Operating Systems

UE22CS242B

4th Semester, Academic Year 2024

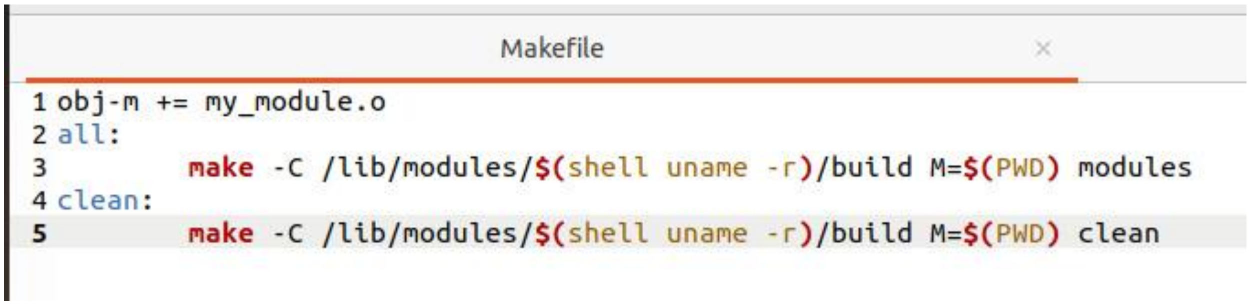
Name: ANKITH GOWDA B S	SRN : PES2UG22CS077	Section : B

Date : 31-03-2024

Question :
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.
Example: my_kernel_module <process id of the program executing>

Makefile

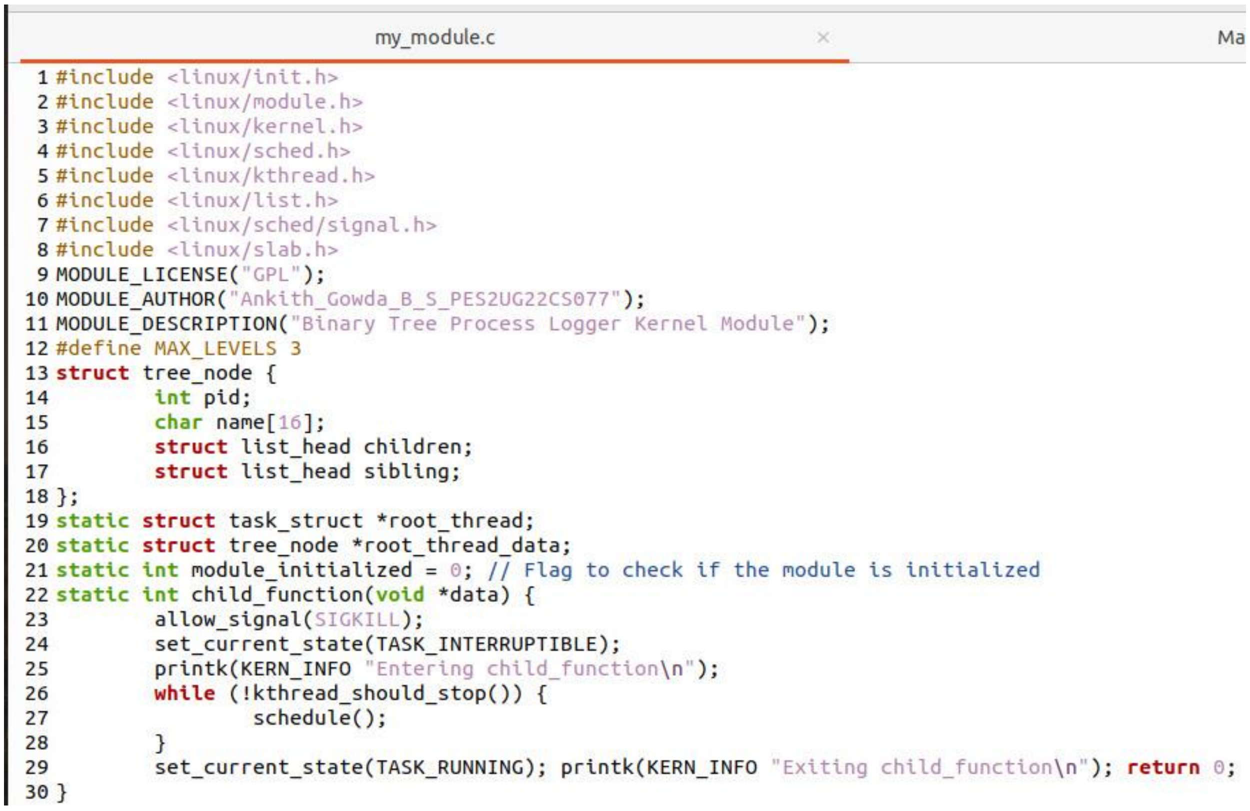
Code :



```
obj-m += my_module.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
```

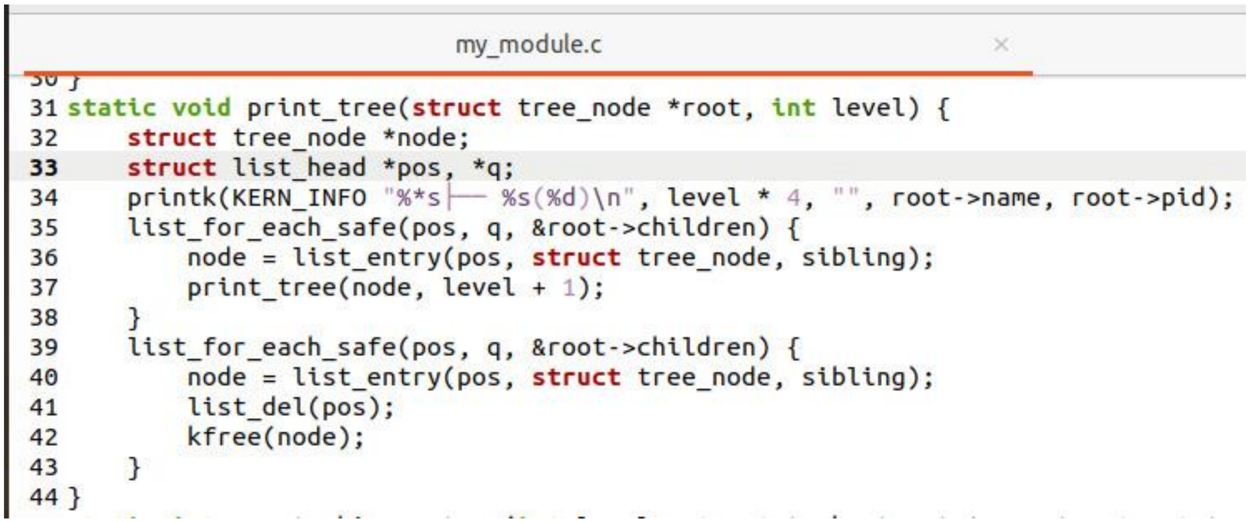
my_module.c

Code :



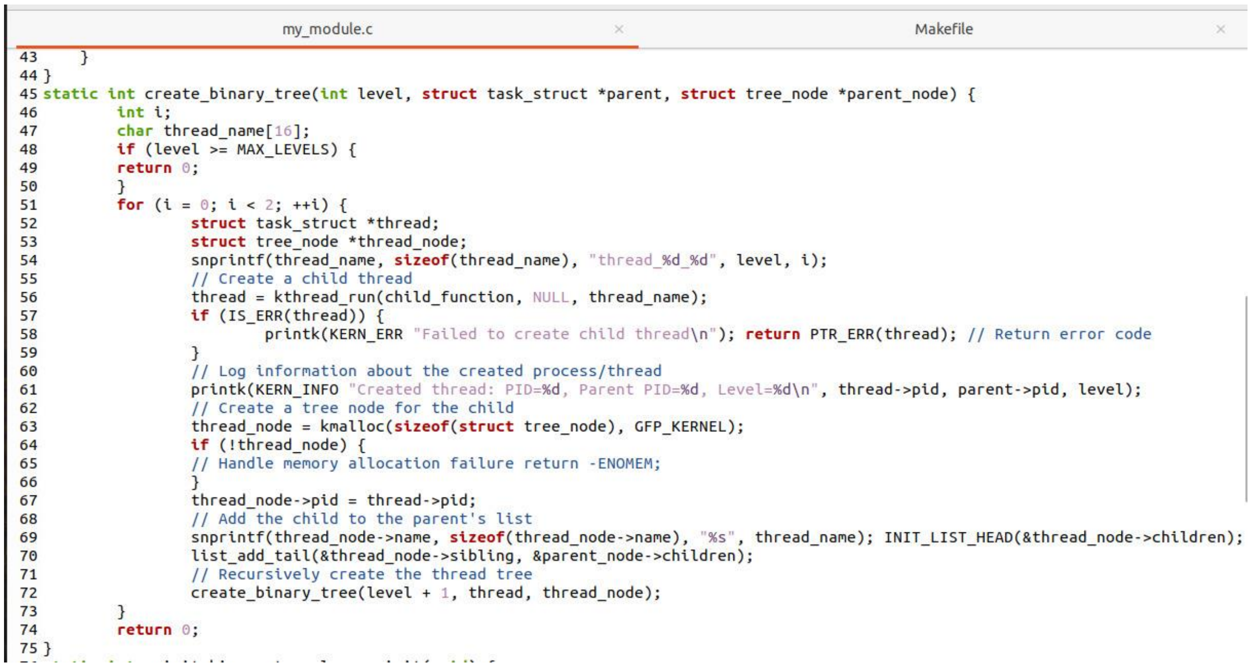
my_module.c

```c
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/sched.h>
#include <linux/kthread.h>
#include <linux/list.h>
#include <linux/sched/signal.h>
#include <linux/slab.h>
MODULE_LICENSE("GPL");
MODULE_AUTHOR("Ankith_Gowda_B_S_PES2UG22CS077");
MODULE_DESCRIPTION("Binary Tree Process Logger Kernel Module");
#define MAX_LEVELS 3
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; // Flag to check if the module is initialized
static int child_function(void *data) {
        allow_signal(SIGKILL);
        set_current_state(TASK_INTERRUPTIBLE);
        printk(KERN_INFO "Entering child_function\n");
        while (!kthread_should_stop()) {
                schedule();
        }
        set_current_state(TASK_RUNNING); printk(KERN_INFO "Exiting child_function\n"); return 0;
}
```

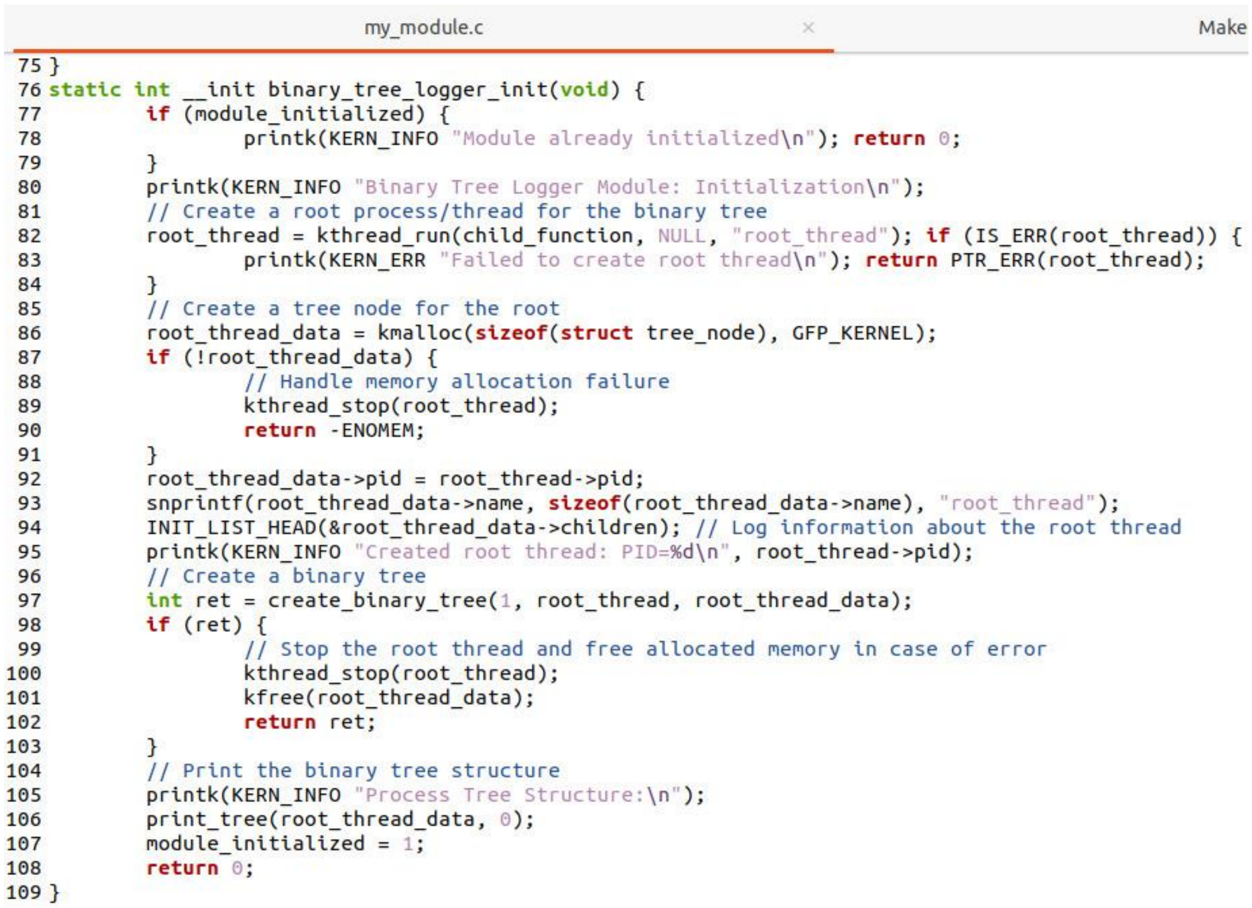


my_module.c

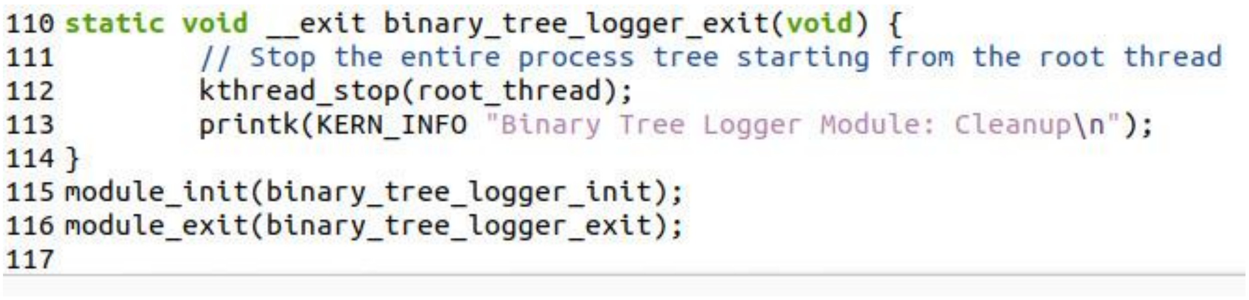
```c
static void print_tree(struct tree_node *root, int level) {
    struct tree_node *node;
    struct list_head *pos, *q;
    printk(KERN_INFO "%*s├── %s(%d)\n", level * 4, "", root->name, root->pid);
    list_for_each_safe(pos, q, &root->children) {
        node = list_entry(pos, struct tree_node, sibling);
        print_tree(node, level + 1);
    }
    list_for_each_safe(pos, q, &root->children) {
        node = list_entry(pos, struct tree_node, sibling);
        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 < 2; ++i) {
                struct task_struct *thread;
                struct tree_node *thread_node;
                snprintf(thread_name, sizeof(thread_name), "thread_%d_%d", level, i);
                // Create a child thread
                thread = kthread_run(child_function, NULL, thread_name);
                if (IS_ERR(thread)) {
                        printk(KERN_ERR "Failed to create child thread\n"); return PTR_ERR(thread); // Return error code
                }
                // Log information about the created process/thread
                printk(KERN_INFO "Created thread: PID=%d, Parent PID=%d, Level=%d\n", thread->pid, parent->pid, level);
                // Create a tree node for the child
                thread_node = kmalloc(sizeof(struct tree_node), GFP_KERNEL);
                if (!thread_node) {
                // Handle memory allocation failure return -ENOMEM;
                }
                thread_node->pid = thread->pid;
                // Add the child to the parent's list
                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);
                // Recursively create the thread tree
                create_binary_tree(level + 1, thread, thread_node);
        }
        return 0;
}
```

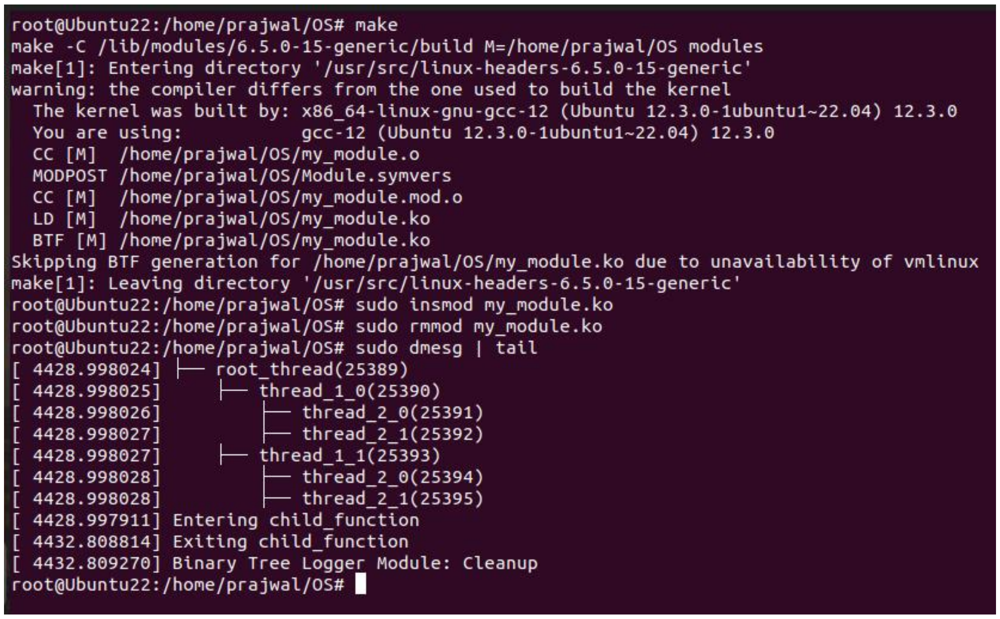


```c
}
static int __init binary_tree_logger_init(void) {
        if (module_initialized) {
                printk(KERN_INFO "Module already initialized\n"); return 0;
        }
        printk(KERN_INFO "Binary Tree Logger Module: Initialization\n");
        // Create a root process/thread for the binary tree
        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);
        }
        // Create a tree node for the root
        root_thread_data = kmalloc(sizeof(struct tree_node), GFP_KERNEL);
        if (!root_thread_data) {
                // Handle memory allocation failure
                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); // Log information about the root thread
        printk(KERN_INFO "Created root thread: PID=%d\n", root_thread->pid);
        // Create a binary tree
        int ret = create_binary_tree(1, root_thread, root_thread_data);
        if (ret) {
                // Stop the root thread and free allocated memory in case of error
                kthread_stop(root_thread);
                kfree(root_thread_data);
                return ret;
        }
        // Print the binary tree structure
        printk(KERN_INFO "Process Tree Structure:\n");
        print_tree(root_thread_data, 0);
        module_initialized = 1;
        return 0;
}
```

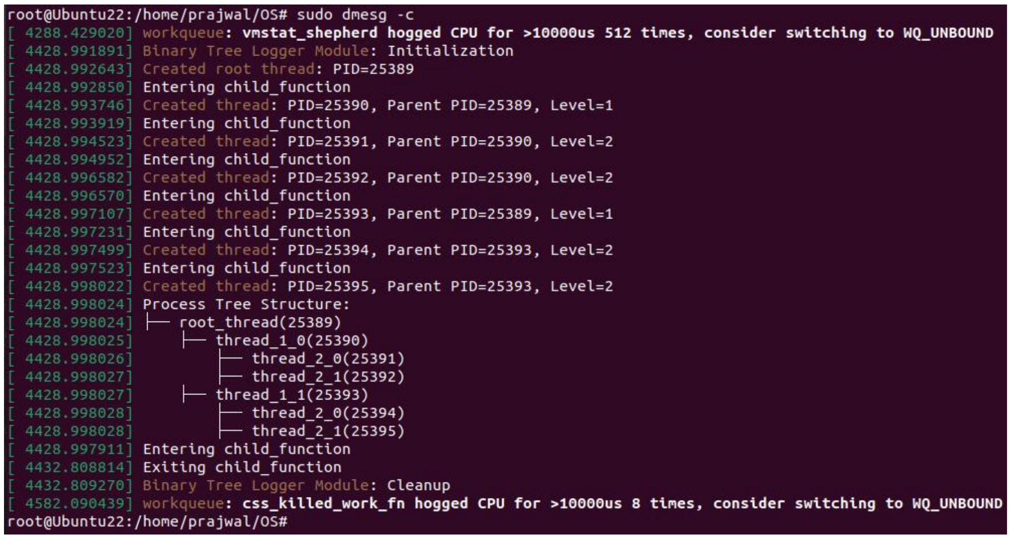


```c
static void __exit binary_tree_logger_exit(void) {
        // Stop the entire process tree starting from the root thread
        kthread_stop(root_thread);
        printk(KERN_INFO "Binary Tree Logger Module: Cleanup\n");
}
module_init(binary_tree_logger_init);
module_exit(binary_tree_logger_exit);
```

OUTPUT :



```
root@Ubuntu22:/home/prajwal/OS# make
make -C /lib/modules/6.5.0-15-generic/build M=/home/prajwal/OS modules
make[1]: Entering directory '/usr/src/linux-headers-6.5.0-15-generic'
warning: the compiler differs from the one used to build the kernel
  The kernel was built by: x86_64-linux-gnu-gcc-12 (Ubuntu 12.3.0-1ubuntu1~22.04) 12.3.0
  You are using:           gcc-12 (Ubuntu 12.3.0-1ubuntu1~22.04) 12.3.0
  CC [M]  /home/prajwal/OS/my_module.o
  MODPOST /home/prajwal/OS/Module.symvers
  CC [M]  /home/prajwal/OS/my_module.mod.o
  LD [M]  /home/prajwal/OS/my_module.ko
  BTF [M] /home/prajwal/OS/my_module.ko
Skipping BTF generation for /home/prajwal/OS/my_module.ko due to unavailability of vmlinux
make[1]: Leaving directory '/usr/src/linux-headers-6.5.0-15-generic'
root@Ubuntu22:/home/prajwal/OS# sudo insmod my_module.ko
root@Ubuntu22:/home/prajwal/OS# sudo rmmod my_module.ko
root@Ubuntu22:/home/prajwal/OS# sudo dmesg | tail
[ 4428.998024] ├── root_thread(25389)
[ 4428.998025]     ├── thread_1_0(25390)
[ 4428.998026]         ├── thread_2_0(25391)
[ 4428.998027]         ├── thread_2_1(25392)
[ 4428.998027]     ├── thread_1_1(25393)
[ 4428.998028]         ├── thread_2_0(25394)
[ 4428.998028]         ├── thread_2_1(25395)
[ 4428.997911] Entering child_function
[ 4432.808814] Exiting child_function
[ 4432.809270] Binary Tree Logger Module: Cleanup
root@Ubuntu22:/home/prajwal/OS# 
```



```
root@Ubuntu22:/home/prajwal/OS# sudo dmesg -c
[ 4288.429020] workqueue: vmstat_shepherd hogged CPU for >10000us 512 times, consider switching to WQ_UNBOUND
[ 4428.991891] Binary Tree Logger Module: Initialization
[ 4428.992643] Created root thread: PID=25389
[ 4428.992850] Entering child_function
[ 4428.993746] Created thread: PID=25390, Parent PID=25389, Level=1
[ 4428.993919] Entering child_function
[ 4428.994523] Created thread: PID=25391, Parent PID=25390, Level=2
[ 4428.994952] Entering child_function
[ 4428.996582] Created thread: PID=25392, Parent PID=25390, Level=2
[ 4428.996570] Entering child_function
[ 4428.997107] Created thread: PID=25393, Parent PID=25389, Level=1
[ 4428.997231] Entering child_function
[ 4428.997499] Created thread: PID=25394, Parent PID=25393, Level=2
[ 4428.997523] Entering child_function
[ 4428.998022] Created thread: PID=25395, Parent PID=25393, Level=2
[ 4428.998024] Process Tree Structure:
[ 4428.998024] ├── root_thread(25389)
[ 4428.998025]     ├── thread_1_0(25390)
[ 4428.998026]         ├── thread_2_0(25391)
[ 4428.998027]         ├── thread_2_1(25392)
[ 4428.998027]     ├── thread_1_1(25393)
[ 4428.998028]         ├── thread_2_0(25394)
[ 4428.998028]         ├── thread_2_1(25395)
[ 4428.997911] Entering child_function
[ 4432.808814] Exiting child_function
[ 4432.809270] Binary Tree Logger Module: Cleanup
[ 4582.090439] workqueue: css_killed_work_fn hogged CPU for >10000us 8 times, consider switching to WQ_UNBOUND
root@Ubuntu22:/home/prajwal/OS#
```