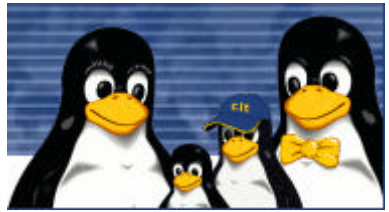
Taller de Tiempo Real para Control Robótico

+Ejem... Plos

Nelson Acosta – nacosta@exa.unicen.edu.ar
INCA / INTIA-Fac. Cs. Exactas - UNCPBA
Tandil - Argentina

Tabla de Contenido

MakeFile clásico



Make File: …

```
CFLAGS = -D__KERNEL__ \
       -DMODULE \
       -O -Wall

all:  hello.o

hello.o:
       gcc $(CFLAGS) -c hello.c

clean:
       rm -f *.o *~ core

nodo:
       mknod  /dev/pap  c  111  0

app:
       gcc app.c –lncurses -o app
```

PARAMETROS

&

Código de Autor:

Código Ejemplo

PARAMETROS: Implementación típica (1)

```
/*****************************************************
 * - Pasaje de parametros asignados en tiempo de instalacion
 *   por medio de insmod o modprobe.
 * - Lectura de datos del modulo que se esta ejecutando, por
 *   medio del uso de la estructura: task_struct. Detalles de
 *   la estructura 'current' pueden verse en <asm/current.h>.
 * - Definicion de datos del modulo: autor, descripción y
 *   datos del dispositivo. Esta info puede verse por medio
 *   de alguna herramienta de dump, tal como 'objdump'.
 * Ejecutar con el comando:
 *   insmod -f hello.o
 *   insmod -f hello.o nro=2333 str="esta es la string"
 *****************************************************/
#include <linux/module.h>
#include <linux/sched.h>

/*--- Datos para que queden en el Objeto (compilado) ---*/
#define MODULE_AUTHOR (Nelson Acosta)
#define MODULE_DESCRIPTION (Prueba de cosas del kernel)
#define MODULE_SUPPORTED_DEVICE (cualquier cosa)
```

PARAMETROS: Implementación típica (2)

```
/* Pasaje de parametros al modulo desde el INSMOD */
int nro=0;   /* cantidad entera a imprimir */
char *str;   /* cadena a imprimir nro veces */
MODULE_PARM (nro, "i");
MODULE_PARM (str, "s");

int init_module( void ){
      printk("<1>Hello, world\nProceso (%s) PID:%i\n",
                              current->comm, current->pid );
      printk(“Los parámetros son (%s)=(%i)\n\n", str, nro );
      return( 0 );
}

void cleanup_module( void ){
      printk("<1>Goodbye cruel world (%s|%i)\n\n",
                              current->comm, current->pid );
      printk(”Los parámetros son (%s)=(%i)\n\n", str, nro );
}
```

Zonas de I/O:

Reserva de zona

Zonas de IO: Implementación típica (0)

FUNCIONES para IO:

- PORT : especifica el puerto donde se trabajará.
- RANGE : la cantidad de direcciones.
- "**hello**": especifica el nombre del módulo.

```
err = check_region( PORT, RANGE );
```

Verifica si una región de IO está ocupada.

```
SI check_region retorna menor que cero ENTONCES
        este rango está ocupado;
```

```
request_region( PORT, RANGE, "hello" );
```

Reserva una región de IO que no esté ocupada.

```
release_region( PORT, RANGE );
```

Libera una región de IO antes reservada.

Zonas de IO: Implementación típica (1)

```
/*************************************************
 *   - Se verifica mirando en el archivo /proc/ioports
 *     para ver si realmente esta registrando el área o
 *     no.
 *   - Chequeo de zonas del bus de IO:  check_region.
 *   - Reserva de zonas del bus de IO:  request_region.
 *   - Liberado de zonas del bus de IO: release_region.
 * Ejecutar con el comando:
 *       insmod -f hello.o
 *       rmmod hello
 *************************************************/

#include <linux/module.h>
#include <linux/sched.h>
#include <linux/ioport.h>
```

Zonas de IO: Implementación típica (2)

```
#define PORT   0x378
#define RANGE  3

int init_module( void ){
        int err;

        printk("Hello, world\n”);
        /* reserva del espacio de direcciones de IO */
        err = check_region( PORT, RANGE );
        if( err < 0 ) return err;    /* esta OCUPADO */
        request_region( PORT, RANGE, "hello" );
        printk("Region(%i,%i,hello)%i\n", PORT, RANGE, err);
        return( 0 );
}

void cleanup_module( void ){
        printk("Goodbye cruel world”);
        release_region( PORT, RANGE );
}
```

Registro de Dispositivo:

Código Ejemplo

Registro de Dispositivo (0)

FUNCIONES de registro de dispositivo:

- PORT : especifica el puerto donde se trabajará.
- RANGE : la cantidad de direcciones.
- "**scull**": especifica el nombre del módulo.
- Scull_fops : estructura de funciones exportadas.
- SCULL_MAYOR: major del archivo de dispositivo.

```
err=register_chrdev(SCULL_MAYOR, "scull",
                                        &scull_fops);
```

Intenta registrar un dispositivo de caracter.

```
SI register_chrdev retorna menor que cero ENTONCES
        Imposible acceder al SCULL_MAYOR;
```

```
unregister_chrdev( SCULL_MAYOR, "scull");
```

Desregistra un dispositivo de carácter ya registrado.

Registro de Dispositivo (1)

```
/*****************************************************
 * (Des)Registro de drivers de caracteres usando la
 * funcion: register_chrdev / unregister_chrdev
 * Ejecutar: insmod -f hello.o / rmmod hello
 * cat /proc/devices: muestra todos los dispositivos, en
 *           este caso, es un disp. de caracter con el
 *           MAJOR=253 y el nombre SCULL.
 * cat /proc/ksyms|grep "scull": muestra las funciones
 *           que esta exportando el modulo HELLO.
 * SOLO DEBE MOSTRARSE MIENTRAS EL MODULO ESTE INSTALADO
 *****************************************************/
#include <linux/module.h>
#include <linux/sched.h>
#include <linux/ioport.h>
#include <linux/fs.h>

#define PORT   0x378
#define RANGE  3
#define SCULL_MAJOR 253
```

Registro de Dispositivo (2)

```
int scull_open(struct inode *inode, struct file *filp){
            return 0;  /* success */
}
int scull_release(struct inode *inode, struct file *filp){
            return 0;
}
int scull_read(struct file *filp, char *buf,
                                size_t count, loff_t *f_pos){
          return 0;
}
int scull_write(struct file *filp, const char *buf,
                                size_t count, loff_t *f_pos){
            return 0;
}

struct file_operations scull_fops = {
           read: scull_read,
          write: scull_write,
           open: scull_open,
        release: scull_release    };
```

Registro de Dispositivo (3)

```
int init_module( void ){
        int err;

        printk("Hello, world\n”);
        /* reserva del espacio de direcciones de IO */
        err = check_region( PORT, RANGE );
        if( err < 0 ) return err;    /* esta OCUPADO */
        request_region( PORT, RANGE, "hello" );
        printk("Reservado la region(%i,%i,hello) %i\n",
                                PORT, RANGE, err);

        /* Registro del driver */
        err=register_chrdev(SCULL_MAJOR, "scull", &scull_fops);
        if( err < 0 ){
           printk("Scull sin acceso al MAJOR %d.\n",
                                                SCULL_MAJOR );
           return err;
        }
        return( 0 );
}
```

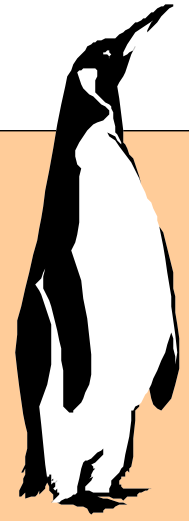
Registro de Dispositivo (4)

```
void cleanup_module( void ){
        printk("Goodbye cruel world.”);
        /* Des registro del driver */
        unregister_chrdev( SCULL_MAYOR, "scull");
        /* libera la region de IO */
        release_region( PORT, RANGE );
}
```

BORRA archivo especial:

```
#!/bin/sh
# Elimina el nodo del dispositivo con RM. Debe ser
# ejecutado como superusuario.
rm  /dev/skull0
```

Registro de Dispositivo (5)



CREA archivo especial:

```
#!/bin/sh
# Crea el nodo del dispositivo con MKNOD. Debe ser
# ejecutado como superusuario. Este comando tiene
# cuatro operandos:
#    /dev/DISPOSITIVO
#    c Para dispositivos de tipo caracter
#    253 es el numero MAYOR
#    0 es el numero MENOR
mknod  /dev/scull0  c  253  0


# Se asigna el grupo y permisos. Los grupos se pueden
# ver en: /etc/group. Normalmente se debe poner en
# el grupo STAFF o WHEEL.
chgrp  wheel  /dev/scull0
chmod  664    /dev/scull0
```

Flujo del Tiempo:

Código Ejemplo

Flow of time: Implementación típica (0)

MEDICION DE INTERVALOS CORTOS DE TIEMPO EN EL KERNEL:

- **HZ**, la frecuencia del mother.
- **JIFFIES**, la cantidad de ticks del mother.
- **TSC** (TimeStamp Counter), reg. interno del procesador.

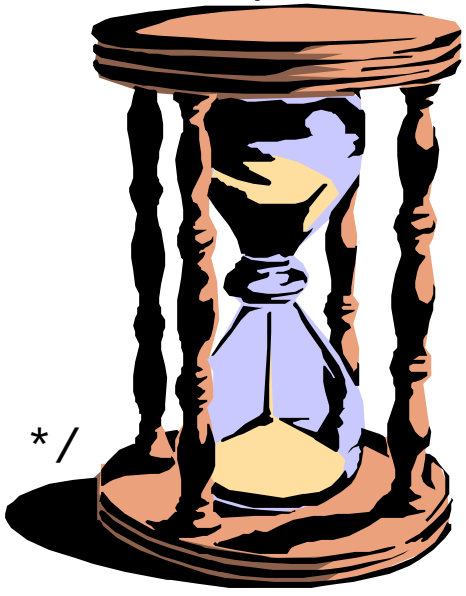
```
udelay( CANTIDAD_EN_MICROSEGUNDOS );
```
Realiza demoras en microsegundos (1000=1 milisegundo).

```
mdelay( CANTIDAD_EN_MILISEGUNDOS );
```
Realiza demoras en milisegundos.

```
rdtscl( REGISTRO_EN_LONGINT );
rdtsc( LOW_REGISTRO_WORD, HIGH_REGISTRO_WORD );
```
Lee registro TSC del procesador (long o dos words).

Flow of time: Implementación típica (1)

```
/*************************************************
 * MEDICION DE INTERVALOS CORTOS DE TIEMPO EN EL KERNEL:
 * - HZ, la frecuencia del mother
 * - JIFFIES, la cantidad de ticks del mother
 * - TSC (TimeStamp Counter), reg. interno del procesador
 * - MSR (Machine Specific Register)
 * - get_cycles
 * Ejecutar: insmod -f hello.o / rmmod hello
 *************************************************/

#include <linux/module.h>
#include <linux/sched.h>
#include <linux/ioport.h>

/* La constante HZ que esta en param.h */
#include <linux/param.h>
/* La variable JIFFIES que esta en sched.h */
#include <linux/sched.h>
```

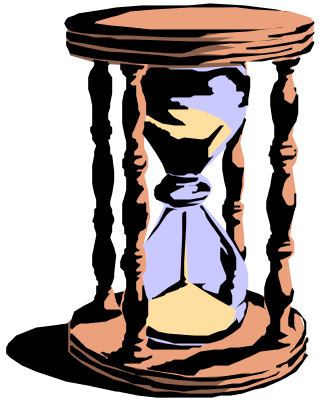


Flow of time: Implementación típica (2)

```
/* MSR para leer TSC (timestamp counter), que cuenta
 * los pulsos a la frec del reloj del motherboard */
#include <asm/msr.h>

/* La funcion GET_CYCLES de timex accede al TSC */
#include <linux/timex.h>

/* time.h -> declarac.de timeval y de do_gettimeofday */
#include <linux/time.h>

/* delay.h necesario para usar udelay y mdelay */
#include <linux/delay.h>
```

```
#define PORT   0x378
#define RANGE  3


struct  timeval  tv;
```

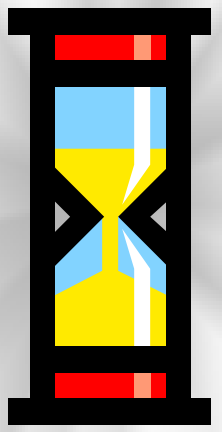
Flow of time: Implementación típica (3)



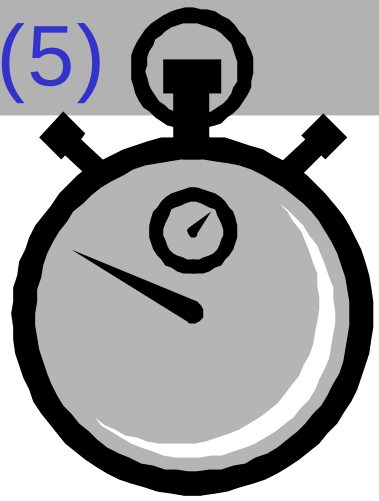
```
int init_module( void ){
        int err;
        unsigned long ini, end;
        unsigned int hh, ll;
        cycles_t  ciclos;
        printk("Hello, world.\n”);
        /* udelay y mdelay */
        printk("uDelay de hasta 1000 microseg (1ms)...");
        udelay( 1000 ); udelay( 1000 ); udelay( 1000 );
        printk("pasaron los TRES...");
        mdelay( 3 );
        printk("y 3 más.\n");
        /* Muestro valores HZ y JIFFIES del sistema */
        rdtscl( ini );
        printk("HZ=%i (%lu)\n", HZ, jiffies );
        rdtscl( end );
        printk("tiempo PrintK es de:%li.\n", end - ini);
        rdtsc( ll, hh );
        ciclos = get_cycles();
        printk("ciclos=%lu/lo=%u/hi=%u\n", ciclos, ll, hh);
```

Flow of time: Implementación típica (4)

```
        /*--- reserva del espacio de direcciones de IO ---*/
        err = check_region( PORT, RANGE );
        if( err < 0 ) return err;    /* esta OCUPADO */
        request_region( PORT, RANGE, "hello" );
        printk("Region(%i,%i,hello)%i\n",PORT, RANGE, err);
        return( 0 );
}

void cleanup_module( void ){
        printk("Goodbye cruel world.\n”);
        /* libera la region de IO */
        release_region( PORT, RANGE );
}
```



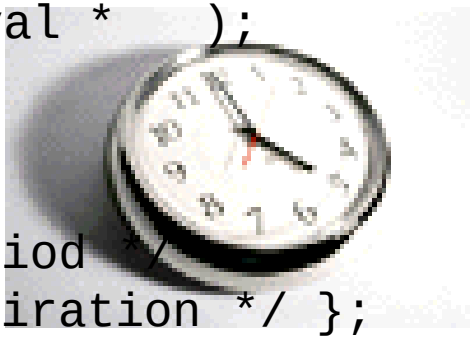
Flow of time: <time.h> (5)



```
struct  timespec {
   time_t tv_sec;    /* seconds */
   long   tv_nsec;   /* nanoseconds */   };


timespec_to_jiffies(  struct timespec *value  )

jiffies_to_timespec(  unsigned long jiffies,
                      struct timespec *value  )


struct  timeval {
   time_t       tv_sec;   /* seconds */
   suseconds_t  tv_usec;  /* microseconds */   };


struct  timezone {
       int tz_minuteswest; /* min west of Greenwich */
       int tz_dsttime;     /* type of dst correction */  };
```

Flow of time: <time.h> (6)

```
extern void do_gettimeofday(     struct timeval *tv );

extern void do_settimeofday(     struct timeval *tv );

extern void get_fast_time(       struct timeval *tv );

extern void (*do_get_fast_time)( struct timeval *    );
```



```
struct  itimerspec {
   struct timespec it_interval;  /* timer period */
   struct timespec it_value;     /* timer expiration */ };


struct  itimerval {
   struct timeval it_interval;    /* timer interval */
   struct timeval it_value;       /* current value */  };
```

Timers:

Código Ejemplo

Timers: Implementación típica (1)

TiMERs:

Permite encolar una tarea en el scheduler, con un tiempo de vencimiento, de tal forma que vencido dicho tiempo se ejecute automáticamente.

* **Instalación:**

```
init_timer( &mi_timer );
mi_timer.function = mi_tarea_rutina;
mi_timer.data = (unsigned long)&mi_tarea_datos;
mi_timer.expires = jiffies + HZ;    /* 1 seg */
add_timer( &mi_timer );   /* sleep por 1 seg */
```

* **Desinstalación:**

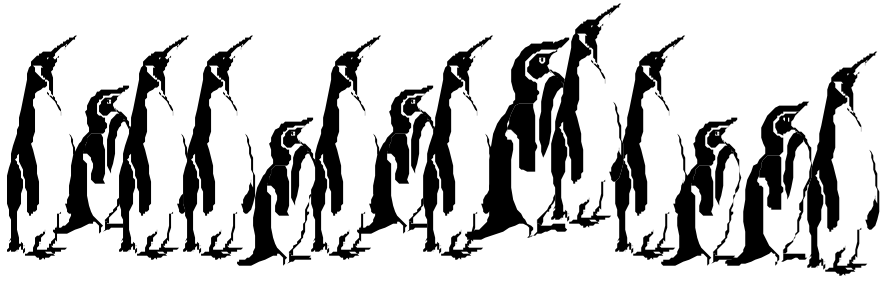
```
del_timer( &mi_timer );
```

* **Función a ejecutar:**

```
void mi_tarea( unsigned long ptr ){ ... }
```

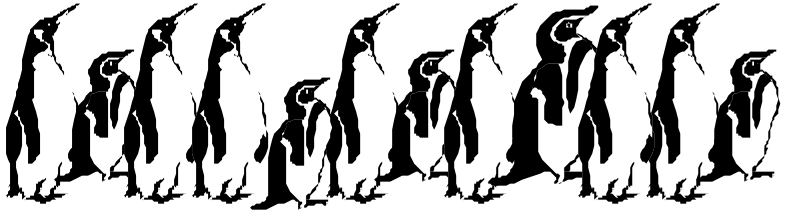
Timers: Implementación típica (2)

```
/*************************************************
 * USO DE TIMERS:
 *************************************************/
#include <linux/module.h>
#include <linux/sched.h>
#include <linux/ioport.h>
#include <linux/timer.h>

#define PORT   0x378
#define RANGE  3

struct  timer_list  mi_timer;

void mi_tarea( unsigned long ptr ){
       printk("\nEjecutando en mi TAREA x TIMER.\n");
}

char  mi_tarea_datos[255];
```



Timers: Implementación típica (3)

```
void cleanup_module( void ){
        del_timer( &mi_timer );
        printk("Goodbye cruel world.\n”);

}

int init_module( void ){
        int err;

        printk("Hello, world\n.”);
        init_timer( &mi_timer );
        mi_timer.function = mi_tarea;
        mi_timer.data = (unsigned long)&mi_tarea_datos;
        mi_timer.expires = jiffies + HZ;    /* 1 seg */
        add_timer( &mi_timer );   /* sleep por 1 seg */
        printk("SALE(timer) 1 segundo + tarde.\n");

        return( 0 );
}
```



Timers

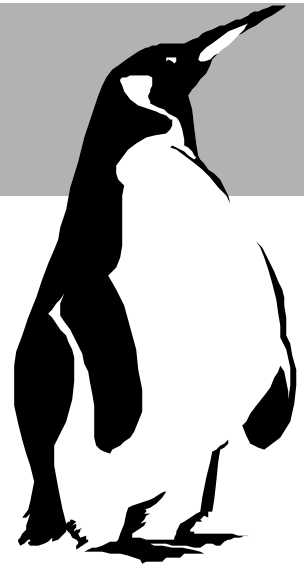
por

Colas:

Código Ejemplo

Timers x COLAs: Implementación típica (0)

COLAs:



Se utiliza la espera por un evento en una cola, de no producirse el evento, sale al vencerse el tiempo máximo asignado (timeout).

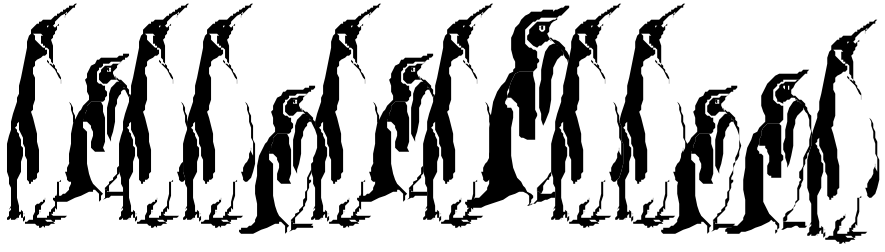
PRIMITIVAS:

```
sleep_on_timeout( &cola, HZ+HZ+HZ+HZ );

interruptible_sleep_on_timeout( &cola, HZ+HZ+HZ+HZ );
```

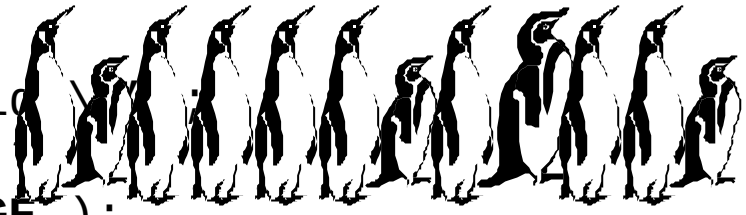
Usa el timeout de una cola para medir el tiempo,
HZ equivale a 4 segundos.

Timers x COLAs: Implementación típica (1)

```
/*************************************************
 * MEDICION DE TIEMPO POR COLAS:
 * - instalo una rutina para que se ejecute inmediatamente,
 *   mientras suspendo el proceso de instalación por 4 seg.
 * - uso de colas (de eventos con timeout) para los 4seg y
 *   timer para el segundo.
 * Ejecutar: insmod -f hello.o / rmmod hello
 *************************************************/
#include <linux/module.h>
#include <linux/sched.h>
#include <linux/ioport.h>
#include <linux/timer.h>

#define PORT   0x378
#define RANGE  3

DECLARE_WAIT_QUEUE_HEAD( cola );
```



Timers x COLAs: Implementación típica (3)



```
void cleanup_module( void ){
        printk("Goodbye cruel world \n");
        /* libera la region de IO */
        release_region( PORT, RANGE );
}
int init_module( void ){
        int err;

        printk("Hello, world\n.”);

        sleep_on_timeout( &cola, HZ+HZ+HZ+HZ );
        printk("SALE(cola) 4*HZ con (HZ=%u).\n", HZ );

        interruptible_sleep_on_timeout( &cola,HZ+HZ+HZ+HZ );
        printk("SALE(cola) 4*HZ con (HZ=%u).\n", HZ );

        err = check_region( PORT, RANGE );
        if( err < 0 ) return err;    /* esta OCUPADO */
        request_region( PORT, RANGE, "hello" );
        printk("Region(%i,%i,hello)%i\n", PORT,RANGE, err);
        return( 0 );
}
```

Interrupciones:

Código Ejemplo

Interrupciones: IRQ del mouse (0)

Manejo de Interrupciones:

- **INT_NRO**: Número de interrupción a atender.
- **atencion**: Función de atención de la interrupción.
- **SA_SHIRQ**: Interrupción compartida por varios manejadores.
- **"pruHello“**: Nombre del módulo de atención.

```
err_install = request_irq( INT_NRO, atencion, SA_SHIRQ,
                                        "pruHello", NULL );
```

Instala la rutina en la interrupción que corresponde.
Retorna 0 si todo está bien, caso contrario negativo.

```
free_irq( INT_NRO, NULL );
```

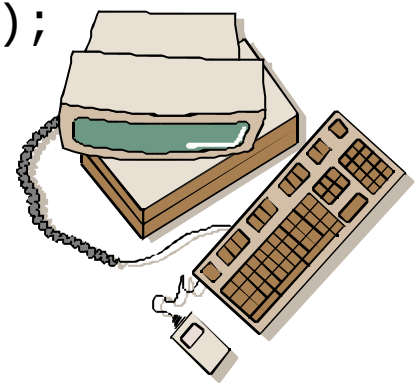
Desinstala la rutina de atención de interrupciones.

Interrupciones: IRQ del mouse (1)

```
/*************************************************
 *   - Instala una rutina de atención de interrupciones que
 *     se ejecuta (y cuenta las int) del MOUSE del notebook,
 *     en este caso la IRQ = 12.
 * Ejecutar: insmod -f hello.o / rmmod hello
 *************************************************/
#include <linux/module.h>
#include <linux/sched.h>

int nroInt = 0;  /* contador de interrupciones atendidas */

/* Rutina de atencion de interrupciones instalado */
void atencion(int irq, void *dev_id, struct pt_regs *regs){
            nroInt++;
            printk(" a %i",nroInt);
}

#define  INT_NRO  12 /* Numero de interrupcion */
int err_install=0;   /* para desinstalar si se instalo OK */
```

Interrupciones: IRQ del mouse (2)

```
int init_module( void ){
        printk("Hello, world.\n”);
        /* instala la rutina en la int que corresponde */
        err_install = request_irq( INT_NRO,
                                   atencion,
                                   SA_SHIRQ,
                                   "pruHello",
                                   NULL );
        printk( "request_IRQ => error(%i).", err_install );
        if( err_install==0 ) printk("OK.\n");
                else printk("FAIL.\n");
        return( 0 );
}

void cleanup_module( void ){
        printk("Goodbye cruel world.\n”);
        /* DESinstala la rutina de atencion de interrup */
        if(err_install==0){ free_irq( INT_NRO, NULL ); }
}
```



IOCTL:

Código Ejemplo

IOCTL: (1)

```
#ifndef __KERNEL__
#define __KERNEL__
#endif
#ifndef MODULE
#define MODULE
#endif
#define _NO_VERSION_
#include <linux/module.h>
#include <linux/sched.h>
#include <linux/kernel.h>
#include <linux/fs.h>
#include <linux/ioport.h>
#include <asm/io.h>
#include <linux/timer.h>
#include "tiempos.h"
#include "mascaras.h"
DECLARE_WAIT_QUEUE_HEAD(cola);

struct  file_operations  DC_FOPS = { write: DC_write,
   open: DC_open, release: DC_release, ioctl: DC_ioctl
};
```

IOCTL: (2)

```
void control_motor(unsigned long mascara){
        buffer=buffer^mascara;
        if ( ifValido( buffer )==0){
            buffer=buffer^buffer;
            buffer=ONOFF^mascara;
        }
        outb(buffer,DCDIR);
}
int DC_open(struct inode *inode, struct file *filp){
        MOD_INC_USE_COUNT;
        return 0;
}
int DC_release(struct inode *inode, struct file *filp){
        MOD_DEC_USE_COUNT;
        return 0;
}
int DC_write(struct file *filp, const char *buf, size_t talla,
loff_t *count){
       int comando=*buf;
        outb(comando,DCDIR);
        return 1;
}
```

IOCTL: (3)

```
int DC_ioctl(struct inode *inode, struct file *filp,
                    unsigned int cmd, unsigned long arg){
        switch(cmd) {
                case ONOFF:             // Encender
                {       control_motor( ONOFF );
                        return ONOFF;           }
                case AVANZAR:           // Avanzar
                {       control_motor( AVANZAR );
                        return AVANZAR;         }
                case RETROCEDER:        // Retroceder
                {       control_motor( RETROCEDER );
                        return RETROCEDER;      }
                case DOBLAR_IZQ:        // Doblar Izquierda
                {       control_motor( DOBLAR_IZQ );
                        return DOBLAR_IZQ;      }
                case DOBLAR_DER:        // Doblar Derecha
                {       control_motor( DOBLAR_DER );
                        return DOBLAR_DER;      }
        }
        return 1;
}
```

IOCTL: (4)

```
int init_module(void){
        int err;
        //--- Obtengo la región del puerto paralelo
        err = check_region( DCDIR, 2 );
        if ( err  < 0 ) { printk( "\nErr en carga del modulo.\n" );
                          return -1;  }
        request_region( DCDIR, 2, "topadora" );
        //--- Registro del driver
        err = register_chrdev( DC_MAJOR, "topadora", &DC_FOPS );
        if( err < 0 ){ printk( "\nInaccesible MAYOR %d.", DC_MAJOR );
                       return err;    }
       printk( "\nModulo cargado.\n" );
        return 0;
}
void cleanup_module(void){
       //--- Libero la región del Puerto Paralelo
       release_region( DCDIR, 2 );
       //--- DesRegistro del driver
       unregister_chrdev( DC_MAJOR, "topadora" );
       printk( "\nModulo fue descargado.\n" );
}
```

NCURSES:

Código Ejemplo

NCURSES: (1)

Librería1

Atributos varios

WA_NORMAL	Texto normal
WA_REVERSE	Texto en reverso (colores F y B)
WA_BLINK	Texto en intermitente
WA_BOLD	Texto en negrita

Colores

COLOR_BLACK	COLOR_RED
COLOR_GREEN	COLOR_YELLOW
COLOR_BLUE	COLOR_MAGENTA
COLOR_CYAN	COLOR_WHITE

NCURSES: (2)

Librería2

Líneas gráficas (terminal tipo VT100)

ACS_ULCORNER	Esquina izquierda de arriba
ACS_LLCORNER	Esquina izquierda de abajo
ACS_URCORNER	Esquina derecha de arriba
ACS_LRCORNER	Esquina izquierda de abajo
ACS_HLINE	Línea horizontal
ACS_VLINE	Línea vertical
ACS_PLUS	Signo MÁS GRANDOTE
ACS_DIAMOND	Diamante
ACS_CKBOARD	Chequeado
ACS_DEGREE	Símbolo de grado
ACS_PLMINUS	Más/Menos
ACS_BULLET	Viñeta

```
gcc -lncurses nn.c -o nn
```

NCURSES: (3)

Librería NCURSES (Funciones)

initscr(**void**); Inicializa la librería CURSES.
has_colors(**void**); (**BOOLEAN**) Verifica si es placa color.
start_color(**void**); Inicializa el modo COLOR.
cbreak(**void**); Evita el partido de las líneas de texto.
noecho(**void**); Evita el ECO en pantalla.
beep(**void**); Sonido de la campanilla usando el speaker.
clear(**void**); Limpia la pantalla.
COLOR_PAIR(**int**); Define un par de colores.
getch(**void**); Lee un caracter.
attrset(**atributo**); Define atributo (**COLOR_PAIR** o atributo).
refresh(**void**); Refresca la pantalla.
endwin(**void**); Termina la ejecución del curses.
init_pair(**short ID**, **short TXT**, **short FONDO**); Par de colores.
mvprintw(**int Y**,**int X**, **NCURSES_CONST char ***,...)
Imprime un string como printf en la posición (X,Y).

NCURSES: (4)

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <curses.h>

void drawMouse(int par){
  attrset(  COLOR_PAIR(par));
  mvprintw( 11,  1, "######################“          );
  mvprintw( 12,  1, "##");  mvprintw( 12, 10, "##“ );
  mvprintw( 12, 19, "##“                             );
  mvprintw( 13,  1, "##");  mvprintw( 13, 10, "##“ );
  mvprintw( 13, 19, "##“                             );
  mvprintw( 14,  1, "######################“          );
  mvprintw( 15,  1, "######################“          );
}
```

NCURSES: (5)

```
int main(int argc,char ** argv){
   char b1, b2, b3; int i;
   initscr();                  /* inicializar la librería curses */
   keypad(stdscr, TRUE);       /* permitir el mapeo de teclado */
   cbreak();     /* leer caracteres 1 cada vez, no esperarlos */
   noecho();                      /* no hacer el eco de entrada */
   if (has_colors()){       start_color();
      init_pair( 10, COLOR_RED,   COLOR_RED    );
      init_pair( 11, COLOR_BLUE,  COLOR_BLUE   );
      init_pair( 12, COLOR_GREEN, COLOR_GREEN  );
      init_pair( 13, COLOR_CYAN,  COLOR_CYAN   );
      init_pair( 14, COLOR_CYAN,  COLOR_BLACK  );
      init_pair( 15, COLOR_GREEN, COLOR_BLUE   );
   };
   clear();    drawMouse(13);     getch();
   attrset( COLOR_PAIR(15) );     mvprintw(16,3, "aca va texto");
   attrset( WA_NORMAL );
   drawMouse(10);   getch();
   drawMouse(11);   getch();
   drawMouse(12);   beep();       getch();
   drawMouse(14);   refresh();    beep();
   endwin();
}
```

Práctico:

de **HOY**

Practico de HOY: (1)

Enunciado: Mouse Driver

Realizar un **sistema** (**driver** + **aplicación**) que permita **contar la cantidad de interrupciones del mouse**.

Una vez cargado el módulo, se debe instalar y desinstalar el controlador por medio de un comando de la aplicación. **Otros comandos** serán obtener el contador de interrupciones, poner el contador a cero, poner un valor en el contador, medir la **velocidad** de las interrupciones usando un timer.

Usar la librería **NCURSES** para la interfaz de la aplicación.

Parámetros

Tipo = **PS2** o **MICROSOFT**

Interrupcion = **NroDeInterrupcion**

Practico de HOY: (2)

GPM: General Purpose Mouse server

GPM es el programa que "lee" el dispositivo donde tengamos conectado nuestro mouse (/dev/psaux si es un mouse PS/2).

Instalación:

```
gpm -m /dev/psaux -t ps2
```

Funciones:

- proveer una **interfaz** entre el **mouse** y la **consola** (para copiar y pegar),
- actuar de "**repetidor**" entre el dispositivo del mouse y el dispositivo virtual que normalmente leerá el servidor X o cualquier otro sistema.

Practico de HOY: (3)

GPM:

Para qué sirve el repetidor ?,
Se puede leer directamente el dispositivo?

Sí. Pero si se desconecta el mouse, bastará reiniciar el **gpm** y todo sigue funcionando.

Dispositivos:

/dev/ttyS0	para **serie** (**COM1**)
/dev/input/mice	para **USB**
/dev/psaux	para **PS2**

Practico de HOY: (4)

Timer:

Cálculo de Velocidad:

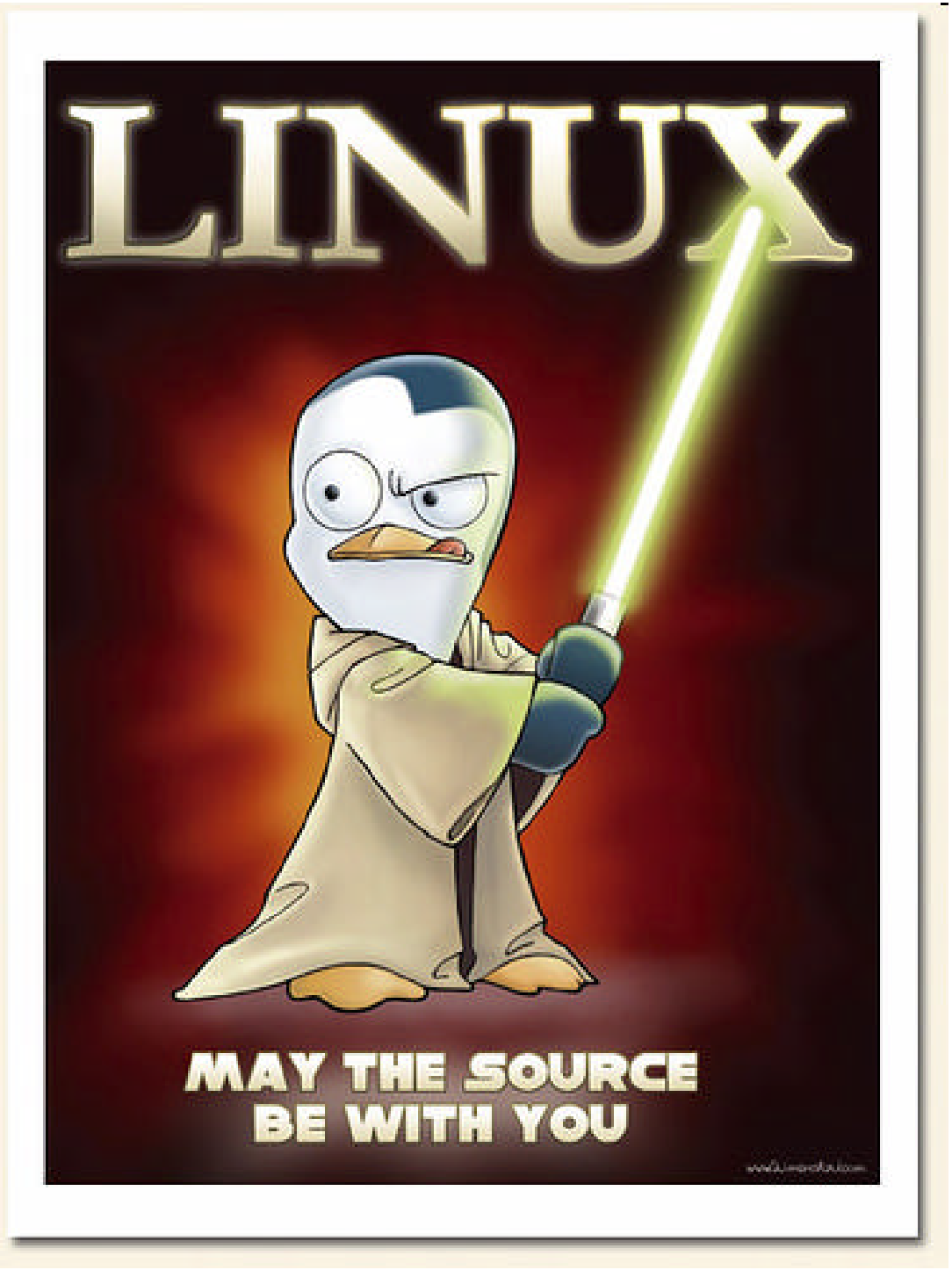
Se supone que las interrupciones por segundo representan la velocidad de desplazamiento de una rueda de un móvil.

Entonces se debe poder parametrizar:

- Cantidad de **interrupciones por revolución** (dientes)
- Tamaño de la rueda (**diámetro**)
- **Tiempo de muestreo** (retraso entre mediciones de int)

La aplicación debe leer la velocidad en CM por segundo.

Similitudes religiosas con Anakin Skywalker ...



Esto es todo ...

Por ahora ...