15-440 Distributed Systems
Homework 2

Due: November 10, 11:59:59 PM

Name:
Andrew: ID

November 1, 2010

1. Fix the code below so that the numbers print in order, using mutexes and conditional variables.

```
#include <pthread.h>
#include <stdio.h>

void* thread_one (void* args)
{
  printf("1");
  printf("5");
  return NULL;
}

void* thread_two (void* args)
{
  printf("2");
  printf("4");
  return NULL;
}

int main()
{
  pthread_t tid_one;
  pthread_t tid_two;
  pthread_create(&tid_one, NULL, thread_one, NULL);
  pthread_create(&tid_two, NULL, thread_two, NULL);
  printf("3");
  pthread_join(tid_one, NULL);
  pthread_join(tid_two, NULL);
  printf("6\n");
}
```

2. In this problem you're going to compare the performance of disk writes both with and without write-ahead logging (WAL). For all of these sub-problems assume that the disk is the performance bottleneck, that the disk has the following performance parameters, and that all specifications are given in SI units:

Capacity	1000 GB
Seek latency	5 ms
Rotational delay	2 ms
Transfer speed	100 MB/s

Assume that a disk address consists of a 48-bit page address plus a 16-bit offset specifying the location within the disk page. All log records contain the 64-bit disk address for the transaction's previous log entry, a 32-bit transaction ID, and an 8-bit record type (104 bits total). The log supports the `BEGIN`, `COMMIT`, `ROLLBACK`, and byte-level `UPDATE` record types as described in class, as well as a new `WRITE` record described below.

The `BEGIN`, `COMMIT`, and `ROLLBACK` records contain just the 104-bit header described above. The `UPDATE` record contains the 104-bit header plus the disk address being updated (64 bits), the length in bytes of the update (16 bits), plus both the old and new values of the location being updated (184 bits total, plus twice the length of the update to store the old and new values).

The `WRITE` record is used for a disk update that does not need to be undone if a transaction fails to commit. (For instance, a write to an otherwise unused block does not need to be erased if the transaction rolls back.) It is the same as the `UPDATE` record except that it does not contain the old value of the location being updated (184 bits total, plus the length of the update to store the new value).

Finally, assume that the operating system implements an in-memory page cache for disk pages. When a transaction commits, *some* persistent write must occur – either to the log (if logging is used) or to the disk page itself (if logging is not used). The system can buffer some disk pages in the page cache and write just the in-memory pages if logging is being used.

(a) Suppose you copy a large, single 4 GB file from a friend's computer, writing the file to your single local disk. Without logging, how long will this take? Clearly state any assumptions you make.

(b) How long will writing the same 4 GB file take on your computer, using WAL?

(c) Suppose that instead of a single 4 GB write, you overwrite the contents of a single 4000-byte file 1 million times. How long will this take without logging? (Hint: When overwriting the same file repeatedly, what causes the delay between disk writes?)

(d) How long would part (c) take, with WAL? (As always, state any assumptions you make.)

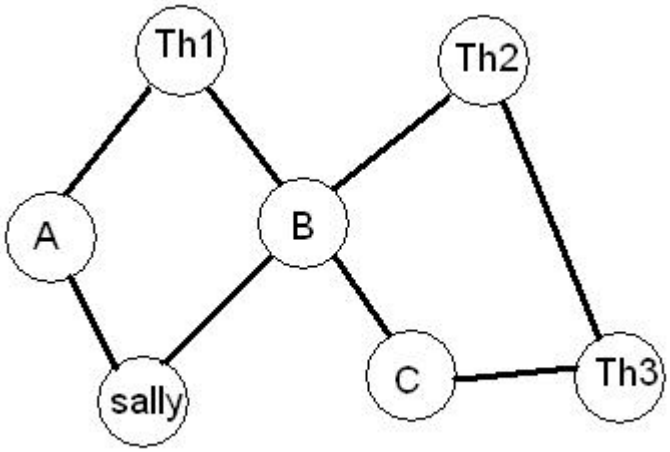
An aside: Many journaling filesystems take an intermediate approach, using WAL only for the filesystem metadata (such as the directory structure, free block list, etc) and not journaling the file data itself. This allows the filesystem to guarantee the ACID properties for some operations but not others. For instance, with metadata journaling, a file move operation will always result in the file existing at exactly one of its new or old locations, even if a power failure occurs. Editing an existing file, however, could result in a non-sensical mix of new and old data, because the file edits themselves would not be journaled.

3. Sally has a very very very large map. She's really excited about finding all the movie theaters on the map, and how long it would take to get to each one (she collects movie tickets from various theaters, it's quite an impressive collection). Sally will need your help to find the shortest path from her house to every movie theater using MapReduce.

   Assume the following:

   - There are arbitrary points on the graph that represent intersections, known as A, B, C, etc.
   - All edges on the graph are undirected.

   Here is an example of small map:



   Clearly, on a graph of smaller size, Sally can use Dijkstra's to find the information she needs, but because this graph is so large, she wants to use MapReduce, where the MapReduce will effectively be another graph traversal algorithm.

   (a) Suppose that your input for the problem looks like this:

```
Sally A 3
B Th2 4
C B 1
...
```

   Set up a possible solution to this problem. Tell us how many MapReduce phases there would be, and what each Mapper and each Reducer would do. The same phase can be run more than once, so make sure you explain how and why. Give examples of input and output for each map and reduce of a phase.

   The general format can be:
   General idea:

   Phase 1: Map:
   Phase 1: Reduce:
   Run Phase 1 X times.

4. If the real-time clocks in a group of workstations can drift 15 seconds max per day, how frequently should the clocks be synchronized for each approach to keep them within 1 second of each other using a) Christian's Algorithm and b) The Berkely Algorithm?

5. You have set up a fault-tolerant banking service. Based upon an examination of other systems, you've decided that the best way to do so is to use Paxos to replicate log entries across three servers, and let one of your employees handle the issue of recovering from a failure using the log.

   The state on the replicas consists of a list of all bank account mutation operations that have been made, each with a unique request ID to prevent retransmitted requests, listed in the order they were committed.

   Assume that the replicas execute Paxos for every operation.[1] Each value that the servers agree on looks like "account action 555 transfers $1,000,000 from Mark Stehlik to David Andersen". When a server receives a request, it looks at its state to find the next unused action number, and uses Paxos to propose that value for the number to use.

   The three servers are S1, S2, and S3.

   At the same time:

   S1 receives a request to withdraw $500 from A. Carnegie.

   - S1 picks proposal number 101 (the n in Paxos)

   - S2 receives a request to transfer $500 from A. Carnegie to A. Mellon.
     - S2 picks proposal number 102

   Both servers lok at their lists of applied account actions and decide that the next action number is 15. So both start executing Paxos as a leader for action 15.

   Each sequence below shows one initial sequence of messages of a possible execution of Paxos when both S1 and S2 are acting as the leader. Each message shown is received successfully. The messages are sent one by one in the indicated order. No other messages are sent until after the last message shown is received.

   Answer three questions about the final outcomes that could result from each of these sequences:

   (a) Is it possible for the servers to agree on the withdrawal as entry 15?

   - Is it possible for the servers to agree on the transfer as entry 15?
   - For each of these outcomes, explain how it either could occur or how Paxos prevents it.

   To be clear on the message terminology: The PREPARE message is the leader election message. RESPONSE is the response to the prepare message. ACCEPT says "you've agreed that I'm the leader, now take a value."

   Sequence 1:

```
S1 -> S1 PREPARE(101)
S1 -> S1 RESPONSE(nil, nil)

S1 -> S2 PREPARE(101)
S2 -> S1 RESPONSE(nil, nil)

S1 -> S3 PREPARE(101)
S3 -> S1 RESPONSE(nil, nil)

S2 -> S1 PREPARE(102)
S2 -> S2 PREPARE(102)
S2 -> S3 PREPARE(102)
... the rest of the Paxos messages.
```

[1] In practice, most systems use Paxos to elect a primary and let it have a lease on the operations for a while, but that adds complexity to the homework problem.

Sequence 2:

```
S1 -> S1 PREPARE(101)
S1 -> S1 RESPONSE(nil, nil)

S1 -> S2 PREPARE(101)
S2 -> S1 RESPONSE(nil, nil)

S1 -> S3 PREPARE(101)
S3 -> S1 RESPONSE(nil, nil)

S1 -> S3 ACCEPT(101, ‘‘withdraw...’’)

S2 -> S1 PREPARE(102)
S2 -> S2 PREPARE(102)
S2 -> S3 PREPARE(102)
... the rest of the Paxos messages.
```

Sequence 3:

```
S1 -> S1 PREPARE(101)
S1 -> S1 RESPONSE(nil, nil)

S1 -> S2 PREPARE(101)
S2 -> S1 RESPONSE(nil, nil)

S1 -> S3 PREPARE(101)
S3 -> S1 RESPONSE(nil, nil)

S1 -> S3 ACCEPT(101, ‘‘withdraw...’’)
```

```
S1 -> S1 ACCEPT(101, ‘‘withdraw..’’)
S2 -> S1 PREPARE(102)
S2 -> S2 PREPARE(102)
S2 -> S3 PREPARE(102)
... the rest of the Paxos messages.
```

(b) Suppose one of the servers received an ACCEPT for a particular instance of Paxos (remember, each "instance" agrees on a value for a particular account event), but it never heard back about what the final outcome was. What steps should the server take to figure out whether agreement was reached and what the agreed-upon value was? Explain why your procedure is correct even if there are still active leaders executing this instance of Paxos.