VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2020 Pavel Vashkevich

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

MINIMALIZACE OPERAČNÍHO SYSTÉMU CENTOS
REDUCTION OF CENTOS OPERATING SYSTEM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **Pavel Vashkevich**
AUTHOR

VEDOUCÍ PRÁCE — **prof. Ing. Dan Komosný, Ph.D.**
SUPERVISOR

BRNO 2020

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Informační bezpečnost**

Ústav telekomunikací

Student: Pavel Vashkevich ***ID:*** 195174
Ročník: 3 ***Akademický rok:*** 2019/20

NÁZEV TÉMATU:

Minimalizace operačního systému CentOS

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se s operačním systémem CentOS. Proveďte redukci tohoto operačního systému (místa zabraného na paměťovém médiu) při zachování provozu grafického webového prohlížeče. Prohlížeč musí zobrazovat aktuální titulní stránku VUT v Brně v nezměněné podobě. Při redukci se zaměřte na systémové programy a moduly jádra. Vytvořte sadu skriptů, které vytvoření redukovaného systému zautomatizují. Vytvořené skripty zveřejněte pod licencí MIT.

V rámci semestrálního projektu proveďte návrh redukce včetně provedení. V rámci bakalářské práce proveďte automatizaci redukce a funkčnost otestujte.

DOPORUČENÁ LITERATURA:

[1] Linux Dokumentační projekt. 4. vyd. Computer Press, 2008. 1336 s. ISBN: 978-80-251-1525-1.

[2] COOPER, M. Advanced Bash-Scripting Guide. Lulu.com, 2010. ISBN: 978-14-357-5218-4.

Termín zadání: 3.2.2020 ***Termín odevzdání:*** 8.6.2020

Vedoucí práce: prof. Ing. Dan Komosný, Ph.D.

doc. Ing. Jan Hajný, Ph.D.
předseda rady studijního programu

ABSTRAKT

Účelem této práce je provést redukci operačního systému Linux. Pro minimalizaci byla zadaná linuxová distribuce CentOS 8, která je vytvořena na základě otevřených zdrojových kódů. Tyto jsou šířeny společností Red Hat. Teoretická část práce popisuje základní součásti operačního systému, mezi které patří linuxové jádro, ovladače a grafické rozhraní. Praktická část popisuje návrh metody pro minimalizaci operačního systému. Tato metoda zahrnuje i porovnání minimalistických webových prohlížečů. Výsledná velikost po provedení redukce je 731 MB. Software pro automatizaci minimalizace je zveřejněn v osobním repozitáře na GitHub pod licencí MIT.

KLÍČOVÁ SLOVA

Minimalizace, Operační systém, Linux, CentOS, Minimalistický webový prohlížeč, Bash

ABSTRACT

The goal of this bachelor's thesis is to perform a reduction of the Linux operating system. For minimalization was entered the Linux distribution CentOS 8, which is created on the basis of open-source code. This code are spread by Red Hat. The theoretical part describes the basic components of the operating system, including the Linux kernel, drivers and graphical interface. The practical part describes the suggestion of a method for minimizing the operating system. This method also includes a comparison of lightweight web browsers. The resulting size after reduction is 731 MB. Minimization automation software is published in the personal repository on GitHub under the MIT license.

KEYWORDS

Minimization, Operating system, Linux, CentOS, Lightweight web browser, Bash

VASHKEVICH, Pavel. *Minimalizace operačního systému CentOS*. Brno, Rok, 54 s. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce: doc. Ing. Dan Komosný, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Minimalizace operačního systému CentOS“ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu bakalářské práce panu prof. Ing. Danu Komosnému, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů

Úvod

Tato bakalářská práce se věnuje minimalizaci linuxové distribuce CentOS, která by mohla být využita na jednoúčelovém zařízení majícím omezenou kapacitu paměťového média. Konečných řešení minimalistických linuxových distribucí existuje celá řada, jejichž vývoj a podpora pokračují již během několika let. Příkladem může sloužit distribuce ArchBang založena na ArchLinux nebo Elive založena na Debian zabírající ne více než 700 MB na pevném disku a jsou plně funkčními desktopovými operačními systémy. Velkým rozdílem v řešení minimalizace, které je využíváno v těchto operačních systémech, a tím, které je představeno v teto práci, je směr postupu. Vývojáři distribuce staví svůj systém od začátku (jako základ je využito vanillo jádro) a postupně přidávají všechny nezbytné části specifické pro konkretní požadavky tohoto systému. V souladu se zadáním práce, jejímž cílem je studium struktury a činnosti operačního systému, byla redukce provedena na již hotové linuxové distribuce. CentOS v zásadě nabízí i minimální instalaci s menším počtem programových balíčků, která bude použita v mé práci. Samotná redukce byla provedena pomocí skriptu napsaného v programovacím jazyce Bash a ve výsledku bylo dosaženo minimalizace operačního systému o více než polovinu oproti zmíněné minimální instalaci CentOS.

Součástí této práce je přehled operačního systému Linux, kterému je věnována kapitola 1. V jejích podkapitolách je popsáno jádro operačního systému, jeho moduly a informace ohledně firmware, který je nezbytný pro částečnou nebo úplnou funkčnost některých hardwarových zařízení. Kapitola 2 se zabývá balíčky distribujícími softwarové komponenty, jsou zde diskutovány jejich závislosti a konfliktní situace. Kapitola rovněž popisuje strukturu balíčků a zmiňuje datová úložiště, odkud se balíčky instalují. V podkapitole 2.8 je vysvětlen způsob kompilace zdrojových kódu a popsán nastroj sloužící pro tento účel spolu s jeho čtyřmi komponenty přesně odpovídajícími čtyřem procesům odehrávajícím se v průběhu kompilace. Kapitola 3 pojednává o systému X Window, který umožňuje grafické aplikaci zobrazit okno na obrazovce, a znázorňuje jeho architekturu. Poslední kapitola 4 obsahuje návrh metody minimalizace linuxové distribuce a je rozdělena na podkapitoly. Každá z těchto podkapitol uvádí metodu pro redukci konkrétní částí operačního systému. K dosažení cíle zachovaní provozu grafického webového prohlížeče v podkapitole 4.6 je provedeno srovnání známých minimalistických prohlížečů podporujících pokročilé funkce. Dále je podrobněji okomentovaná kompilace zdrojového kódu prohlížeče, který byl zvolen tak, aby co nejlépe vyhovoval účelu této práce, a následně je v podkapitole 4.7 rozebrána jeho instalace na minimalizovaném systému. Shrnutí dosažených výsledků minimalizace je uvedeno v podkapitole 4.8.

1 Operační systém Linux

Operační systém lze představit jako formu rozhraní mezi zařízeními výpočetního systému a aplikačními programy. Mezi jeho nejvýznamnější úkoly patří správa zařízení a procesů a zároveň efektivní rozdělení systémových prostředků.

V této práci je používán operační systém Linux, který je třetím nejvíce využívaným systémem pro desktop počítače [1]. Jednou z výhod tohoto systému je, že je distribuován zdarma, a nevyžaduje tak žádnou placenou licenci pro instalace. Kromě toho jsou jeho zdrojové kódy otevřené, což umožňuje používat a upravovat systém podle vlastního uvážení a případně opravovat vyskytující se chyby nebo jiné nedostatky systému. Další možností je rozšiřovaní jeho funkčnosti psaním svých vlastních doplňků nebo programů, které budou fungovat pod jeho vedením.

Linux reprezentuje pouze název jádra operačního systému, které existuje jedno jediné a jehož vývoj koordinují jeho tvůrci [2]. Samotný operační systém je šířen ve formě hotových distribucí s příslušnou sadou programů konfigurovaných pro specifické požadavky. Tyto distribuce lze zařadit do dvou kategorií: vysoce specializované (například pro použití na superpočítačích) a určené pro hromadné použití (například pro instalace na domácí počítače neboli desktopové systémy). Nejznámější distribuce jsou Red Hat Enterprise Linux, Fedora, Debian, SuSE, CentOS, Gentoo a Ubuntu.

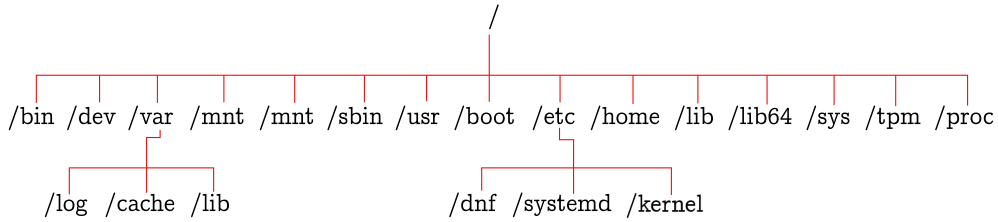
Distribuce CentOS, jejíž minimalizace je cílem mé práce, byla vyvinuta nadšenci v roce 2004. Pro vytvoření konečného produktu vývojáři používají otevřený zdrojový kód distribuce Red Hat Enterprise Linux, který produkující společnost poskytuje v rámci licence pro svobodný software. Cílem projektu je zprovoznit stabilní a bezpečnou linuxovou platformu totožnou se zkušenostmi korporačního operačního systému. Z tohoto důvodu je CentOS vhodnou volbou v případech, kde je žádoucí mít dlouho podporovanou distribuci se stejnými technologiemi, které nabízí Red Hat [3].

Operační systém Linux poskytuje dva typy režimu, pomocí kterých je možné s ním komunikovat: grafický a textový. V textovém režimu je prostředníkem mezi uživatelem a systémem interpret příkazů („**shell**“), jenž je zodpovědný za přijímání příkazů, které jsou mu zadány, za jejich zpracování a provádění. Nejběžnějším příkazovým interpretem je `bash`. Shell také podporuje programovací konstrukty, které umožňují shromažďovat složité i jednoduché příkazy do souboru (skriptu) a sjednotit je pro jeden určitý společný úkol.

1.1 Standardní hierarchie adresářové struktury

Všichni uživatelé linuxových distribucí zjevně nebo implicitně pracují se soubory. Souborový systém CentOS představuje stromovou strukturu (viz obr. 1.1), která

následuje Filesystem Hierarchy Standard navržený pro vývojáře linuxových distribucí [4]. Kořenem teto struktury je adresář, který je reprezentován pomocí lomítka „/“. Cesty k určitým souborům nebo složkám jsou odvozeny z kořenového adresáře, ke kterému jsou postupně na pravé straně zadány názvy vnořených adresářů, kterými systém prochází. Tato struktura umožňuje snadnější orientaci a práci se systémem, jelikož většina souborů se společným účelem použití, například dočasné soubory nebo logy událostí, jsou ukládány do jediného adresáře, který je pro tyto účely speciálně vytvořen.



Obr. 1.1: Kořenový souborový systém

Z pohledu redukce operačního systému je má práce zaměřena na adresáře, jež mají co největší velikost, jako jsou /boot, /usr a /var, ale ve stručnosti budou v této práci zmíněny i další.

1.1.1 Konfigurační adresář /etc

Tento adresář obsahuje velký počet systémových konfiguračních souborů. Konfigurační soubory jsou textové a lze je upravit pomocí libovolného textového editoru, což výrazně zjednodušuje nastavení. Jestliže systém obsahuje konfigurační soubory pro změnu svého chování, má je po přidání nové aplikace nebo služby do systému zachovat v /etc. Proto je název souborů často shodný s názvem aplikace, démonu nebo služby s příponou „.conf“ na konci [5].

1.1.2 Adresář pro start systému /boot

Adresář /boot je odpovědný za rozmístění souborů spojených se zavaděčem systému *grub*2, samotného linuxového jádra *vmlinuz* a obrazu dočasného souborového systému *initramfs* (viz výpis 1.1). Navíc pro neočekávané situace, například pro případ selhání jádra při startu, se v něm nachází záchranná kopie tohoto jádra (**rescue**), kterou zavaděč systému umožňuje zvolit před startem operačního systému. Podobně je to zajištěno i pro *initramfs*. Další důležité soubory jsou:

- **/boot/grub2/grub.cfg** – toto je centrální konfigurační soubor zavaděče systému, který obsahuje různé konfigurace inicializace operačního systému a parametry předávané jádru při jeho zavádění.
- **/boot/config-kernel-version** – tento soubor ukládá všechny konfigurace jádra [4].

```
[root@localhost boot]# ls
config-4.18.0-80.e18.x86_64                 loader
efi                                         lost+found
initrams-4.18.0-80.e18.x86_64kdump.img      grub2
initrams-0-rescue.img                       vmlinuz-0-rescue.img
initrams-4.18.0-80.e18.x86_64.img
vmlinuz-4.18.0-80.e18.x86_64
```

Výpis 1.1: Obsah adresáře /boot

1.1.3 Adresář pro dočasné soubory /var

Soubory a data měnící svou velikost za běhu systému by se měly nalézat v adresáři /var.

/var/log

Klasickým příkladem je /var/log představující centrální sběrnici všech logovacích událostí. Adresář se dále dělí na podadresáře a soubory, do kterých jsou zapisovány události podle informace, kterou poskytují:

- **/var/log/messages** – je globálním systémovým záznamem, do kterého jsou vkládány zprávy od okamžiku spuštění. Zde jsou ukládány různé hlášky jádra, velkého počtu služeb, detekovaných zařízení nebo síťových rozhraní.
- **/var/log/secure** – do tohoto souboru jsou vkládány záznamy o autorizaci uživatele, včetně úspěšných a neúspěšných pokusů, a o příslušných mechanismech autentizace.
- **/var/log/lastlog** – tento binární soubor obsahuje informace o posledním přihlášení uživatele do systému, včetně času a data.
- **/var/log/wtmp** – rozšiřuje předchozí záznam o odhlášení uživatele a zapisuje čas spuštění nebo restartu systému [6].
- **/var/log/cron** – v tomto souboru jsou uloženy zprávy od služby `crond`, která se stará o periodické provádění příkazů [6].

/var/cache

Jiné proměnlivé soubory jsou umístěny v adresáři /var/cache, což jsou soubory mezipaměti, které slouží k usnadnění práce různým nástrojům. Například balíčkovací

systém **dnf** (viz 2.7) zde ukládá informace o tom, jaké balíčky jsou pro něj k dispozici v určitých repozitářích. Tato informace je automaticky aktualizována pokaždé, je-li s ním provedena nějaká akce. Balíčkovací systém může dokonce umisťovat do mezipaměti i samotné balíčky, které byly staženy, ale ještě nejsou nainstalované.

/var/lib

Aplikace v průběhu své práce zapisují užitečné informace o svém stavu nebo stavu celého systému do adresáře /var/lib. Tyto informace obecně používají aplikace mezi dvěma svými spuštěními nebo se přenáší mezi dvěma současně spuštěnými kopiemi stejné aplikace [4]. Data v každém podadresáři se vztahují k jedné konkretní aplikaci, například /var/lib/rpm obsahuje databázi balíčkovacího systému **rpm**. Tato databáze poskytuje kompletní informace o každém balíčku, umístění jeho dokumentace a konfiguračních souborů a může být použita k dotazování, co je již nainstalováno, nebo k určení, zda jsou nainstalovány nejnovější verze softwaru.

1.1.4 Adresář prostředků a programů /usr

Adresář /usr je největší ze všech adresářů v operačním systému Linux a je určen pro ukládání dat pro všechny programy. Na výpisu 1.2 lze vidět, že obsahuje stejné podadresáře jako kořenový adresář. Nicméně do kořenového adresáře spadají pouze nástroje potřebné k zavedení a obnovení systému, všechny ostatní by měly být umístěny v podadresáři /usr [7]. Pro mou práci jsou důležité následující podadresáře tohoto adresáře:

- **/usr/lib** – obsahuje všechny potřebné soubory pro systémové démony běžící na pozadí a provádějící nějakou systémovou úlohu. Kromě toho se do tohoto adresáře ukládají zaváděné moduly jádra a soubory firmware.
- **/usr/lib64** – tento adresář obsahuje sdílené knihovny, jež programy potřebují pro provádění určitých operací v systému.
- **/usr/share** – v tomto adresáři se nacházejí soubory a podadresáře, které nejsou závislé na konkrétní architektuře operačního systému a mohou být sdíleny mezi nimi [4].

```
[root@localhost usr]# ls
bin     lib     local   src
games   lib64   sbin    rmp
include libexec share
```

Výpis 1.2: Obsah adresáře /usr

1.2 Jádro operačního systému

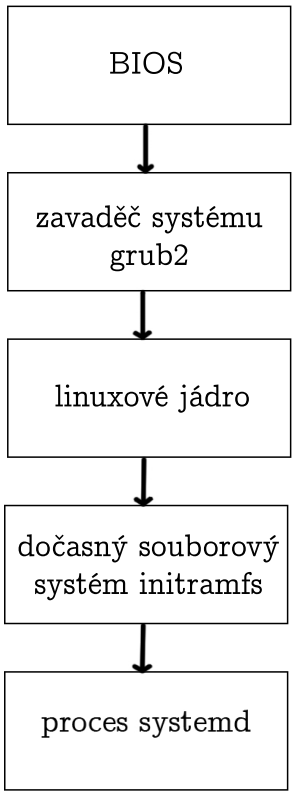
Jádro je takzvaným mozkem operačního systému a jeho úkoly zahrnují co nejúčinnější využití počítačových zdrojů. Jádro monitoruje procesy v systému, zajišťuje jejich spouštění a snaží se optimalizovat využití procesoru tak, aby mohl vykonávat svou funkci efektivně. Pro interakci s hardwarem využívá jádro ovladače, které jsou jeho součástí, anebo jsou distribuovány jako zaváděné moduly a mohou být k jádru v případě potřeby přidány za běhu systému bez jakéhokoliv překompilování.

1.2.1 Zavedení operačního systému

Po spuštění počítače se postupně odehrává řada procesů, z nichž každý má svou určitou roli pro zavedení operačního systému. Tyto procesy jsou zobrazeny na obrázku 1.2. První z nich (**BIOS**) nesouvisí přímo s Linuxem, ale je stejný pro všechny operační systémy. Jednou z jeho povinností je najít a spustit samotný zavaděč systému, který se obvykle nachází v prvních oddílech pevného disku nazývajících se MBR [6]. V CentOS je zavaděčem systému `grub2` a jeho hlavním úkolem je nalézt linuxové jádro v kořenovém adresáři na disku, nahrát jej do paměti RAM a předat konkrétní sadu textových parametrů pro jeho spuštění. Všechny parametry jádra, stejně jako seznam všech dostupných operačních systémů, `grub2` čte z konfiguračního souboru /boot/grub/grub.conf. Po provedení tohoto kroku se na obrazovce uživatele objeví seznam dostupných verzí jádra. Všechna jádra, jak už se zaznělo v podkapitole 1.1.2, jsou uložena v samorozbalovacím komprimovaném formátu (archivu určitého typu) v adresáři /boot. Kromě aktuálního jádra tento adresář obsahuje předchozí verze jádra (obvykle poslední tři), což umožňuje jejich použití, pokud se aktualizovaná verze nemůže načíst správně nebo není kompatibilní s některou důležitou součástí operačního systému.

Mimo spuštění jádra operačního systému při startu počítače zavaděč systému načte do paměti dočasný souborový systém (**initramfs**). Initramfs je umístěn v adresáři /boot spolu s jádrem a představuje `cpio` archiv [8], který obsahuje sadu nástrojů, skripty, knihovny a ovladače. Vzhledem k tomu, že jádro samo o sobě je velmi kompaktní, neobsahuje velké množství ovladačů, které by mohly být potřebné pro komunikaci s diskem. Ovladače jsou soubory, připojují se k jádru pomocí zaváděných modulů a jsou uloženy v kořenovém adresáři na disku (viz 1.2.2). Na tomto místě se může vyskytnout problém. Jelikož jádro ještě nemá přístup k souborům, systém nemůže načíst potřebné ovladače pro spuštění systému. V tento okamžik přichází na řadu initramfs, jehož úkolem je namapovat (připojit) hlavní kořenový adresář.

Jakmile jádro detekuje přítomnost archivu initramfs, extrahuje z něj veškerý



Obr. 1.2: Proces zavedení operačního systému

obsah a spustí v tomto dočasném souborovém systému *init* proces (**systemd** v novějších verzích linuxových distribucí), který má na starosti namapovat klíčový souborový systém. Pak tento proces spustí *init* už v namapovaném souborovém systému pro pokračování bootování systému a uvolní paměť RAM od dočasného souborového systému [6].

1.2.2 Moduly jádra

Linux je monolitické jádro, což znamená, že celý spustitelný kód je soustředěn v jednom souboru. Taková architektura jádra má jednu nevýhodu: v případě potřeby přidat do jádra další funkce (například pro práci se souborovým systémem) je nutné jej znovu zkompilovat a restartovat operační systém. Tento proces je velmi neefektivní i vzhledem k tomu, že v současné době existuje velké množství zařízení, pro interakci s nimiž jádro potřebuje různé ovladače. Z tohoto důvodu (pro usnadnění práce) byla do jádra přidána podpora zavedených modulů [9].

Modul lze popsat jako určitý druh kódu, který může být do jádra kdykoli vložen, anebo odebrán. S jeho pomocí je možné aktualizovat nebo přidat ovladače bez jakékoliv nové kompilace a stejně tak dynamicky rozšířit jeho funkčnost. Navíc moduly umožňují kompaktně sestavit jádro a zahrnout pouze to, co je v danou chvíli

potřebné. Tím šetří používanou paměť a vše ostatní je připojeno s jejich pomocí za běhu systému, je-li to nezbytné.

Moduly jsou shromažďovány vždy pro konkretní verzi jádra a spolu s dalšími konfiguračními soubory jsou uloženy v adresáři /lib/modules/<*verze jadra*>/kernel/. Na výpisu 1.3 je vidět, že jsou seřazeny do kategorií. Kupříkladu adresář /block obsahuje ovladače pro zařízení, která přenášejí náhodně dostupná data v blocích pevné velikosti, jako jsou diskové jednotky [10].

```
[root@localhost ~]# ls /usr/lib/modules/4.18.0-80.e18.x86_64/kernel/
drivers

acpi          cpufreq      firware      iio          media        net
ata           crypto       gpio         infiniband   memstick     ntb
bcma          dax          gpu          input        messages     nvdimn
block         dca          hid          iommu        mfd          nvme
bluetooth     dma          hv           isdn         misc         parport
cdrom         edac         hwmon        leds         mmc          pci
char          firemware    i2c          md           mtd          pcmcia
```

Výpis 1.3: Dostupné moduly zaváděné pro jádro

1.2.3 Mikroprogramové vybavení – firmware

Pro správnou funkčnost zařízení mohou ovladače občas potřebovat malé kusy kódu, které vloží přímo do zařízení. Tento kód je zodpovědný za provádění nízkoúrovňových operací a nazývá se **firmware**. Společnosti vyrábějící hardwarová zařízení distribuují firmware zdarma spolu s otevřeným kódem, anebo pod licencemi zakazujícími zpětnou analýzu či jiné operace reverzního inženýrství [8].

Soubory těchto kódů se nacházejí v adresáři /usr/lib/firmware. Každý z těchto souborů je součástí nějakého programového balíčku, který buď obsahuje firmware pro konkrétní ovladač a zařízení, anebo jako v případě balíčku `linux-firmware` pro všeobecné ovladače.

2 Způsob správy softwaru v operačním systému Linux

2.1 Programové balíčky

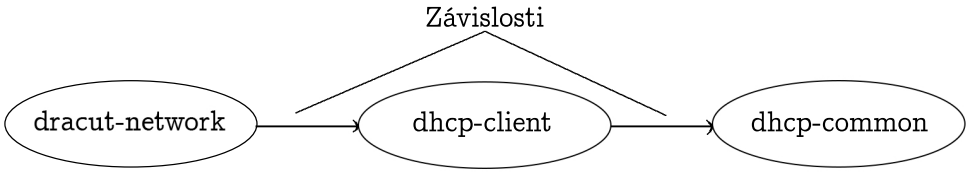
Softwarové komponenty operačního systému Linux, ať už aplikace anebo utility, mohou být reprezentovány formou programových balíčků. S jejich pomocí jsou prováděny všechny operace vztahující se ke změně složení systému, jako jsou instalace programů, jejich odinstalace, kontrola integrity a celková aktualizace [11]. Tato linuxová funkcionalita výrazně usnadňuje správu, jelikož umožňuje pracovat s určitou jednotkou (balíčkem) při provádění těchto činností, ale ne přímo s programovými soubory (jako jsou například zdrojové kódy programů), kterých může být opravdu velké množství.

Navíc se samotné programy neskládají z jednoho spustitelného souboru, pro své provádění mohou používat i jiné pomocné prostředky. Těchto prostředků se může vyskytovat celá řada, jsou to knihovny, konfigurační soubory, a dokonce i další programy. Z důvodu zajištění funkčnosti programu je nutné mít kromě hlavního spustitelného souboru v systému všechny jeho potřebné doplňky [11].

2.2 Závislosti programových balíčků

Jak bylo zmíněno, pro správnou funkčnost programu jsou potřebné různé zdroje, přičemž několik programů může potřebovat stejný zdroj. Z tohoto důvodu jsou ve většině případů tyto zdroje zahrnuty v samostatných vlastních programových balíčcích, aby se nevyskytovaly v několika programech současně. Tyto balíčky by měly být nainstalované v systému. Proto je možné si pod pojmem závislost představit nezbytnost balíčku v prostředcích umístěných v jiném balíčku [11]. Tato vlastnost také zlepšuje vývoj a podporu programu. Zahrnování potřebných zdrojů v samostatných balíčcích a jejich poskytovaní jako závislosti umožňuje v případě jakékoliv změny v těchto zdrojích jednoduše aktualizovat odpovídající balíček, aniž by bylo nutně provádět opětovné stahovaní celého programu.

Na obrázku 2.1 je vidět balíček `dracut-network`, který je využíván pro připojení dočasného souborového systému, jestliže se souborový systém nachází na síťové jednotce [12]. Pro svou funkčnost potřebuje balíček `dhcp-client`. Zároveň balíček `dhcp-client` závisí na balíčku `dhcp-common`. Závislosti tak můžou tvořit celý řetězec, nebo dokonce i stromovou strukturu, kde každý z balíčků bude nositelem nějakých potřebných funkcí pro práci zdroje umístěného v jiných balíčcích.



Obr. 2.1: Řetězec závislostí

V závislosti na využívaném typu balíčkovacího systému pro práci s programovými balíčky je možné v instalačním procesu zkontrolovat, zda jsou všechny potřebné závislosti v operačním systému již přítomny. Pokud některé chybí, bude instalace zastavena anebo bude nabídnuta možnost doinstalovat nezbytné balíčky. Podobně nemůže být balíček odebrán, dokud operační systém obsahuje alespoň jeden balíček, který je na původním balíčku závislý. Odstranění tohoto balíčku by mohlo byt provedeno jen spolu s odstraněním i všech dalších balíčků, které jsou na tomto balíčku závislé.

2.3 Sdílené knihovny

Knihovny implementují společné vlastnosti určené pro použití v programech. Jednoduše řečeno: knihovna je sada funkcí, které mohou sloužit k provádění vstupních a výstupních operací, umožňují přístup k paměti nebo procesoru a práci se sítí.

V Linuxu se knihovny dělí na statické a sdílené. Statická knihovna se k programu přidává ve fázi jejího sestavení. Jelikož většina funkcí knihovny může být používaná více než jedním programem, pro úsporu paměti a lepší údržbu dává smysl jejich shromažďování do sdílených knihoven a jejich poskytování zvlášť pomocí programového balíčku. Princip využití těchto knihoven funguje tak, že kopie sdílené knihovny se načte do paměti a stává se společnou pro všechny programy, které budou její funkce vyžadovat během své činnosti [8] .

V linuxových distribucích mají soubory, v nichž jsou sdílené knihovny uloženy, příponu „.so“ [13], za kterou následuje číslo verze, a nacházejí se v adresářích /lib, /usr/lib, /usr/lib64.

Protože knihovny v průběhu času mohou podléhat změnám kvůli opravě zjištěných chyb, optimalizaci kódu nebo přidaní nové funkcionality, mnohdy se v operačním systému vyskytují různé verze stejné knihovny (viz výpis 2.1). Je to dáno tím, že verze knihovny musí být vždy konzistentní s verzí softwaru a program může odmítnout pracovat, pokud je tato knihovna příliš stará nebo příliš nová [11]. To je pochopitelné, protože starší verze knihovny nemusejí obsahovat potřebné funkce pro

software a naopak v novějších verzích knihovny mohou být některé funkce potřebné pro software smazány.

```
[root@localhost lib64] ls
libcap.so.2
libcap.so.2.25
libcares.so.2
libcares.so.2.2.0
libcollection.so.4
libcollection.so.4.1.1
...
```

Výpis 2.1: Výpis adresáře /use/lib64 obsahující různé verze sdílených knihoven

2.4 Konflikty programových balíčků

Při instalaci programového balíčku může dojít k situaci, kdy potřebný balíček nemůže byt nainstalován kvůli existenci nějakého jiného balíčku. Příčinou tohoto konfliktu mohou být soubory se stejným jménem v balíčcích. Nicméně do konfliktu se dostávají i programy, které nejsou kompatibilní v rámci jednoho systému, kupříkladu poskytují různé implementace stejné systémové funkce, jako jsou poštovní servery [14]. Každý balíček obsahuje informace, se kterými balíčky je v konfliktu, a tato informace je zachována v databázi `rpm`. Balíčkovací systémy zkontrolují tuto informaci při požadované instalaci nového programového balíčku a jestliže bude nalezena shoda, přeruší instalaci s chybovou hláškou. Řešení konfliktní situace spočívá v pouhém odstranění přítomného balíčku ze systému.

Jiným zdrojem konfliktu jsou různé verze stejného balíčku. Každý balíček je kromě svého názvu navíc ještě označen číslem verze značícím stupeň aktualizace softwaru, za který je odpovědný [11]. Linuxová distribuce může obsahovat pouze jednu verzi libovolného balíčku, což je v konfliktu se všemi ostatními verzemi. Výjimkami jsou balíčky odpovídající za sdílené knihovny nebo další podporující zdroje, jelikož představují pouze pomocné soubory pro programy. Jak lze vidět na výpisu 2.2, soubory v balíčku mají přesně definovanou cestu, kde se ukládají v souborovém systému. A z toho plyne, že by neměly existovat situace, kdy jsou různé verze programů nainstalované na rozdílných místech v systému [11]. Tyto balíčky by mohly být aktualizovány na novější verzi, nebo odstraněny a nainstalovány v jejich starší verzi, a to v závislosti na požadavku.

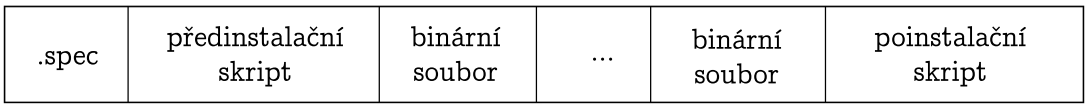
```
[root@localhost lib64] rpm -ql yum
/etc/yum.conf
/etc/yum/pluginconf.d
/etc/yum/protected.d
```

```
/etc/yum/vars
/usr/bin/yum
/usr/share/man/man1/yum-aliases.1.gz
/usr/share/man/man5/yum.conf.5.gz
/usr/share/man/man8/yum-shell.8.gz
/usr/share/man/man8/yum.8.gz
```

Výpis 2.2: Cesty všech souborů balíčku yum

2.5 Instalační metadata

Existuje celá řada typů programových balíčků, pomocí kterých je možné přenést požadovaný software do linuxových systémů. Tento typ je konkretní pro každou linuxovou distribuci a pro manipulaci vyžaduje vlastní balíčkovací systém. V CentOS se primárně používá typ `.rpm`, který představuje speciální archiv a kromě samotného binárního souboru s programem obsahuje celou kolekci užitečných souborů, které je možné vidět na obrázku 2.2 [15]. Jeden z nich je soubor .spec. Balíčkovací systémy vždy zkoumají tento soubor jako první, jelikož zahrnuje veškeré informace o balíčku, jeho účelu a závislostech. Kromě toho poskytuje seznam souborů archivu `.rpm`, kam by se měly v systému zapsat a jaká jsou jejich původní přístupová práva [16].



Obr. 2.2: Struktura rpm

Dalšími soubory jsou předinstalační nebo poinstalační skripty, do kterých může vývojář balíčku například nadefinovat provedení určitých konfigurací nezbytných pro program.

2.6 Datové repozitáře

Repozitář je možné si představit jako internetové úložiště obsahující aktuální testované aplikace, které jsou kompatibilní s konkrétním linuxovým operačním systémem. Repozitáře jsou vytvořeny vývojáři distribuce, aby distribuci udržovali v aktualizovaném a bezpečném stavu. Existují i neoficiální repozitáře, kde mezi sebou různí uživatelé sdílejí balíčky kompilované z otevřených kódů, které ještě nejsou dostupné v oficiálních repozitářích distribuce, anebo jen distribuují užitečné programy a utility.

Pokud balíčkovací systém bude chtít nainstalovat program, přečte si předem seznam dostupných balíčků, které nahraje z repozitáře do své mezipaměti. V případě, že takový program existuje, systém jej nainstaluje. Balíčkovací systémy v CentOS ve výchozím nastavení vyhledávají a instalují balíčky ze dvou oficiálních repozitářů: BaseOS a AppStream [17]. Obecně se v repozitáři BaseOS nacházejí balíčky, které jsou používány pro udržení distribuce v provozu a poskytování základních funkcí systému. Balíčky obsahující dodatečnou funkcionalitu pro systém v závislosti na účelu jeho použití pocházejí z repozitáře AppStream (například balíčky pro nasazení webového serveru).

Úložiště má obvykle spoustu svých kopií po celém Internetu, často se jim říká zrcadla. Jak lze vidět na výpisu 2.3 konfiguračního souboru repozitáře BaseOS, položka *mirrorlist* definuje odkaz na seznam všech dostupných zrcadel pro tento repozitář. Balíčkovací systém může zvolit jedno z těchto zrcadel na základě geografické polohy. Další možnost zvolení nabízí balíčkováci systém `dnf`, který má v sobě zabudovaný zásuvný modul **fastermirror** [18]: nejprve spočítá zpoždění přípojení do každého ze zrcadel a na základě zjištěných údajů zvolí to nejvhodnější. Hlavní výhoda použití kopie repozitáře tkví v rychlejším stahování a ve snížení zátěže hlavního serveru.

Protože existuje riziko, že případný útočník může balíčky na úložišti nahradit svými vlastními upravenými, repozitáře implementují ochranu pomocí digitálních podpisů. Při přidání repozitáře do systému by měl uživatel stáhnout veřejný klíč, který se ukládá do adresáře /etc/pki/ [19].

```
[BaseOS]
name=CentOS-$releasever - Base
mirrorlist=http://mirrorlist.centos.org/?release-$releasever&arch=$base
arch&repo=BaseOS&infra=$infra
gpgcheck=1
enabled=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-centosofficial
```

Výpis 2.3: Výpis konfiguračního souboru repozitáře BaseOS

2.7 Distribuční systémy

Rozsáhlý seznam práce s programovými balíčky, zahrnující jejich vytvoření, instalaci, odinstalaci, aktualizaci a poskytování různých druhů informací, vyžaduje specializovaný nastroj. Tímto nástrojem je balíčkovací systém. Namísto kopírování vícera souborů a spouštění několika skriptů (jak už bylo zmíněno v 2.6, balíček je speciální archiv obsahující sadu souborů) uživatel zadá pouze jeden příkaz v závislosti na jeho potřebách a balíčkovací systém se o vše postará sám. Významným příkladem jeho funkčnosti je instalace nové komponenty. Jednofázová operace se v podstatě skládá

z celého komplexu výpočetních kroků. Nejprve je požadovaný balíček prohledán v seznamu dostupných (jestli se nachází v datových úložištích známých systému). Mezitím musí být zjištěno, jaké balíčky by měly být přidány pro vyhovění všem závislostem a není-li v konfliktu s již nainstalovanými balíčky, a poté už je spuštěn instalátor balíčků [11].

Důležitou vlastností balíčkovacího systému je také možnost provádět komplexní aktualizace celé linuxové distribuce. Pro tento účel balíčkovací systém prozkoumá všechny dostupné repozitáře, ve kterých může být nalezen relevantnější program ve srovnání s tím, který je již nainstalován v systému. V dalším kroku je vypočítán rozsah a realizována aktualizace, při které musí být odstraněn řetězec starších balíčků na sobě navzájem závislých a nahrazen řetězcem novějších verzí [11].

Pro práci s balíčky v CentOS jsou využívány dva balíčkovací systémy: **rpm** a **dnf**. Jeden z jejich hlavních rozdílů spočívá v tom, že **rpm** neumí řešit závislosti při instalaci anebo odebrání na rozdíl od **dnf**.

2.8 Kompilace programových balíčků ze zdrojových kódů

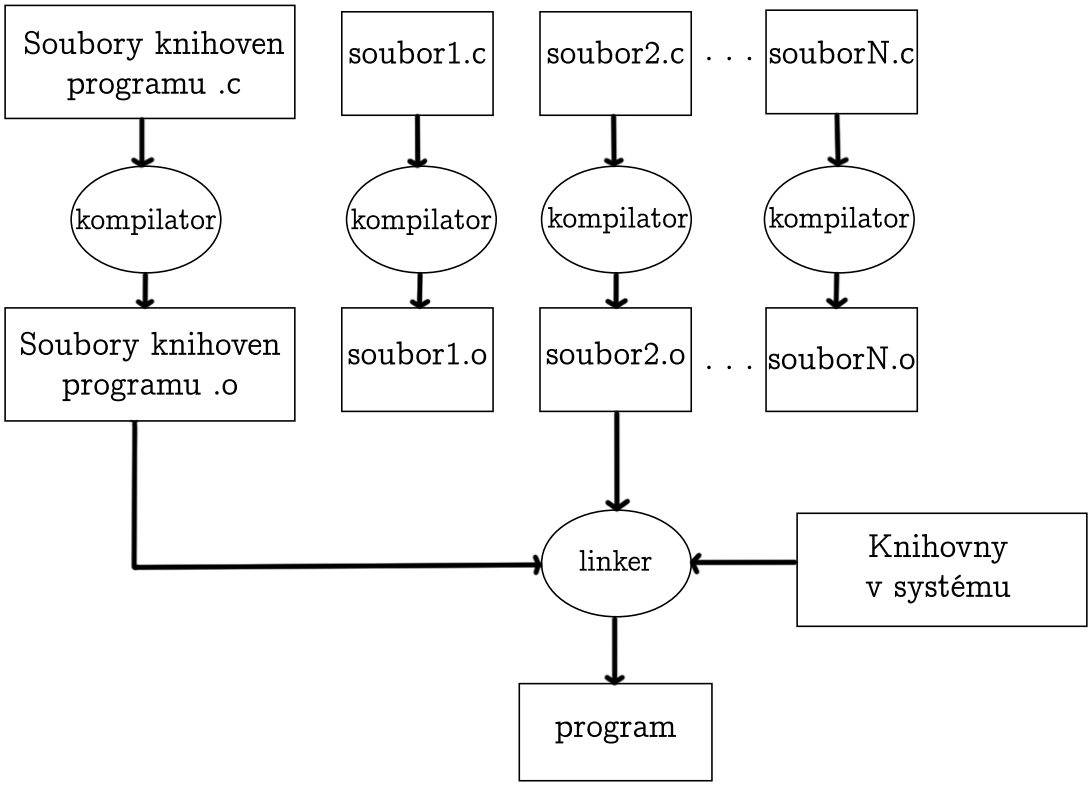
V některých případech nemusí linuxová distribuce poskytovat potřebný softwarový balíček s binárním programovým souborem, který lze snadno nainstalovat z oficiálního datového úložiště nebo z připojeného úložiště třetí strany. Aby mohl být tento program nainstalován, je možné najít a stáhnout jeho zdrojové kódy a zkompilovat je.

Kompilace zdrojových kódů je možná z toho důvodu, že Linux (respektive jeho distribuce) se většinou vyvíjí v programovacím jazyku C [20], stejně jako většina programů pro tento operační systém. Podle principu distribuce svobodného softwaru, pod který spadají i programy napsáné pro Linux, vývojáři distribuují zdrojové kódy svých programů ve formě textů v C nebo C++, které lze i samostatně zkompilovat na linuxové distribuce. Nastroj sloužící k tomuto účelu má název `gcc` a zahrnuje v sobě čtyři komponenty přesně odpovídající čtyřem procesům odehrávajícím se v průběhu kompilace [21]:

- **cpp** - preprocesor,
- **as** - assembler,
- **g++** - sám kompilátor,
- **ld** - sestavovací program (linker).

Nejprve před spuštěním samotného kompilátoru provede svou úlohu preprocesor, který částečně zpracovává zdrojový kód a uskutečňuje jeho úpravy v souladu s některými příkazy, které mohou být ve zdrojových kódech umístěny. Tyto příkazy

specifikují, aby byl do hlavního souboru vnořen obsah všech hlavičkových souborů (označených v kódu pomocí takzvaných direktiv #include). V těchto hlavičkových souborech jsou obvykle obsaženy sady funkcí, které jsou v programu použity, ale nejsou přímo definovány v jeho textu. Tato jejich definice je však umístěna někde jinde, například v jiných zdrojových souborech nebo v knihovnách. Dalším krokem je fáze kompilace (viz obrazek 2.3). Jedna se o převod textu programu (uveden na obrázku jako soubor *.c) pomocí assembleru na sadu strojových instrukcí neboli objektový kód, který se uloží v objektovém souboru (uveden na obrázku jako soubor *.o). Kromě toho zdrojový kód programu často není umístěn v jednom souboru, jednoduše proto, že takový kód je nečitelný, a navíc by po opravě jedné chyby bylo nutné celý kód znovu překompilovat. Proto je zdrojový kód umístěn v několika souborech. Z toho vyplývá, že každý soubor je také zpracován pomocí preprocesoru a kompilován samostatně. V poslední fázi dojde k sestavení programu pomocí `linkeru`. Jeho hlavní úkoly jsou propojit všechny objektové soubory programu do jednoho, propojit volání funkcí s jejich definicemi a připojit soubory knihoven obsahující funkce, které se v projektu volají, ale nejsou definovány. Konečným výsledkem je vytvoření spustitelného souboru programu.



Obr. 2.3: Kompilace zdrojových kódů

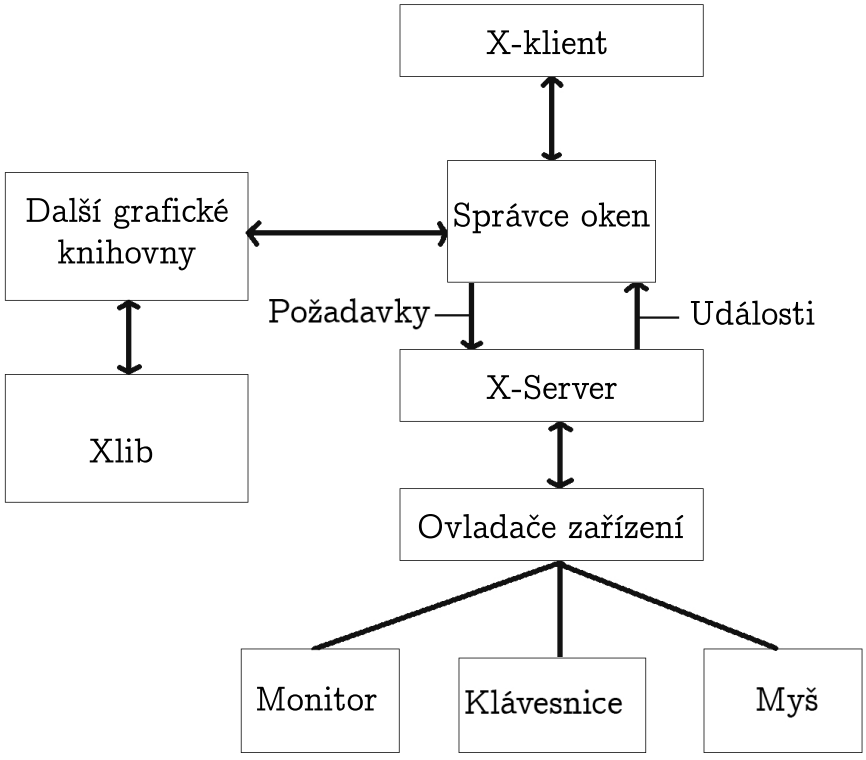
Za zmínku stojí, že na počítačích s různými architekturami procesorů jsou binární soubory vytvářeny v odlišných formátech, tudíž na jednom počítači nelze spustit binární soubor sestavený na jiném počítači. Výjimkou je, mají-li tyto počítače stejnou architekturu procesoru a stejný operační systém [21]. To je jeden z dalších důvodů, proč nejsou některé programy distribuovány ve formě programových balíčků, ale ve formě zdrojových kódů. Tak jsou k dispozici všem uživatelům bez ohledu na to, jaký procesor a operační systém mají.

3 Grafické rozhraní v operačním systému Linux

Pro zobrazení grafického prohlížeče je v této práci používán systém X Window, který představuje soubor knihoven a programů pro komunikaci s grafickou kartou a se vstupními zařízeními. Pomocí tohoto systému je možné přejít z textového režimu do grafického a tam spustit prohlížeč pro požadované zobrazení aktuální titulní stránky VUT v Brně.

3.1 Architektura X Window

Architektura X Window je postavena na principu klient-server [22] a je uvedena na obrázku 3.1



Obr. 3.1: Architektura systému X Window

Jako **X-klient** (dále jen klient) je možné si představit libovolnou aplikaci, která může být spuštěna v grafickém prostředí jako je prohlížeč, poštovní server nebo multimediální přehrávač. Aplikace komunikuje s **X-serverem** (dále jen server) pomocí

speciálních zpráv (datových paketů). Tyto pakety jsou zpracovávané prostřednictvím volání příslušných funkcí z knihovny **Xlib**, která obsahuje celou sadu standardních funkcí pro provádění nízkoúrovňových operací a interakcí mezi všemi částmi X Window systému.

Všechny zprávy lze rozdělit na požadavky a události. V případě, že klient potřebuje vykreslit něco na obrazovce, pošle požadavek na server, který jej zpracuje a zobrazí obrázek na monitoru. Naopak když uživatel stiskne nějakou klávesu, server informuje klienta o události. Jestliže se klient a server nacházejí na jednom počítači, pak jejich komunikace probíhá pomocí soketu. Mezi klientem a serverem se také nachází speciální program (**Správce oken**), který je zodpovědný za umístění oken aplikací na obrazovce a běh všech operací, které jsou s nimi spojené: kreslení rámečků, menu, ikon, posuvníků. Kromě toho umožňuje změnu velikosti okna a jeho polohy. Pro správu a provádění všech těchto transformací volá správce oken funkce z **Xlib** [23].

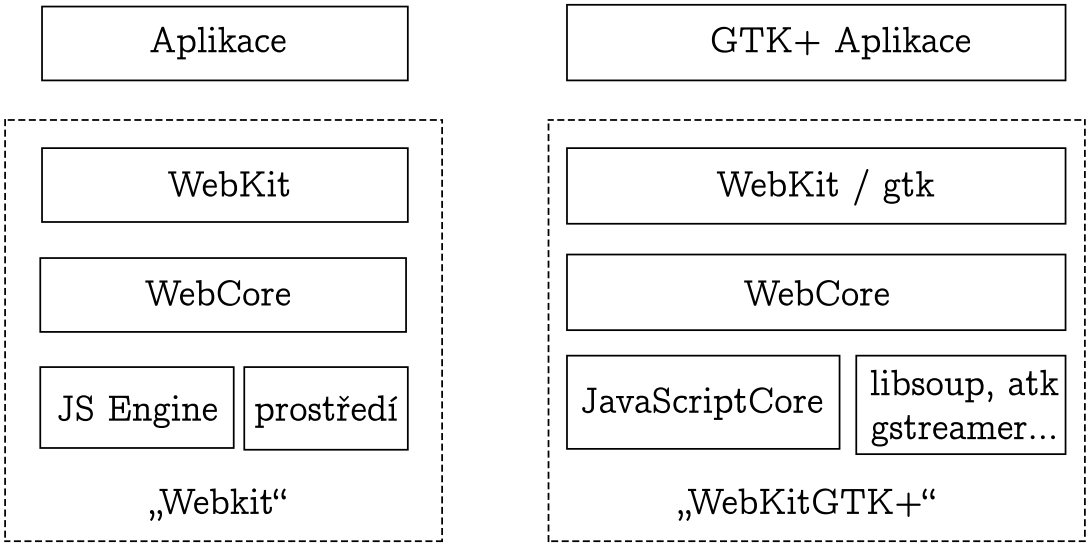
Existuje velký počet správců oken, které nejsou přímo zabudovány do systému X Window, a proto se musí stahovat zvlášť. Jelikož nejsou potřebné pro spuštění serveru, pro úsporu paměti nebude použit v teto práce žádný z nich. V tomto případě bude mít grafický browser pevné zobrazení a nebude možné jej přesunout ani změnit jeho velikost.

Ačkoli jsou všechny prvky grafického rozhraní vykresleny pomocí nízkoúrovňových funkcí **Xlib**, aplikace tyto funkce přímo nevyvolávají, nýbrž apelují na funkce vyšší úrovně, které se nacházejí v grafických knihovnách a nazývají se „toolkit“. Nejznámější z nich jsou knihovny Motif, Qt a GTk [22].

3.2 Grafický prohlížeč

Grafický prohlížeč sám o sobě je webovou aplikací, která musí mít určitou sadu funkcí a komponentů pro zobrazování webového obsahu. Příkladem takových funkcí jsou parsování (HTML, XML, CSS, parsování Javascriptu), textové a grafické vykreslování, dekódování obrázků, interakce s grafický procesorem, přístup k síti atd. K jejich realizaci je k dispozici speciální modul neboli renderovací jádro `Webkit`, které je výkonným nástrojem pro programování grafických webových aplikací a poskytuje určitou sadu mechanismů pro zobrazování webových stránek [24]. Samotný Webkit obsahuje velký počet abstraktních rozhraní, která vyžadují určitou implementaci. Tato implementace (jiný název je „portování“) se uskutečňuje na různých desktopových grafických prostředích pomocí nativních knihoven, poskytovaných konkrétním prostředím, a specifických sad kódu přidaných do stromu zdrojového kódu Webkit. Mezi tyto „Webkit porty“ patří prostředí založené na GTK+ (například Gnome), prostředí založené na Qt (například KDE a Meegoo), Mac OS a další.

Na obrázku 3.2 je možné pozorovat zjednodušenou obecnou architekturu Webkit v porovnání s architekturou grafické webové aplikace využívající WebKitGtk+ (například prohlížeč Midori), založenou na knihovnách Gtk+ [25]. Vrstva **Webkit** je první vrstvou teto architektury, na kterou aplikace odkazuje pro využití všech jejích komponentů. Druhou vrstvou je **WebCore**, který poskytuje základní funkce jako jsou vykreslování, rozvržení, multimédia a jiné. Prvek **JS Engine** na třetí vrstvě odpovídá za spuštění JavaScriptu kódů a u jednotlivých prohlížečů se může lišit (standardní je JavaScriptCore). Dalšími příklady jsou SpiderMonkey u prohlížeče Firefox nebo Chakra u prohlížeče Internet Explorer. Na stejně vrstvě se nachází prvek **prostředí**, který představuje nezbytné knihovny specifické pro toto grafické prostředí, jako je `libsoup` reprezentující HTTP klient-server pro GNOME [8].



Obr. 3.2: Architektura renderovacího jádra Webkit

4 Minimalizace operačního systému CentOS

4.1 Návrh metody minimalizace

Před zahájením realizace jakéhokoli projektu nebo vytvářením aplikace je nutné nejprve určit metody, pomocí kterých bude dosaženo konečných výsledků. Pro splnění účelu mé práce byly stanoveny následující kroky:

1. **Definovat** základní součásti operačního systému.
2. **Identifikovat** skupiny souborů patřící k těmto definovaným součástem a na základě některých jejich vlastností neboli funkcí stanovit, mohou-li být ze systému odstraněny.
3. **Zhodnotit** informace získané z kroku 2 a navrhnou celkovou metodu minimalizace.
4. **Vytvořit** konkretní konečné programové řešení, které automatizuje proces minimalizace.

První krok byl již zmíněn v předchozích kapitolách, ve kterých jsou popsány jednotlivé části linuxové distribuce. Mezi ně patří i adresářová struktura, u které byl v této práci kladen největší důraz na velikost adresáře, a dále části patřící k jádru, jako jsou zavedené moduly, mikroprogramové vybavení a v neposlední řadě také programové balíčky. Po učinění kroků 2 a 3 bylo stanoveno, že samotný návrh minimalizace nelze shrnout do jednoho celku, jelikož metoda minimalizace se pro každou z uvedených dílčích součásti linuxové distribuce naprosto liší a je postavená nejprve na provedení analýzy každé z těchto komponent. Na základě této analýzy se vytváří řešení a určují se nástroje pro provedení redukce. Z tohoto důvodu bylo rozhodnuto uvést návrh pro každou základní součást zvlášť. Tyto návrhy jsou zahrnuty v následujících podkapitolách. Automatizace procesu minimalizace je prezentována jako jediný spustitelný skript v jazyce Bash, do kterého jsou uspořádaně vložena všechna tato řešení. Jakmile bude tento skript spuštěn, redukce operačního systému se provede bez jakéhokoliv zásahu uživatele a po jejím skončení se otevře grafický prohlížeč, ve kterém bude zobrazená aktuální titulní stránka VUT v Brně v nezměněné podobě. Vytvořený skript byl otestován na distribuci CentOS typu Minimal stažené v říjnu roku 2019.

4.2 Minimalizace dočasných souborů

Jak už zaznělo v podkapitole 1.1.3, dočasné soubory se ukládají do adresáře /var. Mezi ně patří logovací soubory, soubory mezipaměti a soubory, do kterých aplikace ukládají informace o svém stavu.

```
find /var/log -name "*.log*" -delete
```

Výpis 4.1: Smazání logovacích souborů

Jelikož logovací soubory mají příponu „.log“, je možné je najít všechny utilitou `find` a smazat (viz výpis 4.1).

To by se však v reálných situacích na zařízeních dělat nemělo, protože logy obsahují důležité informace. Pomáhají při řešení chyb systému a při detekci různých typů útoků a poškození. Pro nakládání s nimi existují nástroje jako `logrotate`, který umožňuje spravovat log-soubory a implementuje velké množství politik pro jejich řízení, například automatickou rotaci, odstranění nebo kompresi [26], a pomáhá chránit zabrané místo na disku při rostoucím počtu logovacích událostí, a tudíž velikostí logovacích souborů.

Soubory mezipaměti nejsou kritické a mohou být smazány. Aplikace, které zde zapisují svá data, s tím musejí počítat a v případe potřeby vygenerovat soubory nové [4]. Tuto vlastnost lze použít i pro minimalizaci, a tím pádem smazat všechno, co se nachází v tomto adresáři. Tento postup je zobrazen na výpise 4.2, kde je příkazu `rm`, který je zodpovědný za smazání souborů, předáván jako parametr cesty do adresáře.

```
rm -rfv /var/cache/*
```

Výpis 4.2: Smazání souborů mezipaměti

Uživatelé by nikdy neměli modifikovat adresář /var/lib/ [4], ve kterém se nacházejí soubory se stavy aplikací, protože to by vedlo k neočekávaným výsledkům. Avšak linuxové distribuce dovolují v případě balíčkovacího systému `rpm` znovu vytvořit jeho databáze. Tím je zmenšena velikost dané distribuce odstraněním nepoužívaných položek (viz výpis 4.3).

```
rpm --rebuilddb
```

Výpis 4.3: Obnovení databáze rpm

4.3 Minimalizace programového vybavení

Různé dodatečné soubory pro programy se nacházejí v adresáři /usr probíraném v podkapitole 1.1.4. Podadresáře tohoto adresáře obsahující manuální stránky, informace a dokumentace k programům je možné smazat.

Navíc se v adresáři /usr/share nachází podadresář /locale, ve kterém jsou uloženy soubory všech dostupných lokalizací pro operační systém Linux, a tento podadresář

zabírá 190 MB místa na disku. Lokalizace se vztahuje na různé části operačního systému: způsob zobrazení data a času, určuje pravidla pro národní reprezentaci čísel s plovoucí desetinnou čárkou a měnových veličin, umožňuje definovat lexikografické pořadí znaků použité abecedy atd. Uživatel může zvolit jakoukoliv lokalizaci při instalaci systému nebo lokalizaci změnit za běhu systému. Všechny položky lokalizace lze zobrazit pomoci příkazu `locale`, jak lze vidět na výpise 4.4.

```
[root@localhost ~]# locale
LANG=en_US.UTF-8
LC_CTYPE="en_US.UTF-8"          LC_NAME="en_US.UTF-8"
LC_NUMERIC="en_US.UTF-8"        LC_ADDRESS="en_US.UTF-8"
LC_TIME="en_US.UTF-8"           LC_TELEPHONE="en_US.UTF-8"
LC_COLLATE="en_US.UTF-8"        LC_MEASUREMENT="en_US.UTF-8"
LC_MONETARY="en_US.UTF-8"       LC_INEDTIFICATION="en_US.UTF-8"
LC_MESSAGES="en_US.UTF-8"       LC_ALL=
LC=PAPER="en_US.UTF-8"
```

Výpis 4.4: Zobrazení používané lokalizace pomocí příkazu locale

Je možné zjistit, která lokalizace je použita. Soubor, který za lokalizaci odpovídá, lze ponechat a zbytek odstranit. Pro tyto účely je potřeba rozparsovat výstup příkazu `locale`, čímž je získána zkratka lokalizace, která je shodná s názvem jejího souboru, v mém případe to je „en_US“. Nakonec je možné smazat nepoužívané lokalizace. Celkový kód pro provedení všech těchto operací je zobrazen na výpise 4.5.

```
# odstranění manuálních stránek a souborů dokumentů
rm -rfv /usr/share/doc/
rm -rfv /usr/share/man/
rm -rfv/usr/share/info/

# rozparsovaní výstupu příkazu locale
system_lang=‘locale | cut -f2 -d= | cut -f1 -d. | head -n 1‘

# odstranění nepoužívaných lokalizací
for target in ‘find /usr/share/locale -maxdepth 1 -not -name $system_
lang*‘; do
    rm -fv $target/LC_MESSAGES/*
done
```

Výpis 4.5: Minimalizace adresáře /usr/share

4.4 Minimalizace struktury jádra

Jakmile je jádro spuštěno při startu systému, snaží se najít a inicializovat všechna přítomná zařízení, a tudíž natáhne všechny potřebné moduly ovladačů (viz 1.2.2)

pro řízení komunikace s hardwarem [6]. Tato vlastnost bude v práci využita pro minimalizaci systému.

Jelikož jádro od začátku ví, které moduly bude potřebovat, a zároveň je nahraje do operační paměti, lze vyvodit závěr, že je možné smazat ostatní moduly, které mohou být přidány, ale nejsou v daný okamžik potřebné. Příkaz `lsmod` v Linuxu může poskytnout seznam všech momentálně zavedených modulů (viz výpis 4.6). V mém systému je jich celkem 71, ale toto číslo se může u jiných počítačů lišit v závislosti na hardwarových součástech.

```
[root@localhost ~]# lsmod
Module                  Size  Used by
nf_tables_set          32768  5
nft_fib_inet           16384  1
nft_fib_ipv4           16384  1 nft_fib_inet
nft_fib_ipv6           16384  1 nft_fib_inet
nft_fib                16384  3
nft_reject_inet        16384  4
nft_reject_ipv4        16384  1 nft_reject_inet
nft_reject_ipv6        16384  1 nft_reject_inet
nft_reject             16384  1 nft_reject_inet
nft_ct                 20480  7
nft_chain_nat_ipv6     16384  6
nf_conntrack_ipv6      20480  8
nf_defrag_ipv6         20480  1 nf_conntrack_ipv6
nf_nat_ipv6            16384  1 nft_chain_nat_ipv6
nft_chain_route_ipv6   16384  1
nft_chain_nat_ipv6     16384  6
nf_conntrack_ipv4      16384  8
nf_defrag_ipv4         16384  1 nf_conntrack_ipv4
nf_nat_ipv4            16384  1 nft_chain_nat_ipv4
nf_nat                 36864  2 nf_nat_ipv6,nf_nat_ipv4
nft_chain_route_ipv4   16384  1
nf_conntrack          155648  6
ip6_tables             32768  0
ip_tables              28672  0
nft_compat             20480  0
ip_set                 45056  0
nf_tables             147456  184
nfnetlink              16384  3 nft_compat,nf_tables,ipset
ext4                  733184  1
mbcache                16384  1 ext4
jbd2                  122880  1 ext4
...
```

Výpis 4.6: Momentálně zavedené moduly

Důležitým sloupcem je **Module**, jelikož ukazuje jména všech zavedených modulů. Informace o závislostech každého z těchto modulů jsou uvedeny ve sloupci **Used by** a označují počet modulů jádra anebo procesu, který používá daný modul. Jména těchto závislých modulů jsou taktéž uvedena v tomto sloupci. Například modul *ip_tables* není právě používán, oproti *nft_fib_ipv*4, jehož hodnota v **Used by** je rovna 1 a nemůže být přímo odebrán, pokud jiný z modulů (v tomto případě jen *nft_fib_inet*) na něm závisí.

Protože je možné zjistit názvy všech momentálně zavedených modulů, zbývá jen zjistit cesty do jejich souborů. Tyto soubory budou ponechány v systému a všechny ostatní soubory odpovídající za jiné moduly budou smazány. Nejprve bude definována funkce **remove**, která bude odpovědná za smazání a bude přijímat seznam souborů jako argument (viz výpis 4.7).

```
function remove {
    for target in "$@"; do
        rm -fv $target
    done
}
```

Výpis 4.7: Funkce pro smazání souborů

Kód dalšího postupu je zobrazen na výpisu 4.8. Pomocí utility `find` jsou hledány všechny soubory dostupné v systému zavedených modulů (mají příponu „.ko“) a jsou zapisovány jejich cesty do prvního zvláštního souboru. Poté je lineárně neboli řadek po řadku procházeno každé jméno modulů poskytnutých příkazem `lsmod` a utilita `modinfo` s argumentem **-n** umožňuje odhalit rovněž jejich cesty. V dalším kroku jsou tyto cesty pouze zapsány do druhého zvláštního souboru. Tyto dva soubory jsou pak seřazeny tak, aby měly podobnou strukturu, a jsou porovnány utilitou `comm`. Nesoudělné položky budou smazány pomocí funkce **remove**.

```
DIR1=$HOME/all_modules.txt
DIR2=$HOME/current_modules.txt

# hledání všech dostupných modulů jádra
find /lib/modules/$(uname -r) -type f -name '*.ko*' > $DIR1

# objevovaní cest modulů jádra
for module in `lsmod | cut -f1 -d" " | tail -n +2`; do
    filename=`modinfo $module -n`
    echo "$filename" >> $DIR2
done
# seřazení souborů pro vytváření podobné struktury
sort $DIR1 | uniq > file1.sorted
sort $DIR2 | uniq > file2.sorted
```

```
# využití funkce remove k vymazání nepoužitých modulů jádra
remove `comm -23 file1.sorted file2.sorted`
```

Výpis 4.8: Smazání zavedených modulů

Jestliže bude ovladač potřebovat jeden z firmware (viz 1.2.3), vyšle požadavek jádru operačního systému. Jádro přesměruje tento požadavek démonu **udev**, který je v Linuxu zodpovědný za správu zařízení. Nakonec **udev** přečte soubor firmware a vrací data jádru. Tímto způsobem ovladač obdrží svůj firmware.

Díky tomu, že **udev** démon otevírá soubory s kódem firmware, je možné zjistit datum posledního přístupu k nim a tím získat cestu k souborům, které používají ovladače, a ostatní smazat. Pro tyto účely je opět využívána utilita `find` (viz výpis 4.9) s parametrem `-atime` umožňující najít soubory, jež nebyly otevřeny v zadaném počtu dní. Nalezené soubory jsou předávány dál jako argument funkce **remove** z výpisu 4.7.

```
remove `find /usr/lib/firmware -atime +2`
```

Výpis 4.9: Smazání nepoužívaných firmware

Jádro operačního systému a archiv dočasného souborového systému společně s jejich záchrannými obrazy jsou ukládány do adresáře /boot (viz 1.1.2). Záchranné obrazy jsou generovány automaticky po instalaci systému, za což je odpovědný balíček `dracut-config-rescue` [12]. Tyto obrazy nejsou nezbytné a jsou potřebné jen v případě, že nelze nastartovat operační systém, a proto je možné je smazat a tímto způsobem redukovat velikost tohoto adresáře na polovinu a ušetřit tak místo zabrané na paměťovém médiu. Po smazání těchto obrazů lze vygenerovat nový konfigurační soubor zavaděče systému tak, aby neukazoval nabídku záložního jádra v bootovacím menu při startu, a také odstranit balíček odpovědný za jeho vygenerování. Kód pro provádění této operace je zobrazen na výpise 4.10.

```
find /boot -maxdepth 1 -type f -name *rescue* -delete
grub2-mkconfig -o /boot/grub2/grub.cfg
```

Výpis 4.10: Smazání záložních obrazů a vygenerování konfiguračního souboru

4.5 Redukce počtu programových balíčků

V linuxové distribuci zabírají balíčky velký objem paměti na disku, protože jsou zodpovědné za fungovaní určitých jeho částí a poskytují tomuto operačnímu systému dodatečnou funkcionalitu. Z toho je možné vyvodit závěr, že odstranění co největší části nainstalovaných balíčků umožňuje v nejvyšší míře minimalizovat systém. V takovém případě budou navíc smazány také všechny soubory a adresáře patřící k těmto balíčkům a vytvořené během jejich instalace.

Jak již zaznělo v podkapitole 2.2, spousta balíčků může byt závislá na ostatních anebo samy mohou poskytovat některé nezbytné zdroje. Navíc je systém může potřebovat pro svůj stabilní provoz. Pomocí skriptu v Bash není možné s jistotou předem určit, zda je balíček pro systém důležitý, nebo bude-li jej potřebovat pro funkčnost některý z jeho komponentů. Proto byla pro minimalizaci rizika zvolena cesta v odstranění pouze osamělých balíčků, které nejsou zdrojem pro ostatní.

Balíčkovací systémy umožňují získat tyto informace stejně jako seznam všech balíčků nainstalovaných v systému. Výpis 4.11 zobrazuje kód tohoto postupu. Nejprve je definován jediný známý osamělý balíček, který musí být chráněn před odstraněním, a jeho nazev je `NetworkManager`. V CentOS je zodpovědný za konfigurace a funkčnost připojení k síti [27]. Jeho odstranění by vedlo ke ztrátě internetového přípojení a tím pádem i k nemožnosti otevřít v grafickém minimalistickém prohlížeči jakoukoli webovou stránku. Z toho důvodu bude v první řadě název tohoto balíčku zapsán do souboru se stejným názvem a příponou „.conf“ a bude umístěn do adresáře /etc/dnf/protected.d/. Tento adresář je prohledáván balíčkovacím systémem `dnf` pokaždé, když musí dojít k odstranění některých z balíčků. V případě nalezení podobnosti názvu bude vyvolaná chyba a balíček zůstane zachován v systému. Navíc je definováno pole *output*, jehož účelem je uložit výsledek činnosti příkazu používaného v dalších krocích. Samotné vyhledávaní osamělých balíčků je relativně snadné. Pomocí příkazu **rpm -qa** lze zjistit seznam všech dostupných programových balíčků v systému. Tento seznam je pak možné lineárně neboli řádek po řádku procházet a číst názvy balíčků, ze kterých lze za pomoci příkazu **dnf repoquery --whatrequires <package-name>** získávat názvy ostatních balíčků, které jsou na zmíněném balíčku závislé. Tyto názvy jsou zapisovány do pole *output* a v případě, že pole nebude obsahovat žádnou položku, kontrolovaný balíček bude odinstalován.

```
echo "NetworkManager" > /etc/dnf/protected.d/NetworkManager.conf

for line in `rpm -qa --queryformat "%{NAME}\n"`; do
    echo "What require package ${line}:"
        mapfile -t output < <(dnf repoquery --installed --queryformat \
        "%{name}" --whatrequires $line)
    echo "${output[@]}"

    if [ ${#output[@]} -eq 0 ]
    then
       echo "Remove package ${line}"
       dnf -y remove $line
    elif [ ${#output[@]} -eq 1 ]
    then
        if [ "${output[0]}" == "$line" ]
      then
```

```
            echo "Remove package ${line}"
            dnf -y remove $line
        fi
    fi
done
```

Výpis 4.11: Odebrání osamělých balíčků

4.6 Výběr a kompilace minimalistického grafického prohlížeče

V tabulce 4.1 je uveden seznam grafických minimalistických prohlížečů, které podporují pokročilé funkce pro otevření webové stránky univerzity VUT v Brně v nezměněné podobě, a to zejména JavaScript. Programové balíčky těchto prohlížečů nejsou v oficiálních repozitářích CentOS a nejsou ani v repozitářích třetích stran. Z tohoto důvodu je nelze přímo stáhnout pomocí balíčkovacích systémů a zajistit tak všechny jejich potřebné závislosti. Jedno z možných řešení je najít otevřené zdrojové kódy těchto prohlížečů buď na jejich oficiálních stránkách, nebo v jejich repozitářích na GitHubu. Zdrojové kódy by měly být poskytovány v rámci licence pro otevřený software (například GPL nebo MIT), tudíž je lze naklonovat na vlastní lokální počítač a pokusit se je zkompilovat (viz 2.8).

V tabulce jsou rovněž uvedeny knihovny, které musejí byt přidány ještě před kompilací, aby byla vůbec spuštěna a správně dokončena. Prohlížeč potřebuje tyto knihovny a bude je využívat i nadále pro své fungování. Příkladem může být knihovna libQt5Svg.so.5 u prohlížeče Otter-browser. V operačním systému se mohou objevovat i sdílené knihovny, které se zkompilují spolu s prohlížečem, například libFalkonPrivate.so.3 u prohlížeče Falkon a libkdeinit5_konqueror.so v případě prohlížeče Konqueror. Množství knihoven nezbytných pro běh jednotlivých prohlížečů je velké (u některých prohlížečů přesahuje počet knihoven 100), ale tyto knihovny se buď nainstalují spolu s programovými balíčky rovněž zmíněnými v tabulce, anebo už existují v linuxové distribuci po její instalaci.

Prohlížeče jsou porovnány na základě hodnoty uvedené ve třetím sloupci tabulky. Tato hodnota zahrnuje velikost operačního systému po provedení minimalizace spolu s prohlížečem a všemi jeho závislostmi (knihovnami a programovými balíčky). Protože velikost operačního systému je v této práci rozhodující a měla by být co nejmenší, jako minimalistický prohlížeč byl zvolen `Otter-browser`.

Prohlížeč	Závislosti	Velikost operačního systému [MB]
Falkon (QupZilla)	*Knihovny:* libFalkonPrivate.so.3, libKF5Archive.so.5, libQt5X11Extras.so.5 *Softwarové balíčky:* qt5-qtwebkit, mesa-dri-drivers, qt5-qtwebengine	986
Otter-browser	*Knihovny:* libQt5Multimedia.so.5, libQt5Svg.so.5, libpulse-mainloop-glib.so.0, libpulse.so.0, libpulsecommon-11.1.so, libsndfile.so.1, libasyncns.so.0, libgsm.so.1 libFLAC.so.8, libatk-bridge-2.0.so.0, libatspi.so.0, libwoff2common.so.1.0.2, libaspell.so.15 *Softwarové balíčky:* qt5-qtwebkit	731
Qutebrowser	*Softwarové balíčky:* qt5-qtwebengine, PyQt5, python36	1205
Surfer	*Softwarové balíčky:* mesa-dri-drivers, webkit2gtk3	889
Konqueror	*Knihovny:* libkdeinit5_konqueror.so, libkonquerorprivate.so.5, libKF5KIOGui.so.5, libKF5KCMUtils.so.5, libKF5KDELibs4Support.so.5, libKF5UnitConversion.so.5, libKF5QuickAddons.so.5, libKF5Declarative.so.5, libKF5Package.so.5, libKF5Konq.so.6 *Softwarové balíčky:* mesa-dri-drivers, qt5-qtwebkit, kio-extras	1105

Tab. 4.1: Porovnání velikosti operačního systému po instalaci různých minimalistických prohlížečů

Provedení kompilace grafického prohlížeče

Ještě před zahájením samotné kompilace zdrojových kódů programu je třeba zajistit, že linuxová distribuce obsahuje všechny nástroje nezbytné pro tento účel. Každý z těchto nástrojů může být stažen samostatně, avšak praktičtějším řešením je nainstalovat celou skupinu programových balíčků sloužících ke kompilaci najednou pomocí možnosti „groupinstall“ balíčkovacího systému `dnf`. Tato skupina balíčků se nazývá „Development Tools“, je umístěná v oficiálním datovém repozitáři CentOS a je vždy k dispozici ke stažení (viz výpis 4.12).

```
[root@localhost ~]# dnf -y groupinstall "Development Tools"
```

Výpis 4.12: Instalace skupiny programových balíčků potřebných pro kompilaci

Jakmile bude instalace dokončena, lze přistoupit k samotné kompilaci, která zahrnuje následující kroky [28]:

1. Získání zdrojového kódu,
2. konfigurace programového balíčku,
3. spuštění nastroje `make` pro sestavení programu,
4. spuštění nastroje `make install` pro instalaci programu.

Získání zdrojového kódu

Aby mohla být kompilace provedena, je nejprve nutné najít a získat zdrojový kód programu. Vývojáři Otter-browser poskytují zdrojový kód ke svému prohlížeči v jimi vytvořeném repozitáři na GitHubu. Tento kód lze buď přímo naklonovat na operační systém pomocí nástroje `git clone`, anebo získat odkaz pro stažení souboru zdrojového kódu uloženého do zip archivu. V této práci byla zvolena druhá možnost a její provedení je zobrazeno na výpisu 4.13. V prvním kroku je nástroji `wget`, který v Linuxu slouží k načtení souborů z Internetu, poskytnut odkaz pro stažení zip archivu. Jméno archivu „master.zip“ na konci tohoto odkazu značí, že bude stažen kód stabilní verze prohlížeče (na GitHubu mohou být umístěny i zdrojové kódy následujících verzí, které jsou stále ve vývoji). Tento archiv pak bude pomocí `unzip` rozbalen a tím se získají zdrojové kódy.

```
[root@localhost ~]# wget https://github.com/OtterBrowser/otter-browser
/archive/master.zip
[root@localhost ~]# unzip master.zip && cd otter-browser-master
```

Výpis 4.13: Získání zdrojového kódu prohlížeče Otter-browser

Konfigurace programového balíčku

Pří sestavování jakéhokoliv programu ze zdrojových kódů se využívají funkce sdílených knihoven, soubory jejich záhlaví (například foo.h) a symbolické odkazy bez verze na tyto knihovny (například libfoo.so -> libfoo.so.1.0). V sekci 2.3 už bylo zmíněno, že hlavní výhodou tohoto postupu je, že není nutno psát své vlastní statické knihovny a zahrnovat tyto knihovny přímo do programu (čímž by byla například zvětšena jeho velikost anebo zkomplikovaná její implementace), ale stačí aplikovat hotové řešení. Z toho vyplývá, že předtím, než bude spuštěna a provedena kompilace Otter-browser prohlížeče, musí být zajištěno, že všechny tyto sdílené knihovny v operačním systému jsou. K tomuto účelu poskytují vývojáři programu společně se zdrojovým kódem také speciální skript, který se jmenuje „configure.sh“, nebo mohou v návodu pro kompilaci, který je obvykle sdílen spolu se zdrojovými kódy, uvést, že je nezbytné použít nástroj `cmake`. Použití tohoto nástroje je nutné i pro prohlížeč Otter-browser.

CMake sám o sobě lze specifikovat jako multiplatformní nástroj pro automatické vytváření programu ze zdrojového kódu [29]. Zároveň ale cmake sám program přímo nesestavuje, ale slouží jako takzvaný obal pro kompilaci. Jedním z cílů tohoto nástroje je zkoumat a zjistit nainstalované sdílené knihovny na linuxové distribuci. Pokud knihovna potřebná pro kompilaci není nainstalovaná, v průběhu činnosti cmake se vyskytne chyba s popisem, ve kterém lze najít jméno chybějící knihovny. Další konaní nastroje a následně i sestavovaní programu není možné až do okamžiku, než bude tato knihovna v systému nalezena.

Zajištění knihoven není to jediné, co by mělo být vyřešeno před kompilací a k čemu slouží cmake. Zdrojové kódy nejsou vždy sdíleny pro konkrétní typ a verzi operačního systému. Proto může být potřeba nejprve určit tuto informaci, dokonce i najít samotný kompilátor, zjistit jeho podporované příznaky a v závislosti na této podpoře některé příznaky přidat. Navíc je třeba provést některé kroky před a po sestavení nebo zaručit přítomnost dalších souborů. Všechny tyto postupy jako i pravidla a cíle sestavení, se definují pomocí deklarativního popisu neboli kódu srozumitelného nástroji cmake v souboru CMakeLists.txt, který se obvykle ukládá v kořenu adresáře spolu se zdrojovými kódy.

Za účelem odhalování chyb při spuštění cmake, které vznikají z důvodu nedostatku sdílených knihoven nebo některých dalších balíčků, je možné předem otevřít soubor CMakeList.txt a najít jména knihoven nebo balíčku, které musí cmake najít v operačním systému. Tato jména se zadávají v příkazu „find_package()“ (viz výpis 4.14). V případě prohlížeče Otter-browser se v systému musí nacházet balíček Qt5 verze 5.6.0 nebo novější a konkrétně jeho komponenty uvedené za parametrem REQUIRED COMPONENTS na prvním řádku. Balíčky uvedené na dalších řád-

cích s parametrem QUIET nejsou povinné a jejich nepřítomnost nebude mít vliv na konfigurace a kompilaci, jelikož mohou být samy sestaveny za jejího běhu.

```
...
find_package(Qt5 5.6.0 REQUIRED COMPONENTS Core Gui Multimedia Network
PrintSupport Qml Svg Widgets)
find_package(Qt5WebEngineWidgets 5.12.0 QUIET)
find_package(Qt5WebKitWidgets 5.212.0 QUIET)
find_package(Hunspell 1.5.0 QUIET)
...
```

Výpis 4.14: Seznam programových balíčků vyhledávaných nástrojem cmake

Linuxová distribuce CentOS neobsahuje po své instalaci sadu uvedených povinných balíčků, a proto musejí být nainstalovány zvlášť (viz výpis 4.15). Název těchto balíčků v sobě zahrnuje speciální slovo „*-devel“, které značí, že se v těchto balíčcích nacházejí všechny soubory potřebné ke kompilaci kódu pro danou knihovnu.

```
[root@localhost ~]# dnf -y install qt5-qtbase-devel
[root@localhost ~]# dnf -y install qt5-qtmultimedia-devel
[root@localhost ~]# dnf -y install qt5-qtsvg-devel
[root@localhost ~]# dnf -y install qt5-qtwebkit-devel
```

Výpis 4.15: Instalace programových balíčků vyhledávaných nástrojem cmake

Jak je zobrazeno na výpisu 4.16, ještě před provedením konfigurací se vytváří prázdny adresář /build, do kterého se během konfigurace vkládají různé dočasné soubory a po ukončení i soubory programu, které budou připraveny ke kompilaci. Do tohoto adresáře je pak nutné přejít a spustit cmake s parametrem „..“ označujícím cestu, kde je vyhledáván soubor CMakeLists.txt. .

```
[root@localhost otter-browser-master]# mkdir build && cd build
[root@localhost build]# cmake ..
```

Výpis 4.16: Spuštění konfigurace programového balíčku

Sestavení a instalace programu

Po úspěšném provedení konfigurace v adresáři /build bude vygenerován soubor Makefile. V podkapitole 2.8 již bylo zmíněno, že program se může skládat z mnoha samostatných souborů zdrojových kódů, z nichž každý je kompilován samostatně a pokud je to možné, nezávisle na ostatních souborech. Z toho plyne, že musí existovat nastroj, který bude znát a spravovat závislosti mezi těmito soubory a stejně tak i pořadí, v jakém je nutné je kompilovat. Tímto nástrojem je `make`. Ke své činnosti make využívá soubor Makefile, ve kterém jsou stanovené sady pokynů, a to především jak a pomocí jakých příkazů musí být program zkompilován ze zdrojových

kódů. Rovněž obsahuje informace o souborech, z nichž program sestává. Po spuštění nástroje make začíná hledat soubor Makefile v aktuálním adresáři a postupuje podle jeho instrukcí, čímž bude uskutečněná kompilace programu (viz výpis 4.17). V dalším kroku lze pomocí make s parametrem „install“ program přímo nainstalovat na operační systém.

```
[root@localhost build]# make
[root@localhost build]# make install
```

Výpis 4.17: Kompilace a instalace programu pomocí make

Pokud budou kompilace a instalace úspěšně dokončeny, spustitelný soubor prohlížeče se objeví v adresáři /usr/local/bin/otter-browser.

Aby nedocházelo ke zvětšování velikosti linuxové distribuce, která zabírá místo na paměťovém médiu, při spuštění skriptu minimalizace opětovným stahováním sady nástrojů určených ke kompilaci a jejím opětovném vykonání, byla zvolena jiná cesta. Jejím cílem je vytvořit zip archiv prohlížeče Otter-browser se všemi zdroji potřebnými pro jeho spuštění a funkčnost (spustitelný soubor a sdílené knihovny) a nahrát jej na existující veřejný FTP server. Z tohoto serveru jej pak lze jednoduše stáhnout a rozbalit v průběhu redukce operačního systému. Tím bude zajištěna přítomnost minimalistického grafického prohlížeče na již minimalizované linuxové distribuci.

Příkazem `ldd` lze zjistit seznam všech sdílených knihoven, které prohlížeč Otter-browser potřebuje (viz výpis 4.18).

```
[root@localhost /usr/local/bin/]# ldd otter-browser
linux-vdso.so.1 (0x00007ffcaa9f9000)
  libQt5WebKitWidgets.so.5 => /lib64/libQt5WebKitWidgets.so.5
  (0x00007f2cc35a0000)
  libQt5DBus.so.5 => /lib64/libQt5DBus.so.5 (0x00007f2cc3507000)
  libQt5Multimedia.so.5 => /lib64/libQt5Multimedia.so.5
  (0x00007f2cc3404000)
  libQt5PrintSupport.so.5 => /lib64/libQt5PrintSupport.so.5
  (0x00007f2cc3356000)
  libQt5Qml.so.5 => /lib64/libQt5Qml.so.5 (0x00007f2cc2f45000)
  libQt5Svg.so.5 => /lib64/libQt5Svg.so.5 (0x00007f2cc2eec000)
  libQt5Widgets.so.5 => /lib64/libQt5Widgets.so.5 (0x00007f2cc2875000)
  libQt5WebKit.so.5 => /lib64/libQt5WebKit.so.5 (0x00007f2cbfb06000)
  libQt5Network.so.5 => /lib64/libQt5Network.so.5 (0x00007f2cbf961000)
  libQt5Gui.so.5 => /lib64/libQt5Gui.so.5 (0x00007f2cbf466000)
  libQt5Core.so.5 => /lib64/libQt5Core.so.5 (0x00007f2cbef81000)
  libstdc++.so.6 => /lib64/libstdc++.so.6 (0x00007f2cbebec000)
  libm.so.6 => /lib64/libm.so.6 (0x00007f2cbe86a000)
  libc.so.6 => /lib64/libc.so.6 (0x00007f2cbe28f000)
  libgcc_s.so.1 => /lib64/libgcc_s.so.1 (0x00007f2cbe652000)
  libpthread.so.0 => /lib64/libpthread.so.0 (0x00007f2cbe06f000)
  libdbus-1.so.3 => /lib64/libdbus-1.so.3 (0x00007f2cbde1b000)
```

```
libGL.so.1 => /lib64/libGL.so.1 (0x00007f2cbdb8f000)
libglib-2.0.so.0 => /lib64/libglib-2.0.so.0 (0x00007f2cbd41e000)
libz.so.1 => /lib64/libz.so.1 (0x00007f2cbcffe000)
...
```

Výpis 4.18: Seznam sdílených knihoven využívaných prohlížečem

Počet těchto knihoven se rovná 100. Takový počet je způsoben tím, že sdílené knihovny samy o sobě mohou být závislé na jiných sdílených knihovnách, jak je ukázáno na výpisu 4.19 na příkladu knihovny libQt5Core.so.5.

```
[root@localhost  /lib64/]# readelf -d libQt5Core.so.5 | grep 'NEEDED'
Shared library: [libpthread.so.0]
Shared library: [libglib-2.0.so.0]
Shared library: [libstdc++.so.6]
Shared library: [libm.so.6]
Shared library: [libgcc_s.so.1]
Shared library: [libc.so.6]
Shared library: [ld-linux-x86-64.so.2]
Shared library: [libsystemd.so.0]
...
```

Výpis 4.19: Závislosti sdílených knihoven na jiných sdílených knihovnách

Seznam poskytnutý příkazem `ldd` zajistí nejen jména knihoven, ale i cesty, kde se nacházejí v operačním systému. Tato informace pomáhá znesnadnit práci jejich vyhledávání v jednotlivých adresářích. Pokud některá z těchto knihoven nebude na ukazované cestě nalezená, spuštění grafického prohlížeče selže. Pro zaručení, že všechny tyto sdílené knihovny budou v operačním systému, nad kterým bude prováděná minimalizace, byl vypracován skript uvedený na výpisu 4.20.

```
#!/bin/bash

browser=$1
browser_zip="$browser.zip"

zip $browser_zip $browser

ldd $browser | while read line; do

        library=`echo $line | grep -o "/.*/.*\\s"`
        zip -u $browser_zip $library

done
```

Výpis 4.20: Skript přidaní spustitelného souboru a sdílených knihoven do archivu

Tento skript je zodpovědný za přidání spustitelného souboru prohlížeče spolu se všemi jeho závislými sdílenými knihovnami do zip archivu. Ve své první časti přijímá jako argument samotný spustitelný soubor a zapisuje do proměnné **browser**. Druhá část se zabývá rozparsovaním výpisu příkazu `ldd` pro tento prohlížeč řádek po řádku. Z každého řádku dostane cestu do jednotlivé knihovny, která bude přiřazena proměnné **library**. V následujícím kroku tuto cestu spolu s knihovnou uloží do archivu. Velikost tohoto archivu na konci činí 61 MB.

4.7 Instalace grafického prohlížeče v minimalizovaném systému

Spuštění grafického rozhraní pomocí X Window lze rozdělit do dvou částí: spuštění serveru a spuštění klienta.

Jako X-Server implementace se v moderních linuxových distribucích používá „X.org“ [13]. Jeho balíček `xorg-x11-server-Xorg` není nainstalován ve výchozím nastavení v CentOS, takže je nutné tento balíček nejprve stáhnout a nainstalovat. Navíc je potřeba stáhnout balíček `xinit`, který odpovídá za spuštění serveru (viz výpis 4.21). Jestliže je server spuštěn bez klienta, zobrazí se pouze šedá obrazovka s charakteristickým křížkem kurzoru uprostřed. Pomocí myši se tento křížek může pohybovat po obrazovce. Při stisknutí tlačítek myši a kláves by neměla nastat žádná viditelná reakce. Jak už bylo popsáno v podkapitole 3.1, je to způsobeno tím, že samotný server nevykresluje žádné obrazy, ale pouze doručí žádost o grafiku do grafického adaptéru a odesílá zprávy o událostech z hardwaru svým klientům, které ještě nejsou spuštěny.

```
[root@localhost ~]# dnf -y install xorg-x11-server-Xorg xinit
```

Výpis 4.21: Instalace X-Windows serveru

Jako klient pro X-Server v této práci vystupuje grafický minimalistický prohlížeč Otter-browser. V podkapitole 4.6 bylo řečeno, že spustitelný soubor tohoto prohlížeče byl společně se všemi jeho zdroji vložen do zip archivu a nahrán na veřejný FTP server. Jako FTP server byl zvolen GoFile, který umožňuje pomocí odkazu stáhnout jakýkoliv soubor, aniž by bylo nutné zadávat přihlašovací údaje. Zároveň podporuje pokročilé bezpečnostní mechanismy, což zaručuje, že tento soubor nebude podléhat změnám, zatímco bude na tomto serveru uložen. Pro zaručení přítomnosti X-Server klienta na operačním systému musí být nejprve stažen a rozbalen archiv prohlížeče (viz výpis 4.22). K tomuto účelu slouží nástroje `wget` a `unzip`, kde prvnímu z nich je poskytován odkaz na umístění archivu na FTP serveru. Nástroji zip je zadána cesta, kam musí být tento archiv rozbalen, a tou je kořenový adresář. Tato možnost byla

zvolena kvůli tomu, že soubory nacházející se v zip archivu obsahují cesty, které bude prohlížeč vyhledávat najednou, a všechny tyto cesty začínají v kořenovém adresáři. Po provedení tohoto postupu nebudou nástroje a stejně tak i zip archiv potřebné, a proto mohou být pro ušetření místa odstraněny.

```
[root@localhost ~]# dnf -y install wget zip
[root@localhost ~]# wget --no-check-certificate https://srv-file4.gofile
.io/download/ob5QFm/otter-browser.zip
[root@localhost ~]# echo "A" | unzip otter-browser.zip -d /
[root@localhost ~]# rm -f otter-browser.zip
[root@localhost ~]# dnf -y remove wget zip
```

Výpis 4.22: Instalace X-Windows klienta

Kromě sdílených knihoven a spustitelného souboru prohlížeče Otter-browser musí být pro jeho spuštění v operačním systému zajištěna další závislost, která byla zmíněna v podkapitole 3.2, a tou je webkit. Otter-browser využívá renderovací jádro založené na knihovnách Qt a z tohoto důvodu je nutné nainstalovat programový balíček s názvem `qt5-qtwebkit` (viz výpis 4.23). Tento balíček není v oficiálních repozitářích CentOS, a proto je v první řadě nezbytné povolit a nainstalovat repozitář třetí strany „epel-release“, odkud je možné jej stáhnout. Uvedené datové úložiště je vytvořeno a udržováno komunitou podporující linuxovou distribuci Fedora a slouží k účelu poskytovat její různé programové balíčky na další s ní kompatibilní distribuce [30]. Mezi nimi jsou Red Hat Enterprise Linux a na jeho otevřených zdrojových kódech postavený CentOS.

```
[root@localhost ~]# dnf -y install https://dl.fedoraproject.org/pub/
epel/epel-release-latest-8.noarch.rpm
[root@localhost ~]# dnf -y install qt5-qtwebkit
```

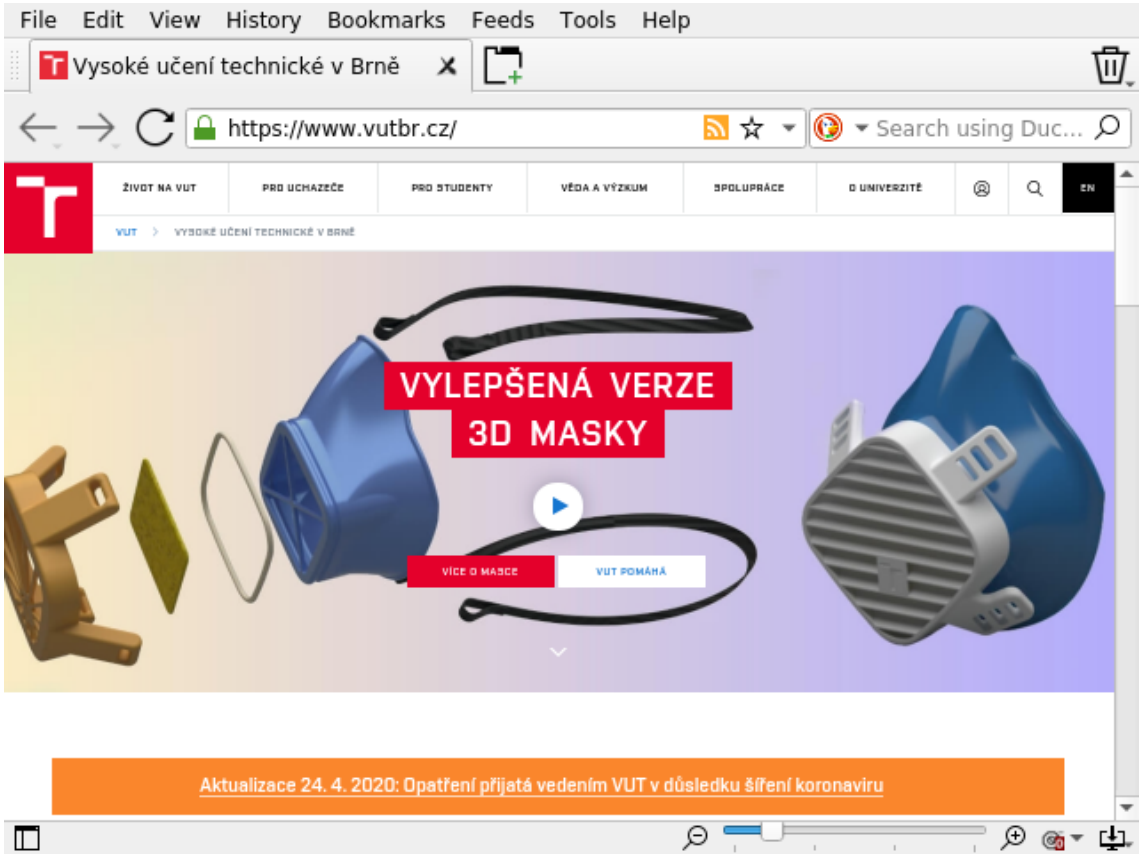
Výpis 4.23: Instalace renderovacího jadra QtWebKit

Předpokladem pro spuštění X Windows klienta po provedení všech popsáných kroků je existence souboru *.xinitrc* v domovském adresáři uživatele systému, ve kterém je ukázán grafický program (nebo cesta do jeho spustitelného souboru), který má být spuštěn. Navíc byl jako parametr přidán webový odkaz pro zobrazení aktuální stránky univerzity VUT v Brně, jak je ukázáno na výpise 4.24.

```
[root@localhost ~]# touch ~/.xinitrc
[root@localhost ~]# echo "/otter-browser https://vutbr.cz"> ~/.xinitrc
```

Výpis 4.24: Příprava konfiguračního souboru pro spuštění grafického prohlížeče

Po spuštění příkazu **xinit**, nejsou-li předány žádné parametry, začne hledat v domovském adresáři soubor *.xinitrc* a spustí jej jako skript [31]. Tím se spustí grafický



Obr. 4.1: Grafický prohlížeč v minimalizovaném systému

prohlížeč. Výsledek lze pozorovat na obrázku 4.1

4.8 Shrnutí výsledků minimalizace

Výsledná velikost linuxové distribuce CentOS na paměťovém médiu po provedení všech kroků minimalizace uvedených v Bash skriptu je 731 MB. Z této hodnoty připadá 496 MB na samotný operační systém, zbývajících 235 MB zabírá grafický minimalisticky prohlížeč se všemi svými zaručenými závislostmi, přičemž největší závislostí je programový balíček `qt5-qtwebkit` o velikosti 197 MB.

V tabulce 4.2 je možné vidět výsledek minimalizace konkretních adresářů, které byly zmíněny v této práci. Z tabulky vyplývá, že nejvíce minimalizován je adresář pro programy a prostředky /usr a jeho podadresáře /lib/firmware, /lib/modules a /lib/share. Stojí za poznamenání, že taková výsledná hodnota v případě podadresáře /lib/firmware je způsobena tím, že ovladače na mém systému nevyžadují mnoho mikroprogramového vybavení (firmware) pro práci s technickým vybavením mého počítače. V ostatních systémech se tato hodnota může lišit, stejně to platí i pro /lib/modules. Adresář /var je minimalizován nejméně ze všech. Logovací soubory na mém systému nezabírají moc místa na paměťovém médiu, jelikož můj systém neod-

povídá za provoz žádných služeb a nevytváří tak velké množství událostí, které by byly zapisovány do logovacích souborů. Odstranění těchto souborů pomohlo uvolnit jen malé množství megabajtů prostoru. Vytvoření nové databáze balíčkovacího systému `rpm` rovněž neumožnilo tento adresář příliš minimalizovat. Podadresář /cache byl redukován úplně a má konečnou velikost 0 MB. Jakmile bude v systému provedena jakákoli operace s balíčkovacími systémy, stáhnou se do mezipaměti veškerá data spojená s datovými úložišti a podadresář bude mít opět téměř stejnou velikost jako předtím. Proto i programové řešení jeho minimalizace se nachází skoro na konci Bash skriptu (po provedení všech operací s balíčkovacími systémy).

Adresář	Před redukcí [MB]	Po redukci [MB]
/boot	124	60
/var/cache	20	0
/var/lib	57	53
/var/log	7	3,3
/usr/share	247	62
/usr/lib/firmware	308	0,012
/usr/lib/modules	53	17

Tab. 4.2: Výsledky minimalizace adresářů

Výsledek redukce počtu programových balíčků je zobrazen v tabulce 4.3, ze které vyplývá, že v linuxové distribuci se nachází celkem 169 osamělých programových balíčků, které se podařilo odstranit (s výjimkou balíčku `NetworkManager`). To je skoro polovina všech balíčků nainstalovaných v systému. Po provedení tohoto kroku se velikost CentOS zmenšila o 339 MB.

	Před redukcí	Po redukci
Celkový počet balíčků	391	222
Velikost operačního systému	1278 MB	939 MB

Tab. 4.3: Výsledky redukce počtu programových balíčků

Závěr

Bakalářská práce realizuje minimalizaci linuxové distribuce CentOS. Pro automatizaci minimalizace bylo vytvořeno programové řešení v jazyce Bash. Rovněž byla představena metoda pro snížení počtu softwarových balíčků, které jsou nainstalovaný ve výchozím nastavení CentOS typu Minimal. Díky ní se podařilo snížit jejich počet z 391 na 222. Výsledná hodnota minimalizace, nezahrnující velikost místa na disku, které je obsazeno grafickým prohlížečem, tak činí 496 MB ve srovnání s 1278 MB při instalaci. Nejvíce redukovaný byl adresář odpovídající za umístění mikroprogramového vybavení, který byl minimalizován z 308 MB na 0,012 MB. Naopak nejméně minimalizován byl adresář pro dočasné soubory, který se podařilo redukovat jen o několik megabajtů.

Za účelem zachování provozu grafického webového prohlížeče a požadovaného zobrazení titulní stránky univerzity VUT v Brně v nezměněné podobě byly porovnány známé minimalistické grafické prohlížeče obsahující pokročilé funkce. Na základě nejmenší velikosti byl zvolen prohlížeč Otter-Browser. Z důvodu nepřítomnosti jeho programového balíčku v repozitářích CentOS a jiných kompatibilních repozitářích byl uveden a podrobně rozebrán způsob jeho zajištění v systému pomocí kompilace zdrojových kódů. Aby nezvětšoval velikost linuxové distribuce nástroji a zase neprováděl kompilace na cílové platformě, spustitelný soubor spolu se všemi nezbytnými pro prohlížeč zdroje byly vloženy do zip archivu a nahrány na veřejný FTP server. Tento archiv bude automatický stažen v průběhu minimalizace z odkazu `https://srv-file12.gofile.io/download/lWjQJK/otter-browser.zip`. Koneční velikost minimalizovaného operačního systému po zaručení prohlížeče a instalaci extra programového balíčku poskytujícího vykreslovací jádro prohlížeče je 731 MB.

Vytvořený skript provede minimalizaci bez jakéhokoliv zásahu uživatele a po jejím skončení se otevře grafický prohlížeč. Tento skript byl zveřejněn pod licencí MIT v osobním repozitáři na GitHub a je k dispozici prostřednictvím odkazu `https://github.com/PavelVashkevich/CentOS-Reduction-Bash`.

Literatura

[1] LIU, Shanhong. *Global market share held by operating systems for desktop PCs, from January 2013 to January 2019* [online]. 2019 [cit. 2019-12-15]. Dostupné z: `https://www.statista.com/statistics/218089/global-market-share-of-windows-7/`

[2] STODDEN, Victoria, Friedrich LEISCH a Roger D. PENG, ed. *Implementiong Reproductible Research.* New York: CRC Press, 2018. ISBN 978-1-4665-6159-5.

[3] SMITH, Jesse. *Review: CentOS 8.0-1905* [online]. 2019 [cit. 2019-12-16]. Dostupné z: `https://distrowatch.com/weekly.php?issue=20191021`

[4] Filesystem Hierarchy Standard Group, RUSSELL, Rusty, Daniel QUINLAN a Christopher YEOH, ed. *Filesystem Hierarchy Standard* [online]. 2004 [cit. 2019-12-07]. Dostupné z: `http://www.pathname.com/fhs/pub/fhs-2.3.pdf`

[5] COBBAUT, Paul. *Linux Fundamentals* [online]. 2015 [cit. 2019-12-06]. Dostupné z: `http://linux-training.be/linuxfun.pdf`

[6] NEMETH, Evi, Garth SNYDER, Trent R. HEIN a Alessandro RUBINI. *Linux: kompletní příručka administrátora.* 3rd ed. Brno: Computer Press, 2004. Administrace (Computer Press). ISBN 80-722-6919-4.

[7] ROBBINS, Daniel, Chris HOUSER a Aron GRIFFIS. *Linux Fundamentals, Part 2* [online]. 2014 [cit. 2019-12-08]. Dostupné z: `https://www.funtoo.org/Linux_Fundamentals,_Part_2`

[8] LABASTIE, Pierre, Ken MOFFAT a Bruce DUBBS, R. RENO, Douglas a DJ LUCAS, ed. *Beyond Linux From Scratch* [online]. Revision 9.0. 2019 [cit. 2019-12-06]. Dostupné z: `www.linuxfromscratch.org/blfs/view/stable/`

[9] HENDERSON, Bryan. *Linux Loadable Kernel Module HOWTO* [online]. Revision v1.09. 2006 [cit. 2020-05-31]. Dostupné z: `https://www.tldp.org/HOWTO/pdf/Module-HOWTO.pdf`

[10] CORBET, Jonathan, Alessandro RUBINI, Greg KROAH-HARTMAN a Alessandro RUBINI. *Linux device drivers.* 3rd ed. Sebastopol, CA: O'Reilly, 2005. ISBN 05-960-0590-3.

[11] Управление пакетами. КУРЯЧИЙ, Георгий a Кирилл МАСЛИНСКИЙ. Операционная система Linux. Курс лекций. 2-е изд. Москва: ДМК Пресс, 2010, s. 222-233. ISBN 978-5-94074-591-4.

[12] HOYER, Harald. *Dracut* [online]. Revision 3.0. 2013 [cit. 2019-12-08]. Dostupné z: https://mirrors.edge.kernel.org/pub/linux/utils/boot/dracut/dracut.html

[13] SOBELL, Mark G. *Mistrovství v RedHat a Fedora Linux: pro verze Fedora Core 2 až 5 a RedHat 3 a 4.* Brno: Computer Press, 2006. ISBN 80-251-1152-0.

[14] FOSTER-JOHNSON, Eric, Stuart ELLIS a Ben COTTON. *Fedora Draft Documentation: RPM Guide* [online]. [cit. 2019-12-08]. Dostupné z: https://docs.fedoraproject.org/en-US/Fedora_Draft_Documentation/0.1/pdf/RPM_Guide/Fedora_Draft_Documentation-0.1-RPM_Guide-en-US.pdf

[15] WALDEN, Chris. *Part 9. Installing software: Using pre-compiled RPMs and compiling applications from source* [online]. 2003 [cit. 2019-12-14]. Dostupné z: https://www.ibm.com/developerworks/linux/library/l-roadmap9/?S_TACT=105AGX99&S_CMP=CP

[16] BAILEY, Edward C. *Maximum RPM.* Indianapolis, Ind.: Redhat Press, c1997. ISBN 06-723-1105-4.

[17] CentOS 8 Documentation. *Using AppStream* [online]. [cit. 2019-12-14]. Dostupné z: https://docs.centos.org/en-US/8-docs/managing-userspace-components/assembly_using-appstream/

[18] *DNF Documentation* [online]. 2019 [cit. 2019-12-12]. Dostupné z: https://readthedocs.org/projects/dnf/downloads/pdf/stable/

[19] CentOS documentation. *CentOS GPG Keys* [online]. [cit. 2019-12-09]. Dostupné z: https://www.centos.org/keys/

[20] MATTHEW, Neil a Richard STONES. *Beginning Linux Programming.* Canada: John Wiley, 2011. ISBN 978-0-470-14762-7.

[21] BARR, Michael a Anthony MASSA. *Programming Embedded Systems.* 2nd. O'Reilly Media, 2006. ISBN 0596009836.

[22] MANRIQUE, Daniel. *X Window System Architecture Overview HOWTO* [online]. Revision 1.0. 2001 [cit. 2019-12-07]. Dostupné z: https://www.tldp.org/HOWTO/XWindow-Overview-HOWTO/index.html

[23] GETTYS, James, Robert W.SCHEIFLER a The Open Group. *Xlib - C Language X Interface: X Window System Standard* [online]. Tektronix [cit. 2020-05-31]. Dostupné z: https://www.x.org/docs/X11/xlib.pdf

[24] ШЛЕЕ, Максим. Qt4.5. Профессиональное программирование на C++. СПб: БХВ-Петербург, 2010. ISBN 978-5-9775-0398-3.

[25] J. SÁNCHEZ, Juan. *WebKit and Blink: Open DevelopmentPowering the HTML5 Revolution* [online]. New Orleans: LinuxCon, 2013 [cit. 2020-05-03]. Dostupné z: `https://events.static.linuxfound.org/sites/events/files/slides/slides.pdf`

[26] TROAN, Erik, Preston BROWN a Jan KALUZA. *Logrotate(8) - Linux man page* [online]. [cit. 2019-12-08]. Dostupné z: `https://linux.die.net/man/8/logrotate`

[27] The NetworkManager Authors. *NetworkManager Reference Manual* [online]. Boston, USA: The Free Software Foundation, 2018 [cit. 2020-05-03]. Dostupné z: `https://developer.gnome.org/NetworkManager/stable/`

[28] WARD, Brian. *How Linux works: what every superuser should know.* Ilustrované vydání. San Francisco: No Starch Press, c2004. ISBN 15-932-7035-6.

[29] BAST, Radovan a Roberto DI REMIGIO. *CMake Cookbook: Building, testing, and packaging modular software with modern CMake.* Livery Place: Packt Publishing, 2018. ISBN 978-1-78847-071-1.

[30] SINGER, David. *How to Enable an EPEL repository* [online]. [cit. 2020-05-30]. Dostupné z: `https://www.liquidweb.com/kb/enable-epel-repository/`

[31] SCHEIFLER, Bob. *XINIT(1) manual page* [online]. [cit. 2019-12-08]. Dostupné z: `https://www.x.org/archive/X11R6.8.1/doc/xinit.1.html`

Seznam symbolů, veličin a zkratek

MBR	Master boot record
RAM	Random Access Memory
MB	Megabajt
kB	Kilobajt

Seznam příloh

A Obsah přiloženého CD

Na přiloženém médiu se nachází elektronická verze bakalářské práce. Kromě toho CD obsahuje zdrojový kód vytvořeného skriptu Bash ve formátu .sh. Tento skript je také určen ke spuštění pro minimalizaci operačního systému.

```
/ ..... kořenový adresář přiloženého CD
└── Bakalářská práce Pavel Vashkevich
    ├── Bakalářská práce Pavel Vashkevich.pdf ..... elektronická verze BP
    └── minimize.sh.....zdrojový kód skriptu Bash
```