Phase 1 (Parts A & B): UDP Programming

Authors:

- **Layth Mallak**

Objectectives:

Part A:

- **Build a simple UDP client and server to demonstrate how datagrams send and receive and how they request and respond.**

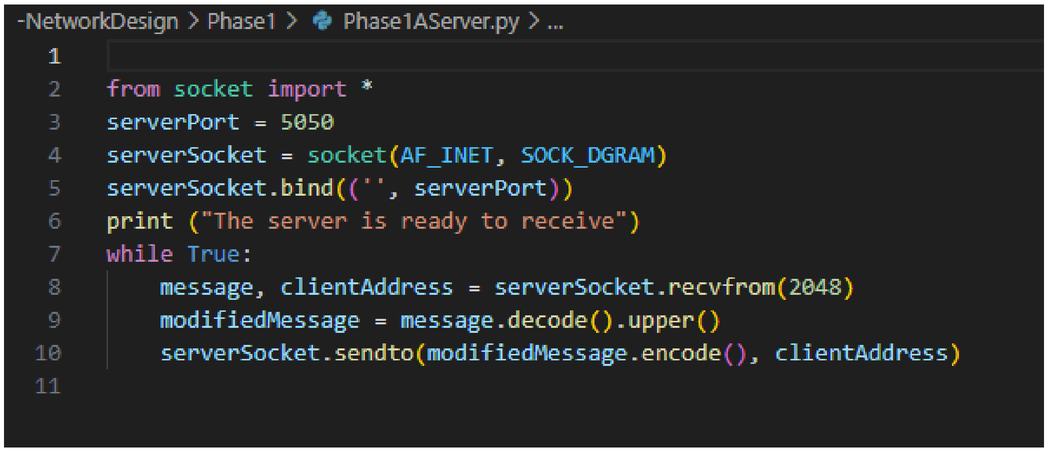
Part B:

- **Build a simple file transfer over UDP using RDT 1.0 protocol.**

Code Explanation:

Part A:

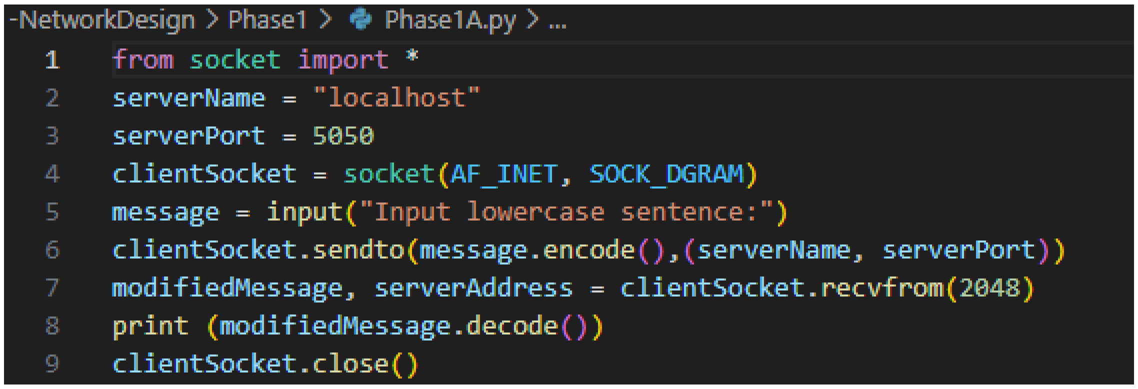
- **Phase1AServer.py**


-NetworkDesign > Phase1 > Phase1AServer.py > ...

```

from socket import *
serverPort = 5050
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('', serverPort))
print ("The server is ready to receive")
while True:
    message, clientAddress = serverSocket.recvfrom(2048)
    modifiedMessage = message.decode().upper()
    serverSocket.sendto(modifiedMessage.encode(), clientAddress)

```

-
- **Uses the socket library to listen for datagrams. Once datagrams are received, the message is made uppercase and sent to the client.**

- **Phase1A.py**

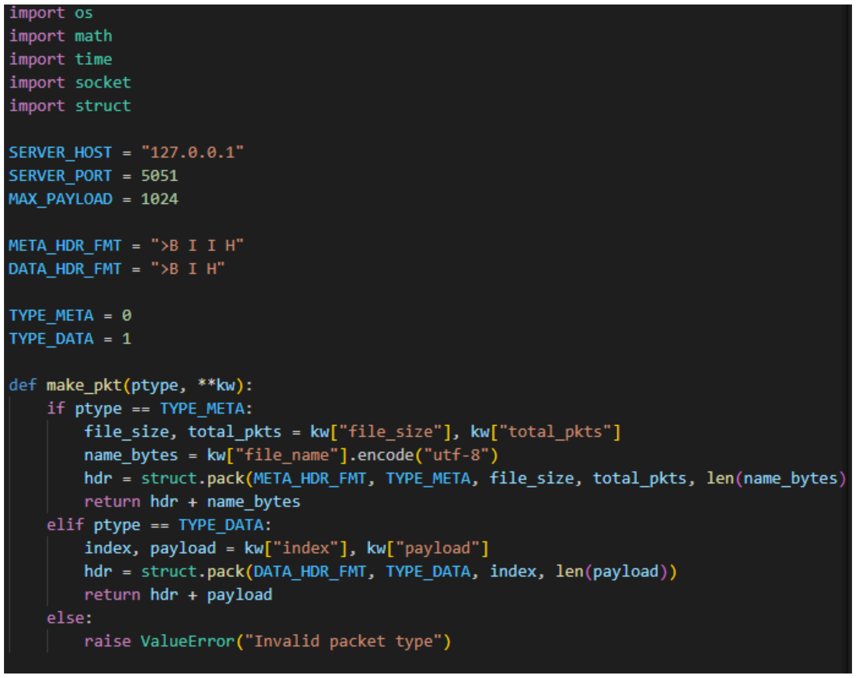


```python
from socket import *
serverName = "localhost"
serverPort = 5050
clientSocket = socket(AF_INET, SOCK_DGRAM)
message = input("Input lowercase sentence:")
clientSocket.sendto(message.encode(),(serverName, serverPort))
modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
print (modifiedMessage.decode())
clientSocket.close()
```

-
- **Uses the socket library to open a socket and send a datagram through that socket. Once sent, the client waits for a response from the server. Once the response arrives, the socket is closed.**

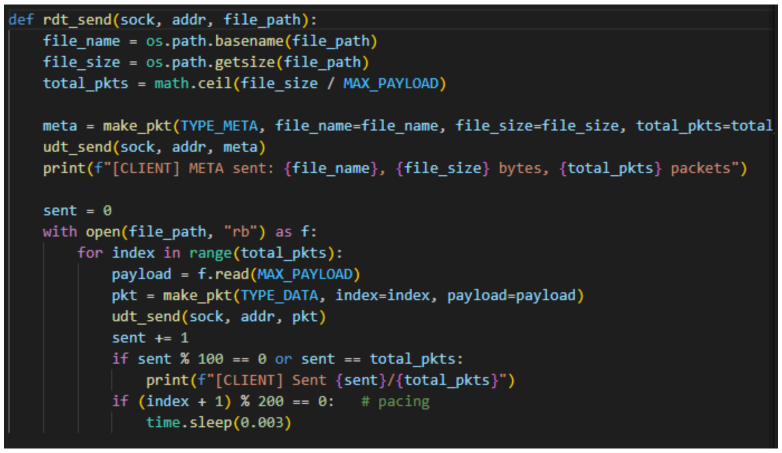
Part B:

- **phase1Bclient.py**



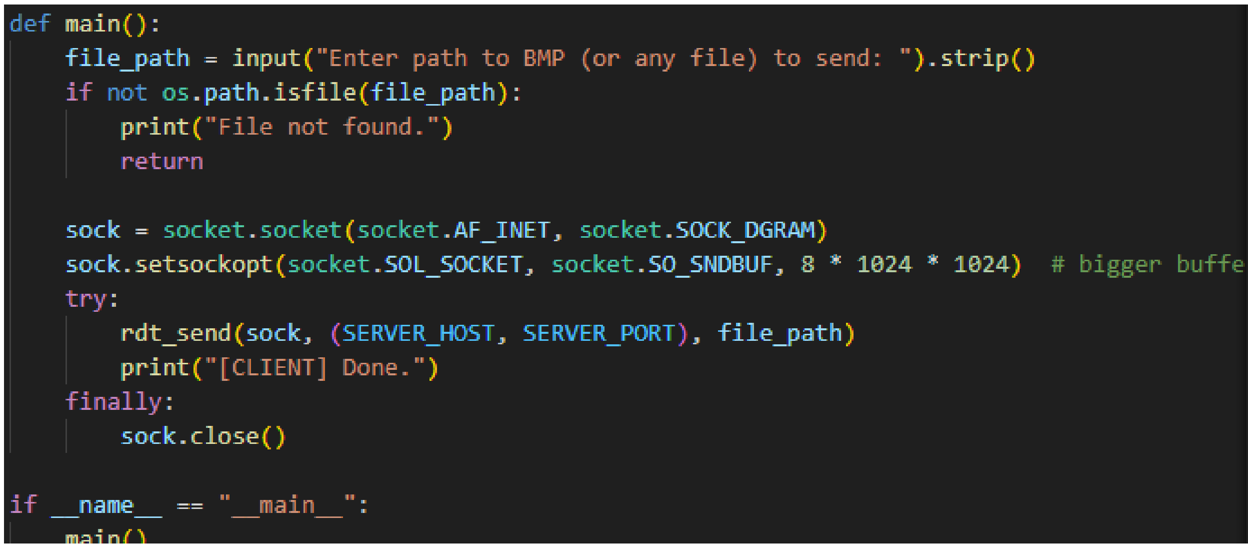
```python
import os
import math
import time
import socket
import struct

SERVER_HOST = "127.0.0.1"
SERVER_PORT = 5051
MAX_PAYLOAD = 1024

META_HDR_FMT = ">B I I H"
DATA_HDR_FMT = ">B I H"

TYPE_META = 0
TYPE_DATA = 1

def make_pkt(ptype, **kw):
    if ptype == TYPE_META:
        file_size, total_pkts = kw["file_size"], kw["total_pkts"]
        name_bytes = kw["file_name"].encode("utf-8")
        hdr = struct.pack(META_HDR_FMT, TYPE_META, file_size, total_pkts, len(name_bytes)
        return hdr + name_bytes
    elif ptype == TYPE_DATA:
        index, payload = kw["index"], kw["payload"]
        hdr = struct.pack(DATA_HDR_FMT, TYPE_DATA, index, len(payload))
        return hdr + payload
    else:
        raise ValueError("Invalid packet type")
```

-
- **Make_pkt takes a file and extractts its data, sorting it into either of two types. The meta type stores filesize, total packets, and file name. The data type stores its index and payload.**

- 

```python
def rdt_send(sock, addr, file_path):
    file_name = os.path.basename(file_path)
    file_size = os.path.getsize(file_path)
    total_pkts = math.ceil(file_size / MAX_PAYLOAD)

    meta = make_pkt(TYPE_META, file_name=file_name, file_size=file_size, total_pkts=total
    udt_send(sock, addr, meta)
    print(f"[CLIENT] META sent: {file_name}, {file_size} bytes, {total_pkts} packets")

    sent = 0
    with open(file_path, "rb") as f:
        for index in range(total_pkts):
            payload = f.read(MAX_PAYLOAD)
            pkt = make_pkt(TYPE_DATA, index=index, payload=payload)
            udt_send(sock, addr, pkt)
            sent += 1
            if sent % 100 == 0 or sent == total_pkts:
                print(f"[CLIENT] Sent {sent}/{total_pkts}")
            if (index + 1) % 200 == 0:   # pacing
                time.sleep(0.003)
```

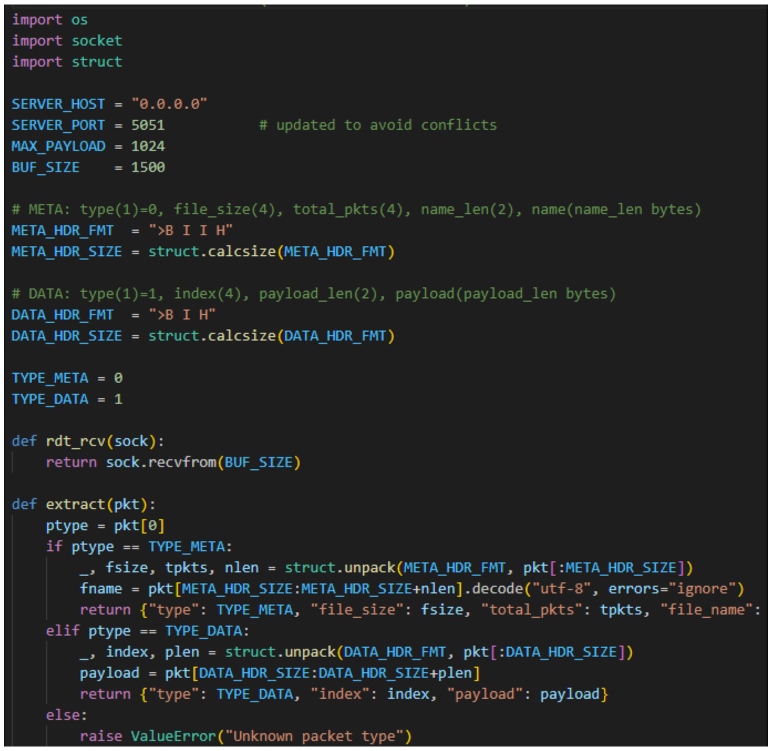
- **Rdt_send is the sending logic. It reads the file from the diskand sends the meta packet to provide file details. The file is split into chunks of 1024 bytes, which are sent as the datapacket and its index. There is a pause every 200 packets to avoid overwhelming the buffers.**

- 

```python
def main():
    file_path = input("Enter path to BMP (or any file) to send: ").strip()
    if not os.path.isfile(file_path):
        print("File not found.")
        return

    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_SNDBUF, 8 * 1024 * 1024)  # bigger buffe
    try:
        rdt_send(sock, (SERVER_HOST, SERVER_PORT), file_path)
        print("[CLIENT] Done.")
    finally:
        sock.close()

if __name__ == "__main__":
    main()
```

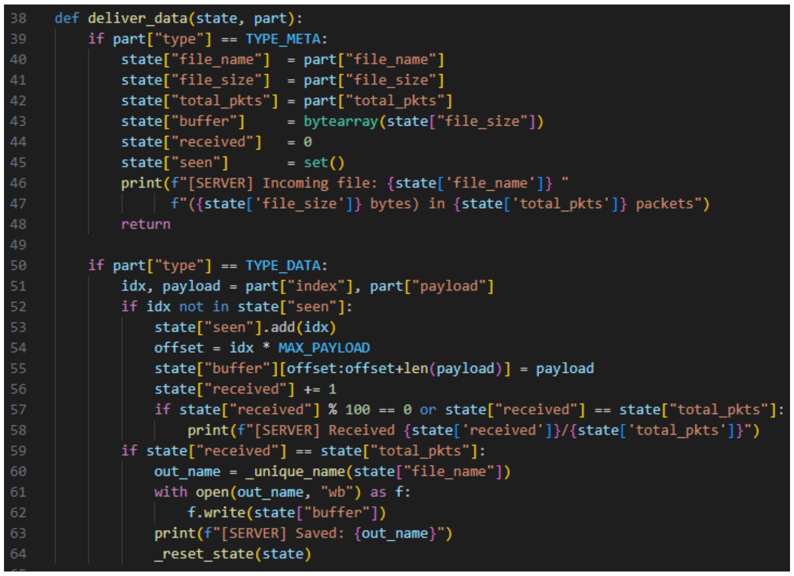
- **The main function prompts the user for the path of the file. It then creates a socket. Calls rdt_send and closes the socket once it is finished.**

- **phase1BServer.py**



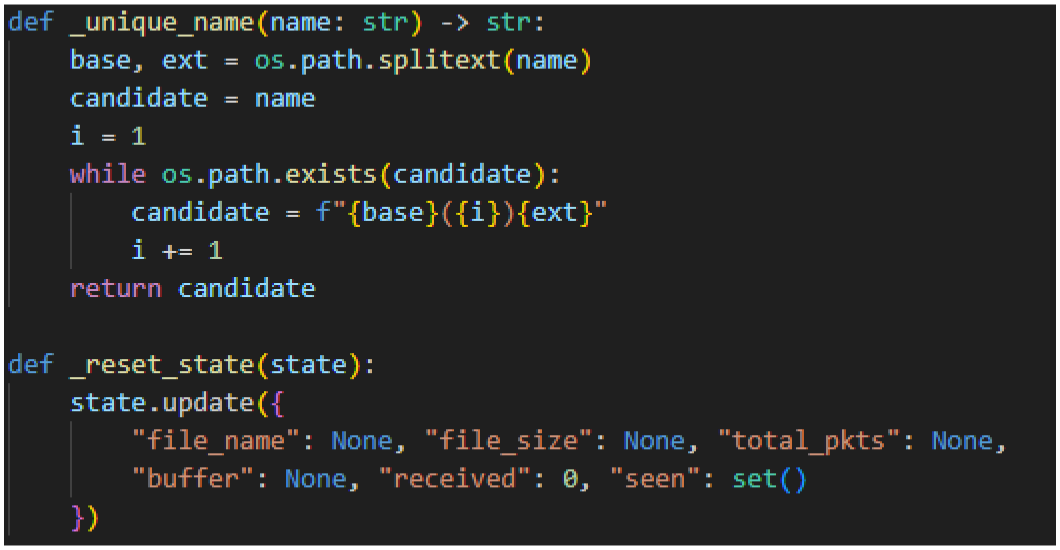
```python
import os
import socket
import struct

SERVER_HOST = "0.0.0.0"
SERVER_PORT = 5051          # updated to avoid conflicts
MAX_PAYLOAD = 1024
BUF_SIZE    = 1500

# META: type(1)=0, file_size(4), total_pkts(4), name_len(2), name(name_len bytes)
META_HDR_FMT  = ">B I I H"
META_HDR_SIZE = struct.calcsize(META_HDR_FMT)

# DATA: type(1)=1, index(4), payload_len(2), payload(payload_len bytes)
DATA_HDR_FMT  = ">B I H"
DATA_HDR_SIZE = struct.calcsize(DATA_HDR_FMT)

TYPE_META = 0
TYPE_DATA = 1

def rdt_rcv(sock):
    return sock.recvfrom(BUF_SIZE)

def extract(pkt):
    ptype = pkt[0]
    if ptype == TYPE_META:
        _, fsize, tpkts, nlen = struct.unpack(META_HDR_FMT, pkt[:META_HDR_SIZE])
        fname = pkt[META_HDR_SIZE:META_HDR_SIZE+nlen].decode("utf-8", errors="ignore")
        return {"type": TYPE_META, "file_size": fsize, "total_pkts": tpkts, "file_name":
    elif ptype == TYPE_DATA:
        _, index, plen = struct.unpack(DATA_HDR_FMT, pkt[:DATA_HDR_SIZE])
        payload = pkt[DATA_HDR_SIZE:DATA_HDR_SIZE+plen]
        return {"type": TYPE_DATA, "index": index, "payload": payload}
    else:
        raise ValueError("Unknown packet type")
```

- This code snippet reads the first byte of a packet to determine its type. It will then unpack file size, total packets, and filename if it is type meta. It will unpack index and payload data if type data.



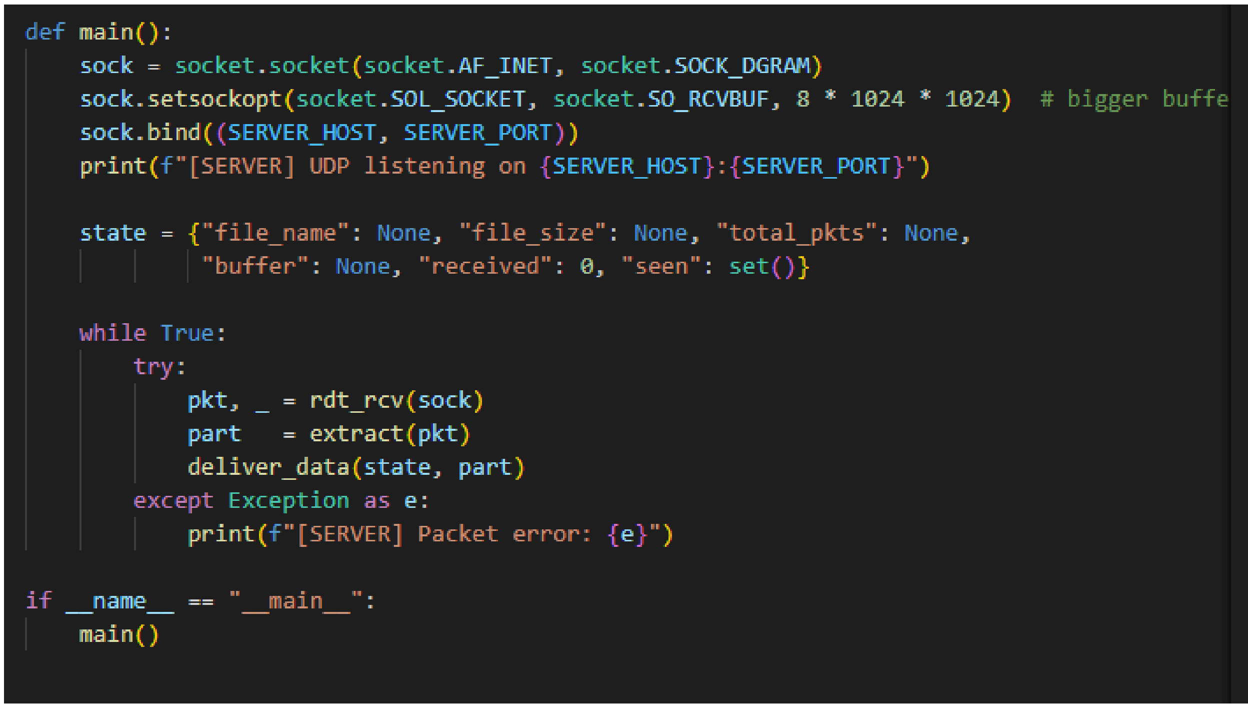
```python
def deliver_data(state, part):
    if part["type"] == TYPE_META:
        state["file_name"]  = part["file_name"]
        state["file_size"]  = part["file_size"]
        state["total_pkts"] = part["total_pkts"]
        state["buffer"]     = bytearray(state["file_size"])
        state["received"]   = 0
        state["seen"]       = set()
        print(f"[SERVER] Incoming file: {state['file_name']} "
              f"({state['file_size']} bytes) in {state['total_pkts']} packets")
        return

    if part["type"] == TYPE_DATA:
        idx, payload = part["index"], part["payload"]
        if idx not in state["seen"]:
            state["seen"].add(idx)
            offset = idx * MAX_PAYLOAD
            state["buffer"][offset:offset+len(payload)] = payload
            state["received"] += 1
            if state["received"] % 100 == 0 or state["received"] == state["total_pkts"]:
                print(f"[SERVER] Received {state['received']}/{state['total_pkts']}")
        if state["received"] == state["total_pkts"]:
            out_name = _unique_name(state["file_name"])
            with open(out_name, "wb") as f:
                f.write(state["buffer"])
            print(f"[SERVER] Saved: {out_name}")
            _reset_state(state)
```

- This function delivers the data back to the client. If it is type meta, it will prepare the states of the metadata. If it is type data it will track the packets received and store them. It will print progress every 100 packets. When all the packets are received, the file is written and the states reset.



```python
def _unique_name(name: str) -> str:
    base, ext = os.path.splitext(name)
    candidate = name
    i = 1
    while os.path.exists(candidate):
        candidate = f"{base}({i}){ext}"
        i += 1
    return candidate

def _reset_state(state):
    state.update({
        "file_name": None, "file_size": None, "total_pkts": None,
        "buffer": None, "received": 0, "seen": set()
    })
```

-
- These are utility functions to help keep the process bug free. _unique_name ensures that files don't overwrite each other by renaming the new file. _*Reset*_state clears memory so the server can take another file.

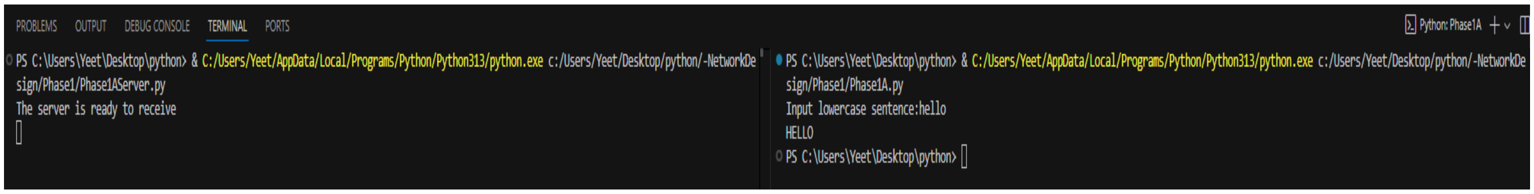


```python
def main():
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_RCVBUF, 8 * 1024 * 1024)  # bigger buffe
    sock.bind((SERVER_HOST, SERVER_PORT))
    print(f"[SERVER] UDP listening on {SERVER_HOST}:{SERVER_PORT}")

    state = {"file_name": None, "file_size": None, "total_pkts": None,
             "buffer": None, "received": 0, "seen": set()}

    while True:
        try:
            pkt, _ = rdt_rcv(sock)
            part   = extract(pkt)
            deliver_data(state, part)
        except Exception as e:
            print(f"[SERVER] Packet error: {e}")

if __name__ == "__main__":
    main()
```

-
- The main loop creates a udp socket that is bound to port 5051. The server will in a loop, wait for and receive packets, decode them, and deliver them. The server will run indefinitely and handle multiple file transfers.

Code Execution:

Part A:

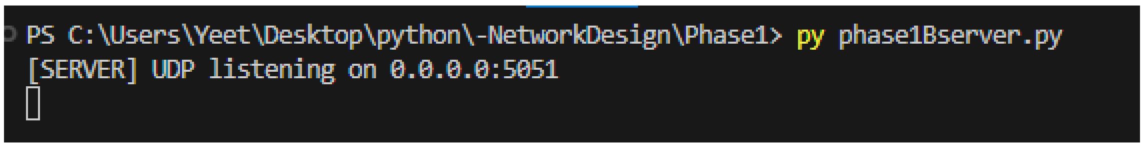


```
PS C:\Users\Yeet\Desktop\python> & C:/Users/Yeet/AppData/Local/Programs/Python/Python313/python.exe c:/Users/Yeet/Desktop/python/-NetworkDe
sign/Phase1/Phase1AServer.py
The server is ready to receive
```

```
PS C:\Users\Yeet\Desktop\python> & C:/Users/Yeet/AppData/Local/Programs/Python/Python313/python.exe c:/Users/Yeet/Desktop/python/-NetworkDe
sign/Phase1/Phase1A.py
Input lowercase sentence:hello
HELLO
PS C:\Users\Yeet\Desktop\python>
```

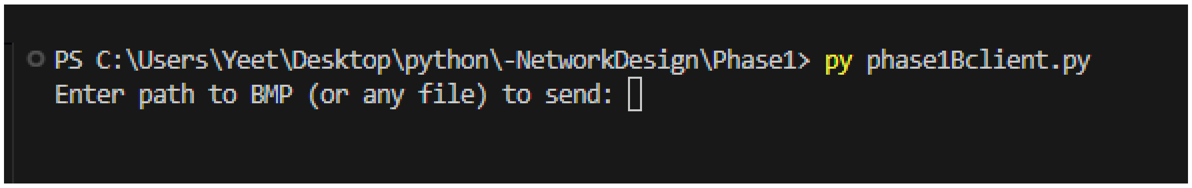
- **In this screen shot, the server is run with:**
- **py Phase1AServer.py**
- **The server becomes ready to receive the message.**
- **The client is run with:**
- **py Phase1A.py**
- **The client sends a message to the server which is then converted uppercase and displayed.**

Part B:



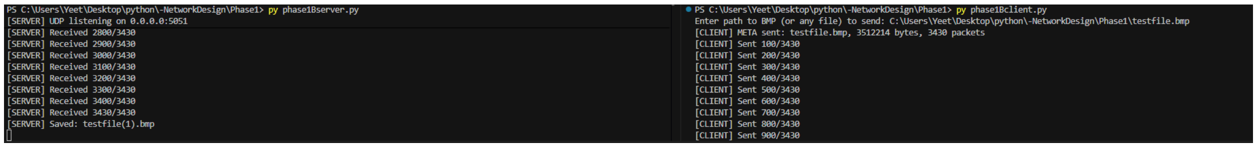
```
PS C:\Users\Yeet\Desktop\python\-NetworkDesign\Phase1> py phase1Bserver.py
[SERVER] UDP listening on 0.0.0.0:5051
```

- **In this screenshot the server is run with py phase1Bserver.py**
- **The server activates and begins to wait for any packets on port 5051.**



```
PS C:\Users\Yeet\Desktop\python\-NetworkDesign\Phase1> py phase1Bclient.py
Enter path to BMP (or any file) to send:
```

-
- **The client is activated with py phase1BClient.py**
- **The client requests a file path for the file that is going to be sent.**
-



```
PS C:\Users\Yeet\Desktop\python\-NetworkDesign\Phase1> py phase1Bserver.py
[SERVER] UDP listening on 0.0.0.0:5051
[SERVER] Received 2800/3430
[SERVER] Received 2900/3430
[SERVER] Received 3000/3430
[SERVER] Received 3100/3430
[SERVER] Received 3200/3430
[SERVER] Received 3300/3430
[SERVER] Received 3400/3430
[SERVER] Received 3430/3430
[SERVER] Saved: testfile(1).bmp
```

```
PS C:\Users\Yeet\Desktop\python\-NetworkDesign\Phase1> py phase1Bclient.py
Enter path to BMP (or any file) to send: C:\Users\Yeet\Desktop\python\-NetworkDesign\Phase1\testfile.bmp
[CLIENT] META sent: testfile.bmp, 3512214 bytes, 3430 packets
[CLIENT] Sent 100/3430
[CLIENT] Sent 200/3430
[CLIENT] Sent 300/3430
[CLIENT] Sent 400/3430
[CLIENT] Sent 500/3430
[CLIENT] Sent 600/3430
[CLIENT] Sent 700/3430
[CLIENT] Sent 800/3430
[CLIENT] Sent 900/3430
```

- **The file path is given and the progress for the data transfer is shown. Once the transfer is complete, the file is saved in the same folder as the server.**