

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH



MẠNG MÁY TÍNH (CO3094)

BÀI TẬP LỚN 1

P2P File Transferring Application

Giảng viên hướng dẫn:	TS. Nguyễn Lê Duy Lai	
Sinh viên thực hiện:	Nguyễn Tuấn Huy	2211253
	Võ Hoàng Huy	2211298
	Nguyễn Văn Nhật Huy	2211254

TP. HỒ CHÍ MINH, THÁNG 11/2024



Mục lục

1	Bảng phân công công việc	2
2	Các chức năng của ứng dụng	3
2.1	Đăng ký	3
2.2	Đăng nhập	3
2.3	Đăng ký file với tracker	3
2.4	Tải file từ các peer khác	4
3	Các protocol của ứng dụng	4
3.1	Cấu trúc thông điệp, cách đóng gói và giải mã thông điệp	4
3.2	Tracker vs Peer protocol	4
3.2.1	Đăng kí	4
3.2.2	Đăng nhập	5
3.2.3	Đăng kí file	6
3.2.4	Lấy thông tin các file	6
3.2.5	Tải file	7
3.2.6	Đăng xuất hệ thống	8
3.2.7	Cập nhật rarity cho các piece của một file	8
3.3	Peer vs Peer protocol	8
3.3.1	Gửi yêu cầu lấy piece từ peer khác	8
3.3.2	Gửi piece cho các peer khác	9
4	Hiện thực ứng dụng	9
4.1	Các file hiện thực trong ứng dụng:	9
4.2	Tracker	9
4.3	Peer	11
4.4	Torrent	15
4.5	Helper function	16
4.6	Chiến thuật tải file	17
5	Demo ứng dụng và đánh giá	17
5.1	Server	17
5.2	Peer	18
5.3	Thí nghiệm tải file	28



1 Bảng phân công công việc

No.	Họ và tên	Nhiệm vụ
1	Nguyễn Tuấn Huy	Hiện thực GUI
2	Võ Hoàng Huy	Hiện thực hệ thống
3	Nguyễn Văn Nhật Huy	Kiểm thử, đánh giá và viết báo cáo

2 Các chức năng của ứng dụng

2.1 Đăng ký

Chức năng đăng ký cho phép người dùng chưa có tài khoản sẽ tạo tài khoản mới. Ứng dụng hiển thị một form gồm 3 field: username (tên người dùng), password (mật khẩu) và confirm password (xác nhận mật khẩu). Các field này sẽ có những ràng buộc:

- Username: tên người dùng sử dụng không được trùng với các người dùng khác.
- Password: mật khẩu người dùng sử dụng cho tài khoản.
- Confirm password: phải có giá trị như field `password`.

Sau khi nhập đầy đủ các field cần có, người dùng cần nhấn vào nút **Register** để gửi thông tin đến **tracker**. **Tracker** sẽ kiểm tra các thông tin của người dùng đăng ký, sau đó trả về thông báo:

- Khi đăng ký thành công: người dùng sẽ trở về giao diện chính để đăng nhập vào hệ thống với các tài khoản đã đăng ký thành công trước đó.
- Khi đăng ký không thành công: người dùng sẽ nhận được thông báo về lỗi đăng ký (tên đăng nhập đã được dùng, mật khẩu xác nhận không đúng, ...), sau đó người dùng quay về giao diện chính, điền lại thông tin và đăng ký lại với **tracker**.

2.2 Đăng nhập

Chức năng đăng nhập cho phép người dùng đã có tài khoản đăng nhập (đã đăng kí thành công với **tracker** trước đó) và sử dụng tính năng của hệ thống. Ứng dụng hiển thị một form gồm 2 field: username (tên người dùng), password (mật khẩu). Các field này sẽ có những ràng buộc:

- Username: tên người dùng đã đăng ký thành công trước đó với **tracker**.
- Password: mật khẩu người dùng đã đăng ký với `username` tương ứng.

Sau khi nhập đầy đủ các field cần có, người dùng cần nhấn vào nút **Log in** để gửi thông tin đến **tracker**. **Tracker** sẽ kiểm tra các thông tin của người dùng đăng nhập, sau đó trả về thông báo:

- Khi đăng nhập thành công: người dùng sẽ được sử dụng chuyển vào giao diện của ứng dụng với các chức năng như `upload file` và `download file`.
- Khi đăng nhập không thành công: người dùng sẽ nhận được thông báo về lỗi đăng nhập (tên người dùng không tồn tại, sai mật khẩu,...), sau đó người dùng sẽ quay về giao diện chính, kiểm tra lại thông tin để đăng nhập.

2.3 Đăng ký file với tracker

Sau khi đăng nhập thành công vào ứng dụng, người dùng sẽ được thực hiện chức năng đăng ký file (`publish file`) với **tracker**. Có 2 chế độ `publish`:

- **Publish whole folder**: chức năng này cho phép người dùng đăng kí cả folder với các file tương ứng trong folder.

- **Publish selected file:** chức năng này cho phép người dùng đăng kí các file người dùng chọn (có thể chọn nhiều file hoặc 1 file).

Sau khi publish file thành công, người dùng sẽ được nhận thông báo từ **tracker** và các file đã publish sẽ được hiển thị với người dùng.

2.4 Tải file từ các peer khác

Sau khi đăng nhập thành công vào ứng dụng, người dùng sẽ được thực hiện chức năng tải file (**fetch** file) với peer khác. Người dùng có thể **fetch** về nhiều file cùng lúc hoặc chỉ 1 file từ các peer khác. Khi thực hiện chức năng này, peer sẽ nhận được thông tin về các peer online đang giữ file đó và kết nối với các peer đó để tải file về.

Sau khi tải file thành công, người dùng sẽ nhận được thông báo và file tải về thành công sẽ được lưu vào folder **repo_username** của người dùng đó.

3 Các protocol của ứng dụng

3.1 Cấu trúc thông điệp, cách đóng gói và giải mã thông điệp

Giữa **tracker** và **peer**, **peer** và **tracker** đều sử dụng chung một hình thức thông điệp thông qua giao thức TCP/IP. Hình thức thông điệp được tổ chức theo cấu trúc chung như sau:

```
{
  'type': type_of_message,
  'another_info': value
}
```

- **type:** định dạng kiểu thông điệp gửi và nhận.
- **another_info:** dùng chứa các thông tin khác của từng loại thông điệp khác nhau.

Cách đóng gói thông điệp và giải mã thông điệp: các thông điệp được tổ chức dưới dạng **dictionary**, nên để gửi đi thông điệp cần được **serialize** (tuần tự hóa) thành chuỗi byte. Khi nhận được thông điệp đã là một chuỗi byte và cần được giải mã (**deserialize**) trở lại **dictionary** ban đầu.

3.2 Tracker vs Peer protocol

3.2.1 Đăng kí

Khi người dùng đăng kí tài khoản với **tracker**, thông tin sẽ được gửi đi dưới dạng:

```
{
  'type': REGISTER,
  'username': register_username,
  'password': register_password,
  'ip': ip,
  'port': listen_port
}
```

Tracker sẽ tiếp nhận thông tin và gửi đi thông tin trong các trường hợp bên dưới:

- Khi tên người dùng đã tồn tại:

```
{
  'type': REGISTER_FAIL,
  'message': 'Account already exists'
}
```

- Khi đăng ký thành công:

```
{
  'type': REGISTER_SUCCESS,
  'message': 'Registration successful! Please login.'
  'peer_id': peer_id
}
```

3.2.2 Đăng nhập

Khi người dùng đăng nhập vào hệ thống và thông báo đến **tracker**, thông tin sẽ được gửi đi dưới dạng:

```
{
  'type': LOGIN,
  'username': login_username,
  'password': login_password,
  'ip': ip,
  'port': listen_port
}
```

Tracker sẽ nhận yêu cầu đăng nhập và gửi lại thông tin theo các trường hợp bên dưới:

- Khi đăng nhập thành công:

```
{
  'type': LOGIN_SUCCESS,
  'message': 'Login successful',
  'peer_id': peer_id
}
```

- Khi đăng nhập không thành công do mật khẩu không đúng:

```
{
  'type': LOGIN_FAIL,
  'message': 'Incorrect password'
}
```

- Khi đăng nhập không thành công do tên người dùng không tồn tại:

```
{
  'type': LOGIN_FAIL,
  'message': 'Account not found'
}
```

3.2.3 Đăng kí file

Khi người dùng sử dụng chức năng đăng kí file với **tracker**, file đăng kí sẽ được tổ chức thông tin dưới dạng một **torrent** và thông tin sẽ được gửi đi dưới dạng:

```
{
  'type': PUBLISH,
  'peer_id': peer_id,
  'info_hash': torrent.info_hash,
  'filename': torrent.file_name,
  'filesize': torrent.file_size
}
```

Tracker sẽ tiếp nhận thông tin, xử lý các file và gửi trả về thông tin cho người dùng:

```
{
  'type': PUBLISH_SUCCESS,
  'message': 'Publish successful'
}
```

3.2.4 Lấy thông tin các file

Khi người dùng cần biết về thông tin các file đang khả dụng, thông tin sẽ được gửi đi dưới dạng:

```
{
  'type': GET_FILES,
  'peer_id': self.peer_id
}
```

Tracker sẽ tiếp nhận, lấy những file khả dụng và gửi lại thông tin cho người dùng:

```
{
  'type': GET_FILES_SUCCESS,
  'files': available_files
}
```

3.2.5 Tải file

Khi người dùng sử dụng chức năng tải file của ứng dụng, người dùng sẽ chọn các file muốn tải trên giao diện, sau đó tải thì thông tin sẽ được gửi đi dưới dạng:

```
{
  'type': FETCH,
  'peer_id': peer_id,
  'info_hash': available_files[index]['info_hash']
}
```

Tracker sẽ nhận thông tin về file, kiểm tra và gửi đi thông tin trong các trường hợp dưới đây:

- File không tồn tại:

```
{
  'type': FETCH_FAIL,
  'message': 'File not found'
}
```

- Khi file tồn tại và có các peer online giữ file đó:

```
{
  'type': FETCH_SUCCESS,
  'peers': online_peers
}
```

- Khi file tồn tại nhưng không có peer nào online:

```
{
  'type': FETCH_FAIL,
  'message': 'No peers available'
}
```


3.2.6 Đăng xuất hệ thống

Khi không còn nhu cầu sử dụng ứng dụng, người dùng có thể đăng xuất hệ thống:

```
{
  'type': LOGOUT,
  'peer_id': peer_id
}
```

Tracker tiếp nhận thông tin, cập nhật tình trạng online của peer và trả lại kết quả:

```
{
  'type': LOGOUT_SUCCESS,
  'message': 'Logout successful'
}
```

3.2.7 Cập nhật rarity cho các piece của một file

Peer khi tải một file phải liên tục cập nhật bitfield của một file và rarity score của các piece của file đó với tracker

```
{
  'type': UPDATE_PIECE_POINT,
  'peer_id': peer_id,
  'info_hash': info_hash,
  'piece_index': piece_index
}
```

Tracker nhận được thông tin sẽ cập nhật bitfield, rarity score của piece thuộc về file đang tải.

3.3 Peer vs Peer protocol

3.3.1 Gửi yêu cầu lấy piece từ peer khác

Khi nhận được thông tin về các peer đang giữ file mà người dùng muốn tải về, ứng dụng sẽ thiết lập kết nối từ người dùng đến các peer khác rồi gửi đi yêu cầu lấy piece:

```
{
  'type': GET_PIECE,
  'info_hash': info_hash,
  'piece_index': piece_index
}
```

3.3.2 Gửi piece cho các peer khác

Khi peer nhận yêu cầu về piece mà peer đang giữ, peer sẽ kiểm tra `info_hash` của file, sau đó gửi lại thông tin cho các peer kia trong các trường hợp:

- Khi file không tồn tại:

```
{  
  'type': GET_PIECE_FAIL,  
  'message': 'File not found'  
}
```

- Khi file tồn tại:

```
{  
  'type': GET_PIECE_SUCCESS,  
  'data': piece_data  
}
```

4 Hiện thực ứng dụng

4.1 Các file hiện thực trong ứng dụng:

```
app/  
|-- tracker.py      # hiện thực tracker server  
|-- peer.py         # hiện thực peer và tính năng của peer  
|-- helper.py       # hiện thực các hàm hỗ trợ  
|-- parameter.py    # chứa các tham số (type của các thông điệp, piece size,...) và các thư viện  
|-- peer_UI.py      # hiện thực giao diện cho peer  
|-- torrent.py      # hiện thực torrent
```

4.2 Tracker

Mô tả lớp

Lớp **Tracker** chịu trách nhiệm quản lý kết nối từ các peer, xử lý việc chia sẻ file, và lưu trữ thông tin người dùng cũng như metadata của các file trong hệ thống chia sẻ tệp theo kiểu BitTorrent.

Thuộc tính

- `conn`:
 - Kết nối cơ sở dữ liệu SQLite để lưu trữ thông tin về các tệp torrent.
 - Đây là đối tượng dùng để quản lý việc kết nối và truy vấn cơ sở dữ liệu trong suốt thời gian chạy của máy chủ.

- **cursor:**

- Con trỏ được sử dụng để thực thi các truy vấn SQL trên cơ sở dữ liệu.
- Con trỏ này cho phép thực hiện các thao tác như thêm, sửa, và truy xuất dữ liệu trong cơ sở dữ liệu SQLite.

- **lock:**

- Đối tượng khóa Lock dùng để đồng bộ hóa các thao tác trong môi trường đa luồng.
- Giúp tránh tình trạng tranh chấp dữ liệu khi nhiều luồng cùng truy cập vào các tài nguyên chung (chẳng hạn như cơ sở dữ liệu hoặc danh sách peers).

- **tracker_socket:**

- Sử dụng để quản lý kết nối mạng của máy chủ tracker, giúp nó giao tiếp với các peer.
- Đây là đối tượng socket giúp gửi và nhận thông điệp giữa tracker và các peer tham gia chia sẻ tệp.

- **files_info:**

- Đây là một dictionary lưu trữ thông tin về các tệp torrent đang được theo dõi bởi tracker.
- Cấu trúc: `info_hash -> { filename, size, peers }`, trong đó:
 - * `info_hash` là mã băm duy nhất của tệp torrent.
 - * `filename` là tên của tệp torrent.
 - * `size` là kích thước của tệp torrent.
 - * `peers` là một dictionary chứa thông tin về các peer, trong đó mỗi peer có `peer_id`, `num_pieces` (số lượng phần tệp) và `pieces_point` (trạng thái tải các phần tệp).

Mô tả chức năng

- **handle_peer(self, client_socket, addr):** Xử lý giao tiếp với một peer đã kết nối. Nó xử lý các yêu cầu dựa trên loại tin nhắn, chẳng hạn như đăng ký, đăng nhập, công bố, lấy dữ liệu, và nhiều hơn nữa.
- **handshake_service(self, client_socket, info):** Xử lý quá trình bắt tay với một peer, gửi thông báo bắt tay thành công lại cho client.
- **register_service(self, client_socket, info):** Xử lý yêu cầu đăng ký từ một peer. Nó kiểm tra xem tên người dùng đã được sử dụng hay chưa và nếu chưa, sẽ thêm người dùng mới vào cơ sở dữ liệu.
- **login_service(self, client_socket, info):** Xử lý yêu cầu đăng nhập từ một peer. Nó kiểm tra tên người dùng và mật khẩu, và nếu hợp lệ, cập nhật trạng thái đăng nhập của người dùng.
- **logout_service(self, client_socket, info):** Xử lý quá trình đăng xuất cho một peer bằng cách cập nhật trạng thái của họ trong cơ sở dữ liệu thành offline.
- **publish_service(self, client_socket, info):** Xử lý việc công bố tệp từ một peer. Nó thêm tệp mới vào tracker hoặc cập nhật các tệp hiện tại với bitfield mới của peer.

- `initialize_peer_start_download(self, peer_id, info_hash)`: Khởi tạo quá trình tải xuống cho một peer bằng cách thiết lập bitfield và các mảnh tệp cho tệp được xác định bằng info hash.
- `update_peer_pieces(self, peer_id, info_hash, piece_index)`: Cập nhật bitfield và thông tin mảnh cho một peer, đánh dấu các mảnh đã tải xuống.
- `fetch_service(self, client_socket, info)`: Xử lý yêu cầu lấy tệp. Nó tìm các peer có tệp yêu cầu và gửi danh sách đó cho peer yêu cầu.
- `get_files_service(self, client_socket, info)`: Lấy danh sách các tệp có sẵn cho peer yêu cầu và gửi chúng cho họ.
- `init_database(self)`: Khởi tạo cơ sở dữ liệu bằng cách tạo bảng người dùng nếu bảng này chưa tồn tại.
- `getAccountByUsername(self, user)`: Lấy thông tin người dùng từ cơ sở dữ liệu theo tên người dùng.
- `getPeerId(self, user)`: Lấy ID của peer cho tên người dùng đã cho từ cơ sở dữ liệu.
- `getIpandPortByPeerID(self, peer_id)`: Lấy địa chỉ IP và cổng của một peer theo ID của peer từ cơ sở dữ liệu.
- `updateLogin(self, id, ip, port)`: Cập nhật trạng thái người dùng thành online (status = 1) trong cơ sở dữ liệu và lưu trữ thông tin IP và cổng của họ.
- `updateLogout(self, id)`: Cập nhật trạng thái người dùng thành offline (status = 0) trong cơ sở dữ liệu.
- `insertUser(self, user, passwd, ip, port)`: Thêm một người dùng mới vào cơ sở dữ liệu với tên người dùng, mật khẩu, IP và cổng, và đặt trạng thái của họ thành offline (0).
- `is_peer_online(self, peer_id)`: Kiểm tra xem một peer có đang online hay không bằng cách xác minh trạng thái của họ trong cơ sở dữ liệu.

4.3 Peer

Peer

Attributes:

- ip: str
- tracker_port: int
- listen_port: int
- peer_socket: socket
- listen_socket: socket
- peer_id: str
- username: str
- torrents: dict
- available_files: list
- ui: PeerUI
- downloading_pieces: dict
- logging_initialized: bool
- _shutdown: bool

Methods:

- + handle_login()
- + handle_register()
- + handle_logout()
- + connect_to_tracker(SERVER: str, PORT: int)
- + connect_to_peer(ip: str, port: int)
- + get_available_files()
- + publish_file(file_path: str)
- + fetch_file(index: int)
- + manage_download(peers, file_info, num_pieces, pieces_point)
- + download_piece(peer_id, ip, port, info_hash, piece_index, pieces, piece_lock, downloaded_pieces)
- + notify_tracker_for_piece_downloaded(info_hash, piece_index)
- + peer_server()
- + listen_for_connections()
- + handle_client_connection(client_socket, addr)
- + handle_piece_transfer(client_socket, request)
- + cleanup()

Mô tả lớp

Lớp **Peer** đại diện cho một đối tượng Peer trong hệ thống chia sẻ tệp tin qua giao thức BitTorrent. Lớp này quản lý các kết nối, tải và chia sẻ các tệp tin, cũng như tương tác với các Tracker và Peer khác.

Thuộc tính

- **ip**: Kiểu: **str**. Mô tả: Địa chỉ IP của peer hiện tại.
- **tracker_port**: Kiểu: **int**. Mô tả: Cổng dùng để kết nối với tracker.
- **listen_port**: Kiểu: **int** (ban đầu **None**). Mô tả: Cổng dùng để lắng nghe kết nối từ các peer khác.
- **peer_socket**: Kiểu: **socket** (ban đầu **None**). Mô tả: Socket dùng để kết nối với tracker.
- **listen_socket**: Kiểu: **socket** (ban đầu **None**). Mô tả: Socket dùng để nhận kết nối từ các peer khác.
- **peer_id**: Kiểu: **str** (ban đầu **None**). Mô tả: ID duy nhất của peer.
- **username**: Kiểu: **str** (ban đầu **None**). Mô tả: Tên người dùng của peer.
- **torrents**: Kiểu: **dict** (ban đầu rỗng). Mô tả: Từ điển ánh xạ **info_hash** của torrent với thông tin về torrent tương ứng.
- **available_files**: Kiểu: **list**. Mô tả: Danh sách các tệp có sẵn để chia sẻ với các peer khác.
- **ui**: Kiểu: **PeerUI**. Mô tả: Giao diện người dùng của peer, quản lý tương tác UI với người dùng.
- **downloading_pieces**: Kiểu: **dict**. Mô tả: Từ điển ánh xạ **info_hash** với các phần dữ liệu đang được tải xuống.
- **_shutdown**: Kiểu: **bool** (mặc định **False**). Mô tả: Cờ để xác định liệu peer có đang được tắt hay không.
- **logging_initialized**: Kiểu: **bool** (mặc định **False**). Mô tả: Cờ cho biết xem quá trình ghi nhật ký (logging) đã được khởi tạo hay chưa.

Mô tả chức năng

- **handle_login**: Xử lý đăng nhập bằng cách gửi thông tin đăng nhập tới tracker, nhận phản hồi và khởi tạo các tệp và cài đặt nhật ký nếu đăng nhập thành công.
- **handle_register**: Xử lý đăng ký tài khoản người dùng mới, gửi thông tin đăng ký tới tracker và tạo thư mục lưu trữ dữ liệu người dùng nếu đăng ký thành công.
- **handle_logout**: Xử lý đăng xuất bằng cách gửi yêu cầu đăng xuất tới tracker và kết thúc phiên hoạt động của peer.
- **connect_to_tracker**: Kết nối với máy chủ tracker bằng cách thiết lập một kết nối TCP và thử lại nếu gặp lỗi.
- **connect_to_peer**: Kết nối tới một peer khác bằng cách sử dụng địa chỉ IP và cổng của peer đó.
- **get_available_files**: Yêu cầu danh sách các tệp có sẵn từ tracker và cập nhật danh sách tệp có sẵn cho người dùng.

- **publish_file**: Đăng tải một tệp lên tracker, gửi thông tin torrent và cập nhật danh sách tệp sau khi đăng tải thành công.
- **fetch_file**: Tải xuống một tệp từ một peer khác thông qua tracker, tính toán số lượng phần tệp và quản lý quá trình tải xuống bằng đa luồng.
- **manage_download**: Quản lý quá trình tải xuống tệp tin bằng cách sử dụng nhiều luồng (threads). Phân bổ các phần (pieces) cho các peers và khởi tạo quá trình tải xuống. Nếu có phần thiếu, phương thức sẽ thử tải lại.
- **download_piece**: Tải xuống một phần của tệp từ một peer cụ thể. Phương thức thử lại nếu có lỗi xảy ra trong quá trình tải xuống.
- **notify_tracker_for_piece_downloaded**: Thông báo cho tracker rằng một phần của tệp đã được tải xuống thành công.
- **save_complete_file**: Lắp ráp và lưu tệp hoàn chỉnh sau khi tất cả các phần đã được tải xuống. Nếu thiếu phần, phương thức sẽ báo lỗi.
- **peer_server**: Khởi tạo máy chủ lắng nghe kết nối từ peers. Nếu cổng hiện tại đang sử dụng, phương thức sẽ thử lại với cổng khác.
- **listen_for_connections**: Lắng nghe và chấp nhận kết nối từ các client. Mỗi kết nối sẽ được xử lý trong một luồng riêng biệt.
- **handle_client_connection**: Xử lý kết nối của từng client, nhận và phản hồi các yêu cầu của client như tải xuống các phần tệp.
- **handle_piece_transfer**: Xử lý việc truyền tải một phần tệp đến client.
- **download_from_peer**: Tải tệp từ một peer cụ thể (phương thức này chưa được triển khai).
- **cleanup**: Dọn dẹp tài nguyên trước khi tắt chương trình, đóng các kết nối socket nếu có.

Quá trình tải file

1. Khởi Tạo Yêu Cầu Tải Tệp:

- Client gửi yêu cầu tới tracker để lấy danh sách các peer và thông tin về các phần tệp cần tải.
- Thông tin phản hồi bao gồm danh sách peer và điểm độ hiếm của từng phần tệp (pieces).
- Nếu không có peer nào, sẽ hiển thị lỗi "No peers available".

2. Tính Toán Các Phần Tệp Cần Tải:

- Dựa trên kích thước tệp và kích thước mỗi phần (PIECE_SIZE), số lượng phần tệp cần tải được tính toán.
- Tạo một tệp tạm thời trên ổ đĩa để chứa các phần này, mỗi phần được ghi vào tệp khi tải xong.

3. Khởi Tạo Đối Tượng Torrent:

- Một đối tượng **Torrent** được khởi tạo chứa các thông tin về tệp, bao gồm:

- Đường dẫn tệp.
 - Tên tệp.
 - Kích thước tệp.
 - Số lượng phần tệp.
 - Bitfield (bitarray dùng để theo dõi các phần đã tải).
- Bitfield được khởi tạo với giá trị là chuỗi các số 0 (tất cả các phần chưa tải).

4. Quản Lý Quá Trình Tải Xuống với Rarest-First:

- Client phân loại các phần tệp theo độ hiếm, bắt đầu tải từ những phần hiếm nhất.
- Các phần được sắp xếp và nhóm lại dựa trên độ hiếm. Các nhóm này sau đó được xáo trộn để tải đồng đều.
- Tạo các luồng để tải các phần tệp từ các peer ngẫu nhiên trong danh sách.

5. Cập Nhật Bitfield Sau Mỗi Phần Được Tải:

- Sau khi một phần tệp được tải thành công, bitfield của torrent sẽ được cập nhật để đánh dấu phần đó là đã tải.
- Cập nhật thông tin về phần đã tải sẽ được thông báo cho tracker, giúp tracker biết tình trạng tải của từng peer.

6. Thông Báo Tracker Khi Hoàn Thành Tải Một Phần:

- Sau khi tải xong mỗi phần tệp, client gửi thông báo đến tracker để cập nhật thông tin về tiến trình tải.
- Thông báo này giúp tracker theo dõi tiến độ của tất cả các peer và điều phối quá trình tải.

7. Lưu Tệp Hoàn Chỉnh:

- Sau khi tất cả các phần tệp được tải xuống, client ghép các phần lại và lưu tệp hoàn chỉnh vào đĩa.
- Nếu có phần tệp nào bị thiếu, client sẽ thực hiện lại quá trình tải cho đến khi tải đầy đủ các phần.
- Thông báo thành công hoặc lỗi sẽ được hiển thị cho người dùng.

4.4 Torrent

Lớp **Torrent** được hiện thực và sử dụng để đại diện cho một file torrent. Lớp này xử lý việc chia nhỏ file thành các phần và tính toán các hash của từng phần, cũng như tính toán hash toàn bộ file. Lớp này hữu ích trong việc chia sẻ file theo cách P2P, nơi mỗi phần của file có thể được tải xuống từ các peer khác nhau.

Thuộc tính

- **file_path:**
 - Đây là đường dẫn đến tệp torrent trên hệ thống.
 - Được sử dụng để xác định vị trí tệp cần chia sẻ hoặc tải.
- **file_name:**

- Tên tệp, lấy từ `file_path`.
- Được sử dụng khi hiển thị tên tệp trong giao diện người dùng hoặc khi cần quản lý các tệp tải về.
- **file_size:**
 - Kích thước của tệp tính bằng bytes, lấy từ hệ thống.
 - Được sử dụng để tính toán số lượng phần tệp cần tải.
- **num_pieces:**
 - Số lượng phần tệp cần chia nhỏ (tính bằng cách chia kích thước tệp cho kích thước của mỗi phần `PIECE_SIZE`).
 - Tham số này giúp xác định số lượng phần tệp cần được tải xuống.
- **bitfield:**
 - Chuỗi nhị phân thể hiện trạng thái của các phần tệp (0: chưa tải, 1: đã tải).
 - Được cập nhật liên tục trong quá trình tải để theo dõi trạng thái các phần tệp.
- **pieces:**
 - Danh sách các phần tệp sau khi tệp được chia nhỏ.
 - Mỗi phần tệp có thể được tải riêng biệt từ các peer khác nhau.
- **info_hash:**
 - Mã băm của tệp torrent, được tính toán từ nội dung của tệp.
 - Được sử dụng để nhận dạng tệp torrent duy nhất và kiểm tra tính toàn vẹn của tệp trong quá trình tải.

4.5 Helper function

Các hàm helper dưới đây được sử dụng trong ứng dụng chuyển file P2P để hỗ trợ các thao tác cơ bản như gửi và nhận tin nhắn, tính toán hash, và quản lý kết nối.

Hàm `recvall(sock, n)`

Mô tả: Hàm này nhận dữ liệu từ một kết nối socket. Nó đảm bảo rằng đủ `n` byte được nhận, tiếp tục nhận dữ liệu cho đến khi đạt được số lượng yêu cầu. Nếu kết thúc file (EOF) được phát hiện, hàm trả về `None`.

Công dụng trong P2P File Transfer:

- Giúp nhận toàn bộ dữ liệu một cách an toàn, đảm bảo rằng kích thước dữ liệu nhận được đúng như yêu cầu.
- Dùng để nhận cả các tin nhắn và phần của file trong quá trình truyền tải.

Hàm `recv_msg(sock)`

Mô tả: Hàm này nhận một tin nhắn từ socket. Đầu tiên, nó nhận độ dài của tin nhắn (4 byte) và sau đó nhận dữ liệu thực tế theo độ dài đó.

Công dụng trong P2P File Transfer:

- Dùng để nhận các tin nhắn từ peer khác, ví dụ như yêu cầu tải xuống một phần của file.

- Giúp xử lý việc nhận dữ liệu với kích thước không xác định (dựa trên độ dài tin nhắn).

Hàm `send_msg(sock, msg)`

Mô tả: Hàm này gửi một tin nhắn qua kết nối socket. Nó đầu tiên đóng gói tin nhắn thành một chuỗi bytes bằng cách sử dụng `pickle.dumps()` và gửi đi, kèm theo độ dài của tin nhắn.

Công dụng trong P2P File Transfer:

- Dùng để gửi các tin nhắn như yêu cầu tải file, thông báo lỗi hoặc thành công trong quá trình chuyển file.
- Đảm bảo tin nhắn được truyền đi đúng cách với định dạng và kích thước đúng.

Hàm `generate_port()`

Mô tả: Hàm này tạo một số cổng ngẫu nhiên từ 10000 đến 65535, giúp tạo ra các cổng khác nhau cho các kết nối peer-to-peer.

Công dụng trong P2P File Transfer:

- Dùng để tạo các cổng kết nối ngẫu nhiên cho các peer trong mạng P2P, đảm bảo mỗi peer có thể giao tiếp độc lập với các peer khác.
- Giúp tránh xung đột cổng trong môi trường nhiều peer.

4.6 Chiến thuật tải file

Ở đây, nhóm sử dụng chiến thuật Rare First. Đây là một phương pháp tối ưu hóa quá trình tải tệp, trong đó bạn ưu tiên tải các phần tệp hiếm (rare parts) trước, thay vì tải các phần tệp phổ biến hoặc dễ dàng có sẵn. Đây là một chiến lược rất hữu ích trong các hệ thống chia sẻ tệp P2P, vì nó giúp giảm thiểu khả năng tệp bị thiếu hoặc không thể tải xuống do tệp đó có ít người chia sẻ hơn.

Cách hoạt động của chiến thuật Rare First:

- Trong một hệ thống P2P, khi một người dùng tải xuống một tệp, tệp đó thường được chia thành nhiều phần nhỏ. Các phần này có thể được chia sẻ và tải xuống từ nhiều người dùng khác nhau.
- Các phần tệp phổ biến (được nhiều người chia sẻ) có xu hướng được tải xuống nhanh hơn, trong khi các phần tệp hiếm (ít người chia sẻ) có thể mất thời gian hơn.
- Rare First ưu tiên tải các phần tệp hiếm trước để đảm bảo rằng các phần quan trọng hoặc ít phổ biến của tệp không bị thiếu trong quá trình tải xuống. Điều này giúp duy trì sự ổn định của quá trình tải tệp, đồng thời giảm khả năng không tải được tệp hoàn chỉnh.

Chiến lược này giúp cải thiện tốc độ tải xuống tổng thể và giúp đảm bảo rằng không có phần nào của tệp bị thiếu, đặc biệt trong các hệ thống chia sẻ tệp P2P nơi mà sự hiện diện của các phần tệp có thể thay đổi nhanh chóng tùy thuộc vào số lượng người chia sẻ.

5 Demo ứng dụng và đánh giá

5.1 Server

Chúng ta sẽ chạy server trên 1 máy để tiến hành khởi động hệ thống, sau khi khởi tạo sẽ có kết quả như sau:

```
PS D:\CN> cd .\C03093_Computer_Network\  
PS D:\CN\C03093_Computer_Network> python  
peer.py  
Listening for peer connections on localhost:20938
```

Hình 1: Khởi động Tracker

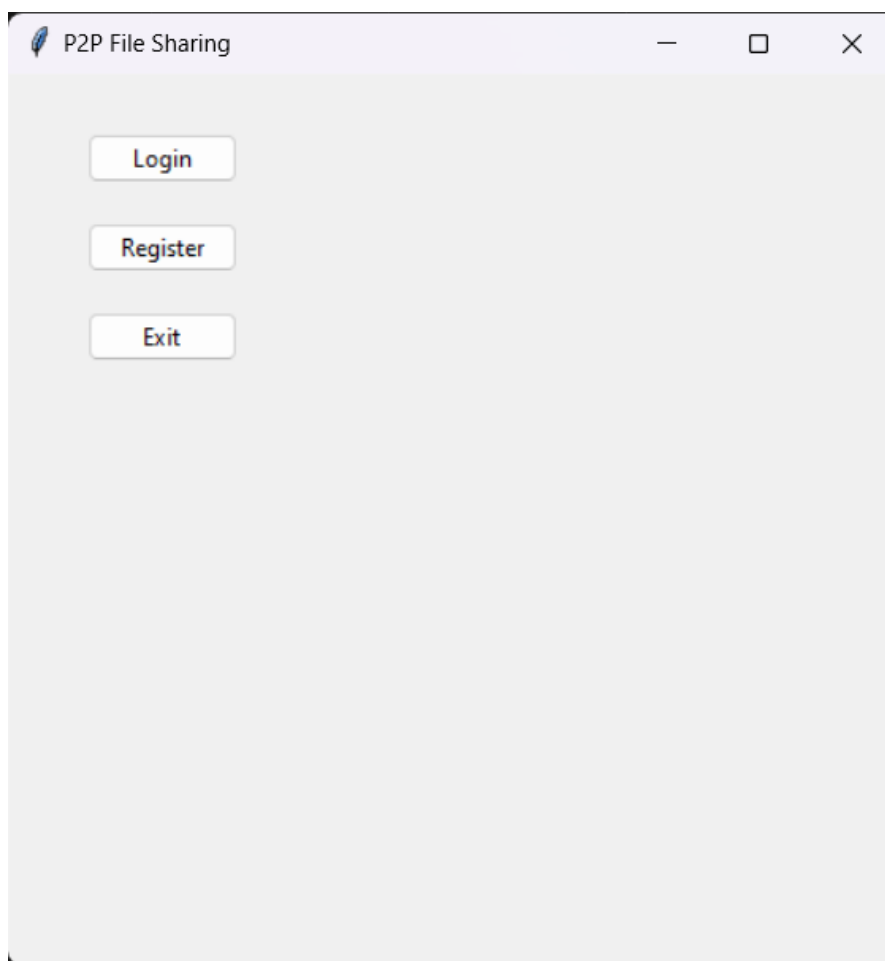
5.2 Peer

Sau khi tiến hành khởi động server, việc tiếp theo chúng ta sẽ cần một account để sử dụng hệ thống này. Khởi tạo giao diện người dùng với câu lệnh:

```
PS D:\CN\C03093_Computer_Network> python  
peer.py  
Listening for peer connections on localhost:36017
```

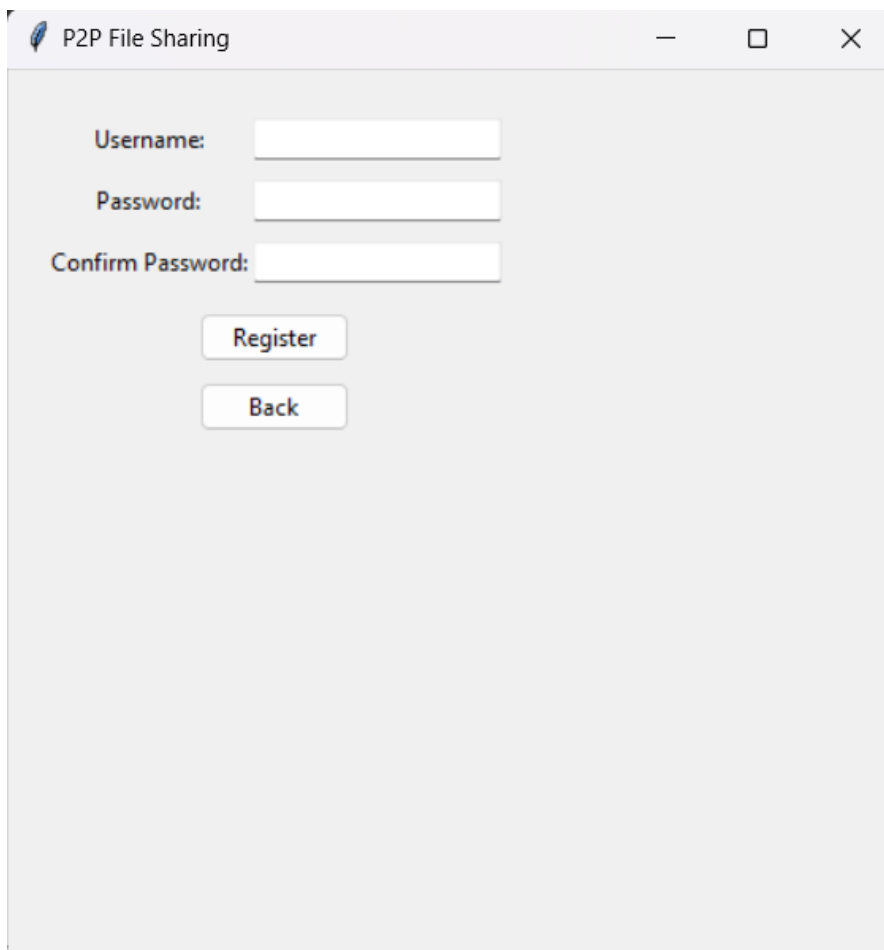
Hình 2: Khởi động Peer

Sau khi khởi tạo, một giao diện sẽ xuất hiện



Hình 3: Giao diện người dùng

Người dùng chọn Register để tạo tài khoản mới



P2P File Sharing

Username:

Password:

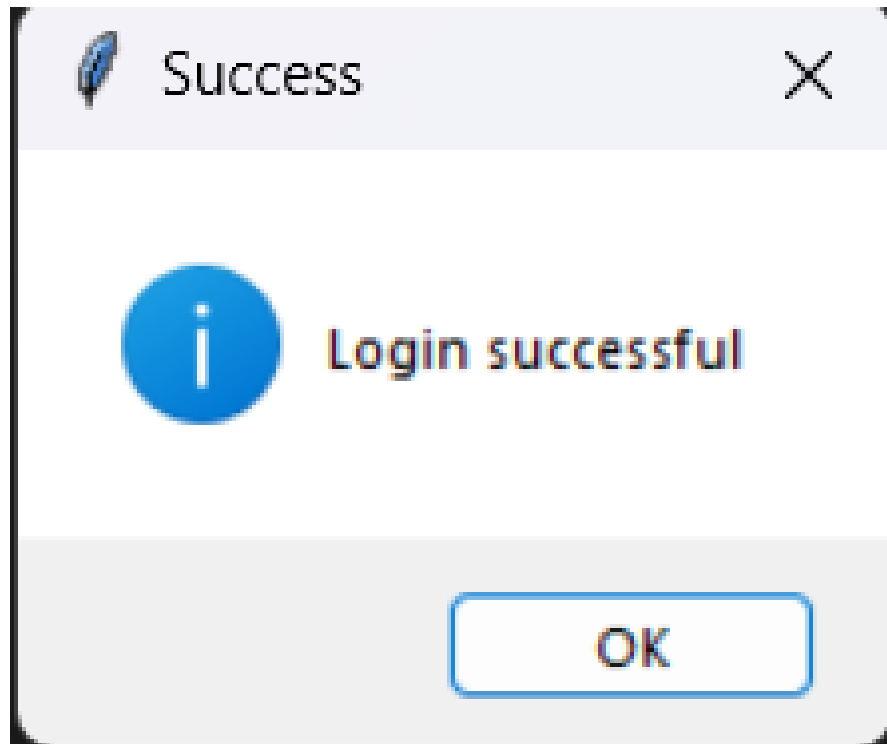
Confirm Password:

Register

Back

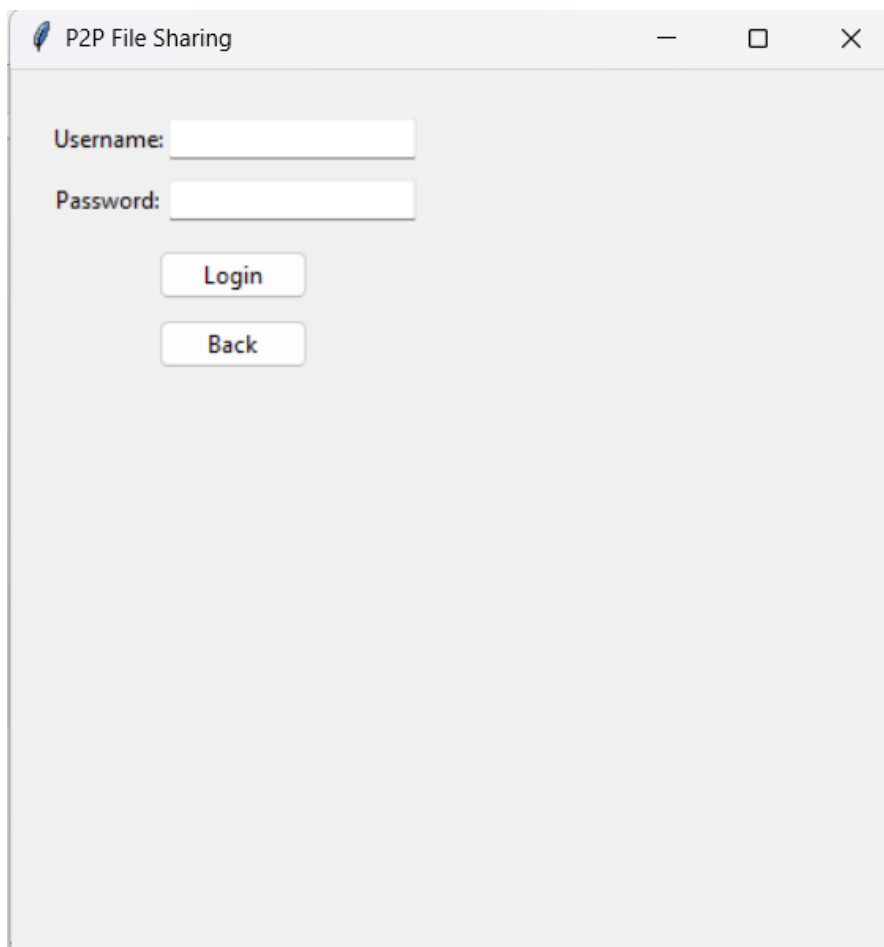
Hình 4: Đăng ký người dùng mới

Sau đó điền thông tin vào các ô input trên giao diện như họ tên, username, password. Khi đăng ký xong, sẽ xuất hiện thông báo trở lại trang đăng nhập để sử dụng dịch vụ.



Hình 5: Nhập thông tin đăng nhập

Sau đó ta quay trở lại trang đăng nhập (nút Login trên client) để tiến hành đăng nhập



P2P File Sharing

Username:

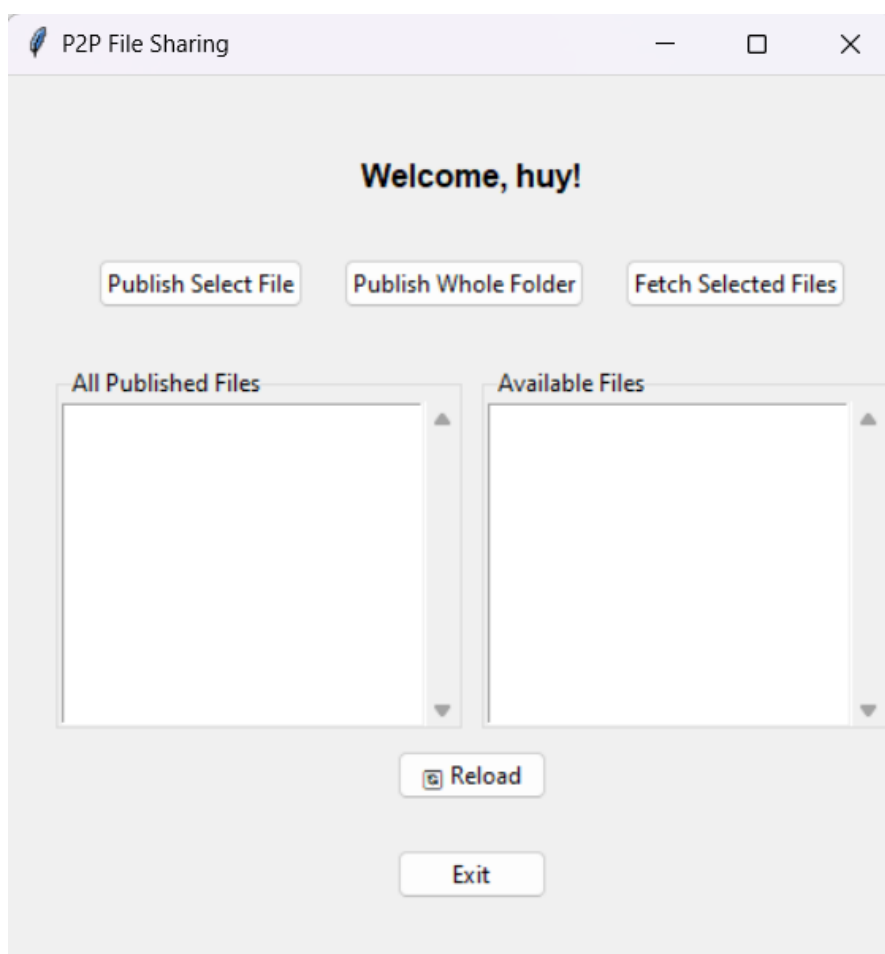
Password:

Login

Back

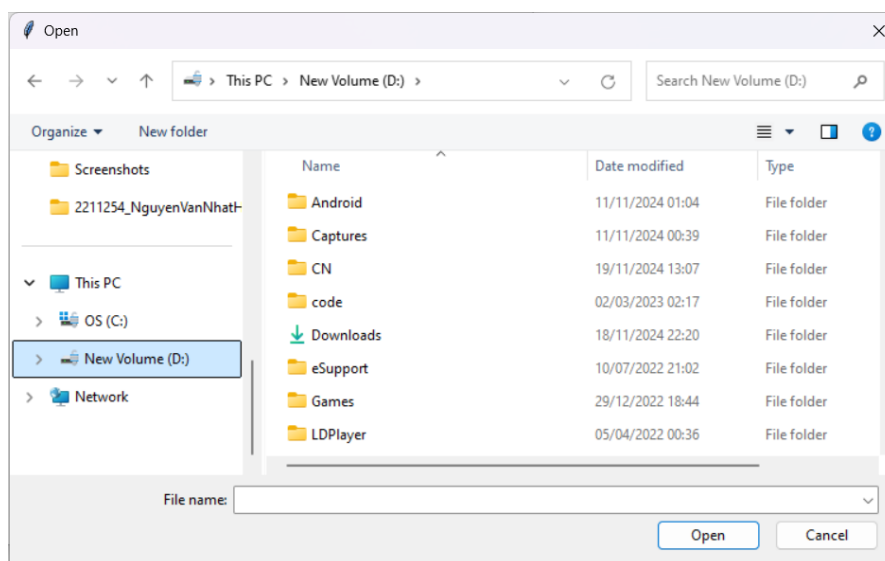
Hình 6: Đăng nhập

Sau khi đăng nhập sẽ xuất hiện giao diện nơi chứa danh sách các file có sẵn trên tracker. Do lúc này chưa có file nào được upload lên nên danh sách này đang trống.



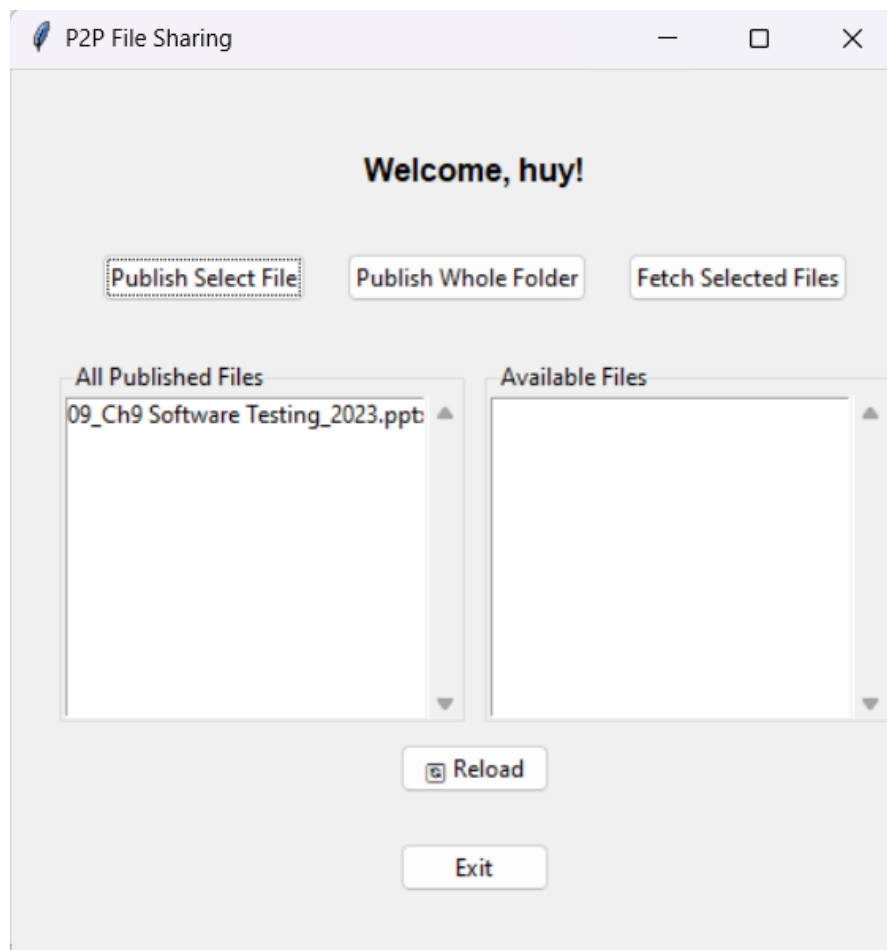
Hình 7: Giao diện khi đăng nhập thành công

Lúc này nếu bạn có thể chọn **Publish Select File** để upload 1 hay nhiều file có sẵn từ máy của bạn hoặc chọn **Publish Whole Folder** để upload nguyên 1 folder có sẵn từ máy bạn.



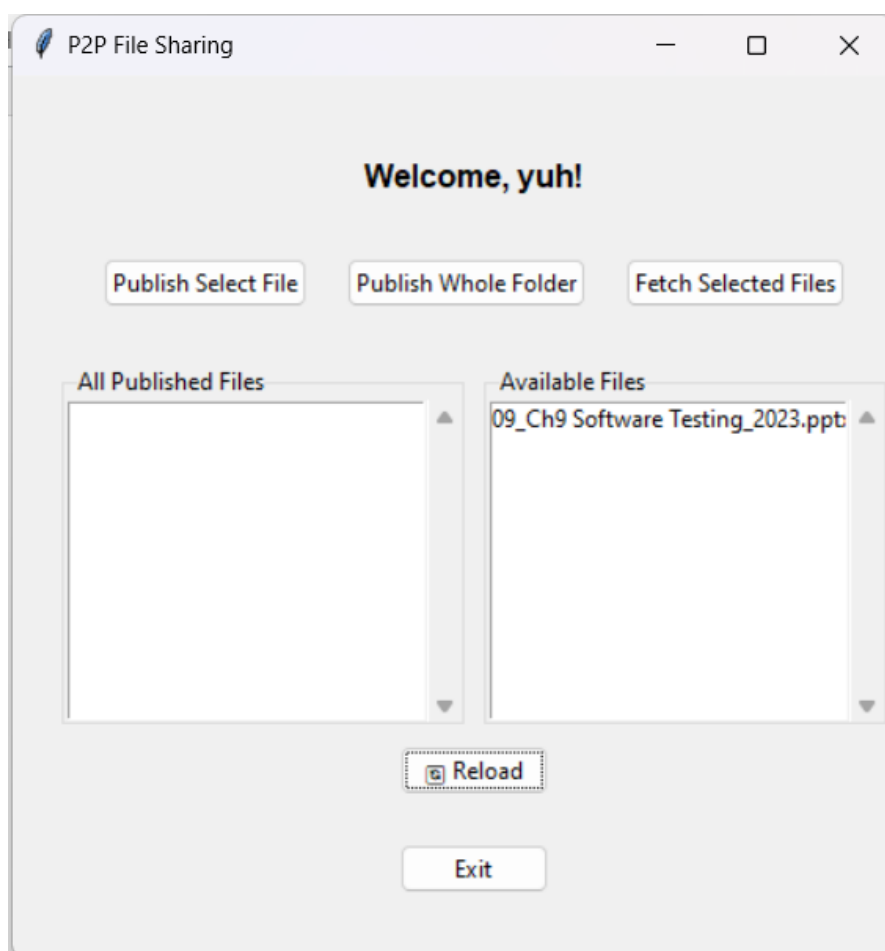
Hình 8: Chọn Files để Upload

Sau khi upload thành công sẽ có giao diện như sau:



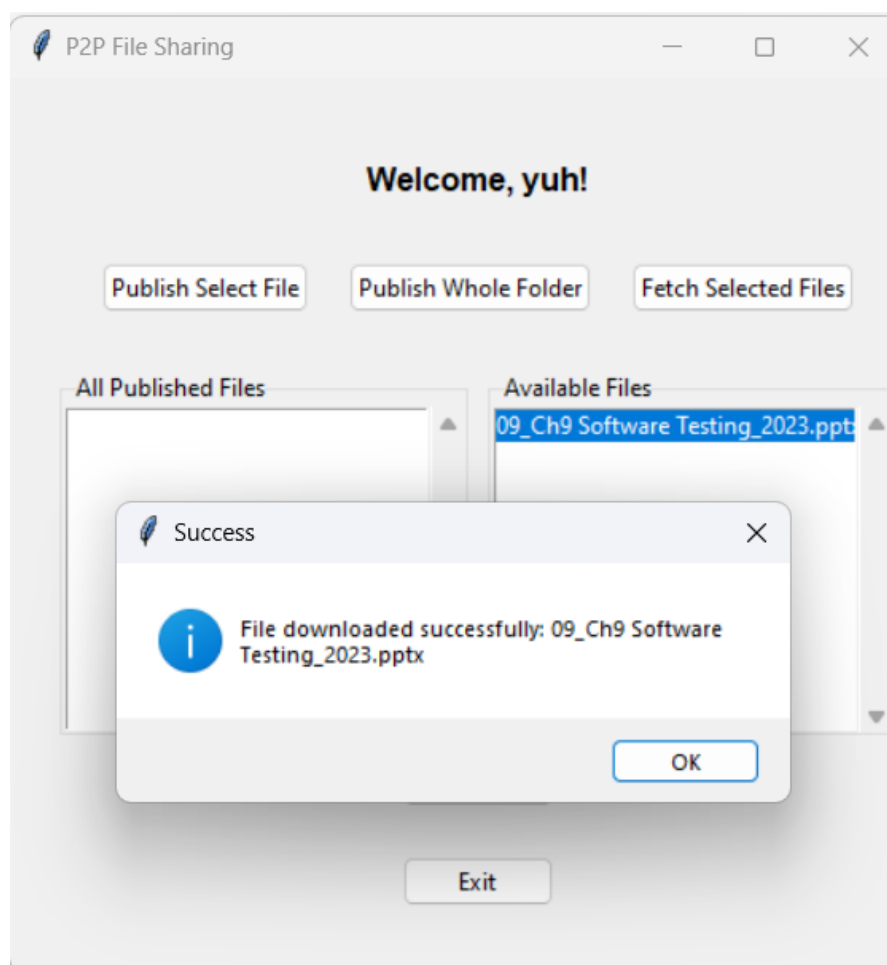
Hình 9: Upload thành công

Lúc này khi một tài khoản khác đăng nhập vào thì ở phía Available Files sẽ có các Files có sẵn.



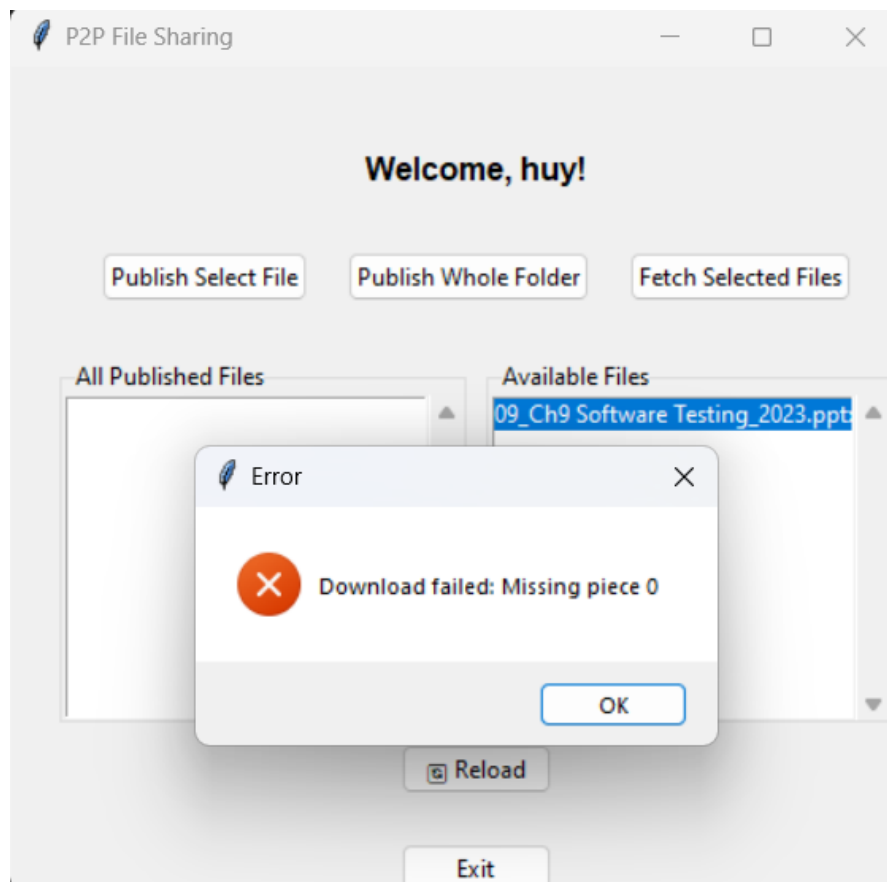
Hình 10: Hiển thị các Available Files

Lúc này bạn sẽ có thể chọn một File muốn tải về sau đó chọn Fetch Selected Files để Download các Files muốn tải về. Trong trường hợp các Peer khác chưa đủ các Pieces của các Files trên thì sẽ thông báo thành công tải về.



Hình 11: Thông báo Download thành công

Trong trường hợp lúc tải về mà thiếu mất Pieces cần thiết thì sẽ có thông báo về các Pieces thiếu.



Hình 12: Thông báo Download không thành công

Nếu bạn muốn đăng xuất có thể chọn Exit trên giao diện đang có.

5.3 Thí nghiệm tải file

Tiến hành thí nghiệm như sau: một peer A upload 1 file có dung lượng khoảng 1GB, sau khi upload thành công, 2 peer khác gọi là peer B và peer C tiến hành tải file. Peer B tiến hành tải file trước, sau đó peer C cũng đồng thời tiến hành tải file. Lúc này các **piece** được tải theo chiến lược rarest-first, peer B vừa download piece từ peer A, vừa upload piece cho peer C.

```
2024-11-28 09:41:59,280 - Connected with tracker server at localhost:5050
2024-11-28 09:41:59,281 - Peer activity log started
2024-11-28 09:41:59,281 - Peer initialized with IP: localhost and Port: 60526
2024-11-28 09:42:00,339 - Listening for peer connections on localhost:60526
2024-11-28 09:42:00,341 - Received list of available files from tracker: []
2024-11-28 09:42:59,303 - Published file: yoochoose-clicks.csv, 1519802169 bytes, 7ea45db917af36bf6dd1a333b3b3255715df7d6c
2024-11-28 09:43:01,262 - Received list of available files from tracker: []
2024-11-28 09:43:08,624 - New connection from ('127.0.0.1', 53854)
2024-11-28 09:43:08,625 - Handling connection from ('127.0.0.1', 53854)
```

Hình 13: Peer A upload file

Peer B đăng nhập thành công vào hệ thống tiến hành download file

```

1924-11-28 09:43:03,197 Received list of available files from tracker: [{'filename': 'yoochoose-clicks.csv', 'size': 1519802169, 'info_hash': '7ea
1924-11-28 09:43:06,020 Fetched peer list: ([1, 'localhost', 60526])
1924-11-28 09:43:06,020 Initiating download file name: yoochoose-clicks.csv, file size: 1519802169 bytes, number of pieces: 2899
1924-11-28 09:43:08,620 Piece available: ([1, 'localhost', 60526])
2024-11-28 09:43:08,623 Pieces rarity: ((2821, 0), (1198, 0), (1429, 0), (2062, 0), (2774, 0), (2167, 0), (1182, 0), (1686, 0), (1460, 0), (1049,
1924-11-28 09:43:08,626 Handshake response from localhost:60526: Handshake successful
1924-11-28 09:43:08,627 Requesting piece 2821 to localhost:60526
1924-11-28 09:43:08,628 Handshake response from localhost:60526: Handshake successful
1924-11-28 09:43:08,629 Requesting piece 1998 to localhost:60526
1924-11-28 09:43:08,638 Successfully downloaded piece 2821 from peer_id 1 localhost:60526
1924-11-28 09:43:08,638 Successfully downloaded piece 1998 from peer_id 1 localhost:60526
1924-11-28 09:43:08,640 Handshake response from localhost:60526: Handshake successful

```

Hình 14: Peer B starting download

Các piece được sắp xếp theo độ hiếm, như hình trên các piece có điểm càng thấp sẽ được sắp lên đầu tiên để tiến hành tải trước.

```
2024-11-28 09:43:14,401 - New connection from ('127.0.0.1', 56639)
2024-11-28 09:43:14,401 - Received request from ('127.0.0.1', 56638): get_piece
2024-11-28 09:43:14,401 - Handling connection from ('127.0.0.1', 56639)
2024-11-28 09:43:14,402 - Received request from ('127.0.0.1', 56639): handshake
2024-11-28 09:43:14,402 - Sent piece 1284 of repo_edith\yoochoose-clicks.csv to peer 2
```

Hình 15: Peer C request from peer B while peer B download from peer A

Peer C gửi yêu cầu về piece cần download cho peer B khi peer B đang tải file.

```
2024-11-28 09:43:18.978 - Fetched peer list: [(1, 'localhost', 60526), (5, 'localhost', 40541)]
2024-11-28 09:43:19.078 - Initiating download file name: yoochoose-clicks, url, file size: 1519862169 bytes, number of pieces: 2899
2024-11-28 09:43:19.361 - Piece available: [(1, 'localhost', 60526), (5, 'localhost', 40541)]
2024-11-28 09:43:14.395 - Pieces parity: [(1284, 0), (1843, 0), (1704, 0), (2334, 0), (2055, 0), (2669, 0), (1479, 0), (1999, 0), (2074, 0), (5
2024-11-28 09:43:14.398 - Handshake response from localhost:40541: Handshake successful
2024-11-28 09:43:14.400 - Requesting piece 1284 from localhost:40541
2024-11-28 09:43:14.402 - Handshake response from localhost:40541: Handshake successful
2024-11-28 09:43:14.403 - Successfully downloaded piece 1284 from peer id 5 localhost:40541
```

Hình 16: Peer C nhận được list các peer và tiến hành tải file

Các piece cũng được sắp xếp theo độ hiếm của chúng. Peer C tiến hành kết nối và tải các piece này về.

```

1404831 2024-11-28 09:43:27.310 - Received UPDATE_PIECE_POINT request from ('127.0.0.1', 53834)
1404832 2024-11-28 09:43:27.310 - Updating piece information:
1404833 2024-11-28 09:43:27.310 - Peer ID: 2
1404834 2024-11-28 09:43:27.310 - File hash: 7ea45db917af36bf6dd1a33b3b3255715df7d6c
1404835 2024-11-28 09:43:27.310 - Piece index: 27
1404836 2024-11-28 09:43:27.311 - Before update:
1404837 2024-11-28 09:43:27.311 - Current bitfield: 1100011101000100001010101010010111100011010100010000100000000001001111111100011110
1404838 2024-11-28 09:43:27.311 - Current piece points: [2, 2, 2, 1, 1, 2, 2, 2, 2, 1, 2, 2, 1, 1, 1, 2, 2, 1, 1, 1, 2, 1, 2, 2, 1, 1, 2, 1, 2, 1, 2, 1, 1]
1404839 2024-11-28 09:43:27.311 - After update:
1404840 2024-11-28 09:43:27.311 - Updated bitfield: 1100011101000100001010101010111000111001010100010000000000000001001111111100011110
1404841 2024-11-28 09:43:27.311 - Updated piece points: [2, 2, 2, 1, 1, 2, 2, 2, 2, 1, 2, 2, 1, 1, 1, 2, 2, 1, 1, 1, 2, 1, 2, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 1]
1404842 2024-11-28 09:43:27.311 - New point value for piece 27: 2
1404843 2024-11-28 09:43:27.312 - Waiting for data from ('127.0.0.1', 53834)
1404844 2024-11-28 09:43:27.313 - Received UPDATE_PIECE_POINT request from ('127.0.0.1', 53834)
1404845 2024-11-28 09:43:27.313 - Updating piece information:
1404846 2024-11-28 09:43:27.313 - Peer ID: 2
1404847 2024-11-28 09:43:27.313 - File hash: 7ea45db917af36bf6dd1a33b3b3255715df7d6c
1404848 2024-11-28 09:43:27.313 - Piece index: 32
1404849 2024-11-28 09:43:27.314 - Before update:
1404850 2024-11-28 09:43:27.314 - Current bitfield: 1100011101000100001010101010010111100011010100010000100000000001001111111100011110
1404851 2024-11-28 09:43:27.314 - Current piece points: [2, 2, 2, 1, 1, 2, 2, 2, 2, 1, 2, 2, 1, 1, 1, 2, 2, 1, 1, 1, 2, 1, 2, 2, 1, 2, 1, 2, 2, 1, 2, 1, 1]
1404852 2024-11-28 09:43:27.314 - After update:
1404853 2024-11-28 09:43:27.314 - Updated bitfield: 1100011101000100001010101010111000111001010100010000100000000001001111111100011110
1404854 2024-11-28 09:43:27.314 - Updated piece points: [2, 2, 2, 1, 1, 2, 2, 2, 2, 1, 2, 2, 1, 1, 1, 2, 2, 1, 1, 1, 2, 1, 2, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 1]
1404855 2024-11-28 09:43:27.314 - New point value for piece 32: 2
1404856 2024-11-28 09:43:27.314 - Waiting for data from ('127.0.0.1', 53834)
1404857 2024-11-28 09:43:27.321 - Received UPDATE_PIECE_POINT request from ('127.0.0.1', 53834)
1404858 2024-11-28 09:43:27.321 - Updating piece information:
1404859 2024-11-28 09:43:27.322 - Peer ID: 2
1404860 2024-11-28 09:43:27.322 - File hash: 7ea45db917af36bf6dd1a33b3b3255715df7d6c

```

Hình 17: Bitfield và độ hiếm của các piece liên tục được update cho tracker

```
2024-11-28 09:43:22,906 - New connection from ('127.0.0.1', 60142)
2024-11-28 09:43:22,907 - Handling connection from ('127.0.0.1', 60142)
2024-11-28 09:43:22,907 - Received request from ('127.0.0.1', 60142): handshake
2024-11-28 09:43:22,908 - File yoochoose-clicks.csv saved successfully
```

Hình 18: Peer B tải thành công

Quá trình tải file của peer B được ghi nhận khoảng 16,888s.

```
2024-11-28 09:43:32,690 - Successfully downloaded piece 1999 from peer_id 1 localhost:60526
2024-11-28 09:43:32,690 - Requesting piece 2461 to localhost:60526
2024-11-28 09:43:32,693 - Successfully downloaded piece 2461 from peer_id 1 localhost:60526
2024-11-28 09:43:36,047 - File yoochoose-clicks.csv saved successfully
```

Hình 19: Peer C tải thành công

Quá trình tải file của peer C được ghi nhận khoảng 25,069s. Quá trình tải file từ peer C tốn nhiều thời gian hơn do khi peer C tiến hành yêu cầu piece đến peer B thì đôi lúc peer B chưa sở hữu piece đó.

Khi các peer sở hữu đầy đủ các piece. Một peer D khác đăng nhập và tiến hành tải file

```
2024-11-28 09:46:26,534 - Fetched peer list: [(1, 'localhost', 60526), (5, 'localhost', 40541), (2, 'localhost', 36900)]
2024-11-28 09:46:26,534 - Initiating download file name: yoochoose-clicks.csv, file size: 1519802160 bytes, number of pieces: 2899
2024-11-28 09:46:29,011 - Piece available: [(1, 'localhost', 60526), (5, 'localhost', 40541), (2, 'localhost', 36900)]
2024-11-28 09:46:29,014 - Pieces rarity: [(1210, 2), (1223, 2), (788, 2), (2454, 2), (313, 2), (1984, 2), (1101, 2), (396, 2), (2193, 2), (1118, 2), (
2024-11-28 09:46:29,017 - Handshake response from localhost:60526: Handshake successful
```

Hình 20: Peer D bắt đầu tải

```
2024-11-28 09:46:49,031 - Requesting piece 1374 to localhost:40541
2024-11-28 09:46:49,036 - Successfully downloaded piece 1374 from peer_id 5 localhost:40541
2024-11-28 09:46:52,039 - File yoochoose-clicks.csv saved successfully
```

Hình 21: Peer D hoàn tất quá trình tải

Quá trình tải được ước tính khoảng 23,079s. Qua thí nghiệm cho thấy, việc xử lý kết nối và tải piece đa luồng chưa được tối ưu làm cho thời gian tải toàn bộ các piece chưa được tối ưu.



Tài liệu tham khảo

- [1] BitTorrentSpecification. Available: <https://wiki.theory.org/BitTorrentSpecification>.