CS 4410 Operating Systems
Prelim II, Fall 2012
Prof. Sirer

Name:__________________________________ NETID:_________________

- **This is a closed book examination. It is 9 pages long. You have 120 minutes.**
- **No electronic devices of any kind are allowed.**
- **If you are taking this exam during the makeup period, you may not leave the exam room prior to the end of the exam, and you may not take an exam booklet with you.**
- **You may use Python, C, or low-level pseudo-code in answer to any coding question. Descriptions written mostly in English will get no credit as responses to coding questions. In addition to correctness, elegance and clarity are important criteria for coding questions.**
- **Assume Mesa semantics for monitors and condition variables.**
- **All essay questions are limited to 8 sentences in length. The course staff will first insert punctuation into your answer as they deem appropriate, then stop reading at the eighth period. Please provide concise answers.**

[15] **1.** From the previous prelim, you know that it is often useful to have a TriableLock construct which enables callers to check if the lock is "in use" (that is, acquired by another thread), and if it is in use, to return a failure indication to the caller so that it can do something else instead of blocking until the TriableLock is released.

Specifically, a TriableLock exports the following interface:

– Lock: acquire the TriableLock if it is currently available; if not, block the calling thread until the lock is released and can be acquired. At most a single thread can acquire the lock at any time.

– Unlock: release the TriableLock.

– TryLock: acquire the TriableLock if it is currently available and return true; if not, return false immediately, without blocking.

Recall that a Lock operation issued by a thread T2 can block for an extended amount of time if the TriableLock has previously been locked by another thread T1. In contrast, the running time of TryLock should be independent of the length of time for which any thread holds the TriableLock.

Your task is to implement a TriableLock using monitors and condition variables. Your implementation should adhere to the functional specification provided above, and it should make progress whenever it is possible to do so.

```
Class TriableLock:
    def __init__(self):



   def Lock(self):



   def Unlock(self):



   def TryLock():
```

[15] **2**. You have been hired by Troll Tech Enterprises to build a distributed sound system. Your goal is to make a collection of laptops at a flashmob event stream the same song at the same time such that one person can DJ the music played on all of these laptops simultaneously as if it were a makeshift sound system. Not surprisingly, Troll Tech's initial attempts to play Gangnam Style using an unsynchronized system failed because the laptops slowly drifted out of synchrony as the song progressed. This is why you have been brought in.

To simplify the problem, we will take a divide and conquer approach: assume that a song consists of a number of chunks. Your task is to ensure that all laptops start playing a chunk at the same time, and wait for all laptops to finish playing their chunk before going onto the next chunk. This will re-synchronize the laptops at the beginning of each chunk; if chunks are small, the occasional pauses that are introduced at the end of each chunk (to wait for the slowest laptop) will be imperceptible (at least, to Gangnam fans). You can see that we're implementing a synchronization barrier: threads reaching the barrier code wait until all other threads have reached the same point.

We will assume that there is a single thread acting as a source for each of the laptops, whose code is shown below.

Your task, then, is to use monitors with Mesa-style condition variables to implement the Barrier synchronization primitive for this sound system.

Hint: *Since we are reusing the same barrier for multiple rounds of synchronization, there may be some subtle issues. In particular, make sure that your barrier works correctly not only for the first chunk, but every subsequent chunk as well. Be sure that a thread which has played chunk i, entered the barrier, exited the barrier because every other thread reached the barrier for the ith chunk, played chunk (i+1) and re-entered the barrier does not cause problems for your implementation.*

Comment*: This question is exactly the same as in the previous prelim. This is intentional.*

```
global chunks[NUMCHUNKS]
global b = Barrier(NUMPLAYERS)

class Player(Thread):
  def run():
      for i in range(NUMCHUNKS):
           streamplay(chunk[i])
           print “I've got Gangnam Style, played round %d!” % i
           b.barrier()


for i in range(NUMPLAYERS):
   Player().start()
```

```
class Barrier:
   def __init__(self, N):
          self.N = N







   def barrier(self):
      # this routine must not return until N threads have reached the barrier
      # the barrier must be reusable over and over again for multiple rounds of synchronization
      # be sure that the situation described in the Hint does not cause a synchronization error
      # your implementation should make forward progress whenever it is possible to do so
```

[20] **3.** You have been brought in to diagnose a performance anomaly, encountered in the real world: User Alice reports that she has eight independent tasks that she would like to run. All the tasks are known to be single-threaded and to perform no communication and no other I/O. When Alice runs any one of the tasks *by itself* on an 8-core machine that is not running any other processes, each task takes 5 minutes to complete and the disk is mostly idle. When she runs all 8 tasks *concurrently* on a machine that is not running any other processes, each and every one of the tasks take 500 minutes to finish, and the disk shows signs of constant activity. She is outraged that executing eight independent tasks one after another takes 40 minutes, and yet executing them concurrently on an 8-core parallel machine takes much longer than serial execution. Assume that her computer is a multi-core machine of the kind you have encountered in CS3410/CS3420/ ECE3140.

a. Describe the most likely scenario that would account for this anomaly.

b. What is the technical term used to describe the underlying situation?

c. What would you recommend Alice to do to her system to speed up the completion time of all of her 8 tasks?

d. What's your ballpark estimate of how long Alice's tasks will take to finish if she enacts your suggestion in part (c)?

______________ minutes

[10] **4.** You have been brought in to diagnose a different performance anomaly, also encountered in the real world.

A caching web proxy is an application deployed within the network to intercept user requests destined to remote web servers. If the contents are not available within the caching web proxy, the caching proxy fetches them from the remote server, provides them to the client, and keeps a local copy in memory. But if they are available locally, they are served from the caching web proxy without contacting the remote server, thereby speeding up access. A large caching web proxy installation is deployed at the network border to the Australian continent, for instance, to avoid having Australian users from having to go all the way across the Pacific to fetch their content.

The particular caching web proxy deployed in Australia is called Squid. As an optimization, Squid developers wanted to use both the memory and the disk available on the Squid host, so they wrote their own code, at user-level, to keep most recently accessed content in memory and to push least recently accessed content to disk. As a result, when Squid faces cache pressure, it takes steps to write in-memory content that is least recently used to disk, and uses the memory, thus freed by pushing contents out to disk, for other, more recently used items (whether they are brought in from remote web servers, or from the local disk).

In essence, Squid explicitly performs, at user-level, some of the same activities that the virtual memory subsystem performs at the kernel level. Note that Squid's LRU operates on top of Linux, which uses the CLOCK algorithm in its virtual memory subsystem.

Users report that Squid, with this optimization enabled, is *slower* than Squid with this optimization turned off!

Describe the problems that can arise from the interaction of explicit page management in Squid and the underlying virtual memory subsystem in Linux. Specifically, describe, in full detail, the extra work that the system has to perform when the eviction choices made by Squid at the application level differ from those made by Linux.

[20] **5.**

a. Recall that TCP's congestion control algorithm uses AIMD (additive-increase, multiplicative-decrease) to manage its window size in response to successful transmission and perceived loss, respectively. Describe why MIMD (multiplicative-increase, multiplicative-decrease) would be a poor choice of a replacement for the AIMD approach.

b. It's the year 2020: the US government is more divided and partisan than ever before and the economy has truly tanked. You operate a green grocer where you buy tomatoes from suppliers and sell them to the public. A newly created agency called the Federal Tomato Commission (FTC) wants to enact regulations to improve public health, which will require you to wash, and keep clean, every tomato to ensure that it is guaranteed free of harmful bacteria at the time of sale. Use the end-to-end argument to argue that such regulation is a bad idea.

c. Your great success as a green grocer has enabled you to rise quickly through the ranks, so now, you work for the Federal Tomato Commission. The FTC wants all green grocer employees to wash their hands after going to the bathroom. Use the end-to-end argument to argue that such regulation is a good idea. Explicitly state why your argument is consistent with both the end to end argument and your answer to part (b).

d. Given that TCP implements a reliable transport layer, achieved through retransmissions and acknowledgements, do applications need to send explicit acknowledgements generated at the application layer and transmitted in the payload field of a TCP packet? Why or why not?

[20] **6.** For the following question, assume that you're operating on a machine with the following parameters and setup:

instruction set is identical to MIPS
all addressing is virtual
maximum size of virtual memory: 64 bits
maximum size of physical memory: 48 bits
page size: 2^{16} bytes
there are two processes, named P1 and P2
P1 consists of a text segment at virtual memory location 0x10000,
extending for just one page, whose contents are on
physical page (frame) at 0xA0000,
and a data segment at virtual memory location 0x20000,
extending for just one page, whose contents are on
physical page (frame) at 0xB0000,
P1 has no stack and no heap.
P2 consists of a text segment at virtual memory location 0x10000,
extending for just one page, whose contents are on
physical page (frame) at 0xC0000,
and a data segment at virtual memory location 0x40000,
extending for just one page, whose contents are on
physical page (frame) at 0x00000, (yes, 0x00000)
P2 has no stack and no heap.
actual size of physical memory: 16 physical pages (frames)

Note that both P1 and P2 use the same virtual memory address for their text segment, but have different contents at that address.

a. Show the layout and contents of the single-level page tables for P1 and P2. Be sure to note the contents of the V field. You may omit repeated entries within a given table.

b. Assume now that P1 and P2 have a library in common. This library has only a single page of code in its text segment (no data, no heap, no stack). It is mapped to virtual memory location 0x30000 in both processes and backed by physical memory at address 0xF0000. Note that P1 and P2 are mutually distrusting. Show the layout and contents of the single-level page tables for P1 and P2. Show the contents of the V, P, R, W, and X bits. You may omit repeated entries within a given table.

c. Assume that process P1 issues a load from address 0xDEADBEEF. Walk through the entire process and highlight what the MMU and OS would do. Hint 1: you should not have to ask whether 0xDEADBEEF is a virtual or physical address. Hint 2: your response should clearly express whether the operation succeeds or not.

d. Assume that process P2 issues a load from address 0x00000. Walk through the entire process and highlight what the MMU and OS would do different from your answer to part c. Hint 1: you should not have to ask whether 0x00000 is a virtual or physical address. Hint 2: your response should clearly express whether the operation succeeds or not.

e. Assume that page table entries are each 8 bytes long. Calculate the sizes of single-level and two-level page tables for P1. For the two-level page table, assume that the first 10 bits are used for the first level, and the remaining bits are used in the second level page table. Assume that the library from part (b) is still mapped into P1's address space.