UNIVERSITY OF Hull

DEPARTMENT OF COMPUTER SCIENCE
ASSESSMENT DESCRIPTION 2019/20
(EXAM TESTS WORTH <= 15% AND COURSEWORK)

MODULE DETAILS:

Module Number:	500081	Semester:	2
Module Title:	Networking and User Interface Design		
Lecturer:	Eur. Ing. Brian Tompsett / Dr Bing Wang		

COURSEWORK DETAILS:

Assessment Number:	1	of	1		
Title of Assessment:	Client and Server with User Interface				
Format:	Program				
Method of Working:	Individual				
Workload Guidance:	Typically, you should expect to spend between	60	and	120	hours on this assessment
Length of Submission:	This assessment should be **no** more than: *(over length submissions **will be** penalised as per University policy)*	N/A - coding exercise			

PUBLICATION:

Date of issue:	29th January 2020

SUBMISSION:

ONE copy of this assessment should be handed in via:	Canvas		If Other (state method)	
Time and date for submission:	**Time**	14:00	**Date**	18th March 2020
If **multiple hand–ins** please provide details*:*	There are numerous formative assessments in the laboratories that form part of this assessment process and permit feedback prior to final submission.			
Will submission be scanned via TurnitIn?	No	If this requires a separate TurnItIn submission, please provide instructions:		
*Late submissions **will be** penalised as per university policy*				

The assessment must be submitted **no later** than the time and date shown above, unless an extension has been authorised on a *Request for an Extension for an Assessment* :
search 'student forms' on https://share.hull.ac.uk.
Canvas allows multiple submissions: only the **last** assessment submitted will be marked and if submitted after the coursework deadline late penalties will be applied.

MARKING:

Marking will be by:	Student Name

ASSESSMENT:

The assessment is marked out of:	100	and is worth	50	% of the module marks
N.B If multiple hand-ins please indicate the marks and % apportioned to each stage above (i.e. Stage 1 – 50, Stage 2 – 50). It is these marks that will be presented to the exam board.				

ASSESSMENT STRATEGY AND LEARNING OUTCOMES:

The overall assessment strategy is designed to evaluate the student's achievement of the module learning outcomes, and is subdivided as follows:

LO	Learning Outcome	Method of Assessment *{e.g. report, demo}*
2	*Select from a range of suggested approaches and techniques to design interactive systems in the context of the user's task.*	Program
3	*Explain, with comprehension, key concepts and principles of networking and network architectures.*	Program

Assessment Criteria	Contributes to Learning Outcome	Mark
Part 1 (The maximum possible mark of 22 is capped at 20)	2,3	
Client Operation		32
Server Operation		32
Network Error Handling		8
Coding Style, Design and Documentation Quality		8
Optional Features		8
Part 2	2, 3	
Server: User Interface capability		6
Server: Ease of Use, Nice design		4
Client: User Interface capability		6
Client: Ease of Use, Nice Design		4

FEEDBACK

Feedback will be given via:	Canvas	Feedback will be given via:	Canvas
Exemption (staff to explain why)			

This assessment is set in the context of the learning outcomes for the module and does not by itself constitute a definitive specification of the assessment. If you are in any doubt as to the relationship between what you have been asked to do and the module content you should take this matter up with the member of staff who set the assessment as soon as possible.

You are advised to read the **NOTES** regarding late penalties, over-length assignments, unfair

means and quality assurance in your student handbook, which is available on Canvas - https://canvas.hull.ac.uk/courses/17835/files/folder/Student-Handbooks-and-Guides.

In particular, please be aware that:

- Up to and including 24 hours after the deadline, a penalty of 10%
- More than 24 hours and up to and including 7 days after the deadline; either a penalty of 10% or the mark awarded is reduced to the pass mark, **whichever results in the lower mark**
- More than 7 days after the deadline, a mark of zero is awarded.
- The overlength penalty applies to your written report (which includes bullet points, and lists of text. It does not include contents page, graphs, data tables and appendices). 10-20% over the word count incurs a penalty of 10%. Your mark will be awarded zero if you exceed the word count by more than 20%.

Please be reminded that you are responsible for reading the University Code of Practice on Academic Misconduct through the Assessment section of the Quality Handbook (via the SharePoint site). This govern all forms of illegitimate academic conduct which may be described as cheating, including plagiarism. The term ‘academic misconduct’ is used in the regulations to indicate that a very wide range of behaviour is punishable.

In case of any subsequent dispute, query, or appeal regarding your coursework, you are reminded that it is your responsibility, not the Department’s, to produce the assignment in question.

Assignment Details

Part 1

The coursework requires you to implement two C# Windows console applications which can be invoked by other programs; one is a client and the other is the server. These applications will eventually be components in a larger software system and their interfaces and names need to be implemented according to this specification to permit the correct interoperation between components. The client shall be called `location`, and the server called `locationserver`. They should use TCP sockets by means of the appropriate classes in `System.Net` and `System.Net.Sockets`. The task is to implement a simple student locating facility. Students will normally be identified by their computer login name, but the server need not be specific to Hull University, so any command argument string can be a student identifier. The client will simply use the Microsoft Command Prompt interface to run. The client and server will communicate using simplified forms of the internet whois protocol and the HTTP protocol as described below. The work can be implemented in stages and lectures will have the information needed to get you started. The second coursework assignment will build on the features of this one, so the more you are able to complete of the features of this program, the more helpful you will find it for later.

BASIC CLIENT INTERFACE SPECIFICATION

The client (which must be called `location`) can have two arguments, a user name and a string giving location information. In the examples below, the text `G:\500081\>` represents the command prompt.

For example:

```
G:\500081\> location cssbct
cssbct is in RB-336
```

In this example, `location` is the name of the executable of the client program, which is given one argument (`cssbct`) and the client shows the response "`cssbct is in RB-336`". This string is composed by the client from the argument supplied and the data returned by the location server. The client must output only this for a successful location lookup.

```
G:\500081\> location cssbct "in RB-310 for a meeting"
cssbct location changed to be in RB-310 for a meeting
```

In this example `location` is given two arguments, the first `cssbct` is the user name being updated, and the second is the string `"in RB-310 for a meeting"`. The quote characters (") are not passed to the client as part of the second argument, they are used to indicate to the windows command interpreter that the sequence of words separated by spaces are part of a single argument and are not five consecutive arguments to the client.

The response output by the client shows that the server has acknowledged the update was successful and is composed from the arguments to the client.

```
G:\500081\> location cssbct
cssbct is in RB-310 for a meeting
```

This example shows an enquiry to the server after making the previous update, which illustrates the new location is returned for further queries. It also allows you to determine the specified syntax of replies.

In summary these examples show first the call of the client indicating the current location of the user `cssbct`. The second was used to change the location and the third showed the changed response.

Protocol for Coursework

INTRODUCTION

Your coursework must implement the following protocol for communication between the client and server; this is based on a subset of features from the internet protocols for whois (RFC3912), HTTP/0.9 (w3.org), HTTP/1.0 (RFC1945) and HTTP/1.1 (RFC2616). Students should not invent their own protocols as the aim of the coursework is to demonstrate interoperability between different implementations as well as existing clients and servers. A working server that implements this protocol is provided by the lecturer on the machine `whois.net.dcs.hull.ac.uk`. You can initially develop a client using the lecturer's server, and later develop your server when you are happy with the correctness of your client. It is requirement of this coursework that your client can communicate with the reference server in addition to your own server in order to achieve successful client operation for assessment.

You will encode your data to be transferred between client and server as US-ASCII strings terminated by US-ASCII characters 13 then 10 (carriage return then line feed). There are other more readily available data encoding options in C# such as Unicode and object serialization, but using US-ASCII makes our communication easily understandable by other non-C# applications and existing clients and servers. Fortunately the default action of network sockets is to do this anyway, so no special coding is normally needed.

The server is prohibited from using file storage when running unless the optional `-f` and `-l` features are implemented to specify the name and location of the files to be used.

PROTOCOL

1. The server waits for clients to contact it.
2. Clients should connect and communicate with the server via sockets and TCP packets.
3. Our protocol will operate over port 43, normally used by the internet whois protocol.
4. The protocol has four styles of commands representing the whois notation, and the HTTP 0.9, 1.0 and 1.1 styles.
5. The protocol contains two types of requests from the client to the server. These are lookup and update.

Whois style requests

1. A request from the client to the server for the server to query the database for a person's location looks like this:

   ```
   <name><CR><LF>
   ```

   Where <name> is the name of the person whose location the client is requesting.

2. A request from the client to the server for the server to update the database with a new location for a person looks like this:

```
<name><space><location><CR><LF>
```

Where: <name> is the name of the person whose location the client wants to update.
<location> is the new location for that person.

3. When the client requests the server to query the database for a person's location and the server finds an entry in the database for that person the server responds like this:

```
<location><CR><LF>
      [drops connection]
```

Where: <location> is the location if found in the database for the person.

4. When the client requests the server to query the database for a person's location but the server is unable to find an entry in the database for that person the server responds like this:

```
ERROR: no entries found<CR><LF>
      [drops connection]
```

5. When the client sends a request to the server to update the database with a new location for a person the server responds like this:

```
OK<CR><LF>
      [drops connection]
```

HTTP 0.9 style requests

1. A request from the client to the server for the server to query the database for a person's location looks like this:

```
GET<space>/<name><CR><LF>
```

Where <name> is the name of the person whose location the client is requesting .

2. A request from the client to the server for the server to update the database with a new location for a person looks like this:

```
PUT<space>/<name><CR><LF>
<CR><LF>
<location><CR><LF>
```

Where: <name> is the name of the person whose location the client wants to update and <location> is the new location for that person.

3. When the client requests the server to query the database for a person's location and the server finds an entry in the database for that person the server responds like this:

```
HTTP/0.9<space>200<space>OK<CR><LF>
Content-Type:<space>text/plain<CR><LF>
<CR><LF>
<location><CR><LF>
      [drops connection]
```

Where: <location> is the location if found in the database for the person.

4. When the client requests the server to query the database for a person's location but the server is unable to find an entry in the database for that person the server responds like this:

```
HTTP/0.9<space>404<space>Not<space>Found<CR><LF>
Content-Type:<space>text/plain<CR><LF>
<CR><LF>
     [drops connection]
```

5. When the client sends a request to the server to update the database with a new location for a person the server responds like this:

```
HTTP/0.9<space>200<space>OK<CR><LF>
Content-Type:<space>text/plain<CR><LF>
<CR><LF>
     [drops connection]
```

HTTP 1.0 style requests

1. A request from the client to the server for the server to query the database for a person's location looks like this:

```
GET<space>/?<name><space>HTTP/1.0<CR><LF>
<optional header lines><CR><LF>
```

   Where <name> is the name of the person whose location the client is requesting . The <optional header lines> are not required to be sent by the clients but other programs using HTTP 1.0 protocol may send them and they should be read and skipped by the server. A blank line marks the end of the header lines.

2. A request from the client to the server for the server to update the database with a new location for a person looks like this:

```
POST<space>/<name><space>HTTP/1.0<CR><LF>
Content-Length:<space><length><CR><LF>
<optional header lines><CR><LF>
<location>
```

   Where: <name> is the name of the person whose location the client wants to update and <location> is the new location for that person. The <optional header lines> are not required to be sent by the clients but other programs using HTTP 1.0 protocol may send them and they should be read and skipped by the server. A blank line marks the end of the header lines. The <length> is the count of the number of characters in the <location>.

3. When the client requests the server to query the database for a person's location and the server finds an entry in the database for that person the server responds like this:

```
HTTP/1.0<space>200<space>OK<CR><LF>
Content-Type:<space>text/plain<CR><LF>
<CR><LF>
<location><CR><LF>
     [drops connection]
```

   Where: <location> is the location if found in the database for the person.

4. When the client requests the server to query the database for a person's location but the server is unable to find an entry in the database for that person the server responds like this:

```
HTTP/1.0<space>404<space>Not<space>Found<CR><LF>
Content-Type:<space>text/plain<CR><LF>
<CR><LF>
     [drops connection]
```

5. When the client sends a request to the server to update the database with a new location for a person the server responds like this:

```
HTTP/1.0<space>200<space>OK<CR><LF>
Content-Type:<space>text/plain<CR><LF>
<CR><LF>
      [drops connection]
```

HTTP 1.1 style requests

1. A request from the client to the server for the server to query the database for a person's location looks like this:

```
GET<space>/?name=<name><space>HTTP/1.1<CR><LF>
Host:<space><hostname><CR><LF>
<optional header lines><CR><LF>
```

   Where <name> is the name of the person whose location the client is requesting and <hostname> is the host name of the server. The <optional header lines> are not required to be sent by the clients but other programs using HTTP 1.1 protocol may send them and they should be read and skipped by the server. The <hostname> line can also be ignored by the server. A blank line marks the end of the header lines.

2. A request from the client to the server for the server to update the database with a new location for a person looks like this:

```
POST<space>/<space>HTTP/1.1<CR><LF>
Host:<space><hostname><CR><LF>
Content-Length:<space><length><CR><LF>
<optional header lines><CR><LF>
name=<name>&location=<location>
```

   Where: <name> is the name of the person whose location the client wants to update and <location> is the new location for that person and <hostname> is the host name of the server.. The <optional header lines> are not required to be sent by the clients but other programs using HTTP 1.1 protocol may send them and they should be read and skipped by the server. The <hostname> line can also be ignored by the server. A blank line marks the end of the header lines. The <length> is the count of the number of characters in the "`name=<name>&location=<location>`" string.

3. When the client requests the server to query the database for a person's location and the server finds an entry in the database for that person the server responds like this:

```
HTTP/1.1<space>200<space>OK<CR><LF>
Content-Type:<space>text/plain<CR><LF>
<optional header lines><CR><LF>
<location><CR><LF>
      [drops connection]
```

   Where: <location> is the location if found in the database for the person.

4. When the client requests the server to query the database for a person's location but the server is unable to find an entry in the database for that person the server responds like this:

```
HTTP/1.1<space>404<space>Not<space>Found<CR><LF>
Content-Type:<space>text/plain<CR><LF>
<optional header lines><CR><LF>
      [drops connection]
```

5. When the client sends a request to the server to update the database with a new location for a person the server responds like this:

```
HTTP/1.1<space>200<space>OK<CR><LF>
Content-Type:<space>text/plain<CR><LF>
<optional header lines><CR><LF>
     [drops connection]
```

In all examples above <CR> is US-ASCII character number 13, the carriage return character, and <LF> is US-ASCII character number 10, the line feed character and <space> is the US-ASCII space character. We are using both CR and LF together as our line terminator. The notation `[drops connection]` is used to indicate that the server closes the connection to the client at that point, without waiting for any response. In all cases both the client and server should give up and drop connection (timeout) if they are left waiting more than 1 second (1000 milliseconds).

The server should be able to handle enquiries from a number of clients simultaneously, however a client only needs to handle a single enquiry at a time.

ADDITIONAL CLIENT INTERFACE FEATURES

When initially constructed, the client, could have the address of the server hardwired into the C# code (such as whois.net.dcs.hull.ac.uk or localhost). This is not very useful, and the delivered client will need a parameter to specify the server to be used. The default address would be whois.net.dcs.hull.ac.uk if the argument is not used. For example:

```
G:\500081\> location -h 150.237.92.99 cssbct
cssbct is in RB-321
G:\500081\> location -h somepc.dcs.hull.ac.uk cssbct "changed room"
cssbct location changed to be changed room
```

Note that the sequence "`-h <servername>`" could occur at any point in the argument list, so it could be either before the username or after.

It will also be useful to be able to specify the port number to be used, to enable the client to connect to other servers on other ports. The `-p` flag will be used for this:

```
G:\500081\> location cssbct -p 43
```

It is also necessary to specify which protocol request form is to be used. By default the `whois` protocol form should be used. If an HTTP format is required then the `-h0, -h1, -h9` flags can be used:

```
G:\500081\> location cssbct -h1
```

When using these `-h1` means HTTP/1.1, `-h0` means HTTP/1.0 and `-h9` means HTTP/0.9 styles of server request.

OPTIONAL FEATURES

Implementing these features is not essential for the coursework, but may be useful in developing your client or server. They can earn extra marks in the marking scheme, but no one can get more than 100%, even if all the options are implemented.

Parameters of `-t` to change the timeout period (and setting the timeout period to zero could disable timeouts) and `-d` to enable any debugging might be useful additions to the client and the server.

The server could, at a minimum, do its job silently. It would be helpful, both for you to debug it, and for assessment, if it reported activity, either to the command window, or to a specially designed administration interface. For example:

```
G:\500081\> locationserver
request for cssbct
replied in RB-321
```

The more information that is recorded the better, and a good server should keep log files:

```
200.4.239.13 - - [28/Dec/2003:03:14:56 +0000] "GET cssbct" OK
150.237.92.99 - - [28/Dec/2003:03:15:23 +0000] "PUT cssbct some room" OK
150.237.92.99 - - [28/Dec/2003:03:19:24 +0000] "GET cshwvv" UNKNOWN
```

This example is shown in "Common Log Format".

The supplied server whois.net.dcs.hull.ac.uk keeps a log of all activity, which can be checked for debugging your client.

If your server is able to handle more than one simultaneous client enquiry at a time, you need to be careful that you do not attempt to write more than one log entry to the file at the same time. Further, you should ensure that if the log file is not writable that the server does not fail. When the programs are assessed, the server will not necessarily have write permissions to the machine to save a log. You should also make the destination of the log file an optional argument to the server to allow for different machine file structures, such as:

```
G:\500081\> locationserver -l c:\log\logfile.txt
```

It is also useful to be able to save the server database when it exits to some form of file, and re-load it when it restarts. You could use the option `-f` for this feature:

```
G:\500081\> locationserver -f G:\500081\locations.txt
```

If you have implemented all your client and server features properly, then it will be possible to use your client to talk to a web server, or get a web browser to talk to your client. For example:

```
G\500081\> location -h www.hull.ac.uk -p 80 -h9 index.html
```

might fetch a web page. You can also try using Internet Explorer on the URL `http://localhost:43/cssbct` to do a database lookup from your server.

There are also further features of the standard protocols that could be implemented, but are not necessary for this assignment. Students may wish to read those standards.

TIMESCALE FOR COMPLETION OF WORK

This timetable shows how the extensive coursework assessment can be completed, by making regular progress. Initially you would want to call your programs manually, but later you should use the test scripts which exercise invoking your code from other programs and provide basic

tests. These test scripts are located on the university network drive at `U:\Computer Science\500081\Labs\.`

Step 0. The first laboratory task introduced you to the sample client used in the lectures. You may need to check about the method `stream.flush` which is not included in the example code.

Step 1. Develop and complete your client to perform a whois request by having it communicate with the working server provided by your lecturer (whois.net.dcs.hull.ac.uk) or an external third party internet whois server. This should be based on the code template given in the lecture Powerpoint. It is often sensible to be sure of the correctness of your client before proceeding to develop your server. It might be a good idea to start using the SVN server to keep track of the development of your solution.

Step 2. Develop a minimal server based on the template in the Powerpoint slides that uses the whois protocol, using your own client to interface with it. You would need to write code to store the person locations in the server memory, and you might find the classes for `Hashtable` or `Dictionary` particularly useful for this. We suggest that you initially develop a server that does not use threads, to ensure the protocol and communications work properly. This can then be augmented to handle multiple clients and multiple protocols later.

Step 3. Extend your basic working client to include the command line arguments of `-h`, `-p`. Add the HTTP protocols and the arguments `-h0`, `-h1`, `-h9` and test against a web server.

Step 4. Extend your basic server to recognise HTTP protocol requests and test with your client and later with a web browser.

Step 5. Upgrade your server to include handling of multiple threads so concurrent enquiries from multiple clients is possible. If you find this difficult to achieve, remember to retain a copy of your working non-threaded server to hand-in for assessment should you not have a working threaded version. Do not damage your only working server when trying to add more capability, or you risk losing all the marks you earned to this point. You might find that using SVN from the first week has become valuable is assisting you in recovering earlier versions!

Step 6. Add any remaining optional features to your client and server, such as logging, and finish testing. Now prepare to add user interface features to the client and server for part2.

Part 2

This requires you to implement a C# Windows Applications for both your client and server. Your client and server should be able to operate as specified in part 1 and the additional user interface should not change its operation or existing user interface, but should only add more functionality.

The client, if launched with no arguments, should now open a Windows interface that permits the user to specify the same operations as possible for the command line but with a, perhaps, better user interface style and design. You should design that user interface and decide what features it should contain based on your learning on the course.

The server, when launched with an argument of –w should open a windowed interface that permits an operator to control the functions of the server. If launched with no arguments it should operate as previously specified in part 1. If –w is used in conjunction with other arguments, these arguments indicate pre-set values that should be shown in the windowed user interface, but the operator should be able to change them using the interface you implement.

Note that the client and server each have a different mechanism for launching the windowed interface. In the certificate stage programming modules (last year) you were set laboratory task in creating windows interfaces using either WPF or Windows Forms. You should refer back to the work from last year to help you implement you chosen interface. The lectures on User Interface Design will not teach you this programming, but focus on issues of design and usability which will occur after this assessment is completed.

This part of the assignment will then permit you to reflect on what makes a good user interface and support your learning of user interface design issues.

DELIVERABLES

You must submit the following:

> Formative assessment submissions for the laboratory tasks. Although these do not accrue marks, the evidence submitted can be used for marking should there be a problem with the running of the final submitted solutions.
>
> An electronic copy of your source code, compiled class files of both applications as two distinct visual studio projects compiled into a ZIP file. This must only be a ZIP file and has the file type ZIP. No other form of archive or compression should be used (i.e. no RAR or 7z etc). The submission should also contain a file, not within a folder, named `readme.txt` which contains the summary of the zip file contents.
>
> The client and server code may be contained within folders, but does not have to be. If the client code is delivered in a folder, perhaps as a Visual Studio project, then this folder should be called `location`. If the server code is delivered in a folder, perhaps as a Visual Studio project, then this folder should be called `locationserver`. There must be only one copy of the client and server source and executable contained within the zip file submitted, otherwise it would not be clear which one should be marked.