CS Computer Networks Theory Assignment

Abhishek Josyula

Roll: 200010021

Link to video: demo

Running the files.

1. First run manager.py

2. Then run peer.py

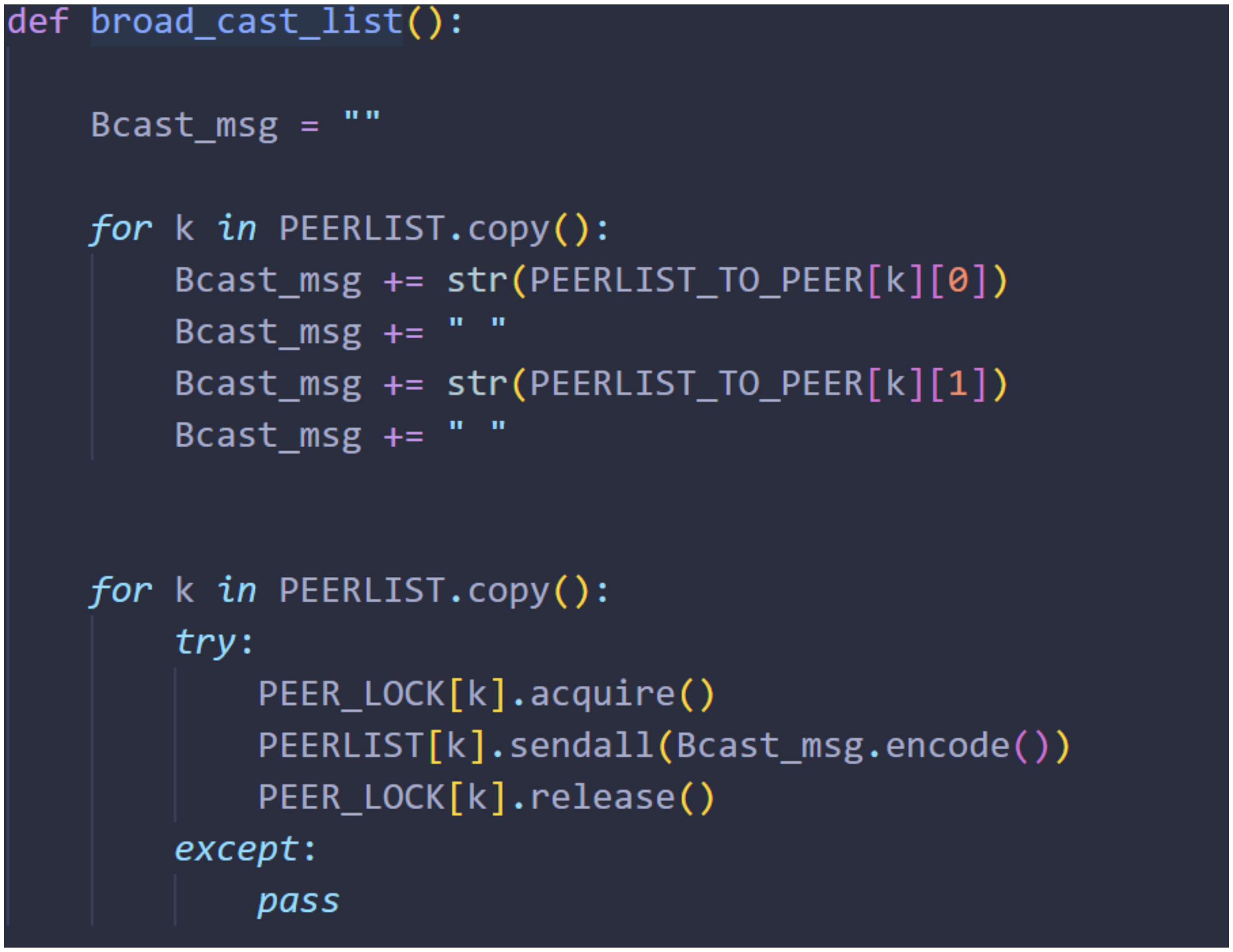
3. Add files to peer.py

4. Try GET from a peer to get a list of available peers.

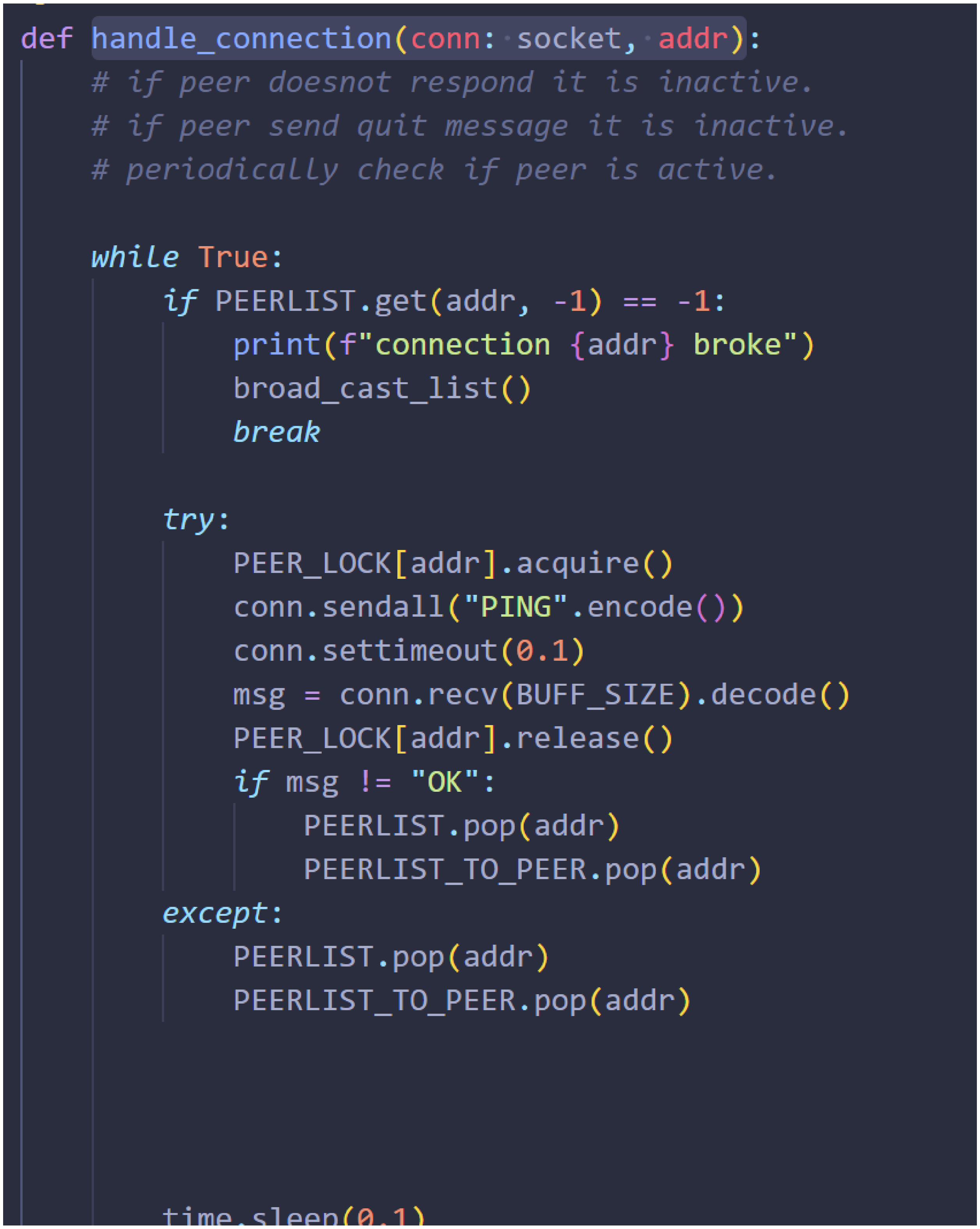
5. Try DOWN from the peer to download file.

Manager.py

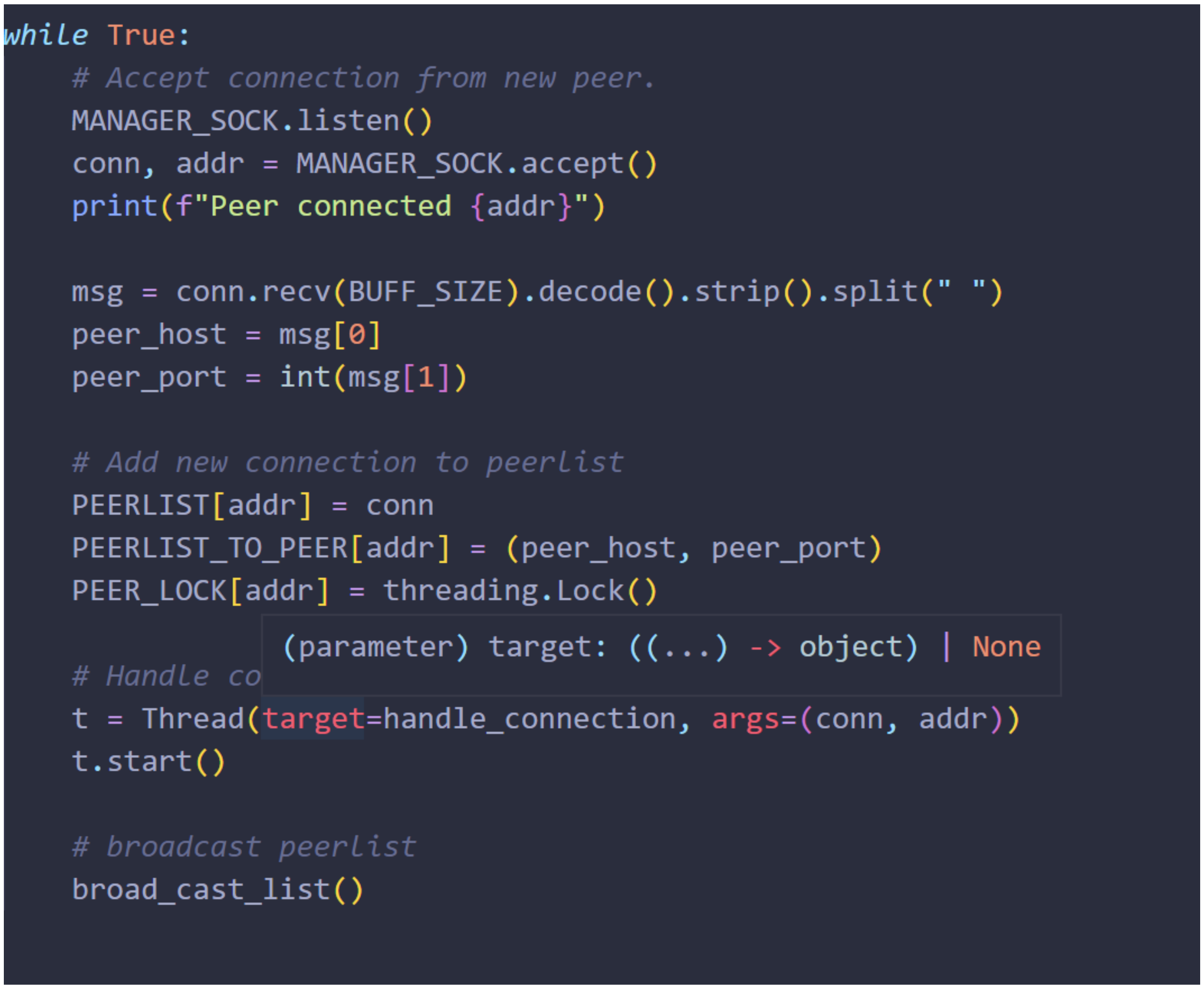
- A manager is an always on server, which maintains the list of currently active peers across the network at all times.
- When a peer connects it updates the list and broadcasts.
- Update is handled by the main thread and,
- broad_cast_list() broadcasts the list.
- handle_connection(conn: socket, addr) manages the connection between manager and peer
- If the peer does not respond to the manager's ping. It disconnects the peer



```python
def broad_cast_list():

    Bcast_msg = ""

    for k in PEERLIST.copy():
        Bcast_msg += str(PEERLIST_TO_PEER[k][0])
        Bcast_msg += " "
        Bcast_msg += str(PEERLIST_TO_PEER[k][1])
        Bcast_msg += " "


    for k in PEERLIST.copy():
        try:
            PEER_LOCK[k].acquire()
            PEERLIST[k].sendall(Bcast_msg.encode())
            PEER_LOCK[k].release()
        except:
            pass
```



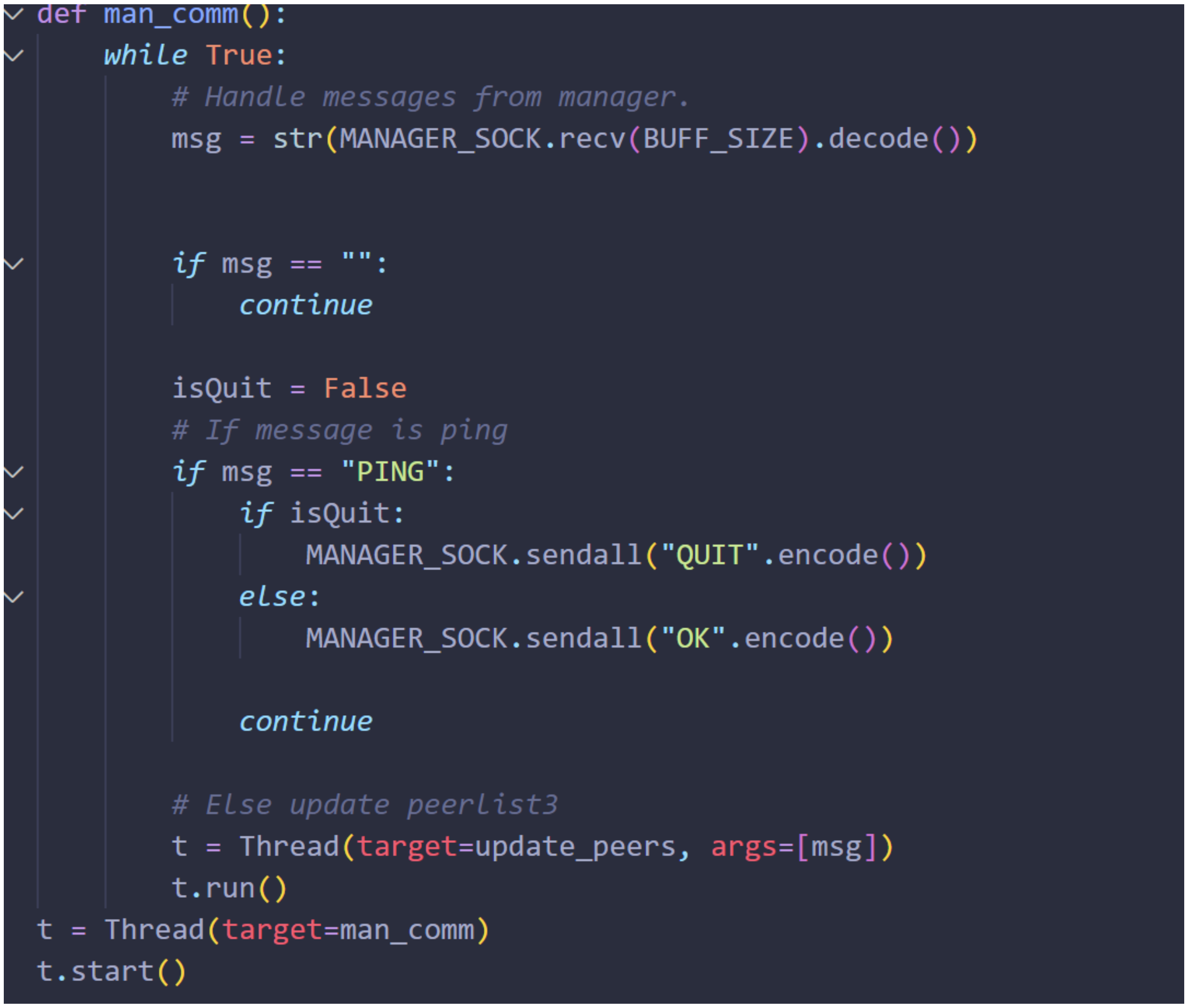
```python
def handle_connection(conn: socket, addr):
    # if peer doesnot respond it is inactive.
    # if peer send quit message it is inactive.
    # periodically check if peer is active.

    while True:
        if PEERLIST.get(addr, -1) == -1:
            print(f"connection {addr} broke")
            broad_cast_list()
            break

        try:
            PEER_LOCK[addr].acquire()
            conn.sendall("PING".encode())
            conn.settimeout(0.1)
            msg = conn.recv(BUFF_SIZE).decode()
            PEER_LOCK[addr].release()
            if msg != "OK":
                PEERLIST.pop(addr)
                PEERLIST_TO_PEER.pop(addr)
        except:
            PEERLIST.pop(addr)
            PEERLIST_TO_PEER.pop(addr)




        time.sleep(0.1)
```



```python
while True:
    # Accept connection from new peer.
    MANAGER_SOCK.listen()
    conn, addr = MANAGER_SOCK.accept()
    print(f"Peer connected {addr}")

    msg = conn.recv(BUFF_SIZE).decode().strip().split(" ")
    peer_host = msg[0]
    peer_port = int(msg[1])

    # Add new connection to peerlist
    PEERLIST[addr] = conn
    PEERLIST_TO_PEER[addr] = (peer_host, peer_port)
    PEER_LOCK[addr] = threading.Lock()
                    (parameter) target: ((...) -> object) | None
    # Handle co
    t = Thread(target=handle_connection, args=(conn, addr))
    t.start()

    # broadcast peerlist
    broad_cast_list()
```

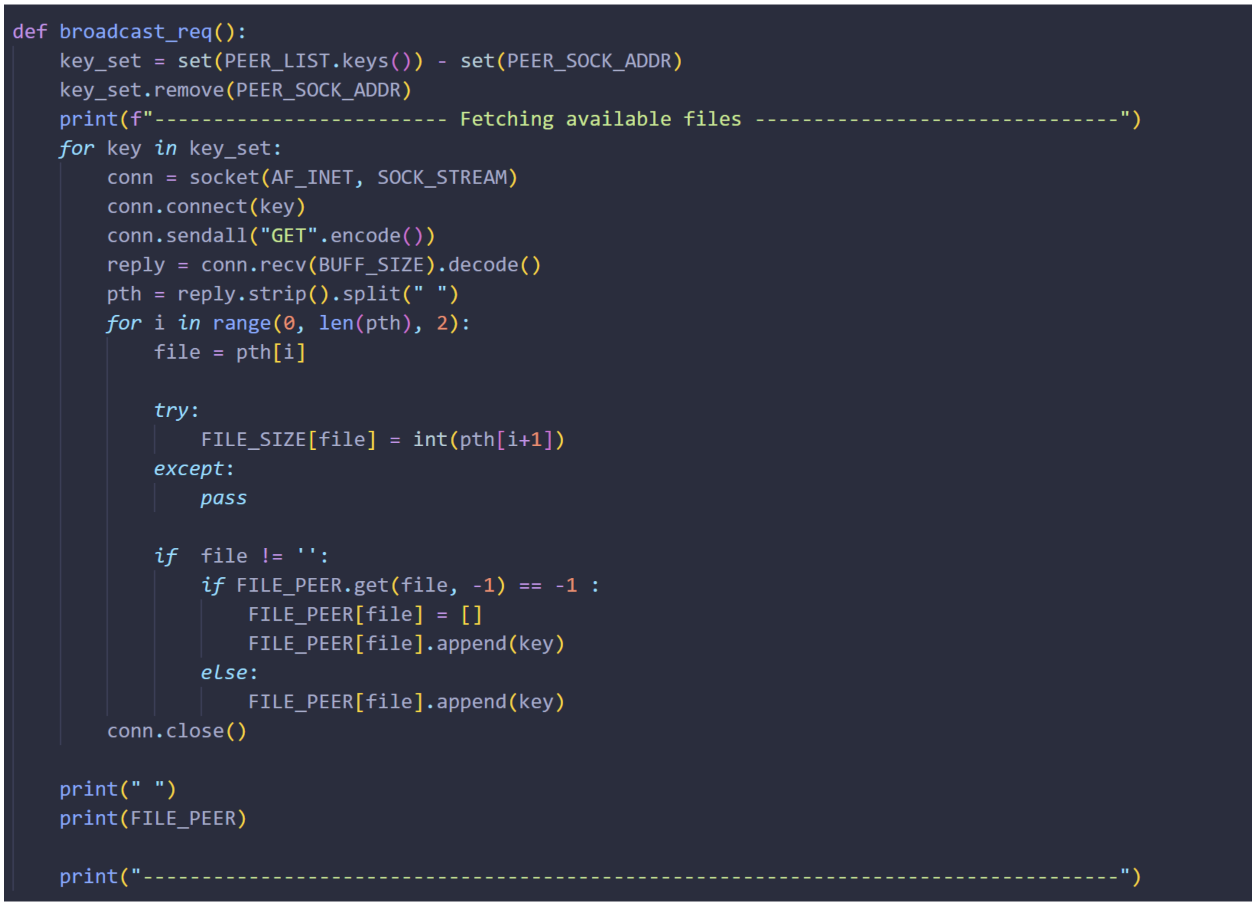
Peer.py

- Peer pings the manager and saves the list sent by manager.



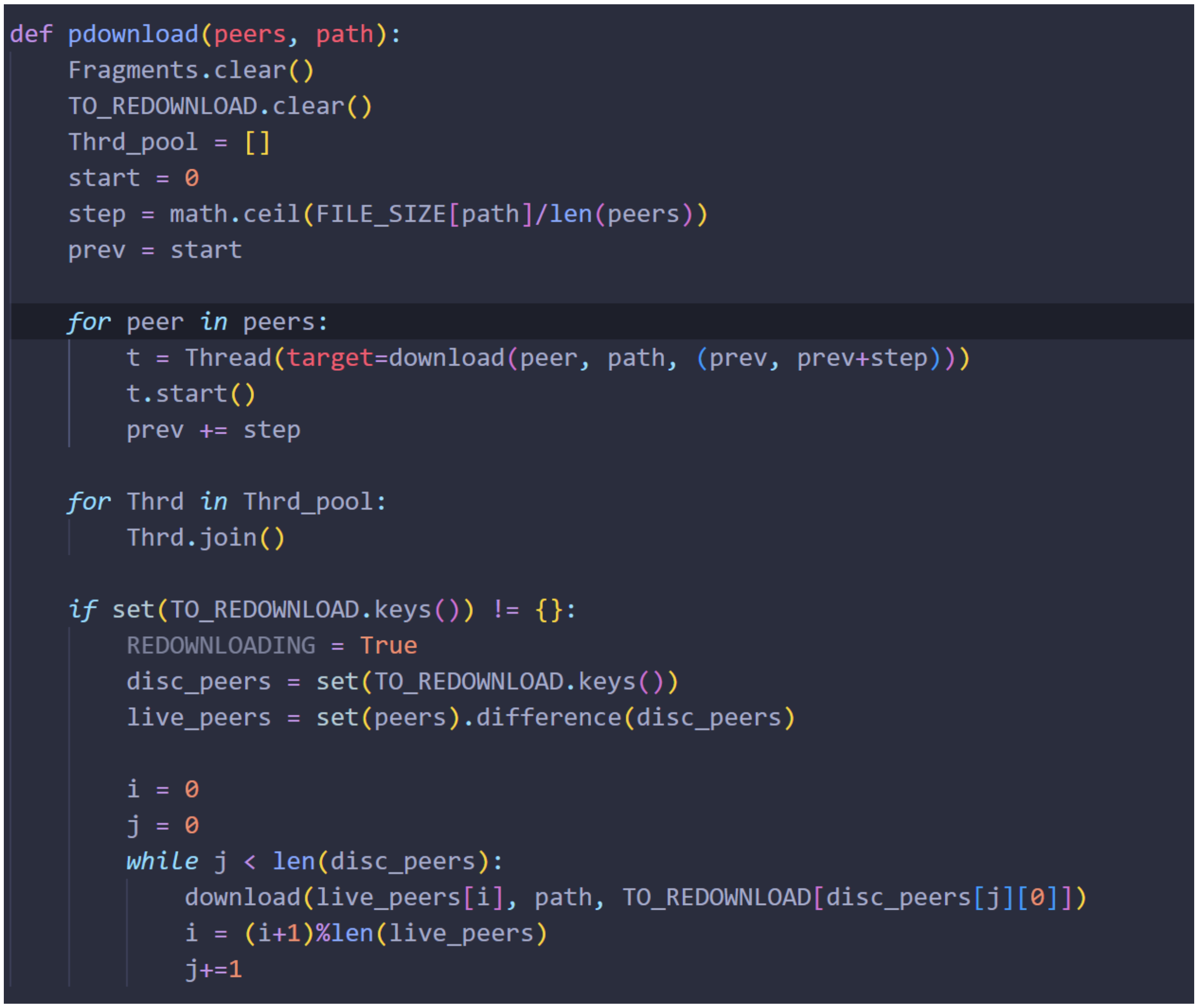
```python
def man_comm():
    while True:
        # Handle messages from manager.
        msg = str(MANAGER_SOCK.recv(BUFF_SIZE).decode())


        if msg == "":
            continue

        isQuit = False
        # If message is ping
        if msg == "PING":
            if isQuit:
                MANAGER_SOCK.sendall("QUIT".encode())
            else:
                MANAGER_SOCK.sendall("OK".encode())

            continue

        # Else update peerlist3
        t = Thread(target=update_peers, args=[msg])
        t.run()
t = Thread(target=man_comm)
t.start()
```

- TYPE ADD to add a file to available files.
- TYPE GET to list of available files from peers.



```python
def broadcast_req():
    key_set = set(PEER_LIST.keys()) - set(PEER_SOCK_ADDR)
    key_set.remove(PEER_SOCK_ADDR)
    print(f"------------------------- Fetching available files -------------------------------")
    for key in key_set:
        conn = socket(AF_INET, SOCK_STREAM)
        conn.connect(key)
        conn.sendall("GET".encode())
        reply = conn.recv(BUFF_SIZE).decode()
        pth = reply.strip().split(" ")
        for i in range(0, len(pth), 2):
            file = pth[i]

            try:
                FILE_SIZE[file] = int(pth[i+1])
            except:
                pass

            if  file != '':
                if FILE_PEER.get(file, -1) == -1 :
                    FILE_PEER[file] = []
                    FILE_PEER[file].append(key)
                else:
                    FILE_PEER[file].append(key)
        conn.close()

    print(" ")
    print(FILE_PEER)

    print("-----------------------------------------------------------------------------------")
```

- DOWN downloads the files in parallel.



```python
def pdownload(peers, path):
    Fragments.clear()
    TO_REDOWNLOAD.clear()
    Thrd_pool = []
    start = 0
    step = math.ceil(FILE_SIZE[path]/len(peers))
    prev = start

    for peer in peers:
        t = Thread(target=download(peer, path, (prev, prev+step)))
        t.start()
        prev += step

    for Thrd in Thrd_pool:
        Thrd.join()

    if set(TO_REDOWNLOAD.keys()) != {}:
        REDOWNLOADING = True
        disc_peers = set(TO_REDOWNLOAD.keys())
        live_peers = set(peers).difference(disc_peers)

        i = 0
        j = 0
        while j < len(disc_peers):
            download(live_peers[i], path, TO_REDOWNLOAD[disc_peers[j][0]])
            i = (i+1)%len(live_peers)
            j+=1
```