Ritvik Durgempudi, Joel Puthankalam, Tymon Vu

Project 3 Report

For this project, we represented each of the main memory structures in the form of classes, and ultimately used the main() method to parse through the command line when the executable is called and display the correct output by parsing through the text file (reference sequence) of addresses that we are given, and we call reference to the ("backing store"/disk device) in main memory to retrieve the correct data.

First after parsing and getting the info from the input on the command line, it will declare a TLB, page table, and memory that can be globally accessed. It will then retrieve each address from the reference sequence and calculate the page number and offset number and try finding it within the TLB. If this fails, it will try to look in the page table, and if that fails, it will load from backing and put it into memory. Depending on the algorithm the user put in at this point, it would either check the next couple of addresses to determine the queue order for page replacement ("OPT") or update the memory order if we had a hit in either page table or TLB ("LRU"). It then retrieves the information for the given address in the classes and finally prints out each addresse's value, frame number, and entire frame, as well as given data.

<u>TLB</u>: Represented as a class that when initialized, has attributes of number of maximum entries, a dictionary to store the entries, and an ordering array that is used as a FIFO queue that is used to replace any pages if we run out of space.

> **-find(page_num)** : Given a page number as a key, it tries to initially find a key/value pair within the dictionary of stored entries, if it fails to find it, it would count as a miss and return None. If it finds it, it would be recorded as a hit and returns the frame value.
> **-insert(page_num, frame_num)**: Given a page number and frame number to be inserted, it checks if the frame number already exists in the dictionary of entries. If it does exist, it will pop off that element from the dictionary and remove it in the replacement queue. Afterwards, it will insert a new key/value pair in the TLB with the key being the page number and the value being the frame number and add the entry number to the end of the queue. If the length of the table is greater than the maximum number of objects it can hold, it will pop the first element that it puts in based on the queue.

<u>Page Table</u>: Represented as a class that when initialized, creates an empty dictionary of entries inside the page table that is represented as key(page numbers)/value(loaded bit, frame numbers) pairs.

> **-find(page_num)** : Given a page number, it will try to retrieve the frame number in the dictionary of entries. If it does not exist, it will return None.

-insert(page_num, frame_num): Given a page number and frame number, it will check if the frame number is active in the value list of the dictionary, and if there are duplicates, it will set every instance of the loaded bit in the page table as inactive.

<u>Memory</u>: Represented as a class that when initialized, creates an empty dictionary of frame entries represented as (frame/data), the queue order based on the chosen replacement algorithm, the backing store, the current frame open, and the actual replacement algorithm.

-find(frame, offset) : Given a frame and an offset, it will try to lookup that frame if loaded into memory and calculate the value to return based on the calculated offset, converting it to a signed decimal.

-insert(frame, data) : Given a frame and data, it will add it to the dictionary as a key/value pair and update the order.

-update_order(remove, insert) : Called within the main method to update order for LRU algorithm.

-default_order() : Will set the default order if all the frames that were initially empty, have been filled up

-load_from_backing(page_num): Will open up the backing and seek 256 bytes based on the page number and tries to find an available frame for it and insert the loaded frame into main memory