Memory Management

Laboratory 5
Duration: 2 weeks

This assignment will help us learn about how page table contents are maintained, and how different page replacement policies work.

You will need to develop a simulator of a Virtual Memory Manager in C++. The simulator must take in the following command line arguments: `<page-size>` `<number-of-memory-frames>` `<replacement-policy>` `<allocation-policy>` `<path-to-memory-trace-file>`.

Experiment with:

- Different page sizes and number of memory frames
- 4 replacement policies: Optimal, FIFO, LRU, Random
- 2 allocation policies: Global and Local. In the case of Local, you can follow the "equal allocation" policy.
- 2 different memory trace files available here: one is a small trace (for testing and debugging) and the other is a longer one (for analysis). These are collected from 4 different real world applications. Assume that these 4 processes are running simultaneously, all of them making accesses to the memory. Each line in the trace file corresponds to one access. The lines are listed in the order accessed by the processes. The format of one line is `<process id>,<virtual address>`. Assume a 64-bit system. Tip: use uint64 type for addresses in your code (reference; note that the documentation varies with different versions of C++ – use the "Standard Revision" dropdown in the top-right of the page).

Study the different combinations of the above parameters in terms of the number of page faults.

Implementation:

- Design a PageTable data structure, and implement it as a class. Create 4 objects of this data structure, one for each process. You can follow the Hashed Page Table approach (see Section 9.4.2 in the textbook). You can use maps available in the C++ standard library.
- Design a FrameStatus data structure, and implement it as a class.
- For every access mentioned in the file, search for the virtual address to physical address mapping in the corresponding page table. If not found, increment the page fault counter (maintain a global page fault counter, as well as per-process page fault counters). Then add the mapping to the page table by finding a free frame if available, or by evicting a frame based on the replacement policy being followed.

How is # page faults varying w/ increase in size of RAM & # pages

Submission:

- Source code with suitable makefiles
- Report containing observations in the form of graphs and their analyses.

4 apps running simultaneously requiring 64 bit virtual address to access.

Simulate MMU should do mapping.

- Assume everything is there in swap space.
- count # faults for sequence of accesses i/p file

- Make page table data structures (one w/ 4 objects for each
- Metadata about physical memory
- Maintain page table for each process.