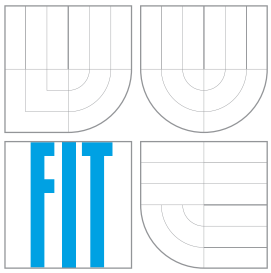
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

# TESTOVÁNÍ PROGRAMOVÉHO VYBAVENÍ
SOFTWARE TESTING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MIROSLAV VADKERTI
AUTHOR

VEDÚCI PRÁCE — Ing. MARTÍNEK TOMÁŠ
SUPERVISOR

BRNO 2007

# Zadanie

## Testování programového vybavení

1. Seznamte se strukturou zdrojového kódu programového vybavení vytvořeného v rámci projektu Liberouter.
2. Navrhněte metodiku testování zdrojového kódu v rámci projektu Liberouter. Zaměřte se na automatizace testů k ověření funkce programového vybavení a detekci chyb ve zdrojovém kódu.
3. Navrhněte metodiku testování konfigurace, kompatibility a použitelnosti vytvořeného programového vybavení v rámci projektu Liberouter.
4. Vytvořte testovací server a prakticky zvolené metodiky testování ověřte. Proveďte měření výskytu chyb ve zdrojovém kódu a chyb při nasazení.

Kategorie: Softwarové inženýrství

# Licenční smlouva

Licenční smlouva je uložena v archívu Fakulty informačních technologií Vysokého učení technického v Brně.

# Abstrakt

Práca si kladie za cieľ vytvoriť metodiku pre automatizované testovanie zdrojového kódu vytvoreného v rámci projektu Liberouter. Navrhnutá metodika umožňuje na jednom počítači testovať kopatibilitu a použitelnosť programového vybavenia vzhľadom na širokú škálu variant podporovaných platform. Metodika je implementovaná v programovacom jazyku Perl ako automatizovaný server s využitím virtualizačného softvéru VMware Server.



# Kľúčová slová

VMware, Perl, Liberouter, Expect, statické testovanie kódu



# Abstract

The aim of the thesis is to design a metholodogy for automatized source code testing for the project Liberouter. The designed metholodogy allows testing the compatibility and usability of the given software on a large scale of variants of supported platforms. The metholodogy is implemented in programming language Perl as an automatized server working with VMware Server virtualization software.



# Keywords

VMware, Perl, Liberouter, Expect, static code analasys


# Citácie

Miroslav Vadkerti: Testování programového vybavení, bakalářská práce, Brno, FIT VUT v Brně, 2007

Testovanie programového vybavenia

## Prehlásenie

Prehlasujem, že túto prácu som vypracoval samostatne, pod vedením Ing. Tomáša Martínka, s použítím literatúry a ďalších zdrojov uvedených na konci tejto práce.

.....
Miroslav Vadkerti
11. mája 2007

# Obsah

# Kapitola 1

# Úvod

S masívnym nástupom informačných technológií od 80. rokov 20. storočia sa počítače rozšírili do všetkých kútov ľudskej činnosti. Overovanie správnej funkcionality, kvality a bezpečnosti sa preto stalo prirodzeným požiadavkom pre vývojový cyklus každého softvérového produktu. Pri veľkom rozsahu projektu, početnosti podporovaných platforiem, prípadne závislostiach na konfiguráciách týchto platforiem, však nastáva rada problémov. Neautomatizované testovanie sa často stáva z časového hľadiska neúnosným. Snaha je preto v testovaní kladená na automatizovanie testových operácií. V súčasnosti nastáva trend využitia virtualizačných nástrojov pre potreby automatizovaného testovania. Toto riešenie prináša viacero výhod nielen z hľadiska úspory času, ale i prostriedkov.

Táto práca si kladie za cieľ vytvoriť metodiku pre automatizované testovanie zdrojového kódu vytvoreného v rámci projektu Liberouter. Projekt Liberouter je zameraný na problematiku náročných sieťových aplikácií, ktoré k urýchleniu využívajú špeciálny hardvér na bázi technológie FPGA. V rámci projektov EÚ 6NET, SCAMPI a GN2 sa projekt podieľa na vývoji IPv6 smerovača, monitorovacieho adaptéru, sondy na bázi NetFlow s cieľovou priepustnosťou 1Gbps a 10Gbps a niektorých ďalších sieťových zariadení.

Testovanie softvéru nie je triviálne a samostatne sa delí na viacero skupín zaoberajúcich sa rôznymi vlastnosťami softvérového produktu. Kapitola 2 opisuje obecné teoretické princípy testovania softvéru a zameriava sa na statické testovanie zdrojového kódu. Virtualizácia a jej úloha v testovaní softvéru je predmetom sekcie 2.4. Pojednáva o virtualizačných metódach a nástrojoch. Poukazuje na výhody, ktoré tieto systémy pri testovaní prinášajú a uvádza ich kľúčové vlastnosti využiteľné pri testovaní.

Usporiadanie programového vybavenia vytvoreného v rámci projektu Liberouter je predmetom kratšej kapitoly 3. Opisuje rozdelenie programového vybavenia a snaží sa z neho vybrať vhodný objekt testovania.

Kapitola 4 sa zaoberá návrhom a implementáciou automatizovaného testovacieho servera LiTS – Liberouter Testing Server podľa navrhnutej metodiky a popisuje vlastnosti vytvoreného riešenia.

# Kapitola 2

# Testovanie softvéru

Testovanie je v dnešnej dobe jednou z neodmysliteľných súčastí vývojového ako aj životného cyklu softvéru. Ide o technickú disciplínu neustálej konfrontácie produktu s jej špecifikáciou a požadovanými vlastnosťami pre získanie informácií o kvalite testovaného produktu. Pod pojmom testovanie sa teda myslí proces identifikácie správnosti, úplnosti, bezpečnosti a kvality vyvíjaného softvérového produktu. V oblasti softvérového testovania je potrebné rozlíšiť dva pojmy:

- Chybu softvéru – failure
- Zlyhanie softvéru – fault

V prípade chyby softvéru, nesplnil softvér svoju úlohu z pohľadu užívateľa, neurobil teda to čo od neho užívateľ očakával. Zlyhanie softvéru naopak vyjadruje chybový, nekorektný stav z pohľadu správnosti sémantiky programu. Zlyhanie sa môže stať chybou ak sa splnia jeho kritéria a nesprávna časť kódu je spustená riadiacou jednotkou – procesorom. Ďalším možným dôvodom zmeny zlyhania na chybu softvéru môže byť prenos softvéru na novú platformu, prechod na novší či nie úplne kompatibilný prekladač, či pri rozširovaní aktuálnej verzie softvéru o nové vlastnosti.

Nezávisle od použitých metód testovania je jeho výsledkom len istý stupeň uspokojenia požiadavkov zo strany tvorcu, či objednávateľa testovaného projektu. Toto uspokojenie sa vyjadruje v miere chybovosti kódu. Vyžadovaná miera chybovosti kódu sa odvíja od typu softvéru. Vyžadovaná miera bude samozrejme nižšia pri softvérovom simulátore automobilu ako pri softvéri v riadiacej jednotke reálneho auta.

## 2.1 Základné praktiky testovania

Problémom softvérového testovania sú nepravidelne sa objavujúce chyby vo vyvíjanom softvéri. Tieto chyby sa obtiažne lokalizujú a ich príčiny zostávajú preto často neodhalené. Nie moc správnym prístupom je pravidlo, že softvér, ktorého bezporuchový beh je plánovaný na určitú dobu, musí byť testovaný minimálne rovnako dlhý čas. Toto má dopad na softvér s predpokladanou dlhou dobou spoľahlivého behu, kde sa stáva nereálne testovanie požadovaný dlhý čas. Akceptovateľná hranica času sa pohybuje v rámci dní alebo týždňov. Dlhšia doba behu musí byť za normálnych okolností odsimulovaná.

Najčastejšou praktikou softvérového testovania býva testovanie nezávislou skupinou testerov po implementácii hlavnej funkcionality softvéru. Ďalším často používaným prístupom

môže byť opakované testovanie softvéru počas celej doby jeho vývoja. Druhý prístup môže spôsobiť viac ťažkostí, keďže oprava nájdenej chyby počas vývoja môže spôsobiť nutnosť opravy zdrojového kódu v rozsahu, ktorý prekračuje rámec chyby. V testovaní sa toto preukáže ako nutnosť prehodnotiť testované oblasti, prípadne ich pretestovať, pretože mohli byť opravou kódu ovplyvnené. Ďalším problémom môže byť nutnosť väčšieho dozoru testerov nad testovanou verziou produktu. Načrtnuté problémy môžu spôsobiť meškanie projektu čo sa môže odraziť na jeho neustálom predražovaní. Ďalšou obecnou praktikou je vytváranie testovacích sád počas špecifikácie softvéru. Tieto testy sú potom využívané v regresných testoch a ich úlohou je zaistenie neopakovania sa všetkých doteraz známych chýb v softvéri. Regresné testy sú pravidelné automatizované testy pre pravidelnú kontrolu problematických častí softvéru. Všeobecne uznávaným pravidlom je, že čím skôr sa chyba v softvéri objaví, tým je jej oprava menej nákladná. Logicky to vyplýva z toho, že objavená chyba v softvéri by mohla spôsobiť niekoľko ďalších problémov počas vývojového cyklu projektu. Objavená chyba by taktiež mohla spôsobiť zmenu štruktúry či povahy dát s ktorými systém pracuje a tým by znemožnila testovanie so starými testovacími dátami.

## 2.2 Rozdelenia softvérového testovania

Pre softvérové testovanie existuje veľké množstvo rozdelení podľa rôznych kritérií. V zásade existujú tri hlavné fáze testovania:

- **Testovanie modulov** (Unit Testing) – tu sa každý modul softvéru testuje pre korektnú implementáciu podľa rozšírenej špecifikácie
- **Testovanie integrácie** (Integration Testing) – tu sa viac modulov spojených v jeden celok navzájom spája a testuje až kým nefunguje ako celok
- **Testovanie systému** (System Testing) – v tomto stupni sa program integruje do celkového produktu a testuje sa či spĺňa všetky požiadavky

Dôležitým základným rozdelením testovania (nielen softvéru) je z hľadiska ponímania problému:

- **Štrukturálne testovanie** (White–box testing) – poníma vnútornú perspektívu testovaného objektu a podľa toho vytvára testovacie prípady. Nájdenie všetkých možností (ciest) v testovanom softvéri nie je jednoduché a vyžaduje programátorské skúsenosti. Pri testovaní sa vyberajú testovacie hodnoty pre vhodné cesty v programe a testuje sa prislúchajúci výstup. Keďže test je založený na aktuálnej vnútornej štruktúre testovaného produktu pri zmene vnútornej štruktúry je nutné aktualizovať aj testovacie prípady. Toto testovanie sa vo väčšine prípadov používa v stupni testovania modulov.
- **Funkcionálne testovanie** (Black–box testing) – poníma vonkajšiu perspektívu o testovanom objekte a podľa toho sa snaží formulovať testovacie prípady. Na základe vyberania správnych a nesprávnych vstupných hodnôt a prechodom systému analyzuje správnosť výstupných údajov v konfrontácii so vstupnými. Vnútorná štruktúra testovaného celku zostáva neznáma. I keď funkcionálne testovanie môže viesť k nájdeniu nových testovacích prípadov nikdy sa nedá zaručiť, aby všetky možnosti (cesty) v softvéri boli pretestované. Testovanie tohto druhu je možné vo všetkých stupňoch testovania.

Testovanie softvéru v ďalšom možno rozdeliť na:

- **Statické testovanie** (Static Testing)
  Je formou softvérového testovania, kde samotný výsledný softvér nie je testovaný. Viac o statickom testovaní v nasledujúcej sekcii 2.3.
- **Dynamické testovanie** (Dynamic Testing)
  Sa zaoberá testovaním dynamického chovania kódu. Dynamické analýza zahŕňa prieskum odpovede systému cez premenné, ktoré sa menia v čase a nie sú konzistentné.

V ďalšom texte sa zameriam na statickú analýzu zdrojového programu rozoberiem jej hlavné smery.

## 2.3 Statické testovanie a statická analýza kódu

Na rozdiel od dynamického testovania, statické testovanie netestuje samotný softvérový produkt. Statické testovanie je formou štrukturálneho testovania a je vykonávané väčšinou vo fáze testovania modulov. Tento druh testovania je zvyčajne prvým typom testovania, ktoré sa vykonáva na každom softvérovom produkte. Metódou statického testovania softvéru je statická analýza kódu. Táto metóda je dobre automatizovateľná. Ukážkou neautomatizovateľnej metódy môže byť napríklad manuálne prezeranie zdrojového kódu softvéru.

### 2.3.1 Statická analýza kódu

Statická analýza kódu je formou statického testovania vykonávaná väčšinou automatickým nástrojom. Sofistikovanosť jednotlivých analýz kódu závisí od toho, či berú do úvahy len chovanie jednotlivých príkazov a deklarácií alebo celého zdrojového kódu. Informácie o výsledkoch analýzy sa môžu zobrazovať ako časti programu s chybovou alebo varovnou hláškou. Výstupy analýzy sa môžu poprípade použiť ako vstup pre formálne metódy, ktoré sa na základe matematických pravidiel pokúsia dokázať správnosť testovaného softvéru – tj. chovanie programu odpovedá špecifikácii. Statická analýza kódu je používaná v počítačových systémoch s kritickou bezpečnosťou pre lokalizáciu potencionálne nebezpečného kódu.

### 2.3.2 Formálne metódy statickej analýzy kódu

Tieto metódy používajú pri overovaní správneho výstupu testovaného softvéru matematické metódy. Techniky formálnej statickej analýzy môžme rozdeliť do nasledujúcich skupín:

- **Testovanie modelu** (Model checking)
  Uvažuje o systémoch s konečným počtom stavov, ktoré je možné zjednodušiť použitím abstrakcie. Toto testovanie sa využíva vo väčšej miere na verifikáciu formálnych systémov. Model týchto systémov je odvodený z hardvérového alebo softvérového návrhu.
- **Abstraktná interpretácia** (Abstract interpretation)
  Je založená na princípe čiastočného spustenia kódu, pri ktorom sa získa dostatok potrebných informácií. Využíva sa napríklad pri vyhľadávaní optimalizácií a transformácii kódu pri analýze kódu v prekladačoch.

- **Použitie predikátov**
  Pri tejto technike sa využíva používanie predikátov v kóde (assertov) ako podklad pre Hoareovu logiku. Táto metóda sa využíva v niektorých programovacích jazykoch ako JML alebo SPARK.

### 2.3.3 Problémy statickej analýzy zdrojového kódu

Statická analýza zdrojového kódu sa zaoberá viacerými problémami v zdrojovom texte. Samozrejme nie všetky problémy sú detekovateľné jedným nástrojom. Týmito problémami v zdrojovom kóde programu môžu byť:

- **Nedosiahnuteľný kód** (Unreachable code) – niekedy nazývaný aj mŕtvy kód (dead code) je existujúci kód v zdrojovom texte programu, ktorý nikdy nebude vykonaný. Mŕtvy kód je v programe nežiadúci, pretože môže byť dôsledkom chyby v softvéri. Ďalším nežiadúcim faktorom môže byť že jeho nadbytočnosť je neznáma a udržovanie takéhoto kódu je zbytočné. Detekcia mŕtveho kódu je formou statickej analýzy. Vyžaduje analýzu riadiaceho toku testovaného programu.
- **Nedeklarované premenné** (Undeclared variables) – premenné ktoré sa majú použiť neboli deklarované v kóde.
- **Neinicializované premenné** (Uninitialized variables) – premenné ktoré sa majú použiť boli deklarované, ale ich hodnota nebola definovaná v predchádzajúcom behu programu.
- **Nesprávne typy parametrov** (Parameter type mismatches) – pri predávaní parametrov nerovnakých typov môže dochádzať k nepríjemným a ťažko odhaliteľným chybám.
- **Nevolané funkcie a procedúry** (Uncalled functions and procedures) – procedúry v zdrojovom kóde, ktoré sa nevolajú nikde v programe.
- **Pretečenie zásobníka** (Buffer overflow) – chyby vyvolaná prístupom do oblasti za alokovanou pamäťou alebo za rozsah poľa.
- **Použitie pamäti mimo rozsahu** (Out of scope memory usage) – vzniká napríklad pri vrátení adresy premennej deklarovanej vo vnútri funkcii.
- **Nesprávne použitie ukazovateľov** (Misuse of pointers) – nastáva v prípade že ukazovateľ ukazuje na iné data ako si myslí.
- **Použitie dealokovanej pamäti** (Use of deallocated memory) – táto chyba vzniká pri použití pamäte, ktorá už bola uvoľnená z používania.
- **Podtečenie zásobníka** (Buffer underflow) – je chyba vyvolaná prístupom do nealokovanej pamäte pred alokovaným blokom.

### 2.3.4 Techniky statickej analýzy zdrojového kódu

Analýza zdrojového kódu je súčasťou každého prekladača či interpreta zdrojového kódu. Táto analýza však často neodhalí často fatálne chyby v zdrojovom kóde (nie je to samozrejme jej prvoradou úlohou). Túto analýzu preto nástroje určené pre statickú analýzu kódu rozširujú, aby dosiahli väčšieho rozsahu testovania nebezpečných či nekorektných konštrukcií. V ďalšom zhrniem niektoré často používané techniky používané pre statickú analýzu kódu.

**Analýza ukazovateľov** je technikou pri ktorej sa zisťuje "pôvod" dát odkazovaných ukazateľmi a referenciami v programe. Táto analýza je dosť náročná na hardvér a preto sa zaviedli zjednodušenia, ktoré umožňujú odbúrať toto obmedzenie. Zjednodušením môže byť napríklad uvažovanie o všetkých ukazateľoch na štruktúrovaný objekt ako ekvivalent.

**Analýza haldy** je technikou statickej analýzy kódu pri ktorej sa kontrolujú vlastnosti dynamických a ukazateľmi previazaných dát. Táto technika je používaná pri čase kompilácie u interpretovaných jazykoch. Dokáže odhaliť miesta v programe s nesprávne uvoľnenou pamäťou, s pamäťou ktorá sa uvoľňuje viac krát či s ukazateľom na odalokovanú pamäť.

### 2.3.5 Voľne dostupné nástroje statickej analýzy zdrojového kódu

V tejto sekcii sa pokúsim poukázať na voľne dostupné nástroje pre statickú analýzu kódu, ktoré dopomohli k odhaleniu chýb vo viacerých významných softvéroch.

**Splint** je potomkom najznámejšieho nástroja pre statickú analýzu zdrojového kódu písaného v jazyku C (LClint). Okrem vlastností LClint–u umožňuje detekovať pretečenie zásobníku a ďalšie bezpečnostné riziká. Pre analýzu používa niekoľko zjednodušených techník statickej analýzy. I bez použitia anotácií v kóde dokáže monitorovať vytváranie a prístup k zásobníkom a kontrolovať prekročenie ich medzí. Pre modelovanie toku riadenia a obecných konštrukcií cyklov používa heuristiku. Splint produkuje veľké množstvo varovných hlásení, z ktorých menšie percento býva reálnou hrozbou.

Nástroj **UNO** je pomenovaný podľa troch hlavných chýb v softvéri, ktoré má detekovať. Týmito chybami sú použitie neinicializovaných premenných, odkazovanie sa prázdnym (NULL) ukazovateľom a prekročenie hranice poľa pri jeho indexovaní. UNO využíva pri analýze voľne dostupné rozšírenie prekladača ctree pre generovnie parsovacieho stromu pre jednotlivé procedúry a funkcie programu. Vzniknuté parsovacie stromy sú transformované na grafy riadiacich tokov. Tieto sú ďalej použité ako vstup pre testovanie modelov pre nájdenie prípadných chýb v indexovaní.

**ARCHER** je nástroj pre statickú analýzu kódu používaný pri vývoji kernelu pre Linux. Objavil veľké množstvo chýb spôsobených neoprávneným prístupom do pamäte. Nástroj používa vnútro–procedurálnu analýzu zospodu–nahor (bottom–up). Po naparsovaní zdrojového kódu do abstraktných stromov je vytvorený približný strom volaní a podľa neho je určené poradie prezerania funkcií. Prezeranie začína na spodku grafu a jeho úlohou je zhromažďovanie podmienok pre zistenie rozsahov funkcií, ktoré spôsobujú neoprávnený prístup do pamäte. Nástroj je limitovaný neschopnosťou modelovania ukazateľov na funkcie.

## 2.4 Virtualizácia a testovanie

Pod pojmom virtualizácie sa v informatike technika pri ktorej je nahradený fyzikálny prostriedok softvérovou vrstvou, ktorá ho emuluje. Takýto emulovaný prostriedok je transparentne v systéme definovaný, i keď fyzicky neexistuje. Pojem virtualizácie je používaný vo veľa odlišných významoch. Táto kapitola sa zameria na virtualizáciu počítačových platform, ktorá zahŕňa simuláciu tzv. virtuálnych strojov a poukáže jej výhody v oblastí testovania softvéru.

### 2.4.1 Virtualizácia počítačových platform

Virtualizácia počítačovej platformy je realizovaná na hardvérovej platforme (tzv. hostiteľ) riadiacim softvérom, ktorý vytvára simulované prostredie (tzv. virtuálny stroj) pre hosťovaný softvér. Týmto softvérom býva často celý operačný systém bežiaci tak isto, ako by bol nainštalovaný na samostatnej hardvérovej platforme. Ako hosťovaný softvér sa v ďalšom texte bude myslieť výhradne operačný systém. Typicky je viac virtuálnych strojov simulovaných na jednom fyzickom stroji. Pre správnu funkcionalitu hosťovaného softvéru, musí byť poskytnutá simulácia dostatočne robustná, aby poskytla podporu pre všetky externé rozhrania využívané softvérom. Existuje viac prístupov k virtualizácii počítačových platform:

- **Emulácia/Simulácia**
  Pri tomto prístupe virtuálny stroj emuluje celý hardvér. To umožňuje hosťovanému operačnému systému bežať na úplne odlišnej hardvérovej architektúre ako pre ktorú bol navrhnutý. Toto riešenie virtualizácie bolo dlhú dobu používané pre vývoj softvéru využívajúcich nový typ procesorov, ktoré ešte neboli fyzicky dostupné. Spôsob implementácie samotnej emulácie sa značne líši – od jednoduchých stavových automatov až po použitie dynamickej rekompilácii. Nevýhodou tohoto typu prístupu je rýchlosť behu softvéru na emulovanom stroji. Tá je značne pomalšia ako beh na neemulovanom stroji. Príklady virtualizačných nástrojov využívajúcich tento prístup môže byť napríklad Bochs, PearPC alebo QEMU bez akcelerácie.

- **Prirodzená virtualizácia** (Native virtualization)
  Tento prístup umožňuje plnohodnotný beh virtuálneho stroja v izolácii bez emulácie samotného procesoru. Virtuálny stroj emuluje kompletne celý hardvér aby umožnil beh nemodifikovaného operačného systému na rovnakom type procesoru ako je hostiteľ. Prirodzená virtualizácia umožňuje využitie hardvérovej podpory pre virtualizáciu akými sú napríklad AMD–V alebo Intel VT. Toto umožňuje beh virtuálnych strojov na takmer rovnakej rýchlosti ako keby bežali na reálnom hardvéri. Výhodou prirodzenej virtualizácie je minimalizovanie operácii potrebných pre údržbu systému a minimalizácia zmien, ktoré sú nutné pre virtualizovaný operačný systém. Nevýhodou sa môže javiť potreba podpory pre jadro operačného systému hostiteľa. Tento spôsob virtualizácie patrí v dnešnej dobe za najrozšírenejší. Príkladom tohto druhu virtualizácie môže byť VMware Server, Virtual PC, Virtual Iron alebo Virtual Box.

- **Čiastočná virtualizácia** (Partial virtualization)
  Virtuálny stroj v tomto prístupe simuluje niektoré (ale nie všetky) hardvérové prostriedky – jedná sa hlavne o virtualizáciu adresového priestoru. Takto vytvorené

prostredie umožňuje síce izoláciu prostriedkov a procesov, ale neumožňujú beh inštancií samostatných operačných systémov. Tento prístup zohral významnú úlohu pri vývoji virtualizácie.

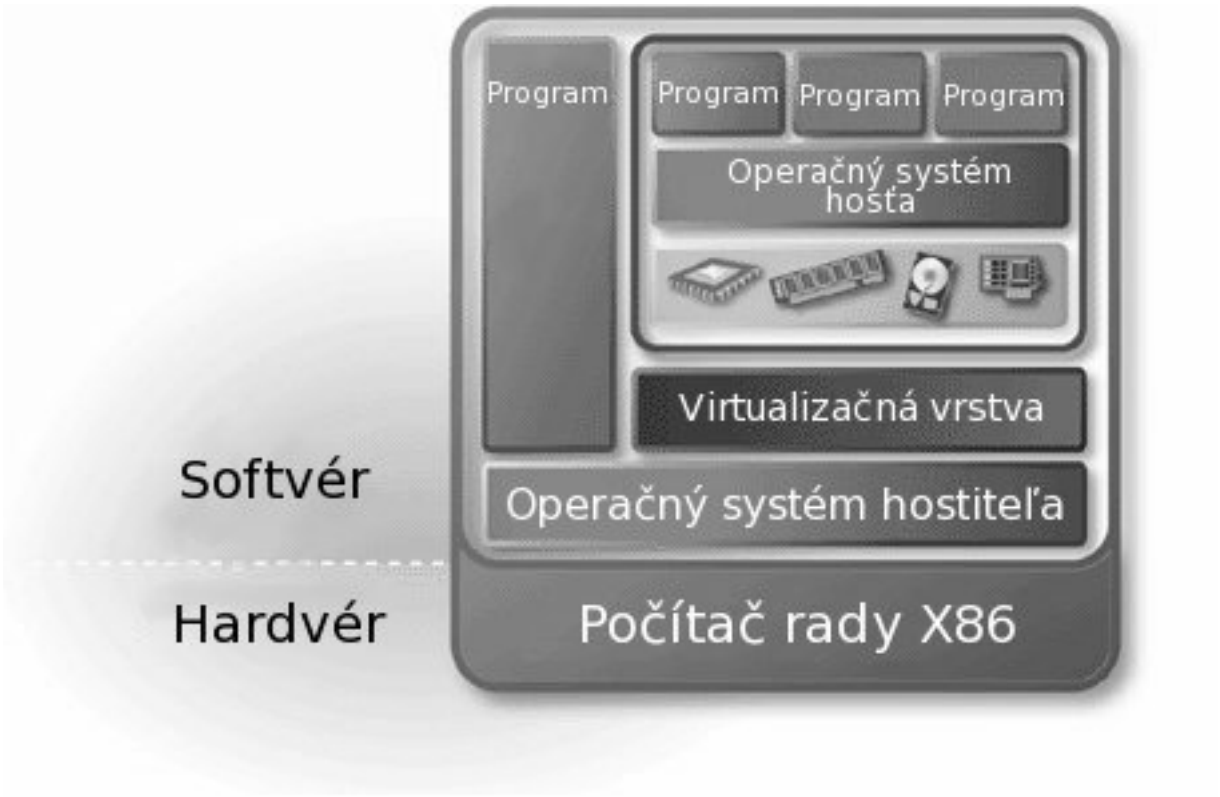
- **Paravirtualizácia** (Paravirtualization)
  Táto technika virtualizácie poskytuje virtuálnym strojom rozhranie podobné, avšak nie úplne identické hardvéru, na ktorom paravirtualizér beží. Toto vyžaduje portovanie operačného systému na platformu paravirtualizéru nazývanú VMM (virtual machine motor – hypervisor). Príkladmi paravirtualizácie sú systémy ako Xen či L4.

- **Virtualizácia na úrovni operačného systému** (Operating system–level virtualization)
  Pod týmto pojmom sa myslí virtualizácia serverov na úrovni jadra operačného systému. Ide vlastne o rozdelenie jedného fyzického jadra na viac samostatných jednotiek transparentných z pohľadu užívateľa. Virtualizované operačné systémy musia byť rovnaké ako hostiteľský, i keď konfiguráciu môžu mať úplne odlišnú. Príkladom virtualizačných nástrojov založených na tomto prístupe môže byť Linux–VServer, Virtuozzo, OpenVZ, Solaris Containers či FreeBSD Jails.

- **Virtualizácia aplikácií** (Virtualization of aplications)
  Je pojem ktorý vyjadruje aplikácie bežiace všetky vo vlastnom, izolovanom, virtuálnom prostredí. Toto virtuálne prostredie zastrešuje len komponenty potrebné pre izolovaný beh ako napríklad súbory, premenné prostredia, globálne objekty a podobne.

### 2.4.2 Úloha virtualizácie v testovaní softvéru

Pri testovaní softvéru je snaha otestovať vyvíjaný softvér na čo najväčšom množstve podporovaných platform. Ak sú komponenty softvéru závislé na konfigurácii týchto platform, stáva sa dokonalé otestovanie softvéru z časového hľadiska veľmi náročné a neefektívne. Príkladom komponenty softvéru závislej od konfigurácie cieľovej platformy môže byť napríklad ovládač pre vyvíjané zariadenie. Ten z hľadiska softvéru predstavuje modul do jadra operačného systému. Jadrá operačných systémoch sa v reálnom nasadení často značne líšia a otestovanie správnej funkcionality vyvíjaného softvéru na viacerých platformách (s rôznymi verziami jadra) nie je jednoducho riešiteľná. Snahou je používať čo najmenší počet finančných prostriedkov a výpočetného výkonu pre testovanie. Rozmýšľať nad neustálou reinštaláciou systému na práve testovaný systém samozrejme nemá zmysel. Takýto úkon je časovo veľmi náročný a navyše neautomatizovateľný. Používanie viacerých počítačov ruší snahy o minimalizácii prostriedkov. Možnosťou by mohlo byť používanie viacerých operačných systémov na jednom testovacom počítači. Vytvorenie automatizovaného testovacieho prostriedku pracujúceho v takomto prostredí by však bola značne zložitá. Dôležítým požiadavkom pri testovaní býva, aby sa testovanie začínalo na vždy čistom systéme, keďže testovací proces často vytvára množstvo údajov a modifikuje bežiaci systém. Vzniká teda potreba čistenia po testovacom procesi, čo v určitých prípadoch môže byť značne komplikované. Využitie virtualizácie pre operačné systémy poskytuje odpoveď na načrtnuté problémy. Na jednom fyzickom počítači umožňuje beh rôznych iných operačných systémov. Tieto operačné systémy bežia oddelene, každý na svojom vlastnom virtuálnom stroji. Spúšťanie a zastavovanie virtuálnych strojov je možné nezávisle na behu hostiteľského počítača.




Obrázek 2.1: Príklad prirodzenej virtualizácie. Operačný systém je spúšťaný v užívateľskom priestore ako ostatné užívateľské aplikácie. Hardvérové prostriedky sú emulované v softvéri.

Virtuálne stroje môžu bežať paralelne, jediným obmedzujúcim faktorom pri ich behu môže byť množstvo voľnej operačnej pamäte či miesta na pevnom disku. Virtuálny stroj používa pre uloženie dát virtuálny disk, ktorý nevyžaduje zvláštnu partíciu na disku pretože sa javí ako obyčajný súbor. Z hľadiska automatizovania poskytuje virtualizačný nástroj funkcie pre jednoduché spúšťanie, vypínanie či zisťovanie stavu virtuálneho stroja. Ďalšou kľúčovou vlastnosťou, z hľadiska testovania, je návrat k predchádzajúcemu stavu virtuálneho stroja. Táto vlastnosť virtualizačných nástrojov plynie z možnosti tvorenia tzv. obraz disku (angl. snapshot). Pri tomto sa celý stav virtuálneho stroja uloží na disk. Pri návrate k uloženému stavu sa všetky dovtedy urobené zmeny zahodia a odpadne teda problém s čistením po testovacom procese.

Virtualizácia je ako vidieť vhodný prostriedok pre testovania softvéru. Poskytuje prostriedky pre paralelné automatizované testovanie na rôznych konfiguráciách operačného systému. Priama virtualizácia naviac umožňuje testovať i na rôznych počítačových platformách paralelne.

# Kapitola 3

# Programové vybavenie projektu Liberouter

Projekt Liberouter sa zaoberá vývojom softvéru i hardvéru. V projekte pracuje desiatok ľudí a je rozdelený na 5 pracovných skupín. Vlastná tvorba programového vybavenia je preto pomerne rôznorodá a rozdelená v rámci jednotlivých projektov.

**Hardvérová skupina** sa zaoberá vývojom hardvérových designov pre FPGA čipy, ktoré sú srdcom vyvíjaných sieťových zariadení. Hlavná časť produkovaného kódu touto skupinou tvorí VHDL kód, ktorý sa pomocou syntézy prevádza na konfiguráciu FPGA čipov.

**Softvérová skupina** sa zaoberá tvorbou nízkoúrovňového softvéru. K týmto patria hlavne ovládače pre vyvíjané zariadenie, knižnica pre ovládanie jednotlivých komponent v FPGA čipu a rôzne konfiguračné nástroje. Programy tvorené v rámci tejto skupiny sú písané takmer výhradne v jazyku C, prípadne v skriptovacom jazyku Bash, Python alebo Perl.

**Skupina testerov** vykonáva viacero činností zameraných na celkové konfrontačné testy vznikajúceho softvéru v Liberoutri oproti nutným štandardom. Okrem toho vykonáva výkonnostné testy, crash testy hardvéru, manuálne testy balíčkov a porovnávacie testy softvéru oproti voľne či komerčne dostupným riešeniami. V rámci testov používa jazyk TCL pre automatizovanie testov a generovanie dokumentácie výsledkov testov.

**Skupina formálnej verifikácie** sa zaoberá testovaním modelov hardvéru vytváraného hardvérovou skupinou. Cieľom týchto testov je dokázanie ich správnosti voči špecifikácii. Vytvára nástroje pre automatizáciu procesu prekladu VHDL kódu do jazykov verifikačných nástrojov.

**Skupina systémovej podpory** vytvára pracovné prostredie pre všetky ostatné skupiny. Projekt zastrešuje desiatky pracovných počítačov s vyvíjaným hardvérom. Pre prehľad dostupnosti a konfigurácie pracovných počítačov skupina systémovej podpory udržuje informačný systém s možnosťou rezervácie strojov. Poskytuje tiež všetkú podporu k pracovným staniciam a má vytvorené nástroje pre vzdialenú správu počítačov a synchronizáciu zdielaných dát na všetkých strojoch.

## 3.1 Liberouter CVS

Všetky zdrojové súbory vytvárané v rámci všetkých skupín sú uložené v spoločnom CVS archíve. CVS umožňuje správu nových revízií, udržiavanie histórie revízií a v neposlednom rade zdieľanie dokumentov medzi vývojármi. CVS Liberoutru je rozdelené na viac častí. Podstatné časti z hľadiska vytváraného softvéru sú

- `vhdl_design` – vetva obsahujúca zdrojové kódy VHDL vývojárov v projekte
- `sys_sw` – vetva obsahuje všetok kód produkovaný v rámci softvérovej skupiny
- `mk` – obsahuje prekladový systém
- `setup` – spustiteľný súbor pre konfiguráciu pre prekladom

Projekt podporuje dve open–source platformy, pre ktoré vyvíja svoj softvér. Týmito platformami sú NetBSD a Linux. Pre potreby správneho prekladu na oboch platformách bol vyvinutý prekladový systém. Ten sa stará o kontrolu potrebných závislostí pre preklad a spúšťanie samotného prekladu v závislosti od platformy. Okrem tohoto umožňuje priamu inštaláciu do systémových adresárov daného stroja. Konfiguračný, kompilačný a inštalačný proces je veľmi jednoducho automatizovateľný. Pri týchto procesoch vzniká množstvo záznamov, ktoré poskytujú dôkladné informácie o ich pribehu a prípadných vzniknutých problémoch.

## 3.2 Inštalačné balíčky

Inštalačné balíčky vyvíjané v rámci projektu Liberouter sú formou špecializovaných balíčkov pre kompletnú inštaláciu a konfiguráciu jednotlivých aplikácií vyvíjaných v rámci projektu. Inštalačné balíčky sú teda ďalšou formou programového vybavenia. Balíčky sú prístupné pomocou sieťového disku na všetkých pracovných počítačoch v adresári `/home/data/packages`. Uložené sú v komprimovanom formáte gzip tar pod názvom `názov-verzia.tgz`. Každý balíček vyžaduje inštalačný proces. Inštalačný proces sa skladá z konfigurácie, prekladu, inštalácie a poinštalačných krokov.

# Kapitola 4

# Návrh a implementácia testovacieho servera

Táto kapitola opisuje návrh a implementáciu testovacieho servera pre programové vybavenie projektu Liberouter. Sekcia 4.1 sa zaoberá špecifikáciou systému. Vymedzuje požiadavky kladené na server a popisuje jeho cieľové vlastnosti. V sekcii 4.2 sa pre testovanie vyberá vhodný virtualizačný nástroj a detailnejšie sa popisujú jeho vlastnosti. Sekcia 4.3 popisuje návrh servera. Podrobne opisuje jednotlivé časti servera a ich vzájomné prepojenie. Sekcia 4.4 popisuje implementačné detaily vytvoreného testovacieho servera. Predstavuje implementované programy a ukazuje príklad testovania. Podrobný opis funkcie a rozdelenia úloh testovacích modulov je v sekcii 4.5. Posledná sekcia 4.7 je zameraná na popis dokumentácie zhotovenej k testovaciemu serveru.

## 4.1 Špecifikácia servera

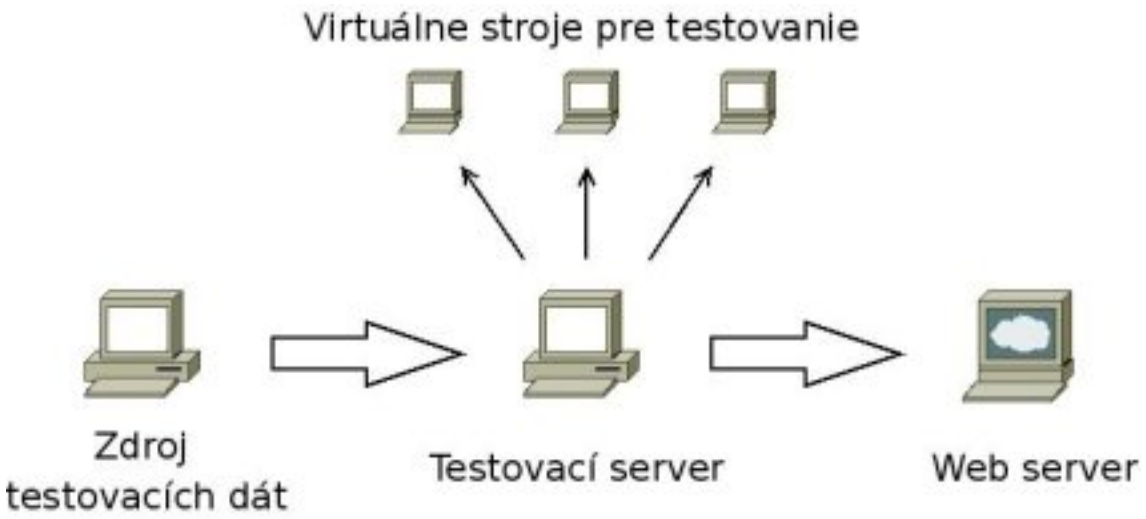
### 4.1.1 Požiadavky na server

Testovací systém má byť automatický nástroj pre revíziu aktuálneho programového vybavenia na rôznych konfiguráciách podporovaných platform. V projekte Liberouter existuje skupina skriptov pre automatickú revíziu zdrojového kódu uloženého v CVS. Toto testovanie je dôležité, keďže softvérová časť CVS umožňuje inštaláciu do systému a je nutné mať informácie o jej konzistentnom stave. Pod pojmom konzistentný stav je myslený bezchybový preklad zdrojových súborov písaných v jazyku C a ich úspešná inštalácia do systému. Používané testovacie skripty vyžadujú beh na samostatnom počítači. Počet testov s rôznou konfiguráciou je teda rovný počtom použitých počítačov. Navrhnutý server by mal pomocou virtualizácie tento nedostatok odstrániť a umožniť beh veľkého množstva testovacích virtuálnych strojov s rôznou konfiguráciou. Testovací server by mal byť ľahko rozšíriteľný vo forme testovacích skriptov napísaných v ľubovoľnom jazyku. Toto by umožnilo ľahké rozšírenie testovacích úloh v budúcnosti.

### 4.1.2 Cieľové vlastnosti servera

Testovací server má bežať na jednom vyčlenenom testovacom stroji v rámci projektu. Testovanie má vykonávať raz denne na všetkých testovacích virtuálnych strojoch. Vstupom testovania je CVS uložená na stroji `andre.liberouter.org`. Testovací server musí riešiť jej automatické sťahovanie bez nutnosti odkrytia hesla. Server musí dbať na vyťaženie

CVS servera a dodržiavať pravidlo jedného stiahnutia pre jeden testovací cyklus. Výsledok testovania by mal byť ľahko dostupný členom projektu, najlepšie formou jednoduchej internetovej prezentácie. Zmyslom testovania má byť prehľad chýb a varovných hlásení pri preklade a inštalácii do systému.




Obrázek 4.1: Obrázok znázorňuje funkciu testovacieho serveru. Vstupom do serveru sú zdrojové testovacie dáta, ktoré sa použijú pre testovaní na rôznych virtuálnych strojoch. Výsledky testovania sa budú prenášať na web server pre možnosť jednoduchého zobrazenia výsledkov.

## 4.2 Výber virtualizačného nástroja

Testovací server bude vykonávať testovania na viacerých variantách operačného systému Linux a operačnom systéme NetBSD. Keďže sa jedná o simuláciu rôznych operačných systémov, musí využívať vhodný typ virtualizácie. Pri virtualizácii celých operačných systémov na rovnakej hardvérovej platforme prichádzajú do úvahy prirodzená virtualizácia a paravirtualizácia. Paravirtualizácia sa popri tom javí ako menej vhodná alternatíva, pretože vyžaduje modifikáciu virtualizovaných operačných systémov. Pre beh virtuálnych strojov bude teda využitá prirodzená virtualizácia. V dnešnej dobe je dostupných niekoľko riešení pre tento typ virtualizácie. Riešenie by malo byť voľne dostupné a poskytovať vhodné a ľahko ovládateľné prostriedky pre správu virtuálnych strojov a komunikáciu s nimi zo strany servera. Pri tomto obmedzení sa ako veľmi vhodné riešenie ukazuje použitie voľne dostupného VMware Servera, ktorý poskytuje nástroje pre riešenie všetkých nastolených požiadavkov. Detailný popis jeho vlastností popisuje nasledujúca sekcia.

### 4.2.1 VMware Server

VMware Server je voľne dostupný freeware virtualizačný nástroj pre architektúru X86. Pri vojom behu využíva prirodzenú virtualizáciu (viď 2.4.1). VMware Server sa snaží vo všetkých možných prípadoch spustiť vykonávaný kód virtuálneho stroja priamo na procesore hostiteľa. Toto je možné napríklad, ak sa vykonáva kód v užívateľskom móde alebo

v móde virtual 8086. V prípadoch, keď toto nie je možné, používa techniku dynamickej rekompilácie. Dynamická rekompilácia je technika, pri ktorej je pri prvom vykonávaní kód dynamicky preložený do vykonateľných blokov a pri druhom spustení sa používa už preložený kód. Nutnosť dynamickej rekompilácie nastáva napríklad pri vykonávaní kódu jadra, teda kódu bežiaceho v privilegovanom režime. Touto pokročilou technikou dosahuje pri behu virtualizovaného operačného systému rýchlosti porovnateľnej s behom na samostatnom nevirtualizovanom stroji. Spomalenie sa pohybuje v rozmedzí 6 až 20%, a je nižšie pri dátovo intenzívnejších operáciách.

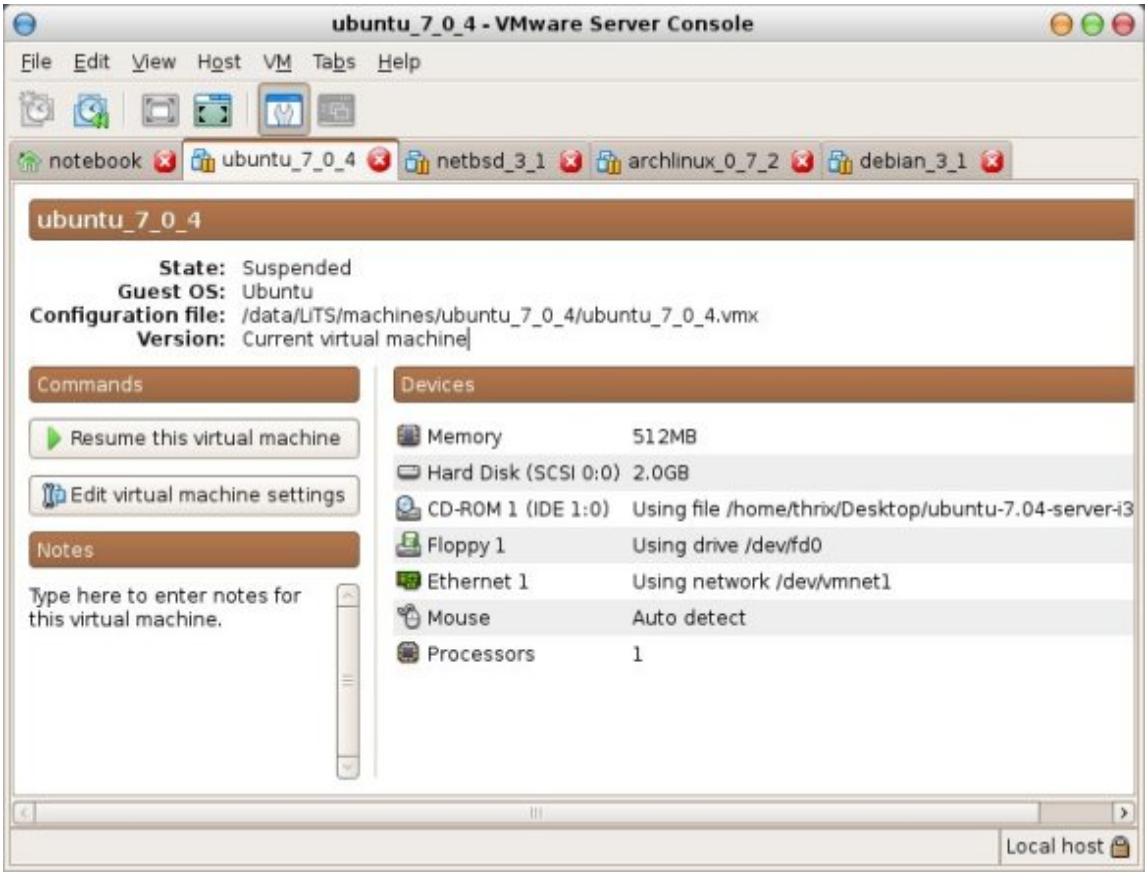
VMware Server umožňuje pomerne jednoducho vytvárať, meniť stav a vlastnosti virtuálnych strojov prostredníctvom intuitívneho GUI. Každý virtuálny stroj môže mať nainštalovaný svoj vlastný operačný systém, ten však musí byť schopný bežať na architektúre X86. VMware Server umožňuje mostenie (angl. bridging) na existujúci hardvér ako sieťová karta, zariadenie CD/DVD–ROM, pevné disky a USB zaradenia. Mostenie znamená že zariadenie môže byť využívané súčasne hostiteľom i hosťovaným systémom. Okrem možnosti mostenia umožňuje aj plnú simuláciu niektorých druhov hardvéru. Toto umožňuje napríklad použitie ISO ako CD/DVD–ROM či špeciálneho súboru s príponou .vmdk ako pevný disk. Ďalším príkladom môže byť možnosť konfigurácie sieťového zariadenia cez NAT, čo si nevynúti ďalšiu pridelenú IP adresu ako pri použitý bridge. Dôležitou, i keď obmedzenou vlastnosťou VMware Servera, je možnosť vytvárať jedného obrazu disku. Toto umožňuje vracať sa k predchádzajúcej konfigurácii virtuálneho stroja so stratou všetkých zmien, ktoré boli dovtedy na virtuálnom stroji vykonané. Snímok disku je možné uzamknúť, čo zabráni jeho prepísaniu. Tiež je možné nastaviť automatický návrat k uloženému obrazu disku po vypnutí virtuálneho stroja. Po jeho zapnutí neprebieha teda štart operačného systému (angl. boot), ale aktuálny stav je navrátený z obrazu disku.

Jednoduché ovládanie ovládanie virtuálnych strojov nie je možné len s použitím GUI. Pretože GUI nie je vhodné pre automatizovanie operácií súvisiacich s virtuálnymi strojmi, poskytuje VMware Server ďalšie dva nástroje pre prácu s nimi. Prvou je ovládanie všetkých operácií pomocou programu s bohatým repertoárom príkazov. Tento program má názov `vmware--cmd` a poskytuje rozhranie k všetkým dostupným funkciám cez GUI. Ďalšou možnosťou ovládania virtuálneho stroja môže byť použitím rozhrania VMware Perl API. Jedná sa o modul do jazyka Perl. Príkaz `vmware--cmd` je povinnou súčasťou inštalácie VMware Servera a zastrešuje všetky operácie poskytované skriptovacím API. Poskytuje k nim jednoduché rozhranie a preto je vhodné pre použitie v skriptoch, ktoré umožňujú ľahkú prácu s príkazovou riadkov (Bash, Perl, Python atď.).

Každý virtuálny stroj, ktorý má bežať pod VMware musí byť pred prvým spustením na danom virtuálnom stroji zaregistrovaný. Tento proces je nevyhnutný, bez neho nie je možné virtuálny stroj spustiť. Použitím príkazu `vmware--cmd` je možné virtuálny stroj jednoducho zaregistrovať. Ako parameter programu stačí predať `.vmx` súbor virtuálneho stroja.

## 4.3 Návrh servera

Pri návrhu testovacieho servera je dôležité dbať na fakt, že bude riadiť beh viacerých virtuálnych strojov. V týchto virtuálnych strojoch bude viacero verzií rôznych distribúcií operačných systémov Linux a NetBSD. S týmito operačnými systémami bude server pracovať ako so samostatnými jednotkami. Bude musieť tieto stroje spúšťať, vypínať, čakať




Obrázek 4.2: VMware Server Console – konzola pre ovládanie stavu a konfigurácie virtuálnych strojov. V konzoli sú otvorené 4 virtuálne stroje.

na ich zotavenie po týchto úkonoch. Okrem riadenia stavu strojov je potrebné na virtuálne stroje prenášať údaje a samozrejme nejaké údaje z nich čerpať. Implementačné riešenie týchto dvoch problémov opisujú sekcie 4.4.5 a 4.4.7.

Každý testovací proces má svoje vstupy i výstupy. Vstupy testovania budú musieť byť pred začiatkom testovania umiestnené na testovacom serveri, aby ich bolo možné jednoducho premiestniť na testovacie virtuálne stroje. Vstupmi testovania budú napríklad CVS alebo jej časti či inštalačné balíčky (viď kapitola 3). Tieto však obecne môžu byť umiestnené na odlišných miestach. Testovací server bude musieť teda pred začiatkom testovania uskutočniť zber aktuálnych testovaných dát. Výstupy testovania budú produkované na jednotlivých virtuálnych testovacích strojoch. Ako už bolo spomínané bude sa jednať o záznamy z prekladu a inštalácie CVS alebo balíčkov a výsledky statickej analýzy kódu. Medzi testovanými virtuálnymi strojmi a testovacím serverom sa vynára teda požiadavok o obojsmerný prenos údajov. Výstupné údaje budú spracované a výsledky exportované v html formáte na web server. Modularita serveru umožní však túto časť prispôsobiť na ľubovoľnú funkciu, napr. na rozosielanie varovných emailov.

Prenos informácií medzi testovacím serverom a virtuálnymi strojmi bude realizovateľný pomocou TCP spojenia, keďže virtuálny stroj obsahuje emulované sieťové zariadenie. Ako najvýhodnejší a najbezpečnejší spôsob pripojenia virtuálnych strojov sa javí použitím NAT,

ktoré umožní zdieľanie internetového spojenia pre virtuálne stroje, ale tie nebudú prístupné na sieti hostiteľa. Internetové pripojenie je potrebné najmä pri jeho inštalácii nových testovacích virtuálnych strojov. Pripojenie cez bridge je nevýhodné, pretože by počítač mohol byť vystavený zbytočným útokom a jeho prítomnosť na sieti hostiteľa je zbytočná. Posledným spôsobom pripojenia je sieť prístupná len v rámci hostiteľa a virtuálnych strojov (angl. host–only networking). Toto pre potreby testovania presne postačuje. Dôležité samozrejme je aby sa všetky virtuálne stroje nachádzali na jednej podsieti spolu s testovacím serverom. Testovací server musí poskytovať prostriedky pre ľahké rozširovanie funkcionality testovania prebiehajúceho na jednotlivých virtuálnych strojoch. Toto rozširovanie má byť nezávislé na testovacom serveri. Pridanie nového tzv. testovacieho modulu, teda nemá vyžadovať nutnosť zásahu do samotného testovacieho servera. Testovací modul sa bude skladať z niekoľkých častí, ktoré budú na sebe závisieť. Bližšie sa závislostiam v testovacích moduloch venuje sekcia 4.5.1.

Testovanie bude prebiehať cyklicky raz denne. Testovania teda musí byť spúšťané v určitú stanovenú dobu. Teoretický počet virtuálnych strojov spustiteľných paralelne je síce neobmedzený, je však dôležité si uvedomiť, že testovanie prebieha na jednom fyzickom stroji a jeho prostriedky sú zdielané. Hlavnými kritériami behu viacerých virtuálnych strojov sú dostatok pamäťových prostriedkov, priestoru na disku pre virtuálne disky virtuálnych strojov a v neposlednom rade rýchlosť procesoru. Server bude schopný riadiť množstvo paralelne bežiacich virtuálnych strojov. Tieto dve informácie budú pre server uloženého v jeho konfigurácii. Popis implementácie konfiguračného rozhrania je v sekcii 4.4.1.

Testované operačné systémy sa musia do virtuálnych strojov nainštalovať manuálne. Ich konfigurácia bude musieť spĺňať požiadavky testovacích modulov, ktoré sa budú na nich vykonávať. Požiadavky na testovacie moduly budu zverejnené v dokumentácii servera. Okrem tohoto musí byť ich sieťové rozhranie aktívne a nakonfigurované podľa konfigurácie testovaného stroja v serveri. Bližší opis konfigurácie poskytuje nasledujúci odstavec a jej implementačné detaily sú v sekcii 4.4.2.

Automatizovaná správa jednotlivých testovacích strojov si bude vyžadovať konfiguračné nastavenia v rámci testovacieho servera. Tieto údaje budú nevyhnutné pre korektný beh testovacieho procesu. Tieto nastavenia budú ukladané v ľudsky čitateľnej ale počítačom ľahko spracovateľnej podobe. K nevyhnutným informáciám pre každý testovací stroj budú tieto informácie:

- **Unikátna identifikácia testovacieho stroja**
  Každá testovaný virtuálny stroj bude musieť byť jednoznačne identifikovateľný. Unikátnym prvkom môže byť číslo, ale ako vhodnejšie riešenie sa javí použitie reťazca z číslom. Ten môže slúžiť ako plnohodnotný unikátny identifikátor, ktorý je ľahko zapamätateľný.

- **Konfigurácia komunikačného prostriedku**
  Keďže komunikácia bude prebiehať pomocou TCP spojenia, bude potrebné uchovávať adresu testovaného stroja na počítačovej sieti. Tá bude musieť byť unikátna, čo je nevyhnutné pre jej správnu funkcionalitu na sieti. Adresa bude musieť byť pre správnu funkcionalitu servera zhodná s adresou nastavenou vo virtuálnom stroji. Okrem informácie o adrese bude treba uchovávať ešte autentizačné údaje. Nutné prep pripojenie sa na server. Pre viac informácií viď 4.4.7.

- **Špecifikácia modulov testovania**
  Časti testovacích modulov použité pre testovanie sú voliteľné. Budú teda musieť byť špecifikované v konfigurácii jednotlivých virtuálnych strojov. Časti testovacích modulov môžu od seba závisieť. Tieto závislosti musia zabezpečiť zakázanie vykonávania ďalšej časti testovacieho modulu ak predchádza časť zlyhala. Časti testovacích modulov budú samostatné jednotky. Súbor závislostí určí teda testovacie moduly zložené z rôznych týchto samostatných jednotiek.

### 4.3.1 Návrh procesu testovania

Táto sekcia popisuje podrobne proces testovacieho cyklu. Testovací cyklus sa bude spúšťať podľa nastavení v konfiguračnom súbore servera (viď 4.4.1) raz denne. Pred spustením testovania bude potrebné mať testovací server správne nakonfigurovaný. K testovaniu taktiež budú musieť byť priradené správne nakonfigurované testovacie virtuálne stroje. Proces testovania bude prebiehať v týchto siedmich fázach:

1. **Zber a príprava testovacích údajov**
   Pred spustením testovaných virtuálnych strojov bude treba najprv na testovací server preniesť údaje, ktoré budú predmetom testovania. Tento úkon je potrebné vykonať iba raz. Na každý virtuálny stroj sa tieto údaju potom už prenesú z lokálneho disku a odpadne zbytočné zaťažovanie samotného zdroja testovaných dát. Okrem samotného prenosu dát môže testovací proces vyžadovať úpravu testovaných dát, poprípade ich archiváciu do jedného súboru. Jej použitie je vhodné hlavne ak sa testovacie dáta skladajú z veľmi veľa súborov. Prenos veľkého množstva malých súborov nie je vhodný, keďže vyžaduje väčšiu réžiu.

2. **Spustenie virtuálnych strojov**
   Po úspešnej príprave všetkých testovacích údajov sa spustia testované virtuálne stroje. Keďže testovaných strojov môže byť mnoho, nebudú sa spúšťať všetky naraz, ale len určitý maximálny počet. Po spustení tohto počtu prebehnú nasledujúce štyri fáze procesu testovania s virtuálnymi strojmi. Po ich ukončení sa bude pokračovať v tejto fáze so zvyšnými testovacími strojmi až kým neostane ani jeden. Spustenie testovacieho stroja nie je okamžité a závisí od nastavenia virtuálneho stroja. Pri vypnutom virtuálnom stroji, štart virtuálneho stroja pozostáva z normálneho procesu štartovania (angl. boot–up proces). Ak virtuálny stroj má vytvorený obraz disku je možné jeho z neho stav nahrať (angl. resume). Tento proces je kratší ako tradičný štart stroja a preto bude využitý v navrhnutom testovacom serveri.

3. **Prenos údajov na virtuálne stroje**
   Údaje pripravené prvou fázou je potrebné preniesť na testovací stroj. Väčšinou sa všetky dostupné testovacie moduly budú spúšťať na všetkých testovaných virtuálnych strojov. Testovacie data budú umiestnené preto spoločne a všetky sa budú prenášať na testované stroje. Prenos údajov bude výhradne v réžii testovacieho servera a nie jeho testovacích modulov. Okrem testovacích údajov je dôležité presunúť na virtuálny stroj aj časti testovacích modulov, ktoré súvisia s nasledujúcim bodom.

4. **Príprava a spustenie testovania**
   Následne po prenose všetkých údajov na testovací stroj sa vykonajú potrebné úkony

pred testovaním. Toto zahŕňa hlavne prípravu testovacích dát. Príprava testovania je v moduloch oddelená časť od samotnej testovacej časti, keďže viac testovacích častí môže využívať rovnaké dáta pre testovanie. Po príprave sa spustia testy nad pripravenými dátami. Pred spustením časti testu je potrebné ešte skontrolovať, či boli splnené podmienky Výsledky testov sa budú ukladať na jednotné miesto v rámci virtuálneho stroja.

5. **Prenos výsledkov testovania na testovací server**
   Prenos výsledkov bude úlohou testovacieho serveru a nie testovacích modulov. Testovací server musí pred prenosom testu vedieť, kde sú výsledky testov uložené. Kvôli tomuto musia byť výsledky testovania z rôznych testovacích modulov uložené na jednotnom mieste. Prenesené výsledky sa v testovacom serveri uložia pre každý testovaný systém zvlášť a podľa dátumu.

6. **Vypnutie virtuálnych strojov**
   Po prenose testovacích záznamov je testovanie na virtuálnom stroji dokončené a je možné virtuálny stroj vypnúť. Je vhodné si vypínanie testovacieho stroja nastaviť na automatický návrat k uloženému obrazu. Pri tomto sa vypnutí stroj javí ako pozastavený a jeho znovu spustenie trvá krátku dobu.

7. **Spracovanie výsledkov testovania**
   Všetky výsledky testovania zo všetkých strojov sa spracujú naraz pri konci testovania. Spracovaním výsledkov sa myslí prechod všetkých testovacích záznamov a dolovanie dôležitých údajov z nich. Z týchto údajov sa potom vytvorí prezentácia, ktorá sa umiestni na internet.

### 4.3.2 Návrh testovacích modulov

Testovacie moduly sa v navrhovanom testovacom serveri budú skladať z maximálne štyroch samostatných častí. Prvá a posledná časť modulov sa bude vykonávať na testovacom serveri. Ich úlohou bude zber a príprava testovacích dát na začiatku testovania a ich spracovanie na konci. Zvyšné dve časti testovacieho modulu budú určené pre beh na virtuálnom stroji. Ich úlohou bude príprava a spustenie testovania. Jednotlivé časti testovacích modulov budú môcť byť ľubovoľne previazané. Toto znamená, že niektoré časti pri rôznych testovacích moduloch môžu byť rovnaké. Medzi časťami testovacích modulov vznikne nutnosť riešenia závislostí. Po zlyhaní niektorej časti testovania je spúšťanie nasledujúcej zbytočné, keďže tá vyžaduje pre svoj správny beh úspešné dokončenie predchádzajúcej. Časti testovacích modulov spúšťané na testovacom serveri budú špecifikované v konfigurácii servera a časti spúšťané na jednotlivých virtuálnych strojoch v konfigurácii pre testovacie stroje. Podľa konfigurácii závislostí testovací server určí spustenie alebo nespustenie nasledujúcej testovacej časti. Podrobnejší popis implementácie testovacích modulov je v sekcii 4.5.

## 4.4 Implementácia servera

Testovací server bude implementovaný na operačnom systéme Linux. Pre jeho implementáciu som si vybral skriptovací jazyk Perl. Perl je stabilný, multi–platformný programovací jazyk. Poskytuje prostriedky pre rýchly vývoj komplexných programov vďaka širokej škále

dostupných modulov. Umožňuje ľahkú prácu s textom a značkovacími jazykmi (napr. HTML, XML).

Virtuálne stroje budú hosťom pre operačné systémy Linux a NetBSD. Časti modulov preto budú musieť byť naprogramované pre korektný beh na oboch týchto operačných systémoch. Testovacie moduly sú podľa návrhu 4.3.2 implementované ako skupiny skriptov. Jednotlivé časti modulov tvoria skripty písané v ľubovoľnom skriptovacom jazyku spustiteľnom na testovaných strojoch. Bližšie sa testovacím modulom venuje sekcia 4.5.

Všetky časti testovacieho servera, okrem súborov uvedených v konfigurácii – viď sekciu 4.4.1, sú povinne umiestnené v jednom adresári. Štruktúra súčastí implementovaného testovacieho servera popisuje nasledujúci zoznam:

- `logs/` – obsahuje všetky záznamy vznikajúce počas behu servera, detailnejší popis v sekcii 4.4.8
- `scripts/` – obsahuje podadresáre `startup/` a `shutdown/` so skriptami spúšťanými na začiatku a konci testovania, viď sekciu 4.5
- `vmdata/` – adresár s testovacími údajmi prenášanými na virtuálne stroje
- `vmscripts/` – adresár so skriptami pre testovanie, spúšťané na virtuálnych strojoch, detaily v sekcii 4.5
- `machines.conf` – obsahuje všetky potrebné údaje pre jednotlivé testovacie stroje nutné pre správny priebeh testu, detailne sa mu venuje sekcia 4.4.2
- `scriptdeps.conf` – obsahuje závislosti medzi skriptami v testovacích moduloch a je detailne popísaný v sekcii 4.5.1
- `litserver` – testovací démon riadiaci testovací proces, bližšie popísaný v sekcii 4.4.4
- `litstool` – program pre administráciu testovania a inštalácie servera, detaily implementácie sa nachádzajú v sekcii 4.4.3
- `lits.conf` – konfiguračný súbor testovacieho servera obsahuje konfiguračné údaje servera, detailne sa mu venuje sekcia 4.4.1

### 4.4.1 Konfigurácie servera

Konfigurácia testovacieho servera sa nachádza v povinnom súbore `lits.conf`. Súbor je ukladaný vo formáte XML. XML je rozšíriteľný značkovací jazyk, ktorý je v súboroch ukladaný v čitateľnej textovej forme. Tento formát je s použitím modulu XML::Simple jazyka Perl ľahko spracovateľný v skriptoch. Konfigurácia musí obsahovať nasledujúce elementy v koreňovom elemente `<config>`:

- Element `<vmwarecmd>`
  Obsahuje umiestnenie programu programu `vmware--cmd`, ktorý sa používa pre riadenie stavu virtuálnych strojov. Detaily riadenia stavu virtuálnych strojov popisuje sekcia 4.4.5.

- Element `<testtime>`
  Udáva čas začiatku testovania vo formáte `hodiny:minúty`. Testovanie sa automaticky spustí v tomto čase.

- Element `<testvms>`
  Tento element udáva maximálny počet paralelne bežiacich testovacích virtuálnych strojov. Tento údaj závisí od hardvérovej konfigurácii testovacieho servera.

- Elementy `<startup>` a `<shutdown>`
  Obsahujú zoznamy názvov skriptov oddelených medzerami, ktoré sa majú vykonávať pri zbere testovacných dát a spracovaní výsledkov testovania. Bližšie o týchto skriptoch pojednáva sekcia 4.5.

### 4.4.2 Konfigurácia virtuálnych strojov

Konfigurácia virtuálnych strojov je uložená v súbore `machines.conf`. Pre jeho uloženia je použitý formát XML. Všetky virtuálne stroje, použité pre testovanie musia mať vytvorený záznam v tomto súbore. Skript pre administráciu (viď 4.4.3) servera umožňuje jednoduché zobrazenie a zmenu hodnôt podľa názvov elementov. Koreňovým elementom v konfigurácii musí byť element `<testgroup>`. Každý virtuálny stroj musí byť definovaný v elemente `<testmachine>` s nasledujúcimi elementami:

- Element `<name>`
  Tento element obsahuje unikátny názov virtuálneho stroja. Tento sa používa pri identifikácii virtuálneho stroja pri administrácii.

- Element `vmxfile`
  Uchováva cestu a názov k súboru virtuálneho stroja. Ten je nevyhnutný pre možnosť riadenia virtuálneho stroja popísaného podrobne v sekcii 4.4.5. Virtuálny stroj môže byť teda umiestnený na ľubovoľnom mieste v adresárovom priestore servera.

- Element `<address>`
  Obsahuje IP adresu virtuálneho stroja na virtuálnej podsieti. Táto adresa sa používa pre komunikáciu s virtuálnym strojom.

- Elementy `<login>` a `<password>`
  Tieto elementy obsahujú prihlasovacie meno a heslo pre pripojenie k virtuálnemu počítaču. Na virtuálnom stroji musí byť vytvorený účet s týmito údajmi.

- Element `<intest>`
  Pri hodnote true tohto elementu je testovaný stroj zaradený do testovania, naopak pri false je vyradený. Pomocou administračného skriptu popísaného v sekcii 4.4.5 je možné modifikáciou tejto hodnoty stroje jednoducho zadávať či vyraďovať z procesu testovania.

- Elementy `<startup>` a `<testing>`
  Tieto elementy obsahujú zoznamy názvov skriptov spúšťaných na testovanom virtuálnom stroji. Musia byť umiestnené v adresári `\vmscripts\startup` respektíve `\vmscripts\testing`. Tieto skripty by mali mať záznam v závislostiach skriptov

testovania (viď 4.5.1). Tieto skripty sú súčasťou testovacích modulov, ktorým sa bližšie venuje sekcia 4.5.

- Element `<guestpath>`
  Tento element udáva adresár, v ktorom bude prebiehať testovací proces na virtuálnom stroji. Do tohoto adresára sa prenesú všetky dáta z adresára `vmdata/` na testovacom serveri, ktorý obsahuje všetky dáta k preneseniu.

### 4.4.3 Administrácia testovania

Pre jednoduchšiu manipuláciu s konfiguráciou testovacieho servera a s konfiguráciou jednotlivých testovacích strojov bol implementovaný skript `litstool`. Ten umožňuje interaktívne pridávanie nových testovacích strojov, odoberanie existujúcich a editovanie všetkých konfiguračných parametrov v oboch konfiguračných súborov. Poskytuje prehľadný výpis o registrovaných strojoch, strojoch, ktoré sú pridelené na testovanie a detailný výpis celej konfigurácie stroja. Príkaz vyžaduje ako prvý parameter špecifikáciu vyžadovanej operácie a podľa nej konkrétny počet ďalších vyžadovaných parametrov. Jednotlivé testovacie stroje sa rozlišujú podľa unikátneho mena alebo konfiguračného súboru VMware Servera spolu s cestou s príponou `.vmx`. Príklad pridania stroja s názvom "debian" k testovaniu:

```
% ./litstool change debian intest true
```

### 4.4.4 Démon testovania

Démon je počítačový program, ktorý beží v pozadí, bez priamej kontroly užívateľom. V testovacom serveri démon `litserver` implementuje proces testovania. Riadi celý testovací proces a pre testovanie strojov využíva testovacie moduly (viď 4.5). Démon spúšťa testovací proces v špecifikovaný čas podľa konfigurácie servera (viď 4.4.1). Testovací proces je možné spustiť aj zaslaním signálu `SIGUSR1` testovaciemu démonovi. Pre tento účel slúži prepínač démona `-r`, ktorý medzi bežiacimi procesmi vyhľadá bežiaci testovací server a zašle mu požadovaný signál. Na základe rovnakého princípu – posielania signálov je implementované korektné ukončenie testovacieho servera.

Démon testovania implementuje fáze 2, 3, 5 a 6 uvedené v návrhu procesu testovania v sekcii 4.3.1. Po spustení démon otestuje, či sú dostupné konfiguračné súbory nutné pre testovanie a či je spustený VMware Server s nakonfigurovaným sieťovým rozhraním. Po úspešnej inicializácii spracuje konfiguračný súbor a spustí nekonečný cyklus, v ktorom cyklicky po sekundových intervaloch čaká na dobu testovania. V každom cykle je znovu načítaný konfiguračný súbor a teda ak sa zmení konfigurácia, testovací démon na túto zmenu okamžite reaguje. Pri dosiahnutí času testovania alebo príchode požiadavku o spustení testovania sa začne testovací proces. Spracuje sa konfiguračný súbor testovaných strojov (popísaný v 4.4.2) a vyhodnotí sa konfigurácia strojov zaradených do testovania. Stroje, ktoré v testovaní nie sú, sa ignorujú. Spustia sa skripty pre prípravu a zber údajov podľa konfigurácie servera (sekcia 4.4.1) z adresára `scripts/startup`. Po tomto sa vstúpi do testovacieho cyklu, kde sa podľa konfigurácii maximálneho počtu paralelne bežiacich testovacích strojov spustia potomci procesu démona. Títo potomci zabezpečia paralelný beh testovania na viacerých strojoch. Potomok najprv spustí pridelený virtuálny stroj a počká na jeho zotavenie. Čakanie na zotavenie je realizované len ako pasívne čakanie istý maximálny čas

v závislosti od stavu vypnutia virtuálneho stroja. Následne po ukončení čakania sa pomocou protokolu SSH iniciuje spojenie s testovaním strojom. Po vytvorení spojenia sa vytvorí adresárová štruktúre pre testovanie. Pri tejto úlohe a ďalších nasledujúcich je potrebné spúšťanie príkazov na virtuálnom stroji cez iniciované SSH spojenie. Implementačné riešenie tohoto problému je ukázané v 4.4.6. Po vytvorení potrebných adresárových štruktúr sa pomocou príkazu SCP prekopírujú na testovacie stroje všetky údaje pre testovanie uložené v adresári `vmdata/` a všetky testovacie skripty z adresára `vmscripts/`. Popis implementácie tejto úlohy je detailne popísaný v sekcii 4.4.7. Testovacie skripty uvedené v konfigurácii testovacieho serveru sa začnú spúšťať v poradí akom sú uvedené – zľava do prava. Najprv sa spustia tzv. štartovacie skripty z adresára `vmscripts/startup`, ktoré pripravia prenesené dáta a po ich ukončení skripty z adresára `vmscripts/testing`. Pred spustením každého skriptu je kontrolované splnenie závislostí, ktoré podľa konfigurácie závislostí testovací skript má (viac v kapitole 4.5.1). Ukončením týchto skriptov je testovanie na virtuálnom stroji skončené. Testovaciemu procesu neostáva než opäť SCP spojením iniciovaným zo strany servera skopírovať vzniknuté záznamy testovania do úložiska testovacích záznamov – do adresára `logs/`. Záznamy sú tu uložené podľa dátumu testovania a unikátneho názvu testovaného virtuálneho stroja. Potomok po tomto poslednom úkone skončí. Ak testovací server zistí že ešte neboli otestované všetky virtuálne stroje v testovaní opakuje cyklus testovania pre všetky zvyšné testovacie stroje. Ako posledný krok v testovacom procese sa spustením skriptov uložených v `scripts/shutdown` spracujú výsledky testovania a publikujú na internetovú stránku `http://www.liberouter.org/~thrix`. Testovací server po tomto prejde do nekonečného cyklu spomínaného na začiatku tohto odstavca a čaká na nové testovanie.

### 4.4.5 Riadenie virtuálneho stroja

Pri procese testovanie sa vynára potreba riadiť a zisťovať stav virtuálnych strojov. Pod pojmom riadiť stav sa myslí vypínanie/zapínanie testovaného stroja. Stav virtuálnych strojov podľa [3] môže nadobúdať tieto hodnoty:

- vypnutý (off) – virtuálny stroj je zapnutý
- zapnutý (on) – virtuálny stroj je vypnutý
- pozastavený (suspended) – virtuálny stroj je pozastavený
- zaseknutý (stuck) – virtuálny stroj vyžaduje užívateľský vstup

Stav virtuálneho stroja je možné zmeniť príkazom `vmware-cmd` použitého v tvare:

```
% vmware-cmd <vmx súbor> start/stop/suspend hard/soft/trysoft
```

Prvý parameter udáva cestu ku konfiguračnému súboru VMware s príponou `.vmx`. Ďalší parameter vyjadruje požadovanú operáciu na stave a posledný spôsob prevedenia príkazu. Posledné dve možnosti pri spôsobe prevedenia príkazu (soft, trysoft) majú zmysel len ak používame na testovaných strojoch nástroje VMware–u, ktoré sú v našom prípade zbytočné.

### 4.4.6 Spúšťanie príkazov na virtuálnom stroji

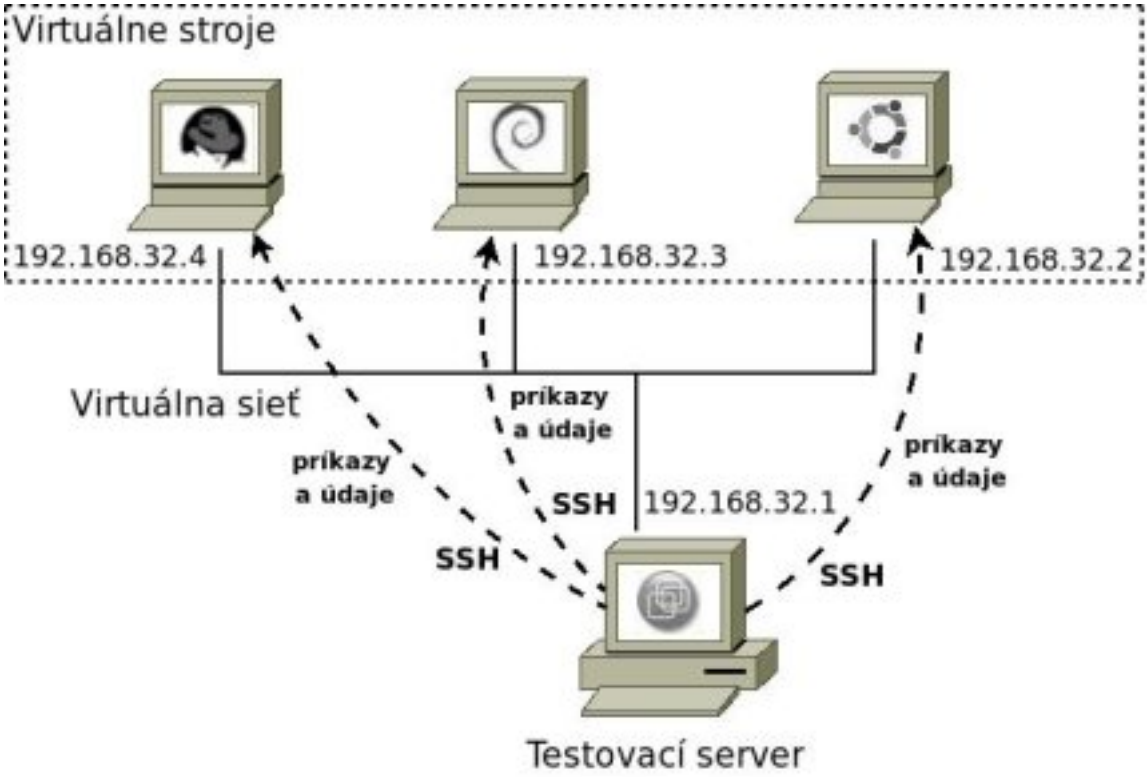
Pre potreby testovania je nutné spúšťať na virtuálnych strojoch príkazy. Vhodným riešením by bola možnosť prihlásiť sa na stroj a interaktívne zadávať príkazy, ale automatizovane pomocou programu. Riešenie tohto problému poskytuje modul Perlu – Expect [2]. Expect bol pôvodne vyvinutí ako rozšírenie jazyka TCL. Postupom času boli naimplementované moduly aj do ostatných skriptovacích jazykov, zastrešujúce funkcionalitu pôvodného Expectu. Expect umožňuje prácu s programami vytvorenými pre interakciu s človekom. Dokáže posielať vstup týmto programom a očakávať od nich odpoveď s využitím vyhľadávania zhody znakov pomocou regulárnych výrazov (angl. matching). Hlavnými príkazmi v rámci modulu sú príkazy:

- `spawn` – iniciuje spustenie príkazu predaného ako parameter a udržuje s nim spojenie až po príkaz `close`
- `send` – umožňuje poslať vstup interagujúcemu procesu
- `expect` – využitím vyhľadávania pomocou regulárnych výrazov dokáže rozpoznávať odpovede od interagujúceho procesu

V testovacom serveri je pre prihlasovanie na virtuálny stroj použitý Secure Shell. Tento využíva architektúru klient–server a vyžaduje beh servera na prihlasovacom stroji. Požiadavkom na virtuálnych strojoch je teda bežiaci SSH server (viď 4.5.1). Ten slúži pre bezpečné autentizované prihlasovanie na vzdialený počítač. Bezpečná autentizácia nie je vyžadovaná, keďže stroje sú podľa 3 odstavca návrhu (viď 4.3) neprístupné z lokálnej sieti hostiteľa. Heslá sú uložené pre každý virtuálny stroj v jeho konfigurácii, viď sekciu 4.4.1.

### 4.4.7 Prenos dát na virtuálny stroj

Pre prenos dát na testovací stroj sa ako najvýhodnejšie riešenie ponúka prenos pomocou SSH servera, ktorý je vyžadovaný podľa implementácie uvedenej v predchádzajúcej sekcii 4.4.6. Prenos súborov je vždy iniciovaný testovacím serverom a je vyžadovaný podľa návrhu procesu testovania (viď sekcia 4.3.1) na začiatku a konci testovania. Pre prenos je používaný program `scp`. Tento vyžaduje interakciu s užívateľom pretože pracuje nad SSH. Pre jeho riadenia je preto tiež použitý modul Expect. Na virtuálny stroj sú prenášané údaje pre testovanie z adresára `vmdata` a všetky testovacie skripty z adresára `vmscripts`. Oba tieto adresáre sú povinnou súčasťou adresárovej štruktúry testovacieho servera. Testovacie dáta sú na virtuálny stroj prenášané do adresára `data`. Skripty sú kopírované do adresára rovnako pomenovaného ako zdrojový na testovacom serveri. Oba prenášané adresáre sa umiestňujú v rámci adresára testovacieho procesu uvedeného v konfigurácii testovacieho servera (viď element `<guestpath>` v sekcii 4.4.2).




Obrázek 4.3: Obrázok znázorňuje 3 virtuálne stroje pripojené na virtuálnu sieť s hostiteľom virtualizačného softvéru – testovacím serverom.

### 4.4.8 Záznamy z testovania

Pri behu testovacieho servera vznikajú 3 rôzne záznamy:

- záznam testovacieho démona – je ukladaný do súboru `logs/lits.log` a obsahuje priebeh jednotlivých procesov v rámci testovacieho démona.
- záznamy testovacích procesov – sú ukladané v koreňovom adresári testovacieho servera do adresára `logs\dátum\názov_stroja\názov_stroja.log` a obsahujú priebeh testovacích procesov pre jednotlivé testovacie stroje.
- záznamy z behu jednotlivých testovacích skriptov – sa ukladajú do adresára `logs\dátum\názov_stroja\názov_skriptu.log` a obsahujú záznamy zo skriptov spúšťaných na testovacích strojoch.

Prvé dva vytvárané záznamy slúžia pre hľadanie príčin zlyhania testovacieho démona resp. testovacích procesov. Posledné záznamy z jednotlivých skriptov sú spracovávané v poslednej fáze testovacieho procesu skriptami uloženými v adresári `scripts/shutdown`. Obsahujú totiž výsledky testovacích skriptov a sú teda výstupom celého systému testovania.

## 4.5 Testovacie moduly

Testovacie moduly sú naimplementované maximálne štvoricou samostatných nezávislých skriptov. Skripty teda nie sú na testovacom démonovi nijako závislé. V priebehu testovania sú testovacím démonom len spúšťané bez akýchkoľvek parametrov. Testovacie skripty musia po úspešnom behu končiť s nulovou hodnotou a pri zlyhaní s ľubovoľnou nenulovou hodnotou. Toto je nutné pre funkcionalitu serveru, pretože to umožní testovaciemu démonovi sledovať závislosti spúšťaných skriptov.

V implementácii sú testovacie moduly ako celok definované len v súbore závislostí uvedeného v konfigurácii testovacieho servera `lits.conf` opísaného v 4.4.1. Skripty v jednotlivých testovacích moduloch sa môžu teda medzi sebou prelínať. Skripty v testovacích moduloch sa rozdeľujú do dvoch skupín:

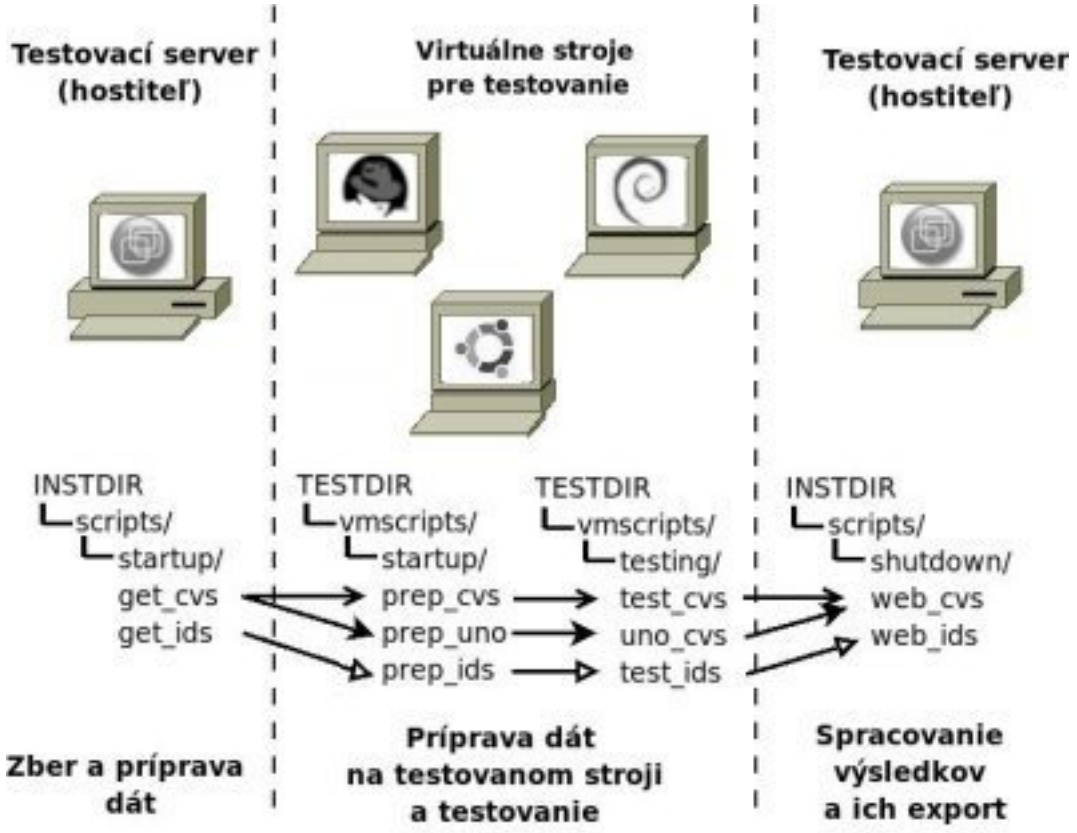
- **skripty pre zber vstupov testovania a spracovanie výsledkov testovania**
  Tieto skripty musia byť povinne uložené v rámci koreňového adresára programu v adresároch `scripts/startup` a `scripts/shutdown`. V rámci testovacieho cyklu sú tieto skripty spustené vždy len raz, na začiatku resp. na konci testovania. Ich úlohou je 1. a 5. fáza v procese testovania navrhnutého v sekcii 4.3.1. Skripty sú spúšťané s aktuálnou cestou nastavenou na koreňový adresár programu. Skripty spúšťané na začiatku testovania umiestňujú pozbierané dáta do adresára `vmdata/`, ktorý sa premiestňuje na testovacie stroje. Skripty určené pre spracovanie výsledkov pracujú so záznamami uloženými v adresári `logs/`.
- **skripty pre prípravu a riadenie testovania na virtuálnych strojoch**
  Tieto skripty musia byť umiestnené v rámci koreňového adresára programu v adresároch `vmscripts/startup` a `vmscripts/testing`. Skripty sa prenášajú do rovnakého adresára na testovanom virtuálnom stroji. Na virtuálnom stroji sú spúšťané z koreňového adresára testovacieho procesu uvedeného v konfigurácii virtuálneho stroja, viď 4.4.2. Skripty musia pracovať s dátami premiestnenými testovacím démonom do adresára `data/`. Vytvorené záznamy musia umiestňovať do adresára `logs/` v koreňovom adresári testovacieho procesu, ktorý sa po ukončení testovacích musí premiestnia na testovací server do adresára `logs/`.

### 4.5.1 Závislosti skriptov testovania

Závislosti skriptov testovania musia byť umiestnené v súbore `scriptdeps.conf`. Pre jeden testovací modul vyjadruje závislosť jeden riadok v tomto súbore. Ten sa skladá z názvov skriptov oddelených medzerami. Riadok má v súbore závislostí tvar:

```
skriptA skriptB skriptC skriptD
```

Tento riadok vyjadruje že každý skript je závislý na skupine predchádzajúcich skriptov. Teda, že skript `skriptB` je závislý na skripte `skriptA`. Skript `skriptC` je závislí na skriptoch `skriptA` a `skriptB` atď. Pri úspešnom vykonaní skriptu testovací démon uloží názov skriptu do poľa vykonaných skriptov. Toto pole a riadok zo súboru závislostí sa použije pri preverení, či skript, ktorý sa má vykonať, má splnené závislosti nutné pre beh.



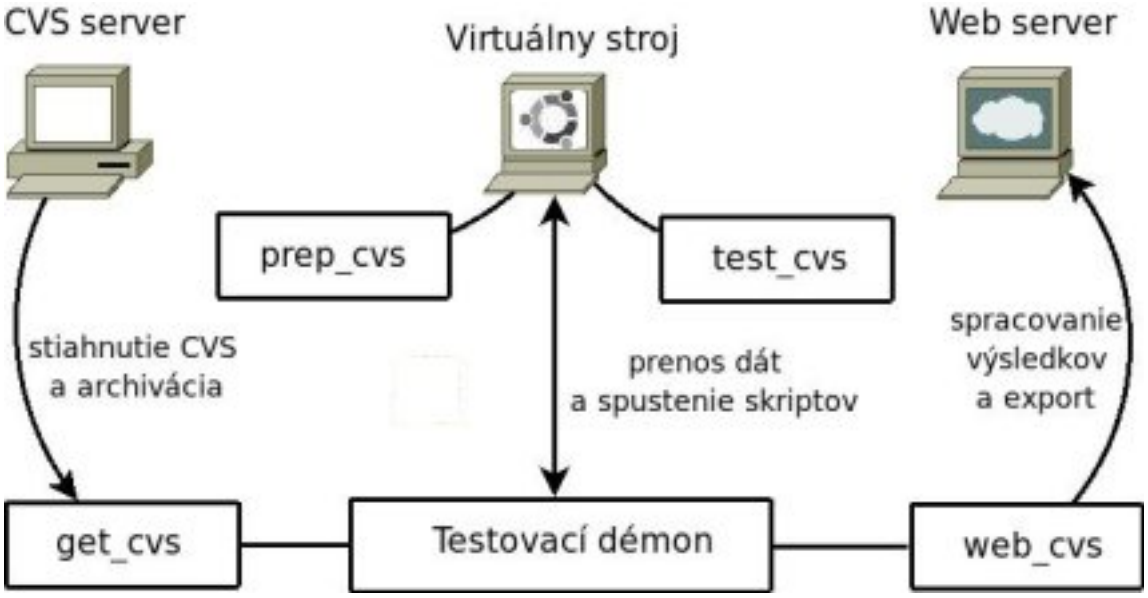

Obrázek 4.4: Na obrázku je znázornený príklad závislostí medzi skriptami a spúšťaným testovacím démonom.

Obrázok 4.4 ukazuje príklad závislostí medzi skriptami testovacích modulov. Každý modul je tvorený štvoricou skriptov. Prvý a posledný skript je spúšťaný na testovacom serveri - hostiteľovi virtualizačného softvéru, len raz na začiatku, resp. konci testovania. Znázornené je na ňom aj umiestnenie týchto skriptov a naznačené miesto behu. Adresár `INSTDIR` je koreňový adresár programu na testovacom serveri. `TESTDIR` je adresár, na testovaných virtuálnych strojoch, v ktorom prebieha sú umiestňované všetky údaje súvisiace s testovaním. Jednotlivé závislosti sú od seba odlíšené rôznym druhom šipiek. Každý skript závisí od všetkých skriptov, ktoré mu predchádzajú. Teda napr. skript `uno_cvs` je závislý od skriptov `prep_uno` a `get_cvs`. Súbor závislostí `scriptsdeps.conf` pre tri testovacie moduly znázornené na obrázku by mal pre prípad na obrázku nasledovný obsah:

```
get_cvs prep_cvs test_cvs web_cvs
get_cvs prep_uno uno_cvs web_cvs
get_ids prep_ids test_ids web_ids
```

## 4.6 CVS testovací modul

Táto sekcia podrobne opisuje implementovaný testovací modul pre preklad a inštaláciu CVS Liberoutru. Na adrese `www.liberouter.org/~thrix/cvs_stats.html` poskytuje testovací modul informácie o výsledkoch testov. Testovací modul sa skladá zo 4 skriptov a ich funkciu naznačuje nasledujúci obrázok:




Obrázek 4.5: Obrázok ukazuje úlohu jednotlivých skriptov v rámci CVS modulu.

### 4.6.1 Skript `get_cvs`

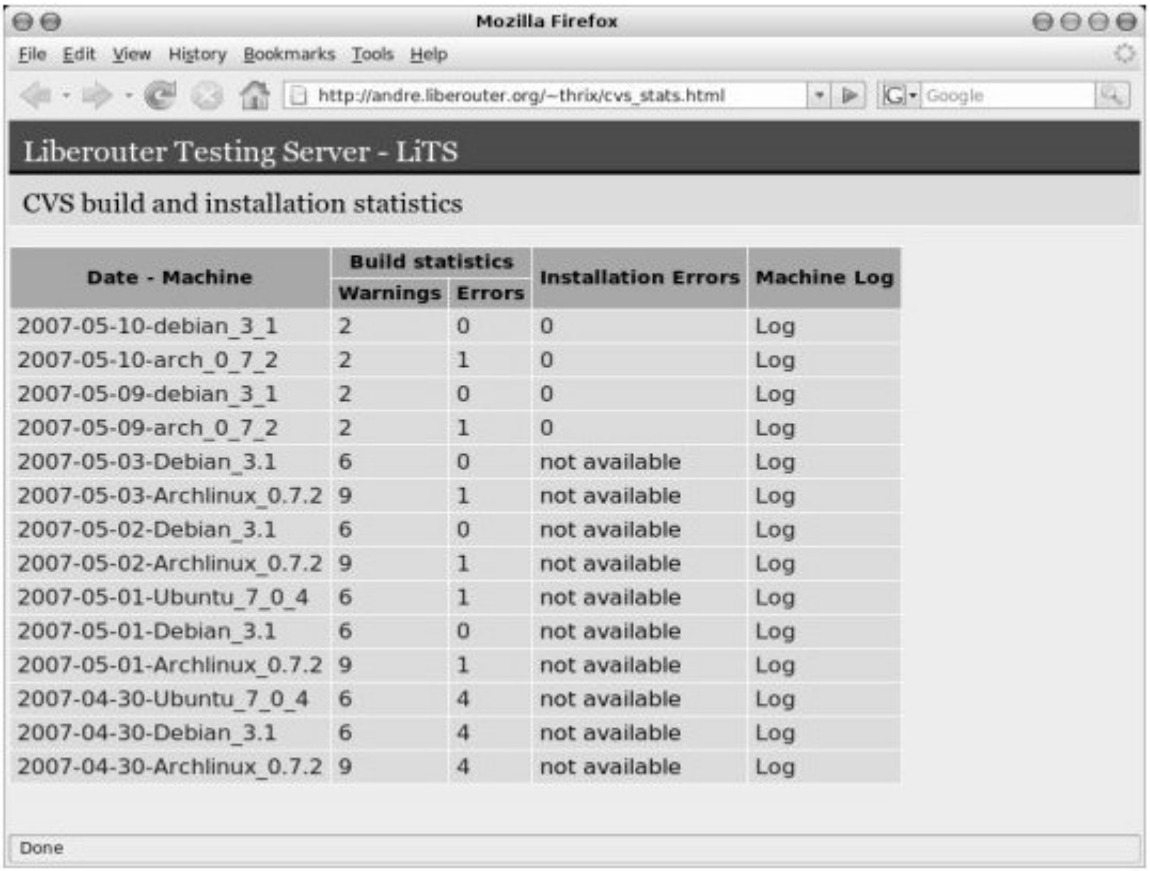
Tento skript je spúšťaný na začiatku testovacieho procesu. Jeho úlohou je export aktuálnej CVS na testovací server. Keďže CVS je sprístupnená heslom cez šifrované SSH spojenie, skript musí riešiť otázku exportovania bez nutnosti odkrytia tohto hesla. Riešením je použitie SSH kľúčov v spojení s SSH agentom pred behom servera. Verejný kľúč musí byť umiestnený na CVS serveri a privátny na testovacom serveri. Po exporte sa testovaná CVS ešte archivuje do formátu TAR, keďže CVS obsahuje veľké množstvo malých súborov. Pripravený archív s aktuálnou CVS sa potom uloží do adresára `vmdata\` a pripraví sa tak pre prenos na testovacie stroje. Prenos súborov na testovací stroj má na starosti testovací démon, viď sekciu 4.4.4.

### 4.6.2 Skript `prep_cvs`

Po prenesení údajov na testovací stroj je tento skript, pomocou riadeného SSH spojenia testovacím démonom (viď sekciu 4.4.7), spustený. Jeho úlohou je pripraviť prenesený archív na testovanie. Toto zahŕňa rozbalenie archívu CVS do koreňového adresára testovanie uvedeného v konfigurácii testovacieho stroja – sekcia 4.4.2.

### 4.6.3 Skript `test_cvs`

Tento skript riadi konfiguráciu, preklad a inštaláciu CVS na testovanom stroji. Skript je spúšťaný rovnako ako skript `prep_cvs` – riadeným SSH spojením. Inštalácia CVS do systému vyžaduje administrátorské práva, takže požiadavkom pre správny beh celého modulu je administrátorsky účet uvedený v konfigurácii testovacieho stroja (sekcia 4.4.2). Konfigurácia CVS sa spúšťa príkazom `setup`. Tá zahŕňa proces generovania súborov potrebných pre preklad, ktorého záznam nie je potrebné ukladať. Preklad sa spúšťa príkazom make a prepínačom pre ignorovanie prípadných chýb. Ten zabezpečí že preklad prebehne na celom CVS strome a nie len po vetvu, kde sa nájde chyba. Záznam z prekladu z štandardného chybového výstupu a z štandardného výstupu je ukladaný do spoločného súboru `test_cvs.log` v adresári `logs/`. Proces inštalácie sa spúšťa príkazom `make install` a jeho záznam je ukladaný do súboru `inst_cvs.log` v adresári `logs/`. Po skončení skriptu je testovanie na stroji ukončené a stroj je vypnutý testovacím démonom.




| Date - Machine | Build statistics | | Installation Errors | Machine Log |
|---|---|---|---|---|
| | Warnings | Errors | | |
| 2007-05-10-debian_3_1 | 2 | 0 | 0 | Log |
| 2007-05-10-arch_0_7_2 | 2 | 1 | 0 | Log |
| 2007-05-09-debian_3_1 | 2 | 0 | 0 | Log |
| 2007-05-09-arch_0_7_2 | 2 | 1 | 0 | Log |
| 2007-05-03-Debian_3.1 | 6 | 0 | not available | Log |
| 2007-05-03-Archlinux_0.7.2 | 9 | 1 | not available | Log |
| 2007-05-02-Debian_3.1 | 6 | 0 | not available | Log |
| 2007-05-02-Archlinux_0.7.2 | 9 | 1 | not available | Log |
| 2007-05-01-Ubuntu_7_0_4 | 6 | 1 | not available | Log |
| 2007-05-01-Debian_3.1 | 6 | 0 | not available | Log |
| 2007-05-01-Archlinux_0.7.2 | 9 | 1 | not available | Log |
| 2007-04-30-Ubuntu_7_0_4 | 6 | 4 | not available | Log |
| 2007-04-30-Debian_3.1 | 6 | 4 | not available | Log |
| 2007-04-30-Archlinux_0.7.2 | 9 | 4 | not available | Log |

Obrázek 4.6: Stránka s výsledkami merania chýb zdrojového kódu a chýb pri inštalácii generovaná CVS testovacím modulom.

### 4.6.4 Skript `web_cvs`

Po otestovaní všetkých strojov určených pre testovanie sú všetky záznamy uložené v adresári `logs\dátum\názov_stroja\`. Tento skript má za úlohu spracovať všetky záznamy a umiestniť ich na web server, ktorý umožňuje prihlasovanie bez hesla, rovnako ako server CVS

v sekcii 4.6.1. Zo záznamov z prekladu sú vytriedené varovné a chybové hlásenia do oddelených súborov. Chyby z inštalácie sú tiež vytriedené a uložené do samostatného súboru. Vytvorené textové súbory sú spolu so záznamom o testovacom procese (viď sekciu 4.4.8) prenesené na web server. Existujúca HTML stránka zo servera je aktualizovaná o odkazy na pridané nové výsledky. Odkazy sú štruktúrované do tabuľky a sú k nahliadnutiu na stránke `www.liberouter.org/~thrix/cvs_stats.html`.

## 4.7 Dokumentácia

K testovaciemu serveru bola vyhotovená dokumentácia. Dokumentácia je písaná v anglickom jazyku v súlade s programovacími zásadami Liberoutru. Je umiestnená v koreňovom adresári programu v súbore README. Dokumentácia popisuje spôsob inštalácie, ukazuje prácu s program, spôsob testovania a sú tu uvedené softvérové požiadavky, ktoré sú nutné pre správny beh testovacieho servera. V adresároch `vmscripts` a `scripts` sa nachádzajú ďalšie README súbory, ktoré zdôrazňujú pravidlá pre písanie nových testovacích skriptov. Formou programovej dokumentácie sú početné komentáre a popisy jednotlivých funkcií.

# Kapitola 5

# Záver

Cieľom bakalárskej práce bolo vytvoriť metodiku pre testovanie zdrojového kódu projektu Liberouter a túto metodiku implementovať ako testovací server. Na obrázku 4.6 vidieť výsledok a zmysel práce. Testovací server bežiaci na jednom z počítačov projektu zobrazuje výsledky o chybách pri preklade a inštalácii archívu CVS na rôznych operačných systémoch. Výsledky poskytujú programátorom v softvérovej skupine projektu jednoduchú a prehľadnú dennú štatistiku o chybovosti aktuálneho kódu v CVS. Testovací server týmto dopomáha k rýchlej lokalizácii problémov v tejto časti programového vybavenia.

Navrhnutá metodika testovania umožňuje úlohu automatizovaného testovania ľubovoľne rozšíriť pomocou modulov. Moduly sú tvorené skupinou skriptov napísaných vo voliteľnom programovacom jazyku. V princípe sa funkcia modulov nemusí obmedzovať len na aplikáciu automatizovaného testovania. Moduly môžu slúžiť napríklad ako automatizovaný buildovací systém inštalačných balíčkov na širokej škále operačných systémov. Server je s výhodou využiteľný, ak je zapotreby automatizovaný multi–platformný systém pre ľubovoľné použitie.

Aktuálny stav práce s výsledkami testovacieho modulu pre CVS ukázali správnosť navrhnutej metodiky testovania. Súčasný stav testovania je k nahliadnutiu na stránke `www.liberouter.org/~thrix`. Testovanie sa v budúcnosti rozšíri o testovanie s použitím statického analyzátoru kódu Splint. Pribudnú tiež moduly pre testovanie prekladu a inštalácie balíčkov vyvíjaných v rámci projektu.

Výzvou k ďalšej práci na testovacom serveri je vyriešenie problému prístupu k hardvéru z testovaných virtuálnych strojov. Súčasné riešenie neumožňuje prístup k hardvérovým prostriedkom vyvíjaným v rámci projektu. Testovaný systém by po odstránení tohto nedostatku umožnil funkčné testovanie vytvoreného programového vybavenia popri statickej analýze.

# Literatura


[1] Dařena, F.: *Myslíme v jazyku Perl*. Praha: Grada, 2005, ISBN 80-247-1147-8.

[2] Giersig, R.: Dokumentácia modulu Expect.pm. 2006, [Online; navštívená 2. 5. 2007].
URL http://search.cpan.org/~rgiersig/Expect/Expect.pod

[3] VMware; inc.: VMware Scripting API. 2003, [Online; navštívená 2. 5. 2007].
URL http:
//www.vmware.com/support/developer/scripting-API/doc/Scripting_API.pdf

[4] Wikipedia: Software testing. 2007, [Online; navštívené 2. 5. 2007].
URL http://en.wikipedia.org/wiki/Software_testing

[5] Wikipedia: Static code analysis. 2007, [Online; navštívené 2. 5. 2007].
URL http://en.wikipedia.org/wiki/Static_code_analysis

[6] Wikipedia: Virtualization. 2007, [Online; navštívené 2. 5. 2007].
URL http://en.wikipedia.org/wiki/Virtualization