Lecture 30 - Page Faults and Page Replacement

CprE 308

March 30, 2015

Page Faults

Page Fault

What happens if the required page is not in memory?

"Page-fault" trap is initiated, OS gets control

1. Find a free page frame
2. Read the desired page from disk into memory
3. Modify the page tables
4. Restart the interrupted instruction

OS Issues

- Fetch policy - when to fetch pages into memory?
- Placement policy - where to place pages?
- Replacement policy
- All combined in the handling of a page fault

A Simple Paging Scheme

Fetch policy

- start process off with no pages in primary storage
- bring in pages on demand (and only on demand)
  - this is known as demand paging

Placement policy

- it doesn't matter - put the incoming page in the first available page frame

Replacement policy

- replace the page that has been in primary storage the longest (FIFO policy)

Improving the Fetch Policy

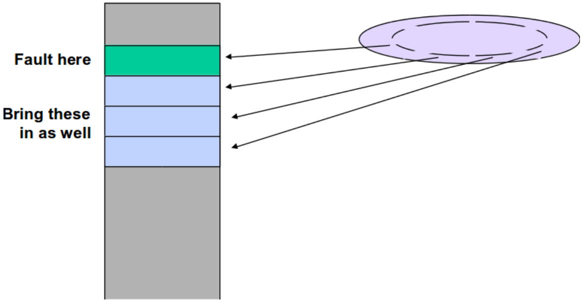

Figure 1: Fetch Policy

Page Replacement

Improving the Replacement Policy

- When is replacement done?
  - doing it "on demand" causes excessive delays
  - should be performed as a separate, concurrent activity
- Which pages are replaced?
  - FIFO policy is not good
  - want to replace those pages least likely to be referenced soon

The “Pageout Daemon”

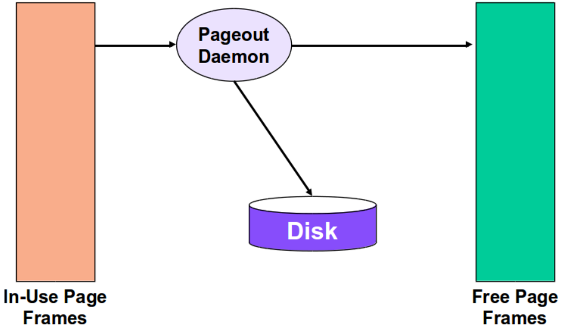

Figure 2:

Page Replacement

Problem Statement:

> *A page is being brought into memory which has no free space. Which page should we replace to make space?*

Page Replacement - Mental Exercise

- What is the optimal policy, if we had the knowledge of the future page requests

Page Replacement - Mental Exercise

- What is the optimal policy, if we had the knowledge of the future page requests
- Policy: Choose the page which will be referenced farthest in the future

Page Replacement - Mental Exercise

- What is the optimal policy, if we had the knowledge of the future page requests
- Policy: Choose the page which will be referenced farthest in the future
- However, we don't know the future

Page Replacement - Mental Exercise

- What is the optimal policy, if we had the knowledge of the future page requests
- Policy: Choose the page which will be referenced farthest in the future
- However, we don't know the future
    - Hope that the next few references will be for pages that were recently referenced

Page Replacement - Mental Exercise

- What is the optimal policy, if we had the knowledge of the future page requests
- Policy: Choose the page which will be referenced farthest in the future
- However, we don't know the future
  - Hope that the next few references will be for pages that were recently referenced
- What's the use of knowing about this policy?

Page Replacement - Mental Exercise

- What is the optimal policy, if we had the knowledge of the future page requests
- Policy: Choose the page which will be referenced farthest in the future
- However, we don't know the future
    - Hope that the next few references will be for pages that were recently referenced
- What's the use of knowing about this policy?
    - Will help us access the performance of a real algorithm

Choosing the Page to Remove

Policies:

- FIFO (First-In-First-Out)
- NRU (Not-Recently-Used)
- Second Chance
- LRU (Least-Recently-Used)
- Clock Algorithm(s)
- Working Set Algorithm

Two issues:

- How good is the decision?
- Overhead?
  - Cost per memory access - should be very small
  - Cost per replacement - can be larger

FIFO

Example: 8 pages, 4 page frames

1	0	2	2	1	7	6	7	0	1	2	0	3	0	4	5	1	5	2	4	5	6	7	6	7	2	4	2	7	3	3	2	3
1	1	1	1	1	1	6	6	6	6	6	6	6	6	4	4	4	4	4	4	4	6	6	6	6	6	6	6	6	6	6	2	2
-	0	0	0	0	0	0	0	0	1	1	1	1	1	1	5	5	5	5	5	5	5	7	7	7	7	7	7	7	7	7	7	7
-	-	2	2	2	2	2	2	2	2	2	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	4	4	4	4	4	4	4
-	-	-	-	-	7	7	7	7	7	7	7	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	3	3	3	3
F	F	F			F	F			F		F	F		F	F	F		F			F	F				F			F		F	

Figure 3:

Hit ratio: 16/33

Help from Hardware

For each page frame:

- Referenced Bit(R) - 1 if page frame has been referenced recently
- Modified Bit(M) - 1 if page has been modified since it has been loaded
    - Also known as “dirty bit”

Not Recently Used Algorithm (NRU)

Pages are classified into 4 classes:

- Class 0: not referenced, not modified (R=0, M=0)
- Class 1: not referenced, modified (R=0, M=1)
- Class 2: referenced, not modified (R=1, M=0)
- Class 3: referenced, modified (R=1, M=1)

NRU removes page at random from lowest number non empty class

The R bit is cleaned periodically (based on a timer)

Second Chance

- Based on FIFO
- Old pages are inspected for replacement
  - But are given a “second chance” if they have been used recently

Second Chance Algorithm

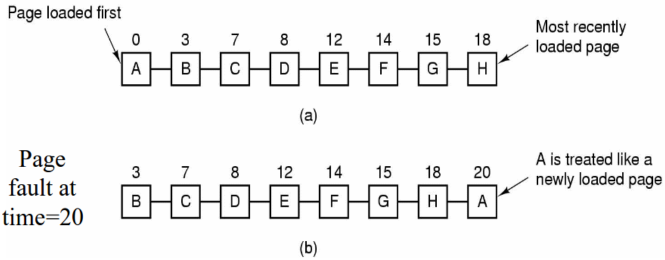

Figure 4:

- Pages sorted in FIFO order (time of arrival)
- If earliest page has R=1, then give it a second chance by moving it to the end of the list

Clock Algorithm - Another Implementation of Second Chance

- Order pages in circular list
- “Hand” of the clock points to the page to be replaced currently
- When required to evict a page
  - If page pointed to has R=0, then evict it
  - If R=1, then reset R and move hand forward
- Clock algorithm can be used with NRU (decision based on both R and M bits)

Least Recent Used (LRU)

- Replace the page in memory which has been unused for the longest time
- Locality of Reference: pages used in the near past will be used in the near future
  - True in typical cases

Least Recently Used (LRU)

Example: 8 pages, 4 page frames

1 0 2 2 1 7 6 7 0 1 2 0 3 0 4 5 1 5 2 4 5 6 7 6 7 2 4 2 7 3 3 2 3

1 1 1 1 1 1 1 1 1 1 1 1 1 1 4 4 4 4 4 4 4 4 4 4 4 2 2 2 2 2 2 2 2

- 0 0 0 0 0 6 6 6 6 2 2 2 2 2 5 5 5 5 5 5 5 5 5 5 4 4 4 4 4 4 4

- - 2 2 2 2 2 2 0 0 0 0 0 0 0 0 0 0 2 2 2 2 7 7 7 7 7 7 7 7 7 7 7

- - - - - 7 7 7 7 7 7 7 3 3 3 3 1 1 1 1 1 6 6 6 6 6 6 6 6 3 3 3 3

Figure 5:

LRU Implementation

- Think of how you would implement it

LRU Implementation

- Think of how you would implement it
- One possible implementation:

LRU Implementation

- Think of how you would implement it
- One possible implementation:
  - list of pages, most recently used at front, least at rear

LRU Implementation

- Think of how you would implement it
- One possible implementation:
  - list of pages, most recently used at front, least at rear
  - update this list every memory reference

LRU Implementation

- Think of how you would implement it
- One possible implementation:
  - list of pages, most recently used at front, least at rear
  - update this list every memory reference
  - when required to evict a page, choose the one at the rear of the list

LRU Implementation

- Think of how you would implement it
- One possible implementation:
  - list of pages, most recently used at front, least at rear
  - update this list every memory reference
  - when required to evict a page, choose the one at the rear of the list
- Way too expensive!

Not Frequently Used (NFU)

- Requires a software counter associated with each page, initially zero
- At each clock interrupt, OS scans all the pages in memory
- For each page, the R bit is added to the counter
- The page with the lowest counter is chosen

Aging - Approximating LRU

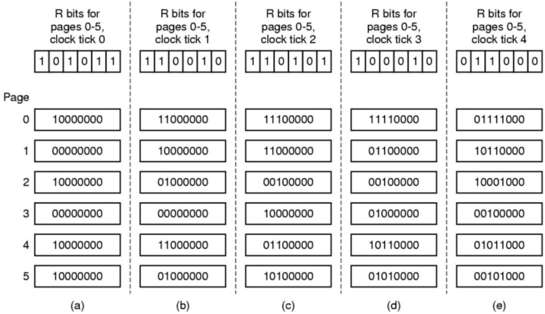

Figure 6:

Example

	Page frame	Time loaded	Time referenced	R bit	M bit
	0	60	161	0	1
	1	130	160	0	0
	2	26	162	1	0
	3	20	163	1	1

Figure 7:

Questions:

Which page frame will be replaced?

- FIFO

Questions:

Which page frame will be replaced?

- FIFO
  - PFN 3 since loaded longest ago at time 20

Questions:

Which page frame will be replaced?

- FIFO
  - PFN 3 since loaded longest ago at time 20
- LRU

Questions:

Which page frame will be replaced?

- FIFO
  - PFN 3 since loaded longest ago at time 20
- LRU
  - PFN 1 since referenced longest ago at time 160

Questions:

Which page frame will be replaced?

- FIFO
  - PFN 3 since loaded longest ago at time 20
- LRU
  - PFN 1 since referenced longest ago at time 160
- Clock

Questions:

Which page frame will be replaced?

- FIFO
  - PFN 3 since loaded longest ago at time 20
- LRU
  - PFN 1 since referenced longest ago at time 160
- Clock
  - Clear R in PFN 3 (oldest loaded), clear R in PFN 2 (next oldest loaded), victim PFN is 0 since R=0