IT308: Operating Systems

Virtual memory: Page replacement policies

Beyond Physical RAM

- Physical RAM not large enough to store all the virtual pages that your processes actually want
- So let's put some of our pages on disk

Beyond Physical RAM

- Physical RAM not large enough to store all the virtual pages that your processes actually want
- So let's put some of our pages on disk
- In each PTE, we keep a bit for "present"
- If the present bit is 0 during translation, then this causes a page fault

Beyond Physical RAM

- Physical RAM not large enough to store all the virtual pages that your processes actually want
- So let's put some of our pages on disk
- In each PTE, we keep a bit for "present"
- If the present bit is 0 during translation, then this causes a page fault
- Who would raise this "page fault"?

Page fault

- The page fault is an exception raised by the CPU which invokes the page fault handler of the OS
- The OS then reads the page from disk into a physical page

Page fault

- The page fault is an exception raised by the CPU which invokes the page fault handler of the OS
- The OS then reads the page from disk into a physical page
- Recall that the page table entry (approx. 4 bytes) contains the physical page number.

Page fault

- The page fault is an exception raised by the CPU which invokes the page fault handler of the OS
- The OS then reads the page from disk into a physical page
- Recall that the page table entry (approx. 4 bytes) contains the physical page number.
- But if it isn't present, it isn't in a physical page. So this number is usually reused as the place on disk

Page faults are slow!

- While the OS reads this page from disk, it will often run another process. So page faults are expensive!

What if memory is full?

- Eventually we run out of physical pages.
- When we read in a page from disk we may have to "evict" or "page out" a page
- Which one to evict? The one you don't need!
- We have to take another process' page, write it to disk, and use that. (make sure you zero it first - why?)

Question

If every physical page on a machine with 4GB of RAM is in use and we need to evict a page to make room for another, how many pages do we have to choose from? (assume pages are 4kB)

(A) ∼10k

(B) ∼100k

(C) ∼1M

(D) ∼10M

Page replacement

- The victim page gets evicted
- There are a lot to choose from. 4GB memory $= 2^{32}/2^{12} = 2^{20}$ pages of physical memory!

Page replacement

- The victim page gets evicted
- There are a lot to choose from. 4GB memory $= 2^{32}/2^{12} = 2^{20}$ pages of physical memory!
- The choice of which page to evict is called the page replacement algorithm

Optimal

- The optimal policy is to evict the page that will be needed the furthest in the future
- This will result in the highest achievable hit rate

Optimal

- The optimal policy is to evict the page that will be needed the furthest in the future
- This will result in the highest achievable hit rate
- Unfortunately it isn't possible to implement, but we can use it as a point of comparison

Example

- Use a cache of size 3
- And accesses of the following pages:
  - 0, 1, 2, 0, 1, 3, 0, 3, 1, 2, 1

Example

Access	Hit/Miss?	Evict	Resulting Cache State
0	Miss		0
1	Miss		0, 1
2	Miss		0, 1, 2
0	Hit		0, 1, 2
1	Hit		0, 1, 2
3	Miss	2	0, 1, 3
0	Hit		0, 1, 3
3	Hit		0, 1, 3
1	Hit		0, 1, 3
2	Miss	3	0, 1, 2
1	Hit		0, 1, 2

Compute the hit rate

Access	Hit/Miss?
0	Miss
1	Miss
2	Miss
0	Hit
1	Hit
3	Miss
0	Hit
3	Hit
1	Hit
2	Miss
1	Hit

- Hits / (Hits + Misses)

Compute the hit rate

Access	Hit/Miss?
0	Miss
1	Miss
2	Miss
0	Hit
1	Hit
3	Miss
0	Hit
3	Hit
1	Hit
2	Miss
1	Hit

- Hits / (Hits + Misses)
  - 6/(6+5) = 54.5%

Compute the hit rate

Access	Hit/Miss?
0	Miss
1	Miss
2	Miss
0	Hit
1	Hit
3	Miss
0	Hit
3	Hit
1	Hit
2	Miss
1	Hit

- Hits / (Hits + Misses)
  - $6/(6+5) = 54.5\%$
- But some of the misses are compulsory!

Compute the hit rate

Access	Hit/Miss?
0	Miss
1	Miss
2	Miss
0	Hit
1	Hit
3	Miss
0	Hit
3	Hit
1	Hit
2	Miss
1	Hit

- Hits / (Hits + Misses)
  - $6/(6+5) = 54.5\%$
- But some of the misses are compulsory!
- Better measure excludes them
  - $6/(6+1) = 85.7\%$

FIFO

- Since we can't implement OPT, let's do something really simple (and $O(1)$)
- Whatever was first into the cache gets evicted

Example

Access	Hit/Miss?	Evict	Resulting Cache State	
0	Miss		First-in→	0
1	Miss		First-in→	0, 1
2	Miss		First-in→	0, 1, 2
0	Hit		First-in→	0, 1, 2
1	Hit		First-in→	0, 1, 2
3	Miss	0	First-in→	1, 2, 3
0	Miss	1	First-in→	2, 3, 0
3	Hit		First-in→	2, 3, 0
1	Miss	2	First-in→	3, 0, 1
2	Miss	3	First-in→	0, 1, 2
1	Hit		First-in→	0, 1, 2

Question

Access	Hit/Miss?
0	Miss
1	Miss
2	Miss
0	Hit
1	Hit
3	Miss
0	Miss
3	Hit
1	Miss
2	Miss
1	Hit

What is the hit rate (excluding compulsory misses) of FIFO in this example?

(A) 4/11

(B) 4/7

(C) 11/4

(D) 7/11

What about implementation?

- In FIFO items are evicted in the order they are inserted
- May be implemented by keeping a queue of page numbers
  - We replace the page at the head of the queue
  - When a page is brought into memory, it is inserted at the tail of the queue

Belady's Anomaly

- In general you would expect cache hit rate to increase when cache gets larger
- Does this always happen?

Belady's Anomaly

- In general you would expect cache hit rate to increase when cache gets larger
- Does this always happen?
- Not necessarily for FIFO!
  - You can sometimes get a worse hit rate with a larger cache size

Example

- Accesses: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

(a) size 3

Access	Hit	State (after)
1	no	1
2	no	1,2
3	no	1,2,3
4	no	2,3,4
1	no	3,4,1
2	no	4,1,2
5	no	1,2,5
1	yes	1,2,5
2	yes	1,2,5
3	no	2,5,3
4	no	5,3,4
5	yes	5,3,4

(b) size 4

Access	Hit	State (after)
1		
2		
3		
4		
1		
2		
5		
1		
2		
3		
4		
5		

Example

- Accesses: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

(a) size 3

Access	Hit	State (after)
1	no	1
2	no	1,2
3	no	1,2,3
4	no	2,3,4
1	no	3,4,1
2	no	4,1,2
5	no	1,2,5
1	yes	1,2,5
2	yes	1,2,5
3	no	2,5,3
4	no	5,3,4
5	yes	5,3,4

(b) size 4

Access	Hit	State (after)
1	no	1
2	no	1,2
3	no	1,2,3
4	no	1,2,3,4
1	yes	1,2,3,4
2	yes	1,2,3,4
5	no	2,3,4,5
1	no	3,4,5,1
2	no	4,5,1,2
3	no	5,1,2,3
4	no	1,2,3,4
5	no	2,3,4,5

Random

- Oh, when things get hard, just roll the dice!
- Simply pick a random victim to evict, not even trying to be intelligent

Least Recently Used (LRU)

- Replace page that hasn't been used for the longest time

Least Recently Used (LRU)

- Replace page that hasn't been used for the longest time
- Programs have locality, so if something not used for a while, unlikely to be used in the near future.
- Seems like LRU should be a good approximation to OPT.

Least-Recently-Used (LRU)

- Accesses: 0, 1, 2, 0, 1, 3, 0, 3, 1, 2, 1

Access	Hit/Miss?	Evict	Resulting Cache State
0	Miss		LRU→ 0
1	Miss		LRU→ 0, 1
2	Miss		LRU→ 0, 1, 2
0			
1			
3			
0			
3			
1			
2			
1			

Least-Recently-Used (LRU)

- Accesses: 0, 1, 2, 0, 1, 3, 0, 3, 1, 2, 1

Access	Hit/Miss?	Evict	Resulting Cache State
0	Miss		LRU→ 0
1	Miss		LRU→ 0, 1
2	Miss		LRU→ 0, 1, 2
0	Hit		LRU→ 1, 2, 0
1			
3			
0			
3			
1			
2			
1			

Least-Recently-Used (LRU)

- Accesses: 0, 1, 2, 0, 1, 3, 0, 3, 1, 2, 1

Access	Hit/Miss?	Evict	Resulting Cache State
0	Miss		LRU→ 0
1	Miss		LRU→ 0, 1
2	Miss		LRU→ 0, 1, 2
0	Hit		LRU→ 1, 2, 0
1	Hit		LRU→ 2, 0, 1
3			
0			
3			
1			
2			
1			

Least-Recently-Used (LRU)

- Accesses: 0, 1, 2, 0, 1, 3, 0, 3, 1, 2, 1

Access	Hit/Miss?	Evict	Resulting Cache State
0	Miss		LRU→ 0
1	Miss		LRU→ 0, 1
2	Miss		LRU→ 0, 1, 2
0	Hit		LRU→ 1, 2, 0
1	Hit		LRU→ 2, 0, 1
3	Miss	2	LRU→ 0, 1, 3
0			
3			
1			
2			
1			

Least-Recently-Used (LRU)

- Accesses: 0, 1, 2, 0, 1, 3, 0, 3, 1, 2, 1

Access	Hit/Miss?	Evict	Resulting Cache State
0	Miss		LRU→ 0
1	Miss		LRU→ 0, 1
2	Miss		LRU→ 0, 1, 2
0	Hit		LRU→ 1, 2, 0
1	Hit		LRU→ 2, 0, 1
3	Miss	2	LRU→ 0, 1, 3
0	Hit		LRU→ 1, 3, 0
3			
1			
2			
1			

Least-Recently-Used (LRU)

- Accesses: 0, 1, 2, 0, 1, 3, 0, 3, 1, 2, 1

Access	Hit/Miss?	Evict	Resulting Cache State	
0	Miss		LRU→	0
1	Miss		LRU→	0, 1
2	Miss		LRU→	0, 1, 2
0	Hit		LRU→	1, 2, 0
1	Hit		LRU→	2, 0, 1
3	Miss	2	LRU→	0, 1, 3
0	Hit		LRU→	1, 3, 0
3	Hit		LRU→	1, 0, 3
1				
2				
1				

Least-Recently-Used (LRU)

- Accesses: 0, 1, 2, 0, 1, 3, 0, 3, 1, 2, 1

Access	Hit/Miss?	Evict	Resulting Cache State	
0	Miss		LRU→	0
1	Miss		LRU→	0, 1
2	Miss		LRU→	0, 1, 2
0	Hit		LRU→	1, 2, 0
1	Hit		LRU→	2, 0, 1
3	Miss	2	LRU→	0, 1, 3
0	Hit		LRU→	1, 3, 0
3	Hit		LRU→	1, 0, 3
1	Hit		LRU→	0, 3, 1
2				
1				

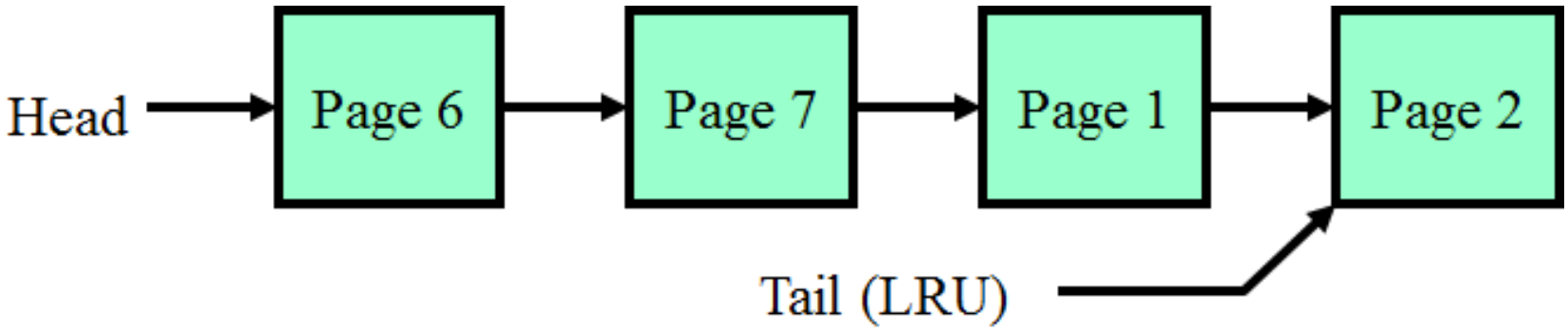
Least-Recently-Used (LRU)

- Accesses: 0, 1, 2, 0, 1, 3, 0, 3, 1, 2, 1

Access	Hit/Miss?	Evict	Resulting Cache State	
0	Miss		LRU→	0
1	Miss		LRU→	0, 1
2	Miss		LRU→	0, 1, 2
0	Hit		LRU→	1, 2, 0
1	Hit		LRU→	2, 0, 1
3	Miss	2	LRU→	0, 1, 3
0	Hit		LRU→	1, 3, 0
3	Hit		LRU→	1, 0, 3
1	Hit		LRU→	0, 3, 1
2	Miss	0	LRU→	3, 1, 2
1	Hit		LRU→	3, 2, 1

What about implementation?

- We can use a list!



- On each use, remove page from list and place at head
- LRU page is at tail
- List must be updated at every memory reference; in FIFO we only have to update the list when evicting

Dirty Pages

- Pages that have only been read, but not modified are the same as what is stored on disk. They are clean

Dirty Pages

- Pages that have only been read, but not modified are the same as what is stored on disk. They are clean
- Pages that have to be written to disk before eviction are dirty

Dirty Pages

- Pages that have only been read, but not modified are the same as what is stored on disk. They are clean
- Pages that have to be written to disk before eviction are dirty
- Dirty pages are more expensive to evict

Optimization

- We can choose to evict clean pages instead of dirty ones
- To support this the MMU will mark pages (the PTE) with a dirty bit $= 1$ when the page gets modified