Page Replacement Algorithms

Demand Paging

- Load pages in memory only on page fault
  - Find location of desired page on disk
  - Find a free frame
    - if one is available, use it
    - otherwise, use a page replacement algorithm
      - select a victim among active mappings
      - free corresponding frame, and load new page in it
      - local vs. global victim selection
  - Restart!

How do we pick a victim?

- We want:
  - low page fault-rate
  - page faults as inexpensive as possible
- We need:
  - a way to compare the relative performance of different page replacement algorithms
  - some absolute notion of what a "good" page replacement algorithm should accomplish

Comparing Page Replacement Algorithms

- Record a trace of the pages accessed by a process
  - E.g. 3,1,4,2,5,2,1,2,3,4 or c,a,d,b,e,b,a,b,c,b
- Simulate behavior of page replacement algorithm on the trace and record the number of page faults generated

Optimal Page Replacement

- Replace page needed furthest in future

Time	0	1	2	3	4	5	6	7	8	9	10
Requests		c	a	d	b	e	b	a	b	c	d
Page Frames 0	a	a	a	a	a	a	a	a	a	a	d
Page Frames 1	b	b	b	b	b	b	b	b	b	b	b
Page Frames 2	c	c	c	c	c	c	c	c	c	c	c
Page Frames 3	d	d	d	d	d	e	e	e	e	e	e
Faults						X					X
Time page needed next					a = 7 b = 6 c = 9 d = 10					b = 11 c = 13 d = 12 e = 15	

b d c b e

FIFO Replacement

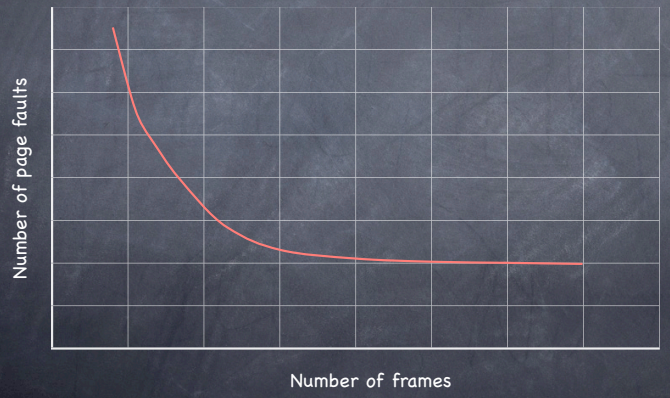
- Replace pages in the order they come into memory

Assume:

a @ -3
b @ -2
c @ -1
d @ 0

Time	0	1	2	3	4	5	6	7	8	9	10
Requests		c	a	d	b	e	b	a	b	c	d
Page Frames 0	a	a	a	a	a	e	e	e	e	e	d
Page Frames 1	b	b	b	b	b	b	b	a	a	a	a
Page Frames 2	c	c	c	c	c	c	c	c	b	b	b
Page Frames 3	d	d	d	d	d	d	e	e	e	c	c
Faults						X		X	X	X	X

+ Frames
- Page Faults



For example...

Time	0	1	2	3	4	5	6	7	8	9	10	11	12
Request		a	b	c	d	a	b	e	a	b	c	d	e
Page Frames 0		a	a	a	d	d	d	e	e	e	e	e	e
Page Frames 1			b	b	b	a	a	a	a	a	c	c	c
Page Frames 2				c	c	c	b	b	b	b	b	d	d
Faults		X	X	X	X	X	X	X			X	X	

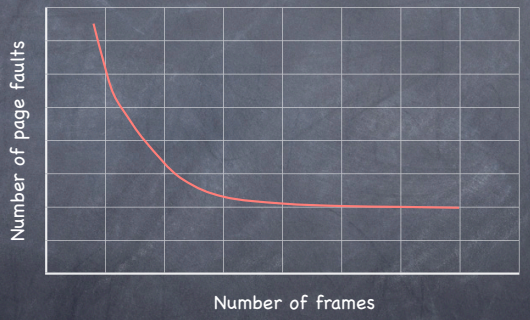
- 3 frames - 9 page faults!

Belady's Anomaly

Time		0	1	2	3	4	5	6	7	8	9	10	11	12
Request			a	b	c	d	a	b	e	a	b	c	d	e
Page Frames	0		a	a	a	a	a	a	e	e	e	e	d	d
	1			b	b	b	b	b	a	a	a	a	a	e
	2				c	c	c	c	c	b	b	b	b	b
	3					d	d	d	d	d	d	c	c	c
Faults			X	X	X	X			X	X	X	X	X	X

- 4 frames - 10 page faults!

+ Frames
- Page Faults?



- Yes, but only for stack page replacement policies, that assign a priority to a page that is independent of the number of page frames

LRU: Least Recently Used

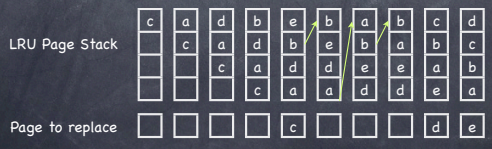
- Replace page not referenced for the longest time

Time		0	1	2	3	4	5	6	7	8	9	10
Requests			c	a	d	b	e	b	a	b	c	d
Page Frames	0	a	a	a	a	a	a	a	a	a	a	a
	1	b	b	b	b	b	b	b	b	b	b	b
	2	c	c	c	c	c	e	e	e	e	e	d
	3	d	d	d	d	d	d	d	d	d	c	c
Faults							X				X	X
Time page last used						a = 2 b = 4 c = 1 d = 3				a = 7 b = 8 e = 5 d = 3	a = 7 b = 8 e = 5 c = 9	

Implementing LRU

- Maintain a "stack" of recently used pages

Time		0	1	2	3	4	5	6	7	8	9	10
Requests			c	a	d	b	e	b	a	b	c	d
Page Frames	0	a	a	a	a	a	a	a	a	a	a	a
	1	b	b	b	b	b	b	b	b	b	b	b
	2	c	c	c	c	c	e	e	e	e	e	d
	3	d	d	d	d	d	d	d	d	d	c	c
Faults							X				X	X



Implementing LRU

- Add a (64-bit) timestamp to each page table entry
  - HW counter incremented on each instruction
  - Page table entry timestamped with counter when referenced
  - Replace page with lowest timestamp

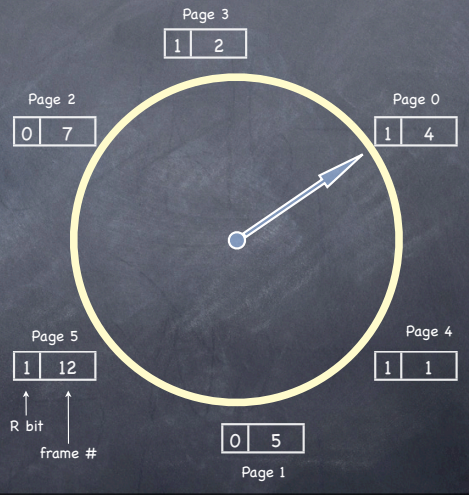
Implementing LRU

- Add a (64-bit) timestamp to each page table entry
  - HW counter incremented on each instruction
  - Page table entry timestamped with counter when referenced
  - Replace page with lowest timestamp
- Approximate LRU through aging
  - keep a k-bit tag in each table entry
  - on every clock tick, shift right one bit
  - copy R bit in most significant bit
  - replace page with lowest tag

	R bits at Tick 0	R bits at Tick 1	R bits at Tick 2	R bits at Tick 4	R bits at Tick 5
	1 0 1 0 1 1	1 1 0 0 1 0	1 1 0 1 0 1	1 0 0 0 1 0	0 1 1 0 0 0
Page 0	10000000	11000000	11100000	11110000	01111000
Page 1	00000000	10000000	11000000	01100000	10110000
Page 2	10000000	01000000	00100000	00100000	10001000
Page 3	00000000	00000000	10000000	01000000	00100000
Page 4	10000000	11000000	01100000	10110000	01011000
Page 5	10000000	01000000	10100000	01010000	00101000

The Clock Algorithm

- Organize pages in memory as a circular list
- When page is referenced, set its reference bit R to 1
- On page fault
  - if R = 0: evict the page
  - if R = 1: clear r
  - advance hand

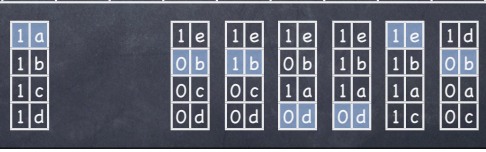


Clock Page Replacement

Time		0	1	2	3	4	5	6	7	8	9	10
Requests			c	a	d	b	e	b	a	b	c	d
Page Frames	0	a	a	a	a	a	e	e	e	e	e	d
	1	b	b	b	b	b	b	b	b	b	b	b
	2	c	c	c	c	c	c	c	a	c	a	a
	3	d	d	d	d	d	d	d	d	d	c	c
Faults							X		X		X	X

Page table entries for resident pages

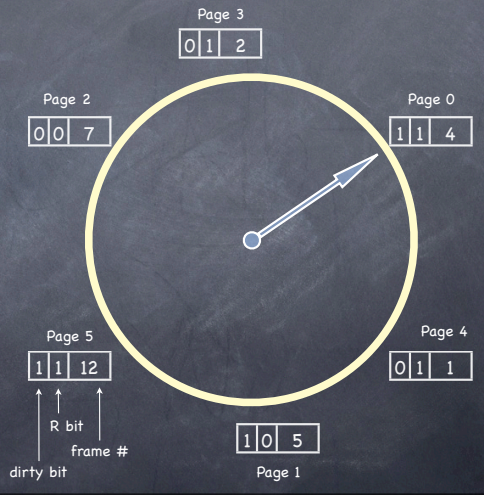
Hand clock:



The Second Chance Algorithm

- Dirty pages get "second chance" before eviction
  - replacing dirty pages is expensive!

Before Clock sweep		After Clock sweep	
dirty	R	dirty	R
0	0	replace page	
0	1	0	0
1	0	0	0
1	1	1	0



Clock Page Replacement

Time	0	1	2	3	4	5	6	7	8	9	10
Requests		c	a^w	d	b^w	e	b	a^w	b	c	d
Page Frames 0	a	a	a	a	a	a	a	a	a	a	a
1	b	b	b	b	b	b	b	b	b	b	d
2	c	c	c	c	c	e	e	e	e	e	e
3	d	d	d	d	d	d	d	d	d	c	c
Faults						X				X	X

Page table entries for resident pages

01 a	11 a	00 a^*	00 a^*	11 a	11 a	00 a^*
01 b	11 b	00 b^*	01 b^*	01 b^*	01 b^*	01 d
01 c	01 c	01 e	01 e	01 e	01 e	00 e
01 d	01 d	00 d	00 d	00 d	01 c	00 c

Hand clock: ■

Local vs. Global Page Replacement

- Local: Select victim only among allocated frames
  - Equal or proportional frame allocation
- Global: Select any free frame, even if allocated to another process
  - No control over page fault rate

Brother, can you spare a frame?

Time	0	1	2	3	4	5	6	7	8	9	10	11	12
Requests		a	b	c	d	a	b	c	d	a	b	c	d
0	a	a	a	a	d	d	d	c	c	c	b	b	b
1	b	b	b	b	b	a	a	a	d	d	d	c	c
2	c	c	c	c	c	c	b	b	b	a	a	a	d
Faults					X	X	X	X	X	X	X	X	X

Brother, can you spare a frame?

Time	0	1	2	3	4	5	6	7	8	9	10	11	12
Requests		a	b	c	d	a	b	c	d	a	b	c	d
0	a	a	a	a	a	a	a	a	a	a	a	a	a
1	b	b	b	b	b	b	b	b	b	b	b	b	b
2	c	c	c	c	c	c	c	c	c	c	c	c	c
3	-	-	-	-	d	d	d	d	d	d	d	d	d
Faults					X								

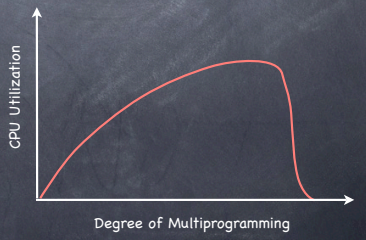
So, what's wrong with global replacement?

A Vicious Cycle

- When not enough frames...
  - high page fault rate
  - low CPU utilization
  - OS may increase degree of multiprogramming!

A Vicious Cycle

- When not enough frames...
  - high page fault rate
  - low CPU utilization
  - OS may increase degree of multiprogramming!
- Thrashing
  - process spends all its time swapping pages in and out



The Principle of Locality

- 90% of the execution of a program is sequential
- Most iterative constructs consist of a relatively small number of instructions
- When processing large data structures the dominant cost is sequential processing on individual structure elements
- Locality can be both spatial and temporal

The Working Set Model

- Choose Δ page references as WS window
- WSS_i = # of pages referenced by p_i in latest Δ
  - Δ too small: does not cover locality
  - Δ too large: covers many localities
- Thrashing if $\Sigma_i WSS_i$ > # frames
  - if so, suspend one of the processes
    - which one?

WS Page Replacement
Δ = 4

Time		0	1	2	3	4	5	6	7	8	9	10
Requests			c	c	d	b	c	e	c	e	a	d
Pages in Memory	Page a	$t=0$ ●										
	Page b											
	Page c											
	Page d	$t=-1$ ●										
	Page e	$t=-2$ ●										
Faults												

WS Page Replacement
Δ = 4

Time		0	1	2	3	4	5	6	7	8	9	10
Requests			c	c	d	b	c	e	c	e	a	d
Pages in Memory	Page a	$t=0$ ●	●	●	●						●	●
	Page b					●	●	●	●			
	Page c		●	●	●	●	●	●	●	●	●	●
	Page d	$t=-1$ ●	●	●	●	●	●	●				●
	Page e	$t=-2$ ●	●					●	●	●	●	●
Faults			X			X		X			X	X

Computing the WS

- Use interval timer τ, the R bit, and k extra bits per page
- $\Delta = \tau \times k$
- When τ elapses, shift k bits right, copy R bit in MSB and reset R bit
- If one of the k bits is 1, the corresponding page is in WS
- Sensitivity?

Tracking Page Fault Frequency

- When too high, increase WS; decrease when too low.

Keep time t_{last} of last page fault

On page fault:

if $t_{\text{current}} - t_{\text{last}} > \tau^*$, then unmap all pages not referenced in $[t_{\text{last}} - t_{\text{current}}]$

else add faulting page to the working set

PFF Page Replacement $\tau^* = 2$

Time	0	1	2	3	4	5	6	7	8	9	10
Requests		c	c	d	b	c	e	c	e	a	d
Pages in Memory: Page a	•										
Page b											
Page c											
Page d	•										
Page e	•										
Faults											
$t_{\text{current}} - t_{\text{last}}$											

PFF Page Replacement $\tau^* = 2$

Time	0	1	2	3	4	5	6	7	8	9	10
Requests		c	c	d	b	c	e	c	e	a	d
Pages in Memory: Page a	•	•	•	•						•	•
Page b					•	•	•	•	•		
Page c		•	•	•	•	•	•	•	•	•	•
Page d	•	•	•	•	•	•	•	•	•		•
Page e	•	•	•	•			•	•	•	•	•
Faults		X			X		X			X	X
$t_{\text{current}} - t_{\text{last}}$		1			3		2			3	1