Operating Systems Project #3 - xv6 Lazy Allocator

SWE3004-42 Introduction to Operating Systems - Spring 2018

Due date: May 27 (Sun) 11:59pm

1 Goal

In this project, you will implement a lazy memory allocator for the heap of a user process.

2 Details

One of the many neat tricks an O/S can play with page table hardware is lazy allocation of heap memory. Xv6 applications ask the kernel for heap memory using the `sbrk()` system call. In the kernel we've given 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 lazy allocation feature to xv6 in this exercise.

3 How to Start Project

Download xv6 source code by running the following git command. Note that we are getting the code from a different GitHub repository this time.

```
$ git clone https://github.com/moyix/xv6-public.git
```

4 Part 1

Your first task is to delete page allocation from the `sbrk(n)` system call implementation, which is the 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()` (but you still need to increase the process's size!). Try to guess what the result of this modification will be: what will break?

Make this modification, 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 and the program counter (`eip`) is 0x12f1. It then kills the process by setting `proc->killed` to 1.

5 Part 2

Modify the code in trap.c to recognize this particular exception and respond to a page fault from user space by mapping the page on demand at the faulting address, and then returning back to user space to let the process continue executing. You should add your code just before the `cprintf` call that produced the "pid 3 sh: trap 14" message. If you have implemented the allocation, running echo command should work normally. Your code is not required to cover all corner cases and error situations; it just needs to be good enough to let sh run simple commands like echo and ls.

Hints:

- Look at the `cprintf` arguments to see how to find the virtual address that caused the page fault.
- The virtual address of the address that triggered the fault is available in the `cr2` register. xv6 provides the `rc2()` function to read its value.
- Look at the code `allocuvm()` in vm.c to figure out how to allocate a page of memory and map it to a specific user address. It is what `sbrk()` calls (via `growproc()`).
- Remember that the first access might be in the middle of a page, and so you'll have to round down to the nearest PGSIZE bytes. You can use `PGROUNDDOWN(va)` to round the faulting virtual address down to a page boundary.
- Break or return in order to avoid the `cprintf` and the `proc->killed = 1`.
- You'll need to call `mappages()`, which is declared as static, meaning it can't be seen from other C files. You'll need to delete the static in the declaration of `mappages()` in vm.c, and you'll need to declare `mappages()` in defs.h file that is included by both trap.c and vm.c. Add this declaration before any call to `mappages()`:

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

- You can check whether a fault is a page fault by checking if `tf->trapno` is equal to `T_PGFLT` in `trap()`. `T_PGFLT` is defined in traps.h and corresponds to the exception number for a page fault.

If all goes well, your lazy 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.

By the way, this is not a fully correct implementation. See the challenges below for a list of problems we're aware of.

6 How to Submit

To submit your project, you must run `make clean` command in xv6-skku directory to delete all executables and object files. To compress your source codes into one .tar.gz file, please run the following command.

```
$ make clean
$ tar czvf project3-2010310123.tar.gz ../xv6-public
```

This .tar.gz file must be uploaded to the iCampus assignment submission menu.

For any questions, please post them in Piazza so that we can share your questions and answers with other students and TAs. Please feel free to raise any issues and post any questions. Also, if you can answer other students' questions, you are welcome to do so. You will get some credits for posting questions and answering other students' questions.

7 Late Submission Policy

Late submissions will be accepted but with 20% penalty per day. That is, if you submit 2 days late and test score is 80 points, your penalized score will be 48 points (80 x 60%).

8 Need Volunteers to help other students

Since OS projects can be very challenging to some of you, I would give some credits to volunteer students for helping other students. If you are willing to volunteer or you like to get some help from other students, please send me an e-mail. If you think you deserve A but not sure about A+, I encourage you to volunteer.