TEMA 17. SISTEMAS OPERATIVOS: GESTIÓN DE MEMORIA

0. INTRODUCCIÓN Y MARCO LEGISLATIVO

El presente tema se desarrolla en el marco de la Orden de 1 de febrero de 1996 (BOE de 13 de febrero de 1996), por la que se establece el temario que ha de regir los procesos selectivos para el ingreso en el Cuerpo de Profesores Técnicos de Formación Profesional, especialidad "Sistemas y Aplicaciones Informáticas", así como de la normativa vigente de acceso al Cuerpo de Profesores de Enseñanza Secundaria (Real Decreto 276/2007 y disposiciones de las convocatorias autonómicas, en este caso la de Aragón para 2027).
La gestión de memoria es una de las funciones nucleares de cualquier sistema operativo y se conecta directamente con los currículos LOMLOE de los ciclos formativos de la familia de Informática y Comunicaciones regulados por el Real Decreto 205/2023 (por el que se establece la ordenación de las enseñanzas de Formación Profesional) y desarrollados en Aragón mediante los currículos autonómicos correspondientes: en concreto, el módulo de *Sistemas Operativos Monopuesto* (CFGM SMR), *Sistemas Operativos en Red* (CFGM SMR) e *Implantación de Sistemas Operativos* (CFGS ASIR, RD 405/2023), donde el resultado de aprendizaje relativo a "gestión de la memoria y los procesos" es contenido curricular explícito.

1. CONCEPTO Y OBJETIVOS DE LA GESTIÓN DE MEMORIA

La memoria principal (RAM) es un recurso escaso y volátil que debe repartirse entre el sistema operativo y los distintos procesos en ejecución. El subsistema del núcleo encargado de esta tarea se denomina **gestor de memoria** (memory manager).
Sus objetivos fundamentales son:

- **Reubicación (relocation):** permitir que un proceso se cargue en distintas zonas de memoria física sin que ello afecte a su ejecución, mediante mecanismos de traducción de direcciones.
- **Protección:** impedir que un proceso acceda al espacio de direcciones de otro proceso o del propio núcleo sin autorización.
- **Compartición:** permitir que varios procesos accedan a una misma zona de memoria de forma controlada (por ejemplo, bibliotecas compartidas).
- **Organización lógica:** ofrecer al programador un espacio de direcciones lógico, independiente de la disposición física real de la memoria.
- **Organización física:** gestionar la jerarquía de memoria (registros, caché, RAM, memoria secundaria) de forma transparente al usuario.

Para lograrlo, el hardware incorpora una **Unidad de Gestión de Memoria** (MMU, Memory Management Unit), que traduce en tiempo de ejecución las *direcciones lógicas* (o virtuales) generadas por el programa en *direcciones físicas* reales de la memoria RAM.

2. MEMORIA MONOPROGRAMADA Y MULTIPROGRAMADA

En los primeros sistemas operativos, con **monoprogramación**, un único proceso ocupaba toda la memoria disponible además del propio sistema operativo, lo que suponía un aprovechamiento muy pobre de la CPU (ésta permanecía inactiva durante las operaciones de E/S).
Los sistemas actuales emplean **multiprogramación**: varios procesos residen simultáneamente en memoria, de forma que cuando uno queda bloqueado esperando una operación de E/S, la CPU puede pasar a ejecutar otro. Esto exige que el gestor de memoria resuelva el reparto del espacio disponible entre múltiples procesos concurrentes, con las consiguientes necesidades de protección y reubicación mencionadas en el punto anterior.

3. TÉCNICAS DE ASIGNACIÓN CONTIGUA

Históricamente, la primera aproximación consistió en asignar a cada proceso un bloque contiguo de memoria física.

3.1. Partición fija (o estática)

La memoria se divide de antemano en un número fijo de particiones, iguales o de distinto tamaño, establecidas en el momento del arranque del sistema. Cada proceso se ubica en una partición libre suficientemente grande.
Presenta el inconveniente de la **fragmentación interna**: si un proceso es más pequeño que la partición que ocupa, el espacio sobrante dentro de esa partición queda desperdiciado, ya que no puede asignarse a otro proceso.

3.2. Partición dinámica (variable)

Las particiones se crean en tiempo de ejecución con el tamaño exacto que solicita cada proceso. Esto elimina la fragmentación interna, pero introduce **fragmentación externa**: a medida que los procesos terminan y liberan memoria, quedan huecos dispersos que, aunque sumen espacio suficiente, no son contiguos y por tanto no sirven para alojar un nuevo proceso de mayor tamaño que cualquiera de esos huecos por separado.
La solución clásica a la fragmentación externa es la **compactación**: desplazar los procesos en memoria para agrupar todo el espacio libre en un único bloque contiguo. Es una operación costosa en tiempo de CPU, ya que implica copiar el contenido de memoria y actualizar los registros de reubicación de cada proceso.

3.3. Algoritmos de ubicación (colocación)

Cuando existen varios huecos libres capaces de alojar un proceso, el sistema operativo debe decidir cuál usar. Los algoritmos clásicos son:

- **Primer ajuste (First Fit):** se asigna el primer hueco de la lista que sea suficientemente grande. Es rápido, pero tiende a fragmentar la zona inicial de memoria.
- **Mejor ajuste (Best Fit):** se recorre toda la lista y se elige el hueco más pequeño que sea capaz de alojar el proceso. Minimiza el espacio desperdiciado en cada asignación individual, pero genera huecos residuales muy pequeños e inútiles, y es más lento al requerir recorrer toda la lista.
- **Peor ajuste (Worst Fit):** se elige el hueco más grande disponible, dejando un resto también grande y potencialmente reutilizable. En la práctica suele comportarse peor que las dos anteriores.

La simulación de estos algoritmos es un ejercicio práctico habitual en el aula de ASIR; a

continuación se muestra un ejemplo simplificado en Java de un algoritmo *Best Fit* sobre una lista de huecos libres:

```
public class GestorMemoria {

    // Representa un hueco libre: posición y tamaño en KB
    record Hueco(int inicio, int tamano) {}

    public static Hueco bestFit(List<Hueco> huecos, int tamanoProceso)
{
        Hueco mejor = null;
        for (Hueco h : huecos) {
            if (h.tamano() >= tamanoProceso) {
              if (mejor == null || h.tamano() < mejor.tamano()) {
                  mejor = h;
              }
            }
        }
        return mejor; // null si no hay hueco suficiente ->
fragmentación externa
    }
}
```

4. MEMORIA VIRTUAL Y ASIGNACIÓN NO CONTIGUA

La asignación contigua exige que un proceso completo resida en memoria física de forma ininterrumpida, lo que limita el número y tamaño de los procesos ejecutables simultáneamente.
La **memoria virtual** resuelve este problema separando el espacio de direcciones lógico que ve el programa del espacio físico realmente disponible, permitiendo que un proceso se ejecute aunque no esté completo en memoria RAM, con partes residiendo temporalmente en memoria secundaria (disco).
Las dos técnicas fundamentales de asignación no contigua, que sustentan la memoria virtual, son la **paginación** y la **segmentación**.

4.1. Paginación

La memoria física se divide en bloques de tamaño fijo llamados **marcos de página** (frames), y el espacio lógico de cada proceso se divide en bloques del mismo tamaño llamados **páginas**. Cualquier página de un proceso puede ubicarse en cualquier marco libre, eliminando así la fragmentación externa (aunque persiste una pequeña fragmentación interna en la última página de cada proceso).
Cada proceso dispone de una **tabla de páginas** que asocia cada página lógica con el marco físico que la contiene. Una dirección lógica se descompone en dos partes: número de página y desplazamiento dentro de la página; la MMU consulta la tabla de páginas para obtener el marco correspondiente y compone la dirección física.
Para acelerar esta traducción, que en caso contrario supondría un acceso adicional a memoria en cada referencia, se emplea una caché especializada llamada **TLB** (Translation Lookaside Buffer), que almacena las traducciones más recientes.
Cuando el espacio de direcciones es muy grande, la propia tabla de páginas puede ocupar

demasiado espacio; se recurre entonces a la **paginación multinivel** (o jerárquica), donde la tabla de páginas se organiza en varios niveles, de forma que solo se mantienen en memoria las tablas correspondientes a las regiones de direcciones realmente en uso.

4.2. Segmentación

La segmentación divide el espacio lógico del proceso en unidades de tamaño variable llamadas **segmentos**, que se corresponden con unidades lógicas del programa (por ejemplo, código, pila, montículo de datos o una función determinada). Cada segmento tiene su propio nombre o número y su propia longitud.
La dirección lógica se compone de un número de segmento y un desplazamiento; una **tabla de segmentos** asocia cada segmento con su dirección base en memoria física y su longitud, lo que además permite un control de protección independiente por segmento (por ejemplo, marcar un segmento de código como solo lectura/ejecución).
La segmentación, al asignar bloques de tamaño variable, reintroduce el problema de la fragmentación externa que la paginación había resuelto.

4.3. Segmentación paginada

Es un esquema híbrido que combina las ventajas de ambas técnicas: el espacio lógico se organiza en segmentos con significado lógico para el programador, y cada segmento se divide internamente en páginas de tamaño fijo. De este modo se obtiene la protección y organización lógica de la segmentación sin sufrir su fragmentación externa. Es el esquema empleado, con matices, por procesadores como la familia x86 en modo protegido.

5. ALGORITMOS DE REEMPLAZO DE PÁGINAS

Cuando la memoria virtual permite que no todas las páginas de un proceso estén presentes en memoria física simultáneamente, se produce un **fallo de página** (page fault) cada vez que se referencia una página que no está en RAM, lo que obliga al sistema operativo a traerla desde memoria secundaria. Si en ese momento no hay marcos libres, debe elegirse una página víctima a la que sustituir. De esta decisión se ocupan los **algoritmos de reemplazo de páginas**:

- **FIFO (First In, First Out):** se sustituye la página que lleva más tiempo en memoria, independientemente de su uso reciente. Es sencillo de implementar, pero puede sufrir la denominada *anomalía de Belady*, por la que aumentar el número de marcos disponibles puede, contraintuitivamente, incrementar el número de fallos de página.
- **Óptimo (OPT / Belady):** sustituye la página que no va a utilizarse durante más tiempo en el futuro. Ofrece el número mínimo teórico de fallos de página, pero es irrealizable en la práctica porque requiere conocer por adelantado la secuencia futura de referencias; se emplea únicamente como referencia teórica para evaluar otros algoritmos.
- **LRU (Least Recently Used):** sustituye la página que no se ha utilizado durante más tiempo, bajo la hipótesis de que el pasado reciente es un buen predictor del futuro próximo (principio de localidad). Su implementación exacta es costosa (requiere registrar el instante de cada acceso), por lo que en la práctica se recurre a aproximaciones.
- **Segunda oportunidad / Reloj (Clock):** aproximación eficiente a LRU. Cada página dispone de un bit de referencia (uso) que el hardware activa al acceder a ella. Las páginas se organizan en una lista circular recorrida por un puntero; al buscar víctima, si la página apuntada tiene el bit a 0 se sustituye, y si está a 1 se pone a 0 y se avanza al siguiente candidato, dando así "una segunda oportunidad" a las páginas recién usadas.

- **NRU (Not Recently Used):** clasifica las páginas en cuatro clases según los bits de referencia y de modificación (dirty bit), y elige la víctima de la clase más baja disponible, priorizando no descartar páginas modificadas (que exigirían escritura a disco).

Ejemplo de simulación en Java del algoritmo FIFO sobre una secuencia de referencias a páginas, útil como recurso didáctico en el aula:

```
import java.util.*;

public class SimuladorFIFO {

    public static int contarFallos(int[] referencias, int numMarcos) {
        Queue<Integer> marcos = new LinkedList<>();
        Set<Integer> presentes = new HashSet<>();
        int fallos = 0;

        for (int pagina : referencias) {
            if (!presentes.contains(pagina)) {
                fallos++;
                if (marcos.size() == numMarcos) {
                    int victima = marcos.poll();
                    presentes.remove(victima);
                }
                marcos.add(pagina);
                presentes.add(pagina);
            }
        }
        return fallos;
    }

    public static void main(String[] args) {
        int[] secuencia = {1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5};
        System.out.println("Fallos de página (3 marcos): " +
                contarFallos(secuencia, 3));
    }
}
```

6. LOCALIDAD DE REFERENCIA, HIPERPAGINACIÓN Y CONJUNTO DE TRABAJO

El buen funcionamiento de la memoria virtual se apoya en el **principio de localidad de referencia**: los programas, durante intervalos cortos de tiempo, tienden a acceder repetidamente a un subconjunto reducido de sus páginas (localidad temporal y espacial), lo que hace viable mantener en memoria solo una fracción del espacio total de direcciones de cada proceso.

Cuando el grado de multiprogramación es excesivo y no hay memoria suficiente para mantener el conjunto de páginas activas de cada proceso, se produce el fenómeno de **hiperpaginación** (thrashing): el sistema dedica la mayor parte del tiempo a intercambiar páginas entre RAM y disco, con una caída drástica del rendimiento efectivo de la CPU. El modelo del **conjunto de**

trabajo (working set) de Denning propone estimar, para cada proceso, el conjunto de páginas que ha referenciado en una ventana reciente de tiempo, y admitir un proceso a ejecución solo si su conjunto de trabajo cabe en la memoria disponible, evitando así la hiperpaginación.

7. GESTIÓN DE MEMORIA EN SISTEMAS OPERATIVOS REALES

Los sistemas operativos actuales aplican estos conceptos de forma combinada:

- **Windows** emplea paginación bajo demanda con un fichero de intercambio (pagefile.sys) de tamaño configurable, visible y ajustable desde el Administrador de tareas y el Panel de sistema, así como técnicas de compresión de memoria y "SuperFetch/SysMain" para precargar páginas de uso probable.
- **Linux** utiliza paginación multinivel, una partición o fichero de intercambio (swap), el algoritmo de reemplazo basado en listas activa/inactiva (aproximación a LRU) y herramientas de monitorización como free, vmstat o /proc/meminfo para observar el estado de la memoria.

Este contenido enlaza directamente con la explotación práctica de los sistemas monousuario y multiusuario que se aborda en los temas 20 y 21 de este mismo temario.

8. APLICACIÓN DIDÁCTICA EN EL AULA

Los contenidos de este tema son propios del módulo profesional de **Sistemas Operativos Monopuesto** (CFGM de Sistemas Microinformáticos y Redes) y de **Implantación de Sistemas Operativos** (CFGS de Administración de Sistemas Informáticos en Red), donde se trabajan resultados de aprendizaje relativos a la instalación, configuración y optimización de los parámetros de memoria virtual del sistema. También resulta aplicable, de forma transversal, en el módulo de **Sistemas Informáticos** como introducción a la arquitectura y funcionamiento del SO.

Como actividades de aula se pueden plantear: la modificación del tamaño del fichero de paginación en Windows y su efecto sobre el rendimiento; el análisis del uso de memoria y swap en Linux mediante comandos de monitorización; y la implementación en Java (como en los ejemplos anteriores) de simuladores de los algoritmos de ubicación y reemplazo de páginas, contrastando el número de fallos de página obtenido por cada algoritmo sobre una misma secuencia de referencias.

9. REFERENCIAS CRUZADAS CON OTROS TEMAS

- **Tema 4** (Memoria interna. Tipos. Direccionamiento): fundamento hardware de la jerarquía de memoria sobre la que actúa el gestor de memoria del SO.
- **Tema 15** (Sistemas operativos. Componentes. Estructura. Funciones. Tipos): sitúa la gestión de memoria como uno de los subsistemas del núcleo.
- **Tema 16** (Gestión de procesos): la memoria virtual y el espacio de direcciones son parte del contexto (bloque de control) de cada proceso gestionado por el planificador.
- **Tema 18** (Gestión de entradas/salidas): el intercambio de páginas con el área de swap es, en última instancia, una operación de E/S sobre memoria secundaria.
- **Tema 20 y 21** (Explotación y administración de sistemas operativos monousuario y multiusuario): aplicación práctica de la configuración de memoria virtual (pagefile, swap).
- **Tema 17 con el Tema 11 y 12** (Organización lógica de los datos): la abstracción de direcciones lógicas y las estructuras de tablas de páginas se apoyan en los mismos

principios de organización de datos estáticos (arrays/tablas) y dinámicos (listas enlazadas) vistos en esos temas.

10. CONCLUSIÓN

La gestión de memoria constituye uno de los pilares fundamentales del funcionamiento de un sistema operativo moderno, al permitir que múltiples procesos compartan de forma segura, eficiente y transparente un recurso físico limitado. La evolución desde la asignación contigua (partición fija y dinámica) hasta la memoria virtual basada en paginación y segmentación ha permitido superar las limitaciones de tamaño físico de la RAM, a costa de una mayor complejidad en el diseño de las estructuras de traducción de direcciones y de los algoritmos de reemplazo de páginas. El dominio de estos conceptos resulta imprescindible tanto para la comprensión teórica del funcionamiento interno de los sistemas operativos como para su aplicación práctica en la configuración y optimización de equipos reales, lo que justifica su presencia central en los currículos de los ciclos formativos de la familia de Informática y Comunicaciones.

BIBLIOGRAFÍA:

Implantación de Sistemas Operativos. Madrid: Editorial Síntesis.
Sistemas Operativos Monopuesto. 2.ª ed. Madrid: Editorial Síntesis.
Sistemas Operativos en Red. Madrid: Editorial Síntesis.
Temario Vol I, Vol II, Vol III FP Sistemas Informáticos Secundaria y FP. Editorial CEP AGAPEA

WEBGRAFÍA:

https://ikastaroak.birt.eus/ BirtLH – Formación Profesional online en el País Vasco
https://ioc.xtec.cat/educacio/recursos Recursos del Instituto Abierto de Cataluña
https://pilooras.com/temas
https://xjesus.net/
https://es.wikibooks.org/wiki/Temario_Oposici%C3%B3n_Profesor_FP_Inform%C3%A1tica
Nota: La disponibilidad y vigencia de algunos de estos recursos web puede haber variado, pudiendo encontrarse el contenido desactualizado o el enlace roto en el momento de la consulta.