CSci551 Fall 2013 Project C

Project C Assigned: Fri. 2013-11-06.

Project C Due: **11pm, Fri. 2013-12-06** (note different time of day). (For this assignment only we will allow an extra 24 hour grace period with no late penalty. If you haven't used your slip day, you may use that in addition. We encourage you to turn it in ahead of time to leave the study days for studying, but that is your choice.)

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: 2013-08-29: none yet

2013-11-27: clarification of "closest" in Stage 7.

2013-11-27: required submission for research projc are added, see Section 7.

2013-11-27: improved discussion of handling finger table updates in stage 7, including option of lazy or preemptive updates (see "Suggestions about handling finger table updates").

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. Although Project C is smaller than Project B, you should expect it will still take a significant amount of time.

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.

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.

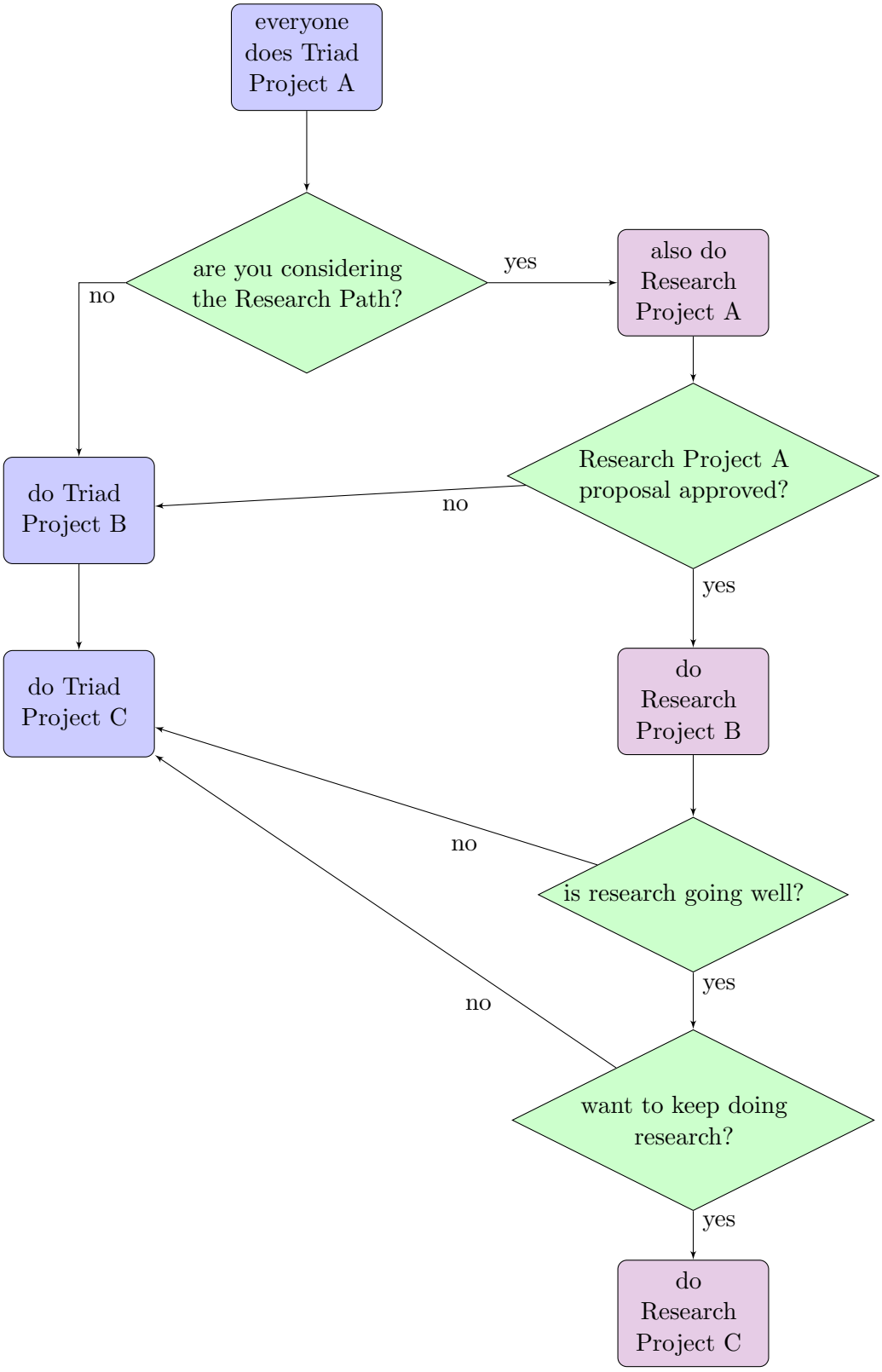

Figure 1: Options for triad/research path projects.

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

We don't reproduce it here, but you may wish to refer back to your Project B for hints and background. In fact, Project C is required to use the same messages, message formats, and logging rules as Project B.

5 Triad Project C: Extending Triad

5.1 Stage 6: Unexpected Node Departure

Stage 5 (in Project B) explored controlled node departure, as a node announced it was leaving and other nodes take over its job. In a real peer-to-peer system, nodes aren't so friendly—they go away when computers are turned off or suspended. And even if computer users tried to gracefully shut them down, they can still fail unexpectedly and reboot, so all real systems have to deal with unexpected failure.

In this phase, we'll add support for unexpected node failure. All nodes need to watch the ring and repair a single unexpected failure. To accomplish this repair, each node will need to keep both a successor pointer and a double-successor pointer. Each node will periodically poll its successor; if it doesn't get a reply or if it gets an error message, it should assume the successor failed and rebuild the ring accordingly.

Note that because data is stored on only one node, any such failure may loose data. Those are the breaks; you won't recover from them in this stage.

Each node should do stabilization, confirming its successor exists reasonably frequently, and that its successor's predecessor is itself. Reasonably frequently means every 10 seconds (or you can chose a different value if you want). You will need to implement this test to run in the background, while you're also doing other message passing. You may find the timer library is helpful to accomplish this goal. To check the predecessor of your successor, a client must use the following two messages:

31. **hello-predecessor-q** (ni) ask target ni to confirm that it is alive and to reply with its predecessor

32. **hello-predecessor-r** (ni, pi, pp) reply stating that ni is alive and has a given predecessor with id pi and port pp

Note that these are the same as the standard predecessor-q and -r messages, but they have different ID codes so we can easily tell why you're using them.

When a node sends or receives a hello-predecessor-q or hello-predecessor-r message, it should record this fact in its per-node log as `hello-predecessor-q sent (0xa9367d92)` or `hello-predecessor-q received (0xa9367d92)` or `hello-predecessor-r sent (0xa9367d92 0xf970bbc8 58336)` and similar for `hello-predecessor-q`.

When a node si sends hello-predecessor-q to its successor, there are three possible replies. (1) it gets a hello-predecessor-r reply, and the $pi = si$. All is well with the ring. It should log `hello-predecessor-r confirms my successor's predecessor is me, 0xa9367d92`.

(2) it gets a hello-predecessor-r reply, and the $pi \neq si$. The ring is broken, This case should not actually happen in the class project, because it is designed to allow the ring to always stabilize after a node change. This result may indicate a bug in your code. Regardless, in this case it should log `hello-predecessor-r reports my successor's predecessor is 0xf970bbc8, not me 0xa9367d92`.

In this case there are two sub-cases: (2a), if $pi > si$, then node si's successor is wrong. It should rebuild its successor and finger table and log `hello-predecessor-r causes repair of my links`. And then do whatever is needed to correct its successor and finger table.

(2b) if $pi < si$, then my successor doesn't know about me. It should rebuild its successor and finger table and log `hello-predecessor-r causes me to repair my successor's predecessor`. It should then cause its successor to correctly rebuild its predecessor link by sending it an update-q message where $i = 0$.

(3) it gets no reply. In this case, it should time out, determining that its successor has unexpectedly died. It should log `hello-predecessor-r non-reply: my successor is non-responsive`. It should then use its double-successor to remove its single successor from the ring. It will need to use update-q to correct its double-successor's predecessor, and correct it's successor and double successor, and confirm and correct its finger table.

Nodes also must maintain their finger tables. In Chord, nodes maintain their finger tables by verifying a random finger table entry periodically. We're going to do something slightly different: the finger tables (other than the successor and predecessor) should be maintained *lazily*. That is, once they're set up, don't do anything to adjust them, *but* verify they're correct before each use. That is to say, if your node wants to use finger table i for id id that points to ni, it should first send a stores-q(ni, id) to ni.

There are three cases with the reply: (1) it gets a stores-r reply from ni. This reply will include ri, the node that ni thinks should hold id. If $ni = ri$, the finger table is correct and we're done. In this case, your node should log `lazy finger table validation for entry 2 to 0xf970bbc8 checks`. (If ni is 0xf970bbc8.)

(2) it gets a stores-r reply from ni, but $ni \neq ri$. This may happen if ni has more information about who *actually* holds id, because it's closer. In this case it should log `lazy finger table validation for entry 2 to 0xf970bbc8 is not itself, but instead recommends 0xff012356`. You do not need to change your local finger table.

(3) it gets no reply from ni. A non-reply indicates ni has failed. It should log `lazy finger table validation for entry 2 to 0xf970bbc8 fails with no reply`. Your node should then correct its finger table entry, rediscovering it however it built it in the first place.

Thus in case (3) your node should rebuild its finger table entry before continuing.

Input is just like stage 5, although the stage line will say 6, and in addition to the

"end_client" command, there will be a new "kill_client" command. When the manager reads this command, it will send it to the relevant client that should then exit immediately. The manager should then wait some time before processing new commands from the input file (say, 60 seconds, or if you need longer, justify why in your README) to give the ring time to rebuild itself.

As a simplification, you can assume there will be only one kill_client command in the input.

Sample input:

```
stage 6
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
kill_client charlie
store delta
store epsilon
search beta
search delta
```

Suggestions about handling finger table updates: *(Clarification and addition 2013-11-27)* From discussions on the class discussion boards, we suggest the following alternatives to handle updating finger table entries after a node failure.

First, the originally proposed method is lazy finger table validation. The intent here is that a node should check that a finger table entry is correct before it is used ("used" means: sends a message to that destination). If the entry is wrong (typically because the destination is down, but we walked through all the cases above), it should then correct the entry and use the new correct one.

One possible confusion: does a node need to check its entry when it returns it? I.e., if node A sends stores-q to B and B is going to return node C, does B need to check C first? This check results in recursion, and could cause a B to delay arbitrarily before replying to A. We suggest you should *not* do recursion. B can just return what it has without checking; it's A's job to check that the result is correct. A should iterate and redo queries if they're wrong.

This leaves two problems for A and B. First, if B doesn't have a good answer, A can't necessarily find a better answer. A should therefore retry queries that fail, after a short delay to give B a chance to fix. However, if B doesn't check what it returns, it will never get any better. It's B's job to fix this, but it doesn't know it has a problem. To fix this

problem, we suggest that A can "shoot down" a bad finger table entry in B. One could do this by using update-q(ni,si,sp,i), perhaps using an unusual sp (say 0) to indicate the this is a shootdown and not an update. With this approach, A might get a reply update-r(ni,si,sp,i) with the corrected node C, or A could reissue the stores-q command. We believe that if nodes do retries and shootdown messages then incorrect finger tables will get corrected and lazy validation will work.

If you implement shootdown messages, please document that in the README for this stage. A node sending a shootdown message should log "update-q sent (0xa9367d92 0xf970bbc8 0 2): shootdown"; a node receiving a shootdown message should log "update-q sent (0xa9367d92 0xf970bbc8 0 2): shootdown received" and then log whatever messages it sends and receives to fix its finger table.

Alternatively, we will allow *preemptive* correction of finger tables. A simple way to do preemptive correction would be to detect node failures using hello-predecessor-q. Once a node detects a failure, it needs to inform all nodes in the ring. It can do this by telling its successor, and having them tell their successor, etc. Each node that hears of a failed node should search their finger table and correct any incorrect entries.

You will need a new message for failed-node-notification. A node that starts preemptive notification should log "hello-predecessor-q triggers node failure notification", and you should log something for your new message that propagates the failure around the ring. If you choose preemptive detection and notification, please document that you're doing this approach and what your new message number, format, and log statements in the README.

Sample output: As expected, the filename should include "stage6". We will provide partial sample output for this stage shortly after the project is released.

Contents of stage6.README.txt:

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

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

6.3) How do you detect that a given client is down?

6.4) How did you support receiving messages and also handling timeouts to do stabilization, when the replies might happen at the same time?

5.2 Stage 7: An Adversary

A problem with some peer-to-peer systems is that they assume that all nodes will cooperate, yet in the real world they don't always do that. Stage 6 looked at the problem of nodes departing the ring unexpectedly, one kind of failure. We next consider a different kind of failure, and a possible solution to both.

The TKCC (Triad Killers Community Club) is an organization that hates peer-to-peer networks. They want to stop Triad, and to do this they actually create a bunch of bogus Triad nodes. Their bogus nodes do everything normal Triad nodes do, except they replace

all content that TKCC doesn't want stored with a file of the appropriate type that says "nyah, nyah". Since there is nothing to stop a client from joining the ring and taking over its piece, this approach allows the TKCC to blockade parts of the ring.

The Triad operators were not happy with this interference, so devised a plan: each data item should be replicated and stored at 4 different places around the ring. Replication forces TKCC to put up many more interfering clients, and it also helps the ring avoid losing data when one node unexpectedly leaves.

For stage 7, implement the following additions to the prior stages: when computing the id of a node, do it normally. However, when computing the id of a data item as part of a store command, store copies of the item in all four locations found by looking at all combinations of the top two bits. Thus, if the item foo is stored at 0xa9367d92, please store copies in 0x29367d92, 0x69367d92, 0xa9367d92, and 0xe9367d92.

When searching (with the search command), you must retrieve the two closest copies to the node doing the search (where closest is defined by what will be found in fewest steps searching around the ring in an ascending order, since that's the direction supported by finger tables). (*Clarification 2013-11-27:* That description of "closest" is meant to mean "in key space". For exaple, if foo is at 0x29367d92, 0x69367d92, 0xa9367d92, and 0xe9367d92 and a search starts from a node with id in the range from 0xe9367d93 to 0x029367d92, then the nearest two are 0x29367d92 and 0x69367d92, while when starting from a node with id 0x293667d94, the nearest are 0x69367d92 and 0xa9367d92.) It should compare the two readings, and log "search S to node NI and NJ, key PRESENT and VERIFIED" if both contents match, or "search S to node NI and NJ, key PRESENT but DISAGREE" if they don't. (Note that you can also tell which contents are correct, since you can compute your own hash of the contents to see if it is correct. And of course, in this case, you happen to know that the content "nyah, nyah" is incorrect. Thus this double search actually allows error correction, provided one of the two polled nodes is valid.)

Finally, we will add one new configuration command: the `start_tkcc` command should create a client that changes the contents of anything it stores as described above. As a simplifying assumption, you may assume that we do not make the first client a tkcc client. And we add two new messages to allow one to retrieve the contents from another client. We add that below with the ext-stores-r reply; we also add the ext-stores-q to draw this reply. (I think you will find these messages are required to complete stage 7.)

17. **ext-stores-q** (ni, di) ask target ni for 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). This version is identical to stores-q, except that it elicits an ext-stores-r reply, described next.

18. **ext-stores-r** $(ni, di, ri, rp, has, SL, S)$ reply stating that ni's best estimate of the node that stores di has id ri at port rp. The has value is 1 if $ni = ri$ and ri has the exact id di, otherwise 0. (In other words, if you queried the node holding the exact hash and that node has stored that data locally.) In addition, this extended version of stores returns the contents via SL and S.

Sample input:

```
stage 7
nonce 1234
start_client delta
start_tkcc bravo
start_client charlie
start_client alpha
start_client echo
start_client foxtrot
store alpha
store beta
store gamma
store delta
store epsilon
search gamma
search delta
```

Sample output: Again, the filename should include "stage7". We will not provide sample output for this stage.

However, for the sample input, we think you should find that searching for gamma results in disagreement, while delta is OK.

Contents of stage7.README.txt:

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

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

7.3) From a technical point of view, TKCC is attacking Triad by putting up false nodes. One way to judge the effectiveness of this attack is to evaluate how many IDs m false nodes will block. First, *without* the replication added in stage 7, suppose there are n nodes in Triad, and 1 false node is added. What fraction of the ID space is preempted in this case? Please consider the *average* case here, not worst case. In the worst case, everything hashes to a single node!

7.4) Of course, stage 7 adds replication. With replication (as per stage 7) and 1 false node, what fraction of the ID space is now impeded? Again, please consider the *average* case here, not worst case.

7.5) Do you have one other suggestion about how to *prevent* Triad from working? (In other words, answer the question: if you were TKCC, what technical step would you take to impede Triad.)

7.6) Do you have one other suggestion about how to make Triad robust to attacks? (In other words, answer the question: if you were the Triad developers, what technical step would you take to protect Triad from your biggest fear of attack. The attack can either be what you mention in 7.5, or other attacks.)

6 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 `projc.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 `projc`. (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.

7 Research Project C

(*Addition 2013-11-27:* these details were omitted in previous project C.)

For Research Project C, 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 "next steps" part of your writeup for Research Project B.

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.

3. Code, data, and analysis results, provided when you submit this portion of the project.

4. With the other files you submit, you should include a README.txt file describing what you submitted. This README should also include a URL or filename for the weekly notes about your progress and meetings with your mentor.

5. A Project C report. This document should be about 4–8 pages summarizing your research project work over the semester. This report should be a mini-paper, providing a self-contained summary of everything you did on your research project. Whereas Project A and Project B could be more informal, Project C should be a final report similar to a conference paper. The papers we read this semester may be useful as a guide for how to write a good research paper. (It is due at the same time as the Triad

Project C, although the poster has a different deadline (see below).) It must include the following numbered sections:

Section 1 An introduction, giving an overview of what you're doing, why is this work important (interesting to the field), and what is new about it?

Section 2 A description of the relationship to other work. As you review related work, include citations in your text. Make sure each piece of related work identifies how it is alike and different from your work.

Section 3 A section listing what your goals were from Research Projects A and B with a statement about which goals you completed, and for those you didn't complete, why you didn't complete them. Possible Reasons may vary from "ran out of time" or "unable to get access to data" to "decided with mentor that result Y was more interesting than result X"; you may have other reasons as well.

(This section is the only one that would not appear in a real paper.)

Section 4 A discussion of your work. You will likely want to use multiple sections or subsections here, and you should identify your goal, methodology/approach, data collection, and results.

Section 5 A description of possible future work following Project C. You are not required to actually do this work, but it's always good to identify where the work might go, especially since the project work might not have perfectly aligned with the semester deadline.

Section 6 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.)

6. A poster summarizing your work. The poster session is tentitively scheduled for 7:30pm Dec. 12, in the same room as the CSci551 Tue-Thu section's final exam. We will confirm the date, time, and place as soon as we can. While it is not required, we strongly hope you will attend the poster session to describe your work to us, to your classmates, and to other attendees. *All* CSci551 students are encouraged to attend as well.

   We will print posters for you, but *only* if a copy is uploaded to the course moodle 48 hours before the presentation date. Alternatively, you may get it printed yourself (it should be about $50 at Kinkos, or you may have access to your own large format printer).

As with project B, since research is not always predictable, you will not be graded on executing exactly the plan in your Research Project B 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.

7.1 About Posters

There are two kinds of posters: (a) ones you stand by and talk about when people come up, and (b) ones that you leave alone and people walk up and read without you.

Medical doctors do type (b), where you go to a conference with 2000 people and have poster sessions with 300 posters. Those posters are little mini-papers with lots of text.

Computer science poster sessions that I've seen are almost *always* like type (a). With type (a), since you're standing there, the poster ends up more like PowerPoint slides, but with enough text that you CAN read them, but lots of pictures and the details are spoken by you standing next to it.

I recommend that posters follow (roughly) NABC format.

A rough format is:

```
.......TITLE.......
......authors......

abstract      problem (need)


              benefits
approach


competition
              conclusion
```

in two column format.

Other general poster advice:

- A poster should *never* just be an array of 8.5x11" power point slides (in a grid). That's like writing slides by pasting a copy of your paper into PowerPoint.
- To make it look non-grid-like, it's usually good to keep the column section at different locations. (see the examples we provide below).
- write interpretation of the graphs directly on the graphs, using arrows to point at what about the graph supports your claim. (Don't just put bullet points below the graph.)
- Always include a date on the poster (typically in gray in the lower margin), otherwise in 2 years you'll look back and forget the context.
- For a public poster, it's great to include a pointer to your paper or technical report or website for more information.

I recommend you do posters in PowerPoint on a 24x36" single page. You can also use Adobe Illustrator or Apple Keynote or Microsoft Viseo or LibreOffice Impress, or whatever you prefer. There are LaTeX templates for posters, although they can be somewhat difficult

to get custom layout. (All electronic posters are *much* easier and neater than physically cutting and pasting, as one of your professors did when he was in graduate school :-)

Some sample posters are on the Moodle.

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.)