////////// I - Introducción

Hardware: conjunto de componentes físicos de los que conforman el equipo

Software: Conjunto de instrucciones

> **Algoritmo**: Conjunto de pasos para llegar a un objetivo

Sistema operativo

Para algunos autores es únicamente el kernel, y para otros es el kernel más un conjunto de programas y rutinas.

Se usa esta definición:

Es un conjunto de programas y rutinas administran y facilitan el uso del hardware por parte del usuario y sus programas de aplicativos.

Componentes del sistema

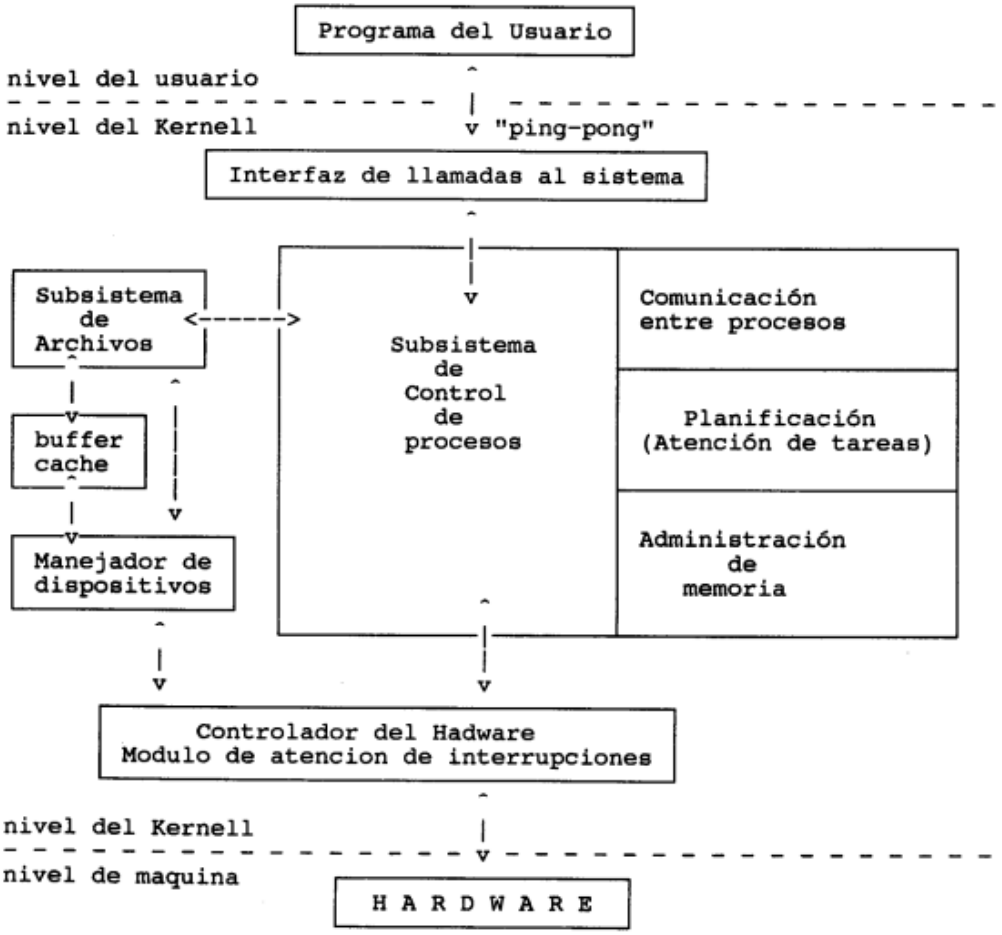
- Gestión de procesos
- Gestión de la memoria principal
- Gestión de archivos
- Gestión del sistema de E/S
- Gestión de almacenamiento secundario
- Redes y protección

El NÚCLEO o KERNEL de un S.O.:
- Es pequeño
- Contiene código altamente utilizado
- Reside en la memoria principal

Se encuentra en un nivel superior, pero muy próximo al Hardware; es el Software de mas bajo nivel.

- Funciones:
- Gestión de memoria principal. (Asig/Desasig)
- Manejo de interrupciones
-Gestión de procesos (Creación/Destrucción, Cambio de estados)
-Manejo de los PCB
-Despacho de procesos
-Sincronización entre procesos
-Comunicación entre procesos
-Soporte de mecanismo de Llamada/Retorno a procedimientos.
-Soporte de actividades de E/S
-Soporte del sistema de archivos
-Ciertas funciones contables/estadísticas del sistema.

Estructura de un Sistema Operativo Ej. Kernell del UNIX



Subsistema de E/S Abstrae el uso de dispositivos. Usado en Unix, consta de: - Componente de gestión de memoria, (buffer, cache y colas) - interfaz general para controladoras de dispositivos - controladoras para dispositivos hardware específicos (engloba los controladores y periféricos)

Controladoras y Cpu :

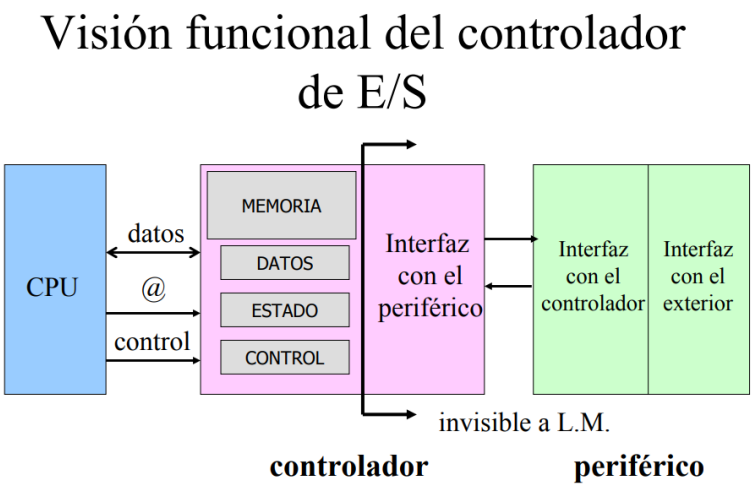
Es la entidad dentro del computador que tiene la responsabilidad de controlar un dispositivo externo e intercambiar los datos entre esos dispositivos y la memoria principal y/o los registros de la CPU. Se conectan mediante el bus.

Funcionan concurrentemente compitiendo por los ciclos de memoria.

Cada controladora se encarga de un dispositivo físico en particular. y poseen buffers locales y conjuntos de registros de propósito especial.

Un módulo se conecta al resto del computador a través del bus del sistema,

Cada SO suele poseer un driver que será el que se comunicara con la controladora



Controlador de interrupciones • Circuito intermediario entre la CPU y los controladores de los periféricos de E/S

Cluster: Dos o más sistemas acoplados que comparten almacenamiento y se conectan por LAN,

diferenciamos programas aplicativos de los del SO:

- Aplicativos de usuario: responden a una necesidad del usuario, usualmente grandes
- Aplicativos partes del SO: Relacionados con el funcionamiento del SO. Pequeños. Ocupan pocos K

Dato:

antes--> El so era un fragmento de código enlazado con los programa, se cargaba todo en memoria y se cargaba como un programa en tiempo de ejecución.

Problemas: bajo uso de recursos principales (throughput). No tenían bit de modo(permite usar modo kernel o modo usuario. PSW)

Proceso de inicio de sistema

El cargador de arranque localiza el kernel, lo carga en memoria principal e inicia su ejecución

en otros casos se tiene un proceso de dos pasos

Al momento de arrancar el procesador busca dentro de la ROM (ya que la ram tiene estado indefinido en este momento)o en EEPROM (memoria de solo lectura programable y eléctricamente borrable)(también se lo conoce como firmware) . la dirección que lleva a la

ejecución de la BIOS, esta hará distintos chequeos y buscará en disco el primer sector (MBR) donde se encuentra la instrucciones para ejecutar al cargador.

se le pasara el control al cargador del gestor de arranque.

el cargador de arranque extrae del disco un gestor de arranque más complejo el cual este a su vez carga el Kernel se encarga de inciar todos los procesos del sistema operativo. Es capaz de realizar otras tareas como diagnósticos.

Control del SO: son controlados por interrupciones y excepciones

SO multitarea debe evitar interferencias entre los procesos de usuarios y entre los procesos y el SO

Instrucciones privilegiadas que pueden dañar a otros procesos (desactivar interrupciones, modificar el mmu)

llamada al sistema lo hacen los procesos de usuario --> el so atiende la sistem call y da tratamiento

Servicios de un SO

El SO nos da servicios para utilizar el hardware (visión como proveedor de servicios)

- Ejecución de procesos: Debe proveernos un ambiente para ejecutar los procesos

- Interfaz: Darnos una forma para comunicarnos con el SO. interfaces de usuario tienen como objetivo facilitar el uso (input/output)

    interfaz texto: cls

    interfaz gráficas: gui

- administración de entrada/salida: manejo de perifericos

- Sistema de archivos (algunos lo incluyen al servicio de E/s)

- Servicio de comunicaciones: comunicación entre procesos de diferentes maquinas.

- Protección y seguridad

en el libro aparecen también..

-Detección de errores
-Asignación de recursos
-responsabilidad

Las llamadas al sistema(system calls) un proceso a nivel de usuario le pide al SO que haga una acción (nivel de kernel).

Se proporciona una interfaz con la que invocar estos servicios . Se usan APIS (interfaz de programación de aplicación) (Por ejemplo Win32, POSIX)

Estructura y diseño de un SO

Diseño: pensamos para qué queremos ese sistema operativo. Se definen objetivos y especificaciones. Afectado por hardware y tipo de sistema.

SO de propósito general: típicamente en pcs caseras o servidores, sistemas administrativos, contables, científicos, etc

SO de propósitos específicos: aquellos diseñados para hacer una tarea en particular (STR) por ejemplo para administrar una máquina.

Estructuras de control del SO: si el SO va a administrar los procesos y los recursos, entonces tiene que disponer de información sobre el estado actual de cada proceso y de cada recurso. el SO construye y mantiene tablas de información sobre cada entidad que está administrando. Se requieren cuatro tipos de tablas:

- Tablas de memoria: se utilizan para seguir la pista de la memoria principal y la secundaria (virtual).
- Tablas de E/S: son utilizadas por el SO para administrar los dispositivos y los canales de E/S del sistema informático.
- Tablas de archivos: ofrecen información sobre la existencia de archivos, su posición en la memoria secundaria, su estado actual y otros atributos.
- Tabla de procesos: el SO debe mantenerlas para poder administrar los procesos.

Mecanismos y políticas:

NO SE DEBEN MEZCLAR

PRIMERO POLITICA DESPUES MECANISMO

políticas: fijo objetivos, quiero hacer algo

mecanismos: la forma en que lo hago

Arquitectura de software:

antes solo se escribían rutinas usando un único espacio de direcciones

- **Modelo monolítico**: Todo el Kernel es un solo espacio de direcciones, y el código va a estar dado por un conjunto de rutinas/procedimientos(dentro del kernel) donde cada uno se llama a otro

  Muchas funciones y se llaman entre ellas
  Muy eficiente
  Al caer un error el error no queda aislado (puede afectar todo)

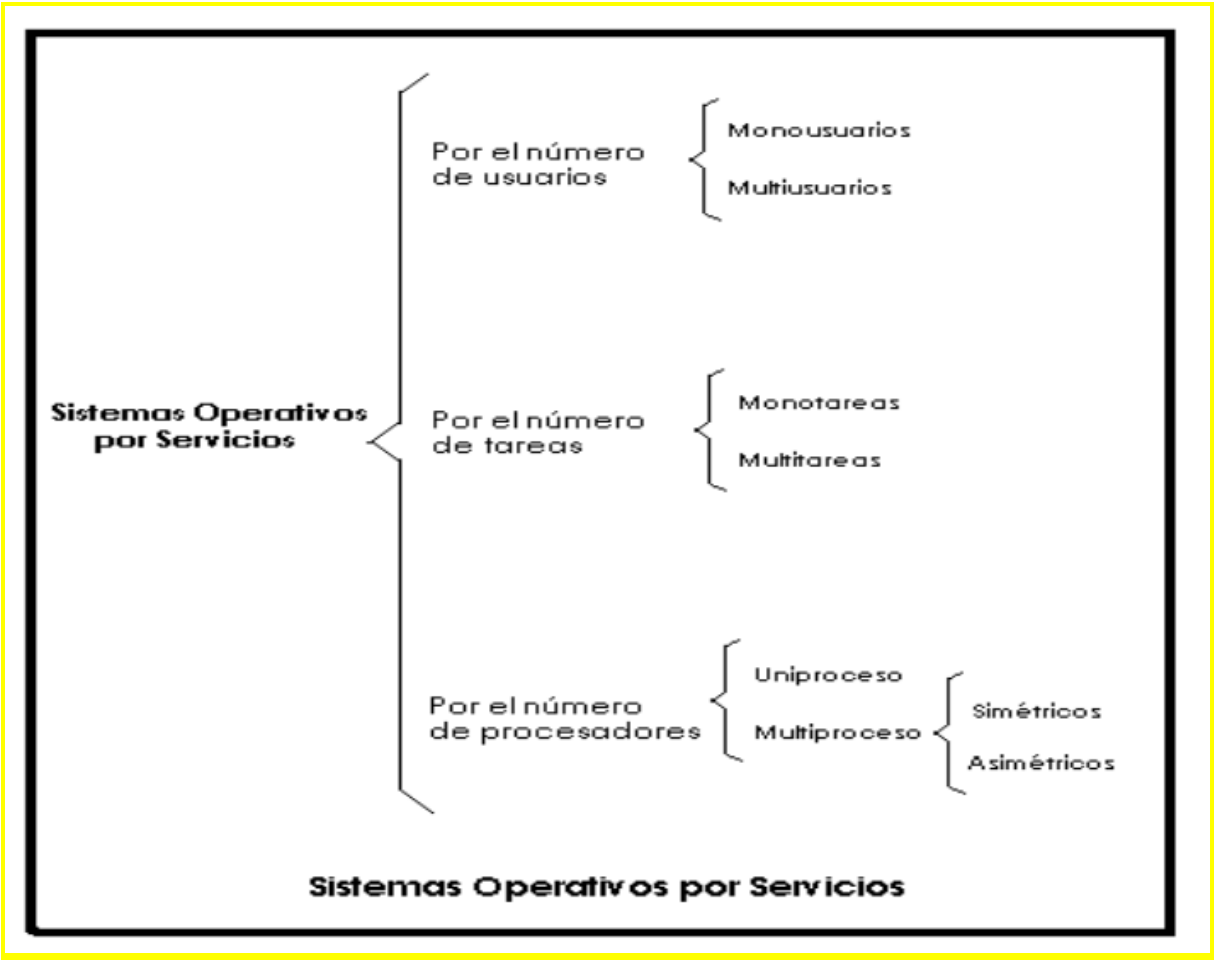
- **Microkernel**: dos capas: capa de kernel con funciones básicas(adm de cpu, memoria,comunicacion de procesos) y una capa de usuario con funciones restantes. utiliza pase de mensajes y es facilmente flexible y ampliable.

- **En capas**: Estructura definida, múltiples capas de software. , se aislan los errores en capas. Era pesado. Necesitaba mucha planificacion.

  En Windows NT se uso el TUNNELING para saltear capaz

- **microkernel modular** Kernel dividido en un Conjunto de modulos cargados dinamicamente e interconectados entre ellos y con el micro kernel(servicios basicos). aislados, no propagaban ningún error.

- **Mixtos**: igual al microkernel con módulos pero con el módulo de microkernel mas completo



Máquinas virtuales: Abstrae el hardware de una computadora, formando varios entornos de ejecución diferentes, simulando que cada entorno tiene su propio computadora.

/////// II - PROCESOS

Un proceso es un programa en un ejecución --> introducción didáctica al tema (**NO DECIR EN FINAL, ES MUCHO MÁS COMPLEJO**)

Proceso

Es una entidad dinámica compuesta por el código a ejecutar, el conjunto de los recursos necesarios para ejecutarse(memoria, periféricos, etc) y el tiempo

Un proceso va a tener cargado en memoria, 4 segmentos, el segmento de código(el programa en si), el segmento de datos, el heap(área de variables “variables”) y el stack(pila de procesos) .y por último los atributos, presentes en la PCB

Esta colección de programa, datos, pila y atributos puede llamarse imagen del proceso. Esta imagen se mantiene en memoria secundaria, pero para que el SO pueda administrar el proceso, una pequeña parte de ésta, que contiene la información a usar por el SO debe mantenerse en memoria principal.

La ejecución de un proceso sigue una ruta de ejecución (**traza**) a través de uno o más programas.

Pila: Se utiliza para registrar las llamadas a procedimientos y los parámetros pasados entre dichos procedimientos.

El *intérprete de comandos o shell* es un proceso que lee los comandos a partir de una terminal, y crea un proceso que se encarga de la ejecución del comando

PCB (Process control block):

Estructura de datos que posee los datos del proceso. Su cantidad de datos depende del SO. Al momento de nacer el proceso se crea una instancia de PCB. Cuenta con:

- Estado (obligatorio)
- Identificador de proceso (obligatorio)
- Dirección de la próxima instrucción (o contador de programa) (obligatorio)
- Registros de la CPU asociados al proceso (obligatorio)
- Información de direcciones de memoria válidas para el proceso (obligatorio)
- Y otro tipo de información

El conjunto de líneas e instrucciones (programas) a ejecutarse es una **entidad estática**. Un programa se convierte en un proceso cuando se carga en memoria un archivo ejecutable

Cola de procesos: Cola de listos, lista enlazada de pcb, cada uno tiene un puntero al siguiente. La cabecera de la cola tiene un puntero al primero y al último.

Cola de dispositivos: lista enlazada de pcb en espera de utilización de dispositivo. Cada dispositivo tiene su cola

- **Ventajas multiprocesador vs sistemas uniprocesador**

  -mayor rendimiento: mejora por número de procesadores un poco menor a N (N:nro de procesadores)

-Barato: más barato ya que comparten bus, memoria y periféricos.

-Tolera fallos

Monotarea/monoprogramación: ejecuta un proceso pasando por todo su ciclo de vida y después pasan al siguiente

Multitarea/multiprocesamiento: (simulado o real en función de si hay uno o varios procesadores, respectivamente) Consta de varios procesadores trabajando al mismo tiempo, permitiendo trabajar en mas de un proceso al mismo tiempo

Multiprogramación: mantiene varios programas en la memoria principal. los procesos se entrelazan en el tiempo (concurrencia)

En Informática, se habla de concurrencia cuando hay una existencia simultánea de varios procesos en ejecución. Ojo, concurrencia existencia simultánea no implica ejecución simultánea.

- Si pregunta qué hardware posibilita que se puedan entrelazar(o solapar) los procesos hay que decir que es el controlador, ya que es el hardware que tomará las E/s mientras el cpu sigue con la ejecución de otro proceso,

- **Si un sistema no es multiusuario, ¿tampoco es multitarea? Justifique**

  No necesariamente. Un sistema puede ser multitarea sin ser multiusuario. Aunque sólo exista un usuario en el sistema, éste puede encontrar útil lanzar varias aplicaciones concurrentes, como puede suceder en un entorno de ventanas

- **¿Si tuviéramos que volver al modelo de monoproceso y monocapa donde ejecutamos un proceso a la vez , es necesario establecer algún tipo de control?:**

  La respuesta es que sí , dado que por más que un proceso solo se ejecute a la vez , aun eso no nos libra de tener problemas de fragmentación externa o fragmentación interna. Ya que en esa época la ram poseía en su dirección base el kernel del SO , mientras que el resto le era asignada al proceso en ejecución , entonces si no controlamos eso , puede pasar que le estemos asignado memoria en exceso a un proceso que en realidad no la utiliza o bien que como le asignamos mucho a otro proceso , si llega un proceso nuevo puede ocurrir que no tengamos suficiente memoria para asignarle.

  Fragmentación Interna: Pérdida de un recurso por ser asignado a un proceso que no lo utiliza por completo.
  Fragmentación Externa: Pérdida de un recurso por no ser asignado a ningún proceso , pero que no puede asignarse dado que la cantidad del recurso, es

insuficiente.

Creaciòn de un proceso:

1. Asignar un identificador de proceso único al proceso (ID) se añade una nueva entrada a la tabla principal de procesos

2. Reservar espacio para el proceso. se reserva espacio en memoria para la imagen del proceso. (MEMORIA)

3. Inicialización del bloque de control de proceso (PCB)

4. Establecer los enlaces apropiados. se sitúa en cola de listos o listos/suspendido (ENCOLA)

5. Creación o expansión de otras estructuras de datos. por ejemplo datos de auditoría (DATOS EXTRA)

¿Cómo nace un proceso ? - creada por usuario mediante **GUI** a través de una system call

- Creada por el **mismo SO**, puede ser por estar en una lista de tareas por hacer supervisada por un demon (cron)

- Un proceso hace nacer otro proceso. **Un padre hace un system call (FORK) para llamar a un proceso hijo** pasandole parametros. El proceso pasa a WAIT y al finalizar el proceso hijo le deja los datos al padre. Crea una estructura de árbol de procesos

- Y por ultimo por planificadores

¿Cómo termina? Cuando ejecuta su última instrucción y pide al SO que lo elimine usando la system call exit() y el SO libera todos los recursos asignados a dicho proceso.

Diagrama de estados: 5 estados de ciclo de vida de un proceso:

1- **Nacimiento**: pcb creado, no cargado en memoria principal

2- **Listo o preparado**(Ready): se carga el pcb en memoria principal. Puede haber muchos procesos en este estado (Lista de procesos). Listo para ejecutarse

3- **Ejecución** (RUN): El SO lo elige para darle el uso del CPU, se ejecutan sus instrucciones.

4- **Bloqueado** (WAIT): Si dentro de la rutina se realiza una system call o interrupción se pasa al modo WAIT esperando por la ejecución de esa system call.

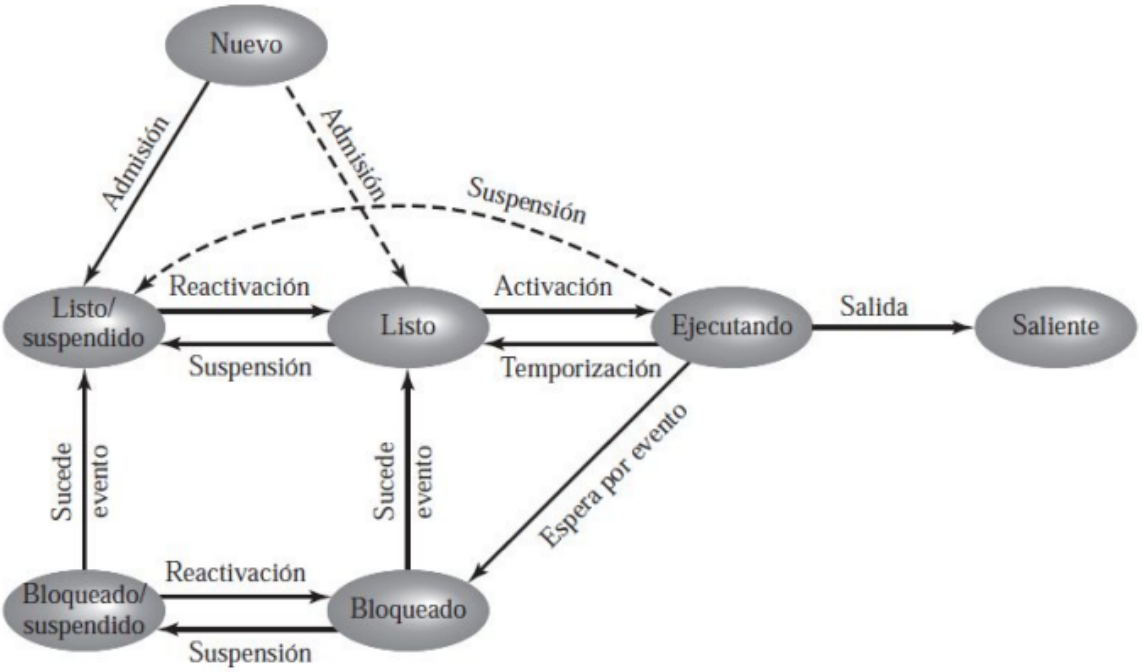
Al finalizar la system call se avisa al SO mediante una interrupción y se vuelve al estado de LISTO (para ejecutar esto necesita hacer ejecución de CPU (desalojando proceso en estado RUN llevándolo a la cola de LISTO))

5- **Finalizado**: se hace una System Call End para finalizar

Con la adición del swapping, se crea el estado de suspendido, uno de los procesos bloqueados se transfiere al disco para hacer espacio para otro proceso que no se encuentra bloqueado.

Se crean los estados

- **bloqueado/suspendido**: donde el proceso está en disco y a espera de un evento y
-
- **listo/suspendido** esta en disco pero su evento que mantenía bloqueado ya sucedió



Planificadores: son componentes del SO que planifica cómo hacer las cosas en el tiempo

El **Planificador de largo plazo** irá haciendo nacer los procesos intentando optimizar el uso de recursos. controla el grado de multiprogramación(nro de procesos en memoria). Este puede no existir, y simplemente se ponen todos los procesos en memoria para que los gestiones el planificador de corto plazo.Se ejecuta cada x minutos.

EL **planificador de corto plazo** selecciona los procesos que ya están preparados para ejecutar y asigna la Cpu a uno de ellos. El **dispatcher** es el que ejecuta el cambio. Cada 100 milisegundos aprox

El **planificador de mediano plazo** (el cual en verdad es una rutina del SO para procesos BATCH) verifica si lo que pensó el planificador de largo plazo está funcionando. alterna procesos E/s y de Cpu. responsable del intercambio.

los procesos de **E/S tienen alta prioridad** y los de **CPU baja prioridad.**

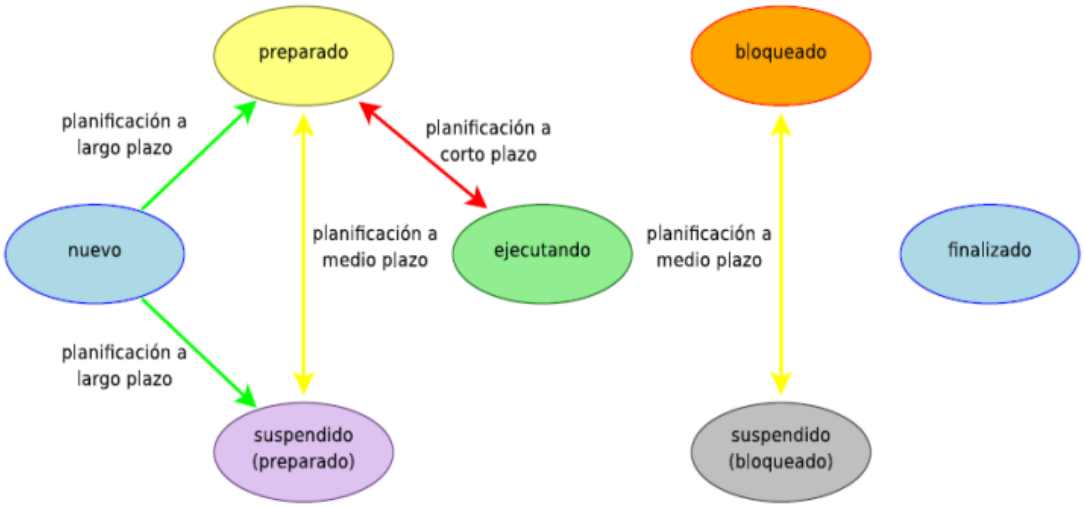
se debe hacer una mezcla de estos tipos de procesos equilibrada.

Se busca Tasa de creación de procesos ~= tasa de salida de procesos

- **¿Para qué tipo de procesamiento los SO utilizan el planificador de mediano plazo?**

  El planificador de mediano plazo es utilizado por los sistemas operativos con procesamiento por lotes para verificar si se está cumpliendo lo predicho por el planificador de largo plazo, es independiente de la administración de memoria.

Tipos de Planificacion



- largo plazo: que procesos **admitir** al sistema $\longrightarrow$ [s - min].
- medio plazo: que procesos **intercambiar** $\longrightarrow$ [ms - s].
- corto plazo: que proceso **ejecutar** $\longrightarrow$ [μs - ms].

Context Switching:

Solo en los de multiprocesamientos, ya sea multi o mono CPU. Es la conmutación de CPU de un proceso a otro, este requiere salvar el estado del proceso actual y restaurar el estado de otro proceso diferente.

Actúa el dispatcher (planifica → planificador corto plazo / ejecuta → dispatcher)

1- Se está ejecutando proceso 0 y se ejecuta una SC O una INT (**CPU:0 0->INT**)

2- primero se guarda los registros de estado de proceso 0 en su PCB, pasa a WAIT o Listo y la cpu pasa a estar controlada por el SO, para atender la interrupción o SC (**0->PCB0 CPU:SO**)

Enseguida la CPU determina qué elemento ha solicitado la interrupción y para cada caso existe un bloque de instrucciones que realiza la tarea correspondiente que es ejecutada a continuación. (se encuentra en el vector de interrupciones)

3- El planificador de corto plazo elige el proceso siguiente y el dispatcher carga en la dirección de la próxima instrucción el program counter del proceso siguiente elegido (dirección de la próxima instrucción en la que quedo el proceso) (**PCB0<-1**)

4- Se ejecuta el proceso siguiente (1) (**CPU:1**)

5- Se produce una interrupción o system call (**1->INT**)

6- Guarda estado en la pcb del proceso 1, pasa a WAIT o listo y cpu pasa a atender la interrupción. (**CPU:SO 1->PCB1**)

7- suponiendo que el planificador vuelve a elegir el proceso 0 ,carga al proceso 0 para que se ejecute (**CPU:0**)

Dispatcher: módulo que proporciona el control de la cpu a los procesos seleccionados. Implica:

-Context switching,
-cambio a modo usuario,
-salto a posición correcta dentro del programa para reiniciar dicho programa. (registro program counter)

Para lograr sincronización entre CPU y dispositivos de E/S hay dos métodos:

- Transferencia de datos por interrupciones
- Transferencia de datos por acceso directo a memoria (DMA)

Antes se realizaba por espera activa (la CPU esperaba que se realizará la E/S).

Interrupción

Las interrupciones son generadas por los dispositivos periféricos habilitando una señal del CPU (llamada IRQ del inglés "interrupt request") para solicitar atención del mismo. Por ejemplo. cuando un disco duro completa una lectura solicita atención al igual que cada vez que se presiona una tecla o se mueve el ratón.

La primera técnica que se empleó para esto fue el ***polling*****,** que consistía en que el propio procesador se encargara de sondear los dispositivos periféricos cada cierto tiempo para averiguar si tenía pendiente alguna comunicación para él. Este método presentaba el inconveniente de ser muy ineficiente, ya que el procesador consumía constantemente tiempo y recursos en realizar estas instrucciones de sondeo.

El mecanismo de interrupciones fue la solución que permitió al procesador desentenderse de esta problemática, y delegar en el dispositivo periférico la responsabilidad de comunicarse con él cuando lo necesitara. El procesador, en este caso, no sondea a ningún dispositivo, sino que queda a la espera de que estos le avisen (le "interrumpan") cuando tengan algo que comunicarle

Definición: Suceso externo al procesador que cambia el flujo normal de ejecución del procesador.

Tipos de interrupción:

Interrupción por hardware: producida por el canal(o bus). Es la interrupción por defecto. el control se transfiere inicialmente al manejador de interrupción y posteriormente al SO.

ejemplo Interrupción de reloj, terminación de E/S, Las causas que las producen son externas al procesador

Interrupción por software: producida por una system call.por ejemplo llamada de E/S

Llamada a E/S → software // Terminación de E/S → Hardware

Fallo de memoria implica una system call por lo tanto va a Wait

Excepción: interrupción por un error. El so conocerá la naturaleza del error. si es irreversible llevará el proceso a saliente

manejador o gestor de interrupción

es habitualmente un pequeño programa que realiza unas pocas tareas básicas relativas a la interrupción. (Es parte del SO (No decirlo por las dudas)) (controlador de interrupciones de primer) nivel Salva contenido de registros de cpu del proceso, determina tipo de interrupción, ejecuta ISR y restaura los registros salvados

Salva registros cpu proceso → determina tipo INT → ejecuta ISR → restaura registros

ISR: rutina de servicio de interrupción. Se obtiene a través de la tabla de vectores de interrupcion.

> El método usual de obtener la dirección de la rutina de tratamiento de la interrupción en ambos casos es a través de la tabla de vectores de interrupción, tabla que contiene las direcciones de las rutinas de tratamiento de cada interrupción.

manejador de excepciones La diferencia con el de interrupciones es que se trata en espacio de usuario, el SO solo las notifica

- **Explique que pasa cuando durante la ejecución de un proceso de usuario interactivo en una interfaz gráfica el usuario mueve el Mouse.**

  Se genera una interrupción (interrupción por hardware). Hay que contar qué pasa después, el cambio de contexto, la toma de control por parte del S. O. ver que es una interrupción que no desaloja, devolverle la CPU al Proceso.

Direct Memory Access, DMA

Es deseable permitir que los intercambios de E/S se produzcan directamente con la memoria para liberar al procesador de la tarea de E/S. En tales casos, el procesador concede a un módulo de E/S la autorización para leer o escribir de la memoria, de manera que la transferencia entre memoria y E/S puede llevarse a cabo sin implicar al procesador. Durante dicha transferencia, el módulo de E/S emite mandatos de lectura y escritura a la memoria, liberando al procesador de la responsabilidad del intercambio.

El sistema operativo sólo puede utilizar DMA si el hardware tiene un controlador de DMA, que la mayoría de los sistemas tienen. Algunas veces este controlador está integrado a los controladores de disco y otros controladores,

Múltiples Interrupciones

Las interrupciones se manejan en estricto orden secuencial.

Al tratar con múltiples interrupciones existen métodos para manejar esto

Hay dos enfoques:

- Uno es inhabilitar las interrupciones mientras se ejecuta una rutina de tratamiento, quedando pendiente hasta que finalice la rutina.
- El segundo enfoque es definir prioridades para las interrupciones y permitir que una interrupción de mayor prioridad pueda interrumpir la rutina de tratamiento de una interrupción de prioridad más baja.

- Inhabilitar las interrupciones mientras que se está procesando una interrupción (enmascaramiento)

  Enmascaramiento de interrupciones El sistema de interrupciones de un procesador dispone en general de la posibilidad de ser inhibido, es decir, impedir que las interrupciones sean atendidas por la CPU. Esta posibilidad hay que utilizarla en determinadas ocasiones en las que por ningún concepto se puede interrumpir el programa en ejecución. Además de la inhibición total del sistema, existe la posibilidad de enmascarar individualmente algunas de las líneas de interrupción utilizando un registro de máscara:

- También se definen prioridades para las interrupciones

- **Spooling vs buffering**

En Spooling, las operaciones de E / S de un trabajo se superponen con la ejecución de otro trabajo. En el almacenamiento en búfer, las operaciones de E / S de un trabajo se superponen con la ejecución del mismo trabajo. Por lo tanto, esta es la principal diferencia entre spooling y buffering..

- **¿Por qué es importante el enmascaramiento de interrupciones?**

  Por que gracias a esta funcionalidad en el tratamiento de interrupciones podemos tratar las interrupciones de forma jerarquizada y por tanto tratar de forma jerarquizada los eventos que debe procesar un Sistema Operativo
- **¿Por qué las operaciones de E/S deben ser privilegiadas?**

Porque así garantizamos que el sistema operativo sea en última instancia quién controle las operaciones de E/S,ya que éste es el administrador de recursos y el que debe garantizar la protección del sistema

- **¿Qué es la espera activa?**

  Es un mecanismo por el cual un proceso espera un evento o condición comprobando continuamente si ésta condición se cumple, con lo que consume recursos de CPU, siendo por tanto altamente ineficiente.

system call: una llamada del proceso en ejecución a un procedimiento que es parte del código del sistema operativo. Utiliza las APIS del SO

- **¿Qué diferencia hay entre las llamadas al sistema y los programas del sistema?**

  Las llamadas al sistema son los mecanismos que utilizan las aplicaciones para solicitar servicios al núcleo del sistema operativo, mientras que los programas del sistema son aplicaciones independientes que se distribuyen junto al sistema operativo y que resuelven necesidades básicas de operación o administración (ej. editores, compiladores, intérpretes de órdenes, etc.) Las llamadas al sistema son la interfaz del s.o. para las aplicaciones, mientras que los programas del sistema son la interfaz del s.o. para los usuarios.

modos de funcionamiento

Modo dual: Proporciona los medios para proteger al sistema operativo de usuarios que puedan causar errores, o de otros usuarios. Las instrucciones de máquina que pueden causar daño pasan a ser **Instrucciones privilegiadas. Solución hardware**

Modo usuario: el intento de ejecución de una instrucción privilegiada en este modo produce una excepción.

modo kernel(o núcleo): en él se pueden ejecutar tareas de cualquier instrucción(incluidas instrucciones privilegiadas).

Cambio de modo:

Si hay una interrupción pendiente, el proceso actúa de la siguiente manera:

1. Coloca el **contador de programa** en la dirección de comienzo de la rutina del programa **manejador de la interrupción**.

2. Cambia de modo usuario a modo núcleo (Con soporte hardware) de forma que el código de tratamiento de la interrupción pueda incluir instrucciones privilegiadas.

……

Procesos padre y procesos Hijo.

Fork: Está llamada crea una copia casi idéntica del proceso padre

¤ Ambos procesos, padre e hijo, continúan ejecutándose en paralelo

¤ El padre obtiene como resultado de la llamada a fork() el pid del hijo y el hijo obtiene 0

Ineficiencias del modelo fork()

¤ Se copian muchos datos que podrían compartirse

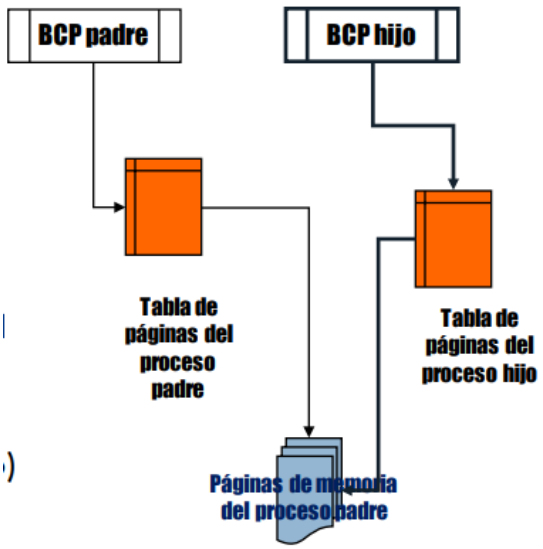
¤ Si al final se carga otra imagen, todavía es peor porque todo lo copiado se deshecha

Muchos UNIX usan COW

¤ Copy-on-Write es una técnica que retrasa o evita la copia de los datos al hacer el fork

¤ Los datos se marcan de manera que si se intentan modificar se realiza una copia para cada proceso (padre e hijo)

¤ Ahora fork() sólo copia la tabla de páginas del padre (no las páginas) y crea un nuevo BCP para el hijo



- fork Crea un nuevo proceso.

- wait Permite a un proceso padre (parent process) sincronizar su ejecución con la llamada exit de un proceso hijo

puede pasar que el padre lo termine al proceso hijo y termine todos los subprocesos de hijo.

Si el padre terminó y el hijo no, pasa al estado **Huérfano**.

- **Explicar estado Zombie**

  Cuando un proceso termina, el SO libera todos los recursos que tenía asociados excepto la estructura de control que guarda los datos del proceso (código de retorno, tiempo de ejecución...) esperando a que el proceso padre la recoja. Si un proceso termina y el proceso padre no recoge estos datos, el proceso que ha terminado se queda en el estado zombie. En un sistema que funciona correctamente no debería haber procesos en este estado durante un tiempo prolongado.

Procesos padre e hijo

Todo proceso (padre) puede lanzar un proceso hijo en cualquier momento, para ello el sistema operativo nos ofrece una llamada al sistema que se denomina fork.

La sintaxis de la llamada efectuada desde el proceso padre es:

valor=fork()

Un proceso hijo es un proceso clon del padre. Sin embargo, procesos padre e hijo no comparten memoria, son completamente independientes.

Todo proceso padre es responsable de los procesos hijos que lanza, por ello, todo proceso padre debe recoger el resultado de la ejecución de los procesos hijos para que estos finalicen adecuadamente. Para ello, el sistema operativo ofrece la llamada wait que nos permite obtener el resultado de la ejecución de uno o varios procesos hijo.

Si un proceso padre no recupera el resultado de la ejecución de su hijo, se dice que el proceso queda en estado zombi. Un proceso hijo zombi es un proceso que ha terminado su ejecución y que está pendiente de que su padre recoja el resultado de su ejecución.

Orphan (huérfanos)

Usualmente un proceso crea un proceso hijo (child) y cuando el proceso hijo termina una señal es emitida al proceso padre para que pueda hacer todo lo requerido cuando el proceso hijo es terminado. Pero hay situaciones en las que los procesos padres son matados (killed). En dicho caso el proceso hijo queda huérfano y entonces es tomado por el proceso init. Aun así el proceso cuyo padre fue matado sigue siendo llamado huérfano ya que su padre original no existe.

Procesos independientes y cooperativos

Independiente: no es influido ni influye en otros procesos. Todo está definido y no requiere interacción humana.

Ejemplo Batch. Usado para STR ya que necesita una tasa de fiabilidad muy alta sin ser influenciado por ningún otro proceso.

Cooperativos: Puede verse afectado o afectar a otros procesos que se ejecutan en el sistema. Comparte datos con otros procesos.

Ventajas: tiempo total menor ya que se solapan tiempos entre CPU y Controladora

tolera errores (excepciones) por modularidad, generando fiabilidad

Menos exigente ya que requiere menos recursos de hardware

En un principio los grandes componentes de un SO,

- adm de cpu
- adm de la memoria
- administración del sistema de archivos
- comunicaciones → Se añadió con los procesos cooperativos
- seguridad.

IPC (interprocess communication) módulo que se encarga de hacer la comunicación entre procesos. Ipc facilita dos SC : **Send()** y **Receive()**

Existen dos modelos de **comunicación procesos cooperativos**

- **Modelo de memoria compartida**: más sencillo, cuando nace un proceso A se le asigna un espacio de memoria, luego este proceso a través de una SC va a hacer nacer un conjunto más de procesos hijos, y le va a permitir a sus hijos leer y escribir en un espacio de memoria del proceso padre sin intervención de SO.

Se establece una región compartida de memoria entre procesos, los cuales podran leer y escribir sin intervención del sistema operativo. se hace una única system call para establecer la región compartida. No se pierde tiempo con Context Switching, la desventaja es que el programador debe verificar que el proceso destino pudo leer el dato.

- **Pase de mensajes**: Proporciona un mecanismo que permite a los procesos comunicarse sin compartir espacio de memoria. provee al menos dos operaciones, Send() y receive() . Debe existir un enlace de comunicaciones entre los procesos, puede ser físico o lógico. En este si interviene el SO. Es necesario para sistemas Mono Cpu.

Funciona mediante “BUZONES” los cuales son direcciones de memoria en el espacio del SO donde un proceso pide al SO que cree el buzón e informa a los otros procesos que puede leer/escribir en dicho “buzón” para comunicarse entre procesos. NO comparten el mismo espacio de direcciones

Comunicación directa: Implica que hay dos Cpu funcionando simultáneamente (una que envíe y una que reciba) (no sirve para mono cpu).

Los enlaces se establecen automáticamente, todo enlace es asociado con un par de procesos, este enlace será normalmente unidireccional aunque puede ser bidireccional

Comunicación indirecta: Funciona mediante “BUZONES” los cuales son direcciones de memoria en el espacio del SO., cada buzón está identificado y el creador del buzón lo hace a través de una SC que pide este recurso, y además establece que otros procesos pueden acceder y con qué permisos.

Si se establece un buzón en el que solo un proceso puede escribir y todos los demás leer es una **comunicación unidireccional**.

Si en el buzón todos pueden leer y escribir es **bidireccional**.

Cada proceso puede acceder a múltiples buzones. Las **operaciones** de este tipo de comunicación son

Crear buzón

mandar/recibir mensajes (send(A,message) / receive(A,message))

destruir mensajes

Como saber quien recibe el mensaje ?

Si p1, p2 y p3 comparten buzón, y p1 send() y p2, p3 receive() à quien recibe el mensaje?

Soluciones para evitar deadlocks:

El sistema seleccione arbitrariamente y se informe al que lo recibe

O encerrar en una sesión crítica en que solo una puede ejecutar la SC receive()

Sincronización o coordinación

La escritura de mensajes puede ser síncrona o asíncrona

> **Síncrona:** es bloqueante, se bloquea el proceso que envía el mensaje hasta que es recibido y el destino es bloqueado hasta que recibe un mensaje. (similar al uso de un walkie tolkie)
>
> **Asincrónica**: no son bloqueantes, el que envía no se bloquea al mandar el mensaje no espera a que se reciba. Lo mismo para el que recibe.

Buffering: los mensajes intercambiados residen en una cola tempotal. Se pueden implementar de 3 maneras:

Buffer 0: sin posibilidad de acumular mensajes, se debe escuchar los mensajes en el momento

Buffer limitado: lista con un número limitado

"Buffer Infinito": buffer circular que consta de Proceso productor y proceso consumidor. El puntero del proceso consumidor debe estar siempre detrás del productor. Posible pérdida de datos

Nombra el LPC pero lo explica así nomas, no creo que entre

Modelo cliente servidor

No es únicamente entre dos pcs sino dentro del mismo proceso. Es así en Windows

3 metodos de comunicacion:

Sockets:

es el punto de conexión de una comunicación determinada, el hardware mapeara el socket físico a una dirección de memoria. Forma de comunicación de bajo nivel. solo permite enviar y recibir flujo no estructurado de bytes.

- **Socket físico** es la capa física del socket
- **Socket lógico** es una dirección de memoria y designada por un puerto, por ejemplo el puerto 80.

Un tipo de socket es un **websocket** el cual tiene una dirección IP y un puerto

Remote process call(RPC): Soluciona de mas alto nivel que el de Socket. A partir de puertos es posible realizar una llamada remota a un proceso. Abstrae los mecanismos de llamada. Debe utilizar esquema de comunicación basado en buzones.

Stubs del lado del cliente , se trata de un encabezado de la rutina pero sin código, permite definir los parámetros

Stubs Del lado del servidor, recibe el mensaje, trata los parámetros y lo ejecuta del lado del servidor

Ejecución de una RPC

- El usuario hace una SC para mandar un mensaje al proceso X (cliente)
- Kernel envía mensaje al matchmaker para encontrar el puerto (cliente)
- Matchmaker recibe el mensaje (servidor)
- Mathmaker envía el pedido al cliente con puerto P (Servidor)
- Kernel establece el puerto p para el RPC message (cliente)

- Kernel envía el RPC (cliente)
- Demon del puerto P recibe el mensaje (servidor)
- Demon procesa el request y envía el output (servidor)
- Kernel recibe el output y se lo pasa al usuario. (cliente)

Remote method invocation: Es un mecanismo similar a RPC pero para objetos, usado en java para ejecutar procesos en objetos externos.

Hilos (Thread)

habíamos dicho que un proceso es una entidad dinámica compuesta por los datos y código a ejecutar, el conjunto de los recursos necesarios para ejecutarse(memoria, periféricos, etc) y el tiempo

Ahora se Divide a cada proceso en múltiples entidades dinámicas pequeñas llamadas hilos, y cada hilo contará con un TCB(thread control block) que contará con los datos mínimos para ejecución,(Id , contador, registros y pila),

Los hilos serán la unidad básica de utilización de la CPU, estas se ejecutarán en paralelo (en caso de ser multi cpu) o competirán entre ellos por el uso de la cpu (en caso de mono cpu)

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

Por lo tanto la visión de un proceso cambio a la de “Contenedor de recursos y múltiples entidades dinámicas llamadas hilos”

Una **rutina NO ES LO MISMO que un thread** (Hilo). ya que el hilo de ejecución en el caso de la rutina es único y es fijo

es decir que si hago un call (rutina A) y después un call (B), siempre se terminará de ejecutar la rutina A antes que la B.

En los hilos no hay secuencialidad obligatoria.

- **¿Cuáles son las ventajas fundamentales que aportan los hilos frente a los procesos?**

La ventaja principal es poder dotar de concurrencia interna a una aplicación, con un coste reducido. Varios hilos dentro de un proceso pesado comparten el mismo espacio de memoria (código y datos), así como los recursos que la aplicación tenga asignados.

Por tanto, la comunicación entre hilos es más sencilla y además al ser entidades que consumen menos recursos que los procesos el cambio de contexto entre hilos es menos costoso.

- **¿Qué diferencia hay entre un hilo y un proceso hijo?**

  Un hilo es parte de un proceso, un proceso hijo es otro proceso, que puede competir con el proceso hijo por los recursos, o puede estar bloqueado mientras se ejecuta el proceso hijo, pero son dos procesos.

  Para ser claro, un hilo es parte del proceso, un proceso hijo es otro proceso.

- **¿Puede haber bloqueo mutuo entre distintos hilos de un mismo proceso?, explique y de ser posible ejemplifique la respuesta.**

  SI, puede haberlos, simplemente cada hilo retiene un recurso y espera por otro.

Planificación con Hebras

Ventajas

- Mayor capacidad de respuesta: si una hebra se bloquea, las demás pueden seguir ejecutándose.
- Puede haber varias hebras compartiendo los mismos recursos (memoria, ficheros, etc).
- Menos costoso que crear procesos, el cambio de contexto también es más ligero.
- Pueden aprovechar arquitecturas multiprocesador.

- La planificación se separa a dos niveles: procesos; hebras.
- Un planificador de procesos elige el proceso. Después un planificador de hebra escoge la hebra.
- No existe apropiación entre hebras. Si la hebra agota el quantum del proceso, se salta a otro proceso. Cuando vuelva, seguirá con la misma hebra
- Si la hebra no agota el quantum, el planificador de hebra puede saltar a otra hebra del mismo proceso.

IMPORTANTE: Cuando se refiere a dos niveles se refiere a los ámbitos de contienda, el nivel de hebras interno del proceso es el ámbito de contienda del proceso(PCS) y el ámbito de contienda donde se planificará por la CPU es el ámbito de contienda del sistema (SCS)

Alguna de las ventajas:

● Lleva mucho menos tiempo crear un nuevo hilo en un proceso existente que crear un proceso totalmente nuevo.

● Lleva menos tiempo finalizar un hilo que un proceso.

● Lleva menos tiempo cambiar entre dos hilos dentro del mismo proceso.

● Los hilos mejoran la eficiencia de la comunicación entre diferentes programas que están ejecutando.

FUNCIONALIDADES DE LOS HILOS

Los hilos, al igual que los procesos, tienen estados de ejecución y se pueden sincronizar entre ellos Estados de los hilos. Igual que con los procesos, los principales estados de los hilos son: Ejecutando, Listo y Bloqueado.

Context switching procesos vs hilos

En muchos de los sistemas operativos que dan facilidades a los hilos, es más rápido cambiar de un hilo a otro dentro del mismo proceso, que cambiar de un proceso a otro. Este fenómeno se debe a que los hilos comparten datos y espacios de direcciones, mientras que los procesos, al ser independientes, no lo hacen. Al cambiar de un proceso a otro el sistema operativo (mediante el *dispatcher*) genera lo que se conoce como *overhead*, que es tiempo desperdiciado por el procesador para realizar un cambio de contexto (*context switch*), en este caso pasar del estado de ejecución (*running*) al estado de espera (*waiting*) y colocar el nuevo proceso en ejecución. En los hilos, como pertenecen a un mismo proceso, al realizar un cambio de hilo el tiempo perdido es casi despreciable.

La principal distinción entre un cambio de hilo y un cambio de proceso es que durante un cambio de hilo, el espacio de memoria virtual permanece igual, mientras que no lo hace durante un cambio de proceso. Ambos tipos implican entregar el control al kernel del sistema operativo para realizar el cambio de contexto. El proceso de entrar y salir del kernel del sistema operativo junto con el costo de cambiar los registros es el mayor costo fijo de realizar un cambio de contexto.

Un costo más difuso es que un cambio de contexto se mete con los mecanismos de almacenamiento en caché de los procesadores. Básicamente, cuando cambia de contexto, todas las direcciones de memoria que el procesador "recuerda" en su caché se vuelven efectivamente inútiles. La única gran distinción aquí es que cuando cambia los espacios de memoria virtual, el búfer de búsqueda de traducción

(TLB) del procesador o equivalente se vacía, lo que hace que los accesos a la memoria sean mucho más costosos durante un tiempo. Esto no sucede durante un cambio de hilo.

Proceso hijo no comparte memoria, hilo si>>>

Diferencia proceso usando memoria compartida y un hilo es que el proceso para obtener la informacion de memoria compartida debe ejecutar system calls para acceder y escribir en dicho espacio, en cambio el hilo al tener exactamente el mismo espacio de memoria que los demas hilos no necesita al hacer una system call, simplemente escribe en su espacio de memoria.

- **Distinguir un proceso cooperativo de un conjunto de hilos de ejecución.** (antes los procesos cooperan entre sí , mientras que ahora un proceso puede dividirse en múltiples hilos que cooperan para una misma tarea)

- **que diferencia existe entre una rutina y un hilo? -->** Esta pregunta iba relacionada al hecho de que una rutina de código se ejecuta de forma secuencial linea tras linea , mientras un hilo puede correr de forma asincrónica , ya que como el mismo contiene codigo , me permite ejecutar múltiples partes de un mismo proceso de forma simultánea.

--es necesario sincronizar las actividades de los hilos para que no interfieran entre ellos o corrompan estructuras de datos.

MULTIHILO

Multihilo se refiere a la capacidad de un sistema operativo de dar soporte a múltiples hilos de ejecución en un solo proceso.

ULT: Threads a nivel de usuario, KLT: Threads a nivel de Kernel y LWP:Lightweight Process

asocia ULT y KLT

Modelo Mx1 (Many to one)

Por otro lado Unix creó la librería (**PThread**) que implemente el muchos a 1 (un thread kernel y mucho threads usuario). Este gestion se hace en el espacio de usuario

El modelo asigna múltiples hilos de usuario a un hilo del kernel. Este caso se corresponde a los hilos implementados a nivel de usuario, ya que el sistema solo reconoce un hilo de control para el proceso. Tiene como inconveniente que si un hilo se bloquea, todo el proceso se bloquea. También, dado que solo un hilo puede acceder al kernel cada vez, no podrán ejecutarse varios hilos en paralelo en múltiples CPUs.

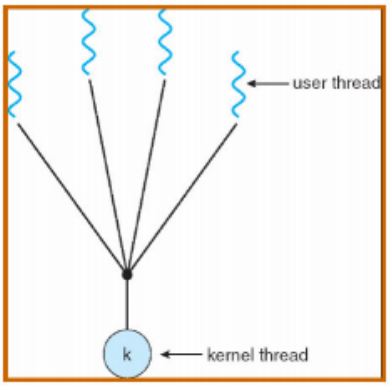
Hilos de nivel de usuario. En un entorno ULT puro, la aplicación gestiona todo el trabajo

de los hilos y el núcleo no es consciente de la existencia de los mismos. La biblioteca de hilos.

El uso de ULT en lugar de KLT, presenta las siguientes ventajas:

● El cambio de hilo no requiere privilegios de modo núcleo porque todas las estructuras de datos de gestión de hilos están en el espacio de direcciones de usuario de un solo proceso. Por consiguiente, el proceso no cambia a modo núcleo para realizar la gestión de hilos. Esto ahorra la sobrecarga de dos cambios de modo (usuario a núcleo; núcleo a usuario).

● La planificación puede especificarse por parte de la aplicación.



• Modelo 1x1 (one to one)

hoy se utiliza el **sistema 1 a 1**, cada vez que nace un thread a nivel de usuario, nace un thread a nivel de SO que va a dialogar con el.

Es eficiente y rapido (por los context switching) pero tiene la desventaja que consume muchos recursos.

Usa unicamente planificación en ámbito de contienda de sistema (scs)

El modelo asigna cada hilo de usuario a un hilo del kernel. Proporciona una mayor concurrencia que el modelo anterior, permitiendo que se ejecute otro hilo si uno se bloqueó. Tiene como inconveniente que cada vez que se crea un hilo a nivel de usuario, se crea un hilo a nivel del kernel, y la cantidad de hilos a nivel del kernel están restringidos en la mayoría de los sistemas

Hilos a nivel de núcleo.

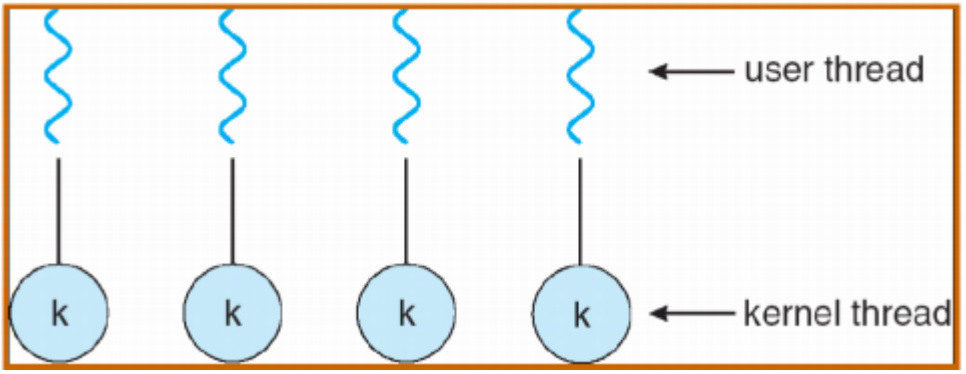
En un entorno KLT puro, el núcleo gestiona todo el trabajo de gestión de hilos. No hay código de gestión de hilos en la aplicación, solamente una interfaz de programación de aplicación (API) para acceder a las utilidades de hilos del núcleo.

Este enfoque resuelve los dos principales inconvenientes del enfoque

ULT. Primero, el núcleo puede planificar simultáneamente múltiples hilos de un solo

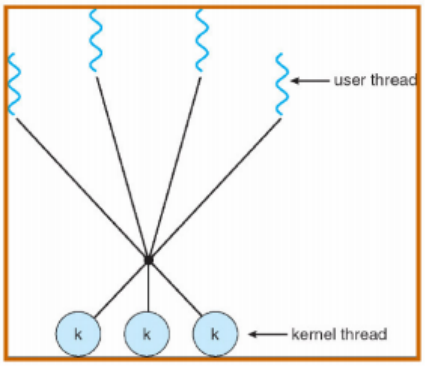
proceso en múltiples procesadores. Segundo, si se bloquea un hilo de un proceso, el núcleo puede planificar otro hilo del mismo proceso. Otra ventaja del enfoque KLT es que las rutinas del núcleo pueden ser en sí mismas multihilo.

La principal desventaja del enfoque KLT en comparación con el enfoque ULT es que la transferencia de control de un hilo a otro del mismo proceso requiere un cambio de modo al núcleo.



• Modelo MxN (many to many)

El modelo multiplexa muchos hilos de usuario sobre un número menor o igual de hilos del kernel. Cada proceso tiene asignado un conjunto de hilos de kernel, independientemente de la cantidad de hilos de usuario que haya creado. No posee ninguno de los inconvenientes de los dos modelos anteriores, ya que saca lo mejor de cada uno. El usuario puede crear tantos hilos como necesite y los hilos de kernel pueden ejecutar en paralelo. Asimismo, cuando un hilo se bloquea, el kernel puede planificar otro hilo para su ejecución. Entonces, el planificador a nivel de usuario asigna los hilos de usuario a los hilos de kernel, y el planificador a nivel de kernel asigna los hilos de kernel a los procesadores.



Para los modelos de muchos a uno y de muchos a muchos se utiliza una estructura intermedio llamada LWP(Light weight process) para coordinar el numero de herbas kernel dinamicamente

Suspender un proceso i**mplica expulsar el espacio de direcciones** de un proceso de memoria principal para dejar hueco a otro espacio de direcciones de otro proceso.

Ya que todos los hilos de un proceso comparten el mismo espacio de direcciones, **todos los hilos se suspenden al mismo tiempo**. De forma similar, la finalización de un proceso finaliza todos los hilos de ese proceso.

● **Procesamiento distribuido**. Gestión de múltiples procesos que ejecutan sobre múltiples sistemas de cómputo distribuidos

Planificador de uso de la CPU

La CPU se usa por ráfagas, El ciclo de ejecución de un proceso se compone por rafagas de CPU y rafagas de E/S/

Los procesos orientados a CPU la ráfaga va a ser larga y en las de E/S va a ser corta. El objetivo de la planificación es hacer mas productivo el sistema mediante la conmutación de la CPU entre procesos.

Las hebras a nivel de kernel son las son planificadas.

Existen múltiples criterios para planificar el uso del CPU, esta planificación la hace el planificador de corto plazo, llevando los procesos de la cola de LISTOS a el estado RUN.

Desalojo: el diagrama de 5 estado estudiado es un SISTEMA con desalojo ya que tiene la posibilidad de quitar el proceso el cual está ejecutándose para poder decidir que otro proceso se ejecutará, este proceso quitado vuelve a LISTOS

Sin desalojo: En los SO sin desalojo se agrega el estado de DESPLAZADO, el cual va a ser destinado a los procesos que se estaban ejecutando y por alguna razón (interrupción/excepción) debe dejar de ejecutarse. Los procesos en este estado tienen mayor prioridad que los procesos en LISTOS

El proceso se mantiene en CPU hasta que pase del modo RUN a WAIT o finalice el proceso

TODOS LOS SISTEMAS ACTUALES SON CON DESALOJO (Prehentive)

- **Explique porque existen distintos algoritmos de planificación de CPU**

Existen diversos algoritmos de planificación de CPU, porque un algoritmo no es más que una forma de resolver un problema particular. Al haber diferentes tipos de problemas, son necesarios diferentes tipos de algoritmos.

- **Indique 3 algoritmos de planificación de CPU NO apropiativos**

  FCFS, SJF,

Existen múltiples algoritmos porque no existe una solución única, todo depende del objetivo de uso.

Tipos de planificación

En los sistemas p**or lotes suele primar el *rendimiento del sistema***, mientras que en los s**istemas interactivos es preferible minimizar, por ejemplo, el tiempo de espera.**

Planificacion uniprocesador:

FCFS (O FIFO): se asemeja a la estructura de COLA, el primero que entra el el primero que sale. Algoritmo sin desalojo. Nace un proceso, y el recurso CPU lo mantiene hasta que finalice o cambie a estado de DESPLAZADO por una interrupción. BATCH

No funciona ya que se genera el efecto "locomotora", es decir, que si el uso del CPU lo toma un proceso que requiere mucho tiempo (orientado a CPU) demora otros procesos que requerirían menos tiempo

Algoritmo de prioridad: a cada uno de los procesos se le asignará un nivel de importancia o prioridad y se le dará el recurso de la CPU al proceso que tenga mayor prioridad. El planificador de corto plazo al terminar un proceso buscará siempre el de mayor prioridad posible.

La prioridad es asignada manualmente por el operador central de consola por el usuario.

No funciona ya que se da problemas al tener que atender dos procesos de igual prioridad y puede causar deadlock, para esto se usa un criterio de desempate en el que compara según otro valor, normalmente se hace por el processId ya que este es único.

También posee el problema de starvation, en el que existan procesos que nunca son atendidos por el hecho de que siempre existan procesos con mayor prioridad. Para evitar este problema se soluciona de varias formas, una es poniendo un límite de "veces no atendidas" , no es muy eficiente.

Otra forma es la de prioridad relativa, dando valores en números a las prioridades, siendo las de mayor prioridad un número mayor, y por cada vez que no es atendido cada proceso aumentará una unidad en la prioridad.

BATCH Puede ser apropiativo o no

- **¿Por qué no sería conveniente utilizar un algoritmo de prioridad puro en un sistema operativo interactivo, como widows 10 desktop?**

Porque al ser un sistema operativo interactivo, si tuviese una operacion en background existe la posibilidad de tildar el sistema. Es más conveniente utilizar un Round Robin. Prioridad pura es para procesamiento en lotes (batch)

SJF: trabajo más corto primero (en verdad se trata primero a los que requieren una ráfaga de uso de CPU de menos tiempo). su tasa de eficiencia es alta, se da mayor prioridad a los procesos orientados a E/S. Es no apropiativo.

El problema mayor es que se hace un cálculo en base a supuestos usos futuros de CPU , para esto se usa una "estimación", una es usando el método "NAIF", el cual usa el valor anterior de uso y le suma una cantidad.

Otra es usando promedio móvil, el cual toma un promedio de las últimas ráfagas de uso de cpu (eliminando aleatoriedades de un uso inmediato)

Y por último es usando un promedio móvil ponderado usando una fórmula de ponderación para los promedios. BATCH

También existe el **SJF apropiativo**, llamado también ráfaga de tiempo RESTANTE más corta, en la cual se toma en cuenta la cantidad de tiempo restante que le queda a los procesos, a medida que se está ejecutando un proceso su tiempo restante disminuye y puede darse que se desaloje para darle lugar a un proceso con menor tiempo restante que surja. Se le dice que es de prioridad dinámica. Puede usarse en interactivo

Todos estos mecanismos hasta ahora NO sirven para sistemas interactivos ya que no garantizan tiempos de respuesta, solo sirve para procesos batch.

Round robin: algoritmo que genera una cola FIFO hasta agotar su QUANTUM de tiempo para volver al final de la cola de LISTOS (NO VA AL WAIT), si es desalojado por una interrupción externa va al estado DESPLAZADO. INTERACTIVO

Uno de los problemas es que es un algoritmo injusto, es decir, ya que se le da a todos los procesos el mismo quantum sin tener en cuenta las necesidades de cada proceso.

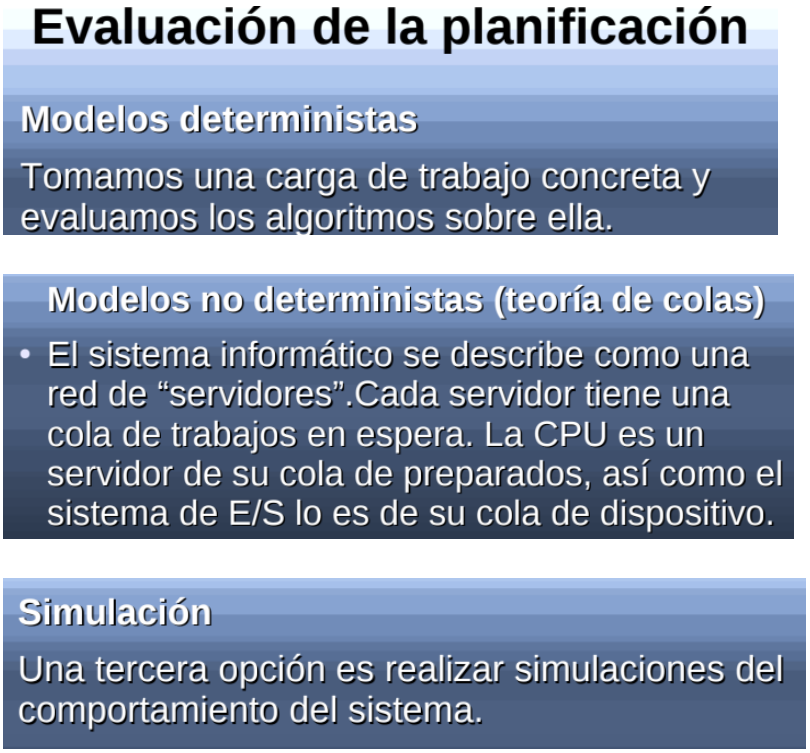
Para solucionar esto se usó un **Round robin "JUSTO"** que posee dos colas, una cola llamada activa y una cola pasiva. Al arrancar se le da un quantum largo a todos los procesos,

En la cola activa se carga los procesos y por cada interrupción se vuelca el proceso al final de la cola activa restándole el tiempo utilizado, al momento de agotar el quantum de tiempo el proceso es volcado al final de la cola pasiva, y se continua con los procesos presentes en la cola activa… y así hasta llenar la cola pasiva, y después se dan vuelta las colas (la pasiva pasa a ser la activa y viceversa) y se comienza de vuelta el proceso con el mismo numero de quamtun. Este caso solo sirve para E/S

Round robin de múltiples colas de LISTO con distinto quamtun (con retroalimentación): la prioridad de las colas irán de menor Q a mayor Q, se carga los procesos en las colas y cuando un proceso supera el quantum de la cola en la que se encuentra, se volcara a la cola siguiente con mayor Q.

¿Se puede usar Round Robin en Batch?

Si, y se usa,



Mecanismos de Sistemas Multi Cpu

Los procesadores comparten el bus y el reloj (clock). Si comparten memoria y periféricos son fuertemente acoplados. El multiprocesamiento puede ser simétrico o asimétrico. Las ventajas de los sistemas paralelos es que mejoran el throughput (resolución de mayor cantidad de procesos en un momento dado)

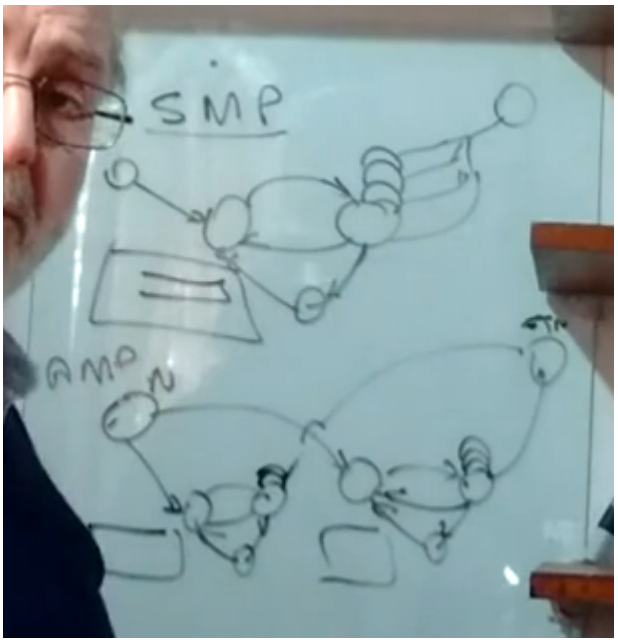
Los sistemas distribuidos: se distribuye el trabajo entre varios procesadores, pero estos no comparten memoria ni reloj.

Planificación multiprocesador

SMP(Symmetric multiple processing): Cada CPU se planifica a si misma.Comparten cola de listo. Realiza todas las tareas correspondientes al SO. Todos los procesadores son iguales, no existe relación esclavo maestro entre los procesadores.

AMP(Asymmetric multiple processing):Cada CPU se le asigna una tarea específica.No comparten las colas de listo. Usualmente puede tenes un AMP compuesta por varias SMP (Apu, celulares, etc)

Un procesador maestro controla el sistema y el resto de los procesadores esperan que el maestro les de instrucciones. Relación esclavo-maestro.



Sistema de cluster:Están formados por dos o más sistemas individuales acoplados

Criterio de equilibrio de carga y afinidad, dos **características en conflicto**

Equilibrio de carga: si en un momento una CPU está más cargada que otra, el proceso va a la cpu menos cargada. Trata de mantener la carga equilibrada entre procesadores.

Migración comandada: Se tiene una tarea que comprobara las cargas

Migración solicitada: procesador extrae de otro procesador una tarea en espera.

Afinidad: procesos tienen afinidad hacia el procesador en el cual se estan ejecutando actualmente, si un hilo proceso se está ejecutando en una CPU y termina, se deja cargado su caché con datos de este proceso, dado que tiene datos de este proceso se tendrá “afinidad” por los hilos del mismo proceso. Esta afinidad puede ser estricta o blanda, si es estricta solo atiende los hilos de un proceso determinado.

Estas dos características entran en conflicto.

Se nombra el Multithreading: Habla de proporcionar procesadores lógicos sobre procesadores físicos, cada uno tiene sus propios registros de estado y uso general y es responsable de la gestión de sus interrupciones.

Se usa para ejecutar hilos.

Sincronización de procesos

Habiendo más de un proceso en ejecución concurrente, uno de los procesos puede llevarse los recursos pertenecientes a otros procesos, es decir, más de un procesos utiliza un mismo recurso compartido al mismo tiempo

Sincronización: Se utilizan mecanismos para asegurar la ejecución ordenada de procesos cooperativos que comparten un espacio de direcciones lógicos.

- **Explique la necesidad de sincronizar procesos**

  Se necesita sincronizar procesos cuando éstos son cooperativos (es decir, se ven afectados por la ejecución de otros procesos, como también ellos afectan a otros. Cualquier proceso que comparte datos con otros, es cooperativo).

  Como dos procesos no pueden ejecutar su sección crítica (área donde comparten las variables, datos, etc) al mismo tiempo, se debe sincronizar la entrada a la misma. Cualquier solución a este problema deberá satisfacer la exclusión mutua, progreso y espera limitada.

Sección crítica (o región crítica): Es aquella parte del código del programa en la cual se accede y se puede modificar un recurso compartido.

Para utilizar el código presente en esta sección,no debe haber otro proceso ejecutando esa sección que afecte al mismo recurso.

El problema de la sección crítica consiste en diseñar un protocolo que los procesos puedan utilizar para cooperar de esta forma

-**Sección de entrada**: sección que implementa la solicitud de permiso de entrada a la sección crítica.

-**Sección restante:** es el código posterior

Toda solución propuesta para la sección crítica **debe cumplir**:

- Principio de **exclusión mutua:** Si un proceso se está ejecutando en su sección crítica los demás no deben poder.
- **Progreso**: en la toma de decisión de qué proceso hace uso del recurso, solo pueden participar los procesos en cola y no estar ejecutando su sección restante.
- **Espera limitada**: Límite que se permite a otros procesos ingresar a la seccion critica y otro proceso hizo la solicitud para ingresar, para evitar inanición

Las dos últimas principios son requeridas si la planificación no es FIFO.

métodos de gestión de secciones críticas

Kernel apropiativo: permite que el proceso sea desalojado mientras se está ejecutando en modo kernel. mejor capacidad de respuesta, adecuado para STR

Kernel no apropiativo:No permite que el proceso en modo kernel sea desalojado cuando se ejecuta en modo kernel. libre de condición de carrera en datos de kernel

Problemas posibles

Condición de carrera

sucede cuando múltiples procesos o hilos leen y escriben datos haciendo que el resultado final dependa del orden de ejecución de las instrucciones en los múltiples procesos.

Interbloqueo o deadlock:

Bloqueo permanente de un conjunto de procesos que o bien compiten por recursos del sistema o se comunican entre sí. Puede darse por recursos fisicos o logicos

Sea dos procesos que para uno para realizar su tarea requiere dos recursos, el proceso 1 se apropia del recurso 1 y el proceso 2 se apropia del recurso 2, y ninguno de los dos procesos libera sus recursos asignados ya que para eso necesita el recurso faltante en posesión del otro proceso.

estado seguro cuando al asignar recursos a cada procesos no produce interbloqueo, es decir se tiene una secuencia segura.

Estado inseguro: Cuando al asignar recursos existen secuencias no seguras, puede llevar a interbloqueo

Grafo de asignación de recursos: define el interbloqueo de forma precisa mediante un grafo dirigido.

Condiciones de deadlock:

- **Exclusión mutua:** el recurso o al menos uno de los recursos debe ser un recurso no compartible, es decir, solo un proceso puede usarlo a la vez.
- **Retención y espera**: Al haberse dado el recurso deben mantener, la política del SO debe permitir mantener recursos otorgados. y esperando otros recursos retenidos por otro proceso.
- **No apropiación**: el SO no debe atender solicitudes de expropiación.
- **Cadena circular**: ciclo circular entre la asignación de recursos y procesos

Tratamientos de deadlock

1- **Política de evitar que pase**

Prevenir: se debe prevenir que no se cumplan las condiciones de interbloqueo:

- fijarse un tiempo más que un proceso retiene un recurso (condición 2) quitándole todos los recursos al cumplirse el tiempo. (requiere que el sistema sea transaccional, es decir, que pueda retrotraerse a una situación anterior)
- Permitir expropiación de recursos por partes de procesos de mayor prioridad (condición 3)
- Cada vez que se solicite un recurso se debe anotar, y se debe usar un algoritmo de detección de grafos (condición 4) (no usada)

Evitar

- Para mantener los recursos en un estado seguro, se debe pedir a los procesos que digan qué recursos van a usar desde el momento de nacer, y asignarlos o reservarlos (algoritmo de blanqueo)

2- **Política de detección y corrección**

Dejar que pase el deadlock, detectar cuales quedaron en wait y hacer el grafo de asignación de recursos(ir matando procesos). Políticas:

- Matar a todos y hacer hacerlos de nuevo
- Ir matando de a uno y ver si arrancan
- Matar según algoritmos

3- Política de No hacer nada

Windows hace esto

Técnicas de sincronización

- Basadas en memoria compartida
  - Inhibición de Interrupciones
  - Espera activa
  - Semáforos
  - Regiones críticas
  - Monitores
- Basadas en el paso de mensajes
  - Canales
  - Buzones

Soluciones implementadas

Son soluciones para la sincronización de procesos, no únicamente para resolver el problema de sección crítica.

Desactivar interrupciones antes de entrar a la región crítica y activarlas al salir. (cerrojo)

Mientras las interrupciones estén desactivadas no puede ocurrir una interrupción de reloj entonces no se alternará otro proceso, pero esto puede afectar la seguridad del sistema. Es poco atractivo que los procesos del usuario tengan el poder de desactivar las interrupciones, qué sucedería si un proceso desactiva las interrupciones y nunca las activa

Algoritmo de Piterson: Por software mediante flags, actualmente en desuso ya que no tenía en cuenta múltiples hilos de ejecucion

Resolución por hardware: A nivel de procesador se implementaron instrucciones que realizan en un solo ciclo de instrucción un conjunto de operaciones. (Test y SET)

Semáforo:

Realizado por Dijkstra. Es un tipo de variable que tiene dominio de los enteros y tiene 3 operadores. (NO ES UNA VARIABLE ENTERA) tiene también asociada una lista de procesos, en la que se incluyen todos los procesos que se encuentran a la espera de acceder al mismo.

inicialización(p):Se asigna un primer valor. cantidad de instancias del mismo recurso.

wait(p) o test: Se pone en la sección de entrada. Si la variable semáforo es menor o igual a 0, espera. Si no si es 1 o más de 1 hay por lo una instancia de ese recurso libre, se resta en 1 e ingresara a la sección crítica.

signal(p) o set: Se pone en la sección de salida. Se ejecuta cuando un proceso sale de la sección sumará 1 a la variable semáforo. y permite a un proceso de lista de espera entrar a la sección

Un semáforo En función del rango de valores que admiten, los semáforos se pueden

clasificar en:

* Semáforos binarios: Son aquellos que solo pueden tomar los valores 0 y 1.

* Semáforos contador: Son aquellos que pueden tomar cualquier valor

Ventajas de los semáforos:
Resuelven todos los problemas que presenta la concurrencia.
Tienen implementación muy eficientes.

Desventajas semáforo

- son unos componentes que por su bajo nivel de abstracción resultan muy peligrosos de manejar y frecuentemente son causa de muchos errores.
- funciona bien cuando son distintas instancias del mismo código, pero cuando son distintos códigos diferentes los que usan el semáforo NO ES CONFIABLE

Wait (S) {
While S <= 0
; //none
S--;
}
	Signal (S) {
	S ++;
	}

- **¿Por qué utilizaría un semáforo con conteo en vez de uno binario?**

Porque uno puede tener recursos con múltiples instancias entonces si se tuviese un recurso con, por ejemplo, tres instancias servirá este tipo de semáforo. De esta forma si un cuarto proceso quisiese utilizar el recurso, no se le permitiría hasta que otro no lo libere. En cambio, en los semáforos binarios (mutex) solo sirven para uso exclusivo, es decir, recursos que solo pueda ser utilizado de a un proceso a la vez. Por ejemplo, una impresora.

- **¿Qué representa un número negativo en un semáforo con lista de espera?**

  La cantidad de procesos dormidos

Monitor:

Es un **tipo abstracto de datos** que va a **encapsular** todas las operaciones sobre los recursos compartidos. Estructura de alto nivel que se invocara por los procesos.**Es brindado por el compilador.**

Un solo proceso puede estar dentro del monitor al mismo tiempo

Para esto se va a necesitar otro tipo de variable de tipo CONDITION

Se contará con las I**NSTRUCCIONES de WAIT Y SIGNAL(no son las mismas del semáforo)** que van a administrar la cola de **entrada al monitor**

Nos garantiza que va a estar bien programado ya que es el mismo mecanismo que utilizarán todos los procesos.

●**cwait(c):** Suspende la ejecución del proceso llamante en la condición c. El monitor queda disponible para ser usado por otro proceso.

● **csignal(c):** Retoma la ejecución de algún proceso bloqueado por un cwait en la misma condición. Si hay varios procesos, elige uno de ellos; si no hay ninguno, no hace nada.

Los monitores son una construcción de lenguaje de programación, así que el compilador sabe que son especiales y puede manejar las llamadas a procedimientos de monitor de una forma diferente a como maneja otras llamadas a procedimientos.

Por lo regular, cuando un proceso invoca un procedimiento de monitor, las primeras instrucciones del procedimiento verifican si hay algún otro proceso activo(verifica c) en ese momento dentro del monitor.

Si así es, el proceso invocador se suspende(cwait(c)) hasta que el otro proceso abandona el monitor.

Si ningún otro proceso está usando el monitor, el proceso invocador puede entrar(Csignal(c)).

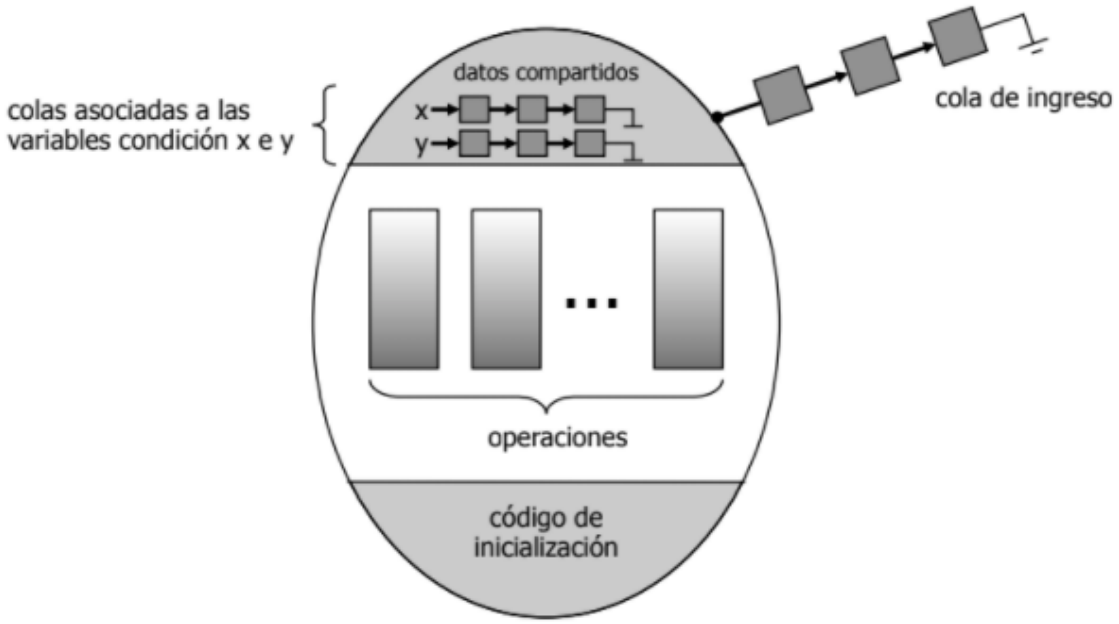
Es responsabilidad del compilador implementar la exclusión mutua en las entradas a monitores, pero una forma común es usar un semáforo binario.

Ventaja:

Al ser de alto nivel de abstracción resuelven internamente el acceso de forma segura a una variable o a un recurso compartido por múltiples procesos concurrentes.

Resuelve los posibles problemas de uso de los semaforos.

No utiliza variables globales, sino que esta limitado a las variables locales del monitor y parametros



Dato:Los semáforos y monitores son una buena solución, pero no se pueden utilizar en computadoras diferentes porque no tenemos computadoras con memoria compartida, y sirven para el intercambio de mensajes

Forma de atomizar las ejecuciones de secciones criticas:

Transacción: Secuencia de operaciones de lectura y escritura que terminar con una operación de confirmación (COMMIT) o de cancelación (ABORT). Se ejecuta atómicamente.(son vistas por el resto del sistema como una sola instrucción indivisible.)

///////// III- Memoria

Manejado por el subsistema de memoria , el cual es parte del Kernel.

Memoria está compuesta de una gran matriz de palabras o bytes, cada uno con su propia dirección

La CPU extrae instrucciones de la memoria de acuerdo con el valor del contador del programa. Las instrucciones solo pueden ser ejecutadas si estan cargadas en memoria.

Pirámide de la memoria: (más rápido y caro desde arriba a abajo)

1- Registros de la CPU
2- Memoria Caché
3- Random access memory (RAM) (memoria principal)
4- Memorias periféricas secundarias
5- Memorias periféricas terciarias

registros de CPU → 1 ciclo de reloj usualmente

memoria principal→ Se accede mediante el bus, puede llevar muchos ciclos de reloj, lo que hara que el procesador deba detenerse para requerir mas datos o instrucciones

Cache → que le CPU se detenga es intolerable, por lo tanto se añade memoria rapida entre cpu y memoria principal

No solo la velocidad de acceso se debe tener en cuenta sino que además se debe garantizar una correcta protección al SO de procesos de usuario y a procesos de usuario de otros procesos. Debe ser implementada por el hardware.

Ciclo normal

Selecciona procesos en cola de entrada → carga proceso en memoria →

A medida que proceso se ejecuta accede a instrucciones y datos →

Proceso termina su ejecución → espacio de memoria se declara disponible

Cola de entrada: compuesta por los programas que residen en disco esperando a ser cargados en memoria para su ejecución.

- **¿Cual es la problemática principal en la gestión de memoria?** Resolver la transformación de una dirección lógica a una dirección física de memoria.

La memoria principal, cache y los registros integrados dentro del propio procesador son las únicas áreas de almacenamiento a las que la CPU puede acceder directamente.

Memoria Caché:

memoria rápida entre la CPU y la memoria principal, es un búfer de memoria (espacio reservado para el almacenamiento temporal)

Memoria principal:

Aquella memoria donde reside tanto el código a ejecutarse como los datos del proceso en ejecución.

cpu puede acceder directamente y donde guarda instrucciones y datos.

Es direccionable a nivel de palabra de memoria (Hoy en día es a nivel de un octeto o Byte) (byte lógico 8 bits/ byte físico puede tener 9 o 10 (son para recuperar error)))

SER VOLÁTIL ES UNA DEFICIENCIA DE LA TECNOLOGÍA NO UNA CARACTERÍSTICA , debe ser rápida.

En verdad no es de acceso “aleatorio” sino acceso “indistinto” es decir que va leer cualquiera de los datos al mismo tiempo y sin necesidad de leer ningún otro dato

Memoria de permanencia: Aquellas en las cuales guardamos datos con menor o mayor grado de permanencia

Protección

Cada proceso tiene su propio espacio de direcciones, que será su rango de direcciones legales que puede acceder.

Para proporcionar esta protección se puede usar dos registros (son cargados por el SO (el dispatcher como parte del proceso de cambio de contexto))

Registro base: menor registro en memoria del proceso.

Registro límite: especifica el tamaño del rango.

el hardware de la CPU compara todas las direcciones generadas en modo usuario con estos registros.

En caso de que un proceso de usuario quiera acceder al espacio del SO o de otros procesos de usuario hará que se produzca una excepción.

El cambio de contexto actualizar´a el valor de dichos registros Ímite

En sistemas actuales la protección la proporciona el sistema de direccionamiento
Tanto la segmentación como la paginaci´on proporcionan una protección eficaz de la memoria. El responsable de la seguridad está en la MMU

Sigue siendo necesario, obviamente, que haya al menos dos modos de ejecucíon, modo usuario y modo sistema (kernel, supervisor . . .)

Dirección de memoria:

Es un número hexadecimal que va a referir a la dirección de una instrucción o un dato.

Espacio de memoria:

Conjunto de direcciones de memoria(visto más arriba,detallo en gris).

Espacio de direccionamiento: Está constituido por el conjunto de direcciones posibles (lógico). Define la capacidad de direccionamiento → longitud de dirección

Define la cantidad de bits que tiene la longitud de palabra de direccionamiento o longitud de direccionamiento

La unidad de almacenamiento es el bit (binary element)aunque normalmente la consideramos estructurada en bytes (8 bits)

(8 bits → 2^8 direcciones 64 bits → 2^64 direcciones) .

Espacio de direcciones: Son todas las direcciones que estoy usando, es decir la actual. Es un subconjunto de espacio de direccionamiento. Cada uno va a tener su identificador constituido por un ACID + la dirección

- **¿Puedo administrar una memoria de 16 TB con un procesador de 32 bits?**

  Si, lleva más ciclos de reloj. Por ej, el MS-DOS administraba 640K con un procesador de 16 bits

TIpos de direcciones:

Direcciones simbólicas: o variables, son las que utilizan los programadores.

Direcciones físicas: Es el lugar donde está guardado ese valor en la memoria principal. Se representan en Hexa. se carga en el registro de direcciones de memoria.

Dirección lógica: Es la dirección genera y trabaja la CPU. Solo es diferente a la física en reasignación en tiempo de ejecución)

Dirección Simbólica	Dirección reubicable	Dirección Absoluta
"saldo"	*"14 bytes a partir de …"*	74014

Los programas de usuario deberán pasar por una serie de pasos para poder ser ejecutado.
Reasignación de direcciones

¿Cómo se transforma de dirección simbólica a dirección lógica?

Se va a encargar el compilador, ensamblador o intérprete. Lo va a generar en código binario.

¿Cómo se transforma de dirección lógica a física?

Pasa de la **CPU a la RAM**(pasar por la reasignación). Puede ser en 3 momentos:

Tiempo de compilación: El compilador ya nos da dir. físicas(dir.absolutas). por lo tanto dir lógica = físicas. obliga a compilar cada vez que se debe ejecutar, ya que no se conocía la dirección base

Tiempo de carga: el compilador genera código absoluto a partir de la dirección 0(en dir lógica), pero en el momento de carga (física) va a leer la dirección base al momento que lee, y se la suma para formar dirección física. Genera código reubicable (solo se le debe sumar una dirección base)

Tiempo de ejecución: Incorpora la MMU(Memory management unit) que sumará la dir. base a la dir lógica, cada vez que se acceda a la ram (es decir en ejecución). Genera código reubicable comercializable (genera un boom)

En la RAM las direcciones bajas están, en su mayoría, ocupadas por el SO(Kernel), a partir de una dirección que se llama **“dirección base”** empiezan a haber direcciones libres.

- **¿Pueden diferentes direcciones lógicas de memoria de dos o más procesos ser traducidas por la MMU en la misma dirección física? Explique su respuesta.**

  Si, debido a que por ejemplo algunas páginas de varios procesos pueden residir en las mismas direcciones físicas en caso de que sean páginas compartidas? (Siempre y cuando sea código puro, y no datos)

Transformación paso a paso: desde que se escribe hasta que se vuelve ejecutable

Originalmente tenemos un código fuente, el cual posee direcciones simbólicas (variables), el código fuente va a ser el imput del compilador que nos va a dar un output un “modulo objeto” que es binario pero no ejecutable. Compilador deja el espacio de direcciones requerida

1) (código fuente → compilador → código binario No ejecutable con stubs(módulo objeto))

Al encontrar referencias que no existen en su código, se crearán “stubs” para las librerías externas. Las cuales a partir del “link editor” va a recorrer el módulo objeto del compilador y en cada stub quedó un link que incorporara de las librerías otros módulos objetos ya compilados. (En este paso solo resuelve los stubs de usuario(enlaces con librerías), no los del SO, ya que sino queda anclado a las librerías del SO del momento de compilación)

2) (Codigo binario → link editor carga los stubs de usuario → modulo de carga)

A partir de esto la salida va a ser un “módulo de carga”, y lo va a recibir la rutina del SO “Loader” o cargador del sistema operativo, en esta parte se encontrarán los stubs de referencia del SO, la salida será un módulo ejecutable binario.

3) (módulo de carga → loader(cargador) cargará stubs obligatorias de SO → binario ejecutable)

Durante la ejecución es posible incorporar las “Librerías dinámicas” las cuales están en memoria secundaria (las .dll).

4) (binario ejecutable → ejecución (puede cargar stubs de SO en ejecución . incorpora .dll)

Carga dinámica: Se cargan en memoria únicamente las rutinas que van a ser utilizadas, usando menos memoria física

1) (código fuente → compilador → código binario No ejecutable con stubs(módulo objeto))

2) (Codigo binario → link editor carga los stubs de usuario → modulo de carga)

3) (módulo de carga → loader(cargador) cargará stubs estaticas de SO → binario ejecutable)

4) (binario ejecutable → ejecución (puede cargar stubs de SO en ejecución . incorpora .dll)

Según este método:

1- Primero se carga el programa principal y se ejecuta

2- Cuando una rutina necesita a otra , se comprueba si esta cargada, sino se invoca al cargador

3- este lo cargue en memoria la rutina deseada, actualiza tablas y se ejecuta

Montaje dinámico y bibliotecas compartidas

El montaje se da se da cuando se deben montar las bibliotecas de lenguaje del SO.

Dos tipos de montaje:

montaje estático se da cuando las bibliotecas del lenguaje utilizado en el sistema son tomadas como un módulo y son integradas en el programa.

El montaje binario se da cuando se pospone hasta el tiempo de ejecución el montaje del programa. Sin embargo el montaje dinámico ocurre cuando en el fragmento de código se incluye un stud, una pieza de código que indica cómo encontrar o agregar la rutina deseada en la imagen binaria.

Es decir, el montaje dinámico retrasa el montaje de los módulos externos hasta que el módulo de carga haya terminado de crearse.

Biblioteca compartida: es cuando puede haber mas de una version de la bibilioteca cargada en memoria utilizada por distintos programas

- **Cite tres ventajas ofrecidas por las bibliotecas de enlace dinámico cuando se las compara con las bibliotecas enlazadas estáticamente para formar un fichero ejecutable.**
  • Los ficheros ejecutables ocupan menos espacio.

• Los procesos compartirán memoria.
• Los programas no tendrán que ser recompilados ni reenlazados para tener acceso a las mejoras de implementación efectuadas en las bibliotecas.

La diferencia entre carga dinámica y montaje binario es que en el caso de la carga dinámica se pospone la carga de la rutina al momento de la ejecución y en el caso del montaje lo que se pospone montaje de las bibliotecas

Asignación de memoria

Asignación contigua de memoria:

Cada proceso en una única sección contigua de memoria

Funcionamiento:

arranca de 0 y suma la longitud de instrucción, y la siguiente irá desde el final de la anterior y se le sumará la nueva longitud(eso forma el segmento de código).

arranca de 0 y suma la longitud de variable, y la siguiente irá desde el final de la anterior y se le sumará la nueva longitud(eso forma el segmento de datos).

(0000(2 bytes) / 0002(4 bytes) / 0006(2 bytes) / 0008 / ...)

Para el caso de las variables “variables” se guardará un puntero de un byte que será un puntero a donde en realidad está guardada esa variable (dentro del denominado HEAP).

Por lo tanto el compilador generará el segmento de código y el segmento de datos.

Calcula el tamaño de segmento de código, obtiene una constante, y a todas las direcciones del segmento de datos le suma esa constante.

(tam de segmento de codigo → constante) // (direcciones de segmento de datos + cte)

(0000(2bytes+cte(2bytes)) / 0004 (4 bytes + cte) / 000A / ...)

El cargador al momento de cargar un proceso en memoria le dará una dirección a la MMU, este le sumará una constante(dir. base) y va a ir cargándose uno al lado del otro en memoria.

Funciona bien cuando queremos cargar un solo proceso. … para varios procesos surgio el MVT y el MFT

Al querer ejecutar más de un proceso se creó el swap...

SWAP:

Solución para cargar varios procesos. En este se cargaba el proceso en disco(area se swap), el cual pasaba a estar “suspendido” y mientras tanto se cargaba otro proceso.

Implica mover parte o todo el proceso de memoria principal al disco. Cuando ninguno de los procesos en memoria principal se encuentra en estado Listo, el sistema operativo intercambia uno de los procesos en WAIT a disco, en la cola de Suspendidos. Esta es una lista de procesos existentes que han sido temporalmente expulsados de la memoria principal, o suspendidos. El sistema operativo trae otro proceso de la cola de Suspendidos o responde a una solicitud de un nuevo proceso. La ejecución continúa con los nuevos procesos que han llegado **La zona de intercambio puede ser una partición dedicada o un fichero de disco. Solo disponible en reasignación en tiempo dinámico**

En sistemas antiguos se intercambiaban procesos enteros para aumentar el grado de multiprogramación

--Es lo que se denomina swapping

En sistemas modernos, con memoria virtual, lo que se intercambia son las ṕaginas poco referenciadas de los procesos

--Es lo que se denomina paginamiento bajo demanda

Sistemas simples: sistemas no multiprogramados

-El sistema más simple de administraciṓn de memoria es carecer de el:

valıdo en sistemas dedicados

-En un sistema operativo no multiprogramado solo dos zonas de

memoria: el S.O y el proceso de usuario

En el caso de sistemas operativos multiprogramados, el ḿetodo mas

simple es dividir la memoria en varias zonas.(Son de carga contigua)

--Un proceso en cada zona

--Dos enfoques:

Con la llegada de multiprogramación(carga en memoria de varios procesos) se crearon dos algoritmos ...

MFT: (multiprogramming with a fixed number of tasks) Con la llegada de la multiprogramación, Ibm creó un SO que poseía el programa de control MFT que Divide la Ram en segmentos (particiones) de tamaño fijo la cual se le asigna un proceso a cada uno. Genera fragmentación interna (en cada segmento quedaba espacio libre sin utilizar)

MVT: (Multiprogramming with a variable number of tasks) Aparece el planificador de largo plazo, las particiones las crea el SO y son variables. En su primera asignación el uso de recursos será óptimo pero a medida que pase el tiempo va a tender a generar fragmentación externa, esta se da cuando hay pérdida de un recurso por no estar asignado a ningún proceso pero no poder ser asignado por su capacidad en ese momento. Para solucionar esto se hacía SWAP para volver a reorganizar las particiones(compactación), pero esto llevaba mucho tiempo.

Fragmentacion interna: Pérdida de un recurso por ser asignado a un proceso que no lo utiliza por completo

Fragmentación externa: Aparece cuando hay espacio de memoria total suficiente como para satisfacer una solicitud, pero esos espacios disponibles no son contiguos, el espacio está fragmentado en gran número de bloques pequeños.

para solucionar este tipo de fragmentación se usa compactación, es decir mover todo el contenido de la memoria de forma que forme un gran bloque de memoria libre contigua

¿Qué tipo de algoritmos se utilizan para gestionar sistemas con asignación contigua de memoria? ¿Qué tipo de fragmentación se puede producir en estos sistemas?

a) Algoritmos:

“el primer ajuste” (first fit): asigna el primer registro lo suficientemente grande

“el mejor ajuste” (best fit): asigna el agujero mas pequeño que tenga tamaño suficiente, se debe explorar la lista completa y después decidir.

“el peor ajuste (worst fit)”: asigna el agujero de mayor tamaño

la de primer y la de mejor ajuste son mas mejor en tiempo y utilizacion de espacio que la peor ajuste. la de primer ajuste es mas rapida de implementar que la de mejor ajuste.

b) Fragmentación externa en el caso de primer y peor

Paginación:

MMU divide la ram en espacios de direcciones de tamaño constantes llamadas “**Marcos**”, el compilador dividirá los espacios de memoria de procesos en **“páginas”** también de tamaño cte.

Cada dirección lógica ahora será una “máscara” que estará compuesta por un número de página + un desplazamiento.

El número de página se mapea con el número de página base del marco donde se encuentre y el desplazamiento va a ser el desplazamiento dentro de ese marco.

Requiere tener una :

- **TABLA DE PÁGINA por proceso**, que será administrada por la **MMU**(y alojado en esta), que contendrá la dirección base de cada marco en la que esté almacenada cada página del proceso, y cada una, mediante el loader, es mapeada con la dirección física de cada una. Hay una tabla para cada proceso, con tantas entradas como páginas.

  No es necesario que las páginas estén contiguas en memoria, por lo que no se necesitan procesos de compactación cuando existen marcos de páginas libres dispersos en la memoria.

  Es definida por el dispatcher como parte del context switching

  La proteccion se da agregando un bit de valides a cada entrada, sera valido si la pagina se encuentra en el espacio valido de direcciones del proceso, de lo contrario sera invalido

  Su funcionamiento se da a partir de la dirección lógica (CPU) de página que contiene número de página + desplazamiento, se usa ese nro de página para buscar en la tabla de páginas la dirección base del marco donde fue cargada la página, se toma la dirección física de esa dirección base y después se le suma el desplazamiento de la dirección lógica, de esa forma encuentra la dirección física (RAM).

  número de página → Tabla de página → dir base física de marco de página + desplazamiento → dirección física

- Requiere **Lista de marcos libres** donde guardar los marcos sin asignar. Existe una sola tabla para todo el sistema.

suele tener fragmentación interna correspondiente sólo en la última página de un proceso. NO tiene fragmentación externa

-La paginación permite compartir memoria entre distintos procesos:

basta que las entradas correspondientes de sus tablas de páginas apunten a los mismos marcos de memoria física. Las paginas compartidas deben ser de código puro.

-El tamaño de página lo define el hardware

Tamaño de página grandes → mejor tiempo de E/S
→ minimiza el número de fallos de página

Tamaño de páginas chicos
→ menos numero de paginas, por lo tanto mayor tamaño de tablas de páginas
→ menor pérdida por fragmentación
→ menos cantidad de E/S (por mejor localidad, se puede ajustar más precisamente a su localidad)
→ Mejor “resolución”, es decir se carga solo lo que se necesita

tendencia histórica es ir a páginas grandes

Mecanismos/Soporte de hardware para paginación

Con el crecimiento de las memorias y los tamaños de los procesos(tamaño de página limitado por el tamaño de registros de la MMU) se tuvo que mover las **tablas de página a la RAM** en la zona del kernel, y se crea la **PTBR**

PTBR

(page table base record) la cual tendrá la **dirección base de cada tabla**, está PTBR es cargada en cada context switching dejando en la MMU la dirección de la tabla de página del proceso.

Esto hacia mas lento todo ya que ahora pasaba a requerirse **dos ingresos a la RAM**

primer ingreso buscando la tabla de pag (dir base tabla + nro de página * longitud de registro)

Segundo ingreso buscando la dirección física de la página (dir base marco + desplazamiento)

Como las tablas de página se asignaban contiguamente en memoria se iba generando fragmentación externa… Esto hizo que surja la paginacion jerárquica.

Posteriormente se implementó la TLB ...

TLB

La cual es **una caché** (a pesar de llamarse buffer es un caché, ya que tiene datos repetidos), esta memoria tendrá unas cuantas entradas de tabla de páginas , es decir va a tener el nro de pagina y su direcciones base del marco correspondiente.

Al momento de buscar con el número de página, se va a buscar con la tlb(1 ciclo de tiempo) y también con el proceso tradicional, todo al mismo tiempo, en caso de encontrarlo en la tlb se anula la búsqueda tradicional, ahorrando tiempo. Se busca que el 98% de las búsquedas se hagan por tlb. En cada context switching es vaciada.

- **Explique qué funciones realiza la TLB, qué ventajas tiene su utilización y en qué tipos de administración de memoria es usada?**

  Tanto en la administración de memoria por paginación como en la administración de memoria por segmentación, en sus diversas implementaciones.No se utiliza en administración de memoria continua. Ni en MVT ni en MFT.
- **¿Es aceptable una taza de acierto de la TLB de 60%?**

  No. Una tasa de acierto del 60% implica que debemos concurrir a la RAM 4 de cada 10 direcciones para resolverla, esto es lentísimo. Se habla de tasas de acierto entre 95 a 98% para ser aceptables.

para evitar que las tablas de páginas sean demasiado grandes y cargadas de forma continua …

Estructura de la tabla de página

Paginación jerárquica:

La dirección de página será una máscara que estará constituida por **página externa + página interna + desplazamiento** (Esto lo hace el compilador)

Permite manejar tablas de páginas muy grandes sin necesidad de asignar espacios contiguos a ella

Con la dirección de página externa entró a la **tabla externa** que tiene guardadas las direcciones base de todas las tablas internas(tabla de pagina).

Lo negativo es que debe ir **3 veces a la memoria ram**, la primera vez, después quedará cargado en la TLB

pagina externa + dir base tabla externa en ptbr → tabla externa →

dir base tabla de página + página interna --> tabla de página →

Dir base marco + Desplazamiento → Dir. fisica

Sin fragmentacion externa. Se usa para 32 bits

como los tamaños de página seguían incrementando entonces se optó por …

Hash

Tenemos 64 bit de espacio de direccionamiento enorme, pero normalmente usaremos 40 , 44 bits… la técnica de paginar las tablas ya no era la mejor idea...

Utiliza una función hash la cual se ingresa con el nro de página muy grande como dominio y aplicando técnicas matemáticas nos proporcionará un codominio mucho más reducido

numero de pagina → función HASH → dirección base física + desplazamiento → dir fisica

equivalente de tabla nos va a decir desde dónde leer en la ram.

Clustering

La función Hash direcciona a un puntero a una lista encadenada de registros (clusters) que tendrán nro de pág. de las direcciones e irá comparando con el nro de pág ingresado a la funciona hash

numero de pagina → función HASH → lista de registros → dirección base física + desplazamiento → dir fisica

Tabla de paginas invertida

-Una entrada por cada marco de memoria física. Tabla única en todo el sistema.

-Cada entrada contiene información de la página que contiene dicho marco (proceso y direccion lógica)

- Reduce la cantidad de memoria para necesaria para almacenar tabla de pagina

-Aumenta el tiempo de busqueda en la tabla de paginas: se usa una tabla hash para reducir el tiempo

- dificulta utilizar paginas de memoria compartida

- **Cuando conviene usar paginacion jerarquica y cuando paginacion por hash?:**

En breves palabras , la paginacion jerarquica hasta sistemas de 32bits y la paginación por hash para sistemas de 64bits... (fundamentalmente dado por el espacio de direccionamiento y la cantidad de direcciones que puede direccionar)

Para el caso de 64 bits se usará la función HASH y clustering

- **Explique qué método de administración de memoria conviene usar cuando la cantidad de la misma supera los 100 GigaBytes.**

Se usa la paginación de memoria, con tablas de páginas en clúster. Estas tablas es una variante de las tablas hash, con la diferencia de que cada entrada de la tabla hash apunta a varias páginas y las entradas de la tabla de página en clúster, apunta a un clúster de páginas

- **¿Por qué los tamaños de página son siempre potencias de 2?**

  Si no fuesen potencia de dos, un programador malicioso puede generar direcciones con desplazamiento mayor a la página, y eso me obliga a controlar cada dirección, como el caso de la segmentación. Usando toda la capacidad de desplazamiento, ese requerimiento desaparece.

- **¿Por qué en distintos sistemas existen diferentes tamaños de páginas?**

  Porque cuando comenzaron a existir distintos tipos de archivos, como los multimedia, se dieron cuenta que no podían utilizar el mismo tamaño de página tanto para un archivo de video como un archivo de texto, necesitando este último un número extremadamente menor de páginas. Por lo tanto, el SO se encarga de modificar el tamaño de página dependiendo el tipo de archivo.

páginas de 4 Kbytes → 32 bits

Entonces cuando las direcciones son demasiado grandes las 3 formas de resolverlo son

1. **Paginación jerárquica**
2. **Hashing**
3. **clustering sobre Hashing**

Estructuración: Divide un problema en múltiples problemas cada vez más pequeños guardando la relación entre las partes, dividiendo hasta que la solución del subproblema sea obvia. A partir de esta idea surgió la segmentaciòn

Segmentación:

Un **segmento** es un espacio propio de direcciones para cada función y cada rutina respetando el punto de vista del programador.
Un proceso se divide en un conjunto de segmentos de tamaño variable

La dirección lógica va a estar dada por **número de segmento**(generado por compilador) y **desplazamiento**.

Cuando un proceso se trae a memoria, todos sus segmentos se cargan en regiones de memoria disponibles, y se crea la tabla de segmentos.

Tabla de segmentos, que va a encontrar la **dirección base** y la **longitud del segmento**), con esta dirección se va a sumar el desplazamiento para transformar a la dirección física.

la **longitud de segmento** se usa para evitar que procesos entren en espacios de memoria de otros segmentos. Se compara con el desplazamiento.

nro de segmento → tabla de segmentos → dir base segmento + desplazamiento → dir fisica

La primera vez que resuelve una dirección base se carga en la TLB.

al ser distinto de tamaño los segmentos hay un poco de fragmentación externa (pero muy poca ya que los segmentos son pequeños)
Se usó hasta los 16 bits

Similar a la MVT, sólo que los distintos segmentos de un proceso no tienen por qué estar contiguos en memoria.

Ventajas:

- El programador puede conocer las unidades lógicas de su programa, dándoles un tratamiento particular.
- Comparado con la paginación, por ejemplo tenemos una rutina que debido a su tamaño esta dividida en 2 o mas páginas, deberán resolverse esas paginas para poder ejecutar la instrucción. En cambio con segmentación esa rutina estaría en un unico segmento que se cargaría completo en memoria.

Segmentación paginada (o segmentación jerárquica):
Se página cada segmento para poder administrarlo en tamaños más pequeños. Dir lógica ahora será **Número de segmento** + **Número de página** + **desplazamiento** (dentro del marco).

En lugar de tratar un segmento como una unidad contigua, este puede dividirse en páginas. Cada segmento puede ser descrito por su propia tabla de páginas.
Se usa a partir de 32 bits

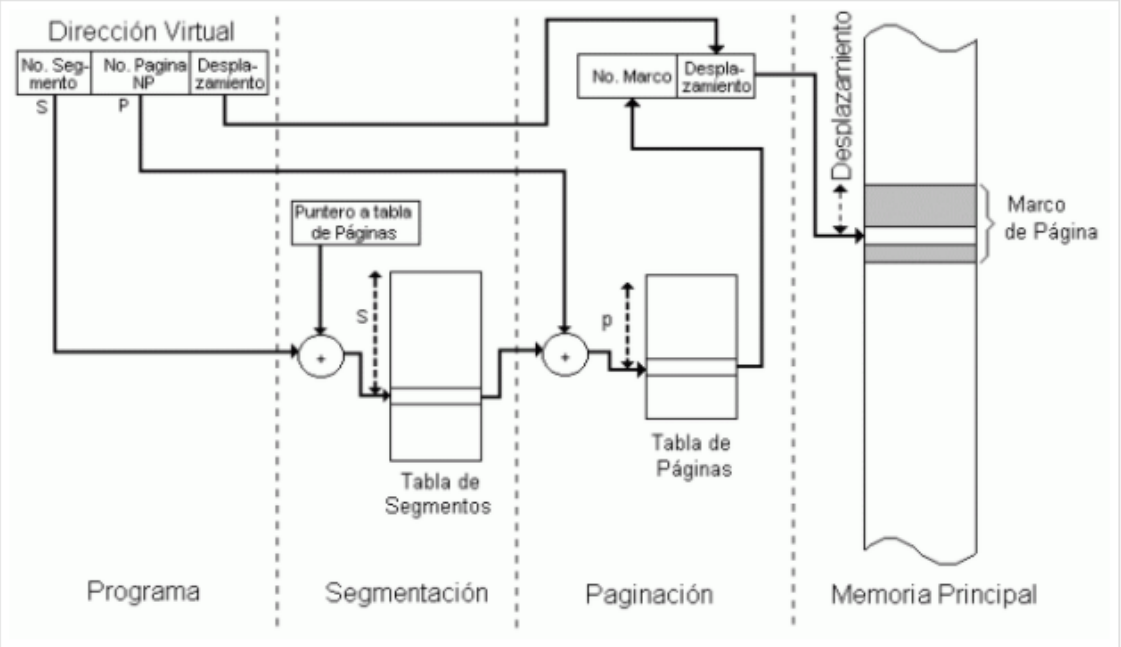
Ventajas:
- El esquema de segmentación paginada tiene todas las ventajas de la segmentación y la paginación
- Se elimina el problema de la fragmentación externa y la necesidad de compactación.

Desventajas:
- El costo es mayor que en el caso de de segmentación pura o paginación pura.

- Sigue existiendo el problema de fragmentación interna de todas -o casi todas- las páginas finales de cada uno de los segmentos. Bajo paginación pura se desperdicia sólo la última página asignada, mientras que bajo segmentación paginada el desperdicio puede ocurrir en todos los segmentos asignados.
-

tabla de de segmentos: una para cada proceso(pongo en gris porque es dudoso)
tabla de páginas: una por segmento
tabla de bloques(o marcos) de memoria: dividida en páginas del mismo tamaño

nro segmento → tabla segmentos → dir base tabla pág + nro página → tabla de página → dir base marco + desplazamiento → dir fisica



2. De entre todos los esquemas de gestión de memoria estudiados (tanto con asignación contigua como con asignación dispersa), indique cuáles cumplen cada una de las siguientes características:

a) Posibilidad de compartir memoria

b) Presentan fragmentación externa

a) Segmentación, paginación, segmentación paginada

b) segmentación, particiones variables

7. Enumere y clasifique las técnicas de gestión contigua y las técnicas de gestión dispersa de la memoria.

a) Gestión Contigua: Particiones múltiples fijas, particiones

múltiples variables.

b) Gestión Dispersa: Paginación, segmentación, segmentación

paginada.

Primera visión
Todos los algoritmos de gestión de memoria vistos hasta ahora requieren que las instrucciones que se estén ejecutando estén en memoria principal.
Se coloca el espacio completo de direcciones lógicas dentro de la memoria física, Puede ser aliviado por los mecanismos de carga dinámica pero requiere que el programador tome precauciones adicionales.

Segunda visión
Uso de memoria virtual, y paginación bajo demanda.

Se usó a partir de 32 bits
A medida que los programas se iban haciendo más grandes y usualmente se componían por varios módulos, no era redituable en términos de aprovechamiento de Ram cargar el programa completo, ya que no se usaban todos sus módulos, entonces surgió...

Memoria virtual:
simular tener más memoria ram que la que en verdad se tiene. De dos a tres veces más, pero puede ser mucho más grande. Se hacen nacer muchos más procesos que la capacidad real.
El programador ya no tiene que preocuparse por la cantidad de memoria fisica disponible

la memoria pasa a estar soportada contra la ram y otra parte contra espacio en disco duro.
Permite la comparticion de memoria mediante comparticion de paginas.

Se utiliza en conjunto con la paginación bajo demanda

- **Explique las ventajas de seguir usando una administración de memoria virtual en sistemas donde la memoria real cubre holgadamente las necesidades de los procesos en ejecución.**

No se agiliza el tiempo de ejecución, se disminuye drásticamente el tiempo de INICIO de la ejecución, y se evitan gran cantidad de cargas de páginas que no se van a utilizar en muchos casos.

Paginación bajo demanda: //nunca se uso de forma pura
Se irá cargando las rutinas y procesos bajo demanda. Permite iniciar los procesos rápidamente ya que se cargarán unas pocas páginas inicialmente

Esta parte del proceso que se encuentra realmente en la memoria principal para, cualquier instante de tiempo, se denomina **conjunto residente del proceso** y cargará las páginas de las rutinas según se requiera.

Si la página no está cargada se da un "**fallo de página**", para detectar qué páginas del proceso no están cargados se usa un bit de validez en la tabla de páginas, el cual estará en 1 si está cargada en ram y en caso de no estar cargada se dará una interrupción por fallo de página. El paginador se encarga de intercambiar las páginas.

- en caso de interrupción por fallo de página→

se llama a la **rutina de fallo de página:**

1. determina si la referencia memoria fue válida o inválida (por ejemplo por indizar mal un vector)
2.Si fue una referencia válida, en vez de abortar el proceso, localizar en disco la página buscada
3.Buscar un marco libre
   - **A** Si lo hay, lo selecciona
   - **B** Sino, aplica un **algoritmo de página víctima** para seleccionar una página a quitar, la guarda en disco y refleja el cambio en la tabla de páginas.
4.Mover la página deseada al nuevo marco libre, reflejar el cambio en la tabla de páginas (pone su bit en 1.)
5.Reejecutar la instrucción que provocó el fallo (el proceso no necesariamente cambio de estado).

- En caso de superar la cantidad real de memoria y la pila de marcos libres está vacía→

Se "sacrifica" una página (Hará swap) y se anota en la pila de marcos libres. Para seleccionar la página víctima se usan distintos algoritmos. Después le va a asignar ese marco al nuevo proceso

La tasa de fallos debe ser baja para llegar a ser viable.1 de cada 400.000 páginas pueden dar fallos de página. Sino entramos en sobrepaginación

Entonces, en paginacion bajo demanda se debe resolver dos problemas principales:

- Desarrollar un algoritmo de asignacion de marcos
- Desarrollar un algoritmo de sustitucion de paginas.

Dato: Cosas que ocupan espacio en la memoria: Buffers de E/S Y PROCESOS

Copia durante escritura Permite que los procesos padre e hijo compartan inicialmente las mismas páginas. En el caso de que alguno de los procesos escriba sobre dicha pagina se generar una copia para cada proceso.

Cambia la forma en que se implementa la llamada fork() ya que antes se copiaban desde un inicio todas las paginas del padre para crear otras iguales para su hijo.

Algoritmo de sustitución de páginas

FIFO: Cada página al ser cargada se le asignaba un número de marco sumando uno al anterior en la tabla de páginas y se sacrifica el primer marco. Sufre de la **anomalía de Bellady** (a pesar de dar más recursos la cantidad de fallos de página aumentaba)

Óptimo: Está basado en seleccionar aquella página que más tiempo falta para volver a usarlo. No es implementable porque no se puede saber la futura demanda. Sirve para testear otros algoritmos

LRU (Least recent used): llamado algoritmo de pila**.** Se busca eliminar la página menos recientemente usada(que hace más tiempo que no usamos), para esto se utiliza una pila en la que se van cargando las páginas utilizadas y se elimina la primera de esta lista una vez se complete.

Aproximación LRU: Se hace uso de 4 bits en la tabla de página los cuales nos van a indicar que proceso fue el que hace más tiempo no usamos, su funcionamiento se da poniendo al principio todos los bits en 0000 y cada vez que se use esa página se le pondrá un 1 en el bit más significativo (0000 -> 1000) cada determinado tiempo se borra el último bit y se pone un 0 como primer bit, entonces pasando un periodo de tiempo los bits quedaran asi: (1000 -> 0100 -> 0010). Al momento de fallo de página se tomará como pagina víctima a la página que tengas mas ceros adelante.
En caso de ser empate (misma cantidad de 0 adelante) se usará bits adicionales para minimizar el error.}

Segunda chance: Se tiene un solo bit, cada vez que se usa la pagina se pone el bit en 1, la página no usada se mantendrá en 0, se usará una cola circular con un puntero el cual ira “apuntando” cada bit cada cierto tiempo y en el caso de estar en 1 lo pondrá en 0 y apuntará

al siguiente, y en el caso de estar en 0 se tomará como pagina víctima (se suele llamar algoritmo de reloj). Es fácil de implementar y muy eficiente (añade un único bit extra)
Existe el algoritmo de reloj puro, en este caso al momento de seleccionar una página victima en la cola circular, el puntero volvía al principio en vez de ir al siguiente.

Segunda chance: con 2 bits: Se puede usar con 2 bits. Un bit anotara usado o no usado, y el otro bit anotará si se escribió en ella (no se anota nada si solo se hizo lectura de datos).
En el caso de estar en 00 se tomará como víctima, si está en 01 quiere decir que no se uso pero se escribió en ella por lo tanto se va a tener que grabar en disco duro, en el caso que sea 10 quiere decir que se usó pero no se grabó, se usa la segunda chance y se pone su bit más significativo en 0 (10-->00) , y estando en 11 quiere decir que se usó y se grabó.
La ventaja sobre el algoritmo de segunda chance de 1 solo bit es que se tiene preferencia por las páginas que no han sido escritas, reduciendo el nro de operaciones de E/S requeridas.

se usan **algoritmos de sustitución y asignación de marcos**.

No se puede dejar la asignación de marcos al sistema a un sistema puro de bajo demanda

La asignación de marcos

Proporcional Se distribuye proporcionalmente

Equitativa: Se asigna la misma cantidad de marcos para cada proceso. Es injusto

Sustitución de marcos

local: Respeta la asignación pero va a tener una performance inferior.

global: Permite a un proceso seleccionar un marco de sustitución de entre el conjunto de todos los marcos Sustituye en caso de haber marcó libre pero asignada a otro proceso

sustitución global → Utiliza de forma más eficiente la ram ya que no sacrifica si hay un marco libre aunque esté preasignado a otro recurso.

procesos interactivos → la asignación es **equitativa** y sustitución **local(porque no se debe castigar a otros usuarios)**

batch → la asignación es **proporcional** (En batch se usa el SCL)y la sustitución puede ser **local o global** según la configuración del SO (un servidor que atiende múltiples clientes será local y un centro de cómputo será global)

cantidad mínima de marcos
es determinada por el hardware (registros de la CPU) y está dada por el nivel de indireccionamiento, es decir al la cantidad de direcciones que necesita para obtener el dato. es decir por ejemplo una instrucción que hace referencia a una dirección indirecta y esta a su vez a otra y asi, eso generaría niveles de indireccionamiento.
El número máximo de marcos esta definido por la cantidad de memoria física disponible.

Hiperpaginación (thrashing):
el proceso está más tiempo paginando que procesando.
Se da porque el número de marcos asignados a un proceso cae por debajo del número mínimo requerido por la arquitectura de la máquina, este generará rápidamente fallos de página ya que deberá sustituir paginas que deberán ser usadas en poco tiempo.
Genera una alta tasa de paginación.

Causas:
Batch el planificador de mediano plazo y el planificador de largo plazo no evalúe correctamente la carga del CPU. poco uso de cpu y mucho E/S

procesos interactivos puede pasar que un usuario habrá muchos procesos y genere el agotamiento de memoria real, o una mala programación de algún programa.

¿Cómo se determina cuántos marcos se van a necesitar ?

-En los sistemas que usen segmentación

Dado que la segmentación respeta el punto de vista del programador, es decir que el código se divide en diferentes segmentos de distinto tamaño según la forma en que se programa, se puede saber cuántos marcos de datos se necesitan(lo dice el compilador)

-En los sistemas que no usen segmentación

Prevenir la sobrepaginación:

Modelo de localidad
Una localidad es un conjunto de páginas que se utilizan de forma combinada. Todo proceso está generalmente compuesto de varias localidades diferentes

Detectar de acuerdo a las llamadas(migración de rutinas por ej) y el nivel de uso que está teniendo, las localidades en las cuales se mueve y asignarle marcos a esas localidades.

Dato: Notar que el modelo de localidad se condice con el uso de las cache y su utilidad.

Modelo de conjunto de trabajo:
Es una aproximación del modelo de localidad el cual examina las referencias a las páginas más recientes para formar las “localidades”, se usa un parametro Delta para saber el numero de referencias en el instante dado, y debe calcularse bien si es muy largo se incorporan rutinas anteriores, y si es corto va a faltar la rutina con la que estoy trabajo.
Entonces le asigna marcos suficientes para todo su conjunto de trabajo

Frecuencia de fallos de página (PFF).

Otra alternativa al modelo de conjunto de trabajo. mucho mas sencillo.
Se suele establecer un máximo y un mínimo de errores de página, y a partir de eso se medirá la tasa de fallos de página o **Frecuencia de fallos de página (PFF)**.

Si la tasa es alta → se proporcionará más marcos al proceso
Si la tasa es baja → se sacrificará marcos del proceso

En caso de que no se tenga marcos para darle, si es un sistema batch se baja a disco el proceso y se libera todos sus marcos.

En los sistemas de usuario solo se podra poner un warning

Los SO NUNCA tener la pila de marcos libres vacía para así evitar los fallos de página,

Prepaginacion:
Busca evitar que se caiga en fallo de página con la pila de marcos vacía. Es un intento de evitar el alto nivel de paginación al iniciar un proceso.

Se busca cargar las páginas que vayan a ser necesarias, para esto se usan archivos que contienen estadísticas de uso. o usar los modelos de conjunto de trabajo.

Presacrificio:
al momento de tener la pila de marcos con **pocos marcos disponibles**, se aprovechan los context switching para **ir ejecutando los algoritmos de selección** de página víctima y pre victimizando (marca como marcos libres pero en realidad está cargada).
Garantiza nunca encontrar la pila de marcos libres vacía

memory map
Se usa para reducir el tiempo de acceso a los periféricos
Al tener mucha ram, y tener que cargar un archivo, se cargue completamente en ram y se use como cache del disco, y utilizar los algoritmos de paginación, y cada parte del archivo que se modifique se baje al disco.

Permite una eficiencia superior, ya que al momento de nacer un proceso, este declara los archivos que va a necesitar y la memoria "mapea" este archivo evitando así las interrupciones bloqueantes.

Dato de color
Para el caso de archivos multimedia los cuales tienen grandes tamaños se tuvo que hacer diferenciar las páginas para datos numéricos/código y las páginas para multimedia

Dentro de estas páginas multimedia lo que la va a diferenciar es:
Para los **segmentos de códigos** se usan bloques de 1,2,4,8k (tamaños chicos)y para el **segmento de datos** se tendrá para 1,2,4,8k...pero además de 1,2,4,8m… lo que va a permitir guardar un archivo en una sola página.

- **¿Para qué se utilizan las páginas de memoria de tamaños de MB?**
  Los sistemas actuales utilizan múltiples tamaños de páginas, no uno solo, típicamente de 1,2,4 y 8 KB, y 1,2,4,y 8 MB. Estas son usadas para variables multimedia, típicamente fotos o archivos de música.

E/S mapeado en memoria
Cada controlador de E/S incluye registros para almacenar comandos y los datos que hay que transferir.
Usualmente una serie de instrucciones permiten transferir estos datos entre registros y memoria del sistema.
Como con los archivos, se pueden las E/S se pueden mapear en memoria, se reservan una serie de rangos de direcciones de memoria y se mapean esas direcciones con los registros de los dispositivos.
Las lecturas y escrituras en estas direcciones de memoria hacen que se transfieren datos desde y hacia los registros de los registros del dispositivo

Dato de color:
Se usa el modelo UMA que establece que se tiene un solo banco de de memoria y el tiempo de lectura siempre es igual

existe el NUMA en el que se tiene distintos bancos de memoria con diferente tiempo de lectura… por ejemplo bancos

//////////IV - Administracion de perifericos

Memoria periférica:
Se conectan mediante el canal. Se caracterizan por su permanencia(No pueden ser volátiles). Se requiere un modulo de administracion(filesystem) de memoria periférica

- memoria secundaria: memorias permanentemente conectadas al equipo
- memoria terciaria: dispositivos removibles.

Archivo:
Definición genérica de archivo: Un archivo es un un conjunto de bits identificable, es decir, contenedor lógico, donde se ponen datos, es guardado algo o puede estar vacío.
Estos son mapeados por el SO sobre los dispositivos físicos.
Anteriormente se denominaba como un conjunto de registros (conjunto de campos)(esta definición no va más)

Es un tipo abstracto de datos, por lo tanto se definen las operaciones que se pueden realizar con el

Desde el punto de vista del usuario es la unidad más pequeña de almacenamiento secundario, es decir, no pueden escribirse datos en almacenamiento secundario a menos que estén en un archivo

Este tiene una estructura la cual estará definido por su tipo

Atributos
Características que lo identifican.
También se lo conoce como metadata. Se guarda en la estructura de directorios, que tambien reside en almacenamiento secundario. Depende del Fs

- Id: identificador hexadecimal que es utilizado por el FS
- Nombre: identificador legible por el ser humano
- tipo: Define qué tipo de archivo es
- Ubicación: directorio donde se aloja
- tamaño: Cantidad de bytes que ocupa
- Permisos: medida de seguridad que define qué se puede hacer con el archivo
- fecha de creación
- fecha de modificación
- etc

operaciones:

Los procesos interactúan con el subsistema de archivos a través de un conjunto de System calls . Se debe hacer una busqueda en el directorio para encontrar la entrada asociada al archivo

- Crear: dos pasos: se encuentra espacio dentro del filesystem y se incluye dentro del directorio una entrada al nuevo archivo.
- Escribir:
- leer:
- eliminar: se explora el directorio en busca del archivo, habiendo encontrado la entrada, se liberara todo el espacio de archivo para poder ser reutilizado y borramos también la entrada.
- modificar
- otros

Como alternativa a la búsqueda del directorio, algunos sistemas proveen una system call open() que se usa la primera vez que se quiere acceder al archivo que guardará un índice del archivo en la tabla de archivos abiertos, evitando asi la busqueda en directorio

En entornos multiprocesos se cuenta con dos niveles de tablas, una global(con informacion independiente del archivo) y una por proceso con los archivos abiertos por el proceso y puntero a la tabla global.
Para sincronizar que dos o mas procesos puedan ingresar a un archivo se usan los bloqueos de archivos. Estos pueden ser : Obligatorio (brindado por el SO) o sugerido (responsabilidad del usuario)

- **Concepto de descriptor de archivo**

  Registro donde están todos los datos del archivo

Sistema de archivos (filesystem):
Módulo del SO que se encarga de la administración de archivos y el espacio de almacenamiento en un dispositivo .Su “poder” va a estar determinada por la cantidad de atributos y operaciones posibles a sus archivos.
Reside en dispositivos.

compuesto por dos partes:
- colección de archivos: cada uno de los cuales almacena una serie de datos relacionados
- Estructura de directorios: Organiza todos los archivos del sistema y proporciona información de los mismos

Por ejemplo: Unix UFS, Microsoft NTFS, Dos FAT, Active directory (muy poderoso)
actualmente se puede usar filesystem sin restricción de SO

- **Explique que es un sistema de archivos. Cómo se implementa**

El sistema de archivos es el componente del sistema operativo encargado de administrar y facilitar el uso de la memoria periférica. Sus componentes básicos son: Administración del espacio asignado a cada archivo. Administración del espacio libre, Ad ministración de la estructura de directorios. Etc...

Funciones
- Ubica espacio para los archivos.
- Administra el espacio libre
- Controla el acceso a los archivos.
- Recupera datos para los usuarios.

Accede a los datos usando un mecanismo de buffering (Buffer Cache) que regula el flujo de datos entre el Kernel y los dispositivos.
El mecanismo de buffering interactúa con los manejadores de dispositivos para iniciar la transferencia hacia y desde el kernel

El filesystem acarrea dos problemas de diseño:

- definir aspecto del sistema para el usuario (implica definir archivo, atributos y operaciones y la estructura de organización de directorio)

- Crear algoritmos y estructura de datos que permitan mapear el sistema lógico de archivos sobre los dispositivos físicos de almacenamiento secundario

Tipos de archivos
Define que se va guardar en el. Define estructura interna.

Campo: es el elemento de datos básico, contiene un valor único. Se caracteriza por su longitud y por el tipo de datos. Pueden ser de tamaño fijo o variable

- **Archivo de texto**: conjunto de bytes. Posee formato de tipo texto, con caracteres delimitadores (fin de renglón). Depende de cada Fs.

- **Archivos binarios**:Contendrán código hexadecimal, puede tener permiso de ser ejecutado(ser ejecutable NO ES UN TIPO DE ARCHIVO)

- **Registros**: Conjunto de campos, en cada campo irá un dato, Es el elemento básico para guardar datos.

- **¿Cuándo se dan los bloqueos sugeridos de archivos?**

Los bloqueos pueden ser sugeridos cuando más de un usuario pretende modificar el archivo, estando los procesos ejecutándose en forma concurrente.

La **estructura del archivo** excepto los registros o texto puro, los demás tipos de archivos estarán formados por un **header**(define datos de estructura interna) y un **body**(contenido) (El SO deberá tener el driver para leer el tipo de archivo).

Antes los tipos de archivos eran manejados por el So, pero a medida que fueron siendo cada vez más se pasó a una …

soporte del so en archivos:
Pocos tipos de datos → dificulta el trabajo de programar
Muchos tipos de datos→ sistema crece en medida

Dato: IBM→ SO se encarga de todos los tipos de archivos
Unix→ SO solo se encarga de texo, binarios y registros, después terceriza

tabla de asociación
Determinará cómo se accederá al archivo , a partir de la extensión del archivo.
La extensión está dada por los caracteres después de un punto (.)
En algunos Fs la extensión = tipo de archivo.

se lee de a un grupo de bytes llamado **sector(unidad de lectura)** por la estructura física del periférico

Ficheros:
Archivo de registros de longitud variable o fija.
En caso de ser de longitud variable se guardará un puntero a una zona de memoria.
En caso de longitud fija se llamará registro lógico
El SO sabe cómo manejarlo
Journaling
es una característica de algunos sistemas de archivos que mantienen un registro conocido como "journal" que contiene los cambios aún no realizados en el sistema, de tal manera que en el caso de un fallo del sistema o de energía, se pueden restablecer los datos afectados más rápidamente con una menor probabilidad de sufrir corrupción de datos.
Proporciona transaccionalidad.

almacenamiento de información en disco

Los sistemas de disco normalmente tienen el tamaño de bloque definido de acuerdo con el tamaño de un sector (o múltiplos)
Todas las operaciones de E/S se realizan de a un bloque (registro físico) del mismo tamaño, normalmente el tamaño del registro físico no concuerda con el del registro lógico.

Se puede empaquetar los registros lógicos en bloques físicos.
El tamaño del registro lógico, del bloque físico y la técnica de empaquetado determinan cuantos registros lógicos hay en cada bloque físico.

Entonces se puede considerar al archivo como una secuencia de bloques

Cuanto más grande es el tamaño del bloque, mayor es la fragmentación interna.

Debido a esta asignación de bloques, se desperdiciara una parte del último bloque de cada archivo, por lo tanto tendrá fragmentación interna.
Cuanto mas grande el tamaño de bloque mas fragmentación

Métodos de acceso a los registros de los archivos
accede a los registros de los archivos para cargarlos en memoria

Archivos de Acceso secuencial:
Debido a que los archivos se encontraban en cinta debían leerse uno atrás del otro y eran grabados de la misma forma. La información del archivo se procesa por orden, un registro después de otro.

En el header puede tener el número de registros, puede ser de longitud variable o fija. Cada registro tendrá su delimitador de registro. Es usado para los backups(cinta)

Método de acceso directo(o indistinto):
Nos permite leer un registro sin necesidad de leer otro registro. Solo aplicable con periféricos de acceso directo(unidad de disco).

- **Método de acceso relativo**:
  Se accede a través de un número de registro lógico(no se puede reutilizar)(variables del tipo record). En el directorio(donde guardo los datos de los archivos) voy a saber dónde empieza cada archivo, usando el tamaño de archivos se puede saber dónde buscar los registros físicos. Debido a que no se puede reutilizar el registro lógico generaba espacio que no se aprovechaba

  SO va a pasar de registro logico → registro físico

- **Método de acceso hashing** :
- Se pasa a tener un identificador que no es conocido por el usuario que está asociado a un dato conocido(dni), se utiliza el método de hashing(método de número aleatorio) que resultara en un número más chico que se usará como identificador físico(al tener codominio de 4, se puede dar que haya repetidos, se llaman sinónimos). En caso de tener sinónimos menor al 60% se compara por el dato original (dni)(hace tardar más). En caso de tener +60% se crea un archivo

secuencial donde se guardan los sinónimos. Otra forma es crear una lista encadenada, entre todos los sinónimos del archivo de sinónimos. //No creo que tome, nombrar que puede tener sinónimos nomas

Dato El SO ve a un disco como un array de sectores, y el driver lo va a transformar en pista y sector, y lo mapeara.

Método de acceso ISAM(index secuencial access method):
Se graban secuencialmente (archivo secuencial)(cada registro del archivo está guardado en un registro lógico)y se genera un segundo archivo(índice) mucho más chico que tendrá la clave y el puntero, este archivo va a estar ordenado. Sobre este archivo se le realizan búsquedas binarias

> Clave: es un campo que identifica al registro (por ejemplo al cuil)(a veces llamado clave primaria)
>
> puntero: dice donde esta guardado el registro físico relacionado

ISAM con grafos de múltiples niveles:
Se usan arboles B (grafos) (árbol que parte de una base y ramificando a medida que baja), que estarán guardados en un archivo, cada nodo tendrá los punteros a sus subnodos. Al llegar al último nodo (hoja) va a tener un puntero al registro físico
Es usado en los motores de base de datos

base → nodo1 → nodoX → hoja → archivo físico

Motor de base de datos
Potente pero lenta ya que incorpora una capa de software en cada lectura o escritura.
conviene para cuando se cuenta con muchos archivos.

Acceso a los archivos

Directorio:
El directorio puede considerarse como una tabla de símbolos que traduce los nombres de archivo a sus correspondientes entradas de directorio.

Una entrada de directorio esta compuesta por el nombre de un archivo y su identificador unívoco, este identificador permite localizar los demas atributos del archivo.

Es un archivo con tratamiento especial, donde guardo informacion de todos los archivos (entradas). y organizan los archivos según cierta estructura.

Es lo primero que graba el FileSystem.
Debe contar con operaciones para manipular los archivos.

Tipos de estructuras logicas de directorio

- **Estructura lineal**: Se forma una lista (1 solo nivel) que contiene cada archivo, va a tener un acceso directo al archivo y una lectura secuencial . simple y facil de mantener

- **Directorio de dos niveles**.Cada usuario tiene su propio directorio de archivos de usuario (UFD). Cada uno de los UFD tiene una estructura similar, pero solo incluye los archivos de un único usuario.

- **Subdirectorio** o carpeta: se tendrá un directorio principal y un conjunto de directorios dependientes. Se usa una estructura de árbol en principio, y posteriormente evolucionó en un grafo, por un tema de acceso a los archivos (quería ingresar a un archivo a partir de dos ramas)

  los grafos pueden permitir ciclos (referencias a nodos padres) lo que genera un bucle infinito, para evitar esto se toma nota del camino tomado para no entrar en un ciclo.

  Se tienen **dos tipos de arcos (o links), los duros y blandos**. En los duros la referencia es directamente al ID del archivo mientras que el blando apunta al nombre(legible por el humano) del archivo.

  En el caso de borrar un **link duro** a un archivo que contiene más de un link duro, no se borrará completamente hasta que se borren todos sus links duros.

- **Diferencias entre enlaces duros y enlaces bland**
- **os**

Los enlaces duros apuntan al ID del file, por lo que todos tienen el mismo nivel de importancia, los enlaces blandos apuntan al nombre del file, por lo que si se borra el file, quedan deslinkeados.

- **¿Por qué se inventaron los accesos directos (links) en los sistemas de archivos?**

  Porque en la estructura de árbol de directorios no se permite la compartición de archivos entre distintos usuarios. Por lo tanto, se crearon los accesos directos para poder tener un mismo archivo (no una copia) en distintos sistemas de archivos, generando una memoria compartida.
- **¿Tiene un límite la profundidad de subdirectorios en un FS indexado?**

  Si por razones de implementación, por ej, en EXT3 son 32.000

Montaje:

Un filesystem debe montarse para poder estar disponible para los procesos del sistema.
Al SO se le proporciona el nombre del dispositivo y el punto de montaje.
Verifica que el dispositivo contiene un filesystem válido y finalmente registra en su estructura de directorios

nombre disp + punto montaje → verificación Fs → registrar directorio

se tienen dos **visiones de montaje** de filesystem

1- **Unidades distintas** (Como microsoft, unidades :A,:B,:C,...), cada directorio es una unidad.

2- **Único árbol** con nodo raíz (/), y desde ese nodo raíz van a ir dependiendo los distintos directorios. Se cuenta con un archivo con los cuales tiene que leer y cuales no.

Volumen(partición): espacio de discos con un filesystem propio, puede tener varios SO

> **punto de montaje**
> **Se define como el directorio que se abrirá al momento de conectar un periférico, y es definible**. Se puede montar o desmontar la unidad (al desmontar antes de retirar el dispositivo se puede dañar el filesystem).
> En linux se tienen puntos de montaje definidos(/mnt,/media).

Compartición de archivos
Para implementar la compartición y protección de archivos se mantiene atributos extra, como los permisos y los conceptos de “propietario” ,”grupo”, y “otros” de los archivos

El propietario es quien define qué operaciones pueden realizar los grupos y otros.
Obviamente el propietario puede realizar todas las operaciones

Al momento de solicitar realizar una operacion en un archivo por parte de un usuario, se verificará si este usuario(comparando su ID con el atributo propietario del archivo) cuenta con los permisos. Se suele implementar utilizando una lista de control de acceso que especifica los nombres de usuario y el tipo de acceso que se permiten para cada uno.

> **NFS**:
> network file system, se trata sobre el acceso a un directorio en la red. Nos permite montar un dispositivo remoto(si tiene los permisos). Se agrega el archivo “…” que corresponden a los datos del equipo. (FTP, DFS)

> **Semántica de consistencia**

> Cuando un proceso/usuario modifica un archivo cuando y como deben verlo los otros procesos/usuarios que estén accediendo al mismo tiempo?

1 visión: microsoft implementó el método en el cual uno solo puede escribir y los demás procesos únicamente pueden leer una copia del archivo actualizado al momento de abrirlo.
En cambio unix implementó el método en el cual uno solo puede escribir pero los demás pueden ver los cambios al momento.

2 visión: Todos pueden leer y todos pueden escribir al mismo tiempo. Utilizado en drive por ejemplo. Mejor forma

Dato: Se diferencia los permisos de modificar un **dato ya escrito**, y los de **agregar contenido** (append)

Capas del filesystem

Nos permiten una abstracción que facilita el trabajo, de forma que no sabemos el funcionamiento interno.
Fue un cambio con respecto a la década del 90 en donde sí se sabía cómo funcionaban los dispositivos físicos.(CAMBIO IMPORTANTE).

- **Capa física:** Consta de dispositivos físicos

  Disco duro: Array de sectores.Cada sector tiene un número que lo identifica. Consta de un plato que gira, con una cabeza lectora grabadora, llamamos pista a sus círculos concéntricos. NO DECIR ARREGLO

  Sector: unidad básica de lectura y escritura. No se puede leer o grabar menos que un sector

- **Capa de control**: administrada por la unidad de control de dispositivos (o controladora) (UC) y rutinas de tratamiento de interrupción.
-

  Controladora de dispositivo: traduce entradas de comandos de alto nivel a instrucciones de bajo nivel específicas del hardware, Posee el mapa interno de sectores.

- **Capa de manejador**: Consta del driver o manejador(software que forma parte del SO) se comunicara con la UC que conoce sus comandos que puede ejecutar..

  Driver: Software que forma parte del SO que conoce los comandos que puede ejecutar la unidad de control. puede ser escrito por el propio autor del SO o por el fabricante del hardware,

- **Capa de software**: Es una capa de software(habla en lenguaje de sectores) del filesystem, Consta de un sistema básico de entrada salida (basic i/o system) que se comunicará e invocara con el driver .

- **Capa lógica**: Sistema lógico. Habla en terminos de archivos, disposición de ellos. mantiene la estructura de los archivos mediante FCB, que son el equivalente a la PCB pero de los archivos. Tambien es responsable de la seguridad y proteccion

- **Capa de usuario**: Incluye las aplicaciones, donde se comunica el usuario

  Proceso:

  Usuario lee o escribe un archivo **(Capa de usuario**) →

  Módulo de Filesystem traduce la acción a un sector del periférico (**Capa lógica**) →

  El Basic I/O system lo tomará y se comunicara con el driver,. (**Capa de software**) →

  Driver se comunicará físicamente con la controladora (**Capa de manejador**) →

  La UC manejará el hardware propiamente dicho (**Capa de control**) →

  El hardware realizará la acción de lectura/escritura físicamente (**Capa física**)

  Dato: Gracias al mapa de sectores que posee la controladora se amplió el ciclo de vida de los discos duros, ya que detecta qué sectores están dañados y los mapea en otra parte, usando sectores o pistas que guarda de reserva el disco. Los discos duros pueden tener un +-20% más espacio físico real, que se usa justamente para esto.

Estructura lógica:

Todo dispositivo físico puede estar dividido en varios dispositivos lógicos, cada dispositivo lógico puede tener su propio filesystem. (particiones)

Más en detalle :
En el periférico se tendrá la pista 0 donde está la tabla de particiones, cada partición genera una unidad lógica, cada unidad lógica tendrá el filesystem de esa unidad y ahí estará guardada toda la información correspondiente .

periférico → pista 0 → tabla de particiones → n particiones → n unidad lógicas → n filesystem → información

Esto nos genera un problema… ya que se podía dar la posibilidad de contar con gran cantidad de filesystem al mismo tiempo, y esto genera una gran complejidad de manejo….

A partir de esto surgió una nueva capa intermedia llamada:

VFS(Virtual file system**):**. .
Un VFS especifica una interfaz entre el núcleo y un sistema de archivos en concreto
Capa de abstracción encima de un sistema de archivos más concreto
se visualizan como parte del mismo file system superior
Aísla la funcionalidad básica de las llamadas al sistema de los detalles de la implementación.

n perifericos → n filesystem → Virtual file system → capa lógica del SO

- **¿Cuáles son las ventajas de usar una misma interfaz de llamadas al sistemas, tanto para la manipulación de archivos como de dispositivos?**

  Esto simplifica drásticamente el S. O. Por ej:

  Para unix todo es archivos, de esta manera, todo periférico se maneja por medio del /dev, o sea, escribir en un periférico es escribir en el "archivo" correspondiente del /dev.

  Entonces no es necesario escribir distintas rutinas dentro del kernel.

Implementación de File system
mapear el sistema lógico de archivos sobre los dispositivos físicos

Implementación de directorio

1- Lineal: Un registro al lado del otro. Lista lineal de nombres de archivos con punteros a los bloques de datos (periféricos pequeños por ejemplo CD). Para localizar un archivo se debe hacer una búsqueda lineal lo que hace que el acceso sea lento.

2- Hash: Método rápido donde conociendo el nombre del archivo o ID de referencia, se puede leer los datos de ese archivo. Se almacenan las entradas de directorio en una lista lineal y se utiliza una tabla hash, la cual toma un nombre de archivo y devuelve un puntero a la ubicación de dicha lista lineal

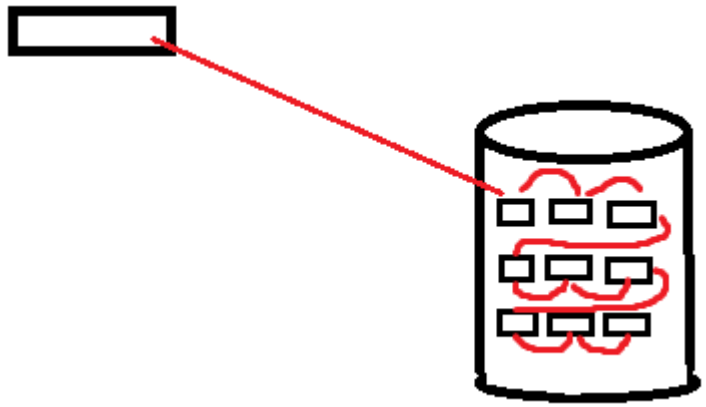
3- Motor de base de datos utilizando árboles (servidores)

Asignación de espacio
Administración del espacio usado.

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

- **Asignación Contiguo:** Usada para acceso secuencial. En la entrada del directorio se tendrá el **nombre del archivo**, la **cantidad de sectores** y un **puntero** q**ue dirá donde arranca** **(el primer registro)**, en base a esos datos leerá todos los sectores.

  Cada archivo ocupa un conjunto de bloques contiguos. Permite acceso secuencial o directo
  Sufre de fragmentación interna y sufre además de problemas para determinar cuánto espacio necesitará un archivo

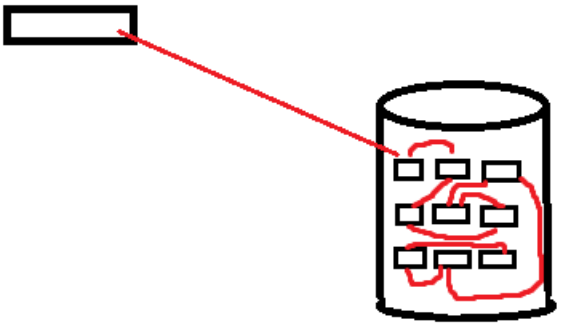


  Sirve para archivos chicos o archivos muy grandes pero de lectura contigua (foto)

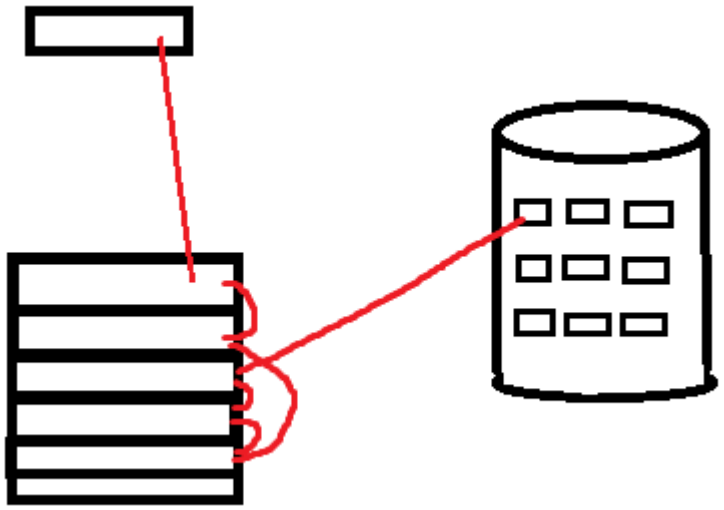
- **Asignación de lista encadenada**: Se tendrá un **puntero** al primero y dentro del sector se tendrá un **puntero al que sigue**, y el proxima un puntero del que le sigue y así… puede ser simple o doblemente encadenada.

  Un archivo será una lista enlazada de bloques dispersos por el disco,

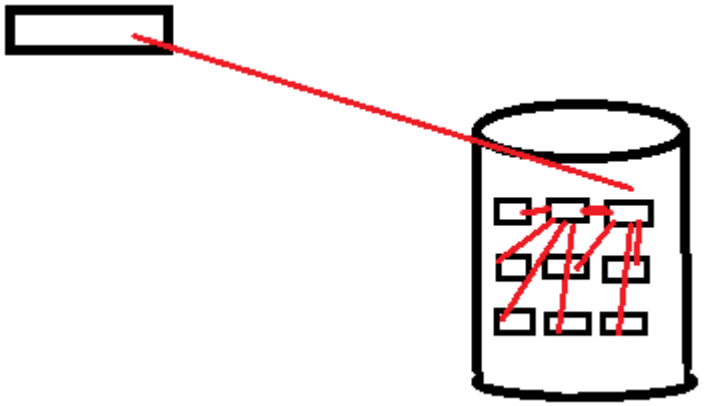
  Soluciona todos los problemas de asignación contigua.



- FAT (Fat allocation table): En vez de leer en el disco cada sector y su puntero, se tiene una **tabla en RAM** (más rápido) con dichos punteros.
  Se define una matriz que se guardará en memoria donde se guardará la l**ista encadenada de punteros**. Una entrada en la fat por cada sector del periférico.
  En el directorio se tendrá por cada archivo el **nombre** del archivo, **datos** y un **puntero al Fat**. Consumía demasiada memoria con discos mayor a 30 mb

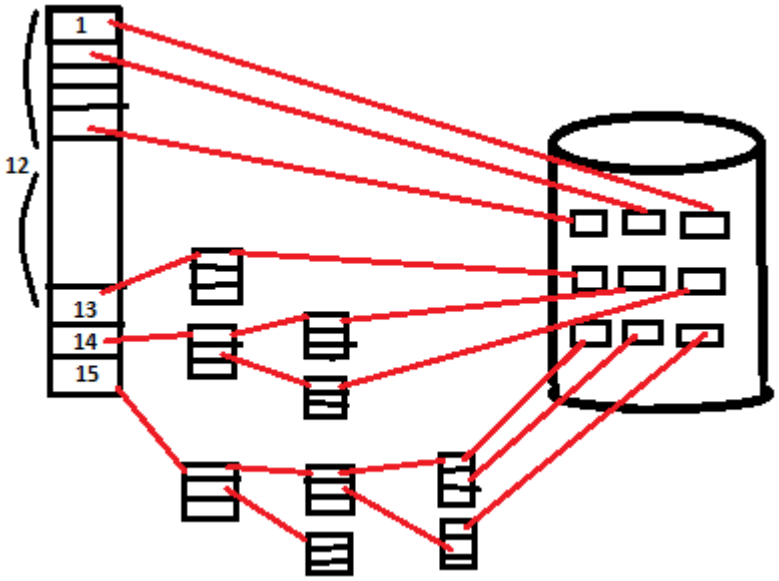


- **Indexado**: Dentro del directorio se tendrá un **puntero** que apunta a un sector del disco con un **conjunto de índices** que "mapean" sectores con archivos.
  Puede encadenar dos o más sectores "de conjuntos de índices" con los mismos punteros, en caso de quedarse corto con un solo sector, el último puntero apunta a otro sector de índices



- Sistema combinado, se tendrá dentro del directorio, los **datos del archivo** y además va a trabajar con una **cantidad de punteros**(se decidió usar 15), los primeros **12 eran punteros directos** (apuntaban a un sector del disco) , el puntero **13 apunta a un sector (1 nivel de indirección)**, que a su vez tiene

256 punteros, en el puntero **14 apuntará a un sector (2 niveles de indirección)** donde habrá 256 punteros y que cada uno tendrá 256 punteros, y el puntero **15 apunta a sector posee 3 niveles de indireccionamiento**. esto permite manejar archivos de hasta 4gb



(cada tabla indica un sector con punteros a otros sectores)

Dato: Esta forma de asignación permite discos duros de la orden de terabytes. Y se dejó de hacer disco con más capacidad que el orden de los terabytes ya que se implementó la tecnología RAID, que permite ver varios dispositivos como si fueran uno solo.

Administración del espacio libre

- **Vector de bits**(BitMap): un bit por sector del disco. 1 = usado / 0 = libre. Rápido, y eficiente para periféricos pequeños. no eficiente para discos grandes

- **Lista encadenada:** En el directorio se tienen un **número total de sectores libres** y un **puntero**. Se usan punteros que apuntan al proximo sector libre (según la capacidad pueden ser 128, 256, etc) que está libre. al liberarse sectores se encadenan. Se enlazan todos los bloques libres
- **Encadenada con agrupamiento y recuento**: se tiene **un sector** con **muchos punteros** a cada sector libre, (equivalente al sistema indexado de espacio usado). Cada puntero va a tener **dos datos**, la **dirección del sector libre** y **cuántos a continuación estarán libres**

- **¿De qué depende y cuál es el método de asignación de espacio de disco óptimo?**

    No existe un método óptimo. Cada método responde a distintas problemáticas.

Esto puede traer incoherencia de datos al ingresar de diferentes formas al mismo archivo (directo(lectura) y por caché (escritura)) al mismo tiempo

Buffer cache de archivos:
Utilización de una caché unificada para las llamadas al sistema read/write(para abrir y acceder un archivo), y el mapeo de memoria, para evitar incoherencias.
.Para evitar esto se creó el "cache unified", que hace que todos pasen a través de la caché(lectura y escritura por igual), es decir, cuando pasa del disco a la ram y viceversa.

Almacenamiento masivo

• La superficie del plato está dividida lógicamente en pistas circulares, las cuales se subdividen en sectores

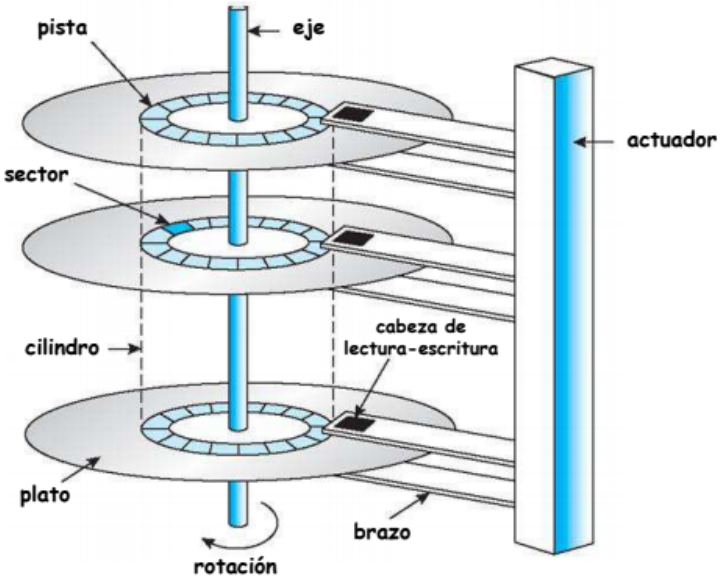
• El conjunto de pistas que se encuentran en una posición del brazo forman un cilindro

Dos caracteristicas claves:

Puede ser reescrito de manera directa, es posible leer un bloque, modificarlo y volverlo a escribir en el mismo lugar.

Se puede acceder directamente a cualquier bloque de informacion que contenga

Almacenamiento masivo



- **¿De qué manera se puede realizar una compactación para solucionar la fragmentación externa en un disco, cuyo método de asignación de espacio es contiguo?**

  Volcandolo a otro disco, borrando todo, y trayéndolo de nuevo, por ejemplo

RAID
Arreglo de discos que presenta los distintos dispositivos como uno solo.
Se pueden utilizar hasta 5 discos como una sola unidad.

Se tiene diferentes tecnologías en cuanto se forma el arreglo.

Mirroring: Es cuando diversos discos tienen repetida información, de tal manera que si alguno falla tenemos el otro para seguir trabajando. La desventaja principal es que la escritura es lenta porque se debe escribir en dos discos al mismo tiempo.

Stripping: El dato se va a fragmentar en uno o más discos, el acceso es mucho más rápido, pero como desventaja es que si uno falla se puede perder información

¿Que no es RAID ? cuando no tengo respaldo de información, ya sea enterminos de mirroring o striping

Niveles de Raid:

Van desde el 0 hasta el 6. Existen híbridos como el 0+1 o el 1+0

Los que más se usan son el 0,1,5 y el 1+0 y el 5+0

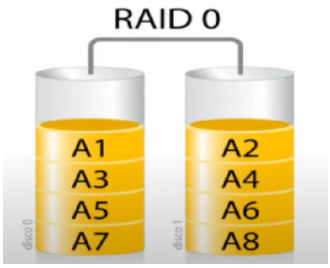
Los que no se usan son el 2, 3, 4 y 6

Raid 0:
Utiliza data Stripping, distribuye los distintos bloques de los datos entre los distintos discos que componen el RAID.

Si algun disco es de menor capacidad que los demás, se tomará al de menor capacidad como el tamaño total disponible de cada disco.

Ej 2 discos de 100Gb y uno de 80GB. Quedará como 1 sola unidad con la capacidad de (80+80+80= 240Gb)

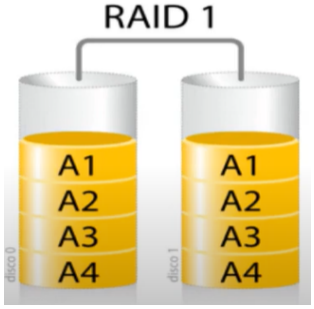
Ventajas: Rápido acceso a información
Desventaja: posible pérdida de información si algún disco falla.



Raid 1:
Utiliza el concepto de mirroring, de manera ideal se hace sobre cantidad par de discos.
Se hace una copia de bloque de datos en cada disco que conforma el Raid.

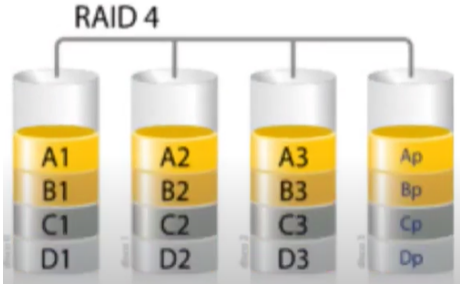
Ventaja:
- Si un disco re rompe se tiene los datos en los demás (replicación de datos)
- Hot swap: permite sacar o agregar un dispositivo en “caliente”

Desventaja:
- mucho más lento en escritura ya que se debe escribir en dos o más discos.



//Los siguientes raid solo se los nombra y se muestra la imagen pero no se los explica ya que no son usados

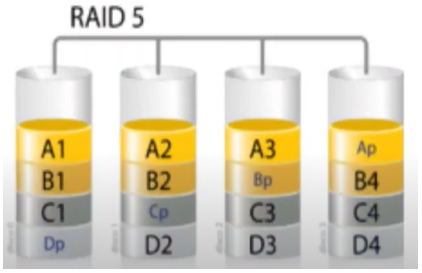
Raid 4:



Raid 5:

Se hace un stripping a nivel de bloque, se usará un bloque adicional que se llamara bloque de paridad para la recuperación de información, este bit de paridad estará distribuido entre los discos.
Funciona así: se tiene los sectores distribuidos (A1=1 A2=0 A3=1 Ap= 1). En caso de fallar el disco que contiene el sector A2, como estaba Ap=1, y solo queda A1=1 y A3=1, entonces A2 era = 0. ya que la paridad estaba seteada en 1

1 y 0 = 1 *///* 1 y 1 = 0 *///* 0 y 1 = 1 *///* 0 y 0 = 0 (utiliza compuertas XOR)



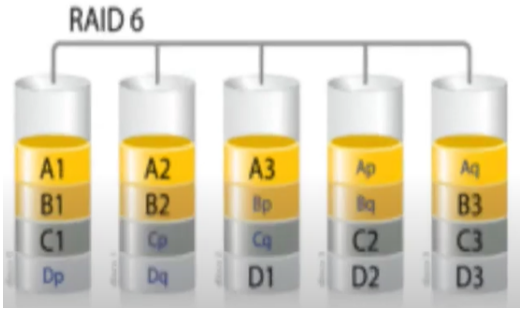
Ventaja: Se brinda “seguridad” (dice seguridad pero creo que sería consistencia) para recuperar los datos sin por eso, sacrificar mucho espacio (se sacrifica ¼ de capacidad)

- **Explique los problemas que surgieron al implementar RAID 4 y cómo se solucionaron**

El problema que tiene es que al estar toda la info de paridad en un solo disco, este se convierte en un cuello de botella.

La solución se da implementando RAID 5 que evita el uso potencial excesivo de un único disco de paridad.

Raid 6:



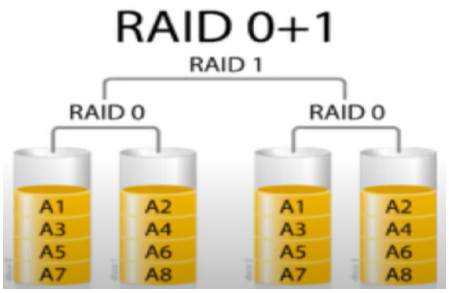
Prácticamente igual que el RAID 5, pero añade un segundo nivel de paridad, lo que nos permite que fallen hasta dos discos duros del RAID y poder sustituirlos. Si fallan 3, entonces toda la información del RAID se pierde.

Raid 0+1

Combina el raid 0 con el raid 1. Primero hace mirroring, y después hace stripping de cada par. Mínimo 4 disco

Ventajas: Combina ventajas. Seguridad en los datos y lectura mas rapida que un raid 0 tradicional

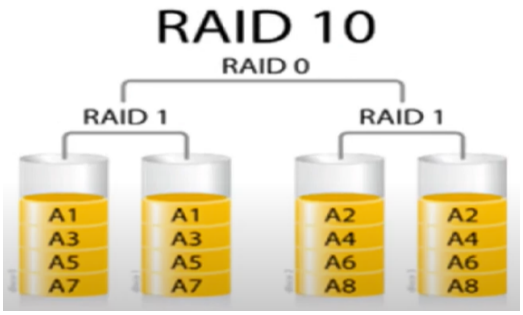
RAID 0+1



Raid 1+0

Combina raid 1 con el 0. Primero hace stripping y después mirroring.

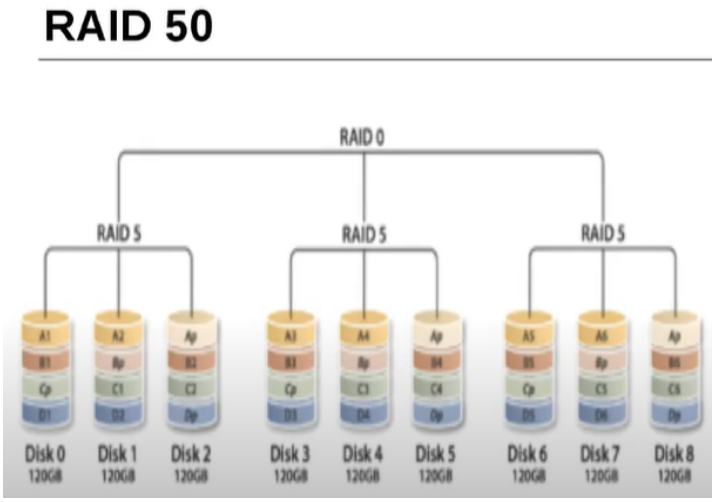
Ventaja: Es incluso más rápido que el 0+1, porque el stripping está a un nivel superior.



Raid 5+0

Se usa para hosteo de gran cantidad de información. combina raid 0 y raid 5. Usualmente se utiliza con motores de base de datos.

Ventajas: brinda confiabilidad, rapidez y poca pérdida de capacidad comparada con un 1+0

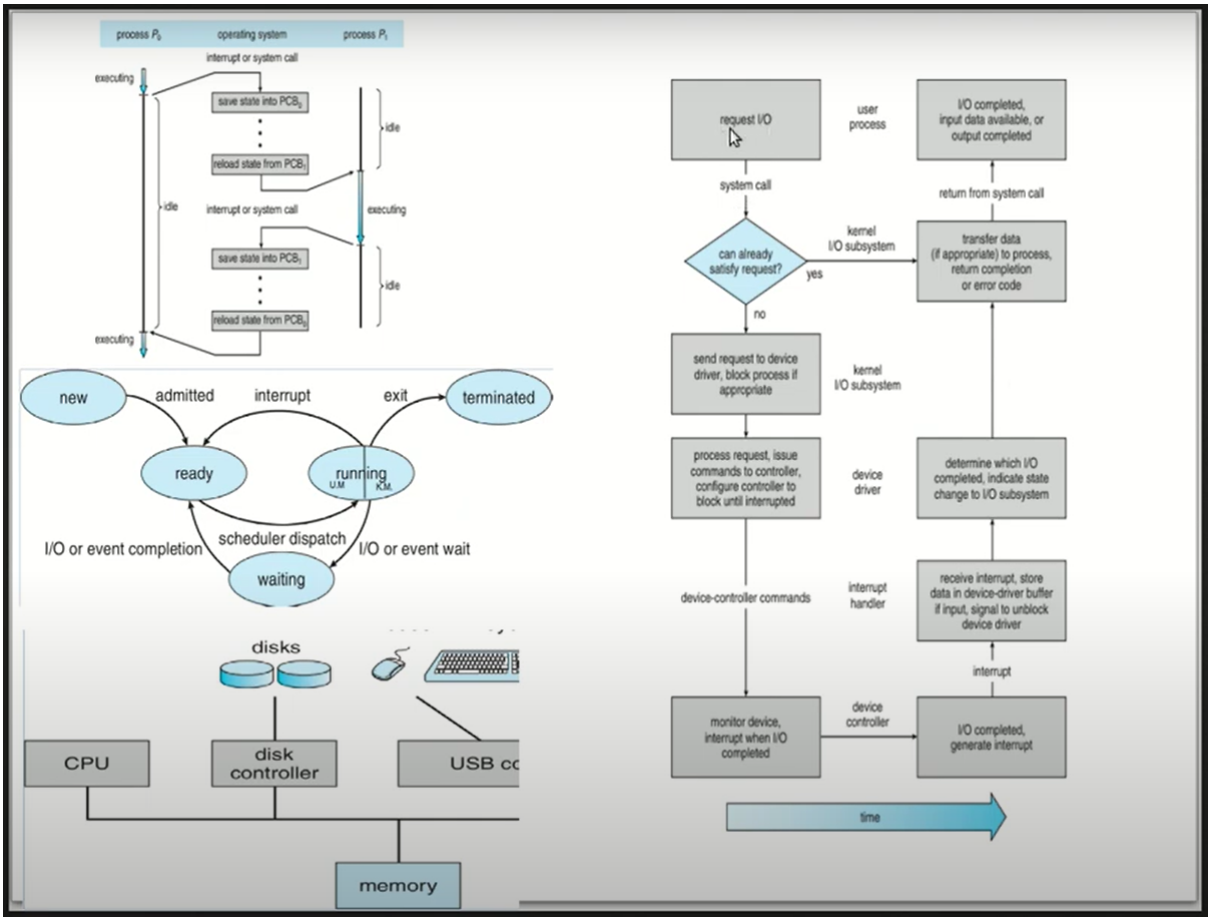


Dato: El raid puede ser por software o por hardware. El hecho por hardware tiene la particularidad de que es realizada con un dispositivo especializado para eso.

//////////V - Integración

Se tienen los siguientes diagramas:

- Diagramas de cambio de contexto
- Diagrama de 5 estados (Con modo usuario y modo kernel en ejecución)
- Estructura básica del equipo
- Secuencia de sucesos



Estado de partida :

Ejecutando un proceso de usuario, con varios procesos en la cola de listo. Este será el proceso 0 el cual tiene el control de la CPU

Acción:
El proceso pide una operación de entrada/salida a través de una system call

Para realizar la system call se debe cambiar a modo kernel(cambio de modo). sera ejecutada por el subsistema de E/S

La rutina de adm de system call via el subsistema de E/S Lo primero que hace es determinar si ya puede satisfacer la solicitud:

- En caso de que el dato se pueda obtener desde un registro de Cpu o esté en memoria caché, o puede ser que esté en la buffer cache(mmap() en vez de open()), se puede satisfacer la acción, no sería necesario hacer Context switching. Los datos se obtienen y continúa en estado RUNNING, es decir, mantiene el control de la CPU. Vuelve a modo usuario

- En caso contrario, la System call, en caso de ser bloqueante, lleva al proceso 0 al estado de WAIT, Se grabará el estado del proceso 0 en la PCB0. La system call enviará el pedido al driver.

El sub E/S pasa el control al driver , es decir, el control de la CPU lo tiene el SO, ejecuta las instrucciones del driver, y prepara instrucciones para la controladora (UC) , configura la controladora para que se bloquee hasta que termine.

Mediante el manejador de interrupciones, se envía a través del canal, los comandos para la controladora.
Al mismo momento pasa a ejecutar el planificador de corto plazo, seleccionará el próximo proceso, se carga la PCB del proceso 1 y este proceso 1 del estado READY pasa al estado RUNNING en modo usuario, es decir, la CPU pasa a control de proceso 1 (ESTAS ACTIVIDADES SON LO QUE PERMITEN LA CONCURRENCIA DE PROCESOS)

La controladora toma las instrucciones enviadas por el driver,las ejecuta y al completar la operación genera una interrupción, puede ser que en este momento acceda mediante DMA a la memoria cargando el buffer, después enviará solicitud de interrupción al canal.

El manejador de interrupciones toma la interrupción. Posteriormente el driver indica el cambio de estado al sub E/S. (La operación ya fue satisfecha)
A su vez se guarda el estado del proceso 1 en su PCB1 y el control de la CPU pasa al SO.
El proceso 1 pasa a estado LISTO (Suponiendo que se está en un sistema con desalojo).
Después pasa control al driver.

El. Se ejecuta el planificador de corto plazo, en este momento pueden suceder varias cosas:

- El proceso 0, al momento de estar en WAIT pudo haber sido swapeado en disco porque faltaba memoria, y para pasarlo a LISTO puede dar fallo de página. Entonces

en el momento de pasarlo a listo deberá ejecutar los algoritmos de fallo de página y cargar el proceso en memoria

- En caso de no estar swapeado simplemente el proceso 0 pasará a estado de LISTO

El proceso 0 recibe los datos buscados en la operación. Al pasar el proceso 0 a LISTO se carga la PCB0 y el control de la CPU pasa al proceso 0.