CS 4410 Spring 2016
Prelim 2

April 28, 2016

Full Name: ______________________

NetID: ______________________

Problem	Points	Score
1	10	
2	20	
3	16	
4	15	
5	12	
6	12	
7	15	
Total	100	

Instructions. The exam is 11 pages long. There are 7 problems in the exam. You will have 120 minutes to work on the problems. The exam is worth 100 points total.

Do all written work on the exam itself.

Show your work: partial credit will be awarded for incorrect answers that are headed in the right direction.

This is a closed-book examination. You may not use books, computers, printed materials, calculators, etc. Please turn off and remove from sight all mobile phones, tablets, and all other electronic devices.

Coding questions You may use Python, C, or low-level pseudocode in answer to any of the coding questions. 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.

Declaration of academic integrity. Academic integrity is expected of all students at all times, whether in the presence or absence of members of the faculty. Understanding this, I declare I shall not give, use, or receive unauthorized aid in this examination. I acknowledge that the minimum penalty for violating academic integrity is failure of the exam.

Signature ______________________

Do not open the exam booklet until instructed to begin.

1. True – False [10 pts]

   Label the following statements with either "true" (T) or "false" (F). Correct answers receive one point, blank or omitted answers receive a half point, and incorrect answers receive zero points.

a. ____ If a packet is dropped, TCP will increase the window size.

b. ____ TCP uses multiplicative increase, additive decrease during its normal operation.

c. ____ DNS maps human-readable names to hardware (NIC) addresses.

d. ____ A computer system is likely to be thrashing if a process' working set size is too small.

e. ____ FIFO page replacement can suffer from Belady's anomaly.

f. ____ Reactive routing protocols discover routes to all possible destinations before they are needed.

g. ____ Proactive routing protocols are not employed on the Internet due to their high cumulative costs.

h. ____ The IP layer ensures in-order delivery.

i. ____ Applications that need in-order delivery, but want to implement a different congestion control mechanism than TCP's congestion control mechanism can implement their own protocol on top of TCP sockets.

j. ____ Applications that need in-order delivery, but want to implement a different congestion control mechanism than TCP's congestion control mechanism can implement their own protocol on top of UDP sockets.

2. Synchronization [20 pts] You and your minion have been hired by the Gonzo Corporation to improve the performance of their key-value store. The key-value store implements a web API consisting of PUT and GET operations, built on top of a database module that stores and retrieves objects on disk.

   The source code for Gonzo's implementation is shown on page 8 in the appendix.

   Gonzo has decided that responding to PUT requests quickly is more important than ensuring that data is not lost in the case of failure, so they have designed their service to store PUT requests in memory and perform the expensive database writes in the background. For the purposes of this question, you may assume that there are no failures.

   (a) [10 pts] Despite Gonzo's implementation strategy, testing has found that in a write-only benchmark, the database seems to work efficiently under low load, but takes just as long as the original code as load gets heavier. Under very heavy load, some operations experience incredibly long delays. Further, read operations have slowed down significantly.

   In one to five sentences, explain what might cause this behavior.

   (b) [10 pts] In an attempt to address some of these problems, your minion produces a new version of the code, shown on page 10 in the appendix. This code seems to run more efficiently, but Gonzo's clients report that calls to GET sometimes return stale data.

   In one to five sentences, explain what might be causing this behavior.

3. Paging [16 pts]

Suppose you wished to design a tiny virtual memory system with an 8-bit physical address space and a 16-bit virtual address space. Suppose you wish to use a page size of 16 (2^4) bytes. Suppose that each page has 3 permission bits (read, write, execute) and a valid bit.

Note: for full credit you must indicate how you got your answers.

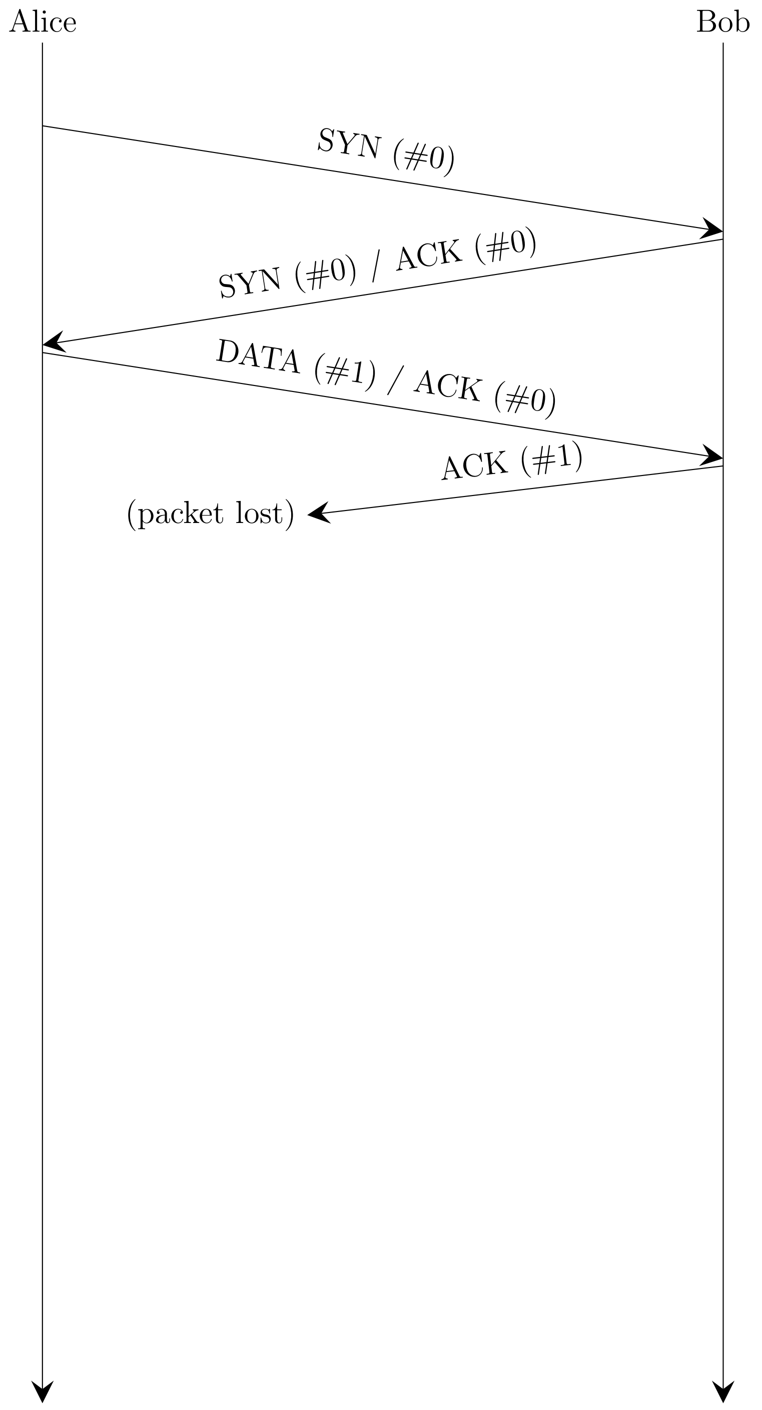
(a) [4 pts] How many bits is a page number?

(b) [4 pts] How many bits in a frame number?

(c) [4 pts] How large would a single-level page table be?

(d) [4 pts] Suppose we used an inverted page table instead. How large would the inverted page table be?

4. TCP [15 pts] Alice initiates a TCP connection to Bob and sends three bytes of data and then closes the connection. During the connection, the first ACK packet from Bob is dropped, as shown. Complete the following diagram, assuming that all data packets carry at most one byte, and assuming that no other packets are lost, until all the packets are delivered and Alice closes her connection. Be sure to indicate the start and end of any relevant timeouts, and to include the sequence and acknowledgement numbers. You may assume the initial window size is 1, and that TCP implements slow-start.



5. UDP [12 pts]

The following code fragments are used to implement a simple client-server protocol over UDP. Note that socket.recv() is a blocking call that will not return until there is some input. The client and server execute on separate hosts, and nothing can be assumed about the route between the two.

```
# interacts with a server over the (connected) UDP socket s
def client(s):
    s.send("PONG")
    response = s.recv(10)
    print response


# interacts with the client over the (connected) UDP socket s
def server(s):
    request = s.recv(10)
    s.send("PING" + request)
```

(a) [6 pts] Note that the client will either print `"PINGPONG"` or it will hang forever. Describe why.

(b) [6 pts] To avoid the hangs, the sockets are changed to use TCP. No other code is modified. Enumerate all the outputs that the client might print.

6. Raid 5 [12 pts]

   Compare the performance characteristics of an N-disk RAID 5 (striped data and parity) array to those of a single disk. Give specific numbers (e.g. it would be N times faster). Explain your answers.

   (a) [4 pts] Sequential read throughput:

   (b) [4 pts] Sequential write throughput:

   (c) [4 pts] Number of disks that can fail without losing data:

7. Unix file system [15 pts]

   List all of the blocks that are accessed when reading byte 0x12005 of the file "`/foo/bar.txt`". Assume that blocks are 0x400 bytes, that disk addresses are 4 bytes, that inodes have 10 direct blocks, one single indirect block, one double indirect block, and one triple indirect block. Make and state additional assumptions as necessary.

   1. Inode for '/' directory

   N. Data block ____ of "bar.txt"

```
## Question 2(a) source code

import db
# db.read(key)
#    reads a database file and returns the value associated with [key]
#
# db.write(key,value)
#    updates the database file so that [key] maps to [value]
#
# the db module is not thread safe; only one thread should call read or
# write at a time

class DBMonitor:
    def __init__(self):
        self.lock = Lock()

        # A dictionary of (key,value) pairs to be written
        self.write_buffer = {}
        self.write_buffer_nonempty = Condition(self.lock)

    # write the values in the write buffer to disk
    def write_to_disk(self):
        with self.lock:
            while is_empty(self.write_buffer):
                self.write_buffer_nonempty.wait()
            for key in self.write_buffer:
                db.write(key, self.write_buffer[value])
            self.write_buffer = {}

    # schedule a new (k,v) pair to be written to disk
    def put(self, key, value):
        with self.lock:
            self.write_buffer[key] = value
            self.write_buffer_nonempty.notify()

    # return the value associated with the given key
    def get(self, key):
        with self.lock:
            if key in self.write_buffer
                return self.write_buffer[key]
            else:
                return db.read(key)
```

(Code continues on next page)

```
# thread to do background writes
def background_writer(dbmon):
    while True:
        dbmon.write_to_disk()

# thread to interact with a single client
def handle_client(connection, dbmon):
    request = parse_request(connection)
    response = None
    if (request.get):
        response = dbmon.get(request.key)
    else:
        response = dbmon.put(request.key, request.value)
    send_response(connection, response)
    connection.close()

# server loop
def main(server_socket):
    dbmon = new DBMonitor()
    start_new_thread(background_writer, dbmon)
    while True:
        conn = server_socket.accept()
        start_new_thread(handle_client, conn, dbmon)
```

```
## Question 2(b) source code

import db
# db module is the same as in part (a)

class DBMonitor:
    def __init__(self):
        self.lock = Lock()

        # A dictionary of (key,value) pairs to be written
        self.write_buffer = {}
        self.write_buffer_nonempty = Condition(self.lock)

    # remove and return at least one pair from the write buffer
    def get_writes(self):
        with self.lock:
            while is_empty(self.write_buffer):
                self.write_buffer_nonempty.wait()
            result = self.write_buffer
            self.write_buffer = {}
            return result

    # schedule a new (k,v) pair to be written to disk
    def put(self, key, value):
        with self.lock:
            self.write_buffer[key] = value
            self.write_buffer_nonempty.notify()

    # return the value associated with key if it is queued for writing, or None
    def check(self, key):
        with self.lock:
            if key in self.write_buffer:
                return self.write_buffer[key]
            else:
                return None
```

(Code continues on next page)

```
db_mutex = Lock()

# thread to do background writes
def background_writer(dbmon):
    while True:
        writes = dbmon.get_writes()
        with db_mutex:
            for (key,value) in writes:
                db.write(key,value)

# thread to interact with a single client
def handle_client(connection, dbmon):
    request = parse_request(connection)
    response = None

    if (request.get):
        response = dbmon.check(request.key)
        if response is None:
            with db_mutex:
                response = db.read(key)

    else:
        response = dbmon.put(request.key, request.value)

    send_response(connection, response)
    connection.close()

# server loop
def main(server_socket):
    dbmon = new DBMonitor()
    start_new_thread(background_writer, dbmon)
    while True:
        conn = server_socket.accept()
        start_new_thread(handle_client, conn, dbmon)
```