Project: Virtual Memory

Originally created by
Lubomir Bic

Presented/modified by
Michael Dillencourt

University of California, Irvine

Project Overview

- Implement a **virtual memory** system using **segmentation** and **paging**
  - manage segment and page tables in a simulated physical memory
  - accept virtual addresses and translate them into physical addresses
- Simple version:
  - assumes that entire VA space in resident in physical memory (PM)
    - earns partial credit
- Extended version:
  - supports **demand paging**
    - earns full credit
  - implements **translation look-aside buffer** to improve efficiency
    - not required for this project

Principles of VM

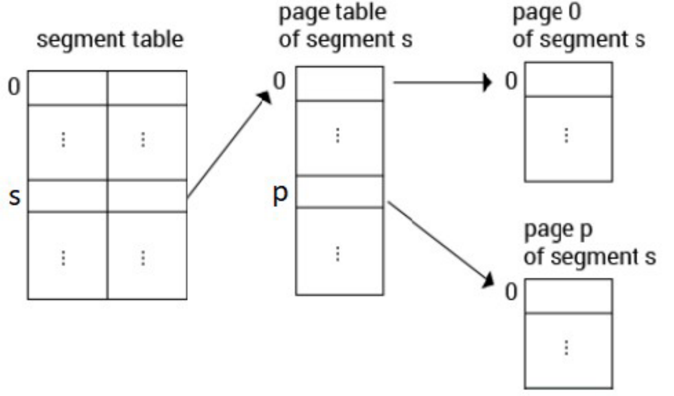
- **VM**: logical address space whose size may exceed size of physical space
  - implementation: segmentation, paging, or both
- **Segmentation**
  - logical address space is divided into variable-size blocks (segments)
  - each segment is contiguous in PM
  - segment table (ST) keeps track of starting addresses of segments
  - VA = (s, w), where s is segment number and w is offset within segment
- **Paging**
  - logical address space is divided into fixed-size blocks (pages)
  - PM is divided into fixed-size blocks (page frames)
  - page table PT keeps track of which page is in which frame
  - VA = (p, w), where p is the page number and w is offset within page

Principles of VM

- Advantage of **segmentation**:
  - each segment corresponds to **logical** component (code, data, stack, etc.)
  - easier for sharing and linking
- Advantage of **paging**:
  - frame size = page size, thus any page may be mapped into any available frame
  - no need to search for and maintain **variable-size** memory partitions
- To gain advantages of both: **combine segmentation with paging**

Segmentation with Paging

- each segment is divided into fixed-size pages
- each ST entry points to PT corresponding to one segment
- each PT entry points to one page



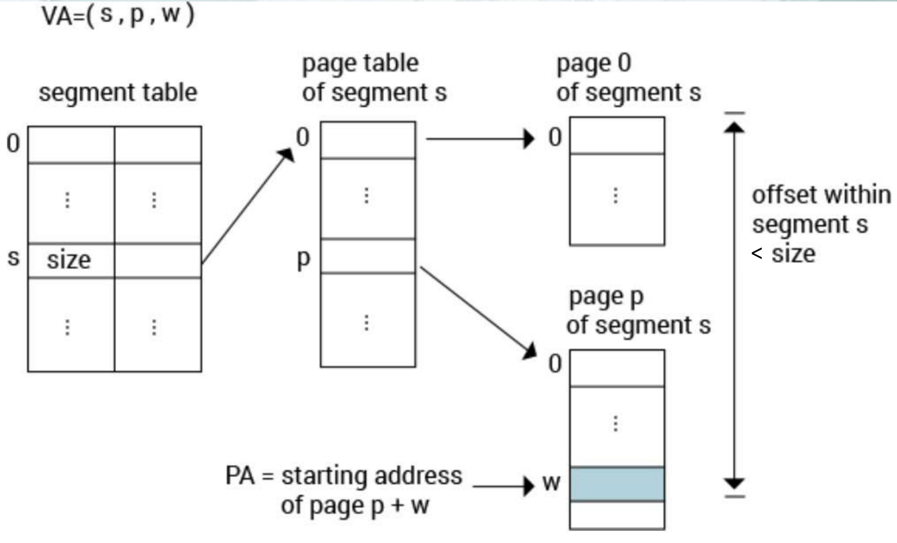
Address Translation

- VA is a nonnegative integer
- Number of bits to represent VA determines size of the VA space
  - Ex: with VA of 32 bits, 2^{32} addresses can be created
- VA = (s, p, w), s is segment number, p is page number, w is offset within page
- Number of bits used to represent s, p, w determine sizes of ST, PT, and page

- PA is also a nonnegative integer
  - Number of bits determines size of PM

- VM manager translates VAs into PAs
  - first, break VA into 3 integer components, s, p, and w
  - use s, p, w as indices into ST, PT, page

Address Translation



- segments have different sizes
- VA must not exceed size

Demand Paging

- If size of VM exceeds size of PM, then
  - not all pages can be present in PM
  - must be loaded from a disk as needed: **demand paging**
- Demand paging applies also to PTs: PT may not be resident
- **Present bit**
  - associated with each entry of the PT and ST
  - if true, then entry contains **frame** number of the page or PT
  - if false, then entry contains location of page or PT on **disk**
- **Page fault***:* reference to nonresident page or PT
  - locate missing page/PT on disk, allocate page frame, load page/PT

Two versions of project

1. Without demand paging (max score is 60%)
2. With demand paging (max score is 100%)

Notes:

- You can submit both versions, and we will use the version that provides the most credit
- This is recommended if you are not confident that your implementation of the demand paging project is reliable.

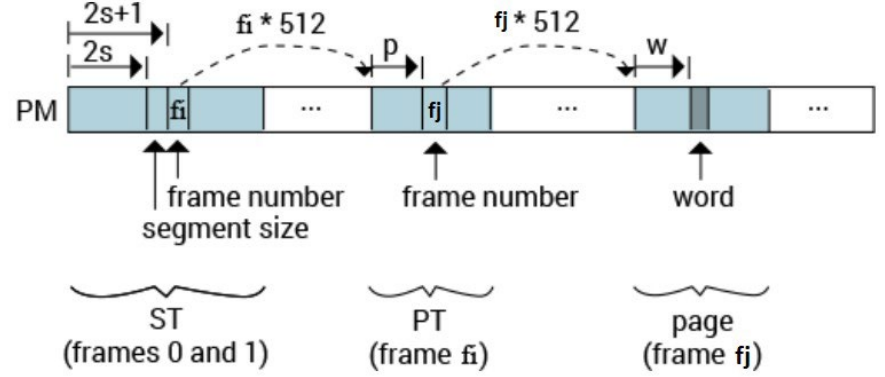
Basic Version – Without Demand Paging

VM Specs w/o Demand Paging

- Memory is word-addressable, where each word is an integer
- There is only a **single** process and hence only a single ST
- Each ST entry consists of 2 integers:
  - **size** of the segment s (number of words)
  - frame number holding PT of segment s
- If segment does not exist, then both fields in the entry are 0
- Each PT entry contains frame number of page
- VA: 32-bit integer, divided into s, p, w
- Each component occupies 9 bits:
  - PT size = page size = 512 words
  - ST size = 1024 words (each entry occupies 2 integers)

Representation of PM

- PM: array of 524,288 integers, divided into 1024 frames of 512 words each

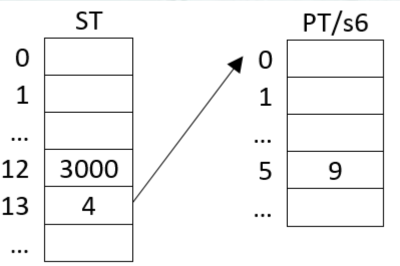


- PM[2s] refers to size of s, PM[2s+1] refers to frame number of PT
- Given s, p, w: PA = PM[PM[2s+1]*512+p]*512+w

Initialization of the PM

- Pre-initialization: PM[i] = 0 for all i
- PM is then initialized from a file, which specifies:
- Line 1: $s_1\ z_1\ f_1\ s_2\ z_2\ f_2 \dots s_n\ z_n\ f_n$ ← defines ST
- $s_i\ z_i\ f_i$ means: PT of segment s_i resides in frame f_i, length of segment s_i is z_i
  - Ex: 6 3000 4: PT of segment 6 resides in frame 4, size of segment 6 is 3000
    Initialize: PM[2*6] = PM[12] = 3000 and PM[2*6+1] = PM[13] = 4
- Line 2: $s_1\ p_1\ f_1\ s_2\ p_2\ f_2 \dots s_m\ p_m\ f_m$ ← defines PTs
- $s_j\ p_j\ f_j$ means: page p_j of segment s_j resides in frame f_j
  - Ex: 6 5 9: page 5 of segment 6 resides in frame 9
    Initialize:
    PM[PM[13]*512+5] = PM[4*512+5] = PM[2053] = 9

Initialization of the PM



- Line 1: 6 3000 4 (VAs 0-2999 are valid)
- Line 2: 6 5 9 … (assume pages 0-5 are valid)

0	1	…	12	13	…	2048	…	2053	…	4608	…
			3000	4				9			

- PM[2*6] = PM[12] = 3000   PM[2*6+1] = PM[13] = 4
- PM[4*512+5] = PM[2053] = 9

Executing VA Translations

- After PM initialization, system processes an input file (separate from the initialization file).
  - The input file consists of a sequence of commands. We will discuss the format of the input file and the commands later.
  - Some of the commands require translating a VA to a PA and writing the result to the output file
  - The result of each translation is either a PA or -1 (error)
- **Translation**
  - break VA into s, p, w, pw
  - if pw ≥ PM[2s], then report error; VA is outside of the segment boundary
  - else PA = PM[PM[2s + 1]*512 + p]*512 + w

Deriving s, p, w, and pw from VA

- s: right-shift VA by 18 bits
    - discards p and w
- w: AND bitwise VA with 9-bit binary constant “1 1111 1111” (1FF)
    - removes all bits other than last 9 — value of w
- p: first right-shift VA by 9 bits to discard w
  then AND result with binary constant “1 1111 1111”
    - removes all bits other than the last 9 — value of p
- pw: AND VA with the 18-bit binary constant “11 1111 1111 1111 1111” (3FFFF)
    - removes the leading s component
- Ex: VA = 789002 = 000000011 000000101 000001010, s = 3, p = 5, w = 10

Executing VA Translations - Example

0	1	…	12	13	…	2048	…	2053	…	4608	…
			3000	4				9			

if pw ≥ PM[2s], then error

else PA = PM[PM[2s + 1]*512 + p]*512 + w

VA = 1575424 = (000000110,000000101,000000000)

- s = 6, p = 5, w = 0, pw = 2560, pw < 3000
- PA = PM[PM[2*6+1]*512+5]*512+0 = PM[2048+5]*512+0 = 9*512+0 = 4608

VA = 1575863, s = 6, p = 5, w = 439, pw = 2999, pw < 3000

- PA = … = PM[2048+5]*512+439 = 9*512+439 = 5047

VA = 1575864, s = 6, p = 5, w = 440, pw = 3000

- pw ≥ 3000: error, PA would be 5048 but this is outside of segment boundary

Extended Version – With Demand Paging

VM with Demand Paging

- Not all pages or PTs are resident in PM
  - must be copied from a paging disk when accessed
  - to avoid page replacement, assume that a free frame is always available
- **Paging Disk**
  - emulated as a two-dimensional integer array, D[B][512]
  - B: number of blocks (e.g., 1024)
  - 512: block size (= page size)
- Disk may only be accessed one block at a time:
  - *read_block(b, m)* copies block D[b] into PM frame starting at location PM[m]

Extensions

Contents of ST and PT

- If PT is not resident, corresponding ST entry contains a negative number −b
  - the absolute value |−b| = b is the block number on disk that contains the PT
- If a page is not resident, corresponding PT entry contains a negative number −b
  - b is the block number on disk that contains the page
- The sign bit is used as the present bit (negative = not resident)

Extensions

Free Frames

- Blocks are moved to PM from disk at page fault
- Memory manager must keep track of which frames are free
  - Use a linked list (or mark frames as free)
- Pre-initialization: all frames except frames 0 and 1 are designated as free
  - Frames 0 and 1 are dedicated to segment table, hence not free
- During initialization, frames that are used are no longer free:
  - Frames used to store page tables
  - Frames used to store pages within a segment

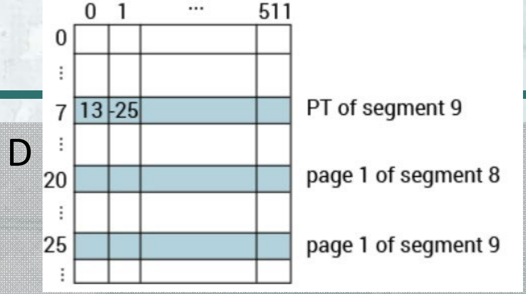
VA Translation

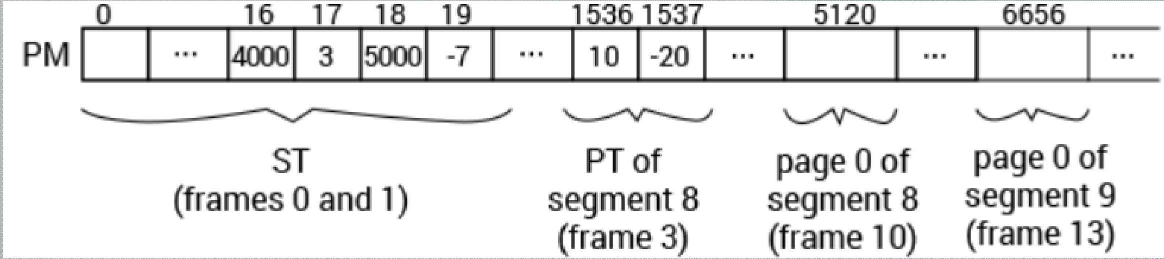
- If pw ≥ PM[2s], report error; VA is outside of the segment boundary
- If PM[2s + 1] < 0, then /* page fault: PT is not resident */
  - Allocate free frame f1 using list of free frames
  - Update list of free frames
  - Read disk block b = |PM[2s + 1]| into PM staring at location f1*512
  - PM[2s + 1] = f1 /* update ST entry */
- If PM[PM[2s + 1]*512 + p] < 0 /* page fault: page is not resident */
  - Allocate free frame f2 using list of free frames
  - Update list of free frames
  - Read disk block b = |PM[PM[2s + 1]*512 + p]| into PM staring at f2*512
  - PM[PM[2s + 1]*512 + p] = f2 /* update PT entry */
- Return PA = PM[PM[2s + 1]*512 + p]*512 + w

Initialization of PM

Example:

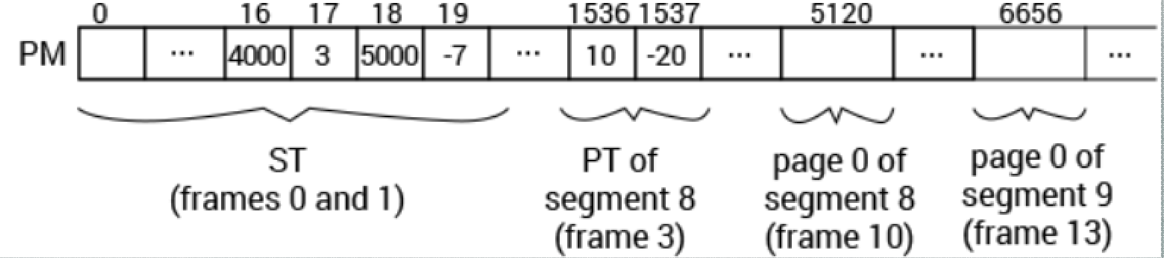
- Line 1: 8 4000 3 9 5000 -7 …
- Line 2: 8 0 10 8 1 -20 9 0 13 9 1 -25 …





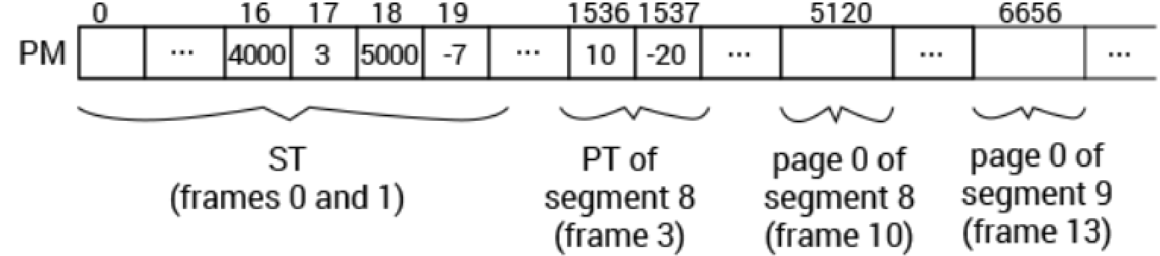
VA Translation: PT and page resident

- VA = 2097162 = 1000 000000000 000001010 = (8, 0, 10)
- PM[2*8+1] = 3 (> 0): PT is resident at 3*512 = 1536
- PM[1536+0] = 10 (> 0): pg is resident at 10*512 = 5120
- PA = 10*512+10 = 5120+10 = 5130



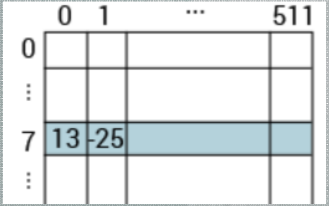
VA Translation: PT resident, pg not resident

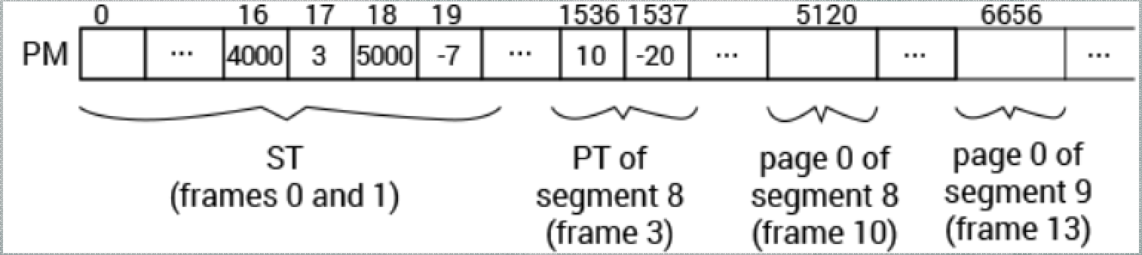
- VA = 2097674 = 1000 000000001 000001010 = (8, 1, 10)
- PM[2*8+1] = 3 (> 0): PT is resident at 3*512 = 1536
- PM[1536+1] = -20 (< 0): pg is in disk block 20
- assume frame 4 is allocated: replace -20 with 4, page 1 starts at 4*512=2048
- PA = 2048+10 = 2058



VA Translation: PT not resident, pg resident

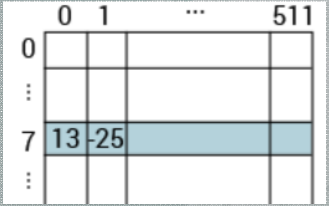
- VA = 2359306 = 1001 000000000 000001010 = (9, 0, 10)
- PM[2*9+1] = -7   (< 0): PT is in disk block 7
- assume frame 5 is allocated: replace -7 with 5
- copy PT from block 7 to PM[5*512] = 2560
- PM[2560+0] = 13   (> 0): pg is resident at 13*512=6656
- PA = 6656+10 = 6666

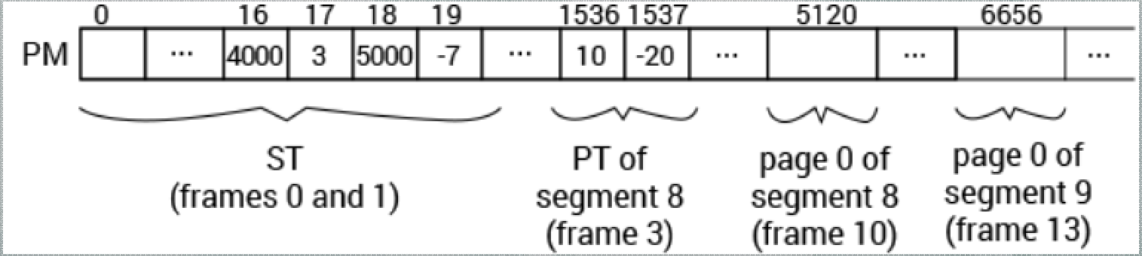




VA Translation: PT and pg not resident

- VA = 2359818 = 1001 000000001 000001010 = (9, 1, 10)
- PM[2*9+1] = -7 (< 0): PT is in disk block 7
- copy PT from block 7 to PM[5*512] = 2560 as before
- PM[2560+1] = -25 (< 0): pg is in block 25
- assume frame 14 is allocated, replace -25 by 14
- PA = 14*512+10 = 7168+10 = 7178





Input file commands

There are three commands that need to be implemented: TA, RP, NL

1. TA (Translate Address):
   - TA <va>
     - <va> is an integer, representing a virtual address
     - Translate <va> to a PA and write the result (as a decimal integer) to the output file
2. RP (Read Physical address):
   - RP <pa>
     - <pa> is an integer, representing a physical address
     - Read the contents of the physical address <pa> and write the result (as a decimal integer) to the output file

(Continued on next slide)

Input file commands (continued)

3. NL (New Line):
   - NL
     - Start a new line in the output file.

Output file format:

- A sequence of integers (one for each TA or RP command).
- The integers are to be written one after another on the same line separated by a space,
- A NL command causes a new line to be started.

Example of init, input, output file

init

```
6 3000 4
6 5 9
```

input

```
TA 1575424
TA 1575863
TA 1575864
NL
RP 12
RP 13
RP 2053
RP 1024
```

output

```
4608 5047 -1
3000 4 9 0
```

Summary of Specific Tasks

- Design and implement a VM manager using segmentation and paging
  - Without demand paging (reduced credit), or
  - With demand paging (full credit)
- Skip the TLB
- Design and implement a driver program that
  - Initializes the system from a given initialization file
  - Reads the input file and executes the commands
  - The results are written into an output file