Lesson Plan 14 | Form 2 | Modified Chatroom: Identify Senders by IP

Objective

Students will analyze the chatroom program introduced previously, this time with the intention of making one or more changes to satisfy the goal of identifying each message's author by IP address.

Warm-up **(requires central file server)**

In lab, have students open the two Python files comprising the chatroom application. For the first half of class, ask them to figure out where we can change the program so as to show the IP address of each message's sender, like so:

10.0.2.14 said: …

See the code below for the highlighted changes required for this. Note that they should be encouraged to work together, but little instructor guidance should be given. Have students with good ideas come to the whiteboard and write them down, as a way to give encouragement.

Presentation

Halfway through class, gather the class together to talk about the various ideas that have been mentioned or tried. Place emphasis on those that are most promising or will accomplish the goal, and give students the remainder of class to actually implement the changes.

Guided Practice

As time runs out, give more advice to get as many students as possible to a working implementation by the end of class. Ideally, most students should be able to run the application and see their changes in effect before they leave.

Independent Practice

None.

Closing

None.

In Hindsight

02/27/2017: As expected, this was challenging for the students. Many of them experimented with making changes to the program, as they were encouraged to do, but few thought critically about where changes could best be applied in order to accomplish the goal. I refocused the class by reminding them of the purpose of various functions, then asking them to find the location in the program where messages are sent to chatroom participants. I gave bonus points to those who came

close or identified the proper location. By the end of the 80-minute class for each stream, several people/groups had made progress but no one had completed both required code changes, so I expanded this lesson to cover two 80-minute class periods. I think this lesson is valuable to identify those students with exceptional analysis abilities, but as a general class-wide exercise, it overwhelmed quite a few of the rest.

chat.py

```
import tumaini_chatroom
import socket, threading
choice = raw_input("Type 'h' to create a chatroom or 'j' to join one: ")
# If the user pressed 'h', create a new chatroom.
if choice == 'h':
    socket = tumaini_chatroom.create_chatroom()
    try:
        while True:
            msg = raw_input()
            tumaini_chatroom.send(msg)
    except KeyboardInterrupt:
        print "Closing your chatroom..."
        socket.close()
        print "Chatroom closed. Goodbye."
# If the user pressed 'j', join the chatroom of their choice.
elif choice == 'j':
    ip = raw_input("What is the IP address of the chatroom you want to join: ")
    port = raw_input("What is the port of the chatroom you want to join: ")
    s = socket.socket()
    s.connect((ip, int(port)))
    def show_new_messages():
        while True:
            messages = s.recv(1024)
            if messages:
                print messages
            else:
                print "The chatroom was closed, goodbye."
                s.close()
                return
    new_msg_thread = threading.Thread(target=show_new_messages)
    new_msg_thread.daemon = True
    new_msg_thread.start()
    try:
        while True:
            msg = raw_input()
            s.send(msg)
    except KeyboardInterrupt:
        s.close()
        print "You are leaving the chatroom. Goodbye."
# If the user didn't press 'j' or 'h',
# tell them that they need to try again.
else:
    print "You didn't type 'h' or 'j', try running this program again."
```

tumaini_chatroom.py

```
import socket, threading


people_in_chatroom = {}
unsent_messages = []


def create_chatroom():
    PORT = 4000
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
```

```
    s.bind(('', PORT))
    s.listen(100)
    print ("Chatroom created on port %d successfully, " + \
            "now listening for people to join.") % PORT
    bcast_thread = threading.Thread(target=broadcast_received_messages)
    bcast_thread.daemon = True
    bcast_thread.start()
    def listen():
        while True:
            connection, ip_and_port = s.accept()
            people_in_chatroom[ip_and_port[0]] = connection
            t = threading.Thread(target=new_person_in_chatroom, \
                    args=(ip_and_port[0],))
            t.start()
    listen_thread = threading.Thread(target=listen)
    listen_thread.daemon = True
    listen_thread.start()
    return s


def new_person_in_chatroom(ip_address):
    unsent_messages.insert(0, '%s has just joined the chatroom.' % ip_address)
    while True:
        message = people_in_chatroom[ip_address].recv(1024)
        if not message:
            break
        unsent_messages.insert(0, “%s said: %” % (ip_address, message))
    people_in_chatroom[ip_address].close()
    people_in_chatroom.pop(ip_address, None)
    unsent_messages.insert(0, '%s has left the chatroom.' % ip_address)


def broadcast_received_messages():
    while True:
        if len(unsent_messages) > 0:
            msg = unsent_messages.pop()
            print msg # Ensures that we see messages ourselves.
            for (ip_address, connection) in people_in_chatroom.items():
                connection.sendall(msg)


def send(msg):
    unsent_messages.insert(0, “Chatroom owner said: %s” % msg)
```