CSci551 Fall 2013 Project B

Project B Assigned: Wed. 2013-10-03.

Project B Due: **noon, Sat. 2013-11-02** (not a class day). (This deadline applies to both Triad and Research track.) (*Correction 2013-10-04:* the above date is the due date; a previous version listed it incorrectly as 2013-10-30.)

You are welcome to discuss your project with other students at the conceptual level. Any cases of plagiarism will result in an F for the entire course. **If you have any questions about policies about academic integrity, please talk to the professor.**

Changes: 2013-10-02: none yet.

2013-10-04: fixed mistaken reference to "closest-q", too-early due date. Normalized Triad message numbers to count from 1.

2013-10-08: addition to client bootstrap about informing clients about the stage number, and two small bugs fixed.

2013-10-14: bug fixed in store command, and some clarifications on existing sample output.

2013-10-22: added clarification that each line ends with a newline character.

2013-10-25: added clarification about what is expected for header files.

2013-10-29: "update-r" messages should be logged as well; log messages have no commas; message code only occupies one byte in raw hex output.

2013-10-30: message code can occupy 1 or 4 bytes.

1 Overview

1.1 The Triad Path Project

The purpose of the CSci551 project is to get some hands on experience building a substantial distributed application running on a multi-process computing infrastructure, then to use it to study networking. It also has a secondary purpose of implementing a program using network programming (sockets) and processes (fork, in stage 1) and Unix development tools (make).

You may reuse algorithms from textbooks. You may possibly reuse functions from libraries, but if you're using anything other than the C or C++ standard library (libc, STL), or libraries mentioned on the TA's web page, you *must* check with the TA and the professor and identify it in your write-up. You need to check allowed libraries in Project Information on the class moodle. Otherwise, each student is expected to write *all* of his or her code independently. All programs will be subject to automated checking for similarity.

Please note: you should expect to spend a significant amount of time on the class projects this semester, probably at least 20 hours or more when they're all put together (or that much per project if you're new to C or C++ and network programming). Please plan accordingly. If you leave all the work until the weekend before it is due, you are unlikely to have a

successful outcome. That said, Project A is a "warm-up" project. Projects B and C will be much larger.

For the course, you will do three separate but related projects. First (Project A), you need to demonstrate that you can read the config file, do basic process control, and submit a complete assignment. (Project A doesn't really evaluate anything advanced, but it should confirm that everyone is on the same page and is comfortable with our software environment.) Project A has an early due date. Project A should be relatively short for students used to working on Linux (or Solaris or Unix) *(Clarification 2013-10-01)*; if not, it should help get you up to speed.

In Project B you will use this facility to implement Triad, a simplified version of a Chord-like Distributed Hash Table as described in the paper by Stoica et al. (see [Stoica00a] in the class syllabus). It will probably be due just after the midterm. Project B will be *much* larger than Project A; you should plan your time accordingly. Project B will be assigned later and may overlap partially with Project A.

Project C will involve extending your DHT to consider additional cases and perhaps apply it to an application. It will probably be due the last week of class.

Each project will build on the previous one.

Project C will be assigned later. It will be smaller than Project B but bigger than Project A. It will build on Project B. We will offer a the TA's implementation of Project B for students who wish to use it instead of their own Project B implementation, but we do not promise to help you understand it.

Triad Projects are *individual* projects (not groups).

1.2 Research Path Projects

This year we are have an alternate *research path* for CSci551 projects. Students will conduct original research on topics related to the course material, helping to further our understanding or develop new approaches for open problems–at least, as much as one can within a semester! For these projects, you can use the language and libraries of your choice, subject to approval from your mentor (more on mentors below). As usual, you must properly credit and cite any code, text, or approaches that you borrow from others. Using and extending the research of others is a crucial part of research, but requires proper attribution.

We will have more details about this as the semester proceeds.

Briefly, *all* students are required to do the Triad Project A. If you're interested in doing a research project, then, in addition to doing Triad Project A, you will also need to propose a research project as described below in Section 3.

For Project B, students will do *either* Triad or Research.

Students who do Triad Project B will also do Triad Project C.

Students who do Research Project B will have the option of continuing on Research Project C (if their research is going well), or changing to Triad Project C (if they prefer, or if their research is not going well). We will provide the TA's implementation of Project B so students who do Research Project B will not have to re-implement Triad Project B.

We will have a poster session in the last week of class where students can present their research path results. This year, this poster session will be part of the PhD student lunch. We encourage all students–not just research path students–to attend.

Research Projects in the Friday section are *individual* projects (not groups), but there may be some linked projects where two people work on different parts of the same problem. (Linked projects will share ideas and possibly some data or code or results, but each student will have a specific part they are responsible for and each student must do their own writeup.)

Research Projects in the Tuesday/Thursday section will allow two-person groups projects or individual projects.

Put a different way, Figure 1 shows your options.

2 Triad Project A: Getting Started

(We don't reproduce Project A here, but you may want to refer to it for background.)

3 Research Project A: Getting Started

(See the original project A assignment for details about Research Project A.)

4 Triad Project B

Now that Project A has demonstrated running basic processes and communication, Project B gets down to the business of implementing *Triad*, a simple Distributed Hash Table modeled after Chord.

4.1 Concepts

Please refer to [Stoica00a] from class for a detailed description of Chord and the Chord algorithms.

Triad is like Chord, but slightly simpler. Like Chord, nodes form themselves into a ring and store keys. Each node and key has an id; a node is responsible for all keys that have its id value from just after the prior key in the ring through its id. In the worst case, nodes find data by linear search through the ring.

Like Chord, Triad can use finger tables to speed search, providing $O(\lg n)$ search time in the best case. Unlike Chord, the first stages of Triad will be done without such acceleration.

Both Chord and Triad use hash values to map data into the storage ring, but Chord uses the full 160-bit SHA-1 value. Triad will use a 32-bit value using a hash function provided by the TA (it will actually be just the first 32-bits of the SHA-1). Chord also talks about hash values abstractly. For Triad, we will provide string names and part of a function to compute the underlying hash value.

While both Chord and Triad map keys to a ring, in Triad we will actually store data with those keys. (We will add this feature in a later stage.)

Chord includes algorithms to deal with ring maintenance as nodes come and go, possibly without notice. In Triad we will build this capability up gradually. In early stages all nodes will join at the beginning and no nodes leave. In later stages, nodes will leave with warning

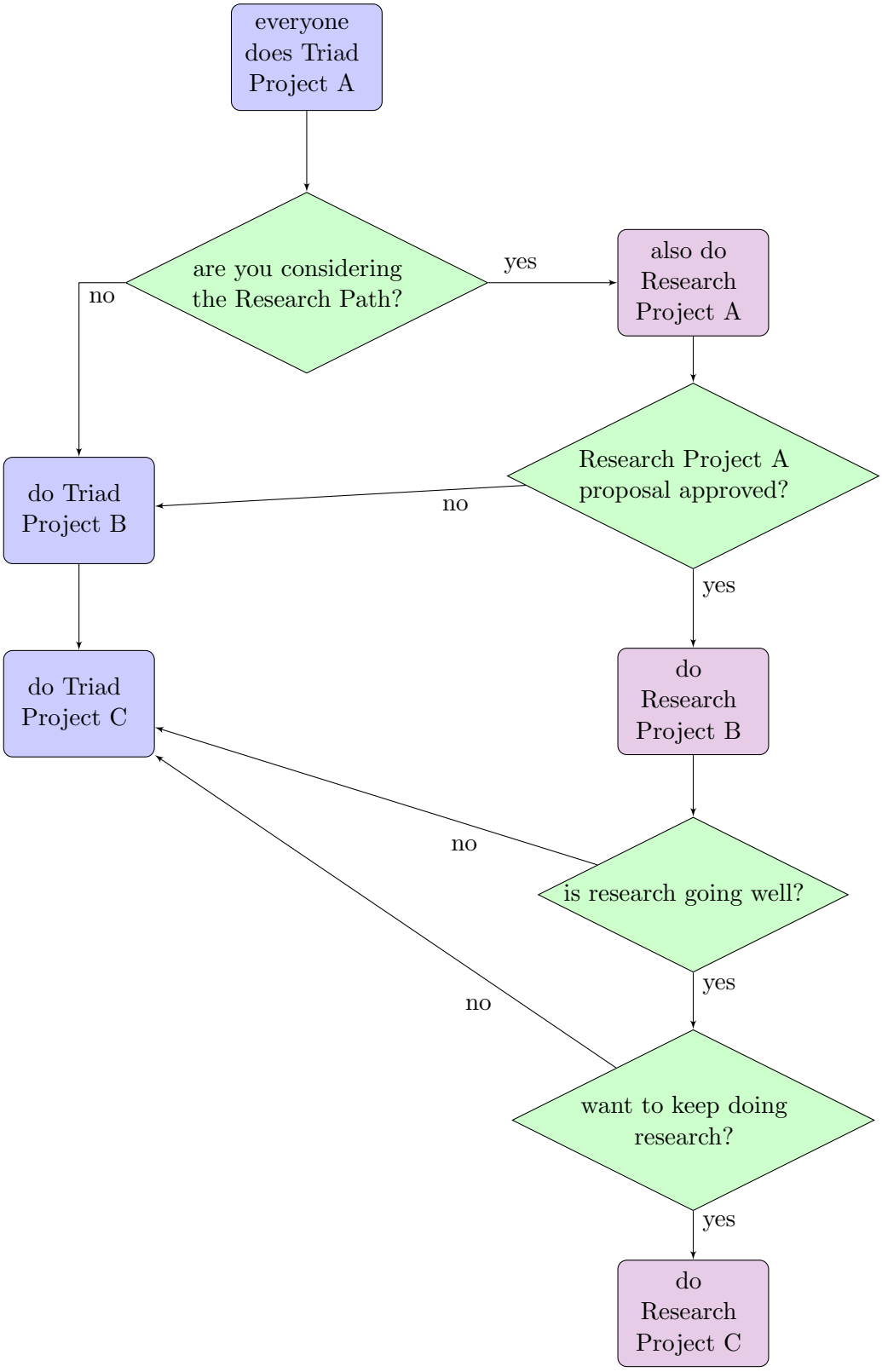

Figure 1: Options for triad/research path projects.

to their neighbors, then join after some keys have been stored, and eventually we will support nodes coming and going without warning.

4.2 Stage 2: Booting Triad

For stage 2 we will bring up the smallest possible correct Triad: you will boot a network of several nodes that will form a Triad network with successor pointers. This stage is about correctness, not performance, so we will not yet implement a finger table. Instead, each node will link directly to its immediate neighbors. In addition, to simplify this stage, nodes will (or should!) never fail and leave the network, so you only need to deal with network growth, not change.

To be able to exercise your Triad network, the configuration file will include a script to create nodes and make them perform operations.

As with stage 1, we expect the manager to read the configuration file. Unlike stage 1, here we will create clients dynamically.

You should expect a "stage 2" and "nonce N" lines in the configuration file (as well as comments). The nonce will be used in computing hashes. It is guaranteed to be an unsigned integer less than 2^{32}. The manager should pass it to all clients, through the TCP socket as in stage 1.

Creating clients: After the nonce line there will be several `start_client S` lines. Each start_client line will specify a name of the client S (some string of 80 characters or less). The manager should create the client by forking and setting up a TCP connection as in stage 1. The manager will write four lines of text to the client to get it going: first the nonce N, then the client's name S, a port number FP (described below), and a second name FS all in ASCII terminated by a newline. (The TCP connection should stay open; it will be used to send later commands from the manager to the client.)

Addition 2013-10-08: The clients will need to know what stage is running. Yes, you may pass the stage number with the server port OR you may extend the TCP information from the manager to include the stage number. You should document your choice and why you make it in README question 2.1. *(Thanks to Juan Fasola for pointing out this omission.)*

For the first client created, the port number FP will be zero, and the second name FS will be the same as the first name S. For all other clients, the port number FP will be the UDP port of the first created client, and the second name FS will be the name of the first created client. Subsequent clients will use FP and FS to connect themselves to the ring.

Each client should compute its Triad identity and enter the Triad ring. To compute the triad identity, it should take the nonce as a 4-byte, network-byte-order value, concatenate the client's name S, not including the newline nor the `'\0'` end-of-string terminator, and pass that to class-provided hash-function that will return a 32-bit unsigned integer. While we provide you the hash function with the signature `unsigned int projb_hash(unsigned char *buffer, int buffer_length)`, you must assemble the buffer that is hashed as described above. (The TA will provide this code on the project moodle page.) You may check your job in that you should see these mappings from (nonce, name) to hash: (1234, foo) to 0xa9367d92, (12346, foo) to 0xcba8f538, (1234, bar) to 0xf970bbc8.

To enter the Triad ring, a client should dynamically allocate a UDP port and start receiving Triad messages on that port. It should then determine and set up its successor and predecessor links. If it is the first node, it can do this itself (it knows it has no successor or predecessor, or it's probably cleaner to consider it to be its own predecessor and successor). If it is not the first node, it can do this by talking to the port of some other established node given to it by the manager. It will need to search around the ring to find its proper location, then tell the old successor and predecessor it belongs between them. Finally, when it's set up, it should reply to the manager on the manager-client TCP port. This reply should have two lines, each in ASCII and terminated by a newline. The first should be the modified nonce as computed in Stage 1. The second line should be its UDP port for Triad messages.

We expect the manager and your clients will never block and busy wait on I/O more than briefly. You may choose to do event-driven programming with a select loop (possibly using the class timer library), or you may use threads if you prefer. This requirement is both because it's the Right Thing To Do, and because it will be important in later stages where nodes may come and go unexpectedly. (While we expect your code to not busy-wait on other processes, the assignment is not designed to torture test concurrent programming.)

The client creation procedure is carefully constructed so that the manager knows when the client is really ready. If the manager waits to hear this reply before continuing, then there should be no race conditions where the next client needs to contact another client that is not yet initialized.

For stage 2 and some following stages, you may assume that no clients are created after the first store or search. We will relax this assumption in a later stage where nodes will come and go more frequently (and possibly unexpectedly).

Each client should create a separate log file "stage2.S.out", where S is the name of the node (as the client finds out from the manager). Just after a client has been created and joined the ring, but before it does anything else, it should log "client S created with hash SI", where S is the name and SI is the hash value it creates in hexadecimal.

Triad Messages: Once the client sends its Triad UDP port to the manager, it needs to loop waiting for both Triad message (on the UDP port) and control messages from the manager (on the TCP connection). (Hint: I recommend you use a select loop so you can multiplex these two operations.)

Clients will send each other Triad messages to manage the ring:

Messages:

1. **successor-q** (ni) ask target with identity ni what its successor is *(Clarification 2013-10-04):* all message have are now renumbered to be consistent (previosuly they started at 0 in some places and 1 elsewhere).

2. **successor-r** (ni, si, sp) reply stating that ni's successor has id si and port sp

3. **predecessor-q** (ni) ask target ni what its predecessor is

4. **predecessor-r** (ni, pi, pp) reply stating that n's predecessor has id pi and port pp

5. **stores-q** (ni, di) ask target ni what its best estimate of the node that stores di (note that its answer may not be correct, in that the reply may not actually cover di, but it is presumably an improvement

6. **stores-r** (ni, di, ri, rp, has) reply stating that ni's best estimate of the node that stores di has id ri at port rp. In addition, the *has* value is 1 if $ni = ri$ and ri has the exact id di. Otherwise, the *has* value is 0. (In other words, *has* is 1 IFF you queried the node holding the exact hash and that node has stored that data locally.)

7. **update-q** (ni, si, sp, i) request node ni update its ith finger table entry, or the predecessor entry if $i = 0$, to point to identity si at port sp. Entry $i = 1$ must be the successor, and in later stages, i can be as high as 32. Entry $i = 0$ is a special case and specifies the predecessor. For stage 2, i will always be 0 or 1 since we do not yet implement full finger tables, just successors and predecessors.

8. **update-r** (ni, R, si, sp, i) reply to an update-q message. The reply code R should be 0 if the update failed, otherwise 1. The other parameters are reflected back (in the unlikely event you have multiple outstanding update-q requests). This message allows a client to know when an update request completes, helping to avoid race conditions when there are multiple sequential updates.

9. **store-q** (ni, SL, S) store string S of length SL at node id ni. Presumably the hash of S should be in ni's currently covered portion of the ring.

10. **store-r** (ni, R, SL, S) reply from ni with reply code R about storing string S with length SL, with reply code R. R should be zero if the store failed (for example, the S doesn't belong to ni), 1 if the store succeeded, and 2 if the string was already stored at the node.

Each message should be encoded as the message code (1 for successor-q, 2 for successor-r, etc.), and each of the parameters stored as a 4-byte, network-byte-order integer. (*Clarification 2013-10-30*: For the message code you can decide whether you encode it in 1 or 4 bytes.) For store-q and store-r, the string length is given by the SL parameter and the string should follow with exactly that many additional bytes.

In the queries, passing ni is actually redundant, because the client receiving the message should know its id already. However, the receiver may wish to check that against its id in order to detect the unlikely event that the code has a bug.

The client should log all Triad messages it sends and receives to this log file in the order they are sent or received. Each log line should be the message name (listed above), the word "sent" or "received", and each parameter. Node identities should be printed in hexadecimal, and other integers in base-10. For example, a successor-r log statement might be "`successor-r sent (0xa9367d92 0xf970bbc8 5473)`" (*Clarification 2013-10-29*: log messages use spaces ONLY as parameter delimiters).

If the client's name (given in the start_client configuration line) is "logmsg", then that client must also log the raw contents all messages that it receives, after it logs their text format. These log entries should start "raw 1234abcd..." where the 1234abcd are replaced with hexadecimal values of each byte. (It should not log messages it sends.) For example, if

logmsg received the above successor-r message, it would log (*Clarification 2013-10-30*: the length of the message code hex representation depends on the number of bytes the code occupies on the wire (you can choose either 1 or 4 bytes), in the example below it occupies one byte):

"`raw:  02a9367d92f970bbc800001561`".

Client interactions: The manager will read commands from the configuration file and respond in these ways:

start_client Handled as described above

store S store a key with name S (some string) in Triad.

The manager should pass a store command to the first created client. That client should then execute the store by finding the right client in the ring (presumably by sending repeated ~~closest-q~~ stores-q (*Correction 2013-10-04*) messages), then sending a ~~store-r~~ store-q (*Correction 2013-10-14*) message.

The client executing the store command should convert the identity into a hash value with the same algorithm as node identities. (We don't actually store any value with the key, so clients will just track the presence or absence of a key.) You can assume S is at most 79 characters long.

After adding a key, the client executing the add (the first client) should log "add S with hash SH to node NI" to its log file. SH is the hash of S (and almost certainly different from the node identifier NI).

search S Find if key named S is in Triad.

Just like ~~"start_client"~~ "store S" (*Correction 2013-10-08*), but check to see if the key is present.

The client executing the ~~add (the first client)~~ search (*Correction 2013-10-08*) should log "search S to node NI, key PRESENT" if the key is found, otherwise "search S to node NI, key ABSENT".

After the first client has executed a command, it should send text text "ok" and a newline back to the manager over the client-manager TCP connection. This response will let the manager know it can proceed with the next command, avoiding race conditions.

You may assume there are fewer than 100 clients and 100 strings, although good code will not need these constants.

Sample input: Here is a sample configuration input for this stage:

```
stage 2
# you are guaranteed a nonce line before any clients are started
nonce 1234
start_client foo
# there may, of course, be comments anywhere
```

```
start_client bar
start_client baz
# you can assume for stage 2 that no clients start after the first store or search
store alpha
store beta
store gamma
search beta
search delta
```

(Note that you should also test a configuration file with a client called "logmsg".)

Sample output: File stage2.foo.out ~~(note the message contents below are currently outdated; we will update them by the second week of October)~~ (*Clarification 2013-10-29* Update replies should be logged as well.):

```
client foo created with hash 0xa9367d92
successor-q received (0xa9367d92)
successor-r sent (0xa9367d92 0xa9367d92 38708)
update-q received (0xa9367d92 0xf970bbc8 58366 0)
update-r sent (0xa9367d92 1 0xf970bbc8 58366 0)
update-q received (0xa9367d92 0xf970bbc8 58366 1)
update-r sent (0xa9367d92 1 0xf970bbc8 58366 1)

successor-q received (0xa9367d92)
successor-r sent (0xa9367d92 0xf970bbc8 58366)
update-q received (0xa9367d92 0xdabfd86c 34089 1)
update-r sent (0xa9367d92 1 0xdabfd86c 34089 1)
stores-q sent (0xdabfd86c 0xbf0a5ea2)
stores-r received (0xdabfd86c 0xbf0a5ea2 0xdabfd86c 34089 0)

store-q sent (0xdabfd86c 5 alpha)
store-r received (0xdabfd86c 1 5 alpha)
add alpha with hash 0xbf0a5ea2 to node 0xdabfd86c
add beta with hash 0x7eba5174 to node 0xa9367d92
add gamma with hash 0x3cee9126 to node 0xa9367d92

search beta to node 0xa9367d92, key PRESENT
search delta to node 0xa9367d92, key ABSENT
```

~~We will provide sample output, announced on the class moodle, in the next week.~~ (*Clarification 2013-10-14* These are valid; no additional output at this time will be on the class moodle.)

File stage2.bar.out:

```
successor-q sent (0xa9367d92)
successor-r received (0xa9367d92 0xa9367d92 38708)
```

```
update-q sent (0xa9367d92 0xf970bbc8 58366 0)
update-r received (0xa9367d92 1 0xf970bbc8 58366 0)

update-q sent (0xa9367d92 0xf970bbc8 58366 1)
update-r received (0xa9367d92 1 0xf970bbc8 58366 1)
client bar created with hash 0xf970bbc8
update-q received (0xf970bbc8 0xdabfd86c 34089 0)
update-r sent (0xf970bbc8 1 0xdabfd86c 34089 0)
```

There will also be a stage2.baz.out (sample will not be posted).

Contents of stage2.README.txt:

2.1) Have you implemented all the requirements of this stage? if not, which are missing?

2.2) Did you use any library other than those presented on the moodle page? if so, what one and to do what?

For this stage, most of the points will be evaluated to assessing if your code runs, and we will do that by feeding it several input configuration files.

4.3 Stage 3: Demonstrating Linear Search

Basic Triad has linear search.

For this stage, you will set the nonce to a value specific to you and look at search performance.

Sample input:

```
stage 3
# note, on the nonce line,
# replace dddd with the first four digits
# of your student id
nonce dddd
start_client alpha
start_client bravo
start_client charlie
start_client delta
start_client echo
start_client foxtrot
store alpha
store beta
store gamma
store delta
store epsilon
search beta
search delta
```

Sample output: Output format is exactly the same as stage 2, but with the filename changed from "stage2.S.out" to "stage3.S.out". Specific output will vary because of the use of different nonces.

Contents of stage3.README.txt:

3.1) If you use 1234 for the nonce, how many steps does it take to search for "beta"?

3.2) If you use 1234 for the nonce, how many steps does it take to search for "delta"?

3.3) What are the first four digits of your student ID?

3.4) With your student ID for a nonce, how many steps does it take to search for "beta"?

3.5) With your student ID for a nonce, how many steps does it take to search for "delta?

4.4 Stage 4: Adding Finger Tables

It's important that real Chord has $O(\lg n)$ search time, not linear. While perhaps not critical for our small network, with millions of nodes linear search would be untenable. Finger tables are the key to guaranteeing, with high probability, that we will get logarithmic search time for a large network.

This stage has the same input format as stages 2 and 3, except that the stage number changes, and all clients are expected to create and maintain finger tables that cover the entire hash space.

Note that when a node is added you may have to update several finger tables. You may do multiple updates recursively (client A updates B, and B notices it must update C and so it does so before it returns), or you may update them iteratively (client A updates B, then A determines it must update C, so A updates C directly.) ([Stoica00a] Figure 6 suggests recursive updates, but you may do either.)

Sample input: Same as stage 3, except the first line is "stage 4".

Sample output: Output filenames should be like stage 2, but with the filename changed from "stage2.S.out" to "stage4.S.out". The end result should be the same as stages 2 and 3, but you should find you do less searching.

Contents of stage4.README.txt:

4.1) Have you implemented all the requirements of this stage? if not, which are missing?

4.2) Did you use any library other than those presented on the moodle page? if so, what one and to do what?

4.3) If you use 1234 for the nonce, how many steps does it take to search for "beta"?

4.4) If you use 1234 for the nonce, how many steps does it take to search for "delta"?

4.5) With your student ID for a nonce, how many steps does it take to search for "beta"?

4.6) With your student ID for a nonce, how many steps does it take to search for "delta"?

4.5 Stage 5: Starting Dynamics with Node Departure

While setting up a Triad network takes some work, the real challenge in peer-to-peer systems like it is network *dynamics*, as nodes come and go (churn). Handling dynamics can become particularly difficult when multiple nodes are entering and leaving the network at the same time, and, in a large enough network, it is *always* the case that nodes are coming and going.

The next stage adds controlled node departure to Triad. With departures, there is no longer the guarantee from stage 2 that nodes do not change after the first search or store command. Nodes can now leave after a search, although new nodes cannot (yet) join. (In Project C we will add additional dynamics.)

This stage requires one new configuration line: "end_client S", which indicates that the client with name S should do a controlled departure. As with other manager-client commands, the client needs to send OK after it's completed this command. The manager will need to keep track of client names so it can send this message to the correct client.

Handling client departure requires four new Triad messages:

21. **leaving-q** (ni, di) tell node ni that node di is leaving the network. Node di will send this to its successor ni to allow ni to take over di's job.

    On receiving this message, ni should fetch all of di's data using the next-data-q message described below. When it has all the data, it should adjust its predecessor link to skip di, and tell it's new predecessor to update its successor. (With the update-q message where $i = 0$.) It should also update anything else that needs to change, like finger tables.

22. **leaving-r** (ni) is the reply to leaving-q, indicating all data has left the network.

23. **next-data-q** (di, id) is a request to di to give the next data item stored at that node with hash less than or equal to id. It responds with next-data-r and the data item.

24. **next-data-r** (di, qid, rid, SL, S) is the reply; it indicates that di's next data item after qid is rid with string S of length SL. If the di stores no data less than qid, it should return rid as one less than the beginning of its range and a zero-length S.

The messages next-data-q and next-data-r allow the node that is taking over to start asking the departing node for its data, then it counts backwards around the id space until it has fetch all data from that node. (Remember that all id arithmetic is modulo around the ring, so 0x0 is before 0x1, and `0xffffffff` is before 0x0.)

Sample input:

```
stage 5
nonce 1234
```

```
start_client alpha
start_client bravo
start_client charlie
start_client delta
start_client echo
start_client foxtrot
store alpha
store beta
store gamma
end_client charlie
store delta
store epsilon
search beta
search delta
```

Sample output: Again, the filename should include "stage5".

The following is *truncated* sample output for `stage5.alpha.out`. There are a lot of messages being sent and received between the two nodes, but the following keys are present (after the `search` commands):

```
 [truncated]

search beta to node 0xbf0a5ea2, key PRESENT
successor-q sent (0xd9aa956e)
successor-r received (0xd9aa956e 0xda5e7b2c 33083)
stores-q sent (0xd9aa956e 0x270056fe)
stores-r received (0xd9aa956e 0x270056fe 0xda5e7b2c 33083 0)
successor-q sent (0xda5e7b2c)
successor-r received (0xda5e7b2c 0x270056fe 42268)
stores-q sent (0x270056fe 0x270056fe)
stores-r received (0x270056fe 0x270056fe 0x270056fe 42268 1)
search delta to node 0x270056fe, key PRESENT
```

Contents of stage5.README.txt:

5.1) Have you implemented all the requirements of this stage? if not, which are missing?

5.2) Did you use any library other than those presented on the moodle page? if so, what one and to do what?

5 Research Project B

For Research Project B, you will be expected to work on your research project regularly throughout the period of the assignment. In particular, you will be responsible for the

following deliverables, as described in your project proposal submitted as part of Research Project A.

1. Regular meetings with your mentor. We expect these will be about weekly, so you should have about 4 of them (and at least 3) over this period. (You may have more if you want.)

2. Written weekly notes about your progress, sent to your mentor the day before your weekly meeting, commented on by the mentor at/after the meeting, and updated by you after the meeting. Of course, this requirement means that you are required to meet with your mentor roughly every week.

3. Code, data, and analysis results, provided when you submit this portion of the project. Additionally, you should include a README.txt file describing what you submitted.

4. A Project B report, probably 2–6 pages summarizing where you are as of the Project B deadline and what you plan for Research Project C (tentatively: noon Wed. 2013-10-30, but final date TBD). The document should be a PDF and needs to include the following numbered sections:

Section 1 an introduction, giving an overview of what you're doing, and a statement about (a) why is it interesting research (to the field)? (b) why is it interesting for you (what will you learn)? (c) what about what you're doing is new.

Section 2 what is relationship to other work. List related work, with citations in your text, and for each piece of related work identify how it is alike or different from your work.

Section 3 a discussion of your work (possibly including subsections on goal, methodology/approach, data collection, and results)

Section 4 a description of what the next steps are for Research Project C. (You must include a description of what the next steps are even if you do not plan to do Research Project C but will switch to Triad Project C—knowing what's next is part of the process of doing research!)

   i. what you would plan to do do for Research Project C. For example, if you did not yet evaluate your approach, the proposed work could describe your evaluation plan.
   ii. if you want to do Research Project C.
   iii. a checklist of specific deliverables for Project C (a) continuing to meet once a week with your mentor, (b) continued weekly notes about your progress, (c) what end results you expect to have (code or experiments), (d) that you will do a project C report

Section 5 a bibliography, with at least what you cite in related work

(We encourage you to write your document in LaTeX and build a PDF, but you can use whatever tool you prefer.)

Your Research Project A proposal will also serve as part of your grade for Research Project B. Since research is not always predictable, you will not be graded on executing exactly the plan in your Research Project A proposal, but it will serve as a guideline. However, the most important factor is to conduct quality work that adapts to what you discover about the problem as you go. Your mentor especially (as an expert on the problem), but also your TAs and professor, can help focus you on interesting directions throughout the semester.

6 Hints about Project C

Project C is still under revision, but we are expecting it will include some extensions to your Triad implementation, and probably some application using Triad.

7 Submitting and Evaluating your Triad Project

To submit each part of the assignment, put *all* the files needed (Makefile, README, all source files, and source to any libraries) in a gzip'ed tar file and upload it to the class moodle with the filename `projb.tar.gz`. *Warning:* when you upload to the moodle, please be careful in that you must both do "Upload a file" *and* do "Save changes"! When you are done you should get a message "File uploaded successfully", and you should see a list with your file there and an option to "update this file". If you do *not* see "file uploaded successfully", then you have *not* completed uploading!

We *strongly* recommend that you confirm that you have included everything needed to build your project by extracting your tarfile in a different directory and building it yourself. (It's easy to miss something if you don't check.)

It is a project requirement that your project come with a Makefile and build with just the make command. To evaluate your project we will type the following command:

```
% make
```

Structure the Makefile such that it creates an executable called `projb` (Note: *not* `a.out`.) For more information please read the make(1) man page. (Your program must run with the make provided in your VM.) We recommend you do not ~~so be careful not to~~ use any make extensions such as those in GNU make (gmake).)

We will then run the manager using a test configuration file. You can assume that the topology description will be syntactically correct. After running the program, we will grade your project based on the output files created by the manager and the clients.

It is a project requirement that your implementation be somewhat modular. This means that you should follow good programming practices—keep functions relatively short, use descriptive variable names. You must use at least one header file (*clarification 2013-10-25:* of your own creation), and multiple files for different parts of your program code. (The whole project should be broken up into *at least* two C/C++ files. If you have a good file hierarchy in mind you can break up into more files but the divisions should be

logical and not just spreading functions into many files.) Indicate in a comment at the front of each file what functions that file contains.

Computer languages other than C or C++ will be considered but *must be approved ahead of time*; please contact the professor and TA if you have an alternative preference. The deadline for approving new computer languages is *one week* before the project due date, so get requests in early. The language must support sockets and process creation. (Please ask *before* you start, we don't want you to waste your work.)

Although we provide a complete sample input file and output, final evaluation of your program will include other input sources. We therefore advise you to test your program with a range of inputs.

Although the exact output from your program may be different from the sample output we provide (due to events happening in different orders), your output should match ours in format.

It is a project requirement that your code builds without warnings (when compiled with -Wall). We will be responsible for making sure there are no warnings in class-provided code (any warnings in class-provided code are our bugs and will not count against you). To make sure that you create code which does not throw warnings you can compile with the `-Werror` flag which converts all warnings to errors. Thus, compilation only succeeds if your code is free of warnings.

It is a project requirement that your code have reasonable good style. Your code should be readable. Make sure you clean up code that you remove or comment out (especially if you commented out whole functions). Your code should use proper indentation and follow some consistent indentation style. (There are many styles, but you should pick one and be consistent.) You should include at least some documentation in the form of comments. You should pick reasonable variable and function names (not all `a1`, `a2`, and `b3`).

There are tools that help some parts of writing good code. or example, `indent` is a tool ensuring proper indentation, including support for common styles (see `http://manpages.ubuntu.com/manpages/lucid/man1/indent.1.html` for more details). Most editors, including emacs and vi can help with indenting as well. IDEs under Linux may also help; see Eclipse (you can install it with (see **yum install eclipse**).

Writing good code is both good practice, and it will make your debugging much, much easier.

8 Hints

The structure of the project is designed to help you by breaking it up into smaller chunks (compared to the size of the whole project). We strongly encourage you to follow this in your implementation, and do the stages in order, testing them as you go. In the past, some students have tried to read and implement the whole assignment all at once, almost always resulting in an unhappy result.

8.1 Common pitfalls

Please do not hardcode any directory path in your code! If you hardcode something like `/home/johnh/...` in your code to access something in your home directory and the grader cannot access these directories during grading, your code will not work (and this will be your fault)! If your code does not work, you get no credit! Instead, assume paths are given external to your program, and that you read and write files in the current directory (wherever that is).

8.2 Doing multiple things at once

Later stages of the project may require you to handle both timers and I/O at the same time.

Note that stage 1 is not complex enough to require timers. One approach would be to use threads, but most operating systems and many network applications don't actually use threads because thread overhead can be quite large (not context switch cost, but more often memory cost—most threads take at least 8–24 KB of memory, and on a machine with 1000s of active connections that adds up, and always in debugging time, in that you have to deal with synchronization and locking). Instead of threads, we strongly encourage you to use timers and event driven programming with a single thread of control. (See the talk "Why Threads Are A Bad Idea (for most purposes)" by John Ousterhout, `http://www.stanford.edu/~ouster/cgi-bin/papers/threads.pdf` for a more careful explanation of why.)

Creating a timer library from scratch is interesting, but non-trivial. We will provide a timer library that makes it easy to schedule timers in a single-threaded process. You may download this code from the TA web page. There is *no* requirement to use this code, but you may if you want. If you want to use it, download it from the class web page. There is no external documentation, but please read the comments in the `timers.hh` and look at `test-app.cc` as an example. If you do use the code, you must add it to your Makefile and you must document how you used it in your README.

8.3 Other sources of help

You should see the Unix man pages for details about socket APIs, fork, and Makefiles. Try `man foo` where foo is a function or program.

The TA can provide *some* help about Unix APIs and functionality, but it is not his job to read the manual page for you, nor is it his job to teach how to log in and use vi or emacs.

You may wish to get the book *Unix Network Programming*, Volume 1, by W. Richard Stevens, as a reference for how to use sockets and fork (it's a great book). We will *not* cover this material in class.

The README file should not just be a few sentences. You need to take some time to describe what you did and especially anything you didn't do. (Expect the grader to take off more points for things they have to figure out are broken than for known deficiencies that you document.)