**Programming Assignment 3**

CS450 Spring, 2018

This assignment is a pair programming effort. It is due on 03/28/2018

**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*](http://valgrind.org/downloads/current.html)

**Deliverables of Part 1:**

1. 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. Create a program that allocates an array of integers called *data* of size 100 using malloc, frees them, and then tries to print the value of one of the elements of the array. Does the program run? What happens when you use valgrind on it?

**Part 2: System call to display memory usage of a process**

Like most of the operating systems, xv6 uses page tables to maintain the mapping between the virtual address space of a process and the physical memory of the system. This allows xv6 to multiplex the address spaces of different processes onto a single physical memory, and to encapsulate the memory spaces of the different processes.

Each process has a separate page table, and xv6 tells the page table hardware to switch page tables when xv6 switches between processes. When a process asks xv6 for more memory, xv6 first finds free physical pages to provide the storage, and then adds Page Table Entries to the process’s page table that point to the new physical pages.

Sometimes a system programmer may need to find out the physical memory space occupied by a user process. This exercise asks you to develop a tool to help her.

**Deliverables of Part 2:**

Create a new system call, myMemory(),on the MIT xv6 (<https://github.com/mit-pdos/xv6-public>) myMemory() displays the current physical memory usage of the process calling it, in terms of number of pages allocated to the process. Further display the number of pages that are accessible and writable by user program. Make sure you write several test programs which tries to allocate and free varying amounts and types of memory (until no more memory can be allocated and freed) and record how memory allocation affects the page table of the process. Your modifications must not prevent xv6 from functioning normally.

**Tips for Part 2:**

1. Make sure you understand how xv6 uses a page directory and page tables 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. Chapter 2 of the [**xv6 textbook**](https://pdos.csail.mit.edu/6.828/2012/xv6/book-rev7.pdf) is a useful reference. How does a Page Table Entry differ for a valid page compared to an invalid page?
2. Have a look at vm.c file of xv6 to understand how page tables are handled in xv6.

**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. Screen shoots of test runs. A document (3 pages or less) to describe the results of the test runs and address the questions.

**Part 2:**

1. Source and executables for the system call and test programs with a readme on how to build and execute them.
2. A document (5 pages or less) that describes the design of the system call including a manual page. 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 call.
3. A document (3 pages or less) that describes your test programs and test data. Explain why do you use only those test cases. If you use the equivalence partitioning method, describe your partitions.

**Both Parts:**

1. Upload all files and folders as a **zip** archive as GroupID\_PA3.zip. Documents and readme only supports: txt, doc, docx and pdf format.
2. Write down the names and CWID of team members in all documents and source files.