CS4411 Project 4. Virtual Memory(VM)

Due March. 12 11:59 PM

Purpose

The purpose of this project is to understand xv6 OS virtual address mapping and the page fault handler. You will implement 1) a system call to return the physical memory address for a virtual memory address, and 2) lazy memory allocation for the heap.

Project summary

1) Implement a system call to return the physical memory address for a virtual memory address
2) Implement the lazy memory allocation for the heap
3) Add the test case code to XV6 to test the new system call.
4) Submit the report and your XV6 files to Canvas

XV6 files

For this project, you will need to understand the following files: defs.h, kalloc.c, mmu.h, syscall.c, syscall.h, sysproc.c, trap.c, user.h, usys.S, vm.c.

1) The files sysproc.c, syscall.c, syscall.h, user.h, usys.S link user system calls to system call implementation code in the kernel.
2) The file defs.h acts as the header file for several parts of the kernel code.
3) The file trap.h contains trap handling code, including page faults.
4) The file mmu.h contains various definitions and macros pertaining to virtual address trans- lation and page table structure.
5) The files vm.c and kalloc.c contain most of the logic for memory management in the xv6 kernel.

More details for these files are discussed in Chapter 2 and Chapter 3 in the book "xv6 a simple, Unix-like teaching operating system", which is a good resource for this project.

Task 1: The system call to return a physical address

You will implement the system call to return the physical address for a given logical address.

vaddr2phyaddr(vaddr).
void * vaddr2phyaddr(void *vaddr)

The test case code “p4test.c” is provide and you need integrate it to XV6, which is similar the integration of the case code in Project 3.

Task 2: Lazy memory allocation

Most modern operating systems perform lazy allocation of heap memory, though vanilla xv6 does not. Xv6 applications ask the kernel for heap memory using the sbrk() system call. For example, this system call is invoked when the shell program does a malloc to allocate memory for the various tokens in the shell command. In the original xv6 kernel, sbrk() allocates physical memory and maps it into the processs virtual address space. In this lab, you will add support for this lazy allocation feature in xv6, by delaying the memory requested by sbrk() until the process actually uses it.

You can do this task in the following steps.

1) Eliminate allocation from sbrk().
   The sbrk(n) system call grows the processs memory size by n bytes, and then returns the start of the newly allocated region (i.e., the old size). Your first task is to delete page allocation from the sbrk(n) system call implementation, which is in the function sys sbrk() in sysproc.c. We have provided a patched sysproc.c file as part of this lab; you may use this file for eliminating this memory allocation. Your new sbrk(n) will just increment the processs size by n and return the old size. It does not allocate memory, because the call to growproc() is commented out. However, it still increases proc->sz by n to trick the process into believing that it has the memory requested. With the patched code in sysproc.c, boot xv6,and type “echo hi” to the shell.
   You should see something like this:

   ```
   init: starting sh
   $ echo hi
   pid 3 sh: trap 14 err 6 on cpu 0 eip 0x12f1 addr 0x4004--kill proc
   ```

   The “pid 3 sh: trap...” message is from the kernel trap handler in trap.c; it has caught a page fault (trap 14, or T PGFLT), which the xv6 kernel does not know how to handle. Make sure you understand why this page fault occurs. The “addr 0x4004” indicates that the virtual address that caused the page fault is 0x4004.

   Before you work on this task, you need to replace the original sys_sbrk(void) with the modified one in the modified sysproc.c, which is available in the project package. The modified sys_brk( ) supports the lazy memory allocation for the heap.

2) Lazy Allocation.
   Modify the code in trap.c to respond to a page fault from user space by mapping a newly-allocated page of physical memory at the faulting address, and then returning

back to user space to let the process continue executing. That is, you must allocate a new memory page, add suitable page table entries, and return from the trap, so that the process can avoid the page fault the next time it runs. Your code is not required to cover all corner cases and error situations; it just needs to be good enough to let the shell run simple commands like echo and ls.

Some helpful hints:
• You can check whether a fault is a page fault by checking if tf->trapno is equal to T PGFLT in trap().
• Look at the cprintf arguments to see how to find the virtual address that caused the page fault.
• Reuse code from allocuvm() in vm.c, which is what sbrk() calls (via growproc()).
• Use PGROUNDDOWN(va) to round the faulting virtual address down to the start of a page boundary.
• Once you correctly handle the page fault, do break or return in order to avoid the cprintf and theproc->killed = 1statements.
• If you think you need to call mappages() from trap.c, you will need to delete the static keyword in the declaration of mappages() in vm.c, and you will need to declare mappages() in trap.c. An easier option would be to write a new function to handle page faults within vm.c itself, and call this new function from the page fault handling code in trap.c.

If all goes well, your lazy allocation code should result in "echo hi" working. I will only test your code to check that simple shell commands are executing properly, so you need not worry about various other corner cases in your implementation.

Submission

1) Submit **a report to Canvas.**

   a) Explain what modifications you did to implement the address translation with the code screenshots. In addition, the screenshot for the test case p4test is needed.
   b) Explain what modifications you did to implement the lazy memory allocation with the code screenshots.

2) Remove the object files, binary files and the FS image file

```
make clean
```

   Issue the above command when you are under the xv6-public folder.

3) Compress the xv6 folder

```
tar  cvfz  xv6-your-last-name.tar.gz   xv6-public/*
```

Issue the above command when you are outside of the xv6-public folder. Note that only files under the xv6-public are include the archive and do not put the sub-folder to it. Otherwise the size of archive is 18MB.

4) Upload the xv6-your-last-name.tar.gz to Canvas

Credit

The task 2 is based on Mythili Vutukuru’s assignment.