COMP3331/9331 Computer Networks and Applications
Assignment 1 for Session 1, 2016
Assignment title: *Networking eDocuments*

Version 1.2
Due: 11:59pm Friday, 22nd April 2016 (Week 7)
Demonstration: Week 8 during regular lab hours (Week of 25 April - 29 April)

Updates to the assignment, including any corrections and clarifications, will be posted on the subject website. Please make sure that you check the subject website regularly for updates.

1 Change Log

1. Version 1.0 released on March 9, 2016.
2. Version 1.1 released on March 19, 2016.
3. Version 1.2 released onMarch 26, 2016

2 Introduction

In recent years, we have seen the introduction of various eBook readers (e.g. Amazon Kindle) and tablet computers (e.g. iPad). A common usage of both classes of devices is to read eBooks. It is quite possible that in the future the sales of eBooks will exceed that of hardcopy books. It may mean that school children or university students in the future may carry a tablet, which contains all their textbooks in electronic form, to schools or universities. You may still remember those weighty school bags (Don't they suck?) and a tablet computer may well save the spinal cords of the future generations. Well, that was a digression and let us get back to eBooks. Some eBooks today are not too different from their hardcopy counterparts but some eBooks may contain multimedia contents such as video, audio and animation, which hardcopy books do not have. Can we possibly add more to eBooks?

Let us think about what may be possible by looking at a story of two school children, whom we will call Sheldon and Penny. Both Sheldon and Penny are at the same stage in their schooling and use the same electronic maths textbook. Let us say that Sheldon has some difficulty in understanding how to derive equation (5.12) from equation (5.11) on page 23 of the text. If Sheldon is a member of an on-line discussion forum (similar to the one you use for this course), he can post a message to the forum saying, "On page 23 of the maths text XXX, how do you get eq (5.12) from eq (5.11)? Can anyone help?" An observation is that there is a separation between the on-line discussion forum and the electronic textbook. Can we merge them?

Let us imagine what we can get if we merge an electronic textbook with an on-line discussion forum. Let us get back to Sheldon. What Sheldon will do now is to highlight equation (5.12) and then choose the menu item "Post a message to the discussion forum". A window will then pop up and Sheldon will type this message into the window, "I have difficulty in figuring out how to derive this equation from (5.11). Can anyone help?". Now, let us switch our focus to Penny who happens to be studying page 22 of the same maths text while Sheldon is posting his message. When Penny turns to page 23 and has come to equation (5.12), she sees a little icon next to the equation which says that there is some discussion around that place.

She clicks on the icon and a window pops up with Sheldon's question there. Penny has some idea on how equation (5.12) is derived so she replies to Sheldon's question.

There seems to be a couple advantages in merging an electronic textbook with an on-line discussion forum. One advantage is that you can keep all the discussion at the same place. It also allows the authors of the text to join the discussion. Or, the authors of a text can look at the discussion items and think about how they can improve the text. In fact, we should not just limit ourselves to electronic textbooks. Using the same idea, we can associate a virtual book club with each eBook. You can even extend this idea to lecture notes or even this document!

Thus, the idea is that you want to network electronic documents (eDocuments) which can include eTextbooks, eBooks, eLectureNotes etc. What this means is that you want to introduce networking capability to the *eDocument reader software* so that they can talk to each other. Note that I have used the phrase *eDocument reader software* to emphasize that we are trying to network pieces of software (or applications) together. Examples of eDocument reader software include Acrobat Reader for PDF files, iBook for reading eBooks on iPAD, Acrobat Digital Edition for reading eBooks on various operating systems etc. These eDocument reader software do not talk to each other today but perhaps they should in the future!

The above description is meant to provide the motivation for what you will be working on in this assignment. Given that this course is on computer networking, the work that you will be doing will primarily be on networking. You will not be working on a real eDocument Reader and you will be using some pretty primitive interfaces to do your work. The key goal is for you to learn how to network pieces of software together.

Before we go on to describe the specifications of this assignment, I would like to raise one point. In the 1960s, computers were **not** connected to each other. We now see the power of the Internet because we connect the computers in the world together. Since then, we have added many devices and application to the Internet, and you also appreciate the power of networking them together. The power of networking may well be endless. You may well be able to do greater things by giving networking capability to objects and software that are not currently connected to the Internet. Perhaps the next time when you see an object or a software, or even a living thing, you may want to imagine what new greater things you can do by giving them the ability to network together - happy discovery :)

2.1 Learning Objectives

On completing this assignment you will gain an understanding of:

1. Socket programming
2. Protocol and message design for applications
3. The difference of push and pull models in network applications
4. The design of network applications

3 Assignment specifications

A note on terminology: The term *message* is used in computer networking to mean the data being passed from one application layer entity to another application layer entity. At the same time, the word *message* is used in everyday language to mean the content of the communication as in the phrase "discussion forum messages". In order to avoid confusion, from now onward, we will use *message* to mean application layer data and *post* to mean the content of the communication sent to on-line discussion forums.

3.1 General

For this assignment, you are asked to write two computer programs to prototype the ideas described in Section 2. Given that the objective of the assignment is to learn how to get computer programs to communicate with each other over a network, the interface part has been stripped down to the bare minimal. The two programs that you need to write are:

1. A server program which acts as the server of an on-line discussion forum and this will be referred to as the *server*.

2. An eDocument reader software and this will be referred to as the *reader*. The reader allows you to read eBooks. It also allows you to post to the on-line discussion forum and to read the posts. The reader can operate in one of two modes depending on how the reader finds out whether there are new posts. If the reader works in the *pull* mode, the reader is responsible for querying the server for new posts. On the other hand, if the reader works in the *push* mode, the server will forward the new posts to the reader when they arrive.

You will be asked to demonstrate that your programs are working in the CSE laboratory in Week 8 during your own laboratory session. We will explain how you will be doing the demonstration in section 3.4. That section will also help you to understand the requirements of the project.

3.1.1 Standard and extended versions

There are two versions of this assignment, a standard version (with a total of 12 marks) and an extended version (with a total of 14 marks of which 2 marks are bonus marks). The standard version requires you to demonstrate Steps 1–3 (see section 3.4), while the extended version requires you to demonstrate Steps 1–4 (see section 3.4).

Note that the bonus marks may not be proportional to the amount of extra work that you will have to do. They are there to encourage you to go beyond the standard assignment. Moreover, the bonus marks will be added to the assignments component of your final mark, and that this component will be capped at 25 marks.

3.1.2 Mandatory file requirements

There are a number of *mandatory* requirements.

1. You must name your server program `server.c` or `Server.java` or `server.py` if you attempt the standard version, or `server_ex.c` or `Server_ex.java` or `server_ex.py` if you attempt the extended version.

2. Similarly, you must name the reader program as `Reader.java`, `reader.c`, `reader.py`, `Reader_ex.java`, `reader_ex.c` or `reader_ex.py`.

3. Each program takes a number of input arguments:

   (a) You start the server by using one of the following commands:

   - If you use java, `java Server port_number`
   - If you use C, `./server port_number`
   - If you use Python, `python server.py port_number`

   where `port_number` specifies the port to which the server listens. If you attempt the extended version, you should replace `server` by `server_ex`.

   (b) You start the reader by using one of the following commands:

   - If you use java:
     `java Reader mode polling_interval user_name server_name server_port_number`

- If you use C:
  `./reader mode polling_interval user_name server_name server_port_number`
- If you use Python:
  `python reader.py mode polling_interval user_name server_name server_port_number`

where `mode` can either be `push` or `pull`, `polling_interval` (measured in seconds) is how often the reader queries the server for message if it works in the `pull` mode (This will be explained in Section 3.2. Note also that `polling_interval` is not used in push mode and you can enter any value for `polling_interval` for push mode), `user_name` is the identity of the user (you may assume that identities will be unique, i.e. each user will have a different name), and `server_name` and `server_port_number` are respectively the hostname and port number of the server.

3.1.3 eBooks

In order to imitate the reading of eBooks using the reader, we will be using paragraphs from the following 3 books:

1. *The Little Prince* by Antoine de Saint Exupery
2. *Frankenstein* by Mary Wollstonecraft Shelley
3. *A Portrait of the Artist as a Young Man* by James Joyce

There are 4 pages from each book and each page is an ascii file. Each page always contains 9 lines though some of them can be blank. These pages are available from the assignment website. The file names of the pages are `exupery_page1`, ..., `exupery_page4`, `shelley_page1`, ..., `shelley_page4`, `joyce_page1`, ..., `joyce_page4`. In the following, we will be referring to these books as exupery, shelly and joyce, using the author's name.

The following is the content of `exupery_page1`:

```
1 Grown-ups never understand anything by themselves, and it is
2 tiresome for children to be always and forever explaining
3 everythings to them.
4
5
6
7
8
9
```

The first three columns contain blank spaces, the fourth column contains the line number and then followed by a blank space and the text. We may be asking you to use any of these books or pages to do the demonstration. You can safely assume that these are the three books that you will use, there are always 4 pages for each book and each page has 9 lines.

The books are stored at the server. The readers do not have local copies of the books. You can download the books from the assignment page.

3.2 Specifications of the reader and the server for the standard assignment

All the messages between the server and the reader are to be sent by using the TCP transport protocol.

Although the description below is divided under the headings of reader and server, it is not possible to totally separate the description along this line because the two programs interact with each other so you may need to take note that some requirements for the server may be found in the section describing the reader, and vice versa.

3.2.1 The reader

The reader should run continuously and is expected to interpret commands issued from the keyboard. There are three commands: `display`, `post_to_forum` and `read_post`.

The reader operates in two different modes: push and pull. The command `display` will cause slightly different actions in push and pull modes. This will be explained below. The behaviour of `post_to_forum` and `read_post` in push and pull modes is identical.

On start up, irrespective of the mode of operation, the reader should first find an available port number on the local machine that it can use to communicate with the server. We leave it to you to figure out how you can do that.

The reader should initialise a database of the discussion forum posts when it starts up. You can assume that the database is empty on start up for this assignment. Note that, we use the term *database* in an abstract sense. We do not mean an SQL database, but rather some way to store this information locally at the reader (i.e. in memory).

3.2.1.1 The reader in pull mode

The behaviour of the reader in pull mode under different commands is:

1. The command `display book_name page_number` should display the specified page of a book, and the status of the posts associated with that page.

   In the pull mode, the `display` command will also trigger the reader to query the server for any *new* posts associated with the page specified in the command before the page is displayed to the user. This means that every time when a page is displayed, it shows the latest status of the forum discussion posts. (Note that this query is not done if the reader is in push mode.) The status of the post is to be shown in the first column of each line of text. The status can either be `n` which means there are new posts associated with that line or `m` which means there are posts (but these have been read) associated with that line. For example, if there are new posts for line 1 and old posts (which have already been read) for line 5 for the page `exupery_page3`, the display should appear as:

   ```
   n  1 This asteroid has only once been seen through the telescope. That was
      2 by a Turkish astronomer, in 1909. On making his discovery, the astronomer
      3 had presented it to the International Astronomical Congress, in a great
      4 demonstration. But he was in Turkish costume, and so nobody would believe
   m  5 what he said. Fortunately, however, for the reputation of Asteroid B-612,
      6 a Turkish dictator made a law that his subjects, under pain of death,
      7 should change to European costume. So in 1920 the astronomer gave his
      8 demonstration all over again, dressed with impressive style and elegance.
      9 And this time everybody accepted his report.
   ```

   In order for the reader to find out whether there are new posts, the reader should maintain a database of the posts that it has. When the reader queries the server for new posts associated with a particular page, it asks the server to send a summary of the posts associated with that page and compares the summary from the server against its own database. If the reader finds that there are new posts, it will download them from the server. The message format for the communications between the reader and server is not specified and you are free to design it.

   In the pull mode, the reader should also start a countdown timer whenever the `display` command is issued by the user. The reader uses only one countdown timer and it is associated with the page specified in the `display` command. When the user issues the `display` command, the countdown timer is set to `polling_interval` and the countdown begins. Assuming that no further

`display` command is issued within the duration of `polling_interval`, the timer will expire, i.e. the timer counts to zero. When this happens, the reader should query for new posts and at the same time reset the timer to `polling_interval` to begin another countdown. If there are new posts, the reader should display the line `There are new posts.` to the terminal. This procedure is repeated until another `display` command is issued, where the time will be reset.

An additional requirement is that reader should keep track of the current book and page number that is being read by the user.

2. The command `post_to_forum line_number content_of_post` allows the user to send a discussion post associated with a line number to the forum. Note that any number of words can be contained in the `content_of_post` part of the command, i.e. all the words after the line number should be interpreted as the discussion post to be sent. The reader should send this post to the server over the network. Note that the reader should in addition provide the book name, page number (we assume that the post is associated with the currently read page, and that is why we ask you to keep track of this information earlier on) and the username of the sender. You are free to design the format of the message(s) exchanged between the reader and server.

   When the server receives a new post, the server should allocate a serial number to this post so that each post can be uniquely identified. The server should then add this post to its database of forum discussion posts. Note that we do not specify how you generate serial numbers as long as the posts can be uniquely identified. We will leave the design decision to you.

   For simplicity, this will be all the processes associated with sending a new post to the forum. Note that the server does not have to send the serial number of the new post to the reader that sends the new post, nor will the reader add the new post to its database. Instead, when the reader next queries for the server, it will download this post, among other new posts, in one go.

   Another simplification is that we are not going to have a reply command, you can use `post_to_forum` for reply.

3. `read_post line_number` will display *all* the *unread* posts of the current page associated with the specified `line_number`. Remember that, this will require querying the server for updates. We do not impose any requirement on how you should display these posts but just ask you to do it in a way so that we can know that the reader has correctly obtained all the unread posts. For example, let us say the Leonard (a user) has 2 unread posts associated with line number 2 of the page `exupery_page1`, you can display the posts as:

   ```
   Book by Exupery, Page 1, Line number 2:
   3 Sheldon:  Adults are tiresome. They lack the imagination.
   4 Penny: Many of them think that there is only one way to do anything!
   ```

   where we have the serial number (3 and 4 in the above example) followed by the username and the content of the post.

3.2.1.2 The reader in push mode

If the reader operates in the push mode, the server will be forwarding any new posts to the reader. This operates as follows:

1. When a reader in push mode starts up, it registers with the server that it wants to operate in the push mode. The reader should inform the server about the port number that the reader will use to communicate with the server. At the same time, the reader should send the server a list of discussion

posts that it currently has. After the server has received the list, the server should forward any new posts that the reader does not currently have. Note that you can accomplish the above in one or multiple message exchanges between the reader and the server, the decision is up to you.

2. Whenever the server receives a new post for whichever book (in practice, a user should be able to choose but let us keep it simple here), it will forward the new post to the reader. Note that the reader is expected to listen for these new posts from the server on the port number that it sends the server when it registers with the server earlier.

3. If the user is reading a particular page and new posts for *that page* arrive, the reader should inform the user by displaying `There are new posts.` to the terminal. Otherwise, if the new posts are for some other pages (i.e. not the one that is currently read by the user), the reader should store the received posts in the database and mark them as unread.

4. Whenever the `display` command is used to display a page, the status of the discussion posts is shown in the first column. Since the server is responsible for pushing the new posts to the reader in push mode, the reader should *not* have to query the server for new posts when the command `display` is issued. Instead the reader should consult its local database for the status of the posts. Note that this is different from the pull node.

Remark 1 *If a reader operates in the push mode, the ideal scenario is that the server should know when the reader starts up and shuts down, so that the server does not have to send posts when the reader is not in used. This can be difficult because the reader can crash and does not inform the server. You will not have to deal with the reader shutting down or crashing in the assignment.*

3.2.2 The server

The server should run continuously.

The server should implement the following functions:

1. It should maintain a database of discussion posts. Refer back to the reader specification for interpretation of the term, *database*. The database is empty at start up. When a new post is received, the server should assign a serial number to the post and adds the post to the database. It should also forward the post to any reader operating in the push mode. This also means that the server should maintain a list of readers in push mode, this will be referred to as the "push list".

2. The server should listen to the port number specified in the input argument for communication from the readers.

3. It should answer the query from the readers operating in the pull mode for new posts.

4. The server should push new messages to those readers operating in push mode.

The server should initialise a database to store the discussion posts. We will let you design the structure of the database. You can assume that the database is *empty* when the server starts. You will also need to decide what information should be stored in the "push list".

In order to check whether your server is working correctly, it should display information to the terminal when certain events have happened or certain actions have been taken. You will see examples of this in the section on demonstration 3.4.

3.3 Specifications of the reader and the server for the extended assignment

The extended assignment will require you to implement a chat function between the readers so that the users can have a real-time discussion.

When the reader starts up, it should send a registration message to the server providing it with certain information so that the server can send chat request messages (which will be explained in the next paragraph) to the reader. This message should be carried over TCP.

The chat function consists of a set-up stage where a chat request message is sent from one user (say, user A), via the server, to another user (say, user B). If user B accepts the request, a response message will be sent to user A via the server. The four messages in this set-up process (user A to server, server to user B, user B to server, and server to user A) are to be carried over TCP.

After setting up, the users will be able to chat to each other directly, without using the server to relay the chat messages. There are a few requirements here. Firstly, the chat messages are to be carried over UDP. Secondly, after setting up, any of the two users can send the first chat message. If we use the example in the last paragraph, even if user A is the one who initiated the chat request, user B can be the first to send the chat message. Thirdly, the sequence of chat messages does not necessarily have to be user A, user B, use A, user B and so on. A user can send multiple chat messages one after another.

Note that certain information about a user has to be passed from the reader to the server (during the registration process when the reader starts up) as well as from one reader to another (during the set up process) to enable the chat to happen. We will not specify it here because there are a few different ways to do that. We will leave the design to you.

The chat function consists of 2 comments:

1. The command `chat_request username` is used to initiate the chat set-up from the user who types this command to the user specified in the command.
2. The command `chat username content_of_chat_message` is used to send a chat message to the user specified in `username`. All the words after the `username` are to be interpreted as the chat message.

You should also Step 4 in section 3.4 on demonstration to understand the requirements for the chat function.

3.4 Steps of demonstration

You will be using 3 CSE machines in a CSE laboratory to do your demonstration. One of the machines will host the on-line discussion forum server. For ease of reference, we will refer to this machine as machine0 and and the other two machines as machine1 and machine2. You will run 4 copies of the readers, two on each of machine1 and machine2. In the following description, we will assume that the users are Sheldon, Penny, Leonard and Amy, though you can use any name you like in the demonstration since this is specified in the user argument. You should not hardcode username in your program. In the illustration below, we will assume that the readers from Sheldon and Leonard will run on machine1 and those from Penny and Amy will run on machine2.

Note that you should not hardcode any information regarding machine name, IP address, port number, user name, book name, page number and line number in your code. These can change during the demonstration.

Step 1

Choose one of the machines to host the server. You should take note of the hostname of the machine (which can be obtained by using the command `hostname -f` in a terminal) and in the following, we will assume the name of the machine is drum10.cse.unsw.edu.au but in the demonstration, you should make sure that you use the name of the machine that hosts the server. Use the following command to start an xterm in which your server program will run.

```
xterm -hold -title "server" -e "java Server 25000 " &
```

if you use java, or

```
xterm -hold -title "server" -e "python server.py 25000 " &
```

if you use python, or

```
xterm -hold -title "server" -e ".\server 25000 " &
```

if you use C. Note that this assumes that the executable for server.c sits in the current directory.

Note that we have assumed that the port number that the sever listens to is 25000. In the following, we will assume that the port number is 25000 but during the demonstration, the marker may ask you to use any non-well known port number. Note again that you should not hard code the server name and port number in your program.

The server should indicate the actions that it has taken:

```
The server is listening on port number [display the actual port number chosen here]
The database for discussion posts has been intialised
```

We of course expect that the server has really done these actions!

Step 2

The aim of this step is to test the pull mode of operation for the reader. On machine1, start 2 xterm's using the following commands:

```
xterm -hold -title "Sheldon" -e "java Reader pull 180 Sheldon drum10.cse.unsw.edu.au 25000" &
xerm -hold title "Leonard" -e "java Reader pull 180 Leonard drum10.cse.unsw.edu.au 25000"
```

or the corresponding commands for C and Python. On machine2, start an xterm using the following command:

```
xterm -hold -title "Penny" -e "java Reader pull 180 Penny drum10.cse.unsw.edu.au 25000" &
```

Note that you should not start Amy's reader yet because hers will operate in the push mode. You will do that later.

You will now carry out the following actions and the expected responses of each step is also specified.

1. Sheldon types `display exupery 1` to display the page in the file `exupery_page1`.

   Expected responses: The page will be displayed and since the database is empty on initialisation, the first column is blank. Sheldon's reader should have queried the server for posts associated with this particular page of the eBook, the server should indicate that a query is received from Sheldon for posts associated with page 1 of the book exupery.

2. Leonard types `display exupery 1`

   Expected responses: The page will be displayed and since the database is empty on initialisation, the first column is blank. Leonard's reader should have queried the server for posts associated with this particular page of the eBook, the server should indicate that a query is received from Leonard for posts associated with page 1 of the book exupery.

3. Sheldon types `post_to_forum 2 Adults are tiresome. They lack the imagination.`

   Expected response: The server should display:

   ```
   New post received from Sheldon.
   Post added to the database and given serial number (exupery, 1, 2, 1).
   Push list empty. No action required.
   ```

4. Penny types `display exupery 1`

   Expected response: The page will be displayed where the letter `n` in the first column of line 2 shows that there are new posts. The server should indicate a query from Penny.

5. Penny types `read_post 2` to read the new post

   Expected response: The post should be displayed. Note that the reader should also mark the post as read.

6. Penny types
   `post_to_forum 2 Many of them think that there is only one way to do something!`

   Expected response: The server should display:

   ```
   New post received from Penny.
   Post added to the database and given serial number (exupery, 1, 2, 2).
   Push list empty. No action required.
   ```

7. Sheldon types `display exupery 1`

   Expected response: The page will be displayed where the letter `n` in the first column of line 2 shows that there are new posts. The server should also show that it has received a query from Sheldon.

8. Sheldon types `read_post 2` to read the new posts

   Expected response: There should be two posts, one from himself and one from Penny.

9. Sheldon types `post_to_forum 3 They are rather narrow-minded.`

   Expected responses: The server should display:

   ```
   New post received from Sheldon.
   Post added to the database and given serial number (exupery, 1, 3, 1).
   Push list empty. No action required.
   ```

10. Penny types `display exupery 1`

    Expected response: The page should be displayed and the letter `n` should appear in the first column of both lines 2 and 3 to indicate that there are new posts. The server should also show that it has received a query from Penny.

11. Penny types `read_post 3` to read the new post.

    Expected response: The new post from Sheldon will be displayed.

12. Since Leonard's reader has been on the page `exupery_page1` for some time, the timer in his reader should expire sometime and it will check for new posts from the server. When this happens, Leonard's reader should indicate that there are new posts. The server should also indicate that a query is received from Leonard for new posts. Leonard will now read the posts and you should check that all the posts are there.

    Once Leonard has read his posts. You should have both Sheldon and Penny sending more posts to the discussion forum for the page `exupery_page1`. Leonard's reader should also stay on the page `exupery_page1`. We would like to see that Leonard's reader checks for new posts on the next expiry of the timer. You can now move onto the next step but you should keep an eye on Leonard's reader to make sure that new posts do arrive. You should of course read them when they arrive to make sure it is working.

Step 3

The aim of this step is to test the push mode of operation for the reader. For this part of the demonstration, you should carry out the following:

1. On machine2, start an xterm to run Amy's reader in push mode using the following commands:

    ```
    xterm -hold -title "Amy" -e "java Reader push 180 Amy drum10.cse.unsw.edu.au 25000" &
    ```

    Expected response: The server should:

    (a) Indicate that it has received a request from Amy's reader to work in push mode and has added it to the "push list"

    (b) Indicate that it has received a summary from Amy's reader

    (c) Indicate that it has forwarded the new posts to Amy's reader

    (d) Provide a list of the posts that it has forwarded to Amy's reader

2. Amy types `display joyce 2`

    Expected response: Since no one has posted any discussion for this page yet, the reader will simply display the page. Also, since Amy's reader is working in push mode, the server should not have received a query (for new posts) from Amy's reader.

3. Amy types `display exupery 1`

    Expected response: The page should be displayed and the letter `n` should appear in the first column of both lines 2 and 3 to indicate that there are new posts assuming that Sheldon and Penny have not sent posts associated with other lines earlier. If they have, the response should be different but you should be able to work that out. The server should not have received a query (for new posts) from Amy's reader.

4. Amy types `read_post 2` and then `read_post 3`

    Expected response: The posts should be displayed.

5. Amy, Sheldon and Penny type `display shelley 4`

    Expected response: The page should be displayed and there are no posts associated with this page. The server should indicate that it has received queries from Sheldon and Penny. No query from Amy should be received by the server.

6. Sheldon types `post_to_forum 2 He should not have played God in the first place.`

   Expected response: The server should indicate the receipt of this post, add it to the database and give it a serial number. It should indicate that Amy is on the "push list" and push the message to her reader. Amy's reader should indicate `There are new posts.`

7. Amy types `read_post 2`

   Expected response: The correct post should be displayed.

8. Amy types `display shelley 3`

   Expected response: The page should be displayed and there are no posts associated with this page. The server should not have received a query.

9. Penny types `post_to_forum 5 He should not have forsaken his own creation.`

   Expected response: The server should indicate the receipt of this post, add it to the database and give it a serial number. It should indicate that Amy is on the "push list" and push the message to her reader. Amy's reader should not indicate the arrival of the new post because she is not reading the page that the new post is associated with.

10. Amy types `display shelley 4`

    Expected response: The page should be displayed and there should be a `n` on line 5 and a `m` on line 2. The server should not have received a query.

11. Amy types `read_post 5`

    Expected response: The correct post should be displayed.

Step 4

The aim of this step is to test the chat function and you will only need to demonstrate this step if you attempt the extended version of the assignment. The chat function will be tested in isolation of the other functions.

The marker will choose a pair of users to participate in the chat. Note that any pair of users may be chosen as long as they are from different machines. In the example below, we will have Leonard initiating a chat request with Amy. Note that the readers, on starting up, should have already registered with the server. The server should have indicated that it has received these messages earlier on.

The chat set up stage consists of the following message exchanges:

1. Leonard enters in his reader the command `chat_request Amy`. Upon receiving this command, the reader should format a chat request message and send it to the server.

2. The server should indicate that it has received a chat request from Leonard to Amy. After that it should send a chat request message to Amy saying that Leonard wants to talk to her. The server should also indicate that this message has been sent.

3. Upon receiving the chat request message, Amy's reader should display the line `Chat request from Leonard. Accept? [y/n]` and the reader waits for a reply. In the demonstration, we will assume that Amy will choose `y` to accept. Amy's reader should then send a response message to the server.

4. When the server receives the response message from Amy, it should send a response message to Leonard saying that Amy has agreed to talk to him. The server should show the actions that it has taken.

5. When Leonard's reader receives the response message from the server, it should inform Leonard that he can now start the real-time chat.

You can now carry out a real-time chat with your marker and the marker will choose a user to send the first message. Note that the chat should not be strictly iterating between the two users. Both of them can send multiple chat messages one after another.

After Leonard has finished chatting with Amy, the marker will choose another pair of users to participate in a chat. For example, the marker may choose to have Leonard initiating a chat request with Penny. After the set up stage, which is similar to that described above, the marker will choose a user to send the first chat message. If the marker has chosen to have the one initiating the chat to send the first chat message earlier, then the marker will for this part choose to have the user accepting the chat request to send the first chat message, and vice versa. Following on the above example, if the marker has chosen Leonard to send the first chat message when he chats with Amy, the marker will choose to have Penny sending the first chat message in the second part of the demonstration. The aim is to test that either one of the users can be the first one to send a chat message. Happy chatting!

4 Additional Notes

- This is not a group assignment. You are expected to work on this individually.
- How to Start: Sample client and server programs have been uploaded to the Week 3 Lecture material page. These are a good starting point to start your development. You will also find several links to network programming resources on that page. Students are also urged to complete the practice programming lab (Week 4).
- Language and Platform: You are free to use either C or JAVA or Python to implement this assignment. Please choose a language that you are comfortable with.
- The programs will be tested on CSE Linux machines. So please make sure that your entire application runs correctly on these machines (i.e. your lab computers). This is especially important if you plan to develop and test the programs on your personal computers (which may possibly use a different OS or version). Note that CSE machines support the following: **gcc version 4.9.2, Java 1.7, Python 2.7.3**.
- You may use the basic socket programming APIs provided in your programming language of choice. However, you are NOT permitted to use ready-to-use libraries such as SocketServer.TCPServer in Python.
- You are free to design your own format and data structure for the messages. Just make sure your program handles these messages appropriately.
- You are strongly encouraged to use the online forum to ask questions and to discuss different approaches to solve the problem. However, you should **not** post your solution nor any code fragment on the forum.
- You can also come to the lecturer's consultation if you have any question.

5 Assignment Submission

Please ensure that you use the mandated file name. You may of course have additional header files and/or helper files. If you are using C, then you MUST submit a makefile/script along with your code (not necessary with Java and Python). In addition you should submit a small report, report.pdf (no more than 3 pages) describing the program design, a brief description of how your system works and your message design. Also discuss any design tradeoffs considered and made. Describe possible improvements and

extensions to your program and indicate how you could realise them. If your program does not work under any particular circumstances please report this here. Also indicate any segments of code that you have borrowed from the Web or other books.

You are required to submit your source code and report.pdf. You can submit your assignment using the give command in an xterm from any CSE machine. Make sure you're in the same directory as your code and report, then do the following (note: instructions apply to students in COMP3331 as well as COMP9331):

1. Type tar -cvf assign.tar filenames
   e.g. tar -cvf assign.tar *.java report.pdf
2. When you're ready to submit, at the bash prompt type 3331
3. Next, type: give cs3331 ass1 assign.tar (You should receive a message stating the result of your submission).

Alternately, you may submit the tar archive via the submission link at the top of the assignment webpage.

Important notes

- The system will only accept assign.tar submission name. All other names will be rejected.
- Ensure that your program/s are tested in CSE Linux machine before submission. In the past, there were cases where students had problems in compiling and running their program during the actual demo. To avoid any disruption, please ensure that you test your program in CSE Linux-based machine before submitting the assignment.
- The demonstration will be conducted based on the files that you have submitted.

You can submit as many times as you wish before the deadline. A later submission will override the earlier submission, so make sure you submit the correct file. Do not leave until the last moment to submit, as there may be technical or communications error and you will not have time to rectify it.

Late Submission Penalty: Late penalty will be applied as follows:

- 1 day after deadline: 10% reduction
- 2 days after deadline: 20% reduction
- 3 days after deadline: 30% reduction
- 4 days after deadline: 40% reduction
- 5 or more days late: NOT accepted

Note that the above penalty is applied to your final total. For example, if you submit your assignment 2 days late and your score on the assignment is 10, then your final mark will be $10 - 2$ (10% penalty) $= 8$.

6 Plagiarism

You are to write all of the code for this assignment yourself. All source codes are subject to strict checks for plagiarism, via highly sophisticated plagiarism detection software. These checks may include comparison with available code from Internet sites and assignments from previous semesters. In addition, each submission will be checked against all other submissions of the current semester. Do not post this assignment on forums where you can pay programmers to write code for you. We will be monitoring such forums. Please note that we take this matter quite seriously. The LIC will decide on appropriate penalty for detected cases of plagiarism. The most likely penalty would be to reduce the assignment mark to ZERO. We are aware that a lot of learning takes place in student conversations, and don't wish to discourage those. However, it is important, for both those helping others and those being helped, not to provide/accept any programming

language code in writing, as this is apt to be used exactly as is, and lead to plagiarism penalties for both the supplier and the copier of the codes. Write something on a piece of paper, by all means, but tear it up/take it away when the discussion is over. It is OK to borrow bits and pieces of code from sample socket code out on the Web and in books. You MUST however acknowledge the source of any borrowed code. This means providing a reference to a book or a URL when the code appears (as comments). Also indicate in your report the portions of your code that were borrowed. Explain any modifications you have made (if any) to the borrowed code.

7 Marking Policy

You should test your program rigorously before submitting your code. Your code will be marked using the following criteria:

- Source code design (good structure and well commented): 1 mark
- Successful demonstration of Steps 1–3: 10 marks. This is broken down into:
  - Correctly displaying a page with the status of the post (0.75 marks)
  - On-demand query in the pull mode triggered by the `display` command (1.75 marks)
  - Query triggered by timer expiry in pull mode (2.5 marks)
  - Push mode (2.5 marks)
  - Posting of messages (1.75 marks)
  - Reading of messages (0.75 marks)
- Successful demonstration of Step 4 (real-time chat): 2 marks (bonus)
- Report: 1 mark

We would of course like you to complete the entire assignment. You should demonstrate as much as you are able to finish to the marker.