

TP5

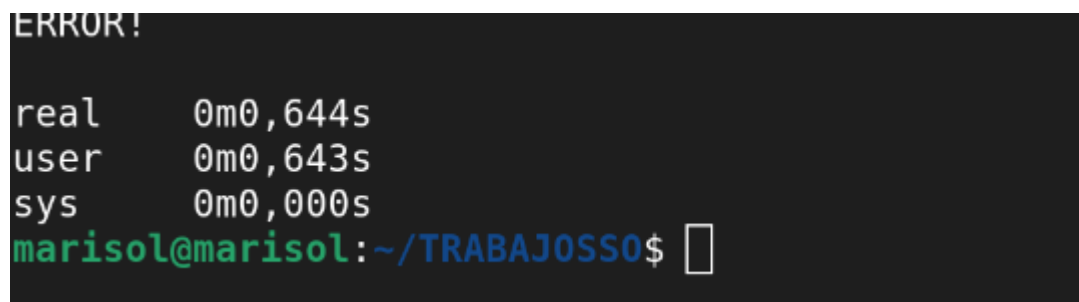
Ejercicio 1

a) ¿Cuántos bytes solicita el programa, al sistema operativo, reservar? (a través de malloc)

```
#define N 240000
#define BSIZE 4096
#define SMALL 4
pp = malloc(N*BSIZE);
```

El sistema operativo solicita al programa reservar $240.000 \times 4096 = 983040000$ Bytes.

b) ¿Cuál es el tiempo de ejecución del programa?. Ayuda: comando time



```
ERROR!
real    0m0,644s
user    0m0,643s
sys     0m0,000s
marisol@marisol:~/TRABAJOSS0$
```

- **real:** 0.644 segundos totales.
- **user:** 10.643 segundos en código de usuario.
- **sys:** 0segundos en código del kernel.

La suma de todos ellos da el timepo de espera.

c) ¿En qué segmento de memoria del proceso se almacena la variable puntero pp? ¿En qué segmento de memoria del proceso se almacenan las variables i,j,k?.

Ayuda: puede utilizar printf para conocer la dirección virtual de las variables. Si la variable es un entero, char, long o float puede acceder a su dirección con &, ejemplo: &i. Si la variable en cambio es un puntero, la variable contiene la dirección virtual de "a lo que apunta". Vea el ejemplo en los últimos slides de clase.

La variable puntero pp se almacena en el st

```
printf("La direccion en donde se almacena la variable pp es 0x%llX\n",&pp);
printf("La direccion en donde se almacena la variable i es 0x%llX\n",&i);
printf("La direccion en donde se almacena la variable j es 0x%llX\n",&j);
printf("La direccion en donde se almacena la variable k es 0x%llX\n",&k);
```

```
marisol@marisol:~/TRABAJOSS0$ ./Ejercicio1TP5
La direccion en donde se almacena la variable pp es 0x4A5B10
La direccion en donde se almacena la variable i es 0x7FFE73F64DAC
La direccion en donde se almacena la variable j es 0x7FFE73F64DA8
La direccion en donde se almacena la variable k es 0x7FFE73F64DA4
```

La variable pp se almacena en el segmento de stack, ya que su dirección de memoria está más cerca del inicio de esta, y crece hacia abajo.

Para guardar espacio en el archivo ejecutable, todas las variables inicializadas explícitamente se encuentran después del segmento de texto, y las variables noinicializadas están luego de las variables inicializadas, así todo lo que tiene que hacer el compilador es poner una palabra en el encabezamiento diciendo cuántos bytes se le deben asignar.

Las variables i,j,k se almacenan en el segmento de datos no inicializados, conocido como BSS (Block Started By Symbol), en este segmento todas las variables son inicializadas con el valor cero después de carga. La existencia de esto es solo una optimización.

d. ¿En qué segmento de memoria del proceso el sistema operativo reserva la memoria solicitada con malloc? (y que es apuntada por pp). Mencione los posibles system calls de Linux que malloc podría utilizar para cumplir su objetivo.

La memoria solicitada "malloc" se almacena en el segmento de HEAP, ya que el tamaño de esta cambia dinámicamente en tiempo de ejecución, en dónde el tamaño heap crece hacia el inicio de la memoria.

e. Incorpore al programa una sentencia printf que reporte la dirección virtual de la función main(), de i y de j. Nota: tenga en cuenta que las direcciones de memoria virtual en PC son de 64bits.

```
printf("La direccion en donde se almacena la funcion main es 0x%llX\n",&main);
```

```
La direccion en donde se almacena la funcion main es 0x401725
La direccion en donde se almacena la variable pp es 0x4A5B10
La direccion en donde se almacena la variable i es 0x7FFCAE97756C
La direccion en donde se almacena la variable j es 0x7FFCAE977568
La direccion en donde se almacena la variable k es 0x7FFCAE977564
```

f. Debajo de /proc/PID_DEL_PROCESO/ el kernel Linux presenta muchos archivos con información relativa a un proceso en ejecución.

Ejecute el programa, y en otra terminal, ejecute `cat /proc/PID/maps` (reemplace PID por el process id de su programa en ejecución), el cual le mostrará los segmentos de memoria del proceso con direcciones virtuales.

NOTA: las direcciones reportadas por el archivo maps están en hexadecimal.
Indique cuáles son las direcciones virtuales de los segmentos de:

- texto (código máquina ejecutable)
- datos (variables globales inicializadas y sin inicializar)
- heap
- stack

Indique también cuáles son sus tamaños.

```
marisol@marisol:~$ cat /proc/13711/maps
00400000-00401000 r--p 00000000 103:05 918364 /home/marisol/TRABAJOSS0/Ejercicio1TP5
00401000-00477000 r-xp 00001000 103:05 918364 /home/marisol/TRABAJOSS0/Ejercicio1TP5
00477000-0049f000 r--p 00077000 103:05 918364 /home/marisol/TRABAJOSS0/Ejercicio1TP5
0049f000-004a4000 r--p 0009e000 103:05 918364 /home/marisol/TRABAJOSS0/Ejercicio1TP5
004a4000-004a6000 rw-p 000a3000 103:05 918364 /home/marisol/TRABAJOSS0/Ejercicio1TP5
004a6000-004ac000 rw-p 00000000 00:00 0
266c3000-266e5000 rw-p 00000000 00:00 0
7fc019f6d000-7fc0548ee000 rw-p 00000000 00:00 0
7fc0548ee000-7fc0548f2000 r--p 00000000 00:00 0
7fc0548f2000-7fc0548f4000 r-xp 00000000 00:00 0
7ffc5d061000-7ffc5d082000 rw-p 00000000 00:00 0
[heap]
[vvar]
[vdso]
[stack]
```

letra	significado
r	lectura
w	escritura
x	ejecución permitida
p	privado
s	compartido

EL formato de el comando `cat /proc/PID/maps` es

```
[ rango de direcciones ] [ permisos ] [ offset ] [ dev ] [ inode ] [
archivo o recurso ]
```

- Stack. Se encuentra desde la dirección 7FFC5D061000 hasta la dirección 7FFC5D08200
- Heap. Se encuentra desde la dirección 266C3000 hasta la dirección 266E5000.
- Segmento de datos. Se encuentra desde el final del segmento de texto hasta el segmento de HEAP, es decir desde la dirección 00477000 hasta la dirección 266C3000 ?
- Segmento de texto. Se encuentra desde la dirección 00401000 hasta la dirección 00477000, ya que tiene permiso 'x' y 'r' (lectura y ejecución).
-

Observe detenidamente el tamaño del segmento cuyo nombre coincide con el que Ud. respondió en el inciso d. ¿Coincide la cantidad de memoria de ese segmento con la reservada por el programa y calculada en a.?

Ejercicio 2

a. ¿Cuánto demora la ejecución de cada versión?

Ejecución de versión 1:

```
ERROR!

real    0m0,634s
user    0m0,634s
sys     0m0,000s
marisol@marisol:~/TRABAJOSSO$
```

Ejecución de versión 2:

```
marisol@marisol:~/TRABAJOSSO$ time ./version2
ERROR!

real    0m0,678s
user    0m0,672s
sys     0m0,004s
marisol@marisol:~/TRABAJOSSO$
```

Ejecución de versión 3:

ERROR!

```
real    0m0,791s
user    0m0,790s
sys     0m0,000s
```

```
marisol@marisol:~/TRABAJOSS0$
```

Primera versión:

```
Performance counter stats for './Ejercicio1TP5':
```

```
<not supported>      dTLB-store-misses
                   42.573      cache-misses
```

```
0,641190594 seconds time elapsed
```

```
0,641236000 seconds user
```

```
0,000000000 seconds sys
```

```
root@marisol:/home/marisol/TRABAJOSS0#
```

27.947 cpu_core/dTLB-store-misses

Segunda versión:

```
Performance counter stats for './version2':
```

```
<not supported>      dTLB-store-misses
                   43.505      cache-misses
```

```
0,686707936 seconds time elapsed
```

```
0,686770000 seconds user
```

```
0,000000000 seconds sys
```

```
root@marisol:/home/marisol/TRABAJOSS0#
```

3.673 cpu_core/dTLB-store-misses

Tercera versión:

```
Performance counter stats for './version3':
```

```
<not supported>      dTLB-store-misses
                   54.077      cache-misses
```

```
0,803217290 seconds time elapsed
```

```
0,799162000 seconds user
```

```
0,003995000 seconds sys
```

```
root@marisol:/home/marisol/TRABAJOSS0#
```

68.833 cpu_core/dTLB-store-misses

¿Qué es la caché de datos TLB de un procesador, qué contiene, y para qué se utiliza? ¿Qué es la caché de datos de un procesador?

Esta es una unidad de traducción de direcciones, mantiene las entradas a la tabla de página recientemente accedidas, permitiendo así que el acceso a las direcciones sea de forma directa y mucho más rápido que un almacenamiento secundario.

Esta contiene un hardware que utiliza de forma paralela llamado CAM, (Content Addressable Memory), el cual dada una dirección, CAM busca los valores guardados en paralelo, retornando la respuesta en solo unos pocos ciclos de reloj.

La caché de datos de un procesador es una memoria pequeña integrada a él de alta velocidad, que mantiene las instrucciones recientemente utilizadas, haciendo así que el procesador encuentre de forma veloz instrucciones que ya utilizó, mejorando la eficiencia y el rendimiento, evitando que este busque las instrucciones en la RAM.

1. ¿Qué versión tiene el peor rendimiento? ¿Encuentra una relación entre ese rendimiento con los fallos de caché y TLB reportados por perf?.

La versión que tiene el peor rendimiento es la primera versión, ya que el arreglo bi-direccional está declarado como N filas por BSIZE columnas, pero luego es recorrido por columnas en vez de filas, generando así más fallos de caché ya que las matrices se guardan por filas contiguamente, es decir si tenemos :

1	2
3	4

En la memoria se almacenará como [1,2,3,4], por lo que al recorrerlo por columnas tendrá más fallos de TLB.

2. ¿Qué versión tiene el mejor rendimiento?. Analice en grupo de qué manera el programa de mejor rendimiento recorre la memoria reservada.

La segunda versión recorre la matriz de forma natural, es decir por filas, podemos ver que tiene menor cantidad de fallos de caché TLB, y en cuanto a tiempo y acceso a la memoria caché es similar a la versión 1, por lo que esta es la de mejor rendimiento.

3. ¿Qué diferencia existe en la manera de recorrer la memoria entre el programa de mejor rendimiento y el de rendimiento medio?. Hipotetice o explique por qué razón el programa de

mejor rendimiento supera al de rendimiento medio. En su análisis incluya una relación entre la manera de recorrer la memoria y los datos reportados por perf.

Realice un diagrama o dibujo de la memoria y la forma en que se recorre.

La primer versión recorre la matriz de la siguiente forma, primero fila 1, llegando hasta el final de esta, y luego fila 2, y así sucesivamente hasta llegar a el último elemento de la matriz.

Como las direcciones de una matriz se almacenan contiguamente en la memoria, el recorrido de esta genera un recorrido secuencial, es decir, existirá un primero fallo al inicio porque no habrá nada en la memoria, pero luego cuándo existe el bloque de datos de la matriz no habrá ningún fallo de dirección hasta llegar al final de esta, por eso este tipo de recorrido es de mejor rendimiento.

1	2	3
↓	↓	↓
↓	↓	↓
↓	↓	↓

En la segunda versión, que es la de peor rendimiento, la matriz se almacena por filas, pero luego se recorre por columnas, es decir, cada búsqueda de cada dirección en esta matriz generará al menos uno o más fallos hasta que se encuentre, ya que no hay una búsqueda secuencial en la memoria, está de forma desorganizada.

El primer elemento generará un fallo, el segundo otro, el tercero otro... así sucesivamente.

1	2	3
→	→	→
→	→	→
→	→	→

La versión 3, recorre de forma desorganizada una matriz, pero aún así es de mejor rendimiento porque recorre de forma natural a esta, aunque cada cierto tiempo tendrá fallo ya que pega un salto de una dirección a otra, si bien es desorganizada tendrá menos fallos que la versión 2.

1	2	3		4	5	6
→	→	→		→	→	→

1	2	3		4	5	6
→	→	→		→	→	→
→	→	→		→	→	→
→	→	→		→	→	→

Ejercicio 3. Hardware y software de memoria virtual en un sistema moderno.

a. Indique cómo son los campos de una dirección virtual en un procesador Intel/AMD de arquitectura AMD64. Considere un sistema Linux actual en ese sistema que usa tabla de páginas de 4KB.

La arquitectura AMD64 define un formato de dirección virtual de 64 bits, del cual los 48 bits de orden inferior se utilizan en las implementaciones actuales. Esto permite hasta 256 (2,48 bytes) de espacio de direcciones virtuales.

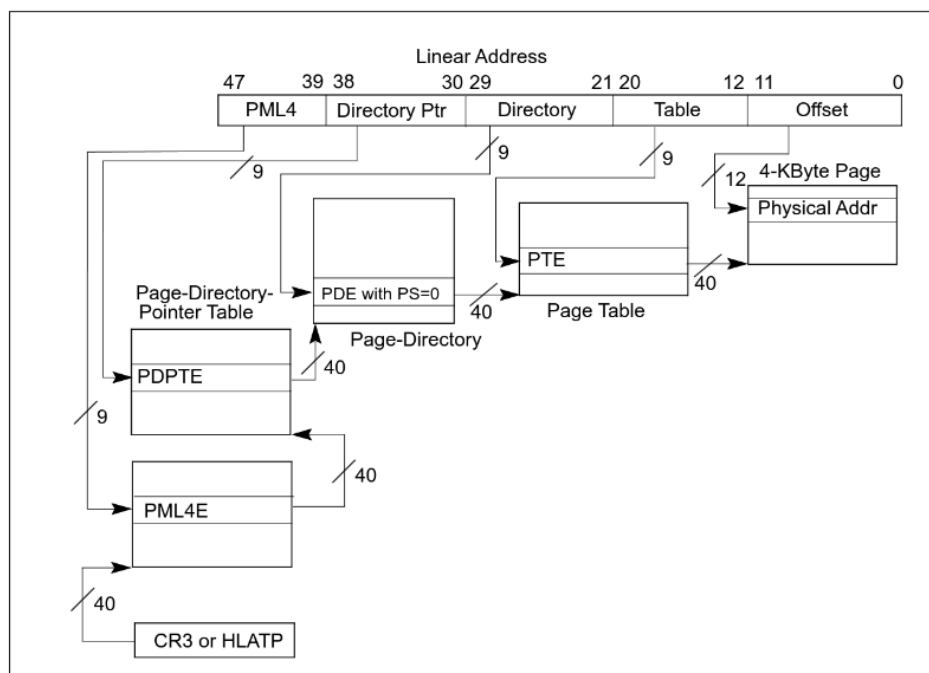


Figure 5-8. Linear-Address Translation to a 4-KByte Page Using 4-Level Paging

La dirección de la memoria virtual se divide en:

- 0-11 Offset. Indica en qué posición de esos 4KB de dentro del marco físico (espacio real en la RAM) se encuentra.
- 12-20 Tabla. Indica la entrada en la "Page Table", cada entrada apunta directamente a un marco físico.
- 21-29 Directory. Le indica la entrada de la "Page Directory", que a su vez le indicará la entrada correspondiente en la "Page Table".

- 30-38 Directory PTR. Se utiliza para acceder a una entrada en la PDPTE. Esta entrada, le indicará la entrada de la "Page-directory" correspondiente.
- 39-47 PML4. Sirve para ubicar la entrada en la tabla PML4, que apunta a la siguiente tabla PDPTE.
- PDPTE(Page directory pointer table, PML3). Esta es una tabla intermedia que se utiliza en el proceso de traducción, mediante en componente PML4 . Cada entrada en esta, apunta a a una Page Directory, que a su vez apunta a page tables.

Responder:

- ¿Cuál es el tamaño de cada tabla de páginas o directorio? Estas son llamadas (AMD programmer manual): Page Map Level 4 Table, Page Directory Pointer Table, Page Directory Table, Page Table.

Cada una tiene 512 entradas de 64 bits, es decir, 4KB.

- ¿Cuántos bits realmente se utilizan de la dirección virtual?
De los 64 bits de la dirección de la memoria virtual, se utilizan 48.
- ¿Cuál es el tamaño máximo de memoria virtual que un proceso puede utilizar?

2^{48} bytes=256TB(terabytes)

- ¿Cómo es posible que un proceso funcione si el tamaño de memoria virtual que utiliza es mayor que la memoria física?

Esto es debido a la técnica llamada "paginación a demanda", se refiere a una técnica en la que se divide el proceso en partes pequeñas de igual tamaño llamadas pages, el sistema mantiene las páginas más referenciadas en la memoria principal, y mueve las copias de otras páginas en almacenamiento secundario como puede ser un disco rígido, a medida que el programa necesita páginas para ejecutarse, el SO trae las distintas páginas a la memoria RAM.

EL hardware necesario para aplicar esta técnica consiste en una **tabla de páginas** y una **unidad de traducción de direcciones** (TLB). Una tabla de páginas reside en la memoria **kernel**, y existe una de estas por cada proceso. Cuando ocurre un cambio de contexto, el SO cambia la página que utiliza.

Sea un proceso que utiliza un total de 128MB de memoria en este sistema. ¿Cuándo el proceso comienza a ejecutarse, cuántas páginas necesita cargar el SO para que el proceso comience a ejecutarse?

En el sistema explicado anteriormente, dónde se utilizan las tablas de página PTML4, PTPE, PDE, PTE y la memoria RAM, el proceso para poder ejecutarse debería tener una página cargada en cada una de estas tablas para poder realizar la traducción de la dirección y así poder asignarle un marco físico en la memoria RAM. Como cada entrada es de 4KB, necesitaría tener 16KB cargados en la memoria.

¿Qué significa fragmentación interna?

Fragmentación interna se define a un problema en el que un proceso no utiliza el tamaño completo que se le es asignado en una página, se llama **interna** porque justamente ocurre dentro del bloque de páginas, no existe un aprovechamiento adecuado del espacio.

b. Sea un proceso que referencia una dirección de memoria virtual. Describa un posible escenario para cada una de las siguientes situaciones (si no es posible tal escenario, explique por qué):

Un **fallo de TLB** ocurre cuando:

La CPU accede a una dirección virtual.

La traducción **no está almacenada en el TLB**.

- El hardware de la CPU (o el sistema operativo) **busca la traducción completa** en la jerarquía de tablas de páginas (PML4, PDPT, PD, PT).
- Cuando la encuentra, **la carga en el TLB** para futuros accesos.
- Si la entrada existe en la tabla y la página está en RAM → **no hay fallo de página**.

El **fallo de página** ocurre cuando:

El proceso accede a una dirección virtual válida.

La página referida no está presente en la memoria física (RAM) en ese momento.

a) fallo del TLB sin fallo de página;

Sí puede suceder y es muy común.

- El proceso accede a una dirección virtual.
- Esa dirección **no está en el TLB** (fallo de TLB).
- Pero la entrada correspondiente en la **tabla de páginas sí existe** y la página está presente en RAM (bit "present"=1).
- El hardware recorre la jerarquía de tablas de páginas (PML4, PDPT, PD, PT) y encuentra la dirección física.
- Actualiza el TLB con esta nueva traducción.

♦ **Ejemplo real:**

Un proceso accede a una dirección virtual que no usó en mucho tiempo → el TLB había sido reemplazado o limpiado, pero la página está en RAM.

b) fallo del TLB con fallo de página;

Sí puede suceder y es común cuando se accede por primera vez a una página.

♦ **Qué ocurre:**

- El proceso accede a una dirección virtual.
- Esa dirección no está en el TLB (fallo de TLB).
- El hardware recorre la tabla de páginas y ve que la página **no está presente en RAM** (bit "present"=0).

- Se produce un **fallo de página (page fault)**.
- El sistema operativo atiende la excepción → carga la página desde disco o la crea en blanco en RAM.
- Después actualiza la tabla de páginas y el TLB.

♦ **Ejemplo real:**

Primera vez que el proceso accede a una variable muy grande que no estaba cargada → la página se trae del archivo binario o de swap.

c) acierto del TLB sin fallo de página;

Sí, es el escenario ideal y muy eficiente.

♦ **Qué ocurre:**

- El proceso accede a una dirección virtual.
- El TLB **tiene la traducción lista** (acierto TLB).
- La página está presente en RAM (no hay fallo de página).
- Acceso inmediato a la dirección física.

♦ **Ejemplo real:**

Accediendo a datos de un bucle donde ya todas las direcciones fueron usadas recientemente y están en el TLB.

d) acierto del TLB con fallo de página.

No puede suceder.

♦ **Por qué:**

- Si el TLB tiene la traducción, significa que ya conoce la dirección física y la página está marcada como “presente” en RAM.
- Un fallo de página solo ocurre si el bit “present”=0.
- Si el bit “present”=0, la entrada **no sería cargada en el TLB** → no puede haber acierto TLB.

♦ **Conclusión:**

Acierto del TLB implica que no hay fallo de página, por definición.

e) ¿Puede suceder también fallo de caché?

El fallo de caché es **independiente** del fallo de TLB o de página.

♦ **Qué ocurre:**

- Una vez que tienes la dirección física, la CPU busca los datos en la jerarquía de caché (L1, L2, L3).
- Si no están ahí, ocurre un **cache miss** → la CPU tiene que ir a la RAM a buscarlos.

♦ **Ejemplo real:**

Leyendo datos de un array grande → parte puede estar en la caché y otra parte en la RAM
→ habrá aciertos y fallos de caché.