Adrick Malekian

Memory Simulator Project Report

Abstract

For this project, students were asked to “mimic” the way memory is handled with the use of paging and proper page replacement algorithms. First and foremost, the vital hardware components need to be mimicked first, then proceeding the completion of those mimicked hardware components is the software that handles the actual logic for page replacement and printing the required fields.

Design

For the tasks described above, I went ahead and constructed four different classes to handle the simulation. The four classes are the TLB, Page Table, Physical Memory, and a Memory Manager class which hold metadata for memory and the statistics. The TLB class holds its size and entries and one can perform a lookup or add an entry by calling one of the classes functions. The page table also holds its size and entries and one can call the add entry function to add an entry to it. The physical memory class is what contains the actual values and frames along with the page replacement algorithm chosen. In this class you can load a page, read a frame, load from the backing store, and get a value from memory. Lastly the memory manager class just contains metadata that is used among all the classes and in the PRA algorithms. Each class is called within my main function defined as “memSim”" which handles the page replacement and executes all necessary code.

Sequence of Operations

First the program starts in main where the arguments from the command line are parsed and sufficient command line argument error checking is also handled here. Then I proceed by instantiating the four classes which will be used in the “memSim” program. After that is complete, the program is prompted to read and parse from the address list and store the addresses to be processed in a list which lies in the Memory Manager class. Once this is done, the main function (memSim) is called with the four classes passed as parameters. Inside memSim, address is parsed into binary and then the page and offset are extracted from it. Moving forward, the TLB checks to see if the entry exists, if not then it will be considered a soft miss and doesn’t need to load from the backing store. If it's not in the TLB or Page Table, then the program loads the data from the backing store into the next available free frame depending on the page replacement algorithm defined in the command line arguments (FIFO if no PRA given). Once that is completed, it will print all of the metadata for that particular address and then loop into the next address. Once all addresses are parsed and print their required metadata, the Memory Simulator prints statistics about memory access such as page faults, hits, misses, etc. Then once that is complete, the program terminates. Overall the use of an ordered dictionary makes it easy for the TLB to pop in FIFO format. However, for designing the LRU page replacement algorithm, a “pagesAccessed” list keeps track of the pages already accessed to see which one was the least

recently used. My implementation for OPT basically looks in the future and checks to see what the furthest away page. If all of the pages in the future aren't related to the pages in the past, it will just pop the first key as the frame to evict. This is the overall functionality of the different components inside of the program and how they intertwined to mimic virtual memory.