RAMAIAH
Institute of Technology

Real-Time Scheduling:

Earliest Deadline First (EDF) Algorithm

Shreya - 1MS22CS138

Shrinidhi Pawar - 1MS22CS141

RAMAIAH
Institute of Technology

Problem Statement

Real-Time Scheduling (Earliest Deadline First)

- Implement the Earliest Deadline First (EDF) scheduling algorithm
- Manage tasks in a real-time system efficiently
- Ensure tasks meet their deadlines to avoid missed deadlines

RAMAIAH
Institute of Technology

Solution Overview

- Developed a Linux Kernel module to simulate EDF and CFS scheduling
- Implemented process queues and scheduling logic in C
- Created a user-space Python GUI for live interactive visualization
- Visualization includes Gantt chart animations and scheduling metrics
- User controls: Start, Stop, and Restart to observe scheduling step-by-step

RAMAIAH
Institute of Technology

Architecture Flow

RAMAIAH
Institute of Technology

Tools and Technologies Used

- Kernel Module Programming: C language, Linux Kernel 5.x environment
- Process Management: Kernel linked lists and scheduling simulation
- User Interface: Python with Tkinter for GUI and Matplotlib for Gantt charts
- Build and Automation: Makefile for kernel module compilation
- Visualization: Real-time animations demonstrating scheduling decisions and metrics

RAMAIAH
Institute of Technology

Kernel Module - edf_scheduler1.c

Purpose:

- Simulates Earliest Deadline First (EDF) and Completely Fair Scheduler (CFS) algorithms
- Manages task scheduling and execution inside the Linux kernel module
- Tracks CPU usage and missed deadlines for each scheduler
- Provides logs via /proc for monitoring and comparison
- Enables real-time testing and visualization of scheduling behavior

RAMAIAH
Institute of Technology

Key Code Snippet

```c
static void add_edf_process(int pid, int deadline, int exec_time, int arrival_time) {
    struct edf_process *new_proc = kmalloc(sizeof(*new_proc), GFP_KERNEL);
    struct edf_process *pos;

    if (!new_proc) return;

    new_proc->pid = pid;
    new_proc->deadline = deadline;
    new_proc->execution_time = exec_time;
    new_proc->arrival_time = arrival_time;

    list_for_each_entry(pos, &edf_ready_queue, list) {
        if (deadline < pos->deadline) {
            list_add_tail(&new_proc->list, &pos->list);
            return;
        }
    }
    list_add_tail(&new_proc->list, &edf_ready_queue);
}
```

RAMAIAH
Institute of Technology

Key Code Snippet

```c
// EDF Scheduler simulation
static void run_edf_scheduler(void) {
    struct edf_process *proc, *tmp;
    edf_current_time = 0;
    edf_missed_deadlines = 0;

    list_for_each_entry_safe(proc, tmp, &edf_ready_queue, list) {
        if (edf_current_time < proc->arrival_time)
            edf_current_time = proc->arrival_time;

        printk("Running P%d (Deadline: %d, Exec: %d) at time %d\n",
               proc->pid, proc->deadline, proc->execution_time, edf_current_time);

        edf_current_time += proc->execution_time;

        if (edf_current_time > proc->deadline) {
            edf_missed_deadlines++;
            printk("Deadline missed by P%d\n", proc->pid);
        }

        list_del(&proc->list);
        kfree(proc);
    }
```

RAMAIAH
Institute of Technology

Makefile Script

Purpose:

- Automates compilation of the kernel module edf_scheduler1.c
- Uses the current kernel build environment for compatibility
- Provides targets for building (all) and cleaning (clean) module files
- Simplifies module management with standard commands

RAMAIAH
Institute of Technology

Code Snippet

```
obj-m += edf_scheduler1.o

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

all:
    $(MAKE) -C $(KDIR) M=$(PWD) modules

clean:
    $(MAKE) -C $(KDIR) M=$(PWD) clean
```

RAMAIAH
Institute of Technology

GUI - edf_scheduler1.py

Purpose:

- Provides a graphical interface to simulate and visualize EDF and CFS scheduling algorithms
- Displays live animated Gantt charts comparing the two schedulers step-by-step
- Offers user controls: Start, Stop, Restart, and Next Step for interactive simulation
- Reads real-time scheduler logs from the kernel module via /proc/edf_cfs_log
- Shows key metrics like missed deadlines and CPU utilization dynamically
- Helps users understand and compare the scheduling behavior visually and interactively

RAMAIAH
Institute of Technology

Key Code Snippet

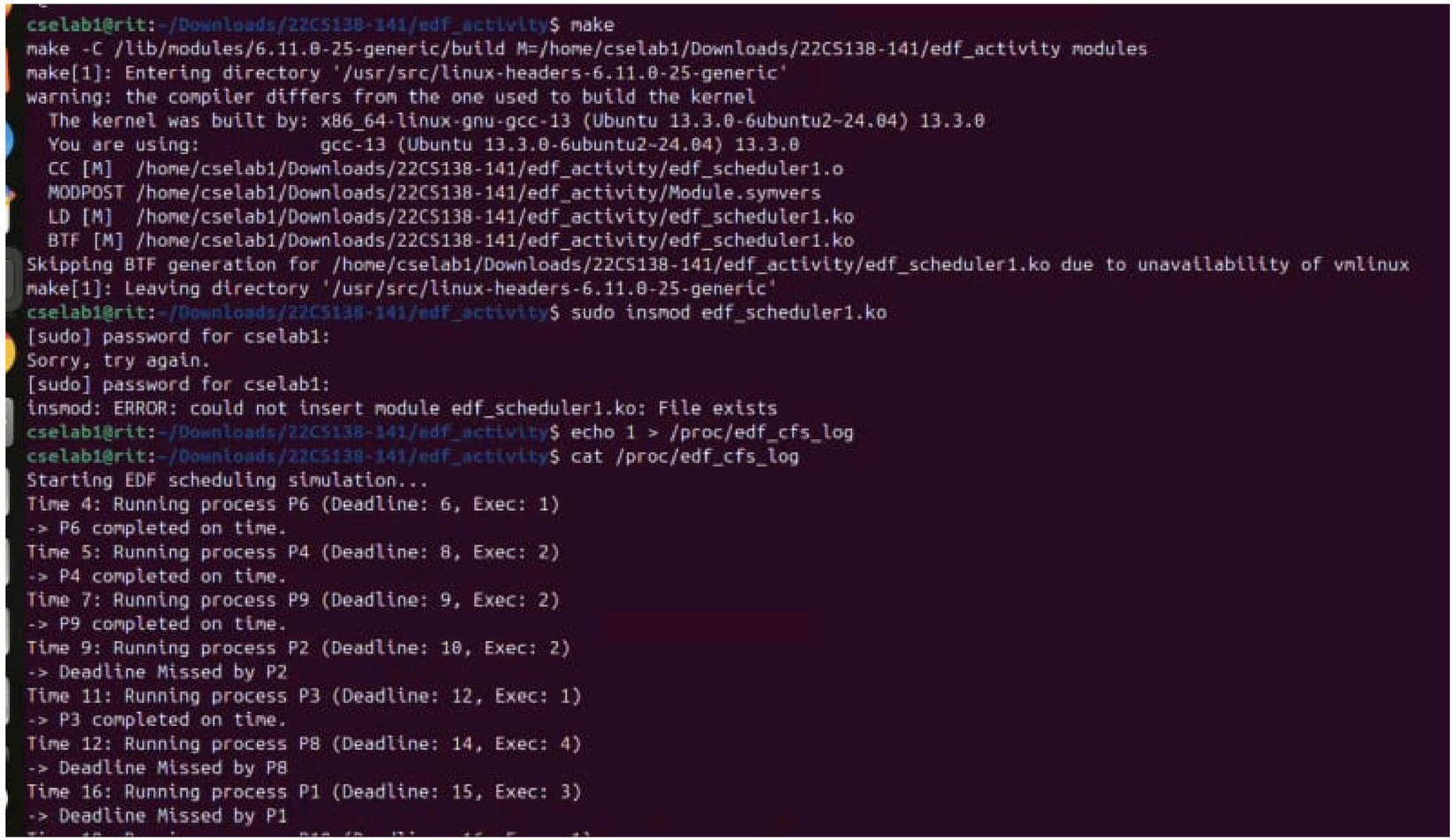
```
def next_step(self):
    if self.current_step >= len(self.all_events):
        self.timer.stop()
        return
    event = self.all_events[self.current_step]
    scheduler, pid, start, end = event
    self.plot_gantt(upto=self.current_step + 1)
    self.current_step += 1

def start_simulation(self):
    self.load_logs()
    self.current_step = 0
    self.timer.start(1000)  # 1-second interval

def stop_simulation(self):
    self.timer.stop()
```

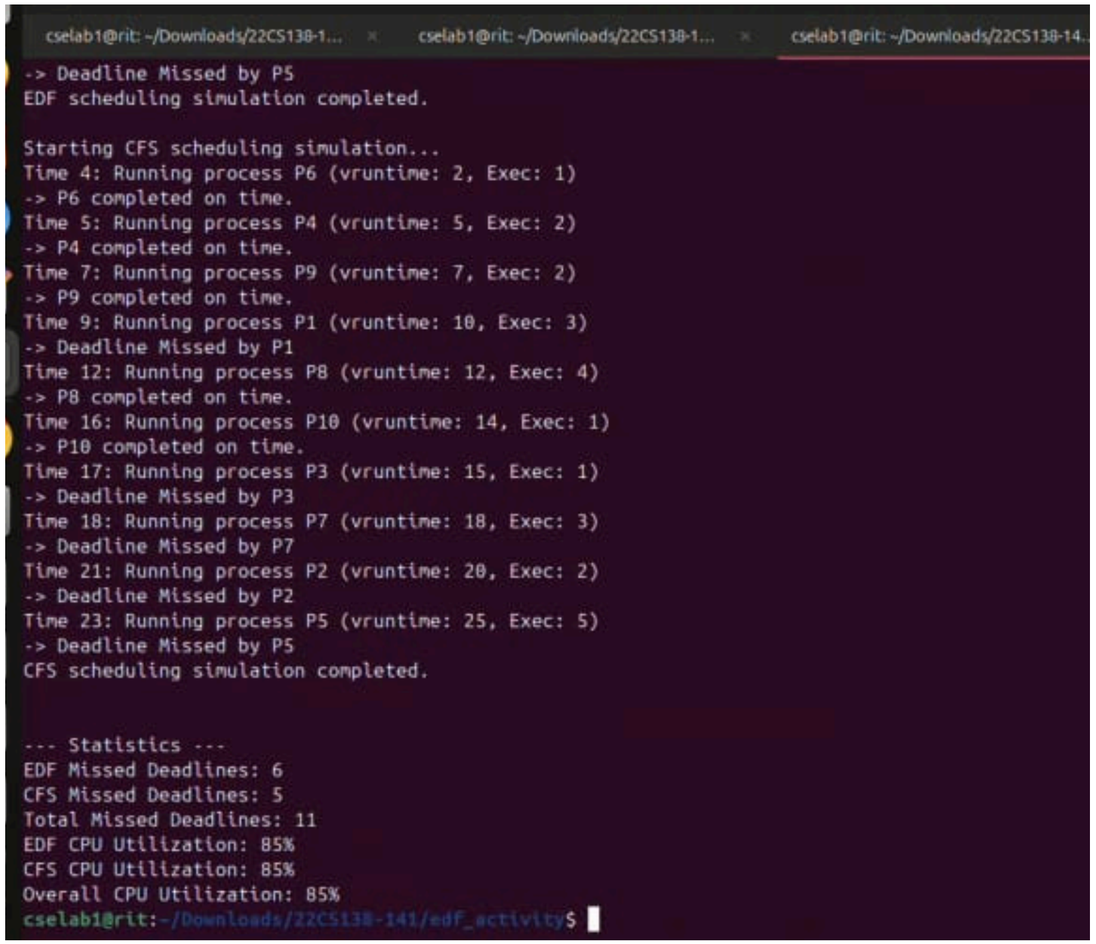
RAMAIAH
Institute of Technology

Output Screenshots



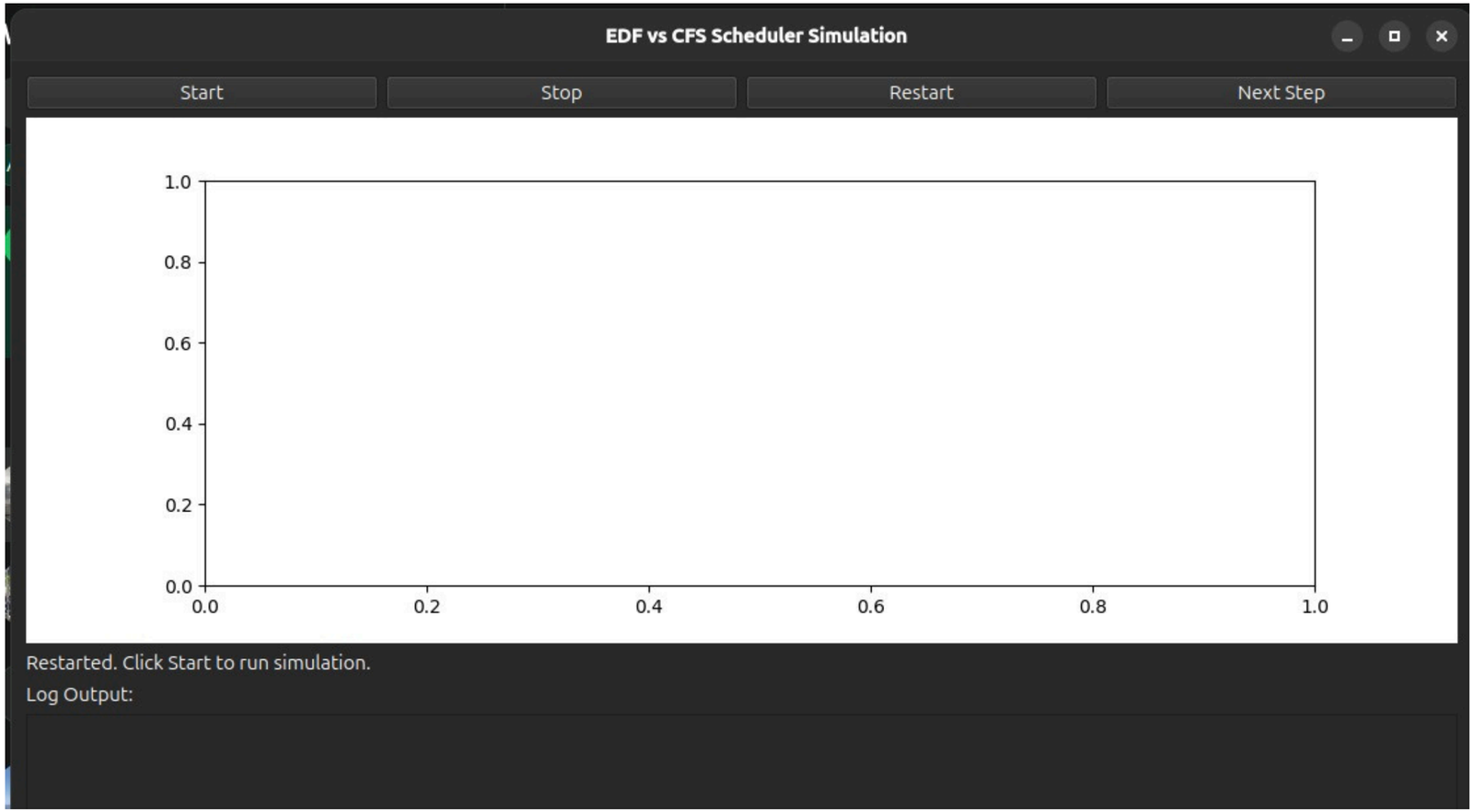
RAMAIAH
Institute of Technology

Output Screenshots



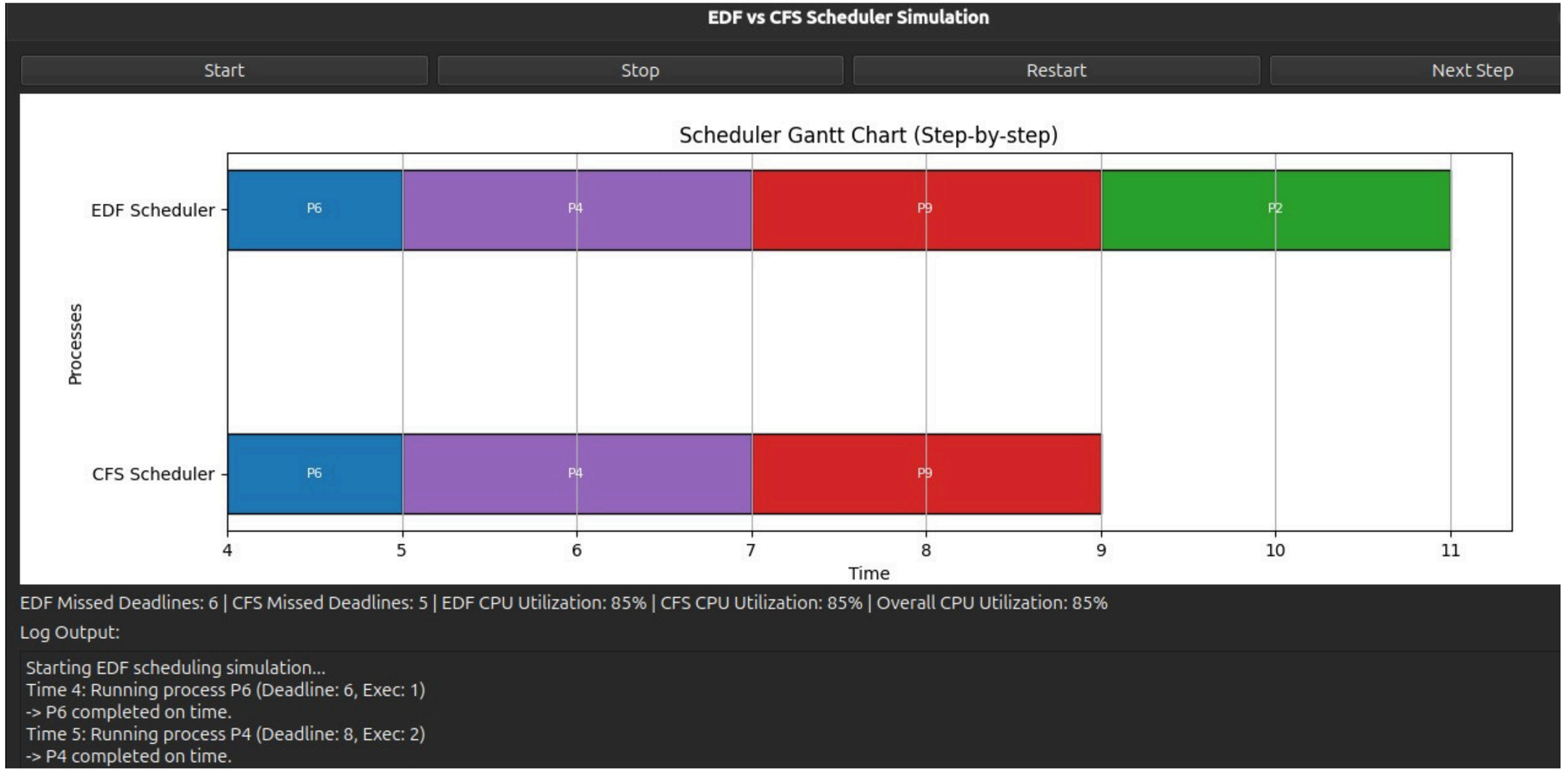
RAMAIAH
Institute of Technology

Output Screenshots



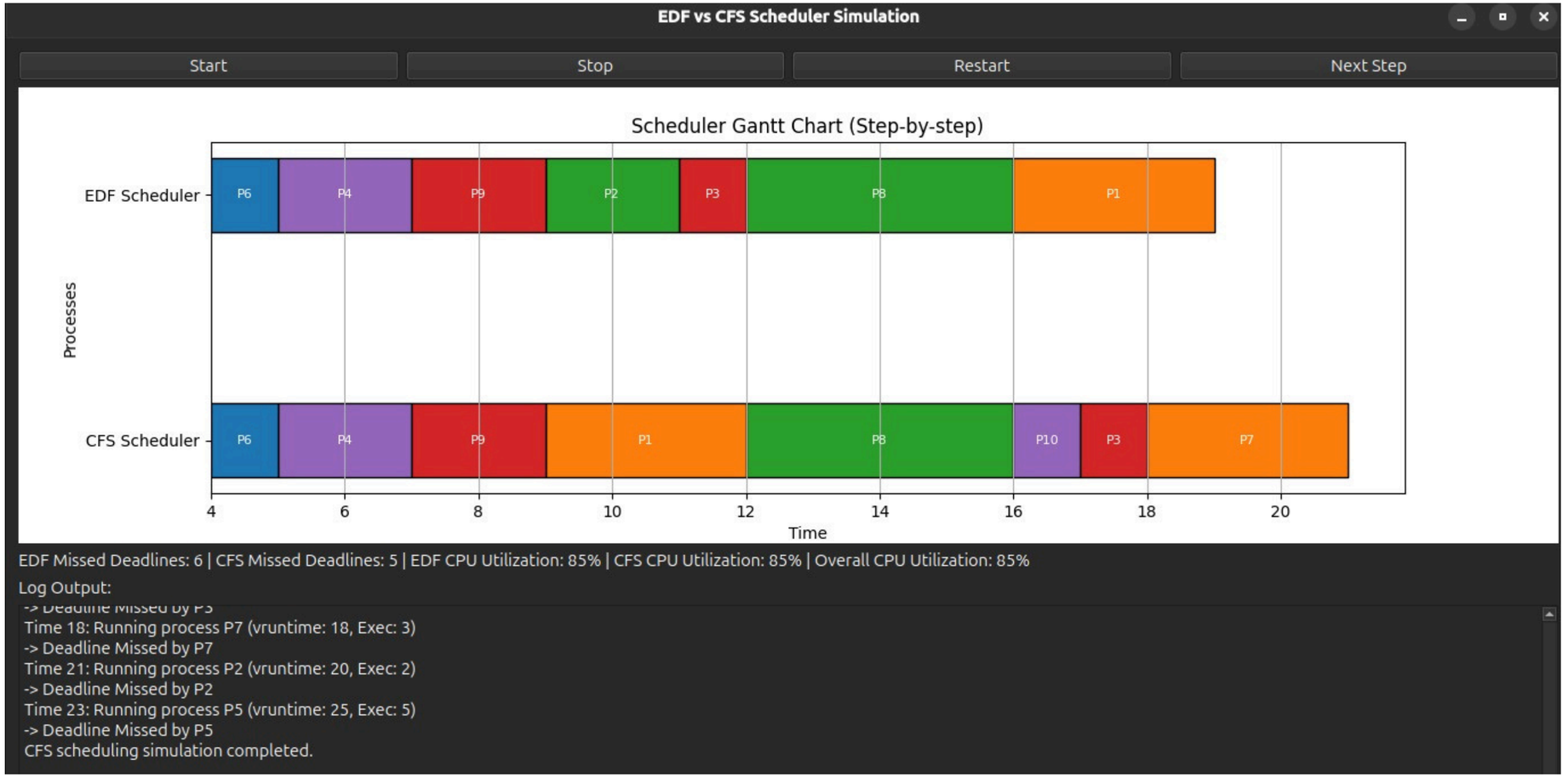
RAMAIAH
Institute of Technology

Output Screenshots



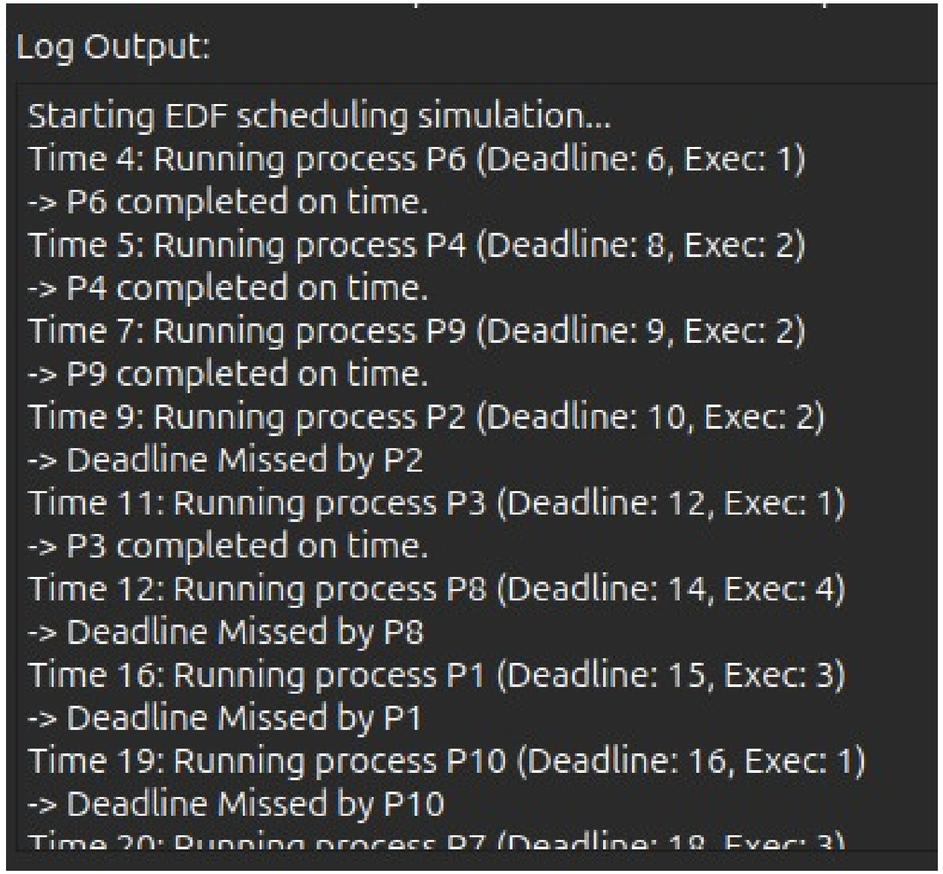
RAMAIAH
Institute of Technology

Output Screenshots



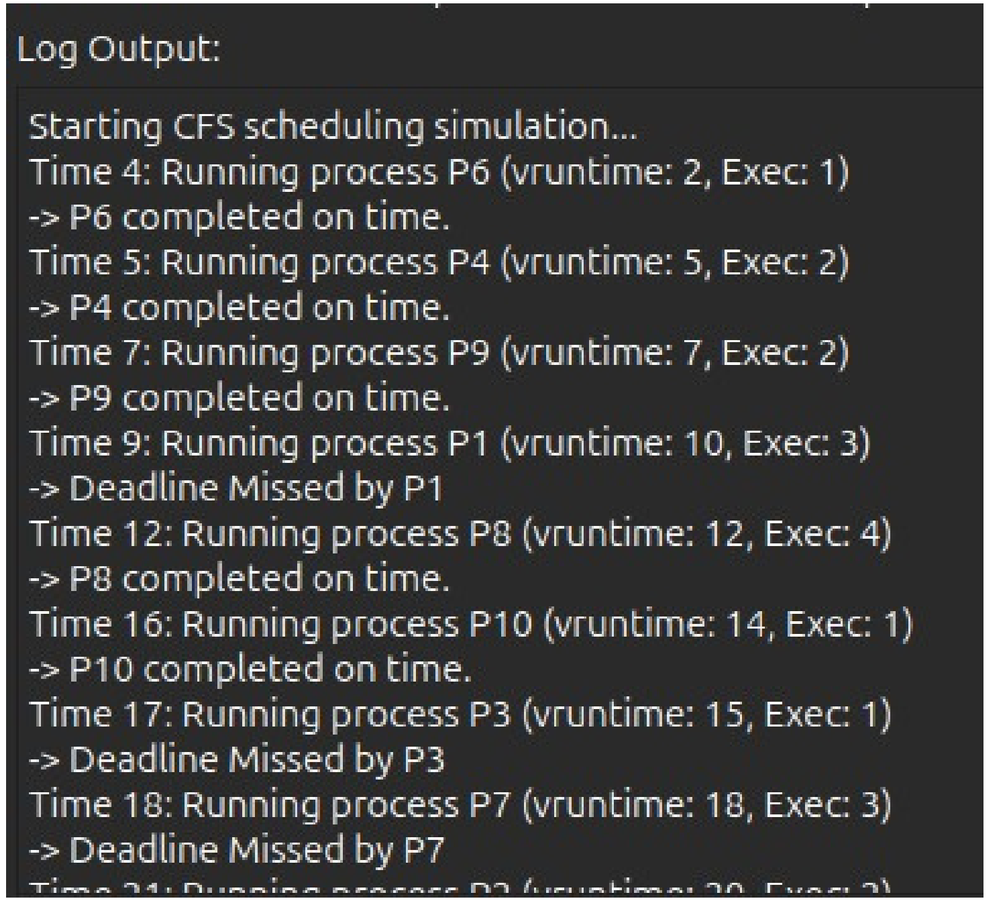
RAMAIAH
Institute of Technology

Output Screenshots



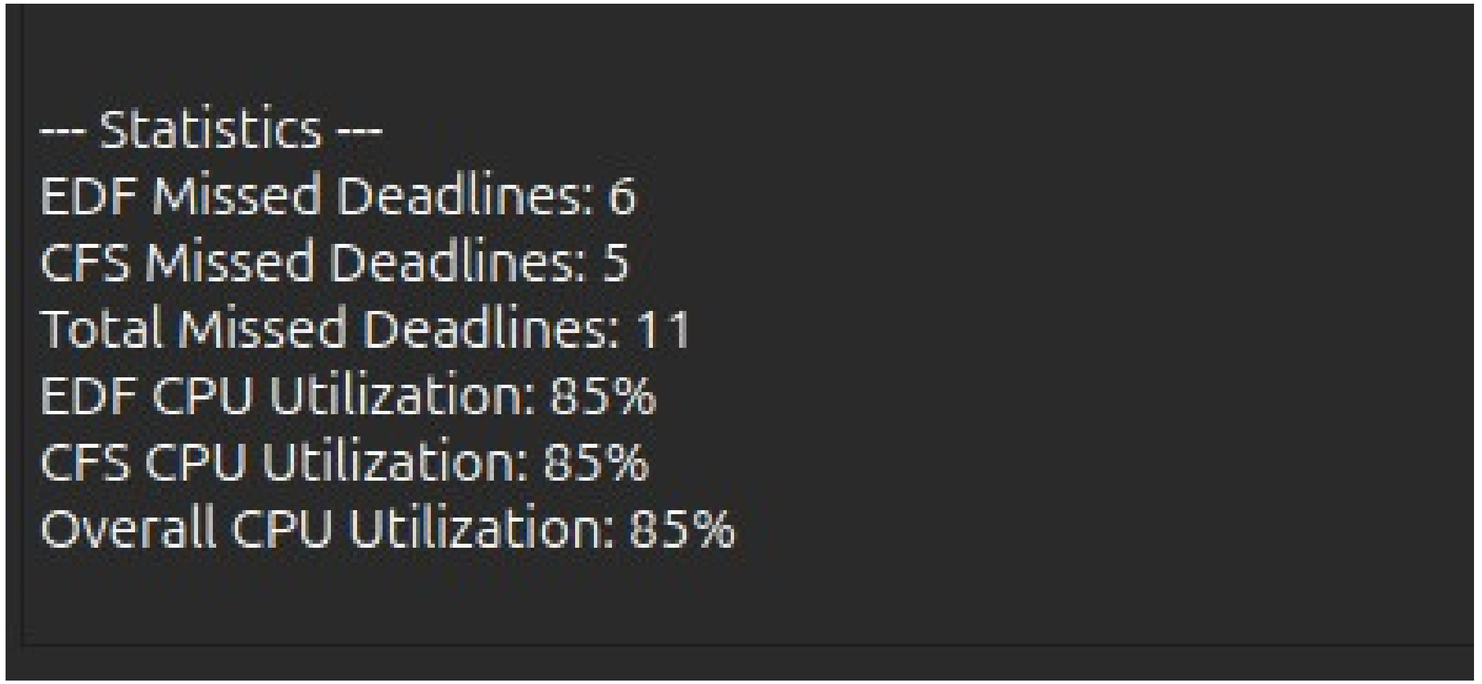
RAMAIAH
Institute of Technology

Output Screenshots



RAMAIAH
Institute of Technology

Output Screenshots



RAMAIAH
Institute of Technology

Challenges Faced

- Handling real-time data reading and updates
- Ensuring smooth step-by-step animation
- Managing synchronization between EDF and CFS events
- Designing an intuitive and responsive UI
- Debugging kernel-user space communication

RAMAIAH
Institute of Technology

Enhancements

- Added start, stop, restart, and step-by-step controls
- Integrated live Gantt chart animations for better visualization
- Displayed comparative metrics for EDF and CFS schedulers
- Improved user interaction for sequential process execution
- Enhanced error handling for proc file operations

RAMAIAH
Institute of Technology

Conclusion

- Successfully implemented EDF real-time scheduling with interactive visualization
- Enabled comparison with CFS scheduler through live Gantt charts and metrics
- Provided user-friendly controls for better understanding of scheduling behavior
- Project demonstrates effective simulation and analysis of scheduling algorithms