

Simulador de Gestión de Memoria

Hugo J. Guevara, Jesús E. Hernández,

Marlem Martínez, Miryam Salazar

Facultad de Ciencias de la Computación

Benemérita Universidad Autónoma de Puebla

Resumen

El sistema operativo proporciona una interfaz para que las aplicaciones accedan a los servicios del sistema, incluyendo la interfaz de usuario y la interfaz del administrador. Además, el sistema operativo desempeña un papel crucial en la gestión de recursos, abordando la gestión temporal (planificación de procesos, acceso a dispositivos y administración de memoria) y la gestión espacial (gestión de almacenamiento). En resumen, el sistema operativo es esencial para optimizar y coordinar los recursos de la computadora, como procesadores, memoria, dispositivos y archivos, para garantizar un rendimiento eficiente.

1. Introducción

En este artículo, nos adentraremos en las profundidades del mundo de los sistemas operativos, descubriendo cómo cumplen un papel crucial en la gestión de recursos de una computadora y cómo su diseño y configuración son esenciales para un rendimiento óptimo. La primera y tal vez la más reconocible función de un sistema operativo es proporcionar una interfaz de llamadas al sistema. Esto se traduce en una especie de "lenguaje común" entre las aplicaciones y el hardware de la

computadora. Es como si el sistema operativo tradujera los deseos de las aplicaciones en comandos que el hardware pueda entender y ejecutar. A menudo, esta interfaz se presenta en forma de una biblioteca de funciones, muchas de las cuales se encuentran encapsuladas en los lenguajes de programación que utilizamos, como C bajo sistemas como Linux o Windows. No obstante, el sistema operativo no se limita a servir de intérprete entre aplicaciones y hardware. También influye en la forma en que interactuamos con nuestra computadora. Desde el modesto intérprete de comandos hasta las modernas interfaces gráficas de usuario (GUI), la interfaz de usuario que elegimos tiene un impacto significativo en nuestra experiencia informática. En esencia, determina la "personalidad" que percibimos en nuestro sistema operativo, convirtiéndolo en una herramienta más accesible y amigable para el usuario no especializado. Pero hay una capa adicional que a menudo pasa desapercibida: la interfaz del administrador.

Esta es una parte crucial que permite a los administradores del sistema configurar parámetros específicos para ajustar el rendimiento y gestionar usuarios y derechos de acceso. Es una puerta de entrada a la configuración de la computadora y un área donde se toman decisiones críticas para garantizar un funcionamiento óptimo y

seguro del sistema. Pasando a la gestión de recursos, el sistema operativo desempeña un papel insustituible. Imaginemos los recursos como los "ingredientes" necesarios para que cualquier tarea en una computadora se complete. Estos recursos incluyen procesadores, memoria, dispositivos y servicios del sistema, entre otros. La gestión de recursos se divide en dos aspectos fundamentales: la gestión temporal y la gestión espacial. La gestión temporal involucra decisiones sobre cómo se utiliza el tiempo en la computadora. Esto se traduce en la planificación de procesos, la administración de las solicitudes de acceso a dispositivos y la optimización de la memoria para minimizar los tiempos de acceso y los fallos. La eficiencia en la gestión temporal es esencial para garantizar que los procesos se ejecuten de manera suave y sin conflictos. La gestión espacial, por otro lado, se enfoca en cómo se distribuyen y se utilizan los recursos físicos. Esto incluye la administración de dispositivos de almacenamiento, tanto volátiles como permanentes, como la memoria RAM y los discos duros, y la asignación de procesadores a los procesos.

Una administración eficiente del espacio es crucial para garantizar que los recursos estén disponibles cuando se necesitan y se utilicen de manera óptima.

2. Simulador de Gestión de Memoria

La herramienta de simulación ha sido desarrollada en lenguaje Java.

Un simulador de gestión de memoria es una herramienta informática que imita el

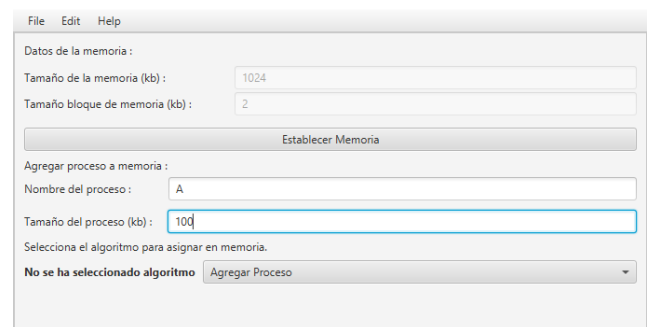
comportamiento de cómo un sistema operativo gestiona la memoria de una computadora. Ayuda a entender y analizar cómo diferentes políticas y algoritmos de gestión de memoria afectan el rendimiento del sistema. Estos simuladores permiten experimentar con asignación de memoria, políticas de reemplazo y otras variables, brindando información valiosa para la investigación y el desarrollo de sistemas operativos.

2.1 Introducción al simulador

El simulador de Gestión de Memoria tiene como objetivo mostrar el funcionamiento de algunos aspectos de la gestión de memoria que intervienen en la ejecución de uno o mas procesos en un entorno de multiprogramación. La herramienta describe la ejecución de un conjunto de procesos mediante el algoritmo de planificación FIFO y LRU, así mismo, buscando los algoritmos que se asignan a memoria, como lo son: primer ajuste, siguiente ajuste, mejor ajuste y peor ajuste.

3. Inicio del simulador principal

El primer paso para poder usar el simulador es establecer los datos de la memoria, agregar los procesos a memoria y seleccionar el algoritmo para asignar en memoria.



The screenshot shows a Java-based application window titled "File Edit Help". It contains several input fields and buttons for configuring the memory management simulator. The "Datos de la memoria:" section has "Tamaño de la memoria (kb):" set to 1024 and "Tamaño bloque de memoria (kb):" set to 2, with an "Establecer Memoria" button below. The "Agregar proceso a memoria:" section has "Nombre del proceso:" set to "A" and "Tamaño del proceso (kb):" set to 100. At the bottom, there is a dropdown menu for "Selecciona el algoritmo para asignar en memoria." currently showing "No se ha seleccionado algoritmo", and an "Agregar Proceso" button.

Figura 1. Datos iniciales llenados

Una vez agregado el proceso, seleccionamos el algoritmo que deseamos usar. Se desglosará toda la información, el mapa de bits, la representación decimal y una tabla que nos muestra el proceso, el inicio y el tamaño.

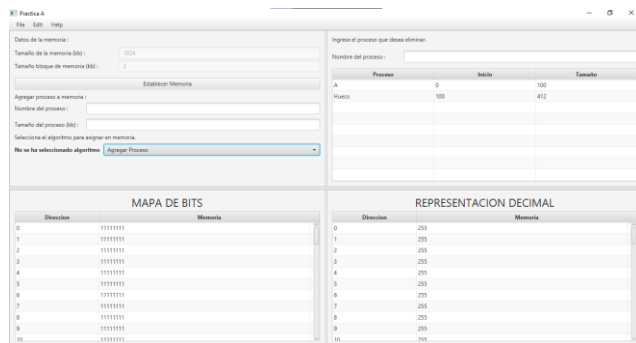


Figura 2. Vista completa del simulador

3.1 Algoritmos

3.1.1 Primer Ajuste

El algoritmo de primer ajuste (First Fit) emerge como una estrategia fundamental en la gestión de la asignación de memoria. Este algoritmo, de naturaleza directa y sencilla, se encarga de asignar bloques de memoria a procesos según un criterio de disponibilidad inmediata.

Cuando un proceso solicita un bloque de memoria, el sistema operativo inicia su búsqueda desde el principio de la memoria principal o la partición asignada al proceso. La asignación se realiza de manera secuencial, asignando el primer bloque disponible que sea lo suficientemente grande para satisfacer las necesidades específicas del proceso.

La simplicidad del algoritmo de primer ajuste facilita su implementación, pero

conlleva desafíos en términos de fragmentación. La fragmentación interna, donde el bloque asignado es más grande que el requerido, y la fragmentación externa, que surge de bloques libres dispersos e inutilizables de manera eficiente, son problemáticas asociadas con este enfoque.

A pesar de estas limitaciones, el algoritmo de primer ajuste sigue siendo utilizado en algunos sistemas operativos debido a su eficiencia en escenarios específicos. Otros algoritmos de asignación de memoria, como el mejor ajuste o el peor ajuste, han sido desarrollados para abordar las cuestiones de fragmentación y optimizar la gestión de la memoria. En última instancia, la elección del algoritmo adecuado dependerá de las características y requisitos particulares del sistema operativo en cuestión.

3.1.2 Mejor ajuste

El algoritmo de mejor ajuste (Best Fit) destaca como una estrategia refinada para la asignación eficiente de bloques de memoria a los procesos. A diferencia del enfoque de primer ajuste, donde se selecciona el primer bloque disponible que cumple con los requisitos del proceso, el algoritmo de mejor ajuste se distingue por su búsqueda meticulosa del bloque de memoria más pequeño que pueda contener al proceso, minimizando así la fragmentación.

Cuando un proceso solicita un bloque de memoria, el sistema operativo emprende la tarea de explorar entre los bloques de memoria libres en busca del más adecuado. En este contexto, "mejor" se refiere al bloque

más pequeño que aún satisface las necesidades dimensionales del proceso en cuestión. La asignación se lleva a cabo seleccionando el bloque que mejor se ajusta al tamaño requerido, dando prioridad a la optimización del espacio y la minimización de la fragmentación.

Este enfoque tiene como objetivo mitigar los problemas asociados con la fragmentación interna y externa. La fragmentación interna se reduce al asignar bloques de tamaño más cercano al necesario, y aunque puede surgir fragmentación externa en forma de pequeños bloques no utilizables entre asignaciones, esta suele ser menor en comparación con otros algoritmos.

A pesar de sus beneficios para reducir la fragmentación, es importante destacar que el algoritmo de mejor ajuste puede ser más complejo de implementar y conlleva cierto costo computacional adicional en comparación con enfoques más simples como el primer ajuste. La elección entre distintos algoritmos de asignación de memoria dependerá de las características particulares del sistema operativo y las necesidades específicas del entorno en el que se implementan.

3.1.3 Peor ajuste

El algoritmo de peor ajuste (Worst Fit) emerge como una estrategia única en la asignación de memoria. A diferencia de enfoques que priorizan bloques más pequeños, este algoritmo selecciona el bloque de memoria libre más grande disponible al

recibir una solicitud de asignación de un proceso.

Cuando un proceso demanda un bloque de memoria, el sistema operativo emprende la tarea de buscar entre los bloques libres, priorizando aquel que sea más extenso. La asignación se realiza seleccionando el bloque de mayor tamaño, con el objetivo de minimizar la fragmentación externa. Esta característica distingue al algoritmo de peor ajuste de otras estrategias, como el mejor ajuste y el primer ajuste.

Si bien el algoritmo de peor ajuste puede reducir la fragmentación externa al asignar bloques grandes y disminuir la probabilidad de dejar pequeños fragmentos no utilizables entre asignaciones, también puede conducir a una asignación menos eficiente. Bloques grandes pueden quedar desocupados, generando fragmentación interna y subutilizando el espacio de memoria.

La elección entre algoritmos de asignación de memoria, como el peor ajuste, mejor ajuste o primer ajuste, depende en gran medida de las características específicas del sistema operativo y de los requisitos particulares del entorno en el que se implementan. En última instancia, la optimización de la gestión de la memoria es esencial para garantizar un rendimiento eficiente del sistema operativo y satisfacer las demandas variadas de las aplicaciones y procesos.

3.1.4 Siguiente ajuste

El algoritmo de siguiente ajuste (Next Fit) destaca como una evolución estratégica del algoritmo de primer ajuste en la asignación de memoria. Este enfoque introduce una mejora al iniciar la búsqueda del próximo bloque de memoria libre desde la última posición en la que se realizó una asignación, en lugar de comenzar desde el principio de la memoria.

Cuando un proceso solicita un bloque de memoria, el sistema operativo inicia la búsqueda desde la última posición de asignación, seleccionando el primer bloque de memoria libre lo suficientemente grande para satisfacer las necesidades del proceso. La asignación se realiza, y la posición de inicio para la próxima búsqueda se actualiza para ser la posición inmediatamente después del bloque recién asignado.

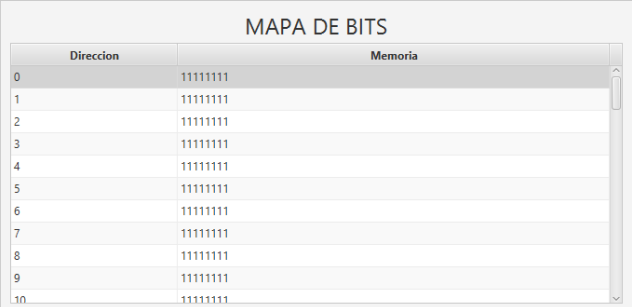
El algoritmo de siguiente ajuste busca mitigar la fragmentación interna al asignar bloques contiguos de memoria. Aunque puede seguir sufriendo de fragmentación externa y no ser tan eficiente como algoritmos más complejos como el de mejor ajuste o peor ajuste, ofrece una mejora con respecto al primer ajuste.

En la elección del algoritmo de asignación de memoria, los desarrolladores y administradores de sistemas deben considerar las características específicas del sistema operativo y los requisitos particulares del entorno. Cada algoritmo tiene sus ventajas y desventajas, y la decisión

final dependerá de la optimización deseada para la gestión eficiente de la memoria.

3.2 Mapa de bits

Un mapa de bits, es una representación compacta y eficiente de las asignaciones de memoria. En este mapa, cada bit corresponde a un bloque de memoria específico. Un bit puede estar establecido (1) si el bloque de memoria está ocupado, o despejado (0) si está libre. Esta representación permite que el sistema operativo realice un seguimiento eficiente de la disponibilidad y ocupación de bloques de memoria.



Direccion	Memoria
0	11111111
1	11111111
2	11111111
3	11111111
4	11111111
5	11111111
6	11111111
7	11111111
8	11111111
9	11111111
10	11111111

Figura 3. Mapa de bits

Cuando un programa solicita memoria al sistema operativo, el gestor de memoria busca en el mapa de bits para encontrar un bloque de memoria contiguo y libre del tamaño requerido. Una vez asignado, el bit correspondiente se establece para indicar que ese bloque está en uso. Cuando la memoria se libera, el bit se borra, indicando que el bloque de memoria está disponible nuevamente.

Este enfoque de mapa de bits para el gestor de memoria es especialmente útil para sistemas con recursos limitados, ya que proporciona una forma eficiente de llevar un

registro del uso de la memoria sin ocupar demasiada memoria adicional para el propio sistema de gestión. Sin embargo, también puede presentar desafíos en términos de fragmentación y asignación eficiente de bloques de memoria contigua. En sistemas operativos modernos, se utilizan diversas técnicas y estructuras de datos para gestionar la memoria de manera más sofisticada y adaptarse a las demandas de las aplicaciones contemporáneas.

3.3 Representación decimal

La representación decimal de un mapa de bits implica la conversión de su secuencia binaria correspondiente a un equivalente en base 10. Cada bit en el mapa de bits, al interpretarse como un dígito binario, contribuye a la formación de un número decimal. Por ejemplo, consideremos un mapa de bits de 8 bits, donde la secuencia binaria '11011010' se traduce a su representación decimal como 218. Esta conversión es crucial en la informática, ya que facilita la interpretación y manipulación de datos almacenados en formato binario, especialmente en el contexto de sistemas operativos y representación gráfica de imágenes.

Así, es como es nuestra tabla de representación decimal se muestra esta conversión.

REPRESENTACION DECIMAL	
Direccion	Memoria
0	255
1	255
2	255
3	255
4	255
5	255
6	255
7	255
8	255
9	255
10	255

Figura 4. Representación decimal

3.4 Eliminar

Aquí, el simulador nos da la opción de eliminar un proceso, indicando que proceso que queremos eliminar

Ingrese el proceso que desea eliminar.

Nombre del proceso :

Proceso	Inicio	Tamaño
A	0	100
B	100	200
C	300	150
Hueco	450	62

Figura 5. Se indica proceso a eliminar

Ingrese el proceso que desea eliminar.

Nombre del proceso :

Proceso	Inicio	Tamaño
A	0	100
B	100	200
Hueco	300	212

Figura 6. Se muestra la tabla sin el proceso

4. Inicio del simulador secundario

El primer paso para hacer uso del simulador es ingresar la cadena de referencia, ya sea

ingresándola manualmente, o bien, tenemos una opción que nos da una cadena de default, posteriormente se selecciona el algoritmo deseado, ya sea FIFO o LRU, así mismo, se selecciona el número de marcos.

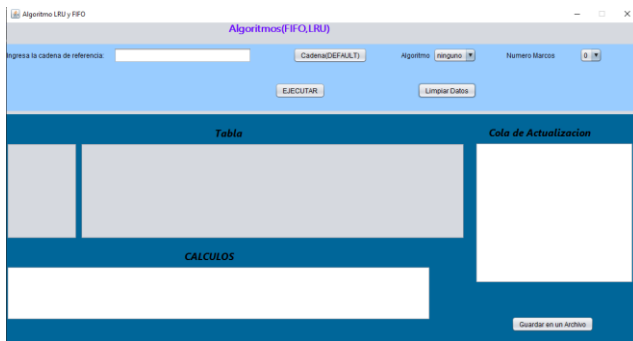


Figura 7. Vista inicial del simulador

4.1 Algoritmos

4.1.1 FIFO

El algoritmo FIFO (First-In-First-Out o Primero en Entrar, Primero en Salir) emerge como una estrategia elemental en la gestión de procesos. Este enfoque sigue el principio simple de asignar prioridad a los procesos según el orden en que llegaron a la cola de listos, de manera que el primer proceso que ingresa es el primero en ser ejecutado.

Cuando los procesos hacen su entrada en el sistema, son encolados en una cola de listos en el orden cronológico de llegada. Al necesitar el sistema operativo seleccionar un proceso para ejecución, opta por el proceso que está al frente de la cola, sin tener en cuenta su complejidad o duración. Una vez seleccionado, el proceso se ejecuta hasta su conclusión o hasta que se requiere alguna operación de entrada/salida.

Este enfoque de "primero en entrar, primero en salir" refleja una filosofía de justicia temporal: los procesos se ejecutan conforme al orden en que ingresaron. Sin embargo, la simplicidad de este algoritmo puede resultar en ciertas limitaciones, ya que no considera la prioridad relativa de los procesos ni su tiempo de ejecución. Esto puede dar lugar a situaciones donde procesos más cortos quedan a la espera detrás de procesos más largos que llegaron primero.

Aunque el algoritmo FIFO es fácil de entender e implementar, su idoneidad depende del contexto. En entornos donde la eficiencia y la equidad en la ejecución de procesos son prioritarias, pueden preferirse otros algoritmos de planificación más avanzados, como Round Robin o aquellos basados en prioridades. La elección entre estos enfoques dependerá de las necesidades específicas y los objetivos de rendimiento del sistema operativo en cuestión.

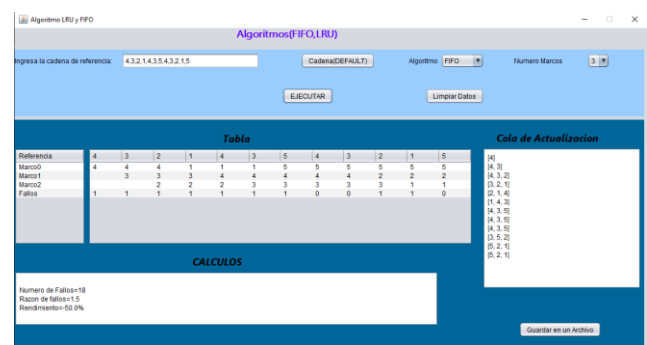


Figura 8. Algoritmo FIFO

4.1.2 LRU

El algoritmo LRU (Least Recently Used o Menos Recientemente Utilizado) destaca como una estrategia esencial para optimizar

el rendimiento y la eficiencia del sistema, especialmente en entornos con memoria virtual.

La premisa fundamental del algoritmo LRU radica en su capacidad para seleccionar la página de memoria que ha sido menos utilizada recientemente cuando es necesario liberar espacio para cargar una nueva página. Esta elección se basa en la intuición de que las páginas que no han sido accedidas en un período más prolongado tienen una menor probabilidad de ser utilizadas nuevamente en el corto plazo.

La implementación del algoritmo LRU implica el seguimiento del uso de cada página en la memoria, ya sea mediante el uso de marcas de tiempo o contadores asociados a cada página. Cuando se requiere espacio y la memoria está llena, el algoritmo identifica la página menos recientemente utilizada y la elimina para dar paso a la nueva página que se va a cargar.

Este proceso de selección cuidadosa de páginas optimiza la gestión de la memoria, reduciendo la probabilidad de fallos de página y mejorando así el rendimiento general del sistema. Sin embargo, es importante destacar que la implementación del algoritmo LRU puede ser más compleja y costosa en términos computacionales en comparación con enfoques más simples como FIFO (First-In-First-Out) o el de reloj.

En resumen, el algoritmo LRU desempeña un papel crucial en la arquitectura de sistemas operativos modernos, especialmente en entornos donde

la eficiencia en la gestión de la memoria es esencial para la ejecución eficiente de procesos y la optimización del rendimiento del sistema.



Figura 9. Algoritmo LRU

4.2 Numero de marcos

El concepto de "número de marcos" se refiere a la cantidad de unidades fundamentales de almacenamiento, conocidas como marcos de página, presentes en la memoria física del sistema. Cada marco de página es un bloque contiguo de memoria física que se utiliza para almacenar una página de datos o instrucciones.

La organización de la memoria física en marcos de página es esencial para facilitar la gestión eficiente de la memoria en sistemas operativos modernos. Estos marcos, numerados secuencialmente, proporcionan una estructura ordenada y definida para almacenar y recuperar datos de manera rápida y efectiva.

El número total de marcos de página disponible en la memoria física es un parámetro crítico que influye directamente en el rendimiento y la capacidad del sistema

operativo. La relación entre el número de marcos y el tamaño de las páginas, que suele ser una potencia de 2, determina la cantidad total de direcciones de memoria física disponibles para los programas y datos.

Una gestión eficiente de los marcos de página es esencial para evitar cuellos de botella en el acceso a la memoria y garantizar un rendimiento óptimo del sistema. La relación entre el número de marcos y el tamaño de las páginas también tiene implicaciones importantes en la capacidad del sistema operativo para manejar múltiples procesos simultáneamente sin incurrir en excesivos fallos de página o tiempos de espera.

El diseño y la configuración del número de marcos son decisiones críticas en el desarrollo de sistemas operativos, ya que deben adaptarse a las demandas específicas de las aplicaciones y asegurar un rendimiento eficiente en una variedad de entornos y escenarios de uso.

4.3 Tabla

En la tabla se muestran las referencias, que son los marcos y los fallos, así como la cadena que ingresamos, el simulador hace los cálculos adecuados de acuerdo con el algoritmo que elegimos (en este ejemplo, LRU) y los números de marcos seleccionados (4).

Tabla												
Referencia	4	3	2	1	4	3	5	4	3	2	1	5
Marcos	4	4	4	4	4	4	4	4	4	4	4	5
Marcos1		3	3	3	3	3	3	3	3	3	3	3
Marcos2			2	2	2	2	5	5	5	5	1	1
Marcos3				1	1	1	1	1	1	2	2	2
Fallos	1	1	1	1	0	0	1	0	0	1	1	1

Figura 10. Tabla de LRU

4.4 Cola de actualización

En la cola de actualización se muestran los datos de la tabla en forma de lista, esto, nos permite visualizar de otra manera la forma en que trabaja el algoritmo.

Cola de Actualizacion												
[4]												
[4, 3]												
[4, 3, 2]												
[4, 3, 2, 1]												
[3, 2, 1, 4]												
[2, 1, 4, 3]												
[1, 4, 3, 5]												
[1, 3, 5, 4]												
[1, 5, 4, 3]												
[5, 4, 3, 2]												
[4, 3, 2, 1]												
[3, 2, 1, 5]												

Figura 11. Cola de actualización de LRU

4.5 Cálculos

El simulador nos muestra los cálculos que se realizaron, como lo son:

- Numero de fallos
- Razón de fallos
- Rendimiento

4.6 Guardar

El botón de guardar en un archivo nos permite guardar en cualquier parte de nuestro dispositivo toda la información que genere el simulador en un archivo txt, o bien, se muestra la opción de "All files".

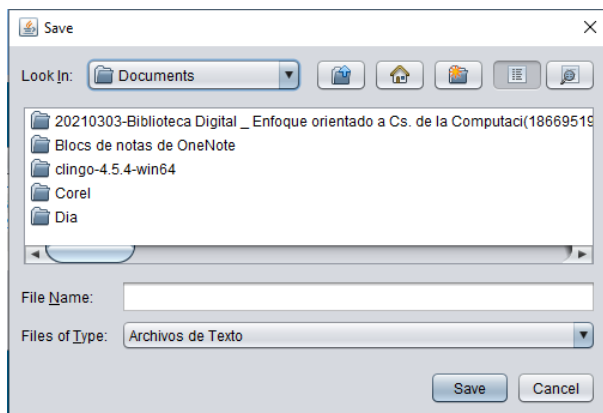


Figura 12. Vista general de la ventana “Guardar”

Posteriormente, muestra un mensaje que indica que el archivo ha sido guardado, y muestra la dirección donde se encuentra.

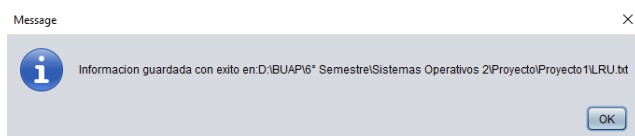


Figura 13. Mensaje de confirmación

Cuando abrimos nuestro archivo, se muestra de la siguiente manera.

```

LRU: Bloc de notas
Archivo  Edición  Formato  Ver  Ayuda
-----Algorithm Lru-----
Referencia| 4 3 2 1 4 3 5 4 3 2 1 5
-----
Marco 0   | 4 4 4 4 4 4 4 4 4 4 5
Marco 1   | 3 3 3 3 3 3 3 3 3 3 3
Marco 2   | 2 2 2 2 5 5 5 5 1 1
Marco 3   | 1 1 1 1 1 1 1 2 2 2
Fallos    | 1 1 1 1 0 0 1 0 0 1 1

Numero de Fallos=8
Razon de fallos=0.666667
Rendimiento=33.333332%
-----Cola de estado-----
[4]
[4, 3]
[4, 3, 2]
[4, 3, 2, 1]
[3, 2, 1, 4]
[2, 1, 4, 3]
[1, 4, 3, 5]
[1, 3, 5, 4]
[1, 5, 4, 3]
[5, 4, 3, 2]
[4, 3, 2, 1]
[3, 2, 1, 5]

```

Figura 14. Información en archivo .txt

5. Conclusiones

En este trabajo, se ha presentado una herramienta dirigida a la simulación de Gestión de Memoria. Dicha herramienta ha sido realizada en lenguaje Java, lo que facilita su utilización en entornos web y su portabilidad a diferentes sistemas operativos. Así mismo proporciona una interfaz visual que favorece su uso por parte de los alumnos.

Nuestro simulador de archivos emerge como una herramienta invaluable para comprender y optimizar la gestión de memoria. Permitiendo la experimentación sin riesgos, este simulador ofrece una visión detallada de cada espacio que se brinda. Desde mapa de bits hasta los algoritmos FIFO y LRU.

La herramienta, invita al alumno a interactuar con la aplicación y permite fácilmente la modificación de algunos de los parámetros, como eliminar un proceso o limpiar datos, así como guardar en un archivo la información generada.

Se espera que con dicha herramienta el aprendizaje y conocimiento de los alumnos se vea beneficiada, para que así, puedan implementar ejercicios de clase o bien, explorarlo por su cuenta.

Bibliografía

- Alia. (2023, 6 mayo). *Gestión de la memoria*. TechEdu.
<https://techlib.net/techedu/gestion-de-la-memoria/>

- Beckerola. (2020, 26 abril). *Gestión de memoria y almacenamiento*. Ordenadores y Portátiles. <https://ordenadores-y-portatiles.com/gestion-de-memoria/>
- *El algoritmo de reemplazo en el sistema de paginación (FIFO, LRU, óptimo) - Programador Clic*. (s. f.). <https://programmerclick.com/article/90603529921/>