Principles of Computer Systems
Fall 2013

Final Exam — Solutions

You have 105 minutes to answer the 5 questions in this exam. The entire exam is worth 100 points, which means you earn about 1 point/minute of work.

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.

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.

Please write your name and SCIPER below.

Do not open the exam until instructed to do so.

Name: _____ SCIPER: _____

The rest of this page reserved for grading

Q1		Q2		Q3	
Q4		Q5			

TOTAL

Question 1 (20 points)

Apply the end-to-end argument to Thompson's reflection on trusting trust: what are the end points, what are the layers in between, and what does the end-to-end argument push for in terms of design?

Answer:

In the context of Thompson's paper, the goal of a computing system is to transfer behavior (functionality) from the programmer to the user. In other words, there is some intended behavior (encoded by the programmer in the source code), and a correct system ensures that the actual behavior (experienced by the user when running the program) is identical to the intended behavior. Thus, the endpoints are the intended behavior (source code) and the actual behavior (execution of the binary). The intermediary layers are the compiler, the OS, the loader, the hardware, etc.

The end-to-end argument says:

1. The endpoints should not depend on the intermediary layers for the correct transmission of the intended behavior, but rather check correctness in the endpoints. This can be done via runtime verification against a specification, by bundling a correctness proof of the binary with the code, etc.
2. It is OK to place trust mechanisms in the intermediary layers, but only for performance optimization. For example, having the compiler generate a proof of the binary and bundling it with the executable might make it faster for the endpoint to validate the binary, because it is generally easier to check a proof than to generate it.

Question 2 (20 points)

Consider the problem of traversing a massive graph (terabytes in size) stored on disk (for example, to test reachability or connectedness). Use a back-of-the-envelope calculation to explain whether a strategy of breadth-first traversal is more (or less) efficient than depth-first traversal? What assumptions about hardware, software, and graph representation are you making?

Answer:

Assuming the traditional memory hierarchy, it is of utmost importance that the graph cut fits (as the working set) in the main memory. Since nothing more is said on the structure of the graph, we must consider a general case — i.e., a random graph; thus, the graph diameter is expected to be small. Even if nodes are indexed or ordered on disk, a breadth-first search will be superior over a depth-first search, because the former will minimize random access. Namely, the cost of a breadth-first search can be roughly expressed as the cost of the sequential scan of all the data on disk multiplied by the graph diameter. On the other hand, the cost of a depth-first search can be estimated as the number of edges multiplied by the cost of a random disk access, which greatly exceeds the cost of a breadth-first search.

Question 3 (20 points)

Assume that it is possible to achieve near-zero overhead dynamic binary translation. What is the impact of this technology on the Popek/Goldberg theorem?

Answer:

The Popek & Goldberg theorem gives sufficient but not necessary conditions to determine whether an architecture supports efficient virtualization. In order to build a VMM it is sufficient, according to the theorem, that all instructions that could affect the correct functioning of the VMM (sensitive instructions) always trap and pass control to the VMM.

Some architectures (e.g., x86 machines that lack hardware-assisted virtualization) do not meet these requirements, so the classic trap-and-emulate virtualization is not applicable. But such architectures can still be fully virtualized using binary translation, which replaces the sensitive instructions that do not generate traps. If we assume near zero-overhead of binary translation then any architecture could be made virtualizable. This assumption effectively eliminates the distinction between the instructions that execute natively (i.e., efficiently) and those that are emulated, making the Popek & Goldberg theorem irrelevant.

Question 4 (20 points)

Identify how the CAP theorem is applied in OLTP systems. What is the most important protocol involved and how does it behave with respect to the theorem?

Answer:

Traditional relational database systems enforce consistency and availability while sacrificing partition tolerance. For example, the two-phase commit protocol (2PC), which ensures atomicity and consistency, cannot successfully commit a transaction that involves multiple sites in the presence of a network partition — it would just hang. A typical way of dealing with partitions is through data replication.

Strong consistency and synchronous replication in RDBMSs come at the expense of limited scalability: 2PC incurs high communication overhead, which in turn increases the latency of committing a transaction and reduces the systems throughput and availability.

Compared with distributed key-value stores, RDBMSs have an additional layer of operational complexity needed to ensure the atomicity of distributed transactions. The more nodes need to synchronize their activity, the longer such transactions will take time to execute.

Question 5 (20 points)

Lampson's Hints paper puts forth two seemingly conflicting principles: "Do not hide power" and "Keep secrets". Is Lampson contradicting himself or do you believe otherwise? Use one example from other papers or principles discussed to justify your argument.

Answer:

The "keep secrets" hint suggests not to expose implementation details through external interfaces in order to keep the freedom to change these details at any time without having to rewrite all clients that use the interfaces. The "do not hide power" hint recommends, on any given layer, to provide interfaces that allow to access the layer below without significant performance overhead.

Let us consider the Exokernel and libOSes as an example. The Exokernel API follows (and emphasizes) the "do not hide power" principle by exposing the lowest-level abstractions directly to the user-space applications. Changes in the hardware or within the Exokernel implementation would require changing the external interfaces and, hence, updating all applications that use them. However, to amortize this cost and be in accordance with the "keep secrets" hint, the Exokernel introduces libOSs in the system design. On one hand, a libOS talks to the Exokernel over a low-level implementation-specific API. On the other hand, a libOS exports high-level and stable APIs (such as POSIX) to the applications using the libOS. With such a two-layer design, most of the applications are shielded from changes in the Exokernel implementation by the libOSs. Only the libOSs themselves, and the applications that bypass libOSes in order to get extra powers of low-level APIs, are required to be updated upon changes in the Exokernel implementation.