CS/SE 4348 S19: Operating Systems

Exercise: Delayed Page Allocation in xv6

Due date 6th March, 2019

Read the description below carefully. This is more like an exercise than a project. This exercise should be done **individually**. You cannot share your work with anyone. This exercise can be done only in **csjaws.**

Delayed Page Allocation

One of the many neat tricks an OS can play with paging is delayed allocation of heap memory. On xv6, applications ask the kernel for heap memory using the sbrk() system call. In the kernel given to you, sbrk() allocates physical memory and maps it into the process's virtual address space. There are programs that allocate memory but never use it, for example to implement large sparse arrays. Sophisticated kernels delay allocation of each page of memory until the application tries to use that page—as signaled by a page fault. You'll add this delayed allocation feature to xv6 in this exercise

Eliminate allocation from sbrk()

Your first task is to delete page allocation from the sbrk(n) system call implementation, which is function sys_sbrk() in sysproc.c. The sbrk(n) system call grows the process's memory size by n bytes, and then returns the start of the newly allocated region (i.e., the old size). Your new sbrk(n) should just increment the process's size (proc->sz) by n and return the old size. It should not allocate memory—so you should delete the call to growproc(). Try to guess what the result of this modification will be and what will break.

Make this modification, boot xv6, and type echo hi in 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.

Implementing delayed 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. Add your code before the cprintf call that produced the “pid 3 sh: trap 14” message. Your code is not required cover all corner cases and error situations; it just needs to be good enough to let sh run simple commands like echo and ls.

- You can check whether a fault is a page fault by checking if tf->trapno is equal to T_PGFLT in trap().
- Find the virtual address that caused the page fault from the cprintf arguments.
- Use PGROUNDDOWN(va) to round the faulting virtual address down to a page boundary.
- Call or steal code from allocuvm() in vm.c, which is what sbrk() calls (via growproc()).
- Break or return in order to avoid the cprintf and the proc->killed = 1.

If all goes well, your delayed allocation code should result in echo hi working. You should get at least one page fault (and thus lazy allocation) in the shell, and perhaps two.

xv6 Source code

The xv6 source code for this project is /cs4348-xv6/src/xv6.tar.gz. You do not need any of the code that you implemented in Project 3a. Copy this file to your local working directory and extract the source code tree using the command

```
tar –zxvf xv6.tar.gz
```

Submission

Copy your modified trap.c source file to the directory ‘/cs4348-xv6/xxxyyyyyyy/E2’.