Question 1.1

Briefly discuss socket family and type within the context of the `socket()` function. What are the differences between the `SOCK_STREAM` and `SOCK_DGRAM` socket types? In which context would you use one over the other?

`AF_UNIX`, `AF_INET` and `AF_INET6` represent the address (and protocol) families, used for the first argument to `socket()`.

`SOCK_STREAM`, `SOCK_DGRAM`, `SOCK_RAW`, `SOCK_RDM` and `SOCK_SEQPACKET` represent the socket types, used for the second argument to `socket()`. (Only `SOCK_STREAM` and `SOCK_DGRAM` appear to be generally useful.)

`SOCK_STREAM` provides sequenced, reliable, two-way, connection-based byte streams. An out-of-band data transmission mechanism may be supported.
`SOCK_DGRAM` supports datagrams (connectionless, unreliable messages of a fixed maximum length).

For small messages with known upper bound on message length, e.g. UDP broadcasts sends are always enabled on a datagram socket, `SOCK_DGRAM` is recommended.
For arbitrary long message payloads such as file tranfer, `SOCK_STREAM` would be better.

Reference:
`https://docs.python.org/2/library/socket.html`
`http://man7.org/linux/man-pages/man2/socket.2.html`

Question 1.2

What is the purpose of the `socket.bind()` function in the Python socket module? What is the format of the address parameter in the `AF_INET` case? Would you use this function when implementing a server, a client or both? Justify your answer.

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 like '`www.unimelb.edu.au`' or an IPv4 address like '`8.8.8.8`', and port is an integer.

I would not use this function when implementing a client, primarily because if a socket does not call `bind()`, the operating system will automatically assign an ephemeral port. However, on servers, well-known ports are used by system processes that provide widely used types of network services, e.g. 80 for HTTP, 22 for SSH, 53 for DNS. `bind()` function is called in order to agree with official assignments of port numbers for specific uses.

Question 1.3

Which functions of the socket API would you need to implement a simple server using a connectionless mode of communication? Which functions of the socket API would you need to implement a simple server using a connection oriented mode of communication?

Connectionless (UDP)

Socket type: `SOCK_DGRAM`
`socket()`, `bind()`, `sendto()`, `recvfrom()`, `close()`

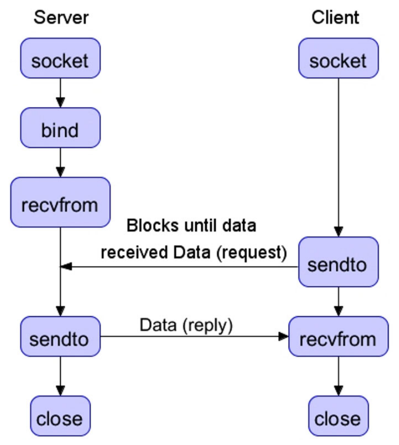

Figure 1: APIs for UDP Server/Client

Connection-oriented (TCP)

Socket type: `SOCK_STREAM`
`socket()`, `connect()`, `bind()`, `listen()`, `accept()`, `send()`, `recv()`, `close()`

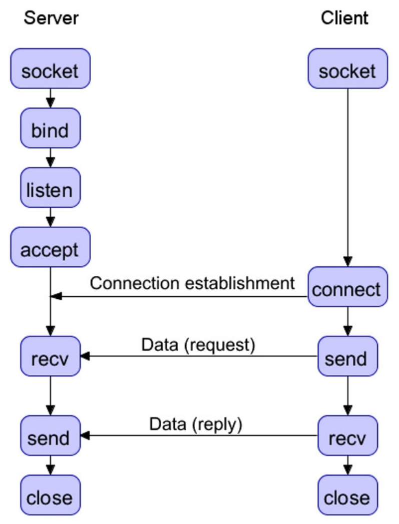

Figure 2: APIs for TCP Server/Client

Reference:
https://docs.python.org/2/library/socket.html#socket-objects
http://liuj.fcu.edu.tw/net_pg/socket.html

Question 1.4

Explain the difference between a blocking and non-blocking socket. Which one is used most commonly in practice?

In non-blocking mode, if a `recv()` call doesn't find any data, or if a `send()` call can't immediately dispose of the data, a error exception is raised; in blocking mode, the calls block until they can proceed.

Blocking mode is used most commonly in practice.

Reference:
`https://docs.python.org/2/library/socket.html#socket.socket.setblocking`

Question 2.1

Describe the use and purpose of the following functions, and indicate if they will be used to implement a client, a server or both: (a) `socket.connect()`, (b) `socket.listen()`, (c) `socket.accept()`.

(a) `socket.connect(address)`

Connect to a remote socket at `address`.

Implemented on a client, because in most cases, a client initiates a connection.

(b) `socket.listen(backlog)`

Listen for connections made to the socket. The `backlog` argument specifies the maximum number of queued connections and should be at least 0; the maximum value is system-dependent (usually 5), the minimum value is forced to 0.

Implemented on a server, because servers usually wait to be connected.

(c) `socket.accept()`

Accept a connection. The socket must be bound to an address and listening for connections. The return value is a pair `(conn, address)` where `conn` is a new socket object usable to send and receive data on the connection, and `address` is the address bound to the socket on the other end of the connection.

Implemented on a server. After a client initiates a connection, a server will accept it.

Reference:
`https://docs.python.org/2/library/socket.html#socket-objects`

Question 2.2

Why does `socket.accept()` return a new socket each time? Should you set it to be blocking or non-blocking in your program (assuming a single threaded server)?

`socket.accept()`
Accept a connection. The socket must be bound to an address and listening for connections. The return value is a pair `(conn, address)` where `conn` is a new socket object usable to send and receive data on the connection, and `address` is the address bound to the socket on the other end of the connection.

New sockets should be generated because the original one is used to wait for new connection from other users.

Blocking mode should be set in the the program. Once a new client is successfully connected, a new socket will be established. If no connections are pending, `socket.accept()` will block until next connection comes.

Appendix

Question 1.5 Server

```
# -*- coding: utf-8 -*-
"""
@author: Ang LI 631317 & Qingyun LIN 682834
"""

import socket
import sys
import time

# define server address (IP (localhost = 127.0.0.1) and port)
serverAddress = ('localhost', 8080)

# The port numbers in the range from 0 to 1023 are the well-known ports or system ports.
    They are used by system processes that provide widely used types of network services. On
     Unix-like operating systems (e.g. Mac OS X), a process must execute with superuser
    privileges (>>> sudo python server.py) to be able to bind a network socket to an IP
    address using one of the well-known ports. Hence, we decide to use 8080.

# define buffer size
bufsize = 140

# create a connectionless socket (SOCK_DGRAM)
mysocket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# bind 'mysocket' to the defined server address
mysocket.bind(serverAddress)

while True:
    # get request (string) and client address
    request, clientAddress = mysocket.recvfrom(bufsize)

    # print the request message
    print 'Message received from client: ', request

    # generate a response with a timestamp
    response = 'Message received at ' + time.ctime()

    # send response message back to client address
    mysocket.sendto(response, clientAddress)

# close the socket
mysocket.close()
print 'socket closed'
```

Question 1.5 Client

```python
# -*- coding: utf-8 -*-
"""
@author: Ang LI 631317 & Qingyun LIN 682834
"""

import socket
import sys

# define server address (IP (localhost = 127.0.0.1) and port)
serverAddress = ('localhost', 8080)

# The port numbers in the range from 0 to 1023 are the well-known ports or system ports.
    They are used by system processes that provide widely used types of network services. On
     Unix-like operating systems (e.g. Mac OS X), a process must execute with superuser
    privileges (>>> sudo python client.py) to be able to bind a network socket to an IP
    address using one of the well-known ports. Hence, we decide to use 8080.

# define buffer size
bufsize = 140

# creat a connectionless socket (SOCK_DGRAM)
mysocket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
# timeout after 5 seconds
mysocket.settimeout(5)

while True:
    # get keyborad inputs
    message =  raw_input("Please input the message or enter 'exit' to terminate the program
        .\n")
    # remove leading and trailing whitespace characters
    message = message.strip()

    # if the message is longer than 140 characters, ask for a new message.
    if len(message) > 140:
        print 'too long message'
        continue

    if message == 'exit':
        break

    try:
        # send the mesaage to a certain address
        mysocket.sendto(message, serverAddress)
        # get server response
        response, address = mysocket.recvfrom(bufsize)
        print 'Server response: ', response
    # if error occurs, show the error information.
    except socket.error as msg:
        print 'Error:', msg

# close the socket
mysocket.close()
print 'socket closed'

# terminate the program
sys.exit()
```

Question 2.3 Server

```
# -*- coding: utf-8 -*-

import socket
import select
import sys

HOST = 'localhost'
PORT = 8080
# define address

# The port numbers in the range from 0 to 1023 are the well-known ports or system ports.
    They are used by system processes that provide widely used types of network services. On
     Unix-like operating systems (e.g. Mac OS X), a process must execute with superuser
    privileges (>>> sudo python server.py) to be able to bind a network socket to an IP
    address using one of the well-known ports. Hence, we decide to use 8080.

bufsize = 1024
# define buffer size

socketList = []
addressList = []
# declare socket list and address list

# server initialization
def serverInit():
    global socketServer
    # set socketServer a global variable

    try:
        socketServer = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        print 'server established'
        # create a connection-oriented socket (SOCK_STREAM)
    except:
        print 'cannot initialize server'
        sys.exit()

    socketServer.setblocking(0)
    # Set non-blocking mode of the socket. (Initially all sockets are in blocking mode.)

    try:
        socketServer.bind((HOST, PORT))
        print 'address bound'
    except:
        print 'cannot bind the host and port'
        sys.exit()

    socketServer.listen(5)
    print 'listening port', PORT

    socketList.append(socketServer)
    addressList.append("%s:%s" % (HOST, PORT))
    # append to socket list and address list

# broadcast message except the originating user
def broadcast(socketExcept, message):
    for s in socketList:
        if s != socketServer and s != socketExcept:
        # exclude the originating user and the server
            try:
            # try to send the message to current socket
```

```
                s.send(message)
            except socket.error, err:
                print err

# accept incoming connections
def acceptClient(s):
    clientsocket, address = s.accept()
    # aspect new connection and assign a new socket

    socketList.append(clientsocket)
    addressList.append("%s:%s" % address)
    # append to socket list and address list

    print "Client (%s:%s) connected" % address
    # 'client connected' notification

    clientsocket.send("Welcome, Client (%s:%s).\nPlease enter your message.\n" % address)
    # send the welcome message to the new user

    broadcast(clientsocket, "Client (%s:%s) connected\n" % address)
    # announce the arrival of new user

# check users for incoming messages
def checkUsers(s):
    try:
    # try to receive message from clients to check on-line status
        message = s.recv(bufsize)
        message = message.strip()
        # remove leading and trailing whitespace characters
        if message:
            s.send('"%s" received by the server\n' % message)
            # send an acknowledgment response to the originating user
            peername = "Client (%s:%s)" % s.getpeername()
            # get address (IP and port)
            broadcast(s, peername + ' said: ' +  message + '\n')
            # broadcast incoming messages to all users except from the originating one
    except:
    # if trial fails, close the socket and remove it from the list
        s.close()
        # close the socket

        index = socketList.index(s)
        # get index number of this socket in the socket list
        del socketList[index]
        # delete this socket from socket list
        addressString = addressList.pop(index)
        # get the address string from address list and delete

        print "Client (%s)" % addressString, 'left'
        # 'client left' notification

        broadcast(None, "Client (%s)" % addressString + ' left\n')
        # announce the departure of user

### ### ### Main ### ### ###

serverInit()
# server initialization

while True:
    rlist, wlist, xlist = select.select(socketList, [], [])
    # Reference: https://docs.python.org/2/howto/sockets.html#non-blocking-sockets
    # Reference: http://www.ibm.com/developerworks/linux/tutorials/l-pysocks/#N103D1

    for s in rlist:
        if s == socketServer:
            acceptClient(s)
            # accept incoming connections
        else:
            checkUsers(s)
            # check users for incoming messages

socketServer.close()
```

Question 2.3 Client

```
# -*- coding: utf-8 -*-

import telnetlib

HOST = 'localhost'
PORT = 8080
# define address

tn = telnetlib.Telnet(HOST, PORT)
tn.interact()
```