Project 3: Virtual Memory Paging

Requirements:

Your team will emulate virtual memory managers, using the following replacement policies:

1. First In, First Out (FIFO).
2. Least Recently Used (LRU).

You shall use standard C++ Object-oriented programming. You may use either Linux or Windows development platforms.

An Interface (Abstract Base Class) for the Memory Manager will be provided as a header file, and your manager MUST implement this interface. **YOU MAY NOT CHANGE THIS CLASS!** This class also provides an enumeration of replacement policies that MUST be used as arguments to the constructor for your manager. Arguments for the constructor shall be as follows:

ReplacementPolicy policy, //where ReplacementPolicy is the enum defined in the provided header
unsigned int pageSize, //2^ pageSize = the physical frame/virtual page size to manage (bytes)
unsigned int numFrames, // total number of physical frames. Phy mem size = numFrames * 2^pageSize
unsigned int virtualAddressSpaceSize, // 2^virtualAddressSpaceSize = total virtual address space (bytes)

Assertions:
virtualAddressSpaceSize >> pageSize
2^virtualAddressSpaceSize >> numFrames * 2^pageSize

Key Methods in Virtual Memory Manager interface:

```
/** This is the method the test bench will drive in emulating memory management.
*          Your memory manager should return the physical address corresponding to the given virtual
*          address. This function must NOT return until any page swapping is completed, if necessary.
*          This function is to effect page swaps by calling the other key function (defined below)
*/
virtual unsigned long long memoryAccess(unsigned long long address) = 0;

/** This is the method your memory manager should call to swap pages.
*          This function has been instrumented by the TAs to report memory system performance.
*          @param frameNumber the physical frame to write to swap file (write is emulated)
*          @param pageNumber the (virtual) page number to read from swap file into the given frame
*/
void swap(unsigned int frameNumber, unsigned int pageNumber) {
///@todo ta instrumentation to go here
///students should instrument memorymanager performance in their own class for their internal verification
/// or may modify this code for their testing purpose; however the TAs instrumentation will used
/// for determining grade basis.
          numSwaps++;
}

/** Report the to-date number of page swaps, used by test benches to quantify performance.
*          @returns the number of page swaps
*/
unsigned long long& numberPageSwaps() { return numSwaps; }
```

You and your team are to implement (1) a memory manager of the correctly interface. Students must also (2) implement your own test bench to self-verify your manager works as required, evaluate their work, and to ensure replacement policies are correctly implemented.

Projects will be graded by running the students' memory managers against the TAs testbenches.

Teams of ***two-three*** students for this effort are required. Teams may be formed from registered members of either section.

What and When to submit: Your code archives (.zip or .7z) are to be submitted on Canvas by 11:59 pm Friday, April 28th. DO be sure all of your team's names are on both files in the comment headers.

Grading: If the TAs are unable to compile your projects, you will have a mandatory 50% penalty – that is, 50% is the maximum possible score for a good effort that doesn't compile. TAs will scale grades between 50% and 100% for those teams whose projects compiles and runs based on (20%) how well document and designed your source code is and (30%) how many of the TA's testbench scores are correct. The average project score is expected to be between 70 and 80%.

You ARE allowed to exchange testbenches by posting your testbenches to the Canvas discussion. You ARE NOT allowed to share source code! ANY case of plagiarism between teams will result in a zero being entered for all members of all teams involved on the first offense, and on the second offense the teams will be failed for the course. Reports of academic misconduct will be made in all cases.