Memory Management

Memory Manager

- Requirements
  - Minimize executable memory access time
  - Maximize executable memory size
  - Executable memory must be cost-effective
- Today’s memory manager:
  - Allocates primary memory to processes
  - Maps process address space to primary memory
  - Minimizes access time using cost-effective memory configuration
  - May use static or dynamic techniques

Storage Hierarchies

The Basic Memory Hierarchy

Exploiting the Hierarchy

- Upward moves are (usually) copy operations
  - Require allocation in upper memory
  - Image exists in both higher & lower memories
- Updates are first applied to upper memory
- Downward move is (usually) destructive
  - Destroy image in upper memory
  - Update image in lower memory
- Place frequently-used info high, infrequently-used info low in the hierarchy
- Reconfigure as process changes phases
Address Space vs Primary Memory

Process Address Space

Hardware Primary Memory

Mapped to object other than memory

Creating an Executable Program

- Compile time: Translate elements
- Link time: Combine elements
- Load time:
  - Allocate primary memory
  - Adjust addresses in address space
  - Copy address space from secondary to primary memory

A Sample Code Segment

```c
... static int gVar;
... int proc_a(int arg){
    ...
    gVar = 7;
    put_record(gVar);
    ...
}
```

The Relocatable Object module

<table>
<thead>
<tr>
<th>Code Segment</th>
<th>Address</th>
<th>Generated Code</th>
<th>Data Segment</th>
<th>Address</th>
<th>Generated Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>0000</td>
<td>entry proc_a</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>0220</td>
<td>load =7, R1</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>0224</td>
<td>store R1, 0136</td>
<td>...</td>
<td>0236</td>
<td>[Space for gVar variable]</td>
</tr>
<tr>
<td>...</td>
<td>0226</td>
<td>call proc_a</td>
<td>...</td>
<td>0228</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>0600</td>
<td>External reference table</td>
<td>0604</td>
<td>'put_record'</td>
<td>0608</td>
</tr>
<tr>
<td>...</td>
<td>0540</td>
<td>External definition table</td>
<td>0548</td>
<td>'proc_a'</td>
<td>0550</td>
</tr>
<tr>
<td>...</td>
<td>0008</td>
<td>(symbol table)</td>
<td>...</td>
<td>0599</td>
<td>(last location in the code segment)</td>
</tr>
</tbody>
</table>

The Absolute Program

<table>
<thead>
<tr>
<th>Code Segment</th>
<th>Address</th>
<th>Generated Code</th>
<th>Data Segment</th>
<th>Address</th>
<th>Generated Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>0000</td>
<td>(Other modules)</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>1020</td>
<td>entry proc_a</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>1220</td>
<td>load =7, R1</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>1224</td>
<td>store R1, 0136</td>
<td>...</td>
<td>1226</td>
<td>[Space for gVar variable]</td>
</tr>
<tr>
<td>...</td>
<td>1226</td>
<td>call proc_a</td>
<td>...</td>
<td>1228</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>0500</td>
<td>(End of proc_a)</td>
<td>(Other modules)</td>
<td>0504</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>2670</td>
<td>entry put_record</td>
<td>...</td>
<td>2534</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>2670</td>
<td>(optional symbol table)</td>
<td>...</td>
<td>0600</td>
<td>(last location in the code segment)</td>
</tr>
</tbody>
</table>

The Program Loaded at Location 4000

<table>
<thead>
<tr>
<th>Relative Code</th>
<th>Address</th>
<th>Generated Code</th>
<th>Data Segment</th>
<th>Address</th>
<th>Generated Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>0000</td>
<td>(Other process’s programs)</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>0800</td>
<td>entry proc_a</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>1116</td>
<td>[Space for gVar variable]</td>
<td>...</td>
<td>1220</td>
<td>load =7, R1</td>
</tr>
<tr>
<td>...</td>
<td>1224</td>
<td>store R1, 0136</td>
<td>...</td>
<td>1226</td>
<td>[Space for gVar variable]</td>
</tr>
<tr>
<td>...</td>
<td>1226</td>
<td>call proc_a</td>
<td>...</td>
<td>1228</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>1500</td>
<td>(End of proc_a)</td>
<td>(Other modules)</td>
<td>1504</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>2674</td>
<td>entry put_record</td>
<td>...</td>
<td>2534</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>2999</td>
<td>(optional symbol table)</td>
<td>...</td>
<td>0600</td>
<td>(last location in the code segment)</td>
</tr>
<tr>
<td>...</td>
<td>0700</td>
<td>(first location in the data segment)</td>
<td>...</td>
<td>0600</td>
<td>(last location in the code segment)</td>
</tr>
<tr>
<td>...</td>
<td>7136</td>
<td>[Space for gVar variable]</td>
<td>...</td>
<td>7136</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>8000</td>
<td>(Other process’s programs)</td>
<td>...</td>
<td>8000</td>
<td>...</td>
</tr>
</tbody>
</table>
Static Memory Allocation

- Issue: Need a mechanism/policy for loading p_i’s address space into primary memory

<table>
<thead>
<tr>
<th>Unused</th>
<th>In Use</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>p_i</td>
</tr>
</tbody>
</table>

Fixed-Partition Memory Mechanism

- p_i needs n_i units

Fixed-Partition Memory Best-Fit

- Internal Fragmentation

Fixed-Partition Memory Worst-Fit

- Loader must adjust every address in the absolute module when placed in memory

Fixed-Partition Memory First-Fit

Fixed-Partition Memory Next-Fit
**Variable Partition Memory**

- Operating System
  - Process 0
  - Process 1
  - Process 2
  - Process 3
  - Process 4

  Loader adjusts every address in every absolute module when placed in memory

- External fragmentation
- Compaction moves program in memory

**Cost of Moving Programs**

- `load R1, 0x02010`
- `3F013010`
- Program loaded at 0x01000

- `3F016010`
- Program loaded at 0x04000

- Must run loader over program again!

*Consider dynamic techniques*

**C-Style Memory Layout**

- Text Segment
- Initialized Part Data Segment
- Uninitialized Part Data Segment
- Heap Storage
- Stack Segment
- Environment Variables, ...

**Program and Process Address Spaces**

- Absolute Program Address Space
- User Process Address Space
- Supervisor Process Address Space

**Multiprogramming Memory Support**

- Unused
- In Use

- Operating System
  - R0
  - R1
  - R2
  - R3
  - R4

- Process 0
- Process 1
- Process 2
- Process 3

**Dynamic Memory Allocation**

- Could use *dynamically allocated* memory
- Process wants to change the size of its address space
  - Smaller ⇒ Creates an external fragment
  - Larger ⇒ May have to move/relocate the program
- Allocate “holes” in memory according to
  - Best- /Worst- / First- /Next-fit
Memory Mgmt Strategies

- Fixed-Partition used only in batch systems
- Variable-Partition used everywhere (except in virtual memory)
- Swapping systems
  - Popularized in timesharing
  - Relies on dynamic address relocation
  - Now dated
- Dynamic Loading (Virtual Memory)
  - Exploit the memory hierarchy
  - Paging -- mainstream in contemporary systems
  - Segmentation -- the future

Moving an Executable Image

Dynamic Address Relocation

- Program loaded at 0x10000 ⇒ Relocation Register = 0x10000
- Program loaded at 0x04000 ⇒ Relocation Register = 0x04000

We never have to change the load module addresses!

Multiple Segment Relocation Registers

Runtime Bound Checking

- Bound checking is inexpensive to add
- Provides excellent memory protection

Special Case: Swapping

- Special case of dynamic memory allocation
- Suppose there is high demand for executable memory
- Equitable policy might be to time-multiplex processes into the memory (also space-mux)
- Means that process can have its address space unloaded when it still needs memory
  - Usually only happens when it is blocked
Swapping

Image for $p_i$

Swap $p_i$ out

Swap $p_j$ in

Image for $p_j$

Sharing a Portion of the Address Space

Process 1

Address Space for Process 1

Process 2

Primary Memory

Address Space for Process 2

Figure 11—26: Multiple Segments

CPU Executing Process 1

Limit Relocation

CPU Executing Process 2

Limit Relocation

Primary Memory