Assignment 4: Virtual Memory Simulation

CSC 139 Operating System Principles - Fall 2021

Posted on Nov. 29, due on Dec. 9 (11:59 pm)

1 Objectives

This programming assignment is to simulate a simple virtual memory system using the two page replacement policies studied in class: Least Recently Used (LRU), and Second-chance (clock). 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 Least Recently Used (LRU), and Second-chance (clock). 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. Note that all indexes start from 0.

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 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.
- In the Second-chance (clock) policy, we inspect the selected page's reference bit. If the value is 0, we proceed to replace the page; but if the reference bit is set to 1, we give the page a second chance and

move on to select the next FIFO page. When a page gets a second chance, its reference bit is cleared, and its arrival time is reset to the current time. Thus, a page that is given a second chance will not be replaced until all other pages have been replaced (or given second chances). One way to implement the policy is as a circular queue. A pointer indicates which page is to be replaced next. When a frame is needed, the pointer advances until it finds a page with a 0 reference bit. Second-chance policy degenerates to FIFO policy if all bits are set. (See more details in Chapter 10.4.5.2 and page 29 in the slides.)

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.

2.2 Command-line Usage

`Usage: proj4 input_file [LRU|CLOCK]`
where `input_file` is the file name with page request information. `LRU` and `CLOCK` 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:

(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
```

(Clock)

```
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 ecs-pa-coding1 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 `<SacLink username>-asgmt4`.
- Use tar command to place all the files in a single file called `<SacLink username>-asgmt4.tar`. Assuming you are in the directory `<SacLink username>-asgmt4` do the following:
  - Goto the parent directory: `cd ..`
  - tar the files:
    `tar -cvf <SacLink username>-asgmt4.tar ./<SacLink username>-asgmt4`
  - Verify the files have been placed in a tar file:
    `tar -tvf <SacLink username>-asgmt4.tar`
  - Compress the files using gzip: `gzip <SacLink username>-asgmt4.tar`
  - Verify that the gzipped file exists: `ls <SacLink username>-asgmt4.tar.gz`