Segmentation
Segmentation

- Memory-management scheme that supports user view of memory.
- A program is a collection of segments. A segment is a logical unit such as:
  - main program
  - procedure
  - function
  - method
  - object
  - local variables, global variables
  - common block
  - stack
  - symbol table, arrays
User’s View of a Program
Logical View of Segmentation

- User Space
  - 1
  - 2
  - 3
  - 4

- Physical Memory Space
  - 1
  - 4
  - 2
  - 3
Segmentation

- A straightforward solution to our compiler problem is to give each area an independent address space, called a *segment*.
- Each segment consists of linear addresses from zero up to a maximum.
- Different segments can have different lengths.
- A segment’s length may even change dynamically during execution (e.g., a segment for a stack).
- Each virtual address has two parts:
  
<table>
<thead>
<tr>
<th>segment-number</th>
<th>offset</th>
</tr>
</thead>
</table>

Segmentation = Paging with variable size pages
Use of Segments

Suppose each procedure is a separate segment.

1. If procedure in segment \( n \) is recompiled, no other procedures need be changed.

In contrast, a 1-dimensional address space requires a linker to layout all procedures compactly, thus affecting many procedure’s entry point addresses.

2. A shared library can put each sharable unit in a different segment.

A paged system essentially simulates segmentation (by putting library elements on page boundaries) to permit shared libraries.
Segmentation Memory Management

- Segments are brought in on demand

- Uses variable size partitioning:
  - Contiguous allocation of each segment
  - Each segment must occupy contiguous locations, but segments may be scattered throughout memory.
  - Use first fit, etc.
Segmentation: Implementation

- Divide each process into logical segments (procedures, arrays, etc.)
- Logical breakdown gives an intuitive structure to main memory
- Managing segments:

Segment Map Table (SMT)

<table>
<thead>
<tr>
<th>Length</th>
<th>Address</th>
</tr>
</thead>
</table>

1 SMT/Job
1 entry/segment
Each offset is checked against the segment length to ensure legal access.
Sharing segments

- Because segmentation is based on a logical ordering instead of a physical one, *sharing and protection are easier to implement*
- Share SQRT function, instead of sharing particular pages
Segmentation: Pros and Cons

**Advantages:**

1. Multiple VA spaces
2. Whole program need not be simultaneously in memory
3. Sharing and protection are easier than any memory management method discussed so far
Segmentation: Pros and Cons

**Disadvantages:**

1. VA to PA translation requires table lookup, increasing memory cycle time
2. Entire segment must be in memory simultaneously -- a problem with big segments!
3. SMT needs one more field than PMT: for segment length
4. Variable size storage allocation requires compaction
5. External fragmentation
Segmentation with Demand Paging

- To handle the problem of large segments, divide each logical segment into fixed size pages

*Virtual Address*

<table>
<thead>
<tr>
<th>segment</th>
<th>page</th>
<th>offset</th>
</tr>
</thead>
</table>

*Example*

| 1 | 2 | 100 |

offset 100 from page 2 of segment 1
Managing Segments and Pages

1. SMTAR (analogous to PMTAR)
   - Length (# of segments)
   - Address of SMT

2. SMT
   - Length (# of pages) for each segment
   - Address of PMT for each segment
   - Residence bit
   - 1 SMT / job
   - 1 entry / segment

3. PMT
   - Status (S, M, IT) of each page
   - Block # for each page
   - 1 PMT / segment
   - 1 entry / page
Pros/Cons of Segmentation with paging

Advantages:

1. Can page unused PMTs
2. Multiple virtual address spaces simplify programming
3. Easy memory allocation
4. Easy sharing and protection
5. Only part of segment need be in memory at once
Pros/Cons of Segmentation with paging

Disadvantages:

1. Even more overhead than segmentation for virtual to physical address translation
2. Internal fragmentation
3. Storage space for both SMTs and PMTs
- Descriptor segment points to page tables
- Segment descriptor – numbers are field lengths
Segmentation with Paging: MULTICS (2)

A 34-bit MULTICS virtual address
Segmentation with Paging: MULTICS (3)

MULTICS virtual address

Segment number

Page number

Offset

Conversion of a 2-part MULTICS address into a main memory address

A 34-bit MULTICS virtual address
### Segmentation with Paging: MULTICS (4)

<table>
<thead>
<tr>
<th>Segment number</th>
<th>Virtual page</th>
<th>Page frame</th>
<th>Protection</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1</td>
<td>7</td>
<td>Read/write</td>
<td>13</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>2</td>
<td>Read only</td>
<td>10</td>
</tr>
<tr>
<td>12</td>
<td>3</td>
<td>1</td>
<td>Read/write</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Execute only</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>Execute only</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>12</td>
<td>Execute only</td>
<td>9</td>
</tr>
</tbody>
</table>

- Simplified version of the MULTICS TLB
- Existence of 2 page sizes makes actual TLB more complicated
Segmentation with Paging: Pentium (1)

A Pentium selector

0 = GDT/1 = LDT
Privilege level (0-3)
Segmentation with Paging: Pentium (2)

- Pentium code segment descriptor
- Data segments differ slightly
Segmentation with Paging: Pentium (3)

Conversion of a (selector, offset) pair to a linear address
Segmentation with Paging: Pentium (4)

Mapping of a linear address onto a physical address
Segmentation with Paging: Pentium (5)

Protection on the Pentium