CMSC421: Principles of Operating Systems

Nilanjan Banerjee

Assistant Professor, University of Maryland
Baltimore County
nilanb@umbc.edu
http://www.csee.umbc.edu/~nilanb/teaching/421/

Acknowledgments: Some of the slides are adapted from Prof. Mark Corner and Prof. Emery Berger's OS course at Umass Amherst

Announcements

- Project 2 progress report due one week from Nov. 9th

Talked about malloc? What about physical frame mgmt?

- **malloc** works in virtual memory (works in user space)
  - Manages free blocks
  - Allocates virtual address on the heap
- Remember the OS still has to manage physical frames
  - The problem that the OS faces with physical frame allocation is the similar to malloc
  - Manage physical frames that all processes in the system requests.
- Difference with malloc
  - Has to work across all processes
    - Each process perceives 4GB of space, but in actuality there is only 4GB of physical memory space

Tasks of the OS physical page management unit

- **Allocate new pages to applications**
  - OS do this lazily
  - malloc call would usually return immediately
  - OS allocates a new physical only when the process reads/writes to the page
  - Similar to the Copy-on-Write policy for fork()
- In the event that all physical frames are taken
  - OS needs to evict pages
    - Take page from main memory and store it on swap space
  - Needs a policy for evicting pages

Page replacement policy for Demand Paging?

What is the optimal page replacement policy?

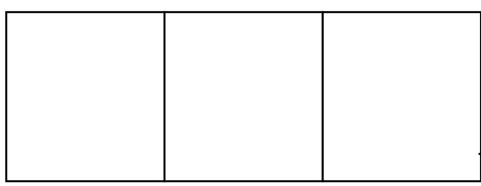
Optimal Page Replacement policy

- **Find the page that is going to used farthest into the future**
  - Evict the page from main memory to swap space
  - Allocate the freed page to the new process
  - Problems: it is impossible to predict the future

- Approximation is LRU (least recently used page)
  - Find the page that is least recently used and evict it
  - Remember this has to be **super-fast**
  - **What would be techniques to implement this in the kernel?**

Implementing Exact LRU

- On each reference, time stamp page
- When we need to evict: select oldest page = least-recently used



A, B, C, B, C, C, D

Implementing Exact LRU

- On each reference, time stamp page
- When we need to evict: select oldest page = least-recently used

A 1		

A, B, C, B, C, C, D

Implementing Exact LRU

- On each reference, time stamp page
- When we need to evict: select oldest page = least-recently used

A	B	
1	2	

A, B, C, B, C, C, D

Implementing Exact LRU

- On each reference, time stamp page
- When we need to evict: select oldest page = least-recently used

A	B	C
1	2	3

A, B, C, B, C, C, D

Implementing Exact LRU

- On each reference, time stamp page
- When we need to evict: select oldest page = least-recently used

A	B	C
1	4	3

A, B, C, B, C, C, D

Implementing Exact LRU

- On each reference, time stamp page
- When we need to evict: select oldest page = least-recently used

A	B	C
1	4	5

A, B, C, B, C, C, D

Implementing Exact LRU

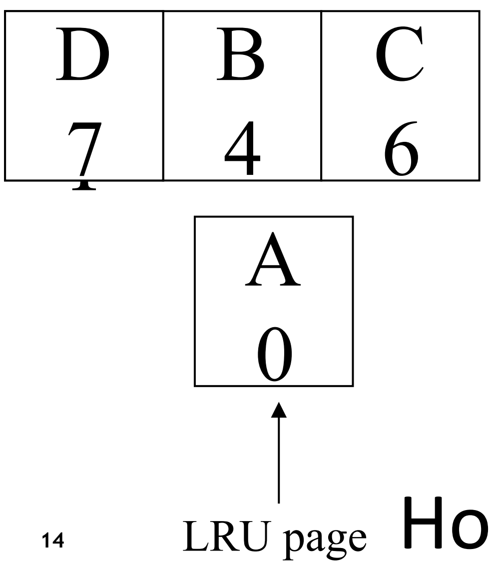
- On each reference, time stamp page
- When we need to evict: select oldest page
  = least-recently used

A	B	C
1	4	6

A, B, C, B, C, C, D

Implementing Exact LRU

- On each reference, time stamp page
- When we need to evict: select oldest page = least-recently used



A, B, C, B, C, C D,

How should we implement this?

Implementing Exact LRU

- Could keep pages in order
  – optimizes eviction
  - Priority queue:
    update = O(log n), eviction = O(log n)

- Optimize for common case!
  - Common case: hits, not misses
  - Hash table:
    update = O(1), eviction = O(n)

Cost of Maintaining Exact LRU

- Hash tables: too expensive
  - On every reference:
    - Compute hash of page address
    - Update time stamp

Cost of Maintaining Exact LRU

- Alternative: doubly-linked list
  - Move items to front when referenced
  - LRU items at end of list
  - Still too expensive
    - 4-6 pointer updates per reference
- Can we do better?

Hardware Support and approximate LRU (Linux Kernel)

- Maintain reference bits for every page
  - On each access, set reference bit to 1
  - Page replacement algorithm periodically resets reference bits

A	B	C
1	1	1

A, B, C, B, C, C, D

Hardware Support

- Maintain reference bits for every page
  - On each access, set reference bit to 1
  - Page replacement algorithm periodically resets reference bits

A	B	C
0	0	0

A, B, C, | B, C, C, D

reset reference bits

Hardware Support

- Maintain reference bits for every page
  - On each access, set reference bit to 1
  - Page replacement algorithm periodically resets reference bits

A	B	C
0	1	0

A, B, C, | B, C, C, D

Hardware Support

- Maintain reference bits for every page
  - On each access, set reference bit to 1
  - Page replacement algorithm periodically resets reference bits

A	B	C
0	1	1

A, B, C, | B, C, C, D

Hardware Support

- Maintain reference bits for every page
  - On each access, set reference bit to 1
  - Page replacement algorithm periodically resets reference bits

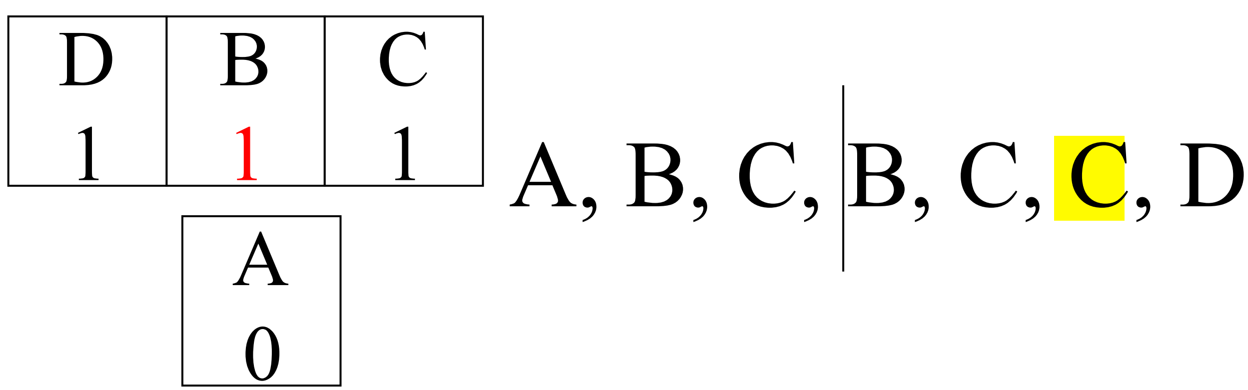
A	B	C
0	1	1

A, B, C, | B, C, C, D

Hardware Support

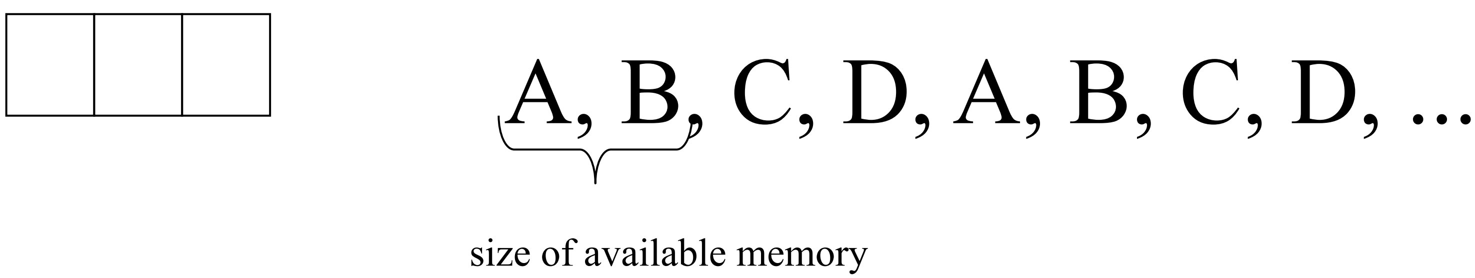
- Maintain reference bits for every page
  - On each access, set reference bit to 1
  - Page replacement algorithm periodically resets reference bits
  - Evict page with reference bit = 0

- Cost per miss = O(n)



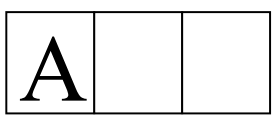
Most-Recently Used (MRU)

- Evict most-recently used page
- Shines for LRU's worst-case:



Most-Recently Used (MRU)

- Evict most-recently used page
- Shines for LRU’s worst-case: loop that exceeds RAM size

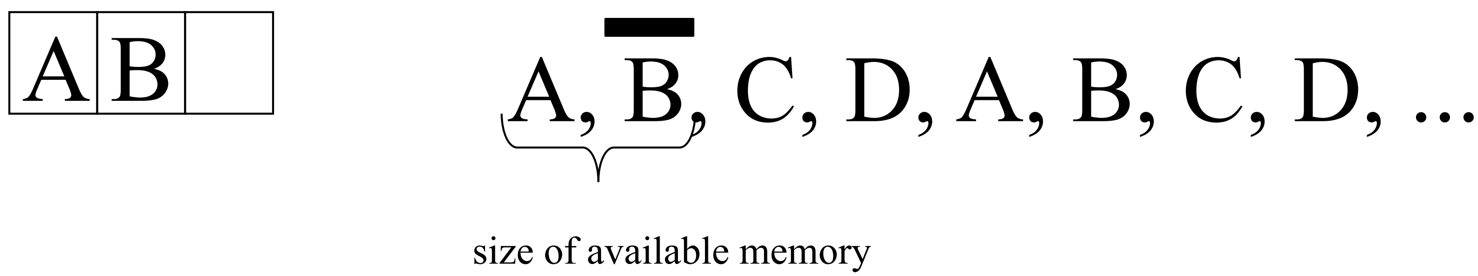


A, B, C, D, A, B, C, D, ...

size of available memory

Most-Recently Used (MRU)

- Evict most-recently used page
- Shines for LRU’s worst-case: loop that exceeds RAM size



Most-Recently Used (MRU)

- Evict most-recently used page
- Shines for LRU's worst-case: loop that exceeds RAM size

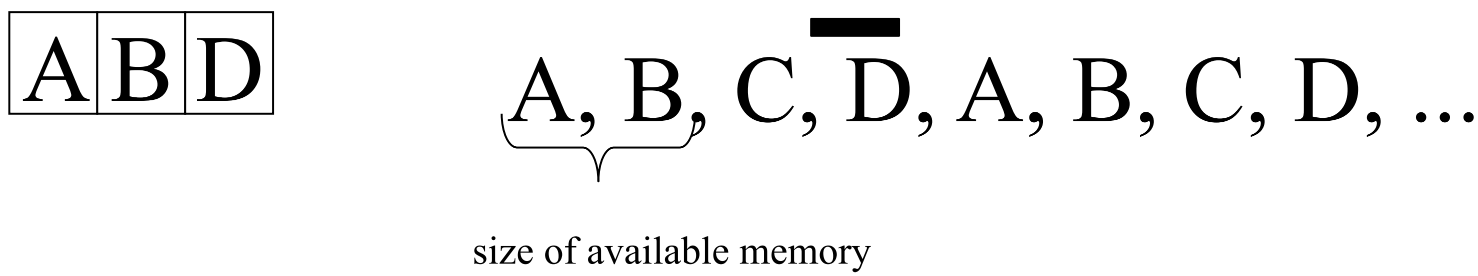
A	B	C

A, B, C, D, A, B, C, D, ...

size of available memory

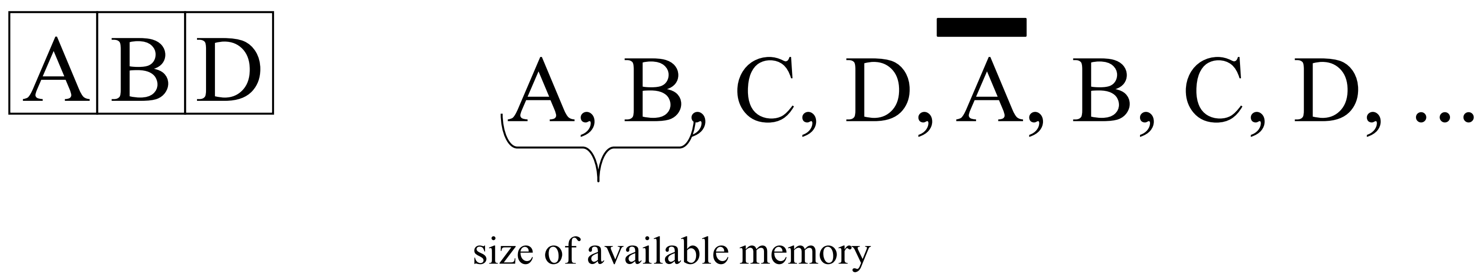
Most-Recently Used (MRU)

- Evict most-recently used page
- Shines for LRU’s worst-case: loop that exceeds RAM size



Most-Recently Used (MRU)

- Evict most-recently used page
- Shines for LRU's worst-case: loop that exceeds RAM size



Most-Recently Used (MRU)

- Evict most-recently used page
- Shines for LRU's worst-case: loop that exceeds RAM size

A	B	D

A, B, C, D, A, B, C, D, ...

size of available memory

Most-Recently Used (MRU)

- Evict most-recently used page
- Shines for LRU’s worst-case: loop that exceeds RAM size

A	B	D

A, B, C, D, A, B, C, D, ...

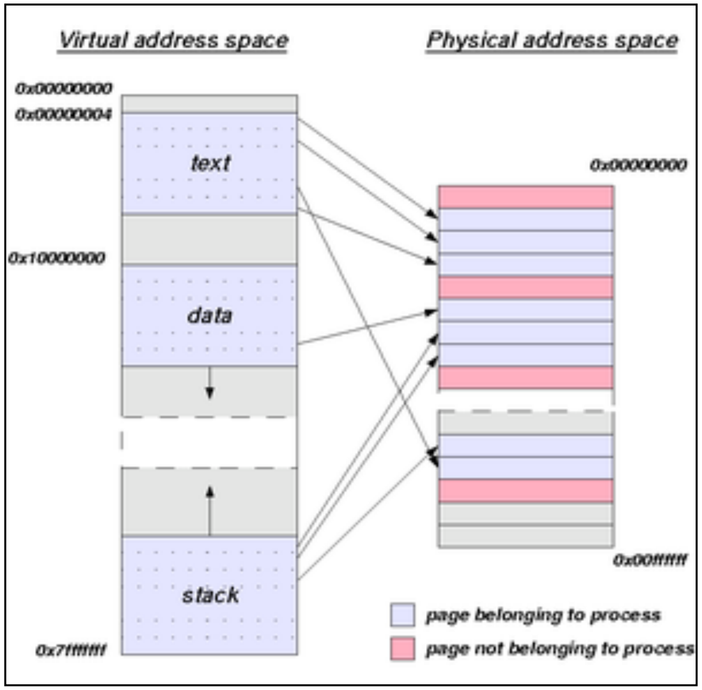
size of available memory

FIFO

- First-in, first-out: evict oldest page
- As competitive as LRU, but performs miserably in practice!
  - Ignores locality

Tricks with Page Tables: Sharing

- Paging allows sharing of memory across processes
  - Reduces memory requirements
- Shared stuff includes code, data
  - Code typically R/O



Mmaping in virtual address space

- Mapping files to virtual address space
  - Try and understand this through an example?
- You can also anonymous mmaping
  - Why would we want to do that?

Tricks with Page Tables: COW

- Copy on write (COW)
  - Just copy page tables
  - Make all pages read-only
- What if process changes mem?

- All processes are created this way!

In-class Discussion