Building the Shell in Novix OS

Congratulations!
You've reached one of the most exciting stages of building your operating system — implementing a **command shell**.

So far, we've learned how to:

- Create new processes with `fork()`
- Load programs with `exec()`
- Exit and clean up with `exit()` and `wait()`
- Communicate through system calls

Now it's time to tie it all together into a **simple interactive shell** — the *heart* of userland.

Learning Goals

By the end of this lesson, you'll understand how to:

- Build a simple interactive command shell for Novix
- Implement built-in commands (`help`, `clear`, etc.)
- Parse user input and tokenize commands
- Launch ELF programs dynamically using `fork()`, `exec()`, and `wait()`
- Handle basic user input editing and echo behavior

What is a Shell?

A **shell** is a command interpreter.
It provides a user interface to the OS, letting users type commands, run programs, and see output.

In Unix systems, `/bin/sh` or `/bin/bash` serve this role.
In Novix, we're building our own minimal version — a **tiny, self-contained shell running in user space**.

Source Code: `user/shell.c`

Let's look at the complete implementation step by step.

```
#include "include/stdio.h"
#include "include/malloc.h"
#include "include/string.h"
#include "include/syscall.h"

#define MAXLINE 128
#define MAX_ARGS 8

void print_help(void) {
    puts("Available commands:\n");
    puts("  help         Show this overview\n");
    puts("  hello        Shows the hello message\n");
    puts("  clear        Clear the screen\n");
}
```

We start with some includes and helper constants.
The `print_help()` function prints the list of supported commands.

Main Shell Loop

The `main()` function implements the interactive loop.

```c
int main() {
    char line[MAXLINE];
    char filename[MAXLINE + 8];  // room for ".elf"

    cls();
    puts("Novix RISC-V 64 OS, (c) NovixManiac, Shell version : 0.0.1\n");

    while (1) {
        // 1. Show prompt
        puts("$ ");

        int pos = 0;
        char c;
```

We begin by clearing the screen and showing a version banner.
The loop continues indefinitely — just like any command-line shell.

Reading Input Character by Character

```c
        while (1) {
            int n = read(0, &c, 1);
            if (n != 1) continue;
            putc(c);

            if (c == '\r' || c == '\n') {
                putc('\n');
                break;
            }

            if ((c == 0x7F || c == '\b') && pos > 0) {
                pos--;
                puts("\b \b");
                continue;
            }

            if (pos < MAXLINE - 1) {
                line[pos++] = c;
            }
        }

        line[pos] = '\0'; // null-terminate string
```

This code manually reads one character at a time from **stdin** (`read(0, ...)`).
It handles:

- **Enter** (`\n` / `\r`) → End of line
- **Backspace** → Delete previous character and update cursor
- Echoing input directly to the terminal

Why not use `fgets()`?
Because in Novix userland, we don't have the C standard library — we're writing everything ourselves at a low level.

Parsing the Command Line

```
        if (line[0] == '\0') continue;

        char* argv[MAX_ARGS];
        int argc = 0;

        char* token = strtok(line, " ");
        while (token && argc < MAX_ARGS) {
            argv[argc++] = token;
            token = strtok(NULL, " ");
        }

        if (argc == 0) continue;
        char* cmd = argv[0];
```

Here we **tokenize** the user input string using `strtok()`.
This splits the line into words separated by spaces — perfect for command arguments.

Handling Built-in Commands

Before attempting to run a program, we check for **internal commands**.

```
        if (!strcmp(line, "clear")) {
            cls();
            puts("Novix (64-bits), Shell version : 0.0.1\n");
            continue;
        }

        if (!strcmp(line, "help")) {
            print_help();
            continue;
        }

        if (!strcmp(line, "shell")) {
            printf("[shell] Command '%s' not allowed\n", line);
            continue;
        }
```

- `clear` — calls our system `cls()` to clear the terminal
- `help` — shows the command list
- Prevents recursive execution of `shell.elf`

Running External Programs

Now the interesting part: launching ELF binaries!

```
        strcpy(filename, cmd);
        strcat(filename, ".elf");

        if (!tarfs_exists(filename)) {
            printf("[shell] Command '%s' not found\n", line);
            continue;
        }
        int pid = fork();
        if (pid == 0) {
            exec(filename);
            puts("[shell] exec failed\n");
            exit(1);
        } else if (pid > 0) {
            int status;
            wait(&status);
        } else {
            puts("[shell] fork failed\n");
        }
    }

    return 0;
}
```

Step-by-Step Breakdown

1. **Add the `.elf` suffix**
   The shell expects all programs to be ELF files in the TARFS filesystem.
2. **Check existence**
   Uses the system call `tarfs_exists()` to verify that the binary is available.
3. **Fork and Exec**
   - The **child** uses `exec()` to replace itself with the new program
   - The **parent** waits for the child to exit (`wait()`)
4. **Error handling**
   If either `fork()` or `exec()` fails, we print an error message.

Testing the Shell

You can now run the shell in your Novix environment.
Try typing commands like:

```
$ help
Available commands:
  help         Show this overview
  hello        Shows the hello message
  clear        Clear the screen

$ hello
[hello] Hello from exec!

$ clear
Novix (64-bits), Shell version : 0.0.1
```

It works! You can now run userland ELF binaries dynamically — directly from your own shell.

Summary

In this chapter, we built the **Novix userland shell**, combining everything we've learned:

- Input handling and character echo
- Command parsing and tokenization

- Built-in commands (`help`, `clear`, etc.)
- External ELF execution with `fork()`, `exec()`, `wait()`
- Dynamic process management from userland

This marks a **huge milestone**: Novix OS is now **interactive** and ready to run user programs just like Unix.

```
*A smile on my face - because Novix OS has now his first shell!*
```