UE22CS242B

OPERATING SYSTEMS

NAME : RAASHI BAFNA

SRN : PES2UG22CS422

SEM : 4   SEC : G

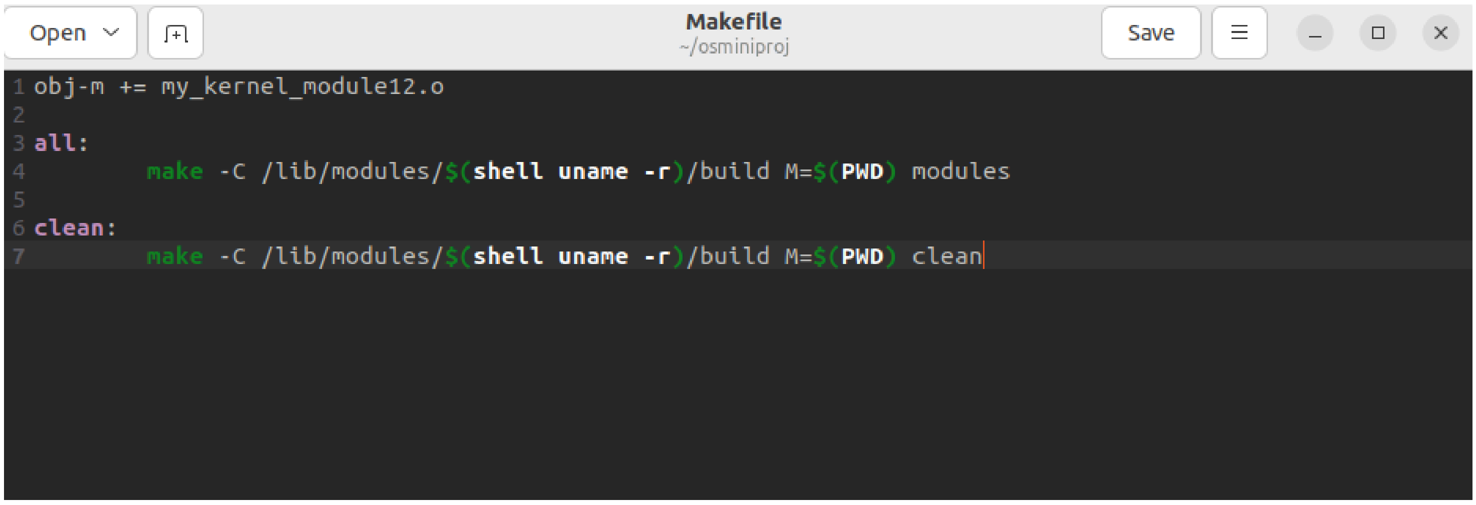
OPERATING SYSTEM MINI PROJECT CODE

PROJECT:

Create Linux kernel modules. Execute a program that will create multiple processes/threads (children and siblings). While this task is executing, output the task name (known as executable name), state and process id of each thread created by the process in a tree structure.

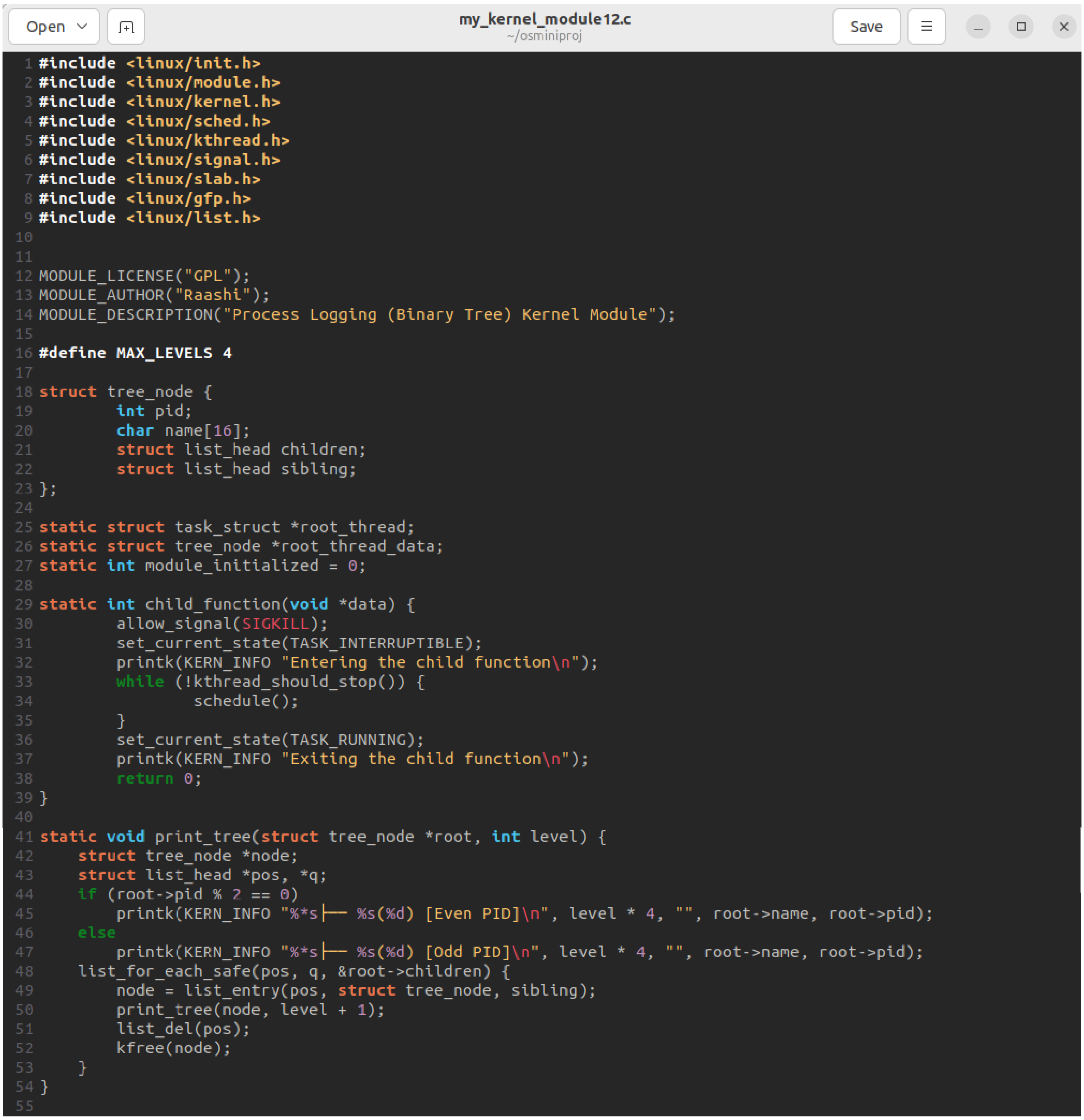
PROJECT CODES :

1) Makefile

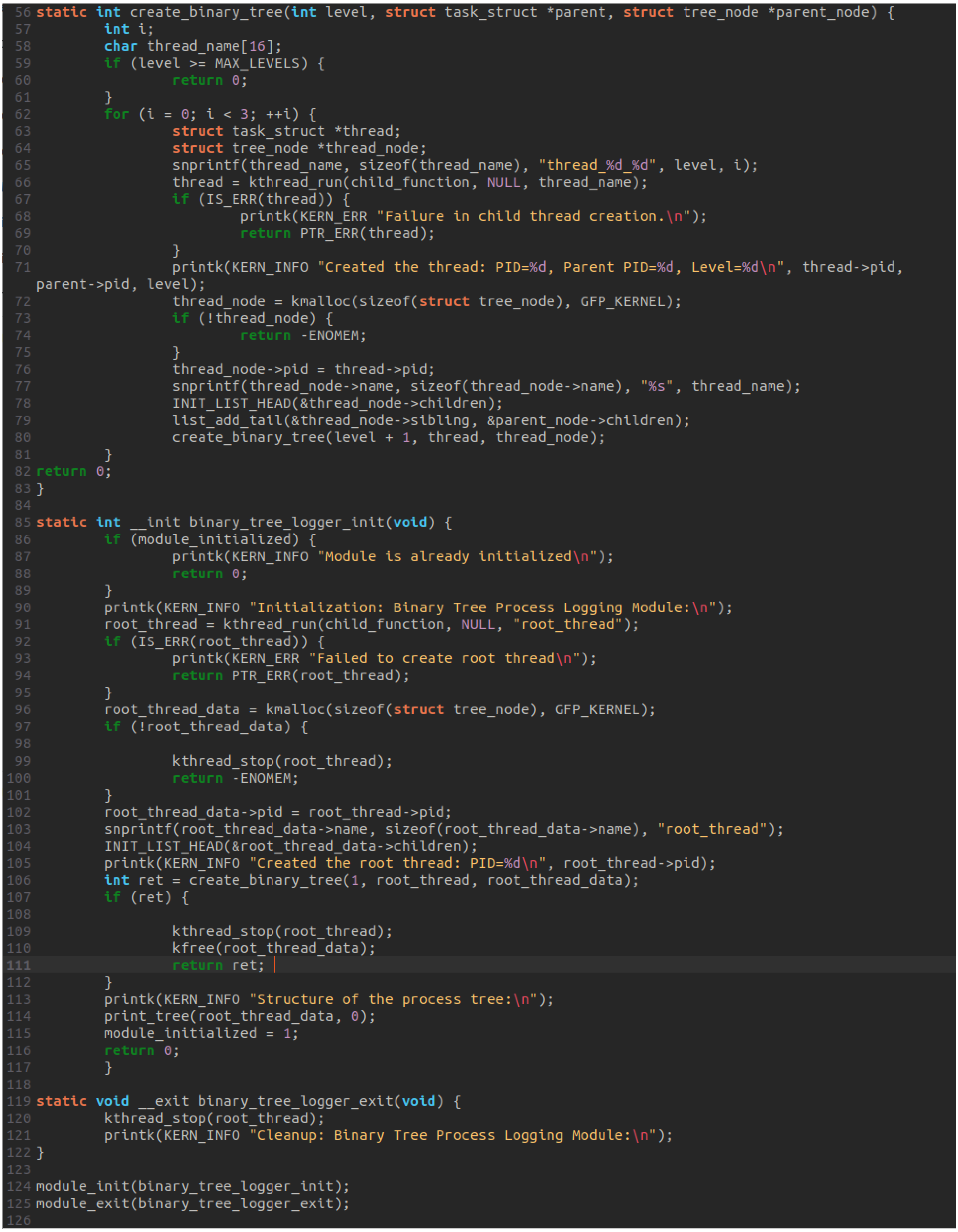


```
obj-m += my_kernel_module12.o

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

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

2) my_kernel_module12.c



```c
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/sched.h>
#include <linux/kthread.h>
#include <linux/signal.h>
#include <linux/slab.h>
#include <linux/gfp.h>
#include <linux/list.h>


MODULE_LICENSE("GPL");
MODULE_AUTHOR("Raashi");
MODULE_DESCRIPTION("Process Logging (Binary Tree) Kernel Module");

#define MAX_LEVELS 4

struct tree_node {
        int pid;
        char name[16];
        struct list_head children;
        struct list_head sibling;
};

static struct task_struct *root_thread;
static struct tree_node *root_thread_data;
static int module_initialized = 0;

static int child_function(void *data) {
        allow_signal(SIGKILL);
        set_current_state(TASK_INTERRUPTIBLE);
        printk(KERN_INFO "Entering the child function\n");
        while (!kthread_should_stop()) {
                schedule();
        }
        set_current_state(TASK_RUNNING);
        printk(KERN_INFO "Exiting the child function\n");
        return 0;
}

static void print_tree(struct tree_node *root, int level) {
    struct tree_node *node;
    struct list_head *pos, *q;
    if (root->pid % 2 == 0)
        printk(KERN_INFO "%*s├── %s(%d) [Even PID]\n", level * 4, "", root->name, root->pid);
    else
        printk(KERN_INFO "%*s├── %s(%d) [Odd PID]\n", level * 4, "", root->name, root->pid);
    list_for_each_safe(pos, q, &root->children) {
        node = list_entry(pos, struct tree_node, sibling);
        print_tree(node, level + 1);
        list_del(pos);
        kfree(node);
    }
}

```



```c
static int create_binary_tree(int level, struct task_struct *parent, struct tree_node *parent_node) {
        int i;
        char thread_name[16];
        if (level >= MAX_LEVELS) {
                return 0;
        }
        for (i = 0; i < 3; ++i) {
                struct task_struct *thread;
                struct tree_node *thread_node;
                snprintf(thread_name, sizeof(thread_name), "thread_%d_%d", level, i);
                thread = kthread_run(child_function, NULL, thread_name);
                if (IS_ERR(thread)) {
                        printk(KERN_ERR "Failure in child thread creation.\n");
                        return PTR_ERR(thread);
                }
                printk(KERN_INFO "Created the thread: PID=%d, Parent PID=%d, Level=%d\n", thread->pid, parent->pid, level);
                thread_node = kmalloc(sizeof(struct tree_node), GFP_KERNEL);
                if (!thread_node) {
                        return -ENOMEM;
                }
                thread_node->pid = thread->pid;
                snprintf(thread_node->name, sizeof(thread_node->name), "%s", thread_name);
                INIT_LIST_HEAD(&thread_node->children);
                list_add_tail(&thread_node->sibling, &parent_node->children);
                create_binary_tree(level + 1, thread, thread_node);
        }
return 0;
}

static int __init binary_tree_logger_init(void) {
        if (module_initialized) {
                printk(KERN_INFO "Module is already initialized\n");
                return 0;
        }
        printk(KERN_INFO "Initialization: Binary Tree Process Logging Module:\n");
        root_thread = kthread_run(child_function, NULL, "root_thread");
        if (IS_ERR(root_thread)) {
                printk(KERN_ERR "Failed to create root thread\n");
                return PTR_ERR(root_thread);
        }
        root_thread_data = kmalloc(sizeof(struct tree_node), GFP_KERNEL);
        if (!root_thread_data) {

                kthread_stop(root_thread);
                return -ENOMEM;
        }
        root_thread_data->pid = root_thread->pid;
        snprintf(root_thread_data->name, sizeof(root_thread_data->name), "root_thread");
        INIT_LIST_HEAD(&root_thread_data->children);
        printk(KERN_INFO "Created the root thread: PID=%d\n", root_thread->pid);
        int ret = create_binary_tree(1, root_thread, root_thread_data);
        if (ret) {

                kthread_stop(root_thread);
                kfree(root_thread_data);
                return ret;
        }
        printk(KERN_INFO "Structure of the process tree:\n");
        print_tree(root_thread_data, 0);
        module_initialized = 1;
        return 0;
        }

static void __exit binary_tree_logger_exit(void) {
        kthread_stop(root_thread);
        printk(KERN_INFO "Cleanup: Binary Tree Process Logging Module:\n");
}

module_init(binary_tree_logger_init);
module_exit(binary_tree_logger_exit);
```