TDDB68/TDDE47 - Lab 3

Felipe Boeira

February 2020

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()`
- Tracking of parent and children relationship

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 track all children of a given parent and the parent of every children?
- What is the information that you will include in the data structures to be shared between parent and children?
- Which synchronization mechanisms can be applied?

4 User Programs

In the previous assignment (Lab 1), 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 set of labs:

Lab 4 is to implement argument passing such that programs can read command line arguments. This will teach you how the operating system can pass arguments to user programs, in particular, how it is solved in 80x86 architecture by pushing them into the program's stack.

In Lab 5, you will implement the `wait()` system call so that parents my wait until children processes exit to check their return value. In this lab you will have to use the data structures you define now for parent and children tracking.

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 and modify `exit()` so that it stores the return value of the child (needed later for `wait()`) and deallocates your tracking data structure if it is not going to be used anymore by either the parent or the child (e.g. if one of them is still executing, then it cannot be deallocated yet).

- exec - Loads a program into memory and executes it in its own thread or process.
- exit - Terminates a program and deallocates resources occupied by the program, for example, closes all files opened by the program. You inherit this system call from earlier labs and extend it.

5 Preparation

In addition to the source code which you have already looked at in the second lab, you should read and understand the code in `userprog/process.c`. This file contains a set of functions that you can use to implement execution of a new user program. You have to clearly understand how the user program is loaded into the memory and how it is started. A diagram to help you understanding the flow of a process creation is depicted in Figure 1.

The main part of this assignment is to implement (extend) the following system calls:

`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. For now you may ignore the arguments in `cmd_line` and use only the program name to execute it.

`void exit (int status)`
Terminates the current user program, returning the exit code status to the kernel.

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.

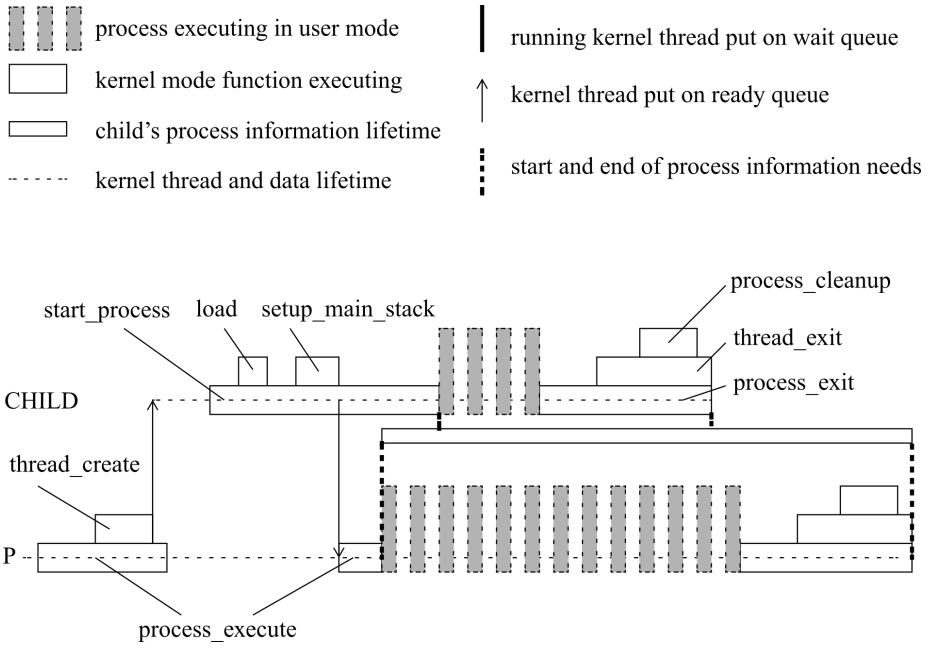

Figure 1: Process execution flow

5.1 Assignment in detail

Once you have clearly understood the flow of executing a new process in pintos, you must decide how to create a relationship between parent and children.

Remember:

- A parent should have a reference to its children
- Every child should have a pointer to its parent
- You may allocate your data structures when the child is spawned and deallocate it when both the parent and the child have exited
- 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 receives the return value to check whether it was successful or not

6 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)

7 Acknowledgement

Part of this document contains material from the LiU TDIU16 course and previous course instructions from the web page.