ELEC 377- Operating Systems

Lab 2: Write a Simple Shell
Design Document

Bradley Stephen & Ben Wyand

Program Description

This program implements a simple command-line shell. Its main features include internal commands such as cd, ls, pwd, and exit. If an internal command is not recognized, the shell checks for external commands by searching a few common directories like /bin and /usr/bin. If the command is still not found, the shell prints an error message indicating the command wasn't recognized.

1. **Main Loop**: The shell operates in an infinite loop, waiting for user input via fgets(). After the input is captured, it is tokenized into commands and arguments by splitCommandLine(). If a command matches an internal function (like cd or exit), it is executed immediately. For external commands, the shell attempts to locate and run the executable from directories specified in the path[] array, using fork() and execv() to execute the command in a child process while the parent waits for it to complete.
2. **Internal Commands**:
   Internal commands (cd, ls, pwd, exit) are mapped to corresponding functions using function pointers. This is done using a commands[] structure, which pairs command names with function pointers to the corresponding handler functions. If the command matches one of these names, the corresponding function is invoked. For example, cd uses chdir() to change the current directory, and pwd uses getcwd() to print the working directory.
3. **External Commands**:
   When an internal command is not detected, the shell assumes it's an external command. It searches through the directories listed in path[], constructing a possible executable path using snprintf(). The stat() system call checks if the file

exists and is executable. If found, the shell forks a new child process, which replaces its image with the command via execv(), while the parent waits using wait().

4. **Error Handling**: The shell ensures proper error handling with system calls like fork(), execv(), and chdir(), printing error messages if any of these functions fail.

Special C Features

- **Function Pointers**:
  A key feature of this shell is the use of **function pointers** for internal command handling. The commands[] array contains command names alongside pointers to their respective functions, allowing the shell to directly map command names to execution functions. This approach avoids hardcoded conditional logic and improves modularity. As seen:

  ```
  struct cmdData commands[] = {
     {"exit", exitFunc},
     {"pwd", pwdFunc},
     {"cd", cdFunc},
     {"ls", lsFunc},
     { NULL, NULL}
  };
  ```

- **System Calls**:
  Several Unix system calls are used throughout the shell for process management and file operations:
  - fork() creates a new child process.
  - execv() replaces the child process with a new program.
  - wait() ensures the parent process waits for the child to finish.
  - stat() checks file attributes, including whether a file is executable.
  - getcwd() retrieves the current working directory for pwd.
  - chdir() changes the directory for cd.

- **Memory Management**:
  The shell uses malloc() to dynamically allocate memory for paths when searching for external commands. The allocated memory is freed appropriately after command execution using free(), preventing memory leaks.
- **Command Line Parsing**:
  The function splitCommandLine() tokenizes the user input into arguments. It processes input strings, splitting them at spaces and ensuring that the number of arguments does not exceed the maximum allowed (MAXARGS). It also handles edge cases like leading and trailing spaces.