L03c. CPU & Device Virtualization

Scan me

Introduction:

- In addition to memory virtualization, the CPU and devices need also to be virtualized.
- Virtualizing the CPU and devices will be more challenging since that they're more explicit to the guest OSs living on top of the HW.

CPU Virtualization:

- The Hypervisor must give the illusion to each guest OS that it owns the CPU:
  - The Hypervisor allocates a certain amount of CPU time to each VM.
  - The Hypervisor doesn't care about how each VM uses the CPU during its allocated time.
  - Similar to memory allocation, the Hypervisor can use different policies for CPU allocation:
    1. Proportional share: Each VM has a share of the CPU proportional to the processes running inside it.
    2. Fair share: An equal share to each VM.
  - The Hypervisor shall reward the guest OS for any CPU time taken from its share to serve another guest OS (e.g. interrupt coming for a VM other than the one owning the CPU at the moment).
- The Hypervisor must also handle program discontinuities:
  - These discontinuities include external interrupts, exceptions, system calls, page faults, etc.
  - The Hypervisor packs these discontinuities as software interrupts and deliver them to the corresponding guest OS.
  - The guest OS may require privileged access to handle some of these events.
  - Whenever a guest OS tries to execute a privileged instruction, it will produce a trap, which will be handled by the Hypervisor.
  - The problem occurs in a full virtualized setting if the privileged instruction that the guest OS is trying to execute **failed silently**.
  - To deal with this problem, the Hypervisor should search the guest OS's unchanged binary for such silently-failing instructions and do binary rewriting to avoid such issues.
  - In a para virtualized system, the Hypervisor provides APIs to the guest OSs to facilitate communication between the guest OS and the Hypervisor.

Device Virtualization:

- Similar to CPU virtualization, the Hypervisor must give the illusion to each guest OS that it owns the I/O devices.
- The two things we need to worry about are:
  - Data transfer.
  - Control transfer.
- Full virtualization: The guest OS already thinks that it owns the devices.
  - Control transfer from the guest OS to the Hypervisor: When the guest OS tries to access the devices, a trap will be issued to the Hypervisor.
  - Control transfer from the Hypervisor to the guest OS: The Hypervisor will emulate the functionality that the guest OS intends for the device.
  - Data transfer: The data transfer happens implicitly through the Hypervisor.
- Para virtualization: The guest OS in fact can see the exact same I/O devices that are available to the Hypervisor.
  - Control transfer from the guest OS to the Hypervisor: When the guest OS needs to access the devices, it issues a Hypercall to the Hypervisor.
  - Control transfer from the Hypervisor to the guest OS: The Hypervisor serves these Hypercalls trough software interrupts.

    NOTE: The guest OS has control (via Hypercalls) on when event notifications (SW interrupts) should be delivered.
  - Data transfer:
    1. The Hypervisor exposes shared buffers to the guest OS to facilitate passing data from the guest OS to the I/O devices without the overhead of copying these data between different address spaces.
    2. The Hypervisor must account for the CPU time needed for demultiplexing the interrupts and managing the buffers for the guest OSs.
- Data transfer in Xen:
  - Xen provides an asynchronous I/O rings that are shared with the guest OSs.
  - Each I/O ring with be populated with the I/O requests with the guest OS owning the ring.
  - Xen checks the ring pointer and picks the I/O request to be processed in a FIFO manner.
  - Xen will then place the responses to these requests in the same ring.
  - Network (and disk I/O) virtualization:
    1. Each guest OS has two I/O rings, one for transmission and one for reception.
    2. The guest OS transmit packets by enqueuing packet descriptors in the transmission ring through Hypercalls.
    3. These requests in the ring data structure will be pointers to the guest OS buffers to avoid copying the packets themselves.
    4. Xen uses a round robin schedular to provide transmission services to all the guest OSs.
    5. Packet reception works in the same way. But in addition to the buffer/pointer method, Xen also supports swapping the machine page that has the received packet with a page that the receiving guest already owns.