

# Network of Message Brokers

Joseph Cordina

March 20, 2010

This is the description for Assignment 2 of unit CSA 2205, System Programming, for the year 2009/2010. This assignment is worth 18 of the total mark (33) for this part of the unit. The deadline for this project is 28th May 2010. You may do this assignment in a group of maximum 2 people, yet you may do this assignment individually also. A single mark will be allocated to the whole group, so choose your partner carefully and make sure that the work gets evenly distributed. Under no circumstances should code be shared outside your group. You will submit your assignment making use of the ASS system<sup>1</sup>. Those of you without login access, please contact me on joseph.cordina at um.edu.mt. Each team might be asked to present their implementation of your assignment during which your program will be executed and the design explained. Please be reminded that you cannot copy and plagiarize to ease your way to a final submission. While you may discuss ideas with others, do not steal. You can find more information at

[http://www.um.edu.mt/\\_\\_data/assets/pdf\\_file/0007/19879/PlagiarismDeclarationForm.pdf](http://www.um.edu.mt/__data/assets/pdf_file/0007/19879/PlagiarismDeclarationForm.pdf)

Read this information well since after-wards no excuses will be accepted. Anyone resorting to such methods will be considered as trying to cheat and will even risk the removal from the degree program.

## 1 Submission Contents

You will submit your C files together with any accompanying header files (C language has to be used obviously) separately. Please do not use any tar or zip file formats. Additionally include any libraries you might have used. The compilation can assume the presence of 'gcc' on which your system will be compiled. Include with your submission *makefiles* that can compile your system<sup>2</sup>. You will need to provide two makefiles, one to generate the .a files (library file to be used by the clients) and another for the **brokers**. You can easily find details on how to generate a library file<sup>3</sup>. I will try your code by building your brokers and running it on any architecture. Then I will run separate instances of clients on Free-BSD, SunOs and Linux machines in any order. With your submission, you will provide a file called report.txt which will be a very **brief** description of how your system works and provides the required functionality. Make sure you mention any special techniques or tools you made use of and also any considerations you have made to improve efficiency. More importantly, give details of the network protocol between the interacting processes. Please do not include code inside this file, since this is obviously not readable. Additionally, please note down any bugs your system might have. Listing bugs is common practice, and will benefit the marking process. This file should also contain references to resources you have made use of in your program. Finally, you will submit a file 'protocol.jpg' containing an image that

---

<sup>1</sup>[www.cs.um.edu.mt/~jcord/ass](http://www.cs.um.edu.mt/~jcord/ass)

<sup>2</sup>You can find plenty of tutorials on the net that teach you how to set one up

<sup>3</sup>An example is [http://www.adp-gmbh.ch/cpp/gcc/create\\_lib.html](http://www.adp-gmbh.ch/cpp/gcc/create_lib.html)

describes in the most concise and clear manner possible your system's architecture and messages that are passed between the components.

Before submitting, please do clean up your code. Remove any dead code and useless comments. Every system calls output should be validated for errors and when appropriate, an error message reported to standard output. Also do place some comments to describe your structures in the code, at least to show that you understand how your own program works. Remember C is only unreadable if you make it unreadable. I cannot emphasize enough to be careful with your naming conventions and visual formatting.

## 2 The API to Messaging

As part of this assignment, you will be providing a library that provides a number of API calls that will allow a client to send and receive messages over the Internet. The functions provided will be very similar to System V message queues. The functions are:

```
struct broker_location {
    struct in_addr *broker_address;
    u_short port;
};

int msgsget(key_t key, int flag,
            struct broker_location *broker_address,
            int no_of_addresses);
int msgsnd(int msgid, const void *ptr,
           size_t nbytes, int flag, char *message_name);
int msgrcv(int msgid, void *ptr, size_t nbytes,
           long type, int flag, char *message_name);
```

I will only explain here the differences between the System V API and the one shown above. You can assume that the rest is the same, explanation of which is given in the notes. `msgsget` takes additionally a number of structures that contain a 32 bit IP addresses in network byte order (just like `struct sockaddr_in`) and also a port number. Also this call takes the number of addresses passed in. This call will connect to the brokers (all or any, depending on your design) that reside in the specified addresses and ports. Note that for this assignment you do not have to worry about setting *group*, *owner* and *other* permissions and also read, write or both permissions on the queue. Assume that anyone has access to write and read from the queue. The calls `msgrcv` and `msgsnd` have an extra parameter which is *message\_name*. This points to a string (null terminated) that gives the name of the queue to post and receive messages from. In other words, if any process posts messages on a queue called "test", then anyone who calls receive on queue called "test" will receive the posted messages. Additionally, for this assignment you **do not** have to support non-blocking operations on the message queue.

You are expected to provide the API above and allow it to be compiled to a static library. You will write a test program (which you do not have to submit with your deliverables) that makes use of this API. I will test your submission by creating sample client programs and link them to your library. To allow linking, a header file has been created, *msg.h*, which contains the api above and can be found on the usual website. This header file cannot be changed and you do not have to include it in your submission.

## 3 The Brokers

Brokers are servers whose responsibilities are to pass messages from one client to another. The API above will be used by clients to talk to the brokers. Your makefile will compile

the broker code base and will generate an executable called `broker`. This command will accept first a port number on which the broker will be listening to, and also a number of IP addresses/ports on which other brokers reside. One should be able to start as many brokers as one wishes on the same server machine and on other server machines. For example if one wants to start two brokers on Machine 1, and another on Machine 2, the following is a typical startup sequence

```
LOG into Machine 1 and go to path where broker has been compiled
./broker 6000
./broker 6001 127.0.0.1:6000
LOG into Machine 2 and go to path where broker has been compiled
./broker 6000 IPofMachine1:6000 IPofMachine1:6001
```

Note that each broker will need to be passed in information about brokers that had been started previously. Brokers should always be started before clients can connect to them. For this assignment you can assume that **all** brokers will be started prior to **any** client being started.

## 4 Redundancy, Load balancing and Failover

You might have been wondering why are we bothering starting multiple brokers, where one could suffice. Reason is that we want to provide facilities to enable load balancing and failover. This is common on the internet since servers do fail, network connections die, and servers get overloaded. Thus here are some rules that have to be satisfied by your implementation.

1. Message semantics have to be the same as the System V messages, thus should ensure ordered and unduplicated delivery. As much as possible, you should try to ensure guaranteed delivery.
2. Just like System V messages, you have to support multiplexing of messages over the same queue name through the use of the *type* field.
3. If a client connects to broker 1 and then calls `imsgsnd`, and another client connects to broker2 and calls `imsgrcv` on the same queue name, the message will still be successfully delivered
4. If a client specifies a number of addresses to connect to, then messages should be load balanced across the brokers specified (you can use any load balancing strategies you wish).
5. If any broker dies (gets killed in our case), then messages will still flow from any client to any client as long as the client's broker list includes a broker that is still alive<sup>4</sup>.

You can assume that startup of brokers after they have been killed and when the clients are already active will not be catered for in this assignment. For any of you that might be wondering what is the use of internet enabled messaging and would like to learn more on an enterprise level messaging system look up `ActiveMQ` and `OpenMQ`. It goes without saying that these messaging system provide solutions that go beyond the level of this assignment. In addition, if you want to learn about communication patterns in general between applications, something that is very important in industry today, check out this book <http://www.eaipatterns.com/>.

---

<sup>4</sup>If there is a very small chance that one message might be lost because the message was in transit when the broker is killed, then it is ok to not worry about it for this assignment.

## 5 What might matter

Marks will be allocated equally to a functional system that allows multiple clients to make requests to multiple brokers and also to a good and efficient design. I will run several concurrent clients connecting to multiple brokers communicating on different message queue. In addition, I will be killing one or more of the brokers when the clients are communicating! It goes without saying that if you find an application implementing something similar you cannot use it. Obviously having a system that is free of race conditions, deadlocks and still considerably efficient would be nice. For your own good, first design your system before starting the coding. It will result in a better system and a quicker turn around time. Instances of the clients will have to communicate with brokers through an Internet connection created from non-privileged user processes.

This system has to be implemented using BSD stream sockets running on the TCP/IP protocol. Both the clients and the broker processes must be fault tolerant. They can both be terminated with a CTRL-C or the kill command. Ensure that proper cleanup is performed in each case. Thus the server must terminate all its child processes (if any) and close all open sockets. The child must close all its connections properly before terminating. A client must be fault tolerant to broken connections, i.e. it will not crash when the connection is broken but will react accordingly. Ensure that you develop an appropriate protocol between the clients and the brokers and between the brokers (if any).

You can also assume that no deadline extensions will be given (as per usual really).

## 6 And To Conclude

Feel free to make use of any optimisations you can think of. Optimisations you should consider are implementation optimisations (better data structures, less indirect references, etc) and also algorithmic optimisations. Keep the allocated time you spend on this project in perspective with the allocated marks. As a suggestion, first sit down and design your solution well before implementing it since this will aid you a lot in your programming.

Good luck and if you need anything, contact me on joseph dot cordina at um.edu.mt. Even better, make use of the discussion group to discuss ideas and ask questions. You can find a link to the discussion group on my website. Other students might help you out in the solution. Just make sure you do not post any code or given concrete solutions.

Note that you will be doing a lot of design work and scrapping lots of ideas, so do not expect to do this assignment in the last few days before the deadline. Do it slowly and the ideas will mature. Remember this is mainly a design assignment (like all good system tools) and you only do the coding after you are sure your design is correct.

I might post on the website a list of designs that impressed me (lets hope there are a couple :)) !