OPERATING SYSTEMS MINIPROJECT

NAME: Keerthan P.V

SRN: PES2UG23CS272

jackfruit.c

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/slab.h>
#include <linux/kthread.h>
#include <linux/delay.h>

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Keerthan");
MODULE_DESCRIPTION("3-Level Binary Process Tree with Ordered Output and Memory Map");

struct bday
{
  int day;
int month;
  int year;
};

static struct bday *allocate_birthday(int d, int m, int y)
{
  struct bday *b = kmalloc(sizeof(struct bday),
GFP_KERNEL);   if (!b) return NULL;   b->day = d;
  b->month = m;
  b->year = y;
return b;
```

```
}

static void print_bday_and_memory_map(const char *label, struct bday *b, const char *prefix, const
char *branch)
{
  printk(KERN_INFO "%s%s%s\n", prefix, branch, label);   printk(KERN_INFO "%s| Birthday :
%02d-%02d-%04d\n", prefix, b->day, b->month, b->year);   printk(KERN_INFO "%s| Memory
Map:\n", prefix);   printk(KERN_INFO "%s| Address : %p\n", prefix, b);   printk(KERN_INFO
"%s| Heap : struct bday\n", prefix);   printk(KERN_INFO "%s| Fields : day, month, year\n",
prefix);
}

struct thread_args
{
  const char *prefix;
const char *label;
int day, month, year;
};

static int ggc_fn(void *data)
{
  struct thread_args *args = (struct thread_args *)data;   struct bday *b
= allocate_birthday(args->day, args->month, args->year);
  if (b)
      print_bday_and_memory_map(args->label, b, args->prefix, "└── ");
    kfree(b);
kfree(args); return 0;
}

static int grandchild_fn(void *data)
{
```

```
 struct thread_args *args = (struct thread_args *)data;   struct bday *b
= allocate_birthday(args->day, args->month, args->year);
 if (b)
   print_bday_and_memory_map(args->label, b, args->prefix, " ├── ");

 if (strcmp(args->label, "Grandchild 1:") == 0) {     struct thread_args *a1 =
kmalloc(sizeof(struct thread_args), GFP_KERNEL);
   *a1 = (struct thread_args){ "| | ", "GGC 1:", 27, 9, 2005 };
kthread_run(ggc_fn, a1, "ggc1_thread");

   ssleep(1);

   struct thread_args *a2 = kmalloc(sizeof(struct thread_args),
GFP_KERNEL);
   *a2 = (struct thread_args){ "| | ", "GGC 2:", 12, 2, 2002 };
kthread_run(ggc_fn, a2, "ggc2_thread");
 } else {
   struct thread_args *a3 = kmalloc(sizeof(struct thread_args),
GFP_KERNEL);
   *a3 = (struct thread_args){ "| | ", "GGC 3:", 27, 2, 2004 };
kthread_run(ggc_fn, a3, "ggc3_thread");
   ssleep(1);
   struct thread_args *a4 = kmalloc(sizeof(struct thread_args),
GFP_KERNEL);
   *a4 = (struct thread_args){ "| | ", "GGC 4:", 5, 4, 2004 };
kthread_run(ggc_fn, a4, "ggc4_thread");
 }
 ssleep(1);
kfree(b);
kfree(args);  return
0;
}
```

```
static int child_fn(void *data) {   struct bday *b
= allocate_birthday(2, 3, 1985);
 if (b)
   print_bday_and_memory_map("Child:", b, "", "");

 struct thread_args *g1 = kmalloc(sizeof(struct thread_args), GFP_KERNEL);
 *g1 = (struct thread_args){ "| ", "Grandchild 1:", 10, 9, 1995
};   kthread_run(grandchild_fn, g1, "gc1_thread");   ssleep(1);
 struct thread_args *g2 = kmalloc(sizeof(struct thread_args), GFP_KERNEL);
 *g2 = (struct thread_args){ "| ", "Grandchild 2:", 2, 1, 1998 };
kthread_run(grandchild_fn, g2, "gc2_thread");
 ssleep(2);
kfree(b);
return 0;
}

static int __init tree_module_init(void) {   printk(KERN_INFO "\n===
Binary Process Tree Module Loaded ===\n");   kthread_run(child_fn,
NULL, "child_thread");   return 0;
}

static void __exit tree_module_exit(void) {  printk(KERN_INFO "===
Binary Process Tree Module Removed ===\n");
}
module_init(tree_module_init);
module_exit(tree_module_exit);
```

Makefile

```
obj-m += jackfruit.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

Commands to execute:

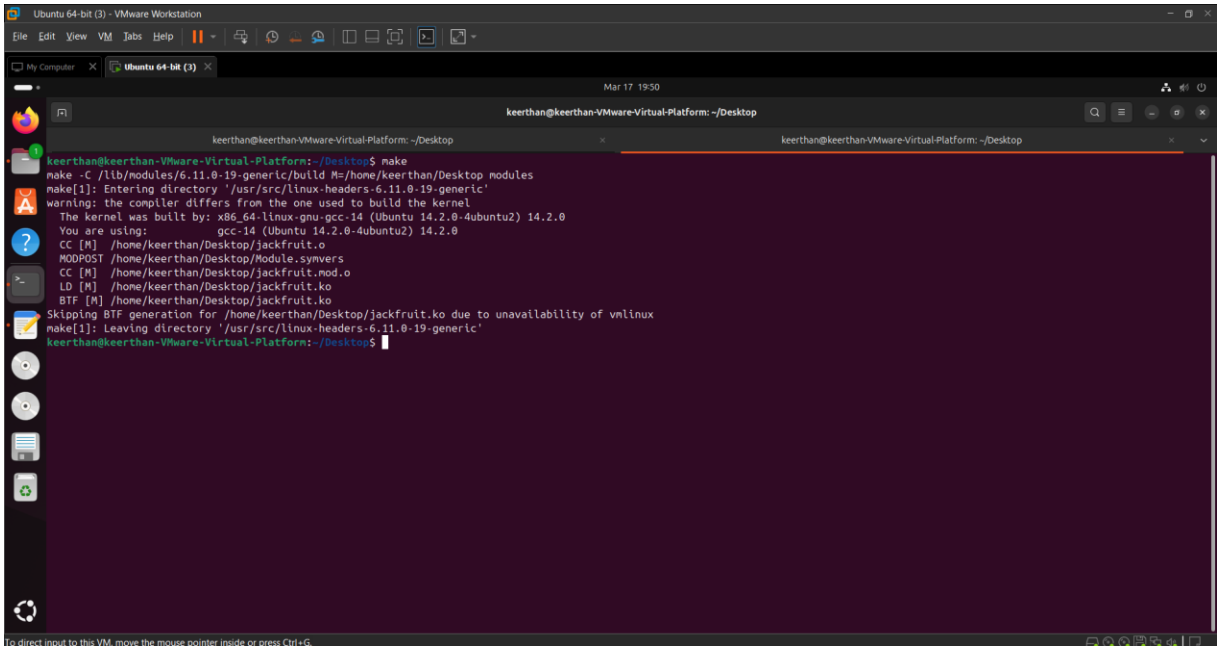
make

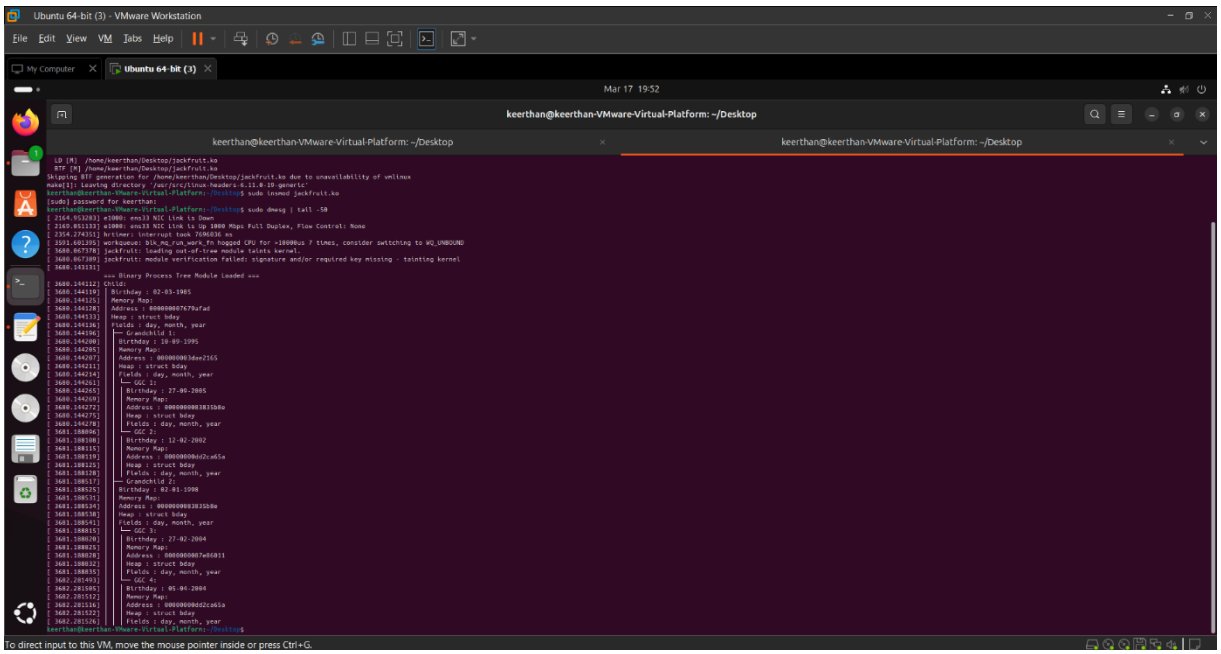
sudo insmod jackfruit.ko

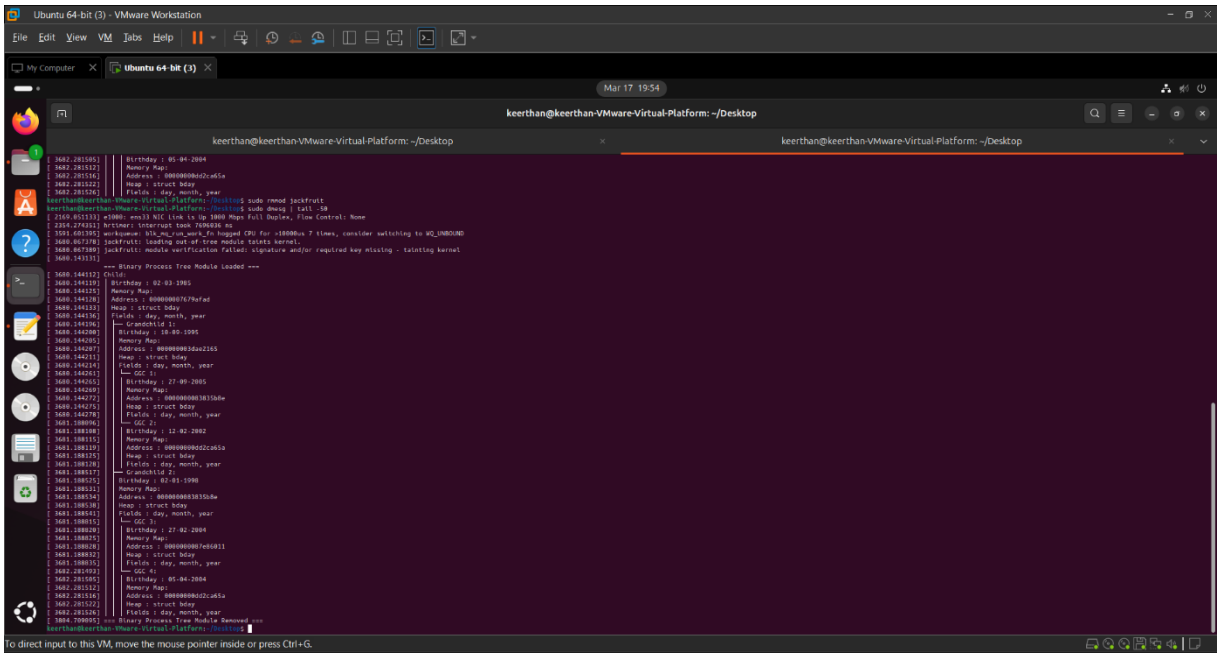
sudo dmesg | tail -50

sudo rmmod jackfruit

Output Screenshot







EXPLANATION:

Linux Kernal Module

This Linux kernel module constructs a 3-level binary process tree while demonstrating key kernel programming concepts such as thread creation, synchronization, and memory allocation. It ensures an ordered output and effectively maps memory usage.

- **License: GPL • Author: Keerthan Key Components & Data Structures 1. Birthday Structure (struct bday)**
- Stores a date of birth with three fields: day, month, and year.
- Used throughout the module to represent birthday data.

Memory Management

- **Allocation:** Uses kmalloc() to allocate kernel memory.
- **Initialization:** The allocate_birthday() function assigns values to the structure.
- **Deallocation:** Frees memory using kfree() to prevent memory leaks.

2. Thread Arguments Structure (struct thread_args)

- Stores thread-specific data, including:
    - Prefix & label strings (used for formatted output).
    - Birthday structure (day, month, year).

Process Tree Hierarchy

This module generates a binary tree-like process structure spanning three levels:

1. Root Level: The kernel module initializes by creating a child thread.
2. Level 1: The child process spawns two grandchild threads.
3. Level 2: Each grandchild process spawns two great-grandchild (GGC) threads.
4. Level 3: The GGC processes form the leaf nodes of the tree.

Process & Thread Creation

- Uses kthread_run() to spawn kernel threads dynamically.
- Parent-child relationships are established using thread functions.

Thread Functions & Execution

1. Child Thread (child_fn)

- Creates a birthday structure (01-01-1990).
- Spawns two grandchild threads.
- Uses ssleep() to maintain an ordered execution sequence.

2. Grandchild Threads (grandchild_fn)

- Generates its own birthday structure.
- Determines which GGC threads to create.
- Synchronizes execution using sleep delays.

3. Great-Grandchild Threads (ggc_fn)

- Final level in the hierarchy.
- Simply creates a birthday structure and prints information.

Output & Visualization

1. Tree Representation

- Uses ASCII characters to format and display the hierarchical structure.
- Threads print their position in the process tree.

2. Memory Mapping

- Displays memory address and allocation type for each birthday structure.
- Provides a detailed memory map of the entire tree.

Synchronization Mechanisms

- Sleep (ssleep()) ensures a sequential execution order.
- Parent processes wait for child processes to complete before proceeding.
- Helps maintain a proper tree structure in output.

Module Lifecycle

1. Module Initialization (tree_module_init)

- Entry point when the module is inserted (insmod Process_tree.ko).
- Prints a startup message.
- Launches the child thread.

2. Module Cleanup (tree_module_exit)

- Triggered when the module is removed (rmmod Process_tree).
- Displays a cleanup message.
- No explicit cleanup required as threads complete their tasks.