

# Virtual Memory

CSE 410, Spring 2006  
Computer Systems

<http://www.cs.washington.edu/education/courses/410/06sp/>

# Readings and References

- Reading

- » Chapter 10 through 10.7.1, *Operating System Concepts*, Silberschatz, Galvin, and Gagne

- Other References

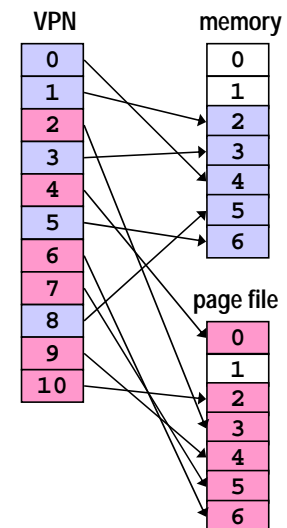
- » Chapter 7, *Inside Microsoft Windows 2000*, Third Edition, Solomon and Russinovich

# Virtual Memory

- Virtual memory paging to disk
  - » manage memory as though we always had enough
  - » if more is needed, use disk as backup storage
- Demand Paging
  - » load program pages in to memory as needed
- Another level of the storage hierarchy
  - » Main memory is a cache
  - » Disk space is the backing store

# Virtual Memory

- Page table entry can point to a PPN or a location on disk (offset into **page file**)
- A page on disk is swapped back in when it is referenced but is not actually present in main memory
  - » **page fault**



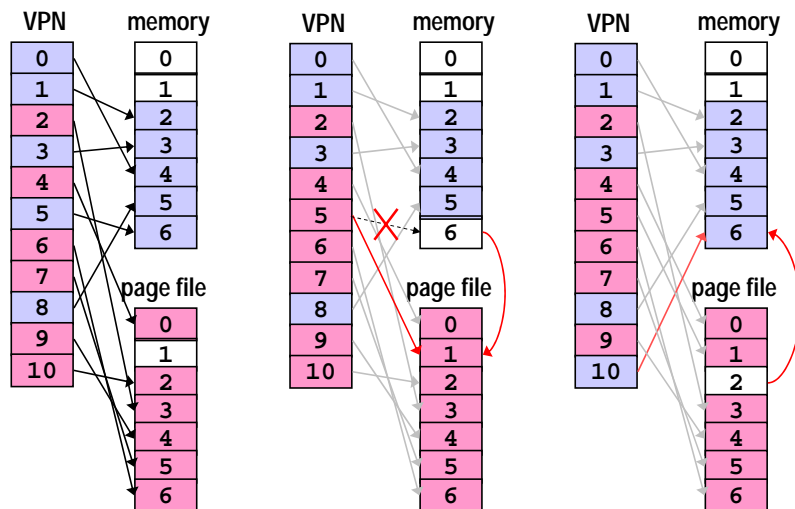
## Demand Paging

- As a program runs, the memory pages that it needs may or may not be in memory when it needs them
  - if in memory, execution proceeds
  - if not in memory, page is read in from disk and stored in memory
- If desired address is not in memory, the result is a page fault

## A reference to memory location **X**

- MMU: Is **X**'s VPN in the Translation Lookaside Buffer?
  - Yes => get data from cache or memory. **Done.**
  - No => Trap to OS to load X's VPN/PPN into the TLB
- OS: Is **X**'s page actually in physical memory?
  - Yes => replace a TLB entry with X's VPN/PPN. Return control to original thread and restart instruction. **Done.**
  - No => must load the page from disk
- OS: replace a current page in memory with **X**'s page from disk
  - pick a page to replace, write it back to disk if dirty
  - load X's page from disk into physical memory
  - Replace the TLB entry with X's VPN/PPN.
  - Return control to original thread and restart instruction. **Done!**

## Page Fault Example



Reference to VPN 10 causes a **page fault** because it is not in memory.

PPN 6 has not been used recently. **Write** it to the page file.

Read VPN 10 from the page file into physical memory at PPN 6.

## Virtual Memory & Memory Caches

- Physical memory can be thought of as a cache of the page file
- Many of the same concepts we learned with memory caches apply to virtual memory
  - both work because of locality
  - dirty bits prevent pages from always being written back
- Some implementation aspects are different
  - Virtual Memory is usually fully associative with complex replacement algorithms because a page fault is so expensive ( at least one disk read is required )

## Replacement Algorithms

---

- FIFO - First In, First Out
  - » throw out the oldest page
  - » often throws out frequently used pages
- RANDOM - toss a random page
  - » works okay, but not good enough
- OPT - toss the one you won't need
  - » pick page that won't be used for the longest time
  - » provably optimal, but impossible to implement

## Approximations to OPT

---

- LRU - Least Recently Used
  - » remember temporal locality?
    - if we have used a page recently, we probably will use it again in the near future
  - » LRU is hard to implement exactly since there is significant record keeping overhead
- CLOCK - approximation of LRU
  - » and LRU is an approximation of OPT

## Perfect LRU

---

- Least Recently Used
  - » timestamp each page on every reference
  - » on page fault, find oldest page
  - » can keep a queue ordered by time of reference
    - but that requires updating the queue every reference
  - » too much overhead per memory reference

## LRU Approximation: Clock

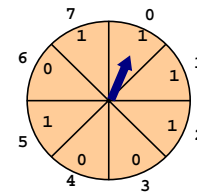
---

- Clock algorithm
  - » replace an old page, not necessarily the oldest page
- Keep a reference bit for every physical page
  - » memory hardware sets the bit on every reference
  - » bit isn't set => page not used since bit last cleared
- Maintain a “next victim” pointer
  - » can think of it as a clock hand, iterating over the collection of physical pages

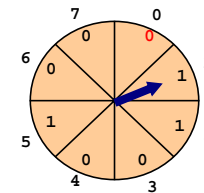
## Tick, tick, ...

- On page fault (we need to replace somebody)
  - » advance the victim pointer to the next page
  - » check state of the reference bit
  - » If set, clear the bit and go to next page
    - this page has been used since the last time we looked. Clear the usage indicator and move on.
  - » If not set, select this page as the victim
    - this page has not been used since we last looked
    - replace it with a new page from disk

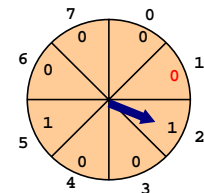
## Find a victim



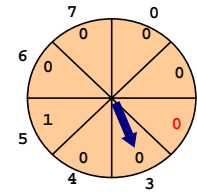
advance; **PPN 0** has been used; clear and advance



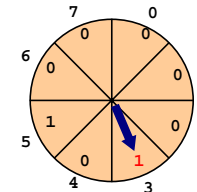
**PPN 1** has been used; clear and advance



**PPN 2** has been used; clear and advance



**PPN 3** has been **not** been used; replace and set use bit

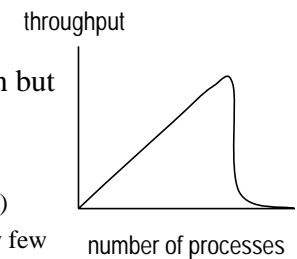


## Clock Questions

- Will Clock always find a page to replace?
  - » at worst it will clear all the reference bits, finally coming around to the oldest page
- If the hand is moving slowly?
  - » not many page faults
- If the hand is moving quickly?
  - » many page faults
  - » lots of reference bits set

## Thrashing

- **Thrashing** occurs when pages are tossed out, but are needed again right away
  - » listen to the hard drive grind
- Example: a program touches 50 pages often but there are only 40 physical pages in system
- What happens to performance?
  - » enough memory 2 **ms**/ref (most refs hit in cache)
  - » not enough memory 2 **ms**/ref (page faults every few instructions)
- Very common with shared machines



## Thrashing Solutions

---

- If one job causes thrashing
  - » rewrite program to have better locality of reference
- If multiple jobs cause thrashing
  - » only run as many processes as can fit in memory
- Big red button
  - » swap out some memory hogs entirely
- Buy more memory

## Working Set

---

- The working set of a process is the set of pages that it is actually using
  - » set of pages a job has used in the last T seconds
  - » usually much smaller than the amount it might use
- If working set fits in memory process won't thrash
- Why do we adjust the working set size?
  - » too big => inefficient because programs keep pages in memory that they are not using very often
  - » too small => thrashing results because programs are losing pages that they are about to use

## Appendix

---

### Windows 2000 Example

## Win2K Memory Management

---

- Win2K Pro/Server/DataCenter
  - » can manage 4 to 64GB physical memory
  - » Virtual address is 2GB user, 2GB system
- Some services of memory manager
  - » allocate / free virtual memory
  - » share memory between processes
  - » map large files into memory
  - » lock pages in memory

## W2K Working Set

- Subset of virtual pages resident in physical memory is the current working set
- W2K allows working set to grow
  - » demand paging causes read from disk
  - » reads in clusters of pages on a fault - 8 pages for code, 4 pages for data
- Working set is trimmed as necessary
  - » using version of the clock algorithm

## Managing allocations

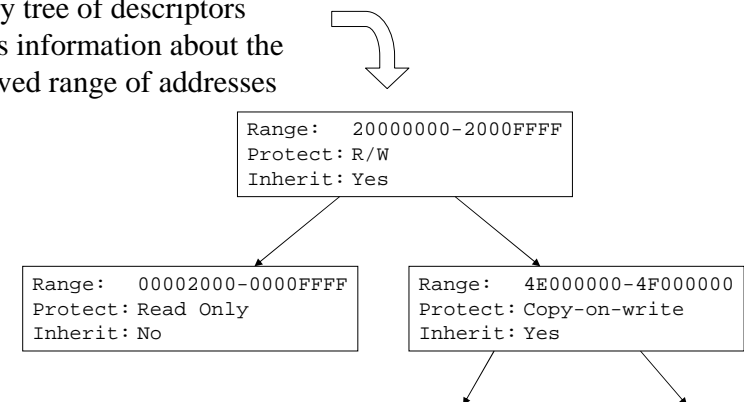
- A process reserves address space
  - » tell the OS that we will need this memory space
  - » OS builds Virtual Address Descriptors but does not build page tables
- then commits pages in the address space
  - » room exists for the pages in memory or on disk
  - » OS builds page table for committed page when a page fault occurs

## Example: Stack Allocation

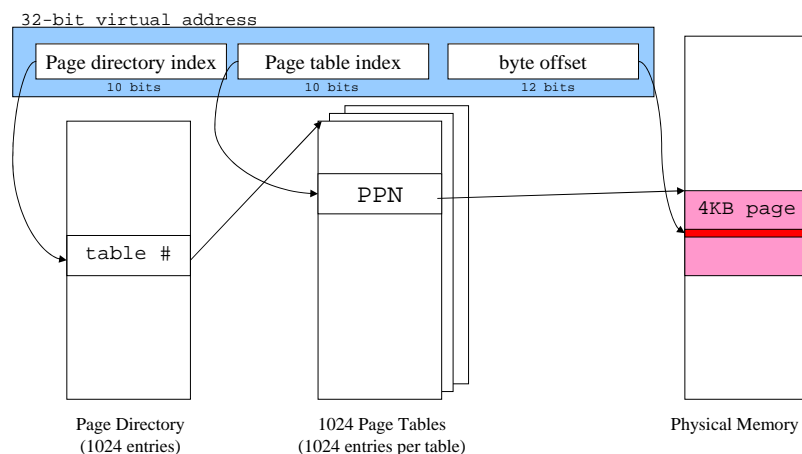
- Stack area is reserved when thread starts
  - » generally 1MB, although this can be changed at thread creation or with a linker switch
  - » Just one page of 4KB is committed
  - » the following page is marked PAGE\_GUARD
  - » if page fault, then one more page is committed and the stack is allowed to grow another 4KB until it happens again

## Virtual Address Descriptors

binary tree of descriptors  
stores information about the  
reserved range of addresses



## Two-level Page Tables



## Shared Memory

- “Section Objects” or file mapping objects
- Map portion of address space to common physical pages
  - » generally backed up with paging to disk
- page file backed - shared memory
- data file backed - memory mapped file, can be shared

## Address Windowing Extensions

- What do you do when 2GB is too small?
- Allocate huge chunks of physical memory
- Designate some virtual pages that are a window into that physical memory
- Remap the virtual pages to point to different parts of the physical memory as needed
- Useful for large database applications, etc

## AWE mapping

