출처: http://earlz.net/view/2017/10/02/0801/thoughts-and-goals-on-qtums-x86-vm

Qtum x86 VM 에 대한 생각과 목표

그 동안 우리는 더 많은 프로그래밍 언어 지원 말고는 Qtum 의 x86 VM 에 대해서 말을 아꼈습니다. 그 이유는 기본적으로 디자인 프로세스에 대해서 범용적인 버전을 만드는 것은 쉽지만 최적화가 잘 되어 있고 효율적이며 사용하기 쉬운 버전을 만드는 것은 쉽지 않기 때문입니다. 그래서 이 글에는 디자인의 세부사항 까지는 아니지만, 우리가 염두해 두고 있는 목표에 대해 말하고 싶습니다.

프로그래밍지원언어

프로그래밍 언어 지원의 가장 큰이유는 x86 VM 을 구축하기 위해서입니다. 저는 개인적으로 2018 년을 Rust 로 작성된 스마트 계약의 해라고 쓰고 싶습니다. Rust 는 믿을 수 없을 정도로 효율적이고 경량이며 무엇보다도 프로그래머의 실수를 피하고 안정성을 중요시 합니다. Rust 보다 좋은 언어들은 훨씬 더 많이 있습니다. C# 혹은 Go 와 같이 사용하기 쉬운 언어를 가져 오는 것 역시 목표입니다.

x86 VM 의 힘의 기본 핵심은 기존의 컴파일러나 거의 모든 프로그래밍 언어를 다 사용 할 수 있다는 것입니다. 그래서 Qtum 운영체제와 같은 환경에서 실행하기 위해서, 일부 수정만 하면 된다는 사실입니다. 대부분의 컴파일러가 x86 을 지원하고 있으며 bytecode(자바 프로그램의 컴파인된 형태)나 아카텍쳐 지원도 이미 존재 합니다.

표준라이브러리

EVM(Ethereum Virtual Machine)의 일반적인 불만 사항은 표준 라이브러리가 없다는 점입니다. 이것은 개발자를 번거롭게 할뿐만 아니라 귀중한 블록체인 공간을 직접 소모한다는 점입니다.

표준 라이브러리를 제공한다면 Qtum 블록체인을 보다 경량화하고 효율적으로 만들 수 있을 뿐만 아니라 이더리움의 pre-compiled 계약과 마찬가지로 표준 라이브러리 함수에 특수 내부 코드를 가질 수 있습니다.

이 기능은 새로운 pre-compiled 계약에 대한 특별한 지원을 추가하지 않고도 사용 할 수 있고, 이 특수 기능을 사용하는 계약에 의존합니다. 대신에 계약서는 예전의 최적화되지 않은 코드를 사용 할 수 있으며, opaque code(외부 인터페이스로 노출 되지 않은 코드)를 호출 할 때 이 코드는 아무런 변경을 하지 않고서도 자유롭게 최적화 할 수 있습니다. 이와 같은 표준 라이브러리의 최적화는 세부적으로 구현 됩니다. 생태계의 구현이 보다 효율적으로 이루어짐에 따라 가스 모델은 가스 비용이 실제 자원 비용을 반영하도록 하여 이런 기능을 조정할 수 있습니다.

게다가 표준 라이브러리를 확장하기 위해서 별도의 포크(fork)가 필요하지 않습니다. 보통의 함수는 분산형 관리방식 프로토콜(Decentralized Governance Protocol, DGP)을 사용하기 때문에 전문화된 표준 라이브러리 메모리 공간에 쉽게 입력이 가능합니다.

이 메커니즘을 사용하면 표준 라이브러리의 문제점을 보완할 수 있겠지만, 스마트 계약의 경우 올바른 기능을 수행하는 과정에서 문제점이 있는 동작에 영향을 받을 수 있으므로 특별한 감시가 필요합니다. 따라서 표준 라이브러리 기능에 대한 잠재적인 업그레이드 작업은 사전 동의 기능에 의해서만 가능할 수도 있고, 전혀 존재하지 않을 수도 있습니다. 우리의 목표는 분산형 관리방식 프로토콜(DGP)에 대해 항상 보수적인 자세를 유지하는 것입니다. 가령 완전한 타협이 있는 경우에도 스마트 계약 로직이 영향 받지 않고 사용자의 자금을 안전하게 보장하는 것입니다.

최적화 된 가스모델

이번 파트는 조금은 까다로운 부분 입니다. 우리는 초기의 EVM(Ethereum Virtual Machine)과 비슷한 간단한 가스 모델로 x86 VM 의 설명을 시작 할 것입니다. 그러나 x86 에서 사용되는 ISA(Instruction Set Architecture)와 기능들은 너무나도 강력하기 때문에 이 공간을 발전시키기에 훨씬 간단한 방법이 있습니다.

간단하면서도 강력한 해결책 중 하나는 스마트 계약 및 전반적인 프로그램에서 사용되는 공통 기능을 갖춘 표준 라이브러리를 제공하는 것입니다. 이 뿐만 아니라 이러한 표준 라이브러리 기능이 일반적인 계산에 사용되는 단순하고 일반적 가스 모델에 의존하는 것이 아닌 사람이 설정한 비용으로 만들게 하는 간단한 방법이 있습니다. 예를 들어 단순한 가스 모델에서 `strawen` 은 문자열에서 문자 하나당 90 개의 가스가 필요할 수 있습니다. 그러나 개발자가 검사할 때 `strawen` 은 Qtum VM 에서 매우 저렴하게 실행이 가능합니다. Qtum 의 DGP 는 이 함수에 대해서 특별한 가스 규칙을 사용합니다. 이제 이 함수를 호출하는데 드는 비용은 10 개의 가스에 대한 초기 비용과 문자 하나당 1 개의 가스를 합친 것입니다. 완벽하고 정확한 가스 모델을 만드는 것은 불가능하기 때문에, Qtum 에서는 DGP 메커니즘을 활용하여 최적화되면서도 효율적인 근사 방식을 만들기를 원합니다.

AAL(Account Abstraction Layer, 계정 추상화 계층)의 모든 권한 잠금 해제

계정 추상화 계층(Account Abstraction Layer, AAL)은 "EVM(Ethereum Virtual Machine)작업을 수행하기에 필요한 것" 입니다. 그러나 AAL 은 Qtum 에서 EVM 을 작동시키는데 필요한 것 이상으로 많은 기능을 제공합니다. Qtum 은 처음부터 다양한 가상 머신을 지원하도록 설계 되어 있으며, EVM 은 지원하는 가상 머신 중 첫 번째 입니다. 이 말은 AAL 이 현재 EVM 을 통해 쉽게 노출 될 수 있기 때문에 제한되고 있습니다. 우리가 설계하고 있는 x86 VM 은 이런 한계에 직면하지 않을 것입니다. 아래는 우리가 공개하고 싶은 강력한 기능들입니다.

- P2SH(Pay to Script Hash, Bacterin 스타일) 스마트 계약을 통한 송금 및 수취를 위한 일급개체로써의 다중 서명(Multi-Signature)

l Qtum 스크립트 기능을 최대로 활용하기 위해 사용자 정보 거래내역을 보내는 원시 거래내역 스크립트(Raw transaction script) 지원

l Signist(증인 분리, Segregated Witness) 거래내역에 스마트 계약 포함 및 실행 허용

스마트계약의 새로운 가능성

x86 을 사용하게 되면 폰 노이만(Von Neumann) 컴퓨팅 아키텍처를 사용할 수 있습니다. 이것이 의미하는 바는 코드는 데이터이고 그 반대도 마찬가지라는 개념입니다. 이런 특징은 하드웨어/소프트웨어 인터럽트와 같은 기능뿐 아니라, 구조 및 기능과 같은 잠재적인 운영 체제가 여러 개의 준 신뢰 관계가 있는 참가자가 단일 스마트 계약에 통합 되도록 합니다. 여기에는 협력 가능한 멀티 태스킹(Cooperative-multitasking), 일시 중지 및 다시 시작을 실행(다시 말하면 이후 거래내역에서 다시 시작) 그리고 워치독 타이머(Watchdog timer, "시간"대신 가스로 작동) 등이 포함됩니다. 물론 이런 개념은 새로운 계약에 자금과 데이터를 전송할 필요 없이 계약 bytecode 를 업데이트 하는 직접적인 메커니즘을 포함합니다.

x86 명령어 세트는 시스템 호출뿐 아니라 특정 코드 공간, 페이징 및 메모리 매핑에 대한 특수 권한과 같은 것을 제어하는 많은 특수 기능을 포함합니다. Qtum 에서 이러한 특수 시스템 수준의 지침을 대부분 공개하지 않을 것으로 예상합니다. 이런 명령어 세트는 가스 모델 디자인을 크고 복잡하게 만들고, 모든 것을 최적화하기 더 어렵게 만듭니다.

그러나 이런 점이 있음에도 불구하고, 해당 지침 내에서 스마트 계약과 관련된 것은 상대적으로 거의 없습니다. 하나의 스마트 계약 코드 내에 별도의 권한이 부여된 ring-0 코드(x86 kernel 수준 보호 링)와 권한이 부여되지 않은 ring-3(어플리케이션 수준 보호링) 코드를 분리해서 공개 블록체인에 적용할 필요성이 거의 없습니다. 이런 기능의 사용 케이스는 일반적으로 권한이 부여되거나 준권한이 부여된 블록체인에 있습니다. 그래서 Qtum 의 엔터프라이즈 측면에서 초점을 맞추기 시작하면서 다시 검토 할 것입니다.

일급 오라클 데이터베이스

거래내역을 위한 x86 VM 모델에서 계약에 필요한 데이터를 알고 있는 경우 계약을 호출 할 필요가 없습니다. 외부 계약의 저장 공간에서 직접 다운로드를 할 수 있습니다. 이는 계약이 자신의 ABI(Application Binary Interface)및 API(Application Programming Interface) 메커니즘을 구축하여 저장 공간을 표준화 할 수 있는 일급 오라클 데이터베이스라는 것을 뜻합니다. 그리하여, 계약서 bytecode 전체를 불러 올 필요없이 저장 데이터를 직접 불러올 수 있습니다. 이것은 스마트 계약의 기능에 의해 제한되는 대신 오라클 데이터베이스를 블록체인에서 일급객체로 만들 것입니다.

블록체인 분석

x86 에서 사용할 수 있는 많은 메모리 공간과 일반적인 연산을 위한 효율적인 연산 코드 세트는 꿈같은 블록체인 분석의 잠재력을 실현 가능하게 합니다. 계약 분석을 위해 전체 블록체인

데이터를 드러내는 것이 가능합니다. 이를 통해 AI 기반 스마트 계약을 통해 스마트 계약이 현재 네트워크 상태에서 가능한 효율적으로 작동하도록 자신의 동작을 조정할 수 있고 오라클 데이터베이스 역할을 할 수 있는 블록체인을 자동으로 모니터링 할 수 있습니다. 이 블록체인 데이터는 전체 거래 데이터 또는 블록체인 노드(합의-결정 방식)에 의해 계산 된 통계를 포함 할 수 있습니다. 이 데이터는 완전히 상수이며 추가 메모리 사용량이 수 메가 바이트 밖에 되지 않기 때문에 데이터를 노출하는데 불리한 점이 있습니다.

대체 데이터 저장소

현재 EVM 은 32 바이트 데이터를 가리키는 데에 32 바이트 키를 사용하도록 강제하고 있습니다. 이는 공간의 세분화와 유지 관리를 필요로 할 때 관리하기 상당히 어려울 수 있습니다. 게다가 이것을 사용해야 하는 특별한 이유도 없습니다. 따라서 x86 시스템에서 우리는 스마트 계약에 일반 목적의 키-값 저장소를 제공하려 합니다. 그러므로 1 에서 몇바이트까지 바이트 수를 키값으로 저장하고, 똑같이 다른 값을 가리키게 할 수 있습니다. 지금까지 이 기능에 대해 제안된 가스 모델은 기본적으로 데이터베이스에 쓰기/읽기에 대한 정액 요금을 지불하고 처리되는 바이트에 대한 요금을 포함합니다. 이 기능은 물론 stateRootHash 아래에 적용되므로 SPV(Simple Payment Verification) 지갑이 이런 데이터베이스를 사용하여 스마트 계약과 상호 작용 할 수 있습니다.

명시적 종속계층

다소 이상적인 다음 목표는 명시적으로 선언되고 실행되는 스마트 계약의 종속성 계층을 허용하는 것입니다. 이것은 알 수 없는 스마트 계약에 관해서 여전히 가능하도록 하는 사전 동의 기능일 뿐입니다. 그러나 의존성을 정확히 알고 있는 계약의 경우 계약에 따라 일부 계약이 병행 될 수 있으므로 가스 비용이 절감되고 다른 혜택도 얻을 수 있습니다. 이 기능을 선택하는 x86 기반 스마트 계약의 경우 확장성의 큰 이점이 됩니다.

왜 x86 인가? ARM 이 안 되는 이유가 무엇인가?

나는 많은 사람들에게 "왜 x86 인가? ARM 이 안 되는 이유는 무엇인가" 라는 질문을 많이 받았습니다. 이는 매우 좋은 질문입니다. 우리는 x86 이 가상 시스템과 에뮬레이터에 대해 가장 잘 이해하고 있는 플랫폼이라고 생각하고 있습니다. x86 을 위해 효율적이면서 안전한 VM 을 만들기 위해 수십 년간의 연구가 있었습니다.

이에 대해 정말 연구하고 싶다면 답을 멀리서 찾지 말고 안드로이드 에뮬레이터를 합리적인 성능 수준으로 작동시키는 것에 대해 Stackoverflow 에 올라와 있는 질문을 보시기 바랍니다. 기본적으로 대부분의 경우의 해결책은 ARM 대신 x86 가상 머신을 사용하는 것입니다.

물론 QEMU 와 같이 꽤 괜찮은 ARM VM 을 위한 일부 프로젝트가 있습니다. 그러나 요점은 x86 에뮬레이션이 상당히 간단한 해결책으로 알려진 문제라는 사실입니다. 유명한 인텔의 매뉴얼은 CPU 아키텍처에서 가장 잘 작성되고 명확한 문서로 간주됩니다. x86 에뮬레이터를 재미로

사용하는 고등학생들도 있습니다. ARM 에 대한 컴파일러 지원은 계속 향상되고 있지만, x86 에 비해서는 매우 부족한 수준입니다.

이제 x86 은 결코 단순한 ISA(Instruction Set Architecture)가 아닙니다. 그것은 8008 과 함께 70 년대부터 존재했고, 8088/8086 이후로 하위 호환성을 유지하고 있습니다. 이것은 실제로는 설계상의 희생을 치르고 있습니다. 왜냐하면 이들의 명령을 피하기 위해서 코드를 쓴 경우보다도, 하드웨어상으로는 아마도 소용없이, 실제로는 더 늦게 실행하는 연산 코드를 포함한 막대한 양의 연산 코드가 있기 때문입니다

제가 가장 좋아하는 예제는 80 년대부터 인기가 없었던 저주받은 2 진 코드화 된 10 진수 입니다. 그러나 이 예제는 몇 시간의 추가작업에 의한 복잡성이 더해지기 때문에 x86 사용의 이점은 우리가 아는 비용보다 훨씬 큽니다.

ARM 은 여전히 고려대상입니다. 특히 ARM 프로세서에서 일반적으로 작동하는 IoT 사용 사례가 있습니다. 현재 우리는 x86 에 초점을 맞추고 있지만 그 이후에는 특히 엔터프라이즈 및 허가된 블록체인에 대해 특별히 중점을 두고 있습니다.