

# PROJECT

# Virtual Memory System

## 1. PROJECT OVERVIEW

This project implements a virtual memory system (VM) using segmentation and paging. The system manages the necessary segment and page tables in a simulated main memory. It accepts virtual addresses and translates them into physical addresses according to the current contents of the segment and page tables. The system also utilizes a translation look-aside buffer (TLB) to make the translation process more efficient.

## 2. THE VIRTUAL MEMORY

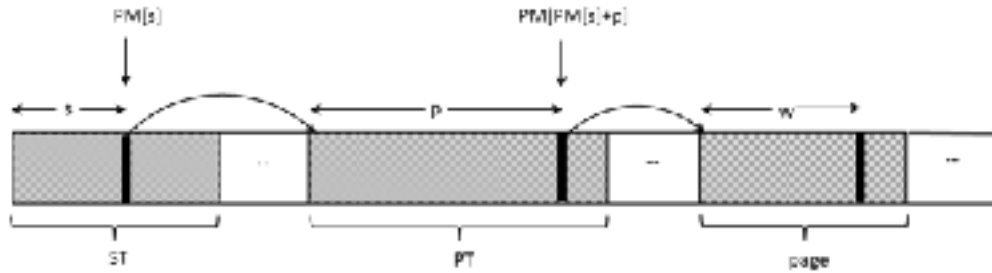
We consider a virtual memory system as described in Section 8.2.3 and make the following assumptions:

- There is only a single process and hence only a single segment table (ST). Each entry of the ST points to a page table (PT), which in turn points to program/data pages (refer to Figure 8-7)
- A virtual address (VA) is assumed to be an integer and thus comprises 32 bits. These are divided into three components: the segment number,  $s$ , the page number,  $p$ , and the offset within the page,  $w$ . The sizes (in bits) of the three components are as follows:  
 $|s| = 9, |p| = 10, |w| = 9$
- Consequently, the size of the ST is 512 words (integers), the size of the each PT is 1024 words, and the size of each program/data page is 512 words. The leading 4 bits of the VA are unused.
- Each entry of the ST can have three types of entry:
  - $-1$  means the PT is currently not resident in physical memory. This would result in a page fault and the missing PT would be loaded from the disk. In this project we are not managing the disk and hence only a message will be generated.
  - $0$  means the corresponding PT does not exist. Consequently, a read access results in an error. A write access causes the creation of a new blank PT.
  - a positive integer,  $f$ , means that the PT starts at physical address  $f$ .
- Similarly, each entry of a PT can have three types of entry:
  - $-1$  means the page is currently not resident in PM. Similar to a missing PT, a message is generated.
  - $0$  means the corresponding page does not exist. A read access results in an error. A write access causes the allocation of a new blank page.
  - a positive integer,  $f$ , means that the page starts at physical address  $f$ .

## 3. THE PHYSICAL MEMORY

- The physical memory (PM) is represented as an array of integers, each corresponding to one addressable memory word. It is implemented as an array of 524,288 integers (= 2MB).
- These are divided into 1024 frames of size 512 words (integers). Consequently the size of a physical address (PA) is 19 bits.
- The ST occupies one frame, each PT occupies two (consecutive) frames, and each program/data page occupies one frame.
- The ST always resides in frame 0 of PM and is never paged out. A program/data page may be placed into any free frame. A PT may be placed into any pair of consecutive free frames.

- The figure below shows the organization of the VM in PM:



ST occupies the first PM frame and thus each  $PM[s]$  is the starting address of a PT. If it is  $>0$  then the PT resides in PM. Similarly, each PT entry  $PM[PM[s]+p]$  is the starting address of a page. If it is  $>0$ , then the corresponding page resides in PM.

- A bit map is used to keep track of which frames are occupied and which are free. The bit map consists of 1024 bits (one per frame) and thus can be implemented as an array of 32 integers. (Normally this would be maintained somewhere within the PM but for the purposes of this project it may be implemented as a separate data structure.)

#### 4. THE ADDRESS TRANSLATION PROCESS

The main task of the VM system is to repeatedly accept VA's and attempt to translate them to the corresponding PA's. The first step is to break each VA into the three components  $s$ ,  $p$ ,  $w$ . The system then accesses the corresponding ST and PT entries to derive the final PA. Depending on the contents of each entry and the type of memory access (read or write), the system acts as follows:

- For a read operation to the VA:
  - If a ST entry ( $PM[s]$ ) or a PT entry ( $PM[PM[s] + p]$ ) equals  $-1$  then output "pf" (page fault) and continue with the next VA.
  - If a ST entry or a PT entry equals 0, then output "err" (error) and continue with the next VA.
  - Otherwise output the corresponding  $PA = PM[PM[s] + p] + w$
- For a write operation to the VA:
  - If a ST entry or a PT entry equals  $-1$  then output "pf" (page fault) and continue with the next VA.
  - If a ST entry equals 0 then allocate a new blank PT (all zeroes), update the ST entry accordingly, and continue with the translation process; if a PT entry equals 0 then create a new blank page, and continue with the translation process.
  - Otherwise output the corresponding  $PA = PM[PM[s] + p] + w$

Note that creating a new page involves modifying the corresponding PT. It also involves searching the bit map for a free frame and updating the bitmap to indicate that this frame is no longer available. Similarly, creating a new PT involves modifying the ST. It also involves searching the bit map for two consecutive free frames and updating the bitmap accordingly. (You may wish to consult Section 7.2.2, which describes the basic implementation of a bit map.)

#### 5. INITIALIZATION OF THE PHYSICAL MEMORY

You will be given the initial contents of PM in an input file. This file specifies the starting address of all PT's and all program/data pages in PM. (ST always starts at address 0.) The file has the following format:

$$s_1 f_1 s_2 f_2 \dots s_n f_n$$

$$p_1 s_1 f_1 p_2 s_2 f_2 \dots p_m s_m f_m$$

Each pair  $s_i f_i$  means that the PT of segment  $s_i$  starts at address  $f_i$ . If  $f_i = -1$  then the corresponding PT is not resident in PM.

For example, 15 512 means that the PT of segment 15 starts at address 512. That is,  $ST[15] = 512$ . Similarly, 9 -1 means that the PT of segment 9 is not resident. That is,  $ST[9] = -1$

Each triple  $p_j s_j f_j$  means that the page  $p_j$  of segment  $s_j$  starts at address  $f_j$ . If  $f_j = -1$  then the corresponding page is not resident in PM.

For example, 7 13 4096 means that page 7 of segment 13 starts at address 4096. That is,  $PT[ST[13]+7] = 4096$ .

Based on the above specifications, the initialization of the PM must proceed as follows:

- Read the series of pairs (line 1) and make the corresponding entries in the ST
- Read the series of triples (line 2) and make the corresponding entries in the PT's
- Create the bitmap to indicate which frames have remained free

## 6. RUNNING THE VA TRANSLATIONS

Once the PM has been initialized, the system is ready to accept VA's and to attempt to translate them into PA's. The VA's will be given in another input file.

Since the translation depends on whether the memory access is a read or a write operation (Section 4), each VA will be preceded by a 0 (indicating a read) or a 1 (indicating a write). Hence this input file has the following format:

$$o_1 VA_1 o_2 VA_2 \dots o_n VA_n$$

where each  $o_i$  is either a 0 or 1 and each  $VA_i$  is a positive integer corresponding to a virtual address. All value are separated by blanks.

Your assignment is to read the file and for each pair  $o_i VA_i$  attempt to translate the VA according to the rules of Section 4. The result of each translation is to be written into another file.

## 7. THE TRANSLATION LOOK-ASIDE BUFFER

To speed up the address translation process, a Translation Look-aside Buffer (TLB) can be employed, as described in Section 8.2.5. For this project we implement a TLB as shown in Figure 8-10 and make the following assumptions:

- The size of the TLB is 4 lines.
- The first field contains the LRU information, which is an integer in the range from 0 to 3. Zero corresponds to the least recently accessed page and 3 corresponds to the most recently accessed one. If a new entry is to be made into the TLB then the least recently accessed entry is replaced.
- The second field contains an integer representing the combined parts  $s$  and  $p$  of the VA.
- The third field,  $f$ , is the starting PA of the frame corresponding to the  $sp$  value.

## 8. RUNNING THE VA TRANSLATIONS WITH THE TLB

For each VM to be translated, the system performs the following operations:

- Break the VA into two components,  $sp$ , and  $w$ , each represented as an integer
- Search the TLB for a match on  $sp$
- If a match is found (called a TLB hit) then:
  - use the corresponding  $f$  from the TLB to form the physical address,  $PA = f+w$

- update the LRU fields as follows:
    - Assume that the match is in line k. Then:
    - Decrement all LRU values that are greater than LRU[k] by 1
    - Set LRU[k] to the maximum, i.e., 3.
- If no match is found (called a TLB miss) then:
  - Resolve the VA as described in Section 4. In case of an error or a page fault, there is no change to the TLB. But if a valid PA is derived then the TLB is updated as follows:
    - Select the line with LRU = 0 and set the LRU value to the maximum, i.e., 3
    - Replace the sp field of that line with the new sp value
    - Replace the f field of that line with PM[PM[s] + p]
    - Decrement all other LRU values by 1
- Output the PA or “pf” or “err” as before. For each translation, indicate also whether a miss or hit occurred in the TLB by outputting “m” or “h” in front of each item.

## 9. SUMMARY OF SPECIFIC TASKS

1. Design and implement a VM system using segmentation and paging as described above.
2. Design and implement a TLB to speed up the address translation process.
3. Design and implement a driver program that initializes the system from a given input file. It then reads another input file and, for each VA, attempts to translate it to the corresponding PA. It outputs the result of each address translation into a new file.