Two views of memory

- view from the hardware -- physical memory
- view from the software -- what program sees
- memory management in the OS separates these two views:
  - relocation: processes can be loaded at any address: logical addresses
  - protection: a process cannot maliciously access memory belonging to another process
  - sharing: controlled access to shared memory

Hardware translation

- translation from logical to physical can be done in software but without protection
- hardware support is needed to ensure protection
- simplest solution with two registers: base and size

Segmentation hardware

- segments are of variable size
- translation done through a set of (base, size, state) registers: segment table
- state: valid/invalid, write permission, reference bit, modified bit
- segments are visible to the programmer and can be used as a convenience for organizing the programs and data (i.e code segment or data segments)
Paging hardware

- pages are of fixed size
- the physical memory corresponding to a page is called page frame
- translation done through a page table (physical page frame, state) indexed by page number
- state: valid/invalid, write permission, reference bit, modified bit, caching
- paging is transparent to the programmer

Page table structure

- one-level PT: one lookup per access but can it can become huge
- two-level PT: saves memory but requires two lookups per access
- inverted page tables (one entry per page frame in physical memory): translation through hash tables

Combined paging and segmentation

- some MMU combine paging with segmentation
- translation is first performed through the segment table
- the segment entry points to a page table for that segment
- the page number portion of the logical address is used to index the page table and look up the corresponding page frame number

Translation Lookaside Buffers (TLB)

- translation occurs on every memory access -> must be done fast
- TLB is a hardware solution: a small cache of the PT inside the CPU (usually no more than 64 entries)
- usually a TLB entry contains the page number plus the PT entry and it acts like a cache
- on a memory access TLB is looked up first
- if the page number is not in the TLB, an existing entry is evicted and the missing entry is brought in from the PT
- TLB misses are handled in hardware or software
Support for multiple processes

- more than one address space can be loaded in memory
- a register points to the current segmentation/page table
- OS updates the register at switches processes
- most TLBs can cache more than one PT
- these TLBs must store the process id to distinguish between logical addresses belonging to different processes
- if TLB caches only one PT then it is (must be) flushed at the process switch time

TLB synchronization problem

- occurs in the multiprocessor case
- a process can be scheduled on one processor then scheduled on another processor -> TLBs within the multiprocessor must be synchronized to reflect the current state of a process: when?
- Solutions:
  - pinning processes to processors
  - TLB flush at every process switch
  - TLB synchronization when protection changes
  - ignore some state information: modified bit, reference bit

Virtual Memory

- the OS memory abstraction which gives the programmer the illusion of an address space larger than the physical address space
- virtual memory is traditionally implemented using MMU hardware and disks as secondary memory storage and is transparent to the programmer
- VM can use either paging or segmentation hardware but the term is usually associated with paging
- to be effective the OS must do a good job in managing the movement of segments/pages between main memory and the secondary storage

Memory hierarchy

- traditional memory hierarchy implemented by the VM in the OS:
  - main memory: fast access, high cost, limited
  - secondary memory: slow, cheap, large capacity
- most OSs make this hierarchy transparent to the user
  - hardware adds another level to the memory hierarchy (caches) which is usually transparent to the OS
VM implementation

- A virtual address space is created for each process.
- When a virtual page is mapped to a physical frame, the corresponding entry in the PT is valid and access translation can be performed.
- When a virtual page is not in physical memory, the corresponding entry in the PT is invalid and a page fault is triggered if an access occurs in that page.
  - A page is selected to be evicted from memory.
  - If the selected page is dirty, it is written to the disk.
  - The faulting page is brought in and mapped.

Page replacement

- Which page currently in memory should be replaced when a page fault occurs?
- Optimal: select the page for which the time to the next reference is the longest, requires knowledge of the future -> unrealizable.
- Realistic policies try to predict future behavior on the basis of past behavior; works because of locality.
  - Least-recently used (LRU): good but difficult to implement.
  - First-in, first-out (FIFO): easy to implement but performance is poor -> the second-chance scheme.

Second chance replacement

- Page frames are inspected in a round-robin fashion.
- Uses the reference bit in the PT.
- First time the page frame is inspected, the page is chosen for replacement if the reference bit is 0.
- If the reference bit is 1 (page was accessed in the recent past), the reference bit is reset to 0 and the page is given a “second chance”.
- If pages are kept in a circular list and the new page uses the evicted one’s slot, the scheme is called “clock replacement”.

Not frequently used replacement

- Software counters for each page frame.
- At each clock interrupt, the OS scans all pages in memory and the reference bit is added to the counter.
- The page with the lowest counter is chosen for replacement.
- Problem: doesn’t forget anything, no sense of time.
- Can be improved with an aging scheme: counters are shifted right before adding the reference bit and the reference bit is added to the leftmost bit (rather than to the rightmost one).
Page Buffering

- pages marked for replacement are not replaced immediately but added to a free list
- if the replacement decision was wrong there is a chance to find the page still in memory at the fault time
- allows page frame selection (placement) of the faulting page which can be used to improve cache behavior

Virtual memory and cache conflicts

- assume an architecture with direct-mapped caches (first-level caches are usually direct-mapped)
- the VM page size partitions a direct-mapped cache into a set of cache-pages
- page frames are colored (partitioned into equivalence classes) where pages with the same color map to the same cache-page
- cache conflicts can occur only between pages with the same color, and no conflicts can occur within a single page

VM mapping to avoid cache misses

- Goal: to assign the active virtual pages to different cache-pages
- a mapping is optimal if it avoids conflict misses
- a mapping that assign two or more active pages to the same cache-page can induce cache conflict misses
- example:
  - a program with 4 active virtual pages
  - 16 KB direct-mapped cache
  - a 4 KB page size partitions the cache into four cache-pages
  - there are 256 mappings of virtual pages into cache-pages but only 4! = 24 are optimal

Page recoloring

- assuming conflict is detected at runtime: counting cache misses on a per-page basis
- conflicts can be solved by copying the content of one or more of the conflicting pages into new page frames of different color: recoloring
Resident set management

- how many pages of a process should be brought in?
- resident set size can be fixed or variable
- replacement scope can be local or global
- most common schemes implemented in the OS:
  - variable allocation with global scope: simple, resident set size is modified at the replacement time
  - variable allocation with local scope: resident set size is modified to approximate the working set size (complicated)

Working Set

- the set of pages that have been referenced in the last window of time
- the size of the working set varies during the execution of the process depending on the locality of accesses
- if the number of pages allocated to a process covers its working set then the number of page faults is small
- schedule a process only if its working set is in memory
- how to determine/approximate the working set size

Page-fault frequency

- a counter per page stores the virtual time between page faults (could be the number of page references)
- an upper threshold for the virtual time is defined
- if the amount of time since the last page fault is less than the threshold, then the page is added to the resident set
- a lower threshold can be used to discard pages from the resident set

Application-controlled paging

- OS kernel provides the mechanism and implements the global policy: chooses the process which has to evict a page
- Application decides the local replacement
- Basic protocol for an external memory manager:
  - at page fault: kernel upcalls the manager asking to provide the page
  - manager provides the page to the kernel and the kernel does the mapping
Application-controlled paging (cont’d)

- when a dirty page is chosen to be evicted following a global scope replacement, the kernel upcalls the manager asking to save the page
- a more elaborated scheme would allow the manager to select another page to be evicted: combine global with local scope replacement

Using remote memory

- Assume a cluster of computers interconnected with a fast network
- Use memory of remote workstations as secondary storage instead of local disks; memory servers
- Pages can be “swapped” out to a remote memory server instead of local disk
- Memory servers extend the memory hierarchy with a new layer
- Memory servers can be dedicated machines or not
- Good performance because network bandwidth is higher than disk bandwidth

Global memory management

- assume a cluster of workstations which trust each other
- the goal is to globally coordinate the memory management
- classify pages at node N as being either local (recently accessed by N) or global (stored in N’s memory on behalf of other nodes)
- pages may also be private or shared. A shared page may be in many active local memories but a page in global memory is always private.

Global memory replacement

- assume node N fails on accessing a nonresident page P
- Case 1: P is in the global memory of M. We swap P in M’s global memory with any page in N’s global memory. Once at N, P becomes a local page. Size of local memory at N increases by 1. M unchanged
- Case 2: P is in the global memory of M but N’s memory contains only local pages. Exchange the LRU local page at N with the faulted page at M. Size of local memory at N and M unchanged.
Global memory replacement (cont’d)

- assume node N fails on accessing a nonresident page P
- Case 3: P is on disk. Read P into local memory of N, Choose the oldest page in the cluster (say at M) for replacement and write it to disk (if necessary). Send a global page (or the LRU local page) from N to M.
- Case 4: P is a shared page in the local memory of M. Bring P to N leaving the original in local memory of M. Choose the oldest page in the cluster for replacement and write it to disk (if necessary). Send a global page (or the LRU local page) from N to M.
- how to determine the oldest page in the cluster?

Global age management

- a probabilistic algorithm
- divide time into epochs: maximum R replacements per epoch
- at the beginning of each epoch, each node sends a summary of the ages to a designated initiator node
- the initiator computes weights \( w_i \) for each node i such that out of the R oldest pages, \( w_i \) resides in i’s memory
- the initiator also computes the minimum age of pages that will be replaced from the cluster
- all \( w_i \) plus the minimum age are sent to all nodes

Global age management (cont’s)

- when N has to decide to evict a page from its memory it first check whether the age of the evicted page is older than the minimum age for the current epoch. If so, it discards the page
- if not, N sends the page to node i, where the probability of choosing node i is proportional to \( w_i \). In this case, the page discarded from N becomes a global page at i, and the oldest page at i is discarded