CS 4593/5463: Fundamentals of Systems – Programming Assignment 4 (100 + 20 points)

Description: Memory and Page Table.

This assignment includes two parts, with a total of **100 points**. There is an additional third part, worth **20 points**, that is extra-credit.

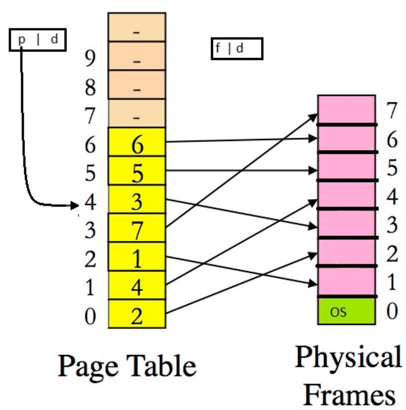
This assignment is on memory management, where we design a simulator that implements the OS's address translation mechanisms. Although an OS can usually support many processes, we only need to design a simulator that handles one process. The reference/example computing system for this assignment has the following properties: **1K physical memory, 4K virtual memory, and 128 bytes per page and frame.**

Before designing the simulator, you should understand the answers to the following questions:

(1) What is the maximum number of pages a process can access? (**Answer: 32 pages**)
(2) What is the total number of frames? (**Answer: 8 frames**)
(3) How many entries does the pagetable contain? (**Answer: 32 entries**)

You will practice the management of pagetable and physical memory allocation by emulating what happens inside the OS kernel.

Part 1: Address Translation and I/O (30 points)
Assume a process has the following page table:



Write a main program called **`assign4-part1.c`** that takes in only two parameter, **`infile`**, which is the name of a sequence file containing logical memory accesses, and **`outfile`**, which is the name of the file to which output is written to. Each logical address in `infile` is saved as **8 bytes (`unsigned long`)** in **binary format**. Your program should read each logical address and then translate it into a corresponding physical address based on the page-table given above. The physical addresses must be printed to the `outfile`, in the same binary format that the sequence file is in.

The logical memory address is saved in a binary format. To verify that you can read in the correct sequence of memory accesses, you can first print out the address that you have analyzed. You can test your program with the given simple file `part1-test-sequence`, where the first address should be 0x0000000000000044 and the second one should be 0x0000000000000224.

For each logical address in the sequence file, you will use the page-table given above to perform the address translation and generate a corresponding physical address that will be printed out to the file specified as the 2nd cmd-line parameter to `assign4-part1.c`. The outfile must have the same format as the given `part1testsequence` file. Each physical address must be written in binary as an 8 byte (`unsigned long`) value.

Once you test your program with the `part1-test-sequence`, and you are sure the program performs correct address translation, use the following `part1-sequence` as the input file for logical addresses sequence to generate the translated physical addresses and put them in a file called `assign4-part1-output`.

```
Note: to simplify Part 1, you can hardcode the mapping from page to frame into an array
before performing any address translation.
```

Part 2: Virtual Memory (70 points)

In this part, you will design a page table and perform physical memory management. You will create two new source files for this part: `phypages.c` and `pagetable.c`, and a new main program named as `assign4-part2.c`, plus any necessary header files. Here, `phypages.c` is used to manage the physical pages and `pagetable.c` will manage the page table for the process.

As implicitly assumed earlier, the first physical frame is reserved for the OS; the other frames are initially free.
You will initially use the following frame allocation scheme:
(1) allocate the physical page in order of frame number, starting from 1, 2, 3, ...
(2) when there are no free physical frames, you will need to use the **LRU policy** for page replacement. That means, the page that is least recently used (accessed) will be allocated to the new request.

Note that, once a frame is selected to be freed, you need to do two things:
(1) First, you should invalidate the old entry of page table so that we don't have two virtual pages pointing to the same physical frame.
(2) Second, you need to initialize a new page-table entry (PTE) to point to the new frame. You may also want to set up a reverse mapping on the frame to the PTE for quick PTE modifications in the future.

If a page is accessed, you must update its placement in the frame list so that it will not be evicted soon (based on the LRU policy).

You should be able to utilize the same function in **part 1** to map virtual addresses into physical addresses. Use this function for translating `part2-sequence` into the output for `assign4-part2-output`.

In `Readme.txt` , report the number of **pagefaults** encountered when translating **part 2's** logical addresses in physical addresses.

Part 3: Making the design adaptive to any situation (20 points, extra credits)
To get the bonus points, you should list whether you have implemented part 3 in your `Readme.txt` file. Also, you should briefly explain how implementation of this part differs from previous two parts and why you think your implementation is correct.

You need a main program named `assign4-part3.c` that must accept the following parameters:

```
./part3 BytesPerPage SizeOfVirtualMemory SizeOfPhysicalMemory SequenceFile OutputFile
```

where the first parameter `BytesPerPage` specifies the number of bytes in each physical frame and virtual page. The second parameter `SizeOfVirtualMemory` is the size of virtual memory in bytes. The third one `SizeOfPhysicalMemory` is the size of physical memory in bytes. The fourth one `SequenceFile` is the name of the file that contains the sequence of logical addresses that need to be translated. The fifth one `OutputFile` is where the translated physical addresses are written.

To test your program's Part 3 functions, you can use the parameters specified in "Part 2". Your program should generate the same output file as that in `output-part2`. In `Readme.txt`, type why you think your implementation is correct.

Programming Environment and Submission:

- You should debug and test your program on one of the Linux machines (`fox01` to `fox04.cs.utsa.edu`);
- You should also have a `Readme.txt`. In this text file, you need to report: a) the status of your program (completed or not; partial credit will be given even the program is not completed); b) List all of the people that you have collaborated with on this assignment. For each person indicate the level of collaboration (small, medium, large). Also write a few sentences describing what was discussed. Indicate whether you were mainly giving help or receiving help; c). comments and suggestions to improve this assignment;
- You should create a tar-ball for all the above files with the name of `assign4-abc123.tar.gz` and upload the file on Blackboard, where abc123 should be your UTSA ID..