TinyShell Building Blocks

C/C++ Reference Guide - Phase 1

Project Overview

TinyShell is a minimal Unix shell written in C/C++ for Linux. It mimics basic shell behavior while remaining lightweight and educational. This guide provides the essential building blocks needed for implementation.

Core Requirements

- Display a command prompt
- Parse command-line arguments
- Locate executables in PATH
- Execute commands using fork() and execve()
- Report exit codes
- Execute standard programs (ls, cat, echo)
- Terminate gracefully on EOF or 'exit' command

1. Required Headers

Header	Key Functions
`&lt;stdio.h&gt;`	printf, fgets, perror
`&lt;stdlib.h&gt;`	exit, malloc, getenv
`&lt;string.h&gt;`	strtok, strcmp, strlen
`&lt;unistd.h&gt;`	fork, execve, access
`&lt;sys/types.h&gt;`	pid_t
`&lt;sys/wait.h&gt;`	wait, waitpid, WIFEXITED

2. Essential System Calls

Process Management

• **fork()** - Creates a child process (returns 0 in child, PID in parent)

• **execve() / execvp()** - Replaces process image with new program

• **wait() / waitpid()** - Waits for child process termination

• **exit()** - Terminates the calling process

PATH Handling

• **getenv("PATH")** - Retrieves PATH environment variable

• **access(path, X_OK)** - Checks if file is executable

3. Implementation Components

Main Loop: Infinite loop: prompt → read → parse → execute → wait

Display Prompt: Use printf() with fflush(stdout)

Read Input: Use fgets() and check for EOF (NULL return)

Parse Arguments: Tokenize input with strtok() using space/tab/newline delimiters

Handle Built-ins: Check for 'exit' command before forking

Find Executable: Search PATH directories or use absolute/relative paths

Execute Command: fork() → child calls execve() → parent calls waitpid()

Report Exit Code: Use WIFEXITED() and WEXITSTATUS() macros

4. Important Concepts

Process Creation Pattern

After fork(), you have two processes. The child (pid == 0) executes the command. The parent (pid > 0) waits for the child to complete. Check fork() return value for errors (pid < 0).

Argument Array Format

The execve() family requires a NULL-terminated array of strings. First element is the program name, remaining elements are arguments.

Exit Status Macros

• **WIFEXITED(status)** - True if process exited normally

• **WEXITSTATUS(status)** - Extracts exit code (0-255)

• **WIFSIGNALED(status)** - True if terminated by signal

5. Error Handling

• Always check fork() return value (< 0 indicates failure)

• Use perror() to print system error messages

• Handle EOF on input (fgets() returns NULL)

• Check if command exists before executing

• Use exit code 127 for 'command not found'

• Ensure proper cleanup in child process on exec failure

6. Suggested Program Structure

```
main() { while (1) { 1. Display prompt 2. Read input (check EOF) 3. Parse into
argument array 4. Check for built-in commands (exit) 5. Locate executable in PATH 6.
Fork process - Child: execute command - Parent: wait and report status } }
```

7. Testing Your Shell

• Test with simple commands: ls, pwd, echo hello

• Test with arguments: ls -la /tmp

• Test built-in exit command

• Test EOF (Ctrl+D) termination

• Test invalid commands (verify error handling)

• Test exit code reporting with: /bin/true and /bin/false

8. Compilation

```
gcc -o tinyshell tinyshell.c -Wall -Wextra
./tinyshell
```

Implementation Tips

• Start with the main loop and prompt display

• Add input reading and basic parsing next

• Implement the 'exit' built-in before fork/exec

• Test fork() and wait() before adding exec()

• Handle PATH searching last

• Use #define for constants (MAX_ARGS, BUFFER_SIZE)

• Remember: strings in C need null terminators

Note: This is supplementary material. Refer to POSIX documentation and your textbook for detailed API specifications.