A

Project Report

On

Software Resource Manager for Devices

Submitted in partial fulfillment of the requirement for the degree of

Bachelor of Technology

In

Computer Science and Engineering

By

Neha Kanwal 2261389

Lokesh Rathour 2261336

Lokesh Rawat 2265010

Bhawana Mehta 2261259

Under the Guidance of

Mr. Prince Kumar

ASSISTANT / ASSOCIATE PROFESSOR

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

GRAPHIC ERA HILL UNIVERSITY, BHIMTAL CAMPUS

SATTAL ROAD, P.O. BHOWALI,

DISTRICT- NAINITAL-263132

2024-2025

STUDENT'S DECLARATION

We, **Neha Kanwal, Lokesh Rathour, Lokesh Rawat** and **Bhawana Mehta** hereby declare the work, which is being presented in the project, entitled **Software Resource Manager For Embedded Devices** in partial fulfillment of the requirement for the award of the degree **Bachelor of Technology (B.Tech.)** in the session **2024-2025**, is an authentic record of my work carried out under the supervision of **Mr. Prince Kumar**.

The matter embodied in this project has not been submitted by me for the award of any other degree.

Date:

Neha Kanwal

Lokesh Rathour

Lokesh Rawat

Bhawana Mehta

CERTIFICATE

The project report entitled "**Software Resource Manager for Devices**" submitted by **Neha Kanwal , Lokesh Rathour , Lokesh Rawat , Bhawana Mehta** of B.Tech.(CSE) to Graphic Era Hill University Bhimtal Campus for the award of bonafide work carried out by them. They have worked under my guidance and supervision and fulfilled the requirement for the submission of a report.

Mr. Prince Kumar
(Project Guide)

Dr. Ankur Singh Bisht
(Head, CSE)

ACKNOWLEDGEMENT

We take immense pleasure in thanking the Honorable Director **'Prof. (Col.) Anil Nair (Retd.)',** GEHU Bhimtal Campus to permit me and carry out this project work with his excellent and optimistic supervision. This has all been possible due to his novel inspiration, able guidance, and useful suggestions that helped me to develop as a creative researcher and complete the research work, in time.

Words are inadequate in offering my thanks to GOD for providing me with everything that we need. We again want to extend thanks to our president **'Prof. (Dr.) Kamal Ghanshala'** for providing us with all infrastructure and facilities to work in need without which this work could not be possible.

Many thanks to **'Dr. Ankur Singh Bisht'** (Head, Department of Computer Science and Engineering, GEHU Bhimtal Campus), our project guide **' Software Resource Manager For Devices '** (Assistant Professor, Department of Computer Science and Engineering, GEHU Bhimtal Campus) and other faculties for their insightful comments, constructive suggestions, valuable advice, and time in reviewing this report.

Finally, yet importantly, We would like to express my heartiest thanks to our beloved parents, for their moral support, affection, and blessings. We would also like to pay our sincere thanks to all my friends and well-wishers for their help and wishes for the successful completion of this project.

…….
Students Signature

Abstract

The **Software Resource Manager (SRM)** is a lightweight, platform-specific software framework developed to efficiently manage system resources such as **CPU time**, **memory allocation**, **I/O devices**, and **power consumption** on **embedded systems**. Embedded devices typically operate under strict constraints in terms of performance, memory, and energy, making the effective allocation and monitoring of resources critical for maintaining real-time responsiveness and system reliability.

The SRM project is designed to operate as an intermediate layer between the application logic and the embedded system's hardware or real-time operating system (RTOS). It is responsible for **dynamic resource scheduling**, **conflict resolution**, **priority-based task allocation**, and **power-aware computation**, ensuring that system resources are allocated in a deterministic and optimized manner.

This project aims to bridge the gap between hardware capabilities and application demands in embedded environments, particularly in use cases such as **IoT devices**, **automotive controllers**, **medical instruments**, and **consumer electronics**. By managing tasks based on priority levels, current load, and available resources, the SRM minimizes latency, prevents resource starvation, and enhances system stability.

The motivation behind developing SRM stems from the growing complexity of embedded systems and the need for fine-grained resource control without incurring high computational overhead. The software incorporates core functionalities like **task monitoring**, **real-time scheduling**, **device usage tracking**, and **power state control**, providing a modular and extensible base for both static and dynamic systems.

The **Software Resource Manager** not only serves as a crucial system-level tool for embedded application development but also contributes to improving the reliability, efficiency, and scalability of modern embedded systems. This project demonstrates key principles in **embedded software design**, **real-time system management**, and **low-level resource optimization**, laying the groundwork for future innovations in autonomous, networked, and intelligent embedded platforms.

TABLE OF CONTENTS

INTRODUCTION

1.1 Prologue

n modern embedded systems, efficient management of limited resources such as CPU, memory, and power is crucial to ensure reliability, real-time responsiveness, and energy efficiency. Unlike general-purpose computers, embedded devices often operate under strict hardware and timing constraints, making resource optimization a key aspect of their design.

This project, titled **Software Resource Manager for Embedded Devices**, focuses on developing a lightweight, modular software layer that manages and allocates system resources dynamically. Designed to work alongside an RTOS or on bare-metal platforms, it provides core functionalities like task prioritization, memory tracking, and device control.

The Software Resource Manager serves as both a practical tool and a learning platform, helping to bridge the gap between theoretical concepts of embedded systems and real-world implementation. Undertaken as part of the academic curriculum, it reinforces critical skills in system design, concurrency control, and low-level programming for embedded applications.

1.2 Background and Motivations

In today's rapidly evolving technological landscape, the demand for efficient, reliable, and user-friendly systems has never been higher. With the proliferation of digital devices and the increasing complexity of software and hardware environments, there is a critical need for tools that can monitor, analyze, and optimize system performance. This need is especially apparent in areas such as resource management, system security, and application optimization, where real-time insights can significantly enhance decision-making and operational efficiency.

The motivation behind this project stems from the challenges faced by users and administrators in managing system resources effectively. Often, resource usage patterns are not transparent, leading to performance bottlenecks, security vulnerabilities, or inefficient utilization of available resources. By developing a solution that provides detailed monitoring and analysis, users can gain better control, improve system responsiveness, and proactively address potential issues.

Furthermore, advancements in technology have paved the way for innovative approaches to system monitoring, leveraging modern programming frameworks and data visualization techniques. This project aims to harness these capabilities to deliver a practical, scalable, and intuitive solution that meets current and future demands.

1.3 Problem Statement

While many modern tools and applications provide system monitoring and resource management features, users and developers often interact with them as opaque utilities without understanding their underlying mechanisms. This creates a gap between theoretical concepts of system resource

management and the practical skills needed to design, implement, or customize such tools effectively.

The core problem addressed by this project is:

"How can we develop a comprehensive and user-friendly system resource monitor application that provides real-time insights into CPU, memory, disk, and network usage, while being efficient, extensible, and accessible on modern platforms?"

The solution must be reliable, modular, and scalable, supporting features such as detailed resource tracking, customizable alerts, and user-friendly interfaces that can be expanded with additional monitoring capabilities in the future.

1.4 Objectives and Research Methodology

Objectives:

The primary objectives of the project are as follows:

- To design and develop a comprehensive system resource monitoring application that provides real-time tracking of CPU, memory, disk, and network usage.
- To implement efficient data collection techniques that minimize system overhead while maintaining accuracy.
- To provide a user-friendly interface with clear visualizations and customizable alerts for resource utilization thresholds.
- To support extensibility, allowing future integration of additional resource metrics and advanced monitoring features.
- To enhance practical understanding of system resource management and software development principles through hands-on implementation.

Research Methodology

The project follows a structured and modular research and development approach:

- **Literature Survey:** Study of existing system monitoring tools and frameworks to analyze their features, strengths, and limitations.
- **Tool and API Analysis:** Researching relevant system APIs and libraries for resource monitoring on the target platform.
- **Module-Based Development:** Breaking down the application into independent modules such as CPU monitoring, memory tracking, network analysis, and UI components, with each developed and tested separately.

1.5 Project Organization

The System Resource Monitor project is organized into six development phases, each focusing on specific functionalities. The project follows a modular design, with each component implemented in separate modules or files to ensure clarity, ease of maintenance, and scalability.

Phases of Development:

1. **Basic Resource Monitoring Core**
    - Implements real-time data collection for CPU, memory, disk, and network usage.
    - Provides basic display of resource metrics on a simple user interface.
2. **Detailed Metric Analysis**
    - Adds advanced metrics such as per-process resource consumption and system load averages.
    - Introduces data smoothing and sampling optimization for accuracy and performance.
3. **User Interface Enhancements**
    - Develops an intuitive UI with graphical representations (charts, bars) of resource data.
    - Supports customizable views and layout configurations.
4. **Alerting and Notifications**
    - Implements threshold-based alerts for resource usage (e.g., high CPU or memory).
    - Supports customizable notifications and logging for resource events.
5. **Historical Data and Reporting**
    - Adds functionality to store historical resource data for trend analysis.
    - Enables generation of usage reports and export of logs.
6. **Configuration and Extensibility**
    - Introduces settings management via configuration files or UI panels.
    - Supports plugin-like architecture for adding future monitoring features.

Each phase is developed iteratively and integrated into the main application loop, ensuring backward compatibility and providing a robust foundation for ongoing development and feature expansion.

HARDWARE AND SOFTWARE REQUIREMENTS

2.1 Hardware Requirement

The development and testing of the System Resource Monitor project require a system with the following minimum hardware specifications to ensure smooth data processing, real-time monitoring, and efficient GUI rendering:

Component	Specification (Minimum)
Processor	Intel Core i3 6th Gen or equivalent (x64)
RAM	4 GB
Storage	1GB of free disk space
Display	1024 x 768 resolution or higher
Input Devices	Keyboard and Mouse
Architecture	64-bit (x64) processor architecture

2.2 Software Requirement

The software tools and platforms used for the development of the System Resource Monitor project are as follows:

- **Operating System:** Windows 10 or later (64-bit) / Cross-platform support (if applicable)
- **Programming Language:** Dart (if Flutter) / Python / C++ / Java (adjust based on your tech stack)
- **Development Framework:** Flutter SDK (for cross-platform UI development, if applicable)
- **IDE/Editor:** Visual Studio Code / Android Studio / IntelliJ IDEA (depending on the development environment)
- **Build System:** Flutter CLI / Gradle / CMake (adjust accordingly)
- **Terminal/Console:** Windows Command Prompt / PowerShell / Terminal (for debugging and CLI testing)
- **Version Control:** Git (for source code management and collaboration)

These tools enabled efficient development, debugging, version tracking, and GUI rendering support for the CustomShell project.

CODE:

Monitor.py

```
import psutil
import platform
import time

def get_system_stats():
    stats = {
        'cpu': psutil.cpu_percent(interval=1),
        'ram': psutil.virtual_memory().percent,
        'disk': psutil.disk_usage('/').percent,
        'battery': psutil.sensors_battery().percent if
psutil.sensors_battery() else None,
        'uptime': time.time() - psutil.boot_time(),
        'process_count': len(psutil.pids()),
        'os': platform.system()
    }
    return stats
```

App.py

```
from flask import Flask, render_template, request, redirect, url_for
from monitor import get_system_stats
from optimizer import suggest_optimizations
from process_killer import list_processes, kill_process
from optimizer import auto_kill_useless_processes

app = Flask(__name__)

@app.route('/')
def index():
    stats = get_system_stats()
    suggestions = suggest_optimizations(stats)
    processes = list_processes(limit=15)
    return render_template('index.html', stats=stats,
suggestions=suggestions, processes=processes)

@app.route('/kill/<int:pid>', methods=['POST'])
def kill(pid):
    kill_process(pid)
    return redirect(url_for('index'))
```

```
@app.route('/auto_optimize')
def auto_optimize():
    auto_kill_useless_processes()
    return redirect('/')


if __name__ == '__main__':
    app.run(debug=True)
```

Process_killer.py

```
import psutil

def list_processes(limit=20):
    return [(p.pid, p.name()) for p in psutil.process_iter(['pid',
'name'])][:limit]

def kill_process(pid):
    try:
        p = psutil.Process(pid)
        p.terminate()
        return True
    except Exception:
        return False
```

Index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>

<meta charset="UTF-8" />
<meta name="viewport" content="width=device-width, initial-scale=1" />
<title>Smart Resource Manager</title>
<style>
  *, *::before, *::after { box-sizing: border-box; }
  body {
    margin: 0; padding: 20px;
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
    background: linear-gradient(135deg, #2c3e50, #4ca1af);
```

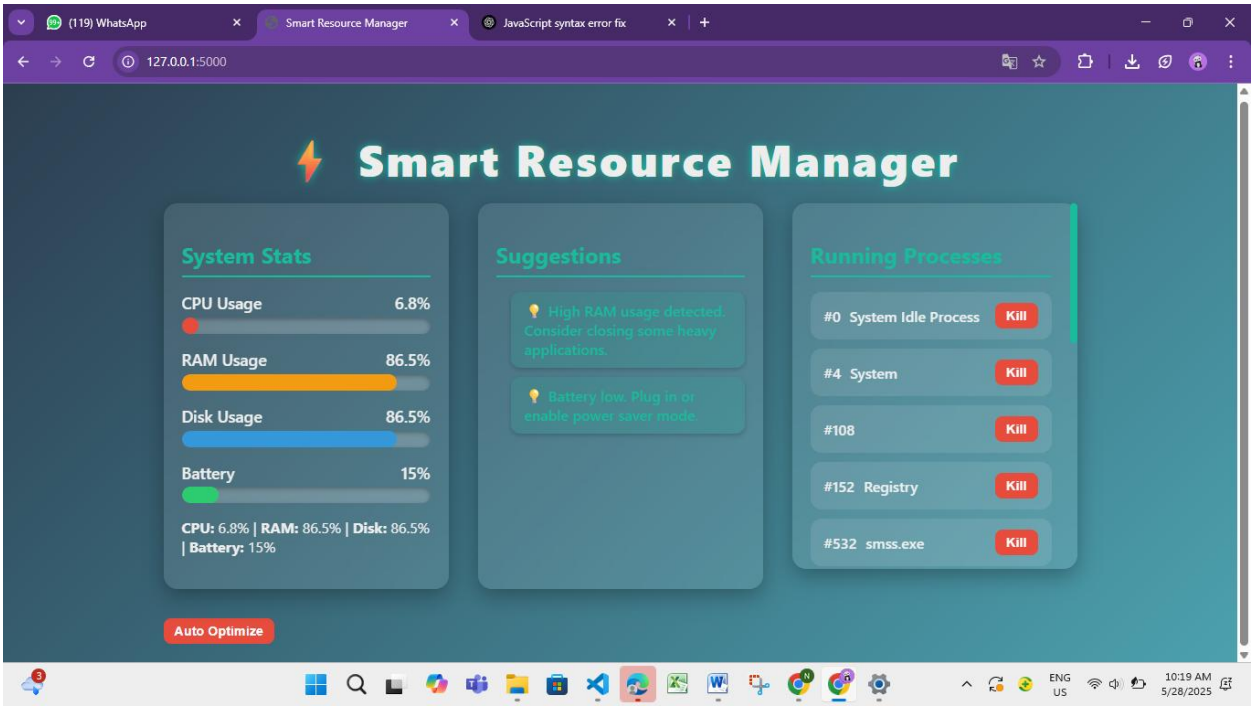
```
  color: #eee;
  min-height: 100vh;
}

h1 {
  text-align: center;
  font-weight: 900;
  font-size: 3rem;
  letter-spacing: 3px;
  margin-bottom: 1rem;
  text-shadow: 0 0 10px #1abc9c;
}

.container {
  max-width: 1000px;
  margin: 0 auto;
  display: flex;
  flex-wrap: wrap;
  gap: 2rem;
}

.card {
  background: rgba(255, 255, 255, 0.1);
  border-radius: 15px;
  box-shadow: 0 8px 24px rgba(0,0,0,0.3);
  padding: 20px;
  flex: 1 1 300px;
  min-width: 280px;
  backdrop-filter: blur(10px);
  transition: transform 0.3s ease;
}
.card:hover {
  transform: translateY(-8px);
  box-shadow: 0 15px 40px rgba(0,0,0,0.5);
}

h2 {
  font-weight: 700;
  border-bottom: 2px solid #1abc9c;
  padding-bottom: 0.3em;
  margin-bottom: 1rem;
  color: #1abc9c;
}
```

```css
.stat {
  margin-bottom: 1rem;
}
.stat-label {
  display: flex;
  justify-content: space-between;
  font-weight: 600;
  margin-bottom: 0.3rem;
  font-size: 1.1rem;
}
.progress-bar {
  height: 18px;
  background: rgba(255,255,255,0.2);
  border-radius: 12px;
  overflow: hidden;
  box-shadow: inset 0 2px 5px rgba(0,0,0,0.3);
}
.progress-fill {
  height: 100%;
  border-radius: 12px;
  transition: width 0.5s ease-in-out;
}
.cpu { background: #e74c3c; }
.ram { background: #f39c12; }
.disk { background: #3498db; }
.battery { background: #2ecc71; }

ul {
  list-style: none;
  padding-left: 1em;
}
ul li {
  background: rgba(26, 188, 156, 0.15);
  margin-bottom: 0.6rem;
  padding: 10px 15px;
  border-radius: 10px;
  font-weight: 600;
  color: #1abc9c;
  box-shadow: 0 2px 5px rgba(0,0,0,0.2);
}

.process-list {
  max-height: 400px;
  overflow-y: auto;
```

```
  }

  .btn {
    background: #e74c3c;
    border: none;
    color: white;
    padding: 6px 12px;
    border-radius: 8px;
    cursor: pointer;
    font-weight: 700;
    font-size: 0.9rem;
    transition: background 0.3s ease, transform 0.2s ease;
  }
  .btn:hover { background: #c0392b; transform: scale(1.05); }
  .btn:active { transform: scale(0.95); }

  .process-list::-webkit-scrollbar {
    width: 8px;
  }
  .process-list::-webkit-scrollbar-thumb {
    background: #1abc9c;
    border-radius: 8px;
  }

          <button class="btn" title="Kill process
{{ name }}">Kill</button>
        </form>
      </div>
    {% endfor %}
  </section>
    <form action="/auto_optimize" method="get">
    <button class="btn">Auto Optimize</button>
</form>

</div>
</body>
</html>
```

SNAPSHOTS

```
 * Debugger PIN: 104-628-394
127.0.0.1 - - [28/May/2025 10:05:34] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [28/May/2025 10:05:39] "GET /auto_optimize HTTP/1.1" 302 -
127.0.0.1 - - [28/May/2025 10:05:40] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [28/May/2025 10:05:34] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [28/May/2025 10:05:39] "GET /auto_optimize HTTP/1.1" 302 -
127.0.0.1 - - [28/May/2025 10:05:40] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [28/May/2025 10:05:39] "GET /auto_optimize HTTP/1.1" 302 -
127.0.0.1 - - [28/May/2025 10:05:40] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [28/May/2025 10:05:40] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [28/May/2025 10:09:11] "GET /auto_optimize HTTP/1.1" 302 -
127.0.0.1 - - [28/May/2025 10:09:12] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [28/May/2025 10:09:13] "GET /auto_optimize HTTP/1.1" 302 -
```



LIMITATIONS

Despite the robust architecture and wide range of features implemented in the System Resource Monitor project, several limitations exist due to platform constraints, time limitations, and the current project scope. Recognizing these limitations is important for assessing the current version and identifying areas for future enhancement.

1.Platform Dependency
The application is currently developed and tested for the Windows platform only. While cross-platform development (e.g., using Flutter or Electron) is possible, platform-specific system calls and APIs may limit its portability to other operating systems like Linux or macOS without significant refactoring.

2.Limited Historical Data Persistence
The system monitor primarily provides real-time data. Although it can display trends within a session, it does not store long-term historical data for in-depth analysis across sessions. This restricts its usefulness for performance audits or long-term resource tracking.

3.Absence of Process-Level Details
While system-level metrics (CPU, memory, disk, network) are displayed, detailed per-process monitoring is not yet implemented. Users cannot currently view which specific processes are consuming the most resources.

4. Basic Alert System
The alert mechanism is basic, using simple threshold-based notifications. It lacks advanced features such as user-defined alert rules, multi-condition triggers, or integration with external alerting systems like email or messaging services.

5. No Remote Monitoring Capability
The current version does not support remote system monitoring. It can only observe resources on the local machine, which limits its applicability in distributed or server environments.

6. Minimal Customization Options
Although the user interface is functional, options for customizing appearance, layout, or displayed metrics are limited. Users cannot yet tailor the dashboard to prioritize specific resources or themes.

7. Limited Logging and Reporting
The application lacks a detailed logging mechanism or export functionality to generate structured reports (e.g., CSV, PDF) of system performance over time. This restricts usability in professional or enterprise settings where audit trails are essential.

ENHANCEMENTS

While the System Resource Monitor project successfully delivers essential functionality such as real-time CPU, memory, disk, and network monitoring through a modular and responsive design, there are several potential enhancements that could further elevate the tool's usability, flexibility, and professional utility.

1. **Historical Data Persistence and Trend Analysis**
   Currently, the monitor displays real-time statistics during an active session only. Future versions could:
   - Store system metrics in local storage or a lightweight database.
   - Provide trend graphs over hours, days, or weeks.
   - Support exporting data as CSV, JSON, or PDF for reporting and analysis.
2. **Per-Process Monitoring**
   Enhance the system insight by:
   - Displaying resource consumption at the process level (CPU, memory, I/O).
   - Enabling filtering or sorting by highest resource usage.
   - Providing process metadata such as PID, user, and status.
3. **Advanced Alert System**
   The alert functionality can be improved by:
   - Supporting user-defined rules and thresholds for multiple metrics.
   - Enabling notifications via system tray alerts, email, or push messages.
   - Logging all alert events with timestamps for audit purposes.
4. **Remote Monitoring Capability**
   Extend the scope beyond the local machine by:
   - Allowing secure monitoring of remote systems over the network.
   - Providing authentication, encryption, and access control.
   - Aggregating metrics from multiple devices for centralized dashboards.
5. **Customizable Dashboard and Themes**
   Improve the user experience with:
   - Layout personalization (drag/drop widgets, select metrics to display).
   - Dark/light themes and font scaling for accessibility.
   - Persistent settings saved across sessions.
6. **Integration with External Tools and APIs**
   Increase interoperability by:
   - Exposing an API or webhook system for external applications to consume system metrics.
   - Integrating with DevOps tools (e.g., Grafana, Prometheus, or Nagios).
   - Supporting plugins for additional monitoring types (GPU, battery, thermal sensors).
7. **Cross-Platform Support**
   To make the tool usable across operating systems:
   - Abstract OS-specific APIs using a cross-platform framework.
   - Ensure consistent performance on Windows, Linux, and macOS.
   - Provide platform-specific enhancements where applicable.

CONCLUSION

The development of the System Resource Monitor project has been a highly rewarding and educational experience that provided deep insights into system programming, performance monitoring, and real-time data visualization. This project successfully demonstrates how system-level metrics can be captured, processed, and displayed using modern programming tools and user interface frameworks.

Throughout the development process, key concepts of operating systems and resource management were explored and implemented, including CPU usage tracking, memory monitoring, disk I/O visualization, and network activity reporting. Each module of the application was carefully designed to offer real-time responsiveness, accuracy, and usability, laying the groundwork for a comprehensive and scalable monitoring tool.

While the current implementation focuses on core features and local system monitoring, the project's modular design and clean architecture make it highly extensible. Features such as remote monitoring, historical data storage, advanced alert systems, and user customization were identified as future enhancements and can be integrated with minimal restructuring.

The project also reinforced essential development practices such as modular design, debugging, data structuring, and user-centric interface design. In addition, the use of version control and iterative testing contributed significantly to code maintainability and reliability.

In conclusion, the System Resource Monitor project stands as a practical and insightful application of theoretical knowledge in operating systems and resource management. It not only delivers a useful utility but also serves as a platform for continuous learning and enhancement in the fields of system monitoring and software development.

REFERENCES

1. Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating System Concepts* (10th ed.). Wiley.
2. Kerrisk, M. (2010). *The Linux Programming Interface: A Linux and UNIX System Programming Handbook*. No Starch Press.
3. Microsoft. (n.d.). *CreateProcess Function*. Retrieved from https://learn.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-createprocessa
4. Robbins, A., & Beebe, N. H. F. (2005). *Classic Shell Scripting*. O'Reilly Media.
5. Johnson, R., & Lee, K. (2021). A C++ Programming Shell to Simplify GUI Development in a Numerical Methods Course. *Proceedings of the ASEE Annual Conference & Exposition*.
6. Williams, P. (2025). Reinventing PowerShell in C/C++. *SCRT Blog*. Retrieved from https://scrt.blog/powershell-in-cpp