Operating Systems Project #3 - xv6 Stack

SWE3004-42 Introduction to Operating Systems - Fall 2018

Due date: November 23 (Fri) 11:59pm

1 Goal

In this project, you will modify memory layout to move the stack to top of address space so that stack can grow.

2 Details

In xv6, the VM system uses a simple two-level page table. If you do not remember the details, please read Section 20.3 of OS three easy steps. You may also find the description in Chapter 1 of the xv6 manual (https://pdos.csail.mit.edu/6.828/2014/xv6/book-rev8.pdf) sufficient (and more relevant to the assignment).

The xv6 address space is currently set up like this:

```
code
stack (fixed-sized, one page)
heap (grows towards the high-end of the address space)
```

In this project, you'll rearrange the address space as as the following:

```
code
heap (grows towards the high-end of the address space)
... (gap)
stack (at end of address space; grows backwards)
```

You can see the general map of the kernel memory in memlayout.h; the user memory starts at 0 and goes up to KERNBASE. Note that we will not be changing the kernel memory layout at all, only the user memory layout

Right now, the program memory map is determined by how we load the program into memory and set up the page table (so that they are pointing to the right physical pages). This is all implemented in exec.c as part of the exec system call using the underlying support provided to implement virtual memory in vm.c. To change the memory layout, you have to change the exec code to load the program and allocate the stack in the new way that we want.

Moving the stack up will give us space to allow it to grow, but it complicates a few things. For example, right now xv6 keeps track of the end of the virtual address space using one value (sz). Now you have to keep more information potentially e.g., the end of the bottom part of the user memory (i.e., the top of the heap, which is called brk), and bottom page of the stack.

Once you figure out in exec.c where xv6 allocates and initializes the user stack; then, you'll have to figure out how to change that to use a page at the high-end of the xv6 user address space, instead of one between the code and heap.

Some tricky parts: one thing you'll have to be very careful with is how xv6 currently tracks the size of a process's address space (currently with the sz field in the proc struct). There are a number of places in the code where this is used (e.g., to check whether an argument passed into the kernel is valid; to copy the address space). We recommend keeping this field to track the size of the code and heap, but doing some other accounting to track the stack, and changing all relevant code (i.e., that used to deal with sz) to now work with your new accounting. Note that this potentially includes the shared memory code.

The final item, which is challenging: automatically growing the stack backwards when needed. When the stack grows beyond its allocated page(s) it will cause a page fault because it is accessing an unmapped page. If you look in traps.h, this trap is `T_PGFLT` which is currently not handled in our trap handler in trap.c. This means that it goes to the default handling of unknown traps, and causes a kernel panic.

So, the first step is to add a case in trap to handle page faults. For now, your trap handler should simply check if the page fault was caused by an access to the page right under the current top of the stack. If this is the case, we allocate and map the page, and we are done. If the page fault is caused by a different address, we can go to the default handler and do a kernel panic like we did before.

3 Template Code

You should download project template by running the following command in our cluster.

```
$ mkdir proj3
$ cd proj3
$ wget http://csl.skku.edu/uploads/SWE3004S16/xv6-project-3-template.tar.gz
$ tar czvf xv6-project-3-template.tar.gz
```

4 How to Submit

To submit your project, you must run `make tarball` command in xv6-skku directory to compress your source codes into one .tar.gz file. This .tar.gz file must be submitted in swin.skku.edu using the following command.

```
$ os_submit project3 your_tar_file_name
```

Note: Please don't forget writing your ID and project number in the Makefile before creating a tarball.