A COURSE PROJECT REPORT

on

CUSTOM UNIX SHELL FOR BASIC TERMINAL OPERATIONS

U18IT307 - OPERATING SYSTEMS

KAKATIYA INSTITUTE OF TECHNOLOGY AND SCIENCE

WARANGAL

By

K. Aryan

B23IT117

Under the supervision of

Dr. K. Praveen Kumar
Assistant Professor

DEPARTMENT OF INFORMATION TECHNOLOGY
KAKATIYA INSTITUTE OF TECHNOLOGY & SCIENCE, WARANGAL-15

2024 - 2025

KAKATIYA INSTITUTE OF TECHNOLOGY & SCIENCE
(An Autonomous Institute under Kakatiya University, Warangal)
Opp: Yerragattu Gutta, Hasanparthy (Mandal), WARANGAL - 506 015, Telangana, INDIA.
काकतीय प्रद्योगिकी एवं विज्ञान संस्थान, वरंगल - ५०६ ०१५
కాకతీయ సాంకేతిక విజ్ఞాన శాస్త్ర విద్యాలయం, వరంగల్ - ౫౦౬ ౦౧౫
Estd 1980 KITSW
website: www.kitsw.ac.in | e-mail: principal@kitsw.ac.in | ✆: +91 870 2564888 | Fax: +91 870 2564320

CERTIFICATE

This is to certify that **K. ARYAN (B23IT117)** of III- Semester B.Tech. Information Technology has satisfactorily completed the **Course project** entitled **"CUSTOM UNIX SHELL FOR BASIC TERMINAL OPERATIONS"** in the partial fulfilment of the requirement of B.Tech. degree during this academic year 2024-2025.

SUPERVISOR

Dr. K. Praveen Kumar
Assistant Professor

CHAIRMAN, DSEC

Dr. T Senthil Murugan
Professor & Head,
Dept of IT.

ACKNOWLEDGEMENT

I wish to take this opportunity to express my deep gratitude to all the people who have extended their cooperation in various ways during my Seminar. It is my pleasure to acknowledge the help of all those individuals.

I thank **Dr. K. Ashoka Reddy**, Principal of Kakatiya Institute of Technology & Science, Warangal, for his strong support.

I thank **Dr. T. Senthil Murugan**, Professor & Head, Department of Information Technology for his constant support in bringing shape of this Seminar.

I would like to thank my supervisor, **Dr. K. Praveen Kumar**, Assistant Professor of Information Technology for his guidance and help throughout the Seminar.

In completing this Seminar successfully all our faculty members have given an excellent cooperation by guiding us in every aspect. All your guidance helped me a lot and I am very grateful to you.

K. Aryan

B23IT117

ABSTRACT

The "ghz-sh" project aims to develop a custom UNIX shell that supports basic terminal operations, providing an accessible platform for learning shell programming. Unlike complex shells like Bash, "ghz-sh" focuses on core functionalities, making it easier to understand shell mechanics. Key features include executing basic commands, supporting built-in commands (`chdir`, `leave`, `sos`, `tell`), handling input/output redirection, and implementing basic signal handling for process control.

The shell operates by parsing user commands, utilizing system calls like `fork()` and `exec()` for process management, and handling file descriptors for redirection. Built-in commands modify the shell's internal state, while external commands are managed as separate processes. Signal handling ensures stability, enabling graceful termination of commands with interrupts like `Ctrl+C`.

By offering essential shell functionalities in a simplified environment, "ghz-sh" serves as a valuable educational tool, bridging theoretical knowledge and practical application in UNIX-based systems. The project establishes a foundation for further exploration in shell programming, system automation, and command-line interface development.

Table of Contents

S. No.	Contents	Page Number
1	Introduction	7
2	Problem Definition	10
3	Proposed Method	12
4	Experiment	15
5	Result	25
6	Conclusions	28
7	Poster	
8	References	

List of Figures

S. No.	Name	Page No
5.1	Sos command	26
5.2	Ls and chdir command	26
5.3	Tell command	26
5.4	Tell Arithmetic operations	27
5.5	Ps command	27

Chapter 1
Introduction

In computing, a shell is a command-line interface that allows users to interact with the operating system by executing commands, running scripts, and managing tasks. UNIX-based systems, known for their powerful command-line capabilities, utilize shells extensively for system operations and automation. This course project focuses on the development of a custom UNIX shell named ghz-sh, a simplified shell that performs basic terminal operations. The goal is to create a functional and approachable shell that supports essential UNIX features, providing practical experience in shell programming and system-level concepts.

The ghz-sh shell aims to replicate some fundamental functionalities found in standard UNIX shells like Bash but with a more streamlined and educational approach. Unlike fully-featured shells, ghz-sh focuses on essential tasks, making it easier to understand how shells work from the ground up. The project offers hands-on experience in understanding basic shell components and operations, setting a foundation for further exploration into advanced shell programming.

Key features of ghz-sh include executing basic commands, handling built-in commands, supporting input/output redirection, and enabling command piping. Command execution is implemented using system calls like `fork()` and `exec()`, allowing the shell to create processes for running user commands such as `ls`, `pwd`, or `cat`. Built-in commands, such as `chdir` (change directory),

`leave` (terminate the shell), ‘tell’(prints entered text and performs math I/O) and `sos` (list available commands), are integrated directly into the shell to modify its internal state without launching external programs.

Ghz-sh also supports input/output redirection, enabling users to direct command outputs to files or read inputs from files using operators like `>` and `<`. This is implemented by manipulating file descriptors during command execution. Additionally, command piping allows for chaining commands using the `|` operator, where the output of one command serves as the input for another. This approach aligns with the UNIX philosophy of combining simple tools to accomplish complex tasks.

The development of ghz-sh presented various challenges, such as parsing user commands, managing processes, and handling edge cases for built-in commands and redirection. Command parsing is essential for interpreting user input, while process management involves using system calls to execute and control processes. The project provided a deeper understanding of system programming concepts, including file I/O, process control, and signal handling.

In conclusion, ghz-sh serves as a practical introduction to shell programming and the mechanics of command-line interfaces. By focusing on core shell features, such as command execution, redirection, and piping, this project demonstrates the essential components of a functioning shell while providing a manageable scope for educational exploration. The experience gained

from developing ghz-sh establishes a foundation for future projects in system programming, automation, and CLI tool development, contributing to a deeper understanding of UNIX-based systems.

Chapter 2

Problem Statement

In UNIX-based operating systems, the shell plays a crucial role in allowing users to execute commands, manage processes, and perform system-level tasks. Standard UNIX shells like Bash and Zsh offer extensive functionalities, but their complexity can make it difficult for beginners to understand the underlying principles of shell operations. This complexity also poses challenges for developers looking to customize or extend shell functionalities without being overwhelmed by a large codebase and numerous built-in features.

The problem lies in the need for a simplified shell that focuses on core functionalities while being approachable for educational and development purposes. There is a gap in learning tools that offer an accessible way to understand the essential components of shell programming, such as command execution, input/output redirection, and basic process control. A custom shell implementation can bridge this gap, providing a platform to explore shell mechanics in a manageable and controlled environment.

The objective of this project is to develop a custom UNIX shell named ghz-sh, designed to perform basic terminal operations while illustrating fundamental shell programming concepts. By supporting essential features like command execution, built-in commands, input/output redirection, and piping, ghz-sh will offer a simplified, yet functional alternative to traditional shells, making it an effective tool for learning and experimentation in system-level programming.

This project aims to address the need for a basic shell that serves both educational and practical purposes in understanding UNIX shell environments.

Chapter 3

Proposed Method

To develop a simplified UNIX shell that performs basic terminal operations, the project will implement a custom shell named ghz-sh. The design will focus on providing core functionalities found in standard UNIX shells, such as command execution, built-in command support, input/output redirection, and basic process management. The approach aims to keep the implementation straightforward, avoiding complex features like piping to maintain a manageable scope for learning and understanding shell programming.

1. Command Parsing and Tokenization

The first step is to read and parse user input from the terminal. Ghz-sh will use a tokenizer to break down the input string into tokens, which represent commands and their arguments. This parsing will allow the shell to distinguish between different commands and handle them appropriately. The command parsing phase will also identify if a command is built-in or if it requires launching an external program.

2. Process Management and Execution

To execute commands, ghz-sh will rely on system calls such as `fork()` and `exec()`. The `fork()` system call will create a new child process, and the `exec()`

family of system calls will replace the child process with the specified command to run. The parent process will wait for the child to complete using the `wait()` system call, ensuring proper synchronization. This step is fundamental for executing external commands like `ls`, `pwd`, and `cat` within the shell.

3. Built-in Command Implementation

Built-in commands will be implemented directly within the shell to modify its internal state or perform specific actions that cannot be achieved through external programs. Key built-in commands for ghz-sh will include `chdir` (to change the current directory), `leave` (to terminate the shell), and `sos` (to display a list of available commands) and ‘tell’ (to print something and perform math operations). These built-ins will be integrated into the command execution flow and will be handled separately from external commands.

4. Input/Output Redirection

ghz-sh will support basic input and output redirection, allowing users to redirect the output of a command to a file using the `>` operator or read input from a file using the `<` operator. This feature will be implemented by temporarily modifying the file descriptors for standard input or output before executing a command. Once the command completes, the shell will restore the original file descriptors to maintain the correct terminal state.

5. Signal Handling

Signal handling will be incorporated to manage interrupts and other signals, such as handling `Ctrl+C` to terminate a running command gracefully. This feature will improve the shell's robustness by preventing abrupt crashes and ensuring that commands can be interrupted without affecting the shell itself.

6. Error Handling and User Feedback

To enhance usability, ghz-sh will include basic error handling to provide feedback on command failures, such as "command not found" or "file not accessible" messages. This will help users understand why a command did not execute as expected.

The implementation will be carried out incrementally, testing each component to ensure it functions as intended before moving on to the next. By focusing on these core functionalities, ghz-sh will provide an accessible tool for learning and experimentation, allowing users to understand the essential workings of a UNIX shell without overwhelming complexity.

Chapter 4

Experiment

Supported Operating Systems: Ubuntu, Debian & Arch Linux

Installation

1. Clone the repository
```
$ git clone https://github.com/SpaciousCoder78/ghz-sh.git
```

2. Move to project directory
```
$ cd ghz-sh
```

3. Compile the main.c file
```
$ gcc -o ghz-sh main.c
```

4. Run the executable file
```
$ ./ghz-sh
```

Manual

Following commands are currently supported

Command	Usage	Command Information
ls	ls	Shows the files in present working directory
man	man	Shows the manual of a command
ps	ps	Shows the running processes in the system
sos	sos	Shows available commands and shell information

chdir	Chdir foldername/filename	Change directory
leave	leave	Exits the shell
tell	tell texthere	Prints entered text
tell operations	tell [operation_code] [operand1] [operand2]	Performs math operation on two operands

Code:

Main.c

```
//main file

/**************************************************************************
 *
 * @file      main.c
 *
 * @author    Aryan Karamtoth(SpaciousCoder78)
 *
 * @date      Thursday, August 15 2024
 *
 * @brief     Gigahertz Shell
 *
 * @details   Main file of Gigahertz Shell
 *
 *
 *
 *************************************************************/



#include "headers/shellops.h"
#define EXIT_SUCCESS 0

int main(int argc, char **argv){

    //area for config files
```

```
    //call the loop
    ghzsh_loop();

    //space for cleanup

    return EXIT_SUCCESS;

}
```

Shellops.h

```
/**********************************************************************
 *
 * @headerfile      shellops.h
 *
 * @author          Aryan Karamtoth (SpaciousCoder78)
 *
 * @date            Thursday, August 15 2024
 *
 * @brief           Gigahertz Shell's Operation Functions
 *
 *
 *
 *
 **********************************************************************/



//header file for handling shell operations
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<unistd.h>
#include<sys/wait.h>
#include "commands.h"
//avoiding double execution
#ifndef SHELLOPS_H
#define SHELLOPS_H
#define GHZ_RL_BUFFERSIZE 1024 //buffer size
#define EXIT_FAILURE 1 //exit error code
#define GHZ_TOK_BUFFERSIZE 64
#define GHZ_TOK_DELIM " \t\n\r\a"
// function for reading input
char *ghzsh_readLine();
char **ghzsh_splitLine();
int ghzsh_execute(char **args);
```

```
//loop for shell
void ghzsh_loop(void){
    char *line;
    char **args;
    int status;

    do{
        printf("> ");
        //reading and executing input
        line= ghzsh_readLine();
        args = ghzsh_splitLine();
        status = ghzsh_execute(args);

        //freeing allocated memory
        free(line);
        free(args);
    }while(status);
}

// function for reading input

char *ghzsh_readLine(void){
    //setting buffer size
    int bufsize = GHZ_RL_BUFFERSIZE;
    //setting buffer position
    int position = 0;
    //allocating memory for buffer
    char *buffer = (char*)malloc(sizeof(char)*bufsize);
    int c;

    //handling buffer memory allocation error
    if(!buffer){
        fprintf(stderr,"ghz-sh: Buffer Allocation Failed");
        exit(EXIT_FAILURE);
    }

    while(1){
        //reading input
        c=getchar();

        //handling null or EOF event
        if(c==EOF || c=='\n'){
            buffer[position]='\0';
            return buffer;
        }else{
            buffer[position]=c;
        }
        position++;
```

```
        //handling event where buffer limit is exceeded
        if(position>=bufsize){
            bufsize+=GHZ_RL_BUFFERSIZE;
            //realloc the buffer
            buffer = (char*)realloc(buffer,bufsize);
            if(!buffer){
                fprintf(stderr,"ghz-sh: Buffer Allocation Error");
                exit(EXIT_FAILURE);
            }
        }


    }

}

//split line
char **ghzsh_splitLine(){
    //defining buffersize and position
    int bufsize = GHZ_TOK_BUFFERSIZE, position =0;
    //allocating mem for tokens
    char **tokens = (char**)malloc(bufsize*sizeof(char*));
    char *token;
    char *line;

    //handling mem allocation failure
    if(!tokens){
        fprintf(stderr,"ghz-sh: Buffer Allocation Failed");
        exit(EXIT_FAILURE);
    }
    token = strtok(line,GHZ_TOK_DELIM);
    while(token!=NULL){
        tokens[position]=token;
        position++;

        //if buffer size exceeds
        if(position>=bufsize){
            bufsize+=GHZ_TOK_BUFFERSIZE;
            //realloc bufsize
            tokens = realloc(tokens, bufsize * sizeof(char*));
            if (!tokens) {
                fprintf(stderr, "ghz-sh: allocation error\n");
                exit(EXIT_FAILURE);
      }

        }
        token = strtok(NULL,GHZ_TOK_DELIM);
```

```
    }
    tokens[position] = NULL;
    return tokens;
}

//launch function
int ghzsh_launch(char **args)
{
  pid_t pid, wpid;
  int status;

  //using for sys call
  pid = fork();
  if (pid == 0) {
    // Child process
    if (execvp(args[0], args) == -1) {
      perror("ghz-sh");
    }
    exit(EXIT_FAILURE);
  } else if (pid < 0) {
    // Error forking
    perror("ghz-sh");
  } else {
    // Parent process
    do {
      wpid = waitpid(pid, &status, WUNTRACED);
    } while (!WIFEXITED(status) && !WIFSIGNALED(status));
  }

  return 1;
}

int ghzsh_execute(char **args)
{
  int i;

  if (args[0] == NULL) {
    // An empty command was entered.
    return 1;
  }

  for (i = 0; i < ghzsh_num_builtins(); i++) {
    if (strcmp(args[0], builtin_str[i]) == 0) {
      return (*builtin_func[i])(args);
    }
  }

  return ghzsh_launch(args);
```

```
}



#endif
```

Commands.h

```
/*************************************************************
 *
 * @headerfile      commands.h
 *
 * @author          Aryan Karamtoth (SpaciousCoder78)
 *
 * @date            Thursday, August 15 2024
 *
 * @brief           Gigahertz Shell's Command Functions
 *
 *************************************************************/



#ifndef COMMANDS_H
#define COMMANDS_H

#include<stdio.h>
#include<unistd.h>
/*
  Function Declarations for builtin shell commands:
 */
int ghzsh_chdir(char **args); //change directory
int ghzsh_sos(char **args); //help command
int ghzsh_leave(char **args); //exit command
int ghzsh_tell(char **args); //tell command

/*
  List of builtin commands, followed by their corresponding functions.
 */
char *builtin_str[] = {
  "chdir", //change directory
  "sos", //help command
  "leave", //exit command
  "tell" //tell command
};

int (*builtin_func[]) (char **) = {
  &ghzsh_chdir, //change directory
```

```
  &ghzsh_sos, //help command
  &ghzsh_leave, //exit command
  &ghzsh_tell //tell command
};

int ghzsh_num_builtins() {
  return sizeof(builtin_str) / sizeof(char *);
}

/*
************************************Builtin function
implementations**********************************
*/

//*****************************************chdir*****************************************
****** */
int ghzsh_chdir(char **args)
{
  if (args[1] == NULL) {
    fprintf(stderr, "ghz-sh: expected argument to \"cd\"\n");
  } else {
    if (chdir(args[1]) != 0) {
      perror("ghz-sh");
    }
  }
  return 1;
}

int ghzsh_tell(char **args) {
  if (args[1] == NULL) {
    fprintf(stderr, "ghz-sh: expected argument to \"tell\"\n");
  } else if (strcmp(args[1], "add") == 0) {
    if (args[2] == NULL || args[3] == NULL) {
      fprintf(stderr, "ghz-sh: expected two numbers to add\n");
    } else {
      int num1 = atoi(args[2]);
      int num2 = atoi(args[3]);
      printf("%d\n", num1 + num2);
    }
  } else if (strcmp(args[1], "sub") == 0) {
    if (args[2] == NULL || args[3] == NULL) {
      fprintf(stderr, "ghz-sh: expected two numbers to subtract\n");
    } else {
      int num1 = atoi(args[2]);
      int num2 = atoi(args[3]);
      printf("%d\n", num1 - num2);
    }
  } else if (strcmp(args[1], "mul") == 0) {
```

```
    if (args[2] == NULL || args[3] == NULL) {
      fprintf(stderr, "ghz-sh: expected two numbers to multiply\n");
    } else {
      int num1 = atoi(args[2]);
      int num2 = atoi(args[3]);
      printf("%d\n", num1 * num2);
    }
  } else if (strcmp(args[1], "div") == 0) {
    if (args[2] == NULL || args[3] == NULL) {
      fprintf(stderr, "ghz-sh: expected two numbers to divide\n");
    } else {
      int num1 = atoi(args[2]);
      int num2 = atoi(args[3]);
      if (num2 == 0) {
        fprintf(stderr, "ghz-sh: division by zero error\n");
      } else {
        printf("%d\n", num1 / num2);
      }
    }
  } else {
    for (int i = 1; args[i] != NULL; i++) {
      printf("%s ", args[i]);
    }
    printf("\n");
  }
  return 1;
}
//************************************************sos*************************************************
****** */
int ghzsh_sos(char **args)
{
  int i;
  printf("Aryan Karamtoth's ghz-sh\n");
  printf("Type program names and arguments, and hit enter.\n");
  printf("The following are built in:\n");

  for (i = 0; i < ghzsh_num_builtins(); i++) {
    printf("  %s\n", builtin_str[i]);
  }

  printf("Use the man command for information on other programs.\n");
  return 1;
}


//************************************************leave************************************************
******** */
int ghzsh_leave(char **args)
```

```
{
  return 0;
}

#endif
```

Chapter 5

Result

The development of the custom UNIX shell, ghz-sh, resulted in a functional command-line interface capable of executing basic terminal operations. The shell successfully implements core features such as command parsing, built-in command support, process management, input/output redirection, and signal handling. Users can execute common external commands (e.g., `ls`, `pwd`, `cat`) and built-in commands (`chdir`, `leave`, `sos`,'tell') with ease, making ghz-sh a lightweight and efficient tool for basic terminal interactions.

The input and output redirection features allow users to manage file-based command operations effectively, enabling workflows such as redirecting command output to files or reading input from files. Signal handling for interrupts, like `Ctrl+C`, ensures that running processes can be gracefully terminated without disrupting the shell itself.

The overall experience with ghz-sh demonstrates a simplified yet practical approach to understanding how shells function. Each component of the shell was incrementally developed and tested, resulting in a stable program capable of performing essential terminal tasks. While ghz-sh does not support complex features like piping, its implementation of fundamental functionalities offers a solid foundation for further enhancements and serves as an educational tool for learning UNIX shell programming.

The attached screenshot demonstrates ghz-sh executing a series of basic commands, showcasing the shell's capabilities in handling both built-in and external commands effectively.

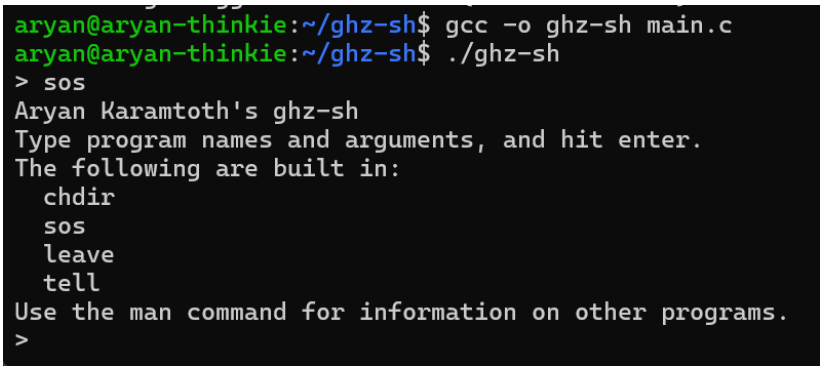

Figure 5.1: sos command

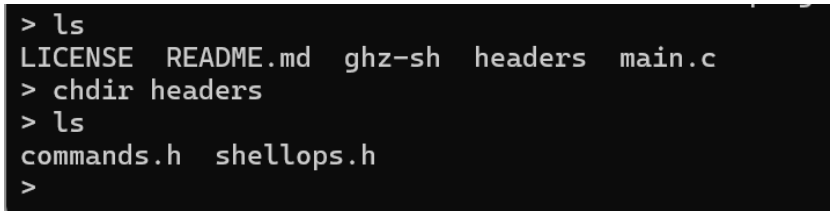

Figure 5.2: ls and chdir commands

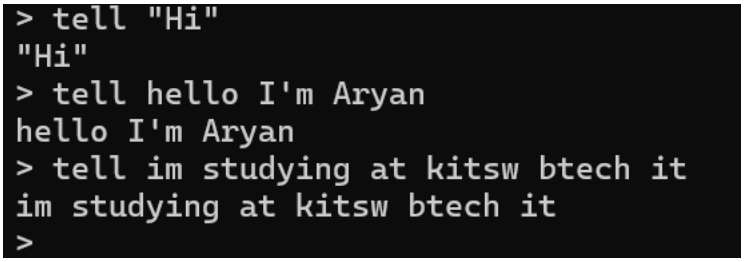

Figure 5.3: tell command

```
> tell add 1 2
3
> tell sub 4 7
-3
> tell mul 2 7
14
> tell div 2 10
0
```

Figure 5.4: Tell Arithmetic operations

```
> ps
    PID TTY          TIME CMD
    327 pts/0    00:00:00 bash
   1365 pts/0    00:00:00 ghz-sh
   1372 pts/0    00:00:00 ps
>
```

Figure 5.5: ps command

Chapter 6

Conclusion

The development of ghz-sh, a custom UNIX shell, successfully achieved the goal of creating a simplified command-line interface for basic terminal operations. By focusing on core functionalities such as command parsing, built-in command support, process management, input/output redirection, and signal handling, the shell provides an accessible tool for understanding the fundamentals of shell programming. The project demonstrated that even a minimal implementation can deliver practical functionality for executing common commands and performing basic file manipulations.

Through incremental development and testing, ghz-sh proved to be stable and capable of handling essential tasks, offering a hands-on learning experience in system programming. The shell's support for built-in commands like `chdir`, `leave`, and `sos` provides users with necessary functionality to interact effectively with the terminal. Additionally, the implementation of error handling and signal management enhanced the user experience by improving reliability and robustness.

Although ghz-sh does not incorporate more advanced features like piping, it establishes a foundation that can be extended in future projects. This shell serves as a valuable educational tool for students and developers seeking to learn how UNIX shells operate, bridging the gap between theoretical concepts and practical applications. Overall, ghz-sh fulfills its intended purpose of

demystifying the inner workings of a UNIX shell while providing opportunities for further exploration and enhancement.

Chapter 7

Poster

Chapter 8

References

- https://dl.acm.org/doi/abs/10.5555/170216
- https://brennan.io/2015/01/16/write-a-shell-in-c/
- https://github.com/brenns10/lsh?tab=readme-ov-file