COS 318: Operating Systems

Virtual Memory Design Issues

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

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 × memory access time + (1 - h) × disk access time

  - E.g. Suppose memory access time = 100ns, disk access time = 10ms
    - If h = 90%, VM access time is 1ms!
  - What's the worst case?

Thrashing

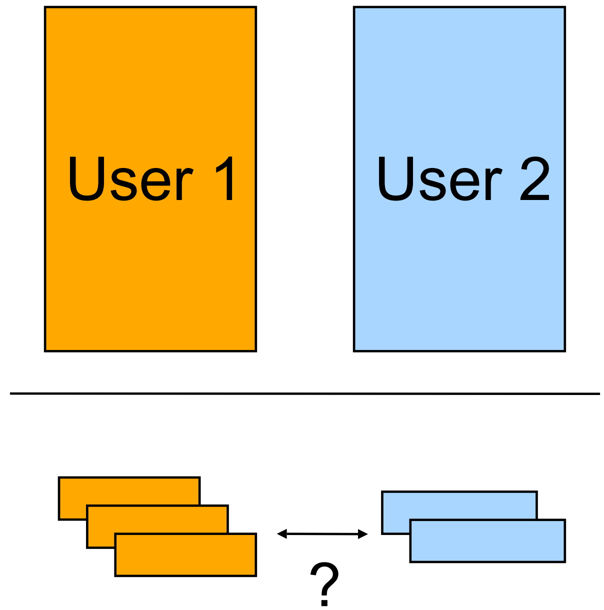
- Thrashing
  - Paging in and paging out all the time
  - Processes block, waiting for pages to be fetched from disk
- Reasons
  - Process requires more physical memory than system has
  - Does not reuse memory well
  - Reuses memory, but it does not fit
  - Too many processes, even though they individually fit
- Solution: **working set** (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?

Working Set: Fit in Memory

- Maintain two groups of processes
  - Active: working set loaded
  - Inactive: working set intentionally not loaded
- Two schedulers
  - A short-term scheduler schedules processes
  - A long-term scheduler decides which one active and which one inactive, such that active working sets fits in memory
- A key design point
  - How to decide which processes should be inactive
  - Typical method is to use a threshold on waiting time

Working Set: Global vs. Local Page Allocation

- The simplest is global allocation only
  - Pros: Pool sizes are adaptable
  - Cons: Too adaptable, little isolation
- 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
- Do global and local always make sense?
- Design questions:
  - What is “slow?”
  - How big is each pool?
  - When to migrate?

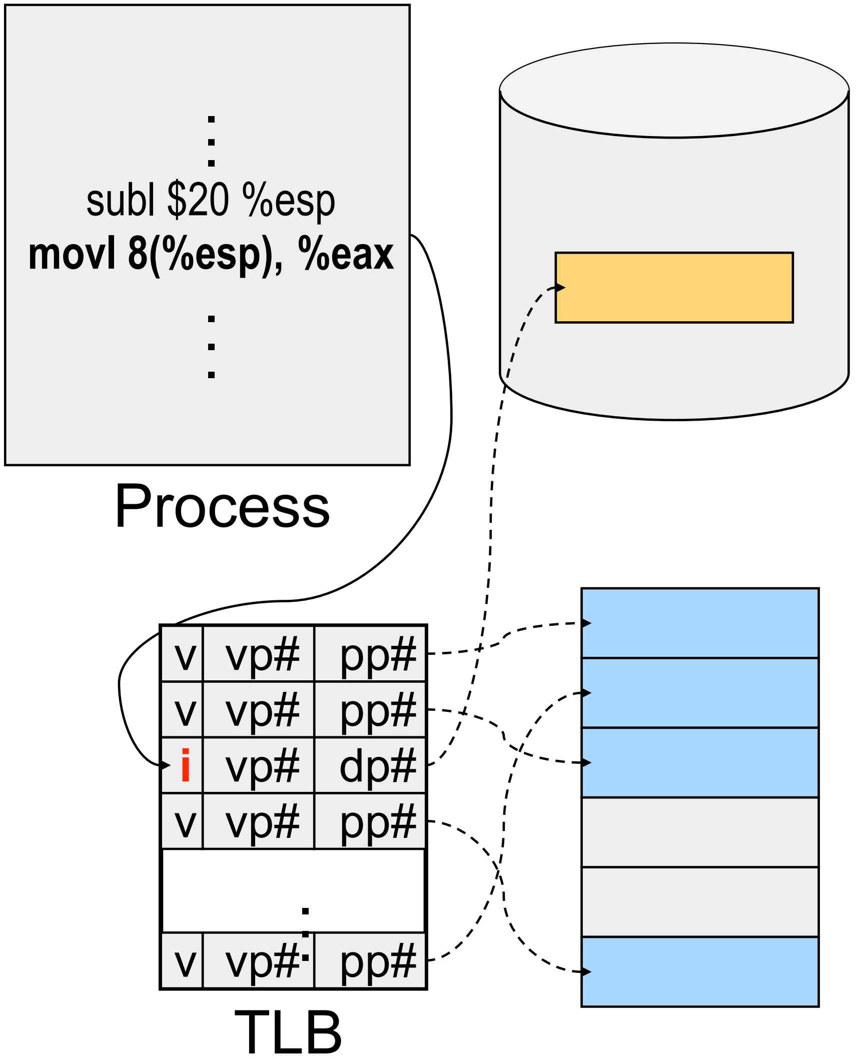


Backing Store

- Swap space
  - Separate partition on disk to handle swap (often separate disk)
  - When process is created, allocate swap space for it (keep disk address in process table entry)
  - Need to load or copy executables to the swap space, or page out as needed
- Dealing with process space growth
  - Separate swap areas for text, data and stack, each with > 1 disk chunk
  - No pre-allocation, just allocate swap page by page as needed
- Mapping pages to swap portion of disk
  - Fixed locations on disk for pages (easy to compute, no disk addr per page)
    - E.g. shadow pages on disk for all pages
  - Select disk pages on demand as needed (need disk addr per page)
- What if no space is available on swap partition?
- Are text files different than data in this regard?

Revisit Address Translation

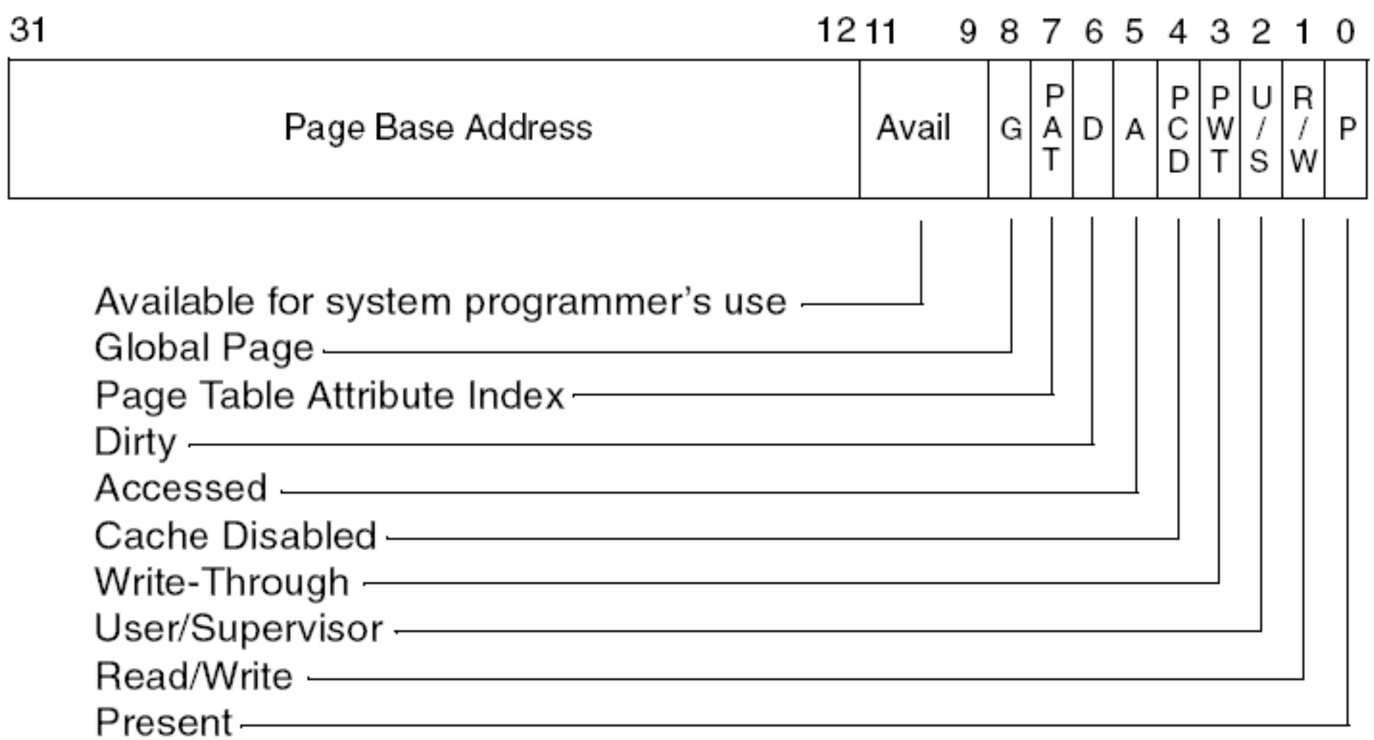
- Map to page frame and disk
  - If valid bit = 1, map to pp# physical page number
  - If valid bit = 0, map to dp# disk page number
- Page out
  - Invalidate page table entry and TLB entry
  - Copy page to disk
  - Set disk page number in PTE
- Page in
  - Find an empty page frame (may trigger replacement)
  - Copy page from disk
  - Set page number in PTE and TLB entry and make them valid



Example: x86 Paging Options

- Flags
  - PG flag (Bit 31 of CR0): enable page translation
  - PSE flag (Bit 4 of CR4): 0 for 4KB page size and 1 for large page size
  - PAE flag (Bit 5 of CR4): 0 for 2MB pages when PSE = 1 and 1 for 4MB pages when PSE = 1 extending physical address space to 36 bit
- 2MB and 4MB pages are mapped directly from directory entries
- 4KB and 4MB pages can be mixed

Page-Table Entry (4-KByte Page)



Pin (or Lock) Page Frames

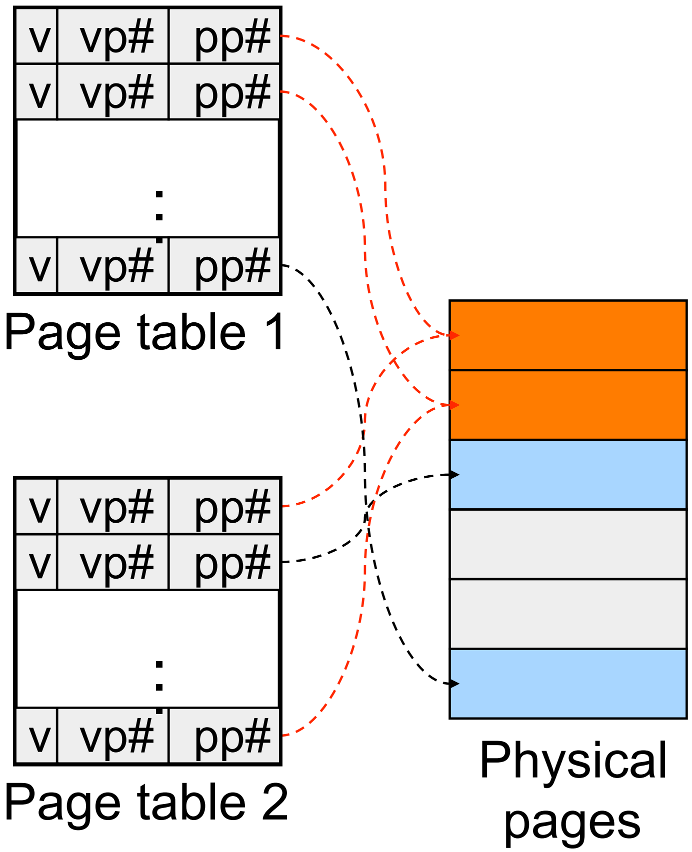
- When do you need it?
  - When I/O is DMA'ing to memory pages
  - If process doing I/O is suspended and another process comes in and pages the I/O (buffer) page out
  - Data could be over-written
- How to design the mechanism?
  - A data structure to remember all pinned pages
  - Paging algorithm checks 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?

Zero Pages

- Zeroing pages
  - Initialize pages with 0's
- 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?

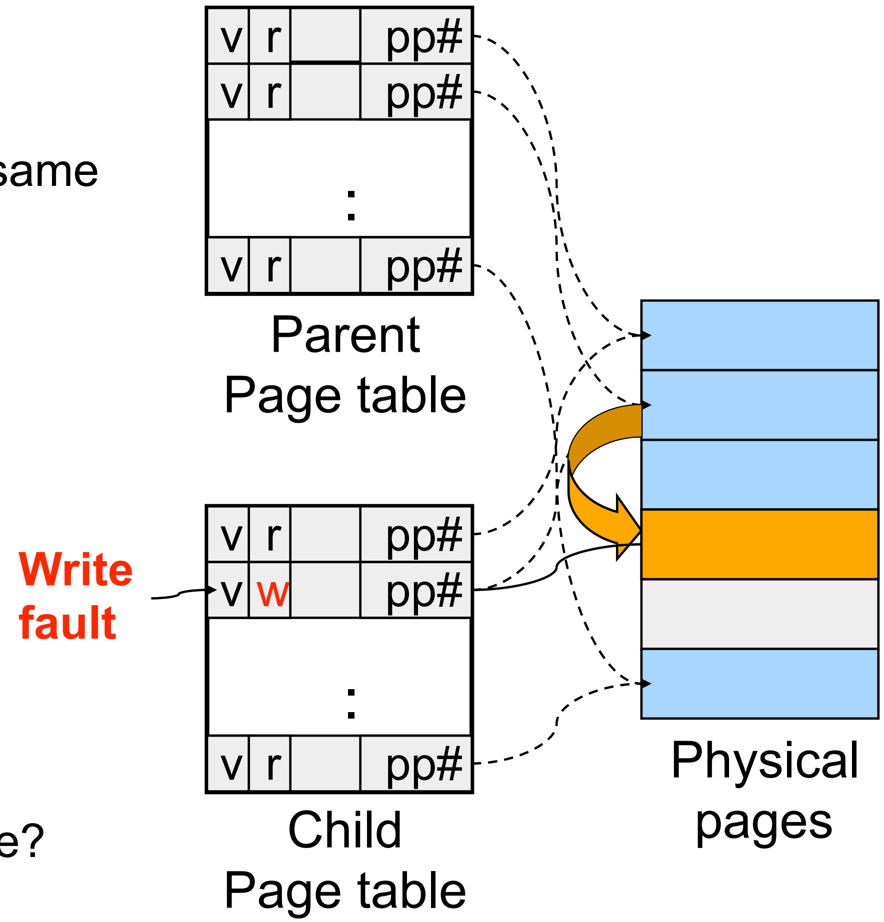
Shared Pages

- PTEs from two processes share the same physical pages
  - What use cases?
- APIs
  - Shared memory calls
- Implementation issues
  - Destroying a process with shared pages?
  - Page in, page out shared pages
  - Pin and unpin shared pages



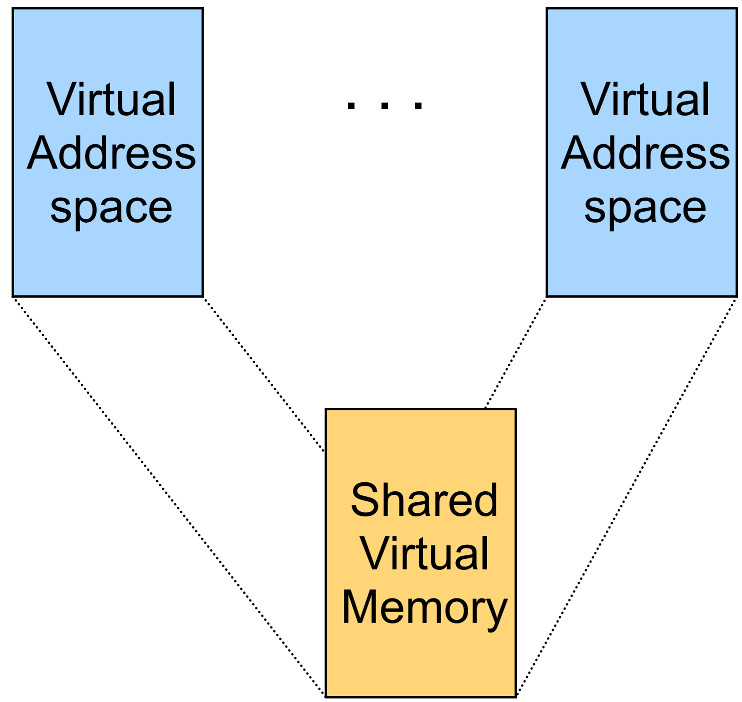
Copy-On-Write

- A technique to avoid copying all pages to run a large process
- Method
  - 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 a fault
    - map to a new page frame
    - copy the page over
    - restart the instruction
  - Only written pages are copied
- Issues
  - How to destroy an address space?
  - How to page in and page out?
  - How to pin and unpin?



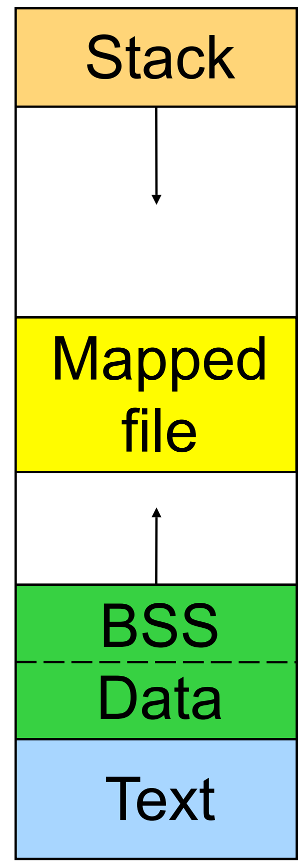
Distributed Shared Memory

- Run shared memory program on a cluster of computers
- Method
  - Multiple address space mapped to “shared virtual memory”
  - Page access bits are set according to coherence rules
    - Exclusive writer
    - N readers
  - A read fault will invalidate the writer, make read only and copy the page
  - A write fault will invalidate another writer or all readers and copy page
- Issues
  - Thrashing
  - Copy page overhead



Address Space in Unix

- 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

Virtual Memory in BSD4

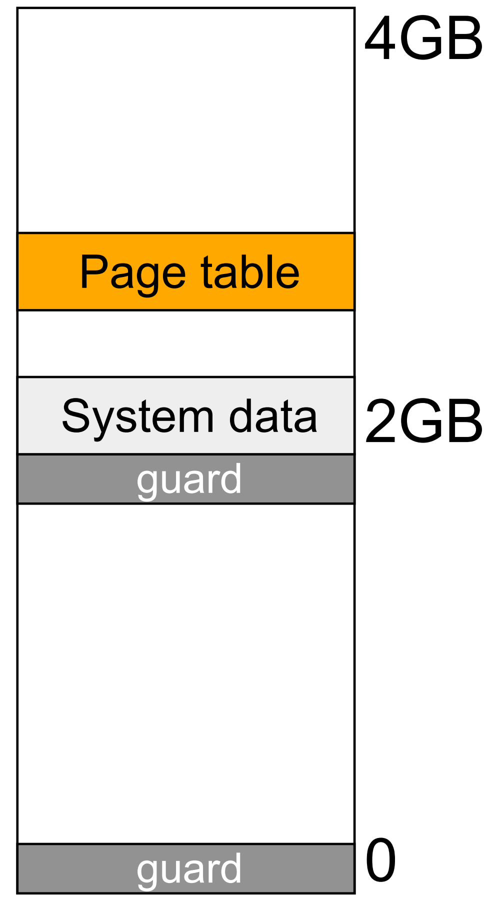
- Physical memory partition
  - Core map (pinned): everything about page frames
  - Kernel (pinned): the rest of the kernel memory
  - Frames: for user processes
- Page replacement
  - Run page daemon until there is enough free pages
  - Early BSD used the basic Clock (FIFO with 2nd chance)
  - Later 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

Virtual Memory in Linux

- Linux address space for 32-bit machines
  - 3GB user space
  - 1GB kernel (invisible at user level)
- Backing store
  - Text segments and mapped files uses file on disk as backing storage
  - Other segments get backing storage on demand (paging files or swap area)
  - Pages are allocated in backing store when needed
- Copy-on-write for forking off processes
- Multi-level paging: supports jumbo pages (4MB)
- 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

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 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



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
  - Multiple processes can share mapped files
  - Implement copy-on-write

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

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
- Real system designs are complex