Programming Assignment 3

CS450 Spring, 2023

This assignment is a pair programming effort. It is due on March 31, 2023
You have until this Friday, March 10 to sign up your team on a spreadsheet. After that Yun and Harrison will pair you up with a partner.

Part 1: Memory leaks and tools to find them (xv6 not required)

Memory leaks degrades system performance over time and may eventually lead to system crash. The problem happens often and is difficult to detect and correct. The purpose of this exercise is to introduce you to some tools that may help you combat this problem.

In this exercise, you will need to use the debugging tools `gdb` and `valgrind`. `valgrind` helps you to find memory leaks and other insidious memory problems. Please find the following link to download and install the tool:

http://valgrind.org/downloads/current.html

Deliverables of Part 1:

1. (15%) Write a program that allocates memory using `malloc()` but forgets to free it before exiting. What happens when this program runs? Can you use `gdb` to find any problems with it? How about `valgrind` (with the command: *valgrind --leak-check=yes null*)?
2. (5%) Create other test cases for `valgrind`. Explain why you choose them and the expected results.

Part 2: System calls on memory management.

1. (20%) Develop the `whereIs()` system call on xv6. The input argument to `whereIs()` is a virtual address. `whereIs()` returns the physical frame number (PFN) of the page containing its physical address or an appropriate error condition (e.g. is the address valid?). You will provide a test program that will call `whereIs()` with various test data. xv6 has similar code. Because of the difference in purpose, your code is not the same, but understanding the code in xv6 will help.
2. (30%) Develop three system calls `isWritable()`, `notWritable()` and `yesWritable()` on xv6. All three system calls take a virtual address returned from `malloc()`. `isWritable()` returns `True` if the page containing the address is writable, `False` otherwise. `notWritable()` will protect the page from being written into in the future, and `yesWritable()` will, yes, allow the page to be written to. They return the appropriate error conditions.

Tips for Part 2:

1. You want to understand how xv6 uses a three-level page table to map a process's virtual memory to physical memory. In particular, understand what the different bits in a Page Directory Entry and Page Table Entry mean. Chapter3 For 2022-book-riscv-rev3 and RISC-V privileged architecture manual. are useful references. For example, how does a valid Page Table Entry differ from an invalid one?
2. The code in vm.c of xv6 is most relevant to this assignment.

What you will submit (only one submission per team):

Part 1:

(1) Source and executables of the test programs. A `readme` on how to build and execute them with the tools.

(2) (5%) Screen shoots of test runs. A document (5 pages or less) to describe the results of the test runs and address the deliverables. If you use equivalence partitioning, explain your partitions.

Part 2:

(3) Source and executables for the system calls and test programs with a `readme` on how to build and execute them.

(4) (10%) A document (5 pages) that describes the design of the system calls including their manual pages. Describe the changes that you made to the xv6 memory management code and why. You do not need to describe xv6 changes to implement the system calls; that was done in PA2.

(5) (15%) A document (3 pages or less) that describes your test programs and test data. Explain why you use only those test cases. If you use the equivalence partitioning method, describe your partitions.

(6) Submit screen shots of different test runs.

Both Parts:

(7) Upload all files and folders as a **zip** archive as GroupID_PA3.zip. Documents and readme only supports: txt, doc, docx and pdf format.

(8) Write down the names and CWID of team members in all documents and source files.

Grading standard:

(9) In general, we give 65% of the points to working and full featured code, 20% to high quality test data and 15% for well written documents. We give extra credits (maximum 10 points) to very well written code with additional useful and original features, very high-quality test data and documents. The test drivers are coding.

(10) When we have a better feel of what is considered difficult (or easy) to the students and what demands more work, we adjust the points allocate to the features.