COSC 40203 - Operating Systems

Project 6: Virtual Memory Simulator

Spring 2022
Due: 23:59, April 29^{th}, 2022 (no late day extensions)

Overview

This project consists of writing a program that translates logical to physical addresses for a virtual address space of size $2^{16} = 65536$ bytes. Your program will read from a file containing logical addresses and, using a TLB as well asa 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 65,636 bytes (256 frames $\times$ 256-byte frames size)

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. Second, check the TLB for the page number. In the case of a TLB-hit, the frame number is obtained from the TLB. Third, iIn the case of a TLB-miss, the page table must be looked up in the page table. Either the frame number is obtained from the page table or a page fault occurs.

Handling Page Faults

Your program will implement demand paging as described in the notes. The backing store is represented by the file `swapfile.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 `swapfile.bin` 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 `swapfile.bin` (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 `swapfile.bin` as a random-access file so that you can randomly seek to certain positions of the file for reading. It is suggested that you use 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 is described a modification to this project using a smaller amount of physical memory; at that point, a page-replacement strategy will be required.

Test File

Provided is the file `addresses.txt`, which contains integer values representing logical addresses ranging from 0 - 65535 (the size of the virtual address space). Your program will open this file, read each logical address and translate it to its corresponding physical address, and output the value of the signed byte at the physical address.

How to Begin

First, write a simple program that extracts the page number and offset from the following integer numbers:

```
1, 33153, 128, 65534, 256, 32768, 32769
```

Perhaps the easiest way to do this is by using the operators for bit-masking and bit-shifting. Once you can correctly establish the page number and offset from an integer number, you are ready to begin. Initially, it is suggested that you bypass the TLB and use only a page table. You can integrate the TLB once your page table is working properly. Remember, address translation can work without a TLB; the TLB just makes it faster. When you are ready to implement the TLB, recall that it has only 16 entries, so you will need to use a replacement strategy when you update a full TLB. You may use either a FIFO or an LRU policy for updating your TLB.

How to Run Your Program

Your program should run as follows:

```
./vmsim -s swapfile -a addressfile -m mode
```

The option `-s` will be followed by the name of the swap file or backing store file used for this simulation. A sample swap file is provided for you on TCU Online (called `swapfile.bin`). This file is exactly 65536 bytes in size. The option `-a` will be followed by the name of the address file (list of addresses) your simulation will use. This file contains 1000 integers ranging from 0 to 65535. Your program will translate each logical address into a physical address and determine the contents of the signed byte stored at the correct physical address. Use the `char` data type in C to represent a byte. The option `-m` will indicate the mode: `DEMAND`, `FIFO`, or `LRU`.

Statistics

After completion, your program is to report the following statistics:

1. Page-fault rate — The percentage of address references that resulted in page faults.
2. TLB hit rate — The percentage of address references that were resolved in the TLB.

Since the logical addresses in `addresses.txt` were generated randomly and do not reflect any memory access locality, do not expect to have a high TLB hit rate.

Modifications

This project assumes that physical memory is the same size as the virtual address space. In practice, physical memory is typically much smaller than a virtual address space. A suggested modification is to use a smaller physical address space. Use a 128 page frames rather than 256. This change will require modifying your program so that it keeps track of free page frames as well as implementing a page-replacement policy using either FIFO or LRU.

Project Deliverables

1. This is an individual project worth 100 points.
2. Follow the project submission guidelines.
3. Follow the project documentation standards.
4. Your project must be written in C/C++.
5. Your project must use procedures and functions.
6. Your project must have a Makefile and a README file.
7. Submit your project via github classroom.
8. There are no late days for this project.