**Operating System Project**

**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.

Virtual-memory segments are partitioned in units called *pages*, each page is either located in real physical memory (RAM) or stored on disk until it is needed. System uses virtual memory to address more memory than is physically available in the system. The management of memory pages in RAM or on disk is handled by the VMM. Our Project mainly focuses on the conversion of virtual address to physical address.

**Specifics**

Our program will read a file containing several 32 ­bit integer numbers that represent logical addresses. However, you need only be concerned with 1 6­bit addresses, so you must mask the rightmost 16 bits of each logical address. These 16 bits are divided into (1) an 8­bit page number and (2) 8 ­bit page offset.

Hence, the addresses are structured as shown:

|  | Page number | Offset |
| --- | --- | --- |

31 1615 87 0

Address Structure

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

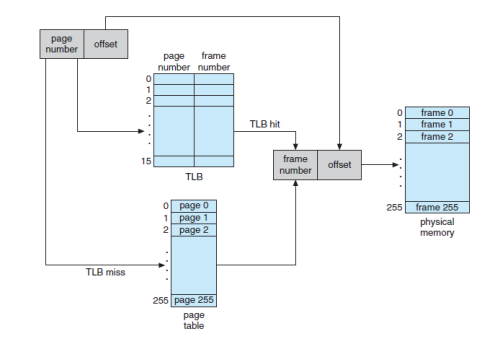
• 256 frames

• Physical memory of 65,536 bytes (256 frames X 256­byte frame size)

Additionally,   our   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**

Our program will translate logical to physical addresses using a TLB and page table as outlined in Section 8.5. 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 address­translation process is:

 A Representation of the Address Translation Process

**Handling Page Faults**

Our program will implement demand paging as described in class. 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 random­access 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.

**Test File**

We   provide   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 (based on Figure 9.33) 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**

Our   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.)

Your program is to 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, Our program is to report the following statistics:

1. PageFault 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.