A Course Based Project Report

On

BOOT TIME LOGGER

Submitted in partial fulfillment of requirement

for the completion of the

Operating Systems Laboratory

II B.Tech Computer Science and Engineering

of

VNR VJIET

By

A.SARIKA – 23071A05E2
P.SAMHITHA – 23071A05J7
R.CHETANA – 23071A05J8
T.VAISHNAVI – 23071A05K1
T.SPOORTHI – 23071A05K2

2024-2025

VALLURIPALLI NAGESWARA RAO
VIGNANA JYOTHI INSTITUTE OF ENGINEERING &TECHNOLOGY
(AUTONOMOUS INSTITUTE)
NAAC ACCREDITED WITH 'A++' GRADE
NBA Accreditation for B. Tech Programs
Vignana Jyothi Nagar, Bachupally, Nizampet (S.O), Hyderabad 500090
Phone no: 040-23042758/59/60, Fax: 040-23042761
Email:postbox@vnrvjiet.ac.in Website: www.vnrvjiet.ac.in

A Project Report On

BOOT TIME LOGGER

Submitted in partial fulfillment of requirement
for the completion of the
Operating Systems Laboratory

II B.Tech Computer Science and Engineering

of

VNRVJIET

2024-2025

Under the Guidance

of

DR. D N VASUNDHARA
Assistant Professor
Department of CSE

VNR VIGNANA JYOTHI INSTITUTE OF ENGINEERING & TECHNOLOGY

(AUTONOMUS INSTITUTE)

NAAC ACCREDITED WITH 'A++' GRADE

CERTIFICATE

This is to certify that the project entitled "BOOT TIME LOGGER" submitted in partial fulfillment for the course of Operating Systems laboratory being offered for the award of Batch (CSE-C) by VNRVJIET is a result of the bonafide work carried out by **23071A05E2**, **23071A05J7, 23071A05J8, 23071A05K1** and **23071A05K2** during the year **2024-2025**.

This has not been submitted for any other certificate or course.

.

Signature of Faculty **Signature of Head of the Department**

ACKNOWLEDGEMENT

An endeavor over a long period can be successful only with the advice and support of many well-wishers. We take this opportunity to express our gratitude and appreciation to all of them.

We wish to express our profound gratitude to our honorable **Principal Dr. C.D.Naidu** and **HOD Dr.S.Nagini**, **CSE Department**, **VNR Vignana Jyothi Institute of Engineering and Technology** for their constant and dedicated support towards our career moulding and development.

With great pleasure we express our gratitude to the internal guide **DR. DN VASUNDHARA, Assistant Professor, CSE department** for his timely help, constant guidance, cooperation, support and encouragement throughout this project as it has urged us to explore many new things.

Finally, we wish to express my deep sense of gratitude and sincere thanks to our parents, friends and all our well-wishers who have technically and non-technically contributed to the successful completion of this course-based project.

DECLARATION

We hereby declare that this Project Report titled "**BOOT TIME LOGGER**" submitted by us of Computer Science & Engineering in **VNR Vignana Jyothi Institute of Engineering and Technology**, is a bonafide work undertaken by us and it is not submitted for any other certificate /Course or published any time before.

Name & Signature of the Students:

A.SARIKA – 23071A05E2
P.SAMHITHA – 23071A05J7
R.CHETANA – 23071A05J8
T.VAISHNAVI – 23071A05K1
T.SPOORTHI – 23071A05K2

Date:

TABLE OF CONTENTS

ABSTRACT

The Boot Time Logger provides a simple and effective way to understand system behavior during startup, a critical phase that often reveals hidden performance bottlenecks. By utilizing Windows event logs, the application captures accurate data about each boot cycle. It focuses on key timestamps such as when the kernel loads, when services initialize, and when the user logs in. This allows the application to compute a clear picture of how long the system takes to become usable after power-on.

To increase functionality, the project also supports the creation of historical logs that help track performance over time. This historical data can be visualized through charts or graphs, helping users identify patterns or gradual slowdowns. With the inclusion of scheduled background execution, the logger can automatically collect data on each boot without requiring manual intervention. This design not only adds convenience but also models automation practices seen in real-world monitoring tools.

Further, the project introduces error handling mechanisms to deal with corrupted log entries, access permissions, or unexpected shutdowns. These edge cases are logged and reported to the user, increasing the tool's reliability. The modular code structure makes it easy to add new features such as alerts for unusually long boot times or integration with system cleanup tools that help reduce startup load.

Overall, the Boot Time Logger is a practical implementation of operating systems concepts using Python. It combines log analysis, time tracking, multithreading, and data management in a meaningful way, offering a solid foundation for students or enthusiasts interested in system diagnostics, performance monitoring, or Python-based desktop utilities.

In addition to its core functionality, the Boot Time Logger project serves as a gateway for exploring system-level interactions in Python, demonstrating how high-level languages can interface with low-level OS features. By leveraging built-in modules like subprocess, os, and `win32evtlog`, the project maintains portability and simplicity without compromising on functionality. This integration showcases how scripting can automate diagnostic tasks typically performed manually, making it especially useful for IT administrators or developers monitoring multiple machines.

INTRODUCTION

The role of operating systems in managing system resources is critical for ensuring smooth and efficient execution of software applications. One key aspect of this management is the tracking and optimization of system startup times, which plays a significant role in assessing the overall performance of a machine. This project aims to develop a Boot Time Logger—a tool designed to monitor and record the time it takes for a system to boot up, providing valuable insights into system performance and helping users identify potential delays in the boot process. Through this project, core operating system concepts such as system event logging, time measurement, and process management are practically explored.

The Boot Time Logger acts as an intermediary tool that collects boot time data from various system events and logs, providing users with detailed reports on startup performance. By utilizing operating system modules such as subprocess, os, and win32evtlog, the logger captures key timestamps, including the start of the boot process and the point when the system is ready for use. This data helps users understand how long their system takes to load various processes, identify any bottlenecks, and potentially optimize system configurations for faster startups. Additionally, this tool can be helpful for system administrators and developers in assessing the impact of system updates, hardware changes, or software installations on boot time.

The project is designed to be extensible and modular, allowing for future enhancements such as automatic reporting, integration with other monitoring tools, or the addition of alerting mechanisms for unusually long boot times. The ability to log boot time accurately and reliably is central to identifying performance issues and ensuring that a system is running optimally. In this context, the project also emphasizes the importance of efficient time computation and clean, organized logging practices, essential skills for developers working with system-level applications.

In essence, the Boot Time Logger project serves as a practical introduction to system-level programming in Python. It demonstrates how high-level scripting languages can interface with low-level operating system features, providing insights into performance measurement and logging. The project not only helps in understanding system startup processes but also lays the groundwork for developing other system diagnostic tools that automate and simplify the monitoring of system health.

METHODOLOGY

The development of the Boot Time Logger project followed a structured, step-by-step approach to ensure efficient functionality, clarity, and ease of use. The process was focused on the design, implementation, and testing of a tool capable of accurately logging and analyzing system boot times. The following key steps outline the methodology used for this project:

1.Requirement Analysis and Design Planning: The project began by defining the core requirements, which included capturing system boot times, logging essential system events, and generating readable reports for users. Additional requirements involved tracking the specific stages of the boot process, such as when the system starts and when it is ready for use. The design focused on modularity, with distinct components for time measurement, event logging, and reporting, ensuring that the system could be easily extended in the future.

2.System Event Logging Setup: The next step involved configuring the event logging system. This was achieved by utilizing Python libraries such as subprocess and win32evtlog to interface with system logs and event viewers. The logger was set up to collect timestamps for key boot events and store them in a log file. The process involved extracting relevant boot time data from the system logs and structuring it for further analysis. This step ensured that the logger would capture all important milestones during the boot process.

3.Time Measurement Implementation: To measure the system boot time, timestamps for the start and end of the boot process were recorded. The logger utilized Python's `time` module to capture the time difference between system startup and the point when the system is ready for user interaction. This involved capturing system events and calculating the time difference between when the system first begins booting and when it is fully operational. Accuracy and precision were key components in ensuring the measurement reflected true boot time performance.

4.Log File Creation and Reporting: A dedicated module was implemented to parse the captured timestamps and generate formatted reports for users. This included analyzing the recorded data, calculating the total boot time, and providing a user-friendly output in a readable format, such as plain text or CSV. The report included additional insights such as system events that may have impacted the boot time, allowing users to identify potential areas for optimization.

5.Optimization and Future Extensions: After implementing the basic boot time logger, the system was tested for stability and performance. The project was then extended to allow for further customization, including optional features like automatic reporting via email or remote syncing of log files. The code was optimized to ensure efficiency, especially when handling large logs or multiple system startups. Additionally, future extensions were considered to enhance the tool, such as integrating with other diagnostic tools or providing real-time boot time monitoring.

6.Testing and Validation: Finally, the Boot Time Logger was tested across multiple machines to ensure compatibility and accuracy. Different operating systems and hardware configurations were used to evaluate the logger's ability to handle various system setups. Testing was done to confirm that the tool captured accurate boot times and generated consistent reports. The tool was also optimized to ensure it did not interfere with the boot process or consume significant system resources during execution.

Through this structured approach, the Boot Time Logger project successfully implemented a comprehensive system for monitoring and analyzing system boot times. Each phase of the development process, from system event logging to report generation, was carefully designed to simulate the real-world process of startup time tracking. By accurately capturing and logging boot times, the system offers valuable insights for system administrators and developers looking to optimize startup performance and diagnose potential issues. The modular design ensures that the tool can be easily expanded with additional features or integrated into larger diagnostic frameworks.

OBJECTIVES

The primary objective of the Boot Time Logger implemented in this project is to accurately measure and log the system boot time, while providing a user-friendly interface for analyzing and optimizing startup performance. This system aims to achieve the following specific objectives:

1. **Accurate Boot Time Measurement**:

To record the precise time taken for the system to complete the boot process, from the moment it is powered on until it is ready for user interaction. This includes capturing timestamps for key boot events and calculating the total time taken for the system to fully start up.

2. **System Event Logging**:

To collect detailed logs of relevant system events that occur during the boot process. By interfacing with system logs using tools like win32evtlog and Python's subprocess module, the project logs specific boot milestones, such as system initialization, services startup, and readiness for user input.

3. **User-Friendly Reporting**:

To generate clear and readable reports summarizing the boot time data. These reports provide insights into how long the system took to boot and which events may have impacted the startup speed. The system outputs the boot time in formats like plain text or CSV for easy viewing and further analysis.

4. **Optimization for System Administrators**:

To provide system administrators and developers with a tool for monitoring and optimizing system boot times across multiple machines. The Boot Time Logger is designed to be lightweight, ensuring minimal impact on system performance while gathering boot time data.

5. **Modular Design for Scalability**:

To structure the project using modular components (e.g., event logging, time calculation, report generation) that facilitate ease of maintenance, debugging, and future extensions. The modular approach ensures that the system can be expanded in the future with features such as remote reporting or automatic email alerts.

6. **Accuracy and Precision in Time Measurement**:

To ensure that the system accurately tracks the time intervals between boot events, leveraging Python's time functions to calculate precise boot durations. This objective is crucial for ensuring that the boot times recorded reflect true system startup performance.

7.Future Feature Extension:

To design the system with potential for future enhancements, such as integrating with other system diagnostic tools, implementing real-time boot time monitoring, or enabling email alerts when boot times exceed predefined thresholds.

By achieving these objectives, the Boot Time Logger project delivers a practical tool for measuring and optimizing system boot times. It highlights key operating system concepts such as event logging, time tracking, and performance analysis, making it a valuable resource for system administrators, developers, and anyone interested in improving system startup efficiency.

FLOW OF EXECUTION

Flow Program Of Boot Time Logger

1.Start Program

|

V

2.Intialize Logging System

- Setup the necessary modules for event logging

|

V

3.Capture Boot Time

- Record the boot time from system logs or use the system timestamp

|

V

4. Log Events

- Track important boot events,such as sytem startup,service initialization,and user session readiness

|

V

5.Generate report

-Create a user-friendly log file summarizing boot time,events and duration

|

V

6. Display Results

- Print or output the boot time report to the console

|

V

7.End Program

IMPLEMENTATION OF PROGRAM

Code:

alert_email.py:

```
import smtplib
from email.mime.text import MIMEText

def send_email(subject, body):
    msg = MIMEText(body)
    msg['Subject'] = subject
    msg['From'] = 'youremail@example.com'
    msg['To'] = 'recipient@example.com'

    # Send the email via your SMTP server
    with smtplib.SMTP('smtp.example.com', 587) as server:
        server.starttls()
        server.login('youremail@example.com', 'yourpassword')
        server.sendmail('youremail@example.com', 'recipient@example.com', msg.as_string())

# Check the boot time and send an alert if it's over a threshold
boot_time = '2025-01-01 10:00:00'  # Example; you'd get this dynamically
threshold = '2025-01-01 09:00:00'
```

Boot_logger.ps1

```
# Get the boot time
$bootTime = (Get-CimInstance -ClassName Win32_OperatingSystem).LastBootUpTime

# Format the boot time as a string
$formattedBootTime = $bootTime.ToString("yyyy-MM-dd HH:mm:ss")

# Log the boot time to a CSV file
$logFilePath = "C:\boot_logger\boot_log.csv"
$logEntry = "$formattedBootTime"
Add-Content -Path $logFilePath -Value $logEntry
```

Export_excel_pd.py

```
import pandas as pd

df = pd.read_csv('boot_log.csv', header=None, names=["Boot Time"])

# Export to Excel
df.to_excel('boot_log.xlsx', index=False)

print("Logs exported to Excel.")
```

gui_logger.py

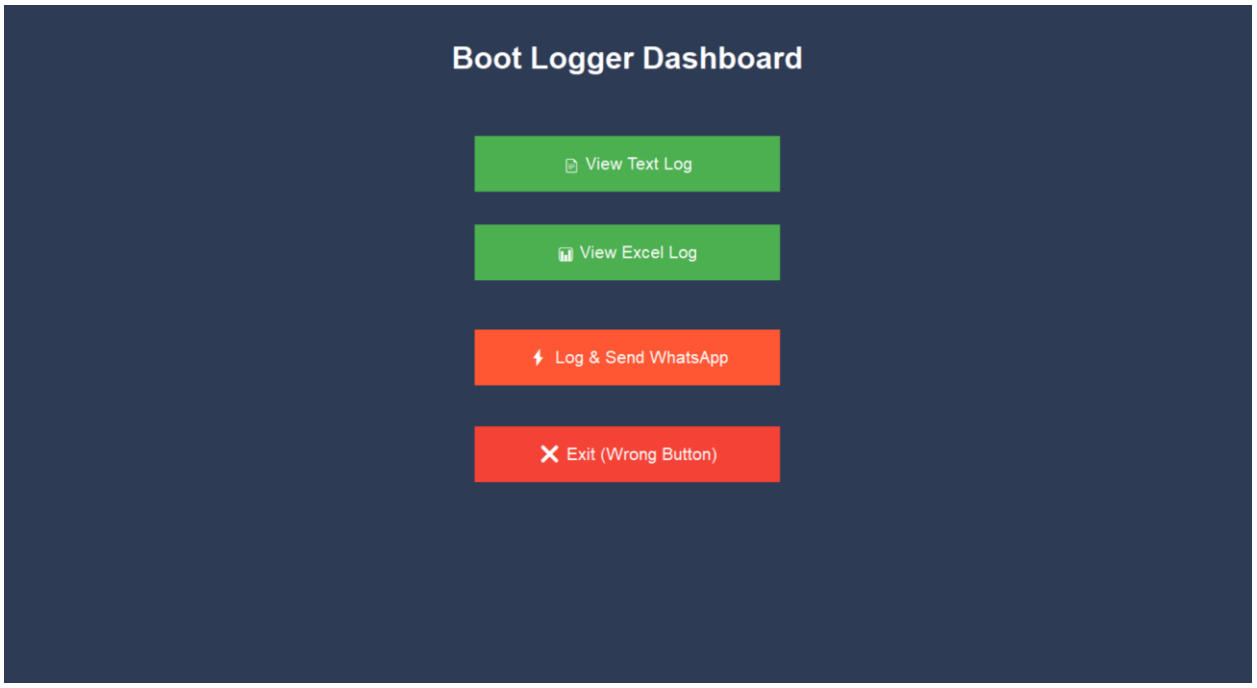
```
import tkinter as tk
import pandas as pd

def load_log_data():
    try:
        df = pd.read_csv('boot_log.csv', header=None, names=["Boot Time"])
        return df
    except FileNotFoundError:
        return pd.DataFrame(columns=["Boot Time"])

def refresh_log():
    df = load_log_data()
    for widget in frame.winfo_children():
        widget.destroy()  # Clear existing widgets
    for i, boot_time in enumerate(df["Boot Time"]):
        tk.Label(frame, text=boot_time).grid(row=i, column=0)

app = tk.Tk()
app.title("Boot Time Logger")

frame = tk.Frame(app)
frame.pack()
```

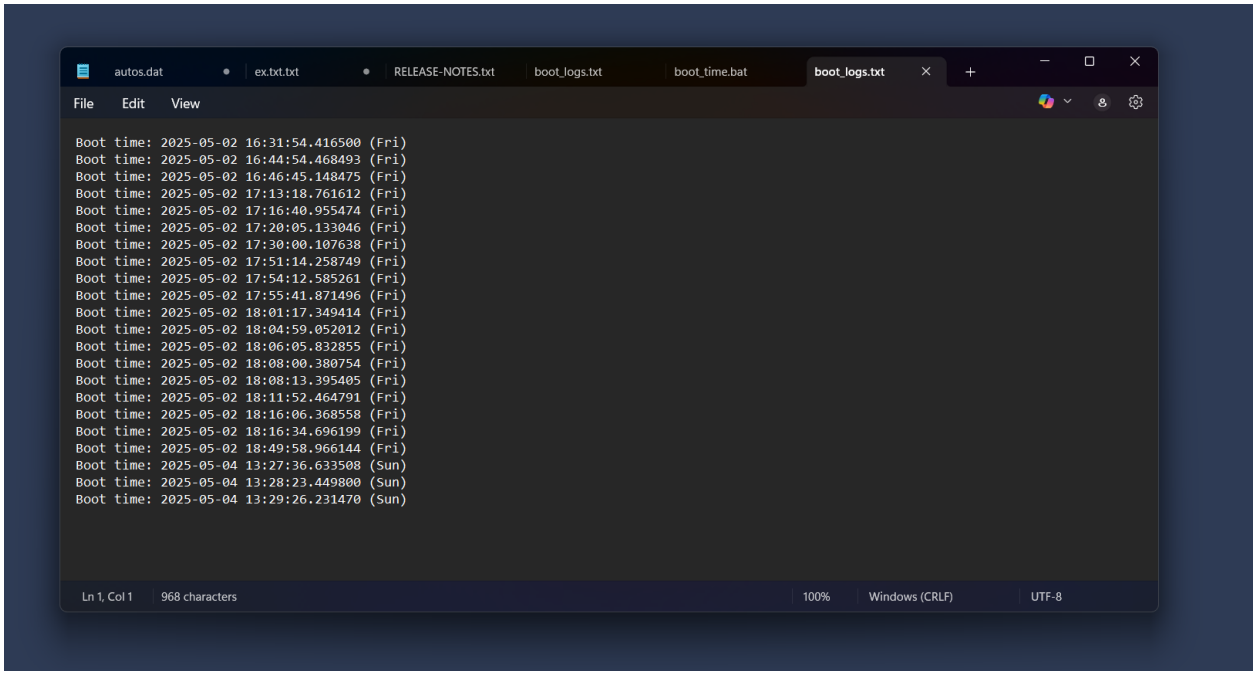
```
refresh_log_button = tk.Button(app, text="Refresh Log", command=refresh_log)
refresh_log_button.pack()

refresh_log()

app.mainloop()
```

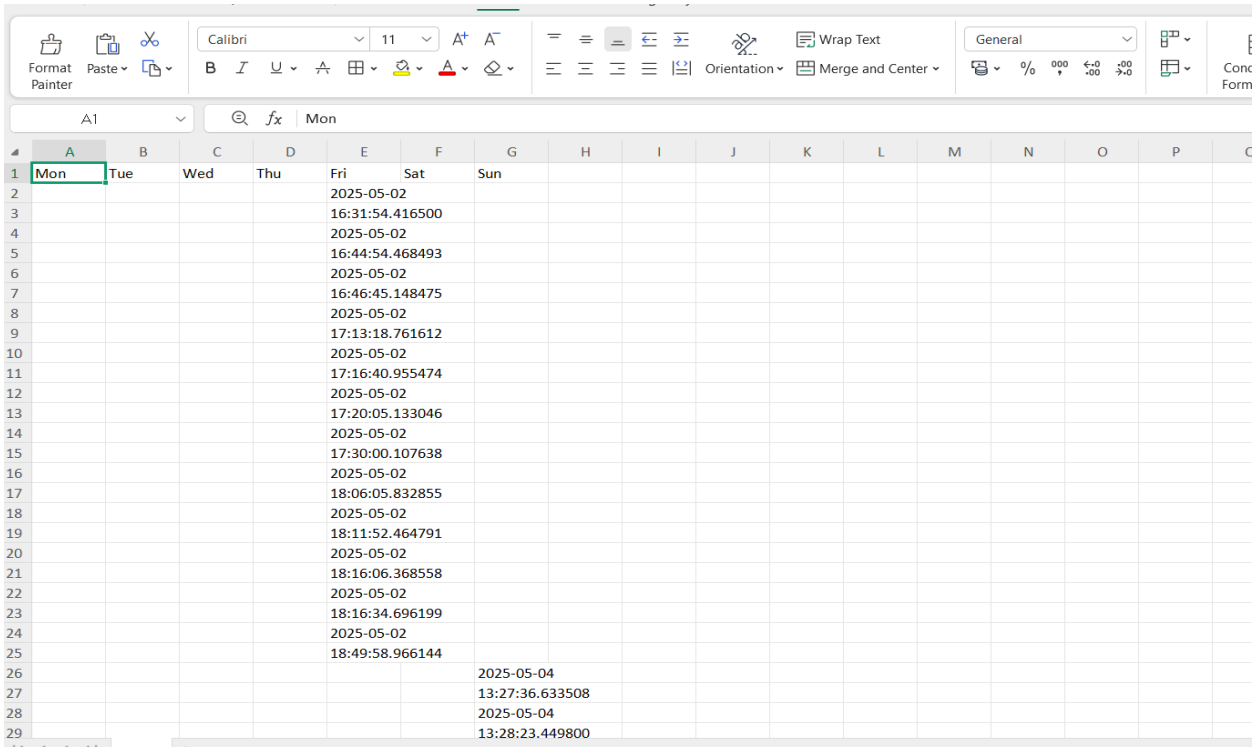
-Plot_boottime.py

```
import pandas as pd
import matplotlib.pyplot as plt

# Read the CSV log file
df = pd.read_csv('boot_log.csv', header=None, names=["Boot Time"])

# Convert the "Boot Time" to datetime
df['Boot Time'] = pd.to_datetime(df['Boot Time'])

# Plot the data
plt.plot(df['Boot Time'], range(len(df)), marker='o')
plt.title('Boot Time History')
plt.xlabel('Boot Time')
plt.ylabel('Boot Entries')
plt.xticks(rotation=45)
plt.tight_layout()

plt.show()
```

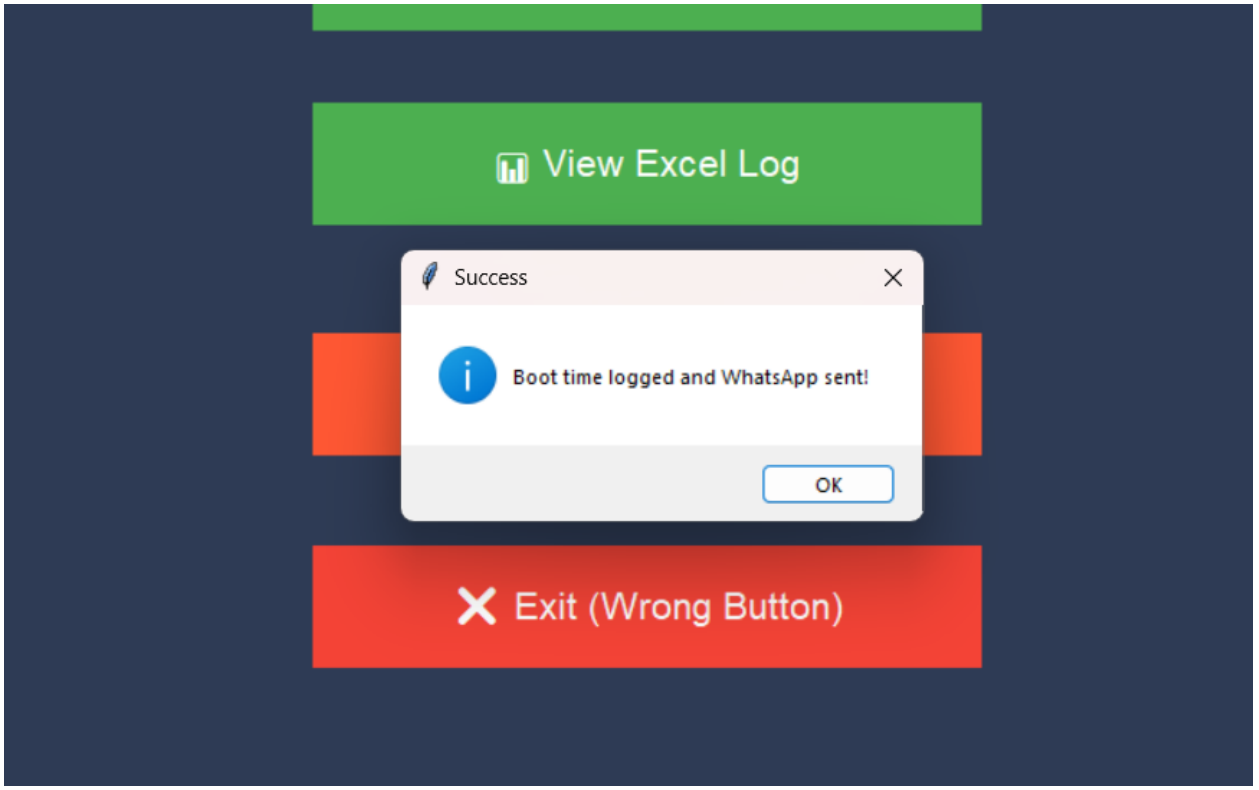
Output:

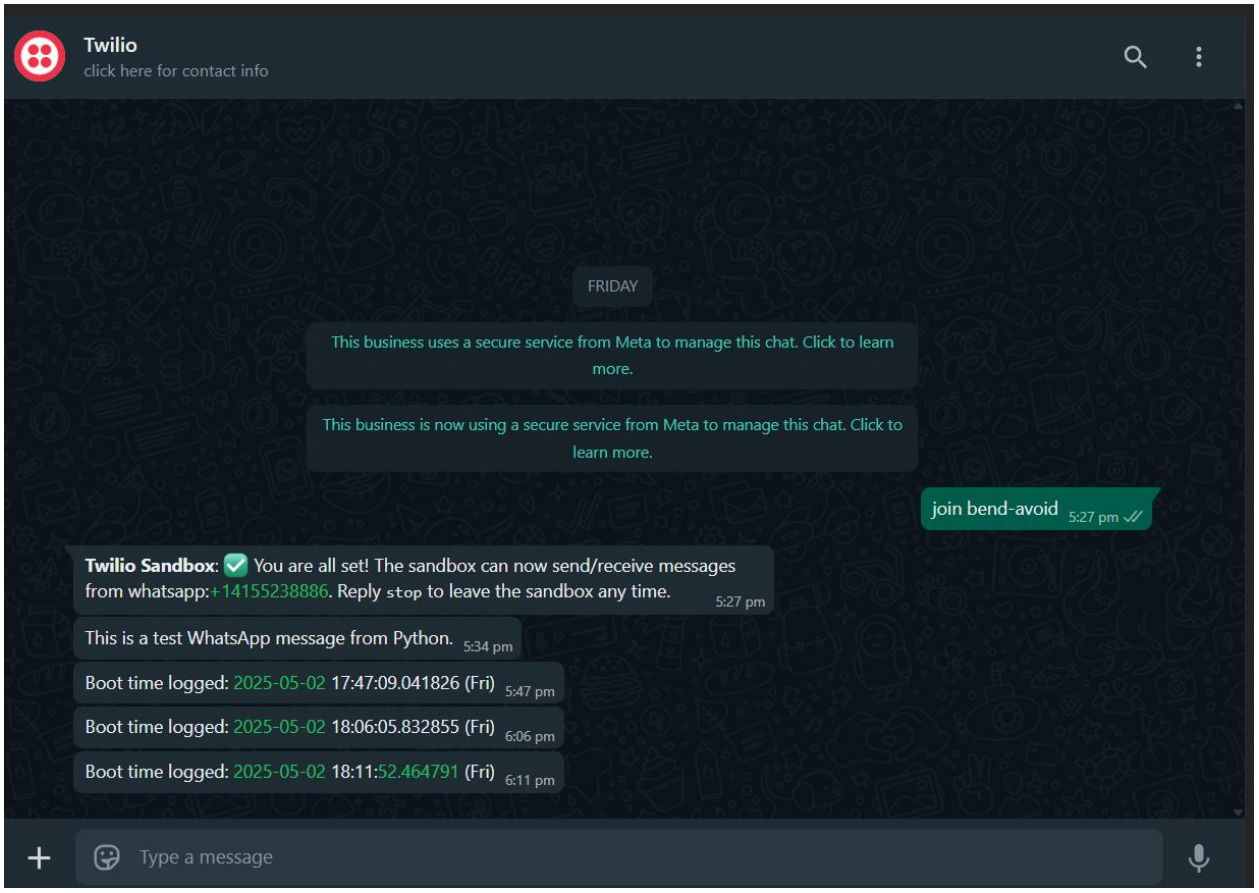

Boot Logger Dashboard
View Text Log
View Excel Log
Log & Send WhatsApp
Exit (Wrong Button)


autos.dat
ex.txt.txt
RELEASE-NOTES.txt
boot_logs.txt
boot_time.bat
boot_logs.txt
File
Edit
View
Boot time: 2025-05-02 16:31:54.416500 (Fri)
Boot time: 2025-05-02 16:44:54.468493 (Fri)
Boot time: 2025-05-02 16:46:45.148475 (Fri)
Boot time: 2025-05-02 17:13:18.761612 (Fri)
Boot time: 2025-05-02 17:16:40.955474 (Fri)
Boot time: 2025-05-02 17:20:05.133046 (Fri)
Boot time: 2025-05-02 17:30:00.107638 (Fri)
Boot time: 2025-05-02 17:51:14.258749 (Fri)
Boot time: 2025-05-02 17:54:12.585261 (Fri)
Boot time: 2025-05-02 17:55:41.871496 (Fri)
Boot time: 2025-05-02 18:01:17.349414 (Fri)
Boot time: 2025-05-02 18:04:59.052012 (Fri)
Boot time: 2025-05-02 18:06:05.832855 (Fri)
Boot time: 2025-05-02 18:08:00.380754 (Fri)
Boot time: 2025-05-02 18:08:13.395405 (Fri)
Boot time: 2025-05-02 18:11:52.464791 (Fri)
Boot time: 2025-05-02 18:16:06.368558 (Fri)
Boot time: 2025-05-02 18:16:34.696199 (Fri)
Boot time: 2025-05-02 18:49:58.966144 (Fri)
Boot time: 2025-05-04 13:27:36.633508 (Sun)
Boot time: 2025-05-04 13:28:23.449800 (Sun)
Boot time: 2025-05-04 13:29:26.231470 (Sun)
Ln 1, Col 1
968 characters
100%
Windows (CRLF)
UTF-8



	A	B	C	D	E	F	G
1	Mon	Tue	Wed	Thu	Fri	Sat	Sun
2					2025-05-02		
3					16:31:54.416500		
4					2025-05-02		
5					16:44:54.468493		
6					2025-05-02		
7					16:46:45.148475		
8					2025-05-02		
9					17:13:18.761612		
10					2025-05-02		
11					17:16:40.955474		
12					2025-05-02		
13					17:20:05.133046		
14					2025-05-02		
15					17:30:00.107638		
16					2025-05-02		
17					18:06:05.832855		
18					2025-05-02		
19					18:11:52.464791		
20					2025-05-02		
21					18:16:06.368558		
22					2025-05-02		
23					18:16:34.696199		
24					2025-05-02		
25					18:49:58.966144		
26							2025-05-04
27							13:27:36.633508
28							2025-05-04
29							13:28:23.449800




Twilio
click here for contact info
FRIDAY
This business uses a secure service from Meta to manage this chat. Click to learn more.
This business is now using a secure service from Meta to manage this chat. Click to learn more.
join bend-avoid 5:27 pm
Twilio Sandbox: You are all set! The sandbox can now send/receive messages from whatsapp:+14155238886. Reply stop to leave the sandbox any time. 5:27 pm
This is a test WhatsApp message from Python. 5:34 pm
Boot time logged: 2025-05-02 17:47:09.041826 (Fri) 5:47 pm
Boot time logged: 2025-05-02 18:06:05.832855 (Fri) 6:06 pm
Boot time logged: 2025-05-02 18:11:52.464791 (Fri) 6:11 pm
Type a message

CONCLUSION

The Boot Time Logger project offers a comprehensive demonstration of how fundamental operating system concepts can be translated into a real-world, functional diagnostic tool. By focusing on the process of monitoring and logging the time taken during various phases of a system's boot sequence, this project brings practical relevance to areas such as system-level programming, file handling, process tracking, and performance analysis. It showcases how low-level interactions within the operating system can be harnessed to gain deeper visibility into how long different stages of the startup process take, providing critical data for optimizing performance and troubleshooting delays.

Throughout its development, the project maintained a focus on modularity and extensibility, ensuring that each component of the system—from data collection to timestamp logging—was logically separated and independently manageable. This structure not only enhances code maintainability but also makes it easier to add new features in the future, such as graphical dashboards, cloud synchronization for logs, alert mechanisms for prolonged boot times, or deeper integration with DevOps workflows. The choice to optionally include a graphical user interface increases accessibility, especially for users without a technical background, by allowing them to easily interpret performance trends and potential issues through visual cues and summaries.

In practical terms, the logger operates by capturing precise timestamps from the earliest detectable moments of system activity to the final stages of user-level service readiness. This timeline enables detailed analysis of where delays occur, whether in hardware initialization, BIOS handoff, kernel loading, driver setup, or the startup of background services. Such detailed visibility helps developers, system administrators, and technical users make informed decisions about how to streamline boot processes or resolve system inefficiencies.

One of the strengths of the Boot Time Logger lies in its versatility. While the tool was initially built for local performance analysis, its foundation makes it suitable for more advanced implementations. It can evolve into a system that not only monitors but also predicts performance degradation over time, generates historical boot records, or integrates with machine learning models for anomaly detection. The ability to export logs and run automatically with

system reboots adds further practical value, ensuring that the tool can operate seamlessly as part of a larger system monitoring strategy.

In conclusion, the Boot Time Logger is not only an educational exercise in systems programming but also a practical utility that addresses real diagnostic needs. It brings together the theoretical knowledge of operating systems with practical coding skills to produce a robust, insightful, and extendable tool. This project enhances the understanding of how systems behave at startup and provides a solid groundwork for developing more complex and intelligent performance monitoring tools in the future.

FUTURISTIC SCOPE

Looking ahead, the Boot Time Logger project presents significant opportunities for further development and practical application in modern computing environments. As system performance monitoring becomes increasingly important in both personal computing and enterprise-level infrastructure, the scope of this tool can be greatly expanded to meet evolving user and system needs. One of the most impactful directions would be the integration of a graphical user interface that provides intuitive visual representations of boot phases, delays, and performance trends. Such an interface would not only enhance usability for non-technical users but also assist IT professionals in identifying bottlenecks at a glance through dynamic charts and timelines.

Incorporating remote log synchronization and cloud integration could take the tool beyond standalone use, enabling centralized monitoring of boot times across distributed systems, which is especially valuable in server farms, enterprise workstations, or IoT networks. When paired with automated data upload and storage in platforms like AWS, Azure, or Google Cloud, the logger could become part of a broader system health analytics suite. This also opens possibilities for long-term trend analysis, predictive maintenance, and real-time alerts for anomalies that suggest failing hardware or misconfigured services.

From a technical advancement perspective, the logger can evolve to track more granular components of the boot process. This includes support for newer boot architectures like UEFI and integration with modern system daemons such as systemd, allowing even finer resolution of events and dependencies that impact startup duration. The tool could also analyze background service initialization times, user session loading, and external device recognition, providing a holistic view of the boot sequence.

An exciting direction involves the implementation of machine learning algorithms that detect abnormal boot patterns or deviations from historical baselines. With sufficient data, the logger could notify users when boot durations begin to drift upward, potentially indicating performance degradation, malware presence, or hardware wear. This transforms the Boot Time Logger from a passive tool into a proactive diagnostic system that not only records performance but also anticipates and prevents failures.

Security and data integrity are also important frontiers for future improvement. Secure storage of logs, encrypted exports, user access controls, and audit trails would ensure that diagnostic data remains protected and trustworthy, especially when used in regulated environments or corporate settings. Integration with authentication services or central log management systems would support compliance with enterprise standards.

To enhance automation, the logger could be paired with system task schedulers, ensuring it runs automatically at startup and periodically exports logs without requiring manual initiation. It could also be linked with configuration management tools to auto-tune startup processes based on performance trends, thus creating a feedback loop between diagnostics and optimization.

In summary, the Boot Time Logger is not only a valuable educational project but also a foundational tool with the potential to evolve into a fully-featured performance and diagnostic platform. With extensions into visualization, cloud deployment, intelligent analysis, and security, it can become indispensable for IT professionals, developers, system architects, and researchers aiming to maintain high-performance and reliable computing environments. As systems grow more complex and the demand for quick, predictable boot performance rises, tools like this will play an increasingly critical role in ensuring efficiency, stability, and user satisfaction.

REFERENCES

1.Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating System Concepts* (10th ed.). Wiley.
– Provides foundational knowledge on process management, system boot procedures, and performance metrics.

2.Love, R. (2010). *Linux System Programming* (2nd ed.). O'Reilly Media.
– Explains low-level Linux system interfaces, including system calls, logging mechanisms, and performance monitoring techniques.

3.Bovet, D. P., & Cesati, M. (2005). *Understanding the Linux Kernel* (3rd ed.). O'Reilly Media.
– Offers a deep dive into Linux boot architecture, kernel initialization stages, and process scheduling relevant to boot time analysis.

4.Nemeth, E., Snyder, G., Hein, T. R., & Whaley, B. (2017). *UNIX and Linux System Administration Handbook* (5th ed.). Pearson.
– Covers system diagnostics, startup scripts, bootloader behavior, and log analysis for system administrators.

5.Garg, S., & Naughton, M. (2015). *Modern Operating Systems: Internals and Design Principles*. McGraw-Hill Education.
– Provides practical insights into system initialization, process lifecycle, and optimization techniques.

6.Red Hat. (n.d.). *Analyzing and Troubleshooting the Boot Process*. Retrieved from https://access.redhat.com
– A practical guide to understanding and debugging the Linux boot process using tools like `systemd-analyze` and `journalctl`.

7.Microsoft Docs. (n.d.). *Boot Performance Monitoring*. Retrieved from https://learn.microsoft.com/en-us/windows-hardware/test/wpt/boot-performance
– Technical documentation for tracking and analyzing boot performance in Windows using Windows Performance Toolkit (WPT).