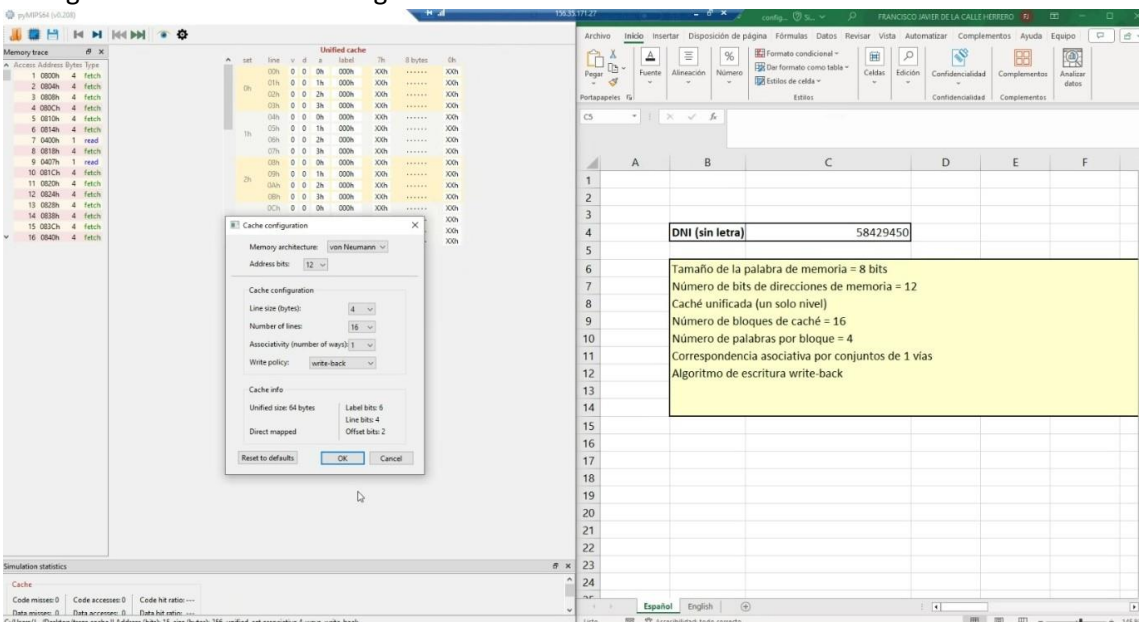
RESUMEN EXAMEN PRACTICO2

EJERCICIO 1- Preguntas sobre PYMIPS varias

1. Configurar la caché con la configuración dada.



NUMERO BITS DE DIRECCIONES DE MEMORIA = Address Bits
CACHÉ UNIFICADA
NUMERO DE BLOQUES DE CACHÉ = Number of lines
NUMERO DE PALABRAS POR BLOQUE * TAMAÑO DE LA PALABRA DE MEMORIA(Bytes) = Line Size (Bytes)
CORRESPONDENCIA ASOCIATIVA POR CONJUNTOS DE 1 VIA = Associativity
WRITE-BACK

2. Enumera un ejemplo de localidad temporal para el código y otro de localidad espacial para los datos.

La localidad temporal de los programas viene motivada principalmente por la existencia de bucles.

Localidad espacial: las palabras próximas en el espacio de memoria a las recientemente referenciadas tienen una alta probabilidad de ser también referenciadas en el futuro cercano.

3. 4 primeros accesos que provoquen fallo

Recuadro de la caché en rojo

4. 2 primeros accesos que provoquen reemplazo y el bloque reemplazado

Bit d (sucio) se pone a rojo y la etiqueta cambia y se pone en rojo también.

5. 2 primeros accesos que provoquen actualización y el bloque reemplazo

Lo mismo que el reemplazo pero hay que esperar a una orden write. Después de eso ejecutamos hasta que el bit d=1 cambia a d=0 y se ponen en rojo

6. Cuál fue el primer acceso de escritura y poner todos los bits de etiqueta, conjunto y desplazamiento.

Esperar a la primera orden write y calcular la etiqueta, conjunto y desplazamiento (pasando el ratón por el acceso en Memory trace)

7. Cuál sería el porcentaje de aciertos de la traza si fueran accesos aleatorios.

(Tamaño de cache / Tamaño memoria) *100

EJERCICIO 2.1- Analizar los accesos de caché L1 o L3 (Bandwidth y Valgrind)

PARTE DE BANDWIDTH:

wget https://www.atc.uniovi.es/grado/2ac/files/session3-3.tar.gz

tar xvfz session3-3.tar.gz

- Instala nasm:

```
$>  sudo apt install nasm
```

- Entra dentro del directorio `bandwidth-1.3.1` que contiene un benchmark de memoria y compila la versión de 32 bits:

```
$>  make bandwidth32
```

Compilar el programa: make bandwidth32

Si te aparecen errores: sudo apt install nasm

Ejecutar programa: . /bandwidth32 | more

Ejecutar programa más rápido: sudo nice -n -2 ./bandwidth32 –fastest

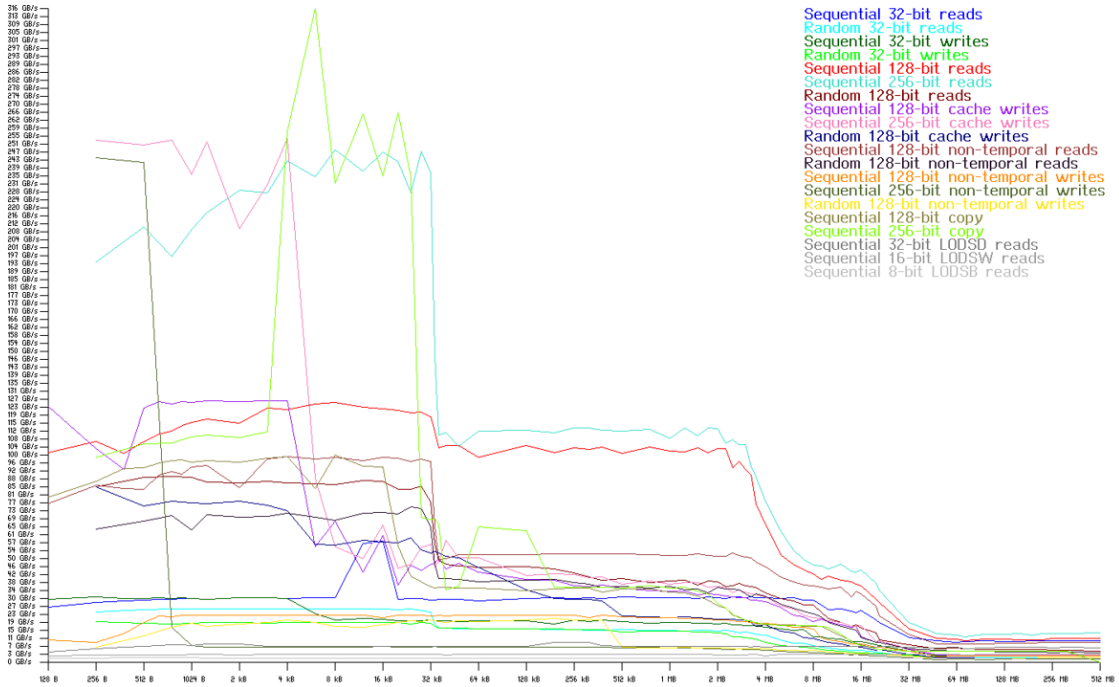
```
This is bandwidth version 1.3.1.
Copyright (C) 2005-2016 by Zack T Smith.

This software is covered by the GNU Public License.
It is provided AS-IS, use at your own risk.
See the file COPYING for more information.

CPU family: GenuineIntel
CPU features: MMX SSE SSE2 SSE3 SSSE3 SSE4.1 SSE4.2 AES AVX AVX2 XD

Cache 0: L1 data cache,        line size 64,  8-ways,    64 sets, size 32k
Cache 1: L1 instruction cache, line size 64,  8-ways,   128 sets, size 64k
Cache 2: L2 unified cache,     line size 64, 16-ways,  4096 sets, size 4096k
Cache 3: L3 unified cache,     line size 64, 12-ways, 40960 sets, size 30720k

Notation: B = byte, kB = 1024 B, MB = 1048576 B.

CPU speed is 3417.6 MHz.
```

Cache L1 dividida en código y datos:

- Datos: tamaño de línea (64 bytes), vías (8), conjuntos (64), tamaño (32 kbytes)
- Código: tamaño de línea (64 bytes), vías (8), conjuntos (128), tamaño (64 kbytes)

Cache L2 unificada: tamaño de línea (64 bytes), vías (16), conjuntos (4096), tamaño (4096 kbytes)

CacheL3 unificada: tamaño de línea (64 bytes), vías (12), conjuntos (40960), tamaño (30720 kbytes)



Si te preguntan el ancho de banda de un experimento das el pico más alto para ese experimento.

El acceso secuencial es más rápido siempre ya que se aprovecha el principio de localidad.

Cada escalón es un nivel de memoria, comienza en L1, después va L2, luego L3 y por último Memoria Principal.

PARTE DE VALDRIND:

Compilar programa: gcc [archivo.c] -o [nombre ejecutable]

Ejecutar con valgrind: valgrind --tool=cachegrind ./ejecutable

Ejecutar programa con tiempo: time ./ejecutable

```
==1671== I   refs:      16,474,406,288
==1671== I1  misses:               713
==1671== LLi misses:               711
==1671== I1  miss rate:           0.00%
==1671== LLi miss rate:           0.00%
==1671==
==1671== D   refs:       8,243,647,773  (6,592,333,405 rd   + 1,651,314,368 wr)
==1671== D1  misses:             1,347  (          938 rd   +           409 wr)
==1671== LLd misses:             1,250  (          848 rd   +           402 wr)
==1671== D1  miss rate:            0.0% (          0.0%     +           0.0%  )
==1671== LLd miss rate:            0.0% (          0.0%     +           0.0%  )
==1671==
==1671== LL refs:                2,060  (        1,651 rd   +           409 wr)
==1671== LL misses:              1,961  (        1,559 rd   +           402 wr)
==1671== LL miss rate:             0.0% (          0.0%     +           0.0%  )
student@2ac:~/sesion3-3/3-3loc1$
```

Alguna de las abreviaturas que utiliza *cachegrind* para mostrar los resultados son:

- *I*: instrucción.
- *D*: datos (*rd*: lectura, *wr*: escritura).
- *I1*: caché L1 de instrucciones.
- *D1*: caché L1 de datos.
- *LL*: resto de cachés de la jerarquía (L2, L3).
- *LLi*: accesos a instrucción en el resto de cachés de la jerarquía.
- *LLd*: accesos a datos en el resto de cachés de la jerarquía.

EJERCICIO 2.2- MAPS

La organización del espacio de direcciones virtuales accesible a una tarea se encuentra en el archivo `/proc/<pid>/maps`, donde `<pid>` es un patrón de sustitución que hace referencia al identificador de la tarea (proceso). Para mostrarlo, lleva a cabo las siguientes operaciones:

- Compila y enlaza el archivo `3-4maps.c` proporcionado por el profesor, usando la orden

```
$> gcc 3-4maps.c -o 3-4maps-1
```

Se trata simplemente de un programa que muestra por pantalla su `pid` y espera por la pulsación de una tecla para terminar, lo que nos permitirá observar su archivo *maps* antes de que termine.

- Ejecuta el programa `3-4maps-1`. No pulses la tecla ENTER.
- Desde una segunda terminal, ejecuta la siguiente orden

```
$> more /proc/<pid>/maps
```

sustituyendo `<pid>` por el identificador de proceso mostrado por el programa. Una vez lo has hecho, ya puedes terminar la tarea pulsando la tecla ENTER en la primera interfaz de comandos.

En la terminal tendrás un resultado similar a la imagen mostrada en la figura 2.

```
student@2ac:~/practica4$ more /proc/4306/maps
08048000-08049000 r-xp 00000000 fc:00 39481      /home/student/practica4/3-4maps-1
08049000-0804a000 r--p 00000000 fc:00 39481      /home/student/practica4/3-4maps-1
0804a000-0804b000 rw-p 00001000 fc:00 39481      /home/student/practica4/3-4maps-1
09fdd000-09fff000 rw-p 00000000 00:00 0          [heap]
b75f1000-b77a0000 r-xp 00000000 fc:00 129053     /lib/i386-linux-gnu/libc-2.23.so
b77a0000-b77a1000 ---p 001af000 fc:00 129053     /lib/i386-linux-gnu/libc-2.23.so
b77a1000-b77a3000 r--p 001af000 fc:00 129053     /lib/i386-linux-gnu/libc-2.23.so
b77a3000-b77a4000 rw-p 001b1000 fc:00 129053     /lib/i386-linux-gnu/libc-2.23.so
b77a4000-b77a7000 rw-p 00000000 00:00 0
b77b1000-b77b3000 rw-p 00000000 00:00 0
b77b3000-b77b5000 r--p 00000000 00:00 0          [vvar]
b77b5000-b77b6000 r-xp 00000000 00:00 0          [vdso]
b77b6000-b77d8000 r-xp 00000000 fc:00 129029     /lib/i386-linux-gnu/ld-2.23.so
b77d8000-b77d9000 rw-p 00000000 00:00 0
b77d9000-b77da000 r--p 00022000 fc:00 129029     /lib/i386-linux-gnu/ld-2.23.so
b77da000-b77db000 rw-p 00023000 fc:00 129029     /lib/i386-linux-gnu/ld-2.23.so
bfb34000-bfb55000 rw-p 00000000 00:00 0          [stack]
student@2ac:~/practica4$
```

Figura 2. Ejemplo de archivo /proc/pid/maps en Linux 32 bits (pid = 4306)

Como podrás observar, en la primera columna del archivo `/proc/pid/maps` aparecen los diferentes rangos de memoria separados por un guion. La primera dirección del rango es la dirección virtual de comienzo, mientras que la segunda es la dirección final (más uno). Comprueba cómo todas estas direcciones se encuentran en la zona de memoria reservada para la tarea, esto es, el rango `00000000h–BFFFFFFFh`.

El programa `3-4maps-1` utiliza la biblioteca estándar de C en su versión dinámica. La razón es que el compilador `gcc` enlaza por defecto con esta biblioteca. Si se desea enlazar con la versión estática de la librería (`libc.a`) debe indicarse de forma explícita añadiendo la opción `-static`.

- Compila y enlaza el programa `3-4maps.c` usando bibliotecas estáticas, empleando la siguiente orden

```
$> gcc -static 3-4maps.c -o 3-4maps-2
```

- Ejecuta el archivo resultante, visualiza su archivo *maps* y observa los cambios experimentados con respecto a la versión estática, usando la figura 2 como referencia.
- Compara los tamaños de los archivos ejecutables `3-4maps-1` y `3-4maps-2` usando la siguiente orden.

```
$> ls -l 3-4maps-*
```

¿Qué diferencia de tamaño hay en KiB? Responde en el cuestionario: pregunta 2.

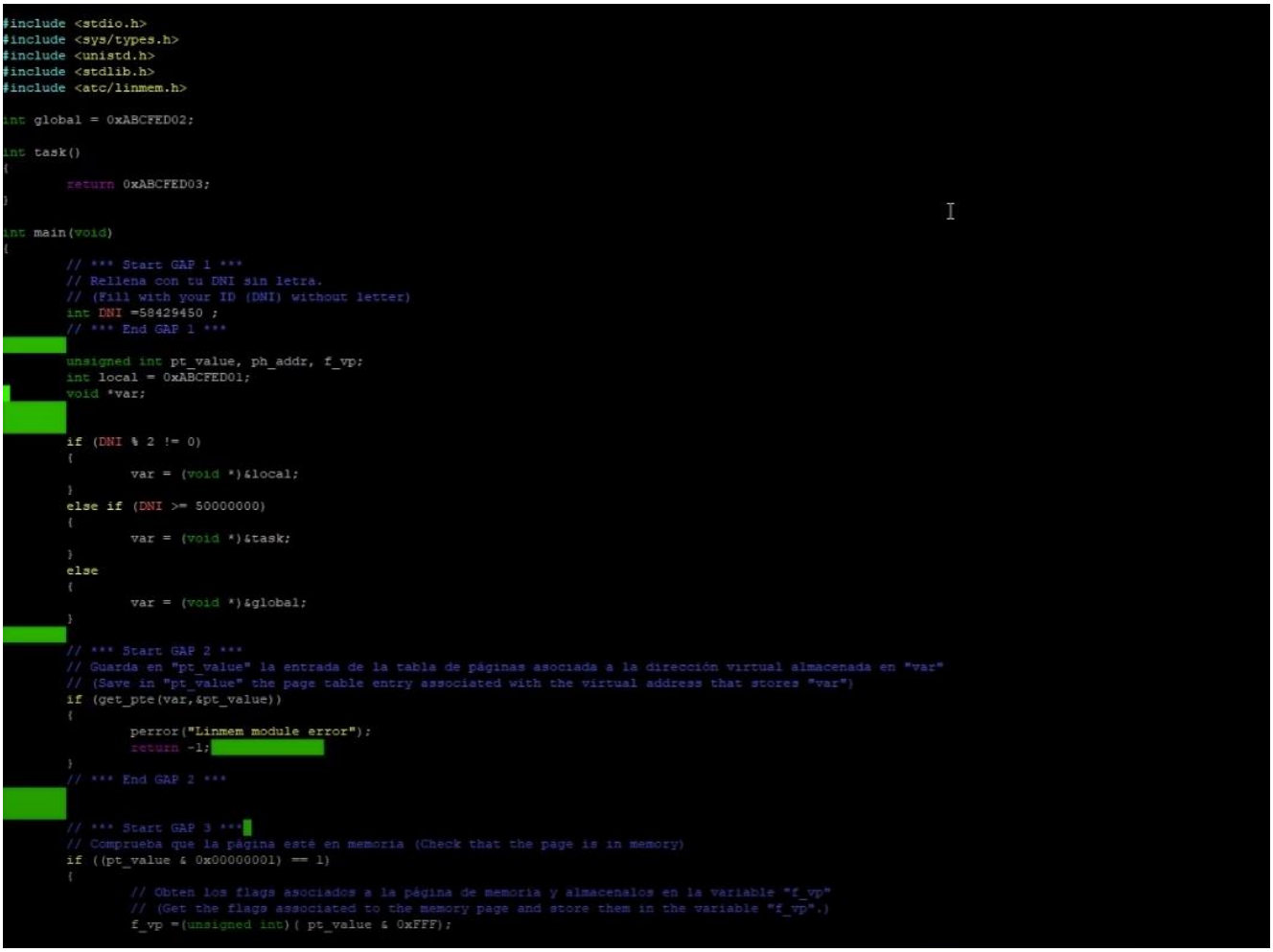
Esta diferencia es debida a que al código del programa `3-4maps-2` se ha añadido código de la biblioteca estándar porque se ha enlazado estáticamente.

EJERCICIO 3 – Memoria Virtual

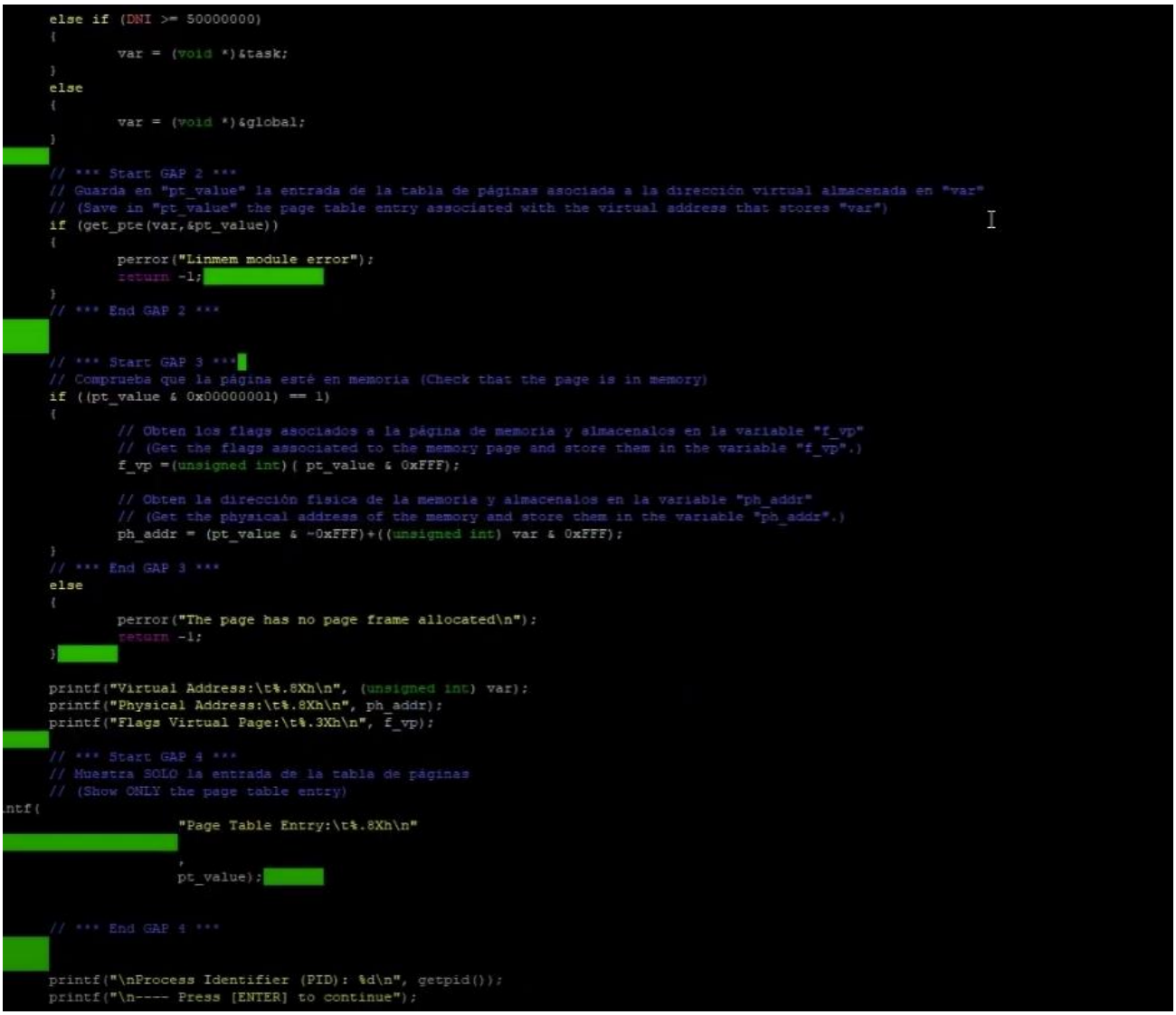
IMPORTANTE!!!!! -> Mirar si está activo el linmem

systemctl status linmem → Tiene que poner una luz verde y tal.

systemctl enable linmem → Para activarlo.



```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <atc/linmem.h>

int global = 0xABCFED02;

int task()
{
        return 0xABCFED03;
}

int main(void)
{
        // *** Start GAP 1 ***
        // Rellena con tu DNI sin letra.
        // (Fill with your ID (DNI) without letter)
        int DNI =58429450 ;
        // *** End GAP 1 ***

        unsigned int pt_value, ph_addr, f_vp;
        int local = 0xABCFED01;
        void *var;

        if (DNI % 2 != 0)
        {
                var = (void *)&local;
        }
        else if (DNI >= 50000000)
        {
                var = (void *)&task;
        }
        else
        {
                var = (void *)&global;
        }

        // *** Start GAP 2 ***
        // Guarda en "pt_value" la entrada de la tabla de páginas asociada a la dirección virtual almacenada en "var"
        // (Save in "pt_value" the page table entry associated with the virtual address that stores "var")
        if (get_pte(var,&pt_value))
        {
                perror("Linmem module error");
                return -1;
        }
        // *** End GAP 2 ***

        // *** Start GAP 3 ***
        // Comprueba que la página esté en memoria (Check that the page is in memory)
        if ((pt_value & 0x00000001) == 1)
        {
                // Obten los flags asociados a la página de memoria y almacenalos en la variable "f_vp"
                // (Get the flags associated to the memory page and store them in the variable "f_vp".)
                f_vp =(unsigned int)( pt_value & 0xFFF);
```



```
else if (DNI >= 50000000)
{
        var = (void *)&task;
}
else
{
        var = (void *)&global;
}

// *** Start GAP 2 ***
// Guarda en "pt_value" la entrada de la tabla de páginas asociada a la dirección virtual almacenada en "var"
// (Save in "pt_value" the page table entry associated with the virtual address that stores "var")
if (get_pte(var,&pt_value))
{
        perror("Linmem module error");
        return -1;
}
// *** End GAP 2 ***

// *** Start GAP 3 ***
// Comprueba que la página esté en memoria (Check that the page is in memory)
if ((pt_value & 0x00000001) == 1)
{
        // Obten los flags asociados a la página de memoria y almacenalos en la variable "f_vp"
        // (Get the flags associated to the memory page and store them in the variable "f_vp".)
        f_vp =(unsigned int)( pt_value & 0xFFF);

        // Obten la dirección física de la memoria y almacenalos en la variable "ph_addr"
        // (Get the physical address of the memory and store them in the variable "ph_addr".)
        ph_addr = (pt_value & ~0xFFF)+((unsigned int) var & 0xFFF);
}
// *** End GAP 3 ***
else
{
        perror("The page has no page frame allocated\n");
        return -1;
}

printf("Virtual Address:\t%.8Xh\n", (unsigned int) var);
printf("Physical Address:\t%.8Xh\n", ph_addr);
printf("Flags Virtual Page:\t%.3Xh\n", f_vp);

// *** Start GAP 4 ***
// Muestra SOLO la entrada de la tabla de páginas
// (Show ONLY the page table entry)
ntf(
                "Page Table Entry:\t%.8Xh\n"
                ,
                pt_value);

// *** End GAP 4 ***

printf("\nProcess Identifier (PID): %d\n", getpid());
printf("\n---- Press [ENTER] to continue");
```

- **Para compilar acordarse de lmen!!!!!!:**

gcc {archivo.c} -o {nombre_ejecutable} -lmen

- **Para que te salgan los warnings:**

gcc -Wall {archivo.c} -o {nombre_ejecutable} -lmen

- **Para el resto de las librerías:**

gcc {archivo.c} -o {nombre_ejecutable} -lmen -lrt -lm -pthread -lpthread

Para mirar que bits se activan.

Por ejemplo si la flag es 025h. En binario seria 0000 0010 0101. Serían los bits 0 a 11.

Se activarían el bit de presencia, el de usuario, y el de dirty.

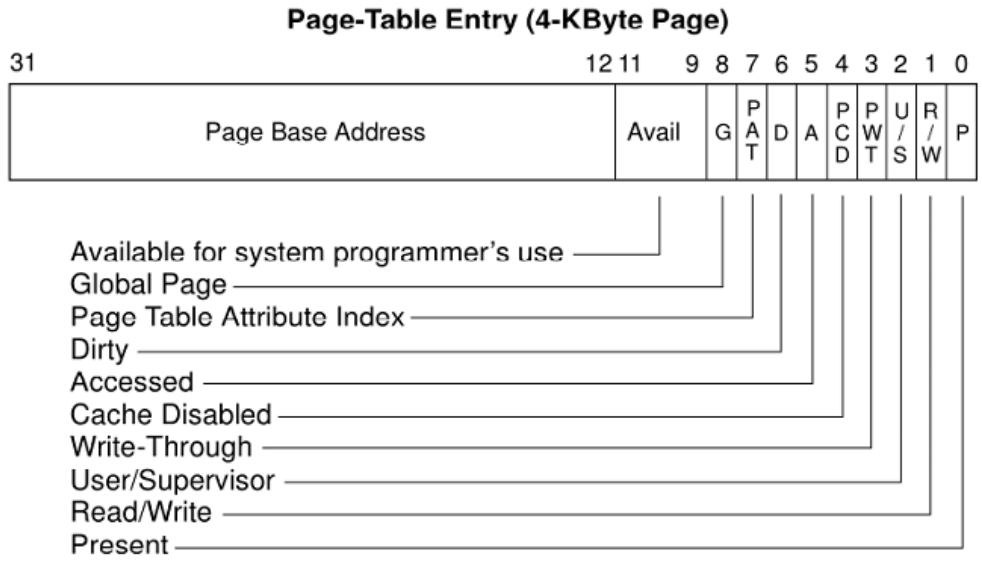


Figura 4. Organización de la ETP