MEMORY MANAGEMENT

Ken Wong
Washington University, St. Louis

kenw@arl.wustl.edu
www.arl.wustl.edu/~kenw

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`

  ```
  U exit
  U printf
  00000000 A __frr_init_value
  00000010 T main
  ```

- **Static Linking**: `cc -Bstatic x.o` or `cc -Bstatic x.c`
- **Dynamic Linking**: `cc -Bdynamic x.c`

PROCESS MEMORY LAYOUT

- **Low Address**
  - Text
  - Initialized Data
  - Uninitialized Data (bss)
  - Heap
  - Stack

- **High Address**
  - Command-line arguments and Environment Variables

- **Read from program file by exec**
- **Initialized to 0 by exec**

PROGRAM LINKING/LOADING

- **Module 1**
- **Module n**

- **Linker**
- **Loader**

- **Library**

- **Memory**

- **Absolute, Relocatable, or Dynamic Run-time**
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

STATIC LINKING RESULT

... 00000000 A _PROCEDURE_LINKAGE_TABLE_
00000000 A __fsr_init_value
000155c0 T _start     // code (text section)
000156c8 T main
00015708 T atexit
00015768 T _exit HANDLE
000157cc T printf
000158c0 T _exit
00015a60 T _doprnt
00018234 T _mkarg1st
00018678 T _getarg
00018be8 T _div64     //
... 260 other names ... 0003dc70 D _environ
... 0003dc94 D __Argv
... 000407c0 B _end

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 __fsr_init_value
000106b0 T _start
000107b8 T main
... a few other names ...
00020988 D _environ
... 000209ac D __Argv
... 000209b0 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
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

INTERNAL FRAGMENTATION

<table>
<thead>
<tr>
<th>OS 8 MB</th>
<th>Internal Fragmentation 12 MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 MB</td>
<td>4 MB</td>
</tr>
<tr>
<td>4 MB</td>
<td>4 MB</td>
</tr>
<tr>
<td>8 MB</td>
<td>8 MB</td>
</tr>
<tr>
<td>16 MB</td>
<td></td>
</tr>
</tbody>
</table>

EXTERNAL FRAGMENTATION

<table>
<thead>
<tr>
<th>OS 8 MB</th>
</tr>
</thead>
<tbody>
<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
**PLACEMENT ALGORITHM EXAMPLE**

- Need
- First-Fit
- Best-Fit
- Next-Fit
- Last Allocated

**BUDDY SYSTEM EXAMPLE**

<table>
<thead>
<tr>
<th>16 MB</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>B</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>C</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>C</td>
<td>B</td>
<td>D</td>
</tr>
<tr>
<td>A</td>
<td>C</td>
<td>B</td>
<td>D</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>C</td>
<td>D</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>E</td>
<td>D</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- A - 1.5 M
- A - 3.5 M
- A - 1 M
- A - 3 M
- A - 3 M
- A - 1.5 M
- A - 1.5 M

**RELOCATION HARDWARE**

- Relative Address
- Base Register
- Adder
- Comparator
- Bounds Register
- Absolute Address
- Interrupt OS
- PCB
- Text
- Data
- Heap
- Stack

**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)
  - Some internal fragmentation (last page), but no external fragmentation
- **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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A.0</td>
<td>A.1</td>
<td>B.0</td>
<td>B.1</td>
<td>C.0</td>
<td>D.0</td>
<td>D.1</td>
<td>D.2</td>
</tr>
<tr>
<td>D.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

B Done

Process D Free Frame List

| Process D Page Table | 0 | 2 |
|----------------------|--|--
| 1                   | 3|
| 2                   | 5|
| 3                   | 6|

Other Page Tables

ADDRESS TRANSLATION (PAGING)

Logical Address

Page # Offset

0 00001 01110111110
1 000011
2 000101

Process Page Table

0001101110111110

Physical Address

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)

Logical Address

Segment # Offset

0 00010 01111110

Process Segment Table

0 0001011101111110

Length Base

Physical Address