

# Na tomto místě bude oficiální zadání vaší práce

- Toto zadání je podepsané děkanem a vedoucím katedry,
- musíte si ho vyzvednout na studijním oddělení Katedry počítačů na Karlově náměstí,
- v jedné odevzdané práci bude originál tohoto zadání (originál zůstává po obhajobě na katedře),
- ve druhé bude na stejném místě neověřená kopie tohoto dokumentu (tato se vám vrátí po obhajobě).



České vysoké učení technické v Praze  
Fakulta elektrotechnická  
Katedra počítačů



Bakalářská práce

## **Simulátor virtuální počítačové sítě Linux**

*Tomáš Pitřinec*

Vedoucí práce: Ing. Pavel Kubalík, Ph.D.

Studijní program: Softwarové technologie a management, Bakalářský

Obor: Softwarové inženýrství

24. května 2010



## Poděkování

Zde můžete napsat své poděkování, pokud chcete a máte komu děkovat.



## Prohlášení

Prohlašuji, že jsem práci vypracoval samostatně a použil jsem pouze podklady uvedené v příloženém seznamu.

Nemám závažný důvod proti užití tohoto školního díla ve smyslu §60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Červeném Kostelci dne 25.5.2010

.....





# Abstract

Translation of Czech abstract into English.

# Abstrakt

Abstrakt práce by měl velmi stručně vystihovat její podstatu. Tedy čím se práce zabývá a co je jejím výsledkem/přínosem.

Očekávají se cca 1 – 2 odstavce, maximálně půl stránky.



# Obsah

<b>1</b>	<b>Úvod</b>	<b>1</b>
1.1	Cíle práce . . . . .	1
1.2	Rozdělení práce . . . . .	1
1.3	Struktura práce . . . . .	2
<b>2</b>	<b>Existující řešení</b>	<b>3</b>
<b>3</b>	<b>Analýsa aplikace</b>	<b>5</b>
3.1	Požadavky na aplikaci . . . . .	5
3.1.1	Funkční požadavky . . . . .	5
3.1.2	Nefunkční požadavky . . . . .	6
3.2	Analýsa požadavků . . . . .	6
3.2.1	Připojení pomocí telnetu . . . . .	6
3.2.2	Podobnost simulátoru se skutečným linuxem . . . . .	6
3.2.3	Počet simulovaných počítačů . . . . .	7
3.3	Programovací jazyk a uživatelské rozhraní . . . . .	7
3.3.1	Programovací jazyk . . . . .	7
3.3.2	Uživatelské rozhraní . . . . .	7
3.4	Návrh architektury . . . . .	7
3.4.1	Virtuální síť . . . . .	7
3.4.1.1	Síťové prvky . . . . .	8
3.4.1.2	Posílání paketů . . . . .	8
3.4.2	Komunikační vrstva . . . . .	8
3.5	Odhad náročnosti aplikace . . . . .	8
3.6	Odhad složitosti práce a jejího průběhu . . . . .	8
<b>4</b>	<b>Implementace virtuální sítě</b>	<b>9</b>
4.1	Popis architektury aplikace . . . . .	9
4.1.1	Komunikační vrstva . . . . .	9
4.1.2	Aplikační vrstva - virtuální síť . . . . .	10
4.2	IP adresa . . . . .	10
4.2.1	Analýza . . . . .	10
4.2.2	Vnitřní reprezentace . . . . .	11
4.2.3	Veřejné metody . . . . .	11
4.3	Virtuální počítač . . . . .	11

4.3.1	Síťové rozhraní	12
4.4	Infrastruktura virtuální sítě	12
4.5	Routovací tabulka	13
4.5.1	Analýza routovací tabulky na skutečném počítači	13
4.5.1.1	Struktura tabulky	13
4.5.1.2	Adresát	14
4.5.1.3	Příznaky	14
4.5.1.4	Přidávání záznamů a jejich řazení	14
4.5.1.5	Mazání záznamů	16
4.5.1.6	Použití pro směrování	16
4.5.2	Implementace routovací tabulky v simulátoru	16
4.5.2.1	Vnitřní reprezentace	16
4.5.2.2	Přidávání, mazání a řazení záznamů	16
4.5.2.3	Použití při směrování	17
4.6	Posílání paketů	17
4.6.1	Teoretický rozbor referenčního modelu ISO/OSI	17
4.6.1.1	Spojová vrstva	17
4.6.1.2	Síťová vrstva	17
4.6.1.3	Transportní vrstva	17
4.6.1.4	Datové bloky	18
4.6.2	Implementace třídy Paket	18
4.6.3	Chování reálného počítače při posílání paketů	18
4.6.3.1	Transportní vrstva - protokol ICMP	18
4.6.3.2	Síťová vrstva - protokol IP	18
4.6.3.3	Linková vrstva	19
4.6.3.4	Zajímavá zjištění	20
4.6.4	Implementace v simulátoru	21
4.6.4.1	Linková vrstva	21
4.6.4.2	Síťová vrstva	21
4.6.4.3	Transportní vrstva	21
<b>5</b>	<b>Implementace příkazů</b>	<b>23</b>
5.1	Popis zpracování příkazové řádky	23
5.1.1	Třída Abstraktni	23
5.1.2	Třída ParserPrikazu	24
5.1.3	Třída AbstraktniPrikaz	24
5.2	Společné znaky příkazů	25
5.3	Příkaz ifconfig	25
5.3.1	Teoretický úvod	25
5.3.2	Rozsah implementace v simulátoru	25
5.3.3	Analýza ifconfigu na skutečném počítači	25
5.3.4	Implementace v simulátoru	27
5.3.5	Možnosti dalšího vylepšení	27
5.3.6	Znamé odchylky	27
<b>6</b>	<b>Popis problému, specifikace cíle</b>	<b>29</b>

<b>7</b>	<b>Analýza a návrh řešení</b>	<b>31</b>
<b>8</b>	<b>Realizace</b>	<b>33</b>
<b>9</b>	<b>Testování</b>	<b>35</b>
<b>10</b>	<b>Závěr</b>	<b>37</b>
<b>A</b>	<b>Testování zaplnění stránky a odsazení odstavců</b>	<b>39</b>
<b>B</b>	<b>Pokyny a návody k formátování textu práce</b>	<b>41</b>
B.1	Vkládání obrázků . . . . .	41
B.2	Kreslení obrázků . . . . .	42
B.3	Tabulky . . . . .	42
B.4	Odkazy v textu . . . . .	43
B.4.1	Odkazy na literaturu . . . . .	43
B.4.2	Odkazy na obrázky, tabulky a kapitoly . . . . .	45
B.5	Rovnice, centrovaná, číslovaná matematika . . . . .	45
B.6	Kódy programu . . . . .	46
B.7	Další poznámky . . . . .	46
B.7.1	České uvozovky . . . . .	46
<b>C</b>	<b>Seznam použitých zkratk</b>	<b>47</b>
<b>D</b>	<b>UML diagramy</b>	<b>49</b>
<b>E</b>	<b>Instalační a uživatelská příručka</b>	<b>51</b>
<b>F</b>	<b>Obsah přiloženého CD</b>	<b>53</b>



# Seznam obrázků

4.1	Komunikační vrstva . . . . .	10
4.2	Architektura aplikační vrstvy - virtuální síť . . . . .	11
4.3	Funkční síť se „špatnými“adresami . . . . .	20
5.1	Třídy pro zpracování příkazové řádky . . . . .	24
B.1	Popiska obrázku . . . . .	42
F.1	Seznam přiloženého CD — příklad . . . . .	53





# Seznam tabulek

B.1 Ukázka tabulky . . . . .	42
------------------------------	----



# Kapitola 1

## Úvod

Jednou z laboratorních úloh předmětu Počítačové sítě (Y36PSI) na FELu je postavení sítě mezi několika linuxovými a ciscovými počítači. Studenti, kteří tento úkol plní, nemají často s takovou činností žádnou osobní zkušenost, a tak během úlohy řeší různé banální problémy, kterým by se mohli vyhnout, kdyby měli možnost zkusit si nastavit podobou síť již před samotnou laboratorní úlohou. Mohl by se jim hodit simulátor, který by jednoduše spustili na svém počítači a na kterém by si mohli nastavování síťových parametrů na linuxu a ciscu zkusit. Právě návrhem a implementací takového síťového simulátoru počítačů s OS Linux se zabývá tato bakalářská práce.

### 1.1 Cíle práce

Cílem práce je v programovacím jazyce Java SE navrhnout a implementovat aplikaci, která umožní vytvoření virtuální počítačové sítě, pro potřeby předmětu Y36PSI. Z pohledu uživatele by aplikace měla vypadat stejně jako reálná síť. Uživatel spustí aplikaci v konsoli a pak se pomocí telnetu připojí k jejím jednotlivým virtuálním počítačům, podobně jako protokolem ssh k počítačům s OS Linux. Aplikace bude podporovat příkazy potřebné ke konfiguraci síťových rozhraní (ifconfig, ip address), směrování (route, ip route) a překladu adres (iptables -t nat). Pro ověření správnosti konfigurace sítě budou implementovány příkazy ping a traceroute.

Nastavenou konfiguraci sítě bude možné uložit do souboru a zase ji ze souboru načíst. Uživatel bude mít možnost vytvářet libovolné sítě s libovolným počtem počítačů typu linux nebo cisco tak, že infrastrukturu sítě napíše do konfiguračního souboru a pak ji z něho načte.

### 1.2 Rozdělení práce

Protože kompletní síťový simulátor pro počítače s Cisco IOS i OS Linux by přesahoval rozsah jedné bakalářské práce, byla práce na aplikaci rozdělena na tři části:

- **Jádro aplikace**

Jedná se především o datové struktury virtuálního počítače, komunikaci s uživatelem

po síti a načítání a ukládání souborů. Na této části jsem spolupracoval se Stanislavem Řehákem.

- **Linuxová část**

V této části je potřeba napsat parsery linuxových příkazů a zjistit, jak se chovají počítače s linuxem v síťové komunikaci a toto chování implementovat.

- **Ciscová část**

Touto částí se tato práce nezabývá, zabývá se jí bakalářská práce Simulátor virtuální počítačové sítě Cisco Stanislava Řeháka.

## 1.3 Struktura práce

Tady bude popis struktry týchle práce, jako toho textu.

## Kapitola 2

# Existující řešení

Tady bude řešení, až ji udělám.



## Kapitola 3

# Analýsa aplikace

V této kapitole se zabývám analýsou a návrhem aplikace jako celku. Shrnuji a analyzuji požadavky, diskutuji zvolený jazyk a uživatelské rozhraní, navrhuji architekturu aplikace a odhaduji její náročnost. Analýsa jednotlivých částí simulátoru je popisována společně s těmito částmi.

### 3.1 Požadavky na aplikaci

Nejprve shrnu všechny požadavky na moji aplikaci.

#### 3.1.1 Funkční požadavky

1. Vytvoření počítačové sítě založené na počítačích OS Linux.
2. Aplikace umožňuje konfiguraci rozhraní pomocí příkazů `ifconfig` a `ip addr`.
3. Aplikace obsahuje funkční směrování a umožňuje jeho nastavování pomocí příkazů `route` a `ip route`.
4. Aplikace implementuje překlad adres
5. Aplikace podporuje ukládání a načítání do/ze souboru.
6. Pro ověření správnosti jsou implementovány příkazy `ping` a `traceroute`.
7. K jednotlivým počítačům aplikace je možné se připojit pomocí `telnetu`.
8. Pomocí `telnetu` bude možno se připojit zároveň k více virtuálním počítačům.
9. Pomocí `telnetu` bude možno připojit se k jednomu počítači vícekrát najednou.

### 3.1.2 Nefunkční požadavky

1. Aplikace bude multiplatformní - alespoň pro operační systémy Windows a Linux
2. Aplikace musí být spustitelná na běžném<sup>1</sup> studentském počítači.
3. Aplikace by měla být co nejvěrnější kopií reálného počítače s Linuxem.

## 3.2 Analýsa požadavků

### 3.2.1 Připojení pomocí telnetu

Jedním z funkčních požadavků mé aplikace je možnost připojit se k jednotlivým virtuálním počítačům pomocí protokolu telnet. Tento požadavek vypadá jednoduše, pokud pod pojmem Telnet chápeme jednoduchý protokol na přenos textových dat. Takový protokol ovšem neumožňuje doplňování příkazů a jejich historii, což je pro práci s počítačem, byť virtuálním, obrovské omezení. Oproti tomu, implementovat telnet protokol, jako NVT<sup>2</sup>, kde se posílá a potvrzuje každý napsaný znak, by překračovalo rozsah této bakalářské práce. Můj kolega našel program rlwrap, který poskytuje historii příkazů a jejich doplňování na straně klienta. Funguje v linuxu, na windowsu jen pod cygwinem. Toto druhé řešení bude o dost jednodušší a rozhodli jsme se ho realizovat, i když pro uživatele bude nevýhodou spouštění přes cygwin. I tak ovšem základní požadavek, že s aplikací bude možno komunikovat pomocí telnetu, zůstane zachován, uživatel ovšem přijde o komfort, který mu nabízí možnost doplňování, editace a historie příkazů.

### 3.2.2 Podobnost simulátoru se skutečným linuxem

Aby byl simulátor využitelný pro výukové účely, musí být dostatečně podobný skutečnému linuxu, aby uživatel mohl věřit, že to, co funguje v simulátoru, bude fungovat i na skutečném linuxu a naopak. K tomu bude stačit implementovat jen ty příkazy, kterými se nastavují síťové parametry, a jen v takovém rozsahu, jaký je pro tyto výukové účely potřeba. Budu tedy implementovat příkazy `ifconfig`, `verb`, `route`, `ping` a `traceroute`, z příkazu `ip` stačí implementovat jeho podpříkazy `addr` a `route`. Pro potřeby nastavení překladu adres je potřeba implementovat malou část příkazu `iptables`. Aby uživatel mohl nastavovat některé hodnoty souborů v adresáři `/proc`, implementuji ve velmi omezené míře i příkazy `cat` a `echo`, ovšem jen pro tyto soubory. Pro ukončení spojení bude implementován příkaz `exit`. Pro potřeby simulátoru ale není potřeba implementovat kompletní příkazy `ifconfig` nebo `ip`, ale jen tu jejich část, kterou se nastavují parametry rozhraní, jako IP, maska a další. O ostatních parametrech pak většinou simulátor vypíše, že ve skutečnosti sice existují, ale simulátorem zatím nejsou podporované.

---

<sup>1</sup> Slovem „běžné“ se myslí v podstatě jakýkoliv počítač, na kterém je možné nainstalovat prostředí Javy - Java Runtime Environment

<sup>2</sup> NVT – Network Virtual Terminal, česky: Síťový virtuální terminál; poskytuje standardní rozhraní příkazové řádky



### 3.2.3 Počet simulovaných počítačů

Na laboratořích Y36PSI studenti konfiguruji 4 počítače, náš simulátor by měl zvládnout simulovat síť o 10 počítačích. Více ani není potřeba, pro výukové účely studenti pravděpodobně nebudou konfigurovat více počítačů.

## 3.3 Programovací jazyk a uživatelské rozhraní

### 3.3.1 Programovací jazyk

Aplikaci jsme se rozhodli programovat v programovacím jazyku Java z několika důvodů. Java je programovací jazyk, který nabízí velký programátorský komfort, stabilitu a zároveň možnost vytvořené aplikace používat pod různými operačními systémy, což je další z nefunkčních požadavků. Tento jazyk navíc disponuje hotovými knihovnami pro práci se sítí v balíčku `java.net`. Dalším důvodem je také to, že s programováním aplikací v Javě mám zatím asi největší zkušenosti.

### 3.3.2 Uživatelské rozhraní

Jak plyne ze zadání, uživatel se přihlašuje k jednotlivým virtuálním počítačům pomocí programu `telnet`, nemusím tedy vytvářet žádného speciálního klienta. S aplikací samotnou nebude uživatel nijak pracovat, jenom ji spustí se správným konfiguračním souborem a případně číslem výchozího portu, dále již bude nastavovat pouze jednotlivé virtuální počítače pomocí `telnetu`. Pro takovou aplikaci je nejlepším uživatelským rozhraním příkazová řádka, vytvoření grafického uživatelského rozhraní by nemělo smysl.

## 3.4 Návrh architektury

Aplikace se bude skládat ze dvou vrstev. Komunikační vrstva by měla zajišťovat síťovou komunikaci s klientem, tedy odesílání a přijímání textových dat. Z velké části bude převzata z jiné práce, kterou jsme kdysi dělali jako domácí úkol na předmět Y36PSI. Aplikační vrstva bude tvořena samotnou virtuální sítí. Tyto vrstvy však od sebe nebudou striktně odděleny. Nejprve si rozebereme druhou vrstvu.

### 3.4.1 Virtuální síť

Virtuální počítačová síť, kterou bude aplikace simulovat, má poskytovat především tyto funkcionality:

- Možnost konfigurace jednotlivých síťových prvků.
- Posílání paketů mezi síťovými prvky.

Skutečná počítačová síť se skládá ze síťových prvků různých druhů. Stejně tak i virtuální síť se bude skládat ze síťových prvků, které budou interně reprezentovány objekty.

### 3.4.1.1 Síťové prvky

V laboratořích předmětu Y36PSI studenti nastavují pouze PC nebo směrovače na 3. (síťové) vrstvě ISO/OSI modelu<sup>3</sup>. Síťové prvky pracující na 2. vrstvě ISO/OSI modelu<sup>4</sup>, switche a bridge se v laboratořích vůbec neuvažují. Proto i ve své práci uvažují jediný druh síťových prvků - počítače s OS Linux.

### 3.4.1.2 Posílání paketů

Virtuální síť musí umět posílat virtuální pakety, aby uživatel pomocí příkazů **ping** nebo **traceroute** zjistil, jestli virtuální síť správně nakonfiguroval. Posílání paketů bude vnitřně re-alisováno vzájemným voláním metod virtuálních počítačů, které si mezi sebou budou předávat objekty typu paket. Tyto metody zřejmě bude vhodné rozdělit tak, aby odpovídaly jednotlivým vrstvám ISO/OSI modelu.

### 3.4.2 Komunikační vrstva

Komunikační vrstva simulátoru bude zajišťovat spojení aplikace s klientem. Z tohoto pohledu bude simulátor klasickým síťovým serverem, který poslouchá na několika portech, přijímá spojení a zpracovává je. Uživatel bude po síti konfigurovat jednotlivé virtuální počítače, proto každý virtuální počítač musí poslouchat na jednom portu. Pro obsluhu této komunikace bude vytvořeno několik tříd. Aby mohl simulátor poslouchat na více portech najednou, bude nutné vytvořit více vláken, každý virtuální počítač tedy poběží v samostatném vláknu. Jak plyne z posledního funkčního požadavku, musí jeden virtuální počítač umět zpracovat i více spojení najednou, jako i na reálný linuxový počítač je možné se připojit k několika jeho terminálům pomocí protokolu ssh nebo telnet. Proto bude nutné, aby vlákno, které poslouchá na portu, pro příchozí spojení vytvořilo jiné vlákno, které spojení obslouží, a samo dále poslouchalo na určeném portu.

## 3.5 Odhad náročnosti aplikace

Aplikace nebude mít žádné uživatelské rozhraní, nebude přistupovat do žádné database a její datové struktury budou pravděpodobně poměrně jednoduché. Proto pro virtuální síť o deseti počítačích by simulátor neměl mít žádné zvláštní paměťové nebo procesorové nároky.

## 3.6 Odhad složitosti práce a jejího průběhu

---

<sup>3</sup>3. vrstva ISO modelu, tzv. síťová vrstva, zajišťuje spojení mezi jakýmkoliv 2 uzly sítě.

<sup>4</sup>2. vrstva ISO/OSI modelu, tzv. spojová nebo linková vrstva, zajišťuje spojení mezi dvěma sousedními systémy.

## Kapitola 4

# Implementace virtuální sítě

V této kapitole se zabývám analysou a implementací jednotlivých částí aplikace. Nejdříve popisuji architekturu aplikace jako celku, dále rozebírám analýzu a implementaci třídy **IpAdresa**, implementaci virtuálního počítače, analýzu a implementaci routovací tabulky a analýsu a implementaci posílání paketů. Nezabývám se zde analysou a implementací jednotlivých příkazů, vzhledem k rozsáhlosti tohoto tématu jsem ho vyčlenil do zvláštní kapitoly, která následuje za touto kapitolou.

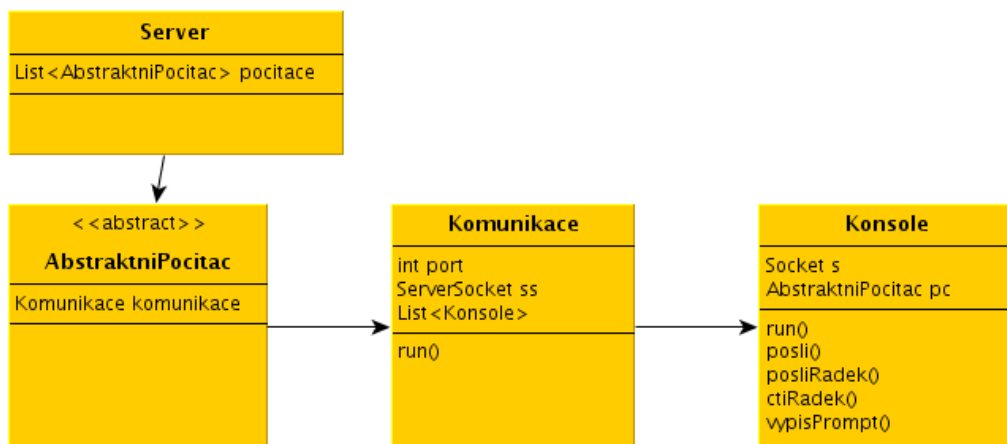
### 4.1 Popis architektury aplikace

Aplikace se skládá ze dvou vrstev. První, komunikační vrstva zajišťuje veškerou síťovou komunikaci s klientem, druhá, aplikační vrstva reprezentuje virtuální síť, která je simulována.

#### 4.1.1 Komunikační vrstva

Komunikační vrstva byla z velké části přejata z úkolu na předmět Y36PSI, který jsme programovali na podzim roku 2008. Síťová komunikace s klientem je velmi jednoduchá, server s klientem si navzájem posílají jen textová data, tzn. klient posílá serveru příkazy v textové podobě a server na ně odpovídá.

Hlavní třídou komunikační vrstvy je třída **Komunikace**. Ta se stará o veškerou komunikaci virtuálního počítače s uživatelem. Je potomkem třídy **Thread**. Běží ve vlastním vlákně, které se startuje v jejím konstruktoru, a poslouchá na portu, který jí byl zadán. Pro každé nové příchozí spojení vytvoří instanci třídy **Konsole**, která spojení obslouží, aby komunikace mohla dále poslouchat na portu a zpracovávat další spojení. Třída **Konsole** je také potomkem třídy **Thread**. Obsluhuje jedno telnetové připojení. Drží si instanci třídy **ParserPrikazu** z balíčku **Prikazy** (o něm v následující kapitole). Přijímá textová data od uživatele až po enter (sekvence `\r\n`), tedy vlastně načítá data po řádcích. Každý řádek, který uživatel pošle, předá parseru na zpracování a pak sama pošle uživateli prompt. Parseru poskytuje metody pro posílání textových dat uživateli. Pro uživatele tak komunikace s touto konsolí vypadá stejně jako práce s příkazovou řádkou na skutečném počítači.



Obrázek 4.1: Komunikační vrstva

#### 4.1.2 Aplikační vrstva - virtuální síť

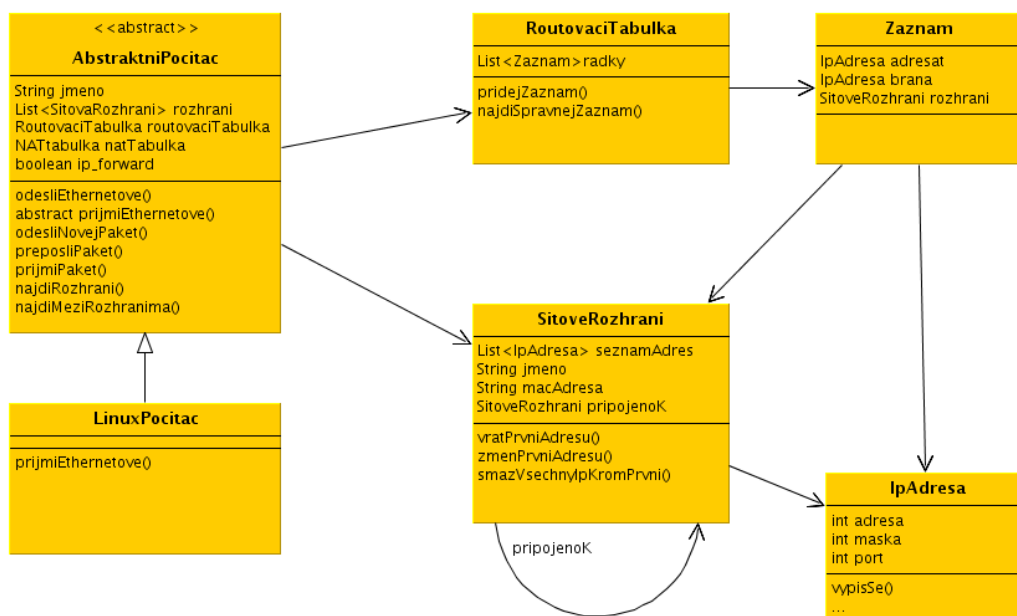
Před tím, než začnu popisovat implementaci jednotlivých komponent aplikační vrstvy, sluší se, popsat ji zhruba jako celek. Virtuální síť se skládá z virtuálních počítačů, což jsou objekty potomků abstraktní třídy **AbstraktniPocitac**. Ty si mezi sebou voláním svých metod předávají pakety, což jsou objekty třídy **Paket**. Virtuální počítače mají síťové rozhraní, což jsou objekty třídy **SitovaRozhrani**. Jejich pomocí jsou počítače mezi sebou propojeny (více o infrastruktuře v 4.4)

## 4.2 IP adresa

Třída **IpAdresa** je sice jen jednou z mnoha tříd, vzhledem k jejímu významu ji ale v následujících odstavcích popíšu podrobněji.

### 4.2.1 Analýza

Protože simulátor se zabývá především simulací síťové vrstvy ISO/OSI modelu, je síťová adresa počítače, tzv. IP adresa velmi často používanou datovou strukturou, pro kterou se vyplatí mít speciální třídu. Ta se v aplikaci jmenuje **IpAdresa** a patří do balíčku **datoveStruktury**. Je používána jako parametr síťového rozhraní, jako prefix v routovací tabulce, jako zdrojová a cílová adresa v paketech. Při bližším pohledu je zřejmé, že kontext jejího použití se v těchto případech částečně liší. Například pro posílání paketů je nutné, aby paket obsahoval zdrojovou a cílovou adresu i s portem. Port by samozřejmě nemusel být součástí adresy, to se ale ukázalo jako jednodušší a pro posílání paketů přehlednější možnost. Pro IP adresu jako parametr rozhraní je naopak port zcela nesmyslný parametr, nutně ale potřebuje parametr pro síťovou masku, která je naproti tomu nesmyslná pro posílání paketů. Paket posílám na IP adresu, ne na adresu s maskou.



Obrázek 4.2: Architektura aplikační vrstvy - virtuální síť

#### 4.2.2 Vnitřní reprezentace

Přes tyto rozdíly jsem se rozhodl vytvořit pro IP adresu jednu třídu, která má parametry **adresa**, **maska** a **port**, přičemž pro danou situaci nepotřebné parametry prostě ignoruji. Protože Java neobsahuje žádný 32-bitový bezeznaménkový datový typ, jsou parametry adresa a maska vnitřně reprezentovány 32-bitovým integerem, který ale obsahuje bity skutečné adresy, jeho číselná hodnota není důležitá. Operace s nimi se provádí především pomocí bitových operátorů. Parametr port je normální integer.

#### 4.2.3 Veřejné metody

**IpAdresa** má konstruktory, aby ji bylo možné vytvořit ze **Stringu**, s maskou zadanou jako **String**, **Integer** nebo v jednu řetězci s adresou. Adresu je možné převést na **String** nebo porovnat s jinou adresou mnoha různými způsoby, například jen podle adresy, adresy s portem, adresy s maskou nebo čísla sítě. **IpAdresa** umí vrátit své číslo sítě nebo broadcast jako jinou **IpAdresu**. O těchto metodách se zde nerozepisuji podrobně, v kódu jsou dobře okomentované.

### 4.3 Virtuální počítač

Virtuální počítač je základním stavebním prvkem naší aplikace. Pracuje na obou jejích vrstvách. Na vrstvě komunikační přijímá a zpracovává příchozí spojení, na aplikační vrstvě,

tj. na vrstvě virtuální sítě přijímá, posílá a přeposílá pakety. Posíláním paketů se zabývám až v posledním odstavci této kapitoly, zde proberu komunikační vrstvu počítače a jeho rozhraní.

Protože linuxový a ciscový počítač, který dělal kolega, mají mnoho společného, vytvořil jsem abstraktní třídu **AbstraktniPocitac**, který je předkem počítačů obou typů. Třída **LinuxPocitac** má ale jen jednu metodu, která se týká posílání paketů, se jí zatím nezabývám.

Všechny virtuální počítače jsou vytvářeny v rámci inicialisace aplikace dle konfiguračního souboru na začátku jejího běhu, za chodu aplikace není již možné další počítač přidat nebo nějaký odebrat. Pro komunikaci s uživatelem má každý počítač vlastní objekt třídy **Komunikace**. Počítač si drží seznam svých síťových rozhraní, svoji routovací tabulku a natovací tabulku. Má jediný konstruktory, kde je mu zadáno jméno (pro přehlednost) a port, na kterém má být dostupný pro uživatele.

### 4.3.1 Síťové rozhraní

Z hlediska infrastruktury sítě jsou základními prvky počítače jeho síťová rozhraní. Ty si počítač drží v seznamu. Jsou vytvořeny při parsování konfiguračního souboru a během běhu aplikace je nelze nijak měnit, přidávat nebo mazat. Třída **SitoveRozhrani** má svoje jméno a fyzickou (mac) adresu. Protože v naší aplikaci není implementován ARP<sup>1</sup> protokol, mac adresa nemá jiný význam, než že je vypisována příkazy jako např. `ifconfig`.

Skutečné síťové rozhraní může mít více adres. Tato možnost však není v předmětu PSI využívána, proto jsem ji neimplementoval. Znamenalo by to totiž poměrně velké problémy v posílání paketů. Musel bych složitě zjišťovat, kdy se paket odešle s jakou síťovou adresou, pokud je jich na daném rozhraní více. Pro potřeby statického překladu adres (NAT) především na ciscovém routeru je ale nutné mít na rozhraní více adres<sup>2</sup>. Proto má třída **SitoveRozhrani** seznam IP adres, ale jeho první adresa je privilegovaná. Každý paket, který je přes dané rozhraní posílán, má jako odchozí adresu adresu právě první tohoto rozhraní. První adresa je vždy nastave, není-li nakonfigurována, je nastavena na `null`. Tuto jedinou adresu lze nastavovat a vypisovat. Ostatní adresy jsou přidávány jen pro potřeby statického natování. Pokud rozhraní nemá nastavenou žádnou adresu, je první (privilegovaná) adresa `null`.

## 4.4 Infrastruktura virtuální sítě

Poté, co jsem popsal implementaci virtuálního počítače, můžu popsat vnitřní reprezentaci infrastruktury virtuální sítě. Ta vychází z toho, že v simulátoru neuvažuji směrovače na linkové vrstvě ISO/OSI modelu (switche). To totiž znamená, že jedno síťové rozhraní počítače může být připojeno nejvýše k jednomu jinému síťovému rozhraní nějakého počítače. Infrastruktura takové sítě je tak jednoznačně určena dvojicemi síťových rozhraní, které jsou mezi sebou propojeny kabelem. Třída **SitoveRozhrani** má proto parametr `pripojenoK`, který obsahuje

---

<sup>1</sup> Address Resolution Protocol se v počítačových sítích s IP protokolem používá k získání ethernetové MAC adresy sousedního stroje z jeho IP adresy. Používá se v situaci, kdy je třeba odeslat IP datagram na adresu ležící ve stejné podsíti jako odesílatel. Data se tedy mají poslat přímo adresátovi, u něhož však odesílatel zná pouze IP adresu. Pro odeslání prostřednictvím např. Ethernetu ale potřebuje znát cílovou ethernetovou adresu.[? ]

<sup>2</sup> Více o překladu adres v bakalářské práci mého kolegy Stanislava Řeháka

odkaz na jiné síťové rozhraní, ke kterému je připojeno. Není-li rozhraní připojeno, je tento parametr nastaven na `null`. Tato infrastruktura je samozřejmě načítána z konfiguračního souboru. Při vytváření infrastruktury aplikace ohlídá, aby rozhraní byla spojena obousměrně a správně. To jest, je-li rozhraní A připojeno k rozhraní B, musí být také rozhraní B připojeno k rozhraní A.

## 4.5 Routovací tabulka

Počítače směrují pakety podle tzv. routovací, neboli směrovací, tabulky. „Routovací tabulka je datový soubor uložený v RAM paměti, který je používán k uchovávání informací ohledně přímo připojených i vzdáleně připojených sítí. Její obsah napovídá routeru, kterým rozhraním je možno neoptimálněji dosáhnout cílové sítě.“[? ]. V této části se zabývám nejprve analýzou routovací tabulky na skutečném linuxu a potom popisují její implementaci v simulátoru.

Třída `RoutovacíTabulka` měla být původně stejně použitelná pro linux i pro cisco. Až po tom, co jsem jí implementoval, kolega zjistil, že pro potřeby Cisca není tato třída bez úprav použitelná. Proto implementoval třídu `CiscoWrapper`, která obaluje třídu `RoutovacíTabulka` a dodává jí funkce potřebné pro cisco. To však není obsahem mé práce.

### 4.5.1 Analýza routovací tabulky na skutečném počítači

#### 4.5.1.1 Struktura tabulky

V řádcích routovací tabulky jsou záznamy pro jednotlivé sítě. Každý záznam má tyto parametry:

- adresát - IP adresa s maskou, pro kterou je tento záznam platný
- brána - IP adresa počítače, na který se má paket poslat. Tento sloupec nemusí být vždy vyplněn.
- příznaky - O těch více píšu v samostatné části.
- metrika - Jedno z kritérií priority.
- rozhraní - Rozhraní, přes které se paket posílá.

Parametr metrika není pro výukové účely potřeba, proto se jím již dále nezabývám.

Pro lepší představu zde vkládám routovací tabulku tak, jak je vypsána příkazem `route -n`:

Adresát	Brána	Maska	Přízn	Metrik	Odkaz	Užt	Rozhraní
147.32.125.128	0.0.0.0	255.255.255.128	U	1	0	0	eth0
169.254.0.0	0.0.0.0	255.255.0.0	U	1000	0	0	eth0
0.0.0.0	147.32.125.129	0.0.0.0	UG	0	0	0	eth0

#### 4.5.1.2 Adresát

V hořejším výpisu tabulky pomocí příkazu **route** se adresáta týkají 2 sloupce, sloupec Adresát a sloupec Masky, ve kterém jsou vypsány masky k IP adresám uvedeným ve sloupci Adresát. Tyto IP adresy s maskami jsou vždy číslem sítě a reprezentují všechny adresy, které do této sítě patří. Tak například adresát 0.0.0.0/0 reprezentuje úplně všechny IP adresy, adresát 147.32.125.128/25 reprezentuje adresy v rozmezí 147.32.125.128 až 147.32.125.255, adresát 1.1.1.1/32 reprezentuje jedinou adresu 1.1.1.1 a adresát 192.168.1.0/24 reprezentuje všechny adresy, které začínají byty 192.168.1.x. Adresáta 147.32.125.128/24 nelze zadat, protože číslo této sítě je 147.32.125.0/24.

#### 4.5.1.3 Příznaky

Záznam routovací tabulky má několik příznaků. Má vždy minimálně jeden příznak, může mít ale všechny 3 příznaky najednou. Zde je jejich popis:

- Příznak **U** znamená, že záznam obsahuje rozhraní. Protože záznam bez vyplněného rozhraní není možné zadat, má tento příznak každý záznam.
- Záznam má příznak **G**, jestliže je vyplněn sloupec brána.
- Příznak **H** znamená, že adresátem daného záznamu je jeden počítač, tzn. adresát má masku 255.255.255.255.

Příznak **H** jen informuje, že adresát není sítí ale jediným počítačem, není tedy nijak důležitý. Podle příznaků existují 2 typy záznamů, záznamy s příznakem **U** a záznamy s příznakem **UG**. Tyto typy se liší jak při přidávání nových záznamů do routovací tabulky, tak při posílání paketu podle tohoto záznamu. Posílá-li počítač paket podle záznamu **U**, není z tohoto záznamu zřejmé, jakému sousednímu počítači (na linkové vrstvě) se má paket poslat. Počítač se tedy pokusí poslat paket přímo na cílovou IP adresu uvedenou v paketu. Posílá-li se paket podle záznamu **UG**, posílá se na adresu brány uvedenou v záznamu. Více se této problematice věnuji v kapitole o posílání paketů.

#### 4.5.1.4 Přidávání záznamů a jejich řazení

Routovací tabulka nesmí obsahovat 2 stejné záznamy. Za stejné záznamy se považují záznamy, které mají stejného adresáta včetně masky, stejné rozhraní a stejnou bránu.

Záznam typu **U** lze přidat vždycky. Záznam typu **UG** lze přidat jen pod podmínkou, že jeho brána je v okamžiku přidání dosažitelná záznamem typu **U**. Tím je možné dosáhnout zajímavého chování: Když do routovací tabulky přidám defaultní routu <sup>3</sup> záznamu typu **U**, můžu pak přidat routu na jakoukoliv síť v internetu se záznamem **UG**. Když potom smažu původní defaultní routu, můžu posílat pakety pouze na tu síť s příznakem **UG** a na počítač v mé síti paket neodešlu. Zde uvádím příklad:

<sup>3</sup>Defaultní routa je záznam platný pro celý internet, jeho adresátem je 0.0.0.0/0



```

root: /home/neiss# route add default eth0
root: /home/neiss# route add -net 89.190.94.0/24 gw 89.190.94.1
root: /home/neiss# route del default
root: /home/neiss# route
Směrovací tabulka v jádru pro IP
Adresát      Brána      Maska      Přízn Metrik Odkaz  Užt Rozhraní
89.190.94.0   89.190.94.1 255.255.255.0 UG    0      0      0 eth0
root: /home/neiss# ping -c1 89.190.94.58
PING 89.190.94.58 (89.190.94.58) 56(84) bytes of data.
64 bytes from 89.190.94.58: icmp_seq=1 ttl=53 time=14.1 ms

--- 89.190.94.58 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 14.141/14.141/14.141/0.000 ms
root: /home/neiss# ping -c1 147.32.125.129 # toto je adresa brány, přes kterou šel minul
connect: Network is unreachable

```

Tento pokus funguje ale jen tehdy, pokud moje brána, v tomto případě 147.32.125.129 je cisco (viz část o posílání paketů).

Záznamy se v tabulce řadí podle masky adresáta. Nahoře jsou záznamy s nejdelší maskou, tzn. záznamy nejkonkrétnější. Pokud vkládám více záznamů se stejnou maskou, chová se routovací tabulka naprosto nepředvídatelně, což je vidět na následujícím příkladě:

```

node-4:/home/dsn# route add -net 1.1.5.0/25 dev eth0
node-4:/home/dsn# route add -net 1.1.6.0/25 dev eth0
node-4:/home/dsn# route add -net 1.1.7.0/25 dev eth0
node-4:/home/dsn# route add -net 1.1.8.0/25 dev eth0
node-4:/home/dsn# route add -net 1.1.9.0/25 dev eth0
node-4:/home/dsn# route add -net 1.1.10.0/25 dev eth0
node-4:/home/dsn# route add -net 1.1.11.0/25 dev eth0
node-4:/home/dsn# route add -net 1.1.12.0/25 dev eth0
node-4:/home/dsn# route add -net 1.1.13.0/25 dev eth0
node-4:/home/dsn# route add -net 1.1.14.0/25 dev eth0
node-4:/home/dsn# route add -net 1.1.15.0/25 dev eth0
node-4:/home/dsn# route
Kernel IP routing table
Destination      Gateway          Genmask         Flags Metric Ref    Use Iface
1.1.10.0         *               255.255.255.128 U        0      0      0 eth0
1.1.11.0         *               255.255.255.128 U        0      0      0 eth0
1.1.8.0          *               255.255.255.128 U        0      0      0 eth0
1.1.9.0          *               255.255.255.128 U        0      0      0 eth0
1.1.14.0         *               255.255.255.128 U        0      0      0 eth0
1.1.6.0          *               255.255.255.128 U        0      0      0 eth0
1.1.15.0         *               255.255.255.128 U        0      0      0 eth0
1.1.7.0          *               255.255.255.128 U        0      0      0 eth0
1.1.12.0         *               255.255.255.128 U        0      0      0 eth0

```

1.1.13.0	*	255.255.255.128	U	0	0	0 eth0
1.1.5.0	*	255.255.255.128	U	0	0	0 eth0

Podle jakého algoritmu jsou nové záznamy zařazovány je mi opravdu záhadou.

#### 4.5.1.5 Mazání záznamů

Smazat je možno jakýkoliv záznam tabulky. Pro mazání záznamů je potřeba zadat správně minimálně adresáta záznamu. Pokud pak existuje více záznamů se zadanými parametry, smaže se první z nich. Smazání jakéhokoliv záznamu nijak neovlivní ostatní záznamy.

#### 4.5.1.6 Použití pro směrování

Ke směrování paketu se použije první záznam odpovídající cílové adrese. To znamená, že bude-li v routovací tabulce dané adrese odpovídat více záznamů, použije se ten nejvíce nahoře. Protože záznamy jsou řazeny podle délky síťové masky, je vrácený záznam ten nejkonkrétnější.

### 4.5.2 Implementace routovací tabulky v simulátoru

Routovací tabulka je implementována třídou `RoutovaciTabulka`, jejíž odkaz si drží `AbstraktniPocitac` a podle ní směruje pakety.

#### 4.5.2.1 Vnitřní reprezentace

Tabulka je vnitřně reprezentována seznamem objektů typu `Zaznam`, který reprezentuje jeden záznam, tj. řádek tabulky. Záznam routovací tabulky má v simulátoru jen tyto parametry: adresát, brána a rozhraní, které fungují tak, jak bylo popsáno v odstavci o analýze. Parametry adresát a brána jsou typu `IpAdresa`, parametr rozhraní je typu `SitoveRozhrani`. Parametr záznamy není vůbec potřeba. Příznak `U` musí mít záznam vždy, příznak `H` má právě tehdy, když adresát má masku `255.255.255.255`, a příznak `U` má záznam právě tehdy, když má vyplněnou položku brána, proto ani ten není potřeba.

#### 4.5.2.2 Přidávání, mazání a řazení záznamů

Záznamy jsou přidávány pomocí 2 metod se stejným názvem `pridejZaznam` ale jinými parametry. Jedna přidává záznam typu `U`, druhá, která má navíc parametr brána, záznam typu `UG`. U obou se kontroluje, jestli tabulka již stejný záznam neobsahuje, u té druhé se navíc kontroluje dosažitelnost brány, jak bylo popsáno v analýze.

Záznamy se samozřejmě řadí podle masky jako v reálné tabulce, záznamy se stejnou maskou se ale vloží vždy nad původní záznam. Tak jsou novější záznamy vždy nahoře. Zmatečné řazení reálné tabulky jsem samozřejmě neimplementoval.

Pro mazání má `RoutovaciTabulka` metodu `SmazZaznam`, funguje stejně jako na reálném počítači.

Navíc obsahuje `RoutovaciTabulka` ještě metodu `pridejZaznamBezKontrol`, která je využívána při vytváření počítače z konfiguračního souboru, jinde se nepoužívá.

### 4.5.2.3 Použití při směrování

K samotnému směrování slouží metoda `najdiSpravnejZaznam`, která vrací celý řádek routovací tabulky. Funguje stejně jako na reálném počítači.

## 4.6 Posílání paketů

Posílání paketů v naší aplikaci slouží k tomu, aby uživatel pomocí příkazů `ping` a `traceroute` mohl ověřit, zda síť správně nakonfiguroval. Bez této části by naší aplikaci nebylo možné nazvat síťovým simulátorem.

### 4.6.1 Teoretický rozbor referenčního modelu ISO/OSI

„Referenční model ISO/OSI vypracovala organizace ISO jako hlavní část snahy o standardizaci počítačových sítí.“[?] . Dle tohoto modelu probíhá síťová komunikace v sedmi vrstvách, z níž každá poskytuje přesně definované funkce a komunikuje jen s vrstvou sousední. Každá vrstva má svůj formát přenášených dat, obvykle dělených do bloků. Pro naši aplikaci jsou důležité vrstvy 2 - 4, tzn. spojová, síťová a transportní vrstva.

#### 4.6.1.1 Spojová vrstva

Spojová nebo linková vrstva<sup>4</sup> „poskytuje spojení mezi dvěma sousedními systémy.“[?] . Tuto vrstvu zajišťuje na skutečné síti v laboratoři technologie Ethernet[?] , ke zjištění fyzické adresy sousedního systému se používá protokol ARP[?] . Souslední systémy se zde rozumí počítače zapojené do stejné sítě. Blok dat na linkové vrstvě se nazývá rámec. Vzhledem k tomu, že simulátor neobsahuje žádné switche, zajišťuje tato vrstva v naší aplikaci spojení mezi 2 počítači propojenými kabelem.

#### 4.6.1.2 Síťová vrstva

Síťová vrstva „se stará o směrování v síti a síťové adresování. Poskytuje spojení mezi systémy, které spolu přímo nesousedí.“[?] . Zajišťuje spojení mezi jakýmkoliv dvěma uzly sítě. Obvykle je realizována protokolem IP<sup>5</sup>. Blok dat na síťové vrstvě se nazývá paket.

#### 4.6.1.3 Transportní vrstva

Transportní vrstva „zajišťuje přenos dat mezi koncovými uzly“[?] . Pro potřeby příkazů `ping` a `traceroute` je tato vrstva realizována protokolem ICMP<sup>6</sup>. Blok dat se v transportní vrstvě nazývá datagram.

---

<sup>4</sup>Více v [?]

<sup>5</sup>Internet Protocol

<sup>6</sup>Internet Control Message Protocol, více v [?]

#### 4.6.1.4 Datové bloky

Datový blok jakékoliv vrstvy se skládá z hlavičky, která obsahuje režijní informace té vrstvy, a z datové části, která obsahuje samotná data, ale i hlavičky vyšších vrstev. Tak například datagram protokolu ICMP je obalen hlavičkou IP na síťové vrstvě a hlavičkou protokolu Ethernet na spojové vrstvě.

#### 4.6.2 Implementace třídy **Paket**

Pro můj síťový simulátor by bylo nesmyslné implementovat posílání paketů včetně jejich zabalování do datových bloků různých vrstev, tak jak je to popsáno v předchozím odstavci 4.6.1.4. Proto jsem vytvořil třídu **Paket**, která obsahuje všechny potřebné informace z datových bloků všech tří vrstev. **Paket** má parametry síťové vrstvy, jako zdrojovou a cílovou adresu a **ttl**, a parametry transportní vrstvy, jako typ a kód protokolu ICMP. Pro snadnější implementaci příkazu ping má parametr **cas**, kam se ukládá náhodně generovaný čas běhu paketu, který vypisuje příkaz ping. Protože na jednom virtuálním počítači může běžet více příkazů ping najednou, nese paket i odkaz na příkaz, který ho poslal, aby ho tento příkaz mohl také po jeho návratu zpracovat.

Typy a kódy ICMP paketů jsou označeny stejně jako ve skutečnosti:

- typ 0 - ozvěna (icmp reply) - odpověď na požadavek icmp request
- typ 3 - vyslaný paket nemohl být doručen
- typ 8 - žádost o ozvěnu (icmp request)
  - kód 0 - network unreachable (nedosažitelná síť)
  - kód 1 - host unreachable (nedosažitelná adresa)
- typ 11 - ttl vypršelo

#### 4.6.3 Chování reálného počítače při posílání paketů

##### 4.6.3.1 Transportní vrstva - protokol ICMP

V souboru `/proc/sys/net/ipv4/icmp_echo_ignore_all` je nastaveno, jestli počítač odpovídá na dotazy icmp reply. Pokud je v tomto souboru 0, počítač na dotazy odpovídá. Toto je defaultní nastavení, proto jsem v simulátoru tento problém vůbec neřešil a počítač odpovídá na icmp request vždy.

##### 4.6.3.2 Síťová vrstva - protokol IP

###### Přeposílání paketů

Počítač preposílá pakety jenom tehdy, pokud je v souboru `/proc/sys/net/ipv4/ip_forward` jednička, jinak ne. Toto nastavení ale již není defaultní, proto ho musím v simulátoru implementovat. Proměnná `ip_forward` je parametrem virtuálního počítače a je nastavována pomocí příkazu `echo`.

### Směrování paketů

Pakety jsou směrovány podle routovací tabulky, kde se vybere první záznam odpovídající cílové adrese paketu, jak je popsáno v 4.5.1.6. Pokud v routovací tabulce nebyl nalezen žádný záznam pro cílovou adresu paketu, pošle se na jeho zdrojovou adresu paket `icmp_net_unreachable`<sup>7</sup>.

#### `ttl`

Každý prvek, který pracuje i na síťové vrstvě sníží procházejícím paketům hodnotu `ttl`. Pokud po tomto snížení dosáhne hodnota `ttl` nuly, pošle počítač na zdrojovou adresu paketu zprávu `icmp_ttl_exceeded`<sup>8</sup>.

#### Next hop

Předtím než síťová vrstva předá odeslání paketu linkové vrstvě, musí pro tento paket zjistit tzv. next hop, neboli sousední adresu. „Next hop je sousední směrovač, na který je paket poslán nebo přeposlán z daného směrovače na své cestě k cíli.“[?] Tato adresa je velmi důležitá pro linkovou vrstvu, která paket posílá právě na tuto adresu. Pokud je paket směrován podle záznamu typu UG (viz 4.5.1.3), je adresa next hop uvedena ve sloupci brána routovací tabulky. Pokud je paket směrován podle záznamu typu U, je adresa next hop cílová adresa paketu. Podle routovací tabulky

Adresát	Brána	Maska	Přízn	Metrika	Rozhraní
147.32.125.128	0.0.0.0	255.255.255.128	U	0	eth0
0.0.0.0	147.32.125.129	0.0.0.0	UG	0	eth0

je pro pakety směrované podle prvního záznamu next hop rovná jejich cílové adrese. Pro pakety směrované podle druhého záznamu je next hop 147.32.125.128. Záznamy typu U jsou tak používány pro počítače v mé síti, které jsou dosažitelné přímo bez jakéhokoliv mezilehlého směrovače. Záznamy UG jsou používány pro počítače, na které je paket poslán přes jeden nebo více směrovačů.

#### 4.6.3.3 Linková vrstva

Na linkové vrstvě se rámce přeposílají jen mezi sousedními počítači. Běžně ji zajišťuje protokol ethernet. IP adresu sousedního počítače, na kterou má rámec poslat, next hop, dostane od síťové vrstvy, musí ji přeložit na fyzickou (MAC<sup>9</sup>) adresu, což dělá protokolem ARP. Protokol ARP vyšle ethernetový rámec na všechny okolní počítače žádost, která obsahuje zadanou IP adresu. Pokud některý z počítačů má tokovou IP adresu, původnímu počítači pošle zpátky svoji fyzickou adresu a ten pak může odeslat paket. Aby počítač nemusel fyzickou adresu zjišťovat při odesílání každého rámce, ukládá si v datové struktuře nazvané ARP tabulka záznamy s IP adresami, které již dříve překládal.

Pokud počítač nemůže linkovou vrstvou odeslat rámec (paket), například proto, že počítač s takovou adresou na síti neexistuje, pošle původnímu odesílateli ICMP paket `net_unreachable`<sup>10</sup>.

<sup>7</sup>Tj ICMP paket typu 3 kódu 0

<sup>8</sup>Tj. icmp paket typu 11

<sup>9</sup>Media Access Control

<sup>10</sup>Tj. ICMP paket typu 3 kódu 1

Počítač s operačním systémem linux odpovídá na ARP dotazy jen tehdy, když má požadovanou IP adresu nastavenou na svém rozhraní. V tom se liší od Cisco, které na ARP dotaz odpoví i v případě, že adresu na svém rozhraní nemá, ale ví, kam má paket dále směřovat, tzn. má pro požadovanou adresu záznam v routovací tabulce, a naopak na ARP dotaz neodpoví v případě, že nemá v routovací tabulce záznam pro IP adresu, odkud dotaz přišel. Pro lepší pochopení přepisuji podmínku ještě v jazyce booleanovských výrazů v javě:

Cisco odpoví na ARP dotaz právě tehdy, když:  
Má záznam v routovací tabulce pro počítač, který ARP dotaz posílá **&&** (Má nastavenou požadovanou IP adresu **nebo** Má záznam v routovací tabulce pro cílovou adresu posílaného paketu)

#### 4.6.3.4 Zajímavá zjištění

Při analýze chování linuxového počítače v síťové komunikaci jsem zjistil několik zajímavých a aspoň pro mě překvapujících faktů. Například 2 počítače v síti spolu mohou komunikovat i tehdy, mají-li nastaveny IP adresy z úplně jiných sítí. Stačí totiž, mají-li v routovací tabulce jeden na druhého záznam. Počítače z obrázku 4.3 spolu opravdu komunikují, jak je vidět na následujícím výpisu.



Obrázek 4.3: Funkční síť se „špatnými“ adresami

```

node-1:/home/dsn# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:40:F4:B7:F0:32
          inet addr:192.168.1.1  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::240:f4ff:feb7:f032/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:43 errors:0 dropped:0 overruns:0 frame:0

node-1:/home/dsn# ping 10.0.0.1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=0.942 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.208 ms

--- 10.0.0.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1003ms

```

rtt min/avg/max/mdev = 0.191/0.447/0.942/0.350 ms

#### 4.6.4 Implementace v simulátoru

Pakety jsou mezi počítači posílány vzájemným voláním metod, které si mezi sebou předávají objekt typu `Paket`. Všechny tyto metody jsou ve třídě `AbstraktniPocitac`, protože to jsou právě virtuální počítače, které si mezi sebou pakety posílají. Jen metoda `prijmiEthernetove` je sice v `AbstraktniPocitac` zadeklarována a implementována v jeho potomcích.

Metody virtuálního počítače pro posílání paketů jsem bylo rozumné implementovat dle vrstev. Je to přehlednější, než kdybych měl jednu metodu pro přijetí paketu a je to dobré i vzhledem k tomu, že linux a cisco se v některých případech liší a to jen na některých vrstvách. Metody jedné vrstvy tak volají jen metody vrstvy sousední, jako na reálné síti jedna vrstva komunikuje jen s vrstvami sousedními.

##### 4.6.4.1 Linková vrstva

ARP protokol na zjišťování fyzických adres nemusel být implementován, protože v síti nejsou žádné switche a v linkové vrstvě tak není potřeba žádné směrování. Aby ale byly splněny podmínky doručení nebo nedoručení paketu uvedené v 4.6.3.3, byly vytvořeny metody `odesliEthernetove` a `prijmiEthernetove`, které si mezi sebou předávají pakety tak, aby byly tyto podmínky splněny. Metoda `odesliEthernetove` je volána nějakou metodou síťové vrstvy. Pokouší se odeslat paket tak, že zavolá metodu `prijmiEthernetove` nějakého jiného počítače. Metoda `prijmiEthernetove` na linuxu přijme paket jen tehdy, pokud souhlasí očekávaná adresa, tj. adresa next hop odesílacího počítače. Přijme tedy paket právě tehdy, když by skutečný počítač odpověděl na ARP dotaz. Pokud se metodě `odesliEthernetove` nepovede paket odeslat proto, že ho metoda `prijmiEthernetove` odmítla, pošle odesílateli pomocí metody `posliNovejPaketOdpoved` zprávu `icmp host unreachable`. Pokud metoda `prijmiEthernetove` paket přijme, zavolá metodu `prijmiPaket` síťové vrstvy.

##### 4.6.4.2 Síťová vrstva

Na síťové vrstvě si pakety předávají metody `odesliNovejPaket`, `preposliPaket` a `prijmiPaket`. Tyto metody směrují pakety dle routovací tabulky a provádí překlad adres podle natovací tabulky. Všechny pakety přijímá metoda `prijmiPaket`, která se nejdříve pokusí přeložit cílové adresy paketů. Pak rozhodne, je-li přijatý paket na tom počítači v cíli, nebo jestli se má dále přeposlat a případně zavolá metodu `preposliPaket` nebo paket nějak zpracuje, například odpoví na `icmp request`. Metoda `preposliPaket` přeposílá pakety, když má nastaveno `ip_forward`, a přeposílaným paketům snižuje ttl. Při odeslání se pokouší přeložit zdrojovou adresu paketu. Metoda `odesliNovejPaket` slouží k odesílání všech nových paketů vytvořených na tom počítači.

##### 4.6.4.3 Transportní vrstva

Do transportní vrstvy patří více metod, které slouží k posílání různých typů ICMP paketů. Například `posliIcmpRequest`, `posliNetUnreachable` a další. Všechny tyto metody volají

metodu `odesliNovejPaket` ze síťové vrstvy.



## Kapitola 5

# Implementace příkazů

V této kapitole popisuji analýzu a realizaci linuxových příkazů, které jsou v simulátoru implementovány. Příkazy jsou nejrozsáhlejší částí mé práce na simulátoru a nutno dodat, že také tou nejméně zábavnou. U každého příkazu jsem nejprve musel složitě zjišťovat, jak funguje a jak se chová v případě zadání nestandardních parametrů. Parsování příkazů bylo prací pořád stejnou a nezábavnou.

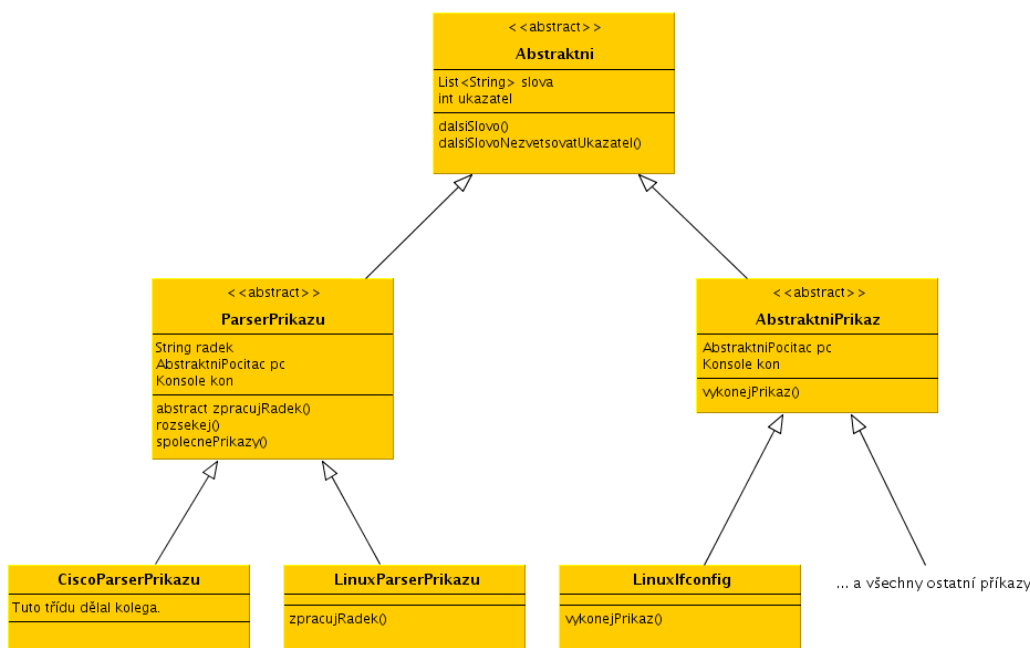
### 5.1 Popis zpracování příkazové řádky

Příkazovou řádku virtuálního počítače obsluhuje třída `Konsole` (více v 4.1.1). Ta načítá textová data posílaná uživatelem až do sekvence `\r\n`, kdy načítání zastaví a načtený řádek předá svému parseru. Každá `Konsole` má svůj vlastní parser příkazů, který zpracovává jen jí poslané řádky a na ně odpovídá. `Konsole` zavolá jeho metodu `zpracujRadek`, která řádek „rozseká“ na jednotlivá slova a pak podle prvního slova, které by mělo být názvem příkazu, spustí odpovídající příkaz. Příkazům `Konsole` poskytuje metody `posliRadek` a `posli`, pomocí kterých příkazy vypisují svůj výstup uživateli do konzole.

Architektura tříd pro zpracování příkazové řádky je celkem složitá, jak je vidět na obrázku 5.1. Všechny třídy mají jednoho předka, kterého jsme z nedostatku fantazie pojmenovali `Abstraktni`. Každý příkaz má svoji vlastní třídu, kterou `ParserPrikazu` vytvoří pro každé použití příkazu novou, jde tedy o použití návrhového vzoru singleton. V následujících odstavcích popisují funkce jednotlivých tříd.

#### 5.1.1 Třída `Abstraktni`

Tato třída je abstraktním předkem všech ostatních tříd sloužících ke zpracování příkazů. Obsahuje pro parsování velmi důležitou metodu `dalsiSlovo` a seznam slov jednoho řádku. Tato metoda postupně vrací jednotlivá slova ze seznamu. Je využívána skoro všemi potomky třídy `Abstraktni` k parsování řádky. Dále jsou v této třídě různé statické metody využívané jednotlivými příkazy, například metoda `zarovnej` na zarovnání slova v řetězci, metoda `cekej`, která čeká zadaný počet milisekund a další.



Obrázek 5.1: Třídy pro zpracování příkazové řádky

### 5.1.2 Třída ParserPrikazu

Instanci jednoho z potomků této třídy, tedy `LinuxParsePrikazu` nebo `CiscoParsePrikazu`, si vytváří při své inicializaci třída `Konsole`, které slouží ke zpracování příkazové řádky. Jeho hlavní metodou je abstraktní `zpracujRadek`, která je implementována v jeho potomcích a zpracovává jeden řádek, který uživatel do konzole napsal. K tomu používá metodu `rozsekej`, jejímž autorem je můj kolega, která řádek zadaný jako `String` „rozseká“ na jednotlivá slova a to tak, že jako oddělovače bere bílé znaky, tj. mezery a tabulátory. Rozsekaná slova pak uloží do seznamu řetězců `slova`, který zdédila od svého předka třídy `Abstraktni`. Třída `LinuxParsePrikazu` podle prvního slova, které by mělo být názvem příkazu, vyvolá konstruktor třídy odpovídajícího příkazu. Nejdříve kontroluje, jestli příkaz není společným příkazem pro linuxový i ciscový počítač (příkaz `uloz` nebo `save`) a poté se snaží najít odpovídající linuxový příkaz. Jestliže uživatelem zadaný příkaz neexistuje nebo není implementován, vypíše `LinuxParsePrikazu` uživateli řetězec `bash: command not found` a zpracování řádku ukončí.

### 5.1.3 Třída AbstraktniPrikaz

Toto třída je potomkem třídy `Abstraktni` a předkem tříd všech příkazů. Nemá žádnou důležitou metodu, ale parametry `abstraktniPocitac pc` a `konsole kon`, což jsou odkazy na počítač, na kterém příkaz probíhá a na konsoli, která ho vyvolala.

## 5.2 Společné znaky příkazů

Příkazy, které jsem implementoval mají mnoho společného. Všechny jsou potomkem třídy `AbstraktniPrikaz`, který je potomkem třídy `Abstraktni`. Text, který byl uživatelem zadán, mají uložen v seznamu řetězců `slova`. V konstruktoru příkazu je postupně voláno několik privátních metod. Metodou `zparsujPrikaz` se nejprve parsuje vstup, většinou pomocí metody `dalsiSlovo` zděděné ze třídy `Abstraktni`. Při parsování se nastavují různé parametry příkazu, které uživatel zadal. Metoda `zkontrolujPrikaz` kontroluje, jestli jsou nastaveny správné parametry a jestli byli zadány všechny. Potom metoda `vykonejPrikaz` příkaz vykoná, to znamená, že změní konfiguraci počítače například u příkazu `ifconfig`, nebo že provede nějakou jinou akci, například u příkazu `ping` pošle několik paketů. Během těchto tří metod jsou ukládány případné chyby zadaného příkazu. Úplně nakonec příkaz vypíše případné chybové hlášení. Tento postup byl zvolen proto, že sladit správné pořadí vypisovaných chybových hlášení s postupem kontroly jednotlivých parametrů by bylo velmi náročné a nepřehledné. Příkazy svůj výstup posílají pomocí metod `posli` a `posliRade` třídy `Konsole`.

## 5.3 Příkaz ifconfig

### 5.3.1 Teoretický úvod

`ifconfig` (zkratka interface **co**nfigurator) je utilita unixových systémů, která slouží ke konfiguraci, kontrole a vypsání informací o parametrech síťových rozhraní z příkazové řádky.[?] . Pomocí této utility lze například vypsat parametry síťových rozhraní, nastavit IP nebo mac adresy, nastavit masky nebo „nahodit“ nebo „schodit“ rozhraní. Kromě toho, že `ifconfig` nastavuje rozhraní, přidává nebo maže i záznamy v routovací tabulce, je-li to potřeba.

### 5.3.2 Rozsah implementace v simulátoru

V simulátoru musí `ifconfig` umět nastavovat a mazat adresy IP a nastavovat masky. Samozřejmě také musí vypisovat nastavené parametry rozhraní. Změna mac adresy naproti tomu v našem simulátoru není vůbec potřeba. Parser příkazu `ifconfig` je ale implementován i pro parametry `add`, `del`, `up` a `down`.

Zde uvádím příklady užití `ifconfig`, které simulátor podporuje:

```
ifconfig eth0 vypíše parametry rozhraní eth0
ifconfig eth0 192.168.1.1 nastaví adresu na rozhraní eth0
ifconfig eth0 10.0.0.1/8 nastaví adresu s maskou na rozhraní eth0
ifconfig eth0 10.0.0.2 netmask 255.255.0.0 nastaví adresu s maskou na rozhraní eth0
```

### 5.3.3 Analýza ifconfigu na skutečném počítači

Oproti jiným příkazům má `ifconfig` pro simulaci jednu obrovskou nevýhodu: chybný parametr nezpůsobí ukončení příkazu, ale příkaz se dále provádí. Některé parametry je možné zadat vícekrát. Zjišťovat, jak se `ifconfig` chová, proto bylo mnohem náročnější, než u ostatních příkazů. Zde popisují výsledky této práce.

IP adresu je možné příkazu zadat vícekrát, všechno, co nelze zparsovat jinak se považuje za IP adresu. Je-li zadáno více adres, použije se adresa, které předchází první špatně adrese. Například:

```
ifconfig eth0 1.1.1.1 2.2.2.2 3.3.3.3 nastaví se poslední adresa, tj. 3.3.3.3
```

```
ifconfig eth0 1.1.1.1 blablabla 3.3.3.3 nastaví se adresa před první špatnou adresou, tj. 1.1.1.1
blablabla: unknown host
ifconfig: '--help' vypíše návod k použití.
```

Adresu je možné zadat i s délkou prefixu za lomítkem. Například: nastaví adresu 10.4.7.1 s maskou 255.255.255.0. Je-li zadáno více adres a u jedné z nich je zadána délka prefixu, nastaví se taková maska, i když se tato adresa nepoužije. Například

```
ifconfig eth0 1.1.1.1/1 2.2.2.2/2 3.3.3.3
```

nastaví adresu 3.3.3.3 s maskou 192.0.0.0, použije se tedy délka prefixu zadaná u druhé adresy. Adresa na rozhraní se maže nastavením adresy na 0.0.0.0.

Maska se nastavuje buď zadáním délky prefixu, jak je popsáno výše, nebo parametrem netmask. Zadá-li uživatel masku oběma způsoby, má přednost délka prefixu před parametrem netmask. Například

```
ifconfig eth0 2.2.2.2/2 netmask 255.255.0.0
```

nastaví masku 192.0.0.0. Při změně IP adresy se automaticky mění i maska a to tak, že se maska dopočítá podle třídy IP adresy. Je-li tedy nejprve zadána maska parametrem netmask a teprve potom IP adresa, parametr netmask nemá žádnou účinnost. Například

```
ifconfig eth0 netmask 255.255.255.0 1.1.1.1
```

nastaví adresu 1.1.1.1 s maskou 255.0.0.0, protože adresa 1.1.1.1 je třídy A.

Příkaz `ifconfig` promítá změny i do routovací tabulky. Změní-li se adresa na rozhraní, všechny záznamy na toto rozhraní se z tabulky vymažou. Při nastavení IP adresy se do routovací tabulky přidá záznam typu U pro síť, ze které je tato adresa. Například při nastavení adresy

```
ifconfig eth0 192.168.1.1/24
```

se do routovací tabulky přidá tento záznam:

Adresát	Brána	Maska	Přízn	Metrik	Odkaz	Užt	Rozhraní
192.168.1.0	0.0.0.0	255.255.255.0	U	0	0	0	eth0

Při zadání chybného vstupu se vykoná jen správná část příkazu a o těch špatných se vypíší chybová hlášení. Ty jsou vypisována v tomto pořadí:

1. špatný přepínač - příkaz se neprovádí
2. špatná adresa - žádná další chybová hlášení se nevypisují
3. špatné rozhraní
4. zakázaná adresa
5. špatné adresy parametrů **add** nebo **del**
6. chyba v gramatice

Ostatními parametry se v tomto odstavci nezabývám, protože v simulátoru nejsou podporovány.

#### 5.3.4 Implementace v simulátoru

V konstruktoru třídy `LinuxIfconfig` jsou popořadě volány tyto metody: `parsujPrikaz`, `zkontrolujPrikaz`, `vypisChybovyHlaseni` a `vykonejPrikaz`. Parsování se v tomto příkaze neprovádí s využitím metody `dalsiSlovo`, což by bylo dobré ještě upravit. Na rozdíl od skutečného `ifconfigu` se parsuje celý řádek, skutečný `ifconfig` parsování zastaví, když dojde ke špatnému přepínači nebo ke špatné adrese. Při parsování se parametry ukládají jako řetězce a ty jsou kontrolovány metodou `zkontrolujPrikaz`. V tomto příkazu jsou chybová hlášení vypisována před již před samotným provedením příkazu a to v pořadí uvedeném v předchozím odstavci. Příkaz `ifconfig` se metodou `vykonejPrikaz` nakonec provede tak, aby to co nejvíce odpovídalo pravidlům uvedeným v 5.3.3. Parametry `add` a `del` jsou v parsetu a kontrole implementovány, ale metoda `vykonejPrikaz` je zatím ignoruje.

#### 5.3.5 Možnosti dalšího vylepšení

Tento příkaz jsem implementoval jako první, kdy jsem s implementací příkazů ještě neměl zkušenosti, proto by se dal ještě v některých věcech vylepšit. Bylo by dobré přepsat metodu `parsujPrikaz` tak, aby využívala metodu `dalsiSlovo`, čímž by se zjednodušila, a dále ji předělat tak, aby se po špatné adrese již dále nic neparsovalo. Bylo by též dobré implementovat i parametry `up` a `down` s jejichž podporou se v analýze nepočítalo.

#### 5.3.6 Známé odchylky

Zde uvádím známé odchylky implementace v simulátoru od skutečného příkazu na reálném počítači.

`ifconfig eth0 netmask 255.0.0.0 2.2.2.2/2` ve skutečnosti spadne kvůli prefixu za adresou, v simulátoru to ale projde.

`ifconfig eth0 blabla netmask 255.255.255.0` ve skutečnosti žádnou masku nenastaví, protože přestane parsovat už u `blabla`.

`ifconfig eth0 netmask 255.255.255.0 10.0.0.1` by měl nastavit masku na `255.0.0.0`, v simulátoru se nastaví `255.255.255.0`



## Kapitola 6

# Popis problému, specifikace cíle

- Popis řešeného problému, vymezení cílů DP/BP a požadavků na implementovaný systém.
- Popis struktury DP/BP ve vztahu k vytyčeným cílům.
- Rešeršní zpracování existujících implementací, pokud jsou známy.





## Kapitola 7

# Analýza a návrh řešení

Analýza a návrh implementace (včetně diskuse různých alternativ a volby implementačního prostředí).



## Kapitola 8

# Realizace

Popis implementace/realizace se zaměřením na nestandardní části řešení.



## Kapitola 9

# Testování

- Způsob, průběh a výsledky testování.
- Srovnání s existujícími řešeními, pokud jsou známy.



## Kapitola 10

### Závěr

- Zhodnocení splnění cílů DP/BP a vlastního přínosu práce (při formulaci je třeba vzít v potaz zadání práce).
- Diskuse dalšího možného pokračování práce.





## Dodatek A

# Testování zaplnění stránky a odsazení odstavců

Tato příloha nebude součástí vaší práce. Slouží pouze jako příklad formátování textu.



## Dodatek B

# Pokyny a návody k formátování textu práce

**Tato příloha samozřejmě nebude součástí vaší práce. Slouží pouze jako příklad formátování textu.**

Používat se dají všechny příkazy systému L<sup>A</sup>T<sub>E</sub>X. Existuje velké množství volně přístupné dokumentace, tutoriálů, příruček a dalších materiálů v elektronické podobě. Výchozím bodem, kromě Googlu, může být stránka CSTUG (Czech Tech Users Group) [? ]. Tam najdete odkazy na další materiály. Většinou dostačující a přehledně organizovanou elektronikou dokumentaci najdete například na [? ] nebo [? ].

Existují i různé nadstavby nad systémy T<sub>E</sub>X a L<sup>A</sup>T<sub>E</sub>X, které výrazně usnadní psaní textu zejména začátečníkům. Velmi rozšířený v Linuxovém prostředí je systém Kile.

### B.1 Vkládání obrázků

Obrázky se umísťují do plovoucího prostředí **figure**. Každý obrázek by měl obsahovat **název** (`\caption`) a **návěští** (`\label`). Použití příkazu pro vložení obrázku `\includegraphics` je podmíněno aktivací (načtením) balíku `graphicx` příkazem `\usepackage{graphicx}`.

Budete-li zdrojový text zpracovávat pomocí programu `pdflatex`, očekávají se obrázky s příponou `*.pdf`<sup>1</sup>, použijete-li k formátování `latex`, očekávají se obrázky s příponou `*.eps`.<sup>2</sup>

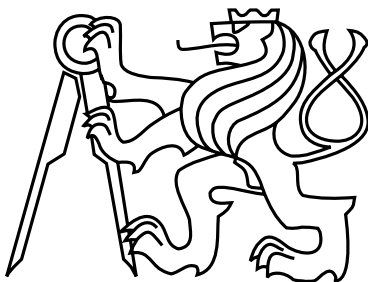
Příklad vložení obrázku:

```
\begin{figure}[h]
\begin{center}
\includegraphics[width=5cm]{figures/LogoCVUT}
\caption{Popiska obrazku}
\label{fig:logo}
```

---

<sup>1</sup>pdflatex umí také formáty PNG a JPG.

<sup>2</sup>Vzájemnou konverzi mezi snad všemi typy obrázku včetně změn velikostí a dalších vymožeností vám může zajistit balík ImageMagick (<http://www.imagemagick.org/script/index.php>). Je dostupný pod Linuxem, Mac OS i MS Windows. Důležité jsou zejména příkazy `convert` a `identify`.



Obrázek B.1: Popiska obrázku

DTD	construction	elimination
	$\text{in1} A B \text{ a:sum } A \ B$ $\text{in1} A B \text{ b:sum } A \ B$	$\text{case}([\_ :A] \ a) ([\_ :B] \ a) \text{ ab:A}$ $\text{case}([\_ :A] \ b) ([\_ :B] \ b) \text{ ba:B}$
+	$\text{do\_reg:A} \rightarrow \text{reg } A$	$\text{undo\_reg:reg } A \rightarrow A$
*, ?	the same like   and + with <code>empty_el:empty</code>	the same like   and + with <code>empty_el:empty</code>
$R(a,b)$	$\text{make\_R:A} \rightarrow \text{B} \rightarrow \text{R}$	$\text{a: R} \rightarrow A$ $\text{b: R} \rightarrow B$

Tabulka B.1: Ukázka tabulky

```
\end{center}
\end{figure}
```

## B.2 Kreslení obrázků

Zřejmě každý z vás má nějaký oblíbený nástroj pro tvorbu obrázků. Jde jen o to, abyste dokázali obrázek uložit v požadovaném formátu nebo jej do něj konvertovat (viz předchozí kapitola). Je zřejmě vhodné kreslit obrázky vektorově. Celkem oblíbený, na ovládání celkem jednoduchý a přitom dostatečně mocný je například program Inkscape.

Zde stojí za to upozornit na kreslicí programe Ipe [? ], který dokáže do obrázku vkládat komentáře přímo v latexovském formátu (vzroce, stejné fonty atd.). Podobné věci umí na Linuxové platformě nástroj Xfig.

Za pozornost ještě stojí schopnost editoru Ipe importovat obrázek (jpg nebo bitmap) a krelit do něj latexovské popisky a komentáře. Výsledek pak umí exportovat přímo do pdf.

## B.3 Tabulky

Existuje více způsobů, jak sázet tabulky. Například je možno použít prostředí `table`, které je velmi podobné prostředí `figure`.

Zdrojový text tabulky B.1 vypadá takto:

```

\begin{table}
\begin{center}
\begin{tabular}{|c|l|l|}
\hline
\textbf{DTD} & \textbf{construction} & \textbf{elimination} \\
\hline
 $\mid$  & \verb+in1|A|B a:sum A B+ & \verb+case([_:A]a)([_:B]a)ab:A+\\
& \verb+in1|A|B b:sum A B+ & \verb+case([_:A]b)([_:B]b)ba:B+\\
\hline
 $\$$  & \verb+do_reg:A -> reg A+ & \verb+undo_reg:reg A -> A+\\
\hline
 $\$,?\$$  & the same like  $\mid$  &  $\$$  & the same like  $\mid$  &  $\$$ \\
& with \verb+empty_el:empty+ & with \verb+empty_el:empty+\\
\hline
R(a,b) & \verb+make_R:A->B->R+ & \verb+a: R -> A+\\
& & \verb+b: R -> B+\\
\hline
\end{tabular}
\end{center}
\caption{Ukázka tabulky}
\label{tab:tab1}
\end{table}
\begin{table}

```

## B.4 Odkazy v textu

### B.4.1 Odkazy na literaturu

Jsou realizovány příkazem `\cite{odkaz}`.

Seznam literatury je dobré zapsat do samostatného souboru a ten pak zpracovat programem bibtex (viz soubor `reference.bib`). Zdrojový soubor pro bibtex vypadá například takto:

```

@Article{Chen01,
  author   = "Yong-Sheng Chen and Yi-Ping Hung and Chiou-Shann Fuh",
  title    = "Fast Block Matching Algorithm Based on
              the Winner-Update Strategy",
  journal  = "IEEE Transactions On Image Processing",
  pages    = "1212--1222",
  volume   = 10,
  number   = 8,
  year     = 2001,
}

@Misc{latexdocweb,

```

```

author = "",
title = "{\LaTeX} --- online manuál",
note = "\verb|http://www.cstug.cz/latex/lm/frames.html|",
year = "",
}
...

```

**Pozor:** Sazba názvů odkazů je dána BibTeX stylem (`\bibliographystyle{abbrv}`). BibTeX tedy obvykle vysází velké pouze počáteční písmeno z názvu zdroje, ostatní písmena zůstanou malá bez ohledu na to, jak je napíšete. Přesněji řečeno, styl může zvolit pro každý typ publikace jiné konverze. Pro časopisecké články třeba výše uvedené, jiné pro monografie (u nich často bývá naopak velikost písmen zachována).

Pokud chcete BibTeXu napovědět, která písmena nechat bez konverzí (viz `title = "{\LaTeX} --- online manuál"` v předchozím příkladu), je nutné příslušné písmeno (zde celé makro) uzavřít do složených závorek. Pro přehlednost je proto vhodné celé parametry uzavírat do uvozovek (`author = "..."`), nikoliv do složených závorek.

Odkazy na literaturu ve zdrojovém textu se pak zapisují:

```

Podívejte se na \cite{Chen01},
další detaily najdete na \cite{latexdocweb}

```

Vazbu mezi soubory `*.tex` a `*.bib` zajistíte příkazem `\bibliography{}` v souboru `*.tex`. V našem případě tedy zdrojový dokument `thesis.tex` obsahuje příkaz `\bibliography{reference}`.

Zpracování zdrojového textu s odkazy se provede postupným voláním programů `pdflatex <soubor>` (případně `latex <soubor>`), `bibtex <soubor>` a opět `pdflatex <soubor>`.<sup>3</sup>

Níže uvedený příklad je převzat z dříve existujících pokynů studentům, kteří dělají svou diplomovou nebo bakalářskou práci v Grafické skupině.<sup>4</sup> Zde se praví:

```

...
j) Seznam literatury a dalších použitých pramenů, odkazy na WWW stránky, ...
Pozor na to, že na veškeré uvedené prameny se musíte v textu práce
odkazovat -- [1].
Pramen, na který neodkazujete, vypadá, že jste ho vlastně nepotřebovali
a je uveden jen do počtu. Příklad citace knihy [1], článku v časopise [2],
statí ve sborníku [3] a html odkazu [4]:
[1] J. Žára, B. Beneš;, and P. Felkel.
    Moderní počítačová grafika. Computer Press s.r.o, Brno, 1 edition, 1998.
    (in Czech).

```

<sup>3</sup>První volání `pdflatex` vytvoří soubor s koncovkou `*.aux`, který je vstupem pro program `bibtex`, pak je potřeba znovu zavolat program `pdflatex (latex)`, který tentokrát zpracuje soubory s příponami `.aux` a `.tex`. Informaci o případných nevyřešených odkazech (cross-reference) vidíte přímo při zpracovávání zdrojového souboru příkazem `pdflatex`. Program `pdflatex (latex)` lze volat vícekrát, pokud stále vidíte nevyřešené závislosti.

<sup>4</sup>Několikrát jsem byl upozorněn, že web s těmito pokyny byl zrušen, proto jej zde přímo necituji. Nicméně příklad sám o sobě dokumentuje obecně přijímaný konsensus ohledně citací v bakalářských a diplomových pracích na KP.

- [2] P. Slavík. Grammars and Rewriting Systems as Models for Graphical User Interfaces. *Cognitive Systems*, 4(4--3):381--399, 1997.
- [3] M. Haindl, Š. Kment, and P. Slavík. Virtual Information Systems. In *WSCG'2000 -- Short communication papers*, pages 22--27, Pilsen, 2000. University of West Bohemia.
- [4] Knihovna grafické skupiny katedry počítačů:  
<http://www.cgg.cvut.cz/Bib/library/>

... abychom výše citované odkazy skutečně našli v (automaticky generovaném) seznamu literatury tohoto textu, musíme je nyní alespoň jednou citovat: Kniha [? ], článek v časopisu [? ], příspěvek na konferenci [? ], [www odkaz \[? \]](#).

Ještě přidáme další ukázkou citací online zdrojů podle české normy. Odkaz na wiki o frameworkch [? ] a ORM [? ]. Použití viz soubor `reference.bib`. V seznamu literatury by nyní měly být živé odkazy na zdroje. V `reference.bib` je zcela nový typ publikace. Detaily dohledal a dodal Petr Dlouhý v dubnu 2010. Podrobnosti najdete ve zdrojovém souboru tohoto textu v komentáři u příkazu `\thebibliography`.

## B.4.2 Odkazy na obrázky, tabulky a kapitoly

- Označení místa v textu, na které chcete později čtenáře práce odkázat, se provede příkazem `\label{navesti}`. Lze použít v prostředích `figure` a `table`, ale též za názvem kapitoly nebo podkapitoly.
- Na návěští se odkážeme příkazem `\ref{navesti}` nebo `\pageref{navesti}`.

## B.5 Rovnice, centrováná, číslovaná matematika

Jednoduchý matematický výraz zapsaný přímo do textu se vysází pomocí prostředí `math`, resp. zkrácený zápis pomocí uzavření textu rovnice mezi znaky `$`.

Kód `$ S = \pi * r^2 $` bude vysázen takto:  $S = \pi * r^2$ .

Pokud chcete nečíslované rovnice, ale umístěné centrováně na samostatné řádky, pak lze použít prostředí `displaymath`, resp. zkrácený zápis pomocí uzavření textu rovnice mezi znaky `$$`. Zdrojový kód: `$$$ S = \pi * r^2 $$$` bude pak vysázen takto:

$$S = \pi * r^2$$

Chcete-li mít rovnice číslované, je třeba použít prostředí `equation`. Kód:

```
\begin{equation}
  S = \pi * r^2
\end{equation}
```

```
\begin{equation}
  V = \pi * r^3
\end{equation}
```

je potom vysázen takto:

$$S = \pi * r^2 \quad (\text{B.1})$$

$$V = \pi * r^3 \quad (\text{B.2})$$

## B.6 Kódy programu

Chceme-li vysázet například část zdrojového kódu programu (bez formátování), hodí se prostředí *verbatim*:

```

      (* nickname2 *)
Lego> Refine in1
      (do_reg (nickname1 h));
Refine by in1 (do_reg (nickname1 h))
    ?4 : pcddata
    ?5 : pcddata
      (* surname2 *)
Lego> Refine surname1 h;
Refine by surname1 h
    ?5 : pcddata
      (* email2 *)
Lego> Refine undo_reg (email1 h);
Refine by undo_reg (email1 h)
*** QED ***
```

## B.7 Další poznámky

### B.7.1 České uvozovky

V souboru `k336_thesis_macros.tex` je příkaz `\uv{}` pro sázení českých uvozovek. „Text uzavřený do českých uvozovek.“



## Dodatek C

# Seznam použitých zkratek

**2D** Two-Dimensional

**ABN** Abstract Boolean Networks

**ASIC** Application-Specific Integrated Circuit

⋮



## Dodatek D

# UML diagramy

Tato příloha není povinná a zřejmě se neobjeví v každé práci. Máte-li ale větší množství podobných diagramů popisujících systém, není nutné všechny umísťovat do hlavního textu, zvláště pokud by to snižovalo jeho čitelnost.



## Dodatek E

# Instalační a uživatelská příručka

Tato příloha velmi žádoucí zejména u softwarových implementačních prací.



## Dodatek F

# Obsah přiloženého CD

Tato příloha je povinná pro každou práci. Každá práce musí totiž obsahovat přiložené CD. Viz dále.

Může vypadat například takto. Váš seznam samozřejmě bude odpovídat typu vaší práce. (viz [? ]):



Obrázek F.1: Seznam přiloženého CD — příklad

Na GNU/Linuxu si strukturu přiloženého CD můžete snadno vyrobit příkazem:

```
$ tree . >tree.txt
```

Ve vzniklém souboru pak stačí pouze doplnit komentáře.

Z **README.TXT** (případně index.html apod.) musí být rovněž zřejmé, jak programy instalovat, spouštět a jaké požadavky mají tyto programy na hardware.

Adresář **text** musí obsahovat soubor s vlastním textem práce v PDF nebo PS formátu, který bude později použit pro prezentaci diplomové práce na WWW.