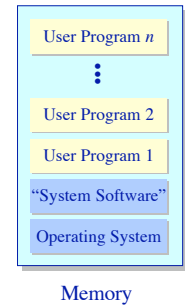


Page Replacement Algorithms

1

Virtual Memory Management Fundamental issues : A Recap

- ◆ Key concept: Demand paging
 - Load pages into memory only when a page fault occurs
- ◆ Issues:
 - Placement strategies
 - ◆ Place pages anywhere - no placement policy required
 - Replacement strategies
 - ◆ What to do when there exist more jobs than can fit in memory
 - Load control strategies
 - ◆ Determining how many jobs can be in memory at one time



2

Page Replacement Algorithms Concept

- ◆ Typically $\Sigma_i VAS_i \gg \text{Physical Memory}$
 - ◆ With demand paging, physical memory fills quickly
 - ◆ When a process faults & memory is full, some page must be swapped out
 - 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 the 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 and record the number of page faults generated
fewer faults $\Rightarrow$ better performance

4

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									
1	b										
2	c										
3	d										
Faults											
Time page needed next											



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	d
1	b	b	b	b	b	b	b	b	b	b	b
2	c	c	c	c	c	c	c	c	c	c	c
3	d	d	d	d	d	e	e	e	e	e	e
Faults											
Time page needed next											

a = 7
b = 6
c = 9
d = 10

a = 15
b = 11
c = 13
d = 14

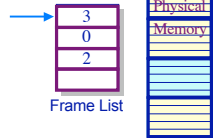
6

Local Page Replacement FIFO replacement

- Simple to implement
 - A single pointer suffices

- Performance with 4 page frames:

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									
1	b										
2	c										
3	d										
Faults											



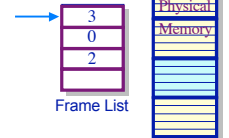
7

Local Page Replacement FIFO replacement

- Simple to implement
 - A single pointer suffices

- Performance with 4 page frames:

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	e	e	e	e	e	d
1	b	b	b	b	b	b	a	a	a	a	a
2	c	c	c	c	c	c	c	b	b	b	b
3	d	d	d	d	d	d	d	d	d	c	c
Faults											



8

Least Recently Used Page Replacement

Use the recent past as a predictor of the near future

- Replace the page that hasn't been 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									
1	b										
2	c										
3	d										
Faults											
Time page last used											



9

Least Recently Used Page Replacement

Use the recent past as a predictor of the near future

- Replace the page that hasn't been 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
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											

10

Least Recently Used Page 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
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											
Page to replace											



11

Least Recently Used Page 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
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	a	d	b	e	b	a	b	c	d	
Page to replace											

12

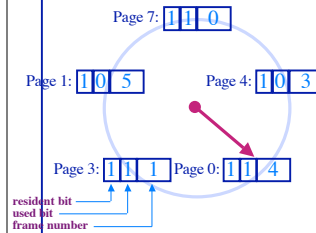
Least Recently Used Page Replacement Alternate Implementation --- Aging Register

- ◆ Maintain an n-bit aging register $R = R_{n-1}R_{n-2}...R_0$ for each page frame
 - On a page reference, set R_{n-1} to 1
 - Every T units of time, shift the aging vector right by one bit
- ◆ Key idea:
 - Aging vector can be interpreted as a positive binary number
 - Value of R decreases periodically unless the page is referenced
- ◆ Page replacement algorithm:
 - On a page fault, replace the page with the smallest value of R

13

Approximate LRU Page Replacement The Clock algorithm

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



```

func Clock_Replacement
begin
  while (victim page not found) 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

```

14

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						
	1	b	b	b	b						
	2	c	c	c	c						
	3	d	d	d	d						
Faults											
Page table entries for resident pages: 1 a 1 b 1 c 1 d											

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	e	e	e	e	e	d
	1	b	b	b	b	b	b	b	b	b	b
	2	c	c	c	c	c	c	a	a	a	a
	3	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	d
	1	b	0	b	1	b	0	b	1	b	0	b
	1	c	0	c	0	c	1	a	1	a	0	a
	1	d	0	d	0	d	0	d	0	d	1	c

- ◆ There is a significant cost to replacing "dirty" pages
- ◆ Modify the Clock algorithm to allow dirty pages to always survive one sweep of the clock hand
 - Use both the *dirty bit* and the *used bit* to drive replacement

[illegible]

age table
entries
resident
pages:

10	<i>a</i>
10	<i>b</i>
10	<i>c</i>
10	<i>d</i>

11	<i>a</i>	00	<i>a</i> [*]	00	<i>a</i> [*]	11	<i>a</i>
11	<i>b</i>	00	<i>b</i> [*]	10	<i>b</i> [*]	10	<i>b</i> [*]
10	<i>c</i>	10	<i>e</i>	10	<i>e</i>	10	<i>e</i>
10	<i>d</i>	00	<i>d</i>	00	<i>d</i>	00	<i>d</i>

11	<i>a</i>	00	<i>a</i> [*]
10	<i>b</i> [*]	10	<i>d</i>
10	<i>e</i>	00	<i>e</i>
10	<i>c</i>	00	<i>c</i>

Time	0	1	2	3	4	5	6	7	8	9	10	11	12
Requests	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	
Page Frames	0	<i>a</i>											
	1	<i>b</i>											
	2	<i>c</i>											
Faults													

The Problem With Local Page Replacement

How much memory do we allocate to a process?

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	
Page Frames	0	a	a	a	a	d	d	d	c	c	c	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													

Page Frames	0	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													

21

Page Replacement Algorithms

Performance

- Local page replacement
 - LRU — Ages pages based on when they were last used
 - FIFO — Ages pages based on when they're brought into memory
- Towards global page replacement ... with variable number of page frames allocated to processes

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

22

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										
Page b											
Page c											
Page d											
Page e											
Faults											

23

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										
Page b											
Page c											
Page d											
Page e											
Faults											

24

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*)
 - Thus pages may be removed even when no page fault occurs
 - The 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
 - The working set model performs implicit load control

25

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	$\tau=0$									
	Page b	-									
	Page c	-									
	Page d	$\tau=1$									
	Page e	$\tau=2$									
Faults											

26

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	$\tau=0$								F	
	Page b	-				F					
	Page c		F								
	Page d	$\tau=1$									F
	Page e	$\tau=2$					F				
Faults											

27