Capstone Assignment:
Memory Management (adapted from F. Ercal, Ohio State U.)

In this assignment, you are tasked with simulating a virtual memory management system.

Broadly speaking, your simulation will have the following responsibilities:
1) Simulate a paging system
2) Implement three different page replacement algorithms
3) Implement a variable page size
4) Implement demand/prepaging
5) Record page swaps during a run

Let's discuss each phase of this assignment in turn.

1: Simulate a paging system

This simulation will use the idea of a 'memory location' atomic unit. The pages in our simulation will be expressed in terms of this idea. Thus, if our page size is 2, we have two memory locations on each page.

The free-frame buffer space, in our program, will be **512** memory locations big.

Supplied with this assignment are two files, *programlist*, which contains the list of programs that we will be loading into main memory, and *programtrace*, which contains a deterministic series of memory access that emulates a real systems memory usage (additionally, program scheduling is implicit through use of this file). Both of these are given in terms of memory locations as well. The former in terms of size, the latter in terms of which location is requested.

You will need to create page tables for every program listed in *programlist*. Each page in each page table will need to be given a unique name or number (with respect to *all* pages in *all* page tables) so you can quickly determine if it is present in the frame table or not.

Once you have these, you will perform a default loading of memory before we start reading *programtrace* commands. That is, we will load an initial resident set of each program's page table into the frame table.

Finally, you will need to begin reading in *programtrace*. Each line of this file represents a memory location request within a program. You will need to translate this location into the unique page number that you stored in the page tables you made later, and figure out if the requested page is in the frame table or not. If it is, simply continue with the next command in *programtrace*. If it is not, record that a page swap was made, and initiate a replacement algorithm. (Page swaps are the metric we will use to compare various page replacement methods).

2: Implement three different page replacement algorithms

2nd Chance LRU policy

Use the simple version of this algorithm with only one reference bit. This can be found in the text, or the slides.

First In, First Out(FIFO)

The oldest page in memory will be the first to leave when a page swap is performed.

Least Recently Used

The page with the oldest access time will be replaced in memory when a page swap is performed.

3: Implement a variable page size

This affects not only how many pages each program will take up, but also the number of frames in the frame table. If the page size is 2, our frame table will have 256 available frames. This simulation should be able to use page sizes that are powers of 2, up to a max of 16.

4: Implement demand paging and prepaging

Demand paging replaces 1 page with the requested page during a page fault. Prepaging will bring 2 pages into the frame table for every swap: the requested page and the next contiguous page. If the requested page is the last page in the page table, place the next frame into the free-frame list.

5:Record page swaps

Anytime we read a memory location, translate it to page number, and do not find it in the frame table, that means we need to initiate a page swap. We will record this in a counter form during the run of the program. It will be used as a metric of each algorithm's performance. This makes sense: if a particular algorithm is using the disk less to swap pages into memory, the whole system will be running faster

Write a Report:

Once you have implemented all of this, please try running each algorithm (2nd chance LRU, LRU, FIFO) with page sizes of 1,2,4,8, and 16. Obtain simulation results for both demand paging and prepaging. In other words, you will obtain 10 measurements (runs) for each page replacement policy (LRU, clock, FIFO). Plot all these results on a graph (x:page size, y: page swaps) which will have 6 curves altogether (each with a different legend). Write a 1-2 page report detailing your findings, including a discussion of complexity of each algorithm vs. its performance benefit. Also, explain how prepaging affects the performance.

Implementation Details:

- We will be implementing global allocation of frames. In other words, all 3 replacement algorithms will have a global scope and the number of frames held by a process will change dynamically (not fixed!) over the lifetime of the process.
- You maybe use the programming language of your choice. Use appropriate structures to store your page tables and your frame table.
- You can name your pages in any unique fashion you like, but numbers are pretty easy. After you finish parsing program 1 which has pages 1...n, don't restart your number. Program 2 should have pages n+1.....m , program 3 should have m+1....o.. etc
- To look up a memory location, simply divide its number by page size. Thus, Program 0's 120th memory location in a pagesize=4 system would be on page 30. 121 would be 30.25(second location on page 30; simply take the floor or integer division). This will give you the page number relative to the process.
- Once you have that, you can look up absolute page number (some unique identifier that may or may not coincide with the relative page number). Go to program 0's page table, and look in spot 30 for that page's unique identifier. Then see if it's in the frame table.
- For LRU and FIFO, time() and clock() functions are not sufficiently sensitive to timestamp memory accesses. Use an external library if you like, or simply keep a global counter that keeps track of memory accesses. This will be a relative measure of age. In real operating system, this counter would have some kind of hardware implementation. Keep in mind this counter may grow very large. Make it unsigned and long (if your language of choice supports types) to give it room to grow.
- Please make all options settable on the command line. A typical command line should look like this:

```
./memorysimulator programlist commandlist 2 lru d
```

- Please check for usage errors, and print a nice error. Do not let segfaults just print out if the program is run with no arguments. Also, don't just check for a minimum argument list. If the arguments don't make sense, print an error and quit.
- The final output should be the number of page faults generated.
- A grading rubric for this assignment is posted.