A Protocol for Domain Name Resolution

컴퓨터네트워크 2 분반 - 컴퓨터공학과 C011103 손지석

1. 개요

Name resolution 을 요청하는 client 와 DNS server 의 역할을 하는 프로세스들이 UDP 통신을 통해 가상의 DNS 서비스를 시뮬레이션한다.

2. 구조

프로세스 간의 통신에서 주고받는 데이터(client-DNS 또는 DNS-DNS)로 python 의 pickle module 을 통해 직렬화된 dictionary object 를 이용한다. 즉 두 프로세스는 dictionary object 를 DNS message 처럼 주고 받으며 통신하며, 해당 프로젝트에서는 이를 message 라 정의한다. message 는 query type 과 reply type 두 가지 중 하나에 속한다. message 의 구조는 실제 DNS message 의 형식 일부와 과제의 요구 사항 등을 고려해 'query', 'recur_desire', 'answer', 'authority', 'additional', 'log'의 필드를 가지도록 정의했다. message 가 query type 인 경우 'query'는 resolve 하고자 하는 이름, 'recur_desire'은 recursive query 의 요청 여부를 의미한다. message 가 reply type 인 경우 'answer', 'authority', 'additional'은 각각 실제 DNS message 에서의 의미와 동일하다. 'log'는 쿼리 과정 중 지나쳐가는 DNS server 를 기록하기 위해 이용된다. 프로젝트의 구현에서는 message 의 역할을 하는 dictionary object 를 직접 다루지 않으며, Message 라는 class 를 통해 접근할 수 있도록 제한했다. Message class 는

message 의 데이터를 읽고 수정할 수 있게 할 뿐만 아니라 통신에 앞서 필요한 인코딩 등의 편의 기능을 제공한다.

Resource Record(이하 RR)를 표현하는 RR class 와 RR 이 저장된 cache(또는 데이터베이스)를 표현하는 Cache class 를 정의했다. RR 은 실제 DNS RR 의 필드 중 'name', 'value', 'type' 필드를 가지도록 정의했다. Cache 는 여러 RR 을 저장하고 있으며 새로운 RR 을 추가하거나 저장된 RR 로부터 query 받은 name 을 resolve 해준다.

과제에 등장하는 DNS server 들의 세부적인 동작은 다르지만 query message 를 수신한 뒤 적절한 resolution 과정을 통해 생성한 reply message 를 송신한다는 점에서 공통적이다. 따라서 caching 여부, recursive query 처리 가능 여부를 인자로 가지는 일반화된 resolution 로직을 설계할 수 있었고 이를 resolver 라는 함수로 구현했다.

DNS server 프로세스는 사용자의 명령 처리를 수행하는 main thread 와 UDP 를 통해 수신되는 message 를 처리하는 resolver thread 를 가진다. 이때 resolver thread 는 위에서 소개된 resolver 함수를 worker 로 가지는 thread 이다. Main thread 와 resolver thread 는 cache 와 recursive query 처리 가능 여부를 공유한다. 따라서 Lock 이라는 python 의 mutex 추상화를 이용해 자원에 대한 동기화된 접근이 가능하도록 구현했다.

3. 알고리즘

resolver 함수는 일반화된 name resolution 로직을 구현하며 caching 여부, recursive query 처리 가능 여부를 인자로 가진다. **1)** Resolution 과정은 UDP 로부터 query message 를 수신하는 것으로부터 시작된다. **2)** 먼저 query 가 resolve 하고자 하는 name 을 cache 에서 resolve 하도록 요청한다. Cache 에 resolve 를 요청하면 완전히 resolve 된 RR 을 반환하거나, resolve 할 수 없다면 resolve 가능한 다른 name server 의 RR 을 반환한다. **3)** 만약 완전히 resolve 되었거나 또는 recursive query 를 처리할 필요가 없다면 그 즉시 2)의 결과로 reply 한다. recursive query 를 처리할 필요가 없는 조건을 생각하기 위해 recursive query 를 처리할 필요가 있을 조건에 대해 먼저 생각해보자. Query 송신자가 recursive query 를 요청하고 자신이 recursive query 처리가 가능하다면, 또는 query 송신자가 recursive query 를 요청하고 그 송신자가 root DNS server 인 경우에 recursive query 를 처리해야 하며, 이 전체 조건의 부정이 recursive query 를 처리할 필요가 없는 조건이 된다. 3)의 이후를 실행하게 된다면 recursive query 를 처리하기로 결정한 것이다. **4)** 앞선 과정에서의 RR 을 통해 query 를 보낼 name server 의 IP 를 얻고 이를 목적지로 query 를 전달한다. 이때 IP 를 실제 목적지 IP 로 지정하지는 않으며 프로젝트에서 해당 IP 로 식별되는 name server 의 port 번호로 변환해 이용한다. **5)** 4)의 query 에 대한 reply 를 수신한 뒤 caching 이 설정되었다면 cache 에 reply 받은 RR 을 추가한다. 만약 resolve 되었다면 1)의 query 송신자에게 reply 하고 종료하며 그렇지 않다면 4)에서부터 반복한다. 이때 Recursive query 를 보냈지만 resolve 된 reply 를

수신받지 못한 것은 4)의 query 를 수신한 쪽이 recursive query 를 처리하지 않았다는 것이며 자신은 iterative query 를 수행해야 한다.

Cache class 의 resolve 함수는 전달받은 RR 의 list 로부터 name 을 resolve 하는 RR 또는 resolve 할 수 있는 name server 의 RR 을 탐색한다. 탐색한 결과는 'answer', 'authority', 'additional'의 의미에 맞게 반환된다.

4. 실행

해당 프로젝트는 macOS Sonoma 14.3.1, python 3.12.3 에서 작성되고 테스트되었다. 정상적인 실행을 위해서는 실행 스크립트(client, localDNSServer, rootDNSServer, comTLDDNSServer, companyDNSServer) 파일에 실행 권한이 부여되었는지, shebang 이 운영체제에 의해서 지원 가능한지, Python 버전이 해당 프로젝트 실행을 지원하는지, 설정 파일과 프로그램 실행 인자가 올바른 지 등을 확인해야 한다.

Github: https://github.com/jiseokson/dns_simulation