VMWare ESX Memory Mgmt.

Goal : Multiplex machine memory across all guest OSes in a fair manner.

1. What are the functions of a memory management system?
- Paging, Swapping, TLB Management etc.

2. Fairness across processes?
- Desktop systems don't care much about being fair, but older systems used to do it. In case of virtualized environment fairness is critical.

3. How does OS multiplex raw memory?
- The OS needs to make sure one process does not monopolize the system by occupying the entire physical memory.
- The OS adopts demand paging, pre-fetching of pages etc. to bring in pages of a process into physical memory.
- A process is not given a fixed amount of physical memory. It is allocated physical memory when it demands and then its pages be brought in to memory.
- OS uses algorithms such as FIFO, LRU, Clock etc to revoke pages and to make room for pages that are about to enter physical memory.

Aside :
- Global LRU -> There is no notion of fairness. It is prone to thrashing.
- Thrashing: Pages are swapped out by the OS but the adversary process demands them back in. This goes on and on and no work gets done.
- MMU (TLB's, Page tables) assists the OS in multiplexing this resource.
- Take away: The OS does not beg the process to tell the OS which pages to evict. The OS snatches it from the process, when there is an impending need.

4. How can a hypervisor perform memory management?
- One simple scheme: Proportionately divide machine memory across VMs. Keep this division constant throughout the VMs' lifetimes.
  - Merit - It's easy!
  - Drawbacks:
    - There can be under-utilization
    - Does not fully justify efficient server consolidation, which is what virtualization is meant for.
- Use Ballooning. (Resort to primitive technique of snatching when ballooning fails.)

5. How to overcome under-utilization?
- Just do what the OS does! Snatch idle pages away from a VM. (Demand Paging).
  - Drawback :
    - Double paging (i.e VMM swaps out a page to disk but if guest OS gets memory pressure at the same time, it might decide to swap out the same page. In such a case the VMM has to bring back the page into memory and then swap it back out.)
  - Where can it happen in a commodity OS?
    - When the application running has its own memory management schemes and the OS is unaware of it! Eg: A database system which uses its own memory management system.

Ballooning:

- Goal: How to reclaim pages from a guest OS?
- DISCO handled memory mgmt. by altering IRIX OS. Not always advisable to do so.
- So, instead of making a change in the guest OS (which is hard to do), install a balloon driver on the guest OS.
- It has access to OS interfaces such as asking for memory.
- Balloon driver interacts with VMM using communication channels like some **out of band exception** which a normal guest OS won't do.
- When a VMM needs free pages it:
  - Selects a VM from which a page should be evicted.
  - Calls up into the balloon driver to get some free pages.
  - Balloon driver "inflates" and puts the guest under stress and asks for pinned pages (Pages that cannot be swapped out).
  - Guest OS could responds:
    - By returning pages from the free list if there are any.
    - Else, it runs its own memory mgmt. schemes and returns the resulting pages to the driver.
    - Else it ignores the request, in case driver has asked for too many pages.
  - Balloon driver passes information about acquired pages to VMM for its use.
  - VMM makes sure that Guest OS's request does not reference these pages again by updating corresponding entries in pmap.
- VMM provides protection against the cases when the guest OS might try to touch the pages that it gave to the driver. This can happen in case of driver crash, machine crash, machine reboot etc.
- Ballooning and Demand Paging allow over-commitment of memory, which is critical for server consolidation.

Content Based Page sharing:

- Goal: to reduce memory pressure by sharing pages across different VMs. Pages that can potentially be shared among VMs :
  - Kernel code : Multiple VMs might be running the same kernel
  - Zeroed pages
  - Re-entrant (or Shared) libraries
- How can ESX detect such pages?
  - Dumb way: Scan and compare each machine page with every other machine page - O(n^2).
  - Smart Way: Use Content based page sharing:
    - Scan pages at some rate.
    - A hash table entry points to a scanned page. Use the hash value as a look up key.
    - When a new guest page is to be brought in, hash its contents. Compare the hash to a hash table containing previously matched pages.
    - If a match is found, then contents of the pages are compared (as there can be chances of collision).
    - If comparison passes: coalesce pages and mark each entry COW.
    - If there is a match with an existing entry, do a byte-by-byte comparison to assert that the pages are really the same.
- Content based sharing is good because it covers those cases as well where user himself does not specify that a duplicate page might be present (something like using a shared library)
- Aside :
  - Shared objects (SO) & DLL Hell! Static EXEs (or Archives) made life easy in some sense.

- Where does sharing happen in OS, among processes?
  - Shared libraries.
  - When a parent spawns a child, they have a common COW address space.
- Bad Hash function -> More collisions -> More byte-by-byte invalid comparisons -> Poor performance
- OS looks for cross-process similarity; VMM looks for cross VM similarity.
- If both VMM and VM were doing content bases sharing: It will still work but VMM will just need to do a comparison for pages across VMs and not for pages belonging to same VM.

Share-based Page allocation:

- Goal: When system is under memory pressure which VM should VMM snatch pages from.
- Some ways of doing this:
  - Detect idle pages and snatch them.
  - Assign shares/tickets to each VM. When a revocation has to be done the VM with least value of (shares/pages) is selected.
  - This leads to under-utilization as rich and lazy VMs are rarely penalized.
  - "idle memory taxing" handles such cases. Charge a VM more for an idle page than for a page it is actively using. So pages will be reclaimed from clients that aren't actively using their full allocations.
  - Instead of (shares/pages), least value of [rho = shares / pages * (f + k (1 - f))] is used to select the VM for revocation.
    - f: fraction of non-idle pages
    - k = 1 / (1 - t), where 0< t < 1 is the tax rate (default : 0.75)
    - tax rate provides control over desired reclaiming policy.
    - t = 0 implies pure share-based allocation.
    - t = 1 implies a policy which allows all of clients memory to be reclaimed for more productive uses.
  - How to find idle memory:
    - Each VM is sampled independently.
    - Select n pages randomly.
    - Sampled page is tracked by Invalidating its cached mappings associated with PPN eg: TLB entries and virtualized MMU state.
    - When the page is touched by guest OS, it results in a fault :
      - re-establish the mappings
      - increment the touched page counter (t).
      - At the end of sampling f = t/n.
- Aside :
  - How can you avoid paying taxes?
    - Touch your pages regularly.
  - Huge pages - Some back ground process that scans all pages to see which ones can be coalesced.

Admission control:

- Ensures that sufficient memory and swap space is available before a VM can be powered ON.
- Available Memory should be min_size + overhead.

Key take-away:

- Memory Management Policy.
  - Ballooning
  - Content based page sharing.
  - Share-based allocation.
  - If (no memory pressure){
    do nothing;
    }else{
    try ballooning on a selected VM.
    tax a VM. Demand paging (Tax sort of decides priority).
    halt a VM when nothing else works and push it out to disk.
    }
- Admission control.