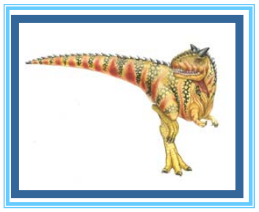
Chapter 9: Virtual Memory



(slides improved by R. Doemer, 05/27/10)

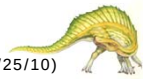
Chapter 9: Virtual Memory

- Background
- Demand Paging
- Copy-on-Write
- Memory-Mapped Files
- Page Replacement
- Allocation of Frames
- Thrashing
- Allocating Kernel Memory
- Other Considerations
- Operating-System Examples

(slide modified by R. Doemer, 02/01/11)

Objectives

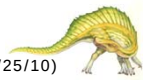
- To describe the benefits of a virtual memory system
- To explain the concepts of
  - demand paging,
  - page-replacement algorithms, and
  - allocation of page frames
- To discuss the principle of the working-set model



(slide modified by R. Doemer, 05/25/10)

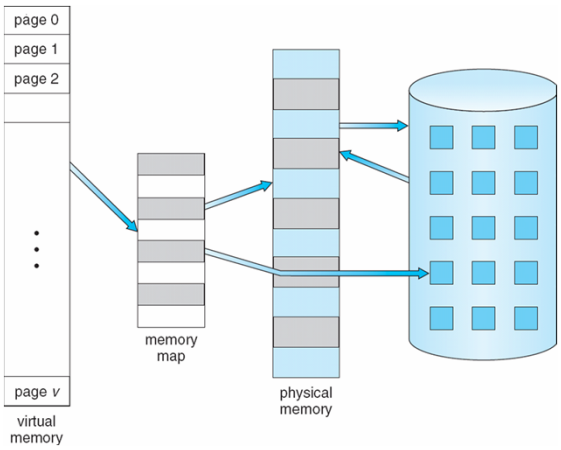
Background

- **Virtual memory** –
  complete separation of user logical memory from physical memory.
  - Only *part* of a program needs to be in memory for its 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

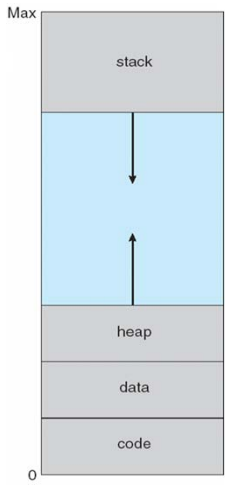


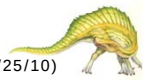
(slide modified by R. Doemer, 05/25/10)

Virtual Memory That is Larger Than Physical Memory



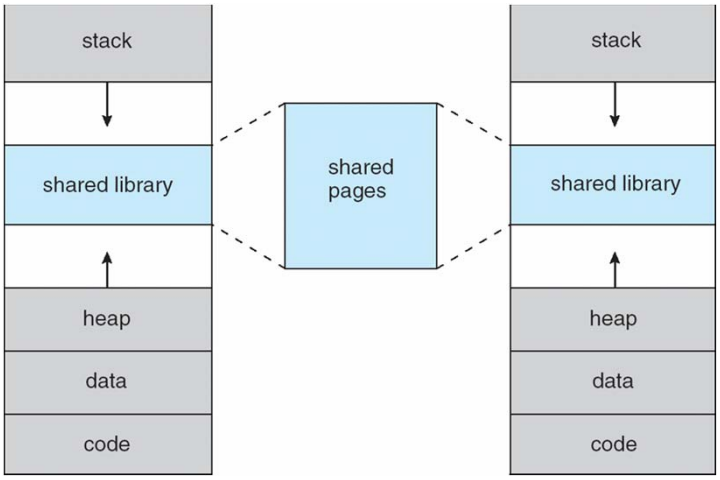
Virtual Address Space





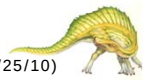
(slide modified by R. Doemer, 05/25/10)

Shared Library Using Virtual Memory



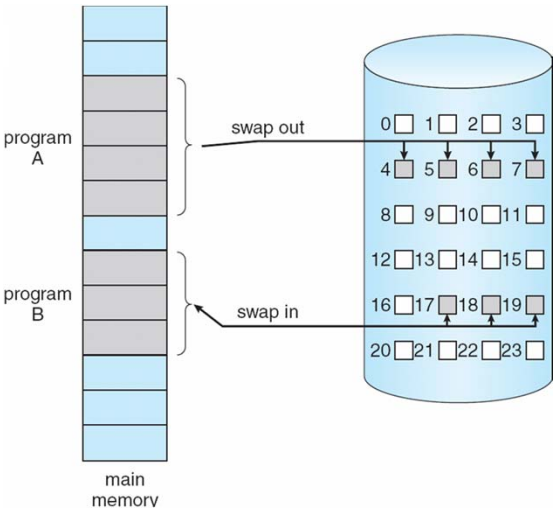
Demand Paging

- Bring a page into memory *only when* it is needed
  - Less I/O needed
  - Less memory needed
  - Faster response
  - More users
- Page is *needed*
  ⇒ when a CPU instruction *references* an address in it (e.g. load, store)
- **Page Fault**
  - invalid reference ⇒ abort
  - not-in-memory ⇒ bring to memory
- **Lazy swapper** –
  never swaps a page into memory unless page will be needed
  - Swapper that deals with pages is a **pager**



(slide modified by R. Doemer, 05/25/10)

Transfer of a Paged Memory to Contiguous Disk Space



Valid-Invalid Bit

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

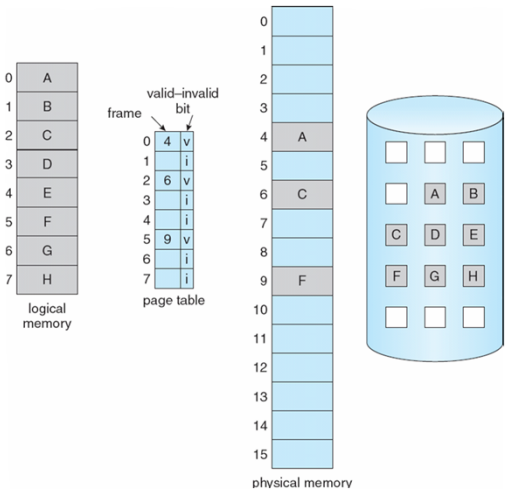
Frame #	valid-invalid bit
	v
	v
	v
	v
	i
….	
	i
	i

page table

- During address translation, if valid–invalid bit in page table entry is **i** ⇒ **page fault**

(slide modified by R. Doemer, 05/25/10)

Page Table When Some Pages Are Not in Main Memory



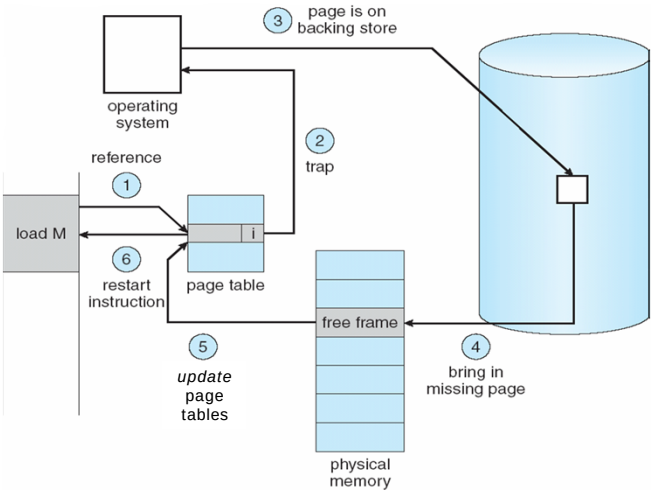
Page Fault

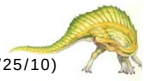
- If a page is not in main memory,
  the first reference to that page will **trap** to the operating system:
  page fault

1. Operating system looks at another table to decide:
   - Invalid reference ⇒ abort
   - Just not in memory ⇒ goto step 2
2. Get empty frame
3. Swap page into frame
4. Update tables
5. Set valid-invalid bit to **v**
6. **Restart** the instruction that caused the page fault

(slide modified by R. Doemer, 05/25/10)

Steps in Handling a Page Fault

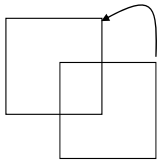




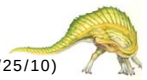
(slide modified by R. Doemer, 05/25/10)

Handling a Page Fault

- Restart instruction:
  - sometimes not trivial!
  - Special care may need to be taken!
- Example 1: block move instruction where blocks span multiple pages



- Example 2: auto increment/decrement instruction



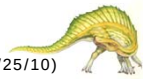
(slide modified by R. Doemer, 05/25/10)

Additional Virtual Memory Benefits

- Virtual memory allows other benefits:

  - During Process Creation: Copy-on-Write

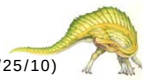
  - Memory-Mapped Files



(slide modified by R. Doemer, 05/25/10)

Copy-on-Write

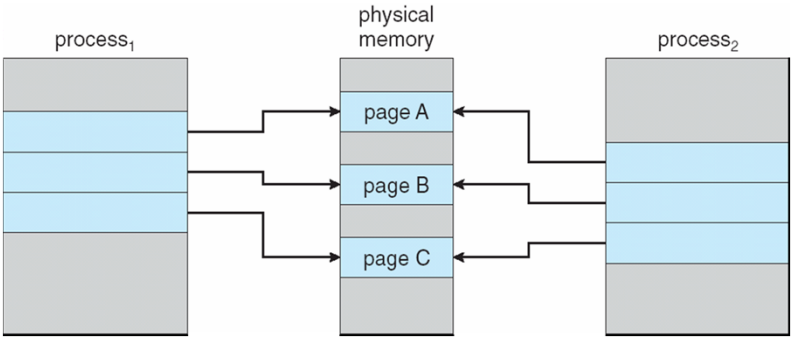
- Consider parent process forks a child process
- **Copy-on-Write** (COW) allows both parent and child processes to initially *share* the same pages in memory
- If either process modifies a shared page, only then is the page copied
- COW allows more efficient process creation as only modified pages are copied
- Free pages are allocated from a **pool** of zeroed-out pages

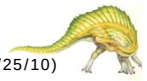


(slide modified by R. Doemer, 05/25/10)

Copy-on-Write Example

- Before Process 1 Modifies Page C

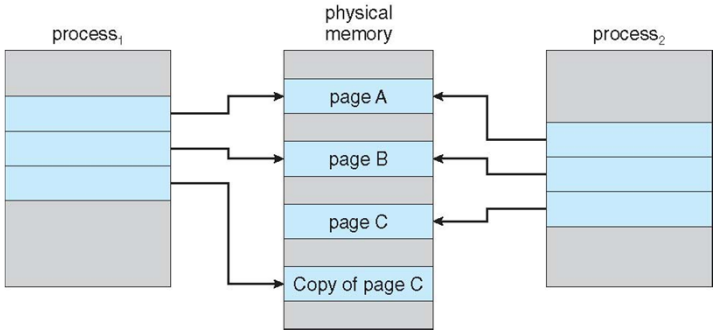




(slide modified by R. Doemer, 05/25/10)

Copy-on-Write Example

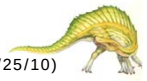
- After Process 1 Modifies Page C



(slide modified by R. Doemer, 05/25/10)

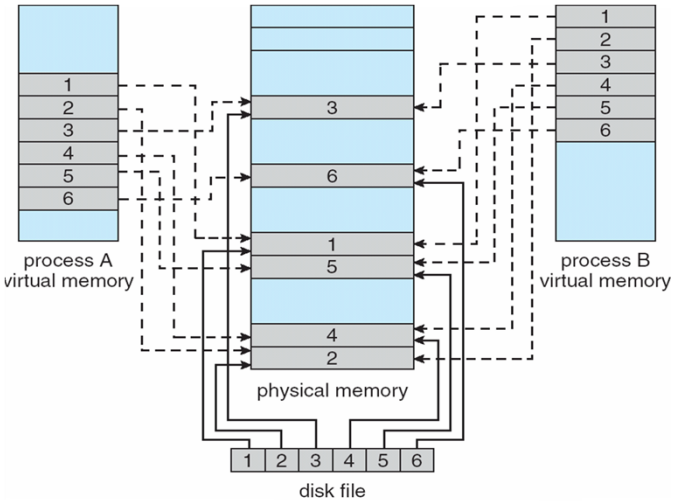
Memory-Mapped Files

- Memory-mapped file I/O allows file I/O to be treated as regular memory access by **mapping** a disk block to a page in memory.
- A file is initially read using demand paging.
- A page-sized portion of the file is read from the file system into a physical memory frame.
- Subsequent reads/writes to/from the file are treated as ordinary memory accesses.
- Simplifies file access by treating file I/O as ordinary memory access rather than **`read()`** and **`write()`** system calls
- Also allows several processes to map the same file allowing the pages in memory to be shared



(slide modified by R. Doemer, 05/25/10)

Memory-Mapped Files



(slide modified by R. Doemer, 05/25/10)