1	2	3	4	5	6	7	8	Total
/8	/8	/8	/8	/8	/8	/8	/8	/60

LUMS School of Science and Engineering
Department of Computer Science

Final Exam – Spring 2013

CS 570 — Advanced Operating Systems

Roll no: _____

Instructions.

- YOU CAN CROSS OUT 4 POINTS WORTH OF QUESTIONS. IF YOU DO NOT, LAST 4 POINTS WORTH OF QUESTIONS WOULD NOT BE CHECKED.
- You have 3 hours to complete the exam. The maximum possible score is 60.
- The exam consists of 8 questions and 9 printed pages.
- It is an open book, open notes exam.
- If there is any confusion, write down your assumptions and proceed to answer the question.
- In questions where an implementation is required, use the language of your choice. Do not worry about access modifiers (public, private). Assume *simple* helper functions. Assume member method names of collection classes (List, Stack, etc.).
- Manage your time. You should not be spending more than 3 minutes for one mark to finish on time.

1. The Google file system

8 pts

```
class ChunkServer {
  class Chunk {
    int _id;
    File _datafile;
  } chunkList[];
  int maxserialnumber;
  List<Pair<int,string>> LRUbuffer;
  int distance(ChunkServer c); // returns network distance
  void data(int id, string data, List<ChunkServer> forwardTo);
  bool primaryWrite(int id, List<ChunkServer> secondaries);
  bool secondaryWrite(int id, int serialnumber);
}
```

Let the above be partial definitions of a GFS ChunkServer. Note that id is passed from the client to identify the data while serialnumber should be assigned by ChunkServer to order writes.

(a) (4 pts) Implement ChunkServer.data which buffers data and forwards data to secondaries optimizing network performance.

Solution:

```
LRUBuffer.add(new Pair(id, data));
if (!forwardTo.isEmpty()) {
  auto closestServer = forwardTo.begin();
  for (auto &server: forwardTo)
    if (distance(server) < distance(closestServer))
      closestServer = server;
  forwardTo.remove(closestServer);
  closestServer->data(id, data, forwardTo);
}
```

(b) (4 pts) Implement ChunkServer.primaryWrite which is passed the id of previously sent data and returns true if and only if it is successfully written on all replicas.

Solution:

```
for(auto &pair: LRUBuffer) {
  if(pair.first()==id) {
    int serialnumber = maxserialnumber++;
    local_write(data); // assumed function
    LRUBuffer.remove(pair);
    for(auto &secondary: secondaries) {
      if(!secondary->secondarywrite(id, serialnumber))
        return false;
    return true;
  }
}
return false;
```

8 pts

2. Making geo-replicated systems fast as possible, consistent when necessary

(a) (2 pts) Consider a course registration system where each course has a fixed capacity and a list of registered and waitlisted students. If add and drop are blue operations, describe a situation where the class capacity requirement can be violated.

Solution: When capacity is $n - 1$ and two add operations hit two *different* servers, the capacity requirement will be violated.

(b) (2 pts) In the above example, divide adding a course into two shadow operations and tell their appropriate colors.

Solution: AddFail (Blue) can check capacity and add to waitlist if class full while AddAck (Red) will check capacity again and register if class not full and waitlist if full.

(c) (2 pts) An eventually consistent system is periodically sampled for a very long period of time and we never find the system to be consistent. Why is that?

Solution: If new operations keep arriving, the system may never get consistent. But older operations will keep getting synchronized across servers.

(d) (2 pts) What does it mean to say that an eventually consistent system does not have stable histories?

Solution: It means operations were played in different order on different servers but when eventually making it consistent, the operations are un-done and re-done in a different order on some servers. So the history is re-written or is "not stable".

3. MapReduce: simplified data processing on large clusters

8 pts

When you visit a profile on Facebook, you see a list of mutual friends. It would be wasteful to calculate this list in real-time. We would like to write a map-reduce job (in language of your choice) that takes input data in the form (friend,list_of_friends) i.e. the key is friend-id while value is list of friend-ids and outputs (pair_of_friend1_friend2, list_of_mutual_friends).

(a) (4 pts) Implement map

Solution:

```
void map (Friend friend, List<friend> friends) {
  for(auto &f, friends)
    emit (new Pair(min(f,friend), max(f,friend)), friends);
}
```

(b) (4 pts) Implement reduce

Solution:

```
void reduce(Pair<Friend,Friend> pair_of_friend1_friend2,
    List<List<friends>> friendLists) {
  List<friend> list_of_mutual_friends;
  set_intersection(friendLists[0].begin(), friendLists[0].end(),
      friendLists[1].begin(), friendLists[1].end(),
      list_of_mutual_friends.begin());
  emit(pair_of_friend1_friend2, list_of_mutual_friends);
}
```

4. Bigtable: A Distributed Storage System for Structured Data

8 pts

```
class SSTable {
  int redo-point;
  void add(string key, string value);
  string readLast(string key);
}
class GFSFile {
  int lastRedoPoint;
  int append(string key, string value); // returns record number aka redoPoint
  Pair<string,string> read(int redoPoint);
}
class TabletServer {
  GFSFile _tabletLog; // most up-to-date log
  SSTable _sstables[]; // _sstables[0] is newest
  Dictionary<string,string> _memTable; // quick lookup for reading
  int _memTableThreshold; // _memTable should not grow bigger than this

  void write(string key, string value);
  void recover(GFSFile tabletLog, SSTable sstables); // tables ordered by time
}
```

Implement the following functions. You can use any language (C++, Java, Python). Ignore timestamps and assume tablet servers store key value pairs i.e. you do not worry about the mapping of row, columnfamily, column, timestamp, and value to a key value pair. You can assume *simple* helper functions.

(a) (4 pts) TabletServer.write

Solution:

```
_tabletLog.append(key, value);
_memTable[key] = value;
if (_memTable.size() == _memTableThreshold) {
  SSTable sstable = new SSTable(_tableLog.lastRedoPoint);
  for(auto &pair: _memTable)
    sstable.add(pair.first(), pair.second());
  _sstables.push_front(sstable);
  _memTable.clear();
}
```

(b) (4 pts) TabletServer.recover

Solution:

```
_tabletLog = tabletLog;
_sstables = sstables;
_memTable.clear();
for(int i=_sstables[0].redo_point;
    i<_tabletLog.lastRedoPoint; ++i)
  _memTable.insert(_tabletLog.read(i));
```

5. Practical, transparent operating system support for superpages

8 pts

(a) (2 pts) Under what circumstances a write to a superpage would cause a demotion?

Solution: If superpage is writable but clean (dirty flag off) then a write causes demotion so dirty pages can be tracked at the base page granularity.

(b) (2 pts) Under what circumstances a write to a superpage would cause a promotion?

Solution: If superpage is writable and all except one page are dirty and this write makes the last page dirty as well, then a promotion is done since the dirty flag can be maintained at the superpage granularity.

(c) (4 pts) Given the following definition of a PopulationMap, implement a maxReservable function that finds the largest page size reservable containing the startingAddress passed to this function. It should return the page size reserable.

```
class PopulationMap {
  class Node {
    int somepop, fullpop;
    int pagesize;
    bool isLeaf;
    Node children[];
  }
  Node root;
  int startingAddress; // start addr of range rep. by this population map
  int maxReservable(int startingAddress); // returns largest reservation possible
}
```

Solution:

```
Node current = root;
int offset = startingAddress - this->startingAddress;
while (!root.isLeaf)
  if (!somepop)
    return current->pagesize;
  int nextpagesize = current->children[0].pagesize;
  int index = offset / nextpagesize;
  offset = offset % nextpagesize;
  current = current->children[index];
}
return 0;
```

6. Virtual memory primitives for user programs

8 pts

(a) (2 pts) In a shared virtual memory system, a page fault comes because a process wants to write to a page, and is forwarded to the user-space page fault handler. List the steps taken by the user-space handler.

Solution: Retrieve page from another processor or disk. Invalidate copies at other processors. Mark page as writable. Resume.

(b) (2 pts) How does concurrent garbage collection allow the collector to perform garbage collection on a page while prohibiting concurrent access from mutators.

Solution: By mapping the same pages at two different virtual addresses with different permissions.

(c) (2 pts) In incremental checkpointing, the process is interrupted for some setup work and then resumes and works concurrently with checkpointing. What happens if the process tries to modify data that is not yet saved in the checkpoint?

Solution: Page fault comes. Checkpointing thread copies the relevant page. Change permissions to make page writable. Resume.

(d) (2 pts) Why shouldn't the TLB be flushed when a read-only page is made read-write? What happens if a write request comes? Would that be denied?

Solution: A harmless TLB miss occurrs. Page table is accessed for fresh information. OS is not even involved.

7. Exokernel: an operating system architecture for application-level resource management

8 pts

(a) (2 pts) List two key differences between an exokernel and a microkernel.

Solution: Exokernel is flexible in allowing applications to make policy decisions while microkernels do not. Exokernels for the most part do not depend on trusted user-space programs whereas microkernels do. Many other correct answers as well.

(b) (2 pts) How is visible revocation different from revocation in normal operating systems. Explain with an example of processor time slice.

Solution: Visible revocation informs the process before revocation whereas normal OS preempts resources without informing. For processor time slice, visible revocation asks the process that its time slice is going to end so it should save necessary state whereas invisible revocation saves *all* state without informing the process.

(c) (2 pts) What does it mean to say that time slices can be allocated in a manner similar to physical memory?

Solution: Like memory frames, future time slices can be viewed as a linear vector where different process try to *allocate* different slices of the vector. Like memory, the OS only ensures the slices are fairly divided by the process can opt for contiguous or well distributed slices.

(d) (2 pts) Since Aegis does not implement demand paging, how can we load a program in memory thats larger than available memory?

Solution: Exokernels expose sufficient information for the library OS to implement demand paging in user space.

8. Xen and the art of virtualization

8 pts

(a) (2 pts) How are shadow page tables used in full virtualization?

Solution: Shadow page tables map from application's address space to guest OS address space and is maintained by guest OS. All changes to this area are trapped by host OS (using protection bits) and the changes are reflected in machine page tables that map application's address space to machine's (i.e. host OS's) address space.

(b) (2 pts) How are machine page tables protected from guests in Xen's para-virtualization?

Solution: Guest OS's have to make hypercalls to update the machine page table. During the hypercall, Xen checks if the update is valid.

(c) (2 pts) List one similarity and one difference between Xen and an exokernel.

Solution: They are similar in that both implement most OS functions in user space. They are different as exokernels target to perform better than a normal OS whereas Xen is necessarily slower than a normal OS. Many other correct answers as well.

(d) (2 pts) Whats the difference between a hypercall and syscall?

Solution: Conceptually they are both calls from less priviledged to more priviledged space. Syscall is when a user program calls in the OS. Hypercall is when a guest OS calls into the virtual machine manager or host OS.

Good Luck!