INFO324 Operating Systems II

Ahmad FAOUR

Page Replacement

Page Replacement

- Assume a normal page table
- User-program is executing
- A *PageInvalidFault* occurs!
  - The page needed is not in memory
- Select some frame and remove the page in it
  - If it has been modified, it must be written back to disk
    - the “dirty” bit in its page table entry tells us if this is necessary
- Local allocation or global allocation?
  - Limit the search to the pages loaded for the process causing the default or all loaded pages
- Figure out which page was needed from the faulting addr
- Read the needed page into this frame
- Restart the interrupted process by retrying the same instruction

Terminology

- **Reference string**: the memory reference sequence generated by a program.
- **Paging** – moving pages to (from) disk
- **Optimal** – the best (theoretical) strategy
- **Eviction** – throwing something out
- **Pollution** – bringing in useless pages/lines

Page Replacement Algorithms

Which frame to replace?

Algorithms:

- The Random Algorithm
- The Optimal Algorithm
- First In First Out (FIFO)
- Not Recently Used (NRU)
- Second Chance / Clock
- Least Recently Used (LRU)
- Not Frequently Used (NFU)
- Working Set (WS)
- WSClock

The Random Algorithm

- Choose a page randomly
- Very easy
- Local and global version
- Used for comparison studies

The Optimal Algorithm

Idea:

Select the page that will not be needed for the longest time

Principal of Optimality

- Description:
  - Assume that each page can be labeled with the number of instructions that will be executed before that page is first references, i.e., we would know the future reference string for a program.
  - Then the optimal page algorithm would choose the page with the highest label to be removed from the memory.
- This algorithm provides a basis for comparison with other schemes.
- Impractical because it needs future references
- If future references are known
  - should not use demand paging
  - should use pre paging to allow paging to be overlapped with computation.

Optimal Example (2)

Select the page that will not be needed for the longest time

Example:

12 references, 7 faults

Page Refs	3 Page Frames			
	Fault?	Page Contents		
A	yes	A		
B	yes	B	A	
C	yes	C	B	A
D	yes	D	B	A
A	no	D	B	A
B	no	D	B	A
E	yes	E	B	A
A	no	E	B	A
B	no	E	B	A
C	yes	C	E	B
D	yes	D	C	E
E	no	D	C	E

Optimal Page Replacement

Idea:

Select the page that will not be needed for the longest time

Problem?

Optimal Page Replacement

Idea:

Select the page that will not be needed for the longest time

Problem:

- Can't know the future of a program
- Can't know when a given page will be needed next
- The optimal algorithm is unrealizable

Optimal Page Replacement

However:

We can use it as a control case for simulation studies

- Run the program once
- Generate a log of all memory references
- Do we need all of them?
- Use the log to simulate various page replacement algorithms
- Can compare others to “optimal” algorithm

FIFO Algorithm

Always replace the oldest page …

- *Replace the page that has been in memory for the longest time ($t_{loading}$)*

FIFO

12 references,
9 faults

Page Refs	3 Page Frames			
	Fault?	Page Contents		
A	yes	A		
B	yes	B	A	
C	yes	C	B	A
D	yes	D	C	B
A	yes	A	D	C
B	yes	B	A	D
E	yes	E	B	A
A	no	E	B	A
B	no	E	B	A
C	yes	C	E	B
D	yes	D	C	E
E	no	D	C	E

Belady's Anomaly (for FIFO)

As the number of page frames increase, so does the fault rate.

12 references, 10 faults

Page Refs	4 Page Frames				
	Fault?	Page Contents			
A	yes	A			
B	yes	B	A		
C	yes	C	B	A	
D	yes	D	C	B	A
A	no	D	C	B	A
B	no	D	C	B	A
E	yes	E	D	C	B
A	yes	A	E	D	C
B	yes	B	A	E	D
C	yes	C	B	A	E
D	yes	D	C	B	A
E	yes	E	D	C	B

FIFO Algorithm

Always replace the oldest page.

- *Replace the page that has been in memory for the longest time*

Implementation

Maintain a linked list of all pages in memory

Keep it in order of when they came into memory

The page at the tail of the list is oldest

Add new page to head of list

FIFO Algorithm

Disadvantage?

FIFO Algorithm

Disadvantage:

- The oldest page may be needed again soon
- Some page may be important throughout execution
- It will get old, but replacing it will cause an immediate page fault

How Can We Do Better?

Need an approximation of how likely each frame is to be accessed in the future

- If we base this on past behavior we need a way to track past behavior
- Tracking memory accesses requires hardware support to be efficient

Referenced and Dirty Bits

Each page table entry (and TLB entry!) has a

- *Referenced bit* - set by TLB when page read / written
- *Dirty / modified bit* - set when page is written
- If TLB entry for this page is valid, it has the most up to date version of these bits for the page
- OS must copy them into the page table entry during fault handling

Idea: use the information contained in these bits to drive the page replacement algorithm

Not Recently Used Algorithm (NRU)

Uses the Referenced Bit and the Dirty Bit

Initially, all pages have

- Referenced Bit = 0
- Dirty Bit = 0

Periodically... (e.g. whenever a timer interrupt occurs)

- Clear the Referenced Bit
- Referenced bit now indicates “recent” access

Not Recently Used Algorithm

When a page fault occurs...

Categorize each page...

Class 1:	Referenced = 0	Dirty = 0
Class 2:	Referenced = 0	Dirty = 1
Class 3:	Referenced = 1	Dirty = 0
Class 4:	Referenced = 1	Dirty = 1

Choose a victim page from class 1 … why?
If none, choose a page from class 2 … why?
If none, choose a page from class 3 … why?
If none, choose a page from class 4 … why?

Second Chance Algorithm

An implementation of NRU based on FIFO

Pages kept in a linked list (oldest at the front)

Look at the oldest page

- If its “referenced bit” is 0...
  - Select it for replacement
- Else
  - It was used recently; don’t want to replace it
  - Clear its “referenced bit”
  - Move it to the end of the list
- Repeat

What if every page was used in last clock tick?

Second Chance Example

12 references,
9 faults

Page Refs	3 Page Frames			
	Fault?	Page Contents		
A	yes	A*		
B	yes	B*	A*	
C	yes	C*	B*	A*
D	yes	D*	C	B
A	yes	A*	D*	C
B	yes	B*	A*	D*
E	yes	E*	B	A
A	no	E*	B	A*
B	no	E*	B*	A*
C	yes	C*	E	B
D	yes	D*	C*	E
E	no	D*	C*	E*

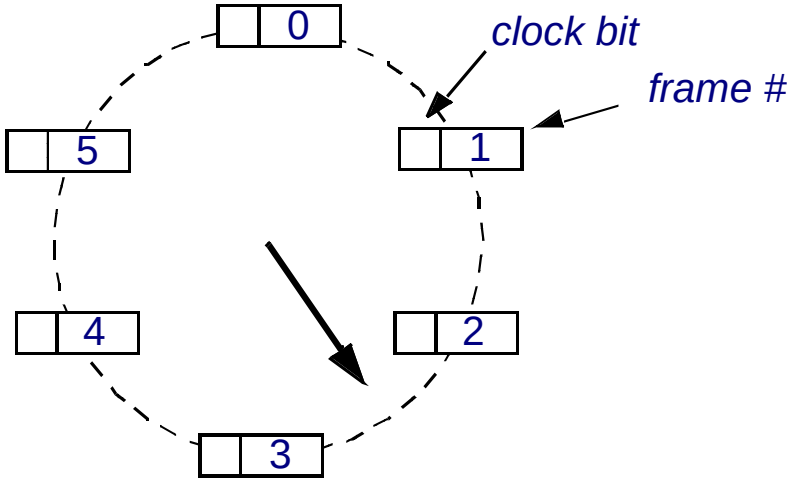
Implementation of Second Chance

Maintain a circular list of pages in memory

Set a bit for the page when a page is referenced

Search list looking for a victim page that does not have the referenced bit set

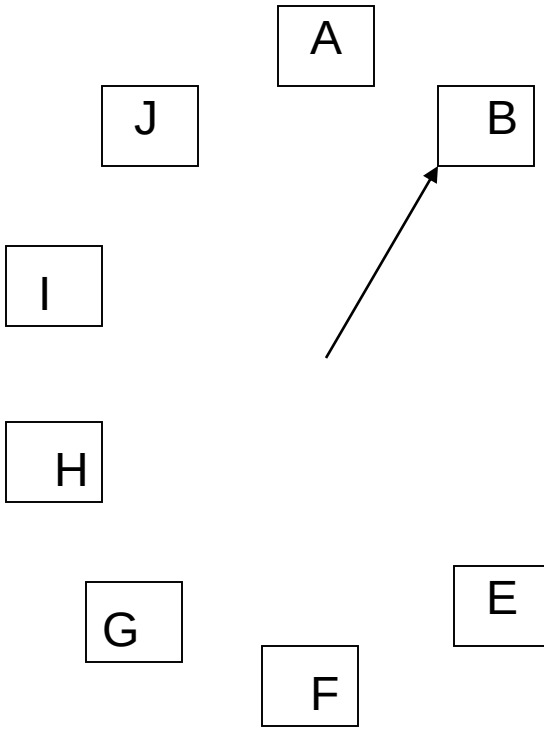
- If the bit is set, clear it and move on to the next page
- Replaces pages that haven't been referenced for one complete clock revolution



Clock Page Replacement Algorithm

Similar to second chance, however

Keep all page frames on a circular list in the form of a clock



Check the R Bit,
if R= 0, evict the page,
if R=1, clear the bit and advance

Least Recently Used Algorithm

A refinement of NRU that orders how recently a page was used

- Keep track of when a page is used ($t_{reference}$)
- Replace the page that has been used least recently

LRU Example

12 references,
10 faults

Page Refs	3 Page Frames			
	Fault?	Page Contents		
A	yes	A		
B	yes	B	A	
C	yes	C	B	A
D	yes	D	C	B
A	yes	A	D	C
B	yes	B	A	D
E	yes	E	B	A
A	no	A	E	B
B	no	B	A	E
C	yes	C	B	A
D	yes	D	C	B
E	yes	E	D	C

LRU and Anomalies

12 references,
8 faults

Page Refs	4 Page Frames				
	Fault?	Page Contents			
A	yes	A			
B	yes	B	A		
C	yes	C	B	A	
D	yes	D	C	B	A
A	no	A	D	C	B
B	no	B	A	D	C
E	yes	E	B	A	D
A	no	A	E	B	D
B	no	B	A	E	D
C	yes	C	B	A	E
D	yes	D	C	B	A
E	yes	E	D	C	B

Least Recently Used Algorithm

But how can we implement LRU?

Least Recently Used Algorithm

But how can we implement LRU?

Idea #1:

- Keep a linked list of all pages
- On every memory reference, Move that page to the front of the list
- The page at the tail of the list is replaced

Least Recently Used Algorithm

But how can we implement LRU?

… without requiring *every access* to be recorded?

Idea #2:

- MMU (hardware) maintains a counter
- Incremented on every clock cycle
- Every time a page table entry is used
  - MMU writes the value to the page table entry
  - This *timestamp* value is the *time-of-last-use*
- When a page fault occurs
  - OS looks through the page table
  - Identifies the entry with the oldest timestamp