TEC | Tecnológico de Costa Rica

Escuela de ingeniería en computación
Ingeniería en computación
IC6600 - Principios de sistemas operativos

Informe Proyecto 2

Simulación MMU

Tim Scarlith
Correo electrónico
Carnet

Josue Torres
Correo electrónico
Carnet

Wilfredo Villegas
Correo electrónico
Carnet

San José, Costa Rica
Mayo 2025

Índice general

Capítulo 1

Introducción

1.1. Gestión de Memoria en los Sistemas Operativos

La gestión de memoria es una de las funciones principales de las cuales se encarga el sistema operativo. Su objetivo es controlar y coordinar el uso de la memoria principal entre los distintos procesos del sistema, optimizando el uso de los recursos y optimizando el sistema en su totalidad. Dentro de las tareas que le competen se encuentra la asignación y liberación de memoria, seguimiento de las direcciones utilizadas y la implementación de políticas de reemplazo de páginas.

1.2. Paginación y Memoria Virtual

La paginación es una técnica de gestión de memoria que permite dividir la memoria física y los espacios de direcciones de los procesos en bloques de tamaño fijo llamados páginas. La memoria virtual, por otro lado, permite a los procesos utilizar más memoria de la que físicamente está disponible, desplazando páginas hacia y desde el disco según sea necesario. Estas técnicas permiten una mayor eficiencia en la ejecución de procesos y una mejor utilización de la memoria del sistema..

Capítulo 2

Desarrollo

2.1. Algoritmo Óptimo (OPT)

2.1.1. Descripción

Primeramente, se elaboró una versión prototipo del algoritmo óptimo, basándose únicamente en el contenido obtenido en las lecciones del curso de Análisis de Algoritmos. Sin embargo, su abordaje a la implementación no era del todo viable con respecto al parseo de operaciones dentro del archivo de procesos. Por ello, se decidió buscar una implementación más adecuada del algoritmo OPT.

Dado que se conoce de antemano la secuencia completa de referencias a páginas, es posible calcular con exactitud qué página será utilizada más adelante. Para ello, se extrae la página lógica previamente requerida y se recorre la lista de operaciones futuras —desde la operación actual hasta el final— con el fin de hallar la próxima referencia a dicha página. Como resultado, se puede determinar que la página con la mayor distancia hasta su próxima referencia debe ser reemplazada. Si ninguna página tiene una referencia futura, se escoge por defecto el primer marco.

En esta implementación, la constante futureOperations se encarga de almacenar las referencias que se utilizarán en la simulación. El índice currentOperationIndex indica la distancia entre la operación actual y las futuras. Por su parte, los resultados del proceso de carga y liberación de información de las páginas se administran mediante la variable victimFrameId. (Int-Main, s.f.)s

2.1.2. Implementación

```
1  export const optAlgorithm: PageReplacementAlgorithmFn = (context) => {
2    const { ramFrames, mmu, futureOperations, currentOperationIndex } = context;
3    let victim: PageFrame | null = null;
4    let maxDistance = -1;
5
6    for (const frame of ramFrames.filter(f => f.isOccupied)) {
7      const page = getLogicalPageFromFrame(frame, mmu);
8      let dist = Infinity;
9      for (let i = currentOperationIndex; i < futureOperations.length; i++) {
10       if (futureOperations[i].type === 'use'
11           && futureOperations[i].ptrId === page!.ptrId) {
12         dist = i - currentOperationIndex;
13         break;
```

```
14          }
15        }
16        if (dist > maxDistance) {
17          maxDistance = dist;
18          victim = frame;
19        }
20      }
21      if (!victim) victim = ramFrames.filter(f => f.isOccupied)[0];
22      return {
23        victimFrameId: victim.frameId,
24        victimLogicalPageId: victim.logicalPageId
25      };
26    };
```

2.2. FIFO

2.2.1. Descripción

En nuestra implementación se decidio que cada vez que se carga una página, se anota la marca de tiempo la cual es llevado en las estructuras PageFrame que implementan un campo loadedTimestamp para asignar los tiempos. Al requerirse un remplazo se recorren todos los espacios en la RAM ocupados y se comparan los loadedTimestamp para verificar cual es el más viejo y por ende el que se debe remplazar(GeeksforGeeks, 2024b).

2.2.2. Implementación

```
1   export const fifoAlgorithm: PageReplacementAlgorithmFn = (context) => {
2     let oldestFrame: PageFrame | null = null;
3
4     for (const frame of context.ramFrames) {
5       if (frame.isOccupied &&
6           (!oldestFrame ||
7            frame.loadedTimestamp! < oldestFrame.loadedTimestamp!)) {
8         oldestFrame = frame;
9       }
10    }
11    if (!oldestFrame) throw new Error("[FIFO] No hay marcos ocupados...");
12
13    return {
14      victimFrameId: oldestFrame.frameId,
15      victimLogicalPageId: oldestFrame.logicalPageId
16    };
17  };
```

2.3. LRU

2.3.1. Descripción

Cada vez que se accede a una página, se actualiza ramLastAccessTimestamp,(reutilizandolos). En caso de que se nececite remplazo el algoritmo busca en todos los marcos ocupados, se compara las paginas y se remplaza la que tenga menor ramLastAccessTimestamp.

2.3.2. Implementación

```
export const lruAlgorithm: PageReplacementAlgorithmFn = (context) => {
  let leastRecentFrame: PageFrame | null = null;
  let minTime = Infinity;
  for (const frame of context.ramFrames) {
    if (frame.isOccupied) {
      const page = getLogicalPageFromFrame(frame, context.mmu);
      if (page && page.ramLastAccessTimestamp! < minTime) {
        minTime = page.ramLastAccessTimestamp!;
        leastRecentFrame = frame;
      }
    }
  }
  if (!leastRecentFrame) {
    throw new Error("[LRU] No hay marcos ocupados para reemplazar.");
  }
  return {
    victimFrameId: leastRecentFrame.frameId,
    victimLogicalPageId: leastRecentFrame.logicalPageId
  };
};
```

2.4. MRU

2.4.1. Descripción

La implementación es muy similar a la del LRU, se comparar ramLastAccessTimestamp y seleccionar la página con timestamp mayor. Tambien se realiza un ajuste a la condición de comparación para expulsar la página más reciente (GeeksforGeeks, 2024a).

```
export const mruAlgorithm: PageReplacementAlgorithmFn = (context) => {
  let mostRecentFrame: PageFrame | null = null;
  let maxTime = -1;
  for (const frame of context.ramFrames) {
    if (frame.isOccupied) {
      const page = getLogicalPageFromFrame(frame, context.mmu);
      if (page && page.ramLastAccessTimestamp! > maxTime) {
        maxTime = page.ramLastAccessTimestamp!;
        mostRecentFrame = frame;
      }
    }
  }
  if (!mostRecentFrame) throw new Error("[MRU] No hay marcos ocupados...");
  return {
    victimFrameId: mostRecentFrame.frameId,
    victimLogicalPageId: mostRecentFrame.logicalPageId
  };
};
```

2.5. Second Chance

2.5.1. Descripción

El algoritmo Second Chance es una evolución del enfoque FIFO (First-In, First-Out) para la gestión de reemplazo de páginas en sistemas de memoria virtual. Su diseño aborda una limitación crítica de FIFO: la posible expulsión de páginas activamente utilizadas debido a su rigidez temporal. A continuación, se describe su operación formal, ventajas comparativas y aplicabilidad en contextos contemporáneos. (GeeksforGeeks, 2018)

2.5.2. Mecanismo

El algoritmo gestiona el reemplazo de páginas mediante los siguientes componentes y pasos:

Estructura de Datos

- **Cola circular**: Almacena los marcos de memoria en orden FIFO.
- **Bit de referencia**: Un bit asociado a cada marco ($R \in \{0, 1\}$).
- **Puntero cíclico**: Indica la posición actual para iniciar búsquedas.

Proceso de Reemplazo

Sea P el puntero actual y n el número de marcos. Para cada fallo de página: (GeeksforGeeks, 2018)

1. Iniciar búsqueda desde P en la cola circular.
2. Para cada marco i desde P hasta $P + n - 1$ (módulo n):
   - Si $R[i] = 0$:
     - Reemplazar la página en el marco i
     - Actualizar puntero: $P \leftarrow (i + 1) \mod n$
     - Terminar búsqueda
   - Si $R[i] = 1$:
     - Resetear bit: $R[i] \leftarrow 0$
     - Continuar búsqueda
3. Si todos los bits eran 1, repetir el ciclo hasta encontrar $R = 0$.

Actualización de Referencias

Para cada acceso a memoria:

$$R[\text{marco\_accedido}] \leftarrow 1 \quad \text{(Siempre que la página esté en memoria)}$$

Complejidad Computacional El algoritmo garantiza:(GeeksforGeeks, 2018)

- Tiempo promedio por reemplazo: $O(1)$ en cargas balanceadas
- Peor caso: $O(n)$ cuando todos los bits son 1
- Espacio adicional: n bits (donde n = número de marcos)

2.5.3. Implementación

```
export const scAlgorithm: PageReplacementAlgorithmFn = (context) => {
  const numFrames = context.ramFrames.length;
  let currentHandPos = context.scHandPosition === undefined ? 0 : context.
      scHandPosition;
  const pagesToClearRBit: string[] = [];

  while (true) {
    const frame = context.ramFrames[currentHandPos];
    if (frame.isOccupied) {
      const logicalPage = getLogicalPageFromFrame(frame, context.mmu);
      if (logicalPage) {
        if (logicalPage.referencedBit) {
          pagesToClearRBit.push(logicalPage.id);
          currentHandPos = (currentHandPos + 1) % numFrames;
        } else {
          return {
            victimFrameId: frame.frameId,
            victimLogicalPageId: frame.logicalPageId,
            nextScHandPosition: (currentHandPos + 1) % numFrames,
            pagesWhoseRBitShouldBeCleared: pagesToClearRBit,
          };
        }
      } else {
         currentHandPos = (currentHandPos + 1) % numFrames; // Avanzar para evitar
             bucle infinito
      }
    } else {
        // Marco no ocupado, la simulaci n no deber a llamar a reemplazo si hay
            marcos libres.
        // Si llega aqu , es un error o un marco vac o en medio del escaneo.
        currentHandPos = (currentHandPos + 1) % numFrames;
    }

    const occupiedFrameCount = context.ramFrames.filter(f => f.isOccupied).length;
    if (pagesToClearRBit.length >= occupiedFrameCount && occupiedFrameCount > 0) {
        const originalHandFrame = context.ramFrames[context.scHandPosition ===
            undefined ? 0 : context.scHandPosition];
         if (!originalHandFrame.isOccupied) throw new Error("[SC] Error: Fallback de
             SC a marco no ocupado.");
        return {
            victimFrameId: originalHandFrame.frameId,
            victimLogicalPageId: originalHandFrame.logicalPageId,
            nextScHandPosition: ( (context.scHandPosition === undefined ? 0 :
                context.scHandPosition) + 1) % numFrames,
            pagesWhoseRBitShouldBeCleared: pagesToClearRBit.filter(id => id !==
                originalHandFrame.logicalPageId), // No limpiar el R bit de la
                v ctima
```

```
40            };
41        }
42    }
43  };
```

2.6. RND (Random)

2.6.1. Descripción

El algoritmo de reemplazo aleatorio de páginas reemplaza cualquier página en memoria aleatoriamente. No sigue un patrón ni una regla específicos. Esto lo hace fácil de implementar, pero impredecible. La idea principal es elegir una página para reemplazar sin considerar el orden ni la frecuencia de acceso. (herovided, 2024) (systems, s.f.)

Ventajas

- Fácil de implementar.
- Sin gastos adicionales para el seguimiento de páginas.

Desventajas

- Rendimiento impredecible.
- Puede llevar a decisiones de reemplazo de páginas deficientes.

2.6.2. Implementación

```
1  export const rndAlgorithm: PageReplacementAlgorithmFn = (context) => {
2    const occupiedFrames = context.ramFrames.filter(f => f.isOccupied);
3    if (occupiedFrames.length === 0) {
4      throw new Error("[RND] No hay marcos ocupados para elegir una v ctima.");
5    }
6    const randomIndex = Math.floor(context.rng() * occupiedFrames.length);
7    const victimFrame = occupiedFrames[randomIndex];
8    return {
9      victimFrameId: victimFrame.frameId,
10     victimLogicalPageId: victimFrame.logicalPageId,
11   };
12 };
```

Capítulo 3

Descripción del proyecto

El presente proyecto consiste en una implementación del un sistema de simulación de MMU (Memory Management Unit) desarrollado con javascript con la librería React. A continuación se le indicarán los pasos de descarga, compilación, ejecución y uso del sistema, para esto se debe tener instalado previamente el software controlador de versiones llamado **git** y el software **nodejs**.

3.1. Descarga

Descargue el proyecto que está alojado en github por medio de este enlace: `https://github.com/JosueTorresN/simulacion-mmu` o clone el repositorio por medio del siguiente comando:

```
1    git clone https://github.com/JosueTorresN/simulacion-mmu
```

Una vez descargado, acceda a la carpeta y ejecute el siguiente comando en la terminal: **git checkout josue** y seguido de este otro comando: **npm install** dentro del directorio del proyecto para instalar las dependencias necesarias para ejecutarlo.

```
PS C:\Users\Usuario\Downloads\IC-6600-Sistemas Operativos> git clone https://github.com/JosueTorresN/simulacion-mmu
Cloning into 'simulacion-mmu'...
remote: Enumerating objects: 53, done.
remote: Counting objects: 100% (53/53), done.
remote: Compressing objects: 100% (38/38), done.
remote: Total 53 (delta 21), reused 42 (delta 12), pack-reused 0 (from 0)
Receiving objects: 100% (53/53), 76.21 KiB | 609.00 KiB/s, done.
Resolving deltas: 100% (21/21), done.
PS C:\Users\Usuario\Downloads\IC-6600-Sistemas Operativos> cd .\simulacion-mmu\
PS C:\Users\Usuario\Downloads\IC-6600-Sistemas Operativos\simulacion-mmu> git checkout josue
branch 'josue' set up to track 'origin/josue'.
Switched to a new branch 'josue'
PS C:\Users\Usuario\Downloads\IC-6600-Sistemas Operativos\simulacion-mmu> npm i

added 272 packages, and audited 273 packages in 12s

58 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
PS C:\Users\Usuario\Downloads\IC-6600-Sistemas Operativos\simulacion-mmu>
```

Figura 3.1: Descarga y ejecución de comandos

3.2. Ejecución

A continuación siguiendo en la misma terminal, ejecuta el siguiente comando: **npm run dev** en la que realiza la ejecución. Luego, dirijase al navegador con la dirección **http://localhost:5173**

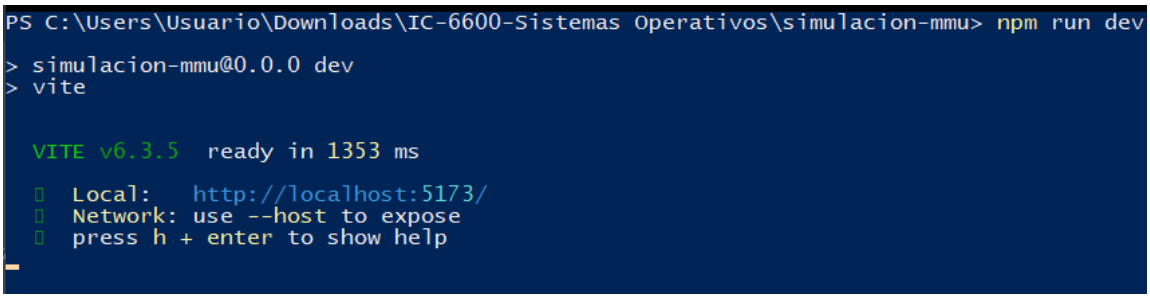

Figura 3.2: Ejecución de comandos para iniciar el sitio web

3.3. Uso

A continuación, se muestra la ventana de configuración del simulador de algoritmos de paginación, herramienta diseñada para comparar el rendimiento del algoritmo óptimo (OPT) frente a otros algoritmos comunes de reemplazo de páginas como:

- FIFO (First-In, First-Out)
- LRU (Least Recently Used)
- MRU (Most Recently Used)
- SC (Second Chance)
- RND (Random)

3.3.1. Componentes de la Interfaz

- **Semilla (Seed):** Campo editable donde se puede ingresar una cadena para generar datos reproducibles. Botón 'Generar Aleatoria' crea una nueva semilla al azar.
- **Algoritmo a Simular:** Seleccione el algoritmo que desea comparar contra OPT. Por defecto, está seleccionado FIFO.
- **Fuente de Operaciones:**
  - **Cargar Archivo:** Permite cargar un archivo con las operaciones a simular.
  - **Generar Nuevas:** Genera operaciones aleatorias según los parámetros seleccionados.

3.3.2. Parámetros de Generación

Configuración para crear un nuevo conjunto de operaciones:

- **Número de Procesos (P):** Cantidad de procesos simultáneos a simular.
- **Cantidad de Operaciones (N):** Total de accesos a memoria a simular. Por ejemplo: 500,1000 o 5000 operaciones

Una vez definidos los parámetros, haga clic en el botón amarillo "Generar y Descargar Archivo"para obtener el archivo con las operaciones generadas.

Después de lo anterior, presionamos el botón de Ïniciar simulación.[en] la cual se dirige a la siguiente página en la que se ejecuta la simulación

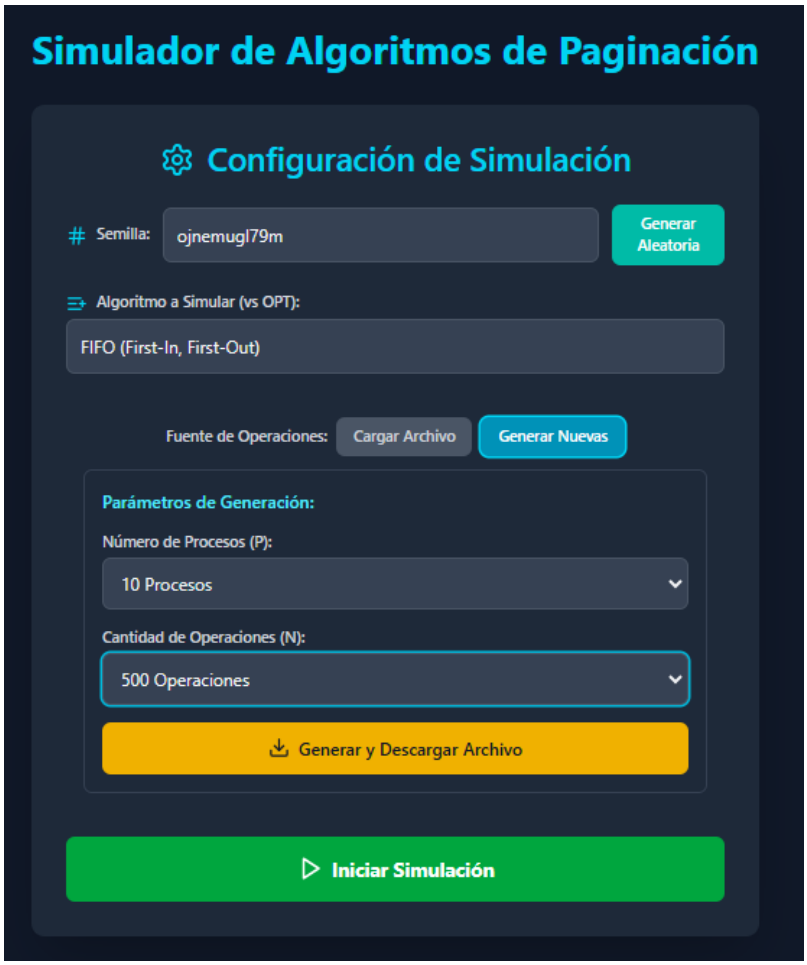

Figura 3.3: configuración inicial del simulador

Capítulo 4

Análisis de resultados

4.1. Contexto

Para este proyecto se propuso realizar un análisis de resultados de los datos proporcionados por los algoritmos propuestos,(RND, SC, FIFO, LRU, MRU), contra el optimo, a modo de comparativa en base a un archivo auto generado por el algoritmo de aleatoriedad que implementa el programa. Este archivo tiene aproximadamente 1000 operaciones y se prueba para todos los algoritmos.

4.2. Pruebas

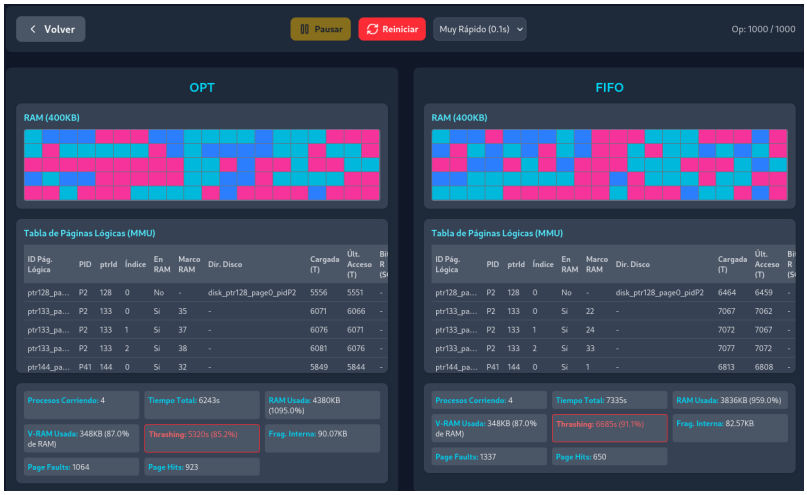

Figura 4.1: Comparativa del óptimo con el FIFO

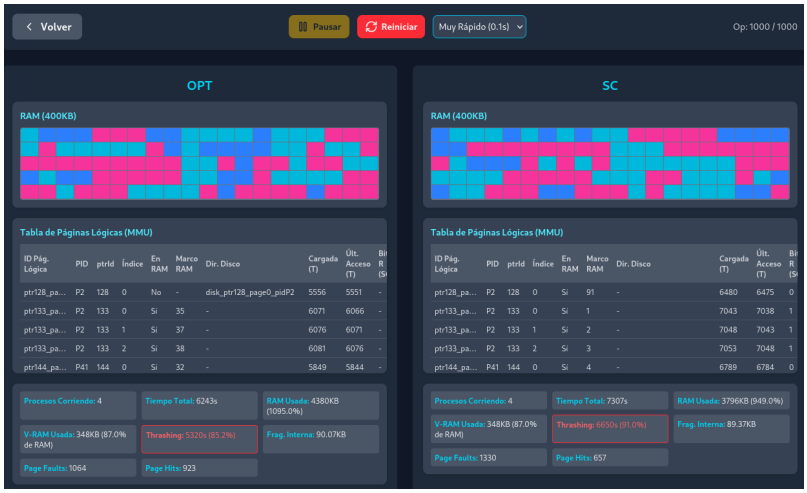

Figura 4.2: Comparativa del óptimo con el Second Chance

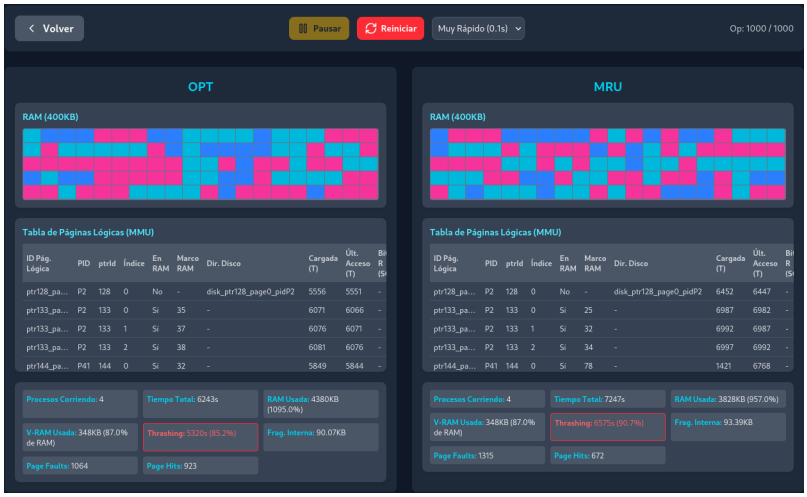

Figura 4.3: Comparativa del óptimo con el MRU

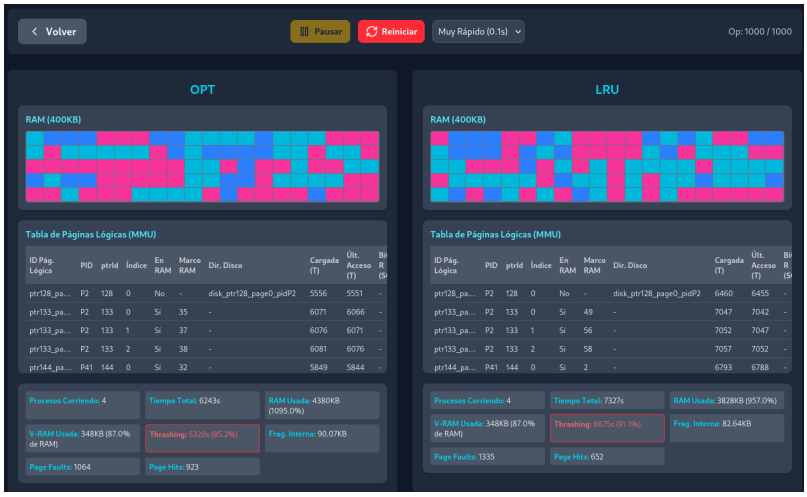

Figura 4.4: Comparativa del óptimo con el LRU

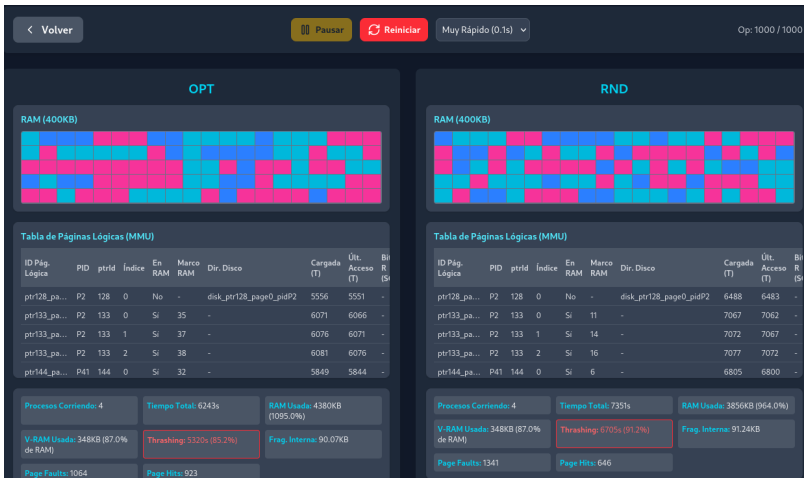

Figura 4.5: Comparativa del óptimo con el RND

4.3. Resultados

En términos generales en base a las pruebas realizadas se encontró que el algoritmo que mostró mejores capacidades en comparación al optimo es el MRU y el peor fue el RND. Estos se puede ver mejor analizando los siguientes resultados de los experimentos de comparación de los experimentos de aciertos de paginas, errores de pagina y thrashing:

4.3.1. Resultados de errores de pagina

Dentro de los resultados optenidos del experimentos se puede concluir que el algoritmo con menor errores de pagina en comparación al optimo es el MRU con una diferencia de 251 errores de pagina. Por otra parte el peor a nivel de errores de pagina es el RND con una diferencia de 277 errores detectados.

4.3.2. Resultados de aciertos de paginas

También se realizó un análisis de los page hits podemos, dónde se busca un número mayor de aciertos se encontró que el algoritmo MRU también fue el que manejó esta métrica de mejor manera con una diferencia con el optimo de 251 aciertos de pagina. A su vez el algorimo con peor comportamiento en esta metrica tambien fue el RND dando una diferencia de 277 aciertos.

4.3.3. Resultados de thrashing

Analisando los resultados de las pruebas de thrashing, se obtuvo un total de 90,7 % equivalente a 6575s de thrashing en MRU con una diferencia de 5,5 % con respecto al optimo que tuvo como resultados 85,2 % equivalestes a 5320s, mientras que el peor resultado lo dio RND con 91,2 % equivalente a 6705s con una diferencia de 6 %

Capítulo 5

Conclusiones

- **MRU (Most Recently Used) presenta menor thrashing en escenarios específicos:** El algoritmo MRU (con un 90.7 % de thrashing), supera claramente a FIFO (91.1 %), RND (91.2 %), SC (91.0 %) y LRU (91.1 %). Esto sugiere que, cuando el patrón de acceso favorece reutilizar páginas recientes, MRU minimiza las recargas, reduciendo la sobrecarga del sistema.
- **RND (Random) es el peor algoritmo en eficiencia:**

  Con un thrashing del 91.2 % y 1341 page faults, su naturaleza aleatoria no prioriza páginas críticas, generando más intercambios innecesarios. Esto lo hace inviable en entornos con alta demanda de memoria, donde la predictibilidad es clave.
- **Algoritmo optimo como punto de referencia:** El uso del algoritmo Óptimo como referencia permitió medir la eficiencia de los demás algoritmos. Al no ser viable en la práctica, pero sí ideal en simulación, su inclusión en las comparaciones ofreció una base para identificar el rendimiento relativo de MRU, FIFO, LRU, SC y RND. Sinenmbargo esto esta altamente ligado a la implementación del algoritmo optimo que se haya realizado, debido a que de ser implementado de una manera errónea este puede lanzar resultados más deficientes que los demás algoritmos o por el contrario, con una implementación más depurada es probable alcanzar mejores resultados en las comparativas.
- **Elección del algoritmo:** Para nuestros experimentos el algoritmo que mejor se acerco a los resultado proporcionado por el optimo fue el MRU. Senembargo no se puede descartar la posibilidad de que para otros escenarios algoritmos como LRU o Second Chance podrían ser más eficientes. Lo cual nos muestra que dependiendo de la tarea la algún algoritmo pude mostrar mejor margen de eficiencia que los demás.

Referencias

GeeksforGeeks. (2018, diciembre). *Second chance (or clock) page replacement policy.* https://www.geeksforgeeks.org/second-chance-or-clock-page-replacement-policy/. (Accessed: 2025-5-20)

GeeksforGeeks. (2024a). *Program for k most recently used (mru) apps.* https://www.geeksforgeeks.org/program-for-k-most-recently-used-mru-apps/. (Accessed: 2025-5-19)

GeeksforGeeks. (2024b, junio). *Program for page replacement algorithms — set 2 (fifo).* https://www.geeksforgeeks.org/program-page-replacement-algorithms-set-2-fifo/. (Accessed: 2025-5-19)

herovided. (2024, junio). *Page replacement algorithms in OS: Types and examples.* https://herovired.com/learning-hub/blogs/page-replacement-algorithms-in-os/. (Accessed: 2025-5-20)

Int-Main. (s.f.). *Operating systems.* https://github.com/int-main/Page-Replacement-Algorithms-in-Python/blob/master/Optimal%20Page%20Replacement%20Algorithm%20%28OPT%29.py?utm_source=chatgpt.com. (Accessed: 2025-5-19)

systems, O. (s.f.). *Operating systems.* https://www2.cs.uregina.ca/~hamilton/courses/330/notes/memory/page_replacement.html. (Accessed: 2025-5-19)