Operating System
CS 370
Spring 2019

Project 7

Due Date: 11:59 PM, April 23rd 2019
Must be submitted before deadline

Total Points: 100
Graduate Assistant: Pradip Maharjan
Office Hours: MoWe 12 PM – 2PM
Teaching Assistant: Luis Maya Aranda
Office Hours: TuTh 10 AM – 12 PM

Course Web Page: http://osserver.cs.unlv.edu/moodle/

Objective: Become familiar with the memory system in xv6 and implement lazy page file allocation.

Introduction:

In xv6, new memory pages are allocated to a process as soon as they are requested. This is inefficient because it creates more overhead for the operating system, even if the pages are not used after they are requested. Therefore, many systems use a lazy scheme to only allocate a page once the first access occurs. This delays a significant amount of the page allocation work until the page is actually needed. In this exercise, you will implement lazy page file allocation in xv6.

Resources:

The following resources provide more in-depth information regarding **xv6**. They include the **xv6** reference book, source code (pdf), and a tutorial on running and debugging.

1. **xv6** Reference Book: https://pdos.csail.mit.edu/6.828/2016/xv6/book-rev9.pdf

   xv6 Source Code PDF: https://pdos.csail.mit.edu/6.828/2016/xv6/xv6-rev9.pdf

   Running and Debugging Tutorial: http://zoo.cs.yale.edu/classes/cs422/2010/lec/l2-hw

Project:

Complete the following steps.

- Become familiar with memory management in xv6
- Modify xv6 to implement lazy page file allocation

Submission

When complete, submit:

- Zip the entire **xv6** folder and submit (not qemu).
- A copy of the ***zipped xv6 folder*** via the class web page (assignment submission link) by class time (1:00 PM).

Submissions received after the due date/time will not be accepted.

<u>Background</u>

Before implementing lazy page file allocation, it is important to familiarize yourself with the existing memory system. First, review the process entry in proc.h. Note that the page directory (the per-process page table), pgdir, and the size of the process's memory, sz, are stored in each process's entry in the process control block. In the existing system, a user program can request more memory for its program by calling malloc() which calls sbrk() (implemented in sysproc.c as sys_sbrk()) which then calls growproc(). See the implementation of sbrk() in sysproc.c and the implementation of growproc() in proc.c. The function growproc() in proc.c calls allocuvm() to allocate n bytes to the calling process's memory. Familiarize yourself with the implementation of allocuvm() in vm.c. The function allocuvm() allocates all the pages up until the new memory size is reached. The exec() function also calls allocuvm() to load a new process, see exec.c.

<u>Modify sbrk()</u>

When sbrk() is called, it calls growproc(), which in turn calls allocuvm() which will allocate all the pages requested immediately. To switch to lazy page allocation, our first task is to remove the call to growproc() from sys_sbrk() in sysproc.c. Instead, have sbrk just increment the size of the process's size by n bytes, and save the allocation for later.

If you run xv6 with this change alone, you will get a page fault error when you run a user program. Try it. You can use traps.h to check what the trap error corresponds to.

<u>Catch Page Faults and Allocate</u>

Recall from an earlier assignment that trap errors are generated in trap.c. You can test the type of trap that has occurred and see if it is a page fault by checking the **trapno**. Use the code that allocates a page present in allocuvm() to allocate the page that the page fault occurred on. You can use the function ***rcr2()*** (read control register 2, see CR2 in x86 architectures for more information) to get the address that caused the page fault. Note that you will have to call mappages(), which is not defined in proc.c. Add the function prototype to the beginning of trap.c to allow the program to find the function:

int mappages(pde_t *pgdir, void *va, uint size, uint pa, int perm);

<u>Modify copyuvm() to make fork() functional</u>

The fork() system call uses copyuvm() in vm.c to copy the parent's heap memory. However, a parent may not have allocated all of the pages that it has requested at the time of fork(). This will cause a page not present error when forking. Modify copyuvm() in vm.c so that it skips copying pages that are not present. *Hint: PTE_P bitwise OR'ed with the page table entry address to determine if a page is present.*

<u>Modify mappages() to Skip Present Pages</u>

Currently, the mappages() function will cause a panic when a page is already present. Remove the panic in mappages() to allow your benchmark program to function.

<u>Testing Your Program</u>

Once you have all of the above changes made, test your program using all of the programs you have created this semester. Xv6 should be functional with all of these programs.