Memory - Paging

Back to Paging

- On heavily-loaded systems, memory can fill up
- Need to make room for newly-accessed pages
  - Heuristic: try to move "inactive" pages out to disk
    - What constitutes an "inactive" page?
- Paging
  - Refers to moving individual pages out to disk (and back)
  - We often use the terms "paging" and "swapping" interchangeably
  - Different from context switching
    - Background processes often have their pages remain resident in memory

Demand Paging

- Never bring a page into primary memory until its needed
- Fetch Strategies
  - When should a page be brought into primary (main) memory from secondary (disk) storage.
- Placement Strategies
  - When a page is brought into primary storage, where should it be put?
- Replacement Strategies
  - Which page now in primary storage should be removed from primary storage when some other page or segment needs to be brought in and there is not enough room

Basic Page Replacement

- How do we replace pages?
  - Find the location of the desired page on disk
  - Find a free frame
    - If there is a free frame, use it
    - If there is no free frame, use a page replacement algorithm to select a *victim* frame
  - Read the desired page into the (newly) free frame. Update the page and frame tables.
  - Restart the process

Basic Page Replacement

- Note: can also evict in advance
  - OS keeps pool of "free pages" around, even when memory is tight
  - Makes allocating a new page fast
  - The process of evicting pages to disk is then performed in the background

Page Eviction: When?

- When do we decide to evict a page from memory?
  - Usually, at the same time that we are trying to allocate a new physical page
  - However, the OS keeps a pool of “free pages” around, even when memory is tight, so that allocating a new page can be done quickly
  - The process of evicting pages to disk is then performed in the background

Page Eviction: Which page?

- Hopefully, kick out a less-useful page
  - Dirty pages require writing, clean pages don’t
  - Where do you write? To “swap space”
- Goal: kick out the page that’s least useful
- Problem: how do you determine utility?
  - Heuristic: temporal locality exists
  - Kick out pages that aren’t likely to be used again

Exploiting Locality

- Temporal locality
  - Memory accessed recently tends to be accessed again soon
- Spatial locality
  - Memory locations near recently-accessed memory is likely to be referenced soon

Exploiting Locality

- Locality helps reduce the frequency of paging
  - Once something is in memory, it should be used many times
- This depends on many things
  - The amount of locality and reference patterns in a program
  - The page replacement algorithm
  - The amount of physical memory and the application footprint

Fundamental technique: Caching

- A cache keeps a subset of a data set in a more accessible but space-limited location

- Caches are everywhere in systems
  - Such as…?

Caching

- Caches are everywhere in systems
  - Registers are a cache for L1 cache which is a cache for L2 cache which is a cache for memory which is a cache for disk which might be a cache for a remote file server
  - Web proxy servers make downloads faster & cheaper
  - Web browser stores downloaded files
  - Local DNS servers remember recently-resolved DNS names
  - Google servers remember your searches
- Key goal: minimize cache miss rate
  - = minimize page fault rate (in context of paging)
  - Requires a good algorithm

Evicting the Best Page

- Goal of the page replacement algorithm
  - Reduce page fault rate by selecting the best page to evict
- The “best” pages are those that will never be used again
  - However, it's impossible to know in general whether a page will be touched
  - If you have information on future access patterns, it is possible to prove that evicting those pages that will be used the furthest in the future will minimize the page fault rate

Evicting the Best Page

- What is the best algorithm for deciding the order to evict pages?
  - Much attention has been paid to this problem
  - Used to be a very hot research topic
  - These days, widely considered solved (at least, solved well enough)

Page Replacement Strategies

- OPT
  - Evict page that won't be used for the longest time in the future
- Random page replacement
  - Choose a page randomly
- FIFO - First in First Out
  - Replace the page that has been in primary memory the longest
- LRU - Least Recently Used
  - Replace the page that has not been used for the longest time
- LFU - Least Frequently Used
  - Replace the page that is used least often
- NRU - Not Recently Used
  - An approximation to LRU.
- Working Set
  - Keep in memory those pages that the process is actively using.

Page Replacement Strategies

- The Optimal Algorithm
  - Among all pages in frames, evict the one that has its next access farthest into the future
  - Can prove formally this does better than any other algorithm
  - OPT is useful as a “yardstick” to compare the performance of other (implementable) algorithms against
  - Realistic?

The Optimal Page Replacement Algorithm

- Idea
  - Select the page that will not be needed for the longest time <u>in the future</u>

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						
Frames	1	b	b	b	b	b						
	2	c	c	c	c	c						
	3	d	d	d	d	d						
Page faults							x					

The Optimal Page Replacement Algorithm

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

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

The Optimal Page Replacement Algorithm

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

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

The Optimal Page Replacement Algorithm

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

FIFO Page Replacement Algorithm

- Always replace the oldest page
- 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	c	c	c	c						
	3	d			d	d						
Page faults							x					

FIFO Page Replacement Algorithm

- Always replace the oldest page
- 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		
	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

- Always replace the oldest page
- 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	
Frames	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

- Always replace the oldest page
- 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

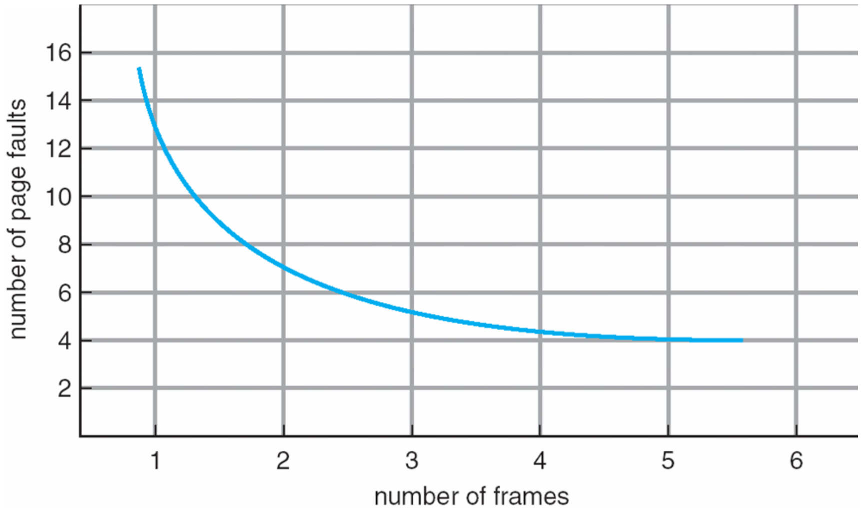
FIFO Page Replacement Algorithm

- Why might FIFO be good?
  - Maybe the page allocated very long ago isn't used anymore
- Why might FIFO not be so good?
  - Doesn't consider locality of reference!
  - The oldest page may be needed again soon
  - Some page may be important throughout execution

Belady's anomaly: Performance of an application might get worse as physical memory increases!!!

Belady's Anomaly

- Given a reference string, it would be natural to assume that
  - The more the total number of frames in main memory, the fewer the number of page faults



- Not true for some algorithms!
  - E.g., for FIFO

Belady's Anomaly

- Consider FIFO page replacement
  - Look at this reference string
    - 012301401234
  - Case 1:
    - 3 frames available
  - Case 2:
    - 4 frames available

Belady's Anomaly

FIFO with 3 page frames

		0	1	2	3	0	1	4	0	1	2	3	4
Youngest Page		0	1	2	3	0	1	4	4	4	2	3	3
			0	1	2	3	0	1	1	1	4	2	2
Oldest Page				0	1	2	3	0	0	0	1	4	4
		P	**P**	**P**	**P**	**P**	**P**	**P**			**P**	**P**	

Belady's Anomaly

FIFO with 3 page frames

		0	1	2	3	0	1	4	0	1	2	3	4
Youngest Page		0	1	2	3	0	1	4	4	4	2	3	3
			0	1	2	3	0	1	1	1	4	2	2
Oldest Page				0	1	2	3	0	0	0	1	4	4
		P	**P**	**P**	**P**	**P**	**P**	**P**			**P**	**P**	

FIFO with 4 page frames

		0	1	2	3	0	1	4	0	1	2	3	4
Youngest Page		0	1	2	3	3	3	4	0	1	2	3	4
			0	1	2	2	2	3	4	0	1	2	3
Oldest Page				0	1	1	1	2	3	4	0	1	2
					0	0	0	1	2	3	4	0	1
		P	**P**	**P**	**P**			**P**	**P**	**P**	**P**	**P**	**P**

Belady's Anomaly

FIFO with 3 page frames

		0	1	2	3	0	1	4	0	1	2	3	4
Youngest Page		0	1	2	3	0	1	4	4	4	2	3	3
			0	1	2					1	4	2	2
Oldest Page				0	1	2	3	0	0	0	1	4	4
		P	P	P	P	P	P	P			P	P	

9 page faults

FIFO with 4 page frames

		0	1	2	3	0	1	4	0	1	2	3	4
Youngest Page		0	1	2	3	3	3	4	0	1	2	3	4
			0	1	2					0	1	2	3
Oldest Page				0	1	1	1	2	3	4	0	1	2
					0	0	0	1	2	3	4	0	1
		P	P	P	P			P	P	P	P	P	P

10 page faults

Least Recently Used Algorithm (LRU)

- Keep track of when a page is used
- Replace the page that has been used least recently
  - Keep track of when pages are referenced to make a better decision
  - Use past behavior to predict future behavior
    - LRU uses past information
    - OPT uses future information
- Not optimal
- Does not suffer from Belady's anomaly

Least Recently Used Algorithm (LRU)

- Keep track of when a page is used
- Replace the page that has been used least recently (farthest in the past)

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

Page faults

Least Recently Used Algorithm (LRU)

- Keep track of when a page is used
- Replace the page that has been used least recently (farthest in the past)

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

Least Recently Used Algorithm (LRU)

- Keep track of when a page is used
- Replace the page that has been used least recently (farthest in the past)

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

Least Recently Used Algorithm (LRU)

- Keep track of when a page is used
- Replace the page that has been used least recently (farthest in the past)

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

Least Recently Used Algorithm (LRU)

- Keep track of when a page is used
- Replace the page that has been used least recently (<u>farthest in the past</u>)

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

- Implementation
  - Use time of last reference
    - Update every time page accessed (use system clock)
    - Page replacement - search for smallest time
  - Use a stack
    - On page access : remove from stack, push on top
    - Victim selection: select page at bottom of stack
- Problems or limitations?

Least Recently Used Issues

- Implementation
  - Use time of last reference
    - Update every time page accessed (use system clock)
    - Page replacement - search for smallest time
  - Use a stack
    - On page access : remove from stack, push on top
    - Victim selection: select page at bottom of stack
- Problems or limitations?
  - Both approaches require large processing overhead, more space, and hardware support
  - 32-bit timestamp would double size of PTE

Least Recently Used Issues

- 3 frames of physical memory
- Run this for a long time with LRU page replacement:

```
while true
    for (i = 0; i < 4; i++)
        read from page i
```

- Q1: What fraction of page accesses are faults?
  - None or almost none
  - About 1 in 4
  - About 2 in 4
  - About 3 in 4
  - All or almost all
- Q2: How well does OPT do?

LRU Approximation Algorithms

- Not used recently/Not recently used (NUR/NRU)
- Reference Bit in each page table entry
  - With each page, associate a bit, initially = 0
  - When page is referenced, bit is set to 1
  - Victim Selection
    - Any page with reference bit == 0 (if one exists, otherwise FIFO)
    - BUT: Do not know order

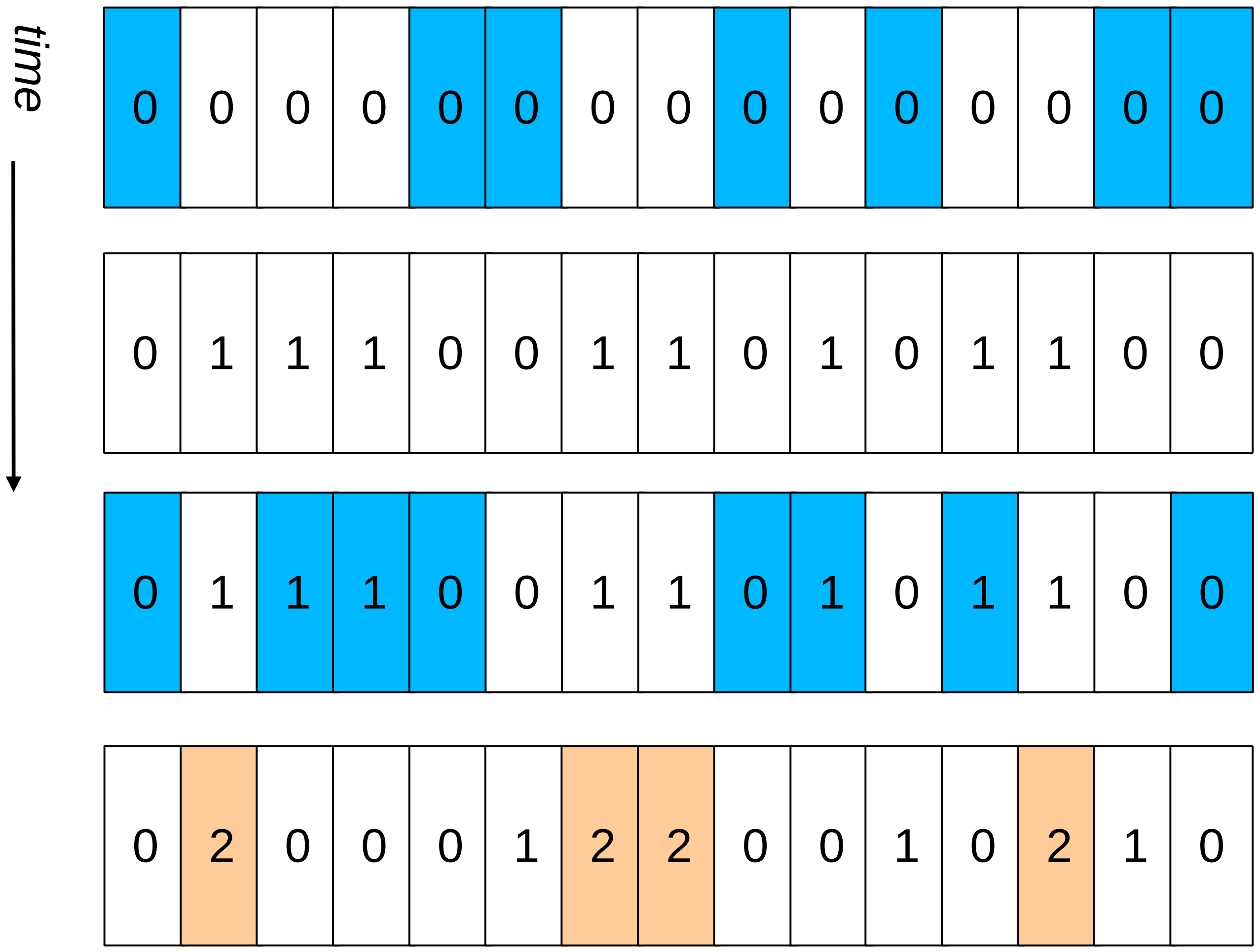
LRU Approximation Algorithms

- Additional Reference Bits Algorithm
  - Use the PTE reference bit and a small counter per page (2 or 3 bits in PTE)
  - Periodically (say every 100 msec), scan all physical pages. For each page:
    - If not accessed recently, (PTE reference bit == 0), `counter++`
    - If accessed recently (PTE reference bit == 1), `counter = 0`
    - Clear the PTE reference bit in either case!

LRU Approximation Algorithms

- Additional Reference Bits Algorithm
  - Counter will contain the number of scans since the last reference to this page
    - PTE that contains the highest counter value is the least recently used
    - So, evict the page with the highest counter

Approximate LRU



Accessed pages in blue

Increment counter for untouched pages

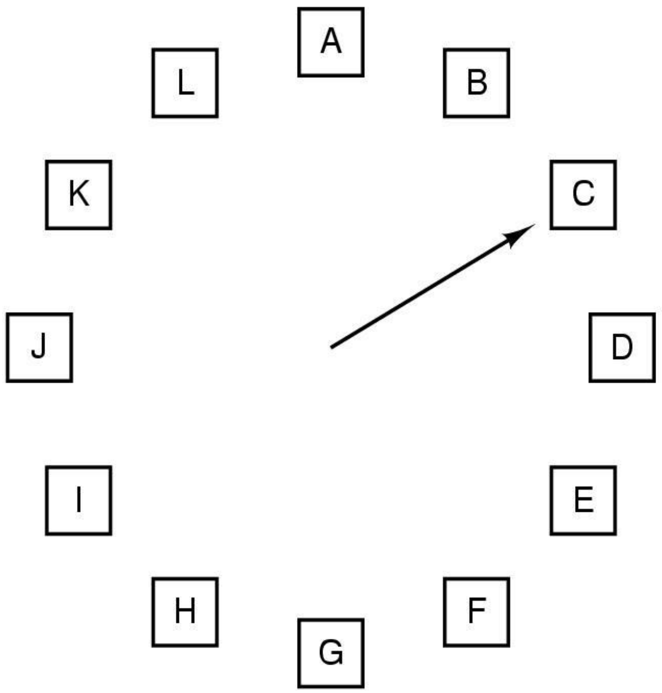
These pages have the highest counter value and can be evicted.

Second Chance Page Replacement

- Modification to FIFO
- Pages kept in a linked list
  - Oldest is at the front of the list
- Look at the oldest page
  - If referenced bit == 0
    - Select for replacement
  - Else
    - Page was recently used -> don't replace it
    - Clear referenced bit
    - Move to the end of list
- What if every page was used in last clock tick?
  - Select a page at random

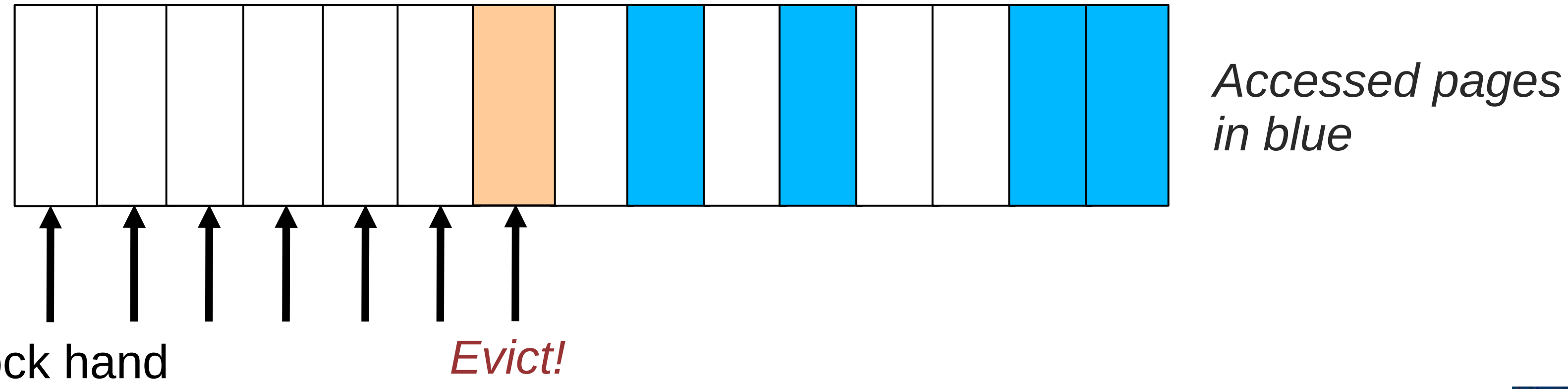
Clock Algorithm (Same effect as Second Chance)

- Maintain a circular list of pages in memory
- Set reference bit on access
- Clock sweeps over memory
  - Look for victim page with referenced bit unset
  - If bit is set, clear it and move on to next page
  - Replace pages that haven't been referenced for one complete clock revolution



Clock Algorithm (Same effect as Second Chance)

- "Clock hand" scans over all physical pages in the system
  - Clock hand loops around to beginning of memory when it gets to end
- If PTE reference bit == 1, clear bit and advance hand to give it a second-chance
- If PTE reference bit == 0, evict this page
  - No need for a counter in the PTE!


Accessed pages in blue
Clock hand
Evict!