1.Create and load a basic LKM into the Linux kernel, which prints a message when loaded and unloaded.

```
#include <linux/module.h>   // Needed by all modules

#include <linux/kernel.h>   // Needed for KERN_INFO

#include <linux/init.h>     // Needed for the macros

// Module metadata

MODULE_LICENSE("GPL");

MODULE_AUTHOR("Your Name");

MODULE_DESCRIPTION("A simple Hello World kernel module");

MODULE_VERSION("1.0");

// Initialization function

static int __init hello_init(void) {

    printk(KERN_INFO "Hello, world!\n");

    return 0; // Success

}

// Cleanup function

static void __exit hello_exit(void) {

    printk(KERN_INFO "Goodbye, world!\n");

}

module_init(hello_init);

module_exit(hello_exit);
```

Makefile:

```
obj-m += hello.o

all:

	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:

	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

4.Create and load an LKM that accepts parameters into the Linux kernel, and observe how parameter values affect the LKM's behavior.

```
#include <linux/module.h>
#include <linux/moduleparam.h>
#include <linux/init.h>
#include <linux/kernel.h>

static int irq=10;
module_param(irq,int,0660);

static int debug=0;
module_param(debug,int,0660);

static char *devname = "simpdev";
module_param(devname,charp,0660);

static int addr[10];
static int count;
module_param_array(addr, int, &count,  0660);

static int simple_init(void)
{
        int i;
        printk(KERN_WARNING "hello... irq=%d name=%s debug=%d\n",irq,devname,debug);
        for(i=0;i<count;i++)
                printk(KERN_WARNING "hello... addr=%d \n",addr[i]);
        return 0;
}
static void simple_cleanup(void)
{
        printk(KERN_WARNING "bye... irq=%d name=%s debug=%d\n",irq,devname,debug);
}


MODULE_LICENSE("GPL");
module_init(simple_init);
module_exit(simple_cleanup);
```

Makefile

```
obj-m := lkm-para.o

KERNELDIR ?= /lib/modules/$(shell uname -r)/build
PWD       := $(shell pwd)

all: default

default:
      $(MAKE) -C $(KERNELDIR) M=$(PWD) modules

clean:
      rm -rf *.o *~ core .depend .*.cmd *.ko *.mod.c .tmp_versions
```

5.Create an LKM that generates a /proc file containing the PIDs and names of all running processes.

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/proc_fs.h>
#include <linux/sched.h>
#include <linux/uaccess.h>


#define PROC_FILENAME "task_details"


static struct proc_dir_entry *proc_file_entry;


static ssize_t proc_file_read(struct file *file, char __user *user_buffer, size_t count, loff_t *position) {
    struct task_struct *task;
    char buffer[1000];
    char *buf_ptr = buffer;
    int len = 0;

    for_each_process(task) {
      if( len < (sizeof(buffer) - 50)){
        len += snprintf(buf_ptr + len, sizeof(buffer) - len,
                        "PID: %d, Name: %s\n",
                        task->pid, task->comm);
    }}

    if (copy_to_user(user_buffer, buffer, len)) {
        return -EFAULT;
    }

    return len;
}

static const struct proc_ops proc_file_fops = {
    .proc_read = proc_file_read,
};

static int __init task_details_lkm_init(void) {
    proc_file_entry = proc_create(PROC_FILENAME, 0, NULL, &proc_file_fops);
    if (!proc_file_entry) {
        printk(KERN_ERR "Failed to create proc file\n");
        return -ENOMEM;
    }

    printk(KERN_INFO "Proc file created\n");
    return 0;
}
```

```
static void __exit task_details_lkm_exit(void) {
    if (proc_file_entry) {
        remove_proc_entry(PROC_FILENAME, NULL);
        printk(KERN_INFO "Proc file removed\n");
    }
}

module_init(task_details_lkm_init);
module_exit(task_details_lkm_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Your Name");
MODULE_DESCRIPTION("LKM for printing task details in a proc file");



#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/proc_fs.h>
#include <linux/sched.h>
#include <linux/uaccess.h>

static int __init task_details_lkm_init(void) {
     struct task_struct *task;

    for_each_process(task)
            printk(KERN_INFO "Current Pid=%d,Nice=%d, Normal Priority=%d, state = %d, and Name: %s\n",task->pid, PRIO_TO_NICE((task)->static_prio), task->normal_prio,  task->__state, task->comm);

    printk(KERN_INFO "LKM initialized\n");
```

```
   return 0;
}

static void __exit task_details_lkm_exit(void) {
       printk(KERN_INFO "Exiting LKM...\n");
   printk(KERN_INFO "LKM exited\n");
}

module_init(task_details_lkm_init);
module_exit(task_details_lkm_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Your Name");
MODULE_DESCRIPTION("LKM for printing task details in a proc file");
```

Makefile

```
CONFIG_MODULE_SIG=n

ifneq	($(KERNELRELEASE),)
obj-m	:= lab1.o

else
KDIR	:= /lib/modules/$(shell uname -r)/build
PWD	:= $(shell pwd)
default:
	$(MAKE) HOSTCC=x86_64-linux-gnu-gcc-11 CC=x86_64-linux-gnu-gcc-11 -C $(KDIR) M=$(PWD) modules
	rm -r -f .tmp_versions *.mod.c .*.cmd *.o *.symvers

endif
```

6.Create an LKM that changes the priority of a specific process identified by its PID.

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

#include <linux/kernel.h>

#include <linux/proc_fs.h>

#include <linux/sched.h>

#include <linux/uaccess.h>

static int pid_to_change = -1; // The PID of the process to change

static int new_priority = 100; // The new priority value (adjust as needed)

module_param(pid_to_change, int, S_IRUGO); // Accept PID as a module parameter

module_param(new_priority, int, S_IRUGO);  // Accept new priority as a module parameter

static int __init task_details_lkm_init(void) {

    struct task_struct *task;

    if (pid_to_change <= 0 || new_priority < -20 || new_priority > 19) {

       printk(KERN_ERR "Invalid parameters\n");

       return -EINVAL;

   }

   printk(KERN_INFO "Initializing LKM for CPU scheduling...\n");

   for_each_process(task) {

     if (task->pid == pid_to_change) {

         // Change the priority of the specified process

         set_user_nice(task, new_priority);

         printk(KERN_INFO "Changed priority of PID %d to %d\n", pid_to_change, new_priority);

         break;

      }

    }


   printk(KERN_INFO "LKM initialized\n");

   return 0;

}
```

```
static void __exit task_details_lkm_exit(void) {

    printk(KERN_INFO "Exiting LKM...\n");

  printk(KERN_INFO "LKM exited\n");

}

module_init(task_details_lkm_init);

module_exit(task_details_lkm_exit);

MODULE_LICENSE("GPL");

MODULE_AUTHOR("Your Name");

MODULE_DESCRIPTION("LKM for printing task details in a proc file");
```

Makefille

```
CONFIG_MODULE_SIG=n

ifneq	($(KERNELRELEASE),)
obj-m	:= lab1.o

else
KDIR	:= /lib/modules/$(shell uname -r)/build
PWD	:= $(shell pwd)
default:
	$(MAKE) HOSTCC=x86_64-linux-gnu-gcc-12 CC=x86_64-linux-gnu-gcc-12 -C $(KDIR) M=$(PWD) modules
	rm -r -f .tmp_versions *.mod.c .*.cmd *.o *.symvers

endif
```

8.Create an LKM that that will display a list of only those tasks which are ‘kernel threads’? (i.e., task->mm == 0). How many ‘kernel threads’ on your list?

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/sched/signal.h>   // for each_process
#include <linux/sched.h>          // for task_struct

MODULE_LICENSE("GPL");
MODULE_AUTHOR("ChatGPT");
MODULE_DESCRIPTION("A kernel module to list all kernel threads (task->mm == NULL)");
MODULE_VERSION("1.0");

static int __init kernel_threads_init(void) {
    struct task_struct *task;
    int count = 0;

    printk(KERN_INFO "Listing all kernel threads (task->mm == NULL):\n");

    for_each_process(task) {
        if (task->mm == NULL) {
            printk(KERN_INFO "PID: %d | Name: %s\n", task->pid, task->comm);
            count++;
        }
    }

    printk(KERN_INFO "Total Kernel Threads: %d\n", count);
    return 0;
}
```

```
static void __exit kernel_threads_exit(void) {
    printk(KERN_INFO "Kernel Threads LKM unloaded.\n");
}

module_init(kernel_threads_init);
module_exit(kernel_threads_exit);
```

Makefile

```
obj-m += kernel_threads.o
all:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
clean:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

All processes and tasks:

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/sched/signal.h>   // for_each_process
#include <linux/sched.h>          // task_struct

MODULE_LICENSE("GPL");
MODULE_AUTHOR("ChatGPT");
MODULE_DESCRIPTION("List kernel threads and total processes");
MODULE_VERSION("1.1");

static int __init kernel_threads_init(void) {
    struct task_struct *task;
    int kernel_thread_count = 0;
    int total_process_count = 0;
    printk(KERN_INFO "Listing all kernel threads (task->mm == NULL):\n");

    for_each_process(task) {
        total_process_count++;
        if (task->mm == NULL) {
            printk(KERN_INFO "PID: %d | Name: %s\n", task->pid, task->comm);
            kernel_thread_count++;
        }
    }

    printk(KERN_INFO "Total Processes: %d\n", total_process_count);
    printk(KERN_INFO "Total Kernel Threads: %d\n", kernel_thread_count);

    return 0;
}

static void __exit kernel_threads_exit(void) {
    printk(KERN_INFO "Kernel Threads LKM unloaded.\n");
}

module_init(kernel_threads_init);
module_exit(kernel_threads_exit);
```