Lab Assignment: Building a Simple Shell in C

DS2040, Satyajit Das

March 13, 2025

1 A few points to be noted

The objective of this lab is to implement a simple command-line shell in C that demonstrates:

- Process creation using `fork()`.
- Executing external commands using `execve()`.
- Handling signals such as `SIGINT` (Ctrl+C).
- Implementing internal commands like `cd`.
- Maintaining command history.
- Running built-in programs (e.g., addition, subtraction).

2 Getting Started

Create a new C file and open it in an editor:

```
touch my_shell.c
```

3 Implementation

Below is the full C implementation of the shell.

3.1 Header Files

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/wait.h>
#include <signal.h>

#define MAX_INPUT_SIZE 1024
#define MAX_ARG_SIZE 100
#define MAX_HISTORY 10
```

3.2 Command History Management

```c
char *history[MAX_HISTORY];
int history_count = 0;

void add_to_history(char *command) {
    if (history_count < MAX_HISTORY) {
        history[history_count] = strdup(command);
    } else {
        free(history[0]);
        for (int i = 1; i < MAX_HISTORY; i++) {
            history[i - 1] = history[i];
        }
        history[MAX_HISTORY - 1] = strdup(command);
    }
    history_count = (history_count < MAX_HISTORY) ?
       history_count + 1 : MAX_HISTORY;
}

void print_history() {
    for (int i = 0; i < history_count; i++) {
        printf("[%d] %s\n", i + 1, history[i]);
    }
}
```

3.3 Handling the cd Command

```c
void handle_cd(char **args) {
    if (args[1] == NULL) {
        fprintf(stderr, "cd: expected argument\n");
    } else {
        if (chdir(args[1]) != 0) {
            perror("cd failed");
```

```
        }
    }
}
```

3.4 Built-in Programs

```
void built_in_programs(char **args) {
    if (strcmp(args[0], "hello") == 0) {
        printf("Hello, welcome to my shell!\n");
    } else if (strcmp(args[0], "add") == 0) {
        if (args[1] && args[2]) {
            int sum = atoi(args[1]) + atoi(args[2]);
            printf("Sum: %d\n", sum);
        } else {
            printf("Usage: add <num1> <num2>\n");
        }
    } else if (strcmp(args[0], "subtract") == 0) {
        if (args[1] && args[2]) {
            int diff = atoi(args[1]) - atoi(args[2]);
            printf("Difference: %d\n", diff);
        } else {
            printf("Usage: subtract <num1> <num2>\n");
        }
    } else {
        printf("Unknown command: %s\n", args[0]);
    }
}
```

3.5 Executing External Commands

```
void execute_external(char **args) {
    pid_t pid = fork();
    if (pid < 0) {
        perror("Fork failed");
        return;
    }
    if (pid == 0) {
        if (execvp(args[0], args) == -1) {
            perror("Execution failed");
        }
        exit(EXIT_FAILURE);
    } else {
        waitpid(pid, NULL, 0);
    }
}
```

3.6 Signal Handling

```
void signal_handler(int sig) {
    if (sig == SIGINT) {
        printf("\nCaught SIGINT (Ctrl+C), shell continuing
            ...\n");
    }
}
```

3.7 Executing Commands

```
void execute_command(char *input) {
    char *args[MAX_ARG_SIZE];
    char *token = strtok(input, " ");
    int i = 0;

    while (token != NULL) {
        args[i] = token;
        i++;
        token = strtok(NULL, " ");
    }
    args[i] = NULL;

    if (args[0] == NULL) {
        return;
    }

    add_to_history(input);

    if (strcmp(args[0], "cd") == 0) {
        handle_cd(args);
    } else if (strcmp(args[0], "history") == 0) {
        print_history();
    } else if (strcmp(args[0], "hello") == 0 || strcmp(args
        [0], "add") == 0 || strcmp(args[0], "subtract") == 0)
        {
        built_in_programs(args);
    } else {
        execute_external(args);
    }
}
```

3.8 Main Shell Loop

```
int main() {
    char input[MAX_INPUT_SIZE];
```

```
    signal(SIGINT, signal_handler);

    while (1) {
        printf("my_shell> ");
        fflush(stdout);

        if (fgets(input, MAX_INPUT_SIZE, stdin) == NULL) {
            break;
        }

        size_t len = strlen(input);
        if (input[len - 1] == '\n') {
            input[len - 1] = '\0';
        }

        execute_command(input);
    }

    return 0;
}
```

4 Compiling and Running

To compile and run the shell:

```
gcc my_shell.c -o my_shell
./my_shell
```

5 Example Usage

```
my_shell> cd /home
my_shell> history
[1] cd /home
my_shell> hello
Hello, welcome to my shell!
my_shell> add 4 5
Sum: 9
my_shell> subtract 10 2
Difference: 8
```

6 Tasks

Wait for the instructions...