University of Windsor

School of Computer Science

60-330 Operating Systems Fundamentals - Winter 2018
Course Project

Deadline: **April 6 2018 at 11:59pm**

Rules: This project will be submitted in **groups of 3 students**, and will be evaluated individually. All students will be required to meet a GA in person **after** the submission and explain the project in 10-15 minutes – every student in the group must be able to discuss about the project and answer questions about it (**note that this step is optional**).

Objectives: The aim of this project is to help students understand the main concepts of **memory management** (Chapter 8) and **virtual memory** (Chapter 9). Students will obtain hands on experience in programming such a system along with its features.

Designing a Virtual Memory Manager

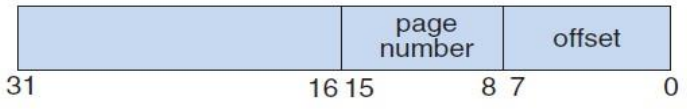
This project consists of writing a program that translates logical to physical addresses for a virtual address space of $2^{16} = 65,536$ bytes. Your program will read from a file containing logical addresses and, using a TLB as well as a page table, will translate each logical address to its corresponding 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. Make sure your program uses fast operations like left/right shift operators.

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 in the figure below:



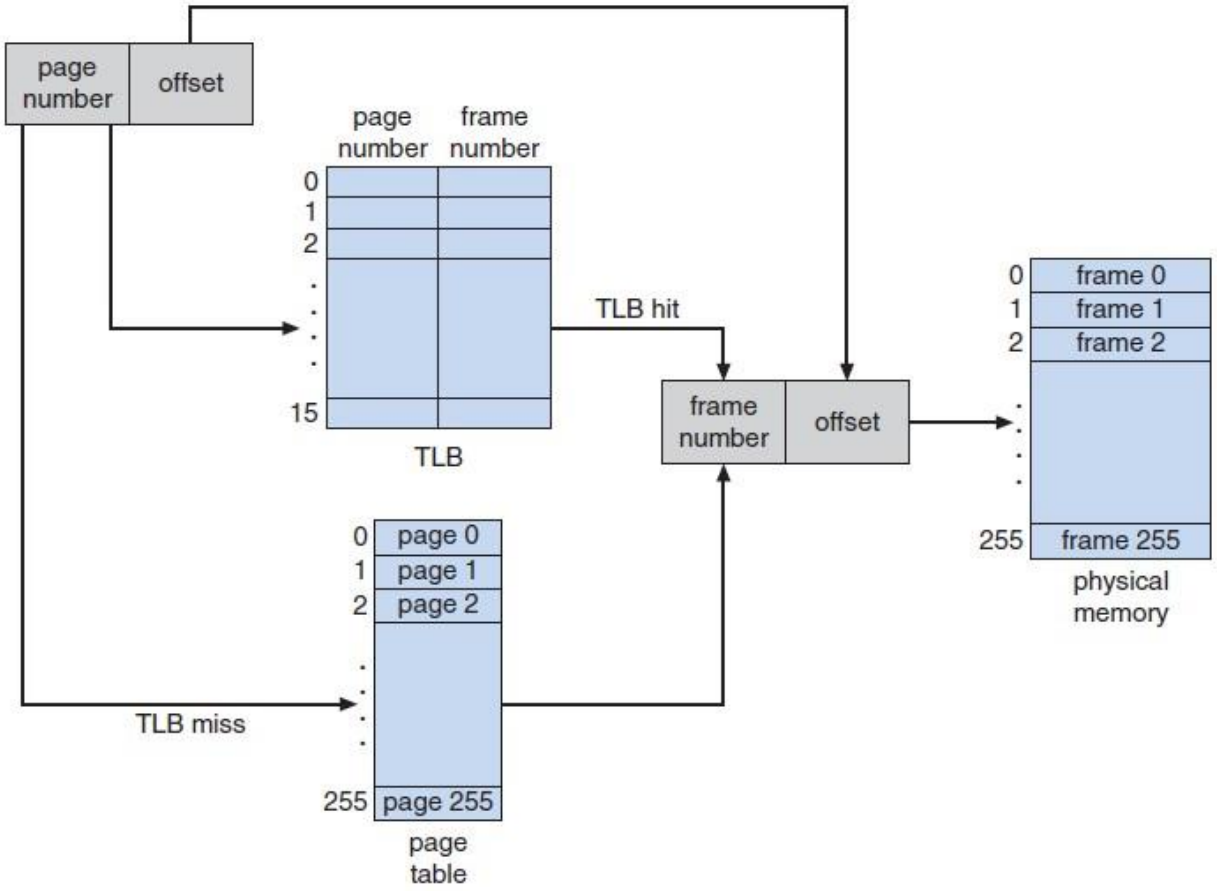
Other specifics include the following:

- $2^7 = 128$ entries in the page table
- Page size of 2^8 bytes
- 16 entries in the TLB
- Frame size of 2^8 bytes
- 128 frames
- Physical memory of 32,768 bytes (128 frames x 256-byte frame size)
- Virtual memory of 65,536 bytes = twice the size of physical memory

Additionally, your program needs 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 outlined in **Section 8.5** of the textbook (the ***Paging* section**). 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 appears in the figure below:



Handling Page Faults

Your program will implement demand paging as described in **Section 9.2** of the textbook (the ***Demand Paging* section**). 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()`.

Specific for the Three-Students-Project: The size of the physical memory is smaller than the size of the virtual address space ---65,536 bytes--- so you need to be concerned about page replacements during a page fault. Since there are only 128 page frames rather than 256, you are therefore required to keep track of free page frames as well as implementing a page-replacement policy using either FIFO or LRU; see **Section 9.4** of the textbook (the ***Page Replacement* section**).

Test file

We provide the file *addresses.txt*, which contains integer values representing logical addresses ranging from 0 to 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 (based on the **first** figure above) 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, we 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 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:

```
./a.out addresses.txt
```

Your 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 C, the `char` data type occupies a bye of storage, so we suggest using `char` values.)

Your program should 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 at the translated physical address.

We also provide the file *correct.txt*, which contains the correct output values for the file *addresses.txt*. You should use this file to determine if your program is correctly translating logical to physical addresses.

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.

3. Any additional statistic of your choice specific to page-replacement.

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. Comment on this in your report.

Submission:

1. You must submit: (i) a report (in PDF or word), which contains the details of the implementation, the run, the input/output of the run, report on the statistics and comments on the TLB hit rate and the page-fault rate, (ii) the source code of your program, and (iii) snapshots of your run. You should upload all these files to the course's Blackboard page, as a **single** zip file.
2. After submission, you should meet the GA for 10 minutes to briefly explain how the project works. **Note that this step is optional**.
3. Add the following note at the beginning of your report: "*I confirm that I will keep the content of this project confidential. I confirm that I have not received any unauthorized assistance in preparing for or writing this project. I acknowledge that a mark of 0 may be assigned for copied/plagiarized work.*" + Name + SID.
4. Any submission after **August 10, 11:59pm** will receive a penalty of 20% for each day up to a maximum of 3 days, with penalty starting immediately after the set due dates and times. After three days, the mark will be zero.
1. Unlimited resubmissions are allowed. But keep in mind that we will consider/mark only the last submission. This means that if you resubmit after the deadline, a penalty will be applied, even if you submitted an earlier version before the deadline.