Imperial College London

Department of Computing

MSc. C++ Programming – Assessed Exercise No. 2

Issued:	Monday 29 October 2012
Due:	Monday 12 November 2012
Lab Sessions:	Monday 29 October (pm)
	Wednesday 31 October (am)
	Friday 2 November (pm)
	Monday 5 November (pm)
	Wednesday 7 November (pm)
	Friday 9 November (pm)
	Monday 12 November (pm)

The Task

The Internet is a massive collection of computer networks, all linked together to allow individual computers and applications to communicate with one another seamlessly. This functionality is implemented using various layers of communcations *protocols*.

In this exercise you are asked to model a simple computer network, allowing different computers to communicate with one another via messages (packets). A very important aspect of network operation is that of *routing* - how packets are transferred from one node of the network to another in order to finally reach their intended destination. Your solution will model both of these. If you choose to take the Computer Networks and Distributed Systems course next term, you will learn all about the technologies and software that drive real-world computer networks.

Below, you will find a description of the components you are required to implement and their operation, as well as a specific scenario which you must simulate using your implementation. You should tackle this problem in two stages:

1. Draw a UML class diagram to represent the problem (you are not required to submit this, it is to help you better understand the problem and plan your solution).
2. Write appropriate class definitions in C++ and a makefile which, together with the main program given in the file `networkMain.cpp` (which you should not alter), generate a simulation of the specific scenario described below.

Information about the Model

1. In our model, there are *hosts*, which represent computers that will communication over the network, and there are *routers* which connect hosts to each other to form the network.
   (a) hosts may only be connected to routers;
   (b) routers can be connected to hosts or to other routers.
2. All connections in the network are *2-way* connections.
3. Each host, and each router, has an (integer) address. These addresses should be unique in order to avoid confusion.
4. Hosts can send *messages* (i.e. packets) to one another. Messages consist of:
   (a) a *source* address, identifying the host that sent the message;

   (b) a *destination* address, identifying the host intended to receive the message;
   (c) a string containing the message to be sent.

5. When a host receives a message (notice that this will necessarily be from a router), it prints out the message that it contains.

6. Routers contain *routing tables* which map host addresses to routes (sequences of router addresses) through the network, allowing messages to be routed to their destination.

7. When a router receives a message, it looks up the message's destination address in its routing table to find the 'next hop' destination;

   - if the router finds that the host is connected to itself, then the message is sent directly to the destination host;
   - otherwise, the message is sent to the next router.

Building and Maintaining the Routing Tables

One solution to the routing problem is for each router to maintain fixed routing tables which state how packets should be routed to their destinations for a particular network configuration. In a large, heterogenous network like the Internet, this is not a viable solution since the network topology may be constantly changing.

To cope with this, so-called *adaptive* routing algorithms exist. There are two main types of adaptive routing algorithm: distance vector, and link-state. Distance vector algorithms work by assigning to each of their neighbours a value indicating how close that neighbour is to the destination of a packet to be delivered. This means that each router has only a *local* view of the network. Link-state algorithms, on the other hand, allow each router to build up a picture of the *entire* network, and then each router will use this information to work out which of their neighbours is the best candidate to deliver a packet to its destination.

Our model will use a link-state algorithm which will be able to cope with hosts and routers being connected and disconnected. The basic idea is that each router will maintain, for each (reachable) host on the network, a list of routes through the network that lead to that host. The routers will build up this list by passing information on to one another about routes that are created or destroyed when hosts and routers are connected and disconnected.

The details of the protocol that your code should implement are as follows:

1. When a host is connected to a router, that router should create a new entry in its routing table associating the address of the new host with itself. The router should then notify its immediate neighbours that the host has been added to the network which can be reached via the singleton path through the network containing the host.

2. When one router is connected to another, they must exchange the information in their routing tables. So, for each route in their table, the routers first insert their own address at the beginning of the route and pass it on to the other.

3. When a router is notified by its neighbour that a new route to some host now exists, it should first check whether that route contains itself, and if so it does nothing. This will allow cycles to be avoided, when both routing messages themselves and propagating routing information, thus ensuring your algorithm will terminate!

   If the route does not contain itself, the router adds this route to its routing table. It then inserts its own address at the beginning of the route, and passes this route on to each of its neighbours.

4. When a host is removed from the network, its parent router removes that host's entry from its routing table; it then notifies its neighbours that the host has been removed.

5. When a router is notified that a host has been removed it first checks to see if it has an entry for this host in its routing table. If not, it does nothing; if it does have an entry, it is removed from the routing table and the router then notifies its neighbours that the host has been removed.

6. When one router is disconnected from another:
   (a) each one must first remove any routes in its routing table that start with its (now unreachable) neighbour router;
   (b) they must then notify each of their (still connected) neighbours that any routes containing a path between the two routers is no longer valid.
7. When a router is notified by its neighbour that a particular connection between two routers no longer exists, it must remove all routes in its routing table that contain this link. If no such routes exist in its table, no further action need be taken; if there were routes to be removed, after removing them, the router should notify its neighbours about the broken connection.

The Scenario to Simulate

You have been given a main program which sets up the a network and simulates the routing of several message as follows:

1. A host and router with address 1 are created and connected.
2. Another host and router, with address 2, are created but not connected.
3. Router 1 is connected to router 2.
4. Host 2 is connected to router 2.
5. Host 1 sends a message to host 2, with the text "This is a test".
6. A host and router with address 3 are created and connected to each other.
7. A host and router with address 4 are created and connected to each other.
8. A router with address 5 is created.
9. Router 3 is connected to both router 4 and router 5.
10. Host 5 is connected to router 5.
11. Host 4 sends a message to host 5 with the text "This is another test".
12. Router 4 is connected to router 5 (creating a loop in the network).
13. Host 4 sends another message to host 5 with the text "Hello world".
14. Router 2 is connected to router 3.
15. Host 1 sends a message to host 3 with the text "What a wonderful world".
16. Host 4 is disconnected from the network.
17. Host 2 tries to send a message to host 4 with the text "Is anybody out there?".
18. Router 3 is disconnected from router 5.
19. Host 3 sends a message to host 5 with the text "How many links must a packet walk down?".

The Output

The output of your program should look like this:

```
Host 1 sent a message to Host 2: This is a test

Host 2 received a message from Host 1: This is a test

Host 4 sent a message to Host 5: This is another test

Host 5 received a message from Host 4: This is another test

Host 4 sent a message to Host 5: Hello world

Host 5 received a message from Host 4: Hello world

Host 1 sent a message to Host 3: What a wonderful world
Host 3 received a message from Host 1: What a wonderful world

Host 2 sent a message to Host 4: Is anybody out there?

Routing of message failed at Router 2.

Host 3 sent a message to Host 5: How many links must a packet walk down?

Host 5 received a message from Host 3: How many links must a packet walk down?
```

Bonus Challenge

The specification above does not specify, of the potentially many routes through the network that may exist, which particular route a message should take. Once you have completed the main task given above, you can modify your program to ensure that messages are always sent along the *shortest* route through the network.

To demonstrate your bonus solution, you should copy your original solution and modify it so that each router prints some output when it forwards a message. So, for example, the output for the last message in the scenario above might be:

```
Host 3 sent a message to Host 5: How many links must a packet walk down?

Router 3 forwarded a message to router 4.

Router 4 forwarded a message to router 5.

Host 5 received a message from Host 3: How many links must a packet walk down?
```

You may also modify the main program to create a different network topology that exercises your bonus solution more rigorously.

N.B. You will only be given credit for the bonus solution if your solution to the main part of the exercise is completed!

Submission

1. In your `makefile` name your executable `network` so that, after running `make`, executing the command `./network` will trigger your program.

2. Your program files will be compiled and tested with the `networkMain.cpp` file that is provided to you; this is why you should not change that file. Your solution will be assessed based on the result of compilation and execution, but also by looking at the design and style of your code (i.e. whether it correctly and efficiently uses object-oriented design principles, and contains appropriate comments, etc.)

3. Put all the files necessary for compiling your program (i.e. the `.h`, `.cpp` files and `makefile`) into a `.tar` archive called `network.tar`. This can be done by running the following command within the directory containing your files:

   ```
   tar -cvf network.tar *.h *.cpp makefile bonus
   ```

   If you are not submitting a bonus solution, you do not need to include the final `bonus` at the end of the `tar` command.

4. Submit your `tar` archive on CATE.

Disclaimer The networking model outlined in this exercise is obviously far from realistic. For a start, network communication is ‘instantaneous’, and *not* concurrent - we have not incorporated a notion of time, and the sending of one message must be completed in its entirety before the next message can be sent. Secondly, in the real world, networks can contain huge numbers of interconnected hosts, and so it is not always feasible for routing tables to contain individual entries for every single host on the network. Real networks get around this problem by using *hierarchical* routing algorithms, such as the path vector protocol.

There are also many more aspects to network communication that we could have modelled, such as load balancing, etc. Considering such details, however, would be far beyond the scope of a two week lab exercise for this course. If you are interested, it would be a good exercise to modify your program to model a more realistic network scenario.