Tutorial del programa MNEME y ejemplos de uso

Curso: *2006/2007*

Asignatura: *Arquitectura e Ingeniería de Computadores*

Autores:

Ana Davinia Guerra Amez (alu2822)

Pablo José Hernández López (alu2829)

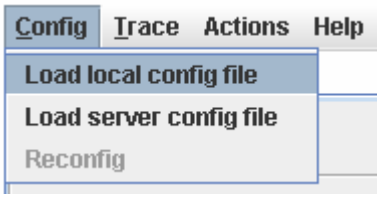
Rubén Cristo Gutiérrez Iglesias (alu2823)

INDICE

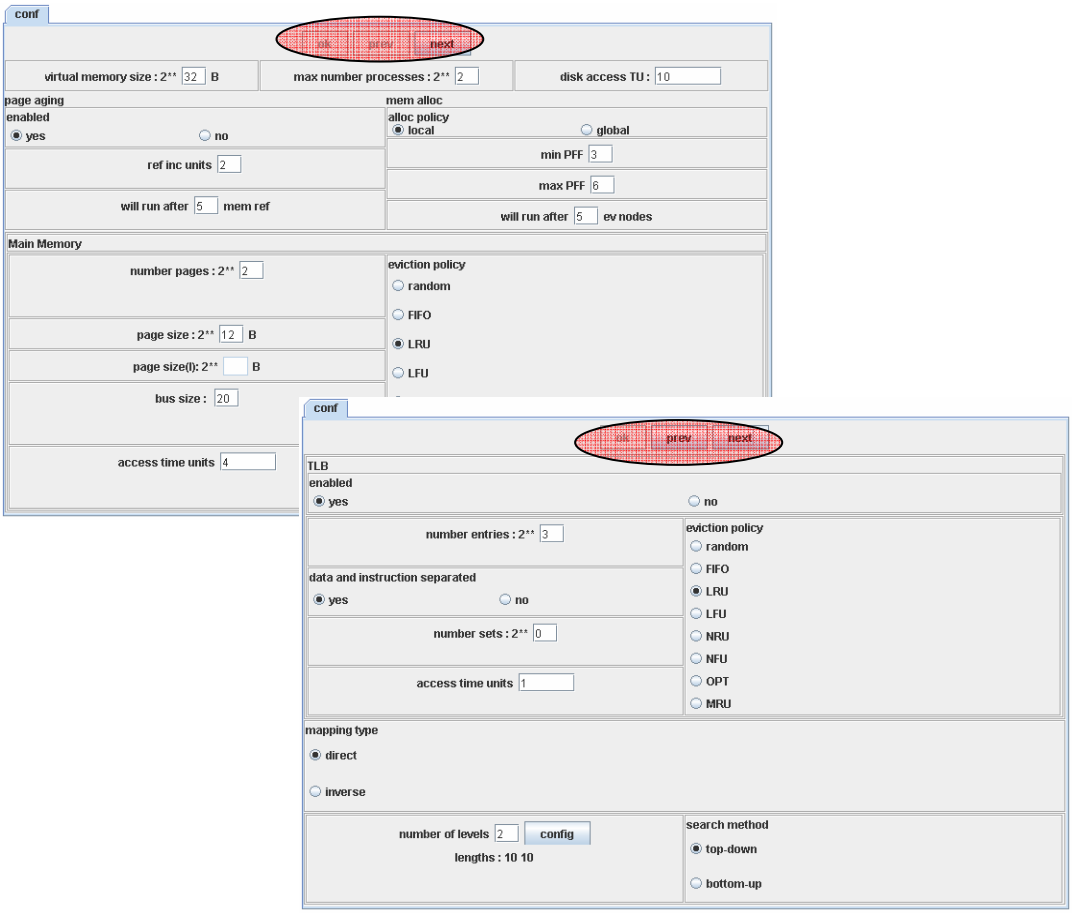
1. PARA COMENZAR A TRABAJAR.

Para poder comenzar a utilizar las tareas de simulación de Mneme, es necesario comenzar con realización de algunos sencillos pasos iniciales que situarán la ejecución en el momento exacto:

1. Realizar la configuración. Para ello bastará con cargar un fichero de configuración desde el menú de configuración *Config*. Dichos ficheros se encuentran alojados en la carpeta *conf* y tienen la extensión *.xml*.

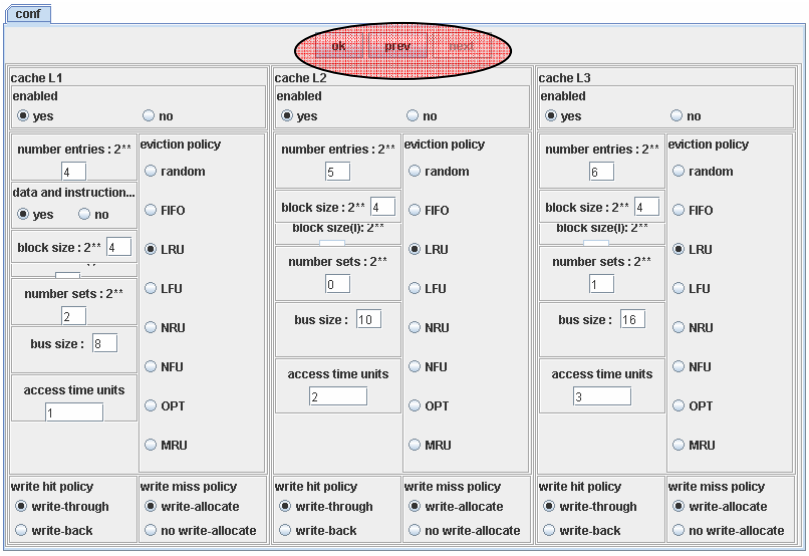


2. Pase a través de las distintas ventanas de configuración que se muestran:
   - Para validar cada una de las ventanas de configuración locales se deben presionar los botones *next* o *prev*.



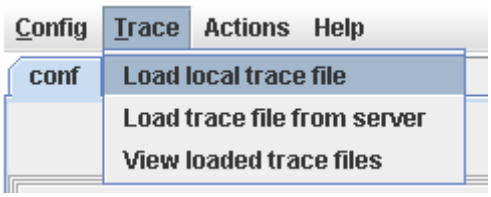
- Para la validación de la configuración global, que depende de cada uno de los campos de las diferentes ventanas de configuración, pulsar el botón *OK*.

El botón de *OK* sólo estará disponible una vez que se halla pasado por todas las ventanas de configuración, de modo que todas las validaciones locales se hayan realizado.



3. Presione la opción *Start* del menú *Actions*.

4. Cargue el fichero de traza desde el menú *Trace*. Estos ficheros se encuentran alojados en la carpeta *trace* y se identifican con la extensión *.trd*.



2. CONFIGURACIÓN.

2.1. Ficheros

2.1.1 Ficheros de configuración

Fichero XML con la configuración como elemento raíz:

ELEMENTOS
config *attr:* ---
virtualAddressNBits *attr:* --- *descr:* el número de páginas virtuales para cada proceso (por defecto = 32)
numberProccessesNBits *attr:* --- *descr:* la potencia de 2 del número máximo de procesos (por defecto = 0)
diskATU *attr:* --- *descr:* unidades de tiempo de acceso a disco (por defecto = 1)
pageTable *attr:* direct *descr:* si direct = true → mapeado directo (offsetLengths y searchMethod) si direct = false → mapeado inverso (hashAnchorSizeNBits)
offssetLengths *attr:* --- *cont:* length..length
length

attr: ---

descr: el tamaño en bits del offset de cada nivel, se toman en orden de aparición

searchMethod

attr: ---

descr: asigna el método de búsqueda como topDown o bottomUp

hashAnchorSizeNBits

attr: ---

descr: potencia de 2 del tamaño del hash en caso de que el mapeado sea inverso

pageAgingConfig

attr: ---

pageAgingIncrease

attr: ---

descr: unidades de incremento de edad para una entrada cuando ésta es referenciada

memRefToBeRun

attr: ---

descr: el número de referencias a memoria antes de que el thread relacionado con la edad de páginas se ejecute (por defecto = 1 = deshabilitado)

memAllocConfig

attr: ---

minPFF

attr: ---

descr: PFF mínimo

maxPFF

attr: ---

descr: PFF máximo

evNodesToRun

attr: ---

descr: número de páginas reemplazadas desde memoria principal antes de que se lance el thread (por defecto = 1 = deshabilitado = política de alojamiento local)

tlbConfig / mainMemoryConfig / cacheConfig

attr: ---

descr: la configuración de la caché se repetirá para cada una de las cachés de las que se quiera disponer

numberEntriesNBits

attr: ---

descr: potencia de 2 del número de entradas en caché

blockSizeNBits

attr: ---

descr: potencia de 2 del tamaño de bloque de la caché

evictionPolicy

attr: ---

descr: define la política de reemplazo, puede tomar los siguientes valores: random, fifo, lru, lfu, nru, nfu.

busSize

attr: ---

descr: potencia de 2 del número de entradas en caché

accessTimeUnits

attr: ---

descr: unidad de tiempo de acceso (por defecto = 1)

numberSetsNBits

attr: ---

descr: potencia de 2 del número de conjuntos en caché; esta valor numérico se relaciona con los valores: directa, completamente asociativa o asociativa por conjuntos.

dataInstrSeparated

attr: ---

descr: si es true → los datos y las instrucciones estás separadas

si es false → se crea una única tabla para datos e instrucciones

hitWritePolicy

attr: ---

descr: la política de escritura en este caso puede tomar valor writeThrough o writeBack (por defecto es writeBack)

missWritePolicy

attr: ---

descr: la política de escritura en este caso puede tomar valor writeAllocate o noWriteAllocate (por defecto noWriteAllocate)

Todos los marcos que no aparezcan como campos de entrada en los paneles de configuración son ignorados; de este modo el busSize es ignorado por la TLB, el numberSetsNBits ignorado por la memoria principal, el dataInstrSeparated ignorado por las cachés C2 y C3…

Todos aquellos valores que no se hayan definido en el fichero de configuración tomarán los valores por defecto definidos.

2.1.2 Ficheros de traza

Ficheros de texto con extensión *.trd* que contienen, en cada línea, la siguiente información:

virtual address	instruction type	time

- **Virtual Address:** expresado en formato hexadecimal. Esta dirección será truncada para adaptarla al número de bits definidos durante la configuración de la dirección virtual (tal y como se verá en futuros apartados).
- **Instruction Type:** representa el tipo de instrucción y puede contener uno de los siguientes valores: MEMREAD, MEMWRITE y FETCH.
- **Time:** la versión actual del simulador no hace uso de este valor.

2.2 Validación de los datos de configuración

Existen dos tipos de validaciones: la validación que se realiza en las páginas de configuración (cuando se pulsa *next* o *prev*) sobre los campos que sólo aparecen en la página actual y la validación final (que se realiza al pulsa *ok*) sobre los campos que no están en la misma página de configuración.

Las validaciones realizadas son:

- para la configuración de la memoria *(primer panel de configuración)*
  - el número máximo de procesos que pueden ser alojados debe estar comprendido entre 1 y 2^8
  - el número total de páginas virtuales debe ser mayor o igual al número de páginas reales (el número de entradas en memoria principal)
  - el tamaño de las direcciones virtuales deben ser menores o iguales a 32
- para la tabla de página *(segundo panel de configuración)*
  - si el mapeado es directo
    - el número de niveles debe estar comprendido entre 1 y el número de páginas virtuales por cada proceso.
    - la suma de los tamaños de offset debe ser igual al número de páginas virtuales por cada proceso.
  - si el mapeado es inverso
    - el tamaño del hash debe ser mayor o igual al número de entradas en la memoria principal y menor o igual que el número total de páginas virtuales.
- para las cachés, incluyendo la memoria principal *(tercer panel de configuración)*
  - si la caché está habilitada el número de entradas debe estar comprendido entre 2 y 2^{32}

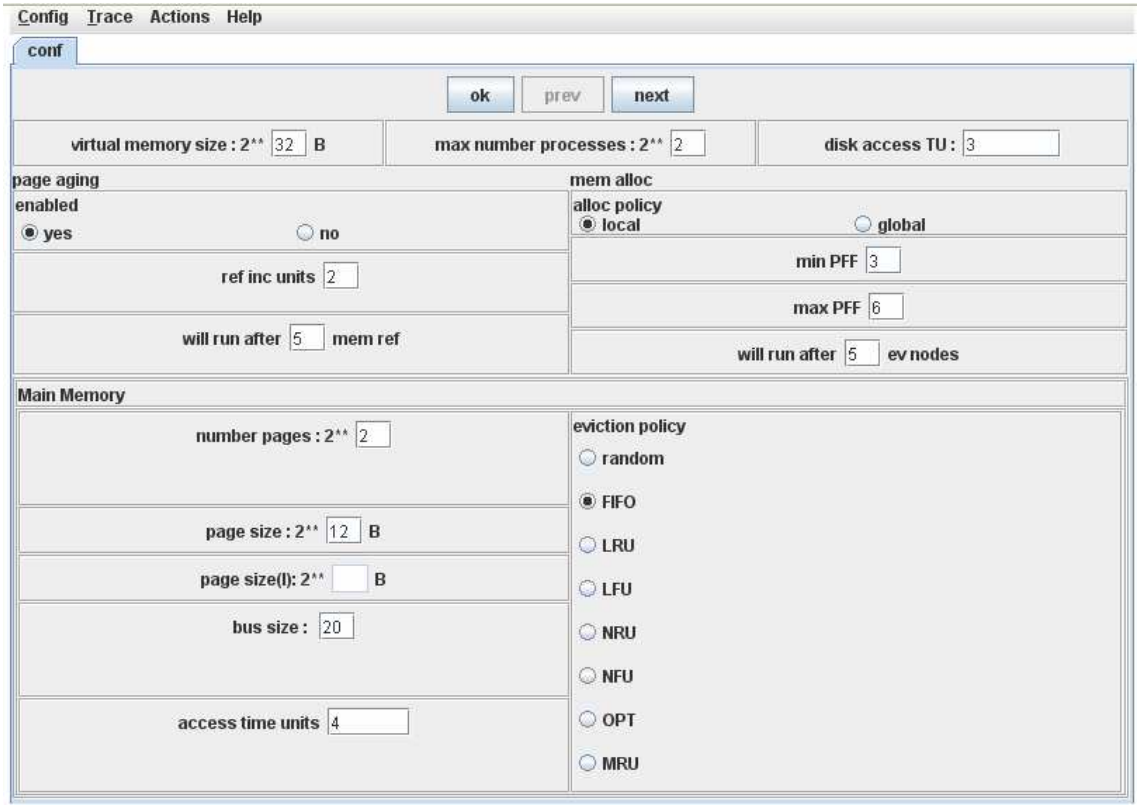
- el tamaño del bloque debe estar comprendido entre 2 y 2^{20}
- el número de conjuntos debe estar comprendido entre 1 y el número de entradas
- el tamaño del bus debe estar comprendido entre 8 y 128
- las unidades de tiempo de acceso a caché deben estar comprendidas entre 0 y 50000
- el tamaño de bloque de las cachés debe ser menor o igual al tamaño de página de la memoria principal

(en este caso se asume que las cachés tienen el mismo tamaño de bloque)

2.3. Pantallas de configuración en MNEME

Aquí se muestran cada una de las pantallas de configuración. Cada campo posee una referencia al apartado donde se explica con algo más de detalle:

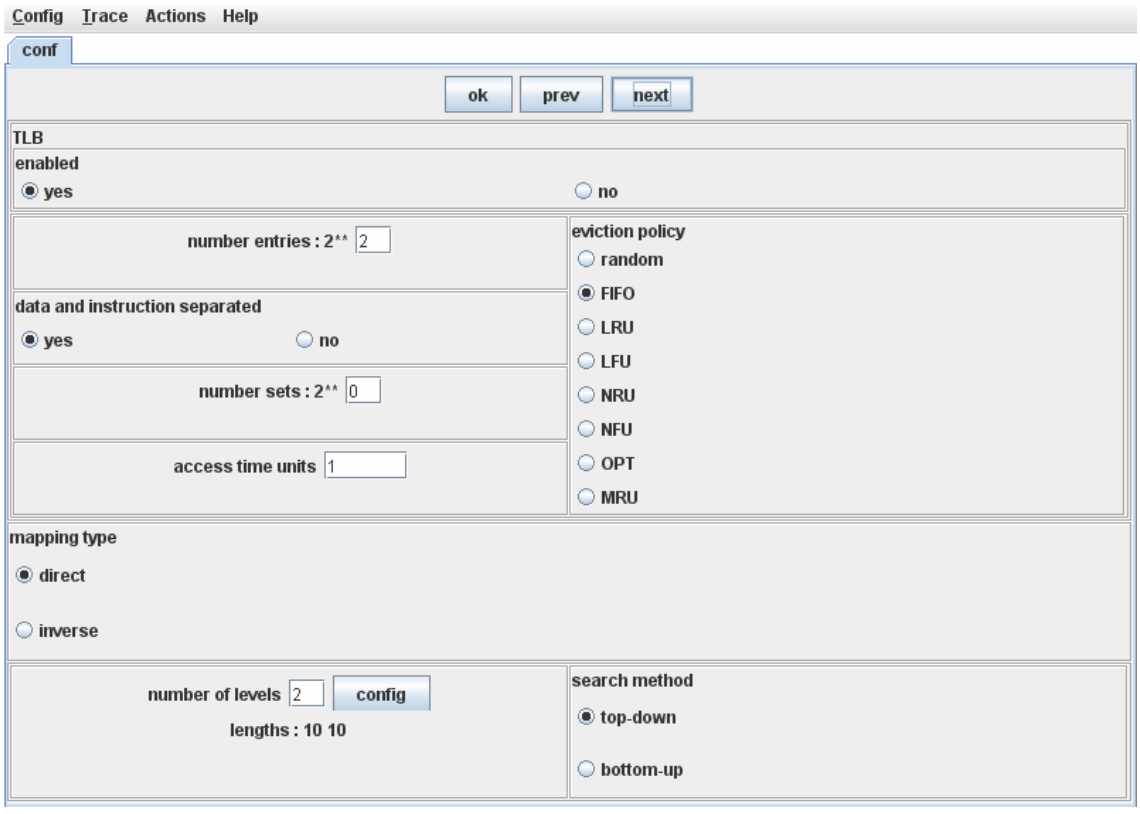
2.3.1 Pantalla 1



- Configuración general:
  - Tamaño de la memoria virtual (apartado 3.3.1)
  - Número máximo de procesos a cargar (apartado 3.3.1)
  - Penalización de acceso a disco (apartado 3.3.1)

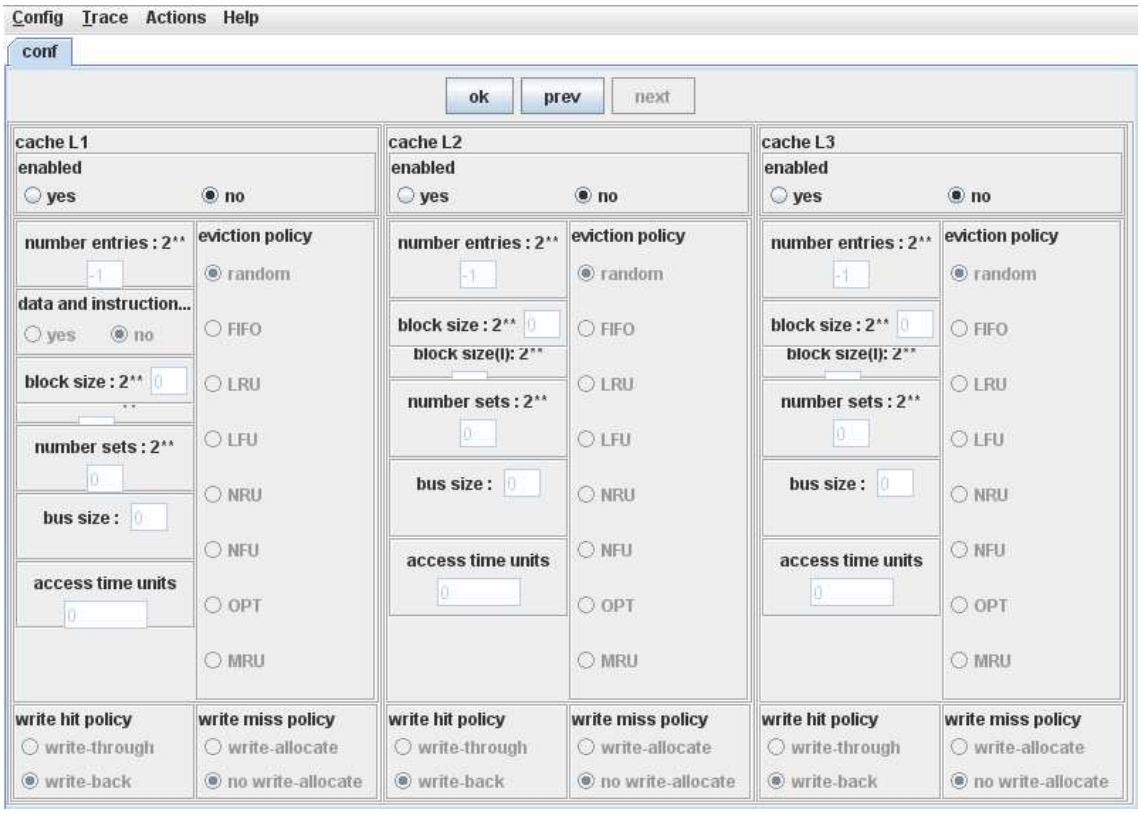
- Mem alloc. Asignación de memoria (apartado 3.3.5)
- Page aging. (apartado 3.3.7)
- Eviction policy (apartado 3.3.2)
- Main memory (apartado 3.3.1)

2.3.2 Pantalla 2



- TLB (apartado 3.3.3)
- Mapping type (apartado 3.3.6)

2.3.3 Pantalla 3



- Caché (apartado 3.3.4)

NOTA: La notación 2**n que aparece en gran parte de los campos de configuración, indicará la enésima potencia de 2.

3. SIMULANDO CON MNEME.

3.1 Código de colores

El MNEME provee un código de colores que permite al usuario detectar de forma visual, cómoda e inmediata cada situación que se produzca durante los pasos de la ejecución.

Para cada evento que ocurra las filas implicadas cambiarán de color. Antes de que un objeto sea eliminado (por ejemplo cuando una página es reemplazada en memoria principal y así no debe ser referenciada en la TLB) o reemplazado hay un retraso, de unos 2 segundos, antes de que desaparezca de la tabla correspondiente.

Los colores utilizados se muestran en la siguiente tabla, adjuntando la situación que representa cada uno de ellos.

COLOR	***SITUACIÓN***
	Tipo de instrucción FETCH o MEMREAD y el objeto está en caché
	Tipo de instrucción MEMWRITE y el objeto está en caché
	El objeto aún no se encuentra en caché
	Antes de que un objeto sea reemplazado
	Antes de que un objeto sea eliminado

3.2 Campos de información

Los campos de información que se muestran para las cachés, la TLB y la memoria principal tienen el siguiente significado:

- *i:* el número de entradas
- *KEY:* la dirección en caché o el número de página, en caso de tratarse de memoria principal
- *m:* 1 si el bloque ha sido modificado, 0 en caso contrario
- *pid:* el número de proceso que referencia la página

Los siguientes campos aparecen asociados a cada tipo específico de caché:

- *TAG, SET:* las direcciones se descomponen en TAG|SET|OFFSET. Las cachés completamente asociativas sólo tiene un conjunto o *set* (el número de bits del SET es 0) y las cachés de mapeado directo tienen para cada entrada en caché un conjunto diferente. Para la TLB no existe offset (la dirección es una dirección virtual) y para las cachés la entrada está compuesta por TAG y SET, que representan la página real / número de bloque. Este campo no aparece en los modelos de caché completamente asociativa.
- *type:* sólo aparece para las cachés que tienen los datos y las instrucciones separados. El valor de este campo puede ser I (instrucciones) o D (datos).
- *VAL:* este campo aparece sólo cuando el valor cargado en la caché tiene alguna importancia: en este caso para la TLB donde se cargan los números de página reales correspondientes a los números de páginas virtuales.

Los siguientes campos se muestran en función de la política de reemplazamiento que se esté usando:

- *ni:* para la política OPT, es el número de instrucción de la traza del proceso actual cuando el bloque/página se pone en la caché. De este modo el algoritmo puede ver qué instrucción del fichero de traza tiene la misma referencia a direcciones de memoria virtual.
- *timestamp:* es el valor de timestamp usado para el algoritmo FIFO
- *notUsed:* valor usado en *MRU* (Most Recently Used / Más Recientemente Usado) y *LRU* (Least Recently Used / Menos Recientemente Usado). Se trata de un contador para los tiempos en que los bloques de la cachés son referenciados (*read y write*).
- *used:* aparece para las políticas *LFU* (Least Frequently Used / Menos Frecuentemente Usado) y *NFU* (Not Frequently Used / No Frecuentemente Usado). Se trata de un contador para los tiempos en que los bloques son referenciados (*read y write*).
- *r:* aparece para la política *NRU* (Not Recently Used / No Recientemente Usado); toma valor 1 si el bloque es leído y 0 en otro caso. También se utiliza para la política NFU; toma valor 1 cuando el bloque es referenciado (*read y write*) y 0 en otro caso.

3.3 Tratamiento de la información

3.3.1 Memoria

La memoria virtual se divide en marcos de páginas y los procesos en páginas que no son más que trozos contiguos de memoria.

Decimos que ha habido un éxito cuando se intenta buscar una página en memoria principal y éste está presente. Si no esté presente, se dice que se ha producido un fracaso y se ha de traer la página de memoria secundaria (disco) a memoria principal.

El tiempo que se tarda en buscar una página en disco y llevarla a memoria principal, se denomina penalización de acceso a disco.

3.3.2 Políticas de reemplazo

Cuando un proceso necesita una página y todo el espacio de memoria virtual está ocupado es necesario reemplazar una de las páginas cargadas para introducir la nueva. La forma de elegir la página a ser reemplazada determina lo que se denomina políticas de reemplazo. Las políticas que se pueden configurar en Mneme son las siguientes:

Random: reemplaza una página al azar

FIFO: selecciona la página que más tiempo lleva en memoria

LRU: reemplaza la página que lleva más tiempo sin ser referenciada

LFU: reemplaza la página que menos veces ha sido referenciada

NRU: reemplaza la página que no haya sido usada recientemente

NFU: reemplaza la página que no haya sido frecuentemente usada

OPT: reemplaza la página que más tiempo ha de estar sin ser referenciada. Este algoritmo es sólo teórico.

3.3.3 TLB

Translation Lookaside Buffer (TLB) es un buffer que contiene partes de la tabla de páginas, es decir, relaciones entre direcciones virtuales y reales. Posee un número fijo de entradas y se utiliza para obtener la traducción rápida de direcciones. Si no existe una entrada buscada, se deberá revisar la tabla de páginas y tardará varios ciclos más, sobre todo si la página que contiene la dirección buscada no está en memoria principal. Si en la tabla de páginas no se encuentra la dirección buscada, se dará un fallo de página.

3.3.4 Caché

En base a los campos de información citados en el apartado 3.2, conviene conocer algunos de los tratamientos que se llevan a cabo sobre los datos de caché.

En función de la política de reemplazamiento que se haya configurado, los bloques que se reemplazarán serán:

- para la política *LFU*: la entrada que tenga el valor mínimo en el campo *used*.
- para la política *LRU*: la entrada que tenga el valor máximo en el campo *notUsed*.
- para la política *MRU*: la entrada que tenga el valor mínimo en el campo *notUsed*.
- para la política *NFU*: la entrada que tenga el valor mínimo para *used+r*.
- para la política *NRU*: las entradas se dividen en 4 clases: no leído y no modificado, no leído y modificado, leído y no modificado, leído y modificado. El bloque a reemplazar es elegido aleatoriamente entre las entradas que estén en la primera de las clases citadas anteriormente.
- para la política *random*: una entrada aleatoria
- para la política *FIFO*: la entrada que entró en primer lugar en caché (que tiene el menor valor para el campo *timestamp*).
- para la política *OPT*: la entrada que tenga el valor máximo para la expresión:

el índice de la siguiente instrucción en el fichero de traza para el proceso que tiene la misma referencia a memoria virtual (contando desde la instrucción actual del fichero de traza) / TUnits (configurada para este proceso)

La región en caché donde el algoritmo buscará en orden una página para reemplazar vendrá definido por el tipo de caché:

- si los datos e instrucciones están separados
- si es set-asocitiva: sólo mirará las entradas en el mismo conjunto en el que el nuevo nodo será colocado
- si la política de alojamiento en memoria es local: sólo mirará las páginas del mismo proceso.

3.3.5 Scheduling de los procesos

Al añadir un nuevo proceso se debe especificar el número de unidades de tiempo. Basándonos en este valor, el proceso puede encontrarse en dos colas: la cola de espera (*waiting queue / **W***) y la cola de ejecución (*executing queue / **E***).

Cuando un proceso inicia su ejecución entra automáticamente en la cola de ejecución, en la que permanecerá mientras su TUnits tenga un valor superior a 0 y no suceda un fallo de página en memoria. Con cada ejecución, las unidades de tiempo serán decrementadas en una unidad de la TUnits de cada uno de los procesos alojados en ambas colas. Cuando la TUnits del proceso actual llegue a 0, dicho proceso pasará a ocupar el último lugar de la cola de espera, con lo que el siguiente proceso de la cola de ejecución pasará a ser ejecutado.

Cuando tiene lugar un fallo de página en memoria, la ejecución actual pasa a ocupar el último lugar de la cola de espera. No obstante, a diferencia del caso anterior, la TUnits con la que entra en la cola de espera no es igual al TUnits configurado, sino que pasa a valer el número de unidades de tiempo que se ha indicado para el acceso a disco (*diskAccessTimeUnits*).

Así mismo, cuando el TUnits de un proceso en la cola de espera llega a ser 0, dicho proceso pasará a ocupar la última posición en la cola de ejecución.

3.3.6 Tabla de página

3.3.6.1 Mapeado directo

El mapeado directo de la tabla de página es jerárquico, suponiendo un número de niveles igual a N. El número de página virtual (en representación binaria) es dividida en N partes. Cada una de estas partes tiene un tamaño definido en la configuración (nótese que la suma de los tamaños debe ser igual al número de bits usados para la representación del número de página virtual).

Cuando se realiza una búsqueda sobre la tabla de página de una dirección virtual se pueden aplicar 2 métodos:

- *bottom-up:* de los niveles inferiores a los superiores.

- *top-down:* de los niveles superiores a los inferiores.

3.3.6.2 Mapeado inverso

Cada página virtual puede ser mapeada a un único frame físico porque el número de página virtual está hasheado al número de páginas en memoria. Por ello, un gran número de páginas virtuales pueden ser mapeadas a la misma página física.

En la tabla de página aparecen todos los número de direcciones virtuales ya mapeadas a un número de página física (el que esté actualmente cargado en memoria y está marcado con un 1).

Si la política de alojamiento es local (cada proceso tiene un número de páginas alojadas y no puede alojar páginas en el área de otro proceso) sólo un proceso puede estar al mismo tiempo en memoria.

3.3.7 Page aging

El *page aging* es un thread que se ejecuta después de que *mem ref* (valor especificado en la configuración) referencias sean realizadas sobre memoria principal. Cada página en memoria tiene una edad (age) que se inicializa a 0. Con cada referencia realizada (read o write) esta edad se incrementa un total de *ref inc units* unidades (valor que igualmente fue especificado en la configuración). Cuando el thread se lanza, su objetivo será la eliminación de todas aquellas páginas que tengan una edad menor o igual a 0 y decrementará la edad de cada página.

3.3.8 Memory Allocation (Alojamiento en memoria)

En memoria principal existen 2 políticas posibles para el alojamiento:

- *global:* cada proceso aloja una página en memoria principal cuando lo necesita, aunque para ello requiera reemplazar páginas de otros procesos.
- *local:* cuando un proceso necesita alojar una página se busca un espacio libre dentro de las páginas reservadas para el mismo proceso. En caso de no haber sitio disponible se realiza un reemplazamiento sobre la misma región, de modo que no se entromete con otros procesos.

Si hablamos de asignación de memoria local, la política de reemplazo será local, es decir, las páginas reemplazadas son las del propio proceso que provocó el fallo. En el caso que hablemos de asignación de memoria global, la política de reemplazo será global, es decir, cualquier página de la tabla de páginas puede ser candidata a ser reemplazada.

Un modelo de asignación de memoria cuando hay varios procesos en ejecución es la asignación por frecuencia de fallos de página (pff). Esta política trata de evitar la hiperpaginación o exceso de fallos de página de un proceso concreto. Cuando se producen demasiados fallos de página y en consecuencia numerosos accesos a disco, el modelo de memoria virtual para el proceso en cuestión deja de ser rentable, por lo que es necesario recurrir a un método que evite esta situación anómala.

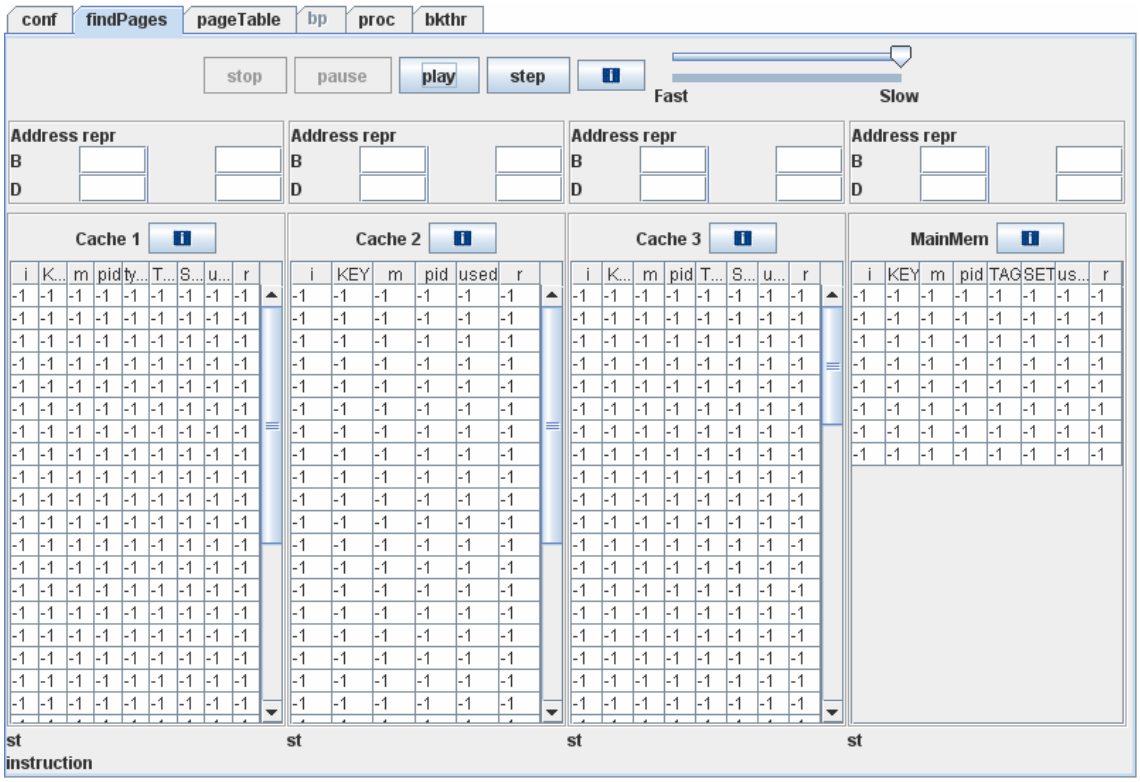
Existe un thread que se ejecuta cada vez que se reemplazan en la caché *NEvNodesToBeRun* nodos, valor que se especifica en el proceso de configuración. Para cada proceso la frecuencia de fallo de página (*PFF*) es definida como el número de páginas reemplazadas desde la última ejecución.

Se establecen unos umbrales máximos y mínimos permitidos (pff_{max} y pff_{min}) que son configurables.

Si la frecuencia de fallos del proceso actual es mayor que pff_{max}, esto quiere decir que el proceso está hiperpaginando. Para evitar esto, es necesario aumentar el número de páginas asociadas a ese proceso “robando” páginas a otro. En esta situación cabe hacerse una pregunta: ¿Cuál será el candidato ideal para ser robado?

El candidato ideal es aquel proceso cuyo pff actual no supere el umbral mínimo de frecuencia de fallos de página (cuyo pff sea menor que pff_{min}), es decir, un proceso que prácticamente no esté produciendo fallos de página y que la pérdida de algunas páginas no supondría un excesivo problema.

3.4 Pestaña findPages

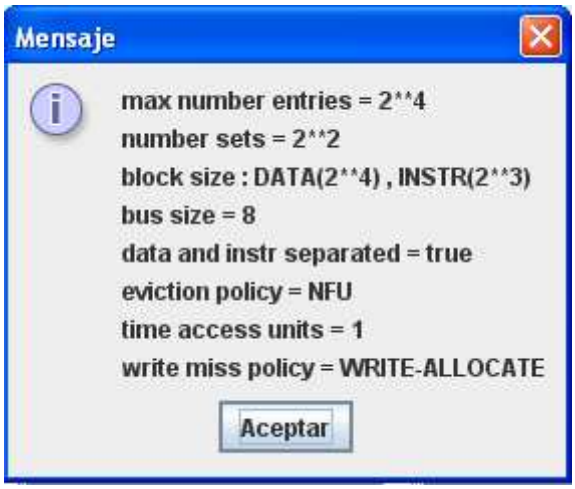


En la pestaña findPages, el MNEME nos ofrece de manera visual el estado actual de las distintas memorias que contiene la máquina configurada.

En la imagen presentada se pueden observar 3 niveles de caché y la memoria principal, los datos que el estado de cada posición de memoria están representados por tablas, donde cada columna es un atributo de los ya explicados y cada fila corresponde a una posición de memoria.

La imagen que nos ocupa muestra todas las casillas de las tablas a -1, lo que significa que están en estado inicial, es decir, aún no se han cargado datos en los distintos niveles de memoria.

En la cabecera de cada memoria podemos observar un botón con el símbolo de información. Cuando pulsamos uno de estos botones, nos aparecerá una ventana con información relativa a la configuración de la memoria correspondiente. Un ejemplo de esta ventana es el siguiente (correspondiente al nivel 1 de la caché que de la imagen anterior):



Nota: Estos datos se corresponden con los configurados en la fase de configuración.

En la parte superior de cada una de las tablas se nos presenta información relativa a la dirección de memoria a la que se accede en el momento actual. Esta información se presenta en decimal y binario.

Se puede obtener más información pulsando sobre "***Address repr***".

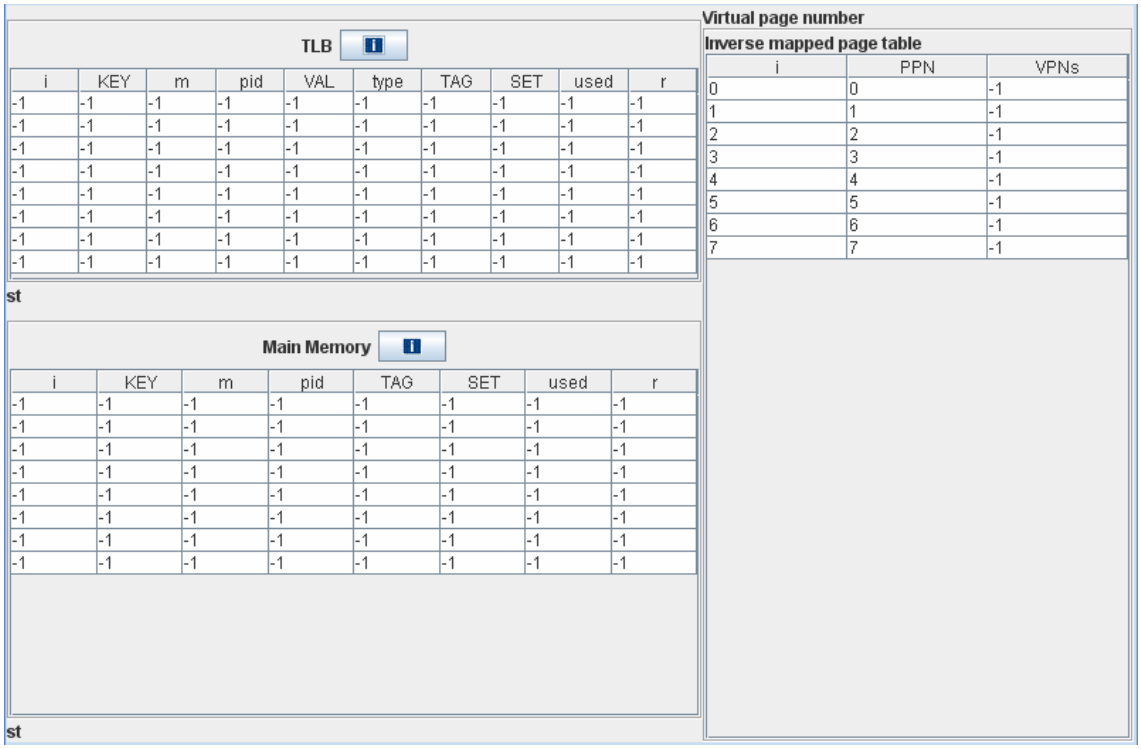
En la esquina inferior izquierda se nos presenta la instrucción que se estaba ejecutando.

La última herramienta que nos ofrece esta ventana es la posibilidad de controlar la velocidad de ejecución, para ello se nos ofrecen 4 botones:

- stop: Para la ejecución.
- pause: Detiene la ejecución en la instrucción actual.
- play: Ejecuta de manera secuencial las instrucciones de la traza, partiendo de la instrucción actual.
- step: ejecuta un paso.

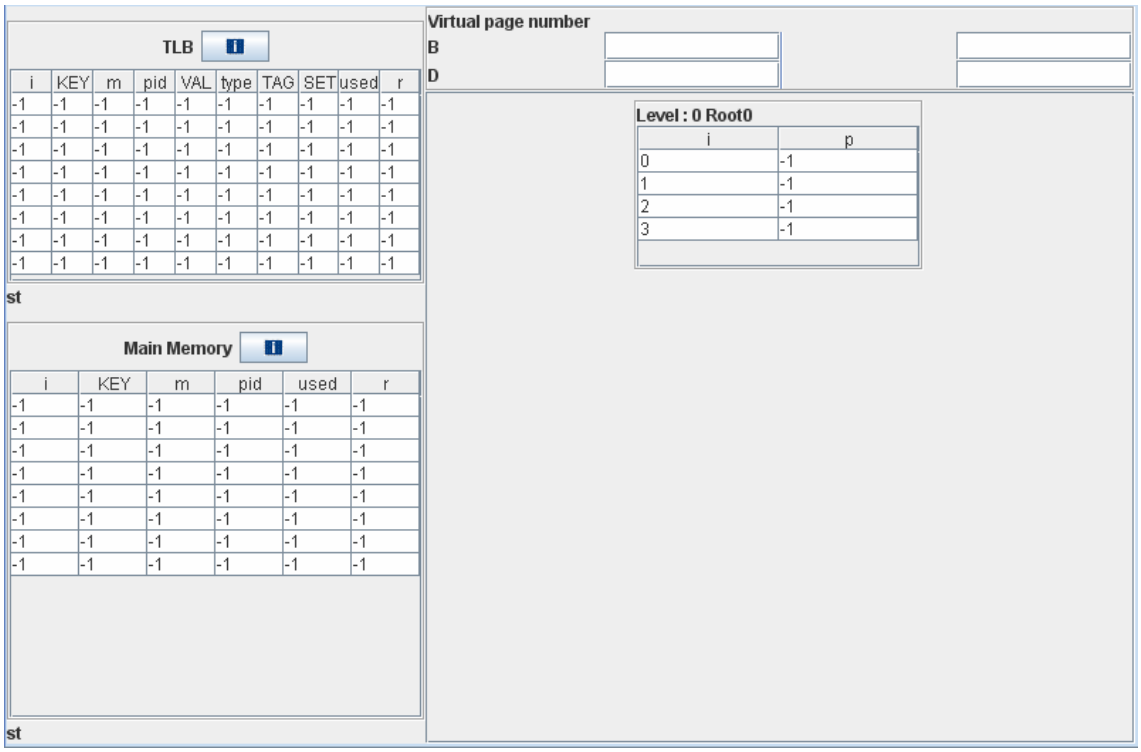
Además de una velocidad para controlar la velocidad a la que se ejecuta un paso.

3.5 Pestaña pageTable

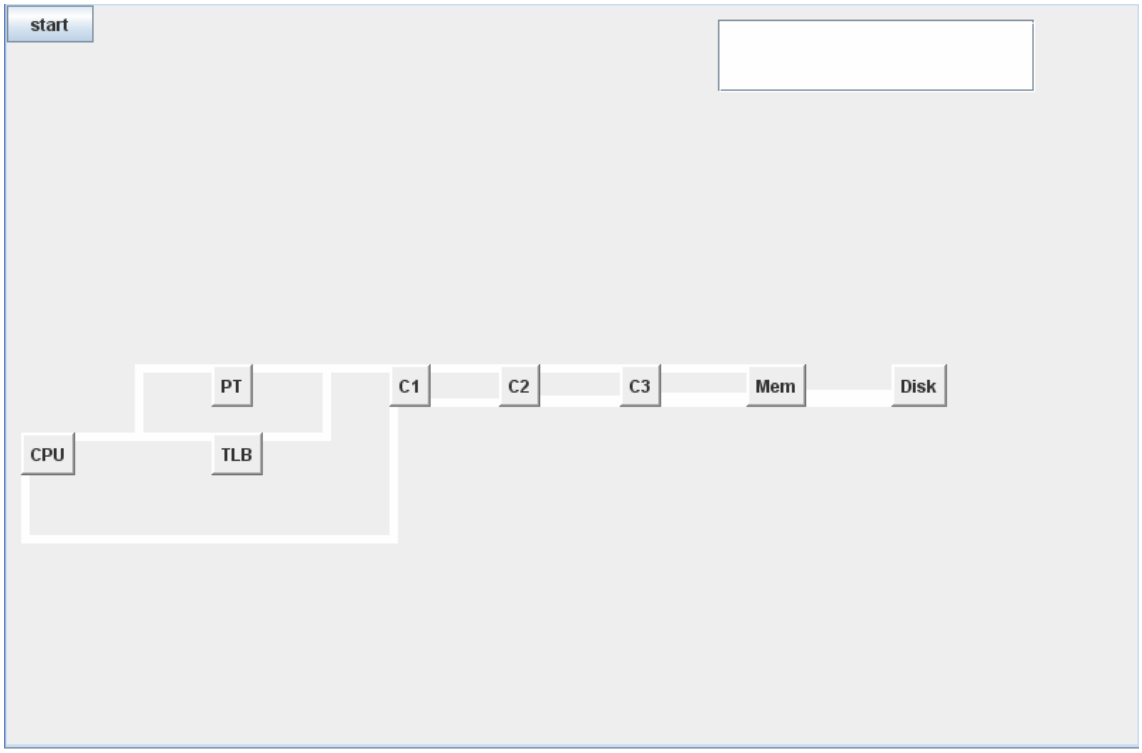


Otra de las vistas que nos ofrece el MNEME es la correspondiente a la vista de las memorias que intervienen en las páginas de tabla. En la imagen expuesta, correspondiente a un mapeado inverso, se pueden observar La memoria principal, TLB y la tabla de mapeo inverso (Un ejemplo de funcionamiento se verá en la traza 2).

En el caso de que el mapeado sea directo, la vista nos presentará el árbol de tablas de información para el mapeo, además de información sobre la dirección virtual (un ejemplo de mapeo directo se verá en la traza 1).



3.6 Pestaña bp



La pestaña bp nos presenta una imagen de la estructura de memorias y las acciones que se realizan en casa paso. Para ello habrá que pulsar el botón Start, ubicado en la esquina superior izquierda.

La cuadro de diálogo que aparece en la parte superior derecha, carece de verdadera utilidad.

3.7 Pestaña proc

En el programa MNEME el número de procesos a cargar estará entre 1 y 2^8 y un proceso puede estar en dos colas: Ejecución (E) y Espera (W).

Las colas de procesos se pueden visualizar en la pestaña proc una vez se ha cargado un fichero de traza:

Processes

pid	instr	TU	ni	cTuLeft	cQueue	cQInd
1	**view**	100	0	100	E	1
0	**view**	100	0	10	W	1
-		-	-	-	-	-
-		-	-	-	-	-

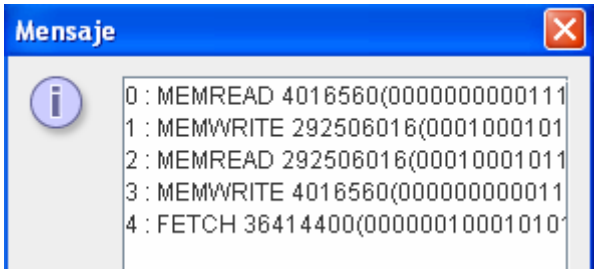
En la tabla mostrada se pueden visualizar los siguientes datos:

Pid → identificador del proceso

Instr → ver las instrucciones del programa

TU → tUnits del proceso Pid

Ni → número de la siguiente instrucción a ejecutar por el proceso Pid:



cTuLeft → tiempo restante en la cola

cQueue → cola (Ejecución (E) o Espera (W)). En este campo hay que decir que, en ocasiones, existe un bug ya que no siempre se actualiza. Veamos un ejemplo con dos procesos:

Processes

pid	instr	TU	ni	cTuLeft	cQueue	cQInd
1	**view**	2	6	2	E	1
0	**view**	2	4	2	W	2
-		-	-	-	-	-
-		-	-	-	-	-

Hay dos procesos cargados, el primero está en la primera posición en la cola de espera y el segundo proceso en la segunda posición de la cola de ejecución. Si hay sólo dos procesos: ¿Qué proceso está en la primera posición de la cola de espera (W)?.

Lo que ha sucedido es que el campo cQueue para el proceso con pid 0, no se ha actualizado: marca W y debería marcar E, es decir, que aunque la tabla marque que un proceso está en ejecución y el otro en espera, los dos procesos están en la cola de ejecución.

cQInd → posición del proceso Pid en la cola indicada en cQueue

Unas normas muy importantes a tener en cuenta durante la ejecución multiproceso serán las siguientes:

- Un proceso estará en cola de ejecución tanto tiempo como se indique en su TUnit a no ser que se produzca un fallo de página.
- Cuando se produce un fallo de página, el proceso que se está ejecutando actualmente pasa a ser el último de la cola de espera con una penalización igual al tiempo de acceso a disco (disk access TU).
- Si no se produce un fallo de página, y al proceso se le agota su tiempo, es decir, se supera el valor asignado a su TUnit, entonces el proceso pasa al último lugar de la cola de ejecución.
- Cuando la penalización de acceso a disco termine para el proceso que está en la primera posición de la cola de espera (proceso que provocó un fallo de página), entonces el proceso en cuestión pasará ser el último de la cola de ejecución.

4. EJEMPLOS DE SIMULACIÓN

4.1. Traza 1

Esta primera traza pretende reflejar un sencillo ejemplo con el que el usuario pueda familiarizarse con el entorno de ejecución y la manera en que se refleja cada condición.

Principalmente nos centraremos en mostrar tres características fundamentales de la configuración que cargaremos, y que se muestra en el siguiente apartado:

- mapeado directo
- algoritmo de sustitución LRU
- número de conjuntos = 0 → correspondencia directa

4.1.1 Configuración

```
<config>

<virtualAddressNBits>32</virtualAddressNBits>
<numberProcessesNBits>2</numberProcessesNBits>
<diskATU>10</diskATU>

<pageTable direct="true">
        <offsetLengths>
                <length>10</length>
                <length>10</length>
        </offsetLengths>
        <searchMethod>topdown</searchMethod>
        <tlbConfig>
                <numberEntriesNBits>3</numberEntriesNBits>
                <evictionPolicy>LRU</evictionPolicy>
                <dataInstrSeparated>true</dataInstrSeparated>
                <accessTimeUnits>1</accessTimeUnits>
        </tlbConfig>
</pageTable>

<pageAgingConfig>
        <pageAgingIncrease>2</pageAgingIncrease>
        <memRefToRun>5</memRefToRun>
</pageAgingConfig>

<memAllocConfig>
        <minPFF>3</minPFF>
        <maxPFF>6</maxPFF>
        <evNodesToRun>5</evNodesToRun>
</memAllocConfig>
```

```
<mainMemoryConfig>
        <numberEntriesNBits>2</numberEntriesNBits>
        <blockSizeNBits>12</blockSizeNBits>
        <evictionPolicy>LRU</evictionPolicy>
        <busSize>20</busSize>
        <accessTimeUnits>4</accessTimeUnits>
        <numberSetsNBits>0</numberSetsNBits>
        <dataInstrSeparated>false</dataInstrSeparated>
</mainMemoryConfig>

<cacheConfig>
        <numberEntriesNBits>4</numberEntriesNBits>
        <blockSizeNBits>4</blockSizeNBits>
        <blockSizeInstrNBits>3</blockSizeInstrNBits>
        <evictionPolicy>LRU</evictionPolicy>
        <busSize>8</busSize>
        <numberSetsNBits>2</numberSetsNBits>
        <accessTimeUnits>1</accessTimeUnits>
        <dataInstrSeparated>true</dataInstrSeparated>
        <hitWritePolicy>writeThrough</hitWritePolicy>
        <missWritePolicy>writeAllocate</missWritePolicy>
</cacheConfig>

<cacheConfig>
        <numberEntriesNBits>5</numberEntriesNBits>
        <blockSizeNBits>4</blockSizeNBits>
        <evictionPolicy>LRU</evictionPolicy>
        <accessTimeUnits>2</accessTimeUnits>
        <busSize>10</busSize>
        <numberSetsNBits>0</numberSetsNBits>
        <dataInstrSeparated>false</dataInstrSeparated>
        <hitWritePolicy>writeThrough</hitWritePolicy>
        <missWritePolicy>writeAllocate</missWritePolicy>
</cacheConfig>

<cacheConfig>
        <numberEntriesNBits>6</numberEntriesNBits>
        <blockSizeNBits>4</blockSizeNBits>
        <evictionPolicy>LRU</evictionPolicy>
        <busSize>16</busSize>
        <accessTimeUnits>3</accessTimeUnits>
        <numberSetsNBits>1</numberSetsNBits>
        <dataInstrSeparated>false</dataInstrSeparated>
        <hitWritePolicy>writeThrough</hitWritePolicy>
        <missWritePolicy>WriteAllocate</missWritePolicy>
</cacheConfig>

</config>
```

4.1.2 Traza

El fichero de traza a cargar contiene el siguiente código:

003d49b0	MEMREAD	9952
116f49a0	MEMWRITE	50
22ba3c0	MEMREAD	5097
311ba3c0	MEMWRITE	50
442ba3b8	MEMREAD	2642
22ba3c0	MEMREAD	5097
0c3d49b0	MEMWRITE	9952
311ba3c0	MEMWRITE	50
2011b8a0	FETCH	20
22ba3c0	FETCH	20

Este ejemplo tratará de realizar lecturas y escrituras sobre memoria, a fin de mostrar como se detectan los reemplazamientos de páginas. Por ello, conviene fijarse en las direcciones que utiliza cada instrucción.

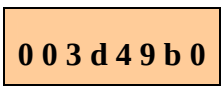
Conviene mencionar que la última instrucción de la traza (FETCH) ha sido añadida con posterioridad, por lo que su ejecución no aporta información alguna. No obstante, su utilidad reside en que nos permitirá ver el estado de la máquina después del FETCH intencionado. Esto se debe a que tras realizar la última ejecución, el simulador resetea de manera automática todos los campos, de modo que añadir una instrucción auxiliar al final es la manera de poder consultar estos campos.

4.1.3 Traducción de direcciones

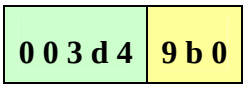
Tal y como se puede observar, las direcciones indicadas en el fichero de traza se expresan en formato hexadecimal. Esta información no se muestra directamente en el simulador, por lo que, para identificarla, se deben realizar unas pequeñas modificaciones.

Veamos como ejemplo, la traducción de la primera dirección del fichero:

- la dirección leída está compuesta por 8 dígitos en hexadecimal

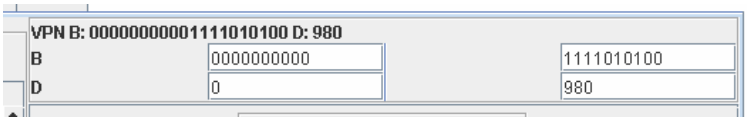


- de esta dirección separaremos los 5 dígitos (20 bits) de la izquierda y los 3 dígitos de la parte derecha (12 bits)



- la parte marcada de color verde se traduce a decimal y es el valor que podemos observar con la etiqueta VPN en la pestaña *pageTable*

H: 003d4 → D: 980



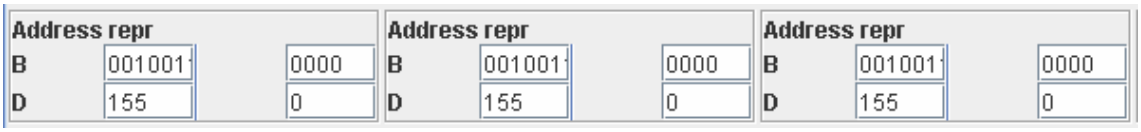
- la parte coloreada de color amarillo vuelve a dividirse en 2 y 1 dígitos tal y como se muestra a continuación

9 b	0

- ambas partes se traducen a decimal y se muestran en la información asociada a las cachés en la pestaña *findPages*

H: 9b → D: 155

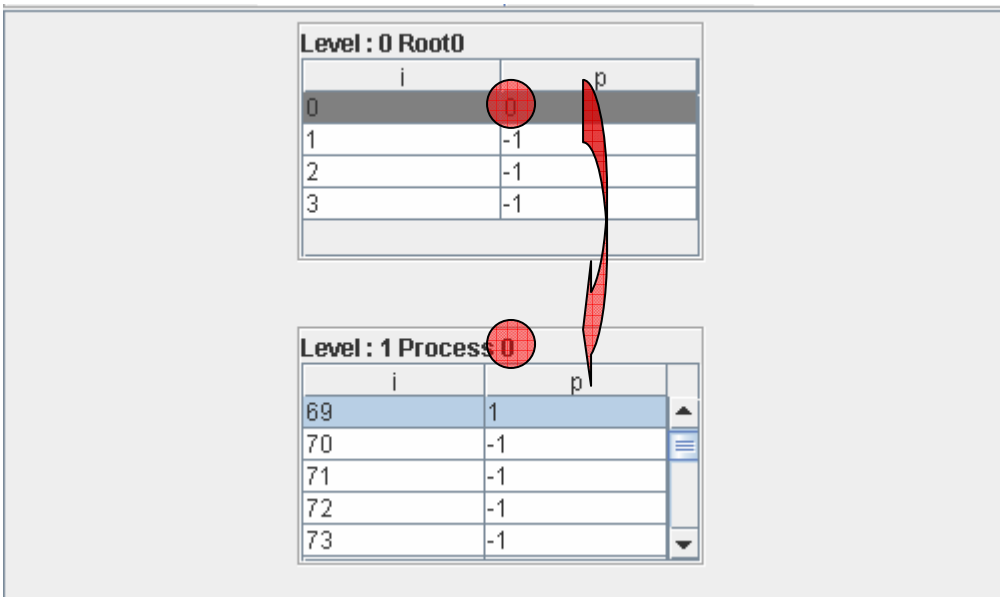
H: 0 → D: 0



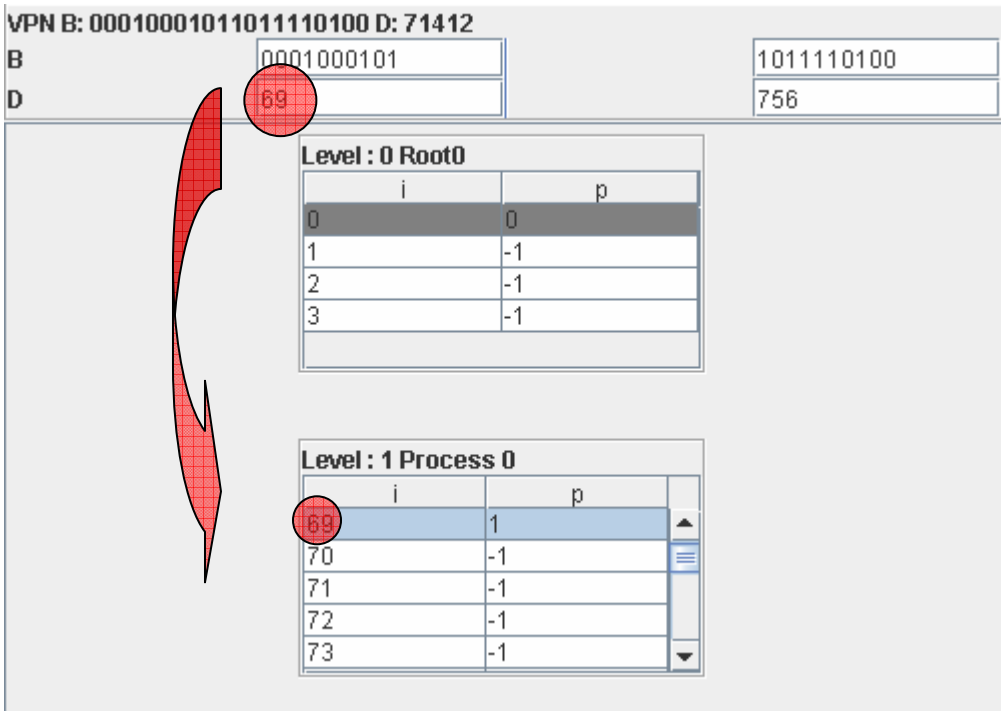
4.1.4 Tabla de página

Ahora centrémonos en la información que nos muestra la pestaña *pageTable*. En la zona derecha podemos observar cómo se utiliza la información de la dirección virtual sobre una configuración de mapeado directo.

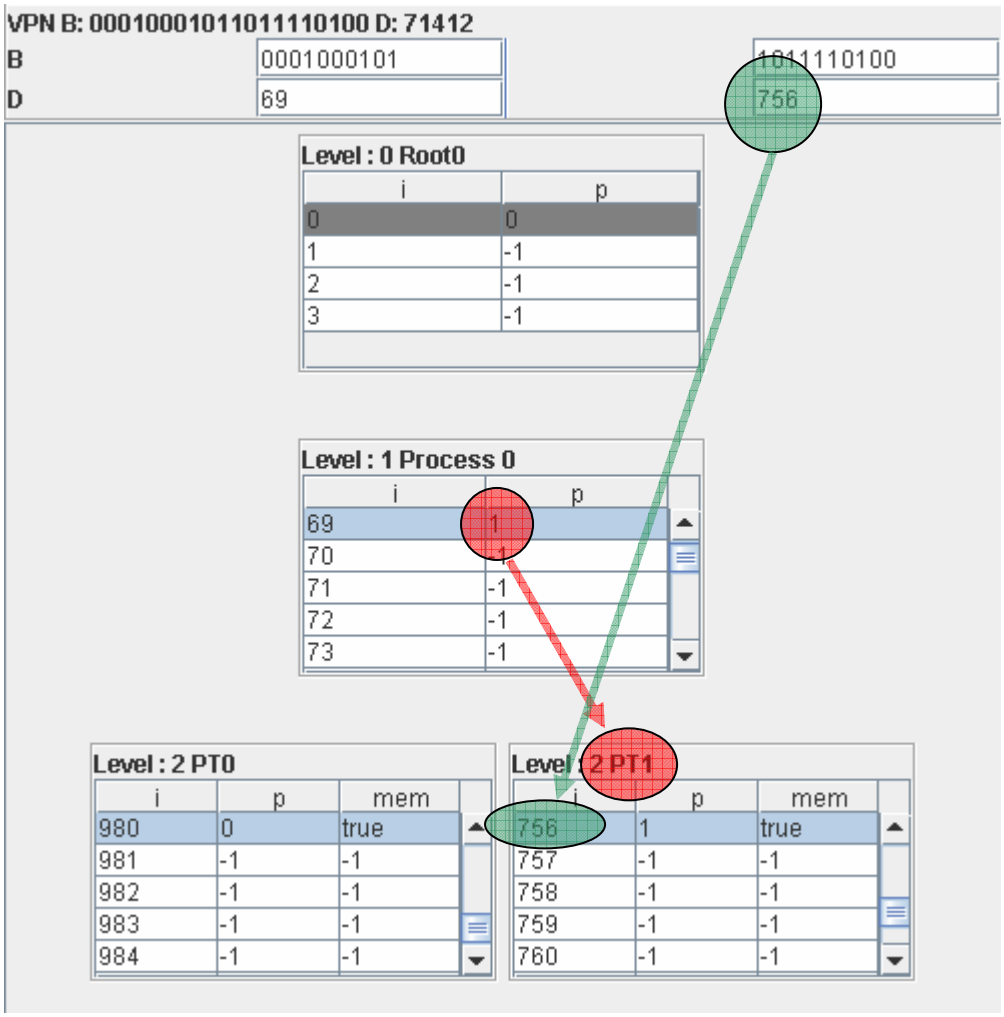
En la tabla raíz podemos observar el número del proceso cuya tabla debe consultarse. En este caso, dado que tenemos cargado un único proceso, esta información resulta trivial; no obstante, el uso se hace del mismo modo para un número de procesos mayor; en cuyo caso, el número de tablas mostrado en la segundo fila sería mayor que 1.



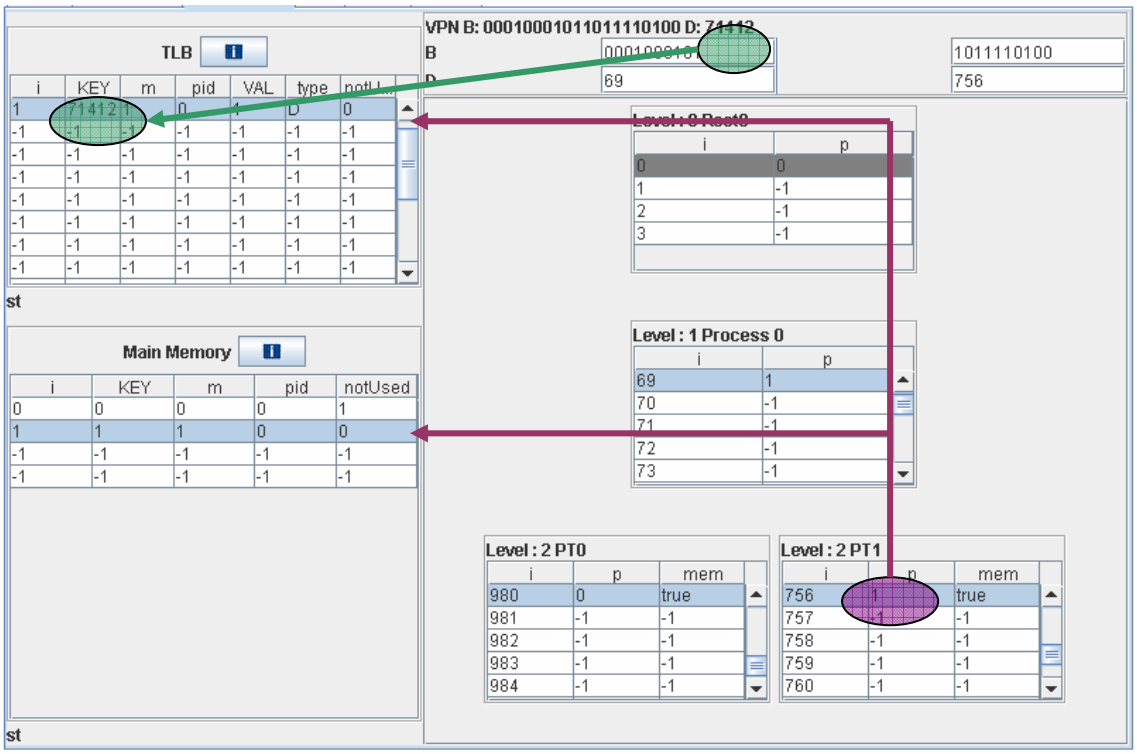
La parte izquierda de la dirección virtual, traducida a decimal, nos indexa la fila que debe consultarse dentro de la tabla actual.



Del mismo el valor nos retorna el campo “p” de esta fila nos señala la tabla de página (PT) que debe consultarse. La fila a consultar dentro de la misma vendrá indicada por la parte derecha de la dirección virtual (VPN).

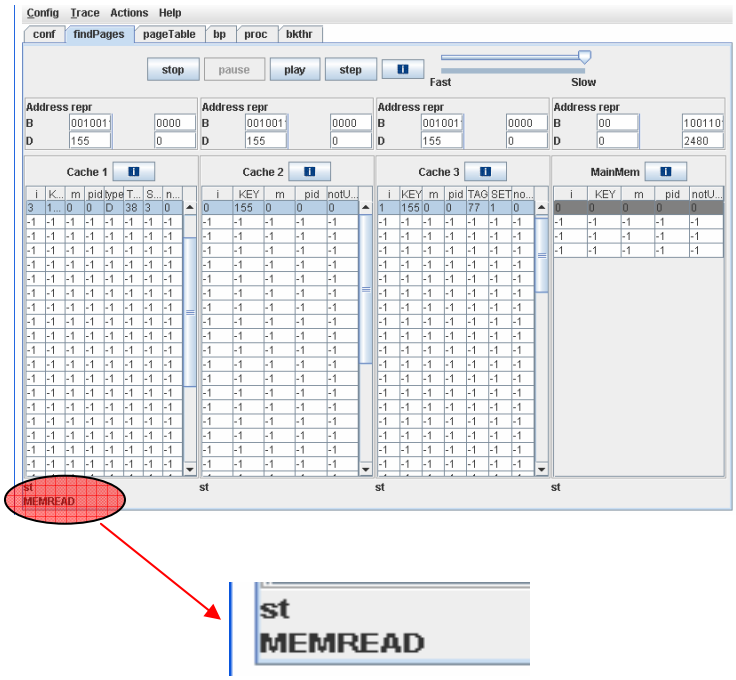


Gracias a estos valores podemos identificar unívocamente e interpretar las filas a consultar, tanto en memoria principal como en la TLB.



4.1.5 Instrucción en ejecución

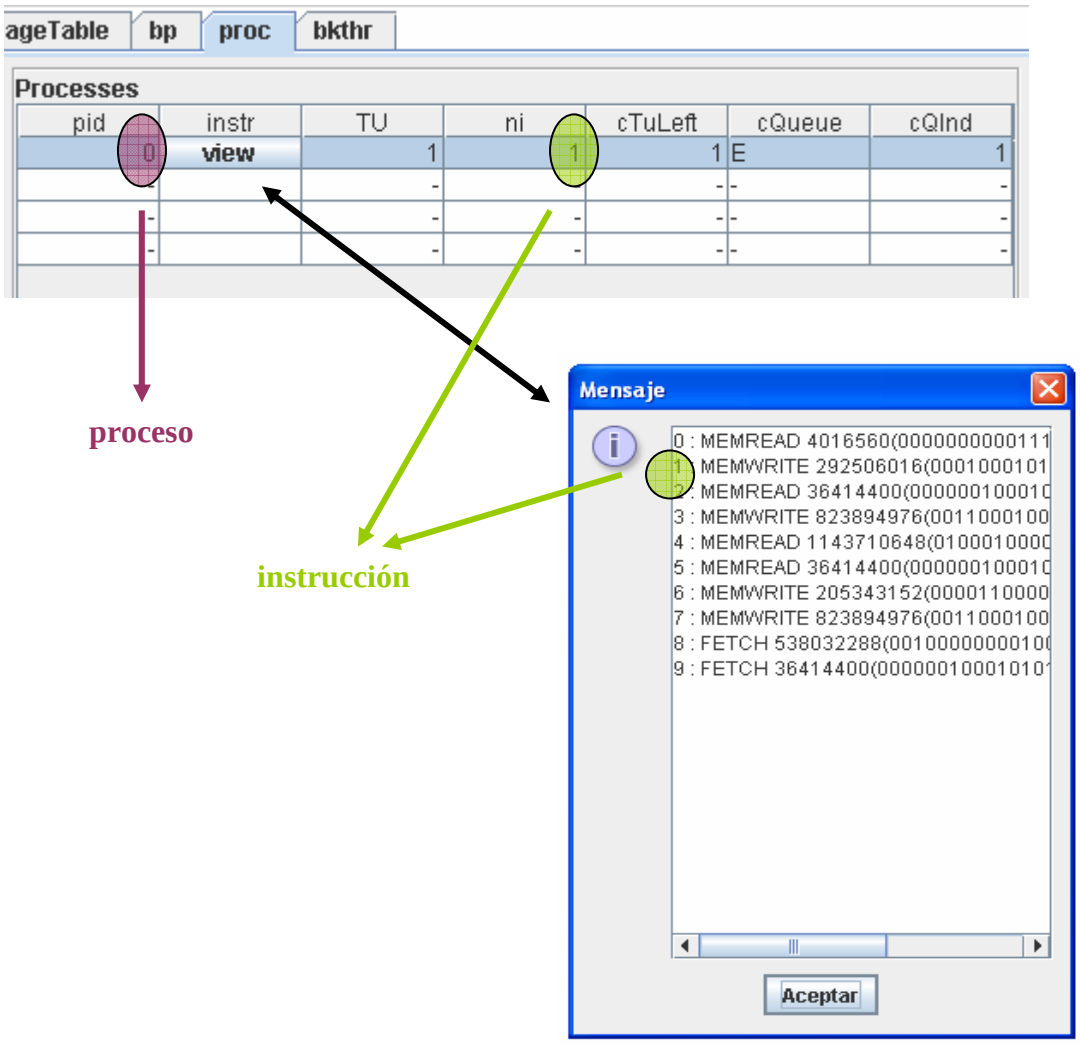
A diferencia de lo que pueda parecer, la instrucción indicada en la parte inferior de la pestaña *findPage* no indica la instrucción en ejecución.



Dicho valor indica la instrucción que se estaba realizando en el paso anterior. No obstante, tampoco se debe tomar como indicativo de la instrucción anterior, pues este

valor hace referencia los pasos y bien es sabido que una instrucción puede ejecutarse en más de un paso.

Por ello, para conocer la instrucción que se está ejecutando actualmente se ha de recurrir a la pestaña *proc*, zona process, donde podemos consultar el total de instrucciones que contiene el proceso sobre el que tenemos interés y la instrucción actual.

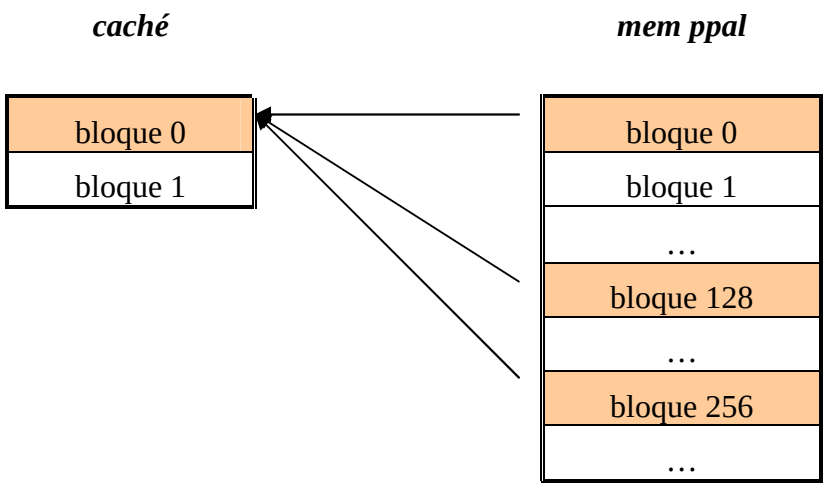


No obstante, se hará hincapié en este dato en próximas trazas.

4.1.6 Colisiones y reemplazamientos de páginas

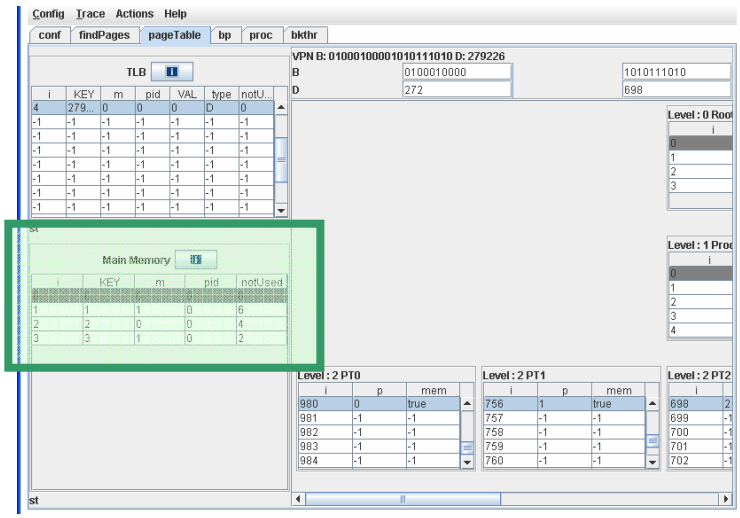
Dada la configuración especificada, en este ejemplo el reemplazamiento sobre las cachés se realizará de forma directa, pues se ha indicado (al poner el número de conjunto igual a 0) que la correspondencia sea directa. Esto significa que la posición en caché que le corresponde a cada bloque se obtiene mediante una operación aritmética (modulo de 128), con lo que no resulta necesario recurrir a ningún algoritmo de sustitución, a pesar de que éste necesite ser configurado.

Por ello, en caso de que se produzca conflicto alguno en caché, el reemplazamiento se resuelve con una sobreescritura automática.



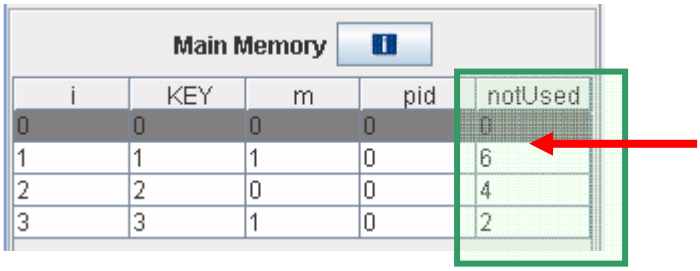
Por el contrario, en memoria si se requerirá la utilización del algoritmo de sustitución para determinar qué página debe ser reemplazada. Recordemos que se ha configurado un algoritmo LRU, por lo que el bloque candidato es aquel que menos recientemente haya sido utilizado. Para realizar dicha comprobación acudiremos al campo *notUsed*, tal y como se muestra en el siguiente ejemplo.

Para representar un ejemplo del uso de este algoritmo, esperamos alguna de las instrucciones donde se esté haciendo uso de una dirección que actualmente no esté cargada en memoria. Al ocurrir esta situación el simulador se ve forzado a determinar qué página, de entre las alojadas actualmente, debe ser reemplazada.



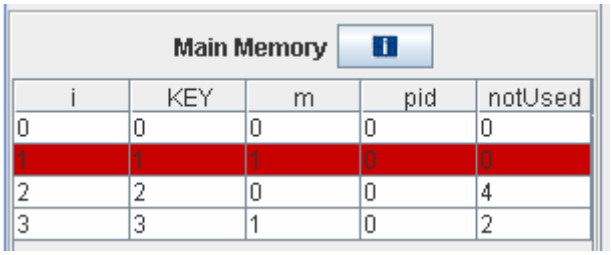
Tal y como se muestra en la figura anterior, para analizar los tres pasos que pueden observarse en el simulador nos centraremos en la información que nos oferta la pestaña *pageTable* sobre la memoria principal.

En el recuadro marcado, podemos observar que la situación en memoria, en el paso inmediatamente anterior a que se requiera una sustitución, indica que la página con índice 1 es la que tiene un mayor valor en el campo *notUsed*, por lo que es la página candidata a ser reemplazada. En caso de que existieran simultáneamente más de una página con el mismo valor *notUsed* máximo, se escogerá la candadita en orden de aparición.



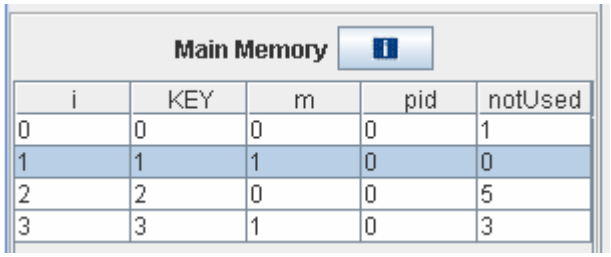
i	KEY	m	pid	notUsed
0	0	0	0	0
1	1	1	0	6
2	2	0	0	4
3	3	1	0	2

Al lanzar el siguiente paso de la ejecución se podrá comprobar cómo, efectivamente, la página a reemplazar (indicada por el simulador en color rojo) es la de índice 1.



i	KEY	m	pid	notUsed
0	0	0	0	0
1	1	1	0	0
2	2	0	0	4
3	3	1	0	2

En el mismo paso, inmediatamente después de indicar en rojo la página reemplazada, se insertar la nueva página y se actualizan/incrementan los valores *notUsed* de todas las páginas alojadas.

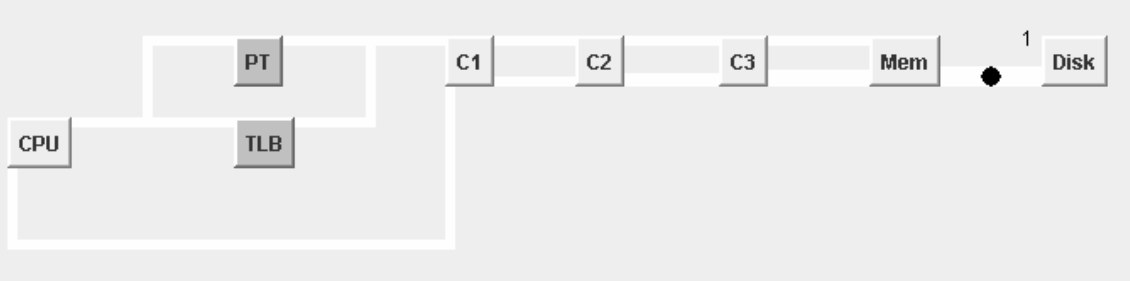


i	KEY	m	pid	notUsed
0	0	0	0	1
1	1	1	0	0
2	2	0	0	5
3	3	1	0	3

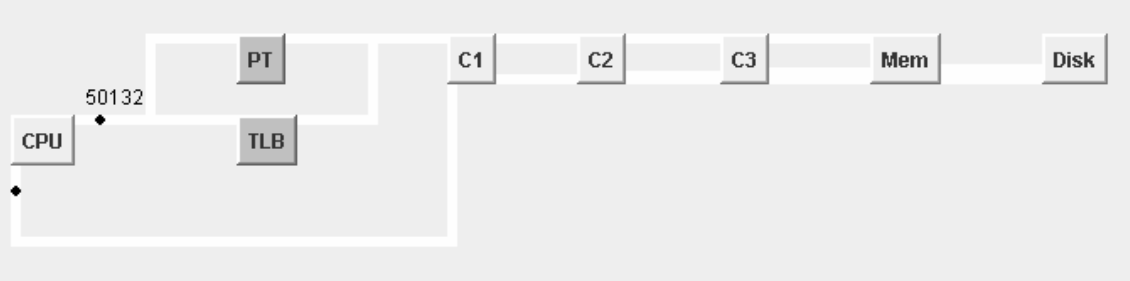
OJO: *el reseteo del valor en el camp notUsed de la página a reemplazar se realiza en cuanto el paso de ejecución se inicia; es decir, se resetea antes de ser marcada en rojo. Por ello se recomienda comprobar este campo antes de lanzar el siguiente paso (en el que se realiza la sustitución).*

Por último, cabe destacar de este sencillo ejemplo la posibilidad de consultar en la pestaña *bp* los pasos que se siguen para realizar la sustitución indicada anteriormente. Dado que la estructura de esta pestaña ya fue especificada en puntos anteriores, a continuación se muestran las tres fases que se pueden apreciar en la interfaz gráfica (para la misma ejecución que se refleja en las capturas anteriores de este mismo apartado).

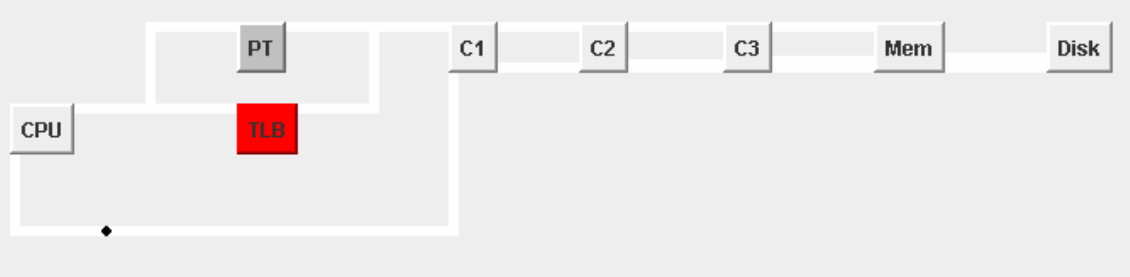
1) Se carga la página desde disco a memoria



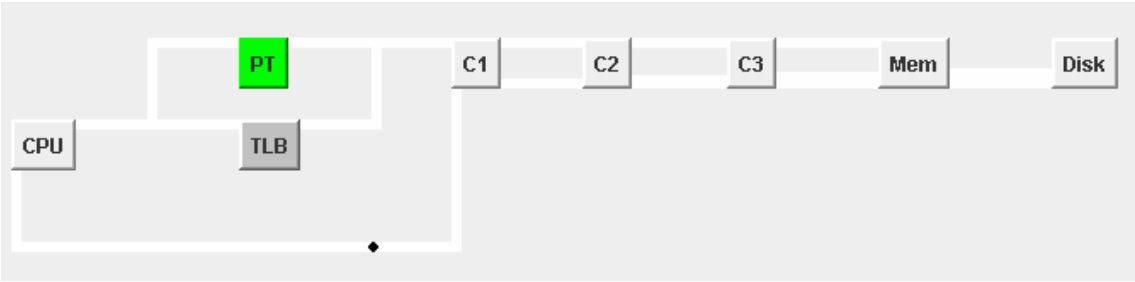
2) Desde la CPU salen la dirección virtual (línea superior) y el dato (línea inferior)



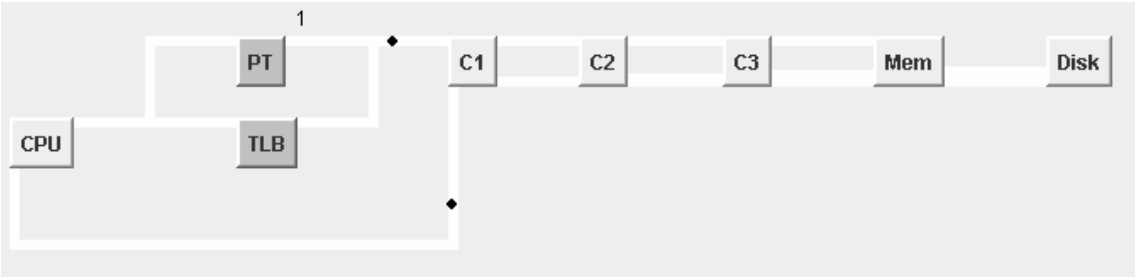
3) La dirección se consulta en la TLB y se detecta que no está disponible:



4) La misma dirección sale de la TLB hasta la PT (tabla de página) donde se encuentra el valor:



5) Una vez resuelta la dirección física, tanto ésta como la información viajan hasta la caché de nivel 1:



4.2. Traza 2

En la siguiente traza se pretende, ante todo, ver un ejemplo de mapeado inverso para el MNEME, así como la utilización de otra política de reemplazamiento.

4.2.1 Configuración

El fichero de configuración que se utilizará para el siguiente ejemplo es el que continúa:

```
<config>

<virtualAddressNBits>32</virtualAddressNBits>
<numberProcessesNBits>2</numberProcessesNBits>
<diskATU>10</diskATU>
```

```
<pageTable direct="false">
        <hashAnchorSizeNBits>3</hashAnchorSizeNBits>
        <tlbConfig>
                <numberEntriesNBits>2</numberEntriesNBits>
                <evictionPolicy>NFU</evictionPolicy>
                <dataInstrSeparated>true</dataInstrSeparated>
                <numberSetsNBits>1</numberSetsNBits>
                <accessTimeUnits>1</accessTimeUnits>
        </tlbConfig>
</pageTable>

<pageAgingConfig>
        <pageAgingIncrease>2</pageAgingIncrease>
        <memRefToRun>5</memRefToRun>
</pageAgingConfig>

<memAllocConfig>
        <minPFF>3</minPFF>
        <maxPFF>6</maxPFF>
        <evNodesToRun>5</evNodesToRun>
</memAllocConfig>

<mainMemoryConfig>
        <numberEntriesNBits>3</numberEntriesNBits>
        <blockSizeNBits>12</blockSizeNBits>
        <evictionPolicy>NFU</evictionPolicy>
        <busSize>20</busSize>
        <accessTimeUnits>4</accessTimeUnits>
        <numberSetsNBits>0</numberSetsNBits>
        <dataInstrSeparated>false</dataInstrSeparated>
</mainMemoryConfig>

<cacheConfig>
        <numberEntriesNBits>4</numberEntriesNBits>
        <blockSizeNBits>4</blockSizeNBits>
        <blockSizeInstrNBits>3</blockSizeInstrNBits>
```

```
        <evictionPolicy>NFU</evictionPolicy>
        <busSize>8</busSize>
        <numberSetsNBits>2</numberSetsNBits>
        <accessTimeUnits>1</accessTimeUnits>
        <dataInstrSeparated>true</dataInstrSeparated>
        <hitWritePolicy>writeBack</hitWritePolicy>
        <missWritePolicy>writeAllocate</missWritePolicy>
</cacheConfig>

<cacheConfig>
        <numberEntriesNBits>5</numberEntriesNBits>
        <blockSizeNBits>4</blockSizeNBits>
        <evictionPolicy>NFU</evictionPolicy>
        <accessTimeUnits>2</accessTimeUnits>
        <busSize>10</busSize>
        <numberSetsNBits>0</numberSetsNBits>
        <dataInstrSeparated>false</dataInstrSeparated>
        <hitWritePolicy>writeBack</hitWritePolicy>
        <missWritePolicy>writeAllocate</missWritePolicy>
</cacheConfig>

<cacheConfig>
        <numberEntriesNBits>6</numberEntriesNBits>
        <blockSizeNBits>4</blockSizeNBits>
        <evictionPolicy>NFU</evictionPolicy>
        <busSize>16</busSize>
        <accessTimeUnits>3</accessTimeUnits>
        <numberSetsNBits>1</numberSetsNBits>
        <dataInstrSeparated>false</dataInstrSeparated>
        <hitWritePolicy>writeBack</hitWritePolicy>
        <missWritePolicy>writeAllocate</missWritePolicy>
</cacheConfig>

</config>
```

4.2.2 Traza

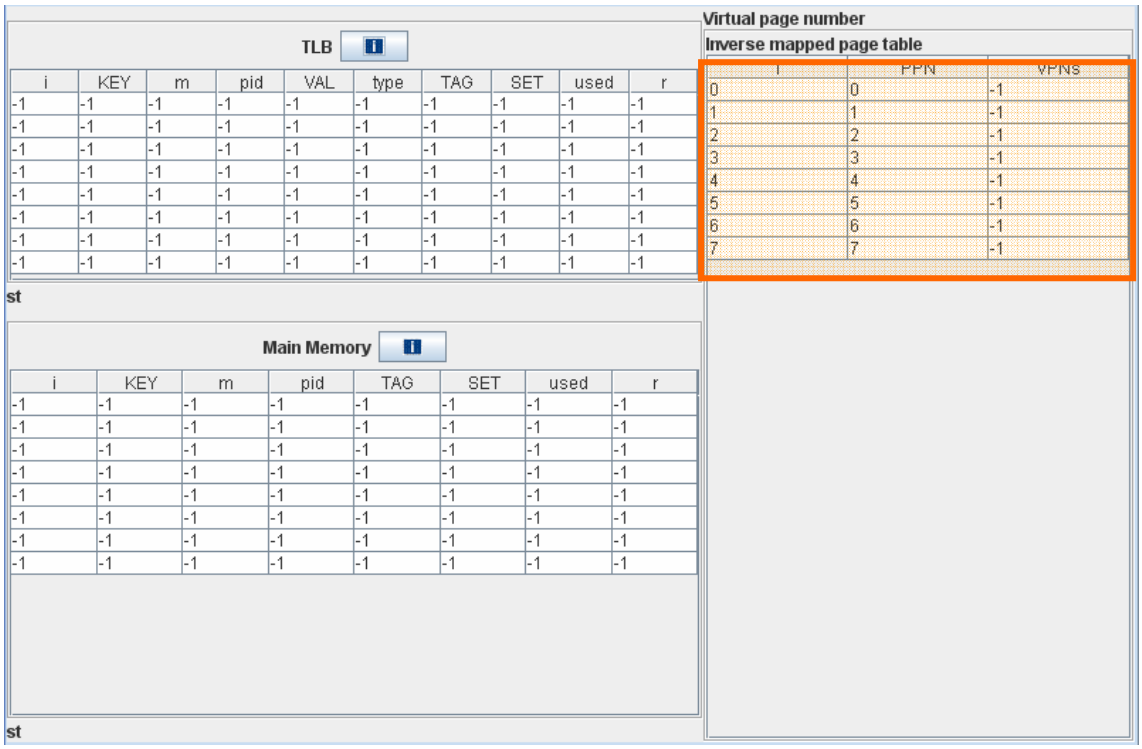
La traza a utilizar es prácticamente la misma que en el ejemplo anterior, salvo por la modificación de algunas direcciones virtuales:

```
003d49b0 MEMREAD     9952
116f49a0 MEMWRITE    50
22ba3c0 MEMREAD      5097
311ba3d0 MEMWRITE    50
442bb3b8 MEMREAD     2642
003d49b0 MEMREAD     9952
116f49a0 MEMWRITE    50
22ba3c0 MEMREAD      5097
311ba3d0 MEMWRITE    50
442bb3b8 MEMREAD     2642
22ba3c0 FETCH    5097
22ba3c0 FETCH    5097
```

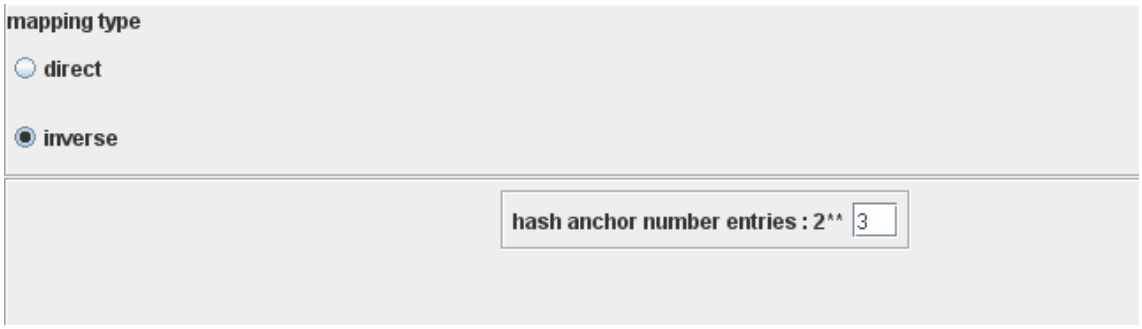
Nuevamente, hay que comentar que se ha añadido una instrucción al final para poder observar el estado de las memorias, ya que, el MNEME, una vez termina la ejecución de la última instrucción resetea el estado de la memoria.

4.2.3 Tablas de páginas para mapeado inverso y ejecución de la traza.

Como se ha comentado anteriormente, esta traza intentará mostrar la ejecución de un ejemplo de mapeado inverso en la aplicación MNEME. Así, pues, es interesante echar un vistazo a la y explicar como funciona la pestaña pageTable cuando la máquina simulada ha sido configurada con esta opción:



En esta pestaña aparece un elemento nuevo, la tabla de mapeo inverso, esta consta de 3 columnas y 2**n filas, n lo hemos configurado en la fase de configuración:



Las columnas son:

i → Posición de la tabla de mapeo.

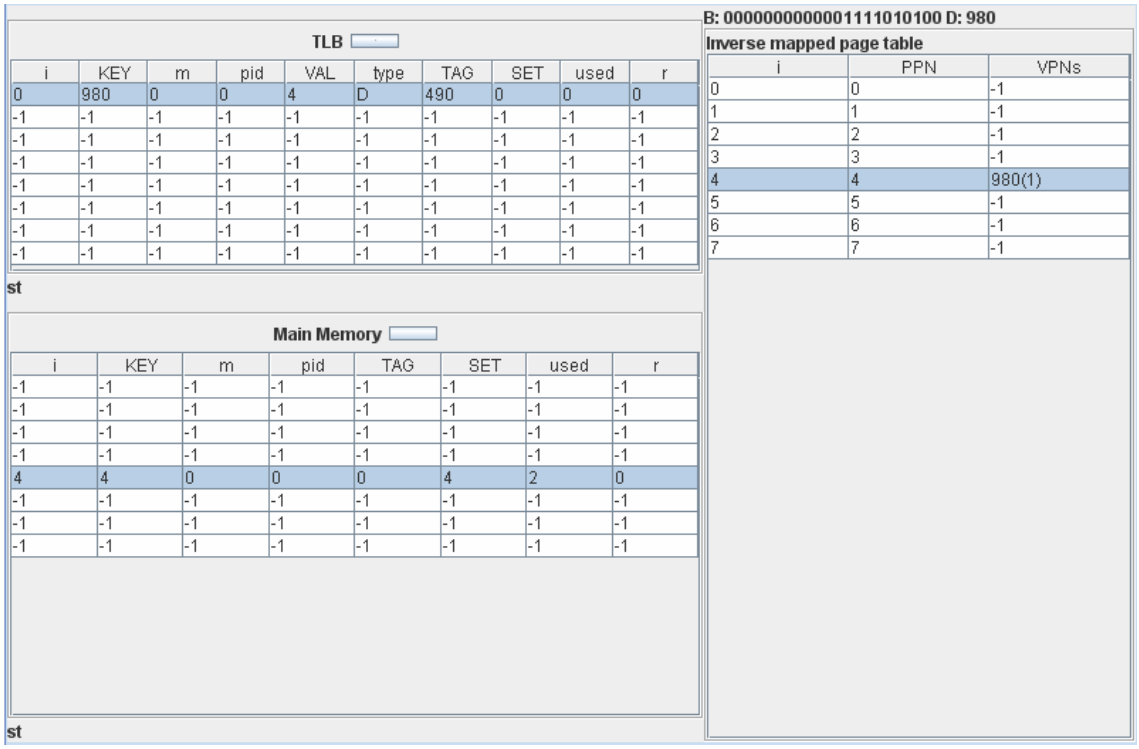
PPN → Posición que ocupa la página en la memoria principal.

VPNs → Página que ha pasado por esa posición, aquellas cargadas actualmente en memoria, se representan por un “(1)”, aquellas no cargadas, tienen un “(0)”

4	4	980(0),71412(1)

Página cargada en memoria → 71412

En la siguiente imagen se puede observar que hay una página cargada en tlb, tabla de mapeo y memoria principal, ¿pero qué proceso ha llevado a esta situación?



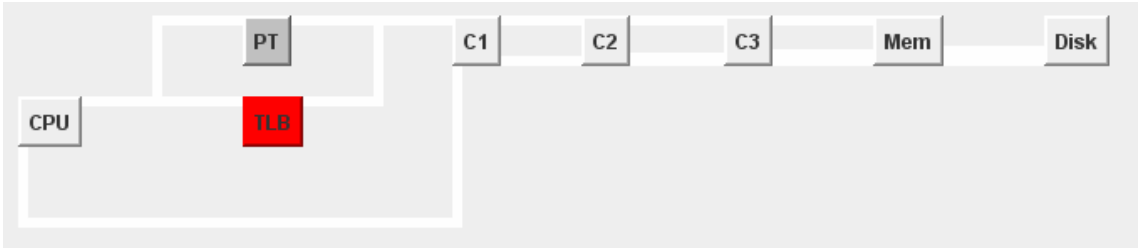
i	KEY	m	pid	VAL	type	TAG	SET	used	r
0	980	0	0	4	D	490	0	0	0
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

i	PPN	VPNs
0	0	-1
1	1	-1
2	2	-1
3	3	-1
4	4	980(1)
5	5	-1
6	6	-1
7	7	-1

i	KEY	m	pid	TAG	SET	used	r
-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1
4	4	0	0	0	4	2	0
-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1

La respuesta a esta pregunta es sencilla:

Esa página, no se encontraba anteriormente en la TLB (esto se puede comprobar en la pestaña bp):



Así que la cargamos en la TLB, como está vacía y, revisando la configuración, el número de conjuntos asociativos es de 2**1, deberemos ponerla en la primera posición del conjunto que le corresponda. Este conjunto se consigue haciendo el módulo 2**1 de la dirección de página:

→ 00…0001111010100 % 2 = 0

La página pertenece al conjunto 0.

La posición correspondiente en la memoria principal nos la dirá la tabla de mapeo, para ello deberemos saber que posición en la tabla le corresponde a la página, para ello se hace el módulo dirección por tamaño de la tabla de mapeo:

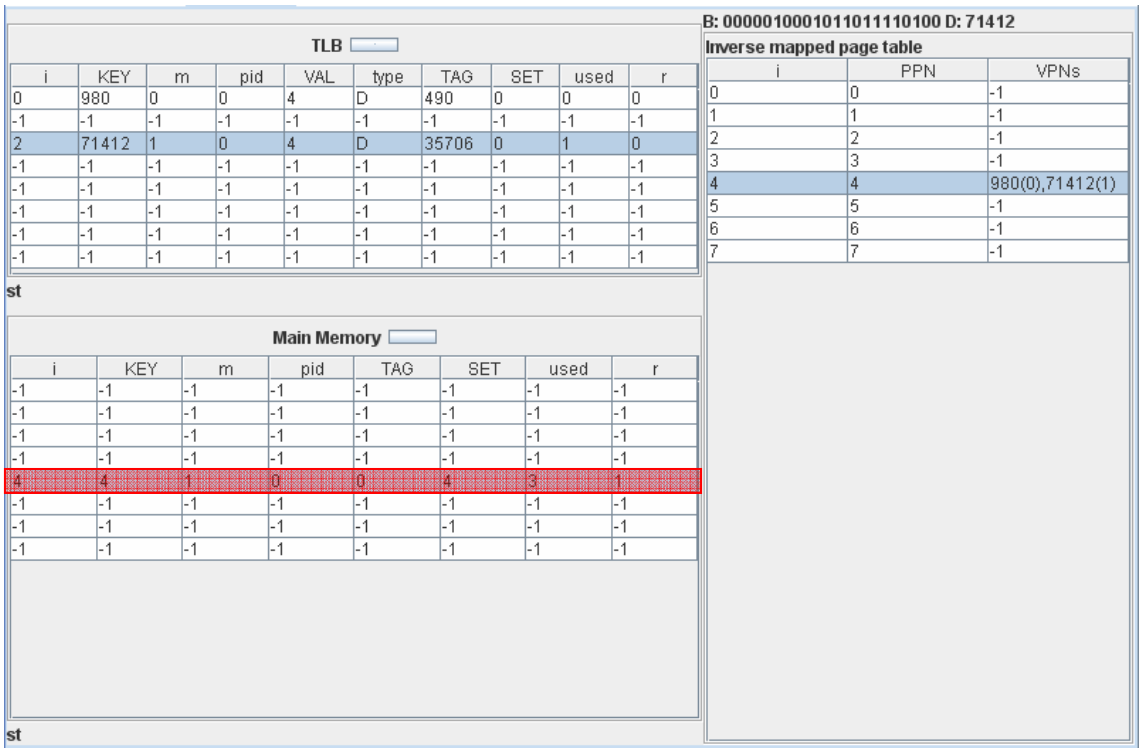
→ 00…0001111010100 % 8 = 4

En esta posición, de la tabla encontramos lo siguiente:

PPN = 4 → Se busca la página en la posición 4 de la memoria principal.

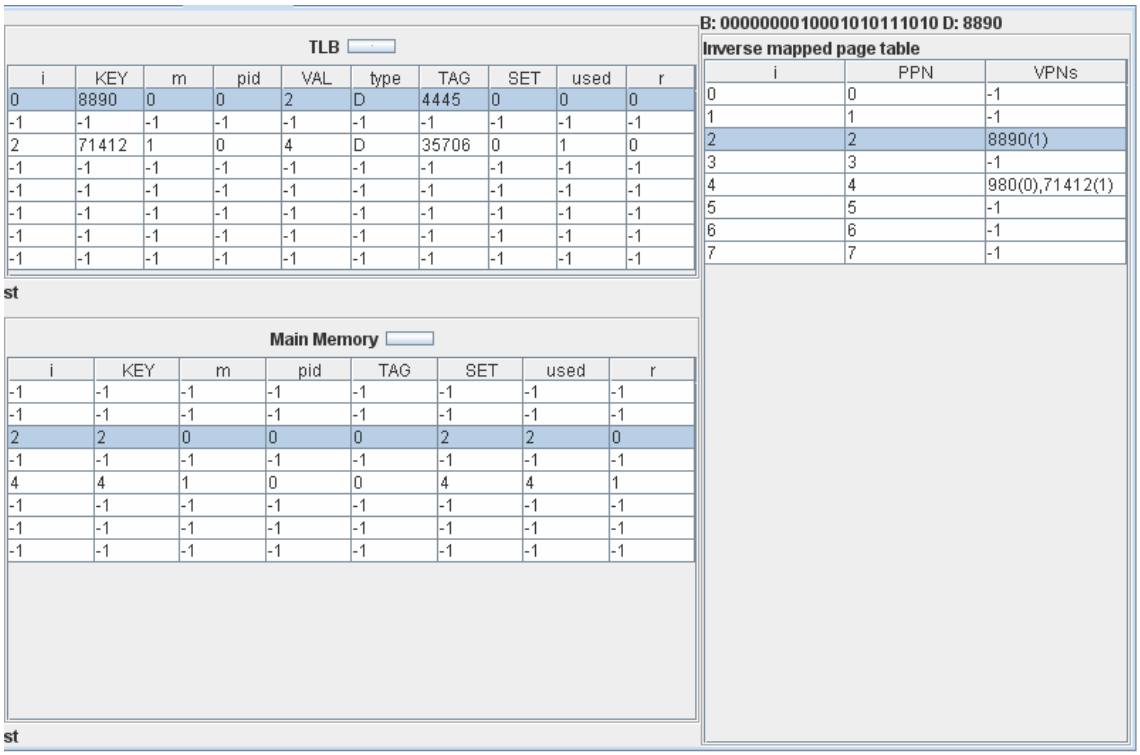
VPNs = 980(1) → La página 980 está cargada en memoria principal.

Siguiendo el mismo método se intenta cargar otra instrucción, y resulta que la dirección de página pertenece al mismo conjunto asociativo que la instrucción anterior, con lo que la dirección se guardará en la siguiente dirección libre del conjunto asociativo 0 (posición 2 de la TLB), ya que este aún tiene posiciones de memoria libres.



Podemos comprobar que además, también le corresponde la misma posición en la tabla de mapeado, lo que se traduce en una colisión en la memoria principal. Esto se representa en la tabla de mapeado, poniendo a 0 el indicador de la página anterior (marcándola como que ya no se encuentra en memoria) y a 1 el indicador de la nueva página.

Cargando una nueva instrucción (imagen siguiente), se puede observar cómo esta vez la dirección de página corresponde al mismo conjunto asociativo, de la TLB, que las anteriores y que por ello se produce una colisión y se actualiza esta posición. En cuanto a la tabla de mapeo y la memoria principal, esta vez, se accede a la posición 2 de la tabla de dispersión, que también corresponde a la posición 2 de la memoria principal. Al estar esta libre, se representará en la tabla de dispersión como que la página actual es la única y que además está cargada en memoria.



TLB

i	KEY	m	pid	VAL	type	TAG	SET	used	r
0	8890	0	0	2	D	4445	0	0	0
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
2	71412	1	0	4	D	35706	0	1	0
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

st

Main Memory

i	KEY	m	pid	TAG	SET	used	r
-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1
2	2	0	0	0	2	2	0
-1	-1	-1	-1	-1	-1	-1	-1
4	4	1	0	0	4	4	1
-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1

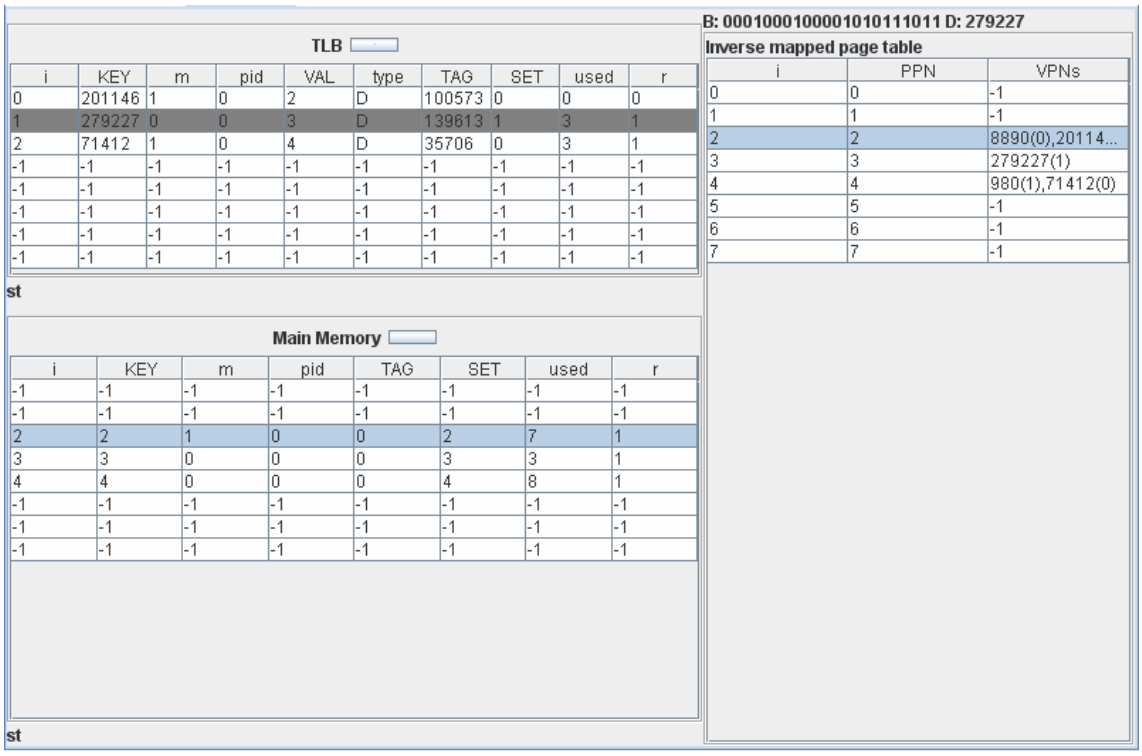
st

B: 00000000100010101111010 D: 8890

Inverse mapped page table

i	PPN	VPNs
0	0	-1
1	1	-1
2	2	8890(1)
3	3	-1
4	4	980(0),71412(1)
5	5	-1
6	6	-1
7	7	-1

Con ánimo de no hacer tediosa el estudio de la traza, ya que los siguientes pasos se repiten casos ya vistos con anterioridad, se avanzará hasta la posición anterior a la última instrucción (el fetch).



i	KEY	m	pid	VAL	type	TAG	SET	used	r
0	201146	1	0	2	D	100573	0	0	0
1	279227	0	0	3	D	139613	1	3	1
2	71412	1	0	4	D	35706	0	3	1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

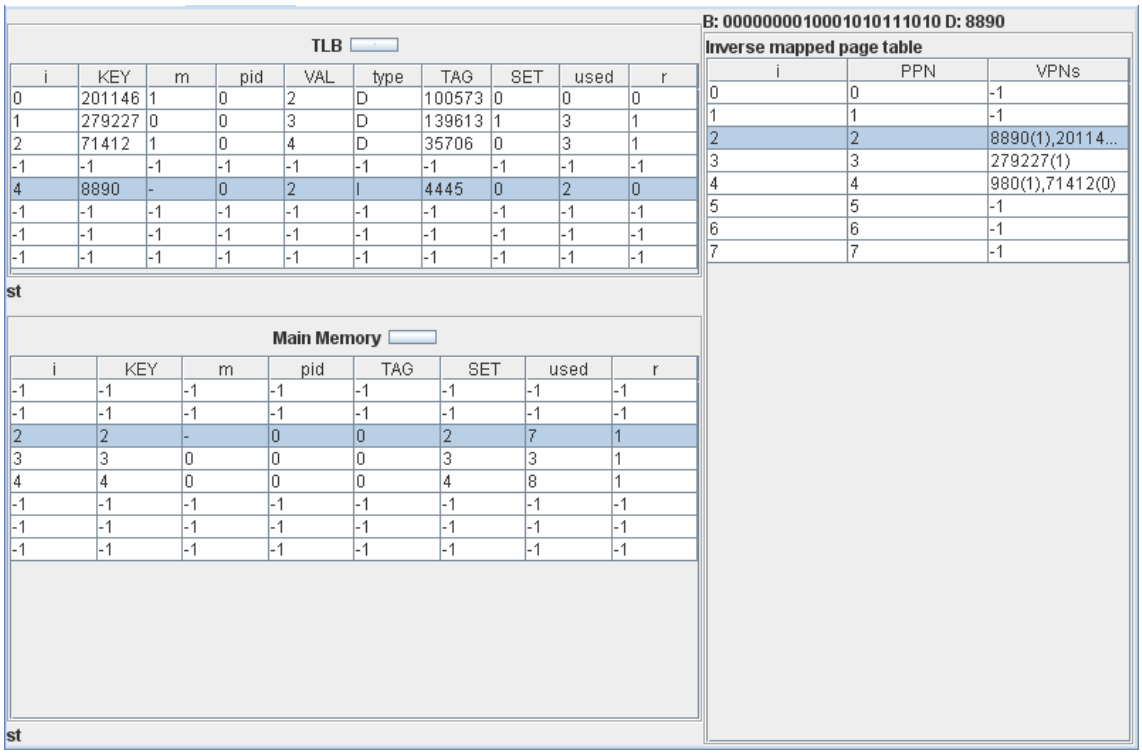
i	PPN	VPNs
0	0	-1
1	1	-1
2	2	8890(0),20114...
3	3	279227(1)
4	4	980(1),71412(0)
5	5	-1
6	6	-1
7	7	-1

i	KEY	m	pid	TAG	SET	used	r
-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1
2	2	1	0	0	2	7	1
3	3	0	0	0	3	3	1
4	4	0	0	0	4	8	1
-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1

En este punto cabe destacar la aparición de dos nuevos datos (se pueden observar en la tabla de dispersión). Además, uno de ellos pertenecerá a un conjunto de dispersión distinto al resto, al igual que será el único que haya ocupado la posición 3 de la memoria principal (también se puede ver esto en las VPNs de la tabla de mapeo) con lo cual se podría afirmar, sin conocer lo que ha sucedido en la traza, que esta posición no ha tenido colisiones.

Durante la ejecución de la traza se ha podido ver como en la TLB, se han dado varias colisiones, pero todas ellas siempre en la posición 0, esto se debe a la implementación de la política de reemplazo, ya que esta se fija en el contador r, coge la primera con menor contador, si ambas tienen el contador a 0, siempre se cogerá la política 0. También se ha podido observar como este contador se ha incrementado a la hora de acceder a la página de la posición 2 (Lo mismo sucede en el contador r de la posición 1).

Finalmente, la instrucción FETCH no encontrará la página en memoria principal, aunque sí en caché, con lo que la llevará a desde disco a memoria y la almacenará. Pero aunque esta vez se trata de una instrucción no producirá colisión, como cabría esperar, ya que si se revisa la configuración de la máquina, nos damos cuenta que la TLB separa datos de instrucciones.

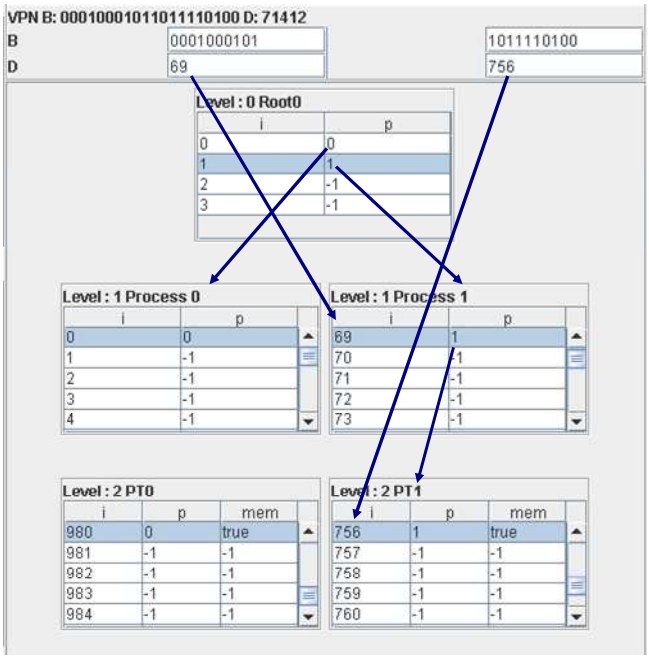


4.3. Traza 3

En esta traza se mostrará un ejemplo de ejecución multiproceso. Concretamente se llevará a cabo una ejecución de dos procesos.

4.3.1 Tablas de páginas para varios procesos

Antes de realizar la simulación de la traza 3 es conveniente explicar la pantalla de tablas de páginas en el caso que haya varios procesos cargados:



Como se ha explicado antes, en este ejemplo se van a cargar dos procesos cada uno respectivamente con dos tablas de páginas de nivel 2.

Para cualquier ejemplo, se va a tener siempre una tabla de páginas de nivel 0 o Root que contendrá los índices a la tabla de página de nivel 1 para los procesos cargados.

En este ejemplo, los 10 bits más significativos de la dirección virtual constituyen un índice dentro de la tabla de páginas seleccionada por el valor p de la tabla Root. El valor asociado a este índice a su vez seleccionará la tabla de páginas de nivel 2 a ser seleccionada.

Una vez seleccionada la tabla de páginas de nivel 2, sólo queda indexar, con los 10 bits menos significativos de la dirección virtual, la página física asociada a la página virtual.

4.3.2 Configuración

El fichero de configuración es el siguiente:

```
<config>

<virtualAddressNBits>32</virtualAddressNBits>
<numberProcessesNBits>2</numberProcessesNBits>
<diskATU>3</diskATU>

<pageTable direct="true">
        <offsetLengths>
                <length>10</length>
                <length>10</length>
        </offsetLengths>
        <searchMethod>topdown</searchMethod>
        <tlbConfig>
                <numberEntriesNBits>2</numberEntriesNBits>
                <evictionPolicy>FIFO</evictionPolicy>
                <dataInstrSeparated>true</dataInstrSeparated>
                <accessTimeUnits>1</accessTimeUnits>
        </tlbConfig>
</pageTable>

<pageAgingConfig>
        <pageAgingIncrease>2</pageAgingIncrease>
        <memRefToRun>5</memRefToRun>
</pageAgingConfig>

<memAllocConfig>
        <minPFF>3</minPFF>
        <maxPFF>6</maxPFF>
        <evNodesToRun>5</evNodesToRun>
```

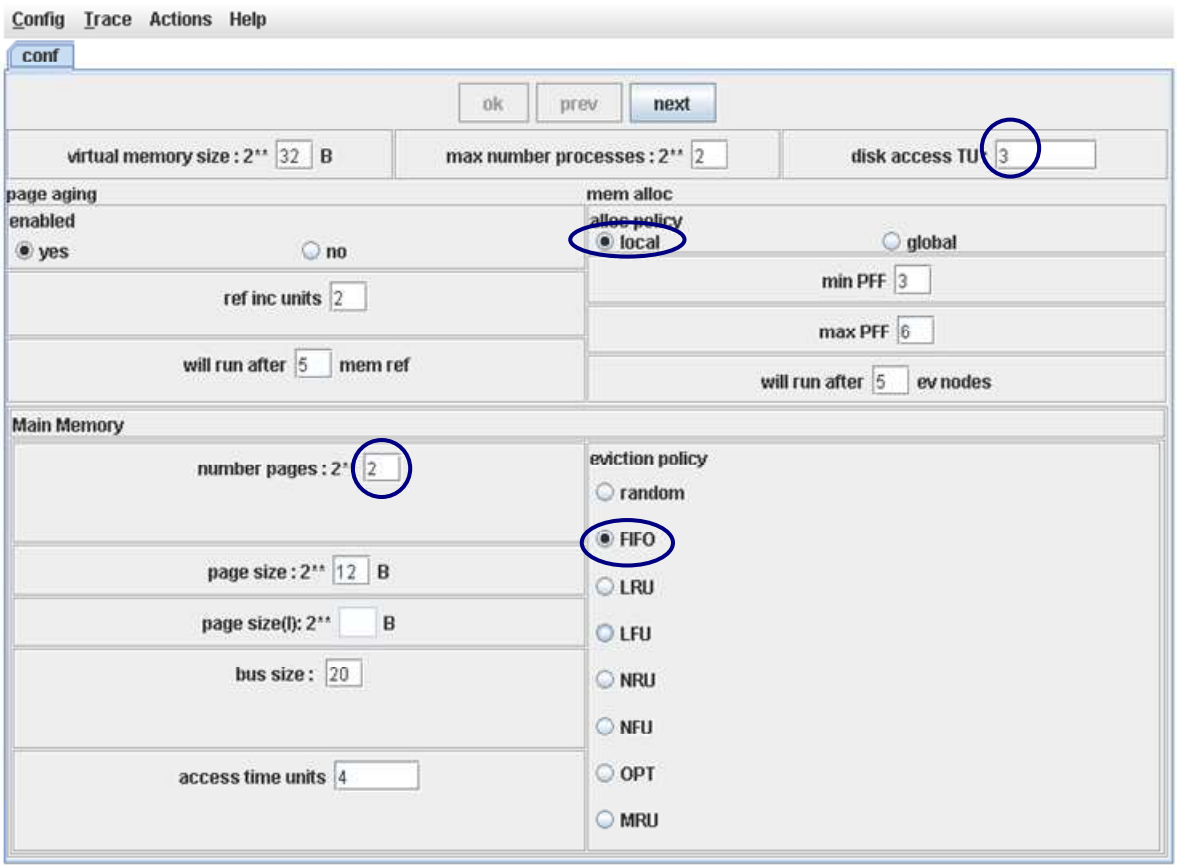
```
</memAllocConfig>

<mainMemoryConfig>
        <numberEntriesNBits>2</numberEntriesNBits>
        <blockSizeNBits>12</blockSizeNBits>
        <evictionPolicy>FIFO</evictionPolicy>
        <busSize>20</busSize>
        <accessTimeUnits>4</accessTimeUnits>
        <numberSetsNBits>0</numberSetsNBits>
        <dataInstrSeparated>false</dataInstrSeparated>
</mainMemoryConfig>

</config>
```

4.3.3 Cargando el fichero de configuración

Una vez se ha cargado el fichero de configuración pasamos a detallar las peculiaridades que tiene este ejemplo. Los valores establecidos para la primera pantalla de configuración serán los siguientes:



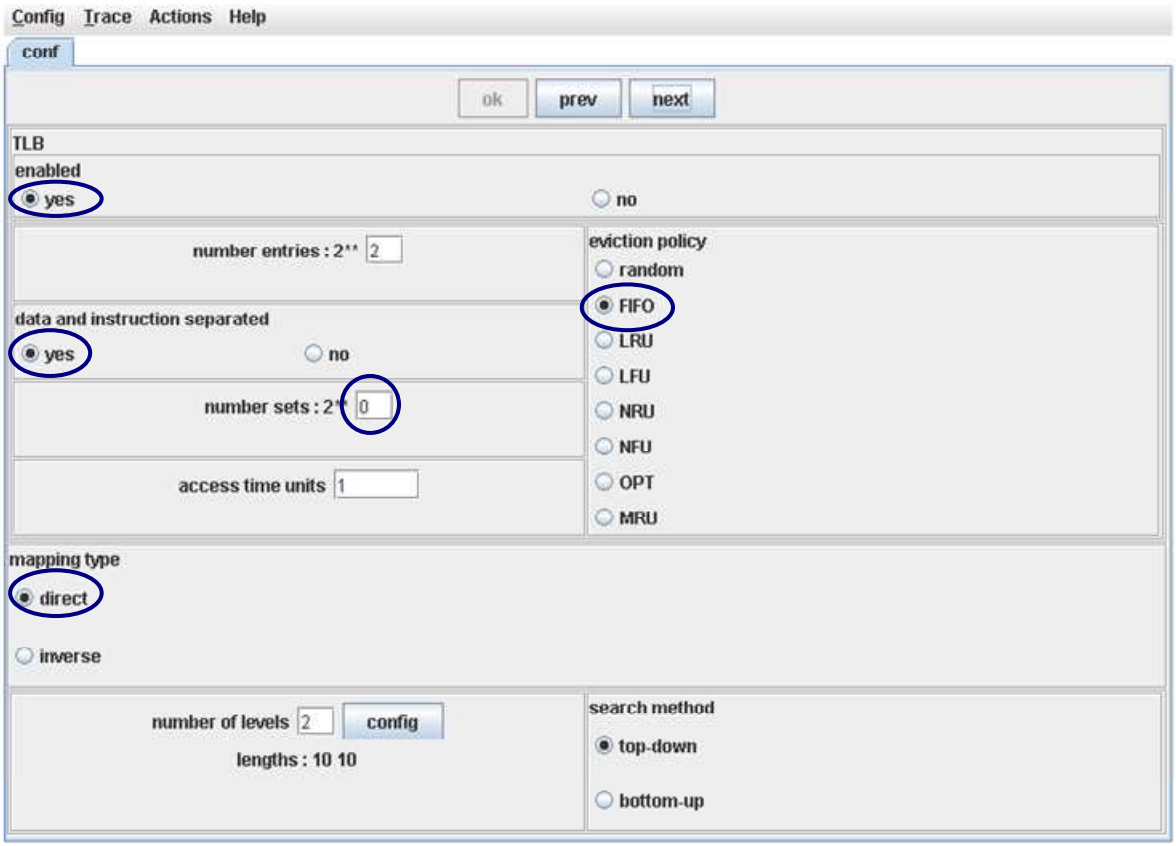
El número de páginas seleccionado (number pages) será pequeño, concretamente de 4 páginas, para que se pueda llenar la memoria con relativa facilidad y haya reemplazos.

La política de reemplazo (eviction policy) que se utilizará será la FIFO

La política de asignación de memora utilizada será local con umbrales máximos y mínimos de frecuencia de fallos de página de 3 y 6.

La penalización de acceso a disco (disk access TU) será de 3. Se ha establecido un valor pequeño para que un proceso pueda abandonar la cola de espera y pasar a la cola de ejecución una vez se le ha consumido la penalización de acceso a disco.

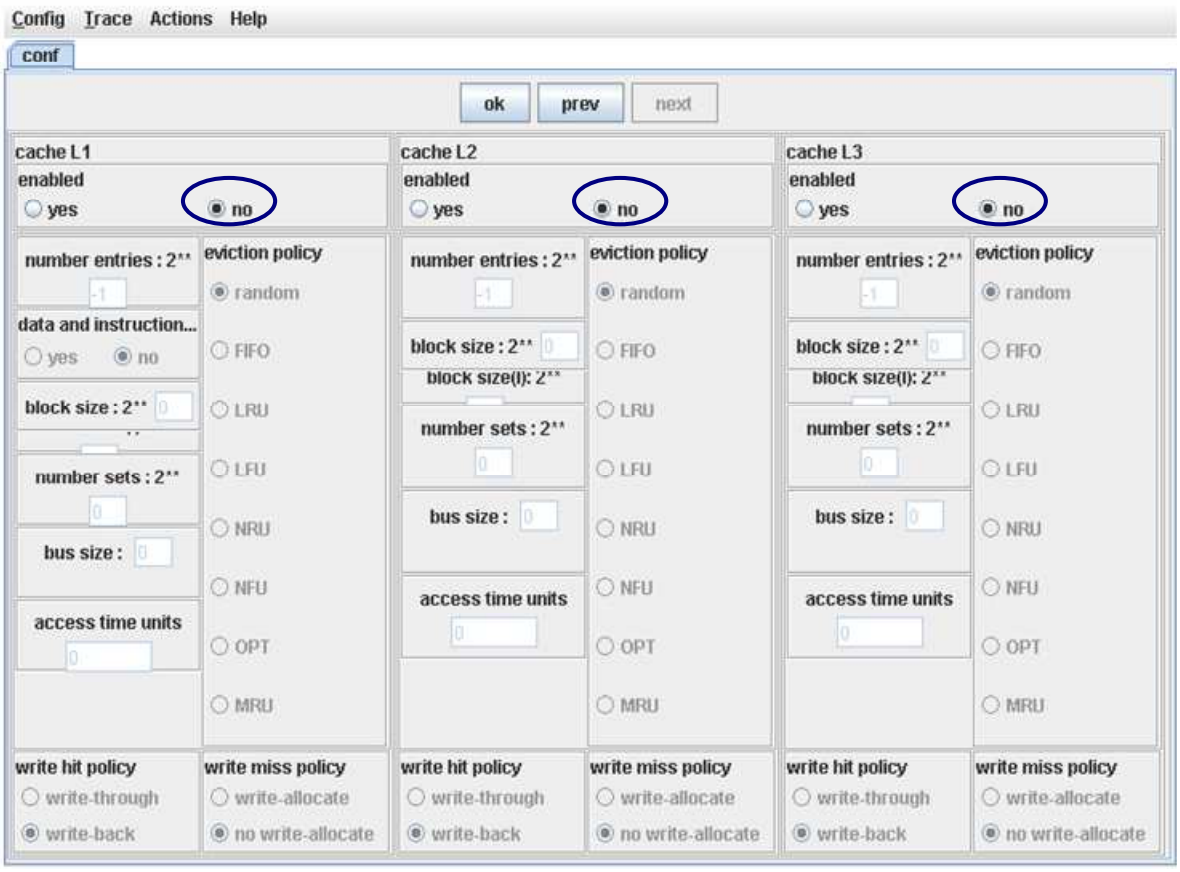
Para la segunda pantalla de configuración, se han establecido los siguientes valores para los campos:



Se ha habilitado una TLB de mapeado directo (2^0 sets) con datos e instrucciones separados. La política de reemplazo de la TLB, al igual que en la memoria, será FIFO.

El tipo de mapeado en memoria será directo.

Para finalizar, la última pantalla de configuración es la siguiente:



Se puede observar como se han deshabilitado todas las cachés. La dificultad de seguir una ejecución multiproceso es lo suficientemente grande que se ha visto conveniente simplificar el problema eliminando todas las cachés.

4.3.4 Traza

Las instrucciones de los procesos a ejecutar serán las siguientes:

Proceso 1:

```
1       003d49b0 MEMREAD   9952
2       116f49a0 MEMWRITE  50
3       116f49a0 MEMREAD   50
4       003d49b0 MEMWRITE  9952
5       22ba3c0 FETCH      509
```

Proceso 2:

```
1       116f49a0 MEMWRITE  50
2       003d49b0 MEMREAD   9952
3       003d49b0 MEMWRITE  9952
4       116f49a0 MEMREAD   50
```

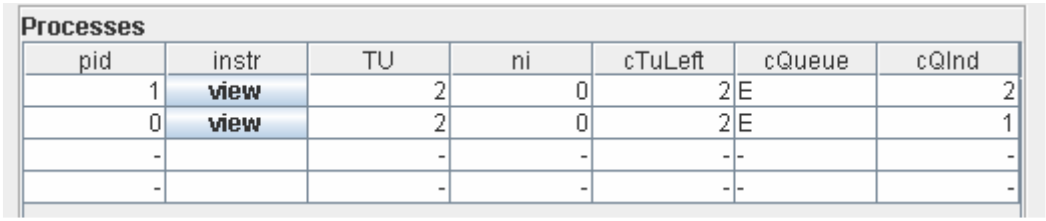
```
5       003d49b0 MEMREAD   5097
6       003d49b0 MEMREAD   5097
7       003d49b0 MEMREAD   5097
        8       003d49b0 MEMREAD   5097
9       003d49b0 MEMREAD   509
```

NOTA: La primera columna no forma parte del fichero .trd. Pretende simplemente identificar a cada instrucción

4.1.4 Ejecución

Cargamos los dos procesos: Trace → Load local trace file y le asignamos a ambos un TUnit de 2 unidades.

En la pestaña proc pude observarse lo siguiente:

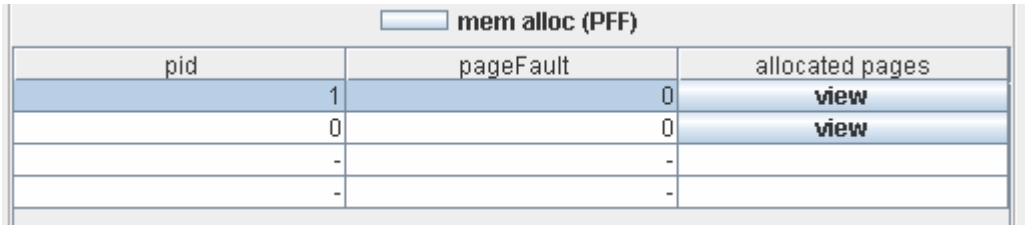


Processes

pid	instr	TU	ni	cTuLeft	cQueue	cQInd
1	**view**	2	0	2	E	2
0	**view**	2	0	2	E	1
-		-	-	-	-	-
-		-	-	-	-	-

En la imagen se puede observar que se han cargado los dos procesos, ambos con TU de dos unidades. Los dos procesos se encuentran en cola de ejecución (E) siendo el proceso con pid 0 (proceso 0) el primero que va a ejecutarse ya que se encuentra en la primera posición de la cola de ejecución. El tiempo restante de ambos procesos en esa posición de la cola (marcado por cTuLeft) es también de dos.

Siguiendo en la pestaña proc, podemos observar la asignación de memoria para cada proceso:



mem alloc (PFF)

pid	pageFault	allocated pages
1	0	**view**
0	0	**view**
-	-	
-	-	

En allocated pages, pinchamos en view para el proceso 0:

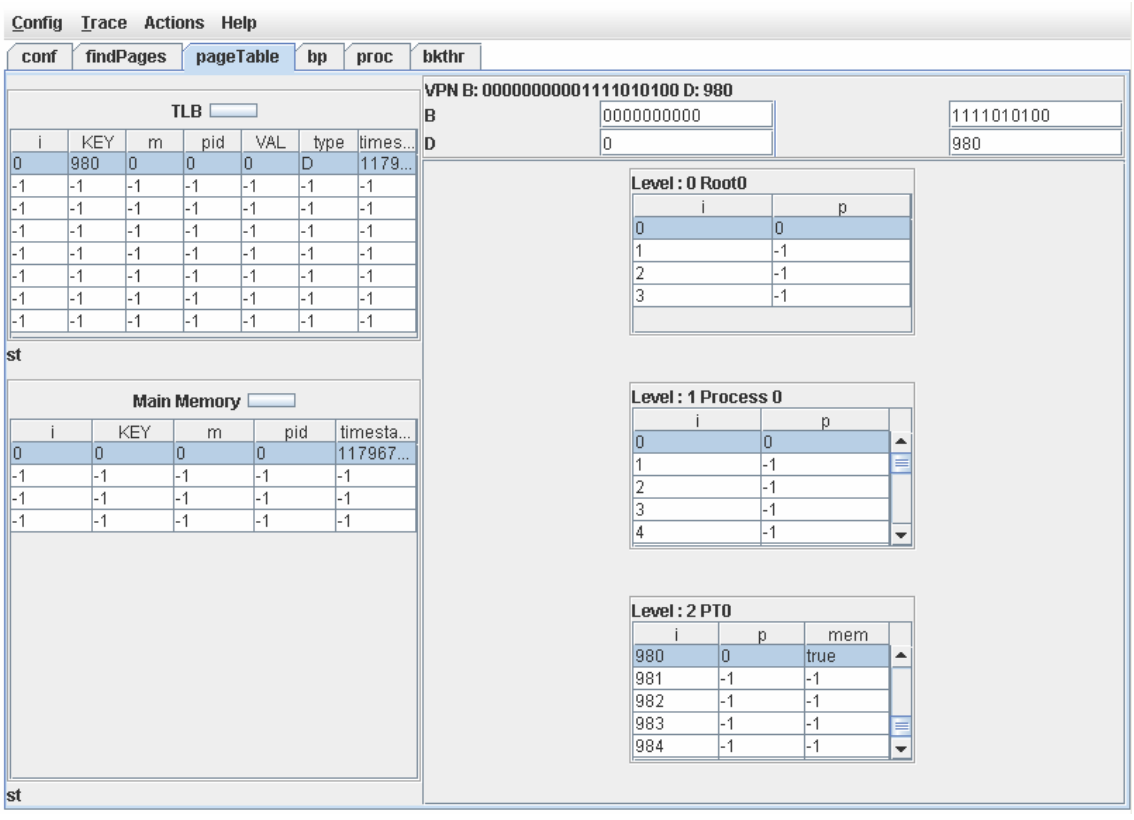
Se observa que ese proceso tiene 2 páginas asignadas (recordemos que son 4 las páginas totales en memoria). Como puede observarse en la imagen, de las 4 páginas, al proceso 0 se le han asignado las páginas 0 y 3.

Lo mismo para el proceso 1:

Observamos que al proceso 1 se le han asignado las páginas 1 y 2 de las 4 páginas totales.

Le damos a step y ejecutamos la primera instrucción del proceso 0:
003d49b0 MEMREAD 9952

Si visualizamos la pestaña pageTable, podemos observar lo siguiente:



Se ha generado la dirección virtual 003d49b0 cuyos 20 bits más significativos servirán para obtener la página real asociada a esa dirección virtual como se explicó en el apartado de Tablas de Páginas Para Varios Procesos.

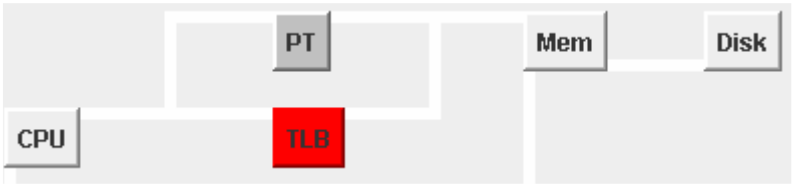
La dirección virtual se divide de la siguiente forma:

003d49b0 = 0000000000 1111010100 100110110000
0 *980* *desplazamiento*

En la pantalla de ejecución se puede ver que el desplazamiento no interesa en este momento y sólo se necesitan los 20 bits más significativos para obtener la página física.

Como la dirección virtual 003d49b0 no ha sido referenciada con anterioridad, se ha producido un fallo de página, se ha traído la página de disco y se ha añadido tanto a la TLB como a la memoria principal.

Observemos en la pestaña bp el fallo producido en la TLB:



Hay que resaltar que este MEMREAD todavía no se ha terminado de ejecutar. Se ha producido un fallo de página, se ha traído la página de disco, pero no se ha trasladado el dato a la CPU. Cada vez que se produzca un fallo de página, tanto los MEMREAD como los MEMWRITE se harán en dos steps: el primer step trae de disco a memoria la página, y el segundo step lleva el dato a la CPU.

Pasamos de nuevo a la pestaña proc y vemos lo que va a suceder durante la segunda instrucción:

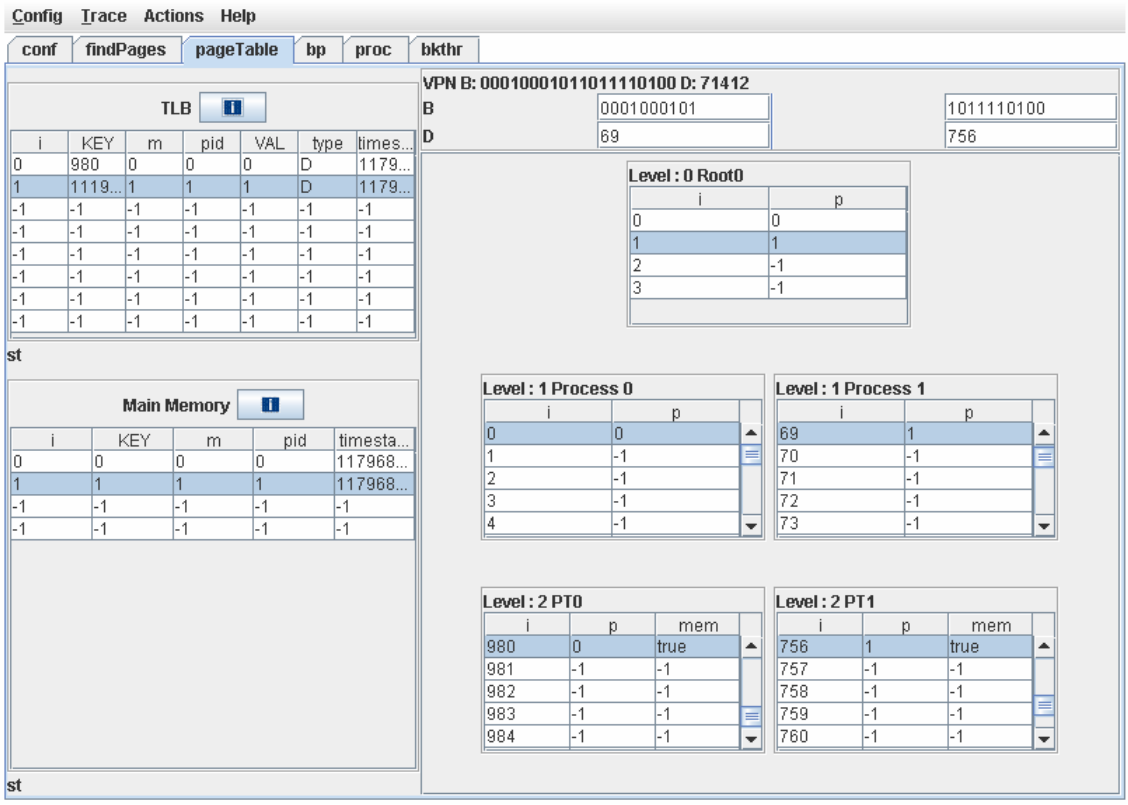
Processes

pid	instr	TU	ni	cTuLeft	cQueue	cQInd
1	**view**	2	0	2	E	1
0	**view**	2	0	3	W	1
-		-	-	-	-	-
-		-	-	-	-	-

Como se explicó en el apartado de colas de procesos, al producirse un fallo de página va a ocurrir un cambio de contexto en la CPU: ahora va a ser el proceso 1 el que va a ejecutarse en la siguiente instrucción y el proceso 0 va a pasar a la cola de espera con la penalización del tiempo de acceso a disco debido al fallo de página. Recordemos que el valor asignado a disk access TU era de 3.

Le damos a step y pasa a ejecutarse la primera instrucción del proceso 1:
116f49a0 MEMWRITE 50

Observamos la pestaña pageTable:



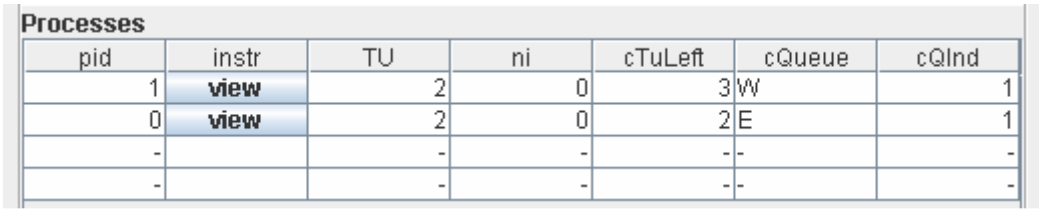
En la imagen anterior se observa que la dirección virtual se divide de la siguiente manera:

116f49a0 = 0001000101 1011110100 100110100000

Esta dirección virtual no ha sido referenciada con anterioridad, de modo que se ha producido un fallo de página, se ha traído la página de disco y se ha añadido tanto a la TLB como a la memoria principal.

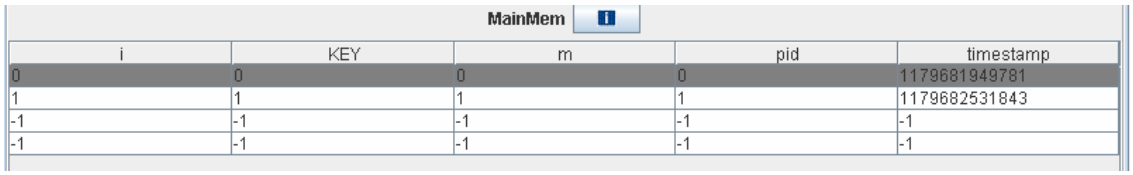
Vemos que se le ha asignado la página 1 a esta dirección virtual. No se le podría haber asignado ni la página 0, ni la página 3 ya que las únicas páginas asociadas al proceso 1 son las páginas 1 y 2.

Como se ha producido un fallo de página, se va a producir un cambio de contexto en la CPU, se va a ejecutar en el siguiente paso el proceso 0 y el proceso 1 va a pasar a la cola de espera con la penalización de acceso a disco:



Como se comentó anteriormente, la primera instrucción del proceso 0 no se terminó de ejecutar: falta llevar el dato a la CPU.

Le damos a step y observamos que el dato se encuentra en memoria (por que ya se trajo de disco en el paso anterior).

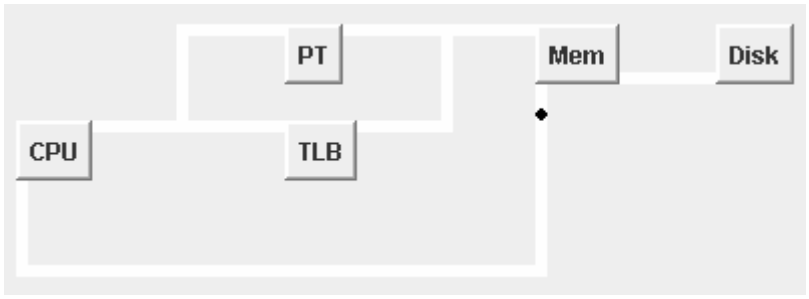


MainMem

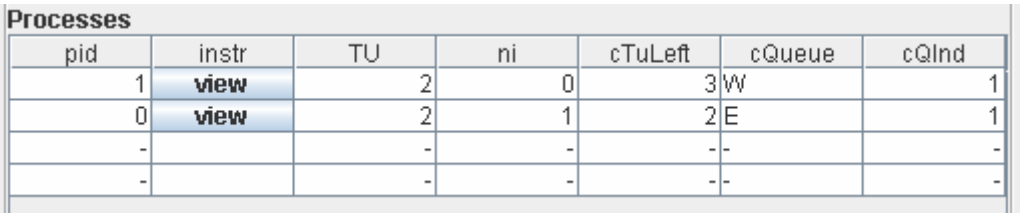
i	KEY	m	pid	timestamp
0	0	0	0	1179681949781
1	1	1	1	1179682531843
-1	-1	-1	-1	-1
-1	-1	-1	-1	-1

Como es un MEMREAD y el dato se encuentra en memoria, se pone la página en gris.

Si visualizamos bp observamos que efectivamente el dato viaja de memoria a la CPU:



Como en este paso no se ha producido un fallo de página, no va a ocurrir un cambio de contexto en la CPU y va a seguir siendo el proceso 0 el “dueño” del procesador. Si nos vamos a la pestaña proc, vemos que efectivamente eso es lo que va a ocurrir:



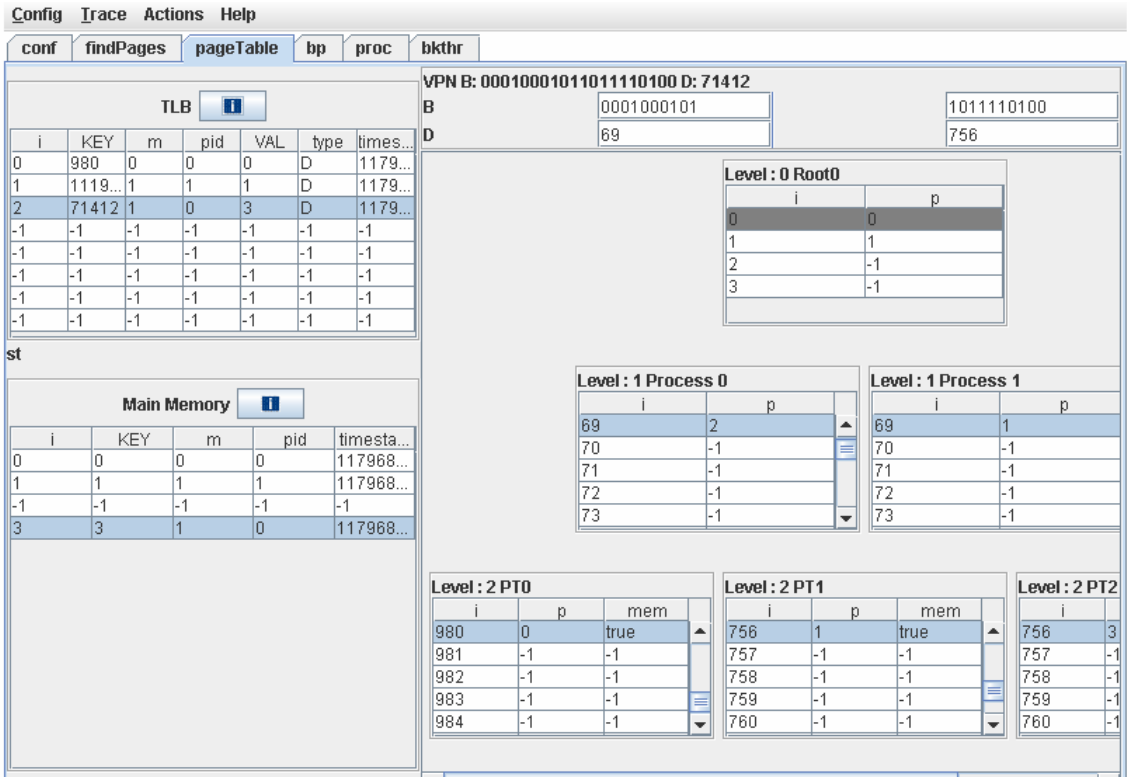
Processes

pid	instr	TU	ni	cTuLeft	cQueue	cQInd
1	view	2	0	3	W	1
0	view	2	1	2	E	1
-		-	-	-	-	-
-		-	-	-	-	-

Se ha terminado de ejecutar la primera instrucción del proceso 0. Si le damos a step pasamos a ejecutar la instrucción 2:

116f49a0 MEMWRITE 50

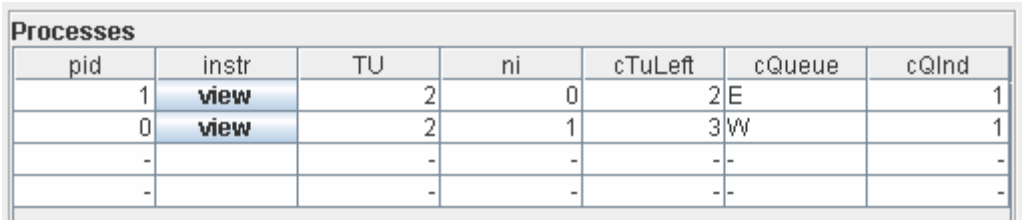
La pestaña pageTable queda como sigue:



La dirección 116f49a0 no se ha referenciado anteriormente para ese proceso, de manera que se ha producido un fallo de página, se ha traído la página de disco y se ha añadido tanto a la TLB como a la memoria principal.

Observando la pestaña pageTable, vemos que las páginas 0 y 3 asociadas al proceso 0 están ya ocupadas.

Como se ha producido un fallo de página, se va a producir un cambio de contexto en la CPU, se va a ejecutar en el siguiente paso el proceso 1 y el proceso 0 va a pasar a la cola de espera con la penalización de acceso a disco:



El MEMWRITE de la primera instrucción no se ha terminado de ejecutar. Si le damos a step, se encuentra la página en la que se desea escribir:

MainMem

i	KEY	m	pid	timestamp
0	0	0	0	1179681949781
1	1	1	1	1179682531843
-1	-1	-1	-1	-1
3	3	1	0	1179684290890

Como es un MEMWRITE y se ha encontrado la página en la que escribir, se colorea la correspondiente página.

En este paso no se ha producido ningún fallo de modo que el proceso 1 va a seguir con la ejecución:

Processes

pid	instr	TU	ni	cTuLeft	cQueue	cQInd
1	**view**	2	1	2	E	1
0	**view**	2	1	3	W	1
-		-	-	-	-	-
-		-	-	-	-	-

Si presionamos step pasa a ejecutarse la segunda instrucción del proceso 1:
003d49b0 MEMREAD 9952

Como la dirección 003d49b0 no ha sido referenciada se ha producido un fallo de página, se ha traído la página de disco y se ha añadido tanto a la TLB como a la memoria principal.

Ahora podemos observar en la pestaña findPages que la memoria principal está llena. Eso quiere decir que la próxima vez que se traiga una página de disco, se va a producir un reemplazo:

MainMem

i	KEY	m	pid	timestamp
0	0	0	0	1179681949781
1	1	1	1	1179682531843
2	2	0	1	1179685489968
3	3	1	0	1179684290890

Se ha producido un fallo de página y por lo tanto va a ocurrir un cambio de contexto. Va a pasar a ejecutarse, de nuevo, el proceso 0 y el proceso 1 se va a colocar el último (y el primero ya que está sólo) en la cola de espera.

Si presionamos step, el MEMWRITE de la segunda instrucción del proceso 0 se va a terminar de ejecutar:

MainMem

i	KEY	m	pid	timestamp
0	0	0	0	1179910142979
1	1	1	1	1179910148980
2	2	0	1	1179910173140
3	3	1	0	1179910159997

Como es un MEMWRITE y la página a escribir ya se encuentra en memoria, se ha coloreado la página en verde oscuro.

No se ha producido un fallo de página, de modo que el proceso 0 sigue siendo el "dueño" de la CPU, por lo que ejecuta la tercera instrucción:

116f49a0 MEMREAD 50

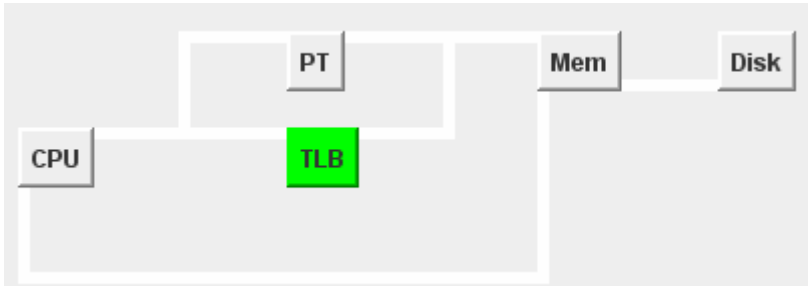
Observamos que se va a realizar una lectura de memoria de una dirección virtual que ya ha sido referenciada, concretamente en la instrucción anterior. Esto quiere decir que la página física ha sido cargada en memoria y también está presente en la TLB.

Se va a leer una página que ha sido escrita en la instrucción anterior (página 3), como es un MEMREAD y el dato está presente en memoria, se colorea la página leída en gris:

MainMem

i	KEY	m	pid	timestamp
0	0	0	0	1179910142979
1	1	1	1	1179910148980
2	2	0	1	1179910173140
3	3	1	0	1179910159997

Vemos cómo el dato esta vez sí se encuentra en la TLB (se ha producido un hit):

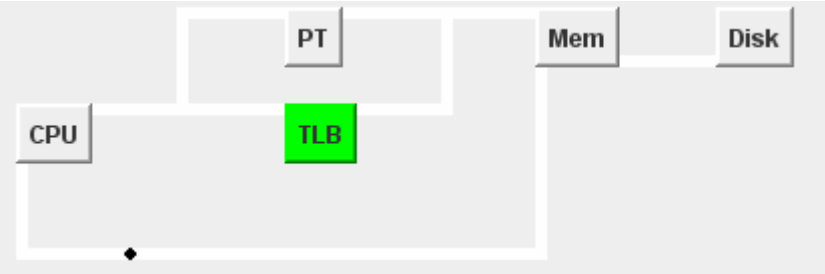


Como la página está presente en memoria, se lleva el dato de memoria a la CPU:

Si presionamos step, en la siguiente instrucción a ejecutar también se va a producir un éxito (la página referenciada está en memoria principal):

003d49b0 MEMWRITE 9952

La dirección virtual 003d49b0 ha sido referenciada en la primera instrucción del programa:



Vemos en la imagen anterior que el dato se encuentra en la TLB y escribimos en memoria.

Siguen sin producirse fallos de página y por otro lado el TUnit del proceso 0 tampoco se ha agotado, de modo que sigue éste ejecutándose.

Dándole a step pasa a ejecutarse la última instrucción del proceso 0:
22ba3c0 FETCH 5097

La dirección virtual no se ha referenciado anteriormente de modo que se va a producir un fallo de página. Va a ser necesario traerse la página de disco y como la memoria está llena va a producirse un reemplazo.

¿Qué página debe ser reemplazada? En principio, o bien la página 0 o la página 3. Como la política de reemplazo que se está utilizando es la política FIFO, se va a eliminar la primera página cargada en memoria o lo que es lo mismo la que tenga en el campo timestamp el valor más alto.

La página a reemplazar se colorea en rojo indicando que será reemplazada por la nueva.

Main Memory

i	KEY	m	pid	timesta...
0	0	1	0	117991...
1	1	1	1	117991...
2	2	0	1	117991...
3	3	1	0	117991...

Se coloca la nueva página actualizando su valor para el timestamp.

Para terminar la ejecución del proceso 0, sólo queda eliminar las páginas asociadas a él: páginas 0 y 3, pero como se ha producido un fallo de página se va a volver a la ejecución el proceso 1. Observando la pestaña proc observamos este hecho:

Processes

pid	instr	TU	ni	cTuLeft	cQueue	cQInd
0	**view**	2	4	3	W	1
1	**view**	2	1	2	E	1
-		-	-	-	-	-
-		-	-	-	-	-

Hasta ahora sólo se han visto cambios de contexto debidos a fallos de página. Ahora se va a producir una alternancia en el dominio de la CPU pero debido a que el TUnit del proceso 1 expira.

Presionando step, termina de ejecutarse la segunda instrucción del proceso 1, ya que éste provocó un fallo de página. Como se trata de un MEMREAD y estamos accediendo para leer una página presente en memoria, se colorea en gris la página a ser leída:

MainMem

i	KEY	m	pid	timestamp
0	0	-	0	1179912800008
1	1	1	1	1179910148980
2	2	0	1	1179910173140
3	3	1	0	1179910159997

Si presionamos step, la siguiente instrucción a ejecutar va a referenciar a la misma instrucción ejecutada anteriormente:
003d49b0 MEMWRITE 9952

Como se trata de un MEMWRITE y la página ha sido cargada en memoria en la instrucción anterior, se colorea en verde la página 2:

MainMem

i	KEY	m	pid	timestamp
0	0	-	0	1179912800008
1	1	1	1	1179910148980
2	2	1	1	1179910173140
3	3	1	0	1179910159997

Observamos que el tiempo restante en las correspondientes colas de ejecución y espera (cTuLeft) se ha decrementado en una unidad, pasando de 3 a 2 para el proceso 0 y de 2 a 1 para el proceso 1:

Processes

pid	instr	TU	ni	cTuLeft	cQueue	cQInd
0	**view**	2	4	2	W	1
1	**view**	2	3	1	E	1
-		-	-	-	-	-
-		-	-	-	-	-

Presionando step, se pasa a la siguiente instrucción:
116f49a0 MEMREAD 50

El campo de dirección virtual 116f49a0 de esta instrucción ya ha sido referenciado en instrucciones anteriores por lo que el resultado es análogo a casos anteriores. Al ser un MEMWRITE y estar presente la página, se colorea ésta a gris.

Si observamos la pestaña proc:

Processes

pid	instr	TU	ni	cTuLeft	cQueue	cQInd
0	**view**	2	4	1	W	1
1	**view**	2	4	2	E	1
-		-	-	-	-	-
-		-	-	-	-	-

Como el cTuLeft del proceso 1 (cola E) llegó a cero, este proceso pasó a la última posición de la cola E reiniciando su cTuLeft a 2 unidades. Como sólo hay un proceso en cola de ejecución que es él mismo, el dominio de la CPU lo sigue teniendo el proceso 1 ya que el último y el primero de dicha cola son el mismo proceso. Esto quiere decir que no se va a producir un cambio de contexto hasta que el cTuLeft del proceso 0 que está en cola de espera (W) llegue a 0.

Las siguientes instrucciones son todas iguales:
003d49b0 MEMREAD 5097

Tampoco van a producir fallo de página ya que se utiliza la dirección virtual 003d49b0 utilizada con anterioridad. El aspecto interesante de estas instrucciones es ver cómo el proceso 1 es eliminado de la cola de ejecución sin producirse fallos de página.

Processes

pid	instr	TU	ni	cTuLeft	cQueue	cQInd
0	**view**	2	4	0	W	1
1	**view**	2	5	1	E	1
-		-	-	-	-	-
-		-	-	-	-	-

En la imagen anterior vemos como al proceso 0 ya se le ha agotado su tiempo de penalización por acceso a disco, o lo que es lo mismo, su cTuLeft ha llegado a 0.

Como se comentó en el apartado de colas de procesos, el proceso 0 en la siguiente instrucción va a abandonar la cola de espera (W) y se va a colocar el último de la cola de ejecución.

Si le damos a step vemos que efectivamente ocurre eso, si nos damos cuenta del bug nombrado en el apartado de colas de procesos, es decir, no se ha actualizado el campo cQueue:

Processes

pid	instr	TU	ni	cTuLeft	cQueue	cQInd
0	**view**	2	4	2	W	2
1	**view**	2	6	2	E	1
-		-	-	-	-	-
-		-	-	-	-	-

Aunque la pestaña proc diga lo contrario, ambos procesos están en cola de ejecución.

Continuando la ejecución, se presionamos 2 veces step: ejecutando hasta la instrucción 7 del proceso 1 (ésta incluida), se observa como el proceso 0 toma el control de la CPU ya que el cTuLeft del proceso 1 llega a cero.

Finalmente se termina de ejecutar el proceso 0, y se produce la eliminación de las páginas 1 y 3 asociadas a dicho proceso (amarillo claro):

MainMem

i	KEY	m	pid	timestamp
-1	-1	-1	-1	-1
1	1	1	1	1179910148980
2	2	1	1	1179910173140
-1	-1	-1	-1	-1

Como el proceso 0 se ha terminado de ejecutar, pasa a terminar de ejecutarse el proceso 1. Si le damos a step termina de ejecutarse ese proceso y se nos muestran las estadísticas finales: