CS 4410 Operating Systems
Prelim I, Fall 2011
Prof. Sirer

Name:_____ NETID:_____

- **This is a closed book examination. It is 8 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 all monitors. Show your work for partial credit. Brevity is key.**

[7] **1.** Which of the following operations require the executing code to be operating with high privilege, and why?

- Issuing a system call
- disabling interrupts
- issuing a write to an I/O device, e.g. the disk, keyboard or the network card
- issuing a read from an I/O device, e.g. the disk, keyboard, or the network card
- reading the system clock
- setting the system clock
- implementing a monitor lock for threads

Answers which do not describe why will receive no credit.

[5] **2.** Give two reasons why this is a bad implementation for a lock, assuming that the system provides two atomic functions, disable_interrupts and enable_interrupts, that perform their namesake operations:

```
class Lock:
    def __init__(self):
        self.oldlevel = 0
    def acquire(self):
        self.oldlevel = disable_interrupts()
   def release(self):
        enable_interrupts(self.oldlevel)
```

[8] **3.** Indicate if true or false, then explain your answer. Answers without an explanation will receive no credit.

A. Threads in the same address space share the same heap and stack.

B. A microkernel operating system uses multiple address spaces inside the operating system, with components such as the file system, the networking stack, the user authentication module, etc. executing in separate user-level processes.

C. Every semaphore initialized to 1 is a binary semaphore.

D. Every semaphore initialized to a value higher than 1 is a counting semaphore.

[20] **4.** Use monitors and condition variables to synchronize the simulation for a chemical reaction. In this simulation, threads model atoms that enter a reaction chamber. An atom announces its presence by calling the appropriate *_available() method for its type. This method should implement the following functionality:

1. The *_available() method should not return to its caller until a successful reaction has taken place.

2. The reaction chamber is very small. So, while a carbon or oxygen atom is in the reaction chamber, all other available carbon and oxygen atoms should wait until the chamber is empty.

3. Hydrogen molecules will preferentially react with a carbon atom if carbon is present, or an oxygen atom if a carbon atom is not available but oxygen is. Any number of hydrogen atoms can enter the reaction chamber.

4. Only two reactions are possible: 4 hydrogen atoms can react with a carbon atom to form methane, or 2 hydrogen atoms can react with an oxygen atom to form water. Once the reaction chamber contains sufficient atoms of the right kind for a reaction to take place, all atoms involved in that reaction, (4H and 1C) in the former case, 2H and 1O) in the latter case, should leave the reaction chamber.

On the next page, fill in the monitor for the reaction chamber, making sure that the system can make progress whenever it is possible to do so.

```
class ReactionChamber:
        def __init__(self):



        def hydrogen_available(self):



        def oxygen_available(self):



        def carbon_available(self):
```

[20] **5**. Use semaphores to provide synchronization for a simulation of a EuroSoviet bureaucracy. In this simulation, we have government workers and citizens. There are N threads corresponding to the bureaucrats, and any number of threads that correspond to the citizens. The citizens go up to a counter one at a time to receive an approval from each of the bureaucrats (one checks educational history, one checks criminal history, one checks tax history, and so forth). When a citizen have received a stamp from all N unique bureaucrats, are free to go. Make sure your code works for any N. Also, make sure that the citizen's arrival triggers all the bureaucrats into working concurrently with each other.

```
class Bureaucracy:
   def __init__(self):



   def citizen_ready(self):
      # this routine must not return until N unique bureaucrats have issued an approval



   def grant_approval(self, workerid):



french_embassy=Bureaucracy()
for i in range(0, N):
         b = Bureaucrat(i)

class Bureaucrat(Thread):  #thread startup code has been omitted for brevity, assume its presence
   def __init__(self, id):
         self.id = id

   def run(self):
      while True:
        french_embassy.grant_approval(self, self.id)

class Citizen:
         french_embassy.citizen_ready()
         print “Yay, I have been approved”
```

[20] **6.** You have been hired to fix the following code, which was found in the guts of a real, deployed web service for online discussions. The discussion service is split up into different forums, and clients of the service can post comments in different forums. A naïve implementation of online forums would perform a database transaction every time a user entered a comment. But since database transactions are slow and discussions on hot topics may overwhelm the database, it makes sense to delay and batch database updates.

The code below is intended to do just that. When a client posts to a forum, he sends an HTTP request that is handled by a freshly created thread, which in turn invokes the post_comment method on the web service. Instead of immediately writing to the database, this method simply stores the modification in memory, and marks the forum as needing to be flushed to the database, by placing it on the list of modified_forums. A separate thread runs periodically every 10 minutes (MIN_WAIT_TIME), waits for modified forums to be posted, and writes any modified forums back to the database.

You are told that the program “does not work.” The programmer who wrote it has been banished to the great cold up north and cannot be contacted.

Find and fix the problem.

```
MIN_WAIT_TIME = 10 * 60
class BatchedForums:
   def __init__(self):
         self.lock = Lock()
         self.modifications = Condition(self.lock)
         self.modified_forums = set()

   def post_comment(self, forum, comment):
      with self.lock:
         forum.contents += comment
         if forum not in self.modified_forums:
            self.modified_forums.add(forum)
            self.modifications.notify()

   def run(self):
      while True:
         with self.lock:
            # wait until there is work to do
            self.modifications.wait()
            while len(self.modified_forums) > 0:
               forum = self.modified_forums.pop()
               write_to_database(forum)
         time.sleep(MIN_WAIT_TIME)
```

[10] **7**. The table below shows the arrival of tasks on a computer system.

Process	**Arrival Time**	**Processing Time**
P1	0	2
P2	1	6
P3	4	1
P4	7	4
P5	8	3

A. Show the scheduling order for these processes under 3 policies: First Come First Serve (FCFS), Shortest-Remaining-Time-First (SRTF), Round-Robin (RR) with timeslice quantum = 1. Assume that context switch overhead is 0 and that new RR processes are added to the head of the queue and new FCFS processes are added to the tail of the queue.

Time Slot	FCFS	SRTF	RR
0			
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			
16			
17			

B. What is the average response time under SRTF versus RR?

[10] **8**.
A.Name and describe the four necessary conditions for deadlock.

B. You have been hired to fix the following code, which was intended to perform a swap of the contents of shopping carts A and B.

```
class ShoppingCart:
        def __init__(self, id):
                self.id = id
                self.lock = Lock()
                self.contents = []
        ...
def swap_cart_contents(cartA, cartB):
        with cartA.lock:
                with cartB.lock:
                        tmp = cartA.contents
                        cartA.contents = cartB.contents
                        cartB.contents  = tmp
```

The users report that the code just “hangs” sometimes. Find and fix the bug in swap_cart_contents.