Recall: Process Address Space

Text: program instructions
- Execute-only, fixed size

Data: variables (static, heap)
- Read/write, variable size
- Dynamic allocation by request

Stack: activation records (local/auto)
- Read/write, variable size
- Automatic growth/shrinkage

Fitting Process Into Physical Memory

Need to find a large enough hole
May not succeed (even if enough fragment space)
Even if successful, still inefficient
- Space must be allocated for potential growth areas
Solution: break process into pieces
- Distribute into available holes
Fitting Process Into Physical Memory

Need to find a large enough hole

May not succeed (even if enough fragment space)

Even if successful, still inefficient
  • Space must be allocated for potential growth areas

Solution: break process into pieces
  • Distribute into available holes

Two Approaches

Segmentation
  • Break process up into logical segments
  • Segments are generally of different sizes

Paging
  • Break process up into physical pages
  • Pages are the same size

Segmented Address Space

Address space is a set of segments

Segment: a linear address memory
  • Typically contains logically-related information
  • Examples: program code, data, stack

Each segment has an identifier $s$, and a size $N_s$
  • $s$ between 0 and $S-1$, $S = \text{max number of segments}$

Logical addresses are of the form $(s, i)$
  • Offset $i$ within segment $s$, $i$ must be less than $N_s$

Example: Segmented Address Space
Segment-based Address Translation

- logical address (s, i) → Address translation → physical address

Problem: how to translate
- a segment-based logical address (s, i)
- into physical address x?

Solution: use a segment (translation) table
- Translate (s, i) into physical address x = t(s, i)

Segment Table

- One per process (typically)

   Table entry elements
   - V: valid bit
   - Base: segment location
   - Bound: segment size
   - Perm: permissions

   Located in memory
   - Segment table base register
   - Segment table size register

Address Translation

- Logical Address

<table>
<thead>
<tr>
<th>Segment s</th>
<th>Offset i</th>
</tr>
</thead>
</table>

  Physical address = segment base of s + offset i

  BUT, first check validity

  Physical Address

Check if Segment Number s is Valid

- Logical Address

  | Segment s | Offset i |

  Physical Address

  STBR

  STSR

  s < STSR

  Physical Address
Check if Segment Entry $s$ is Valid

Check if Offset $i$ is Within Bounds

Check if Operation is Permitted

Translate Address
Sizing the Segment Table

Number of bits \( n \) specifies max size of table, where number of entries = \( 2^n \)

Number of bits \( n \) specifies max size of segment

Number of bits needed to specify max segment size

Number of bits needed to address physical memory

Example of Sizing the Segment Table

Given 32 bit logical address, 1 GB physical memory (max)

- 5 bit segment number, 27 bit offset

Segment \( s \): 5 bits

Offset \( i \): 27 bits

5 bits to address \( 2^5 = 32 \) entries

30 bits needed to address 1 GB

27 bits needed to size up to 128 MB

8 bytes needed to contain \( 61 \) (1+30+27+3+) bits

Table size = \( 32 \times 8 = 256 \) bytes

Pros and Cons of Segmentation

Pro: Each segment can be
- Located independently
- Separately protected
- Grown/shrunk independently

Pro: Segments can be shared between processes

Con: Variable-size allocation
- Difficult to find holes in physical memory
- External fragmentation

Paged Address Space

Logical (process) memory
- Linear sequence of pages

Physical memory
- Linear sequence of frames

Pages and frames
- Frame: a physical unit of information
- A page fits exactly into a frame
- Fixed size, all pages/frames same size
Page-based Logical Addressing

Logical addresses are of form \((p, i)\)
- \(p\) is page number, 0 to \(N_L-1\)
- \(i\) is offset within page
- Note: \(i\) is less than page size

Size of logical address space
- \(N_L = \) max number of pages
- \(N_L \times \) page size = size of logical address space

Frame-based Physical Addressing

Physical addresses are of form \((f, i)\)
- \(f\) is frame number, 0 to \(N_P-1\)
- \(i\) is offset within frame
- Note: \(i\) is less than frame size

Size of physical address space
- \(N_P = \) max number of frames
- \(N_P \times \) frame size = size of physical address space

Page-based Address Translation

Problem: how to translate an address-based logical address \((p, i)\) into a physical address \((f, i)\)?
Solution: use a page translation table
- \(\text{Translate page } \ p \text{ into frame } f = t(p)\)
**Page Table**

One per process (typically)

Table entry elements

- V: valid bit
- R: reference bit
- M: modified bit
- Frame: page location

Located in memory

- Page table base register
- Page table size register

**Address Translation**

Physical address = frame of p + offset i

BUT, first check validity

**Check if Page Number p is Valid**

Logical Address

Page p Offset i

Physical address

Check if Page Table Entry p is Valid

Logical Address

Page p Offset i

V == 1
Check if Operation is Permitted

Translate Address

Physical Address by Concatenation

Sizing the Page Table
Example of Sizing the Page Table

Given 32 bit logical address, 1 GB physical memory (max)
• 20 bit page number, 12 bit offset

<table>
<thead>
<tr>
<th>Page p: 20 bits</th>
<th>Offset i: 12 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>20 bits to address (2^{20} = 1 \text{ M entries})</td>
<td>Page size = frame size = (2^{12} = 4096 \text{ bytes})</td>
</tr>
<tr>
<td>18 bits to address (2^{18}/2^{12} \text{ frames})</td>
<td>Points to (base) frame containing 2nd-level page table</td>
</tr>
<tr>
<td>4 bytes needed to contain (24 (1+1+1+18+3+...)) bits</td>
<td>Table size = (1 \text{ M} \times 4 = 4 \text{ MB})</td>
</tr>
</tbody>
</table>

Multi-Level Page Tables

Logical Address

1st-level Page p 2nd-level Page p Offset i

Points to (base) frame containing 2nd-level page table

Physical Address

Segmentation vs. Paging

Segment is good “logical” unit of information
• Can be sized to fit any contents
• Makes sense to share (e.g., code, data)
• Can be protected according to contents

Page is good “physical” unit of information
• Simple memory management

Why not have best of both
• Segmentation on top of paging

Combining Segmentation and Paging

Logical memory composed of segments
Each segment composed of a set of pages
Segment table: maps each segment s to page table
Page tables (like before)
### Address Translation

Logical address: \([\text{segment } s, \text{ page } p, \text{ offset } i]\)

Do various checks
- \(s < \text{STSR}, \ V == 1, \ p < \text{bound}, \ \text{perm (op)}\)
- May get a segmentation violation

Use segment \(s\) to index segment table to get page table
Use page \(p\) to index page table to get frame \(f\)
Concatenate frame and offset to get physical address

### Segment/Page Address Translation

![Segment/Page Address Translation Diagram](image)

### Cost of Translation

Each page table lookup costs a memory reference
- For each reference, additional references required
- Slows machine down by factor of 2 or more

Take advantage of locality of reference
- Most references are to a small number of pages
- Keep translations of these in high-speed memory

Problem: we don’t know which pages until referenced

### Translation Lookaside Buffer (TLB)

![Translation Lookaside Buffer (TLB) Diagram](image)

Fast associative memory keeps most recent translations
If key matches, get frame number for physical address, else wait for normal memory translation (in parallel)
Translation Cost with TLB

Cost is determined by

- Speed of memory: ~ 50 nsec
- Speed of TLB: ~ 10 nsec
- Hit ratio: fraction of refs satisfied by TLB, ~95%

Speed with no address translation: 50 nsec

Speed with address translation

- TLB miss: 100 nsec (100% slowdown)
- TLB hit: 60 nsec (20% slowdown)
- Average: \(60 \times .95 + 100 \times .05 = 62\) nsec

TLB Design Issues

The larger the TLB

- the higher the hit rate
- the slower the response
- the greater the expense

TLB has a major effect on performance!

- Must be flushed on context switches
- Alternative: tagging entries with PIDs

MIPS: has only a TLB, no page tables!

- Devote more chip space to TLB