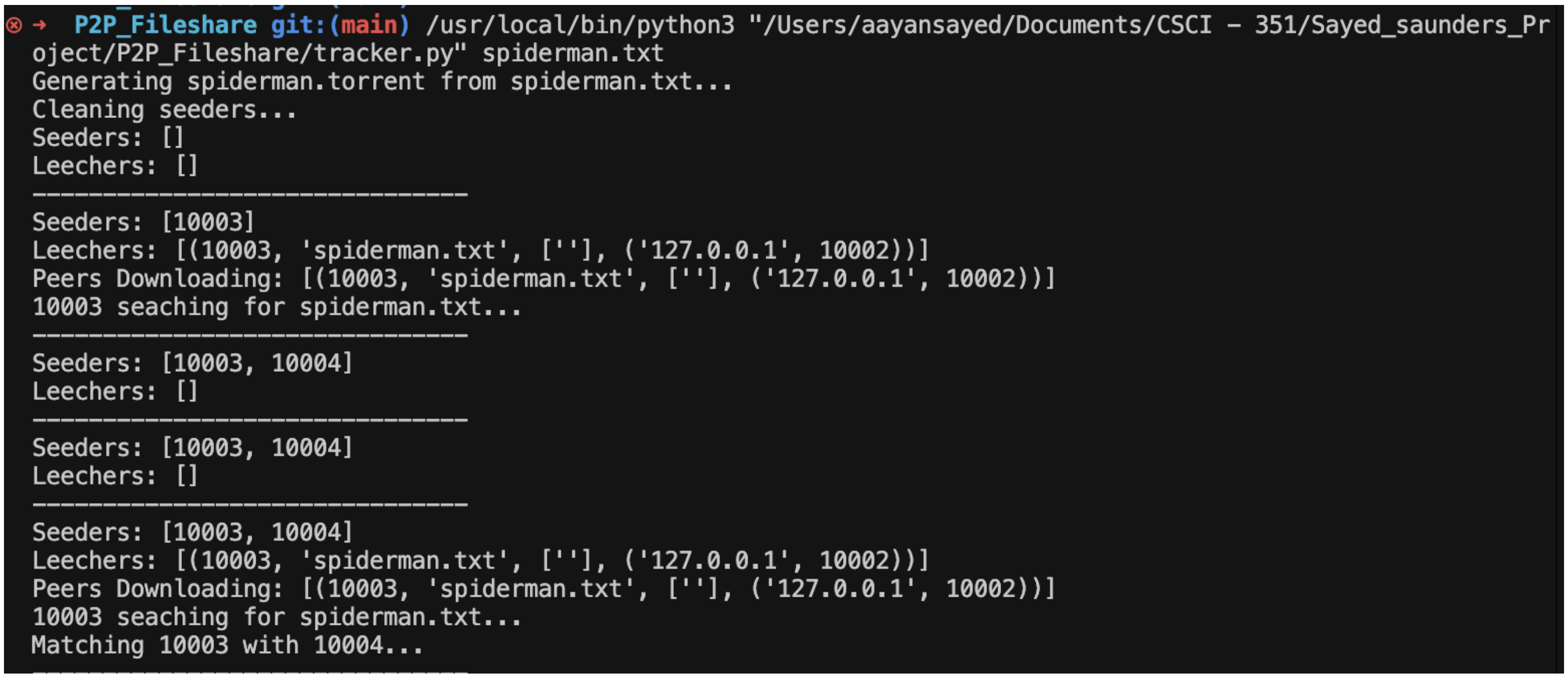
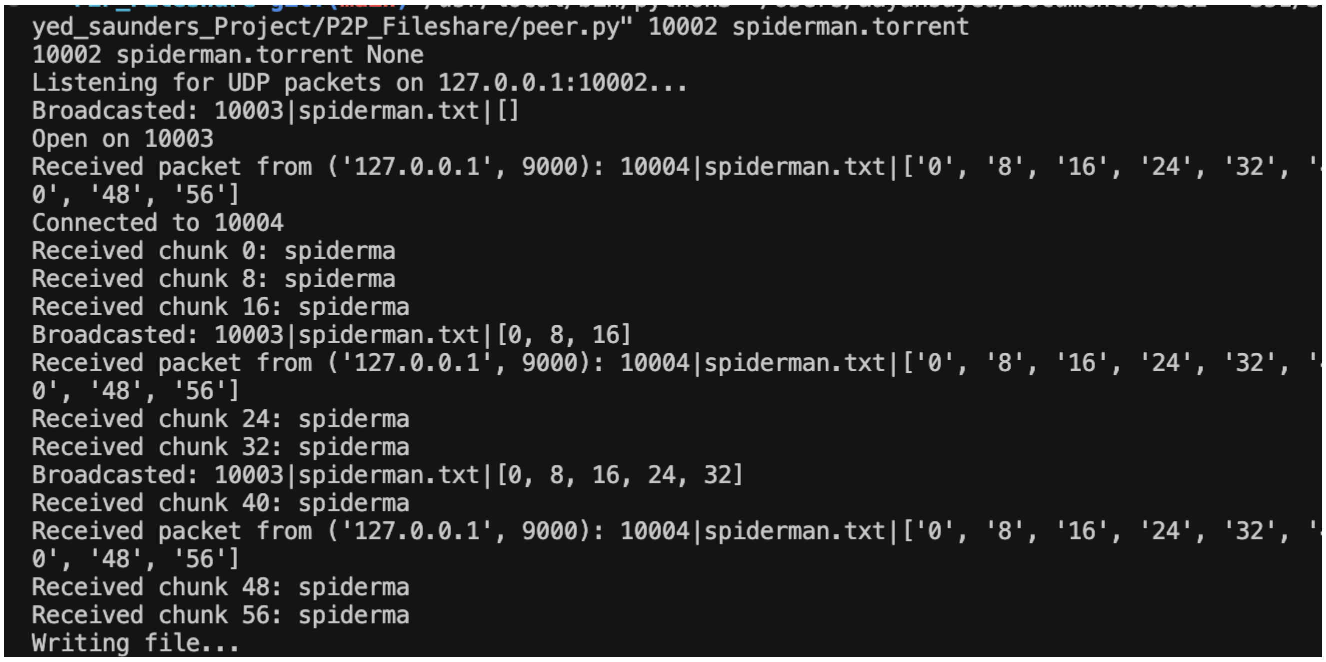
1) **Peer Discovery:** Implement a mechanism for peers to discover each other.



```
⊗ → P2P_Fileshare git:(main) /usr/local/bin/python3 "/Users/aayansayed/Documents/CSCI - 351/Sayed_saunders_Pr
  oject/P2P_Fileshare/tracker.py" spiderman.txt
  Generating spiderman.torrent from spiderman.txt...
  Cleaning seeders...
  Seeders: []
  Leechers: []
  -------------------------------
  Seeders: [10003]
  Leechers: [(10003, 'spiderman.txt', [''], ('127.0.0.1', 10002))]
  Peers Downloading: [(10003, 'spiderman.txt', [''], ('127.0.0.1', 10002))]
  10003 seaching for spiderman.txt...
  -------------------------------
  Seeders: [10003, 10004]
  Leechers: []
  -------------------------------
  Seeders: [10003, 10004]
  Leechers: []
  -------------------------------
  Seeders: [10003, 10004]
  Leechers: [(10003, 'spiderman.txt', [''], ('127.0.0.1', 10002))]
  Peers Downloading: [(10003, 'spiderman.txt', [''], ('127.0.0.1', 10002))]
  10003 seaching for spiderman.txt...
  Matching 10003 with 10004...
```

Here when the program tracker is run the tracker begins discovering peers. Once the tracker sees that there is 1 peer which has the file(peer on port 10004) and another that needs it (peer on port 10003), it will match them. This way peers can discover each other through the tracker and then connect directly to other peers.

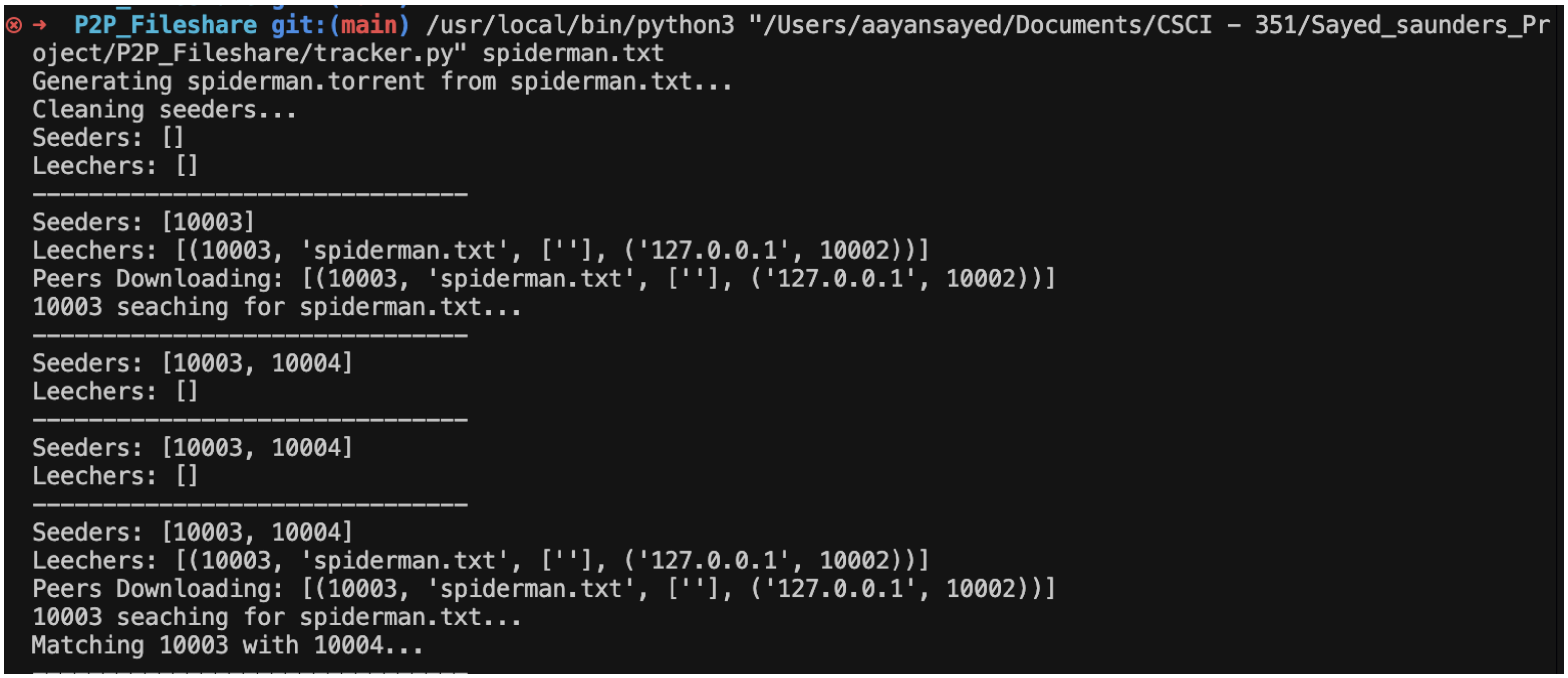
2) **Indexing and Searching:** Maintain a list of available data on each peer.



```
  yed_saunders_Project/P2P_Fileshare/peer.py" 10002 spiderman.torrent
  10002 spiderman.torrent None
  Listening for UDP packets on 127.0.0.1:10002...
  Broadcasted: 10003|spiderman.txt|[]
  Open on 10003
  Received packet from ('127.0.0.1', 9000): 10004|spiderman.txt|['0', '8', '16', '24', '32', '
  0', '48', '56']
  Connected to 10004
  Received chunk 0: spiderma
  Received chunk 8: spiderma
  Received chunk 16: spiderma
  Broadcasted: 10003|spiderman.txt|[0, 8, 16]
  Received packet from ('127.0.0.1', 9000): 10004|spiderman.txt|['0', '8', '16', '24', '32', '
  0', '48', '56']
  Received chunk 24: spiderma
  Received chunk 32: spiderma
  Broadcasted: 10003|spiderman.txt|[0, 8, 16, 24, 32]
  Received chunk 40: spiderma
  Received packet from ('127.0.0.1', 9000): 10004|spiderman.txt|['0', '8', '16', '24', '32', '
  0', '48', '56']
  Received chunk 48: spiderma
  Received chunk 56: spiderma
  Writing file...
```

Each peers maintains and can get a list of available data from each peer via the tracker

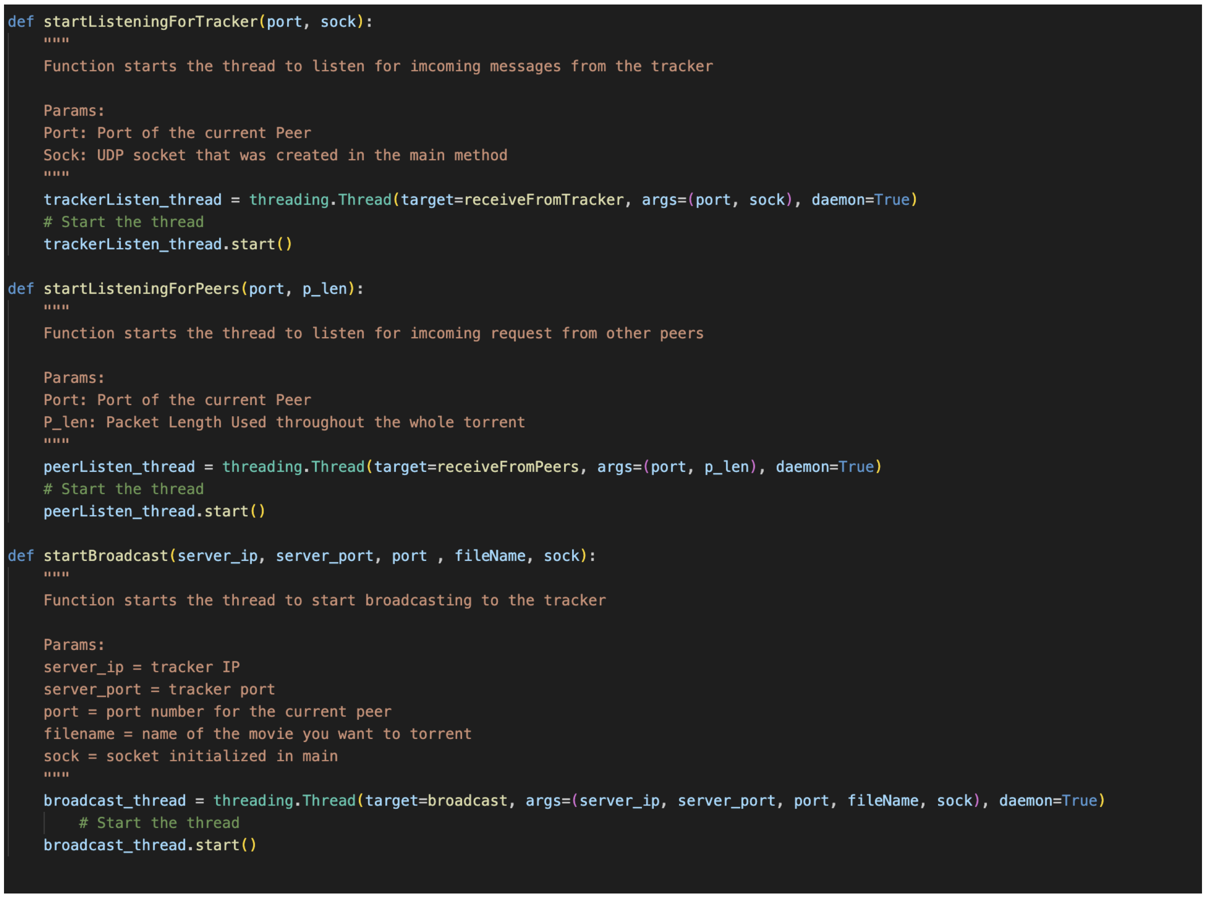
3) **Data Transfer Mechanism:** Peers should be able to request and download data from other peers.



```
⊗ → P2P_Fileshare git:(main) /usr/local/bin/python3 "/Users/aayansayed/Documents/CSCI - 351/Sayed_saunders_Pr
oject/P2P_Fileshare/tracker.py" spiderman.txt
Generating spiderman.torrent from spiderman.txt...
Cleaning seeders...
Seeders: []
Leechers: []
-------------------------------
Seeders: [10003]
Leechers: [(10003, 'spiderman.txt', [''], ('127.0.0.1', 10002))]
Peers Downloading: [(10003, 'spiderman.txt', [''], ('127.0.0.1', 10002))]
10003 seaching for spiderman.txt...
-------------------------------
Seeders: [10003, 10004]
Leechers: []
-------------------------------
Seeders: [10003, 10004]
Leechers: []
-------------------------------
Seeders: [10003, 10004]
Leechers: [(10003, 'spiderman.txt', [''], ('127.0.0.1', 10002))]
Peers Downloading: [(10003, 'spiderman.txt', [''], ('127.0.0.1', 10002))]
10003 seaching for spiderman.txt...
Matching 10003 with 10004...
-------------------------------
```

Our P2P share maintains a list of all peers that are sending, or leeching from other peers. It also displays all the peers that are currently downloading via a TCP connection. In this case the Peer with nothing matches and begins downloading file chunks.

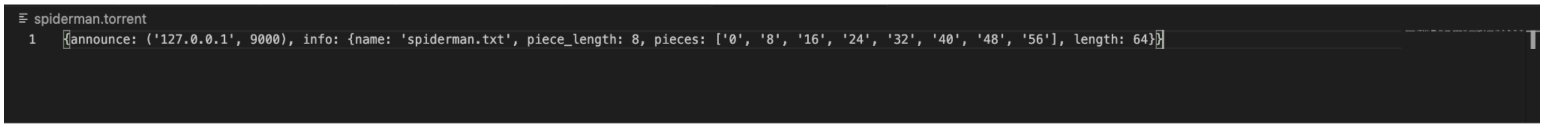
4) **Concurrency and Threading:** Implement multi-threading to handle multiple connections simultaneously.



```python
def startListeningForTracker(port, sock):
    """
    Function starts the thread to listen for imcoming messages from the tracker

    Params:
    Port: Port of the current Peer
    Sock: UDP socket that was created in the main method
    """
    trackerListen_thread = threading.Thread(target=receiveFromTracker, args=(port, sock), daemon=True)
    # Start the thread
    trackerListen_thread.start()

def startListeningForPeers(port, p_len):
    """
    Function starts the thread to listen for imcoming request from other peers

    Params:
    Port: Port of the current Peer
    P_len: Packet Length Used throughout the whole torrent
    """
    peerListen_thread = threading.Thread(target=receiveFromPeers, args=(port, p_len), daemon=True)
    # Start the thread
    peerListen_thread.start()

def startBroadcast(server_ip, server_port, port , fileName, sock):
    """
    Function starts the thread to start broadcasting to the tracker

    Params:
    server_ip = tracker IP
    server_port = tracker port
    port = port number for the current peer
    filename = name of the movie you want to torrent
    sock = socket initialized in main
    """
    broadcast_thread = threading.Thread(target=broadcast, args=(server_ip, server_port, port, fileName, sock), daemon=True)
        # Start the thread
    broadcast_thread.start()
```

Our project fully supports concurrency and each peer is running 3 different threads to allow multi-threads and multiple TCP connections on different threads. The tracker also runs multiple threads for discovery, matching, and cleanup.

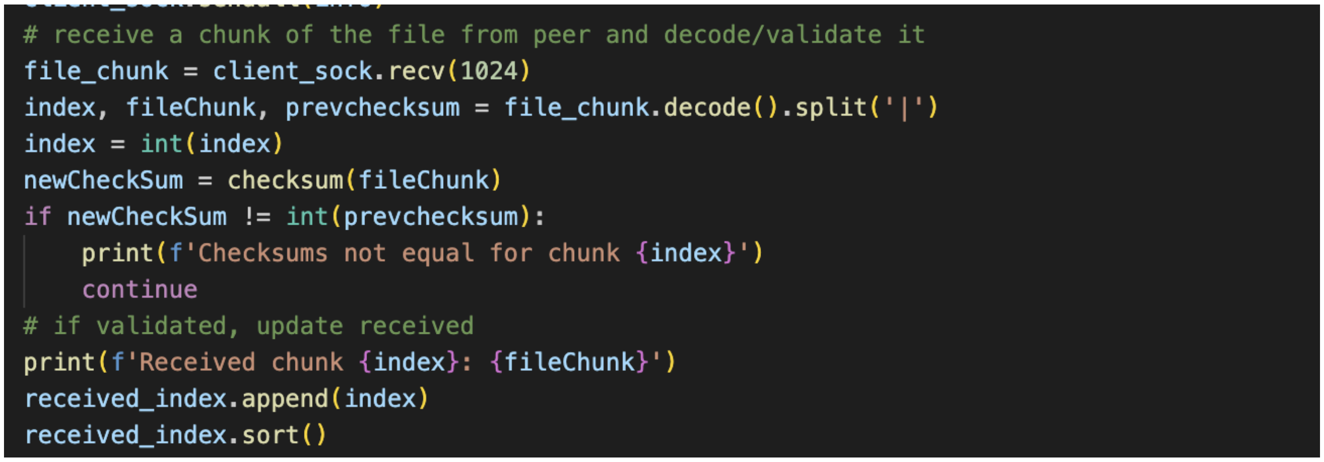
5) **Chunked File Transfer:** Implement file transfer in chunks.



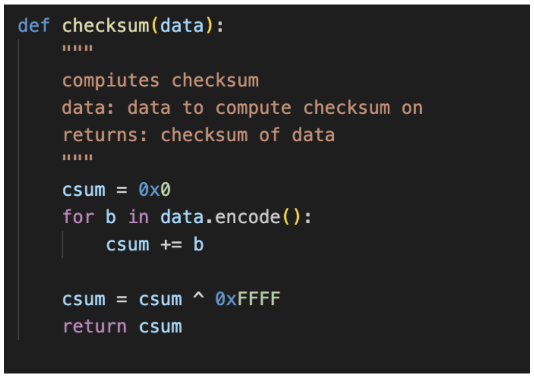
```
spiderman.torrent
{announce: ('127.0.0.1', 9000), info: {name: 'spiderman.txt', piece_length: 8, pieces: ['0', '8', '16', '24', '32', '40', '48', '56'], length: 64}}
```

Our P2P file share used file chunking and indexing to send packets via TCP connections. Our torrent file shows how the indexes and packets are set up. Announce is the port, IP of the tracker, and Info is general info about the specific spiderman torrent if you wanted to download it.

6) **Verifying File Integrity:** Implement a verification mechanism that checks if the downloaded file was corrupted or not.



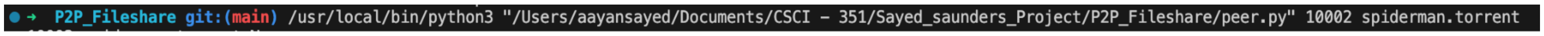
```python
# receive a chunk of the file from peer and decode/validate it
file_chunk = client_sock.recv(1024)
index, fileChunk, prevchecksum = file_chunk.decode().split('|')
index = int(index)
newCheckSum = checksum(fileChunk)
if newCheckSum != int(prevchecksum):
    print(f'Checksums not equal for chunk {index}')
    continue
# if validated, update received
print(f'Received chunk {index}: {fileChunk}')
received_index.append(index)
received_index.sort()
```



```python
def checksum(data):
    """
    compiutes checksum
    data: data to compute checksum on
    returns: checksum of data
    """
    csum = 0x0
    for b in data.encode():
        csum += b

    csum = csum ^ 0xFFFF
    return csum
```

Our P2P file share system validated each portion of the packet using a checksum function. Peers would only be able to broadcast file chunks that passed the checksum so they won’t save or broadcast corrupted chunks.

7) **User Interface:** A basic command-line or graphical interface.



```
● → P2P_Fileshare git:(main) /usr/local/bin/python3 "/Users/aayansayed/Documents/CSCI - 351/Sayed_saunders_Project/P2P_Fileshare/peer.py" 10002 spiderman.torrent
```

Users are able to run and specify which files they want to torrent via the commands line. Here they specify which port they would like to run their peer on and then the movie they want to select (spiderman).