THE UNIVERSITY OF NEWCASTLE AUSTRALIA

School of Information and Physical Sciences
Semester 2, 2023 - COMP2240/COMP6240 - Operating Systems

Assignment 3 (15%)

Submit using Canvas by **11:59 pm, Friday 20th October 2023**

version 1.0.0

1. Introduction

In this assignment you are once again to produce a simulation – this time we will be simulating Paging with Virtual Memory, with two different Page Replacement Policies.

You will be required to produce a small Java application demonstrating the simulation of these two Page Replacement Policies, as well as a written report documenting aspects of your code and results.

You are advised to make yourself familiar with the Submission Requirements (*Section 3*), and assignment expectations before beginning the coding task.

2. Assignment Task

You are to compare the performance of the **Clock** page replacement algorithms for the fixed allocation with local replacement and variable allocation with global replacement strategies, as introduced in lectures.

In Assignment 1, we assumed that the system had *an infinite amount of memory*. In this assignment, the Operating System more realistically has access to a limited amount of memory, and this needs to be managed to meet process demands.

You will write a program to simulate a system that uses paging with virtual memory. The system characteristics are described below:

2.1. Memory

The system has **`F`** frames available in user memory space, a value that will be supplied as an argument. During execution, the processor will determine if the page required for the currently running process is in main memory.

a. If the page is in main memory, the processor will access the instruction and continue.

b. If the page is not in main memory, the processor will issue a *page fault* and *block* the process until the page has been transferred to main memory.
c. Initially no page is in the memory. I.e., the simulation will be strictly *demand paging*, where pages are only brought into main memory when they are requested.
d. In fixed allocation scheme frames are equally divided among processes, additional frames remain unused. In variable allocation scheme all frames are available to the processes.

2.2. Paging & Virtual Memory

The system using paging (with no segmentation).

a. Each process can have **a maximum 50 pages** where each page contains a single instruction.
b. For Page Replacement you will need to apply **Clock** policy as taught in this course.
c. You will need to implement two different schemes for resident set management
    i. '***Fixed Allocation with Local Replacement Scope***' – In this scheme, the frames allocated to a process do not change over the simulation period i.e. even after a process finishes, the allocated frames do not change. And the page replacements (if necessary) will occur within the frames allocated to a process using **Clock** policy.
    ii. '***Variable Allocation with Global Replacement Scope***' – In this scheme, no specific frame is allocated to any process rather all frames are available to the processes for use. A process can use an unused frame in the user memory space to bring in its own page. For page replacement it will use **Clock** policy but will consider all the pages in the user memory space. I.e. the page selected for replacement may belong to any process running in the system. When a process finish execution the frames to that finished process are not returned immediately rather will be replaced by pages from other processes using **Clock** replacement policy.
d. For simplicity, all pages are read-only, so no page needs to be written-back to disk.

2.3. Page Fault Handling

When a page fault occurs, the interrupt routine will handle the fault by blocking that process and placing an I/O request. The I/O controller will process the I/O request and bring the page into main memory. This may require replacing a page in main memory using a page replacement policy (**Clock**).

Other ready processes should be scheduled to run while such an I/O operation is occurring.

a. Issuing a page fault and blocking a process takes *no time*, so *multiple page faults may occur* and then another ready process can run immediately *at the same time unit.*
b. Swapping in a page takes *6 units* of time (*if a page required by a process is not in main memory, the process must be put into its blocked state until the required page is available*).

c. If a process is unblocked (*ie. the requested page is placed in the main memory*) at time *t* then it can be scheduled and the requested page can be executed at *t*.

2.4. Scheduling

The system is to use a **Round Robin short-term scheduling algorithm** with time a quantum of ***Q***.

a. Executing a single instruction (*ie. a page*) takes *1 unit* of time.
b. Switching the processes *does not take any time*.
c. All the processes start execution at time *t = 0*. And they will be processed in the order the *process names appear in the input*.
d. If a process becomes ready at time unit *t* then execution of that process may occur in the same time unit *t* without any delay (if there is no other process running or waiting in the ready queue).
e. If *multiple process* becomes ready at the same time then they will enter the ready queue *in the order they became blocked*.
f. If a process **`P1`** is finishes it's time quantum at *t1* and another process `P2` becomes unblocked at the same time *t1*, then the unblocked process **`P2`** is added in the ready queue *first* and the *time-quantum expired* process **`P1`** is added *after* that.

You are to compare the performance of the ***Clock with Fixed Allocation & Local Page Replacement*** and ***Clock with Variable Allocation & Global Page Replacement*** algorithms.

Please use the basic versions of the policies introduced in the lectures.

2.5. Process File Format

Since we assume that each page contains a single instruction, an execution trace is simply a page request sequence.

For example: (`process1.txt`)

```
name: process1;
page: 1;
page: 2;
page: 3;
page: 4;
page: 5;
end;
```

This specifies a process that first reads page `1`, then page `2`, and so on until the `end` is reached.

Note 1) white space plays not part in the file format – the above file is just as valid if supplied as a single line with no spaces at all!

Note 2) the filename has no relevance – it will be a text file, but should not be restricted to any specific extension. The 'name' of the process is included inside the data file.

2.6. Simulation Output

For each replacement strategy, the simulation should produce a summary showing, for each process, the total number of page faults, the time the page faults were generated, and the total turnaround time (*including I/O time*).

The output should be printed to the console (*ie: the terminal window, from where your program was run*), using the following sample layout:

```
Clock - Fixed-Local Replacement:
PID  Process-Name  Turnaround  # Faults  Fault-Times
1    process1      38          5         {0, 7, 16, 23, 30}
2    process2      39          5         {0, 8, 17, 24, 31}
3    process3      18          1         {0}
4    process4      37          4         {0, 12, 19, 28}

------------------------------------------------------------

Clock - Variable-Global Replacement:
PID  Process-Name  Turnaround  # Faults  Fault-Times
1    process1      38          5         {0, 7, 16, 23, 30}
2    process2      39          5         {0, 8, 17, 24, 31}
3    process3      18          1         {0}
4    process4      37          4         {0, 12, 19, 28}
```

See `\Sample1` for the dataset that generate this output, and `\Sample2` for a further dataset and sample results.

It is suggested you generate some further testing data than that provided.

2.7. Report

Finally you will submit a **brief 2 page** (*A4*) **report** documenting:

a. Your testing approach and methodology.
b. A comparison of the page replacement methods based on the results from your program and any interesting observations.
c. A discussion on edge cases you considered, and behaviour of your algorithm on those cases.
d. A review of the results from your program, including any interesting observations or unexpected behaviour you encountered during development, including any specific issues you faced.

3. Programming Standards

Your submission must also conform to the follow requirements.

3.1. Programming Language

The programming language is Java, versioned as per the University Lab Environment (*Note this is currently a **subversion of Java 17***). You may only use standard Java libraries as part of your submission.

3.2. Code Standards

Yes, *Code standards matter*! There is a mark component that covers commenting and general code form, so make sure your submitted code is consistent, valid variable names, appropriate commenting, etc.

Additionally, `there will be no nested classes*` – each java class will be in a separate file. Submitting something like ALL classes inside a single class will attract a heavier penalty, so please make sure you code your submission in a manner that keeps a single class in a single file.

*Nested Class are permitted in specific use cases, in line with Java Best Practices, such as Iterators being written as a Private Inner Class of its container. But there are no valid use cases immediately apparent for this assignment. *If you feel you have a use case that requires Nested Classes, speak to Dan!*

Yes, multiple source files can be compiled in line with the specification, as compilation of the program entry point class will invoke compilation of any referenced classes, and so forth!

3.3. User Interface

The output should be printed to the console (*ie: the terminal window, from where your program was run*); there is no need to print to file, create a menu system, or display in a graphical manner.

3.4. Input and Output

Your program will accept data from a set of input files, whose names are specified as part of a command line argument. **Hint:** the Java `Scanner` Library is something you will likely want to use here!

Your submission will be tested with the above data and will also be tested with other input files, that may or may not be formatted the same, with respect to white space!

For each replacement strategy, the simulation should produce a summary showing, for each process, the turnaround time, the total number of page faults and the time the page faults were generated.

Sample inputs/outputs are provided. Working of the first example is presented with details of different levels for visualisation. Your submission will be tested with this data and will also be tested with other input files.

Your program should output to standard output (*this means output to the Console*). Output should strictly follow the provided format samples (*see the output files*). `If output is not generated in the required format then your program will be considered incorrect.`

3.4.1. Command Line Parameters

Input to your program should be via **command line arguments** (*see also in* **Section 3.6. Deliverable:**), where the arguments are system configurations and the names of files that specify the execution trace of different processes. **All processes start execution at time** `0`, and are entered into the system in the order of the command line arguments (*with the third argument being the first process*).

```
java A3 F Q data1 data2 ... datan
```

For example:

```
java A3 30 3 process1.txt process2.txt process3.txt
```

…where `30` is the number of frames (`F`) and `3` is the quantum size (`Q`) for Round Robin and related algorithms. Values `F` and `Q` are assumed to be integer values.

This is the only valid means of execution for this assignment.

The following arguments are text file names containing page references for each process, as defined in the next section. These are relative filenames and **SHOULD NOT BE HARDCODED** in anyway (*be it path, filename, extension, etc*). You may assume the number of supplied processes is a *minimum of two,* with no upper bound.

To be clear, the filenames above can be *any* filename with any extension (*or no extension*) – **do not hardcode filenames or extensions**! Please note that submissions that diverge from this will be **marked with heavy penalty**.

3.4.2. Input Notes

You may assume all arguments and data is ‘sunny day’ – we will not be testing with invalid values (*say a negative* `F` *value*) or invalid data files (*thus they will all follow the provided format, with white space to be ignored*) or invalid file paths – therefore you are not expected to do any validation.

4. Submission Guidelines

4.1. Deliverable

The following must be observed in your deliverable:

1. Your submission will contain your **program source code** with documentation and the report (*below*) in the root of the submission. These files will be **zipped and submitted in an archive** named `c9876543.zip` (*where* `c9876543` *is your student number*) – do not submit a `.rar`, or a `.7z`, or etc.

   If you vary from this format, `there will be a deduction` for non-standard submission.

2. Your **main class** will be named `A3.java`; your program will thus compile with the command `javac A3.java`, and your program will be executed by running `java A3 30 3 process1.txt process2.txt ... processn.txt`.…where the files can be *any* filename – `do not hardcode filenames or extensions`!

   Note: If your program can not be compiled and executed in the expected manner (*above*) then please add a `readme.txt` (*containing any special instructions required to compile and run the source code*) in the root of the submission. Please note that such non-standard submissions will be `marked with penalty`.

3. Your **report** – you will compile this as a **PDF document** called `Report.pdf`, and place it in the root of the submission archive.

Variation from these deliverable standards will attract marking deductions.

4.1. Mark Distribution

Mark distribution can be found in the assignment draft marking breakdown documents: `23s2_Assign3_Feedback.pdf`.

Note: these are draft marking breakdowns and maybe updated.

4.2. Submission Notes

While a formality, please be aware of the following:

1. Assignments submitted after the deadline (**11:59 pm 20th August 2023)** will have the final result reduced by 10% of the maximum possible marks per day.

   A '*day*' begins at 12:00 Midnight, and once this time is passed, the penalty for the whole day is applied.

   This is UoN Policy.

2. If your assignment is not prepared and submitted following provide instructions then you will lose marks, in spite of parts of the task being correctly calculated.

3. If you have a RAP, please submit this to the course coordinator as soon as possible at the start of Semester.

4. If you have been granted an extension, please include a copy of the extension approval in as a **PDF document**, called `Extension.pdf`, and place this in the root of your submission, along with your other documents and code.

Nasim and Dan
v1.0