Virtual Memory

● last level of the hierarchy

● program works in a “virtual” address space

● the physical address of data is determined dynamically at runtime

● a particular virtual address may correspond to many different physical addresses over the lifetime of a program
• Motivation:

• virtual address space larger than physical address space, e.g., 32 bit virtual address and 20 bit physical

• multiple users – each user has access to only a portion of physical memory (which may or may not be the same size as virtual memory)
• VM divides physical memory into blocks and allocates these blocks to different programs (processes)

• provides a mapping between blocks in physical memory and blocks on disk (virtual memory)

• each process can work in its own virtual address space e.g., two copies of the same program running simultaneously can access the same virtual address but access transparently different physical addresses

• a process can have only a portion of its virtual memory present in physical memory (working set like a cache and/or a process may only use a small fraction of its virtual memory space)
• reduces startup time by reducing data initialization

• protection from other processes can be implemented at block level

• liberates user from tedious part of memory management and lets them concentrate on the more important parts (locality enhancement)
VM Terms

• A block in VM is called a **page**

• A VM miss in physical memory is called a **page fault**

• the address used to access the main memory and (typically) the cache is called the physical address

• the mapping between virtual and physical addresses is facilitated by a data structure called the **page table**

• a cache that accelerates accesses to the page table is called a **translation lookaside buffer**
insert figure 7.20
• more virtual than physical pages for each process

• all pages (virtual and physical) have the same size

• translation takes a virtual address and returns either a physical address (page hit) or a disk address (page miss)

• the physical address is passed to the cache a page miss requires OS intervention
insert figure 7.21
• since pages are the same size, the low order bits of the virtual address are the same as the low order bits of the physical address

• translation maps the high order bits of the virtual address to the high order bits of the physical address

• 12 bit offset in page $\rightarrow 2^{12} = 4K\, B$ page size

• 20 bits in page number $\rightarrow 2^{20} = 1M$ pages

• therefore $2^{12} \times 2^{20} = 2^{32} = 4GB$ of virtual memory

• $2^{12} \times 2^{18} = 2^{30} = 1GB$ physical memory
• access times are much longer than physical memory (ms vs ns) due to large volume and mechanical component e.g. rotational latency.

• page size must be large enough to amortize high cost of disk access (mostly latency)

• 32KB to 64KB are common now

• reduction of page faults is crucial – fully associative page placement with some approximation of LRU replacement policy

• page faults are handled by SW (with some HW to accelerate it) to support better algorithms to reduce page faults
- write-back policy typically used to keep amount of disk communication down (cannot tolerate a disk access for every physical memory write!)
Page Tables

- number of entries in a page table is equal to the number of virtual pages (very large)

- number of virtual pages is the size of the address space divided by the size of a page

- each process requires its own page table

- a page table register points to the page table for the process (HW)
• a page table entry must contain
  – physical page number
  – valid bit
  – dirty bit
  – use bits (used it LRU replacement is approximated)
  – protection field, e.g., read only
  – disk address
insert figures 7.22 and 7.23
• two main problems for efficiency
  – size of page table
  – speed of translation

• consider size of page table first
Page Table Size reduction

- general idea: exploit locality and sparsity

- locality – pages accessed tend to be grouped in memory

- sparsity – most processes tend to use a fraction of their virtual address space
• Start with a small page table with a page bound register. Add to page table as the bounds on the pages accessed is exceeded

• separate heap (high addresses) pages from stack (low addresses) pages using two page tables with separate bounds registers

• exploit sparsity by using multiple levels of page tables, i.e., entries in high level table is accessed by the high order bits in virtual address and indicates if any pages in that segment of VM are in use.

• page the page tables – OS maintains page tables for the page tables of the active processes, beyond scope of this class

• use inverted page tables.
Inverted Page Table

- one entry for each physical page

- number of the virtual page currently stored in the physical page and a link to the next physical page with the same hash number are stored in each entry

- given a virtual address it is hashed to get the head of the list of physical page entries with the same hash value

- list is search and if no match found then a page miss is reported.

- saves space but increases page hit time