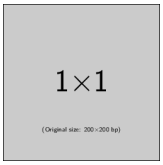

Fakulta informačních technologií
VUT v Brně

ISA projekt

DHCP server - manuál

David Kozák

20. listopadu 2016

Obsah

Kapitola 1

Úvod

Tento manuál popisuje způsob použití a implementační detaily dhcp serveru, který jsem naprogramoval v rámci projektu do předmětu ISA v roce 2016. Manuál je rozdělen následovně. V první fázi je stručně popsán způsob použití serveru. V druhé části se nachází popis tříd a modulů, ze kterých se program skládá. V další kapitole naleznete popis algoritmů využitých v projektu. Poslední část shrnuje rozšíření implementovaná nad rámec klasického zadání. Detailní popis mechanismů umožňujících tato rozšíření je umístěn přímo v těch částech dokumentu, kam tyto mechanismy logicky patří. Popis použití Po stažení je nejdříve nutné aplikaci přeložit za pomoci příkazu make. Vytvoří se spustitelný soubor ./dserver. Při spuštění je třeba uvést jeden povinný parametr -p <net-address>/<prefix>, který specifikuje pool adres, ze kterého může dhcp server adresy přidělovat. Volitelný přepínač -e exlcuded1,...excludedN, umožňuje z tohoto poolu vyloučit adresy. V praxi toto lze využít u klíčových komponent,u kterých je lepší adresu přidělit staticky. V rámci rozšíření byl ještě implementován přepínač -s, který specifikuje soubor se staticky mapovanými adresami. V tomto souboru se musí nacházet na každém řádku jedna MAC adresa následovaná IP adresou. Nevalidní formát vstupu, ať už špatně zadané přepínače či špatný formát MAC či IP adres způsobí ukončení aplikace s návratovým kódem 1. Po spuštění serveru bude aplikace poslouchat na příslušném portu, přijímat a odesílat zprávy. Pokud některá ze přijatých zpráv bude mít nesprávný formát, dojde pouze k vypsání chybového hlášení a server bude pracovat dále. K přerušení jeho činnosti povede pouze výjimka SocketException generovaná při chybě při práci s BSD sockety. Aplikace se také ukončí při obdržení signálu SIGINT.

Kapitola 2

Popis tříd a modulů

V této sekci jsou postupně popsány všechny třídy a moduly, ze kterých se projekt skládá. Jedná se spíše o stručný popis chování a propojení mezi třídami než o komplexní popis všech tříd, neboť ten by byl jistě velice vyčerpávající a poskytoval by více implementačních detailů, než je na této dokumentační úrovni nutné. Pokud máte zájem o skutečně detailní a kompletní popis, odkazuji Vás přímo na zdrojové kódy, kde jsou všechny důležité metody detailně komentovány. Projekt se skládá z několika balíků a třech top-level souborů: main.cpp, BaseObject.h a constants.h. Balíky v projektu jsou následující: addressing, exceptions, request, sockets, tests a threads.

main.cpp

V tomto souboru se nachází main funkce a parsování vstupních argumentů a v případě přepínač -e i vstupního souboru. Též je zde přítomna hlavní smyčka aplikace, ve které program čeká na přijetí zprávy, kterou následně zpracovává a případně odešle odpověď.

BaseObject.h

Obsahuje definici třídy BaseObject sloužící jako obecná supertřída pro všechny třídy v projektu. Tento mechanismus sloužil vynucení implementace metod jako například toString už všech tříd a zároveň též umožňuje uložit si libovolný objekt do obecné reference či pointeru.

constants.h

Zde se nacházejí globální konstanty jako například LEASE_TIME či návratové kódy.

Addressing

Tento balík modeluje pool adres využívaných dhcp serverem. Obsahuje následující třídy: AddressHandler,AddressCollector,AddressInfo,AddressPool, IpAddress,MacAddress a Timestamp.

IpAddress

Tato třída modeluji jedno ip adresu. Ta je interně reprezentována čtyřmi proměnnými typu unsigned char, ale v některých metodách se pro zjednodušenou manipulaci tento formát převádí do možná více typického jednoho integeru.

MacAddress

Tato třída reprezentuje jednu mac adresu. Ta je interně reprezentována šesti proměnnými typu unsigned char.

Timestamp

Tato třída reprezentuje časový otisk využitý pro detekování doby expirace adres. Při potvrzení bindingu po přijetí zprávy typu request se vytvoří toto časové razítko, které je poté testováno vláknem AddressCollector.

AddressInfo

Tato třída sdružuje všechny užitečné informace o jednom bindingu. Její položky jsou ipadresa, na kterou se zbylé informace váží, stav a v jakém tato adresa mometálně je. Pokud je adresa ve stavu TO_BE_BINDED,BINDED či DIRECT_MAPPING, obsahuje adres info též informace o mac adrese klienta, se kterou je tato ip spojena. Navíc pokud je adresa ve stavu BINDED, obsahuje i časovou známku Timestamp určující okamžik vzniku bindingu.

AddressPool

Tato třída reprezentuje pool adres, ze kterého dhcp server čerpá. Interně v sobě udržuje v lineráním seznamu instance třídy AdressInfo a poskytuje kominkační rozhraní pro přidělování, bindování a uvolňování adres.

AdressCollector

Tato vlákno má na starosti označování adres, jejichž doba bindingu již vypršela. Jedná se periodicky pracující vlákno, které projde celým seznamem, u adres s prošou dobou bindingu změní stav z BINDED na EXPIRED. Poté se na chvíli uspí, následně svou činnost opět opakuje.

AddressHandler

Tato třída zastřešuje práci s poolem adres. Jejím hlavním cílem je poskytnout synchronizaci při práci s poolem, ke kterému přistupuje více vláken, konkrétně v tomto případě dvě. Synchronizace je dosažena za využití instance třídy reentrant_mutex.

Exceptions

Jako mechanismus ošetřování chybových stavů se v projektu téměř výhradně využívají výjimky. Všechny vycházejí ze společné supertřídy BaseException, což umžňuje v případě nutnosti všechny druhy výjimek odchytnou jedním catch blokem. Toto samozřejmě v mnoha případech není příliš dobrá praktika, nicméně existují situace, kdy se to může hodit. V mém projektu tento mechanismus mimo jiné využívám jako poslední záchranu zabraňující, aby aplikace skončila jinak než v mé vlastní režii. Tento modul obsahuje následující třídy: BaseException,InvalidArgumentException,OutOfAddressException, ParseException a SocketException. Věřím, že jména jejdnotlivých výjimek jsou dostatečně výřečná, aby nebylo nutné je dále popisovat.

Request

Třidy v tomto balíku reprezentují jednotlivé zprávy protokolu DHCP. Nachází se zda dvě bázové abstraktní třídy AbstractRequest, společná supertřída pro Discover,Request a Release, a AbstractReply, společná supertřída pro Offer,Ack a Nack. AbstractRequest i AbsractReply obsahují pure virtual metodu perfromRequest, jejíž implementací jednotlivé subtřídy specifikují

svoje chování. Další třídou v tomto balíku je ProtocolParser, který zpracuje dhcp zprávu a vytvoří příslušnou instanci subtřídy AbstractRequest. Jako zajímavost mohu uvést, že zde je vyžito modelování konstrukce switch s využitím polymorphismu. Metoda parseRequest objektu ProtocolParser vrací ukazatel na objekt typu AbstractRequest, nad kterým je později volána výše zmíněná metoda performRequest, která v závislosti na konkrétní subtřídě bude mít odlišnou implementaci. Poslední, ale také velice důležitou třídou, je DhcpMessage, která reprezentuje jednu dhcp zprávu. Objekty této třídy v konstruktoru přijmou vector obsahující dhcp zprávu přijatou od klienta a rozparsují ji. Implementace této třídy může vypadat na první pohled celkem děsivě, poskytuje ale poměrně dobrou formu abstrakce. Třída též obsahuje metodu pro vytvoření výstupní zprávy, která je poté socketem odeslána zpět klientovi. Ještě jedna zajímavost může být objevena ve třídě AbstractReply, ta totiž metodu performRequest implementuje i přesto, že se jedná o čistě virtuální metodu. Toto chování je v c++ validní a má za následek, že subtřídy musí tuto metodu stejně implementovat jako by se jednalo o klasickou čistě virtuální metodu. V programu je toto schéma využito následujícím způsobem. V metodě performRequest v supertřídě je umístěna implementace společná pro všechny tři subtřídy typu AbstractReply, ty tuto metodu volají na počátku implementace svých metod performRequest a poté už pouze dolní implementaci specifickou pro danou odpověď.

Sockets

Tento balík obsahuje jedinou třídu Socket. Tato třida interně využívá implementaci BSD socketů a poskytuje čisté rozhraní pro komunikace.

Threads

Tento balík obsahuje třídy pro paralelní programování. Všechny tyto třídy interně využívají std::thread, ale poskytují vyšší míru abstrakce. První třídou je ThreadWrapper. Jedná se o abstraktní třídu, ze které je při dědění nutné naimplementovat abstraktní virtuální metodu run. Výhoda využití této třídy tkví v tom, že můžete objekt vytvořit na jednom místě a spustit ho až později, což klasická implementace vláken std::thread neumožňuje. Z této třídy dědí ResponseThread, které v metodě run implementuje parsování a zpracování jedné dhcp zprávy. Dalším rozšířením je třída CancellableThread, která z ThreadWrapper dědí a umožňuje definovat task prováděný asynchronně ve smyčce, dokud z řídícího vlákna nedojde k volání metody interrupt. Z tohoto

vlákna dědí třída MainThread, která v této cyklicky volané metodě načítá data ze socketu, a následně vytváří instanci třídy ResponseThread, které přijatou zprávu zpracuje a případně pošle odpověď. Tento balík obsahuje třídy ThreadWrapper,CancellableThread,MainThread a ResponseThread.

Tests

Tento balík obsahuje unit testy hlavně tříd balíku addressing.

Kapitola 3

Algoritmy využité v projektu

3.1 Vyhledání adresy

Při inicializaci AddressPoolu dojde k vytvoření listu objektů typu AddressInfo. Tyto objekty se mnou nacházet v těchto stavech: FREE – volná adresa připravená k použití TO_BE_BINDED – adresa, která byla již ve zprávě offered někomu nabídnuta, ale ještě od něj nepřišla zpráva Request. U tohoto stavu již je poznačena MAC adresa clienta, na základě které je poté klient identifikován, neboť dhcp option client identifier není podporována. BINDED – adresa je momentálně používána nějakým klientem, její lease time ještě nevypršel EXPIRED – adresa byla a možná i stále je používána nějakým klientem, ale už jí vypršel lease time. Adresa je v tomto speciálním stavu, aby mohla byla prioritně opět přiřazena původnímu klientovi, pokud zašle zprávu offer. Pokud ale dojdou v poolu adresy, server ji již může přiřadit někomu jinému. RESERVED – rezervovaná adresa, pool ji nesmí nikdy poskytnout DIRECT_MAPPING – staticky mapovaná adresa, pool ji poskytne pouze klientovi s danou MAC adresou Při obdržení zprávy Discover se nejdříve provede kontrola, zda se nejedná o přímé mapování, které je uloženo v asociativní mapě MAC adress na IP adresy. Pokud je adresa nalezena, vyhledávání úspěšně končí. Jinak se postupně několikrát prochází výše zmíněný seznam adres. Nejdříve dochází k vyhledání adresy ve stavu FREE. Pokud žádná adresa ve stavu FREE není, dochází k vyhledání adresy ve stavu TO_BE_BINDED. Pokud ani takováto adresa v poolu není, dochází k vyhledání adresy ve stavu EXPIRED. Pokud bylo některé z předchozích vyhledání úspěšné, daná adresa přejde do stavu TO_BE_BINDED a je nabídnuta klientovi zprávou offer. Pokud žádná adresa nalezena nebyla, server vypíše chybové hlášení a klientovi neodpovídá. Při obdržení zprávy Request opět dochází k průchodu seznamu adres a vy-

hledání adresy ve stavu TO_BE_BINDED,BINDED či EXPIRED. Pokud byla ve zprávě request specifikována IP adresa v poli ciaddr, tak dochází k vyhledání za pomoci této IP adresy. Pokud záznam pro danou IP nalezen nebyl či IP specifikována nebyla, dojde k vyhledání na základě MAC adresy. Pokud záznam byl nalezen, dojde k vytvoření nového Timestampu a záznam přechází do stavu BINDED. Ze stavu BINDED do EXPIRED záznam případně přesune AddressCollector při svém periodickém průchodu seznamem. Jelikož zde dochází k vyhledávání v neseřazeném seznamu, musí dojít k lineárnímu průchodu, v tomto případě dokonce opakovaně. Zde se nabízí prostor pro optimalizaci. Jako možné zlepšení se nabízí například využít seřazený seznam, ve kterém by byly adresy na základě stavu v tomto pořadí: DIRECT_MAPPING,FREE,TO_BE_BINDED,EXPIRED a na pořadí dalších stavů RESERVED a BINDED již nezáleží, neboť se nevyhledávají. Samotné seřazení seznamu by umožňovalo seznamem procházet pouze jednou, ale za cenu nutnosti neustále seznam znovu řadit při při změně stavu adresy. Další možnost pro vylepšení by bylo rozdělit tento velký list na sérii menších listů, které by byly postupně procházeny ve výše specifikovaném pořadí. Nevýhoda tohoto řešení nicméně spočívá v náročnější implementaci operací nad poolem adres. Jako poslední a dle mého názoru celkem zajímavá varianta by bylo využít datové struktury skip listu, který by umožnil definovat nad seznamem adres operace průchodu jen pro daný stav adresy. Toto řešení by bylo jistě velice zajímavé a vzhledem k nutnosti modifikovat pouze ukazatele, přes které by docházelo k jednotlivých průchodům, by mohlo být i efektivní. Bohužel mi nezbyl čas tento pokus implementovat. Nicméně při testování jsem došel k závěru, že i současný nepříliš efektivní algoritmus je pro standardní nasazení dostatečně rychlý.

3.2 Zajištění thread safety

Mainthread a AdressCollector

Jelikož obě výše zmíněná vlákna přistupují k objektu třídy ddressHandler, je nutno jejich činnost synchronizovat. Tato synchronizace probíhá za pomoci mutexu, který je jedním z fieldů AddressHandleru. Každá metoda AddressHandleru začíná synchronizací za pomoci vytvoření objektu lock_guard, který ve svém konstruktoru volá nad mutexem metodu lock a v destruktoru volá metodu unlock. Tímto jednoduchým způsobem je na základě principu RAE vytvořena jedním příkazem typická javovská konstrukce try-finally.

MainThread a SignalHandler

Zde je synchronizace zajištěna méně čistým způsobem. Obě tyto vlákna spolu sdílejí objekt typu Mutex, boolovský flag isInterrupted a ukazatel na objekt socketu, který server využívá. Při obdržení signálu SIGINT signal handler uzavře socket a nastaví flag isInterrupted na true. Synchronizace je naimplementována tak, že po obdržení signálu SIGINT server ještě může právě zpracovávanou žádost dokončit, ale poté je jeho cyklus přerušen při ověřování flagu isInterrupted, který již bude v hodnotě True. Při využití samotného interrupt flagu nastal problém, že hlavní vlákno zůstávalo zablokováno na volání getMessage. Proto bylo třeba do sdílených objektů přidat i referenci na socket, aby ho signal handler mohl uzavřít a tímto tuto blokující operaci přerušit. V odchytnutí výjimky SocketException je ještě potřeba odlišit, zda byla tato výjimka způsobena uzavřením socketu z signal handleru, či zda se jednalo o nějakou chybu socketu. Jelikož signal handler zároveň nastavuje flag isInterrupted, stačí v catch bloku tento flag zkontrolovat a na základě jeho hodnoty nastavit příslušný návratový kód.

Kapitola 4

Rozšíření

Prvním rozšířením, které jsem v projektu implementoval, byl přepínač -s file, kterým lze specifikovat soubor se staticky přidělenými adresami. Jak již bylo popsáno výše, tento soubor interně reprezentuji jako map mac adres na ip adresy. Tyto adresy jsou poté do poolu umístěny ve specifickém a neměnné stavu DIRECT_MAPPING. Při vyhledání volné adresy pro zprávy discover či potvrzení bindingu pro zprávu request jsou poté tyto žádosti nejdříve prověřovány, zda se nejedná o statickou alokaci, a pokud ano, je danému klientovi přidělena staticky alokovaná ip adresa. Dalším rozšířením, které jsem implementoval, je modul threading. Když jsem na projektu začal pracovat, ještě jsem netušil, že bude stačit iterativní server. Jako první jsem se rozhodl naimplementovat právě modul umožňující serveru pracovat konkurentně. I když bylo poté určeno, že server bude iterativní, rozhodnul jsem se tento modul přiložit k projektu alespoň pro zajímavost, byť není momentálně programem využíván. Základní kód je přichystán, chybí pouze ošetřování chybových stavů a případné propagování chyby.