

Sistemas operativos: Gestión
de memoria.

TEMA 17

ABACUS NT

Oposiciones 2021

Índice

1. Administración de memoria

1.1. Introducción

1.2. Eficiencia

1.3. Jerarquía de Memoria.

1.3.1. Funciones por niveles

1.4. Swapping

1.5. Administración de la memoria

1.5.1. Monoprogramación

1.5.2. Multiprogramación

1.5.3. Control de uso de la memoria

2. Memoria Virtual

2.1. Concepto

3. Algoritmos de reemplazo

3.1.1. Algoritmo óptimo

3.1.2. NRU

3.1.3. FIFO (First in- First out)

3.1.4. Algoritmos de la segunda oportunidad

3.1.5. Algoritmo del reloj:

3.1.6. LRU (Last Recently Used):

3.1.7. Simulación de LRU en software

3.1.8. Conjunto de trabajo

3.1.9. WSClock

3.2. Memoria caché

3.3. Niveles de caché

3.4. Localidad de referencia

3.4.1. Localidad temporal

3.4.2. Localidad espacial

3.5. Diseño

3.5.1. Política de ubicación

3.5.2. Política de extracción

3.5.3. Política de reemplazo

3.6. Protección de la Memoria

4. Gestión de la memoria en los sistemas operativos actuales

4.1. Gestión de Memoria en Android.

5. Conclusión

5.1. Relación del tema con el currículo

6. Bibliografía

1. Administración de memoria

1.1. Introducción

En informática, el término “Memoria” define un componente con una facultad similar a la de la memoria humana, esto es, la de retener información. En un sistema informático existen diversos tipos de memoria, diferenciadas según para qué son usadas y qué elementos la conforman.

De forma similar a como los humanos trabajamos con la memoria a corto y largo plazo, los ordenadores también tienen distintos tipos de memoria clasificada en una jerarquía.

Es función del sistema operativo velar por el buen uso de esa memoria, gestionándola adecuadamente, protegiéndola frente a usos malintencionados, aislando unos programas de otros, garantizando que hay memoria para todos gracias a un uso eficiente de la jerarquía de memoria y recuperando el control de la memoria que no está siendo utilizada.

Veremos en este tema todas esas cuestiones y finalmente veremos cómo gestiona la memoria un sistema operativo actual, tal como Android.

1.2. Eficiencia

La parte del SO que administra la memoria es el administrador de memoria. Su labor consiste en administrar la memoria con eficiencia, realizando las siguientes funciones:

- Asignar memoria a los procesos cuando lo necesiten
- Liberar memoria cuando los procesos terminen
- Gestionar el intercambio entre la memoria principal y la memoria externa
- Llevar un registro de las partes de memoria que se están utilizando
- Establecer mecanismos de protección

1.3. Jerarquía de Memoria.

La memoria es considerada un único componente funcional, compuesto de celdas que pueden almacenar **bits** de forma independiente, con dos operaciones básicas: **lectura y escritura**.

Sin embargo, las limitaciones en términos de **rendimiento, coste económico y velocidad** hacen necesaria una estructuración de la misma.

El autor **Pedro de Miguel de Anasagasti**, en su libro Fundamentos de los computadores, caracteriza la memoria según:

El coste por bit

Tiempo que se tarda en acceder a la información

Capacidad de almacenamiento o tamaño

Los valores deseados para una memoria son una gran capacidad de almacenamiento, un tiempo de acceso pequeño, y un precio reducido. Sin embargo, esta combinación no existe: **una memoria rápida es una memoria cara**. Esta limitación es válida desde que surgieron las primeras memorias hasta la actualidad. Las memorias rápidas, sin ciclos de refresco, con latencias casi inexistentes, requieren del uso de componentes caros: biestables integrados de alta conductividad, cuyo proceso de fabricación es caro y cuyos materiales son caros. De este imperativo se deduce que, **a más capacidad de memoria, más coste económico**.

Afortunadamente **no toda la memoria necesita ser accedida en un momento dado**. Esto posibilita que las instrucciones y datos requeridos por el procesador **se puedan jerarquizar en distintas memorias** dependiendo de la inmediatez necesaria para su proceso.

La jerarquía de memoria se conforma entonces con **memorias muy rápidas, caras y de baja capacidad** para los datos en proceso hasta **memorias relativamente lentes, de alta capacidad y bajo coste por bit** para los datos que no van a ser procesados en un futuro próximo (no en los próximos segundos). Las distintas memorias de menor a mayor capacidad y de mayor a menor velocidad y coste se clasifican en:

- **Registros** (del procesador)
- **Memoria caché** (estructurada en niveles: L1, L2, L3, etc.)
- **Memoria Principal** (RAM Dinámica)
- **Memoria secundaria** (Discos duros y de estado sólido)
- **Memoria masiva auxiliar** (Soportes de respaldo)

Los datos deben ser transferidos de una memoria a otra según la disponibilidad necesaria.

1.3.1. Funciones por niveles

Como vemos en la jerarquía de memoria, la **memoria interna** está compuesta por 3 niveles. Cada uno de los cuales tienen las siguientes funciones específicas:

- **Nivel 0** - Registros: Almacena datos sobre los que la unidad de control está trabajando directamente.
- **Nivel 1** - Caché: Memoria formada por circuitos integrados SRAM. A su vez se encuentra dividida en varios niveles. Se utiliza para minimizar los accesos a la memoria principal.
- **Nivel 2** - Memoria principal. Memoria formada por DRAM, cuya función es almacenar las instrucciones y datos que se están utilizando en la ejecución de un proceso.

1.4. Swapping

Con un sistema por lotes, la organización de la memoria en particiones fijas es simple y efectiva; pueden mantenerse suficientes trabajos en la CPU como para mantenerla ocupada todo el

tiempo, no hay razón para utilizar nada más complicado. Con el procesamiento con tiempo compartido o con multiprogramación la cosa cambia: normalmente hay más procesos que memoria para contenerlos, de forma que hay que gestionar parte de los mismos desde disco y traer a memoria principal únicamente aquellos fragmentos que se estén utilizando o se vayan a utilizar próximamente. El paso de los procesos de memoria principal a disco y su retorno es lo que se denomina como **intercambio** o **swapping** y es uno de los pilares en los que se basa la **gestión de memoria virtual**.

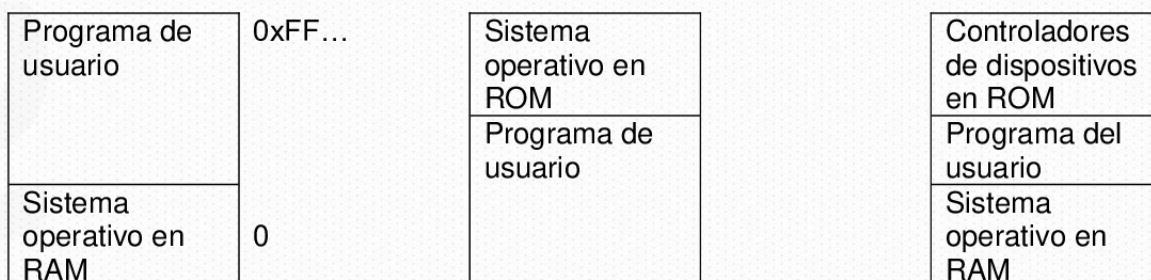
1.5. Administración de la memoria

Dependiendo del tipo de sistema, el administrador de memoria aplicará diferentes estrategias, con objeto de obtener el mejor aprovechamiento de la memoria. Veamos las más comúnmente aplicadas:

1.5.1. Monoprogramación

En los sistemas monoprogramables solamente hay un proceso en ejecución, el cual puede utilizar todo el espacio disponible (exceptuando el ocupado por el SO). Estos presentan normalmente una gran fragmentación interna (diferencia de tamaño entre la partición de memoria y el objeto residente dentro de ella).

La memoria se podrá organizar de las siguientes formas:



Con este tipo de gestión, en la memoria sólo puede encontrarse un proceso en un instante determinado. Cuando éste termina, es necesario sacarlo hacia un soporte externo y luego introducir el siguiente en ejecución. Por tanto, sólo las operaciones de E/S consumen gran parte del tiempo, mientras la CPU permanece ociosa.

1.5.2. Multiprogramación

En la multiprogramación varios procesos se van a ejecutar a la vez, por lo que el SO deberá establecer qué espacio de la memoria utilizan cada uno de ellos. Además, puede ocurrir que no haya suficiente espacio en la memoria principal para todos los procesos que están en ejecución. Para solucionar esta problemática, se suele mantener el exceso de los procesos en el disco. A la operación de

traspase de procesos de la memoria principal a la memoria externa y viceversa se le denomina intercambio o swapping.

Utilizar multiprogramación es necesario porque:

- Simplifica la programación de una aplicación al poder disociarla en dos o más procesos
- Se podrá ofrecer servicio interactivo a varias personas a la vez
- Los procesos suelen emplear mucho tiempo esperando que terminen las operaciones de E/S

Una forma de conseguir la multiprogramación consiste en dividir la memoria disponible en partes, a cada una de las cuales se le asignará un proceso o partes del mismo. Si todas las partes del programa se cargan juntas, se dice que la **asignación es contigua** y, por el contrario, si el programa está grabado en diferentes espacios, se dice que su **asignación es no contigua**.

De **asignación contigua** son los modelos de **particiones fijas (estáticas)** y **particiones variables (dinámicas)**. La **paginación y la segmentación** son modelos de **asignación no contigua**.

En multiprogramación, los procesos trabajan con direcciones lógicas en lugar de físicas.

Particiones estáticas:

Consiste en dividir el espacio disponible en varias particiones de tamaño fijo, cada una de las cuales podrá contener un proceso.

Presenta como contras tanto **fragmentación interna**, como **externa** (bloque libres de memoria en los que no se pueden ubicar nuevos procesos).

Particiones dinámicas:

En este método los procesos son introducidos en particiones consecutivas de memoria creadas a demanda.

No obstante, existirá fragmentación externa. Esto se soluciona con la **compactación**, aunque computacionalmente es bastante costosa.

Paginación

La mayoría de sistemas que utilizan memoria virtual (así lo hacen Unix System V, Linux, Windows NT, etc.) emplean la técnica de paginación.

Con este procedimiento la memoria principal se encuentra estructurada en marcos de páginas de un **tamaño fijo**, permitiendo que los procesos se puedan dividir en **páginas lógicas**, que se almacenarán en diferentes marcos de páginas.

La estructura de datos que relaciona cada página con el marco donde está almacenada es la **tabla de páginas**. El hardware de gestión de memoria (MMU, Memory Management Unit) usa esta tabla para traducir todas las direcciones que genera un programa.

Para controlar las partes libres y asignadas, el S.O. mantiene una **tabla de descripción de particiones**. Cada vez que una partición es creada se introduce en la tabla base el tamaño y el estado asignado.

Este tipo de gestión de memoria se denomina **MVT (Multiprogramación con un número variable de tareas)**. El soporte hardware que se necesita es al igual que el de asignación estática, dos registros para contener el límite superior e inferior de la memoria asignada.

Segmentación:

La ventaja de la segmentación respecto a la paginación es que los segmentos pueden ser de distinto tamaño, reduciendo así el problema de la fragmentación.

En los sistemas de memoria virtual que utilizan la técnica de segmentación, una dirección virtual se compone de dos partes, que se corresponden al desplazamiento dentro de un segmento y al segmento referenciado.

La segmentación, sin embargo, introduce un factor de complejidad en los sistemas de protección en los entornos multiusuario. Para resolverlos, se acude principalmente al uso de claves de protección, de forma que cada segmento es protegido por una clave que sólo conocen los procesos con permiso.

Paginación Segmentada

Los sistemas que combinan la paginación con la segmentación ofrecen las ventajas de ambas técnicas de organización del almacenamiento virtual. Los segmentos tienen por lo general y un tamaño múltiplo de páginas. Ahora las direcciones son tridimensionales: Segmento, número de página dentro del segmento y desplazamiento dentro de la página.

1.5.3. Control de uso de la memoria

Existen tres maneras de llevar a cabo el control del uso de la memoria:

- **Mapa de bits:** la memoria se divide en unidades de asignación de un tamaño determinado y a cada unidad se le asigna un bit, que será 0 si la unidad está libre y 1 si está ocupada.
- **Listas:** consiste en conservar en memoria una lista enlazada de segmentos de la misma asignados y libres.
- **Sistema compañero** (método de los colegas): consiste en ir dividiendo la memoria en fragmentos iguales, de manera que la asignación y desasignación es más rápida, pues cuando se busca un bloque determinado sólo se analizan los de tamaños más próximos.

2. Memoria Virtual

2.1. Concepto

En la memoria virtual el tamaño de la memoria que necesita un proceso puede exceder la cantidad de memoria física disponible para él, para ello, la información completa del proceso se mantiene en la memoria externa.

Este método se utiliza con una de las siguientes técnicas de gestión:

- Gestión de memoria segmentada (suele ser menos común debido a su complejidad)
- Gestión de memoria por páginas
- Gestión de memoria segmentada-paginada

3. Algoritmos de reemplazo

Con la memoria paginada y/o segmentada, en determinadas situaciones, será necesario sustituir una página por otra nueva, en estos casos, los algoritmos de reemplazo deciden qué páginas deben sustituirse, con objeto de minimizar el número de fallos de página (accesos a páginas que no se encuentran en la memoria principal).

El autor, **Andrews S. Tanenbaum**, en su libro Sistemas operativos modernos (2009), presenta los siguientes 9 algoritmos de reemplazo:

3.1.1. Algoritmo óptimo

Este algoritmo es el que ofrece mejores resultados, aunque es imposible de implementar. En éste se sustituye la página que va a tardar más tiempo en volver a ser referenciada.

3.1.2. NRU

El algoritmo de no usada recientemente (NRU): En este algoritmo cada página tiene dos bits asociados, un bit de referencia R y un bit M, de modificación. Con la combinación de estos bits se crean 4 categorías las cuales indican la preferencia a ser eliminadas.

3.1.3. FIFO (First in- First out)

Se sustituye la página que lleva más tiempo en memoria.

3.1.4. Algoritmos de la segunda oportunidad

Para mejorar el anterior algoritmo, se incorpora un bit de referencia R, de tal forma que antes de eliminar una página se examina su bit de referencia, si este es igual a 0 se sustituye, si está activo, se desactiva y se trata la página como si acabase de llegar a memoria.

3.1.5. Algoritmo del reloj:

Es un algoritmo similar al anterior, pero en el que se utiliza una cola circular.

3.1.6. LRU (Last Recently Used):

En este caso se sustituye la página menos usada en el pasado inmediato.

3.1.7. Simulación de LRU en software

Debido a que pocas máquinas disponen del hardware necesario para ejecutar el algoritmo anterior, existen simulaciones por software, como el no utilizada frecuentemente (NFU) o el envejecimiento (aging).

3.1.8. Conjunto de trabajo

En este algoritmo, en lugar de cargar páginas a demanda, se propone cargar un conjunto de páginas.

3.1.9. WSClock

Se trata de una mejora del algoritmo anterior y del reloj.

3.2. Memoria caché

Es una memoria rápida, cara y pequeña, formada por biestables.

Se encuentra situada entre el procesador y la memoria principal. En principio se puede considerar que funciona como un buffer de la memoria principal.

La caché se usa para almacenar una copia de aquellas palabras de la memoria principal que están siendo usadas actualmente (y que son las que se usarán con mayor probabilidad en un futuro cercano), ya que, debido a su reducido tamaño, no podemos almacenar toda la información que nuestro programa necesita.

La memoria caché trabaja almacenando los datos en sus chips, que son tomados de ella por la CPU cuando los necesita; al encontrarlos en la caché, se ahorra tener que ir a buscarlos a la RAM, redundando en un considerable ahorro de tiempo.

El procesador, cuando tiene que acceder a memoria, siempre mira antes si la información que busca se encuentra en la memoria caché. Pueden darse dos posibilidades:

- La información solicitada se encuentra en la caché: lee la copia en caché
- La información solicitada no se encuentra en la caché (fallo de caché): lee en la memoria principal la información buscada y guarda una copia en la caché.
A la hora de escribir el nuevo dato en la caché pueden ocurrir dos cosas:

- Hay espacio libre en la caché: se escribe el dato en las posiciones libres.
- No hay espacio libre en la caché: hay que buscar el dato a sustituir por el que acaba de llegar. Si algún dato de los que se van a sacar ha sido modificado, se debe actualizar la copia en memoria del mismo.

3.3. Niveles de caché

Los ordenadores actuales suelen montar hasta tres niveles de caché:

- L1: caché interna asociada a los núcleos de la CPU. Es utilizada para almacenar y acceder a datos e instrucciones importantes y de uso frecuente, agilizando los procesos al ser el nivel que ofrece un tiempo de respuesta menor. Se divide en dos subniveles:
 - Nivel 1 Data Cache: se encarga de almacenar datos usados frecuentemente.
 - Nivel 1 Instruction Cache: se encarga de almacenar instrucciones usadas frecuentemente.
- L2: caché interna al microprocesador, pero comúnmente compartida por varios núcleos al mismo tiempo.
- L3: este nivel de caché suele ser de menor velocidad y mayor capacidad que las anteriores y frecuentemente se sitúa dentro del encapsulado del procesador, pero en su periferia, compartida por todos los núcleos del mismo.

3.4. Localidad de referencia

Los programas manifiestan una propiedad que se explota en el diseño del sistema de gestión de memoria de los computadores en general y de la memoria caché en particular, la localidad de referencias: los programas tienden a reutilizar los datos e instrucciones que utilizaron recientemente. Una regla empírica que se suele cumplir en la mayoría de los programas revela que gastan el 90% de su tiempo de ejecución sobre sólo el 10% de su código. Una consecuencia de la localidad de referencia es que se puede predecir con razonable precisión las instrucciones y datos que el programa utilizará en el futuro cercano a partir del conocimiento de los accesos a memoria realizados en el pasado reciente. La localidad de referencia se manifiesta en una doble dimensión: temporal y espacial.

3.4.1. Localidad temporal

las palabras de memoria accedidas recientemente tienen una alta probabilidad de volver a ser accedidas en el futuro cercano. La localidad temporal de los programas viene motivada principalmente por la existencia de bucles.

3.4.2. Localidad espacial

las palabras próximas en el espacio de memoria a las recientemente referenciadas tienen una alta probabilidad de ser también referenciadas en el futuro cercano. Es decir, que las palabras próximas en memoria tienden a ser referenciadas juntas en el tiempo. La localidad espacial viene motivada fundamentalmente por la linealidad de los programas (secuenciamiento lineal de las instrucciones) y el acceso a las estructuras de datos regulares.

3.5. Diseño

En el diseño de la memoria caché se deben considerar varios factores que influyen directamente en el rendimiento de la memoria y por lo tanto en su objetivo de aumentar la velocidad de respuesta de la jerarquía de memoria. Estos factores son las políticas de ubicación, extracción, reemplazo y escritura.

3.5.1. Política de ubicación

Decide dónde debe colocarse un bloque de memoria principal que entra en la memoria caché. Las más utilizadas son:

Directa: al bloque i -ésimo de memoria principal le corresponde la posición i módulo n , donde n es el número de bloques de la memoria caché. Cada bloque de la memoria principal tiene su posición en la caché y siempre en el mismo sitio. Su inconveniente es que cada bloque tiene asignada una posición fija en la memoria caché y ante continuas referencias a palabras de dos bloques con la misma localización en caché, hay continuos fallos habiendo sitio libre en la caché.

Asociativa: los bloques de la memoria principal se alojan en cualquier bloque de la memoria caché, comprobando solamente la etiqueta de todos y cada uno de los bloques para verificar acierto. Su principal inconveniente es la cantidad de comparaciones que realiza.

Asociativa por conjuntos: cada bloque de la memoria principal tiene asignado un conjunto de la caché, pero se puede ubicar en cualquiera de los bloques que pertenecen a dicho conjunto. Ello permite mayor flexibilidad que la correspondencia directa y menor cantidad de comparaciones que la totalmente asociativa.

3.5.2. Política de extracción

La política de extracción determina cuándo y qué bloque de memoria principal hay que traer a memoria caché. Existen dos políticas muy extendidas:

- **Por demanda:** un bloque solo se trae a memoria caché cuando ha sido referenciado y no se encuentre en memoria caché.
- Con prebúsqueda: cuando se referencia el bloque i -ésimo de memoria principal, se trae además el bloque $(i+1)$ -ésimo. Esta política se basa en la propiedad de localidad espacial de los programas.

3.5.3. Política de reemplazo

Determina qué bloque de memoria caché debe abandonarla cuando no existe espacio disponible para un bloque entrante. Básicamente hay cuatro políticas:

- Aleatoria: el bloque es reemplazado de forma aleatoria.
- FIFO: se usa el algoritmo First In First Out (FIFO) (primero en entrar primero en salir) para determinar qué bloque debe abandonar la caché. Este algoritmo generalmente es poco eficiente.
- Usado menos recientemente (LRU): sustituye el bloque que hace más tiempo que no se ha usado en la caché, traeremos a caché el bloque en cuestión y lo modificaremos ahí.
- Usado con menor frecuencia (LFU): sustituye el bloque que ha experimentado menos referencias.

3.6. Protección de la Memoria

Cada proceso debe protegerse contra interferencias no deseadas de otros procesos, tanto accidentales como intencionadas. Así pues, el código de un proceso no puede hacer interferencia a posiciones de memoria de otros procesos, con fines de lectura o escritura sin permiso. Esta tarea la lleva a cabo principalmente el hardware y no el sistema operativo. Esto es debido a que el sistema operativo no puede anticiparse a todas las referencias a memoria que hará un programa, por lo que es necesaria una protección a varios niveles:

1. A nivel de Sistema Operativo.

Casi todos los sistemas operativos hacen uso de la protección de memoria para evitar que un software maligno modifique el código o los datos de otro programa; por ejemplo, un virus podría cambiar el vector de una interrupción y asignárselo a él mismo para posteriormente volverlo a redirigir a la rutina original si que se notase la intromisión.

Mediante el uso de llaves de asignación en la gestión segmentada de la memoria, y el mapeo eficaz en el caso de la memoria paginada se consigue establecer un primer nivel de protección.

Otra herramienta eficaz es la protección de la memoria auxiliar mediante la asignación de permisos extendidos como hacen los sistemas de archivos modernos (NTFS, EXT4, etc.)

2. A nivel de Memoria Caché.

La separación de Instrucciones y datos en el nivel L1 de caché interno al núcleo del procesador y la asignación del bit NX (No eXecution) en procesadores Intel a los datos ubicados en la misma se consiguen frenar vulnerabilidades de Overflow que posibilitaría convertir datos malintencionados en instrucciones precisas. Esa es una de las razones por las que las CPU actuales siguen la más pura

arquitectura Harvard hasta el nivel L1; Otra ventaja es acelerar el funcionamiento mediante buses separados para datos e instrucciones.

3. A nivel de CPU: Memoria protegida.

Actualmente todos los sistemas operativos se ejecutan con la CPU en modo de funcionamiento de Memoria Protegida que, al contrario del modo de Memoria Real, oculta las direcciones físicas y desplazamientos en los distintos chips de memoria, abstrayendo así la memoria y presentándola como una matriz homogénea, pero también ocultando detalles reales de la asignación de la misma. Además, el modo de memoria protegida cuenta con características particulares para multitarea y establece así mismo varios anillos de protección o privilegios sobre las instrucciones que se pueden ejecutar en el mismo. El anillo 0 de privilegio corresponderá al núcleo del sistema operativo, mientras que un proceso de usuario se ejecutaría en el anillo 3.

4. Gestión de la memoria en los sistemas operativos actuales

Windows

Utiliza una memoria virtual con paginación, aplica prepaginación mediante la tecnología superfetch, y utiliza el algoritmo LRU.

GNU/Linux

Dispone una memoria virtual paginada y segmentada. El algoritmo de reemplazo que utiliza se denomina PFRA (Page Frame Reclaiming Algorithm).

Gestión de la memoria en Android

4.1. Gestión de Memoria en Android.

La plataforma de Android se basa en la premisa de que la memoria disponible es una pérdida de memoria, de modo que intenta utilizar toda la memoria disponible en todo momento.

Android Runtime (ART) y la máquina virtual Dalvik usan las funciones de paginación y mapeo de memoria (mmapping) para administrar la memoria.

Jerarquía de Memoria

La jerarquía de Memoria en Android se basa en la gestión de tres niveles: RAM, zRAM y ROM.

- La memoria RAM es la memoria principal del dispositivo y se gestiona mediante paginación.
- La zona libre de RAM se utiliza como una memoria de intercambio SWAP en la que se almacenan datos y programas de forma comprimida, aumentando así su

aprovechamiento en término de capacidad, de ahí el nombre de zRAM, para diferenciarla.

- La memoria ROM está compuesta por dispositivos extremadamente lentos compuestos por memoria EEPROM Flash Integrada y opcionalmente una tarjeta SD.
- Existe un nivel cero en esta jerarquía compuesto por la memoria caché y registros del microprocesador, pero Android no los gestiona directamente.

Paginación de la memoria.

La memoria RAM está dividida en páginas. Por lo general, cada página tiene 4 KB de memoria.

Las páginas se consideran **libres o usadas**. Las páginas libres son memoria RAM sin usar. Las páginas usadas son memoria RAM que el sistema está utilizando de manera activa y se agrupan en las siguientes categorías:

- Almacenamiento en caché: Memoria respaldada por un archivo del almacenamiento (por ejemplo, código o archivos asignados a la memoria). Hay dos tipos de memoria caché:
 - Privada: Es propiedad de un proceso y no se comparte.
 - Compartida: Es utilizada por varios procesos.
- Anónima: Memoria no respaldada por un archivo en almacenamiento (por ejemplo, asignada por mmap() con la marca MAP_ANONYMOUS).

Las páginas a su vez pueden ser **Limpias o Sucias**: Las páginas limpias contienen una copia exacta de un archivo (o parte de un archivo) que existe en el almacenamiento. Una página limpia se convierte en una página sucia cuando ya no contiene una copia exacta del archivo (por ejemplo, del resultado de la operación de una app).

Es posible borrar las páginas limpias porque siempre se pueden volver a generar con los datos del almacenamiento. En cambio, no se pueden borrar las páginas sucias, ya que se perderían los datos.

Algoritmo de Reemplazo de página.

El algoritmo de reemplazo de página es el mismo que utiliza el núcleo Linux 2.6 y que se denomina “Page Frame Reclamation Algorithm”. Es una modificación del algoritmo LRU pero basado en listas y con prioridad para los procesos activos.

Las páginas de memoria en uso se dividen en dos listas por cada zona de memoria, una para aplicaciones en uso y otra para aplicaciones inactivas. Las páginas de memoria de la lista de inactivas pueden además ser marcadas como swappables o descartables dependiendo del tiempo de inactividad y previsión de uso.

El algoritmo PFRA Sólo se ejecuta cuando la memoria empieza a escasear, momento en el cual selecciona todas las páginas marcadas como “descartables/limpias” y a continuación se seleccionan las páginas menos usadas recientemente (LRU) de la lista de aplicaciones inactivas.

Este algoritmo se ejecuta de forma incremental, intentando liberar el máximo de memoria pero cargando en zRAM/swap páginas inactivas sucias, volcando a memoria masiva (limpiando) solo en las últimas iteraciones si no se alcanza liberar la memoria deseada.

La implementación en Android de este algoritmo la realiza el demonio **kswapd** que se ejecutará automáticamente al alcanzarse un umbral mínimo de memoria disponible y dejará de utilizarse cuando se alcance un umbral máximo determinado también de antemano.

Si tras la ejecución de **kswapd** aún no hay suficiente memoria para la carga de una determinada aplicación en memoria, se ejecutará un asistente de limpieza (LMK) que sacará de memoria procesos activos, según un nivel de prioridad asignado, comenzando con los procesos en segundo plano menos usados, y siguiendo incluso con la app de inicio, servicios, apps en primer plano, procesos del sistema, etc.

5. Conclusión

La gestión de memoria es un aspecto crucial del desempeño de un sistema operativo. Con aplicaciones ávidas de recursos, que no escatiman en el uso de la memoria principal, se hace imprescindible contar con un sistema operativo que haga buen uso de la misma, con algoritmos que garanticen un sistema fluido y sin acaparación de recursos.

Los sistemas operativos actuales cuentan con mecanismos de protección de la memoria frente a comportamientos erráticos o malintencionados del software, así como de recolectores que permiten liberar recursos asignados, pero no utilizados.

5.1. Relación del tema con el currículo

Este tema es aplicado en el aula en los módulos profesionales siguientes, con las atribuciones docentes indicadas (PES/SAI):

Grado Medio

- Sistemas operativos monopuesto (SMR) (PES/SAI)

Grado Superior

- Sistemas informáticos (DAM / DAW) (PES/SAI)
- Implantación de sistemas operativos (ASIR) (PES/SAI)

6. Bibliografía

- De Anasagasti, Miguel. "Fundamentos de la Computadora" 9^aed 2004 Edt. Paraninfo
- Patterson D.A. y Hennessy JL. "Estructura y diseño de computadoras: la interfaz hardware/software" 4^a Ed. (2005) Edt McGraw-Hill
- Prieto A, Lloris A, Torres JC. "Introducción a la Informática" 4^aed. (2006) Edt. McGraw-Hill

- Stallings W. "Organización y Arquitectura de Computadoras" (2006) 5^a Ed. Edt. Prentice-Hall
- Ramos A, Ramos MJ y Viñas S "Montaje y Mantenimiento de Equipos" (2012). Edt McGraw-Hill
- Jiménez Cumbreras, Isabel M^a "Sistemas Informáticos" 2^aEd (2018) Edt. Garceta