HOMEWORK 7

Page Replacement Algorithms

Issued	4/29/14
Due Date	5/5/14 @ midnight CST

NOTE: As a reminder, homework is considered handed in when it is committed to your personal subversion repository before the deadline. Changes made after the deadline will be ignored.

Goal

For this homework, we implementing some common page replacement algorithms, and evaluate how these perform for a given workload.

Evaluation

The homework will be evaluated using the following factors:

- Code Quality: Your code should compile without warnings or errors
- Documentation of the code -- The code is clearly documented, explaining what the intent is and how it is accomplished.
- Correctness of execution -- The code performs the functions requested.
- Creativity -- Your solution is different. Make sure to **document** your code or we might miss your special trick!

What to hand in

As for all the other homework assignments, you will need to commit your homework to your assigned subversion repository, under the correct directory (`homework/homework7`).

Use EXACTLY the requested file and directory names, including case!

Please commit the following files:
- `repalg.c`, `repalg.h`, `memref.c` and any other files required to build the test program and the mkfs program.
- The `Makefile`
- A text file `DESCRIPTION` containing a description of what your code does, how you achieved it, clearly listing which tasks (see above) you completed.

Is is OK if you commit more files than requested, but make sure the files requested above are present and exactly named as specified. If you add extra C files (header, code) make sure to update your Makefile.

Note: there should be no need to modify `memref.c`.

Details

Environment

Your code needs to compile under your virtual machine (ubuntu). While a part of your code will eventually be used in our own kernel, for the purpose of this homework, all development happens under linux.

A simple `Makefile` is provided. Feel free to enhance this makefile (for example to automate invoking your `mkfs` program or to automate testing.

`repalg.c`

All development should be within `repalg.c`. There should be no need to modify `repalg.h` or to modify the test program (`memref.c`).

A minimal code skeleton is already provided in `repalg.c`.

Task 1: Implement Page Replacement Algorithms

Take a look at `repalg.c`. The file implements some infrastructure for initializing and switching between different algorithms.

Your task is to:

- Implement the second change algorithm;
- Implement the least recently used algorithm.
- (For optional credit) Implement a random strategy.

For the first two, please see the book for a full description of the algorithm. For the last one, the random algorithm chooses a random page *if the requested page is not already mapped*. This algorithm is useful to have a baseline comparison to evaluate the performance (number of pagefaults) of the other two algorithms.

Note: code is already provided for 2 algorithms (`alg1_xxx` and `alg2_xxx` functions). To add a third algorithm, you'll need to create the functions and add them to the `algorithms` array in `repalg.c`.

In any case, make sure to update the name accordingly for each algorithm in the `algorithms` array.

To implement an algorithm, you need to implement 3 functions:

- init
- done
- access

As usual, the first two provide you with a place to allocate memory (if needed) and cleanup, and initialize data structures. The number of physical memory pages is passed to the `init` function as a parameter.

The `access` function takes the requested virtual page number and the type of the access (read or write). It returns the physical page number to be used to hold the contents of the virtual page, and it also sets the `status` variable to indicate if this page was already mapped or if this is a new mapping (and thus a page fault occured).

Please see the code for a more detailed description of the functions.

Task 2: Compare Algorithms

Now that you have implemented (2 or more) algorithms, we want to compare their performance. Using the commandline options to `memref`, run some experiments and note down the parameters in the text file and list the number of page faults for each algorithm.

Is there a difference in performance? Briefly discuss the follow points (5-20 lines total is sufficient, but make sure to answer each question below):

- How do you explain the existence (or absence) of this difference?
- How is the performance affected by the number of page frames available? Why?

Help and hints

- The `valgrind` tool can help you find bugs in your program. http://valgrind.org/ It can be installed using `apt-get` under ubuntu.