3SH3.Questions.txt

Question: What is a microkernel?
Answer: A microkernel aims to move as much code from kernel space into user space. The goal is to have the minimum amount of code running in kernel space. An advantage of this is increased security, because less code is running in kernel space. A disadvantage is the performance overhead of communication between user and kernel space.

Question: What is the difference between Linux and Unix?
Answer: Both are a family of operating systems. Unix was developed at Bell Labs by Ken Thompson, Dennis Ritchie, and others. Linux was was developed later by Linus Torvalds. Unix is (mostly) closed source. On the other hand, Linux is open source. Linux is based off of Unix; it is a Unix-like operating system - almost a clone.

Question: Why does Windows and macOS report different storage capacity for storage devices (i.e. Hard Drives)?
Answer: The size is the same, but the units are different. Windows reports the size in GiB and macOS uses GB. If you do the math and convert from one unit to the other, the size is the same. For example: 1,000,000,000 Bytes is 1GB. But when converted to GiB it is 0.931322.

Question: Is it possible to have so many interrupts that it causes a stack overflow? And if this happens how does the OS deal with it?
Answer: This is possible, but very rare and almost never happens. The OS does a very good job at making sure something like this never happens.

Question: Conventionally, cache is faster than main memory (RAM). However, doesn’t this statement only apply to reads, because if you are constantly changing the values in cache, don't they need to be updated in main memory (RAM).
Answer: Yes, this statement is partially true. If a value is changed in cache, then the same value needs to be updated in RAM. When cache contains an updated value, it is referred to as a dirty block. Although, in most cases it does not affect the performance of the application because cache is still the main source of data/information - so there is no need to access RAM.

Question: Out of the two architectures mentioned in class, Von-Neumann and Harvard, is there an advantage to using one over the other?
Answer: In Von-Neumann architecture, program data and instruction data are stored in the same memory, and travel along the same bus. Since you cannot access program data and instruction data simultaneously, the Von-Neumann architecture is susceptible to bottlenecks. Harvard architecture separates program data and instruction data, and they have their own bus. This overcomes the bottleneck in Von-Neumann architecture. However, Harvard architecture is more complicated.

Question: DMA transfers memory from an I/O device to main memory, directly. If this memory isn't checked properly, in theory, can DMA be used to compromise a system? (i.e. Hack it).
Answer: In theory it can be used to obtain arbitrary RWX privileges, but newer operating systems disable DMA by default, so this is no longer possible.

Question: Since System Calls are provided to programs via an API and not direct access, is there a negative performance cost to this?
Answer: There is a performance tradeoff in creating an intermediate layer between programs and the system, but it is worth it because the benefits vastly outweigh the disadvantages. Plus, the performance degradation is very minute.

Question: If there is a System Call inside a System Call, does it save the state of the first System Call in the stack? If so, can you overflow the stack with lots of System Calls?
Answer: Yes, the state of the first system call (i.e. It's data) is saved in memory (i.e. Cache). Overflowing the stack with repeated system calls is extremely unlikely to happen. The OS does a good job at making sure this does not happen.

Question: Why are microkernels more secure? Is it because less code is running with high-access; thus, the attack surface is less.
Answer: Microkernels move code from the kernel to the user space. So, if there is some vulnerability in the code, and an attacker manages to exploit it, then the attacker does not have full control of the kernel, but only user space. Compromising the kernel gives attackers full control to do whatever they want. We want the kernel to be as secure as possible, and minimizing the attack vector is a valid strategy.

Question: Is the third column of 'lsmod' the 'pid'?
Answer: No, the third column in 'lsmod' is not the PID. The third column indicates how many instances of the module is being used. A value of 0 means that the module is not being used.

Question: In `simple.ko` does the `ko` stand for Kernel Object? Similar to how `o` stands for object in the `simple.o` file?
Answer: Yes. [(*.ko)] stands for kernel object, and [(*.o)] stands for object.

Question: Where do drivers sit in the hierarchy of software, OS and kernel?
Answer: Drivers can be in the user-space or the kernel-space, depending on the driver. Typically, drivers are kernel space. However, in micro kernels, drivers are sent to user-space, and only the essentials are in kernel-space.

Question: If I remove the battery from a phone, does it lose the

ability to keep track of real time?
Answer: This entirely depends on the source of power. If the internal real time clock is powered solely by a single battery, then removing it will cause the phone to lose its ability to keep track of real time. However, if the internal clock is powered by a separate battery and micro-controller, then it will be able to keep track of time without the main battery.

Question: Is the kernel part of the operating system or is the OS part of the kernel?
Answer: The kernel is part of the operating system. It is one of the first processes that runs when the operating system starts up. The relationship between the OS and the kernel is like a car and its engine.

Question: What does the OS do that the kernel doesn’t do?
Answer: The kernel runs all the time, while the operating system runs only when it is told to do something. The kernel is the middle man between hardware and software. If any program/process/application wants CPU time, memory, etc., then it must ask the kernel.

Question: Does the operating system get to directly interface with the hardware? Or does it have to go through the kernel?
Answer: No, the operating system does not directly interface with the hardware. The OS must go through the kernel. Every process/program/application goes through the kernel for hardware resources.

Question: The software or algorithm that is used for scheduling, where is it stored when it’s in use?
Answer: The scheduling software is part of the operating system and is stored in secondary memory. When the operating system is booted, essential system processes like the scheduling algorithm are written to main memory, and this region is protected.

Question: What stops processes from writing or reading to other processes memory?
Answer: This is the job of the programmer, and is accomplished using mutex and semaphores. These help in preventing processes from writing to the same block of memory. On the contrary, multiple processes reading the same block of memory is not an issue.

Question: Does the kernel have a PID? If so, what is it?
Answer: The PID of the kernel process is 0.

Question: What is a page fault? Yield?
Answer: Page fault occurs when you try to access a page that does not exist. Yield is when a process gives control to another process.

Question: Is it possible to load data into registers directly from a HDD, or secondary storage, and skip the RAM/Cache altogether? And vice

versa?
Answer: This is not recommended or allowed, nor is it normal practice. In fact, this is bad practice. However, there may be a workaround to accomplish this. Nonetheless, swap memory is a better solution to insufficient main memory.

Question: Since the dispatcher and scheduling algorithm are system processes, do they have their own state or attributes that need to be saved?
Answer: Yes; and this is supported by the operating system. Both are system processes that are part of the OS.

Question: Why is the PID of the child process 0? Isn't a PID of 0 reserved for the Kernel?
Answer: The PID, relative to the child process, is always 0. But getting the PID of the child process from the parent is greater than zero.

Question: Would you say that it is good practice to always perform NULL checks and check the exit status of functions?
Answer: Yes; this is good practice and it helps prevent bugs/errors.

Question: Is it good practice to immediately close a pipe or remove shared memory the second you are done using it, or can you wait a little while?
Answer: It is better to wait before removing it.

Question: What is the difference between a thread and a process?
Answer: Both are independent sequences of execution. A process has its own memory, data, address space, etc. A thread shares this data with other threads, except for the stack. Processes are heavy on the system, and threads are light. Every process is a thread, but not every thread is a process.

Question: Let's say you have a process, called 'P'. 'P' calls fork and creates a child process called 'C'. Both 'P' and 'C' create two threads. How many threads are on each process? And where does the primary thread fit in all of this?
Answer: Depends on syntax. You can duplicate all threads when creating a new process OR only create new process and no threads.

Question: If I started a thread in a child process, does the thread go away when the child is terminated? What if the child is not terminated and is a zombie process?
Answer: They will also be zombies; zombie threads. However, modern operating systems are smart enough to prevent this.

Question: What is an example of something that cannot be parallelized?
Answer: Anything that has a huge data dependency. Here are two

examples:

1. Assume you have two numbers, A and B. You add them together to get C. Then you add A and C to get D. Then you add A and D to get E. Then you add A and E to get F. You repeat this for many iterations. This cannot be parallelized.
2. Assume you have a list of numbers called X. You compute some statistic on X and get Y. Then you use X and Y to compute another statistic and get Z. Then you use X and Z to get another statistic and get Q. This cannot be parallelized because the next calculation depends on the previous ones.

Question: What is a good example of something that has huge benefits if parallelized?
Answer: Image rendering is very good for processing, large matrix multiplication, etc.

Question: In what scenarios would you use a thread over a process? And vice versa?
Answer: Assume you have a word processor application running with one window. The current window is a process with many threads associated to it. Each thread does a specific task (i.e. Spell checking, formatting, etc). If you create a new window that is independent of the first window, then the new window is its own process and has its own set of threads for spell checking, formatting, etc.

Question: When would you want to use the different types of user/kernel threads?
Answer: If your system has limited computing power, then you would want to use one kernel thread mapped to many user threads. This is good for embedded devices, especially if the device has a very niche application/use. On the other hand, the main operating systems use one-to-one kernel thread mapped to user thread – used by Windows and Linux.

Question: When the OS restricts the number of threads per process, what error message does it print?
Answer: The exit status code that is returned will indicate whether it was successful in creating a new thread or process. You can probably view the error in the `stderr` var/file.

Question: Is it possible to have a three level (thread) model? (i.e. Lightweight applications follow the one-to-many model. Moderate applications follow the many-to-many model. And, heavy applications follow the one-to-one model)
Answer: Yes, this is possible, but very hard to implement. It's a pain in the arse. Not even Microsoft with its deep pockets can do it.

Question: Are race conditions caused by concurrency?
Answer: Yes; anything with more than one thread/process is susceptible to race conditions. But if you only had one process, then you don't

have to deal with race conditions.

Question: What is “atoi”?
Answer: C function to convert string into integer

Question: How would you recover from an error like: Failed to fork, failed to create thread, failed to setup shared memory, system call failed, etc.
Answer: This mostly depends on the device, but you could: reboot, reset, force quit the application, stall the application, or retry. In embedded systems, the convention is to reset the device.

Question: What is the relationship between parallelism and concurrency? Is parallelism a subset of concurrency?
Answer: They are separate entities. One is not a subset or superset of the other. Parallelism is doing multiple things at once, like walking and chewing gum at the same time. Concurrency is doing multiple things, but one at a time (i.e. Walk for 10 steps, chew gum for 5 seconds, walk for 8 steps, chew gum for 3 seconds, etc.)

Question: Signal handling is an issue for threads, but is it also an issue for processes?
Answer: Yes it is. If processes are sharing the same memory location, then updating the processes on the value of the memory is a challenge, similar to signal handling in threads. The problems are identical for both threads and processes.

Question: What is the difference between a Mutex and a Semaphore?
Answer: Semaphores are more flexible than mutexes.

Question: Did the original DOS support concurrency?
Answer: The original DOS did NOT support concurrency.

Question: When a process is stopped, what saves its current state? The dispatcher?
Answer: Yes, it is the dispatcher’s job to move process from the ready queue to running state. The dispatcher saves the state of the current running process, loads the next process, and repeats.

Question: What is the relationship between Mac OS X and Darwin? Is OS X based off of Darwin? Where does the kernel play into all of this?
Answer: The OS X KERNEL is based off of Darwin; both are micro kernel architecture.

Question: Since the stack and heap grow toward each other, how can you ensure that they do not overlap each other?
Answer: Through things like Garbage Collectors, and good programming practices (i.e. Freeing memory when not in use).

Question: Does using a mutex on a global variable prevent other

threads from reading it?
Answer: No, Mutex only protects writing to memory. Other threads can read the value.

Question: What are the benefits of using implicit threading over explicit threading? And vice versa?
Answer:

Benefit: In a thread pool, threads aren't created from scratch. The system takes care of creation and termination - this is good for avoiding bad programming decisions. The compiler may also perform some optimizations.

Disadvantage: You lose control and the ability to tweak minuscule options.

Question: Would you say that race conditions are the least reliable security holes to take advantage of?
Answer: Mostly yes. If the race condition is hard to achieve, then the reliability of the security exploit is very low. However, in some instances, race conditiond can be easily triggered, despite going undetected for years. Most of the time, race conditions have a 50% chance of succeeding. The best exploits to take advantage of are overflows (i.e. Buffer, Stack, etc.)

Question: Can deadlocks be caused by an infinite loop? (i.e. Thread 1 has mutex lock. Thread 1 enters infinite loop, and does not release the lock. Thread 2 cannot acquire mutex lock)
Answer: Yes, this is possible. It is called a Livelock. A live lock is when a thread acquires a lock but enters an infinite loop. Even though the lock is not released, and other threads are unable to operate, the thread with the lock is operating and making progress. In a deadlock, the entire program is “frozen”, and no thread is making progress.

Question: In lecture you said that only writing to memory can cause race conditions; reading memory cannot cause race conditions. However, when you create a new thread/process, you request the kernel for “next_available_pid”. If two threads get the same value from the kernel, then this will cause problems. In this example, you are not writing to shared memory, instead you are reading data.
Answer: Reading memory cannot cause race conditions. In the example above, the issue is with writing memory. The value of “next_available_pid” needs to be incremented every time a process is created. The function that does this should be indivisible (i.e. Should not be broken up).

Question: Between deadlock prevention and avoidance, where does a watchdog timer fit?
Answer: It’s a different thing. It doesn’t prevent or avoid deadlock, but releases it. A Watchdog timer is good for livelocks. In a livelock the system is running and making progress, but the locks are never released.

Question: A major issue with the deadlock detection algorithm is when should it run? And there does not seem to be an ideal number. It’s a tradeoff between efficiency and time. However, is it possible to run a deadlock detection algorithm asynchronously. (i.e. User presses “Refresh” if the system hangs)
Answer: This is possible, but probably not advisable because running deadlock detection algorithm is computationally expensive and probably not something we want the user to be able to do. However, the implementation does matter. Good code won’t crash the system and will perform the necessary checks.

Question: When the allotted time quantum expires, the system checks to see if other processes need to be executed. Compared to context switching, is this less or more computationally expensive?
Answer: It is much (much) less computationally expensive. For reference, context switching takes less than 10 usec.

Question: What is a zero-day attack
Answer: A zero-day attack, also known as 0-day, is an exploit that takes advantage of a bug that is new, and unknown to the vendors of the product. This type of attack is very dangerous because there is no publicly available security patch/fix for the bug that is exploited.

Question: Can you give an example of how race conditions can lead to a security hole/bug?
Answer: Assume there's a region of protected memory at location X in RAM. A process requests to access the information at X. Typically, the system needs to check if the process is allowed to read the memory at X. However, simultaneously, the system loads the information at location X into cache. This is done to improve performance, because if the process is allowed to access the data, then putting it in cache improves performance. But, if the process is not allowed to access the data at location X, and the system is unable to verify the process' permissions BEFORE the information is copied to cache, then an unauthorized process will be able to view the data in cache (via some other tricks). This holds true even after the unauthorized process is denied access to the information at location X - the unauthorized process can still retrieve the data from cache (via special tricks).

Question: What is “smash-the-stack”?
Answer: On many C implementations it is possible to corrupt the execution stack by writing past the end of an array declared auto in a routine. Code that does this is said to smash the stack, and can cause return from the routine to jump to a random address.

Question: If a program (i.e. Photoshop) needs more RAM, does it ask the operating system and the OS asks the kernel? OR does it directly interface with the kernel through system calls?
Answer: The kernel is the part of the operating system that mediates

access to system resources. It's responsible for controlling access to CPU, memory, I/O, networking, etc. If a program (i.e. Photoshop) needs data that is currently not in RAM, the CPU signals this to the kernel, and the kernel responds by writing the contents of an inactive memory block to disk and replacing it with the data requested by the program. On a related note, the kernel dictates which memory a process can use, and appropriate action when memory is insufficient.

Question: What resources are shared among a parent and child process?
Answer: The processes have identical copies of the text, data, and (user) stack regions. Also, if the parent process had a file open at the time of the fork, then the child also has access to it.

Question: What is the difference between a parent and child process?
Answer: Neither process can access each others variables. Both processes have a different PID. The child process does not inherit process locks, text locks, and data locks.

Question: What is a daemon? And what is the point of it?
Answer: A daemon (pronounced as "dee-min" or "day-min") is a program that runs as a background process, rather than being in (direct) control of the user. The purpose of a daemon is to handle periodic service requests. For example, the `sshd` daemon listens for incoming `ssh` requests. It silently runs in the background, does not require user interaction, and services `ssh` requests.

Question: What is the difference between physical and virtual memory. Where would you use each one? And in what scenarios would you use one over the other?
Answer: Physical memory refers to the actual RAM of the system that is attached to the motherboard (or SoC) and contains instructions/data. On the other hand, virtual memory is a memory management technique that allows users to execute programs larger than the actual physical memory. Memory management allows processes to move between main memory and hard disk during the time of executing the program. In addition it provides things like security, through isolation. Both physical and virtual memory are used in a computer.

Question: Let’s assume there’s a process, X. Can ‘X’ tell if it is a zombie or orphan process? Can other processes tell if a process is a zombie or orphan process?
Answer: If a parent process is killed, then the children of the parent process are "adopted" by the `init` process and its children. The children processes can tell if the parent process has died if a signal is sent (from the parent to the children) before termination. However, other processes cannot tell if a child process is a zombie/orphan.

Question: What is an example of something that will have no improvements if parallelized?
Answer: Algorithms that are inherently sequential are difficult to

parallelize and yield little to no improvements. For example, many hash-chaining algorithms are very difficult to parallelize. The "Circuit Value Problem" is easy to solve with sequential algorithms, but very difficult to parallelize efficiently.

Question: What is a UAF?
Answer: UAF stands for "use after free". It is a class of software vulnerabilities where memory is erroneously used again after being deallocated. If the program does not clear the pointer to the previously allocated memory, then an attacker can use this to attack the system. The attacker will load malicious code into the section of memory that is referenced by the (dangling) pointer, and then use it to execute the malicious code.

Question: In a multi-threaded program, when you call `fork()` on a parent process, is it possible to only duplicate certain threads?
Answer: When `fork()` is executed, it makes a copy of all the threads. Then, the child process can call `exec` and discard all the threads. There are 2 different implementations of fork:

1. Duplicate all threads
2. Only duplicate the executing thread

Question: Can you explain NUMA?
Answer: NUMA stands for "non-uniform memory access". It is a computer memory design used in multiprocessor systems where the memory access time depends on the memory location relative to the processor. For instance, L1 cache is faster than L2 cache, which is faster than L3 cache, and so on. In a NUMA design, a processor can access its own local memory faster than non-local memory; memory that is local to another processor (i.e. Another processor's cache), or memory that is shared between processors. NUMA improves performance and the ability of the system to be expanded.

Question: How is a virus’ signature calculated? And if you change the (source) code of a virus, does that also change its signature?
Answer: A virus signature is an algorithm or hash that uniquely identifies a virus. If a virus signature is based on a static hash, then it is calculated via a snippet of code that is unique to the virus. In addition, a virus signature can also be behavior-based. For instance if a virus tries to delete a certain system file, modify a certain registry, and open a specific webpage, then this behavior is unique to the virus and becomes its signature. Depending on how the virus' signature is calculated, modifying the source code of a virus may change its signature.

Question: What techniques are used to protect the kernel? Is there hardware/software that protects the Kernel?
Answer: One way to protect the kernel is to use a language-based protection system. In a language-based protection system, the kernel will allow execute code that has been produced by a trusted language

compiler. This technique is software-based. Another way to protect the kernel is to minimize the attack surface. This involves moving as much code from kernel space to user space; which is exactly what micro kernels do. In Apple's A10 chips and newer, KTRR - a hardware based kernel protection method - is used prevent modification of an iOS kernel at runtime. For more techniques, read into "kernel hardening".

Question: Do attacks that rely on buffer overflows have a success rate like attacks that rely on exploiting race conditions?
Answer: Yes, buffer overflows do have a success rate, similar to race conditions. However, buffer overflows are much more reliable than race conditions.

Question: What is the best kind of vulnerability for an attacker? (i.e. Provides a 100% guarantee compromise of the system when the exploit is deployed. In other words, what type of bug is the most reliable to exploit?)
Answer: No exploit guarantees a 100% success rate. The success rate can be very close to 100%, but it won't be 100%. Furthermore, the way the exploit is crafted to take advantage of a vulnerability affects its success rate.

Question: Can you give more information on MULTICS? (i.e. What does it stand for?)
Answer: MULTICS stands for "Multiplexed Information and Computing Service". It is a time-sharing operating system based on the concept of a single-level memory. MULTICS removes the distinction between files and process memory. Despite its flaws, MULTICS has heavily influenced computer science.

Question: What is SMC, SVC, HVC, & ERET, and how does it relate to ARM CPU architecture?
Answer: SMC stands for "Secure Monitor Call". SVC stands for "Supervisor Call". HVC stands for "Hypervisor Call". ERET stands for "Exception Return". All of these are implemented in the ARM architecture, for the purpose of security.