



# COS 318: Operating Systems

## Virtual Memory Design Issues

Kai Li  
Computer Science Department  
Princeton University

(<http://www.cs.princeton.edu/courses/cos318/>)



## Design Issues



- ◆ Thrashing and working set
- ◆ Backing store
- ◆ Simulate certain PTE bits
- ◆ Pin/lock pages
- ◆ Zero pages
- ◆ Shared pages
- ◆ Copy-on-write
- ◆ Distributed shared memory
- ◆ Virtual memory in Unix and Linux
- ◆ Virtual memory in Windows 2000



2

## Virtual Memory Design Implications



- ◆ Revisit Design goals
  - Protection
    - Isolate faults among processes
  - Virtualization
    - Use disk to extend physical memory
    - Make virtualized memory user friendly (from 0 to high address)
- ◆ Implications
  - TLB overhead and TLB entry management
  - Paging between DRAM and disk
- ◆ VM access time

Access time =  $h \times \text{memory access time} + (1 - h) \times \text{disk access time}$

  - Suppose memory access time = 100ns, disk access time = 10ms
  - If  $h = 90\%$ , VM access time is **1ms!**
  - Is this the worst case?



3

## Thrashing



- ◆ Thrashing
  - Paging in and paging out all the time, I/O devices fully utilized
  - Processes block, waiting for pages to be fetched from disk
- ◆ Reasons
  - Processes require more physical memory than it has
  - Does not reuse memory well
  - Reuses memory, but it does not fit
  - Too many processes, even though they individually fit
- ◆ Solution: working set (recall the last lecture)
  - Pages referenced by a process in the last T seconds
  - Two design questions
    - Which working set should be in memory?
    - How to allocate pages?



4

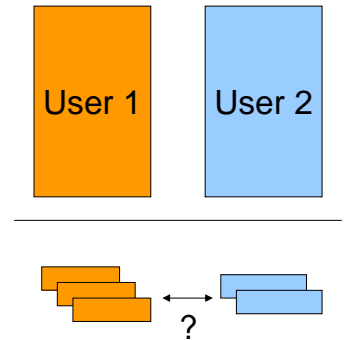
# Fitting Working Sets in Memory

- ◆ Maintain two groups
  - Active: working set loaded
  - Inactive: working set intentionally not loaded
- ◆ Two schedulers
  - A short-term scheduler schedules processes
  - A long-term scheduler moves active  $\leftrightarrow$  inactive to make sure active working sets fits in memory
- ◆ Key design point
  - How to decide which processes should be inactive
  - Typical method is to use a threshold on waiting time



# Global vs. Local Allocation

- ◆ The simplest is global allocation only
  - Pros: Pool sizes are adaptable
  - Cons: Too adaptable, little isolation (example?)
- ◆ A balanced allocation strategy
  - Each process has its own pool of pages
  - Paging allocates from its own pool and replaces from its own working set
  - Use a “slow” mechanism to change the allocations to each pool while providing isolation
- ◆ Design questions:
  - What is “slow?”
  - How big is each pool?
  - When to migrate?



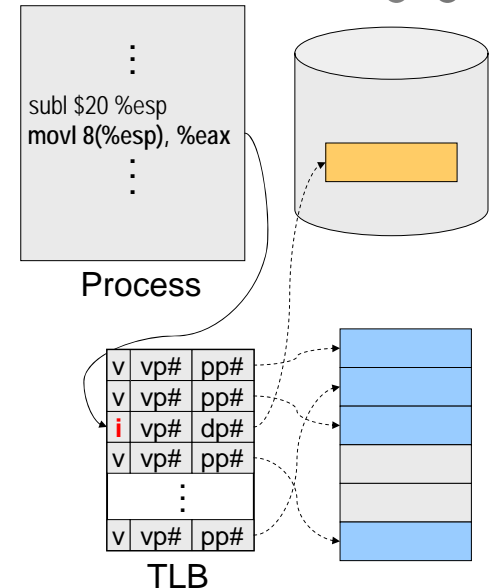
# Backing Store

- ◆ When process is created, allocate a swap space for it
  - Need to load or copy executables to the swap space
  - Need to consider process space growth
- ◆ When a page is allocated
  - Need a disk address
  - May not be the best disk allocation
  - May waste space if the page is never written back
- ◆ When a page is paged out
  - Probably save space
  - Can be a complex strategy
- ◆ Do you want to reallocate when the page is going out again?
  - Potential advantage is to improve disk write performance
- ◆ What about text pages?
  - Get from file?

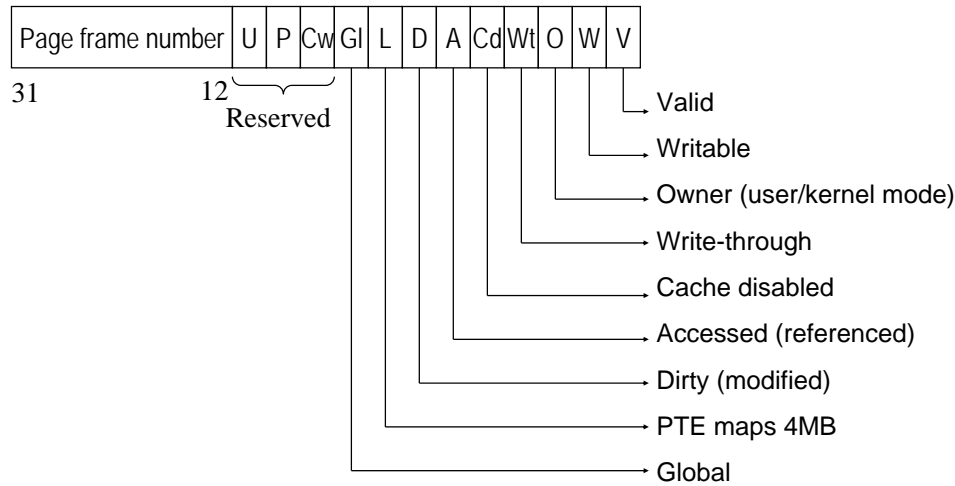


# Disk Address

- ◆ If valid bit = 1, pp# is the physical page number
- ◆ If valid bit = 0, it is disk page number
- ◆ Does this method work?



## x86 Page Table Entry



## Simulating Modify Bit with Access Bits

- ◆ Set pages read-only if they are read-write
- ◆ Use a reserved bit to remember if the page is really read-only
- ◆ On a write fault
  - If it is not really read-only, then record a modify in the data structure and change it to read-write
  - Restart the instruction

9

## Approximating LRU without Reference Bit

- ◆ Some machines have no reference bit
  - VAX, for example
- ◆ Use the valid bit or access bit to simulate
  - Invalidate all valid bits (even they are valid)
  - Use a reserved bit to remember if a page is really valid
  - On a page fault
    - If it is a valid reference, set the valid bit and place the page in the LRU list
    - If it is a invalid reference, do the page replacement
    - Restart the faulting instruction

## Pin (or Lock) Page Frames

- ◆ When do you need it?
  - When DMA is in progress, you don't want to page the pages out to avoid CPU from overwriting the pages
- ◆ How to design the mechanism?
  - A data structure to remember all pinned pages
  - Paging algorithm checks the data structure to decide on page replacement
  - Special calls to pin and unpin certain pages
- ◆ How would you implement the pin/unpin calls?
- ◆ If the entire kernel is in physical memory, do we still need these calls?

12

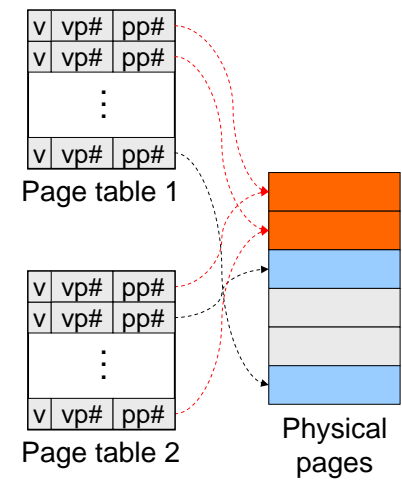
# Zero Pages

- ◆ Zeroing pages
  - Initialize pages with 0's
  - Heap and static data are initialized
- ◆ How to implement?
  - On the first page fault on a data page or stack page, zero it
  - Have a special thread zeroing pages
- ◆ Can you get away without zeroing pages?



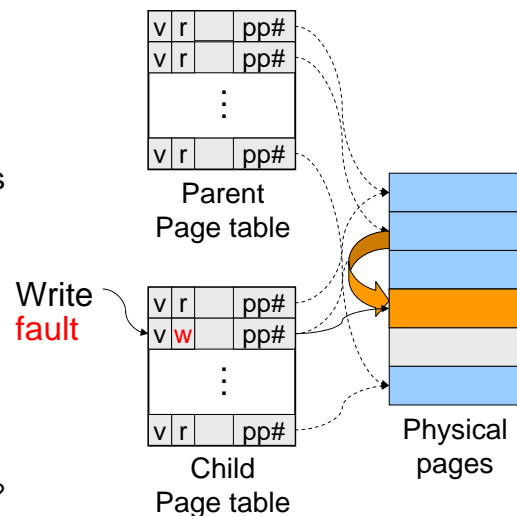
# Shared Pages

- ◆ Two or more PTEs from different processes share the same physical page frames
- ◆ Special VM calls to implement shared pages
- ◆ How to destroy an address space that share pages?
- ◆ How to page in and page out shared pages?
- ◆ How to pin and unpin shared pages?
- ◆ How to calculate working set for shared pages?



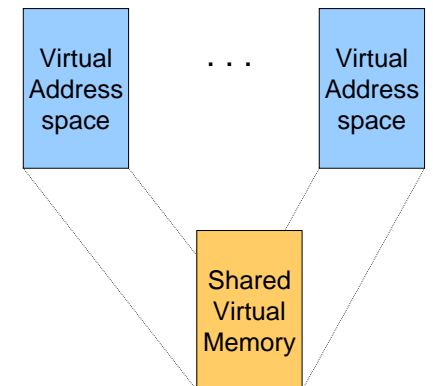
# Copy-On-Write

- ◆ Child's address space uses the same mapping as parent's
- ◆ Make all pages read-only
- ◆ Make child process ready
- ◆ On a read, nothing happens
- ◆ On a write, generates an access fault
  - map to a new page frame
  - copy the page over
  - restart the instruction
- ◆ Issues
  - How to destroy an address space?
  - How to page in and page out?
  - How to pin and unpin?



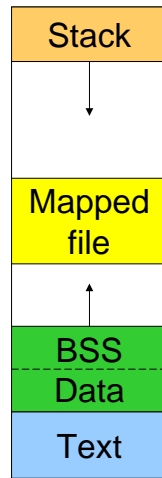
# Distributed Shared Memory

- ◆ Multiple address space mapped to "shared virtual memory" to implement distributed shared memory
- ◆ Page access are set according to coherence rules such as
  - Exclusive writer
  - N readers
- ◆ A read fault will invalidate the writer and copy page
- ◆ A write fault will invalidate another writer or all readers and copy page



## VM in Unix and Linux: Address Space

- ◆ Stack
- ◆ Data
  - Un-initialized: BSS (Block Started by Symbol)
  - Initialized
  - `brk(addr)` to grow or shrink
- ◆ Text: read-only
- ◆ Mapped files
  - Map a file in memory
  - `mmap(addr, len, prot, flags, fd, offset)`
  - `unmap(addr, len)`



Address space

17

## Virtual Memory in BSD4

- ◆ Physical memory partition
  - Core map (pinned): everything about page frames
  - Kernel (pinned): the rest of the kernel memory
  - Frames: for user processes
- ◆ Page replacement
  - Run page daemon until there is enough free pages
  - Early BSD used the basic Clock (FIFO with 2nd chance)
  - Later BSD used Two-handed Clock algorithm
  - Swapper runs if page daemon can't get enough free pages
    - Looks for processes idling for 20 seconds or more
    - 4 largest processes
    - Check when a process should be swapped in

18

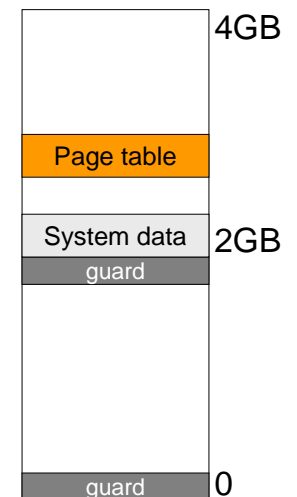
## Virtual Memory in Linux

- ◆ Linux address space for 32-bit machines
  - 3GB user space
  - 1GB kernel (invisible at user level)
- ◆ Implement copy-on-write for forking
- ◆ Backing store
  - Text segment uses executable binary file as backing storage
  - Other segments get backing storage on demand
- ◆ Multi-level paging
  - Directory, middle (nil for Pentium), page, offset
  - Kernel is pinned
  - Buddy algorithm with carving slabs for page frame allocation
- ◆ Replacement
  - Keep certain number of pages free
  - Clock algorithm on paging cache and file buffer cache
  - Clock algorithm on unused shared pages
  - Modified Clock on memory of user processes (most physical pages first)

19

## Address Space in Windows 2K/XP

- ◆ Win2k user address space
  - Upper 2GB for kernel (shared)
  - Lower 2GB – 256MB are for user code and data (Advanced server uses 3GB instead)
  - The 256MB contains for system data (counters and stats) for user to read
  - 64KB guard at both ends
- ◆ Virtual pages
  - Page size
    - 4KB for x86
    - 8 or 16KB for IA64
  - States
    - Free: not in use and cause a fault
    - Committed: mapped and in use
    - Reserved: not mapped but allocated



20

## Backing Store in Windows 2K/XP

- ◆ Backing store allocation
  - Win2k delays backing store page assignments until paging out
  - There are up to 16 paging files, each with an initial and max sizes
- ◆ Memory mapped files
  - Delayed write back
  - Multiple processes can share mapped files w/ different accesses
  - Implement copy-on-write

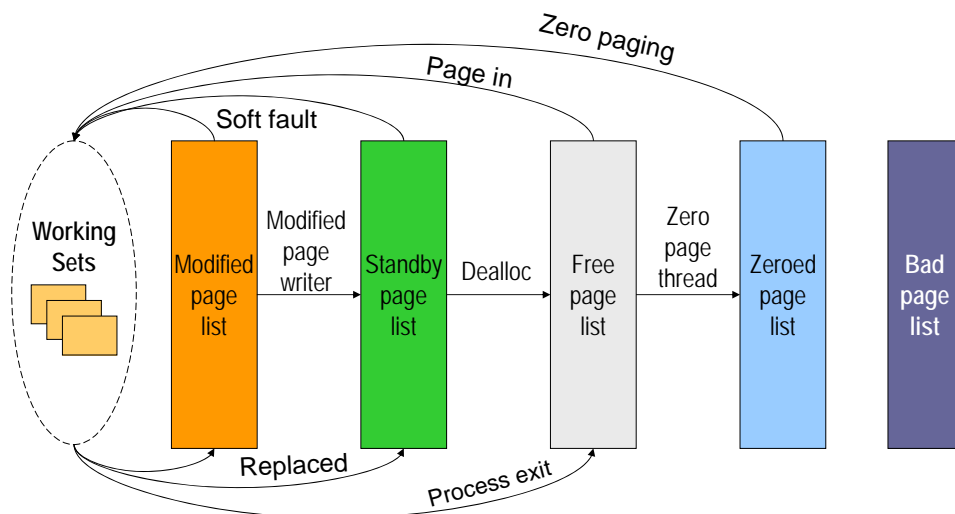
21

## Paging in Windows 2K/XP

- ◆ Each process has a working set with
  - Min size with initial value of 20-50 pages
  - Max size with initial value of 45-345 pages
- ◆ On a page fault
  - If working set < min, add a page to the working set
  - If working set > max, replace a page from the working set
- ◆ If a process has a lot of paging activities, increase its max
- ◆ Working set manager maintains a large number of free pages
  - In the order of process size and idle time
  - If working set < min, do nothing
  - Otherwise, page out the pages with highest “non-reference” counters in a working set for uniprocessors
  - Page out the oldest pages in a working set for multiprocessors
- ◆ The last 512 pages are never taken for paging

22

## More Paging in Windows 2K/XP



23

## Summary

- ◆ Must consider many issues
  - Global and local replacement strategies
  - Management of backing store
  - Primitive operations
    - Pin/lock pages
    - Zero pages
    - Shared pages
    - Copy-on-write
- ◆ Shared virtual memory can be implemented using access bits
- ◆ Learn from real systems

24