Assignment 4: Virtual Memory Simulation

CSC 139 Operating System Principles - Spring 2020

Posted on April 26, due on May 5 (11:59 pm)

1 Objectives

This programming assignment is to simulate a simple virtual memory system using the three page replacement policies studied in class: First-In First-Out (FIFO), Optimal, and Least Recently Used (LRU). Your program reads an input file containing a sequence of page requests (page numbers) and generates an output that shows how the requested pages are mapped into physical frames.

2 Description

The page replacement algorithms to implement in this assignment are First-In First-Out (FIFO), Optimal, and Least Recently Used (LRU). The detailed algorithms are already described in class slides and textbook Chapter 10.

2.1 Task Information

The page request information will be read from an input file. The first line in the input file has three integers. The first integer is the number of pages, the second integer is the number of frames, and the third integer is the number of page access requests. The remaining lines in the input file are page access requests (page numbers), with each page request appearing on a separate line. In an interesting input, the number of frames is less than the number of pages; otherwise, the problem is trivial. Furthermore, in an interesting input, some pages are requested multiple times (the same page number appears multiple times in the sequence). Because the number of frames is typically smaller than the number of pages, the same page may be mapped to a different frame each time it is requested.

In your simulation, assume *pure demand paging*, that is, pages are loaded into physical frames only when they are requested. So, initially, no pages are loaded, and all frames are free. If the number of frames is x, the first x pages accessed will be trivially mapped to the x frames without having to replace any pages. When a request to access the (x+1)-th page arrives, your program must find one of the x pages that have been loaded into physical frames and replace it with the new page. The selection of the page to be replaced depends on the page replacement policy.

- In the FIFO policy, the first page loaded into a physical frame is selected for replacement.
- In the Optimal policy, the page that will not be accessed for the longest time in the future is selected for replacement. This policy may be implemented by associating with each page table entry a number indicating its next-use time in the future. When replacement is needed, replace the page with the greatest next-use number. If a page is not referenced in the future, set its next-use time to INFINITY.

If there are multiple pages with infinite next-use times, replace the page that is currently in the smallest frame number.

- In the LRU policy, the page that has not been accessed for the longest time is selected for replacement. This policy may be implemented by associating with each page table entry a time stamp indicating the latest time at which the page was accessed. When replacement is needed, replace the page with the smallest time stamp.

The output will have one line for each page request, indicating how that page request is handled. The last line for each algorithm reports the total number of page faults. You may write your implementation in C.

2.2 Command-line Usage

```
Usage:  proj4 input_file [FIFO|LRU|OPT]
```

where `input_file` is the file name with page request information. `FIFO`, `LRU`, and `OPT` are names of page replacement policies.

2.3 Sample Inputs and Outputs

Sample input file and expected outputs are shown below. You can use it to verify your results.

Example: Here is an example input file and the corresponding outputs. The input has 8 pages, 4 frames, and 12 page requests.

Input:

```
% more input1.txt
8  4  12
4
3
4
6
1
6
4
5
2
4
6
1
```

Output:

(FIFO)

```
Page 4 loaded into Frame 0
Page 3 loaded into Frame 1
Page 4 already in Frame 0
Page 6 loaded into Frame 2
Page 1 loaded into Frame 3
Page 6 already in Frame 2
Page 4 already in Frame 0
Page 4 unloaded from Frame 0, Page 5 loaded into Frame 0
Page 3 unloaded from Frame 1, Page 2 loaded into Frame 1
Page 6 unloaded from Frame 2, Page 4 loaded into Frame 2
Page 1 unloaded from Frame 3, Page 6 loaded into Frame 3
Page 5 unloaded from Frame 0, Page 1 loaded into Frame 0
9 page faults
```

(Optimal)

```
Page 4 loaded into Frame 0
Page 3 loaded into Frame 1
Page 4 already in Frame 0
Page 6 loaded into Frame 2
Page 1 loaded into Frame 3
Page 6 already in Frame 2
Page 4 already in Frame 0
Page 3 unloaded from Frame 1, Page 5 loaded into Frame 1
Page 5 unloaded from Frame 1, Page 2 loaded into Frame 1
Page 4 already in Frame 0
Page 6 already in Frame 2
Page 1 already in Frame 3
6 page faults
```

(LRU)

```
Page 4 loaded into Frame 0
Page 3 loaded into Frame 1
Page 4 already in Frame 0
Page 6 loaded into Frame 2
Page 1 loaded into Frame 3
Page 6 already in Frame 2
Page 4 already in Frame 0
Page 3 unloaded from Frame 1, Page 5 loaded into Frame 1
Page 1 unloaded from Frame 3, Page 2 loaded into Frame 3
Page 4 already in Frame 0
Page 6 already in Frame 2
Page 5 unloaded from Frame 1, Page 1 loaded into Frame 1
7 page faults
```

3 Deliverables

Make sure your code can be compiled and work on Athena server correctly. Upload to Canvas the following:

- All source codes/files that you have added/modified and the demonstrative results.
- A `README.TXT` file that briefly describes each file, how to compile the file(s), and how to run the file.
- These files should be placed in a directory called `<ECS username>-asgmt4`.
- Use tar command to place all the files in a single file called `<ECS username>-asgmt4.tar`. Assuming you are in the directory `<ECS username>-asgmt4` do the following:

- Goto the parent directory: `cd ..`
- tar the files:
  `tar -cvf <ECS username>-asgmt4.tar ./<ECS username>-asgmt4`
- Verify the files have been placed in a tar file:
  `tar -tvf <ECS username>-asgmt4.tar`
- Compress the files using gzip: `gzip <ECS username>-asgmt4.tar`
- Verify that the gzipped file exists: `ls <ECS username>-asgmt4.tar.gz`