ในการ communication ระหว่าง user level และ module level โดยการ implement มีแบบ

1.) virtual file system (VFS) interface → user level application ทำงานกับ VFS API ในการเข้าถึง device file ใช้ในการ communicate กับ specific device modeled ที่จำลองเป็น file

2.) RAM based filesystem (sysfs) → ส่ง kernel data structure ออกสู่ user level ใช้ provide debug information

sysfs File System

- RAM based file system containing directories or files ที่ถูกสร้างโดย Linux kernel
- แต่ละ: directory or file จะ: contain ข้อมูลเกี่ยวกับ portions ของ kernel ที่ถูก set visible ใน user
  - content ไม่ได้ถูก define โดย any specific APIs
  - Kernel developers สามารถ export ข้อมูลใดก็ได้ที่ต้องการ
- user สามารถ read or write kernel objects ได้ผ่านทาง sysfs

The user can write to GPIOs by

- Exporting the GPIO, e.g., 105: `echo 105 > /sys/class/gpio/export`
- Setting the direction: `echo "out" > /sys/class/gpio/gpio105/direction`
- Writing the desired value: `echo 1 > /sys/class/gpio/gpio105/value`

The user can read from GPIOs by

- Exporting the GPIO, e.g., 105: `echo 105 > /sys/class/gpio/export`
- Setting the direction: `echo "in" > /sys/class/gpio/gpio105/direction`
- Reading the GPIO value: `cat /sys/class/gpio/gpio105/value`

Adding Entries to the sysfs File System

```
struct kobject {
  char                  *k_name;
  char                  name[KOBJ_NAME_LEN];
  struct kref           kref;
  struct list_head      entry;
  struct kobject        *parent;
  struct kset           *kset;
  struct kobj_type      *ktype;
  struct dentry         *dentry;
};
```

add new directory to sysfs file system Kernel object จะถูก define โดยใช้ kobject data structure

- kobject_create_and_add() : add object to file system
- kobject_put() : destroy object

```
struct kobj_attribute {
  struct attribute attr;
  ssize_t (*show)(struct kobject *kobj,
                  struct kobj_attribute *attr,
                  char *buf);
  ssize_t (*store)(struct kobject *kobj,
                   struct kobj_attribute *attr,
                   const char *buf,
                   size_t count);
};
```

} execute เมื่อ file ถูก read

} execute เมื่อ file ถูก write

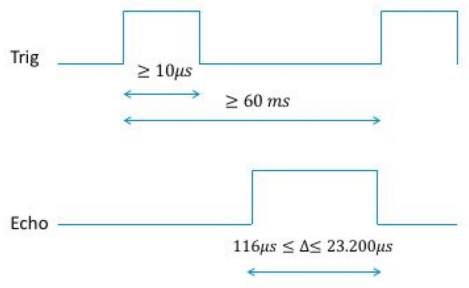
- sysfs_create_file() : add new file

add new file to sysfs file system (new object attribute ต้อง define kobject_attribute data structure)

HC-SR04 Ultrasonic Ranging Sensor

ใช้ในการคำนวณระยะห่าง ultrasonic กับ object

- Vcc, 5V power supply input
- GND, ground input
- Trig, TTL input to trigger the operation of the sensor
- Echo, TTL output with a pulse-modulated square waveform providing the distance readout



- ในการ enable sensor, pulse ต้องถูก generate on Trig input
- pulse จะต้องมีช่วง 10 ms, ต้องมีช่วง 60 ms จะถูกแบ่งเป็น 2 trigger pulses ที่ต่อเนื่องกัน
- Echo pulse อยู่ระหว่าง 116 ms (2 cm) ถึง 23200 ms (400 cm)

(time diagram)

loadable kernel module จะใช้ VFS API

- write() → trigger the sensor เพื่อคำนวณ echo pulse duration
- read() → provide ค่าของ echo pulse duration ที่ถูกคำนวณแล้วในหน่วย ms ให้ user level

Module Data Structure Definition

```
static dev_t hcsr04_dev;
struct cdev hcsr04_cdev;
```
Data structures for a new character-based device

```
static int hcsr04_lock = 0;
```
Module usage flag

```
static struct kobject *hcsr04_kobject;
```
Kernel object for adding a directory entry to sysfs

```
static ktime_t rising,
               falling;
```
Data structures to keep the time of the echo pulse rising edge/falling edge

```
static struct kobj_attribute hcsr04_attribute =
__ATTR(hcsr04, 0660, hcsr04_show, hcsr04_store);
```
Kernel object attribute

Name of the file | File access right | Functions invoked when the file is read/written

```
struct file_operations hcsr04_fops = {
  .owner = THIS_MODULE,
  .read = hcsr04_read,
  .write = hcsr04_write,
  .open = hcsr04_open,
  .release = hcsr04_close,};
```

(file operations ของ kernel module ที่ corresponding กับ VFS API)

function

1.) initialization

```
static int __init hcsr04_module_init(void)
{
  char buffer[64];

  alloc_chrdev_region(&hcsr04_dev, 0, 1, "hcsr04_dev");
  printk(KERN_INFO "%s\n", format_dev_t(buffer, hcsr04_dev));
  cdev_init(&hcsr04_cdev, &hcsr04_fops);
  hcsr04_cdev.owner = THIS_MODULE;
  cdev_add(&hcsr04_cdev, hcsr04_dev, 1);
  gpio_request( GPIO_OUT, "hcsr04_dev" );
  gpio_request( GPIO_IN, "hcsr04_dev" );
  gpio_direction_output( GPIO_OUT, 0 );
  gpio_direction_input( GPIO_IN );
  hcsr04_kobject = kobject_create_and_add("hcsr04", kernel_kobj);
  sysfs_create_file(hcsr04_kobject, &hcsr04_attribute.attr);
  return 0;
}
```

insert new character device ใน Linux kernel

ขอ GPIOs ของ character device (กำหนดให้ output for Trig signal, input ของ echo signal)

add hcsr04 file ใน /sys/kernel/hcsr04

add hcsr04 directory ใน /sys/kernel

2.) clean - up

```
static void __exit hcsr04_module_cleanup(void)
{
  gpio_free( GPIO_OUT );
  gpio_free( GPIO_IN );

  hcsr04_lock = 0;

  cdev_del(&hcsr04_cdev);
  unregister_chrdev_region( hcsr04_dev, 1 );

  kobject_put( hcsr04_kobject );
}
```

Release the used GPIOs.

Mark the module as free.

Remove the character device from the kernel.

Remove the hcsr04 directory from sysfs.

3.) open

```
int hcsr04_open(struct inode *inode, struct file *file)
{
  int ret = 0;

  if( hcsr04_lock > 0 )
    ret = -EBUSY;
  else
    hcsr04_lock++;

  return( ret );
}
```

ในเวลาๆนึง จะมีแค่ application เดียวที่สามารถใช้ device ได้

4.) close

```
int hcsr04_close(struct inode *inode, struct file *file)
{
  hcsr04_lock = 0;

  return( 0 );
}
```

ตั้งค่า device ให้ว่างเมื่อใช้งาน

5.) write

```
ssize_t hcsr04_write(struct file *filp, const char *buffer, size_t length, loff_t * offset)
{
  gpio_set_value( GPIO_OUT, 0 );
  gpio_set_value( GPIO_OUT, 1 );
  udelay( 10 );
  gpio_set_value( GPIO_OUT, 0 );

  while( gpio_get_value( GPIO_IN ) == 0 )
    ;
  rising = ktime_get();

  while( gpio_get_value( GPIO_IN ) == 1 )
    ;
  falling = ktime_get();

  return( 1 );
}
```

Generate a pulse on the output connected to Trig. After setting the output to 1, we wait for 10 µs, and then we return the output to 0.

We wait as long as the input connected to Echo is 0. We then record the kernel time when the rising edge occurred.

We wait as long as the input connected to Echo is 1. We then record the kernel time when the falling edge happened.

6.) read

```
ssize_t hcsr04_read(struct file *filp, char __user *buf, size_t count, loff_t *f_pos)
{
  int ret;
  int pulse;

  pulse = (int)ktime_to_us( ktime_sub( falling, rising ) );

  ret = copy_to_user( buf, &pulse, 4 );

  return 4;
}
```

The pulse duration is computed subtracting rising time from falling time, and then translating the result in µs.
`ktime_sub()` and `ktime_to_us()` are used to handle the specific data structure used to store the Kernel time measured using `ktime_get()`.

We provide to the user space the four bytes storing the pulse duration represented as integer number.

Four bytes are read.

7.) show

```
static ssize_t hcsr04_show(struct kobject *kobj,
                           struct kobj_attribute *attr,
                           char *buf)
{
  return sprintf(buf, "%d\n", ktime_to_us(ktime_sub(falling,rising)));
}
```

execute ISR0 user reads
/sys/kernel/hcsr04/hcsr04

8.) store

```
static ssize_t hcsr04_store(struct kobject *kobj,
                            struct kobj_attribute *attr,
                            char *buf,
                            size_t count)
{
  return 1;
}
```

execute ISR0 user writes
/sys/kernel/hcsr04/hcsr04

Module test Application

```
int main(int argc, char **argv)
{
  char *app_name = argv[0];
  char *dev_name = "/dev/hcsr04";
  int fd = -1;
  char c;
  int d;

  fd = open(dev_name, O_RDWR);
  c = 1;
  write( fd, &c, 1 );
  read( fd, &d, 4 );

  printf( "%d: %f\n", d, d/58.0 );
  close( fd );
  return 0;
}
```

We assume that the module is loaded and that the /dev/hcsr04 device file has been created.

We open the device file.

We trigger the sensor by executing the write system call. The written value is meaningless.

We read the four bytes storing the echo pulse duration.

We display the duration and the corresponding distance.

We close the device file.

Quiz

1. "Which function adds a "kobject" data structure to a 'sysfs' file system?"

- ○ "kobject_store()"
- ✔ "kobject_create_and_add()"
- ○ "kobject_create()"
- ○ "kobject_aggregate()"

2. "With regard to the "HC-SR04 Ultrasonic Ranging Sensor – which connector is for TTL input which is used to trigger the operation of the sensor?"

- ○ Echo
- ○ Vcc
- ✔ Trig
- ○ GND

3. "The sensor will readout the distance as a pulse from the.."

- ✔ Echo connector
- ○ Vcc connector
- ○ Trig connector
- ○ GND connector

4. "In the implementation of support for the HC-SR04 Sensor in module 7 – What was the write() system call used for?"

- ○ Provide the user level with the measured echo pulse duration in microseconds
- ✔ Trigger the sensor and measure the echo pulse duration
- ○ Output a pulse duration measurement to the sensor
- ○ Trigger the sensor then deactivate

5. "In regard to the sysfs file system – how can a GPIO number (105) be exported via the command line?"

- ○ "echo 105 > /sys/class/gpio/gpio105"
- ○ "echo "out" > /sys/class/gpio/gpio105"
- ○ "echo "out" > /sys/class/gpio/gpio105/direction"
- ✔ "echo 105 > /sys/class/gpio/export"