Full Name: _____

Andrew Id: _____

CMU 15-418/618 Practice Exercise 4

Problem 1: Scheduling Requests

Immediately after announcing THE DEAL, Prof. Kayvon realizes he has a throughput problem. Students are arriving to take the deal at a **rate of one per minute**, but Kayvon can only administer deals at a **rate of one per ten minutes**. The line is growing!!!

A. (2 pts) Assume the line is five students deep when a new student arrives. How long will it be before the student is able to complete their DEAL exam? (Assume the line is processed in FIFO order.)

B. (3 pts) Assume Kayvon can be cloned at a cost of $1M per clone. Assuming multiple Prof. Kayvons can process different students perfectly in parallel, how much does it cost to keep the line from growing as more students arrive? (Clarification: Assume that your system consists of the original Prof. Kayvon plus any required clones.)

C. (5 pts) Recall that Exam 1 had six questions, and assume that Prof. Kayvon takes **1 minute per question** to ask about Q1-Q5, and **five minutes** to ask about Q6 (that's why it takes 10 minutes per student).

The cloning process can also produce modified versions of Kayvon clones. Prof. Kayvon version 2.0 (K2) is a clone specialized to ask only about Q6. K2 does not know how to ask about Q1-Q5, but K2 takes **only one minute** to administer Q6. Assuming each clone still costs $1M, please describe how many original Kayvon clones from part B (K1's) and modified clones (K2's) should be purchased to keep the line from growing at minimal cost. **Also precisely describe how you will schedule work to Prof. Kayvon plus his clones.** Please assume that questions must be asked in exam order (Q1-Q5 first, Q6 last) to each student and that there is no cost to passing a student between the clones.

D. (5 pts) [This question is unrelated to the previous ones.] You and your business partner buy a **single-core computer** to host your startup's web site. Your site receives exactly one type of request: it involves a disk read with latency 900 ms (during which the processor is idle, and waits) followed by 100 ms of processing. Thus, its response latency is 1 sec.

As your business grows you start receiving requests at steady rate of one request every 100 ms. Your partner claims, "Let's buy nine more machines so we can keep up!" You claim, "Wait a minute, we can save money and handle this load without all those machines!" Who is right? How many machines do you actually need to service 10 requests per second, and how would you architect your web server to achieve this throughput? **Assume that disk I/O bandwidth is infinite, multiple I/O operations can be outstanding at once, and that all requests are unique and so optimizations like caching are not possible.**

Problem 2: Implementing Cache Coherence

Imagine you are implementing a snooping cache controller for a cache-coherent multi-core system with an **atomic bus**. Cache-coherence is implemented using an invalidation-based MESI protocol. Following the examples we discussed in class, all processors in the system have a private L1 cache.

A. (5 pts) Consider the case where processor P1 writes to address A, which triggers the following sequence of steps:

1. Processor P1 issues store instruction.
2. Cache C1 detects that A is not present in the cache. It chooses a clean line to replace.
3. Cache C1 obtains bus access and issues a BusRdX(A) bus transaction.
4. Memory responds with data for A, C1 stores the line in the cache.
5. Processor P1 performs the write, updating the contents of the line containing A in C1.
6. Processor P1 proceeds, performing more reads or writes.

What does it mean for a write to commit in a cache coherent system, and at what point in this sequence is the write by P1 said to *commit* with respect to the other processors?

Problem 3: Implementing Load Linked / Store Conditional (2 BONUS PTS)

A common set of instructions that enable atomic execution is load linked-store conditional (LL-SC). The idea is that when a processor loads from an address using a `load_linked` operation, the corresponding `store_conditional` to that address will succeed only if no other writes to that address from another processor have intervened. Note that unlike `test_and_set` or `compare_and_swap`, which are single atomic operations, load linked and store conditional are different operations and the processor may execute other instructions in between these two operations. Pseudocode for these instructions is given below.

```
int load_linked(int* addr) {
  return *addr;
}

bool store_conditional(int* addr, int new_val) {

  if (data in addr has not been changed since the corresponding load_linked) {
    *addr = new_val;
    return true;
  } else
     return false;
}
```

Given that you now have a good understanding of a basic implementation of cache coherence, describe how you would extend the behavior of the caches to implement the load-linked and store conditional instructions. (No, you do not have to consider the case where LL-SC is used in a nested fashion on the same address.) Also, for fun, you could implement a basic spin-lock using LL-SC.