TDDE47/TDDE68
Lab 4: exec

Dag Jönsson

February 5, 2024

1 Goal

In the following assignments you are supposed to learn about multiprogramming environment, synchronization between user programs, and setting up the program stack. The main goal is to understand how important synchronization is in the multiprogramming environment and implement a set of system calls in Pintos that allow synchronization between the user programs.

2 Overview

This assignment covers:

- Execution of multiple user programs in Pintos with `exec()`
- Synchronization

3 Preparatory Questions

Before you begin doing your lab assignment, you have to answer the following set of questions to ensure that you are ready to continue:

- How are you going to enable communication between the parent and child processes during startup of the child process?
- What is the information that you will need to include in the data structures to be shared between parent and child?
- Which synchronization mechanisms can be applied?

4 User Programs

In the previous assignment (Lab 2), you had only one user program running by the operating system. In this assignment, you are supposed to implement the `exec()` system call so that user programs are able to invoke other programs. Hence, programs will be allowed to invoke children user programs and, if needed, may wait for completion of these child programs (the `wait()` system call will be implemented in lab 5!). In such a multiprogramming environment, synchronization becomes important since the kernel may access shared data structures between threads. The implementation of a multiprogramming environment will be accomplished in this and the next lab.

In Lab 5, you will implement the `wait()` system call so that parents may wait until children processes exit to check their return value. In Lab 5 you will add the structures needed to track the relationship between a parent process and its children. This will most likely mean you will have to revist your solution in this lab, so make sure you write your solution in a structured fashion to make it easier in the future.

Note that in this assignment you are not supposed to implement synchronized access to the file system!

In this assignment, you will need to implement one new system call, exec().

- exec - Loads a program into memory and executes it in its own thread or process.

5 Preparation

Read through `userprog/process.c` and `threads/thread.c` to make sure you have an solid understanding of the flow when starting a new process. You need to have a clear understanding of how the user program is loaded into memory and how it is then started. You have probably already read through parts of this code, but repeat it again to make sure you understand what is going on. A diagram to help you understand the flow of a process creation is depicted in Figure 1.

The main part of this assignment is to implement the following system call:

`pid_t exec (const char *cmd_line)`
Runs the executable whose name is given in `cmd_line`, passing any given arguments, and returns the new process' program id (`pid`). Must return pid -1 if the program cannot load or run for any reason.

Figure 1 depicts the process creation flow in pintos. Note that **P** represents the parent and **CHILD** is the new process being spawned by an `exec()` call from the parent. Revisit this diagram if you need to think about when to initialize data structures or how to share information between the parent and the child processes.

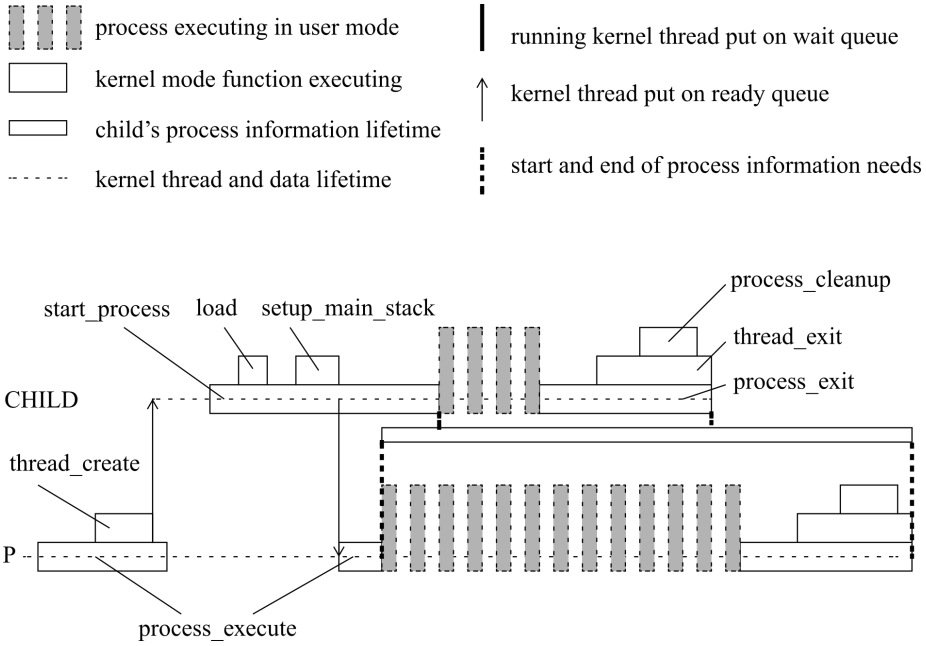

Figure 1: Process execution flow

5.1 Assignment in detail

Once you have clearly understood the flow of executing a new process in Pintos, you must decide what kind of structure you need to create to be able to start, and get the status of the new process to decide if it was successfully started or not. Remember to deallocate any allocated memory that isn't used any longer.
Hints:

- `start_process` is only called via `process_execute`, which means you can change the parameter of `start_process` to something other than `cmd_line`. It still needs to be a `void*` though. How is this handled currently with the `cmd_line`?
- When the parent creates a new child, it is put into the process queue and the child thread will run `start_process()` to be initialised. Use synchronisation mechanisms so that the parent waits until its child initialisation is complete and can check whether it was successful or not.

5.2 Testing

There are two test programs you can use to test your solution, `examples/lab4test1.c` and `examples/lab4test2.c`. Their function is documented at the top of the respective files.

6 Helpful Information

Code directory: `userprog`, `threads`, `lib`, `lib/kernel`
Textbook chapters:

- Chapter 2.3: System Calls
- Chapter 4.6: Threading Issues
- Chapter 6.2: The Critical-Section Problem
- Chapter 9.3: Paging

Documentation: Pintos documentation
(Always remember that the TDDE47/TDDE68 lab instructions have higher precedence)

7 Acknowledgement

Parts of this document contains material from the TDIU16 course at LiU, previous lab instructions found on the course web page, or from previous lab instructions written by Felipe Boeira.