Đề bài: Tạo 1 hệ thống trao đổi tiền điện tử phi tập trung với nhiều chức năng tương tự như sàn giao dịch Uniswap. Ngoài ra tạo mã thông báo ERC-20 của riêng mình. Hệ thống phải có khả năng phục hồi trước các cuộc tấn công.

Project có 3 phần chính:

* Contract/token.sol
* Contract/exchange.sol
* Web\_app/exchange.js

npx hardhat node

npx hardhat run --network localhost scripts/deploy\_token.js

npx hardhat run --network localhost scripts/deploy\_exchange.js

3. Tạo và triển khai token.sol

Trong phần đầu của dự án này, bạn sẽ tạo và triển khai token ERC-20 của riêng mình. ERC-20, như đã đề cập trong lớp học, là một tiêu chuẩn để triển khai fungible token. May mắn cho chúng ta, phần lớn mã của tiêu chuẩn ERC-20 đã được viết và được công khai. Trong dự án này, chúng ta sẽ sử dụng triển khai tiêu chuẩn ERC-20 từ dự án OpenZeppelin. Hãy đảm bảo bạn hiểu rõ về tiêu chuẩn ERC-20, cũng như việc triển khai ERC-20 và Ownable từ OpenZeppelin.

Sau khi đọc qua mã khởi đầu, hoàn thành các bước sau trong tệp contracts/token.sol và web\_app/exchange.js:

1. Chọn một tên vui nhộn (nhưng phù hợp!) cho token của bạn. Bất kỳ tên nào cũng được, vì vậy hãy vui lòng sáng tạo và đặt tên gì đó độc đáo. Đặt chuỗi \_name của private trong token.sol là tên của token của bạn. Ngoài ra, cập nhật biến token\_name ở đầu của exchange.js để có cùng tên.
2. Quyết định một ký hiệu ngắn cho token của bạn (ví dụ: ETH, thay vì Ethereum). Đặt chuỗi \_symbol là ký hiệu đó và cập nhật biến symbol của token ở đầu của exchange.js với cùng tên.
3. Triển khai các hàm mint(uint amount) và disable\_mint(). Hàm mint là một hàm công khai tạo ra số lượng token được chỉ định. Mặc dù hợp đồng ERC-20 của OpenZeppelin không cung cấp hàm mint(), bạn vẫn có thể gọi \_mint() một cách phù hợp để thực hiện việc này. Hàm disable\_mint() làm cho việc gọi mint() không bao giờ thành công nữa. Do đó, triển khai mint của bạn phải báo lỗi nếu disable\_mint() đã được gọi. Lưu ý cũng có từ khóa modifier Ownable trên cả hai hàm, điều này đảm bảo chỉ có quản trị viên hợp đồng mới có thể gọi hai hàm này. Phong cách mint()/disable mint() là một cách đơn giản để tạo ra ban đầu các token và đảm bảo rằng tổng cung sẽ không thay đổi sau khi cung cấp token đã được tạo.
4. Triển khai hợp đồng token. Sao chép địa chỉ và ABI của hợp đồng token vào biến token\_address và token\_bi trong exchange.js.
5. Cuối cùng, sao chép địa chỉ của hợp đồng token vào biến tokenAddr trong exchange.sol. Hãy nhớ sử dụng phiên bản đã được kiểm tra đúng của địa chỉ trong hợp đồng. Mỗi khi bạn triển khai lại hợp đồng token, bạn phải lặp lại bước này.

Sau khi hoàn thành các bước trên, bạn sẽ có token riêng của mình đã được triển khai trên Hardhat!

Bây giờ bạn có thể thử deploy exchange contract của mình. Sao chép địa chỉ và ABI của hợp đồng token vào biến exchange\_address và exchange\_abi trong exchange.js. Tất cả các chức năng trừ việc thiết lập ban đầu cho nhóm thanh khoản (đã được triển khai sẵn) sẽ không có. Tuy nhiên, nếu bạn đã hoàn thành Phần 2 và quá trình triển khai một cách đúng đắn, bạn không nên thấy bất kỳ lỗi nào xuất hiện trên bảng điều khiển trình duyệt của bạn và bạn nên nhìn thấy tỷ lệ trao đổi 1-1 giữa ETH và token của bạn. Sẽ có 5000 ETH và 5000 token của bạn hiển thị dưới mục "Current Liquidity”.

4. Thiết lập exchange cơ bản của bạn

Trong phần này của bài tập, bạn sẽ triển khai chức năng cơ bản cho sàn giao dịch tiền điện tử của mình. Sàn giao dịch này được mô phỏng theo Uniswap V1. Sàn giao dịch của bạn chỉ cho phép hoán đổi giữa Token của bạn và ETH thử nghiệm. Các thay đổi trong phần này chủ yếu ảnh hưởng đến hai tệp: exchange.js và exchange.sol. Hãy làm quen với mã khởi đầu cho các tệp đó.

Một sàn giao dịch phi tập trung bao gồm hai loại người tham gia: nhà cung cấp thanh khoản (LP) và người giao dịch. Đối với một cặp giao dịch giữa hai loại token, nhà cung cấp thanh khoản cung cấp một số lượng tương đương của cả hai loại thanh khoản (trong trường hợp này là ETH và token của bạn). Khi người giao dịch hoán đổi giữa hai loại tiền tệ này, họ sẽ thêm một số lượng tiền tệ vào nhóm thanh khoản và nhận được một giá trị tương đương của loại tiền tệ khác từ nhóm thanh khoản. Tỷ giá hoán đổi giữa hai loại tiền tệ được xác định bằng công thức tổng quát:

Cho x là số lượng tiền tệ A trong nhóm thanh khoản và y là số lượng tiền tệ B.

Gọi k là một hằng số. Sau mỗi lần hoán đổi, ta có điều kiện:

x \* y = k

Trong mỗi lần hoán đổi, sàn giao dịch phải chuyển đúng số lượng tiền tệ hoán đổi tương ứng để đảm bảo nhóm thanh khoản luôn nằm trên đường cong sản phẩm không đổi. Giá của tiền tệ B theo giá trị của tiền tệ A có thể được tính là x/y, và giá của tiền tệ A theo giá trị của tiền tệ B có thể được tính là y/x. Mỗi lần hoán đổi sẽ thay đổi tỷ giá giữa hai loại tiền tệ. Điều này có ý nghĩa, vì mỗi lần hoán đổi là một dấu hiệu của nhu cầu cho một loại tiền tệ cụ thể. Hiệu ứng của mỗi lần hoán đổi đối với giá trị của các loại tiền tệ sẽ quan trọng trong phần 6.

Khi một nhà cung cấp thanh khoản thêm thanh khoản, họ phải cung cấp giá trị tương đương của cả tiền tệ A và tiền tệ B, dựa trên tỷ giá hoán đổi hiện tại. Lưu ý rằng việc thêm thanh khoản sẽ không thay đổi tỷ giá hoán đổi giữa các loại tiền tệ, nhưng sẽ tăng giá trị của hằng số k. Tương tự, khi một nhà cung cấp thanh khoản muốn rút lại thanh khoản của mình, họ phải rút số lượng tương đương của cả tiền tệ A và tiền tệ B. Do đó, giá trị k sẽ giảm nhưng tỷ giá hoán đổi sẽ không thay đổi.

Điều này có một hệ quả đáng chú ý khác: vì một nhà cung cấp thanh khoản chỉ có thể rút lại số tiền tương đương của mỗi loại tiền tệ, họ không thực sự có quyền rút lại đúng khoản đầu tư ban đầu của họ (theo số lượng của mỗi loại token). Thay vào đó, việc cung cấp thanh khoản tương đương với việc sở hữu một phần trăm trong nhóm thanh khoản, và nhà cung cấp sẽ có quyền rút lại phần trăm đó của từng nguồn cung cho các loại tiền tệ đó (với điều kiện tỷ lệ của họ không bị làm mờ bởi nhà cung cấp khác), ngay cả khi phần trăm đó không chính xác với số lượng token ban đầu mà họ cung cấp. Điều này có nghĩa là nhà cung cấp thanh khoản có thể gánh chịu sự mất mát tạm thời, nếu giá trị của những gì họ được phép rút lại nhỏ hơn giá trị ban đầu mà họ đã đầu tư. Nếu bạn không quen thuộc với khái niệm mất mát tạm thời, hãy xem lại bài giảng và phần về sàn giao dịch phi tập trung.

Với những điều trên, bạn sẽ triển khai chức năng cơ bản của sàn giao dịch của mình. Chúng tôi đã lo việc khởi tạo nhóm thanh khoản cho bạn bằng cách triển khai và gọi hàm createPool. Chúng tôi lấy ETH và token từ địa chỉ đầu tiên để khởi tạo nhóm thanh khoản, và bạn không cần theo dõi số lượng này khi theo dõi nhà cung cấp thanh khoản. Để cho phép các địa chỉ khác nhận token và những địa chỉ khác cung cấp thanh khoản, họ phải trước tiên hoán đổi cho token trên sàn giao dịch. Trong exchange.sol, bạn sẽ triển khai các hàm sau:

* function addLiquidity() external payable: Thêm thanh khoản vào nhóm thanh khoản nếu nhà cung cấp có đủ ETH và token (nếu không, giao dịch sẽ thất bại). Người gọi sẽ gửi ETH vào hợp đồng, có thể truy cập bằng cách sử dụng msg.value. Hàm này cũng sẽ chuyển số lượng tương ứng của token dựa trên tỷ giá hoán đổi hiện tại từ địa chỉ của người gửi đến hợp đồng (sử dụng phương thức transfer hoặc transferFrom của token) và cập nhật trạng thái của sàn giao dịch. Giao dịch sẽ thất bại nếu số dư của nhà cung cấp không đủ. Xem Phần 9 để biết lời khuyên về cách giữ theo dõi thanh khoản một cách tốt nhất.
* function removeLiquidity(uint amountETH) public payable: Rút một lượng thanh khoản cụ thể từ nhóm thanh khoản (nếu nhà cung cấp được phép rút số lượng thanh khoản đó) và cập nhật trạng thái của sàn giao dịch tương ứng. amountETH là số lượng ETH mà nhà cung cấp thanh khoản muốn rút, vì vậy họ sẽ nhận tổng giá trị tương đương với 2 \* amountETH sau khi họ nhận được token và ETH. Đảm bảo cập nhật số lượng thanh khoản được cung cấp bởi mỗi nhà cung cấp thanh khoản một cách phù hợp. Hàm sẽ thất bại nếu người dùng cố gắng rút nhiều hơn số lượng thanh khoản mà họ được phép, hoặc nếu họ cố gắng làm cạn dự trữ ETH hoặc token thành 0.
* function removeAllLiquidity() external payable: Rút số lượng thanh khoản tối đa mà người gửi được phép rút và cập nhật trạng thái của sàn giao dịch tương ứng. Đồng thời, đảm bảo cập nhật số lượng thanh khoản được cung cấp bởi mỗi nhà cung cấp thanh khoản. Tương tự, hàm này sẽ thất bại nếu nhà cung cấp thanh khoản làm cạn dự trữ ETH hoặc token thành 0.
* function swapTokensForETH(uint amountTokens) external payable: Đổi số lượng token cho giá trị tương đương của ETH và cập nhật trạng thái của sàn giao dịch tương ứng. Nếu nhà cung cấp không có đủ token để thực hiện giao dịch, giao dịch sẽ thất bại. Ngoài ra, nếu việc thực hiện giao dịch này sẽ làm cho toàn bộ ETH trong nhóm thanh khoản bị rút hết, giao dịch cũng sẽ thất bại để tránh có số lượng ETH bằng 0 và (do đó) tỷ giá hoán đổi không xác định. Đảm bảo luôn để lại ít nhất 1 ETH và 1 token trong nhóm thanh khoản.
* function swapETHForTokens() external payable: Đổi số lượng ETH cho giá trị tương đương của token của bạn và cập nhật trạng thái của sàn giao dịch tương ứng. Tương tự như addLiquidity(), người gửi sẽ gửi ETH vào hợp đồng, có thể truy cập thông qua msg.value. Nếu việc thực hiện giao dịch này sẽ làm cho toàn bộ token trong nhóm thanh khoản bị rút hết, giao dịch cũng sẽ thất bại để tránh có số lượng token bằng 0 và (do đó) tỷ giá hoán đổi không xác định. Đảm bảo luôn để lại ít nhất 1 ETH và 1 token trong nhóm thanh khoản.

Trong mỗi hàm trên, hãy đảm bảo rằng bạn điều chỉnh dự trữ token, dự trữ ETH và/hoặc k theo cách chính xác để sàn giao dịch luôn nằm trên đường cong sản phẩm hằng số mô tả ở trên. Hơn nữa, hãy đảm bảo rằng các hàm thất bại khi người gọi không có đủ số dư. Cuối cùng, hãy nhớ đặt địa chỉ tokenAddr thành địa chỉ hợp đồng token bạn deploy. Bây giờ, bạn có thể chạy npx hardhat run --network localhost scripts/deploy exchange.js để gỡ lỗi và triển khai mã của bạn.

Xử lý lỗi làm tròn số. Việc hoán đổi token theo quy tắc sản phẩm hằng số liên quan đến việc thực hiện các phép tính như cộng và chia, và điều này tất nhiên dẫn đến các lỗi làm tròn số. Quan trọng là đảm bảo rằng các lỗi làm tròn số không thể tích lũy. Điều này có thể dẫn đến thiệt hại cho nhà cung cấp thanh khoản nếu số lượng token rút khỏi nhóm thanh khoản lớn hơn kỳ vọng ở mỗi lần hoán đổi, ví dụ như.

Do đó, có thể xảy ra sai số làm tròn khi thêm hoặc rút thanh khoản vì token và ETH của bạn không thể chia thành các đơn vị nhỏ hơn 1 đơn vị (trong dự án này ít nhất là 1 đơn vị, xem phần 8 để biết thêm chi tiết). Chúng tôi không mong đợi bạn xử lý các loại lỗi làm tròn này trong dự án này. Vì vậy, nếu x ∗ y không chính xác bằng k khi thực hiện các phép toán, cũng như các lỗi làm tròn tương tự khác khi tính toán thanh khoản, điều này không gây vấn đề. Tuy nhiên, vẫn cần cập nhật k tại các vị trí thích hợp.

5. Implementing the Backend

Sau khi hoàn thành việc triển khai các hàm trong smart contract, bạn cần triển khai các hàm sau trong tệp exchange.js. Bạn có thể bỏ qua biến maxSlippagePct trong lúc này - nó sẽ được sử dụng trong phần 6. Đa phần các hàm này chỉ gọi các hàm token và exchange mà bạn đã viết ở trên:

* Hàm async function addLiquidity(amountEth, maxSlippagePct): Thêm thanh khoản vào hệ thống bằng cách cung cấp một lượng Eth và tỷ lệ tối đa của sự biến động giá (slippage).
* Hàm async function removeLiquidity(amountEth, maxSlippagePct): Rút thanh khoản khỏi hệ thống bằng cách xác định số lượng Eth muốn rút và tỷ lệ tối đa của sự biến động giá (slippage).
* Hàm async function removeAllLiquidity(maxSlippagePct): Rút toàn bộ thanh khoản khỏi hệ thống với tỷ lệ tối đa của sự biến động giá (slippage).
* Hàm async function swapTokensForETH(amountToken, maxSlippagePct): Hoán đổi tiền tệ Token thành Eth với số lượng Token muốn hoán đổi và tỷ lệ tối đa của sự biến động giá (slippage).
* Hàm async function swapETHForTokens(amountEth, maxSlippagePct): Hoán đổi Eth thành tiền tệ Token với số lượng Eth muốn hoán đổi và tỷ lệ tối đa của sự biến động giá (slippage).

Bạn có thể gọi mã hợp đồng bằng cách sử dụng await contract.functionName(args) hoặc await contract.connect(anotherSigner).functionName(args). Để biết thêm thông tin về cú pháp, chúng tôi khuyến khích bạn tìm hiểu tài liệu ethers.js, cũng như phần 9 để có một số gợi ý.

Sau khi bạn đã triển khai hoàn chỉnh smart contract và mã JavaScript tương ứng, hãy cập nhật các biến token\_abi và exchange\_abi ở đầu tệp và sao chép các địa chỉ hợp đồng vào các biến token\_address và exchange\_address trong tệp exchange.js. Hãy chắc chắn bao gồm các dấu ngoặc vuông bên ngoài khi sao chép ABI. Nếu bạn tải lại index.html, bạn nên có thể cung cấp thanh khoản, rút thanh khoản và thực hiện các giao dịch hoán đổi.

6. Handling Slippage: xử lý trượt giá

Có một vấn đề quan trọng với hệ thống giao dịch của chúng ta như đã triển khai trong phần 3, đó là hệ thống không xem xét đến "slippage" (độ chênh lệch giá). Hãy nhớ rằng với mỗi lần hoán đổi trên một sàn giao dịch phi tập trung, giá của mỗi tài sản sẽ chuyển đổi một chút. Vì nhiều người dùng có thể cố gắng hoán đổi tiền tệ cùng một lúc trên một sàn giao dịch phi tập trung, có thể xảy ra một chuyển đổi tỷ giá giữa khi gửi giao dịch hoán đổi và khi giao dịch được xử lý thực sự. Chuyển đổi tỷ giá này giữa giá trị tham chiếu và giá trị thực tế được gọi là "slippage" (độ chênh lệch giá). Độ chênh lệch giá đặc biệt quan trọng khi giao dịch các tài sản biến động. Ví dụ, nếu một người dùng gửi một giao dịch hoán đổi một số lượng tiền tệ A sang tiền tệ B và sau đó giá của tiền tệ B tăng đột ngột so với giá trị tham chiếu, người dùng có thể không muốn hoàn thành giao dịch hoán đổi đó.

Ngoài ra, không xử lý đúng sẽ khiến người dùng trở thành mục tiêu của một loại tấn công được biết đến là tấn công sandwich. Một tấn công sandwich hoạt động như sau:

1. Alice gửi một giao dịch hoán đổi để chuyển đổi một số lượng lớn tiền tệ A thành tiền tệ B.
2. Một kẻ thù nhìn thấy giao dịch của Alice và đặt mua một số lượng rất lớn tiền tệ B trước, từ đó đẩy giá của tài sản B lên.
3. Alice mua tiền tệ B với giá cao mới, gây tăng giá tiền tệ B hơn nữa.
4. Kẻ tấn công sau đó ngay lập tức bán hết tiền tệ B vừa mua được với giá cao, thu được lợi nhuận nhanh chóng.

Sự tồn tại của lỗ hổng về tấn công sandwich là không tốt cho người dùng của một sàn giao dịch phi tập trung, vì người dùng liên tục phải trả mức giá hơn so với giá trị thực của tài sản. Do đó, quan trọng là chúng ta nâng cấp hệ thống giao dịch của mình để xử lý đúng slippage và phòng ngừa tấn công sandwich.

Biện pháp phổ biến nhất để phòng ngừa tấn công sandwich là cho phép người dùng đặt một độ chênh lệch giá tối đa khi gửi giao dịch. Tham số này thường là một phần trăm, và sẽ làm cho giao dịch thất bại nếu giá của tài sản đã thay đổi hơn so với độ chênh lệch giá tối đa được cho phép. Điều này giới hạn tổn hại có thể xảy ra do tấn công sandwich và bảo vệ người dùng khi giao dịch các tài sản biến động.

Để thực hiện yêu cầu về độ chênh lệch giá tối đa, thực hiện các bước sau:

1. Trong tệp exchange.sol, cập nhật các hàm swapTokensForETH và swapETHForTokens để nhận một tham số uint max\_exchange\_rate. Bạn cũng có thể truyền vào các tham số khác nếu cần cho thiết kế của bạn. Trong quá trình hoán đổi, giao dịch sẽ thất bại nếu giá hiện tại của tài sản mới (tức là tài sản mà người dùng đang hoán đổi đến) tăng lên hơn mức tỷ giá hoán đổi tối đa.
2. Cập nhật các hàm addLiquidity, removeLiquidity và removeAllLiquidity để nhận tham số uint max\_exchange\_rate và uint min\_exchange\_rate. Bạn cũng có thể truyền vào các tham số khác nếu cần cho thiết kế của bạn. Trong quá trình cung cấp thanh khoản, giao dịch sẽ thất bại nếu giá hiện tại của tài sản mới (tức là tài sản mà người dùng đang hoán đổi đến) tăng lên hơn mức tỷ giá hoán đổi tối đa hoặc giảm xuống dưới mức tỷ giá hoán đổi tối thiểu. Sự thay đổi giá đột ngột theo cả hai hướng có thể khiến các nhà cung cấp bị mất lợi nhuận tạm thời trước khi họ gửi thanh khoản của mình, do đó chúng ta muốn nhà cung cấp thanh khoản xác định mức tỷ giá hoán đổi tối đa và tối thiểu.
3. Cập nhật tệp exchange.js để giao tiếp với hợp đồng về các mức tỷ giá hoán đổi tối đa/tối thiểu. Tham số maxSlippagePct được cung cấp, đại diện cho phần trăm tối đa cho phép thay đổi giá trị trước khi giao dịch thất bại. Trong quá trình kiểm tra và trong giao diện trình duyệt, tham số này được truyền dưới dạng số nguyên, không phải số thập phân - ví dụ: 4% được truyền dưới dạng 4, không phải 0.04. Tham số này có thể được sử dụng trong mỗi hàm JavaScript để tính toán các giá trị chính xác cho tỷ giá hoán đổi tối đa và/hoặc tỷ giá hoán đổi tối thiểu, sau đó chuyển cho hợp đồng. Hàm getPoolState mà chúng tôi cung cấp có thể hữu ích ở đây.

Như thường lệ, sau khi cập nhật hợp đồng, hãy chắc chắn biên dịch lại, triển khai lại và sao chép ABI và địa chỉ hợp đồng mới vào biến ở đầu tệp exchange.js của bạn. Tại điểm này, bạn cũng có thể bỏ chú thích (uncomment) cho hàm sanityCheck() để kiểm tra việc triển khai của bạn. Xem phần 10 để biết thêm chi tiết về sanityCheck.

7. Rewarding Liquidity Providers

Sau khi hoàn thành các phần trên, bạn đã có một sàn giao dịch hoạt động cho phép người dùng giới hạn mức độ chênh lệch giá mà họ muốn chịu đựng! Tuy nhiên, vẫn còn một vấn đề lớn nữa. Chúng ta đã thảo luận nhiều lần về việc nhà cung cấp thanh khoản đang đảm nhận rủi ro dưới hình thức mất mát tạm thời (impermanent loss). Điều này có nghĩa là giá trị của sự tham gia thanh khoản của họ có thể giảm nếu giá của bất kỳ tài sản nào thay đổi. Trong thực tế, vì nhiều loại tiền điện tử có tính biến động cao, đây là một mức độ rủi ro mà không có nhà cung cấp thanh khoản nào sẵn lòng chịu đồng ý miễn phí.

Do đó, chúng ta cần khuyến khích nhà cung cấp thanh khoản cung cấp thanh khoản cho nhóm thanh khoản. Trong các sàn giao dịch thực tế, nhà cung cấp thanh khoản được khuyến khích cung cấp thanh khoản bởi vì họ nhận được một khoản phí nhỏ từ mỗi giao dịch hoán đổi. Những khoản phí này được tự động đầu tư lại vào nhóm thanh khoản thay mặt cho mỗi nhà cung cấp thanh khoản. Khi một nhà cung cấp muốn rút thanh khoản của mình, số tiền mà họ được rút bao gồm tất cả các khoản phí mà họ đã nhận được kể từ khi cung cấp thanh khoản.

Bạn sẽ thực hiện cùng một hệ thống thưởng phí cho nhà cung cấp thanh khoản. Bạn có thể tự thiết kế theo yêu cầu được nêu cuối phần này. Tuy nhiên, chúng tôi mạnh dạn đề xuất theo cách sau, được giải thích một cách đơn giản:

* Mỗi nhà cung cấp thanh khoản sở hữu một phần tử f của nhóm thanh khoản. Phần tử này được lưu trữ trong hợp đồng thông minh cho mỗi nhà cung cấp thanh khoản (ví dụ: một mảng lưu trữ cho mỗi địa chỉ tỷ lệ phần tử của nhóm thanh khoản thuộc sở hữu của địa chỉ đó). Trong nhiệm vụ này, chúng ta sẽ đặt mức phí hoán đổi là 3% để hỗ trợ công cụ tự động kiểm tra (autograder), như được đại diện bằng các trường swap\_fee\_numerator và swap\_fee\_denominator. Tỷ lệ sở hữu của nhà cung cấp thanh khoản không thay đổi khi các hoán đổi được thực hiện, vì phần thưởng thanh khoản được phân phối dựa trên tỷ lệ sở hữu của mỗi nhà cung cấp thanh khoản. Lưu ý rằng do sự tồn tại của các khoản phí, tổng thanh khoản l = √xy tăng lên, và do đó k sẽ có chút thay đổi. Điều này dẫn đến lợi nhuận cho nhà cung cấp thanh khoản nếu tỷ giá hoán đổi không thay đổi, và đồng thời giúp chống lại mất lợi tạm thời (impermanent loss).
* Như đã đề cập ở trên, do sự thưởng thanh khoản, k sẽ thay đổi nhỏ mỗi khi có hoán đổi. Tuy nhiên, trong dự án này, bạn chỉ cần thay đổi k khi thêm hoặc rút thanh khoản. Vì vậy, bạn không cần quan tâm đến những thay đổi nhỏ trong k do sự thay đổi trong x \* y trong quá trình hoán đổi.
* Khi một nhà cung cấp thanh khoản rút thanh khoản của mình, họ sẽ nhận được một tỷ lệ f của cả hai token trong nhóm thanh khoản tương ứng với tỷ lệ sở hữu của họ, cũng như các khoản thưởng của họ. Các nhà cung cấp thanh khoản khác sẽ có tỷ lệ sở hữu tăng lên tương ứng. Tất cả các tỷ lệ phải cộng lại thành 1.
* Khi một nhà cung cấp thanh khoản thêm thanh khoản vào nhóm, họ sẽ có một tỷ lệ sở hữu f trên nhóm thanh khoản tương ứng với tỷ lệ của token của họ trong trạng thái mới của nhóm thanh khoản. Các nhà cung cấp thanh khoản khác sẽ có tỷ lệ sở hữu thu hẹp tương ứng. Một lần nữa, tất cả các tỷ lệ phải cộng lại thành 1.

Nếu bạn đã triển khai theo dõi thanh khoản bằng cách sử dụng tỷ lệ, thì phần này không nên đòi hỏi quá nhiều công việc bổ sung. Ngoài ra, chúng tôi cũng chấp nhận bất kỳ thiết kế nào đáp ứng các yêu cầu được liệt kê dưới đây.

*Yêu cầu phần thanh toán:*

1. Nhóm thanh khoản của bạn phải thu phí một tỷ lệ nonzero percent cho mỗi giao dịch hoán đổi. Bạn có thể xác định giá trị này bằng cách sử dụng tỷ lệ phần tử hoán đổi và mẫu số tỷ lệ phần tử hoán đổi, như đã được giải thích ở trên.
2. Khi có một giao dịch hoán đổi, giá trị của token hoặc ETH được gửi cho người thực hiện giao dịch phải bằng (1 - p) lần giá trị của tài sản mà họ đang hoán đổi, trong đó p là tỷ lệ phần trăm phí được thu của người cung cấp thanh khoản. Ví dụ, nếu phí là 1% và người dùng đang hoán đổi 100 ETH cho token của bạn, họ chỉ nhận được tương đương 99 ETH.
3. Khi thu phí trong một giao dịch hoán đổi, phí đó phải được phân phối cho các nhà cung cấp thanh khoản sao cho mỗi nhà cung cấp sau này có thể rút lại số vốn ban đầu của họ cộng với phí. Phí phải được phân phối theo tỷ lệ dựa trên tỷ lệ phân đoạn của các nhà cung cấp thanh khoản trong nhóm thanh khoản tại thời điểm giao dịch hoán đổi diễn ra. Ví dụ, nếu một nhà cung cấp thanh khoản cung cấp một nửa tổng thanh khoản trong nhóm thanh khoản tại thời điểm t, thì không chính xác nếu cho phép nhà cung cấp rút lại một nửa phí đã thu trước thời điểm t. Các nhà cung cấp thanh khoản không cần thực hiện bất kỳ bước bổ sung nào để nhận phí của mình ngoài việc gọi hàm removeLiquidity. Ngoài ra, phần thưởng thanh khoản không nên được gửi ra khỏi sàn giao dịch cho các nhà cung cấp sau mỗi lần hoán đổi diễn ra, vì điều này sẽ quá đắt đỏ trong thực tế.
4. Khi lựa chọn giữa các tùy chọn thiết kế khác nhau, chúng tôi khuyến khích bạn chọn giải pháp giảm thiểu chi phí gas. Chúng tôi sẽ không đánh giá nghiêm ngặt về việc sử dụng gas; tuy nhiên, bạn sẽ phải giải thích những quyết định thiết kế của mình trong tài liệu thiết kế ở Phần 6.

Sau khi thiết kế và triển khai phần trên, bạn sẽ có một sàn giao dịch hoạt động hoàn chỉnh! Xin chúc mừng! Hãy kiểm tra các chức năng của bạn bằng cách sử dụng giao diện UI được cung cấp trong index.html, hoặc viết mã kiểm thử bằng JavaScript. Việc triển khai dự án này đại diện cho một thành tựu rất đáng nể, vì vậy hãy tự thưởng cho mình. Thực tế, với một số điều chỉnh bảo mật, bạn có thể triển khai cả token của mình và sàn giao dịch lên Ethereum mainnet, và từ đó có một sàn giao dịch thuộc sở hữu riêng của bạn!

8. Note on Solidity and Javascirpt Decimals

Tại sao token lưu trữ 1 biến decimals?

Khác với hầu hết các ngôn ngữ lập trình khác, Solidity không hỗ trợ phép tính số thực. Do đó, tất cả các token ERC-20 sẽ lưu trữ một biến decimals, chỉ ra số thập phân phía trước mà số liệu nên được hiểu là. Ví dụ, ether sử dụng 18 chữ số thập phân, vì vậy 1 ETH sẽ được biểu diễn dưới dạng 10^18 trong hợp đồng. Tương tự, 1 wei = 10^-18 ETH, vì vậy 1 wei được biểu diễn chỉ là 1. Thật không may, Javascript cũng có giới hạn về sự lớn của các số nguyên: Numbers.MAX\_INT = 9 \* 10^15. Để cân bằng giữa hai yếu tố này, trong giao dịch của chúng tôi, chúng tôi khởi tạo nhóm thanh khoản có 10^10 token, có nghĩa là nhóm thanh khoản bắt đầu với 10^-8 ETH và 10^-8 của token của bạn.

Hệ quả của việc Solidity không hỗ trợ phép tính số thực là tất cả các số thập phân sẽ bị làm tròn. Ví dụ, 5/2 = 2. Đây là lý do tại sao biến decimals là cần thiết, vì biểu diễn 5 ETH dưới dạng 5 \* 10^18 sẽ dẫn đến 5 \* 10^18 / 2 = 25 \* 10^16, và vì chúng ta biết có 18 chữ số thập phân, ta có thể thấy rằng điều này tương ứng với 2.5 ETH, như mong muốn. Tuy nhiên, trong một số trường hợp, có thể xảy ra lỗi tràn số nhỏ hơn, đặc biệt nếu số tử số nhỏ hơn số mẫu số trong các phép tính, dẫn đến kết quả bằng 0. Trong trường hợp này, bạn cần phải cẩn thận với thứ tự các phép tính, chẳng hạn như nhân hoặc cộng trước khi chia. Nói chung, việc trì hoãn phép chia cho đến cuối cùng là một ý tưởng tốt để tránh gặp lỗi làm tròn này.

Trong dự án này, chúng tôi sẽ không kiểm tra vấn đề tràn số và chúng tôi đã cố gắng tạo trừu tượng cho tất cả các số thập phân. Do đó, khi kiểm tra giao dịch của bạn, chú ng tôi sẽ chọn các giá trị không làm cho giao dịch của bạn vượt quá giới hạn INT MAX của Javascript. Tuy nhiên, chúng tôi sẽ kiểm tra triển khai của bạn để đảm bảo rằng bạn đã tính đến trường hợp tràn số nhỏ trong giao dịch của mình.

9. Implementation Advice

Trong khi thiết kế tổng thể của hợp đồng của bạn là linh hoạt, dưới đây là một số lời khuyên có thể giúp tối ưu quá trình triển khai của bạn:

1. Theo dõi tỷ lệ của người cung cấp thanh khoản thay vì giá trị tuyệt đối. Ví dụ, nếu có 1000 ETH và 1000 token trong nhóm thanh khoản và Alice cung cấp 500 ETH và 500 token, thì Alice sở hữu 1/3 tổng số. Thay vì lưu trữ 500 ETH cho thanh khoản của cô ấy, chúng tôi khuyên bạn nên lưu trữ rằng cô ấy sở hữu 1/3. Trong quá trình hoán đổi, giá trị này không thay đổi (kể cả khi thực hiện phần thưởng lp). Tuy nhiên, khi người cung cấp thanh khoản thêm hoặc xóa thanh khoản, giá trị này (và tỷ lệ phần còn lại của các nhà cung cấp thanh khoản khác) cần được cập nhật tương ứng. Ngoài ra, vì Solidity không hỗ trợ số thập phân dấu phẩy động, chúng tôi khuyên bạn nên chọn một mẫu số (ví dụ: 100, 1000, vv.) để lưu trữ tử số của phần trăm dưới dạng uint.
2. Xử lý lỗi làm tròn biên. Tương tự, vì Solidity không hỗ trợ số thập phân dấu phẩy động, chúng tôi khuyên bạn nên thực hiện phép nhân trước khi chia mỗi khi có thể. Điều này tránh gặp phải lỗi làm tròn chia thành 0 và sau đó nhân lại để được 0. Tương tự, với kiểu uint, hãy chắc chắn thực hiện phép cộng trước khi trừ khi có thể để tránh hiện tượng tràn số.
3. Xác nhận chuyển giao token. Để một địa chỉ bên thứ ba (ví dụ: hợp đồng) có thể gửi hoặc nhận token của bạn thay mặt bạn, bạn phải trước tiên cấp quyền cho họ bằng cách sử dụng hàm approve() của hợp đồng token. Hàm này cần được khởi tạo bởi người dùng, bạn sẽ không thể chạy approve() từ chính hợp đồng. Vì vậy, hãy chắc chắn gọi hàm này trong mã JavaScript trước khi gọi hàm trao đổi khi cần thiết. Chi tiết về hàm này có thể được tìm thấy trong phiên bản ERC20 của OpenZeppelin.
4. Gửi và nhận ETH từ Hợp đồng. Để chuyển ETH từ tài khoản người dùng đến hợp đồng thành công, hàm xử lý việc chuyển giao phải được đánh dấu là payable. Điều quan trọng cần lưu ý là chỉ xác định một đối số trong hàm hợp đồng để chỉ định số lượng ETH sẽ không chuyển ETH; thay vào đó, ETH được chuyển qua tham số msg.value. Để tránh số thập phân, hãy sử dụng ethers.utils.parseUnits() với đơn vị được thiết lập là "wei" khi chuyển ETH vào hợp đồng. Tương tự, nếu bạn muốn gửi ETH từ hợp đồng đến địa chỉ người dùng, bạn có thể sử dụng hàm payable() một cách thích hợp. Thêm thông tin chi tiết có thể được tìm thấy ở đây.
5. Lặp qua các khóa của mapping. Solidity không hỗ trợ lặp qua các khóa của mapping. Vì vậy, chúng tôi đã định nghĩa address[] private lpProviders để lưu trữ địa chỉ của các nhà cung cấp thanh khoản. Ngoài ra, hãy nhớ rằng mảng Solidity không tự động "di chuyển" tất cả các phần tử khi giá trị ở giữa mảng bị xóa. Do đó, chúng tôi cung cấp một hàm trợ giúp removeLP() để xóa một nhà cung cấp thanh khoản khỏi mảng và điền vào "khe hở". Điều này sẽ cập nhật lpProviders.length tương ứng. Tuy nhiên, hãy cẩn thận khi gọi hàm này trong quá trình lặp qua mảng, vì rủi ro thay đổi độ dài của mảng trong quá trình lặp qua mảng nếu bạn muốn truy cập tất cả các phần tử.
6. Truy cập địa chỉ hợp đồng. Bạn có thể lấy địa chỉ hợp đồng trong Solidity bằng cách gọi address(this).
7. Truy cập số lượng token và ETH của hợp đồng. Trong khi bạn có thể tính toán số lượng token và ETH của hợp đồng bằng tay trong quá trình tính toán trung gian, bạn có thể truy cập các số dư thực sự bằng cách sử dụng các hàm sau:
   1. Số dư ETH: address(this).balance
   2. Số dư Token: token.balanceOf(address(this))

Chúng tôi đề nghị cao sử dụng các giá trị thực sự này để tính toán k thay vì sử dụng các dự trữ token và dự trữ ETH và hãy kiểm tra lại công việc của bạn.

1. Hàm Javascript Async. Trong Javascript, có nhiều lần khi các hàm được chạy bất đồng bộ. Điều đó có nghĩa là nếu mã của bạn gọi một hàm bất đồng bộ, thì mã sẽ tiếp tục chạy qua dòng đó mà không đợi hàm bất đồng bộ đó hoàn thành thực thi. Mặc định, các hàm async trả về một đối tượng Promise, cho biết một giá trị sẽ được trả về sau khi thực thi hàm kết thúc. Để mã của bạn đợi hàm bất đồng bộ thực thi và lấy kết quả, bạn cần sử dụng từ khóa await. Bạn có thể tìm hiểu thêm về hàm async trong Javascript tại đây. Quan trọng để lưu ý rằng tất cả các cuộc gọi hàm Solidity sẽ là bất đồng bộ từ phía backend. Do đó, trong exchange.js, để gọi một hàm từ hợp đồng và nhận kết quả, hãy chắc chắn sử dụng từ khóa await, ví dụ: var num = await tokenContract.function(args).
2. Bảo mật của Hợp đồng. Hãy nhớ rằng vì tất cả các hợp đồng triển khai đều công khai trên chuỗi, việc giữ hợp đồng của bạn an toàn là rất quan trọng. Mặc dù chúng tôi sẽ không kiểm tra hợp đồng của bạn một cách chủ động về bảo mật, nhưng chúng tôi vẫn mong muốn thấy một số biện pháp phòng thủ thông qua việc sử dụng các câu lệnh require() hoặc assert().

10. Sanity Check

Để kiểm tra triển khai của bạn, chúng tôi đã thực hiện hai chương trình kiểm tra hợp lý (sanity check) chạy dựa trên việc bạn đã triển khai phần thưởng thanh khoản (liquidity rewards) hay chưa. Chúng tôi kiểm tra giá trị này bằng cách đọc giá trị tử số phí hoán đổi (swap fee numerator) trong file exchange.sol: nếu giá trị đó là 0, thì chúng tôi giả định rằng bạn chưa triển khai phí hoán đổi và phần thưởng thanh khoản. Bạn sẽ không nhận được điểm đầy đủ cho đến khi phần thưởng thanh khoản được triển khai hoàn chỉnh, nhưng chúng tôi cũng muốn đánh giá công việc hoàn thành cơ bản của dự án này.

Để bật chế độ kiểm tra hợp lý (sanity check), hãy gỡ bỏ dấu chú thích (uncomment) cho hàm setTimeout() với sanityCheck() và làm mới trang. Chúng tôi đã cố gắng thiết kế sanityCheck để chạy đúng ngay cả sau lần tải đầu tiên, vì vậy bạn không cần triển khai lại các hợp đồng và thiết lập lại trạng thái nhóm thanh khoản (pool state) mỗi khi bạn muốn chạy sanityCheck. Tuy nhiên, do lỗi làm tròn, có thể có một điểm mà sanityCheck vượt qua khi tỷ giá hoán đổi là 1:1 nhưng không phải là trạng thái hoán đổi hiện tại của bạn. Điểm sanityCheck của bạn cho dự án này sẽ dựa trên trạng thái ban đầu của nhóm thanh khoản - tức là tỷ giá hoán đổi giữa các token là 1:1 và có 5000 ETH và 5000 token trong nhóm thanh khoản.

11. Design Document

Hãy điền thông tin vào tệp DesignDoc.txt với câu trả lời cho các câu hỏi sau:

1. Giải thích tại sao việc thêm và xóa thanh khoản vào sàn giao dịch của bạn không thay đổi tỷ giá trao đổi.
2. Giải thích kế hoạch của bạn để thưởng cho người cung cấp thanh khoản và giải thích các quyết định thiết kế bạn đã đưa ra. Lý do nào đã giúp đáp ứng yêu cầu về thưởng thanh khoản được mô tả trong Mục 7?
3. Mô tả ít nhất một phương pháp bạn đã sử dụng để giảm tiêu thụ gas của hợp đồng giao dịch. Tại sao phương pháp này hiệu quả?
4. Phản hồi tùy chọn:
   1. Bạn đã dành bao nhiêu thời gian cho nhiệm vụ này?
   2. Một điều gì đó mà có thể hữu ích khi bạn bắt đầu nhiệm vụ này?
   3. Nếu bạn có thể thay đổi một điều về nhiệm vụ này, bạn sẽ thay đổi điều gì?
   4. Hãy gửi thêm bất kỳ phản hồi nào khác mà bạn có thể có.

Ví dụ

\* Nhà cung cấp thanh khoản:

- Nam: 6000 ETH và 600.000 DHN (60%)

- Duy: 4000 ETH và 400.000 DHN (40%)

Phí thưởng: 1%

\* Tỷ lệ hoán đổi trước giao dịch

x = 10.000 ETH

y = 1.000.000 DHN

x\*y = 10.000 \* 1.000.000 = k

==> Tỷ lệ hoán đổi: 1 ETH = 100 DHN hay 1 DHN = 0.01 ETH

\* Người thực hiện hoán đổi

- Hải: 100 ETH ---> 10.000 DHN

Chịu phí: 1% ---> 1ETH

Mất: 100 ETH

Nhận: 9.900 DHN

\* Nhà cung cấp thanh khoản:

1 ETH phí thưởng:

- Nam: Nhận 0.6 ETH

- Duy: Nhận 0.4 ETH

\* Tỷ lệ hoán đổi sau giao dịch

x = 10.100 ETH

y = 990.000 DHN

x\*y = 10.100 \* 990.000 = k

==> Tỷ lệ hoán đổi: 101 ETH = 9900 DHN hay 1 DHN = 0.0102 ETH