**Procesos e hilos:**

**Generalidades:**

* Un sistema informático debe poder realizar las siguientes acciones:
* Intercalar la ejecución de múltiples procesos para maximizar la utilización del procesador ofreciendo a la vez un tiempo de respuesta razonable.
* Asignar los recursos a los procesos.
* Dar soporte a la comunicación entre procesos y la creación de procesos por parte del usuario.

Al proceso también se le llama tarea o ejecución de un programa individual.

Traza del proceso:

- Listado de la secuencia de instrucciones que se ejecutan para dicho proceso.

- Distinguir entre Proceso (concepto dinámico) y Programa (concepto estático).

Componentes:

Proceso = Código en forma de objeto

+ BCP (PID, estado, prioridad, CP, registros, ficheros, E/S, ...)

+ Pila (datos temporales, parámetros,direcciones de retorno...)

+ Datos (globales)

**Modelos de estados del proceso:**

Estados: Ejecución, No ejecución: listos para ejecutarse o esperando a que termine una operación de E/S.

Diagrama

Descripción generada automáticamente

**Problema**: Sólo una cola de “no ejecución” ⇒ el distribuidor podría no seleccionar el proceso que lleva más tiempo en la cola porque este podría estar bloqueado.

* **Ejecución**: el proceso que está actualmente en ejecución.
* **Listo**: procesos que están preparados para ejecutarse en cuanto se le dé la oportunidad.
* **Bloqueado**: procesos que no se pueden ejecutar porque esperan que termine algún suceso (operación de E/S).
* **Nuevo**: procesos que se acaban de crear, pero que aún no se han cargado en memoria principal.
* Finalizado: procesos que han sido excluidos por el S.O. del grupo de procesos **ejecutables**.

Diagrama

Descripción generada automáticamente

Típicamente existen múltiples colas para procesos en estado bloqueado, una para cada tipo de suceso.

En los diagramas de Gantt no se suele marcar el planificador dado que su tiempo de ejecución suele ser despreciable frente a los tiempos de ejecución de los procesos

**Situaciones en las que se crea un proceso:**

- Emisión de un trabajo por lotes.

- Un nuevo usuario intenta conectarse.

- Se crea para ofrecer un servicio, como por ejemplo la impresión.

- Un proceso origina la creación de otro.

**Situaciones para el cierre de un proceso:**

- Un trabajo por lotes debe incluir una instrucción de detención (End).

- El usuario se desconecta.

- El usuario puede abandonar una aplicación.

- Una serie de errores y condiciones de fallo pueden llevarnos a la terminación de un proceso.

**Motivos de cierre de un proceso:**

Terminación normal. Tiempo límite excedido. No hay memoria disponible. Violación de límites. Error de protección. Por ejemplo: escribir en un archivo que es solo de lectura. Error aritmético. Tiempo máximo de espera rebasado. El proceso ha esperado más allá del tiempo máximo especificado para que se produzca cierto suceso. Fallo de E/S. Instrucción ilegal a menudo cuando intenta ejecutar los datos. Instrucción privilegiada. Mal uso de los datos. Intervención del operador o del S.O. Por ejemplo, si se produce un interbloqueo. Solicitud del padre.

**Problema**: El procesador es más rápido que la E/S, por lo que suele ser habitual que todos los procesos de memoria están esperando por E/S y por lo tanto se tiene toda la memoria ocupada con procesos, pero ninguno se está ejecutando.

**Solución**: Intercambiar una parte del proceso o todo el proceso al disco para liberar la memoria principal → **Intercambio**. Creación de un nuevo estado, el de **Suspendido**, en el que el proceso se almacena en el disco. Cuando los procesos de la memoria principal están en el estado Bloqueado, el sistema operativo puede suspender un proceso poniéndolo en estado Suspendido.

Diagrama

Descripción generada automáticamente

**Suspendido**: el proceso está en la memoria secundaria esperando un suceso.

Para suspenderlo hay que pasar una parte de (o todo) el proceso al disco para liberar la memoria principal.

El procesador es más rápido que la E/S ⇒ es habitual que haya procesos en memoria esperando a E/S. Cuando los procesos en memoria principal están bloqueados, el S.O. puede eliminarlo de la memoria poniéndolo en estado Suspendido, y así queda más memoria disponible para los procesos listos y para admitir nuevos procesos.

**Problema**: En este modelo sólo pueden ser suspendidos aquellos procesos que están en estado de bloqueo. Podría necesitar memoria para admitir procesos nuevos y no existir procesos bloqueados candidatos a ser suspendidos.

**Solución**: Suspender procesos que estén en el estado de listos. El estado de suspendido se divide en dos estados: Suspendido-bloqueado y Suspendido-listo

**Bloqueado y suspendido**: el proceso está en la memoria secundaria esperando un suceso.

**Listo y suspendido**: el proceso está en la memoria secundaria disponible para su ejecución tan pronto como se cargue en la memoria principal.

Diagrama

Descripción generada automáticamente

Motivos para suspender proceso:

* **Intercambio**: El sistema operativo necesita liberar suficiente memoria principal para cargar un proceso que está listo para ejecutarse.
* **Otra razón del S.O.**: El sistema operativo puede suspender a un proceso subordinado o de utilidad, o a un proceso que se sospecha que sea el causante de un problema.
* **Solicitud de un usuario interactivo:** Un usuario puede querer suspender la ejecución de un programa con fines de depuración o en conexión con el uso de un recurso.
* **Temporización:** Un proceso puede ejecutarse periódicamente (por ejemplo, un proceso de contabilidad o de supervisión del sistema) y puede ser suspendido mientras espera el siguiente intervalo de tiempo.
* **Solicitud del proceso padre:** Un proceso padre puede querer suspender la ejecución de un descendiente para examinar o modificar el proceso suspendido o para coordinar la actividad de varios descendientes.

**Estructuras de control:**

El objetivo es mantener la información sobre el estado actual de cada proceso y de cada recurso.

El sistema operativo construye tablas de información sobre cada entidad que esté administrando.

**Las tablas de memoria almacenan la siguiente información:**

* La asignación de memoria principal a los procesos.
* La asignación de memoria secundaria a los procesos.
* Atributos de protección de bloques de memoria principal o virtual. P.ej: qué procesos pueden acceder a ciertas regiones compartidas de memoria.
* Cualquier información necesaria para gestionar la memoria virtual.

**Las tablas de E/S almacenan la siguiente información:**

* Estado del dispositivo de E/S, que puede estar disponible o asignado a un proceso en particular.
* Estado de la operación de E/S.
* Posición de memoria principal que se está utilizando como origen o destino de la transferencia de E/S

**Las tablas de archivos almacenan la siguiente información:**

* La existencia de los archivos.
* La posición de los archivos en la memoria secundaria.
* El estado actual de los archivos.
* Otros atributos de archivos.
* A veces esta información es mantenida por un sistema de gestión de archivos independiente.

**Las tablas de procesos almacenan la siguiente información:**

* Ubicación del proceso, que incluye:
  + Instrucciones a ejecutar.
  + Datos para las variables locales y globales.
  + Constantes definidas.
  + Pila para almacenamiento de parámetros y direcciones de retorno de procedimientos.
* Bloque de control del proceso, que incluye una colección de atributos. Conjunto de metadatos necesarios para la administración del proceso:
* Atributos del proceso necesarios para su administración:
  + Diagrama

    Descripción generada automáticamenteID del proceso.
  + Estado del proceso.
  + Ubicación en la memoria.

**Bloque de control de proceso:**

Diagrama

Descripción generada automáticamente

Son una serie de identificadores numéricos entre los que deben estar:

* Identificador de este proceso.
* Identificador del proceso que creó a este proceso (el proceso padre).
* Identificador del usuario.

Información del estado del procesador:

Formada por el contenido de los registros del procesador.

Registros visibles para el usuario.

Registros de control y de estado.

Punteros de pila.

Registros visibles para el usuario:

Son aquellos a los que puede hacerse referencia por medio del lenguaje de máquina que ejecuta el procesador. Normalmente, existen de 8 a 32, aunque algunas implementaciones RISC tienen más de 100. Por ejemplo, en C se puede hacer referencia a ellos de la siguiente forma: register int i;

Punteros de pila:

Cada proceso tiene una o más pilas LIFO del sistema asociadas. Las pilas se utilizan para almacenar los parámetros y las direcciones de retorno de los procedimientos y de las llamadas al sistema. El puntero de pila siempre apunta a la cima de la pila.

Registros de control y de estado:

Hay varios registros del procesador que se emplean para controlar su funcionamiento. Entre estos se incluyen:

Contador de programa: contiene la dirección de la próxima instrucción a leer. Códigos de condición: muestran el resultado de la operación aritmética o lógica más reciente (signo, cero, acarreo, igualdad, desbordamiento).

Información de estado: incluye los indicadores de habilitación o inhabilitación de interrupciones y de modo de ejecución.

Información de planificación y de estado. La necesita el sistema operativo para llevar a cabo sus funciones de planificación. Elementos típicos de esta información:

* Estado del proceso: disposición del proceso para ser elegido para su ejecución (en ejecución, listo, bloqueado).
* Prioridad: se puede usar uno o más campos para describir la prioridad de planificación de los procesos. En algunos sistemas se necesitan varios valores (por omisión, actual, la más alta permitida).
* Información de planificación: dependerá del algoritmo de planificación utilizado. Ejemplos: cantidad de tiempo que el proceso ha estado esperando, cantidad de tiempo que el proceso se ejecutó la última vez, etc.
* Suceso: identidad del suceso que el proceso (bloqueado) está esperando antes de poder reanudarse.

Estructuración de datos:

Un proceso puede estar enlazado con otros procesos en una cola, un anillo o alguna otra estructura. Por ejemplo, todos los procesos que están en estado de espera de un nivel determinado de prioridad pueden estar enlazados en una cola. Un proceso puede mostrar una relación padre-hijo (creador-creado) con otro proceso. El bloque de control de proceso puede contener punteros a otros procesos para dar soporte a estas estructuras.

Comunicación entre procesos:

Puede haber varios indicadores, señales y mensajes asociados con la comunicación entre dos procesos independientes. Una parte de esta información o toda ella se puede guardar en el bloque de control de proceso.

Privilegios de los procesos.

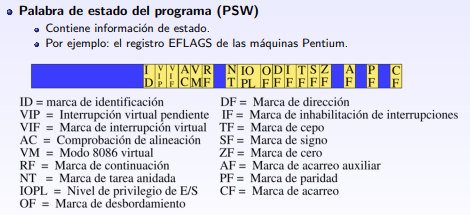
A los procesos se les otorgan privilegios en términos de la memoria a la que pueden acceder y el tipo de instrucciones que pueden ejecutar. Además, también se pueden aplicar privilegios al uso de los servicios y utilidades del sistema.

Gestión de memoria.

Esta sección puede incluir punteros a las tablas de páginas o segmentos que describen la memoria virtual asignada al proceso.

Propiedad de los recursos y utilización.

Se pueden indicar los recursos controlados por el proceso, como los archivos abiertos. También puede incluir un historial de la utilización del procesador o de otros recursos; esta información puede ser necesaria para el planificador.



**Modos de ejecución:**

Modo de usuario:

Es el modo menos privilegiado. Los programas de usuarios se ejecutan normalmente de ese modo.

Modo del sistema, modo de control o modo del núcleo:

Es el modo más privilegiado. Núcleo del sistema operativo

**Manipulación de procesos**

**Creación de procesos**:

* Asignar un único identificador al nuevo proceso.
* Asignar espacio para el proceso.
* Iniciar el bloque de control del proceso.
* Establecer los enlaces apropiados.
  + Por ejemplo: añadir un proceso nuevo a una lista enlazada que se utiliza como cola de planificación.
* Crear o ampliar otras estructuras de datos.
  + Por ejemplo: mantener un archivo de contabilidad, actualizar la tabla de gestión de memoria.

Existen varias situaciones que pueden hacer que el SO cambie el estado de un proceso:

* Interrupción de reloj: El proceso en ejecución ha consumido la fracción máxima de tiempo permitida.
* Interrupción de E/S.
* Fallo de memoria (con memoria virtual): La dirección de memoria se encuentra en la memoria virtual, por lo tanto, debe ser llevada a la memoria principal.
* Cepos:
  + Se ha producido un error.
  + Puede hacer que el proceso que se estaba ejecutando pase al estado de Terminado.
* Llamada del supervisor o llamadas al sistema:
  + Como la operación de abrir un archivo.

**Llamadas al sistema:**

Los procesos se comunican con el SO a través de llamadas al sistema.

Cada llamada se corresponde normalmente con un procedimiento que lee los parámetros de la llamada y los pasa al SO, junto con el control, mediante un TRAP. Esta llamada pasa de modo usuario a modo supervisor.

**Cambio de contexto:**

Guardar contexto:

* Guarda CP en la pila de núcleo del proceso (o en el BCP).
* Guarda el contenido de los registros del procesador en el BCP del proceso (o en la pila de núcleo del proceso).
* Guarda el puntero de pila en el BCP.

Recuperar contexto: copiar en los registros del procesador los valores almacenados en el BCP del proceso a ejecutar.

* Recupera puntero de pila del BCP.
* Recupera los demás registros del BCP y de la pila.
* Cambia algunos bits de la palabra de estado:
  + Cambia el bit para habilitar las interrupciones.
  + Cambia el modo de ejecución (pasa a modo usuario).
* Recupera el CP. Al recuperar el valor de CP deja de ejecutar el S.O. y se ejecuta el programa de usuario.

Lo que ocurre siempre que llega una interrupción, no sólo cuando hay que cambiar de un proceso a otro.

* Se salva el contexto del programa que se ejecuta (copia la información del estado del procesador: CP en la pila, registros en BCP, puntero de pila en el BCP).
* Asigna al CP el valor de la dirección de comienzo del programa de atención a la interrupción.
* Cambia de modo usuario a modo núcleo, para que en el procesamiento de interrupción pueda haber instrucciones privilegiadas.
* Ejecuta la rutina de atención a la interrupción.

**Dos posibilidades**:

* El sistema operativo determina que el proceso actual debe continuar con su ejecución ⇒ Se ha interrumpido un proceso de usuario para tratar una rutina del S.O. y continúa con el mismo proceso. Solo cambio de modo: usuario ⇒ núcleo ⇒ usuario
* El S.O. determina que se debe realizar un cambio de proceso ⇒ El control pasa a la rutina de sistema de cambio de proceso. Hay cambio de proceso.

**Cambio de modo**

* El cambio de modo no implica cambio del proceso que se esté ejecutando
* Se puede cambiar de modo, ejecutar instrucciones privilegiadas en modo núcleo y volver a ejecutar instrucciones del mismo proceso y por tanto:
  + No es necesario hacer cambio completo de proceso
  + Basta con guardar información del estado del procesador (contexto).

**Secuencia en un cambio de proceso**:

* Salva el contexto de P1 (BCP).
* Asigna al registro CP la dirección de comienzo del programa atención a la interrupción de cambio de proceso.
* Cambia a modo núcleo.
* Ejecuta la rutina de atención a la interrupción.
  + Actualiza el bloque de control del proceso que está en estado de ejecución (nuevo estado: listo/bloqueado/. . . ).
  + Mueve el bloque de control del proceso a la cola apropiada (listos/bloqueados/). Selecciona otro proceso para su ejecución P2.
  + Actualiza el BCP seleccionado (nuevo estado: ejecución).
  + Actualiza las estructuras de datos de la gestión de memoria (para traducción de direcciones).
  + Restaura el contexto del proceso seleccionado (actualizar registros del procesador), incluyendo modo (usuario).

**Ejecución del SO**

* Núcleo fuera de todo proceso:
  + Ejecuta el núcleo del sistema operativo fuera de cualquier proceso.
  + El código del sistema operativo se ejecuta como una entidad separada que opera en modo privilegiado.
* Ejecución dentro de los procesos de usuario:
  + Software del sistema operativo en el contexto de un proceso de usuario.
  + Un proceso se ejecuta en modo privilegiado cuando se ejecuta el código del sistema operativo.
* Sistema operativo basado en procesos:
  + Las funciones más importantes del núcleo se organizan en procesos separados.
  + Útil en un entorno de multiprocesador o de varios computadores.

**Procesos en UNIX**

Estados:

* **Ejecución en modo de usuario**: Ejecutando en modo de usuario.
* **Ejecución en modo del núcleo**: Ejecutando en modo de núcleo.
* **Listo para ejecutar y en memoria**: Listo para ejecutar tan pronto como el núcleo lo planifique.
* **Dormido y en memoria**: Incapaz de ejecutar hasta que se produzca un suceso; el proceso está en memoria principal.
* **Listo** para ejecutar y descargado: El proceso está listo para ejecutar, pero se debe cargar el proceso en memoria principal antes de que el núcleo pueda planificarlo para la ejecución.
* **Dormido y descargado**: El proceso está esperando un suceso y ha sido expulsado al almacenamiento secundario.
* **Expulsado**: El proceso retorna del modo del núcleo al modo de usuario, pero el núcleo lo expulsa y realiza un cambio de contexto para planificar otro proceso.
* **Creado**: El proceso está recién creado y aún no está listo para ejecutar.
* **Zombie**: El proceso ya no existe, pero deja un registro para que lo recoja el proceso padre.

En UNIX la mayor parte del sistema operativo se ejecuta como procesos de usuario. Existen usuarios privilegiados.

**Características**:

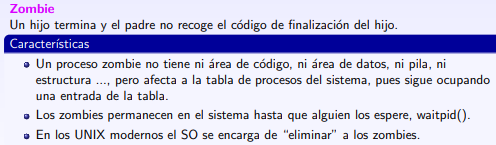
* Un proceso UNIX tiene dos áreas principales de memoria: ejecución y control.
* El área de ejecución es el programa (texto) más las zonas de memoria reservadas para la pila y los datos. Está en la zona de memoria virtual asignada al proceso.
* El área de control
  + contiene los bloques de control que conserva la información acerca del proceso. Esta zona la gestiona el núcleo del SO
  + tiene información que debe residir en memoria continuamente e información que puede ser enviada a disco con el área de ejecución cuando el planificador de procesos decida.

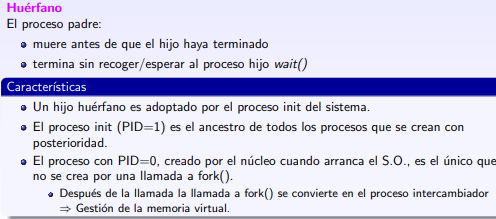
**Creación de un proceso en UNIX**:

* Un proceso se crea mediante una llamada a sistema fork().
* Efectos del fork:
  + Crea una nueva entrada (estructura de proceso) en la tabla de procesos, en la que da un nuevo ID al proceso hijo.
  + Asigna memoria y copia en ella los segmentos de datos y la pila del padre, junto a la cual copia la estructura de usuario.
  + Pone al hijo en estado Listo para ejecutar.
  + El código del padre y del hijo son idénticos salvo el valor de retorno del fork.
  + El hijo recibe 0 como retorno del fork.
  + El padre recibe el pid del hijo como retorno del fork.

**Proceso de creación**:

* Asignar una nueva entrada a la tabla principal de procesos.
* Asignar espacio para la imagen del proceso.
* Inicializar el bloque de control de proceso.
* Establecer los enlaces apropiados
  + Ej: Añadir un proceso nuevo a una lista enlazada que se utiliza como cola de planificación.
* Crear o ampliar otras estructuras de datos
  + Ej: Mantener un archivo de contabilidad (tiempo ejecución del hijo igual a cero), actualizar la tabla de gestión de memoria…

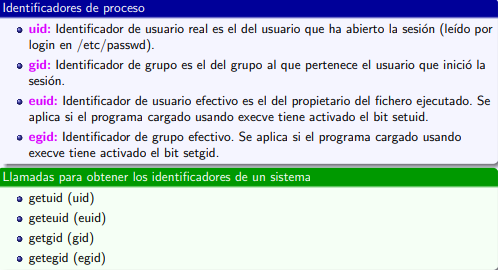




Otras dos funciones importantes en la creación de procesos son:

**execve**: es realmente una familia completa de funciones. Son usadas por un proceso para cargar un nuevo ejecutable binario en el espacio virtual de memoria del proceso que hace la llamada.

**vfork**: es igual que fork, pero no copia los datos ni la pila.



**Comunicación entre procesos**

Los procesos se pueden comunicar mediante tuberías (pipes) que deben ser creadas antes de la ejecución del fork.

A los pipes se accede a través de un descriptor de fichero, como cualquier otro “fichero”. Tienen tamaño variable.

**Señales**:

* Son recursos (similares a las interrupciones) para atender algunas circunstancias excepcionales.
* Origen: teclado, error en el proceso, eventos, llamada a kill.
* Señales principales:
  + SIGINT: interrupción desde teclado (normalmente ˆC). Detiene un comando antes de que termine su ejecución.
  + SIGQUIT: desde teclado (ˆbs). Detiene la ejecución y vuelca el contenido de la memoria en un fichero core.
  + SIGSEGV: Acceso no válido a memoria.
  + SIGKILL: Termina con un proceso (no se puede ignorar o atrapar).
* Las señales pueden ser ignoradas o atendidas por una rutina en el proceso de usuario.
* Las señales pueden perderse y no se acumulan.

**Llamadas al sistema:**

**pid = fork()**: crea un proceso hijo, que es igual al padre. Devuelve pid=0 en el hijo y el pid del hijo al padre.

* **s = waitpid(pid,&status,opts):** ejecutada por el proceso padre, le pone en espera hasta la terminación del hijo. El proceso padre puede esperar a la terminación de un hijo o de cualquiera (pid=-1) y recibe el estado del proceso saliente (status). El padre puede quedar bloqueado o regresar si ningún proceso ha terminado (opt).
* **s = execve(name,argv,envp):** reemplaza el código del proceso llamante por un nuevo código. Ejecuta el proceso name, con un puntero de argumentos argv y un puntero a entorno envp (nombre=valor).
* **exit(status):** es enviada por un proceso que termina su ejecución. Devuelve el estado de conclusión y salida del proceso hijo al padre, que está esperando mediante un waitpid.
* **s=sigaction(sig,&act,&oact):** manifiesta la disponibilidad del proceso frente a señales, y especifica la acción a llevar a cabo como respuesta. **sig** indica qué señal se atiende; **act** apunta a una estructura que contiene un puntero al procedimiento de manejo de la señal; **oact** apunta a una estructura en la que el sistema devuelve información acerca del manejo de señales, por si debe ser restaurado.
* **s = kill(pid,sig):** envía una señal a otro proceso relacionado con él.
* **s = alarm(segundos):** genera una señal después de un periodo de tiempo especificado.
* **pause()**: interrumpe la ejecución del proceso hasta la llegada de la siguiente señal.

**Hilos**

Características

**Unidad de propiedad de los recursos**:

* Proceso necesita espacio de direcciones virtuales para mantener su imagen
* Proceso tiene asignados recursos en un momento determinado

**Unidad de expedición**:

* Procesos sigue un camino de ejecución que puede ser intercalada con la de otros procesos.

Estas dos características son tratadas de **manera independiente** por el sistema operativo:

* La unidad de expedición se conoce como hilo
* La unidad de propiedad de los recursos se conoce como proceso o tarea

**Sistema Multihilo**

Sistema operativo que mantiene varios hilos de ejecución dentro de un mismo proceso

* MS-DOS soporta un solo hilo.
* UNIX soporta múltiples procesos de usuarios, pero solo un hilo por proceso.
* Windows 2000, Solaris, Linux, Mach, y OS/2 soportan múltiples hilos.

**Proceso:**

* Tiene un espacio de direcciones virtuales, que contiene la imagen del proceso.
* Acceso protegido a los procesadores, a otros procesos, archivos y a recursos de E/S.
* Tiene un hilo o más.

**Hilo:**

* Bloque de control de hilo: posee un estado de ejecución (Ejecución, Listo, etc.) y el contexto del procesador se salva cuando no está ejecutando.
* Tiene una pila de ejecución: almacenamiento para las variables locales.
* Acceso a la memoria y a los recursos del proceso, compartidos con todos los hilos del mismo.

**Beneficios de los hilos:**

* Se tarda menos tiempo en crear un nuevo hilo en un proceso existente.
* Se tarda menos tiempo en terminar un hilo que un proceso.
* Se tarda menos tiempo en cambiar entre dos hilos de un mismo proceso.
* Puesto que los hilos de un mismo proceso comparten recursos (memoria, archivos, etc.), pueden comunicarse entre sí sin invocar al núcleo → coordinación más rápida → mayor eficiencia.

**Uso de los hilos:**

En un sistema multihilo y monousuario los hilos permiten:

* Trabajo interactivo y en segundo plano.
  + Ej: calcular – recibir datos de teclado (bloques)
* Procesamiento asíncrono.
  + Ej: cada X tiempo copia de seguridad. No son necesarias comprobaciones externas de tiempo, ni coordinación entre E/S, porque el hilo se ocupa de ello
* Aceleración de la ejecución.
  + Ej: procesar lotes – leer siguiente
* Estructuración modular de los programas.
  + Ej: varias actividades a la vez y/o varios dispositivos

**Estados de un hilo**

**Creación**: Se crea un nuevo hilo

* Crear un proceso = crear primer hilo
* Un hilo puede crear otros hilos
* El nuevo hilo tiene su propio contexto y espacio de pila. Pasa a la cola de listos.

**Bloqueo:**

* El hilo necesita esperar suceso.
* Se guarda el contexto del hilo (registros, contador de programa, puntero de pila).
* No tienen por qué bloquearse el resto de los hilos del mismo proceso.

**Desbloqueo**: El suceso ocurre, el hilo pasa a cola de listos

**Terminación**: Se liberan su contexto (del hilo) y sus pilas.

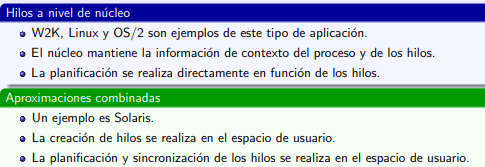
La suspensión de un proceso implica la suspensión de todos los hilos de un proceso, puesto que todos comparten el mismo espacio de direcciones.

La terminación de un proceso supone terminar con todos los hilos dentro de dicho proceso.

**Tipos de Hilos:**

**Hilos a nivel de usuario:**

* La aplicación (biblioteca de hilos) realiza todo el trabajo de gestión de hilos.
* El núcleo no tiene conocimiento de la existencia de hilos.
* Biblioteca de hilos - código para ...
  + ... crear y destruir hilos
  + ... intercambiar mensajes y datos entre hilos
  + ... planificar ejecución de hilos
  + ... salvar y restaurar el contexto de los hilos.



**Ventajas**:

* El intercambio de hilos no necesita privilegios del modo núcleo.
  + Gestión de hilos en el espacio de direcciones de usuario.
  + Se evitan 2 cambios de modo: usuario → núcleo → usuario.
* Se puede realizar planificación específica a nivel de hilos.
* Los hilos se pueden ejecutar en cualquier S.O.
  + Biblioteca hilos = utilidades de aplicación

**Desventajas:**

* Las llamadas al sistema suelen ser bloqueantes en el S.O.
  + Un hilo ejecuta llamada al sistema → bloqueo del resto de hilos (todo el proceso).
* No se aprovechan las ventajas de los multiprocesadores.
  + Núcleo asigna el procesador a un proceso → solo 1 hilo del proceso en ejecución
  + Los hilos a nivel de usuario son “invisibles” para el S.O

**Hilos a nivel de núcleo:**

* El núcleo mantiene la información de contexto del proceso y de los hilos
  + Hilos visibles y gestionados por el núcleo.
* La planificación se realiza en función de los hilos → hilos del mismo o distinto proceso compiten por el procesador.

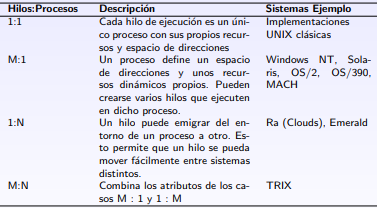
**Ventajas:** Si hay varios procesadores, es posible ejecutar en paralelo hilos del mismo proceso. Las propias funciones del núcleo pueden ser multihilo.

**Desventajas** Es necesario cambiar a modo núcleo para pasar de un hilo a otro (incluso con hilos de un mismo proceso).

**Aproximaciones combinadas:**

* Los hilos se crean en el espacio de usuario.
* La planificación y sincronización se realizan en el espacio de usuario.
* Varios hilos de usuario se asocian con varios hilos a nivel de núcleo.
  + De igual número o distinto, ajustable por el programador.
  + Los puede planificar el núcleo.
* Esquema combinado de planificación Ej.: Solaris.

**Ventajas:** Hilos de un mismo proceso se pueden ejecutar en paralelo en varios procesadores Las llamadas al sistema bloqueantes no necesitan bloquear todo el proceso.



**Planificación**

Objetivos de la planificación

Entre los objetivos básicos de la multiprogramación destacan:

* Mejora del tiempo de respuesta.
* Aumento de la productividad.
* Aumento de la eficiencia del procesador.

Tipos:

**Planificación a largo plazo:** decisión de añadir procesos al conjunto de procesos a ejecutar.

**Planificación a medio plazo:** decisión de añadir procesos al conjunto de procesos que se encuentran parcial o completamente en memoria.

**Planificación a corto plazo:** decisión sobre qué proceso disponible será ejecutado en el procesador.

**Planificación de E/S:** decisión sobre qué solicitud de E/S pendiente será tratada por un dispositivo de E/S disponible. Diagrama

Descripción generada automáticamente



**Planificación a largo plazo**

* Controla el grado de multiprogramación.
  + Limitar el número para dar buen servicio.
  + Nuevo: cada vez que termina un proceso o si el porcentaje de utilización del procesador es bajo
* Determina cuáles son los programas admitidos en el sistema.
  + Algoritmos de planificación Simples (ej., FIFO-FCFS). Por rendimiento del sistema: prioridades, carga procesador, carga E/S, recurso E/S a solicitar,
* Cuantos más procesos se crean, mayor sobrecarga del sistema: mayor es el tiempo medio de finalización de los procesos y mayor estrés, en general, en los recursos del sistema.

**Planificación a medio plazo**

Forma parte de la función de intercambio Gestión de memoria, Memoria Virtual, Estados Suspendidos

Se basa en la necesidad de controlar el grado de multiprogramación

**Planificación a corto plazo**

También conocido como distribuidor o “dispatcher”:

Es el de ejecución más frecuente.

Se ejecuta cuando ocurre un suceso de entre los siguientes:

Interrupciones del reloj.

Interrupciones de E/S.

Llamadas bloqueantes al sistema operativo.

Señales.

**Criterios de planificación a corto plazo:**

**Orientados al usuario.**

Cuantitativos.

* Tiempo de retorno:
  + Desde el lanzamiento hasta la finalización de un proceso.
  + Apropiado para trabajos por lotes.
* Tiempo de respuesta:
  + Desde que se emite la solicitud hasta que la respuesta aparece en la salida.
  + Apropiada para procesos interactivos
* Plazos:
  + Si hay plazos, maximizar porcentaje de plazos cumplidos
  + Caminos críticos: a seguir si se quieren cumplir los requisitos.
  + Apropiado para procesos en tiempo real
  + Cualitativos .

Cualitativos:

Previsibilidad: Tiempo y coste independiente de la carga del sistema

**Orientados al sistema**

Cuantitativos:

* Productividad: Maximizar no procesos por unidad de tiempo.
* Utilización del procesador
  + Importante en sistemas compartidos caros, menos importante en monousuario y en tiempo real.

Cualitativos:

* Equidad.
* No inanición.
* Prioridades: si hay, favorecer a procesos con mayor prioridad
* Equilibrio de ocupación de recursos ⇒ Afecta también a la planificación a largo y medio plazo.
  + Mantener ocupados los recursos.
  + Favorecer procesos que no usen recursos sobrecargados.

**Prioridades**

* El planificador seleccionará siempre a un proceso de mayor prioridad antes que a los de menor prioridad.
* Tiene múltiples colas de procesos en estado listo para representar cada nivel de prioridad. Los procesos de prioridad más baja pueden sufrir inanición.
* Solución: permitir que un proceso cambie su prioridad en función de su edad o su historial de ejecución.

**Cómo se selecciona el siguiente proceso a ejecutar:** Es necesario tener en cuenta múltiples características. Por ejemplo:

* Prioridades
* Necesidades de recursos
* Características de ejecución:
  + Tiempo en el sistema, Tiempo ejecutado, Tiempo total estimado.

**Modo de selección**

Momento en que se aplica la función de selección.

* No preferente, no expulsiva, (apropiativa):
  + Una vez que el proceso pasa al estado de Ejecución, continua ejecutando hasta que termina, se bloquea en espera de una E/S o solicita el servicio del SO.
* Preferente, expulsiva, (no apropiativa):
  + El proceso que se está ejecutando actualmente puede ser interrumpido y pasado al estado de Listos por el sistema operativo.
    - Nuevo proceso.
    - Proceso de mayor prioridad sale de bloqueo
    - Interrupción de reloj.
  + Permiten dar un mejor servicio ya que evitan que un proceso pueda monopolizar el procesador durante mucho tiempo.
  + Mayor coste: más cambios de contexto.

**Algoritmos de planificación:**

Existen múltiples algoritmos de planificación. En esta sección vamos a ver:

FCFS (First-come, First-served)

Turno rotatorio (Round-Robin)

SPN (Shortest Process Next)

SRT (Shortest Remaining Time)

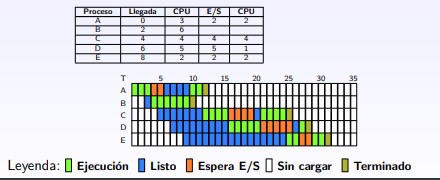
HRRN (Highest Response Ratio Next)

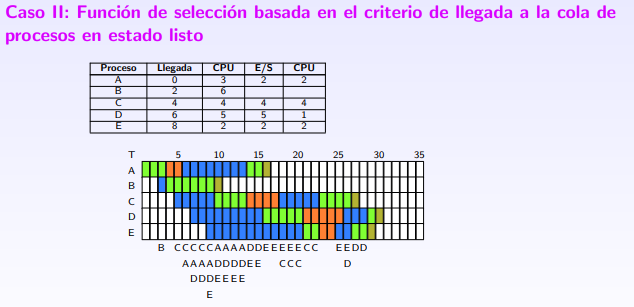
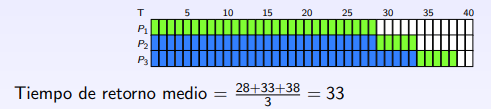
Realimentación

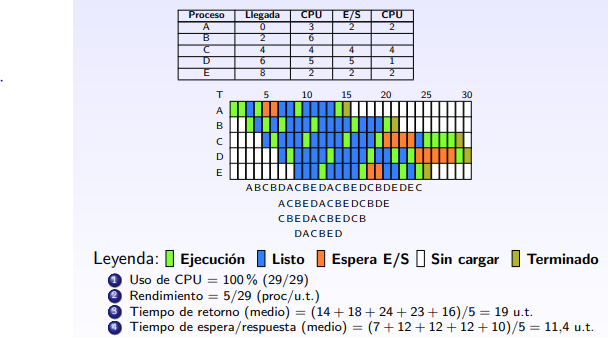
Reparto equitativo

Planificación garantizad

**FCFS (First-come, First-served)**: Servicio por orden de llegada (First Come First Served) FCFS. Cuando un proceso solicita uso de CPU (pasa a estar listo para ejecución) su BCP se pone el último en una cola (FIFO) de los procesos en espera de tiempo de CPU. Es un algoritmo apropiativo.

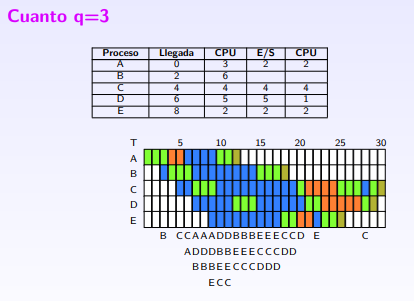




Efecto convoy, predominio de los procesos que usan CPU frente a los que usan E/S

**Round-Robin, Turno Rotatorio:**

* Periódicamente, se genera una interrupción de reloj.
* Cuando se genera la interrupción, el proceso que está en ejecución se sitúa en la cola de procesos en estado listo y se selecciona el siguiente trabajo (política expulsiva, preferente, no apropiativa)
* Está diseñado específicamente para sistemas de tiempo compartido. Se asigna un cuanto de tiempo (10-100 ms.) de igual duración a todos los procesos listos para ser ejecutados. Entre ellos, la selección se realiza mediante una cola FIFO (el primero que llega a la cola es el primero que se selecciona).
* Parámetro crítico: tamaño del cuánto. La efectividad depende del tamaño del cuánto, pero hay que tener en cuenta el tiempo dedicado al cambio de proceso.
* Se basa en el orden de llegada a la cola de listos.

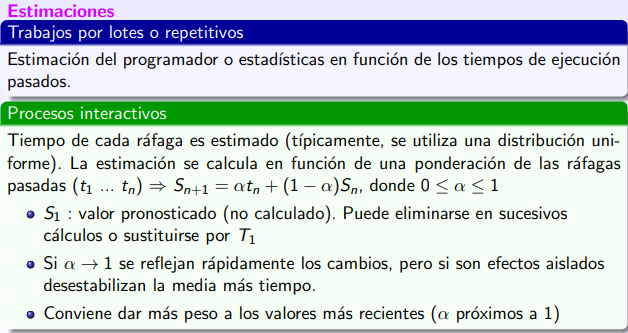


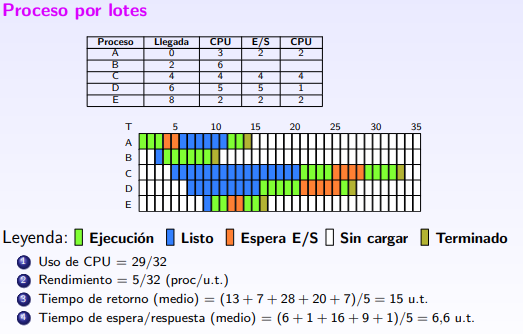
Parámetro crítico de diseño: longitud del cuanto,

* Si es muy pequeño los procesos cortos pasan rápidamente, pero hay sobrecarga del procesador (gestión interrupciones de reloj, planificación, expedición)
* Si es muy grande degenera en FCFS.
* Referencia: debe ser algo mayor que el tiempo necesario para una interacción normal.
* Efectivo en sistemas de carácter general, de tiempo compartido o procesos de transacciones.
* Favorece procesos con carga de procesador frente a procesos con carga de E/S (estos no aprovechan el cuanto).

**Primero el proceso más corto (SPN):**

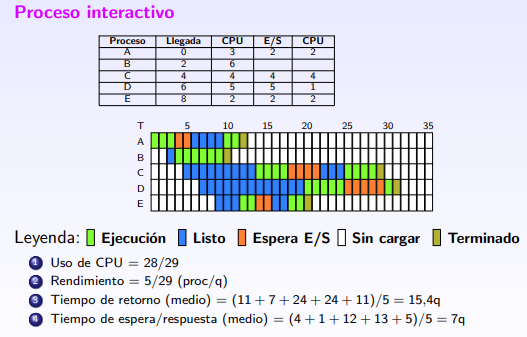
* Suele tener una política no expulsiva, no preferente (apropiativa).
* Se selecciona el proceso con menor tiempo esperado de ejecución. ¿Estimación de tiempo de ejecución?
* Implementación: Un proceso corto saltará a la cabeza de la cola, sobrepasando a trabajos largos.
* Inconvenientes: Se reduce la previsibilidad de los procesos largos. Si la estimación de tiempo del proceso no es correcta, el sistema puede abandonar el trabajo. Posibilidad de inanición para los procesos largos.





**Proceso interactivo:**

* No se conoce cuánto se necesitará en la siguiente ráfaga de CPU (depende de la interacción del usuario), por lo que se estima. Si la estimación sigue una distribución uniforme: Sn+1 = αtn + (1 − α)Sn.
* Hay múltiples formas para estimar el tiempo inicial de un proceso nuevo S1, por ejemplo:
  + No sabemos nada sobre las ráfagas: S1 = 0 (los procesos nuevos son preferentes).
  + Tomando S1 = T1 (problema: hay que saber el T1)
  + Tomando S1 = cte (ej: media de las ráfagas de procesos interactivos anteriores en el sistema).



**Características:**

* Mejora el rendimiento global: tiempo de retorno y tiempo de espera/respuesta.
* Es posible la inanición para los procesos largos.
* No es conveniente para tiempo compartido o procesamiento de transacciones (por ser apropiativa).
* Se reduce la previsibilidad de los procesos largos (pueden variar mucho con pequeños cambios en las condiciones).

**Menor tiempo restante (SRT):**

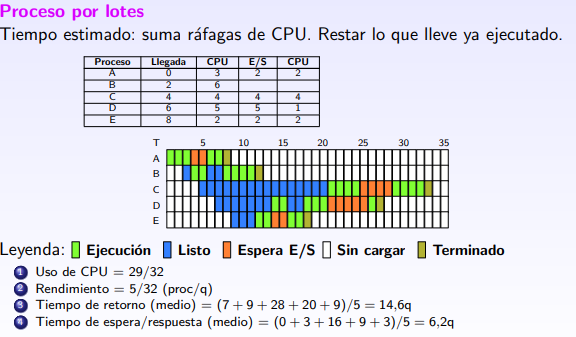
* Es una versión preferente de la política de primero el proceso más corto.
* Debe estimar el tiempo de proceso.

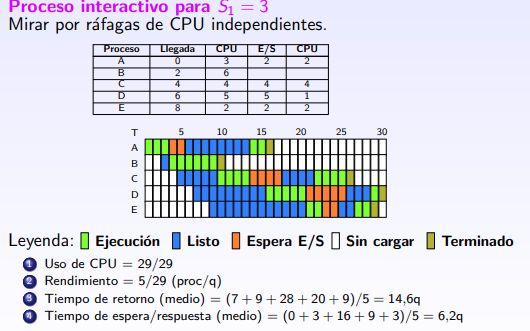
**Función de selección**.

Mínimo tiempo restante de ejecución (t. total – t. consumido).

**Modo de decisión**.

Preferente, expulsiva (no apropiativa): cada vez que llega un proceso a la cola de listos se repiten los cálculos y se elige para su ejecución con el que corresponda.





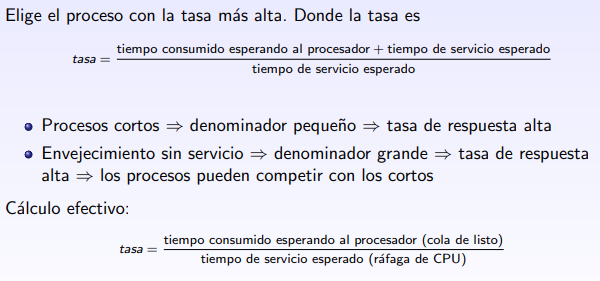
ç

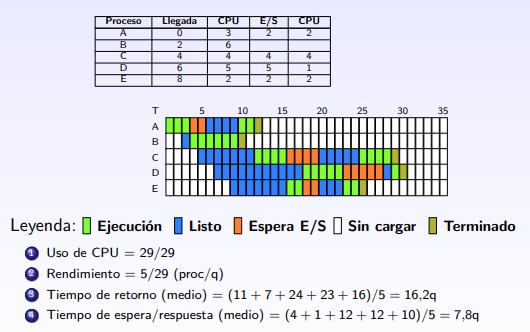
SRT favorece a los procesos cortos

**Ventaja:** No genera interrupciones adicionales (vs. Round Robin)

**Desventaja:** Debe contabilizar los tiempos de servicio transcurridos ⇒ sobrecarga

**HRRN**:





Realimentación Multinivel:

* Penaliza a los trabajos que han estado ejecutándose durante más tiempo.
* No se conoce el tiempo de ejecución restante del proceso.
* Política FIFO no apropiativo (FIFO preferente).



**Problema**: Los procesos largos: llevados gradualmente hacia abajo. Problema: pueden sufrir inanición en colas de prioridad baja si llegan muchos procesos cortos continuamente.

**Solución**: Cuanta menor es la prioridad se pueden asignar más cuantos de tiempo de ejecución. Tras cierto tiempo de espera en cola, se le cambia a una cola de prioridad mayor.

Imagen que contiene Gráfico

Descripción generada automáticamente

**Planificación Garantizada:**

* A cada proceso se le garantiza un uso equitativo de la CPU (1/n, siendo n el número de procesos en espera de ser ejecutados).
* Cada vez que un proceso va a ser asignado tiempo de CPU se comprueba la relación tiempo real/tiempo prometido de todos los procesos y se adjudica la CPU a aquel proceso que tiene el ratio más pequeña.

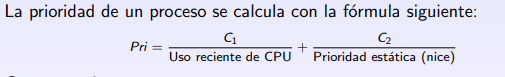
**Planificación en UNIX**

**Características:**

* Emplea realimentación multinivel usando turno rotatorio en cada una de las colas de prioridad.
* La prioridad de cada proceso se calcula cada segundo.
* La prioridad base divide los procesos en bandas fijas de prioridad.
* Se utiliza un factor de ajuste para impedir que un proceso salga fuera de la banda que tiene asignada.

**Proceso:**

* Cada segundo (1s) el planificador recalcula las prioridades de los procesos y los organiza en niveles de prioridad en función de dichos valores.
* Cada décima de segundo (0.1 s) el planificador selecciona el proceso que tenga máxima prioridad y le asigna tiempo de CPU.
* Si el proceso termina su cuanto de ejecución (no hay bloqueo), el proceso pasa a la cola de su nivel de prioridad.
* Si el proceso se bloquea durante su cuanto, el planificador selecciona inmediatamente otro proceso y le asigna tiempo de CPU.
* Si un proceso retorna de una llamada al sistema y hay un proceso listo con mayor prioridad, el proceso de menor prioridad es desalojado de la CPU.
* Cada 4 centésimas de segundo (0.04 s) el planificador recalcula la prioridad del proceso que está usando tiempo de CPU



**Consecuencias:**

* La prioridad de los procesos disminuye si utilizan mucho tiempo de CPU en una ventana de tiempo determinada.
* Por el contrario, procesos con mucha demanda de E/S tenderán a tener prioridades altas.
* Los procesos con un valor de prioridad estática (nice) alto, tendrán menor prioridad.

**Prioridad por bandas. En orden decreciente de prioridad:**

* Intercambio.
* Control de dispositivos de E/S de bloques.
* Gestión de archivos.
* Control de dispositivos de E/S de caracteres.
* Procesos de usuario.

Imagen que contiene Tabla

Descripción generada automáticamente

Seleccionar criterio de optimización.

Métodos de evaluación:

**Modelado Determinista:**

* Medida (números exactos) de la carga de CPU proceso.
* Utilidad académica o en sistemas que ejecutan los mismos programas.

**Modelo de colas:**

* Estimación (probabilística) de la carga de CPU
* Requisitos de E/S de los sistemas y tiempos de llegada de procesos.
* Permite comparar los distintos algoritmos.
* Problema: Arbitrariedad del modelo matemático derivado de cada algoritmo.

**Simulaciones:** Medida de la respuesta de los algoritmos a secuencias generadas aleatoriamente, mediante distribuciones o eventos grabados

Apuntes Sistemas Operativos

**Introducción**

**Definición**:

Un sistema Operativo es un programa que controla la ejecución de los programas de aplicación permitiendo un acceso eficiente a recursos compartidos limitados. Actúa como interfaz entre las aplicaciones del usuario y el hardware:

* Simplifica la labor del programador.
* Una interfaz es un conjunto de comandos y/o métodos que permiten la intercomunicación del programa con:
  + Cualquier otro programa
  + Módulos del propio programa
  + Elementos internos o externos (los periféricos son controlados por interfaces)

**Objetivos de un Sistema Operativo**

* Comodidad: Hace que un computador sea más cómodo de utilizar. Abstracción
* Eficiencia: Permite un uso eficiente de los recursos de un sistema informático. Concurrencia
* Capacidad de evolución: Permite el desarrollo efectivo, la verificación y la introducción de nuevas funciones en el sistema sin interferir en los servicios. Modularidad

**Servicios de un Sistema Operativo**

* Creación de programas: Editores. Compiladores. Depuradores. Profilers.
* Acceso ordenado a los dispositivos de E/S.
* Acceso controlado a los archivos.
* Acceso al sistema.
* Detección y respuesta a errores.
  + Errores internos y externos del hardware: ´
    - Error de memoria.
    - Fallo de dispositivos.
  + Errores de software:
    - Desbordamiento aritmético. Overflow
    - Acceso a una posición prohibida de memoria. Segmentation Fault
  + Incapacidad del sistema operativo para satisfacer la solicitud de una aplicación
* Contabilidad:
  + Recoger estadísticas.
  + Supervisar su rendimiento.
  + Utilizado para anticiparse a las mejoras futuras.
  + Adaptabilidad: interfaces adaptativas.
  + Utilizado para los usuarios de cuotas.

**Características**

Funciona de la misma manera que el software normal de un computador:

Es un programa ejecutado por el procesador.

Compite, por tanto, por el procesador y otros recursos.

El sistema operativo abandona el control del procesador para ejecutar otros programas.

**El núcleo**

* Parte del sistema operativo que reside en la memoria principal.
* Incluye las funciones utilizadas con más frecuencia.
* Carece de estructura.
* Se suele denominar kernel (Sistemas Operativos relativamente antiguos: DOS, Windows, Unices). Microkernel o micronúcleo (Sistemas Operativos más avanzados: WNT, W2K). En este caso, el S.O. se ocupa sólo de unas pocas funciones muy relevantes (gestión de memoria, procesos, Inter Process Communication - IPC-), reduciendo el núcleo a su mínima expresión. El resto de las funciones del S.O. pasan a estar en el espacio de usuario.

**Mejoras en el Sistema**

Los sistemas operativos son dinámicos, necesitan puestas a punto motivadas por: Correcciones. Actualizaciones del hardware. Aparición de nuevos tipos de hardware. Nuevos servicios

Evolución de las estructuras del sistema: Sistemas Monolíticos, Sistemas en Capas Máquinas Virtuales, Sistemas Cliente/Servidor

Su estructura (o falta de ella) es la más común (MS-DOS, UNIX).

El S.O. es un conjunto de procedimientos que pueden llamarse mutuamente.

No hay modos usuario/núcleo (el hardware no lo permite).

PROBLEMAS: No hay ocultación de datos y es difícil de modificar y depurar. Todo el sistema operativo es un único programa.

Sistema por capas El S.O. se organiza en una jerarquía de capas, cada una cimentada en la que está por debajo.

El sistema en capas es una ayuda para el diseño ⇒ todas las partes del sistema se enlazan en un solo programa objeto. Problema: distribución de tareas en capas, baja eficiencia

Diagrama

Descripción generada automáticamente

**Máquinas Virtuales**

* Extensión del sistema por capas, donde cada proceso tiene la ilusión de estar ejecutándose en un sistema monoprogramado. Ej: IBM VM, VM/370 (1972). La ilusión se consigue mediante el uso de planificación de CPU y de memoria virtual.
* La Máquina virtual proporciona copias exactas del hardware subyacente.
* Ejemplo: JAVA Virtual Machine (JVM).

**Ventajas:**

El SO (virtual) no tiene problemas de seguridad y protección de recursos compartidos (memoria, disco, periféricos), ya que se ejecuta sobre una máquina (virtual) monoprogramada.

Coexistencia de distintos SSOO sobre un mismo hardware.

**Desventajas:**

Perdida de eficiencia en las “traducciones”.

Ejemplos:

Emuladores: Interpretes software de las instrucciones de la CPU que se simula. Ej: Ejecución de programas para MS-DOS (16 bits) en un pentium (32 bits) o wine para emular Windows sobre Linux.

VMWare, virtualbox o Xen: virtualización de sistemas operativos. JAVA Virtual Machine (JVM): Máquina virtual de aplicación.

**Modelo Cliente-Servidor**

* Simplificación del núcleo, al mover el código correspondiente de algunas funcionalidades a capas superiores.
* Acceso a los servicios mediante mensajes desde los procesos de usuario (clientes) a los procesos que controlan los distintos servicios (servidores) que se ejecutan en modo usuario.
* El núcleo se limita a encauzar los mensajes.
* Ejemplos: UNIX moderno, Linux (Linus Torvalds, 1991)

**Ventajas**

Facilidad de mantenimiento, debido a la modularización de los servicios. Robustez, al ejecutarse los procesos de servicio en modo usuario: un servicio puede dejar de funcionar sin que todo el sistema tenga que interrumpir. Extensión natural del modelo a sistemas distribuidos.

**Arquitectura micronúcleo**

Arquitectura micronúcleo Asigna solamente unas pocas funciones esenciales al núcleo: Espacios de direcciones.

Comunicación entre procesos (IPC).

Planificación básica de procesos.

Gestión de memoria.

**Sistema operativo distribuido**

* Colección de computadoras separadas físicamente y conectadas entre sí por una red de comunicaciones
* Cada máquina posee sus componentes de hardware y software que el programador percibe como un solo sistema. Proporciona la ilusión de un único espacio de memoria principal y un único espacio de memoria secundaria.
* Utilizado para el sistema de archivos distribuido. Permite almacenar y acceder a archivos remotos como si fueran locales, sin que se note perdidas en el rendimiento

**Diseño orientado a objetos**

* Añade extensiones modulares a un pequeño núcleo.
* El S.O. estará formado por un conjunto de objetos que proporcionan funcionalidades.
* Permite a los programadores personalizar un sistema operativo sin romper la integridad del sistema.

**Registros del procesador**

Visibles al usuario

* Permiten al programador minimizar referencias a memoria principal, optimizando el uso de estos registros
* De control y de estado
  + Usados por el procesador para controlar las operaciones del procesador
  + Usados por las rutinas del SO para controlar la ejecución de los programas
  + Ej: contador del programa

**Registros visibles al usuario**

* Pueden ser referenciados mediante lenguaje máquina
* Disponibles para todos los programas (de aplicación y del sistema)
* Tipos de registros
  + Datos
  + Direcciones (de memoria principal)
    - De instrucciones
    - De datos

Registros de direcciones (de instrucciones o de datos). Ejemplos:

* Registro de índice
  + Implica sumar un índice a un valor base para obtener la dirección efectiva
* Puntero de segmento
  + Cuando la memoria se divide en segmentos, se referencia mediante referencia al segmento particular + desplazamiento dentro del segmento
* Puntero de pila
  + Apunta a la cima (tope) de la pila

**Registros de control y de estado**

* Contador del programa (PC)
  + Contiene la dirección de la instrucción a ser leída
* Registro de instrucción (IR)
  + Contiene la última instrucción leída
* Palabra de estado del programa (PSW)
  + Contiene información de estado
    - Códigos de condición (flags)
      * Activados por el Hw como resultado de operaciones
      * Programa puede leerlos, pero no modificarlos
      * Ejs: resultado positivo, resultado negativo, cero (zero), desbordamiento (overflow)
    - Bit para habilitar/deshabilitar interrupciones
    - Bit indicando modo supervisor/usuario

Diagrama

Descripción generada automáticamente

Fase de búsqueda: procesador busca instrucción de memoria

Contador de programa (PC) mantiene dirección de la siguiente instrucción a leer. Se incrementa después de cada lectura

**Registro de instrucción (IR)**

En él se coloca la instrucción leída

Tipos de instrucciones

* Procesador-memoria
  + Se transfieren datos entre ambos (en cualquier dirección)
* Procesador-E/S
  + Se transfieren datos desde o hacia un dispositivo periférico (a través del módulo E/S)
* Tratamiento de datos
  + Operaciones aritméticas o lógicas sobre los datos
* Control
  + Altera la secuencia de la ejecución (saltos)

**Interrupciones**

¿Que son y para qué valen?

* Interrupción de la secuencia normal de ejecución
* Interrupción de un proceso causada por un evento externo al mismo de forma que el proceso podrá ser retomado
* Mejora la eficiencia del procesamiento
* Permite al procesador ejecutar otras instrucciones durante E/S

Tipos de interrupciones

* De programa
  + Desbordamiento aritmético
  + División por cero
  + Intento de ejecutar instrucción ilegal
  + Referencia a zona de memoria fuera del espacio de trabajo del usuario
* De reloj: para funciones periódicas
* De E/S: para indicar normalidad/error
* Fallo de Hw: cortes de energía, errores de paridad de memoria, etc.

**Atención a las interrupciones**

* El control se transfiere a este programa
* Determina la naturaleza de la interrupción y realiza las acciones necesarias (detener proceso, ... )
* Generalmente es una rutina que forma parte del SO
* Después de cada ejecución de instrucción, el procesador comprueba si hay interrupciones:
  + Si no hay, lee la siguiente instrucción del programa.
  + Si hay interrupción pendiente, suspende la ejecución del programa y ejecuta el manejador de interrupciones.

**Interrupciones múltiples**

Dos posibilidades

* Deshabilitar interrupciones
* Definir prioridades

Diagrama, Esquemático

Descripción generada automáticamente

**Concurrencia de Procesos: Exclusión Mutua y Sincronización**

**Afecta a**: la comunicación entre procesos. La compartición y competencia por los recursos. La sincronización de la ejecución de varios procesos. La asignación del tiempo de procesador a los procesos.

**Presente en**: la ejecución de múltiples aplicaciones: Multiprogramación. Las aplicaciones estructuradas: Algunas aplicaciones pueden implementarse eficazmente como un conjunto de procesos concurrentes. La estructura del sistema operativo: Algunos sistemas operativos están implementados como un conjunto de procesos hilos.

**Sincronización:** Los procesos coordinan sus actividades

**Sección crítica:** Región de código que sólo puede ser accedida por un proceso simultáneamente (variables compartidas).

**Exclusión mutua**: Sólo un proceso puede estar en la sección crítica accediendo a recursos compartidos

**Interbloqueo:** Varios procesos, todos tienen algo que otros esperan, y a su vez esperan algo de los otros.

**Círculo vicioso:** Procesos cambian continuamente de estado como respuesta a cambios en otros procesos, sin que sea útil (ej: liberar recurso)

**Condición de carrera:** Varios hilos/procesos leen y escriben dato compartido. El resultado final depende de la coordinación.

**Inanición:** Proceso que está listo y se le deniega siempre el acceso a un recurso compartido (procesador y otros).

La ejecución intercalada de procesos mejora rendimiento, pero la velocidad relativa de los procesos no puede predecirse puesto que depende de: Actividades de otros procesos Forma de tratar interrupciones Políticas de planificación.

La imprevisibilidad de la velocidad relativa de los procesos implica que es difícil

* Compartir recursos. Ej: orden de lecturas y escrituras. ...
* Gestionar la asignación óptima de recursos. Ej: recursos asignados a un proceso y este se bloquea, ¿recurso bloqueado? ⇒ posible interbloqueo
* Detectar errores de programación (resultados no deterministas, no reproducibles)

LD ACC, # (Carga el contenido de una dirección en el ACC).

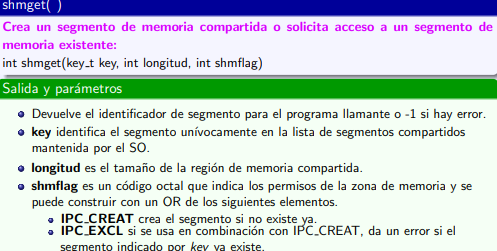
ACC++ (Incrementa el acumulador).

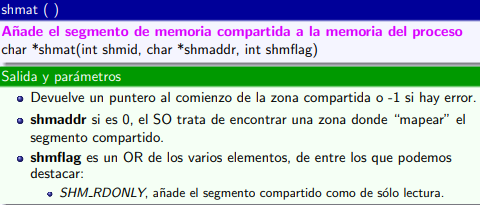
SV ACC, # (Almacena el valor del acumulador en una dirección).

El proveedor/productor produce información para el consumidor. La concurrencia se produce mediante el uso de buffers y variables compartidas (compartición de memoria) o mediante compartición de ficheros.

En el que se tienen las siguientes condiciones:

* Uno o más productores generan datos y los sitúan en un buffer.
* Un único consumidor saca elementos del buffer de uno en uno.
* Sólo un productor o consumidor puede acceder al buffer en un instante dado.





Seguir la pista de los distintos procesos activos.

**Asignar y retirar los recursos**:

Tiempo de procesador.

Memoria. Archivos.

Dispositivos de E/S.

Proteger los datos y los recursos físicos.

Los resultados de un proceso deben ser independientes de la velocidad relativa a la que se realiza la ejecución de otros procesos concurrentes

**Existen tres formas diferentes de interacción entre procesos:**

Los procesos no tienen conocimiento de los demás. Competencia

Los procesos tienen un conocimiento indirecto de los otros. Cooperación por compartimiento

Los procesos tienen un conocimiento directo de los otros (conocen el PID de los procesos). Cooperación por comunicación.

Cuando varios procesos entran en competencia se pueden producir las siguientes situaciones:

**Exclusión mutua:**

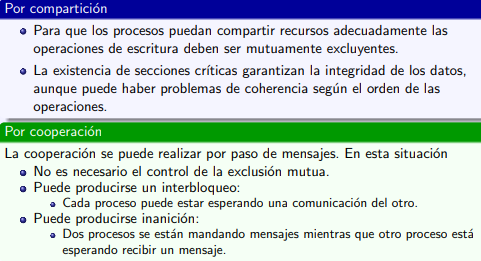
Recurso en sección crítica:

Sólo un programa puede acceder a su sección crítica en un momento dado.

Por ejemplo, solo se permite que un proceso envíe una orden a la impresora en un momento dado.

**Interbloqueo**. Ej: 2 procesos necesitan 2 recursos; se asigna 1 a cada 1; ambos esperan conseguir el otro (tmb con señales).

**Inanición**. Ej: 3 procesos necesitan 1 recurso; se va asignando al 1 y al 2 intermitentemente; el 3 sufre inanición



Requisitos Exclusión Mutua:

1 Sólo un proceso debe tener permiso para entrar en la sección crítica por un recurso en un instante dado.

2 No puede permitirse el interbloqueo o la inanición.

3 Cuando ningún proceso está en su sección cŕıtica, cualquier proceso que solicite entrar en la suya debe poder hacerlo sin dilación.

4 No se deben hacer suposiciones sobre la velocidad relativa de los procesos o el número de procesadores.

5 Un proceso permanece en su sección crítica sólo por un tiempo finito

Soluciones:

Software con Espera Activa

Hardware

Deshabilitar interrupciones

Instrucciones especiales de hardware

Con Soporte del SO o del lenguaje de programación (biblioteca): Semáforos Monitores

Un proceso está siempre en espera hasta que obtiene permiso (turno) para entrar en su sección crítica

Cada proceso puede examinar el estado del otro pero no lo puede alterar.

Cuando un proceso desea entrar en su sección crítica comprueba en primer lugar el otro proceso.

Si no hay otro proceso en su sección crítica fija su estado para la sección crítica.

Señala el interés por entrar en la sección crítica antes de comprobar otros procesos. Si otro proceso ha mostrado el interés de entrar en la sección crítica, el proceso queda bloqueado hasta que el otro proceso abandona la sección crítica.

Un proceso activa su señal para indicar que desea entrar en la sección crítica (interesado[i]=TRUE), pero debe estar listo para desactivar la variable señal. Se comprueban los otros procesos.

Si están en la sección crítica, la señal se desactiva (interesado[i]=FALSE) y luego se vuelve a activar para indicar que desea entrar en la sección crítica. Esto se repite hasta que el proceso puede entrar en la sección crítica.

Se impone un orden de actividad de los procesos.

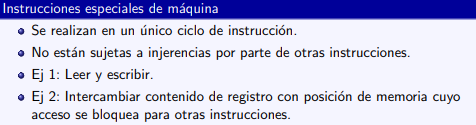
Si un proceso desea entrar en la sección crítica, debe activar su señal y puede que tenga que esperar a que llegue su turno.

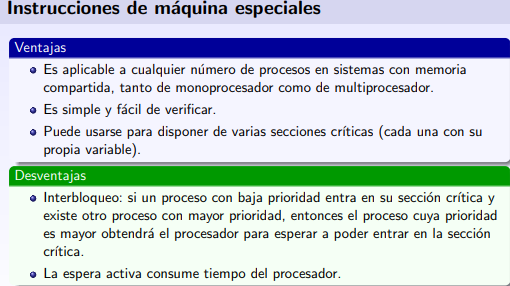
Para garantizar la exclusión mutua es suficiente con impedir que un proceso sea interrumpido.

Un proceso continuará ejecutándose hasta que solicite un servicio del sistema operativo o hasta que sea interrumpido.

Se limita la capacidad del procesador para intercalar programas.

Multiprocesador: Inhabilitar las interrupciones de un procesador no garantiza la exclusión mutua.





Con el fin de solucionar estos problemas en la sincronización se crean múltiples soluciones que veremos en las siguientes secciones y entre las que cabe destacar: Semáforos, Mensajes, Monitores.

**Semáforos:**

Los procesos se pueden coordinar mediante el traspaso de señales.

La señalización se tramita mediante una variable especial llamada semáforo.

Una señal se transmite mediante una operación atómica up( ) /signal( ).

Una señal se recibe mediante una operación atómica down( )/wait( ).

Un proceso en espera de recibir una señal es bloqueado hasta que tenga lugar la transmisión de la señal.

Los procesos en espera se organizan en una cola de procesos.

Dependiendo de la política de ordenamiento de procesos en espera:

Semáforo robusto: FIFO. Garantizan la no inanición y fuerzan un orden. (Linux) Semáforos débiles: otra política. No garantizan la no inanici´on (Mac OS X)

**Un semáforo se puede ver como una variable que tiene un valor entero:**

Puede iniciarse con un valor no negativo.

La operación down( )/wait( ) disminuye el valor del semáforo, si fuera posible (cota inferior).

La operación up( ) /signal( ) incrementa el valor del semáforo, si fuera posible (cota superior).

Si la variable solo puede tomar valores 0 y 1 el semáforo se denomina binario. En caso contrario, sem´aforo general o N–ario, valores de 0, . . . , N.

**DOWN down( )/wait( ):**

Comprueba el valor del semáforo antes de realizar la operación:

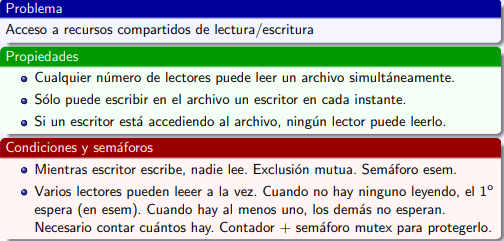
Si el semáforo > 0: decrementa el semáforo. El proceso continúa su ejecución.

Si semaforo ≡ 0, el proceso se echa a dormir (estado bloqueado) hasta que pueda decrementarlo.

**UP up( ) /signal( ):**

Si el semáforo > 0 → Incrementa el valor del semáforo (sin superar la cota superior, por ejemplo semáforos binarios).

Si semaforo ≡ 0 y no hay procesos en la cola del semáforo → Incrementa el valor del sem´aforo. Si semaforo ≡ 0 y hay procesos en la cola del semáforo → Despierta uno de los procesos bloqueados en este semáforo y termina su down( )/wait( ) → la variable no cambia de valor si había algún proceso durmiendo en ese semáforo



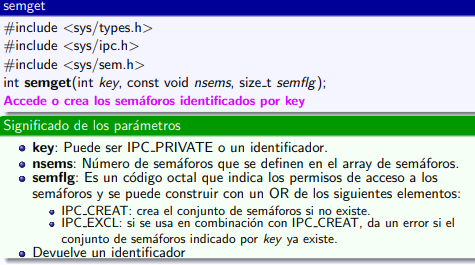
**Prioridad a los escritores**

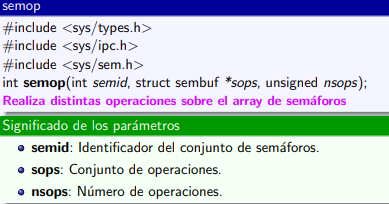
No se permite acceder a nuevo lector si un escritor declara su deseo de escribir.

Inhibir lecturas mientras haya algún escritor que desee escribir. Semáforo lsem (cola de lectores != cola de escritores)

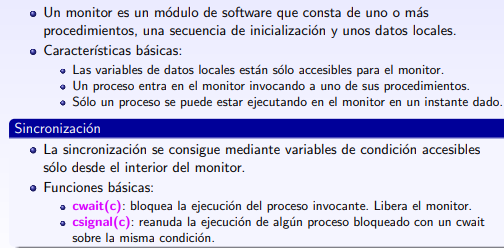
Necesario contar cuántos escritores desean escribir: variable contest. Necesario proteger acceso a dicha variable: semáforo mutex 2.

Si hay lectores esperando y escritor desea escribir, debe “colarse”. Separar lect/escr. Cola adicional de lectores en espera





**Monitores:**



**Mensajes:**

**Se utilizan como:**

Refuerzo de la exclusión mutua, para sincronizar procesos.

Medio de intercambio de información.

La funcionalidad de paso de mensajes se implementa mediante dos primitivas: **send (destino, mensaje). receive (origen, mensaje)**

El emisor y el receptor pueden ser bloqueantes o no bloqueantes (esperando a que se lea un mensaje o a que se escriba un nuevo mensaje).

Hay varias combinaciones posibles:

**Envío bloqueante, recepción bloqueante:**

Tanto el emisor como el receptor se bloquean hasta que se entrega el mensaje.

Esta técnica se conoce como rendezvous.

**Envío no bloqueante, recepción bloqueante:**

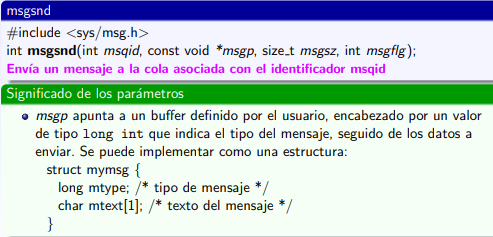
Permite que un proceso envíe uno o más mensajes a varios destinos tan rápido como sea posible.

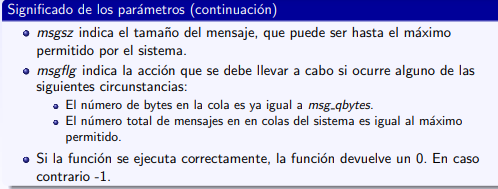
El receptor se bloquea hasta que llega el mensaje solicitado.

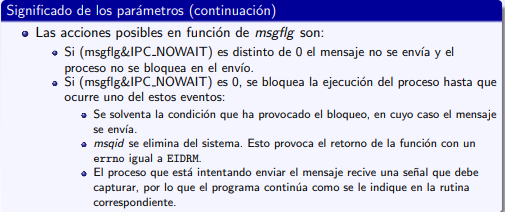
**Envío no bloqueante, recepción no bloqueante:**

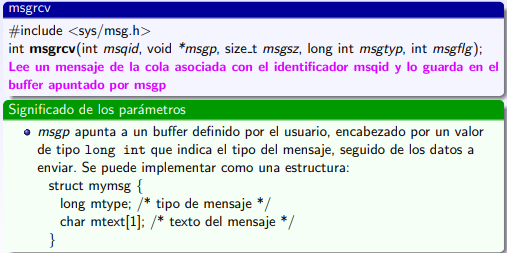
Nadie debe esperar

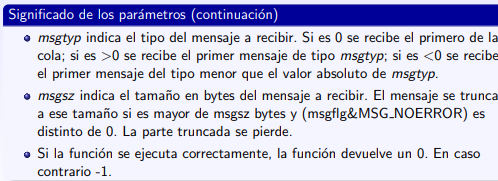


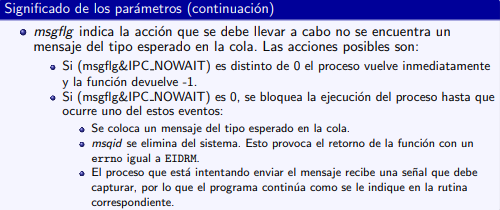












**Concurrencia de procesos: Interbloqueo e inanición**

**Bloqueo permanente** de un conjunto de procesos que compiten por los recursos o bien se comunican unos con otros.

Suponen necesidades conflictivas de recursos por parte de dos o más procesos.

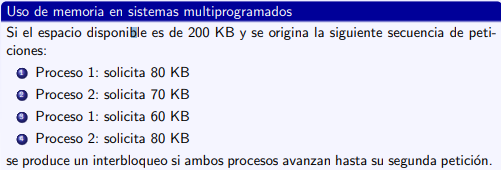
El bloqueo ocurre cuando un proceso monopoliza el acceso a un recurso y requiere otro recurso que ha sido ya asignado a un segundo proceso, que a su vez necesita el recurso monopolizado por el primer proceso.

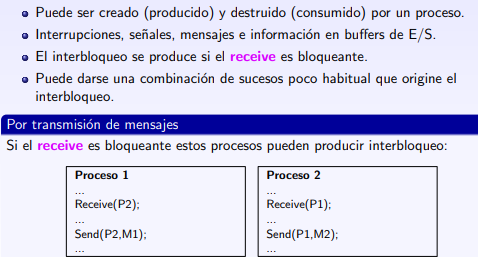
El interbloqueo es permanente porque ninguno de los eventos esperados puede producirse. No existe una solución eficiente.

Pueden ser usados por un proceso y no se agotan con el uso.

Los procesos obtienen unidades de recursos que liberan posteriormente para que otros procesos las reutilicen.

Procesadores, canales de E/S, memoria principal y secundaria, archivos, bases de datos y semáforos. El interbloqueo se produce si cada proceso retiene un recurso y solicita el otro





**Que se produzca interbloqueo depende de:**

Lógica del programa.

Dinámica de la ejecución:puede darse una combinación de sucesos poco habitual. El diseño de programas concurrentes es complejo.

A veces es difícil detectar errores de diseño.

**Exclusión mutua:** Sólo un proceso puede usar un recurso cada vez.

**Retención y espera:** Hay al menos un proceso que tiene asignado un recurso y se encuentra en espera de que otro proceso libere otro recurso.

**No apropiación / Sin expropiación:** No se puede forzar la expropiación de un recurso al proceso que lo tiene.

**Espera circular:** Si se dibuja el grafo de asignación y solicitud de recursos, tiene forma circular (ciclo).

Las tres primeras condiciones son necesarias pero no suficientes para que exista interbloqueo. La cuarta condición es una consecuencia potencial de las tres primeras

**No hacer nada** (algoritmo del avestruz) Ej. UNIX. Decisión de diseño, por motivos de flexibilidad. Ej. No de archivos abiertos.

**Uso de protocolos** que aseguren que el sistema no se bloqueará:

Mediante prevención: garantizar que una (o m) de las condiciones necesarias para la formación de interbloqueos no se cumpla.

Mediante evasión (basada en predicción): proporcionar al sistema información anticipada sobre las necesidades de recursos de los procesos, para que pueda predecir qué ocurrirá y encontrar secuencias de asignación de recursos que eviten los interbloqueos.

**Permitir que el sistema se bloquee** y proporcionar mecanismos de detección periódica y recuperación de interbloqueos.

La forma de garantizar que no se producirán interbloqueos se basa en garantizar que no se cumpla una de las condiciones necesarias para la formación de los mismos.

**Prevención de la condición de exclusión mutua:**

Problema: hay recursos que necesariamente requieren la exclusi´on mutua:

Archivos con permiso de escritura.

La tabla de procesos.

La tabla de nodo-i activos.

La memoria. El espacio en disco. . . .

En general, no se puede impedir la exclusión mutua. No es una solución

**Prevención de la condición de Retención y Espera:**

Se exige a los procesos que soliciten todos los recursos al comenzar su ejecución. Un proceso queda bloqueado hasta que se le conceden simultaneamente todas sus solicitudes.

Problemas:

Un proceso no ‘sabe’ a priori qué recursos va a utilizar.

Uso muy ineficiente de los recursos: un proceso acapara todos los recursos desde su inicio, aunque no se usen hasta el final del proceso.

Se puede producir la inanición: procesos no bloqueados pueden no tener nunca acceso a un recurso acaparado por otro.

P**revención contra la condición de no apropiación / sin expropiación:**

Si un proceso solicita un recurso no disponible, se interrumpe y además se le quitan todos los recursos que tenga asignados (se expropian).

El proceso se reinicia cuando se le pueden proporcionar todos los recursos que tenía más el que solicitó y no estaba disponible.

Es práctico y realizable con recursos cuyo estado puede ser fácilmente almacenable/recuperable:

Procesador (registros de CPU).

Memoria.

No es trivial expropiar sin causar efectos secundarios nocivos con otro tipo de recursos: Ficheros inconsistentes.

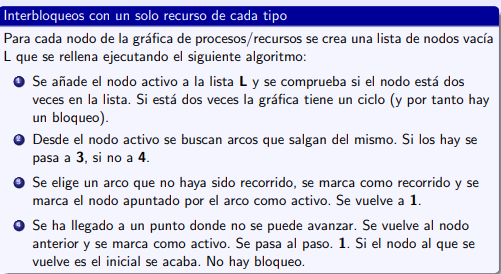
**Prevención contra la condición de espera circular:**

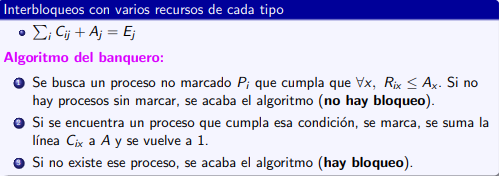
Hacer que los recursos sólo puedan utilizarse uno a uno. Esto no es posible en muchos casos: ej. copia de un fichero de cinta a disco.

Numerar los recursos, de modo que solo se puedan solicitar en un orden determinado.

Variante: Sólo se pueden solicitar recursos que tengan un número mayor del último que se haya solicitado.

Problema: cuando hay un número alto de recursos, no es fácil diseñar un orden que satisfaga a todos los procesos.





Cuando se detecta un interbloqueo, hay varias formas posibles de recuperación: **Recuperación mediante apropiación:** Se selecciona un proceso (o varios) y se le ‘requisan’ los recursos para cederlos a otros procesos bloqueados.

**Recuperación mediante Rollback:** Se almacena en un fichero periódicamente el estado del proceso (imagen de memoria) y estado de los recursos utilizados. Al producirse un bloqueo se detectan los recursos que son necesarios y un proceso que tenga alguno de esos recursos se interrumpe y se retrasa hasta el punto de verificación anterior a la solicitud del recurso (ojo: puede repetirse el bloqueo).

**Recuperación mediante eliminación de procesos. Hay dos posibilidades de actuación:**

Se abortan todos los procesos bloqueados.

Se selecciona un proceso bloqueado o uno no bloqueado que tenga recursos necesarios. Elegir procesos que se puedan reiniciar (si existen). Se termina de eliminar procesos cuando se desbloquea el sistema.

Criterios para la selección de procesos a terminar:

Menor prioridad.

Menor líneas de salida producidas al terminar.

Menor tiempo de CPU consumido.

Mayor tiempo restante.

Menor número de recursos asignados.

Tipo de recursos asignados.

Mayor número de recursos necesarios para terminación.

Problemas: Si el proceso está actualizando un archivo, su eliminación puede causar inconsistencias. Si el proceso está imprimiendo, se reinicia la impresora

Para poder evadir interbloqueos, antes de asignar un recurso hay que comprobar que al asignarlo no se conduce al interbloqueo. Se trata de predecir qué ocurrirá si se asigna. Algoritmo: se simula que se concede el recurso (se ’anota’ como asignado) y se aplica el algoritmo del banquero para comprobar si existe interbloqueo en caso de producirse esa asignación.

Si se encuentra una secuencia de asignación de recursos factible que no conduzca al interbloqueo (es decir, si tras la asignación del recurso al proceso el estado es seguro), el recurso se concede.

Requisito para la predicción: Es necesario que todos los procesos declaren sus necesidades totales de recursos al principio.

**Gestión y administración de la memoria**

**1. Introducción**

* La memoria es una matriz de palabras o bytes direccionables (accesibles mediante una dirección única) por la CPU para la carga de instrucciones o datos (ej: operandos) y para el almacenamiento de datos (ej: resultados).
* Las direcciones de un proceso pueden ser representadas de modo diferente en las sucesivas etapas del ciclo de un programa de usuario (compilación, carga y ejecución).
* Las direcciones que entiende el controlador de la memoria son direcciones absolutas. La conversión entre las distintas representaciones de las direcciones y las direcciones absolutas se denomina vinculación, y puede llevarse a cabo en cualquiera de las fases del programa.

Contiene:

* El sistema operativo: núcleo.
* Una zona de usuarios: imágenes de los procesos en ejecución.

**Direccionamiento**

* Dirección lógica: Dirección generada por la CPU.
* Dirección física**:** Dirección real percibida por la unidad de memoria.
* Si la vinculación de direcciones (lógica-física) se realiza en tiempo de compilación o de carga, entonces la dirección física = dirección lógica.
* Si la vinculación de direcciones se realiza en tiempo de ejecución (por ser el código reasignable), las direcciones lógicas y físicas difieren y llamamos a las direcciones lógicas direcciones virtuales.

**Gestion**

La necesidad de la gestión de la memoria se debe a que tenemos que:

* Ubicar (cargar) los procesos en la memoria.
* Subdividir la memoria para hacer sitio a varios procesos.Repartir eficientemente la memoria para introducir tantos procesos como sea posible.

**Requisitos de la gestión**

* **Reubicación**:
  + El programador no conoce que otros programas residirán en la memoria en el momento de la ejecución.
  + Mientras que se está ejecutando el programa, puede que se descargue al disco y que vuelva después de nuevo a la memoria principal, pero en una ubicación distinta a la anterior (reubicación).
  + Se deben traducir las referencias a la memoria encontradas en el código del programa a las direcciones físicas reales.
* **Protección:** 
  + El código de un proceso no puede hacer referencia a posiciones de memoria de otros procesos sin permiso.
  + Es imposible comprobar las direcciones absolutas de los programas, puesto que se desconoce la ubicación de un programa en la memoria principal.
  + Debe comprobarse durante la ejecución:
    - El sistema operativo no puede anticiparse a todas las referencias a la memoria que hará un programa.

La protección de memoria se puede realizar utilizando dos registros (base=reasignación y límite) que controlan el acceso a la memoria física.

Solo el SO puede modificar los registros de base y límite.

* **Compartición:**
  + Permite el acceso de varios procesos a la misma zona de la memoria principal.
    - Para la cooperación es necesario tener acceso compartido a estructuras de datos.
    - Es mejor permitir a cada proceso (persona) que acceda a la misma copia del programa, en lugar de tener cada uno su propia copia aparte.
* **Organización lógica:** 
  + La memoria está organizada como un espacio de direcciones unidimensional.
  + La mayoría de los programas se organizan en módulos.
    - Los módulos pueden escribirse y compilarse independientemente.
    - Pueden otorgarse un grado distinto de protección a cada uno de los módulos (solo lectura, solo ejecución).
    - Se pueden compartir módulos.
* **Organización física:**
  + Memoria principal: rápida, costosa, pequeña, volátil.
  + Memoria secundaria: más lenta, barata, gran capacidad, no volátil, ...
  + El flujo entre ambas es responsabilidad del S.O. (antes lo realizaba el programador):
    - La memoria disponible para un programa y sus datos puede ser insuficiente: la superposición es una técnica que permite que varios módulos sean asignados a la misma región de memoria.
    - El programador en principio no conoce el espacio disponible.

**2. Técnicas de gestión de memoria**

La tarea principal del sistema de gestión de memoriaes la carga de programas en memoria principal para su ejecución en el procesador (ubicar las imágenes de los procesos en memoria principal)

Técnicas simples:

Particionamiento (utilizada con distintas variantes en antiguos SO) Paginación simple Segmentación simple

Memoria Virtual:

Sistemas multiprogramados modernos Basada en segmentación y paginación

**2.1 Particiones estáticas**

**Condiciones:**

* El Sistema Operativo ocupa una parte fija de la memoria.
* El resto está disponible para ser usada por los procesos:
  + Límite en el número de procesos listos, bloqueados o en ejecución, determinado por el número de particiones existentes.
* Tamaños de las particiones y algoritmos de ubicación:
  + Particiones de igual tamaño
  + Particiones de distinto tamaño.

**De igual tamaño:**

**Condiciones:**

* Cualquier proceso cuyo tamaño sea menor o igual que el tamaño de la partición puede cargarse en cualquier partición libre.
* Si todas las particiones están ocupadas, el sistema operativo puede sacar un proceso de una partición (suspensión).

**Problemas:**

* Un programa puede que no se ajuste a una partición (porque sea mayor). El programador debe diseñar el programa mediante superposiciones.
* El uso de la memoria principal es ineficiente. Cualquier programa, aunque sea pequeño, ocupará una partición completa. Se produce el fenómeno de la fragmentación interna: existen espacios de memoria no utilizados, dentro de una partición, que no están disponibles para poderse utilizar/asignar.

**De distinto tamaño:**

**Reducen los problemas**.

* Se pueden alojar programas más grandes sin superposición.
* Se reduce el desperdicio de memoria producido por programas pequeños.

**Problemas:**

* Caben menos programas grandes sin superposición.
* Si hay muchos programas pequeños la fragmentación interna es muy grande en las particiones grandes.

**Dos formas de gestionarlas:** Una cola por partición. Una cola única.

**Una cola por partición**

Método: Ubicar cada proceso en la partición de tamaño menor en la que quepa. Si la partición no está libre, espera en la cola correspondiente. Hace falta una cola para cada partición.

**Ventaja:** Se minimiza la memoria desaprovechada dentro de cada partición (fragmentación interna)

**Desventaja:** Pueden existir procesos suspendidos esperando en colas correspondientes a las particiones que mejor se ajustan a su tamaño, habiendo otras particiones vacías.

**Una cola única**

Método: Cuando se va a cargar un proceso, se selecciona la partición más pequeña disponible donde quepa.   
Si todas las particiones están ocupadas, los criterios para suspender otro proceso se relacionan con:

Usar la partición más ajustada (minimiza la fragmentación interna).

Considerar las prioridades de procesos ya cargados.

Suspender a los procesos bloqueados antes que a los procesos listos.

**Desventajas:**

* El número y el tamaño de las particiones especificadas al principio limitan el número y el tamaño máximo de procesos activos (no suspendidos)
* Si hay muchos procesos pequeños no se aprovecha bien la memoria (fragmentación interna)
* Si hay muchos procesos grandes, las particiones pequeñas pueden resultar infrautilizada.

**2.2 Particiones dinámicas**

**Definición**

Las particiones son variables en número y longitud. Al proceso se le asigna exactamente tanta memoria como se necesite.

**Fragmentación externa**:

* En un sistema de particiones dinámicas, tras la reubicación de procesos, se van generando huecos en memoria entre las distintas particiones que, unidos, serían útiles para albergar procesos, pero, separados, no sirven. Este fenómeno se denomina fragmentación externa.
* Se debe usar la compactación para desplazar los procesos de modo que queden situados en particiones contiguas (normalmente en el comienzo de la memoria) y toda la memoria libre quede junta en un único bloque grande (normalmente al final de la memoria).

**Fragmentación interna:**

* Es el desperdicio de memoria que se produce al asignar a un proceso más memoria de la que solicita (por conveniencia o simplicidad en la asignación).
* No es frecuente encontrar fragmentación externa en sistemas de gestión de memoria basados en particiones dinámicas, pues normalmente se asigna a cada proceso justo el espacio que necesita.

**Ubicación con mapa de bits:**

* La memoria principal está dividida en unidades de asignación (bloques).
* Para conocer qué bloques están libres/ocupados, se utiliza un mapa de bits.
* Cada bloque se representa mediante un bit en el mapa de bits
  + Compromiso entre el tamaño del mapa y la optimización de la gestión.
* La asignación de espacio a un proceso de tamaño k unidades consiste en una búsqueda de k ceros contiguos en el mapa de bits: LENTO.

**Ubicación con listas enlazadas:** La memoria está representada por una lista ligada de zonas de memoria ocupadas (P) y libres (H).

**Ventajas:**

* La búsqueda de huecos es más rápida, puesto que cada elemento de la lista contiene el tamaño de este.
* Es fácil reorganizar la lista al terminar un proceso o al introducir un proceso nuevo.

**Ubicación por ajuste**

**Best-fit:**

* Elige el bloque de tamaño más próximo al solicitado.
* Proporciona en general los peores resultados.
* Como este algoritmo busca el hueco más pequeño para el proceso, garantiza que el fragmento que se deja es lo más pequeño posible y, por lo tanto, suele ser insuficiente en el futuro y se debe compactar más frecuentemente.

**First-fit**:

* Es más rápido.
* Puede tener varios procesos cargados en el extremo inicial de la memoria que es necesario recorrer cuando se intente encontrar un bloque libre.

**Next-fit:**

* Lleva frecuentemente a la asignación de un bloque de memoria de la última ubicación, donde se encuentra el bloque más grande.
* El bloque de memoria más grande se divide en fragmentos pequeños.
* Hará falta la compactación para obtener un bloque de memoria grande al final del espacio de memoria.

**Sistema de colegas**

El espacio entero disponible para la asignación se trata como un solo bloque de tamaño 2U. Si se hace una solicitud del tamaño tal que 2U−1 < s <= 2U, entonces el bloque entero se asigna:

* En otro caso, el bloque se divide en dos colegas de igual tamaño.
* Este proceso continúa mientras que el bloque más pequeño sea mayor o igual que s.

**3. Paginación**

**Definición:**

* La memoria principal se encuentra dividida en pequeñas partes iguales de tamaño fijo. Cada una de esas partes se denomina marco.
* La imagen de cada proceso se divide (desde el punto de vista lógico) en pequeñas partes iguales de tamaño fijo e igual al de los marcos. Cada una de esas partes se denomina página.
* Los marcos de memoria principal albergan (’envuelven’) las páginas.
* El sistema operativo mantiene una tabla de páginas para cada proceso:
  + Cada entrada en esa tabla se corresponde con una página del proceso e indica el número del marco donde se encuentra dicha página en memoria principal.
* Una dirección lógica consta de un número de página y de un desplazamiento dentro de la página.
* Una dirección física consta de un número de marco y de un desplazamiento dentro de ese marco.

**4. Segmentación**

**Definición:**

* La imagen de un proceso está formada por múltiples segmentos.
* No es necesario que todos los segmentos de todos los programas tengan la misma longitud. Cada segmento se almacena entero de forma contigua en la memoria principal.
* Existe una longitud máxima de segmento.
* En un sistema con gestión de memoria basada en segmentación, una dirección lógica consta de dos partes: un número de segmento y un desplazamiento.
* Se crea una tabla de segmentos para cada proceso.
  + La tabla tiene una entrada por cada segmento del proceso.
  + Cada entrada de la tabla indica, para el segmento correspondiente, la dirección base donde comienza en memoria principal y su tamaño.

Como consecuencia del empleo de segmentos de distinto tamaño, la segmentación resulta similar a la participación dinámica.

* Un proceso puede ocupar más de un segmento.
* En memoria no tienen por qué estar contiguos los distintos segmentos del proceso.
* No se produce fragmentación interna.
* Sí se puede producir fragmentación externa, aunque menor que con las particiones dinámicas.

**Proceso de carga:**

Se divide el programa en segmentos

Se cargan todos sus segmentos en bloques libres de memoria

Se actualiza la lista de bloques libres.

**5. Intercambio**

**Definición:**

* En determinadas circunstancias, un proceso que está en memoria puede ser almacenado temporalmente (intercambiado) en un almacenamiento auxiliar (normalmente en disco magnético), de modo que el planificador pueda asignar su espacio de CPU a otro proceso situado en el almacenamiento auxiliar.
* Normalmente, un proceso intercambiado a disco se intercambia de regreso al mismo espacio de memoria que ocupó anteriormente. Esto es obligatorio si la vinculación de direcciones se realizó en tiempo de compilación o carga, no así si la vinculación de direcciones se realiza en tiempo de ejecución.
* El sistema de almacenamiento debe tener espacio suficiente para alojar las imágenes de memoria de todos los procesos y proporcionar un acceso rápido a las mismas.
* Los procesos listos que están en disco pueden estar intercalados con los procesos listos en memoria en la lista de procesos listos del planificador o estar en una cola de segundo nivel para ser intercambiados a disco en grupos cada cierto tiempo.
* No deben realizarse intercambios de procesos que tengan pendientes operaciones de E/S que requieran el uso de buffers en el espacio de memoria propio (OK si utilizan buffers del SO).
* La duración del cambio de contexto desde disco depende principalmente del tamaño de la zona de memoria a intercambiar (la zona de intercambio es un área independiente del sistema de archivos que permite un acceso rápido a la información). En cualquier caso, es superior al cambio de contexto entre procesos que están en memoria.
* Los algoritmos de asignación de espacio en disco para intercambios son los mismos que los explicados para la asignación de memoria.

**Memoria Virtual**

**1. Introducción**

**Conceptos:**

* Concepto de Memoria Virtual: Método para conseguir que la suma de los espacios de pila, datos y texto (código) de un programa pueda ser mayor que el tamaño físico de la memoria disponible para él.
* A cada proceso se le asigna un área de direcciones, pudiendo ser no contiguo.
* El SO mantiene en memoria solamente las partes del programa que se están utilizando y mantiene en disco (intercambiadas) el resto.
  + Conjunto residente: parte del proceso que está en la memoria principal (memoria real).
  + Un usuario percibe en potencia una memoria mucho mayor disponible para sus procesos, que combina la memoria principal con la parte de la disco destinada a este fin. La memoria en total que parece que tiene disponible se llama (Memoria Virtual)
* Sirve para sistemas de mono y multiprogramación.
* Se necesita disponer del hardware que dé soporte a la gestión de la memoria virtual.

**Ventajas:**

* Permite optimizar el uso de la memoria principal:
  + Mantiene más procesos en memoria principal, albergando solo una parte (suficiente) de cada uno de ellos.
    - Permite la multiprogramación de forma efectiva
* Mantiene en disco partes del proceso poco usadas (rutinas de atención a errores poco frecuentes, funciones de uso esporádico, datos menos usados, ...)
* Permite que un proceso sea más grande que toda la memoria principal.
* El encargado de gestionar la memoria virtual es el sistema operativo.

**Problemas**:

* Fallos de direccionamiento: se intenta acceder a una posición/dirección (de una instrucción/dato) que no está en memoria principal.
  + Se genera una interrupción, indicando fallo de acceso a memoria.
  + El proceso pasa a estado bloqueado, y el SO a ejecución.
  + El SO emite una solicitud de E/S al disco
  + El SO expide otro proceso para que se ejecute (planificador del S.O.)
  + La operación de E/S se realiza, actualizando la memoria principal. Llega la interrupción indicando que la operación de E/S se ha realizado.
  + Se devuelve el control al S.O., que pasa el proceso a estado listo. El planificador decidirá cuando debe pasar a ejecución.
* En algunos casos, se puede dar hiperpaginación (hyperthrashing)
  + El SO más tiempo intercambiando bloques que ejecutando procesos.

**2. Paginación**

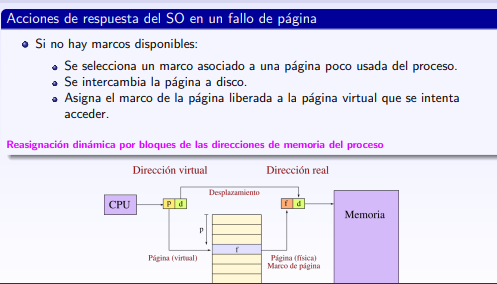
**Conceptos**

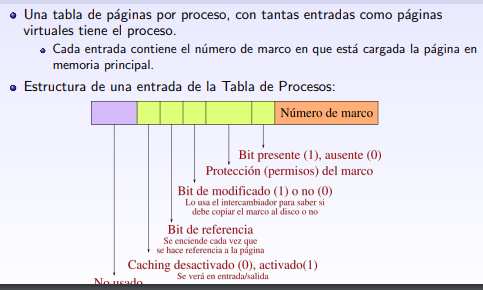
* La memoria física se divide en bloques de tamaño fijo: marcos.
* La memoria virtual se divide en bloques del mismo tamaño: páginas (los procesos se dividen en páginas).
* Al ejecutar un proceso (también para ponerlo listo), se cargan sus páginas en los marcos disponibles. La vinculación de direcciones requiere soporte por hardware (Manejador de Memoria).
* En un sistema con paginación no se produce fragmentación externa y sí fragmentación interna.
* Tabla de páginas:
  + Normalmente una por proceso (más adelante veremos que en procesos grandes se complica)
  + Requiere soporte por hardware: manejador de memoria (MMU)

Un intento de acceso a una página virtual que no esté asociada a un marco produce un señalamiento al SO (trap), llamado fallo de página.

**Acciones de respuesta del SO en un fallo de página**

* Si no hay marcos disponibles:
  + Se selecciona un marco asociado a una página poco usada del proceso.
  + Se intercambia la página a disco.
  + Asigna el marco de la página liberada a la página virtual que se intenta acceder.



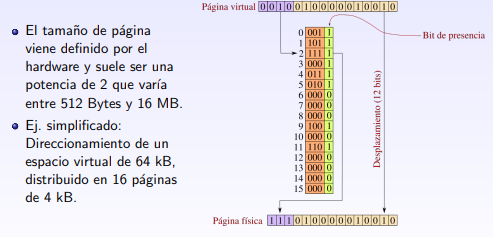


* Contiene (para cada proceso) el número de marco que corresponde a cada página virtual del proceso.
* Normalmente una tabla de páginas por proceso
* Las tablas de páginas de los procesos tienen una longitud variable (dependen del tamaño de cada proceso - 1 entrada por cada página).
* La tabla está cargada en memoria principal y, si es muy grande, puede estar sujeta a su vez a paginación ⇒ Paginación multinivel

**Al ejecutar un proceso**

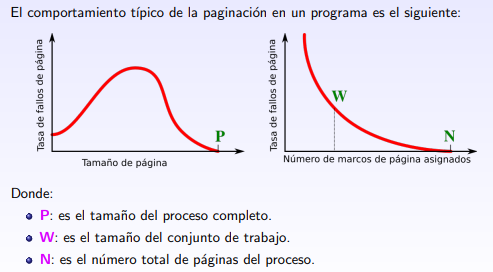
La dirección de comienzo de la tabla de páginas de ese proceso se mantiene en un registro

Se cargan las páginas necesarias en los marcos disponibles



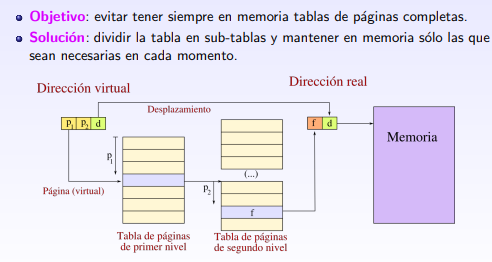
**Aspectos de diseño**

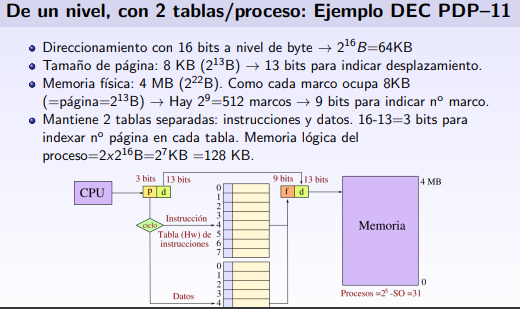
* El tiempo de asociación (página-marco) debe ser reducido
  + Soluciones basadas completamente en hardware (utilizando registros) son las más rápidas, pero esto solo es válido si las tablas de páginas son pequeñas.
* Relativo al tamaño de las páginas:
  + Cuanto menor sea el tamaño de página, menor será la cantidad de fragmentación interna (En promedio la mitad del tamaño de la página).
  + Cuanto menor sea la página, mayor será el número de páginas que se necesitan por proceso.
  + Un número mayor de páginas por proceso significa que las tablas de páginas serán mayores.
  + Esto puede significar que una gran parte de las tablas de páginas de los procesos activos deben estar en la memoria virtual.
* Considerando páginas pequeñas**:**
  + Si el tamaño de página es muy pequeño, estarán disponibles en la memoria principal un gran número de páginas para cada proceso.
  + Después de un tiempo, todas las páginas de la memoria contendrán parte de las referencias más recientes del proceso. La tasa de fallos de página será menor.
* Considerando páginas grandes**:**
  + La memoria secundaria está diseñada para transferir eficazmente los bloques de datos de mayor tamaño, de manera que es propicia para tamaños de página mayores.
  + Cuando se incrementa el tamaño de la página, cada página individual contendrá posiciones cada vez más distantes de cualquier referencia reciente. La tasa de fallos será mayor.

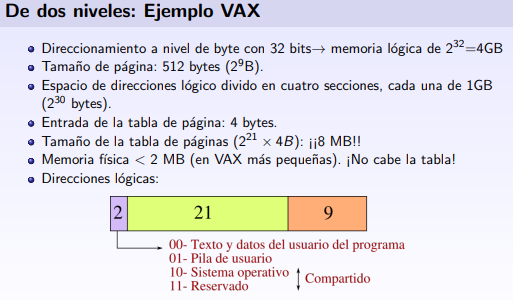


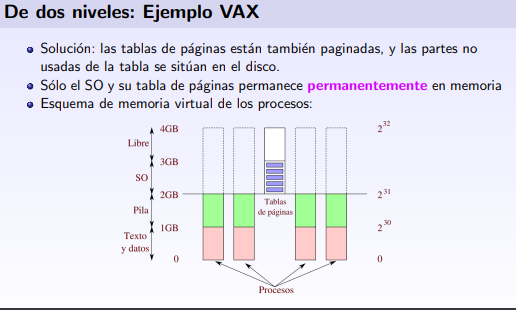
**Soluciones al tamaño de tablas grandes**

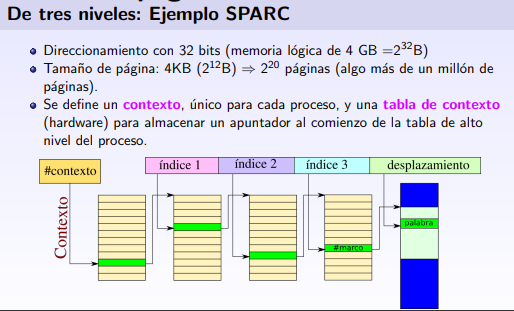
* Paginación multinivel
  + Una parte de las tablas de páginas deben estar en la memoria virtual, sujetas a paginación también.
* Utilizar tablas de páginas invertidas en lugar de tablas de página

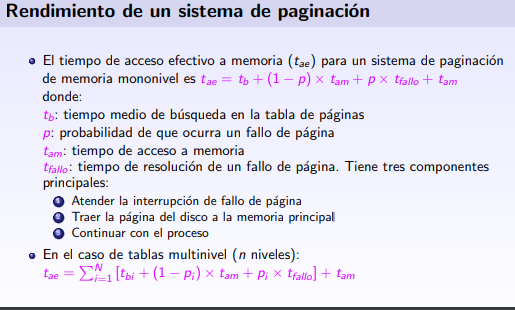








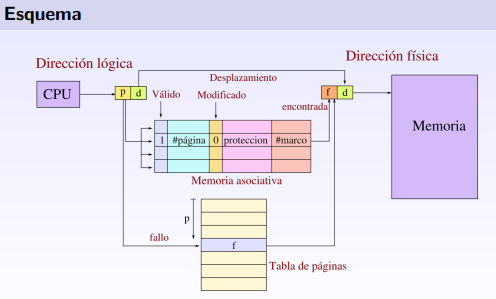


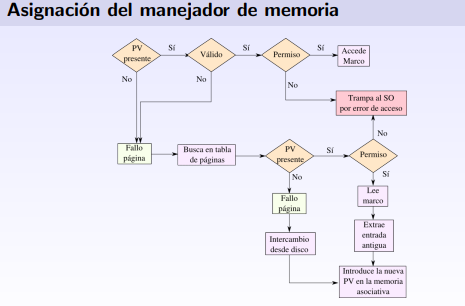


**Memoria Asociativa**

**Conceptos:**

* Translation Lookaside Buffer (TLB o Buffer de Traducción Adelantada): solución para acelerar la traducción de direcciones y, por tanto, el acceso a los marcos, cuando las tablas de procesos son muy grandes (búsqueda lenta) y/o están organizadas en niveles (requiere múltiples accesos a memoria).
* Observación: los procesos acceden normalmente a un número pequeño de páginas (y esporádicamente al resto).
* Mejora/solución: dotar a los ordenadores con Hw (memoria asociativa, caché de memoria administrada por la MMU) para asociar algunos no p´aginas de uso frecuente con los no marcos en que están en MP sin necesidad de acceder a la tabla de páginas. Tamaño de la memoria asociativa usual: de 8 a 32 entradas.
* Proporción de encuentros: Proporción de accesos a la Memoria Asociativa que son exitosos (la Página Virtual buscada se encuentra en la Memoria).

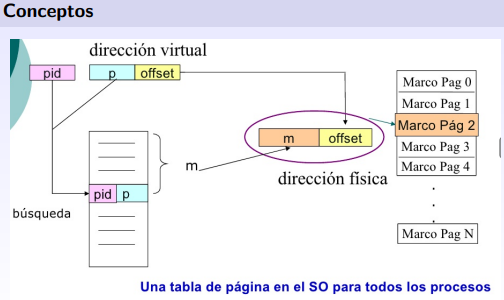




**Tablas de páginas invertidas**

**Conceptos:**

* Sistemas con direccionamiento por 64 bits a nivel de Byte (16 EB= 264B ), el número de páginas de tamaño (p.ej) 4 KB es 252 ⇒ Impensable mantener tablas de páginas de esa longitud, mayor que la memoria física de la mayoría de los sistemas.
* En sistemas donde el número de marcos físicos es sustancialmente menor, permite organizar la tabla de entradas alrededor de la memoria física en lugar de la memoria virtual (IBM S/38, HP Spectrum).
* La tabla de páginas invertida tiene tantas entradas como marcos, y cada entrada contiene la dirección virtual de la página.
* Se utiliza siempre con una memoria asociativa.
* Si una dirección virtual no se encuentra en la tabla invertida, se hace una búsqueda en una tabla convencional, que puede estar en memoria o en disco.
* Si se encuentra, se obtiene el número de marco.



**3. Políticas del gestor de memoria**

**Características**

Encaminadas a minimizar porcentaje de fallos de páginas para maximizar el rendimiento ¿De qué depende el rendimiento?

* Del tamaño de memoria principal
* De la velocidad relativa de memoria principal y secundaria
* Del tamaño y número de procesos que compiten por los recursos
* Del comportamiento de programas individuales:
  + Aplicación
  + Lenguaje de programación y compilador
  + Estilo de programador
  + Comportamiento dinámico del usuario en programas interactivos.

**Tipos de políticas**

**No hay una política definitivamente mejor que las otras**

Políticas de Lectura ¿Cuando cargo una página en memoria principal?

Políticas de Ubicación ¿Donde coloco la nueva página?

Políticas de Reemplazo: Si la memoria principal está llena ¿qué página/s reemplazo?

Gestión del Conjunto Residente ¿Cuánta memoria principal debe asignarse a un proceso cuando se carga?

Políticas de Vaciado ¿Cuando escribo una página modificada en memoria secundaria? Control de Carga ¿Cuál es el grado de multiprogramación?

**Políticas de Lectura (fetch)**

¿Cuándo cargar una página en memoria principal?

* Paginación por Demanda: Se carga una página solo cuando se produce un fallo en esa página
  + Número elevado de fallos al inicio
  + Con el tiempo y si no hay cambio de localización:
    - Estabilización gracias al principio de cercanía.
    - El número de fallos disminuye hasta un nivel bajo.
* Paginación Previa: se carga la página que ha producido el fallo y las páginas cercanas a esta.
  + Aprovecha características de los discos

**Políticas de Ubicación**

¿Dónde va a residir una parte de un proceso en la memoria principal?

* Es una decisión importante en sistemas con segmentación
  + Mejor ajuste, primer ajuste, siguiente ajuste, etc
* Carece de importancia en sistemas con paginación (con y sin segmentación)

**Políticas de Reemplazo**

**Si la memoria está ocupada:**

¿Qué Páginas se eligen para ser reemplazadas? Gestión del Conjunto Residente

¿Número de marcos a asignar para cada proceso? ¿Reemplazar sólo páginas del propio proceso o de cualquier otro?

Algoritmos de Reemplazo

* Tras un fallo de página y con toda la memoria principal ocupada, se debe elegir que página situada en memoria se lleva al disco para dejar el marco libre.
* Objetivo: reemplazar la página con menor posibilidad de ser referenciada en un futuro cercano ⇒ Intentar predecir el comportamiento futuro en función del comportamiento pasado.
* Cuanto más elaborada es la política, mayor sobrecarga.

**Políticas de gestión del conjunto residente**

Tamaño del Conjunto Residente El S.O. decide cuánta memoria asignar a un proceso (no marcos = no páginas que puede cargar en memoria principal)

Factores positivos que influyen en la decisión:

Cuanta menos memoria necesite cada proceso, mayor cantidad de procesos en memoria ⇒ mayor probabilidad de procesos en estado listo ⇒ aumenta el grado de multiprogramación.

Factores negativos que influyen en la decisión**:**

* Si hay pocas páginas de un proceso en memoria, aumenta la probabilidad de fallos de página.
* Por encima de un determinado número, más memoria no tendrá un efecto notable (por el principio de cercanía: se tendrán cargadas las páginas que se necesitan durante algún tiempo).

**Asignación Fija: otorga a cada proceso un número fijo de páginas**

* Se decide la cantidad de memoria asignada al proceso en la carga inicial, según el tipo de proceso o directrices del programador o administrador.
* Cuando hay fallos de página, siempre se reemplaza una página del mismo proceso.

**Problemas**

* **Asignación por exceso:**
  + Se desperdicia espacio: el proceso no necesita todos los marcos que se le han asignado.
  + Posible procesador ocioso: hay pocos procesos en memoria principal.
* **Asignación por defecto**:
  + Alto porcentaje de fallos de páginas, aunque en el sistema haya marcos vacíos
  + Multiprogramación más lenta (intercambios páginas con disco).

Asignación Variable: el número de marcos de un proceso cambia durante su vida

* Cambia en función de la tasa de fallos de página. Se asignará dinámicamente el no de marcos atendiendo a las necesidades del proceso.
* La asignación variable es más potente pero costosa (el S.O. debe evaluar el comportamiento de los procesos dinámicamente).

Los algoritmos de reemplazo de páginas pueden ser de:

**Ámbito global:** Un proceso puede utilizar marcos que pertenecen a otro proceso (ej: en asignación con prioridades).

Sólo tiene sentido con asignación variable.

**Ámbito local:** Un proceso solo puede utilizar marcos que le han sido asignados a dicho proceso.

Tiene sentido en asignación fija y en asignación variable.

**Problemas:**

Desperdicia espacio si el proceso no necesita todos los marcos que se le han asignado.

Si el proceso necesita más marcos de los asignados, no podrá ejecutarse correctamente, aunque en el sistema haya muchos marcos vacíos.

**Asignación fija y reemplazo de ámbito local**

Al cargar un nuevo proceso, se le asigna un número de marcos fijo (depende de aplicación y solicitud del programa)

**Desventajas**

Si se asignan pocas páginas se produce un alto porcentaje de fallos de páginas.

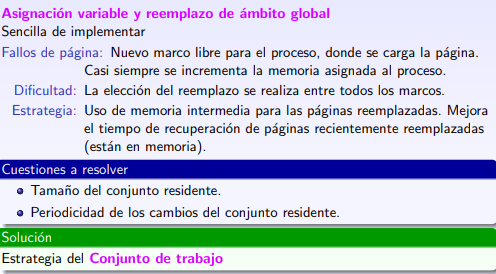
Si se asignan muchas páginas hay pocos procesos en memoria ⇒ procesador ocioso

**Asignación variable y reemplazo de ámbito local**

Al cargar un nuevo proceso, se le asigna un número de marcos (paginación previa o bajo demanda)

Cuando hay un fallo de página con reemplazo se selecciona una página del proceso que produjo el fallo de página

De vez en cuando, se evalúa la asignación de un proceso y se aumenta o disminuye para mejorar el rendimiento global.



**Conjunto de Trabajo**

Se utiliza para determinar el tamaño del conjunto residente y el momento de los cambios

El conjunto de trabajo de un proceso en un instante t es W (t, ∆t): conjunto de páginas a las que el proceso ha hecho referencia en las últimas ∆t unidades de tiempo

W (t, ∆t):

W crece con ∆t

W depende del instante t: puede ser 1 o llegar hasta el número total de páginas del proceso Su tamaño puede variar durante la ejecución

Crece al iniciarse un proceso

Estabilidad (principio de cercanía)

Las oscilaciones indican el desplazamiento del programa a otra ubicación

* Si el número de marcos disponibles es inferior al tamaño del conjunto de trabajo, se producirán frecuentes fallos de página
* Un proceso hiperpaginado pasa más tiempo intercambiando páginas que ejecutándose, y puede “robar” paginas de otros procesos, provocando, a su vez, su hiperpaginación (hypertrashing o trasiego)
* Consecuencia del hypertrashing: reducción drástica del uso de CPU ⇒ Aumenta el número de procesos en estado bloqueado.
* La hiperpaginación se limita con asignación local, y si se asigna a cada proceso un número de marcos suficiente.

**Problemas:** ¿Cómo calcular el número de marcos que un proceso necesita?

¿Cómo calcular el tamaño del conjunto de trabajo para un proceso?

**Modelo del conjunto de trabajo**

**Finalidad:** reducir la tasa de fallos de página

**Principio de localidad/cercanía:** las páginas que se utilizan en cada momento se pueden agrupar en grupos que varían a lo largo de la ejecución del proceso (P.e., una llamada a una función provoca un cambio de localidad).

**Estrategia**

**Supervisar** el conjunto de trabajo de cada proceso

**Eliminar periódicamente** del conjunto residente las páginas que no pertenezcan a su conjunto de trabajo

Un proceso se puede ejecutar sólo si su conjunto de trabajo está en memoria principal (en conjunto residente)

**Problemas**

El pasado no siempre predice el futuro

Es impracticable una medida real del conjunto de trabajo de cada proceso

El valor óptimo de ∆t a considerar es desconocido y puede variar

**Estrategia alternativa**

**Algoritmo de frecuencia de fallos de página**

* Cada página tiene un bit de uso que se pone a 1 cuando accede a la página
* Al producirse fallo de página el S.O. Mira el tiempo transcurrido desde el último fallo de página para el proceso
  + Si el tiempo es menor que F (umbral), se añade una página al conjunto residente.
  + En caso contrario:
    - Se descartan las páginas con el bit de uso a 0, reduciéndose el conjunto residente.
    - Se restaura a 0 el valor del bit de uso en las páginas restantes

**Estrategia alternativa (Cont.)**

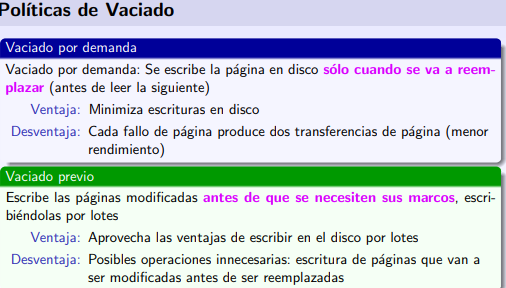
**Problema:** periodos de transición entre dos regiones de referencia El conjunto residente crece antes de que las páginas de la antigua región de referencia hayan sido expulsadas y provoca: Aumento del grado de multiprogramación, sobrecargando la CPU. Suspensión de procesos innecesaria

**Otra estrategia alternativa**

**Política de conjunto de trabajo con muestreos en intervalos variables:**

Evalúa el conjunto de trabajo mediante muestras temporales:

* Al comienzo de un intervalo de muestreo, se restauran los bits de uso de todas las páginas residentes del proceso
* Al final, solo las páginas que han sido referenciadas se mantendrán en el conjunto residente para el próximo intervalo de tiempo



**Políticas de Vaciado**

**Alternativa:** incorporar almacenamiento intermedio

Mantiene temporalmente las páginas recién reemplazadas en una zona de la memoria reservada para ello (simil: contenedores de basura)

**Dos tipos de contenedores:**

* Lista de páginas modificadas
* Lista de páginas no modificadas

**Lista de modificadas son enviadas a disco** periódicamente por lotes y despu´es se realiza un intercambio de los contenedores:

* Conjunto de páginas modificadas se convierte en el contenedor de no modificadas. (Veremos porque en el tema de E/S).
* Conjunto de páginas no modificadas pasa a ser el de modificadas.

**Control de carga**

**Grado de multiprogramación**

* Si hay pocos procesos es probable que todos estén en estado bloqueado
* Si hay muchos, el tamaño medio del conjunto de trabajo no es el adecuado: aumenta la frecuencia de fallos de página (posible hiperpaginación)

Grado de multiprogramación se debe aumentar cuando el número de fallos de página sea pequeño y disminuir cuando sea grande

**Control de carga**

**Criterios de suspensión de procesos**

Baja prioridad: Decisión de política de planificación

Muchos fallos de página: Conjunto de trabajo inadecuado

Último activado: Conjunto de trabajo residente no formado

Conjunto residente más pequeño: Penaliza procesos con ubicaciones pequeñas

Proceso mayor: Libera una cantidad de marcos grande

Mayor ventana de ejecución restante:

Planificación: primero el proceso más corto

La elección depende de los objetivos y el tipo de programa

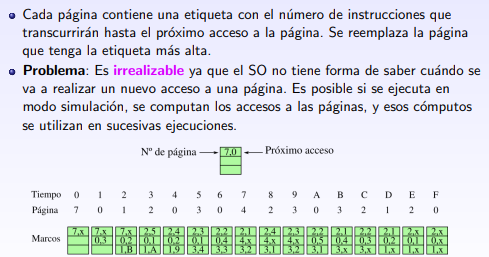
**4. Algoritmos de reemplazo**

**Conceptos**

Tras un fallo de página, el SO debe elegir qué marco de página de la memoria debe ser intercambiado a disco para hacer sitio a la nueva página que se está solicitando.

Criterio general: sustituir marcos de páginas poco usados por el proceso

Óptimo:



**LRU (Least Recently Used)**

* Implementa una aproximación del algoritmo óptimo, basado en mirar al pasado y a partir de él estimar cuál podría ser el uso de la página.
* Selecciona el marco en el que está la página que lleva más tiempo sin acceso (la que hace más tiempo que no se referencia).
* Hay varias implementaciones posibles

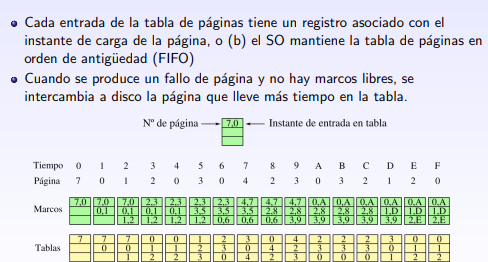
**Posibles implementaciones**

**La tabla se implementa como pila**

* Pila de números de página que conserva en la salida la página más recientemente usada y en la base la menos recientemente usada
* La actualización de la tabla es muy lenta, aun usando HW especial.

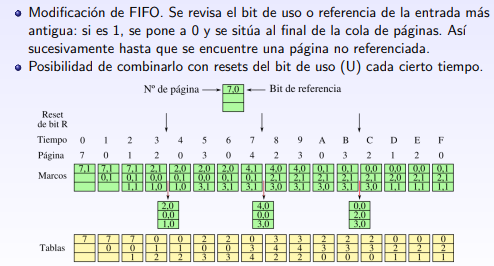
**Uso de contadores**

* Las entradas de la tabla de páginas tienen un campo de ‘tiempo de uso’ en el que el Manejador de Memoria escribe el tiempo de cada referencia.
* Cada acceso a memoria requiere una búsqueda en la Tabla de Páginas y una escritura en memoria por cada acceso a memoria.
* Requiere HW especial.



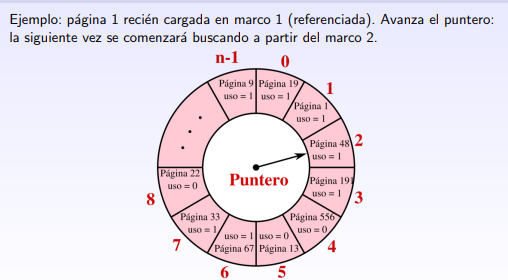
Fifo:

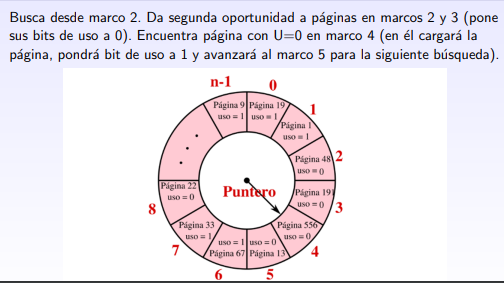
Segunda oportunidad:



**Del reloj**

* Las páginas se mantienen en una cola circular, con un puntero a la página más antigua (cargada hace más tiempo en memoria). Si hay un fallo de página y el bit de uso del marco apuntado es U=0, la página que estaba en ese marco se reemplaza y se avanza el puntero. Si U=1, se pone U=0 y se avanza hasta encontrar una página con U=0 (página que se reemplaza, y el puntero avanza al siguiente marco).
* Es solo una implementación alternativa del algoritmo de segunda oportunidad, más eficiente: es el puntero el que se desplaza, no las entradas de la tabla.





**NRU (Not Recently Used)**

Utiliza los bits de Uso (U) y Modificado (M) de las entradas de la tabla de páginas. Se definen cuatro clases de páginas.

Clase 0: Páginas no referenciadas recientemente ni modificadas (U=0,M=0).

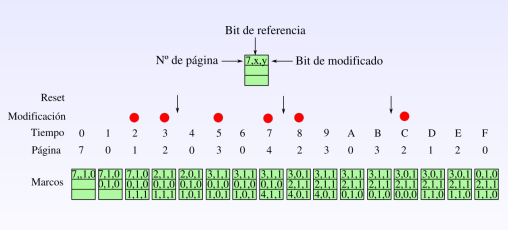
Clase 1: Páginas no referenciadas recientemente, modificadas (U=0,M=1).

Clase 2: Páginas referenciadas recientemente, no modificadas (U=1,M=0).

Clase 3: Páginas referenciadas recientemente, modificadas (U=1,M=1).

**Algoritmo de selección de página a reemplazar:**

* El algoritmo NRU elimina una página de manera aleatoria de la primera clase con el número más bajo.
* De manera periódica (o bien cada interrupción del reloj) se pone a cero el bit de Uso (o Referencia), para distinguir las páginas que no tienen referencias recientes de las que sí.
* Una hipótesis impl´ıcita de este algoritmo es que es mejor eliminar una página modificada sin referencias en al menos un intervalo temporal que una página sin modificar de uso frecuente.



**Algoritmo del Reloj Mejorado**

Utiliza los bits de Uso (U) y Modificado (M) de las entradas de la tabla de páginas. Se definen cuatro clases de páginas.

Clase 0: Paginas no referenciadas recientemente ni modificadas (U=0,M=0).

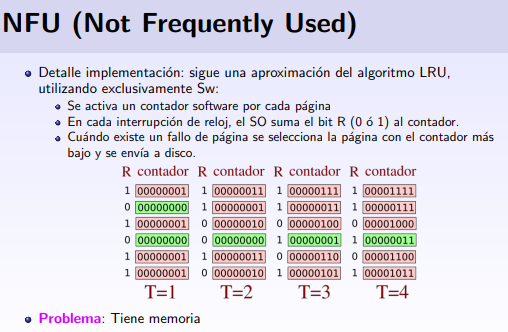
Clase 1: Páginas no referenciadas recientemente, modificadas (U=0,M=1).

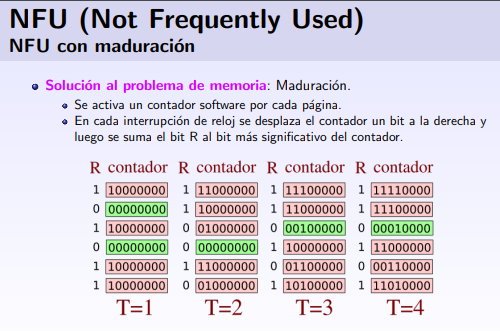
Clase 2: Páginas referenciadas recientemente, no modificadas (U=1,M=0).

Clase 3: Páginas referenciadas recientemente, modificadas (U=1,M=1).

**Algoritmo de selección de página a reemplazar**:

* Buscar entre las de clase 0 (U=0, M=0) Si no se encuentra ninguna (el puntero ha dado la vuelta entera), buscar de clase 1 (U=0, M=1) y durante la búsqueda cambiar los U=1 que vaya encontrando por U=0.
* Si no se encuentra ninguna, buscar de nuevo de clase 0.
* Si no encuentra ninguna, buscar de nuevo de clase 1 (alguna encontrará, pues todos los bits de uso se han puesto a 0 en la 2a vuelta).





**NFU con maduración**

**Ventaja de reloj modificado (NFU) sobre reloj**

Entre las páginas menos usadas, las no modificadas se reemplazan antes que las modificadas, lo que implica un ahorro de tiempo de escritura en disco (aunque una página se reemplace, no es necesario volver a escribirla en disco, pues no ha sido modificada).

**En ambos, reloj y reloj modificado o NRU:** Al tener en cuenta el uso de las páginas, aunque haya que escribir en disco una página que ha sido reemplazada, se supone que dicha página tardará en ser usada de nuevo (principio de cercanía)

**Almacenamiento Intermedio de páginas**

* Estrategia para mejorar rendimiento de paginación.
* Creación de dos listas:
  + Lista de páginas libres = paginas sin modificar reemplazadas”(marcos libres)
  + Lista de páginas modificadas reemplazadas”
* La página a reemplazar en realidad no se quita de la memoria principal:
  + Se pone su bit de presencia=0 en su entrada en la tabla de páginas
  + Se pone en la lista de páginas libres o modificadas

**Ventajas del almacenamiento intermedio de páginas**

* Reduce el coste de cargar páginas que han sido reemplazadas hace poco tiempo
* Lista de páginas libres = lista de marcos disponibles para cargar páginas (se mantiene un mínimo)
* Las páginas modificadas son reescritas en disco por bloques en vez de una a una, reduciendo el número de E/S y el tiempo de acceso al disco.

**Bloqueo de marcos**

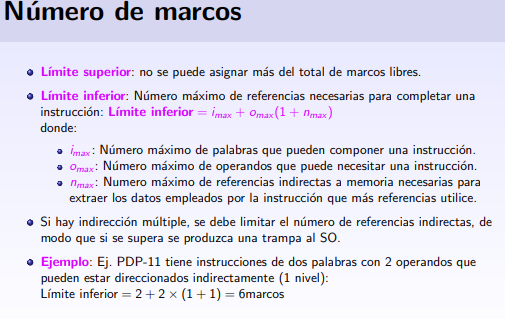
Algunos marcos de la memoria principal pueden bloquearse imponiendo una restricción a la política de reemplazo

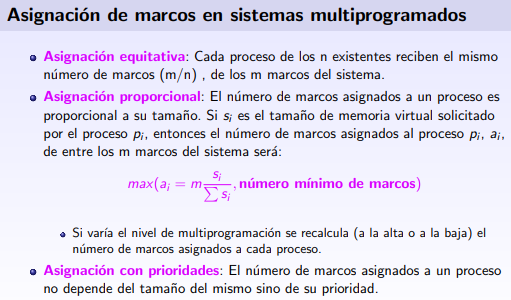
* La página situada en un marco bloqueado no puede ser reemplazada
* Se utiliza para el núcleo del S.O. y otras áreas críticas
* Se asocia un bit de bloqueo a cada marco (en tabla de marcos o tabla de páginas)

**5. Criterios de diseño de paginación**

**Asignación de marcos en sistemas monoprogramados**

* Se asignan páginas al SO y las restantes páginas libres se van asignando tras los correspondientes fallos de página a las páginas del proceso
* Una vez llenas todas las páginas, el manejador utiliza uno de los algoritmos de asignación por demanda para intercambiar páginas de la memoria y habilitar huecos para las nuevas páginas.
* Variación: el SO reserva parte de su espacio (libre) para apoyar la paginación. El espacio reservado sirve para almacenar temporalmente la página entrante mientras se selecciona la página que se va a intercambiar.





**Tamaño de pagina**

Razones para escoger un tamaño pequeño:

* Reduce la fragmentación interna.
* Favorece la localidad (lo que se carga en memoria se ajusta a lo que se necesita).

Razones para escoger un tamaño grande:

* Reduce el tamaño de la tabla de páginas (solo el 1 % del tiempo de
* Lectura/Escrituras de/a disco, se debe a la transferencia, el 99 % son los tiempos de latencia y posicionamiento).
* Reduce el número de fallos de página.

**Área de almacenamiento en disco**

Lo más simple es asignar un área dedicada (inicialmente vacía), separada del sistema de directorios. Opciones:

1 Cada vez que se crea un proceso se reserva una zona del área de intercambio igual al tamaño de imagen del proceso. A cada proceso se le asigna la dirección en disco de su área de intercambio, que se almacena en la Tabla de Proceso.La lectura se realiza sumando el número de página virtual a la dirección de comienzo del área asignada al proceso.

2 Si los datos y/o la pila pueden crecer, es mejor reservar zonas zonas independientes. A cada zona se le asignan varios bloques.

3 No se asigna nada inicialmente. A cada página se le asigna su espacio en disco cuando se va a intercambiar, y el espacio se libera cuando la página vuelve a memoria.

**Problema**: se debe llevar contabilidad en memoria (página a página) de la localización de las páginas en disco.

**Administración de fallos de página**

1 El HW hace un señalamiento al núcleo, que guarda el contador de programa (y a veces el estado de instrucción interrumpida)

2 Ejecuta una rutina en ensamblador que resguarda los datos volátiles (registros) y llama al SO como un procedimiento

3 El SO detecta el fallo de página e intenta determinar la pagina virtual requerida (normalmente contenida en un registro, o a partir del contador de programa almacenado) 4 El SO verifica que la pagina es v´alida y los permisos. En caso de que no sea válida o no se disponga de los permisos adecuados el SO envía una señal al proceso o lo elimina. En caso contrario el SO intenta encontrar un marco libre y si no lo hay ejecuta el algoritmo de reemplazo de páginas.

5 Si la página seleccionada ha sido modificada se escribe en disco. Durante ese tiempo el planificador da paso a otro proceso

El SO busca en disco la página solicitada y planifica una operación de disco para recuperarla. Durante el tiempo de carga, el proceso solicitante sigue bloqueado, y el planificador puede dar paso a otro proceso de usuario. El marco se pone “ocupado”.

7 Al terminarse la carga de la página el SO actualiza las tablas de páginas del proceso(s) involucrados.

8 La instrucción que cometió el fallo se reinicia (o vuelve al estado en que quedó al ser interrumpida). El Contador de Programa se modifica para que apunte a esa instrucción.

9 El proceso que provocó el fallo se planifica de nuevo y el SO regresa a la rutina que lo llamó.

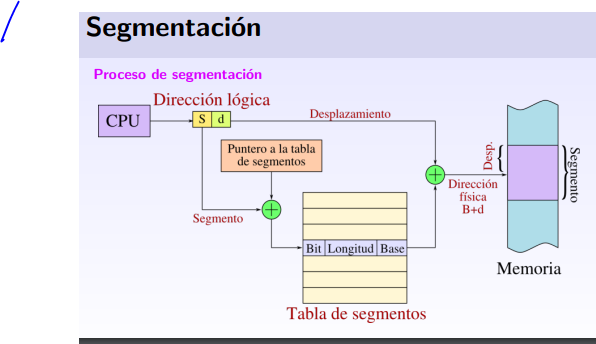
10 La rutina restaura los registros y demás información volátil y regresa al espacio de usuario para continuar la ejecución.

**Hiperpaginación (Trashing)**

* Definimos conjunto de trabajo como el número de páginas activas que un proceso tiene en un momento dado. Es el número suficiente, mayor al mínimo. Si el número de marcos disponibles es inferior al tamaño del conjunto de trabajo, se producirán frecuentes fallos de página (hiperpaginación).
* Un proceso hiperpaginado pasa más tiempo intercambiando páginas que ejecutándose, y puedo ‘robar’ páginas de otros procesos, provocando su hiperpaginación.
* **Consecuencia**: reducción drástica del uso de CPU. El planificador de procesos responde incrementando el nivel de multiprogramación. Este proceso se realimenta positivamente hasta que el sistema se desploma.
* La hiperpaginación se limita si se limita el número de marcos que el proceso puede utilizar (Asignación local), y si se asigna a cada proceso un número de marcos suficiente.
* **Problema:** Cómo calcular el número suficiente de marcos que un proceso necesita.
* Para calcular el tamaño del conjunto de trabajo, se utiliza el modelo de conjunto de trabajo, basado en el supuesto de localidad, que dice que en la ejecución de un proceso, las páginas que se utilizan en cada momento se pueden agrupar en grupos que var´ıan a lo largo de la ejecución del proceso. Ej: una llamada a una función provoca un cambio de localidad.

**6. Segmentación**

* Técnica para mantener espacios independientes de direcciones virtuales, llamados segmentos. Las distintas partes del proceso tienen espacios virtuales independientes entre sí.
* La segmentación permite:
  + 1 Asignar permisos distintos a las partes del proceso.
  + 2 Compartir ciertos datos o procedimientos entre procesos.
* Los segmentos se direccionan desde 0 hasta una dirección máxima, que puede variar.
* La segmentación pura puede producir problemas de fragmentación externa.



**Segmentación paginada**

**Segmentación + Paginación**

* Espacio de direcciones dividido en segmentos
* Cada segmento dividido en páginas de tamaño fijo = tamaño de marco en memoria
* Memoria principal dividida en marcos

Si el segmento es menor que una página, ocupa un marco completo.

**Direcciones**

* Desde el punto de vista del programador: número de segmento + desplazamiento
* Desde el punto de vista del sistema: n´umero de segmento + número de página dentro de ese segmento + desplazamiento

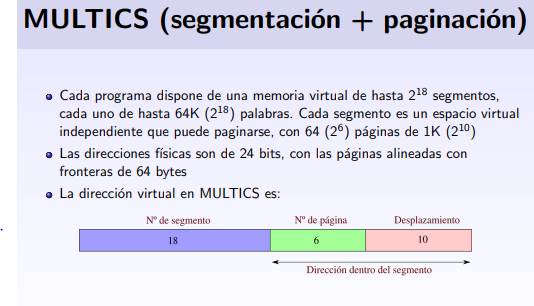
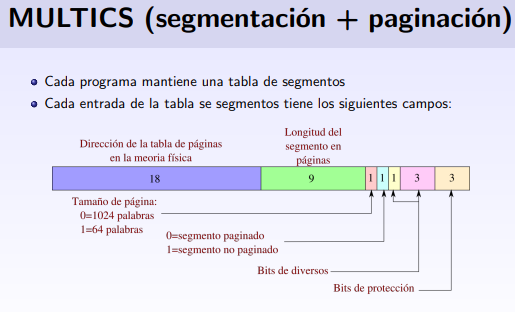


Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente

****

**7. Diseño del gestor de memoria**

**Dependencias**

1 Uso o no de memoria virtual (Si hay hardware disponible)

2 Uso de paginación, segmentación o ambas (Si hay hardware disponible)

3 Algoritmos empleados para los problemas de gestión de memoria

**Información**

Actualmente, casi todos los SS.OO. Ofrecen Memoria Virtual (DOS y primeros UNIX no) Al usar segmentación paginada, la mayoría

**Gestión de archivos**

**1. Introducción**

Un archivo es una colección de datos permanente asociada a un nombre. Los archivos proporcionan a los procesos entrada/salida permanente.

Proporcionar al usuario o a la aplicación acceso transparente a los archivos.

El programador no necesita desarrollar software de gestión de archivos y por lo tanto no necesita conocer los detalles del hardware o de la organización lógica empleada por el sistema.

**Objetivos particulares**

* Cumplir con las necesidades de gestión de datos y con los requerimientos del usuario (Ej. tamaño de los ficheros, permisos, etc ...).
* Garantizar que los datos de los archivos sean válidos.
* Optimizar el rendimiento (tiempo de acceso a ficheros, número de ficheros, etc ...).
* Ofrecer soporte de E/S para la variedad de tipos de dispositivos de almacenamiento (cinta, disco magnético, disco óptico etc.) a través de un conjunto estándar de rutinas de interfaz de E/S.
* Minimizar o eliminar la posibilidad de pérdida o destrucción de datos (copias de seguridad).
* Proporcionar soporte de E/S para múltiples usuarios.

**Cada usuario debe ser capaz de:**

* Crear, borrar y modificar sus archivos.
* Acceder (si tiene permiso) a los archivos de otros usuarios.
* Controlar qué tipos de accesos estarán permitidos a sus archivos.
* Reestructurar sus archivos de manera adecuada al problema.
* Mover datos entre los archivos.
* Guardar una copia de reserva y recuperar sus archivos en el caso de que hayan sufrido algún daño.
* Acceder a sus archivos mediante un nombre simbólico.

**Arquitectura**

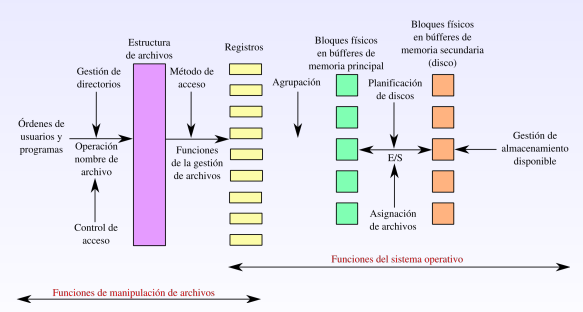
1 Solicitud de acceso a archivos, por el nombre de los mismos (pila, secuencial, secuencial indexado, indexado o por dispersión).

2 Solicitud de acceso a bloques lógicos que pertenecen a un archivo. Reconoce nombres y busca en directorios de archivos.

3 Responsable del comienzo y final de toda E/S. Mantiene las estructuras de control. Solicita acceso a bloques físicos que pertenecen a un archivo.

4 Instrucciones de alto nivel para acceder a bloques de disco. Identifica cada bloque por su dirección en disco: Unidad, cilindro, pista, sector. Ubica los bloques en memoria.

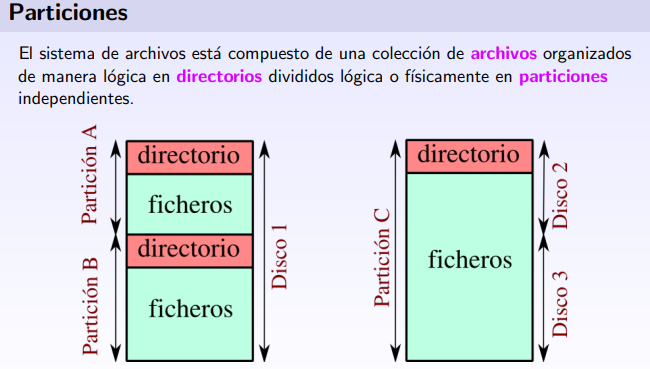
5 Instrucciones de bajo nivel (al hardware) para acceder a bloques de disco, comenzar y terminar operaciones de E/S.



**Funciones:**

* Identifica y ubica el archivo en cuestión.
* Utiliza un directorio que describe la ubicación de todos los archivos y sus atributos. Los sistemas compartidos aplican algún control de acceso a los usuarios.
* La E/S se lleva a cabo por bloques.
* Asigna los archivos a los bloques disponibles.
* Gestiona el espacio libre, de manera que se conozca qué bloques están disponibles.

**2. Sistemas de archivos**

**Organización**

**MBR (registro maestro de arranque):**

* Contiene tabla de particiones. Cada entrada contiene las direcciones inicial y final de la partición
* Una de las particiones de la tabla está marcada como activa
* Al encender un computador, la BIOS lee el sector de arranque y ejecuta el bloque de arranque de la partición activa.

**Bloque de arranque:**

* Carga el S.O. contenido en su partición y en el caso de ordenadores con múltiples sistemas operativos en ese bloque se encuentra el sistema de arranque de sistemas (LILO, grub, etc.)
* Por uniformidad, cada partición inicia con el bloque de arranque, aunque no contenga un S.O. arrancable

**Superbloque:**

* Contiene parámetros claves:
  + Tipo de sistemas de archivos (FAT, NTFS,...).
  + Número de bloques, tamaño de bloque. Información administrativa.
* Es el bloque siguiente al de arranque.
* Se transfiere del disco a la memoria al arrancar el sistema.

**Archivos**

**Tipos**

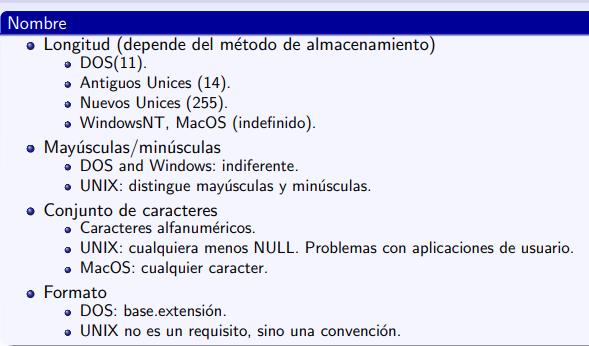
**normales**: contienen información de usuario

* Texto (ASCII, editables e imprimibles, líneas con salto de línea/retorno de carro)
* Binarios

**Directorios**: mantienen estructura del sistema de archivos

**Especiales**:

* Caracteres (para modelar dispositivos de E/S)
* Bloques (para modelar discos)
* De dispositivo



Tipo

* El tipo se define mediante datos asociados, externos al fichero (Metadata):
  + Ejemplo: UNIX (d,l,s,r).
* El tipo de fichero (o la aplicación que puede “entender” el fichero está codificado en la cabecera (primeros bytes) del fichero:
  + perl, ps, pdf ...
* El tipo está codificado en el nombre (extensión): DOS: forzado por el SO. Forzado por la aplicación: por ejemplo el compilador, etc.

Otros atributos:

* Localización.
* (actual, máximo).
* Propietario.
* Permisos.
* Día, hora (creación, accesos, cambios).

Miscelanea. UNIX: codificado en el nombre (por ejemplo ficheros ocultos). WindowsNT, MacOS: permiten definir y almacenar nuevos atributos.

Operaciones con archivos

CREAR(nombre).

ESCRIBIR(nombre, información) Usa un puntero.

LEER(nombre, almacenamiento) Usa un puntero.

REPOSICIONAMIENTO.

RENOMBRAR(antiguo nombre, nuevo nombre).

AÑADIR

TRUNCAR(nombre).

ELIMINAR(nombre).

ABRIR (cuenta de referencias).

CERRAR

**Métodos de acceso**

Reflejan distintas estructuras de archivos y formas diferentes de acceder y procesar los datos.

**Tipos**: LIFO (Pila).

* Secuencial: Lee o escribe los siguentes n bytes del fichero.
* Aleatorio: Lee o escribe el byte n-´ésimo.
* Indexado: Lee o escribe el registro que tenga una llave (key) determinada.

**Directorios**

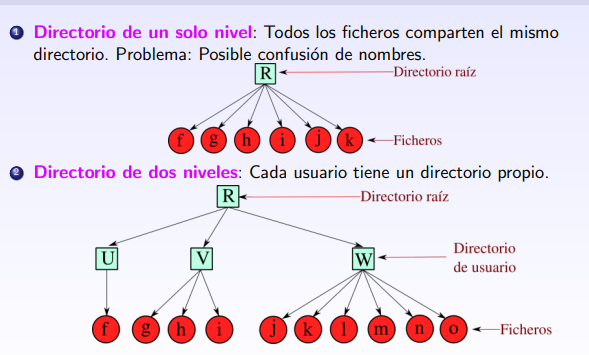
**Conceptos**

Contienen información sobre los archivos:

* Atributos.
* Ubicación.
* Propietario.

El directorio es propiamente un archivo, poseído por el sistema operativo.

Ofrece una traducción entre los nombres de archivo y los archivos propiamente dichos.



**Directorios con estructura de árbol:** Generalización de los directorios de dos niveles (MS-DOS, UNIX): ´

* Cada archivo tiene un nombre y una ruta de acceso absoluta, que es el camino (archivos ligados por separadores) desde el directorio raíz hasta el archivo.
* Separadores: MS-DOS: \ UNIX: / MULTICS: >
* La ruta de acceso relativo indica el camino hasta un archivo a partir del directorio de trabajo o directorio activo.
* Indentificadores especiales: Directorio de trabajo: “.” Directorio padre: “..”

**Estructura mononivel**

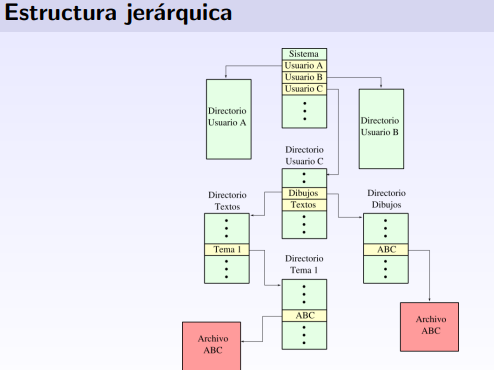
* Es una lista de entradas, una para cada archivo.
* Puede representarse con un simple archivo secuencial, con el nombre del archivo haciendo las veces de clave.
* No ofrece ayuda en la organización de los archivos.
* Obliga al usuario(s) a tener cuidado de no usar el mismo nombre para dos tipos diferentes de archivo.
* Búsquedas lentas.
* Complica el uso de comodines

**Estructura de dos niveles**

* Un directorio para cada usuario y un directorio maestro.
* El directorio maestro contiene una entrada para cada directorio de usuario:
  + Incluye una dirección e información de control de acceso.
* Cada directorio de usuario es una simple lista de los archivos del usuario.
* Todavía no ofrece a los usuarios ayuda alguna para estructurar sus conjuntos de archivos.

**Estructura jerárquica**

* Existe un directorio maestro que contiene un número determinado de directorios de usuario.
* Cada uno de estos directorios puede tener a su vez subdirectorios y archivos como entradas.
* Separadores: / , \ , : , >
* Cualquier archivo puede ser localizado siguiendo un camino desde el directorio raíz o maestro, descendiendo por varias ramas:
  + Este es el nombre de camino del archivo.
* Se pueden tener varios archivos con el mismo nombre de archivo mientras tengan nombres de camino únicos.
* El directorio actual es el directorio de trabajo.
* Las referencias a los archivos son relativas al directorio de trabajo.



**Implantación**

**Asignación contigua de bloques**

**Ventajas**

* Fácil implantación: la localización de un archivo se realiza a través de un único número (la dirección del primer bloque que compone el archivo).
* Alto rendimiento: Los accesos requieren una única búsqueda (movimiento inicial) y luego a lo máximo movimientos de un sector (del bloque final de un cilindro al bloque inicial del siguiente).

**Inconvenientes**

* La asignación dinámica de espacio se complica. Normalmente se usan los métodos del primer ajuste o del mejor ajuste.
* Se produce fragmentación externa. Se asigna espacio a un archivo sin saber su tamaño final. Se soluciona sobre asignando espacio (fragmentación interna) y/o reubicando los archivos cuando ocupen todo el hueco existente (perdida de tiempo).



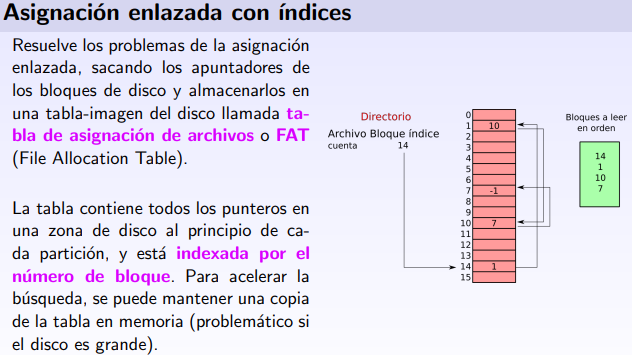
**Asignación enlazada con listas**

**Ventajas**

* Evita la fragmentación externa (todos los bloques libres pueden ser asignados a un archivo) e interna (no requiere que se defina el tamaño del archivo en el momento de su creación).

**Inconvenientes**

* El acceso directo (no secuencial) es muy lento: cada acceso a disco requiere recorrer toda la lista de punteros, lo que requiere un acceso (lectura y posible búsqueda) por puntero.
* Requiere espacio para almacenar los punteros, por lo que los archivos ocupan más espacio del que por su tamaño les corresponde. Solución: asignar espacio por clusters de bloques y no por bloques (a costa de fragmentación interna).
* Fiabilidad: Si se produce un fallo software o hardware, y se obtiene un puntero equivocado, todos los accesos posteriores serán equivocados.



**Asignación indexada**

**Ventajas**

* Facilita el acceso directo, al reducir la búsqueda del bloque (se realiza en un mismo bloque).

**Inconvenientes**

* Fragmentación interna: reserva un bloque para punteros.

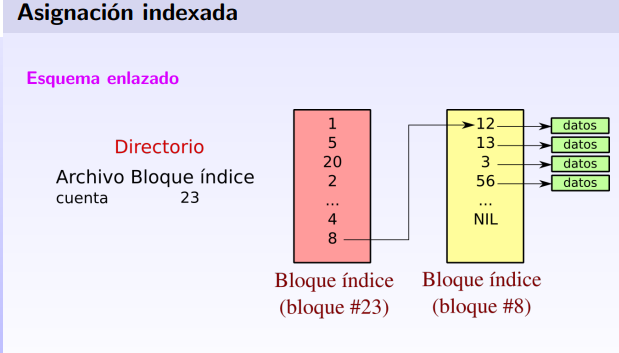
Selección del tamaño del bloque índice. Idealmente debe ser lo más pequeño posible (reduce fragmentación interna), pero debe ser suficientemente grande como para contener todos los punteros a los bloques que componen el archivo.

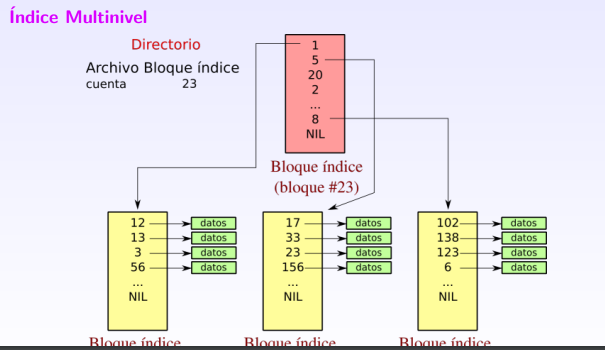
Soluciones:

1 Esquema enlazados.

2 Índice multinivel.

3 Esquema combinado.





**Esquema combinado:** i-nodos

**Características**

* i-nodos: Identificador interno de un fichero. A cada fichero se le asigna un número para poder localizarlo directamente.
* En el directorio a cada fichero se le asigna su i-nodo.
* Cada fichero tiene un nodo-i distinto salvo que se trate de un enlace rígido o simbólico (único fichero que puede ser visto por varios nombres). ls -i

Disco

* La tabla de i-nodos contiene los i-nodos de todos los posibles ficheros del sistema de ficheros.
* Al formatear el disco se le asigna un número de i-nodos máximo y se reserva espacio para ellos.

Puede haber i-nodos que no se correspondan con ningún fichero.

El estándar POSIX establece un modelo de sistema de archivos que se ajusta al empleado en los UNIX tradicionales. Un archivo ordinario tendrá las propiedades siguientes:

1 El identificador de dispositivo del dispositivo que alberga al sistema de archivos.

2 El número de i-nodo que identifica al archivo dentro del sistema de archivos.

3 La longitud del archivo en bytes.

4 El identificador de usuario del creador o un propietario del archivo con derechos diferenciados.

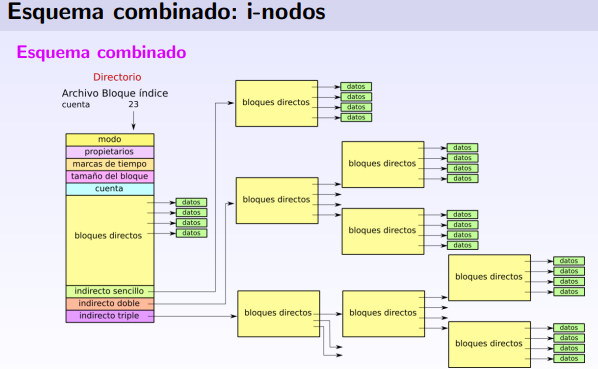
5 El identificador de grupo de un grupo de usuarios con derechos diferenciados.

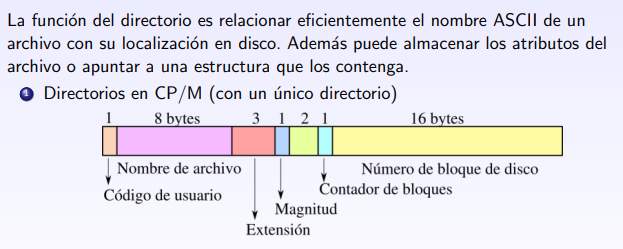
6 El modo de acceso: capacidad de leer, escribir, y ejecutar el archivo por parte del propietario, del grupo y de otros usuarios.

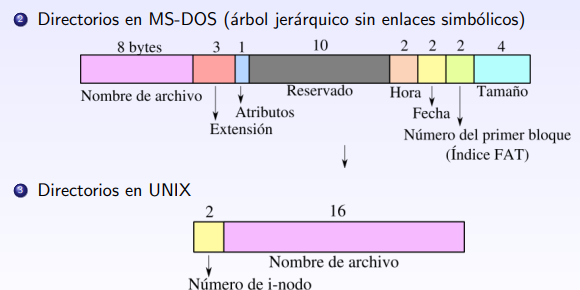
7 Las marcas de tiempo con las fechas de última modificación (mtime), acceso (atime) y de alteración del propio inodo (ctime).

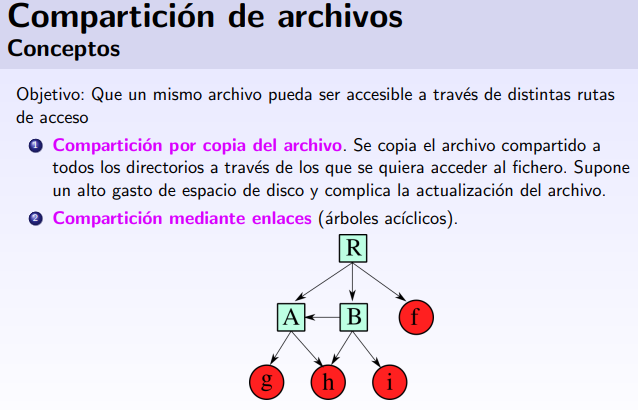
8 El número de enlaces, esto es, el número de nombres (entradas de directorio) asociados con este i-nodo.

En formato ext2 se disponen de 12 punteros directos, 1 indirecto simple, 1 doble y 1 triple









**Enlaces duros**

* Las entradas de los directorios que comparten el archivo apuntan a una estructura (i-nodo en UNIX) que contiene los atributos y apuntadores a bloques de disco del mismo.
* El creador del archivo es el que figura en el i-nodo. En él se lleva cuenta de las referencias (veces que el archivo está siendo compartido).
* Al borrar el archivo de uno de los directorios, sólo se decrementa el contador de referencias. Sólo se borra la entrada (i-nodo) cuando el contador es 0.

**Inconvenientes**

* No se pueden eliminar de una vez todas las referencias.
* Un fichero creado (y borrado) por un usuario puede serle contabilizado en el cómputo de su cuota de disco aunque sea otro usuario el que lo esté utilizando.

**Enlaces simbólicos**

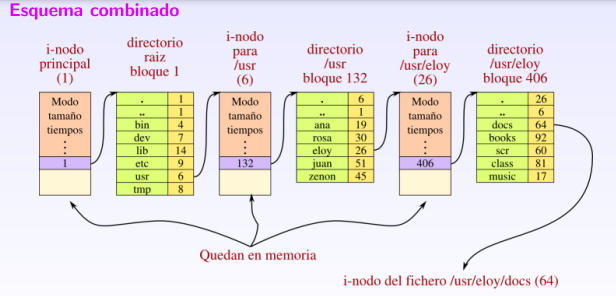
* Para compartir, se crea un archivo de un tipo especial (LINK), que contiene la ruta de acceso al archivo real (enlace simbólico).
* Sólo el propietario puede borrar un archivo, ya que solo él tiene acceso a su i-nodo. Si el propietario borra un archivo, lo destruye.

**Ventajas**

Permite enlaces entre máquinas.

**Inconvenientes**

Excesivo coste: cada referencia requiere una cadena de accesos a disco hasta poder acceder al i-nodo del archivo deseado, un nuevo i-nodo por enlace y un bloque en disco para almacenar la ruta de acceso.



**Tamaño del bloque**

* Bloques grandes tienden a desperdiciar mucho espacio de disco.
* Bloques pequeños para realentizar el acceso a los datos.
* Compromiso entre la optimización de memoria y la velocidad de acceso a los datos.
* Tamaños habituales: 512 b, 1 kB ´o 2 kB.

**Registro de bloques libres**

* Mediante lista ligada de bloques de disco (búsqueda sencilla).
* Mediante mapas de bits (ocupa menos espacio).

**Disk Quotas**

* El i-nodo de cada archivo abierto contiene un apuntador a una tabla de cuotas del usuario que abrió el fichero. La tabla contienen los límites flexibles y estrictos del número de bloques y archivos permitidos.

**Bloques defectuosos**

* Solución software (IDE). Se detectan los bloques defectuosos utilizando una función (format, chkdsk). El programa indica en la FAT que el bloque es defectuoso, y por lo tanto no debe ser usado.
* Solución hardware (SCSI). El fabricante proporciona una lista de sectores defectuosos tras realizar el formateo a bajo nivel. Esta lista se almacena en un bloque o sector, junto a los sustitutos de los bloques defectuosos.

**Respaldo (backup) y restauración**

* Cuando un bloque se corrompe durante el tiempo de funcionamiento, los datos normalmente se pierden, y deben ser restaurados desde un sistema de respaldo, que contenga copia en cinta, disco óptico, o en otro disco magn´etico del sistema de archivos.
* Backups incrementales.

**Consistencia del Sistema de Archivos**

* Las faltas de consistencia se producen cuando hay un fallo en sistema entre la modificación de un bloque (en memoria) y su copia en disco. Esto es especialmente grave si el bloque contiene i-nodos, directorios o listas de bloques libres.

**Solución UNIX para inconsistencia entre bloques:**

* El programa que analiza la consistencia crea una tabla de bloques ocupados y una tabla de bloques libres, que actualiza leyendo los i-nodos y la lista de bloques libres del sistema.
* Inconsistencias posibles:
  + Un bloque no está en ninguna de las dos tablas.
  + Un bloque aparece en las dos tablas
  + Un bloque aparece más de una vez en una de las tablas.

**Solución UNIX para la inconsistencia de archivos:** el programa recorre el árbol de directorios, creando una tabla de i-nodos que contiene el número referencias hechas a un archivo. El número de referencias se comparan con el número de enlaces que aparece en los propios i-nodos.

**Inconsistencias posibles:** El número de enlaces es mayor que el número de referencias desde los directorios. El número de enlaces es menor que el número de referencias.

**Otras comprobaciones:** búsqueda de inconsistencias en el número de i-nodo (si es mayor que el número de i-nodos en disco, o si el modo el i-nodo es extraño.