Question 1

Implement counting semaphores using mutexes and mesa-style condition variables.

```
struct semaphore {
  int   count;
  mutex lock;
  cv    incr;
}

P(semaphore s) {
  s.lock.acquire();
  while s.count <= 0
     s.incr.wait(s.lock);
  s.count--;
  s.lock.release();
}

V(semaphore s) {
  s.lock.acquire();
  s.count++;
  s.incr.signal();
  s.lock.release();
}
```

Question 2

With 1KB file system blocks and 4 byte block pointers, what is the maximum file size on a UFS-like system with 12 direct blocks, 1 single-indirect block, and 1 double-indirect block?

$$1024 * (12 + 256 + 256^2)$$

Question 3

Why have we not discussed RAM access scheduling (as we have disk head scheduling)?

RAM is random access; there is no benefit to reordering requests.

Question 4

L1-cache block size: 32 bytes (2^5)
L1-cache access time: 1ns
L1-cache size: 1024 bytes (2^{10})
Page size: 4096 bytes (2^{12})
RAM access time: 10ns
Memory size: 8192 bytes (2^{13})
Disk access time: 10ms

Thee L1-cache is fully-associative and uses a random replacement policy. The pager is LRU. **What is the effective access time for an application walking through an array of size 16384 (2^{14})bytes (it accesses every 4-byte word)?** Assume the application is looping through the array and has been running for a long time.

Begin with the general cache access time formula:

$$EAT = P_{hit} * T_{hit} + P_{miss} * T_{miss}$$

We want the EAT of the L1-cache. The T_{L1miss} is just the EAT of the next level in the memory hierarchy, the RAM:

$$\begin{aligned} EAT_{L1} &= P_{L1hit} * 1ns + P_{L1miss} * EAT_{RAM} \\ EAT_{RAM} &= P_{RAMhit} * 10ns + P_{RAMmiss} * 10ms \end{aligned}$$

What is P_{L1hit}? The worst we can do is 1 miss every 8 references, because 8 references in a row go to the same cache block. There is some very small chance that, because of the random replacement policy, a given block is in the L1 even though it hasn't been referenced for a long time (i.e. the other 511 cache-block references the application makes while walking the loop). It is safe to say this probability is too small to be of concern. Thus, $P_{L1hit} = 7/8$.

What is P_{RAMhit}? The cache makes 32 byte requests to memory. This means it will make 128 requests in a row for each block. Since LRU will never have the next needed block in memory, the first of these 128 references will miss and require a disk access. $P_{RAMhit} = 127/128$.

Plugging these numbers in, $EAT_{L1} = 9.7677\mu s$. Note that despite the terrifically small probability of a RAM miss (1/1024), going to disk is so expensive that the EAT is more than 4500 times the EAT of a system with 4 physical pages.

If the pager is replaced with a most-recently used pager, what is the EAT?

One in four of the page faults can be made into hits by using MRU. (you should run through a few loops to see this)[1].

Thus, out of the 512 total memory references, only 3 will be page faults (rather than 4 before); $P_{RAMhit} = 509/512$.

$EAT_{L1} = 7.3263\mu s$.

[1]This means that MRU is a better approximation to OPT than LRU for this reference pattern.