L03b. Memory Virtualization

Introduction:

- Memory hierarchy is very important when it comes to virtualization.
- Handling memory while we have multiple OSs running on the same HW has a great effect on performance.
- Recall that each process lives in a separate HW address space, and the OS maintains a page table containing the mapping of the virtual and physical addresses. The virtual address space of a given process is not contiguous in physical memory.

Memory Management:

- There're user processes running inside each OS running on top of the Hypervisor. Each of these processes has its own protection domain and a distinct page table.
- The Hypervisor doesn't have any information about these page tables.
- Each guest OS thinks that its memory regions are located contiguously in the physical memory, where in fact it's non-contiguous in the machine memory controlled by the Hypervisor.
- Even if these regions are contiguous, they can never start at the same address the OS thinks they start with.
- Shadow page table:
  - The process's page table, maintained by the guest OS, provides the mapping between the virtual page number and the physical page number.
  - The shadow page table (S-PT), maintained by the Hypervisor, provides the mapping between the physical page number (PPN) to the real machine page number (MPN) of the HW memory.
  - This adds a redundant level of indirection between the virtual address and the memory address.
  - How to make address translation efficient in a full virtualization setting?
    1. Whenever an OS tries to update its page table, a trap will be generated (Privileged operation).
    2. This trap is caught by the Hypervisor, which as a result updates the shadow page table.
    3. Now whenever a process generates a virtual address, the Hypervisor will directly translate this virtual address to a machine address, without going through the guest OS.
  - In a para virtualization setting, the guest OS knows that it's physical memory isn't contiguous, so the translation process is shifted to it:
    1. A guest OS can issue a "Hypercall" to the Hypervisor to **create** (allocate and initialize) a HW page frame and the guest OS can target this page frame to host a page table data structure.
    2. When a process starts to run, the guest OS issues another Hypercall to the Hypervisor to **switch** the page table to the previously given location.
    3. The guest OS can also **update** this page table.

Dynamically Increasing Memory:

- If a guest OS starts running an application that needs more memory, the Hypervisor can take back a memory region from another guest OS and provide it to the requesting OS.
- This concept can lead to unexpected behavior, if that OS was using the memory that was taken.
- Ballooning:
  - Instead of ripping the guest OS of its memory, another idea is to ask the guest OS to voluntarily give away memory.
  - A Balloon is a device driver that the Hypervisor installs inside the guest OS to manage memory pressures.
  - Whenever a guest OS requires more memory, the Hypervisor will contact the Balloon inside any guest OS that is not using the whole memory given to it through a private communication channel and tells it to inflate.
  - This means the Balloon will start requesting more memory from the guest OS. The guest OS will page out to disk to free extra memory for the Balloon driver.
  - Once the Balloon driver gets the memory, it returns it back to the Hypervisor, which in turn gives it to the guest OS requiring more memory.
  - Whenever the memory pressure situation is fixed, the Hypervisor instructs the Balloon to deflate, giving out more memory to the guest OS that provided the memory in the first place.
  - The Ballooning technique is applicable to both full and para virtualization.

Sharing Memory across VMs:

- Sharing memory between VMs can facilitate maximum utilization of the physical resources, but on the other hand, it might affect protection.
- If we're having different instances of the same application running on different guest OSs, the core pages of these instance can share the same physical memory page.
  - Solution #1: The guest OS has hooks that allows the Hypervisor to mark pages as copy-on-write and have the physical page numbers point to the same machine page.
  - Solution #2:
    1. The Hypervisor (e.g. VMWare) has a data structure (hash table). This hash table contains a content hash of the machine pages.
    2. For each physical page (on the guest OS), the Hypervisor creates a content hash and searches the hash table for a match.
    3. If a match found, the Hypervisor performs full comparison between the contents of the matching pages.
    4. If the two pages have the exact same content, the Hypervisor modifies the PPN to MPN mapping of the guest OSs to point to a single machine page.
    5. This machine page will be marked as copy-on-write.
    6. A reference count in the hash table is maintained to indicate how many guest Oss are using the same machine page.
    7. This approach requires no modification to the guest OSs.

Memory Allocation Policies:

- Pure share-based approach: You have control over the resources whether you're using it or not.
- Working-set based approach: You get as much resources as you need.
- Dynamic idle-adjusted shares approach: If you're not using some of the resources you have, a percentage of these resources will be taken away from you.
  - Making this "tax percentage" less than 100% allows for sudden working set surges.