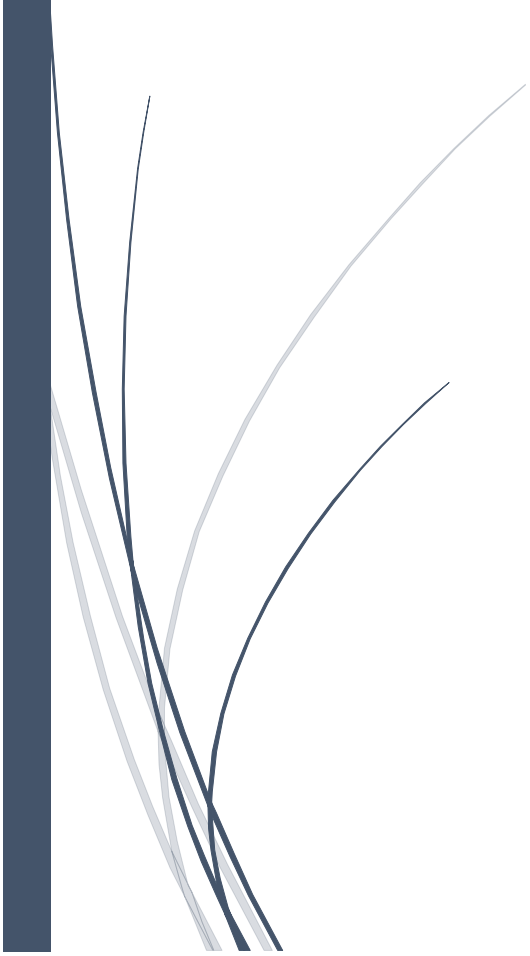
10/29/2016

Safaa Hassan Wally [35]

Lab 2 : Synchronization in Linux Kernel



1.organization the code:

The project has two classes first one the Kernel Space and the User Space

The User Space:

In this class takes the command as a parameter and execute it by opening and writing into the files which is related to kernel like num/caps/scroll .

The Kernel Space:

It divide into some functions :

1- Functions for creating k-object and its attributes (num/scroll/caps).

2-Functions for reading and writing data into keyboard ports.

3-Functions related to the User Space like : set_led_state and get_led_state.

2.Main functions:

1-Function Reading data and writing data from ports :

- Kbd_read_status ()
- Kbd_read_data ()
- Kbd_write_data ()

2-Function to create the kernel object:

- __Init example_init () >> create the kernel

```
static int __init example_init(void)
{
    int retval;
    example_kobj = kobject_create_and_add("kobject_safaa", kernel_kobj);
    if (!example_kobj)
        return -ENOMEM;
    printk("Create k_object");
    /* Create the files associated with this kobject */
    retval = sysfs_create_group(example_kobj, &attr_group);
    if (retval)
        kobject_put(example_kobj);
    init();
    return retval;
}
```

- attribute_group attr_group ()>> create the file related to the kernel module like num/caps/scroll

- child_store () >> used for store the new state from the user and calls function set_led_state()

```
// Generalization function for the rest button(SCROLL_LOCK/CAPS_LOCK) to set t
static ssize_t child_store(struct kobject *kobj, struct kobj_attribute *attr,
                           const char *buf, size_t count)
{
    int ret,var;
    up(&sema);
    ret = kstrtoint(buf, 10, &var);
    if (ret < 0)
        return ret;

    if (strcmp(attr->attr.name, "caps") == 0)
    {
        caps = var;
        set_led_state(2,var);

    }
    else if(strcmp(attr->attr.name, "scroll") == 0)
    {
        scroll = var;
        set_led_state(0,var);

    }
    down(&sema);
    return count;
}
```

- child_show () >> used to get the state of the led this function calls get_led_state()

```
// Generalization function of the rest button (SCROLL_LOCK/CAPS_LOCK) to get the
static ssize_t child_show(struct kobject *kobj, struct kobj_attribute *attr,
                          char *buf)
{
    int var;
    up(&sema);
    if (strcmp(attr->attr.name, "caps") == 0)
    {
        var = get_led_state(2);
    }
    else
    {
        var = get_led_state(0);
    }
    down(&sema);
    return sprintf(buf, "%d\n", var);
}
```

3-Functions for the user space module :

- get_led_state () >> this function get the state stored in the kernel by the last set operation happened to the led initialization for all the leds to be zero

- set_led_state () >> make bitwise and ot bitwise or to the general led state according to the state value to keep the states of leds.

```
void set_led_state(int led, int state)
{
    // Set the new state of the led
    printk(" process[%d] 1 start set function\n",current->pi
    if(state)
    {
        //If state is one do bitwise or the old led_state to
        switch(led)
        {
        case 0:
            led_status = led_status | LED_SCROLL_LOCK;
            break;
        case 1:
            led_status = led_status | LED_NUM_LOCK;
            break;
        case 2:
            led_status = led_status | LED_CAPS_LOCK;
            break;
        }
    }
    else
    {
        // If state is zero do bitwise and with
        switch(led)
        {
        case 0:
            led_status =led_status & LED_SCROLL_LOCK_OFF;
            break;
        case 1:
            led_status  = led_status & LED_NUM_LOCK_OFF;
            break;
        case 2:
            led_status= led_status & LED_CAPS_LOCK_OFF;
            break;
        }
    }
    printk(" process[%d] 2 will update in set\n",current->pid);
    update_leds(led_status);
    printk(" process[%d] 11 finish update in set\n",current->pid);
}
```

- update_leds () >> wait acknowledge for writing data in ports of keyboard

```
int update_leds(unsigned char led_status_word)
{
    printk(" process[%d] 3 start update function\n",current->pid);
    // Disabling the interruption of keyboard
    disable_irq(1);
    printk(" process[%d] 4 Disable irq\n",current->pid);
    // send 'Set LEDs' command
    kbd_write_data(KBD_CMD_SET_LEDS);
    printk(" process[%d] 5 sent set leds \n",current->pid);
    msleep(500);
    // wait for ACK
    if (kbd_read_data() != KBD_REPLY_ACK )
    {
        enable_irq(1);
        return -1;
    }
    printk(" process[%d] 6 wait for first ACK\n",current->pid);
    // now send LED states
    kbd_write_data(led_status_word);
    printk(" process[%d] 7 send led states\n",current->pid);
    // wait for ACK
    if (kbd_read_data() != KBD_REPLY_ACK )
    {
        enable_irq(1);
        return -1;
    }
    printk(" process[%d] 8 wait for second ACK\n",current->pid);
    // success and enable the interruption
    enable_irq(1);
    printk(" process[%d] 9 success \n",current->pid);
    printk(" process[%d] 10 end update function\n",current->pid);
    return 0;
}
```

4.Semaphore function:

- Init():

3.Running and compiling the code:

There is a make file in the folder KernelSpace for kernel mode and a make file for the UserSpace for user mode.

And scripts to run the cd the directories in my folder to execute make

You should first change to the host then

First compile the kernel Space t :

cd path for my folder

./scriptkernel.sh

It will compile the kernel space >> [KernelSpace.ko]

Second compile the User Space :

cd path for my folder

 ./scriptuser.sh

It will compile the user space >> [leds]

 You can added the tests to this scripts

Example ./leds set caps on

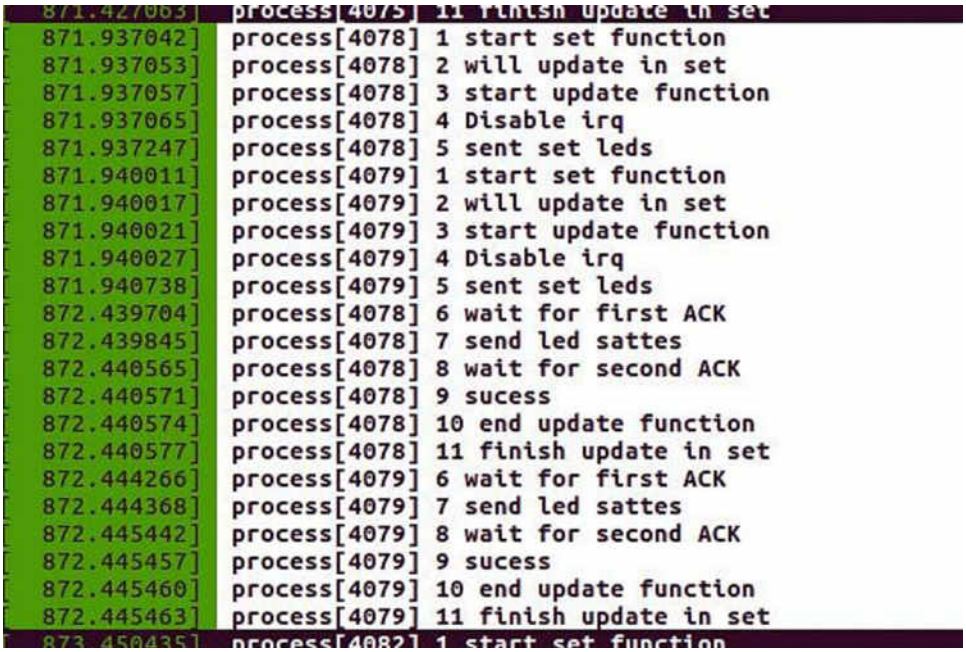
4.Sample runs:

Race condition:

In the two examples the process didn't finished and it was interrupted by another process and this led to race condition and the results depends on the relative ordering of executing.

For example:

Process with id 4078 was interrupted by process 4079 .



```
855.778768] process[4025] 1 start set function
855.778777] process[4025] 2 will update in set
855.778781] process[4025] 3 start update function
855.778787] process[4025] 4 Disable irq
855.779297] process[4025] 5 sent set leds
856.257103] process[4024] 6 wait for first ACK
856.257178] process[4024] 7 send led sattes
856.258577] process[4024] 8 wait for second ACK
856.258582] process[4024] 9 sucess
856.258584] process[4024] 10 end update function
856.258586] process[4024] 11 finish update in set
856.281142] process[4025] 6 wait for first ACK
856.281319] process[4025] 7 send led sattes
856.281601] process[4025] 8 wait for second ACK
856.281629] process[4025] 9 sucess
856.281633] process[4025] 10 end update function
856.281636] process[4025] 11 finish update in set
856.765420] process[4028] 1 start set function
856.765428] process[4028] 2 will update in set
856.765432] process[4028] 3 start update function
856.765438] process[4028] 4 Disable irq
856.765712] process[4028] 5 sent set leds
856.790314] process[4029] 1 start set function
856.790323] process[4029] 2 will update in set
856.790327] process[4029] 3 start update function
856.790332] process[4029] 4 Disable irq
856.790718] process[4029] 5 sent set leds
857.268751] process[4028] 6 wait for first ACK
857.268830] process[4028] 7 send led sattes
857.269988] process[4028] 8 wait for second ACK
857.269995] process[4028] 9 sucess
```

After adding semaphores:

Every process wasn’t interrupted until it finished.

```
[  285.545433] Create k_object
[  285.687177]  process[3408] 1 start set function
[  285.687180]  process[3408] 2 will update in set
[  285.687181]  process[3408] 3 start update function
[  285.687182]  process[3408] 4 Disable irq
[  285.687661]  process[3408] 5 sent set leds
[  286.190461]  process[3408] 6 wait for first ACK
[  286.190583]  process[3408] 7 send led sattes
[  286.190996]  process[3408] 8 wait for second ACK
[  286.191012]  process[3408] 9 sucess
[  286.191014]  process[3408] 10 end update function
[  286.191016]  process[3408] 11 finish update in set
[  286.695846]  process[3410] 1 start set function
[  286.695853]  process[3410] 2 will update in set
[  286.695856]  process[3410] 3 start update function
[  286.695861]  process[3410] 4 Disable irq
[  286.696495]  process[3410] 5 sent set leds
[  287.197931]  process[3410] 6 wait for first ACK
[  287.197995]  process[3410] 7 send led sattes
[  287.221594]  process[3410] 8 wait for second ACK
[  287.221599]  process[3410] 9 sucess
[  287.221600]  process[3410] 10 end update function
[  287.221600]  process[3410] 11 finish update in set
[  287.724636]  process[3412] 1 start set function
[  287.724642]  process[3412] 2 will update in set
[  287.724645]  process[3412] 3 start update function
[  287.724648]  process[3412] 4 Disable irq
[  287.725066]  process[3412] 5 sent set leds
[  288.225582]  process[3412] 6 wait for first ACK
[  288.225876]  process[3412] 7 send led sattes
[  288.225962]  process[3412] 8 wait for second ACK
[  288.225980]  process[3412] 9 sucess
[  288.225983]  process[3412] 10 end update function
[  288.225985]  process[3412] 11 finish update in set
[  288.742154]  process[3414] 1 start set function
[  288.742163]  process[3414] 2 will update in set
[  288.742168]  process[3414] 3 start update function
[  288.742199]  process[3414] 4 Disable irq
[  288.742535]  process[3414] 5 sent set leds
[  289.245413]  process[3414] 6 wait for first ACK
[  289.245556]  process[3414] 7 send led sattes
[  289.246480]  process[3414] 8 wait for second ACK
[  289.246500]  process[3414] 9 sucess
[  289.246503]  process[3414] 10 end update function
[  289.246506]  process[3414] 11 finish update in set
[  289.753838]  process[3416] 1 start set function
[  289.753847]  process[3416] 2 will update in set
[  289.753851]  process[3416] 3 start update function
[  289.753857]  process[3416] 4 Disable irq
[  289.754135]  process[3416] 5 sent set leds
[  290.257082]  process[3416] 6 wait for first ACK
[  290.257327]  process[3416] 7 send led sattes
[  290.257907]  process[3416] 8 wait for second ACK
```

General Runs :

```
root@ubuntu:/home/safaa/Desktop/Lab2_35/User# ./leds set caps on

root@ubuntu:/home/safaa/Desktop/Lab2_35/User#
root@ubuntu:/home/safaa/Desktop/Lab2_35/User# ./leds get caps
ON
root@ubuntu:/home/safaa/Desktop/Lab2_35/User# ./leds set num on

root@ubuntu:/home/safaa/Desktop/Lab2_35/User#
root@ubuntu:/home/safaa/Desktop/Lab2_35/User# ./leds get num
ON
root@ubuntu:/home/safaa/Desktop/Lab2_35/User# ./leds set scroll on

root@ubuntu:/home/safaa/Desktop/Lab2_35/User#
root@ubuntu:/home/safaa/Desktop/Lab2_35/User# ./leds get scroll
ON
```