Page Replacement

Dr. Karim Sobh

Computer Science Department
Jack Baskin School of Engineering
University of California, Santa Cruz

Spring 2017
Assigned: Tuesday 9th May *at 15:00*
Due: Thursday 18th *May at 15:00*

Goals

The primary goal for this project is to experiment with the FreeBSD *pageout daemon* (§6.12 of the FreeBSD book) to modify how it selects pages for replacement, and to see how effective the policy is.

As with Assignment 2, this project will give you further experience in experimenting with operating system kernels, and doing work in such a way that when done incorrectly will almost certainly crash a computer.

Basics

The goal of this assignment is to change the behavior of the *pageout daemon* in FreeBSD, which is responsible for deciding which pages to make available for replacement and which pages to flush back to disk. Obviously, this matters most in a system where there's virtual memory pressure, so you'll probably want to experiment using programs that consume (*slightly*) more memory than your VM has available. You can use a memory testing tool as well.

Details

The current virtual memory algorithm is a relative of the CLOCK algorithm that uses two queues. Pages have an activity count with a maximum value of 64 associated with them. The general rule is that a page that is found to be referenced in both this scan and the previous scan is moved to the next lower queue, and a page that's found to be referenced in neither this pass nor the previous pass is moved to the next higher queue. Pages that are referenced in exactly one of the previous two passes remain in their current queue. Pages in the highest queue that are not referenced and dirty are queued for write (remaining in the highest queue), and pages that are in the highest queue and unreferenced in this scan and the previous scan are moved to cache, much as pages in INACTIVE are moved to cache in the current pageout algorithm. New pages are placed into Queue 0 and migrate to higher-numbered queues as their time between references decreases.

Well, that's all well and good, except you're going to implement the *SLIM CHANCE* algorithm. It's like *SECOND CHANCE*, except it's not. Instead of putting inactive and invalid pages on the front of the free list, you will put them on the *rear*. Instead of subtracting from the activity count, you will *divide it by two* and move it to the *front* of the active list instead of to the rear. Finally, when you move a page to the inactive list, it goes on the *front* instead of the rear.

You'll need to scan more pages during each pageout than it does currently (you should scan all pages in 10 seconds rather than 10 minutes or something) to see much effect; this will also require modifying the kernel.

Most of the VM-relevant code is in `sys/vm`, and the page queues are only referenced in `vm_pageout.c` and `vm_page.c` as well as `vm_page.h`. You may need to modify code elsewhere, but look here first.

As part of the code, pageout should write statistics about its performance to the system log. This should include, for each run of pageout the number of pages in each queue as well as the queues scanned in that run and, for each queue, the number of pages moved to a higher or lower queue. You should also log the number of pages moved to the cache, the number of pages queued for flush etc. You may also get information using the `sysctl()` system call.

As part of the code, pageout should write statistics about its performance to the system log: the number of pages in each queue as well as the queues scanned in that run, the number of pages moved from ACTIVE to INACTIVE and INACTIVE to cache / free, and the number of pages queued for flush. This is done for each run of pageout. We recommend doing this to the existing kernel before modifying it to add your code, and tagging the commit in which this code works. Then, run programs to stress the memory system.

Building the Kernel

As with Assignment 2,

- Try building a kernel with no changes first. That means branch off from the `master` branch, to a branch called `asgn3`. Create your own `KERNCONFIG` file, build the kernel, and boot from it. If you can't do this, it's likely you won't be able to boot from a kernel after you've made changes.
- Make sure all your changes are committed and pushed to server every time before you reboot into your custom kernel. It's unlikely that bugs will kill the file system, but it can happen. Commit anything you care about using `git`, and push your changes to the server before rebooting. "*The OS ate my code*" isn't a valid excuse for not getting the assignment done.

As before, your repository (checked out from `git`) contains all the code you'll need. Don't check out a new version of the source code from elsewhere.

How to submit

Select one team member as the "CAPTAIN". **This person is the only one in whose repository work will be done and only this person's repository will be graded**.

Each team must do the following, if not done so already:

1. The team captain needs to give each team member write access to the repository using a command that looks like this:

```
ssh git@git.soe.ucsc.edu perms classes/cmps111/winter17-02/captain_cruzid + WRITERS team_member_cruzid
```

   This needs to be done **once** for each team member.

2. Each team member (other than the CAPTAIN, who already has a copy) needs to clone the CAPTAIN's repo:

```
git clone git@git.soe.ucsc.edu:classes/cmps111/winter17-02/captain_cruzid
```

You may rename the repo anything you want; it remembers where it came from when you push changes.

As you work on the assignment

Repeat as needed:

- Make changes to/Add one or more files.
- Add one or more modified/new files to the repository, or all modified/new ones:

```
git add file1 file2 ...
git add -A
```

- Commit all your changes to the repository:

```
git commit -am "Your commit message goes here"
```

- As desired, push all of your changes to the `git` server. This includes *all* commits you've already made, not just the most recent one. Of course, only those that are "missing" on the remote side are sent.

```
git push origin <branch_name>
```

Testing your project

Run a few test programs, like stress(1) for example, and see how many **pages get scanned and moved for your code as well as the standard FreeBSD code.** You've got the logs or raw data that lists how many pages are scanned / moved for both your code and the existing FreeBSD pageout daemon. Please write up your findings (text and graphs) and submit them as

WRITEUP.pdf. Therefore, you'll need to first add logging statements to capture the raw data in the unmodified FreeBSD source, test that, then add your code modifications, then test and lastly compare in your Writeup.pdf.

Deliverables

1. In a directory called `asgn3` in the `asgn3` branch, in the root of the captain's repository:
    a. Design Document (PDF or plain text): Your code modifications explained so that an experienced programmer would be able to reproduce your code.
    b. Readme.txt: Should contain any special instructions that we should know.
    c. Writeup.pdf: Should have an analysis and comparison between the unmodified and your pageout mechanisms. Graphs and visualizations are encouraged. It is worth equal to the code modifications in terms of the grade.
    d. Data.csv: Raw data that you obtained from the logs, and organized in a CSV format. Other formats such as Microsoft Excel or Open Document Spreadsheet are also acceptable.
2. Your code modifications:
    a. Modifications to existing source files. Your kernel should build, install and boot successfully, and manage to run reasonably well.
3. eCommons:
    a. Final Git commit ID, referencing the final state of the captain's repository. This must be submitted only by the captain to their eCommons. The submission timestamp recorded by eCommons will be proof that you have created a commit before the deadline.
    b. Each team member's contribution, including the captain as a plain-text (ASCII) file. This would entail a few paragraphs describing what *you* contributed to the group effort, and how you'd rate the other members of your group. The goal here is for us to understand how each person contributed to the group effort. We don't expect everyone to have done the same thing, but we expect that everyone will contribute to the project. This must be submitted individually by each team member to their own eCommons.

Deliverables Summary:

- Design Document (.txt or .pdf) [git]
- Raw Data (.csv or .ods or .xlsx or .xls)
- README.txt [git]
- Source code only. [git]
- Commit Id. [eCommons]
- Your contribution [eCommons]

Hints

- *START NOW*! Meet with your group *NOW* to discuss your plan and write up your design document. design, and check it over with the course staff.

- *EXPERIMENT*! You're running in virtual machine—you can't crash the whole computer (and if you can, let us know…).
- Test your memory allocation. To do this, write a program that uses slightly more than 1/*n* of your virtual memory space, and run *n* copies of it. We can provide or suggest such a program, but please feel free to use your own. This project doesn't require a lot of coding (typically several hundred lines of code), but does require that you understand FreeBSD and how to use basic system calls. You're encouraged to go to the class lab section or talk with TAs during office hours to get help if you need it.

IMPORTANT: As with all the projects this quarter, the key to success is starting early. You can always take a break if you finish early, but it's impossible to complete a 20-hour project in the remaining 12 hours before it's due.