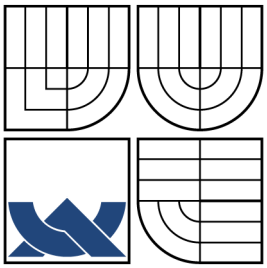
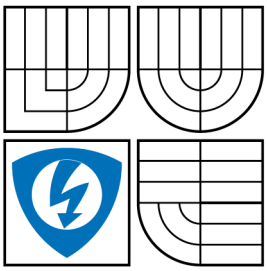


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND
COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

AUTOMATIZOVANÝ SYSTÉM PRO INSTALACE VIRTUÁLNÍCH STROJŮ V PROSTŘEDÍ OPERAČNÍHO SYSTÉMU GNU/LINUX

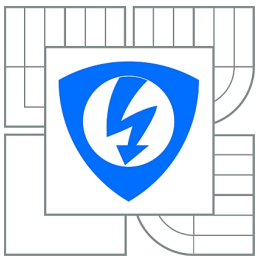
AUTOMATED SYSTEM FOR INSTALLING AND CONFIGURING GNU/LINUX VIRTUAL HOSTS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. DANIEL HORÁK
AUTHOR

VEDOUCÍ PRÁCE — Ing. TOMÁŠ PELKA
SUPERVISOR

BRNO 2010



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Diplomová práce

magisterský navazující studijní obor
Telekomunikační a informační technika

Student:	Bc. Daniel Horák	***ID:***	73030
Ročník:	2	***Akademický rok:***	2010/2011

NÁZEV TÉMATU:

Automatizovaný systém pro instalace virtuálních strojů v prostředí operačního systému GNU/Linux

POKYNY PRO VYPRACOVÁNÍ:

Úkolem studenta je v diplomové práci prostudovat možnosti vzdáleného volání procedur a navrhnout architekturu automatizovaného systému pro instalaci a konfiguraci virtualizovaných strojů pro potřeby výuky předmětu MNSB. Architerktura bude dostatečně modulární a rozšiřitelná tak, aby výsledný produkt byl použitelný i pro potřeby jiných laboratoří. Přesná specifikace systému bude součástí textu semestrálního projektu.

DOPORUČENÁ LITERATURA:

[1] GOERZEN, John. Foundations of Python Network Programming. [s.l.] : Apress, 2004. 512 s. ISBN 978-1590593714

[2] JONES, Christopher. Python & XML. 1. [s.l.] : O'Reilly Media, 2001. 450 s. ISBN 978-0596001285

Termín zadání:	7.2.2011	***Termín odevzdání:***	26.5.2011

Vedoucí práce: Ing. Tomáš Pelka

prof. Ing. Kamil Vrba, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

ABSTRAKT

Práce se zabývá možnostmi správy virtuálních strojů (především z programátorského hlediska) a dostupnými metodami pro automatickou instalaci distribucí operačního systému GNU/Linux. Úvodní část je věnována virtualizačním technologiím a možnostem jejich ovládání s využitím knihovny *libvirt*. Je zde popsán způsob definice a popisu jednotlivých součástí virtualizovaného prostředí (virtuální stroje, disky, sítě) za pomoci XML. Dále jsou probrány možnosti automatické instalace linuxových distribucí založených na RedHatu a Debianu s přihlédnutím k virtualizovanému prostředí. Je zde zmíněna možnost kopírování existujícího disku s nainstalovaným systémem. V praktické části je navržena a realizována serverová aplikace pro ovládání a správu virtuálních strojů včetně automatické instalace operačního systému.

KLÍČOVÁ SLOVA

Virtualizace, GNU/Linux, automatická instalace, Libvirt, KVM, XML–RPC

ABSTRACT

This work deals with possibilities of administration of virtual machines (mainly from the developer's point of view) and available methods for automatic installation of GNU/Linux distributions. First part contains the information about virtualisation technologies and possibilities to configure them with a libvirt library. The definition and specification of each part of virtualisation environment (virtual machines, disks, networks) using XML is described. Further more the possibilities of automatic installation of Linux distributions based on RedHat and Debian considering the virtualisation environment are discussed. The possibility of copying an existing disk with installed system is also mentioned. In a practical part a server application for control and management of virtual machines including automatic installation of operating system is proposed and realised.

KEYWORDS

Virtualization, GNU/Linux, automatic installation, Libvirt, KVM, XML–RPC

HORÁK, Daniel *Automatizovaný systém pro instalace virtuálních strojů v prostředí operačního systému GNU/Linux*: diplomová práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2010. 77 s. Vedoucí práce byl Ing. Tomáš Pelka

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Automatizovaný systém pro instalace virtuálních strojů v prostředí operačního systému GNU/Linux“ jsem vypracoval samostatně pod vedením vedoucího diplomové 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 diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

V Brně dne ....

(podpis autora)

Děkuji vedoucímu své diplomové práce, Ing. Tomáši Pelkovi, za odborné vedení a poskytování cenných rad a podnětů při zpracování diplomové práce.

V Brně dne ....

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

ÚVOD

Virtualizace je oblast výpočetní techniky, která v posledních letech stále více získává popularitu. Výkon hardware roste rychleji, než je možné jej v běžných aplikacích využít. Virtualizace tak představuje zajímavý způsob využití výkonu jednoho počítače k více různým úlohám rozděleným v podstatě již na úrovni hardwaru (v některých případech emulovaného).

Při výuce různých předmětů zabývajících se instalací a správou operačních systémů je vhodné, aby každý student měl k dispozici vlastní počítač pro výuku. Tento požadavek je v případě použití fyzického stroje velice těžko splnitelný, virtualizace však umožňuje docílit téměř shodného výsledku s podstatně menšími náklady. Každému studentovi je nutné přidělit pouze část kapacity datového úložiště a část výkonu počítače, na kterém virtuální stroj běží. Na začátku semestru pak samozřejmě vyvstane otázka, jak nejjednodušeji připravit velké množství (desítky) virtuálních strojů (pro každého studenta).

Virtualizačních nástrojů existuje velké množství a každý má svůj vlastní způsob ovládání. Sjednocení přístupu hlavně z programátorského hlediska k jednotlivým typům virtualizačních řešení přináší *libvirt*.

Na připravené stroje je potřeba nainstalovat operační systém. Přístup k automatické instalaci řeší, více či méně úspěšně, většinou každá distribuce samostatně v závislosti na použitém instalačním programu.

Cílem práce je zjištění možností, navržení a realizace aplikace hromadné správy virtuálních strojů a automatické instalace linuxových distribucí za účelem zjednodušení přípravy technického zázemí pro výuku předmětu Návrh, správa a bezpečnost počítačových sítí.

1 VIRTUALIZACE, KVM A LIBVIRT

Kernel-based Virtual Machine – virtualizace na úrovni jádra (KVM) je řešení plné virtualizace pro Linux a x86 hardware obsahující virtualizační rozšíření (Intel VT nebo AMD-V). Skládá se ze zaveditelného jaderného modulu *kvm.ko*, který poskytuje jádro virtualizační infrastruktury a specifický modul pro konkrétní rodinu procesorů *kvm-intel.ko* nebo *kvm-amd.ko*. KVM také požaduje upravený QEMU.

Na virtuálních strojích využívajících KVM lze provozovat nemodifikované operační systémy GNU/Linux nebo MS Windows. KVM je součástí hlavní větve Linuxu od verze 2.6.20.[7]

libvirt je sada nástrojů umožňující přístup a práci (vytváření, úprava, monitorování, kontrola, zastavení,...) s mnoha různými virtualizačnímy technologiemi v operačním systému Linux (a ostatních operačních systémech). Jde o svobodný software dostupný pod GNU Lesser General Public License[1]. Nabízí rozhraní pro nejpoužívanější programovací jazyky a podporuje následující virtualizační nástroje:[10]

- Xen hypervisor (Linux a Solaris)
- QEMU emulátor
- KVM Linux hypervisor
- LXC Linux container system
- OpenVZ Linux container system
- User Mode Linux paravirtualized kernel
- VirtualBox hypervisor
- VMware ESX and GSX hypervisory
- Úložiště na IDE/SCSI/USB discích, FibreChannel, LVM, iSCSI, NFS a souborovém systému.

libvirt poskytuje:

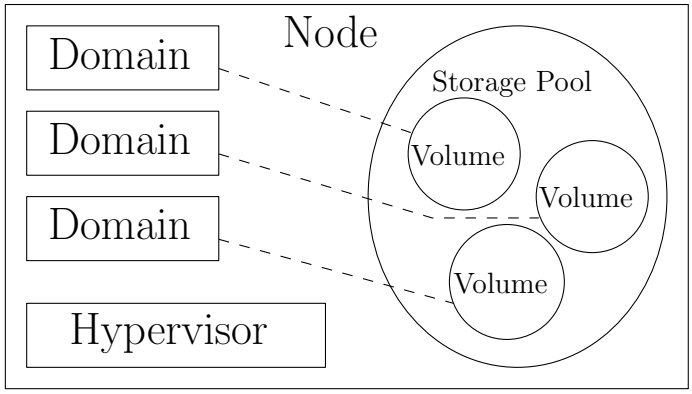
- vzdálenou správu s využitím Transport Layer Security – zabezpečení transportní vrstvy (TLS) šifrování, x509 certifikáty a autentizací pomocí Kerberos a SASL
- lokální kontrolu přístupu používající PolicyKit
- správu virtuálních strojů, virtuálních sítí a datových úložišť
- přenositelné API pro klienty na operačních systémech Linux, Solaris a Windows

[1] <http://www.opensource.org/licenses/lgpl-license.html>

1.1 Použitá terminologie

V dokumentaci ke knihovně *libvirt* se používají následující termíny (souvislosti mezi nimi jsou zobrazeny na obrázku 1.1).

- **Node** (uzel) – jeden fyzický stroj,
- **Hypervisor** – softwarová vrstva umožňující uzlu virtualizaci různých virtuálních strojů i s rozdílnou konfigurací,
- **Domain** (doména) – instance operačního systému spuštěná na virtualizovaném stroji poskytnutém hypervisorem,
- **Volume** – prostor pro uložení dat představující virtuální pevný disk, cd-rom,...
- **Storage Pool** – prostor pro shromažďování pevných disků a ostatních úložišť (*Volume*).



Obr. 1.1: Vzájemná souvislost mezi používanými termíny v knihovně *libvirt*.

1.2 Koncept *libvirt* API

Základní objekt, který přináší *libvirt* API, je `virConnectPtr` a představuje spojení s hypervisorem. Všechny aplikace využívající *libvirt* při svém startu volají některou z funkcí pro připojení k hypervisoru, který je jednoznačně určen pomocí Uniform Resource Identifier – jednotný identifikátor zdroje (URI), kterému se budeme podrobněji věnovat dále v textu. Spojení s hypervisorem je dále využíváno pro vytváření a získávání dalších objektů, kterými jsou:

- **virDomainPtr** – reprezentuje jednu konkrétní doménu,
- **virNetworkPtr** – reprezentuje jednu konkrétní virtuální síť,

- **virStorageVolPtr** – reprezentuje konkrétní úložiště dat (*volume*) připojitelné k doméně (např. pevný disk, cd-rom,...),
- **virStoragePoolPtr** – reprezentuje *storage pool*.

K identifikování těchto objektů máme několik možností:

- **name** (jméno) – uživatelsky přívětivý název, u kterého ale není zaručena unikátnost mezi dvěma uzly,
- **ID** – unikátní identifikátor poskytnutý hypervisorem za dobu běhu, po zrušení objektu ztrácí platnost,
- **UUID** – je 16ti bytový identifikátor definovaný v RFC 4122[22], u kterého je zaručena dlouhodobá unikátnost napříč různými uzly.

1.3 *libvirt* URI

Pro odlišení různých typů virtualizačních technologií a určení konkrétního hypervisora používá *libvirt* URI (RFC 2396[21]). Tento identifikátor se předává například jako parametr funkcím nebo programům pro připojení ke konkrétnímu hypervisoru.

Obecný formát lokální URI pro *libvirt* může mít jednu z následujících podob:

```
driver:///system
driver:///session
driver+unix:///system
driver+unix:///session
```

Položka `driver` představuje název ovladače pro konkrétní hypervisor. Mezi podporované typy hypervisory patří například:

- **qemu** – pro ovládání *qemu* a KVM,
- **xen** – pro ovládání starších verzí *Xen*u (*Xen* 3.1 a starší),
- **xenapi** – pro ovládání novějších verzí *Xen*u,
- **vbox** – pro ovládání *VirtualBox*u,
- **esx** – pro ovládání *VMware ESX*.

Další možnosti jsou popsány v dokumentaci[18].

Pro vzdálené připojení je potom formát URI následující:

```
driver[+transport]://[username@][hostname] ↩
                    [:port]/[path][?extraparameters].
```

Kde jednotlivé položky mají následující význam:

- **driver** – stejné jako u lokální URI,

- **transport** – použitá transportní vrstva (možné hodnoty: tls, tcp, unix, ssh a ext),
- **username** – v případě použití SSH jako transportní vrstvy, určuje tato volba uživatelské jméno,
- **hostname** – úplné jméno vzdáleného stroje,
- **port**– změna standardního portu (22 pro SSH, 16509 pro TCP a 16514 pro TLS),
- **path** – cesta může být stejná jako v případě lokální URI,
- **extraparameters** – URI parametry ke specifikování některých dalších vlastností vzdáleného připojení.

1.4 *libvirt* XML

Pro vytváření a definici domén, sítí a dalších prvků využívaných doménami se používá Extensible Markup Language – rozšiřitelný značkovací jazyk (XML) předpis. V následujících oddílech budou popsány základní konstrukce XML souboru pro konkrétní prvek. Schéma pro validaci jsou ve formátu RELAX NG a jsou uloženy v adresáři `/usr/share/libvirt/schemas/` (platí pro distribuci Debian).

1.4.1 Domény

Kořenový element XML definujícího doménu je <`domain`>. Povinným atributem tohoto elementu je `type` definující typ hypervisora. Na výběr máme z následujících možností: `xen`, `kvm`, `kqemu`, `qemu`, `lxc`, `openvz` případně `test`.

Základní metadata

name Element <`name`> je první povinný vnořený element obsahující jméno domény. Může obsahovat pouze malá a velká písmena a číslice plus několik málo dalších znaků jako je podtržítko, pomlčka nebo tečka.

uuid Element <`uuid`> je nepovinným elementem následujícím po <`name`>. Pokud je uvedeno, přiřadí se vytvářené doméně specifikované UUID, pokud uvedeno není, je náhodně vygenerováno.

description <`description`> je dalším volitelným elementem s popisem daného virtuálního stroje.

```
<domain type='kvm'>
  <name>Test</name>
  <uuid>1ec85053-0538-4440-72fd-3aca641a2b12</uuid>
  <description>Popis testovaciho stroje.</description>
  . . .
```

Start operačního systému

Pro start operačního systému uvnitř virtuálního stroje jsou dostupné tři různé způsoby.

- **BIOS bootloader** – Start systému je nejvíce podobný startu normálního počítače. Hypervisor emuluje BIOS, který má na starosti vyhledání a zavedení operačního systému. Stejně jako u běžného počítače existuje i zde seznam dostupných médií seřazený podle priority, na kterém zařízení se má hledat zavaděč operačního systému. Podrobnosti k nastavení jsou v tabulce 1.1. Všechny elementy jsou uvnitř tagu <os>.

```
  . . .
  <os>
    <type arch='i686' machine='pc-0.12'>hvm</type>
    <boot dev='hd'/>
    <boot dev='cdrom'/>
    <bootmenu enable='no'/>
  </os>
  . . .
```

- **Host bootloader** – Hypervisor provádějící paravirtualizaci obvykle neemuluje BIOS, namísto toho se pro zahájení startu systému použije pseudo-bootloader, který zajistí výběr jádra a započetí startu systému. Možnosti konfigurace jsou v tabulce 1.2.

```
  . . .
  <bootloader>/usr/bin/pygrub</bootloader>
  <bootloader_args>--append single</bootloader_args>
  . . .
```

- **Direct kernel boot** (přímý boot jádra) – Tato možnost je obvykle přístupná pro para i plně virtualizované hosty. V konfiguraci domény je nastavena cesta k jádru operačního systému, které má být zavedeno. Tento způsob umožňuje

předat parametry jádra startujícímu systému, což může být v mnoha případech užitečné, například pro nastartování automatické instalace. Všechny potřebné elementy jsou uvedeny v tabulce 1.3. Stejně jako v prvním případě i zde jsou všechny elementy uvnitř tagu <os>.

```
. . .
<os>
  <type>hvm</type>
  <loader>/usr/lib/xen/boot/hvmloader</loader>
  <kernel>/root/f8-i386-vmlinuz</kernel>
  <initrd>/root/f8-i386-initrd</initrd>
  <cmdline>console=ttyS0 ks=http://example.com/f8-↩
      i386/os/</cmdline>
</os>
. . .
```

<type>	Obsah elementu <type> specifikuje typ bootovaného operačního systému. - **hvm** Operační systém je určen pro běh na fyzickém stroji (tedy není nijak upraven pro běh ve viruálním prostředí). - **linux** (špatně pojmenováno!) označuje operační systém podporující *Xen 3 hypervisor guest ABI*. Element dále obsahuje dva další volitelné atributy `arch` (architektura CPU) a `machine` (typ stroje)
<loader>	Volitelný element ukazuje na firmware přispívající k vytváření domény (v tuto chvíli potřebný pouze pro plnou virtualizaci s využitím Xenu).
<boot>	Určuje médium na kterém se má hledat zavaděč systému. Element <boot> může být použit vícekrát pro určení více médií v určitém pořadí. Může obsahovat jednu z následujících možností: `fd`, `hd`, `cdrom` nebo `network`.
<bootmenu>	Volitelný element určující atributem `enable` zda se má zobrazit interaktivní menu pro výběr bootovacího média.

Tab. 1.1: Elementy pro nastavení startu systému za pomoci BIOS bootloaderu

<bootloader>	Úplná cesta ke spustitelnému bootloaderu, který má na starosti výběr jádra pro boot.
<bootloader_args>	Volitelný element pro přidání parametrů bootloaderu.

Tab. 1.2: Elementy pro nastavení startu systému za pomoci pseudo-bootloaderu

<type>	stejné jako v případě BIOS bootloaderu
<loader>	stejné jako v případě BIOS bootloaderu
<kernel>	úplná cesta k obrazu jádra hostovaného systému
<initrd>	volitelný element obsahující úplnou cestu k obrazu ramdisku
<cmdline>	parametry předané jádru hostovanému systému

Tab. 1.3: Elementy pro nastavení startu systému za pomoci direct kernel boot

Základní prostředky

Po určení způsobu startování operačního systému je potřeba nastavit základní parametry virtuálního stroje jako je paměť (maximální a aktuální) či počet virtuálních procesorů.

memory Maximální velikost alokované paměti při startu počítače. Jednotkou zadané hodnoty jsou kilobyty.

currentMemory Aktuální velikost alokované paměti, musí být menší nebo rovna maximální velikosti paměti. Pokud není uvedeno, použije se stejná hodnota jako u <memory>.

vcpu Element <vcpu> definuje počet virtuálních procesorů. Minimální hodnota je 1. Maximální hodnota závisí na konkrétním hypervisoru. Volitelným atributem tohoto elementu je `cpuset` obsahující čárkami oddělený seznam čísel fyzických procesorů, které mohou být přiřazeny k virtuálnímu stroji.

libvirt v této části nabízí ještě několik dalších elementů umožňujících určení některých dalších parametrů pro práci s pamětí a swapem.

```
. . .
<memory>524288</memory>
<currentMemory>524288</currentMemory>
<vcpu>1</vcpu>
. . .
```

Model CPU

Volitelné požadavky na procesor mohou být definovány prostřednictvím následujících elementů: <cpu> (<minimum>, <exact>, <strict>), <model>, <vendor>, <topology> a <feature> (<force>, <require>, <optional>, <disable>, <forbid>). Konkrétní význam a použití je možno nalézt v dokumentaci.[10]

Kontrola životního cyklu virtuálního stroje

Pro změnu výchozího chování při vypnutí, restartování či havarování virtuálního stroje můžeme požadovanou akci definovat některým z následujících elementů: <on_poweroff>, <on_reboot> a <on_crash>. Každý z uvedených elementů může obsahovat jednu z následujících akcí:

- **destroy** – doména je kompletně ukončena a všechny zdroje jsou uvolněny,
- **restart** – doména je ukončena a po té nastartována se stejnou konfigurací,
- **preserve** – doména je kompletně ukončena, její zdroje jsou uchovány pro analýzu,
- **rename-restart** – doména je ukončena a po té nastartována s novým jménem.

<on_crash> podporuje ještě tyto další akce:

- **coredump-destroy** – jádro havarované domény je uloženo a po té je doména kompletně ukončena a všechny zdroje uvolněny,
- **coredump-restart** – jádro havarované domény je uloženo a po té je restartována se stejnou konfigurací.

Vlastnosti hypervisora

Některé další vlastnosti mohou být hypervisorem zpřístupněny hostovanému systému. Seznam povolených vlastností se uvádí jako samostatné elementy uvnitř elementu <features>. Možnosti jsou následující:

<pae> rozšíření fyzického adresování paměti umožňuje 32-bit hostům adresovat více než 4 GB paměti.

<acpi> je užitečné pro správu napájení.

<apic> pokročilý programovatelný kontrolér přerušení.

```
. . .
<features>
  <acpi/>
  <apic/>
```

```
    <pae/>
  </features>
  . . .
```

Práce s časem

Hodiny virtuálního stroje jsou obvykle řízeny hostitelskými hodinami. Většina operačních systémů předpokládá, že jsou hardwarové hodiny seřízeny na tzv. Coordinated Universal Time – koordinovaný světový čas (UTC). V některých případech je však toto chování potřeba změnit (například systémy Windows předpokládají že hardwarové hodiny udávají lokální čas). Toto nastavení je možné provést v elementu <clock> atributem offset. Možné hodnoty kromě utc a localtime jsou timezone a variable. Poslední dva zmíněné parametry vyžadují další nastavení patřičnými atributy.[10]

```
  . . .
  <clock offset='utc'/>
  . . .
```

Zařízení

Poslední část XML předpisu domény tvoří specifikace jednotlivých zařízení jako jsou datová úložiště, síťové karty, další USB či PCI zařízení, grafické adaptéry a podobně. Všechny specifikace zařízení jsou zabaleny v elementu <devices>.

emulator Element <emulator> obsahuje úplnou cestu k emulátoru zařízení.

```
  . . .
  <devices>
    <emulator>/usr/bin/kvm</emulator>
  . . .
```

disk Veškerá datová úložiště jako jsou pevné disky, CD či DVD se specifikují pomocí element <disk>. Důležitým atributem elementu <disk> je type, který může nabývat hodnot file nebo block. Tato hodnota určuje, zda je zdrojem soubor nebo blokové zařízení.

Volitelným atributem je device, který určuje, jakým způsobem bude disk představen hostovanému operačnímu systému. Možné hodnoty jsou floppy, disk a cdrom.

Všechny použitelné elementy uvnitř <disk> jsou uvedeny v tabulce 1.4.

```
. . .
<disk type='file' device='disk'>
  <driver name='qemu'/>
  <source file='/media/kvm/disk.raw'/>
  <target dev='vda' bus='virtio'/>
</disk>
<disk type='file' device='cdrom'>
  <driver name='qemu'/>
  <source file='/media/data/Debian/debian-506-i386-↩
      netinst.iso'/>
  <target dev='hdc' bus='ide'/>
  <readonly/>
</disk>
. . .
```

input Jako vstupní zařízení je možné atributem `type` definovat myš (`mouse`), nebo tablet (`tablet`). Pokud je povolen framebuffer, je vstupní zařízení automaticky přiřazeno – pomocí <input> je možno specifikovat další. Volitelným atributem `bus` lze upravit způsob „připojení“ zařízení. Možné hodnoty jsou `xen`, `ps2` a `usb`.

interface Možnosti nastavení sítě s pomocí *libvirt*u jsou veliké. Samotné konfiguraci sítě (konfiguračnímu XML pro síť) se bude věnovat samostatná část tohoto textu. Zde pouze představím tři základní možnosti, které se při zapojování virtuálních strojů do sítě nabízejí. Další možnosti jako je například i propojení více virtuálních sítí jsou popsány v dokumentaci[10].

- **Virtual network** (virtuální síť) – Poskytuje virtuální síť postavenou kolem síťového mostu hostujícího systému. Takováto síť může být kompletně izolována, nebo může být spojena s okolím prostřednictvím Network Address Translation – překlad síťových adres (NAT). Je zajištěna služba DHCP a DNS serveru. Nastavení parametrů sítě (jako jsou rozsahy přidělovaných adres a podobně) je možné upravit prostřednictvím XML souboru pro definici virtuální sítě. Tento způsob síťování bude využit v navrhovaném řešení.

```
. . .
<interface type='network'>
  <source network='ninthlama'/>
```

```
    <mac address='52:54:00:3f:27:e9'/>
    <model type='virtio'/>
  </interface>
  . . .
```

<source>	Obsahuje úplnou cestu k: • Pro typ `file` – atribut `file` obsahuje úplnou cestu k souboru obsahujícího disk. • Pro typ `block` – atribut `dev` obsahuje úplnou cestu k blokovému zařízení hostujícího systému sloužícího jako disk virtuálního stroje.
<target>	kontroluje sběrnici / zařízení, pod kterým je disk zpřístupněn hostovanému OS. • Atribut `dev` určuje logický název disku. • Volitelný atribut `bus` specifikuje typ sběrnice, na kterou je disk připojen. Možné hodnoty jsou `ide`, `scsi`, `virtio`, `xen` or `usb`. Pokud není atribut `bus` uveden, je zvolen odpovídající podle parametru `deb` (například pro `hda` bude `bus` zvolen `ide`).
<driver>	Umožňuje zvolení (pokud to hypervisor podporuje) konkrétního ovladače. • Atribut `name` udává jméno ovladače (např.: `qemu`). • Volitelný atribut `type` specifikuje podtyp. • Volitelný atribut `cache` upravuje nastavení cache (možné volby: `default`, `none`, `writethrough`, `writeback`).
<encryption>	Pokud je přítomný, specifikuje, jak je svazek šifrován.[11]
<shareable>	Přítomnost tohoto elementu indikuje, že zařízení je sdíleno mezi více doménami (musí být podporováno hypervisorem a OS).
<serial>	Volitelně specifikuje sériové číslo virtuálního pevného disku.
<readonly>	Určuje, že připojený disk je pouze pro čtení (například obraz disku CD).

Tab. 1.4: Elementy pro definici disku.

- **Bridge to LAN** (most do LAN) – Tato možnost vytvoří síťový most mezi síťovým rozhraním virtuálního stroje a fyzickým rozhraním hostitele. Všechny síťové parametry (služby DHCP a DNS serveru, atd.) jsou závislé na okolní síti. Virtuální stroj se chová, jako by byl fyzicky připojen do lokální sítě hostitele.

```
. . .
<interface type='bridge'>
  <source bridge='br0'/>
  <target dev='vnet1'/>
  <mac address="52:54:00:3f:27:e9"/>
</interface>
. . .
```

- **Generic ethernet connection** (obecné připojení) – Představuje nejflexibilnější způsob připojení domény do sítě, neboť prostřednictvím *libvirt*u je pouze vytvořeno virtuální síťové rozhraní a následně spuštěn uživatelsky definovaný skript, který má za úkol provést připojení virtuálního systému do sítě.

```
. . .
<interface type='ethernet'>
  <target dev='vnet2'/>
  <script path='/etc/qemu-ifup-mynet'/>
</interface>
. . .
```

video Element <video> je rodičovský prvek pro nastavení vlastností grafického zařízení. Toto nastavení se provádí pomocí atributů elementu <model>. Povinný atribut type přebírá jeden z následujících typů grafického adaptéru (možnosti jsou závislé na možnostech konkrétního hypervisora): vga, cirrus, vmvga, qxl, xen nebo vbox.

```
. . .
<video>
  <model type='vga' vram='8192' heads='1' />
</video>
. . .
```

graphics Grafický přístup k virtuálnímu stroji je možný několika různými způsoby. Každý z nich se nastavuje prostřednictvím elementu <graphics> a určuje povinným atributem `type`. Tři z možných typů grafického přístupu jsou následující:

- **sdl** – Typ `sdl` zobrazí v hostujícím systému okno virtuálního stroje. Může přebírat některé z následujících třech atributů: `display` – číslo displeje, na kterém se má okno zobrazit, `xauth` – ověřovací identifikátor a `fullscreen` – přebírající hodnoty `yes` nebo `no`.

```
. . .
<graphics type='sdl' display=':0.0'/>
. . .
```

- **vnc** – Nastartuje Virtual Network Computing (VNC) server naslouchající na IP adrese zadané pomocí `listen` a portu zadaném pomocí `port`.

```
. . .
<graphics type='vnc' port='5904'/>
. . .
```

  Význam dalších atributů je uveden v tabulce 1.5.

- **rdp** – Nastartuje Remote Desktop Protocol (RDP) server. Atribut `port` má stejný význam jako v případě `vnc`.

```
. . .
<graphics type='rdp' autoport='yes' />
. . .
```

`listen`	IP adresa, na které má server naslouchat.
`port`	Port, na kterém má VNC server naslouchat. Je-li `port` nastaven na -1 nebo atribut `autoport` na `yes`, bude port zvolen automaticky.
`passwd`	Heslo pro připojení k serveru (vloženo jako čistý text).
`keymap`	Specifikace klávesové mapy.
`passwdValidTo`	Určení doby platnosti hesla (nemusí být dostupné na všech hypervisorech).

Tab. 1.5: Atributy pro nastavení přístupu přes VNC.

1.4.2 Disky

Pro práci s virtuálními disky jsou v *libvirt*u určeny dva XML předpisy. První se stará o úložiště pevných disků (*storage pool*), druhý pak popisuje konkrétní disk (*storage volume*).

Storage pool XML

Storage pool XML definuje jakýsi kontejner, úložiště několika pevných disků. Fyzicky může být představováno konkrétním adresářem, diskovým oddílem nebo některou z dalších možností. Kořenový element je <`pool`> s povinným atributem `type` udávajícím typ úložiště (`dir`, `fs`, `netfs`, `disk`, `iscsi`, `logical`).

Elementy <`name`> a <`uuid`> jsou totožné s XML pro doménu. Následující elementy <`allocation`>, <`capacity`> a <`available`> jsou pouze informativní a při vytváření se nepoužívají. Důležitou částí je <`target`>, kde je za pomoci <`path`> uvedena cesta k úložišti. Je zde také možno nastavit výchozí oprávnění k vytvářeným souborům (diskům) pomocí <`permissions`>.

```
<pool type="dir">
  <name>ninthlama</name>
  <uuid>76179634-4d47-4dd9-9f08-30aa9b720aea</uuid>
  <target>
    <path>/home/ninthlama/data/storage_pool</path>
  </target>
</pool>
```

Storage volume XML

Jednotlivé disky jsou konkrétně definovány pomocí *Storage volume XML*. Element <`name`> určuje název virtuálního disku (je zároveň názvem souboru). Kapacita disku je určena elementem <`capacity`> v bytech, případně v jednotkách specifikovaných atributem `unit` (`K` – kilobyty, `M` – megabyty, `G` – gigabity,...).

Element <`target`> poskytuje cestu k souboru s diskem (<`path`> – pouze pro čtení), možnost specifikace formátu disku (atribut `type` elementu <`format`>) a nastavení práv k souboru.

```
<volume>
  <name>test.raw</name>
  <capacity unit="M">1500</capacity>
```

```
  <target>
    <format type='raw'/>
  </target>
</volume>
```

1.4.3 Síť

S pomocí *libvirt*u je možné nastavovat velké množství parametrů virtuálních sítí. Do těchto sítí lze posléze připojovat jednotlivé domény. Kořenovým elementem definice sítě je <network>, který neobsahuje žádné atributy.

Základní metadata

Obdobně jako v případě XML jsou pro doménu první dva elementy <name> a <uuid>. První definuje jméno sítě a může obsahovat pouze alfa-numerické znaky. Element <uuid> pak obsahuje globální identifikátor podle RFC 4122[22].

```
<network>
  <name>ninthlama</name>
  <uuid>f4cbd04e-2040-8c3d-f87f-af2716d3b10a</uuid>
  . . .
```

Připojení

V této části jsou k disposici tři elementy upravující parametry připojení virtuální sítě k okolí.

forward Element <forward> určuje způsob připojení virtuální sítě k vnější síti. Atribut `mode` určuje, zda bude prováděn překlad adres (`nat`), či zda bude provoz do virtuální sítě směrován bez překladu adres (`route`) – v tomto případě se předpokládá, že server v lokální síti má odpovídající záznamy ve směrovací tabulce. Volitelný atribut `dev` specifikuje rozhraní, kterému je předáván provoz. Pokud není uveden, je překlad adres prováděn na všechna přítomná rozhraní.

bridge Tag <bridge> určuje název (atributem `name`) síťového mostu, nad kterým je postavena virtuální síť. Je doporučeno používat jména s prefixem `vir`. Název `virtbr0` je však rezervován pro „defaultní“ virtuální síť. Atribut `stp` určuje hodnotami `on` nebo `off` zapnutí nebo vypnutí *Spanning Tree Protocol*u (protokol linkové

vrstvy zamezující vznikání smyček v přemostěných sítích). Atribut `delay` nastavuje hodnotu zpoždění v sekundách.

domain Atributem `name` elementu <`domain`> můžeme definovat název domény DHCP serveru.

```
. . .
<forward mode='nat'/>
<bridge name='virbr1' stp='on' delay='0' />
. . .
```

Adresování

Poslední část uzavřená v elementu <`ip`> určuje způsob adresování ve virtuální síti. Samotný element <`ip`> má dva atributy `address` a `netmask` určující IPv4 adresu a masku síťového mostu.

tftp Volitelný <`tftp`> slouží k nastavení *tftp* serveru. Jediným povinným atributem je `root` určující kořenový adresář dostupný přes TFTP.

dhcp Nastavení DHCP serveru umožňuje tag <`dhcp`>, jenž je rodičem pro následující elementy:

- **range** – Specifikace rozsahu automaticky přidělovaných adres je možná díky elementu <`range`> a jeho atributů `start` a `end`.
- **host** – Pro statické přiřazení IP adresy konkrétní doméně podle MAC adresy je použit element <`host`> s atributy `mac` (MAC adresa virtuálního stroje), `name` (jméno pro DNS server) a `ip` specifikující IP adresu.
- **bootp** – Volitelný element <`bootp`> umožňuje nastavení cesty k obrazu pro start ze sítě (atribut `file`) a případně adresu tftp serveru atributem `server` (defaultně se předpokládá tftp server na stejné adrese jako DHCP server).

```
. . .
<ip address='192.168.200.1' netmask='255.255.255.0'>
  <dhcp>
    <range start='192.168.200.100' end='192.168.200.200' />
    <host mac='52:54:00:3f:27:ee' name='server' ip='↩
        192.168.200.2' />
    <host mac='52:54:00:3f:27:e9' name='testkvm' ip='↩
        192.168.200.10' />
```

```
    </dhcp>
  </ip>
</network>
```

V předchozí části nejsou popsány všechny elementy a atributy definující XML pro doménu, disk nebo virtuální síť. Snažil jsem se uvést pouze ty které jsou významné pro tuto práci, bližší specifikaci lze nalézt v dokumentaci [10], odkud byly poznatky čerpány.

2 AUTOMATICKÁ INSTALACE OS

Plánované nasazení virtuálních strojů v rámci výuky předpokládá vytvoření nárazově (na začátku semestru) velkého množství virtuálních strojů včetně nainstalovaného operačního systému. Pro automatickou instalaci operačního systému Linux jsou dostupné různé nástroje. Většinou bývají určené pro konkrétní typ distribuce/instalátoru. Nejznámější je *kickstart* pro distribuce vycházející z RedHatu. Podobnou funkci nabízí též instalátor distribuce Debian. Linuxová distribuce SUSE nabízí automatický instalační nástroj *AutoYaST*[1].[9]

Zajímavým projektem je též *FAI* (*Fully Automatic Installation*)[2], který se dá použít pro různé distribuce, je však primárně určen pro Debian. Možnosti *FAI* jsou veliké a přináší značné zjednodušení při správě většího množství počítačů. Dobře si poradí s různorodým prostředím a různým určením jednotlivých počítačů. Pro naše účely jde však o zbytečně komplikovaný nástroj.

2.1 Kopírování virtuálních strojů

Samotná instalace jednoho stroje však také zabere určitý čas a jelikož poběží všechny virtuální stroje na jednom serveru, není vhodné instalovat všechny na jednou. V našem případě můžeme využít faktu, že budou všechny instalované systémy v podstatě totožné. Virtuální disk instalovaného stroje může být představován obyčejným souborem, který je možno zkopírovat a vytvořit tak libovolný počet kopií. Jednotlivé kopie lze pak připojit jako samostatné disky k jednotlivým virtuálním strojům.

Jediný prozatím zjištěný problém, který se vyskytuje na některých distribucích v případě naklonování pevného disku, je změna názvu síťové karty. Název síťové karty bývá v některých případech přiřazován podle MAC adresy (aby v případě existence dvou a více síťových karet v systému nedocházelo při každém startu k náhodnému prohazování jmen). V případě naklonování virtuálního stroje se nám však MAC adresa musí změnit a tím dojde k přejmenování síťové karty např. z `eth0` na `eth1` (předpokládám jednu síťovou kartu v systému). Řešení spočívá ve smazání záznamu přiřazujícímu název `eth0` MAC adrese původního stroje.

O správu zařízení se ve většině linuxových distribucí stará *udev*, který hledá záznamy o pravidlech v souborech v adresáři `/etc/udev/rules.d/`. V případě distribuce Debian se jedná o soubor `70-persistent-net.rules`, v ostatních distri-

[1] <http://www.suse.com/~ug/autoyast_doc/index.html>
[2] <http://fai-project.org/>

bucích se může soubor jmenovat jinak. Obsahem souboru jsou záznamy podobné tomuto:

```
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*", ↩
    ATTR{address}=="52:54:00:3f:27:ee", ↩
    ATTR{dev_id}=="0x0", ATTR{type}=="1", KERNEL=="eth*", ↩
    NAME="eth0"
```

Pokud tyto řádky smažeme, dojde při startu systému k novému pojmenování síťových zařízení, jelikož již není blokován název `eth0` předcházejícím záznamem, bude přiřazen první síťové kartě v systému.

Tuto změnu je možné udělat dodatečně na každém zkopírovaném stroji nebo ji provést jednou těsně před zkopírováním na výchozím virtuálním stroji.

2.2 KickStart

Systém *KickStart* je prostředek pro zautomatizování instalace linuxových distribucí vycházejících z RedHatu (RedHat Enterprise Linux, Fedora, CentOS). Snaha o určitou kompatibilitu s konfiguračním souborem KickStartu je i v distribuci Ubuntu. Konfigurační soubor pro KickStart v podstatě obsahuje odpovědi na dotazy položené v průběhu instalace instalačním programem *Anaconda*.

Cesta k tomuto souboru je následně předána startujícímu systému jako parametr jádra např.: `ks=http://192.168.222.121/ks.cfg` pro soubor umístěný na webovém serveru. Nechceme-li nebo nemůžeme-li zapisovat tento parametr ručně při startu instalace, je možné jej předat prostřednictvím DHCP. V případě instalace na virtuálním stroji můžeme též s výhodou využít tzv. direct kernel boot, tedy přímý start jádra popsaný v části 1.4.1. Díky tomu máme možnost automaticky předat jádru startujícího systému potřebné parametry a celá instalace může proběhnout bez zásahu uživatele (a není nutné upravovat nastavení DHCP serveru).

Instalační obrazy mohou být uloženy v datovém úložišti hostujícího počítače nebo připojeny pomocí protokolu Network File System – protokol pro připojování síťových disků (NFS) ze vzdáleného serveru. Pro instalaci balíčků je vhodné vytvořit lokální zrcadlo na místní síti. Nebude tak docházet ke zbytečnému zatěžování oficiálních serverů a připojení k internetu.[17]

2.2.1 Konfigurační soubor pro KickStart

Konfigurační soubor pro KickStart je obyčejný textový soubor s posloupností příkazů ovlivňujících probíhající instalaci. Je možné takovýto soubor vytvořit ručně, jednodušší je však využít možnosti speciálního programu `system-config-kickstart` nebo jej získat z nainstalovaného systému. V čerstvě nainstalovaném systému je uložen na adrese `/root/anaconda-ks.cfg`. Získaný soubor je následně možné upravit podle vlastních potřeb.

První část souboru obsahuje odpovědi na otázky instalátoru: jazyk, rozložení klávesnice, nastavení sítě, heslo pro uživatele root, časová zóna, rozložení disků a další. V druhé části souboru jsou vypsány instalované sekce balíčků či konkrétní balíčky.[4]

```
# Instalace systému (muže být upgrade)
install
# Instalace bude probíhat v textovém režimu
text
# Adresa pro získání instalačních souborů
# (může být cdrom pro instalaci z_CD)
url --url http://mirror.centos.org/centos/5.6/os/i386
# Jazyk
lang cs_CZ.UTF-8
# Rozložení klávesnice
keyboard cz-us-qwertz
# Nastavení síťove karty eth0 (nastavení z DHCP serveru)
network --device eth0 --bootproto dhcp
# Root heslo (poslední znak dolaru na řádek nepatří)
rootpw --iscrypted $1$MDAHekLS$uDBwerX5KxoWERGR.mgsK1$
# Firewall
firewall --enabled --port=22:tcp
# Způsob ukládání hesel
authconfig --enableshadow --enablemd5
# Časové pásmo
timezone --utc Europe/Prague
# Umístění zavaděče systému
bootloader --location=mbr --driveorder=vda
# Smazání všech diskových oddílů
clearpart --all --initlabel
# Vytvoření oddílu pro swap
```

```
part swap --size 128
# Vytvoření systémového oddílu
part / --size 500 --grow

# Instalované balíčky (nebo skupiny balíčků)
%packages
@base
@core
@editors
@text-internet
keyutils
openssh-server
device-mapper-multipath
```

2.3 Přednastavená instalace distribuce Debian

Pro distribuci Debian existuje obdobný způsob automatického projití instalačního procesu jako je popsáno v předchozím oddílu o KickStartu, ze subjektivního hlediska je ale systém KickStart v různých ohledech o něco lepší. Vše je opět založeno na textovém konfiguračním souboru obsahujícím odpovědi na jednotlivé otázky instalačního programu.[2]

Cesta k tomuto souboru se taktéž předává při startu instalace jako parametr jádra a je tedy možno využít stejných postupů jako v případě instalace pomocí KickStartu (přímého bootu jádra, předání skrze DHCP server). Konkrétní parametr je závislý na typu média, na kterém je konfigurační soubor uložen. Pro načtení z webového serveru je potřeba použít parametr ve tvaru `preseed/url`, tedy například `preseed/url=http://192.168.123.1/preseed.cfg`. Volitelně je možno předat též kontrolní součet (metodou MD5) konfiguračního souboru prostřednictvím `preseed/-url/checksum`. Pokud tento součet nesouhlasí se staženým souborem, instalátor jej odmítne použít.

Ke správnému fungování podle našich požadavků je též nutné předat ještě několik dalších parametrů zajišťujících automatické nastavení sítě, načtení konfigurace a spuštění automatické instalace:

- `auto=true` – nastavení automatické instalace
- `priority=critical` – zamezení zobrazování dotazů instalačního programu s prioritou menší než *critical*

- `interface=auto netcfg/dhcp_timeout=60` – načtení konfigurace síťové karty z DHCP

Celý seznam parametrů předávaných jádru tak může vypadat následovně:

```
auto=true priority=critical interface=auto ↩
    netcfg/dhcp_timeout=60 ↩
    preseed/url=http://192.168.123.1/preseed.cfg}
```

2.3.1 Konfigurační soubor automatické instalace Debianu

Pro získání konfiguračního souboru s nastavením automatické instalace Debianu pravděpodobně neexistuje žádný alternativní program jako je `system-config-kickstart` pro KickStart. Můžeme ale uložit nastavení z existujícího systému do souboru `file` s pomocí příkazu `debconf-get-selections` z balíčku `debconf-utils`:

```
# debconf-get-selections --installer > file}
# debconf-get-selections >> file}
```

Získaný soubor je poté potřeba upravit a přizpůsobit aktuálním potřebám. Druhou (o něco lepší) variantou je využití ukázkového souboru[3] z dokumentace[2]. Ukázkový soubor je vzhledem k rozsahu přiložen v příloze A.2.

[3] <http://www.debian.org/releases/stable/example-preseed.txt>

3 PROGRAM PRO SPRÁVU VIRTUÁLNÍCH STROJŮ

NinthLama je název navrhované aplikace sloužící ke správě virtuálních strojů pro potřeby výuky předmětu MNSB (Návrh, správa a bezpečnost počítačových sítí), případně pro použití v dalších laboratořích. Úkolem aplikace je umožnit automatizovanou přípravu virtuálních strojů včetně nainstalovaného systému pro jednotlivé studenty a jejich správu (spouštění, zastavování, sběr informací).

Přístup ke konkrétnímu hypervisoru bude řešen pomocí dříve popsaného *libvirt*u. Tato volba umožní jednotný přístup nezávislý na použité virtualizační technologii.

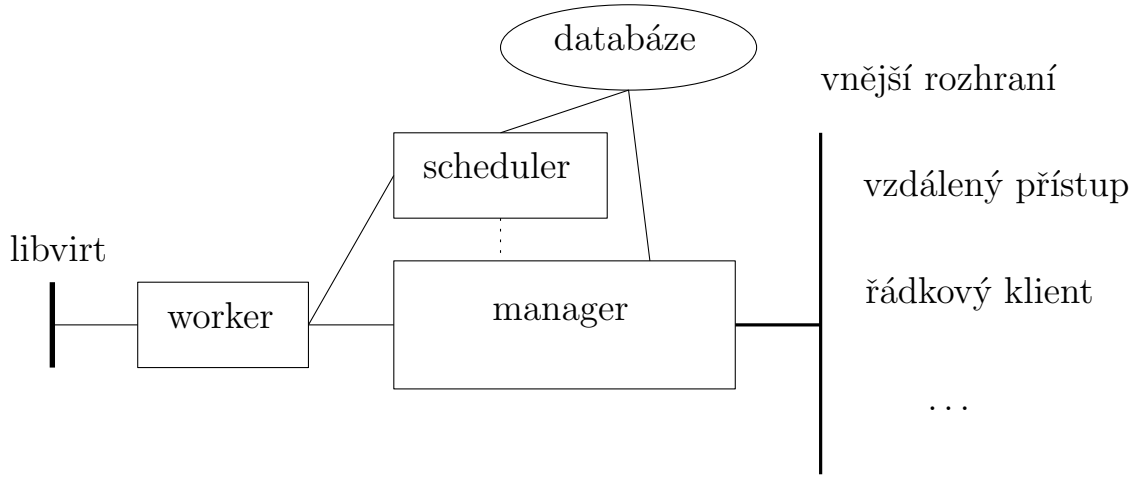
Jako programovací jazyk byl zvolen Python[1]. Knihovna *libvirt* nabízí rozhraní pro několik programovacích jazyků jako je C#, Java, Perl, PHP a další, Python však vyniká svojí jednoduchostí a přímočarostí při psaní kódu. Jedná se o dynamický objektově orientovaný interpretovaný jazyk. Vývoj v jazyce Python probíhá rychleji než v jiných programovacích jazycích, daň za tuto výhodu je obecně nižší rychlost výsledné aplikace v porovnání např. s jazykem C/C++. Zpomalení však není v běžných aplikacích nijak kritické (existují dokonce případy, kdy může být program v Pythonu rychlejší, neboť při práci se složitějšími datovými typy má Python vnitřně kód velice dobře odladěn). Pokud by však byl v programu kritický bod náročný na výpočet, lze tuto část implementovat v nižším programovacím jazyce (C/C++) a zbytek ponechat v Pythonu.

3.1 Koncept programu

Koncept programu NinthLama–backend je zobrazen na obrázku 3.1. Ústřední částí programu je *manager* (manažer), který má na starosti zprostředkovávat vyřizování požadavků mezi vnějším rozhraním a *worker*em, případně předávání požadavků do fronty na pozdější vykonání. O tyto úkoly se poté stará *scheduler* (plánovač).

worker má na starosti vykonávání jednotlivých požadavků (ať již prostřednictvím *libvirt*u nebo jiným způsobem). Způsob spojení (komunikace) mezi *manager*em či *scheduler*em a *worker*em využívá Remote procedure call – vzdálené volání procedur (RPC), které je také použito pro komunikaci na vnějším rozhraní.

[1] <http://www.python.org/>



Obr. 3.1: Koncept schématu programu NinthLama–backend

3.1.1 Databáze

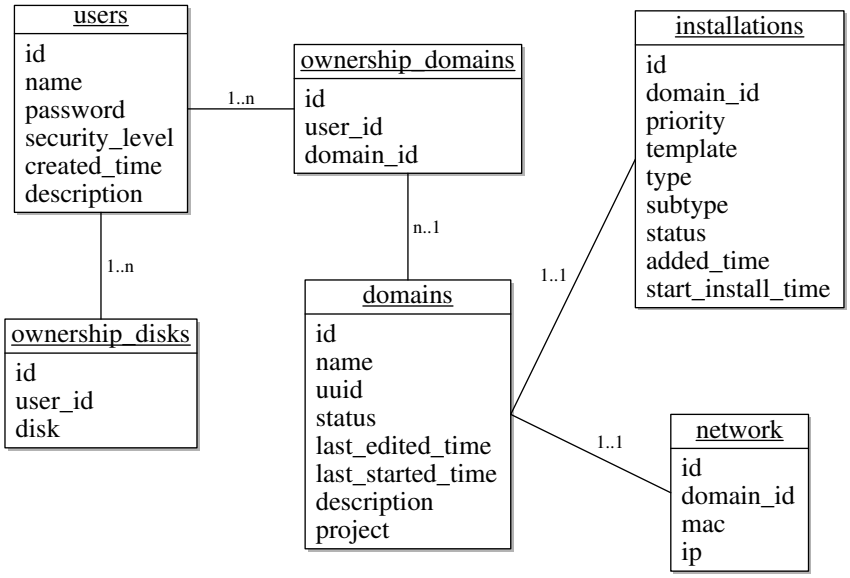
Uchovávání nastavení jednotlivých virtuálních strojů je možné nechat plně v režii *libvirt*u nebo vše ukládat do vlastní databáze. Ponechání domén v persistentním stavu (tedy tak, že se *libvirt* postará o jejich uchování) nabízí možnost přistupovat k nim a ovládat je i mimo prostředky programu NinthLama. Naopak jejich uchovávání ve vlastní databázi přináší větší flexibilitu v možnostech změny nastavení a podobně. Přístup k virtuálnímu stroji však bude plně závislý na programu NinthLama. Jako nejvhodnější se jeví určitý kompromis, kdy budou stroje uchovávány v persistentním režimu samotným *libvirt*em, určité informace však budou (zároveň nebo navíc) uchovávány ve vlastní databázi.

Obrázek 3.2 představuje návrh tabulek použitých v programu NinthLama–backend. Nejdůležitějšími tabulkami jsou `users`, `domain` a `installations`. Zbylé tabulky slouží buď k vytvoření spojení M:N (`ownership_domains`) nebo k uchovávání dodatečných informací (`ownership_disks`, `network`).

Tabulka `domains`

Tabulka `domains` zajišťuje uchovávání dodatečných informací k vytvořeným doménám. K identifikaci domény a její provázání s *libvirt*em je použito její jméno (sloupec `name`) případně identifikátor `uuid` zmiňovaný v části 1.4.1. Pole `status` určuje stav domény:

- `READY` – doména je připravena
- `INSTALLATION` – probíhá (nebo je připravena) automatická instalace



Obr. 3.2: Návrh tabulky uchovávající informace o virtuálních strojích.

- `MANUAL_INSTALL` – probíhá (nebo je připravena) manuální instalace
- `LOCKED` – doména je uzamčena
- `UNKNOWN` – neznámý stav

Posledním sloupcem je `project`, ten obsahuje ID projektu, ke kterému může být doména přiřazena. Vlastnictví domén řeší tabulka `ownership_domains`, která přiřazuje domény k uživatelům popisovaným v následující části. Vyhrazení samostatné tabulky pro toto propojení umožňuje vytvořit propojení označované M:N, tedy jednu doménu může vlastnit více uživatelů a naopak každý uživatel může být vlastníkem více domén.

Tabulka `users`

Jak bylo naznačeno v předchozím odstavci, tabulka `users` definuje jednotlivé uživatele systému. Jsou zde jak uživatelé, kteří mají právo se systémem pracovat (vytvářet, instalovat, spouštět virtuální stroje…), tak také uživatelé, kterým je pouze přiřazeno vlastnictví určitých domén, sami ale nemají právo k interakci se systémem. Případně kombinace obou.

Základními informacemi o uživateli jsou jméno (`name`) a heslo (`password`). Heslo je ukládáno v zahašované podobě společně se jménem, jak naznačuje vzorec 3.1:

$$X = \mathrm{H}(P + N), \tag{3.1}$$

kde X je výstup hašovací funkce, který je uložen v databázi, H je hašovací funkce (použitým algoritmem je SHA256), P je heslo a N jméno uživatele.

Připojením jména za hašované heslo zamezíme odhalení stejných hesel (jedná se tedy o jednoduchý způsob „zasolení“, kdy dvě stejná hesla na vstupu produkují rozdílný výstup právě z důvodu přidání rozdílných jmen).

Má-li být uživatel pouze vlastníkem některých virtuálních strojů, bez nutnosti aby se sám přihlašoval do systému, může být položka `password` nastavena na '`!`' (tento stav je také indikován nejnižší úrovní oprávnění).

Posledním důležitým údajem pro každého uživatele je úroveň oprávnění (`security_level`). Jedná se o celočíselnou hodnotu určující úroveň pravomocí v systému. Jednotlivé hodnoty jsou následující:

0 – `SL_UNAUTHORIZED`
: Nejnižší hodnota oprávnění, uživatel nemá žádná práva.

1– `SL_OWNREADONLY`
: Uživatel může přistupovat k vlastním doménám (vypsat seznam, stav).

2 – `SL_OWNWRITE`
: Uživatel může ovládat vlastní domény (spouštět, zastavovat,...).

3 – `SL_ALLREADONLY`
: Uživatel má přístup ke všem doménám stejný jako `SL_OWNREADONLY` (+ může ovládat vlastní domény).

4 – `SL_ALLWRITE`
: Uživatel může ovládat všechny domény.

5 – `SL_ADMINISTRATOR`
: Nejvyšší oprávnění.

Tabulka `installations`

Tabulka `installation` udržuje informace o připravených a probíhajících automatických instalacích. Jejím prostřednictvím tady probíhá komunikace mezi *manager*em a *scheduler*em (*manager* vkládá do tabulky požadavky pro instalaci a *scheduler* kontroluje přítomnost těchto požadavků a postupně zajišťuje jejich vyřízení).

Původním návrhem bylo použít pro komunikaci mezi *manager*em a *scheduler*em možnosti modulu `multiprocessing` (například `multiprocessing.Queue`). Využití databáze je však jednodušší a přináší větší možnosti: lepší oddělení *manager*u a *scheduler*u, snadnější možnost implementace prioritní fronty, možnost správy položek ve frontě.

Výpadek jedné či druhé strany nemá za následek ztrátu všech položek, po opětovném spuštění jsou všechny požadavky opět dostupné.

Položka `status` určuje stav instalace (`READY_TO_INSTALL` – připravenost k instalaci, `INSTALLATION_RUNNING` – probíhající instalace). Priorita zpracování požadavku (čím nižší číslo, tím dříve) se uchovává ve sloupci `priority`, výchozí hodnota je `50`.

Nastavení jednotlivých způsobů instalace jsou uloženy v šablonách pro instalaci popisovaných dále v textu. Výběr konkrétního způsobu instalace tedy znamená výběr z několika šablon. Zvolená šablona je uchovávána ve sloupci `template`. Sloupce `type`, `subtype` určují typ instalace převzatý z šablony (při každé kontrole průběhu instalace není nutné načítat celou šablonu).

3.1.2 Remote procedure call

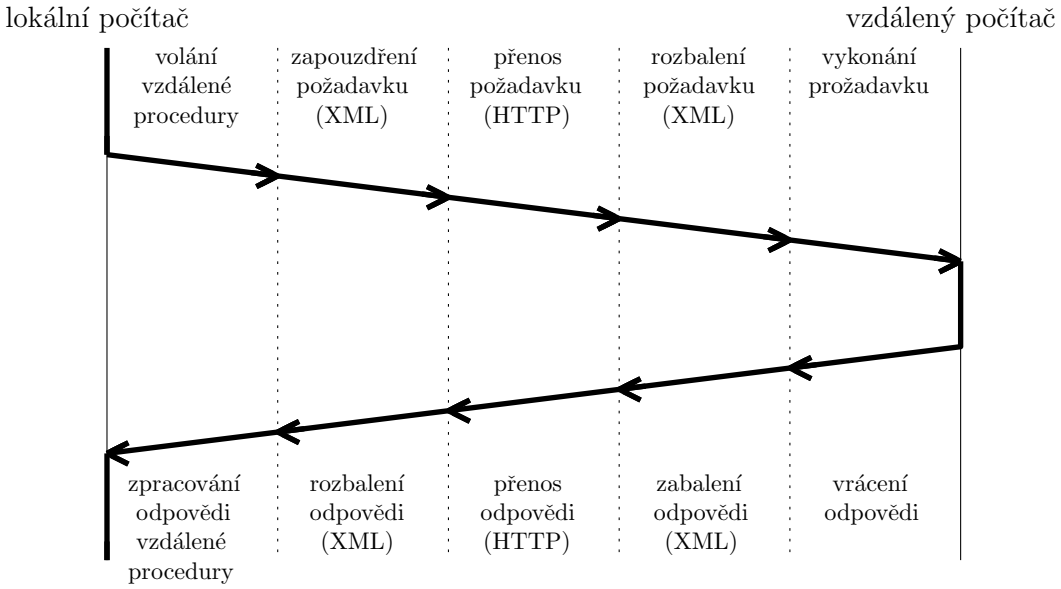
Velice důležitým prvkem programu NinthLama–backend je komunikace s okolím a navázání na webové rozhraní NinthLama–web (není součástí této práce). Toto spojení a provázání je zprostředkováno pomocí RPC (stejně jako komunikace mezi *manager*em a *worker*em). Program NinthLama–backend tvoří serverovou část RPC. Jednotlivé programy potřebující zadat požadavek či získat informace budou v pozici klienta.

Schéma volání vzdálené procedury je na obrázku 3.3 (poznámky v závorkách se vztahují k *XML–RPC*). Požadavek je nejprve zabalen do formátu vhodného k transportu (například do XML, existují ale i binární formy reprezentace). Následně je s využitím vhodného protokolu přenesen k serverové části RPC, kde je požadavek rozbalen a předán k vykonání. Odpověď se obdobně zabalí a přenese ke klientovi.

Implementací RPC pro Python existuje více. Díky své jednoduchosti použití se jako vhodný kandidát jeví *XML–RPC*[2]. V Pythonu je implementace v modulu `xmlrpclib` (v Pythonu od verze 3.0 bude modul přejmenován na `xmlrpc.client`).[15]

Jednotlivé požadavky na vzdálený server i vrácené odpovědi jsou zabaleny do XML kódu. Díky tomu je komunikace sice poněkud rozsáhlejší než v případě binární reprezentace. Umožňuje to však snadnější ladění a odhalování chyb, neboť je XML kód čitelný i pro člověka. Pro přenos požadavků je využit protokol HTTP. Ten je ve světě sítí a internetu velice rozšířen a nebývá problém například s jeho blokováním na firewallech.[3, 6]

[2] <http://www.xmlrpc.com/>



Obr. 3.3: Schéma volání vzdálené procedury.

Implementace XML–RPC klienta

Výše zmíněný modul `xmlrpclib` zprostředkovává klientskou část komunikace. [15] Spojení se serverem je představováno objektem třídy `ServerProxy`, jehož jediným povinným parametrem při vytváření je URI serveru. Prostřednictvím URI je nejen určena adresa serveru, ale také použitý přenosový protokol (`http` – nešifrovaný, `https` – šifrovaný), autentizační údaje (jméno a heslo), port a případně cesta. Celkový formát je tedy následující:

`PROTOKOL://[JMENO:HESLO@]ADRESA[:PORT][/CESTA]`,

pro šifrované spojení na lokální server naslouchající na portu 65500 a autentizaci jménem a heslem tedy může vypadat například následovně:

`https://jmeno:heslo@127.0.0.1:65500`.

Jelikož je část `/CESTA` také předána serverové části ke zpracování, je možné ji též využít jako autentizační metodu předáním náhodného řetězce (fráze). Toho je využito při komunikaci mezi *manager*em a *worker*em, neboť se jedná o komunikaci dvou částí jednoho programu a je tedy zbytečné předávat jméno a heslo, ale stačí na konec adresy vložit autentizační frázi. Celé URI potom získá následující tvar:

`https://127.0.0.1:65501/abeb42a478401e2`.

Implementace XML–RPC serveru

Základní implementace XML–RPC serveru je v modulu `SimpleXMLRPCServer`, ta však neobsahuje některé důležité možnosti jako je autentizace a šifrování komunikace. Z tohoto důvodu vznikl modul `nl_xmlrpc_server`, který přebírá vlastnosti modulu `SimpleXMLRPCServer`, navíc však přidává možnost šifrované komunikace a autentizaci klienta. Umožňuje také zpracování požadavků ve více procesech.[16, 1]

O vytvoření serveru se stará funkce `createXMLRPCServer`, která přebírá skrze parametry nastavení adresy a portu pro naslouchání a další volby pro nastavení šifrované komunikace, autentizace a paralelního zpracovávání požadavků ve více procesech.

K zajištění šifrovaného spojení skrze SSL/TLS je kromě předání logické hodnoty `True` do argumentu `ssl` též potřeba předložit certifikát a soukromý klíč serveru. Pro vygenerování vlastního certifikátu je možné použít příkaz v terminálu:

```
# openssl req -new -x509 -nodes -out server.cert.pem -keyout \
    server.key.pem -days 1098
```

Pro autentizaci existuje několik možností, konkrétní požadovaný typ je předáván funkci `createXMLRPCServer` parametrem `auth`:

- `None` – žádná autentizace se neprovádí
  příklad uri: `https://localhost:65500`
- `phrase` – autentizace frází
  příklad uri: `https://localhost:65501/BEZPECNOSTNI_FRAZE`
- `password` – autentizace jménem a heslem
  příklad uri: `https://JMENO:HESLO@localhost:65500`
- `both` – autentizace frází a/nebo jménem a heslem

O autentizaci/autorizaci se může starat libovolná funkce implementující definované rozhraní `authHandler(method, parameters, phrase, name, password)`. Odkaz na tuto funkci je předáván serveru prostřednictvím funkce `setAuth_handler`.

3.1.3 Konfigurace

Konfigurace celého programu je uložena v textovém konfiguračním souboru. O jeho zpracování se stará modul `nl_config.py`. Jedná se o standardní konfigurační soubor rozdělený do jednotlivých sekcí podle toho, čeho se uvedené volby týkají. Volby jsou

uvedeny ve tvaru `PROMENNA=HODNOTA` a jednotlivé sekce jsou rozděleny názvem sekce v hranatých závorkách (např.: `[logging]`).

Nyní uvedu obecné sekce konfiguračního souboru, volby týkajících se konkrétních částí programu NinthLama–backend budou uvedeny dále v příslušném oddílu.

`[ninthlama]`

Hlavní oddíl konfiguračního souboru obsahuje volby pro určení uživatele a skupiny, pod kterým má neprivilegovaná část programu běžet, cesty k šablonám a schématům a názvy výchozích datových úložišť a sítě.

```
# hlavní sekce konfigurace
[ninthlama]
# uživatel a skupina pod kterým poběží
# neprivilegovaná část NinthLama-backend
user=ninthlama
group=ninthlama
# cesta k šablonám virtuálních strojů, sítě a disků
# (v uvedeném adresáři musí být adresáře:
#    domains/  networks/  storage_pool/  storage_volume/)
templates=/home/ninthlama/backend/templates/
# cesta k RNG souborům (definujícím xml
# pro instalační šablony a podobně)
schemas=/home/ninthlama/backend/schemas/
# Storage pool-(defaultní) úložiště virtuálních pevných disků
storage_pool=ninthlama
# Storage pool- úložiště iso obrazů instalačních cd
storage_pool_cd=ninthlama_cd
# Network-(defaultní) virtuální síť do které mají být
# virtuální stroje připojeny
network=ninthlama
```

`[logging]`

Při konfiguraci logování jsou nedůležitější položky `log_console_level`, `log_file_level` a `log_file`. První dvě určují, jaké množství informací bude zaznamenáváno do logu, třetí pak umístění log souboru. Pokud program neběží v debugovacím režimu, bývá standardní výstup a standardní chybový výstup přesměrován do `/dev/null`

(tedy zahozen). Ostatní volby upřesňují nastavení modulu `logging` a objektu `loggeru` a starají se například o rotaci a zálohování souborů logů.

```
# konfigurace logování
[logging]
# jméno základního loggeru
log_root=NinthLama
# level logování do konzole
# (možné hodnoty: ALL, DEBUG, INFO, WARNING, ERROR,
# CRITICAL, NOTHING,  nebo celé číslo)
log_console_level=ALL
# název a cesta k log souboru
log_file=/tmp/ninthlama.log
# level logování do souboru
# (možné hodnoty: ALL, DEBUG, INFO, WARNING, ERROR,
# CRITICAL, NOTHING,  nebo celé číslo)
#log_file_level=WARNING
log_file_level=ALL
# maximální velikost log souboru (Bytes)
log_file_size=12800000
# maximální počet záloh log souboru
log_file_backups=1
```

`[database]`

Pro spojení s databází a práci s ní je použit modul `sqlalchemy`. Díky němu by mělo být možné použít různé typy databází (*sqlite*, *PostgreSQL*, *MySQL*,...) bez nutnosti změny kódu. Připojení k databázi je definováno parametrem `url`, jehož formát je definován v dokumentaci k modulu `sqlalchemy`[20].

```
# Konfigurace připojení k databázi
[database]
# adresa pro spojení s databází
# (bližší informace viz funkce create_engine z modulu ↩
    sqlalchemy)
url=sqlite:////home/ninthlama/ninthlama-backend.db
```

3.1.4 Šablony (templates)

Při vytváření virtuálních strojů, disků a dalších součástí virtualizovaného prostředí je možné (prostřednictvím XML souborů) definovat velké množství parametrů. U virtuálního stroje se jedná například o velikost operační paměti, způsob zavádění operačního systému, typ a umístění pevného disku, nastavení grafického výstupu a mnoho dalších součástí. Mnoho parametrů bude u většiny vytvářených strojů stejných a je tedy zbytečné je zadávat stále znovu. Z tohoto důvodu jsou pro vytváření virtuálních strojů jejich instalaci i při dalších úkonech využity šablony. Při požadované akci nemusí být zadáváno nepřeberné množství parametrů, ale pouze jméno vybrané šablony, která poslouží například jako základ pro vytvářenou doménu. Všechny použité šablony mají podobu XML souborů a k jejich popisu je použit formát RelaxNG[3].

Šablona virtuálních strojů

Šablony pro vytvoření virtuálních strojů jsou v podstatě stejné jako XML popisující konkrétní doménu (jsou také validovány podle stejného schématu).

Popis šablony je umístěn v tagu <description>. Neuvádí se zde element <uuid> ani například MAC adresa u síťové karty (tyto hodnoty jsou přiřazeny *libvirtem* konkrétní doméně až při jejím vytváření). Jméno domény a její popis, stejně jako seznam připojených disků, síťové karty a podobně jsou nastaveny při vytváření virtuálního stroje na konkrétní požadované hodnoty. Příklad šablony pro virtuální stroj s 256 MB RAM je uveden v příloze B.1.

Šablony diskových úložišť, disků a sítí

Šablony pro pevné disky a konkrétní XML popis disku jsou analogicky podobné šablonám pro domény a jejich konkrétnímu XML. Příklad pro disk s kapacitou 3 GB je uveden v příloze B.2

Co se týká šablon pro úložiště virtuálních disků (*storage pools*) a sítě, je zde určitý rozdíl. Zatímco domény a disky jsou vytvářeny na žádost uživatele, výchozí virtuální síť a úložiště disků musí existovat (nebo být vytvořeno) již při spuštění programu NinthLama–backend. Z toho důvodu musí být mezi šablonami pro úložiště virtuálních disků a sítě šablony s názvem uvedeným v konfiguračním souboru (sekce

[3] <http://relaxng.org>

`[ninthlama]`: položky `storage_pool`, `storage_pool_cd` a `network`). Při startu programu je provedena kontrola na existenci těchto zdrojů a v případě jejich nenalezení jsou vytvořeny podle odkazovaných šablon. Příklady jsou uvedeny v příloze B. Šablona definovaná v `ninthlama_pool_cd` definuje cestu k obrazům instalačních médií.

Šablony pro instalace

Pro instalace operačního systému nejsou v *libvirt*u žádné odpovídající předlohy pro šablony. Jejich formát je však relativně jednoduchý a popisný soubor ve formátu RelaxNG je připojen v příloze C.

Celá šablona je obalena do elementu <`installation`>, který obsahuje důležitý atribut `type`. Ten určuje obecný způsob instalace a může nabývat následujících hodnot:

- **automated_install** – automatická instalace bez zásahu uživatele.
  Atribut `subtype` nabývá hodnoty `direct_kernel_boot`, spouštění instalace je tedy umožněno přímým bootem jádra a instalační šablona obsahuje elementy nastavující cestu k jádru a initrd souboru a seznam parametrů předaných startujícímu instalátoru. Je zde také možno specifikovat cestu k kpřipojení instalačního média. V závislosti na možnostech konkrétního instalátoru a na předaných parametrech (ve kterých je především důležité specifikovat cestu ke stažení konfiguračního souboru automatické instalace), lze tímto způsobem provést instalaci z instalačního média i ze sítě.
- **manual_install** – manuální instalace, NinthLama–backend se stará pouze o připojení instalačního média (určeného elementem <`source`>) a o nastavení správné bootovací sekvence.
- **copy** – zkopírování existujícího disku s nainstalovaným systémem.

3.2 *Worker*

Na rozhraní mezi virtualizační technologií a programem NinthLama–backend je na jedné straně *libvirt* a na straně druhé *worker* (který je součástí programu NinthLama–backend). Jeho úkolem je předávat konkrétní požadavky (vytvoření, spuštění, zastavení domény, výpis virtuálních disků,...) směrem k *libvirtu* a naopak zpracovat a předat vrácenou odpověď zpět programu NinthLama–backend.

Důvodem proč je mezi *manager* (či *scheduler*) a *libvirt* vložen ještě jeden článek, je umožnění budoucího vývoje programu NinthLama–backend pro práci s virtuálními stroji na více fyzických počítačích. Hlavní část programu (*manager* a *scheduler*) může běžet na jednom hlavním serveru, zatímco každý počítač v roli hostitele pro virtuální stroje bude obsahovat pouze část *worker*. Jelikož je komunikace XML–RPC zapouzdřena do HTTP/HTTPS protokolu není problém s jejím přenosem přes síť.

Implementace *worker*u je v modulu `nl_worker`. Jeho průběh při startu je velice jednoduchý. Po načtení konfigurace (z parametrů na příkazové řádce a z konfiguračního souboru) je vytvořen objekt XML-RPC serveru a navázáno spojení s *libvirt*em. K tomuto serveru jsou zaregistrovány metody pro obsluhu vnějších požadavků z objektu třídy `WorkerMethods`. Následně je server spuštěn a očekává příchozí požadavky na určené adrese a portu, které zpracovává zaregistrovanými metodami. Při startu je (obdobně jako u *manager*u a *scheduler*u) uložen PID spuštěného procesu do souboru, při korektním ukončení je tento soubor smazán. Existence souboru, případně existence procesu se shodným PID jako je v souboru, nám umožňuje jednoduše ověřit, zda již není některá část spuštěna, případně jí zaslat signál na ukončení.

Konfigurace pro `[worker]`

Konfigurace *worker*u obsahuje hlavně adresu a port, kde má jako server přijímat požadavky a URI pro připojení k *libvirt*u. Dále pak cestu k uložení *.pid souboru (souboru s uloženým číslem procesu), autentizační frázi a nastavení serverové části XML–RPC (šifrování, cesty k souboru s certifikátem a tajným klíčem).

```
# konfigurace workeru
[worker]
# soubor pro uložení čísla procesu (pid)
pid_file=/tmp/nl_worker.pid
# adresa a port kde má worker naslouchat
address=127.0.0.1
port=65501
# použít přenos https (ssl)? (True, False)
ssl=False
# soubory s certifikátem pro ssl
keyfile=certs/server.key.pem
certfile=certs/server.cert.pem
# maximální počet současně běžících procesů
```

```
# pro zpracování požadavku
max_subprocess=1
# autentizační řetězec
# získaný například s pomocí:
# hashlib.sha1(str(random.random())).hexdigest()[:15]
phrase=b8980177936eabc
# URI pro připojení k libvirtu
libvirt_uri=qemu:///session
```

3.3 *Manager*

Hlavním řídícím prvkem celého programu je *manager*. Stará se o přijímání požadavků z vnějšího rozhraní, jejich zpracování a vrácení odpovědi. Požadavky mohou přicházet od různých klientů.

Spouštění *manager*u probíhá obdobně jako v případě *worker*u, pouze místo připojení k *libvirt*u dochází ke spojení s *worker*em a databází. Před spuštěním serveru je také inicializován plánovač (*scheduler*).

Autentizace a autorizace klientů je zajištěna jménem a heslem a tyto údaje jsou kontrolovány proti záznamům uloženým v databázi (podrobnosti v části 3.1.1). O ověření práv jednotlivých uživatelů a hlavně o autorizaci přístupu ke konkrétním funkcím se stará modul `nl_auth`. Ten obsahuje pro každou serverovou funkci vlastní autorizační metodu s názvem `auth_<NAZEV_FUNKCE>` (například pro funkci `startDomain` je v modulu `nl_auth` odpovídající metoda `auth_startDomain`).

Zpracování požadavků se skládá z rozhodnutí, zda se jedná o okamžitou žádost (spuštění virtuálního stroje, výpis existujících strojů) nebo o požadavek, jehož zpracování zabere více času (například instalace operačního systému). Tyto požadavky nejsou vykonávány přímo, ale jsou zařazeny do fronty (tabulka v databázi) a posléze vykonávány a zpracovávány *scheduler*em.

Konfigurace pro `[manager]`

Konfigurace je velice podobná konfiguraci serverové části *worker*u. Další volby (připojení k databázi, k *worker*u, nastavení logování) jsou přebírány z ostatních sekcí konfiguračního souboru.

```
# konfigurace manageru
[manager]
```

```
# soubor pro uložení čísla procesu (pid)
pid_file=/tmp/nl_manager.pid
# adresa a port kde má manager naslouchat
address=127.0.0.1
port=65500
# použít přenos https (ssl)? (True, False)
ssl=False
# soubory s certifikátem pro ssl
keyfile=certs/server.key.pem
certfile=certs/server.cert.pem
# maximální počet současně běžících procesů
# pro zpracování požadavku
max_subprocess=1
```

3.4 *Scheduler*

Zpracování déle trvajících úkolů má na starosti plánovač, který periodicky kontroluje existenci nových požadavků a stav právě probíhajících úkonů. Sem patří například automatická instalace operačního systému některou z metod zmíněných v kapitole 2.

Pokud do tabulky instalací přibude požadavek na automatickou instalaci operačního systému podle zvolené šablony a právě probíhající počet instalací je menší než maximální možný (volba `max_tasks` v konfiguračním souboru), je zahájena nová instalace. Ze zvolené šablony je patrné, zda se jedná o přímou kopii existujícího disku či o instalaci s pomocí KickStartu nebo jiného automatizačního nástroje. Podle potřeby je upraven virtuální stroj pro přímý boot jádra, jsou připojeny instalační obrazy a podobně. Následně je virtuální stroj spuštěn.

Celá instalace je v režii zvoleného automatizačního nástroje a úkol plánovače je nyní pouze čekat na dokončení instalace (signalizováno vypnutím domény). Poté dojde k překonfigurování domény do normálního stavu (nastavena startovací sekvence, odstraněn přímý boot jádra a instalační obrazy) a k odstranění záznamu z tabulky instalací.

Konfigurace pro `[scheduler]`

Konfigurační část pro *scheduler* obsahuje v podstatě pouze dvě volby: cestu k *.pid souboru a maximální počet probíhajících úloh. Ostatní potřebné nastavení je přebíráno stejně jako v případě *manager*u z ostatních částí konfiguračního souboru.

```
# konfigurace plánovače
[scheduler]
# soubor pro uložení čísla procesu (pid)
pid_file=/tmp/nl_scheduler.pid
# maximální počet současně probíhajících úloh (instalací)
max_tasks=2
```

3.5 Skript ninthlama–backend.py

Každou součást programu lze spouštět samostatně z příkazového řádku s parametry `start`, `stop`, `restart`. Při startu je možné přidat parametr `-d`/`--debug` pro spuštění v režimu ladění. Část *manager* a *scheduler* mohou být spuštěny až po úspěšném spuštění *worker*u (při spouštění dochází ke kontrole spojení a v případě chyby není daná část spuštěna).

Pro snadnější ovládání jednotlivých součástí je ve zdrojových kódech přítomen skript `ninthlama-backend.py`, který umožňuje ovládání celého programu (hromadné spouštění – `start`, zastavování – `stop`, restart – `restart`).

Mimoto je také s jeho pomocí možné vytvářet a upravovat uživatele programu NinthLama–backend. To je důležité hlavně při prvotní instalaci programu, neboť při prvním spuštění sice dochází k vytvoření potřebného databázového schématu, žádný uživatel ale vytvořen není, a proto není možné se k programu připojit vzdáleně.

Vytvoření uživatele jménem `manager` s heslem `heslo` a právy `ALLWRITE` (viz oddíl 3.1.1) je možné následujícím příkazem (podrobnosti k jednotlivým parametrům jsou dostupné v nápovědě k programu, parametr `--help`):

```
#./ninthlama-backend.py -u manager -p heslo -l 4 -d "Manager"
```

3.6 Řádkový klient ninthlama–client.py

Vnější rozhraní využívající XML–RPC je navrženo pro možnost připojení různých klientských aplikací. V rámci výuky se předpokládá využití webového rozhraní. Pro testovací účely a pro hromadnou správu nemusí být webová aplikace nejlepším řešením. Součástí zdrojových kódů je proto skript `ninthlama-client.py`, který představuje klientskou část programu NinthLama.

Při spuštění vyžaduje program parametr `-c` nebo `--connection` s URI pro připojení k *manager*u. URI je ve formátu definovaném v části 3.1.2 a jsou s jeho pomocí specifikovány kromě adresy a portu také přihlašovací údaje.

Ovládání programu je možné pomocí jednoduchých příkazů, jejichž seznam je dostupný po zadání `help`. V programu funguje doplňování příkazů a parametrů stiskem klávesy Tab.

4 NASAZENÍ ZVOLENÉHO ŘEŠENÍ

Celý program byl nasazen na testovacím serveru na Ústavu telekomunikací FEKT VUT v Brně. Jelikož je na serveru nainstalován operační systém Debian, následující informace budou platit pro něj (mělo by však být možné postup přizpůsobit a aplikovat na libovolnou distribuci).

Závislosti

Program pro svůj běh vyžaduje Python ve verzi 2.6, navíc kromě knihoven, které jsou standardně v systému (předpokládám již nainstalované *KVM* a *libvirt*), vyžaduje balíčky `python2.6-libvirt`, `python2.6-openssl` a `python-sqlalchemy`. Tyto balíčky je možné doinstalovat příkazem:

```
# aptitude install python2.6-libvirt python2.6-openssl \
    python-sqlalchemy
```

Vytvoření uživatelského účtu

Vytvoření neprivilegovaného uživatele, pod kterým poběží hlavní část programu a vytvoření jeho domovského adresáře zajistíme následujícími příkazy:

```
# useradd --system --home-dir /home/ninthlama \
    --shell /bin/false  ninthlama
# mkdir /home/ninthlama
# chown ninthlama:ninthlama /home/ninthlama
# mkdir /home/ninthlama/backend
# chown ninthlama:ninthlama /home/ninthlama/backend
```

Poslední dva příkazy vytvářejí a nastavují adresář `backend`, který bude obsahovat aplikaci NinthLama-backend. Do tohoto adresáře nakopírujeme zdrojové kódy včetně podadresářů `lib`, `schemas` a `templates`.

Vygenerování serverového certifikátu

V dalším kroku je potřeba získat serverový certifikát (chceme-li využívat možnost šifrování komunikace). Nejsme-li majiteli komerčního certifikátu, můžeme si vygenerovat certifikát a odpovídající tajný klíč pomocí příkazu:

```
# openssl req -new -x509 -nodes -out server.cert.pem -keyout \
    server.key.pem -days 1098
```

Získanou dvojici souborů `server.cert.pem` a `server.key.pem` je vhodné uložit například do adresáře `certs`.

Adresářová struktura

Dále je potřeba připravit místo pro virtuální disky, případně pro další používané soubory. Já jsem zvolil následující strukturu:

```
~ninthlama/data/
    |-- direct_kernel_boot
    |-- ninthlama_cd
    |-- storage_pool
    '-- www,
```

kde `~ninthlama` představuje domácí adresář uživatele ninthlama (cestu k němu jsme zadávali při vytváření uživatele). Adresáře `storage_pool` a `ninthlama_cd` jsou úložiště pevných disků a instalačních obrazů CD. Jednotlivé virtuální pevné disky jsou vytvářeny na základě požadavků, instalační média je potřeba stáhnou a nakopírovat do zvoleného adresáře ručně. V adresáři `direct_kernel_boot` jsou uloženy jádra a initrd soubory pro přímý boot jádra. Adresář `www` je kořenový adresář webového serveru (bude zmíněno dále), odtud je stahována konfigurace pro automatické instalace.

Konfigurace programu NinthLama a přizpůsobení šablon

Nyní je potřeba nakonfigurovat hlavní konfigurační soubor programu NinthLama. Jeho konkrétní části jsou popsány v předchozí kapitole. Největší pozornost je potřeba věnovat nastavení cest k souborům a konfiguraci XML–RPC serveru *worker*u a *manager*u. Toto nastavení mimo jiné určuje, zda bude některá část dostupná ze sítě či jenom v rámci lokálního počítače.

Součástí zdrojových kódů jsou též ukázkové šablony. Ty je vhodné upravit a přizpůsobit vlastním potřebám. Mají-li být například vytvořené domény přístupné přes protokol VNC i z jiného počítače, je potřeba v šabloně pro doménu přidat/upravit atribut `listen` u elementu <`graphics`> konfigurujícího VNC server a nastavit jej na veřejnou adresu serveru.

Podle zvoleného nastavení v hlavním konfiguračním souboru je také potřeba upravit šablony pro úložiště virtuálních disků, obrazů CD a síť.

První spuštění

Při prvním spuštění je nejlepší postupně spouštět jednotlivé části v ladícím režimu, odhalíme tak jednoduše případné chyby v konfiguraci či závislostech.

Prvně spustíme část *worker* příkazem

```
# ./nl_worker.py -d start
```

Pokud start proběhne v pořádku, můžeme jej zastavit (klávesami CTRL+c) a spustit v režimu démona (příkazem bez parametru `-d`). Necháme-li jej však ještě běžet v ladícím režimu, uvidíme navázání spojení z *manager*u.

Obdobným způsobem nyní můžeme spustit *manager* příkazem

```
# ./nl_manager.py -d start
```

Pokud obě části proběhnou v pořádku, můžeme je zastavit a spustit obě najednou s pomocí ovládacího skriptu `ninthlama-backend.py`:

```
# ninthlama-backend.py start
```

V tuto chvíli ještě není možné se k serveru z vnějšku připojit, neboť neexistuje žádný uživatel, pod kterým bychom se mohli autentizovat. Pro vytvoření uživatele (se jménem `admin` a heslem `heslo`) použijeme také skript `ninthlama-backend.py` v následující podobě:

```
# ninthlama-backend.py --user admin --password heslo --level \
    5 --description "Administrator"
```

Nyní by již mělo být možné připojit se k serveru například s pomocí klientského rozhraní `ninthlama-client.py`.

Vytvoření webového serveru

Pro zpřístupnění konfiguračních souborů pro automatické instalace lze využít libovolný existující webový server přístupný z virtuální sítě. Pokud zatím žádný nepoužíváme, je nejjednodušším řešením server *lighttpd*[1] ze stejnojmenného balíčku, který je možno nainstalovat příkazem:

```
# aptitude install lighttpd
```

Jedná se o rychlý nenáročný webový server s jednoduchou konfigurací. Ta je uložena v adresáři `/etc/lighttpd/` v souboru `lighttpd.conf`. Pro naše potřeby jsou důležité volby `server.document-root`, `server.bind` a `server.port`. První

[1] <http://www.lighttpd.net/>

určuje uložení kořenového adresáře webového serveru, druhé dvě adresu a port na kterém má *lighttpd* naslouchat.

Z důvodu ladění je vhodné zapnout modul `accesslog` pro logování přístupů. Povolení modulu je založeno na vytvoření symbolického odkazu z adresáře `conf-available` do `conf-enabled`, nejlépe pomoci nástroje `lighttpd-enable-mod`. Záznamy jsou poté ukládány do souboru na adrese `/var/log/lighttpd/access.log`. V případě testování automatické instalace například za pomoci KickStartu je možné z logu poznat, zda došlo ke stažení konfiguračního souboru či nikoli.

5 ZÁVĚR

Práce se zabývá možnostmi hromadné správy virtuálních strojů z programátorského hlediska. V první části je rozebírána knihovna *libvirt* umožňující správu virtuálních strojů postavených na různých virtualizačních řešeních.

Vytvářené virtuální stroje, sítě a další prvky jsou definovány pomocí XML předpisu a takto předány *libvirtu* ke zpracování. Popisu struktury a možností XML souboru pro vytváření virtuálních strojů, disků a sítí je věnována první část práce.

Další kapitola je věnována možnostem automatické instalace operačního systému na vytvořené virtuální stroje. Je bráno v úvahu, že disky virtuálních strojů mohou být obyčejné soubory a je tedy možné naklonovat existující virtuální disk s již nainstalovaným systémem. Dále jsou představeny a popsány možnosti automatické instalace na základě předepsaných pravidel pro dvě základní rodiny distribucí operačního systému GNU/Linux, pro distribuce založené na RedHatu a pro distribuci Debian. První skupině distribucí je určen nástroj *KickStart*, pro Debian se jedná o takzvanou „předpřipravenou“ instalaci.

Praktická část práce se zabývá návrhem a realizací programu NinthLama–backend, tedy programu poskytujícímu možnosti automatického vytváření a správy většího množství virtuálních strojů. Jedná se o serverovou aplikaci, jejíž funkce může využívat webový interface, terminálový klient či jakýkoliv program přistupující k serveru s pomocí XML–RPC s nadefinovaným rozhraním. Program byl nasazen a úspěšně odzkoušen na testovacím serveru na Ústavu telekomunikací FEKT VUT v Brně.

Navržený a realizovaný program umožňuje:

- zprostředkování komunikace mezi klientskými programy a různými virtualizačními technologiemi,
- vytváření virtuálních strojů na základě připravených šablon,
- instalaci operačního systému ze sítě, z instalačních obrazů,
- plánovanou automatickou instalaci s pomocí rozličných automatizačních nástrojů např. KickStart,
- nastavení manuální instalace OS,
- správu oprávnění k jednotlivým virtuálním strojům a jejich přiřazení k projektům.

LITERATURA

[1] ActiveState Code : Python recipes [online]. c2011 [cit. 2011-05-20]. Simple XML RPC server over HTTPS (Python recipe) . Dostupné z WWW: <http://code.activestate.com/recipes/496786/>.

[2] *Debian GNU/Linux Installation Guide* [online]. c2010 [cit. 2011-04-20]. Dostupné z WWW: <http://www.debian.org/releases/stable/i386/>.

[3] GOERZEN, John. *Foundations of Python Network Programming*. USA : Apress, 2004. 512 s. ISBN 1-59059-371-5.

[4] HAMILTON, Martin. *Internet FAQ Archives* [online]. V0.2. 11 January 1999 [cit. 2010-11-27]. RedHat Linux KickStart HOWTO. Dostupné z WWW: <http://www.faqs.org/docs/Linux-HOWTO/KickStart-HOWTO.html>.

[5] HARMS, Daryl; MCDONALD, Kenneth. *Začínáme programovat v jazyce Python*. Vyd. 1. Praha : Computer Press, a.s, 2003. 456 s. ISBN 80-7226-799-X.

[6] JONES, Christopher A.; DRAKE, Fred L. *Python & XML*. First Edition. USA : O'Reilly, 2002. 384 s. ISBN 0-596-00128-2.

[7] *KVM : Kernel Based Virtual Machine* [online]. 2005 [cit. 2010-11-13]. Dostupné z WWW: <http://www.linux-kvm.org/>.

[8] LANGE, Thomas. *FAI – Fully Automatic Installation* [online]. 2010 [cit. 2010-12-1]. Dostupné z WWW: <http://fai-project.org/>.

[9] NEMETH, Evi; SNYDER, Garth; HEIN, Trent R. *Linux : Kompletní příručka administrátora*. Překlad: David Vozák, Miloš Průdek, Lubomír Ptáček, Jiří Huf, Jiří Berka. 2. aktualizované vydání. Brno : Computer Press, a.s, 2008. 984 s. ISBN 978-80-251-2410-9.

[10] *libvirt : The virtualization API* [online]. 2005 [cit. 2010-11-17]. Dostupné z WWW: <http://libvirt.org/>.

[11] *libvirt : The virtualization API* [online]. 2010 [cit. 2010-11-18]. Storage volume encryption XML format. Dostupné z WWW: <http://libvirt.org/formatstorageencryption.html>.

[12] MLÝNKOVÁ, I., et al. *XML technologie : principy a aplikace v praxi*. 1. vyd. Praha : Grada, 2008. 267 s. ISBN 978-80-247-2725-7.

[13] *Open Source Initiative* [online]. version 2.1. February 1999 [cit. 2010-11-13]. GNU Lesser General Public License. Dostupné z WWW: <http://www.opensource.org/licenses/lgpl-license.html>.

[14] *Python v2.6.6 documentation* [online]. c2010, Last updated on Aug 24, 2010. [cit. 2011-3-20]. Dostupné z WWW: <http://docs.python.org/release/2.6.6/>.

[15] *Python v2.6.6 documentation* [online]. c2010, Last updated on Aug 24, 2010. [cit. 2011-05-2]. Xmlrpclib – XML-RPC client access. Dostupné z WWW: <http://docs.python.org/release/2.6.6/library/xmlrpclib.html>.

[16] *Python v2.6.6 documentation* [online]. c2010, Last updated on Aug 24, 2010. [cit. 2011-05-2]. SimpleXMLRPCServer – Basic XML-RPC server. Dostupné z WWW: <http://docs.python.org/release/2.6.6/library/simplexmlrpcserver.html>.

[17] *Red Hat Documentation : Chapter 31. Kickstart Installations* [online]. c2010 [cit. 2010-11-27]. Dostupné z WWW: <http://docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/5/html/Installation_Guide/ch-kickstart2.html>.

[18] Red Hat, Inc. *Libvirt : The virtualization API* [online]. Revision 4. c2010, Mon Aug 23 2010 [cit. 2010-11-17]. Application Development Guide (libvirt 0.7.5). Dostupné z WWW: <http://libvirt.org/guide/html-single/>.

[19] RUEST, Danielle; RUEST, Nelson. *Virtualizace : podrobný průvodce*. Vyd. 1. Brno : Computer Press, 2010. 408 s. ISBN 978-80-251-2676-9.

[20] *SQLAlchemy Documentation* [online]. Version: 0.6.8. c2011, Last Updated: 04/22/2011 [cit. 2011-04-10]. Dostupné z WWW: <http://www.sqlalchemy.org/docs/index.html>.

[21] *The Internet Engineering Task Force* [online]. August 1998 [cit. 2010-11-15]. RFC 2396 – Uniform Resource Identifiers (URI): Generic Syntax. Dostupné z WWW: <http://www.ietf.org/rfc/rfc2396.txt>.

[22] *The Internet Engineering Task Force* [online]. July 2005 [cit. 2010-11-17]. RFC 4122 – A Universally Unique IDentifier (UUID) URN Namespace. Dostupné z WWW: <http://www.ietf.org/rfc/rfc4122.txt>.

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
KVM	Kernel-based Virtual Machine – virtualizace na úrovni jádra
LAN	Local Area Network – lokální síť
LVM	Logical Volume Management – řízení logických disků
MAC	Media Access Control (address)
NAT	Network Address Translation – překlad síťových adres
NFS	Network File System – protokol pro připojování síťových disků
OS	Operační systém
RDP	Remote Desktop Protocol
RPC	Remote procedure call – vzdálené volání procedur
SSH	Secure Shell
SSL	Secure Sockets Layer
TLS	Transport Layer Security – zabezpečení transportní vrstvy
URI	Uniform Resource Identifier – jednotný identifikátor zdroje
UTC	Coordinated Universal Time – koordinovaný světový čas
UUID	Universally unique identifier – univerzální unikátní identifikátor
VNC	Virtual Network Computing
XML	Extensible Markup Language – rozšiřitelný značkovací jazyk

SEZNAM PŘÍLOH

A KONFIGURAČNÍ SOUBORY AUTOMATICKÉ INSTALACE

A.1 KickStart

```
# Instalace systému (muže být upgrade)
install
# Instalace bude probíhat v textovém režimu
text
# Adresa pro získání instalačních souborů (může být cdrom ↩
    pro instalaci z_CD)
url --url http://mirror.centos.org/centos/5.6/os/i386
# Jazyk
lang cs_CZ.UTF-8
# Rozložení klávesnice
keyboard cz-us-qwertz
# Nastavení síťove karty eth0 (nastavení z DHCP serveru)
network --device eth0 --bootproto dhcp
# Root heslo (poslední znak dolaru na řádek nepatří)
rootpw --iscrypted $1$MDAHekLS$uDBwerX5KxoWERGR.mgsK1$
# Firewall
firewall --enabled --port=22:tcp
# Způsob ukládání hesel
authconfig --enableshadow --enablemd5
# Časové pásmo
timezone --utc Europe/Prague
# Umístění zavaděče systému
bootloader --location=mbr --driveorder=vda
# Smazání všech diskových oddílů
clearpart --all --initlabel
# Vytvoření oddílu pro swap
part swap --size 128
# Vytvoření systémového oddílu
part / --size 500 --grow

# Instalované balíčky (nebo skupiny balíčků)
%packages
```

```
@base
@core
@editors
@text-internet
keyutils
openssh-server
device-mapper-multipath
```

A.2 Pro distribuci Debian

```
### Localization
# Preseeding only locale sets language, country and locale.
d-i debian-installer/locale string cs_CZ

# Keyboard selection.
d-i console-keymaps-at/keymap select us
d-i keyboard-configuration/xkb-keymap select us

### Network configuration
# netcfg will choose an interface that has link if possible.
# This makes it skip displaying a list if there is more
# than one interface.
d-i netcfg/choose_interface select auto

# Any hostname and domain names assigned from dhcp take
# precedence over values set here.
# However, setting the values still prevents the questions
# from being shown, even if values come from dhcp.
d-i netcfg/get_hostname string unassigned-hostname
d-i netcfg/get_domain string unassigned-domain

### Mirror settings
# If you select ftp, the mirror/country string
# does not need to be set.
#d-i mirror/protocol string ftp
d-i mirror/country string manual
d-i mirror/http/hostname string http.us.debian.org
```

```
d-i mirror/http/directory string /debian
d-i mirror/http/proxy string

# Root password, either in clear text
d-i passwd/root-password password heslo
d-i passwd/root-password-again password heslo
# or encrypted using an MD5 hash.
#d-i passwd/root-password-crypted password [MD5 hash]

# To create a normal user account.
d-i passwd/user-fullname string Test
d-i passwd/username string test
# Normal user's password, either in clear text
d-i passwd/user-password password test
d-i passwd/user-password-again password test
# or encrypted using an MD5 hash.
#d-i passwd/user-password-crypted password [MD5 hash]

### Clock and time zone setup
# Controls whether or not the hardware clock is set to UTC.
d-i clock-setup/utc boolean true

# You may set this to any valid setting for \$TZ;
# see the contents of /usr/share/zoneinfo/ for valid values.
d-i time/zone string Europe/Prague

# Controls whether to use NTP
# to set the clock during the install
d-i clock-setup/ntp boolean true
# NTP server to use. The default is almost always fine here.
#d-i clock-setup/ntp-server string ntp.example.com

### Partitioning
# The presently available methods are:
# - regular: use the usual partition types
#            for your architecture
# - lvm:     use LVM to partition the disk
# - crypto:  use LVM within an encrypted partition
```

```
d-i partman-auto/method string regular

# If one of the disks that are going to be automatically
# partitioned contains an old LVM configuration,
# the user will normally receive a warning.
# This can be preseeded away...
d-i partman-lvm/device_remove_lvm boolean true
# The same applies to pre-existing software RAID array:
d-i partman-md/device_remove_md boolean true
# And the same goes for the confirmation
# to write the lvm partitions.
d-i partman-lvm/confirm boolean true

# You can choose one of the three predefined
# partitioning recipes:
# - atomic: all files in one partition
# - home:   separate /home partition
# - multi:  separate /home, /usr, /var, and /tmp partitions
d-i partman-auto/choose_recipe select atomic

# This makes partman automatically partition
# without confirmation, provided that you told it
# what to do using one of the methods above.
d-i partman-partitioning/confirm_write_new_label boolean ↩
    true
d-i partman/choose_partition select finish
d-i partman/confirm boolean true
d-i partman/confirm_nooverwrite boolean true

### Apt setup
# You can choose to install non-free and contrib software.
d-i apt-setup/non-free boolean true
d-i apt-setup/contrib boolean true

### Package selection
tasksel tasksel/first multiselect standard

# Individual additional packages to install
```

```
d-i pkgsel/include string openssh-server mc

### Finishing up the installation
# Avoid that last message about the install being complete.
d-i finish-install/reboot_in_progress note
```

B PŘÍKLADY ŠABLON

B.1 Šablona virtuálních strojů

```
<domain type='kvm'>
  <name>basic_domain</name>
  <description>
    Popis šablony testovacího stroje.
    Základní doména
    RAM: 256MB
  </description>
  <memory>262144</memory>
  <currentMemory>262144</currentMemory>
  <vcpu>1</vcpu>
  <os>
    <type arch='i686' machine='pc-0.12'>hvm</type>
    <boot dev='hd'/>
  </os>
  <features>
    <acpi/> <apic/> <pae/>
  </features>
  <clock offset='utc'/>
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>restart</on_crash>
  <devices>
    <emulator>/usr/bin/kvm</emulator>
    <console type='pty'>
      <target port='0'/>
    </console>
    <input type='mouse' bus='ps2'/>
    <graphics type='vnc' autoport='yes'/>
    <video>
      <model type='cirrus' vram='16384' heads='1'/>
    </video>
  </devices>
</domain>
```

B.2 Šablona virtuálních disků

```
<volume>
  <name>volume_3GB</name>
  <source/>
  <capacity unit="M">3000</capacity>
  <target>
    <format type='raw'/>
  </target>
</volume>
```

B.3 Šablona diskových úložišť (disků)

```
<pool type="dir">
  <name>ninthlama</name>
  <target>
    <path>/home/ninthlama/data/storage_pool</path>
  </target>
</pool>
```

B.4 Šablona virtuální sítě

```
<network>
  <name>ninthlama</name>
  <forward mode='nat'/>
  <bridge name='virbr1' stp='on' delay='0' />
  <ip address='192.168.123.1' netmask='255.255.255.0'>
      <dhcp>
          <range start='192.168.123.100' end='↩
              192.168.123.254' />
      </dhcp>
  </ip>
</network>
```

B.5 Šablony instalací

B.5.1 Manuální instalace

```
<installation type='manual_install' subtype='install_image'>
  <name>basic_manual_install_debian</name>
  <source>/home/ninthlama/data/ninthlama_cd/debian-506-i386-↩
      netinst.iso</source>
  <description>
    Sablona pro manualni instalaci Debianu
    z instalacniho media.
  </description>
</installation>
```

B.5.2 Automatická instalace

```
<installation type='automated_install' subtype='↩
    direct_kernel_boot'>
  <name>basic_direct_kernel_boot</name>
  <description>
    Sablona pro automatickou instalaci CentOS
    s pomoci direct kernel boot (primeho bootu jadra).
  </description>
  <kernel>/home/ninthlama/data/direct_kernel_boot/↩
      vmlinuz_centos</kernel>
  <initrd>/home/ninthlama/data/direct_kernel_boot/↩
      initrd_centos</initrd>
  <cmdline>ks=http://192.168.123.1/ks.cfg</cmdline>
</installation>
```

```
<installation type='automated_install' subtype='↩
    direct_kernel_boot'>
  <name>automatic_debian_installation</name>
  <description>
    Sablona pro automatickou instalaci Debianu
    s pomoci direct kernel boot (primeho bootu jadra).
  </description>
  <kernel>/home/ninthlama/direct_kernel_boot/vmlinuz_debian
```

```
    </kernel>
  <initrd>/home/ninthlama/direct_kernel_boot/initrd_debian
    </initrd>
  <cmdline>auto=true priority=critical interface=auto netcfg↩
      /dhcp_timeout=60 preseed/url=http://192.168.123.1/↩
      preseed.cfg</cmdline>
  <source>/home/ninthlama/data/ninthlama_cd/debian-506-i386-↩
      netinst.iso</source>
</installation>
```

B.5.3 Kopírování disku s nainstalovaným systémem

```
<installation type='copy'>
  <name>basic_copy</name>
  <description>
    Sablona kopirujici existujici disk.
  </description>
  <source>/home/ninthlama/data/storage_pool/Debian_basic.img↩
      </source>
</installation>
```

C POPISNÝ SOUBOR K ŠABLONÁM PRO INSTALACI

```
<?xml version="1.0"?>
<!-- A Relax NG schema for the NinthLama installation XML ↩
    format -->
<grammar xmlns="http://relaxng.org/ns/structure/1.0" ↩
    datatypeLibrary="http://www.w3.org/2001/XMLSchema-↩
    datatypes">
  <!-- We handle only document defining a installation-->
  <start>
    <ref name="installation"/>
  </start>

  <!--
  description element, maybe placed anywhere under the root
    -->
  <define name="description">
    <element name="description">
      <text/>
    </element>
  </define>

  <!--
      We handle only document defining a installation
    -->
  <define name="installation">
    <element name="installation">
      <!-- Name of installation template -->
      <element name="name">
        <ref name="templateName"/>
      </element>
      <interleave>
        <optional>
          <ref name="description"/>
        </optional>
        <choice>
```

```
        <!--chose installation type-->
        <ref name="type_copy"/>
        <ref name="type_manual_install"/>
        <ref name="type_direct_kernel_boot"/>
      </choice>
    </interleave>
  </element>
</define>

<!-- Installation copy: copy an existing disc.-->
<define name="type_copy">
  <attribute name="type">
    <value>copy</value>
  </attribute>
  <!-- Absolute file path of source disk -->
  <element name="source">
    <ref name="absFilePath"/>
  </element>
</define>

<!-- Installation manual_install:
  manual instalation from install media image.-->
<define name="type_manual_install">
  <attribute name="type">
    <value>manual_install</value>
  </attribute>
  <attribute name="subtype">
    <value>install_image</value>
  </attribute>
  <!-- Absolute file path of installation media image -->
  <element name="source">
    <ref name="absFilePath"/>
  </element>
</define>

<!-- Installation direct_kernel_boot:
  instalation through direct kernel boot with boot ↩
      parameters.-->
```

```
<define name="type_direct_kernel_boot">
  <attribute name="type">
    <value>automated_install</value>
  </attribute>
  <attribute name="subtype">
    <value>direct_kernel_boot</value>
  </attribute>
  <interleave>
    <element name="kernel">
      <ref name="absFilePath"/>
    </element>
    <optional>
      <element name="initrd">
        <ref name="absFilePath"/>
      </element>
    </optional>
    <optional>
      <element name="root">
        <ref name="devicePath"/>
      </element>
    </optional>
    <optional>
      <element name="cmdline">
        <text/>
      </element>
    </optional>
    <optional>
      <element name="source">
        <ref name="absFilePath"/>
      </element>
    </optional>
  </interleave>
</define>

<define name="templateName">
  <data type="string">
    <param name="pattern">[A-Za-z0-9_\.\+\-&amp;:/]+
      </param>
```

```
    </data>
  </define>

  <define name="absFilePath">
    <data type="string">
      <param name="pattern">/[a-zA-Z0-9_\.\+\-&amp;/%]+
        </param>
    </data>
  </define>

  <define name="devicePath">
    <data type="string">
      <param name="pattern">/[a-zA-Z0-9_\+\-/%]+</param>
    </data>
  </define>
</grammar>
```

D DOKUMENTACE K VNĚJŠÍMU ROZHRANÍ

addMissingDomainsToDatabase()

```
Porovná existující domény s databází, chybějící přidá.
```

addOwnershipDomain(user_name, domain_name)

```
Přidání vlastnictví k doméně.
```

createDisk(name, template, kwargs={})

```
Vytvoření disku z šablony template s názvem name.

Další volitelné možnosti konfigurace jsou předány
pomocí slovníku kwargs.
kwargs = {
  'capacity': KAPACITA, # Změna kapacity disku oproti šabloně.
  'unit': JEDNOTKY,     # Jednotka ve kterých je uvedeno capacity
                        # (viz. nl_xml.LibvirtXmlStorageVolume.
                        setCapacity).
  'owner': 'VLASTNIK',  # Přiřazení disku určitému vlastníkovi.
}
```

createDomain(name, template, kwargs={})

```
Vytvoření domény se jménem name podle šablony template.

Další možnosti konfigurace (povinné i nepovinné) jsou předány
pomocí slovníku kwargs.
kwargs = {
  'description': '''Popis k vytvářené doméně.''',
  'disk': CESTA_K_DISKU,  # absolutní cesta k disku
  'cd': CESTA_K_OBRAZU_CD # absolutní cesta k obrazu cd
  'owner': 'VLASTNIK',    # přiřazení domény určitému vlastníkovi.
  'boot_sequence':['hd','cdrom'], # pořadí pro bootování
  'project': ID_PROJEKTU  # číslo projektu
                          # do kterého má doména patřit
}
```

deleteDisk(name)

```
Smazání disku.
```

deleteDomain(name, delete_associated_disks=False)

```
Smazání domény 'name'.

Je-li delete_associated_disks True, budou smazány
i všechny disky připojené k této doméně.
```

deleteOwnershipDomain(user_name, domain_name)

```
Odebrání vlastnictví uživateli.
```

getCDPath(name)

```
Získání absolutní cesty k obrazu CD.
```

getDiskPath(name)

```
Získání absolutní cesty k disku.
```

getDomainXml(domain_name)

```
Získání xml předpisu domény.
```

infoDomain(domain_name)

```
Vrátí kompletní informace o doméně
(jako v případě listování více domén).
```

install(domain_name, template, priority=50)

```
Zajištění instalace (nebo přípravy na instalaci) podle šablony.
Parametr 'priority' udává prioritu požadavku
(0 největší, 50 výchozí,...)
```

installFinish(domain_name)

```
Dokončení (manuální) instalace.
Odpojení instalačních obrazů, změna bootovací sekvence.
```

listCDs()

```
Výpis seznamu CD obrazů.
Vrací seznam obrazů.
```

listDisks(user=None)

```
Výpis seznamu disků.
Parametr user slouží jako filtr.
Vrací seznam disků.
```

listDomains(user=None, project=None)

```
Výpis seznamu domén.
Parametr user slouží jako filtr.
Vrací seznam domén.
```

listDomainsByProject(project)

```
Výpis seznamu domén,
(stejně jako listDomains, pouze se filtruje podle ID projektu).
```

listDomainsByProjectInfo(project)

```
Výpis seznamu domén,
(stejně jako listDomainsInfo, pouze se filtruje podle ID projektu).
```

listDomainsInfo(user=None, project=None)

```
Výpis seznamu domén.

Parametr user slouží jako filtr.
Vrací slovník slovníků (ke každé doméně jeden slovník):
{'jmeno_domeny':
    {'description': 'POPIS',
    'id': ID,
    'last_edited_time': ČAS PODLEDNÍ EDITACE,
    'last_started_time': ČAS POSLEDNÍHO SPUŠTĚNÍ,
    'name': 'JMÉNO',
    'nic': [['MAC', 'IP'], ['MAC2', 'IP2']],
    'owners': ['VLASTNIK', VLASTNIK2],
    'database_status': 'STATUS_V_DATABÁZI',
    'libvirt_status': 'STATUS_V_LIBVIRTU',
    'uuid': 'UUID',
    'project': PROJEKT_ID},
'jmeno_2_domeny':
    { ...}, ...}

kde: ČASY jsou ve formátu datetime
```

```
  STATUS_V_LIBVIRTU může být: RUNNING, STOPED, UNDEFINED
  STATUS_V_DATABÁZI může být: UNKNOWN, LOCKED, READY,
                              INSTALLATION, MANUAL_INSTALL
```

listInstallations()

```
Výpis fronty požadavků na instalaci.
```

listTemplatesDisk()

```
Výpis šablon pro vytvoření disků.
```

listTemplatesDomain()

```
Výpis šablon pro vytvoření domén.
```

listTemplatesInstallation()

```
Výpis šablon pro instalace.
```

listUserNames()

```
Výpis dostupných uživatelů.
```

mountCD(domain_name, cd_path)

```
Připojení CD k doméně.
```

pingWorker()

```
Metoda testující spojení na workera.
```

removeDirectKernelBoot(domain_name)

```
Odstranění nastavení direct_kernel_boot z domény.
```

setBootSequence(domain_name, boot_sequence)

```
Nastavení boot_sequence u vybrané domény
(doména musí být ve stavu STOPED).
boot_sequence = ['hd', 'cdrom']
```

setDomainProject(domain_name, project)

```
Přiřazení domény do projektu.
```

startDomain(domain, user=None)

```
Spuštění domény domain.
Parametr user je využit v metodě pro autorizaci,
v této metodě využit není.
```

statusDomain(domain)

```
Zjištění stavu domény.
```

stopDomain(domain, user=None)

```
Zastavení domény domain.
Parametr user je využit v metodě pro autorizaci,
v této metodě využit není.
```

umountCD(domain_name)

```
Odpojení CD od vybrané domény.
```

E OBSAH PŘILOŽENÉHO MÉDIA

Diplomova_prace_Daniel_Horak.pdf – text diplomové práce

NinthLama-backend/ – zdrojové kódy aplikace NinthLama–backend

Documentation/ – Dokumentace k jednotlivým modulům programu NinthLama–backend ve formátu HTML (vygenerováno s pomocí `pydoc`)