**Project 3: Memory Address Translation**

The goal of this project is to practice the address translation that is usually performed by the hardware and practice the management of a page table. In this assignment, you are asked to design a simulator that simulates memory management inside the user space.  We consider a tiny computer system (even much smaller than Raspberry Pi) that supports up to 1KB physical memory and **12-bit virtual addresses**. The size of a virtual page and a physical page is **128 bytes**.

**Part1: 40 points**

Assume that the page table of the process is shown in the following diagram.

   A diagram of a physical frames

Description automatically generated  
Note that only a few entries of the process' page table are present, as we will only use the first seven entries in part1.  
  
Write a program called mempart1.c for this part. The program will take only one parameter, the input sequence filename, which contains a sequence of virtual memory accesses (addresses). Here, each address is stored as 8 bytes (unsigned long type). Your program should read and analyze each virtual address and translate it into the corresponding physical address base on the given page table (shown in the above diagram). Note: to simplify your implementation, you may hard code the page-to-frame mapping in your program.

In the input file, each memory address is stored in a binary format. You should not attempt to to view the file by the command "cat", "less", or any text editing application. Instead, you may use "hexdump input\_file" command to view the binary contents. You can test your program with the given simple    [part1test](https://bb-csuohio.blackboard.com/bbcswebdav/pid-7269901-dt-content-rid-79870831_1/xid-79870831_1)   [Click for more options](https://bb-csuohio.blackboard.com/webapps/blackboard/content/listContentEditable.jsp?content_id=_6912385_1&course_id=_175168_1#contextMenu) , where the first address should be 0x00000044, and the second one should be 0x00000224. Physical address for these two virtual addresses should be 0x0144 and 0x01A4.

Once you test your program correctly with the above simple test file, which contains only 8 virtual addresses, you should use this larger    [part1sequence](https://bb-csuohio.blackboard.com/bbcswebdav/pid-7269901-dt-content-rid-79870832_1/xid-79870832_1) [Click for more options](https://bb-csuohio.blackboard.com/webapps/blackboard/content/listContentEditable.jsp?content_id=_6912385_1&course_id=_175168_1#contextMenu) as the input file and place all the physical addresses in another file called "part1out". Then you can use the md5sum utility to compute the corresponding checksum, via running "md5sum part1out", and you only need to write the checksum into a file called "p1result.txt". Note: make sure you write prefix "0x" in front of each physical address (2 bytes for each address as shown above) and one physical address in each line (make sure you write a newline character "\n" at the end of each line, including the last line, and leave no space between, or your md5checksum won't be correct).

**Part 2: 50 points**

In this part, you will design the page table and deal with allocations of physical machines on the same machine as discussed above. You will create two new source files: phyframe.c and pagetable.c, as well as a new main program, named "mempart2.c", plus any necessary header files. Here, phyframe.c is used to manage the physical frames, while pagetable.c will hold all functions that manage the page table for the process.  
  
For this part, we will also assume the first physical frame is reserved for the OS, while other framess are initially available for the process. To manage physical frames, you will use a simple physical page allocation scheme: You will allocate each physical page in the order of frame number initially, starting from 1, 2, 3, .... If physical frames are available, you will always allocate from these available frames at first. Once there are no free physical frames, you will need to use LRU page replacement policy to choose a physical frame to be replaced, which means the least recently used page will be the first to be evicted.

First, you should update the old page table entry such that no two virtual pages are pointing to the same physical frame. In reality, it is better to quickly locate which page table entry is actually pointing to this physical frame, typically called as ``reverse mapping''. However, you may search the whole page table of all active processes (one process in the assignment) to find out this, with significant performance overhead.   **You must design and implement a reverse mapping scheme. Otherwise, you will automatically lose 15 points.**

Second, you should change the page table entry of the target page to point to the frame. Note that you should also set up the reverse mapping correctly.  
  
Note: you can use any mechanism discussed in the class to implement the strict LRU policy, such as counter mechanism, but not second-chance algorithm. Importantly, you should update the corresponding information upon each access.  
  
The input sequence of the program is the same as that used in Part 1. Thus, you should be able to utilize the same parser to obtain the corresponding virtual address, then translate it to a physical address with your implemented page table.  
  
In the end, you can use the same function as in Part1 to output the translated physical address sequence into an output file. You are provided a part2 input file,    [part2sequence](https://bb-csuohio.blackboard.com/bbcswebdav/pid-7269901-dt-content-rid-79870833_1/xid-79870833_1)   [Click for more options](https://bb-csuohio.blackboard.com/webapps/blackboard/content/listContentEditable.jsp?content_id=_6912385_1&course_id=_175168_1#contextMenu) . Similar to Part1, you only need to report the md5sum of your output file. and write it to p2result.txt. In addition, your program should report the number of page faults encountered for the given access sequence, and this number should be reported in p2result.txt file too.

**Write-up: 10 Points**

You are required to write a README document (in TXT format only) that describes your project design detail and the execution sequence (with the commands). In particular, please explicitly state which part, if there is any, does not work and the possible reasons why that module does not work. For those working modules, please give a brief (in short) sample output.

**Submissison:**

* Create a folder with the name of your group name, and concatenate with "\_p3".  For example, if the group is "gruop1", then the folder should be "group1\_p3".
* Copy all source files, \*.c and \*.h, into the folder you created.  Also copy the README and makefile to that folder.
* Log in grail server, change the directory to the parent of the folder (created above), and run the following submission command (suppose the submisison folder is "group1\_p3"):
  + turnin -c  cis345w  -p  p3   group1\_p3

If the submission is successful, you should see:

Compressing submitted files... please wait Your files have been submitted to cis345w, p3 for grading.