

# Programming Assignment 2

---

## News server and News client

*Handed on March 24, 2007*

*Phase 1 Delivery Due on April 10, 2007 (11:59pm)*

*Phase 2 Delivery Due on April 24, 2007 (11:59pm)*

### 1 What are we going do this time?..The Introduction

No.No..No Web clients..No usual stuff this time.. This programming assignment is about implementing the NNTP(come on, go Google..) in a simplified way. In canonical terms, this assignment requires you to implement a pull/push protocol. Putting it a scenario, the client can either pull/push messages from/to the server in which these message are stored.

### 2 Let's cook..Oh no..!..The Scenario

To tell things in simpler way,

- The client can post(push) a message to the server which the server stores.
- The client can download(pull) a message or a number of messages from the server.

The recipe seems to be raw right..Lets bake it a little..Lets see how the server stores these messages..

- The server sequences the order of arrival of these messages by timestamping them.
- The server uniquely identifies these messages by assigning them an id when they arrive. So at the server, each message has a header consisting of its *id* and its *timestamp* of arrival and a body which contains the actual message. This is how the server stores the messages. The *ids* are not random, they get incremented by 1 for each new message. But, the starting *id* assigned for the first message may be random. So these *ids* also cause a sequencing of the messages.

The structure of the stored message is as follows

$$stored\_msg := (header(id,timestamp),msg)$$

- The server should store these messages in a file for later retrieval. But still the server can hold the latest arrived messages in the main memory for fast retrieval by the client. That makes sense right!

Now that we know how a message is stored in the server, lets define the client and thereby the services of the server.

- The client can send a hello message to the server to learn the start message id and the last message id.

- The client can request and download a message by specifying its id. So now by knowing the first and last message id using the hello message, the client can download all the messages stored in the server.
- Since the messages are timestamped at the server, the client can query the server for ids of all the messages that arrived after/before the specified time. Here also, the client by asking the server all the ids of messages that had been posted before the current time, the client can download all the messages posted before current time.
- Of course, each of the server's reply should contain a status code conveying the success/failure of the request.

Lets add salt and spices to our recipe.. Lets define what these messages are..

Each message is a news headline. Each news headline consists of a headline text and a topic associated with it. To keep it simple, the topic is a single word string identifying the category of the news. Some examples of topic are 'politics', 'business', 'sports' etc. And as you guessed, the headline text is a single line of text. So,

*news\_headline := (topic, headline\_text)*

*Examples:*

(science, Star explodes halfway across universe)

(business, JPMorgan Chase makes \$1B-plus on Visa IPO)

Now that we know what a message is, let's see the functionalities provided by the client to the user.

- The client gets the hostname of the news server from/to which, the user will be reading/posting news headlines.
- The client should present an user interface through which an user can read and post headlines.
- The client should allow the user to post a headline news to the specified news server. The user supplies the headline news text and the topic.
- The client should should download all the headlines available on the specified news server and allow the user to view these headlines. The client should download the headlines into a file so that the user can read those even offline.
- The client should periodically query the news server for new headlines and update its store. To keep things simple, lets assume that the user issues a command to update the store with the new news headlines. For this, the client asks the server whether any new headline news has been posted after the last synchronized time. The client has to store the time every time its store. If yes, the server gives the ids of those posts. After receiving these ids, the client downloads each of the message by the id.
- The user who is using the client can ask the client to show headlines of a particular topic. If such a topic did not exist, then the clients return with an helpful error.
- The user can ask the client to list the various topics on which headlines are available. For this, the client collects all the topics of the headlines as and when they are downloaded.

### 3 Let's see how it looks...The Implementation

Now let me carve out the server and the client..And this is how you will be designing your client.. The following are the sample executions of the client and comments are there to help you understand.

#### 3.1 News server

The server will listen on a PORT specified as a commandline argument. The server has to provide reply following requests.

```
ssslab04:~/newstools$./newsserver 1191
```

1. HELLO

The server should send the ID of the first headline and the ID of the last headline in its store. If server does not have anything in its store, it sends zero as ID for both the first and last headline.

2. POST headline H of Topic T

The server should assign a new unique ID to this headline, timestamp it and store it. The server should respond the client with a status code conveying the success of the request.

3. GET news headline of ID

The server finds the headline with ID and sends it to the client. If it does not find any headline with that ID, it responds with a status code conveying failure.

4. GETALL news headlines AFTER/BEFORE mm:dd:yy:hh:mm

ie., Get IDs of headlines that have been posted before/after time mm:dd:yy:hh:mm  
The server finds the IDs of those headlines which were posted before/after the time mm:dd:yy:hh:mm and sends them to the client. If it does not find any headline before/after the specified time, the server responds with a status code conveying the failure or it can send a ID equal to zero.

The server must be able to handle *multiple clients* at the same time, which means the server must have some threading/forking mechanism to handle multiple clients at the same time.Regarding status codes, you can borrow ideas from HTTP. Even NNTP has status codes. You get ideas from either of them.

The termination of the server process will by either SIGTERM(using `kill` command) or SIGINT(using `Ctrl-C`) signals. So handle these signals and make the server quit gracefully without damaging any connected clients.

*You don't have to stick to the above requests alone. You don't need to design the requests same as above. Also its your design choice to use status codes or not in the protocol. The only condition is the server and client should obey the scenario described in Section 2. But the Client will have to stick to the execution traces given below.*

## 3.2 News Client

The News client should connect to the specified news server and present a prompt(>>) to accept commands from the user. The prompt is the user interface provided by the news client.

If the news server is up and running, the client sends a HELLO msg and gets the IDs of first and last headline on the server. If the client is connecting to this news server for the first time, it should create a file called 'news store file' named <newsserver\_hostname>.news eg. sslab04.cs.purdue.edu.news, in the current directory and download all the headlines on the server into that file and then display a status message indicating how many headlines were downloaded (shown below in the piece of execution trace). If the client had previously connected to this news server, it would have created this file. Now the client checks the ID of the last headline on the file and compares with the ID of the last headline received from the server in reply to the HELLO msg. If it is higher, it downloads the new headlines and stores them into the file and prints a status message indicating how many new headlines were downloaded. In the case of first time connecting to a news server, this number displayed indicating the number of headlines downloaded will be equal to all the headlines on the server.

```
$newsclient sslab04.cs.purdue.edu 1191
5 new headline(s) downloaded.
>>
```

If the news server is not up, the client does not quit immediately. The client checks the existence of the news store file corresponding to the news server and if it is present allows the user to read (using the show command described in the later sections) the stored headlines that were downloaded in the past. This allows the user to read headlines even when the server is offline.

```
$newsclient sslab04.cs.purdue.edu 1191
cannot connect to sslab04.cs.purdue.edu.
news store found for the server.
ready for offline use.
>>
```

What if the client was not able to connect to the specified news server and there was no news store file corresponding to that news server? Print a helpful message and then quit.

```
$newsclient sslab04.cs.purdue.edu 1191
cannot connect to sslab04.cs.purdue.edu.
no news store found for this server.
so quitting..
$
```

The prompt should accept the following commands.

- post

- sync
- show
- quit

These commands are explained in detail in the following sections.

### 3.2.1 post

The syntax of the `post` command is `post <topic> <headline_text>` As it implies from the name itself, the `post` command is used to post headlines on a topic.

```
>>post business "Starbucks pays $100M in tip suit."
POST success.
>>post tech "It's alive! Robot rises during spacewalk!"
POST success.
>>post science "Star explodes halfway across universe."
POST success.
>>post tech "Apple considering free access to iTunes."
POST success.
>>post science "Iceland phasing out fossil fuels for clean energy."
POST success.
>>post business "Goldman, Lehman may face downgrade."
POST failed.
```

### 3.2.2 sync

The `sync` command is used to download the latest news headlines and update the news store. The command returns with the number of new articles read.

```
>>sync
1 new headline(s) downloaded.
>>sync
0 new headline(s) downloaded.
>>sync
4 new headline(s) downloaded.
```

The `sync` command works by using the `GETALL AFTER <last_sync_time>` request (refer previous section). This asks the server to give all the IDs of headlines that were posted after the `last_sync_time`. There is also another ugly way to get the new headlines. As you would have thought, by this time, you can use the `HELLO` request to get the ID of the last headline posted.

### 3.2.3 show

The syntax of the `show` command is `show [all|topic <topic>|topics]` The `show` command is used to view the locally downloaded news headline. The `show` command has 3 options.

- The `all` option is used to view all the downloaded news articles.
  - It displays all the downloaded headlines in the format `<serial_no>::<id>::<headline>`

- At the last line, it displays the total number of headlines displayed.
- The `topic` option is used to view news articles posted on the given topic.
  - It displays all the downloaded headlines of given topic in the format `<serial_no>::<id>::<headline>`
  - At the last line, it displays the total number of headlines displayed.
- The `topics` option is used to list the various topics on which people have posted.
  - It displays the topics in the format `<serial_no>::<topic>`
  - At the last line, it displays the total number of topics displayed.

```
>>show all
#1::1::business::Starbucks pays $100M in tip suit.
#2::2::tech::It's alive! Robot rises during spacewalk!
#4::3::science::Star explodes halfway across universe.
#5::4::tech::Apple considering free access to iTunes.
#6::5::science::Iceland phasing out fossil fuels for clean energy.
6 headline(s) total.
>>show topic business
#1::1::business::Starbucks pays $100M in tip suit.
1 headline(s) total.
>>show topic politics
0 headline(s) total.
>>show topics
#1::business
#2::tech
#3::science
3 topic(s) total.
```

### 3.2.4 quit

The `quit` command, as implied, quits the client.

```
>>quit
$
```

*I hope I have given a good delicious recipe..*

## 4 Now its you turn..to try it out..!!

**So Tasks for you are..**

- Design a protocol for the client to interact with the server to support the above given scenario.
- Implement the server and client. The client should conform to the sample executions presented in the previous section.
- You can choose either C or C++ to implement. Make sure it compiles with `gcc/g++` available in the SSLAB machines.

## 5 Submission

### 5.1 Phased submission - 2 Phases

You will be developing this assignment in a phased out manner.

**Phase 1** For phase 1, you will deliver the server and client with following functionalities.

1. Your News server should support
  - (a) the `HELLO` message(refer Section 1)
  - (b) Posting of headlines (No need to store them into a file. You will be doing that in Phase 2.)
  - (c) Download of headline by the clients (Just give the support of download headline by ID)
2. Your News client should support
  - (a) the `HELLO` request (No need to store the downloaded headlines into a file, but have them in memory. You will be doing that in Phase 2. By this, the offline use is ruled out.)
  - (b) the `post` command
  - (c) the `show all` command
  - (d) the `quit` command

**Phase 2** For phase 2, you will add the following functionality to your previous deliveries.

1. Your News server should now(in addition to the previous phase requirements) support
  - (a) storing the posted headlines into a file.(which means even if the server is restarted, new clients must be able to read the old posts.)
  - (b) the `GETALL BEFORE/AFTER mm:dd:yy:hh:mm` request.(Refer Section 4)
  - (c) multiple clients
  - (d) graceful exit meaning handle `SIG_TERM` and `SIG_KILL` signals.
2. Your News client should now(in addition to the previous phase requirements) support
  - (a) downloading of headlines into a file, thereby supporting offline use.
  - (b) the `sync` command
  - (c) the `show topic <topic>` command
  - (d) the `show topics` command

So there will be 2 checkpoints/submissions. Your submission should contain a directory called `newstools` which should include the following.

- All source files.
- A README describing how to compile and execute your server and client
- A DOC/PDF document describing the protocol design. Submit this just in Phase 1. If you have any corrections, you can submit the updated one in Phase 2 but there will be deductions. So make sure, you first design and refine the protocol before your start to code.
- Please do not submit any binaries or object files.

## 5.2 How to submit?

- Make sure you login on to a SSLAB lab machine.
- In the parent directory of your submission directory `newstools`, type the command:  
`turnin -c cs422 -p lab2 <submission-dir-name>`  
where `<submission-dir-name>` is your directory containing your submission (`newstools`).
- Verify your submission by typing the following command:  
`turnin -v -c cs422 -p lab2`  
Do not forget the `-v` above, as otherwise your earlier submission will be erased (it is overwritten by a blank submission). Note that resubmitting overwrites any earlier submission and erases any record of the date/time of any such earlier submission.

## 6 Grading Criteria

- Protocol Designed.
- Correctness of the implementation of server and client.
- Organization and documentation of the Code.

## 7 Tips..

Things that might help before you start writing your code..

- Get your hands dirty with Socket Programming..
- Know how news servers and news clients in real world work.
- Just get an idea how NNTP works which will help you in designing a protocol for the scenario persented above. You can borrow ideas from NNTP.
- Paperwork the protocol that supports the given scenario between news server and news client. Discuss among your team and come up with a well refined one.
- Design test cases to test your server and client.

## 8 Jumpstart Pointers

Pointers that can help you get you started..Of course, you have Google..

- Beej's Guide to Network Programming, <http://beej.us/guide/bgnet/>
- How the Usenet News Protocols Work, <http://people.dsv.su.se/~jpalme/e-mail-book/usenet-news.html>
- History of Usenet, <http://en.wikipedia.org/wiki/Usenet>
- NNTP(rfc977), <http://www.faqs.org/rfcs/rfc977.html>

After you have a design of the protocol, you can start to code.

---

ALL THE BEST

---

*If you have any queries, please post it on the news group.  
Or discuss with me([fjohnber@cs.purdue.edu](mailto:fjohnber@cs.purdue.edu)) during my PSO session(Tuesday session 1:30pm-3pm, B131 SSLAB)/office hours(Wednesday 1-3pm, B116G).*