Segmentation & Shared Memory

Steve Goddard
goddard@cse.unl.edu

http://www.cse.unl.edu/~goddard/Courses/CSCE451

Memory Management

- Name spaces

  » Schemes to date have considered only a single name space per process
    (Why? Why not have multiple spaces?)

Program P's

<table>
<thead>
<tr>
<th>Program</th>
<th>Data</th>
<th>Stack</th>
<th>Heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2^n - 1

### Multiple Name Spaces

**Example — Protection/Fault isolation & sharing**

- **Run-Time Stack**
- **Program Data**
- **Program Text**
- **Heap**
- **User Code**
- **Libraries**


### Supporting Multiple Name Spaces

#### Segmentation

- **Segment** — a memory "object"
  - A virtual address space
- **A process now addresses objects** — a pair \((s, \text{addr})\)
  - \(s\) — segment number
  - \(\text{addr}\) — a virtual address within an object offset
- **A virtual address is now a triple** \((s, p, o)\)

Segment + Address register scheme:

```
0
\begin{array}{c}
| s | \_ | \_ |
\end{array}
```

Single address scheme:

```
0
\begin{array}{c}
| s | p | o |
\end{array}
```
Segmentation
Virtual address translation

- One additional level of indirection — The *segment table*

```
CPU
```

```
STBR
```

```
Segment Table
```

```
Page Table
```

```
Memory
```

```
Process P's Segment Table
```

```
Segment S's Page Table
```

---

Shared Memory
Sharing of procedures and data

- Why share?

```
Levels of sharing
``` - source code
- object code
- executable binaries

---
Shared Memory
Requirements for sharing

Sharing Code
- Programs cannot modify their own code
  » "Pure procedures"
- Serially sharable code
  » Programs that are self-initializing
  » Programs that contain their own global data
- Concurrently sharable code
  » Programs must be \textit{reentrant}
  » Run-time stack cannot be shared

Sharing Data
- Data cannot (typically) contain pointers

Sharing in Non-Segmented/Paged Systems
Trading-off protection for sharing

- Space can no longer be allocated contiguously
  » Base + Limit register schemes not applicable
- Loaders must determine if copies of shared objects already exist in memory
  » And if so where is it?
- Similar problems with garbage collection
Sharing in Paged Systems

Code sharing

- Shared code must reside in the same place in all process’s VAS

```
shared proc()
begin
    x := 6
    :     :
end while
end proc
```

Thus shared pages must have the same page number in all process’s VAS.
Sharing in Paged Systems

Data sharing

- “Raw data” — pages can appear anywhere in a process’s VAS

- Shared data containing pointers must also reside in the same place in all process’s VAS
Sharing in Paged Systems

Data sharing

- If the data contains pointers treat pages the same as shared code

```
shared var
foo : char
bar : ptr to char
```

A's Virtual Address Space

Shared Data Object

Physical Memory

B's Virtual Address Space

Shared Data Object

```
bar: (p:2, o:1)
```

Sharing in Segmented Systems

The simplest form of sharing

- Sharing portion of a process’s “segment space”
Sharing in Segmented Systems

Sharing segments

- Unit of sharing a segment rather than a set of pages
  - Processes need only agree on 1 number rather than a sequence of numbers
  - If segments are paged then the page tables are automatically shared

### Diagram

- A's Segment Table
  - Code Seg
  - Heap Seg

- B's Segment Table
  - Code Seg
  - Heap Seg

- Shared Segment Page Table

Dynamic Linking & Sharing

A key technology for realizing shared memory

- Concept
  - Dynamically bind to shared modules at run-time

- Advantages
  - On demand linking ensures that only those modules that are actually used will be loaded
  - Possibly allows modules to be replaced or upgraded at run-time

- Disadvantages
  - Possible performance penalty at run-time for invoking new modules
  - Overhead of indirect/interpretive access
  - Complexity of the overall scheme