Sistemas Operativos

Simuladores e Inteligencias Artificiales Generativas en la Docencia de Sistemas Operativos

Lorena Otero-Cerdeira
0000-0002-7829-4104

María Encarnación González-Rufino
0000-0001-9095-8367

Francisco Javier Rodríguez-Martínez
0000-0002-6623-1704

Alma Gómez-Rodríguez
0000-0001-7971-8282

David Rodríguez-Martínez
0009-0003-5892-5612

2025

Universidade Vigo

Escuela Superior de Ingeniería Informática
Universidade de Vigo

Centro de Posgrao e
Formación Permanente

ISBN: 978-84-1188-054-1
Ourense, on March 11, 2025

Contents

Agradecimientos **vii**

Prólogo **ix**

1 Introducción **1**

I Planificación de Procesos **5**

2 Contexto Teórico **7**

2.1 Introducción . . . 7
2.2 Concepto de proceso y sus estados . . . 7
2.3 Planificación Apropiativa y No Apropiativa . . . 10
2.3.1 Planificación Apropiativa . . . 10
2.3.2 Planificación No Apropiativa . . . 11
2.4 Prioridades . . . 12
2.5 Quantum . . . 13

3 Algoritmos de Planificación de la CPU **15**

3.1 Algoritmos de Planificación . . . 15
3.1.1 FIFO: First Input, First Output o FCFS: First Come, First Served . . . 15
3.1.2 SJF: Shortest Job First . . . 16
3.1.3 SRT: Shortest Remaining Time . . . 16
3.1.4 RR: Round Robin . . . 17
3.1.5 Colas de Niveles Múltiples (Colas de prioridad múltiple) . . . 18
3.1.6 Colas de Retroalimentación de niveles múltiples . . . 20
3.2 Ejercicios: Carga de Trabajo, Diagrama de Ocupación . . . 21
3.2.1 Generación automática de ejercicios con IA generativa . . . 23

4 Simulador 1: Generación del Diagrama de Ocupación **25**

4.1 Objetivo . . . 25
4.2 Arquitectura y Tecnologías . . . 26

4.3 Diseño 28
4.4 Uso del Simulador 30

5 Simulador 2: Identificación del Algoritmo de Planificación 35
5.1 Objetivo 35
5.2 Diseño 37
5.2.1 Reglas de Validación del Diagrama 37
5.2.2 Reglas de Análisis del Algoritmo de Planificación 39
5.3 Arquitectura y Tecnologías 43
5.4 Uso del Simulador 46

II Gestión de Memoria Virtual 53

6 Introducción 55

7 Conceptos básicos sobre memoria virtual 57

8 Estrategias para gestionar la Memoria Virtual 63
8.1 Paginación 63
8.2 Segmentación 69
8.3 Paginación/Segmentación (Segmentación paginada o Paginación segmentada) 75

9 Simulador 1: Paginación 85
9.1 Objetivo 85
9.2 Arquitectura y Tecnologías 86
9.3 Uso del Simulador 90
9.4 Generación automática de ejercicios con IA generativa 96

10 Simulador 2: Segmentación 103
10.1 Objetivo 103
10.2 Arquitectura y Tecnologías 104
10.3 Uso del Simulador 109
10.4 Generación automática de ejercicios con IA generativa 116

11 Simulador 3: Paginación/Segmentación 121
11.1 Objetivo 121

11.2 Arquitectura y Tecnologías 122
11.3 Uso del Simulador 126
11.4 Generación automática de ejercicios con IA generativa 142

Bibliography **147**

Agradecimientos

Esta publicación ha sido financiada por las *Ayudas Económicas para las Actividades de los Grupos de Innovación Docente de la Universidade de Vigo (2024)*.

Prólogo

El presente libro tiene un enfoque didáctico y pretende ser un manual de ayuda para las personas interesadas en los sistemas operativos. Tanto los docentes de la materia, como los estudiantes pueden beneficiarse de sus contenidos para una mejor comprensión tanto de la planificación de procesos de la CPU como de la gestión de la memoria virtual. Los contenidos del libro se han concebido como un curso introductorio a los Sistemas Operativos en el ámbito universitario, pero puede adaptarse para su uso en otros niveles educativos.

La estructura del trabajo permite que un lector interesado pueda ir adquiriendo en secuencia los conocimientos necesarios para una mejor comprensión de los sistemas operativos. Sin embargo, cada uno de los capítulos es autocontenido, por lo que puede ser usado de modo individual para profundizar en dicho tema. Así, el primer bloque del texto se centra en los algoritmos de planificación de procesos que permiten determinar al sistema operativo el orden en que se usará el procesador por los procesos que lo soliciten, intentado lograr un uso optimizado del tiempo de procesamiento y su reparto entre los distintos solicitantes. El segundo bloque del libro se focaliza en la gestión de la memoria virtual, que facilita que se pueda hacer uso de una cantidad de memoria incluso superior al hardware disponible en la máquina. Adicionalmente, en este bloque se presentan una serie de herramientas software, disponibles en la red, que permiten simular estos procesos de gestión de la memoria virtual, para que el lector logre una comprensión profunda de los conceptos introducidos. Además, como innovación, se aporta al final de muchos de los capítulos, claves para que se puedan generar nuevos ejercicios mediante el uso de la Inteligencia Artificial (IA) generativa, de modo que el estudiante pueda obtener nuevos ejemplos que le permitan una mejor y más profunda comprensión de los conceptos teóricos del capítulo. Se realiza así un uso correcto y novedoso de la IA para complementar la labor de enseñanza-aprendizaje, frente al uso en algunos casos no adecuado de la misma, lo que pretende también animar al lector a explorar las facilidades que puede proporcionarnos.

Esperamos que este libro sirva de guía al viaje al centro del computador: al software fundamental para la gestión del hardware. ¡Anímate a hacer ese viaje con nosotros, lector!

I

Introducción

El propósito de la enseñanza reside en facilitar la comprensión del receptor de los conceptos expuestos. Para lograr este objetivo, el docente debe deliberar cuidadosamente sobre qué metodología de aprendizaje podría ser más efectiva.

La metodología de aprendizaje conocida como Aprendizaje Basado en la Simulación [Aln13] se enfoca en recrear escenarios para facilitar la comprensión de conceptos teóricos. Los simuladores destacan como la principal herramienta pedagógica empleada en este enfoque educativo.

Sistemas Operativos (S.O.) es una asignatura obligatoria en los planes de estudio del Grado en Ingeniería Informática cuyo objetivo principal es que los estudiantes conozcan el funcionamiento interno de un S.O., propósito difícil de alcanzar debido a la complejidad de las funciones que desempeña.

Durante tres cursos se probó a mitigar esta dificultad haciendo que el alumno implementase una versión acotada de los algoritmos empleados por los S.O. El resultado no fue satisfactorio, principalmente porque el estudiante obtenía un entendimiento limitado del funcionamiento a la estrategia o algoritmo asignado, además de dedicarle un tiempo excesivo, debido a que la asignatura está englobada en el primer cuatrimestre del segundo curso, lo que implica que el alumnado aún no ha adquirido las habilidades prácticas requeridas en el ámbito de la programación.

Esto obligó a cambiar el enfoque, optando por ofrecer al alumnado herramientas ya desarrolladas que le permitiesen explorar los conceptos de los S.O. Por ello, se decidió diseñar e implementar un conjunto de simuladores, herramientas software interactivas, con los que los estudiantes visualizan y experimentan diversas estrategias empleadas por los S.O. en la planificación del procesador y en la gestión de la memoria virtual.[1]

Desafíos en el Estudio de Sistemas Operativos

Todo profesional en el ámbito de la informática debe poseer conocimiento sobre las funciones, estructura y diseño de los S.O. Este logro demanda un esfuerzo considerable debido a múltiples razones, entre ellas caben destacar:

- **Complejidad:** los S.O. son software que interactúan con el hardware y gestionan los recursos

[1] Se desea mencionar que, en este documento, todas las referencias a alumnos deben interpretarse de manera inclusiva, abarcando tanto a estudiantes femeninos como masculinos.

(memoria, procesadores, etc.). Entenderlo requiere comprender conceptos avanzados de computación (concurrencia, sincronización, etc.) y estructuras internas soportadas por los S.O.

- **Abstracción:** muchos conceptos en S.O. resultan difíciles de visualizar o comprender intuitivamente, ya que no tienen una representación física directa lo que complica su asimilación.
- **Experimentación limitada:** probar con S.O. en un entorno real puede ser acotado debido a recelos sobre la estabilidad del sistema o la disponibilidad del hardware específico.

Herramienta de aprendizaje: Simulador

En el ámbito educativo, un simulador [CLD13] es una herramienta que recrea una situación para permitir experimentar y practicar habilidades, resolver problemas o aprender conceptos. De forma resumida, sus beneficios son:

- **Retroalimentación instantánea:** los estudiantes reciben respuesta inmediata a sus acciones y decisiones dentro del simulador, lo que les permite corregir errores y mejorar sus conocimientos.
- **Experiencia práctica en un entorno libre de riesgos:** los estudiantes prueban con situaciones de la vida real de manera segura y controlada, facilitando la adquisición de habilidades y conocimientos prácticos.
- **Motivación y compromiso:** la interactividad y el carácter inmersivo de los simuladores pueden aumentar la motivación y el compromiso de los estudiantes con el proceso de aprendizaje.

Explorando los S. O. a través de Simuladores Didácticos Interactivos.

Dos de los cometidos fundamentales de los S.O. son la planificación del procesador y la gestión de la memoria virtual. En los siguientes capítulos se presentan los conceptos teóricos que dan soporte a los simuladores desarrollados en estos ámbitos y se describen en detalle dichos simuladores. Estos simuladores se utilizan durante todo el estudio de estos dos componentes del S.O. Primero para exponer los conceptos teóricos y, posteriormente, para describir cómo funcionan los distintos algoritmos y estrategias empleadas. También los estudiantes pueden usarlos parar realizar los ejercicios adicionales que se le proponen, profundizando en la comprensión de los mismos al poder analizar y comparar los resultados que ofrece cada simulación. Estos simuladores constituyen una herramienta interactiva de gran valor para los estudiantes, ya que les ayuda a comprender y analizar el comportamiento de estos dos componentes primordiales de los S.O.

En lo relativo a la *Planificación del Procesador*, se han desarrollado dos simuladores:

1. Aplicar algoritmos de planificación a diversas cargas de trabajo obteniendo los respectivos diagramas de ocupación de la CPU. Este simulador se describe en el capítulo 4 y está disponible en `https://n9.cl/egprg`

2. Determinar los algoritmos de planificación aplicados a una carga de trabajo a partir de un diagrama de ocupación de la CPU, razonando los motivos por los que no corresponde dicho diagrama a otros algoritmos. Este simulador se describe en el capítulo 5 y está disponible en `https://n9.cl/hbc9j7`.

En lo relativo a la *Gestión de la Memoria Virtual*, se han desarrollado tres simuladores:

1. Paginación. Todos los bloques tienen el mismo tamaño (páginas). Este simulador se describe en el capítulo 9 y está disponible en `https://n9.cl/epr9z`.
2. Segmentación. Los bloques tienen diferentes tamaños (segmentos). Este simulador se describe en el capítulo 10 y está disponible en `https://n9.cl/ryz1m`.
3. Segmentación Paginada. Cada bloque es un segmento de tamaño variable compuesto de páginas de tamaño fijo. Este simulador se describe en el capítulo 11 y está disponible en `https://n9.cl/lxvm7`.

Es fundamental resaltar que los diversos proyectos que han dado lugar a los distintos simuladores previamente identificados son el resultado de un trabajo acumulado a lo largo de varios años. El diseño y la construcción de estos simuladores no se llevaron a cabo en un mismo momento temporal, lo que explica que no compartan la misma pila tecnológica. Además, desde una perspectiva metodológica, estos proyectos han experimentado una evolución lógica, producto de la constante innovación y el proceso de mejora continua. Dicho contexto debe tenerse en cuenta para entender los apartados del presente libro en el que presentaremos desde un punto de vista técnico los distintos simuladores facilitados como apoyo al aprendizaje de los estudiantes.

El trabajo recogido en este libro no está concluido, sino que se sigue desarrollando y ampliando. En concreto, se está desarrollando una herramienta para integrar los simuladores en la plataforma de teleformación y realizar un estudio sobre su impacto en la docencia, así como incorporar este aprendizaje basado en simulación para mejorar la comprensión de otros componentes del S.O.

Cabe destacar que en este libro se utilizan las inteligencias artificiales generativas para configurar ejercicios que alimenten a estos simuladores. Estas IA permiten la creación automática de situaciones y datos realistas, ofreciendo una amplia variedad de escenarios para que los estudiantes puedan explorar diferentes circunstancias. Esto facilita el estudio de condiciones que, de otro modo, serían difíciles de replicar en un entorno educativo estándar debido a la complejidad o al tiempo requerido para diseñarlas manualmente. Al automatizar la generación de ejercicios, los estudiantes pueden enfocarse más en el análisis y la resolución de problemas, en lugar de dedicar tiempo excesivo a preparar los escenarios.

Además, el uso de IA generativas aporta un nivel de personalización que permite adaptarse a los conocimientos y habilidades previas de cada estudiante, lo cual es esencial en el proceso de aprendizaje. De esta forma, los simuladores configuran actividades según distintos niveles de dificultad y combinaciones

de parámetros, proporcionando una experiencia enriquecedora y desafiante. Esta capacidad de generar ejercicios bajo múltiples condiciones posibilita la exploración de cuestiones complejas y la experimentación con distintos enfoques, contribuyendo a una comprensión más profunda de los conceptos fundamentales.

Part I

Planificación de Procesos

Los caminos de los procesos no son misteriosos si estudiamos la documentación técnica.

2 Contexto Teórico **7**

2.1 Introducción . . . 7
2.2 Concepto de proceso y sus estados . . . 7
2.3 Planificación Apropiativa y No Apropiativa . . . 10
2.3.1 Planificación Apropiativa . . . 10
2.3.2 Planificación No Apropiativa . . . 11
2.4 Prioridades . . . 12
2.5 Quantum . . . 13

3 Algoritmos de Planificación de la CPU **15**

3.1 Algoritmos de Planificación . . . 15
3.1.1 FIFO: First Input, First Output o FCFS: First Come, First Served . . . 15
3.1.2 SJF: Shortest Job First . . . 16
3.1.3 SRT: Shortest Remaining Time . . . 16
3.1.4 RR: Round Robin . . . 17
3.1.5 Colas de Niveles Múltiples (Colas de prioridad múltiple) . . . 18
3.1.6 Colas de Retroalimentación de niveles múltiples . . . 20
3.2 Ejercicios: Carga de Trabajo, Diagrama de Ocupación . . . 21
3.2.1 Generación automática de ejercicios con IA generativa . . . 23

4 Simulador 1: Generación del Diagrama de Ocupación **25**

4.1 Objetivo . . . 25
4.2 Arquitectura y Tecnologías . . . 26
4.3 Diseño . . . 28
4.4 Uso del Simulador . . . 30

5 Simulador 2: Identificación del Algoritmo de Planificación **35**

5.1 Objetivo . . . 35
5.2 Diseño . . . 37
5.2.1 Reglas de Validación del Diagrama . . . 37
5.2.2 Reglas de Análisis del Algoritmo de Planificación . . . 39
5.3 Arquitectura y Tecnologías . . . 43
5.4 Uso del Simulador . . . 46

II

Contexto Teórico

En este capítulo se presentan de forma general los conceptos teóricos que se emplean en esta primera parte relativos a las herramientas de simulación desarrolladas para trabajar con planificación del procesador.

2.1

Introducción

La planificación en sistemas operativos es un conjunto de políticas y mecanismos esenciales para determinar el orden en que se completan las tareas en un sistema, con el objetivo primordial de optimizar su rendimiento. La necesidad de una planificación eficiente radica en la capacidad de gestionar múltiples procesos de manera equitativa y eficaz, asegurando que todos los procesos reciban tiempo de CPU adecuado sin causar postergaciones indefinidas.

Los principios fundamentales que deben garantizarse incluyen la justicia, para que todos los procesos sean tratados por igual; la maximización de la capacidad de ejecución, permitiendo servir al mayor número posible de procesos en el menor tiempo; y el equilibrio en el uso de recursos, favoreciendo a los procesos que utilizan recursos infrautilizados. Adicionalmente, es crucial asegurar que los procesos de alta prioridad reciban el tratamiento adecuado y que se evite la postergación indefinida mediante técnicas como el envejecimiento, que incrementa la prioridad de los procesos en espera para asegurar su ejecución.

En este capítulo se presentan de forma general los conceptos teóricos que se emplean en los Capítulos 4 y 5 de esta primera parte relativos a las herramientas de simulación desarrolladas para trabajar con planificación del procesador.

2.2

Concepto de proceso y sus estados

En el ámbito de los sistemas operativos, la secuencia de acciones o actividades resultantes de la ejecución de un conjunto predefinido de instrucciones es lo que denominamos un proceso. Puede verse como la instancia

activa de un programa que se encuentra en memoria y consume recursos del sistema como la CPU, la memoria y los dispositivos de entrada/salida.

Desde su creación hasta su finalización un proceso atraviesa tres estados básicos, tal y como se recoge en la figura 2.1:

- *Activo:* El proceso tiene asignado un procesador y está ejecutando sus instrucciones. Es el único estado en el que el proceso consume tiempo de CPU de forma activa.
- *Listo:* El proceso se encuentra en memoria principal y está listo para ser ejecutado por la CPU. Sin embargo, todavía no tiene asignado un procesador. El proceso se encuentra en la cola de procesos en estado listo, esperando su turno para ser ejecutado. Esta lista se ordenará según algunos criterios, dependiendo del algoritmo de planificación de la CPU que utilice el sistema operativo.
- *Bloqueado:* El proceso se encuentra esperando un evento externo para continuar su ejecución. Este evento puede ser la finalización de una operación de entrada/salida, la recepción de una señal o la disponibilidad de un recurso compartido. El proceso se coloca en una lista de procesos en estado bloqueado a la espera de que suceda el evento por el que espera.

Figure 2.1: Estados básicos de los procesos y sus transiciones.

Los cambios de estado de un proceso pueden darse en las siguientes situaciones, que también se ven en la figura 2.1, son los siguientes:

- Cuando un trabajo es aceptado por el sistema, se crea su proceso correspondiente y entra en estado "Nuevo".
- Cuando un proceso está preparado para usar el procesador, pasa del estado "Nuevo" a estado "Listo".
- Cuando el proceso es elegido por el planificador para asignarle la CPU, pasa de estado "Listo" a estado

"Activo".

- Cuando el proceso está en estado "Activo", pueden darse tres situaciones:
  - El proceso completa su ejecución y abandona el sistema. En este caso pasa de estado "Activo" a "Terminado".
  - El proceso puede ser expulsado de la CPU, pasando de estado "Activo" a "Listo".
  - El proceso puede requerir que se produzca un evento o una operación de entrada/salida, por tanto debe esperar y pasa de estado "Activo" a "Bloqueado".
- Cuando el proceso está en estado "Bloqueado" y llega el evento por el que estaba esperando o finaliza la operación de entrada/salida, pasa de estado "Bloqueado" a "Listo".

Es importante destacar que el sistema operativo mantiene dos listas actualizadas: una para los procesos en estado "Listo" y otra para los procesos en estado "Bloqueado". La lista de procesos en estado "Listo" está ordenada según la estrategia del algoritmo de planificación que utilice el sistema operativo, lo que permite gestionar eficientemente la asignación del tiempo de CPU a los procesos. En cambio, la lista de procesos en estado "Bloqueado" no sigue un orden específico, ya que no existe un criterio que determine en qué orden se van a desbloquear. Los procesos en esta lista se desbloquearán cuando ocurra el evento que están esperando.

El sistema operativo mantiene por cada proceso toda la información que necesita para su control mediante un descriptor de proceso o bloque de control de proceso PCB **(Process Control Block)**. El PCB es crucial para la gestión eficiente de procesos y recursos en un sistema operativo multitarea, proporcionando la información necesaria para suspender, reanudar y gestionar los procesos de manera efectiva.

El PCB de un proceso puede contener, entre otros, los siguiente datos:

1. Identificación del proceso: Un identificador único que distingue al proceso de otros en el sistema.
2. Estado del proceso: Indica si el proceso está en ejecución, listo, bloqueado u otro estado definido por el sistema operativo.
3. Contador de programa (Program Counter - PC): La dirección de la próxima instrucción que debe ejecutar el proceso.
4. Registros de CPU: Valores actuales de todos los registros del procesador asignados al proceso, incluyendo registros generales y de estado.
5. Información de planificación: Datos relacionados con la prioridad del proceso, su tiempo de ejecución restante, y otros parámetros utilizados por el planificador de procesos.
6. Espacio de memoria: Información sobre la ubicación y tamaño del espacio de memoria asignado al proceso, incluyendo la tabla de páginas si se usa paginación.
7. Información de recursos: Listas de archivos abiertos, dispositivos asignados y otros recursos asignados

al proceso.

8. Información de cuenta: Datos como el tiempo de CPU utilizado, tiempos de inicio y finalización, y otros datos de rendimiento que pueden ser útiles para la monitorización y la contabilidad del sistema.

2.3

Planificación Apropiativa y No Apropiativa

2.3.1 Planificación Apropiativa

Una disciplina de planificación del procesador se dice que es **apropiativa** cuando el sistema operativo puede interrumpir y reasignar la CPU de un proceso en ejecución a otro proceso. Los algoritmos que utilizan apropiatividad emplean diferentes técnicas para identificar la necesidad de realizar esta retirada forzosa del procesador, como es el uso del Quantum de tiempo o prioridades.

Entre las **ventajas de la planificación apropiativa**, podemos destacar:

- *Respuesta rápida:* Permite que los procesos de alta prioridad se ejecuten de forma más rápida e interactiva, mejorando la experiencia del usuario. Los procesos interactivos, que requieren respuestas rápidas, pueden ser atendidos más prontamente.
- *Equidad:* Todos los procesos tienen la oportunidad de ejecutarse, evitando que los procesos de baja prioridad se queden indefinidamente bloqueados por procesos de larga duración.
- *Mejor uso de la CPU:* Permite aprovechar al máximo el tiempo de ejecución de la CPU, ya que puede reasignarse a otro proceso cuando sea necesario.
- *Implementación de algoritmos complejos:* Permite la implementación de algoritmos de planificación más sofisticados que consideren diferentes factores como la prioridad, el tiempo de espera, el tiempo de ráfaga, etc.
- *Adecuada para sistemas multiusuario:* Es ideal para sistemas donde hay varios usuarios compitiendo por los recursos de la CPU.

Por su parte, los principales **inconvenientes de la planificación apropiativa** son:

- *Mayor complejidad:* La implementación de este tipo de planificación es más compleja que la planificación no apropiativa, ya que requiere mecanismos para interrumpir y reanudar procesos.
- *Overhead:*La gestión de las interrupciones y cambios de contexto puede generar un overhead adicional, lo que puede afectar al rendimiento general del sistema.

- *Inanición:* Si no se gestionan las prioridades correctamente, puede ocurrir que algunos procesos de baja prioridad nunca lleguen a ejecutarse, lo que se conoce como inanición.
- *Menos predecible:*El comportamiento del sistema puede ser menos predecible, ya que el tiempo de ejecución de cada proceso depende de la competencia con otros procesos.
- *No adecuada para sistemas de tiempo real:* No es adecuada para sistemas donde se requieren tiempos de respuesta deterministas, ya que la planificación apropiativa puede introducir retrasos impredecibles.

2.3.2 Planificación No Apropiativa

Una disciplina de planificación del procesador se dice que es **no apropiativa** cuando un proceso, una vez que obtiene el control de la CPU, lo mantiene hasta que finalice o ceda voluntariamente el control, por ejemplo, al entrar en estado de espera o bloqueo (como esperando una operación de E/S). En los algoritmos que siguen esta disciplina, el sistema operativo no interrumpe ni reasigna el CPU a otro proceso de manera forzada.

Entre las **ventajas de la planificación no apropiativa** podemos destacar:

- *Sencillez:* Es una estrategia de planificación más simple de implementar y gestionar, ya que no requiere mecanismos para interrumpir y reanudar procesos.
- *Menor overhead:* Al no haber interrupciones ni cambios de contexto, se reduce el overhead del sistema, lo que puede mejorar el rendimiento general.
- *Predictibilidad:* El comportamiento del sistema es más predecible, ya que el tiempo de ejecución de cada proceso está definido por su propia duración.
- *Adecuada para sistemas de tiempo real:* Es ideal para sistemas donde se requieren tiempos de respuesta deterministas, ya que la planificación no apropiativa garantiza que cada proceso se ejecute durante un tiempo determinado.
- *Evita la inanición:* Todos los procesos tienen la oportunidad de ejecutarse hasta su finalización, lo que evita que algunos procesos queden bloqueados indefinidamente.

Por su parte, los principales **inconvenientes de la planificación no apropiativa** son:

- *Menor respuesta:* Los procesos de alta prioridad pueden tener que esperar a que finalicen los procesos de baja prioridad.
- *Menos justicia:* Los procesos de baja prioridad pueden verse relegados a un segundo plano si hay procesos de larga duración en el sistema.
- *Rigidez:* No puede adaptarse fácilmente a cambios en la carga de trabajo del sistema o a procesos de alta prioridad que necesiten atención inmediata.

- *No adecuada para sistemas multiusuario:* No es ideal para sistemas donde hay varios usuarios compitiendo por los recursos de la CPU, ya que puede dar lugar a una mala distribución del tiempo de ejecución.

La elección entre la planificación apropiativa y la no apropiativa depende de las necesidades específicas del sistema y de las características de los procesos que se ejecutan en él.

La planificación apropiativa de la CPU ofrece una mayor respuesta, justicia y eficiencia en el uso de la CPU, pero también es más compleja y puede generar overhead e inanición. Es una buena opción para sistemas multiusuario donde la interacción con el usuario es importante, pero no es adecuada para sistemas de tiempo real.

Por su parte, la planificación no apropiativa ofrece simplicidad y bajo overhead a costa de una menor capacidad de respuesta y flexibilidad. Es adecuada para sistemas donde la carga de trabajo es predecible y donde la equidad y la respuesta no son críticos. Sin embargo, en entornos donde los procesos interactivos y las aplicaciones en tiempo real son comunes, la planificación apropiativa suele ser preferida debido a su capacidad de responder rápidamente a las necesidades cambiantes del sistema.

2.4 Prioridades

La prioridad es un calificador que se asigna a cada proceso para determinar la atención que va a recibir del sistema. El objetivo de la planificación por prioridades es optimizar el uso de la CPU y garantizar que los procesos más importantes se ejecuten primero, de modo que se cumplan los objetivos del sistema de la mejor manera posible. En el contexto de la planificación del procesador, y en el marco de las aplicaciones desarrolladas, se consideran estos tipos de prioridades:

- *Prioridad Estática:* Las prioridades son asignadas a los procesos en el momento de su creación y no cambian durante la vida del proceso. Se utilizan en sistemas donde se requiere una predictibilidad constante. Los procesos con mayor prioridad se ejecutan antes que los de menor prioridad. Es un método simple de implementar, pero puede ser injusto para los procesos de baja prioridad si hay procesos de larga duración en el sistema.
- *Prioridad Dinámica:* Las prioridades dinámicas pueden cambiar durante la vida del proceso, basándose en varios factores como el tiempo de espera, la edad del proceso, y el comportamiento del mismo. Permite una mayor justicia y eficiencia en el uso de la CPU, pero es más complejo de implementar que la prioridad fija.

Un tipo de prioridad dinámica es la prioridad por envejecimiento donde la prioridad de un proceso aumenta cuanto más tiempo ha estado esperando, de tal forma que previene la inanición de procesos de baja prioridad.

– *Prioridad de Usuario:* La prioridad la asigna el usuario o administrador del sistema, generalmente basándose en la importancia del proceso o la aplicación. Es útil para sistemas multiusuario donde los usuarios tienen diferentes necesidades de rendimiento, aunque puede caerse en el problema de hacer un mal uso o abuso, donde usuarios pueden asignar prioridades incorrectamente.

2.5 Quantum

El Quantum de tiempo es un intervalo fijo de tiempo asignado a cada proceso en un sistema de planificación apropiativa, durante el cual el proceso puede ejecutar antes de ser interrumpido y reasignado al CPU a otro proceso.

Se pueden considerar estos tipos de Quantum:

– *Quantum Fijo:* Es el tipo de Quantum más común. En este caso, todos los procesos reciben la misma cantidad de tiempo de CPU, independientemente de su tiempo de ejecución estimado o su importancia. Es un sistema sencillo de implementar que garantiza un principio básico de equidad sin embargo puede no ser óptimo para procesos con diferentes necesidades de tiempo de CPU, resultando en un uso ineficiente del CPU.

– *Quantum Variable:* En este caso, el Quantum de tiempo puede variar en función de ciertos criterios, como el tiempo de espera del proceso, el tiempo de ráfaga restante o la importancia del proceso. Este tipo de Quantum puede ser más eficiente que el Quantum de tiempo fijo, ya que permite dar más tiempo a los procesos que lo necesitan.

Es importante en este contexto reflexionar sobre el tamaño del Quantum, ya que un Quantum demasiado grande puede llevar a que los procesos interactivos experimenten retrasos, mientras que un Quantum demasiado pequeño puede causar un aumento significativo en el overhead debido a los frecuentes cambios de contexto.

III

Algoritmos de Planificación de la CPU

3.1

Algoritmos de Planificación

Existen numerosos algoritmos de planificación del procesador que se utilizan para gestionar la ejecución de procesos en un sistema operativo. Cada uno de estos algoritmos tiene sus propias ventajas y desventajas, y la elección del algoritmo adecuado depende de las necesidades específicas del sistema y de las características de las cargas de trabajo.

Para el propósito de este trabajo se recogen aquellos algoritmos y sus variantes que son trabajados dentro de la docencia de la asignatura de Sistemas Operativos I, dentro del Grado en Ingeniería Informática, y debe entenderse este listado dentro de dicho contexto.

3.1.1 FIFO: First Input, First Output o FCFS: First Come, First Served

Criterio

En este algoritmo, la lista de procesos en estado listo está ordenada según el orden de llegada al sistema, con el proceso más antiguo en la cabecera de dicha lista. Por lo tanto, cuando se crea un nuevo proceso, este se introduce al final de la lista de procesos en estado listo. El criterio para elegir el proceso que deberá pasar a estado activo es el de seleccionar el primero que está en dicha lista. Esta es una disciplina no apropiativa, lo que significa que al proceso que está activo no se le puede retirar la CPU hasta que él la abandone voluntariamente.

Inconvenientes

Los principales inconvenientes incluyen bajos rendimientos, ya que los procesos largos hacen esperar a los procesos cortos, lo que puede generar ineficiencias significativas. Además, no es útil en sistemas interactivos, ya que no garantiza buenos tiempos de respuesta, afectando negativamente la experiencia del usuario y la eficacia del sistema en entornos donde la rapidez es crucial.

Ventajas

Por su parte, las principales ventajas incluyen que permite predecir el orden de ejecución de los procesos, lo

que facilita la comprensión y el análisis del comportamiento del sistema. Además, es fácil de implementar, ya que su lógica de funcionamiento es sencilla y directa, lo que reduce la complejidad en el desarrollo y mantenimiento del sistema operativo.

Variantes

1. Considerando el tiempo que el proceso lleva en estado listo en vez del tiempo que lleva en el sistema, por lo que se selecciona, para asignarle el procesador, el proceso que ha estado más tiempo en la lista de procesos en estado listo.
2. Considerando que la disciplina es apropiativa, esto es, que a un proceso se le puede retirar la CPU y pasar de activo a listo si llega a la lista de procesos en estado listo un proceso más antiguo que él, al que se le asignará la CPU.

3.1.2 SJF: Shortest Job First

Criterio

Se trata de una disciplina no apropiativa. La lista de procesos en estado listo está ordenada por la cantidad de tiempo de ejecución estimada, de forma que el proceso más corto es el primero en la cola. Para establecer este orden, se necesita conocer el periodo de tiempo de ejecución de un proceso y, como esto raramente está disponible, se confía en las estimaciones de los usuarios.

Inconvenientes

Con respecto al algoritmo FIFO, los tiempos de respuesta son menos predecibles, especialmente para los procesos largos, lo que puede generar incertidumbre en la gestión del tiempo de ejecución. Además, este algoritmo es poco útil en sistemas interactivos, ya que no garantiza tiempos de respuesta razonables, afectando negativamente la eficiencia y la experiencia del usuario en dichos entornos.

Ventajas

Con respecto al algoritmo FIFO, reduce al mínimo el tiempo promedio de espera de los trabajos, ya que al favorecer los trabajos cortos sobre los largos reduce el número de trabajos en espera.

3.1.3 SRT: Shortest Remaining Time

Criterio

Esta disciplina elige siempre al proceso que le queda menos tiempo de ejecución estimado para completar su ejecución; de esta forma, aunque un proceso requiera mucho tiempo de ejecución, a medida que se va ejecutando iría avanzando en la lista de procesos en estado listo hasta llegar a ser el primero.

Es una disciplina apropiativa, ya que a un proceso activo se le puede retirar la CPU si llega a la lista de procesos en estado listo otro con un tiempo restante de ejecución estimado menor.

Inconvenientes

Su principal inconveniente es su alta sobrecarga debido a la necesidad de actualizar constantemente el tiempo restante de ejecución estimado de cada proceso en ejecución y en espera. Esta actualización frecuente del PCB introduce una carga adicional significativa en el sistema operativo, lo cual puede afectar el rendimiento global. Además, aunque SRT prioriza procesos con tiempos de ejecución más cortos, procesos largos podrían experimentar bloqueos si constantemente llegan procesos con tiempos restantes aún menores, lo que podría generar ineficiencias en la gestión de recursos y tiempos de respuesta impredecibles para algunos procesos.

Ventajas

Una ventaja significativa radica en su capacidad para ofrecer tiempos de respuesta muy cortos para procesos con tiempos de ejecución pequeños. Al priorizar constantemente el proceso con el menor tiempo restante de ejecución, SRT optimiza eficientemente la utilización del CPU, permitiendo que los procesos más cortos se ejecuten rápidamente. Esto mejora la capacidad de respuesta del sistema operativo en entornos donde la rapidez es crucial, como en sistemas interactivos o en aplicaciones que requieren respuestas rápidas a las entradas del usuario.

3.1.4 RR: Round Robin

Criterio

La lista de procesos en estado listo se organiza como una cola circular, lo que significa que el proceso al que se le asigna la CPU pasará a ser posteriormente el último de dicha cola. Cuando un proceso obtiene la CPU, puede utilizarla durante el tiempo máximo especificado por su Quantum. Si el proceso no agota su Quantum debido a que realiza una solicitud de evento y debe esperar (pasando a estado bloqueado), entonces la CPU se asigna al siguiente proceso en la cola circular (el proceso que ha estado más tiempo en estado listo), que pasa a estar activo sin esperar a que se complete el Quantum del proceso anterior. En caso de que un proceso agote completamente su Quantum, al pasar de activo a listo, se coloca al final de la lista de procesos en estado listo, lo que caracteriza esta estructura como una cola circular.

Inconvenientes

Sus principales inconvenientes son una potencial baja eficiencia en términos de utilización de la CPU, especialmente en entornos donde los procesos varían considerablemente en su requerimiento de tiempo de CPU. Debido a que cada proceso obtiene un Quantum fijo de tiempo de CPU antes de ser retirado para dar paso al siguiente proceso en la cola, los procesos cortos pueden experimentar una sobrecarga significativa de cambio de contexto en comparación con su tiempo de ejecución real, lo que puede afectar negativamente el rendimiento general del sistema. Además, en aplicaciones con procesos interactivos o con tiempos de respuesta críticos, el RR puede no garantizar tiempos de respuesta consistentemente rápidos, ya que los procesos pueden esperar hasta que se les asigne la CPU nuevamente al final de la cola circular, dependiendo del Quantum asignado y la carga del sistema.

Ventajas

Es un mecanismo con una implementación simple y justa en la asignación de tiempo de CPU entre los procesos. Al asignar a cada proceso un Quantum fijo de tiempo de CPU antes de pasar al siguiente proceso en la cola, RR asegura que todos los procesos obtengan oportunidades regulares de ejecución, lo que promueve una distribución equitativa de recursos. Esto es especialmente beneficioso en entornos multitarea donde múltiples usuarios o aplicaciones compiten por recursos de manera simultánea. Además, RR es efectivo en la prevención de inanición, ya que ningún proceso puede monopolizar la CPU indefinidamente, asegurando que todos los procesos tengan la oportunidad de ejecutarse dentro de un intervalo de tiempo predecible.

Variantes

Pueden darse implementaciones con Quantum fijo para cada proceso o variable, además de que el Quantum puede, o no, ser igual para todos los procesos.

3.1.5 Colas de Niveles Múltiples (Colas de prioridad múltiple)

Criterio

A cada proceso se le asigna una prioridad, y la lista de procesos en estado listo está formada por una serie de colas circulares, donde cada una de ellas contiene a todos aquellos procesos que tienen una misma prioridad. La CPU se asigna siempre al proceso de la cola de mayor prioridad que no este vacía. Este proceso puede agotar su Quantum o no, en ambos casos cuando vuelva a la lista de procesos en estado listo se colocará al final de la cola circular correspondiente a su prioridad.

Inconvenientes

Los principales inconvenientes radican en la complejidad añadida en la implementación y gestión del sistema operativo. A medida que se añaden más colas con diferentes prioridades y algoritmos de planificación, aumenta la complejidad del código del sistema operativo, lo que puede llevar a una mayor posibilidad de errores y dificultades en la depuración. Además, la asignación de prioridades puede ser subjetiva y requerir ajustes frecuentes para optimizar el rendimiento del sistema según las cargas de trabajo cambiantes.

Además, cabe destacar que existe la posibilidad de que los procesos de baja prioridad pueden quedar postergados indefinidamente. Una forma de solucionarlo es con la llamada *prioridad por envejecimiento*, que consiste en que la prioridad del proceso aumenta gradualmente a medida que éste pasa cierto tiempo sin que se le asigne la CPU, garantizando su terminación en un tiempo finito.

Ventajas

Este algoritmo permite gestionar eficazmente la prioridad y los recursos entre diferentes tipos de procesos, asegurando que los procesos críticos reciban atención preferencial mientras se evita la inanición de procesos de baja prioridad. Esta estructura optimiza el rendimiento del sistema operativo al aplicar algoritmos de planificación específicos a cada nivel de prioridad, adaptándose dinámicamente a las necesidades cambiantes del entorno. Además, proporcionan flexibilidad para ajustar las políticas de prioridad según las cargas de trabajo, mejorando así la equidad en la distribución de recursos y optimizando la eficiencia del sistema operativo en diversos escenarios.

Variantes

En términos generales, cada cola puede tener asignado un algoritmo de planificación diferente, adaptado a las necesidades y características de los procesos que la componen.

En los simuladores desarrollados, se han creado variantes de este algoritmo que proponen diferentes criterios a la hora de ordenar la lista de procesos en estado "Listo".

Tanto el algoritmo base como sus variantes pueden implementarse en una variante apropiativa y no apropiativa.

1. Colas de niveles múltiples [apropiativo/no apropiativo] 1. Los procesos en cada una de las colas se tratan de forma FIFO, por orden de llegada a la lista de procesos en estado “Listo”. La CPU se asigna al proceso que se encuentre en la cabecera de la cola de mayor prioridad.
2. Colas de niveles múltiples [apropiativo/no apropiativo] 2. Los procesos en cada una de las colas se ordenan por el tiempo estimado de ejecución inicial de cada proceso, de menor a mayor. La CPU se asigna al proceso que se encuentre en la cabecera de la cola de mayor prioridad.
3. Colas de niveles múltiples [apropiativo/no apropiativo] 3. Los procesos en cada una de las colas se

ordenan en función del tiempo que lleven en estado "Listo", de mayor a menor. La CPU se asigna al proceso que se encuentre en la cabecera de la cola de mayor prioridad, y por tanto, más tiempo en estado "Listo".

Es importante destacar que estas variantes deben entenderse dentro del contexto didáctico de la asignatura.

3.1.6 Colas de Retroalimentación de niveles múltiples

Criterio

Este algoritmo de planificación pretende tratar a los procesos de acuerdo a su comportamiento (utilización que hace el proceso del procesador), es decir:

- si el proceso se suspende frecuentemente se le debe asignar bastantes veces el procesador con un Quantum pequeño;
- si el proceso se suspende poco se le debe asignar el procesador menos veces pero con un Quantum grande.

Para conseguir esto la lista de procesos en estado listo esta formada por varios niveles y cada uno de ellos con una cola circular. Cuando se inserta un nuevo proceso este se añade al final de la cola de mayor nivel moviéndose por dicha cola en orden FIFO hasta obtener la CPU.

La CPU se asigna al proceso que se encuentra en la cabecera de la cola de mayor nivel que no este vacía. Cuando un proceso recibe la CPU puede ocurrir:

- que agote su Quantum: entonces al pasar a estado listo se coloca al final de la cola del siguiente nivel inferior al que pertenecía. Mientras que el proceso consuma totalmente su Quantum en cada nivel, irá moviéndose hacía el final de las colas de los niveles inferiores. La cola del nivel más profundo se gestiona mediante asignación de rueda, cuyos procesos permanecen hasta que terminan.
- que no agote su Quantum: entonces cuando pase de estado bloqueado a listo volverá al final de la cola del nivel al que pertenecía. Se supone que el comportamiento inmediato anterior de un proceso puede ser un buen indicador del comportamiento futuro cercano.

Normalmente, el Quantum otorgado al proceso aumenta a medida que el proceso se mueve hacía las colas de los niveles inferiores. De esta forma, cuanto más tiempo estén los procesos en la lista de procesos en estado listo, mayor será su Quantum cada vez que obtienen la CPU. En definitiva, este algoritmo trata a los procesos según su comportamiento, y los separa por categorías según sus necesidades de CPU.

Inconvenientes

El principal inconveniente es la complejidad incrementada en la implementación y gestión del sistema operativo. Esta complejidad se debe a la necesidad de mantener y ajustar múltiples colas con diferentes parámetros de planificación, como Quantum y prioridad, lo cual puede aumentar la carga administrativa y potencialmente introducir errores en el código del sistema operativo. Además, el rendimiento del sistema puede verse afectado si las políticas de planificación no están adecuadamente configuradas, lo que podría resultar en una asignación ineficiente de recursos y tiempos de respuesta inconsistentes para los procesos. Otra limitación es la posibilidad de que procesos de baja prioridad experimenten inanición si las políticas de retroalimentación no están equilibradas correctamente, lo que podría degradar la equidad en la distribución de recursos del sistema operativo.

Ventajas

El tiempo de respuesta percibido por los usuarios mejora y optimiza la utilización de la `CPU`, especialmente en entornos multitarea donde las cargas de trabajo varían. Además, las colas de retroalimentación de niveles múltiples son flexibles y adaptativas, permitiendo ajustar las políticas de planificación en tiempo real según las condiciones del sistema y las necesidades de los procesos, lo que contribuye a una mejor gestión de recursos y una mayor eficiencia operativa del sistema operativo.

Variantes

1. Permitir al proceso moverse hacía un nivel superior al anterior cada vez que abandona voluntariamente la `CPU` antes de agotar su Quantum. De esta forma, se responde a los cambios en el comportamiento de un proceso.
2. Mantener al proceso circulando en asignación de rueda varias veces a través de cada cola antes de pasarlo a la siguiente inferior. El número de ciclos en cada uno de los niveles se incrementa a medida que el proceso pasa a la cola del nivel siguiente inferior.

3.2

Ejercicios: Carga de Trabajo, Diagrama de Ocupación

A la hora de trabajar de forma práctica con los algoritmos de planificación los ejercicios se estructuran en distintos tipos, pero siempre entorno a la elaboración o validación de *Diagramas de Ocupación*, considerando una *Carga de Trabajo* dada.

Una carga de trabajo es una tabla que recoge información general de cada uno de los procesos, así como el

detalle de su comportamiento durante su ejecución que se refleja mediante una secuencia de ráfagas de CPU y Bloqueado. En concreto las columnas que se incluyen son las siguientes:

- *Proceso:* letra que identifica el proceso. Esta nomenclatura se usa por simplicidad en la realización de los ejercicios. Este identificador hace las funciones del PID del proceso.
- *Tiempo estimado de ejecución inicial:* tiempo que se estima que un proceso va a estar ejecutándose. Rara vez está disponible, por tanto se suele confiar en las estimaciones de los usuarios.
- *Tiempo de llegada al sistema:* instante de tiempo en el que el proceso llega al sistema. Por simplicidad en los ejercicios el primer proceso siempre llega en el instante 0.
- *Prioridad:* calificador asignado a un proceso que determina la atención que va a recibir del sistema. A menor valor, mayor prioridad, por ejemplo 1 es más prioritario que 2. Se utilizan prioridades estáticas.
- *Quantum:* es el tiempo máximo de estancia de un proceso en Activo. Se utilizan Quantum fijos.
- *Ráfagas de CPU-BLOQ:* representan respectivamente las unidades de tiempo de las ráfagas de CPU y bloqueo que necesita el proceso para completar su ejecución.

En la tabla 3.1 se muestra un ejemplo de una posible carga de trabajo.

Table 3.1: Ejemplo de Carga de Trabajo.

Proceso	**T. estimado**	**Llegada**	**Prioridad**	**Quantum**	CPU	**BLOQ**	CPU	**BLOQ**	CPU
C	20	20	1	5	5	5	10	15	5
B	80	15	1	10	15	15	10		
A	55	0	2	5	10	15	10	5	15

Por su parte el diagrama de ocupación de la CPU ayuda a representar la ocupación de un procesador por parte de los procesos que coexisten en el sistema. En el diagrama se emplea el eje X para representar el tiempo y el eje Y para especificar los procesos existentes en el sistema. A lo largo del tiempo que tardan en ejecutarse estos procesos se debe indicar el estado en que se encuentra cada uno de ellos mediante la nomenclatura acordada, en este caso, el ejemplo de la figura 3.1 representa un diagrama de ocupación tal y como se genera en el Simulador que se describe en la sección 5.

En este caso:

- El estado "Activo" se representa con A.
- El estado "Listo" se representa con L.
- El estado "Bloqueado" se representa con B.
- Una X en un instante de tiempo, indica que el proceso ha finalizado su ejecución.

Figure 3.1: Ejemplo de diagrama de ocupación de la CPU.

3.2.1 Generación automática de ejercicios con IA generativa

Para generar cargas de trabajo, se pueden emplear alguno de los siguientes prompts en alguna herramienta de inteligencia artificial generativa. Los ejemplos que se recogen en este apartado han sido creados utilizando ChatGPT [Ope24]:

1. "Genera una carga de trabajo para realizar ejercicios de diagramas de ocupación de la CPU. La carga de trabajo es una tabla que debe identificar a 3 procesos, identificados en la primera columna como A, B, C. La siguiente columna debe mostrar los instantes de llegada que pueden varias entre 0 y 30. La siguiente columna debe mostrar un tiempo estimado de ejecución para cada proceso y otra columna con la prioridad que debe variar entre 1 y 5. Además, para cada proceso de deben indicar en unidades de tiempo múltiplos de 5 entre 6 y 10 columnas que representen alternativamente ráfagas de CPU y de Bloqueo."
2. "Crea una carga de trabajo para ejercicios de diagramas de ocupación de la CPU. La carga debe ser una tabla que contenga 3 procesos, nombrados A, B, C, con columnas para los instantes de llegada (valores entre 0 y 30), tiempos estimados de ejecución, prioridades (entre 1 y 5), y columnas adicionales que muestren ráfagas de CPU y Bloqueo en unidades de 5."
3. "Genera una tabla de carga de trabajo para representar 3 procesos en un diagrama de ocupación de CPU. La tabla debe incluir: columna de identificación (A, B, C), columna de instantes de llegada (entre 0 y 30), columna de tiempos estimados de ejecución, y una columna de prioridad (de 1 a 5). Añade entre 6 y 10 columnas que alternen ráfagas de CPU y Bloqueo, cada una múltiplo de 5."
4. "Proporciona una carga de trabajo en formato de tabla para analizar la ocupación de la CPU con 3 procesos (A, B, C). Incluir columnas para los instantes de llegada (0-30), tiempos de ejecución, prioridades (1 a 5), y columnas adicionales representando ráfagas de CPU y Bloqueo, alternando, y en múltiplos de 5."

5. "Diseña una tabla de carga de trabajo para ejercicios de ocupación de CPU. La tabla debe tener 3 procesos (A, B, C), con instantes de llegada entre 0 y 30, tiempos de ejecución, prioridades entre 1 y 5, y entre 6 y 10 columnas que representen alternativamente ráfagas de CPU y Bloqueo, todas en múltiplos de 5."
6. "Genera una carga de trabajo para diagramas de ocupación de la CPU en forma de tabla. Debe incluir tres procesos (A, B, C), con columnas para tiempos de llegada (0-30), tiempos de ejecución estimados, prioridades (de 1 a 5) y de 6 a 10 columnas con ráfagas alternas de CPU y Bloqueo, en múltiplos de 5."
7. "Crea una tabla con una carga de trabajo para realizar diagramas de ocupación de CPU. La tabla debe tener 3 procesos identificados como A, B, C, con llegada entre 0 y 30, un tiempo de ejecución para cada proceso, prioridad entre 1 y 5, y columnas adicionales con ráfagas alternas de CPU y Bloqueo en múltiplos de 5."
8. "Proporciona una carga de trabajo en formato tabular con 3 procesos (A, B, C) para estudiar la ocupación de la CPU. Cada proceso tiene una llegada entre 0 y 30, tiempo estimado de ejecución, prioridad (1-5), y entre 6 y 10 columnas adicionales que muestran ráfagas alternas de CPU y Bloqueo, en múltiplos de 5."
9. "Diseña una carga de trabajo para la CPU en forma de tabla, con 3 procesos (A, B, C). Cada proceso tiene un instante de llegada (0 a 30), un tiempo de ejecución estimado, una prioridad (de 1 a 5), y entre 6 y 10 columnas adicionales representando ráfagas alternas de CPU y Bloqueo en múltiplos de 5."
10. "Elabora una tabla con la carga de trabajo de 3 procesos (A, B, C) para realizar ejercicios de diagramas de ocupación de la CPU. Debe incluir tiempos de llegada (0-30), tiempos de ejecución estimados, prioridades de 1 a 5, y varias columnas (6-10) que alternen ráfagas de CPU y Bloqueo, en múltiplos de 5."
11. "Genera una tabla de carga de trabajo para análisis de CPU. La tabla debe tener 3 procesos etiquetados como A, B, C, una columna para los instantes de llegada (entre 0 y 30), una para los tiempos estimados de ejecución, una para las prioridades (1 a 5), y entre 6 y 10 columnas que representen ráfagas alternas de CPU y Bloqueo en múltiplos de 5."

Se obtienen tablas de cargas de trabajo como las figura 3.2 y 3.3:

Proceso	Instante de llegada	Tiempo estimado de ejecución	Prioridad	Ráfaga 1	Bloqueo 1	Ráfaga 2	Bloqueo 2	Ráfaga 3	Bloqueo 3	Ráfaga 4	Bloqueo 4	Ráfaga 5
A	5	25	3	5	5	10	5	5	5	5	-	-
B	10	20	2	10	5	5	5	5	5	-	-	-
C	15	30	4	5	10	10	5	5	5	5	5	5

Table 3.2: Carga de trabajo para ejercicios de diagramas de ocupación de la CPU I

Proceso	Instante de llegada	Tiempo estimado de ejecución	Prioridad	Ráfaga 1	Bloqueo 1	Ráfaga 2	Bloqueo 2	Ráfaga 3	Bloqueo 3	Ráfaga 4	Bloqueo 4	Ráfaga 5
A	0	20	1	5	10	5	5	5	-	-	-	-
B	15	30	3	10	5	5	10	5	5	5	-	-
C	25	25	2	5	5	10	5	5	5	5	5	-

Table 3.3: Carga de trabajo para ejercicios de diagramas de ocupación de la CPU II

IV

Simulador 1: Generación del Diagrama de Ocupación

4.1

Objetivo

El objetivo de este simulador es ayudar a los estudiantes en el proceso de estudio de los algoritmos de planificación de procesos que se trabajan en asignatura de Sistemas Operativos. Para lograr este objetivo el simulador permite generar, a partir de una carga de trabajo una simulación instante a instante del algoritmo de planificación seleccionada.

Además, conforme avanza la simulación y se van modificando los estados de los procesos, se va mostrando de forma visual los cambio que se producen y se actualizan de forma dinámica, la lista de procesos en estado listo y bloqueado respectivamente.

Las funcionalidades más importantes del simulador son:

- *Descripción de la carga de trabajo*: se permite detallar el comportamiento de los procesos, especificando para cada uno de ellos, las ráfagas de `CPU`, las de bloqueado, el tiempo de llegada al sistema, etc. Por defecto, al conectarse se generará automáticamente se genera una carga de trabajo aleatoria y con ella un diagrama de ocupación para un algoritmo FIFO No Apropiativo, considerando como criterio de ordenación para la lista de procesos en estado listo el orden de llegada al sistema. Sin embargo, el usuario puede escoger modificar algún dato específico de esa carga de trabajo o introducir manualmente una completa, con toda la información.
- *Selección del algoritmo de planificación a simular*: se permite seleccionar el algoritmo de planificación a simular, junto con la configuración de los parámetros por los que se rige el algoritmo (Quantum, tiempo estimado, prioridad, etc.)
- *Evolución de la ejecución de los procesos*: se muestra gráficamente la transición de los procesos entre los estados y el contenido de las diversas listas (de procesos en estado listo, y en estado bloqueado), que utiliza el sistema operativo, al ir aplicando el algoritmo de planificación seleccionado.
- *Obtención del diagrama de ocupación de la `CPU` y otras métricas*: se permite obtener información asociada y derivada de la ejecución de los procesos según el algoritmo de planificación simulado.
- *Generación de un informe del diagrama de ocupación de la `CPU` y análisis*: se permite, una vez

completada una simulación, extraer un informe en PDF donde se recoja el resultado de la simulación, incluyendo tanto el diagrama de ocupación y las distintas métricas calculadas.

- *Insertar Algoritmo*: el simulador permite actualizar su repositorio de algoritmos implementados.

4.2

Arquitectura y Tecnologías

El proyecto desarrollado ha seguido una arquitectura `JSP modelo-2`. Esta arquitectura está caracterizada por dar a un único componente en el servidor, la responsabilidad de procesar la petición entrante, mientras que otro componente, generalmente un `JSP`, se encarga de generar la respuesta del cliente. De la misma forma, se sigue una estructura similar a la establecida en las arquitecturas basadas en plugins ya que se ha querido que pueda ser extendida por terceros.

Este modelo arquitectónico, presentado en la figura 4.1 consiste en el desarrollo de una aplicación según el patrón Modelo-Vista-Controlador, pero especificando que el controlador debe estar formado por un único `Servlet`, que centralice el control de todas las peticiones al sistema, y que basándose en la `URL` de la petición `HTTP` y en el estado actual del sistema, derive la gestión y control de la petición a una determinada acción de entre las registradas en la capa controlador. Será otro componente, generalmente un `JSP`, el que se encargue de enviar la respuesta al cliente.

Figure 4.1: Arquitectura JSP modelo-2

Es fundamental para el desarrollo del simulador, permitir a los usuarios que introduzcan sus propios algoritmos de planificación de la `CPU`. Es por esto mismo, por lo que se sigue una estructura similar a la establecida en las arquitecturas basadas en plugins, como se muestra en la figura 4.2. Implementar la aplicación bajo esta arquitectura, permite que pueda ser extendida por terceros, incluyendo un modo de registro automático por parte de los plugins en la aplicación, así como, un protocolo para el intercambio de datos con esos plugins. A pesar de que estos dependen de una serie de servicios proporcionados por la aplicación, su inclusión o modificación no supone ningún cambio en el funcionamiento de la misma.

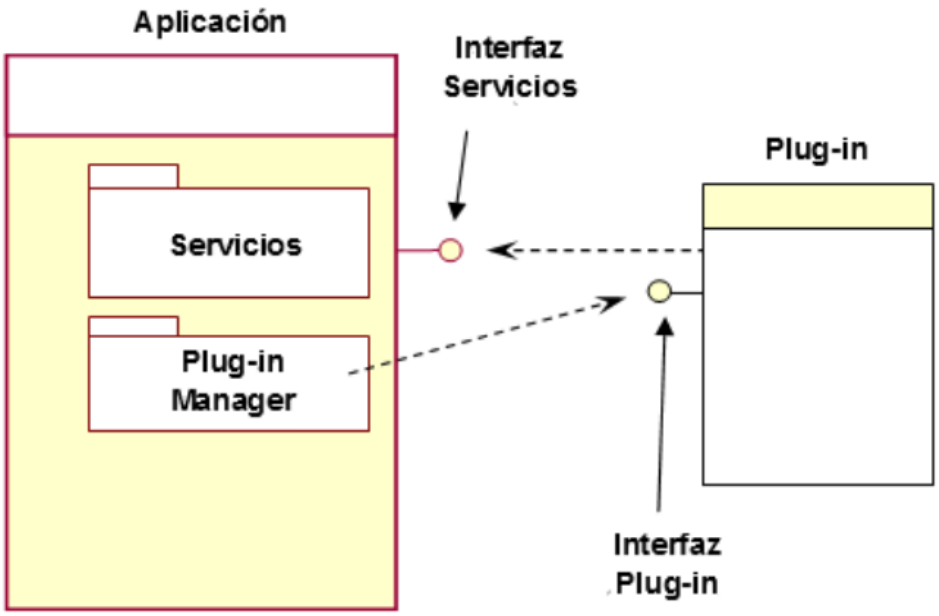


Figure 4.2: Arquitectura basada en plugins

A continuación se muestra la clasificación de las tecnologías mencionadas en **frontend** y **backend**, junto con una breve descripción del uso de cada una:

- **Frontend**
  - *HTML:* sirve como estructura general de la web. Se emplea `HTML5` y `CSS3`.
  - *JavaScript:* sirve para añadir dinamismo a la parte cliente e incluye algo de lógica. Se emplea las librerías `jQuery`, `Chart.js`, `Highcharts`, `BonsaiJS`, `Canvg`, `jsPDF`.
- **Backend:**
  - *Java:* En el contexto del backend, `Java` se utiliza para crear la lógica del servidor, proporcionando funcionalidades que interactúan con bases de datos y gestionan el funcionamiento del servidor.
  - *Servlets:* Se utilizan para manejar solicitudes del cliente, en particular los formularios que se muestran a los usuarios y generar respuestas dinámicas. Constituyen una parte fundamental en la creación de aplicaciones web basadas en `Java`.

4.3

Diseño

En la figura se muestra la 4.3 que permite expresar la estructura estática del sistema en función de sus clases, métodos, atributos y las diferentes relaciones que existen entre ellos. Esta "vista" define los servicios que el sistema proporciona a los usuarios, es decir, los requisitos funcionales del sistema.

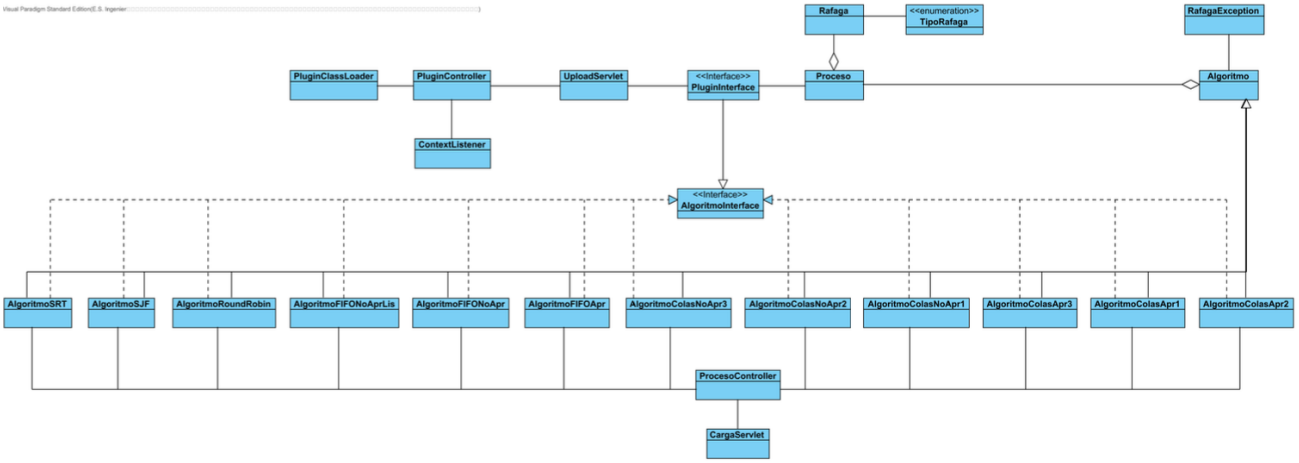


Figure 4.3: Diagrama de Clases

Dentro de las funcionalidades que se ofrecen, es destacable el hecho de que los usuarios pueden añadir sus propios algoritmos para simular. A continuación se exponen una serie de consideraciones que se deben tener en cuenta por los usuarios que quieran implementar el código con el que simular algoritmos de planificación e integrarlos en el sistema.

- *Los archivos deben estar comprimidos en formato `JAR`.* Estos ficheros permiten empaquetar varios ficheros en un solo archivo. Típicamente un fichero `JAR` contiene los ficheros de clases y los recursos auxiliares asociados con aplicaciones y `Applets`. Se ha elegido este formato porque facilita la posterior inclusión del algoritmo en la aplicación. Para que su funcionamiento sea correcto y puedan ser validados en el sistema, las clases de los algoritmos implementados no deben incluirse dentro de ningún paquete y los archivos `.class` generados deben incorporarse al archivo `JAR` que se incluye en el proyecto.
- *Se debe implementar la interfaz `PluginInterface` y extender la clase Algoritmo.* Para que el sistema verifique el algoritmo de planificación y lo incluya con el resto de algoritmos, éste tiene que implementar los métodos descritos en la interfaz `PluginInterface` y extender la clase `Algoritmo` haciendo uso al menos de los métodos `JavaToMap()`, `mapToJson()` y las variables `activo`, `listaEspera` y `listaBloqueado`. Concretamente, la interfaz `PluginInterface` describe los

métodos:

- `public String getPluginName()`, que devuelve el nombre del algoritmo introducido en el sistema.
- `public JSONObject ejecutar(List<Proceso> listaProcesos)` encargado de procesar el algoritmo bajo una lista de procesos que se le pasa como entrada. Este método devuelve el resultado de la ejecución del algoritmo en formato JSON.

A grandes rasgos, la implementación de un algoritmo de planificación sería la siguiente. Por cada unidad de tiempo (que debe empezar en 0 e incrementarse bajo múltiplos de 5) la implementación del algoritmo debe indicar el proceso que está en estado activo y los procesos que se encuentran en estado listo y estado bloqueado bajo las variables `activo, listaEspera y listaBloqueado`. Asimismo, debe llamar al método `JavaToMap()`, que obtiene los objetos Proceso que se encuentran en esas variables, y los convierte a objetos de tipo `Map`.

Una vez finalizada la ejecución del algoritmo debe invocarse el método `mapToJson()`, que transforma esos objetos `Map` en un objeto `JSON`, el cual permite generar los diagramas de ocupación de la `CPU` y de transición de estados. En la figura 4.4 se muestra un detalle de la estructura que debe tener la clase que implemente el nuevo algoritmo y los métodos a implementar.

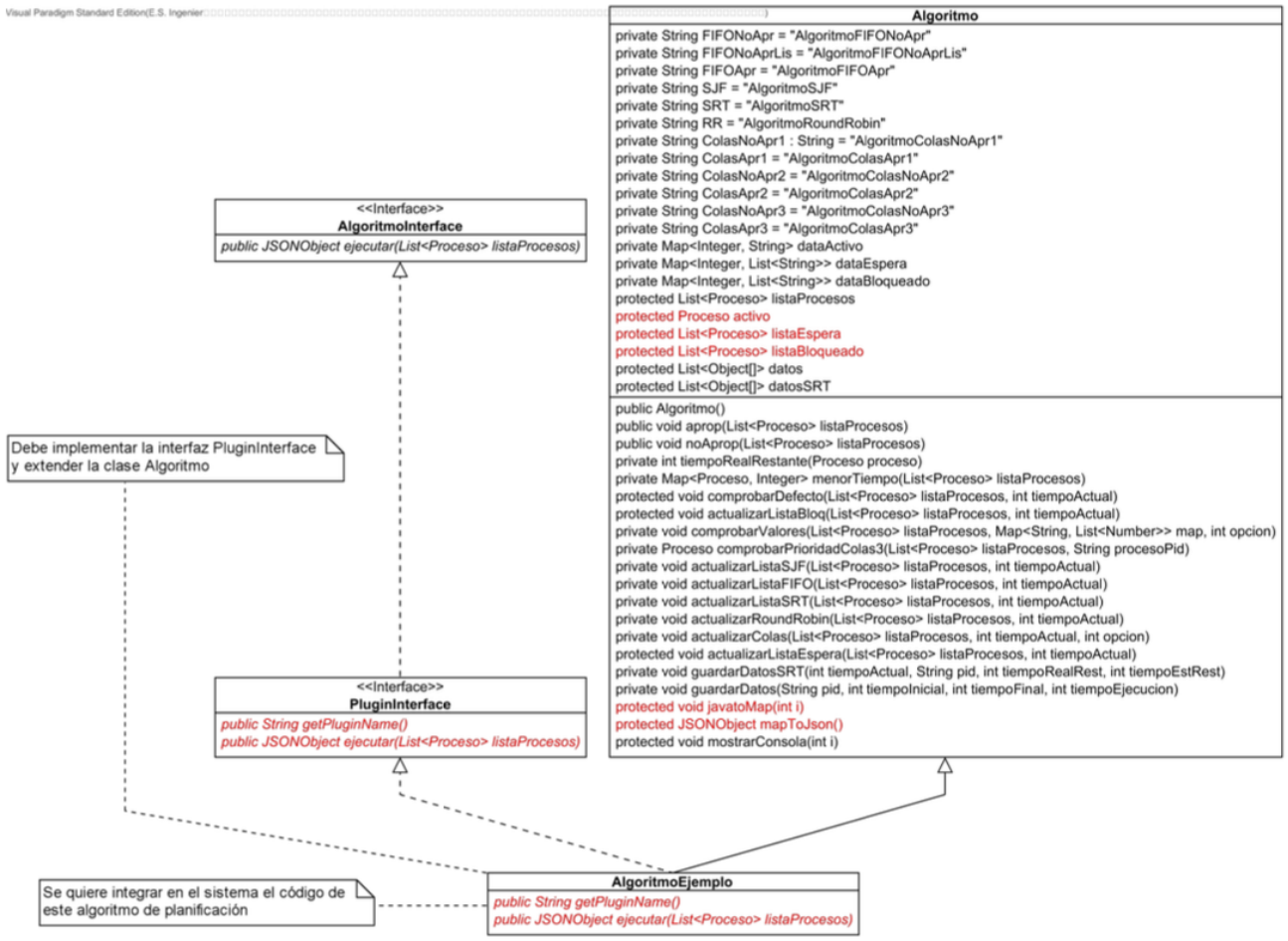


Figure 4.4: Diagrama de Clases. Inserción algoritmo de planificación

4.4

Uso del Simulador

El simulador está disponible en la siguiente dirección: `http://193.147.87.86:8080/proy_plan/`. Desde la ventana principal, se puede seleccionar mediante el menú superior alguna de las opciones que se vinculan con las funcionalidades descritas en el punto anterior y que se muestran en la figura 4.5.

Figure 4.5: Menú Superior

En caso de querer modificar la carga se despliega una ventana, como la de la figura 4.6 que permite añadir o borrar procesos, añadir o borrar ráfagas, ya sea de CPU o de Bloqueo, además de poder modificar los valores de Tiempo de Llegada y duración de las ráfagas actuales.

Figure 4.6: Ventana Modificar Carga Trabajo

En caso de escoger la opción de Nueva Carga de Trabajo, se muestra una ventana análoga a la anterior pero sin los valores ya precargados, sino que aparecen todos vacíos.

Desde la opción del menú de Algoritmo, se puede seleccionar aquel algoritmo que se desea simular. Para el propósito de este ejemplo, se simulará un SRT. En este caso, es preciso añadir a la simulación los datos del Tiempo Estimado de ejecución, para lo cual se despliega una ventana como la siguiente 4.7:

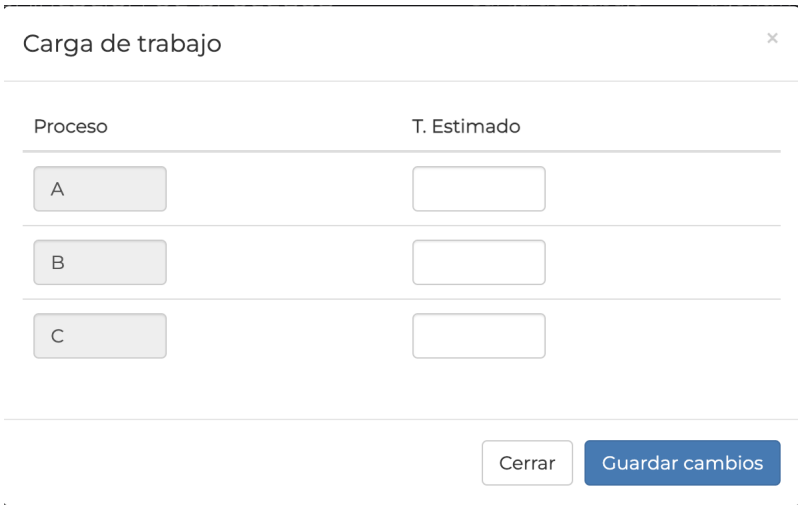


Figure 4.7: Introducción Datos.

En caso de algoritmos como RR, que precisa de concretar los valores del Quantum, o algoritmos de colas, que precisan de concretar los valores de prioridad, se desplegarán pantallas similares.

Una vez configurados los parámetros de la simulación, se puede comenzar pulsando en el botón *Comenzar Simulación*.

Desde ese momento, el diagrama se va generando en cada intervalo de tiempo de 5 unidades, mostrando los procesos que están activos en cada momento. Así mismo en el diagrama superior se mantiene actualizadas las listas de procesos en estado listo y bloqueado respectivamente. En las siguientes figuras 4.8 se muestran dos momentos de la simulación.

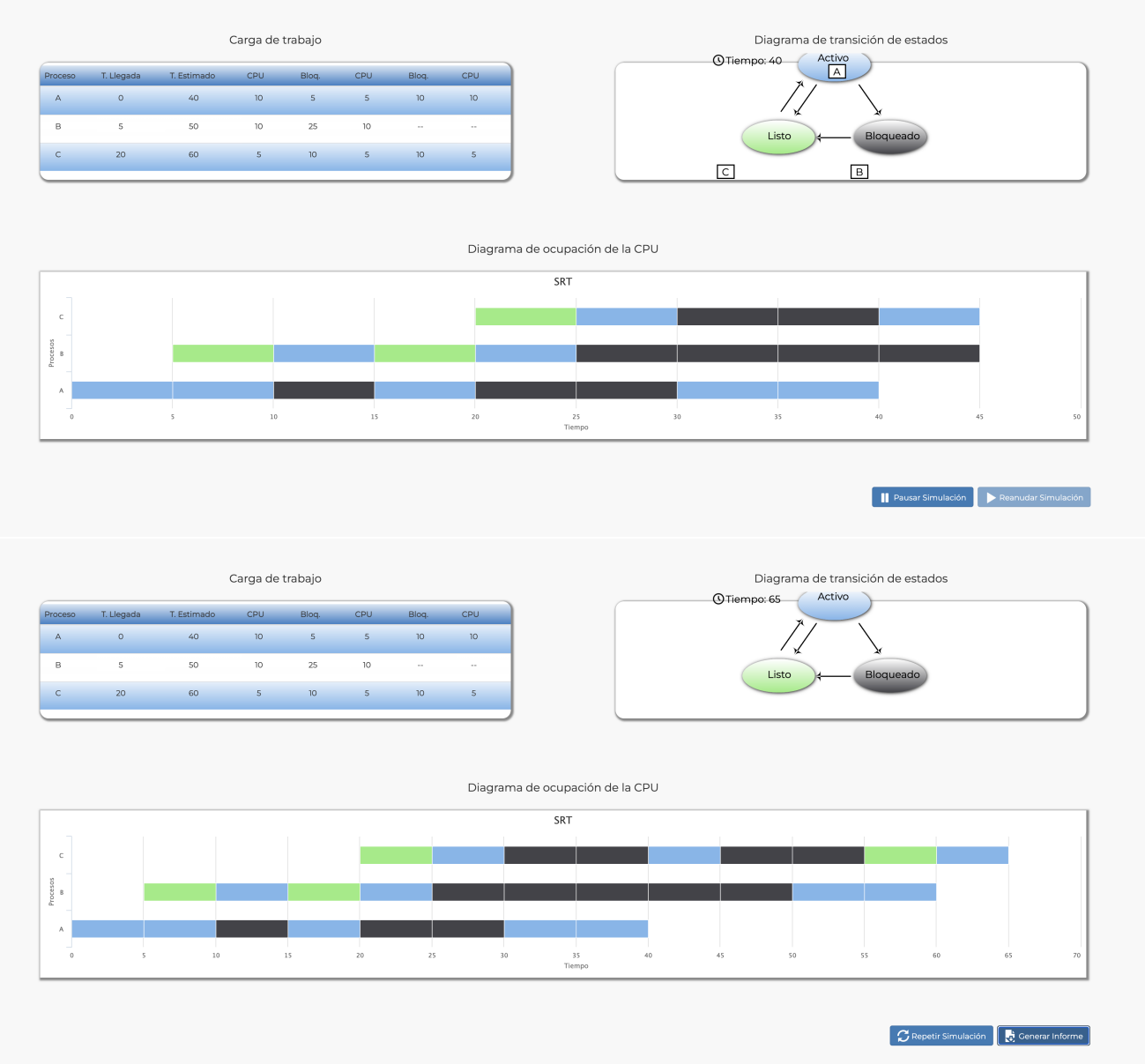


Figure 4.8: Desarrollo de la Simulación.

Una vez finalizada la simulación, se puede repetir o generar el informe de resumen de la misma.

4.4.0.1 Ejercicios con IA

De cara a que el estudiante pueda probar el funcionamiento de alguna característica específica, se pueden ejecutar alguno de los siguientes promtps que permitirían testear el funcionamiento de alguna característica concreta.

Conflictos de Prioridades

"Genera una carga de trabajo para realizar ejercicios de diagramas de ocupación de la CPU. La carga de trabajo es una tabla que debe identificar a 3 procesos, identificados en la primera columna como A, B, C. La siguiente columna debe mostrar los instantes de llegada que pueden varias entre 0 y 30. La siguiente columna debe mostrar un tiempo estimado de ejecución para cada proceso y otra columna con la prioridad que debe variar entre 1 y 5. Además, para cada proceso de deben indicar en unidades de tiempo múltiplos de 5 entre 6 y 10 columnas que representen alternativamente ráfagas de CPU y de Bloqueo. Esta carga de trabajo debe estar diseñada de tal forma que en un cierto momento al menos dos de los procesos que tengan la misma prioridad lleguen a estado listo a la vez."

Este prompt, genera una tabla como la de la siguiente 4.1 que puede utilizarse para que el estudiante aplique correctamente el segundo criterio de definición del algoritmo de planificación de colas que esté revisando. En concreto, en el ejemplo generado se garantiza que los procesos A y C, que tienen la misma prioridad, estarán en estado listo simultáneamente, proporcionando un buen ejemplo para analizar la planificación del CPU en estos casos.

Proceso	Instante de llegada	Tiempo estimado de ejecución	Prioridad	Ráfaga 1	Bloqueo 1	Ráfaga 2	Bloqueo 2	Ráfaga 3	Bloqueo 3	Ráfaga 4	Bloqueo 4	Ráfaga 5
A	0	30	2	10	5	10	5	5	-	-	-	-
B	10	20	3	5	10	5	5	5	5	-	-	-
C	15	25	2	5	5	10	5	5	5	5	-	-

Table 4.1: Conflictos de Prioridades

Llegada a Estado Listo en RR

"Genera una carga de trabajo para realizar ejercicios de diagramas de ocupación de la CPU. La carga de trabajo es una tabla que debe identificar a 3 procesos, identificados en la primera columna como A, B, C. La siguiente columna debe mostrar los instantes de llegada que pueden varias entre 0 y 30. La siguiente columna debe mostrar un tiempo estimado de ejecución para cada proceso y otra columna con la prioridad que debe variar entre 1 y 5. Además, para cada proceso de deben indicar en unidades de tiempo múltiplos de 5 entre 6 y 10 columnas que representen alternativamente ráfagas de CPU y de Bloqueo. Esta carga de trabajo debe estar diseñada especialmente para un algoritmo RR de tal forma que se den varios instantes de tiempo en los que se produzcan los siguientes conflictos: varios procesos llegan a estado listo a la vez, tras agotar el Quantum, abandonar el bloqueo y llegar nuevos al sistema."

Este prompt, genera una tabla como la de la siguiente 4.2 que puede utilizarse para que el estudiante aplique correctamente la regla NBA para identificar el orden adecuado en la inserción del procesos en la lista de procesos en estado listo.

En los ejemplos generados todos los procesos tienen ráfagas mayores que 5, por lo que agotarán el Quantum y volverán al estado listo. Además en ciertos instantes, como en el T=15 en el ejemplo que se muestra a continuación tanto B como C estarán listos (B llega tras agotar su primer Quantum, y C llega nuevo al sistema). Posteriormente, cuando los bloqueos finalicen, los procesos volverán a estar listos al mismo tiempo, creando conflictos.

Proceso	Instante de llegada	Tiempo estimado de ejecución	Prioridad	Ráfaga 1	Bloqueo 1	Ráfaga 2	Bloqueo 2	Ráfaga 3	Bloqueo 3	Ráfaga 4	Bloqueo 4	Ráfaga 5
A	0	30	2	10	5	10	5	5	5	5	-	-
B	10	25	3	5	5	10	5	5	5	5	5	-
C	15	20	1	5	10	5	5	5	5	-	-	-

Table 4.2: Carga de trabajo para ejercicios de diagramas de ocupación de la CPU (Algoritmo RR)

V

Simulador 2: Identificación del Algoritmo de Planificación

5.1

Objetivo

El objetivo de este simulador es ayudar a los estudiantes en el proceso de estudio y aprendizaje, a la vez que les permita evaluar sus conocimientos sobre un conjunto concreto de algoritmos de planificación de procesos. Para lograr este objetivo el simulador presenta dos formas distintas de trabajo para el estudiante. Por un lado, el simulador presenta como funcionan los distintos algoritmos de planificación de procesos y por otro, también ofrece la posibilidad de que se evalúen los conocimientos sobre estas estrategias de planificación de procesos. Cabe destacar que la información recogida en este capítulo también ha sido publicada en [Rod23].

Un punto destacado y muy importante del simulador desarrollado es que debe cumplir con esta doble funcionalidad, por un lado debe permitir al estudiante evaluar sus conocimientos sobre los algoritmos de planificación. En este caso se inserta o genera automáticamente un diagrama de ocupación y es el estudiante quien indica si un cierto algoritmo de planificación satisface dicho diagrama, indicando la causa de que no se satisfaga de ser el caso. Por otro lado, el simulador permite aprender, de tal forma que el va guiando con una serie de preguntas y opciones al estudiante para ayudarle a identificar las claves que permite determinar si el algoritmo satisface el diagrama y porqué no, de ser el caso.

Este doble comportamiento se representa en el diagrama de flujo de la figura 5.1 que recoge el funcionamiento general del simulador.

Las funcionalidades más importantes del simulador son:

- *Introducción de un diagrama de ocupación de la CPU.*
- *Obtención de un diagrama de ocupación de la CPU:* el simulador selecciona aleatoriamente un diagrama de ocupación de la CPU de una batería de diagramas que dispone.
- *Generación de un informe del diagrama de ocupación de la CPU y análisis:* un estudiante obtiene en un fichero PDF la carga de trabajo y el diagrama de ocupación de la CPU, así como el resultado de analizar si el diagrama de ocupación de la CPU cumple o no un algoritmo de planificación elegido.
- *Aprender sobre algoritmos de planificación de procesos:* un estudiante selecciona el algoritmo de planificación de procesos con el que quiera analizar un diagrama de ocupación de la CPU. Después, el

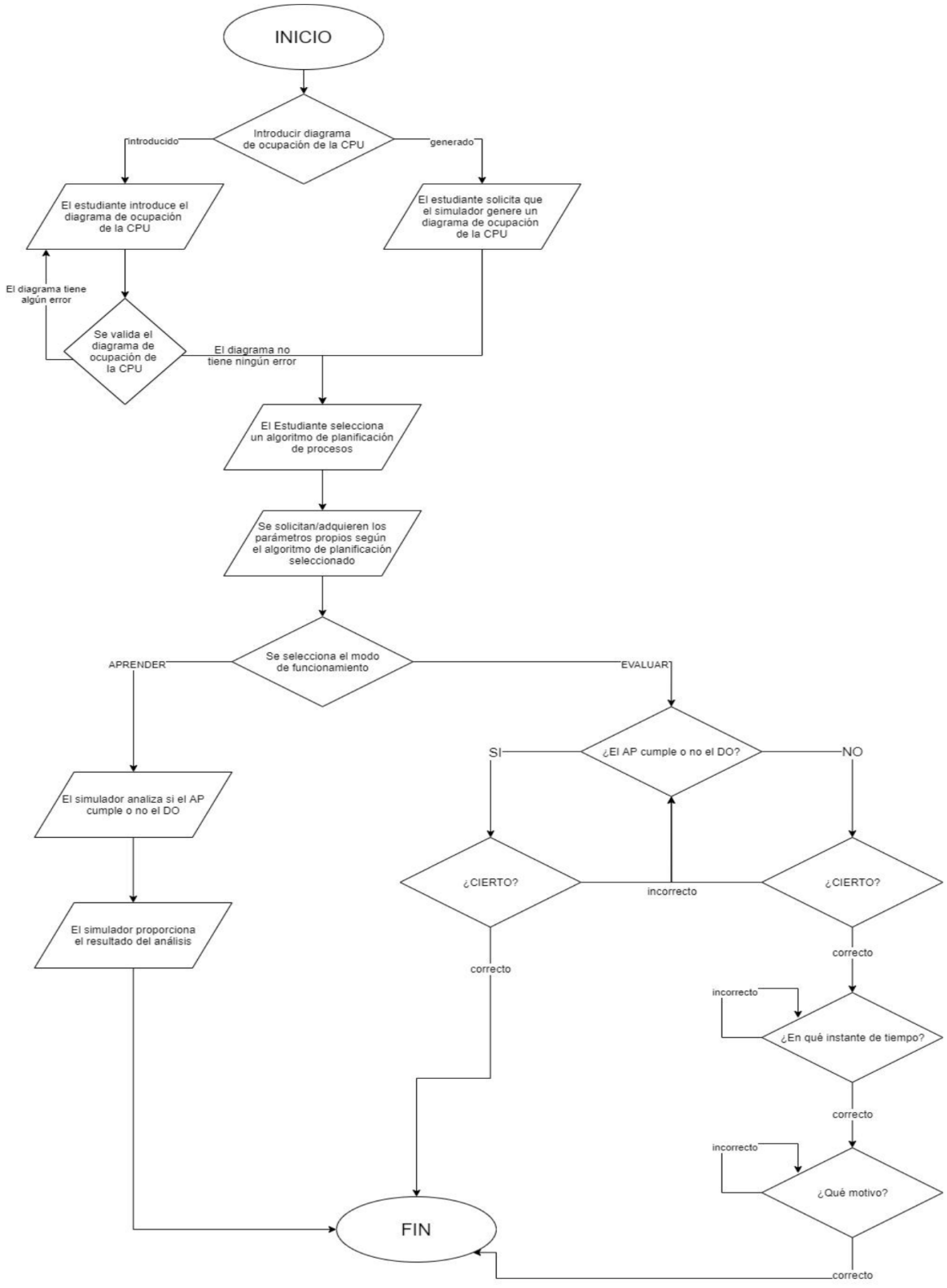


Figure 5.1: Doble Comportamiento.

simulador presenta al estudiante el resultado de comprobar si ese algoritmo de planificación de procesos cumple o no el diagrama de ocupación de la CPU. En caso de que no lo cumpla, tiene que indicar el primer instante en que deja de cumplirse razonando el motivo por el que no se cumple

- *Evaluar conocimientos sobre algoritmos de planificación de procesos:* un estudiante selecciona el algoritmo de planificación de procesos con el que quiera analizar un diagrama de ocupación de la CPU. Después, el simulador actúa como asistente para evaluarlo. El estudiante justifica si un algoritmo de planificación de procesos cumple o no un diagrama de ocupación de la CPU, y en caso de no cumplirlo indicar el instante a partir del cual no se cumple y el motivo.

5.2

Diseño

Como se comentaba previamente, el principal objetivo de este proyecto es el desarrollo de un simulador didáctico mediante el cual el estudiante pueda saber si el algoritmo de planificación de la CPU (sección 3) que ha elegido se corresponde con el diagrama de planificación de la CPU (sección 3.2) dado. Para este cometido, teniendo en cuenta los conocimientos anteriormente explicados, se han desarrollado dos **conjuntos de reglas.**

En concreto los conjuntos de reglas que ejecuta el simulador diseñado son:

1. **Reglas de Validación del Diagrama:** se encargan de corroborar que el diagrama de ocupación de la CPU introducido es correcto (sección 5.2.1).
2. **Reglas de Análisis del Algoritmo de Planificación:** se encargan de comprobar que el diagrama de ocupación de la CPU cumple o no el algoritmo de planificación de procesos seleccionado por el estudiante (sección 5.2.2).

Las reglas mencionadas se aplican en un orden específico. Primero, se debe verificar que el diagrama de ocupación de la CPU no contenga errores, aplicando el primer conjunto de reglas. Si se detecta un error, no se permite ejecutar el segundo conjunto de reglas hasta que este sea corregido.

5.2.1 Reglas de Validación del Diagrama

Este primer conjunto de reglas sirve para evaluar si el diagrama de ocupación de la CPU proporcionado por el estudiante presenta algún error. Los posibles errores en un diagrama de ocupación de la CPU incluyen, por

ejemplo, que un proceso cambie directamente de un estado "Listo" a un estado "Bloqueado", o que la CPU esté inactiva mientras exista un proceso en estado "Listo" durante el período de inactividad de la CPU.

Se describen a continuación las reglas de validación de este apartado.

Generales: En este apartado se validan aspectos generales del diagrama de ocupación de la CPU, que se concretan en las siguientes reglas.

1. Todos los procesos deben tener al menos un estado diferente a "Sin Estado" (N)[1].
2. En el sistema no pueden entrar dos procesos al mismo tiempo.
3. Todos los proceso deben estar, al menos una vez, en estado "Activo" (A).
4. La CPU no puede estar ociosa si existe algún proceso en estado "Listo" (L).

Solapamiento Estado Activos: En este apartado se validan aspectos que se desprenden de la condición del simulador que representa una máquina con una sola CPU y que se concretan en la siguiente regla.

5. No puede haber más de un proceso en estado “Activo” (A) al mismo tiempo.

Transiciones Incorrectas: En este apartado se validan aspectos sobre los estados válidos de los procesos, que se concretan en las siguientes reglas.

6. Todos los procesos deben empezar en estado “Activo” (A) o “Listo” (L).
7. No se pueden realizar las siguiente transiciones (de Estado A a Estado B).

Table 5.1: Transiciones prohibidas

Estado A	**Estado B**
Sin Estado	Bloqueado
Sin Estado	Fin
Listo	Sin Estado
Listo	Bloqueado
Listo	Fin
Activo	Sin Estado
Bloqueado	Sin Estado
Bloqueado	Fin
Fin	Listo
Fin	Activo
Fin	Bloqueado
Fin	Fin

[1]Este estado se utiliza con los diagramas que introduce el estudiante, mientras el diagrama aún está vacío antes de especificar los distintos estados del diagrama que se va a trabajar.

Finalizaciones Incorrectas: En este apartado se validan aspectos sobre la finalización de los procesos, que se concretan en las siguientes reglas.

8. Todos los procesos deben tener una única marca de "Fin" (X).
9. Todos los procesos deben terminar en estado "Activo" (A).

5.2.2 Reglas de Análisis del Algoritmo de Planificación

El conjunto de reglas de análisis del algoritmo de planificación ayuda a determinar si un diagrama de ocupación de la CPU se ajusta o no a un algoritmo de planificación de procesos seleccionado por el estudiante. En caso de ajustarse, el simulador solo tiene que indicar al estudiante que sí se cumple. En caso contrario, las reglas deben ayudar a determinar el instante de tiempo donde dejan de ajustarse y el motivo.

A continuación se describen las reglas específicas de cada uno de los algoritmos de planificación de procesos que tiene el simulador. Para cada algoritmo se indica:

- **Parámetros necesarios:** conjunto de parámetros que el simulador necesita para comprobar el algoritmo. Estos parámetros se encuentran en la tabla de carga de trabajo, explicados en la sección 3.2.
- **Política:** describe cómo el algoritmo de planificación de procesos organiza los procesos del sistema que están en la lista de procesos en estado "Listo".
- **Comprobación:** comprobaciones que se realizan para saber si un diagrama de ocupación de la CPU se ajusta o no a un algoritmo de planificación de procesos.

Cabe destacar que en el caso de los **algoritmos no apropiativos**, se añade una nueva regla al conjunto de análisis del algoritmo de planificación. La nueva regla comprueba que no se produce una transición del estado "Activo" al estado "Listo".

FIFO No Apropiativo

- **Parámetros necesarios:** tiempo de llegada al sistema que se deduce del diagrama de ocupación de la CPU.
- **Política:** la lista de procesos en estado "Listo" está ordenada en función del orden de llegada al sistema, siendo los primeros los que tienen un tiempo de llegada al sistema menor. Según se vayan creando los procesos, estos se van colocando al final de la lista.
- **Comprobación:** Para pasar un proceso A a "Activo", se comprueba que no existe otro proceso B, que en ese instante esté en estado "Listo", cuyo tiempo de llegada es anterior al de A.

SJF (Shortest Job First)

- **Parámetros necesarios:** tiempo estimado de ejecución inicial de cada proceso.
- **Política:** la lista de procesos en estado "Listo" está ordenada en función del tiempo estimado de ejecución inicial, siendo los primeros los que tienen un tiempo estimado de ejecución menor. La CPU se asigna al proceso con menor tiempo estimado de ejecución dentro de la cola de procesos en estado "Listo".
- **Comprobación:** Para pasar un proceso A a "Activo", se comprueba que no existe otro proceso B, que en ese instante de tiempo esté en estado "Listo", cuyo tiempo estimado de ejecución es menor.

Colas de Niveles Múltiples No Apropiativo

Este algoritmo tiene varias variantes, pero todas tienen en común una cola por cada una de las prioridades.

Colas de Niveles Múltiples No Apropiativo 1

- **Parámetros necesarios:** prioridad de cada proceso y el tiempo de llegada al sistema que se deduce del diagrama de ocupación de la CPU.
- **Política:** Los procesos en cada una de las colas se tratan de forma FIFO, por orden de llegada a la lista de procesos en estado "Listo". La CPU se asigna al proceso que se encuentra en la cabecera de la cola de mayor prioridad.
- **Comprobación:** Para pasar un proceso A a "Activo", se comprueba que no existe otro proceso B, que en ese instante de tiempo esté en estado "Listo", cuya prioridad es mayor, estando por delante en su cola.

Colas de Niveles Múltiples No Apropiativo 2

- **Parámetros necesarios:** prioridad y tiempo estimado de ejecución inicial de cada proceso.
- **Política:** Los procesos en cada una de las colas se ordenan por el tiempo estimado de ejecución inicial de cada proceso, de menor a mayor. La CPU se asigna al proceso que se encuentra en la cabecera de la cola de mayor prioridad.
- **Comprobación:** Para pasar un proceso A a "Activo", se comprueba que no existe otro proceso B, que en ese instante de tiempo esté en estado "Listo", cuya prioridad es mayor. En el caso de que ambos procesos tengan la misma prioridad y esta sea la mayor entre los procesos en estado "Listo", se escoge el proceso con menor tiempo estimado de ejecución inicial.

Colas de Niveles Múltiples No Apropiativo 3

- **Parámetros necesarios:** prioridad de cada proceso.
- **Política:** Los procesos en cada una de las colas se ordenan en función del tiempo que lleven en estado "Listo", de mayor a menor. La CPU se asigna al proceso que se encuentra en la cabecera de la cola de mayor prioridad, y por tanto, más tiempo en estado "Listo".
- **Comprobación:** Para pasar un proceso *A* a "Activo", se comprueba que no existe otro proceso *B*, que en ese instante de tiempo esté en estado "Listo", cuya prioridad es mayor. En el caso de que ambos procesos tengan la misma prioridad, se escoge al proceso que lleva más tiempo en estado "Listo", calculado desde la última vez que el proceso se incorporó a la lista de procesos en estado "Listo".

FIFO Apropiativo

- **Parámetros necesarios:** tiempo de llegada al sistema que se deduce del diagrama de ocupación de la CPU.
- **Política:** La lista de procesos en estado "Listo" está ordenada en función del orden de llegada al sistema, siendo los primeros los que tienen un tiempo de llegada menor. A medida que los procesos llegan al sistema, se colocan al final de la lista.
- **Comprobación:** Cuando un proceso *A* está en estado "Activo", se comprueba que no llegue otro proceso a estado "Listo" cuyo tiempo de llegada al sistema sea menor que el de *A*. Si llega tal proceso, se le retira la CPU al proceso *A* y se asigna al nuevo proceso.

SRT (Shortest Remaining Time)

- **Parámetros necesarios:** tiempo estimado de ejecución inicial de cada proceso y tiempo real de ejecución inicial de cada proceso.
- **Política:** La lista de procesos en estado "Listo" se ordena en función del tiempo estimado restante de ejecución, siendo los primeros los que tienen un tiempo estimado restante menor.
- **Comprobación:** Cuando un proceso *A* está en estado "Activo", se comprueba que no llegue otro proceso a estado "Listo" cuyo tiempo estimado de ejecución restante sea menor. Si llega un proceso con un tiempo estimado menor, se le asigna la CPU.

Colas de Niveles Múltiples Apropiativo

Este algoritmo tiene varias variantes, pero todas tienen en común una cola por cada prioridad.

Colas de Niveles Múltiples Apropiativo 1

- **Parámetros necesarios:** prioridad de cada proceso y tiempo de llegada al sistema que se deduce del diagrama de ocupación de la CPU.
- **Política:** Los procesos en cada cola se manejan de forma FIFO, por orden de llegada a la lista de procesos en estado "Listo". La CPU se asigna al proceso que se encuentra en la cabecera de la cola de mayor prioridad.
- **Comprobación:** Cuando un proceso *A* está en estado "Activo", se comprueba que no llegue otro proceso a estado "Listo" con mayor prioridad. En caso de que ambos procesos tengan la misma prioridad, se asigna la CPU al que haya llegado antes al sistema.

Colas de Niveles Múltiples Apropiativo 2

- **Parámetros necesarios:** prioridad y tiempo estimado de ejecución inicial de cada proceso.
- **Política:** Los procesos en cada cola se ordenan por el tiempo estimado de ejecución inicial, de menor a mayor. La CPU se asigna al proceso que se encuentra en la cabecera de la cola de mayor prioridad.
- **Comprobación:** Cuando un proceso *A* está en estado "Activo", se comprueba que no llegue otro proceso a estado "Listo" con mayor prioridad. Si los procesos tienen la misma prioridad, se asigna la CPU al proceso con menor tiempo estimado de ejecución inicial. Si ambos tienen el mismo tiempo y prioridad, se asigna al que llegó primero al sistema.

Colas de Niveles Múltiples Apropiativo 3

- **Parámetros necesarios:** prioridad de cada proceso.
- **Política:** Los procesos en cada cola se ordenan en función del tiempo que llevan en estado "Listo", de mayor a menor. La CPU se asigna al proceso en la cabecera de la cola de mayor prioridad.
- **Comprobación:** Cuando un proceso *A* está en estado "Activo", se comprueba que no llegue otro proceso a estado "Listo" con mayor prioridad. Si los procesos tienen la misma prioridad, se asigna la CPU al que ha estado más tiempo en estado "Listo".

RR (Round Robin)

- **Parámetros necesarios:** Quantum de cada proceso y tiempo de llegada al sistema que se deduce del diagrama de ocupación de la CPU.
- **Política:** La lista de procesos en estado "Listo" está ordenada en función del instante en el que los

procesos se incorporan a dicha lista. El objetivo es asignar la CPU al proceso que lleva más tiempo sin usarla. Los procesos ingresan a la lista de procesos en estado listo, cuando agotan la ejecución de su Quantum, cuando salen de un estado "Bloqueado" o cuando llegan nuevos al sistema. Si en un instante dado, sólo un proceso ingresa a esa lista, este se colocaría al final de la misma y esperaría hasta ser el primero de dicha cola para poder ejecutarse. En el caso de que varias de estas circunstancias puedan coincidir, y, en un mismo instante haya dos o más, procesos que ingresen a la lista de procesos en estado listo, estos deben ordenarse correctamente para cumplir el criterio del algoritmo. Para determinar en qué orden se deben añadir los procesos a la lista de procesos en estado "Listo", nos podemos ayudar de la regla NBA. Es importante destacar, que esta regla sólo es de aplicación cuando en el mismo instante llega más de un proceso a dicha lista. En caso de este conflicto al ingreso a la lista, los procesos se ordenaría de la siguiente forma:

  - **N**: Primero los procesos nuevos, acaban de llegar al sistema y no han usado nunca la CPU.
  - **B**:Luego los procesos que han terminado de estar bloqueados, usaron ya la CPU pero no recientemente.
  - **A**:Y por último, los procesos que completan su Quantum, y por lo que acaban de hacer uso de la CPU.
- **Comprobación:** Cuando un proceso *A* está en estado "Activo" y completa su Quantum, se comprueba que no llegue otro proceso a estado "Listo". Si hay más de un proceso en estado "Listo", se selecciona el que lleva más tiempo en la cola de procesos en estado "Listo". En caso de conflicto, se usa la regla `NBA (Nuevos, Bloqueados, Activos)` para determinar el orden.

5.3

Arquitectura y Tecnologías

La arquitectura de este simulador se basa en el modelo cliente-servidor [Som05], el cual está compuesto por dos elementos principales:

- **Cliente:** actúa como el solicitante de recursos. Su función es invocar los servicios proporcionados por el servidor, siguiendo un protocolo de solicitud-respuesta. Los clientes envían solicitudes al servidor y esperan su respuesta.
- **Servidor:** es el encargado de proveer los recursos. Procesa las solicitudes recibidas y retorna los resultados al cliente.

Se ha optado por esta arquitectura en este trabajo, ya que permite trasladar el procesamiento al servidor,

evitando la necesidad de contar con equipos personales de gran capacidad para cada estudiante. Esto hace que el dispositivo utilizado por el estudiante no requiera un alto rendimiento. Además, la arquitectura cliente-servidor facilita la implementación del patrón arquitectónico MVC (modelo-vista-controlador).

El patrón MVC, tal y como se representa en la figura 5.2, es un patrón arquitectónico de software cuya principal característica es la separación de la lógica de negocio y del módulo encargado de gestionar los eventos.

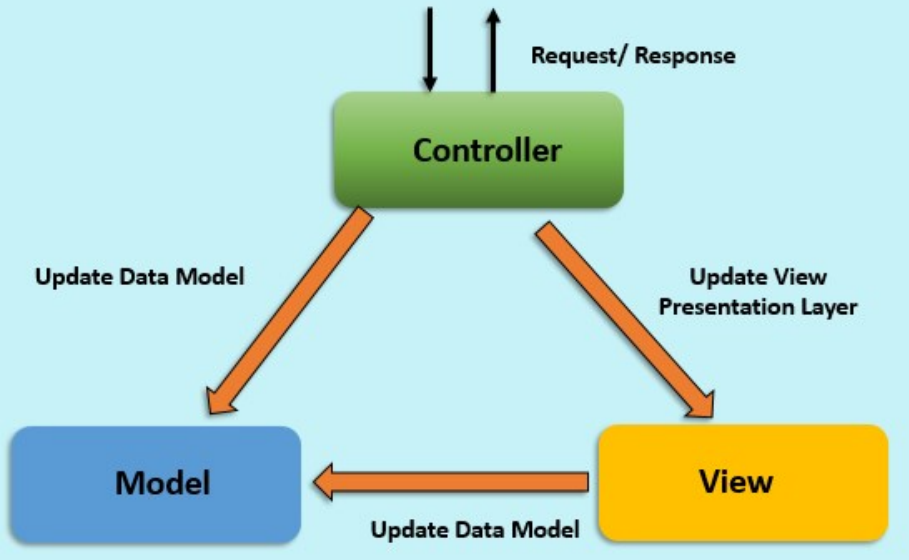


Figure 5.2: Patrón MVC. Fuente: EDUCBA

En particular, las responsabilidades de los distintos componentes de este patrón son las siguientes.

- **Modelo:** gestiona todo el contenido y la lógica de procesamiento específica de la aplicación, encargándose de trabajar con los datos. En este proyecto, se dispondrá de un archivo de persistencia que contendrá la información necesaria para la generación de los diagramas de ocupación de la CPU.
- **Vista:** se encarga de todas las funciones relativas a la interfaz, permitiendo la visualización del contenido.
- **Controlador:** implementa el código necesario para responder a las acciones solicitadas por la aplicación, actuando como intermediario entre el modelo y la vista.

Por otro lado, además de la arquitectura y patrón arquitectónico seguidos, cabe destacar el esquema de la aplicación implementada, que se puede mostrar en la figura 5.3.

En concreto las tecnologías utilizadas son las siguientes:

- **Frontend**
  - *HTML:* sirve como estructura general de la web. Se emplea HTML5 y CSS3.

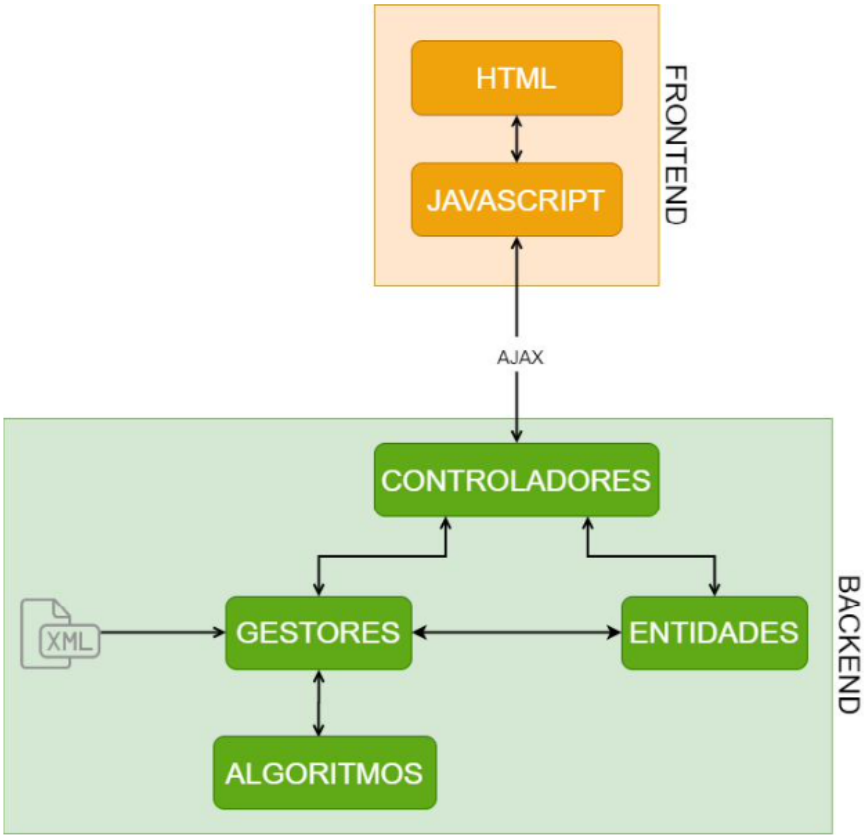


Figure 5.3: Esquema de la arquitectura de la aplicación.

  - *`JavaScript`:* sirve para añadir dinamismo a la parte cliente e incluye algo de lógica. Se emplea las librerías `jQuery` y `jspDF v2.1.1`.
- **Backend:** Se implementa en `Java`, y se utiliza `JSP`, `JavaServerPages`, y `JSTL(Standard Tag Library)`. Entrando en detalle de los componentes descritos en la figura 5.3.
  - *Controladores:* recibe los eventos de la parte del cliente, y aplica la lógica necesaria para responder a sus peticiones.
  - *Gestores:* oculta a los controladores complejidad, como son la comprobación de algoritmos y la lectura de ficheros.
  - *Algoritmos:* contienen toda la lógica necesaria de los algoritmos de planificación de procesos.
  - *Entidades:* representan a las entidades necesarias para el simulador, como son el diagrama de ocupación de la `CPU`, los procesos o las ráfagas.
  - *Fichero `XML`:* contiene los diagramas de ocupación de la `CPU` que el gestor de diagrama recuperará cuando se lo pida el controlador.

La comunicación entre el cliente y el servidor se lleva a cabo mediante llamadas `AJAX`.

5.4

Uso del Simulador

El simulador está disponible en la siguiente dirección `http://193.147.87.86:8080/simuladorDidacticoPlanificacionProcesos/`. Desde la ventana principal, se puede seleccionar generar un diagrama automáticamente o por el contrario introducir uno específico. Para el propósito de este ejemplo vamos a seleccionar la opción "Generar Diagrama".

En el primer momento se muestra una ventana, que contiene el diagrama de ocupación generado en la parte superior, y un resumen de la información mostrada en dicho diagrama en la parte inferior, tal y como se pueden ver en la figura 5.4.

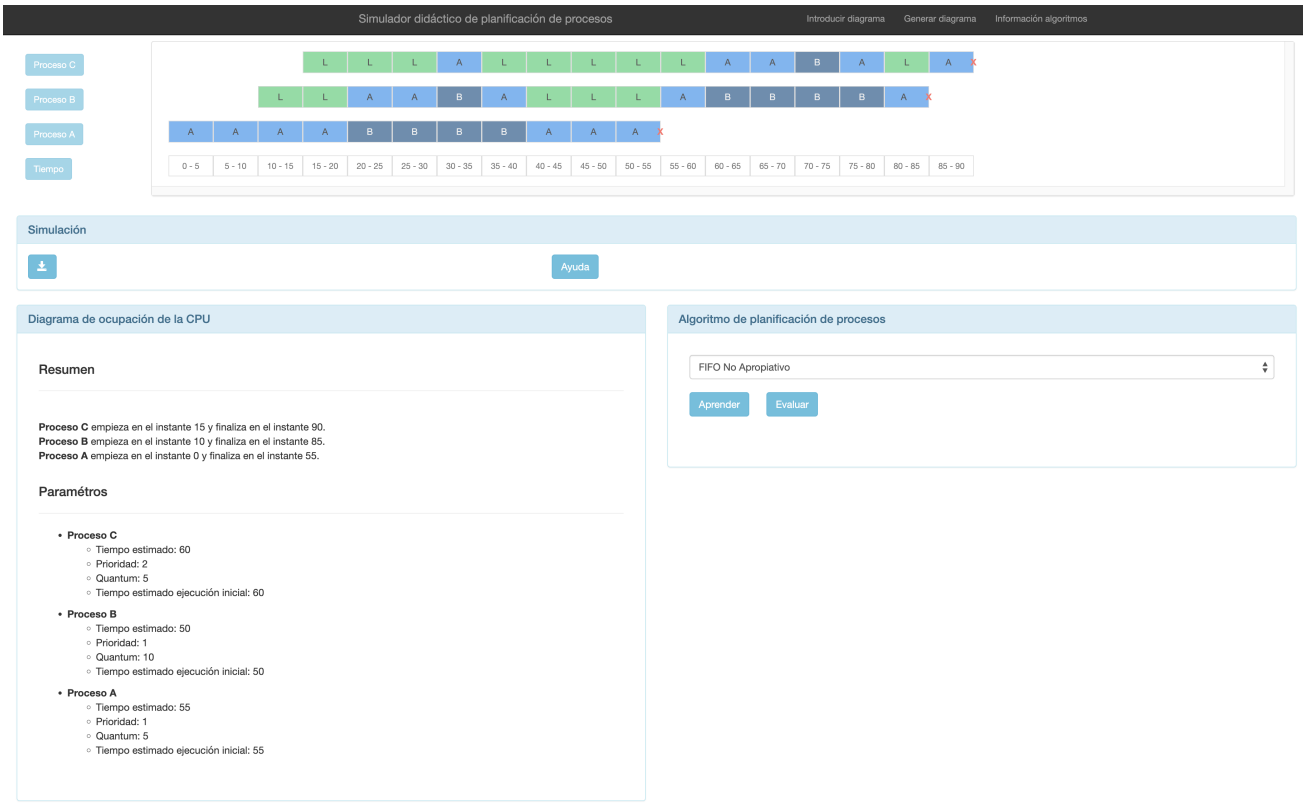


Figure 5.4: Diagrama de Ocupación Generado Automáticamente.

En la derecha se puede seleccionar el algoritmo de planificación de procesos y uno de los dos modos de funcionamiento del simulador *Aprender* o *Evaluar*, con el propósito de atender a la doble funcionalidad, tal y como se describe en la sección 5.1.

Independientemente de la opción seleccionada, se puede cambiar el algoritmo que se quiere validar, que por defecto es FIFO No Apropativo. Las opciones disponibles se muestran en la figura 5.5.

Figure 5.5: Algoritmos disponibles.

Modo Aprender

Para continuar con el ejemplo, se selecciona el RR (Round Robin) y se escoge la opción de *Aprender*. En este caso, el diagrama de ocupación generado no satisface el algoritmo seleccionado y el simulador informa de ello, indicando el instante en que se deja de satisfacer y el motivo, tal y como se muestra en la figura 5.6.

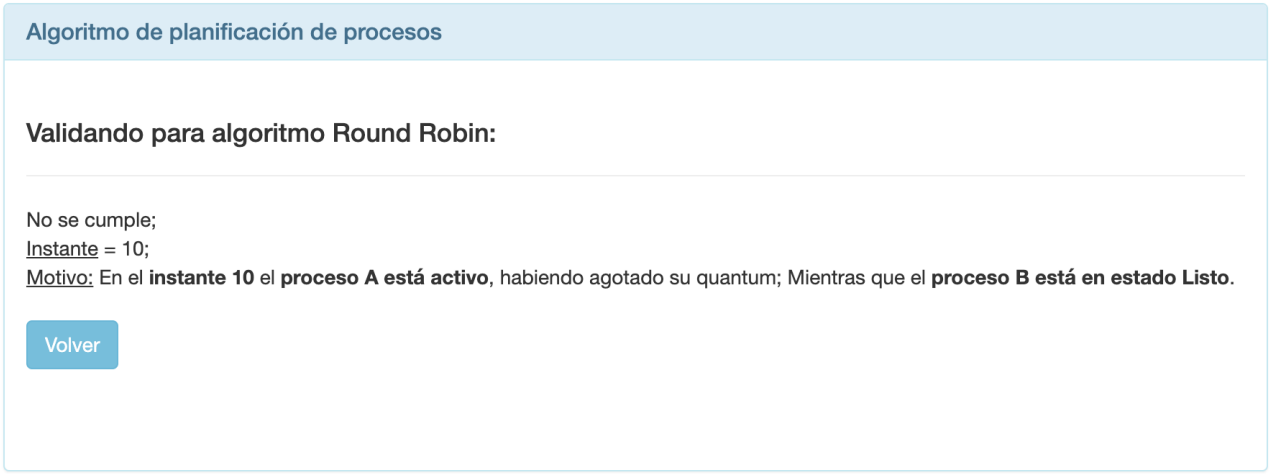


Figure 5.6: Modo Aprender I

Partiendo del mismo diagrama de ocupación generado, es posible comprobar otros algoritmos. Si se prueba por ejemplo el algoritmo SRT (Shortest Remaining Time), el simulador informa de que dicho diagrama satisface ese algoritmo, tal y como se muestra en la figura 5.7.

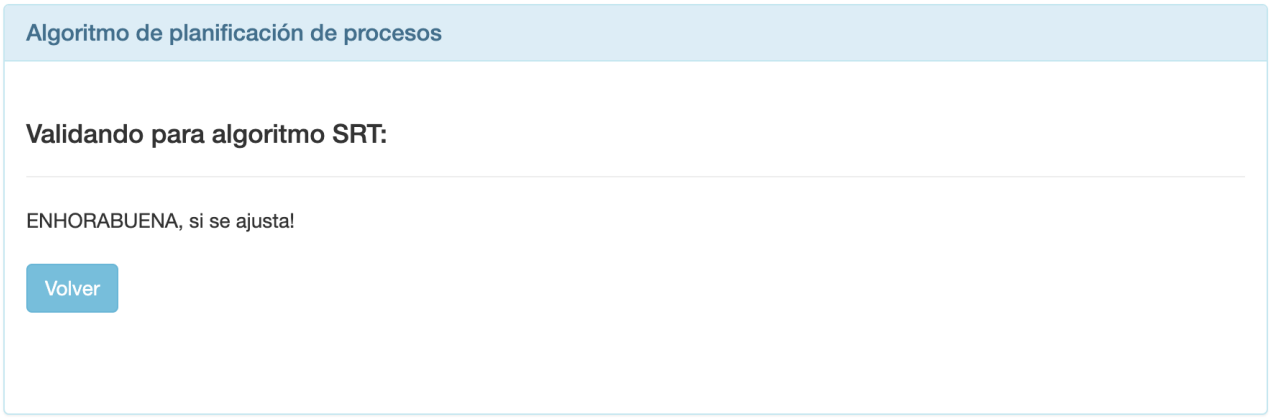


Figure 5.7: Modo Aprender II

Modo Evaluar

Si seleccionásemos la opción de *Evaluar* para un algoritmo, el simulador va guiando al estudiante con una serie de preguntas, para ayudarle a determinar si el diagrama se ajusta o no al comportamiento esperado del algoritmo seleccionado.

Por ejemplo, si nuevamente se seleccionase el algoritmo RR en modo evaluar, se le muestra al estudiante la pantalla de la figura 5.8:

Si el usuario escoge una opción incorrecta indicando que un algoritmo si satisface un diagrama cuando no lo hace, se informa del error, pero sin dar la respuesta válida, como se recoge en la imagen 5.9.

Si por el contrario, el usuario indica correctamente que el algoritmo no satisface un diagrama, se le pregunta por el instante en que se produce el error y la causa del mismo, como se presenta en la figura 5.10.

El simulador acompaña al estudiante con distintas preguntas, e informando de los errores que comete hasta que identifica correctamente el instante en el que se produce el error. Llegados a ese punto se le plantean una serie de alternativas para que indique el motivo por el cual el diagrama no es correcto, tal y como se recoge en la figura 5.11.

Una vez el usuario identifica la opción correcta, se informa del acierto, como se muestra en la figura 5.12.

El manual de usuario completo del funcionamiento del simulador, se encuentra disponible en la página principal del mismo, así como la explicación de los modos de funcionamiento y una breve documentación técnica sobre los algoritmos que se simulan y las validaciones que se realizan.

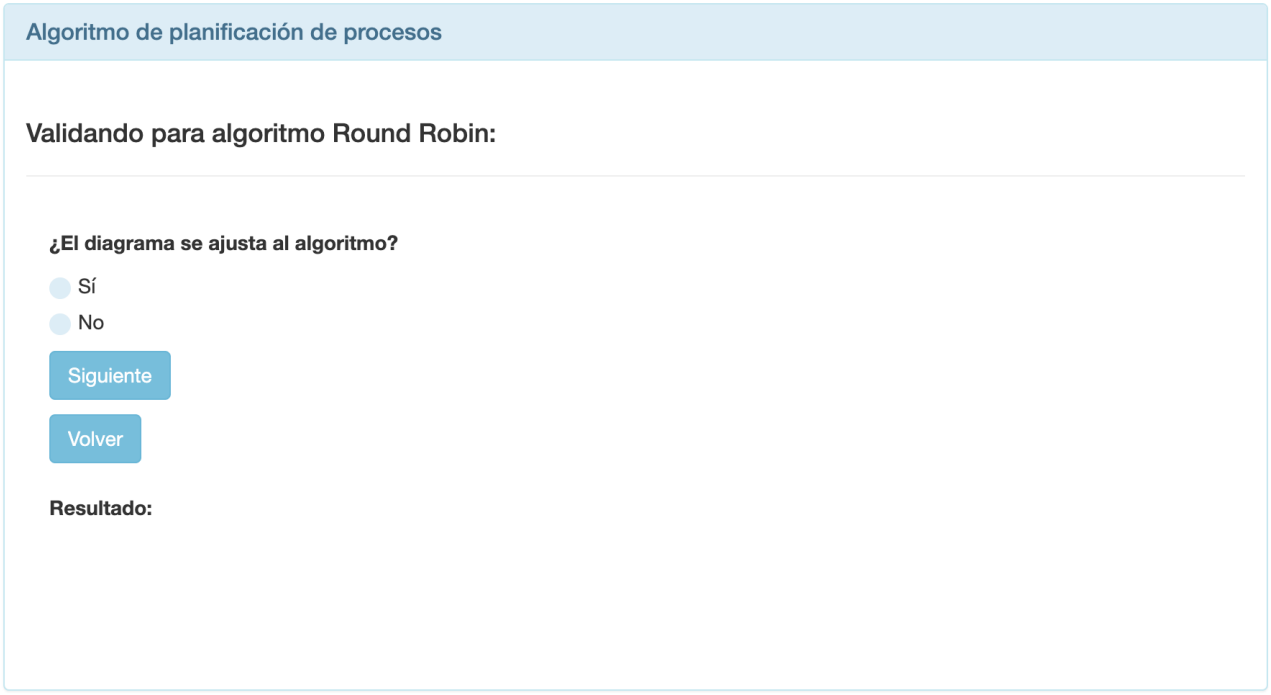


Figure 5.8: Modo Evaluar I

Figure 5.9: Modo Evaluar II

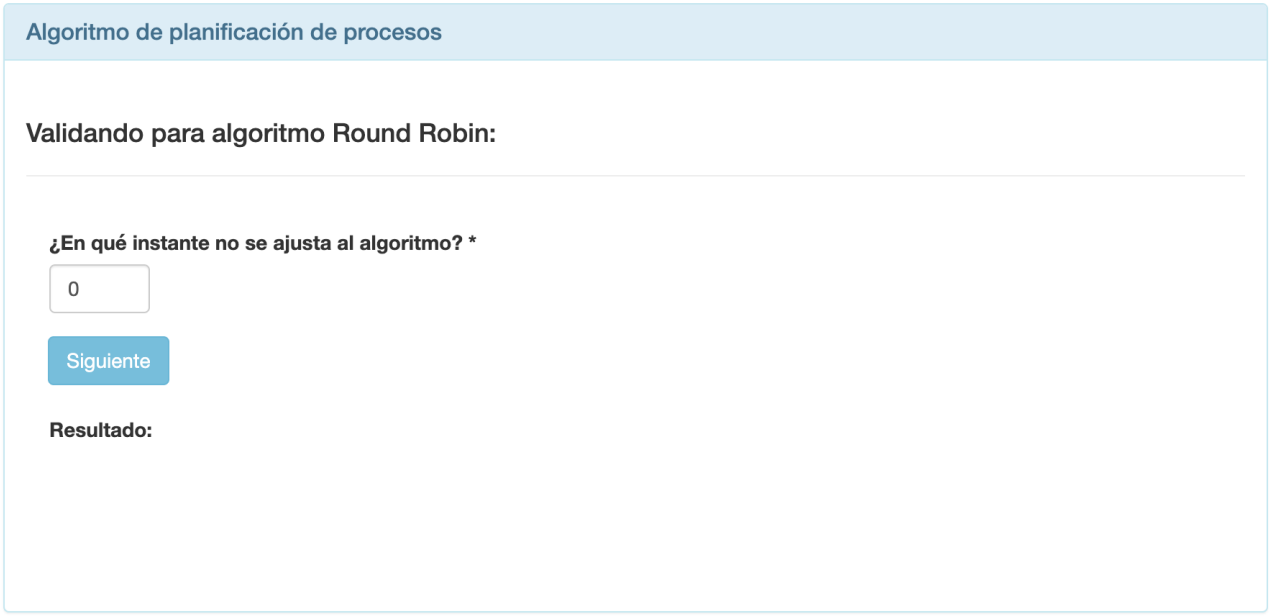


Figure 5.10: Modo Evaluar III

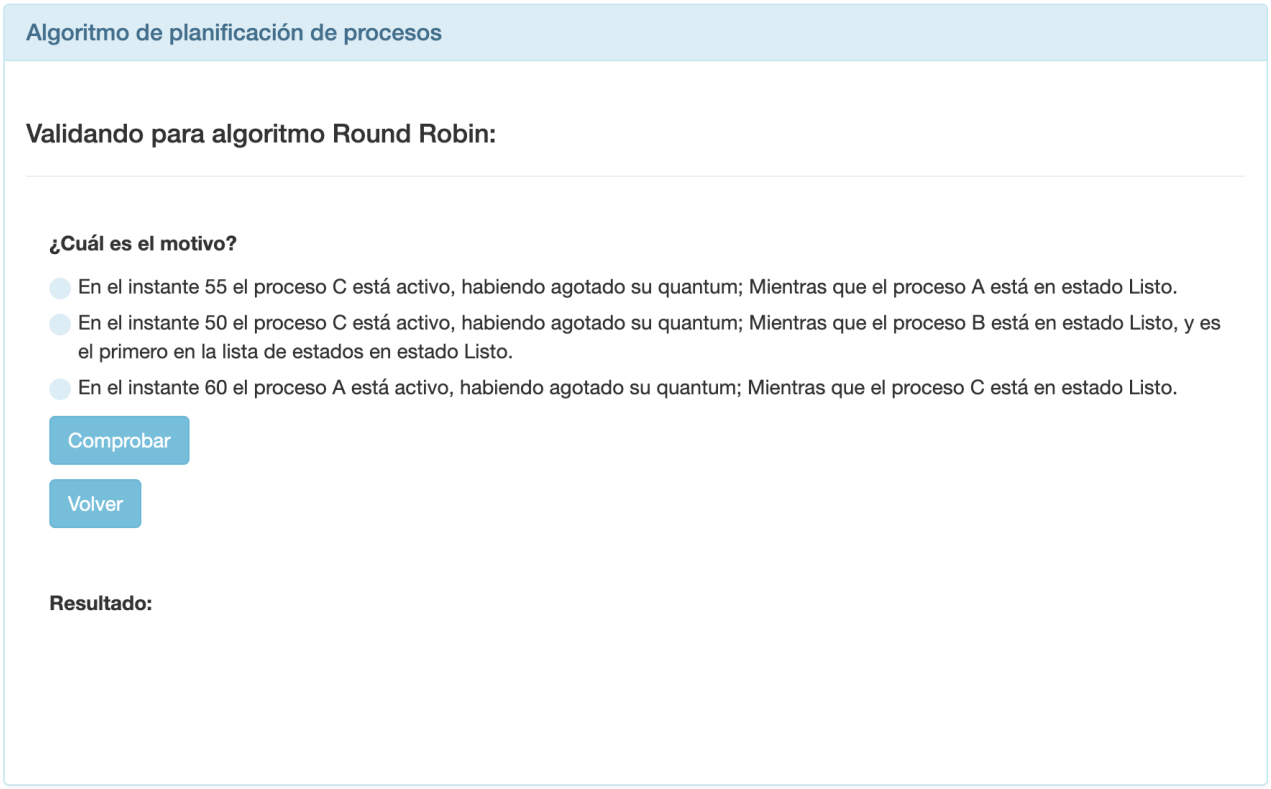


Figure 5.11: Modo Evaluar IV

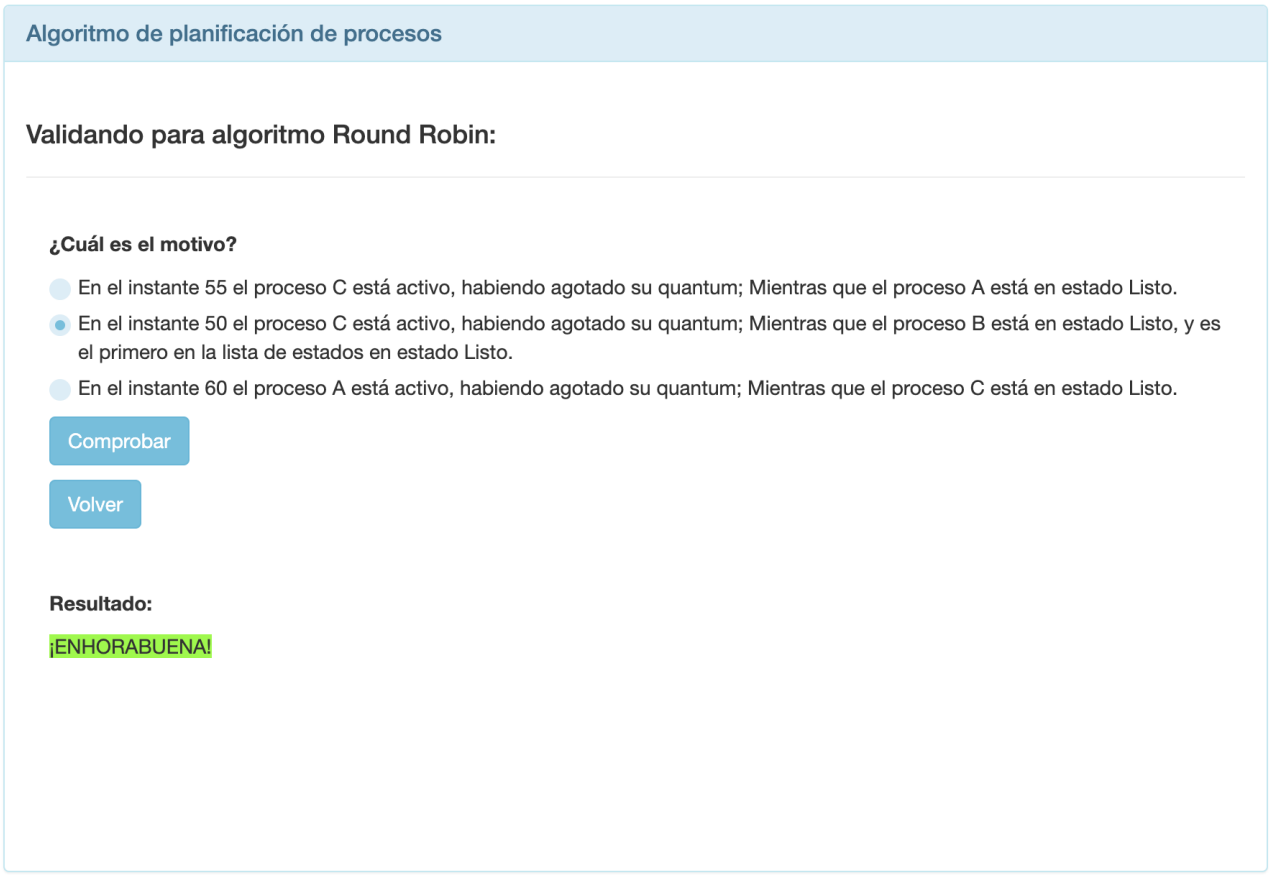


Figure 5.12: Modo Evaluar V

Part II

Gestión de Memoria Virtual

Si no la notas, está haciendo bien su trabajo.

6 Introducción **55**

7 Conceptos básicos sobre memoria virtual **57**

8 Estrategias para gestionar la Memoria Virtual **63**

8.1 Paginación 63

8.2 Segmentación 69

8.3 Paginación/Segmentación (Segmentación paginada o Paginación segmentada) 75

9 Simulador 1: Paginación **85**

9.1 Objetivo 85

9.2 Arquitectura y Tecnologías 86

9.3 Uso del Simulador 90

9.4 Generación automática de ejercicios con IA generativa 96

10 Simulador 2: Segmentación **103**

10.1 Objetivo 103

10.2 Arquitectura y Tecnologías 104

10.3 Uso del Simulador 109

10.4 Generación automática de ejercicios con IA generativa 116

11 Simulador 3: Paginación/Segmentación **121**

11.1 Objetivo 121

11.2 Arquitectura y Tecnologías 122

11.3 Uso del Simulador 126

11.4 Generación automática de ejercicios con IA generativa 142

Bibliography **147**

VI

Introducción

La gestión de la memoria virtual es fundamental en los sistemas operativos modernos, ya que permite que los programas se ejecuten de manera eficiente incluso cuando la memoria física disponible es limitada. Gracias a la memoria virtual, el sistema operativo puede ofrecer a cada proceso la ilusión de contar con un espacio de memoria continuo y suficiente, independientemente de las limitaciones físicas del hardware. Esto se logra mediante la creación de una abstracción que combina tanto la memoria RAM como el almacenamiento secundario, permitiendo que el sistema cargue en memoria solo las partes del trabajo (código más datos) necesarias en un momento determinado. Esta capacidad optimiza el uso de la memoria y evita cuellos de botella, lo cual es crucial para la estabilidad y el rendimiento del sistema.

Además de mejorar la eficiencia en la utilización de la memoria, la memoria virtual también contribuye a la seguridad y la protección del sistema. Al proporcionar a cada proceso su propio espacio de direcciones virtuales, evita que un proceso acceda directamente a la memoria de otro, lo cual protege la integridad de los datos y previene fallos por corrupción de memoria. Esta característica es particularmente importante en sistemas multiprogramados, donde múltiples aplicaciones deben ejecutarse de manera simultánea y aislada, garantizando que un error en un proceso no afecte a los demás. Así, la memoria virtual desempeña un papel crítico en la contención de errores y en la estabilidad general del sistema.

La capacidad de la memoria virtual de manejar programas grandes con una cantidad limitada de memoria física también permite a los desarrolladores escribir aplicaciones sin preocuparse de las restricciones de la memoria disponible en el hardware. Esto habilita el desarrollo de software más complejo y con mayores capacidades, como bases de datos, simulaciones científicas y aplicaciones de procesamiento multimedia que requieren grandes cantidades de datos. La memoria virtual permite que estas aplicaciones funcionen sin interrupciones y proporciona un ambiente donde los recursos se pueden asignar y liberar dinámicamente según las necesidades, mejorando la experiencia del usuario y aprovechando mejor los recursos del sistema.

VII

Conceptos básicos sobre memoria virtual

En este capítulo se presentan de forma general los conceptos teóricos, que se emplean en esta segunda parte, relativos a las herramientas de simulación desarrolladas para trabajar con memoria virtual.

Es sabido que para que una instrucción se pueda ejecutar es necesario que está resida en Memoria Principal. Sin embargo, no siempre se dispone de la cantidad de Memoria Principal suficiente como para mantener en ella un trabajo completo. La primera solución que se adoptó para ejecutar trabajos cuyos tamaños eran mayor que el de la memoria real consistía en la programación *overlays*. Esta forma de programar hacia recaer en el programador la responsabilidad de dividir su trabajo en capas independientes, es decir que se pudiesen ejecutar cada capa (*overlay*) sin necesidad de ninguna otra. De esta forma, en la memoria principal solo se tendría la capa que se estaba ejecutando en ese momento.

Trabajar con *overlays* supone para el programador extender el límite de la memoria principal, pero a costa de tener que realizar una planificación cuidadosa y tediosa a la hora de crear esas capas, limitando la libertad de programación del propio programador. Además, esta forma de programar requiere que el programador conozca de antemano, durante la etapa de implementación, el espacio de memoria principal del que puede disponer, ya que limita el tamaño máximo de las capas a crear. Esto hace que esta solución solo sea factible en Sistemas Operativos monoprogramados.

En 1961, Fotheringham ideó un método para que fuese el sistema operativo el que se encargase de todo llamado **memoria virtual**. Este método permite direccionar un espacio de almacenamiento mucho mayor que el disponible en la memoria principal. Para ello,se usan los dos niveles de almacenamiento (principal y secundaria) tal como se presenta en la figura 7.1. El sistema operativo particiona en bloques los trabajos que están completos en memoria secundaria y copia en memoria principal aquellos bloques que se estén usando en ese momento.

Teniendo en cuenta el tamaño de los bloques se obtienen varias formas de organizar la memoria virtual:

1. Si todos los bloques tienen el mismo tamaño, entonces los bloques reciben el nombre de *páginas* y la organización asociada se denomina *Paginación*.
2. Si los bloques tienen diferentes tamaños, entonces se denominan *segmentos* y su organización *Segmentación*.

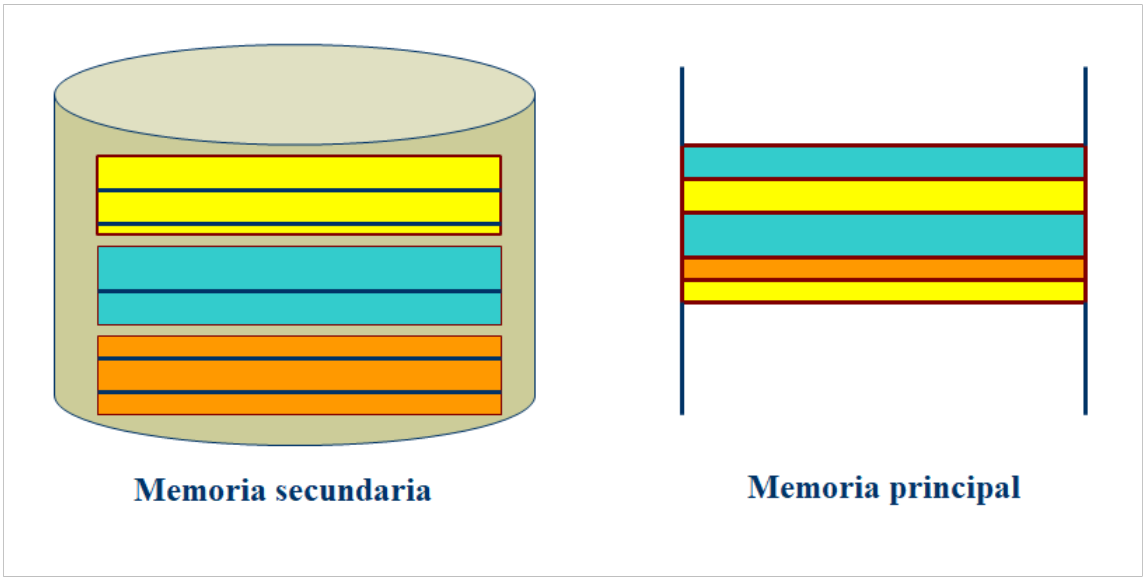


Figure 7.1: Método Memoria Virtual

También se puede combinar ambas técnicas, dando lugar a segmentos de tamaño variable de páginas de tamaño fijo. A esta técnica se le denomina *Paginación/Segmentación* o *Segmentación Paginada*.

Las direcciones a las que hace referencia un proceso durante su ejecución se denominan *direcciones virtuales* y el conjunto de todas ellas forma el *espacio de direcciones virtuales* de un trabajo, tal como se puede observar en la figura 7.2.

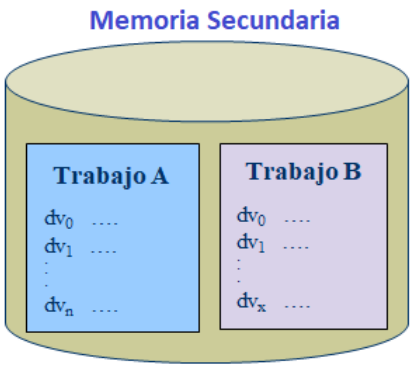


Figure 7.2: Espacio de direcciones virtuales de los trabajos A y B.

Por otra parte, las direcciones que tiene la memoria principal se denominan *direcciones reales* o *direcciones físicas* y al conjunto de todas ellas *espacio de direcciones reales*. Gráficamente se puede observar en la figura 7.3.

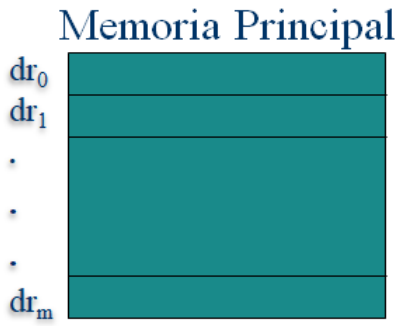


Figure 7.3: Espacio de direcciones reales de la Memoria Principal.

Los procesos durante su ejecución sólo hacen referencia a direcciones virtuales, pero éstas deben ejecutarse en la memoria principal. Por ello, es necesario traducir (transformar) las direcciones virtuales a direcciones reales mientras que el proceso se está ejecutando. Esta transformación se debe realizar rápidamente para conseguir las ventajas que proporciona la memoria virtual.

Los mecanismos de *traducción dinámica de direcciones* (DAT) se encargan de esta transformación durante la ejecución del proceso, basándose en que direcciones contiguas dentro del espacio de direcciones virtuales de un trabajo, no tienen por qué corresponder a direcciones reales contiguas dentro del almacenamiento real.

Para poder llevar a cabo esta transformación, los DAT mantienen por cada trabajo un mapa que indica qué direcciones virtuales se encuentran en memoria principal y dónde. Para que este mapa no sea muy grande las transformaciones se realizan a nivel de bloque y no de palabra de memoria. Por lo tanto, dicho mapa tendrá tantos elementos (filas) como el número de bloques en los que se haya dividido el trabajo, denominándose *tabla de mapa de bloques* o *tabla de bloques*. Los elementos de esta tabla están ordenados de forma secuencial, es decir; el primer elemento da información del primer bloque en que se divide el trabajo, el segundo elemento del segundo bloque, y así sucesivamente. Entre la información que se almacena en cada elemento de la tabla de bloques está la dirección real del inicio de ese bloque en memoria principal, en el caso de que dicho bloque esté almacenado en la memoria principal.

Estas tablas de mapas de bloques siempre se mantienen en memoria principal para que la traducción sea lo más eficiente posible. El espacio de memoria principal ocupado por dichas tablas de mapas de bloques se denomina *fragmentación de tablas*. Cuanto mayor sea el tamaño del bloque, menor será el espacio de memoria principal que ocupa la tabla de mapa de bloques, por lo que menor será la fagmentación de tablas, siendo más rápida la transformación y requiriendo más tiempo la transferencia de información entre los dos niveles de almacenamiento, además de limitar el número de trabajos que van a compartir la memoria principal. Por el contrario, cuanto menor sea el tamaño del bloque, mayor será el número de bloques en los que se divide el

trabajo y mayor será el número de filas (elementos) que tendrá su tabla de mapa de bloques, ocupando por lo tanto más espacio en memoria principal y por lo tanto mayor será la fragmentación de tablas.

Una dirección virtual, *dv*, es un par ordenado *(b, d)*, dónde *b* indica el número del bloque dónde reside el elemento a referenciar, y *d* es el desplazamiento, es decir el número de elementos existentes desde el inicio del bloque al elemento en cuestión.

La traducción (transformación) de una dirección virtual a una dirección real se lleva a cabo siguiendo el siguiente procedimiento:

1. La CPU tiene un registro denominado *registro origen de la tabla de mapa de bloques* que contiene siempre la dirección real, *a*, donde comienza la tabla de mapa de bloques del proceso activo, es decir, que se está ejecutando sobre dicha CPU.
2. Al contenido de ese registro se le añade (concatena) el número del bloque, *b* (primera componente de la dirección virtual), para obtener la dirección real donde está el elemento (fila) dentro de la tabla de bloques que contiene la información correspondiente a dicho bloque.
3. Se accede al contenido de esa dirección real (fila de la tabla de bloques) para obtener la dirección real (*b'*) donde comienza el bloque referenciado en memoria principal.
4. A esa dirección real de inicio del bloque, *b'*, se le añade (concatena) el desplazamiento (*d*, segunda componente de la dirección virtual) obteniendo la dirección real deseada (*dr*).

Estos pasos se ven reflejados gráficamente en la figura 7.4.

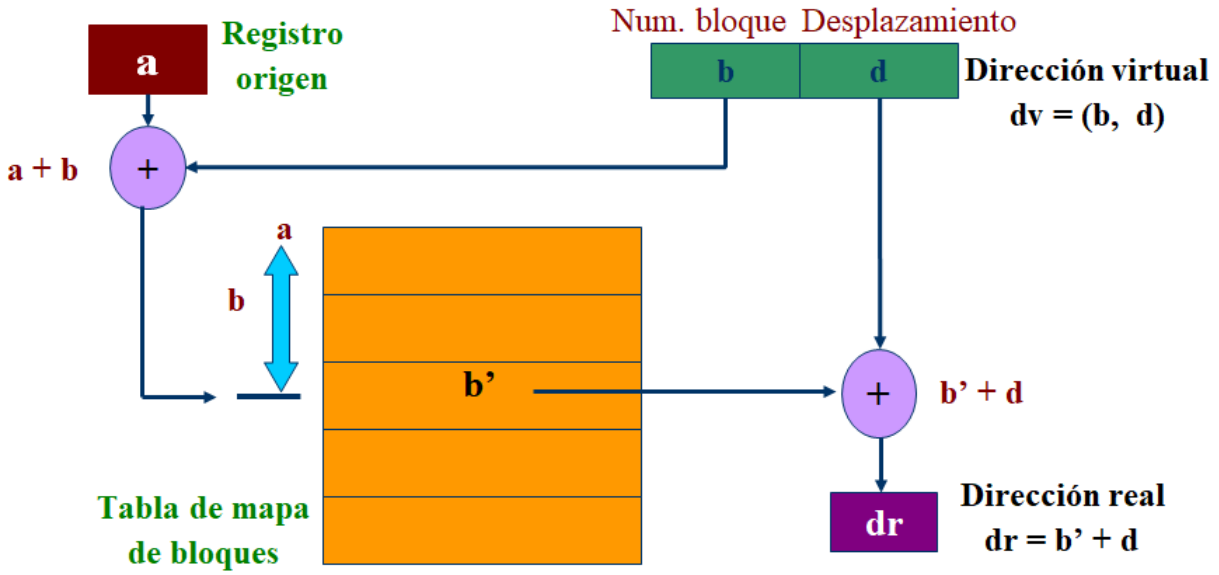


Figure 7.4: Traducción de una dirección virtual a una dirección real.

Todo mecanismo que pretenda gestionar la memoria debe proporcionar protección, es decir evitar que un proceso, durante su ejecución, invada el contenido de otro trabajo. En memoria virtual la protección se consigue mediante las *claves de protección*. Cada bloque en memoria principal tiene asociado la clave del usuario al que pertenece el contenido de dicho bloque. Todos los bloques pertenecientes al mismo trabajo tendrán la misma clave de protección. La CPU dispone de un registro que va a contener en todo momento la clave de protección del bloque que está ejecutando el proceso que está activo, de forma que cualquier referencia a memoria que se haga durante la ejecución de dicho proceso solo podrá ser a bloques que tengan esa misma clave de protección. Esto se puede observar en la figura 7.5.

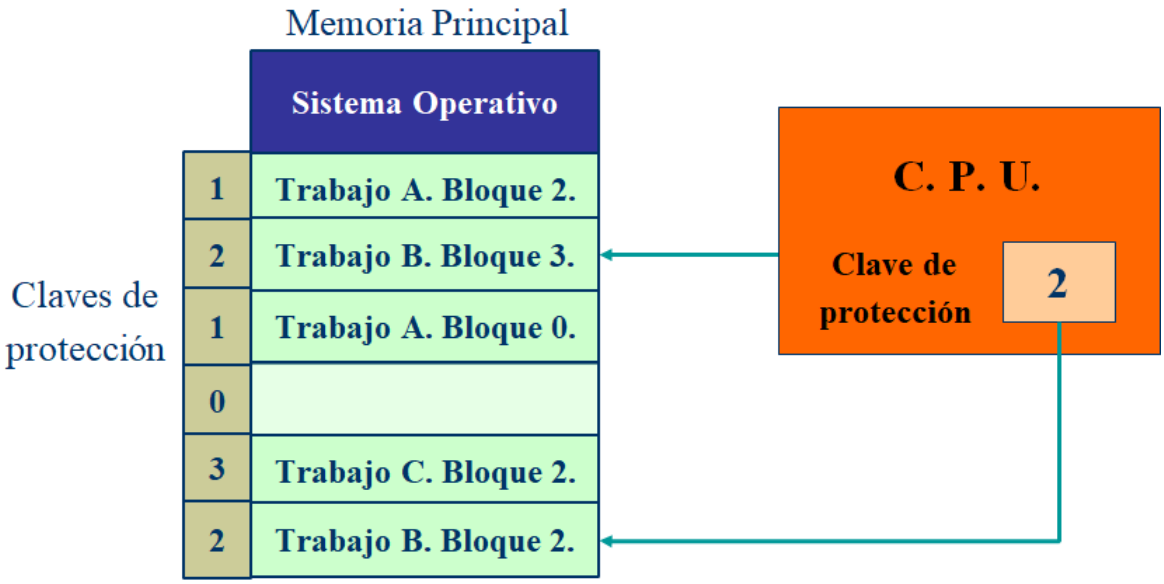


Figure 7.5: Protección: claves de protección.

En el siguiente apartado se van a analizar las distintas estrategias que existen para organizar la memoria virtual. Todas ellas dividen un trabajo en bloques de un tamaño suficiente para ser cargado en memoria principal y permitiendo que solo los bloques activamente necesarios residan en memoria principal, manteniendo todo el trabajo en memoria secundaria.

Tanto en ME QUEDO AQUI CON ESTRATEGIAS DE REPOSICION DE pagina

VIII

Estrategias para gestionar la Memoria Virtual

8.1

Paginación

Esta organización de la memoria virtual consiste en dividir el espacio virtual de direcciones de un trabajo en trozos del mismo tamaño, denominadas *páginas* o *páginas lógicas*, y la memoria principal en bloques del mismo tamaño que el tamaño máximo de las páginas lógicas, denominados *marcos de página* o *páginas físicas*, tal y como se recoge en la figura 8.1.

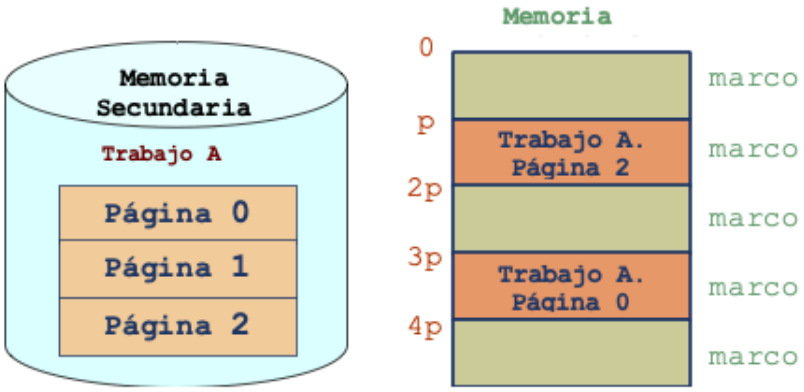


Figure 8.1: Organización del espacio virtual de direcciones y de la memoria principal en Paginación

De esta organización se pueden deducir las siguientes conclusiones:

1. Los marcos de páginas van a comenzar en direcciones de almacenamiento real que son múltiplos enteros del tamaño del marco de página (este tamaño coincide con el tamaño máximo de la página). Si *p* es el tamaño del marco de página, el primer marco de página comienza en la dirección *0* y termina en la dirección *p-1*, el segundo marco de página comienza en la dirección *p* y termina en la dirección *2p-1*, y así sucesivamente.
2. El intercambio de información se hace a nivel de páginas, es decir; se copia una página completa del espacio virtual de direcciones en cualquier marco de página disponible, y viceversa, se lleva una página completa de la memoria principal al espacio virtual de direcciones (memoria secundaria).
3. Las páginas continuas de un trabajo en el espacio virtual de direcciones no tienen por qué ocupar marcos de página continuos en memoria principal.

Para realizar la traducción dinámica de direcciones, cada trabajo tiene asociado una *tabla de mapas de páginas*, con tantos elementos como el número de páginas en las que se ha dividido el trabajo, tal y como se presenta en la figura 8.2.

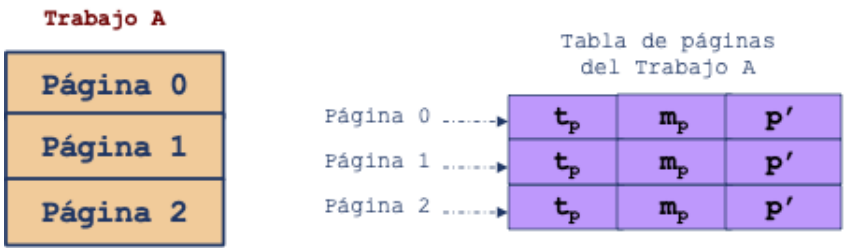


Figure 8.2: Formato de la tabla de mapa de páginas de un trabajo

El formato típico de un elemento de la tabla de páginas sería:

- t_p *bit de residencia* de página en memoria principal (normalmente t_p = 0 la página no reside; t_p = 1, la página si reside);
- m_p *dirección* donde está la página en el *almacenamiento secundario*;
- p' *dirección real del comienzo del marco de página*. A veces no contiene esa dirección real, sino que contiene el *número del marco de página*.

Una dirección virtual *dv* viene dada por un par ordenado *(p,d)*, donde *p* (página lógica) es el número de página en el espacio de direcciones virtual, y *d* es el desplazamiento en la página, es decir el número de elementos que existe desde el comienzo de la página al elemento que se está referenciando dentro de esa página.

Existen varias técnicas para realizar la traducción de direcciones virtuales, que se describen en los siguientes apartados.

8.1.0.1 Por transformación directa

El Sistema Operativo tiene cargado en el *registro origen* la dirección de memoria principal, *a,* donde comienza la tabla de páginas del trabajo. Cuando un proceso en ejecución hace referencia a una dirección virtual *dv = (p, d)*, los pasos que se siguen para obtener la dirección real son:

1. se añade a la dirección base de la tabla de páginas, *a*, el número de página *p* para formar la dirección real de memoria donde se encuentra la casilla de la tabla de páginas que contiene la información de la página *p*.
2. se accede a esa dirección real de memoria para comprobar si la página reside o no en memoria principal (usando el bit de residencia, t_p) y para conocer la dirección real donde comienza el marco de página, *p',*

que contiene la página virtual *p*,

3. se concatena la dirección real donde comienza el marco de página con el desplazamiento, *d*, obteniendo la dirección real, *dr,* deseada.

Este proceso se puede representar gráficamente de la siguiente forma (figura 8.3):

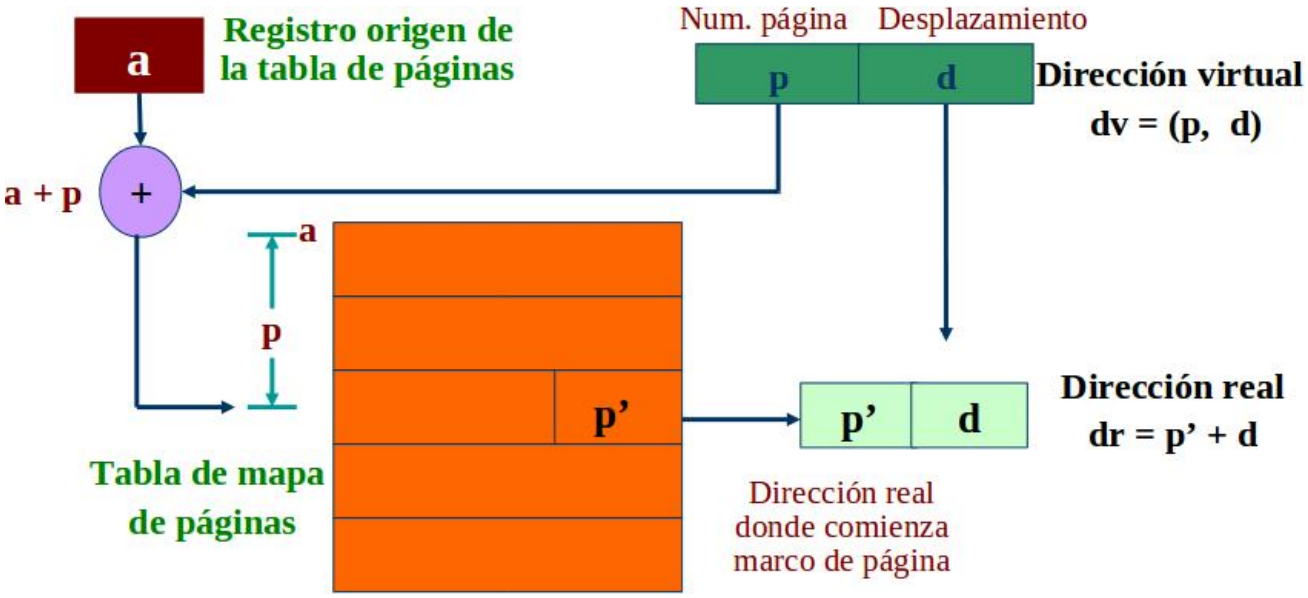


Figure 8.3: Traducción de dirección virtual a dirección real Por Transformación Directa

La dirección virtual y la dirección base de la tabla de páginas se mantienen en registros de alta velocidad de la CPU. Sin embargo, la tabla de páginas, que puede ser bastante grande, se mantiene completamente en memoria principal. Esto da lugar a que cada acceso a la tabla requiera un ciclo completo, haciendo que los programas puedan tardar el doble de tiempo en ser ejecutados, ya que la mayoría del tiempo de su ejecución se pierde en el acceso a memoria.

8.1.0.2 Por transformación asociativa

Para acelerar esta traducción se coloca la tabla de páginas completa en un *almacenamiento asociativo.* La arquitectura que soporta la memoria asociativa conlleva que se acceda a todo su contenido en un solo acceso, el cual requiere un tiempo de ciclo menor que la memoria principal. El proceso de traducción sería el siguiente:

1. se accede simultáneamente a todas las casillas que componen la tabla de páginas, que está en memoria asociativa, para localizar aquella que contiene la información referente a la página virtual *p*,
2. se comprueba con la información que tiene dicha casilla si la página reside o no en memoria principal (usando el bit de residencia t_p) y se obtiene, de esa misma casilla, la dirección real donde comienza el

marco de página, *p'*, que contiene la página virtual *p*,

3. se concatena la dirección real donde comienza el marco de página con el desplazamiento, *d*, obteniendo la dirección real, d*r*, deseada.

Gráficamente sería como se recoge en la figura 8.4:

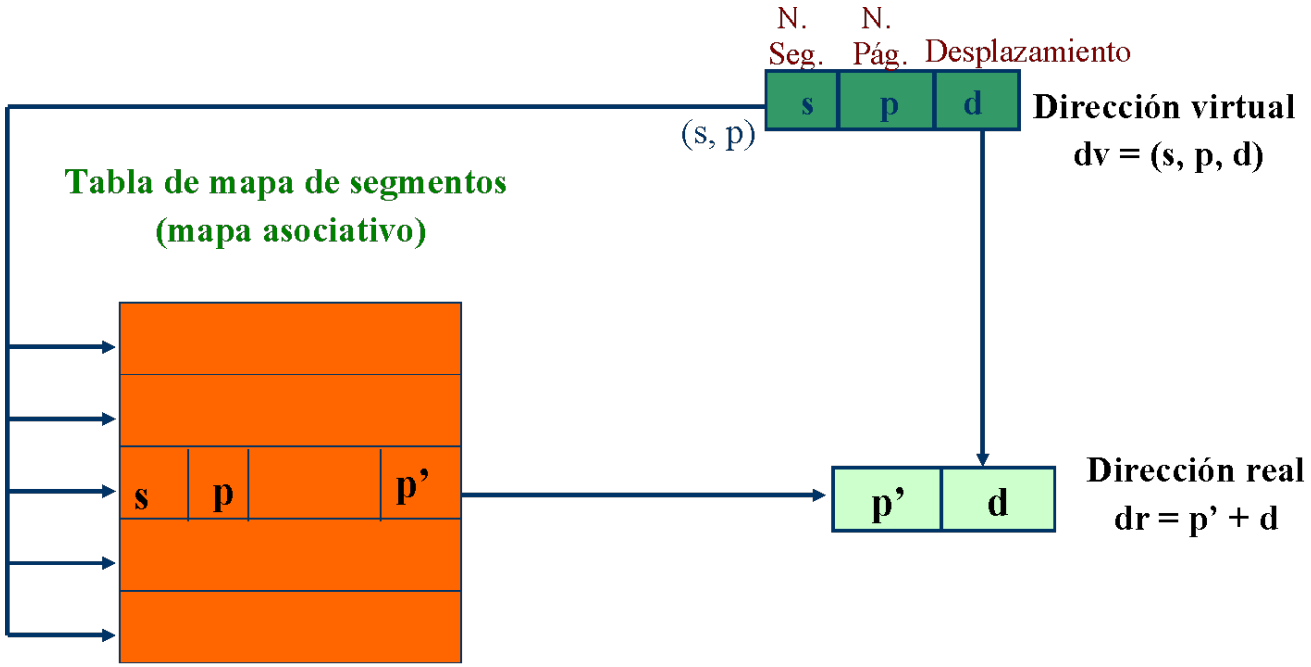


Figure 8.4: Traducción de dirección virtual a dirección real Por Transformación Asociativa

8.1.0.3 Por combinación de la transformación asociativa y la directa

Se usa un almacenamiento asociativo para mantener sólo un pequeño trozo de la tabla de páginas de un trabajo. Normalmente contendrá la información sobre las páginas que han sido referenciadas recientemente, esto es debido a que una página que ha sido referenciada hace poco, tiene más posibilidades de ser referenciada de nuevo, en el futuro. Pero, además la tabla de páginas de un trabajo se mantendrá completa en memoria principal.

1. primero, se intenta encontrar la página referenciada, *p*, en el trozo de tabla contenida en memoria asociativa, accediendo simultáneamente a todas las casillas que componen dicha memoria asociativa, para localizar aquella que contiene la información referente a la página virtual,
2. si se localiza, se comprueba si la página reside o no en memoria principal (bit de residencia t_p) obteniendo la dirección real donde comienza el marco de página, *p'*, que contiene la página virtual *p*, y se concatena la dirección real donde comienza el marco de página con el desplazamiento, *d*, para

obtener la dirección real,

3. si no se localiza, entonces se utiliza la tabla de páginas almacenada en memoria principal aplicando la primera técnica de traducción, *por transformación directa*. Primero, se añade a la dirección base de la tabla de páginas, a, el número de página p para formar la dirección real de memoria donde se encuentra la casilla dentro de la tabla de páginas que contiene la información de la página p. Luego, se accede a esa dirección real para comprobar si la página reside o no en memoria principal (bit de residencia t_p) y para conocer la dirección real donde comienza el marco de página, p', que contiene la página virtual p. Y por último, se concatena la dirección real donde comienza el marco de página con el desplazamiento, d, obteniendo la dirección real, dr, deseada.

En la figura 8.5 se recoge el proceso anteriormente descrito.

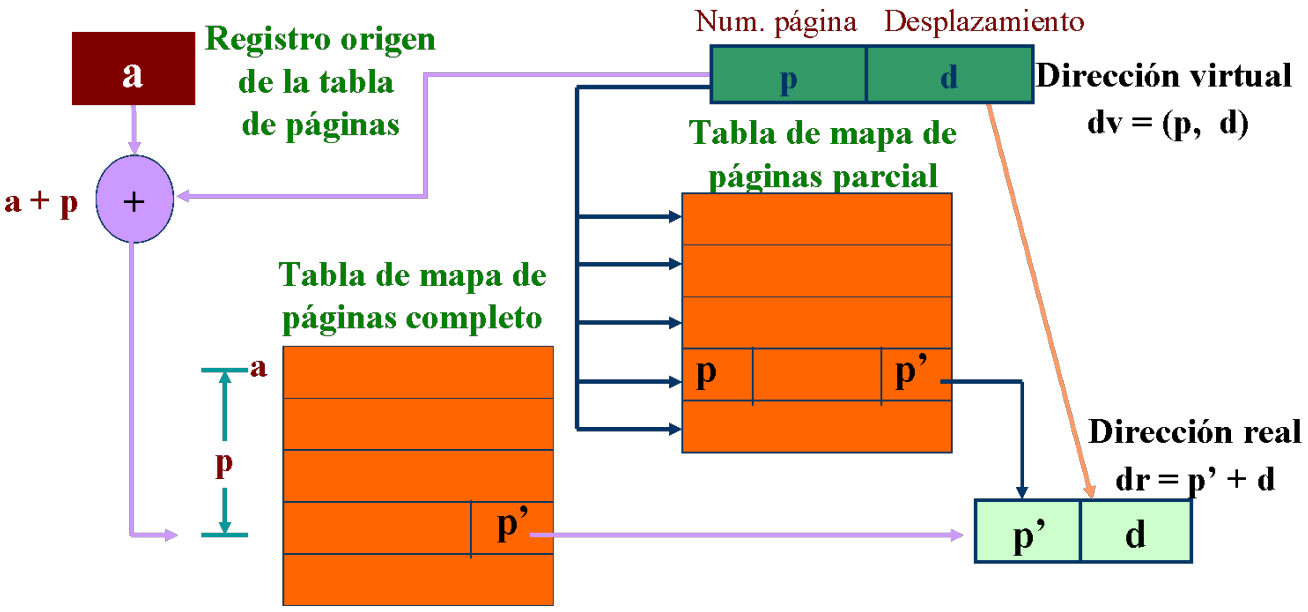


Figure 8.5: Traducción de dirección virtual a dirección real Por Combinación de la Transformación Asociativa y la directa

Con cualquiera de las tres técnicas de traducción analizadas, se produce un fallo denominado **fallo de pérdida de página**. Este fallo se produce cuando al acceder a la casilla, de la tabla de páginas, que contiene información referente a la página lógica, p, se comprueba que el bit de referencia, t_p, informa de que dicha página no se encuentra en memoria principal. La solución a este fallo estriba en hacer una copia de dicha página de memoria secundaria a memoria principal, ocupando un marco de página libre. Para localizar esa página en memoria secundaria se usa el campo m_p.

Durante la etapa del diseño del Sistema Operativo se decide el tamaño de los marcos de página, es decir el tamaño máximo de las páginas lógicas. Cuando el Sistema Operativo acepta un nuevo trabajo, este se divide en

páginas lógicas teniendo en cuenta dicho tamaño máximo. Como consecuencia, el contenido de una página no tiene identidad propia, es decir, puede contener parte del programa, o parte de los datos, o en algunos casos parte del programa y de los datos del trabajo. Esto hace que las páginas sean difíciles de compartir por varios procesos, pues en el momento en que una página contenga un dato no compartible, hace que toda la página sea no compartible. En definitiva, cada página se identifica como compartible o no.

La figura 8.6 muestra cómo se puede compartir páginas. Si las tablas de páginas de varios trabajos hacen referencia al mismo marco de página, la página contenida en ese marco será compartida por los procesos procedentes de esos trabajos. Con la compartición, se reduce la cantidad de memoria principal necesaria para la ejecución de un conjunto de procesos permitiendo que se mantenga un mayor número de ellos.

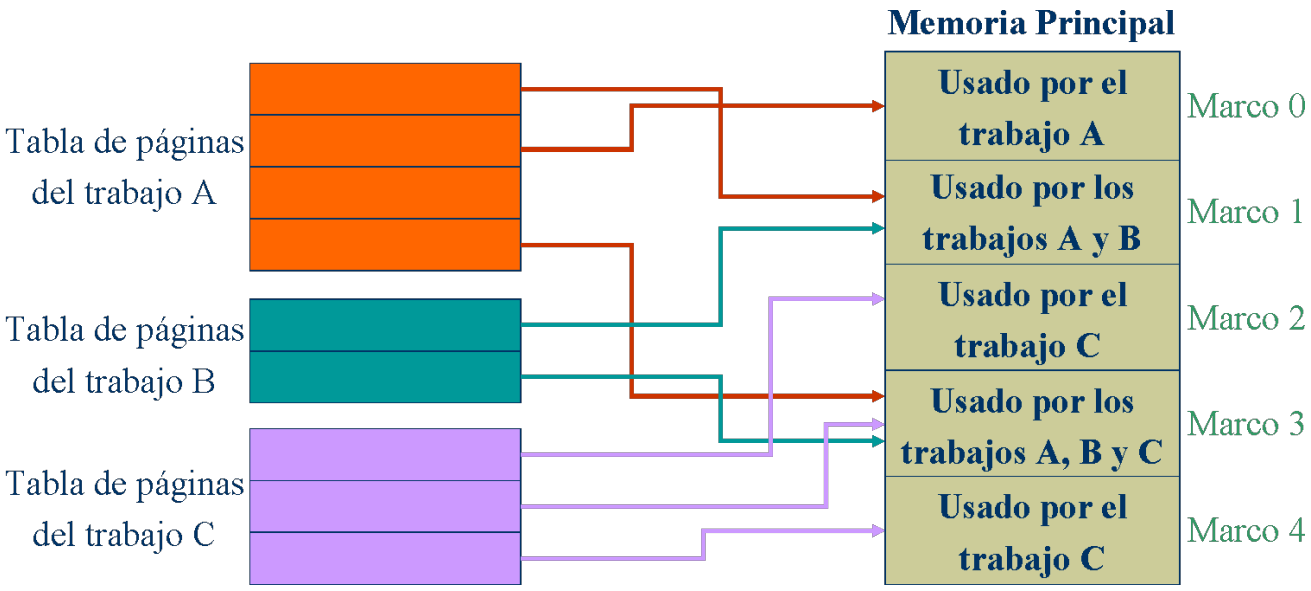


Figure 8.6: Compartición de páginas

La probabilidad de que el tamaño de un trabajo sea múltiplo entero del tamaño máximo de página (tamaño de marco de página) es muy pequeña, por lo tanto, la última página de cada trabajo tendrá casi siempre un tamaño menor que el máximo citado. Cuando esta última página se copie en un marco de página de la memoria principal no ocupará todo ese marco, por lo que el trozo de memoria principal perteneciente a ese marco de página pero no usado por la página lógica se desperdicia. A este desperdicio se denomina *fragmentación interna*, pues es espacio de memoria principal asignado al trabajo pero no usado por él.

8.2

Segmentación

En esta organización el espacio virtual de direcciones de un trabajo se divide en trozos de distinto tamaño, llamados *segmentos*, permitiendo que el contenido de cada uno de ellos tenga *identidad propia*. Esta es la ventaja de la segmentación sobre la paginación, ya que es un concepto más lógico que físico. Los segmentos, en principio, no tienen un tamaño restringido. Un segmento que contenga una matriz tendrá el tamaño de dicha matriz. Si el segmento contiene una estructura dinámica de datos su tamaño aumentará o disminuirá conforme lo haga la estructura de datos. Si el segmento contiene el código obtenido de compilar un programa tendrá el tamaño de ese código. De esta forma, se consigue la organización lógica que se deseaba. Es el Sistema Operativo el que realiza esta división en segmentos teniendo en cuenta la información que le proporciona los compiladores y lincadores (donde comienza y termina cada unidad de código, el tamaño de cada estructura de datos, etc.).

Para que un proceso este activo es necesario que su segmento actual esté en memoria principal. Los segmentos se transfieren de la memoria secundaria a la principal y viceversa de forma completa, ocupando todas las posiciones del segmento posiciones contiguas en la memoria principal. Un nuevo segmento ocupará una serie de posiciones contiguas de la memoria principal de tamaño suficiente para alojar el segmento completo. En definitiva, el intercambio de información se hace a nivel de segmentos. No obstante, es conveniente tener en cuenta que los segmentos contiguos de un trabajo en su espacio de direcciones virtuales pueden ocupar bloques no adyacentes en memoria principal.

La figura 8.7 muestra una representación de la organización del espacio virtual de direcciones y de la memoria principal en Segmentación.

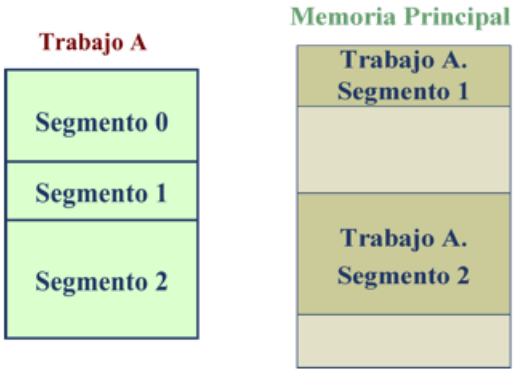


Figure 8.7: Organización del espacio virtual de direcciones y de la memoria principal en Segmentación

Debido a que cada segmento tiene identidad propia, se puede controlar el acceso de los distintos procesos a

los segmentos, dándole o negándole ciertos derechos de acceso. Los tipos más comunes de control de acceso a un segmento son:

1. *Acceso de lectura: R*. El proceso puede obtener la información que contiene el segmento.
2. *Acceso de escritura: W*. El proceso puede modificar toda la información que contiene el segmento, puede añadirle información, e incluso puede destruir todo el contenido del segmento.
3. *Acceso de ejecución: E*. El proceso puede ejecutar el segmento como si fuese un programa. Obviamente los segmentos de datos no dispondrán de este acceso.
4. *Acceso de adición: A*. El proceso puede añadirle información al final del segmento, pero no puede modificar el contenido existente.

Con estos cuatro tipos de acceso se pueden crear 16 diferentes modos de control de acceso. Algunos son incoherentes y otros interesantes. Por ejemplo, si a un proceso se le permite modificar un segmento, pero no leerlo, no tiene sentido. Con este control de acceso se protege a los segmentos.

Cada trabajo tiene asociado una tabla de mapas de segmentos, con tantos elementos como el número de segmentos en los que se ha dividido el trabajo, tal y como se muestra en la figura 8.8.

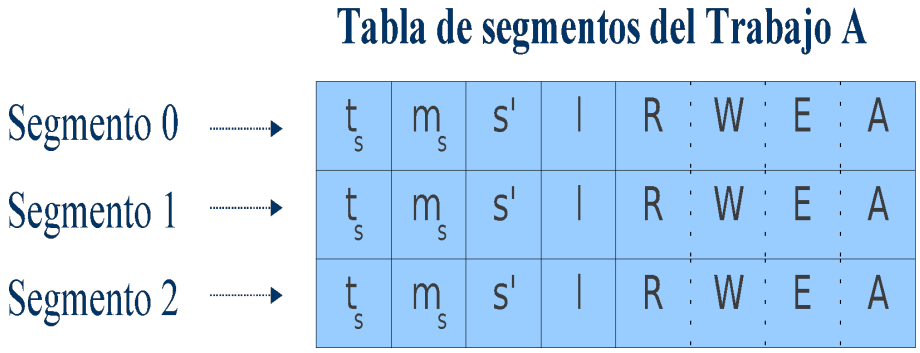


Figure 8.8: Formato de la tabla de mapa de segmentos de un trabajo

El formato típico de un elemento de la tabla de segmentos sería:

- t_s : *bit de residencia* del segmento en memoria principal (normalmente t_s = 0 el segmento no reside, t_s = 1 el segmento si reside);
- m_s : *dirección* donde está el segmento en el *almacenamiento secundario*;
- s' : *dirección real del comienzo (base) del segmento*.
- l : *longitud del segmento* o *dirección real del final del segmento*. Como cada segmento puede tener un tamaño diferente es necesario conocerlo para poder protegerlo.
- R, W, E, A : *bits de protección* (si vale 1 si posee ese acceso y si vale cero no lo posee)

Una dirección virtual en un sistema segmentado viene dada por un par ordenado *dv = (s,d)*, donde *s* es

el número del segmento dentro del almacenamiento virtual, y *d* es el desplazamiento en el segmento *s*, es decir el número de elementos que existen desde el comienzo de ese segmento hasta el elemento que se está referenciando dentro de ese segmento.

Para realizar la *traducción dinámica de direcciones*, en segmentación existen las mismas técnicas que en paginación; es decir *por transformación directa*, *por transformación asociativa* o *por combinación de transformación asociativa/directa*. Se describen en las siguientes secciones.

8.2.0.1 Por transformación directa

El Sistema Operativo tiene cargado en el *registro origen* la dirección de memoria principal donde comienza la tabla de segmentos del trabajo, *a*. Cuando un proceso en ejecución hace referencia a una dirección virtual *dv = (s, d)*, los pasos que se siguen para obtener la dirección real son:

1. se añade a la dirección base de la tabla de segmentos, *a,* el número de segmento, *s,* para formar la dirección real de memoria donde se encuentra la casilla de la tabla de segmentos que contiene la información del segmento *s*.
2. se accede a esa dirección real de memoria para comprobar si el segmento reside o no en memoria principal (usando el bit de residencia t_s), para controlar si se está solicitando el acceso a un elemento que pertenece o no al segmento (usando *l*), para comprobar si la operación solicitada se puede o no realizar (utilizando los bits de protección) y para conocer la dirección real donde comienza el segmento, *s',* en memoria principal,
3. se concatena la dirección real donde comienza el segmento con el desplazamiento, *d*, obteniendo la dirección real, *dr,* deseada.

En la figura 8.9 se recoge una representación gráfica de este proceso.

La dirección virtual y la dirección base de la tabla de segmentos se mantienen en registros de alta velocidad de la CPU. Sin embargo, la tabla de segmentos se mantiene completamente en memoria principal. Esto da lugar a que cada acceso a la tabla requiera un ciclo completo, haciendo que los programas puedan tardar el doble de tiempo en ser ejecutados, ya que la mayoría del tiempo de su ejecución se pierde en el acceso a memoria.

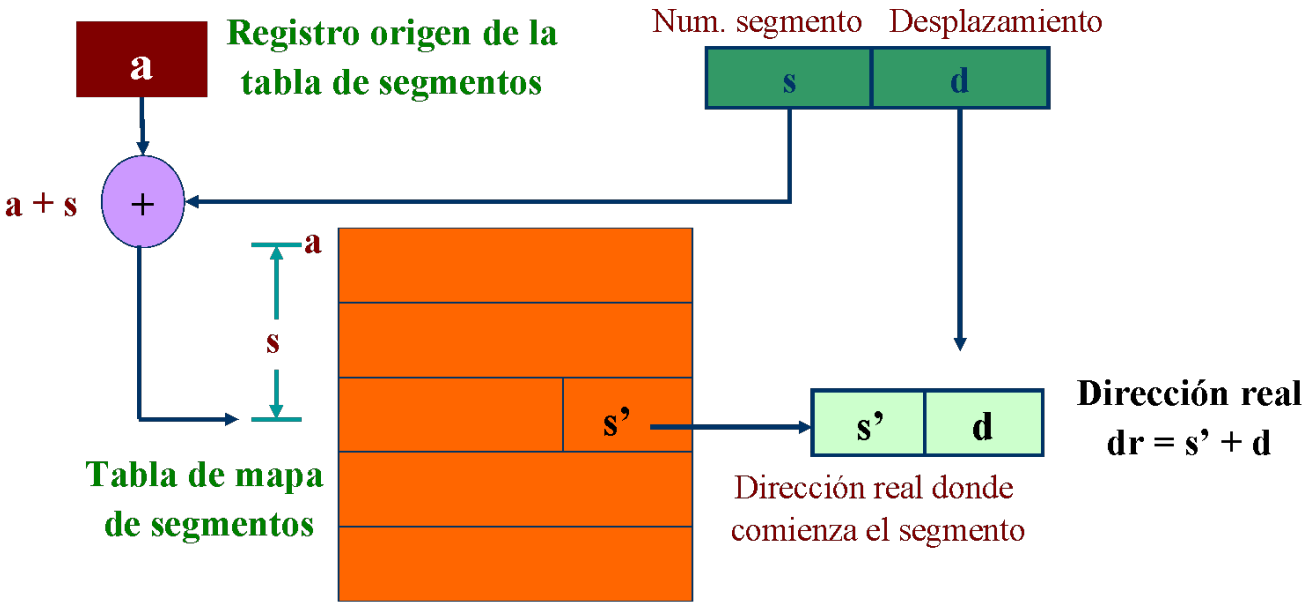


Figure 8.9: Traducción de dirección virtual a dirección real Por Transformación Directa

8.2.0.2 Por transformación asociativa

Al igual que en paginación, se acelera la traducción si se dispone de memoria asociativa para almacenar en ella toda la tabla de segmentos. El proceso de traducción sería:

1. se accede simultáneamente a todas las casillas que componen la tabla de segmentos almacenada en memoria asociativa, para localizar aquella que contiene la información referente al segmento s,
2. con la información que tiene esa casilla se comprueba si el segmento reside o no en memoria principal (usando el bit de residencia t_s), se controla si se está solicitando el acceso a un elemento que pertenece o no al segmento (usando l), se comprueba si la operación solicitada se puede o no realizar (empleando los bits de protección) y se conoce la dirección real donde comienza el segmento, s', en memoria principal,
3. se concatena la dirección real donde comienza el segmento con el desplazamiento, d, obteniendo la dirección real, dr, deseada.

Gráficamente sería como se muestra en la figura 8.10.

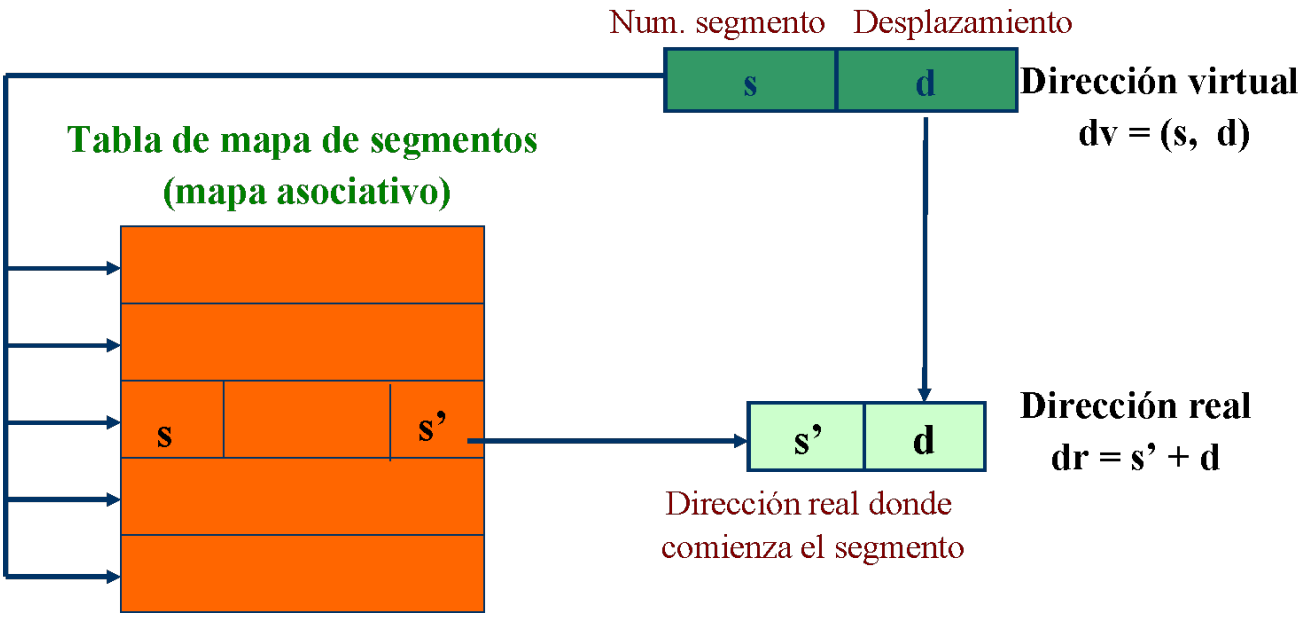


Figure 8.10: Traducción de dirección virtual a dirección real Por Transformación Asociativa

8.2.0.3 Por combinación de la transformación asociativa y la directa

En memoria asociativa se mantiene un pequeño trozo de la tabla de segmentos del trabajo (normalmente corresponderá a los segmentos que han sido referenciados recientemente), y en memoria principal se mantendrá toda la tabla de segmentos del trabajo. El proceso a seguir para realizar la traducción sería:

1. primero, se intenta encontrar el segmento referenciado, s, en el trozo de tabla contenida en memoria asociativa, accediendo simultáneamente a todas las casillas que componen dicha memoria asociativa, para localizar aquella que contiene la información referente al segmento,
2. si se localiza, con la información que tiene dicha casilla se comprueba si el segmento reside o no en memoria principal (utilizando el bit de residencia t_s), se comprueba si se desea acceder a un elemento que pertenece o no al segmento (usando l), se controla si la operación solicitada se puede o no realizar (empleando los bits de protección) y se obtiene la dirección real donde comienza el segmento, s', que se concatena con el desplazamiento, d, para obtener la dirección real, dr, deseada.
3. si no se localiza, entonces se utiliza la tabla de segmentos almacenada en memoria principal. Primero, se añade a la dirección base de la tabla de segmentos, a, el número de segmento s para formar la dirección real de memoria donde se encuentra la casilla dentro de la tabla de segmentos que contiene la información del segmento s. Luego, se accede a esa dirección real para comprobar si el segmento reside o no en memoria principal (utilizando el bit de residencia t_s), se comprueba si se desea acceder a un elemento que pertenece o no al segmento (usando l), se controla si la operación solicitada se puede o no realizar (empleando los bits de protección) y se obtiene la dirección real donde comienza el segmento,

s'. Y por último, se concatena esta dirección real con el desplazamiento, *d*, obteniendo la dirección real, *dr*, deseada.

Gráficamente sería como en la figura 8.11:

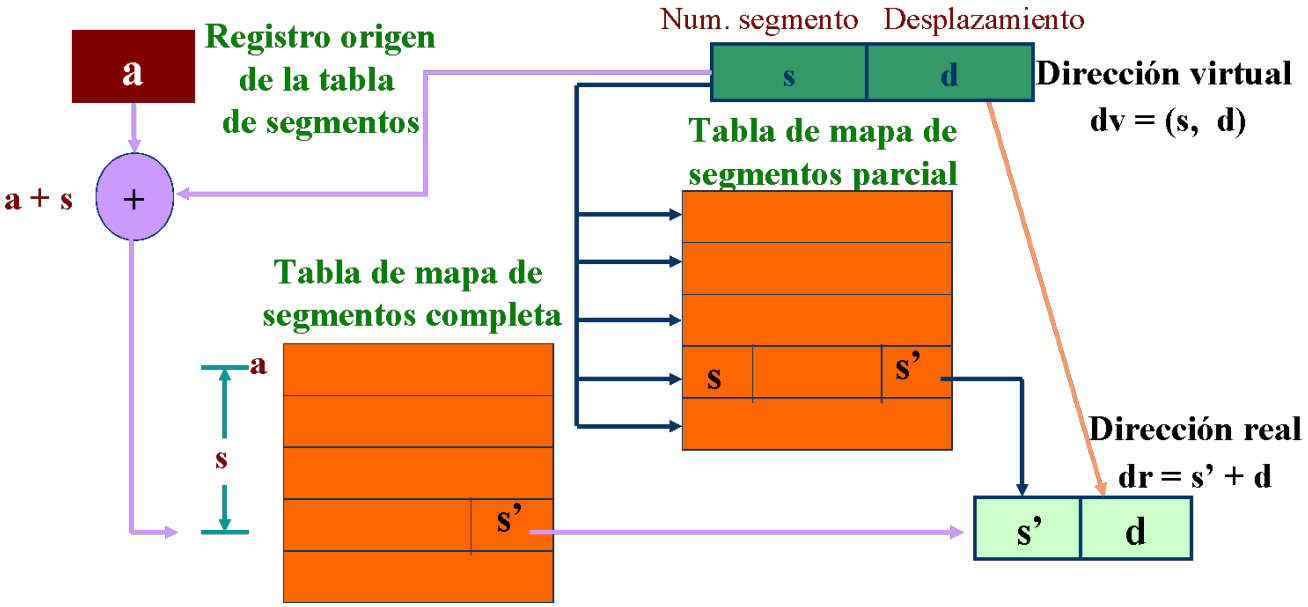


Figure 8.11: Traducción de dirección virtual a dirección real Por Combinación de la Transformación Asociativa y la directa

Con cualquiera de las tres técnicas de traducción descritas, puede producirse los siguientes fallos:

- *Fallo de pérdida de segmento*. Se produce cuando el segmento al que se hace referencia no reside en la memoria principal. Este fallo se detecta comprobando el bit de residencia,t_s, y se soluciona haciendo una copia en memoria principal de dicho segmento. La dirección de dicho segmento en el almacenamiento secundario viene indicada por m_s.
- *Fallo de desbordamiento de segmento*. Se produce cuando se hace referencia a una posición que no pertenece al segmento especificado. Este fallo se detecta realizando la siguiente verificación:
  1. Si l indica la longitud del segmento, se comprueba que $d \leq l$.
  2. Si l indica la dirección real donde termina el segmento, se comprueba que $s' + d \leq l$.

  Si no se cumple la comprobación correspondiente se determina que se ha producido este fallo y se aborta la ejecución del proceso.
- *Fallo de protección de segmento*. Se produce cuando se solicita una operación que no está permitida. Este fallo se detecta comprobando los bits de protección, y si ocurre se cancela la ejecución del proceso.

Los segmentos son más fáciles de compartir que las páginas. Si una estructura de datos dinámica ocupa varias páginas, tendremos varios elementos en las tablas, uno por cada página, y además si la estructura aumenta de

tamaño dando lugar a una nueva página, las entradas en la tabla de páginas deben modificarse en tiempo de ejecución. Esto no ocurre en segmentación, ya que sólo se tendrá un segmento para contener la estructura de datos y su tamaño variará según el tamaño de está.

Un segmento es compartido por dos procesos si en sus tablas de mapa de segmentos existe una entrada por ese mismo segmento, tal y como se muestra en la figura 8.12.

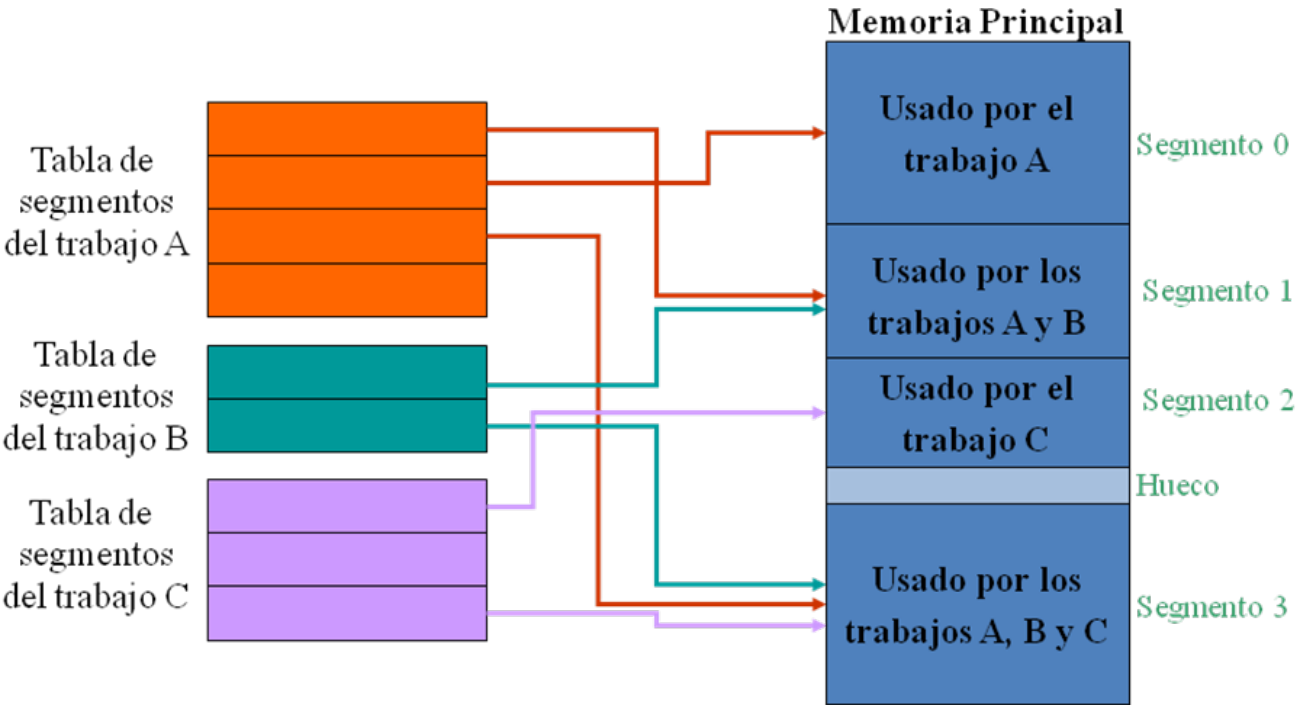


Figure 8.12: Compartición de segmentos

En segmentación se produce *fragmentación externa*, ya que cuando un segmento no se necesita, el espacio ocupado por él en memoria principal se libera (hueco) para que se use a la hora de copiar otro segmento en memoria principal. Con el fin de que no proliferen muchos huecos pequeños en memoria principal y esta se deteriore, se usa en segmentación las mismas estrategias de colocación que en Multiprogramación con particiones variables y las técnicas de combinación de huecos adyacentes y compactación de memoria.

8.3

Paginación/Segmentación (Segmentación paginada o Paginación segmentada)

Este esquema de organización de la memoria virtual intenta unir las ventajas de los dos esquemas anteriores (paginación y segmentación). Para ello, el Sistema Operativo divide el espacio virtual de direcciones de

un trabajo en segmentos que se gestionan a nivel de páginas; es decir un segmento está compuesto de un número entero de páginas. La memoria principal se divide en marcos de página, por lo que el intercambio de información se hace a nivel de páginas.

Al igual que con los esquemas anteriores, se pueden deducir las siguientes conclusiones, que se recogen también en la figura 8.13:

1. no es necesario que todas las páginas de un segmento estén al mismo tiempo en memoria principal, basta con que figure la página del segmento que en ese momento se esté usando;
2. y además, que las páginas que son contiguas en memoria virtual no tienen por qué ser contiguas en memoria principal.

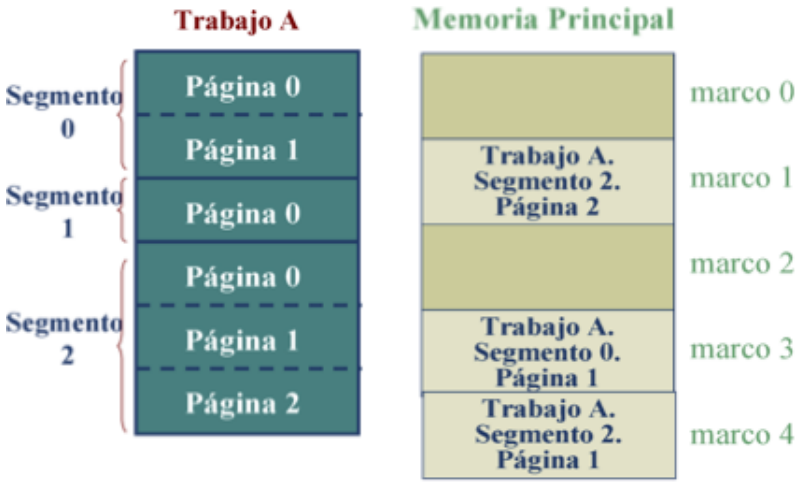


Figure 8.13: Organización del espacio virtual de direcciones y de la memoria principal en Paginación/Segmentación

Cada trabajo tiene asociado una tabla de mapas de segmentos, con tantos elementos como el número de segmentos en los que se ha dividido el trabajo. Además, por cada segmento también existe una tabla de mapas de páginas que tiene tantos elementos como el número de páginas en las que se ha dividido el segmento.

De forma gráfica se muestra en las figuras 8.14 y 8.15.

El formato típico de un elemento de la tabla de segmentos sería:

- t_s : *bit de residencia del segmento en memoria principal* (normalmente $t_s = 1$ indica que al menos una página de ese segmento reside en memoria principal, y $t_s = 0$ indica que ninguna página de ese segmento reside en memoria principal);
- m_s : *dirección* donde está el segmento en el *almacenamiento secundario*;
- s' : *dirección real donde comienza la tabla de páginas* correspondiente a ese segmento;

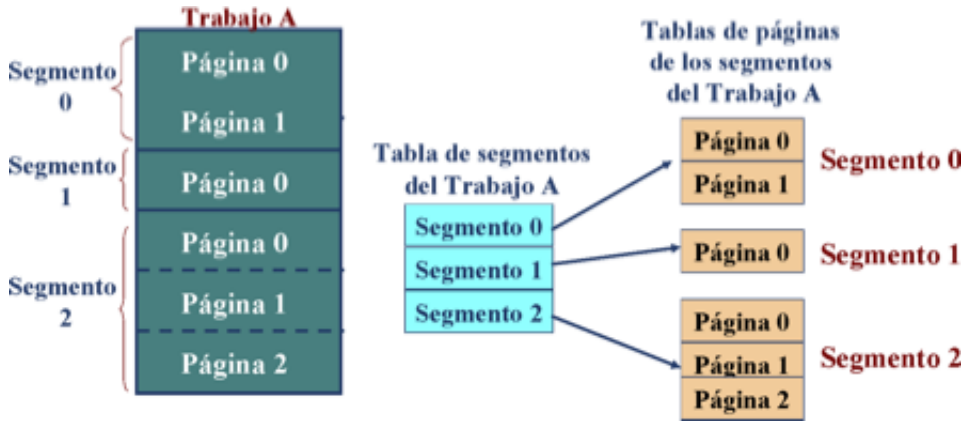


Figure 8.14: Tabla de mapa de segmentos y tablas de mapa de páginas de un trabajo

Figure 8.15: Formato de la tabla de segmentos y de las tablas de páginas de un trabajo

- l : *longitud del segmento* (número de páginas);
- R, W, E, A : *bits de protección* (si vale 1 si posee ese acceso y si vale cero no lo posee)

Por otra parte, el formato típico de un elemento de la tabla de páginas sería:

- t_p *bit de residencia de página en memoria principal* (normalmente t_p = 0 indica que esa página no reside en memoria principal, y t_p = 1 que si reside);
- m_p *dirección* donde está la página en el *almacenamiento secundario*;
- p' *dirección real del comienzo del marco de página o número del marco de página*.

Una dirección virtual en este esquema viene dada por un trío ordenado *dv = (s, p, d)*, donde *s* es el número del segmento dentro del almacenamiento virtual, *p* es el número de página dentro del segmento y *d* es el desplazamiento relativo a la página, es decir; el número de elementos que existen desde el comienzo de esa página hasta el elemento que se está referenciando.

En paginación/segmentación existen las mismas técnicas para realizar la *traducción dinámica de direcciones* que en los dos esquemas anteriores; es decir *por transformación directa*, *por transformación asociativa* o *por combinación de transformación asociativa/directa*. Se describen en las siguientes secciones.

8.3.0.1 Por transformación directa

El Sistema Operativo tiene cargado en el *registro origen* la dirección de memoria principal, *a,* donde comienza la tabla de segmentos del trabajo. Cuando un proceso en ejecución hace referencia a una dirección virtual *dv = (s,p,d)*, los pasos que se siguen para obtener la dirección real son:

1. se añade a la dirección base de la tabla de segmentos, *a,* el número de segmento *s* para formar la dirección real de memoria donde se encuentra la casilla de la tabla de segmentos que contiene la información del segmento *s,*
2. se accede a esa dirección real de memoria para comprobar si al menos alguna página del segmento reside o no en memoria principal (usando el bit de residencia t_s), se comprueba si se desea acceder a una página que pertenece o no al segmento (usando *l*), se controla si la operación solicitada está o no permitida (empleando los bits de protección) y se obtiene la dirección real donde comienza la tabla de páginas para dicho segmento, *s',*
3. se concatena la dirección real donde comienza la tabla de páginas del segmento con el número de página *p*, obteniendo la dirección real de la entrada en la tabla de páginas para dicha página *p* del segmento *s*,
4. se accede a esa dirección real para comprobar si la página reside o no en memoria principal (usando el bit de residencia t_p) y se obtiene la dirección real, *p',* donde comienza el marco de página que contiene a dicha página *p*,
5. se concatena la dirección real donde comienza el marco de página, *p',* con el desplazamiento, *d*, para obtener la dirección real, *dr,* deseada.

Gráficamente sería como se muestra en la figura 8.16.

La dirección virtual y la dirección base de la tabla de segmentos se mantienen en registros de alta velocidad de la CPU. Sin embargo, la tabla de segmentos y las tablas de páginas correspondientes a los distintos segmentos se mantienen completamente en memoria principal. Esto da lugar a que cada acceso a cada una de las citadas tablas requiera un ciclo completo, haciendo que los programas puedan tardar el triple de tiempo en ser ejecutados, ya que la mayoría del tiempo de su ejecución se pierde en el acceso a memoria. En concreto, para cada traducción sería necesario tres ciclos: uno para acceder a la tabla de segmentos, otro para acceder a la tabla de páginas, y otro para acceder al elemento en cuestión.

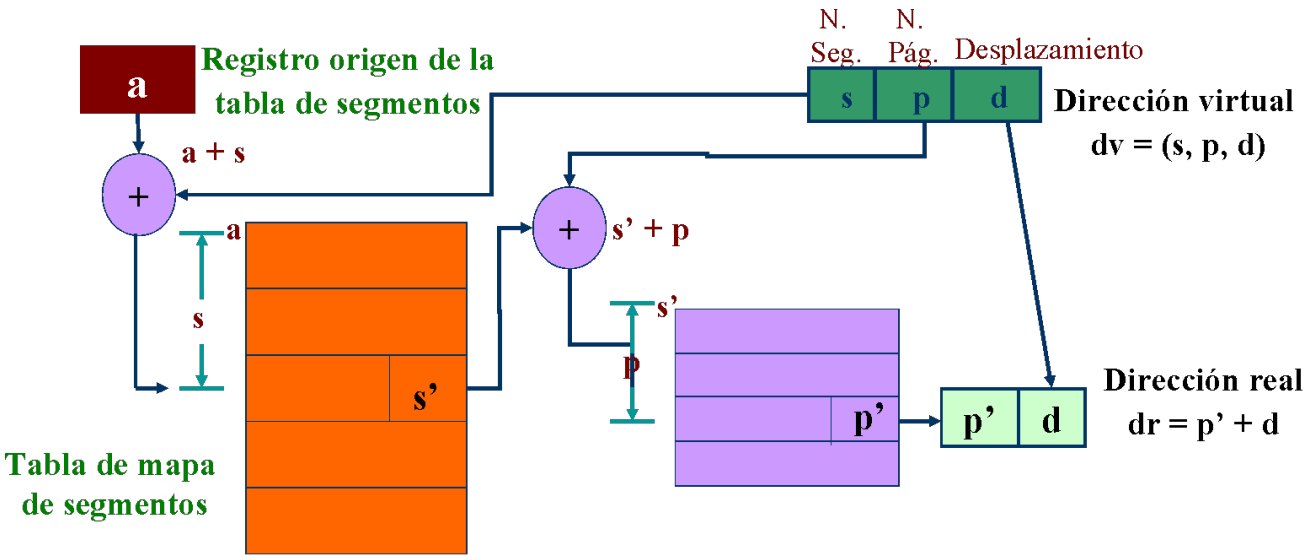


Figure 8.16: Traducción de dirección virtual a dirección real Por Transformación Directa

8.3.0.2 Por transformación asociativa

Al igual que en los esquemas anteriores, se acelera la traducción si se dispone de memoria asociativa para almacenar en ella la unión de la tabla de segmentos con las tablas de páginas correspondientes a esos segmentos. El proceso de traducción sería:

1. se accede simultáneamente a todas las casillas que componen la tabla almacenada en memoria asociativa, para localizar aquella que contiene la información referente a la página *p* perteneciente al segmento *s*, es decir, se busca el par *s, p* en la tabla;
2. de dicha casilla se obtiene el bit de residencia t_p para comprobar si la página *p* del segmento *s* reside o no en memoria principal, también se comprueba, mediante los bits de protección, si la operación solicitada está o no permitida para las páginas del segmento *s* y por último se obtiene la dirección real donde comienza el marco de página, *p'*, que contiene la página *p* del segmento *s*;
3. se concatena la dirección real donde comienza el marco de página con el desplazamiento, *d*, obteniendo la dirección real, *dr,* deseada.

En la figura 8.17 se muestra una representación de este proceso.

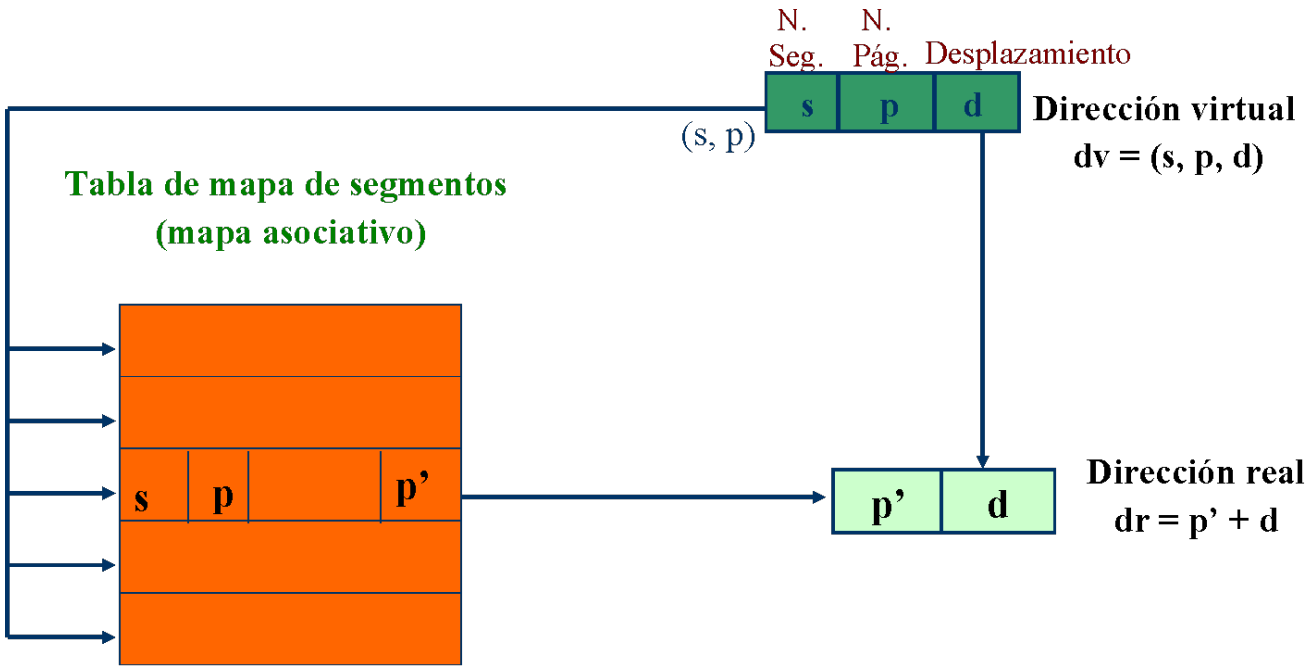


Figure 8.17: Traducción de dirección virtual a dirección real Por Transformación Asociativa

8.3.0.3 Por combinación de transformación asociativa y directa

En el almacenamiento asociativo disponible se encontrará la información referente a las páginas que han sido referenciadas más recientemente, y en memoria principal se mantendrán, de forma completa, tanto la tabla de segmentos como las tablas de páginas correspondientes a esos segmentos.

1. primero, se intenta encontrar el par *s, p* en el trozo de tabla contenida en memoria asociativa, accediendo simultáneamente a todas las casillas que componen dicha memoria asociativa. Es decir, se pretende localizar aquella casilla que contiene la información referente a la página *p* del segmento *s*,
2. si se localiza, se obtiene de dicha casilla el bit de residencia t_p para comprobar si la página *p* del segmento *s* reside o no en memoria principal, también se comprueba, mediante los bits de protección, si la operación solicitada está o no permitida para las páginas del segmento *s* y por último se obtiene la dirección real donde comienza el marco de página, *p'*, que contiene la página *p* del segmento *s,* y se concatena con el desplazamiento, *d*, para obtener la dirección real *dr* deseada.
3. si no se localiza, entonces se utilizan la tabla de segmentos y las tablas de páginas almacenadas en memoria principal. Primero, se añade a la dirección base de la tabla de segmentos, *a,* el número de segmento, *s,* para formar la dirección real de memoria donde se encuentra la casilla dentro de dicha tabla. Accediendo a esa casilla se comprueba si al menos alguna página del segmento reside o no en memoria principal (usando el bit de residencia t_s), se controla si se desea acceder a una página que pertenece o no al segmento (usando *l*), se determina si la operación solicitada está o no permitida (empleando los bits

de protección) y se obtiene la dirección real donde comienza la tabla de páginas para dicho segmento, *s'*. A esta dirección se le concatena el número de página *p*, para obtener la dirección real de la entrada en la tabla de páginas para dicha página *p* del segmento *s*. Luego se accede a esa dirección real para comprobar si la página reside o no en memoria principal (usando el bit de residencia t_p) y se obtiene la dirección real, *p'*, donde comienza el marco de página que contiene a dicha página *p*. Y por último, se concatena esta dirección real con el desplazamiento, *d*, obteniendo la dirección real, *dr*, deseada.

Gráficamente sería como se muestra en la figura 8.18:

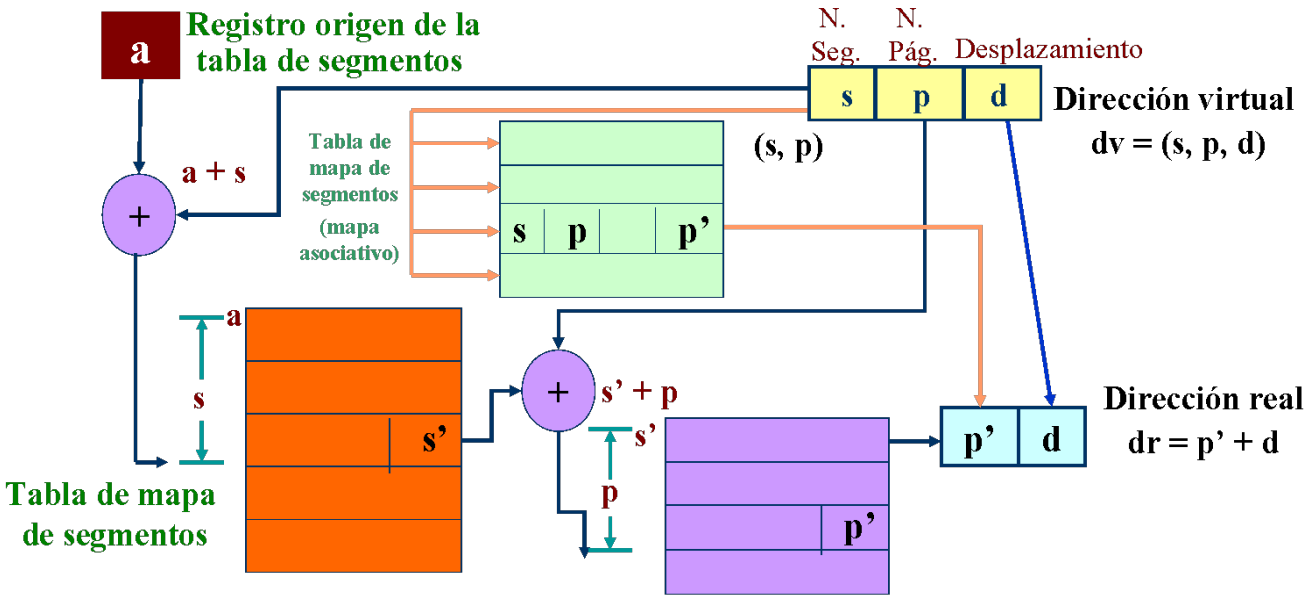


Figure 8.18: Traducción de dirección virtual a dirección real Por Combinación de la Transformación Asociativa y la directa

Con cualquiera de las tres técnicas de traducción descritas, puede producirse los siguientes fallos:

- *Fallo de pérdida de segmento*. Se produce cuando ninguna de las páginas del segmento está en memoria principal. Este fallo se detecta comprobando el bit de residencia, t_s, de la tabla de segmentos y se soluciona creando la tabla de páginas para dicho segmento y cargando la página referenciada en memoria principal. Es probable que exista en memoria principal otra página del trabajo, pero perteneciente a otro segmento.
- *Fallo de pérdida de página*. Alguna de las páginas del segmento al que pertenece la página referenciada está en memoria principal, pero la página en cuestión no. Esto se detecta comprobando el bit de residencia, t_p, de la tabla de páginas para el citado segmento y se soluciona cargando dicha página en algún marco de página disponible desde el almacenamiento secundario.
- *Fallo de desbordamiento de segmento*. Se produce cuando se hace referencia a una página que no posee el

segmento, y por lo tanto se detecta al realizar la verificación $p \leq l$, donde p es el número de la página a la que se hace referencia y l es el tamaño del segmento (número de páginas que comprende el segmento). Si no se cumple esta comprobación se determina que se ha producido este fallo y se aborta la ejecución del proceso.

- *Fallo de protección de segmento*. Se produce cuando se solicita una operación que no está permitida. Este fallo se detecta comprobando los bits de protección, y si ocurre se cancela la ejecución del proceso.

Como se observó al analizar la segmentación, los segmentos son más fáciles de compartir que las páginas, por lo tanto, en paginación/segmentación se van a compartir segmentos. Un segmento es compartido por dos o más procesos, si en sus tablas de mapa de segmentos hacen referencia a la misma tabla de mapa de páginas. En definitiva, todas las páginas pertenecientes a ese segmento van a ser compartidas, tal y como se muestra en la figura 8.19.

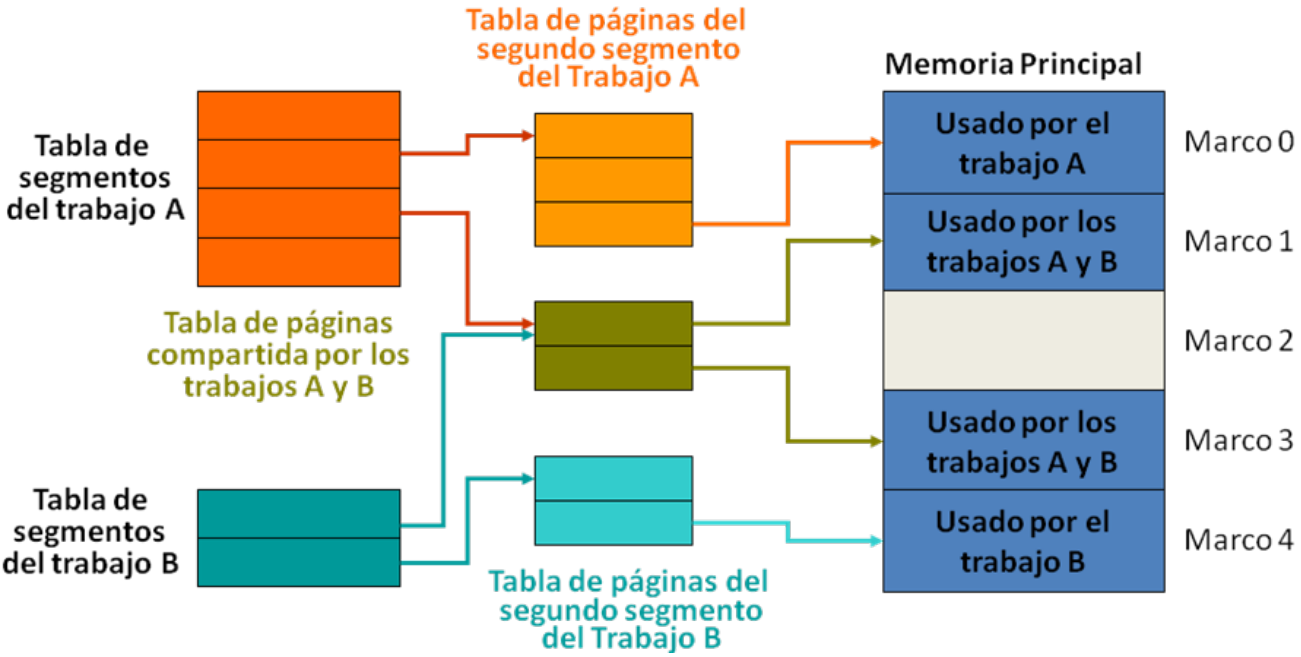


Figure 8.19: Compartición de segmentos

Las tablas necesarias para este esquema de organización de la memoria virtual consumen una parte considerable de memoria principal, dando lugar a que disminuya el número de trabajos que pueden mantenerse simultáneamente. Se denomina *fragmentación de tablas* a este espacio desperdiciado de memoria principal, es decir a la memoria principal usada para mantener las tablas requeridas por cada sistema.

En paginación/segmentación también se da *fragmentación interna*, ya que la probabilidad de que el tamaño de un segmento sea múltiplo entero del tamaño máximo de página (tamaño de marco de página) es muy pequeña, por lo tanto, la última página de cada segmento tendrá casi siempre un tamaño menor que el máximo

citado. Cuando esta última página se copie en un marco de página de la memoria principal no ocupará todo ese marco, por lo que el trozo de memoria principal perteneciente a ese marco de página, pero no usado por la página lógica se desperdicia.

IX

Simulador 1: Paginación

9.1

Objetivo

El objetivo principal de este simulador es mostrar gráficamente y de una manera didáctica cómo se organiza la memoria principal y los trabajos y, cómo, partiendo de una dirección virtual, ésta se traduce a dirección real. Para ello, este simulador se desarrolló como una aplicación web didáctica, la cual es independiente del sistema operativo y que recoge una serie de funcionalidades para lograr el objetivo.

Entre estas funcionalidades están:

- **Gestión de Memoria**
  - Describir a través de un conjunto de datos de entrada, la memoria principal y los trabajos. Específicamente se han usado cinco conjuntos diferentes de datos para describir la MP (Memoria Principal) y la MV (Memoria Virtual) usando diversas unidades seleccionables por el usuario. La descripción de la MP y MV deberá permitir introducir múltiples tipos de entrada de datos entre los que se encuentran:
    1. Tamaño de la MP, el tamaño del marco de página (o lo que es lo mismo, el tamaño máximo de la página lógica) y el número de bits empleadas para especificar la dirección virtual o para especificar la página dentro de la dirección virtual.
    2. Número de marcos de página, el tamaño del marco de página (o lo que es lo mismo, el tamaño máximo de la página lógica) y el número de bits empleadas para especificar la dirección virtual o para especificar la página dentro de la dirección virtual.
    3. Número de bits utilizados para especificar la dirección real de una posición de memoria, el tamaño del marco de página (o lo que es lo mismo, el tamaño máximo de la página lógica) el número de bits empleadas para especificar la dirección virtual o para especificar la página dentro de la dirección virtual
    4. Número de bits requeridos para una (dv) y cuántos de ellos se usan para indicar el número de página (p) o para el desplazamiento (d) y además también el número de marcos de página.
    5. Exactamente los mismos parámetros que la opción anterior salvo que en lugar del número

de marcos de página se permitirá introducir el tamaño de la MP.

- Permitir crear hasta dos trabajos. Se seleccionó el número de trabajos con el objetivo de que el alumnado tuviese todos los elementos necesarios para la mejor comprensión del tema. Para definir un trabajo concreto en la simulación se permitirá introducir el tamaño del trabajo. Se podrá una vez creada la MP y creados los trabajos, cargar una página de un trabajo que todavía no esté copiada en memoria principal en un marco de página libre elegido por el usuario. Por cada trabajo existe una tabla de páginas que el usuario podrá consultar en cualquier momento que desee.
- Ofrecer una descripción de la MP, de la MV y de los trabajos con datos calculados a partir de los aportados como entrada.
- Determinar la fragmentación de tablas del correspondiente trabajo. Para la fragmentación de tablas, el usuario deberá introducir el número de bits utilizados para indicar la dirección de memoria secundaria donde está esa página.
- Determinar la fragmentación interna que se produce cuando se carga en MP la última página de un trabajo.

- **Traducción de Memoria**
  - Traducir una dirección virtual en hexadecimal dada como dato de entrada a una dirección real, explicando de forma detallada cada paso en el proceso de traducción. En el caso de que se produzca un fallo de perdida de página, se explica detalladamente como se ha deducido ese fallo. Para la traducción de dirección virtual a dirección real, el usuario deberá indicar la dirección virtual a traducir (en hexadecimal). Se obtendrá la dirección real, explicando detalladamente los correspondientes pasos seguidos para la obtención de ésta, mostrando en todo caso el error que se pudo originar en caso de no poder producirse ésta.

9.2 Arquitectura y Tecnologías

El proyecto siguió el modelo-vista-controlador (`MVC`), tal como ya se ha descrito previamente en la figura 5.2 de la sección I, Planificación de Procesos.

En este simulador el usuario interactúa a través de la vista. La vista constituye la interfaz de la aplicación, y está implementada en archivos `JSP` que contiene código `HTML` y `Java`, además de referencias a archivos `CSS` y `JavaScript`. La vista es la encargada de recibir datos del modelo y mostrarlos al usuario. El usuario usa el controlador a través de una llamada a una acción que es llamada por la vista. Esta acción la realiza el controlador, que está compuesto por los `Servlets` (tecnología `Java` web) que contienen las acciones

realizadas sobre los objetos y que fueron implementada usando Java.

Por último, el controlador para realizar la acción manipula datos del modelo y también la vista usa el modelo para obtener datos y mostrarlos en pantalla. El modelo viene dado por todas las clases, las cuales representan los objetos que son utilizados en nuestro proyecto, y que son implementadas utilizando el lenguaje de programación Java.

En la figura 9.1 se recoge la arquitectura general de este simulador.

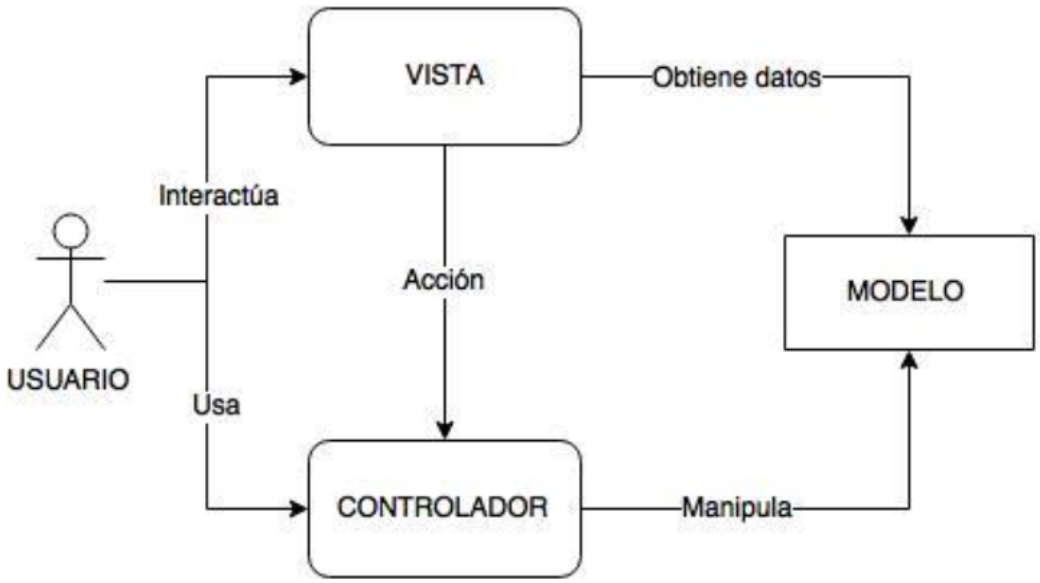


Figure 9.1: Detalle de la Arquitectura

Las vistas fueron implementadas utilizando archivos JSP. JavaServer Page o JSP es una tecnología utilizada para la creación de páginas web basadas en HTML y XML entre otras. Los controladores se implementan con Servlets, que definen clases Java que atienden a peticiones del usuario para realizar las acciones convenientes. El modelo se implementa utilizando tecnología Java. Cada clase del modelo se corresponderá con una clase Java. En la figura 9.2, se muestra un detalle de esta implementación.

Se incluye a continuación, en la figura 9.3 el modelo de dominio que permite reflejar en un diagrama, la estructura estática del sistema en función de sus clases, con sus atributos y las distintas relaciones que existen entre ellos.

Así mismo, se incluye el diagrama de clases, en la figura 9.4 que ofrece una representación visual de la estructura estática de un sistema orientado a objetos, mostrando las clases, sus atributos, métodos y las relaciones entre ellas, como herencia, asociación y composición. Facilita la comprensión del diseño del sistema al ilustrar cómo interactúan los componentes internos [1].

[1]Por motivos de espacio, se han eliminado atributos y métodos de esta representación

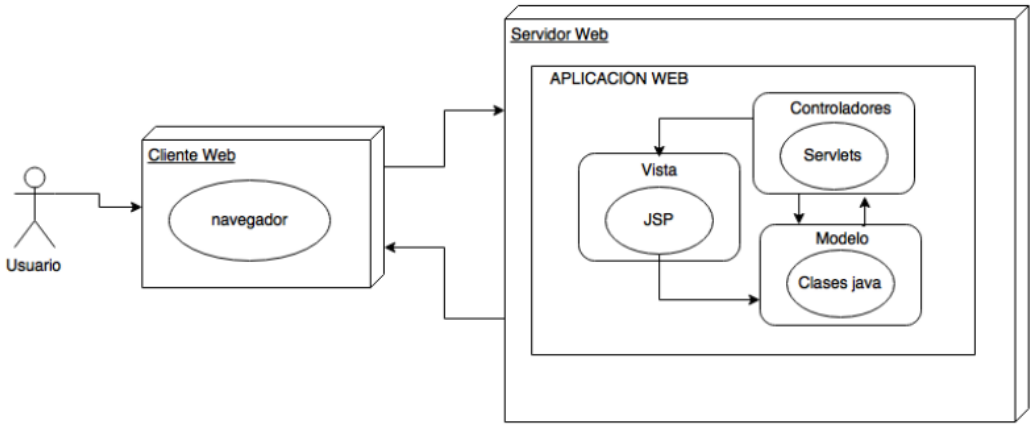


Figure 9.2: Diagrama de Despliegue

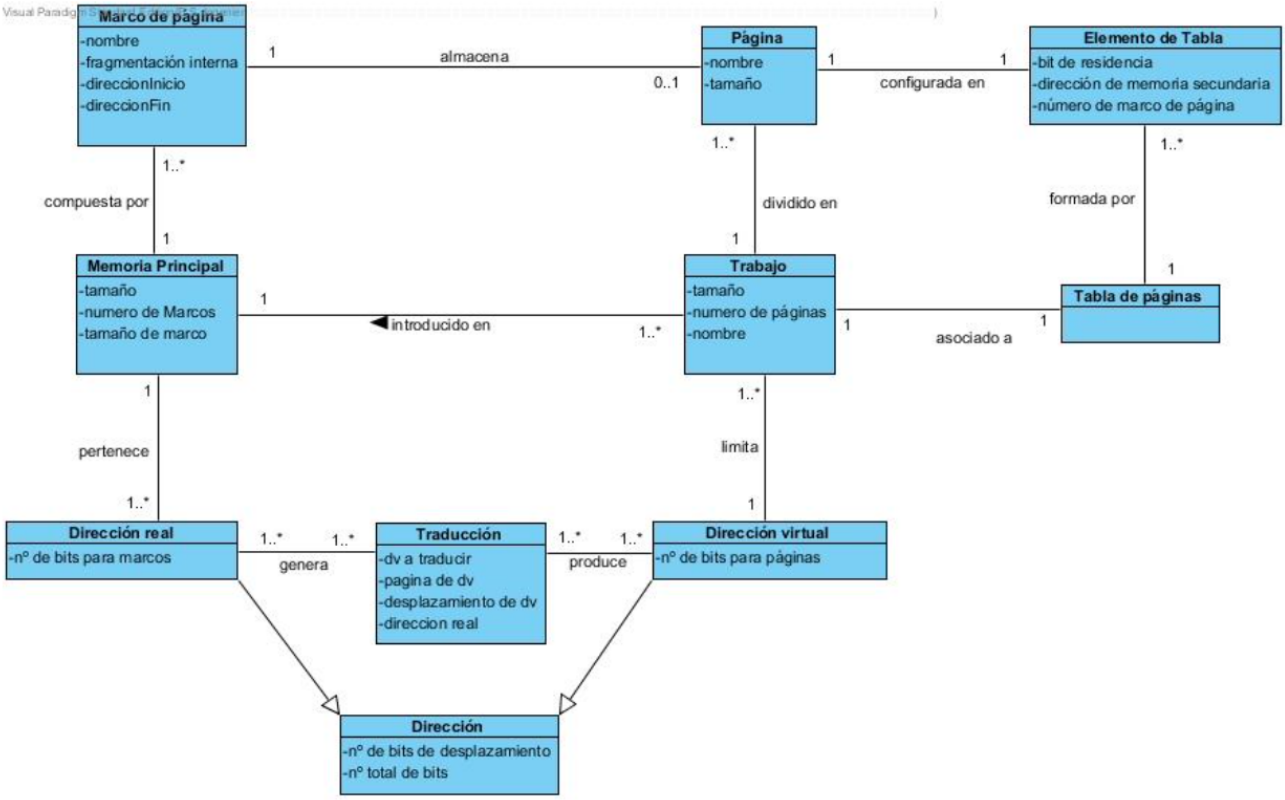


Figure 9.3: Modelo de Dominio

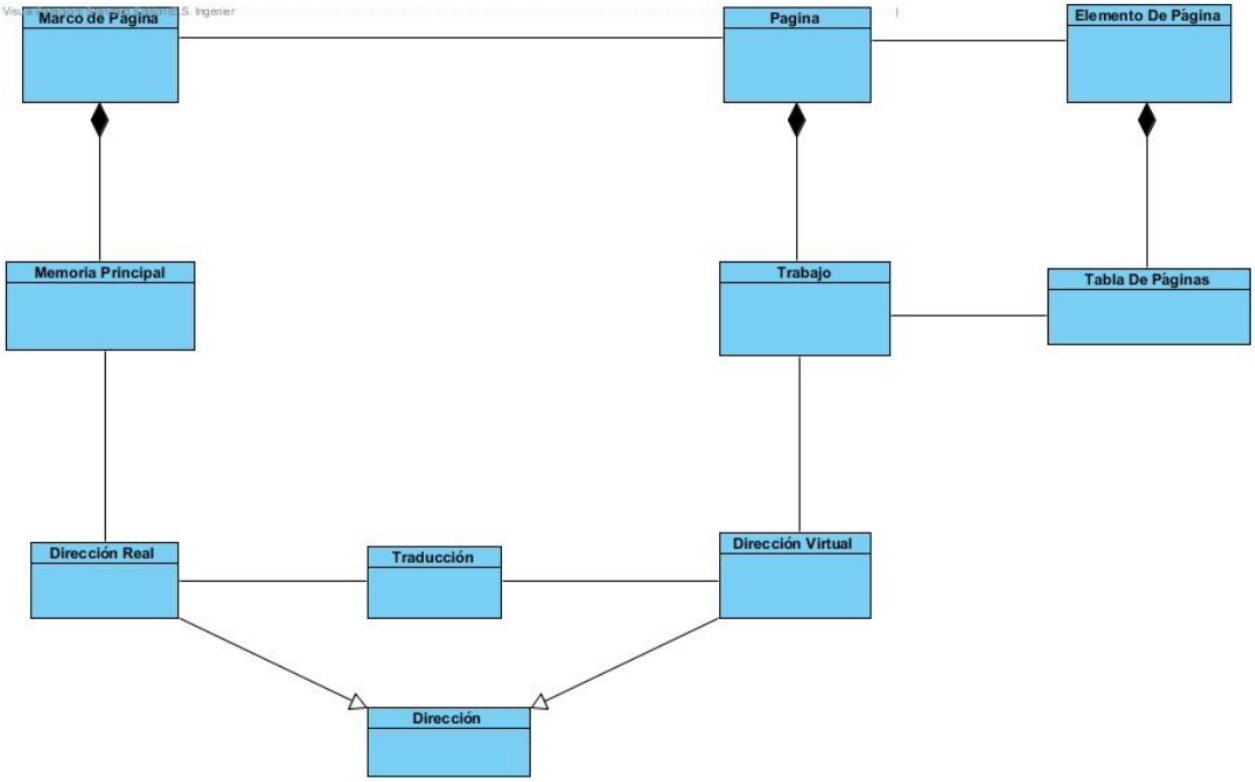


Figure 9.4: Diagrama de Clases

9.3

Uso del Simulador

En la siguiente dirección `http://193.147.87.86:8080/TFGSimuladorPaginacion/`, está disponible el simulador. En el centro de la página principal, figura 9.5, a través de una serie de diapositivas informativas de transición horizontal automática se describe gráficamente:

- la estrategia de paginación,
- las funcionalidades más importantes del simulador.

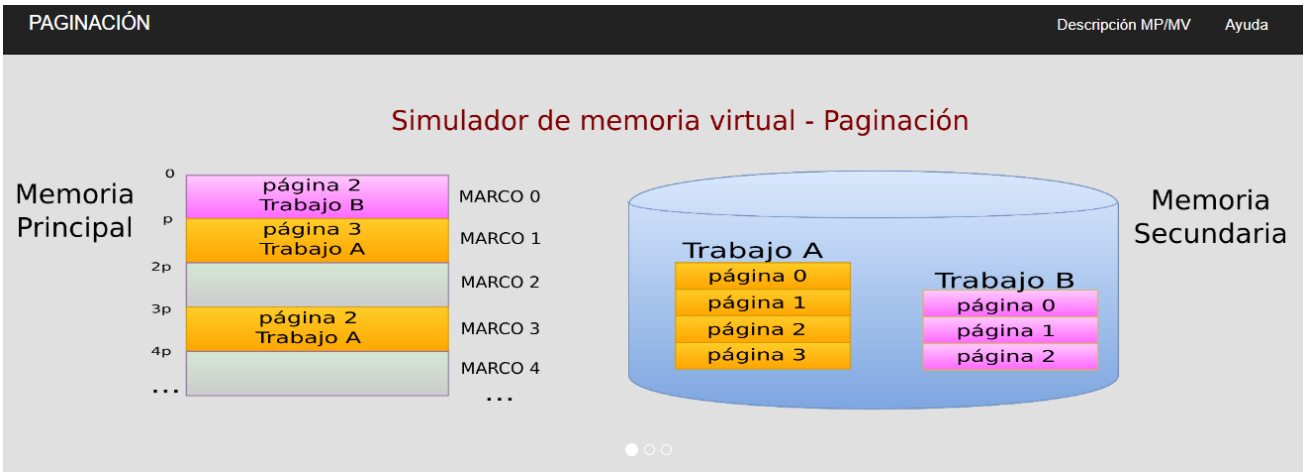


Figure 9.5: Página principal

La opción "Descripción MP/MV" abre un menú, figura 9.6, con las cinco opciones disponibles para configurar tanto la memoria principal como la memoria virtual. Estas opciones son las siguientes:

A. Se introduce el tamaño de la memoria principal, el tamaño máximo de página y el número de bits empleados para indicar el número de página o los utilizados para especificar una dirección virtual.
B. Se introduce el número de marcos de página, el tamaño máximo de página y el número de bits empleados para indicar el número de página o los utilizados para especificar una dirección virtual.
C. Se introduce el número de bits utilizados para especificar la dirección real, el tamaño máximo de página y el número de bits empleados para indicar el número de página o los utilizados para especificar una dirección virtual.
D. Se introduce el número de bits utilizados para especificar una dirección virtual, el número de bits empleados para indicar la página o el desplazamiento y el número de marcos de página en los que se divide la memoria principal.
E. Se introduce el número de bits empleados para especificar una dirección virtual, el número de bits

empleados para indicar la página o el desplazamiento y el tamaño de la memoria principal.

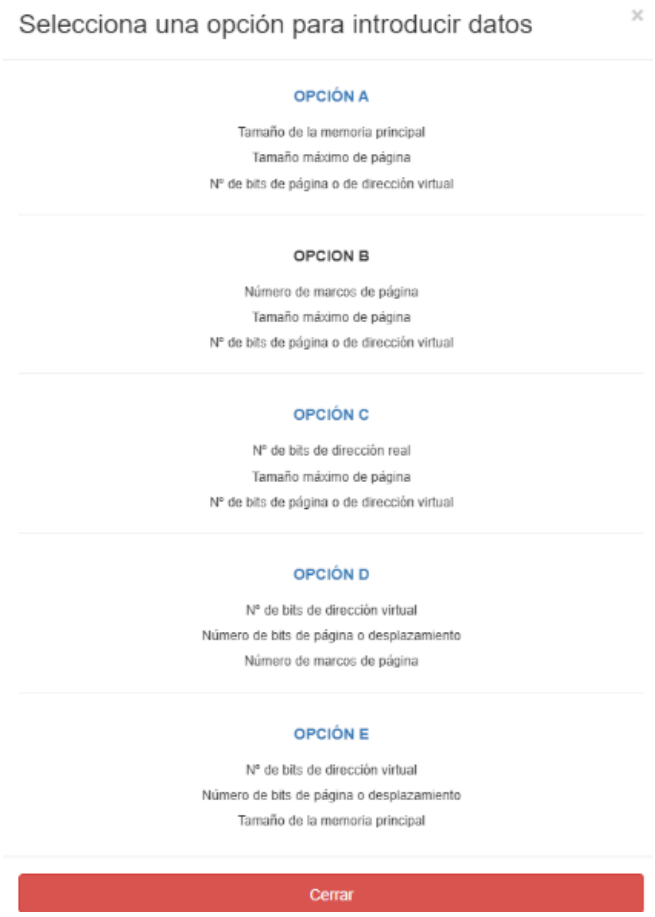


Figure 9.6: Opciones para describir MP/MV

Una vez descrita la memoria principal y la memoria virtual se muestran los marcos de pagina libres en los que se divide la memoria principal, tal como se ve en la figura 9.7. Al lado de cada marco de página aparece la dirección real de inicio y la dirección real final de cada uno de ellos.

Figure 9.7: Memoria Principal

Crear Trabajo

La opción de "Crear Trabajo" permite introducir los datos con los que se describe un trabajo: tamaño y color que lo identifica. Una vez creado se mostrará el trabajo (Figura 9.8) dividido en páginas del mismo tamaño que el del marco de página salvo la última página que puede tener un tamaño inferior dependiendo del tamaño dado al trabajo.

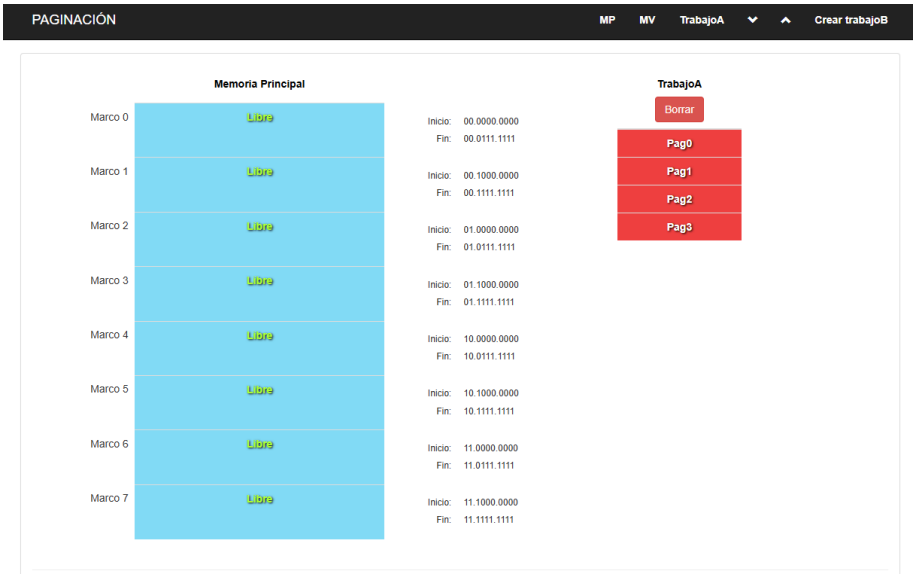


Figure 9.8: Trabajo creado

Cargar página en marco de página libre

La opción "Cargar" permite copiar una página del trabajo en un marco de página libre de la memoria principal. Una vez introducidos ambos datos, se actualiza gráficamente la ocupación de los marcos de página de la memoria principal. Esto se puede ver reflejado en la figura 9.9 donde se carga la segunda página del trabajo en el cuarto marco de página.

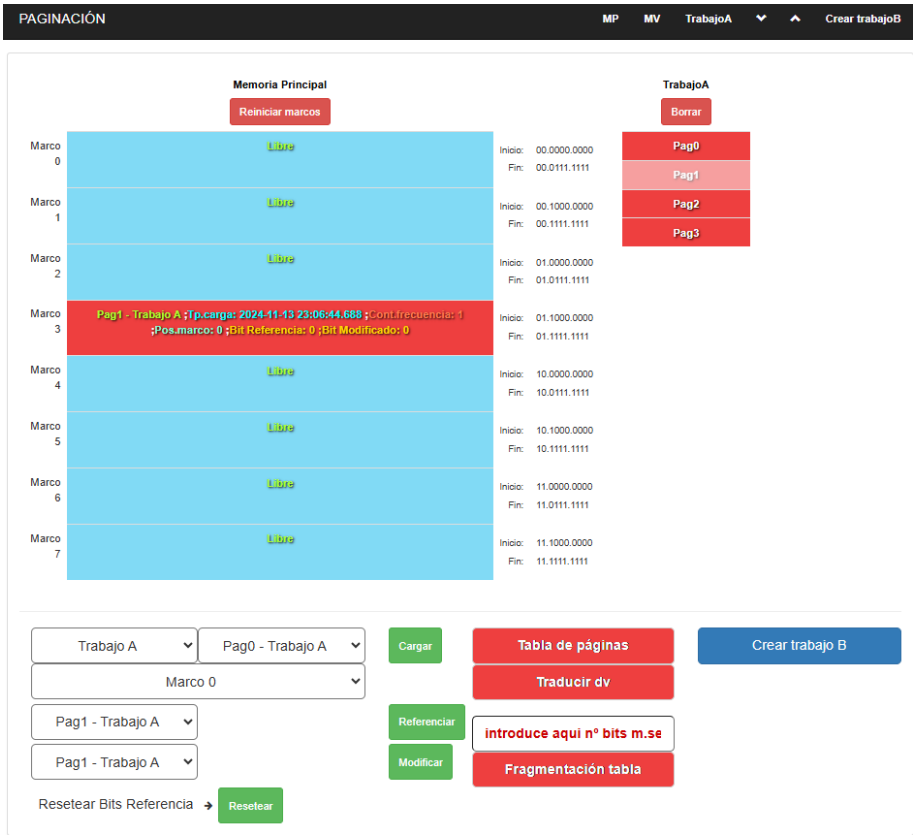


Figure 9.9: Carga de página en Memoria Principal

Consultar tabla de mapa de páginas

En Paginación, todo trabajo tiene una tabla de páginas asociada. La aplicación permite consultar el contenido actual de dicha tabla mediante la opción "Tabla de páginas". Cada fila de dicha tabla proporciona información de una página, de forma que la fila 0 corresponde a la página 0 del trabajo, la fila 1 a la página 1 y así sucesivamente. Si la página esta cargada en memoria principal, entonces p' indica el marco de página que la contiene. En la figura 9.10 se puede comprobar que el marco 3 contiene a la página 1 del trabajo.

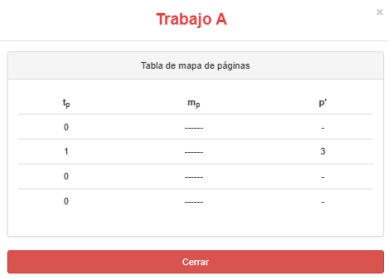


Figure 9.10: Tabla de mapa de páginas

Fragmentación de tablas

El simulador también proporciona la posibilidad de calcular la fragmentación de tablas, es decir la cantidad de memoria principal ocupada por la tabla de páginas.

A la hora de seleccionar esta opción, se solicita al usuario la introducción del número de bits necesarios para especificar la dirección en memoria secundaria donde reside la página.

Traducción de una dirección virtual en una dirección real

Por último, también se puede solicitar la traducción de una dirección virtual a una dirección real mediante la opción "Traducir *dv* a *dr*".

Una vez que se introduce en hexadecimal la dirección virtual, se muestran los pasos a seguir para realizar dicha traducción (Figura 9.11).

Figure 9.11: Traducción de dirección virtual a dirección real

El proceso de traducción de dirección virtual a dirección real comprende los siguientes pasos:

1. Se traduce la dirección virtual de hexadecimal a binario para identificar las dos componentes de una dirección virtual, es decir el número de página (p) donde está el elemento al que se referencia y el desplazamiento (d) que es la distancia que dista desde el comienzo de la página al elemento en cuestión.

2. Se comprueba si la página reside en memoria principal. Para ello, se determina la fila de la tabla de páginas que proporciona la información sobre la página en cuestión. De dicha fila se comprueba el valor del bit de residencia t_p. Si este bit indica que la página no reside en memoria principal, se habrá producido un *Fallo de pérdida de página*.
3. Si la página reside en memoria principal, se utiliza el número de marco de página (p') para obtener la dirección real donde comienza el marco de página que contiene a esa página.
4. Por último, se obtiene la dirección real deseada concatenando la dirección real donde comienza el marco de página calculada en el paso anterior con el desplazamiento. Esta dirección real se pasa de binario a hexadecimal.

9.4

Generación automática de ejercicios con IA generativa

Usando herramientas de inteligencia artificial generativa se han obtenido las soluciones a una relación de ejercicios que se ha elaborado incrementando su complejidad de forma progresiva sobre aspectos relacionados con la Paginación. El comienzo de ésta relación contiene el siguiente enunciado común al conjunto de ejercicios.

> Supongamos un sistema paginado cuyo tamaño máximo de página lógica es de 1 Kibibyte y donde el tamaño de cada posición de memoria es de 1 byte

A continuación se muestran tanto los prompts como las respuestas obtenidas por `ChatGPT`:

a. Número de marcos de página

La memoria principal es de 1 Mebibyte, mientras que el tamaño de cada marco de página es de 1 Kibibyte. Para calcular el número de marcos:

$$\text{Número de marcos} = \frac{\text{Tamaño de la memoria principal}}{\text{Tamaño de un marco}}$$

$$= \frac{2^{20}\ \text{bytes}}{2^{10}\ \text{bytes/marco}} = 2^{10} = 1024\ \text{marcos}$$

Respuesta: La memoria principal se dividirá en 1024 marcos de página.

b. Bits necesarios para especificar la dirección real

Para representar una dirección real en una memoria de 1 Mebibyte (2^{20} bytes):

$$\text{Número de bits} = \log_2(2^{20}) = 20 \text{ bits}$$

Respuesta: Se necesitarán 20 bits para especificar la dirección real de una posición de memoria.

c. ¿Cuál será la dirección real donde comienza el segundo marco de página (marco 1) (expresada en binario)?, ¿y la dirección real donde termina el décimo marco de página (marco 9) (expresada en binario)?

Solución:

- Cada marco tiene un tamaño de 1 Kibibyte (2^{10} bytes).

1. **Dirección inicial del segundo marco de página (marco 1):**
   - La dirección inicial del primer marco es 0, por lo tanto, la dirección inicial del segundo marco será:

     $$2^{10} = 1024 \text{ en decimal}$$

     Convertimos 1024 en binario usando 20 bits:

     $$00000000010000000000$$
2. **Dirección final del décimo marco de página (marco 9):**
   - La dirección inicial del décimo marco es $9 \times 1024 = 9216$ en decimal.
   - La dirección final del décimo marco será la posición inmediatamente anterior a la dirección de inicio del siguiente marco (el marco 10), es decir, $10240 - 1 = 10239$.

     Convertimos 10239 en binario usando 20 bits:

     $$00000010011111111111$$

Respuesta:

- La dirección real donde comienza el segundo marco de página (marco 1) es **00000000010000000000** en binario.
- La dirección real donde termina el décimo marco de página (marco 9) es **00000010011111111111** en binario.

d. Si se usa 6 bits para indicar el número de página, p, dentro de una dirección virtual, ¿cuál sería el tamaño máximo del trabajo que puede aceptar este Sistema Operativo?

Solución:

- Si se usan 6 bits para el número de página, el máximo número de páginas posibles es $2^6 = 64$.
- Como cada página tiene un tamaño de 1 Kibibyte (1024 bytes), el tamaño máximo del trabajo es:

$$64 \times 1024 = 65536 \text{ bytes} = 64 \text{ Kibibytes}$$

Respuesta: El tamaño máximo del trabajo que puede aceptar el Sistema Operativo es de **64 Kibibytes**.

e. Número de páginas para un trabajo de 8,25 KiB

Un trabajo de 8,25 KiB se divide en páginas de 1 KiB, por lo que el número de páginas necesarias es:

$$\lceil \frac{8.25 \text{ KiB}}{1 \text{ KiB}} \rceil = \lceil 8.25 \rceil = 9 \text{ páginas}$$

f. Fragmentación interna y externa

- **Fragmentación interna**: ocurre cuando la última página no está completamente ocupada. La última página tiene un tamaño de 1 Kibibyte, y el trabajo usa 0,25 Kibibytes de esta página.

$$\text{Fragmentación interna} = 1 \text{ KiB} - 0.25 \text{ KiB} = 0.75 \text{ KiB}$$

- **Fragmentación externa**: en sistemas de paginación pura, no hay fragmentación externa, ya que cada marco de página puede asignarse a diferentes trabajos sin dejar "huecos".

Respuesta:

- Fragmentación interna: 0.75 KiB.

g. Fragmentación de tablas de páginas

Cada entrada de la tabla de páginas tiene tres campos:

- **Bit de residencia**: asumimos que ocupa 1 bit.
- **Dirección en memoria secundaria**: ocupa 10 bits.
- **Dirección real del marco**: en este caso, como la memoria tiene 20 bits de dirección, este campo ocuparía 20 bits.

Entonces, cada entrada ocupa:

$$1 + 10 + 20 = 31 \text{ bits}$$

Para 9 páginas, el total de fragmentación de la tabla de páginas sería:

$$9 \times 31 = 279 \text{ bits}$$

h. Teniendo en cuenta que una dirección virtual viene dada por 16 bits, de los cuales los 10 últimos son el desplazamiento y los 6 bits primeros el número de página. El tamaño de la memoria principal es 1 Mebibyte y el tamaño del marco 1 Kibibyte.

Suponiendo que la tabla de páginas asociada a un trabajo es la siguiente:

tp	mp	p'
0	---	2
1	---	0
1	---	7

- tp es el bit de residencia (0 no reside, 1 sí reside).
- mp es la dirección en memoria secundaria.
- p' es el número del marco de página expresado en decimal.

Suponiendo que la dirección virtual es de 16 bits, traducir las siguientes direcciones virtuales expresadas en hexadecimal a direcciones físicas expresadas también en hexadecimal.

En el caso de que durante alguna de las traducciones se produjese un fallo, se debe explicar con claridad dicho fallo.

Dirección a traducir: 080F (en hexadecimal)

2. **Conversión de la dirección virtual (080F en hexadecimal) a binario:**
   - $080F_{16} = 0000\ 1000\ 0000\ 1111_2$ (16 bits)
     - Los **6 primeros bits** (número de página): $000010_2 = 2$ (número de página 2).
     - Los **10 últimos bits** (desplazamiento): $0000001111_2 = 0x00F$ (desplazamiento 0x00F).
3. **Consulta de la tabla de páginas:**
   - El número de página es **2**, por lo que se debe consultar la tabla de páginas para la página 2.
   - En la tabla de páginas:
     - Para la página 2, el bit de residencia **tp** es **1** (lo que significa que **sí reside en memoria**).
     - El número de marco **p'** es **7**, por lo que la página 2 se encuentra en el marco de memoria 7.
4. **Cálculo de la dirección física:**

   La dirección física se obtiene al combinar el número de marco con el desplazamiento:
   - **Número de marco:** 7 (de la tabla de páginas).
   - **Desplazamiento:** 0x00F (de la dirección virtual).

   La dirección física será:

$$\text{Dirección física} = (\text{Número de marco} \times \text{Tamaño del marco}) + \text{Desplazamiento}$$

$$\text{Dirección física} = (7 \times 1024) + 0x00F = 7168 + 0x00F = 7168 + 15 = 7183$$

   Ahora, convertimos 7183 a hexadecimal:

$$7183_{10} = 1C0F_{16}$$

Resultado final:

La dirección virtual $080F_{16}$ se traduce a la dirección física **1C0F**.

Conclusión:

La dirección virtual $080F_{16}$ se traduce correctamente a la dirección física $1C0F_{16}$ sin que se produzca ningún fallo de página, ya que la página 2 está cargada en memoria según la tabla de páginas proporcionada.

Dirección virtual: 0005H

1. **Convertimos a binario:**
   - **0005**H = **0000 0000 0000 0101** en binario (16 bits).
2. **Dividimos la dirección:**
   - Los **6 primeros bits** son el **número de página:** **000000** (en decimal, **0**).
   - Los **10 últimos bits** son el **desplazamiento:** **0000000101** (en decimal, **5**).
3. **Consultamos la tabla de páginas:**
   - La página **0** no reside en memoria principal (**tp = 0**).
   - Según la tabla, esto implica un **"fallo de pérdida de página"**.

Conclusión

Como la página **0** no reside en memoria principal, ocurre un **"fallo de pérdida de página"**. Esto significa que el sistema debe cargar la página desde la memoria secundaria a la memoria principal antes de poder traducir la dirección a una dirección física.

Resultado: No es posible realizar la traducción directa de la dirección virtual **0005**H a una dirección física sin antes manejar el **fallo de pérdida de página**.

X

Simulador 2: Segmentación

10.1

Objetivo

El objetivo principal de este simulador es tener una herramienta software que simule, de una forma didáctica, el esquema de gestión de memoria virtual conocida como segmentación. Este simulador se realiza como una aplicación web, la cual debe tratar la simulación de la técnica de segmentación.

Se realiza como aplicación web facilitar el acceso al simulador desde cualquier lugar, como puede ser la universidad o en sus hogares, sin necesidad de instalar software adicional en el equipo.

En este simulador la memoria principal se describe a través de unos parámetros de entrada introducidos por el usuario. Una vez creada se permitirá la creación de diferentes trabajos. Si hay trabajos creados se podrán cargar sus segmentos en la memoria principal, pudiendo observar como residen y como se originan los nuevos huecos en ésta. También se permite en todo momento consultar los datos que describen la memoria principal (tamaño, fragmentación externa, tamaño ocupado, etc.), la memoria virtual (bits de dirección virtual, tamaño máximo de trabajo, bits de desplazamiento, etc.) y los distintos trabajos creados (tamaño, número de segmentos que residen en MP, etc.).

Las funcionalidades más importantes del simulador son:

- *Cargar segmento de trabajo en memoria principal:* hay dos métodos para cargar el segmento en memoria principal.
    1. El primer método consiste en seleccionar el hueco en el que se va a cargar el segmento. Si el tamaño del segmento es menor o igual que el del hueco se cargará.
    2. El segundo método consiste en indicar la dirección real en la que se desea que comience el segmento. Dicho segmento residirá en memoria principal si la dirección real dada coincide con un hueco y además el hueco tiene mayor o igual tamaño que el segmento seleccionado.
- *Descargar segmento de memoria principal:* se elimina un segmento que reside en memoria principal, generándose en ésta un nuevo hueco del tamaño del segmento que fue eliminado. Si el segmento eliminado era adyacente a un hueco, el hueco nuevo que se genera es una combinación del hueco adyacente con el hueco que se genera al eliminar el segmento.

- *Traducir dirección virtual a dirección real:* el usuario introduce una dirección virtual a partir de la cual se obtiene una dirección real, siempre y cuando no se produzca alguno de los posibles fallos al realizar dicha traducción. Para obtener esta dirección real se siguen los pasos descritos en el apartado 8.2.
- *Fragmentación de tablas:* se obtiene el espacio utilizado en memoria principal para mantener la tabla de segmentos correspondiente a un trabajo determinado.

10.2

Arquitectura y Tecnologías

La arquitectura del proyecto se basa en el modelo cliente-servidor. En este tipo de arquitectura la funcionalidad del sistema se organiza en servicios, y cada servicio es entregado por un servidor. Los clientes son usuarios de dichos servicios. Dicha arquitectura se centra en la organización del modelo en tres capas, que consta de una capa de presentación, una capa de procesamiento de la aplicación y la capa de administración de datos. La capa de presentación es la encargada de presentar la información al cliente y gestionar todas las interacciones con éste. La capa de procesamiento de la aplicación se ocupa de gestionar la lógica de la aplicación y proporciona la funcionalidad requerida a los usuarios finales. La capa de administración de datos se refiere a la administración de los diferentes datos que existen en el proyecto, en este proyecto no se gestiona sobre una base de datos, si no que se almacenan como variables de sesión.

Este modelo fue el escogido para el proyecto debido a dos ventajas frente a otros modelos.

- Una de las ventajas para este proyecto es que la capa de presentación no tiene que ser enviada por el servidor a través de la red, ya que puede implementarse en el lado del cliente. Para esto se utiliza normalmente un navegador web que mostrará los datos al cliente. También permite que los navegadores puedan realizar parte del procesamiento local al ejecutar scripts (por ejemplo, `Javascript`) en la página web a la que se ingresa mediante el navegador.
- Otra de las ventajas es que la arquitectura cliente-servidor facilita la utilización del patrón `MVC`.

En cuanto a las funciones del patrón `MVC` en la aplicación web desarrollada son las siguientes:

- El usuario será el encargado de interactuar con las diferentes vistas. Las vistas contienen diferentes tipos de acciones que pueden ser solicitadas por el usuario. Si el usuario solicita una de esas acciones, la vista mandará al controlador realizar la acción solicitada, el controlador realizará la acción obteniendo datos necesarios de los diferentes modelos y cambiando el contenido de la vista.
- La vista por su parte también puede estar en contacto con el modelo siempre que no se requiera de una acción, si no que se requieren datos del modelo para renderizarlos.

El funcionamiento del sistema se recoge en la figura 10.1.

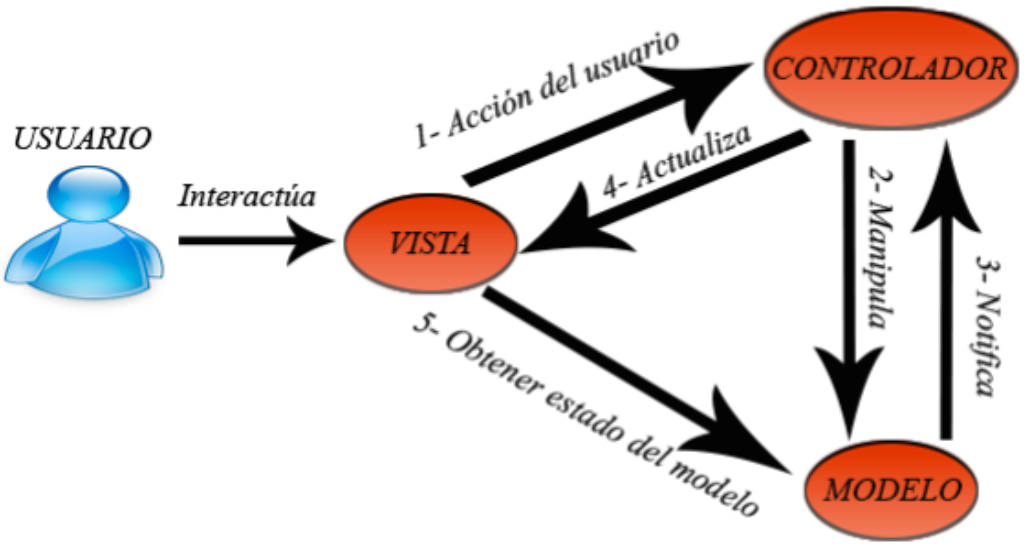


Figure 10.1: Funcionamiento MVC

Las principales tecnologías utilizadas en el simulador son:

- **Frontend**
  - *HTML:* sirve como estructura general de la web. Se emplea HTML5 y CSS3.
  - *JavaScript:* se utiliza este lenguaje para no sobrecargar el servidor, obteniendo diferentes datos o comunicando la vista con el controlador del lado del cliente.
- **Backend:**
  - *Java:* se utiliza para crear la lógica del servidor, proporcionando funcionalidades que interactúan con bases de datos y gestionan el funcionamiento del servidor.
  - *Servlets:* Se utilizan para manejar solicitudes del cliente, en particular los formularios que se muestran a los usuarios y generar respuestas dinámicas. Constituyen una parte fundamental en la creación de aplicaciones web basadas en Java.

Se incluye en la figura 10.2, el modelo de dominio con las relaciones entre las clases y las multiplicidades.

Así mismo, se incluye el diagrama de clases, en la figura 10.3 que ofrece visión más específica de las distintas clases del sistema [1].

[1]Por motivos de espacio, se han eliminado atributos y métodos de esta representación

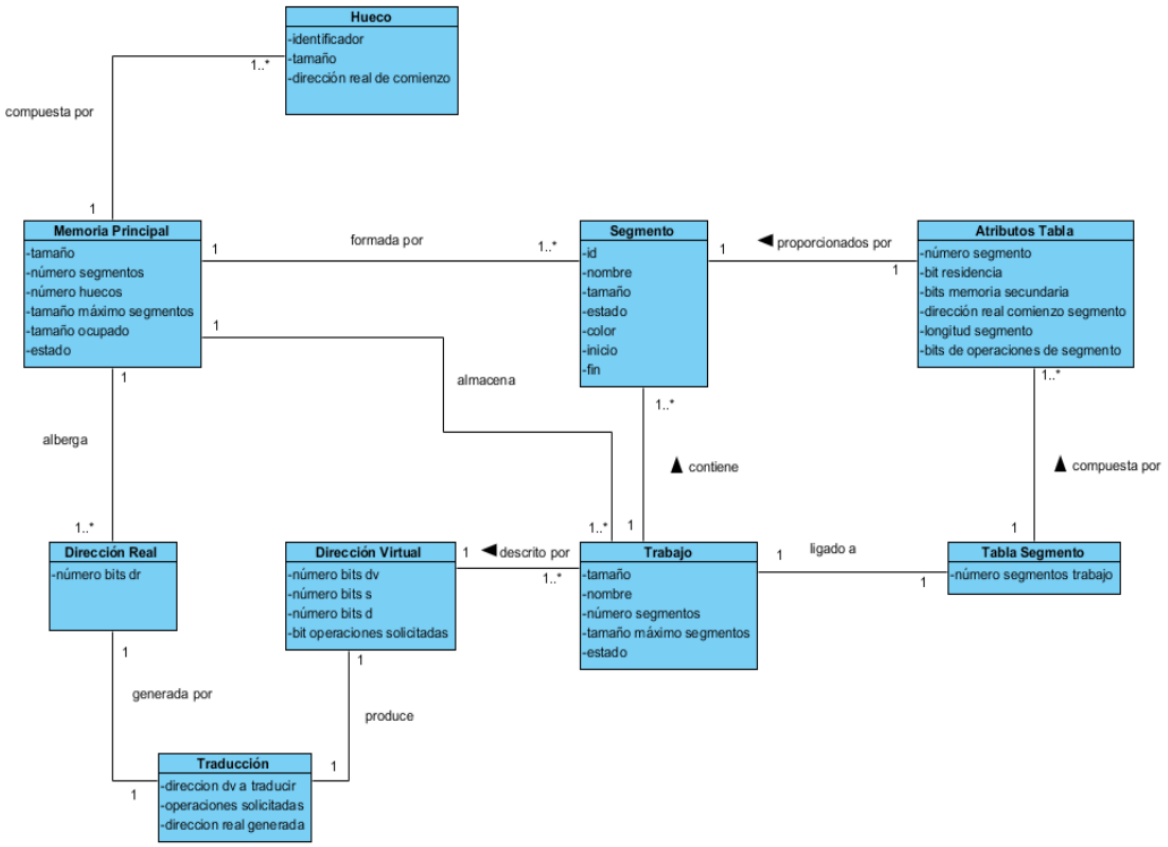


Figure 10.2: Modelo de dominio

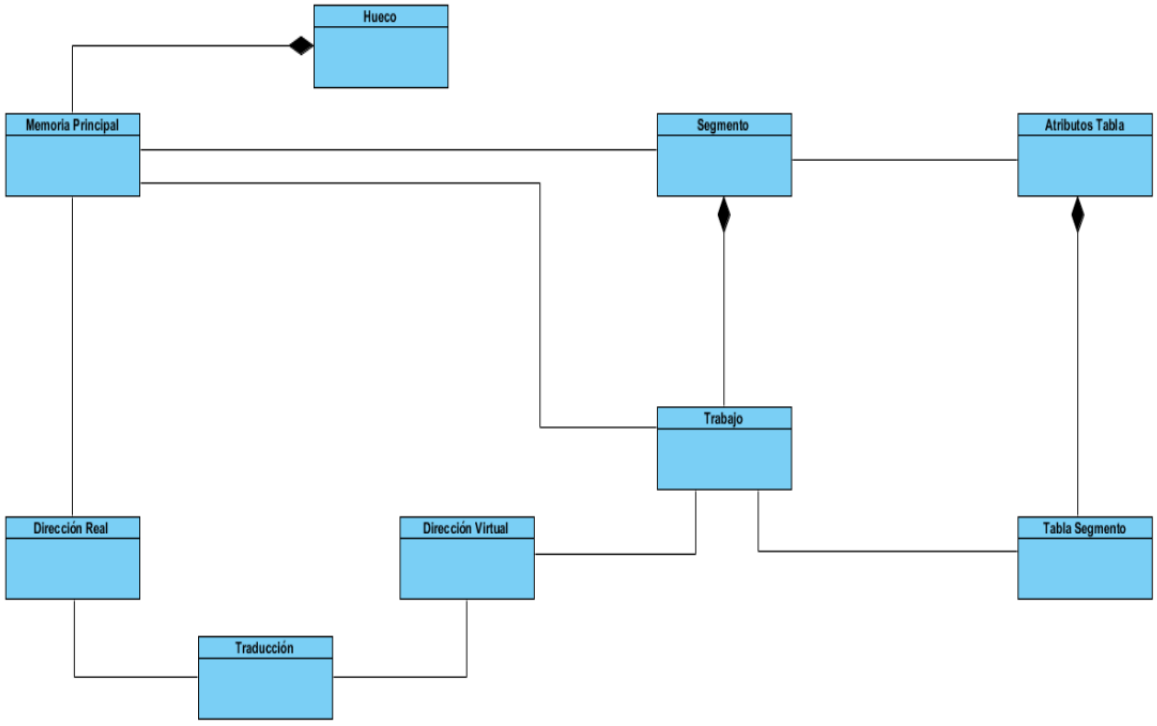


Figure 10.3: Diagrama de clases

Este proyecto se caracteriza por no tener una base de datos a la hora de gestionar los diferentes datos. La información y datos son gestionados a través de variables de sesión, con `Java Session` [Jav24]. En la plataforma `JavaEE` se suele utilizar de forma muy habitual la clase `HttpSession`, la cual permite almacenar cualquier tipo de objeto en ella de manera que pueda ser compartido por las diferentes páginas que existen en la aplicación web.

En la figura 10.4 se puede ver como las distintas páginas pueden acceder a las variables de sesión.

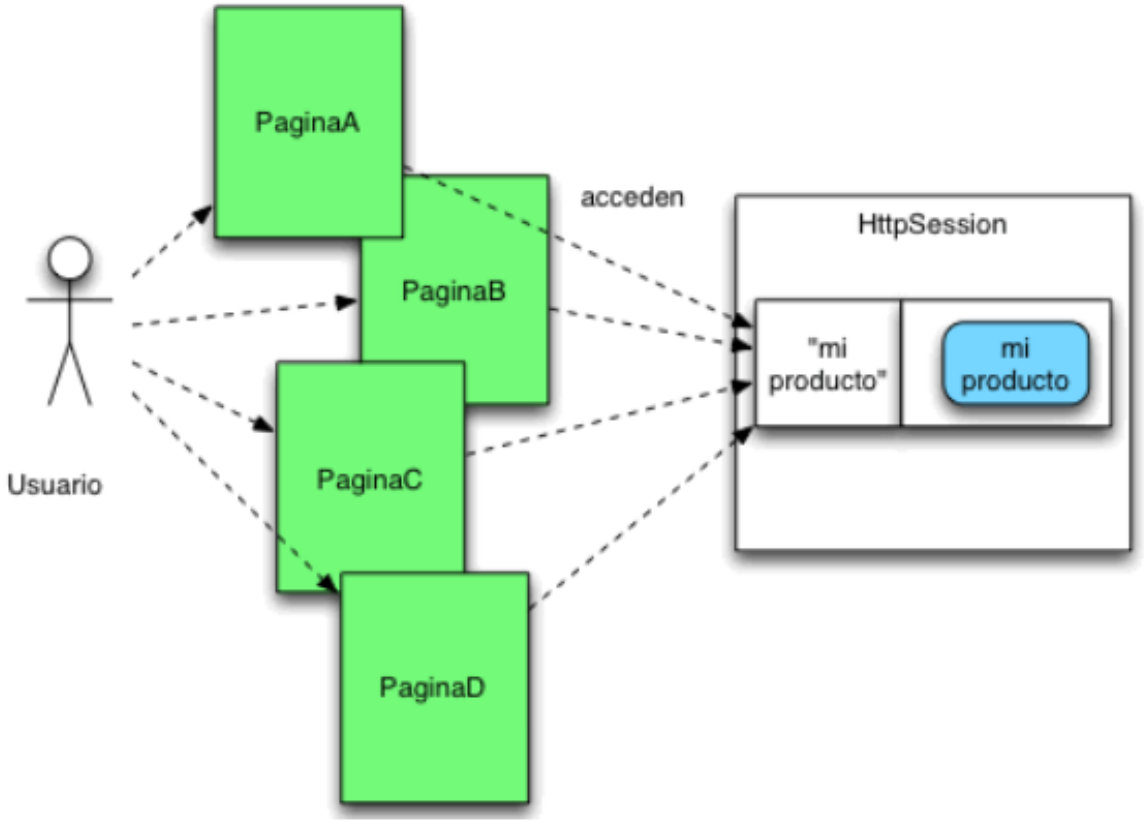


Figure 10.4: Acceso a variables de sesión

El funcionamiento de `Java Session`, puede observarse gráficamente en la figura 10.5, es el siguiente: cada vez que un usuario crea una sesión accediendo a una página se crea un objeto a nivel de servidor con un `HashMap` vacío que nos permite almacenar la información que necesitamos relativa a este usuario. Realizado este paso se envía al navegador del usuario una Cookie que le identifica y le asocia el `HashMap` creado anteriormente. El usuario podrá acceder a dicho `HashMap` desde cualquier vista.

Para obtener las variables de sesión en la aplicación web realizada, es necesario la creación de algún objeto. Este objeto se crea en un `Servlet` específico, por lo que dicho `Servlet` será el encargado de crear la variable de sesión correspondiente al objeto. También hay otro `Servlet` que es el encargado de eliminar las distintas variables de sesión creadas.

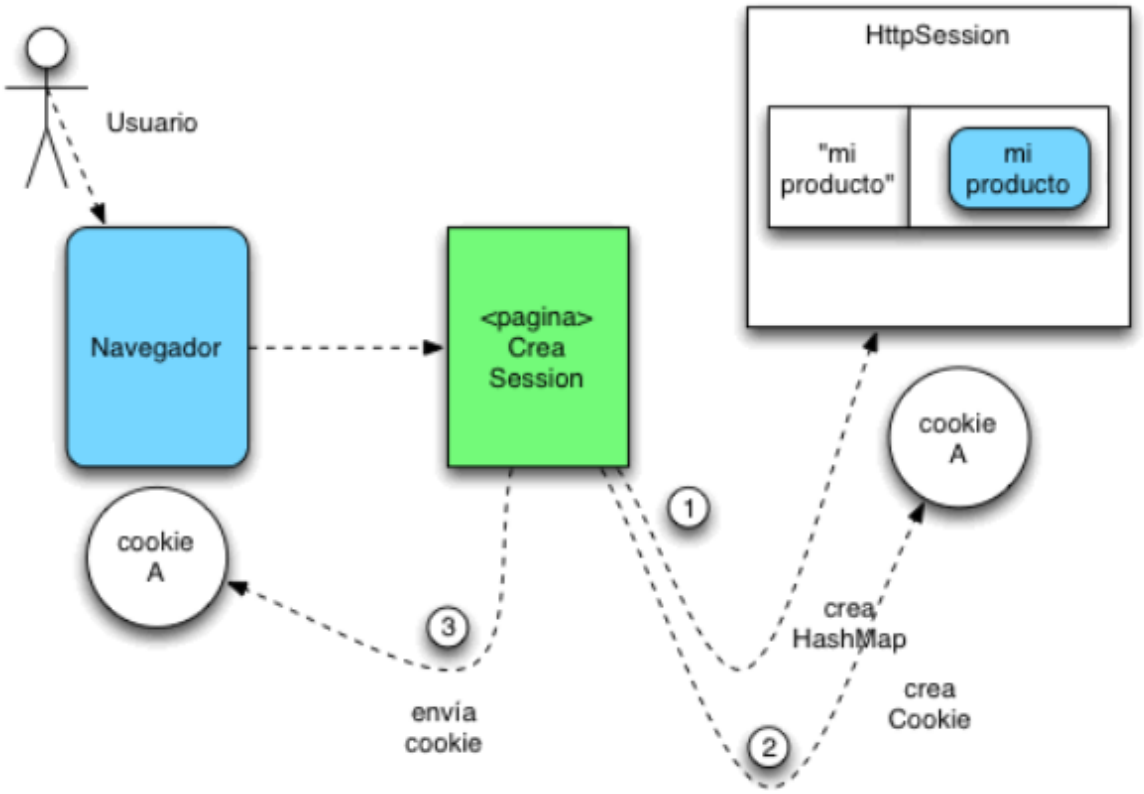


Figure 10.5: Acceso a variables de sesión

En la aplicación web desarrollada dicho `HashMap` contendrá los datos de las variables de sesión, creadas en los `Servlet` del proyecto que son las siguientes:

- *Variable de sesión de memoria principal:* esta variable permite acceder al objeto correspondiente a memoria principal creado por el usuario, el cual permite utilizar todos los métodos correspondientes a la clase Memoria Principal. Esta variable es creada cuando el usuario especifica los datos con los que se describe la memoria principal.
- *Variable de sesión de dirección real:* variable para referenciar al objeto dirección real creado por el usuario, permitiendo acceder a los diferentes métodos de la clase Dirección Real. Esta variable es creada después de que el usuario especifique los datos con los que se describe la memoria principal.
- *Variable de sesión de dirección virtual:* se utiliza para contener el objeto de dirección virtual creado por el usuario, permitiendo acceder a los distintos métodos de la clase Dirección Virtual. Esta variable de sesión es creada cuando el usuario introduce los datos con los que se describe la memoria virtual.
- *Variables de sesión de trabajo:* variable que contiene un trabajo específico creado por el usuario, pudiendo acceder a los distintos métodos de la clase Trabajo. En el proyecto existirá una variable de sesión trabajo por cada trabajo creado por el usuario, es decir, cada trabajo tendrá una variable de sesión específica. Esta variable de sesión es creada cuando el usuario crea un trabajo.

- *Variables de sesión de tabla de segmento:* a través de esta variable de sesión obtenemos un objeto de la clase Tabla Segmento, permitiendo el acceso a los distintos métodos de la clase. En el proyecto existirá una variable de sesión de tabla de segmento por cada trabajo creado por el usuario, es decir, cada tabla de segmento tendrá una variable de sesión específica y que hace referencia a un trabajo específico. Esta variable de sesión es creada cuando el usuario crea un trabajo.
- *Variable de sesión de traducción:* con esta variable obtenemos un objeto de la clase Traducción, permitiendo acceder a los diferentes métodos de la clase. Esta variable es creada cuando el usuario solicita la traducción de una dirección virtual en una dirección real.

10.3

Uso del Simulador

Se puede acceder al simulador a través de la siguiente dirección `http://193.147.87.86:8080/segmentacionUvigo/`. En la página inicial, figura 10.6, se describen usando gráficos conceptos relacionados con la estrategia de segmentación, tales como tabla de segmentos y traducción de dirección virtual a dirección real.

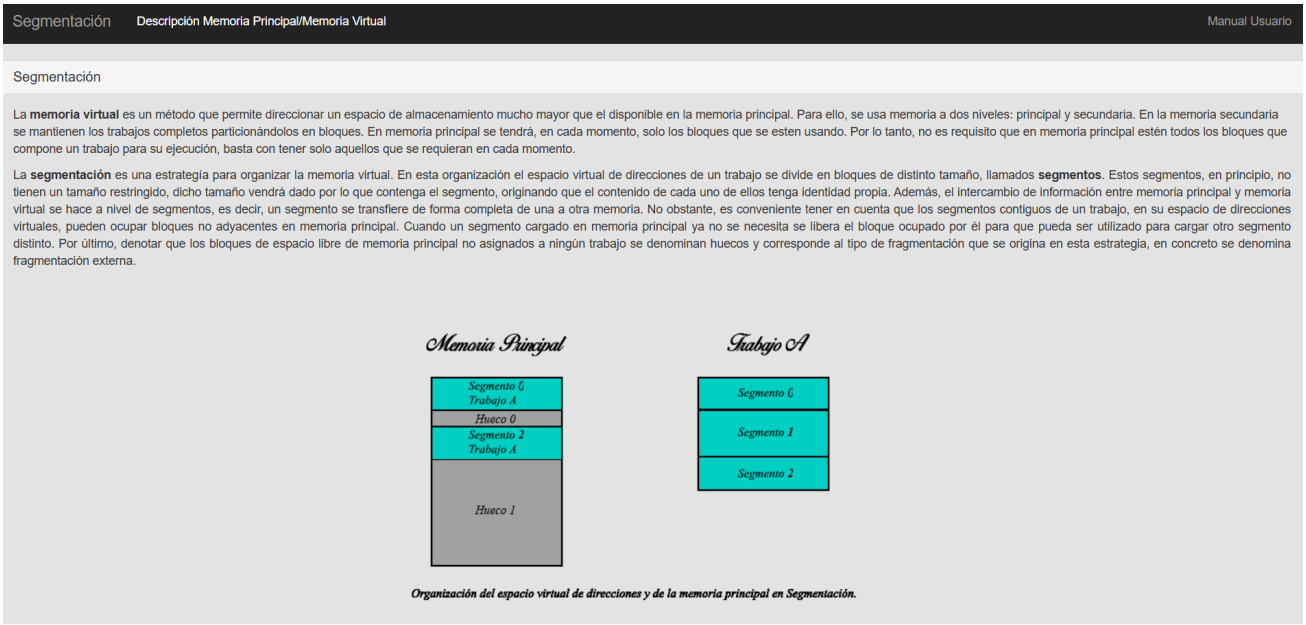


Figure 10.6: Página inicial

Al igual que con el simulador de Paginación, la opción "Descripción Memoria Principal/Memoria Virtual" abre un menú, figura 10.7, con las cuatro opciones disponibles para configurar tanto la memoria principal como la memoria virtual. Estas opciones son las siguientes:

A. Se introduce el tamaño de la memoria principal, el tamaño máximo de segmento y el número de bits utilizados para especificar una dirección virtual.
B. Se introduce el número de bits empleados para especificar la dirección real y el número de bits utilizados para cada una de las componentes de la dirección virtual: número de segmento y desplazamiento.
C. Se introduce el tamaño de la memoria principal, el número de bits utilizados para especificar la dirección virtual y de ellos cuantos se usan para indicar el número de segmento.
D. Se introduce el tamaño de la memoria principal y el número de bits utilizados para cada una de las componentes de la dirección virtual: número de segmento y desplazamiento.

Figure 10.7: Opciones para describir MP/MV

Una vez descrita la memoria virtual y la memoria principal se muestra ésta gráficamente, figura 10.8. En el estado inicial de la memoria principal solo existe un hueco cuyo tamaño se especifica en el gráfico.

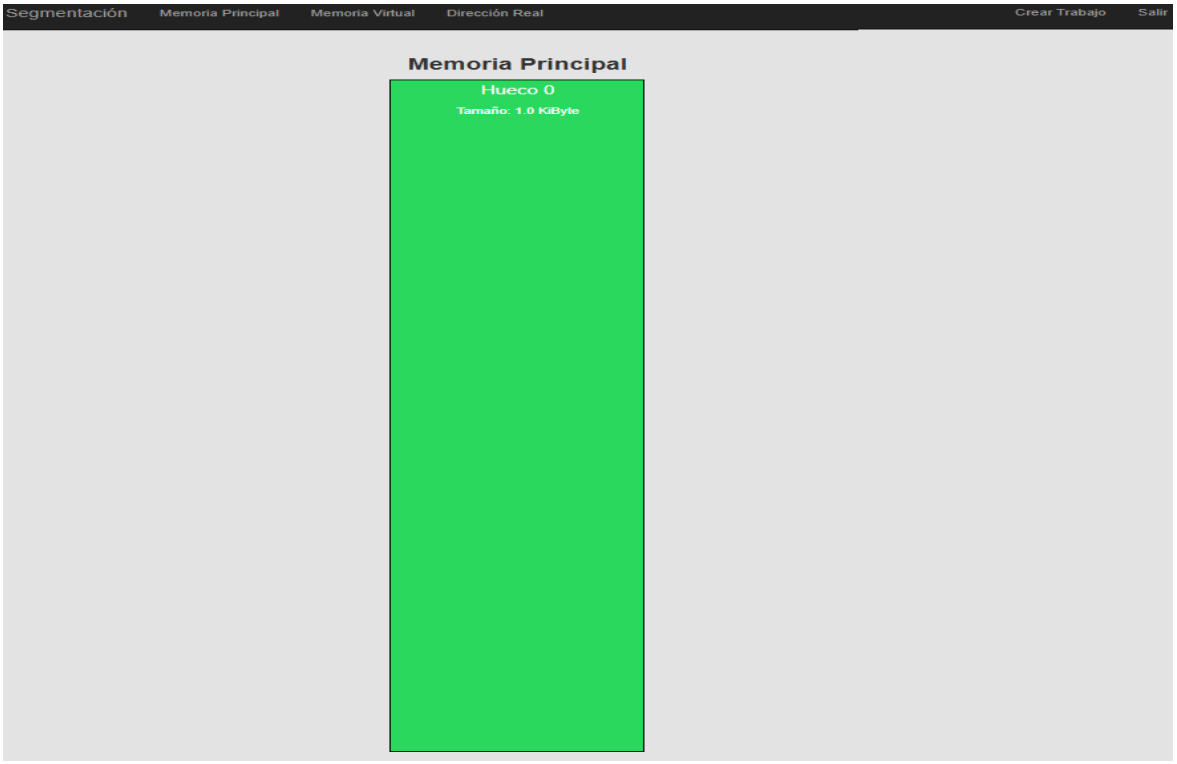


Figure 10.8: Memoria Principal

Crear Trabajo

La opción de "Crear Trabajo" se inicia solicitando el número de segmentos de los que consta el nuevo trabajo, para posteriormente, además de elegir el color que represente el trabajo, se especifiquen los datos de cada segmento: tamaño y operaciones permitidas. Esto se muestra en la figura 10.9.

Una vez creado el trabajo se mostrará gráficamente (Figura 10.10) dividido en los segmentos del tamaño especificado.

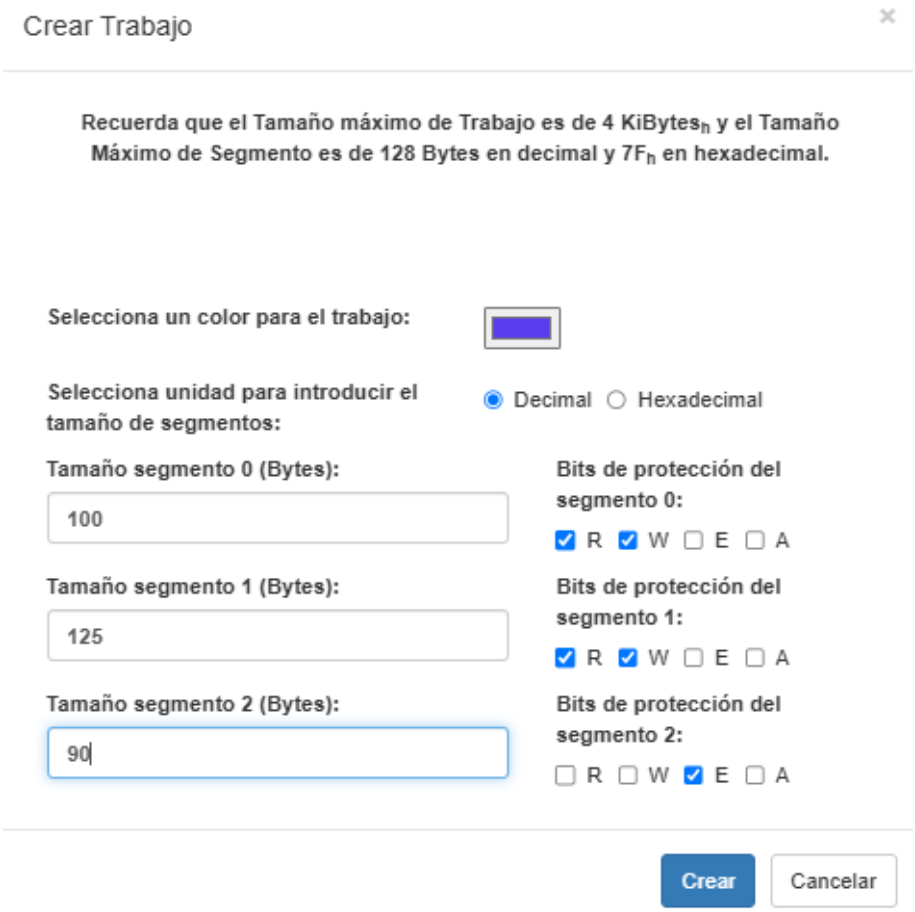


Figure 10.9: Datos de los segmentos de un trabajo

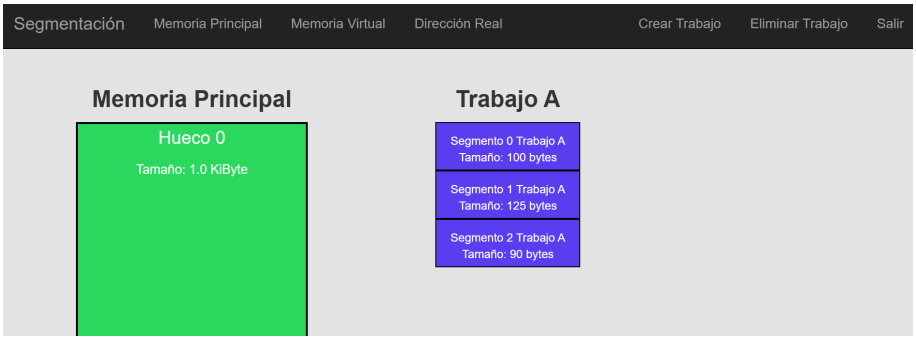


Figure 10.10: Trabajo creado

Cargar segmento en memoria principal

Al pulsar sobre el nombre del trabajo, aparecen dos formas de indicar dónde se va a cargar uno de sus segmentos en memoria principal, figura 10.11:

- Seleccionando el hueco.

- Especificando la dirección real.

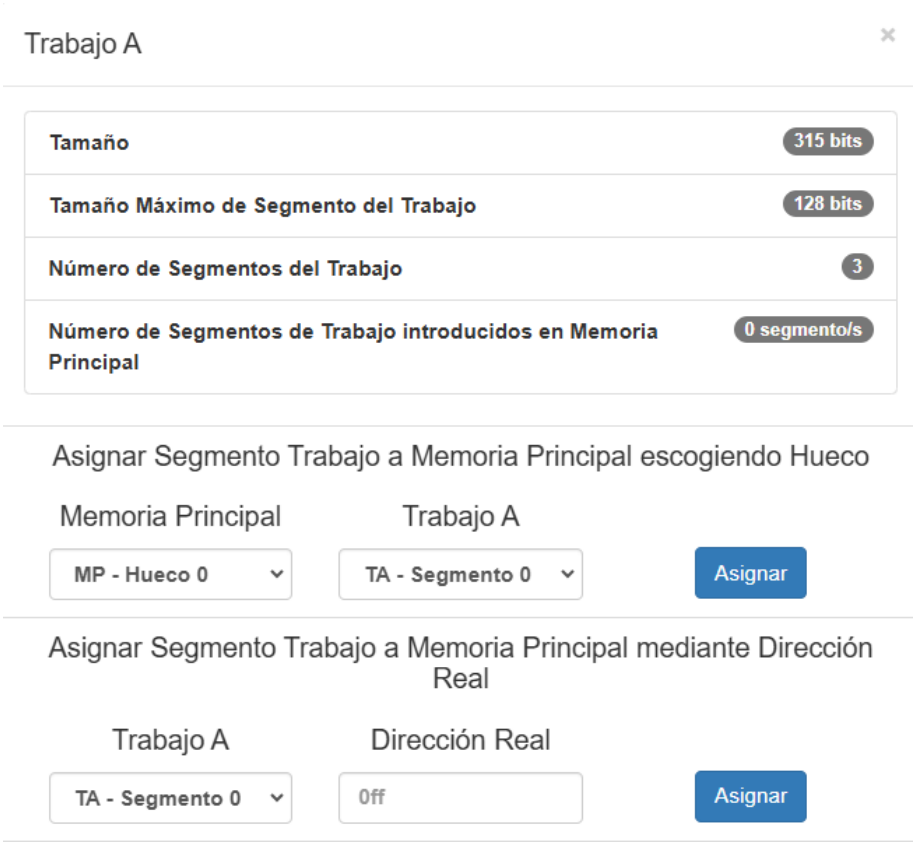


Figure 10.11: Métodos para cargar un segmento en Memoria Principal

Una vez introducido el hueco o la dirección real, se actualiza gráficamente el estado de la memoria principal y de los segmentos que componen el trabajo. Esto se puede ver reflejado en la figura 10.12 donde se carga el segundo segmento del trabajo en el primer hueco de la memoria principal.

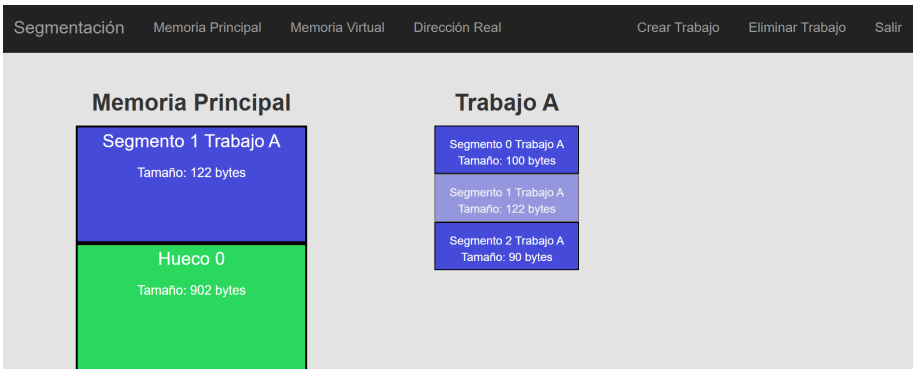
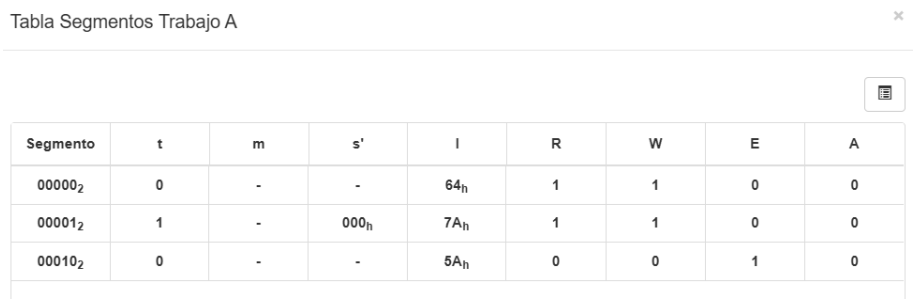


Figure 10.12: Carga de segmento en Memoria Principal

Consultar tabla de mapa de segmentos

En Segmentación, todo trabajo tiene una tabla de segmentos asociada. La aplicación permite consultar el contenido actual de dicha tabla mediante la opción "Tabla Segmentos". Cada fila de dicha tabla proporciona información de un segmento, de forma que la fila 0 corresponde al segmento 0 del trabajo, la fila 1 al segmento 1 y así sucesivamente. Si el segmento está cargado en memoria principal, entonces s' indica la dirección real a partir de la cuál esta cargado dicho segmento. En la figura 10.13 se puede comprobar que el segundo segmento está cargado a partir de la dirección real $000t_{)H}$.



Segmento	t	m	s'	l	R	W	E	A
00000_2	0	-	-	64_h	1	1	0	0
00001_2	1	-	000_h	$7A_h$	1	1	0	0
00010_2	0	-	-	$5A_h$	0	0	1	0

Figure 10.13: Tabla de mapa de segmentos

Fragmentación de tablas

El simulador también proporciona la posibilidad de calcular la fragmentación de tablas, es decir la cantidad de memoria principal ocupada por la tabla de segmentos.

A la hora de seleccionar esta opción, se solicita al usuario la introducción del número de bits necesarios para especificar la dirección en memoria secundaria donde reside la página.

Traducción de una dirección virtual en una dirección real

Por último, también se puede solicitar la traducción de una dirección virtual a una dirección real mediante la opción "Traducir DV" que aparece al seleccionar el nombre del trabajo. Una vez que se introduce en hexadecimal la dirección virtual y las operaciones que se desean realizar, se muestran los siguientes pasos para realizar dicha traducción

1. Se traduce la dirección virtual de hexadecimal a binario para identificar las dos componentes de una dirección virtual, es decir el número de segmento (s) donde está el elemento al que se referencia y el desplazamiento (d) que es la distancia que dista desde el comienzo del segmento al elemento en cuestión, tal como figura en la figura 10.14.
2. Se comprueba si el segmento reside en memoria principal. Para ello, se determina la fila de la tabla de segmentos que proporciona la información sobre el segmento en cuestión. De dicha fila se comprueba

Figure 10.14: Traducción *dv* a *dr*: identificación componentes de dirección virtual

el valor del bit de residencia t_s (figura 10.15). Si este bit indica que el segmento no reside en memoria principal, se habrá producido un *Fallo de pérdida de segmento*.

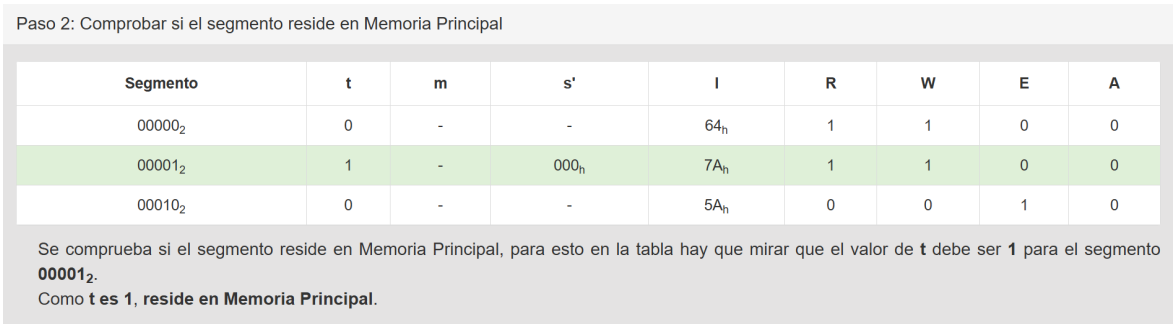


Figure 10.15: Traducción *dv* a *dr*: comprobar si el segmento reside en Memoria Principal

3. Como cada segmento tiene un tamaño variable es necesario comprobar que el elemento al que se hace referencia en la dirección virtual pertenece al segmento. Para ello, se comprueba que **l** >= **d**, dónde **l** es la longitud (número de elementos menos uno) del segmento referenciado y **d** es el desplazamiento obtenido en el primer paso. Esto se refleja en la figura 10.16.

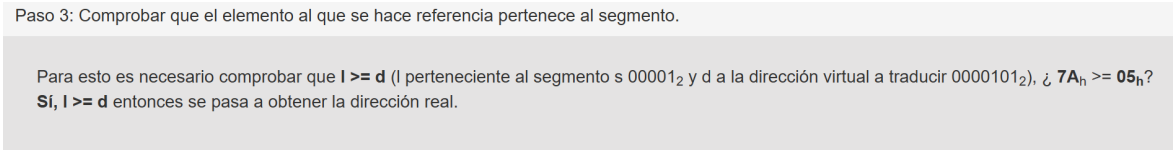


Figure 10.16: Traducción *dv* a *dr*: comprobar si el elemento pertenece al segmento

En el caso de que esta condición no se cumpla se habrá producido un *Fallo de desbordamiento de segmento*, ya que se desea referenciar un elemento que está más allá del final del segmento.

4. El contenido de cada segmento tiene identidad propia por lo que se pueden controlar las operaciones que el proceso puede realizar sobre su contenido. Para ello, se usan los bits de protección asociados al segmento referenciado. (Figura 10.17).
   En el caso de que alguna de las operaciones solicitadas no se pueda realizar se habrá producido un *Fallo de protección de segmento*, ya que el proceso intenta realizar una acción para la que no tiene permiso.

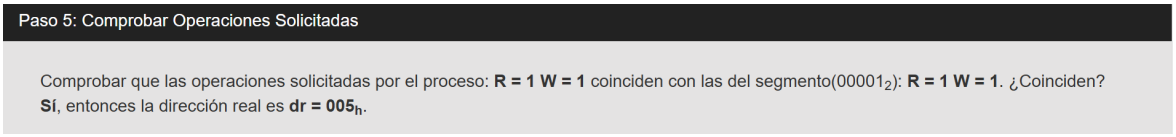


Figure 10.17: Traducción dv a dr: comprobar permiso sobre las operaciones solicitadas

5. Por último, se obtiene la dirección real deseada concatenando la dirección real donde comienza el segmento en memoria principal (s') con el desplazamiento (d). (Figura 10.18).

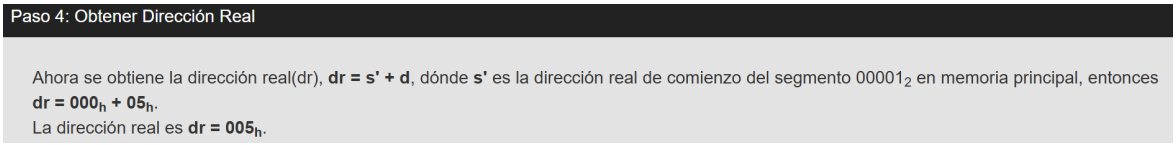


Figure 10.18: Traducción dv a dr: concatenar s' y d

10.4

Generación automática de ejercicios con IA generativa

Al igual que en Paginación, también se han usando herramientas de inteligencia artificial generativa para obtener las soluciones a una relación de ejercicios que se ha elaborado incrementando su complejidad de forma progresiva sobre aspectos relacionados con la Segmentación. El comienzo de ésta relación contiene el siguiente enunciado común al conjunto de ejercicios.

> Supongamos un sistema segmentado con una memoria principal de 1 Mebibyte y donde cada dirección virtual es de 16 bits repartidos de la siguiente forma:
> - 6 bits para el número de segmento (s),
> - 10 bits para el desplazamiento dentro del segmento (d).
>
> El tamaño de cada posición de memoria es de 1 byte.

A continuación se muestran tanto los prompts como las respuestas obtenidas por `ChatGPT`:

Pregunta a: ¿Cuál es el tamaño máximo de un segmento?

Dado que la dirección virtual es de 16 bits y está dividida en 6 bits para el número de segmento y 10 bits para el desplazamiento, el tamaño máximo de un segmento es el máximo valor que puede representar 10 bits de desplazamiento. Así:

$$2^{10} = 1024 \text{ bytes} = 1 \text{ KiB}$$

Por lo tanto, el tamaño máximo de un segmento es 1 KiB.

Pregunta b: ¿Puede este Sistema Operativo aceptar un trabajo de 80 KiB?

Cada segmento tiene un tamaño máximo de 1 KiB. Para un trabajo de 80 KiB, el sistema requeriría al menos 80 segmentos. Dado que el número de segmento está limitado a 6 bits, solo se pueden definir un máximo de $2^6 = 64$ segmentos.

Por lo tanto, el sistema **no puede aceptar** un trabajo de 80 KiB debido a esta limitación.

Pregunta c: Traducción de Direcciones Virtuales a Direcciones Físicas (Corrección de Dirección de Inicio)

1. **Dirección Virtual: 046C)H (Operaciones solicitadas: Lectura y Escritura)**
   - Convertimos 046C)H a binario:

$$046C)_{16} = 0000\ 0100\ 0110\ 1100)_2$$

   - Usamos los primeros 6 bits para el número de segmento y los últimos 10 bits para el desplazamiento:
     - **Número de segmento**: 000001 (1 en decimal)
     - **Desplazamiento**: 000001101100 (06C en hexadecimal)
   - Verificamos el segmento 1 en la tabla de segmentos:
     - El bit de residencia (ts) es 1, por lo que el segmento reside en memoria principal.
     - La longitud del segmento (l) es 35F)H.
   - Comprobamos si el desplazamiento 06C)H está dentro de los límites del segmento:
     - Como 06C)H es menor o igual a 35F)H, el acceso es válido.
   - Verificación de permisos:
     - Las operaciones solicitadas son lectura y escritura.
     - Los bits de protección para el segmento 1 son: R = 1 (lectura permitida), W = 1 (escritura permitida).
     - Dado que ambos permisos están habilitados, las operaciones solicitadas son válidas.
   - La dirección física se obtiene sumando el desplazamiento a la dirección de inicio del segmento:

$$s' + \text{desplazamiento} = A0026)H + 06C)H = A0092)H$$

Dirección física: A0092)H

Traducción de la Dirección Virtual **0BCF)H** con Operación de Lectura

1. **Conversión de la Dirección Virtual a Binario**
   - Dirección virtual en hexadecimal: **0BCF**
   - Convertimos a binario:
   $$0BCF)_{16} = 0000\ 1011\ 1100\ 1111)_2$$
2. **Separación de Número de Segmento y Desplazamiento**
   - Usamos los primeros 6 bits para el número de segmento y los últimos 10 bits para el desplazamiento:
     - **Número de segmento**: 000010 (que es 2 en decimal)
     - **Desplazamiento**: 1111001111 (que es **3CF** en hexadecimal)
3. **Consulta de la Tabla de Segmentos**
   - Para el segmento **2**, consultamos en la tabla de segmentos:
     - El bit de residencia (ts) es 1, indicando que el segmento reside en memoria principal.
     - La dirección de inicio del segmento (s') es **00000)H**.
     - La longitud del segmento (l) es **0A1)H**.
     - Bits de protección: R = 1 (lectura permitida), W = 1, E = 1, A = 0.
4. **Verificación del Desplazamiento y Permisos**
   - Comparamos el desplazamiento **3CF)H** con la longitud del segmento **0A1)H**:
     - **3CF)H es mayor que 0A1)H**, lo cual significa que el desplazamiento excede la longitud del segmento.
   - Como el desplazamiento excede los límites permitidos, se produce un **fallo de desbordamiento de segmento**.

XI

Simulador 3: Paginación/Segmentación

II.1

Objetivo

El objetivo principal de esta aplicación no es solo simular el comportamiento de la memoria principal y de la virtual siguiendo la estrategia de paginación/segmentación, también es hacerlo de forma intuitiva y, sobre todo, didáctica.

Este simulador se desarrolla cómo una aplicación web que simula la técnica de organización de la memoria virtual paginación/segmentación, sirviendo como herramienta de aprendizaje y de autoevaluación para alumnos de Sistemas Operativos I. Para reforzar el aprendizaje del alumno, todas las acciones que puede realizar en la aplicación son guiadas e intuitivas. Esto se consigue gracias a los numerosos paneles de ayuda repartidos en las partes más importantes del simulador.

De esta forma, las principales funcionalidades del simulador son:

- **Describir la memoria principal y virtual:** existen varias opciones para describir la memoria principal y la virtual, sin embargo, todas tienen algo en común: los datos proporcionados permiten calcular los datos restantes. Por eso, en vez de presentar un largo menú con todas las opciones, se optó por uno único formulario en el que:
  - Conociendo el tamaño de la memoria principal se calcula el tamaño de una dirección real (y viceversa).
  - Conociendo el Tamaño de un marco de página se calcula el número de bits de desplazamiento (y viceversa).
  - Tamaño de la memoria principal,tamaño de un marcos de página y número de marcos de página: conociendo dos de estos datos permite calcular el restante.

  La memoria virtual se describe indicando el tamaño de cada una de sus componentes (s, p y d). Si antes se describió la memoria principal, la componente d (bits de desplazamiento) ya estará calculada. Una vez descrita la memoria principal y la virtual se carga la representación de la memoria principal en el simulador (pudiendo consultar en todo momento a configuración especificada) y se le permite al usuario crear trabajos.

- **Crear trabajos:** el usuario crea trabajos definiendo los segmentos que los forman, indicando tamaño y operaciones permitidas. Estos trabajos se van añadiendo a la bandeja de trabajos en la que se puede consultar toda la información referente a dichos trabajos: segmentos en los que se dividen, páginas en las que se dividen dichos segmentos y el tamaño de estas. Para crear los trabajos, el usuario puede añadir hasta 2^S segmentos; siendo s el número de bits de la dirección virtual que se usan para indicar el número de segmento. Además, los segmentos no podrán tener un tamaño mayor a $T_{pag} \cdot 2^P$; siendo p los bits de la dirección virtual que se usan para indicar el número de página dentro de un segmento y T_{pag} el tamaño máximo de una página.
- **Cargar páginas en memoria principal:** el proceso para cargar una página es sencillo y rápido. El usuario simplemente sin que seleccionar una página desde la bandeja de trabajos e indicar el marco de página en el que la quiere cargar. Una vez cargada, se actualizará la representación de la memoria principal, mostrando el marco en el que reside en caso de que este estuviera oculto e indicando la fragmentación interna que se produce en ese marco.
- **Consultar las tablas de segmentos y páginas:** para cada trabajo, el simulador coge toda la información necesaria y construye su tabla de segmentos y las tablas de páginas. La tabla de segmentos puede ser accedida desde el panel del trabajo en la bandeja de trabajos. La tabla de páginas de un segmento puede ser accedida desde el panel del segmento o desde su entrada en la tabla de segmentos.
- **Calcular la fragmentación de tablas y traducción de direcciones virtuales a direcciones reales:** optando por que sea un proceso didáctico, no se muestra simplemente el resultado final. Los cálculos se realizan paso a paso de forma gráfica, permitiendo que el alumno avance a su ritmo. Además, se le permite al alumno realizar esos cálculos siguiendo el mismo procedimiento paso a paso.

II.2

Arquitectura y Tecnologías

La arquitectura del simulador se basa en el modelo cliente-servidor [Somo5]. Como se ha descrito previamente, se trata de un modelo en el que el sistema se organiza como un conjunto de servicios y servidores asociados, además de unos clientes que acceden y usan dichos servicios.

- **Servidores:** Si bien en un modelo cliente-servidor se puede contar con varios servidores independientes, en este simulador se encuentra todo centralizado en un único servidor que ofrece todos los servicios.
- **Clientes:** Invocan los servicios ofrecidos mediante un protocolo de petición-respuesta, que en este caso es el protocolo HTTP.

- **Red:** Permite a los clientes acceder a los servicios, si bien esto no es estrictamente necesario ya que los clientes y servidores se podrían ejecutar en la misma máquina, para el propósito del simulador se considera que los clientes acceden de forma externa a los servicios.

En la arquitectura cliente-servidor se distinguen 3 capas lógicas:

- **Capa de presentación:** se encarga de mostrar la información e interactúa con el usuario. En este simulador las vistas se construyen en el servidor y se envían a los clientes.
- **Capa de procesamiento de la aplicación:** implementa la lógica de la aplicación. En este simulador, está lógica reside bien en el cliente o en el servidor en función de la naturaleza de la lógica.
  - Los cálculos cuyos resultados persisten durante la duración de la sesión se realizan en el servidor, como la organización de la memoria principal y virtual, los trabajos creados, las tablas de segmentos y páginas, etc.
  - Los cálculos cuyos resultados son temporales y se pueden desechar una vez realizados se realizan en el cliente, como es el cálculo de la fragmentación de tablas y la traducción de direcciones virtuales.
- **Capa de administración de datos:** se refiere a todas las operaciones de la base de datos. En esta aplicación no hay persistencia y la administración de datos la realizan los `Managed Beans` [Cor13].

Concretando mas en la arquitectura, al estar programada en `Java` con la especificación `Java EE`, la aplicación presenta la arquitectura propia de dicha plataforma [Fou19], dividida en contenedores y componentes. Se distinguen los siguientes componentes:

- **Clientes:** componente que se ejecuta en el lado del cliente.
  - *Clientes de aplicación:* se ejecutan como aplicaciones de escritorio. Se utilizan cuando los usuarios necesitan llevar a cabo tareas que requieren interfaces de usuario más complejas que las que puede proporcionar un lenguaje de marcado.
  - *Clientes web:* es el que se emplea en este simulador. Los clientes web consisten en dos partes:
    - Paginas web dinámicas que contienen varios tipos de lenguajes de marcado, y que son generadas por componentes web.
    - Navegador web que representa las páginas recibidas por el servidor.
- **Componentes web:** se ejecutan en el contenedor web y son los responsables de componer la presentación de los datos de la aplicación en formato `HTML`. En este simulador hay dos tipos de componentes web:
  - *`Servlets`:* clases `Java` que procesan dinámicamente las peticiones enviadas por el cliente y construyen las respuestas.

  - *Páginas web creadas utilizando JSF y/o JSP:* documentos de texto que ejecutan los servidores y crean dinámicamente el contenido estático que se envía al navegador.
- **Componentes de negocio:** implementan la capa de la lógica de negocio. En este simulador sin las clases que maneja el `Servlet`.
- **Capa EIS:** gestiona la información permanente del sistema. En esta aplicación no hay persistencia de datos

En la figura 11.1 se muestra un detalle de la estructura anterior.

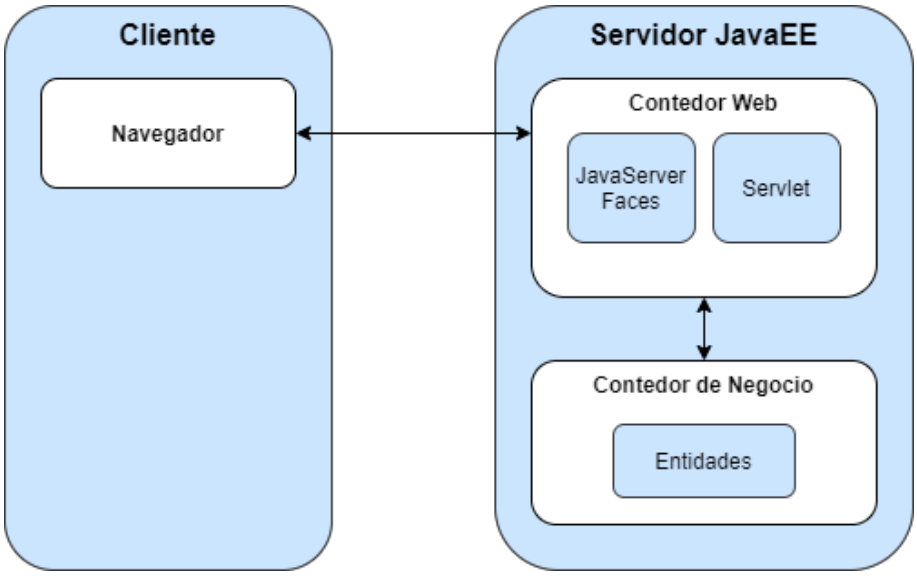


Figure 11.1: Detalle arquitectura cliente-servidor de JavaEE

Dentro de las tecnologías de la especificación JavaEE, para la presentación web se utilizó el framework Java Server Faces (JSF) que implementa el patrón modelo-vista-controlador (MVC). Este patrón se para los datos de una aplicación, la interfaz y su lógica en distintos componentes [Uni].

La estructura del simulador en base a dicho patrón, se recoge en la figura 11.2 y se compone de los siguientes elementos:

- **Modelo:** contiene una representación de los datos que maneja el sistema, su lógica de negocio y los mecanismos de persistencia. En JSF son los `Managed Beans`, `Java Beans` responsables de la lógica de la aplicación que responden a los eventos generados por los componentes JSF y controlan la navegación entre páginas.
- **Vista:** compone la información que se envía al cliente y los mecanismos de interacción con este. En JSF está formado por `Facelets` que son fichero JSP con las bibliotecas de etiquetas de JSF y otros lenguajes de declaración de páginas (PDLs) que describen la jerarquía de componentes JSF que

conforman cada una de las páginas de la interfaz de usuario y que vinculan los componentes con los Managed Beans.

- **Controlador:** actúa como intermediario entre el modelo y la vista, gestionando el flujo de información entre ellos y la transformaciones para adaptar los datos a las necesidades de cada uno. En JSF con los `Faces Servlet` que contienen los métodos de acción de los Managed Beans. Cabe destacar que todas las peticiones HTTP del usuario se pasan al `Faces Servlet`que se encarga de examinar las peticiones, actualizar la representación de la interfaz y los datos de los `Managed Beans` además de invocar a gestores de de eventos y acciones sobre el modelo

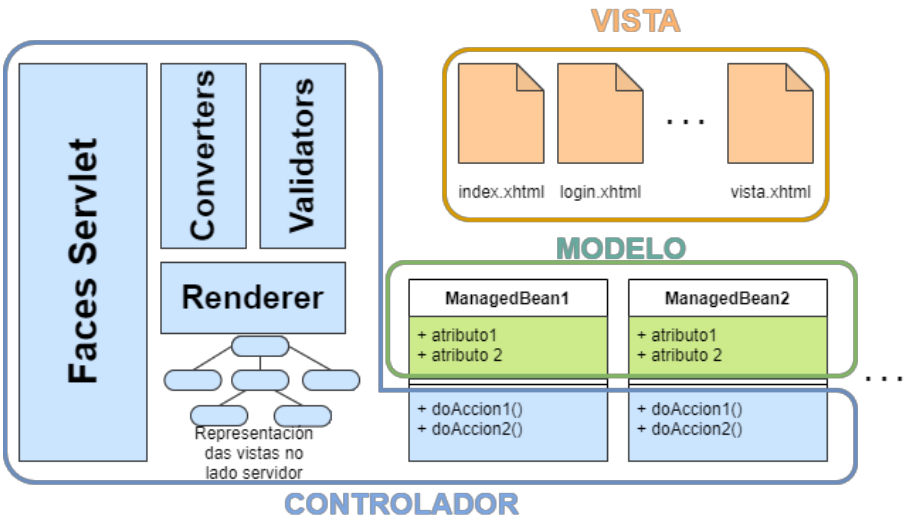


Figure II.2: Detalle arquitectura cliente-servidor de `JavaEE`

De forma concreta las tecnologías empleadas en la implementación de este simulador son:

1. `HTML5`: quinta revisión de `HTML`, el lenguaje de marcado para la elaboración de páginas web. En el simulador se emplea para la programación de las páginas y componentes `JSF`.
2. `JavaScript`: lenguaje de programación interpretada que se emplea en la lógica del lado del cliente, en este caso para la comunicación entre la vista y el servidor, para la actualización de componentes y para editar la interfaz de forma dinámica. Además, se usa de forma específica `JQuery` que se permite interactuar con los documentos `HTML`, manipular el árbol `DOM` y crear animaciones.
3. `JSP`: se emplea por `JSF` como la tecnología que permite hacer el despliegue de las páginas.
4. `JSTL`: extiende a `JSP` proporcionando librerías de etiquetas con utilidades para el desarrollo de webs dinámicas como el de este simulador.

11.3

Uso del Simulador

El simulador desarrollado está disponible en la siguiente dirección `http://193.147.87.86:8080/TFGSimuladorPaxSegSergio`. En la página inicial, figura 11.3, existe la opción "Descubre la Paginación/Segmentación" dónde se describen conceptos relacionados con la estrategia de Segmentación Paginada, en concreto, además de detallar como se organiza la memoria principal (en marcos de página) y los trabajos almacenados en memoria secundaria (cada trabajo en segmentos y estos a su vez en páginas), también se exponen los siguientes aspectos, tal como aparecen en la figura 11.4:

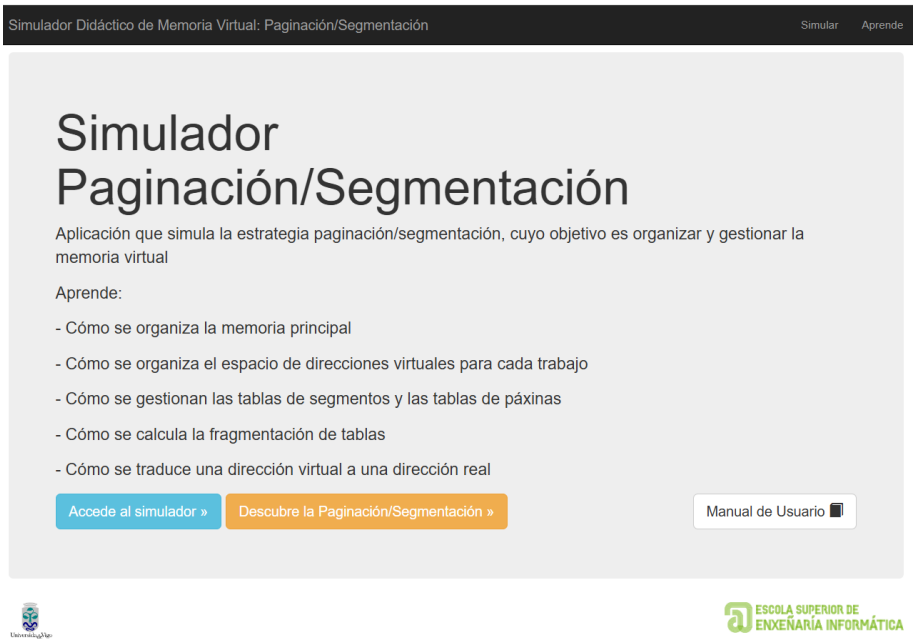


Figure 11.3: Página inicial

1. Tablas o mapas de bloques de segmentos y páginas.
2. Fragmentación de tablas.
3. Traducción dinámica de direcciones virtuales a direcciones reales.

Por otra parte, mediante la opción "Accede al simulador" de la página inicial se comienza seleccionando la opción "Describir MP/MV" que permite describir tanto la memoria principal como la memoria virtual, figura 11.5.

En el caso de la memoria principal, los datos requeridos se van autocompletándo a medida que se van introduciendo el resto. Por ejemplo, si se introduce el tamaño de la memoria principal se autocompletará el número de bits requeridos para una dirección real, o viceversa. Otro ejemplo es si se introduce el tamaño

Figure 11.4: Descubre la Paginación/Segmentación

de la memoria principal y el número de marcos de página en los que se divide, se autocompletará el tamaño del marco de página así como el número de bits con los que se especificará el desplazamiento en la dirección virtual.

A la hora de describir la memoria virtual, excepto los bits para el desplazamiento, el resto (s número de bits para especificar el segmento y p número de bits para especificar la página dentro del segmento) son necesarios que se precisen.

Una vez descritas la memoria virtual y la memoria principal se muestra una representación gráfica de ésta última, figura 11.6, dividida en el número de marcos de página especificados. Por cada marco de página se detalla tanto la dirección real donde comienza el marco como la dirección real donde finaliza.

Figure 11.5: Definir MP/MV

Figure 11.6: Organización de la Memoria Principal

Añadir un trabajo

La opción de "Añadir trabajo" permite introducir un nuevo trabajo eligiendo el color que lo representa y especificando cada uno de sus segmentos. Por cada segmento, además de indicar su tamaño también se debe detallar las operaciones permitidas sobre él, tal como aparece en la figura 11.7.

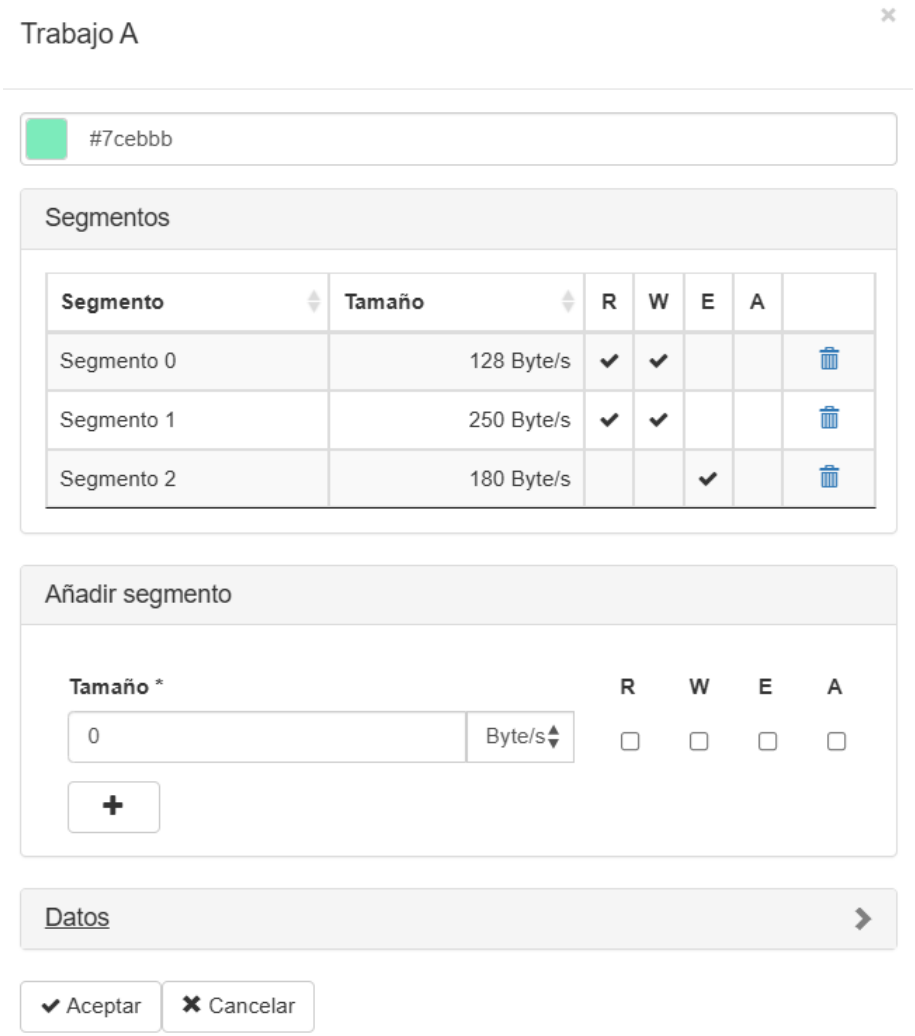


Figure 11.7: Descripción de un trabajo

Una vez creado el trabajo se mostrará en la bandeja de trabajos y al seleccionar el nombre del trabajo éste se despliega viendo su organización (segmentos y páginas). (Figura 11.8)

Figure 11.8: Organización de un trabajo

Cargar página en memoria principal

Desplegado la organización de un trabajo, se puede cargar en memoria principal una página concreta de dicho trabajo si se pulsa sobre el icono "+" que la acompaña. Tal como aparece en la figura 11.9 se seleccionará el marco de página entre los que siguen libres.

Después de cargar la página, el marco de la memoria principal seleccionado aparecerá del color elegido para el trabajo al que pertenece la página, indicando el trabajo y el segmento de dicho trabajo. También aparecerá sombreada la página cargada en la bandeja de trabajos. Todo esto se muestra en la figura 11.10.

Figure 11.9: Elección de marco donde cargar la página seleccionada

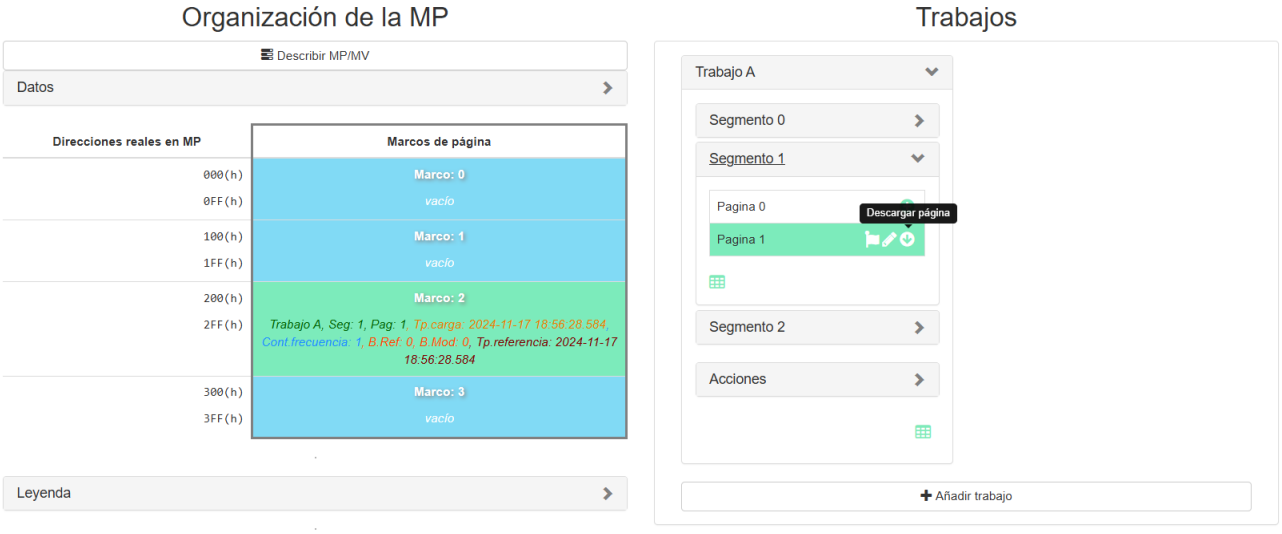


Figure 11.10: Carga de página en un marco de la Memoria Principal

Consultar tablas de mapa de segmentos y de páginas

En Segmentación Paginada todo trabajo tiene una tabla de segmentos y una tabla de páginas por cada

segmento. La aplicación permite consultar el contenido actual tanto de la tabla de segmentos como de las tablas de páginas asociadas a cada segmento de un trabajo.

La tabla de segmentos se puede consultar desde la bandeja de trabajos abriendo el menú "Acciones" y seleccionando "Ver tabla de segmentos". Estos pasos se muestran en la figura 11.11.

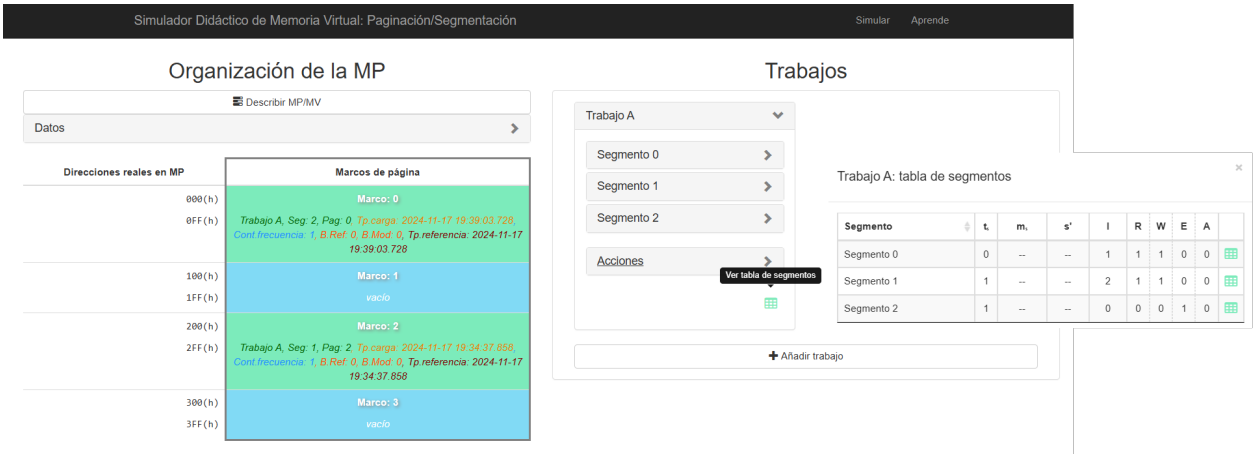


Figure 11.11: Tabla de segmentos de un trabajo

Cada fila de la tabla de segmentos proporciona información de un segmento del trabajo, de forma que la fila 0 corresponde al segmento 0 del trabajo, la fila 1 al segmento 1 y así sucesivamente.

En la figura 11.12 se puede ver las dos formas que ofrece la aplicación para consultar la tabla de páginas de un segmento:

1. Desde la bandeja de trabajos, desplegando el panel del segmento y haciendo clic en el icono de la tabla que figura en el extremo inferior.
2. Desde la tabla de segmentos del trabajo haciendo clic en el icono de la tabla situado en la fila del segmento que se desea consultar.

El contenido de una de esas tabla de páginas también se muestra en la figura 11.12.

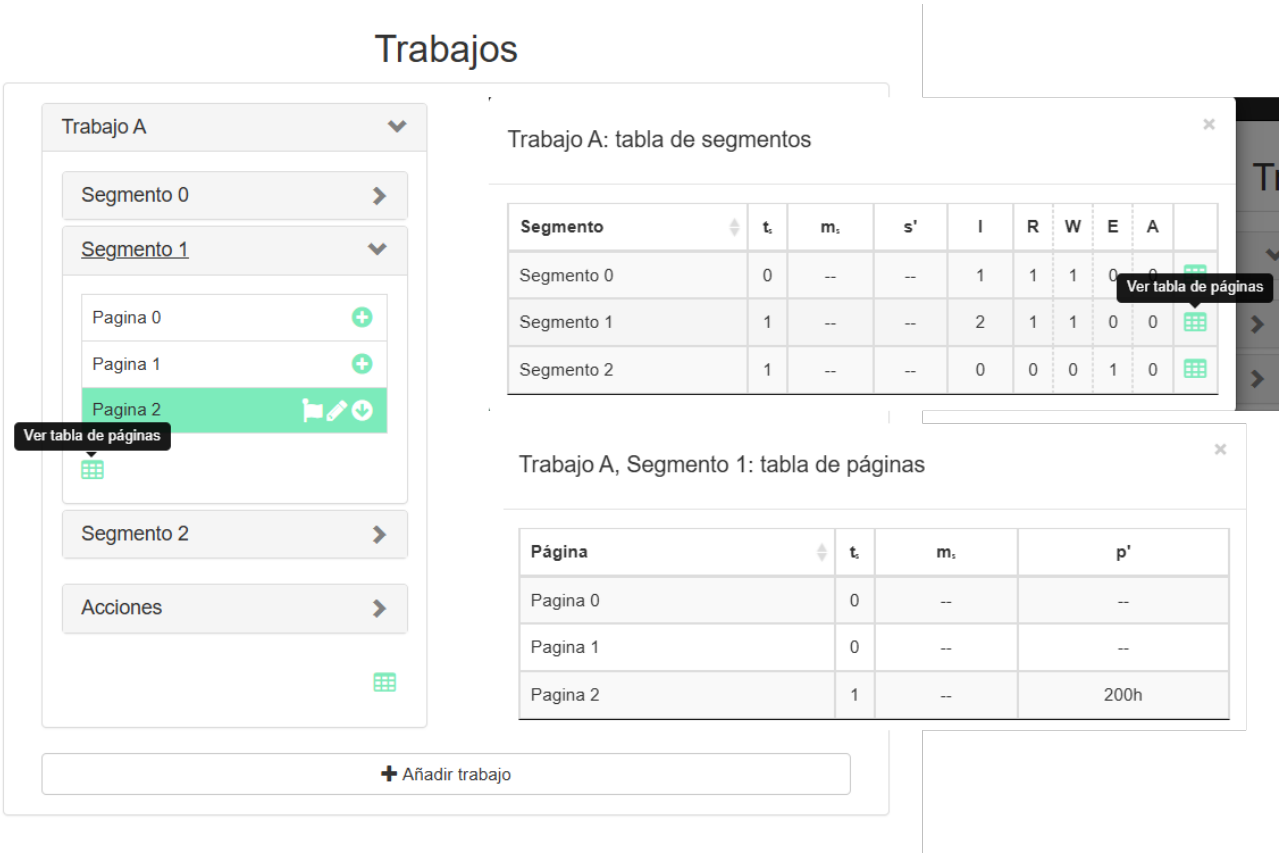


Figure 11.12: Consulta y estructura de la tabla de páginas de un segmento

Fragmentación de tablas

En esta estrategia Segmentación Paginada, la fragmentación de tablas corresponde a la cantidad de memoria principal ocupada por la tabla de segmentos y también por las tablas de páginas asociadas a esos segmentos. El simulador ofrece dos métodos para consultar/calcular dicha fragmentación de tablas a partir del menú "Acciones" del panel del trabajo. Estos métodos son:

1. Calculo automático. La opción "Fragmentación de tablas → Calcular automáticamente". En la ventana que aparece van mostrándose progresivamente los pasos a seguir para obtener la fragmentación que genera la tabla de segmentos, las tablas de páginas y por último la fragmentación de tablas total. Al ir pulsando sobre el botón "Siguiente paso" se irán completando automáticamente el espacio que ocupa los campos de cada tabla. La barra de progreso va mostrando las acciones realizadas de las totales requeridas.
2. Evaluar paso a paso. La opción "Fragmentación de tablas → Evaluar paso a paso" permite que el usuario vaya calculando paso a paso la fragmentación de tablas. La ventana que muestra es la misma que en el método anterior, pero en este caso es el usuario el que deberá ir completando el espacio ocupado por

cada campo de cada tabla. Al ir pulsando el botón "Siguiente paso" puede ocurrir dos situaciones:

1. que los datos introducidos sean correctos y por lo tanto se habilitarán los siguientes para seguir con el cálculo,
2. que los datos introducidos sean incorrectos o no completos. En este caso se marcarán esos datos incorrectos para que el usuario los corrija y poder avanzar.

La figura 11.13 muestra la ventana que se abre en ambos métodos. En el segundo paso, cuando se obtiene la fragmentación de tablas el simulador solicita al usuario que es lo que se almacena en el campo p' de las tablas de páginas, ya que puede ser o bien la dirección real donde comienza el marco que contiene a la página o el número del marco de página que contiene a dicha página.

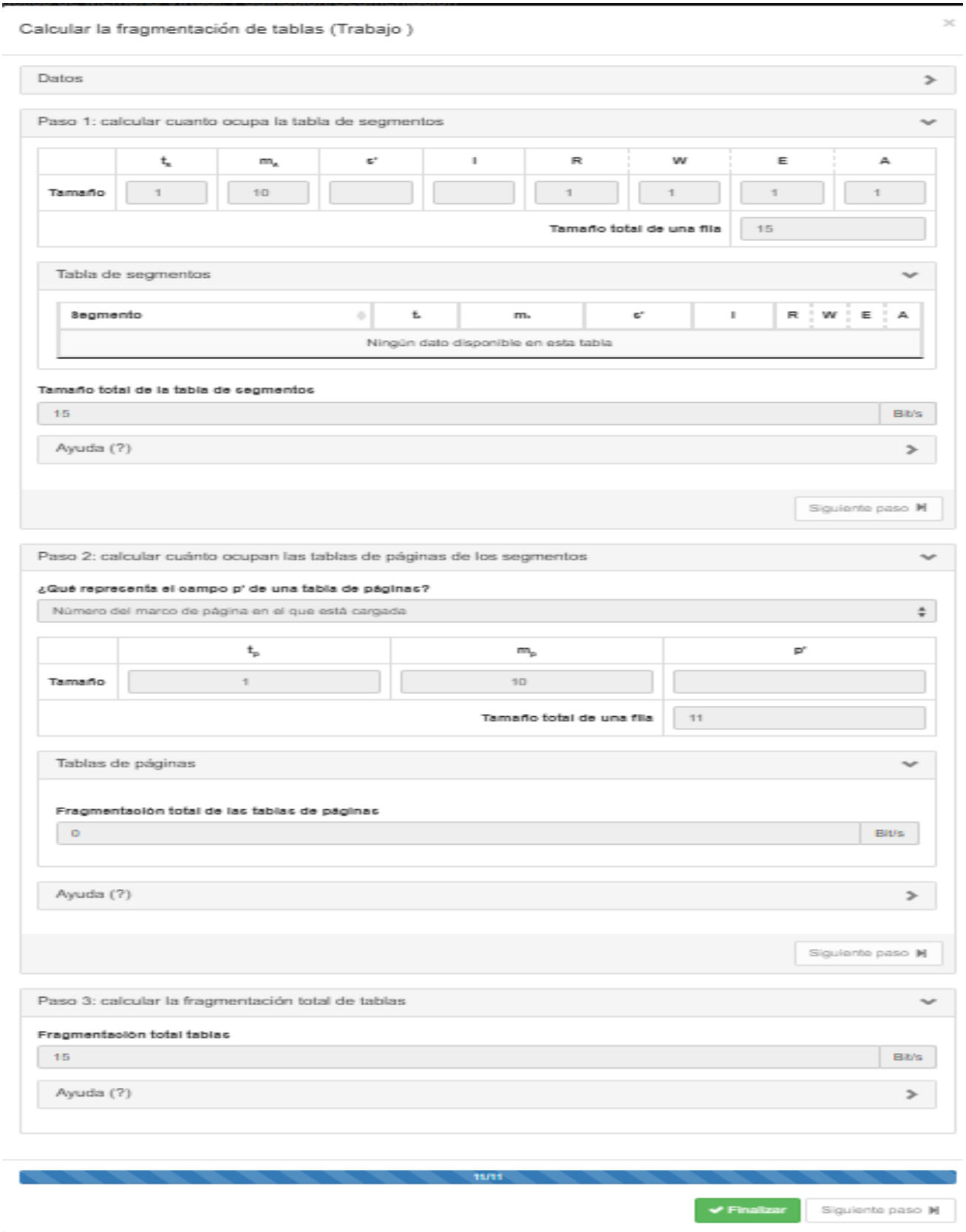


Figure 11.13: Calcular/Evaluar fragmentación de tablas

Traducción de una dirección virtual en una dirección real

Al igual que para obtener la fragmentación de tablas, la aplicación también ofrece dos métodos para traducir una dirección virtual a una dirección real: o la realiza la aplicación automáticamente o el usuario puede ir haciendo los pasos y auto evaluarse. Ambas opciones están en el panel del trabajo seleccionando la opción "Acciones". Una vez introducida la dirección virtual en hexadecimal y las operaciones que se desean realizar sobre el elemento referenciado, los pasos que siguen para hacer la traducción son los siguientes:

1. Se traduce la dirección virtual de hexadecimal a binario para identificar las tres componentes de una dirección virtual, es decir el número de segmento (s) donde está el elemento al que se referencia, el número de página dentro del segmento y el desplazamiento (d) dentro de la página, es decir; la distancia que dista desde el comienzo de la pagina al elemento en cuestión, tal como figura en la figura 11.14.

Figure 11.14: Traducción dv a dr: identificación componentes de dirección virtual

2. Se comprueba si al menos una página del segmento reside en memoria principal. Para ello, se determina la fila de la tabla de segmentos que proporciona la información sobre el segmento en cuestión. De dicha fila se comprueba el valor del bit de residencia t_s (figura 11.15). Si este bit indica que ninguna página del segmento reside en memoria principal, se habrá producido un *Fallo de pérdida de segmento*.
3. Como cada segmento se divide en un número variable de páginas es necesario comprobar que la página pertenece al segmento. Para ello, se comprueba que **l** >= **p**, dónde **l** es el número de páginas menos una en la que se divide el segmento y **p** es el número de página referenciado en la dirección virtual. Esto se refleja en la figura 11.16.

   En el caso de que esta condición no se cumpla se habrá producido un *Fallo de desbordamiento de segmento*, ya que se desea referenciar una página que está más allá del final del segmento.
4. El contenido de cada segmento tiene identidad propia por lo que se pueden controlar las operaciones

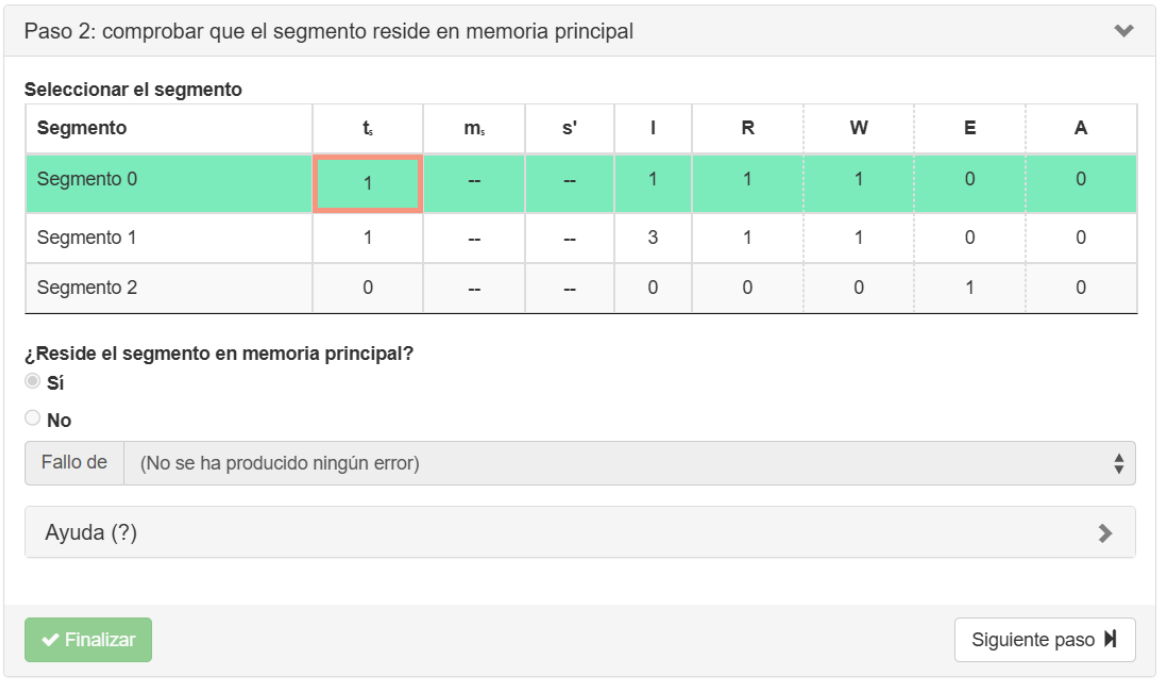


Figure 11.15: Traducción *dv* a *dr*: comprobar si al menos una página del segmento reside en Memoria Principal

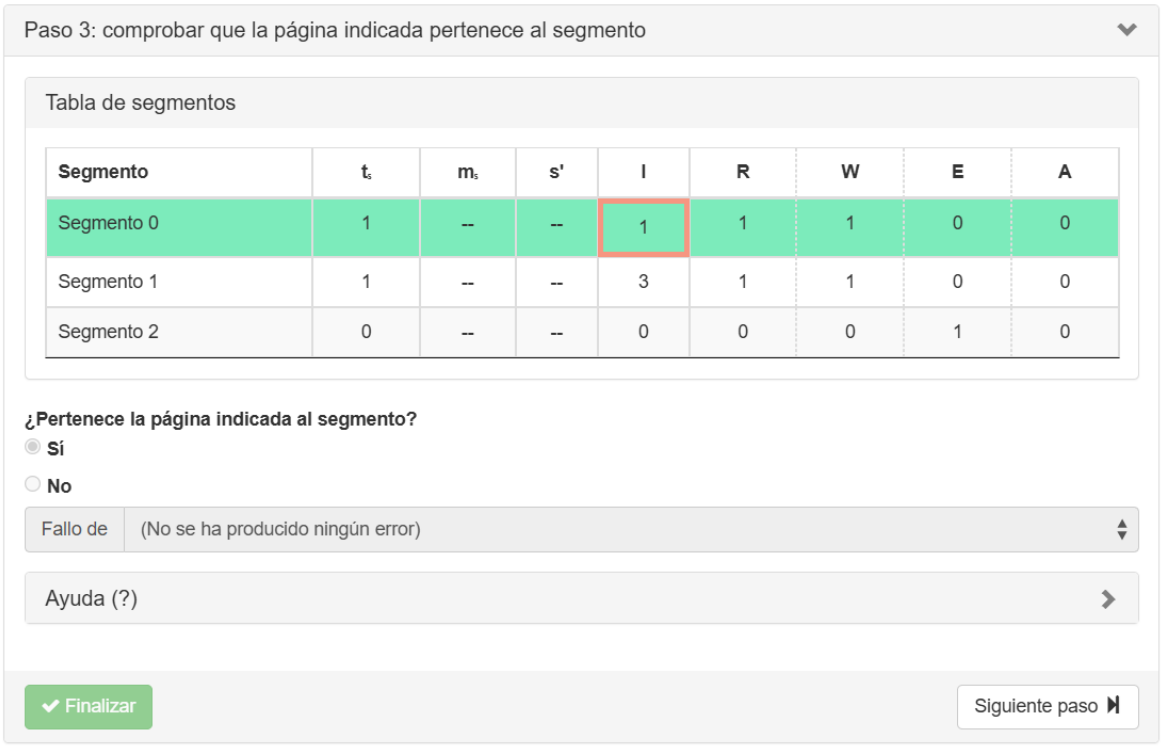


Figure 11.16: Traducción *dv* a *dr*: comprobar si la página pertenece al segmento

que el proceso puede realizar sobre su contenido. Para ello, se usan los bits de protección asociados al segmento referenciado. (Figura 11.17).

Figure 11.17: Traducción dv a dr: comprobar permiso sobre las operaciones solicitadas

En el caso de que alguna de las operaciones solicitadas no se pueda realizar se habrá producido un *Fallo de protección de segmento*, ya que el proceso intenta realizar una acción para la que no tiene permiso.

5. Se comprueba que la página referenciada reside en memoria principal. Para ello, se determina la fila de la tabla de páginas que proporciona la información sobre la página en cuestión. De dicha fila se comprueba el valor del bit de residencia t_p. Si este bit indica que la página no reside en memoria principal, se habrá producido un *Fallo de pérdida de página*. (Figura 11.18).
6. Se obtiene la dirección real donde empieza el marco de página que contiene a la página referenciada. Para ello, se pregunta al usuario que representa el campo p' de la tabla de páginas. Si p' contiene el número del marco de página será necesario obtener la dirección real donde comienza dicho marco de página. Por el contrario, si p' contiene la dirección real donde comienza el marco de página, entonces ya se tiene el dato deseado. Este paso se describe en la figura 11.19).
7. Por último, se obtiene la dirección real deseada concatenando la dirección real donde comienza el marco de página calculada en el paso anterior con el desplazamiento. (Figura 11.20).

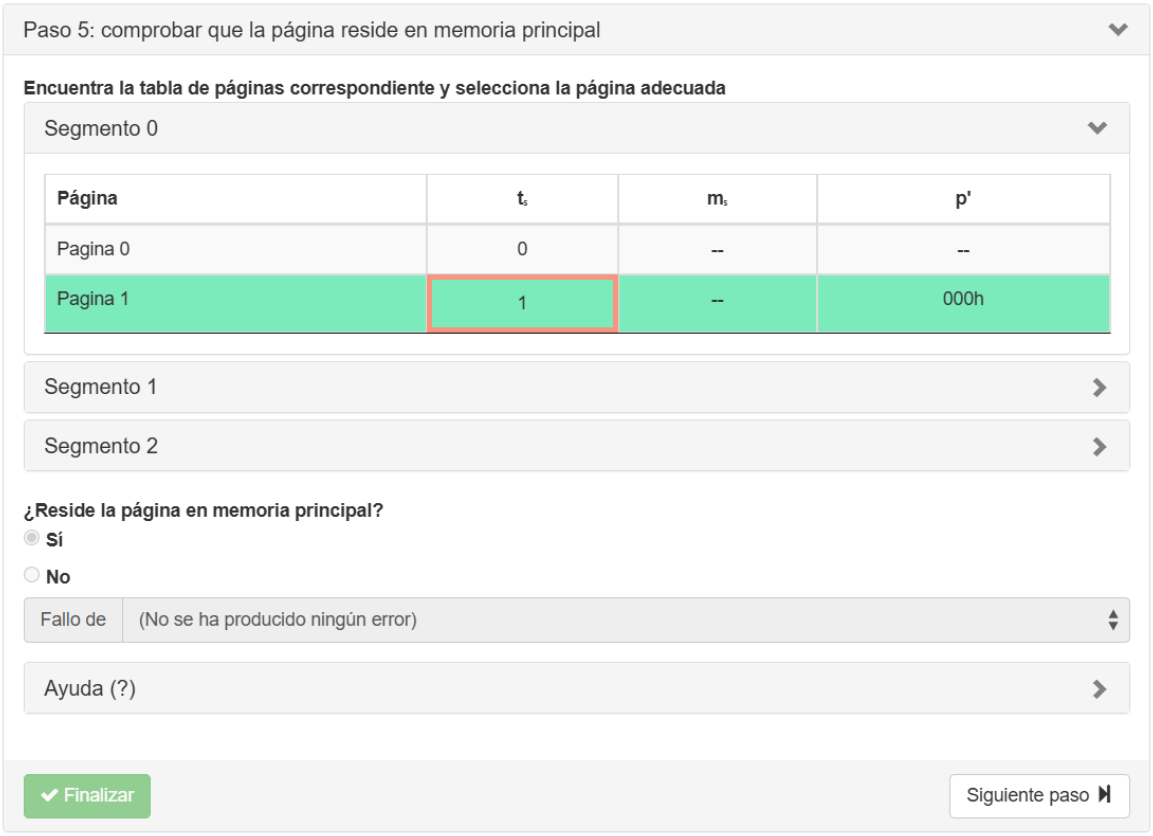


Figure 11.18: Traducción *dv* a *dr*: comprobar que la página reside en memoria principal

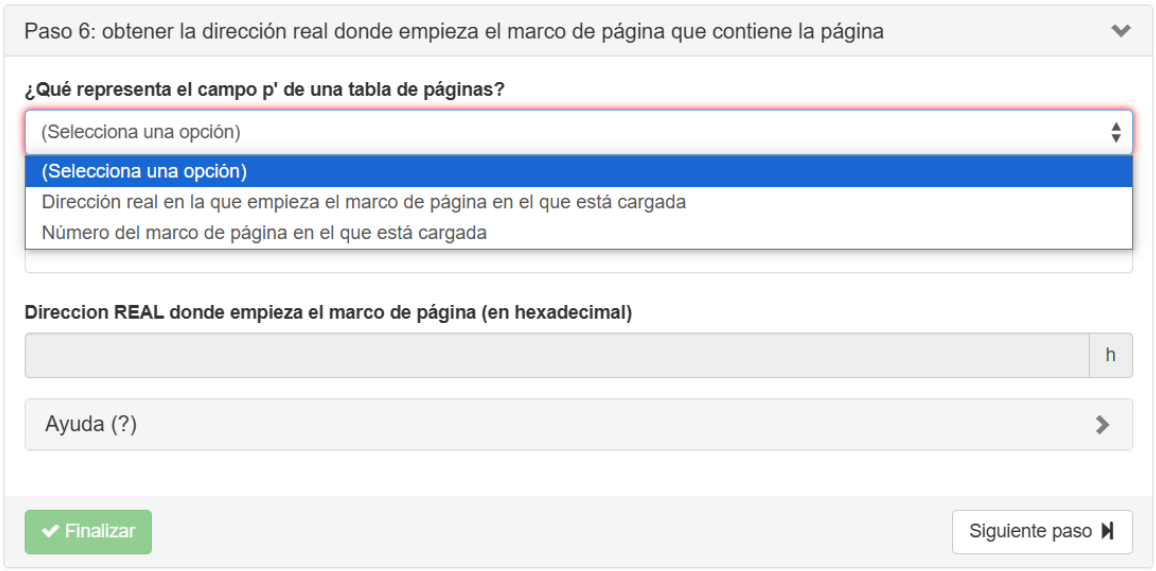


Figure 11.19: Traducción *dv* a *dr*: obtener la dirección real donde comienza el marco de página

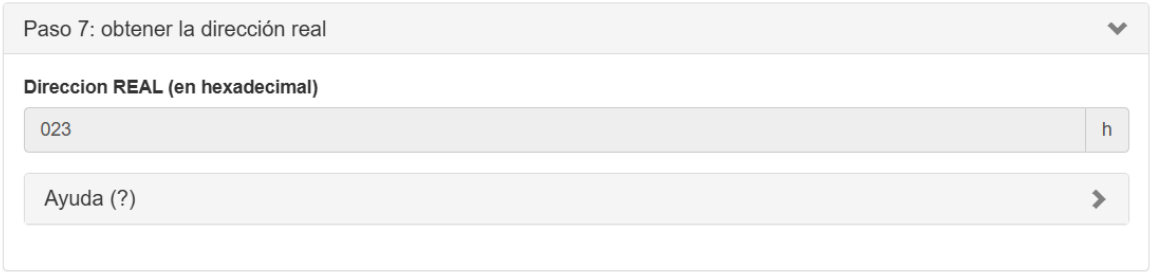


Figure 11.20: Traducción *dv* a *dr*: concatenar *p'* y *d*

11.4

Generación automática de ejercicios con IA generativa

Por último, al igual que en las estrategias para gestionar la memoria virtual Paginación y Segmentación, también se han usando herramientas de inteligencia artificial generativa para obtener las soluciones a una relación de ejercicios que se han elaborado incrementando su complejidad de forma progresiva sobre aspectos relacionados con la Segmentación Paginada. El comienzo de ésta relación contiene el siguiente enunciado común al conjunto de ejercicios.

> Supongamos un sistema segmentado/paginado en donde la memoria principal tiene un tamaño de 1 Mebibyte. Cada dirección virtual ocupa 15 bits repartidos de la siguiente forma:
> - 2 bits para el número de segmento (s),
> - 3 bits para el número de página dentro del segmento (p), y
> - 10 bits para el desplazamiento dentro de la página (d).
>
> El tamaño de cada posición de memoria es de 1 byte.

A continuación se muestran tanto los prompts como las respuestas obtenidas por `ChatGPT`:

En un sistema de segmentación/paginación, la memoria principal tiene un tamaño de **1 Mebibyte (1 MiB)**, y cada página tiene un tamaño de **1 KiB (1024 bytes)**. ¿En cuántos marcos de página se divide la memoria principal?

Resolución:

Para calcular el número de marcos de página en la memoria principal, usamos la fórmula:

$$\text{Número de marcos de página} = \frac{\text{Tamaño de la memoria principal}}{\text{Tamaño de un marco de página}}$$

Sustituyendo los valores:

$$\text{Número de marcos de página} = \frac{1\,\text{MiB}}{1\,\text{KiB}} = \frac{2^{20}\,\text{bytes}}{2^{10}\,\text{bytes}} = 2^{10} = 1024.$$

Respuesta:

La memoria principal se divide en **1024 marcos de página**.

b. ¿Cuántos bits tendrá el campo número de marco de la tabla de páginas, sabiendo que la memoria principal se divide en 1024 marcos de página?

Resolución:

El número de bits necesarios para representar el **número de marco** en la tabla de páginas depende de la cantidad total de marcos en la memoria principal.

Paso 1: Determinar el número de bits para 1024 marcos

Para representar n elementos distintos, se necesitan $\lceil \log_2 n \rceil$ bits. En este caso, el número total de marcos es 1024, por lo que:

$$\text{Número de bits} = \log_2(1024) = 10.$$

Esto se debe a que $2^{10} = 1024$, lo que implica que 10 bits son suficientes para identificar cualquier marco de página en la memoria principal.

Respuesta:

El campo **número de marco** en la tabla de páginas tendrá **10 bits**.

c. ¿Cuál es el tamaño máximo de un segmento, considerando que la memoria principal tiene un tamaño de 1 Mebibyte (1 MiB), la memoria virtual utiliza direcciones de 15 bits, y el tamaño de página es de 1 KiB (1024 bytes)?

Resolución:

1. La dirección virtual tiene 15 bits, de los cuales:
   - **2 bits** se destinan al número de segmento.
   - **3 bits** se destinan al número de página dentro del segmento.
   - **10 bits** se destinan al desplazamiento dentro de la página.
2. Con 2 bits para el número de segmento, se pueden tener hasta $2^2 = 4$ segmentos.
3. Con 3 bits para el número de página dentro de cada segmento, se pueden tener hasta $2^3 = 8$ páginas por segmento.
4. Dado que cada página tiene un tamaño de **1 KiB (1024 bytes)**, el tamaño máximo de un segmento será:

$$\text{Tamaño máximo de un segmento} = 8\,\text{páginas} \times 1\,\text{KiB/página} = 8\,\text{KiB}.$$

Respuesta:

El **tamaño máximo de un segmento es 8 KiB**.

d. ¿Se puede ejecutar un trabajo de 40 Kibibytes (KiB), considerando que la memoria principal tiene un tamaño de 1 Mebibyte (1 MiB), la memoria virtual utiliza direcciones de 15 bits, el tamaño de página es de 1 KiB (1024 bytes), y el sistema está segmentado y paginado?

Resolución:

Características dadas:

1. **Memoria principal:** 1 MiB (2^{20} bytes = **1024 KiB**).
2. **Tamaño de página:** 1 KiB (2^{10} bytes).
3. **Memoria virtual:** 15 bits de dirección virtual, lo que permite direccionar hasta 2^{15} bytes = **32 KiB**.
4. El trabajo a ejecutar es de **40 KiB**.

Paso 1: Verificar el espacio necesario en la memoria virtual

La memoria virtual disponible es de **32 KiB** (por los 15 bits de dirección), lo que no es suficiente para almacenar un trabajo de **40 KiB**, ya que $40\,\text{KiB} > 32\,\text{KiB}$.

Paso 2: Verificar si se puede usar la memoria principal

La memoria principal tiene un tamaño de **1024 KiB**, que es suficiente para almacenar los 40 KiB del trabajo, pero la limitación proviene del espacio de direcciones virtuales.

Conclusión:

Aunque la memoria principal es capaz de almacenar el trabajo de 40 KiB, la **memoria virtual tiene un límite de 32 KiB**, lo que significa que el trabajo no puede ser ejecutado debido a la falta de espacio en la memoria virtual.

Respuesta:

No, no se puede ejecutar un trabajo de 40 KiB debido a la limitación en el tamaño de la memoria virtual (32 KiB).

Apartado e

Enunciado:

Suponer que un trabajo de 21 Kibibytes (KiB) se divide en tres segmentos cuyos tamaños son los siguientes:

- Primer segmento: 7495 bytes.
- Segundo segmento: 7980 bytes.
- Tercer segmento: el resto (el tamaño restante para completar los 21 KiB).

¿Qué cantidad de fragmentación interna y externa se produciría si se acepta dicho trabajo?

Resolución:

Datos dados:

- El trabajo tiene un tamaño de **21 KiB**, lo que equivale a **21 × 1024 = 21504 bytes.**
- Primer segmento: **7495 bytes.**
- Segundo segmento: **7980 bytes.**
- Tercer segmento: **21 KiB - (7495 + 7980) bytes.**

Paso 1: Calcular el tamaño del tercer segmento

$$\text{Tercer segmento} = 21504\,\text{bytes} - (7495 + 7980)\,\text{bytes} = 21504 - 15475 = 6029\,\text{bytes}.$$

El tercer segmento tiene un tamaño de **6029 bytes.**

Paso 2: Fragmentación externa

En un sistema **segmentado y paginado**, **no existe fragmentación externa** debido a que la memoria principal está organizada en **marcos de página.** Los segmentos se dividen en páginas que se almacenan en los marcos de memoria disponibles, evitando huecos entre ellos.

Paso 3: Fragmentación interna

La fragmentación interna ocurre cuando los segmentos no se ajustan completamente a las páginas. Dado que cada página tiene un tamaño de **1 KiB (1024 bytes)**, cada segmento ocupará un número entero de páginas, y si no llena completamente una página, el espacio restante dentro de esa página se considera fragmentación interna.

Cálculos de la fragmentación interna:

- **Primer segmento** (7495 bytes):
  - Número de páginas necesarias: $\lceil \frac{7495}{1024} \rceil = 8 \, \text{páginas}.$
  - Tamaño utilizado por las páginas: $8 \times 1024 = 8192 \, \text{bytes}.$
  - Fragmentación interna para el primer segmento:
    $$8192 - 7495 = 697 \, \text{bytes}.$$
- **Segundo segmento** (7980 bytes):
  - Número de páginas necesarias: $\lceil \frac{7980}{1024} \rceil = 8 \, \text{páginas}.$
  - Tamaño utilizado por las páginas: $8 \times 1024 = 8192 \, \text{bytes}.$
  - Fragmentación interna para el segundo segmento:
    $$8192 - 7980 = 212 \, \text{bytes}.$$
- **Tercer segmento** (6029 bytes):
  - Número de páginas necesarias: $\lceil \frac{6029}{1024} \rceil = 6 \, \text{páginas}$, ya que $\frac{6029}{1024} \approx 5.89.$
  - Tamaño utilizado por las páginas: $6 \times 1024 = 6144 \, \text{bytes}.$
  - Fragmentación interna para el tercer segmento:
    $$6144 - 6029 = 115 \, \text{bytes}.$$

Paso 4: Sumar las fragmentaciones internas

La fragmentación interna total es la suma de las fragmentaciones internas de cada segmento:

$$\text{Fragmentación interna total} = 697 \, \text{bytes} + 212 \, \text{bytes} + 115 \, \text{bytes} = 1024 \, \text{bytes}.$$

Bibliography

[DDC04] H. M. Deitel, P. J. Deitel, and D. R. Choffnes. *Operating Systems*. 3rd. Prentice Hall, 2004.

[Som05] Ian Sommerville. *Ingeniería del Software*. Madrid: Pearson Educación, 2005. Chap. 11.2.2 El modelo cliente servidor, pp. 226–227.

[Tan09] Herbert Tanembaum Andrew S.; Bos. *Modern Operating Systems*. 4th. Pearson, 2009.

[Sta12] William; Stallings. *Operating Systems. Internals and Design Principles*. 7th. Prentice Hall, 2012.

[Aln13] M. Alnoukari. «Simulation for Computer Sciences Education». In: *Communication of the ACS* 6 (2013).

[CLD13] Z. Cataldi, F. J. Lage, and C. Dominighini. «Fundamentos para el uso de simulaciones en la enseñanza». In: *Revista de Informática Educativa y Medios Audiovisuales* 10.17 (2013), pp. 8–16. issn: 1667-8338.

[Cor13] Oracle Corporation. *Java EE 7 Tutorial: JavaServer Faces Technology*. 2013. url: https://docs.oracle.com/javaee/7/tutorial/jsf-develop001.htm.
Recuperado el 2024-10-06.

[SGG18] Abraham Silberschatz, Peter Baer Galvin, and Greg Gagne. *Operating System Concepts, 10th Edition*. Wiley, 2018. isbn: 978-1-118-06333-0. url: http://os-book.com/OS10/index.html.

[Fou19] Eclipse Foundation. *The Jakarta EE 8 Tutorial*. 2019. url: https://eclipse-ee4j.github.io/jakartaee-tutorial/toc.html.
Recuperado el 2024-10-06.

[Rod23] Rubén Yáñez-Martínez; Lorena Otero-Cerdeira; M. Encarnación González-Rufino; Francisco J. Rodríguez-Martínez. «Simulador didáctico de planificación de procesos. ¿A qué algoritmo de planificación corresponde este diagrama de ocupación de la CPU?» In: *X Jornadas Iberoamericanas de Innovación Educativa en el ámbito de las TIC y las TAC*. 2023.

[Jav24] Arquitectura Java. *Usando Java Session en Aplicaciones Web*. 2024. url: https://www.arquitecturajava.com/usando-java-session-en-aplicaciones-web/.
Accedido: 14 de octubre de 2024.

[Ope24] OpenAI. *ChatGPT: Language Model by OpenAI*. https://www.openai.com. 2024.
Recuperado el 2024-10-05.

[Uni] Universidade de Alicante. *Modelo Vista Controlador (MVC)*. url: https://si.ua.es/es/documentacion/asp-net-mvc-3/1-dia/modelo-vista-controlador-mvc.html.
Recuperado el 2024-10-06.