This document is not complete. Materials are subject to change without notification.

CASYS Master Programming Project

(PART 2) Advanced user-level loader

- Returning to the program loader (developed in Part 1)

The first goal is to make a test program return to the loader. The test program should not execute exit() syscall and should jump back to the loader instead. Your loader will confirm its return and exit with a proper message.

There are two options to handle this task. First, you can modify the test program. You are asked to implement a “return_to_loader” function in a test program. Test programs will be given. You may need to recover a stack pointer of the loader before return to the loader. This task is similar to how your loader loads ELF binary and jump. But, you should care for other registers such as the frame pointer register (e.g., rbp) and segment registers (e.g., cs, gs, fs, …) as well. Otherwise, the loader terminates with a segmentation fault after the test program returns to the loader. The test program explicitly jumps to the loader instead of calling exit() system call.

In the second & advanced option, you are not allowed to modify test programs.

Note that we also have not completed this project yet. We evaluate your approach and critical thinkings instead of done or not. You are free to select one of the options above. You can start with either apager or dpager.

- Back-to-back loading

In the second part of this project, your loader is asked to execute more than two ELF binaries one by one. For example, suppose that two ELF files, A and B, are given to your loader. Then, it executes A to its completion, and A returns to the loader, the loader loads the second ELF, B. B also needs to return to the loader. After your loader executes all ELF binaries, it terminates. To simplify the task, you can assume that the test programs do not have command-line arguments. For example, the loader will run with the command below:

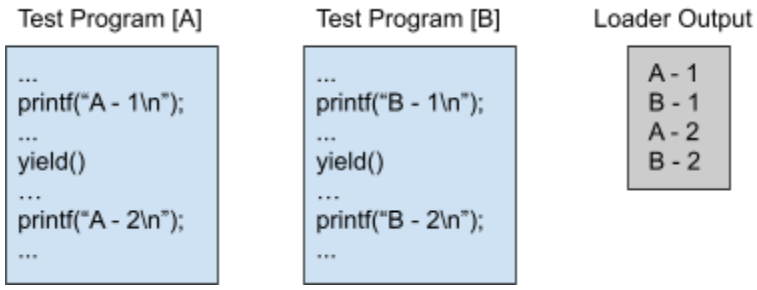
```
./loader [test1] [test2] … [testn]
```

Your loader should be able to take any number of test programs and run them one by one.

- User-level threading

In this part, you are asked to execute multiple test programs at the same time. Instead of using pthread, your loader will schedule test programs in a round-robin manner. The test programs

should call the “yield” function to return to the loader in the middle of execution and the loader may schedule another test program. Test programs are given for this task. Although it is called user-level threading, it is actually a single-threaded execution. The loader will load the multiple test programs at the same time and schedule them one by one until the test program meets the yield function.



[Figure 1. Example]

Figure 1 shows the example of scheduling two test programs with the loader. The test program [A] first runs until it meets yield(), then, the next test program [B] starts to run. When the test program [B] sees yield(), it returns to the loader and the loader will run the next test program, in this case. [A].

For this purpose, setjmp and longjmp system calls can be used. The materials below would be helpful to understand how setjmp and longjmp work.

- http://www.csl.mtu.edu/cs4411.ck/common/Coroutines.pdf
- http://web.eecs.utk.edu/~huangj/cs360/360/notes/Setjmp/lecture.html