A SYNOPSIS ON

CUSTOM SHELL

Submitted in partial fulfilment of the requirement for the award of the degree of

BACHELOR OF TECHNOLOGY

In

Computer Science & Engineering

Submitted by:

Deepak Birkhani	**2261165**
Dinesh Joshi	**2261189**
Esha Mahara	**2261204**
Harshita Mehta	**2261259**

Under the Guidance of

Mr. Prince Kumar

Assistant Professor

Project Team ID: 29

Department of Computer Science & Engineering

Graphic Era Hill University, Bhimtal, Uttarakhand

March-2025

Graphic Era Hill University
BHIMTAL CAMPUS

CANDIDATE'S DECLARATION

We hereby certify that the work which is being presented in the Synopsis entitled **"Custom Shell"** in partial fulfilment of the requirements for the award of the Degree of Bachelor of Technology in Computer Science & Engineering of the **Graphic Era Hill University, Bhimtal Campus** and shall be carried out by the undersigned under the supervision of **Mr. Prince Kumar, Assistant Professor**, Department of Computer Science & Engineering, Graphic Era Hill University, Bhimtal.

Deepak Birkhani	**2261165**
Dinesh Joshi	**2261189**
Esha Mahara	**2261204**
Harshita Mehta	**2261259**

The above mentioned students shall be working under the supervision of the undersigned on the **"Custom Shell"**

Signature

Supervisor

Signature

Head of the Department

Internal Evaluation (By DPRC Committee)

Status of the Synopsis: Accepted / Rejected

Any Comments:

Name of the Committee Members: **Signature with Date**

1.

2.

Table of Contents

Chapter 1

Introduction

A shell is a command-line interface that allows users to communicate with the operating system by executing commands. It acts as a bridge between the user and system functionalities. Traditional shells like Bash (on Linux) and Command Prompt/PowerShell (on Windows) provide extensive functionality but often come with unnecessary complexities for specific user needs.

Shells are integral components of modern operating systems, enabling users to interact with the system efficiently through text-based commands. Unlike graphical user interfaces (GUIs), which rely on icons and visual elements, shells provide a lightweight and flexible means of performing operations such as file manipulation, process control, and system configuration. They are particularly useful for developers, system administrators, and power users who require precise control over their systems.

This project focuses on building a custom shell in C++ that offers a simplified and userfriendly interface while maintaining essential shell functionalities. The shell will execute system commands, manage files and directories, launch applications, and process user input efficiently.

Existing system shells may be overwhelming for beginners due to their complex syntax and command structure. Some of the main issues include:

- Overhead of unnecessary features: Default shells include numerous commands that are rarely used by average users, making navigation confusing.
- Limited customization: Traditional shells have predefined behaviors, making it difficult to modify or adapt without scripting knowledge.
- Performance concerns: Running a full-fledged shell in specific embedded or lightweight environments can be resource-intensive, leading to inefficiencies.
- Lack of intuitive application launching: Many users struggle to execute programs and manage files efficiently using CLI-based systems, as they require specific syntax and commands.
- Absence of user-friendly feedback: Most shells return minimal feedback, making error handling difficult for non-technical users.

To address these challenges, this project will develop a lightweight, user-friendly, and efficient custom shell tailored to essential user needs while eliminating unnecessary complexities. The goal is to provide a seamless experience for executing commands, launching applications, and managing files, ensuring accessibility for users at all levels of technical expertise.

Chapter 2

Background / Literature Survey

Existing Shells and Their Functionality

Several shells have been developed over time, each with unique features:

- Bash (Bourne Again Shell) – The most commonly used shell in Linux, featuring scripting capabilities, process management, and automation tools. Bash is widely used due to its robust scripting abilities and extensive community support.
- PowerShell – A Microsoft-developed shell with object-oriented scripting for administrative tasks. Unlike traditional shells, PowerShell is designed to handle objects rather than text-based input/output, making it ideal for system administration.
- Zsh (Z Shell) – An extended Unix shell with advanced auto-completion, enhanced scripting capabilities, and plugin support. It is popular among developers due to its customization potential.
- Fish Shell – Designed for ease of use with a focus on user-friendliness and modern scripting. Fish includes syntax highlighting, autosuggestions, and a well-structured command syntax.

Research on Custom Shell Development

Research into custom shells shows that they can be optimized for specific use cases:

- Lightweight Shells: Custom-built shells focus on minimalism, reducing memory consumption and execution time.
- User-Friendly Interfaces: Simplified command structures improve accessibility for beginners. Features such as auto-suggestions, command history, and error handling make shells more user-friendly.
- Application-Oriented Shells: Some custom shells are designed for specific environments, such as embedded systems or cybersecurity applications, where lightweight and efficient performance is critical.

Our custom shell will integrate the most relevant features from traditional shells while eliminating unnecessary complexities. By focusing on core functionalities such as command execution, directory navigation, and application launching, the shell will cater to both novice and experienced users who seek efficiency and simplicity.

Chapter 3

Objectives

The primary objective of this project is to design and implement a custom shell in C++ with the following features:

1. Basic Command Execution: Support for essential system commands like cd, dir, mkdir, del, and copy. These commands allow users to navigate and manipulate directories and files efficiently.
2. Application Launching: Users can open applications such as Notepad, Calculator, or File Explorer directly from the shell.
3. Custom Commands: Implement an open command to simplify launching applications and directories without requiring specific paths or syntax.
4. Error Handling: Provide meaningful error messages for invalid commands, guiding users toward correct usage.
5. Cross-Platform Capability: Focus primarily on Windows while maintaining extensibility for Linux in future iterations.
6. Interactive User Interface: A prompt that dynamically accepts and processes user input, enhancing user experience.
7. Process Management: Support for background processes, command chaining, and multiple command execution in a single line.

By focusing on these objectives, the project aims to create a powerful yet intuitive shell that enhances user experience by simplifying common tasks and automating repetitive actions where necessary.

Chapter 4

Hardware and Software Requirements

Hardware Requirements

- Processor: Minimum Intel Core i3 or equivalent for optimal execution speed.
- RAM: 4GB (Recommended: 8GB for optimal performance when handling multiple processes).
- Storage: At least 500MB free disk space to store compiled binaries, logs, and shell history.
- Operating System: Windows 10/11, with future support for Linux planned.

Software Requirements

- Programming Language: C++, chosen for its performance and low-level system interaction capabilities.
- Compiler: MinGW (g++) or Microsoft Visual Studio Compiler for compiling the C++ source code.
- Libraries Used:
    - <windows.h> (for system calls and process execution)
    - <iostream> (for input and output operations)
    - <sstream> (for parsing and tokenizing commands)
    - <vector> (for managing command arguments)
    - <unistd.h> (for process handling in Unix-like systems)

The project requires a development environment such as Visual Studio Code or Code::Blocks, along with debugging tools to identify and resolve issues during implementation.

Chapter 5

Project Approach

Windows APIs & Libraries

- Process Management: CreateProcess(), TerminateProcess(), GetExitCodeProcess()
- File Management: CreateFile(), ReadFile(), WriteFile()
- Command Execution: system(), ShellExecute()
- I/O Handling: SetConsoleMode(), GetStdHandle()
- Threading & Synchronization: CreateThread(), WaitForSingleObject()

Command Execution

- Read user input (cin >> command).
- Parse the command and check if it's a built-in command.
- If it's an external command, use CreateProcess() to execute it.

Custom Built-in Commands

Command	Functionality	Implementation
mycd	Change directory	SetCurrentDirectory()
myls	List files in a folder	FindFirstFile(), FindNextFile()
mypwd	Print current directory	GetCurrentDirectory()
myexit	Exit shell	exit(0)
myclear	Clear screen	system("cls")

Listing Files in a Directory

Use Windows API (FindFirstFile() and FindNextFile()):

Command History

Use Windows Console Input API (GetStdHandle()):

- Store past commands in a vector.
- Allow the user to navigate history using arrow keys.

Process Management

Feature	**Windows API**
Launch Process	CreateProcess()
Terminate Process	TerminateProcess()
Get Process List	EnumProcesses()
Wait for Process Completion	WaitForSingleObject()

Architecture & Execution Flow

1. Display the shell prompt (MyShell>).
2. Read user input and parse command.
3. Check if it's a built-in command, if yes, execute it.
4. If it's an external command, use CreateProcess() to run it.
5. If the command involves redirection (>, <) or piping (|), handle it using CreatePipe().
6. If it’s a background process (&), run it without waiting.
7. Store command in history.
8. Repeat.

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 Documentation: CreateProcess Function
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. ASEE
6. Williams, P. (2025). Reinventing PowerShell in C/C++. SCRT Blog