2. Demonstrate how to avoid the "zombie" state by properly waiting for child processes to exit in a parent process.

nano avoid_zombie.c

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

int main() {
    pid_t pid = fork();

    if (pid < 0) {
        perror("Fork failed");
        return 1;
    }
    else if (pid == 0) {
        // Child process
        printf("Child: My PID is %d\n", getpid());
        sleep(2);
        printf("Child: Exiting...\n");
        exit(0);
    }
    else {
        // Parent process
        printf("Parent: My PID is %d, waiting for child...\n", getpid());
        int status;
        waitpid(pid, &status, 0);  // Properly wait for the child
        printf("Parent: Child exited. No zombie created.\n");
    }

    return 0;
}
```

gcc avoid_zombie.c -o avoid_zombie

./avoid_zombie

9. **Create an LKM that prints information about a specific process, including its PID, resident set size (RSS), virtual memory size (VSZ), and command name.**

```
sudo apt update
sudo apt install build-essential linux-headers-$(uname -r)


mkdir ~/lkm_process_info
cd ~/lkm_process_info
```

nano process_info.c

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/sched/signal.h>
#include <linux/mm.h>
#include <linux/pid.h>

static int pid = -1;
module_param(pid, int, 0);
MODULE_PARM_DESC(pid, "Process ID");

static int __init process_info_init(void)
{
    struct task_struct *task;
    struct mm_struct *mm;

    if (pid == -1) {
        printk(KERN_INFO "No PID provided.\n");
        return -1;
    }

    task = pid_task(find_vpid(pid), PIDTYPE_PID);
    if (!task) {
        printk(KERN_INFO "Process with PID %d not found.\n", pid);
        return -1;
    }

    mm = task->mm;

    if (mm) {
        unsigned long rss = get_mm_rss(mm) << PAGE_SHIFT;
        unsigned long vsz = mm->total_vm << PAGE_SHIFT;

        printk(KERN_INFO "=== Process Info ===\n");
        printk(KERN_INFO "PID: %d\n", pid);
        printk(KERN_INFO "Comm: %s\n", task->comm);
        printk(KERN_INFO "RSS: %lu KB\n", rss / 1024);
        printk(KERN_INFO "VSZ: %lu KB\n", vsz / 1024);
    } else {
        printk(KERN_INFO "Process has no memory descriptor (kernel thread?)\n");
    }

    return 0;
}

static void __exit process_info_exit(void)
{
    printk(KERN_INFO "Process info module unloaded.\n");
}
```

```
MODULE_LICENSE("GPL");
MODULE_AUTHOR("YourName");
MODULE_DESCRIPTION("LKM to print process info by PID");

module_init(process_info_init);
module_exit(process_info_exit);
```

nano Makefile

```
obj-m += process_info.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
```

make

ps aux | head

sudo insmod process_info.ko pid=1234

dmesg | tail -20

10. Create an LKM using kmalloc for allocating memory for the struct process_info objects that initializes, tracks, and prints information about running processes.

sudo apt install build-essential linux-headers-$(uname -r)

mkdir ~/lkm_kmalloc_process
cd ~/lkm_kmalloc_process

nano kmalloc_process.c

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

#define MAX_PROCESSES 1024

struct process_info {
    pid_t pid;
    char comm[TASK_COMM_LEN];
    long state;
};
```

```
static struct process_info *proc_info_array = NULL;

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

    printk(KERN_INFO "[kmalloc_process] Module loading...\n");

    // Allocate memory for MAX_PROCESSES
    proc_info_array = kmalloc_array(MAX_PROCESSES, sizeof(struct process_info),
GFP_KERNEL);
    if (!proc_info_array) {
        printk(KERN_ERR "[kmalloc_process] Memory allocation failed!\n");
        return -ENOMEM;
    }

    for_each_process(task) {
        if (count >= MAX_PROCESSES)
            break;

        proc_info_array[count].pid = task->pid;
        proc_info_array[count].state = task->state;
        strncpy(proc_info_array[count].comm, task->comm, TASK_COMM_LEN);
        count++;
    }

    printk(KERN_INFO "[kmalloc_process] Process list:\n");
    for (int i = 0; i < count; i++) {
        printk(KERN_INFO "[PID: %d] [Name: %s] [State: %ld]\n",
               proc_info_array[i].pid,
               proc_info_array[i].comm,
               proc_info_array[i].state);
    }

    printk(KERN_INFO "[kmalloc_process] Loaded successfully with %d processes tracked.\n",
count);
    return 0;
}

static void __exit kmalloc_process_exit(void) {
    printk(KERN_INFO "[kmalloc_process] Module unloading...\n");

    if (proc_info_array) {
        kfree(proc_info_array);
        printk(KERN_INFO "[kmalloc_process] Memory freed.\n");
    }

    printk(KERN_INFO "[kmalloc_process] Unloaded successfully.\n");
}

module_init(kmalloc_process_init);
module_exit(kmalloc_process_exit);
```

```
MODULE_LICENSE("GPL");
MODULE_AUTHOR("YourName");
MODULE_DESCRIPTION("LKM using kmalloc to track process info");
```

nano Makefile

```
obj-m += kmalloc_process.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
```

make

sudo insmod kmalloc_process.ko

dmesg | tail -n 50

11.Create an LKM that allocates memory using both kmalloc and vmalloc, and then compare their characteristics.

sudo apt-get install linux-headers-$(uname -r) build-essential

mkdir ~/lkm_mem_compare
cd ~/lkm_mem_compare

nano mem_compare.c

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/slab.h>     // For kmalloc and kfree
#include <linux/vmalloc.h>  // For vmalloc and vfree

#define ALLOC_SIZE (1024 * 1024)  // 1 MB

static char *kmalloc_ptr;
static char *vmalloc_ptr;

static int __init mem_compare_init(void)
{
    printk(KERN_INFO "mem_compare: Module init\n");

    // Allocate memory using kmalloc
    kmalloc_ptr = kmalloc(ALLOC_SIZE, GFP_KERNEL);
    if (!kmalloc_ptr) {
        printk(KERN_ERR "mem_compare: kmalloc failed\n");
```

```
        return -ENOMEM;
    }
    printk(KERN_INFO "mem_compare: kmalloc allocated at %p\n", kmalloc_ptr);

    // Allocate memory using vmalloc
    vmalloc_ptr = vmalloc(ALLOC_SIZE);
    if (!vmalloc_ptr) {
        printk(KERN_ERR "mem_compare: vmalloc failed\n");
        kfree(kmalloc_ptr);
        return -ENOMEM;
    }
    printk(KERN_INFO "mem_compare: vmalloc allocated at %p\n", vmalloc_ptr);

    return 0;
}

static void __exit mem_compare_exit(void)
{
    printk(KERN_INFO "mem_compare: Module exit\n");

    if (kmalloc_ptr) {
        kfree(kmalloc_ptr);
        printk(KERN_INFO "mem_compare: kmalloc memory freed\n");
    }

    if (vmalloc_ptr) {
        vfree(vmalloc_ptr);
        printk(KERN_INFO "mem_compare: vmalloc memory freed\n");
    }
}

module_init(mem_compare_init);
module_exit(mem_compare_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("ChatGPT");
MODULE_DESCRIPTION("LKM that compares kmalloc and vmalloc");
```

nano Makefile

```
obj-m += mem_compare.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
```

make

sudo insmod mem_compare.ko

dmesg | tail -20

12. Load the LKM into the Linux kernel, observe two threads running concurrently and using a mutex for synchronization.

mkdir ~/lkm_thread_mutex
cd ~/lkm_thread_mutex

nano thread_mutex.c

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/kthread.h>    // For kthreads
#include <linux/delay.h>      // For msleep
#include <linux/mutex.h>      // For mutex

MODULE_LICENSE("GPL");
MODULE_AUTHOR("ChatGPT");
MODULE_DESCRIPTION("Two concurrent kernel threads synchronized by a mutex");

static struct task_struct *thread1;
static struct task_struct *thread2;

static DEFINE_MUTEX(my_mutex);
static int run_threads = 1;

// Thread function 1
static int thread_fn1(void *data)
{
    int count = 0;
    while (!kthread_should_stop() && run_threads) {
        mutex_lock(&my_mutex);
        printk(KERN_INFO "Thread 1 acquired mutex, count=%d\n", count++);
        msleep(500);  // simulate work while holding the mutex
        mutex_unlock(&my_mutex);
        msleep(500);  // simulate work outside mutex
    }
    printk(KERN_INFO "Thread 1 stopping\n");
    return 0;
}

// Thread function 2
static int thread_fn2(void *data)
{
```

```
    int count = 0;
    while (!kthread_should_stop() && run_threads) {
        mutex_lock(&my_mutex);
        printk(KERN_INFO "Thread 2 acquired mutex, count=%d\n", count++);
        msleep(700);  // simulate work while holding the mutex
        mutex_unlock(&my_mutex);
        msleep(700);  // simulate work outside mutex
    }
    printk(KERN_INFO "Thread 2 stopping\n");
    return 0;
}

static int __init thread_mutex_init(void)
{
    printk(KERN_INFO "thread_mutex: Module init\n");

    // Start thread1
    thread1 = kthread_run(thread_fn1, NULL, "thread1");
    if (IS_ERR(thread1)) {
        printk(KERN_ERR "Failed to create thread1\n");
        return PTR_ERR(thread1);
    }

    // Start thread2
    thread2 = kthread_run(thread_fn2, NULL, "thread2");
    if (IS_ERR(thread2)) {
        printk(KERN_ERR "Failed to create thread2\n");
        kthread_stop(thread1);
        return PTR_ERR(thread2);
    }

    return 0;
}

static void __exit thread_mutex_exit(void)
{
    printk(KERN_INFO "thread_mutex: Module exit\n");

    run_threads = 0;

    if (thread1) {
        kthread_stop(thread1);
        printk(KERN_INFO "thread1 stopped\n");
    }

    if (thread2) {
        kthread_stop(thread2);
        printk(KERN_INFO "thread2 stopped\n");
    }
}

module_init(thread_mutex_init);
```

```
module_exit(thread_mutex_exit);
```

nano Makefile

```
obj-m += thread_mutex.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
```

make

sudo insmod thread_mutex.ko

dmesg –follow

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

sudo apt update

sudo apt install build-essential linux-headers-$(uname -r)

mkdir ~/basic_lkm

cd ~/basic_lkm

gedit hello_lkm.c

#include <linux/init.h>

#include <linux/module.h>

#include <linux/kernel.h>

```
MODULE_LICENSE("GPL");
MODULE_AUTHOR("YourName");
MODULE_DESCRIPTION("A simple Hello World LKM");
MODULE_VERSION("1.0");

static int __init hello_lkm_init(void) {
    printk(KERN_INFO "Hello: LKM loaded into the kernel\n");
    return 0;
}

static void __exit hello_lkm_exit(void) {
    printk(KERN_INFO "Goodbye: LKM unloaded from the kernel\n");
}

module_init(hello_lkm_init);
module_exit(hello_lkm_exit);
```

gedit Makefile

```
obj-m += hello_lkm.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
```

make

sudo insmod hello_lkm.ko

dmesg | tail -n 10

Hello: LKM loaded into the kernel

sudo rmmod hello_lkm

dmesg | tail -n 10

Goodbye: LKM unloaded from the kernel

make clean

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

```
sudo apt update
sudo apt install build-essential linux-headers-$(uname -r)
```

```
mkdir ~/my_lkm
cd ~/my_lkm
```

```
gedit param_lkm.c
```

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

MODULE_LICENSE("GPL");
MODULE_AUTHOR("YourName");
MODULE_DESCRIPTION("A simple Linux driver with parameters");
MODULE_VERSION("1.0");

// Declare module parameters
static int myint = 0;
module_param(myint, int, 0660);
MODULE_PARM_DESC(myint, "An integer");

static char *mystring = "default";
module_param(mystring, charp, 0660);
```

```
 MODULE_PARM_DESC(mystring, "A string");

// Init and Exit functions
static int __init param_lkm_init(void) {
    printk(KERN_INFO "param_lkm: Module loaded\n");
    printk(KERN_INFO "param_lkm: myint = %d, mystring = %s\n", myint, mystring);
    return 0;
}

static void __exit param_lkm_exit(void) {
    printk(KERN_INFO "param_lkm: Module unloaded\n");
}

module_init(param_lkm_init);
module_exit(param_lkm_exit);
```

```
gedit Makefile
obj-m += param_lkm.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
```

```
make
```

```
sudo insmod param_lkm.ko myint=42 mystring="hello_kernel"
```

```
dmesg | tail -n 20
```

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

```
sudo apt update
sudo apt install build-essential linux-headers-$(uname -r)
```

```
mkdir ~/proc_lkm
cd ~/proc_lkm
```

```
gedit proc_lkm.c
```

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/proc_fs.h>
#include <linux/seq_file.h>
#include <linux/sched/signal.h>

MODULE_LICENSE("GPL");
```

```
MODULE_AUTHOR("YourName");
MODULE_DESCRIPTION("LKM that lists all process names and PIDs in /proc/process_list");
MODULE_VERSION("1.0");

#define PROC_NAME "process_list"

static int show_processes(struct seq_file *m, void *v) {
    struct task_struct *task;

    seq_printf(m, "PID\tProcess Name\n");
    for_each_process(task) {
        seq_printf(m, "%d\t%s\n", task->pid, task->comm);
    }

    return 0;
}

static int proc_open(struct inode *inode, struct file *file) {
    return single_open(file, show_processes, NULL);
}

static const struct proc_ops proc_file_ops = {
    .proc_open    = proc_open,
    .proc_read    = seq_read,
    .proc_lseek   = seq_lseek,
    .proc_release = single_release,
};
```

```
static int __init proc_lkm_init(void) {
    proc_create(PROC_NAME, 0, NULL, &proc_file_ops);
    printk(KERN_INFO "/proc/%s created\n", PROC_NAME);
    return 0;
}

static void __exit proc_lkm_exit(void) {
    remove_proc_entry(PROC_NAME, NULL);
    printk(KERN_INFO "/proc/%s removed\n", PROC_NAME);
}

module_init(proc_lkm_init);
module_exit(proc_lkm_exit);
```

```
gedit Makefile
```

```
obj-m += proc_lkm.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
```

```
make
```

```
sudo insmod proc_lkm.ko
```

dmesg | tail -n 5

cat /proc/process_list

sudo rmmod proc_lkm

dmesg | tail -n 5

ls /proc/process_list

ls /proc/process_list

make clean

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

sudo apt update

sudo apt install build-essential linux-headers-$(uname -r)

mkdir ~/priority_lkm

cd ~/priority_lkm

gedit priority_lkm.c

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

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

MODULE_LICENSE("GPL");
MODULE_AUTHOR("YourName");
MODULE_DESCRIPTION("LKM to change process priority by PID");
MODULE_VERSION("1.0");

static int pid = -1;
static int new_nice = 0;

module_param(pid, int, 0644);
MODULE_PARM_DESC(pid, "PID of the process to modify");

module_param(new_nice, int, 0644);
MODULE_PARM_DESC(new_nice, "New nice value (priority) for the process");

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

    if (pid <= 0 || new_nice < -20 || new_nice > 19) {
        printk(KERN_ERR "Invalid PID or nice value (must be between -20 and 19)\n");
        return -EINVAL;
    }

    for_each_process(task) {
        if (task->pid == pid) {
            printk(KERN_INFO "Found process: %s (PID: %d), current nice: %d\n",
                task->comm, task->pid, task_nice(task));
```

```
            set_user_nice(task, new_nice);
            printk(KERN_INFO "Priority changed: new nice value = %d\n", task_nice(task));
            return 0;
        }
    }

    printk(KERN_ERR "Process with PID %d not found\n", pid);
    return -ESRCH;
}

static void __exit priority_lkm_exit(void) {
    printk(KERN_INFO "priority_lkm: Module unloaded\n");
}

module_init(priority_lkm_init);
module_exit(priority_lkm_exit);
```

gedit Makefile

```
obj-m += priority_lkm.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
```

make

sudo insmod priority_lkm.ko pid=1234 new_nice=5

dmesg | tail -n 10

Found process: bash (PID: 1234), current nice: 0

Priority changed: new nice value = 5

Process with PID 1234 not found

sudo rmmod priority_lkm

make clean

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?

sudo apt update

sudo apt install build-essential linux-headers-$(uname -r)

mkdir ~/kernel_threads_lkm

cd ~/kernel_threads_lkm

gedit kernel_threads_lkm.c

#include <linux/init.h>

#include <linux/module.h>

```
#include <linux/kernel.h>
#include <linux/sched/signal.h>

MODULE_LICENSE("GPL");
MODULE_AUTHOR("YourName");
MODULE_DESCRIPTION("LKM to list 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\tName: %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 thread lister unloaded.\n");
}
```

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

```
gedit Makefile
```

```
obj-m += kernel_threads_lkm.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
```

```
make
```

```
sudo insmod kernel_threads_lkm.ko
```

```
dmesg | tail -n 30
```

1. Configure the Linux kernel according to specific hardware and software requirements

```
sudo apt update
sudo apt install git build-essential libncurses-dev bison flex libssl-dev libelf-dev

mkdir ~/kernel_config
cd ~/kernel_config
git clone https://github.com/torvalds/linux.git
cd linux
```

```
git checkout v6.1

make defconfig

make menuconfig

make -j$(nproc)

sudo make modules_install

sudo make install

sudo update-initramfs -c -k $(make kernelrelease)
sudo update-grub
sudo reboot

make clean
make mrproper
```