

# CS138 Homework Assignment 1

*Due March 2, 2010*

1. Suppose we have two disjoint Chord rings, labeled A and B. Assuming that one node in ring B knows the address of a node in ring A, give an algorithm for combining the two rings into a single well formed Chord ring. You may assume that no nodes are added, fail, or are removed while the combining is taking place. Give a good argument for why your algorithm works. (Hint: use the stabilization procedure given in slide II-30.)
2. Suppose a node is to be deleted in Tapestry (it didn't just temporarily fail — it was sold). Explain what must be done to remove it completely, without losing any important information or data. Note that some information about nodes being deleted can be safely ignored — be sure to indicate what this information is and why it can be ignored.
3. The textbook, starting on page 195, discusses how distributed garbage collection is done in RMI. However, it leaves out some important details. These details are easily obtained from the original papers on garbage collection in RMI, so please do not look at these papers. This problem is not difficult and it's much more useful to yourself for you to figure this out on your own than to look up the answer (though feel free to read the explanation in the textbook!).

The basic idea is that garbage collection is done with the aid of reference counts (let's not worry about the obvious problem of cycles). Each client process that has at least one reference to a remote object notifies the process (which is probably on a different machine) in which the actual object resides that it has a reference. When the client process has no more references (they've been deleted), it notifies the owning process to remove its reference. The owning process deletes the object when there are no more references to it.

The operations to add references and remove references are idempotent. Offhand, this seems difficult to accomplish when references are implemented using reference counts; so, rather than using actual reference counts, the server maintains a set of all its clients. Adding a reference simply adds the client to the set, removing the reference removes the client from the set.

- a. The add-reference and remove-reference operations are implemented by sending messages from the client to the server. The textbook mentions that there's a potential problem when one client does an add-reference at about the same time that another does a remove-reference — if the remove-reference arrives at the server first, and it removes the only reference, the object might be deleted before the add reference arrives. Describe a plausible scenario in which this might happen.
- b. The textbook's explanation of how the scenario of part a is handled doesn't make any sense unless client A is also the server S. Explain a simple approach for avoiding the problem. (Hint: this really is simple.)
- c. The textbook mentions that if an add-reference request results in an exception, meaning the request was acted upon either once or not at all on the server, the client follows up with a remove-reference request. Explain the rationale for doing this. (Hint: sending the remove-reference request definitely does not handle all possible problems.)
- d. Suppose the exception mentioned in part b was generated because of a network failure. The server is not down, but cannot be reached from the client. The network connection might be restored at any moment, but then again it might be down for hours. How can we ensure correct operation of both client and server? (Hint: consider and describe the use of leases by which remote references time-out after a certain period of time.)