

## LECTURE - XIII VIRTUAL MEMORY

Tevfik Koşar

Louisiana State University  
March 20<sup>th</sup>, 2007

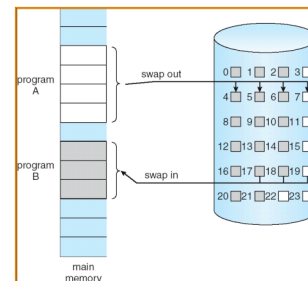
## Background

- **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

## 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  $\Rightarrow$  reference to it
  - invalid reference  $\Rightarrow$  abort
  - not-in-memory  $\Rightarrow$  bring to memory

## Transfer of a Paged Memory to Contiguous Disk Space



## Valid-Invalid Bit

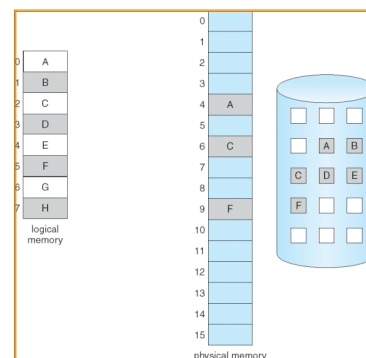
- With each page table entry a valid-invalid bit is associated (1  $\Rightarrow$  in-memory and legal, 0  $\Rightarrow$  not-in-memory or invalid)
- Initially valid-invalid bit is set to 0 on all entries
- Example of a page table snapshot:

Frame #	valid-invalid bit
0	1
1	1
2	1
3	1
4	0
5	0
6	0
7	0
8	0
9	0
10	0
11	0
12	0
13	0
14	0
15	0

page table

- During address translation, if valid-invalid bit in page table entry is 0  $\Rightarrow$  page fault

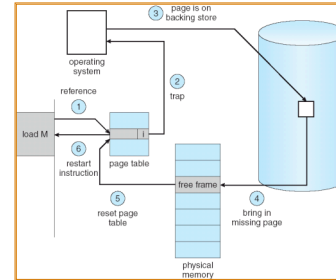
## Page Table When Some Pages Are Not in Main Memory



## Page Fault

- If there is ever a reference to a page, first reference will trap to OS  $\Rightarrow$  page fault
- OS looks at another table to decide:
  - Invalid reference  $\Rightarrow$  abort.
  - Just not in memory.
- Get empty frame.
- Swap page into frame.
- Reset tables, validation bit = 1.
- Restart instruction: Least Recently Used
  - block move
  - auto increment/decrement location

## Steps in Handling a Page Fault



## What happens if there is no free frame?

- Page replacement - find some page in memory, but not really in use, swap it out
  - Algorithms (FIFO, LRU ..)
  - performance - want an algorithm which will result in minimum number of page faults
- Same page may be brought into memory several times

## 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)

$$\begin{aligned} \text{EAT} = & (1 - p) \times \text{memory access} \\ & + p \times (\text{page fault overhead} \\ & \quad + [\text{swap page out}] \\ & \quad + \text{swap page in} \\ & \quad + \text{restart overhead}) \end{aligned}$$

## Demand Paging Example

- Memory access time = 1 microsecond
- 50% of the time the page that is being replaced has been modified and therefore needs to be swapped out
- Swap Page Time = 10 msec = 10,000 microsec
- $\text{EAT} = (1 - p) \times 1 + p \times (10,000 + 1/2 \times 10,000)$   
 $= 1 + 14,999 \times p$  (in microsec)
- What if 1 out of 1000 memory accesses cause a page fault?
- What if we only want 30% performance degradation?

## 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 (zero-fill-on-demand pages)

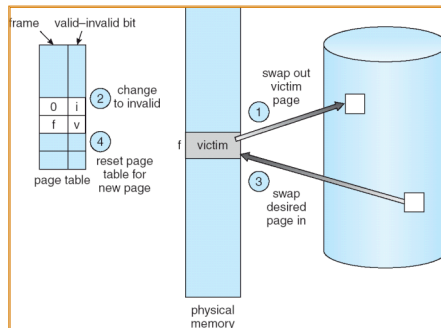
## Page Replacement

- 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

- Find the location of the desired page on disk
- Find a free frame:
  - If there is a free frame, use it
  - If there is no free frame, use a page replacement algorithm to select a **victim** frame
- Read the desired page into the (newly) free frame. Update the page and frame tables.
- Restart the process

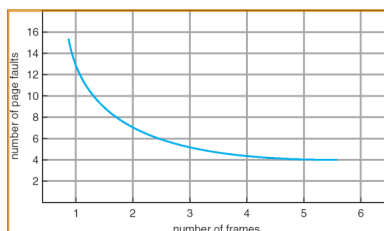
## Page Replacement



## Page Replacement Algorithms

- Want lowest page-fault rate
- Evaluate algorithm by running it on a particular string of memory references (reference string) and computing 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

Graph of Page Faults Versus The Number of Frames



## 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
3	3	2	4

9 page faults

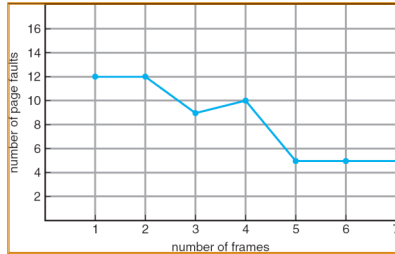
- 4 frames

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

10 page faults

- FIFO Replacement - Belady's Anomaly
  - more frames  $\Rightarrow$  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

1	4
2	
3	
4	5

6 page faults

- 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

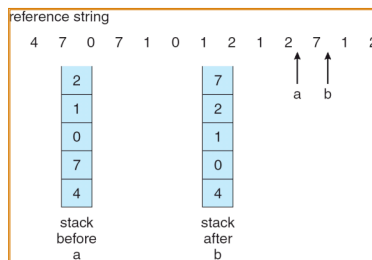
1	5
2	
3	5 4
4	3

- Needs hardware assistance
- 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

## 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

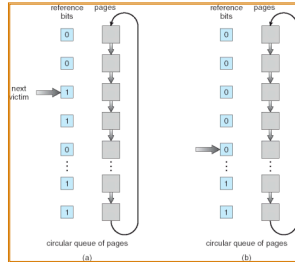
## Use Of A Stack to Record The Most Recent Page References



## 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.
- Additional Reference bits
  - 1 byte for each page: eg. 00110011
  - Shift right at each time interval
- 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

### Allocation of Frames

- Each process needs *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*
- Two major allocation schemes
  - fixed allocation
  - priority allocation

### 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 process

$$\begin{aligned}
 s_i &= \text{size of process } p_i & m &= 64 \\
 S &= \sum s_i & s_i &= 10 \\
 m &= \text{total number of frames} & s_2 &= 127 \\
 a_i &= \text{allocation for } p_i = \frac{s_i}{S} \times m & a_1 &= \frac{10}{137} \times 64 \approx 5 \\
 & & a_2 &= \frac{127}{137} \times 64 \approx 59
 \end{aligned}$$

### 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 frames
  - select for replacement a frame from a process with lower priority number

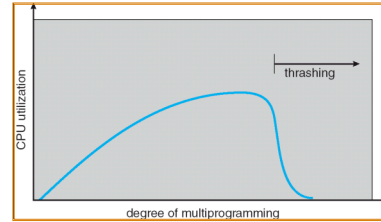
### Global vs. Local Allocation

- **Global replacement** - process selects a replacement frame from the set of all frames; one process can take a frame from another
- **Local replacement** - each process selects from only its own set of allocated frames

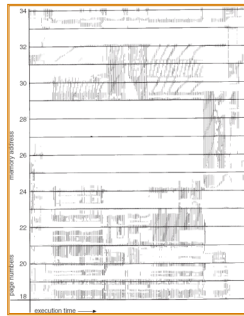
## Thrashing

- If a process does not have “enough” frames, the page-fault rate is very high. This leads to:
  - Replacement of active pages which will be needed soon again
  - **Thrashing** = a process is busy swapping pages in and out
- Which will in turn cause:
  - low CPU utilization
  - operating system thinks that it needs to increase the degree of multiprogramming
  - another process added to the system

## Thrashing (Cont.)



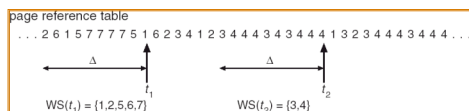
## Locality In A Memory-Reference Pattern



## Working-Set Model

- $\Delta$  = working-set window = a fixed number of page references  
Example: 10,000 instruction
- $WSS_i$  (working set of Process  $P_i$ ) = total number of pages referenced in the most recent  $\Delta$  (varies in time)
  - if  $\Delta$  too small will not encompass entire locality
  - if  $\Delta$  too large will encompass several localities
  - if  $\Delta = \infty \Rightarrow$  will encompass entire program
- $D = \sum WSS_i$  = total demand frames
- if  $D > m \Rightarrow$  Thrashing
- Policy if  $D > m$ , then suspend one of the processes

## Working-set model

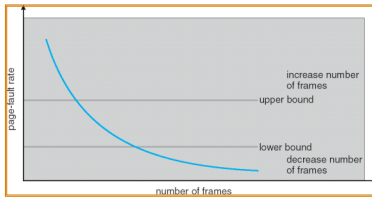


## Keeping Track of the Working Set

- Approximate with interval timer + a reference bit
- Example:  $\Delta = 10,000$ 
  - Timer interrupts after every 5000 time units
  - Keep in memory 2 bits for each page
  - Whenever a timer interrupts copy and sets the values of all reference bits to 0
  - If one of the bits in memory = 1  $\Rightarrow$  page in working set
- Why is this not completely accurate?
- Improvement = 10 bits and interrupt every 1000 time units

### 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



### Any Questions?



38

### Reading Assignment

- Read chapter 9 from Silberschatz.

39

### Acknowledgements

- “Operating Systems Concepts” book and supplementary material by Silberschatz, Galvin and Gagne.

40