a la memoria, los intenta subir ya balanceados para evitar este trabajo administrativo de movimiento de carga. Si se decide un cambio debe hacerse cuando el overhead será menos costoso que seguir procesando de esa manera. Hasta en ocasiones, un procesamiento cruzado es más costoso que bajar un proceso y subir a otro bloque NUMA. El SO debe saber administrar los bloques NUMA, sino sería inútil utilizarlos.

Con menos de 16 CPUs normalmente no es beneficioso tener bloques NUMA y que se busque tener alta performance. Tener bloques NUMA implica tener el doble de buses y un mecanismo de sincronización entre ellos. Normalmente se usan en servidores.

DMA (Direct Memory Access)

Es un canal de trabajo.

- El módulo de E/S y la memoria central intercambian datos directamente, sin involucrar a la CPU.
- El DMA requiere un módulo adicional en el bus del sistema.
- El módulo de DMA es capaz de simular el procesador y tomar el control del sistema desde la CPU.
- Cuando el procesador desea leer o escribir un bloque de datos, emite un comando al módulo de DMA, enviándole la siguiente información:
 - 1) Si la operación solicitada es una lectura o escritura
 - 2) La dirección del dispositivo de E/S involucrado
 - 3) La posición inicial en memoria a leer o a escribir
 - 4) El número de palabras a ser leído o escrito.

Con esos datos el DMA toma el control del bus de sistema y hace la transferencia de la información entre la memoria y el módulo de E/S. El procesador le envía estos 4 datos al DMA y sigue procesando otras operaciones. Cuando la operación ha terminado, el DMA enviará la interrupción para que el procesador siga procesando. Esta interrupción se realizará cuando los datos ya estén en el lugar que les corresponde (por ej en memoria)

Sin el DMA el acceso a memoria para el siguiente ejemplo sería:

- Necesito leer del disco. El módulo de entrada salida solicitará al disco un dato específico. El disco obtendrá los datos y los depositará en un buffer del módulo entrada salida. El módulo de E/S activa el vector de interrupciones. La CPU atiende la interrupción y ella misma toma los datos del buffer y los envía a memoria principal para ser procesado (esto hace que ocupe varios ciclos en esta tarea de transporte).

Con DMA:

- Cuando mi módulo de E/S tiene DMA, sabe ingresar a la memoria. Necesito leer del disco. El módulo de entrada salida solicitará al disco un dato específico y le indicará donde colocarlo en memoria. El disco obtendrá los datos y los depositará en un buffer del módulo entrada salida. El módulo de E/S usando los buses intentará llevarlo a la memoria directamente.

Cómputo distribuido

Se refiere al trabajo realizado por computadoras independientes (o bien, procesadores que no comparten memoria). Los tipos más comunes son:

Clústers: Computadoras conectadas a una red local de alta velocidad, cada una ejecutando su propia instancia de SO. Se ven como un único equipo de cómputo.

Grids: Computadoras heterogéneas distribuidas geográficamente e interconectadas a una red. Se diseñan para adaptarse a enlaces de baja velocidad.

Computo en la nube: se refiere a la terciarización de servicios. La implementación de los servicios deja de ser relevante. Se aplican conceptos como:

SaaS: se ofrece la aplicación completa.

PaaS: se ofrece un entorno completo de desarrollo o despliegue de aplicaciones.

IaaS: se ofrece un HW completo (real o virtual) pero con gran flexibilidad para modificar sus recursos.

Tipos de SO:

- Monousuario: un solo usuario
- Multiusuario: más de un usuario trabajando simultáneamente con la computadora.

Según la cantidad de procesos que soporta:

- Uniprocador: este SO es capaz de manejar solamente un procesador de la computadora, de manera que si la computadora tuviese más de uno le sería inútil.
- Multiprocador: Son sistemas capaces de administrar más de un procesador, compartiendo memoria y periféricos. La mayoría de los SOs actuales se diseñan para sacar provecho del multiprocesamiento.

Según la cantidad de procesos que ejecutan concurrentemente:

- Monoprogramados o monoproceso: Solo se puede ejecutar un proceso a la vez. Recién cuando éste finalice, se puede ejecutar otro proceso. Se los conoce también como SOs monotarea.
- Multiprogramados: También llamados multitarea o multitask, se refieren a SOs capaces de maximizar el uso del procesador. Cuando un proceso se encuentra haciendo uso de algún dispositivo, se otorga el procesador a otro proceso.
- La ejecución concurrente no es lo mismo que la ejecución paralela. Para esto último se requieren 2 o más procesadores y un SO capaz de administrarlo.

Según sus aplicaciones:

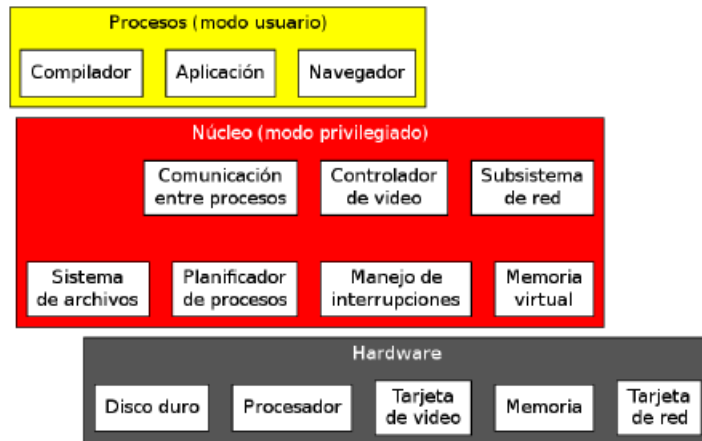
- De propósito general: Son los que proporcionan una amplia gama de servicios y deben adaptarse a cualquier ambiente, tipo de aplicaciones, modos de operación, dispositivos, etc.

- De propósito especial: Construidos a medida debido a arquitecturas especiales o aplicaciones con requerimientos especiales como control de proceso industriales. Históricamente se clasifican como:
 - Sistemas de tiempo real: Debe garantizar la respuesta a eventos externos dentro de los límites de tiempo preestablecidos. Son muy usuales en todo lo que es la industria. Están pensados para escenarios en donde sea necesario el manejo de tiempo por excepciones. El tiempo pasa a ser muy importante y el SO debe garantizar que da respuestas dentro del tiempo pactado.
 - Sistemas tolerantes a fallas: se utiliza en aplicaciones donde se debe proveer un servicio continuo. Se suele utilizar un conjunto de redundancias en recursos y chequeos internos. El SO detecta y corrige errores (que no son de hardware). Sistemas espaciales, sistemas de seguridad en el área nuclear, base de datos online. Puede reaccionar ante roturas del HW.
 - Sistemas virtuales o virtualizadores: SOs capaces de administrar y gestionar otros SOs que ejecutan bajo su órbita en forma concurrente usando el mismo HW. Nacen en los 60s, posteriormente esta tecnología cae en desuso, ya que el HW se hacía más accesible económicamente. Hacia fines de los 90s vuelve a surgir el concepto y hoy en día prácticamente todos los centros de datos utilizan virtualización.

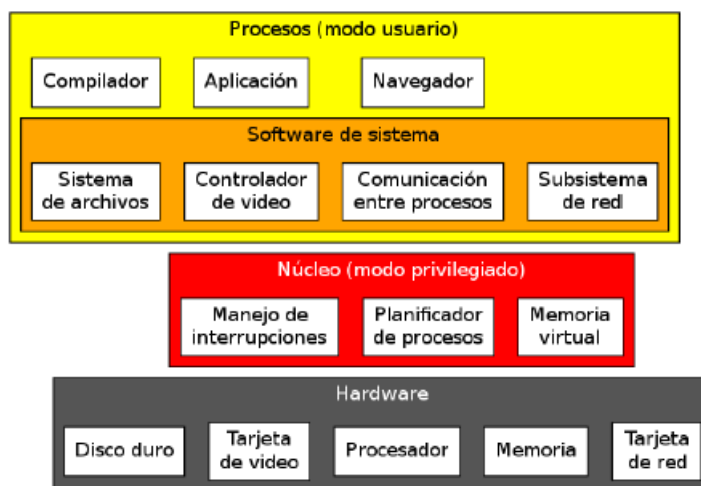
Arquitectura de los sistemas operativos

A este nivel el SO es un gran programa, que ejecuta otros programas y les provee un conjunto de interfaces para que puedan aprovechar los recursos de cómputo.

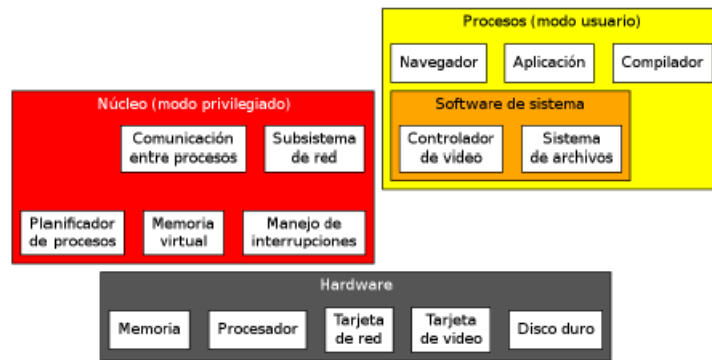
- Sistemas monolíticos: Se tiene un único módulo de SW que opera en modo privilegiado, dentro del cual se encuentran las rutinas requeridas para las distintas tareas realizadas por el SO (toda la lógica del SO se encuentra en un único módulo). Al no requerir muchos mecanismos de comunicación, ofrecen una buena performance de ejecución. Se hace más difícil la actualización y el mantenimiento del SO (si tengo que actualizar porque tengo un dispositivo nuevo debo recompilar el kernel).



- **Sistemas de microkernel:** El núcleo del SO se mantiene en el mínimo posible de funcionalidad. Poseen una lógica más limpia y resulta más simple el reemplazo de componentes. Pueden auto repararse con mayor facilidad, dado que en caso de fallar uno de los componentes, el núcleo puede reiniciarlo o incluso reemplazarlo. Mediante los drivers (componente SW) le añadimos funcionalidad al SO. Si bien no toca el kernel interactúa mediante APIs con el kernel, el driver se ejecuta en modo kernel. Como el modo dual de ejecución nos obliga a que para interactuar con el HW tenemos que ejecutar en modo kernel, dicho driver ejecuta instrucciones privilegiadas (pedimos acceso al SO).



- **Sistemas híbridos:** Mayormente monolíticos pero que manejan algunos procesos que parecerían centrales mediante procesos de nivel de usuario como los microkernel.



Módulo 2: De Programas a Procesos

Programa

Conjunto **ordenado** de instrucciones que pretenden resolver un problema. Es una **entidad pasiva porque no está en memoria central**. Es una serie de códigos, algo estático.

Instrucción

Unidad de ejecución que dura un tiempo finito y se ejecuta sobre un procesador, no necesariamente es indivisible (a nivel procesador sí, pero en un lenguaje de alto nivel no es así).

Proceso

Es **dinámico**, tiene “vida” y nació a partir de un programa.

Porción de un programa **cargado en Memoria Central al cual se le asocia su contexto de ejecución (run time environment) mediante una estructura de datos llamada vector de estado o Bloque de Control del Proceso** (Process Control Block – PCB). Un proceso sólo sabe trabajar sobre memoria central.

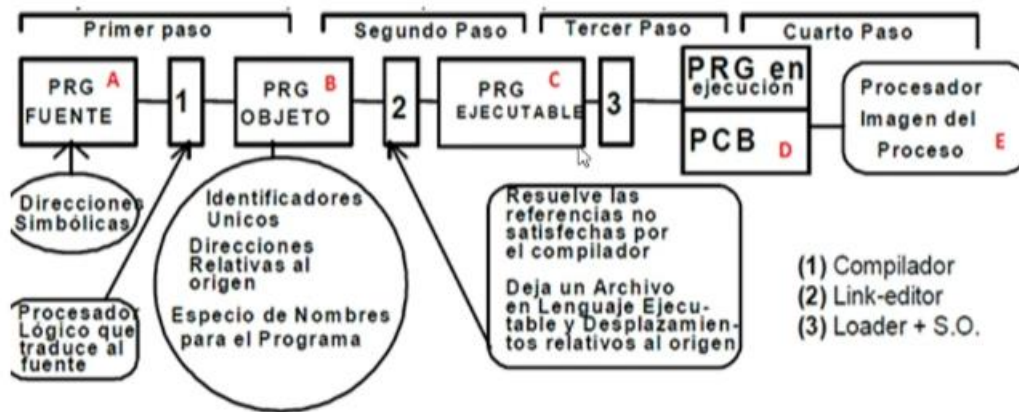
Runtime environment: todo lo que el proceso tiene como para poder ejecutar- datos y recursos asignados, permisos, etc.-).

Un proceso consiste en una secuencia de acciones llevadas a cabo a través de las instrucciones cuyo resultado es el de proveer alguna función del sistema

Resumiendo, **un proceso es una secuencia de acciones y es dinámico (entidad activa), mientras que un programa es una secuencia de instrucciones y es estático (entidad pasiva).**

Es una entidad activa, tiene runtime environment, recursos asignados.

Compilación y carga de un programa



- Partimos del programa con lenguaje de alto nivel, con direcciones simbólicas, migrables entre plataformas y con referencias sin resolver.
- Al compilar traduzco el código en alto nivel en un código objeto. Hay parte en código binario y parte que no, es un lenguaje intermedio. Las direcciones son relativas al origen. No resuelve ninguna referencia (las funciones, variables o símbolos externos usados en el código fuente). Es migrable entre plataformas. Cuando quiero compartir mi código, no comparto el código fuente sino el objeto y de esa manera se puede compilar y no estoy compartiendo el código. Muchas veces no son descriptables.
- El link-editor resuelve las referencias no satisfechas por el compilador. Deja un archivo en lenguaje ejecutable y desplazamientos relativos al origen. El servicio de traducción es estático y siempre está en el mismo lugar. En ese momento ya tengo un ejecutable. Tengo un lenguaje binario, tengo direcciones reales o relativas al origen, tengo las referencias resueltas, no es migrable entre plataformas.
- El loader lo carga en memoria. Dependiendo de la tecnología de este loader puede solamente cargarlo o, cargarlo y además traducirlo (si el SO sabe interpretar direcciones relativas al origen, las traducciones a direcciones reales se hacen en tiempo de ejecución, sino ya genera las direcciones reales). Además, crea el PCB y a veces carga al programa en memoria, total o parcialmente (dependiendo del método de administración de memoria). La salida del loader es el proceso.

- Cuando llegamos al proceso tenemos un archivo binario, direcciones reales o relativas al origen, tengo las referencias resueltas. No es migrable entre plataformas.

Trabajos, procesos y threads

- Trabajo (Job): Lenguaje de alto nivel que ejecuta paso a paso distintos procesos (un Shell en un script es un Job). En los hosts es obligatorio crear jobs para hacer ejecuciones. El Job genera procesos. Son optativos para nuestros SO.
- Un proceso (Process): El proceso puede generar otro proceso o distintos threads. Son Obligatorios. Es la imagen en memoria de un programa junto con la información relacionada con el estado de su ejecución.
- Un Hilo o Hebra (thread) también llamado proceso liviano, es un trozo o sección de un proceso que tiene sus propios registros, pila, program counter y puede compartir la memoria con todos aquellos threads que forman parte del mismo proceso. Optativos en el caso de Linux, obligatorio en el caso de Windows

Administración de procesos

Los datos sobre el proceso se guardan en una estructura de datos llamado vector de estado o PCB (Process control block). Está en memoria privada del sistema operativo y el único que puede acceder al PCB es el SO (puedo acceder haciendo un syscall). El PCB guarda para cada proceso la información necesaria para reanudarlo (cuando es suspendido o desalojado del uso del procesador) además de otros datos. El SO mantiene para cada proceso un bloque de control o Process control block (PCB). La información contenida en el PCB varía de SO en SO. El PCB de los procesos activos está en un área de la memoria específica administrada por el SO.

Contenido del PCB

- Identificación (PID única en el sistema): identifica unívocamente el proceso en el sistema.
- Identificadores varios del proceso (identificador del dueño, padre PPID, hijos, etc.). Todos los procesos tienen un padre, al nivel más alto está el SO.
- Estado (ejecutando, listo bloqueado)
- Program counter (PC): guarda la dirección de la próxima instrucción, es un registro del procesador.
- Registro de CPU (acumuladores, program status Word).
- Información para la planificación (Ej: prioridad): dependiendo de la metodología que se utilice para la planificación se guarda determinada información desde el procesador al PCB.
- información para administración de memoria (Ej: registros base y límite). También depende la metodología de administración es lo que se guarda.

- Información de I/O: dispositivos y recursos asignados al proceso, archivos abiertos en uso, etc.
- Estadísticas y otros: tiempo real y tiempo de CPU usado, etc.
- Privilegios (Ej: quien es el owner)
- Otros objetos vinculados al proceso.

Toda esta información la toma el procesador cuando arranca y en el momento que se detiene por alguna interrupción, se guardan todos estos estados en el PCB para cuando se retome el programa se pueda seguir el flujo.

Creación de procesos

Todos los procesos son creados por el SO. Sin embargo, puede ser útil permitir que un proceso pueda originar la creación de otros procesos (ejemplo: instrucción fork en C a través de un syscall). Cuando un proceso genera a otro, el proceso generador se conoce como proceso padre y el proceso generado es el proceso hijo. Normalmente estos procesos necesitan comunicarse y cooperar.

Existen dos tipos de creación

- Jerárquica: cada proceso que se crea es hijo del proceso creador y hereda el entorno de ejecución de su padre.
El padre continúa ejecutando concurrentemente (paralelo) con sus hijos
El padre espera a que todos sus hijos hayan terminado y luego sigue el.
- No jerárquica: El padre no espera a que todos sus hijos hayan terminado para terminar.

Motivaciones o razones para crear un proceso:

- Llega un nuevo trabajo al sistema: generalmente en forma de batch, entonces el SO debe escribirlo y comenzarlo a ejecutar creando una secuencia de procesos nuevos.
- Llegada de un usuario al sistema: entonces el SO ejecuta un proceso llamado login.
- Un servicio al programa de ejecución: creado por el SO; por ejemplo, realiza una lectura en disco (en este caso el proceso que solicitó el servicio es bloqueado).
- Por un proceso existente: por razones de modularidad o paralelismo.

Pasos del SO al crear un proceso:

1. Asignar un único identificador al nuevo proceso (PID)
2. Asignar espacio de memoria para el proceso
3. Inicializar el bloque de control de proceso (PCB)
4. Establecer los enlaces apropiados con otras estructuras de datos
5. Ampliar o crear otras estructuras de datos en el caso de que fueran necesarias.

Pasos cuando un proceso muere:

1. Desaparece el PCB
2. Recursos comunes son liberados
3. Recursos locales son eliminados

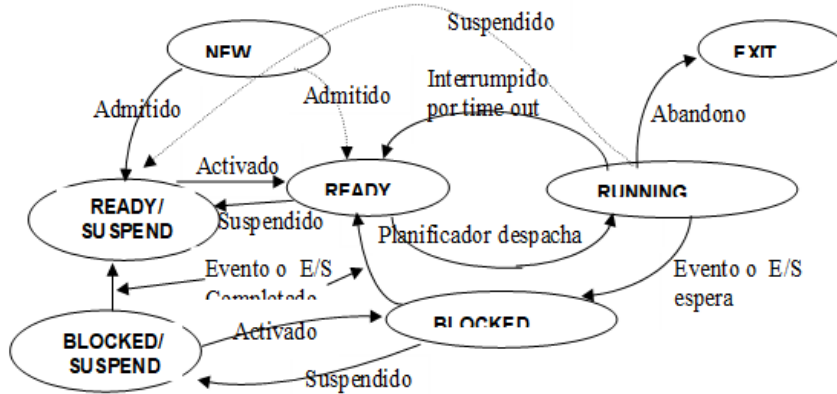
Cuando un proceso termina (muere) también deber terminar sus hijos (normal o anormalmente). Esto se conoce como terminación en cascada. Lo lógico sería que el padre espere la terminación de los hijos y entonces el termine, en definitiva, es una cuestión de cómo está programado el SO.

Ciclo de vida de un proceso

Diagrama de transición de estados de un proceso



Modelo de Siete Estados



Los estados agregados `READY/SUSPEND` y `BLOCKED/SUSPEND` corresponden a situaciones relativas a la administración de memoria

Lo que hace que funcione y se del ciclo de vida del proceso son las interrupciones.

- **Nuevo:** Se solicitó al SO la creación de un proceso y sus recursos y estructuras están siendo creadas (creando el PCB).
- **Listo:** Está listo para iniciar o continuar su ejecución, pero el sistema no le ha asignado un procesador. Está cargado en memoria con su PCB. Tiene todos los recursos para ejecutar menos el procesador. **De ejecución a listo puede salir por quantum, se da un context switch. De bloqueado a listo IHE.**
- **En ejecución:** el proceso está siendo ejecutado en este momento. Sus instrucciones están siendo procesadas en algún procesador. **Cuando paso de listo a running tengo un context switch.**
- **Bloqueado:** en espera de algún evento para poder continuar su ejecución (aún si hubiera un procesador disponible, no podría avanzar). Está en espera de recursos para continuar la ejecución. **De ejecución a bloqueado hay un syscall.**
- **Zombi:** El proceso finalizado su ejecución, pero el SO debe realizar ciertas operaciones de limpieza para poder eliminarlo de la lista.
- **Terminado:** El proceso terminó de ejecutarse, sus estructuras están a la espera de ser limpiadas por el SO. La finalización se la conoce como la muerte del proceso. Esta muerte puede ser normal (el proceso quiso terminar o termino controladamente), si es anormal (por algún trap). Kill -9 normal pero que no se

puede evitar, kill –15 normal se puede evitar, si hago un catch de un trap y termino, es muerte anormal. **De ejecutando a terminado hay context switch.**

Estados activos:

Son aquellos que compiten por el procesador o están en condiciones de hacerlo

- **Ejecución (Running):** Estado en el que se encuentra un proceso cuando tiene el control del procesador.
- **Listo o preparado (Ready):** Aquellos procesos que están dispuestos para ser ejecutados. Disponen de todos los recursos para su ejecución y aguardan su turno en una cola de listos(Ready Queue).
- **Bloqueado (Blocked):** Son los procesos que no pueden ejecutarse de momento por necesitar algún recurso no disponible (generalmente recursos de entrada / salida).

Estados inactivos:

Cuando un proceso está suspendido lo sacamos de memoria.

- **Suspendido bloqueado (Suspended - Blocked):** Es el proceso que fue suspendido en espera de un evento, sin que hayan desaparecido las causas de su bloqueo.
- **Suspendido listo (Suspended – Ready):** Es el proceso que ha sido suspendido, pero no tiene causa para estar bloqueado.

Las colas de Estados de los procesos:

Los PCB se almacenan en colas, cada una de ellas representa un estado (status) particular de los procesos.

Razones de un cambio de estado de proceso

- **Por interrupciones de HW externas.** (ejecución a listo, listo a ejecución (quantum)), bloqueado a listo
- **Por una excepción** (trap – interrupción HW interna) (ejecución a finalizado, listo a ejecución)
- **Por una llamada al sistema** (syscall) (ejecución a finalizado, listo a ejecución, ejecución a bloqueado)

El bloqueado es un estado en el cual yo pedí estar. No es que me obligaron a estar bloqueado.

Transiciones de estado

Todo proceso a lo largo de su existencia puede cambiar de estado varias veces.

Cada uno de estos cambios se denomina transición de estado.

Cada vez que entro o saco un proceso de la CPU estoy haciendo un context switch.

Ciclos de vida de los threads

Dentro de los diferentes estados, los Threads se encuentran en la parte correspondiente al short term schedule (planificador de corto plazo).

Entonces los estados en los que pueden estar un thread son: listo (spawn), bloqueado (block), ejecutando (running) o terminado (finish).

El estado bloqueado del thread no va a depender del thread sino de que tipo de thread sea. No tengo que cargarlo en memoria porque los threads comparten la memoria, no tengo que asignarle recursos (no existe el estado nuevo para un thread).



Hilo o Hebra (Threads)

- También llamado proceso liviano, debido a que mantiene la estructura de un proceso con su PCB, pero también de otra estructura más pequeña llamada TCB (Thread control block) que contiene una información reducida del PCB, lo que hace que se ejecute más eficientemente.
- En sí un proceso es igual a uno o más tareas (tasks) cada una con su Hilo (thread) asociado.
- Un thread es una unidad elemental de uso de CPU.
- Cada hilo posee TID (Thread Identifier) un Contador de programa (PC – Program Counter), un juego de registros de CPU (Register Set) y una Pila (Stack).
- En muchos sentidos, los hilos son como pequeños miniprocesos.
- Cada Thread se ejecuta en forma estrictamente secuencial compartiendo la CPU de la misma forma que lo hacen los procesos.
- Solo en un multiprocesador se pueden realizar en paralelo.
- Los hilos pueden crear hilos hijos y se pueden bloquear en espera de llamadas al sistema, al igual que los procesos regulares.
- Mientras un hilo está bloqueado se puede ejecutar otro hilo del mismo proceso. (Depende de la implementación, hay dos tipos de hilos)

- Puesto que cada hilo tiene acceso a cada dirección virtual (comparten un mismo espacio de direccionamiento, comparten PCB), un hilo puede leer, escribir o limpiar la pila de otro hilo.
- No existe protección entre hilos debido a que es imposible y no es necesario ya que cooperan entre sí la mayoría de las veces
- Aparte del espacio de direcciones, comparten el mismo conjunto de archivos abiertos, procesos hijos, relojes, señales, etc.
- Los procesos livianos dentro de un mismo proceso pesado no son independientes, pues cualquiera puede acceder toda la memoria correspondiente al proceso. Podrían comunicarse entre sí sin ningún proceso de IPC).
- Si un proceso está bloqueado no necesariamente implica que los hilos estén bloqueados. Esto depende de la implementación.

Ventajas respecto a los procesos

- Toma menos tiempo realizar el cambio de procesar un nuevo thread (del mismo proceso) – Context switch liviano.
- Comparten un mismo espacio de memoria y datos entre sí debido a que forman parte de un mismo proceso.

Implementación de Hilos (Threads)

Los hilos pueden ser implementados en tres niveles por la forma en que son generados y tratados:

Hilos a nivel de usuario (ULT)

Todo el trabajo del hilo es manejado por la aplicación, el SO sabe de la existencia de los hilos, pero **no los administra**, es obligación del proceso administrar los hilos.

El context switch es aún más liviano que un context switch de un hilo KLT porque ni siquiera tengo que hacer un syscall para realizarlo. Se hace como proceso usuario.

Cualquier aplicación puede ser programada para ser multithreaded mediante el uso de Threads library (paquete de rutinas para ULT)

La generación de los ULT se hace en el momento de compilación y no se requiere intervención del Kernel.

El kernel continúa con la planificación del proceso como unidad y le asigna un solo estado de ejecución (listo, corriendo, etc.).

A nivel usuario **cada hilo no tiene su propio program counter (PC)**

Ventajas

- El cambio de hilo no requiere el modo kernel

- El proceso no cambia al modo kernel para manejar el hilo.
- El algoritmo de planificación puede ser adaptado sin molestar la planificación del SO.
- ULT puede correr en cualquier SO.
- Es muy rápido en la ejecución.
- **No necesita mecanismos de IPC para comunicarse con otros Threads del mismo proceso.**

Desventajas

- En un SO típico, la mayoría de los system call son bloqueantes. **Cuando un hilo ejecuta un system call no solo se bloquea ese hilo, sino que también se bloquean todos los hilos del proceso.** Se puede utilizar jacketing (cuando un hilo de usuario hace un syscall, el SO lo posterga, espera que se ejecuten otros hilos de usuario y en el momento que los otros hilos hagan syscall bloquea y ejecuta todas las syscall en conjunto).
- **En una estrategia pura de ULT, una aplicación multithreaded no puede tomar ventaja del multiprocesamiento. Un kernel asigna un proceso a sólo un procesador por vez.**

Hilos a nivel de Kernel (KLT)

Todo el trabajo de manejo de hilos es hecho por el kernel. No hay código de manejo de hilo en el área de aplicación. Cualquier aplicación puede ser programada para ser multithreaded . Todos los hilos dentro de una aplicación son soportados dentro de un solo proceso. El kernel mantiene la información de contexto para el proceso e individualmente para los hilos dentro del proceso. **A nivel kernel cada hilo tiene su propio PC. Los cambios de proceso los hace el kernel. Los hilos KLT siempre usan jacketing.**

Ventajas

- **Simultáneamente el kernel puede planificar múltiples hilos del mismo proceso en múltiples procesadores**
- Si un hijo de un proceso se bloquea, el kernel puede planificar otro hilo del mismo proceso.
- Las rutinas mismas del kernel pueden ser multithreaded.

Desventaja

- **La transferencia de control de un hilo a otro dentro del mismo proceso le requiere al kernel un cambio de modo.**

Hilos a nivel de proceso (PLT)

En desuso.

Uso de los hilos

Estructura servidor trabajador: Existe un hilo en el servidor que lee las solicitudes de trabajo en un buzón del sistema, examina éstas y elige a un hilo trabajador inactivo y le envía la solicitud. El servidor despierta entonces al trabajador dormido (un signal al semáforo asociado)

Estructura de equipo: Todos son iguales y cada uno obtiene y procesa sus propias solicitudes.

Estructura de entubamiento (pipeline): El primer hilo genera ciertos datos y los transfiere al siguiente para su procesamiento. Los datos pasan de hilo en hilo y en cada etapa se lleva a cabo cierto procesamiento. Esta puede ser una buena opción para el modelo productor / consumidor, no así para los servidores de archivos. Posiblemente en una tarea estrictamente secuencial no sea ideal el uso de threads, puesto que no saco ventaja del multiprocesamiento, debo hacer un cambio de contexto por cada hilo y se pierde tiempo. En este caso posiblemente sea mejor el proceso pesado.

Cambio de contexto

Mecanismo mediante el cual el sistema almacena la información del proceso que se está ejecutando y pasa a ejecutar otra rutina. Puede haber un cambio de contexto, pero no un cambio de proceso (por ej. cuando se produce una interrupción). Context Switch liviano.

Solo cambio el PC y los registros de la ALU, lo dejo cerca de la CPU. **Se da con un cambio entre threads DEL MISMO PROCESO o cuando cambio entre un proceso usuario y un proceso kernell del sistema operativo (el caso del estado spawned).**

Como los procesos del SO no usan los mismos datos ni registros que los procesos usuarios, entonces ese cambio de contexto es liviano, no lleva mucho tiempo.

Cambio de proceso

Cuando el SO entrega a la CPU un nuevo proceso, debe guardar el estado del proceso que se estaba ejecutando, y cargar el estado del nuevo. Cuando hay un Process switch hay un context switch.

Un cambio de proceso implica un cambio de contexto. La mayoría de las veces el cambio de contexto se da con un cambio de proceso, pero no siempre. Cuando tengo que hacer un cambio de contexto entre 2 procesos, se carga todo el PCB. En cambio,

cuando tengo que atender una rutina no se utilizan todos los registros del procesador. Entonces libero algunos registros y el cambio de contexto es menor (hard switch).

Módulo 3 - Planificación de Procesos

Tipos de planificación

La planificación es necesaria para poder determinar el orden en que van a ingresar los procesos.

Reciben su nombre de acuerdo con la frecuencia con la que se ejecutan.

A largo plazo

Decide que procesos (o trabajos) serán iniciados y como finalizan. Se ejecuta periódicamente una vez cada varios segundos, minutos o incluso horas. Se encarga de pasar los procesos del estado de nuevo al estado de listo.

A mediano plazo (o planificador de Swapping)

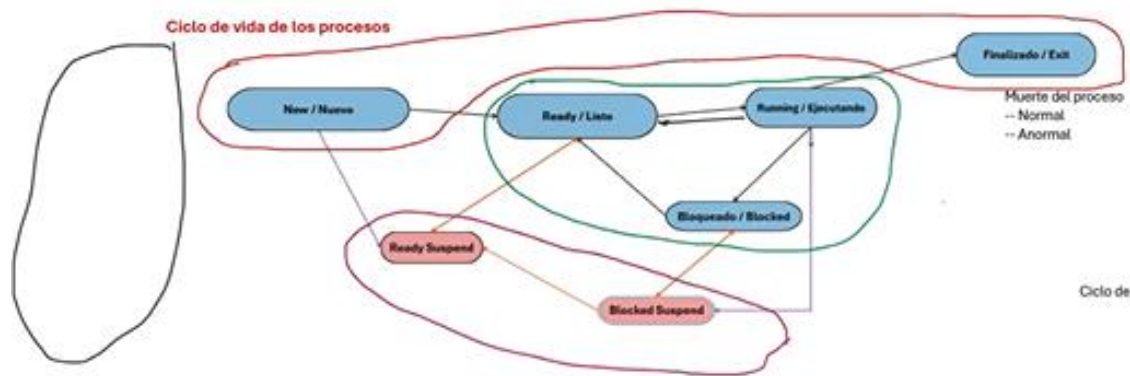
Decide que procesos suspender y despertar / activar. Esto ocurre debido a que los procesos típicamente se bloquean por escasez de algún recurso (típicamente la memoria principal o primaria). En algunas bibliografías también llamado agendador o scheduler. Está en el orden de los milisegundos. Cuando hay mucha carga de procesos activos en el sistema, el SO suspende a alguno de estos (suspendido bloqueado, suspendido listo) y los pasa a memoria secundaria. Puede ocurrir por algún problema con los recursos o más que nada cuando la memoria está muy cargada. Cuando hay demasiado procesos en memoria para ejecutar y hay baja de rendimiento, se ocupa de suspenderlos.

A corto plazo

Decide como compartir momento a momento la CPU entre los procesos que la requieren. Se ejecuta decenas de veces por segundo. Se lo conoce también como despachador o dispatcher. Funciona en el orden de los nanosegundos o milisegundos.

A extralargo plazo

Esta fuera del SO (la hace el administrador del sistema) pero va a afectar el tiempo de respuesta de los procesos. No afecta al ciclo de vida directamente, pero si afecta a cómo funcionan los procesos. Como va a priorizar los procesos o que tiempo de respuesta le dará a cada proceso. La prioridad de un proceso es una planificación a extralargo plazo.



Relación entre los planificadores y los estados de los procesos.

- El planificador de largo plazo admite nuevos procesos. Se encarga de la transición del estado de nuevo a listo.
- El planificador de mediano plazo maneja las transiciones entre estados suspendidos y estados activos y viceversa.
- El planificador de corto plazo administra las transiciones entre listo y ejecutando y bloqueado.

Algoritmos de planificación

Esquemas de planificación

- Sistemas cooperativos, no expropiativos o Non-Preemptive: Son aquellos en donde el SO no interrumpe al proceso en ejecución. Es éste, el que cede el uso del procesador, o bien a través de una instrucción Yield (ceder el paso) o debido a un Syscall (llamada al sistema). El SO no puede quitarle el uso de procesador al proceso en ejecución. Se atienden las interrupciones, pero no se desaloja al proceso.
- Sistemas expropiativos o Preemptive: Son aquellos donde el reloj (timer) del sistema interrumpe periódicamente el proceso en ejecución para devolver el control al SO y que este decida qué proceso será el próximo en ejecutarse. Se saca un proceso solo cuando se termina el Quantum y hay algún proceso en listo y tiene más prioridad. Esto se da sólo en la planificación a corto plazo.

Primero en llegar, primero servido (FCFS)

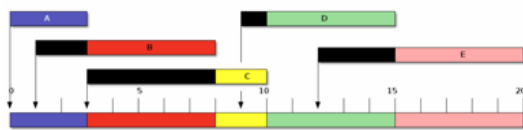
- Utiliza multitarea no expropiativos o non-preemptive
- Es el algoritmo más simple. El primero que llega a la cola de listos es el primero en ser atendido (First Come, First Serve)
- Reduce el mínimo la sobrecarga administrativa (overhead) ya que es muy simple de implementar.

- El rendimiento percibido por los últimos procesos en llegar resulta muchas veces inaceptable.
- Este algoritmo da salida a todos los procesos siempre que $p \leq 1$. Si $p > 1$ la demora en iniciar a los nuevos procesos aumentará cada vez produciéndose inanición.

Primero en llegar, primero servido (FCFS) (cont.)

Ejemplo:

Proceso	Tiempo de llegada	t	Inicio	Fin	T	E	P
A	0	3	0	3	3	0	1
B	1	5	3	8	7	2	1.4
C	3	2	8	10	7	5	3.5
D	9	5	10	15	6	1	1.2
E	12	5	15	20	8	3	1.6
Promedio		4			6.2	2.2	1.74



- *Notar qué malo es para los procesos interactivos (proceso C), donde la proporción de respuesta R es muy mala*

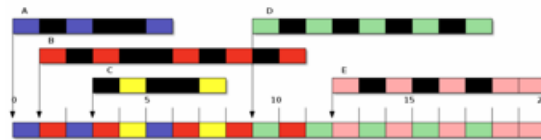
Ronda (Round Robin)

- Emplea multitarea apropiativa o Preemptive
- Busca dar una relación de respuesta buena para los procesos largos y cortos.
- Cada proceso listo para ejecutarse puede hacerlo por un solo quantum.
- Si un proceso no ha concluido dentro de su quantum se lo expulsará y será puesto al final en la cola de listos donde deberá esperar su turno nuevamente.
- Los procesos que son despertados de estado de suspensión son también puestos al final de la cola de listos.
- La ronda puede ejecutarse modificando la duración del quantum
- Cuando más grande es el quantum, más se parece a FCFS
- Se puede observar que aumentar el quantum mejora los tiempos promedios de respuesta, pero penaliza a los procesos cortos pudiendo llevar a la inanición (si $p > 1$)
- En la definición del quantum radicaré la eficiencia del algoritmo de round robin.
- Mediciones estadísticas indican que el quantum debería mantenerse por debajo al 80% a la duración promedio de los procesos.

Ronda (*Round Robin*) (cont.)

Ejemplo:

Proceso	Tiempo de llegada	t	Inicio	Fin	T	E	P
A	0	3	0	6	6	3	2.0
B	1	5	1	11	10	5	2.0
C	3	2	4	8	5	3	2.5
D	9	5	9	18	9	4	1.8
E	12	5	12	20	8	3	1.6
Promedio		4			7.6	3.6	1.98

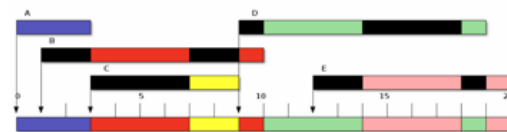


Ejemplo de Round Robin con quantum de 1 tick

Ronda (*Round Robin*) (cont.)

Ejemplo:

Proceso	Tiempo de llegada	t	Inicio	Fin	T	E	P
A	0	3	0	3	3	0	1.0
B	1	5	3	10	9	4	1.8
C	3	2	7	9	6	4	3.0
D	9	5	10	19	10	5	2.0
E	12	5	14	20	8	3	1.6
Promedio		4			7.2	3.2	1.88



Ejemplo de Round Robin con quantum de 4 ticks

El proceso más corto a continuación (SPN, Shortest Process Next)

- Utiliza multitarea no apropiativa o non-preemptive
- Se necesita información por adelantado acerca del tiempo requerido por cada proceso
- Es más justo que FCFS
- Como es difícil contar con la duración del proceso, se atiende a caracterizar sus necesidades. Para eso se nutre de la información de contabilidad del propio proceso (examina las ráfagas pasadas y concluye si es un proceso tendiendo a I/O Bound o CPU Bound)
- Para la estimación de la próxima ejecución se utiliza el promedio exponencial e_p . Además, se define un factor atenuante $0 \leq f \leq 1$ que indica cuán reactivo será dicho promedio a la última ráfaga de ejecución. Si por ejemplo el proceso p empleó q quantums en su última ráfaga de ejecución: $e'_p = fe_p + (1 - f)q$

Para el primer e_p (semilla) se puede tomar el e_p de los procesos actualmente en ejecución.

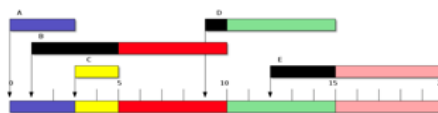
Conclusiones:

- SPN favorece a los procesos cortos
- Un proceso largo puede esperar mucho para su ejecución especialmente con un $p = 1$ o superior.
- Un proceso más largo que el promedio está predispuesto a sufrir inanición.
- En un sistema poco ocupado, con una cola de listos corta, SPN tiende a arrojar resultados muy similares a FCFS.

El proceso más corto a continuación (*SPN, Shortest Process Next*) (cont.)

Ejemplo:

Proceso	Tiempo de llegada	t	Inicio	Fin	T	E	P
A	0	3	0	3	3	0	1.0
B	1	5	5	10	9	4	1.8
C	3	2	3	5	2	0	1.0
D	9	5	10	15	6	1	1.2
E	12	5	15	20	8	3	1.6
Promedio		4			5.6	1.6	1.32



SPN apropiativo (PSPN, Preemptive Shortest Process Next)

- Combina las estrategias de SPN con un esquema apropiativo
- El comportamiento obtenido es similar para la mayoría de los procesos
- A los procesos muy largos no los penaliza mucho más que el Round Robin
- Obtiene mejores promedios en forma consistente debido a que mantiene la cola más corta por despachar a los procesos más cortos primero.
- Cuando me llega un proceso nuevo al sistema ejecuto el algoritmo, y si es más corto lo cambio.

Tipos de procesos

CPU bound u Orientados al uso de la CPU

Los que típicamente realizan mucho computo interno y su ejecución esta típicamente alternada por ráfagas u bursts. Mucho tiempo de ejecución. Uso intensivo de la CPU.

I/O bound u Orientados a E/S

Los que se centran su atención en transmitir datos desde o hacia los dispositivos externos. Son procesos que están en muy poco tiempo en ejecución. Están mucho tiempo bloqueados. Uso intensivo de E/S. Mucho tiempo bloqueado, muy poco tiempo en ejecución.

Procesos largos

Aquellos que por mucho tiempo han estado listos o en ejecución. Los que han estado en una larga ráfaga de CPU.

Procesos cortos

Aquellos que son de tipo I/O Bound pero ocasionalmente se encuentran en ejecución o tienden a estar bloqueados a la espera de eventos, típico de los procesos interactivos.

Generalmente lo que se busca en un planificador a corto plazo es dar un tratamiento preferente a los procesos cortos, en particular a los interactivos.

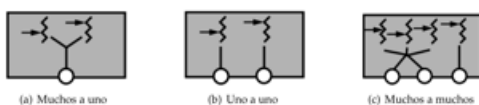
Planificación de Hilos

Para la planificación de hilos depende de cómo estos son mapeados a proceso desde el punto de vista del planificador.

Los hilos pueden ser de dos tipos:

- Hilos de usuario o hilos verdes (ULT): Solo son conocidos por el proceso que los crea y gestionados por él.
- Hilos de núcleo o kernel (KLT): Son conocidos por el SO y gestionados por él. El SO debe tener soporte para la creación y gestión de este tipo de hilos.

En base a este tipo de hilos existen los siguientes tipos de mapeos:



- Muchos a uno: Muchos hilos son agrupados dentro de un mismo proceso. Los ULT entran dentro de esta categoría. En este tipo de hilos no se aprovecha realmente el paralelismo. Si un hilo se bloquea hace que todos los demás también se tengan que bloquear.
- Uno a uno: Cada hilo es ejecutado como un proceso ligero (light weight Process LWP). Son más rápidos de crear y gestionar que un proceso pesado ya que la información de estado que se necesita para esto es mucho menor. Comparten el espacio de memoria, descriptores de archivos y demás estructuras. Aprovechan el paralelismo ya que cada hilo puede ejecutar en un procesador diferente.

Planificación de multiprocesadores

Básicamente para la planificación de múltiples procesadores se manejan dos enfoques:

- Mantener una sola lista de procesos y despacharlos a cada procesador como si estos fueran unidades de ejecución equivalentes e idénticas.
- Mantener las listas de procesos separadas para cada procesador.

Afinidad a procesador

Los procesadores actuales contienen varios niveles de cache (L1, L2, L3). Resulta obvio tratar de mantener a un proceso ejecutando en el mismo procesador para no tener que invalidar sus entradas en el cache cuando el mismo es migrado a otro procesador.

La afinidad a un procesador indica la preferencia de un proceso a ejecutarse en un determinado procesador. Puede ser:

- Afinidad suave: El planificador setea una preferencia a que determinado proceso se ejecute en un determinado procesador. Sin embargo, diferentes patrones de carga en el sistema pueden hacer que proceso sea migrado a otro procesador.
- Afinidad dura: Se da cuando en determinados SO se le permite al usuario declarar una afinidad en la cual se restringe el/los procesadores/es a ser utilizado/s para un determinado proceso.

En un entorno NUMA (Not Uniform Memory Access) funcionará mejor si el sistema maneja un esquema de afinidad dura que le permita al proceso ejecutar donde sus datos estén más cerca.

Balanceo de carga

Actúa cuando la divergencia en la carga de cada uno de los procesadores se vuelve grande, migrando procesos entre las colas de listos de un procesador a otro para homogeneizarlas. Esta técnica puede ir en sentido contrario a la afinidad que vimos anteriormente.

Existen dos técnicas de balanceo de carga:

- Migración activa o por empuje (push migration): Periódicamente se ejecuta una tarea que analiza la ocupación de los procesadores y si la misma pasa de determinado umbral, migra el proceso de la cola de dicho procesador a la cola del procesador más desocupado. Linux ejecuta esta tarea cada 200 ms.
- Migración pasiva o por jalón (pull migration): Cuando un procesador queda ocioso, ejecuta la tarea idle, que entre otras cosas en lugar de parar al

procesador analiza la ocupación de los procesadores activos. Si hay procesadores muy ocupados, jala algún proceso para migrarlo a su propia cola.

Ambas tareas son usualmente utilizadas en SO modernos. Cualquier tipo de migración conllevará una penalización en términos de afinidad de CPU.

Unidad 4 - Sincronización

En los SOs , en general, los procesos que trabajan juntos comparten con frecuencia un espacio común para almacenamiento en el que cada uno puede leer o escribir, o también comparten un recurso.

El acceso a estos recursos compartidos genera problemas de uso y comunicación entre los procesos.

Para resolver estos problemas de competencia entre procesos, se utilizan dos mecanismos:

- Sincronización entre procesos: Ordenamiento de las operaciones en el tiempo debido a las condiciones de carrera (acceder a diversos recursos asíncronamente).
- Comunicación entre procesos: Intercambio de datos.
La comunicación permite que los procesos cooperen entre sí en la ejecución de un objetivo global, mientras que la sincronización permite que un proceso continúe su ejecución después de la ocurrencia de un determinado evento.

Problemas concurrentes

- Programa secuencial: Especifica una secuencia de instrucciones que se ejecutan sobre un procesador que definimos como proceso o tarea.
- Programa concurrente: Especifica dos o más procesos secuenciales que pueden ejecutarse concurrentemente como tareas paralelas.

Un proceso secuencial se caracteriza por no ser dependiente de la velocidad de ejecución y de producir el mismo resultado para un mismo conjunto de datos de entrada, mientras que un proceso concurrente (o lógicamente paralelo) las actividades están superpuestas en el tiempo (una operación puede ser comenzado en función de la ocurrencia de algún evento, antes de que termine la operación que se estaba ejecutando).

La programación concurrente requiere de mecanismos de sincronización y comunicación entre los procesos.

Concurrencia

No se refiere a dos eventos que ocurren a la vez sino a dos o más eventos cuyo orden es no determinista, esto es, eventos acerca de los cuales no se puede predecir el orden relativo en que ocurrirán

Si bien dos procesos (o hilos) completamente independientes entre sí ejecutándose simultáneamente son concurrentes, nos ocuparemos principalmente de procesos cuya ejecución está vinculada de alguna manera.

Operación atómica

Manipulación de datos que requiere la garantía de que se ejecutara como una sola unidad de ejecución, o fallara completamente o, sin resultados o estados parciales observables por otros procesos o el entorno. Esto no necesariamente implica que el sistema no retirara el flujo de ejecución en medio de la operación, sino que el efecto de que se le retire el flujo no llevara a un comportamiento inconsistente.

Race condition

Situación en el cual el resultado de la ejecución de 2 o más procesos interactuantes depende del orden de ejecución de estos.

Sección (o región) crítica

Es la fase o etapa en la vida de ese proceso concurrente en el cual accede a recurso crítico para modificarlo o alterarlo. El área de código que se requiere ser protegida de accesos simultáneos donde se realiza la modificación de datos compartidos. Hay que garantizar de que un proceso si está utilizando una región critica otro proceso no puede utilizar ese mismo recurso crítico.

Recurso compartido

Un recurso al que se puede tener acceso desde más de un proceso. En muchos escenarios esto es un variable en memoria (como cuenta en el jardín ornamental), pero podrían ser archivos, periféricos, etc.

Mutua exclusión

Solo un proceso a la vez puede estar ejecutando en su región critica (lo accede y lo usa).

Es función del programador asegurar la atomicidad de forma explícita, mediante la sincronización de los procesos.

El sistema no debe permitir la ejecución de parte de esa área en dos procesos de forma simultánea (solo puede haber un proceso en la sección crítica en un momento dado).

Instrucciones FORK – JOIN

fork (tenedor, horqueta, separador): indica el comienzo de la concurrencia. Es en esencia un goto que simultáneamente bifurca y continua la ejecución.

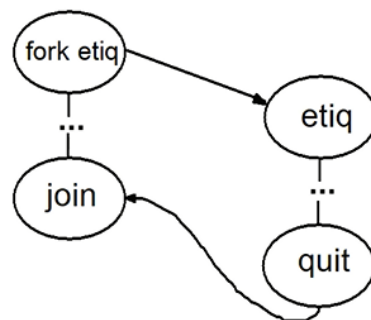
Join y quit: recombina la concurrencia en una sola instrucción indicando que ha concluido la concurrencia.

Quit: lo ejecuta el proceso hijo cuando terminó su tarea.

Join: lo ejecuta el proceso padre para esperar a que termine el hijo

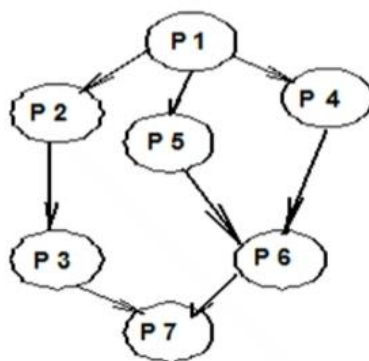
No es estructurado por su estructura de control

```
• sent1;  
• sent2;  
• fork etiq;  
• ...  
• join;  
• ...  
• end;  
  
• etiq: sent1;  
• sent2;  
• ...  
• sentn;  
• quit;
```



Grafos de precedencia

Un grafo de precedencia es un grafo sin ciclos donde cada nodo representa una única sentencia o un conjunto secuencial de instrucciones



Condiciones de concurrencia (Bernstein)

Dos sentencias cualesquiera S_i y S_j pueden ejecutarse concurrentemente produciendo el mismo resultado que si se ejecutaran secuencialmente si solo si se cumplen las siguientes condiciones:

- 1. $R(S_i) \cap W(S_j) = (\emptyset)$
- 2. $W(S_i) \cap R(S_j) = (\emptyset)$
- 3. $W(S_i) \cap W(S_j) = (\emptyset)$

Si las 3 condiciones producen conjunto vacío, podemos asegurar que no hay dependencia entre las sentencias.

Semáforos

1968 – Edsger Dijkstra

Herramienta genérica de sincronización de procesos, o sea, permite el ordenamiento de las operaciones que realizan los procesos en el tiempo.

Un semáforo es una variable de tipo entero que está protegida y la cual se puede acceder mediante la siguiente interfaz:

Inicialización: Se puede inicializar el semáforo a cualquier valor entero, pero después de esto, su valor no puede ser ya leído. Un semáforo es una estructura abstracta, y su valor es tomado como opaco (invisible) al programador.

Decrementar: Cuando un hilo decrementa el semáforo, si el valor es negativo, el hilo se bloquea y no puede continuar hasta que otro hilo incremente el semáforo. Según la implementación esta operación puede denominarse wait, down, acquire o incluso P (por ser la inicial de proberen te verlagen, intentar decrementar en holandés, el planteamiento original en el artículo de Dijkstra).

Incrementar: Cuando un hilo incrementa el semáforo, si hay hilos esperando, uno de ellos es despertado. Los nombres que recibe esta operación son signal, up, reléase, post, o V (de verhogen, incrementar).

La característica fundamental de estos operadores: Su ejecución es indivisible o sea atómica. Son atómicos porque el sistema operativo los ejecutará.

Me permite implementar varios patrones entre los cuales se mencionarán los siguientes:

Señalizar

Un hilo debe informar a otro que cierta condición está cumplida (por ej, un hilo prepara una conexión de red mientras que otro calcula lo que tiene que enviar. No se puede arriesgar a comenzar a enviar antes de que la conexión esté lista. Se inicializa con el semáforo a 0, y:

- 1 # Antes de lanzar los hilos
- 2 **from threading import Semaphore**
- 3 **senal = Semaphore(0)**
- 4
- 5 **def envia_datos():**
- 6 **calcula_datos()**
- 7 **P(senal)**
- 8 **envia_por_red()**
- 9
- 10 **def prepara_conexion():**
- 11 **crea_conexion()**
- 12 **V(senal)**

Rendezvous

Quedar en una cita. Este patrón busca que dos hilos se esperen mutuamente en cierto punto para continuar en conjunto. Los dos procesos se avisan mutuamente.

- 1 `from threading import Semaphore, Thread`
- 2 `guiListo = Semaphore(1)`
- 3 `calculoListo = Semaphore(1)`
- 4
- 5 `Thread(target=maneja_gui, args=[]).start()`
- 6 `Thread(target=maneja_calculo, args=[]).start()`
- 7
- 8 `def maneja_gui():`
- 9 `inicializa_gui()`
- 10 `V(guiListo)`
- 11 `P(calculoListo)`
- 12 `recibe_eventos()`
- 13
- 14 `def maneja_calculo():`
- 15 `inicializa_datos()`
- 16 `V(calculoListo)`
- 17 `P(guiListo)`
- 18 `procesa_calculo()`

Mutex

El uso de un semáforo inicializado a uno puede implementar fácilmente un mutex. Cada proceso que toma el semáforo también lo libera.

- 1 **`mutex = Semaphore(1)`**
- 2 # ...Inicializar estado y lanzar hilos
- 3 **`P(mutex)`**
- 4 # Aquí se está en la region de exclusión mutua
- 5 **`x = x + 1`**
- 6 **`V(mutex)`**
- 7 # Continúa la ejecución paralela. En Python:

Multiplex

Permite la entrada de no más de n procesos a la región crítica. Si se lo ve como una generalización del mutex, basta con inicializar el semáforo al número máximo de procesos deseado. Su construcción es idéntica a la del mutex, pero es inicializado al número de procesos que se quiere permitir que ejecuten de forma simultánea.

Me aseguro de que la cantidad de procesos que yo deseo estén haciendo una tarea a la vez.

Torniquete

Una construcción por sí sola no hace mucho, pero resulta útil para patrones posteriores. Esta construcción garantiza que un grupo de hilos o procesos pasa por un punto determinado de uno en uno (incluso en un ambiente multiprocesador):

- 1 **torniquete = Semaphore(0)**
- 2 # (...)
- 3 **if alguna_condicion():**
- 4 **V(torniquete)**
- 5 # (...)
- 6 **P(torniquete)**
- 7 **V(torniquete)**

En este caso, se ve primero una señalización que hace que todos los procesos esperen frente al torniquete hasta que alguno marque alguna_condición() se ha cumplido y libere el paso. Posteriormente, los procesos que esperan pasarán ordenadamente por el torniquete. El torniquete por sí solo no es tan útil, pero su función se hará clara a continuación.

Apagador

Cuando se tiene una situación de exclusión categórica (basada en categorías y no en procesos individuales – varios procesos de la misma categoría pueden entrar a la sección crítica, pero procesos de dos categorías distintas deben tenerlo prohibido), un apagador permite evitar la inanición de una de las categorías ante un flujo constante de procesos de la otra. El apagador usa, como uno de sus componentes, un torniquete. Para ver una implementación ejemplo de un apagador referirse a la solución presentada para el problema de los lectores y escritores.

Problema de los lectores y los escritores

Primera aproximación

- 1 import threading
- 2 lectores = 0
- 3 mutex = threading.Semaphore(1)
- 4 cuarto_vacio = threading.Semaphore(1)
- 5
- 6 def escritor():
- 7 P(cuarto_vacio)
- 8 escribe()
- 9 V(cuarto_vacio)
- 10
- 11 def lector():
- 12 P(mutex)
- 13 lectores = lectores + 1
- 14 if lectores == 1:
- 15 P(cuarto_vacio)
- 16 V(mutex)
- 17
- 18 lee()
- 19
- 20 P(mutex)
- 21 lectores = lectores - 1
- 22 if lectores == 0:
- 23 V(cuarto_vacio)
- 24 V(mutex)

Solución

```
• 1 import threading
• 2 lectores = 0
• 3 mutex = threading.Semaphore(1)
• 4 cuarto_vacio = threading.Semaphore(1)
• 5 torniquete = threading.Semaphore(1)
• 6
• 7 def escritor():
• 8     P(Torniquete)
• 9     P(cuarto_vacio)
• 10    escribe()
• 11    V(cuarto_vacio)
• 12    V(torniquete)
• 13
• 14 def lector():
• 15     global lectores
• 16     P(torniquete)
• 17     V(torniquete)
• 18 //Continúa
• 19     P(mutex)
• 20     lectores = lectores + 1
• 21     if lectores == 1:
• 22         P(cuarto_vacio)
• 23     V(mutex)
• 24
• 25     lee()
• 26
• 27     P(mutex)
• 28     lectores = lectores - 1
• 29     if lectores == 0:
• 30         V(cuarto_vacio)
• 31 V(mutex)
```

Barrera

```
• 1 num_hilos = 10
• 2 cuenta = 0
• 3 mutex = Semaphore(1)
• 4 barrera = Semaphore(0)
```

Una barrera es una generalización del Rendezvous que permite la sincronización entre varios hilos (no solo dos), y no requiere que el papel de cada uno de los hilos sea distinto. Esta construcción busca que ninguno de los hilos continúe ejecutando hasta que todos hayan llegado a un punto dado. Para implementar una barrera es necesario que esta guarde algo de información adicional además del semáforo, particularmente el número de hilos que se han lanzado (para esperarlos a todos). Esta será una variable compartida y por tanto requiere un mutex. La inicialización (que se ejecuta antes de iniciar los hilos) será:

Suponiendo que todos los hilos tienen que realizar por separado la inicialización de su estado, y ninguno de ellos debe comenzar el procesamiento hasta que todos hayan efectuado su inicialización:

```
• 1 inicializa_estado()
• 2
• 3 P(mutex)
• 4 cuenta = cuenta + 1
• 5 V(mutex)
• 6
• 7 if cuenta == num_hilos:
• 8     V(barrera)
• 9
• 10 P(barrera)
• 11 V(barrera)
• 12
• 13 procesamiento()
```

Monitores

Son estructuras provistas por el lenguaje o entorno de desarrollo que encapsulan tanto los datos como las funciones que los pueden manipular, impidiendo el acceso directo a las funciones potencialmente peligrosas. Son tipos de datos abstractos (ADT), clases de objetos y exponen una serie de métodos públicos, además de poseer métodos privados que emplean internamente.

Por ej. Java implementa sincronización vía monitores entre hilos como una propiedad de la declaración de método, y lo hace directamente en la JVM.

Comunicación entre procesos (IPC – Inter Process Communication)

La comunicación entre procesos puede ser realizada de dos maneras:

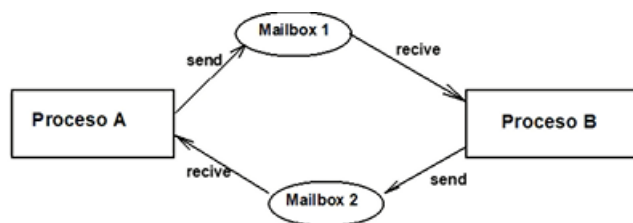
- 1) Comunicación a través de un área común de memoria (herramienta del SO).
- 2) Comunicación mediante el intercambio de mensajes (lo usamos cuando hay dos procesos en distintos SOs).

El propósito es permitir que dos procesos se sincronicen o simplemente se envíen datos mediante un mecanismo explícito.

Mensaje: porción discreta de datos (generalmente compuesto con un conjunto de bits). Tienen una cabecera (header) y un cuerpo. El header tiene el identificador del transmisor, el identificador del receptor, la longitud y el tipo.

Tipos de comunicación entre procesos

- **Comunicación directa:** Los procesos envían y reciben los mensajes entre sí. Dependen de las velocidades relativas entre sí (si son distintas requieren un buffer de mensajes para su sincronización).
- **Comunicación indirecta:** Los mensajes son enviados a buzón o mailbox y se retiran del buzón.



Mailbox: interface entre procesos y S.O. Se crea y quita fácilmente.

Procesos: A Pide un mailbox y envía MSG, B se activa y recibe MSG.

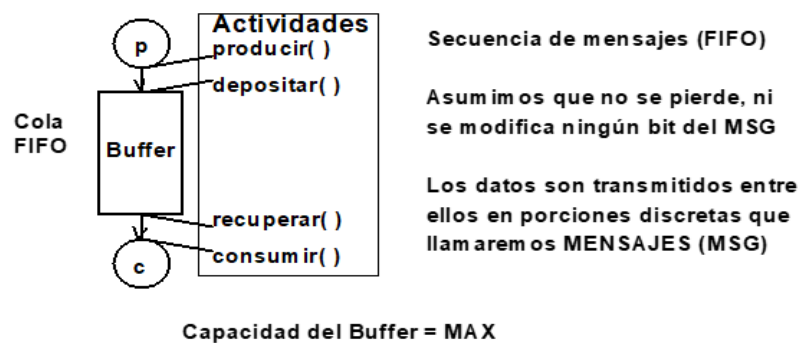
Tipos de sincronizaciones mediante mensajes

- **Comunicación Sincrónica:** El proceso emisor es bloqueado hasta que el receptor esté listo para recibir el mensaje. Cuando el proceso receptor ejecuta el receive y el mensaje no se encuentra disponible queda bloqueado hasta la llegada de este. Una vez que se ha producido el intercambio de mensajes ambos procesos continúan su ejecución concurrentemente.
- **Comunicación asincrónica:** Las primitivas de este tipo de comunicación se caracterizan por no bloquear a los procesos que las ejecutan. Así cada uno sigue su ejecución. Esto es importante en el caso del receptor ya que sigue ejecutando, aunque no le llegue ningún mensaje. Depende de la implementación si los mensajes siguientes serán atendidos o no.
- **Comunicación semi-sincronica:** Se usa un send no bloqueante y receive bloqueantes. Esto es riesgoso pues se pueden acumular una gran cantidad de mensajes en colas.

Modelo productor-consumidor

Usado para describir dos procesos ejecutando en forma concurrente:

- Productor: genera un conjunto de datos necesarios para la ejecución de otro proceso.
- Consumidor: Toma los datos generados por el productor y los utiliza para su procesamiento.



- Productor: Genera elementos mediante producir() y los ingresa en el buffer mediante depositar()
- Buffer: Zona de memoria utilizada para amortiguar las diferencias de velocidad entre dos procesos. Almacena temporalmente los elementos generados por p.
- Consumidor: Retira los elementos del buffer mediante recuperar() y los consume con consumir()

Algoritmos para el modelo productor-consumidor

A. Con semáforos

C. Con Semáforos

Usa tres semáforos:

semáforo mutex = 1, vacío = N, lleno = 0;

```

p()
{
    while (1)
    {
        x = producir();
        P(vacío);
        P(mutex);
        ingresar(x);
        V(mutex);
        V(lleno);
    }
}

c()
{
    while(1)
    {
        P(lleno);
        P(mutex);
        x = sacar();
        V(mutex);
        V(vacío);
        consumir(x);
    }
}

```

Cualesquiera de estos algoritmos propuestos sincronizan el modelo de productor-consumidor que se presentan en múltiples situaciones dentro de un computador. Por ejemplo la CPU produciendo mensajes para un módulo de E/S o viceversa.

Mecanismos de IPC

- Pipes y fifos
- Señales
- Memoria compartida
- Sockets
- RPC / RMI

Bloqueos mutuos e inanición

Deadlock: Situación que ocurre cuando dos o más procesos poseen determinados recursos y cada uno queda detenido esperando a alguno de los que tiene el otro. El sistema puede seguir operando normalmente, pero ninguno de los procesos involucrados podrá avanzar.

Puede darse por 2 motivos:

- Comunicación entre procesos: Tengo varios procesos bloqueados, uno de los mismo debe enviar un mensaje para desbloquear a los otros y el mensaje nunca sale.
- Petición de recursos

Se debe fundamentalmente por el uso de recursos. Se pueden distinguir dos categorías generales de recursos: reutilizables y consumibles.

Condiciones de Coffman (necesarias y suficientes)

Para que se produzca un estado de deadlock, se tienen que producir las 4 condiciones simultáneamente.

- 1) Mutua exclusión: Los procesos reclaman control exclusivo de los recursos que piden.
- 2) Retener y esperar: Los procesos mantienen los recursos que ya les han sido asignados mientras esperan por recursos adicionales.
- 3) No expropiación: Los recursos no pueden ser extraídos de los procesos que los tienen hasta su completa utilización.
- 4) Espera circular: Hay una cadena circular de procesos en la que cada uno mantiene a uno o más recursos que son requeridos por el siguiente proceso de la cadena.

Recursos reutilizables: Puede ser usado por un proceso y no se agota con el uso.

Los procesos obtienen unidades de recursos que liberan posteriormente para que otros procesos las reutilicen. Como ejemplos, se tienen los procesadores, canales E/S, memoria principal y secundaria, dispositivos y estructuras de datos tales como archivos, bases de datos y semáforos.

Recursos consumibles: Puede ser creado (producido) y destruido (consumido).

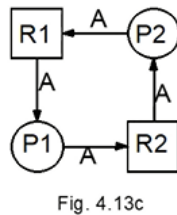
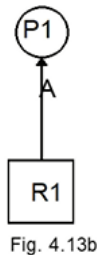
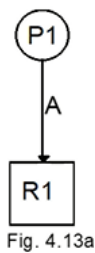
Normalmente, no hay límite en el número de recursos consumibles de un tipo en particular. Un proceso productor que no está bloqueado puede liberar a cualquier número de recursos consumibles. Cuando un proceso adquiere un recurso este deja de existir. Como ejemplos están las interrupciones, señales, mensajes, e información en buffer de E/S.

Inanición (resource starvation): Situación en que un proceso no puede avanzar en su ejecución dado que necesita recursos que están (alternativamente) asignados a otros procesos.

Grafos de asignación de recursos

Manera de definir los deadlocks a través de grafos. Estos grafos están formados básicamente por dos elementos:

- Un conjunto de vértices formado por los procesos y los recursos del sistema.
- Un conjunto de arcos que representan la asignación o solicitud de recursos.



Los recursos fueron representados con cuadrados y los procesos con círculos.

Estrategias para tratar deadlocks:

- Ignorarlos: La mayoría de los sistemas operativos utilizan esta técnica, incluyendo UNIX. Es una de las formas más simple de tratarlo. Algunas veces el costo de ignorarlo es mucho menor al costo que implicaría prevenirlo o detectarlo y recuperarlo. Tanenbaum lo llama el ‘algoritmo del avestruz’.
 - Prevención: Siendo las 4 condiciones necesarias para que ocurra un deadlock, basta con asegurarnos que una de ellas no ocurrirá.
 - o Mutua exclusión: las soluciones para aquellos recursos que no pueden ser compartidos son diversas, pero todas se basan en que un proceso no quede esperando en caso de la falta de disponibilidad de dicho recurso.
 - o Toma y espera (Hold & Wait): Para solucionar este problema, se trata de garantizar que cuando un proceso tenga un recurso asignado no pueda solicitar otro. Hay dos caminos para lograrlo: 1) Los procesos solicitan todos los recursos en el momento previo a comenzar la ejecución, de no poder ser entregados el proceso queda bloqueado. Asignación estática completa de todos los recursos que necesita para su ejecución (caso COBOL). 2) Un proceso primero debe liberar aquellos recursos que posee y luego recién podrá solicitar otros, es decir solo está en condiciones de solicitar un recurso cuando no tiene ninguno asignado.
 - o No expropiación (No Preemption): 1) Si un proceso solicita un recurso que no está disponible, este debe devolver todos aquellos recursos que tenía previamente asignados. 2) Si un proceso pide un recurso que tiene otro proceso, el SO puede obligar a liberar los recursos al otro proceso.
- El primer método es viable solo en aquellos procesos cuyos estados pueden ser fácilmente grabados y restaurados. El segundo método presenta el inconveniente del estado de inanición en caso de que a un proceso siempre le quiten los recursos y nunca pueda finalizar su tarea.
- o Espera circular: Consiste en imponer un orden lineal de ejecución que evite las esperas circulares: 1) Estableciendo un orden lineal a los recursos: Si tenemos una lista de recursos R1, R2, ..., Rn, un proceso que solicite Rh solo puede pedir aquellos recursos Rk con $k > h$. Esto evita que se forme un círculo. 2) Hold & Wait: un proceso solo está en

condiciones de solicitar un recurso cuando no tiene ninguno asignado. El problema del orden lineal es que también tiene cierto grado ineficiente ya que estaría negando recursos a procesos, innecesariamente.

- Detectar y recuperar: Consiste en abortar un proceso cuando se detecta o se presupone que puede ocurrir el deadlock. La ventaja de esta táctica es que no limita el acceso a los recursos. Presenta ciertos inconvenientes como el de decidir la frecuencia con los que se llevará a cabo el algoritmo de detección. El algoritmo podría ser ejecutado cada vez que solicita un recurso, a cada hora, etc.

Métodos para recuperar a los procesos y a los recursos una vez detectada la situación:

- o Abortar todos los procesos involucrados.
- o Abortar los procesos uno a uno, hasta que el deadlock desaparezca. Mejor que el anterior, menos costoso.
- o Hacer un backup de cada proceso en un punto anterior: Checkpoint. A este proceso de reinicio se lo llama Rollback. Consiste en llevar el proceso a un punto anterior al de haberle sido asignado el recurso causante del Deadlock. No es sencillo hacer un Rollback en un SO y además si no cambio nada el deadlock puede volver a generarse tranquilamente.
- o Quitar el recurso a un proceso y entregárselo a otro que lo haya solicitado. También hay que ejecutar el algoritmo de detección luego de que se quitó el recurso.

Unidad 5 - Administración de Memoria Central

Funciones y Operaciones

- Por cuestiones de diseño, el único espacio de memoria que el procesador puede utilizar es la Memoria Central (MC)
- Los caches son para mejorar la performance y en general replican el contenido de la MC. No se usan para guardar datos (solo la MC)
- Los registros del procesador son muy pequeños y solo los utiliza para realizar sus operaciones (podríamos guardar los datos, pero no lo hacemos por su tamaño tan limitado)
- Los discos son un almacenamiento secundario. No es accesible directamente por el procesador. Están conectados al sistema mediante el módulo I/O
- Los programas deben cargarse en la MC antes de ser ejecutados. Depende de cómo se gestione la memoria se cargará o no todo completo.

- También está relacionado con el HW que disponga el equipo, en especial el MMU

Espacio de direccionamiento

Está estructurada como un arreglo unidireccional de bytes. Cada operación de lectura / escritura se hará de a 8 bits (no menos).

Un procesador que soporta un espacio de direccionamiento de 16 bits puede referirse directamente a hasta 2^{16} bytes, esto es 64 KB.

Un procesador de 32 bits, sus registros pueden referenciar hasta 4294967296 bytes (4GB) de RAM.

A través de un mecanismo llamado PAE (extensión de direcciones físicas, Physical Address Extension) permite extender eso a rangos de hasta 2^{52} bytes a cambio de un nivel más de indirección. Este es un mecanismo propio del procesador y además la Mother debe soportarlo.

Un procesador de 64 bits podría direccionar hasta 18 446 744 073 709 551 616 bytes (16 Hexabytes)

En la práctica por cuestiones económicas, es de 2^{40} a 2^{48} .

La capacidad de direccionamiento depende del procesador y del SO, hay sistemas operativos de 32bits y de 64bits, si alguno de los dos es de 32 aunque el otro sea de 64 solo podré utilizar 32. Los programas, aunque sean de 32 bits funcionan igual con SO de 64 bits y procesadores de 64 bits solo que funcionan más lento que uno de 64.

Hardware

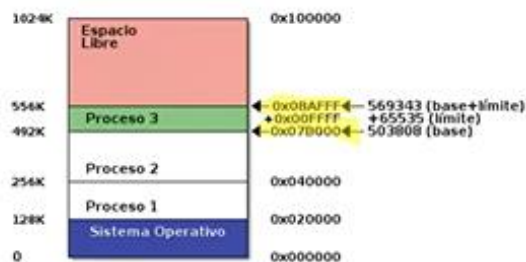
MMU (Memory Manager Unit)

Nos va a permitir varias funciones de administración de memoria. Resuelve tareas para mejorar la performance. Controla que no hay un segmentation fault. Es parte de los circuitos del procesador. Está ligado a los buses porque los buses transportan los datos, pero no tiene relación directa. El MMU hace determinadas operaciones para saber las posiciones de memoria. Me ayuda en el proceso de cálculo.

Hace falta cuando:

- Hay multitarea (multiprogramación). Hay múltiples tareas en MC. El SO debe resolver como ubicar los programas en la memoria física disponible.
- Ir más allá de la memoria física disponible (Memoria Virtual)
- Verificar los límites entre los que un proceso puede acceder al espacio asignado (ejemplo: Registro base / limite)

Pasa la información al MMU y el MMU realiza la función por una cuestión de performance. Registra el registro base y el registro límite, y si quiero ir a un registro fuera de este rango se lanzará un trap (fuera del límite de memoria). Cuando se carga un proceso, se llena el MMU para evitar que ese proceso tenga información de cuáles son los límites, y de esa manera, controlarlo. Para el sistema operativo sería muy costoso ir a memoria central (por ejemplo, a través de syscalls) para verificar los límites en cada operación, por eso se usa el MMU.



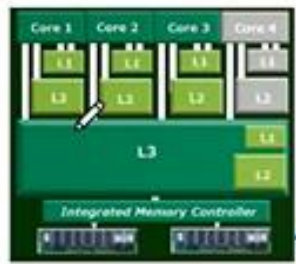
Memoria Caché

La velocidad del procesador es mayor que la de la MC. Influye directamente en el rendimiento ya que, si no está el dato, el sistema detiene su ejecución (stall). Agregar demasiada memoria caché puede terminar siendo contraproducente para la performance. La curva de rendimiento de la memoria cache no es siempre en ascenso (si es demasiado grande es por el costo de cargarla y mantenerla), cuando quiero hacer una modificación si el bloque de memoria cache es muy grande, el proceso de llevarlo a disco y modificarlo puede ser lento.

Es una memoria de alta velocidad que está entre el Procesador y la MC que guarda copias de las páginas accedidas partiendo del principio de localidad de referencia:

- Localidad Temporal: probabilidad de reutilizar un recurso recientemente utilizado.
- Localidad espacial: La probabilidad de que un recurso aún no requerido sea accedido es mucho mayor si fue requerido algún recurso cercano.
- Localidad secuencial: Un recurso, y muy particularmente la memoria tiende a ser requerido de forma secuencial.

El precio de un sistema varía significativamente dependiendo de los tamaños de caches de Nivel 2 y Nivel 1 con el que cuenta.



Espacio de memoria de un proceso

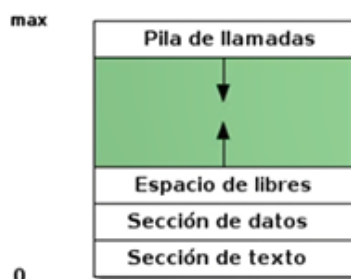
Cuando se crea un proceso, la carga se realiza con la siguiente estructura (4 áreas):

- Sección (o segmento) de texto o código: Es la imagen en memoria de las instrucciones a ser ejecutadas. Usualmente, ocupa las direcciones más bajas del espacio de memoria. Ahí va el código. Es un área fija porque el código no cambiará en la ejecución. (el Assembler del programa está ahí)
- Sección de datos: Espacio fijo preasignado para las variables globales y datos inicializados (como las cadenas de caracteres, por ejemplo). Se fija en tiempo de compilación, y no puede cambiar (aunque los datos cargados allí si cambian en el tiempo de vida del proceso).
- Espacio de libres (HEAP): Espacio para la asignación dinámica de memoria durante la ejecución del proceso. Este espacio se ubica por encima de la sección de datos, y crece hacia arriba. Esta área si puede cambiar.

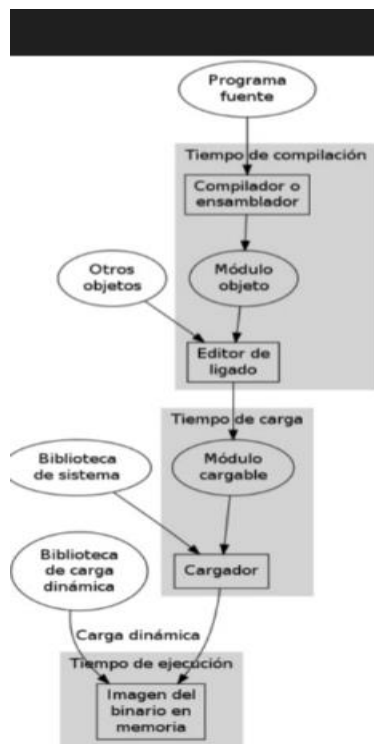
Lenguajes dinámicos (C) -> Free / Malloc

Lenguajes gestión automática (java) -> garbage collector.

- Pila de llamadas (Stack): Espacio de memoria que se usa para almacenar la secuencia de funciones que han sido llamadas dentro del proceso, con sus parámetros, direcciones de retorno, variables locales, etc. La pila ocupa la parte más alta del espacio en memoria y crece hacia abajo. También puede cambiar. El PCB va al stack.



Resolución de direcciones



El compilador reemplaza las variables / funciones simbólicas por las direcciones de memoria a donde se ubican. (ej. posición de inicio + 80 a posición AC897AB). Para poder coexistir con otros procesos, esas direcciones deben ser traducidas a la posición relativa, con alguna de las siguientes estrategias:

- En tiempo de compilación: Las direcciones son absolutas / fijas. Ej command.com. Esto no se utiliza porque si se compilan por ej, en dos direcciones iguales dos programas distintos esto provocará una falla. No se usa.

- En tiempo de carga: El loader calcula las ubicaciones al momento del inicio del programa (ej: Reg Base + offset). Para esa ejecución del programa, la ubicación será definida en el tiempo de carga y no cambiará hasta que

termine la ejecución.

- En tiempo de ejecución: El programa NUNCA hace referencia a una ubicación fija (ej: base_frame + offset). Esta traducción la hace el MMU. Cuando el programa hace referencia a la ubicación lógica, en ese momento se traduce a la dirección física.

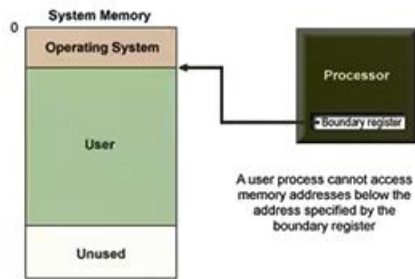
Asignación de memoria contigua

En todas estas, en cada partición hay un registro base y un registro límite. Estos registros se guardan en el PCB. En este tipo de asignación no tengo el MMU. En este tipo de asignación no podría usar multiprogramación porque puedo cargar un solo proceso y el resto de la memoria se desperdicia.

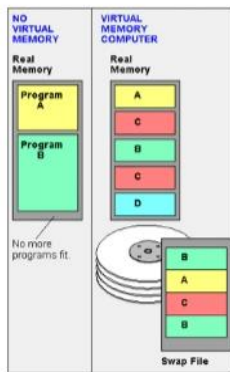
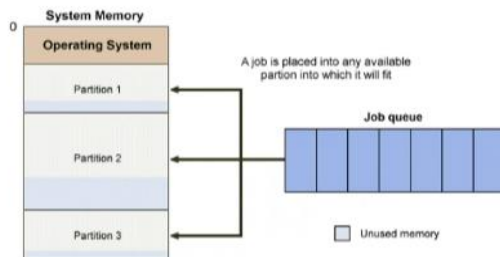
Simple

Esto funcionaba con sistemas Monoprogramados / por lotes / Monousuario -> el SO no necesitaba gestionar la memoria. (ej. DOS)

Lo único que hace el MMU es validar que cualquier referencia que haga el proceso usuario este por encima del espacio boundary (si el proceso hace referencia a una dirección por debajo se gatilla un trap). Toda la memoria es para el proceso de usuario. Si el proceso mide menos que la memoria, el espacio que queda se desperdicia. Si es más grande, el programa no puede ejecutarse.



Particiones fijas



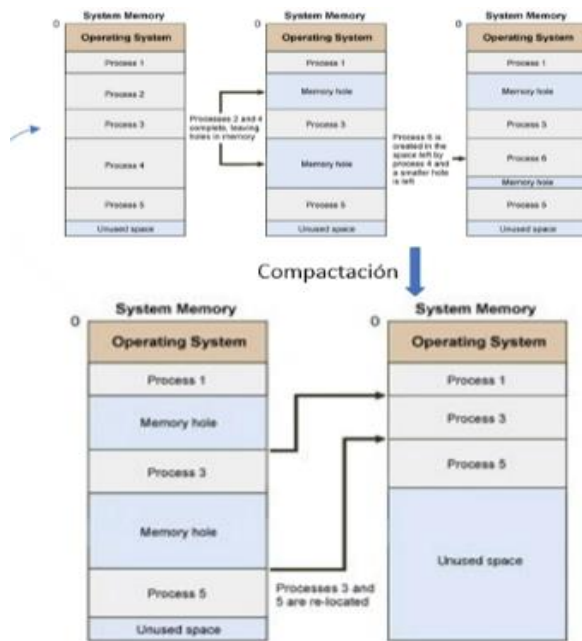
Se parte de la memoria en espacios fijos predeterminados (no tiene por qué ser igual para todas las particiones). Sistemas multiprogramados.

La cantidad de particiones que yo tenga determinara la cantidad de procesos que yo pueda tener activos simultáneamente (nivel de multiprogramación). Es importante definir bien el tamaño de las particiones y la cantidad de las particiones (aunque es mejor tener más particiones por la multiprogramación, a más particiones, más chicas). El proceso más grande que se va a ejecutar debe ser como máximo del tamaño de la partición más

grande. Fragmentación interna (es cuando hay parte de la memoria de la partición que no se está utilizando).

Prácticamente ya no se usa. Al llegar un proceso nuevo, me fijo que el proceso quepa en alguna de las particiones libres y después ver en que partición me conviene alojarlo. (uso un algoritmo).

Particiones variables



Compactación: Para minimizar la fragmentación, el sistema puede realizar una "compactación", que implica reubicar la memoria de los procesos juntando el espacio libre

Asignación dinámica de acuerdo con el tamaño a ubicar. Se van definiendo a medida que las voy necesitando.

- First fit: hace hincapié en la velocidad.

Lo mete en el primero lugar que cabe.

- Best fit: hago hincapié en el espacio. Lo mete en el espacio más chico donde quepa.

- Worst fit: Lo ubica siempre en el hueco más grande que encuentra.

Los que mejor rendimiento tienen son los dos primeros. El tercero casi no se usa.

Fragmentación externa (huecos libres que no corresponden a ninguna partición). Para solucionarla, aplico desfragmentación (tiene un costo). Al principio cada proceso tiene asignada

una partición con tamaño acorde, pero a medida que se van terminando los procesos, empiezan a quedar huecos de distinto tamaño sin utilizar. Cuando llega un nuevo proceso, ahí utilizo los algoritmos antes mencionados de asignación de espacio libre.

Segmentación

- Divide la memoria en segmentos de tamaños variables
- Los segmentos se corresponden a con las distintas partes del programa (código, tabla de símbolos, Stack, heap)
- Los segmentos pueden tener distintos permisos de acceso (read only, write, execution). Esto me permite compartir segmentos.
- Se pueden compartir segmentos entre distintos procesos.
- Las bibliotecas ligadas dinámicamente (link edited) se presentan en segmentos independientes.
- Se puede utilizar memoria virtual
- El espacio libre se gestiona en forma similar a particiones variables.

Se genera un segmento por cada área de memoria del programa. Cuando creo un programa le voy a asignar un segmento para código, un segmento para heap, un segmento para Stack, un segmento para datos.

Traducción en tiempo de ejecución.

Ventajas: puedo compartir segmentos (no con particiones variables). Por ejemplo, con el WinWord, mismo segmento de código (winword.exe), distinto segmento para los datos. Esta funcionalidad, posible para diferentes procesos, funciona de la misma manera para diferentes threads de mismos procesos (comparten el área de código). Como reparto el proceso en distintos segmentos no necesito que los espacios de memoria sean contiguos.

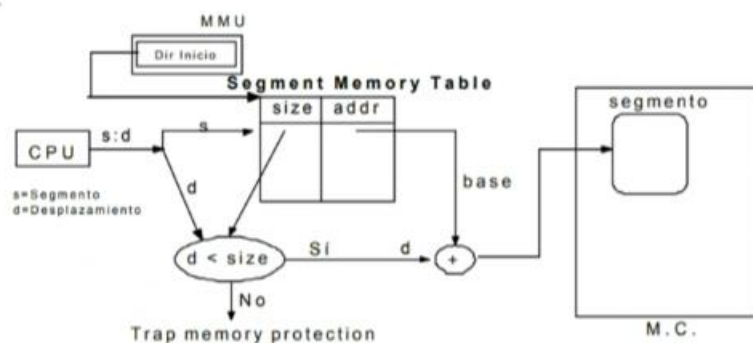
Código entrante o puro: segmentos de código que no modifican ningún estado y son completamente independientes del entorno en el que se ejecutan.

Las direcciones están compuestas por Nro. de segmento + desplazamiento (s:d)

El SO mantiene una SMT (Segment Memory Table) por proceso (guardada en el PCB cuando no se está ejecutando y en el MMU cuando se está ejecutando) que tiene el Nro. de segmento, el tamaño y la dirección de inicio. Además, debe existir una sola tabla con todos los espacios libres de la memoria. El espacio libre se gestiona con el criterio de particiones variables (soluciono compactando o desfragmentando).

Cuando se requiere una dirección, se recupera el inicio del segmento en la SMT y le suma el desplazamiento (Offset) y se obtiene la dirección en la MC (memoria central).

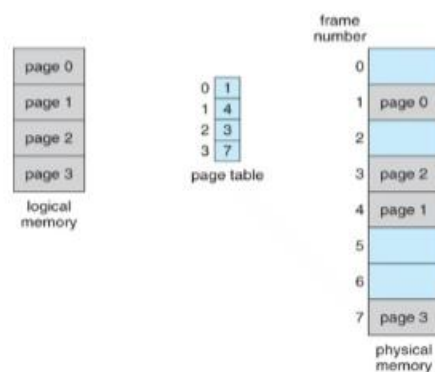
Puede haber fragmentación externa y se resuelve desfragmentando. El soporte de HW es la MMU, la cual controla que un segmento no tome dirección de otro segmento.



Paginación

El principal problema de las técnicas de administración de memoria hasta ahora era que la memoria para un proceso tenía que ser contigua. Para solucionar este tema surge la paginación. En esta técnica de memoria, la misma se divide en frames. Los frames tienen un tamaño variable (siempre en potencias de 2).

Pura o simple

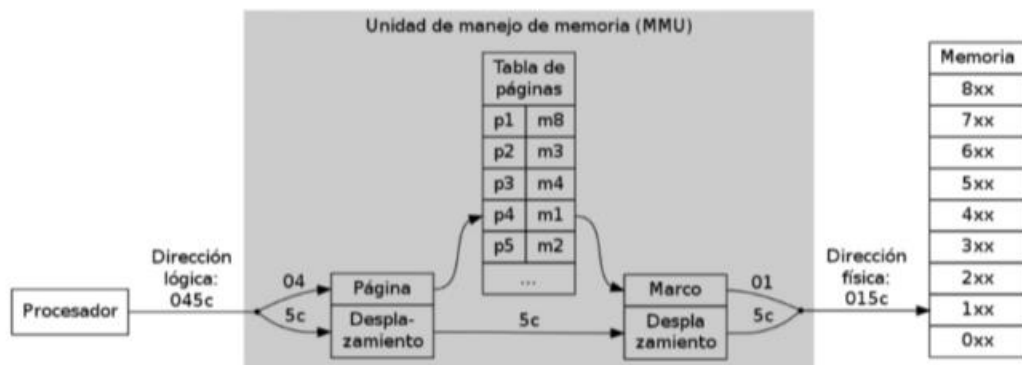


El proceso se divide en bloques de tamaño fijo llamados páginas (pages).

La memoria física se divide en una serie de marcos (frames) del mismo tamaño que las páginas. Gracias a esto, en mi memoria tendré n frames, y podre cargar en ellos páginas de cualquier proceso (todas iguales). Mi proceso podrá cargar sus páginas en cualquier frame (no hay más necesidad de que la memoria sea

contigua). Por lo general los frames/pages se dividen en porciones de entre 512B (2^9) y 16MB (2^{24}). Siempre son potencias de 2. Para el direccionamiento se divide el bus de direcciones considerando N bits para la página y M bits para el desplazamiento (offset).

Los frames son todos del mismo tamaño y las páginas son todas del mismo tamaño. Normalmente el tamaño de los frames se setea al inicio (depende de cómo se organice el SO). Desaparece la fragmentación externa, vuelve a aparecer algo (no mucho) de fragmentación interna en la última página de cada proceso.



Las direcciones están compuestas por Nro. de Página + desplazamiento (p:d)

El SO mantiene una MPT (Memory Page Table) **por proceso** que tiene el Nro. de página, el tamaño y la dirección de inicio. Es espacio ocupado / libre se administra con una tabla de bits por frame MFT (Memory Frame Table) (**una sola para todo el sistema**). En tiempo de ejecución el MMU pasa de dirección lógica a física utilizando la MPT.

Cuando se requiere una dirección, se recupera el inicio del frame en la MPT y se le suma el desplazamiento (Offset) y se obtiene la dirección en la MC. Teniendo en cuenta que el tamaño de bits que yo tengo en la dirección es límite de direccionamiento que puedo referenciar, la cantidad de bits necesarios para las páginas tiene que estar adecuada también a la cantidad de bits que necesito para el offset dentro de las páginas (los bits a la derecha).

Necesito un MMU como soporte de HW para manejar todas las páginas. Carga el control de la tabla de páginas en donde está registrado que página fue a cuál marco y verifica que se estén cumpliendo los límites y calcula las direcciones de número de página + offset. Además, se pueden implementar TLBs y caches.

La fragmentación se reduce mucho respecto a la asignación contigua, sólo se fragmentan porciones de páginas. Hay fragmentación interna. No tiene el problema de la segmentación. Cuando un frame se libera, se puede volver a reasignar.

El tamaño de página óptimo deberá estar dentro de determinados límites. Si las páginas son muy grandes, hay mucha fragmentación interna; si hay páginas demasiado chicas, la tabla de páginas es muy grande y el PCB también lo es, esto lleva a transferencias de MC y disco más costosas.

- Ejemplos: Procesador 16 bits y páginas de 13bits (8KB) => $2^{13} = 8$ entradas. ¡Insignificante!
Procesador 32 bits y páginas de 12bits(4KB) => $2^{20} = 1.048.576$ entradas . x (20b frame + 20b pág) ¡Unos SMB!
¿¿¿Procesador de 64bits ??? ¡Inmanejable!

Se puede compartir memoria como en segmentación. Cuando se crea un proceso hijo se copia el espacio de memoria del padre, apuntando a las páginas del padre, cuando se modifican se reescriben en un frame nuevo, de esta forma, las páginas de solo lectura y/o ejecución son compartidas por ambos procesos.

Direccionamiento con TLB

Translation Lookaside Buffer (TLB) / Buffer de traducción anticipada: El TLB es una tabla asociativa (un hash) en memoria de alta velocidad, una suerte de registros residentes dentro de la MMU, donde las llaves son las páginas y los valores son los marcos correspondientes. De este modo, las búsquedas se efectúan en tiempo constante.

A los efectos funcionales son una especie de memoria cache, la diferencia es que son registros que están dentro de la MMU y se usan para almacenar parte de mis tablas de páginas de memoria. Funciona como una memoria cache asociativa (hash). Tengo como entrada las páginas y como referencia la ubicación. Como son de acceso asociativo, con un solo acceso leo todas las entradas que tenga el TLB.

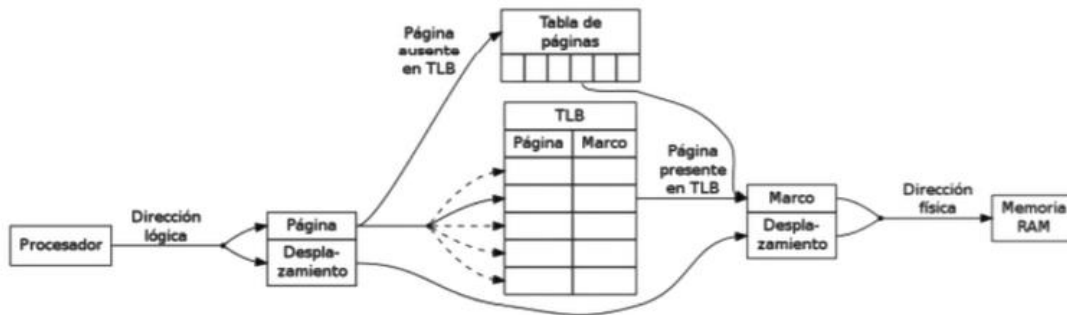


Figura 5.10: Esquema de paginación empleando un *buffer de traducción adelantada* (TLB).

Direccionamiento con TLB y Cache

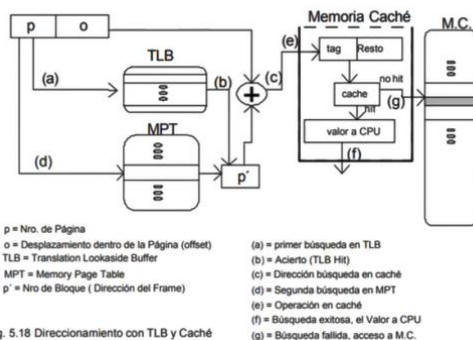


Fig. 5.18 Direccionamiento con TLB y Caché

Direccionamiento con TLB y Cache

Se usa TLB + memoria cache.

Paginación Multinivel

Se definen tablas de directorios que a su vez apuntan a tablas de páginas. Consiste en dividir mis páginas en dos niveles. Ej. sistemas de 32 bits, con 10 para la tabla externa y 12 para el desplazamiento. Al tener una tabla que usare como índices de las páginas solo tendré dos accesos en vez de ir a buscar página por página lo que necesito. Ej en vez de tener un registro de más de un millón de elementos tendré dos páginas de 1024. Cuando yo tengo tamaño de página muy grande en vez de tener particionado mi bus en dos, los particionaré en 3 o 4 y de esa manera lograré tamaño de páginas más chicas.

En vez de tener tablas de páginas muy grandes, resuelvo mi problema con particiones más chicas.

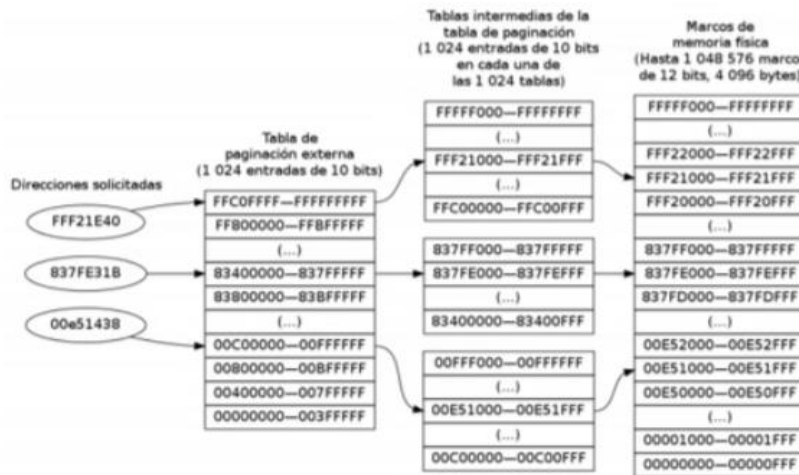
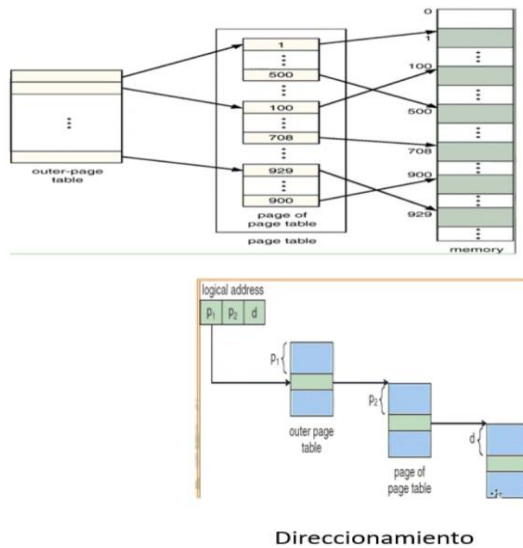


Figura 5.11: Paginación en dos niveles: una tabla externa de 10 bits, tablas intermedias de 10 bits, y marcos de 12 bits (esquema común para procesadores de 32 bits).



Memoria Compartida

Se pueden compartir solo las páginas que son de solo lectura. Sigue el mismo criterio de segmentos compartidos.

Copiar al escribir (copy on write, CoW)



(a) Inmediatamente después de la creación del proceso hijo por `fork()`



(b) Cuando el proceso hijo modifica información en la primera página de su memoria, se crea como una página nueva.

Memoria Virtual

Paginación sobre demanda

En general, memoria virtual es la utilización de una memoria secundaria (por ejemplo, parte de un disco). De esta forma los procesos trabajan con una idealización de la memoria en el cual pueden ocupar hasta el 100% de la capacidad de direccionamiento, independiente de la memoria física con la cuenta el sistema. La memoria virtual es gestionada de forma automática y transparente por el sistema operativo. No se considera memoria virtual, por ejemplo, si un proceso pide explícitamente intercambiar determinadas páginas.

El estado de bloqueado suspendido y listo suspendido, son para hacer swapping porque es la parte más costosa (overhead y tiempo). Entonces en el mientras tanto suspendo el proceso.

Swapping: Es el intercambio de información entre distintos niveles de memoria.

Swap-in: Es cuando el intercambio se hace desde un dispositivo de menor jerarquía a uno de mayor jerarquía (ej. disco duro a MC)

Swap-out: Es cuando el intercambio se hace desde un dispositivo de mayor jerarquía a uno de menor jerarquía. (ej. de cache L1 a MC)

Solo se pueden pasar a suspendidos los procesos que no tengan pendientes operaciones I/O, o ejecutar operaciones de I/O sobre buffers del SO.

Si el intercambio es excesivo hay hiperpaginacion o trashing.

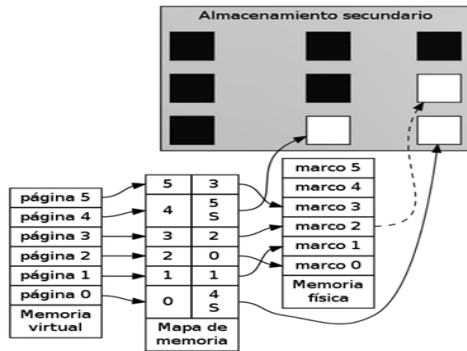
Políticas de Memoria Virtual:

- Fetch (búsqueda) -> Cuando debe llevarse una página a MC
- Placement (colocación) -> A donde debe ubicarse en la MC
- Replacement (reemplazo) -> Cual es la página que se va a reemplazar.

En paginación sobre demanda, el sistema emplea espacio en almacenamiento secundario (típicamente, disco duro) mediante un esquema de intercambio (swap) guardando y trayendo páginas enteras. Utiliza un cargador (pager) “lazy” (flojo o perezoso). Al comenzar la ejecución de un proceso solo se cargan a memoria las páginas necesarias a medida que se van requiriendo. Cargo algunas de las páginas del proceso en MC y el resto en memoria secundaria. Si la página requerida no se encuentra en memoria el pager deberá cargarla.

La técnica opuesta a Lazy es Eager en el que se carga todo al momento inicial.

También tendré una MPT que me indicará cuales de las páginas están en memoria y cuales en memoria secundaria.

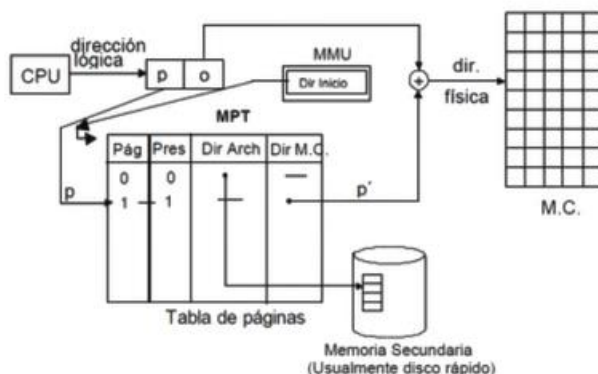


Fallo de página

Cuando un proceso hace referencia a una dirección y la misma no está en la memoria se produce el fallo de página (hay 10 páginas, se hace referencia a la 9 y no está cargada en memoria). No es una excepción o trap, es solo un fallo de página. Se produce un trap si por ej busco la página 14 (que no existe). Habrá una MPT por cada proceso y una MFT única para todo el sistema (como en el caso de la paginación pura o simple).

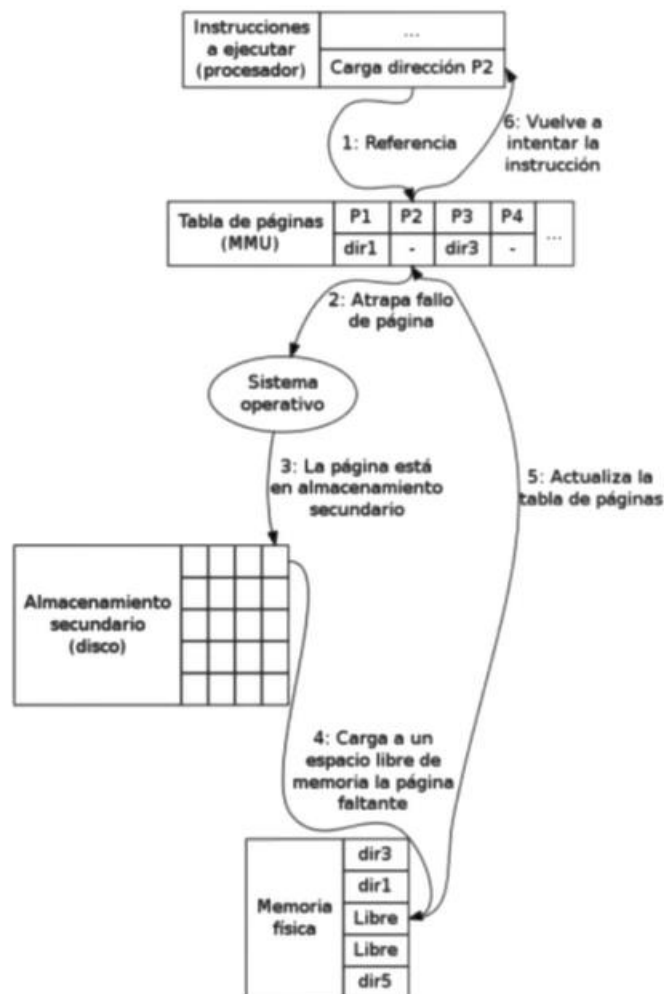
- 1) Se verifica en el PCB si esta solicitud corresponde a una página que ya ha sido asignada a este proceso.
- 2) En caso de que la referencia sea inválida, se termina (suspende) el proceso.
- 3) Procede a traer la página del disco a la memoria. El primer paso es buscar un marco disponible (por ejemplo, por medio de una tabla de asignación de marcos).
- 4) Solicita al disco la lectura de la página en cuestión hacia el marco especificado.
- 5) Una vez que finaliza la lectura de disco, modifica tanto al PCB como al TLB para indicar que la página está en memoria.
- 6) Termina la suspensión del proceso, continuando con la instrucción que desencadenó el fallo. El proceso puede continuar sin notar que la página había sido intercambiada.

Si no hay ningún frame libre escojo un frame, quito una página de MC y coloco la que necesito.



En la tabla MPT tendré el número de página, un bit que me indique si la página está en memoria, la dirección en memoria central (en el frame) si está, o la dirección en la memoria virtual. Además, un bit que indique si la página fue modificada, porque cuando escojo una víctima, si la página fue modificada hay que guardarla en la memoria virtual.

El direccionamiento es similar al de segmentación o paginación puro simple. Numero página, busco el frame, le sumo el desplazamiento y accedo.



Elección de la víctima

- La página ya se encuentra en MV (memoria virtual)??
- Fue modificada en MC ?
- Es del mismo proceso o de otro?
- Cuándo se la accedió por última vez?
- Cuándo se modificó?

Considerando esto, se pueden implementar diferentes algoritmos para la elección de la página víctima.

Anomalía de Belady

En general, si se le asignan más frames a un proceso, en consecuencia, deberían producirse menos fallos de página, pero en algunos algoritmos, con determinadas secuencias de llamadas, se pueden presentar más fallos de página.

Reemplazo de páginas. Algoritmos

FIFO

Mantiene por cada página una marca de tiempo de cuando se cargó y cuando escoge la víctima, elige la que tiene la más antigua. Este algoritmo es simple pero no es de lo más optimo y además es vulnerable a la anomalía de Belady.

Optimo (OPT, MIN)

El algoritmo va a ir escogiendo la que no se va a usar por más tiempo. Requiere reconocer la cadena de referencias a priori (futurismo). Solo sirve para la comparación con respecto a otros algoritmos. No se puede implementar, se usa con fines comparativos al momento de implementar un algoritmo. Si la tasa de fallo de página se aproxima al optimo, el algoritmo es muy bueno.

LRU (Less recently used)

Busca acercarse a OPT prediciendo cuando será la próxima vez en que se emplee cada una de las páginas que tiene en memoria basado en la historia reciente de su ejecución. Elige la página que no ha sido empleada desde hace más tiempo. Por su complejidad, requiere soporte de HW. También usa una timestamp en cada actualización de la página (lectura o escritura).

MFU (Most frequently used) / LFU (Less frequently used)

Se implementa como LRU, pero en lugar de registrar tiempo se registran las invocaciones. Costoso de implementar. Bajo rendimiento. Usan un contador de referencias.

MFU elimina a la página más usada (supone que no se seguirá utilizando). El LFU escoge la página menos utilizada.

Bit de Referencia

La tabla de páginas tiene un bit adicional “referencia de acceso”. Cuando inicia la ejecución el bit de “referencia” está apagado. Cada vez que se referencia el frame, el bit se enciende. Periódicamente el SO resetea el bit de referencia. Ante un fallo, se aplica FIFO entre los que tengan el bit de referencia apagado.

Columna de Referencia

Similar al bit de referencia, pero usando una “columna de bits”. Cuando ejecuta el SO el RESET, hace un right shift del valor a la siguiente posición y se descarta el bit menos significativo. Ante un fallo, se aplica FIFO entre los que tengan el valor de la columna más bajo.

Segunda Oportunidad (o reloj)

Similar a Bit de Referencia. Mantiene un apagador del bit de referencia. Ante un fallo si el bit está apagado, es la víctima, sino se enciende para darle una segunda oportunidad.

Segunda Oportunidad mejorado

Similar a la segunda oportunidad, pero implementa 2 bits (referencia, modificación).

- (0,0) candidato ideal para su reemplazo.
- (0,1) no es tan buena opción, porque es necesario escribir la página a disco antes de reemplazarla, pero puede ser elegida.
- (1,0) fue empleado recientemente, por lo que probablemente se vuelva a requerir pronto.
- (1,1) sería necesario escribir la página a disco antes de reemplazar, hay que evitar reemplazarla.

Ante un fallo, si el bit esta apagado, es la víctima, si no, se enciende para darle una segunda oportunidad.

Asignación de Marcos

Cuando yo cargo un proceso el SO tiene que asignar un mínimo de marcos que necesita el proceso para ejecutarse. Después dependiente de si el pager es Eager o no, se asignarán más o menos frames, pero siempre hay un mínimo de marcos necesarios.

También se debe definir el ámbito de algoritmo de reemplazo de páginas.

- Reemplazo local: El objetivo es mantener tan estable como sea posible el cálculo hecho por el esquema de asignación empleado (la tasa de fallo de un proceso va a depender solo del mismo proceso). Las únicas páginas que se considerarán para su intercambio serán aquellas pertenecientes al mismo proceso que el que causó el fallo.
- Reemplazo global: Los algoritmos de asignación determinan el espacio asignado a los procesos al ser inicializados. Los algoritmos de reemplazo de páginas operan sobre el espacio completo de memoria, y la asignación física de cada proceso puede variar según el estado del sistema momento a momento. Al momento de escoger una víctima, se pueden elegir páginas de cualquier proceso. Tiene mejor rendimiento que el de reemplazo local, pero puede generar que la tasa de fallo de un proceso se vea afectada por este reemplazo.
- Reemplazo global con prioridad: Es un esquema mixto, en el que un proceso puede sobrepasar su límite siempre que le robe espacio en memoria física exclusivamente a procesos de prioridad inferior a él. Esto es consistente con el comportamiento de los algoritmos planificadores, que siempre dan preferencia a un proceso de mayor prioridad por sobre de uno de prioridad más baja.

Hiperpaginación

Sucede cuando la frecuencia de reemplazo de páginas es tan alta que el Sistema no puede avanzar. Todo (o casi todo) el trabajo realizado es overhead. Un pager perezoso va a cargar pocas páginas y esto va a permitir que el sistema tenga más procesos corriendo. Todo proceso que yo quiero ejecutar debe tener un mínimo de páginas cargadas en MC. El nivel de multiprogramación dependerá del tamaño de mi memoria RAM, del tamaño de mi memoria virtual y del manejo que el SO haga de esto. Si tengo una cantidad de memoria virtual excesiva y tengo un porcentaje de las páginas cargadas en memoria virtual muy alta, cuando el proceso se ejecute se producirán muchos fallos de página. Ahí se produce la hiperpaginación.

- Si la política de asignación es local, alguno/s proceso/s tiene/n poco/s frames asignado/s
- Si la política es de asignación global, hay demasiados procesos en ejecución.

Síntomas:

- La tasa de page faults aumenta considerablemente.
- Se incrementa el tiempo de acceso efectivo a memoria.
- La utilización del procesador decae
- No se realiza ningún trabajo ya que los procesos se dedican a paginar.

Solución posible: Reducir el grado de multiprogramación (suspender procesos)

Sistemas Mixtos

Segmentación con paginación por demanda

- Combinan las técnicas de Segmentación y de paginación.
- Agrupo mis páginas de código en un segmento, las de heap en otro segmento, las de archivos en otro segmento, etc.
- No existe el concepto físico de segmento, el segmento es una agrupación lógica de páginas.
- La memoria está dividida en frames y el proceso en páginas.
- En general los segmentos tienen un tamaño múltiplo de páginas.
- No se necesitan tener cargadas en memoria central todas las páginas de los segmentos.
- La dirección virtual se organiza en 3 partes.

Nro. de Segmento	Nro. de Página	Offset (Desplazamiento)
s	p	o

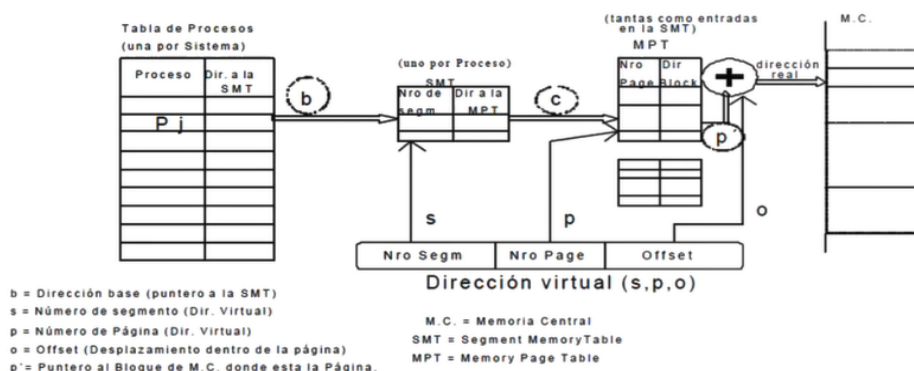
Ventajas:

- Permite compartir segmentos.
- No es necesario cargar la totalidad de los segmentos en memoria central ni la totalidad de las páginas. Solo lo que se necesite.
- No se requiere compactación

Desventajas:

- Requiere más HW para el direccionamiento.
- Es más lento en la ejecución (por el mecanismo de traducción de las direcciones virtuales)
- El SO ocupa más memoria.
- Aumenta la fragmentación interna.

Direccionamiento

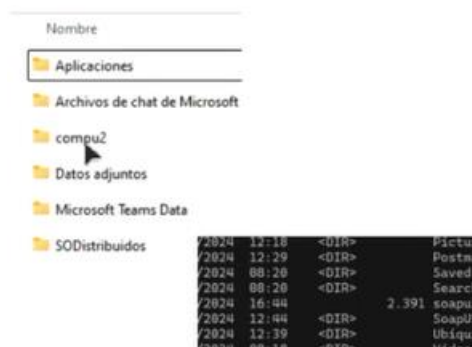


Módulo 6: File System y Entrada / Salida

File System

El módulo File System es que se encarga de mostrarnos la información, cómo se muestra abstrayendo la complejidad de funcionamiento. La información de los distintos dispositivos de E/S que tenemos se muestra abstrayendo la complejidad. El file system no trabaja en el mundo sincrónico. Trabaja desde el módulo de entrada salida hasta los dispositivos.

La visión del file system:



Esta es la visión que nos muestra el SO. Es la visión lógica de lo que el usuario quiere ver, lo que el usuario conoce. La unidad de trabajo es el archivo.

- Las computadoras pueden almacenar información en diferentes soportes físicos: disco, cintas, etc. Cada uno de estos dispositivos tienen su propia característica y organización física.
- El SO hace una abstracción de las características físicas de los dispositivos de almacenamiento para definir una unidad lógica de almacenamiento llamada archivo (file).
- Un archivo es una estructura (de una sola dimensión - vector) de datos que el SO vincula en el aspecto lógico y en el físico.
- Un archivo, desde el punto de vista del sistema es la unidad mínima de almacenamiento de información.
- El sistema de Gestión de Archivos (File System) se integra con el Kernel, el Administrador de Memoria Central y el Gestor de Entrada / Salida.

Funciones del File System

El principal objetivo de un sistema de archivos es permitir las operaciones y accesos en forma segura sobre los soportes para almacenar, modificar, eliminar o recuperar la información. Y administrar el espacio de almacenamiento secundario.

Soporte del espacio ocupado

Una de las funciones del file system es la de mantener (tarea de soporte) el espacio ocupado. Necesito saber cómo está mi espacio ocupado para tener referencias de donde está cada cosa.

Soporte del espacio libre

Además, necesito mantener el espacio libre.

Tener una estructura de directorios

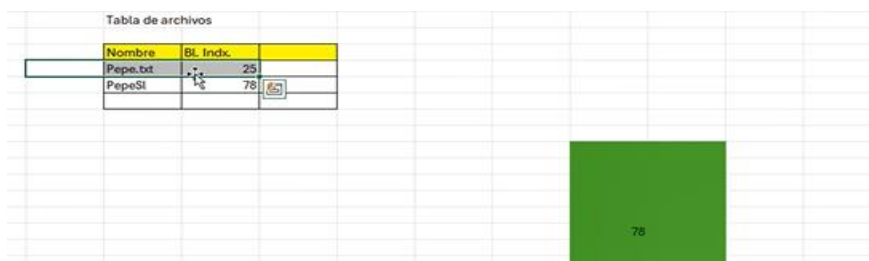
- Árbol enario (Windows): podemos trabajar con un árbol invertido. Estos árboles pueden tener n cantidad de niveles (no hay máximo), son los llamados árboles enarios. Soporta solo soft links.
- Grafos acíclicos (Unix): Soporta soft links y hard links.

Links

Estructura pensada para poder compartir cosas aprovechando tener el dato en un solo lugar del disco. Nos permiten tener una referencia a otra parte de mi file system (puede ser a un archivo o directorio).

Soft link (Acceso directo) link simbólico

Crea un nuevo archivo con un bloque de disco que contiene la información del archivo al que apunta. Ocupa lugar en el file system (es un archivo). Es como un puntero indirecto.



En este ejemplo el bloque de archivo 78 tiene una referencia a la tabla de archivos.

Hard link

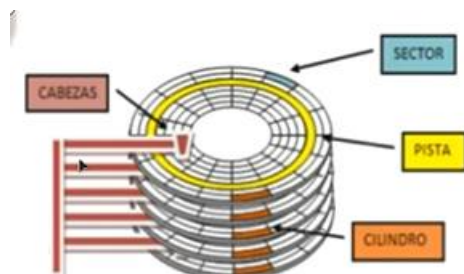
Es una referencia nueva en la table de archivos a un archivo que ya existe (no se puede hacer en Windows). Esta referencia, es una referencia en el árbol de directorios a un archivo apuntado. Un archivo se borra recién cuando se borre el último hard link de ese archivo. Cada hard link puede tener distintos permisos. No ocupa lugar en el file system, por lo que no tengo desperdicio de espacio.

Discos

Son el principal medio de almacenamiento desde hace más de 40 años. Se utilizan para la denominada área de intercambio (swap). Prácticamente todo el SW hace uso de este tipo de almacenamiento, sea en forma permanente o temporal.

Hardware del Disco

- Los discos se constituyen sobre láminas de óxido ferroso.
- Poseen distintas superficies (platos), que pueden tener dos caras.
- Estas caras se dividen en pistas y las pistas en sectores.
- Las pistas equivalentes sobre distintas superficies se denominan cilindro.
- La cabeza lecto-grabadora se mueve a la pista correcta (tiempo de búsqueda) y electrónicamente se selecciona la cara correcta; entonces, esperamos (tiempo de latencia) que el sector requerido pase bajo la cabeza.



En los discos actuales se utilizan las dos caras, en los viejos esto no pasaba porque son más proclives a dañarse (más delicados). Cada pista concéntrica tiene siempre la misma cantidad de sectores y siempre pueden guardar la misma cantidad de información en cada sector.

El disco siempre tiene la misma velocidad de trabajo y sólo puede escribir y leer cuando está en la velocidad de trabajo (régimen). Normalmente cuando está leyendo una cabeza lectora grabadora, en la otra no se puede leer porque se mezclarían los datos en el bus. El sector central del disco (parking zone) es el lugar donde la cabeza lectora grabadora descansa cuando no está trabajando.

Cuando quiero leer el disco indico HCS (Cabeza -Cilindro – Sector), el brazo se mueve desde adentro hacia afuera (ahí se produce el tiempo de búsqueda) hacia el cilindro que quiero leer, una vez que estoy posicionado ahí tengo que esperar que el sector que quiero leer pase por debajo de la cabeza lecto grabadora (tiempo de latencia), luego me queda el tiempo de leer propiamente dicho (tiempo de transferencia).

Dispositivos en estado sólido

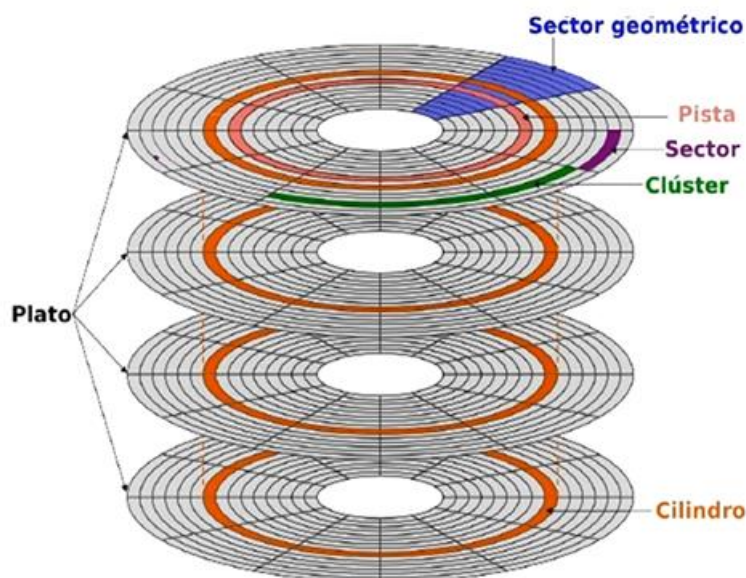
Son dispositivos de almacenamiento que no incluyen piezas móviles. Los tiempos de búsqueda y latencia son muy bajos. Existen implementaciones basadas en:

- NVRAM
- Memorias Flash
- SSD

Si tengo un sistema operativo con un dispositivo de estado sólido. Que algoritmo usar. Es prácticamente lo mismo, pero sería mejor un FCFS, para no perder tiempo en el algoritmo.

Sistemas basados en Disco

- Un disco se divide en pistas
- Cada pista se divide en sectores
- El sector es la más pequeña unidad de información que puede ser leída o escrita en un disco
- Los sectores varían de 512 bytes a 1024 (hay más grandes)
- De 9 a 65 sectores por pista
- Y de 75 a 3000 pistas por superficie
- Los sistemas grandes pueden tener varios platos de discos (cada plato con 2 superficies: superior e inferior)
- Para acceder a un sector se debe especificar la pista (cilindro), la superficie o cara (Cabeza Lectora / Escritora) y el sector.
- Estos elementos constituyen la dirección en el soporte (Cylinder, Head, Sector)



Un cilindro es un conjunto de pistas que están en la misma posición en el disco, pero en diferentes platos. El SO trata al disco como a un arreglo unidimensional de bloques de disco, donde cada bloque contiene uno o más sectores.

Típicamente las direcciones de un bloque se incrementan a partir de todos los sectores de una pista, luego todas las pistas de un cilindro y finalmente desde cilindro 0 al último disco.

El sistema operativo va grabando por cilindro. Por ej. Sector 0 de la pista 0 superficie 0, luego sector 0 de la pista 0 superficie 1. Esto es porque el plato debe moverse lo menos posible. En el mismo plato no se mueve.

Sea ns (número de sectores por pista) y np (número de pistas por cilindro), entonces podemos convertir una dirección de disco (CHS) dado por cilindro c , superficie h , sector s a un bloque unidimensional b (LBA).

$$b = s + ns * (h + c * np)$$

Catalogación de los archivos en el soporte

Para poder asignar y desasignar espacio en el soporte del Sistema Operativo divide el disco básicamente en 3 áreas:

- Área de datos fijos. Fixed Data Área -FDA, que incluye la Tabla de Particiones (es una tabla que tiene las particiones lógicas del disco. (Ej. disco C y disco D) y el Sector de Boot (donde está el loader del SO) (partition table y Boot Sector). Hay un solo Master Boot Rector activo, aunque haya varias particiones.
- Área de Catálogo: área del Espacio libre y del espacio ocupado (Free Space List y File Allocation Table). Esta última está compuesta por un área específica de Directorio (File Directory Block) de varios niveles, que incluye el Master File Directory y al User File Directory. (Dinámica). Es donde el SO guarda todos sus datos administrativos. Normalmente tiene toda la información administrativa para poder mantener el file system. Cada partición tiene un área de catálogo.
- Área de Datos (donde se localizan físicamente los datos de los archivos).

Técnicas para mantener el espacio ocupado

La gestión del Almacenamiento Secundario básicamente está representada por la asignación del espacio, cuyo objetivo es utilizar eficazmente el espacio y posibilitar el acceso rápido a la información almacenada.

El espacio de almacenamiento se relaciona entre lo que no está asignado (libre) y el que está ocupado por los archivos (asignado) y generalmente se utiliza una tabla para administrar ese espacio llamado FAT (File Allocation Table).

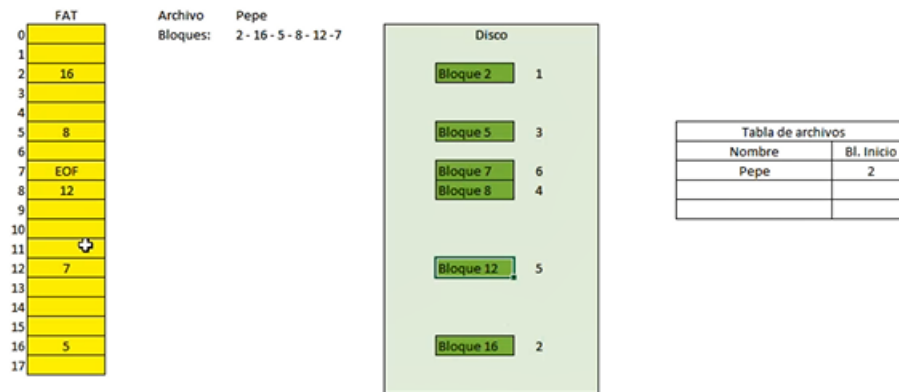
FAT (File Allocation Table)

Tengo una estructura, la cual es un vector de punteros con un puntero por cada bloque de disco. Además, tengo un archivo que mide n bloques. En mi tabla de archivo (tabla de directorios - si esta tabla de directorios está dentro del área de catálogo depende de la implementación del SO) tengo el archivo y el puntero al bloque inicial, para encontrar los sucesivos bloques debo buscar en la FAT. Es como una lista enlazada que en vez de

tener el puntero dentro del bloque de datos lo tiene por fuera. La FAT está guardada en el área de catálogos.

FAT-16 – Punteros de 16 bits. Nos dice cuanto es el tamaño máximo de partición o de tamaño de disco por cada file system utilizado.

Esta técnica no tiene acceso directo. Es un sistema linkeado.



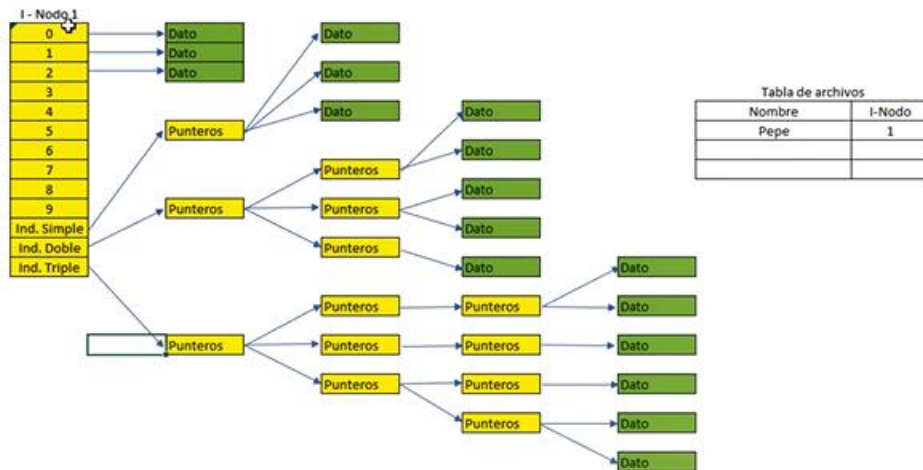
I-Nodos (nodos índices)

Es una estructura de datos que consta de un encabezado donde hay datos del archivo y luego los 39 bytes de información de direcciones. Los atributos del archivo se almacenan en el I-nodo. Cuando se abre un archivo se trae su I-nodo a memoria. Los I-Nodos son una estructura indexada (haciendo cuentas se dónde debo acceder para poder leer determinado dato).

Los I-Nodos se encuentran en la I-Lista, la cual está en el área de catálogo. La I-Lista se crea a la hora de formatear el File System. Se reserva el lugar para el I-Nodo, pero no de los punteros de indirección, los mismos van en el área de datos.

Un I-nodo incluye 39 bytes de información de direcciones:

- 10 punteros a bloques de datos
- 1 indirección simple (puntero que apunta a un bloque con punteros que apuntan a bloques de datos)
- 1 indirección doble
- 1 indirección triple



Ejemplo de un tamaño máximo de un archivo:

- En UNIX System V el bloque mide 1 Kb
- Cada bloque puede contener 256 punteros a bloque
- Por lo tanto, el tamaño máximo de un archivo es:

$$(10 + 256 + 256^2 + 256^3) \times 1\text{kb} \approx 16\text{ Gb}$$

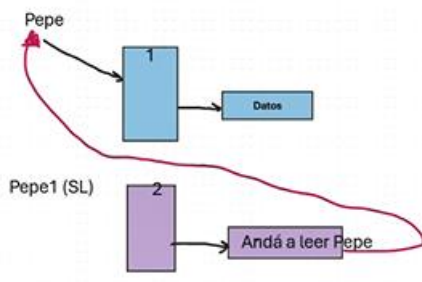
Ventajas:

- El I-nodo es de tamaño fijo y relativamente pequeño, por lo tanto, puede almacenarse en memoria durante períodos largos.
- A los archivos más pequeños se puede acceder con pocas o ninguna indirección.
- El tamaño máximo teórico de un archivo es suficientemente grande para satisfacer prácticamente todas las aplicaciones.

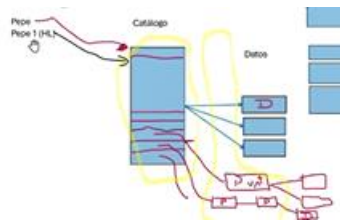
Desventaja

- Los punteros de indirección están en área de datos. Por lo menos la primera vez debo traer eso a memoria. Esto hace que sea lento traerlo en principio.

Soft Link



Hard Link

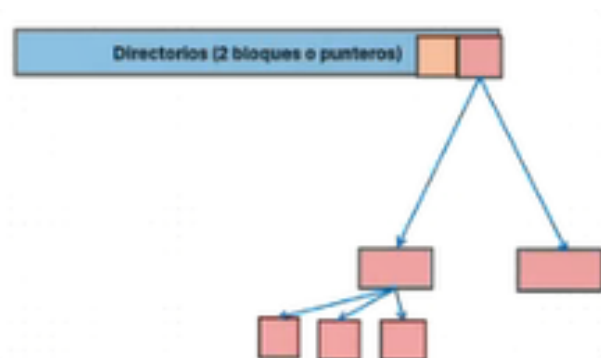


Bi-Tree

Árboles binarios, hay muchos tipos. Puede estar o no balanceado, hay muchos file system con estructura de Bi-Trees porque son muy rápidas para buscar datos. Cada bloque de datos tendrá muchas hojas (podría estar todo el Bi-Tree en un sólo bloque).

HPFS (High Performance File System)

Lo único que quedo de los mismos es el NTFS de Windows. Genera una estructura en el disco, donde en el área de directorio tengo dos bloques o dos punteros (mezclo área de datos y área de directorios). Si mi archivo ocupa hasta dos bloques los ocupo en el área de directorio, si el archivo empieza a crecer lo transformo en un Bi-Tree. Puedo dejar el primer bloque de datos en el directorio y el segundo lo pueda transformar en el Bi-Tree.



Administración del Espacio Libre

Bit Map o Bit Vector (Mapa de Bits)

- Frecuentemente se implementa como un vector o mapa de bits en la FAT.
- Cada bloque del disco se representa con un bit.
- Si el bloque está libre se le asigna un 0, caso contrario el bit vale 1 que indica que está ocupado.

Ventajas: Sencilla implementación. Relativamente poco espacio ocupado.

Desventajas: No posee grandes desventajas.

(Utilizado por DOS)

Lista Enlazada de Bloques Libres

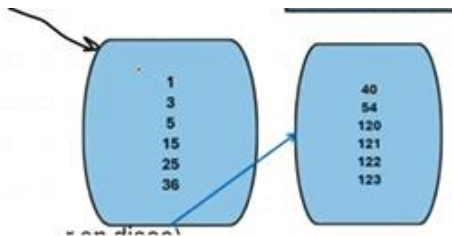
Otra solución es vincular (link) todos los bloques libres, guardando un puntero al primer bloque libre. Este bloque tiene un puntero al siguiente bloque libre y así sucesivamente.

El comienzo de la lista lo marca el puntero Free Space List Head (FSLH)

Ventajas: Muy poco espacio ocupado

Desventajas: No es muy eficiente ya que para recorrer la lista debemos leer cada bloque del disco. Es altamente vulnerable ya que si pierdo un nodo de la lista pierdo el resto de la lista.

Bloques de Direcciones Libres



Almacena las direcciones de n bloques libres en el primer bloque libre disponible en el soporte (en el área de catálogo).

Se apuntan con un puntero a los primeros $(n-1)$ bloques que están realmente libres.

Luego el último puntero contiene la dirección del próximo bloque que contiene las direcciones de otros n bloques libres. El primer bloque está en el área de catálogo, si se expande, el segundo bloque va a estar en el área de datos. Esta técnica es la más usada.

(Utilizado por Linux)

Ventajas: Es más eficiente en cuanto a velocidad ya que con leer un solo bloque se conoce la dirección de una gran cantidad de bloques libres. Mientras más espacio haya ocupado en el disco menos espacio requiere la lista de bloques libres.

Desventajas: Cuando el disco está poco ocupado, la lista es muy grande y lleva mucho tiempo recorrerla.

Bloques de Direcciones Libres Contiguas

Se guarda la dirección del primer bloque libre y el número de bloques contiguos libres que le siguen (en el área de catálogo).

Cada entrada en la lista de espacios libres consiste entonces en una dirección en disco y un número.

Este método es particularmente útil cuando se utiliza almacenamiento continuo o secuencial.

Ventajas: Utiliza menos espacio del almacenamiento que el método anterior. Muy eficiente para almacenamiento continuo.

Desventajas: Si los sectores están muy dispersos, es poco eficiente.

Métodos de Asignación de espacio para los Archivos

Se usan tres métodos de asignación de espacio de disco:

Contiguo

- Requiere que cada archivo ocupe un conjunto de direcciones contiguas o consecutivas en disco cuyo espacio debe ser declarado en la creación del archivo.
- La posición del archivo en el disco queda definida por la dirección del primer bloque y su longitud.
- El acceso al bloque $b+1$ luego del b normalmente no requiere movimiento de cabeza y a lo sumo solo una pista
- Para un acceso directo el bloque i de un archivo que comienza en el bloque b , se accede inmediatamente al bloque $b+i$
- Así vemos que para asignación contigua es posible tanto el acceso secuencial como el directo.
- Tabla de archivos: Nombre - Bl. Inicio - Cant. Bl.

Nombre	Bl. Inicial	Cant. Bloq
Pepe.txt	7	4

Ventajas:

- Es muy fácil acceder a los archivos.
- Es ideal para accesos secuenciales ya que la cabeza no tiene mucho que viajar.

Desventajas:

- A medida que el disco se va llena, se torna difícil encontrar espacio para un nuevo archivo.
- Cuando un archivo se crea, generalmente no se sabe cuántos bloques va a ocupar.
- Si un archivo tiene que crecer y no tiene más espacio consecutivo al final, hay que moverlo a una nueva localización.
- Hay fragmentación externa (cada bloque es para un archivo y cada archivo tiene sus propios bloques mínimo uno, entonces hay bloques sueltos que no corresponden a ningún archivo).

Asignación Dinámica de Almacenamiento

- El espacio en disco puede considerarse como una colección de segmentos libres y usados, siendo cada segmento un conjunto contiguo de bloques.
- Un segmento libre se lo conoce como hueco
- El conjunto de huecos es examinado para determinar cuál es el mejor a asignar.


Las estrategias más comunes son: First-fit, Best-fit, Worst-fit.

Las simulaciones han demostrado que tanto el First-fit como el Best-fit son mejores que el Worst-fit tanto en tiempo como en utilización de espacio.

Compactación o Defragmentación: Se compactan todos los huecos en uno solo. El costo de este proceso es el tiempo necesario para efectuarlo.

Vinculado -link-

- Cada archivo es una lista enlazada de bloques de disco.
- Los bloques pueden estar distribuidos en cualquier parte del disco.
- La tabla de archivos contiene el nombre del archivo, un puntero al primer bloque y un puntero al último bloque (si corresponde).

Nombre	BL. Inicial	
Pepe.txt		7

Siempre guardo un espacio para este puntero y en un caso border (tamaño del archivo igual al del bloque) tendré que reservar otro bloque para ese byte que falta del archivo (ya que en el anterior el ultimo estaba reservado para el puntero) más el byte de puntero de este nuevo bloque.

Ventajas:

- No hay fragmentación externa.
- Tampoco es necesario declarar la longitud al crearlo, ya que puede crecer en tanto existan bloques libres.
- Un archivo puede continuar creciendo tanto como bloques libres haya en el disco.

Desventajas:

- Los punteros ocupan espacio extra.
- Es contraproducente implementar acceso directo con este método.
- Altamente vulnerable (Una solución parcial es usar una lista doblemente enlazada)

Indexado

- Cada archivo tiene su propio bloque de índice (index block), en el área de catálogo, que es un vector de direcciones de bloques en el disco.
- La i-ésima entrada en el l bloque de índices apunta al i-ésimo bloque del archivo.
- El directorio contiene la dirección del index block
- Para un archivo más grande se utilizaría un puntero a otro index block, y para archivos aún más grandes se podría tener un tercer index block y así sucesivamente.
- Una variante es usar un bloque de índices separado que apuntan a los distintos bloques índices que a su vez apuntan a los bloques del archivo.

Nombre	Bl. Indc.	
Pepe.txt	25	

Ventajas:

- Soporta acceso directo y secuencial
- No tiene fragmentación externa

Desventajas:

- El espacio ocupado por punteros index block es generalmente mayor que el ocupado por la asignación enlazada.
- Es ineficiente para bases de datos grandes.
- Sufre de desperdicio de espacio en disco debido al bloque de índices.

Acceso Secuencial

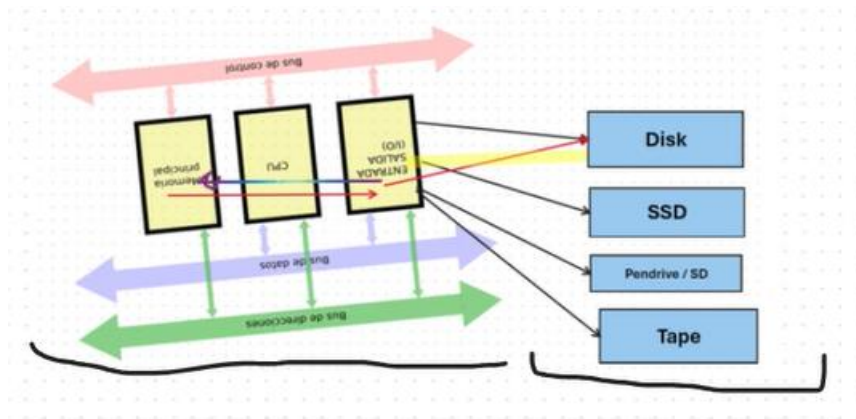
Es el método más simple, la información del archivo se procesa ordenadamente y de a un registro por vez. Un read lee la próxima porción del archivo y automáticamente avanza el puntero de archivo. Un write agrega datos al final del archivo y avanza al nuevo fin de archivo.

Acceso Directo

El archivo está formado por registros lógicos de tamaño fijo para permitir que los programas lean y escriban rápidamente y sin un orden en particular. Este método se basa en el modelo de archivo de disco. El acceso directo permite leer o escribir bloques en forma arbitraria. Así se puede leer el bloque 14 y luego el 53 y luego escribir el 7.

ENTRADA SALIDA

La visión de entrada salida:



Cuando hablamos de cómo está almacenado lo que el usuario usa, como lo recupero, como lo traigo, hablamos de E/S.

El bus de sistema está compuesto por el bus de direcciones (se transmiten las direcciones), el bus de datos (se transmiten los datos) y el bus de control (se transmiten las señales de control que genera la unidad de control y que sirven entre otras cosas para definir las operaciones y la sincronización de las tareas). Tanto la CPU como la memoria se comunican a través de los buses del sistema, esta comunicación es sincrónica porque está regida por un clock. Además, el bus de sistema se comunica con un broadcast (envía un mensaje que llega a todos los dispositivos, y el dispositivo al cual le corresponde el mensaje lo toma). Cuando por ejemplo el procesador le quiere enviar una palabra a la memoria, pone la palabra en el bus de datos, pone la dirección de la memoria en el bus de direcciones, y en el bus de instrucción pone la instrucción, entonces con el siguiente pulso de reloj viajan todos por su respectivo bus todo lo mencionado anteriormente. El procesador a su vez tiene especificado cuál es la velocidad de cada bus. Cuando los periféricos no se conectan directamente al bus del sistema, eso se debe a los motivos mencionados arriba del gráfico anterior. El módulo que se conecta con los dispositivos de entrada/salida (los cuales tienen todas distintas velocidades) trabaja a la velocidad del dispositivo más lento. Además, el formato de los datos que se transmiten a través de los buses (palabra), no es el mismo formato que utiliza por ejemplo un disco (bloques), o una pantalla (píxeles).

8 bits = 1 byte, 16 bits = una palabra, 32 bits = una doble palabra, 64 bits = una cuádruple palabra.

En consecuencia, es necesario un módulo de E/S que tiene dos funciones principales desde el punto de vista del HW.

- Proveer una interfase con el procesador y la memoria central vía el bus del sistema.

- Proveer una interfase a uno o más de los dispositivos periféricos por vínculos especialmente diseñados.

El sistema de Gestión de la Entrada / Salida se ocupa de la organización, administración y operación de los dispositivos de E/S, este tiene la lógica para gestionar dichos dispositivos (chips de menor poder con sets de instrucciones para manejar el dispositivo). Hay un módulo de entrada salida para cada dispositivo (Ej. placa de red, placa de video, etc.)

Este módulo de E/S gestiona el acceso de los dispositivos a los distintos buses del sistema. Hace años estas comunicaciones se gestionan mediante los chips denominados Puente Norte y Puente Sur:

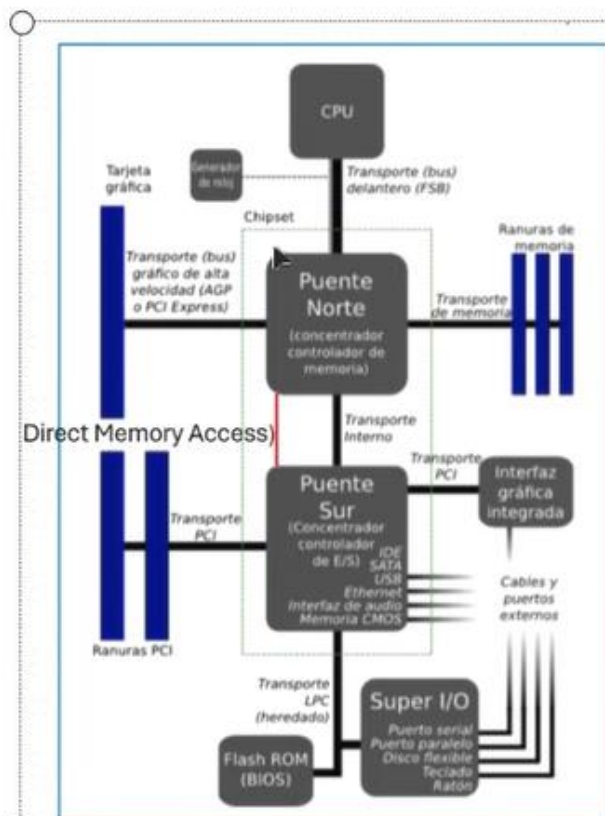
Puente Norte (Northbridge): Conectado directamente a la CPU, gestiona los dispositivos de más velocidad. Ej. memoria, GPU, PCI Express.

Puente Sur (Southbridge): Se conecta al puente norte, gestiona el resto de los dispositivos. Ej. discos, placa de red, USB, interfaz de audio.

Lo importante a nivel de SO es que el dispositivo externo se conecta a un módulo de I/O que es el que hace de interfaz a través del bus del sistema para llegar a la memoria y al controlador.

Funciones del Módulo de E/S:

1. Administrar el HW (con un driver).
2. Buffering de datos (para amortiguar diferencias de velocidades). Este buffer de datos, cuanto más cerca del disco este, mucho más rápida y precisa hará a mi E/S.
3. Comunicación con el procesador (para poder generar las interrupciones).
4. Detectar errores (con el driver).
5. Manejo de errores (con el driver).



En el puente norte van conectados los dispositivos más rápidos. En el puente sur van conectados los dispositivos más lentos.

Los dispositivos lentos van a entrar a nuestra memoria desde el puente sur a través del puente norte.

El DMA se encarga de que los dispositivos externos puedan escribir directamente en memoria. Esto lo hago a través del puente norte (el DMA le roba el uso de los buses al CPU y pasa por el puente norte para llegar a la memoria)

Cuando hay un error el puente norte escribe en el vector de interrupciones, y el vector enciende el FLIH. Ahí ya es una cuestión de la CPU.

Control y Temporización

El controlador le pregunta al dispositivo si está en condiciones de operar y el procesador comienza con la transferencia o informa que el dispositivo está fuera de línea.

Pasos:

- El procesador interroga al módulo de E/S para verificar el estado(estatus) del dispositivo conectado
- El módulo de E/S devuelve el estado del dispositivo.
- Si el dispositivo esta operable y listo para transmitir, el procesador solicita la transferencia de datos, por medio de un comando al módulo de E/S.

- El módulo de E/S obtiene una unidad de datos (por ejemplo 8 o 16 bits) del dispositivo externo.
- Los datos son transferidos desde el módulo de E/S al procesador.

Comunicación con el procesador:

Si bien el set de instrucciones del dispositivo de entrada y salida está en el módulo de E/S, dichas ejecuciones surgen desde el procesador. Esto significa que hay una serie de comandos que utiliza el SO desde el procesador para comunicarse con el módulo de E/S, el cual por supuesto, debe tener capacidad de entender. También tiene que ser capaz de manejar los datos en el formato que maneja el sistema internamente (traduce para ambos lados).

Incluye:

1. Decodificación del comando: El módulo de E/S acepta comandos desde el procesador. Estos comandos generalmente son enviados como señales a través del bus de control.
2. Datos: Los datos son intercambiados entre el procesador y el módulo de E/S a través del bus de datos.
3. Informe de estados: Por ser los periféricos tan lentos, resulta imposible saber el estado del módulo de E/S. Las señales de estados más comunes son BUSY y READY.
4. Reconocimiento de direcciones: De la misma forma que cada palabra en la memoria tiene una dirección, también la tiene cada dispositivo de E/S.

Comunicación con el dispositivo:

El módulo de E/S no solo debe tener los comandos para comunicarse con el procesador, también debe tener los comandos para poder comunicarse con el dispositivo externo y manejar el formato de datos que maneja el dispositivo. Ej. Si yo quiero leer un disco, le pedirá al disco un bloque y se lo enviará a la memoria en formato de palabras.

- Realiza intercambio de comandos.
- Realiza informe de estados.
- Realiza transferencia de datos.
- Los dispositivos funcionan independientemente del procesador y no utilizan el reloj del procesador, sino el suyo propio.
- Esto constituye una operación asincrónica manejada por las señales de control que son generadas por los dispositivos y sus interfaces.
- A estas señales se las conoce como señales de dialogo (handshaking) que son intercambiadas entre el controlador – dispositivo y el procesador antes de comenzar y al finalizar cada operación de E/S.

Amortiguación (buffering) de datos:

Es una de las tareas esenciales de este módulo. Por ej: Si yo envío información de la memoria al disco, el mismo recibirá la información en formato de palabras a muy alta velocidad; las guardará en un buffer interno, armará un bloque y se las pasará al disco a la velocidad que el mismo las pueda leer.

- Es una tarea esencial del módulo de E/S.
- Mientras que la velocidad de transferencia hacia y desde la memoria o el procesador es bastante alta, la velocidad es varios órdenes de magnitud inferior para la mayoría de los dispositivos.
- El módulo de E/S debe ser capaz de operar a ambas velocidades.

Detección de errores:

- Una clase de error puede ser el mal funcionamiento, mecánico o electrónico, reportado por el dispositivo.
- Otra clase de error consiste en los cambios no intencionales de los patrones de bits como son transmitidos desde el dispositivo al módulo de E/S (puede ser que en el camino algún bit se transforme).

Manejo de interrupciones a bajo nivel:

El manejo de interrupciones al más bajo nivel debe ser realizado por el módulo de E/S.

Por ej. Cuando un disco finalizó una escritura, no es el propio disco quien notifica al procesador, sino el módulo de E/S.

Componentes principales

Controlador o Driver

Es la interfaz del sistema de E/S con el SO a nivel software. Se ocupa de convertir el flujo de bits en bloques o bytes o caracteres.

Software Independiente del Dispositivo

Presenta al dispositivo como un conjunto de registros dedicados que se leen o escriben en cada operación de E/S. A estos registros generalmente se los denomina puerto de E/S.

Procesadores de E/S (IOP)

En los grandes equipos (Mainframe) generalmente se conectan mediante varios buses y un procesador especial de E/S que se ocupa de la gestión. A esta configuración se denomina subsistema de E/S que hace de interface con el procesador y memoria central. Todas las operaciones sobre los periféricos se arrancan con una orden

generada en el procesador para el procesador de E/S o canal, quien realizará las transacciones en forma independiente asumiendo el completo control de estas.

Una alternativa a todo lo anterior son los procesadores de E/S. Esta alternativa es muy poco flexible (aunque en los mainframes no se suelen conectar muchos dispositivos).

Técnicas de E/S

Hay 3 técnicas posibles para las operaciones de E/S:

E/S Programada, también llamada Polling (Escrutinio)

Esta técnica es la más antigua. No permite multiprogramación (no hay tiempos muertos)

- Los datos son intercambiados entre el procesador y el módulo de E/S. El procesador ejecuta el programa que otorga el control directo de la operación de E/S, incluyendo el censado del estado del dispositivo, enviar un comando READ o WRITE, y la transferencia de los datos
- Cuando el procesador emite un comando al módulo de E/S, debe esperar hasta que la operación de E/S se complete.
- Si el procesador es más rápido que el módulo de E/S, se desperdicia tiempo de la CPU.

Esta técnica es bastante poco performante (el SO debe quedarse esperando en loop hasta que el dispositivo termine). Además, cuando la operación terminaba, era el propio procesador el que se ocupaba de transferir información desde el módulo hasta la memoria (o viceversa).

E/S dirigida por interrupciones (Interrupt Driven)

Se agrega HW específico: Vector de interrupciones, FLIH, SLIH. Se puede usar multiprogramación.

Con esta técnica, el procesador no se queda esperando que el dispositivo termine con su trabajo, envía el comando de entrada y salida al módulo y se desentiende de la acción y sigue procesando otras ejecuciones. Cuando el módulo termina envía una señal para que el SO retome la ejecución

- El procesador emite un comando de E/S, continúa con la ejecución de otras instrucciones, y es interrumpida por el módulo de E/S cuando éste ha completado su trabajo.

Tanto la E/S Programada o por interrupciones, el procesador es responsable de extraer los datos de la memoria central para el output o almacenar los datos en la memoria central para el input.

E/S mediante DMA

¿Cómo se resuelve la competencia por el acceso al Bus?

- Por ráfagas: Prioridad de transferencia entre I/O y memoria. Cuando el DMA toma el control del bus, no lo libera hasta haber transmitido el bloque de datos pedido. Se consigue la mayor velocidad de transferencia, pero se tiene inactiva la CPU (parada del procesador). Es mejor para procesos I/O bound.
- Por robos de ciclos: Prioridad la tiene la CPU. Cuando el DMA toma el control del bus, lo retiene durante un solo ciclo (y el CPU no puede tomarlo). Transmite una palabra y libera el bus. Es la forma más usual. Reduce al máximo la velocidad de transferencia y la interferencia del DMA sobre la actividad del procesador. Es mejor para procesos CPU bound.
- DMA transparente: Prioridad la tiene la CPU, I/O solo utiliza si está libre. Se elimina completamente la interferencia entre el DMA y la CPU. Solo se roban ciclos cuando la CPU no está utilizando el bus del sistema (si la CPU va a tomar el bus el DMA no puede tomarlo). No se obtiene una velocidad de transferencia muy elevada. Es mejor para procesos CPU bound.

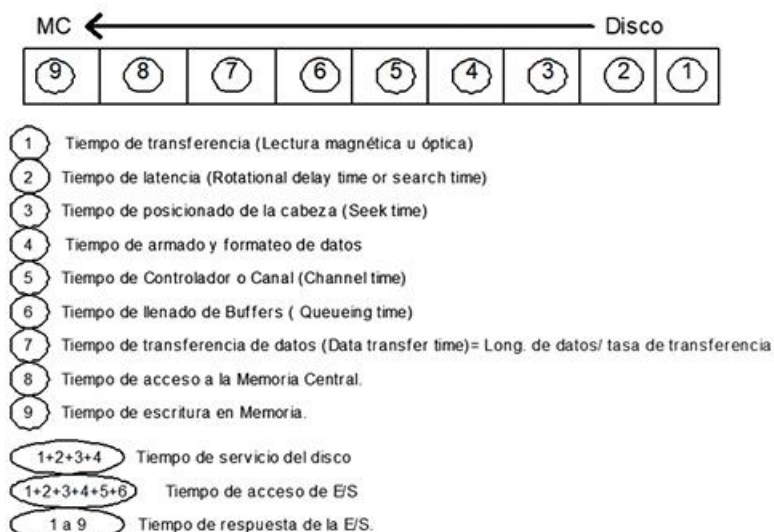
Discos

En entrada salida, depende de qué tipo de disco estemos hablando podemos trabajar:

- Disk: 3 dimensiones
- SSD: 1 dimensión
- Pendrive: 1 dimensión
- Tape: 1 dimensión

En E/S cuando hablamos de disco hablamos de bloques (en File System hablamos de archivo)

Tiempos de entrada salida



Los 3 primeros tiempos son físicos.

1. Es lo que tarda la cabeza lecto grabadora en moverse para pararse en la pista que quiero leer
2. Es el tiempo que tarda el disco en rotar hasta llegar al sector que quiero leer
3. Es el tiempo que tarda en recorrer todo el sector para obtener los datos

Los siguientes son tiempos electrónicos:

4. Esto ocurre en el mismo disco, en la placa controladora que tiene el mismo.
5. Es el tiempo que tarda la información en llegar desde el dispositivo externo hasta la controladora (de red, de disco, etc.)
6. La controladora tiene que guardar los datos y mandarlo al módulo genérico de entrada y salida
7. El tiempo de transferencia hacia la memoria central
8. El tiempo en meterlo a la memoria central
9. El tiempo para escribir en la memoria central

Performance en los discos

De todos los tiempos involucrados en una lectura / escritura a disco, queda claro que los dos tiempos a optimizar son:

- Tiempo de latencia: Con discos que giren más rápido, interleaving (obsoleto), técnicas de cache (como cache de una pista por vez).
- Tiempo de búsqueda: Logrando que la cabeza lecto grabadora deba moverse lo menos posible. Para esto se usan Algoritmos de Planificación de Brazo de Disco. Estos algoritmos se encargaban de ordenar las peticiones de lectura de manera que el brazo tuviese que moverse lo menos posible.

Algoritmos de Planificación

- FCFS (First Come First Served)
- SSTF (Shortest Seek Time First)
- Planificador SCAN
- Planificador C-SCAN
- Planificador LOOK-UP (Algoritmo del Ascensor)
- Planificador C-LOOK-UP (Algoritmo del Ascensor Modificado)

Uno de los cambios más importantes de los últimos tiempos es que las propias controladoras de discos tienen sus propios algoritmos para optimizar el acceso a los bloques de datos, ya que poseen más y mejor información sobre la ubicación de estos.

Los SO modernos poseen algoritmos de planificación que incluyen mejoras, como división en múltiples colas, división entre lecturas y escrituras, entre otras. (Noop, CFQ, Deadline, BFQ, Anticipatory).

