CSCI 6730 Operating System
Project 3

Xiaodong Jiang
Department of Statistics
University of Georgia

In this project, we implement a virtual memory simulator to understand the behavior of a TLB, page table, and the page replacement algorithms. There are mainly three parts required to do, including simulating a Page Table, three Page Replacement Algorithm, and two-way associative TLB table with LRU algorithm for replacement. These three parts are highly related to each other, so we will first explain the framework on the top as something like a flow chart.

When a request comes in, we have the following hierarchical process.

1. **Check TLB**
   (1.1). **If TLB hit**, its great!
   We should update the TLB and page table associated with the page replacement algorithm, LRU and CLOCK.
   For TLB, we need only update the LRU index pointer;
   For page table (PT), we should change the dirty indicator if the request is "W", and change the LRU, CLOCK related index.
   (1.2). **If TLB mis**s, don't worry, let's **check PT**.
      (1.2.1). **If PT hit**, it's good!
      We now should update both PT and TLB.
      For PT, since there is a page hit, we should change the dirty indicator if the request is "W", and change the LRU, CLOCK related index.
      For TLB, we need also update the LRU index pointer, since we have to make sure the cache(TLB) is up-to-date for PT.
      (1.2.2). **If PT miss**, we have some trouble, which leads to a page fault.
      Here we don't have a hit from both PT and TLB.
      And there are two cases here: when PT is full, or not.
         (1.2.2.1). If PT is full, which means we achieve the MAX_FRAME
         We should do the page replacement to find a victim frame.
         For PT, we should swap out the victim frame to disk, and swap in the targeting frame back and update in PT, if the request is "W", we need also update the dirty index. Then do the usual updates related to the LRU and CLOCK algorithm.
         For TLB, we should also update to make sure it's up-to-date.

Here is a little bit complicated. If the victim frame only appears in PT, no problem, we can simply do the swap out/in, and update PT and TLB; but if the victim frame appears in both PT and TLB, when we update the PT, we also have to update the TLB - remove/update the victim frame in TLB! In other words, every time we update the PT, don't forget to update TLB accordingly!
(1.2.2.2). If PT not full, good news! We can simply update the PT and update the associated LRU and CLOCK algorithm index/pointer. As well as updating the TLB.

We implemented the following functions

1. is_tlb_hit()
   Check if there is a TLB hit, if so, update as in **(1.1)**
   If not, return -1 and continue
2. is_page_hit()
   Check if there is a page hit, if so, update as in **(1.2)** and **(1.2.1)**
   If not, return -1 and we have a page fault
3. page_fault_handler()
   Achieve as in **(1.2.2)** including **(1.2.2.1)** and **(1.2.2.2)**
4. FIFO(), page replacement with fifo algorithm
5. LRU(), page replacement with LRU by using doubly linked list provided by Professor Lee.
6. CLOCK(), page replacement with CLOCK.

Results:
We test with small and large files updated in ELC, and verified the results, and we here report the results for CLOCK algorithm with small and large input.

large_input, with CLOCK:
Request: 10000
Page Hit: 2475 (24.75%)
Page Miss: 7525 (75.25%)
TLB Hit: 611 (6.11%)
TLB Miss: 9389 (93.89%)
Disk read: 7525
Disk write: 4063

Small_inut with CLOCK

Request: 20
Page Hit: 6 (30.00%)
Page Miss: 14 (70.00%)
TLB Hit: 4 (20.00%)
TLB Miss: 16 (80.00%)
Disk read: 14
Disk write: 5