University of St Andrews School of Computer Science

CS2003 — Internet and the Web — 2020/21

Exercise 1:	Some Linux network tools and socket programming
Date:	2020-09-xx

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. *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.*
This exercise class will take place on Teams, in particular the "2020/1 Second Year Computer Science" Team. The meeting is entitled "CS2003 Week 1 Exercise Class". If you cannot find it in Microsoft Teams, then you can try this URL: https://teams.microsoft.com/l/meetup-join/19%3ab4ae4a21ba9b40188ecc458330ed99bc%40thread.tacv2/1600076902684?context=%7b%22Tid%22%3a%22f85626cb-0da8-49d3-aa58-64ef678ef01a%22%2c%22Oid%22%3a%22ad2fcb36-eb79-431d-8540-225ffbc71158%22%7d
The aim of this exercise class is to use the Linux command line interface (CLI) and some simple Java programs to explore some basic client-server communications, using some simple Internet protocols. You can access the Linux command line on a lab client by using an SSH client application to log in. This brings up a *shell*, in which you can type commands. The key sequence 'CTRL-C' can be used to stop running programs from the command line. In this document, when you see $ next to a word like this:

```
$ id
```

that is the 'prompt' from the shell, and what is next to it (in this case `id`) is what you need to type.

1 Logging into a CS host

Start by using SSH to log in to one of the lab clients. If you are in the lab, then you should make sure that the machine that you are using is booted into Linux. If you are working remotely, you should use one of the clients listed at https://systems.wiki.cs.st-andrews.ac.uk/index.php/Lab_PCs#Remotely_Accessible_Linux_Clients

You should also read https://systems.wiki.cs.st-andrews.ac.uk/index.php/Working_remotely#Host_service_and_SSH
If you are accessing the CS systems from outside of the University network, you will first need to log in to one of the host servers and from there access one of the lab clients. For instance, you might do

```
$ ssh <username>@<username>.host.cs.st-andrews.ac.uk
$ ssh <username>@pcY-XXX-l.cs.st-andrews.ac.uk
```

(where `<username>` is replaced with your CS username, and `pcY-XXX-l.cs.st-andrews` is one of the lab clients listed at https://systems.wiki.cs.st-andrews.ac.uk/index.php/Lab_PCs#Remotely_Accessible_Linux_Clients)
Alternatively you can set up a proxy or use a jump host as outlined in the wiki (https://systems.wiki.cs.st-andrews.ac.uk/index.php/Working_remotely#SSH_proxy). But for now I suggest you just log in to a host server and then a lab client.

2 Check that your environment is correct

For compiling and running Java code, please use `javac` and `java` from the Linux CLI. Note: There are several Java compilers on the School machines. If you type:

```
$ which java javac
```

you should see:

```
/usr/local/amazon-corretto-11/bin/java
/usr/local/amazon-corretto-11/bin/javac
```

If you do not see this output, then please contact someone from Systems Group (fixit@cs.st-andrews.ac.uk) and explain that your default Java installation appears not to be the Amazon Corretto installation.
For editing code on a remote host, I recommend using a text editor that is installed on the lab machines to edit files there directly. Some possibilities include `nano`, `vim` or `emacs` (in order of difficulty). There are a number of video tutorials on the systems wiki that can help here: https://systems.wiki.cs.st-andrews.ac.uk/index.php/Video_tutorials In particular I recommend that you view the Systems briefing on command-line Java at https://st-andrews.cloud.panopto.eu/Panopto/Pages/Viewer.aspx?id=5de01a4a-63b2-4565-b701-ac3300ce93c5
IMPORTANT: Do not use VS Code or similar “remote ssh” functionality to edit code directly on the host servers. The host servers are not designed for this, and indeed it is poor programming practice, as outlined in Section 11.6. You may wish to do this on the lab clients, but it is not advised.

3 port number for a server

Now you will try to run a server. To do this you will need to choose a port number for running a server process. On the command line, type:

```
$ id
```

and you should see output, where the first part looks like this:

```
uid=DDDDD(uuuu)
```

where $DDDDD$ is a decimal number (should be greater than 10000), and $uuuu$ is your username (login ID). The value $DDDDD$ is the value you should use for a port number when you need to start a server program. Each student should use their own port number to avoid servers clashing.

4 nc

nc (netcat) is a command line interface to the socket API. First, familiarise yourself with the nc command by reading the manual page i.e., `man nc`.
Open two terminal windows on your lab client (or two terminal windows with ssh connections if you are connecting remotely). In one terminal window, start nc listening on a specific port for a connection. Use your port number, for example:

```
$ nc -l <n>
```

You should replace `<n>` with y our own port number (see above). nc is now *listening* on port `<n>` for an incoming connection, just as a server application might. In your second window, connect to the port being listened on using the special loopback address 127.0.0.1:

```
$ nc 127.0.0.1 <n>
```

There should now be a TCP connection between these ports. Anything typed at the second terminal will be sent to the first, and vice versa. After the connection has been set up, nc does not care which side started as a server and which side started as a client: the TCP connection is bi-directional (*duplex*). The connection can be terminated using CTRL-D.

5 netstat

The *netstat* command can be used to list open TCP sockets on the local machine. In a separate terminal window, type:

```
$ netstat -plnt
```

6 Communicating between two different hosts

Use nc to communicate using two different nodes. Open a terminal window and ssh into a different lab client. Once you have logged into a machine you will need to know its IP address and/or name. To find out the name and IPv4 address of a Linux system use the commands:

```
$ hostname -f
```

and:

```
$ hostname -i
```

7 Daytime protocol (RFC867)

Look at the Daytime protocol standard (RFC867 – https://tools.ietf.org/html/rfc867). Select a server from the list of time servers at http://tf.nist.gov/tf-cgi/servers.cgi, or alternatively use minervamedia.cs.st-andrews.ac.uk. Use nc to access a time server on the Daytime protocol port 13.

8 ping

The result returned from the Daytime protocol is not completely accurate due to the delay between the client and the server. This can be estimated from the *round trip time (RTT)* returned by the `ping` command e.g.:

```
$ ping ntp1.ja.net
```

or:

```
$ ping minervamedia.cs.st-andrews.ac.uk
```

Are the RTT times shown always the same? If not, why do you think they are different?

9 echo and chargen

The *Echo* (RFC862) and *Chargen* (RFC864) protocols date from the same early Internet period as the Daytime protocol. Use nc to connect to the servers for these protocols at minervamedia.cs.st-andrews.ac.uk to see their function.

10 Daytime protocol client

Copy the *DayTimeClient.java* client code from studres to your own home directory. Examine the source code, compile and run the program.
For instance, you might do

```
$ mkdir ~/CS2003
$ cp /cs/studres/CS2003/Examples/CS2003-Examples-wk01/DayTimeClient.java
  ~/CS2003
$ cd ~/CS2003
$ nano DayTimeClient.java
$ javac DayTimeClient.java
$ java DayTimeClient
```

(to exit the `nano` text editor you should type CTRL-C — there is a menu at the bottom of the screen with some common shortcuts. For more help please view the video tutorial mentioned at the start of this document.)

11 A simple server

11.1 Compile and set-up

Choose a port number as outlined at the beginning of this exercise sheet. Then edit `TcpServerSimpleNB.java` and assign this value to the variable `port_`. (NB: as you are doing this, have a think about compile-time versus run-time configuration. Does it make sense to have to recompile a program to change a port number?)
Use `javac` to compile the files `TcpClientSimpleNB.java` and `TcpServerSimpleNB.java`. Then connect to an additional lab client and do the same so that you can run a client on one machine and a server on another.

11.2 Running the server

On one of the machines that we will call hostA, start the server, for example:

```
hostA $ java TcpServerSimpleNB
```

Make sure that the program starts (a message will appear).
From the other machine (we will call it hostB), connect to the server using `nc`, e.g.:

```
hostB $ nc hostA.cs.st-andrews.ac.uk <n>
```

You should use your own port number in place of `<n>`. Then, type something at hostB and observe the behaviour. (What you typed should be sent back to you by the server.)
The server will quit.

11.3 Running the client

Start the server again on hostA.
Then, from hostB, use the TCP client code to connect:

```
hostB $ java TcpClientSimpleNB hostA.cs.st-andrews.ac.uk <n>
```

You should see similar behaviour as that to using `nc`. (There will be some additional text printed by the program.)

11.4 Things to note

11.4.1 Addresses and port numbers

At hostA, check the server is listening on its port number by again using netstat, e.g.:

```
    $ netstat -plnt
```

Also, note the address information that is printed at the server (you will need to look at the code to see what is printed). Does it seem strange? (We will look at this later in the course, but for now, look at the documentation for ServerSocket.)

11.4.2 Number of bytes sent/received

When running `TCPClientSimpleNB`, what do you notice about the number of bytes you type, and the number of bytes sent/received? In the code we have looked at so far, we "wrapped" the raw byte-stream From the TCP connection in the Java abstractions `PrintWriter` and `BufferedReader`. These treat the byte-stream as consisting of printable characters. However, the code in

`TcpClientSimpleNB.java` and `TcpServerSimpleNB.java` access the Input-Stream and OutputStream from the Socket obejct, which give access to the “raw” byte-stream that forms the content of the TCP segments that are transmitted.

11.5 Looking at the code

Notice the use of the ServerSocket in TcpServerSimpleNB.java. Recall that this is a “door” on which the server listens for incoming connection requests. The `accept()` method accepts incoming connection requests, and assigns a new Socket object to the connection. The ServerSocket is still available to accept other connections, if we reorganised our code to do that. Also, notice the use of the `soTimeout()` method. Look this up in the Java documentation. This makes the API for the connection non-blocking – the Socket operations can timeout, e.g. reading on a socket times out if there is nothing to read. This allows your program to do other things and remain “stuck” on that method call until something happens. This is purely an API issue, and is nothing to do with the operation of the TCP protocol.

11.6 Running code on a host server

Up till now you have been working on the lab clients. For all of the practicals we would like your code to run on the host servers. This does not mean that you should edit your code on the host servers. Instead, you should develop on a lab client (or even your local machine), and then copy your code to the host servers to run there. This is generally good practice; you should develop code in a test environment and then *deploy* it to your production environment.
If you have developed on a lab client then there is very little that you need to do to copy your code to a host server. Your network home directory is available at `/cs/home/<username>` from both lab clients and host servers. So if you have been developing in your local home directory, you could do something similar to:

```
$ mkdir -p /cs/home/<username>/CS2003/Examples-wk01
$ cp ~/CS2003/TCPClientSimpleNB.java
    /cs/home/<username>/CS2003/Examples-wk01/
$ ssh <username>@<username>.host.cs.st-andrews.ac.uk
$ cd /cs/home/<username>/CS2003/Examples-wk01/
$ javac TCPClientSimpleNB.java
$ java TCPClientSimpleNB
```

(where `<username>` should be replaced with your CS username)

If you are developing on a local machine, then you can use SFTP to copy files to your network home directory (see https://systems.wiki.cs.st-andrews.ac.uk/index.php/Working_remotely#Accessing_your_home_directory) and then use ssh to log in to a host server, compile and run your code.

11.7 Modifying the code

See if you can modify the code so that:

- The server becomes an "echo" server: it keeps sending strings back to the client, as long as the client sends it things and is connected to the server.
- The server does not quit after a client has disconnected, but remains waiting for another client. Hint: do not close the ServerSocket, and think about how `accept()` can be used if the ServerSocket is made non-blocking.

12 Some common Linux commands

A good online resource for such things is https://ss64.com.

`cd <path>`	change directory
`cd ..`	change to parent directory
`cd`	change to home directory
`pwd`	print working (current) directory
`ls`	show/list contents of current directory
`ls -l`	long listing
`ls -t`	list in order of time created (reverse chronological order)
`ls -ld`	list this directory entry
`ls -R`	recursively list directory tree from this point
`cp`	copy file or directory
`rm`	remove (delete) a file
`rmdir`	remove (delete) a directory
`mv`	rename (move) a file or directory
`mkdir`	create a new directory

`cat <filename>`	concatenate file (to screen)
`less <filename>`	show file (type *q* to quit)
`man <command>`	show manual page for this command
`more <filename>`	show file (type *q* to quit)
`nano <filename>`	a simple text editor

`ps`	process I am running
`ps -elf`	all process running
`ps -aux`	all proccess running
`kill <process id>`	terminate process
`netstat -plnt`	shows listening tcp sockets on the local machine

`<command> &`	run command in the background
`bg <command>`	run command in the background
`fg`	run in / bring to foreground (see man page)
`jobs`	show / list my jobs (typically processes in background)
`kill <job number>`	terminate job (kill process)

For all of these commands you can find more help in the `man` pages; for instance `man mkdir` might help you understand why I used `mkdir -p` in one of the examples above.

Tristan Henderson, with thanks to Colin Allison, Saleem Bhatti and Ryo Yanagida