Virtual Memory Manager

In this part, you will implement a virtual memory manager similar to the one described with "Chapter 9" above.

Part 1:

The first version of the memory manager will be implemented with the assumption that physical address space is the same size as the virtual address space. Therefore you will not implement any page-replacement policy.

The virtual memory manager will use a TLB (Translation Lookaside Buffer) and a page table. You are required to use a FIFO replacement policy for the TLB. Differently from the specification in Programming Projects, you will use 20 bits addresses instead of 16 bits. The address bits will be divided into a 10-bit page number and a 10-bit page offset.

Part 2:

The implementation in Part I assumes that physical memory is the same size as the virtual address space. In practice, physical memory (PM) is typically smaller than virtual memory. Part II will implement the case when VM is larger than PM. Your implementation for Part II will use 256 page frames rather than 1024 for physical memory. This change will require modifying the provided program so that it keeps track of free page frames as well as implementing a page-replacement policy. We are asking you to implement both FIFO and LRU replacement policies. Add a command line argument to select a policy such as -p 0 for FIFO and -p 1 for LRU.

Compare these two policies with the address streams provided in the project folder.

Chapter 9:

Designing a Virtual Memory Manager

This project consists of writing a program that translates logical to physical addresses for a virtual address space of size 216 = 65,536 bytes. Your program will read from a file containing logical addresses and, using a TLB as well as a page table, will translate each logical address to its corresponding physical address and output the value of the byte stored at the translated physical address. The goal behind this project is to simulate the steps involved in translating logical to physical addresses.

Specifics

Your program will read a file containing several 32-bit integer numbers that represent logical addresses. However, you need only be concerned with 16-bit addresses, so you must mask the rightmost 16 bits of each logical address. These 16 bits are divided into (1) an 8-bit page number and (2) 8-bit page offset.

Other specifics include the following:

• 2^8 entries in the page table

• Page size of 2^8 bytes

• 16 entries in the TLB

• Frame size of 2^8 bytes

• 256 frames

• Physical memory of 65,536 bytes (256frames × 256-byte framesize)

Additionally, your program need only be concerned with reading logical addresses and translating them to their corresponding physical addresses. You do not need to support writing to the logical address space.

Address Translation

Your program will translate logical to physical addresses using a TLB and page table. First, the page number is extracted from the logical address, and the TLB is consulted. In the case of a TLB-hit, the frame number is obtained from the TLB. In the case of a TLB-miss, the page table must be consulted. In the latter case, either the frame number is obtained from the page table or a page fault occurs.

Handling Page Faults

Your program will implement demand paging. The backing store is represented by the file BACKING STORE.bin, a binary file of size 65,536 bytes. When a page fault occurs, you will read in a 256-byte page from the file BACKING STORE and store it in an available page frame in physical memory. For example, if a logical address with page number 15 resulted in a page fault, your program would read in page 15 from BACKING STORE (remember that pages begin at 0 and are 256 bytes in size) and store it in a page frame in physical memory. Once this frame is stored (and the page table and TLB are updated), subsequent accesses to page 15 will be resolved by either the TLB or the page table.

You will need to treat BACKING STORE.bin as a random-access file so that you can randomly seek to certain positions of the file for reading. We suggest using the standard C library functions for performing I/O, including fopen(), fread(), fseek(), and fclose().

The size of physical memory is the same as the size of the virtual address space—65,536 bytes—so you do not need to be concerned about page replacements during a page fault. Later, we describe a modification to this project using a smaller amount of physical memory; at that point, a page-replacement strategy will be required.