CS 5/7343 Fall 2011

Programming Homework 4

Due :12/11 (Sat) 11:59pm, No extensions.

The goal of this homework is to implement page replacement algorithm.

You are given a program that implement a version of FIFO page replacement policy. The program consists of 3 parts:

- Class PRDS_FIFO: a class that store the data structure that is used by the page replacement algorithm. In this case this is a just a simple deque<int>
- int Page_Replacement(vector<int>& pages, int nextpage, PRDS_FIFO* p). Implementation of the page replacement algorithm.
  - Parameter:
    - pages: a vector storing the set of pages that is currently in main memory
    - nextpage: the next page to be accessed (nextpage is a non-negative number, between 0 and MAXINT).
    - p : a pointer to the data structure that is used for the page replacement
  - Output:
    - -1: if no page replacement is needed
    - any other number: the index in the pages vector that is to be replaced
- The main program, who generate a list of page references and execute the page replacement algorithm. For each page request, it will print a line
  - The first number is the page that is requested
  - The second number is the value returned by the page replacement function.
  - The rest of the numbers denote the pages in the pages vector (i.e. those who are "in memory")

The code has comments that should help you understand the it.

Task (Base case) (90 points)

You need to implement 2 page replacement algorithms:

Case 1: LRU

You will need to define a class (PRDS_LRU) that store the data structure that will be used to implement the algorithm. It must have a constructor that take an integer as a parameter, which denotes the number of pages available in main memory. Notice that you are NOT required to use that number if you do not want to. You can define any other methods that you need.

You also need to implement an function: *int Page_Replacement_LRU(vector<int>& pages, int nextpage, PRDS_LRU* p)*, that implement the LRU algorithm. The three parameters are the same as the three parameters in the Page_Replacement function provided in the code.

You should put the code for these two parts in a file name "prog4_lru.h".

Case 2: modified 2nd chance algorithm.

You should implement the following modification of the 2nd chance algorithm.

- Each page in main memory has a reference number (between 0 and 3 inclusive) associated with it. When a page is read in, its reference number is initialized to 0
- Once a page is in memory, if it is reference again, its reference number is incremented by 1 (but it will never be larger than 3)
- A FIFO queue is maintained, whenever a new page is brought into memory, it is inserted at the end of the queue
- When a victim for replacement is needed, we apply the following algorithm to select it
    - If the page a the front of the FIFO queue is 0, then this page is selected and removed from the queue
    - Otherwise, it is moved to the back of the queue, and its reference number is decreased by 1
    - Repeat until you find a victim

  (Notice that you do NOT have to implement this algorithm as stated, as long as you achieve the same effect).

You will need to define a class (PRDS_2nd) that store the data structure that will be used to implement the algorithm. It must have a constructor that take an integer as a parameter, which denotes the number of pages available in main memory. Notice that you are NOT required to use that number if you do not want to. You can define any other methods that you need.

You also need to implement an function: *int Page_Replacement_2nd(vector<int>& pages, int nextpage, PRDS_LRU* p*), that implement the LRU algorithm. The three parameters are the same as the three parameters in the Page_Replacement function provided in the code.

You should put the code for these two parts in a file name "prog4_2nd.h".

Below are extra credits, which are only open to students that get the full 90 points for case 1 and 2.

Extra credit 1

We will compare the running time of both algorithms. I will run multiple tests, each of them over a sequence of 1000s or more page references. For each case the running time will be compared and ranked. The ranks for all cases will be added up and a final rank is obtained. The running time of the programs will be ranked, and the fastest programs will earn extra bonus based on the following table:

Rank	Extra Bonus
1	35
2	25
3-4	20
5-8	10
9-14	5
15-22	3

If there are ties, the score will be averaged. For example, if the 2nd and 3rd place student tied, each of them will get (25+20)/2 = 22.5 bonus points

Extra credit 2

You are to implement a page replacement algorithm of your choice. You can come up with your own algorithm, or just modify the ones you implemented. The only requirement is that you need to show your algorithm are not the same as the three other algorithms here (FIFO, LRU, and modified 2nd chance). You do this by generating three input cases (they can be the same case), such that for one case, your output for your algorithm is different from that of FIFO, LRU and modified 2nd chance respectively.

You should follow the same format as the base cases: having a class called *PRDS_MyOwn* and a method called *int Page_Replacement_MyOwn(vector<int>& pages, int nextpage, PRDS_MyOwn* p).* You should put those code in a file "prog4_myown.h"

Your algorithm will be run on multiple cases, and the number of page faults will be ranked for each case. The total rank over all cases will be ordered, and the best performers will get bonus as below:

Rank	Extra Bonus
1-2	35
3-5	25
6-8	20
9-12	10
13-16	5
17-22	3

If there are ties, the score will be averaged. For example, if the 2nd and 3rd place student tied, each of them will get (25+20)/2 = 22.5 bonus points

In order to qualify for this bonus, your program cannot run too slow. The baseline is as follows:

- For each case, the time for each student in bonus 1 for the 2nd chance algorithm is recorded.
- We find the median of those times.
- Your program cannot run 4 times slower than that time.

Comment bonus

The 5 points comment bonus is still in effect.

No-output alert

You should not output ANYTHING in your code. If you have debugging code in your program, please make sure you comment it before you hand in. Otherwise, I will take 5 points off.

What to hand in

You should hand in ONLY the source code for the .h file specified (zipped). If you attempt extra credit 2. You should also hand in a half to one page document that describe your algorithm for extra credit 2.

Full marks

Full mark for the 5000-level class is 100, and full mark for the 7000 level is 105.