Sistemas Operativos

# Unidad 1

## Que es un sistema operativo?

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

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

* Inicialización: Prepara la maquina y la lleva a una estado tal que pueda ejecutar el primer trabajo. Existen el cold boot y el warm boot; difieren en las tareas de control realizadas previa carga del SO. Menos del 1% de las tareas del SO se destinan a esta función. A mediados de los 90s surge UEFI como sucesor de la BIOS.

Diagrama

Descripción generada automáticamente

La BIOS (Basic input output system) es un firmware que se ejecuta apenas se enciende la máquina. El mismo chequea todo el hardware (memoria disponible, procesador funcional, etc.) y por último va a buscar al MBR (master boot record) el loader del SO. El loader chequea el HW y arranca el sistema operativo.

* Abstracción o maquina extendida (interface hombre maquina): Los programas no deben tener que preocuparse de los detalles de acceso a hardware, o de la configuración particular de una computadora. El SO se encarga de proporcionar estas abstracciones ( Ej: la información organizada en archivos o directorios en uno o mucho dispositivos de almacenamiento). Facilita la comunicación con el usuario y acepta entradas de nuevos trabajos (ejecución de los nuevos programas). Típicamente se clasifican en: GUI, CLI, NUI (natural user interface). El SO no solo debe proveer las abstracciones necesarias sino que también, ninguno de sus usuarios puede evadir dichas abstracciones.
* Administración de recursos (memoria, espacio de almacenamiento, tiempo de procesamiento, etc.): Es la función principal de cualquier SO. El SO tiene a su disposición una gran cantidad de recursos y los diferentes procesos compiten por ellos. El SO puede implementar políticas que los asignen de forma efectiva y acorde a las necesidades establecidas para dicho sistema. Más del 70% del código está dedicado a esta función. Si el SO estableció determinada política de asignación de recursos, debe evitar que el usuario exceda las asignaciones aceptables, sea durante el curso de su uso normal, o incluso ante patrones de uso oportunista.
* Aislamiento (seguridad): En un sistema multiusuario y multitarea cada proceso y cada usuario no tendrá que preocuparse por otros que estén usando el mismo sistema. Para implementar este aislamiento el SO utiliza hardware específico para dicha protección. Su objetivo es el de garantizar la integridad de los Recursos y de los Procesos, como también validar los usuarios en el sistema. Originalmente, la seguridad se refería únicamente a la integridad del propio sistema operativo y la protección entre usuarios. Los mecanismos fundamentales para lograr esto son: Modo dual de ejecución de procesador (modo usuario y modo kernel), el cual cambia según el estado de un bit que indica el modo de operación. Juego de instrucciones diferenciadas del Procesador (instrucciones ordinarias o del usuario e instrucciones privilegiadas, que son ejecutadas solo por el SO). Cuando se quieren ejecutar instrucciones privilegiadas el SO pasa a modo kernel y ejecuta estas instrucciones. Al ejecutar en modo kernel también se liberan las restricciones de memoria. La mayoría de las funciones del SO requieren del modo kernel para ejecutarse.

Actualmente muchos sistemas incluyen herramientas para mitigar ataques (malware, exploits), por ejemplo con la automatización de actualizaciones, incorporación de software antimalware, application firewalls, etc.

Si el SO ofrece separación entre los datos, procesos y recursos de sus distintos usuarios, ninguno de ellos debe tener acceso a la información que otro haya marcado como privada.

#### Tipos de SO:

* Monousuario: un solo usuario
* Multiusuario: mas de un usuario trabajando simultáneamente con la computadora.

#### Según la cantidad de procesos que soporta:

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

#### Según la cantidad de procesos que ejecutan concurrentemente:

* Monoprogramados: Solo se puede ejecutar un proceso a la vez. Recién cuando éste finalice, se puede ejecutar otro proceso. Se los conoce también como SOs monotarea.
* Multiprogramados: También llamados multitarea o multitask, se refiera a SOs capaces de maximizar el uso del procesador. Cuando un proceso se encuentra haciendo uso de algún dispositivo, se otorga el procesador a otro proceso.

La ejecución concurrente no es lo mismo que la ejecución paralela. Para esto último se requieren 2 o más procesadores y un SO capaz de administrarlo.

#### Según sus aplicaciones:

* De propósito general: Son los que proporcionan una amplia gama de servicios y deben adaptarse a cualquier ambiente, tipo de aplicaciones, modos de operación, dispositivos, etc.
* De propósito especial: Construidos a medida debido a arquitecturas especiales o aplicaciones con requerimientos especiales como control de proceso industriales. Históricamente se clasifican como:

Sistemas de tiempo real: Debe garantizar la respuesta a eventos externos dentro de los límites de tiempo preestablecidos.

Sistemas tolerantes a fallas: se utiliza en aplicaciones donde se debe proveer un servicio continuo. Se suele utilizar un conjunto de redundancias en recursos y chequeos internos. El SO detecta y corrige errores (que no son de hardware). Sistemas espaciales, sistemas de seguridad en el área nuclear, base de datos online.

Sistemas virtuales: SOs capaces de administrar y gestionar otros SOs que ejecutan bajo su órbita en forma concurrente usando el mismo HW. Nacen en los 60s, posteriormente esta tecnología cae en desuso, ya que el HW se hacia mas accesible económicamente. Hacia fines de los 90s vuelve a surgir el concepto y hoy en día prácticamente todos los centro de datos utilizan virtualización.

## Evolución de los SO

### Proceso por lotes (batch processing)

En los 50 aparecieron los dispositivos perforadores/lectores de tarjetas de papel. Los programadores entregaban su lote de tarjetas perforadas a los operadores quienes las alimentaban a los dispositivos lectores, que lo cargaban en memoria, iniciaban y monitoreaban la ejecución y producían resultados. Los sistemas monitores se fueron sofisticando al implementar protecciones que evitaran la corrupción de otros trabajos, o que entraran en un ciclo infinito, estableciendo timers que interrumpirían la ejecución de un proceso si este duraba más allá del tiempo estipulado.

### Sistemas en lotes con dispositivos de carga (spool)

Spool: mecanismo de entrada salida que permitía que una computadora de propósito específico (más económica y limitada), leyera las tarjetas y las fuera convirtiendo a una cinta magnética (mucho más rápido), luego la computadora central procesara los datos. A su vez, la computadora central guardaba sus resultados en cinta para que sus equipos especializados la leyeran e imprimieran para el usuario solicitante.

### Sistemas multiprogramados

Estos dispositivos buscaban maximizar el tiempo de uso efectivo del procesador ejecutando varios procesos al mismo tiempo. Esto trajo un fuerte cambio en el HW. Resultó necesario que apareciera la infraestructura de protección de recursos: un proceso no debe sobrescribir espacio de memoria de otro (ni código, ni datos), mucho menos el espacio del monitor. Está protección se encuentra en la Unidad de Manejo de Memoria (MMU). Ciertos dispositivos requieren bloqueo para ofrecer acceso exclusivo / único: cintas e impresoras. Se multiplexan los procesos para generar un espejismo de multiprocesamiento.

### Sistemas de tiempo compartido

60s. Se extiende la multitarea para convertirse en sistemas interactivos y multiusuario. Aparecen las terminales, el programador ejecuta directamente los cambios y somete al programa a la ejecución inmediata. Mientras el programador editaba o compilaba su programa, la computadora seguía calculando lo que otros procesos requerían.

Cambia el control sobre la multitarea:

* Multitarea cooperativa o no apropiativa (Cooperative multitasking): La implementación de sistemas multiprogramados: cada proceso tenía control del CPU hasta que este hacía una llamada al sistema (mediante la llamada yield). Un cálculo largo no era interrumpido por el SO, por lo que un error de programador podía congelar la computadora.
* Multitarea preventiva o apropiativa (Preemptive multitasking): En los sistemas de tiempo compartido el reloj del sistema interrumpe periódicamente a diversos procesos, transfiriendo forzosamente el control nuevamente al SO. Éste puede elegir otro proceso para continuar con la ejecución.

Fueron naciendo paulatinamente las abstracciones que se conocen hoy en día como los conceptos de archivos y directorios.

La velocidad de cambio entre una tarea y otra es mucho más rápido (ilusión de uso exclusivo del sistema).

Distintos tipos de proceso pueden tener un distinto nivel de importancia. Esto requiere la implementación de diversas prioridades para cada uno de estos.

## Computadoras personales:

Las primeras eran distribuidas sin SO o lenguajes de programación; la interfaz primaria para programar era mediante llaves y para recibir resultados se utilizaban bancos Leds.

Luego aparecieron las computadoras personales con salida de video y entrada por un teclado, lenguaje BASIC. Permite a los usuarios gestionar recursos (unidades de cinta, pantalla posicionable unidades de disco, impresoras, modem, etc.). Llevaban un proto sistema operativo.

A principio de los 80s aparecen las computadoras orientadas a la oficina más que al hobby. Separan el entorno de desarrollo en algún lenguaje de programación del entorno de ejecución. El papel del sistema operativo es administrar los archivos de las diversas aplicaciones mediante una sencilla interfaz de línea de comando y lanzar las aplicaciones que el usuario seleccionaba. IBM con su PC acaparan prácticamente el 100% del mercado. En sus primeros años muchos programas se ejecutaban directamente sobre el hardware, arrancando desde la BIOS y sin emplear el SO.

A mediados de los 80s comenzaron a aparecer computadoras con interfaces usuario gráficas (GUI, textsguis) basadas en el paradigma WIMP (Windows, icons, menus, pointers) que permitían la interacción con varios programas al mismo tiempo ( no necesariamente multitareas). Los problemas de protección y separación entre procesos concurrentes comenzaron a hacerse evidentes. Se empezaron a compartir recursos.

A mediados de los 90s las dos arquitecturas que se convirtieron en predominantes fueron las derivadas de IBM y Apple Macintosh. Los sistemas operativos para ambas plataformas fueron respondiendo a las nuevas características del HW: en las IBM Microsoft Windows, que a partir de 2003 el núcleo de Windows en más amplio uso fue reemplazado por un desarrollo hecho de inicio como un sistema operativo completo y ya no como un programa bajo MS-DOS. Por el lado de Apple MacOS X, sistema basado en un núcleo Unix BSD, cobre el microkernel Mach. El SW libre también entró en escena (implementaciones distintas de sistemas tipo Unix, principalmente Linux).

## Características diferenciadoras entre dispositivos móviles y PCs.

La diferencia entre el segmento móvil y el cómputo tradicional no está en el SO mismo, sino en sus capas superiores. Sin embargo, la diferencia va mucho más allá de un cambio en la interfaz de usuario, las características de estos dispositivos indudablemente determinan cuestiones de fondo:

* Almacenamiento en estado sólido: Los dispositivos móviles operan con almacenamiento en estado sólido. Son menos frágiles que los no sólidos y además esto implica que sus SO no emplean memoria virtual. No pueden, por tanto, mantener en ejecución programas que excedan del espacio real de memoria con que cuente el sistema.
* Multitarea pero monocontexto: Al carecer los dispositivos móviles de memoria virtual, el SO se ve obligado a limitar el número de procesos interactivos en ejecución. Además, las interfaces usuario empleadas por los sistemas móviles abandonan el modelo de interacción WIMP así como la metáfora del escritorio, para volver a la de un solo programa visible en todo momento. El usuario no solicita la finalización de los programas, estos van siendo lanzados, utilizados y cuando se cambia de programa, si los anteriores caben en memoria son mantenidos abiertos para evitar demoras de volver a inicializar. El programa gestiona cuales programas serán finalizados y evacuados de la memoria al llegar a determinados umbrales.
* Consumo eléctrico: El ahorro de consumo eléctrico tiene dos principales vertientes, por un lado el desarrollo de HW más eficiente energéticamente, con independencia del modo en que opere y por el otro, la creación de mecanismos por medio de los cuales un equipo de cómputo pueda detectar cambios en el patrón de actividad y este reaccione reduciendo su demanda. La verdadera diferencia es la frecuencia de los cambios de estado. Un servidor o computadora de escritorio tiene un solo evento constante, una computadora portátil debe adoptar diferentes perfiles dependiendo de si está conectada a la red eléctrica u operando por batería, o si tiene la tapa abierta o cerrada. En el cómputo móvil, los eventos son muchos y muy distintos. En primer lugar, los dispositivos operan bajo una filosofía de siempre encendido (siempre encendido y pendiente al entorno). Los desarrolladores de las aplicaciones buscan patrones eficientes, fácil suspensión y minimizando la necesidad de despertar el hardware.
* Entorno cambiante: Los centro de datos, las computadoras de escritorio e incluso las portátiles, tienen una forma de operación bastante estable. Un usuario puede mantener por largos periodos su configuración de consumo energético. Las interfaces y direcciones de red son típicamente estables. El formato de la pantalla también es estable. El dispositivo móvil debe ser más enérgico en sus cambios de perfil de energía, respondiendo en un entorno cambiante (luminosidad, conectividad y sincronización, detención de procesos y ágil rotación de pantalla).
* El jardín amurallado (o plataforma cerrada): Los dispositivos móviles buscan satisfacer un mercado mucho mayor al de los entusiastas del cómputo. Tanto Apple como Google mediante sus modelos de distribución y autorización de SW se reservan el derecho de aprobar o eliminar cualquier aplicación de sus market places.

## Arquitectura de los sistemas operativos

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

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

Diagrama

Descripción generada automáticamente

* Sistemas de microkernel: El núcleo del SO se mantiene en el mínimo posible de funcionalidad. Poseen una lógica mas limpia y resulta mas simple el reemplazo de componentes. Pueden auto repararse con mayor facilidad dado que en caso de fallar uno de los componentes, el núcleo puede reiniciarlo o incluso reemplazarlo.

Diagrama

Descripción generada automáticamente

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

Diagrama

Descripción generada automáticamente

## Unidad de Procesamiento

Es la parte fundamental de todo sistema de cómputo. Esta es la encargada de ejecutar tanto los programas del usuario como el sistema operativo en si mismo. Las funciones del SO respecto a la unidad de procesamiento son:

* Inicialización: Luego de ser cargado el SO debe realizar varias tareas de inicialización como habilitar las interrupciones de HW y SW, configurar el sistema de memoria virtual, etc.
* Atender las interrupciones y excepciones: la UP puede encontrar una situación que no pueda resolver por sí misma (instrucción o dirección invalida, una división por cero, etc.) ante lo cual le pasa el control al SO para que este trate o resuelva la situación.
* Multiplexación: En un sistema multiproceso, el SO es el encargado de administrar la unidad de procesamiento dando la ilusión a los procesos que se están ejecutando en forma exclusiva.

## Jerarquía de almacenamiento

Una computadora Von Neumann significa básicamente que es una computadora de programa almacenado en la memoria primaria ( se usa el mismo almacenamiento para el programa que está siendo ejecutado y para sus datos, sirviéndose de un registro especial para indicar al CPU cual es la dirección en memoria de la siguiente instrucción a ejecutar. Con el paso del tiempo (entre 1975 y el 2000) los procesadores aumentaron su velocidad 1000 veces pero la memoria aumento su velocidad a un ritmo mucho menor. Una respuesta parcial a esto es la creación de una jerarquía de almacenamiento, yendo de una pequeña área de memoria mucho mas cara pero extremadamente rápida y hasta un gran espacio de memoria muy económica, aunque mucho más lenta.

Diagrama

Descripción generada automáticamente

Los registros son parte integral del procesador y la memoria está solo a un paso de distancia (el procesador puede referirse a ella directamente, de forma transparente, indicando la dirección desde un programa). Para efectos prácticos, el caché no se maneja explícitamente: el procesador no hace referencia directa a él, sino que es manejado por los controladores de acceso a memoria. Y por último, el acceso o modificación de cualquier dato almacenado en el disco requiere en primer término de la transferencia a la memoria, y solamente cuando ésta haya terminado, el llamado a las rutinas que guardan los datos.

* Registros: la memoria más rápida de la computadora. Están ubicados en cada uno de los núcleos de cada CPU. Las arquitecturas de tipo RISC solo permiten la ejecución de instrucciones entre registros (excepto claro, las de carga y almacenamiento a memoria primaria). El compilador busca realizar muchas operaciones que deben ocurrir reiteradamente, donde la rapidez es fundamental, con sus operadores cargados en los registros. El estado del DPU a cada momento está determinado por el contenido de los registros. El contenido de la memoria obviamente debe estar sincronizado con lo que ocurre dentro de este ( pero el estado actual de la CPU, lo que está haciendo, las indicaciones respecto a las operaciones recién realizadas que se deben entregar al programa en ejecución están todas representadas en los registros. Se debe mantener esto en mente cuando posteriormente se habla de todas las situaciones en que el flujo de ejecución debe ser quitado de un proceso y entregado a otro.

## Interrupciones

### Interrupción de HW externo

Se llama interrupción a la detención del programa en ejecución debido a una condición externa al procesador, es decir, que este es forzado a reconocer la ocurrencia de un evento en el sistema. Cuando un dispositivo requiere atención debe avisar al procesador, mediante una línea (IRQ interruption request) conectada al controlador de interrupciones, un HW dedicado a su tratamiento. A partir del numero de IRQ se busca la dirección de memoria de la rutina de atención que se debe ejecutar para atender la petición (vector de interrupciones).

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

IRQ: solicitud de atención que hacen los dispositivos.

RAI: rutina de atención de interrupciones.

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

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

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

### Clasificación según prioridad

Las interrupciones pueden organizarse por prioridades, de modo que una interrupción de menor jerarquía no interrumpa a una más importante, dado que las interrupciones muchas veces indican que hay datos disponibles en algún buffer, el no atenderlas a tiempo podría llevar a la perdida de datos. Hay un número limitado de interrupciones definidas para cada arquitectura. Las interrupciones son generadas por el controlador de canal en que son producidas. El SO puede elegir ignorar (enmascarar) ciertas interrupciones, pero hay algunas que son no enmascarables.

* No enmascarables: Son las de mayor prioridad del sistema. Cuando se detecta una interrupción de este tipo, el SO detiene lo que está haciendo y se ocupa de atender la interrupción. Este tipo de interrupción puede detener hasta la ejecución de rutinas modo kernel. En general se utilizan para la notificación de errores irrecuperables por parte de algún HW.
* Enmascarables: Son interrupciones de menor prioridad (dentro de las interrupciones, pero siempre tienen mayor prioridad a cualquier proceso de usuario). Este tipo de interrupciones puede ser interrumpida por una interrupción de mayor prioridad.

### Clasificación según su origen

* Software: se denominan así a las llamadas del sistema (syscalls) que hacen los procesos. Si bien no son eventos que el procesador debe reconocer, se categorizan así porque en definitiva se interrumpe el código de usuario para dar lugar a la atención de dicha llamada al SO.
* Hardware: Interrupciones generadas por algún componente físico del sistema. Se subdividen en: Internas: se producen dentro del procesador (división por cero, código de operación invalido). Externas: fuera del entorno del procesador (finalización de I/O, falta de papel, error de dispositivos).

## Multiprogramación / Multiprocesamiento

Diagrama

Descripción generada automáticamente

### Multiprocesamiento simétrico

* Uniform Memory Access (UMA): Varios procesadores compartiendo la memoria con igualdad de condiciones. Utilizan un BUS compartido, por lo que se debe administrar el uso.
* Non-Uniform Memory Access (NUMA): cada procesador tiene afinidad con bancos específicos de memoria. Pueden acceder a su memoria local más rápido que a la memoria de otros procesadores o compartida. El objetivo es mejorar la performance.

### Multiprocesamiento asimétrico

Varios procesadores con objetivos y hasta arquitecturas diferentes. Son considerados coprocesadores. Por ejemplo, los que se incluyen en las placas de video avanzadas (GPU).

## Cómputo distribuido

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

* Clústers: Computadoras conectadas a una red local de alta velocidad, cada una ejecutando su propia instancia de SO. Se ven como un único equipo de cómputo.
* Grids: Computadoras heterogéneas distribuidas geográficamente e interconectadas a una red. Se diseñan para adaptarse a enlaces de baja velocidad.
* Computo en la nube: se refiere a la terciarización de servicios. La implementación de los servicios deja de ser relevante. Se aplican conceptos como :

SaaS: se ofrece la aplicación completa.

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

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

AUTOEVALUACION

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

No amplía el set de instrucciones de la maquina sino que permite usar las instrucciones de modo protegido

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

La rutina de verificación de recursos después de un warn boot no lo hace.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

# UNIDAD 2

## De programas a procesos

### Programa

Conjunto ordenado de instrucciones que pretenden resolver un problema.

### Instrucción

Unidad de ejecución que dura un tiempo finito y se ejecuta sobre un procesador (es indivisible, no se descompone ni se interrumpe y se dice que ejecuta atómicamente).

### Proceso

Porción de un programa cargado en Memoria Central al cual se le asocia su contexto de ejecución (run time environment) mediante una estructura de datos llamada vector de estado o Bloque de Control del Proceso (Process Control Block – PCB)

Consiste en una secuencia de acciones llevadas a cabo a través de las instrucciones cuyo resultado consiste en proveer alguna función del sistema

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

Diagrama

Descripción generada automáticamente

### Administración de procesos

Los datos sobre el proceso se guardan en una estructura de datos llamado vector de estado o PCB (Process control block). El PCB se guarda para cada proceso la información necesaria para reanudarlo (cuando es suspendido o desalojado del uso del procesador) y otros datos. El SO mantiene para cada proceso un bloque de control o Process control block (PCB). La información contenida en el PCB varia de SO en SO. El PCB de los procesos activos está en un área de la memoria específica para el PCB administrada por el SO. Desde memoria se copia una imagen del proceso que se carga sobre CPU y se ejecuta.

### Contenido del PCB

* Identificación (PID única en el sistema): identifica unívocamente el proceso en el sistema.
* Identificadores varios del proceso (identificador del dueño, padre PPID, hijos, etc.). Todos los procesos tienen un padre, al nivel más alto está el SO.
* Estado (ejecutando, listo bloqueado)
* Program counter (PC): guarda la dirección de la próxima instrucción, es un registro del procesador.
* Registro de CPU (acumuladores, program status Word).
* Información para la planificación (Ej: prioridad): dependiendo de la metodología que se utilice para la planificación se guarda determinada información desde el procesador al PCB.
* información para administración de memoria (Ej: registros base y límite). También depende la metodología de administración es lo que se guarda.
* Información de I/O: dispositivos y recursos asignados al proceso, archivos abiertos en uso, etc.
* Estadísticas y otros: tiempo real y tiempo de CPU usado, etc.
* Privilegios (Ej: quien es el owner)
* Otros objetos vinculados al proceso.

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

### Estados

Diagrama

Descripción generada automáticamente

Diagrama

Descripción generada automáticamente

#### Estados de un proceso

* Nuevo: Se solicitó al SO la creación de un proceso y sus recursos y estructuras están siendo creadas.
* Listo: Está listo para iniciar o continuar su ejecución pero el sistema no le ha asignado un procesador.
* En ejecución: el proceso está siendo ejecutado en este momento. Sus instrucciones están siendo procesadas en algún procesador.
* Bloqueado: en espera de algún evento para poder continuar su ejecución (aún si hubiera un procesador disponible, no podría avanzar).
* Zombi: El proceso finalizado su ejecución pero el SO debe realizar ciertas operaciones de limpieza para poder eliminarlo de la lista.
* Terminado: El proceso terminó de ejecutarse, sus estructuras están a la espera de ser limpiadas por el SO.

#### Razones de un cambio de estado de proceso

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

#### Estados activos:

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

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

#### Estados inactivos:

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

#### Las colas de Estados de los procesos:

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

#### Transiciones de estado

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

Cada uno de estos cambios se denomina transición de estado. Estas transiciones pueden ser las siguientes:

* Comienzo de la ejecución
* Paso a estado listo o preparado
* Paso a estado a ejecución
* Paso a estado bloqueado
* Paso a estado suspendido bloqueado
* Paso a estado suspendido listo
* Finalización

### Cambio de contexto

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

### Cambio de proceso

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

Un cambio de proceso implica un cambio de contexto. La mayoría de las veces el cambio de contexto se da con un cambio de proceso pero no siempre. Cuando tengo que hacer un cambio de contexto entre 2 procesos, se carga todo el PCB. En cambio cuando tengo que atender una rutina no se utilizan todos los registros del procesador. Entonces libero algunos registros y el cambio de contexto es menor (hard switch).

### Cambio de modo de ejecución

Para ejecutar un array, para hacer un cambio de proceso o de contexto tengo que pasar a modo kernel. Una vez q se realiza el cambio y se empieza a ejecutar el nuevo proceso se pasa a modo usuario.

### Creación de procesos

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

#### Existen dos tipos de creación

* Jerárquica: cada proceso que se crea es hijo del proceso creador y hereda el entorno de ejecución de su padre.

1. El padre continúa ejecutando concurrentemente (paralelo) con sus hijos
2. El padre espera a que todos sus hijos hayan terminado y luego sigue el.

* No jerárquica: Cada proceso creado por otro proceso se ejecuta independientemente de sus creador en un entorno diferente (no es muy común).

#### Motivaciones o razones para crear un proceso:

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

#### Pasos del SO al crear un proceso:

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

#### Pasos cuando un proceso muere:

* Desaparece el PCB
* Recursos comunes son liberados
* Recursos locales son eliminados

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

### Trabajos, procesos y threads

* Trabajo (Job): En un sistema por lotes se habla de tareas, Una tarea requiere mucha menos estructura, típicamente basta con guardar la información relacionada con la contabilidad de los recursos empleados. Una tarea no es interrumpida en el transcurso de su ejecución (en desuso actualmente).
* Un proceso (Process) es la imagen en memoria de un programa junto con la información relacionada con el estado de su ejecución.
* Un Hilo o Hebra (thread) también llamado proceso liviano, es un trozo o sección de un proceso que tiene sus propios registros, pila, program counter y puede compartir la memoria con todos aquellos threads que forman parte del mismo proceso.

### Hilo o Hebra (Threads)

* También llamado proceso liviano, debido a que mantiene la estructura de un proceso con su PCB pero también de otra estructura más pequeña llamada TCB (Thread control block) que contiene una información reducida del PCB, lo que hace que se ejecute más eficientemente.
* En sí un proceso es igual a uno o más tareas (tasks) cada una con su Hilo (thread) asociado.
* Un thread es una unidad elemental de uso de CPU.
* Cada hilo posee TID (Thread Identifier) un Contador de programa (PC – Program Counter), un juego de registros de CPU (Register Set) y una Pila (Stack).
* En muchos sentidos, los hilos son como pequeños miniprocesos.
* Cada Thread se ejecuta en forma estrictamente secuencial compartiendo la CPU de la misma forma que lo hacen los procesos.
* Solo en un multiprocesador se pueden realizar en paralelo.
* Los hilos pueden crear hilos hijos y se pueden bloquear en espera de llamadas al sistema, al igual que los procesos regulares.
* Mientras un hilo está bloqueado se puede ejecutar otro hilo del mismo proceso. (Depende de la implementación, hay dos tipos de hilos)
* Puesto que cada hilo tiene acceso a cada dirección virtual (comparten un mismo espacio de direccionamiento, comparten PCB), un hilo puede leer, escribir o limpiar la pila de otro hilo.
* No existe protección entre hilos debido a que es imposible y no es necesario ya que generalmente cooperan entre si la mayoría de las veces
* Aparte del espacio de direcciones, comparten el mismo conjunto de archivos abiertos, procesos hijos, relojes, señales, etc.
* Los procesos livianos dentro de un mismo proceso pesado no son independientes, pues cualquiera puede acceder toda la memoria correspondiente al proceso. Podrían comunicarse entre sí sin ningún proceso de IPC).
* Si un proceso está bloqueado no necesariamente implica que los hilos estén bloqueados. Esto depende de la implementación.

#### Ventajas respecto a los procesos

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

#### Implementación de Hilos (Threads)

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

Hilos a nivel de usuario (ULT)

Todo el trabajo del hilo es manejado por la aplicación, el kernel no se entera de la existencia de los hilos.

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

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

El kernel no se entera de lo anterior, continúa con la planificación del proceso como unidad y le asigna un solo estado de ejecución (listo, corriendo, etc.).

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

Ventajas

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

Desventajas

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

##### Hilos a nivel de Kernel (KLT)

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

Ventajas

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

Desventaja

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

#### Hilos a nivel de proceso (PLT)

En desuso.

#### Estado de los threads

Dentro de los diferentes estados, los Threads se encuentran en la parte correspondiente al short term schedule.

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

Diagrama

Descripción generada automáticamente

#### Uso de los hilos

* Estructura servidor trabajador: Existe un hilo en el servidor que lee las solicitudes de trabajo en un buzón del sistema, examina éstas y elige a un hilo trabajador inactivo y le envía la solicitud. El servidor despierta entonces al trabajador dormido ( un signal al semáforo asociado)
* Estructura de equipo: Todos son iguales y cada uno obtiene y procesa sus propias solicitudes.
* Estructura de entubamiento (pipeline): El primer hilo genera ciertos datos y los transfiere al siguiente para su procesamiento. Los datos pasan de hilo en hilo y en cada etapa se lleva a cabo cierto procesamiento. Esta puede ser una buena opción para el modelo productor / consumidor, no así para los servidores de archivos. Posiblemente en una tarea estrictamente secuencial no sea ideal el uso de threads, puesto que no saco ventaja del multiprocesamiento, debo hacer un cambio de contexto por cada hilo y se pierde tiempo. En este caso posiblemente sea mejor el proceso pesado.

## UNIDAD 3

Inanición

Es cuando un proceso se queda esperando indefinidamente por algo que nunca va a ocurrir.

### Overhead

Tiempo que utiliza el SO en tareas administrativas ( por ej. Ejecutar un algoritmo de planificación)

### Concepto de planificación

La planificación de procesos se refiere a como determina el sistema operativo al orden en que se irá cediendo el uso del procesador a los procesos que lo vayan solicitando, y a las políticas que empleará para que el uso que den a dicho tiempo no sea excesivo respecto al uso esperado del sistema.

#### Objetivos de la planificación

* Ser justo: tratar de igual manera a los procesos de determinadas características, y nunca postergar indefinidamente uno de ellos (inanición o starvation)
* Maximizar el rendimiento: dar servicio a la mayor parte de procesos por unidad de tiempo. De esto también depende el tipo de algoritmo que estos utilizando.
* Ser predecible: Un mismo trabajo debe tomar aproximadamente la misma cantidad de tiempo en completarse independientemente de la carga del sistema.

Esto no es tan lineal ni sencillo, esto es relativo.

* Minimizar la sobrecarga: El tiempo que el algoritmo pierda en burocracia (overhead) debe mantenerse al mínimo, dado que éste es el tiempo de procesamiento útil perdido.
* Equilibrar el uso de recursos: Favorecer a los procesos que empleen recursos subutilizados, penalizar a los que peleen por un recurso sobre utilizado causando contención en el sistema.
* Evitar la postergación indefinida (starvation): Aumenta la prioridad de los procesos más viejos (técnicas de aging), para favorecer que alcancen a obtener algún recurso por el cual estén esperando.
* Favorecer el uso esperado del sistema: En un sistema con usuarios interactivos, maximizar la prioridad de los procesos que sirvan a solicitudes iniciadas por éstos ( aún a cambio de penalizar a los procesos de sistema).
* Dar preferencia a los procesos que podrían causar bloqueo: Si un proceso de baja prioridad está empleando un recurso del sistema por el cual más procesos estén esperando, favorecer que éste termine de emplearlo más rápido.
* Favorecer a los procesos con un comportamiento deseable: Si un proceso causa muchas demoras (por ejemplo, atraviesa una ráfaga de entrada / salida que le requiere hacer muchas llamadas a sistema o interrupciones), se le pueden penalizar porque degrada el rendimiento global del sistema.
* Degradarse suavemente: Si bien el nivel de utilización del procesador es al 100%, es imposible mantener siempre a este nivel. Un algoritmo puede buscar responder con la menor penalización a los procesos preexistentes al momento de exceder este umbral.

#### Tipos de planificación

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

##### A largo plazo

Era frecuente en los sistemas por lotes. Decide que procesos (o trabajos) serán iniciados. Se ejecuta periódicamente una vez cada varios segundos, minutos o incluso horas. En los sistemas modernos ya no se utiliza más debido a que el usuario es quien elige qué procesos se ejecutarán. Se encarga de pasar los procesos del estado de nuevo al estado de listo.

##### A mediano plazo

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

#### A corto plazo

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

##### A extralargo plazo

Esta fuera del SO ( la hace el administrador del sistema) pero va a afectar el tiempo de respuesta de los procesos. Como va a priorizar los procesos o que tiempo de respuesta le dará a cada procesos.

### Relación entre los planificadores y los estados de los procesos.

El planificador de largo plazo admite nuevos procesos. Se encarga de la transición del estado de nuevo a listo.

El planificado de mediano plazo maneja las transiciones entre estados suspendidos y estados activos y viceversa.

El planificador de corto plazo administra las transiciones entre listo y ejecutando y bloqueado.

### Tipos de procesos

* CPU bound u Orientados al uso de la CPU

Los que típicamente realizan mucho computo interno y su ejecución esta típicamente alternada por ráfagas u bursts.

* I/O bound u Orientados a E/S

Los que se centran su atención en transmitir datos desde o hacia los dispositivos externos

* Procesos largos

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

* Procesos cortos

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

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

### Midiendo la respuesta

Cada patrón de uso del sistema debe seguir políticas de planificación diferentes.

Por ejemplo, para los procesos interactivos se pretende tener una cola preferente, aunque cuando hay demoras es preferible dar una respuesta consistente, aunque el promedio de tiempo de respuesta sea mayor.

Se definen como medidas de tiempo tick y quantum.

* Tick: Una fracción de tiempo en donde se puede realizar trabajo útil, es decir, usar la CPU sin interrupción. Su duración lo determina el timer del sistema. En Linux el tick es de 1 mili segundo y en Windows entre 10 y 15 ms.
* Quantum: El tiempo máximo que se le permitirá a un proceso el uso del procesador (por cada turno de ejecución).

En Linux el quantum varía entre 10 y 200 ticks (10 y 200 ms) y en Windows entre 2 y 12 ticks (10 y 180 ms).

### Métricas utilizadas

Para un proceso p que requiere tiempo t de ejecución:

* Tiempo de respuesta (T): Tiempo total necesario para completar el trabajo pendiente de un proceso p, incluyendo el tiempo que está inactivo esperando ejecución (pero está en cola de procesos listos).
* Tiempo de espera (E = T – t): También tiempo perdido. Del tiempo de respuesta total, cuanto tiempo p está listo y esperando ejecutar. Desde la óptica de p, se desearía que Ep -> 0.
* Proporción de penalización (P = T/t): Proporción de tiempo de respuesta en relación con el tiempo de uso del procesador (en qué proporción fue penalizado el proceso).
* Proporción de respuesta (R=t/T): Inverso de P. Fracción de tiempo de respuesta durante la cual p pudo ejecutarse.
* Tiempo núcleo o kernel: Tiempo que pasa el SO está usando el procesador para tareas propias del SO (cambio de contexto, ejecutar algoritmo de planificación). Tareas administrativas (overhead). Tiempo en espacio de núcleo, incluyendo entre otras funciones el empleado en decidir e implementar la política de planificación y los cambios de contexto. Este tiempo no se contabiliza cuando se calcula el tiempo de CPU utilizado por un proceso. Tareas propias.
* Tiempo de sistema: Tiempo que pasa el sistema en espacio núcleo atendiendo el pedido de un proceso (syscall). Se incluye dentro del tiempo de uso de CPU de un proceso y suele discriminarse del tiempo de usuario. Tareas de un proceso de usuario.
* Tiempo de usuario: Tiempo que pasa un proceso en modo usuario, es decir, ejecutando las instrucciones que forman parte explicita y directamente del programa.
* Tiempo de uso del procesador: Tiempo durante el cual el procesador ejecuto instrucciones por cuenta del proceso (sean en modo usuario o en modo núcleo).
* Tiempo desocupado (idle): Tiempo en que la cola de procesos listos está vacía y no puedo realizarse ningún trabajo. El SO no podría introducir un nuevo proceso mediante una interrupción si no existiese el proceso idle (no habría un proceso que interrumpir). Actualmente el proceso idle se utiliza para calcular estadísticas de los distintos procesos entre otras tareas administrativas.
* Utilización del CPU: porcentaje del tiempo en que el CPU está realizando trabajo útil.

### Algoritmos de planificación

#### Esquemas de planificación

* Sistemas cooperativos, no expropiativos o Non-Preemptive: Son aquellos en donde el SO no interrumpe al proceso en ejecución. Es éste el que cede el uso del procesador, o bien a través de una instrucción Yield ( ceder el paso) o debido a un Syscall (llamada al sistema). El SO no puede quitarle el uso de procesador al proceso en ejecución.
* Sistemas expropiativos o Preemptive: Son aquellos donde el reloj (timer) del sistema interrumpe periódicamente el proceso en ejecución para devolver el control al SO y que este decida que proceso será el próximo en ejecutarse.

#### Primero en llegar, primero servido (FCFS)

* Utiliza multitarea no expropiativos o non-preemtive
* Es el algoritmo más simple. El primero que llega a la cola de listos es el primero en ser atendido (First Come, First Serve)
* Reduce el mínimo la sobrecarga administrativa (overhead) ya que es muy simple de implementar.
* El rendimiento percibido por los últimos proceso en llegar resulta muchas veces inaceptable.
* Este algoritmo da salida a todos los procesos siempre que p <= 1. Si p > 1 la demora en iniciar a los nuevos procesos aumentará cada vez produciéndose inanición.

Gráfico, Gráfico en cascada

Descripción generada automáticamente

#### Ronda (Round Robin)

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

Gráfico

Descripción generada automáticamente

Gráfico, Gráfico en cascada

Descripción generada automáticamente

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

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

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

Conclusiones:

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

Gráfico

Descripción generada automáticamente

#### SPN apropiativo (PSPN, Preemptive Shortest Process Next)

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

#### El más penalizado a continuación (HPRN, Highest Penalty Ratio Next)

* Trabaja con esquemas no apropiativos o Non-Preemptive (puede ejecutarse Preemptive también)
* Intenta situarse en un nivel de justicia entre el FCFS (que favorece a los procesos largos) y SPN (que favorece a los procesos cortos).
* Al comienzo todos los procesos tienen una penalización de P=1. Cada vez que un proceso espera un tiempo w, la penalización de este se ajusta por P = (w+t)/t. El proceso que se elige como el próximo será el que tenga mayor P.
* Mientras que p < 1 este algoritmo incluso evita que los procesos más largos sufran inanición.
* Se han realizado experimentos (Finkel) y los resultados siempre arrojan que este algoritmo se sitúa entre FCFS y SPN
* Su principal desventaja se presenta cuando la cola de listos es muy larga ya que el planificador tiene que calcular P para cada uno de los procesos cada vez que se ejecuta.

#### Ronda Egoísta (SRR, Selfish Round Robin)

* Intenta favorecer a los procesos que ya han pasado tiempo ejecutando sobre los que recién llegan al sistema.
* Los procesos nuevos entran en una cola de procesos nuevos y solo se avanza con la ejecución de los procesos en la cola de procesos aceptados.
* Trabaja con dos parámetros iniciales. El parámetro α indica como se incrementará la prioridad de los procesos en la cola de procesos nuevos. El parámetro β indica como se incrementará la prioridad de los procesos aceptados.
* Cuando la prioridad de un nuevo proceso alcanza la prioridad de un proceso aceptado el proceso nuevo se convierte en aceptado.
* Si la cola de procesos aceptados queda vacía se acepta el proceso nuevo con mayor prioridad.

Gráfico, Gráfico en cascada

Descripción generada automáticamente

### Algoritmos con múltiples colas de listos

Sistema con múltiples colas de listos:

Diagrama

Descripción generada automáticamenteSe definen múltiples colas cada una con una propiedad.

Se atienden solo a los procesos de la cola de más prioridad hasta que está vacía (por terminación de sus procesos o porque los mismos fueron migrados a una menor prioridad). Luego se pasa a ejecutar los procesos de la cola siguiente. Puede haber colas vacías.

#### Retroalimentación multinivel (FB, Multilevel Feedback)

* Múltiples colas según prioridad (C0 a Cn)
* Se escoge al primer proceso de la cola más prioritaria con procesos en ella (Ci)
* Tras un predeterminado de ejecuciones del proceso escogido, si no finaliza, el mismo es degradado en la siguiente cola (Ci+1) y se ejecutan el siguiente proceso en la cola más prioritaria
* Favorece a los procesos cortos, ya que estos terminan antes de degradarse.
* Permite ajustar dos variables:

Cantidad de ejecuciones del proceso antes de ser degradado. (cada cola puede variar)

Duración del quantum (cada cola puede tener un quantum diferente).

* Los procesos más largos se van degradando, por lo que se penaliza a los procesos largos.

Gráfico

Descripción generada automáticamente

Gráfico

Descripción generada automáticamente

#### Lotería

* Se basa en un esquema de probabilidades para escoger al próximo proceso a ejecutar.
* A cada proceso se le asigna un número determinado de boletos.
* Cada vez que el planificador de corto plazo se ejecuta, elije un número aleatorio. El proceso que tiene el boleto con ese número es el que obtiene el siguiente quantum de ejecución (el algoritmo de generación de números aleatorios no es tan refinado, siendo en realidad pseudoaleatorio, y con solo 12 instrucciones RISC se ejecuta ).
* Cuantos más números de boleto tiene un proceso, más prioridades tiene de ser elegido.
* A pesar de ser simple, es muy justo, tanto para procesos largos como cortos.
* No se puede poner un ejemplo ya que su base radica en la aleatoriedad.

### Resumen y características de los algoritmos

Hasta ahora los algoritmos vistos se podrían calificar según dos criterios:

1. Aquellos que se utilizan en sistemas preemptive y otros que se utilizan en sistemas non-preemptive.
2. Aquellos que utilizan información intrínseca a los procesos (son tratados de diferente manera según su historial de ejecución) y aquellos no consideran esa información.

Tabla

Descripción generada automáticamente

### Algoritmos híbridos

#### Algoritmo por cola dentro de FB

A diferencia del FB tradicional, cada cola tiene un tratamiento diferente. Por ejemplo, parte de las colas podrían tener una especie de algoritmo PSPN que empuje a los proceso más largos a las colas que le permitan tener menos interrupciones. Las colas de menor prioridad podrían aplicar un SRR ya que ahí residen procesos que ya han esperado mucho tiempo para su ejecución, para que terminen lo antes posible su ejecución, sin esto alterar los tiempos de respuesta de procesos cortos que van entrando a las colas superiores.

#### Métodos dependientes del estado del sistema

La idea de estos algoritmos se nutra del estado actual del sistema e incluso de variables ajenas al dispatcher.

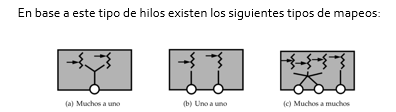
* Si los procesos listos son en promedio no muy largos y la saturación del sistema en baja (p<1) elegir algoritmos con poco overhead (FCFS o SPN). Si la cola de los listos comienza a crecer o e sistema comienza a saturarse, cambiar por ejemplo a RR con quantum corto o PSPN para garantizar una mejor distribución de la atención.
* Usar un RR simple con quantum variable periódicamente (por ejemplo con cada cambio de contexto o un cálculo periódico) de tal manera que sea proporcional a la cantidad de procesos en la cola de listos: q =q/n. Si hay pocos proceso esperando, su quantum será mayor, lo que reduce la cantidad de cambios de contexto. Si hay muchos, cada uno de ellos deberá esperar menos para comenzar con sus tareas.
* Utilizar un RR pero con quantum proporcional a la prioridad externa (definida por el usuario). A más prioridad quantum más largo.
* Peor servicio a continuación (WSN, Worst Service Next). Su principal diferencia respecto a HPRN es que no sólo se considera penalización al tiempo esperado, sino que también considera las veces que el temporizador lo interrumpió o su prioridad externa y se considera (a favor o en contra) el tiempo esperado por E/S. La gran desventaja es el overhead de tener que considerar tanas variables. Una solución a otra es acudir a WNS periódicamente y no en cada intervención del dispatcher, para que reordene las colas según los criterios generales, aunque esto repercute negativamente con respecto al tiempo de reacción del algoritmo ante cambios de comportamiento.
* Algunas versiones de Unix utilizan un esquema en donde la prioridad definida por el usuario era matizada y reevaluada durante el transcurso de su ejecución. Se maneja una prioridad interna que depende de la externa y el tiempo consumido recientemente por el proceso (conocido como lindura o niceness)

### Planificación de Hilos

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

#### Los hilos pueden ser de dos tipos:

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



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

### Planificación de multiprocesadores

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

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

### Afinidad a procesador

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

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

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

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

### Balanceo de carga

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

Existen dos técnicas de balanceo de carga:

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

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

### Una sola cola de listos para todos los procesadores

En esta técnica se utiliza una sola cola de listos para todos los procesadores. A medida que un procesador queda libre se escoge el primer proceso en la cola. Esto hace que incluso se ahorre la implementación de las tareas de migración vistas anteriormente. Sin embargo, este enfoque no es utilizado hoy en día en ningún SO masivo. Básicamente porque el mismo casi imposibilita la afinidad de CPU.

## UNIDAD 4

## Sincronización y comunicación entre procesos

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

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

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

* Sincronización entre procesos: Ordenamiento de las operaciones en el tiempo debido a las condiciones de carrera (acceder a diversos recursos asíncronamente).
* Comunicación entre procesos: Intercambio de datos.

La comunicación permite que los procesos cooperen entre sí en la ejecución de un objetivo global, mientras que la sincronización permite que un proceso continue su ejecución después de la ocurrencia de un determinado evento.

Problemas concurrentes

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

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

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

### Concurrencia

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

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

### Operación atómica

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

Race condition: Situación en el cual el resultado de la ejecución de 2 o mas procesos interactuantes depende del orden de ejecución de estos.

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

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

Mutua exclusión: solo un proceso a la vez puede estar ejecutando en su regio critica (lo accede y lo usa).

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

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

### Instrucciones FORK – JOIN

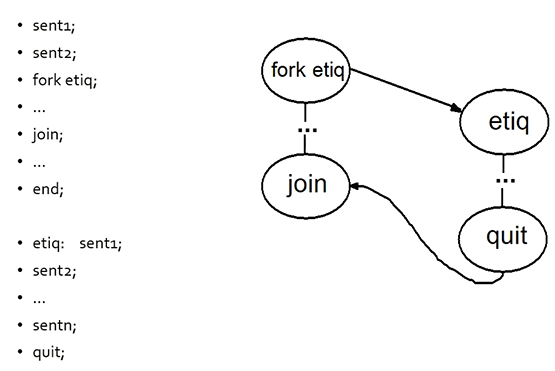
fork (tenedor, horqueta, separador): indica el comienzo de la concurrencia. Es en esencia un goto que simultáneamente bifurca y continua la ejecución.

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

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

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

No es estructurado por su estructura de control



### Grafos de precedencia

Diagrama

Descripción generada automáticamente

### Condiciones de concurrencia (Bernstein)

Dos sentencia cualesquiera Si y Sj pueden ejecutarse concurrentemente produciendo el mismo resultado que si se ejecutaran secuencialmente si solo si se cumplen las siguientes condiciones:

Texto

Descripción generada automáticamente

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

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente con confianza media

Diagrama

Descripción generada automáticamente

Intento 1: Una posible solución es utilizar únicamente una entrada (o torniquete). Esto podría ser una solución en tanto que no haya mucha gente que haga cola para entrar.

Sin embargo, es muy costoso.

Intento 2: Suspender la multitarea durante la región critica

Una versión mas relajada de la alternativa anterior es suspender la multitarea durante la ejecución de la sección critica. Así que el torniquete deberá hacer:

1 disable();

2 cuenta = cuenta + 1;

3 enable();

Inviable en multiusuarios: Deshabilitar en forma indefinida y capturar el CPU. No funciona en multicore ya que solo se deshabilitan interrupciones por procesador. Posibles errores de dispositivos si el tiempo en sección es muy largo.

Intento 3: Utilizar una bandera

Mediante una variable de bandera se indica si hay un proceso en la región critica:

Texto

Descripción generada automáticamente

Intento 4

Algunas arquitecturas de computadoras permiten realizar determinadas operaciones sencillas (como actualizar una bandera) de forma atómica.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

No falla pero tiene 3 problemas. La instrucción de comprobar-cambiar el valor de la variable, la cual es atómica, no está en todas las arquitecturas. Podría generar inanición debido a que se utiliza espera activa. Además es costoso porque la espera es ocupada.

Espera activa: Un proceso quiere acceder a un recurso, el mismo no esta disponible, y la variable se queda esperando el recurso haciendo uso del procesador.

Intento 5: Utilizar turnos

Una alternancia para evitar el problema de la actualización múltiple a una bandera es emplear una variable adicional que indique a que proceso corresponde avanzar en todo momento, esto es, utilizar turnos:

Texto, Carta

Descripción generada automáticamente

### Semáforos

1968 – Edsger Dijkstra

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

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

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

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

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

La característica fundamenta de estos operadores: Su ejecución es indivisible o sea atómica.

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

#### Señalizar

Un hilo debe informar a otro que cierta condición está cumplida (por ej, un hilo prepara una conexión de red mientras que otro calcula lo que tiene que enviar:

Tabla

Descripción generada automáticamente

#### Rendezvous

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

Imagen que contiene Texto

Descripción generada automáticamente

#### Mutex

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

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

#### Multiplex

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

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

#### Torniquete

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

Texto

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

#### Apagador

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

Tabla

Descripción generada automáticamente

Tabla

Descripción generada automáticamente

#### Barrera

Texto

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

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

Texto

Descripción generada automáticamente con confianza media

#### Monitores

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

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

### Comunicación entre procesos (IPC – Inter Process Communication)

La comunicación entre procesos puede ser realizada de dos maneras:

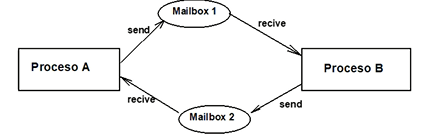
1. Comunicación a través de un área común de memoria.
2. Comunicación mediante el intercambio de mensajes.

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

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

#### Tipos de comunicación entre procesos

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



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

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

### Tipos de sincronizaciones mediante mensajes

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

### Modelo productor-consumidor

Usado para describir dos proceso ejecutando en forma concurrente:

Productor: genera un conjunto de datos necesarios para la ejecución de otro proceso.

Consumidor: Toma los datos generados por el productor y los utiliza para su procesamiento.

Texto

Descripción generada automáticamente con confianza baja

Productor: Genera elementos mediante producir() y los ingresa en el buffer mediante depositar()

Buffer: Zona de memoria utilizada para amortiguar las diferencias de velocidad entre dos procesos. Almacena temporalmente los elementos generados por p.

Consumidor: Retira los elementos del buffer mediante recuperar() y los consume con consumir()

Ej. ls | more El spooler del SO y los procesos que quieren imprimir

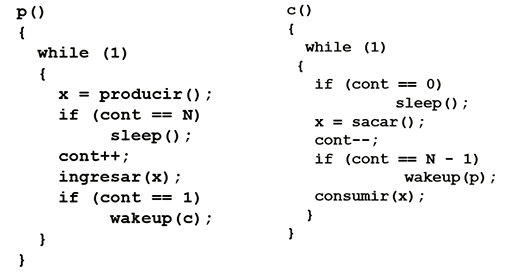
#### Algoritmos para el modelo productor-consumidor

1. Con sleep() & wakeup()

Sleep(): Llamada al sistema que bloquea al proceso solicitante.

Wakeup(): tiene como parámetro el PID del proceso a desbloquear.

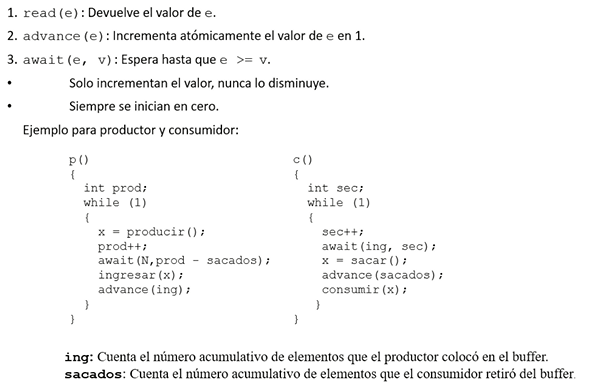
Usa una variable cont que indica la cantidad de lugares ocupados que tiene el buffer, donde N es el total de lugares



Este algoritmo no funciona. El if y el sleep no se ejecutan atómicamente. Esto puede llevar a una deadlock.

1. Con contador de eventos

Un contador de eventos “e” es una variable especial que tiene 3 operaciones definidas en las siguientes primitivas.



También tiene problemas

1. Con semáforos

Tabla

Descripción generada automáticamente

### Mecanismos de IPC

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

### Bloqueos mutuos e inanición

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

Puede darse por 2 motivos:

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

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

##### Condiciones de Coffman (necesarias y suficientes)

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

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

Recursos reutilizables: Puede ser usado por un proceso y no se agota con el uso. Los procesos obtienen unidades de recursos que liberan posteriormente para que otros procesos las reutilicen. Como ejemplos, se tienen los procesadores, canales E/S, memoria principal y secundaria, dispositivos y estructuras de datos tales como archivos, bases de datos y semáforos.

Recursos consumibles: Puede ser creado (producido) y destruido (consumido). Normalmente, no hay limite en el numero de recursos consumibles de un tipo en particular. Un proceso productor que no está bloqueado puede liberar a cualquier número de recursos consumibles. Cuando un proceso adquiere un recurso este deja de existir. Como ejemplos están las interrupciones, señales, mensajes, e información en buffer de E/S.

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

#### Grafos de asignación de recursos

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

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

Un dibujo de una persona

Descripción generada automáticamente con confianza mediaLos recursos fueron representados con cuadrados y los procesos con círculos.

Estrategias para tratar deadlocks:

* Ignorarlos: La mayoría de los sistemas operativos utilizan esta técnica, incluyendo UNIX. Es una de las formas mas simple de tratarlo. Algunas veces el costo de ignorarlo es mucho menor al costo que implicaría prevenirlo o detectarlo y recuperarlo. Tanenbaum lo llama el ‘algoritmo del avestruz’.
* Prevención: Siendo las 4 condiciones necesarias para que ocurra un deadlock, basta con asegurarnos que una de ellas no ocurrirá.
  + Mutua exclusión: las soluciones para aquellos recursos que no pueden ser compartidos son diversas, pero todas se basan en que un proceso no quede esperando en caso de la falta de disponibilidad de dicho recurso.
  + Toma y espera (Hold & Wait): Para solucionar este problema, se trata de garantizar que cuando un proceso tenga un recurso asignado no pueda solicitar otro. Hay dos caminos para lograrlo: 1) Los procesos solicitan todos los recursos en el momento previo a comenzar la ejecución, de no poder ser entregados el proceso queda bloqueado. Asignación estática completa de todos los recursos que necesita para su ejecución (caso COBOL). 2) Un proceso primero debe liberar aquellos recursos que posee y luego recién podrá solicitar otros, es decir solo está en condiciones de solicitar un recurso cuando no tiene ninguno asignado.
  + No expropiación (No Preemption): 1) Si un proceso solicita un recurso que no esta disponible, este debe devolver todos aquellos recursos que tenía previamente asignados. 2) Si un proceso pide un recurso que tiene otro proceso, el SO puede obligar a liberar los recursos al otro proceso.

El primer método es viable solo en aquellos procesos cuyos estados pueden ser fácilmente grabados y restaurados. El segundo método presenta el inconveniente del estado de inanición en caso de que a un proceso siempre le quiten los recursos y nunca pueda finalizar su tarea.

* + Espera circular: Consiste en imponer un orden lineal de ejecución que evite las esperas circulares: 1) Estableciendo un orden lineal a los recursos: Si tenemos una lista de recursos R1, R2, …, Rn, un proceso que solicito Rh solo puede pedir aquellos recursos Rk con k > h. Esto evita que se forme un círculo. 2) Hold & Wait: un proceso solo está en condiciones de solicitar un recursos cuando no tiene ninguno asignado. El problema del orden lineal es que también tiene cierto grado ineficiente ya que estaría negando recursos a procesos, innecesariamente.
* Detectar y recuperar: Consiste en abortar un proceso cuando se detecta o se presupone que puede ocurrir el deadlock. La ventaja de esta táctica es que no limita el acceso a los recursos. Presenta ciertos inconvenientes como el de decidir la frecuencia con los que se llevará a cabo el algoritmo de detección. El algoritmo podría ser ejecutado cada vez que solicita un recurso, a cada hora, etc.

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

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

## Unidad 5

## Administración de Memoria Central

### Funciones y Operaciones

* Por cuestiones de diseño, el único espacio de memoria que el procesador puede utilizar es la Memoria Central (MC)
* Los caches son para mejorar la performance y en general replican el contenido de la MC
* Los registros del procesador son muy pequeños y solo los utiliza para realizar sus operaciones
* Los discos son un almacenamiento secundario. No es accesible directamente por el procesador. Están conectados al sistema mediante el módulo I/O
* Los programas deben cargarse en la MC antes de ser ejecutados. Depende de como se gestione la memoria se cargará o no todo completo.
* También está relacionado con el HW que disponga el equipo, en especial el MMU

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

Un procesador que soporta un espacio de direccionamiento de 16 bits puede referirse directamente a hasta 216 bytes, esto es 64 KB.

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

A través de un mecanismo llamado PAE (extensión de direcciones físicas, Physical Address Extension) permite extender eso a rangos de hasta 252 bytes a cambio de un nivel más de indirección.

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

En la práctica por cuestiones económicas, es de 240 a 248.

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

### Hardware

#### MMU (Memory Manager Unit)

Nos va a permitir varias funciones de administración de memoria. Resuelve tareas para mejorar la performance. Controla que no hay un segmentation fault. Resuelve estas tareas para mejorar la performance.

Hace falta cuando:

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

Pasa la información al MMU y el MMU realiza la función por una cuestión de performance.

#### Memoria Caché

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

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

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

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

### Espacio de memoria de un proceso

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

* Sección (o segmento) de texto o código: Es la imagen en memoria de las instrucciones a ser ejecutadas. Usualmente, ocupa las direcciones más bajas del espacio de memoria. Ahí va el código. Es un área fija porque el código no cambiará en la ejecución.
* Sección de datos: Espacio fijo preasignado para las variables globales y datos inicializados (como las cadenas de caracteres por ejemplo). Se fija en tiempo de compilación , y no puede cambiar (aunque los datos que cargados allí si cambian en el tiempo de vida del proceso).
* Diagrama

  Descripción generada automáticamenteEspacios de libres (HEAP): Espacio para la asignación dinámica de memoria durante la ejecución del proceso. Este espacio se ubica por encima de la sección de datos, y crece hacia arriba. Esta área si puede cambiar.

Lenguajes dinámicos (C) -> Free / Malloc

Lenguajes gestión automática (java) -> garbage collector.

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

### Resolución de direcciones

Diagrama

Descripción generada automáticamenteEl compilador reemplaza las variables / funciones simbólicas por las direcciones de memoria a donde se ubican. (ej. posición de inicio + 80 a posición AC897AB)

Para poder coexistir con otros procesos, esas direcciones deben ser traducidas a la posición relativa, con alguna de las siguientes estrategias:

* En tiempo de compilación: Las direcciones son absolutas / fijas. Ej command.com. Esto no se utiliza porque si se compilan por ej, dos direcciones iguales en dos programas distintos esto provocará una falla. No se usa
* En tiempo de carga: El loader calcula las ubicaciones al momento del inicio del programa (ej: Reg Base + offset). Para esa ejecución del programa, la ubicación será definida en el tiempo de carga y no cambiará hasta que termine la ejecución.
* En tiempo de ejecución: El programa NUNCA hace referencia a una ubicación fija (ej: base\_frame + offset). Esta traducción la hace el MMU. Cuando el programa hace referencia a la ubicación lógica, en ese momento se traduce a la dirección física.

### Asignación de memoria contigua

En todas estas, en cada partición hay un registro base y un registro límite. Estos registros se guardar en el PCB.

#### Simple

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

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

#### Particiones fijas

Diagrama

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

La cantidad de particiones que yo tenga determinara la cantidad de procesos que yo pueda tener activos simultáneamente (nivel de multiprogramación). Es importante definir bien el tamaño de las particiones y la cantidad de las particiones (aunque es mejor tener más particiones por la multiprogramación, a más particiones, más chicas). El proceso más grande que se va a ejecutar debe ser como máximo del tamaño de la partición más grande. Fragmentación interna (es cuando hay parte de la memoria de la partición que no se está utilizando).

Diagrama

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

#### Particiones variables

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

First fit: hace hincapié en la velocidad. Lo mete en el primero lugar que cabe.

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

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

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

Fragmentación externa( huecos libres que no corresponden a ninguna partición). Para solucionarla, aplico desfragmentación (tiene un costo). Al principio cada proceso tiene asignada una partición con tamaño acorde, pero a medida que se van terminando los procesos, empiezan a quedar huecos de distinto tamaño sin utilizar. Cuando llega un nuevo proceso, ahí utilizo los algoritmos antes mencionados de asignación de espacio libre.

### Segmentación

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

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

Traducción en tiempo de ejecución.

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

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

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

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

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

Puede haber fragmentación y se resuelve desfragmentando.

Diagrama

Descripción generada automáticamente

### Paginación

El principal problema de las técnicas de administración de memoria hasta ahora era que la memoria para un proceso tenia que ser contigua. Para solucionar este tema surge la paginación.

#### Pura o simple

Tabla

Descripción generada automáticamente con confianza bajaEl proceso se divide en bloques de tamaño fijo llamados páginas (pages).  
La memoria física se divide en una serie de marcos (frames) del mismo tamaño que las páginas. Gracias a esto, en mi memoria tendré n frames, y podre cargar en ellos páginas de cualquier proceso (todas iguales). Mi proceso podrá cargar sus páginas en cualquier frame (no hay más necesidad de que la memoria sea contigua). Por lo general los frames/pages se dividen en porciones de entre 512B (29) y 16MB (224). Siempre son potencias de 2. Para el direccionamiento se divide el bus de direcciones considerando N bits para la página y M bits para el desplazamiento (offset).

Los frames son todos del mismo tamaño y las paginas son todas del mismo tamaño. Desaparece la fragmentación externa, vuelve a aparecer algo (no mucho) de fragmentación interna en la última página de cada proceso.

Diagrama

Descripción generada automáticamente

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

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

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

### Direccionamiento con TLB

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

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

Diagrama

Descripción generada automáticamente

### Direccionamiento con TLB y Cache

Diagrama

Descripción generada automáticamente

Se usa TLB + memoria cache.

### Paginación Multinivel

Se definen tablas de directorios que a su vez apuntan a tablas de páginas. Consiste en dividir mis páginas en dos niveles. Ej. sistemas de 32 bits, con 10 para la tabla externa y 12 para el desplazamiento. Al tener una tabla que usare como índices de las páginas solo tendré dos accesos en vez de ir a buscar pagina por pagina lo que necesito. Ej en vez de tener un registro de más de un millón de elementos tendré dos paginas de 1024.

Tabla

Descripción generada automáticamente con confianza media

Diagrama

Descripción generada automáticamente

### Memoria Compartida

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

Diagrama

Descripción generada automáticamente

### Memoria Virtual

### Paginación sobre demanda

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

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

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

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

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

Si el intercambio es excesivo hay hiperpaginacion o trashing.

Políticas de Memoria Virtual:

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

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

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

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

Diagrama

Descripción generada automáticamente

### Fallo de página

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

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

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

Diagrama

Descripción generada automáticamente

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

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

Diagrama

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

### Anomalía de Belady

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

### Reemplazo de páginas. Algoritmos

#### FIFO

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

#### Optimo (OPT, MIN)

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

#### LRU (Less recently used)

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

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

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

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

#### Bit de Referencia

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

#### Columna de Referencia

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

#### Segunda Oportunidad (o reloj)

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

#### Segunda Oportunidad mejorado

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

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

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

### Asignación de Marcos

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

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

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

### Hiperpaginación

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

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

Síntomas:

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

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

### Sistemas Mixtos

#### Segmentación con paginación por demanda

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



Ventajas:

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

Desventajas:

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

Diagrama

Descripción generada automáticamente

## Unidad 6

## Administración de Entrada / Salida

## File System

Hay una amplia variedad de periféricos con varios métodos de operación. Sería poco práctico incorporar la lógica necesaria dentro del procesador para controlar un rango de dispositivos uno por uno. La velocidad de transferencia de datos de los periféricos a menudo es bastante menor que la de la memoria central o procesador. Los periféricos generalmente emplean distintos formatos de datos y longitudes de palabra que los utilizados en el computador que se conecta.

Diagrama

Descripción generada automáticamente

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

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

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

* Proveer una interfase con el procesador y la memoria central vía el bus del sistema.
* Proveer una interfase a uno o más de los dispositivos periféricos por vínculos especialmente diseñados.

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

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

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

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

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

### Funciones del Módulo de E/S:

Control y Temporización

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

Pasos:

* + - El procesador interroga al módulo de E/S para verificar el estado(status) del dispositivo conectado
    - El modulo de E/S devuelve el estado del dispositivo.
    - Si el dispositivo esta operable y listo para transmitir, el procesador solicita la transferencia de datos, por medio de un comando al módulo de E/S.
    - El modulo de E/S obtiene una unidad de datos (por ejemplo 8 o 16 bits) del dispositivo externo.
    - Los datos son transferidos desde el módulo de E/S al procesador.

#### Comunicación con el procesador:

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

Incluye:

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

#### Comunicación con el dispositivo:

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

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

#### Amortiguación (buffering) de datos:

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

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

#### Detección de errores:

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

#### Manejo de interrupciones a bajo nivel:

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

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

### Componentes principales

#### Controlador o Driver

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

#### Software Independiente del Dispositivo

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

### Procesadores de E/S (IOP)

En los grandes equipos (Mainframe) generalmente se conectan mediante varios buses y un procesador especial de E/S que se ocupa de la gestión. A esta configuración se denomina subsistema de E/S que hace de interface con el procesador y memoria central. Todas las operaciones sobre los periféricos se arrancan con una orden generada en el procesador para el procesador de E/S o canal, quien realizará las transacciones en forma independiente asumiendo el completo control de estas.

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

### Técnicas de E/S

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

#### E/S Programada, también llamada Polling (Escrutinio)

Esta técnica es la más antigua.

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

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

#### E/S dirigida por interrupciones (Interrupt Driven)

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

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

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

#### Acceso Directo a Memoria (DMA)

* El módulo de E/S y la memoria central intercambian datos directamente, sin involucrar a la CPU.
* El DMA requiere un módulo adicional en el bus del sistema.
* El modulo de DMA es capaz de simular el procesador y tomar el control del sistema desde la CPU.
* Cuando el procesador desea leer o escribir un bloque de datos, emite un comando al módulo de DMA, enviándole la siguiente información:

1. Si la operación solicitada es una lectura o escritura
2. La dirección del dispositivo de E/S involucrado
3. La posición inicial en memoria a leer o a escribir
4. El número de palabras a ser leído o escrito.

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

#### Transferencia de datos mediante DMA

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

* Por ráfagas: Cuando el DMA toma el control del bus, no lo libera hasta haber transmitido el bloque de datos pedido. Se consigue la mayor velocidad de transferencia, pero se tiene inactiva la CPU (parada del procesador).
* Por robos de ciclos: Cuando el DMA toma el control del bus, lo retiene durante un solo ciclo. Transmite una palabra y libera el bus. Es la forma más usual. Reduce al máximo la velocidad de transferencia y la interferencia del DMA sobre la actividad del procesador.
* DMA transparente: Se elimina completamente la interferencia entre el DMA y la CPU. Solo se roban ciclos cuando la CPU no está utilizando el bus del sistema. No se obtiene una velocidad de transferencia muy elevada.

### Discos

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

#### Hardware del Disco

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

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

#### Tiempos que intervienen en una lectura de disco y la transferencia de datos a la memoria central

Interfaz de usuario gráfica, Texto, Correo electrónico

Descripción generada automáticamente

#### Performance en los discos

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

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

#### Algoritmos de Planificación

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

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

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

### Dispositivos en estado sólido

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

* NVRAM
* Memorias Flash
* SSD

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

## File System

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

### Sistemas basados en Disco

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

Diagrama, Esquemático

Descripción generada automáticamente

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

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

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

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

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

### Objetivos y Funciones del Sistema de Gestión de Archivos

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

### Catalogación de los archivos en el soporte

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

* Área de datos fijos. Fixed Data Área -FDA, que incluye la Tabla de Particiones (es una tabla que tiene las particiones lógicas del disco. (Ej. disco C y disco D) y el Sector de Booteo (donde esta el loader del SO) (partition table y Boot Sector).
* Área de Catálogo: área del Espacio libre y del espacio ocupado (Free Space List y File Allocation Table). Esta última está compuesta por un área especifica de Directorio (File Directory Block) de varios niveles, que incluye el Master File Directory y al User File Directory. (Dinámica)
* Area de Datos (donde se localizan físicamente los datos de los archivos).

### Administración del espacio de almacenamiento

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

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

### Administración del Espacio Libre

#### Bit Map o Bit Vector (Mapa de Bits)

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

Ventajas: Sencilla implementación. Relativamente poco espacio ocupado.

Desventajas: No posee grandes desventajas.

(Utilizado por DOS)

#### Lista Enlazada de Bloques Libres

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

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

Ventajas: Muy poco espacio ocupado

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

#### Bloques de Direcciones Libres

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

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

Luego el último puntero contiene la dirección del próximo bloque que contiene las direcciones de otros bloques n bloques libres

(Utilizado por Linux)

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

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

#### Bloques de Direcciones Libres Contiguas

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

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

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

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

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

### Métodos de Asignación de espacio para los Archivos

Se usan tres métodos de asignación de espacio de disco:

#### Contiguo

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

Tabla

Descripción generada automáticamente

Ventajas:

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

Desventajas:

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

###### Asignación Dinámica de Almacenamiento

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

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

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

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

#### Vinculado -link-

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

Imagen que contiene Diagrama

Descripción generada automáticamente

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

Ventajas:

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

Desventajas:

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

#### Indexado

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

Tabla

Descripción generada automáticamente

Ventajas:

* Soporta acceso directo y secuencial
* No tiene fragmentación externa

Desventajas:

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

### Acceso Secuencial

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

### Acceso Directo

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

### I-Nodos

Es una estructura de datos Es una estructura de datos que consta de un encabezado donde hay datos del archivo y luego los 39 bytes de información de direcciones. Los atributos del archivo se almacenan en el I-nodo. Cuando se abre un archivo se trae su I-nodo a memoria.

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

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

Diagrama

Descripción generada automáticamente

Ejemplo de un tamaño máximo de un archivo:

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

(10 + 256 + 2562 + 2563) x 1kb ≈ 16 Gb

Ventajas:

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

### FAT (File Allocation Table)

Tengo una estructura, la cual es un vector de punteros con un puntero por cada bloque de disco. Además tengo un archivo que mide n bloques. En mi tabla de archivo tengo el archivo y el puntero al bloque inicial, para encontrar los sucesivos bloques debo buscar en la FAT. Es como una lista enlazada que en vez de tener el puntero dentro del bloque de datos lo tiene por fuera.

Interfaz de usuario gráfica, Diagrama

Descripción generada automáticamente

### Soft link (Acceso directo) link simbólico

Crea un nuevo archivo con un bloque de disco que contiene la información del archivo al que apunta.

Interfaz de usuario gráfica, Aplicación, Tabla, Excel

Descripción generada automáticamente

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

### Hard link

Es una referencia nueva en la table de archivos a un archivo que ya existe ( no se puede hacer en Windows). Un archivo se borra recién cuando se borre el último hard link de ese archivo. Cada hard link puede tener distintos permisos.