CS 536 Spring 2024

Lab 1: System Programming Review

Alireza Lotfi

January 30, 2024

Contents

1 Problem 1

1.1 Issue Analysis

The failure to execute the `ls -l` command stems from the method used to handle user input. Upon receiving user input, the program directly passes the input buffer to the `execlp()` function without parsing or tokenizing it.

The function expects the command and its arguments to be passed as separate arguments. However, when the `ls -l` command is entered as input, it is treated as a single string within the input buffer. As a result, `execlp()` interprets `ls -l` as the command itself, rather than recognizing `ls` as the command and `-l` as its argument. This lack of argument separation leads to failing to locate the command in the system directories.

1.2 Recommendation

To address this issue and enable successful execution of commands with arguments, it is imperative to implement input parsing or tokenization. By breaking down user input into distinct command and argument components, the program can pass them to the `execvp()` function for execution as the problem itself also suggests in the later steps.

2 Problem 2

2.1 Input Length Limit

2.1.1 Client Side

In the client side, we will easily be able to check the length as below and ignore it if it is greater than 30 which is our max command length:

```c
if (strlen(input) > MAX_COMMAND_LENGTH) {
    printf("Error: Command exceeds maximum length\n");
    continue;
}
```

2.1.2 Server Side

Although we are handling the case in the client side, we will make sure to handle any unforeseen issues regarding the length limit in this side as well. We will follow the steps as below:

1. Read from FIFO

```c
total_bytes_read = 0;
while (total_bytes_read < MAX_FIFO_READ - 1) {
    bytes_read = read(fd, buf + total_bytes_read, MAX_FIFO_READ - 1 -
    total_bytes_read);
    if (bytes_read == -1) {
        perror("Error reading from FIFO");
        close(fd);
        exit(EXIT_FAILURE);
    }
    else if (bytes_read == 0)
        break;
    total_bytes_read += bytes_read;
}
```

2. Check end of command

```c
if (buf[total_bytes_read - 1] == '\n') {
    break;
}
```

3. Check command length

```c
if (total_bytes_read > MAX_COMMAND_LENGTH) {
    continue;
}
```

2.2 Linux FIFOs interleaving

In Linux FIFOs, the system call is guaranteed to be atomic when writing up to a certain maximum number of bytes. In most Linux systems, the atomic pipe buffer size is typically **4 KB (4096 bytes)**. If the amount of data being written exceeds the atomic pipe buffer size, the `write()` operation may still succeed, but it won't be atomic.

3 Problem 3

3.1 Main Differences

There are two main differences between `execvp()` and `execve()`:

- The `execvp()` function takes only **2 parameters**, but `execve()` takes **3 parameters** which are listed below:
  1. Filename
  2. Array of ”command line” arguments
  3. Array of environment variable
- The `execvp()` function will examine the directories in your PATH environment variable when searching for the executable file. However, this is not the case for `execve()` and the filename should be an absolute or relative path. For example for `ls`, you would need to type the path as `/bin/ls` instead of `ls`.

3.2 Restrictions

The main restriction of using `execve()` instead of `execvp()` is that `execve()` requires you to provide the full path to the executable file you want to execute, whereas `execvp()` allows you to specify just the name of the executable and relies on the system’s search path to find the executable file.