# IT308: Operating Systems

## Virtual memory: Page replacement policies cont.

# Last Time: Page Replacement Policy

- When the OS must allocate a frame but none are available, the page replacement policy chooses a page to evict from a frame
- Optimal page replacement (OPT) policy:
  - Evict the page that will be used farthest in the future
  - Produces the smallest number of page-faults possible

# Last Time: Page Replacement Policy

- When the OS must allocate a frame but none are available, the page replacement policy chooses a page to evict from a frame
- Optimal page replacement (OPT) policy:
  - Evict the page that will be used farthest in the future
  - Produces the smallest number of page-faults possible
- Problem: OPT is unrealizable – can't really implement it (why?)

# Last Time: Page Replacement Policy

- Least Recently Used (LRU) policy:
  - When a page must be evicted, always choose the one that was used farthest in the past
  - (LRU is basically OPT applied to the reference string in reverse)
- Typically requires dedicated hardware to implement

# Last Time: Page Replacement Policy

- Least Recently Used (LRU) policy:
  - When a page must be evicted, always choose the one that was used farthest in the past
  - (LRU is basically OPT applied to the reference string in reverse)
- Typically requires dedicated hardware to implement
- As stated, LRU policy is still quite difficult to implement
  - Problem: must update the data needed to implement LRU on every memory access

# Last Time: Page Replacement Policy

- Least Recently Used (LRU) policy:
  - When a page must be evicted, always choose the one that was used farthest in the past
  - (LRU is basically OPT applied to the reference string in reverse)
- Typically requires dedicated hardware to implement
- As stated, LRU policy is still quite difficult to implement
  - Problem: must update the data needed to implement LRU on every memory access
  - (And, this data is usually stored in memory as well . . . )

# Example

- Suppose we have 3 page frames, 4 virtual pages (0–3), and following reference stream:
  - 0, 1, 2, 0, 1, 3, 0, 3, 1, 2, 1
- Consider OPT page replacement:

# Example

- Suppose we have 3 page frames, 4 virtual pages (0–3), and following reference stream:
  - 0, 1, 2, 0, 1, 3, 0, 3, 1, 2, 1
- Consider OPT page replacement:

| 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 |

# Example

- Suppose we have 3 page frames, 4 virtual pages (0–3), and following reference stream:
  - 0, 1, 2, 0, 1, 3, 0, 3, 1, 2, 1
- Consider OPT page replacement:

| 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 |

- What will LRU do?

# Example

- Suppose we have 3 page frames, 4 virtual pages (0–3), and following reference stream:
  - 0, 1, 2, 0, 1, 3, 0, 3, 1, 2, 1
- Consider OPT page replacement:

| 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 |

- What will LRU do?
  - LRU will make same decisions as OPT here, but won't always be true!

# Another Example

- As before suppose we have 3 page frames, 4 virtual pages (0–3)
- Consider the following reference stream:
  - 0, 1, 2, 3, 0, 1, 2, 3
- Here LRU performs quite badly (same as FIFO)
  - 8 misses

# Another Example

- As before suppose we have 3 page frames, 4 virtual pages (0–3)
- Consider the following reference stream:
  - 0, 1, 2, 3, 0, 1, 2, 3
- Here LRU performs quite badly (same as FIFO)
  - 8 misses
  - When referencing 3, evicting 0 is a bad choice, since need 0 right away

# Another Example

- As before suppose we have 3 page frames, 4 virtual pages (0–3)
- Consider the following reference stream:
  - 0, 1, 2, 3, 0, 1, 2, 3
- Here LRU performs quite badly (same as FIFO)
  - 8 misses
  - When referencing 3, evicting 0 is a bad choice, since need 0 right away
- OPT does much better!
  - 5 misses

# Another Example

- As before suppose we have 3 page frames, 4 virtual pages (0–3)
- Consider the following reference stream:
  - 0, 1, 2, 3, 0, 1, 2, 3
- Here LRU performs quite badly (same as FIFO)
  - 8 misses
  - When referencing 3, evicting 0 is a bad choice, since need 0 right away
- OPT does much better!
  - 5 misses
  - When referencing 3, evict 2 since it is referenced farthest in future

# Least Recently Used Policy

- Two general approaches for implementing LRU policy
- Option 1: Use a counter to record the last time each page is accessed
  - e.g. update counter on every instruction, or on every memory access

# Least Recently Used Policy

- Two general approaches for implementing LRU policy
- Option 1: Use a counter to record the last time each page is accessed
  - e.g. update counter on every instruction, or on every memory access
- Extend page-table entries to hold the counter value when the memory was last accessed
  - The MMU must update this value on every page access

# Least Recently Used Policy

- When a page must be evicted:
  - Scan thru all pages in memory to find page with oldest counter value
  - $O(N)$ scan to find oldest page-counter value

# Least Recently Used Policy

- When a page must be evicted:
  - Scan thru all pages in memory to find page with oldest counter value
  - $O(N)$ scan to find oldest page-counter value
- Memory accesses incur additional accesses to update a page's counter-value
  - Can cache values in TLB entries to reduce writes to main memory

# Least Recently Used Policy (2)

- Option 2: Use a queue to track access order of all pages
- When a page is accessed, move it to front of the queue
  - Again, this must happen on every page access

# Least Recently Used Policy (2)

- Option 2: Use a queue to track access order of all pages
- When a page is accessed, move it to front of the queue
  - Again, this must happen on every page access
- When a page must be evicted:
  - Page at back of the queue is the least recently used; evict that one!

# Least Recently Used Policy (2)

- Option 2: Use a queue to track access order of all pages
- When a page is accessed, move it to front of the queue
  - Again, this must happen on every page access
- When a page must be evicted:
  - Page at back of the queue is the least recently used; evict that one!
- As unappealing as counter approach, but in different ways

# Least Recently Used Policy (3)

- Choosing a page to evict is fast and easy
  - Just pull the last element off the end of the queue

# Least Recently Used Policy (3)

- Choosing a page to evict is fast and easy
  - Just pull the last element off the end of the queue
- . . . but the per-memory-access cost is significantly higher!
  - Most accesses will incur linked-list manipulations, requiring multiple additional memory accesses per access

# Least Recently Used Policy (3)

- Choosing a page to evict is fast and easy
  - Just pull the last element off the end of the queue
- . . . but the per-memory-access cost is significantly higher!
  - Most accesses will incur linked-list manipulations, requiring multiple additional memory accesses per access
- In practice, LRU is too slow / difficult to implement for virtual memory systems

# Approximating the LRU Policy

- Systems can implement a policy that approximates LRU
- MMUs usually maintain several bits in page table entries:
  - An "accessed" bit recording if the page was read or written
  - A "dirty" bit recording if the page was written

# Approximating the LRU Policy

- Systems can implement a policy that approximates LRU
- MMUs usually maintain several bits in page table entries:
  - An “accessed” bit recording if the page was read or written
  - A “dirty” bit recording if the page was written
- Replacement policies can examine the “accessed” bit on regular interval, to see if a page was accessed recently

# Approximating the LRU Policy (2)

- Example: Not Frequently Used Policy
  - Maintain a counter for each page in memory
- Periodically scan through all pages on a timer interrupt:
  - If a page's "accessed" bit is set to 1, increment the page's counter and clear the page's "accessed" bit

# Approximating the LRU Policy (2)

- Example: Not Frequently Used Policy
  - Maintain a counter for each page in memory
- Periodically scan through all pages on a timer interrupt:
  - If a page's "accessed" bit is set to 1, increment the page's counter and clear the page's "accessed" bit
- When a page must be evicted, choose the page with the lowest count

# Approximating the LRU Policy (3)

- Problem with Not Frequently Used policy is that it never forgets a page's history
- e.g. if a page is accessed heavily in the early parts of a program's execution, then never again – it will be unlikely to be paged out

# Approximating the LRU Policy (4)

- A much better policy is called the Aging policy
- As before, the OS maintains a b-bit value for each page
- On a periodic timer, OS traverses all pages in memory:
  - Shift the page's value to the right by one bit, store the page's "accessed" bit as the new topmost bit, then clear "accessed" bit

# Approximating the LRU Policy (4)

- A much better policy is called the Aging policy
- As before, the OS maintains a b-bit value for each page
- On a periodic timer, OS traverses all pages in memory:
  - Shift the page's value to the right by one bit, store the page's "accessed" bit as the new topmost bit, then clear "accessed" bit
- Pages with more recent accesses will have a larger value than pages with less recent accesses
- Evict the page(s) with the lowest value

# The Aging Policy

- Each interrupt, page table is scanned and age values are updated
- Smallest age approximates least recently used pages
- Example:

| | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ... | 2 | 6 | 1 | 5 | 7 | 7 | 7 | 7 | 5 | 1 | 6 | 2 | 3 | 4 | 1 | 2 | 3 | 4 | 4 | 4 | 3 | 4 | ... |

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

| Page | $Age_2$ | $Age_{10}$ |
|---|---|---|
| 1 | 10000000 | 128 |
| 2 | 10000000 | 128 |
| 3 | 00000000 | 0 |
| 4 | 00000000 | 0 |
| 5 | 10000000 | 128 |
| 6 | 10000000 | 128 |
| 7 | 10000000 | 128 |

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

| Page | $Age_2$ | $Age_{10}$ |
|---|---|---|
| 1 | 11000000 | 192 |
| 2 | 01000000 | 64 |
| 3 | 00000000 | 0 |
| 4 | 00000000 | 0 |
| 5 | 11000000 | 192 |
| 6 | 01000000 | 64 |
| 7 | 11000000 | 192 |

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

| Page | $Age_2$ | $Age_{10}$ |
|---|---|---|
| 1 | 11100000 | 224 |
| 2 | 10100000 | 160 |
| 3 | 10000000 | 128 |
| 4 | 10000000 | 128 |
| 5 | 01100000 | 96 |
| 6 | 10100000 | 160 |
| 7 | 01100000 | 96 |

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

| Page | $Age_2$ | $Age_{10}$ |
|---|---|---|
| 1 | 01110000 | 112 |
| 2 | 11010000 | 160 |
| 3 | 11000000 | 192 |
| 4 | 11000000 | 192 |
| 5 | 00110000 | 48 |
| 6 | 01010000 | 80 |
| 7 | 00110000 | 48 |

# The Aging Policy (2)

- With aging policy, common to have multiple pages with the same age value
  - LRU would know which page was accessed furthest in past, but aging policy treats them the same

# The Aging Policy (2)

- With aging policy, common to have multiple pages with the same age value
  - LRU would know which page was accessed furthest in past, but aging policy treats them the same
- Similarly, if two pages have a value of 0:
  - LRU would know which one was accessed most recently, but aging views both as having not been accessed recently

# The Aging Policy (3)

- The main difference between aging policy and LRU is that aging has a much lower resolution on its "recency" info
- Nonetheless, aging policy generally performs well with a relatively small number of bits, e.g. 8 or 16 bits per page

# Other Policies Using the Accessed Bit

- Many other replacement policies that use “accessed” bit
- Example: make FIFO policy more intelligent
  - Original policy: always evict the page at the front of the FIFO
  - Tweak this policy to also use a page’s “accessed” bit

# Other Policies Using the Accessed Bit

- When a page must be evicted:
  - Consider the page at the front of the FIFO
  - If the page's "accessed" bit is 1, clear the "accessed" bit and then move the page back to the end of the FIFO
  - Otherwise, evict the page at the front of the FIFO
- Called the Second-Chance replacement policy
  - If a page has been accessed during its time in the FIFO, it is given a second chance

# Second-Chance Replacement Policy

- What happens if all pages have their "accessed" bits set?
  - Pager will scan through all pages in the FIFO ...
  - Every page's "accessed" bit will be cleared during this pass ...
  - On second pass, pager will simply evict the first page in the FIFO
- Second-chance policy degenerates to FIFO replacement if all pages have been accessed since the last timer tick

# Question

Suppose you had a virtual memory system with a physical memory that consists of only 4 page frames. Suppose also that “accessed” bits are supported and are cleared after every 6 memory accesses.

Consider the following sequence of page accesses:

- 3, 0, 1, 2, 3, 0, 1, 4

Which page is evicted when page 4 needs to be brought into memory, for the following policies?

- LRU
- Second-chance