

**CSCI 3431: Operating System**

**Fall 2018**

**Assignment 4- Virtual Memory Manager**

**Date out: November 18, 2018**

**Due on: December 8, 2018 (11:45pm)**

**Instructions:**

* Final submission should include the code which is compilable and runnable, a 2-3 page report describing the approach (including the pseudo-code), results and discussion, any innovative features added, reasons for failure ( if any) and References (important).
* Cite in the report if you have adapted your algorithm from any web resources/text books/papers. As the solutions to some of these problems are already available in the web, it’s perfectly fine to refer to them and understand the context and their approach. However, I strongly encourage you to write your own code from the scratch. That is the only way you can practice and get more insights into the problem and the OS in general.

|  |
| --- |
| * Final submissions should be a zip file (code and report) and to be uploaded to the Moodle before 11:45pm on the due date. * **Late assignments** will be penalized by 5 points/day for the first three days and 25 points/day from the fourth day onward. * Name your file following this convention: CSCI3431-Assignment1- *<StudentID>*, where *StudentID* is your Banner ID number starting with A. |

* During the evaluation, the students are expected to download the zipped file from the Moodle and show the results on either their laptop or the lab machine. The evaluation will be done in the following two/three recitations or office hours.
* **You may work in groups of two, if you wish to.**

**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 and 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. Your learning goal is to use simulation to understand the steps involved in translating logical to physical addresses. This will include resolving page faults using demand paging, managing a TLB, and implementing a page-replacement algorithm.

**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) an 8-bit page offset. Hence, the addresses are structured as shown as:



Other specifics include the following:

28 entries in the page table

Page size of 28 bytes

16 entries in the TLB

Frame size of 28 bytes

256 frames

Physical memory of 65,536 bytes (256 frames × 256-byte frame 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 as discussed in the lecture (refer to Section 9.3). 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. A visual representation of the address-translation process is:

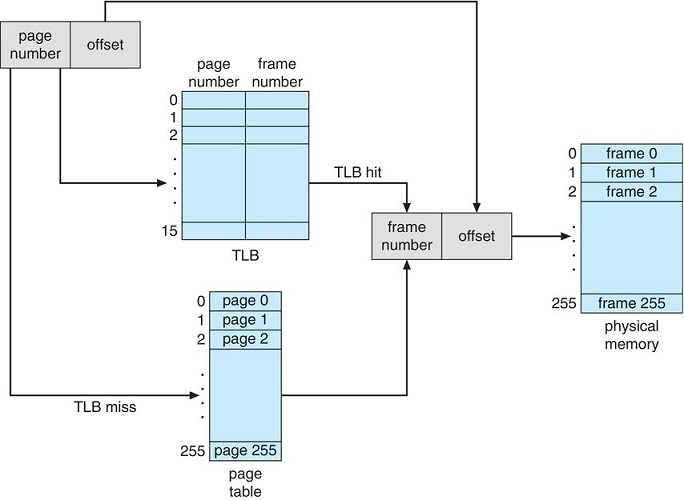


Illustration shows page number and offset leads to page number and frame number, TLB hit to frame number and offset, to physical memory, et cetera.

**Handling Page Faults**

Your program will implement demand paging as described in Section 10.2 of the text book. 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. I suggest using the standard C library functions for performing I/O, including fopen(), fread(), fseek(), and fclose().

In the example program, 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. However, in the assignment, you are expected to work with a smaller amount of physical memory (i.e., physical memory is less than virtual memory); and hence a page-replacement strategy is required.

**Test File**

A file namely addresses.txt is provided, which contains integer values representing logical addresses ranging from 0to65535 (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 based on:



from the following integer numbers:

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

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, I suggest 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 sixteen 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. Note, this has been already taken care in the example program. Understand the logic.

**How to Run Your Program**

The example program can be run as follows:

./a.out addresses.txt

The program will read in the file addresses.txt, which contains 1,000 logical addresses ranging from 0 to 65535. Your program is to translate each logical address to a physical address and determine the contents of the signed byte stored at the correct physical address. (Recall that in the C language, the char data type occupies a byte of storage, so we suggest using char values.)

The program will output the following values:

1. The logical address being translated (the integer value being read from addresses.txt).

2. The corresponding physical address (what your program translates the logical address to).

3. The signed byte value stored in physical memory at the translated physical address.

**Your Task: Page Replacement**

In practice, physical memory is typically much smaller than a virtual address space. As compared to the example, now you have a smaller physical address space with 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 (Section [10.4](https://jigsaw.vitalsource.com/books/9781119320913/epub/OPS/c10.xhtml#c10-sec-0004)) to resolve page faults when there is no free memory.

The skeleton code is provided at . Your task is to implement:

1. LRU via stack
2. LRU via time stamp

**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.

**Suggestions**

* Start early!! If you are not comfortable with the language/concepts, it may take you a bit longer to implement. In the second assignment, I noticed that many students had several questions towards the end. For a
* Backup your work frequently. It’s possible (and most likely) you go try a new feature and your program crashes!
* Document your work properly

**Grading Scheme:**

Program compiles, runs 15

LRU with stack 15

LRU with time stamp 15

Code quality 10

Report 25

Viva (covering virtual memory) 20

**References:**

1. Operating System Concepts, Silbershatz, Gagne and Galvin, 10th Ed.