CSci 4061: Introduction to Operating Systems

Project 3 - Virtual-Memory Manager **due: Tuesday March 25th, 2020**

Ground Rules. You may choose to complete this project in a group of up to three students. Each group should turn in **one** copy with the names of all group members on it. The code must be originally written by your group. No code from outside the course texts and slides may be used—your code cannot be copied or derived from the Web, from past offerings, other students, programmer friends, etc. All submissions must compile and run on any CSE Labs machine located in KH 4-250. A zip file should be submitted through Canvas by 11:59pm on Tuesday, April 2nd. **Note**: Do not publicize this assignment or your answer to the Internet, e.g., public GitHub repo.

Objectives: The main focus of this project is to implement (1) an algorithm that translates virtual addresses to physical addresses using two-level page tables (PTs) and (2) a TLB (Translation Lookaside Buffer) that implements the "First In First Out" policy for page replacement. The two parts should be done by implementing the functions (with "TODO") in `vmemory.c`.

1 Translate Virtual addresses to Physical Addresses

1.1 The page table structure

Use a two-level page table to translate virtual addresses to physical addresses assuming the following specifications about the PT. We assume 32-bit physical and virtual addresses. The first level of page table contains 1024 page-table entries (PTE), and each PTE further points to the bases of the second-level page tables. Each second-level page table contains 1024 PTE that contain the physical frame numbers (i.e., the frame bases of physical addresses). Given a 32-bit virtual address, the 10 most significant bits(MSB), i.e., 22-31 bits, are used to index into the first-level page table and the next 10 bits (i.e., 12-21 bits) index into the second-level page table to determine the corresponding physical address's base. The 12 least significant bits determine the offset within a given page/frame. The page size within virtual memory is 4KB, as is the frame size in physical memory. Offsets can range from 0x000 to 0xfff. The two-level PT management code is in `page_table.o`, exposing only the base address of the first-level page table as the `cr3` variable.

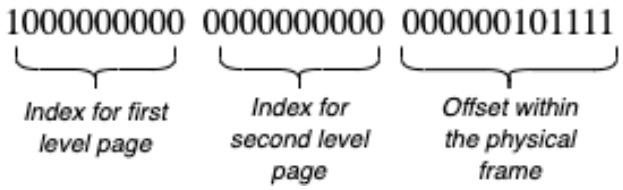

Figure 1: Translation of virtual address

1.2 Translating the virtual address

Using the `cr3` page-table pointer available in `vmemory.c`, translate each virtual address provided in file `virtual.txt`, containing a 32-bit virtual address in the hex format, to its physical address. The

translation should use the two-level PTs as provided. A virtual address is valid if it has a physical frame number in the second-level PTs. Use the physical frame number from the second-level PT and the offset (12 LSB of virtual address) to determine the exact physical address. An invalid entry for a virtual address is called a page fault and should return -1. Specifically, implement the following functions in `vmemory.c`.

1. `int translate_virtual_address(unsigned int)`
2. `void print_physical_address(int frame, int offset)`

Notice that the virtual address translation in **function 1** takes in an `unsigned int` as a parameter to account for the entire range of 32 bits. However, the return type is an `int` to ensure you can return `-1`.
In **function 2**, printing physical address should be in hex format with 8 digits in lower case, zero (if applicable) padded to the left, and preceded by "0x", such as `0x009ea628`. When the frame parameter is `-1`, just print `-1`. The input are the physical frame number and offset parameters.

2 Implementation of TLB

In the second part, implement a TLB. While, in practice, the TLB is typically in CPU, we emulate it as a buffer in the main memory in this project. Your TLB can contain up to 8 frequently used virtual address entries, to avoid page table lookup latency. TLB is filled whenever there is a TLB miss but no page fault in the PT lookup. When the TLB is full, the page that is first in is removed, following the FIFO policy. One way to implement TLB is a 2D int array that implements the queue data structure. The first entry in the TLB contains the 20 MSB of virtual address page number and second entry is physical frame base address. The offset is not needed in the TLB. Specifically, implement the following functions in `vmemory.c`.

3. `int get_tlb_enty(int n)`
4. `void populate_tlb(int v_addr, int p_addr)`
5. `float get_hit_ratio()`
6. `void print_tlb()`

Function 3 takes an `int` parameter containing the virtual address 20 MSB. It should return the physical frame base address mapped to that particular virtual address. TLB miss should return `-1`.
Function 4 should take the 20 MSB of a virtual address, i.e., the virtual page number, and the 20 MSB of a physical address, i.e., the physical frame number, as parameters and populate the TLB.
Function 5 should provide the hit ratio of the TLB, which is the ratio of the number of times the TLB served the page request without invoking the PT to the total addresses translated, so far.
Function 6 should print the state of all the 8 TLB entries to the output file `tlb_out.txt`. Each line in the output file contains two columns delimited by a single space containing the virtual page number and the physical frame number respectively, in hex. (20 MSB should give you 5 digits; the 5 digits should be preceded by `0x`). Initial TLB entry should be `-1`. Each call to `print_tlb()` should print a blank line to separate TLB entries and append its contents to `tlb_out.txt`. Functions 5 and 6 can be called anytime during the program's execution.

2.1 Extra credit

Implement a Least Recently Used (LRU) page replacement policy along with FIFO policy in TLB. Compare the two policies based on history of virtual address requests to TLB. You can assume that the history window to be of size 10 and determine when an appropriate policy is useful. This can be implemented by adding extra functions in `vmanager.c` that are invoked if a command line argument, `-lru`), is specified. By default, without entering any command line arguments, your TLB should run FIFO policy. `print_tlb()` should be a general function that is able to print the TLB maintained by either FIFO or LRU.

3 Deliverables

Students should upload to Canvas a zip file containing their C code, a Makefile, and a README that includes the group member names, what each member contributed, any known bugs, test cases used (we will also use our own), whether the extra credit has been attempted, and any special instructions for running the code.

4 Grading Rubric

- 5% For correct README contents
- 5% Code quality such as using descriptive variable names and comments
- 5% Correct invocation of function calls within `main.c`; the code does not print any extra characters other than the ones required (we will use `diff` to test the results)
- 40% Correct address translations
- 45% Correct implementation of TLB
- 5% Extra credit for implementing LRU policy

5 Example Translation

Consider the virtual address `0x72ae2247` from `virtual.txt` Its binary transformation is `0111001010 1011100010 001001000111`. Its 10 MSB (0111001010) in decimal is `458`. We then look for the base pointer (i.e., the base of the corresponding second-level page table) with index `458` in the first-level page table, and check the content which in decimal is `738`. The entry in second level page table with index `738` is `0x71728`. This is the physical frame base address. The offset of the virtual address is `0x247`. The final physical address is `0x71728000 + 0x247 = 0x72728247`.

Hints & other requirements Don't modify `vmemory.h`; instead implement helper functions via static qualifier. Other than error checking and required prints, avoid any other prints in vmemory.c After using make, run the executable ./vmanager to run the program with FIFO or with an optional parameter (`-lru`) to run LRU policy. A missing specification can be implemented by any reasonable assumption.