1 Virtual Machines

a) List all machine resources that must be virtualized and discuss why. List some uses of virtual machines.

To a hypervisor, the different guest operating systems are like different untrusted user applications to a normal operating system. So any resource that needs to be protected from untrusted applications must be virtualized by the hypervisor such as memory, CPU, MMU, devices, network.

Virtualization has been used for server consolidation, better performance isolation, running legacy code, building & debugging operating systems, etc.

b) What is the difference between Native (type 1) and Hosted (type 2) virtual machines? Which might be more suitable for a data center (cloud provider) and which might be more suitable for your laptop?

The native type virtual machines run on bare-metal. They are more suited for a cloud provider as there are fewer layers between the hardware and the application. Note that on these types of virtual machines, one guest OS is privileged (it can be used to configure the VM).

The hosted type runs on top of an existing operating system. This is more suited for your laptop. Your native OS is probably the one you use most regularly switching to the guest to run special cased software.

c) Discuss the differences between and the (dis)advantages of full- and para-virtualization.

In full virtualization, guest OSes don’t know (or shouldn’t know) that they are running on virtual hardware rather than real hardware. In para-virtualization, guest OSes are aware of the virtualization layer.

In full-, guests need not be modified, whereas in para-, guests have to be modified. The modification of guest allows it to run more efficiently on virtual hardware.

d) Discuss the functionality of ballooning. What problem is it trying to solve and how does it solve it?

Ballooning is used reclaim memory from guest OSes. Classical operating systems use the MMU to reclaim memory from applications. When the system runs low of memory, they reclaim paged memory from applications. When applications try to access it, they page fault and the memory is returned. In the absence of such hardware support, virtual machines use ballooning.

A balloon is installed on every guest OS. To reclaim memory from the guest, the balloon expands taking up memory from the guest and returning it to the hypervisor. To return memory to the guest, the balloon contracts.
2 Network stack

a) Name one case of multiplexing and demultiplexing that happens at the level of a modern NIC

In most Operating Systems, the driver manages the NIC queues and there is no correspondence between applications and NIC queues. Following this fact, a queue in many cases will be multiplexed between different Applications that require sending packets.

When a packet is received, it will be demultiplexed to one of the receive queues.

b) What are potential problems that can arise when the TCP protocol state is stored in the kernel?

Each connections required state which needs to be stored somewhere. In the UNIX example (and most other OSes) of the lecture, the state is stored in the Kernel. If a server now receives many requests to set up a TCP connection, there are two possible ways of handling this. First, we can try to accept all the requests and allocate memory in the kernel potentially running out of memory. Second, the number of connections we accept is limited and also the memory that is required. If the number of connections is limited, it can also potentially lead to denial of service attacks (TCP SYN flood attack)

There is also a performance problem. For each connection we require a system call to allocated the state. Depending on the operating system, system calls can be expensive and a limiting factor on how many TCP connections per second a server can accept. Number of TCP connections per second is a metric that is also often used to show the scalability of a network stack.

c) What is the difference between Receive Side Scaling and Flow Steering?

Flow steering uses a table to figure out to which core a flow should be steered while RSS uses a hash function. Flow steering is normally implemented in software while RSS is implemented in the NIC hardware using a predefined hash function.

d) Receive Side Scaling assumes that one core is as good as any other. When does this assumption not apply? Why can this hurt performance?

Receive Side Scaling will balance the flows on any number of cores and does not care if these cores belong to another NUMA node. As soon as we receive the data on a core of an other NUMA node, it will increase the latency and might limit bandwidth. Each PCI card is attached to a certain NUMA node. As long as the cores that are use for RSS are on the same NUMA node, the locality is given and most of the data structures etc. will be in cache. If the core is not part of the same NUMA node the packets will go through memory across the NUMA nodes.

e) What are problems that might come up when implementing a zero-copy (no copies of the packet payload) network stack that uses only a single buffer instead of a linked list like structure of the mbuf chain?

First of all, when sending a packet the payload has to be written at a certain offset of the buffer so that the header still has enough space. The distances actually depends on the packet type an other information so it is only possible to know this size if we exactly know the type of the packet. Alternatively there are maximal header sizes which we can expect per packet layer and we can simply always assume the offset to be at least as large as all the maximal header sizes combined

Second, we no longer have the possibility to just extend the buffer’s size by adding an element to the linked list (or adding another fragment of an IP packet). In general buffers used by the network stack are preallocated for performance reasons as a malloc() on the fast path is not desirable. IP packets can be up to 65536 bytes and always allocating a buffer of this size will result in low memory utilization as most packets are small.

f) In the script we investigate the time the OS has to process a packet of a modern 10 Gbit NIC. We will now compute some of the numbers for a 100 Gbit NIC. Assume the processor runs at 3 GHz and the packet size is 1536 bytes. For conversions use base 10 (i.e. 1 GB/s = 1000 MB/s).

- Compute the number of packets per second as well as the time a CPU has to process a packet for a 100 Gbit NIC.
- How many CPU cores (rounded to the next natural number) would it take handle the 100 Gbit NIC assuming we can process a packet on average in about 1800 cycles?
• 100 Gbit/s converts to 12.5 GB/s or 12'500'000'000 bytes/s (The numbers in Networking use base 10 and not base 2 i.e. 12.5 GB/s = 12'500 MB/s = 12'500'000 KB/s ...). As mentioned in the description the packet size is 1536 bytes which results in 8'138'020.83333 packets per second. In this scenario the processor has \( \frac{1}{8'138'020.83333} \) s per packet which results in 122.88 nano seconds per packet.

• A 3 GHz processor executes 3 cycles per nano second. From the previous subquestion we computed that every 122.88 nano second a packet arrives. We can processes a packet in 1800 cycles which is 600 ns. From this we can conclude that we need \( \frac{600}{122.88} = 4.88 \) CPU cores or when rounding up 5 CPU cores.