The University of Haripur(KPK)

Project Title: Mini Bash Shell Implementation Using C++

Project carried out by: Hammad Younis Abbasi

Under the Supervision of: Mam Ayesha Riaz

Lecturer, Department of Information Technology

Course: Operating Systems
Programming Language
C++ (POSIX-based Linux/Ubuntu environment)
Project Type
Command Line Interface (CLI) System Project

1. Introduction

The **Mini Bash Shell** project is a simplified implementation of a Unix/Linux shell developed using the C++ programming language. This project is designed to demonstrate core **Operating System concepts**, including **process creation**, **process control**, **system calls**, **command execution**, **directory management**, **I/O redirection**, and **command history handling**.

The shell mimics the behavior of the original **Bash shell** in Ubuntu by allowing users to execute common system commands (such as `ls`, `pwd`, `cat`, etc.), manage directories, redirect output to files, and maintain a history of executed commands. The project also introduces the fundamentals behind tools like **Bash** and **Nano editor interaction** at a conceptual level.

---

2. How to Run C++ on Ubuntu Terminal

- **Using the Terminal (Command Line)**

This method is standard across most Linux distributions and is excellent for learning the compilation process.

- **Prerequisites**

First, ensure you have the necessary compiler tools installed. Open your terminal (press Ctrl + Alt + T) and run the following command to install the build-essential package, which includes the G++ compiler and required libraries:

```
sudo apt update && sudo apt install build-essential
```

Verify the installation by checking the G++ version:

```
g++ --version
```

- **Steps to Compile and Run**

Write your code: Use a text editor (like nano or gedit) to create your C++ source file. Save it with a .cpp extension (e.g., hello.cpp).

nano hello.cpp

Inside the editor, add a simple "Hello, World!" program:

```
#include <iostream>
using namespace std;
int main() {
   cout << "Hello, World!" << endl;
   return 0;
}
```

Save the file (Ctrl+O, then Enter) and exit the editor (Ctrl+X).

Navigate to the file directory: Use the cd command to change your current directory to where you saved the file.

cd /path/to/your/code/directory

Compile the code: Use the g++ command to compile your source file into an executable program. The -o flag allows you to specify the output name for the executable (e.g., output_program):

g++ hello.cpp -o output_program

If you don't use the -o flag, the executable will be named a.out by default.

Run the executable: Execute the compiled program by typing ./ followed by the name you assigned to your executable file:

./output_program

3. Nano Editor (Conceptual) Explanation

3.1 What is Nano Editor?

Nano is a simple, user-friendly text editor available in most Linux distributions. It runs inside the terminal and allows users to create and edit text files interactively.

Although this Mini Bash Shell does not implement a full Nano editor, it **supports launching Nano** as an external program, which demonstrates how shells interact with text editors.

3.2 How Mini Bash Interacts with Nano

When the user types:

```
nano file.txt
```

The following happens internally:

1. Mini Bash parses the command
2. A child process is created using `fork()`
3. `execvp()` replaces the child process with Nano editor
4. Nano takes full control of the terminal
5. Upon exit, control returns to Mini Bash shell

This behavior is identical to the original Bash shell.

3.3 Educational Importance of Nano Support

- Demonstrates **process replacement**
- Shows **terminal control transfer**
- Reinforces understanding of how shells launch editors
- Bridges theory with real-world Linux usage

---

4. Source Code:

```
#include<iostream>
#include<string>
#include<vector>
#include<sstream>
#include<unistd.h> //for fork(),execvp(),chdir(),getcwd()
#include<wait.h> // for waitpid()
#include<fcntl.h> // for open(),O_WRONLY,O_CREAT,O_TRUNC
#include<cstdlib> //for exit(),getenv()
#include<pwd.h>
using namespace std;
// function to execute user command
vector <string> parseLine(const string& line){
vector<string> tokens;
stringstream ss(line);
string word;
while (ss>>word){
tokens.push_back(word);
}
return tokens;
}

// function to execute user command
void executeCommand(vector<string>& tokens, vector<string>& history){
if (tokens.empty()) return;
string command=tokens[0];

//----Built-in commands---
// Exit command
if(command == "exit" || command =="shutdown" || command =="poweroff"){
cout<<"Exiting Mini Bash. Have a nice day!" <<endl;
exit(0);
}
//change directory
if(command == "cd"){
```

```
string path;
if(tokens.size()<2){
char* home  = getenv("HOME");
if(home != nullptr){
path = home;
}
else{
struct passwd* pw = getpwuid(getuid());
path = pw->pw_dir;
}}
else if(tokens[1] == "~"){
char* home = getenv("HOME");
if (home != nullptr)
path=home;
}else{
path = tokens[1];
}
if(chdir(path.c_str()) !=0){
perror("cd failed");
}
return;
}

// Show history

if (command == "history"){
cout<<"\n-----Command history ------\n";

for (size_t i = 0; i < history.size(); i++) {
cout << i + 1 << " " << history[i] << endl;
}

cout<<"---------------------------\n";
return;
}

string outputFile = "";
int redirectPos = -1;

for (size_t i = 0; i < tokens.size(); i++) {
if (tokens[i] == ">") {
redirectPos = i;
if (i + 1 < tokens.size()) {
outputFile = tokens[i + 1];
}
```

```cpp
break;
}
}

//Fork a new process
pid_t  pid = fork();
if(pid==0){
// --child process--
// handle output redirection
if (redirectPos !=-1 && !outputFile.empty()){
int fd = open(outputFile.c_str(),O_WRONLY | O_CREAT |O_TRUNC,0644);
if (fd <0){
perror("Cannot open file");
exit(1);
}
dup2(fd, STDOUT_FILENO);
close(fd);
tokens.erase(tokens.begin() + redirectPos, tokens.begin() +redirectPos+2);
}

vector<char*> args;
for (auto& t : tokens){
args.push_back(const_cast<char*>(t.c_str()));
}

args.push_back(nullptr);

if (execvp(args[0], args.data()) < 0) {
cerr << "Command not found:" << tokens[0] << endl;
exit(1);
}

} else if (pid > 0) {
int status;
waitpid(pid, &status, 0);
} else {
perror("Fork failed");
}
}

int main() {
string line;
vector<string> history;

cout <<"---- Simple Mini Bash Shell ----\n";
cout <<"Supported: cd, history, exit, > redirection, and system commands (ls,
cat, etc.)\n";

while (true) {
char cwd[1024];
getcwd(cwd, sizeof(cwd));
cout <<"\033[1;32mminibash:" <<cwd <<"$ \033[0m";

if (!getline(cin, line)) break;
if(line.empty()) continue;

history.push_back(line);

vector<string> tokens = parseLine(line);

executeCommand(tokens, history);
}

return 0;

}
```

5. Objectives of the Project

The main objectives of this project are:

- To understand how a Linux shell works internally
- To implement process creation using `fork()`
- To execute system commands using `execvp()`
- To manage parent and child processes using `waitpid()`
- To implement built-in shell commands
- To understand file handling and output redirection
- To apply Operating System concepts in a practical project

---

6. Features of Mini Bash Shell

The Mini Bash Shell supports the following features:

6.1 Built-in Commands

- `cd` – Change directory
- `history` – Display command history

- `exit`, `shutdown`, `poweroff` – Exit the shell

6.2 System Commands

- Supports Ubuntu/Linux system commands such as:
  - `ls`
  - `pwd`
  - `cat`
  - `date`
  - `whoami`

6.3 Output Redirection

- Redirect output to a file using > operator
- `ls > output.txt`

6.4 Command History

- Stores and displays all previously executed commands

6.5 Dynamic Prompt

- Displays current working directory in the shell prompt

---

7. System Requirements

Hardware Requirements

- Any system capable of running Ubuntu/Linux

Software Requirements

- Ubuntu/Linux OS
- GCC Compiler
- POSIX-compliant environment

---

8. Libraries and System Calls Used

Header File	Purpose
`<iostream>`	Input/output operations
`<string>`	String handling

Header File	Purpose
`<vector>`	Dynamic arrays
`<sstream>`	Parsing user input
`<unistd.h>`	fork(), execvp(), chdir(), getcwd()
`<wait.h>`	waitpid()
`<fcntl.h>`	File handling and redirection
`<cstdlib>`	exit(), getenv()

---

9. Program Architecture

The Mini Bash Shell follows a modular and layered architecture inspired by Unix shells.

9.1 High-Level Architecture Diagram (Textual Representation)

```
+----------------------+
|      User Input      |
+----------+-----------+
           |
           v
+----------------------+
|   Command Parser     |
|   (parseLine)        |
+----------+-----------+
           |
           v
+------------------------------+
|   Built-in Command Checker   |
| (cd, history, exit)          |
+----------+-------------------+
           |
           v
+------------------------------+
| Process Creation (fork)      |
+----------+-------------------+
           |
     +-----+------+
     |            |
     v            v
+------------+  +----------------+
| Child      |  | Parent Process |
| execvp()   |  | waitpid()      |
+------------+  +----------------+
           |
           v
+------------------------------+
| Output to Terminal / File    |
+------------------------------+
```

9.2 Control Flow Explanation

1. User enters a command
2. Input is parsed into tokens
3. Shell checks for built-in commands
4. If external command:
    - `fork()` creates child process
    - Child executes command using `execvp()`
    - Parent waits using `waitpid()`
5. Output is displayed or redirected to file

---

10. Detailed Code Explanation

10.1 Command Parsing (`parseLine` Function)

This function splits the user input into individual tokens using a string stream.

Purpose:

- Convert a command line string into a list of arguments

Example:

```
ls -l /home
```

Parsed as:

```
["ls", "-l", "/home"]
```

10.2 Command Execution (`executeCommand` Function)

This function identifies whether a command is built-in or an external system command.

Built-in Command Handling:

- **exit / shutdown / poweroff**
    - Terminates the Mini Bash shell
- **cd**
    - Changes the current directory using `chdir()`
    - Defaults to HOME directory if no path is provided
- **history**
    - Displays all previously executed commands

10.3 Output Redirection

The shell checks for the > symbol and redirects output to a file using:

- `open()`
- `dup2()`
- `close()`

Example:

```
ls > files.txt
```

This stores the output of `ls` into `files.txt`.

10.4 Process Creation and Execution

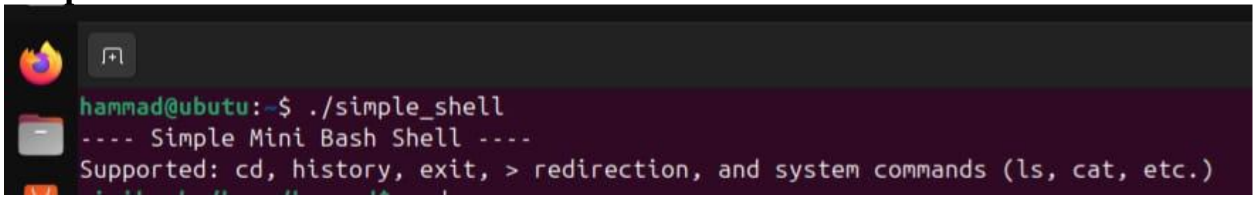
- `fork()` creates a child process
- `execvp()` executes system commands
- `waitpid()` ensures the parent waits for child completion

This follows the standard **Unix process execution model**.

---

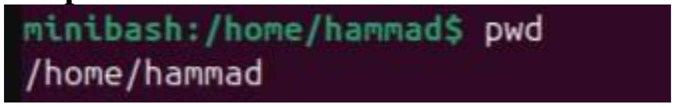
11. Sample Outputs and Execution Examples

- Start the Mini Bash Program
  `$ ./simple_shell`
  Output:



```
hammad@ubutu:~$ ./simple_shell
---- Simple Mini Bash Shell ----
Supported: cd, history, exit, > redirection, and system commands (ls, cat, etc.)
```

- Print Working Directory
  `minibash:/home/hammad$ pwd`
  Output:

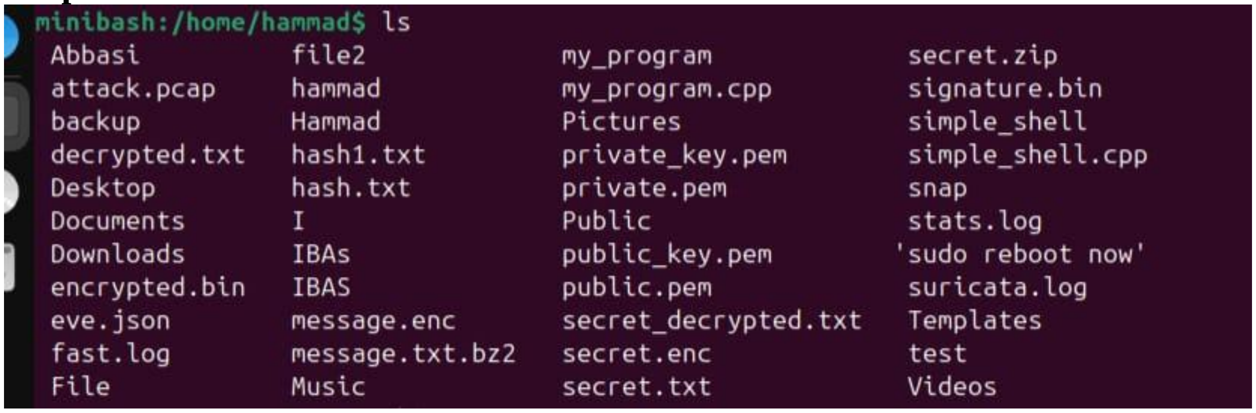


```
minibash:/home/hammad$ pwd
/home/hammad
```

- List Files & Folders

```
minibash:/home/hammad$ ls
```

Output:



```
minibash:/home/hammad$ ls
 Abbasi           file2             my_program             secret.zip
 attack.pcap      hammad            my_program.cpp         signature.bin
 backup           Hammad            Pictures               simple_shell
 decrypted.txt    hash1.txt         private_key.pem        simple_shell.cpp
 Desktop          hash.txt          private.pem            snap
 Documents        I                 Public                 stats.log
 Downloads        IBAs              public_key.pem        'sudo reboot now'
 encrypted.bin    IBAS              public.pem             suricata.log
 eve.json         message.enc       secret_decrypted.txt   Templates
 fast.log         message.txt.bz2   secret.enc             test
 File             Music             secret.txt             Videos
```

- Create Directory

```
minibash:/home/hammad$ mkdir first
```

Output:

```
    (no output)
```

Directory created successfully.

- Change Directory

```
minibash:/home/hammad$ cd first
```

Output (Prompt changes):

```
  minibash:/home/hammad/first$
```

- Create File

```
minibash:/home/hammad/first$ touch a.first
```

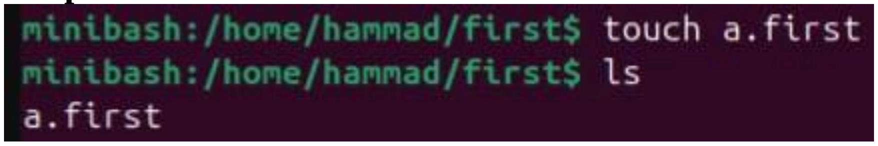
Output:

```
(no output)
```

- Check File

```
minibash:/home/hammad/first$ ls
```

Output:

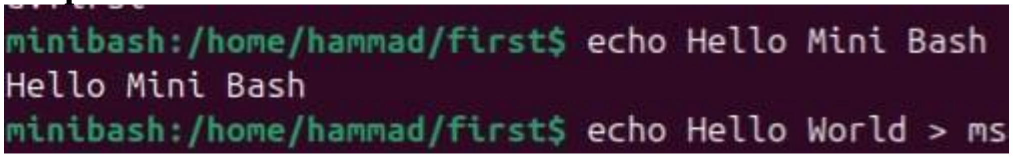


```
minibash:/home/hammad/first$ touch a.first
minibash:/home/hammad/first$ ls
a.first
```

- Display Text

```
minibash:/home/hammad/first$ echo Hello Mini Bash
```

Output:



```
minibash:/home/hammad/first$ echo Hello Mini Bash
Hello Mini Bash
minibash:/home/hammad/first$ echo Hello World > ms
```

- Output Redirection (>)

```
minibash:/home/hammad/first$ echo Hello World > msg.txt
```

Output:

```
    (no output)
```

Output saved to file.

- View File Content

```
minibash:/home/hammad/first$ cat msg.txt
```

Output:

```
Hello World
```

- Go Back to Home

```
minibash:/home/hammad/first$ cd
```

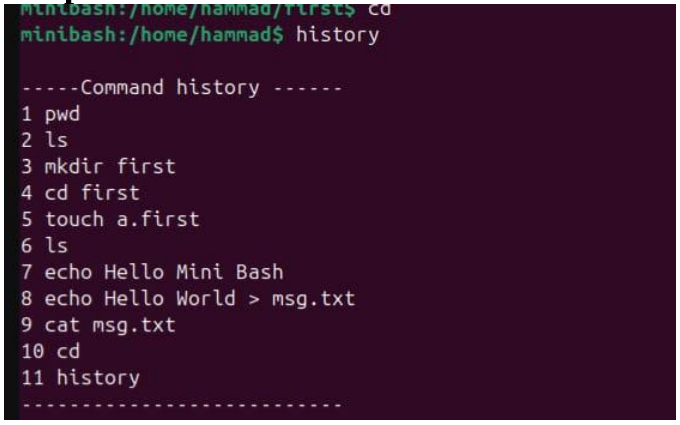
Output:

```
minibash:/home/hammad$
```

- Show All Commands

```
minibash:/home/hammad$ history
```

Output:



```
minibash:/home/hammad/first$ cd
minibash:/home/hammad$ history

-----Command history ------
1 pwd
2 ls
3 mkdir first
4 cd first
5 touch a.first
6 ls
7 echo Hello Mini Bash
8 echo Hello World > msg.txt
9 cat msg.txt
10 cd
11 history
---------------------------
```

- Show Current User

```
minibash:/home/hammad$ whoami
```

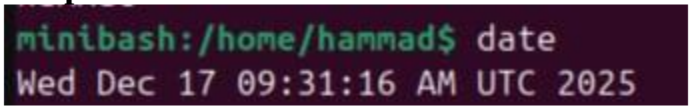
Output:

```
hammad
```

- Show Date & Time

```
minibash:/home/hammad$ date
```

Output:

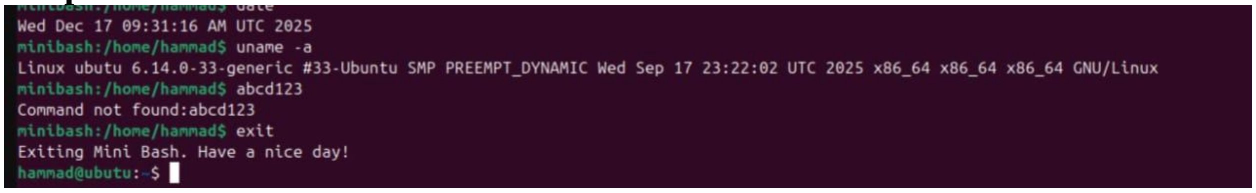


```
minibash:/home/hammad$ date
Wed Dec 17 09:31:16 AM UTC 2025
```

- System Info

```
minibash:/home/hammad$ uname -a
```

Output:



```
minibash:/home/hammad$ date
Wed Dec 17 09:31:16 AM UTC 2025
minibash:/home/hammad$ uname -a
Linux ubutu 6.14.0-33-generic #33-Ubuntu SMP PREEMPT_DYNAMIC Wed Sep 17 23:22:02 UTC 2025 x86_64 x86_64 x86_64 GNU/Linux
minibash:/home/hammad$ abcd123
Command not found:abcd123
minibash:/home/hammad$ exit
Exiting Mini Bash. Have a nice day!
hammad@ubutu:~$
```

- Invalid Command Test

```
minibash:/home/hammad$ abcd123
```

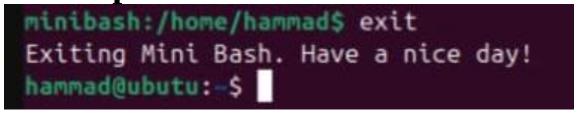
Output:

```
Command not found: abcd123
```

- **Exit Mini Bash**

```
minibash:/home/hammad$ exit
```

Output:



```
minibash:/home/hammad$ exit
Exiting Mini Bash. Have a nice day!
hammad@ubutu:~$
```

---

12. Limitations of the Project

- No support for piping (`|`)
- No background execution (`&`)
- No advanced text editing like full Nano editor
- Limited error handling

---

13. Future Enhancements

- Implement pipe (`|`) functionality
- Add background process support
- Implement command auto-completion
- Add Nano-like text editor support
- Improve error handling and security

---

14. Learning Outcomes

Through this project, the following concepts were learned:

- Linux shell working principles
- Process management in Operating Systems
- System calls and file descriptors
- Command parsing and execution
- Practical use of fork-exec-wait model

---

15. Conclusion

The Mini Bash Shell project successfully demonstrates core Operating System concepts through a practical and interactive implementation. It provides a strong foundation for understanding how Unix-based shells work and serves as an excellent academic project for Operating Systems courses.

This project bridges theoretical OS concepts with real-world system programming using C++.

---

16. References

- Linux Man Pages
- Advanced Programming in the UNIX Environment
- Operating System Concepts – Silberschatz
