**CS5250 – Advanced Operating Systems**
**AY2020/2021 Semester 2**

# Assignment 4

**Deadline: Monday, 19 April 2021 • 11.59pm**

## Section A (30 marks)

## 1. Objectives

1. Get a deeper understanding on memory management.

## 2. Rules

1. It is fine to ask for "reasonable" amount of help from others, but ensure that you do all the tasks on your own and write the report on your own. The University's policy on plagiarism applies here and any breaches will be dealt with severely.
2. For this assignment, there are two parts, and write a report (check assignment section for more details).
3. Working, reasoning and/or documentation are expected. Merely stating the final answer – even if it is correct – will at best yield partial marks.
4. Generate your report as a pdf file, name it as "*Name (Student Number) Assignment4.pdf*" and upload your report in the IVLE folder Assignment-4 of CS5250.
5. The deadline is 19 April 2021. Late assignments lose 4 marks per day.

## 3. Assignment

In this assignment, you shall attempt to implement a simulator for the buddy algorithm with the two-LRU list of Slide 117 of the PPT on Memory Management. The actual algorithm is very complex and tightly knitted to many aspects of the Linux memory management system including page tables etc. with multiple heuristics used. We can't do all of that. Instead we will make some simplifications so that you can simulate the general idea of the allocator and reclamation algorithm.

You task for the assignment will be to write a simulator in C (in the spirit of Linux and to ease my marking load – I need to compile your code, so if everyone start using all sorts of programming languages, I will have lots of difficulties just installing them) to do the following:

1. Implement a buddy allocator. For our assignment, we will assume that the system has 2 MB of physical memory, i.e., 512 pages.
2. Implement the 2-LRU page reclamation list. Use the basic one on Slide 117. Assume that the *maximum* number of active and inactive entries is **250 each**. The actual code in Linux allows these two to vary with various ways of determining what's active, what's inactive, and when is each list "too long". The goal is to accommodate as much as possible, the *working set* (the pages actively worked on) of a process, while balancing the overall demand for memory. But for simplicity of our assignment, let's fix the numbers.

3. There is a statically linked app in the assignment folder that will generate a unique input file for your application. It should run on any 64-bit Linux machine – it even ran on my bash on Windows. Start by inputting your student ID (this time I made it case insensitive – see, I listen to user feedback!). The file will be created in the same folder. The content of the file are lines of three varieties below. Each line is a single page transaction. Each line consists of 3 field separated by a tab.
    a. **`A <seqno> <num>`**
       Allocate **`<num>`** physical page frames. We name this allocation as **`<seqno>`**. **`<seqno>`** is used because the app have no idea what physical page number is allocated to it. Your simulator will have to do such tracking.
    b. **`X <seqno> <num>`**
       An access to the **`<num>`** page of allocation **`<seqno>`**.
    c. **`F <seqno> <num>`**
       Deallocate the **`<num>`** page of allocation **`<seqno>`**.
4. If a free is attempted on a frame that has been evicted, just ignore.
5. If an access is attempted on a frame that has been evicted, must treat as a page fault, and reallocate that a physical page frame – though not necessary the same one as before. This newly allocated physical page frame will take the **`<seqno><num>`** of the allocation corresponding to this access.
6. Document your algorithm for eviction.

At the end of the simulation, i.e., after processing the very last line, your simulator is to dump the final buddy lists, as well as the final state of the two LRU lists. Please format and print these out clearly for easy reading.

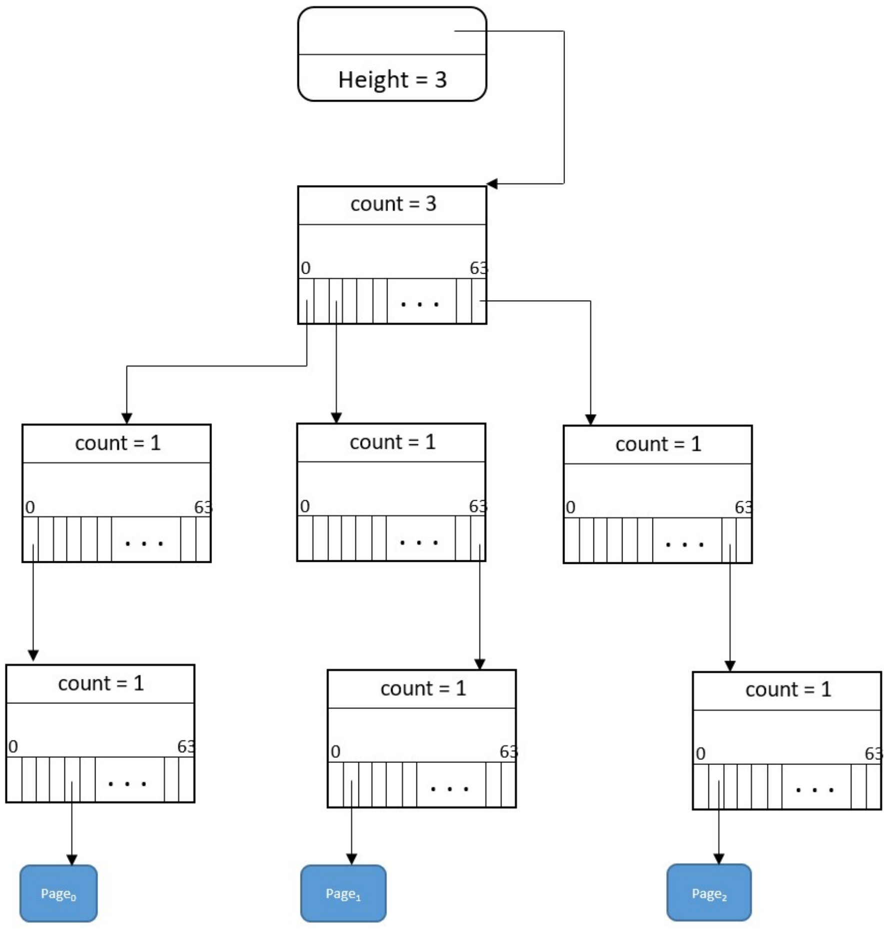
Submit a ZIP file containing:

1. A single PDF that consists of a description of your algorithms and documentation of your code, as well as the answer to Section B to the IVLE submission folder for this assignment before the deadline.

2. Your simulator code complete with a *simple* Makefile to compile and instruction to execute it. Your code should compile and run correctly with any GCC compiler > version 7. Please don't ask me to install a complex build facility or other packages just to compile your simulator – I will penalize you for this… I need to go through 92 answer scripts!

## Section B (15 marks)

B-1. (5 marks) Given the following Linux radix tree, what are the indexes of the three pages (in base 10)? Remember to show your workings.



B-2. (10 marks) Consider the following compare-and-exchange <u>atomic</u> instruction (abstracted as a function – in other words, assume that the following function is atomic):

```
int cas(int *ptr, int oldval, int newval);
```

If the integer pointed to by **ptr** is equal to **oldval**, then the content pointed to by **ptr** will be replaced with **newval**, and a '1' will be returned by **cas**. Otherwise, a '0' is returned. Using this function/instruction, show how a spinlock can be implemented by giving the C code for the **spin_lock()** and **spin_unlock()** functions. Write down any assumptions that you make.

*- End of Assignment -*