Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencias de la Computación

IIC2333 – Sistemas Operativos y Redes

Ayudantía complementaria – Paginación – Solución propuesta

Profesor: Cristian Ruz

Ayudante: Germán Leandro Contreras Sagredo (glcontreras@uc.cl)

Nota al lector

El título dice "solución propuesta" por una razón bien sencilla: Estos ejercicios pueden tener más de un desarrollo correcto. Lo que se pretende hacer aquí es mostrar un camino a la solución, sin excluir la posibilidad de rutas alternativas igual de correctas.

Preguntas

1. **(Midterm - II/2018)** Considere un sistema de direcciones virtuales de 40 bit, con soporte de hasta 16GB de memoria física y páginas de 4KB. Cada dirección de memoria referencia 1 Byte.

   a. ¿Cuánto es la máxima cantidad de memoria que puede direccionar cada proceso?
   Con 40 bits podemos tener un total de 2^{40} direcciones diferentes. Luego, si cada dirección de memoria referencia 1 Byte, entonces se tiene una cantidad de memoria direccionable igual a 2^{40}B = 1TB.

   b. Si el tamaño de cada entrada en la tabla de páginas (PTE) está alineado a una cantidad de Byte que sea potencia de 2, ¿de qué tamaño, en Byte, es la tabla de páginas? ¿Cuántos bit quedan disponibles para *metadata*?
   Para responder a esta pregunta, es necesario obtener los siguientes datos:

   - **Bits de *offset*:** El *offset* estará dado por el tamaño de página, dado que cada dirección de *offset* nos entrega una ubicación dentro de una página. Al tener un tamaño de páginas igual a 4KB, tenemos entonces 2^2KB = 2^{12}B. Es decir, se necesitan 12 bits de *offset* para poder direccionar el espacio completo de una página.
   - **Bits de página:** Para obtener este valor, es necesario desglosar las direcciones virtuales en dos fragmentos: `n_página|offset`. Dado que sabemos que existen 12 bits de *offset* dentro de la dirección virtual: $b_p + b_{off} = 40 \rightarrow b_p = 28$. Es decir, necesitamos 28 bits para poder direccionar a todas las páginas existentes.

- **Bits de direccionamiento físico:** Para este valor, basta con hacer uso del tamaño de la memoria física y el hecho de que **cada dirección de memoria referencia a 1 Byte**. Al tener un 16GB de memoria física, podemos ver que 2^4GB = $2^3$4B, es decir, tenemos un total de $2^3$4 direcciones físicas, lo que se traduce en 34 bits de direccionamiento físico.
- **Bits de marco:** Para obtener este valor, es necesario desglosar las direcciones físicas en dos fragmentos: `n_marco|offset`. Dado que el *offset* es igual al de las páginas[1]: $b_m + b_{off} = 34 \rightarrow b_m = 22$. Es decir, necesitamos 22 bits para poder direccionar a todos los marcos físicos existentes.
- **Número de entradas (PTE):** Este número será igual al **número de páginas existente por proceso**. Por lo tanto, existen 2^{28} entradas en total.
- **Tamaño de PTE:** Este valor es igual al número de marco físico al que la página correspondiente está asignada, sumado a los bits de *metadata* que podemos usar. Dado que el número de marco físico hace uso de 22 bits y nuestra entrada debe tener un tamaño que sea una potencia de 2, debemos rellenar con 10 bits adicionales, los que serán nuestra *metadata*.

Finalmente, calculamos el tamaño de la tabla de páginas como sigue:

$$T_p = \#PTE \times \texttt{sizeof}(PTE) = 2^{28} \times 32\text{b} = 2^{28} \times 4\text{B} = 2^{30}\text{B} = 1\text{GB}$$

c. Para reducir la cantidad de memoria necesaria por cada proceso se propone aumentar el tamaño de las páginas. ¿Hasta cuánto debe crecer el tamaño de la página para que la tabla de páginas quepa completamente en una página? Puede ignorar los bits de *metadata*, pero debe considerar el alineamiento a una potencia de 2 en los Byte de la PTE.

De partida, sabemos que nuestra tabla de páginas inicialmente no va a cambiar su tamaño ignorando los bits de *metadata*. Si bien estos ya no se usan, el *padding* para tener un tamaño de potencia de 2 sigue siendo requisito. Ahora, al ver que la tabla ocupa 1GB en total, mientras que el tamaño de página es de 4KB, sabemos que es necesario hacer una reducción para poder cumplir lo pedido. Para poder obtener el valor de forma ordenada, es necesario notar lo siguiente: si duplico el tamaño de mi página, estaré usando un bit adicional de *offset* y un bit menos de número de página[2], lo que se replicará en el caso de las direcciones físicas -disminuyendo en una unidad la cantidad de bits de los marcos físicos-. Por lo tanto, haremos este cálculo secuencialmente hasta cumplir el requisito:

- **Tamaño de página igual a 8KB:** $2^{27} \times (21 + 11)\text{b} = 2^{27} \times 4\text{B} = 512\text{MB}$
- **Tamaño de página igual a 16KB:** $2^{26} \times (20 + 12)\text{b} = 2^{26} \times 4\text{B} = 256\text{MB}$
- **Tamaño de página igual a 32KB:** $2^{25} \times (19 + 13)\text{b} = 2^{25} \times 4\text{B} = 128\text{MB}$
- **Tamaño de página igual a 64KB:** $2^{24} \times (18 + 14)\text{b} = 2^{24} \times 4\text{B} = 64\text{MB}$
- **Tamaño de página igual a 128KB:** $2^{23} \times (17 + 15)\text{b} = 2^{23} \times 4\text{B} = 32\text{MB}$
- **Tamaño de página igual a 256KB[3]:** $2^{22} \times (16)\text{b} = 2^{22} \times 2\text{B} = 8\text{MB}$
- **Tamaño de página igual a 512KB:** $2^{21} \times (15 + 1)\text{b} = 2^{21} \times 2\text{B} = 4\text{MB}$
- **Tamaño de página igual a 1MB:** $2^{20} \times (14 + 2)\text{b} = 2^{20} \times 2\text{B} = 2\text{MB}$

[1] Siempre debe ser consistente el tamaño de página con el tamaño de marco físico.
[2] Recordar que el número de bits de direccionamiento virtual no cambia.
[3] Notar que aquí no es necesario añadir *padding*, dado que el número de bits de marco físico ya es una potencia de 2.

- **Tamaño de página igual a 2MB:** $2^{19} \times (13+3)\text{b} = 2^{19} \times 2\text{B} = 1\text{MB}$

Vemos entonces que, en este esquema, es necesario tener páginas de 2MB para poder almacenar en una de estas la tabla de páginas de un proceso en su totalidad.

2. **(I2 - I/2018)** Considere un sistema con páginas de 16KB, direcciones virtuales de 48 bit, direcciones físicas de 30 bit.

   a. Indique la cantidad máxima, en Byte, de memoria virtual que puede direccionar cada proceso; y la cantidad máxima, en Byte, de memoria física que puede direccionar el sistema.

   Con 48 bits podemos tener un total de 2^{48} direcciones virtuales diferentes. Luego, si cada dirección de memoria referencia 1 Byte, entonces se tiene una cantidad de memoria virtual direccionable igual a $2^{48}\text{B} = 2^{8}\text{TB} = 256\text{TB}$.

   De forma similar, con 30 bits podemos tener un total de 2^{30} direcciones físicas diferentes. Dado que cada dirección de memoria también referencia 1 Byte, entonces se tiene una cantidad de memoria física direccionable igual a $2^{30}\text{B} = 1\text{GB}$.

   b. Calcule el tamaño, en Byte, de una tabla de páginas de 1 nivel para este sistema. Considere que se utilizan 4 bit para *metadata*: *valid*, *reference*, *dirty* y *RW*. ¿Cabe la tabla de páginas en la memoria física?

   Sabemos que el tamaño de la tabla de páginas está dado por $T_p = \#PTE \times \texttt{sizeof}(PTE)$. Entonces, llevaremos a cabo un análisis similar al del problema anterior.

   - **Bits de *offset*:** El *offset* estará dado por el tamaño de página, dado que cada dirección de *offset* nos entrega una ubicación dentro de una página. Al tener un tamaño de páginas igual a 16KB, tenemos entonces $2^{4}\text{KB} = 2^{14}\text{B}$. Es decir, se necesitan 14 bits de *offset* para poder direccionar el espacio completo de una página.
   - **Bits de página:** Para obtener este valor, es necesario desglosar las direcciones virtuales en dos fragmentos: `n_página|offset`. Dado que sabemos que existen 14 bits de *offset* dentro de la dirección virtual: $b_p + b_{off} = 48 \rightarrow b_p = 34$. Es decir, necesitamos 34 bits para poder direccionar a todas las páginas existentes.
   - **Bits de marco:** Para obtener este valor, es necesario desglosar las direcciones físicas en dos fragmentos: `n_marco|offset`. Dado que el *offset* es igual al de las páginas: $b_m + b_{off} = 30 \rightarrow b_m = 16$. Es decir, necesitamos 16 bits para poder direccionar a todos los marcos físicos existentes.
   - **Número de entradas (PTE):** Este número será igual al **número de páginas existente por proceso**. Por lo tanto, existen 2^{34} entradas en total.
   - **Tamaño de PTE:** Este valor es igual al número de marco físico al que la página correspondiente está asignada, sumado a los bits de *metadata* que podemos usar. Dado que el número de marco físico hace uso de 16 bits y que debemos incluir 4 bits de *metadata*, entonces cada PTE tendrá un ancho de 20 bits.

   Con la información necesaria, procedemos a aplicar la fórmula[4]:

$$T_p = \#PTE \times \texttt{sizeof}(PTE) = 2^{34} \times 20\text{b} = \frac{2^{34}}{2^{3}} \times 20\text{B} = 2^{31} \times 20\text{B} = 2^{32} \times 10\text{B} = 40\text{GB}$$

[4]**IMPORTANTE:** Notar que en el ejercicio anterior se hizo la transformación de bits a Byte de forma directa dado que es sencillo ver que 32b = 4B. No obstante, hay casos en los que la respuesta no es tan directa. En estos casos, se debe aplicar la siguiente transformación: $x\text{b} \times \frac{1\text{B}}{2^3\text{b}} = \frac{x}{2^3}\text{B}$.

Finalmente, es directo ver que una tabla de páginas de 40GB **no cabe** en una página de 16KB.

c. Proponga un esquema de paginación multinivel con la mínima cantidad de niveles necesarios en el cual, para cada nivel, una tabla de ese nivel cabe completamente en una página de memoria. Para justificar su solución debe indicar el tamaño de una tabla (en Byte) de cada nivel y dibujar el esquema de direccionamiento con las dimensiones de cada tabla. Puede suponer que la *metadata* se encuentra únicamente en la última tabla.

Para este ejercicio, se puede considerar lo siguiente:

- Salvo para la tabla de último nivel, todas las tablas de páginas poseen un ancho de entrada igual al **número de bits de dirección física**. Esto, dado que las tablas de los primeros niveles deben contener **direcciones físicas** que nos trasladen a la ubicación de las tablas siguientes. En cambio, la tabla de último nivel debe contener un número de bits igual a la suma de la cantidad de bits de *frame* (o marco físico) y la cantidad de bits de *metadata*, siguiendo lo impuesto por el ejercicio.
- Los bits de direccionamiento que se distribuyen por nivel son los del **número de página**. Los bits de *offset* **NO SE TOCAN**, dado que estos direccionan dentro de la página, cuyo tamaño **no cambia**.
- Para resolver este problema rápidamente es necesario encontrar la distribución de bits por nivel ideal para un menor tamaño de tabla de páginas en todos los niveles. Como es algo que en general se debe llevar a cabo de forma manual, una **buena aproximación** es distribuir los bits de forma equitativa por nivel.

Siguiendo las consideraciones anteriores, primero veremos los tamaños de tabla de página resultante para un esquema de paginación de dos niveles distribuyendo los bits de direccionamiento virtual de forma equitativa, es decir, 17 bits por nivel. El esquema a continuación:

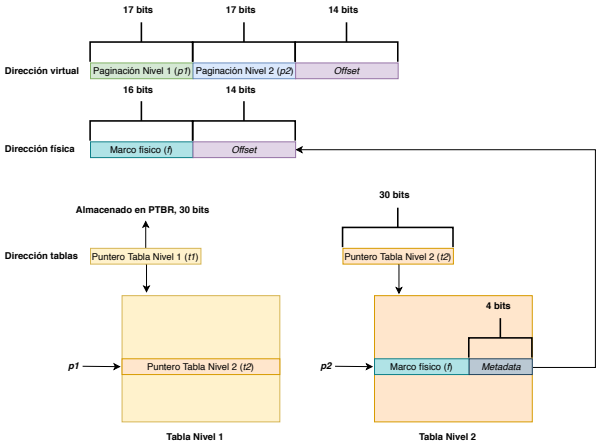

Figura 1: Esquema de paginación multinivel de dos niveles en base al contexto de este problema.

Bajo este esquema, los tamaños son los siguientes[5]:

- **Tabla primer nivel:** $2^{17} \times 30\text{b} = 2^{14} \times 30\text{B} = 32 \times 15\text{KB} = 480\text{KB} > 16\text{KB}$
- **Tabla segundo nivel:** $2^{17} \times 20\text{b} = 2^{14} \times 20\text{B} = 16 \times 20\text{KB} = 320\text{KB} > 16\text{KB}$

Vemos que en ningún nivel la tabla correspondiente cabe en una página, por lo que el esquema no cumple lo solicitado. Por lo tanto, pasamos a utilizar un esquema de paginación de tres niveles. Tratando de ser lo más equitativo posible, haremos uso de 12 bits para el primer nivel y 11 bits para los siguientes dos niveles. El esquema resultante es el siguiente.

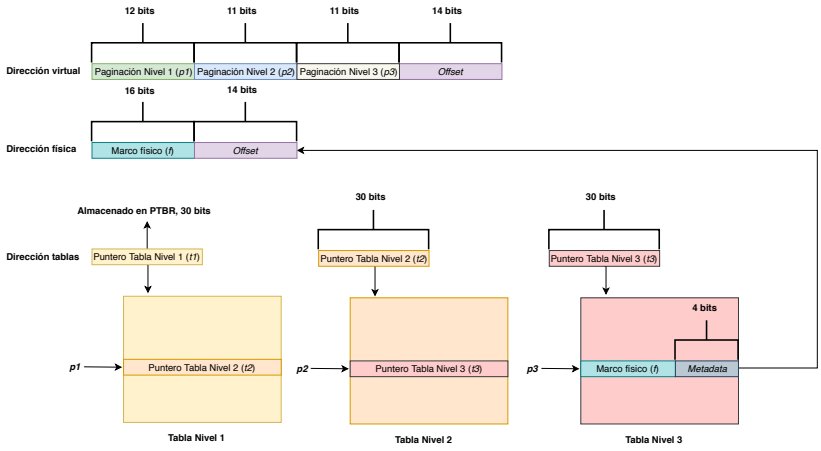

Figura 2: Esquema de paginación multinivel de tres niveles en base al contexto de este problema.

Bajo este esquema, los tamaños son los siguientes:

- **Tabla primer nivel:** $2^{12} \times 30\text{b} = 2^{9} \times 30\text{B} = 2^{10} \times 15\text{B} = 15\text{KB} < 16\text{KB}$
- **Tabla segundo nivel:** $2^{11} \times 30\text{b} = 2^{8} \times 30\text{B} = 512 \times 15\text{B} = 7680\text{B} = 7{,}5\text{KB} < 16\text{KB}$
- **Tabla tercer nivel:** $2^{11} \times 20\text{b} = 2^{8} \times 20\text{B} = 256 \times 20\text{B} = 5120\text{B} = 5\text{KB} < 16\text{KB}$

Vemos que ahora toda tabla, independiente del nivel al que pertenece, cabe dentro de una página como fue solicitado. Es importante destacar, además, que existe más de una distribución de bits por nivel en este esquema que cumple lo pedido.

[5] Notar que seguimos usando la misma fórmula para obtener los tamaños, solo que ahora el ancho de las entradas de cada tabla variará dependiendo del nivel según lo mencionado en las consideraciones.

d. Calcule el tamaño, en Byte, de una tabla de páginas invertida de 1 nivel para este sistema. Suponga que los procesos utilizan 8 bits para almacenar su identificador.

Las tablas invertidas tienen una cantidad de entradas igual al número de marcos físicos existentes en el esquema. Para ubicar el marco físico y traducir la dirección, se busca el número de la entrada cuyo contenido sea la llave <`pid,p`>, siendo `pid` el identificador del proceso y `p` la página correspondiente a la dirección virtual[6]. Dado que el `pid` usa 8 bits y el número de página 34 bits, con 4 bits de *metadata* y 2^{16} marcos físicos tenemos el siguiente esquema:

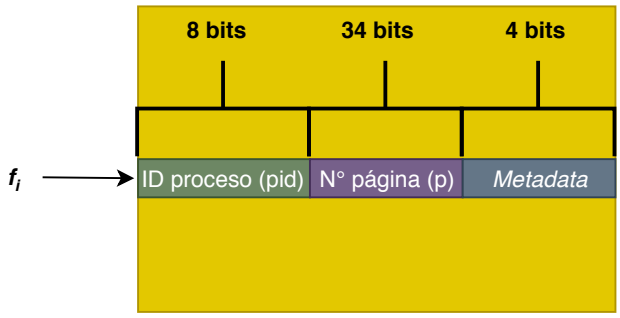

Figura 3: Esquema de tabla invertida, siendo f_i la posición de la tabla que representa el número del marco físico que coincide con la llave <`pid,p`> buscada.

Finalmente, el tamaño se calcula como sigue:

$$T_{p-inv} = \#PTE\text{-}INV \times \texttt{sizeof}(PTE\text{-}INV) = 2^{16} \times 46\text{b} = 2^{13} \times 46\text{B} = 2^{4} \times 23\text{KB} = 368\text{KB}$$

[6]Importante destacar que aquí volvemos a trabajar en el esquema de paginación de un nivel.

3. **(Midterm - II/2018)** Responda verdadero o falso.

   a. El puntero a una tabla de páginas de un proceso se encuentra almacenado en su PCB (*Process Control Block*).

      Verdadero. Como cada proceso tiene una tabla de páginas, existe un puntero a esta por proceso, *i.e.* un puntero en cada PCB.

   b. El uso de segmentación permite eliminar completamente la fragmentación externa.

      Falso. La paginación es la que permite eliminar completamente la fragmentación externa (con el costo de la fragmentación interna). La segmentación solo elimina la fragmentación interna al hacer uso de todo el espacio asignado.

   c. El tamaño del *working set* de un proceso es constante durante la ejecución y está definido como un parámetro del sistema operativo.

      Falso. La cantidad de accesos (Δ) de un proceso puede ser constante, pero no así el tamaño del *working set* dado que la cantidad de accesos no determina la cantidad de páginas exacta que referencie el proceso. Solo estará limitada por Δ: $|\{working\ set\}| \leq \Delta$.

4. **(Midterm - II/2018)** ¿Por qué el concepto de localidad de referencia es importante al momento de diseñar un algoritmo de reemplazo de páginas?

   El concepto de localidad de referencia alude a dos conceptos fundamentales:

   - **Principio de localidad espacial:** Al acceder a un espacio de memoria, lo más probable es que los siguientes accesos sean a espacios contiguos.
   - **Principio de localidad temporal:** Al acceder a un espacio de memoria, lo más probable es que en el corto plazo vuelva a acceder al mismo.

   El cumplimiento de estos dos principios hacen que cobre importancia el diseño de los algoritmos de reemplazo de páginas. El objetivo principal es evitar el reemplazo de páginas que son utilizadas con mayor regularidad, dado que en otro caso se llevarían a cabo muchos *swapping*, lo que disminuiría de sobremanera el rendimiento de la ejecución de nuestros programas.

5. **(I2 - I/2018)** Memoria y algoritmos de reemplazo.

   a. ¿Qué aspecto del problema de direccionamiento de memoria se intenta resolver con el multinivel?

      Con paginación multinivel se busca disminuir el tamaño de las tablas de página, de forma que estas puedan ser almacenadas en una página. Por otra parte, otro motivo válido es la posibilidad de no tener registro de todas las páginas todo el tiempo, dado que nunca se usan todas de forma simultánea.

   b. ¿Qué relación hay entre el concepto de *working set* y el fenómeno de *thrashing*?

      El *working set* consiste en el conjunto de páginas utilizadas por un proceso en un tiempo determinado t. Si la suma de los largos de los *working set* de todos los procesos es mayor al número de marcos físicos disponibles en el sistema en un tiempo t, sabemos que en ese instante habrán constantes *page faults* y *swappings*, mermando el rendimiento del sistema $\rightarrow$ situación conocida como *thrashing*.

c. ¿Qué ventaja presenta la segmentación respecto a la paginación? ¿Qué método se usa en los sistemas modernos?

La ventaja que tiene la segmentación por sobre la paginación es la inexistencia de fragmentación interna, dado que el espacio asignado en cada segmento es utilizado en su totalidad. No obstante lo anterior, esto dificulta de sobremanera el proceso de traducción de direcciones virtuales a físicas, el que es muy directo en el esquema de paginación.

Con respecto a los sistemas modernos, hoy en día se hace uso de "segmentos paginados". En estos, el espacio direccionable se divide en segmentos, pero estos son almacenados en páginas.

d. Se ha mencionado que LRU es un algoritmo que aproxima bastante bien la decisión óptima de qué *frame* conviene reemplazar. ¿Qué dificultades prácticas tiene la implementación de LRU? Mencione otro algoritmo que se implemente en lugar de LRU y por qué su implementación es mejor (si bien no entrega un siempre un mejor resultado que LRU).

La gran dificultad que presenta LRU es el *timestamp* que se necesita almacenar. Si bien se puede hacer uso del *reference bit* para identificar si una página fue accedida o no, el hecho de almacenar el *timestamp* implica el aumento de las tablas de página (dado que cada entrada deberá tener este valor guardado) y la actualización constante de las entradas (lo que es muy ineficiente dado que implica interrupciones sobre una página que está en memoria). Como última opción está la medida del tiempo mediante *hardware*, pero esto de todas formas implicará la comparación de los *timestamps* de todas las páginas, lo que sigue siendo ineficiente.

Los algoritmos que se aproximan a LRU y son más fáciles de implementar son los siguientes:

- **Algoritmo del reloj:** Se mantiene un puntero a una página de forma aleatoria. Al momento de tener que llevar a cabo un reemplazo, se revisa el *reference bit* y la acción a tomar dependerá de su valor: si el bit vale 1, lo cambiamos a 0 (*i.e.* consideramos que no ha sido accedida) y movemos el puntero a la siguiente página[7]. En otro caso, si el bit vale 0, se hace uso de dicha página para el reemplazo. Se puede implementar fácilmente con un arreglo del largo de número de páginas y con el contenido del *reference bit*, lo que implica un menor uso de información.
- **Algoritmo de *aging*:** En vez de almacenar un *reference bit*, guardamos para cada página un *reference Byte* que irá actualizando su valor de la siguiente forma: cada intervalo de tiempo (*tick*) se le añade un *reference* bit al resultado (que vendrá a ser el bit más significativo del registro) y se aplica la operación `shift right` -dividir por 2- al *reference Byte* de la página. Esto ayuda no solo a tener una aproximación de los tiempos en los que ha sido accedida cada página, sino que requiere menos actualizaciones en un tiempo definido, además de que estas se pueden hacer mediante *hardware* (operaciones lógicas fácilmente implementables).
- **Uso de *Dirty bit* + *Reference bit*:** En vez de hacer uso únicamente del *reference bit* para ver si una página ha sido accedida o no, se puede hacer uso del *dirty bit* para ver, además, si una página ha sido modificada o no. Si se usa bajo el esquema del algoritmo del reloj, la elección de la página candidata a seleccionar puede ser mucho mejor al seleccionar una que no haya sido accedida ni modificada en el corto plazo.

[7] En este algoritmo, las páginas se ordenan de forma "circular", es decir, al revisar la última el puntero vuelve a apuntar a la primera página.

e. Describa el significado y el uso de *valid bit*, *dirty bit* y *reference bit*. Si corresponde, incluya para qué algoritmo se usan.

- ***Valid bit*:** Se utiliza para indicar si una entrada de la tabla de páginas o de la TLB está asignada, lo que en el caso de la tabla de páginas indicaría la asignación a un marco físico. Se utiliza para ver si se debe gatillar o no un *page fault* (*valid bit* $= 0$).
- ***Reference bit*:** Se usa para indicar que una página ha sido accedida. Se hace uso de este recurrentemente en el algoritmo del reloj o en otros que busquen aproximar LRU (como *aging*).
- ***Dirty bit*:** Se utiliza para indicar si una entrada ha sido modificada con respecto a la última vez que fue cargada en memoria. Se puede usar como complemento en el algoritmo del reloj para el reemplazo de página, o bien para priorizar mejor en caso de *swapping*.

6. **(Midterm - I/2019)** En los sistemas de paginación, se suele incluir un sistema de *caché* llamado TLB (*Translation Look-aside Buffer*), el cual almacena algunas de las entradas de la tabla de páginas.

   a. ¿Cuántos bits necesita, como mínimo, cada entrada de este TLB? Suponga que el sistema de paginación utiliza p bit para el número de página, f bit para el número de *frame*, y d bit para el *offset*.

   Una entrada de la TLB necesita almacenar tanto el número de página como el número de *frame*. El número de página es necesario para ver si la dirección virtual, asociada a una página, tiene un marco físico asignado a través del cual se pueda llevar a cabo la traducción, mientras que el número de *frame* permite hacer la traducción de forma inmediata. El *offset* en este caso es irrelevante, dado que tanto la dirección virtual como su traducción física tendrán la misma secuencia. Por lo tanto, se necesita un mínimo de $p + f$ bits por entrada de la TLB.

   b. Los TLB, aún teniendo pocas entradas, suelen tener tasas de acierto (*hit rate*) bastante altas. ¿A qué se debe esto?

   Esto se debe a los principios de localidad:

   - Gracias al principio de localidad espacial, es probable que se acceda a muchas direcciones asociadas a una misma página, lo que hace deseable almacenar esta en *caché* para evitar el acceso a la tabla de páginas.
   - Gracias al principio de localidad temporal, es probable que los mismos accesos se repitan varias veces en intervalos de tiempo cortos, lo que implicará que la mayoría hará *hit* luego de haberse almacenado en *caché*.