ในการ design application ที่จะ communicate กับ custom hardware จะมี 2 level ที่ควรจะต่างกันอย่างไร

- User level, where the behavior of the application is defined using virtual file system (VFS) calls to communicate with the hardware
- Module level, where the behavior of each VFS function is implemented based upon the functionalities the hardware implements

programmer ควรจะต่าง aware of the functionalities ที่ hardware มีให้ และวิธีใช้ VFS ที่เรียกมา accordingly

(different hardware อาจจะ require การ implement ที่ต่างกันแม้จะใน VFS function เดียวกัน)

Module Level

ต้องรองรับ funtionalities ที่ device ไม่ support

1. Reset: L is disabled, and the blink rate register is set to zero.
2. Program: The blink rate register is set to a user-defined value.
3. Enable: L is enabled.
4. Disable: L is disabled.
5. Poll rate: L returns the content of the blink rate register.
6. Poll state: L returns the content of the enable register.
7. Power-off: L terminates its operation and starts waiting for the next reset.

1. Both the blink rate and enable registers are set to zero.
   This operation is done once, when starting using the device.
2. The blink rate register is set to a user defined value.
   This operation can be done multiple times, during the device usage
3. The enable register is set to one.
   This operation can be done multiple times, during the device usage
4. The enable register is set to zero.
   This operation can be done multiple times, during the device usage
5. The blink rate register content is provided to the user level.
   This operation can be done multiple times, during the device usage
6. The enable register content is provided to the user level.
   This operation can be done multiple times, during the device usage
7. The enable register content is set to zero.
   This operation is done once, after the last usage of the device.

VFS functions ที่มักจะใช้

- open, which initiates the operations with the device
- release, which terminates the operation with the device
- write, which sends data coming from the user space to the device
- read, which reads from the device and send them to the user space
- ioctl, which performs custom operations

Device functionality	Virtual file system function	Notes
Reset	open	open() is used once, to establish the connection with the device.
Program	write	write() is used to send data to the device. The blink rate register shall be selected as target for the write operation using the ioctl() function.
Enable	write	write() is used to send data to the device. The enable register shall be selected as target for the write operation using the ioctl() function.
Disable	write	write() is used to send data to the device. The enable register shall be selected as target for the write operation using the ioctl() function.

(associated with)

Device functionality	Virtual file system function	Notes
Poll rate	read	read() is used to send data to the application. The blink rate register shall be selected as target for the read operation using the ioctl() function.
Poll state	read	read() is used to send data to the application. The enable register shall be selected as target for the read operation using the ioctl() function.
Power-off	release	release() is used to terminate the connection with the device.
None	ioctl	ioctl() is used to select the target for read/write operations.

การอ่านจาก device

Hidden in

- `READ_DATA_FROM_THE_HW()`
- `WRITE_DATA_TO_HW()`

Implementation depends on the CPU/Device L connection

Memory mapped example:

- Blink rate register: 0xf0080000
- Enable register: 0xf0080004

GPIO example:

- Blink rate register: GPIO(0-31) (MSB first)
- Enable register: GPIO(32)

most significant bit

Module Mapped I/O

int check_region · check ว่า address ที่ต้องการจะ available มั้ย

int request_region · จอง address ที่ต้องการ

int release_region · set ไม่ address ที่ต้องการจอง

function

1.) initialization

```
static int __init L_module_init(void)
{
  int res;
  alloc_chrdev_region(&L_dev, 0, 1, "L_dev");
  printk(KERN_INFO "%s\n", format_dev_t(buffer, L_dev));
  cdev_init(&L_cdev, &L_fops);
  L_cdev.owner = THIS_MODULE;
  cdev_add(&L_cdev, L_dev, 1);

  r = check_region( ioremap(0xf0080000, 4 ), 8 );
  if(r) {
   printk( KERN_ALERT "Unable to reserve I/O memory\n");
   return -EINVAL;
  }
  request_region(ioremap(0xf0080000, 4), 8, "DevL");
  return 0;
}
```

แปลง physical address ของ device (ที่ถูก define โดย memory map) ให้เป็น virtual address

2.) clean - up

```
static void __exit L_module_cleanup(void)
{
  cdev_del(&L_cdev);
  unregister_chrdev_region(L_dev, 1);

  release_region(ioremap(0xf0080000, 4 ), 8);
}
```

(free occupy address)

3.) read

```
int     READ_DATA_FROM_THE_HW( int *data )
{
  int   tmp;

  switch( selected_register )
  {
    case BLINK_RATE:
      tmp = inl( ioremap(0xf0080000, 4) );
      break;
    case ENABLE:
      tmp = inl( ioremap(0xf0080000, 4)+4 );
      break;
    }
    *data = tmp;

    return 4;
}
```

inb ( ) · reads 8 - bit words

inw ( ) · reads 16 - bit words

inl ( ) · reads 32 - bit words

4.) Write

```
int     WRITE_DATA_TO_THE_HW( char *data )
{
  switch( selected_register )
  {
    case BLINK_RATE:
      outl( (int)*data, ioremap(0xf0080000, 4) );
      break;
    case ENABLE:
      outl( (int)*data, ioremap(0xf0080000, 4)+4 );
      break;
    }

    return 4;
}
```

outb() : writes 8-bit words

outw() : writes 16-bit words

outl() : writes 32-bit words

GPIO-based I/O

int gpio_request : check GPIO ที่เราต้องการว่า available มั้ย ถ้า yes : จอง

void gpio_free : set GPIO ที่จองแล้วให้ free

function

1.) initialization (check + reserve one by one)

```
static int __init L_module_init(void)
{
  int i, r;
  alloc_chrdev_region(&L_dev, 0, 1, "L_dev");
  printk(KERN_INFO "%s\n", format_dev_t(buffer, L_dev));
  cdev_init(&L_cdev, &L_fops);
  L_cdev.owner = THIS_MODULE;
  cdev_add(&L_cdev, L_dev, 1);

  for( i = 0; i < 32; i++ ) {
    r = gpio_request( i );
    if (r) {
      printk( KERN_ALERT "Unable to reserve GPIO\n");
      return -EINVAL;
    }
  }
  return 0;
}
```

2.) clean-up (free one by one)

```
static void __exit L_module_cleanup(void)
{
  int i;

  cdev_del(&L_cdev);
  unregister_chrdev_region(L_dev, 1);

  for( i = 0; i < 32; i++ )
    gpio_free( i );
}
```

3.) read

```c
int      READ_DATA_FROM_THE_HW( int *data )
{
  int    tmp = 0, i;

  switch( selected_register )
  {
    case BLINK_RATE:
      for( i = 0; i < 31; i++ ) {
        gpio_direction_input( i );
        tmp = (tmp << 1) | gpio_get_value( i );
      }
      break;
    case ENABLE:
      gpio_direction_input( 32 );
      tmp |= gpio_get_value( 32 )
      break;
  }
  *data = tmp;

  return 4;
}
```

value read แบบ one by one
resulting word is built
most significant bit first

4.) write

```c
int      WRITE_DATA_TO_THE_HW( int data )
{
  int    i;
  switch( selected_register ) {
    case BLINK_RATE:
      for( i = 0; i < 31; i++ ) {
        gpio_direction_output( i, 0 );
        gpio_set_value( i, (data & (1 << i)) );
      }
      break;
    case ENABLE:
      gpio_direction_output( 32, 0 );
      gpio_set_value( 32, data & 0x00000001 );
      break;
  }

  return 4;
}
```

0 : default value

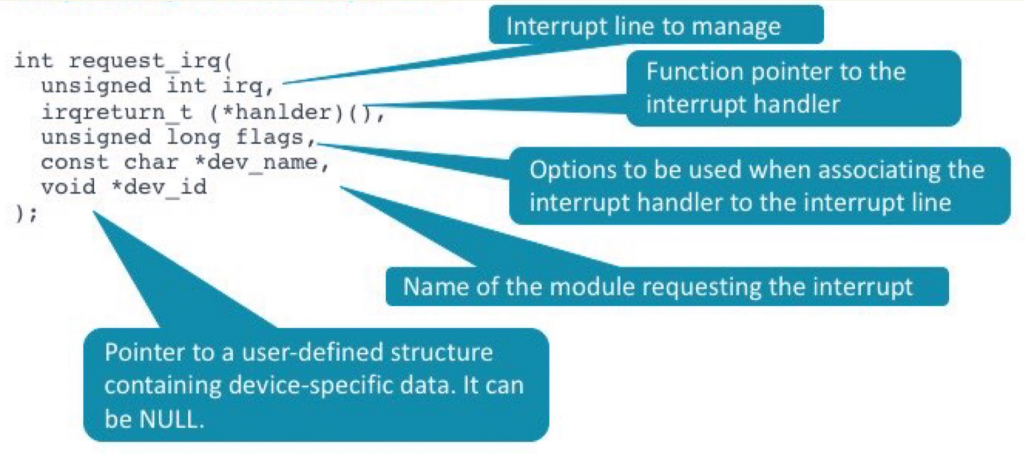
Interrupts (จาก hardware)

ขั้นตอนการ handle interrupts ของ kernel module

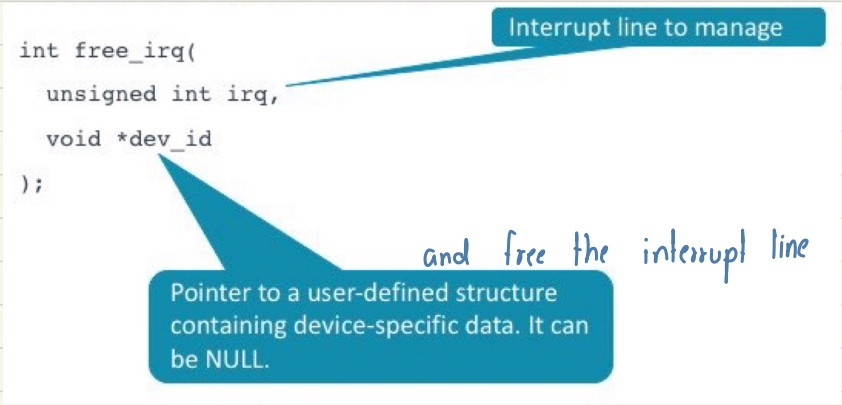
- Request an interrupt line
- Associate an interrupt handler to an interrupt line
- Implement the interrupt handler

function

1.) requesting the interrupt line



2.) freeing the interrupt



3.) the interrupt handler

```
static irqreturn_t hlr( int irq, void *dev_id )
{
      /*
   * Do something to handle the interrupt *
   */

  return IRQ_RETVAL(1);
}
```

Interrupt handling -- long latencies

- low piority interrupt ต่ำๆ
- ถ้า interrupts are disabled , ทุกอย่างช้าๆ

* keep intempt handlers as short as possible

Work Queue

เป็น list ของ activities ที่ executed

- work - data ที่ processed (define โดยใช้ work_struct structure)
- callback - function to process การทำงาน (เป็น C function)

} ส่วนที่ใช้ในการ define activity

The User Level

application จะมีการทำงานกับ VFS calls ในการ implement the intended behivior

For the considered example, the application shall

- Open the connection with the device
- Set the desired blinking rate
- Enable the device
- Adjust the blinking rate (if needed)
- Disable/enable the device (if needed)
- Close the connection with the device

Quiz

1. "Which virtual file system function is the device function "Reset" associated to?"

- ○ Write
- ✔ Open
- ○ Read
- ○ Release

2. "Which virtual file system function is the device function "Power-off" associated to?"

- ○ Write
- ○ Open
- ○ Read
- ✔ Release

3. "What is a potential problem caused by interrupt handlers?"

- ○ Crashes the system
- ○ Causes the kernel to lose track of ongoing tasks
- ✔ Increased latency

4. "What is a "work queue"?"

- ✔ A structure in the Linux Kernel that contains a list of activities that need to be executed.
- ○ A structure in the Linux Kernel that contains pointers to running tasks.
- ○ A list of process currently running on the kernel

5. "Which function allows the transfer of data from kernel space to user space?"

- ○ "copy_from_kernel"
- ○ "copy_from_user"
- ○ "copy_to_kernel"
- ✔ "copy_to_user"