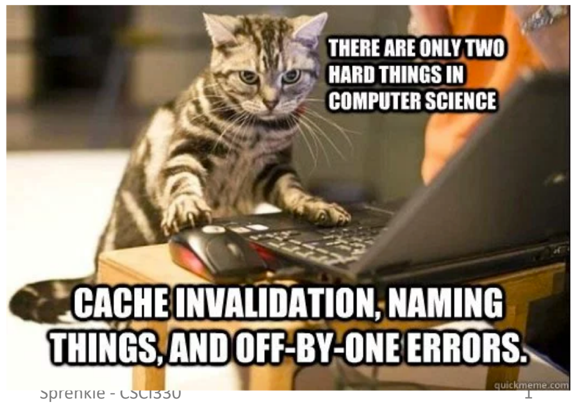
Today

- Memory Management
  - Page Replacement
  - VM Policies: Cleaning
- OS retrospective



Review

- What are page replacement algorithms?
  - What are their goals?
- Why do we need page replacement algorithms?
  - When is the algorithm triggered?

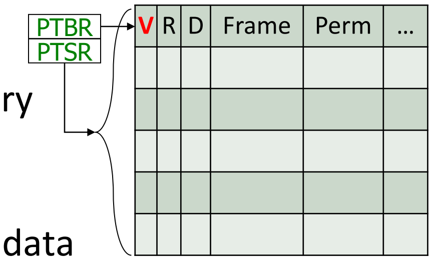
Review: Virtual Memory

- **Idea**: use physical memory to hold only the portions of each executing process that are currently being used
  - Only part of the program needs to be in memory for execution
  - Parts of executing process that are not currently being used are held on secondary storage until needed.
- **Impact**:
  - Logical address space can be much larger than physical address space
  - Allows address spaces to be shared by several processes
  - Less I/O needed to load or swap processes

Review: "Swapping" Pages to Disk

- Intuition: If a process isn't using a page, why keep it in physical memory? Instead, send it to disk and reclaim that space
- Illusion: memory size is physical memory + disk (with non-uniform access times)
- Supporting this idea requires:
  - Identifying where a chunk of memory is (physical memory or disk?)
  - Moving data between physical memory and disk (mechanism)
  - Algorithm for governing what gets moved to disk and what stays (policy)

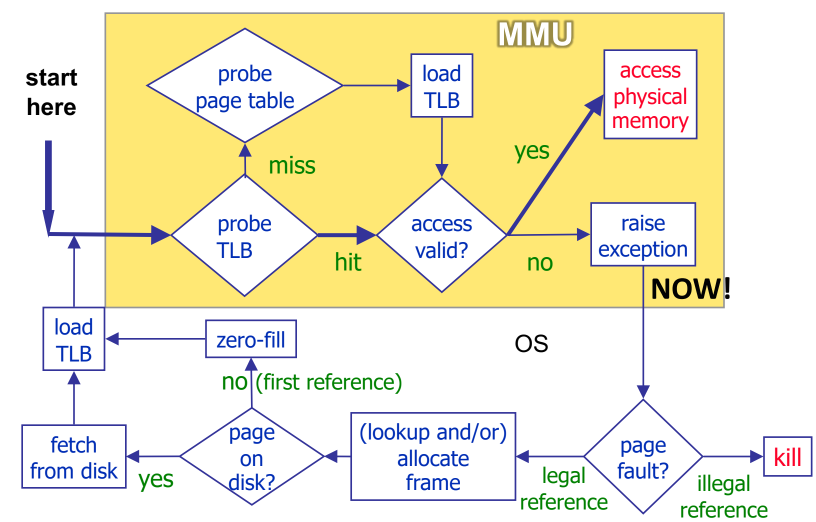
Review: Page Table: Revisited



- One table per process
- Table parameters in memory
  - Page table base register
  - Page table size register
- Table elements: Page metadata
  - **V: valid bit**
  - R: referenced bit
  - D: dirty bit
    - If page has been moc
  - Frame: location in phy
  - Perm: access permissions

Valid bit, checkable by hardware, says if the page is in physical memory:
- 1: in memory, use frame field to find where
- 0: not in memory

Review: Virtual Addressing



Demand Paging: bring page into memory (only) when requested

Review: Page Faults are Expensive

- Disk: 5-6 orders magnitude slower than RAM
  - Very expensive; but if very rare, tolerable
- Example
  - RAM access time: 100 **n**sec
  - Disk access time: 10 **m**sec
  - *p* = page-fault probability
  - Effective access time: 100 + *p* × 10,000,000 nsec
  - If *p* = 0.1%, effective access time = 10,100 nsec !

Analogy: Most of the time, to get what you need, you walk to the Commons.

Occasionally, you have to walk to Seattle.

We need to be smart about what we send to disk.
Goal: minimize the slowdown.

Review: Policy Decisions for Virtual Memory

- Placement: Where should we put items in physical memory?
  - Irrelevant for page-based systems
  - Any frame is equally good
- Replacement: Which page should we evict from memory to disk?
  - Which page do we pick?
  - Local vs global: Which process should the page come from?
- Cleaning: for modified (dirty) pages, when to write them to disk?

Review: Page Replacement Goals

1. Minimize page faults
   - Achieve good **temporal locality**: reuse of pages within a short period of time
   - (Spatial locality: use of close data elements)

2. Easy to implement and low overhead to manage
   - Don't need a lot of state
   - Better if HW can handle most of requests

Review: Page Replacement Algorithms

- Can't know the future of page accesses…
  - BUT Bélády's Optimal Algorithm – good for comparisons
- Straightforward algorithm: FIFO
  - Always replace the oldest page
  - BUT bad locality
- Classic cache replacement algorithm: LRU
  - Replace the page that hasn't been used for the longest time

LRU Replacement

New page sequence!

Pages Accessed: 2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2

First four pages: fill in free frames.

F_0	2*	2	2	2								
F_1		3*	3	3								
F_2				1*								

* Indicates page fault

LRU Replacement

Pages Accessed: 2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2

F_0	2*	2	2	2	2							
F_1		3*	3	3	5*							
F_2				1*	1							

* Indicates page fault

LRU Replacement

Pages Accessed: 2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2

F_0	**2***	2	**2**	2	2	**2**						
F_1		**3***	3	3	**5***	5						
F_2				**1***	1	1						

* Indicates page fault

LRU Replacement

Pages Accessed: 2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2

F_0	**2***	2	**2**	2	2	**2**	2					
F_1		**3***	3	3	**5***	5	5					
F_2				**1***	1	1	4*					

* Indicates page fault

LRU Replacement

Pages Accessed: 2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2

F_0	**2***	2	**2**	2	2	**2**	2	2				
F_1		**3***	3	3	**5***	5	5	5				
F_2				**1***	1	1	**4***	4				

* Indicates page fault

LRU Replacement

Pages Accessed: 2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2

F_0	**2***	2	**2**	2	2	**2**	2	2	3*			
F_1		**3***	3	3	**5***	5	5	**5**	5			
F_2				**1***	1	1	**4***	4	4			

* Indicates page fault

LRU Replacement

Pages Accessed: 2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2

F_0	**2***	2	**2**	2	2	**2**	2	2	**3***	3		
F_1		**3***	3	3	**5***	5	5	**5**	5	5		
F_2				**1***	1	1	**4***	4	4	2*		

* Indicates page fault

LRU Replacement

Pages Accessed: 2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2

F_0	**2***	2	**2**	2	2	**2**	2	2	**3***	3	3	3
F_1		**3***	3	3	**5***	5	5	**5**	5	5	**5**	5
F_2				**1***	1	1	**4***	4	4	**2***	2	**2**

* Indicates page fault

Total: 7 Page faults.

LRU Replacement

Pages Accessed: 2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2

F_0	**2***	2	**2**	2	2	**2**	2	2	**3***	3	3	3
F_1		**3***	3	3	**5***	5	5	**5**	5	5	**5**	5
F_2				**1***	1	1	**4***	4	4	**2***	2	**2**

* Indicates page fault

Total: 7 Page faults

For this sequence,
- FIFO faults 9 times
- Optimal faults 7 times

Analyze LRU

- Recall our goals:
  - Minimize page faults
  - Easy to implement, low overhead to manage

✓ Better locality → fewer page faults
- A lot of bookkeeping
  - Look backwards to figure out access

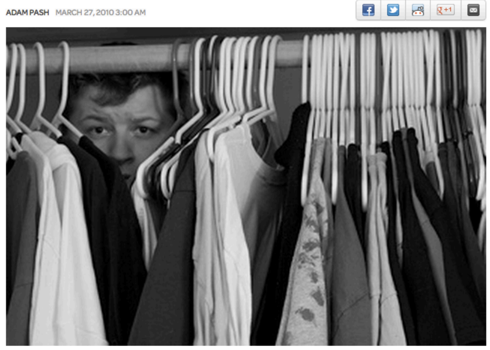
Implementing LRU for Page Replacement

- Take advantage of MMU hardware for performance
  - Avoid switching to OS execution on *every memory access*
- For each memory access, MMU must update LRU information
- Option 1: Timestamp the page
  - Problem: lots of time lookups, lots of bits to store time in each page table row
- Option 2: Rearrange queue/list containing order of page accesses
  - Problem: now we have hardware chasing pointers?

An analogy: Replacement for your closet

Weed Out The Clothes You Don't Wear With A Simple Hanger Trick

ADAM PASH MARCH 27, 2010 3:00 AM

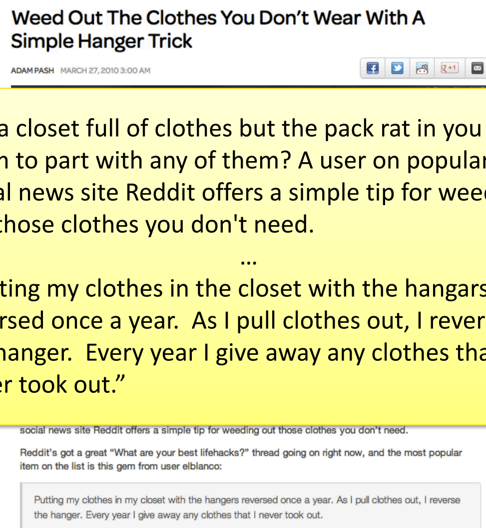


Got a closet full of clothes but the pack rat in you can't seem to part with any of them? A user on popular social news site Reddit offers a simple tip for weeding out those clothes you don't need.

Reddit's got a great "What are your best lifehacks?" thread going on right now, and the most popular item on the list is this gem from user elblanco:

> Putting my clothes in my closet with the hangers reversed once a year. As I pull clothes out, I reverse the hanger. Every year I give away any clothes that I never took out.

An analogy: Replacement for your closet


Weed Out The Clothes You Don't Wear With A Simple Hanger Trick

ADAM PASH MARCH 27, 2010 3:00 AM

social news site Reddit offers a simple tip for weeding out those clothes you don't need.

Reddit's got a great "What are your best lifehacks?" thread going on right now, and the most popular item on the list is this gem from user elblanco:

Putting my clothes in my closet with the hangers reversed once a year. As I pull clothes out, I reverse the hanger. Every year I give away any clothes that I never took out.

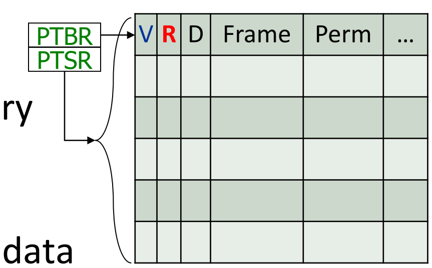
Got a closet full of clothes but the pack rat in you can't seem to part with any of them? A user on popular social news site Reddit offers a simple tip for weeding out those clothes you don't need.

...

"Putting my clothes in the closet with the hangars reversed once a year. As I pull clothes out, I reverse the hanger. Every year I give away any clothes that I never took out."

Page Table: Re-Revisited

- One table per process
- Table parameters in memory
  - Page table base register
  - Page table size register
- Table elements: Page metadata
  - V: valid bit
  - **R: referenced bit**
  - D: dirty bit
    - If page has been m
  - Frame: location in ph
  - Perm: access permissions



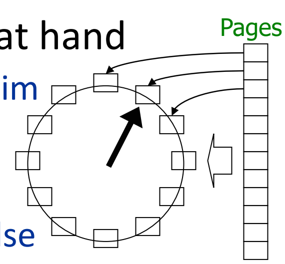
- Use this ONE BIT to approximate LRU
- Easy to do in MMU hardware: When access a page, MMU sets the bit to 1
- Intuition: has this page been used recently?

Approximating LRU: Clock Algorithm

- Select page that is old and not recently used
  - Clock (a.k.a. "second chance") is approximation of LRU
- Hardware support: reference bit
  - Associated with each page
  - MMU sets on page access
  - "Have you been accessed since the last time I looked for a victim?"
- On page fault, look through the pages in numerical order (starting from where we left off during last fault)
  - If page is referenced recently (ref bit set), unset the reference bit
  - If page is not referenced recently, evict it

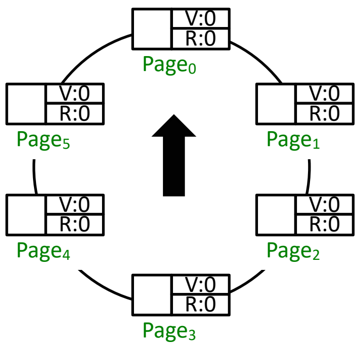
Clock Algorithm Model

- Arrange all pages in circle (like a… clock)
- Clock "hand": next page to consider
  - Skip over invalid entries, they have no frame
- Page fault: scan forward, starting at hand
  - if reference bit 0, select page as victim
  - otherwise, set reference bit to 0
  - advance clock hand to next page
  - if victim found, break out of loop (else repeat)
- Hand position preserved across faults



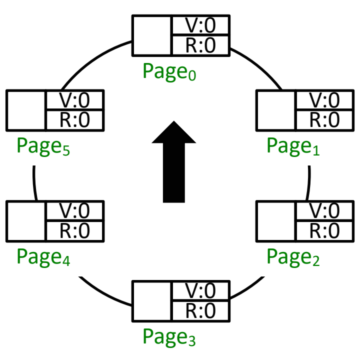
Clock Example

- Pages accessed: 2, 4, 5, 1, 4, 2, 4
  Free frames: 0, 1, 2



Clock Example

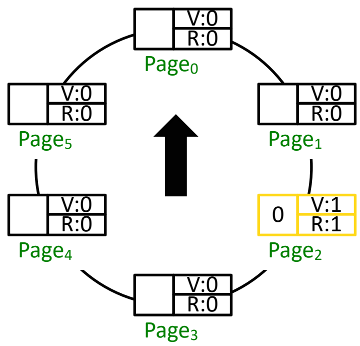
- Pages accessed: 2, 4, 5, 1, 4, 2, 4
  Free frames: 0, 1, 2



- First, use free frames

Clock Example

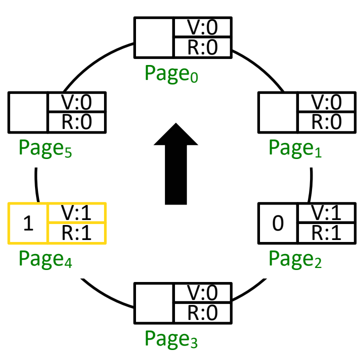
- Pages accessed: 2, 4, 5, 1, 4, 2, 4
  Free frames: ~~0~~, 1, 2



- First, use free frames
- Set valid bit because page has a frame
- Set reference bit because page was accessed
- Access page 2: page fault (unavoidable)
  - We have a free frame, use it!

Clock Example

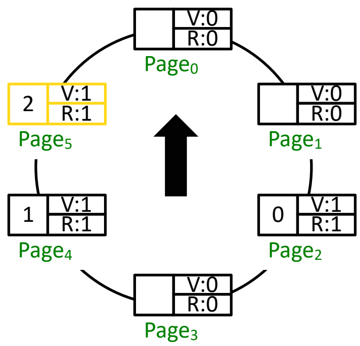
- Pages accessed: 2, 4, 5, 1, 4, 2, 4
  Free frames: ~~0~~, ~~1~~, 2



- First, use free frames
- Set valid bit because page has a frame
- Set reference bit because page was accessed
- Access page 4: page fault (unavoidable)
  - We have a free frame, use it!

Clock Example

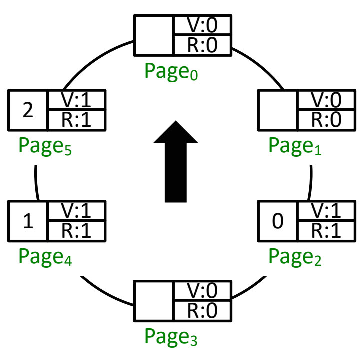
- Pages accessed: 2, 4, 5, 1, 4, 2, 4
  Free frames: ~~0~~, ~~1~~, ~~2~~



- First, use free frames
- Set valid bit because page has a frame
- Set reference bit because page was accessed
- Access page 5: page fault (unavoidable)
  - We have a free frame, use it!

Clock Example

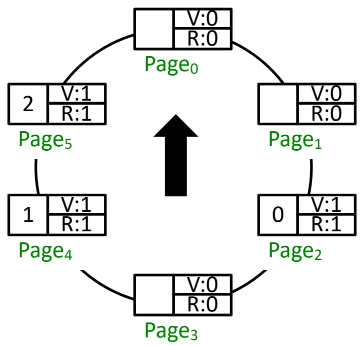
- Pages accessed: 2, 4, 5, 1, 4, 2, 4
  Free frames: ~~0~~, ~~1~~, ~~2~~



- Access page 1: page fault.
- We have no free frames, so one of the pages with a frame needs to be *evicted*.

Clock Example

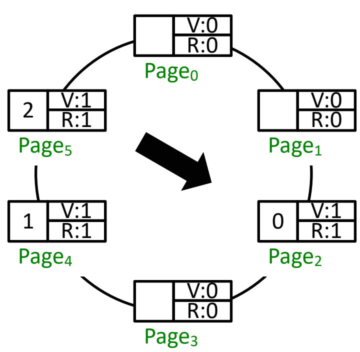
- Pages accessed: 2, 4, 5, <u>1</u>, 4, 2, 4
  Free frames: ~~0~~, ~~1~~, ~~2~~



- Which page should we *evict* for page 1, according to the clock algorithm?
  a) 2
  b) 4
  c) 5

Clock Example

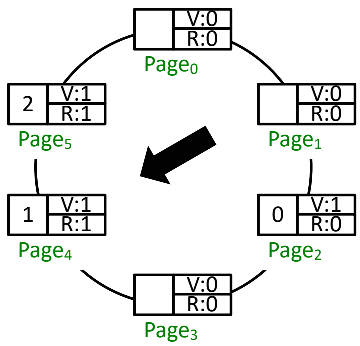
- Pages accessed: 2, 4, 5, <u>1</u>, 4, 2, 4
  Free frames: ~~0~~, ~~1~~, ~~2~~



- Ignore any invalid page entries, they don't have frames, so we can't evict them

Clock Example

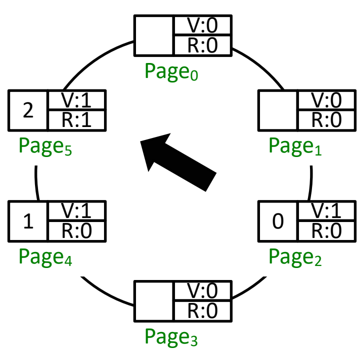
- Pages accessed: 2, 4, 5, <u>1</u>, 4, 2, 4
  Free frames: ~~0~~, ~~1~~, ~~2~~



- Hand points to a referenced page
- Set ref bit to 0, advance hand, try again

Clock Example

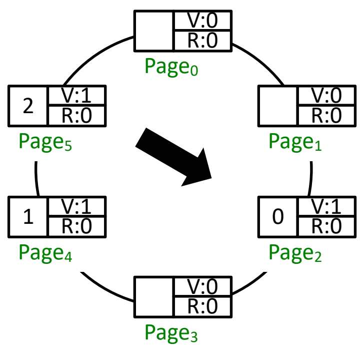
- Pages accessed: 2, 4, 5, <u>1</u>, 4, 2, 4
  Free frames: ~~0~~, ~~1~~, ~~2~~



- Hand points to a referenced page
- Set ref bit to 0, advance hand, try again

Clock Example

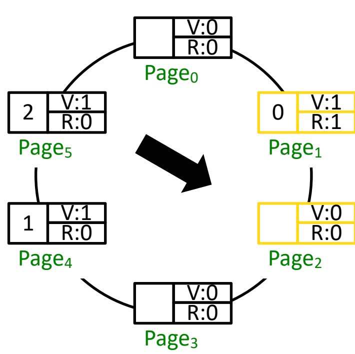
- Pages accessed: 2, 4, 5, <u>1</u>, 4, 2, 4
  Free frames: ~~0~~, ~~1~~, ~~2~~



- Hand points to a page with ref bit 0!
  - We've found our *victim*

Clock Example

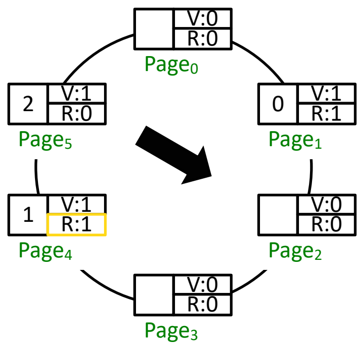
- Pages accessed: 2, 4, 5, <u>1</u>, 4, 2, 4
  Free frames: ~~0~~, ~~1~~, ~~2~~



- Hand points to a page with ref bit 0!
  - We've found our *victim*
- Evict page 2 (mark invalid)
- Assign its old frame (frame 0) to the faulting page (page 1)

Clock Example

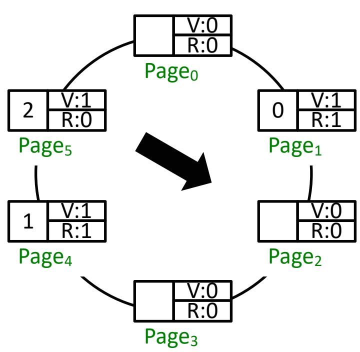
- Pages accessed: 2, 4, 5, 1, 4, 2, 4
  Free frames: ~~0~~, ~~1~~, ~~2~~



- Access page 4
  - Page 4 is already in memory: no fault
  - OS does nothing!
  - MMU hardware sets ref bit to 1

Clock Example

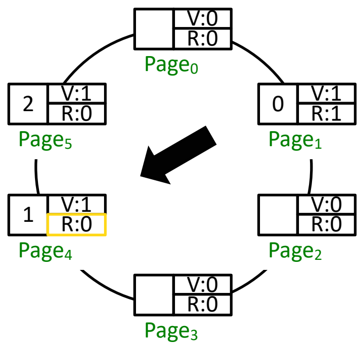
- Pages accessed: 2, 4, 5, 1, 4, 2, 4
  Free frames: ~~0~~, ~~1~~, ~~2~~



- Access page 2: page fault.
- We have no free frames, so one of the pages with a frame needs to be *evicted.*
- *Which page will it be?*

Clock Example

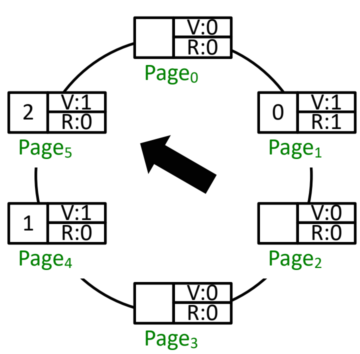
- Pages accessed: 2, 4, 5, 1, 4, <u>2</u>, 4
  Free frames: ~~0~~, ~~1~~, ~~2~~



- Skip the invalid pages
- Hand points to a referenced page.
- Set ref bit to 0, advance hand, try again.

Clock Example

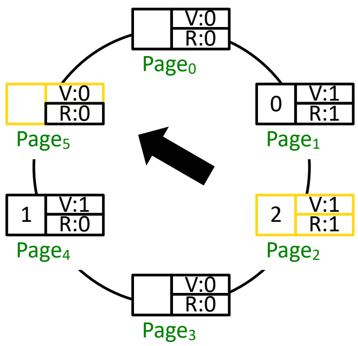
- Pages accessed: 2, 4, 5, 1, 4, <u>2</u>, 4
  Free frames: ~~0~~, ~~1~~, ~~2~~



- Hand points to a page with ref bit 0!
  - We've found our victim

Clock Example

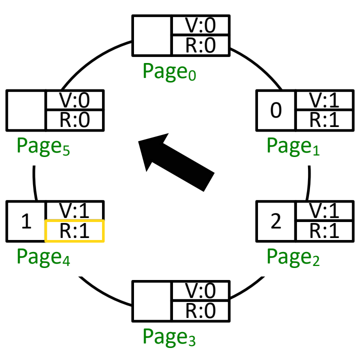
- Pages accessed: 2, 4, 5, 1, 4, 2, 4
  Free frames: ~~0~~, ~~1~~, ~~2~~



- Hand points to a page with ref bit 0!
  - We've found our victim
- Evict page 5 (mark invalid)
- Assign its old frame (frame 2) to the faulting page (page 2)

Clock Example

- Pages accessed: 2, 4, 5, 1, 4, 2, 4
  Free frames: ~~0~~, ~~1~~, ~~2~~

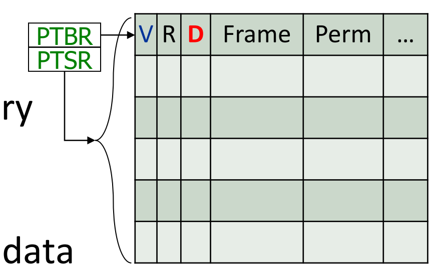


- Access page 4
  - Page 4 is already in memory: no fault
  - OS does nothing!
  - MMU hardware sets ref bit to 1

Policy Decisions for Virtual Memory

- Placement: Where should we put items in physical memory.
  - Irrelevant for page-based systems. Any frame is equally good.
- Replacement: Which page should we evict from memory to disk?
  - Which page do we pick?
  - Local vs global: Which process should the page come from?
- **Cleaning: for modified (dirty) pages, when to write them to disk?**

Page Table: Re-Revisited



- One table per process
- Table parameters in memory
  - Page table base register
  - Page table size register
- Table elements: Page metadata
  - V: valid bit
  - R: referenced bit
  - **D: dirty bit**
    - If page has been modified
  - Frame: location in physical memory
  - Perm: access permissions

Has this page been modified?
If so, it **no longer matches** the contents on disk.

When evicting a page…

- If there are no free frames, we must evict a page
  - If an identical copy of victim page is on disk, no write necessary!
    - Dirty bit not set
  - BUT, if victim page is dirty (or not on disk at all), must write it to disk first
- Problem? Not for correctness, but this isn't great for performance
  - Not only do we need to read a page from disk, now we have to write one too!
  - Double the disk latency…

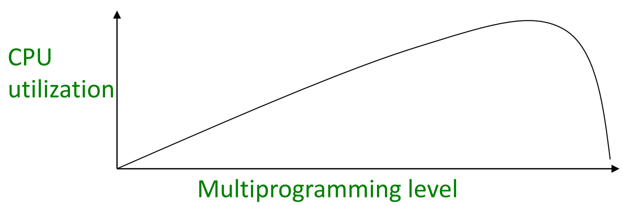
Paging Daemon

- "Daemon": system background process
  - see Wikipedia for etymology
- Paging daemon: if the system has spare CPU cycles, check memory
  - If it looks like a page is likely to be swapped to disk soon, write it to disk now!
  - (e.g., page wasn't referenced recently, clock hand near its entry)
- Intuition: keep a small reserve of free frames, do writes in advance of eviction.

Recall old assumption…

- For now, assume one process and that it has a fixed number of frames
- Reality: multiprogramming! Lots of processes available. They all need memory.
- How do we decide how much memory to give each one?

Multiprogramming



- Having more processes to choose from keeps CPU busy
- TOO many processes causes us to spend all our time shuffling data to/from disk: ***thrashing***

Assigning Frames to Processes…

- Local replacement
  - Give each process the same amount of memory (# of frames)
  - Perform page replacement among a process's own frames
- Global replacement
  - Allow each process to have varying amounts of memory (# of frames)
  - Perform page replacement among *all* frames in the system

Assigning Frames to Processes Tradeoffs

- Local replacement:
  - Fair to all processes – they all get equal memory
  - BUT, some processes are MUCH larger than others
  - What size do we choose?
- Global replacement:
  - Better reflects diversity in process memory needs
  - BUT, processes are now competing with one another
    - one bad process might gobble up all the memory and ruin everything

Hybrid Approach

- Processes don't directly take pages from one another (like local)
- OS examines processes to see if they're using the memory they've been given
  - If not, ***reclaim*** some for others (like global)
- Idealized solution: Denning's "working set"
- More realistic: Page fault frequency

Working Set

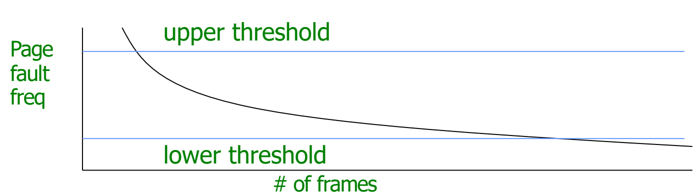
- Intuition: the set of pages a process is *actually using* right now
  - so that we know the number of frames it needs to store them
- Definition: the number of pages referenced in the interval (t, t - w)
- Few (no?) commercial systems track working set precisely due to cost of doing so
  - BUT an important theoretical concept

Working Set: Challenges

- Must timestamp pages in working set to identify set size
- Must determine time interval w

- Bottom line: working set is interesting as an abstraction, but not reasonable to build

Page Fault Frequency



- If fault frequency too high, working set not present
  - Give process more frames
- If fault frequency too low, resident set too large
  - Take away frames

Summary

- Virtual memory effectively extends main memory by swapping to disk
- Disk is slow and must be used judiciously
- Selecting which pages to swap (page replacement) is a challenging problem
- Real systems typically implement approximations of idealized policies (e.g., clock vs. LRU)

OS RETROSPECTIVE

Why Study Operating Systems?

- Understanding the OS helps you write better code
  - Learn how to manage complexity through appropriate abstractions
- Understand a wide range of system designs and tradeoffs of those designs
  - Performance vs. simplicity, HW vs. SW, etc
    - What should be in the hardware? In the OS? In the user applications?
      - What are the tradeoffs of these decisions?
  - Design tradeoffs made in the past do not necessarily apply now
  - Those made now will not necessarily apply in the future
- Operating Systems are everywhere!

Course Objectives

- to demystify the interactions between the software you have written in other courses and hardware,
- to familiarize you with the issues involved in the design and implementation of modern operating systems,
- and to explain the more general systems principles that are used in the design of **all** computer systems

Student Learning Objectives

- Describe the importance of abstraction in modern systems
- Differentiate between policy and mechanism
- Explain how operating systems manage concurrent processes including the complete life-cycle of user processes, threads, process synchronization, and deadlock avoidance
- Evaluate the suitability of algorithms used for process scheduling, memory allocation, and disk access for various use cases
- Understand how operating systems manage physical and virtual memory including segmentation and paging
- Develop programs that emulate or interact with operating system code

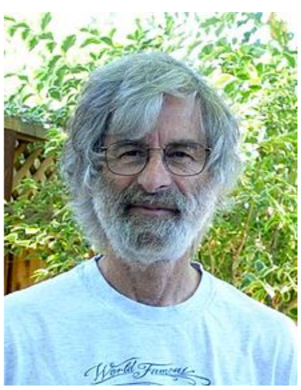
Topics We Could Have Covered

- Storage
  - Disk allocation, caching
  - NFS (distributed file systems)
- Memory management
  - More policies (prefetching, freeing)
- I/O
- Security

WHERE DO WE GO FROM HERE?

Dec 11, 2015 Sprenkle - CSCI330 65

What is a distributed system?



Leslie Lamport
2013 Turing Award Winner

"A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable." -- Leslie Lamport

Dec 9, 2015 Sprenkle - CSCI330 66

Looking Ahead

- Final Exam
  - Take Home Question – typed, PDF
    - 20% of final exam
  - In-class portion
- Evaluations – due Sunday
  - Add EC points to OS project grade, worth 50% of course grade
- Project due today
- Office hours
  - Monday and Tuesday afternoon and by appointment