EE488: Unix Kernel Design, Fall 2020

Project 4: Demand Paging

Due: Oct. 19, 2020 11:59 pm (submission: by email)

Submit a report to TA: Joontaek Oh (email: na94jun@gmail.com)

Name your file with “hw4_[your studentid]”. ex: hw4_20201234.c

This assignment will help you understand the mechanism for allocating memory for a process. Furthermore, we will implement a memory allocation policy with better performance by modifying the memory allocation policy currently implemented in xv6.
First, in xv6, the `exec()` allocates memory for the program, loads the program, and stores the mapping information in the page table. At this point, the memory required for program loading is allocated and the data is loaded from the storage. However, the code or data stored in memory is not always used. Therefore, modify the code so that the memory for the program is allocated and loaded at runtime, rather than allocated and loaded before runtime. These modifications can reduce physical memory usage and increase load speed by not actually loading unused data into memory.
Second, the process uses the `sbrk()` system call when it wants to allocate memory to the heap area. `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.
You'll add this Demand Page feature to xv6 in this exercise.

Part One: demand paging for heap segment

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 newly allocated region (i.e., the old size). Your new `sbrk(n)`

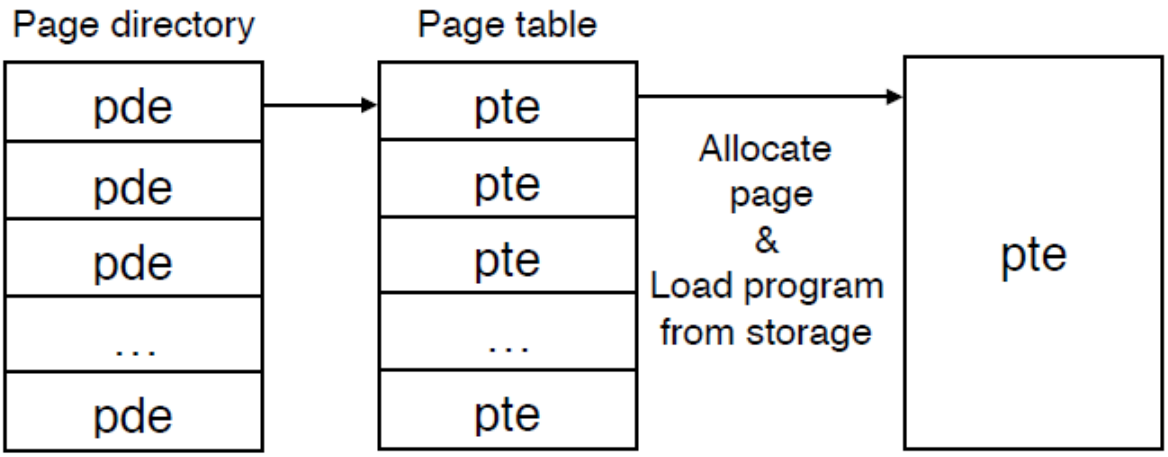
should just increment the process's size (`proc->sz`) by n and return the old size. It should not allocate memory -- so you should delete the call to `growproc()` (but you still need to increase the process's size!).
Try to guess what the result of this modification will be: what will break?
Make this 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`.

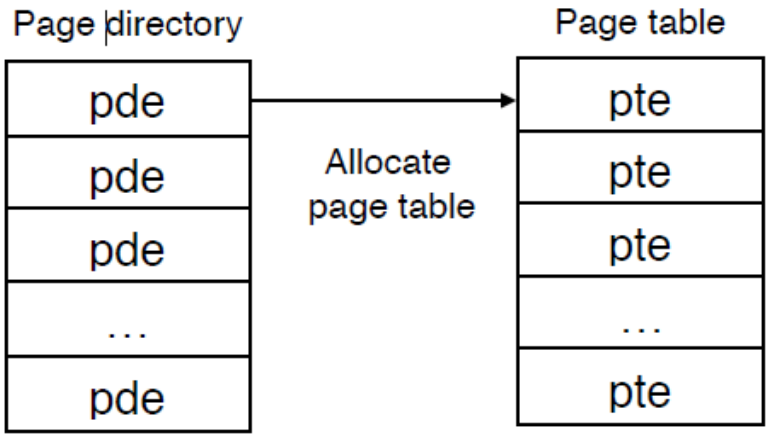
Part Two: demand paging for text segment and data segment

The second task is to modify `exec()` in the `exec.c` file for demand paging. In original xv6, `exec()` system call loads the text segment and the data segment of the binary file when the `exec()` is called. Modify the `exec()` so that it initializes the page table using the information from the header of the ELF binary, but it does not map the page table entries. The actual pages are going to be



Access the actual page

allocated and the associated contents will be read from the disk to memory when it is accessed.



exec

Part Three: struct proc

In proc structure(`proc.h`) file, add the fields to represent the following information.

* file name: including path

* segment information for text: start address and end address of text segment
* segment information for data: start address and end address of data segment
* segment information....

Part Four: trap()

Finally, it is time to add a handler to handle page faults caused by changes in part1 and part2.

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. You should add your code just before the `cprintf` call that produced the "`pid 3 sh: trap 14...`" message.

As mentioned earlier, it is necessary to distinguish whether the address where the page fault occurred is a new allocation address or and address which needs to be loaded from storage after allocation. You must write code to handle each situation.

Hint1 : It is convenient to analyze and use the `mappages()` when mapping the newly allocated page to the page table of the process. To use `mappages()` in the trap.c file, you need to remove the static word of `mappages()` stored in the vm.x file and describe the prototype of the `mappages()` in the trap.c file.

Hint2: It is convenient to analyze and use the functions `allocuvm()` and `loaduvm()` which are the codes used to load the program into memory.

Submit : modified `sysproc.c`, `exec.c`, `trap.c`, `proc.h` file