TDDB68/TDDD47 - Lab 4

Felipe Boeira

February 2020

1 Goal

Now that you have implemented the `exec()` system call, new processes can be spawned by user programs. One way to send instructions to programs is through the usage of command line arguments, and this will be covered in this assignment. The main goal is to understand how the arguments are organized in the x86 architecture, and implement the parsing and layout of arguments when spawning new processes.

2 Overview

This assignment covers:

- Understanding the memory layout of arguments in the x86 architecture
- Parse arguments received in a `char` array
- Pushing arguments to the stack according to the calling convention

3 Preparatory Questions

Take some time to go through these questions before running into the code:

- What are `argc` and `argv` that `main` in a **C** program takes as arguments?
- Where are they stored when the program begins executing?

4 Assignment in Detail

A user program may be called with arguments in the command line. Implement arguments passing, so the arguments of a user program can be accessible within it (**details**).

At first, you should go into `userprog/process.c`, find the function `setup_stack()` and change back the following line:

esp = PHYS_BASE - 12;
into
esp = PHYS_BASE;

Now your program will always fail until you implement argument passing. Try to run any user program.

Stop! Before continuing, think about why you have KERNEL PANIC after you have removed "-12". What is wrong and why did the program work before?

The user program with arguments should be called with apostrophes (') from the Pintos command line. Consider we have a program called "binary", then we run `pintos --qemu -- run 'binary -s 17'`, where binary is called with arguments `-s` and `17`.

When the user program with arguments is called from `exec()`, you have to call it as follows: `exec("binary -s 17")`. Figure 1 depicts an example of stack layout when executing such command, your task is to build such layout when spawning a new process according to *char *cmd_line*.

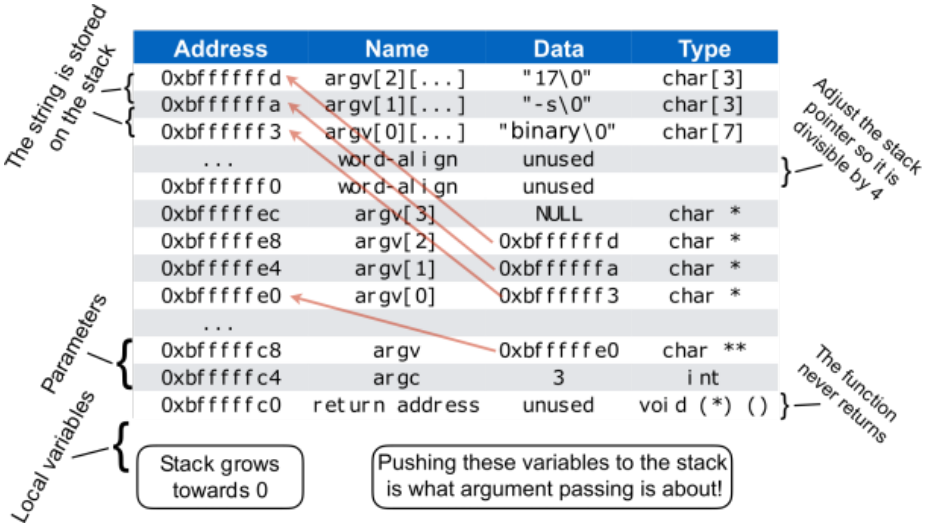

Figure 1: Argument passing details

Although you can parse the string from the command line in the way you prefer, we recommend you to have a look at the function `strtok_r()`, prototyped

in `lib/string.h` and implemented with thorough comments in `lib/string.c`. You can find out more about it by looking at the man page (run `man strtok_r` at the prompt). We suggest that you limit the number of arguments to, for example, 32, which will simplify your implementation because you can use a static array of a fixed size to store the parsed arguments.

Necessary details about setting up the stack for this task you can find in **Program Startup Details** section of the Pintos documentation.

5 Helpful Information

Code directory: `src/userprog`, `src/threads`, `src/lib`, `src/lib/kernel`
Textbook chapters: Chapter 2.3: System Calls
Chapter 2.4: Types of System Calls
Chapter 4.4: Threading Issues
Chapter 6.2: The Critical-Section Problem
Chapter 8.4: Paging
Documentation: Pintos documentation
(Always remember that the TDDB68 lab instructions always have higher precedence)

6 Acknowledgement

This document is based on previous content from the course web pages.