Part 1. Get with the program (60 points)

Consider the short (and useless) C program off to the right. Each of the variables \{a, b, c\} references an integer value that occupies some memory at runtime. Answer the following questions for each of the integer values referenced by \{a, b, c\}.

Where is/are \{a, b, c\} stored in virtual memory? (i.e., what segment)

Data, stack, heap (BSS)

How/when is the virtual memory for \{a, b, c\} allocated? How/when is it released?

Process initialization (e.g., exec), stack frame push for main, malloc call.
Or: process creation for a and b, sbrk() for c, but VM is assigned to b on frame push, and to *c on malloc call.

When the main thread accesses \{a, b, c\}, how is the virtual address computed or determined?

Offset from data segment base (or static virtual address), fixed offset from stack pointer or frame pointer register, fixed offset from heap block base pointer stored on stack (in int* c).

How/when is the machine memory for \{a, b, c\} allocated? How/when is it released?

At a time of the OS choosing: typically on process instantiation, or on a later page fault.

When the main thread accesses \{a, b, c\}, how is the machine address computed or determined?

Insert short summary of page tables and TLBs, and protected page table registers (e.g., ASID register) as part of the process context.
Part 2. Déjà vu all over again (60 points)

Implement monitors using semaphores. As always, “any kind of pseudocode is fine as long as its meaning is clear”. Don’t forget to implement the condition variables! You may omit broadcast/notifyAll, and you may assume that all threads using your implementation are “well-behaved”.

You need four methods: lock/acquire, lock/release, wait, signal/notify.
Lock and unlock are P and V on a binary semaphore with initial value 1.
Wait must release the lock, P on a semaphore to sleep, and then reacquire the lock.
Signal must V on a semaphore to wake up a thread.

To really do it right you need one semaphore per thread, in addition to the binary semaphore. And you’ll need a list of waiting threads, which must be protected by another binary semaphore. See related problem on midterm2-13f.
Part 3. Waiting for a core (60 points)

In general, threads in the Ready state (Runnable) are linked into a list called a **ready queue** or runqueue. (More precisely, their thread objects or Thread Control Blocks are linked into a list.) Answer the following questions about ready queues. Each question is asking for **short phrases in the “lingo”** of this class. You can explain a little, but please keep it brief.

(a) What events would cause the kernel to place a thread on the ready queue? List as many causes as you can.

New thread, sleeping thread is awakened, thread is preempted while running.

(b) What events would cause the kernel to remove a thread from the ready queue? List as many causes as you can.

Thread or process death (killed or exited), dispatch of ready thread to a core (i.e., because the core was idle when this thread entered the ready list, or because the thread running on the core yielded (preempt), blocked, or exited.

(c) Most modern kernels have multiple ready queues. Why is it useful to have multiple ready queues? List as many reasons as you can.

Priority, concurrency. (1) Insert brief discussion of use of a ready list for each priority level, e.g., multilevel feedback queue. The getNextReadyThread for dispatch is in constant time, once we identify the highest-priority nonempty queue (which could be done with a special instruction). (2) Add brief statement that we might use a separate ready list for each core or group of cores, to reduce locking contention on the ready lists.

(d) If a thread is **not** on a ready queue, what other data structures in the kernel might point to it? List as many examples as you can.

Sleep queue, e.g., associated with a mutex, semaphore, or condition variable. Or maybe some sort of process/thread table. Or an array of pointers to the current thread running on each core.
Part 4. Brave new world (60 points)

Alice, Bob, and their friends interact via the social networking service Faith. They protect their communication using **secure sockets (SSL) for transport-layer security**. Nancy wants to spy on their communication. Assume that Nancy can subvert any network provider and so intercept and/or inject packets at any point in the network.

(a) Nancy cannot “break” the cryptography, but she can still spy on SSL-encrypted communication if she can obtain the right keys. Outline two approaches that Nancy might use to obtain useful keys. Be sure to explain how Nancy would use the keys obtained in each approach (if it succeeds).

1. Crack into Faith, e.g., by social engineering attack, buffer overflow exploits, or other means, and steal Faith’s private key. Then use the key to masquerade as Faith in the handshake, or forge signatures of Faith, or decrypt communications to/from Faith.

2. Get approved by a browser distributor as a Certifying Authority (CA), or use the means above to steal the private key of some other CA, or infiltrate the CA by other means (e.g., warrant, bribes). Then mint a keypair, and use the CA key to forge a certificate endorsing the new public key falsely as Faith’s public key. The fake cert enables the attacker to masquerade as Faith, even when “secure HTTP” is used.
Part 4. Brave new world (continued)

(b) Now suppose that Nancy also wants to modify the SSL-encrypted communication between Faith and her clients as it occurs, without detection by any of them. Can she do it using the approaches from (a)? Explain.

Either approach above, if successful, allows Nancy to mount a man-in-the-middle attack. Describe MITM attack and how it allows Nancy to arbitrarily modify communications between Faith and her clients without detection.

(c) Faith and other service providers are planning to enhance their security with a scheme called Perfect Forward Secrecy to defend against the following vulnerability of SSL: if Nancy obtains Faith's private key, then Nancy can decrypt all previous communication between Faith and her clients, assuming that Nancy has also intercepted and retained a copy of those network packets. Explain how Nancy would use the key to decrypt the saved traffic.

Each SSL connection uses symmetric encryption with a session key (shared secret) negotiated during the initial SSL handshake. The session key is chosen by the client (Alice, Bob) and encrypted with the server's public key (or the MITM attacker’s public key). Anyone possessing the corresponding private key can decrypt the session key and use it to decrypt all of the data passed over the connection.
Part 5. Spinning in circles (60 points)

This part deals with performance and data safety for a data storage volume under various designs. Suppose that the system issues a stream of read and/or write requests for random block numbers in the volume. For example, these requests might result from some workload on a file system similar to your DeFiler (Lab #4).

In the baseline design the storage volume resides on a single disk drive (HDD). The following questions each propose a change to the baseline design, and ask you to state the effect of the change relative to the baseline design. You may assume that the request stream is continuous and balanced: the next request arrives for a disk just before its previous request completes. I am looking for answers that are specific and quantitative. You might need some other assumptions: if so, be sure to state them.

(a) If we double the rotational speed or areal density of the single disk, how does it affect throughput and the average per-request response time, other things being equal?

Transfer time is cut in half, and if rotational speed doubles, then rotational delay is also cut in half. Seek time is unchanged. So response time decreases, but by less than half. Therefore the throughput will increase, but by less than a factor of two.

(b) If we double the block size, how does it affect throughput and the average per-request response time, other things being equal?

Transfer time is doubled. Seek time and rotational delay are unchanged. So response time increase, but by less than double. Therefore the throughput in operations/second will decrease, but by less than half. Throughput in bytes per second will increase, but by less than double.

(c) Suppose we distribute the volume across N disks using striping (RAID-0). How does it affect throughput and the average per-request response time, other things being equal? How does it affect the probability of a failure that results in data loss?

Response time is unaffected: each disk takes just as long to access a block as it did before. Throughput increases by a factor of N, on average, since we can do N block accesses in parallel. Probability of a failure increases by a factor of N.

(d) Suppose we distribute the volume across N disks using mirroring (RAID-1). How does it affect throughput and the average per-request response time, other things being equal? How does it affect the probability of a failure that results in data loss?

Response time is unaffected. Read throughput increases by a factor of N, on average, since we can do N block reads in parallel. Write throughput is unaffected: we can’t write to N disks any faster than we can write to one disk. Probability of a failure decreases enormously: all disks must fail to lose data: that probability is taken to the Nth power.
Part 5. Spinning in circles (continued)

(e) Suppose the volume has N disks, and blocks are distributed across the disks using **striping with parity** (RAID-5). How does it affect throughput and the average per-request response time, other things being equal? How does it affect the probability of a failure that results in data loss?

*Read throughput improves by a factor of N-1. Response time is unchanged (barring queuing delays). Write throughput improves by a factor of N/2. Every random block write must write to two drives: one for data and one for parity, so on average only N/2 write operations may proceed in parallel.*

(f) Now suppose the system implements **atomic commit** for groups of writes using **logging**. How does it affect throughput, other things being equal?

*Every write causes two block writes: one to the log and one to the home block location. If blocks are only partially written, then we can write just the changed bytes to the log. But if we have to write the whole block, then write throughput declines by half, and write response time roughly doubles. Write response time might be worse since we have to wait for the write to complete. Of course, the log might be on a separate disk, but in any case, write bandwidth is gated by the log bandwidth, which might be faster than random writes, given that the log is written sequentially. Read throughput and response time are unaffected, leaving aside queuing delays.*

(g) Now suppose the system implements **atomic commit** for groups of writes using **shadowing**. How does it affect throughput, other things being equal?

*When we modify a block, we write a new block wherever we want on the disk, and leave the old one unchanged. We have to write some block maps (small), and we have to release old block copies we don’t need anymore (fast), so generally write throughput may even improve given that we can write new blocks close together. Write response time may go up if we have to wait for all of the writes in the group to complete. Read throughput and response time should be unaffected.*

(h) Suppose the system caches blocks in memory, using a structure similar to your Lab #4 buffer cache. You may assume that the buffer cache can hold **M** blocks out of a capacity of **C** blocks for the disk volume, where **M < C**. How does adding the cache affect throughput and the average per-request response time, other things being equal?

*Since the accesses are random, we expect a hit rate of M/C. If hits have zero cost, then throughput could go up by a factor of C/(C-M): we can serve C requests in the time it would have taken to serve C-M (the misses that actually go to disk). Average response time is reduced by a factor of (C-M)/C. Of course throughput and response time at the disk system itself are unchanged.*
Part 6. Extra credit

What is a thread? Illustrate with a picture. Extra extra points if you can explain it in a more entertaining way than I did. (Low bar?)
program: Init
Thread: start

horr...