MEMORY MANAGEMENT

Ken Wong
Washington University, St. Louis
kenw@arl.wustl.edu
www.arl.wustl.edu/~kenw

MEMORY MANAGEMENT REQUIREMENTS

- Relocation
  - Swap processes in and out of main memory
- Protection
  - Each process should be protected against unwanted references by other processes
- Sharing
  - Allow processes to share data and programs in main memory
- Support for Program Modules
  - Treat program modules as units (e.g., segmentation)
- Effective Memory Usage
  - Keep most active parts of a process in main memory
  - Store the rest of on secondary storage

PROCESS MEMORY LAYOUT

<table>
<thead>
<tr>
<th>Low Address</th>
<th>Text</th>
<th>Read from program file by exec</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Initialized Data</td>
<td>Initialized to 0 by exec</td>
</tr>
<tr>
<td></td>
<td>Uninitialized Data (bss)</td>
<td></td>
</tr>
<tr>
<td>Heap</td>
<td></td>
<td>Command-line arguments and Environment Variables</td>
</tr>
<tr>
<td></td>
<td>Stack</td>
<td></td>
</tr>
</tbody>
</table>

High Address

PROGRAM IMAGE

```c
main () {
    int x;
    x = 2;
    printf ("x = %d\n", x);
    exit (0);
}
```

- Object File (Partial Program Image)
  ```
  cc -c x.c; nm -ng x.o
  ```

- Static Linking: `cc -static x.o` or `cc -static x.c`
- Dynamic Linking: `cc -dynamic x.c`

U exit
U printf
00000000 A __fser_init_value
00000010 T main
STATIC LINKING RESULT

... 00000000 A Procedure_Linkage_Table...
00000000 A __isr_init_value
00155c0 T _start  // code (text section)
00155c8 T main
0015708 T atexit
0015768 T__exit_handle
00157cc T printf
00158c0 T__exit
0015a60 T__doprnt
0018234 T__mkarg1st
0018678 T__getarg
0018be8 T__div64  //
... 260 other names...
003dc70 D__environ
...
003dc94 D__argv
...
00407c0 B_end

BINDING TIME TO MEMORY ADDRESSES

- Programming Time
  - Actual physical address is specified by the programmer in the program itself
- Compile Time
  - Recompile if starting location changes
- Load Time
  - Need to generate relocatable code
- Execution Time
  - For efficiency, need hardware support if the code will be moved during execution

THREE APPROACHES TO LOADING

- Absolute Loading
  - Each load module is loaded into the same starting main memory location
  - Great potential for address conflict or memory waste
  - Module expansion may require recompiling
- Relocatable Loading
  - Language translator produces addresses which are relative to an address
  - Load module contains data telling loader where relative address references are located in the module
- Dynamic Run-Time Loading
  - Needed if module will be swapped in/out of main memory at dynamically changing locations

OUTPUT OF "nm -ng a.out" FOR -Bdynamic

U__ex_deregister  // From dynamic library
U__ex_register
U__exit
U_atexit
U_exit
U_printf
00000000 A__isr_init_value
00106b0 T__start
00107b8 T main
... a few other names...
0020988 D__environ
...
00209ac D__argv
...
00209b0 B_end
MEMORY PARTITIONING APPROACHES

- Fixed Partitioning
- Dynamic Partitioning
- Buddy System
- Simple Paging
- Simple Segmentation
- Virtual Memory Paging
- Virtual Memory Segmentation

FIXED PARTITIONING

- Divide memory into static partitions at system generation time
- Load process into a partition of equal or greater size

<table>
<thead>
<tr>
<th>OS</th>
<th>OS</th>
<th>OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 MB</td>
<td>4 MB</td>
<td>4 MB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Internal Fragmentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 MB</td>
</tr>
<tr>
<td>12 MB</td>
</tr>
<tr>
<td>4 MB</td>
</tr>
</tbody>
</table>

DYNAMIC PARTITIONING

- Approach: Dynamically create partitions as needed
- Strengths: No internal Fragmentation ⇒ More efficient memory usage
- Weaknesses: Processor overhead to reduce external fragmentation
- Placement Algorithm
  - Decide which free block to place a process during loading or swap in
- Replacement Algorithms (for Dynamic Partitioning)
  - Used when free space is depleted and a process must be loaded or swapped into memory
  - A major issue in virtual memory systems

DYNAMIC PARTITIONING
**PLACEMENT ALGORITHM EXAMPLE**

- **Need**
- **First-Fit**
- **Last Allocated**
- **Next-Fit**

**BUDDY SYSTEM EXAMPLE**

<table>
<thead>
<tr>
<th></th>
<th>16 MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>-1.5M</td>
</tr>
<tr>
<td>A B</td>
<td>-3.5M</td>
</tr>
<tr>
<td>A C B</td>
<td>-1M</td>
</tr>
<tr>
<td>A C B D</td>
<td>-3M</td>
</tr>
<tr>
<td>A C D</td>
<td>+B</td>
</tr>
<tr>
<td>C D</td>
<td>+A</td>
</tr>
<tr>
<td>E C D</td>
<td>-1.5M</td>
</tr>
<tr>
<td>E D</td>
<td>+C</td>
</tr>
<tr>
<td>D</td>
<td>+E</td>
</tr>
</tbody>
</table>

**RELOCATION HARDWARE**

- **Relative Address**
  - **Base Register**
  - **Adder**
    - **Comparator**
    - **Bounds Register**
      - **interrupt OS check**

**SIMPLE PAGING**

- **Basic Idea**
  - Break processes into equal sized **pages** (e.g., $2^{10}$)
  - Break memory into the same sized **page frames**
  - Load process pages into memory page frames (not necessarily contiguous)
- **Like fixed partitioning, but:**
  - Partitions are small
  - A program can occupy more than 1 partition
  - Partitions need not be contiguous
**SIMPLE PAGING EXAMPLE**

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>A.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>A.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>B.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>B.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>C.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>D.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>D.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>D.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>D.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

B. D. o. n. e.

<table>
<thead>
<tr>
<th>Process D</th>
<th>Free Frame List</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page Table</td>
<td>Other Page Tables</td>
</tr>
</tbody>
</table>

**ADDRESS TRANSLATION (PAGING)**

**SIMPLE SEGMENTATION**

- The dynamic partitioning analog to paging
  - Programs are divided into segments (perhaps variable length)
  - A program can occupy more than one segment
  - Segments need not be contiguous

- Comments
  - Suffers from external fragmentation
  - Not invisible to the programmer like paging since the programmer must be aware of the maximum segment size
  - No simple relationship between logical addresses and physical addresses as there is in paging

**ADDRESS TRANSLATION (SEGMENTATION)**