Engineering Assignment Coversheet

Please note that you:

- Must keep a full copy of your submission for this assignment
- Must staple this assignment
- Must NOT use binders or plastic folders except for large assignments

Student Number(s)

850509
872301

Group Code (if applicable):

Assignment Title:	Workshop4 - Network and Web-based Communications
Subject Number:	ELEN90061
Subject Name:	Communication Networks
Student Name:	Yi Jian, Yue Chang
Lecturer/Tutor:	Tansu Alpcan
Due Date:	26/Oct/2019

For Late Assignments Only

Has an extension been granted? Yes / No (circle)

A per-day late penalty may apply if you submit this assignment after the due date/extension. Please check with your Department/coordinator for further information.

Plagiarism

Plagiarism is the act of representing as one's own original work the creative works of another, without appropriate acknowledgment of the author or source.

Collusion

Collusion is the presentation by a student of an assignment as his or her own which is in fact the result in whole or in part of unauthorised collaboration with another person or persons. Collusion involves the cooperation of two or more students in plagiarism or other forms of academic misconduct.

Both collusion and plagiarism can occur in group work. For examples of plagiarism, collusion and academic misconduct in group work please see the University's policy on Academic Honesty and Plagiarism: http://academichonesty.unimelb.edu.au/

Plagiarism and collusion constitute cheating. Disciplinary action will be taken against students who engage in plagiarism and collusion as outlined in University policy. Proven involvement in plagiarism or collusion may be recorded on my academic file in accordance with Statute 13.1.18.

STUDENT DECLARATION

Please sign below to indicate that you understand the following statements:

I declare that:

- This assignment is my own original work, except where I have appropriately cited the original source.
- This assignment has not previously been submitted for assessment in this or any other subject.

For the purposes of assessment, I give the assessor of this assignment the permission to:

- Reproduce this assignment and provide a copy to another member of staff; and
- Take steps to authenticate the assignment, including communicating a copy of this assignment to a checking service (which may retain a copy of the assignment on its database for future plagiarism checking).

18/Oct/2019

Student signature ..... Date .....

Q1.1

The *socket()* function returns a *socket object* which support various socket families. The address format is automatically selected based on the address family specified when the socket object was created [1].

Socket address families include AF_UNIX, AF_INET(default), AF_INET6, AF_NETLINK, AF_CAN and AF_TIPC (Linux-only support for TIPC) etc. The socket types include SOCK_STREAM(default), SOCK_DGRAM, SOCK_RAW, SOCK_RDM and SOCK_SEQPACKET etc.

SOCK_STREAM indicating the type of connection-oriented socket (TCP). It provides a bidirectional, sequenced, and reliable channel of communication which is similar to pipes [2]. We can choose this one when we want to ensure the accuracy of transfer.

SOCK_DGRAM indicating the type of connectionless socket (UDP). It does not guarantee sequenced, reliable delivery, but they do guarantee that message boundaries will be preserved when read [2]. We can choose this one when a faster transfer is required and the data error can be tolerated.

Q1.2

The *socket.bind()* function can bind the socket to address. A pair *(host, port)* is used for the AF_INET address family, where host is a string representing either a hostname in Internet domain notation or an IPv4 address, and port is an integer [1]. We can use this when implement both client and server.

Q1.3

Connectionless sockets use User Datagram Protocol (UDP) instead of TCP/IP. The function, socket.AF_INET and socket.SOCK_DGRAM, could be used in this case.

We can use the function socket.AF_INET and socket.SOCK_STREAM if implementing in a connection oriented mode.

Q1.4

Sockets can be controlled by calling function *ioctlsocket()* to determine working in blocking or non-blocking mode.

By default, TCP sockets are in blocking mode. In blocking mode, the *recv(), send(), connect()* (TCP only) and *accept()* (TCP only) socket API calls will block indefinitely until the requested action has been performed. It is easier to implement but only one socket is active in one thread.

In non-blocking mode, these functions return immediately instead of waiting for an operation to stop. This is an invaluable tool if you need to switch between many different connected sockets, and want to ensure that none of them cause the program to "lock up" [3]. *Select()* will block until the socket is ready and it takes a time-out parameter which controls the amount of time to wait for the action to complete or an error to be returned [4].

Q1.5

The code [5] and the result are shown in the following figures. We can see that the server can receive the message sent by client. And when ‘exit’ is input by client, the connection will be terminated.

```python
import socket
import time

# set up the socket using local address
socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
socket.bind(('localhost', 9999))
while 1:
    # get the data sent to us
    data, ip = socket.recvfrom(1024)
    # display
    print("{}: {}".format(ip, data.decode(encoding="utf-8").strip()))
    # send acknowledge with timestamp back
    now = int(round(time.time()*1000))
    now02 = time.strftime('%Y-%m-%d %H:%M:%S ack',time.localtime(now/1000))
    socket.sendto(now02.encode(), ip)
    if data.decode()=='exit':
        socket.shutdown(0)
        socket.close()
        break
```

```
In [13]: runfile('C:/AUS/semester 5/comm net/workshop/ws_4/sever.py', wdir='C:/AUS/
semester 5/comm net/workshop/ws_4')
('127.0.0.1', 54293): Hello
('127.0.0.1', 54293): This is client
('127.0.0.1', 54293): exit

In [14]:
```

Figure 1 The server(code and result)

```python
#Code reference:https://medium.com/@makerhacks/python-client-and-server-internet-communication-using-udp-c4f5fc608945

import socket

try:
    socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
except socket.error:
    print("Oops, something went wrong connecting the socket")
    exit()

while 1:
    message = input("> ")
    if len(message) > 140:
        print("invalid")
        continue
    # encode the message
    message_ = message.encode()
    # send the message
    socket.sendto(message_, ('localhost', 9999))
    # output the response (if any)
    data, ip = socket.recvfrom(1024)
    print("{}: {}".format(ip, data.decode()))
    if message=='exit':
        socket.close()
        break
```

```
In [10]: runfile('C:/AUS/semester 5/comm net/workshop/ws_4/client.py', wdir='C:/AUS/
semester 5/comm net/workshop/ws_4')

> Hello
('127.0.0.1', 9999): 2019-10-21 15:29:36 ack

> This is client
('127.0.0.1', 9999): 2019-10-21 15:29:45 ack

> exit
('127.0.0.1', 9999): 2019-10-21 15:29:47 ack

In [11]:
```

Figure 2 The client(code and result)

Q2.1

The request methods determines how form data is submitted to the server.

GET is used to request data from a specified resource. When the method is GET, all form data is encoded into the URL, appended to the action URL as query string parameters [6].

POST is used to send data to a server to create or update a resource [7]. With POST, form data appears within the message body of the HTTP request [6].

Other request methods [8]:

HEAD

The HEAD method asks for a response identical to that of a GET request, but without the response body.

DELETE

The DELETE method deletes the specified resource.

CONNECT

The CONNECT method establishes a tunnel to the server identified by the target resource.

OPTIONS

The OPTIONS method is used to describe the communication options for the target resource.

TRACE

The TRACE method performs a message loop-back test along the path to the target resource.

PATCH

The PATCH method is used to apply partial modifications to a resource.

Q2.2

The program defines a simple wed server. In the main function: first, creating an HTTP web server by using **HTTPServer** function which stores the server address consist of host name and port name, then print the starting time of the server; then, use **http.serve_forever()** function to handle one request at a time until interrupt by user; after that, the server is closed by using **httpd.server_close()** function, then print the stop time of the server.

In the handler class, three functions are defined:

info_message(self) create information message based on the GET request, including the client address, command used, path, request version, server version and protocol version etc.

do_GET(self) responds to a GET request. The request is mapped to a local file by interpreting the request as a path relative to the current working directory [9]. The message text is assembled and then written to wfile, the file handle wrapping the response socket. Each response needs a response code, set via send_response() [10].

do_POST(self) responds to a POST request. This method serves the 'POST' request type, only allowed for CGI scripts [9].

Q2.3

Each response needs a response code, set via **send_response()**. The response code 200 means 200 OK, which means the request is success. If an error code is used (404, 501, etc.), an appropriate default error message is included in the header, or a message can be passed with the error code [10]. For example 404 Not Found, which means the server did not find anything matching the request.

Q2.4

do_GET(self) responds to a GET request. First, it responds to the request by sending 200 to the client; then, generate the headers using **self.send_header()**; next, sending the encoded content, the message text is assembled and then written to wfile; then, using **self.info_message()** to add the information message to the response.

Q2.5

do_POST(self) responds to a POST request. First, is parses the form data posted; then responds to the request by sending 200 to the client, together with other information like client, user agent, path and form data. In the end, the function echoes back information about what was posted in the form.

Q2.6

A web server is server software that can satisfy World Wide Web client requests. A web server can contain one or more websites. A web server processes incoming network requests over HTTP and several other related protocols [11]. It makes it easy for users to access the main information by translating the HTML language.

The browser we wrote only fetched the original information from the server, but this is incomprehensible for human. We can use our browser when facing machine, but we need a web server for human.

Q3.1

Design details:

For this question, the program of client will first ask the user to input the desired temperature as well as the desired price, then we have the client that will read temperature data, and it will ask the server for the price information at the same time index, based on temperature and price information as well as the user pre-setting desired temperature and desired price, it will determine a temperature set point.

The set point is designed as follow:

$$set\ point = \ desired\ temp - 0.1 \times (price - desired\ price)$$

Simulation result:

The simulation is run through the whole of 44 time indexes. With user input the desired temperature is 22 and desired price is 40.

```
Please input the desired temperature in °C:22

Please input the desired price:40
```

Figure. User input

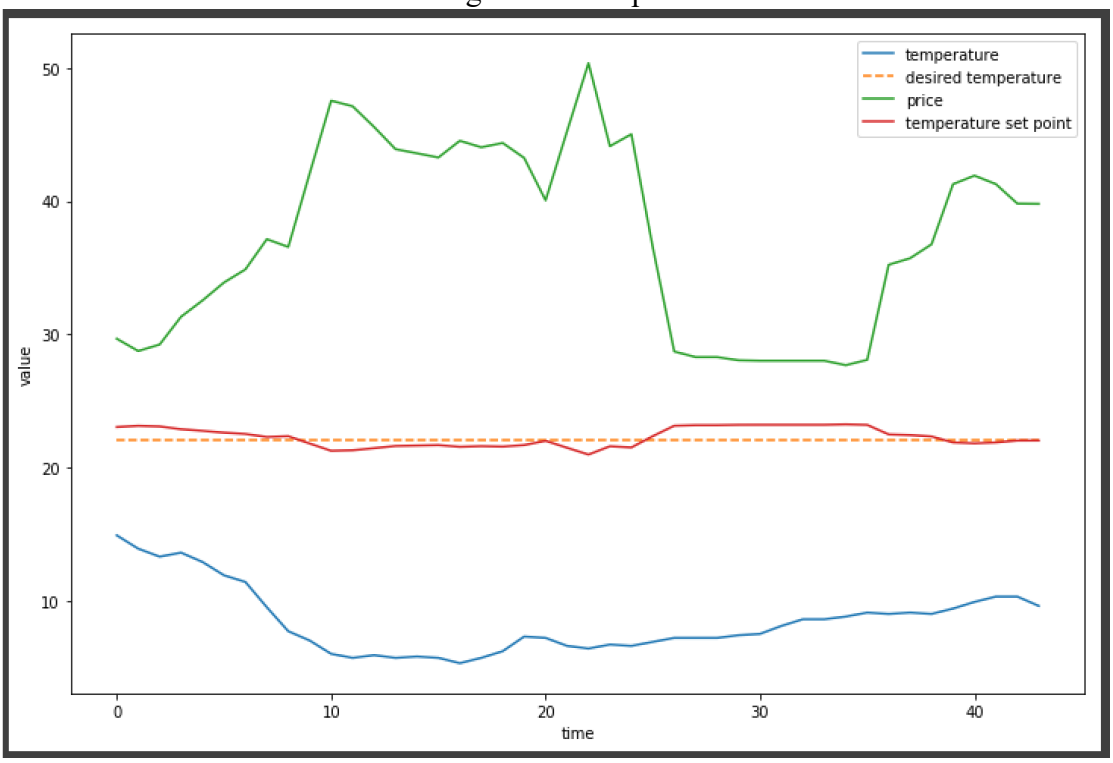

Figure. Simulation result

From the result, we can see that the temperature is set around the desired temperature of 22, and small adjustments are made according to the difference between the price and desired price information.

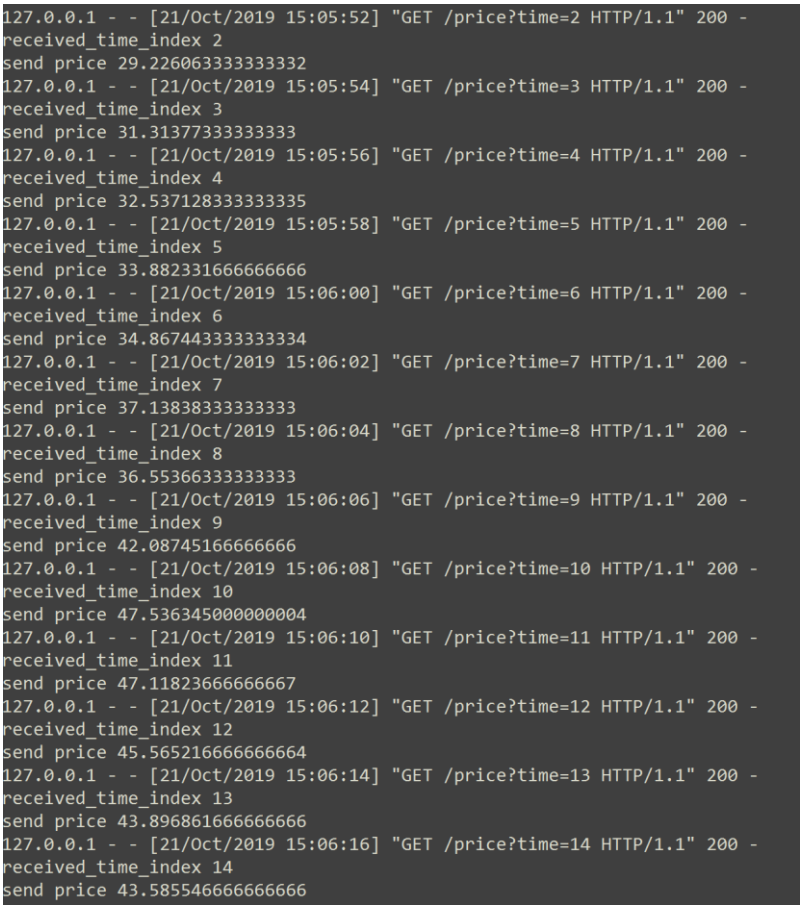

Figure. Screen shot of server sending price according to time index

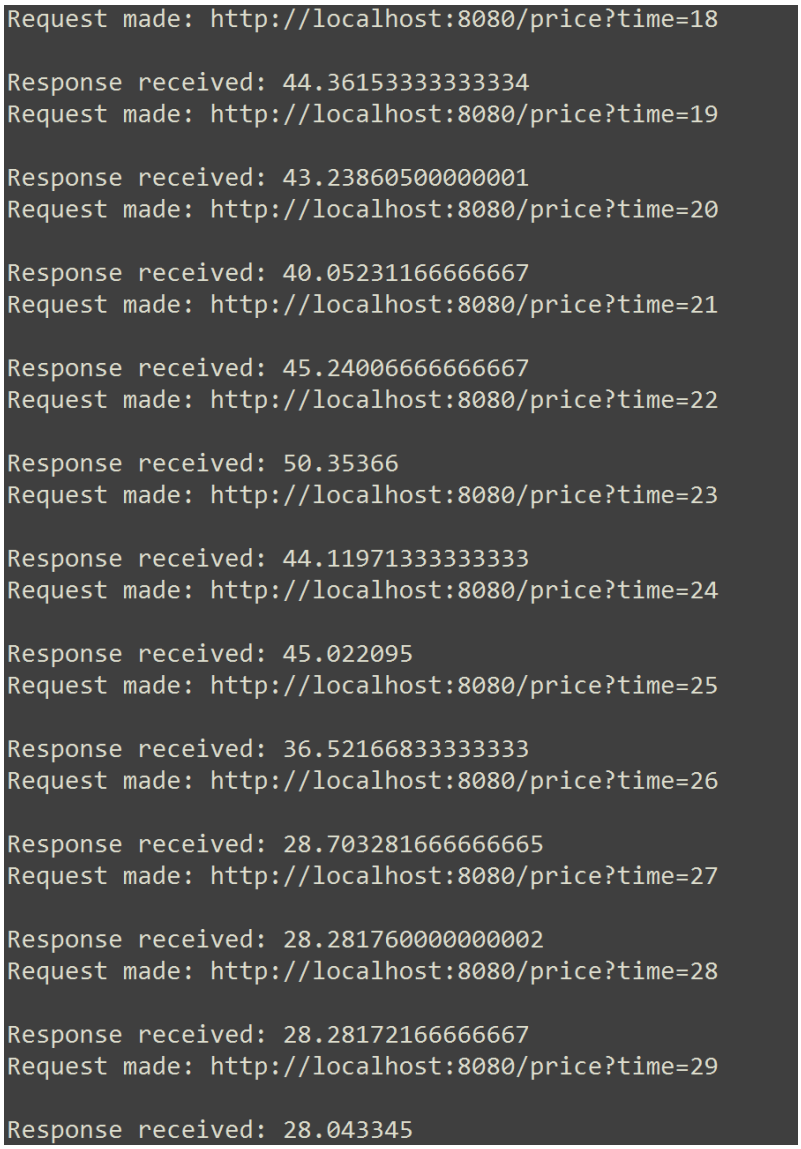

Figure. Screen shot of client requesting and receiving price

Reference

[1] python, "socket," [Online]. Available: https://docs.python.org/3/library/socket.html.

[2] p. cookbook, "sockets," [Online]. Available: http://nnc3.com/mags/Perl3/cookbook/ch17_01.htm.

[3] scottklement, "nonblocking," [Online]. Available: https://www.scottklement.com/rpg/socktut/nonblocking.html.

[4] on-time, "blocking-and-non-blocking-sockets," [Online]. Available: http://www.on-time.com/rtos-32-docs/rtip-32/programming-manual/tcp-ip-networking/blocking-and-non-blocking-sockets.htm.

[5] [Online]. Available: https://medium.com/@makerhacks/python-client-and-server-internet-communication-using-udp-c4f5fc608945.

[6] "GET-vs-POST-HTTP-Requests," [Online]. Available: https://www.diffen.com/difference/GET-vs-POST-HTTP-Requests.

[7] "ref_httpmethods," [Online]. Available: https://www.w3schools.com/tags/ref_httpmethods.asp.

[8] "Methods," [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods.

[9] "http.server," [Online]. Available: https://docs.python.org/3/library/http.server.html.

[10] BaseHTTPServer. [Online]. Available: https://pymotw.com/2/BaseHTTPServer/.

[11] wikipedia, "Web_server," [Online]. Available: https://en.wikipedia.org/wiki/Web_server.