Test Plan for Shell Implementation

1. Objectives

1. Validate the behaviour of internal commands (e.g., cd, exit)
    - Ensure that the shell correctly handles internal commands like cd (change directory) and exit (terminate the shell).
2. Ensure proper handling of signals (e.g., SIGINT)
    - Verify that the shell handles signals like SIGINT (Ctrl+C) gracefully, terminating foreground processes without crashing the shell.
3. Confirm that the shell exits gracefully.
    - Ensure that the shell exits cleanly when the exit command is issued, freeing all allocated memory and closing open file descriptors.
4. Validate behaviour of jobs and command parsing
    - Test the shell's ability to parse commands correctly, including handling of pipelines, input/output redirection, and background processing.
5. Validate the behaviour of run and get jobs
    - Ensure that the run_job and get_job functions work as expected, correctly executing commands and handling job structures.
6. Validate successful memory allocation and free all memory
    - Verify that all memory allocated during the shell's operation is properly freed, preventing memory leaks.
7. Ensure proper background processing
    - Test the shell's ability to run commands in the background (using &) and manage background jobs.
8. Ensure proper FILE I/O
    - Validate that input/output redirection (<, >) works correctly, reading from and writing to the specified files.

---

2. Scope

The test plan covers the following areas:

- Command parsing and execution.
- Signal handling.
- Memory management.
- Background processing.
- Input/output redirection.
- Job management (e.g., get_job, run_job).

---

3. Test Environment

- Operating System: Linux
- Compiler: GCC

- Shell Implementation: Custom shell written in C
- Tools: Valgrind (for memory leak detection), GDB (for debugging)

---

4. Test Cases

4.1. Internal Commands

1. Test Case: cd Command
    - Input: cd /path/to/directory
    - Expected Output: The shell changes the current working directory to /path/to/directory.
    - Validation: Use pwd to verify the directory change.
2. Test Case: exit Command
    - Input: exit
    - Expected Output: The shell terminates gracefully.
    - Validation: Ensure the shell process exits with status code 0.

4.2. Signal Handling

3. Test Case: SIGINT Handling
    - Input: Run a long-running command (e.g., sleep 10) and press Ctrl+C.
    - Expected Output: The foreground process terminates, but the shell continues running.
    - Validation: Verify that the shell does not crash and remains responsive.

4.3. Memory Management

4. Test Case: Memory Allocation and Free
    - Input: Execute multiple commands with varying complexity (e.g., pipelines, redirection).
    - Expected Output: No memory leaks.
    - Validation: Use Valgrind to check for memory leaks after executing a series of commands.

4.4. Command Parsing

5. Test Case: Simple Command
    - Input: /bin/ls -l
    - Expected Output: The shell executes the ls -l command and displays the directory listing.
    - Validation: Compare the output with the output from a standard shell.
6. Test Case: Pipeline Command
    - Input: /bin/ls | /usr/bin/grep
    - Expected Output: The shell executes the pipeline and displays only .c files.
    - Validation: Verify the output matches the expected filtered list.

4.5. Background Processing

7. Test Case: Background Command
    - Input: /bin/sleep 10 &
    - Expected Output: The shell runs the sleep command in the background and immediately returns the prompt.

- Validation: Use ps to verify that the sleep process is running in the background.

4.6. Input/Output Redirection

8. Test Case: Input Redirection
    - Input: /bin/wc < input.txt
    - Expected Output: The shell reads from input.txt and displays the word count.
    - Validation: Compare the output with the result of wc input.txt.
9. Test Case: Output Redirection
    - Input: /bin/ls > output.txt
    - Expected Output: The shell writes the directory listing to output.txt.
    - Validation: Verify the contents of output.txt.

4.7. Job Management

10. Test Case: get_job Function
    - Input: A command with input/output redirection (e.g., wc < input.txt > output.txt).
    - Expected Output: The get_job function correctly parses the command and sets the infile_path and outfile_path fields in the Job structure.
    - Validation: Print the Job structure to verify the fields are set correctly.
11. Test Case: run_job Function
    - Input: A pipeline command (e.g., ls | grep .c).
    - Expected Output: The run_job function executes the pipeline and displays the correct output.
    - Validation: Compare the output with the result from a standard shell.

---

5. Issues Reported

- Issue 1: usr/bin/wc < infile > outfile does not work.
    - Description: The shell fails to handle input/output redirection for certain commands.
- Issue 2: Unable to get Ctrl+Z to work.
    - Description: The shell does not handle the SIGTSTP signal (Ctrl+Z) correctly.

---

6. Test Execution

- Test Execution Steps:
    1. Compile the shell using makefile.
    2. Run each test case manually or using a test script.
    3. Record the results and compare them with the expected output.

---

7. Expected Results

- All test cases should pass, with the shell behaving as expected for each scenario.
- No memory leaks should be detected.
- The shell should handle signals and background processes correctly.

8. Conclusion

This test plan ensures comprehensive coverage of the shell's functionality, including command parsing, signal handling, memory management, and job execution. Any issues identified during testing should be documented and addressed before final submission.