CMSC 321: Operating Systems

Lecture 11

Memory Management

Reading: Silberschatz Chs. 8 and 9
Memory Management?

• Principle Operation: 
  bring programs into RAM for execution

• Memory Hierarchy:
  – speed/cost vs. size; volatility

• OS must coordinate and manage
  – transparent to user
Memory Requirements

• Relocation
  – move processes w/in RAM: swapping

• Protection
  – protect processes from one another

• Sharing
  – allowed some controlled access (e.g., editor)

• Logical Organization
  – we program using modules (functions, globals, etc.)
  – rather than RAM as contiguous block

• Physical Organization
  – RAM vs. secondary: transparent to user
Fixed Partitioning

- Partition RAM into fixed-size regions

- Problems?
  - if program too big for partition
  - internal fragmentation

- Non-equal partitions help little
Dynamic Partitioning

- Partitions: vary # and length
- When program brought in allocate amt needed, no more

Problems?
Dynamic Partitioning

- Partitions: vary # and length
- When program brought in allocate amt needed, no more
- Problems? external fragmentation
Dynamic Partitioning Algs

• How to place programs into memory?
  – Best-fit:
    find partition that results in least waste
  – First-fit:
    scan from beginning until program fits
  – Next-fit:
    scan starting from last placement

• Which is best?
  – first-fit: simplest, best, fastest
  – best-fit: usually worst performer (why?)
  – next-fit: slightly worse than first-fit
What is the Problem?

- Requires contiguous blocks in memory
  - What if there is enough memory for a new process, but not in one chunk?
  - Compaction (like disk defragmentation)

- What happens if a process needs to grow?

- What do we have to do when a process terminates?

- Would like to “spread” a process across RAM if necessary
Paging

• Partition **RAM** into small, equal fixed-sized chunks
• Partition **process** into similarly sized chunks

• **Frame**: chunk of physical memory
• **Page**: chuck of logical memory (process)

• Fragmentation?
  – no external
  – internal only in last page of process
Paging

(a) Fifteen Available Frames
(b) Load Process A
(c) Load Process B
### Paging

<table>
<thead>
<tr>
<th>Main memory</th>
<th>Main memory</th>
<th>Main memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A.0</td>
<td>A.0</td>
</tr>
<tr>
<td>1</td>
<td>A.1</td>
<td>A.1</td>
</tr>
<tr>
<td>2</td>
<td>A.2</td>
<td>A.2</td>
</tr>
<tr>
<td>3</td>
<td>A.3</td>
<td>A.3</td>
</tr>
<tr>
<td>4</td>
<td>B.0</td>
<td>D.0</td>
</tr>
<tr>
<td>5</td>
<td>B.1</td>
<td>D.1</td>
</tr>
<tr>
<td>6</td>
<td>B.2</td>
<td>D.2</td>
</tr>
<tr>
<td>7</td>
<td>C.0</td>
<td>C.0</td>
</tr>
<tr>
<td>8</td>
<td>C.1</td>
<td>C.1</td>
</tr>
<tr>
<td>9</td>
<td>C.2</td>
<td>C.2</td>
</tr>
<tr>
<td>10</td>
<td>C.3</td>
<td>C.3</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- (d) Load Process C
- (e) Swap out B
- (f) Load Process D
Paging
Address Translation in Paging System

Diagram showing the process of address translation in a paging system, with virtual addresses being translated into physical frame addresses through the paging mechanism and page table.
Data Structures Needed

- Page table
  - one per process
  - associate page w/ frame
  - pointer to table in PCB

- Frame table
  - track free/occupied frames (e.g. w/ bitmap)
Hierarchical: Two-Level Scheme
Addr. Trans. – 2 Level

Program

Virtual Address

10 bits 10 bits 12 bits

Root page table (contains 1024 PTEs)

Frame # Offset

4-kbyte page table (contains 1024 PTEs)

Paging Mechanism

Main Memory
Page Size

• Factors to consider:
  - 1. internal fragmentation
    small pages : less IF
    small pages : more pages/process
    - larger page tables
  - 2. secondary memory
    large pages : more efficient xfers
  - 3. effect on page fault rate
    As page size increases, principle of locality weakens
    fewer pages available in memory (More on this later)
  - 4. program complexity
    OOP, threads reduce locality
Advantages of Paging

- Don’t need contiguous space in RAM
- Reduces IF, eliminates EF
- Abstract user view of memory from physical
- Allow more processes in RAM
- Protection (bits w/ each page)
- Ability to share reentrant code
  - never changes during execution (more later)