PRINCIPLES OF COMPUTER SYSTEMS DESIGN

CSE130

Winter 2020

Memory Management V - Frame Allocation

Baskin Engineering
UC SANTA CRUZ

Notices

- **Lab 3** due **Sunday March 1**
- **Assignment 4** due **TODAY**

Today's Lecture

- Page vs. Frame
- Bélády's Anomaly
- Page Replacement Algorithms
- Frame Allocation Algorithms
- Lab 3 - System Calls Secret Sauce 

Naming Confusions ☹

- **Page Fault**
  - In the TLB:
    - A Page Table entry was not found
  - In the Page Table:
    - A Virtual Page does not currently map to a Physical Page
- **Page v.s. Frame**
  - Virtual Page = Page
  - Physical Page = Frame (really confusingly, also known as a "Page Frame")
- **Important Algorithms**
  - Page Replacement
  - Frame Allocation

Memory Management

- Process memory requirements tend to grow and grow
  - Each page fault requires the allocation of a frame
  - Eventually we run out of frames ☹
- **When / How do we reclaim memory?**
  - **Process terminates**
    - All frames reclaimed
  - **Swap entire processes out**
    - Some or all frames reclaimed
  - **Swap out some pages from an active process**
    - Some frames reclaimed

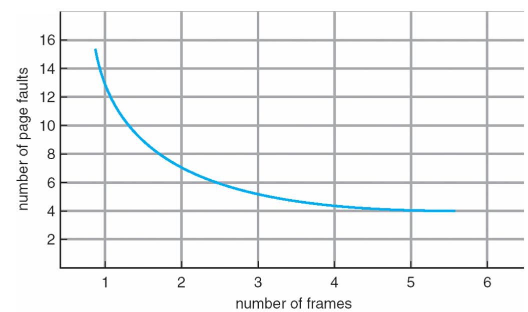
Page Replacement

- Prevent **over-allocation** of memory by modifying page-fault service routine to include page replacement
- Use **modified** (dirty) **bit** in page table entry to reduce overhead of page transfers - only modified pages are written to disk
- Page replacement completes the separation between logical memory and physical memory - **a large virtual memory space can be provided on a much smaller physical memory**

Page Replacement & Frame Allocation

- **Page-Replacement** algorithm
  - Wants lowest page-fault rate both on first access and subsequent re-access
  - Decides which pages to replace
- **Frame-Allocation** algorithm
  - How many frames should we give each process?
  - Which frames should be replaced?
- We evaluate algorithms by running them on a **particular sequence of memory references** and computing the number of page faults
  - Sequence is just page numbers, not full addresses
  - Repeated access to the same page does not cause a page fault

Page Faults vs. Available Frames



Page Replacement Algorithm : FIFO

- Simple **First-In-First-Out** Queue
  - Once queue is full, page at the front of the queue is evicted on page fault
  - New pages are added at the back of the queue

Page Request	7	0	1	2	0	3	0	4
	7	7	7	**2**	2	2	2	**4**
		0	0	0	0	**3**	3	3
			1	1	1	1	**0**	0
Queue	**7**	**0,7**	**1,0,7**	**2,1,0**	**2,1,0**	**3,2,1**	**0,3,2**	**4,0,3**
Page Faults	1	2	3	4	4	5	6	7

Bélády's Anomaly

Page Request	**3**	**2**	**1**	**0**	**3**	**2**	**4**	**3**	**2**	**1**	**0**	**4**
	3	3	3	**0**	0	0	**4**	4	4	4	4	4
		2	2	2	**3**	3	3	3	3	**1**	1	1
			1	1	1	**2**	2	2	2	2	**0**	0
Page Faults	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**7**	**7**	**8**	**9**	**9**

	3	3	3	3	3	3	**4**	4	4	4	**0**	0
		2	2	2	2	2	2	**3**	3	3	3	**4**
			1	1	1	1	1	1	**2**	2	2	2
				0	0	0	0	0	0	**1**	1	1
Page Faults	**1**	**2**	**3**	**4**	**4**	**4**	**5**	**6**	**7**	**8**	**9**	**10!**

Adding more frames does not necessarily result in fewer page faults ☹

Page Replacement Algorithm : OPT

- **Optimal Page Replacement**
  - Replace page that will next be used **furthest in the future**
  - Currently impossible to implement ☹
  - But can give us a theoretical benchmark against which to assess alternatives

Page Replacement Algorithms : LRU & LFU

- **Least Recently Used**
  - Use past rather than future knowledge
  - Replace page that has not been used for the longest time
  - Associate **time of last use** with each page
- **Least Frequently Used**
  - Like LRU, use past knowledge
  - Replace page that has been used least ***over a given time period***
  - Associate **usage count** with each page

Stack Based Page Replacement Algorithms

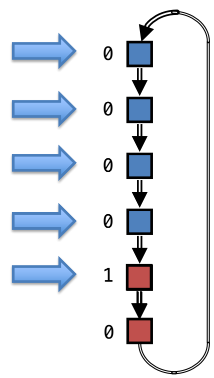
- LRU, LFU, and OPT are stack algorithms ***without*** Bélády's Anomaly
- **Stack Algorithms**
  - Keep a stack of page numbers in a double linked list
  - When page referenced:
    - Move it to the top of the stack (head of the list)
    - 6 pointers need to be changed (inexpensive)
  - But each update more expensive
  - No search for replacement

LRU Approximations

- **Problem**:
  - LRU tends to need hardware assistance and can still be slow
- **Solution:**
  - Add a reference bit to each page
    - Initial value = 0
  - When page is referenced bit set to 1
  - Replace any pages where reference bit = 0 (we're obviously not using them)
  - **Problems?**
    - We **do not know the order** in which the referenced pages were accessed
    - What happens when all pages have been referenced?

LRU Approximations

- **Second Chance**
  - Extends FIFO
  - Each page gets <u>two</u> chances
  - On second access, page goes to back of queue
  - Frequently referenced pages don't get replaced quite so often
- **Circular Queue**
  - Pointer indicates next page to replace
  - Advance until page with 0 reference bit found
  - Clear reference bits as you go
  - Replace victim with new page and set reference bit to 1
  - Reference bit also set to 1 when accessed



See class video for animation

Counting Page Replacement Algorithms

- Every page table entry has a **counter**; every time page is referenced through this entry, the counter is incremented
- When a page needs to be replaced, examine the counters to find **smallest value**
- But… linear search through table required ☹
- ***Least Frequently Updated***
  - Replaces page with the smallest count
- ***Most Frequently Updated***
  - Replaces page with the largest count

Clock Based Page Replacement Algorithms

- Every page entry has a **timestamp**
- Every time page is referenced, copy the clock into the timestamp
- When a page needs to be replaced, look at the timestamps to find **smallest value**
- But… linear search through table required ☹

Page Buffering Algorithms

- Idea: **Keep a pool of free frames**
  - A frame is “always” available when needed, not searched for at fault time
  - Read page into free frame and select victim to evict
  - When convenient, evict victim and add recovered frame to free pool
- Enhancement: **Keep list of modified pages**
  - When backing store otherwise idle, write pages there and set to non-dirty
- Enhancement: **Keep free frame contents intact**
  - If referenced again before frame is reused, no need to re-load contents from disk
    - Generally useful to reduce penalty if wrong victim frame selected

Frame Allocation

- Each process needs a **minimum number of frames**
  - Consider our pseudo machine instruction `ADD C,A,B`
    - Adds the integer at address A to the integer at address B, saves the result at address C
  - Lets say the instruction itself is 6 bytes and could span 2 pages
  - Possibly need 2 pages to handle each of A, B, and C
  - Possible 8 pages required, so allocate 8 frames
- Maximum is total number of frames in the system
- Two major allocation schemes
  - **Fixed allocation**
  - **Dynamic allocation**
- Many variations

Frame Allocation

- **Equal allocation**
  - E.g. If there are 93 frames (after allocating frames for the OS) and 5 processes, give each process 16 frames
    - i.e. keep some as a free-frame pool
- **Proportional Allocation (by size)**
  - Allocate according to the size of each process
  - Dynamic as the degree of multiprogramming and process sizes change
- **Proportional Allocation (by priority)**
  - If process **P** generates a page fault,
    - Instead of selecting one of **P**’s pages for replacement
    - Select a page from process with lower priority and allocate the victim’s freshly de-allocated frame to **P** (**P** ☺ but lower priority process ☹)
    - **Is this a reasonable idea?**
    - **On SMP and/or multi-core systems, how might you modify the algorithm?**

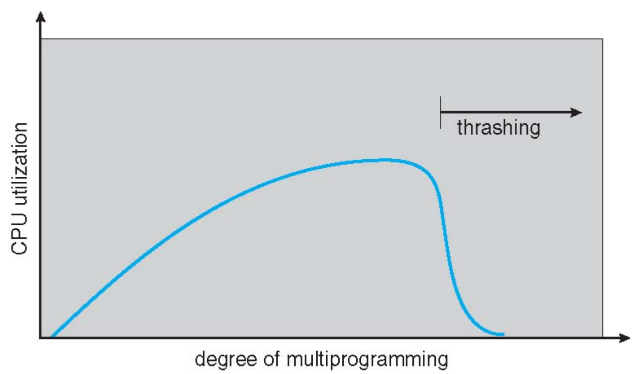
Global vs. Local Allocation

- **Global Allocation**
  - Process selects a replacement frame from the set of all frames
  - One process can take a frame from any another
    - Remember, memory is fundamentally preemptable
  - But process execution time can vary greatly
- **Local Allocation**
  - Each process selects from only its own set of allocated frames
  - More consistent per-process performance
  - But possibly underutilized memory
- Global allocation leads to better throughput than local allocation at the expense of unpredictable performance for individual processes

Thrashing Revisited

- When a process accesses a physical page that was swapped out, the page is swapped back in to physical memory
  - May cause another physical page to be swapped out
  - If this happens too rapidly, it's known as **THRASHING**
  - Extreme performance degradation ☹
- If a process does not have "enough" pages, the page-fault rate can be very high indeed
  - Page fault raised to get desired page
  - Replaces existing page
  - But quickly need replaced page back again
  - Leads to Low CPU utilization…
    - Operating system thinks that it needs to increase the degree of multiprogramming
    - Another process gets added to the system with negative results ☹

Thrashing

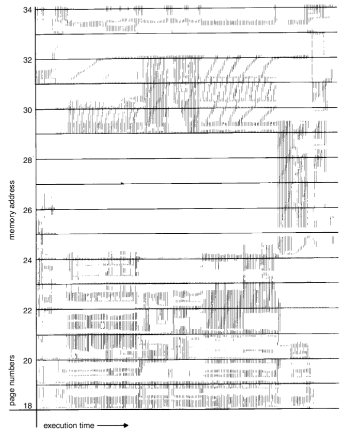


Preventing Thrashing

- **Problem:**
  - Processes stealing frames from one another leads to thrashing ☹
- **Solution:**
  - Only allow processes to thrash themselves
  - Still affects other processes through contention for disk ☹
- **Better Solution?**
  - Observe that most processes access clustered sets of pages at a time
  - Use this to work out optimal per-process allocation

Locality Model

- A set of frames in frequent use by a process is a known as a **locality**
- A process usually consists of a number of distinct localities
- As a process executes they tend to move from locality to locality
- Localities may overlap



Working Set Model

- Dynamically adjusts frames allocated during the previous τ
- $\mathbf{W_t(\tau)}$ = set of pages referenced in $\mathbf{(t-\tau,t)}$
- $\mathbf{w_t(\tau)} = \mathbf{|\ W_t t)\ |}$ = working set size at time t

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

τ τ

$w_t(\tau) = \{1,5,6,7\}$ $w_t(\tau) = \{3,4\}$

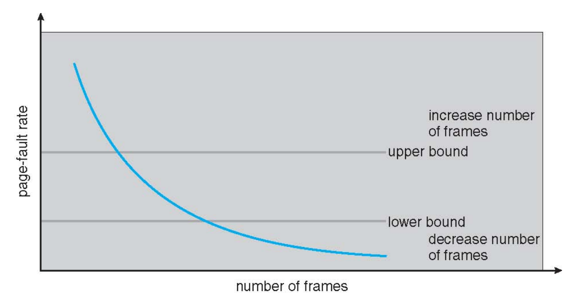
- **Problems?**

Keeping Track of the Working Set

- Approximate with **interval timer** Δ + **reference & history bits**
  - Timer interrupts every Δ /2 time units
  - Keep in memory **2 history bits** for each page
  - On timer interrupt - **set all reference bits to 0**
  - Reference bit set or one of the history bits set => page is in working set
- Example: Δ = 10,000
  - @ 0 : reset reference bits to 0 and history bits to 00
  - @ 5,000 : if reference bit = 1, set first history bit (i.e. bits are 0**1**)
  - @ 10,000 : if reference bit = 1, set second history bit (i.e. bits now **1**0 or **1**1)
  - On page fault:
    - If history bits are **01**, **10**, or **11**, or reference bit set, page is in working set
- **Problems?**
  - It's a coarse grained approach, a tradeoff between performance and accuracy

Track Page Fault Frequency

- More direct approach than the Working Set Model
- Establish "acceptable" page-fault frequency rate and use a local replacement policy



Page Replacement & Frame Allocation

- **Page v.s. Frame**
  - Virtual Page = Page
  - Physical Page = Frame
- **Page-Replacement** algorithm
  - Wants lowest page-fault rate both on first access and subsequent re-access
  - Decides which pages to replace
- **Frame-Allocation** algorithm
  - Decides how many frames to give each process
  - Decides which frames to re-allocate when necessary

Next Lecture

- File Systems I - Basics