Lab3 Write Up

Airi Kokuryo

November 18, 2023

1. Review the example multi-connection client and server examples from the Python socket programming guide and explain each function.

a. What do the accept_wrapper(), service_connection(), and start_connections() functions do in the client and server script?

- accept_wrapper(self, sock):

```
1 def accept_wrapper(sock):
2 conn, addr = sock.accept()
3 # Should be ready to read
4 print(f"Accepted connection from {addr}")
5 conn.setblocking(False)
6 data = types.SimpleNamespace(addr=addr, inb=b"",
    outb=b"")
7 events = selectors.EVENT_READ | selectors.
    EVENT_WRITE
8 sel.register(conn, events, data=data)
```

This function accepts incoming connections and inputs the connection to conn, and address in addr. Here, the SimpleNamespace called data is created to hold the data that it receives upon connection. The events are defined for the selectors, whether it is in a reading data state or a writing data state. On line 8, the sel.register registers the connection and its associated data with the selector.

- service_connection(self, key:selectors.SelectorKey, mask):

```
1 def service_connection(key, mask):
2 sock = key.fileobj
3 data = key.data
4 if mask & selectors.EVENT_READ:
5     recv_data = sock.recv(1024)
6     # Should be ready to read
7     if recv_data:
8         data.outb += recv_data
```

```
    else:
        print(f"Closing connection to {data.addr}"
            )
        sel.unregister(sock)
        sock.close()
if mask & selectors.EVENT_WRITE:
    if data.outb:
        print(f"Echoing {data.outb!r} to {data.
            addr}")
        sent = sock.send(data.outb)
        # Should be ready to write
        data.outb = data.outb[sent:]
```

In this function, the I/O events are managed by the event triggered by the selector. if the event is EVENT_READ, check for the read event. In the read event, it checks if it receives any data from the socket, and if there is data, add it to the data.outb. If no data is received, it closes the connection through sock.close function. If the event is EVENT_WRITE, it sends the data back to the client if there is any data received.

- start_connections(self, sock:socket.socket):

```
def start_connections(host, port, num_conns):
server_addr = (host, port)
for i in range(0, num_conns):
    connid = i + 1
    print(f"Starting connection {connid} to {
        server_addr}")
    sock = socket.socket(socket.AF_INET, socket.
        SOCK_STREAM)
    sock.setblocking(False)
    sock.connect_ex(server_addr)
    events = selectors.EVENT_READ | selectors.
        EVENT_WRITE
    data = types.SimpleNamespace(
        connid=connid,
        msg_total=sum(len(m) for m in messages),
        recv_total=0,
        messages=messages.copy(),
        outb=b"",
    )
    sel.register(sock, events, data=data)
```

This function initiates the multiple non-blocking TCP connections to a specified host and port using the selectors module. The for loop creates

the specified number of connections. Here, the function creates a non-blocking socket and attempts to initiate a connection to the server. Events and data are defined and prepared for connection. At last, it registers the socket and its associated data with the selector.

b. What is the main difference between the multi-connection server and the single-connection server?

The multi-connection server can handle multiple client connections simultaneously, while the single-connection server can only handle one client connection at a time. This means that the multi-connection server may serve multiple servers without blocking its precision, whereas the single-connection server must block or wait for its connection to end before connecting to the other servers.

2. Explain your design decisions for your implementation of polling using the three Pis.

In the primary.py, it calls out the three classes we have created separately(sink_server.py and classes.py). In the main function, it starts the sink server and the loop where it continuously polls the data until it is interrupted by the key input. The IP address of the primary Pi would be "192.168.1.1" and the two secondary Pis would use the IP addresses "192.168.1.2" for host1 and "192.168.1.3" for host2. In the while loop, the data request message is first sent to host1 and listens for host1 to send back a data packet. It will run the function that will allow the sent message to be encrypted into a list for each data type: temperature, humidity, soil moisture, and speed. If the packet fails to send back, the list will add a 0 for their data. It closes the connection with the host1 after this process and sends the request to host2. After this, the public list will be used for plotting the data, and the list will be reset for the start of another loop.

a. Include a picture of your system design and logic in the form of a flow chart or pseudo-code.

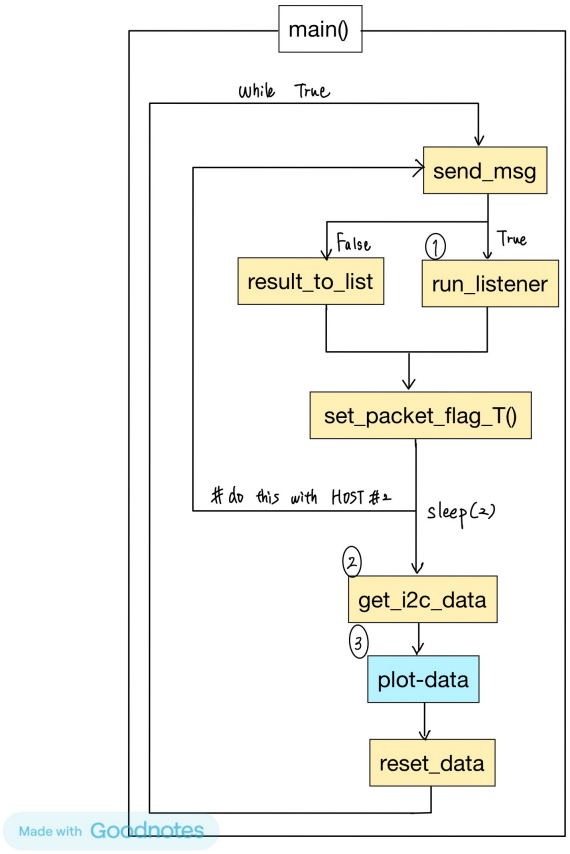

Figure 1: System design and logic for polling 3 Pis from codes

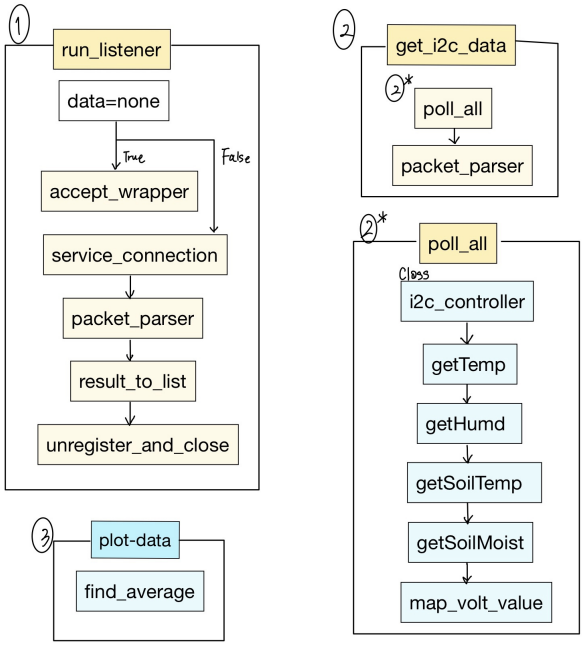

Figure 2: System design and logic for polling 3 Pis from codes

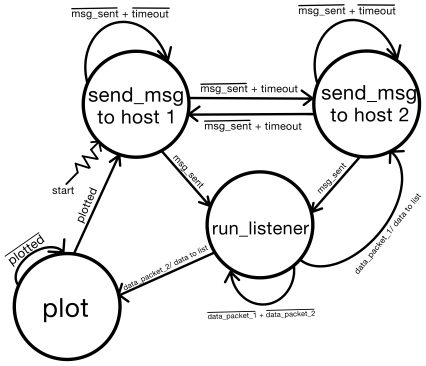

Figure 3: System design and logic for polling 3 Pis

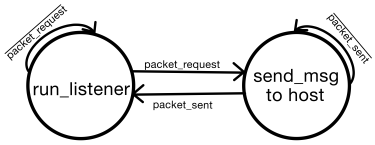

Figure 4: System design and logic for polling 3 Pis

b. Were there any difficulties you encountered when setting up communication between the Pis?

- Difficulties in the ad-hoc connection between the pis.

  Set the pis into ad-hoc mode and start communication:

```
1 raspberrypi: ~ $ wpa_cli terminate
2 raspberrypi: ~ $ sudo ifconfig wlan0 down
3 raspberrypi: ~ $ sudo iwconfig wlan0 mode ad-hoc
     channel 04 essid <ssid>
4 raspberrypi: ~ $ sudo ifconfig wlan0 up
```

  The SSID must be the same as the other Raspberry Pis you would like to connect. To send and receive data from these Pis, the port number must be the same as well. However, the IP address must be different, so just change the last digit number to identify which pi is the first, second, and third. (e.g., "192.168.1.1", "192.168.1.2", "192.168.1.3")

- Difficulties where the data type that is being sent from the Pis and the data type for receiving I2C data in its own Pi are different.

  The data type of the received data is "byte", whereas the data type of the I2C data polled in its own Raspberry Pi is a "list". Therefore, different parsing method was taken between these two data, and it was hard to notice until we kept on running the debug log of the data type. After parsing, the data was turned into a double list of string and float type so this could be used for plotting.

3. Explain your design decisions for your implementation of the token-ring approach using the three Pis.

First, the Python file will ask the user, "Are you the token bearer?". If you say yes, you will be the token bearer and will be the first one to send the data. If you say no, you will be in the listening state and will wait for the data to be sent. The data will be decoded and put into the list in the packet_ring_handler and will also be plotted there if it receives two messages. This will be handled by using the sequence number, which is updated every turn and is added at the end of its byte message. The parser function will parse this message and take out the sequence number to be used in many other functions.

a. Include a picture of your system design and logic in the form of a flow chart or pseudo-code.

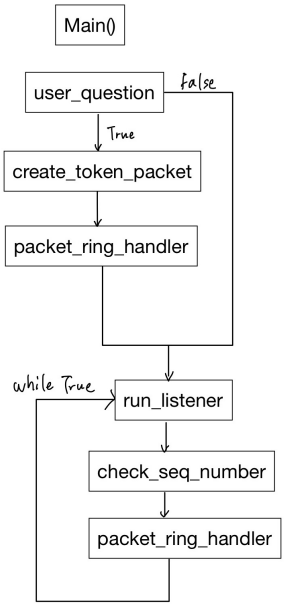

Figure 5: System and logic design for token-ring

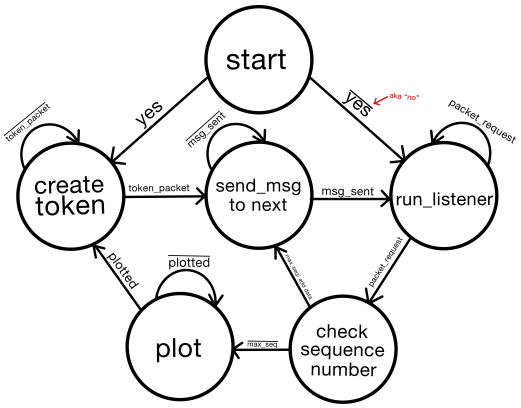

Figure 6: System and logic design for token-ring

b. Were there any difficulties you encountered when setting up communication between the Pis?

Starting the communication was difficult in having the three Raspberry Pi run the same Python file. Since all Pis should have the same Python file, the identification of which Raspberry Pi to have the token first was difficult. I thought about identifying it by its IP address, however, it was better to make the decision dynamic by asking if the Raspberry Pi is the token bearer and typing yes or no.

4. Describe how you implemented the timeout mechanism in the two approaches.

- primary.py

In the primary.py, we implemented the timeout mechanism in the socket connection and the select events. In the function where we send a socket message to a specified address and port, we set a timeout of 2 seconds so it will return an error for failing to send the message. In the run_listener function where it listens for connections and starts the main event loop, we set a 5-second timeout for getting no packets.

- secondary.py and token_ring.py

For the secondary.py, we implemented the timeout mechanism in the run_listener only since it only needs to send the message to the primary pi. Also for the token ring, the timeout is implemented in the run_listener when there is a token loss in its connection.

a. In what instance would a timeout mechanism be useful in the polling approach? How about the token-ring-based approach?

   In the polling approach, it is useful to use a timeout when there is no resource. Since the primary pi requests for the data, listens for the reply, and skips to the other secondary pi if there is no reply, the timeout method is useful in deciding when to close its connection with the first secondary pi and move on to the other secondary pi. On the other hand, in the token-ring-based approach, it is useful to have a timeout when the token is lost. Since the Raspberry Pi will keep on listening for the connection even when the token is lost and fails in connection, the timeout method would be useful to stop this loop and close the connection when they get no data packets for a long time and assume that the token has been lost.

b. In addition to timeout, what are some other possible corner cases in the polling and token-ring-based approaches?

   Other than the timeout, we did the error handling in both the polling and the token-ring-based approaches. In the polling approach, we input 6 try and excepts.

   - When there is a failure in sending the message
   - When it fails to listen and does not get the packet
   - When it succeeds in listening but gets no packet after 5 seconds
   - When the Raspberry Pi fails to poll its data because it receives no voltage
   - When it fails to get its I2C data
   - When it fails to change the received value into a float

   In the token-ring-based approach, we input 8 try and excepts.

   - When it fails to listen and does not get the packet
   - When it succeeds in listening but gets no packet after 5 seconds
   - When the Raspberry Pi fails to connect to other Pis and can not send the data
   - when trying to unregister the connection
   - When trying to close the connection
   - When the Raspberry Pi fails to poll its data because it receives no voltage
   - When the Raspberry Pi can't encapsulate its own data
   - When it can't change its value into a float for the list to plot

5. Compare the polling and token-ring-based approaches you implemented in terms of latency of calculating the averages, the freshness of the data, and the resources used.

a. What are the advantages and disadvantages of each approach?

   Token-ring-based approach has less latency because it does not have to wait for the second Pi to return the message. This will be a disadvantage for the polling approach, as it has more processes before sending it can send its data. This is the same for data freshness, and the token ring method cycles quicker than the polling method, which allows the Pi to plot more fresh data. Since these data would be deleted soon after the plotting is done, the resources used would be less for the token-ring-based approach. Overall, the token-ring-based approach is more beneficial than the polling approach for its latency, freshness, and the resources used.

b. In what scenarios would it be beneficial to use a polling approach?

   The polling approach has a benefit where it could create a more simple networking system. The polling also allows precise control over when and how the data is fetched, which allows the polling to be beneficial under scenarios where the data needs a more complex control and wants to poll the data from other IoT devices directly.

c. In what scenarios would it be beneficial to use a token-ring-based approach?

   The token-ring-based approach would be beneficial under real-time or time-sensitive scenarios since it is a good approach for reducing time latency. This is also a good approach for this lab, where three Raspberry Pis had to send each other messages and plot the data.

6. Conclusion: Summarize the takeaways you learned in this lab.

   Throughout the lab, I have learned the ways to apply the client-server model and the token-passing-based model using three Raspberry Pis in its Ad-hoc mode. I have learned how to use socket programming and handle communication. Another important part of this lab was creating the system design and logic visualization before working on the code programming. This has helped in keeping the minds organized while we programmed the system we wanted it to have. In the section 2, we have implemented the polling, and in the section 3, the token-ring approach. Thinking of the corner cases that may occur in each approach and logging out where it may cause an error has helped in finding bugs more easily and debugging quicker. Overall, I was able to understand the trade-offs between different networking approaches, the significance of error handling, and the importance of choosing an approach aligned with specific system requirements.