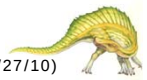
Chapter 9: Virtual Memory

- 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, 02/03/11)

Virtual Memory

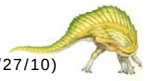
- When handling a page fault,
  what happens if there is no free frame?
- **Page replacement** –
  find some page in memory, that is not really in use, swap it out
  - Algorithm needed to find victim page
  - Performance – we want an algorithm
    which will result in *minimum number of page faults*
- Thus, same page may be brought into memory several times



(slide modified by R. Doemer, 05/27/10)

Page Replacement

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

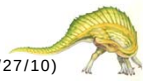


(slide modified by R. Doemer, 05/27/10)

Page Replacement

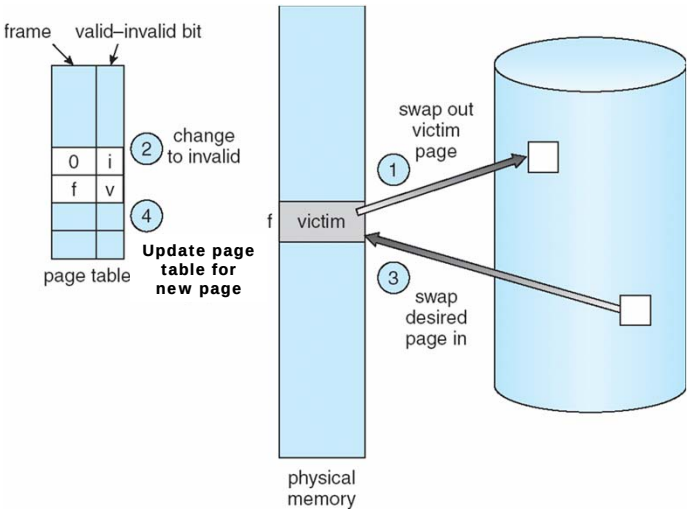
Extended page fault service routine:

- Page fault, find the location of the desired page on disk
- Find a free frame:
  - If there is a free frame,
    use the free frame
  - If there is no free frame,
    use **page replacement** algorithm to select a **victim** frame
    if modified/dirty, swap out the victim page
- Bring the desired page into the (new) free frame
- *Update* the page and frame tables
- *Restart* the instruction



(slide modified by R. Doemer, 05/27/10)

Page Replacement



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

Page Replacement Algorithms

- Many page replacement algorithms are possible
- 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 the following examples, the reference string is

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

(slide modified by R. Doemer, 05/27/10)

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 modified by R. Doemer, 05/27/10)

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		

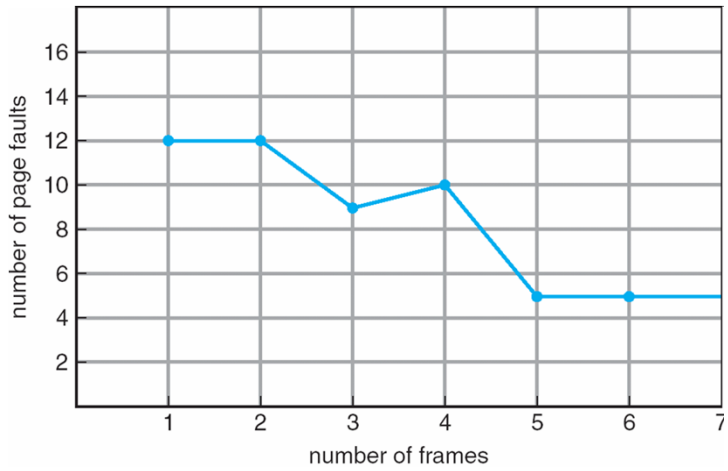
- 4 frames

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

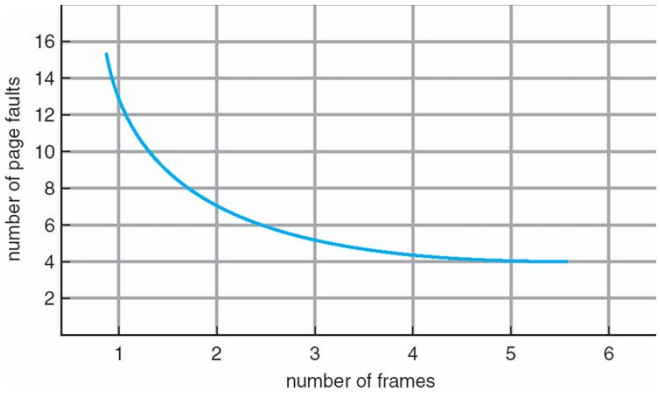
(slide modified by R. Doemer, 05/27/10)

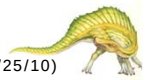
FIFO Illustrating Belady's Anomaly



Page Faults Versus The Number of Frames

General expectation:





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

Optimal Algorithm

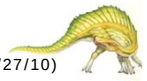
- Replace page that *will not be used* for longest period of time
- 4 frames example

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

6 page faults

- How do you know this?

- Optimal Algorithm: For comparison only!
  Used for measuring how well other algorithms perform.



(slide modified by R. Doemer, 05/27/10)

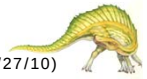
Least Recently Used (LRU) Algorithm

- 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

8 page faults

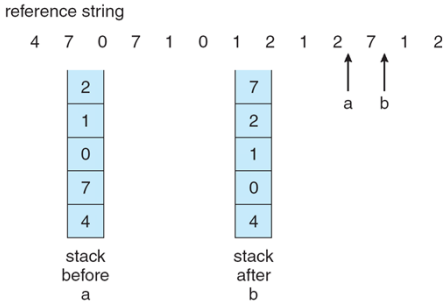
- Possible implementation by counters/clock
  - Every page entry has a counter/clock associated with it
  - Every time a page is referenced, copy clock into its counter
  - When a page needs to be changed,
    find the smallest counter to determine which page to replace

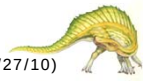


(slide modified by R. Doemer, 05/27/10)

Least Recently Used (LRU) Algorithm

- Alternative implementation by use of a stack – keep a stack of page numbers in a doubly linked list:
  - Whenever a page is referenced
    - move it to the top
  - Requires 6 pointers to be changed
  - No search needed for replacement

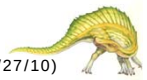




(slide modified by R. Doemer, 05/27/10)

LRU Approximation Algorithms

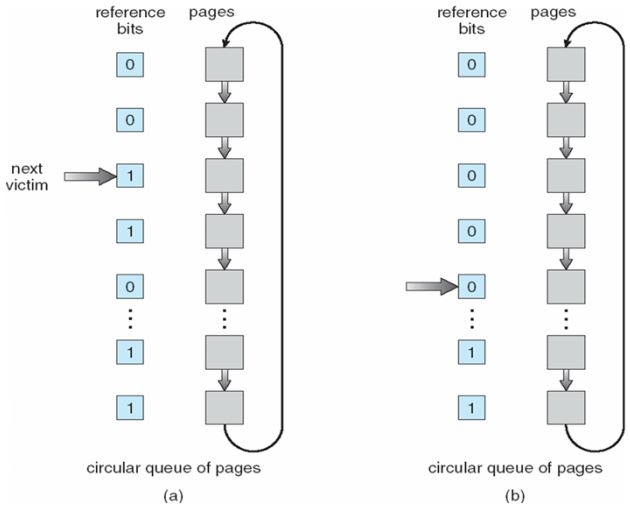
- LRU Algorithm is quite expensive to implement
  - **LRU approximation algorithms** are often used instead
- **Reference bit**
  - With each page associate a bit, initially set to 0
  - When page is referenced, set bit to 1 (in hardware)
  - Replace a page whose bit is 0 (if one exists)
    - We do not know the order, however
- **Second chance algorithm** (aka. **Clock algorithm**)
  - Use FIFO replacement as basic algorithm
  - Add a reference bit as above
  - Consider pages to be replaced in circular order (clock order)
  - If a page is to be replaced
    - if reference bit = 1, then reset bit = 0 and leave page in memory
    - if reference bit = 0, replace this page



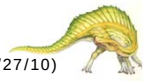
(slide modified by R. Doemer, 05/27/10)

LRU Approximation Algorithms

- **Second-Chance** (**Clock**) Page Replacement Algorithm

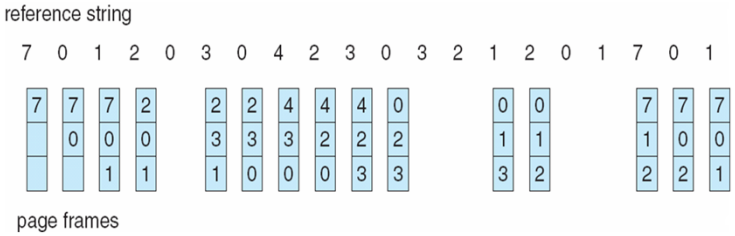


(slide modified by R. Doemer, 05/27/10)



Page Replacement Algorithms

- Example: FIFO Algorithm

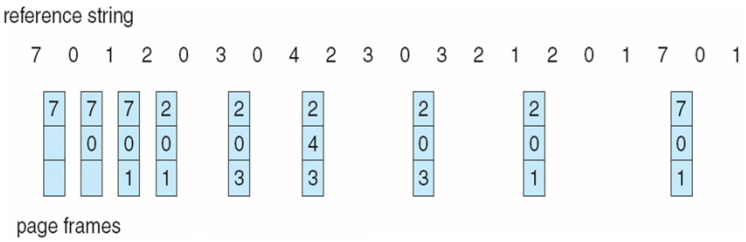


- 15 page faults

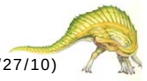
(slide modified by R. Doemer, 05/27/10)

Page Replacement Algorithms

- Example: Optimal Algorithm



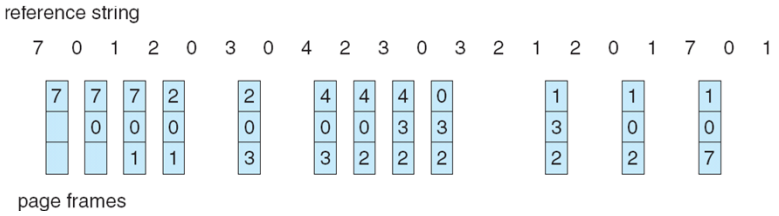
- 9 page faults



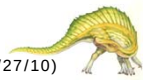
(slide modified by R. Doemer, 05/27/10)

Page Replacement Algorithms

- Example: LRU Algorithm



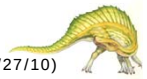
- 12 page faults



(slide modified by R. Doemer, 05/27/10)

Page Replacement Algorithms

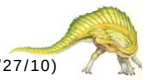
- Alternative Algorithms include **Counting Algorithms**
  - Keep a counter of the number of references that have been made to each page
  - **LFU Algorithm**: least-frequently used replacement
    - replaces page with smallest count
    - frequently used pages stay in memory
  - **MFU Algorithm**: most-frequently used replacement
    - replaces page with largest count
    - based on the argument that the page with the smallest count was probably just brought in and has yet to be used



(slide modified by R. Doemer, 05/27/10)

Allocation of Frames

- Each process needs a *minimum* number of pages
- Example: IBM 370 – 6 pages to handle SS MOVE instruction:
  - instruction is 6 bytes, might span 2 pages
  - 2 pages to handle *from*
  - 2 pages to handle *to*
- **Allocation of Frames**:
  Two major schemes exist
  - Fixed allocation
  - Priority allocation



(slide modified by R. Doemer, 05/27/10)

Fixed Allocation

- **Equal allocation** –
  For example, if there are 100 frames and 5 processes, give each process 20 frames.
- **Proportional allocation** –
  Allocate according to the size of the process
  Example:
  - s_i = size of process p_i
  - $S = \sum s_i$
  - m = total number of frames
  - a_i = allocation for $p_i = \frac{s_i}{S} \times m$

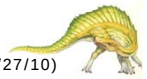
$m = 64$

$s_i = 10$

$s_2 = 127$

$a_1 = \frac{10}{137} \times 64 \approx 5$

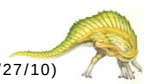
$a_2 = \frac{127}{137} \times 64 \approx 59$



(slide modified by R. Doemer, 05/27/10)

Priority Allocation

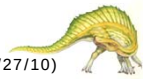
- Use a proportional allocation scheme using **priorities** rather than size
- If process P_i generates a page fault,
  - select for replacement one of its own frames, or
  - select for replacement a frame from a process with lower priority number
- **Global replacement** –
  select a replacement frame from the set of all frames; one process can take a frame from another
- **Local replacement** –
  select a replacement frame from only processes' own set of allocated frames



(slide modified by R. Doemer, 05/27/10)

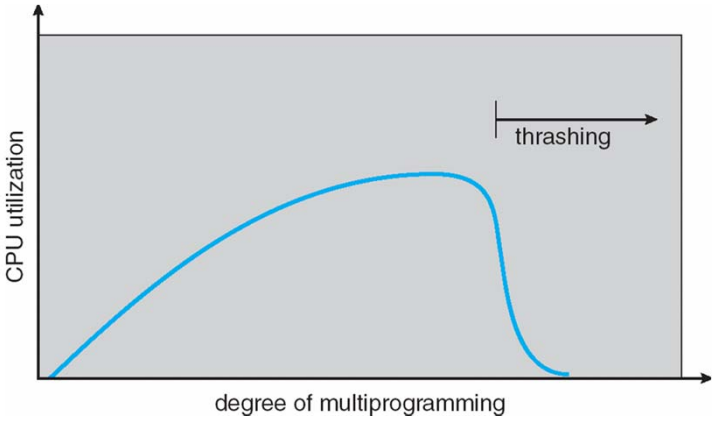
Thrashing

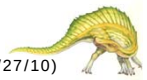
- If a process has *"not enough"* pages, the page-fault rate is very high.
- This leads to:
  - low CPU utilization
  - operating system thinks that it needs to increase the degree of multiprogramming
  - another process is added to the system
  - even less pages become available…
- **Thrashing** ≡ a process is constantly swapping pages in and out



(slide modified by R. Doemer, 05/27/10)

Thrashing Phenomenon

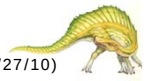




(slide modified by R. Doemer, 05/27/10)

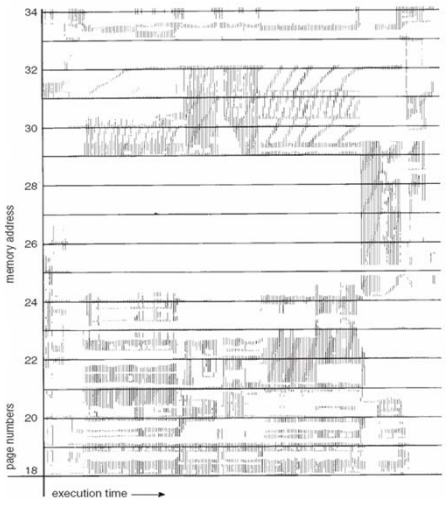
Demand Paging and Thrashing

- Why does demand paging work?
- **Locality model**
  - Process migrates from one locality to another
  - Localities may overlap
- When does thrashing occur?
- Σ size of locality > available memory size



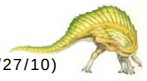
(slide modified by R. Doemer, 05/27/10)

Locality In A Memory-Reference Pattern



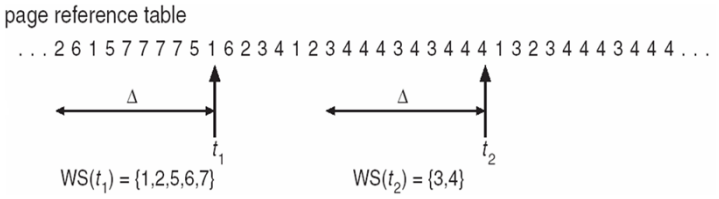
Working-Set Model

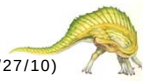
- $\Delta \equiv$ **working-set window** $\equiv$ a fixed number of page references
  Example: sequence of 10,000 instructions
- WSS_i (**working set size** of Process P_i) =
  total number of pages referenced in the most recent Δ
  (varies in time)
  - if Δ is too small, it will not encompass the entire locality
  - if Δ is too large, it will encompass several localities
  - if $\Delta = \infty$, it will encompass the entire program
- $D = \Sigma\ WSS_i \equiv$ total demand of frames of all processes
- if $D > m \Rightarrow$ Thrashing occurs!
- Policy:
  if $D >$ m, then suspend (swap out) one of the processes



(slide modified by R. Doemer, 05/27/10)

Working-Set Model

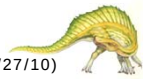




(slide modified by R. Doemer, 05/27/10)

Keeping Track of the Working Set

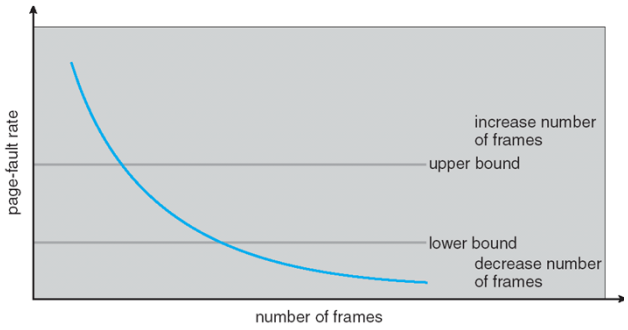
- Approximation with
  - interval timer
  - a reference bit in hardware
  - Set of reference bits associated with each page
- Example: Δ = 10,000 time units
  - Timer interrupts after every 5000 time units
  - Keep in memory 2 additional bits for each page
  - Whenever the timer interrupts,
    shift the bits in memory,
    copy the hardware bits to the first bit in memory, and
    set the values of all hardware reference bits to 0
  - If one of the memory bits = 1 $\Rightarrow$ page in working set
- Why is this not completely accurate?
  - Can't tell when exactly reference occurred
- Improvement: 10 bits and interrupt every 1000 time units



(slide modified by R. Doemer, 05/27/10)

Page-Fault Frequency Scheme

- Establish "acceptable" page-fault rate
  - If actual rate too low, process loses frame
  - If actual rate too high, process gains frame



Other Issues – Program Structure

- Program structure
  - `int data[128,128];`
  - Each row is stored in one page
  - Program 1

```
for (j = 0; j < 128; j++)
    for (i = 0; i < 128; i++)
        data[i,j] = 0;
```

128 x 128 = 16,384 page faults

  - Program 2

```
for (i = 0; i < 128; i++)
    for (j = 0; j < 128; j++)
        data[i,j] = 0;
```

128 page faults

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

End of Chapter 9