CS3480 – Principles of Secure Application Development

Lab 1 Report: Understanding, Creating and Using Linux Processes

Name: Parishith Ragumar
Student ID: 220444K

Part 1 – `lab1p1.c`

Approach

1. Looped through the input arguments passed to the program.
2. Whenever a `'+'` sign was encountered, parsed arguments so far (maximum of 8 arguments) were stored in a 2D string array as a command.
3. For each command, a new child process was created using `fork()`, and the command was executed using `execve()` as specified.
4. The parser continued scanning the remaining arguments in the same way.
5. After the last `'+'` sign was encountered, the remaining arguments were considered as a separate command and executed as in step 3.
6. For cases where no arguments existed between two `'+'` signs, the special command `/bin/true` was inserted, which performs no action but exits successfully.

Challenges & Resolutions

- **Empty command segments:** Initially caused crashes when no arguments were passed. Solved by checking if the segment was empty and inserting `/bin/true`.
- **Argument limit handling:** Enforced a maximum of 8 arguments using a boundary check to prevent buffer overflows.
- **Debugging execution failures:** Used `perror()` to output meaningful error messages during `execve()` failures, which helped identify incorrect paths or argument formats.

Part 2 – `lab1p2.c`

Approach

1. Counted the number of `%` placeholders in the command-line arguments. If the number exceeded 8, exited the program.
2. Read multiple lines from `stdin`, each line corresponding to inputs for `%` replacements. Each line could contain fewer or equal words than the number of `%` symbols. (If the inputs exceeded, those were not considered.)
3. Stored input words in a 3D array and, for each line, created a command by replacing the `%` placeholders with corresponding input words.
4. Executed each resulting command using a separate child process via `fork()` and `execvp()`.

Challenges & Resolutions

- **Position handling of %:** Initially assumed `%` appeared only at the end. Later modified the code to identify the exact index of `%` symbols and replace them accordingly.
- **Handling lines with fewer words:** Modified logic to gracefully skip unused `%` symbols if a line had fewer words.
- **Command argument assembly:** Used dynamic construction of the `exec_args[]` array per line to handle the different combinations of fixed arguments and user-provided input.
- **Input limits and safety:** Enforced bounds for input lines and word length, ensuring safe handling for all test cases.