بسم الله الرحمن الرحیم

«سیستم عامل»

۱۹۱

جلسه ۲۰: مدیریت حافظه (۸)

The optimal page replacement algorithm

- **Idea:**
  - Given all the data, how to find the optimal page replacement?
  - **Longest Forward Distance (LFD):** Select the page that will not be needed for the longest time

LFD = OPT

The optimal page replacement algorithm

- **Idea:**
  - Select the page that will not be needed for the longest time

- **Problem?**

The optimal page replacement algorithm

- **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

The optimal page replacement algorithm

- **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 page replacement algorithm

- Always replace the oldest page …
  - "Replace the page that has been in memory for the longest time."

FIFO page replacement algorithm

- Replace the page that was first brought into memory
- Example: Memory system with 4 frames:

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

FIFO page replacement algorithm

- Replace the page that was first brought into memory
- Example: Memory system with 4 frames:

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

FIFO page replacement algorithm

- Replace the page that was first brought into memory
- Example: Memory system with 4 frames:

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

FIFO page replacement algorithm

- Replace the page that was first brought into memory
- Example: Memory system with 4 frames:

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

FIFO page replacement 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 page replacement algorithm

- **Disadvantage?**

FIFO page replacement 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

Page table: 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**

Page table: referenced and dirty bits

- **Some hardware does not have support for the dirty bit**
- **Instead, memory protection can be used to emulate it**
- **Idea:**
  - Software sets the protection bits for all pages to "read only"
  - When program tries to update the page...
    - A trap occurs
    - Software sets the Dirty Bit in the page table and clears the ReadOnly bit
    - Resumes execution of the program

Not recently used page replacement alg.

- **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 page replacement alg.

- 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

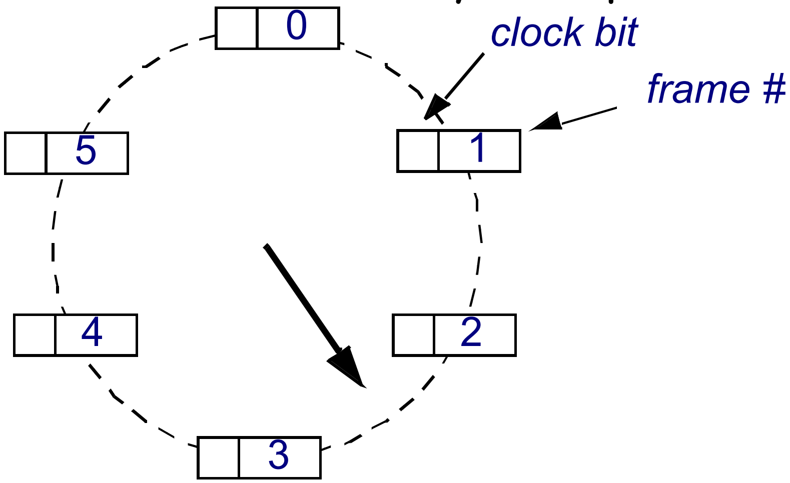
- Choose a victim page from class 1 … why?
- If none, choose a page from class 2 … why?
- If none, choose a page from class 3 … why?
- If none, choose a page from class 4 … why?

Second chance page replacement alg.

- **An implementation of NRU based on FIFO**
- **Pages kept in a linked list**
  - Oldest is at the front of the list
- **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?**
  - Select a page at random
-

Clock algorithm (an implementation of NRU)

- **Maintain a circular list of pages in memory**
- **Set a bit for the page when a page is referenced**
- **Clock sweeps over memory 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 – essentially an implementation of NRU



Least recently used algorithm (LRU)

- **A refinement of NRU that orders how recently a page was used**
  - Keep track of when a page is used
  - Replace the page that has been used least recently

LRU page replacement

- Replace the page that hasn't been referenced in 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	0	a	a	a	a	a	a	a	a	a	a	a
Frames	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
Page faults							X				X	X

Least recently used algorithm (LRU)

- But how can we implement this?

Least recently used algorithm (LRU)

- **But how can we implement this?**

- **Implementation #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 (LRU)

- **But how can we implement this?**
  - … without requiring "every access" to be recorded?

- **Implementation #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
    - **Software looks through the page table**
    - **Idenitifies the entry with the oldest timestamp**

Least recently used algorithm (LRU)

- **What if we don't have hardware support for a counter?**

- **Implementation #3:**
  - Maintain a counter in software
  - On every timer interrupt...
    - **Increment counter**
    - **Run through the page table**
    - **For every entry that has "ReferencedBit" = 1**
      - **Update its timestamp**
      - **Clear the ReferencedBit**
  - Approximates LRU
  - If several have oldest time, choose one arbitrarily