Simulating a Virtual Memory Manager

2016-17, CSCI 3150 - Assignment 5

Release: 1 Dec 2016

Deadline: 15 Dec 2016 11:59AM

Contents

1 Introduction

In this assignment, you are going to simulate a virtual memory manager. The (simulated) manager shall be implemented with a variant of LRU paging algorithm. First, the manager is instantiated with a hypothetical configuration (e.g., there are 8 pages, 4 frames, etc.). Then, we will access logical addresses through the manager. For each logical address, the manager shall return the corresponding (simulated) physical address based on the given hypothetical configuration.

1.1 The manager

On page fault, the manager shall choose a frame as follows:

- If there are several available frames, select the one with smallest frame id;
- If there is only one free frame, choose it;
- If all of the frames are occupied, swap out a victim page based on the paging algorithm.

1.2 The paging algorithm

The paging algorithm is a variant of the LRU algorithm, which keeps track of the timestamp of the last m references to each page, where m is a positive integer.

Let $time(p, m)$ be the time of the m^{th}**-most recent reference to page** p. If page p has less than m references, $time(p, m)$ would be the time of the most recent reference to p. Hence,

$$time(p,m) = \begin{cases} time(p,1) & \text{the total number of references to } p \text{ is less than } m \\ time(p,m) & \text{otherwise} \end{cases}$$

Therefore, when choosing a victim page to swap out, the algorithm **chooses the one with the earliest/smallest** $time(p, m)$. Note that when $m = 1$, this algorithm degrades back to the conventional LRU paging algorithm.

1.3 Example

Assume we have 8 pages indexed from 0 to 7 and 4 frames numbered from 0 to 3. The reference sequence (page IDs) is 0, 0, 0, 1, 1, 3, 3, 3, 3, 4, 2, 2, 6, 2, 1, 5, 5, 5, 6. The following shows the sequence of actions of the manager when $m = 3$.

Time	**Reference**	**Frame ID**	**Explanation**
1	0	0	Page 0 is not in physical memory now. Frames 0, 1, 2, 3 are all available. We choose the frame whose ID is the smallest. So frame 0 is returned.
2	0	0	Page 0 is already in frame 0. Return frame 0.
3	0	0	Page 0 is already in frame 0. Return frame 0.
4	1	1	Page 1 is not in physical memory now. Frames 1, 2, 3 are available. We choose the frame whose ID is the smallest. So frame 1 is returned.
5	1	1	Page 1 is already in frame 1. Return frame 1.
6	3	2	Page 3 is not in physical memory now. Frames 2, 3 are available. We choose the frame whose ID is the smallest. So frame 2 is returned.
7	3	2	Page 3 is already in frame 2. Return frame 2.
8	3	2	Page 3 is already in frame 2. Return frame 2.
9	3	2	Page 3 is already in frame 2. Return frame 2.
10	4	3	Page 4 is not in physical memory now. Only frame 3 is available. So frame 3 is returned.
11	2	0	$time(0,3) = 1$, $time(1,3) = time(1,1) = 5$, $time(3,3) = 7$, $time(4,3) = time(4,1) = 10$. Swap page 0 out. Return frame 0.
12	2	0	Page 2 is already in frame 0. Return frame 0.
13	6	1	$time(1,3) = time(1,1) = 5$, $time(3,3) = 7$, $time(4,3) = time(4,1) = 10$, $time(2,3) = time(2,1) = 12$. Swap page 1 out. Return frame 1.
14	2	0	Page 2 is already in frame 0. Return frame 0.
15	1	2	$time(3,3) = 7$, $time(4,3) = time(4,1) = 10$, $time(2,3) = 11$, $time(6,3) = time(6,1) = 13$. Swap page 3 out. Return frame 2.
16	5	2	$time(4,3) = time(4,1) = 10$, $time(2,3) = 11$, $time(6,3) = time(6,1) = 13$, $time(1,3) = 4$. Swap page 1 out. Return frame 2.
17	5	2	Page 5 is already in frame 2. Return frame 2.
18	5	2	Page 5 is already in frame 2. Return frame 2.
19	6	1	Page 6 is already in frame 1. Return frame 1.

2 Your assignment

You are given the following files:

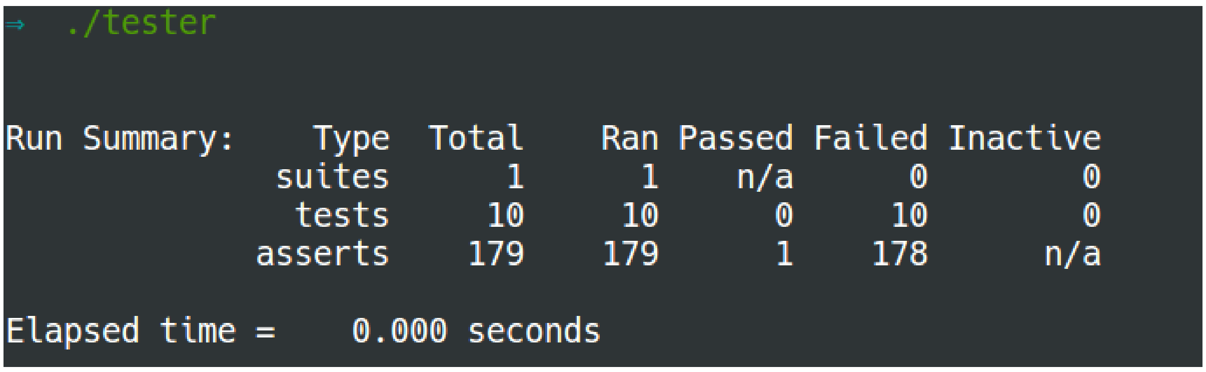
Name	Description
`/manager.c`	Source code of a runnable but non-functioning manager.c (**Work on it**).
`/manager.h`	Header file of manager.c (**Work on it**).
`/tester.c`	We will run this program to grade your assignment (**Don't touch**).
`/Makefile`	Makefile to compile `tester` (**Don't touch**).
`/testcase`	Test cases used in tester (**Don't touch**).
`/testcase_ans`	Answer to the test cases in tester (**Don't touch**).

2.1 To begin

This time, we use CUnit to test your program. Please use the following command to install CUnit first:

```
sudo apt-get install libcunit1 libcunit1-doc libcunit1-dev
```

Then make and run `tester`, you shall see something like this:



```
⇒  ./tester


Run Summary:    Type  Total    Ran Passed Failed Inactive
              suites      1      1    n/a      0        0
               tests     10     10      0     10        0
             asserts    179    179      1    178      n/a

Elapsed time =    0.000 seconds
```

Ignore other rows and focus on the row "`tests`". Before you work on your assignment, you shall see it has run 10 test cases and 0 passed.

2.2 Your job

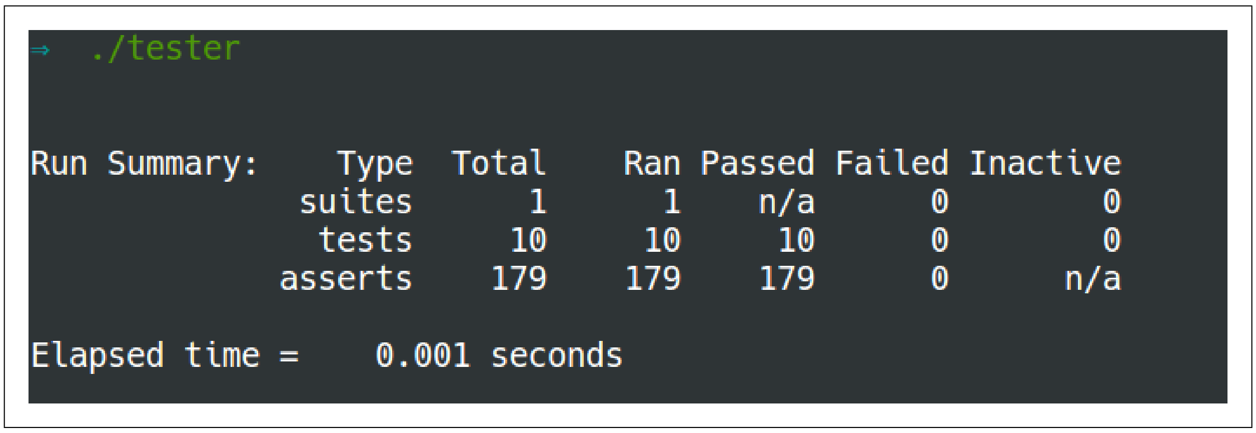
You should start from the given `manager.c` and `manager.h`. Read them carefully. The following is a brief introduction:

- `manager_t`, a structure defined in `manager.h` to represent the memory manager. The suffix "`_t`" of `manager_t` identifies this as a type. Currently there are four members in the structure `manager_t`:
  - `_page_num`, the number of pages;
  - `_frame_num`, the number of frame;
  - `_frame_size`, the size of frame;
  - `_lru_parameter`, the value of parameter m

  The prefix "`_`" means that this variable is a member of the structure. You can add other member variables to `manager_t` if necessary.
- `manager_t* new_memory_manager(uint32_t page_num, uint32_t frame_num, uint32_t frame_size, uint32_t lru_parameter)`, this function instantiates a `manager_t`.
- `void deconstruct_manager(manager_t* self)`, this function frees the `manager_t`.
- `uint32_t access(manager_t* self, uint32_t addr)`, return the physical address given the logical address `addr`. You have to handle the details here (e.g., extracting the page id and page offset from `addr`, which is introduced in 2.4. Note that here we take logical address as input and require to return physical address.)

You can add other functions to `manager.c` and `manager.h`. Do NOT change the name and usage of any existing functions.

If your program works correctly, running `tester` will show:



```
⇒  ./tester


Run Summary:    Type  Total    Ran Passed Failed Inactive
              suites      1      1    n/a      0        0
               tests     10     10     10      0        0
             asserts    179    179    179      0      n/a

Elapsed time =    0.001 seconds
```

2.3 Submission

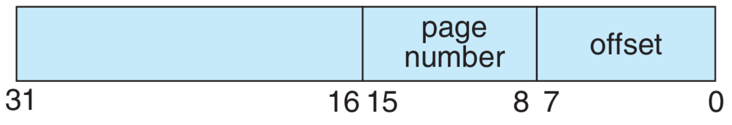
Submit two files, `manager.c` and `manager.h`. Please compress them to **asgn5-*SID*.tar.gz** and submit it to eLearning. E.g., if your SID is 1155031234, please submit asgn5-1155031234.tar.gz to eLearning.

Warning: DON'T change the file names, otherwise you will get zero marks

2.4 Assumptions

You can assume the following assumptions always hold in this assignment:

- The logical address, page_id and frame_id are of type `uint32_t`.
- The size of each frame equals to that of each page.
- All of the frame size, the amount of pages and the number of frames are power of 2.
- The rightmost bits of a logical address represent the page offset and the bits next to the page offset represent the page ID (page number). Consider a configuration with 256 pages and suppose the page size is 2^8 bytes. As shown in the following figure, only the rightmost 16 bits of each logical address matter:



3 Grading

1. 10 test cases in total.
2. Passing one test case will get 10 marks.
3. **Hardcoding won't work**. We use some other inputs to check if you have done any hardcoding.
4. The grading will be fully automated. The grading platform is our course's VM: Ubuntu 14.04 32-bit. Once we start the grading processing, we reserve the right to deduct marks from you for any request that requires TA's extra manual effort.

3.1 Late Submission Policy

We follow the late submission policy specified in the course outline.

4 Questions

If you have doubts about the assignment, you are encouraged to ask questions on our Facebook group.

5 Academic Honesty

We follow the University guide on academic honesty against any plagiarism.