Chapter 9: Virtual Memory
Virtual memory – separation of user logical memory from physical memory.

- Only part of the program needs to be in memory for execution
- Logical address space can therefore be much larger than physical address space
- Allows address spaces to be shared by several processes
- Allows for more efficient process creation

Virtual memory can be implemented via:
- Demand paging
- Demand segmentation
Virtual-address Space

![Diagram of virtual-address space]

- **Code**
- **Data**
- **Heap**
- **Stack**
Virtual Memory That is Larger Than Physical Memory

- page 0
- page 1
- page 2
- ...
- ...
- page \( v \)

Virtual memory

Memory map

Physical memory
Shared Library Using Virtual Memory

Shared library

stack

heap

data

code

Shared pages

stack

shared library

heap

data

code
Copy-on-Write

- Copy-on-Write (COW) allows both parent and child processes to initially share the same pages in memory.
  
  If either process modifies a shared page, only then is the page copied.

- COW allows more efficient process creation as only modified pages are copied.

- Free pages are allocated from a pool of zeroed-out pages.
Before Process 1 Modifies Page C
After Process 1 Modifies Page C
Demand Paging

- Bring a page into memory only when it is needed: swapper that deals with pages is a pager
  - Less I/O needed
  - Less memory needed
  - Faster response
  - More users
Valid-Invalid Bit

- With each page table entry a valid–invalid bit is associated (v ⇒ in-memory, i ⇒ not-in-memory)
- Initially valid–invalid bit is set to i on all entries
- Example of a page table snapshot:

<table>
<thead>
<tr>
<th>Frame #</th>
<th>valid-invalid bit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>v</td>
</tr>
<tr>
<td></td>
<td>v</td>
</tr>
<tr>
<td></td>
<td>v</td>
</tr>
<tr>
<td></td>
<td>v</td>
</tr>
<tr>
<td></td>
<td>i</td>
</tr>
<tr>
<td></td>
<td>i</td>
</tr>
<tr>
<td></td>
<td>i</td>
</tr>
</tbody>
</table>

- During address translation, if valid–invalid bit in page table entry is i ⇒ page fault
Page Table When Some Pages Are Not in Main Memory

- Logical memory:
  - Pages: A, B, C, D, E, F, G, H
  - Frame bit: 0, 1, 2, 3, 4, 5, 6, 7

- Page table:
  - Frame bit: 0, 1, 2, 3
  - Valid-Invalid bit: v, i

- Physical memory:
  - Pages: A, B, C, D, E, F, G, H

- Frame:
  - Pages A, C, F, H in memory
  - Pages B, D, E, G not in memory
Page Fault

1. Operating system looks at another table to decide:
   - Invalid reference ⇒ abort
   - Just not in memory
2. Get empty frame
3. Swap page into frame
4. Reset tables
5. Set validation bit = v
6. Restart the instruction that caused the page fault
Performance of Demand Paging

- Page Fault Rate \(0 \leq p \leq 1.0\)
  - if \(p = 0\) no page faults
  - if \(p = 1\), every reference is a fault

- Effective Access Time (EAT)
  \[
  EAT = (1 - p) \times \text{memory access} + p \times (\text{page fault overhead} + \text{swap page out} + \text{swap page in} + \text{restart overhead})
  \]
Demand Paging Example

- Memory access time = 200 nanoseconds, Average page-fault service time = 8 milliseconds

- EAT = (1 – p) x 200 + p (8 milliseconds)
  = (1 – p) x 200 + p x 8,000,000
  = 200 + p x 7,999,800

- If one access out of 1,000 causes a page fault, then
  EAT = 8.2 microseconds.
  This is a slowdown by a factor of 40!!