COMP104 - 2017 - Second CA Assignment
Storage Management
Page Replacement Methods

Assessment Information

Assignment Number	2 (of 2)
Weighting	10%
Assignment Circulated	Monday 13th March 2017
Deadline	Wednesday 3rd May 2017; 12.00
Submission Mode	Electronic
Learning outcome assessed Marking criteria	4, viz “Construct programs which demonstrate in a simple form the operation of examples of systems programs Scheme provided at end of document.
Submission necessary in order to satisfy Module requirements?	No

1 Introductory Background

The organization of memory into *Page Frames* each of which can store a single fixed size *Page*, is a standard method of presenting a view that the available "fast" memory is much larger than is actually the case. Typically fast **RAM** and cache storage may be less than 1% of the total physical storage available.

As has been described in the lectures, *paged memory* uses a *physical* division assigning numbers to page *frames* from 0 up to $N-1$ (with N dependent on the total amount of storage space provided) and a *logical* division of pages between running processes. At any given time **page number** k will be resident in some **page frame** and, in order to keep track of which page number is (currently) resident in a given page frame, the Memory Manager makes use of a **Page Table**. This being a mapping from page numbers to the frames in which they are held, so that if $PT[k] = m$ this captures the fact that page *number* k is in page *frame* m.

In order to be read and written to, a page has to reside in a frame corresponding to one of those forming part of the **RAM**, and since this cannot hold every physical page, from time to time a page held in **RAM** may be swopped out (ie moved to the secondary disk store) in order to free a page frame for a page which has been referenced but (after checking the page table) has been found to reside in a frame outside main memory: an event referred to as a *page fault*.

As was discussed in lectures, in order to alleviate this problem a decision has to be made regarding which of those pages currently held in **RAM** ought to be removed and, to facilitate this decision a number of *page replacement policies* have been proposed. In particular:

A. Longest Resident: use the page frame which is holding the page that has been held in memory for the longest time.

B. Least recently used (LRU): use the the page frame holding the page which has not been accessed for the longest time.

C. Least frequently used (LFU): use the page frame holding the page which has been accessed the least number of times.

2 Assignment Details

The purpose of the present assignment is to provide a comparison of three approaches with respect to a given *trace* of page references. Specifically, you should implement (in Java) a simulation of the following:

a. Using a `Store size` of `1024` page frames, the first `8` of which are **RAM** and the remaining `1016` disk, page frames (i.e. secondary memory), a *page trace* is any sequence of numbers $< p_1,\ p_2,\ \ldots\ , p_k,\ \ldots\ >$ all of which are at least 0 and at most 1023. The item p_i gives the page *number* referenced at time i.

b. If the page *frame*, Y say, in which page *number* p_i is resident is *greater than* 7 then a **page fault** has occurred. Dealing with this requires,

   1. Identifying the page **frame** in **RAM** (ie with a number between 0 and 7) that contains the page **number** to be replaced according to the page replacement method used. Frame X and page P say.
   2. Swopping page P with page p_i: after this swop, page **number** P is in the page **frame** that had held p_i, (ie page frame Y) while page **number** p_i is now in page **frame** X.
   3. Updating the page table so that $PT[p_i] = X$ and $PT[P] = Y$

c. Deal with the next page reference in the page trace.

The **three** (3) files located at

```
www.csc.liv.ac.uk/~ped/COMP104/COMP104-2016-17/Page_Trace_Oldest
www.csc.liv.ac.uk/~ped/COMP104/COMP104-2016-17/Page_Trace_LRU
www.csc.liv.ac.uk/~ped/COMP104/COMP104-2016-17/Page_Trace_Random
```

contain "randomly" generated lists[1] of **20,000** of page references, the first number in each indicating the page replacement policy to be applied so that

First Number in File	Page Replacement Policy
0	OLDEST – choose the page that has been held in RAM **longest**
1	LRU – choose the least recently **used**
2	RANDOM – choose a **random** page in RAM to swop out

Your Java program should carry out the following, using these data files as sample inputs:

1. Determine the page replacement method to be used (as specified by the first number in the file and the mapping just described). (Note the "random" policy – not discussed in lectures – is simply to choose any one of the 8 page frames in **RAM** at random).

2. Process the sequence of page references (at most **20,000**) maintaining details of

   A. Which page **number** is (currently) held in page **frame** k (for $0 \leq k < 1024$).

   B. Which page **frame** (currently) holds page **number** p (again for $0 \leq p < 1024$): note this is the basic information that is maintained by the **Page Table**.

[1] For anyone interested in more detail, this is not simply produced by generating a sequence of random numbers between 0 and 1023, but is intended to reflect a Poisson arrival process for new processes: this is one of the standard models of event occurrences studied in Queuing Theory, once an important topic forming part of most Computer Science degree programmes.

It may be assumed at the start of the simulation that page **frame** k contains page **number** k.

3. Using the page reference data in the files, output a log of the page fault data. This should be of form

```
Page Replacement Method used

Fr0 | Fr1 | Fr2 | Fr3 | Fr4 | Fr5 | Fr6 | Fr7 | Time | Page Faults since last check
```

Here `FrX` is the page **number** of the page in `Frame X`; `Time` is the total number of page references processed so far (and thus will be at most $20,000$). The data should be output every 100 page references and the final column of output is the number of page faults seen in the last 100 references.

3 Further Details

In addition to the main program details you should implement a `Page` class using the following fields and methods:

Fields	
private int `current_frame`	The frame occupied by **this** page
private int `loaded_at`	The Clocktime **this** page was loaded into **RAM**
private int `last_read`	The Clocktime **this** page was last accessed
Constructor	
`Page()`	Initiates all fields to -1
Methods	
private int `GetFrame()`	Return `current_frame` for **this** page

private void `LoadPage(int frame_value, int ClockTime)`	Assigns **this** page to frame `frame_value`
	if `frame_value` is ≤ 7 `loaded_at=ClockTime`
	if `frame_value` is > 7 `loaded_at=-1`

private void `Update(int ClockTime)`	Updates `last_read` to `ClockTime` for **this** page
private int `GetAge()`	returns the value of `loaded_at` for **this** page
private int `GetLastAccess()`	returns the value of `last_read` for **this** page
Additional Methods	
public static int `Find_Oldest(int[] Store, Page[] Table)`	return the *frame number* of the frame with the longest *resident* page
public static int `Find_LRU(int[] Store, Page[] Table)`	return the *frame number* of the frame with the least *recently used* page

With this class the `Page` Table is simply an *array of Page.* It is important to note that this table should **not** be confused or conflated with the array (of **int**[] corresponding to the **physical memory** (ie the array `Store[]` mentioned in the `Find_Oldest` and `Find_LRU` methods): when `Store[k]=p` this means that "*page **number** p is currently held in page **frame** k*" so that `Table[p].GetFrame()` should return the value k.

Submission Instructions

Firstly, check that you have adhered to the following list:

1. All of your code is within a **single** file. Do **NOT** use more than one file.
2. Both your **name** AND **User ID** are clearly indicated at the start of your code, eg by

   ```
   // Name: My Name ; ID u?????
   ```

3. The file's name **MUST** be

   `Paging.java`

   This means that the main class name must also be `Paging`.

   Submit **only** the Java source: design documentation, compiled .class files, sample outputs, extraneous commentary and similar ephemera are neither required nor desired.

4. Please note that it is **NOT** required to submit the output from your program: this can be generated independently by running your code.
5. Your program is written in Java, not some other language.
6. The file is a **text** file: not compressed or encoded or otherwise mangled.
7. Your program compiles and runs on the Departmental Windows system. If you have developed your code elsewhere (eg your home PC), port it to our system and perform a compile/check test before submission. It is your responsibility to check that you can log onto the departmental system well in advance of the submission deadline.
8. Your program does not bear undue resemblance to anybody else's. Electronic checks for code similarity will be performed on all submissions and instances of plagiarism will be dealt with in accordance with the procedures and sanctions prescribed by the relevant University Code of Practice. The rules on plagiarism and collusion are explicit: do not copy anything from anyone else's code, do not let anyone else copy from your code and do not hand in "jointly developed" solutions.

Your solution must be

SUBMITTED ***ELECTRONICALLY***

Electronic submission: Your code must be submitted to the departmental electronic submission system at:

`http://intranet.csc.liv.ac.uk/cgi-bin/submit.pl`

You need to login in to the above system and select **COMP104-2: Storage Management** from the drop-down menu. You then locate the file containing your program that you wish to submit, check the box stating that you have read and understood the University Code of Practice on Plagiarism and Collusion, then click the Upload File button.

MARKING SCHEME

Below is the breakdown of the mark scheme for this assignment. Each category will be judged on the correctness, efficiency and modularity of the code, as well as whether or not it compiles and produces the desired output.

- Adherence to specification (ie information requested, correct naming etc.) = 10
- Implementation of Page class and methods = 25.
- Simulation Structure = 15
- Implementation of replacement algorithms = 25
- Output form = 15 marks
- Comments and layout = 10 marks

This assignment contributes 10% to your overall mark for COMP104.

Finally, please remember that it is always better to hand in an incomplete piece of work, which will result in some marks being awarded, as opposed to handing in nothing, which will guarantee a mark of 0 being awarded. Demonstrators will be on hand during the COMP104 practical sessions to provide assistance, should you need it.