Dokumentace k projektu ISA DHCP Server

Dokumentace je rozdělena 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.

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 modul obsahuje následující třídy:
AddressHandler,AddressCollector,AddressInfo,AddressPool,IpAddress,MacAddress a Timestamp.
Exceptions
Tento modul obsahuje následující třídy:
BaseException,InvalidArgumentException,OutOfAddressException,ParseException a SocketException.

Request
Tento modul obsahuje následující třídy: DhcpMessage,ProtocolParser, AbstractRequest, Discover, Request,Release, AbstractReply, Offer, Ack, Nack.

Sockets
Tento modul obsahuje pouze třídu Socket.

Threads
Tento modul obsahuje třídy ThreadWrapper,CancellableThread,MainThread a ResponseThread.

Tests
Tento modul obsahuje unit testy hlavně tříd balíky addressing.

Algoritmy využité v projektu.
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 vyhledání 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.

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.