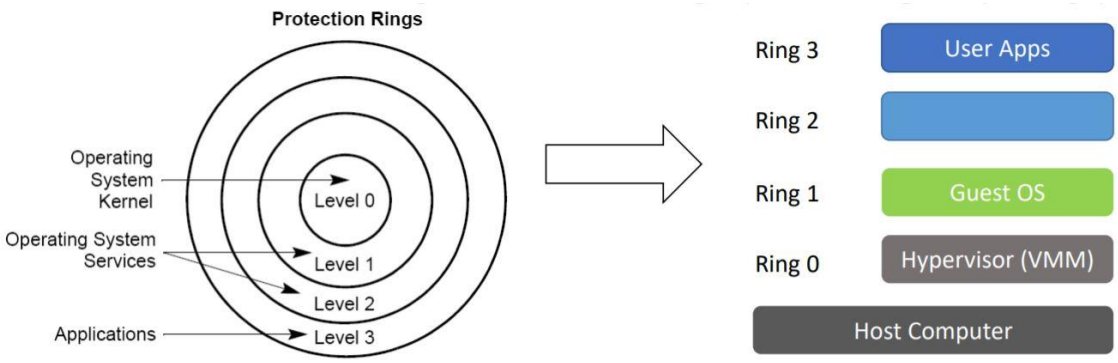
CS580K CLOUD COMPUTING :B00818310

Q1.

(1) Please describe the high-level design ideas to virtualize CPU, given that all of the sensitive instructions are the privilege instructions? (6 points)
Solution:




Some instructions cannot be executes directly by Guest OS since they can only be executed in RING 0., and if we keep guest OS in ring 0 , it may not coordinate with native/host os.
For example, there will be security breach issues, processed will be killed, etc. We cannot allow the guest os to have too much control.

However, what we can do is, we can keep the hypervisor in Ring 0 and the guest os in ring 1 as shown in figure.

This way, it will cause a trap/execution in privilege level instruction other than ring 0, that is in the guest OS.

Then, the hypervisor will be able to interpret such instructions by notifying it to the OS.

Then , the underlying OS is able to modify the privileged instructions once it gets notified.

(1) Please describe the high-level design ideas to virtualize CPU, given that only part of the sensitive instructions are the privilege instructions? (6 points):

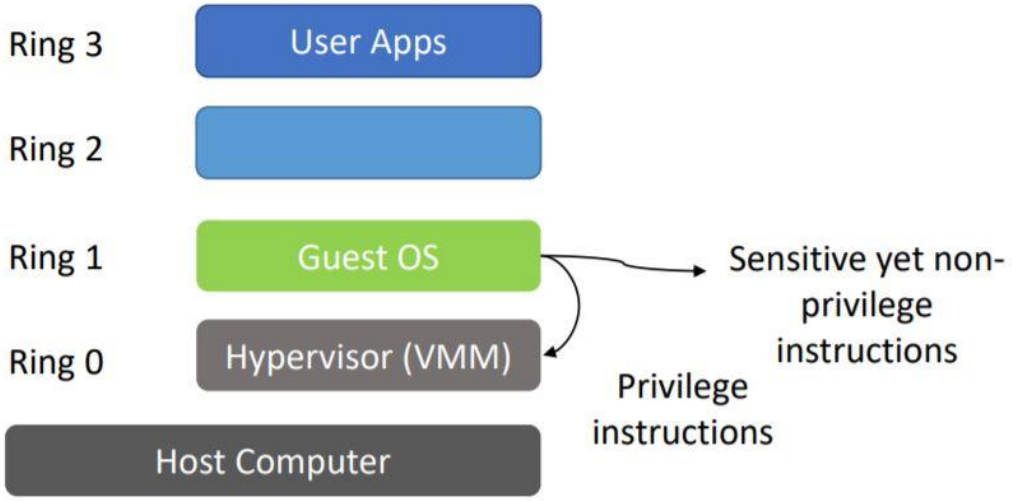
Solution :
Unlike privileged instructions , sensitive instructions won't impact other processes but they should not be able to directly execute in VM virtualization since it may be able to manipulate VM's processes.

Here too, guest os is in ring 1 and hypervisor in ring 0 .

The solution is to do emulation - binary translation , if we can capture each instruction before it is sent to the CPU.

With the help of an interpreter, the instructions will be interpreted and translated into a binary code.

Interpreter will trap sensitive instructions and interpret the. Here, interpreter will behave like hypervisor. However , some files and not easy to emulate.



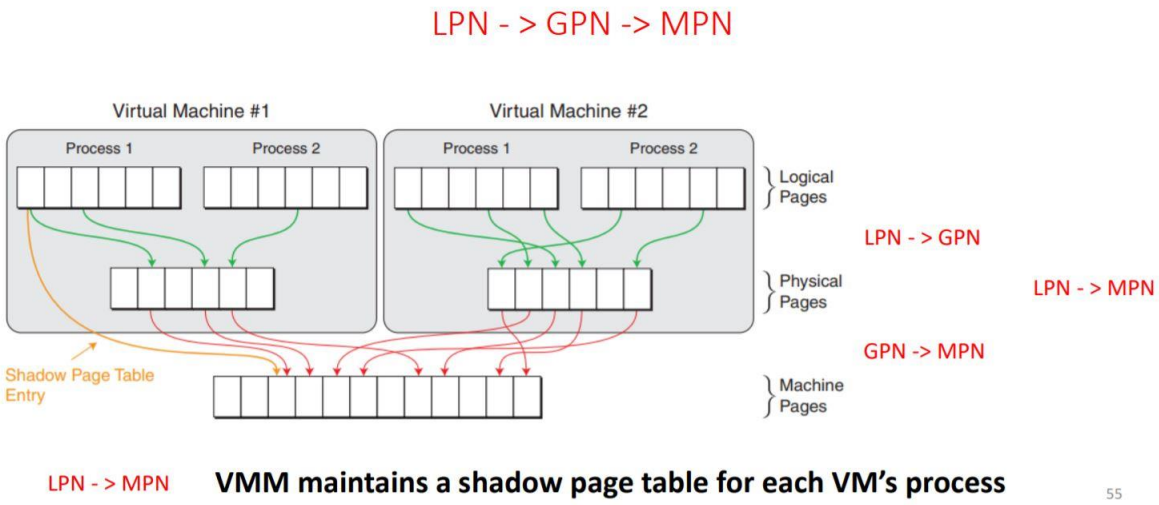

Drawbacks : Low performance due to much translation

Much overhead.

(2) Without hardware MMU (i.e., memory management unit) like Intel EPT, how do you use a software manner to realize memory management in virtualization — mapping the address space of a process running in a VM to its machine's physical memory? Please describe the high-level design ideas. (6 points)

Solution :

Use a shadow page table :




We can use a shadow page table to realise memory management in virtualization.

Traditionally, there will be page tables, one process page and one VM page table.

However in shadow page table, the LPN(process namespace) and GPN(Guest VM namespace) page tables are merged together, which is the shadow page table

Here,

Shadow page table needs to look at all the updates for 1st level and 2nd level page tables.

Hypervisor maintains page table for each VM. Any update to the 2nd level page table can be easily captured by the hypervisor.

For the notification mechanism, we can leverage the page fault mechanism, that is, every time VM tries to access , a page fault will occur.

Q2 .

(1) Please state the main sources that negatively impact I/O performance in KVM or Xen based virtualization solutions in comparison with the native (i.e., no virtualization). (6 points)

Solution :

1. Xen split driver model:
   The driver model uses a split driver model.
   It uses a round robin mechanism which has a large effect on latency.
   For example, as packets arrive delay will be proportional to the number of runnable domains(VMs)
   If dom 0 is up and running , interrupt will be sent in a queue.
   Thus, depending on the VM's interrupt will be delayed

2. MMU transition also creates some I/O delay. Which should be improved

3. Small scheduling time splice :
   If we have small time splice, then context switch frequency increases.
   This results in high overhead.
   This will in turn give rise to cold start penalty for next process

4. Running I/O domains with high priorities :
   This is because I/O intensive workloads may frequently interrupt CPU intensive workload leading to high context switch and high overload.
   This may again lead to problems like cold start penalty for next process.

(2) To achieve high I/O performance under Xen/KVM virtualization, what are the possible optimization solutions, and why do you think these solutions will work? (name three) (6 points)

Possible solutions:

1. Para Virtualisation in KVM:
   Basically, we can make any request go through qemu and qemu emulates I/O accesses back and forth to guest in a simpler manner unlike that of previous which lead to Guest -> KVM kernel -> QEMU -> OS and reverse.
   This will much reduce the over head

2. vHost :

Here, vHost puts virtual environment into the kernel, taking qemu out.
This allows device emulation code to directly call in to kernel subsystems instead of performing system calls from user space.
Transition: Guest process -> Guest OS -:> native
It will achieve close to native performance.

3. Hardware support :
   It allows to bypass kernel to directly access VT technology .
   However, here hardware support is needed.We cannot run without hardware support.
   We can design a hardware device directly to guest for our VM.

(3) Under a round-robin vCPU scheduler, suppose the time-slice (i.e., maximum time that a vCPU can run) is 30 ms. Four VMs (VM1 to VM4, each with 1 vCPU) share a single CPU. When a client sends an I/O request to VM1, how long does the client expect to receive the response from VM1 in the worst case? (4 points)

Since each VM have a time splice of 30 ms ,

IO wait time = No. of virtual machine * schedule time splice

IO wait time = 4 * 30

= 120 ms

Thus, in the worst case, the client can expect to receive response in 120ms

What are the results if the time-slice is 10 ms and 1 ms, separately? (4 points)

1. IO wait time = No. of virtual machine * schedule time splice

   = 4*10

   = 40 ms

2. IO wait time = No. of virtual machine * schedule time splice

   = 4 * 1

   =4 ms

What are the gains and losses using a small time-slice (e.g, 1 ms) in comparison with a large one? (4 points)

If we have small time splice, then context switch frequency increases.
This results in high overhead.
This will in turn give rise to cold start penalty for next process
However if we have large time splice, we will get I/O delay.

(4) We do want to schedule I/O-intensive VMs with small time-slice and CPU-intensive VMs with large time-slice. Please design a vCPU scheduler that can approximately achieve this goal. Please make reasonable assumptions and justifications. If you want to distinguish types of VMs (e.g., I/O-intensive or CPU-intensive), please describe how. (8 points)
We can use mixed model for this.
There will regular cores and turbo cores.
Turbo cores will be used for priority high intensive I/o tasks.
For CPu , regular can be used.
We can distinguish using the cores.

3. (42 points) Question 3 – OS-level Virtualization!
   (1) What are the two systems-level techniques that underpin containerization? (6 points)

Two main underpinning techniques :

Namespace and Resource Limits :

1. Namespace :

   Namespace provides lightweight process virtualizarion.

   Namespace allows a process or a group of processes of a container to have different views.

   There are 6 main namespaces :

   - mnt
   - pid
   - net
   - ipc
   - uts
   - user
2. Resource Limits :
   Also known as cgroups :
   It is a subsystem and a resource management solution.
   The implementation of cgroups requires a few tweaks into the kernel and none of them in performance critical paths

(2) Why is I/O performance under containerization much better than KVM or Xen based virtualization? (If you don't think so, please also justify.) (6 points)

Performance of containers is very close to that of native OS because running applications inside containers is just like running processes.

These processes will be regulated by cgroups.

While, XEN or KVM are system level virtualization techniques, and thus they must have other system level properties such as CPU scheduler, MMU , etc. Answers previously mentioned give rise to I/O latency.

(3) Then would the I/O performance of containers be the same as the native case (i.e., running processes directly on an OS), and why? (6 points)
Yes, as described before, running a container is just like running a process.
Hence, running a process in a native OS will definitely be near to running a container.

(4) Containers are much light-weight in comparison with machine-level virtualization solutions like VMs. However, in practice we do not completely replace VMs with containers. What are the restrictions of using containers? Or in what scenarios, it's better not to use containers; instead, we should use KVM or Xen based virtualization? (You can list the main drawbacks of containers, and figure out the scenarios unsuitable for using containers) (6 points)

Solution:

Containers are not safe.

They share the same host kernel.

There is weak isolation.

It can break the current system isolation and can access the underlying OS since container is a process.

It lacks kernel level development support.

(5) gVisor (Google's container solution) provides another isolation mechanism for container, which intercepts application system calls and acts as the guest kernel. Instead of passing system calls to the native kernel, gVisor implements a substantial portion of the Linux system surface. Thus each container may have its own user-level kernel. This design supposes to have better security properties than traditional ones (e.g., Docker containers). How does gVisor achieve this? (6 points)

Solution:

It achieves using sentry

Sentry is the largest component of gvisor.

System call are redirected to Sentry

Sentry will make some host system calls to support its operation, but it will not allow the application to directly control the system calls it makes

It does this using ptrace . It's a way to intercept syscalls.

(6) Actually, gVisor trades performance for such better security properties. Which types of applications do you think suffer less in terms of performance drop when we use gVisor containers, and why? Do you have some ideas to mitigate performance overhead of gVisor? (6 points)

Mitigation:

Rule-based execution (seccomp, SELinux, AppArmor )

- Allows the specification of a fine -grained security policy for an application or container
- In practice, not easy especially for unknow applications.

gVisor just mitigates but does not completely eliminate the security vulnerability – e.g., it is possible that if any one of the gVisor containers breaks out, it can allow unauthorized access across containers running on the same host. Please describe your ideas to further protect gVisor containers. (6 points)

We can use Kata idea,

Basically it runs containers inside VM.

There will be additional level of protection from hypervisor.

(8 points) Question 4 – Server Consolidation One typical usage scenario for containers and/or virtual machines (i.e., server virtualization techniques) is server consolidation. Server consolidation is an approach to place multiple virtualized servers (either in containers or virtual machines) on the same physical machine in order to reduce the total number of servers or server locations that an organisation requires. The practice originally developed in response to the problem of server sprawl, a situation in which multiple, under-utilised servers take up more space and consume more resources than can be justified by their workload. Later, the concept of server consolidation has been well materialized in cloud platforms where a single physical machine is shared by multiple cloud tenants. Figure 1 shows the total performance (summed from all VMs running on the same physical host) changes as we consolidate more VMs on the same physical host.

(1) As you can observe, after point 1 in Figure 1, the total performance stop increasing. Can you guess why? (4 points) (2) After point 2 in Figure 1, the total performance start dropping with more VMs, and why? (4 points

Solution:

Because, as VM increase, the Vm will compete for each other.