Santiago Vela - 202026767

Verónica Escobar – 201922197

Caso 2: Memoria Virtual

Contenido

Introducción

El objetivo de este caso fue el de crear una simulación de como el sistema operativo administra la memoria RAM de un dispositivo por medio de la creación de la memoria virtual y otros elementos tales como el sistema de paginación. Es por esto que el caso se enfoca principalmente en el Funcionamiento del sistema de paginación y su Funcionamiento. Para simular esto se escribió un programa en Java que simula el sistema de paginación usando el algoritmo de envejecimiento. En esta simulación se observa el comportamiento del sistema de paginación mientras que un proceso corre, en este caso la suma de dos matrices. En el transcurso de este proceso el sistema de paginación debe tomar decisiones de remplazo de página basados en el algoritmo de envejecimiento.

Funcionamiento del Sistema (general)

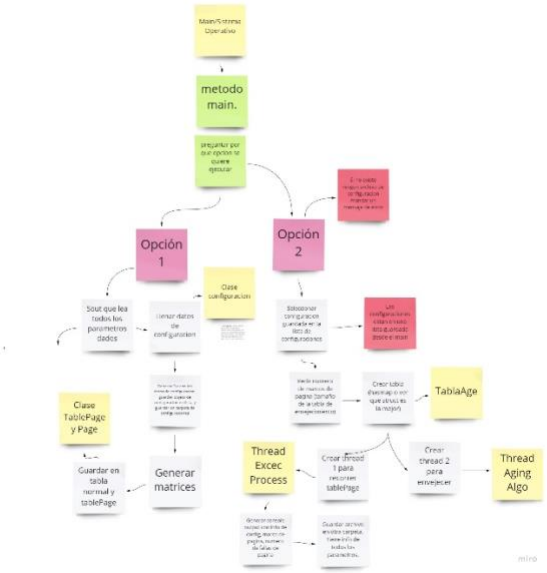


Ilustración 1. Diagrama de la solución

El diagrama anterior muestra la estructura general de la solución dada al caso y el flujo de decisiones para solucionar tanto la opción 1 como la opción 2. Aquí se ven presente algunas de las clases que fueron usadas en la implementación en java.

Insertar diagrama de clases cuando terminemos

En el diagrama se puede ver como Main es nuestra clase principal, sus responsabilidades constan de manejar todo lo que la interfaz de usuario, selección de opciones y la carga de archivos de configuración (que es donde se guardan los resultados de opción 1). Nuestra clase main se podrá ver como la implementación del sistema operativo.

La clase de Configuración es donde se encuentra la implementación de la opción 1, ya que es donde se genera la tabla de páginas y la tabla de referencias a dichas paginas dependiendo de los parámetros dados por el usuario (tamaño de página, tamaño del entero, tamaño de las matrices, recorrido). Una instancia de la clase configuración guarda el estado de la configuración dada por parte del usuario, generando las tablas y reportes necesarios para la opción 1. Adicionalmente, la clase Mein guarda un arreglo con las configuraciones preexistentes, para que sean de fácil acceso al ejecutar la opción 2. Clases como Pagina y ElementInfo son auxiliares a la clase de Configuración, y estas se usan para crear las diferentes tablas que produce esta clase. Internamente la clase Configuración crea varias tablas que facilitan la eventual creación de la tabla de páginas (que es la usada para crear los reportes) y para crear la tabla de referencias (usada en opción 2).

La clase de Opcion2 actúa como un administrador para todos los procesos que se tienen que llevar a cabo para similar el sistema de paginación en conjunto con el algoritmo de envejecimiento. Para

cumplir con los requisitos de la opción 2 se usan 3 clases auxiliares: Thread1, Thread2 y Buffer. Son estas clases auxiliares las que implementan el algoritmo de envejecimiento y el sistema de paginación a detalle. La clase de Thread1 tiene una referencia a la tabla de referencias, donde se almacenan las referencias (en orden) hechas a cada página por el recorrido que fue previamente seleccionado en opción 1. El trabajo de thread 1 es recorrer la tabla de referencias, después intentar acceder al buffer para modificar la table del marco de página, donde se guardan las páginas accedidas y su "edad" para ingresar una nueva página, actualizar su edad (reseteando a 0, al ser la referencia más reciente), o generar un fallo de página para después swapear la página más reciente por la página referenciada actualmente. Adicionalmente, después de hacer esta acción el thread 1 se va a dormir por dos milisegundos. La clase Thread2 es responsable de ejecutar el algoritmo de envejecimiento, lo cual incurre acceder al buffer a la table del marco de página para incrementar por 1 la "edad" de cada referencia de página dentro del marco. Después de ejecutar el envejecimiento este thread se va a dormir por 1 milisegundo. Ambos threads correrán hasta que el thread 1 termine de recorrer la tabla de referencias. La clase del buffer, como antes mencionado, contiene la table del marco de página, la cual tiene las páginas referenciadas y su edad (que simboliza que tan reciente fue la última referencia a ellas dentro de la table de referencias). En esencia la clase del buffer esta sincronizada en relación a la table de marco de página y solo permite que un thread actúe sobre ella al tiempo, por medio de un método sincronizado sobre el cual expondremos en otra sección.

Funcionamiento opción 1

Para poder explicar el algoritmo de generación de referencias de página, el cual está en su esencia en los métodos de createReferenceTable () y getIntegerAccessTrack() de la clase de Configuration, antes se tiene que explicar todo el proceso y estructuras de datos usadas para llegar a este. Ya que se necesitan varias capas de procesamiento para llegar a la tabla de referencias de página.

Algoritmo de generación de referencias de pagina y estructuras de datos usadas

Dentro del Proyecto, la opción 1 y todos sus requerimientos están implementados en la clase de Configuration.java. A una instancia de esta clase se le son pasados todos los parámetros de ejecución de la opción 1 (tamaño de página, tamaño de un entero, numero de filas, numero de columnas, tipo de recorrido).

Matriz 1

a	b	c
d	e	f
g	h	i

Matriz 2

j	k	l
m	n	o
p	q	r

Matriz 3

s	t	u
v	w	x
y	z	aa

Ilustración 2 Ejemplo matrices (3x3)

posicion	memoria virtual	
0	a	*Matriz 1*
1	b	
2	c	
3	d	
4	e	
5	f	
6	g	
7	h	
8	i	
9	j	*Matriz 2*
10	k	
11	l	
12	m	
13	n	
14	o	
15	p	
16	q	
17	r	
18	s	*Matriz 3*
19	t	
20	u	
21	v	
22	w	
23	x	
24	y	
25	z	
26	aa	

Ilustración 3 Ejemplo Tabla Memoria Virtual (3x3)

Al ser creada una instancia lo primero que se hace es la creación de las tres matrices: matrix1, matrix2 y matrix3. Matrix 3 es creada como la suma de las otras dos matrices, pero la forma que se realiza esta suma es definida por el tipo de recorrido seleccionado. Mientras que se realiza esta suma se van guardando las referencias a los enteros de las matrices en el arreglo de

virtualMemory, la cual representa la memoria virtual. En las ilustraciones anteriores se puede ver un ejemplo de cómo se ven las matrices y como se ve el resultado de las referencias hechas por la suma en la memoria virtual.

pagina	elementos
0	a
	b
	c
	d
1	e
	f
	g
	h
2	i
	j
	k
	l
3	m
	n
	o
	p
4	q
	r
	s
	t
5	u
	v
	w
	x
6	y
	z
	aa
	bb
7	cc
	dd
	ee
	ff
8	gg
	hh
	ii
	jj

	kk
	ll
	mm
9	nn
	oo
	pp
	qq
10	rr
	ss
	tt
	uu
11	vv

Ilustración 4 Tabla de paginas de una (4x4)

Después de haber creado las matrices y la table de la memoria virtual se continua a crear la table de páginas. La table de páginas es un arreglo que contiene objetos Pagina, los cuales tienen la información del número de página, la capacidad de esa página (calculada con el tamaño de un entero y el tamaño de una página), y un arreglo que contiene las referencias a las referencias a los elementos contenidos dentro de esa página. Para crear esta tabla se usa un algoritmo que recorre la table de la memoria virtual de tal manera que se crea una página por cada n elementos, ese n siendo calculado como n=tamaño de página / tamaño de entero. La ilustración anterior ayuda a visualizar como está compuesta la tabla de páginas.

Elemento	Pagina	Desplazamiento
a	0	0
b	0	2
c	0	4
d	0	6
e	1	0
f	1	2
g	1	4
h	1	6
i	2	0
j	2	2
k	2	4
l	2	6
m	3	0
n	3	2
o	3	4
p	3	6
q	4	0
r	4	2
s	4	4

t	4	6
u	5	0
v	5	2
w	5	4
x	5	6
y	6	0
z	6	2
aa	6	4
bb	6	6
cc	7	0
dd	7	2
ee	7	4
ff	7	6
gg	8	0
hh	8	2
ii	8	4
jj	8	6
kk	9	0
ll	9	2
mm	9	4
nn	9	6
oo	10	0
pp	10	2
qq	10	4
rr	10	6
ss	11	0
tt	11	2
uu	11	4
vv	11	6

Ilustración 5 Tabla de paginas especial (4x4)

Después de haber sido creada la table de páginas se crea un arreglo que se llama tabla de páginas especiales, y es el precursor a la tabla de referencia de páginas. Esta table contiene para cada elemento de la table de la memoria virtual una referencia al elemento, la página en la que está ubicado y el desplazamiento dentro de esa página. Esta table es creada al recorrer la tabla de páginas y guardar la información ya dicha de cada elemento en un nuevo arreglo llamado specialPageTable. Lo que se guarda dentro de specialPageTable son objetos de ElementInfo, los cuales resumen los datos de la referencia al elemento, el número de página al que pertenece y el desplazamiento del elemento dentro de la página.

pagina	elemento referenciado
0	a
2	e
4	i
0	b
2	f
4	j
1	c
3	g
5	k
1	d
3	h
5	l

Ilustración 6 Tabla de referencias para recorrido 1 (2x2)

Finalmente, para generar la table de referencias de página lo que se hace es que dependiendo del recorrido genera un arreglo llamado accessPositionTable (creado en el método de getIntegerAccessTrack ()), el cual guarda las los elementos de la matriz accedidos, en orden de la memoria virtual, que fueron accedidas por el recorrido al hacer la suma. Después, se itera sobre este arreglo, y con el uso de la table de páginas especial se obtiene a que página referencia cada uno de estos elementos. Así obteniendo la lista de referencias de página hechas por el recorrido seleccionado. La ilustración anterior visualiza como se ve la estructura de datos de la table de referencias.

Funcionamiento opción 2

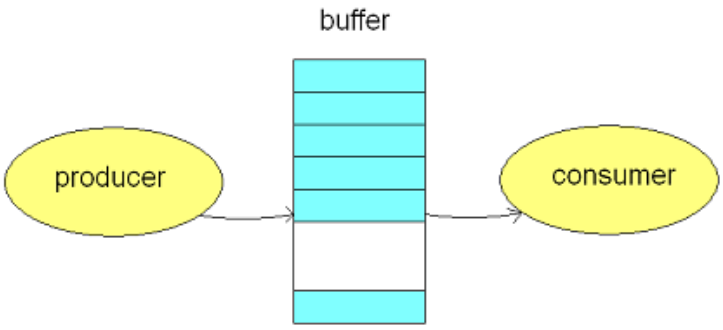


Ilustración 7 Problema de sincronizacion de producer consumer

La esencia del Funcionamiento de la opción 2 es muy similar a la del problema de sincronización de ProducerConsumer. En este caso la clase Buffer aloja a la table de marco de página, la cual

contiene la información de un numero de páginas (definido por el tamaño del marco de página dado por el usuario) y un número que representa que tan recientemente fue referenciada esa página en la tabla de referencias (entre menor el número menor tiempo desde la última referencia). El acceso a la esta sincronizado y vigilado por el monitor de la clase Buffer, solo permitiendo que un thread entre a esta a hacer una operación deseada en un momento dado. La clase de Thread1, la cual actúa como un Producer/Consumer (ambos son intercambiables en este caso), ya que el Thread 1 es responsable de ir recorriendo la table de páginas y actualizar en la table de marco de página cual fue la última página accedida. Ahora, en el caso que la table de marco de página se haya quedado sin espacio lo que sucede es que se produce un fallo de página, el cual implica que se tiene que hacer un swap entre la referencia a la página más Antigua y la pagina que se quiere ingresar. El thread 1 después de hacer una operación sobre la tabla de marco de página se va a dormir por 2 milisegundos antes de despertar de nuevo e intentar leer la siguiente página referenciada en la table de referencias. El thread 1 correrá hasta que haya leído todas las referencias en la table de referencias. Ahora, el Thread 2, un Producer/Consumer, es responsable de ejecutar el algoritmo de envejecimiento, el cual en esencia consiste en aumentar el tiempo de ultima referencia de cada página en la table de marco de página. Después de haber actualizado este valor se va a dormir por 1 milisegundo, antes de Volver a intentar ingresar a la tabla. Este thread correrá hasta que el thread 1 termine de recorrer toda la tabla de referencias.

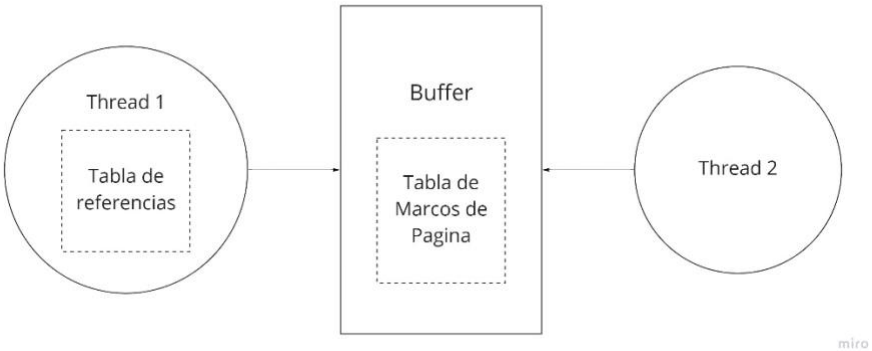


Ilustración 8 Implementacion Opcion 2

Es así con esta implementación que se puede similar el sistema de paginación y el uso del algoritmo de envejecimiento, al emplear la analogía del problema de Producer/Consumir.

Sistema de sincronización

La forma en la que desarrollamos el algoritmo de envejecimiento fue mediante listas de listas, para ser mas especifico usamos Arraylist, fue gracias a ellos que tanto el thread 1 como el thread 2 pudieron hacer el algoritmo, de modo que para poder leer una referencia de pagina se debían haber cumplido unos requerimientos posteriores como lo son: el que haya espacio en mis marcos de pagina y que al menos haya una pagina dentro de mis marcos de pagina, de igual manera es

evidente que para hacer esta sincronización se debió haber sido muy exactos ya que para ejecutar el algoritmo de manera correcta y según el enunciado, no se iba realizar por ciclos por reloj si no cada milisegundos, permitiendo de esta manera que cada dos milisegundos iba a entrar un referencia de pagina a mi memoria bien sea para hacer un fallo de pagina o envejecer de nuevo a la pagina referenciada.

Es muy importante decir que la sincronización no dependerá en este caso de los turnos de acceso si no que depende exclusivamente del programador y como este se va a ejecutar, por ejemplo en este caso no se va a meter paginas cada vez que se envejezca si no que, si va a meter paginas cada 1 o 2 milisegundos. Era muy importante saber que en este caso ambos threads iban a compartir una misma clase o elemento, en este caso el Buffer era el que nos ayudaba a darle acceso o ejecución a un thread, un thread no depende de nadie pero su ejecución y sus procesos dependerá de los accesos que le demos al mismo.

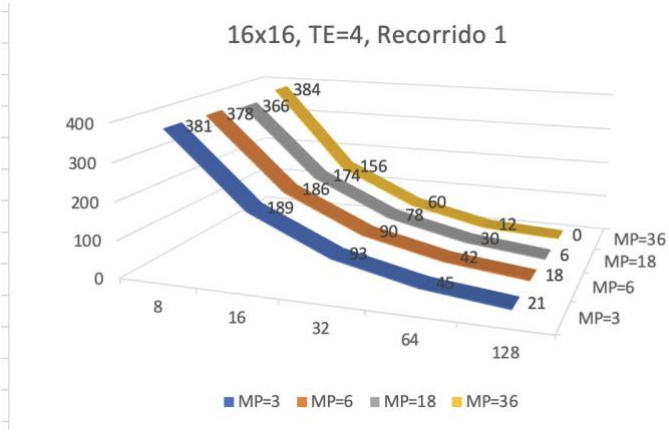
Datos recopilados

	MP=3	MP=6	MP=18	MP=36
8	756	378	750	732
16	765	762	750	732
32	765	762	750	732
64	765	762	750	732
128	381	378	366	0

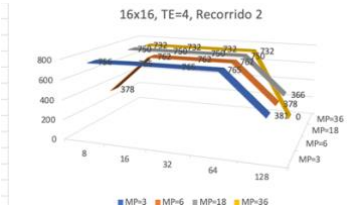
Recorrido 2

Graficas del comportamiento del sistema

Graficas



matrices de 16x16 enteros, tamaño de entero 4B. Recorrido tipo 1



matrices de 16x16 enteros, tamaño de entero 4B. Recorrido tipo 2

Interpretación de resultados

Nos podemos dar cuenta que los datos para el tipo de recorrido son iguales razón por la que el recorrido no es uniforme y el numero de paginas que se van a a listar no tendrán un orden uniforme como es el caso de tipo de recorrido 1 en el cual podemos ver que hay una tendencia y un orden de paginas según el tipo de marco y el tamaño de pagina

Conclusiones generales

Concluimos que este no era un caso complejo de entender, sin embargo pensamos que fue muy importante su implementación debido a que no fue simplemente unos recorridos con un nos números que los identificaban, si no que era entender un proceso y un funcionamientos que implementa cada computador. Pensamos que este caso y esta actividad contribuyo mucho a nuestro entendimiento formativo, debido que a interiorizamos conocimientos de niveles anteriores en temas que demandaban mayor pensamiento y mayor solución critica.

Pensamos que en el momento de las pruebas fue muy importante una buena implementación de fondo, ya que quisimos usar diferentes formas como HashMap, listas encadenadas, y objetos encadenados. Pero lo importante acá no fue como se hacían las cosas si no porque. Debido a que gracias a este porque, y gracias a ese entendimiento, la ejecución y la participación tanto de los Threads como de los Buffers fue mucho mas sencilla. Por otro lado Para nosotros el algoritmo de envejecimiento es un buen algoritmo debido a que siempre le da prioridad a los procesos que se están usando y aquellos que no tengan participación en el momento de solicitar recursos no se necesitaran. Pensamos que tanto por ciclos de reloj como por milisegundos , este algoritmo sirve mucho en la implementación hoy en día.