REAL-TIME PROCESS MONITORING AND ANOMALY DETECTION SYSTEM

A MINI PROJECT REPORT

Submitted by

KAVIYA V	**231901020**
SIVARANGINI Y	**231901051**
KEERTHANA S	**231901022**

in partial fulfillment of the award of the degree of

BACHELOR OF ENGINEERING
IN
COMPUTER SCIENCE AND ENGINEERING

RAJALAKSHMI ENGINEERING COLLEGE, CHENNAI

An Autonomous Institute

CHENNAI

MAY 2025

BONAFIDE CERTIFICATE

Certified that this project **“REAL-TIME PROCESS MONITORING AND ANOMALY DETECTION SYSTEM”** is the bonafide work of **“KAVIYA V, SIVARANGINI Y, KEERTHANA S”** who carried out the project work under my supervision.

SIGNATURE

Mrs. JANANEE V

ASSISTANT PROFESSOR

Dept. of Computer Science and Engg,

Rajalakshmi Engineering College,

Chennai.

This mini project report is submitted for the viva voce examination to be held on _

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ABSTRACT

This project presents a Real-Time Process Monitoring and Anomaly Detection System to enhance system stability, security, and performance. It tracks active processes, CPU/memory usage, and system logs to detect anomalies such as resource overuse or unauthorized executions. Designed for IT environments, it helps identify potential threats like malware attacks or system failures. Key features include process tracking, resource utilization analysis, anomaly detection, log analysis, and a user-friendly dashboard. In addition to these features, the system offers real-time alerts and notifications, enabling swift responses to suspicious activities or performance issues. It integrates seamlessly with existing IT infrastructure, minimizing deployment complexity and ensuring smooth adoption. The dashboard provides customizable views and filters, allowing users to focus on specific processes, timeframes, or anomaly types for improved monitoring. Furthermore, historical data analysis capabilities support the identification of long-term trends and patterns, enabling predictive maintenance and strategic decision-making.

ACKNOWLEDGEMENT

We express our sincere thanks to our honorable chairman **MR. S. MEGANATHAN** and the chairperson **DR. M.THANGAM MEGANATHAN** for their timely support and encouragement.

We are greatly indebted to our respected and honorable principal **Dr. S.N. MURUGESAN**

for his able support and guidance.

No words of gratitude will suffice for the unquestioning support extended to us by our Head Of the Department **Mr. BENIDICT JAYAPRAKASH NICHOLAS M.E Ph.D.,** for being ever supporting force during our project work.

We also extend our sincere and hearty thanks to our internal guide **Mrs.V JANANEE,** for her valuable guidance and motivation during the completion of this project.

Our sincere thanks to our family members, friends and other staff members of computer

science engineering.

1. KAVIYA V

2. SIVARANGINI Y

3. KEERTHANA S

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTIO N

1.1 INTRODUCTION

The Real-Time Process Monitoring and Anomaly Detection System is designed to monitor system processes dynamically and detect any unusual activity related to CPU, memory, or I/O usage. This system also simulates CPU scheduling (e.g., Round Robin or SJF) to optimize performance. The tool helps in visualizing real-time data on a web-based dashboard and provides alerts for anomalies**.**

1.2 SCOPE OF THE WORK

This project aims to offer a reliable, real-time tool for operating system-level process management. It helps system administrators and developers by offering live monitoring, scheduling simulation, and anomaly detection using visual insights. The scope includes cross-platform compatibility, modular design, and integration with web dashboards for easy accessibility.

1.3 PROBLEM STATEMENT

Modern operating systems handle numerous background and foreground processes, but lack detailed real-time visualization tools with intelligent anomaly detection capabilities. System overloads and abnormal process behavior often go unnoticed until performance deteriorates. There is a need for a lightweight, interactive, and intelligent process monitoring tool that combines scheduling logic and anomaly detection into a unified system.

1.4 AIM AND OBJECTIVES OF THE PROJECT

This project aims to build a web-based system for real-time process monitoring and resource usage logging. It simulates CPU scheduling algorithms to optimize performance and detect anomalies in CPU, memory, and I/O usage. The system visualizes data using HTML, CSS, JavaScript, and Chart.js. It also provides real-time alerts for suspicious behavior. Overall, it enhances system awareness and response through a user-friendly interface.

CHAPTER 2

SYSTEM SPECIFICATIONS:

2.1 HARDWARE SPECIFICATIONS

Component	: Specification
Processor	: Intel i5
Memory Size	: 8 GB (Minimum)
HDD/SSD	: 256 GB (Minimum)

2.2 SOFTWARE SPECIFICATIONS

Component	: Technology Used
Operating System	: Windows 10
Frontend	: HTML, CSS, JavaScript
Backend	: Python (Flask)
Database	: SQLite

Visualization	**:** Chart.js
Used	**:** Python, JavaScript, SQL Languages

CHAPTER 3

MODULE DESCRIPTION

3.1. Admin Module

The Admin has full access to monitor all system processes in real-time, configure alert thresholds, and view detailed logs.

3.2. User Module

Users can log in to view their own system performance, including CPU, memory, and I/O usage. They can access visualizations of resource usage, view alerts related to anomalies, and simulate CPU scheduling for educational or analysis purposes.

3.3. Process Monitoring Module

This module continuously tracks active processes and logs their resource usage. It collects real-time data on CPU, memory, and I/O performance.

3.4. Anomaly Detection Module

This module uses defined thresholds or machine learning models to detect unusual behavior in system resources. When anomalies are detected, alerts are triggered and displayed to the Admin.

3.5. Visualization Module

Utilizes modern web technologies such as HTML, CSS, JavaScript, and Chart.js to present real-time system data in an intuitive and interactive format like graphs and dashboards**.**

3.6. Scheduling Simulation Module

Implements common CPU scheduling algorithms like FCFS, SJF, and Round Robin to simulate process handling efficiency. Useful for academic analysis or optimizing system performance.

CHAPTER 4

SOURCE CODE:

4.1. app.py

```
from flask import Flask, render_template, jsonify import
psutil

import          random
import time

app = Flask(__name__)

#    Home    Page    -    Displays    CPU    usage    graph
@app.route("/")

def home():

    return render_template("index.html")

#    Load    Processes    -    Categorizes    processes    into    "App"    and    "Background"
@app.route("/load_processes")

def load_processes():


    apps, background = [], []


    for  process  in  psutil.process_iter(attrs=['pid',  'name',  'cpu_percent']):
        try:

            if    process.info['cpu_percent']    >    5:    #    Assume    apps    use    more    CPU
                apps.append(process.info)

            else:


                background.append(process.info)


        except    (psutil.NoSuchProcess,    psutil.AccessDenied):
            pass

    return render_template("load_processes.html", apps=apps, background=background)


# Run Scheduler - Implements First-Come, First-Serve (FCFS) def
get_processes():

processes = []
```

```
    for  proc  in  psutil.process_iter(attrs=['pid',  'name',  'cpu_percent']):
        try:

            processes.append({
                "pid": proc.info['pid'],

                "name": proc.info['name'],

                "cpu": proc.info['cpu_percent']  # Correctly fetch CPU usage


            })


        except (psutil.NoSuchProcess, psutil.AccessDenied, psutil.ZombieProcess):
            continue

    return processes

@app.route('/run_scheduler')
def run_scheduler():

    processes = get_processes()


    return render_template("run_scheduler.html", processes=processes)
#   Check   Anomalies   -   Detects   high   CPU   usage   processes
@app.route("/check_anomalies")

def              check_anomalies():
    anomalies = []

    for  process  in  psutil.process_iter(attrs=['pid',  'name',  'cpu_percent']):
        try:

            if      process.info['cpu_percent']      >      50:      #      Threshold      for      high      usage
                anomalies.append(process.info)

        except (psutil.NoSuchProcess, psutil.AccessDenied):
            pass
    return render_template("check_anomalies.html", anomalies=anomalies)
@app.route("/cpu_usage")
def cpu_usage():
    processes = []

    for proc in psutil.process_iter(attrs=['pid', 'name', 'cpu_percent']):
        try:
            processes.append({

                "name":      proc.info['name'],
                "cpu": proc.info['cpu_percent']
            })

        except     (psutil.NoSuchProcess,      psutil.AccessDenied):
            continue
    return jsonify({"cpu": psutil.cpu_percent(interval=1), "processes": processes})
if _name___== "_main_":
    app.run(debug=True)
```

This Flask app monitors system processes using psutil, categorizing them, checking CPU usage anomalies, and implementing a simple First-Come, First-Serve (FCFS) scheduler. It provides routes to display real-time CPU data, processes with high/low usage, and anomalies. The app serves HTML pages using render_template and sends CPU stats as JSON via /cpu_usage.

4.2. anomaly _detection.py

```
import random

def detect_anomalies():
    anomalies = []
    for _ in range(5):
        anomalies.append({
            "pid": random.randint(1000, 9999),
            "name": f"Process_{random.randint(1,10)}",
            "cpu": random.uniform(50, 99)  # Simulated high CPU usage
        })
    return anomalies
```

This function simulates detection of 5 high-CPU usage processes by generating random process IDs, names, and CPU usage values between 50 and 99. It creates a list of dictionaries, each representing a fake "anomalous" process. The function returns this list of simulated anomalies.

4.3. scheduler.py

```
import psutil

def run_scheduler():
    processes = []
    for proc in psutil.process_iter(['pid', 'name', 'cpu_percent']):
        processes.append(proc.info)
    processes.sort(key=lambda x: x['cpu_percent'], reverse=True)  # Highest CPU first
    return processes[:5]  # Show top 5 CPU-consuming processes
```

This function collects information (PID, name, CPU%) of all running processes using psutil. It sorts the processes in descending order based on CPU usage. Finally, it returns the top 5 processes consuming the most CPU.

CHAPTER 5

SCREENSHOTS

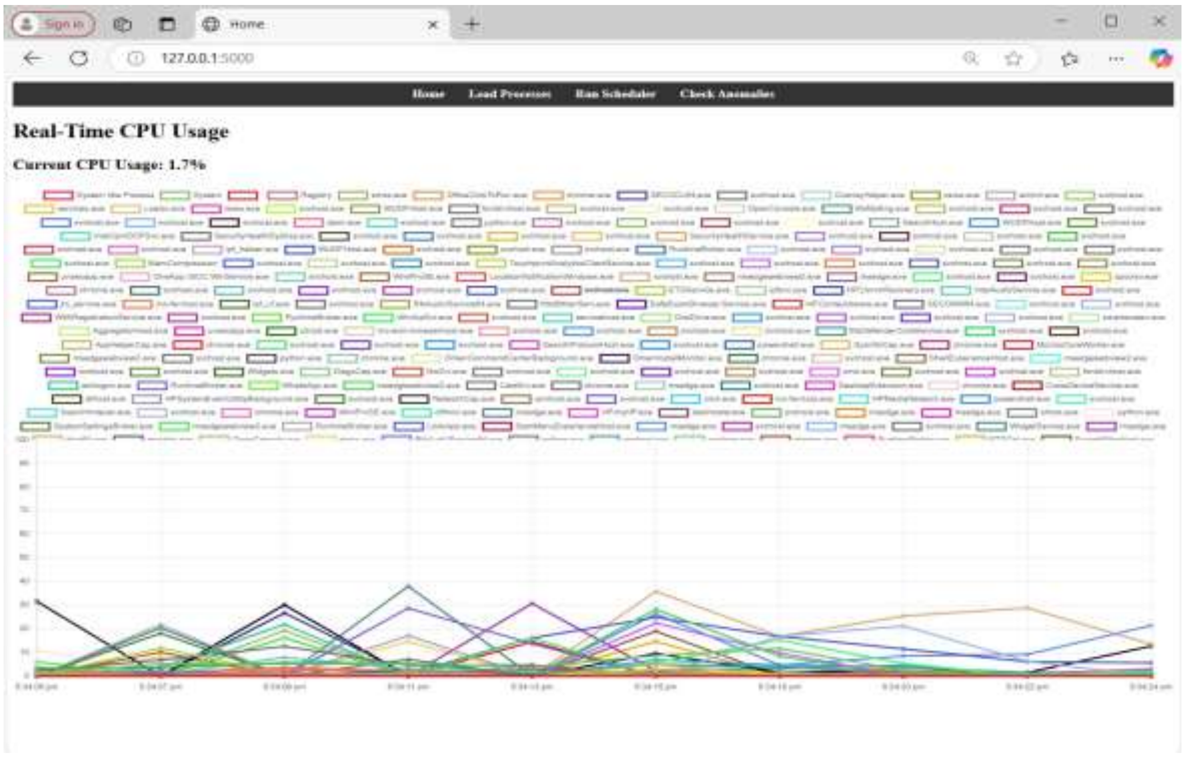
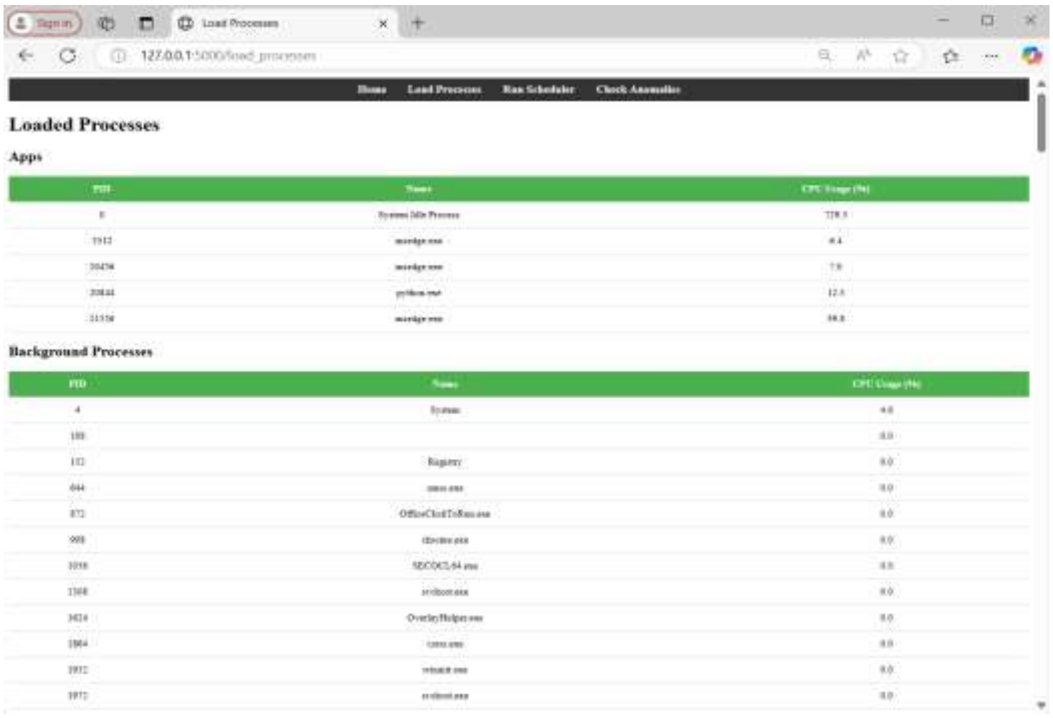


Fig 5.1 Home page



Loaded Processes

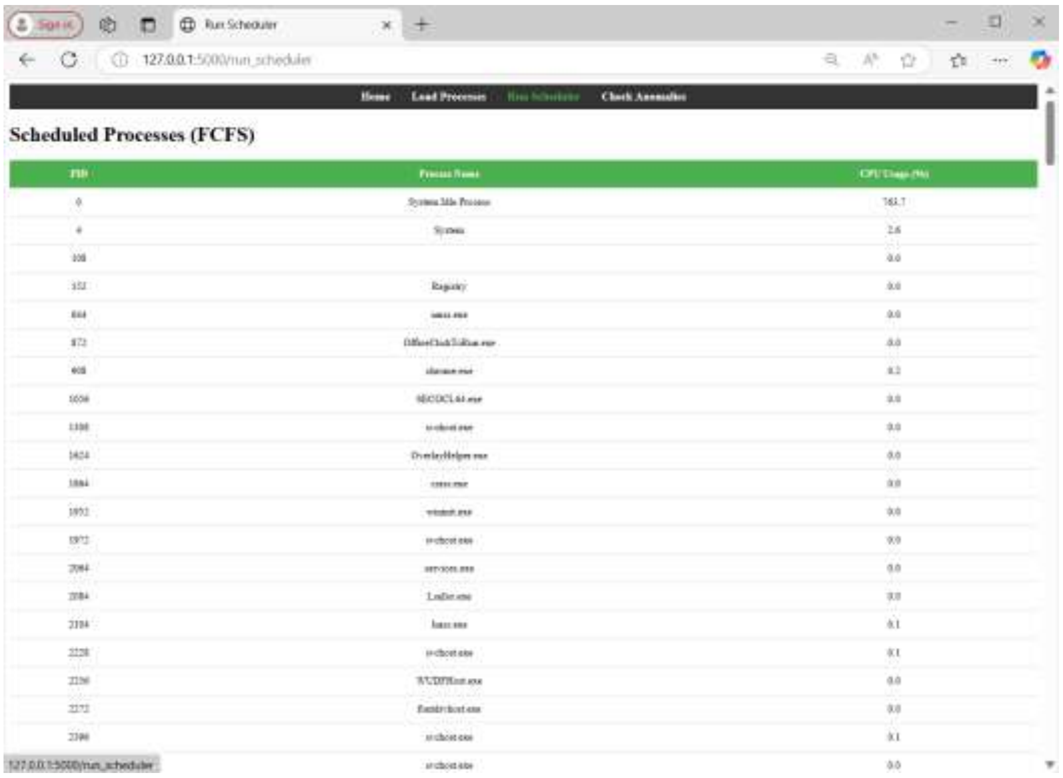
Apps

PID	Name	CPU Usage (%)
0	System Idle Process	728.3
7912	msedge.exe	8.4
20456	msedge.exe	7.8
20844	python.exe	12.5
21536	msedge.exe	98.8

Background Processes

PID	Name	CPU Usage (%)
4	System	4.0
108		0.0
132	Registry	0.0
644	smss.exe	0.0
872	OfficeClickToRun.exe	0.0
908	ctfmon.exe	0.0
1036	SECOCL64.exe	0.0
1308	svchost.exe	0.0
1624	OverlayHelper.exe	0.0
1884	csrss.exe	0.0
1932	wininit.exe	0.0
1972	svchost.exe	0.0

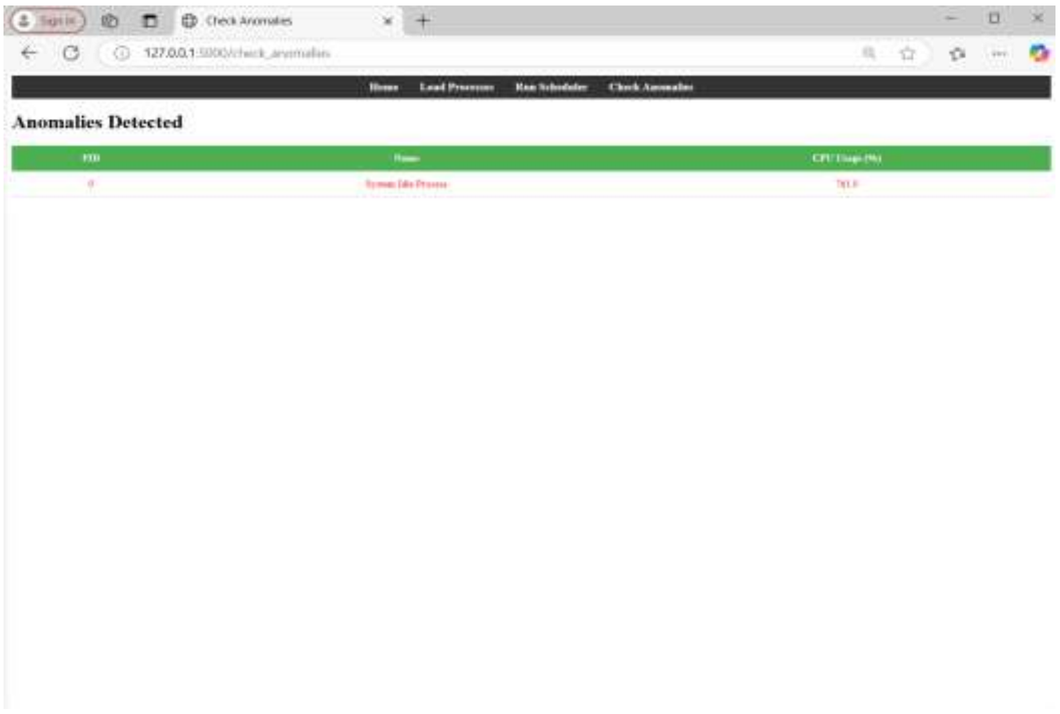
Fig 5.2 Process loading page



Scheduled Processes (FCFS)

PID	Process Name	CPU Usage (%)
0	System Idle Process	783.7
4	System	2.6
108		0.0
132	Registry	0.0
644	smss.exe	0.0
872	OfficeClickToRun.exe	0.0
908	ctfmon.exe	0.2
1036	SECOCL64.exe	0.0
1308	svchost.exe	0.0
1624	OverlayHelper.exe	0.0
1884	csrss.exe	0.0
1932	wininit.exe	0.0
1972	svchost.exe	0.0
2084	services.exe	0.0
2084	LsaIso.exe	0.0
2104	lsass.exe	0.1
2228	svchost.exe	0.1
2236	WUDFHost.exe	0.0
2272	fontdrvhost.exe	0.0
2300	svchost.exe	0.1
[illegible]	svchost.exe	0.0

Fig 5.3 Scheduled process page



Anomalies Detected

PID	Name	CPU Usage (%)
0	System Idle Process	[illegible]

Fig 5.4 Anomaly detection page

CHAPTER 6

CONCLUSION:

This project demonstrates a simple yet effective real-time process monitoring system using Flask and psutil. It allows users to visualize CPU usage, categorize processes, detect anomalies, and simulate scheduling using the FCFS approach. The web-based interface makes system diagnostics and resource tracking more accessible, highlighting the potential for integrating backend process analysis with frontend visualization tools.

CHAPTER 7

REFERENCES

1. William Stallings, *Operating Systems: Internals and Design Principles*, Pearson Education.
2. S. Tanenbaum and H. Bos, *Modern Operating Systems*, 4th Edition, Pearson Education.
3. Psutil Documentation, *Process and System Utilities for Python*, available at: https://psutil.readthedocs.io/