Principles of Computer Systems
Midterm Exam

1-Nov-2018

This exam has 7 questions, totaling 100 points. You have 105 minutes to answer them, which means you earn about 1 point per minute of work – please consider spending on each question no more minutes than the number of points attributed to it.

If you exit the room during the exam, you will have to turn in your exam, and you will not be permitted to return to the room until the end of the exam. Please plan accordingly.

You are allowed to have any amount of printed material you like (books, papers, notes) but no laptops, tablets, cellphones, etc. are permitted during the exam. You must take the seat assigned by the course staff and present your CAMIPRO card to the staff upon request.

Do not open the exam until instructed to do so.

Your name: _____ SCIPER: _____

Question 1 *(8 points)*

Most RPC designs handle the no-response case by choosing one of three strategies: at-least-once, at-most-once, exactly-once. For the following scenarios decide which strategy is the most appropriate. Put exactly one checkmark on each row, in the column corresponding to the most likely answer.

Scenario	At-least-once	At-most-once	Exactly-once
An RPC that increments a counter			X
An RPC that invalidates an entry in a distributed caching system	X		
A read request from a key-value store that acts as a caching layer to some back-end storage	X		
An RPC that retrieves content for your Facebook feed		X	

Explanation (not required for answering the exam):

An RPC that increments a counter should be executed exactly once because otherwise the counter will end up with the wrong value.

An RPC that invalidates an entry in a distributed caching system should be executed at-least once. If it's executed at most once there is a chance that the cache will reply stale-data. Executing exactly once is expensive. Also, executing more than once can only harm the performance but it is semantically correct.

A read request from a KV-store that acts as a cache should be executed at-least once. Reads are idempotent, so executing the same read request multiple times is semantically correct, while this read request requires a reply since all requests are served through the KV-store.

Your facebook feed is a latency critical workload without any consistency guarantees. So, an RPC retrieving content for the Facebook feed should be executed at-most once and within some latency service level objective. There is no need for implementing the expensive exactly-once semantics, while not returning the necessary content is acceptable.

Grading: 2 points per row

Question 2 *(12 points)*

One advantage of a microkernel over a monolithic kernel is that it reduces the load on the translation lookaside buffer (TLB), and thereby increases the TLB hit rate, which in turn improves performance. True or False? Explain in max. 1 paragraph.

Answer:

False.

The primary performance impact comes from the fact that microkernels run each kernel module/service in a different address space, and the TLB needs to be flushed upon context-switching (assuming no PID-tagging). This reduces the TLB's overall hit rate, thereby making end-to-end performance of applications suffer, since they and the kernel services must perform more page walks than if they were running on a monolithic kernel.

A secondary performance effect comes from it being likely to have more page mappings overall, given that there is more code running in more address spaces. Thus, regardless of whether the TLB is PID-tagged or not, it will need to accommodate more mappings, which increases pressure on the TLB and thus can lead to more misses.

If the TLB is PID-tagged, then the primary effect above disappears, and only the secondary one remains.

Grading:

- *4 points for correctly answering True/False*
- *8 points for the Explanation*
    - *If you described the primary effect, then you received full points.*
    - *If you only described the secondary effect without mentioning how PID-tagged TLBs make the primary effect go away, then you received 4 points for the explanation.*

Question 3 *(8 points)*

DNS is used extensively in DDoS attacks. Provide some reasoning behind why DNS would be a favorite for attackers. (Maximum 2 paragraphs).

Answer:

DNS is favored by attackers because of favorable imbalance: with little effort, they can inflict a disproportionate amount of damage. There are two reasons for this:

1. DNS plays a key role in communicating over the Internet, because hosts typically need to contact DNS every time they initiate a conversation with another host; hence, by disrupting DNS, an attacker can disrupt the ability to access content and services for a significant number of hosts around the world.
2. DNS has a hierarchical structure, so making the DNS root servers and/or top-level domain (TLD) servers inaccessible/slow will disrupt/slow down DNS as a whole, and thus all hosts that use DNS.

Additionally,

- it is relatively easy to impersonate a DNS server and provide false mappings, because DNS uses UDP as its transport-layer protocol; and
- DNS interactions typically involve one DNS request and one DNS response.

These two factors enable at least two types of attacks:

1. Several malicious end-hosts M1, M2, ... spoof a victim end-host V's IP address and sends a high rate of DNS requests, which causes V to receive all the responses and be crippled by their high rate of arrival. This is called a reflection attack. Such attacks can be amplified by querying, for example, for all known information about a DNS zone in a single request, making the size of the response be an order of magnitude greater than the size of the request.
2. If a malicious end-host M can guess when victim end-host V will send a DNS request to resolve name N, it can spoof (at the right moment) the IP address of V's local DNS server and send to V a fake DNS response before V receives the true response. As a result, V maps name N to an incorrect IP address, typically controlled by M, and contacts that fake service.

Grading:

- *Any suitably explained combination of enabling DNS feature + enabled attack received full credit. E.g., pointing out how the connectionless aspect of UDP enables the reflector attack was worth 8 points.*

Question 4 (*20 points)*

Question 4.A (10 points)

Provide three key properties of a good naming system, and then describe how they are achieved in Lampson's Global Name Service (GNS)[1]. Max. 1 paragraph per property.

Answer:

Here are some of the properties:

- Scalability of management (i.e., managing the naming system should not become harder as more mappings are added to it). GNS achieves this by using a hierarchical name space and local names. As a result of this design choice, each directory may belong to a different administrative entity, and each entity that owns a directory can add mappings to it without coordinating with other entities.

- Well-defined behavior (i.e., given the mappings stored in the system at time T1, and a sequence of client operations that start at T1 and end at T2, it should be clear what mappings should be stored in the system at T2): GNS achieves this by making client operations commutative (i.e., their order does not matter) and idempotent (i.e., repeating the same operation does not change the system's status).

- Reliability (i.e., a server crash should, with a significant probability, not lead to loss of data):GNS achieves these through replication of each directory across multiple servers.

- Availability (i.e., a client should be able to obtain a mapping with a very high probability): also achieved through replication.

- Reasonable performance (i.e., a client should typically not need to contact multiple servers in order to resolve a name): GNS achieves this by allowing clients to cache mappings until they expire, according to an assigned per-mapping expiration date.

Grading:

- *10/3 points per property.*
- *The explanation of how the property is achieved is worth half the points.*

[1] "Designing a global name service" by Butler W. Lampson, PODC 1986

Question 4.B (10 points)

Consider the case where you have a global**,** flat namespace which is managed by multiple administrative entities. Does the flat namespace provide any challenges in managing the namespace? Describe how a client can identify the server/directory that resolves a particular name. (Maximum 1 paragraph)

Answer:

The main challenge with a flat namespace is simultaneously achieving scalability in terms of management, scalability in terms of performance, availability, and well-defined behavior. If the naming directory is kept on a single server, then it's easy to manage and the behavior is well-defined, but performance does not scale well with the size of the directory and the number of clients, and availability is (probabilistically) easier compromised. Keeping the directory on multiple servers, as intended for this problem, helps scale performance and improves availability, but now management is harder and providing well-defined behavior is more difficult, because the flat namespace does not provide a “natural” way of determining which server handles the mapping of a given name, thus the mechanism for finding a mapping, deciding where a new mapping needs to be placed, and updating a mapping are trickier.

One way to solve these challenges is by partitioning the namespace across system nodes according to a function F, and having the naming system provide the implementation of F. Each valid name is now "owned" by a particular node; this "owner" node stores the corresponding mappings and ensures consistency for these mappings. Whichever subdirectory is asked for a name would know how to route the request (according to F) to the node that is responsible for the mapping.

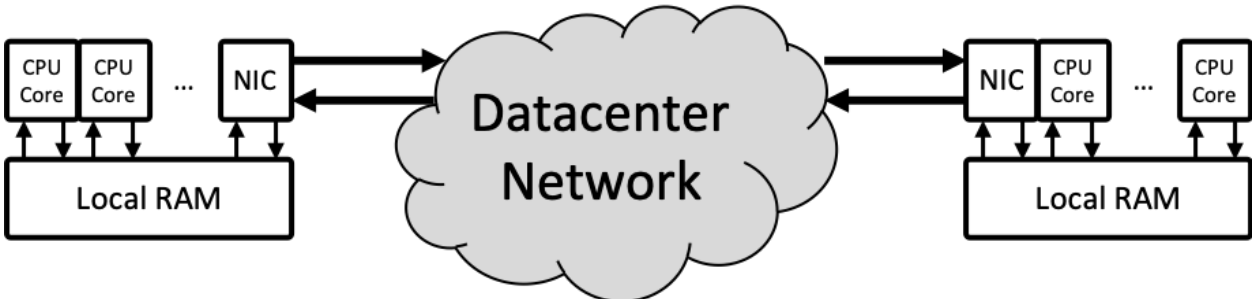
This is how peer-to-peer (P2P) systems typically work: When a P2P client wants to read content C, it first computes C's name N (typically by hashing C) and uses F to get the lookup request routed to the node that owns name N, and that node then maps N to the IP address of a node that stores content C.

Grading:

- *Identification of the challenges was worth 3 points.*
- *A good solution was worth 7 points. Some examples: ARP-like broadcast, Dynamo-like ring routing through hashing the name, hash name and ask an intermediate that maintains hash-to-nameserver mappings, etc.*

Question 5 *(32 points)*

You are the CTO of *REMLink,* a hardware startup that has prototyped a next-generation NIC that can access the memory of other servers in a datacenter without requiring the use of an RPC. The NIC is integrated into the server motherboard as an architectural block and is allowed to read/write local memory pages, as well as inject network packets destined for remote NICs, which in turn access their local memory and return the values across the network. The following diagram conceptually shows the *REMLink* platform.



Question 5.A (12 points)

Using the virtual memory system discussed in class (x86_64), describe a method by which REMLink can provide complete transparency to applications, i.e., enable a program to use both local and remote memory in the same way, without doing anything special.

Answer:

By adding information to the page tables, the cores and NIC can be made aware of the location of any page in question. This information includes both the page's status as either local or remote and, if remote, its location. When a CPU core accesses an address that resides in a locally hosted page, the system works as usual. If the page is remote, that will be signalled in the page table entry, and the core's page walk hardware can request the NIC to access that remote page on the corresponding server.

The page can either be copied into local memory for locality reasons, or left on the remote server. If it is left on the remote server, the CPU core will need to have tags added to its TLB to forward memory accesses to this page to the NIC, which will again access the remote page over the network.

Grading:

- *10 points for using PTEs and/or TLB entries to specify the location of accesses*
- *2 points for using the transparent remote-access capability of the new NIC.*
    - *Many answers simply suggested that the kernel should migrate pages to and from remote memory upon page faults. That is functionally correct but does not use the direct-memory access capabilities of the NIC in this question.*

Question 5.B (20 points)

In order to not corrupt data in remote memory or receive partial data in responses, any remote accesses must be performed exactly once. Your networking team has proposed that this guarantee can be met by providing a reliable link layer. Is this true? If so, describe a way to ensure that network packets/frames are reliably transmitted across links. If you think the statement is not true, then design a protocol to be implemented in the NIC that accomplishes the exactly-once semantics. Describe the protocol as precisely as you can.

Answer:

No, link-level reliability isn't enough. By the standard E2E principle, receiving the packets don't exclude other errors in the NIC such as bit corruptions, or deal with the issue of reconstructing the messages before writing them to remote memory.

An elegant solution is to use a hardware-implemented transport protocol that contains the message's remote memory address and unique id in each link-layer packet, so the NIC can buffer & reconstruct the message before writing it to remote memory. This transport protocol operates in a "request & response" fashion, where every packet from the sender is responded to by the receiver with either an ACK/NACK. Given that the links themselves are reliable and error-corrected, the receiving NIC will eventually receive each of the packets and send its replies, guaranteeing exactly-once delivery.

Grading:

- *5 points describing why link-layer reliability is not enough*
- *10 points functionality of solution*
- *5 points for correct layer implementation*

Question 6 *(10 points)*

An important principle of successful systems is to make the common case fast and the uncommon case merely correct. Lampson refers to this as the “separate the normal case from the worst case” idea, and then advocates handling them separately. Explain one way in which virtual machine monitors employ this principle.

Answer:

When executing a VM’s instructions, the VMM treats instructions that are neither sensitive nor privileged as the common case, and sensitive or privileged instructions as the uncommon case. VMMs optimize the common case for speed by allowing direct execution of regular instructions on the hardware, and ensure the uncommon case is correct by handling sensitive/privileged instructions with trap & emulate or binary translation.

Grading:

- *5 points for identifying common/uncommon case*
- *5 points for explaining how each one is fast/correct*

Question 7 (*10 points*)

VxLAN is a technology that encapsulates Ethernet frames in UDP datagrams. Analyze VxLAN from a layering perspective (i.e., discuss which layer(s) it concerns and whether you would view it as a layering violation) and provide a use-case where this technology is applicable.

Answer:

VxLAN is used to encapsulate a link layer protocol (layer-2) on top of a transport protocol (layer-4). In hardware-assisted VxLAN, switches perform packet forwarding (L2 operation) based on the UDP payload (L4). So, this can be considered a layer violation. In other cases though, the encapsulation can be implemented at the end-hosts and VxLAN packets are routed as normal UDP packets. This layer nesting can not be viewed as layer violation.

A common use-case for VxLAN is implementing private LANs in a public cloud infrastructure. The VM-hosts are in charge of doing the encapsulation based on source and destination VM.

Grading:

- *Link layer protocol on top of a transport protocol (3 points)*
- *Layering violation (3 points):*
    - *Yes if hardware assisted NAT where switches do forwarding based on the UDP payload*
    - *No, if a software approach is described where end hosts craft packets correctly so VxLAN traffic is routed as normal UDP traffic.*
- *Use-cases, e.g. virtual networking (4 points)*