University of St Andrews School of Computer Science

CS2003 — Internet and the Web — 2020/21

Exercise 5: Multicast, Queues and Regex
Date: 2020-10-15

As with all lab exercises, this exercise is not directly assessed. It is intended to complement the material presented in lectures and may turn up in the exam. You are encouraged to complete it in your own time if you do not finish during the scheduled exercise class time. *Even more than in previous weeks, you are also encouraged to work with other people in the class to complete the exercises — you can also work on the exercise classes during lab sessions and use the relevant Teams channels to discuss things with each other or to ask for help from a demonstrator.*
The aim of this exercise class is to look at topics from week 5's lectures: multicast, queues and regular expressions. Notes:

- Do not be concerned if it seems like there is a lot of material compared to previous weeks: you have the Independent Learning Week (ILW) to further work through the material. Some of the work will also feature in the week 07 tutorial.
- Some of the material will also be useful for the Net2 coursework, so you are encouraged to work through it.
- The first questions ask you to work with your fellow classmates. I would like you to split into tutorial groups — find your tutorial group channel in the CS2003 Team and then start a meeting so that you can ask in there to find students who are working on the same tasks. If you find that no-one is in your group, then try to join a group in a different channel. If you or anyone else in the group has questions, ask in the General channel and a lecturer or demonstrator can join your meeting.

1 UDP multicast communication — heartbeat

Please examine the code in `CS2003-Examples-wk05/multicast/`, especially `MulticastEndpoint.java`, looking at the use of the Java classes DatagramPacket, and MulticastSocket.
For a MulticastSocket, the IPv4 address is a *Class D address* (see slide 2 of https://studres.cs.st-andrews.ac.uk/CS2003/Lectures/cs2003_net-06_communication_models.pdf). We can choose any address from this range to use for our applications. To ensure that you don't end up choosing the same address as someone else while testing, you can once again use your numerical user ID value

U to create an address of the form $239.x.y.z$. Set the values of x, y, and z as follows:

```
x = 0
y = U DIV 256
z = U MOD 256
```

For instance my userid is 10608 so I would use $239.0.41.112$.
When a multicast address value is used as a destination address, the IPv4 address does not identify a single host, but identifies a group of hosts. The mechanism for identifying members of the group is maintained by network devices, using a protocol called the *Internet Group Management Protocol (IGMP)*, but we will not cover this in detail in this module (see CS3102). It does mean, however, that for group communication, an application has to *join* the group to receive packets sent to that group. An invocation of the `MulticastSocket.joinGroup()` method does just this. There are various other parameters for a `MulticastSocket`, but the only other one we need to concern ourselves with is a port number, which is used as it is for unicast UDP.
The Java class MulticastEndpoint is an abstraction over both the classes MulticastSocket and DatagramPacket, to allow sending to, and receiving from, a multicast group. You will see methods to `join()` and `leave()` a group, and `tx()` and `rx()` to send to and receive from a group.
The `HeartBeat` program implements the basis of a simple *discovery protocol*.

1. Make sure that everyone in your group is logged into lab machines in the same subnet. To do this pick machines that all start with the same prefix, e.g. pc3 or pc5. Do not have some members in pc3 and others in pc5. Remember that the list of lab machines is found here: https://systems.wiki.cs.st-andrews.ac.uk/index.php/Lab_PCs#Remotely_Accessible_Linux_Clients

2. Run the `HeartBeat` program, with one or more classmates also running it on another lab machine. Note and discuss your observations, and discuss the operation of the program.

3. Look at the unicast UDP program `CS2003-Examples-wk03/udp/UdpChat.java`. Modify this program to create a simple multicast-based 'text chat' program. You will need to use both the Configuration and MulticastEndpoint classes to do this. To test your chat program with others in your group, you can use the multicast address / port combination: `239.2.20.`n `/ 2003`, where n is your tutorial group number. If you want to test with the whole class (or at least everyone who is on the same subnet as you), then try using `239.2.20.20 / 2003`.

2 Multi-user TCP server: MultiChat

Please examine the code for the MultiChat application in `CS2003-Examples-wk05/MultiChat/`.
Please also look at pages 16-20 of: https://studres.cs.st-andrews.ac.uk/CS2003/Lectures/cs2003_net-07_server_systems.pdf
Get the application working, with the server running on one lab machine, and multiple clients connecting to it running on other lab machines. You should again do this with other people in your group.

1. How is it that MultiChatSever can listen for incoming connections, as well as relay messages to all connected users?
2. How does the program maintain state about all connected users, and all incoming / outgoing messages?
3. How are the protocol messages defined?
4. The class ChatMessage was designed to show you an example of encoding and decoding of application-level messages in a text-based protocol, by using regular expressions. As such, it contains a step-by-step breakdown of the message format as a regular expression, rather than having one long string for the regular expression. So, of course, if you wish, you can modify the program so that most of the `static private final String` definitions disappear. Of course, to see the components of the regular expression, or the whole regular expression, you can simply modify the code to print the strings.

   Notice how relatively simple the (`encode()` and `decode()` methods are, once the regular expressions are finalised. In this case, the use of *named capture groups* in the regular expressions helps greatly with decoding. (Rather than having a constructor, the (`encode()` and `decode()` methods are *factory methods* for the ChatMessage class.)

3 Regular Expressions

1. Modify `CS2003-Examples-wk05/regex/FindMe.java` so that it can:
   - Match the string “FindMe” only if it appears as a separate word (i.e., it is not a sequence within another string); and
   - Perform that match regardless of whether or not the letters in “FindMe” are uppercase, or lowercase, or a mix of uppercase and lowercase.

2. Compile and run

   `CS2003-Examples-wk05/regex/RegexCSV.java` with `CS2003-usernames-2020.csv` as an argument. This CSV file contains a list of your usernames and your server port numbers (`id -u`). You will see that the program splits the CSV file and prints each line. It also prints out a number of error messages about regexes.

   - Redefine `String r` with a regular expression that has two groups with a comma: the first group should be the username and the second the port number. Recompile and if your regular expression is successful, the output should change.
   - Double-check that your regular expression is sufficiently robust that the first line in the CSV file ("#username,port") does not match. You may already have done this! But if not, a hint is that a username is made up of word characters: see slide 7 of https://studres.cs.st-andrews.ac.uk/CS2003/Lectures/cs2003_net-08_java_regex.pdf

3. Write a program, `EmailAddressCheck`, that takes a single command-line argument, which is an email address, and checks that it is in the correct (expected) format: 'user@domain.name', by using a regular expression. Assume that the only permitted characters in both 'user' and 'domain.name' are:

   - letter ranges: A-Z and a-z
   - digit range: 0-9
   - special characters: + - _

   and that the character '.' is permitted only in the 'domain.name' part, to separate the components of the domain name, as would be expected, e.g. 'tnhh@st-andrews.ac.uk'.

   Why is this regular expression check insufficient to determine if this email address is valid and usable? (For starters, think of the address that you use to contact the CS2003 lecturers)

4 FIFO queues

Please examine the code in `CS2003-Examples-wk05/SimpleStringQueue/`.
Please also look at pages 12-15 of: https://studres.cs.st-andrews.ac.uk/CS2003/Lectures/cs2003_net-07_server_systems.pdf
Compile and run the `SimpleStringQueue`program: `[a]dd` strings to the queue, `[r]emove` strings, and `[p]rint` the queue contents, which includes the position of the tail and the head.

Notice how simple, yet effective this ‘circular’ FIFO is: the interface really just consists of the methods `add()` and `remove()`. The largest method (in terms of lines of code) is the `printQ()` method, which is only for demonstration purposes, and is not related to the operation of the FIFO queue.

Tristan Henderson, with thanks to Saleem Bhatti