دانشگاه صنعتی شریف
دانشکده مهندسی کامپیوتر
پروژه پایانی درس شبکه‌های کامپیوتری

عنوان:

پیاده‌سازی شبکه تورنت

نگارش

روژین تقی‌زادگان
۴۰۱۱۰۵۷۷۵

استاد

دکتر امیرمهدی صادق‌زاده مسگر

بهمن ۱۴۰۳

توضیح کد Peer:

کد زیر یک peer را در شبکه torrent پیاده‌سازی می‌کند که با یک tracker از طریق UDP برای پیام‌های کنترلی و از طریق TCP برای انتقال فایل‌ها ارتباط برقرار می‌کند. این برنامه این امکان را به‌peerها می‌دهد تا به شبکه join شوند، فایل‌ها را به اشتراک بگذارند، فایل‌ها را از دیگر peers دریافت کنند و به درخواست‌ها پاسخ دهند. همچنین فعالیت شبکه را ثبت می‌کند.

الف) Initialize کردن سوکت‌ها:

```
def __init__(self, tracker_ip, tracker_port, peer_id):
    self.tracker_ip = tracker_ip
    self.tracker_port = tracker_port
    self.peer_id = peer_id
    self.files = {}  # {file_name: file_path}
    self.sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

    self.tcp_sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    self.tcp_sock.bind(('0.0.0.0', 0))  # Bind to any available port
    self.tcp_sock.listen(5)
    self.tcp_port = self.tcp_sock.getsockname()[1]  # Get assigned port

    self.log_file = f'.peer_{self.peer_id}.log'
    logging.basicConfig(filename=self.log_file, level=logging.INFO, format='%(asctime)s - %(message)s')

    threading.Thread(target=self.listen_for_file_requests, daemon=True).start()
    self.join_network()
```

در این تابع ابتدا آدرس ip و پورت tracker برای برقراری ارتباط با آن ذخیره می‌شود. همچنین به هر peer یک peer_id یکتا داده می‌شود. همچنین یک دیکشنری خالی از فایل‌ها شامل نام فایل و آدرس آن ذخیره می‌شود.

در ادامه دو نوع سوکت ایجاد می‌شود:

1) سوکت UDP (self.sock): با tracker برای پیام‌های کنترلی (مثلاً join شدن به شبکه یا اشتراک‌گذاری فایل‌ها) ارتباط برقرار می‌کند.

2) سوکت TCP (self.tcp_sock): منتظر درخواست‌های ارسال فایل از سمت peerهای دیگر است و ارتباطات TCP برای انتقال فایل‌ها را برقرار می‌کند.

سوکت tcp_sock به یک پورت موقت متصل می‌شود و self.tcp_port برای اطلاع‌رسانی پورت انتقال فایل‌ها به tracker استفاده می‌شود.

ب) Join شدن به شبکه:

```
def join_network(self):
    request = {
        'action': 'join',
        'peer_id': self.peer_id,
        'files': list(self.files.keys()),
        'tcp_port': self.tcp_port  # Send TCP port
    }
    self.sock.sendto(json.dumps(request).encode(), (self.tracker_ip, self.tracker_port))
    data, _ = self.sock.recvfrom(1024)
    response = json.loads(data.decode())
    if response.get('status') == 'ok':
        logging.info(f"Peer {self.peer_id} joined the network on TCP port {self.tcp_port}")
        print(f"Peer {self.peer_id} joined the network on TCP port {self.tcp_port}")
```

برای ملحق شدن به شبکه، یک درخواست UDP همراه با جزئیات peer به tracker ارسال می‌شود و peer منتظر تأیید می‌ماند. پس از ارسال درخواست ملحق شدن، peer منتظر پاسخ از tracker می‌ماند. اگر وضعیت ok باشد، peer با موفقیت به شبکه ملحق می‌شود.

ج) به اشتراک‌گذاری فایل‌ها

```
1 usage
def share_file(self, file_name, file_path):
    if os.path.exists(file_path):
        if file_name not in self.files:
            self.files[file_name] = file_path

        request = {
            'action': 'share_file',
            'peer_id': self.peer_id,
            'file': file_name
        }
        self.sock.sendto(json.dumps(request).encode(), (self.tracker_ip, self.tracker_port))
        logging.info(f"Peer {self.peer_id} updated shared files: {self.files}")
        print(f"Peer {self.peer_id} shared a new files: {file_name}")
    else:
        logging.error(f"File {file_path} does not exist")
        print(f"File {file_path} does not exist")
```

زمانی که یک peer فایلی را به اشتراک می‌گذارد، تابع share_file() وجود فایل را بررسی کرده و tracker را با فایل جدید به اشتراک گذاشته شده به‌روز می‌کند. peer فایل را به دیکشنری محلی خود اضافه کرده و پیامی به tracker ارسال می‌کند تا فهرست فایل‌های به اشتراک گذاشته شده را به‌روزرسانی کند.

د) دانلود کردن فایل‌ها

```
1 usage
def get_file(self, file_name):
    request = {
        'action': 'get_peers',
        'peer_id': self.peer_id,
        'file_name': file_name
    }
    self.sock.sendto(json.dumps(request).encode(), (self.tracker_ip, self.tracker_port))
    data, _ = self.sock.recvfrom(1024)
    response = json.loads(data.decode())
```

تابع get_file() به یک peer این امکان را می‌دهد که فایلی را از tracker درخواست کند، که لیستی از peers به اشتراک‌گذارنده فایل را باز می‌گرداند. کاربر یک peer را انتخاب کرده و ارتباط TCP برای انتقال فایل برقرار می‌شود.

```
peers = response.get('peers', [])
if not peers:
    print(f"No peers found with file {file_name}")
    return

print("Available Peers: ")
for i in range(len(peers)):
    print(f"{i+1}: ip: {peers[i][0]}, port: {peers[i][1]}")

peer_index = int(input("Desired Peer: ")) - 1

peer_ip, peer_port = peers[peer_index]  # Connect to the first available peer
print(f"Connecting to peer at {peer_ip}:{peer_port}")
```

زمانی که tracker پاسخ می‌دهد، peer یک target peer را انتخاب کرده، ارتباط TCP برقرار می‌کند و فایل را دانلود می‌کند.

```
try:
    tcp_client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    tcp_client.connect((peer_ip, peer_port))
    tcp_client.send(file_name.encode())

    with open(file_name, 'wb') as f:
        while True:
            data = tcp_client.recv(1024)
            if not data:
                break
            f.write(data)
    tcp_client.close()

    new_request = {
        'action': 'got_the_file',
        'peer_id': self.peer_id,
        'file_name': file_name
    }

    self.sock.sendto(json.dumps(new_request).encode(), (self.tracker_ip, self.tracker_port))
    data, _ = self.sock.recvfrom(1024)

    logging.info(f"Downloaded file {file_name} from {peer_ip}:{peer_port}")
    print(f"Downloaded file {file_name} successfully.")
except Exception as e:
    logging.info(f"Error downloading file {file_name} from {peer_ip}:{peer_port}")
    print(f"Error downloading file {file_name}: {e}")
```

فایل به صورت بخش‌بخش دریافت شده و به سیستم فایل local نوشته می‌شود.
همچنین پس از دریافت فایل، پیامی به tracker ارسال می‌شود تا دستگاه را به لیست دستگاه‌هایی که فایل مربوطه را در اختیار دارند اضافه کند.

ه) رسیدگی به درخواست‌های فایل‌ها

تابع listen_for_file_requests() در یک thread جداگانه اجرا می‌شود و منتظر اتصالات ورودی TCP می‌ماند. اگر یک peer فایلی درخواست کند، peerبررسی می‌کند که آیا فایل به‌طور محلی موجود است و آن را به صورت بخش‌بخش ارسال می‌کند
این اطمینان حاصل می‌کند که peer بتواند درخواست‌های فایل را به صورت هم‌زمان پاسخ دهد و همزمان به عملیات دیگر ادامه دهد.

```
1 usage
def listen_for_file_requests(self):
    while True:
        conn, addr = self.tcp_sock.accept()
        file_name = conn.recv(1024).decode()
        print(f"Sending file {file_name} to {addr}")

        if file_name in self.files:
            with open(self.files[file_name], 'rb') as f:
                while chunk := f.read(1024):
                    conn.send(chunk)
        conn.close()
```

و) مدیریت لاگ‌ها

تابع request_logs() فایل لاگ peer را می‌خواند و ورودی‌های لاگ مربوط به دانلودها را فیلتر کرده و به کاربر این امکان را می‌دهد تا فعالیت‌های خود را مرور کند.این تابع به پیگیری تاریخچه انتقال فایل کمک می‌کند.

```
1 usage
def request_logs(self):
    try:
        with open(self.log_file, 'r') as log_file:
            for line in log_file:
                if 'download' in line.lower():
                    print(line.strip())
    except Exception as e:
        print(f"Failed to read logs: {e}")
```

ز) خروج از شبکه

برای ترک کردن شبکه، متد leave_network() یک درخواست UDP به tracker ارسال می‌کند و به آن اطلاع می‌دهد که peer در حال قطع اتصال است.

```
1 usage
def leave_network(self):
    request = {
        "action": "leave",
        "peer_id": self.peer_id
    }
    self.sock.sendto(json.dumps(request).encode(), (self.tracker_ip, self.tracker_port))
    print(f"Peer {self.peer_id} is leaving the network.")
```

توضیح کد Tracker:

الف) تعریف tracker:

```
1 usage
class Tracker:
    def __init__(self, ip, port):
        self.ip = ip
        self.port = port
        self.peers = {}  # {peer_id: {'files': [file1, file2], 'address': (ip, tcp_port)}}
        self.file_to_peers = {}  # {file_name: [peer1, peer2]}
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        self.sock.bind((self.ip, self.port))

        self.log_file = '.tracker.log'
        logging.basicConfig(filename=self.log_file, level=logging.INFO, format='%(asctime)s - %(message)s')

        logging.info(f"Tracker started at {self.ip}:{self.port}")
        print(f"Tracker started at {self.ip}:{self.port}")
```

در این تابع که tracker initialization است در ابتدا آدرس ip و شماره پورت tracker ذخیره می‌شود.
در ادامه یک dictionary از peerها برای هر tracker ساخته می‌شود که شامل اطلاعات هر peer است که به tracker متصل می‌شود. کلید هر داده در این dictionary شناسه هر peer است که یکتاست و value هر داده، یک دیکشنری دیگر است که شامل لیستی از فایل‌هایی‌ست peer دارد و همچنین شامل آدرس ip و شماره پورت peer است. در واقع به طور خلاصه برای هر peer، یک لیست از فایل‌هایی که آن peer دارد و آدرس آن peer ذخیره می‌شود.
در ادامه یک دیکشنری ساخته می‌شود که نشان می‌دهد هر فایل در اختیار چه peerهایی می‌باشد. در واقع به ازای هر file یک لیست از peerها که آن فایل را در اختیار دارند ذخیره شده است.

در ادامه یک سوکت UDP برای اتصال به tracker ساخته می‌شود و tracker شروع به گوش کردن روی این سوکت می‌کند تا درخواست‌ها از سمت peerها به tracker برسند.

همچنین به ازای هر tracker یک log file تشکیل می‌شود که تمام لاگ‌های برنامه در آن فایل ذخیره می‌شود.

ب) شروع اجرای tracker:

```
if __name__ == "__main__":
    tracker = Tracker( ip: '127.0.0.1',  port: 6771)
    tracker.start()
```

در ابتدا تابع start صدا زده می‌شود:

```
1 usage
def start(self):
    print(f"Tracker running at {self.ip}:{self.port}")
    threading.Thread(target=self.listen_for_peers).start()

    while True:
        command = input()
        if command.startswith('logs request'):
            self.logs_request()

        elif command.startswith('all-logs'):
            self.all_logs()

        elif command.startswith('file_logs'):
            _, file_name = command.split()
            print(file_name)
            self.file_logs(file_name)

        else:
            print("Please enter a valid command!")
```

در تابع start دو اتفاق صورت می‌گیرد:

- یک thread از برنامه tracker شروع به گوش کردن روی سوکت UDP می‌کند تا به درخواست peerها رسیدگی کند.
- یک thread از برنامه tracker به ورودی‌های کاربر در ترمینال رسیدگی می‌کند. این ورودی‌ها مربوط به دستورات log در سیستم هستند.

ج) تابع handle_peer_request:

```
1 usage
def handle_peer_request(self, data, addr):
    try:
        request = json.loads(data.decode())
        action = request.get('action')

        if action == 'join':...

        elif action == 'share_file':...

        elif action == 'get_peers':...

        elif action == "leave":...

    except Exception as e:
        logging.error(f"Error handling request from {addr}: {e}")
        print(f"Error handling request from {addr}: {e}")
```

در این تابع به درخواست‌های join، share، get و exit از سمت peerها رسیدگی می‌شود. حال هر درخواست را به صورت مجزا بررسی می‌کنیم:

۱) درخواست join:

```
if action == 'join':
    peer_id = request.get('peer_id')
    files = request.get('files', [])
    tcp_port = request.get('tcp_port')  # Get TCP port

    self.peers[peer_id] = {'files': files, 'address': (addr[0], tcp_port)}  # Store IP & TCP Port
    for file in files:
        if file not in self.file_to_peers:
            self.file_to_peers[file] = []
        self.file_to_peers[file].append(peer_id)

    logging.info(f"Peer {peer_id} joined on TCP port {tcp_port}")
    print(f"Peer {peer_id} joined on TCP port {tcp_port}")

    self.sock.sendto(json.dumps({'status': 'ok'}).encode(), addr)
```

این بخش مربوط به درخواست پیوستن به شبکه از سوی یک peer است. Peer با استفاده از یک پیام json درخواست خود را به tracker ارسال می‌کند و tracker با گرفتن peer_id، فایل‌هایی که آن peer با خود دارد (که معمولا در ابتدا فایلی را با خود نمی‌آورد و پس از پیوستن به شبکه شروع به افزودن فایل‌ها می‌کند)، و پورت TCP (برای اتصال سایر peerها به این peer در حالت get)، peer را به شبکه اضافه می‌کند.
این بخش در ابتدا دیکشنری peers را آپدیت می‌کند، یعنی فایل‌های مربوط به آن peer و آدرس سوکت TCP آن را ذخیره می‌کند.
در ادامه دیکشنری فایل‌ها را نیز آپدیت می‌کند و اطلاعات فایل‌های اضافه شده توسط peer جدید را اضافه می‌کند.
در ادامه لاگی مربوط به اضافه شدن peer در log file ایجاد می‌کند و یک پیام ok به peer اضافه شده ارسال می‌کند.

۲) درخواست share:

```
elif action == 'share_file':
    peer_id = request.get('peer_id')
    new_file = request.get('file')

    if peer_id in self.peers:
        # Update the peer's file list
        self.peers[peer_id]['files'].append(new_file)

        # Register new file associations
        if new_file not in self.file_to_peers:
            self.file_to_peers[new_file] = []
        self.file_to_peers[new_file].append(peer_id)

        logging.info(f"Peer {peer_id} shared file: {new_file}, status: success")
        print(f"Peer {peer_id} updated files: {new_file}")
        self.sock.sendto(json.dumps({'status': 'ok'}).encode(), addr)
    else:
        logging.info(f"Peer {peer_id} shared file: {new_file}, status: fail")
        self.sock.sendto(json.dumps({'status': 'error', 'message': 'Peer not found'}).encode(), addr)
```

در این بخش، پس از آن که یک peer فایلی را در شبکه به اشتراک گذاشت، اطلاعات آن فایل در tracker ذخیره می‌شود. یعنی فایل جدید به لیست فایل‌های peer اضافه می‌شود، همچنین peer جدید به لیست peerهایی که به آن فایل دسترسی دارند، اضافه می‌شود. در صورت موفقیت، یک لاگ برای موفقیت در به اشتراک‌گذاری فایل توسط peer ذخیره می‌شود و در غیر این صورت یک لاگ مبنی بر عدم موفقیت به اشتراک‌گذاری فایل در log file ذخیره می‌شود.

۳) درخواست get:

```
elif action == 'get_peers':
    peer_id = request.get('peer_id')
    file_name = request.get('file_name')
    if file_name in self.file_to_peers:
        peers_with_file = self.file_to_peers[file_name]
        peer_addresses = [self.peers[peer_id]['address'] for peer_id in peers_with_file]
        self.sock.sendto(json.dumps({'peers': peer_addresses}).encode(), addr)
        logging.info(f"Peer {peer_id} wants peers' names with file: {file_name}, status: success")
    else:
        logging.info(f"Peer {peer_id} wants peers' names with file: {file_name}, status: fail")
        self.sock.sendto(json.dumps({'peers': []}).encode(), addr)
```

در صورتی که یک peer درخواست گرفتن فایلی را به tracker بدهد، tracker اطلاعات peerهایی که به آن فایل دسترسی دارند، به همراه آدرس سوکت TCP آن‌ها (برای برقراری ارتباط مستقیم بین دو peer) را برای آن peer ارسال می‌کند. همچنین لاگی در log file مبنی بر موفقیت یا عدم موفقیت این عدم نوشته می‌شود.

۴) درخواست آپدیت اطلاعات پس از دریافت فایل:

```
elif action == "got_the_file":
    peer_id = request.get('peer_id')
    file_name = request.get('file_name')
    if peer_id in self.peers:
        # Update the peer's file list
        self.peers[peer_id]['files'].append(file_name)

        # Register new file associations
        if file_name not in self.file_to_peers:
            self.file_to_peers[file_name] = []
        self.file_to_peers[file_name].append(peer_id)

        logging.info(f"Peer {peer_id} downloaded the file file: {file_name}, status: success")
        print(f"Peer {peer_id} downloaded the file file: {file_name}")
        self.sock.sendto(json.dumps({'status': 'ok'}).encode(), addr)
    else:
        logging.info(f"Peer {peer_id} shared file: {new_file}, status: fail")
        self.sock.sendto(json.dumps({'status': 'error', 'message': 'Peer not found'}).encode(), addr)
```

پس از اینکه یک peer به طور مستقیم فایلی را از یک peer دیگر دریافت کرد، به tracker درخواست می‌دهد که اطلاعاتش را آپدیت کند. یعنی فایل جدید را به لیست فایل‌های آن peer اضافه کند و نام peer را به لیست peerهایی که به آن فایل دسترسی دارند، اضافه کند.

۴) درخواست leave:

```
elif action == "leave":
    peer_id = request.get('peer_id')
    if peer_id in self.peers:
        for file_name in self.peers[peer_id]['files']:
            self.file_to_peers[file_name].remove(peer_id)
            if not self.file_to_peers[file_name]:  # Remove empty file records
                del self.file_to_peers[file_name]
        del self.peers[peer_id]
        logging.info(f"Peer {peer_id} left the network.")
        print(f"Peer {peer_id} left the network.")
    self.sock.sendto(json.dumps({'status': 'ok'}).encode(), addr)
```

در صورتی که یک peer درخواست خروج از شبکه را بدهد، به ازای تمام فایل‌هایی که آن peer در اختیار داشته است، peer_id آن از لیست peerهایی که آن فایل‌ها را دارند، حذف می‌شود. در ادامه اطلاعات مربوط به آن از دیکشنری peers حذف می‌شود و لاگی مبنی بر خروج آن peer از شبکه در log file ذخیره می‌شود.

د) درخواست‌های log:

دستورهای مربوط به log، از log file مربوط به tracker می‌خوانند و با توجه به خواسته سیستم، لاگ‌های مورد نظر را خروجی می‌دهند. مثلا در دستور file_logs، تمام لاگ‌هایی که شامل اسم فایل مورد نظر است خروجی داده می‌شود. یا در دستور all-logs تمام دستوراتی که مربوط به اشتراک‌گذاری فایل جدید است، ذخیره می‌شود.

```
1 usage
def logs_request(self):
    try:
        with open(self.log_file, 'r') as log_file:
            print("Logs from tracker.log:")
            for line in log_file:
                print(line.strip())
    except Exception as e:
        print(f"Failed to read logs: {e}")

1 usage
def all_logs(self):
    try:
        with open(self.log_file, 'r') as log_file:
            for line in log_file:
                if 'shared' in line:
                    print(line.strip())
    except Exception as e:
        print(f"Failed to read logs: {e}")

1 usage
def file_logs(self, file_name):
    if file_name not in self.file_to_peers:
        print("File name does not exist.")
    else:
        try:
            with open(self.log_file, 'r') as log_file:
                for line in log_file:
                    if file_name in line:
                        print(line.strip())
        except Exception as e:
            print(f"Failed to read logs: {e}")
```

تست کد:

ترتیب اجرای دستورات در log مربوط به tracker مشخص است.

~/desktop/torrent — Python tracker3.py | ~/desktop/torrent — -zsh

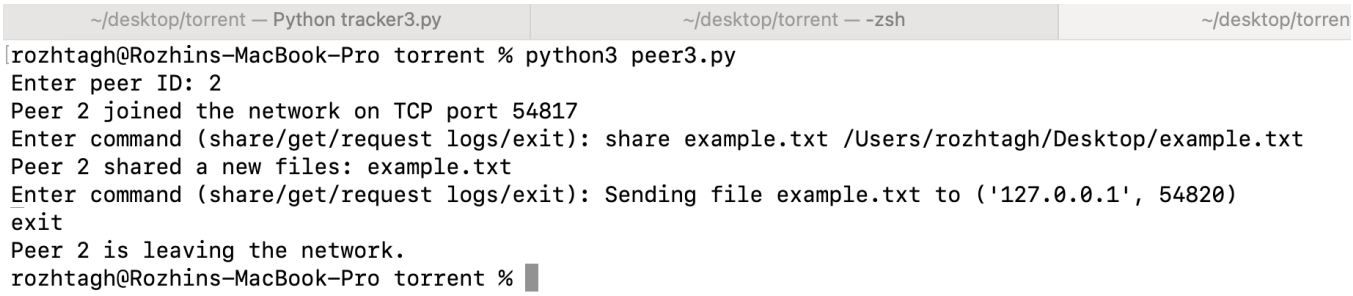
```
rozhtagh@Rozhins-MacBook-Pro torrent % python3 tracker3.py
Tracker started at 127.0.0.1:6771
Tracker running at 127.0.0.1:6771
Peer 1 joined on TCP port 54816
Peer 2 joined on TCP port 54817
Peer 1 updated files: example.txt
Peer 1 updated files: meow.txt
Peer 2 updated files: example.txt
Peer 3 joined on TCP port 54819
Peer 3 downloaded the file file: example.txt
Peer 1 left the network.
Peer 2 left the network.
Peer 3 left the network.
logs request
Logs from tracker.log:
2025-02-06 23:14:21,598 - Tracker started at 127.0.0.1:6771
2025-02-06 23:14:24,950 - Peer 1 joined on TCP port 54816
2025-02-06 23:14:34,222 - Peer 2 joined on TCP port 54817
2025-02-06 23:14:43,857 - Peer 1 shared file: example.txt, status: success
2025-02-06 23:14:53,540 - Peer 1 shared file: meow.txt, status: success
2025-02-06 23:15:03,023 - Peer 2 shared file: example.txt, status: success
2025-02-06 23:15:20,512 - Peer 3 joined on TCP port 54819
2025-02-06 23:15:32,550 - Peer 3 wants peers' names with file: example.txt, status: success
2025-02-06 23:15:38,614 - Peer 3 downloaded the file file: example.txt, status: success
2025-02-06 23:16:03,539 - Peer 1 left the network.
2025-02-06 23:16:07,331 - Peer 2 left the network.
2025-02-06 23:16:10,917 - Peer 3 left the network.
all-logs
2025-02-06 23:14:43,857 - Peer 1 shared file: example.txt, status: success
2025-02-06 23:14:53,540 - Peer 1 shared file: meow.txt, status: success
2025-02-06 23:15:03,023 - Peer 2 shared file: example.txt, status: success
file_logs example.txt
2025-02-06 23:14:43,857 - Peer 1 shared file: example.txt, status: success
2025-02-06 23:15:03,023 - Peer 2 shared file: example.txt, status: success
2025-02-06 23:15:32,550 - Peer 3 wants peers' names with file: example.txt, status: success
2025-02-06 23:15:38,614 - Peer 3 downloaded the file file: example.txt, status: success
file_logs meow.txt
2025-02-06 23:14:53,540 - Peer 1 shared file: meow.txt, status: success
```

Tracker

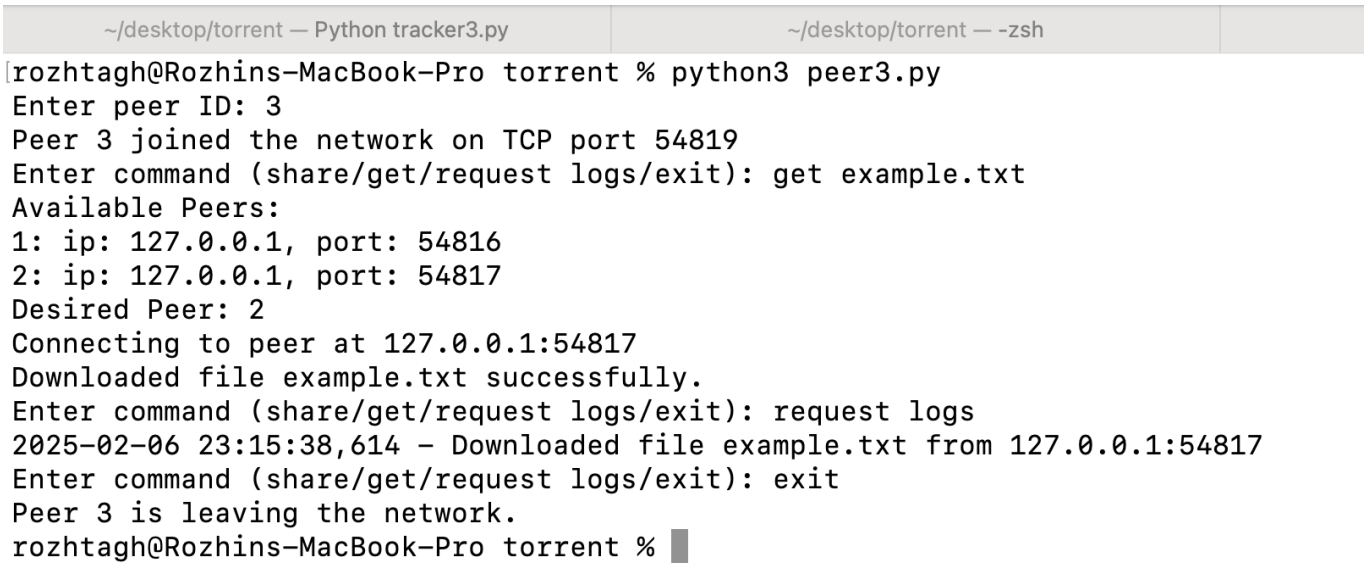
~/desktop/torrent — Python tracker3.py | × | ~/desktop/torrent — -zsh | ~/desktop/torrent -

```
rozhtagh@Rozhins-MacBook-Pro torrent % python3 peer3.py
Enter peer ID: 1
Peer 1 joined the network on TCP port 54816
Enter command (share/get/request logs/exit): share example.txt /Users/rozhtagh/Desktop/example.txt
Peer 1 shared a new files: example.txt
Enter command (share/get/request logs/exit): share meow.txt /Users/rozhtagh/Desktop/meow.txt
Peer 1 shared a new files: meow.txt
Enter command (share/get/request logs/exit): exit
Peer 1 is leaving the network.
rozhtagh@Rozhins-MacBook-Pro torrent %
```

Peer 1


~/desktop/torrent — Python tracker3.py
~/desktop/torrent — -zsh
~/desktop/torren
rozhtagh@Rozhins-MacBook-Pro torrent % python3 peer3.py
Enter peer ID: 2
Peer 2 joined the network on TCP port 54817
Enter command (share/get/request logs/exit): share example.txt /Users/rozhtagh/Desktop/example.txt
Peer 2 shared a new files: example.txt
Enter command (share/get/request logs/exit): Sending file example.txt to ('127.0.0.1', 54820)
exit
Peer 2 is leaving the network.
rozhtagh@Rozhins-MacBook-Pro torrent %

Peer 2


~/desktop/torrent — Python tracker3.py
~/desktop/torrent — -zsh
rozhtagh@Rozhins-MacBook-Pro torrent % python3 peer3.py
Enter peer ID: 3
Peer 3 joined the network on TCP port 54819
Enter command (share/get/request logs/exit): get example.txt
Available Peers:
1: ip: 127.0.0.1, port: 54816
2: ip: 127.0.0.1, port: 54817
Desired Peer: 2
Connecting to peer at 127.0.0.1:54817
Downloaded file example.txt successfully.
Enter command (share/get/request logs/exit): request logs
2025-02-06 23:15:38,614 - Downloaded file example.txt from 127.0.0.1:54817
Enter command (share/get/request logs/exit): exit
Peer 3 is leaving the network.
rozhtagh@Rozhins-MacBook-Pro torrent %

Peer 3