COM3610
Programming Assignment 4
Memory Management Unit Simulator (with eviction)

This project must be implemented in C

Notes

- It is **NOT OK** to share code with others. If you are having trouble, **come talk to me.**
- In this document, code and things you are expected to type are presented in `Courier fixed width font`.

Overview

In this assignment, you will be developing a simulated MMU.

MMU simulator – Implementation Details

When you execute your simulator, you will create your own page table of the appropriate size. Initially, no physical pages of memory will exist, because no memory accesses will have occurred yet.

Since the translation of virtual memory addresses into physical memory addresses require that both virtual pages and physical pages be appropriately aligned (your implementation must follow the scheme detailed in lecture), you will use the `posix_memalign()` command in order to create physical pages of memory which are appropriately aligned.

Virtual memory begins at memory location 0x00.

When your program exits, your physical pages should be `free()`ed in the order they were allocated.

NOTE: Your submission will be tested and graded using the COM3610 Virtual Machine, so be sure your code compiles and runs correctly there before handing in your submission.

This assignment will add the concept of pages of memory being moved from physical memory to disk when physical memory becomes full (this happens when the number of virtual pages that have content exceeds `pmpc`).

The eviction scheme used in this assignment is *First-in, First-out* (see 3/9 lecture).

When a virtual page is mapped to a physical page, a corresponding page on disc is assigned to be its "shadow". Since the page table tracks either a virtual page's physical memory page or disc page, You'll need a data structure that you can use to keep track of the address of each virtual page's disc shadow when the page table entry is storing the physical memory address.

When a physical page's content is changed, that change does not propagate to the corresponding page on disc. You need to keep track of the fact that the physical page has changed, so if the physical page has changed, when it's evicted, the physical page is copied over the "dirty" copy on disc. If the physical page has not changed, you don't copy the physical page to disc, since the copy on disc is not "dirty".

To make your implementation easier, simulate the disc by allocating pages in memory using `posix_memalign()` as you did with physical memory pages. At the beginning of execution, create an array of 100 pointers to these pages, and fill these pointers with references from 100 calls to `posix_memalign()`. This is "the disc".

MMU simulator – User Interface

You will implement an MMU simulator. It is invoked as follows:

```
mmusim [pagesize] [vmpc] [pmpc]
```

The command line arguments for your MMU simulator are to be interpreted as follows:

- `pagesize:` The size of a page of memory. This value must be a power of 2. If it is not, your program should display `error: pagesize must be power of 2` and exit.
- `vmpc:` The number of virtual pages of memory.
- `pmpc:` The number of physical pages of memory. ~~If pmpc is less than vmpc, your program should exit with `error: pmpc must be larger than or equal to vmpc`~~

Your program will execute and present this prompt, and display the prompt again on a newline after every command execution:

```
>
```

Your program will accept the following commands:

```
readbyte [location]
```

- `location:` The location in virtual memory to read. Using the page table you create, your program will find the corresponding location in physical memory and return the value from that location. Your program will display:

  ```
  readbyte: VM location 0x1122334455667788, which is PM location
  0x1122334455667788, contains value 0x11
  ```

  The byte will be displayed as hex (0xNN), where NN is the two-digit hex representation of the byte.

  If `location` is outside of the valid range of virtual memory, this command should instead display:

  ```
  readbyte: segmentation fault
  ```

```
writebyte [location] [value]
```

- `location:` The location in virtual memory to write. Using the page table you create, your program will find the corresponding location in physical memory and write `value` to that location.
- `value:` The value to write. The byte will be entered by the user as hex (0xNN), where NN is the two-digit hex representation of the byte.
- Your program will display:

  ```
  writebyte: VM location 0x1122334455667788, which is PM location
  0x1122334455667788, now contains value 0x11
  ```

- If `location` is outside of the valid range of virtual memory, this command should instead display:

  ```
  writebyte: segmentation fault
  ```

```
exit
```

- `exit` causes your program to exit.

Additional outputs to screen are:

- Whenever a physical page is mapped to a new or different virtual page, your program should display:

  ~~created~~ physical page at 0x1122334455667788~~,~~ mapped to virtual page at 0x1122334455667788

  Note: This will also occur whenever a virtual page needs to be accessed, and it's contents are on disc, so it needs to be copied into physical memory.

- Whenever a physical page needs to be evicted, and the content on the disc is out of date, your program should display:

  ```
  physical page at 0x1122334455667788, which corresponds to virtual
  page 0x1122334455667788, is evicted and dirty. Copied to disc at
  0x1122334455667788 and removed from physical memory
  ```

- Whenever a physical page needs to be evicted, and the content on the disc is not out of date, your program should display:

  ```
  physical page at 0x1122334455667788, which corresponds to virtual
  page 0x1122334455667788, is evicted and not dirty. Removed from
  physical memory
  ```

- ~~Whenever a physical page is destroyed, your program should display:~~

  ~~physical page at 0x1122334455667788 destroyed~~

MMU simulator – Example

In addition to examining your code, I will write a script that issues commands to your program, so be sure that your output is identical to what is detailed above and shown below:

```
bwymore@MSI:~$ mmusim 2000 100 1
error: pagesize must be power of 2
bwymore@MSI:~$ mmusim 2048 100
>readbyte 0xFF00AA00BB00CC00
readbyte: segmentation fault
>readbyte 0x0000000000000020
physical page at 0x0000000000BB8000 mapped to virtual page at 0x0000000000000000
readbyte: VM location 0x0000000000000020, which is PM location 0x0000000000BB8020, contains value 0x00
>readbyte 0x0000000000000021
readbyte: VM location 0x0000000000000021, which is PM location 0x0000000000BB8021, contains value 0x00
>writebyte 0x0000000000000021 0xAC
writebyte: VM location 0x0000000000000021, which is PM location 0x0000000000BB8021, now contains value 0xAC
>readbyte 0x0000000000000021
readbyte: VM location 0x0000000000000021, which is PM location 0x0000000000BB8021, contains value 0xAC
>writebyte 0x0000000000000021 0xAC
writebyte: VM location 0x0000000000000021, which is PM location 0x0000000000BB8021, now contains value 0xAC
>writebyte 0x000000000000080A 0xFF
physical page at 0x0000000000BB8000, which corresponds to virtual page 0x0000000000000000, is evicted and dirty. Copied
to disc at 0x0000000004455667788 and removed from physical memory
physical page at 0x0000000000BB8000 mapped to virtual page at 0x0000000000000800
writebyte: VM location 0x000000000000080A, which is PM location 0x0000000000BB800A, now contains value 0xFF
>readbyte 0x0000000000000021
physical page at 0x0000000000BB8000, which corresponds to virtual page 0x0000000000000800, is evicted and dirty. Copied
to disc at 0x00000000FF55FF77FF and removed from physical memory
physical page at 0x0000000000BB8000 mapped to virtual page at 0x0000000000000000
readbyte: VM location 0x0000000000000021, which is PM location 0x0000000000BB8021, contains value 0x00
>readbyte 0x000000000000080A
physical page at 0x0000000000BB8000, which corresponds to virtual page 0x0000000000000000, is evicted and not dirty.
Removed from physical memory
physical page at 0x0000000000BB8000 mapped to virtual page at 0x0000000000000800
readbyte: VM location 0x000000000000080A, which is PM location 0x0000000000BB800A, contains value 0xFF
>exit
bwymore@MSI:~$
```

Grading

Hand-in your c file via Canvas.