BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA

SIMULADOR INTERACTIVO PARA LA ENSEÑANZA-APRENDIZAJE DE ALGORITMOS DE REMPLAZO DE PÁGINAS EN LOS SISTEMAS OPERATIVOS

Presenta: Julián Adair Reyes Reyes

Tesis presentada como requisito para obtener el título de Licenciatura en Ingeniería en Ciencias de la Computación.

Facultad de Ciencias de la Computación

Asesores

Dra. Hilda Castillo Zacatelco

Dr. José Luis Carballido Carranza

Noviembre 2023

Resumen

En este trabajo se describe el desarrollo de un simulador para el estudio de sistemas operativos y la gestión de memoria a través de la técnica de paginación y el algoritmo de envejecimiento para el reemplazo de páginas. El sistema cuenta con distintas ventanas para inspeccionar en todo momento el estado de los componentes del sistema como el CPU, la tabla de páginas, la tabla de procesos, las instrucciones a ejecutar, entre otros elementos involucrados en el funcionamiento de un sistema operativo. El objetivo del simulador es fungir como una herramienta de estudio avanzada para explorar conceptos que de otra manera resultan complicados de analizar y visualizar. Este sistema se presenta como una aplicación web con una ejecución por pasos en donde el usuario debe confirmar de forma explícita la ejecución de cada paso para permitirle analizar el estado del sistema y avanzar a su propio ritmo.

Introducción

El estudio de los sistemas operativos y en específico, los mecanismos involucrados en la gestión de memoria son temas de suma importancia para los alumnos de ciencias de la computación. Estos temas, a la par de trascendentes, son bastante complejos y pueden representar un reto para el alumno, pues se requiere que el estudiante esté familiarizado, en mayor o menor medida, con temas de programación de bajo nivel, diseño digital y arquitectura de computadoras. Haciendo a un lado la complejidad, resulta fundamental que los alumnos comprendan a profundidad estos conceptos, pues los conceptos involucrados pueden ser aplicados en muchas otras áreas de la computación.

La paginación es una técnica de gestión de memoria virtual y se considera un tema fundamental para los cursos de sistemas operativos por su alto impacto y relevancia que ha tenido hasta hoy. Para entender la técnica de paginación, el alumno debe tener una importante noción de gran parte de los mecanismos que cooperan en un sistema operativo, como lo son los procesos, el funcionamiento del CPU y su comunicación con el resto del sistema, la memoria física, memoria secundaria, el planificador de procesos, entre otros elementos.

Lograr entender estos conceptos requiere un gran compromiso de parte del estudiante, pero también es muy importante emplear las herramientas de estudio adecuadas. Numerosos libros, artículos y videos se han dedicado al análisis y estudio de la paginación, no obstante, el principal problema con estos recursos es que no ofrecen al alumno la habilidad de interaccionar con los mecanismos que se están estudiando.

Utilizar herramientas que le permitan al alumno visualizar el funcionamiento y ejecución de algoritmos, ha demostrado jugar un papel importante en el entendimiento del tema a profundidad [1]. Este fue el punto de partida para el trabajo actual, en el cual se documenta el desarrollo de un simulador del mecanismo de paginación para mejorar la experiencia de estudio y la comprensión del estudiante.

En el área de sistemas operativos, se han desarrollado múltiples herramientas para la visualización de algoritmos de remplazo de página, por ejemplo, PAGE [2], MOSS [3] y Alg_OS (v2) [4]. El problema de estas soluciones es que varias de ellas fueron creadas con herramientas hoy en día obsoletas. PAGE fue escrita con Java applets y es imposible ejecutarse en un navegador web,

MOSS fue escrita utilizando características de Java 1.0 y Alg_OS (v2) se encuentra actualmente inaccesible. Para solventar estas carencias, se optó por desarrollar un sistema de simulación de algoritmos de reemplazo de páginas que puede ser ejecutado en cualquier navegador web sin importar el sistema operativo del usuario. Además, el simulador está disponible en español, cuenta con una interfaz de usuario amigable y es accesible para el público en general.

Antes de iniciar con el proceso de desarrollo, primero se atravesó una etapa de investigación para conocer a profundidad el tema. Como resultado de esta investigación, nos dimos cuenta de que había bastantes elementos involucrados en el mecanismo de paginación, y que no era posible omitir muchos detalles sin hacer de lado una experiencia de aprendizaje completa. Por esto se decidió modelar gran parte de los elementos de un sistema computacional involucrados en la paginación, desde el hardware hasta el software, desde la MMU *(Memory Management Unit)* que intercepta el bus de datos que comunica un CPU virtual con la memoria principal, hasta un despachador de procesos que decide cuál es el siguiente proceso a ejecutar y por cuánto tiempo. Los componentes del sistema simulado son una versión bastante simplificada de los de un sistema computacional real, no obstante, son los suficientemente completos para desempeñar su papel relacionado con la paginación.

En un punto del desarrollo, se contaba con un sistema de componentes que se comunicaban y cooperaban entre sí para simular un conjunto de procesos compitiendo por tiempo de ejecución y espacio en la memoria principal. Siendo el despachador de procesos y el algoritmo de reemplazo de páginas los encargados de regir el comportamiento del sistema. El resultado era bastante bueno, pero rápidamente surgió un problema: Como un simulador del sistema, la aplicación era bastante buena, pero como una herramienta de estudio interactiva no era más útil que observar a una computadora funcionando e imaginarnos lo que estaba ocurriendo. En este punto del desarrollo se decidió cambiar la dirección del proyecto para tomar un enfoque más educacional e interactivo. De esta decisión nació un sistema de simulación basado en un flujo de pasos cuyo funcionamiento está a merced del usuario, es decir, no se realiza ninguna acción hasta que el usuario lo indica y el sistema debe reportar cada uno de los efectos y resultados que se obtienen tras ejecutar cada paso.

Como resultado, obtenemos un simulador bastante completo que abarca los suficientes conceptos para considerarse material de estudio de un curso completo de sistemas operativos, haciendo un

especial énfasis en la paginación como mecanismo de gestión de memoria y los algoritmos de reemplazo de página.

Reconocimientos

Quiero agradecer a todas las personas e instituciones involucradas en el desarrollo de este proyecto.

En primer lugar, a la Vicerrectoría de Investigación y Estudios de Posgrado (VIEP), por su apoyo que hizo posible la elaboración de esta tesis.

A la Dra. Hilda Castillo Zacatelco, asesora de tesis, por su apoyo en la elección del tema, su paciencia y dedicación al responder mis dudas, su valiosa retroalimentación sobre la usabilidad del simulador, por contagiarme su entusiasmo por el estudio de los sistemas operativos y en general, por todo el apoyo que me brindó.

Por último, quiero agradecer a mi madre, por su apoyo incondicional, por todo su esfuerzo y sacrificios durante todos mis estudios. A mis familiares, cuyo apoyo fue de suma importancia para llegar hasta esta última etapa. Agradezco a mis amigos por permanecer a mi lado y a todas aquellas personas que hicieron más agradable mi estancia en la universidad y que formaron parte de mi vida en esta etapa tan importante.

Contenido

Marco Teórico 11
1.1 Definición de un sistema computacional 11
1.2 El proceso como unidad de trabajo 12
1.3 Gestión de memoria 13
1.4 Paginación 14
1.5 El algoritmo de envejecimiento para reemplazo de páginas 14
Desarrollo del simulador 16
2.1 El hardware del sistema 16
2.2 Administración de procesos y sistema de archivos 19
2.4 Paginación dentro del simulador 22
1.5 Ejecución por pasos 24
Resultados 27
Conclusiones y trabajo a futuro 29
Bibliografía 30

Capítulo 1

Marco Teórico

En este capítulo se exploran los conceptos necesarios para comprender el funcionamiento de un sistema computacional moderno. Se parte con la explicación de un sistema sencillo capaz de ejecutar instrucciones directamente de memoria principal y poco a poco se añaden componentes y conceptos hasta incorporar la paginación como técnica de gestión de memoria virtual.

1.1 Definición de un sistema computacional

Una computadora digital es una máquina que puede realizar tareas para sus usuarios al ejecutar las instrucciones proporcionadas. Un sistema de este tipo está compuesto por un sistema interconectado de procesadores, memoria y dispositivos de entrada y salida. Una secuencia de instrucciones que describen cómo realizar una cierta tarea, se conoce como un programa. El CPU (Unidad de procesamiento central) es el "cerebro" de una computadora. Su función consiste en ejecutar tareas almacenadas en memoria principal, obteniendo instrucciones de memoria, analizando y ejecutando una instrucción tras otra [5].

El CPU contiene una pequeña memoria de alta velocidad que se usa para almacenar valores temporales y de control. Esta memoria está conformada por un conjunto de registros donde cada uno tiene un determinado tamaño y función. El registro más importante es el **contador de programa** (PC en inglés), que almacena la dirección de la siguiente instrucción a ser recuperada para su ejecución. Otro registro importante es el **registro de instrucción**, el cual aloja la instrucción que está siendo actualmente ejecutada.

A grandes rasgos, el CPU sigue los siguientes pasos para ejecutar cada instrucción [5]:

1. Obtener la siguiente instrucción a ejecutar y almacenarla en el registro de instrucción (IR)
2. Actualizar el contador de programa para que apunte a la siguiente instrucción a ejecutar
3. Determinar el tipo de instrucción a ejecutar

4. Obtener los operandos de la instrucción ya sea de memoria o registros, en caso de ser necesario
5. Ejecutar la instrucción
6. Regresar al paso uno para ejecutar la siguiente instrucción

1.2 El proceso como unidad de trabajo

En un principio, un sistema computacional solo tenía la capacidad de ejecutar un único programa en un momento dado. Este programa al ser ejecutado tenía control total de los recursos del sistema, como el procesador, la memoria y los dispositivos de entrada y salida. Actualmente, los sistemas computacionales contemporáneos son capaces de ejecutar múltiples aplicaciones al mismo tiempo, por ejemplo, es común navegar por internet mientras se reproduce música y se trabaja con una hoja de cálculo. Para lograr este modelo de ejecución, se emplea el concepto de **proceso.** Un proceso es un programa en ejecución y se considera la unidad de trabajo en un sistema computacional moderno. De esta forma, podemos concluir que los sistemas computacionales contemporáneos son capaces de ejecutar varios procesos de forma concurrente, donde cada proceso compite por la obtención de los recursos del sistema para completar tareas en favor del usuario. Un proceso puede estar dividido en varias secciones (segmentos) donde cada una cumple una función específica. Para lo que nos compete en este trabajo, consideraremos el segmento de datos y el segmento de texto. El segmento de texto contiene las instrucciones que deben ser ejecutadas por el CPU, mientras que el segmento de datos contiene la información necesaria para ejecutar las instrucciones, por ejemplo, contiene variables, operandos y direcciones de memoria cruciales para la ejecución del programa [7].

Un proceso puede encontrarse en distintos estados: en ejecución, espera, terminado y listo. Un proceso se encuentra en ejecución cuando tiene el control del CPU y los recursos del sistema. Durante la ejecución del proceso, es probable que se ejecute una instrucción que requiera de la intervención del kernel, como solicitar datos de un dispositivo de E/S. Mientras la solicitud se lleva a cabo, el proceso queda en modo espera hasta que se completa la tarea. Una vez la tarea se completó, el proceso pasa al estado “listo” que lo vuelve un candidato para ser el próximo proceso

a ejecutar. Cuando un proceso ha finalizado todas sus instrucciones, este pasa finalmente al estado de terminado [6].

El despachador de procesos es el componente que se encarga de determinar cuál es el siguiente proceso a ejecutar y por cuánto tiempo. Esta decisión se toma en base al algoritmo de despacho implementado. Uno de los algoritmos más sencillos y con un buen resultado es el de *Round Robin*, el cual elige un proceso de entre una cola de procesos listos y además asigna un tiempo máximo de ejecución permitido. Si el proceso excede este límite de tiempo, se activa un mecanismo del despachador que obliga al sistema a hacer un cambio de proceso [7].

1.3 Gestión de memoria

Inicialmente, las computadoras no ofrecían una abstracción para acceder a la memoria principal, ocasionando que los procesos interaccionaran directamente con la memoria física disponible. En este modelo, los programas hacen referencia directamente a direcciones físicas absolutas. Si bien esta aproximación sigue teniendo utilidad en sistemas embebidos donde se conoce con anticipación el proceso a ejecutar (p. Ej. un horno de microondas sencillo), no satisface las necesidades en equipos de cómputo de uso general, donde se asume la capacidad de ejecutar varias aplicaciones simultáneamente como un navegador web, un procesador de texto, un lector de correo electrónico, etc. Este modelo permite alojar únicamente un programa en memoria en cualquier momento determinado [7].

Una de las soluciones de gestión de memoria más empleadas en los sistemas actuales es utilizar memoria virtual. La memoria virtual nace para solventar el hecho de que la memoria es capaz de albergar cada proceso por separado en su totalidad, pero la cantidad de memoria requerida para almacenar a todos los procesos juntos excede a la memoria disponible. La idea detrás de la memoria virtual consiste en dotar a cada proceso con su espacio de memoria, el cual está particionado en bloques llamados páginas. Cada página es un rango consecutivo de direcciones y es mapeada a la memoria física. Cuando un programa hace referencia a su espacio de direcciones, el hardware en automático traduce las direcciones locales a direcciones físicas. En caso de que la dirección referenciada no tenga asociada una dirección física, es decir, dicha página no está mapeada, el sistema lanza una alerta (fallo de página) y procede a traer la información faltante y re-ejecutar la instrucción. Uno de los aspectos más importantes de la memoria virtual, es la

posibilidad de un proceso para ejecutarse sin la necesidad de que todas sus páginas se encuentren en memoria física [7].

1.4 Paginación

La mayoría de los sistemas de memoria virtual utilizan una técnica llamada paginación, en la cual cada referencia a memoria pasa primero por la MMU. Este dispositivo se encarga de traducir el espacio de direcciones virtuales al espacio de direcciones físicas. Cada espacio de direcciones virtuales consiste de un conjunto de bloques de tamaño fijo llamado páginas. Por cada página existe un bloque en la memoria física llamado marco de página (*page frame*, en inglés). Al momento de que se hace referencia a una página que no tiene correspondencia con un marco de página, se ejecuta una TRAP especial llamada **fallo de página**. Ante este evento, el sistema operativo debe elegir una página para escribir su contenido de vuelta a disco y traer la nueva página al marco de página recién desocupado. El sistema operativo utiliza una estructura de datos llamada tabla de páginas para llevar un control de las páginas virtuales, incluyendo a aquellas que tienen asociado un marco de página. Por cada entrada en esta tabla se almacena la siguiente información: un bit que indica si dicha página tiene un marco asociado y en dado caso su número, un campo de protección, un bit para indicar si la página ha sido recientemente modificada (W bit) o referenciada (R bit), entre otros campos que pueden depender de la máquina y/o sistema [7].

Al momento que ocurre un fallo de página, si bien se podría seleccionar una página al azar para ser remplazada, se obtiene mejor rendimiento al seleccionar una página no utilizada frecuentemente. Para determinar la página que es la mejor candidata a ser remplazada, se han desarrollado múltiples algoritmos de remplazo de página [7].

1.5 El algoritmo de envejecimiento para reemplazo de páginas

Uno de los algoritmos mejor aceptados por su eficiencia y practicidad es el de envejecimiento (*aging*, en inglés) [7]. El algoritmo de envejecimiento se basa en la idea de dotar a cada página con un contador de *k* bits. A cada interrupción de reloj, el contador de cada página es desplazado un bit a la derecha y a su bit más significativo (el de más a la izquierda) se le suma el valor del bit R (Que puede ser 0 o 1 según si ha sido recientemente accedida o no). Cuando ocurre un fallo de página, este algoritmo elige a la página cuyo valor de contador es el más pequeño para ser reemplazada. Si la página a reemplazar fue recientemente modificada, esta debe ser escrita

nuevamente en disco para preservar su estado. Este algoritmo nos proporciona un mecanismo que es fácil de implementar y relativamente económico (computacionalmente) para llevar un registro de la frecuencia de uso de cada página.

Capítulo 2

Desarrollo del simulador

2.1 El hardware del sistema

La técnica de paginación como estrategia de gestión de memoria, involucra múltiples componentes del sistema comunicándose y cooperando entre sí. Antes de pensar en los componentes de alto nivel involucrados en la paginación como lo son la tabla de páginas y el algoritmo de reemplazo de páginas, necesitamos tener una base sólida que de soporte al comportamiento básico de un sistema computacional. Partiendo de esta idea, se decidió considerar los componentes básicos de una computadora digital para formar parte del simulador: El CPU, la memoria principal y un bus de datos para la comunicación.

El diseño del CPU (así como del resto de componentes) es bastante sencillo y simplificado. Se decidió que sería un procesador de 8 bits, con 16 de registros de 16 bits cada uno, donde los registros del 0 al 7 son de propósito general, los registros 8 y 9 almacenan el contador de programa y registro de instrucción, respectivamente, y el resto de registros quedan disponibles para futuras características. El tamaño de los registros se eligió con el tamaño de una instrucción en mente. Una instrucción estaría compuesta de 16 bits, donde los primeros 4 bits representan el código de operación, los siguientes 4 bits son para especificar un número de registro como operando y los últimos 8 registros son para especificar un segundo operando que puede ser tanto una dirección de memoria o un valor de 8 bits. No obstante, más adelante se explica por qué se eligió no almacenar las instrucciones en memoria principal y solo almacenar datos.

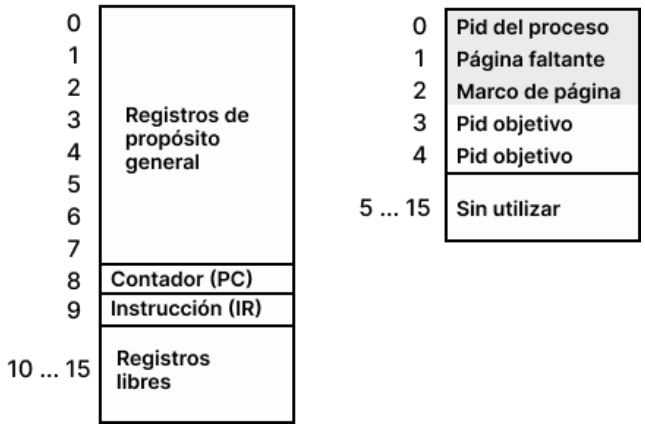

Figura 1. El primer diagrama muestra la configuración base de los registros del CPU del simulador. El segundo diagrama representa la aplicación de los registros cuando se ejecuta la rutina de fallo de página.

El conjunto de instrucciones está compuesto por tan solo 3 instrucciones: LOADV, LOADA y STORE. Las instrucciones LOADV y LOADA sirven para cargan un valor específico en un registro del CPU (*Load value*) y para cargar el valor almacenado en una dirección de memoria en un registro del CPU (*Load register*), respectivamente. Estas instrucciones son suficientes para representar la gestión de memoria con paginación ya que nos ofrecen acceso a la memoria para lectura y escritura. Incorporar instrucciones adicionales al simulador es una tarea bastante sencilla, pues solo hace falta proporcionar la implementación de una función que recibe tres parámetros: la instancia del CPU, y los dos operandos de la instrucción que está siendo definida.

```
export const instructionSet = new InstructionSet({
  LOADV: {
  usage: "LOADV REG VALUE",
  validation: /LOADV \d+ \d+/,
  description: "Carga el valor VALUE en el registro REG",
  execute(cpu, register, value) {
    // solo permitir escribir los registros del 0 - 7
    cpu.registers.write(value, Math.min(register, 7));
  },
  // El resto de instrucciones...
})
```

Figura 2. Fragmento de código donde se pueden agregar instrucciones adicionales al conjunto de instrucciones del CPU.

El CPU implementa métodos que le permiten ejecutar instrucciones como lo haría un CPU real. Por ejemplo, el método *fetch* de la clase *CPU* se encarga de usar la dirección almacenada en el registro de contador de programa para obtener la siguiente instrucción a ejecutar y almacenarla en el registro de instrucción. Por otro lado, el método *decode* implementa la lógica para descifrar la instrucción, es decir, conocer su nombre, sus operadores y la tarea a realizar. Por último, el método *execute* ejecuta la instrucción y accede a memoria o registros según sea necesario.

Para incorporar la dimensión temporal en el simulador, se implementó un reloj parecido al que encontramos en un sistema computacional real. Desde una vista de alto nivel, el reloj emite pulsos cada cierto tiempo y al CPU le toma exactamente un ciclo de reloj (pulso) ejecutar una instrucción cualquiera. Esta noción del tiempo resulta importante para describir los eventos del sistema y aplicar políticas relacionadas con la paginación. Por ejemplo, los ciclos de reloj representan la unidad de tiempo que será utilizada para determinar el tiempo que le toma al sistema recuperar una página desde el almacenamiento secundario y cargarla en memoria principal.

La implementación de la memoria principal y de los registros del CPU son especializaciones de la clase *ReadWriteStorage* (Almacenamiento con lectura y escritura). Esta clase base ofrece métodos para leer y escribir sobre una estructura de datos en forma de arreglo. Las clases que heredan de esta clase base deben proporcionar una instancia de la clase *TypedArray* que determina el tamaño del almacenamiento, así como el tamaño de cada uno de los elementos, esto nos permite establecer el tamaño de cada localidad de memoria en 8 bits mientras que el tamaño de cada registro se establece en 16 bits. Un objeto de tipo *T*ypedArray representa una vista de tipo arreglo de un buffer de datos binarios, lo que nos otorga una mayor eficiencia a la hora de trabajar directamente con bits [8].

Cuando se ejecuta una instrucción que involucra acceder a memoria, el CPU lo hace a través de un objeto de la clase *DataBus,* cuya función es concentrar en un solo lugar y bajo la misma interfaz el acceso a memoria, de esta forma, el MMU tiene la capacidad de interceptar la comunicación y realizar traducciones de direcciones virtuales a direcciones físicas.

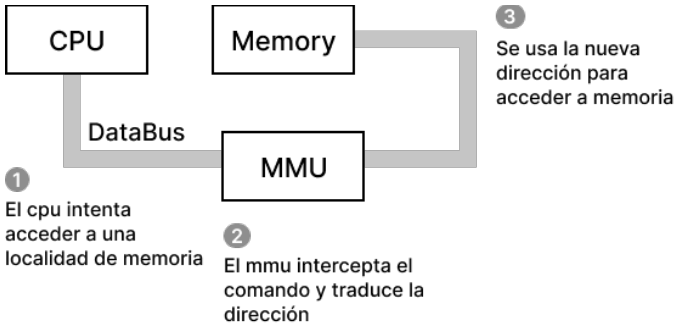

Figura 3. Diagrama que muestra el proceso que sigue el CPU para acceder a memoria

2.2 Administración de procesos y sistema de archivos

Con las características hasta ahora descritas, el sistema es capaz de ejecutar una serie de instrucciones almacenadas en la memoria física. El siguiente paso lógico fue implementar la abstracción de procesos para encapsular el trabajo y lograr un entorno de multiprogramación. La clase *Process,* almacena un conjunto de registros idénticos a los que encontramos en la clase CPU y contiene propiedades para guardar el identificador del proceso (pid), su estado de ejecución, el nombre del archivo ejecutable y el ciclo de reloj en el que inició su ejecución.

A la hora de lanzar la ejecución de un proceso, se debe cargar el proceso en memoria y replicar el contenido de sus registros internos en los registros del CPU. Para administrar los procesos existentes en el sistema, se implementó la clase *ProcessTable* (Tabla de procesos) que como su nombre indica, es una tabla que almacena referencias a los procesos actuales y ofrece mecanismos para acceder y manipular procesos a través de su pid. La clase *Scheduler* contiene una lista de procesos en espera y una cola de procesos listos para ejecución. Cuando es tiempo de seleccionar un nuevo proceso para ejecución, la clase Scheduler expone el método *pickProcessToRun* que desencola un proceso de la cola de procesos listos y lo establece como proceso en ejecución.

La conexión entre el “hardware” de este sistema y el mecanismo de gestión de procesos se da a través de la clase *Kernel.* Esta clase tiene acceso a todos los elementos del sistema hasta ahora descritos y es la clase encargada de orquestar las funciones del sistema y facilitar la comunicación entre componentes. La clase kernel cuenta con la propiedad *currentProcess* para llevar el control de cuál es el proceso actualmente en ejecución.

Para completar el factor de multiprogramación, se necesitaba un espacio de almacenamiento que sirviera como sistema de archivos para almacenar programas para su posterior ejecución. Este almacenamiento secundario también sería una pieza fundamental en el mecanismo de paginación, pues resulta indispensable el poder leer, escribir y transferir páginas desde disco a memoria principal.

Una de las “limitantes" en el desarrollo web del lado del cliente (código que se ejecuta en el navegador del usuario), es que no se tiene acceso a muchos de los recursos del sistema en busca de preservar un entorno seguro de ejecución sin comprometer la integridad del sistema ante aplicaciones maliciosas o que presentan errores, por este motivo, desde el navegador no se puede acceder directamente al sistema de archivos del sistema, lo cual hubiera simplificado en gran medida la tarea de presentar un sistema de archivos listo para el uso dentro del simulador. Dada la enorme complejidad que supondría implementar un sistema de archivos más o menos realista desde cero, se optó por buscar una opción ya existente que ofreciera una funcionalidad similar. Para nuestra suerte, existe un proyecto que se encarga precisamente de traer toda la funcionalidad que se espera de un sistema de archivos, pero dentro del navegador. Esta librería se llama BrowserFs[9] e implementa la *API* de *Node* para manipular e interactuar con el sistema de archivos.

En este punto del desarrollo, era posible llevar a cabo la ejecución de procesos sobre programas almacenados en disco. Para ejecutar un proceso se debía leer el código fuente del programa y cargarlo en su totalidad en memoria e inicializar los registros del CPU. Para esclarecer el mecanismo de multiprogramación se implementó la técnica de los registros base y límite para almacenar múltiples procesos en memoria y otorgarles un tiempo determinado de ejecución. Para llevar a cabo esta técnica, se ocuparon los registros 10 y 11 del CPU (y la imagen de cada proceso) para almacenar el valor de base y límite del proceso en ejecución. Al cargar un proceso en memoria, su valor base es igual a la dirección de memoria en la que el proceso inicia, mientras que el valor de límite corresponde al tamaño en bytes del proceso. De esta manera, para ejecutar alguno de los procesos en memoria bastaba con respaldar el valor de los registros del anterior proceso y cargar el valor de los registros del proceso entrante. Los procesos hacían referencia a direcciones de memoria locales, es decir, solo tomaban en cuenta su propio espacio de direcciones y el CPU se encargaba de sumar el valor del registro base a las direcciones y de verificar que estas

direcciones estuvieran acotadas por el rango comprendido por los registros base y límite. Llevar a cabo este experimento fue de gran ayuda para tomar las decisiones de diseño posteriores.

Con todos los elementos anteriores en su lugar, fue posible continuar con la siguiente fase: implementar el mecanismo de paginación en conjunción del algoritmo de reemplazo de página.

2.4 Paginación dentro del simulador

La paginación en el simulador empieza por la inclusión del MMU como componente del sistema. Se trata de una clase que ofrece principalmente un método para traducir entre direcciones virtuales y direcciones físicas. El MMU se sitúa entre el Bus de datos y la memoria principal interceptando los accesos a memoria y realizando la traducción de direcciones según sea necesario. Para realizar la traducción de direcciones, el MMU determina el número de página que corresponde a la dirección entrante y consulta la tabla de páginas para determinar si dicha página está asignada a un marco de página en memoria principal. Las entradas de esta tabla contienen información que indica si la página se encuentra en memoria principal, en qué marco de página y si ha sido recientemente referenciada o modificada. Cada entrada almacena datos requeridos para la operación del algoritmo de reemplazo de página utilizado, en este caso, el algoritmo de envejecimiento (*aging*).

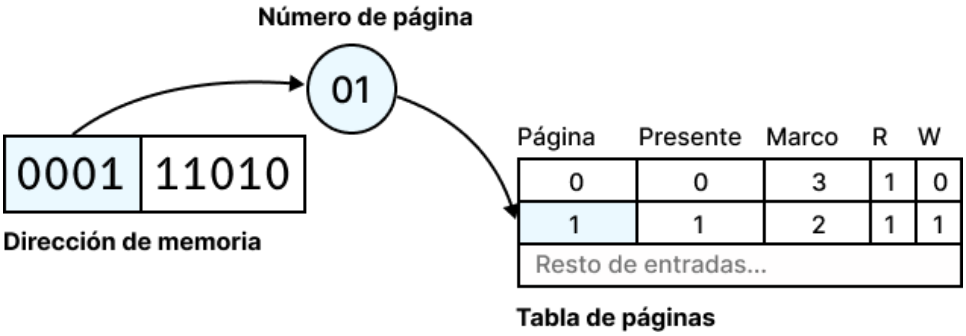

Figura 4. Visualización del proceso llevado a cabo por el MMU para determinar si la página que contiene la dirección referenciada se encuentra en memoria principal.

Todos los procesos cuentan con una versión de su propia tabla de páginas con el estado de su última ejecución. Cuando un proceso entra en ejecución, su tabla de páginas se copia a la tabla de páginas del MMU y cuando éste sale, la tabla del MMU se respalda en la imagen del proceso. Lo anterior ocurre de forma similar a como los procesos almacenan el estado de los registros del CPU para posteriormente reanudar su ejecución.

Otra clase involucrada en la paginación es la de *PageFrameTable* que como su nombre indica, es una tabla cuyas entradas representan los marcos de página del sistema. Cada entrada de esta tabla relaciona un marco de página con información acerca de si está ocupado, por cuál proceso y cuál

de las páginas de dicho proceso. Cuando ocurre un fallo de página, se analiza la tabla de marcos de página para determinar si existe un marco disponible. En caso de que la tabla se encuentre llena, se invoca el algoritmo de reemplazo de página para seleccionar una página para ser reemplazada.

Para el algoritmo de envejecimiento se almacena un contador de ocho bits (edad) que será actualizado a cada ciclo de reloj según si la página ha sido referenciada recientemente (valor del bit R). Recordemos que el bit R de una página tiene un valor igual a 1 cuando la página ha sido referenciada, a cada ciclo de reloj se realiza un corrimiento de bits del contador, recorriendo todos los bits una posición a la derecha para finalmente sumar el valor del bit R al bit más significativo del contador. El bit R se establece en 0 cada cierto tiempo para que el algoritmo sea capaz de distinguir entre páginas que fueron recientemente referenciadas y páginas que fueron frecuentemente referenciadas en el pasado. A la hora de seleccionar una página para ser reemplazada se selecciona aquella con el contador de edad más bajo, pues representa la página que ha sido referenciada menos frecuentemente.

Tras ocurrir un fallo de página, sin importar si había inicialmente un marco disponible o este fue liberado mediante el algoritmo de reemplazo de página, esta página debe ser traída desde disco a memoria principal. La ventaja de usar paginación es que mientras una página es recuperada, se puede seleccionar otro proceso para ejecutarse mientras se completa la transferencia. Para simular este comportamiento, se optó por un sistema de ejecución pseudo asíncrono basado en ciclos de reloj. Es decir, la transferencia se realiza inmediatamente, pero esta tarea no se marca como completada hasta cierto número de ciclos de reloj. Para facilitar la observación del comportamiento del algoritmo de reemplazo, se decidió que la transferencia de una página se completa en dos ciclos de reloj. Para implementar esta funcionalidad, se creó la clase *InterruptController* que almacena una cola de tareas pendientes, donde de cada tarea se cuenta con una descripción de la tarea, el ciclo en el que inició, cuántos ciclos toma la tarea y, por último, la función a ejecutar una vez se completaron los ciclos pendientes. En esta función, se especifica que el proceso que ocasionó el fallo de página vuelve a estar en estado de “listo” y la página ahora se marca como presente en la memoria principal. Dentro del simulador, se muestra un componente de interfaz de usuario con un listado de tareas pendientes y cuántos ciclos de reloj hacen falta para que se complete la tarea. Eventualmente las tareas completadas son eliminadas de la cola para dar lugar a nuevas operaciones.

1.5 Ejecución por pasos

El objetivo de este simulador es servir como un recurso que mejore la experiencia de aprendizaje sobre paginación en los sistemas operativos. Se buscaba diseñar un mecanismo que permitiera a los usuarios saber qué es lo que estaba ocurriendo en cualquier momento y por qué ocurría. Se buscaba pues, un sistema operativo que se ejecutara paso por paso, explicándose a sí mismo para facilitar el aprendizaje de los conceptos involucrados en sus decisiones.

Los requisitos para orquestar la ejecución del simulador fueron las siguientes:

- El simulador no debe realizar ninguna tarea, salvo que el usuario lo indique
- El simulador debe indicar cuál es el siguiente paso a ejecutar
- Una vez ejecutada una acción o un paso, el simulador debe mostrar un reporte del resultado de ejecutar el último paso.
- Cualquier evento relevante ocurrido durante el último paso debe reportarse.

Al diseñar este sistema de ejecución por pasos, se observó que este debía girar en torno al reloj del sistema, ejecutando una serie de pasos a cada nuevo ciclo, no obstante, no podía ser una serie de pasos lineal, pues según el resultado de la ejecución de un paso, se necesitaría tomar una decisión u otra. Por ejemplo, tras ejecutar una instrucción, puede que ocurra o no ocurra un fallo de página. Incluso una vez que ocurrió un fallo de página, se presentan dos escenarios distintos: cuando hay un marco disponible o cuando se necesita la intervención del algoritmo de reemplazo de página. A partir de estos ejemplos, podemos observar que el modelo a seguir es similar al de un diagrama de flujo, en el que se según el resultado de cada paso, por medio de un conjunto de condicionales, se evalúa cuál es el siguiente paso a ejecutar.

Para implementar este modelo de ejecución, se deben definir los pasos a ejecutar y un mecanismo para almacenarlos y reproducirlos. Se decidió utilizar una tabla que relaciona el identificador único de cada paso con un objeto que lo caracteriza. Este objeto cuenta con dos propiedades principales: el título del paso y la función de implementación que se almacenan en las propiedades de *title* y *execute*, respectivamente. La función de implementación define la lógica del paso y debe retornar el identificador del siguiente paso a ejecutar, así como una cadena o un arreglo de cadenas que explican los resultados obtenidos tras ejecutar el paso. La función de implementación es llamada con el “sujeto de la simulación” como único argumento. Como su nombre indica, el sujeto es el

protagonista de la simulación y el objeto sobre el que se realizarán operaciones para observar su comportamiento. En este caso, el sujeto será una instancia de la clase *Kernel,* la cual expone todos los métodos necesarios para orquestar el funcionamiento del sistema operativo y el manejo de memoria. De esta forma, la lógica del sistema operativo permanece encapsulada y la clase del Simulador se limita a invocar métodos de la clase Kernel.

```
.addStep("write_dirty_page", {
	title: "Respaldar página modificada",
	execute(kernel, message) {
		// This pid and page correspond to the page to be replaced/evicted
		const [page, pid] = kernel.cpu.registers.read(
			PageFaultRoutine.Register.EvictPage.Pid,
			2
		);
		const pageData = kernel.memory.read(
			page * kernel.systemDescriptor.PAGE_SIZE,
			kernel.systemDescriptor.PAGE_SIZE
		);
		kernel.fileSystem.writePage(page, pid, pageData);
		message.set("La página fue respaldada en disco");
		return {
			goTo: "clean_removed_page_references",
			message,
		};
	},
})
```

Figura 5. Fragmento de código para definir un paso del simulador, su implementación, los valores que reporta y el siguiente paso a ejecutar.

Cuando se inicializa el simulador se proporcionan dos elementos importantes: el diagrama de pasos y el identificador del primer paso a ejecutar. La clase simulador expone el método *executeNextStep* para ejecutar el siguiente paso. Internamente, este método obtiene el paso que corresponde al siguiente paso a ejecutar, valida que existe, ejecuta su función de implementación y toma su valor de retorno para actualizar cuál es el siguiente paso a ejecutar.

El diagrama de pasos a ejecutar se diseñó para ser lo más atómico y explicativo posible, lo cual nos deja con un flujo que consiste en más de 20 pasos diferentes. Debido a que el simulador se pausa a la espera de que el usuario ejecute el siguiente paso, esto se traduce en que se necesitan ejecutar muchos pasos para poder observar el comportamiento de la paginación y su influencia en el resto del sistema. Para solventar este problema, se optó por una solución que consiste en definir un subconjunto de pasos que serán los pasos de interés. Cuando el simulador ejecuta un paso, determina si el siguiente paso a ejecutar es un paso de interés. En caso de que el paso sea de interés, se pausa la ejecución normalmente a la espera de la ejecución manual ocasionada por el usuario. En caso contrario, automáticamente se ejecuta el siguiente paso hasta que se encuentre un paso de interés. A través de esta sencilla regla, obtenemos un mecanismo para pausar la ejecución solo cuando sea relevante para el usuario, facilitando así la observación de fenómenos como la participación del algoritmo de reemplazo de páginas, pues este solo entra en juego después de que no haya marcos de página disponibles.

Capítulo 3

Resultados

Como resultado tenemos una aplicación que simula el comportamiento de un sistema operativo haciendo énfasis en la administración de memoria mediante la técnica de paginación. El flujo de ejecución es bastante detallado y se esfuerza en ofrecer un alto nivel de realismo sin llegar a ser excesivamente complejo. El comportamiento de los componentes involucrados en la paginación en un sistema operativo real es representado por una versión simplificada dentro del simulador.

El usuario final interacciona con una aplicación web compuesta por varios componentes donde cada componente se centra en representar el estado de un elemento o concepto de interés dentro del simulador.

El componente de "Proceso en ejecución" se encarga de mostrar la información del proceso actualmente en ejecución, donde podemos ver el pid del proceso, el nombre del archivo ejecutable, las tres instrucciones más recientes (resaltando cual es la siguiente instrucción a ejecutar) y la tabla de páginas del proceso actual (tabla que reside en el MMU). Al pasar el puntero por encima de una instrucción se presenta una ventana flotante que explica la sintaxis y el comportamiento de dicha instrucción. El componente de "Tareas en segundo plano" muestra una lista de tareas pendientes, así como una lista de las tareas completadas. De cada tarea se muestra una breve descripción y los ciclos de reloj que hacen falta para que se complete. Este componente es útil para conocer las transferencias de página que están ocurriendo desde disco a memoria principal. Por otra parte, tenemos el componente de "Marcos de página y Páginas presentes" que se divide en dos pestañas, respectivamente. La pestaña de marcos de página es útil para saber rápidamente cuáles son los marcos de página ocupados, así como saber el proceso y página (del proceso) que reside en cada marco. La pestaña de páginas presentes muestra únicamente las páginas actualmente en memoria y, además, permite visualizar la edad de cada página para apreciar los corrimientos de bits y el estado de los bits R y M de cada página. Otro componente es el de "Tabla de procesos" que permite visualizar los procesos del sistema junto a su estado, ya sea que estén en ejecución, en espera, listos o finalizados. Al hacer clic en un proceso podemos visualizar una copia de su tabla

de páginas y el segmento de texto del proceso. Por último, tenemos el componente de “Control de simulación” para avanzar sobre la ejecución del simulador y observar los resultados obtenidos. Se presenta un botón principal que al accionarse el siguiente paso es ejecutado. Adicionalmente, se presenta una lista con los eventos del sistema y el usuario puede seleccionar solo aquellos eventos que sean de su interés. En ese mismo componente se puede visualizar una lista de los últimos quince pasos ejecutados en conjunción con el resultado obtenido tras ejecutar cada uno de ellos. En cada uno de los componentes anteriormente descritos, se muestran iconos de ayuda que ofrecen pistas y explican cómo interpretar e interaccionar con la información que se muestra.

Capítulo 4

Conclusiones y trabajo a futuro

El simulador desarrollado representa una herramienta de estudio que puede facilitar la compresión de conceptos de sistemas operativos y la administración de memoria virtual. La paginación siendo un mecanismo complejo, es representado paso por paso, lo cual ofrece una experiencia de estudio más flexible que se adapta al ritmo de aprendizaje propio de cada alumno. En el componente de "Páginas presentes" podemos observar las páginas en memoria principal y cómo se actualizan sus edades en tiempo real. Esta visualización resulta de gran ayuda para comprender el funcionamiento del algoritmo, que de otra forma suele ser difícil de representar en un medio estático como un pizarrón o libreta. Gracias a la libertad de seleccionar los eventos de interés, el alumno se puede centrar en aquellos eventos que le interesen, para así observar más específicamente un fenómeno en particular o analizar el comportamiento general del sistema y cómo cada uno de sus componentes participan en la ejecución del sistema.

El simulador puede ser extendido en varios aspectos. Como parte de una futura versión se espera incluir un catálogo de algoritmos de reemplazo de página, donde cada uno cuente con su propia representación visual que facilite el análisis del algoritmo. Además, El planificador de procesos también se beneficiaría de contar con un catálogo de algoritmos de despacho. Para estos catálogos de algoritmos, el usuario podría seleccionar el algoritmo a utilizar al inicio de la simulación. Por otra parte, se planea incluir la funcionalidad para que el usuario cree nuevos procesos y además defina las instrucciones que formarán el segmento de texto de cada proceso, es decir, que el usuario pueda crear sus propios programas valiéndose del conjunto de instrucciones del simulador.

Bibliografía

[1] Lawrence, A. W., Badre, A. M., & Stasko, J. T. (1994, October). Empirically evaluating the use of animations to teach algorithms. In *Proceedings of 1994 IEEE Symposium on Visual Languages* (pp. 48-54). IEEE.

[2] Khuri, S., & Hsu, H. C. (1999, March). Visualizing the CPU scheduler and page replacement algorithms. In *The proceedings of the thirtieth SIGCSE technical symposium on Computer science education* (pp. 227-231).

[3] Ontko, R., & Reeder, A. (2008, November 23). *MOSS*. Modern operating systems simulators. Retrieved February 23, 2023, from http://www.ontko.com/moss/

[4] Garmpis, A. (2011). Design and development of a web-based interactive software tool for teaching operating systems. *Journal of Information Technology Education: Research*, *10*(1), 1-17.

[5] Tanenbaum, A. S. (2016). *Structured computer organization*. Pearson Education India.

[6] Peterson, J. L., & Silberschatz, A. (1985). *Operating system concepts*. Addison-Wesley Longman Publishing Co., Inc.

[7] Tanenbaum, A., & Bos, H. (2015). *Modern operating systems* (4th ed.). Pearson.

[8] Mozilla Developer Network. (s. f.). *TypedArray*. MDN. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/TypedArray

[9] BrowserFS. (s. f.). *BrowserFS API Documentation.* BrowserFS. https://jvilk.com/browserfs/1.4.1/index.html