# ANEXO C

# TRANSCRIPCION DEL SOFTWARE

En el capítulo correspondiente a la revisión del código hemos visto sólo las partes de éste más significativas. Para un más sencillo acceso al mismo, transcribimos en este anexo el contenido completo de los ficheros que componen la aplicación.

Las modificaciones efectuadas al kernel han sido expuestas en su totalidad en el mismo capítulo, por lo que no hacemos mención a las mismas en este anexo.

Tampoco se transcribe la aplicación de usuario `matrix.c` que se adjunta al software, puesto que se ha hecho en el epígrafe correspondiente.

## *C.1 p4pc_dev.c*

```
/*************************************************************************/
/*Fichero:  p4pc_dev.c                                                   */
/*Autor:    Agustin Isasa Cuartero - UOC TFC GNU/LINUX                   */
/*Resumen:  Device driver para acceso a los performance counters del     */
/*          Pentium 4.  Mediante write(), una aplicacion de usuario      */
/*          podra configurar adecuadamente los correspondientes MSRs,    */
/*          mientras que con read() podra consultar el resultado         */
/*          acumulado en el correspondiente contador.                    */
/*Interface:                                                             */
/*Notas:                                                                 */
/*************************************************************************/

#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/fs.h>
#include <linux/types.h>
#include <asm/uaccess.h>      /* copy_from/to_user */
#include <asm/processor.h>    /* cpuid_eax() */
#include <asm/semaphore.h>    /* DECLARE_MUTEX */
#include <asm/msr.h>          /* rdmsr y wrmsr64 */

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Agustin Isasa, TFC GNU/LINUX, UOC 2004");
MODULE_DESCRIPTION("Driver para acceso a performance counters en Pentium
4");

/* Declaraciones*************************************************/
#define MAX_COUNTERS          18    /* 18 contadores en el Pentium 4 */
#define DEVICE_NAME           "p4pc_dev"
#define P4PC_MAJOR            100
#define MAX_ENTRY_MSG_BUF     20          /* Tamaño buffer entrada */
#ifndef TRUE
#define TRUE                  1
#endif
#ifndef FALSE
#define FALSE                 0
#endif
```

```
int p4pc_dev_init(void);
void p4pc_dev_exit(void);
static int p4pc_dev_open(struct inode *, struct file *);
static int p4pc_dev_release(struct inode *, struct file *);
static ssize_t p4pc_dev_read(struct file *, char *, size_t, loff_t *);
static ssize_t p4pc_dev_write(struct file *, const char *, size_t, loff_t
*);
static void printkf(char *);         /* Escribir en consola X */
static int pentium4_confirm(void);  /* Comprobar proc. es Pentium 4 */
static int kernel_24_confirm(void); /* Comprobar si kernel >= 2.4 */

/* Variables globales ************************************************/
static unsigned char entry_msg[MAX_ENTRY_MSG_BUF];  /* Buffer entrada */

/* Configuracion de un determinado contador */
struct counter_config{
      __u32 escr_bits;  /* Vector de bits del escr */
      __u32 cccr_bits;  /* Vector de bits del cccr */
      __u16 escr_addr;  /* Direccion del escr */
      __u16 cccr_addr;  /* Direccion del cccr */
      __u16 counter_addr;     /* Direccion del contador */
      __s64 account;    /* Cuenta atras hasta overflow */
      _Bool pebs_on;    /* Precise Event Based Sample? */
      __u64 result;     /* Resultado a devolver al espacio de usuario */
      _Bool active;     /* ¿Esta el contador activo? */
      pid_t process;    /* Proceso que inicia el contador */
      struct semaphore sem;   /* Semaforo posible codigo conflictivo */
};

/* Array de MAX_COUNTERS (18) posibles configuraciones de contadores */
static struct counter_config counters_structure[MAX_COUNTERS];

/* Estructura declarando las funciones del driver */
static struct file_operations fops ={
      .read = p4pc_dev_read,
      .write = p4pc_dev_write,
      .open = p4pc_dev_open,
      .release = p4pc_dev_release
};

/* Funciones *********************************************************/
/* Iniciamos el modulo */
int p4pc_dev_init(void)
{
      int ret, i;

#ifdef __SMP__          /* Confirmamos que corremos en UP */
      printkf("P4PC_ERROR_K:  El driver no esta diseñado para SMP.  El");
      printkf("    modulo no sera cargado.");
      return -EPERM;
#endif

      if(pentium4_confirm() < 0)    /* Confirmamos corremos Pentium 4 */
            return -EPERM;
      if(kernel_24_confirm() < 0)   /* Confirmamos kernel >= 2.4 */
            return -EPERM;
      if((ret = register_chrdev(P4PC_MAJOR, DEVICE_NAME, &fops)) < 0){
            printkf("P4PC_ERROR_K:  Fallo al registrar dispositivo
                  p4pc_dev");
            return ret;
```

```
	}
	for(i=0; i<MAX_COUNTERS; i++)
		init_MUTEX(&counters_structure[i].sem);

	printkf("P4PC_AVISO_K:  Asignado mayor 100 al dispositivo
		p4pc_dev");

	return 0;
}

/* Liberamos el modulo */
void p4pc_dev_exit(void)
{
	int ret;
	if((ret = unregister_chrdev(P4PC_MAJOR, DEVICE_NAME)) < 0)
		printkf("P4PC_ERROR_K:  Fallo al retirar dispositivo
			p4pc_dev");
}

/* Abrimos driver */
static int p4pc_dev_open(struct inode *inode, struct file *file)
{
	int o_minor;
	o_minor = MINOR(inode->i_rdev);

	/* Si el dispositivo ya esta en uso activo por un proceso, y si
	otro lo requiere... */
	if((counters_structure[o_minor].active == TRUE)
	&& (current->pid != counters_structure[o_minor].process)){
		printkf("P4PC_ERROR_K:  Contador en uso por otro proceso\n");
		return -EBUSY; /* ...salimos indicando dispositivo ocupado */
	}

	/* Si el dispositivo no esta activo, se mantiene inactivo hasta que
	se escribe en el, pero ya guardamos el proceso */
	if(counters_structure[o_minor].active == FALSE){
		counters_structure[o_minor].process = current->pid;
	}

	return 0;
}

/* Cerramos driver */
static int p4pc_dev_release(struct inode *inode, struct file *file)
{
	int c_minor;
	c_minor = MINOR(inode->i_rdev);

	/* Si un proceso pretende cerrar dispositivo abierto y en uso por
	otro... */
	if((counters_structure[c_minor].active == TRUE)
	&& (current->pid != counters_structure[c_minor].process)){
		printkf("P4PC_ERROR_K:  Contador en uso por otro proceso\n");
		return -EBUSY; /* ...salimos indicando dispositivo ocupado */
	}

	/* Si venimos de un stop_counter(), habremos puesto a 0 el bit
	"enable" del cccr */
	if(!(counters_structure[c_minor].cccr_bits & 0x00001000)){
		counters_structure[c_minor].active = FALSE;
	}
```

```
/* ////////////////////////////////////////////////////////////////// */
/* Codigo para discriminacion por proceso:  actuamos en su task_struct */
      current->nr_active_counters--;
      clear_bit((counters_structure[c_minor].cccr_addr % MAX_COUNTERS),
            &current->active_cccrs_flag);
/* ////////////////////////////////////////////////////////////////// */


      return 0;
}

/* read() al driver */
static ssize_t p4pc_dev_read(struct file *filp, char *buffer, size_t count,
loff_t *offset)
{
      int r_minor;
      __u32 low, high;

      /* Controlamos tamaño lectura */
      if(count != sizeof(counters_structure[r_minor].result))
            return -1;

      struct inode *inode = filp->f_dentry->d_inode;
      r_minor = MINOR(inode->i_rdev);

      /* Controlamos que el contador tiene algo para leer */
      if(counters_structure[r_minor].active == FALSE){
            printkf("P4PC_ERROR_K:  El contador no esta activo\n");
            return -ENODEV;
      }

      /* Leemos contador y lo pasamos a result */
      rdmsr(counters_structure[r_minor].counter_addr, low, high);
      counters_structure[r_minor].result = (__u64)high;
      counters_structure[r_minor].result =
            (counters_structure[r_minor].result) << 32;
      counters_structure[r_minor].result =
            counters_structure[r_minor].result | (__u64)low;

      /* Pasamos resultado de lectura al espacio de usuario */
      if(copy_to_user(buffer, (char*) &
            (counters_structure[r_minor].result),
            sizeof(counters_structure[r_minor].result)))
            return -EFAULT;

      return count;
}

/* write() al driver */
static ssize_t p4pc_dev_write(struct file *filp, const char *buffer,
      size_t count, loff_t *offset)
{
      int w_minor;
      unsigned int aux, i;
      __u16 counter_addr_aux;
      unsigned char *ptr_aux;

      /* Controlamos tamaño escritura */
      if(count != MAX_ENTRY_MSG_BUF)
            return -1;
```

```
struct inode *inode = filp->f_dentry->d_inode;
w_minor = MINOR(inode->i_rdev);

/* Si el dispositivo ya esta en uso activo por un proceso, y si
otro quiere escribir los MSRs en uso... */
if((counters_structure[w_minor].active == TRUE)
&& (current->pid != counters_structure[w_minor].process)){
      printkf("P4PC_ERROR_K:  Contador en uso por otro proceso\n");
      return -EBUSY; /* ...salimos indicando dispositivo ocupado */
}

/* Copiamos del espacio de usuario */
if(copy_from_user(entry_msg, buffer, count))
      return -EFAULT;

/* Calculamos la entrada en el array de configuraciones de contadores
*/
ptr_aux = (char *)&counter_addr_aux;
for(i=0; i<sizeof(counter_addr_aux); i++)
      *(ptr_aux+i) = entry_msg[12+i];
aux = counter_addr_aux % MAX_COUNTERS;

/* Controlamos que el minor coincide con entrada en array de
contadores, es decir, que el dispositivo es el que corresponde a la
direccion del contador que figura en el mensaje que pasa la
aplicacion de usuario */
if(w_minor != (int)aux){
      printkf("P4PC_ERROR_K:  Se ha pasado al driver una pareja");
      printkf("    dispositivo/contador erronea.");
      return -EPERM;
}

/* Rellenamos la entrada correspondiente en array de contadores */
/* Vector de bits escr */
ptr_aux = (char *)&counters_structure[aux].escr_bits;
for(i=0; i<sizeof(counters_structure[aux].escr_bits); i++)
      *(ptr_aux+i) = entry_msg[i];
/* Vector de bits cccr */
ptr_aux = (char *)&counters_structure[aux].cccr_bits;
for(i=0; i<sizeof(counters_structure[aux].cccr_bits); i++)
      *(ptr_aux+i) = entry_msg[4+i];
/* Direccion del escr */
ptr_aux = (char *)&counters_structure[aux].escr_addr;
for(i=0; i<sizeof(counters_structure[aux].escr_addr); i++)
      *(ptr_aux+i) = entry_msg[8+i];
/* Direccion del cccr */
ptr_aux = (char *)&counters_structure[aux].cccr_addr;
for(i=0; i<sizeof(counters_structure[aux].cccr_addr); i++)
      *(ptr_aux+i) = entry_msg[10+i];
/* Direccion del contador */
counters_structure[aux].counter_addr = counter_addr_aux;
/* Magnitud de cuenta atras, que... */
ptr_aux = (char *)&counters_structure[aux].account;
for(i=0; i<5; i++)            /* ...limitamos a 5*8 = 40 bits */
      *(ptr_aux+i) = entry_msg[14+i];
if(counters_structure[aux].account > 0) /* Aseguramos account <0 */
      counters_structure[aux].account =
            (counters_structure[aux].account)*(-1);
/* pebs? */
if((unsigned char)entry_msg[19] == 0)
```

```
                counters_structure[aux].pebs_on = FALSE;
        else
                counters_structure[aux].pebs_on = TRUE;
        /* active */
        counters_structure[aux].active = TRUE;

/* ////////////////////////////////////////////////////////////////// */
/* Codigo para discriminacion por proceso:  actuamos en su task_struct */
        set_bit((counters_structure[aux].cccr_addr % MAX_COUNTERS),
                &current->active_cccrs_flag);
        current->nr_active_counters++;
/* ////////////////////////////////////////////////////////////////// */

        /* Escribimos, por fin, los contenidos de escr, counter y cccr */
        wrmsr64((__u32)counters_structure[aux].escr_addr,
                (__u64)counters_structure[aux].escr_bits);
        wrmsr64((__u32)counters_structure[aux].counter_addr,
                (__u64)counters_structure[aux].account);
        wrmsr64((__u32)counters_structure[aux].cccr_addr,
                (__u64)counters_structure[aux].cccr_bits);

        return count;
}

/* Inicio y salida del modulo */
module_init(p4pc_dev_init);
module_exit(p4pc_dev_exit);

/* Funcion printkf(char *) para escribir en consola X */
static void printkf(char *str)
{
        struct tty_struct *this_tty;
        this_tty = current->tty;
        printk(str); printk("\n");    /* Ademas de a consola x, al log */
        if (this_tty != NULL){
        (*(this_tty->driver).write)(
                this_tty,               /* La propia tty */
                0,              /* No tomamos el string del espacio usuario */
                str,                    /* String pasado como parametro */
                strlen(str));           /* Longitud del string */
        /* Incluimos LF y CR tras la salida a consola del string */
        (*(this_tty->driver).write)(this_tty, 0, "\015\012", 2);
        }
}

/* Funcion pentium4_confirm() que averigua si corremos bajo un procesador
de la familia Pentium 4 */
static int pentium4_confirm(void)
{
        unsigned int op = 1; /* Cargando eax con 1 y ejecutando cpuid... */
        __u32 eax_output;
        eax_output = (cpuid_eax(op)); /* ...tenemos Processor Signature */
        eax_output = eax_output & 0x00000f00;
        if (eax_output != 0x00000f00){ /* Familia P4 es xx00111100xxxxxx */
                printkf("");
                printkf("P4PC_ERROR_K:  Este modulo solo puede correr bajo
                un");
                printkf("    procesador de la familia Pentium 4.  No se ha");
                printkf("    detectado este tipo de procesador, por lo que
                el");
                printkf("    modulo no sera cargado.");
```

```
            printkf("");
            return -1;
      }
      return 0;
}

/* Funcion k_24_confirm(), que habilita este modulo solo para versiones del
kernel 2.4 o superiores */
static int kernel_24_confirm(void)
{
      if (LINUX_VERSION_CODE < KERNEL_VERSION(2,4,0)){
            printkf("");
            printkf("P4PC_ERROR_K:  El kernel debe ser de serie 2.4 o
            superior.");
            printkf("    Este modulo puede causar errores con el actual");
            printkf("    kernel, por lo que no sera cargado.");
            printkf("");
            return -1;
      }
      else
      return 0;
}
```

## C.2 p4pc_dev.c

```
/*************************************************************************/
/*Fichero:  p4pc_event.h                                                 */
/*Autor:    Agustin Isasa Cuartero - UOC TFC GNU/LINUX                   */
/*Resumen:  header para acceso a performance counters del Pentium 4,     */
/*          tanto para el device driver como para aplicacion usuario.    */
/*          Para mayor generalidad se consideran todas las parejas de    */
/*          registros, aunque en esta aplicacion no se contemplara la    */
/*          tecnologia Hyper-Threading.                                  */
/*Interface:                                                             */
/*Notas:    Utilizamos en los nombres de las variables y funciones el    */
/*          prefijo p4pc (de pentium 4 perf. counters) para evitar       */
/*          posibles colisiones en el espacio de nombres del nucleo.     */
/*************************************************************************/

#ifndef P4PC_EVENT_HDR
#define P4PC_EVENT_HDR

#include <linux/types.h>      /* Diversos tipos */

/* Algunas definiciones e informacion previas */
#define MAX_COUNTERS    18    /* 18 contadores en el Pentium 4 */
#define P4PC_MAJOR      100
#define P4PC_MAXBUF     20  /* Tamaño mensaje al driver mediante write() */
/* Codigos de errores */
#define P4PC_EMKNOD      1     /* Error en mknod()
                                  (install_p4pc_devices()) */
#define P4PC_ENOMSG     2     /* No existe mensaje para write().
                              ¿Se ha seleccionado evento para el contador? */
#define P4PC_EOPEN      3     /* Error en open(). ¿Esta cargado el
                                 modulo p4pc_dev.o? */
#define P4PC_EWRITE     4     /* Error en write() */
#define P4PC_ENODESCR   5     /* No existe descriptor.  ¿Se ha iniciado
                                 el contador? */
#define P4PC_EREAD      6     /* Error en read() */
int fd[MAX_COUNTERS];         /* Array de descriptores, uno por contador */
/* Array de buffers para mensajes a incluir en write() y pasarlo al driver.
Incluye formatos a nivel bit de los escr y cccr, asi como resto recursos */
__u8 p4pc_write_msg_array[MAX_COUNTERS][P4PC_MAXBUF];

/* Vinculacion entre contadores y dispositivos, ordenados segun
(counter_addr % MAX_COUNTERS), es decir, (direccion del contador modulo 18)
*/
#define BPU_COUNTER0          12
#define BPU_COUNTER1          13
#define BPU_COUNTER2          14
#define BPU_COUNTER3          15
#define MS_COUNTER0           16
#define MS_COUNTER1           17
#define MS_COUNTER2           0
#define MS_COUNTER3           1
#define FLAME_COUNTER0        2
#define FLAME_COUNTER1        3
#define FLAME_COUNTER2        4
#define FLAME_COUNTER3        5
#define IQ_COUNTER0           6
#define IQ_COUNTER1           7
#define IQ_COUNTER2           8
#define IQ_COUNTER3           9
#define IQ_COUNTER4           10
```

```
#define IQ_COUNTER5           11
/* Array auxiliar con los nombres de los dispositivos /dev */
static const char *p4pc_dev_names[MAX_COUNTERS] = {"/dev/p4pc0",
"/dev/p4pc1", "/dev/p4pc2", "/dev/p4pc3", "/dev/p4pc4", "/dev/p4pc5",
"/dev/p4pc6", "/dev/p4pc7", "/dev/p4pc8", "/dev/p4pc9", "/dev/p4pc10",
"/dev/p4pc11", "/dev/p4pc12", "/dev/p4pc13", "/dev/p4pc14", "/dev/p4pc15",
"/dev/p4pc16",    "/dev/p4pc17"};

/* Auxiliares */
#ifndef TRUE
#define TRUE      1
#endif
#ifndef FALSE
#define FALSE     0
#endif

/*
 * En el siguiente bloque se incluyen definiciones para las direcciones
 * de los MSR implicados (ESCR, CCCR, contadores y algunos registros
 * relevantes), asi como el ordinal de los ESCR en su grupo.  En este
 * caso, se ha preferido seguir fielmente la nomenclatura seguida por
 * Intel en sus diferentes manuales.  No aplicamos, por lo tanto, prefijo
 * en los nombres, aunque previamente se ha comprobado que no hay
 * colisiones con el codigo del kernel (asm/msr.h).
 */

/* Direcciones de los contadores */
#define MSR_BPU_COUNTER0      0x0300
#define MSR_BPU_COUNTER1      0x0301
#define MSR_BPU_COUNTER2      0x0302
#define MSR_BPU_COUNTER3      0x0303
#define MSR_MS_COUNTER0       0x0304
#define MSR_MS_COUNTER1       0x0305
#define MSR_MS_COUNTER2       0x0306
#define MSR_MS_COUNTER3       0x0307
#define MSR_FLAME_COUNTER0    0x0308
#define MSR_FLAME_COUNTER1    0x0309
#define MSR_FLAME_COUNTER2    0x030a
#define MSR_FLAME_COUNTER3    0x030b
#define MSR_IQ_COUNTER0       0x030c
#define MSR_IQ_COUNTER1       0x030d
#define MSR_IQ_COUNTER2       0x030e
#define MSR_IQ_COUNTER3       0x030f
#define MSR_IQ_COUNTER4       0x0310
#define MSR_IQ_COUNTER5       0x0311

/* Direcciones de los CCCRs */
#define MSR_BPU_CCCR0         0x0360
#define MSR_BPU_CCCR1         0x0361
#define MSR_BPU_CCCR2         0x0362
#define MSR_BPU_CCCR3         0x0363
#define MSR_MS_CCCR0          0x0364
#define MSR_MS_CCCR1          0x0365
#define MSR_MS_CCCR2          0x0366
#define MSR_MS_CCCR3          0x0367
#define MSR_FLAME_CCCR0       0x0368
#define MSR_FLAME_CCCR1       0x0369
#define MSR_FLAME_CCCR2       0x036a
#define MSR_FLAME_CCCR3       0x036b
#define MSR_IQ_CCCR0          0x036c
#define MSR_IQ_CCCR1          0x036d
```

```
#define MSR_IQ_CCCR2          0x036e
#define MSR_IQ_CCCR3          0x036f
#define MSR_IQ_CCCR4          0x0370
#define MSR_IQ_CCCR5          0x0371

/* Direcciones de los ESCRs */
#define MSR_BSU_ESCR0         0x03a0
#define MSR_BSU_ESCR1         0x03a1
#define MSR_FSB_ESCR0         0x03a2
#define MSR_FSB_ESCR1         0x03a3
#define MSR_FIRM_ESCR0        0x03a4
#define MSR_FIRM_ESCR1        0x03a5
#define MSR_FLAME_ESCR0       0x03a6
#define MSR_FLAME_ESCR1       0x03a7
#define MSR_DAC_ESCR0         0x03a8
#define MSR_DAC_ESCR1         0x03a9
#define MSR_MOB_ESCR0         0x03aa
#define MSR_MOB_ESCR1         0x03ab
#define MSR_PMH_ESCR0         0x03ac
#define MSR_PMH_ESCR1         0x03ad
#define MSR_SAAT_ESCR0        0x03ae
#define MSR_SAAT_ESCR1        0x03af
#define MSR_U2L_ESCR0         0x03b0
#define MSR_U2L_ESCR1         0x03b1
#define MSR_BPU_ESCR0         0x03b2
#define MSR_BPU_ESCR1         0x03b3
#define MSR_IS_ESCR0          0x03b4
#define MSR_IS_ESCR1          0x03b5
#define MSR_ITLB_ESCR0        0x03b6
#define MSR_ITLB_ESCR1        0x03b7
#define MSR_CRU_ESCR0         0x03b8
#define MSR_CRU_ESCR1         0x03b9
#define MSR_IQ_ESCR0          0x03ba
#define MSR_IQ_ESCR1          0x03bb
#define MSR_RAT_ESCR0         0x03bc
#define MSR_RAT_ESCR1         0x03bd
#define MSR_SSU_ESCR0         0x03be
#define MSR_MS_ESCR0          0x03c0
#define MSR_MS_ESCR1          0x03c1
#define MSR_TBPU_ESCR0        0x03c2
#define MSR_TBPU_ESCR1        0x03c3
#define MSR_TC_ESCR0          0x03c4
#define MSR_TC_ESCR1          0x03c5
#define MSR_IX_ESCR0          0x03c8
#define MSR_IX_ESCR1          0x03c9
#define MSR_ALF_ESCR0         0x03ca
#define MSR_ALF_ESCR1         0x03cb
#define MSR_CRU_ESCR2         0x03cc
#define MSR_CRU_ESCR3         0x03cd
#define MSR_CRU_ESCR4         0x03e0
#define MSR_CRU_ESCR5         0x03e1

/* Otros registros relevantes para performance counters */
#define IA32_MISC_ENABLE      0x01a0
#define IA32_DS_AREA          0x0600
#define IA32_PEBS_ENABLE      0x03f1

/* Ordinales de los ESCRs en su grupo */
#define BPU       0x00
#define IS        0x01
#define MOB       0x02
```

```
#define ITLB      0x03
#define PMH       0x04
#define IX        0x05
#define FSB       0x06
#define BSU       0x07

#define MS        0x00
#define TC        0x01
#define TBPU      0x02

#define FLAME     0x00
#define FIRM      0x01
#define SAAT      0x02
#define U2L       0x03
#define DAC       0x04

#define IQ        0x00
#define ALF       0x01
#define RAT       0x02
#define SSU       0x03
#define CRU0      0x04
#define CRU1      0x05
#define CRU2      0x06


/*
 * En el siguiente bloque procedemos a las definiciones de las
 * estructuras de datos.  Utilizamos dos niveles que resultaran
 * complementarios:
 *
 * 1 - En el nivel mas alto seguimos la estructura y nomenclatura de la
 *     documentacion de Intel ("Events on Intel Pentium 4 Processors...",
 *     "IA-32 Intel Architecture Optimization" y "IA-32 Intel
 *     Architecture Soft. Develop. Manual vol. 3:  System Programming
 *     Guide").  Definimos primero que evento vamos a analizar
 *     (estructura p4pc_event_id), y luego lo completamos con la
 *     informacion necesaria para saber como lo vamos a hacer (estructura
 *     p4pc_event_count_config, que ademas de la struct event_id, incluye
 *     otras estructuras vinculadas a cada uno de los aspectos de la
 *     configuracion de los registros).
 *
 * 2 - En un nivel mas bajo implementamos las estructuras de cada uno de
 *     los registros implicados (escr y cccr) a nivel de bits, utilizando
 *     un buffer de chars.  En este buffer, ademas, incluimos los
 *     restantes datos a pasar al device driver, y que son necesarios
 *     para la correcta configuracion de todos los recursos involucrados
 *     en el muestreo de los contadores.
 *
 *     La informacion, como puede verse, es la misma en ambos casos, pero
 *     su formato se adecua al uso que se haga desde la aplicacion o el
 *     kernel:  en general, la aplicacion de usuario configurara el
 *     evento y su muestreo con las estructuras del nivel mas alto.
 *     Estas se traduciran, con la funcion p4pc_ecc_to_wm(), al buffer de
 *     nivel mas bajo, que se convertira en el mensaje contenido en el
 *     write() dirigido al driver y originado en la aplicacion de
 *     usuario.
 */


/* Identificamos un evento segun su documentacion, tal y como queda
definida en "Events on Intel Pentium 4 Processors..." */
```

```
struct p4pc_event_id{
      __u8 event_group;       //6 bits - Clase de eventos
      __u16 event_mask;       //2 bytes - Mascara en una clase
      __u32 escr;             //Direccion ESCR
      __u32 counter;          //Direccion Counter
      __u32 cccr;             //Direccion CCCR
      __u8 cccr_escr;         //3 bits - Ordinal 0-7 de ESCR en su grupo
};

/* Info relativa a PCL y threads */
struct p4pc_usr_os_t{
      _Bool t1_usr;           //1 bit - Hilo 1 de user, sin uso si no HT
      _Bool t1_os;            //1 bit - Hilo 1 de SO, sin uso si no HT
      _Bool t0_usr;           //1 bit - Hilo 0 de usuario
      _Bool t0_os;            //1 bit - Hilo 0 de SO
      __u8 active_thread;     //2 bits -Hilos HT en cccr, 11 si no HT
};

/* Info relativa a etiquetado de uops */
struct p4pc_tag{
      _Bool tag_enable;       //1 bit - Habilitamos etiquetar uops...
      __u8 tag_value;         //4 bits - ...con este valor
};

/* Filtrado de contadores */
struct p4pc_filter{
      _Bool compare;          //1 bit - Habilitamos filtrado
      _Bool complement;       //1 bit - Si 0, >; si 1, =<
      __u8 threshold;         //4 bits - Umbral de filtrado
      _Bool edge;             //1 bit - Deteccion flanco de subida
};

/* Tratamiento de overflows */
struct p4pc_ovf{
      _Bool force_ovf;        //1 bit - Fuerza overflow en cada incremento
      _Bool ovf_pmi_t0;       //1 bit - Interrupcion si ovf, thread 0
      _Bool ovf_pmi_t1;       //1 bit - Interrupcion si ovf, thread 1
      //p4pc_boolean ovf;     //1 bit  //No necesita activarlo usuario}
      __s64 account;       //Hasta 5 bytes - Cuenta atras hasta ovf (+ o -)
      _Bool cascade;          //1 bit - Activa counters en cascada si ovf
};

/* Resumimos toda la informacion anterior recogiendo en un struct la
configuracion completa de un determinado evento a analizar */
struct p4pc_event_count_config{
      struct p4pc_event_id *event_id;
      struct p4pc_usr_os_t *usr_os_t;
      struct p4pc_tag *tag;
      struct p4pc_filter *filter;
      struct p4pc_ovf *ovf;
      _Bool pebs_on;          //1 bit - Activa precise event based sampl.
};

/* Otras declaraciones de datos y funciones */
/* Funcion que convierte una estructura p4pc_event_count_config en el
buffer p4pc_write_msg.  Codigo en p4pc_low_level_lib.c */
void p4pc_ecc_to_wm(struct p4pc_event_count_config *ecc, char *wm);
/* Funciones que modifican el bit "enable" del cccr en el mensaje a
write().  Codigo en p4pc_low_level_lib.c */
void deactivate_counter(char *);
void activate_counter(char *);
```

```
/* Grupo de funciones que componen la API de nuestra aplicacion.  Incluye
la funcion que instala en /dev los 18 dispositivos para leer cada contador,
las funciones que inician, leen y paran los mismos, y funciones para
inicializar las estructuras de datos necesarias.  Codigo y mas informacion
en p4pc_api.c */
int install_p4pc_devices(void);
int init_counter(int);
__s64 get_counter(int);
__s64 stop_counter(int);
void p4pc_set_event_id(struct p4pc_event_id *, __u8, __u16, __u32, __u32,
      __u32, __u8);
void p4pc_set_event_config(struct p4pc_event_count_config *,
                                          struct p4pc_event_id *,
                                          struct p4pc_usr_os_t *,
                                          struct p4pc_tag *,
                                          struct p4pc_filter *,
                                          struct p4pc_ovf *,
                                          _Bool);
void p4pc_set_BASIC_event_config(struct p4pc_event_count_config *,
                                          struct p4pc_event_id *);
void select_event(struct p4pc_event_count_config *);

#endif      /* P4PC_EVENT_HDR */
```

## C.3 p4pc_event_catalog.h

```
/*************************************************************************/
/*Fichero:  p4pc_event_catalog                                           */
/*Autor:    Agustin Isasa Cuartero - UOC TFC GNU/LINUX                   */
/*Resumen:  header para almacenar eventos predifinidos que puedan ser    */
/*          utilizados inmediatamente por una aplicacion de usuario.     */
/*          Hace uso del fichero p4pc_event.h, donde se definen las      */
/*          estructuras de datos necesarias para la identificacion y     */
/*          gestion de los eventos.                                      */
/*Interface:                                                             */
/*Notas:                                                                 */
/*************************************************************************/

#ifndef P4PC_EVENT_CATALOG_HDR
#define P4PC_EVENT_CATALOG_HDR

#include "p4pc_event.h"

/* Algunas definiciones previas */
/*
 * Las siguientes configuraciones no son especificas del evento en
 * estudio, sino que pueden ser aplicadas a cualquiera.  Se trata de
 * configuraciones basicas del resto de estructuras, que suponen:
 * - Activar recuento tanto con threads de usuario como de SO
 * - No etiquetado de uops
 * - No activar el filtrado de resultados
 * - No gestionar las interrupciones por overflows
 * - No activar muestreo de eventos preciso
 */

static struct p4pc_usr_os_t UsrOsNoHT = {
      FALSE,      /* Sin uso en ausencia de HT */
      FALSE,      /* Sin uso en ausencia de HT */
      TRUE,       /* Activamos hilo de usuario */
      TRUE,       /* Activamos hilo de SO */
      0x3         /* 11 en ausencia de HT */
};

static struct p4pc_tag NoTag = {
      FALSE,      /* Sin tagging */
      0x0         /* Valor 0 en etiqueta */
};

static struct p4pc_filter NoFilter = {
      FALSE,      /* Sin filtrado */
      FALSE,      /* Complement a 0 */
      0x0,        /* Umbral a 0 */
      FALSE       /* Sin deteccion de flanco */
};

static struct p4pc_ovf NoOvf = {
      FALSE,      /* Sin overflow en cada incremento */
      FALSE,      /* Sin interrupcion cuando ovf hilo 0 con HT */
      FALSE,      /* Sin interrupcion cuando ovf hilo 1 con HT */
      0x00000,    /* Cuenta atras a 0 */
      FALSE       /* Sin recuento en cascada */
};

/*
 * De todos los posibles eventos a analizar, en este trabajo hemos
```

```
 * elegido utilizar la metrica "2nd Level Cache Read Misses",
 * perteneciente al grupo de eventos BSQ_cache_reference, siguiendo el
 * "IA-32 Intel Architecture Optimization Reference Manual".  El unico
 * bit activo es el 8:  RD_2ndL_MISS.  No obstante, definimos los
 * diferentes bits (mascara de eventos) que se utilizan en este grupo de
 * eventos e implementamos un segundo ejemplo "2nd Level Cache Read
 * References".  Tanto la definicion de los bits como los ejemplos deben
 * aclarar el camino a seguir para la implementacin de eventos
 * pertenecientes a cualquier grupo.
 */

/* Grupo de eventos BSQ_cache_reference */
#define BSQ_cache_reference 0x0C
#define RD_2ndL_HITS          0x001 /* 000 0000 0001 */
#define RD_2ndL_HITE          0x002 /* 000 0000 0010 */
#define RD_2ndL_HITM          0x004 /* 000 0000 0100 */
#define RD_3rdL_HITS          0x008 /* 000 0000 1000 */
#define RD_3rdL_HITE          0x010 /* 000 0001 0000 */
#define RD_3rdL_HITM          0x020 /* 000 0010 0000 */
#define WR_2ndL_HIT           0x040 /* 000 0100 0000 */
#define RD_2ndL_MISS          0x100 /* 001 0000 0000 */
#define RD_3rdL_MISS          0x200 /* 010 0000 0000 */
#define WR_2ndL_MISS          0x400 /* 100 0000 0000 */

/* Definimos la identificacion del evento para fallos read de L2 */
static struct p4pc_event_id _2LevelCacheReadMisses_id = {
      BSQ_cache_reference,    /* 0x0C */
      RD_2ndL_MISS,           /* 0x100 = 001 0000 0000b */
      MSR_BSU_ESCR0,          /* 0x03A0 */
      MSR_BPU_COUNTER0,       /* 0x0300 */
      MSR_BPU_CCCR0,          /* 0x0360 */
      BSU                     /* 0x07 */
};

/* Resumiremos en un solo objeto "_2LevelCacheReadMisses" del tipo struct
 * p4pc_event_count_config toda la informacion necesaria para configurar
 * el hardware de tal modo que consigamos en el correspondiente contador
 * el resultado del recuento de todos los fallos de acceso a la cache L2.
 * Esta estructura la inicializamos en nuestro trabajo, con su
 * configuracion mas basica, en p4pc_TFC_api.h.
 */
struct p4pc_event_count_config _2LevelCacheReadMisses;


/* Otro ejemplo de identificacion del mismo grupo:  referencias read a
L2... */
static struct p4pc_event_id _2LevelCacheReadReferences_id = {
      BSQ_cache_reference,
      RD_2ndL_HITS | RD_2ndL_HITE | RD_2ndL_HITM | RD_2ndL_MISS,
      MSR_BSU_ESCR1,
      MSR_BPU_COUNTER2,
      MSR_BPU_CCCR2,
      BSU
};

/* ...que, al igual que el anterior evento, resumimos en un solo objeto */
struct p4pc_event_count_config _2LevelCacheReadReferences;

/*
 * Damos a continuacion otro ejemplo de grupo de eventos:
 * retired_mispred_branch_type.  De igual manera a lo hecho
```

```
 * anteriormente, definimos primero los flags identificativos de cada uno
 * de los eventos individuales identificables en la documentacion, para
 * posteriormente instanciar la identificacion del evento "saltos
 * condicionales mal predichos".
 */
#define retired_mispred_branch_type 0x05
#define CONDITIONAL                 0x002 /* 000 0000 0010 */
#define CALL                        0x004 /* 000 0000 0100 */
#define RETURN                      0x008 /* 000 0000 1000 */
#define INDIRECT                    0x010 /* 000 0001 0000 */

/* Un evento de este grupo: saltos condicionales mal predichos */
static struct p4pc_event_id mispredicted_conditionals = {
      retired_mispred_branch_type,
      CONDITIONAL,
      MSR_TBPU_ESCR0,
      MSR_MS_COUNTER0,
      MSR_MS_CCCR0,
      TBPU
};

#endif      /* P4PC_EVENT_CATALOG_HDR */
```

## C.4 p4pc_low_level_lib.c

```
/*************************************************************************/
/*Fichero:  p4pc_low_level_library.c                                     */
/*Autor:    Agustin Isasa Cuartero - UOC TFC GNU/LINUX                   */
/*Resumen:  codigo que implementa la funcion que convierte las           */
/*          estructuras de datos que definen el evento a estudiar, en    */
/*          las de bits con las que configurar los correspondientes      */
/*          registros del procesador, salvando los vectores de bits      */
/*          obtenidos en el buffer correspondiente, asi como funciones   */
/*          auxiliares.                                                  */
/*Interface:    Todas las funciones:  Parametros:                        */
/*          *ecc:  puntero a struct del tipo p4pc_event_count_config     */
/*          que representa a alto nivel la informacion necesaria para    */
/*          describir con precision un evento (ver p4pc_event.h)         */
/*          *wm:  puntero a una cadena que representa la misma           */
/*          que el anterior parametro, pero a nivel de bits cuando se    */
/*          trata de la informacion a pasar a los correspondientes       */
/*          registros.                                                   */
/*Notas:                                                                 */
/*************************************************************************/

#include <sys/types.h>        /* mknod() */
#include <sys/stat.h>         /* mknod() */
#include <linux/kdev_t.h>     /* mknod() */
#include "p4pc_event.h"

/* Funcion que convierte una estructura p4pc_event_count_config en el
buffer p4pc_write_msg */
void p4pc_ecc_to_wm(struct p4pc_event_count_config *ecc, char *wm)
{
      memset(wm, 0, P4PC_MAXBUF);   /* Inicializamos a ceros */
      /* comienza escr */
      wm[0] = ecc->usr_os_t->t0_usr << 2 | ecc->usr_os_t->t0_os << 3 |\
            ecc->tag->tag_enable << 4 | ecc->tag->tag_value << 5;
      wm[1] = ecc->tag->tag_value >> 3 | ecc->event_id->event_mask << 1;
      wm[2] = ecc->event_id->event_mask >> 7;
      wm[3] = ecc->event_id->event_mask >> 15 | ecc->event_id-event_group
            << 1;
      /* fin escr y comienza cccr */
      /* wm[4] a ceros */
      wm[5] = TRUE << 4 | ecc->event_id->cccr_escr <<5;//cccr enable+escr
      wm[6] = ecc->usr_os_t->active_thread | ecc->filter->compare << 2 |\
            ecc->filter->complement << 3 | ecc->filter->threshold << 4;
      wm[7] = ecc->filter->edge | ecc->ovf->force_ovf << 1 |\
            ecc->ovf->ovf_pmi_t0 << 2 | ecc->ovf->ovf_pmi_t1 << 3 |\
            ecc->ovf->cascade << 6;
      /* fin cccr y comienza resto info:  escr, cccr, counter, account,
      pebs */
      wm[8] = ecc->event_id->escr;
      wm[9] = ecc->event_id->escr >> 8;
      wm[10] = ecc->event_id->cccr;
      wm[11] = ecc->event_id->cccr >> 8;
      wm[12] = ecc->event_id->counter;
      wm[13] = ecc->event_id->counter >> 8;
      if(ecc->ovf->account < 0) /* Pasa al driver account positiva... */
            ecc->ovf->account = ecc->ovf->account * (-1);
      if(ecc->ovf->account > 0x000000fffffffff)
            ecc->ovf->account = 0x000000fffffffff; /* ...max. 40 bits */
      wm[14] = ecc->ovf->account;
      wm[15] = ecc->ovf->account >> 8;
```

```
        wm[16] = ecc->ovf->account >> 16;
        wm[17] = ecc->ovf->account >> 24;
        wm[18] = ecc->ovf->account >> 32;
        wm[19] = ecc->pebs_on;
}

/* Funciones que modifican el mensaje a pasar al driver haciendo que el bit
"enable" del cccr vaya a 0 o a 1 */
void deactivate_counter(char *wm)
{
        wm[5] = wm[5] & 0xef;   /* 1110 1111 */
}

void activate_counter(char *wm)
{
        wm[5] = wm[5] | 0x10;   /* 0001 0000 */
}
```

## C.5 p4pc_api.c

```
/*************************************************************************/
/*Fichero:  p4pc_api.c                                                   */
/*Autor:    Agustin Isasa Cuartero - UOC TFC GNU/LINUX                   */
/*Resumen:  Implementacion de la API de nuestra aplicacion.              */
/*Interface:                                                             */
/*Notas:                                                                 */
/*************************************************************************/

#include <stdio.h>
#include <fcntl.h>
#include <sys/types.h>        /* mknod() */
#include <sys/stat.h>         /* mknod() */
#include <linux/kdev_t.h>     /* mknod() */
#include <linux/kdev_t.h>     /* MKDEV() */
#include <errno.h>            /* EEXIST */
#include "p4pc_event_catalog.h"

/*
 * Funcion que instala en /dev los 18 dispositivos (uno por contador) que
 * nos permitiran acceder a los contadores.  Si mknod() devuelve error,
 * se analiza si corresponde a EEXIST, es decir, que el dispositivo ya
 * existe, en cuyo caso continuamos sin incidencias.
 */
int install_p4pc_devices(void)
{
      int i;
      for(i=0; i< MAX_COUNTERS; i++){
            if((mknod(p4pc_dev_names[i], S_IFCHR | 0664, MKDEV(P4PC_MAJOR,
            i)) < 0) && (errno != EEXIST)){
                  return -P4PC_EMKNOD;
            }
      }
      return 0;
}

/*
 * Dada una configuracion de evento determinada, que incluye un contador
 * que registre sus ocurrencias, esta funcion traduce (mediante la
 * funcion de bajo nivel p4pc_ecc_to_wm()) las estructuras de datos de
 * alto nivel a un mensaje formado por vectores de bits y otras
 * informaciones necesarias para la configuracion de los correspondientes
 * MSRs, guardando dicho mensaje en la correspondiente posicion de un
 * vector que recoge todos los posibles mensajes vigentes, cada uno de
 * ellos correspondiente a un contador.  Para ello:
 *
 * 1 - Calcula la entrada en el array p4pc_write_msg_array[] haciendo
 *     (direccion contador) modulo 18.
 * 2 - Llama a la funcion p4pc_ecc_to_wm() pasandole el puntero a la
 *     estructura de alto nivel y el puntero al buffer donde almacenar
 *     el mensaje.
 */
void select_event(struct p4pc_event_count_config *e_config)
{
      int counter;
      counter = e_config->event_id->counter % MAX_COUNTERS;
      p4pc_ecc_to_wm(e_config, p4pc_write_msg_array[counter]);
}

/*
```

```
 * init_counter() se ocupa de de pasar al driver, mediante un write() al
 * correspondiente dispositivo, el mensaje que contiene la configuracion
 * del contador a inicializar.  Para ello:
 *
 * 1 - Comprueba que existe realmente mensaje a enviar al driver.
 * 2 - Efectua un write() tomando como buffer de origen el elemento del
 *     array de mensajes que corresponde al contador pasado como
 *     parametro.
 */
int init_counter(int counter)
{
      if(!p4pc_write_msg_array[counter]){
            return -P4PC_ENOMSG;
      }
      if((fd[counter] = open(p4pc_dev_names[counter], O_RDWR)) < 0){
            return -P4PC_EOPEN;
      }
      if(write(fd[counter], p4pc_write_msg_array[counter], P4PC_MAXBUF)
            != P4PC_MAXBUF){
            return -P4PC_EWRITE;
      }
      return 0;
}

/*
 * Esta funcion permite leer el contenido del contador que se le pasa
 * como parametro.  Devuelve, precisamente, el valor de la lectura.
 * Comprueba previamente si existe el descriptor del dispositivo.
 */
__s64 get_counter(int counter)
{
      __s64 result;

      if(!fd[counter]){
            return -P4PC_ENODESCR;
      }
      if(read(fd[counter], (char *)&result, sizeof(result)) !=
            sizeof(result)){
            return -P4PC_EREAD;
      }
      return result;
}

/*
 * Con stop_counter() enviamos un mensaje que contiene a 0 el bit
 * "enable" del corresponiente cccr, lo que detiene el proceso de
 * recuento.  Para ello,
 *
 * 1 - Comprueba que existe el correspondiente descriptor.
 * 2 - Llama a la funcion de bajo nivel deactivate_counter(), que se
 *     encarga de desactivar el bit "enable" del cccr del contador pasado
 *     como parametro, desactivacion que se realiza en el correspondiente
 *     mensaje a pasar al driver.
 * 3 - Toma una ultima lectura del contenido del contador a parar,
 *     lectura que sera el valor devuelto por la funcion.
 * 4 - Escribe el mensaje al driver.
 * 5 - Cierra el dispositivo.
 * 6 - Anula la entrada en el array de descriptores.
 * 7 - Llama a activate_counter() para dejar el mensaje en el mismo
 *     estado en el que se encontraba.
 * 8 - Devuelve la lectura efectuada.
```

```
 */
__s64 stop_counter(int counter)
{
      __s64 result;

      if(!fd[counter]){
            return -P4PC_ENODESCR;
      }

      deactivate_counter(p4pc_write_msg_array[counter]);

      if(read(fd[counter], (char *)&result, sizeof(result)) !=
            sizeof(result)){
            return -P4PC_EREAD;
      }
      if(write(fd[counter], p4pc_write_msg_array[counter], P4PC_MAXBUF)
            != P4PC_MAXBUF){
            return -P4PC_EWRITE;
      }
      close(fd[counter]);
      fd[counter] = 0;
      activate_counter(p4pc_write_msg_array[counter]);
      return result;
}

/*
 * Tras las funciones que constituyen la interface principal usuario/
 * driver, definimos a continuacion otras auxiliares que facilitan la
 * inicializacion de las estructuras de datos de alto nivel que definen
 * los eventos a muestrear y la forma de realizar el recuento.
 *
 * p4pc_set_event_id() inicializa una estructura del tipo p4pc_event_id,
 * es decir, la estructura que identifica el evento a analizar.  Se le
 * pasan como parametros los correspondientes registros.
 */
void p4pc_set_event_id(struct p4pc_event_id *e_id, __u8 group, __u16
      mask, __u32 escr, __u32 counter, __u32 cccr, __u8 cccr_escr)
{
      e_id->event_group = group;
      e_id->event_mask = mask;
      e_id->escr = escr;
      e_id->counter = counter;
      e_id->cccr = cccr;
      e_id->cccr_escr = cccr_escr;
}

/* Podrian venir aqui "setters" para todas las estructuras de datos
relevantes definidas en p4pc_event.h y que componen la struct
p4pc_event_count_config , pero las obviamos porque utilizaremos sus
configuraciones basicas */

/*
 * La siguiente funcion inicializa una estructura del tipo
 * p4pc_event_count_config, es decir, un objeto que recoge toda la
 * informacion necesaria para conocer que evento analizar y como hacerlo.
 * Los correspondientes registros son pasados como parametros.
 */
void p4pc_set_event_config(struct p4pc_event_count_config *e_config,
                           struct p4pc_event_id *e_id,
                           struct p4pc_usr_os_t *u_o_t,
                           struct p4pc_tag *tag,
```

```
                                        struct p4pc_filter *filter,
                                        struct p4pc_ovf *ovf,
                                        _Bool pebs)
{
      e_config->event_id = e_id;
      e_config->usr_os_t = u_o_t;
      e_config->tag = tag;
      e_config->filter = filter;
      e_config->ovf = ovf;
      e_config->pebs_on = pebs;
}

/*
 * Definimos, por ultimo, una funcion auxiliar que nos resultara muy
 * util, y que permite configurar un objeto del tipo
 * p4pc_event_count_config para que el recuento del evento que representa
 * sea efectuado en su configuracion mas basica, es decir:
 *
 * 1 - Recuento tanto de hilos de usuario como de SO, y sin tecnologia
 *     Hyper-Threading.
 * 2 - Sin etiquetado de uops.
 * 3 - Sin filtrado.
 * 4 - Sin tratamiento de desbordamientos.
 * 5 - Sin muestreo preciso.
 */
void p4pc_set_BASIC_event_config(struct p4pc_event_count_config
      *e_config, struct p4pc_event_id *e_id)
{
      e_config->event_id = e_id;
      e_config->usr_os_t = &UsrOsNoHT;
      e_config->tag = &NoTag;
      e_config->filter = &NoFilter;
      e_config->ovf = &NoOvf;
      e_config->pebs_on = FALSE;
}
```

## C.6 p4pc_TFC_api.h

```
/*************************************************************************/
/*Fichero:  p4pc_TFC_api.h                                               */
/*Autor:    Agustin Isasa Cuartero - UOC TFC GNU/LINUX                   */
/*Resumen:  header para declaracion de macros que envuelven el evento    */
/*          en  estudio en el TFC, asi como un ejemplo adicional de      */
/*          evento                                                       */
*Interface:                                                              */
/*Notas:    El codigo de las funciones utilizadas esta en p4pc_api.c     */
/*************************************************************************/

#ifndef P4PC_TFC_API_HDR
#define P4PC_TFC_API_HDR

#include "p4pc_event_catalog.h"

/* Macro que:
 *
 * 1 - Resume en un solo objeto "_2LevelCacheReadMisses" del tipo struct
 *     p4pc_event_count_config toda la informacion necesaria para
 *     configurar el hardware de tal modo que consigamos en el
 *     correspondiente contador el resultado del recuento de los fallos
 *     de acceso a la cache L2, con un contexto basico en el contador y
 *     que se resume en los siguientes valores pasados a los miembros:
 *
 *     &_2LevelCacheReadMisses_id, // Identific. evento a muestrear...
 *     &UsrOsNoHT,                 // ...en todos hilos (sin HT),...
 *     &NoTag,                     // ...sin etiquetado de uops,...
 *     &NoFilter,                  // ...sin filtrado,...
 *     &NoOvf,                  // ...sin gestion de desbordamientos y...
 *     FALSE                       // ...sin PEBS
 * 2 - Aplicamos por ultimo la funcion que permite obtener los vectores
 *     de bits y otra informacion necesaria y que almacenaremos en el
 *     correspondiente mensaje a pasar mas tarde, mediante
 *     init_counter(), al driver.
 */
#define SELECT_2LEVEL_CACHE_READ_MISSES                                 \
      do {                                                              \
            p4pc_set_BASIC_event_config(&_2LevelCacheReadMisses, \
                  &_2LevelCacheReadMisses_id);                          \
            select_event(&_2LevelCacheReadMisses);                      \
            } while(0)

/* Un ejemplo adicional:  evento "referencias de lectura a cache 2L" */
#define SELECT_2LEVEL_CACHE_READ_REFERENCES                             \
      do {                                                              \
            p4pc_set_BASIC_event_config(&_2LevelCacheReadReferences,\
                  &_2LevelCacheReadReferences_id);                      \
            select_event(&_2LevelCacheReadReferences);                  \
            } while(0)

#endif      /* P4PC_TFC_API_HDR */
```