CSCE 451/851
Operating Systems Principles

Page Replacement Algorithms

Steve Goddard
goddard@cse.unl.edu

http://www.cse.unl.edu/~goddard/Courses/CSCE451

Virtual Memory Management
Fundamental issues

- Placement strategy
- Replacement strategies
- Load control strategies

Example: *Demand paging*

» No placement strategy required *per se*

» Load control: load pages only when faults occur

» Replacement: ...

Page Replacement Algorithms
Concept

- Typically $\Sigma_i |VAS_i|$ >> *Physical Memory*
- With demand paging, physical memory fills quickly
- So when a process faults & memory is full, some page must be swapped out

  (So handling a page fault now requires ***2*** disk accesses not 1!)
- Which page should be replaced?
  - » *Local replacement* — replace a page of the faulting process
  - » *Global replacement* — possibly replace the page of another process

3

Page Replacement Algorithms
Evaluation methodology

- Record a *trace* of pages accessed by a process
  - » Example: (Virtual) address trace...

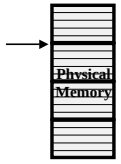
    (3,0), (1,9), (4,1), (2,1), (5,3), (2,0), (1,9), (2,4), (3,1), (4,8)
  - » ... generates page trace

    3, 1, 4, 2, 5, 2, 1, 2, 3, 4 (represented as *c*, *a*, *d*, *b*, *e*, *b*, *a*, *b*, *c*, *d*)
- Simulate the behavior of a page replacement algorithm on the trace & record the number of page faults generated
  - » *fewer faults == better performance*

4

Local Page Replacement
FIFO replacement

- Simple to implement
  - » A single pointer suffices



- Performance with 4 page frames:

Page Trace

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*	*d*	*d*	*d*	***c***	*c*
Faults						•		•	•	•	•

5

Optimal Page Replacement
Clairvoyant replacement

- Replace the page that won't be needed for the longest time in the 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						•					•

Time page needed next		
	a = 7	*a* = 15
	b = 6	*b* = 11
	c = 9	*c* = 13
	d = 10	*d* = 14

6

Least Recently Used Replacement
"Back to the future"

- Replace the page that hasn't been referenced for the longest time
  - » Use the recent past as a predictor of the 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*	*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						•				•	•
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	

7

Least Recently Used Replacement
Implementation

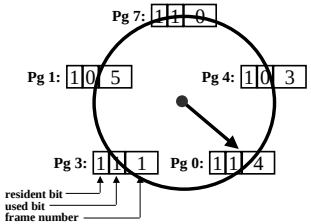
- Maintain a "stack" of recently used pages

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						•				•	•
LRU Page Stack						*c*				*d*	*e*
					c	*a*	*a*	*d*	*d*	*e*	*a*
				c	*a*	*d*	*d*	*e*	*e*	*a*	*b*
			c	*a*	*d*	*b*	*e*	*b*	*a*	*b*	*c*
		c	*a*	*d*	*b*	*e*	*b*	*a*	*b*	*c*	*d*

8

Approximate *LRU* Replacement
The *Clock* algorithm

- Maintain a circular list of pages resident in memory
  - » Use a *clock* (or *used*) bit to track how often a page is accessed — bit set whenever a page is referenced
- Clock sweeps over pages looking for one with used bit = 0
  - » Replaces pages that haven't been referenced for one complete revolution of the clock




```
func Clock_Replacement
begin
  while (page not found for replacement) do
    if(used bit for current page = 0) then
      replace current page
    else
      reset used bit
    end if
    advance clock pointer
  end while
end Clock_Replacement
```


Clock Page Replacement
Example

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***	*a*	*a*	*a*
3	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	***c***	*c*
Faults						•		•		•	•

Page table entries for resident pages:

1 *a*	1 *e*	1 *e*	1 *e*	1 *e*	1 *e*	1 *d*
1 *b*	0 *b*	1 *b*	1 *b*	1 *b*	1 *b*	0 *b*
1 *c*	0 *c*	0 *c*	1 *a*	1 *a*	1 *a*	0 *a*
1 *d*	0 *d*	0 *d*	0 *d*	0 *d*	1 *c*	0 *c*

The LRU page

Performance of Page Replacement Algorithms

- Least recently used
  - » Ages pages based on when they were last used
- FIFO
  - » Ages pages based on when they're brought into memory

The principle of locality

- 90% of the execution of a program is sequential
- Most interative constructs consist of a relatively small number of instructions
- When processing large data structures, the dominant cost is sequential processing on individual structure elements

11

Belady's Anomaly

More memory ≠ Better performance!

FIFO Replacement

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

12

Explicitly Using Locality
The working set model of page replacement

- Assume recently referenced pages are likely to be referenced again soon...
- ... and only keep those pages recently referenced in memory (called *the working set*)
  - » pages may be removed even when no page fault occurs
  - » number of frames allocated to a process will vary over time
- A process is allowed to execute only if its working set fits into memory
  - » implicit load control

13

Working Set Page Replacement
Implementation

- Keep track of the last τ references
  - » The pages referenced during the last τ memory accesses are the working set
  - » τ is called the *window size*
- Example: working set computation, $\tau = 4$ references:

	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		•			•		•			•	•

14

Optimal Page Replacement
For processes with a variable number of frames

- *VMIN* — Replace a page that is not referenced in the *next* τ accesses
- Example: $\tau = 4$

	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*	-	-	-	-	-	-	•	•	•	-	-
	Faults		•			•		•			•	•

15

Page-Fault-Frequency Page Replacement
An alternate working set computation

- Explicitly attempt to minimize page faults
  - » When page fault frequency is high — *increase working set*
  - » When page fault frequency is low — *decrease working set*
- Algorithm:
  - » Keep track of the rate at which faults occur
    - ❖ Record the time, t_{last}, of the last page fault
  - » If the time between page faults is "large" then reduce working set
    - ❖ If $t_{current} - t_{last} > \tau$, then remove from memory all pages not referenced in $[t_{last}, t_{current}]$
  - » If the time between page faults is "small" then increase working set
    - ❖ If $t_{current} - t_{last} \leq \tau$, then add faulting page to the working set

16

Page-Fault-Frequency Page Replacement
Example, window size = 2

- If $t_{current} - t_{last} > 2$, remove pages not referenced in $[t_{last}, t_{current}]$ from the working set
- If $t_{current} - t_{last} \leq 2$, add faulting page to the working set

	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		•			•		•			•	•
	$t_{cur} - t_{last}$					3		2			3	1

17