

CSE 120

Principles of Operating Systems

Spring 2023

Lecture 13: Page Replacement and
Memory Allocation

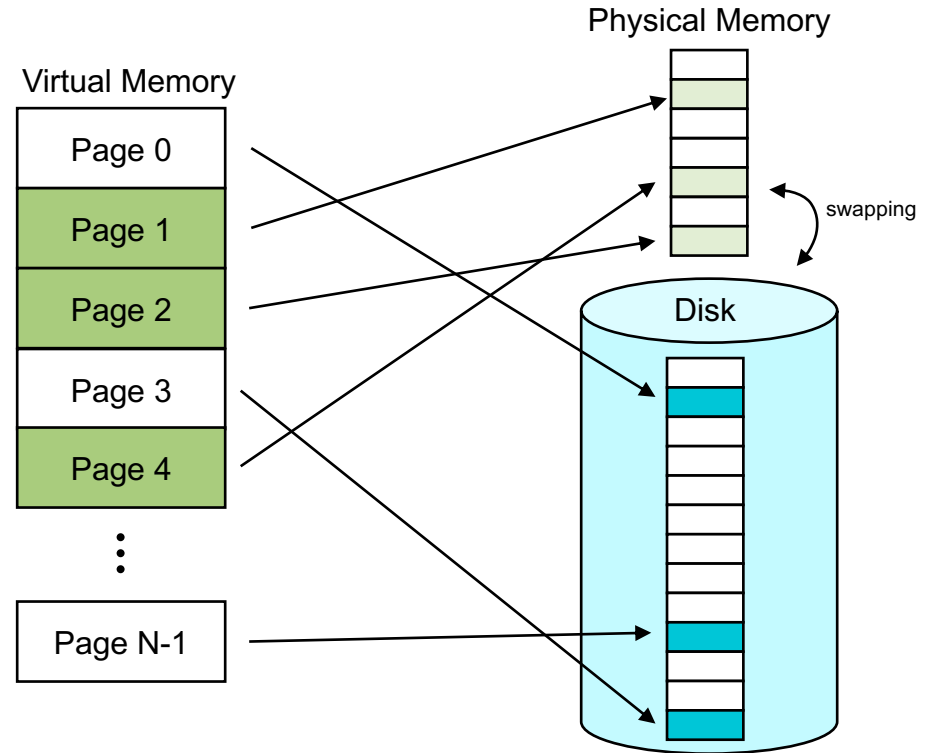
Amy Ousterhout

Administrivia

- Project 2 ongoing
 - ♦ Extended, now due Friday 5/19 (tomorrow)
- Homework #3 ongoing
 - ♦ Due Tuesday 5/23
- Project 3 released
 - ♦ Let us know if you would like to change teams
 - ♦ Due June 10th – there will be no extensions
 - ♦ Get started early!

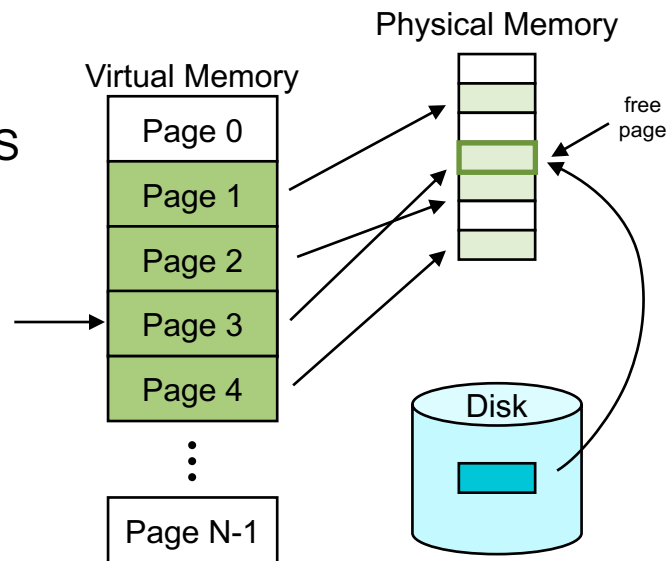
Demand Paging

- Store data on disk in a **swap file**
- Use main memory as a cache
- **Demand paging**: load pages on demand
- Process creation:
 - ◆ All page table entries are invalid
 - ◆ No pages in physical memory
- As a process executes:
 - ◆ Allocate pages on first access
 - ◆ Swap pages in (if not present)
 - ◆ Evict pages as needed to make room



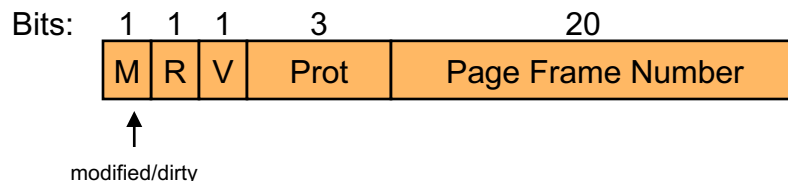
Page Faults

- What happens when a process references a page that is in the swap file?
 - ◆ Process references memory (e.g., LDR or STR)
 - ◆ Valid/present bit in PTE is set to 0
 - » Indicates that the page is not accessible
 - ◆ Triggers an exception (**page fault**) and a trap to the OS
 - ◆ OS runs the page fault handler
 - » Finds a free page frame in physical memory
 - » Reads the page in from the swap file to the page frame
 - » OS updates the PTE, sets valid=1
 - » Returns to the process
 - ◆ Process re-executes the memory reference

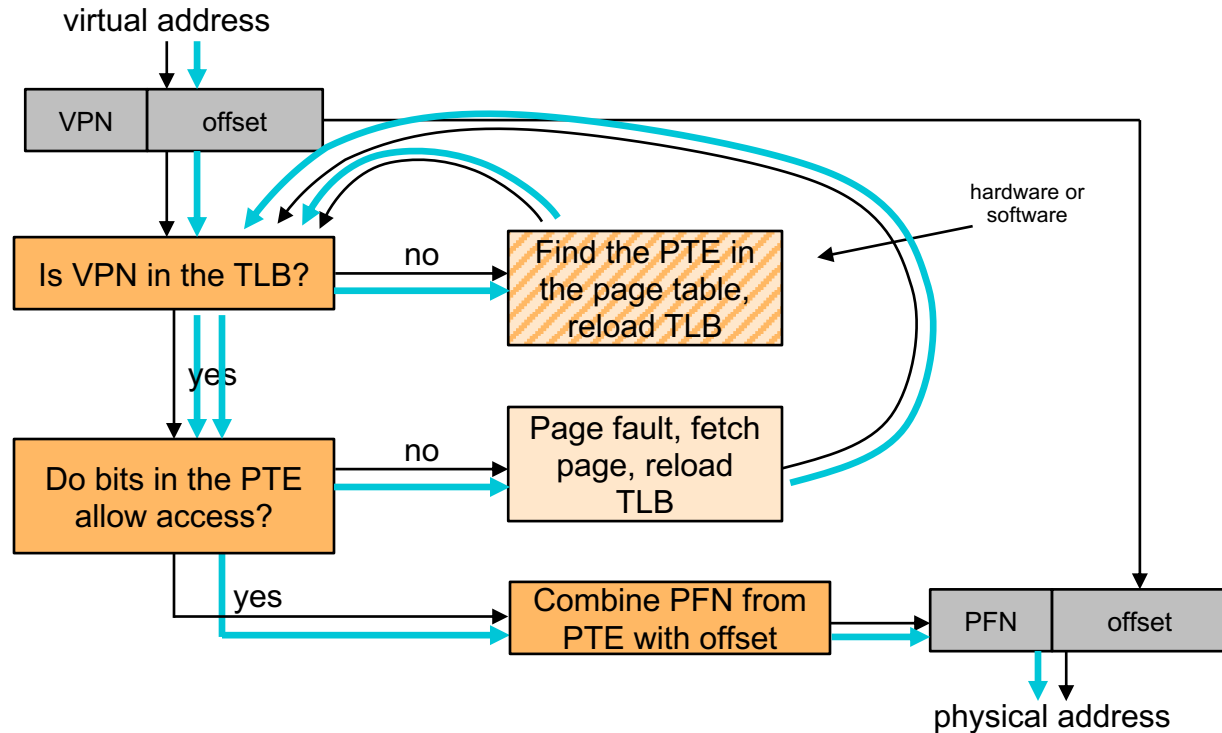


Page Frame Questions

- Is it safe for the OS to reclaim a physical page from process A and grant it to process B?
 - ♦ Yes, but the OS must **zero the page** first if it's a newly allocated page
- When we evict a page, do we always need to write the contents to the disk?
 - ♦ **Only modified (dirty) pages need to be written to disk**
 - ♦ Clean pages do not (they're already on disk)
 - ♦ Use the modified/dirty bit in the PTE



Address Translation for a Swapped-Out Page



Today's Outline

- Page replacement policies
 - ♦ Which page(s) should we evict?
- Virtual Memory allocation
 - ♦ Stack
 - ♦ Heap

Running at Memory Capacity

- Expect to run with most physical pages in use
 - ◆ OS maintains a small buffer of free page frames
- Every demand paging request requires an eviction
 - ◆ The OS must choose which page to **replace**

Page Replacement Policies

- **Page replacement policy**: determines which page to evict
- Goal: maximize hit rate
 - ◆ Equivalently, minimize page faults
- Challenges
 - ◆ It is hard to predict the future
 - » The OS does not know when a process will access each page in the future
 - ◆ Minimizing overhead
 - » Make the common case fast (page hit)
 - » Don't make the uncommon case too slow (page miss)

Page Replacement Policies

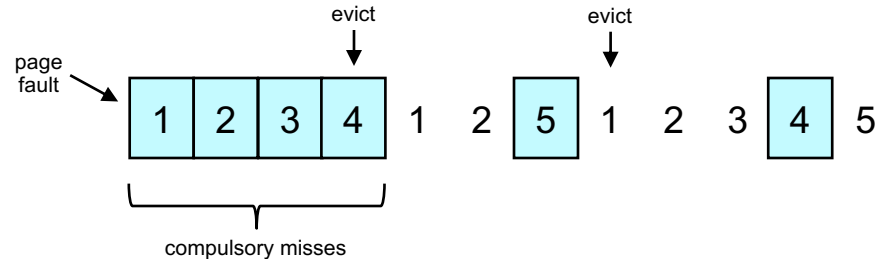
- Lots of different page replacement policies:
 - ◆ Optimal or MIN
 - ◆ Random
 - ◆ FIFO (First-In First-Out)
 - ◆ LRU (Least Recently Used)
 - ◆ Clock
 - ◆ NRU (Not Recently Used)
 - ◆ NFU (Not Frequently Used)
 - ◆ Aging (approximate LRU)
 - ◆ MRU (Most Recently Used)
 - ◆ MFU (Most Frequently Used)
 - ◆ ...

Evicting the Best Page

- Which is the “best page” to evict?
- The best page to evict is one that will never be accessed again
 - ♦ Will never fault on it
- Otherwise, the best page to evict is the one that will be **accessed the farthest in the future**
- Belady’s Algorithm implements this policy

Optimal or MIN (Belady's Algorithm)

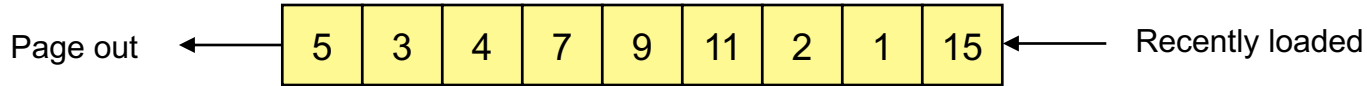
- Algorithm:
 - ♦ Replace the page that won't be used for the longest time
- Example:
 - ♦ 4 page frames, reference string:
 - ♦ 6 faults
- Pros:
 - ♦ Proven to be the optimal solution
 - ♦ Can be used as a yardstick to evaluate other algorithms
- Cons:
 - ♦ Requires that you know all future page accesses
 - ♦ No online implementation



Random

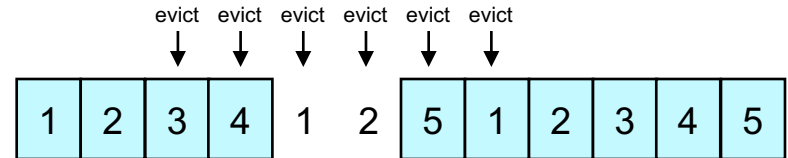
- Algorithm:
 - ♦ Choose a page at random to replace
- Pros:
 - ♦ Does not require knowledge of future accesses
 - ♦ Easy to implement
- Cons:
 - ♦ May not perform well

First-In First-Out (FIFO)



- Algorithm:
 - ♦ Maintain a list of pages in the order they were brought in
 - ♦ On replacement, evict the oldest page

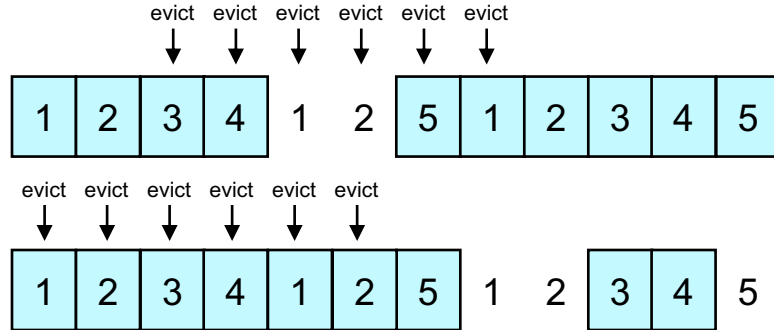
- Example:
 - ♦ 4 page frames, reference string:
 - ♦ 10 page faults



- Pros:
 - ♦ Low-overhead implementation
- Cons:
 - ♦ May replace heavily used pages

Belady's Anomaly

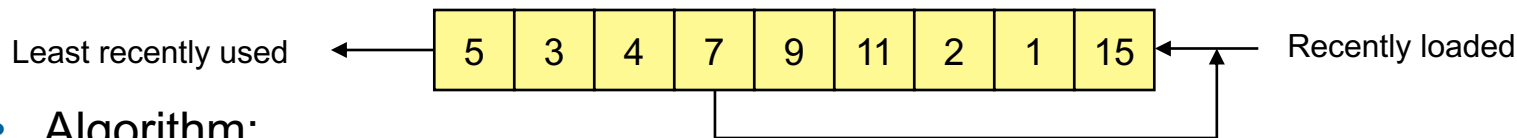
- Does more page frames result in fewer page faults?
- FIFO with 4 page frames
 - ♦ Reference string: 1 2 3 4 1 2 5 1 2 3 4 5
 - ♦ 10 page faults
- FIFO with 3 page frames
 - ♦ Reference string: 1 2 3 4 1 2 5 1 2 3 4 5
 - ♦ **9 page faults**
- This is called **Belady's Anomaly**
 - ♦ With some policies, more page frames does not result in fewer page faults



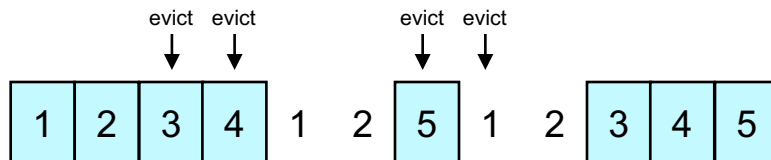
Locality

- **Locality** – programs tend to access certain code and data frequently
- **Temporal locality**
 - ◆ Locations referenced recently are likely to be referenced again
- **Spatial locality**
 - ◆ Locations near recently referenced locations are likely to be referenced soon
- Goal: leverage temporal locality to improve our page replacement policy

Least Recently Used (LRU)

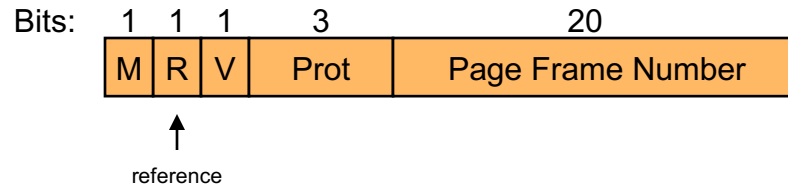


- Algorithm:
 - ♦ Replace the page that hasn't been used for the longest time
 - » Order pages by time of reference
- Example:
 - ♦ 4 page frames, reference string:
 - ♦ 8 page faults
- Pros:
 - ♦ Good approximation of MIN
- Cons:
 - ♦ Need to time stamp every memory access – **too much overhead**



Approximating LRU

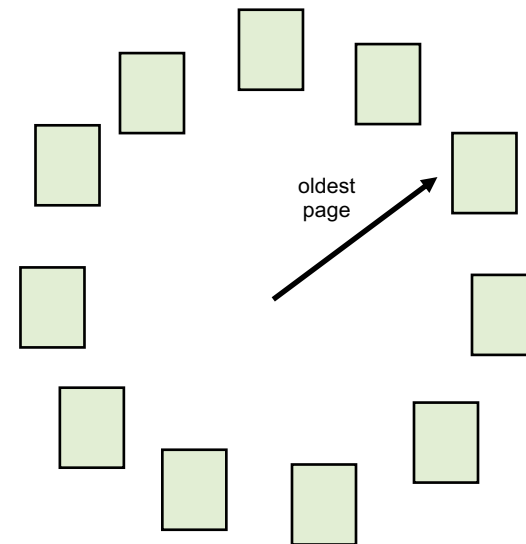
- Use the PTE **reference bit**
 - ♦ Set the bit when a page is accessed
 - ♦ Clear the reference bits periodically
- Reference bits tell us if a page was used recently or not



Clock



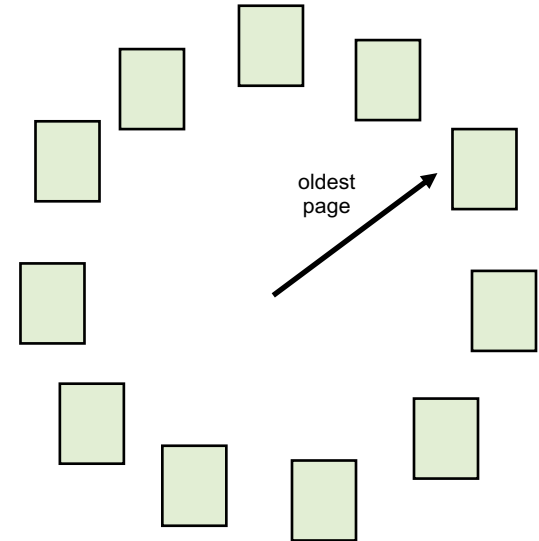
- Algorithm:
 - ♦ Arrange physical page frames in a circle (clock)
 - ♦ Clock hand points to the oldest page
 - ♦ Sweep through pages in a circular order
 - » If the reference bit is 0, use it for replacement (hasn't been used recently)
 - » If the reference bit is 1, set it to 0 and advance the hand
- What is the minimum “age” if ref bit is 0?
- What if every ref bit is set to 1?
- What happens if memory is very large?



Clock

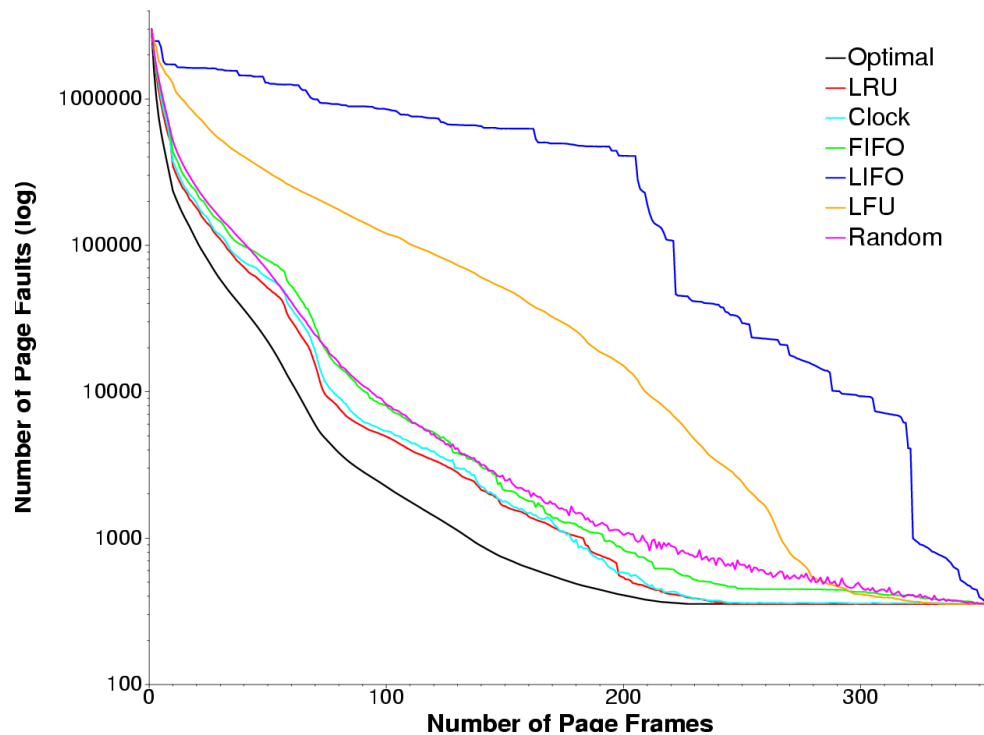


- Pros:
 - ♦ Simple to implement
- Cons:
 - ♦ Worst case may take a long time



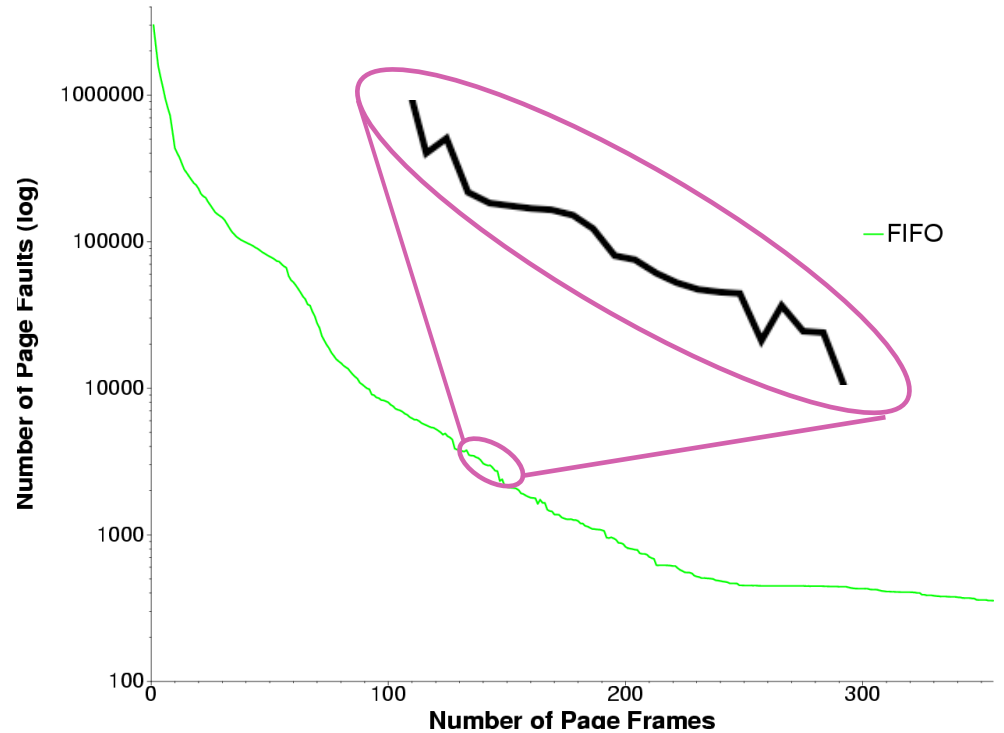
Example: gcc Page Replace

- Vary the size of allocated physical memory (x-axis)
- Measure the number of page faults (y-axis)



Example: Belady's Anomaly

- More page frames sometimes results in more page faults



Synchronous vs. Asynchronous

- Synchronous eviction
 - ◆ In the page fault handler, run the algorithm to find a page to evict
 - ◆ Might require writing changes to disk first
- Asynchronous eviction
 - ◆ A background thread maintains a pool of unused, clean pages
 - ◆ Occasionally evicts more pages to keep the pool large enough
 - ◆ Allows batching of page evictions for better efficiency

Handling Multiple Processes

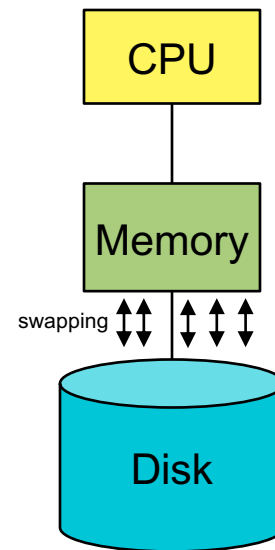
- We need to allocate memory to competing processes
- How much memory should we give to each process?
- **Global replacement**
 - ◆ All pages from all processes are in a single replacement pool
 - ◆ Pro: process' set of pages grows and shrinks dynamically
 - ◆ Con: processes compete for page frames
- **Local replacement/per-process replacement**
 - ◆ Each process has a separate pool of pages
 - ◆ A page fault in one process can only replace one its own frames
 - ◆ Pro: eliminates interference between processes
 - ◆ Con: how many pages should be in each pool?

Working Set Model

- The working set of a process is used to model the dynamic locality of its memory usage
 - ♦ Defined by Peter Denning in the 60s
- **Working set**: the set of pages in the most recent K page references
- Working set size: number of unique pages in the working set
- Working set changes dynamically as a program executes
- Challenges:
 - ♦ How do we pick K?
 - ♦ How can we track this information efficiently?
 - ♦ How do we know when the working set changes?
- Not used in practice for page replacement

Thrashing

- Memory overcommitment
 - ◆ Pages that are actively used by the current processes don't fit in physical memory
- Thrashing
 - ◆ Swapping in and out all the time, I/O devices fully utilized
 - ◆ Processes block, waiting for pages to be fetched from disk
- Solutions
 - ◆ Kill some processes
 - ◆ Buy more memory

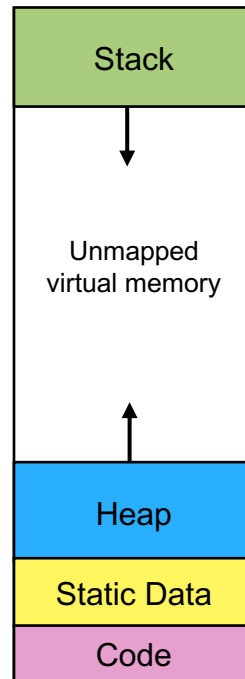


Today's Outline

- Page replacement policies
 - ◆ Which page(s) should we evict?
- Virtual Memory allocation
 - ◆ Stack
 - ◆ Heap

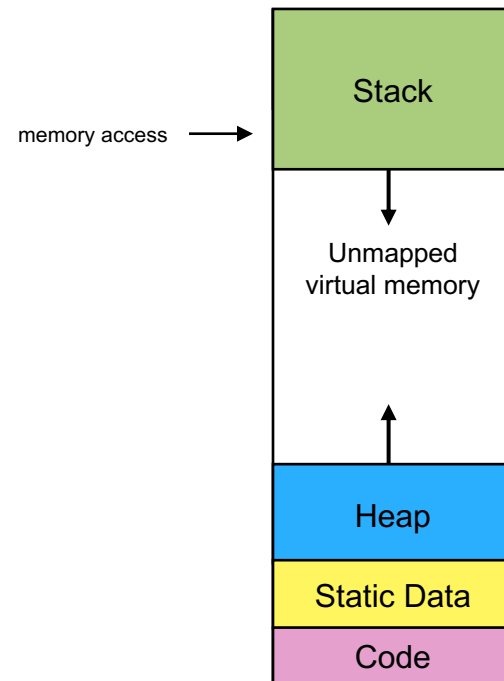
Virtual Memory Allocation

- At any given time, some of a process's virtual address space is **mapped** and some is not
 - ♦ The OS needs this information to know when to raise a segmentation fault
- How can a process allocate more virtual memory?
 - ♦ Stack
 - ♦ Heap



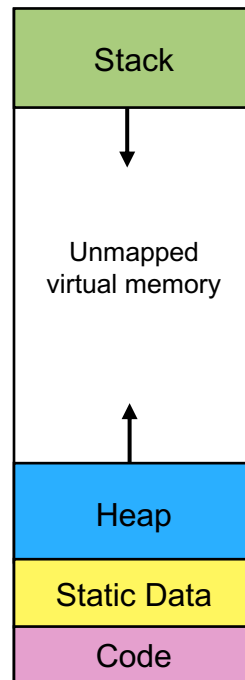
Growing the Stack

- Stack behavior
 - ◆ Grows linearly downward
- To grow the stack:
 - ◆ Process tries to grow the stack
 - ◆ Accesses unmapped memory
 - ◆ Triggers page fault
 - ◆ Page fault handler allocates a new page, zeroes it
 - ◆ Process resumes



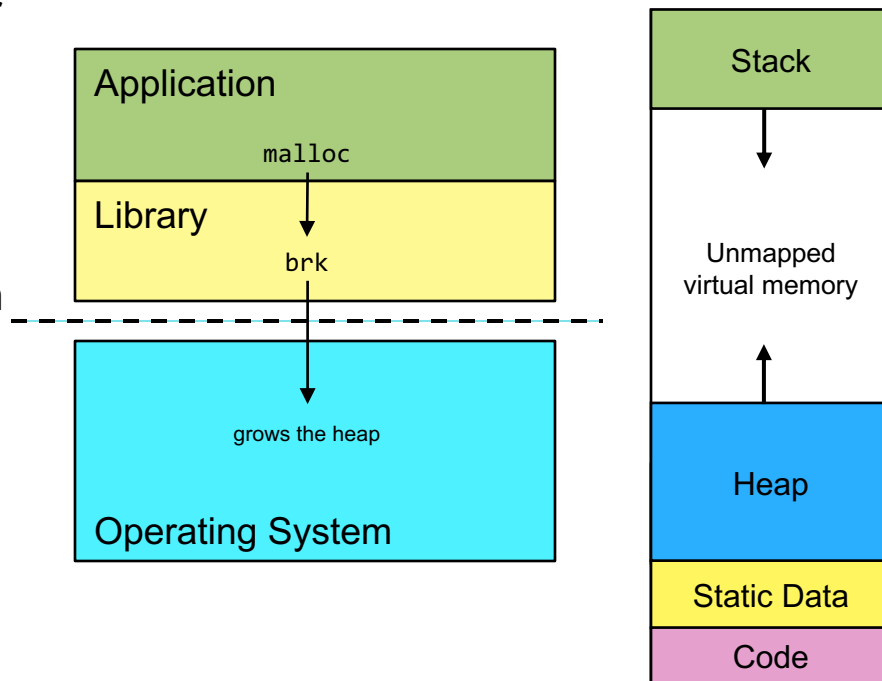
Growing the Heap

- Heap behavior
 - ♦ Grows and shrinks with `malloc` and `free`
 - ♦ Allocation and freeing are unpredictable
 - ♦ Memory has allocated areas and free areas
 - » Can suffer from fragmentation
- Heap management
 - ♦ Maintain a **free list** of holes
 - ♦ Managed by a library (e.g., `libc`)



Growing the Heap

- Use a system call to ask the OS for more memory
 - ♦ `brk` – grow the heap by a certain size
 - ♦ `mmap` – allocate a chunk of virtual memory, starting virtual address can be anywhere



Memory Management Summary

- We're done with memory management!
- **Goals** of memory management
 - ◆ Multitasking, transparency, isolation, and efficiency
- **Mechanisms**
 - ◆ Physical and virtual addressing
 - ◆ Partitioning, segmentation, paging
 - ◆ Page tables, TLBs, shared memory, copy-on-write
- **Policies**
 - ◆ Page replacement algorithms

Review Question

- What is the main benefit (or benefits) of each technique?
- Multi-level page tables
 - ♦ Reduce memory used by page tables
- Huge pages
 - ♦ Speed up address translations, reduce memory used by page tables
- Effective page replacement algorithms
 - ♦ Reduce page faults
- TLBs
 - ♦ Speed up address translations
- Copy on write
 - ♦ Reduce memory usage, reduce CPU time spent copying

Hint: reduce page faults,
reduce memory usage, etc.

For next class...

- Read chapter 37