CS 4410 Operating Systems
Prelim II, Fall 2010
Prof. Sirer

Name:___________________________________ NETID:_________________

- **This is a <u>closed book</u> examination. It is 7 pages long. You have 120 minutes.**
- **No electronic devices may be used during the exam.**
- **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. Show your work for partial credit. Brevity is key.**

[15 points] 1. **Synchronization**
Recall the dining philosophers problem, where every philosopher is modeled by a thread with a unique philosopher identifier between 0 and N. The correctness criteria for this problem are two-fold:

SAFETY: Every philosopher must have exclusive access to a left and right chopstick prior to eating (i.e. executing the eat() routine below).
PROGRESS: The philosophers make progress whenever it is possible to do so, without running into a deadlock condition.

A semaphore-based solution for the dining philosophers problem is shown below. Recall that Python semaphores provide acquire() and release(), which are synonyms for P() and V(), and are initialized with an integer at the time of their creation.

Describe if the code is correct or not. If it is incorrect, *describe the problem(s)* and *provide a new version of the code* that is correct. Your solution must use only semaphores for synchronization, and must not unnecessarily limit the concurrency at the philosophers' table.

```
#create an array of chopsticks
chopstick = []
for i in range(0, N):
      chopsstick.append(Semaphore(0))

def philosopher(int id):
   while True:
      left = id
      right = (id+1) % N
      chopstick[right].acquire()
      chopstick[left].acquire()
      eat()
      chopstick[right].release()
      chopstick[left].release()
```

[15 points] 2. **Synchronization**
It's the year 2030. After years of “corporation-friendly” administrations, the country is falling apart. Because no one wants to pay taxes, all bridges across the Cornell campus have crumbled. Lack of clean water and air standards have released pollutants that cause organisms to change color – living organisms on north side of the campus turn blue; the same organisms crossing into the main campus spontaneously turn red. And severe budget cuts have forced Cornell to raise money by admitting monkeys into its undergraduate program (the faculty were replaced by orangutans a long time ago). Monkeys are housed in North Campus dorms, and cross across the gorge via a rope bridge.

Your task is to write a computer program to simulate and coordinate the behavior at the rope bridge. Each monkey is modeled by a separate thread. Red monkeys want to cross the bridge towards the north campus, and blue monkeys want to go in the opposite direction. The bridge can hold at most three monkeys at any time; if more than three monkeys are on the bridge, it collapses, and all monkeys fall off and die. If monkeys moving in opposite directions are let onto the bridge at the same time, they collide, fall off the bridge and die.

On the next page, use Python monitors and condition variables (which provide Mesa-style semantics) to implement the RopeBridge monitor that solves this problem efficiently without killing any monkeys. Make sure that your solution makes forward progress whenever it is possible to do so. For each use of your condition variables, *explain in English what that particular operation is trying to achieve.*

The code for a monkey appears below.

```
bridge = RopeBridge()

class Monkey(Thread):
      def __init__(self, initialcolor):
            self.color = initialcolor

      def run(self):
            while True:
                  # call the monitor to indicate I'm ready to cross
                  if self.color == RED:
                        redmonkey_ready()
                  else:
                        bluemonkey_ready()
                  #
                  # yay, I can cross now! Hope the monitor did the
                  # right thing, so I won't fall off.
                  #
                  cross_bridge()

                  # change color
                  if self.color == RED:
                        redmonkey_done()
                        self.color = BLUE
                  else:
                        bluemonkey_done()
                        self.color = RED
```

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



      # returns only when it is safe for the red monkey to cross
      def redmonkey_ready(self):



      # red monkey finished crossing the bridge
      def redmonkey_done(self):



      # returns only when it is safe for the blue monkey to cross
      def bluemonkey_ready(self):



      # blue monkey finished crossing the bridge
      def bluemonkey_done(self):
```

[15 points] 3. **Memory Management and Page Table Structure**

Assume that pointers and page table entries are 4 bytes. Suppose we have 2^{32} bytes of virtual memory and a page contains 2^{14} bytes. Process P1 accesses virtual memory range 0 through 2^{18}-1.

a. What is the size of a 1-level page table for Process P1?

b. What is the size of a 2-level page table for Process P1? Use 9 bits for 1st level and 2nd level page

c. What is the size of 3-level page table for Process P1? Use 6 bits for 1st, 2nd, and 3rd level.

[15 points] 4. **Page Replacement**

Consider a computer with an address space of size 2^{20} bytes. Each page on this computer is 2^{16} bytes big. This computer has only 2^{18} bytes of physical memory.
Consider the following sequence of accesses, where each value in the sequence below refers to a virtual page number:

7, 5, 3, 1, 2, 7, 1, 4, 5, 2

a. Assuming that memory starts empty, how many page faults will occur and what will be the final contents of memory under the FIFO page replacement policy?

b. Assuming that memory starts empty, how many page faults will occur and what will be the final contents of memory under the LRU page replacement policy?

c. Assuming that memory starts empty, how many page faults will occur and what will be the final contents of memory under the OPT page replacement policy?

5. Networking

a. [5 points] Describe the purpose of the “port” field in a UDP packet.

b. [5 points] Describe the purpose of the “port” field in a TCP packet.

c. [5 points] Describe why the “port” fields are not part of the IP packet., but appear separately in both UDP and TCP packets.

d. [10 points] In three sentences or less, state the end-to-end argument.
Note: The graders will first insert punctuation into your answer as dictated by standard English rules, and will then stop reading when they encounter three periods, dashes, or semicolons., or when they've decided that they have encountered three sentences worth of discussion. So, please express your answer succinctly in the space below; there is no need to re-generate the entire paper.

e. [5 points] Ethernet hosts detect collisions and retransmit packets in response. TCP also performs end-to-end retransmission of packets. Describe if the retransmission performed at the Ethernet level, within the network, violates the end-to-end principle? Why or why not?

f. [10 points] What is marshalling? When, during an RPC, does marshalling take place?