Linking and Sharing of Code and Data in Memory

- Shared memory systems
  - Real
  - Virtual
- Distributed systems

Linking and Sharing

- Single-Copy Sharing
- Static Linking and Sharing
- Dynamic Linking and Sharing
- Principles of DSM
- Implementations of DSM
**Single-copy sharing**

- **why share**
  - processes need to access common data
  - better utilization of memory (code, system tables, databases)
- **requirement for sharing**
  - how to express what is shared
    - a priori agreement (e.g., system components)
    - language construct (e.g., shmget/shmat)
  - shared code must be reentrant (read-only)
  - stack, heap must be replicated per process

**Static linking/sharing**

- linking resolves external references
- sharing links to the same module
- static linking/sharing:
  - resolve references before execution starts
Sharing w/o segmentation/paging

- with one or no RR:
  - all memory of a process is contiguous
  - share user programs:
    * possible by partial overlap only
    * too restrictive and difficult; generally not used
  - share system components:
    * agree on a starting position
    * linker resolves references to that location

Figure 9-1
Sharing w/o segmentation/paging

- with multiple RRs
  - CBR’s can point to same copy of code
  - SBR’s can point to private copies of stack, etc.

Figure 9-2
Sharing in paging systems

- PT entries of different processes point to the same page frame
- data pages: unrestricted
- code pages: must have the same page#'s in all PTs to allow self-references

Figure 9-3

- Note: BR can’t be used -- paged system has no concept of function boundaries
sharing in paged systems

Fig 9-3: sharing in paged systems
Sharing of code

- Code pages contain references to other pages (p, w)

- Page numbers are absolute and this requires that code pages are mapped into the same virtual page in all process address spaces sharing the code area!

- Consequences:
  - implement shared code as self-contained unit
  - make use offsets and base registers only
  - delay linking until runtime – DLL
  - mechanism: transfer vector and stubs

Fig 9-4: dynamic linking
Sharing in segmented systems

- ST entries of different processes point to the same segment in PM
- data pages: unrestricted
- code pages:
  - assign same segment#'s in all STs, or
  - use BR:
    * function call loads cbr
    * self-references have the form \text{w(CBR)}
    * other references have the form (s,w)

Dynamic linking/sharing

- basic principles:
  - self-references resolved using CBR
  - external references are indirect via a private linkage section
  - external reference (S,W) is resolved to (s,w) at runtime when first used (S,W are symbolic names)
  - (s,w) is entered in linkage section of process
  - code is unchanged
  - subsequent references use (s,w) without involving OS
Dynamic linking/sharing

- compiler setup prior to execution
  Figure 9-4(a)
- memory after execution of load instruction
  Figure 9-4(b)

Fig 9-5: dynamic linking - sharing
(a) before external references is executed
Fig 9-5: dynamic linking - sharing 
(b) after external references is executed

Fig 9-6: unstructured DSM
Fig 9-7: Structured DSM

Local address space

P1
MM1

P2
MM2

Pn
MMn

shared

shared

shared

DSM
Example of Strict Consistency

\[
\text{initial: } x = 0;
\]
\[
\text{real time: } t_1 \quad t_2 \quad t_3 \quad t_4
\]
\[
p_1: \{ \quad x=1; \quad a_1=x; \quad x=2; \quad b_1=x; \} 
\]
\[
p_2: \{ \quad a_2=x; \quad b_2=x; \}
\]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Page Location</th>
<th>Page Status</th>
<th>Actions Taken Before Local Read/Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>local</td>
<td>read-only</td>
<td>x</td>
</tr>
<tr>
<td>Write</td>
<td>local</td>
<td>read-only</td>
<td>invalidate remote copies; upgrade local copy to writable</td>
</tr>
<tr>
<td>Read</td>
<td>remote</td>
<td>read-only</td>
<td>make local read-only copy</td>
</tr>
<tr>
<td>Write</td>
<td>remote</td>
<td>read-only</td>
<td>invalidate remote copies; make local writable copy</td>
</tr>
<tr>
<td>Read</td>
<td>local</td>
<td>writable</td>
<td></td>
</tr>
<tr>
<td>Write</td>
<td>local</td>
<td>writable</td>
<td></td>
</tr>
<tr>
<td>Read</td>
<td>remote</td>
<td>writable</td>
<td>downgrade page to read-only; make local read-only copy</td>
</tr>
<tr>
<td>Write</td>
<td>remote</td>
<td>writable</td>
<td>transfer remote writable copy to local memory;</td>
</tr>
</tbody>
</table>

Protocol for handling DSM pages
### Processor actions and System actions

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1 reads A</td>
<td>Operation is done locally</td>
</tr>
<tr>
<td>P1 writes A</td>
<td>Operation is done locally (page is writable)</td>
</tr>
<tr>
<td>P1 writes B</td>
<td>Invalidate copy in MM2; upgrade copy in MM1 to writable</td>
</tr>
<tr>
<td>P2 reads A</td>
<td>Downgrade page in MM1 to read only; make copy in MM2</td>
</tr>
<tr>
<td>P2 writes A</td>
<td>Transfer page from MM1 to MM2</td>
</tr>
</tbody>
</table>

---

### Fig 9-10: example of page operations

- **DSM**
  - Page A writable
  - Page B read-only

- **MM1**
  - Page A writable
  - Page B read-only
  - Page B read-only

- **MM2**
  - Page B read-only
  - Page B read-only
  - Page B read-only
Fig 9-11: example of possible instruction interleaving

Initial: \( x = 0 \)

(a) \( p1: \{ \quad x=1; \quad a1=x; \quad x=2; \quad b1=x; \} \)
\( p2: \{ a2=x; \quad b2=x; \} \)

(b) \( p1: \{ \quad x=1; \quad a1=x; \quad x=2; \quad b1=x; \} \)
\( p2: \{ a2=x; \quad b2=x; \} \)

(c) \( p1: \{ x=1; \quad a1=x; \quad x=2; \quad b1=x; \} \)
\( p2: \{ a2=x; \quad b2 = x; \} \)

Fig 9-12: weak memory consistency

\[ x := 1; \]
\[ a1 := x; \]
\[ S; \]
\[ b1 := x; \]

\[ x := 1; \]
\[ a2 := x; \]
\[ S; \]
\[ b2 := x; \]
Fig 9-13: release memory consistency

lock;
\( x := 1; \)
unlock;

cause propagation
delayed until \( p1 \) unlocks

Fig 9-14: entry memory consistency

lock;
\( x := 1; \)
unlock;

delayed until \( p1 \) unlocks