Module	Computer Fundamentals / COMP1027 (CSF) / Semester 1
Module Convenor(s)	Tissa Chandesa

Assessment Name	Coursework 4	**Weight**	22.5%

Description and Deliverable(s)

The coursework (details below) focuses on **Socket Programming** of this module. It is an **INDIVIDUAL WORK.**

Preliminaries

- In this part of the coursework, a server implementation in C using WinSock API (Application Program Interface) is provided. You are required to test out functionality using a Telnet client, understand how the program works. Then carry out the required tasks in accordance with the specifications listed below.
- **For Mac users**, please use the alternative approach explained in the laboratory session(s). Please note that MacOS does not support WinSock API and Putty. Also, you **need** to create a separate Client.c to perform the requested task.

Files to Download

1. Download the relevant file(s) from the "Coursework 4" area on Moodle.

Important Notes – Windows User

1. If you do not have a standard C compiler, you can install MinGW (or any equivalent environment to use the *gcc* compiler or equivalent).
2. Test the provided server:
   a) Download, compile and run Server.c in one command window.
   b) In a separate window, use **puttytel** (or an equivalent Telnet client) to connect to the server program.
   c) Observe the execution.
   d) Explore the code and understand how it relates to the flowcharts on slide 29 from the lecture.

Important Notes – MacOs/Linux User

1. Download and install MS Visual Code or any IDE that you are comfortable with.
2. Click on this video which teaches you how to install *gcc*. Follow the instructions given accordingly. (Homebrew: https://brew.sh/)
3. Once complete, test your connection by creating a Client.c (*you may use this link as a guide if you have not done so during the lab sessions.*). Compile and run the provided Server.c using your MS Visual Code. If everything is set-up properly, you should see the corresponding result (on the server side) as you input on the Client side.
4. Observe the execution.
5. Explore the code and understand how it relates to the flowchart on slide 28 from the lecture.

Important Notes

1. Your program should be clearly structured and have proper comments.
2. **No External libraries** to be used (**unless approved beforehand**) and included within the submission. This is important to avoid compilation failure (and subsequently **loss of marks**).
3. Please make sure to use the **correct argumentative channel** (e.g., **argv[0]**, **argv[1]**). The use of other argumentative channels will result in your codes being **unable to run**, subsequently **affecting your marks for that task**.

Task 1:

Modify the original server code so that its output message is in ***reverse*** order to the client's input message and all alphabets converted to ***lowercase***. Please note that your input (in the client side) MUST BE IN UPPERCASE.

SUBMISSION: The modified server should be saved as **Server1.c**.

Task 2:

Improve the server code (**from Server1.c**) to handle **multiple requests** (instead of one) from the same Putty (*or equivalent Telnet client*) session. For each new connection, the server window should display:

1. The **IP address** of the client,
2. The **port number** targeted by the client's request.

For each message received, the server should **display the length** of the message.

Finally, the modified server code should close the connection and terminate if the client input the message: "**exit server**".

SUBMISSION: The modified server should be saved as **Server2.c**.

Task 3:

Instead of using Putty, develop a client program to read characters from the console (input by the user). If the characters "exit client" are read, the client should terminate. Otherwise send the inputs to the server. The server response should be displayed in the client's window.

SUBMISSION: The developed client should be saved as **EchoClient.c**.

Task 4:

Modify the server code (**from Server2.c**) to handle specific **COMMANDS** sent from the client. A particular command, in this task, is "**DATE**".

Type "**DATE**" in the Client, which will be sent to the Server, the server should respond with the ***Current Day, Date and Time*** (in its *Current Time-Zone*).

1. This should be presented as a single line, that is terminated with a carriage return (ASCII code 13) and line feed (ASCII code 10).
2. Example of output: *Wed Nov 30 17:00:54 2022 GMT*

If any other characters (i.e., not command), the server will handle them as in task 1 and 2.

SUBMISSION: The modified server should be saved as **Server3.c**.

Task 5:

Modify the server code (**from Server3.c**) to handle additional commands sent from the client. A particular command, in this task, is "**TIME**" and a code of a time-zone.

Typing "**TIME**" in the Client, which will be sent to the Server, the server should respond with only the **Current Time** (in the **Current Time-Zone**, which is in "*Malaysia*" time).

Typing, for example, "**TIME GMT**" in the Client, which will be sent to the Server, the server should respond with the Current Time (in the **GMT Time-Zone**). List of

acceptable time-zones and the associated offset from GMT (in hours), is available in Table 1, below. *Please assume that the server is based in Malaysia, hence, you would need to modify the offsets in accordance with the current time-zone, so that the returned times are correct.*

If any other Time-Zone codes provided, the server should respond with "**ERROR"** string.

Table 1. Time Zones

Time-zone Command	Offset from GMT (in hours)
PST	-8
MST	-7
CST	-6
EST	-5
GMT	0
CET	+1
MSK	+3
JST	+9
AEDT	+11

Expansion of the above-mentioned abbreviation can be found here.

SUBMISSION: The modified server should be saved as **Server4.c**.

Final Submission Instructions

You are required to submit **ALL** the required tasks, as per the coursework instructions above (see the "**SUBMISSION**" line under each task).

Your comments within the code files will serve as your brief reporting on the tasks.

For **ALL tasks**, each student should submit his/her own files (*as listed* below). **We reserve the right to ask all/some students to explain any/all of your submitted work at any time. Failure to explain it properly could affect YOUR mark.**

Adherence to the files' naming rule (i.e., Server1.c **IS NOT THE SAME AS** *server1.c* and EchoClient.c **IS NOT THE SAME AS** *Echoclient.c*). Changing the letter(s) of submitted file(s) will result in you **being awarded a 0%** on that corresponding task. So, please be careful!

All students **MUST** submit on Moodle a **ZIP** archive file. **Any other archive file formats WILL result in a penalty of a 5% deduction of your overall mark.** Within your ZIP archive, it should contain all your individual worked files. Name that ZIP file as **YYY-CSF-CW4-XXXXXXXX.zip**, where the "XXXXXXXX" is the 8-digits of your student ID and "YYY" implies whether you are using WIN (WindowsOS) or MAC (MacOS) platform to build the codes.

All the files should have the (**EXACT**) file names as detailed below:

Windows User	Mac Users
Server1.c	Client.c
Server2.c	Server1.c
EchoClient.c	Server2.c
Server3.c	EchoClient.c
Server4.c	Server3.c
-	Server4.c
CW 4 Marking Sheet.docx	

On Moodle, please click on the "**Coursework 4**" link within the "Coursework" section to perform **your** submission.

Release Date	Wednesday, 30th November 2022
Submission Date	Monday, 19th December 2022, by 11:59pm
Late Policy (University of Nottingham default will apply, if blank)	Work submitted after the deadline will be subject to a penalty of 5 marks (the standard 5% absolute) for each late **working day** out of the total 100 marks.
Feedback Mechanism and Date	Marks and written individual feedback will be returned via Moodle within the week commencing 24 January 2023.
Assessment Criteria	**Assessment Breakdown:**

Components	Marks Distribution
Task 1: Server1	
- Outputs in lowercase	5
- Reverse order outputs	5
Task 2: Server2	
- Multiple requests handling	10
- Display of IP address	10
- Display the port number	10
- Display the message length	10
- Implement the command “exit server”	5
Task 3: EchoClient	
- Client code, meeting specifications (as in the CW sheet), successfully compile & run.	10
- Implement the command “exit client”	5
Task 4: Server3	
- Server3, handling the "DATE" command (as in the CW sheet), successfully compile & run.	10
Task 5: Server4	
- Server4, handling the "TIME" command (and its parameters, as per the CW sheet), successfully compile & run.	20