Project 2 Programming - Virtual Memory Manager **CS 300, Hong**

All the coding must be done in C language, being able to compile and execute in cs-intro.ua.edu. If you use a MAC, you should take extra care to create an executable C code in cs-intro.ua.edu.

The Project Descriptions to be found in the attached PDF file *VirtualMemoryManager-ProgrammingAssignment.pdf*.
The project has two parts, the first part implements a paging system with TLB. It includes both address translation Handling Page Faults. This is the main part, instructions on testing, how to begin, how to run and collecting statistics are given. After this is part is fulling working, you can proceed to the second part "Modifications". This part adds to the first part by using free page list and implementing page-replacement policy. Again, part 2 builds on the code written in part 1.

Clarifications and Additional Requirements:
(1) The page fault occurs starting from the first page request (yes, in part 1), i.e., it is pure demanding paging.
(2) The free frames are allocated sequentially from address 0, as the fig 9.34 shows.
(3) For TLB replacement, FIFO replacement policy should be used.
(4) For the part 2 Modification, page replacement uses LRU policy. You could use a counter implementation, and implement a simple sequence number as the clock.
(5) output format: (a) for each address translation, print in one line: < logical address, physical address, the signed byte value read from the physical address>. The addresses are integers, the content from backing store is "signed byte". (b) at the end of the program, print Statistics, one metric a line. (c) For part 2 Modification, the outputs follow (a) and (b).
(6) Note, the program should read from the command line about the name of the file that contains the addresses, e.g., the file "addresses.txt". Then, your code will open and read from the file. For example: `./a.out filename`. For the file of `BACKING_STORE.bin`, you can hardcode the filename `BACKING_STORE.bin` in your program.
(7) For part 2 Modifications, add to the program for reading a second command line input for the size of physical memory. Thus, if the physical memory is 64, the program runs like: `./a.out addresses.txt 64`
Q/A:
(8) The physical memory starts as all free frames. They are allocated to the referenced pages when program goes.
(9) A logical address is the location to read from the backing store.

Software requirements:
(10) Generate two separate pieces of source code in C: (a) one for the first part, name it as *YourFullName*_mm1.c; and (b) the second for the modification part (with page replacement), name it as *YourFullName*_mm2.c. Yes, Part I code can and should appear in Part II code. This helps us to identify the sources of errors when necessary.
(11) Testing: you should create your own small test cases (say, 30 page addresses and resulting physical address) to debug and test your code. You can also use the given files to test your results.
(12) Comment lines: In your two source files, the first line must be a comment line with your full name and CWID. Also add comment lines for this project, optional, but recommend.

Submission requirements:
1) Submit to Blackboard "Proj 2" before deadline.
2) Submit two separate pieces of source code in C: one for the first part and the other for the modification part. Name them according to the convention given in (10). You **must not** zip the source files. points off, if so. Don't use Makefile.
3) Compiling: Grading will use gcc -Wall filename -std=c99 at the cs-intro.ua.edu server. You should use the same command at the same server to compile your two source code files. Errors leads to points drop.
4) 10% points are allocated to the submission and compiling.

Delay policy:
Follow the course policy. Each one-day delay will drop 20% points. You should start your projects early to avoid potential last minute issues that cause late turn-in.

Grading will count:
(1) Submission and Compilation; (2) Outputs from test cases for the two parts: address translation, statistics of page fault and TLB hit; (3) Code inspection, including FIFO and LRU implementations.