Background

We provide you with four memory traces to use with your simulator. Each trace is a real recording of a running program, taken from the SPEC benchmarks. Real traces are enormously big: billions and billions of memory accesses. However, a relatively small trace will be more than enough to capture their memory access patterns. Each trace consists of only one million memory accesses taken from the beginning of each program.

Each trace is a series of lines, each listing a hexadecimal memory address followed by R or W to indicate a read or a write. For example, **gcc.trace** trace starts like this:

0041f7a0 R   
13f5e2c0 R   
05e78900 R   
004758a0 R   
31348900 W

Each trace is compressed with gzip  
it with a command like this:

> gunzip –d gcc.trace.gz

Tasks

Task 1: Simulator

Your job is to build a simulator that reads a memory trace and simulates the action of a virtual memory system with a single level page table. The current simulator fixes the pages and page frames size to 4 KB (4096 bytes). Your program should keep track of what pages are loaded into memory. The simulator accepts 4 arguments as follows:

* the name of the memory trace file to use.
* the number of page frames in the simulated memory.
* the page replacement algorithm to use: rand/lru/esc
* the mode to run: quiet/debug

If the mode is "debug", the simulator prints out messages displaying the details of each event in the trace. The output from “debug” it is simply there to help you develop and test your code. If the mode is "quiet", then the simulator should run silently with no output until the very end, at which point it prints out a summary of disk accesses and the page fault rate.

As it processes each memory event from the trace, the simulator checks to see if the corresponding page is loaded. If not, it should choose a page to remove from memory. Of course, if the page to be replaced is dirty, it must be saved to disk. Finally, the new page is to be loaded into memory from disk, and the page table is updated.  As this is just a simulation of the page table, we do not actually need to read and write data from disk. When a simulated disk read or disk write must occur, we simply increment a counter to keep track of disk reads and writes, respectively.

Most of the input (reading a trace), simulation counters and output messages has already being implemented in the skeleton files provided for you.

The skeleton reads the parameters, processes the trace files and for each access it generates a page   
read or write request. Your job is to complete the simulation of the memory management unit for   
each replacement policy:

* **rand**replaces a page chosen completely at random,
* **lru**always replaces the least recently used page,
* **clock**performs the replacement algorithm described in the textbook section 22.8.

You should start thinking how you can keep track of what pages are loaded, how to find if the page is resident or not, and how to allocate frames to pages. Some short traces (trace1, trace2 and trace3) will be used in the testing script and are provided to facilitate local testing of your code.