```c
/*
 * Lab3: developing a Linux device driver.
 *
 * LCD display module for the bcm2708 board family.
 *
 */

/* Required headers. */
#include <linux/module.h>
#include <linux/init.h>
#include <linux/cdev.h>
#include <linux/fs.h>
#include <linux/kernel.h>
#include <linux/slab.h>
#include <linux/types.h>
#include <linux/errno.h>
#include <linux/uaccess.h>
#include <linux/gpio.h>

#include <asm/delay.h>


/* For spinlocks */
#include <linux/spinlock.h>



/* For ioctl */
#include <linux/ioctl.h>


/* The name of the driver. */
#define BCM2708_LCD_DRIVER_NAME "bcm2708_lcd"


/* Modinfo - Informations about this module */
MODULE_AUTHOR("NASR ALLAH Mounir");
MODULE_DESCRIPTION("Drivers pour le contrôleur lcd Hitachi HD44780");
MODULE_SUPPORTED_DEVICE("Raspberry Pi - BCM2708");
MODULE_LICENSE("GPL");


/* Commands definitions. */
#define LCD_CMD_CLR         0x01
#define LCD_CMD_HOME        0x02
#define LCD_CMD_ENTRY       0x04
#define LCD_CMD_ON_OFF      0x08
#define LCD_CMD_CDSHIFT     0x10
#define LCD_CMD_FUNC        0x20
#define LCD_CMD_CGRAM       0x40
#define LCD_CMD_DGRAM       0x80

#define LCD_CMD_ENTRY_SH    0x01
#define LCD_CMD_ENTRY_ID    0x02

#define LCD_CMD_ON_OFF_B    0x01
#define LCD_CMD_ON_OFF_C    0x02
#define LCD_CMD_ON_OFF_D    0x04

#define LCD_CMD_FUNC_F      0x04
```

```
#define LCD_CMD_FUNC_N      0x08
#define LCD_CMD_FUNC_DL     0x10

#define LCD_CMD_CDSHIFT_RL  0x04

#define LCD_GPIO_RS         18
#define LCD_GPIO_EN         23

#define LCD_GPIO_D0         4
#define LCD_GPIO_D1         17
#define LCD_GPIO_D2         27
#define LCD_GPIO_D3         22




// ***  GPIOs definitions
#define GPIO_NR            6
#define GPIO_DATA_NR       4
static const int gpio_data[]={LCD_GPIO_D0,LCD_GPIO_D1,LCD_GPIO_D2,LCD_GPIO_D3};




/* Declare GPIO used to interface the LCD display. */
static struct gpio bcm2708_lcd_gpios[] = {
  {
      .gpio = LCD_GPIO_RS
    , .flags = GPIOF_OUT_INIT_LOW
    , .label = "bcm2708_lcd_rs"
  } ,
  {
      .gpio = LCD_GPIO_EN
    , .flags = GPIOF_OUT_INIT_LOW
    , .label = "bcm2708_lcd_en"
  },
  {
    .gpio = LCD_GPIO_D0
    , .flags = GPIOF_OUT_INIT_LOW
    , .label = "bcm2708_lcd_d0"
  },
  {
    .gpio = LCD_GPIO_D1
    , .flags = GPIOF_OUT_INIT_LOW
    , .label = "bcm2708_lcd_d1"
  },
  {
    .gpio = LCD_GPIO_D2
    , .flags = GPIOF_OUT_INIT_LOW
    , .label = "bcm2708_lcd_d2"
  },
  {
    .gpio = LCD_GPIO_D3
    , .flags = GPIOF_OUT_INIT_LOW
    , .label = "bcm2708_lcd_d3"
  }
};
```

```
/* Release the GPIOs. */
static
void
bcm2708_lcd_release_gpio ( void )
{
  gpio_free_array(bcm2708_lcd_gpios, GPIO_NR);
}



/* Set up GPIOs using gpiolib */
static
int
bcm2708_lcd_setup_gpio ( void )
{

  int error;
  int i;

  for ( i = 0; i < GPIO_NR; ++i ) {
    if ( !gpio_is_valid(bcm2708_lcd_gpios[i].gpio) ){
      printk ( KERN_ALERT "lcd gpio %s (%d) is not valid.\n",
               bcm2708_lcd_gpios[i].label ,
               bcm2708_lcd_gpios[i].gpio );
      return -EINVAL;
    }
  }

  error = gpio_request_array(bcm2708_lcd_gpios, GPIO_NR);

  if(error){
    return error;
  }

  return 0;
}



/* Simulate a falling edge */
static
void
bcm2708_lcd_strobe ( void )
{
    gpio_set_value ( LCD_GPIO_EN, 1 );
    udelay ( 50 );
    gpio_set_value ( LCD_GPIO_EN, 0 );
    udelay ( 50 );
}



/* Send 4 bits to the LCD */
static
void
bcm2708_lcd_write_4bit_value ( uint8_t value )
{
  int i;
```

```
  for(i=0;i<GPIO_DATA_NR;i++){
    gpio_set_value(gpio_data[i], value & 0x1);
    value >>= 1;
  }

  bcm2708_lcd_strobe();
}



/* Send 8 bits to the LCD */
static
inline
void
bcm2708_lcd_write_value ( uint8_t value )
{
    /* Write the 4 upper bits. */
    bcm2708_lcd_write_4bit_value ( value >> 4 );

    /* Write the 4 lower bits. */
    bcm2708_lcd_write_4bit_value ( value );
}



/* Send 4 bits command */
static
inline
void
bcm2708_lcd_send_cmd_4bits ( uint8_t value )
{
    gpio_set_value ( LCD_GPIO_RS, 0 );
    bcm2708_lcd_write_4bit_value ( value );
}



/* Send 8 bits command */
static
inline
void
bcm2708_lcd_send_cmd ( uint8_t cmd )
{
    gpio_set_value ( LCD_GPIO_RS, 0 );
    bcm2708_lcd_write_value ( cmd );
}



/* Set position */
static
inline
void
bcm2708_lcd_set_position ( int x, int y )
{
    static uint8_t const row_offset[] = { 0x00, 0x40, 0x14, 0x54 };
    bcm2708_lcd_send_cmd ( x + ( LCD_CMD_DGRAM | row_offset[y] ) );
}
```

```
/* Send an char data to the LCD */
static
inline
void
bcm2708_lcd_put ( char c )
{
  gpio_set_value ( LCD_GPIO_RS, 1 );
  bcm2708_lcd_write_value ( ( uint8_t ) c );
}



/* Send an string to the LCD */
static
inline
void
bcm2708_lcd_put_string ( const char * str )
{

    while ( *str ) {
      bcm2708_lcd_put ( *str++ );
    }
    udelay ( 50 );


}



/* Envoyer la commande "Home" */
static
inline
void
bcm2708_lcd_go_home ( void )
{
    gpio_set_value ( LCD_GPIO_RS, 0 );
    bcm2708_lcd_send_cmd ( LCD_CMD_HOME );
    udelay ( 2000 );
}



/* Send the command to clear LCD*/
static
inline
void
bcm2708_lcd_clear ( void )
{
    gpio_set_value ( LCD_GPIO_RS, 0 );
    bcm2708_lcd_send_cmd ( LCD_CMD_CLR );
    udelay ( 2000 );
}



/* Initialize the LCD */
static
int
bcm2708_lcd_init ( void )
```

```
{
    int     err;
    uint8_t func;

    err = bcm2708_lcd_setup_gpio ();
    if ( err < 0 ) {
        return err;
    }

    udelay ( 2000 );

    /* Command mode. */
    gpio_set_value ( LCD_GPIO_RS, 0 );

    /* Init 8-bit mode. */
    func = LCD_CMD_FUNC | LCD_CMD_FUNC_DL;
    bcm2708_lcd_send_cmd_4bits ( func >> 4 );
    udelay ( 50 );
    bcm2708_lcd_send_cmd_4bits ( func >> 4 );
    udelay ( 50 );
    bcm2708_lcd_send_cmd_4bits ( func >> 4 );
    udelay ( 50 );

    /* Init 4-bit mode. */
    func = LCD_CMD_FUNC;
    bcm2708_lcd_send_cmd_4bits ( func >> 4 );
    udelay ( 50 );

    /* Setup rows. */
    func |= LCD_CMD_FUNC_N;
    bcm2708_lcd_send_cmd ( func );
    udelay ( 50 );

    /* Remainder of initialization. */
    bcm2708_lcd_send_cmd ( LCD_CMD_ON_OFF | LCD_CMD_ON_OFF_D );
    udelay ( 50 );

    bcm2708_lcd_send_cmd ( LCD_CMD_ENTRY | LCD_CMD_ENTRY_ID );
    udelay ( 50 );

    bcm2708_lcd_send_cmd ( LCD_CMD_CDSHIFT | LCD_CMD_CDSHIFT_RL );
    udelay ( 50 );

    bcm2708_lcd_send_cmd ( LCD_CMD_CLR );
    udelay ( 2000 );

    return 0;
}



/* Deinitialization of the LCD */
static
inline
void
bcm2708_lcd_deinit ( void )
{
    /* Clear display. */
    bcm2708_lcd_clear ();
```

```
    /* Deinitialize gpios. */
    bcm2708_lcd_release_gpio ();

}



#define BUFFER_SIZE       256  // Taille du buffer
#define LCD_X               4  // Nombre de lignes
#define LCD_Y              20  // Nombre de caractères par ligne



/* Numéro "Magique" du pilote */
#define BCM2708_LCD_MAGIC 'l'

/* Commande "clear", commande sans argument */
#define BCM2708_LCD_IOCCLEAR _IO( BCM2708_LCD_MAGIC, 1)

/* Commande "Home" , commande sans argument */
#define BCM2708_LCD_IOCHOME _IO( BCM2708_LCD_MAGIC, 2 );

/* Récupère le paramètre par valeur */
#define BCM2708_LCD_IOCQCURPOS _IO( BCM2708_LCD_MAGIC, 3 );

/* Récupère le paramètre par pointeur. */
#define BCM2708_LCD_IOCGCURPOS _IOR( BCM2708_LCD_MAGIC, 4, int )

/* Nombre de commandes définis */
#define BCM2708_LCD_MAXNR 4



struct bcm2708_lcd_dev
{

  /* Peripherique caractère */
  struct cdev cdev;

  /* X : position représentant la ligne sélectionné
     Y : position représentant la position sur la ligne */
  size_t xpos, ypos;

  /* Buffer */
  char buffer[BUFFER_SIZE];

  /* Spin lock, Verrou tournant pour gérer la concurence */
  spinlock_t lock;

};



/* Pointeur sur la structure de donnée utilisé par le pilote du LCD */
static struct bcm2708_lcd_dev * bcm2708_lcdp;
```

```
/* Nombre majeur du LCD */
static int bcm2708_lcd_major;


/* Operation d'ouverture */
int
bcm2708_lcd_open ( struct inode * inodep
                 , struct file *  filep )
{
    struct bcm2708_lcd_dev * lcdp;

    lcdp = container_of( inodep->i_cdev, struct bcm2708_lcd_dev , cdev );
    filep-> private_data = lcdp;

    lcdp->xpos = 0;
    lcdp->ypos = 0;

    memset(lcdp->buffer,0,sizeof(lcdp->buffer));

    return 0;
}




/* Operation de fermeture*/
int
bcm2708_lcd_close ( struct inode * inodep
                  , struct file *  filep )
{
    return 0;
}



/* Opération d'écriture  */
ssize_t
bcm2708_lcd_write ( struct file * filep
                  , const char *  buf
                  , size_t        length
                  , loff_t *      ppos )
{
    struct bcm2708_lcd_dev * lcdp;
    int i;
    int err;

    // le champ pivate_data contient la structure qui représente
    // le device ( position du curseur, cdev, etc... )
    lcdp = filep->private_data;


    if(length > BUFFER_SIZE){
      printk ( KERN_ALERT "Error : Size to big.\n");
      return -1;
    }
```

```
    // On verouille le peripherique, principe du verrou tournant
    spin_lock(&(lcdp->lock));


    // On copie le buffer de l'utilisateur
    err = copy_from_user( lcdp->buffer, buf, length );
    if(err != 0 ){
      printk ( KERN_ALERT "Error : copy_from_user.\n");
      return -1;
    }

    // On va a la position "courante"
    bcm2708_lcd_set_position( lcdp->xpos, lcdp->ypos );

    // Tant que l'on a des données à envoyer
    for ( i=0; i<(int)length ; i++ ){

      // Saut de ligne, changement de position
      if((lcdp->buffer)[i]=='\n'){
        lcdp->xpos++;
        bcm2708_lcd_set_position(lcdp->xpos, lcdp->ypos );
      }
      // Si c'est la fin de la ligne, on passe à la suivante
      else if(lcdp->ypos >= LCD_Y){
        lcdp->xpos++;
        bcm2708_lcd_set_position(lcdp->xpos, lcdp->ypos );
        bcm2708_lcd_put((lcdp->buffer)[i]);
        lcdp->ypos++;
      }
      // Sinon on écrit un caractère et on incrémente la position
      else{
        bcm2708_lcd_put((lcdp->buffer)[i]);
        lcdp->ypos++;
      }

    }


    // On relache le périphérique
    spin_unlock(&(lcdp->lock));



    // On retourne le nombre de données écrites
    return i;
}




/* Implémentation de la fonction ioctl */
int
bcm2708_lcd_ioctl(struct inode * inodep
                  ,struct file * filep
                  ,unsigned long cmd
                  ,unsigned long arg ){


  // Erreur et valeur de retour
  int err = 0, retval, curpos;
```

```
  // Si le numero magique donne dans la commande est different du
  // numero magique du pilote on renvoie une erreur
  if( _IOC_TYPE(cmd) != BCM2708_LCD_MAGIC ) return -EINVAL;


  // Si le numéro le numéro local de la commande est plus grand
  // que le nombre de commandes définis, on renvoie une erreur.
  if ( _IOC_NR( cmd ) > BCM2708_LCD_MAXNR ) return -EINVAL;


  // Dans le cas où l'on doit lire l'argument, alors on vérifie
  // que l'on peut accéder à l'adresse donnée par l'utiisateur
  if ( _IOC_DIR ( cmd ) & _IOC_READ ) {
    err = access_ok ( VERIFY_READ
                      , ( void __user * ) arg
                      , _IOC_SIZE ( cmd ) );
  }


  // Dans le cas où l'on doit écrire l'argument, alors on vérifie
  // que l'on peut accéder à l'adresse donnée par l'utiisateur
  else if ( _IOC_DIR ( cmd ) & _IOC_WRITE ){
    err = access_ok ( VERIFY_WRITE
                      , ( void __user * ) arg
                      , _IOC_SIZE ( cmd ) );
  }

  // Gestion de la commande
  switch ( cmd ) {

  // Si la commande cmd est clear
  case BCM2708_LCD_IOCCLEAR:
    bcm2708_lcd_clear();
    break;

  // Si la commande cmd est Home
  case BCM2708_LCD_IOCHOME :
    bcm2708_lcd_go_home();
    break;

  // Si la commande "cmd" est de recuperer le paramètre par valeur
  case BCM2708_LCD_IOCQCURPOS :
    retval = arg;
    break;

  // Si la commande "cmd" est de recuperer le paramètre par pointeur
  case BCM2708_LCD_IOCGCURPOS :
    retval = __get_user(curpos,( int __user * )arg );
    break;

  default:
    return EINVAL;

  }

  return retval;

}
```

```
// Opérations disponibles sur le fichier spécial,
// qui permet à l'utilisateur d'interagir avec le périphérique.
struct file_operations bcm2708_lcd_fops = {
    .owner   = THIS_MODULE,
    .open    = bcm2708_lcd_open,
    .write   = bcm2708_lcd_write,
    .ioctl   = bcm2708_lcd_ioctl,
    .release = bcm2708_lcd_close
};



// Initialisation du module
static
int
__init
bcm2708_lcd_init_module ( void )
{
    int   err;
    dev_t dev;


    printk("Bonjour \n");

    // On récupère dynamiquement un nombre majeur,
    // Ce qui initialise la structure dev
    alloc_chrdev_region ( &dev
                          , 0 /* Premier numéro mineur */
                          , 1 /* Nombre de périphériques */
                          , BCM2708_LCD_DRIVER_NAME );


    // On récupère le nombre majeur qui a été initialisé dynamiquement
    bcm2708_lcd_major = MAJOR ( dev );

    // Allocation de la structure de donnée en utilisant la fonction kzalloc
    bcm2708_lcdp = kzalloc(sizeof(struct bcm2708_lcd_dev),GFP_KERNEL);
    if(bcm2708_lcdp == NULL) {
      printk ( KERN_ALERT "Error : kzalloc in bcm2708_lcd_init_module.\n");
      return 5;
    }


    // Initialisation du periphérique caractère
    cdev_init( &bcm2708_lcdp->cdev, &bcm2708_lcd_fops );
    bcm2708_lcdp->cdev.owner = THIS_MODULE;


    // Initialisation du spinlock
    spin_lock_init(&(bcm2708_lcdp->lock));


    // On ajoute le periphérique caratère au noyeau de l'OS
    err = cdev_add ( &bcm2708_lcdp->cdev, dev, 1 );
    if( err < 0 ){
```

```
      printk ( KERN_ALERT "Error : cdev_init in bcm2708_lcd_init_module.\n");
      return 6;
    }


    /* Initialisation du LCD */
    err = bcm2708_lcd_init();
    if( err < 0 ){
      printk ( KERN_ALERT "Error : bcm2708_lcd_init in bcm2708_lcd_init_module.
          \n");
      return 7;
    }


    return 0;
}




/* Désactivation du module */
static
void
__exit
bcm2708_lcd_cleanup_module ( void )
{
    dev_t dev;

    printk("Bye bye \n");

    /* Deinitialisation du LCD */
    bcm2708_lcd_deinit();

    /* Unregister the chardev driver from the kernel. */
    cdev_del(&bcm2708_lcdp->cdev);

    /* On libère le nombre majeur */
    dev = MKDEV( bcm2708_lcd_major, 0 );
    unregister_chrdev_region( dev, 1 );

    /* On libère la mémoire alloué pour le peripherique caractère */
    kfree(bcm2708_lcdp);
}


/* Fonction qui sera appelé pour l'initialisation du module */
module_init ( bcm2708_lcd_init_module );

/* Fonction qui sera appelé pour la suppression du module */
module_exit ( bcm2708_lcd_cleanup_module );
```