Principles of Computer Systems

December 17, 2024

This test has 16 questions totaling 100 points. You have 105 minutes to answer them. If you find yourself spending on any given question more minutes than the number of points attributed to it, you might consider moving on to the next question. The exam has 8 pages in total, including this one.

There is a mix of four kinds of questions:

- *ONE*: One-sentence answers related to a system or concept discussed in class. ONEs allow you to demonstrate your ability to identify, synthesize, and clearly state the essence of what you have learned.
- *OPAR*: One-paragraph answers related to a system or concept discussed. These are similar to ONEs, but with an opportunity to include more ample details and go in more depth.
- *BOE*: Back-of-the-envelope calculations that assess your ability to perform the quantitative reasoning that underlies intelligent design decisions.
- *MCQ*: Multiple-choice questions where you must check 1 or more of the 5 options provided. No justification is required, and nothing other than the checked boxes will be taken into account during grading. The MCQs efficiently test the breadth of your knowledge, covering many topics in a short amount of time.

Your answer to each question must be entirely within the box provided on the exam sheet. You have a few spare sheets at the end of the exam. Anything outside the boxes will be discarded prior to grading.

The exam takes place in room BC 01, and you must be physically present to take it. The exam is a written exam, and you need to use your own pen. The exam starts at 11:15 and ends at 13:00.

Rules:

- The exam is closed-book. You are allowed to bring with you one double-sided A4 cheat sheet with whatever information you want on it, in whatever font size you like.
- You are not permitted to interact with or receive or give any assistance for the exam except with/from the course staff.
- If you are noise-sensitive, you may wear ear plugs or ear muffs, but no electronic device is permitted (no headphones, no earbuds, no active noise-canceling device, etc.).

Read each question carefully. You need to provide a correct and complete answer to the *correct* question in order to receive full credit. A correct answer to a *wrong or misinterpreted* question will not earn credit. If you have any doubts, raise your hand, and the course staff will come to help.

SCIPER ☐☐☐☐☐☐

Question 1 **(5 points)**
One performance concern with the microkernel architecture is that the latency cost of IPC is too high. Where does this cost come from, and how does Liedtke address it in "On µ-Kernel Construction"?

(Answer in 1 sentence)

The latency cost of IPC comes from user-kernel space crossings and address space switches. Liedtke proposes to use PCIDs to avoid flushing the TLB on context switches, and to use segmentation-based virtual memory instead of page-based.

Question 2 **(5 points)**
The RON paper says that BGP "achieves scalability at the cost of fault tolerance." Name one mechanism that BGP uses to achieve scalability, then state how this mechanism helps scalability and how it hurts fault tolerance.

(*Answer in 1 sentence)*

Route aggregation helps scalability by reducing the size of forwarding tables, and it hurts fault tolerance by hiding the existence of separate routes to different local networks.

(Also... delayed updates help scalability by reducing the number of updates and messages that each router must process, and they hurt fault tolerance by delaying the propagation of route withdrawals due to failures.)

(Also... coarse failure monitoring helps scalability because it requires each router to keep monitoring state per BGP peer (as opposed to state per route), and it hurts fault tolerance because it cannot detect when a route becomes unusable due to congestion.)

Question 3 **(5 points)**
How does Twizzler enable regular applications to access persistent objects via 64-bit pointers?

(Answer in 1 sentence)

Through indirection: Objects from storage are mapped into the address space on demand (a page table-like view object maps OIDs to their temporary in-memory location) and then, at runtime, the FOT maps a reference to an <OID:offset>, which the view maps to an application pointer [base VA + offset].

Question 4 **(4 points)**
Which of the following cache configurations could lead to aliasing problems (i.e., ~~multiple virtual addresses mapping to the same physical address~~ the cache cannot tell that two different virtual addresses actually map to the same physical address)?

(Circle all the correct options)

A. Virtually Indexed, Virtually Tagged (VIVT)
B. Virtually Indexed, Physically Tagged (VIPT)
C. Physically Indexed, Virtually Tagged (PIVT)
D. Physically Indexed, Physically Tagged (PIPT)
E. None of the above

Explanation: Remember the bits of an address are split into

```
[...page number...|......page offset.....]
```

A fundamental property of paging is that the page offset is not changed by VA-to-PA translation. Therefore, to be immune from aliasing, it is sufficient for a cache to be able to tell if two different cache entries correspond to locations in the same physical page or not. Virtually tagged caches are inherently unaware of which physical pages their entries correspond to, so VIVT and PIVT suffer from aliasing. Options (A) and (C) are correct.

Option (D) is trivially immune to aliasing, since the VA plays no role in identifying the cache entries.

VIPT introduces a twist: if we have VA_1 and VA_2, both mapping to a given PA, physical tagging will prevent aliasing problems as long as indexing the two VAs leads to the same associativity set. Remember that, to do a lookup in the cache, we divide the address into

```
[........tag........|..index..|..offset..]
```

The cache circuitry then uses the *index* to find the right associativity set in the cache, then uses the *tag* to select the right cache entry from within that set, and finally uses the *offset* to select the right memory word from within the selected cache entry.

In a properly designed cache hierarchy, the tag field will include *all* the address bits needed to distinguish pages, so the cache index and offset bits end up coming entirely from the page offset portion of the address (as illustrated above). Since VA_1, VA_2, and PP have the same page offset, VA_1 and VA_2 will therefore always

index into the same set if they map to the same PA. The physical tag will therefore be effective in preventing the cache from holding two different entries for the same physical address PA.

If, however, the tag field does not include all page number bits, then VA_1 and VA_2 could index into different sets, and physical tagging will be insufficient to distinguish the two entries.

Grading notes: Answering (A,B,C) or (A,C) earns 4 points, since the VIPT twist above is subtle. Answering only (A) earns 3 points, because missing PIVT is partly excusable, since it's so rare. An (A,B) answer without (C) earns 2 points, because it points to a certain amount of confusion around how tagging works. Answering (D) alone denotes a deep misunderstanding of what causes aliasing, so it earns 0 points.

Question 5 (4 points)
According to Popek & Goldberg's theorem, which of the following is true of an ISA for a conventional third-generation computer?

(Circle all the correct options)

A. If it has a sensitive instruction that is not privileged, a VMM cannot be constructed
B. If the ISA consists entirely of innocuous instructions, a VMM may be constructed
C. If all innocuous instructions trap in user mode, a VMM may be constructed
D. If behavior-sensitive instructions do not trap, it is not possible for the VMM to maintain control of all resources
E. None of the above

Explanation: Popek & Goldberg's first theorem says that "For any conventional third-generation computer, an effective VMM may be constructed if the set of sensitive instructions for that computer is a subset of the set of privileged instructions." Innocuous instructions are those that are not sensitive so, if all instructions are innocuous, the set of sensitive instructions is empty, which satisfies the theorem trivially, hence option (B).

The theorem implies nothing about when a VMM cannot be constructed, so option (A) is incorrect. Option (C), besides proposing a preposterous execution model, also allows sensitive instructions to not trap, and thus could break the equivalence or resource control properties. Option (D) is incorrect because behavior-sensitive instructions that do not trap compromise equivalence, not resource control.

Grading notes: A (B) answer earned 4 points. We awarded 2 points for (A,B), because it denotes a good understanding of the theorem, but misses the fact that the converse is not necessarily true. All other answer configurations denotes a misunderstanding of the theorem and/or key concepts related to the theorem

---

Question 6 (4 points)
PIPT caches are less prone to aliasing but can incur higher access latency compared to VIPT caches. Why?

(Circle all the correct options)

A. Because they require virtual-to-physical address translation before every access
B. Because they require larger page sizes for address mapping
C. Because they require complex hardware for virtual address translation
D. Because tags in PIPT caches slow down indexing more than in VIPT caches
E. None of the above

Explanation: Think of the index as the name of the associativity set. In a physically indexed cache, the index is computed off the physical address, so a PIPT cache must wait for the VA-to-PA translation (i.e., the VA name lookup in the TLB or page tables) before it can start indexing. In a virtually indexed cache, the name of the associativity set is already contained in the VA name, so a VIPT cache can start the indexing in parallel with the VA-to-PA translation. In other words, VIPT exploits the property that (in a well designed cache) the index is not changed by VA-to-PA translation. This makes option (A) be the correct one.

All the other options refer to aspects that are orthogonal to the type of cache: Page size is not inherent to the cache; the complexity of address translation hardware is unrelated to the cache (the main difference between PIPT and VIPT is in the timing of when the TLB lookup completes); and, finally, the tags and corresponding circuitry are the same in PIPT and VIPT anyway.

Grading notes: Answering (A) earned 4 points. Answering (A,C) earned 3 points, because the complexity of the hardware is irrelevant. Answering (A,D) earned only 2 points, because it suggests a misunderstanding of how tags work.

Question 7 (4 points)
What is a (potential) benefit of performing certain operations lazily in a system?

(Circle all the correct options)

A. Faster allocation of the needed resources
B. Reduction in unnecessary computation
C. Fewer cache misses
D. Predictable execution timing
E. None of the above

Explanation: Performing operations lazily delays them until they are strictly needed; if the work is never required, then the system avoids doing it altogether. This results in fewer operations executed – option (B) – and, therefore, fewer associated memory accesses (and thus fewer opportunities to miss in the cache) – option (C).

Resource allocation speed is not directly affected by when operations are executed, so option (A) is wrong. Deferring some operations may improve short-term predictability of execution timing, but does not directly affect the overall predictability (if anything, laziness can make execution timing less predictable, as work may be deferred until a point in time that's harder to anticipate), which is why option (D) is wrong.

Grading notes: We awarded 4 points for (B,C). Answering only (B) earned 3 points, because it misses the fact that laziness can reduce cache pressure. Answering (B,D) earned only 2 points, because execution timing is not more predictable, but one could argue about whether the time horizon is short-term or long-term. Answering (A,B) earned 2 points because overall there is no faster allocation, but one could argue about whether the time horizon is short-term or long-term. Answering (A,B,C) earned 3 points. Answering only (C) earned 1 point, because it is correct but misses the main benefit of laziness.

Question 8 **(4 points)**
What is a (potential) benefit of performing certain operations speculatively in a system?

(Circle all the correct options)

A. They guarantee correctness even in the presence of failures
B. They reduce resource usage by avoiding unnecessary computation and memory accesses
C. They eliminate the need for rollback mechanisms
D. They ensure better predictability of execution timing
E. The system can use otherwise-idle time to execute instructions before their dependencies are resolved

Explanation: Speculative execution allows a system to employ unutilized resources to do work ahead of time, before it becomes certain that the work is needed (e.g., prefetch data) or before all inputs are available (e.g., start a calculation that depends on some inputs that are not available yet). Hence option (E).

Far from eliminating the need for rollback, it's common that the effects of speculation need to be undone, in order to preserve correctness. It also does little toward improving the ability to guarantee correctness, it even makes things more complicated, and thus more error prone. To a first degree of approximation, speculation does the opposite of reducing resource usage, because it utilizes them for work that might be unnecessary. Just like laziness, speculation does not directly impact the predictability of execution timing — one might improve the predictability of one operation by speculatively executing another (e.g., prefetching data in order to avoid a future cache miss), but overall the predictability does not improve.

Grading notes: Answering (E) earned 4 points. Answering (D,E) earned only 2 points, because execution timing is not more predictable, but one could argue about whether the time horizon is short-term or long-term. Answering (A,E) earned only 1 point, because correctness is a red herring; if anything, guaranteeing correctness is harder when using speculation.

Question 9 **(4 points)**
How does knowing the "working set" of a process help?

(Circle all the correct options)

A. It ~~identifies~~ is the total memory used by the process over its lifetime and enables pre-allocation
B. It determines the most frequently used data, ~~optimizing~~ enabling cache allocation optimization
C. It improves the process's CPU usage patterns
D. It provides a good estimate of the process's network bandwidth requirements
E. None of the above

Explanation: The working set of a process *P* is the set of pages that *P* has accessed in the most recent time interval Δ. Under the assumption of locality of reference, knowing this working set enables an OS to predict which pages are likely to be accessed in the immediately next interval Δ, and thus to predict which of the cached data is likely to be accessed again soon. This enables the OS to keep "hot" data in the cache. Hence answering (B) is correct.

The core idea of Denning's working set model is that the OS wants to keep only the currently needed pages in physical memory, not every page the process has ever touched or might touch again. In other words, the working set is fundamentally a "windowed" concept that refers to the pages accessed in the recent past. So answering (A) is incorrect. It is theoretically envisageable to calculate the union of all possible working sets

over all possible intervals of *P* 's lifetime, and thus getting the set of all pages that *P* will ever access, and then pre-allocating them, to avoid page faults. This effectively implies that the time interval referenced in the definition of the working set is the entire lifetime of the process, and this exercise is decidable only for a subset of all possible programs, so it is hard to imagine a system doing this. Finally, pre-allocation goes against the essence of the working set, which is a dynamic notion about current or recent usage.

The notion of "CPU usage pattern" is ill defined, and the notion of improving said pattern even more so. The working set does not relate in any direct way to network bandwidth and is thus irrelevant to estimating bandwidth requirements.

Grading notes: Answering (B) earns the full 4 points. Answering (A,B) together earns 2 points. Any other combination of answers denotes a sufficiently deep misunderstanding of the working set that it earns 0 points. In particular, any answer that selects either option (C) or (D) results in 0 points.

Question 10 (4 points)
Which RAID level provides the highest write throughput for a given number of disks?

(Circle all the correct options)

A. RAID 0 (i.e., block-level striping without parity or mirroring)
B. RAID 1
C. RAID 5
D. RAID 4
E. RAID 10

Explanation: RAID 0 achieves the highest write throughput: it uses block-level striping without any form of parity or mirroring, so it can distribute the write requests across all available disks (thus utilizing their full aggregate bandwidth) without incurring any delay for computing parity (thus reducing overhead to the minimum). Hence option (A).

In contrast, RAID 1 and RAID 10 involve mirroring, which doubles the amount of data written for each operation, therefore their throughput cannot match RAID 0 (for the same number of disks). RAID 4 and RAID 5 both involve parity calculations and updates, which introduce extra delays in the write path (i.e., increased overhead), which also causes them to not be able to match RAID 0.

Grading notes: Answer (A) earned 4 points. Answering (A,B) earned only 1 point, because it demonstrates a poor understanding of either RAID 1 or of what determines write throughput, or both. Option (C) strikes a good balance between performance and availability, but does not provide the highest throughput.

Question 11 (4 points)
What is true of the Global Name Service (GNS)?

(Circle all the correct options)

A. It eliminates the need for caching on the endpoint
B. It provides a centralized point of control for all resources
C. It allows resources to be accessed using location-independent names
D. It simplifies the network protocol stack
E. None of the above

Explanation: GNS names are independent of where the resource is stored. Directories are abstract collections of names and they may be physically stored in, and duplicated across, arbitrary sets of servers. To facilitate this, GNS maintains a mapping between directories and servers, using itself as a naming service to this end. For example, `/SRC/DEC/Lampson/Password` is a name that does not embed any information about the location where its value is stored – it could be stored in Switzerland or the US. Option (C) is correct.

Option A is wrong, because caching of lookup results on the endpoints is actually crucial to scalability in GNS. Since GNS is decentralized (each directory can be owned and managed by a different entity), option B is wrong too. Option D is incorrect, since GNS is implemented entirely at the application level and does not influence the network protocol stack.

Grading notes: Answering (C) earned 4 points. Answering (A,C) earned only 2 points, because it demonstrates a poor understanding of how client-side caching allows GNS to scale. Answering (E) earned 2 points, because it implies an understanding of some of the finer details of GNS but not its main goal.

Question 12 (4 points)
Which of the following is true of a domain-specific hardware accelerator?

(Circle all the correct options)

A. It can be faster than a CPU because it can ~~load and store more data per unit of time~~ load/store data from/to main memory faster than a CPU
B. It can never be faster than a CPU if the PCIe latency is higher than the CPU's memory access latency
C. If it is "slower" than the CPU (i.e., performs the same operations slower than the CPU could), then there is no reason to use it
D. If it is "slower" than the CPU, we ~~should~~ must improve its internal design to exploit more parallelism
E. None of the above

Explanation: Many accelerators today connect over PCIe to the system, and therefore access main memory over PCIe. So we accepted (B) as an "almost fully correct" choice.

However, there are also accelerators that are integrated on the SoC (e.g., Qualcomm's Snapdragon), and can thus have direct access to the main memory. Others connect to memory over CXL (like NVIDIA BlueField-3) – even though this goes over the PCIe physical layer, it uses its own specialized protocol that can achieve significantly lower latency than PCIe. Or, as in the NVIDIA Grace Hopper Superchip, both CPU and accelerator can use high-bandwidth memory (HBM) located with the accelerator chip, to which the accelerator has faster access than the CPU. Bypassing the CPU's memory hierarchy can often achieve higher raw data transfer rates than the CPU, e.g., when streaming large datasets. The specialization of an accelerator can enable its memory controller to do better than a more general-purpose MMU in CPUs (e.g., it could better exploit the certainty of batched accesses to contiguous memory). Finally, a loaded CPU that keeps stalling because of cache misses may well be slower in doing memory operations than the accelerator. Answer (A) is perfect.

Option (C) is incorrect because, even when an accelerator is "slower" than the CPU (e.g., a programmable NIC), offloading computation to it can free up CPU resources, enabling the CPU to do other things that might be more critical to system performance. Doing so will still accelerate the system, even if the accelerator itself is not faster than the CPU for the offloaded work. Option (D) is not correct either, because making a slower-than-CPU accelerator exploit parallelism typically entails adding more processing elements, larger memory structures and/or more sophisticated interconnects to handle concurrent operations – such additions

tend to increase chip area, complexity, and power consumption, leading to higher manufacturing cost and energy usage, while the increased performance might well be unnecessary. Therefore, an accelerator may well not exploit all available parallelism, yet still be a useful (and commercially successful) device.

Grading notes: Answering (A) or (A,B) earned 4 points. Answering (B) only earned 3 points. Selecting (C) in your answer resulted in a 2-point penalty, because it shows an incomplete understanding of how accelerators are used and what it takes to accelerate a system. Selecting (D) also incurred a 2-point penalty, because it shows a misunderstanding of the trade-offs in systems, where a slower hardware device may still be very useful (e.g., because of lower cost or energy consumption than a faster one). Making a device faster entails a cost, and whether incurring that cost is a good idea or not really depends on the system – it's a trade-off. We exceptionally accepted (E) as a correct answer, because the original formulation "load and store more data per unit of time than a CPU" could lead to an interpretation that would annul option (A).

Question 13 (4 points)
According to the end-to-end argument, which functionality must be implemented solely at the application layer, not within the network?

(Circle all the correct options)

A. Error detection and correction
B. Packet forwarding and routing
C. Congestion control
D. Address resolution
E. None of the above

Explanation: The end-to-end (e2e) argument concerns functionality that can be completely and correctly implemented only at the endpoints, but may additionally be implemented within the network to improve performance. The classic example of such functionality is detection and retransmission of corrupted data. This can only be done completely and correctly at the endpoints, because no network-layer mechanism can guarantee correct data delivery end-to-end. At the same time, a network may provide early detection and retransmission (e.g., around a wireless link) to reduce the amount of corrupted data that the application sees, hence improve application performance.

Answering anything other than (E) is wrong. Regarding option (D): Address resolution refers to translating a higher-layer address to a lower-layer address, and this must obviously happen above the lower layer. The reason for this, however, is not related to the e2e argument; rather, it is that the lower layer does not understand higher-layer addresses, hence could not plausibly perform the translation. For example, ARP translates IP addresses to MAC addresses; performing this translation within the MAC layer is not an option, because the MAC layer does not understand IP addresses. So, IP-address resolution must happen above the MAC layer. But this has nothing to do with the e2e argument.

Question 14 (15 points)

You're building an operating system for devices with limited memory. The processor can provide paged virtual memory (VM) with a two-level hierarchical ~~page table~~ paging scheme, or segments, or direct physical memory access. In order to decide whether you can afford the convenience of VM, compute the memory overhead of the VM option when fully mapping a 1-GiB address space. Assume that virtual address (VA) size is 64 bits, physical address (PA) size is 48 bits, page size (PS) is 4 KiB, each page table entry (PTE) is 8 bytes wide, and a page table (PT) always fits in one memory page.

(*Write the answer in the small box below, and optionally provide your calculation to justify it.*)

Answer: $2^{21} + 2^{12}$ bytes

Calculation (optional):

1 GiB = 2^{30} bytes

4 KiB = 2^{12} bytes

1 GiB consists therefore of $2^{30} / 2^{12} = 2^{18}$ pages

A PT fits in 1 page => $2^{12} / 2^{3} = 2^{9}$ PTEs in a PT

We need $2^{18} / 2^{9} = 2^{9}$ second-level PTs

Each 2nd-level PT takes 2^{12} bytes => need 2^{9} x $2^{12} = 2^{21}$ bytes to store the 2nd-level PTs

Overhead = 2nd-level PTs + top-level PT = $2^{21} + 2^{12}$ bytes

Question 15 (15 points)

Consider N applications running on top of the Aegis exokernel, with each application having a working set of 32 distinct pages. The software-managed TLB can store 2048 entries. You are considering modifying Aegis to use an SSD or an NVRAM to cache TLB entries (instead of using DRAM, as Aegis did). You want the mean memory access latency to be ≤100 ns including access control.

Determine the value range of N for which using an SSD makes sense and the value range of N for which using an NVRAM makes.

Assume that memory accesses are all served by DRAM (i.e., no caches), TLB lookups are instantaneous, and DRAM / NVRAM / SSD access latencies are 50 ns / 100 ns / 100 microsec respectively.

(*Write the answer in the small box below, and optionally provide your calculation to justify it.*)

Answer: $\nexists N$ for which SSD makes sense NVRAM makes sense if $N \leq 2^7$

Calculation (optional):

The TLB occupancy is $2^5 N / 2^{11} = N / 2^6$ ⇒ probability of a TLB miss is $1 - 2^6/N$

The average memory access latency is 50 ns + $(1 - 2^6/N)$ x cache_lookup_time_in_ns

This needs to be ≤100 ns ⇒ $(1 - 2^6/N)$ x cache_lookup_time_in_ns ≤ 50 ns

⇒ $N \leq 2^6 / (1 - 50/\text{cache\_lookup\_time\_in\_ns})$

For an SSD, this implies $N \leq 2^6$ + some ε amount, so effectively $N \leq 2^6$

The working set for 2^6 applications is at most 2^6 x $2^5 = 2^{11}$ pages, which already fits in the TLB, so there is no N for which an SSD makes sense.

For an NVRAM, the inequality is $N \leq 2^6 / (1 - 0.5)$ ⇒ $N \leq 2^7$

Question 16 **(15 points)**

Suppose you want to allow any Internet end-system to specify a set of Autonomous Systems (ASes) that its outgoing traffic should ~~not cross~~ avoid, if possible. At which layer(s) of the Internet stack would you implement the necessary functionality? Explain the trade-off(s).

(Answer in maximum 10 sentences)

One option would be at the network layer: extend BGP to discover multiple routes per destination prefix, then extend IP to enable senders to specify undesired ASes and routers to pick a route that does not include the latter (if BGP has discovered such a route). Another option would be at the application layer, using a RON-style overlay: if the path from an entry node to an exit node contains an undesired AS, the entry node looks for an indirect path that does not. The former option could, in principle, discover all the AS paths from a source to a destination AS, hence satisfy more end-system requests. However, it would worsen the scalability of the network layer, because it would require routers to keep more forwarding state and perform more expensive IP lookups. Also, it would enable malicious end-systems to affect router performance by making complicated requests. I would choose the latter option, which would satisfy fewer requests but would be significantly easier to deploy and would not affect the scalability of the network layer. One could also make an end-to-end argument in favor of option 2: implementing the functionality in question at the network layer would negatively impact the applications that do not need it (by making the network slower and more expensive).

– – – – – END OF THE EXAM – – – – –

(You can use the following pages for scratch notes. They will be discarded prior to grading.)