The architecture of virtual machines

1

Hierarchies in computer systems mean different levels of abstraction. These abstractions hide some functionality beneath them and users atop of these abstractions use them via interfaces.

Examples of these are for example user level applications and high-level languages that are running on interpreters.

User doesn't know, what exactly goes on under the hood of the applications they use work.

Also high-level languages, ie. python, that are interpreted are not aware of the hardware they are run on. The interpreter takes care of the commands that user writes and produces the output.

2

Interfaces define functionality of some parts of the systems, that other parts don't need to implement but can access and use.

Levels of abstraction means wrapping some functionality inside other functionality, that uses wrapped functionality. When some part of the system then uses the functionality that has some other functionality wrapped inside it, the user don't need to know about it or use it.

3

Interface 4 consists of user ISA and interface 3 is a superset of interface 4 ISA and it also consists of system level ISA as well. Interface 3 includes those aspects visible to OS hardware resource manager, but interface 4 doesn't have those visible to it.

4

for level 3 implementation of ISA that takes care of storing data to persistent memory, ie. disks or SSDs. For users these are invisible and only exposed via interfaces.

for level 4

5

The question seems to be that can application be written so, that it hijacks some part of the physical computer, ie. CPU, and use it as it seems fit. Yes, because OS couldn't detect if some other process is accessing the some recourse it is using.

6

Considering "Hello, world" C program, that needs to be run on writers own machine.

In Figure 4a code is compiled to intermediate to Object code, that ca be loaded into memory and run in the machine. The Object code in this case is writers machine architecture dependant, so for example it is written

in x86 architecture machine and therefore it cannot be transfered as is to ARM machine and run there.

In Figure 4b code is compiled in VM to portable code. This code can be then transfered as is to other VM and run there. Compiled code doesn't care what the underlying machine is, VM takes care of interpreting or compiling the code for underlying machine.

7

docker overview

Docker uses linux namespaces to provision slices of host OS where containers run. So container application run on host OS.

Docker containers can run any containerized OS, ie. Alpine and Arch linux. Therefore they are classical VMs.

8

Comparing docker on bare-metal vs. VMs

1. Easy portability between physical machines. In need of switching from machine to machine, ie. outdated hardware, one can easily create an image on running VM and using that image spin up new VM on new machine with the same configurations.
2. Consistent environment. There is no need to port container runtime to work on special OS environments, when using only one kind of VM runtime.

9

1. Underutlization of resources. Introducing another layer of abstraction between application and hardware increases the time of hardware doing nothing.
2. Special hardware. VMs can't usually access special hardware out of the box, ie. GPUs or TPUs. these require extra steps and priviledges to acces these.