


De acuerdo con la forma de acceder a la información contenida en sus celdas, la memoria puede ser clasificada en:

- **Memoria convencional:** dada una dirección de memoria, el sistema devuelve el dato guardado en esa dirección.
- **Memoria asociativa:** dado un tag (datos de búsqueda), la CAM (memorias direccionables por contenido) busca en toda su memoria si ese tag está guardado ahí. Si el tag se encuentra, la CAM devuelve una lista de una o más direcciones de almacenamiento o el contenido completo de esas direcciones. Por lo tanto, un CAM es parecido a una tabla hash. Utilizada en memoria caché.

DIBUJO EN LA PÁGINA 5 HECHO POR EL PROFESOR

### Jerarquía de memoria:


El tiempo de acceso a memoria es el tiempo necesario para acceder a instrucciones o datos en memoria. La memoria está organizada en capas según el tiempo de acceso y la capacidad:

1. Registros del procesador
2. Memoria Caché
3. Memoria Principal
4. Unidades de disco duro 
5. Unidades de cinta y discos ópticos

El Sistema Operativo es un administrador de recursos, lo que implica que:

- El SO debe llevar la cuenta de la memoria asignada a cada proceso (a través de tablas de procesos) y de la cantidad de memoria libre.
- El SO debe tener una política para asignar memoria.
- El SO debe asignar memoria a los procesos cuando éstos la necesiten.
- El SO debe liberar memoria cuando ya no se necesita (cuando termina).
- El SO debe gestionar el sistema de Memoria Virtual

Los segmentos para un espacio de direcciones virtuales de un proceso son:

- Código (texto)
- Datos estáticos 
- Datos dinámicos (**Heap**)
- El **stack** (Argumentos y variables locales): devuelve direcciones.

DIBUJO PÁGINA 14 HECHO POR EL PROFESOR.

El área de Swap (intercambio) es una parte del disco usada como memoria auxiliar. Un proceso en ejecución debe estar en memoria. El Swapping puede aumentar el nivel de multiprogramación:

- Si un proceso que se encuentra en la zona de Swap es seleccionado, el proceso deberá cargarse en memoria.
- Para intercambiar procesos que están esperando a que se complete la E/S, ~~el SO debe transferir la E/S a los búferes del espacio Kernel y luego al dispositivo de E/S. Esto ocasiona gastos. Por ello,~~ los SO modernos con memoria virtual intercambian las páginas menos referenciadas y no procesos completos.
- El área de intercambio puede ser una partición de disco dedicada o un archivo en el disco. Un archivo de intercambio es una solución más flexible debido a su localización y que su tamaño puede ser cambiado fácilmente, aunque acceder a este tipo de archivo es poco eficiente.

### Gestión de memoria: reubicación

- **Código absoluto:** las direcciones se obtienen en tiempo de compilación. ~~En este momento, es necesario conocer las direcciones de ejecución del programa.~~
- **Reubicación estática:** las direcciones se obtienen cuando el programa se está cargando en memoria. Después de ser cargado en memoria, el programa no se puede mover a otra ubicación de memoria. El Swapping es posible solo si los procesos regresan a las mismas posiciones de memoria que usaban antes de ser intercambiados.
- **Reubicación dinámica:** las direcciones se obtienen en tiempo de ejecución. El proceso en ejecución utiliza referencias de memoria que no son referencias a posiciones de memoria física. No hay restricciones de Swap y hay una distinción entre el espacio de direcciones virtuales y el espacio de direcciones físicas.

La memoria debe estar protegida, por lo que debe tener hardware que respalde esta protección: dos registros límite o un registro límite y un registro base (registro de reubicación).

En los registros límite, cada dirección generada por un proceso en ejecución debe estar en el rango de los valores almacenados del registro, de lo contrario se produce una excepción. Estos valores se actualizan en un cambio de contexto y se almacenan en el PCB (Bloque de Control de Procesos).

En el otro caso, el registro base contiene el valor de la dirección física más pequeña (es decir, contiene la primera dirección de memoria de la tabla de páginas) y el registro límite contiene un rango de direcciones lógicas. Cada dirección lógica, que debe ser menor que el registro límite, se suma a la dirección del registro base. Estos valores se actualizan en un cambio de contexto y se almacenan en el PCB.

La segmentación y la paginación brindan protección y reubicación de memorias efectivas.

La **segmentación** es el esquema de administración de memoria que admite la vista de la memoria por parte del usuario. Un programa es una colección de segmentos y un segmento un bloque de tamaño variable que tiene el espacio de direcciones de un proceso.

Una **dirección lógica** está compuesta por un número de segmento y un offset. El número de segmento es el número de entrada de la tabla de segmentos para ese proceso. Cada entrada de la tabla de segmentos contiene la dirección base, es decir, la dirección física inicial para el segmento asociado y el tamaño del segmento. Un número de segmento es válido si y solo si es más pequeño que el registro que contiene el número de segmentos utilizados por el programa. Si el offset es más pequeño que el tamaño de segmento, la dirección física se obtiene sumando el offset a la dirección base, sino se producirá un error.

Estas direcciones tienen su propia **protección**: un bit de validez, los privilegios de lectura/escritura/ejecución y el modo kernel/usuario. Dado que los segmentos varían en longitud, la asignación de memoria es un problema dinámico de asignación de almacenamiento.

**Fragmentación en sistemas segmentados:**

- **Fragmentación interna:** ~~el tamaño del segmento es un múltiplo de un número fijo de bytes, por lo tanto, la memoria asignada puede ser un poco más grande que la memoria solicitada. Esta diferencia de tamaño es la memoria interna del segmento, pero no se utiliza,~~
- **Fragmentación externa:** ~~Después de asignar y liberar memoria, los bloques de memoria de varios tamaños se dispersan a través de la memoria. El espacio total de memoria existe para satisfacer una propuesta específica, pero no es contiguo.~~

Existen diferentes técnicas de asignación dinámica de memoria: First-fit, Next-fit, Best-fit y Worst-fit. First-fit y Best-fit funcionan mejor que Worst-fit en términos de velocidad y utilización del almacenamiento.

Los sistemas segmentados necesitan soporte del hardware, permite la reubicación dinámica, habilita la protección de la memoria y permite compartir datos o segmentos de código.

En un **sistema paginado**, el espacio de direcciones físicas puede no ser contiguo y se puede asignar memoria física al proceso siempre que esté disponible. En estos sistemas no hay fragmentación externa y se evita que los fragmentos de memoria sean de diferente tamaño.

Los sistemas paginados dividen la memoria física en bloques de tamaño fijo llamados **frames** y divide la memoria lógica en bloques del mismo tamaño llamados páginas. El SO configura una **tabla de páginas** para traducir direcciones lógicas a físicas. Un registro del procesador contiene la dirección base de la tabla de página para un proceso (registro base). La dirección generada por la CPU se divide en los siguientes campos: número de página y desplazamiento de página.

Los tamaños de página más pequeños significan más entradas en las tablas de páginas para los procesos, por lo que se pierde más memoria en las tablas de páginas. El SO debe tener en cuenta los frames libres y los frames asignados a procesos.

En cuanto a la protección de memoria, debido a la forma en la que se implementa la paginación, un proceso solo puede acceder a su propia memoria siguiendo su tabla de páginas, y su tabla de páginas solo puede ser cambiada por el SO.

La paginación permite compartir memoria entre los procesos: las entradas asociadas en sus tablas de páginas apuntan a las mismas páginas físicas.

Cada entrada de la tabla de páginas tiene otra información además de la dirección de memoria física. Esta entrada está formada por un bit de validez, un bit de acceso, un dirty bit, privilegios de lectura/escritura, etc.

La implementación de las tablas de páginas se puede realizar de dos formas diferentes:

- **Registros dedicados:** el procesador tiene registros que guardan las tablas de páginas de los procesos en ejecución. La tabla de páginas se guarda en el PCB y la traducción de direcciones es rápida.
- **En memoria:** las tablas de páginas se guardan en la memoria principal. El registro base apunta a la tabla de páginas del proceso en ejecución. La tabla de páginas se guarda en el PCB y la traducción de direcciones es lenta (Se realizan dos accesos a memoria: uno para la tabla de páginas y otro para los datos).

~~Algunas TLBs almacenan identificadores de espacio de direcciones en cada entrada de TLB. Esto, identifica de forma única cada proceso para proporcionar protección en el espacio de direcciones del proceso.~~

## Memoria Virtual

Todas las referencias a memoria son direcciones lógicas que son traducidas dinámicamente en direcciones físicas en tiempo de ejecución. Además, un proceso puede dividirse en varias partes que no necesitan ser contiguas en la memoria principal durante la ejecución.

La memoria virtual es un esquema de asignación de almacenamiento en el que la memoria secundaria se puede direccionar como si fuera parte de la memoria principal. ~~Las direcciones que un programa puede usar para hacer referencia a la memoria se distinguen de las direcciones que usa el sistema de memoria para identificar los sitios de almacenamiento físico y las direcciones generadas por el programa se traducen automáticamente a las direcciones de la máquina correspondiente.~~ El tamaño del almacenamiento virtual está limitado por el esquema de direccionamiento del sistema y por la cantidad de memoria secundaria disponible.

A la hora de ejecutar un programa, el SO lleva a memoria principal algunas partes del programa. La parte del proceso que se encuentra en memoria se llama **conjunto residente**. Cuando se necesita una dirección que no está en memoria principal, se genera una interrupción y el SO coloca el proceso en un estado de bloqueo. La parte del proceso que contiene la dirección lógica se lleva a la memoria principal. El SO emite una solicitud de lectura de E/S de disco y se envía otro proceso para que se ejecute mientras se realiza la E/S. Por último, se emite una interrupción cuando se completa la E/S del disco, lo que hace que el SO coloque el proceso afectado en el estado "Listo".

El **Thrashing** es un estado en el que el sistema utiliza la mayoría del tiempo intercambiando partes de procesos en vez de ejecutar instrucciones. Para evitar esto, el sistema intenta adivinar qué piezas son las que menos se utilizarán en un futuro.

El **Principio de Localidad** dice que los programas y datos referenciados dentro de un proceso tienden a agruparse. Solo se necesitan algunas partes de un proceso en un periodo de tiempo, por lo tanto, es posible adivinar qué piezas se necesitarán en un futuro. Además, se evita el Thrashing.

La memoria virtual suele asociarse a los sistemas que emplean **paginación**. Cada proceso, tiene su propia tabla de páginas, y cada entrada de la tabla de páginas contiene el número de frame de la página correspondiente en la memoria principal.

Con **tablas de páginas invertidas**, la parte del número de página de una dirección virtual se mapea con un valor hash (este valor apunta a la tabla de páginas invertida). Esta estructura se denomina invertida porque indexa las entradas por número de frame en vez de por número de página. Cada entrada de la tabla de páginas está formada por: número de página, identificador de proceso, bits de control y un puntero al valor del índice de la siguiente entrada en la cadena.

Cada referencia a memoria virtual puede provocar dos accesos a memoria física (uno para obtener la entrada de la tabla de páginas y otro para obtener los datos). Para solucionar esto, la mayoría de los esquemas de memoria virtual hacen uso de una caché especial de alta velocidad llamada **TLB**. La TLB sólo contiene algunas de las entradas de la tabla de páginas, por lo que cada entrada de la TLB debe incluir el número de página, así como la entrada completa de la tabla de páginas.

Cuanto menor sea el **tamaño de la página**, menor será la fragmentación interna. Sin embargo, se necesitan más páginas por proceso, lo que significa que las tablas de páginas serán más grandes. Para los programas grandes, una parte de las tablas de páginas de los procesos activos debe estar en la memoria virtual. Un tamaño de página mayor favorece una transferencia de datos en bloque más eficiente.

Utilizar un sistema segmentado presenta las siguientes ventajas:

- Simplifica el manejo de estructuras.
- Permite compartir datos entre procesos.
- Aporta protección.

En estos sistemas, cada entrada de la tabla de segmentos contiene la dirección inicial del segmento correspondiente en la memoria principal y la longitud del segmento. Se necesita un bit para determinar si el segmento ya está en la memoria principal y otro para determinar si el segmento ha sido modificado.

Además, los sistemas segmentados aportan protección, ya que cada entrada tiene una dirección base y una longitud, lo que permite controlar el acceso involuntario a la memoria, y compartición (algunos segmentos hacen referencia a varios procesos).

El diseño de la parte de gestión de memoria de un SO depende de tres áreas fundamentales:

- El uso o no de memoria virtual.
- Paginación, segmentación o ambas.
- Los algoritmos empleados para los distintos aspectos de gestión de memoria.

La **política de búsqueda** del SO determina cuándo debe traerse una página a la memoria. Hay dos tipos:

- **Demand paging:** sólo trae páginas a la memoria principal cuando se hace una referencia a una ubicación en la página. Se producen muchos fallos de página cuando el proceso se inicia por primera vez.
- **Prepaging:** trae páginas distintas a las demandadas por un fallo de página. Si las páginas de un proceso se almacenan de forma contigua en la memoria secundaria, es más eficiente introducir varias páginas a la vez.

La **política de ubicación** determina en qué lugar de la memoria real debe residir un trozo de proceso.

La **política de sustitución** se ocupa de la selección de una página en la memoria principal para ser reemplazada cuando se debe introducir una nueva página. El objetivo es que la página que se elimine sea la que menos probabilidades tenga de ser referenciada en un futuro próximo.

El **bloqueo de frames** consiste en que, cuando se bloquea un frame, la página almacenada actualmente en ese frame no puede ser reemplazada. Este bloqueo se consigue asociando un bit de bloqueo a cada frame.

Los algoritmos utilizados para la selección de una página a sustituir son:

- **Política óptima:** selecciona la página cuyo tiempo hasta la siguiente referencia es el más largo. Produce tres fallos de página después de llenar la asignación de frames.
- **LRU:** reemplaza la página que no ha sido referenciada durante más tiempo (difícil de implementar).
- **FIFO:** trata los frames de páginas asignados a un proceso como un buffer circular. Las páginas se eliminan en estilo round-robin. Se sustituye la página que lleva más tiempo en la memoria (fácil de implementar).
- **Política del reloj:** requiere un bit adicional (bit de uso). Cuando una página se carga por primera vez en la memoria o se hace referencia a ella, el bit de uso se pone a 1. El conjunto de tramas se considera un buffer circular. Cualquier frame con el bit de uso en 1 es pasado por el algoritmo.

El búfer de página mejora el rendimiento de la paginación y permite utilizar una política de páginas más sencilla. Una página sustituida no se pierde, sino que se asigna a una de estas dos listas: lista de páginas libres (lista de frames disponibles para leer) y lista de páginas modificadas (las páginas se escriben en clusters).

El sistema operativo debe decidir cuántas páginas traer a la memoria principal. Cuanto menor sea la cantidad de memoria asignada a cada proceso, más procesos podrán residir en la memoria. Un número reducido de páginas cargadas aumenta los fallos de página. El tamaño de estas páginas se puede decidir de dos formas:

- **Asignación fija:** da a un proceso un número fijo de frames en la memoria principal.
- **Asignación variable:** permite variar el número de frames de página asignados a un proceso a lo largo de su vida.

El **alcance** de una estrategia de sustitución puede clasificarse como global o local. Ambos tipos se activan por un fallo de página cuando no hay frames de páginas libres.

- **Local:** elige sólo entre las páginas residentes del proceso que generó el fallo de página.
- **Global:** considera todas las páginas desbloqueadas en la memoria principal.

Utilizando **asignación fija y alcance local**: si la asignación de memoria es demasiado pequeña, habrá una alta tasa de fallos de página. Si es demasiado grande, habrá pocos programas en la memoria principal.

Utilizando **asignación variable y alcance global**: el SO mantiene una lista de frames libres. El frame libre se añade al conjunto residente del proceso cuando se produce un fallo de página. Si no hay frames disponibles, el SO debe elegir una página que esté en memoria.

Utilizando **asignación variable y alcance local**: cuando se carga un nuevo proceso en la memoria principal, se le asigna cierto número de frames de página como su conjunto residente. Cuando se produce un fallo de página, se selecciona la página a sustituir de entre el conjunto residente del proceso que sufre el fallo. La decisión de aumentar o disminuir el tamaño del conjunto residente se basa en la evaluación de las probables demandas futuras de los procesos activos.

La **frecuencia de fallo de página** (PFF): tiempo virtual del proceso = tiempo transcurrido/número de procesos. Si ocurre un fallo de página: si  $t < T$  (umbral de tiempo) entonces aumenta la PFF (hay que añadir más páginas) y si  $t > T$  entonces disminuye la PFF (se descartan páginas con  $R = 0$ ).

El control de carga determina el número de procesos que residirán en la memoria principal. Es fundamental una gestión eficaz de la memoria, ya que, si hay demasiado pocos procesos, todos los procesos se bloquearán y se perderá mucho tiempo, y si hay demasiados, habrá mucho Thrashing.



# Procesos

Un **proceso** es una abstracción que se refiere a cada ejecución de un programa. Importante: un proceso no tiene por qué estar siempre en ejecución. También podemos ver a un proceso como la entidad que el SO crea para ejecutar un programa. Un proceso consiste en:

- Espacio de direcciones.
- Punto de control (siguiente instrucción a ejecutar).

Algunos sistemas permiten que un proceso tenga más de un punto de control: este proceso se ejecuta simultáneamente en varios lugares dentro de sí mismo (**hilos/threads**). El comando **pmap** de Unix nos muestra el espacio de direcciones de un proceso.

Para **gestionar los procesos**, el SO necesita asignar memoria para que se cargue el programa. Luego, como varios procesos pueden ejecutar el mismo programa, el SO tiene que identificarlos con un **descriptor** de proceso o un **identificador** de proceso. Cuando el proceso no se está ejecutando, el SO **mantiene la información** de ejecución en registros, memoria o recursos. El SO necesita conocer la lista de procesos del sistema y el estado en el que se encuentra cada uno de ellos (normalmente a través de listas de descriptores).

El **bloque de control del sistema** (SCB) es un conjunto de estructuras de datos utilizadas por el SO para controlar la ejecución de los procesos en el sistema. Suele incluir una lista de todos los descriptores del proceso, un puntero al proceso que se encuentra en la CPU, punteros a listas de procesos en diferentes estados y un puntero a una lista de descriptores de recursos.

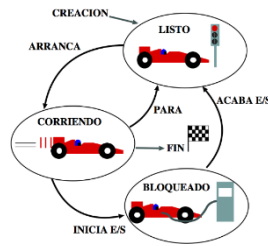
El SO se ejecuta cuando se produce una excepción, algún proceso pide al SO que haga algo, algún dispositivo externo requiere atención o periódicamente.

El SO dispone de una tabla de procesos donde guarda la información de cada proceso en el sistema. Cada entrada de esta tabla tiene información sobre **UN proceso**. El bloque de control de procesos (PCB) guarda los datos relevantes de un proceso que el SO utiliza para gestionarlo.

El PCB suele incluir:

- **Identificación:** identificador de proceso, identificador del proceso principal e identificadores de usuario y de grupo.
- **Programación:** estado del proceso, si el proceso está bloqueado y parámetros de programación (prioridad y otra información relevante).
- **Referencias a las regiones de memoria asignadas:** región de datos, de código y de pila.
- **Recursos asignados:** archivos abiertos y puertos de comunicación asignados.
- **Punteros para ordenar los procesos en listas.**
- **Información de comunicación entre procesos.**

Cada proceso del sistema comienza su vida con una llamada al sistema para crear un proceso. El proceso que hace esa llamada al sistema se llama proceso padre. Todos los procesos del sistema terminan con la llamada al sistema de terminar proceso.



En los sistemas modernos, una parte de la memoria secundaria se utiliza para intercambiar piezas de procesos de la memoria primaria, de modo que se pueda aumentar el grado de multiprogramación. Esto también permite que un proceso que no está cargado completamente en la memoria se ejecute. Los procesos pueden ejecutarse sin estar completamente cargados en la memoria.

Los procesos tienen diferentes estados:

- CPU, en marcha o en ejecución.
- Listo para ejecutarse o disponible.
- Bloqueado, dormido o en espera.
- Intercambiado o suspendido.

Hablando del estado de ejecución, el primero en la cola de procesos listos para ejecutar está programado para ejecutarse.

Hablando del estado de “listo para ejecutar”, hay cuatro escenarios diferentes:

- Se ha creado un nuevo proceso y entra en la cola de ejecutables.
- Desde la CPU: otro proceso está programado para ejecutarse a través de un cambio de contexto. decimos que el proceso se ha adelantado.
- Desde el bloqueo: el evento que el proceso estaba esperando ha ocurrido. A esta transición la llamamos desbloquear o despertar.
- Desde listo para **ejecutar**: el SO decide llevarlo a memoria primaria. Esta transición se llama swap in.

Hablando del estado de bloqueo, hay dos escenarios diferentes:

- Desde la CPU: el proceso hace alguna llamada al sistema que no puede completarse en ese momento, por lo que se bloquea.
- Desde el bloqueo/intercambiado: el SO intercambia un proceso bloqueado.

Los estados bloqueo/intercambio y **listo para ejecutar** pueden entrar cuando el SO decide intercambiar un proceso para liberar algo de memoria primaria.

Cuando un proceso realiza una llamada al sistema para crear un proceso, el SO debe:

1. Asignar un identificador al nuevo proceso.
2. Crear e inicializar su PCB
3. Actualizar el SCB para incluir el nuevo proceso
4. Asignarle memoria
5. Ponerlo en la cola de espera de ejecución

Un proceso se puede crear de diferentes maneras:

- Inicialización del sistema: muchos procesos se crean cuando arranca el sistema.
- Llamada al sistema dentro de algún proceso para crear un nuevo proceso.
- Petición explícita del usuario.
- Como parte de un proceso por lotes.

En todos estos escenarios los procesos se crean realmente de la misma manera: mediante una llamada al sistema.

Cuando un proceso se termina, se borra su PCB. El SO reclama todos los recursos asignados a ese proceso. Si el proceso tiene algunos procesos hijos puede esperar a que éstos terminen, terminarlos o dejarlos. Hay dos formas de terminación:

- Terminación normal: el proceso llama voluntariamente al sistema para terminarse.
- Terminación anormal: no está previsto en el código del proceso. El proceso se ve forzado a llamar al sistema para terminarse.

En un SO multiprogramado, varios procesos y/o hilos compiten por la CPU. Cada proceso está formado por una sucesión de ráfagas de CPU y E/S. El organizador (scheduler) es la parte del SO que decide qué proceso obtiene la CPU. Hay dos tipos de algoritmos de programación/organización:

- Algoritmos no **preferentes**: el proceso que se está ejecutando en ese momento permanece en la CPU hasta que termina su ráfaga de CPU.
- Algoritmos preventivos: el scheduler puede sacar de la CPU el proceso que se está ejecutando actualmente antes de la que termine su ráfaga de CPU.

Hay tres tipos de scheduler:

- Planificador a corto plazo: decide qué proceso entra en la CPU entre los procesos ejecutables.
- Planificador a medio plazo: en los sistemas de intercambio decide qué procesos intercambiados entrarán.
- Planificador a largo plazo: decide qué procesos del dispositivo se cargarán en la memoria principal.

Los objetivos de un planificador dependen del entorno en el que se utilice:

- Entornos por lotes: planificador a largo plazo que decide el orden en el que se procesan los trabajos. Su principal objetivo es ser eficiente y tener un gran rendimiento.
- Entornos interactivos: su objetivo principal es dar al menos una parte de la CPU a todos los procesos de manera oportuna.
- Entornos en tiempo real: algunos procesos tienen limitaciones de tiempo que deben cumplirse. Por lo general, se asigna a los procesos con necesidades especiales la mayor prioridad en el sistema.

Existen tres métodos para evaluar el comportamiento de un algoritmo en un sistema determinado: métodos analíticos, simulación e implantación. Existen diferentes medidas de tiempo que se necesitan para evaluar un algoritmo analíticamente:

- Tiempo de entrega ( $T_R$ ) =  $T_f - T_i$
- Tiempo de espera ( $T_E$ ) =  $T_R - T_{CPU} - T_{E/S}$
- Tiempo de servicio ( $T_S$ ) =  $T_R - T_E = T_{CPU} + T_{E/S}$
- Índice de servicio ( $i$ ) =  $T_S / T_R$

Otra opción es simular el comportamiento del sistema. Los datos de los procesos se generan de forma aleatoria o se toman como muestra de un sistema real. Hablando de un método de implantación, el algoritmo se implementa en un sistema en funcionamiento para ser evaluado. Los datos obtenidos corresponden a procesos reales en un sistema real. Hay diferentes algoritmos de planificación:

- **FCFS** (First Come First Served): Se implementa con una cola FIFO.
- **SJF** (Shortest Job First): Utilización sólo teórica, ya que necesita conocer de antemano la duración de las ráfagas de la CPU (Ejemplo página 55).
- **Planificación de prioridades**: SJF es un ejemplo de planificación de prioridades. En este tipo de algoritmos, el planificador selecciona el proceso de mayor prioridad entre todos los procesos listos para ejecutarse. Estas prioridades pueden ser: internas (asignadas por el SO), externas (asignadas por los usuarios) o mixtas. También se pueden considerar estáticas (su prioridad no cambia) o dinámicas (el sistema recalcula las prioridades de los procesos).
- **SRTF** (Shortest Remaining Time First): si uno de los nuevos trabajos tiene una ráfaga de CPU más corta que el tiempo restante del que está en CPU, el nuevo trabajo se queda con la CPU.
- **RR** (Round Robin): cada proceso tiene un límite de tiempo en su CPU llamado quantum. Los procesos listos para ejecutarse se organizan en una cola FIFO.
- **Cola multinivel**: tenemos una cola para cada nivel de prioridad. Cada cola puede tener su propio algoritmo de planificación. Además, se utilizan prioridades dinámicas.

Hablando de planificación en tiempo real, uno o varios dispositivos físicos generan estímulos a los que el sistema debe reaccionar en un tiempo limitado. Hay sistemas en los que estos tiempos deben cumplirse siempre (tiempo real duro) y otros en los que se puede perder algo de tiempo, aunque no sea deseable (tiempo real blando). El planificador debe organizar los procesos para que se cumplan los límites de tiempo. En un sistema de tiempo real existen dos tipos de eventos: periódicos (intervalos regulares) y no periódicos (imprevisibles). Un sistema de tiempo real puede responder a varios flujos de eventos periódicos. Si cada evento requiere demasiado procesamiento, puede resultar inmanejable.

Un proceso tiene una vida limitada: es creado por una llamada al sistema `fork`, termina con una llamada al sistema `exit` y puede ejecutar un programa con la llamada al sistema `exec`. Todo proceso tiene un proceso padre y puede tener uno o varios procesos hijo. Una estructura en forma de árbol con `init` como ancestro común a la mayoría de los procesos del sistema. Cuando un proceso termina, sus procesos hijos son heredados por `init`.

Los procesos en un sistema Unix presentan los siguientes estados: idle (creado, pero no preparado), listo para ejecutar, bloqueado o dormido, ejecutando por usuario, ejecutando en modo kernel, zombie (finalizado y esperando una llamada `wait`) y parado (para comprobar si está parado recibiendo una señal `SIGSTOP`, `SIGTSTP`...).

Para implementar el concepto de proceso, unix utiliza: un espacio de direcciones de usuario, información de control (estructura `proc`, `u_area`, pila del kernel y mapas de traducción de direcciones), credenciales, variables de entorno (un método alternativo para pasar información al proceso) y contexto de hardware (el contenido de los registros).

La estructura `proc` contiene la información de los procesos que el kernel necesita en todo momento. El `u_area` contiene información que sólo se necesita cuando el proceso está en marcha. Las credenciales de un proceso permiten al sistema determinar qué privilegios tiene un proceso. Cada usuario y cada grupo es identificado por un número (`uid` y `gid`). Un proceso tiene en realidad tres pares de credenciales (efectiva, real y guardada):

- Efectiva: regla de acceso a los archivos.
- Real y efectiva: regla de envío y recepción de señales. Una señal se recibe si el `uid` real o efectivo del proceso emisor coincide con el `uid` real del proceso receptor.
- Real y guardada: regula qué cambios en la credencial efectiva se pueden hacer a través de las llamadas al sistema `setuid` y `setgid`.

PÁGINA 216 PARA LAS FUNCIONES `FORK`, `EXEC` Y `EXIT`.

## Entrada/Salida

Los dispositivos de entrada/salida permiten a la CPU comunicarse con el mundo exterior. La comunicación entre la CPU y un elemento externo es similar a la comunicación con la memoria principal, ya que los datos se escriben y se leen. Sin embargo, el comportamiento es diferente. Esto se debe a que los datos no siempre están disponibles y a que el dispositivo puede no estar preparado para recibirlos.


En teoría, los dispositivos de E/S se comunican con la CPU a través de los buses del sistema. Los dispositivos se conectan a una pieza de hardware llamada controlador de dispositivos (device controller). El controlador de dispositivos admite/recibe órdenes de la CPU y es el encargado de transmitirlas al dispositivo. Cada controlador puede manejar varios dispositivos del mismo tipo. Los controladores se comunican con la CPU a través de unos registros o puertos que suelen incluir un registro de control (para enviar comandos al dispositivo), un registro de estado (para obtener información) y registros de datos (registros de entrada, registros de salida o registros bidireccionales). Estos dispositivos se encargan de la coordinación de datos entre la CPU y el periférico, la comunicación con la CPU, la comunicación con el dispositivo E/S, almacenamiento temporal de datos y la detección de errores.

Añadir un nuevo dispositivo sin perturbar el funcionamiento de un ordenador se consigue con:

- ~~Abstracción: elimina las diferencias entre dispositivos identificando unas pocas clases genéricas. Para cada clase se define una interfaz.~~
- ~~Encapsulación: los controladores de dispositivos (device drivers) se adaptan a las particularidades del dispositivo, pero exportan una de las interfaces estándar (sus funciones).~~
- Capas: el software de E/S está estructurado en capas.

Las llamadas al subsistema de E/S pueden ver el comportamiento de los diferentes dispositivos dentro de unas clases genéricas. Este subsistema está por encima de las capa de **controladores de dispositivos** y oculta las diferencias entre los distintos **controladores de dispositivos**. Este subsistema es independiente del hardware.

El software de E/S está estructurada según las siguientes capas:

- Software a nivel de usuario: interfaz con el usuario.
- Software independiente del dispositivo, se encarga de todas las tareas relacionadas con la asignación de espacio, el control de privilegios, el uso de la caché.
- **Controlador de dispositivo**: se comunica con el device **driver**. Permite que un dispositivo sea utilizado por el SO. 
- Controlador de interrupciones.
- Ejemplos en la página 15.

Según el método de comunicación entre la CPU y los dispositivos, podemos distinguir dos tipos de E/S: E/S explícita y E/S mapeada en memoria. Otra forma de clasificar los tipos de E/S es según la percepción que tenga el proceso respecto a cómo se realiza la E/S: E/S síncrona y E/S asíncrona.

En dispositivos mapeados en memoria, los dispositivos aparecen dentro del espacio de direcciones, que se comparte entre la memoria y los dispositivos. Utilizando un espacio de puertos de E/S separado, a cada registro de control de E/S se le asigna un número de puerto. Todos esos puertos constituyen el espacio de puertos. Normalmente, hay instrucciones especiales para acceder a este espacio.

Con una E/S síncrona, el proceso “siente” que tiene que esperar hasta que la operación se complete. Sin embargo, con una asíncrona, el proceso “siente” que no tiene que esperar y que algo le notificará que la operación se ha completado. Como la CPU es mucho más rápida que los dispositivos de E/S, una vez que la operación ha comenzado, el SO cede la CPU a una tarea diferente y deja al proceso que espera la E/S en espera.

Hay diferentes métodos para la E/S:

- E/S con votación: es la forma más sencilla de realizar una E/S. Para conseguir la sincronización, la CPU debe preguntar al dispositivo si tiene un nuevo dato que entregar o si está preparado para recibir un nuevo dato. Se pierde tiempo cada vez que preguntamos al dispositivo, por lo que es un método lento y se podrían perder datos.
- E/S con interrupción: el dispositivo solicita atención de la CPU con una interrupción. De esta forma, se detiene el proceso actual y se guarda su estado, se transfiere el control al procedimiento que realizó la interrupción, se ejecuta y luego se continúa la ejecución del proceso interrumpido.
- E/S usando DMA (Acceso Directo a la Memoria): el controlador DMA proporciona todas las señales de dirección y de control del bus. La CPU debe proporcionar al DMA el tipo de operación (lectura/escritura), la dirección de memoria para transferir los datos y la cantidad de bytes. Cuando se realiza la transferencia, el DMA lanza una interrupción a la CPU. El bus debe ser compartido entre la CPU y el controlador DMA. Existen tres métodos de compartición:
  - DMA en modo ráfaga: una vez que el DMA se apodera del bus, transfiere un bloque entero. Es el bloque más rápido.
  - DMA de robo de ciclo: cada vez que el DMA toma el bus, transfiere una palabra y luego devuelve el control del bus a la CPU.
  - Bus transparente: el DMA sólo utiliza el bus cuando la CPU no lo utiliza.

Un disco tradicional está formado por un conjunto de platos que giran juntos. Cada una de las superficies de los platos suele denominarse cara. Cada cara/superficie está compuesta por una serie de pistas circulares concéntricas. El conjunto con la misma

pista a lo largo de los diferentes lados constituye un cilindro. Cada cilindro contiene una serie de sectores. El sector es la unidad básica de E/S en un disco duro.

Normalmente, la numeración de los lados y los cilindros empieza por el 0, y los sectores se numeran a partir del 1. Tradicionalmente, el primer sector (lado 0, cilindro 0, sector 1) contiene una tabla que indica las diferentes áreas del disco.

En un sistema en el que se generan múltiples peticiones de E/S a los discos, dichas peticiones de E/S pueden ser planificadas. El tiempo de acceso tiene dos componentes principales:

- Tiempo de búsqueda: tiempo necesario para mover los cabezales hasta el cilindro que contiene el bloque deseado
- Latencia rotacional: tiempo de espera para que el disco gire el sector deseado hacia la cabeza del disco

Existen varios algoritmos para planificar el servicio de las peticiones de E/S del disco:

- FIFO
- SSTF (Shortest Seek Time First): las peticiones con menor tiempo de búsqueda se sirven primero.
- SCAN: el cabezal del disco comienza en un extremo y se mueve hacia el otro extremo. Sirve las peticiones hasta que llega al otro extremo, donde el movimiento del cabezal se invierte.
- C-SCAN: funciona de forma similar a SCAN, pero cuando el cabezal llega al final del disco, este vuelve al principio sin atender ninguna petición en el viaje de vuelta.
- C-LOOK (Variante de C-SCAN): el cabezal sólo llega hasta la última petición en cada dirección, luego invierte la dirección sin llegar primero hasta el final del disco.
- CFQ (Complete Fair Queuing): asigna diferentes colas a las peticiones de E/S del disco y a cada cola se le asigna un quantum. La longitud del quantum y el número de peticiones dependen de la prioridad.

En UNNIX, los dispositivos aparecen como un archivo. Los dispositivos reciben un inodo. Este inodo guarda dos números (major number y minor number), que indican, respectivamente, el manejador de dispositivos que se utiliza para acceder al dispositivo, y qué unidad se utiliza. Para acceder a los dispositivos podemos utilizar las mismas llamadas que se utilizan para acceder a los archivos (open, read, write) siempre que el proceso que llama tenga los privilegios adecuados.

Los descriptores de archivo 0, 1 y 2 corresponden respectivamente a la entrada estándar, la salida estándar y el error estándar de un proceso determinado. y el error estándar de un proceso determinado. La llamada al sistema dup duplica el descriptor de archivo y usa el número más pequeño disponible en la tabla de archivos abiertos.