Memory Management

March 29, 2016

1 Memory Management

This week, the designing of several memory page replacement algorithms and comparing their effectiveness against certain memory access patterns will be explored. **Please** take a look at the code provided and ask questions if certain things do not make sense. The majority of the code has been completed for you, though it is still recommended to understand how the pieces fit together.

1.1 Setting up the environment

We first need to setup the enviornment we are working in. Since this is a fairly spread out lab and setting up that enviornment takes several steps we can group all of that into one operation. Note that this operation is only good in the terminal that it is run in. Each time you open a new terminal to work on the project you will have to run this following line **in the root directory of the project.**

```
$ source source.me
```

It will print out a message about "is a directory" which can be safely ignored.

1.2 Compiling the libraries and classes

This lab uses one library and contains a few classes, each of which need to be compiled and the library needs to be installed.

1.2.1 Linked List

The linked list library provides a full featured linked list with itterators. This libary is used extensively throughout the entire project and should be the first library installed. For documentation on the linked list libary look at `include/llist/isu_llist.h` after installing.

To compile and install the linked list library from the root directory of the project run

```
$ cd lib/llist
$ make
$ make install
```

1.2.2 Compiling the memory request class

Once the environment is set up, and the library installed, the memory request class can be compiled. The memory request class is needed so that there can be a way to provide the algorithms with a memory request and a way to keep track of whether or not a memory request was a hit or a miss. Also this class helps with the book keeping of the data in the pages for data logging. To compile the memory request class, run the commands:

```
$ cd page_req/
$ make
```

1.2.3 Compiling the Memory Management Unit

Now that the memory request class is compiled, the memory management unit(or MMU) class can be worked on. The memory management unit class is in the folder `isu_mmu` and it is where all the memory page management happens in this lab, meaning this is where the implementations of the page replacement algorithms will reside. The FIFO replacement algorithm has been implemented as an example. Note that there are some TODOs in the code that are in the functions `isu_mmu_page_rep_lru`, `isu_mmu_page_rep_clock`, and `isu_mmu_page_rep_second_chance`. These are for the students to implement. Follow the FIFO example on the general flow of what the algorithms should do. Once the changes are made, to compile the MMU run the commands:

```
$ make
```

This will generate an object file that will be used in the compilation of the test program.

1.3 Compiling the test program

With the MMU and memory request classes compiled, the MMU can now be tested. A testing program `mem_test.c`, in the root folder of the lab, is provided. To compile this program, run the command:

```
$ make
```

This would compile the previously mentioned pieces and the `mem_test` program. The program takes two arguments, both are integers. The first argument is the algorithm to run, and the second argument is the memory access pattern. For example, to run the program to test the FIFO page replacement algorithm with the sequential memory access pattern, run the program as follows:

```
$ ./mem_test 0 0
```

This will generate a log file named `fifo-seqt.log` that shows the working set at each memory page request, which page was requested, and whether or not it was a hit.

To see the full list of arguments for the `mem_test` program, run `mem_test` without arguments.

1.4 Tasks for this lab

Your tasks for this lab are to implement the Least Recently Used, and Clock memory page replacement algorithms. The expected outputs from each replacement algorithm and memory access pattern are given in the `answers` directory to compair with. *note, I can't garuntee that the answers are correct, I finished them up at 5AM and have not verfied them.* Comment in your lab report the outputs of each algorithm and how they compare with each other. Is it what you would expect? Which do you consider the best?

1.5 Going futher

After you have finished implementing the page replacement algorithms you should now play around with the numbers to see how it effects the hit rate. In the `isu_mmu.c` file, at the top of the file there is a define for the number of pages in the system. Currently, it is defaulted to 8. Play around with this value to see how the hit rate changes, and maybe find an optimal value of pages independent of the algorithm. Comment on what you did and your observations on the effects in your lab report.

2 Extra Credit

For extra credit you can implement your own page replacement algorithm. Feel free to get as creative as you would like on this and follow the given examples to get started. One option is to implement the Second Chance algorithm.

3 License