Question 1: Virtual Machines [23]

a) [4] Explain the flow of control through the processor when an application, running on an unmodified OS, running on top of a virtual machine, makes a system call.

Average Score: 20.4 +/- 2.4
Full credit requires mentioned all three levels and also noting that the hypervisor actually invokes the guest OS trap handler.

[Yahel Ben-David:] I'll presume the hardware used is the easier case of MIPS. The x86 architecture complicates things as some instructions may silently fail if not run under a real "ring-0" privileges. Before delving into the flow of control for handling a system-call (syscall) on a virtualized system, let's briefly discuss what happens in a traditional setting (without virtualization):

When the running application executes an instruction requiring access to restricted resources controlled by the OS it would make a syscall. In a syscall first the arguments will be pushed onto the stack and then a special trap/interrupt will be issued to transfer control to the kernel. The interrupt handler will transfer control to a well-defined location for the system call code (established during boot-time and thereafter is well known by the hardware as the target jump location for the interrupt). At the time of the interrupt the hardware also changes the processor mode to a protected kernel mode. Let's list a simplified form of these steps and those that follow:

1. User-space process - Executes instruction requiring privileged access.
2. User-space process – makes a system call which interrupts the OS.
3. Hardware/CPU – Switch to protected kernel mode and jump to syscall code.
4. OS running in protected kernel mode – execute the syscall code and return.
5. Hardware/CPU - Switch to unprotected user mode and return to execute user code.

On a virtualized platform the application does exactly the same thing as in steps 1 and 2, however since it's the VMM that controls the machine, the interrupt handler implemented in the VMM will get executed. The VMM does not know how to handle the syscall (it can't know that for every OS) but it does know the location for the OS's interrupt handler code. It knows that because on boot-time, the OS tried to install its own interrupt handlers through use of privileged mode and hence got trapped by the VMM which then recorded all memory locations for interrupt handlers. Since the VMM is running in protected kernel mode, it now jumps to the OS's interrupt handler and successfully executes that code as should have been done without virtualization.
When the guest OS finishes, it needs to execute privileged instructions to return from the interrupt handler, which again are being bounced into the VMM, which then realizes that the OS is trying to return from the syscall and thus returns control to the user process and puts the machine back into non-privileged user mode.

b) [4] Explain what happens when the same application has a page fault (i.e. explain the flow of control).

[Grey Ballard:] The guest OS maintains page tables to access pages in its virtualized address space, and the VMM maintains shadow tables for the guest OS which translates virtualized page locations into physical locations (in memory or on disk). In the case of a page fault, the VMM must handle it (load page into physical memory if necessary) and update its shadow tables as well as the guest OS’s page tables. The authors of the Xen paper note that page fault handling can't simply be reflected to the guest OS because access to the faulting address register is restricted to ring 0. VMWare also makes guest OS page tables read-only by trapping attempts by the guest OS to update them, thereby putting all control of virtual memory in the VMM layer. Thus the flow of control in the case of a page fault passes from application to the VMM where all the memory management is handled, and then control is passed back up to the application. need to cover two-level of page tables, translation on page faults from real fault to guest os fault.

[alternative from Chia Yuan Cho:] The virtual machine monitor also traps page faults, in a similar way to system calls. To virtualize physical memory, the virtual machine monitor maintains its shadow page table and mappings between physical to machine addresses. Consistency is maintained by trapping all guest OS page table updates. When a page fault occurs, the processor traps to the monitor. The monitor loads a real physical page into memory, updates the page table entry in the virtual machine using the machine address of that page, and updates its own shadow page table. Just like system calls, it may need to update the privileged registers of the virtual processor and put it into supervisor mode. It then jumps to the virtual machine's page fault handler to execute it on real CPU.

c) Assume that one physical machine is running 10 virtual machines, all running mostly the same OS and applications (e.g. an Apache web server on top of Linux). Normally, each VM has independent copies of all of the physical pages needed for its guest OS and app.

i) [2] Explain why this is.

[Michael Greenbaum:] The VMs under a VMM are executed entirely independently from one another, and allocated their own sets of resources. Without special intervention, no sharing between them is possible.

ii) [3] What can you do to eliminate this duplication?

[Shaon Barman:] This duplication can be eliminated by first identifying identical pages between different virtual machines. Only one copy of each page needs to be kept, and this copy can be shared amongst all of the VMs. Since it is shared amongst all VMs, this page should be marked copy on write or read-only by the virtual machine monitor, so that the VMM can intervene when a VM tries to write to the page.
iii) [3] What should happen if a guest OS modifies a shared physical page? (that used to be a duplicate)

[Erin Carson:] A copy-on-write scheme should be used when a guest OS modifies a shared physical page. This means that when the guest OS tries to write to the page, a duplicate is made. This way, pages can still be shared and duplicates are only created when necessary.

d) VM security

i) [2] Explain why a VM can do secure logging but an OS can’t?

[David Burkett:] An OS can’t do secure logging because if the OS kernel is compromised, then the logs are no longer trustworthy. In particular, the attacker will have access to all logs stored on the OS’s filesystem, and will be able to modify or delete logs in order to conceal the intrusion. A process running directly on a VMM can write secure logs, even if a guest OS hosted by that same VM is compromised.

ii) [2] Explain why a hypervisor is generally more trustworthy than an OS.

[Bill Marczak:] The VM represents a much smaller trusted computing base than an OS. For example, the implementation is several [orders of magnitude] smaller in lines of code, and the interface is far narrower (and thus restricts attackers better) than the richer interface provided by an OS (processes, virtual memory, etc.).

[Javier Rosa:] Explain why a hypervisor is generally more trustworthy than an OS. Modern hypervisors are written on top of well specified hardware and (for now) are much simpler than the guest OSs they are hosting. This allows for the possibility of verifying the security of the hypervisor, perhaps the hypervisor could be formally proven correct. A hypervisor can be written with the specific intent of providing security above all else whereas most OSs are geared towards general purpose operation. Additionally, the guest OS is running in user mode all of the time. If the intruder gains control of the guest OS all they can do it muck around with the guest OS or compromise other guest OSs on the network that are in user-mode.

iii) [3] Explain in general how a VM manager can limit what an OS can do, and compare these limits to those of the underlying physical machine.

[Jonathan Barron:] At the very high level, the VMM sits between the OS and the hardware, and can therefore control any interaction between the OS and the hardware. In a situation without a VMM, the OS can interact with the hardware whenever it decides to (it is unlimited), but with a VMM we can enforce arbitrary constraints. Also, because the VMM sits between the OS and any network connection (in general), the VMM controls any communication into and out of the OS (which also helps us during secure logging).
a) [5] Assume that we want to implement a (more) secure web browser using K42. Explain how you might limit the impact of poorly written or malicious plug-ins.

[Chris Wilkens:] A “safe” browser implementation would leverage K42’s protected procedure calls between clients and servers. The foundation of the browser would be a service running in a protected address space that handled all interaction with the local machine. Plug-ins would be wrapped and run as clients in this model. The wrapper would provide the necessary API to the plug-in and translate that API to PPCs to the browser. In this implementation, PPC limits a plug-in's sphere of influence to its address space and the functions available to it from the browser service.

b) You are building a locking library for a many core programming framework. In addition to the basic acquire/release calls, you must deal with the complication that threads may be killed while the application is expected to continue operating.

i) [2] Left unhandled, what bad things happens relative to locks when a thread is killed?

Locks are not released, typically leading to deadlock.

ii) [3] Explain how you might track which locks are held by a thread at the time of its death.

[Priyanka Reddy:] When a thread obtains a lock, you can store this information within each thread or in a process-level hashtable that maps a thread to a list of its locks. Storing this data in the thread so it's as close to the source as possible is a good choice and prevents us from keeping a centralized table updated. However, if the lock data needs to be used for any other purpose, such as for statistics, then a centralized hashtable would be better.

iii) [2] Explain why it is not sufficient to just release those locks.

[Anna Rafferty:] Just releasing those locks is problematic for the reason stated in (i): a thread may be partway through what should be an atomic operation and may have left the system in an inconsistent state or not have completed all necessary book-keeping tasks. This means that simply releasing the locks may not preserve consistency. When using locking, we're often preserving some invariant that we guarantee holds when the lock is entered and that the thread will restore when the lock is exited. If the lock is not exited properly, the invariant may no longer hold.

iv) [4] Explain how a language like Java with exception handling and try/finally clauses can be used to clean up locks well.
I took off one point if you had the cleaning up of the invariant in the finally clause in stead of the catch clause. The latter is preferred as it only runs in the exceptional case, not all of the time.

[Samuel Zats:] Java provides the try-catch-finally clauses that allow the code to attempt to execute code in the try block. In the event an exception is raised; the invariant can be fixed through the catch block with an exception handler. The finally block is always executed, regardless of whether an exception was raised. The finally block includes clean up and lock release. If an exception or kill is thrown, the catch will unwind before resuming to the finally clause. If an exception does not occur, the sequence is simply try and finally.

[Bill Marczak:] You can put code that acquires a lock in the “try” portion. If an exception is caught, you can put some custom code in the catch block that restores the invariant. Then, the finally block may release the lock. This works because Java is always expected to fail by throwing an exception, rather than crashing the JVM, for example.

v) [5] Propose some mechanism to cleanly handle the locks that are held by a (just) killed thread for a language that does not have exception handling, try/finally, or some similar unwinding mechanism.

[Meromit Schuster:] Given no help from the thread (as no unwind mechanism) I think the best we could do is make sure that for all locks we have to release are released in a state that holds the invariant. For this, we could have a function at the lock level, “restore_invariant()” which is used in case of a thread death and sets the element that was locked to some state (either previously fixed or based on the state the element is in) which fulfills the invariant.

c) [3] Explain why locks tend to be better for many core applications, while distributed applications tend to prefer expiration or leases.

[Jie Tang:] Locks tend to be better for many core because they are designed for high frequency, low cost, reliable computations. Processes and threads are likely to acquire locks, update shared data structures, and release them quickly and repeatedly. This can happen on the order of milliseconds. Flexibility in lock management is crucial for high performance in this setting, something that cannot be done with prespecified timeouts. More importantly, links between processors in a many core architecture are extremely reliable, making it unlikely that a lock will become unavailable.

In a distributed application, on the other hand, leases and expirations for resources are nice because they can operate in the absence of future communication. This is important because network links are much less reliable than an onboard bus. And applications tend to hold resources for much longer: for example, dynamically assigned IP addresses may be held for several hours or days. This makes the overhead of timeouts and slow lease acquisition and handling less pronounced.

[Ali Koksal:] Remembering the CAP theorem, many core applications can opt for consistency and availability as they are not concerned about partition tolerance. In order to achieve strong consistency, locking is the better approach. Meanwhile, distributed applications focus on partition tolerance and availability, and provide “best-effort” consistency. In presence of network failures,
the systems should be partition tolerant, and it is impossible to achieve consistency using locks in such a setting if the service needs to be highly available too. These application thus have weaker consistency semantics achieved using leases and expirations, aiming to be “eventually consistent”.

**Question 3: DB vs. OS thinking [6]**

*Average Score: 4.17 +/- 1.43*

a) [2] Explain why “systems” folks use lock ordering, while “database” folks allow any order of lock acquisition.

[Luke Hodgkinson:] “Systems” folks use lock ordering to prevent cycles and deadlock because they have no clear way of aborting transactions. They write the code that acquires locks themselves so they can control the ordering in which the locks are acquired (Notes, #9, p8). “Database” folks allow any order of lock acquisition in order not to limit queries written by users. They use deadlock detection and transaction abort to resolve cycles in the locks that are held.

b) [4] Give two answers as to what can a “no steal” system do when it has run out of memory?

[Regis Blanc:] Let’s recall that a no-steal system will never write to non-volatile storage before the transaction commits. So if it runs out of memory, it means it has too many pages to write to disk, but cannot write them except if the transactions has committed. Well first it should write every page from transactions that has committed. Assuming this is done and it is still out of memory there are several options. Well first it could stop [crash] -- maybe this is not the best decision... A more serious possibility would be to abort a transaction and to empty its memory for it. And then to restart the transaction (the system could even wait for another transaction to commit before restarting the aborted transaction).

Another solution could be to use swapping in the same way an operating system does. It can store some page on the disc, and move it back when needed. Note that it still follows the no-steal policy because this swap space is not real durable storage, if a crash occurs we would not recover any data from this swap space, we will consider them as lost.
**Question 4: Graphs [10]**

Average Score: 6.20 +/- 2.33

Match up the numbered graphs with the lettered axes below. The match is one-to-one.

The overall point of this question is to push the skill of critical thinking about graphs. When writing papers, you should start with the graphs and what you expect them to look like, and what you expect them to reveal or prove about your work. You can work out the desired graphs before you run any experiments and in fact the desired graphs should determine your experiments. Conversely, as a reviewer I always look at the graphs early to see if they make sense and also to know the range of evidence provided in favor of the research. Specific notes:

a) 8 -- has to start at zero and increase exponentially; then periodically drops in half when it detects congestion
b) 10 -- This is straight from the U-Net paper.
c) 2 -- Performances falls off a cliff once the working set no longer fits in memory.
d) 5 -- This is similar to plots from the Savage TCP paper
e) 7 -- Starts high due to the seek time, then bumps up every time a new track is needed.
f) 6 -- Must start from zero; asymptotically smooth, but bursty due to high variance
g) 4 -- Also starts from zero and should have the same overall slope (since it is the same process), but now with low variance. The horizontal sections occur because time passing while the process is not running (something else is scheduled).

h) 1 -- as the stride gets longer it gets less value from each level of the memory hierarchy. The first jump is when it no longer hits in the level 1 cache, while the second jump occurs later because the 2nd-level cache is much bigger (4x bigger in this picture).

i) 3 -- This is the famous “bathtub curve” -- problems at the beginning when it is new (and the bad ones are being returned) and problems at the end when it is old. This was not really in the lecture notes (I think), but it is readily on line.

j) 9 -- Once the working set no longer fits in the cache, the hit rate drops quickly...

Question 5: File systems [12]

Average Score: 10.78 +/- 1.22

a) [2] Why are seeks the most important aspect of file-system design?

[Seth Fowler:] Because, at least with mechanical storage, seeks are expensive; a mechanical drive is able to deliver dramatically higher performance on sequence reads than it can on random reads, in which it has to seek before accessing each block. To illustrate the point, one Western Digital Velociraptor drive had measured sequential read performance of 255.1 MB/s and a measured random access seek time of 7.1 ms in a review. If a block is 4KB, then the drive could read about 65,000 blocks in a second if they were placed sequentially. However, if the drive had to seek for each block, then the drive could only read about 140 blocks in a second even if the time required to actually read the block was zero! That's two orders of magnitude difference in performance.

b) [4] How could you trade (growing) excess disk capacity to improve performance?

[Michael Zhang:] The excess disk capacity can be used to store duplicate copies of data in various locations on the disk. Whenever a block is read, the disk will access the closest copy of the data relative to the disk head, hopefully reducing the average distance the disk head has to seek to access data. The goal is to minimize seek time using the excess disk capacity.

Another possibility is to take more advantage of sequential reads and writes using the excess disk capacity, further reducing seek time. For example, lets say three pieces of data (A, B, C) are accessed very frequently by the file system, but in different orders (A, B, C), (A, C, B), (B, C, A), etc... One way to minimize access time is to store data sequentially on disk so that we can maximize the data read per seek and rotation. In this case, we could save access time by reading every sequence of these three pieces of data sequentially, so we could store each sequence separately using the excess disk capacity, allowing us to sequentially read these three pieces of data regardless of which sequence we need to read them in. This is a second way we could trade excess disk capacity to improve performance.

Obviously, certain problems arise in both these methods (i.e. the need to maintain the duplicate copies). However, I'm just assuming these blocks are mostly read rather than written, and that solving these problems doesn't fall in the scope of this question.
If one were to sacrifice disk capacity by only writing to the outermost portions of disk, thus allowing for all data to be stored at a greater rate of throughput (constant angular velocity times greater radius = greater rotational velocity = faster data throughput), one could have greater disk bandwidth by having less capacity. Furthermore, this would improve performance because a smaller seek window (lower radial band of disk information) would necessarily lead to lower seek times as less distance would need to be traversed.

c) [6] How might you redesign AutoRAID given the presence of 1TB disks (10 cents/GB) and 64GB flash cards ($2/GB)?

[Greg Ballard:] AutoRAID sets up an automated hierarchy between a small amount of fast mirrored storage and a large amount of slower (but more storage efficient) RAID 5 storage. Under the assumptions that only part of the data is in active use at any time, and that this working set changes slowly over time, AutoRAID strikes a balance between performance and storage efficiency. In general, about 10% of all disks are devoted to mirrored storage. If we consider the hierarchy to be static (unlike original AutoRAID), we can make one set of design decisions for the smaller faster level and a different set of decisions for the larger slower level. For instance, performance is more important at the smaller level—in the case of AutoRAID we’re willing to pay for twice as many disks as RAID5 would require in order to obtain say 3x performance improvement (optimistic estimate, based on RAID’s small-write problem, although mirrored storage still has to do two parallel writes). Given this tradeoff, it seems justified to pay 20 times the cost for flash cards instead of disks and obtain 100x performance improvement (another estimate, based on what I saw on the internet). At the larger level, we care less about performance—AutoRAID chooses to sacrifice performance for cost efficiency. Thus, at this level, it doesn't seem justified to buy the flash cards. So my redesign of AutoRAID would be to use flash cards for the mirrored storage of the working set and use disks in RAID 5 configuration for the inactive data. However, making these design decisions and choosing the tradeoffs between performance and storage at each level is contrary to the philosophy of AutoRAID, in which you use all the disks you can afford, and the system strikes all the balances automatically (not to mention the loss of flexibility to change system parameters—like the mirrored storage watermark—in software).

[Amita Muralidharan:] There are at least 3 1TB disks, whose purpose is to store data in RAID-5 format, and three 64GB flash cards whose purpose is to store hot data in mirrored format. Every time a disk is added, one flash card (or more) could be added to match or increase hot capacity, depending on how important it was to the customer to have speed, and how much he was willing to pay for it.

Other than the different location of hot data, and the fixed capacity assigned to hot data (v.s 10% of total capacity), the AutoRAID logic remains the same. Data is still kept track of by the controller as hot or cold, and demoted and promoted by the same rules.

If the flash cards were more expensive, or turn out to overflow too often (i.e. there is not enough space for hot data) an intermediate class of data (“warm”) can be introduced. Warm data would be stored in mirrored format on the terabyte disks, in 10% of their capacity, just like hot data was in the original AutoRAID. When data is not frequently-used enough to remain in hot data, which could happen when the flash cards become full, warm data could be written to the mirrored storage on disk, in the same way as in the original AutoRAID. It would be from this “warm” storage that data will be demoted to RAID-5 storage.
Question 6: Dynamo [25]

Average Score: 20.44 +/- 3.02

a) [2] Explain the value of “symmetry”

[Anna Rafferty:] Symmetry is the notion that all nodes should have the same responsibilities. No node should have specialized capabilities or extra organizing responsibilities. In this way, each node is equivalent to any other node, meaning there are not “special case” nodes to handle and the same routines should be usable by and for all nodes. This makes maintenance of the system easier and simplifies the design.

b) [2] Give an example in which Dynamo breaks this principle

[Priyanka Reddy:] Dynamo breaks this principle in its external discovery process. In order to avoid logical partitioning, Dynamo designates some nodes as seeds. Seed nodes are discovered through an external mechanism and known to all the nodes. These seed nodes effectively prevent logical partitioning from taking place. Seed nodes are also members in the Dynamo ring but have extra functionality.

c) [3] List two situations in which Dynamo does not use the first N successor nodes to store N replicas.

[Anupam Prakash:] Dynamo uses the first N distinct available successor nodes to store replicas a) If a successor node is unavailable the replica will be stored on the next available node. b) If some of the N successors correspond to the same physical node, the next physically distinct node will be used.

d) [3] Assuming there are partitions possible, explain how to set N, R and W to achieve consistency (but not availability) and vice versa.

-1 if it wasn’t clear that you understood \( R+W>N \) implies consistency

[Chris Wilkens:] To achieve consistency, we need \( R + W > N \) so that any read/write combination is guaranteed to overlap on some node. A choice of (3; 2; 2) seems to be common for this purpose. On the other hand, to achieve availability, we want to minimize R and W, thereby reducing the number of nodes that must be up. A choice of (N; 1; 1) would achieve this while destroying any guarantees of consistency.

e) [2] True/false: higher load also leads to higher load imbalance

[Bill Marczak:] False -- They say in the paper that in practice, lower load leads to a higher load imbalance.

f) [4] Explain why Dynamo is a better fit to implement LRVM than a file system.

1) Don’t need a log -- dynamo is already providing replication for fault tolerance.
2) Both LRVM and Dynamo target small contiguous data, unlike file systems.
3) LRVM needs fast writes but can tolerate slow reads (used for recovery).
4) **Dynamo supports concurrent writes and provides data replication.**

[Parth Sagdeo:] It makes more sense to implement LRVM on Dynamo for several reasons. First, the problems of resiliency and scalability are already handled by Dynamo – it abstracts away all details except key/value pairings, and with a much larger and more scalable storage medium. Second, LRVM’s use case involves small segments, which correspond well to key/value pairs in Dynamo.

[Nick Knight:] The Dynamo paper stresses that many services which use the RDBMS model actually do nothing more than primary-key lookups, and do not need to take advantage of the complicated query processing that a RDBMS offers. LRVM is perfect example of this. The LRVM paper presents a typical transactional implementation, using no-undo/redo logging, reminiscent of a RDBMS. But really, all LRVM does is key-value lookups and writes, the key is the region id (memory address) and the value is the binary data to be written/read. Using Dynamo we can implement LRVM (see next part), doing away with the overhead of a log manager, and in general use resources more efficiently. There are other advantages to Dynamo, namely, availability (you want to minimize latency, certainly) and scalability.

g) [5] **Explain how to implement LRVM using Dynamo.**

[David Burkett:] If you implemented LRVM with Dynamo, you wouldn't need to keep a separate write-ahead log since writes can go directly to permanent storage without too much latency (so long as your settings are designed for high write availability). Instead, you'd have a key associated with each segment, which is stored locally whenever a segment is paged into memory (if the segment is pre-existing, you get the data with a simple read). When you commit a transaction (or page that segment out of memory), you issue a write to update that segment directly. If you need to restore a segment after a failed transaction or as part of a disaster recovery, you can just issue a read to get the required data. Dynamo's consistency semantics will ensure that the read gets the correct data so long as you set \( R + W > N \). The other operations, related to log management, are no longer necessary.

h) [4] **Explain why churn is less of an issue for Dynamo than for other DHT systems (such as Chord).**

[Javier Rosa:] Chord is intended for untrustworthy apathetic nodes. This would be appropriate for the DHTs backing torrents for example, but Amazon can leverage the quality of their data centers. That being the case, churn only occurs when nodes fail or communication links are broken. The failure rate of a particular node is reasonably low and communication links are usually reestablished quickly.