Daniel Osorio Cárdenas - 202022996
Juan Pablo Ramírez - 201921728
Mariana Zamora - 201923708

Informe Caso 2 – Memoria Vrtual

Estructura del proyecto

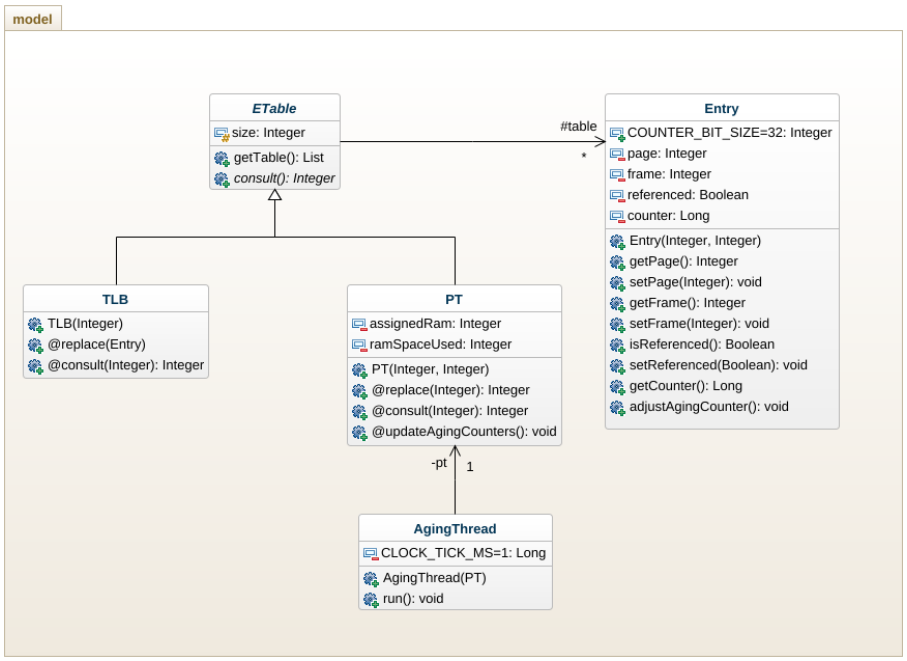


Figura 1. Diagrama de clases del paquete model

El proyecto se divide en dos paquetes: uno llamado *main* donde se encuentra la clase App que es el ejecutable y otro paquete llamado *model* donde se encuentran las clases que utilizamos para simular el funcionamiento de la memoria virtual. En el paquete *model* hay 5 clases: *Aging Thread, Entry, ETable, PT y TLB.* La clase fundamental es *Entry*, que representa una entrada de una tabla, ya sea la tabla de páginas o de la TLB. La clase *ETable* es una clase abstracta que representa una tabla de páginas de un tamaño fijo. La tabla se representa como una lista de entradas (de la clase *Entry)*. Esta clase también tiene dos métodos, uno que es el *getTable()* que retorna la tabla, y otro abstracto que es el *consult()*. Este último se implementa luego en las otras clases ya que se hace diferente dependiendo si la tabla es la Tabla de Páginas (clase *PT)* o si es el búfer de traducción anticipada (clase *TLB)*. Finalmente está la clase *Aging Thread* que se encarga de ejecutar el algoritmo de envejecimiento sobre las entradas de la tabla de páginas.

Teniendo en cuenta esto, la clase *Entry* tiene cinco atributos: *counter* que es de tipo long y que lleva la cuenta de las referencias hechas a esa entrada (de acuerdo al algoritmo de envejecimiento), *COUNTER_BIT_SIZE* que es int que lleva el tamaño fijo del número de bits que se van a tener en cuenta en el counter, *referenced* hace referencia al referenced bit que es un booleano inicializado en False y que se convierte en True cuando la entrada es consultada en la tabla de páginas, *page* que representa el número de página de la entrada y finalmente *frame* que representa la referencia a memoria RAM que indica en qué parte de la memoria física está guardada la traducción de la dirección virtual, o en su defecto toma el valor de -1 si la página no está cargada en la RAM. Esta clase tiene varios métodos, en su mayoría getters y setters para los atributos, pero el método principal que implementa es el *adjustAgingCounter*() que se encarga de actualizar el atributo counter dependiendo de si la página fue referenciada o no a partir del algoritmo de envejecimiento (shift a la derecha de los bits y si fue referenciada cambiar el bit más a la izquierda por 1).

La clase *PT* extiende de la clase *ETable* y representa la tabla de páginas. Esta clase tiene cuatro atributos: el tamaño (size) de la tabla de páginas y la tabla en sí (table), ambos heredados de la clase *ETable*, la RAM asignada al proceso (assignedRam) y el espacio de RAM utilizado (ramSpaceUsed). La RAM asignada al proceso es especificada por el usuario, mientras que el espacio usado de RAM se inicializa en cero y se va aumentando a medida que se va llenando la memoria. El tamaño de la tabla es fijo ya que el enunciado dice que se asignan 64 páginas al proceso. La tabla se inicializa como un ArrayList<Entry>, que se llena con 64 entradas cada una con el *frame* de referencia inicializado en -1, ya que al inicio la RAM está vacía y no se ha cargado ninguna referencia a la memoria física. Esta clase tiene tres métodos: *replace()* que encarga de actualizar las entradas de la tabla de páginas teniendo en cuenta el espacio de RAM usado y el algoritmo de envejecimiento, *updateAgingCounter()* que se encarga de actualizar los contadores de todas las entradas de la tabla, y el método *consult()* que busca una página específica en la tabla de páginas y retorna el *frame* donde está referenciada esa página en memoria física, o -1 si no está cargada en la RAM. Para la tabla de páginas se usa la estructura de datos de arreglo debido a que la operación más realizada es una búsqueda de una entrada, ya sea por consulta o reemplazo, y esta debe ser los más eficiente posible, lo cual se puede lograr en O(1) con un arreglo. Lo anterior se debe a que en la posición cero, está la entrada con página 0, en la posición 1, la entrada con página 1, así hasta *n* páginas del programa. De esta forma al consultar la posición *i*, solo se debe desplazar en el arreglo *dir. incial+i* y no tener que realizar una búsqueda total.

Por otro lado, está la clase *TLB* que solo tiene los dos atributos heredados de la clase *ETable*, que son el *size* de la TLB y la tabla que se inicializa como una LinkedList<Entry> para favorecer la complejidad del algoritmo FIFO que se utiliza para actualizar la tabla. Tiene dos métodos: *replace()* que se encarga de actualizar la TLB y *consult()* que busca la traducción de una dirección virtual en la TLB. Las entradas (referencias) que se cargan en la TLB son las mismas que las de la TP, es decir que cada elemento es un pointer a una entrada de la TP, con el fin de evitar inconsistencia entre ambas tablas de entradas. Para la TLB se usa la estructura de datos de lista encadenada porque la operación más realizada es el reemplazo, pues es más común en esta tabla que suceda un TLB miss y cambiar las entradas debe ser lo más eficiente posible, en este caso se logra en O(1). Lo anterior se debe a que al usar el algoritmo de reemplazo FIFO, al tener una nueva entrada en la TLB solo se cambia la referencia del primer elemento (frente de la cola) al segundo elemento, y al último elemento hacer que referencie a la nueva entrada (último en la cola). Sin embargo, para la búsqueda es necesario iterar sobre todas las entradas de la TLB O(n), pero como es una caché, no hay muchas entradas y el tiempo de respuesta es bajo.

Por último, está la clase *AgingThread* que extiende de la clase *Thread*. Tiene dos atributos, *CLOCK_TICK_MS* que es un long que representa el tiempo en milisegundos para simular cada cuanto se va a ejecutar el algoritmo, y el otro atributo es *PT* que lleva la referencia la tabla de páginas. Esta clase solo ejecuta el método *run()* siempre y cuando la aplicación no haya terminado de leer las referencias de la entrada, y que se encarga de actualizar los contadores de las entradas de la tabla de páginas de acuerdo al algoritmo de reemplazo por envejecimiento.

Funcionamiento y esquema de sincronización

El programa inicia en la clase *App*, en donde se tienen cinco atributos, dos de estos son constantes que son las PAGES_PER_PROCESS que son 64 páginas como lo muestra el enunciado, y el DIR_PROCESSES que es un string con el directorio donde se guardan los archivos que representan los procesos con referencias a memoria virtual, los otros tres atributos son el transTime, loadTime y el isFinished. El transTime lleva la cuenta del tiempo que se toma la memoria en traducir las direcciones de memoria virtual en direcciones de memoria física. El loadTime lleva la cuenta del tiempo que toma en cargar los datos en memoria principal y en la CPU. Finalmente, el isFinished es un booleano que indica si el programa ha terminado. En el *main()* primero se leen los inputs necesarios para el funcionamiento del programa: el número de entradas de la TLB, el número de marcos de página en RAM asignados al proceso

y el archivo que representa al proceso y del cual se leen las referencias a memoria virtual. Se inicializa la tabla de páginas (*PT)* y la *TLB*, se inicializa también el *AgingThread* para empezar a actualizar los counters de las entradas de la PT.

Después se llama al método *resolveAdresses()*, que se encarga de leer las referencias del archivo dado por parámetro y actualizar la *PT* y la *TLB* respectivamente a medida que se van leyendo las referencias de direcciones virtuales del archivo. Se va leyendo el archivo línea por línea, en donde cada línea (que representa la dirección en memoria virtual) se busca y se van sumando los tiempos de traducción a memoria física y de cargar los datos en memoria. Primero, se busca la dirección en la TLB, si ésta se encuentra ahí se suman 2 ns al transTime por acertar la búsqueda en la TLB y se suman 30 ns al loadTime de cargar la página desde la memoria principal. Si la página no se encuentra en la TLB, pasa a buscarla en la PT, en donde la sola búsqueda, independientemente de si la encuentra o no, le suma 30 ns al transTime. Si la traducción indica que la dirección no está cargada en memoria RAM, toca reemplazar una entrada en la tabla de páginas para cargar la nueva referencia en RAM, y este proceso le suma 10 ms al loadTime de cargar la página desde el SWAP del disco duro en la RAM, y también le suma otros 30 ns al transTime de volver a buscar la página en la PT una vez está cargada la página. En el caso de que la página que se consulta en la PT efectivamente está cargada en RAM, solo se le suma 30 ns al loadTime por cargar la página desde la memoria física. Este proceso se repite con todas las referencias del proceso para finalmente calcular el tiempo total de la traducción de las direcciones virtuales y de cargar los datos en RAM y en la CPU.

Los métodos de *replace()* y *consult()*, presentes en las clases *TLB* y *PT*, y el método *updateAgingCounters*() presente en la *PT*, son synchronized porque todos trabajan bajo la misma estructura de datos que es la tabla de páginas. En este caso, la tabla de páginas (*PT)* y la *TLB* actúan como un monitor para preservar la integridad y consistencia de las entradas de la tabla de páginas, además debido a que el *AgingThread* también accede a la tabla de páginas de forma concurrente para actualizar los contadores de cada entrada de la tabla. En concreto, si se están actualizando los contadores de envejecimiento de las entradas, todos deben ser actualizados en las mismas condiciones en un momento específico de tiempo, y no mientras se actualiza un contador, se estén cambiando el bit de referencia de las demás entradas (consulta y reemplazo), pues a la hora de reemplazar una entrada no se garantiza cuál es la página menos referenciada en un periodo de tiempo.

Resultados

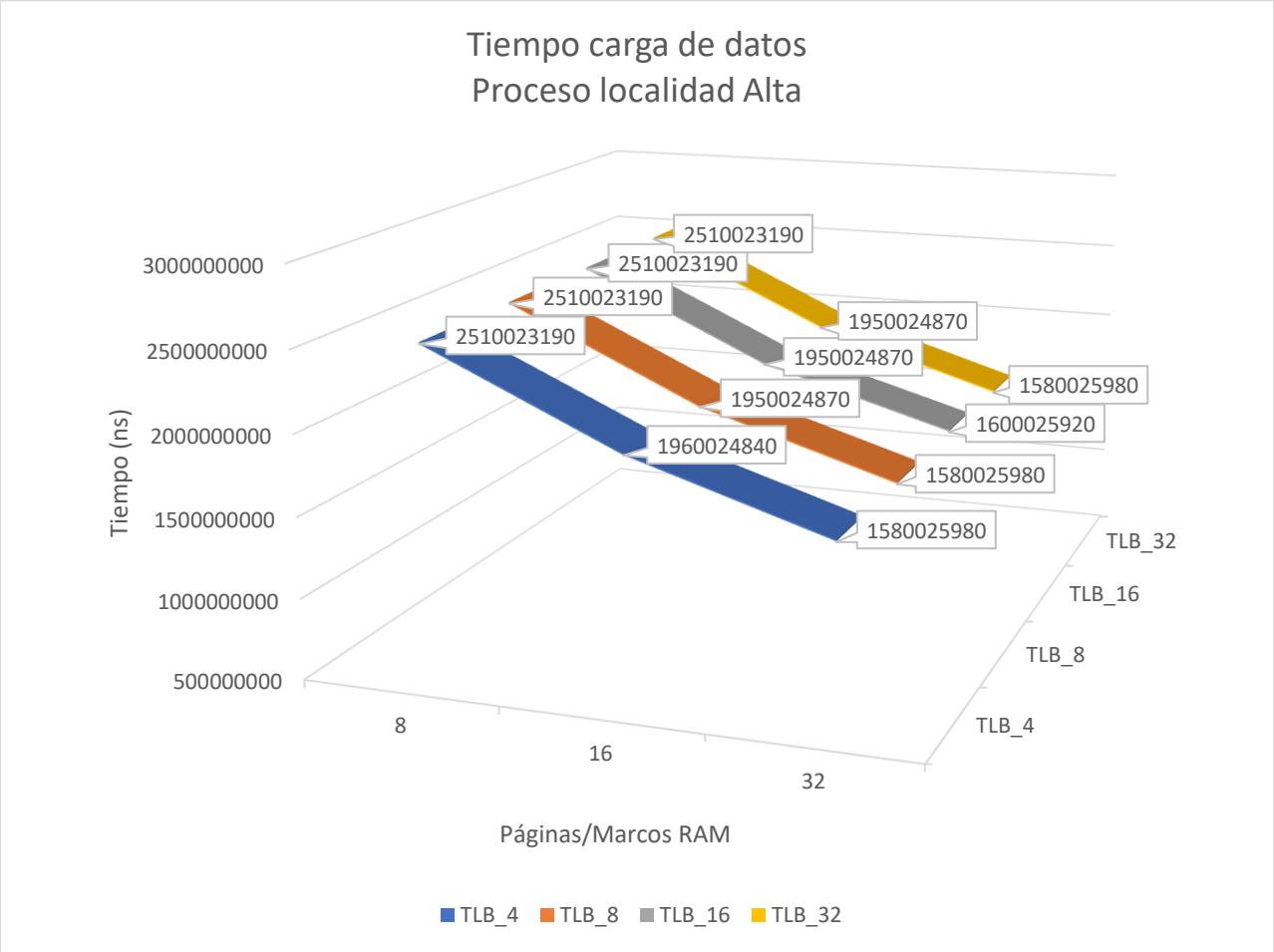
A continuación, se muestran las tablas y gráficas realizadas a partir de los casos de prueba. Para estas se hizo uso de dos archivos con distribuciones de referencias que corresponden a procesos de localidad alta y baja. Los procesos de localidad alta se refieren a aquellos procesos en los que las páginas referenciadas recientemente tienen una alta probabilidad de volver a ser referenciadas en un futuro próximo; mientras que los procesos con localidad baja son aquellos procesos en los que las mismas páginas no se referencian tan frecuentemente, y aquellas referenciadas recientemente tienen menos probabilidad de ser referenciadas nuevamente en un futuro próximo. Las pruebas realizadas se hicieron cambiando la cantidad de páginas/marcos en RAM y el tamaño de la TLB, los valores usados fueron: 8, 16 y 32; y 4, 8, 16 y 32 respectivamente. Se mostrarán primero las pruebas de tiempos de carga de datos y luego las de tiempos de traducción de direcciones virtuales.

Tiempo de carga de datos (ns) - Alta				
Páginas-Marcos en RAM / Tamaño de la TLB	TLB_4	TLB_8	TLB_16	TLB_32
8	2510023190	2510023190	2510023190	2510023190
16	1960024840	1950024870	1950024870	1950024870
32	1580025980	1580025980	1600025920	1580025980

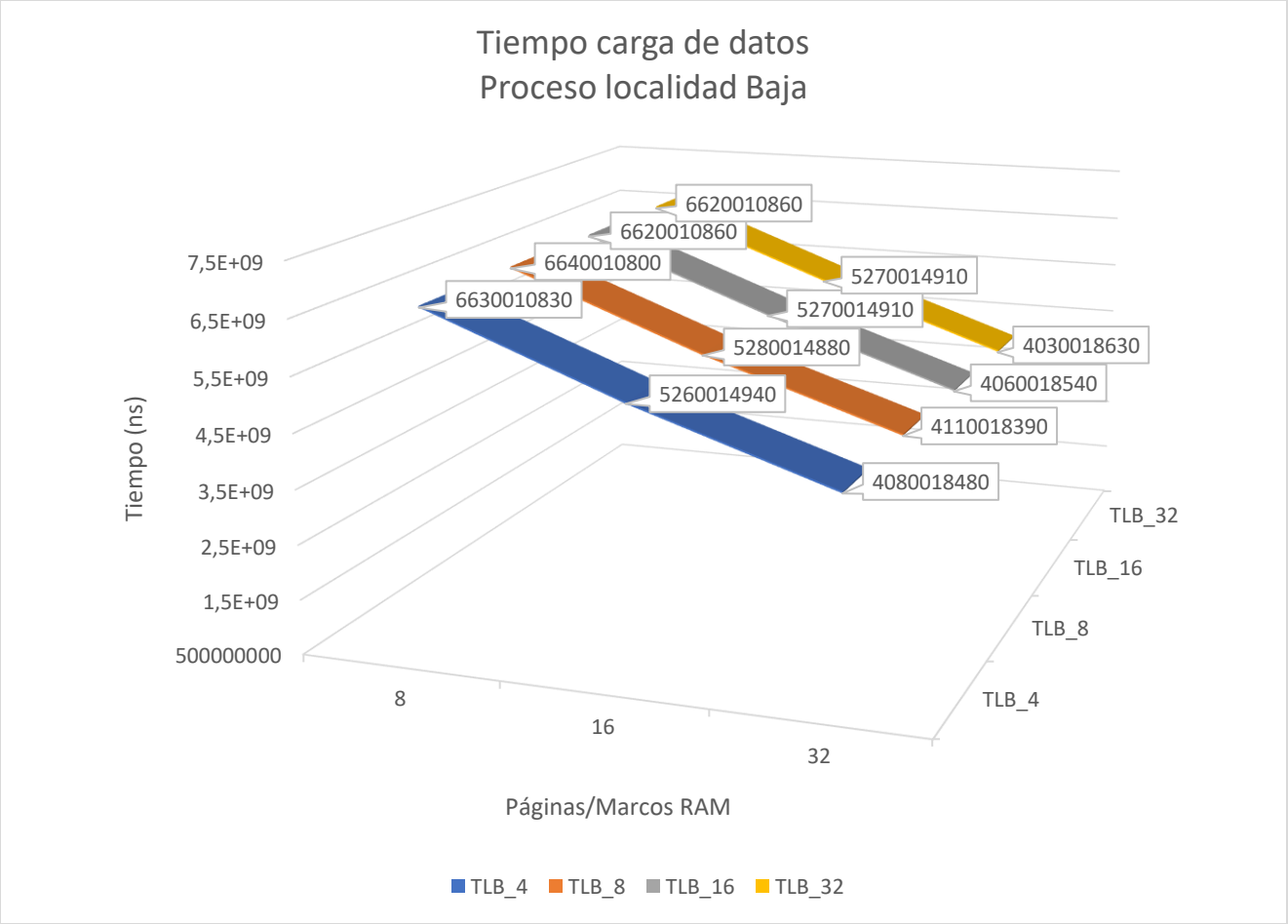
Tabla 1. Tiempo de carga de datos - localidad Alta

Tiempo de carga de datos (ns) - Baja				
Páginas-Marcos en RAM / Tamaño de la TLB	TLB_4	TLB_8	TLB_16	TLB_32
8	6630010830	6640010800	6620010860	6620010860
16	5260014940	5280014880	5270014910	5270014910
32	4080018480	4110018390	4060018540	4030018630

Tabla 2. Tiempo de carga de datos - localidad Baja



Gráfica 1. Tiempo de carga de datos - localidad Alta



Gráfica 2. Tiempo de carga de datos - localidad Baja

Como se puede observar en las gráficas, los tiempos de carga en los procesos con localidad baja son siempre más altos que los tiempos en los procesos con localidad alta. Esto se debe a que en los procesos con localidad baja las páginas son referenciadas menos frecuentemente lo que implica que es más probable que sean copiadas al área de swap tras un tiempo de no haber sido referenciadas. Esto afecta los tiempos de carga, pues al momento de volver a necesitar una de las páginas es más probable que esta se encuentre en el área de swap, la cual implica un tiempo de carga mucho más alto. Mientras que, en los procesos con localidad alta, al referenciarse las páginas con mayor frecuencia es más probable que estas sigan cargadas en memoria física, por lo que se generarán menos fallos de página llevando así a tener un menor tiempo de carga.

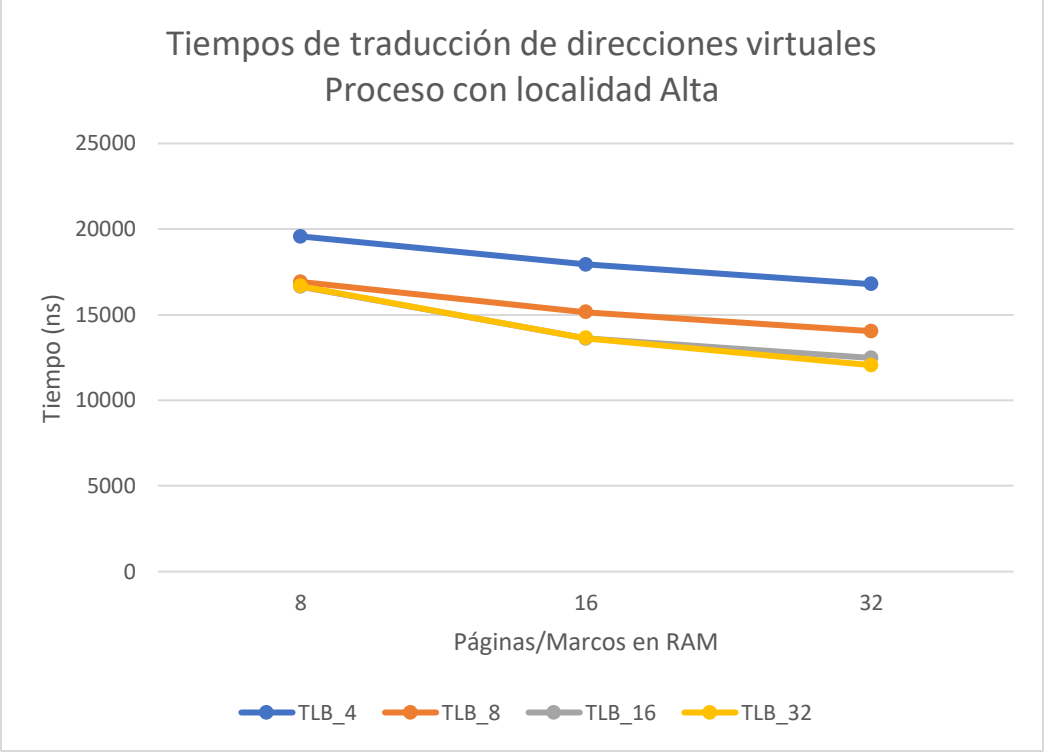
Se puede observar también que en ambos casos el tamaño de la TLB no es lo que afecta los tiempos de carga, sino la cantidad de páginas/marcos en la RAM asignados al proceso. Esto se debe a que la TLB sirve para almacenar las entradas que han sido recientemente referenciadas y tiene la misma información que la TP (es un subconjunto), por esto, el tiempo de carga de las páginas es siempre el mismo independiente si se busca en la TLB o TP, pues se deben cargar desde memoria física, o desde el área de swap de ser el caso. El número de páginas/marcos de la memoria RAM sí influye, pues tener un número más alto implica que se puedan guardar más páginas en memoria primaria disminuyendo la cantidad de fallos de página, pues al haber más espacio se vuelve menos necesario utilizar el área de swap, así más páginas son cargadas desde memoria física que tiene un tiempo de carga significativamente menor al tiempo de carga del área de swap.

Tiempos de traducción direcciones virtuales (ns)				
Páginas-Marcos en RAM / Tamaño de la TLB	TLB_4	TLB_8	TLB_16	TLB_32
8	19574	16914	16634	16662
16	17924	15150	13610	13638
32	16784	14040	12476	12052

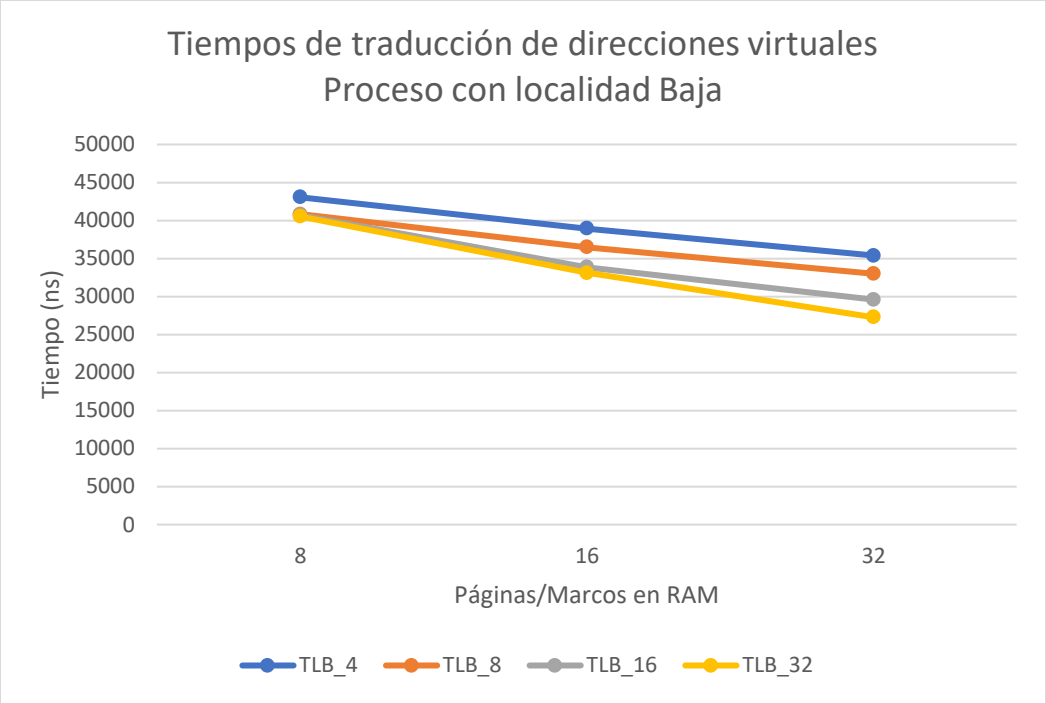
Tabla 3. Tiempo de traducción direcciones virtuales - localidad Alta

Tiempos de traducción direcciones virtuales (ns)				
Páginas-Marcos en RAM / Tamaño de la TLB	TLB_4	TLB_8	TLB_16	TLB_32
8	43050	40840	40695	40528
16	38940	36478	33874	33118
32	35400	32998	29600	27270

Tabla 4. Tiempo de traducción direcciones virtuales - localidad Baja



Gráfica 3. Tiempo de traducción de direcciones virtuales – localidad Alta



Gráfica 4. Tiempo de traducción de direcciones virtuales – localidad Baja

Como se puede observar en las gráficas y en las tablas, el tiempo de traducción de las direcciones virtuales es nuevamente mayor en todos los casos en los procesos de localidad baja. Esto se debe a que, como en estos procesos las páginas son referenciadas con menos frecuencia, no solo es más probable que se encuentren en el área de swap, sino que también es menos probable que se encuentren en la TLB, lo que hace que el tiempo de traducción sea más alto, pues se debe consultar la TP más veces. En los procesos de localidad alta es más probable que las páginas referenciadas estén ya en la TLB, por lo que el tiempo disminuye significativamente.

Se puede ver, además, que igual que en el caso del tiempo de carga, el número de páginas/marcos influye, esto se debe a que al haber más páginas o marcos en la memoria RAM habrá menos fallos de página, ocasionando que el tiempo de traducción disminuya, pues no es necesario hacer tantos reemplazos del área swap a la RAM.

También se puede observar que en este caso el tamaño de la TLB sí afecta el tiempo de traducción. Esto se debe a que esta vez se toma en cuenta el tiempo de consulta a la TLB y a la TP, siendo el tiempo de consulta a la TLB menor. Cuando el tamaño de la TLB aumenta se pueden guardar más referencias de páginas en esta, lo que implica que es más probable que una página se encuentre ya en la TLB, disminuyendo así el tiempo de consulta; pues cuando una página se halla en la TLB ya no es necesario consultar la TP.

Se puede notar también que en los casos en que la RAM tiene un tamaño de 8 marcos de página, y la TLB un tamaño mayor a 4, no se afectan los tiempos de traducción. Esto se debe a que, al haber menos marcos se producen más fallos de página, por lo que, el tamaño de la TLB no es tan significativo si la memoria RAM no tiene suficiente espacio para almacenar las páginas necesarias.

En conclusión, los resultados obtenidos tienen sentido tanto para los tiempos de carga como para los tiempos de traducción. Pues, en el caso de los tiempos de carga se puede notar claramente como es el tamaño asignado de la RAM y la localidad de los procesos lo que afecta el tiempo, pues estos son dos factores que influyen directamente en la cantidad de fallos de página que pueden llegar a ocurrir en este

caso de administración de la memoria virtual, que utiliza el algoritmo de envejecimiento para mandar las páginas menos recientemente referenciadas al área de swap. En el caso del tiempo de traducción de direcciones virtuales, los resultados demostraron nuevamente la importancia del tamaño de la RAM en relación con los fallos de página que pueden ocurrir. Además, fueron consistentes con el hecho de que el tamaño de la TLB afecta el tiempo de traducción pues se pueden almacenar más referencias de página y bajar el tiempo de consulta. Por último, cabe resaltar la relación lineal que muestran los datos donde a mayor cantidad de páginas asignadas en RAM, menor tiempo de traducción y carga de datos. Así mismo, a mayor cantidad de entradas en la TLB, menor tiempo de traducción, pero sin poder determinar bajo qué función ocurre esta relación.