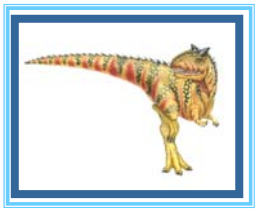
Chapter 9: Virtual Memory



(slides selected/reordered/fixed by R. Doemer, 02/02/09)

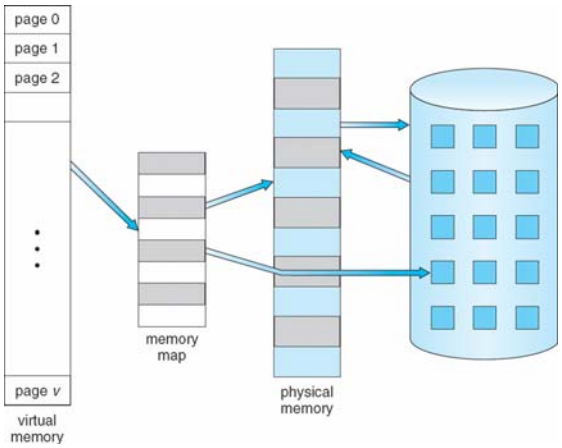
Chapter 9: Virtual Memory

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

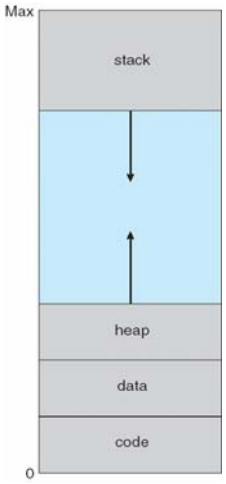
Background

- **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

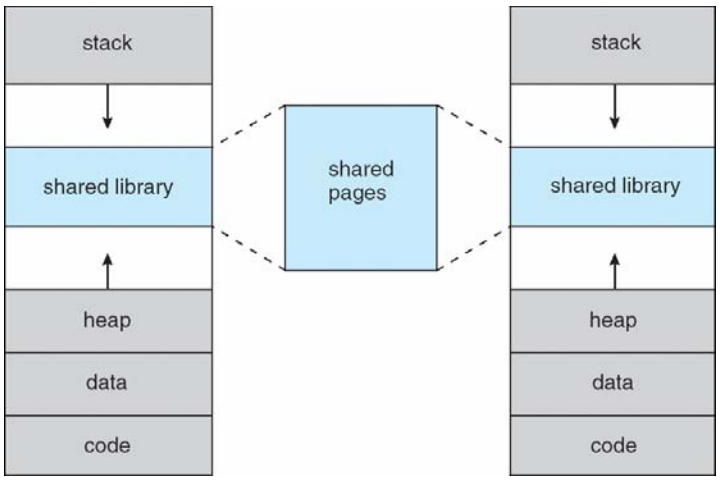
Virtual Memory That is Larger Than Physical Memory



Virtual-address Space



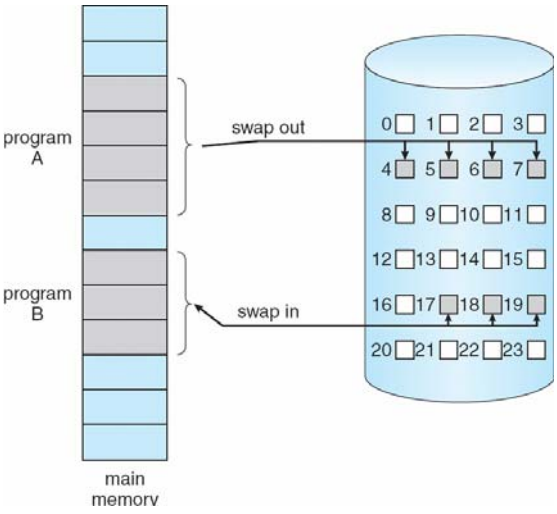
Shared Library Using Virtual Memory



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 ⇒ reference to it
  - 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**

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:

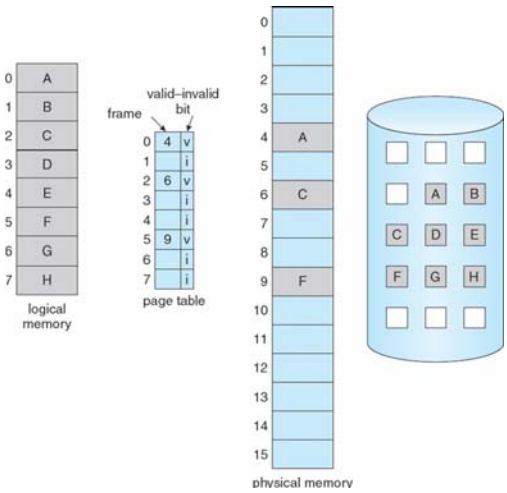
Frame #	valid-invalid bit
	v
	v
	v
	v
	i
….	
	i
	i

page table

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

(slide fixed by R. Doemer, 02/02/09)

Page Table When Some Pages Are Not in Main Memory



Page Fault

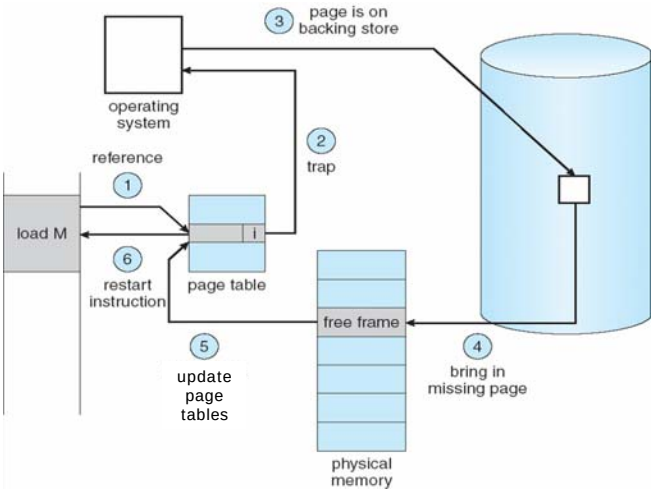
- If there is a reference to a page, first reference to that page will trap to operating system:

  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 fixed by R. Doemer, 02/02/09)

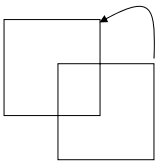
Steps in Handling a Page Fault



(slide fixed by R. Doemer, 02/02/09)

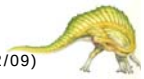
Page Fault (Cont.)

- Restart instruction: sometimes not trivial!
  Special care may need to be taken!
  - Example 1: block move



  - Example 2: auto increment/decrement location

(slide fixed by R. Doemer, 02/02/09)



Process Creation

- Virtual memory allows other benefits during process creation:

  - Copy-on-Write

  - Memory-Mapped Files (later)

Copy-on-Write

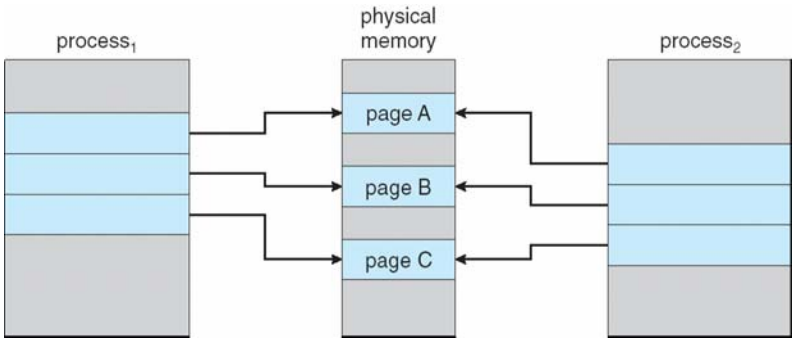
- Copy-on-Write (COW) allows both parent and child processes to initially *share* the same pages in memory

  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

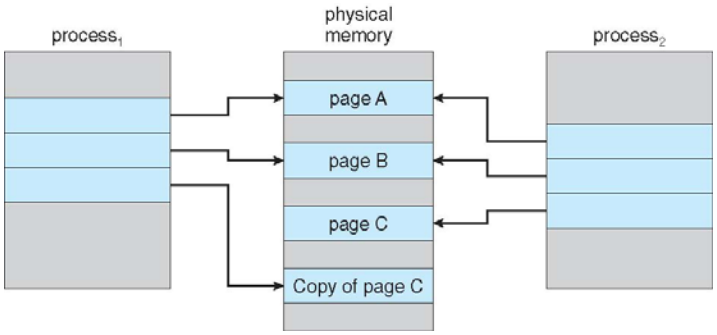
- Free pages are allocated from a **pool** of zeroed-out pages

Before Process 1 Modifies Page C



(slide fixed by R. Doemer, 02/02/09)

After Process 1 Modifies Page C



What happens if there is no free frame?

- Page replacement – find some page in memory, that is not really in use, swap it out
  - Algorithm to find victim page
  - Performance – we want an algorithm which will result in minimum number of page faults
- Same page may be brought into memory several times

(slide fixed by R. Doemer, 02/02/09)

Page Replacement

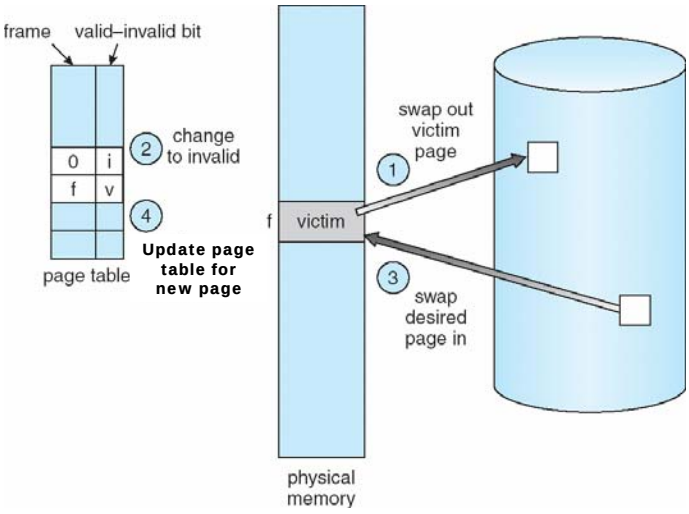
- Prevent over-allocation of memory by modifying page-fault service routine to include page replacement
- Use **modify (dirty) bit** to reduce overhead of page transfers – only modified pages are written to disk
- Page replacement completes separation between logical memory and physical memory – large virtual memory can be provided on a smaller physical memory

Basic Page Replacement

1. Find the location of the desired page on disk
2. Find a free frame:
   - If there is a free frame,
     use it
   - If there is no free frame,
     use page replacement algorithm to select a **victim** frame
3. Swap out the victim page;
   bring the desired page into the (newly) free frame;
   update the page and frame tables
4. Restart the instruction

(slide fixed by R. Doemer, 02/02/09)

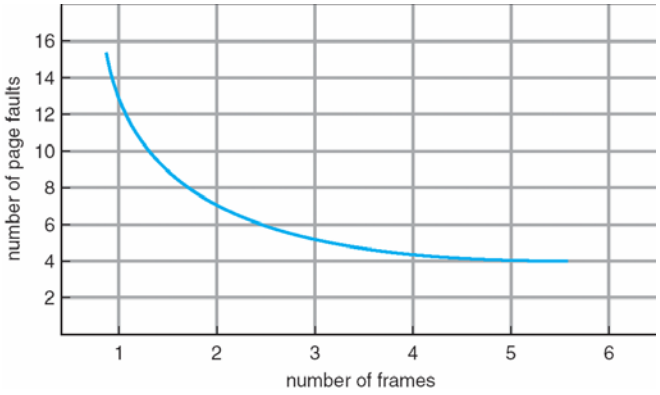
Page Replacement



(slide fixed by R. Doemer, 02/02/09)

Graph of Page Faults Versus The Number of Frames

General expectation:



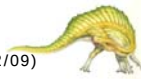
(slide fixed by R. Doemer, 02/02/09)

Page Replacement Algorithms

- Want lowest page-fault rate
- Evaluate algorithm by
  running it on a particular string of memory references (reference string)
  and counting the number of page faults on that string
- In all our examples, the reference string is

  1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

(slide fixed by R. Doemer, 02/02/09)



First-In-First-Out (FIFO) Algorithm

- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- 3 frames (3 pages can be in memory at a time per process)

1	1	4	5	
2	2	1	3	9 page faults
3	3	2	4	

(slide split by R. Doemer, 02/02/09)

First-In-First-Out (FIFO) Algorithm

- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- 3 frames (3 pages can be in memory at a time per process)

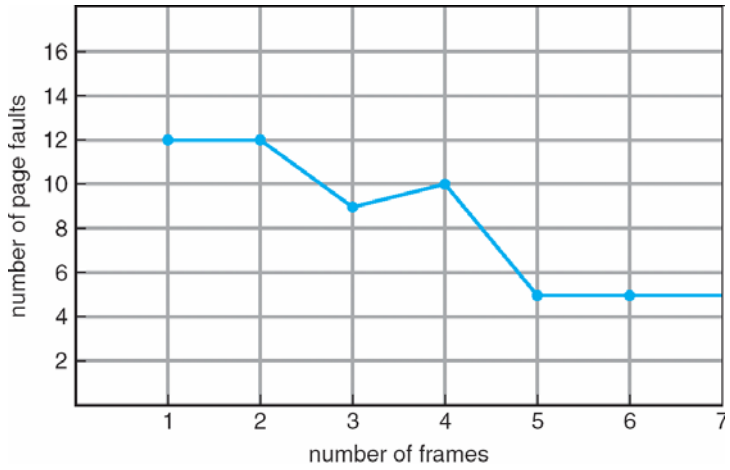
1	1	4	5		
2	2	1	3	9 page faults	
3	3	2	4		

- 4 frames

1	1	5	4		
2	2	1	5	10 page faults	
3	3	2			
4	4	3			

- Belady's Anomaly: more frames ⇒ more page faults

FIFO Illustrating Belady's Anomaly



Optimal Algorithm

- Replace page that will not be used for longest period of time
- 4 frames example

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

4

6 page faults

5

- How do you know this?
- Used for measuring how well your algorithm performs

Least Recently Used (LRU) Algorithm

- Reference string: 1, 2, 3, 4, 1, 2, **5**, 1, 2, **3**, **4**, **5**

1	1	1	1	**5**
2	2	2	2	2
3	**5**	5	**4**	4
4	4	**3**	3	3

- Counter implementation
  - Every page entry has a counter; every time page is referenced through this entry, copy the clock into the counter
  - When a page needs to be changed, look at the counters to determine which are to change

(slide fixed by R. Doemer, 02/02/09)

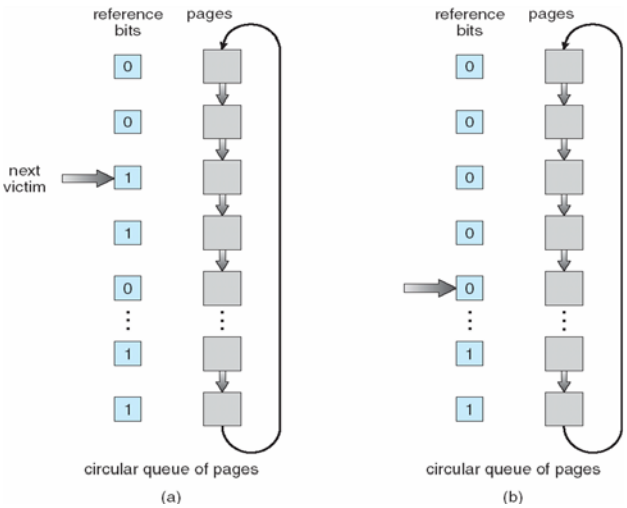
LRU Algorithm (Cont.)

- Stack implementation – keep a stack of page numbers in a double link form:
  - Page referenced:
    - move it to the top
    - requires 6 pointers to be changed
  - No search for replacement

LRU Approximation Algorithms

- Reference bit
  - With each page associate a bit, initially = 0
  - When page is referenced bit set to 1
  - Replace the one which is 0 (if one exists)
    - We do not know the order, however
- Second chance
  - Need reference bit
  - Clock replacement
  - If page to be replaced (in clock order) has reference bit = 1 then:
    - set reference bit 0
    - leave page in memory
    - replace next page (in clock order), subject to same rules

Second-Chance (clock) Page-Replacement Algorithm



Counting Algorithms

- Keep a counter of the number of references that have been made to each page
- **LFU Algorithm**: replaces page with smallest count
- **MFU Algorithm**: based on the argument that the page with the smallest count was probably just brought in and has yet to be used