VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
AUTOMATIZACE A MĚŘICÍ TECHNIKA

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
CONTROL AND INSTRUMENTATION

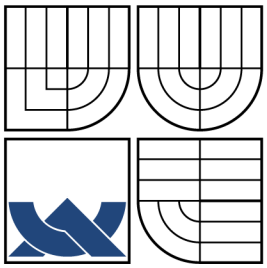
AUTOMATICKÉ ZPRACOVÁNÍ POKRYTÍ KÓDU JÁDRA OPERAČNÍHO SYSTÉMU GNU/LINUX

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

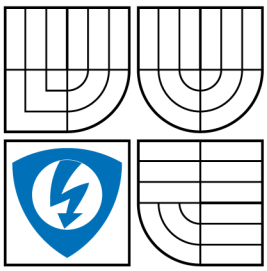
AUTOR PRÁCE
AUTHOR

LUKÁŠ DOKTOR

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ
AUTOMATIZACE A MĚŘICÍ TECHNIKA

FACULTY OF ELECTRICAL ENGINEERING AND
COMMUNICATION
CONTROL AND INSTRUMENTATION

AUTOMATICKÉ ZPRACOVÁNÍ POKRYTÍ KÓDU JÁDRA OPERAČNÍHO SYSTÉMU GNU/LINUX

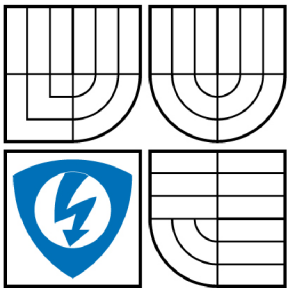
AUTOMATIC PROCESS OF CODE COVERAGE FOR GNU/LINUX'S KERNEL

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — LUKÁŠ DOKTOR
AUTHOR

VEDOUCÍ PRÁCE — DOC. ING. JOZEF HONEC, CSC.
SUPERVISOR

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav automatizace a měřicí techniky

Bakalářská práce

bakalářský studijní obor
Automatizační a měřicí technika

Student:	Doktor Lukáš	***ID:***	83845
Ročník:	3	***Akademický rok:***	2007/2008

NÁZEV TÉMATU:

Automatické zpracování pokrytí kódu u jádra operačního systému GNU/Linux

POKYNY PRO VYPRACOVÁNÍ:

Umožnit profilaci jádra operačního systému GNU/Linux. Zautomatizovat instalaci a spouštění sad testů pod těmito jádry. Umožnit celkové i detailní - souborově oddělené - pokrytí kódu. Test musí být proveditelný s možností, ale bez nutnosti zásahu uživatele. Podporované architektury jsou i386, x86_64, s390, s390x, ppc, ppc64 a ia64.

DOPORUČENÁ LITERATURA:

Termín zadání:	1.2.2008	***Termín odevzdání:***	2.6.2008

Vedoucí práce: doc. Ing. Jozef Honec, CSc.

prof. Ing. Pavel Jura, CSc.
předseda oborové rady

UPOZORNĚNÍ:

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami:

1. Pan/paní

Jméno a příjmení: Lukáš Doktor
Bytem: Krumlovská 1238/74, 37007, České Budějovice - České Budějovice 7
Narozen/a (datum a místo): 1.6.1985, Jindřichův Hradec

(dále jen "autor")

a

2. Vysoké učení technické v Brně

Fakulta elektrotechniky a komunikačních technologií
se sídlem Údolní 244/53, 60200 Brno 2
jejímž jménem jedná na základě písemného pověření děkanem fakulty:
doc. Ing. Václav Jirsík, CSc.

(dále jen "nabyvatel")

Článek 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
   - ☐ disertační práce
   - ☐ diplomová práce
   - ☑ bakalářská práce

   jiná práce, jejíž druh je specifikován jako .....

(dále jen VŠKP nebo dílo)

Název VŠKP: Automatické zpracování pokrytí kódu u jádra operačního systému GNU/Linux
Vedoucí/školitel VŠKP: doc. Ing. Jozef Honec, CSc.
Ústav: Ústav automatizace a měřicí techniky
Datum obhajoby VŠKP: .....

VŠKP odevzdal autor nabyvateli v:

- ☑ tištěné formě - počet exemplářů 1
- ☑ elektronické formě - počet exemplářů 1

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.

3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizovaní výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti
   - ☑ ihned po uzavření této smlouvy
   - ☐ 1 rok po uzavření této smlouvy
   - ☐ 3 roky po uzavření této smlouvy
   - ☐ 5 let po uzavření této smlouvy
   - ☐ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: .....

....

Nabyvatel Autor

ABSTRAKT

Řeší automatické získávání profilačních informací jádra operačního systému GNU/Linux. Využívá spojení existujícího projektu gcov-kernel, Red Hat Test System a volitelných testovacích programů.

Red Hat Test System pomáhá s instalací a nastavení počítače. Modifikované jádro umožňuje přistupovat k profilačním datům běžícího jádra. Po ukončení práce jsou tyto informace zpracovány do lidsky čitelné formy.

KLÍČOVÁ SLOVA

GNU, Linux, OS, kernel, pokrytí kódu, Red Hat, RHEL, Fedora, RHTS, Red Hat Test System

ABSTRACT

Solves an automatic getting of profilation informations of the GNU/Linux's kernel. Solution is based on an interconnection of existing project gcov-kernel, Red Hat Test System and optional test suit.

Red Hat Test system helps with an installation and setting up an computer. Modificated kernel by gcov-kernel provides profillations data of the running system. After all job is done, those informations are proceed into human readable form.

KEYWORDS

GNU, Linux, OS, kernel, code coverage, Red Hat, RHEL, Fedora, RHTS, Red Hat Test System

DOKTOR, L. *Automatické zpracování pokrytí kódu u jádra operačního systému GNU/Linux*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2008. 54 s. Vedoucí bakalářské práce doc. Ing. Jozef Honec, CSc.

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Automatické zpracování pokrytí kódu jádra operačního systému GNU/Linux“ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a 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)

PODĚKOVÁNÍ

Děkuji vedoucímu semestrální a bakalářské práce panu doc. Ing. Jozefu Honcovi, CSc. za umožnění práce ze strany VUT.
Dále děkuji panu Jeraldovi Turnerovi za užitečnou metodickou pomoc při realizaci práce a za vedení ze strany firmy Red Hat, Inc.
V neposlední řadě děkuji Luďkovi Šmídovi za podrobné uvedení do problematiky testování.

V Brně dne ....

(podpis autora)

OBSAH

SEZNAM OBRÁZKŮ

1 ÚVOD

Operační systém GNU/linux[1] je znám svou velkou stabilitou. Vzhledem k tomu vzrůstá jeho oblíbenost i na kriticky důležitých trzích jako je bankovnictví, finanční burzy či armáda. Aby tento trend mohl dále pokračovat, je bezpodmínečně nutné udržet, stabilitu celého operačního systému, i přes rozsáhlý vývoj. Toho lze dosáhnout pouze dostatečně silným testováním všech částí a zároveň celku.

Jedna z nedílných součástí každého operačního systému je jádro. To obstarává přidělování procesoru, paměti, umožňuje přístup k periferiím a mnoho dalšího, čímž připravuje živnou půdu pro uživatelské aplikace. Hlavní rozdíl oproti aplikacím je v době chodu. Jádro operačního systému bývá zavedeno a běží řady let bez restartu. Proto se každá chyba, každý ztracený kilobyte i deathlock[2] nutně projeví. Právě proto je kontrola jádra primární.

V oblasti výpočetní techniky prozatím nebyla odhalena metoda, která by nám sdělila, kdy je testování dostatečné. Existuje ale několik projektů, které se vcelku úspěšně touto problematikou zabývají a vytvářejí různé způsoby, jež mohou zkušenému odborníkovi pomoct v rozhodování.

Tato práce se zabývá využitím jedné z metod, která pomáhá vyhledat bílá místa[3] přímo v běžícím jádře operačního systému.

[1] Plný název GNU/Linux je v dokumentu často zkrácen na Linux

[2] Deathlock - termín označující souběžný přístup k jednomu zdroji vedoucí k zablokování komponenty

[3] Bílé místo - část kódu, která nebyla otestována

2 TEORETICKÝ ÚVOD

2.1 Testování

Ke správnému pochopení celého problému je nutná alespoň základní znalost testování.

Základní pravidlo testování zní: „**Test je úspěšný, POUZE když najde problém!**“

Druhy testování

- Black box - pouze z vnějšku nahlíží na testovaný objekt
- White box - zkoumá vnitřní strukturu testovaného objektu

Úrovně testování

Unit
* Typ White box
* Ověřuje určitou funkčnost (malou část kódu)
* Testováno většinou vývojářem
* Většinou bez jakéhokoliv test-plánu

Integration
* Typ Black box
* Testuje celkovou funkčnost projektu složeného z více modulů
* Většinou se týká testování rozhraní a protokolů
* Probíhají na základě test-plánu

System
* Typ Black box
* Testování kompletního systému projektů na definovanou způsobilost
* Probíhá na základě test-plánu
* Smoke / Sanity testy, které předcházejí přímému testování, mají odhalit totální selhání, jak nejrychleji je to možné

Testování v této práci

Tato práce se zabývá možností, jak změřit úroveň otestování jádra. Rozsahem toto testování spadá do kategorie System testing.

Z pohledu certifikace je ideální otestovat veškerou funkčnost. U rozsáhlých projektů toto není možné vykonat. V tomto případě je za potřebí vytvořit dostatečně širokou škálu testů, která pokryje většinu případných nedostatků.

Když systém projde testováním, je zapotřebí posoudit úroveň testování. Jak bylo naznačeno v úvodu, existuje mnoho způsobů, jak tuto míru určit. Tato práce se dále zabývá tím, jak tuto míru určit pomocí pokrytí kódu[1].

2.2 Pokrytí kódu

Pokrytí kódu je měřítko používané při testování softwaru. Popisuje stupeň ověření zdrojových kódů. Principiálně funguje tak, že při běhu programu je zaznamenáno vykonávání jednotlivých bloků programu.

Techniky pokrytí kódu byly jedny z prvních vynalezených systémů pro systematické testování. Poprvé se o něm zmiňují Müler a Maloney v knize Communication of the ACM již v roce 1963.

V průběhu let se rozvinuly tyto čtyři formy:

1. Statement Coverage - byly všechny řádky kódu provedeny?
2. Condition Coverage - byla všechna rozhodující místa (true/false) provedena otestována?
3. Path Coverage - byly všechny rozhodující možnosti použity a otestovány?
4. Entry/Exit Coverage - byla všechny volání a návraty provedeny a otestovány?

Výsledky jednotlivých metod jsou značně rozlišné. Proto je nutné dobře znát testovaný projekt a zvolit správnou metodu.

Důležité je také připomenout, že se jedná o navázání informací z přeloženého spustitelného binárního souboru na jeho zdrojové kódy. Vzhledem k rozšířeným instrukcím procesoru a optimalizacím při překladu často tento kód neodpovídá a výsledky mohou být zavádějící.

2.3 Použité projekty

Pro řešení byly využity některé již existující projekty. V následujících částech je vysvětlena jejich funkce a způsob použití.

2.3.1 GCC

GCC je nejrozšířenější překladač nejen jazyka C pro linux a je jediný podporovaný pro linuxové jádro.

[1] Pokrytí kódu - *z ang. Code coverage* - posuzuje míru testování podle vykonaných řádků aplikace

Umožňuje pomocí speciálních voleb doplnit překládaný kód o značky, na jejichž základě je při chodu programu zaznamenán každý blok binárního kódu a je zaznamenán k odpovídající řádce zdrojového kódu. Dále zaznamenává i procesorový čas na odpovídající řádku kódu. Informace jsou uloženy v souborech `.gcno` a `.gcda`.

GCC **je využit pro překlad jádra a obohacení o profilační data.**

2.3.2 GCOV

GCOV zpracovává `.gcno` a `.gcda` soubory do lidsky čitelného formátu `.gcov`. Umožňuje všechny čtyři základní formy pokrytí kódu. Má mnoho pomocných voleb, které jsou podrobně vysvětleny v manuálových stránkách programu.

Výstupní soubor odpovídá zdrojovému souboru v pravém sloupci a zvoleným informacím v sloupečku levém. Pokud zdrojové kódy nejsou k dispozici, je možné získat pouze celkové procentuální pokrytí kódu.

Ukázka výstupního souboru:

```
 -:    0:Source:tmp.c
 -:    0:Graph:tmp.gcno
 -:    0:Data:tmp.gcda
 -:    0:Runs:1
 -:    0:Programs:1
 -:    1:#include<stdio.h>
10:    2:int sum (int a, int b) {
10:    3:    return (a + b);
 -:    4:}
 -:    5:int main(void)
 1:    6:{
 -:    7:    int i, total;
 1:    8:    total = 0;
11:    9:    for (i = 0; i < 10; i++)
10:   10:        total = sum(total, i);
 1:   11:    total != 45 ? printf(,,Failure\n‘‘)
                          : printf(,,Success\n‘‘);
 1:   12:    return 0;
 -:   13:}
```

GCOV **může být použit pro podrobné informace o pokrytí kódu v průběhu chodu.**

2.3.3 gcov-kernel

Jaderný modul[1] poskytující informace o pokrytí kódu jádra podobně ve formátu, který je známý z userspace[2] aplikací.

U userspace programů je jednoduché vytvořit soubor a využívat jej pro uchovávání informací o běhu programu. V kernelspace[3] toto možné není. Proto jsme sice schopni s jistými úpravami přeložit jádro s profilačními informacemi, ovšem bez jakékoliv možnosti přistoupit k těmto informacím.

Tento projekt vytváří adresářovou strukturu v adresáři `/proc/gcov`, která odpovídá adresářové struktuře zdrojových kódů jádra. V této struktuře pak zpřístupňuje informace o pokrytí kódu. Důležitým faktem je, že se jedná o stejný formát používaný v userspacu, soubory `.gcno` a `.gcda`. To nám umožňuje použít již mnoho let existující utility z userspacu.

Patch GCOV-KERNEL **je využit pro zpřístupnění** `.gcno` **a** `.gcda` **souborů.**

2.3.4 LCOV

Projekt LCOV je sadou perl scriptů, jež rozšiřují projekt GCOV o přehledné rozhraní. Jeho použití je hlavně v rozsáhlých projektech, což přesně vyhovuje zadání. Uživatelsky jsou výrazné dva základní skripty:

- `geninfo` - sbírá informace z `.gcno` a `.gcda` souborů a vytváří souhrný soubor `.info`.
- `genhtml` - zpracovává `.info` soubor a vytvoří hypertextovou stránku ke každému zdrojovému souboru. Zpětně pak dotváří obecné a souhrnné informace, které též prezentuje pomocí hypertextové stránky. Ve výsledku dostane člověk velice přehledné hypertextově propojené stránky, a je tedy schopen i u rozsáhlejšího projektu rychle najít bílá místa.

LCOV **je defaultním řešením pro prezentování výsledků**

2.3.5 GenSum

Poskytuje souhrnné informace o pokrytí kódu na základě `.info` souboru.

[1] Jaderný modul - program, jenž je schopen se za běhu jádra do něj zavést a doplňit jej tím o svou funkci

[2] Userspace - část počítače vyhrazená pro běžné uživatelské aplikace; opakem je Kernelspace[3]

[3] Kernelspace - část počítače vyhrazená pro běh jádra operačního systému; opakem je Userspace[2]

Vzhledem k časové náročnosti generování informací programem LCOV jsem vytvořil tento drobný program, který je schopen zpracovat `.info` mnohonásobně rychleji. Za tuto rychlost je ovšem zaplaceno mírou poskytnutých informací. V některých případech je celkové pokrytí dostačující, a tak je možné zapnout tuto formu výstupu.

Dalším důvodem vzniku tohoto programu byla problematičnost použití LCOV na jiných než x86 architekturách.

GenSum je alternativním řešením prezentování výsledků.

2.3.6 RHTS

RHTS je primární testovací systém na testování RHELu. Rozděluje se na dvě části.

- Test scheduler *(Plánovač testů)* - řídí a obhospodařuje počítačové labolatoře pro běh testů. Řeší vše kolem vykonávání testů, výběry strojů dle požadavků, instalace distribucí, správu (restartování a reinstalace hostů, kteří přesáhli nastavený čas) a řídí spolupráci testů na více strojích, které spolu potřebují komunikovat. Zároveň obstarává způsoby spuštění testů.
- Individual test *(Jednotlivý test)* - představuje nejdůležitější část RHTS. Jednotlivé testy jsou napsány ve formátu srozumitelném plánovači, takže jím mohou být automaticky spuštěny na různých distribucích, například: RHEL3, RHEL4, RHEL5 a Fedora. Zároveň umožňuje spuštění na různých architekturách, například: `i386`, `ia64`, `ppc, s390`, `s390x` a `x86_64`. Testy napsané v RHTS formátu mohou být spuštěny pomocí *lab controller*u *(Řadiče labolatoře)*, nebo, pokud je na lokálním stroji nainstalován balík `rhts-devel`, přímo z příkazové řádky.

Testy se do fronty přidávají jako jednotlivé jobs *(práce)*. Ty je možné zadat pomocí webového rozhraní, nebo přímo z příkazové řádky jako workflow *(pracovní tok)*. Jedná se o pythnový skript, který vytvoří, přesně definuje a přes `XML/RPC` pošle job na scheduler. Umožňuje nám zadat mnohem více parametrů a tím rozšiřuje možnosti RHTS.

RHTS je použito pro spojení všech projektů od instalace, přes testy až ke sběru informací o pokrytí kódu

3 ŘEŠENÍ STUDENTSKÉ PRÁCE

Vlastní řešení jsem rozdělil do dvou základních etap, neboť samotný projekt se nejprve vyvíjel jedním směrem, a až na základě nových požadavků kolegů na Developer Conferenci[1] jsem nakonec celý projekt předělal. Tím jsem výrazně zjednodušil a rozšířil jeho použitelnost. Vzniku druhé verze znatelně pomohlo vydání nové verze RHTS, ve které byly uvedeny nové, nezbytné funkce.

Pracovní název projektu je kCOV.

3.1 kCOV - obecný úvod

3.1.1 Pokrytí kódu v kernelu

Firma Red Hat při tvorbě RHELu dbá nejen na rychlost systému, ale zároveň na stabilitu. Ta je rapidně zvýšena garancí stejných verzí knihoven, programů i jádra po celou dobu podporovaného cyklu operačního systému. Tento fakt je zde uveden proto, že výrazně ovlivnil použitelnost a styl návrhu projektu. Pro představu, průměrná doba podporovaného cyklu je cca 5 let, přičemž několik dalších let jsou stále dodávány bezpečnostní záplaty. Vývoj kolem operačního systému linux je ovšem velmi rychlý. Za jeden rok vyjde zhruba 3 - 4 nové verze linuxového jádra s mnoha novinkami. Po celou dobu podpory distribuce jsme nuceni tyto novinky přebírat a modifikovat je tak, aby běžely na starých jádrech.

V době založení tohoto projektu byly aktuální verze RHEL3 a RHEL4 s jádry `2.4.21` a `2.6.9` a aktuální stabilní jádro `2.6.11`. Vzhledem k tomuto faktu se RHEL jádra značně lišila od oficiálních jader. Proto patch poskytovaný projektem gcov-kernel nebyl na tato jádra aplikovatelný. Bylo nutné jej téměř kompletně přepsat, a i tak se funkce na jádrech RHEL3 výrazně lišily. Pro úspěšné použití jsem vytvořil sadu skriptů, jež plně eliminovala nedostatky. S vydáním RHEL5 se upustilo od myšlenky certifikace RHEL3 jader. U RHEL4 a RHEL5 se podařilo prosadit změny, které vedly ke shodnému rozhraní a tím ke správné funkčnosti bez nutnosti podpůrných skriptů.

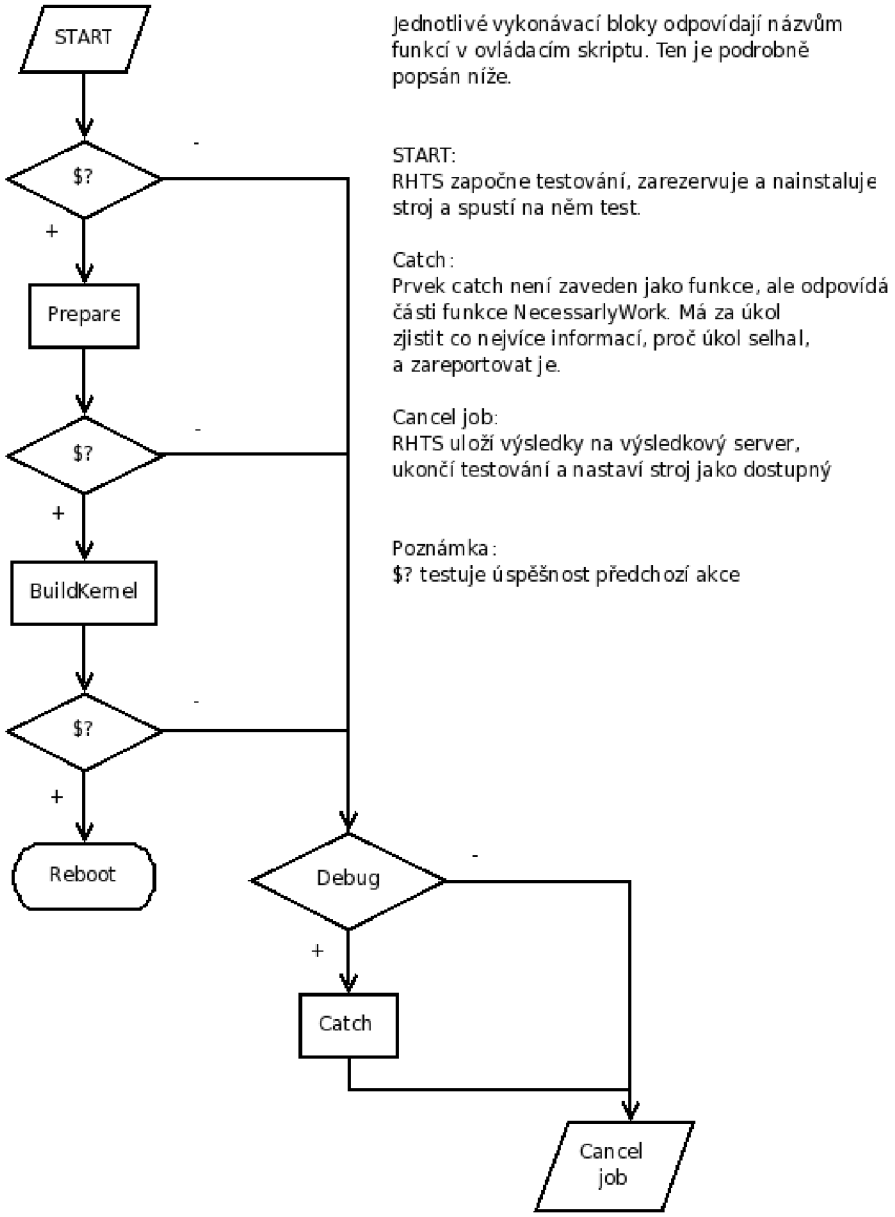
3.1.2 kCOV

kCOV je RHTS test, který definuje celou procedůru úpravy jádra, sady testů, sběru informací a poskytnutí výsledků. Je definován jako BASH script, který spouští,

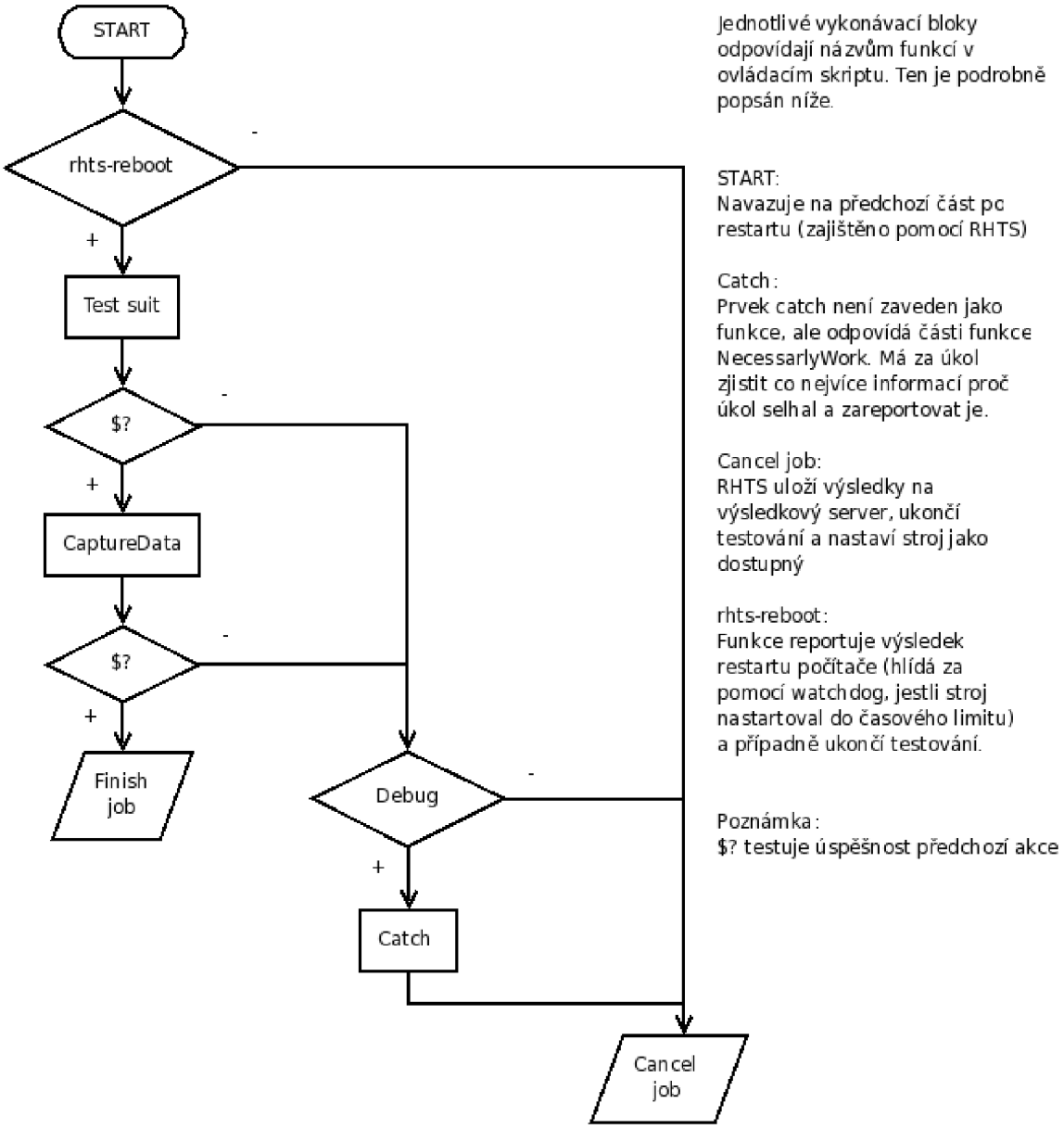
[1] Developer Conference - každoroční interní konference zaměstnanců firmy Red Hat Czech s. r. o.

reaguje a reportuje jednotlivé části.

Celý proces testování a sběru dat je zobrazen na diagramech 3.1 a 3.2.



Obrázek 3.1: Diagram chodu kCOV před restartem



Obrázek 3.2: Diagram chodu kCOV po restartu

Přesný popis je uveden v následujících dvou podkapitolách rozčleněn dle jednotlivých verzí.

3.2 kCOV v1

3.2.1 Obecný popis

Sestává se z jednoho testovacího balíku. Je napsaný víceméně staticky a neumožňuje mnoho nastavení bez přímého zásahu do zdrojového kódu. To odpovídá původnímu záměru pouze pro cerifikaci jader. Test obsahuje několik souborů, jejichž smysl je popsán níže:

- `Makefile` - definuje metadata o testu, doplňuje se z defaultního Makefilu RHTS (`/usr/share/rhts/lib/rhts-make.include`). Je určen k vytvoření balíku, odeslání na CVS[2] i SCHEDULERu a na jeho následné spuštění na cílovém stroji.
- `PURPOSE` - obsahuje informace o testu, jeho použití a případně licenci.
- `runtest.sh` - `BASH` script řídící chod testování. Na začátku spustí `/usr/bin/rhts_environment.sh`, čímž nastaví prostředí a přidá nutné funkce RHTS. Podrobný rozbor je uveden níže.
- `gcov-$KERNEL_VERSION-gcov.$RHEL_VERSION.patch` - gcov-kernel patch na aktuálně testovaný kernel. Jedná se o upravenou verzi na RHEL kernely. Novější verze testu jej stahuje přímo z internetu.
- `gensum.c` - Zdrojové kódy aplikace GenSum, ve zdrojové podobě je zde obsažen kvůli multiplatformnosti. Nutnost kompilace nezhoršuje celkový čas chodu.
- `kernel-$KERNEL_VERSION.$RHEL_VERSION.src.rpm` - Balík se zdrojovými kódy testovaného jádra. Novější verze testu jej stahuje přímo z internetu
- **Zabalená sada testů** - spustitelné testy, v původním záměru se jednalo o balík LTP[3]. Jeho spuštění probíhá ze skriptu `runtest.sh`.

3.2.2 Popis kódu

Jak je již uvedeno výše, celý proces testování je řízen BASH **scriptem, který je v plném rozsahu na přiloženém CD. Pro jeho pochopení jsou zde vysvětleny použité vytvořené funkce.**

Kód je doplněn o komentáře vždy pod řádkou s komentovaným kódem. Komentář začíná `'#-#'`.

[2]CVS - Concurrent Version System - systém sloužící ke správě verzí projektu

[3]LTP - Linux Test Project - projekt zabývající se testováním stability linuxu ze všech možných hledisek

Proměnné a prostředí

Před zahájením je nastaveno prostředí a používané proměnné:

```
. /usr/bin/rhts_environment.sh
#-# Nastaví prostředí dle RHTS, přidá několik nutných funkcí

if [ -z ,,$OUTPUTDIR‘‘ ]; then
        OUTPUTDIR=/mnt/testarea
fi
#-# Nastaví defaulntí výstupní adresář, pokud již není učiněno dříve

# DEBUG: control, if debugging information will be stored
DEBUG=1
if [ $DEBUG ]; then
    DEBUGLOG=$(mktemp /tmp/tmp.XXXXXX)
    lck=$OUTPUTDIR/$(basename $0).lck
fi
#-# Obohatí výstup o více hlášek

http_base=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#-# Adresa, odkud se stahují pomocné soubory (z důvodu bezpečnosti zakryto)
KERNELV=2.6.18-gcov.el5
#-# Aktuální testovací verze
UNAME=$(uname -i)
#-# Architektura stroje běžící tento test

# If TESTARGS are not defined, use the installed version-release
if [ -z $MYKERNEL ]; then
    KERNSRC=kernel-2.6.18-8.1.1.el5
else
    KERNSRC=$MYKERNEL
fi
#-# Pokud není definována verze jádra, definuje defaultní
#-# (V této verzi ještě nevyužito, jedná se o přípravu do kCOV v2)
```

Main

Funkce celého projektu je rozdělena do několika částí. První z nich se spouští následující část (pomyslná funkce Main).

```
cd $OUTPUTDIR
DeBug ,,cd $OUTPUTDIR‘‘

if [ ,,$REBOOTCOUNT‘‘ != ,,1‘‘ ]; then
    #-# První průchod, před restartem počítače (NASTAVENí)
    RunPart ,,Prepare‘‘
    #-# Nainstaluje potřebné utility, opatchuje jádro a nastaví
    #-# .config kernelu
    RunPart ,,BuildKernel‘‘
    #-# Zkompiluje jádro, nainstaluje jej a nastaví jako defaultní
    DeBug ,,rhts-reboot‘‘
    rhts-reboot
    #-# Bezpečně restartuje počítač a zaručí spuštění testu opět
    #-# po startu
    DeBug ,,!!! Should never get there !!!‘‘
else
    #-# Druhý průchod, po restartu počítače (TEST A SBĚR DAT)
    RprtRslt $TEST/Reboot PASS 0
    #-# Oznámí výsledkovému serveru, že úspěšně skončila první
    #-# část testování
    RunPart ,,LTP‘‘
    #-# Spustí LTP test, případně jinou sadu testů
    RunPart ,,CaptureData‘‘
    #-# Získá informace o pokrytí kódu aktuálního jádra
    SubmitLog kernel.info
    #-# Odešle soubor ,,kernel.info‘‘ na výsledkový server
    DeBug ,,$(/mnt/tests$TEST/gensum kernel.info)‘‘
    #-# Spustí program GenSum a výsledek předá do logu jobu
    SubmitLog $DEBUGLOG
    #-# Odešle log na výsledkový server
    RprtRslt $TEST PASS $REBOOTCOUNT
    #-# Oznámí konec testování serveru, jako skóre udá počet
    #-# restartů
    exit $RESULT
fi
```

DeBug

Pomocná funkce obohacující test o možnost přesných výstupních informací. Výstup je upraven na formát `,,$DATUM-$ČAS : $ŘÁDKA VÝSTUPU‘‘`. Přídavnou funcí je i blokování souběžného přístupu dvou zapisovatelů pomocí *lockfile*.

```
function DeBug ()
{
    if [ $DEBUG ]; then
    #-# Pokud není zaplý DEBUG mód, nic nedělej
        lockfile -r 1 $lck
        #-# Vstup do zalockované sekce
        if [ ,,$?‘‘ = ,,0‘‘ ]; then
            echo -n ,,$(date +%Y%m%d-%H%M%S) : ‘‘ >> $DEBUGLOG 2>&1
            echo ,,$1‘‘ >> $DEBUGLOG 2>&1
            #-# Zapiš zprávu v nastaveném tvaru
            rm -f $lck > /dev/null 2>&1
            #-# Odstraň zámek
        fi
    fi
}
```

Prepare

Příprava systému na instalaci nového jádra. Tento script stáhne zdrojové kódy jádra, GCOV-KERNEL patch a balíček programu LCOV. Následně rozbalí zdrojové kódy, aplikuje patch a nastaví překlad (`.config`) jádra.

```
function Prepare ()
{
    NecessarlyWork ,,rpm -Uvh $http_base/$KERNSRC.src.rpm‘‘\
 || return $?
    #-# Rozbalí balíček zdrojových kódů jádra
    # --nodeps dependences are necessarily for genhtml...,
    # not for lcov
    NecessarlyWork ,,rpm -Uvh --nodeps $http_base/lcov.el5.rpm‘‘\
 || return $?
    #-# Nainstaluje aplikaci lcov, ta není obsažena v balíčkách RHELu
    NecessarlyWork ,,rpmbuild -bp --target=$UNAME /usr/src/redhat/\
SPECS/kernel-2.6.spec‘‘ || return $?
    #-# Sestaví zdrojové kódy jádra
```

```
    DeBug ,, --- < gcov patch > ---‘‘
    cd /usr/src/redhat/BUILD/kernel-2.6.18/linux-2.6.18.$UNAME/
    #-# Změní adresář na zdrojové kódy jádra
    # In this version s390x did not have configured kernel
#   if [ $UNAME == ,,s390x‘‘ -o $UNAME == ,,s390‘‘ -o $UNAME == ,,ppc‘‘\
 -o $UNAME == ,,ppc64‘‘ ]; then
#       DeBug ,,s390/ppc .config patch‘‘
#       rm -f .config
#       cp -f configs/kernel-2.6.18-$UNAME.config .config
#   fi
#-# Zakomentováno, jednalo se o problémy špatných verzí zdrojových kódů
    # Download and apply gcov-$KERNELSRC patch
    wget -O $OUTPUTDIR/gcov-$KERNSRC.patch $http_base/\
gcov-$KERNSRC.patch
    #-# Stáhne gcov-kernel patch
    NecessarlyWork ,,patch -p1 < $OUTPUTDIR/gcov-$KERNSRC.patch‘‘ \
 || return $?
    #-# Aplikuje patch na jádro
    echo ,,CONFIG_GCOV_PROFILE=y‘‘ >> .config
    # I could not profile entire kernel on x86_64 (SEGFAULT in arch/)
    if [ $UNAME == ,,x86_64‘‘ ]; then
        echo ,,# CONFIG_GCOV_ALL is not set‘‘ >> .config
        for DIRS in block crypto drivers fs init ipc kernel lib mm \
net scripts security sound usr; do
            sed -i '1i\CFLAGS += $(GCOV_FLAGS)' $DIRS/Makefile
        done
    else
        echo ,,CONFIG_GCOV_ALL=y‘‘ >> .config
    fi
    echo ,,CONFIG_GCOV_PROC=y‘‘ >> .config
    echo ,,# CONFIG_GCOV_HAMMER is not set‘‘ >> .config
    #-# Nastaví profilaci GCOV
    return 0
}
```

BuildKernel

Zkompiluje jádro s podporou profilace a nastaví jej jako defaultní.

```
function BuildKernel ()
```

```
{
    ARCH_RHTS=$ARCH
    unset ARCH
    #-# RHTS a Makefile jádra používají stejný název proměnné, ovšem
    #-# u některých architektur s jinou hodnotou. Proto je nutné
    #-# tuto proměnnou uchovat a dočasně přenastavit!
    J=$(expr 1 + $(cat /proc/cpuinfo | grep -c processor))
    #-# Pro urychlení kompilace využíjeme všechny procesory
    make nonint_oldconfig
    #-# Opraví prípadné nesrovnalosti v .config souboru
    NecessarlyWork ,,make -j $J V=1‘‘ || return $?
    #-# Přeloží jádro
    NecessarlyWork ,,make -j $J V=1 modules_install‘‘ || return $?
    #-# Nainstaluje jaderné moduly
    if [ $UNAME != ,,ppc‘‘ -a $UNAME != ,,ppc64‘‘ ]; then
        NecessarlyWork ,,make -j $J V=1 install‘‘ || return $?
    fi
    #-# Pokud se nejedná o powerPC, nainstaluje jádro - ppc nemá
    #-# instalační script, proto je instalace provedena níže
    ARCH=$ARCH_RHTS
    #-# Obnovení RHTS proměnné

    # make $KERNELV default kernel
    CONFIGURED=1
    #-# Proměnná použitá pro detekci úspěšnosti nastavení
    if [ -x /sbin/grubby ]; then
        grubby --set-default=/boot/vmlinuz-$KERNELV
        CONFIGURED=$?
        DeBug ,,x86/s390x - GRUB bootloader ... $CONFIGURED‘‘
    fi
    #-# Pokud stroj používá grub, použije jej pro nastavení

    if [ $UNAME == ,,ppc64‘‘ -o $UNAME == ,,ppc‘‘ ]; then
        DeBug ,,ppc64 - yaboot bootloader‘‘
        cp -f vmlinux.strip /boot/vmlinuz-$KERNELV
        tar -zcf Module.symvers.gz Module.symvers
        cp -f Module.symvers.gz /boot/symvers-$KERNELV.gz
        cp -f .config /boot/config-$KERNELV
        cp -f System.map /boot/System.map-2.6.18-gcov.el5
```

```
        /sbin/new-kernel-pkg --mkinitrd --depmod --install $KERNELV \
--make-default
        CONFIGURED=$?
        if [ -x /sbin/weak-modules ]; then
            /sbin/weak-modules --add-kernel $KERNELV
        fi
        # label=linux is default boot
        sed -i -e 's/label=linux/label=lin_stable/g' -e 's/label='\
$KERNELV'/label=linux/g' /boot/etc/yaboot.conf
    fi
    #-# Jedná se o PowerPC, nainstaluje jádro a nastaví jej jako
    #-# defaultní

    if [ $UNAME == ,,ia64`` ]; then
        DeBug ,,ia64 - efi bootloader``
        cp -f vmlinux.gz /boot/efi/efi/redhat/vmlinuz-$KERNELV
        /sbin/new-kernel-pkg --mkinitrd --depmod --install $KERNELV \
--make-default
        CONFIGURED=$?
    fi
    #-# Jedná se o Intel Ithanium, nastaví efi zavaděč

    if [ -x /sbin/zipl ]; then
        DeBug ,,s390x - zipl bootloader``
        /sbin/zipl
        CONFIGURED=$?
        DeBug ,,s390x - zipl bootloader ... $CONFIGURED``
    fi
    #-# Jedná se o s390, použije zipl jako zavaděč

    DeBug ,,CONFIGURED = $?``
    if [ ,,$CONFIGURED`` -eq ,,0`` ]; then
        return 0
    else
        DeBug ,,No bootloader founded``
        DeBug ,,$(ls -al /boot)``
        DeBug ,,$(ls -al /etc)``
        return 1
    fi
```

```
    #-# Pokud se povedlo nastavit, vrátí 0, v opačném případě vypíše
    #-# chybu, obsahy adresářů /boot, /etc a ukončí celý job.
}
```

CaptureData

Získá informace o pokrytí kódu a uloží je do $OUTPUTDIR/kernel.info. V závislosti na dalším nastavení je tento soubor dále zpracován.

```
function CaptureData ()
{
    NecessarlyWork ,,lcov -c -o $OUTPUTDIR/kernel.info‘‘ || return $?
    return 0
}
```

RprtRslt

Pouze wrapper pro odeslání logů a výsledků na server.

SubmitLog

Pouze wrapper volající fci pro odeslání logu na výsledkový server.

RunPart

Wrapper spouštějící parametrem definovanou úlohu. V závislosti na návratové hodnotě úlohy zašle zprávu na výsledkový server „FAIL“ či „PASS“. Pokud část skončí s chybou, ukončí testování.

Tato funkce umožnila strukturializaci skriptu a tím výrazné zpřehlednění kódu.

```
function RunPart ()
{
    DeBug ,, --- < $1 > ---‘‘
    #-# Zapíše do logu, že spouští úlohu. Tím oddělí jednotlivé části
    eval $1
    #-# Spustí úlohu definovanou prvním parametrem
    RESULT=$?
    #-# Uloží si návratovou hodnotu úlohy
    DeBug ,,  === < $1 - $RESULT > ===‘‘
    #-# Označí ukončení úlohy a její návratovou hodnotu
    if [ ,,$RESULT‘‘ -ne ,,0‘‘ ]; then
        RprtRslt $TEST/$1 FAIL $RESULT
```

```
        exit 1
    else
        RprtRslt $TEST/$1 PASS $RESULT
    fi
    #-# Odešle výsledky, a pokud vše proběhne úspěšně, pokračuje dál
    return 0
}
```

NecessarlyWork

Wraper, který označuje nezbytnou úlohu. Provede úlohu definovanou parametrem a zkontroluje návratovou hodnotu. Pokud vše proběhlo v pořádku, zaznamená do logu tuto událost. Pokud došlo k chybě, pokusí se pomocí STRACE získat co nejvíce informací o chybě a zalogovat je. Dále přidá informaci o přesném prováděném příkazu, návratové hodnotě, velikosti místa na discích, aktuální adresář včetně výpisu souboru, výpis `$OUTPUTDIR` adresáře a nastavení prostředí. Následně skončí s návratovou hodnotou 1.

Návratová hodnota 1 s sebou nese další důležitou věc, a to reakci ve volající funkci. Aby toto jednání mělo nějaký smysl, je nutné správně zareagovat při nenulové návratové hodnotě. Bylo možné ukončit testování přímo uvnitř této funkce, ale je mnohem univerzálnější toto neučinit. V některých případech je zapotřebí větvit proces na základě nezbytných úloh a i v těchto případech potřebujeme vědět, proč běh selhal. Proto bylo zvoleno logování s následnou návratovou hodnotou. Volání funkce pak můžeme provést následovně:

```
NecessarlyWork ,,make -j $J V=1‘‘ || return $?
#-# NecessarlyWork ,,$ULOHA $PARAMETRY_ULOHY‘‘ || return $?
```

Stavba zavolat funkci NecessarlyWork s parametrem zaručí spuštění úlohy. To co je uvedeno za || se spustí pouze, pokud je návratová hodnota jiná, než 0. `return $?` vyskočí z právě prováděné funkce s návratovou hodnotou `$?`. `$?` je použito proto, že v budoucnu může být funkce NecessarlyWork rozšířena o jiné návratové hodnoty, jež mohou být potřeba v nadřazené funkci.

```
function NecessarlyWork ()
{
    eval $1
    #-# Spustí příkaz
    RES=$?
    #-# Uloží si návratovou hodnotu
    if [ ,,$RES‘‘ -ne ,,0‘‘ ]; then
```

```
        #-# Pokud nastala chyba
        STRACE_TMP=$(mktemp /tmp/strace.XXXX)
        strace -o $STRACE_TMP -f $1
        SubmitLog $STRACE_TMP
        #-# Spustí strace a výstup odešle na výsledkový server
        DeBug ,,-------------------------------------------‘‘
        DeBug ,,CMD: $1 - RES: $RES‘‘
        DeBug ,,-------------------------------------------‘‘
        DeBug ,,df: $(df)‘‘
        DeBug ,,-------------------------------------------‘‘
        DeBug ,,PWD: $(pwd)‘‘
        DeBug ,,$(ls -al)‘‘
        DeBug ,,-------------------------------------------‘‘
        DeBug ,,OUTPUTDIR: $OUTPUTDIR‘‘
        DeBug ,,$(ls -al $OUTPUTDIR)‘‘
        DeBug ,,-------------------------------------------‘‘
        DeBug ,,ENV:‘‘
        DeBug ,,$(set)‘‘
        DeBug ,,-------------------------------------------‘‘
        #-# Do logu přidá výše zmíněné informace
      return 1
    else
        #-# Pokud nenastala chyba
        DeBug ,,$1 DONE‘‘
        #-# Zaloguje úspěšné splnění
        return 0
    fi
}
```

LTP

Libovolný BASH script, který spustí sadu testů.

3.3 kCOV v2

3.3.1 Obecný popis

Návrh kCOV v1 byl vytvořen výhradně pro potřeby certifikace, z čehož vyplývají jeho silné nedostatky. Na prezentaci na Developer Conferenci 2007 byl o tuto

utilitu veliký zájem. Byl podán nový náhled na celý problém. Proto byla navázána komunikace se zástupci z jednotlivých odvětví. Jejich požadavky se týkaly hlavně jednoduchosti ovládání, možnosti přímo přistupovat k cílovému počítači, možnosti rychlé změny testovací sady, jednodušší použití jiné verze jádra, či použití vlastního modifikovaného jádra.

Tyto požadavky se podařilo uspokojit. Nutností bylo rozdělit kCOV projekt do dvou testů, využít nových - i kvůli tomuto projektu - přidaných funkcí RHTS a vytvoření vlastního workflow.

Obecný diagram uvedený na začátku je platný i zde, 3.1 odpovídá `/kernel/kCOV/install` a 3.2 odpovídá `/kernel/kCOV/collect`

Workflow definuje chod jednotlivých testů a umožňuje přidávat další. Základní chod vypadá následovně:

1. `/distribution/install` - nainstaluje distribuci
2. `/kernel/kCOV/install` - nastaví jádro pro profilaci
3. `/kernel/kCOV/collect` - sebere informace o pokrytí kódu

Mezi `/kernel/kCOV/install` a `/kernel/kCOV/collect` lze vložit libovolný počet testů z RHTS. Více informací viz sekce A Použití a možnosti kCOV-workflow v přílohách.

Pro větší flexibilitu testu byla přidána čtveřice volitelných parametrů, které pokryly většinu potřeb vývojářů i testerů. Jedná se o tyto parametry:

- MYKERNEL - jakou verzi jádra použít {x.y.z-ver.sysver} (defaultně: běžící)
- GCOV_ALL - profilovat celé jádro {1,0} (defaultně: 1)
- GENINFO - zalogovat soubor .info - umožňuje pozdější detailní rozbor {1,0} (defaultně: 1)
- GENSUM - generovat souhrnné pokrytí kódu pomocí GenSum {1,0} (defaultně: 1)

3.3.2 Popis kódu - kCOV install

Stáhne zdrojové kódy jádra a k němu odpovídající patch. Potom aplikuje patch, zkompiluje jádro, nastaví jako aktuální a restartuje počítač. Úspěšným startem počítače je test ukončen a začíná v pořadí další naplánovaný test spuštěný nad jádrem s podporou profilace. Oproti předchozí verzi zavádí nové proměnné do prostředí, jimiž je možné podrobněji specifikovat požadavky. Jedná se o sekci `TEST params`. Dále pak vylepšenou

inteligenci volby jaderných patchů, takže není potřeba pro každé jádro dělat nový patch. Poslední změnou je systém restartu, kde po opětovném nastartování RHTS úloha pouze reportuje výsledky.

Nezměněné sekce zde nejsou podrobně rozebrány, celý kód je na přiloženém CD.

Kód je doplněn o komentáře vždy pod řádkou s komentovaným kódem. Komentář začíná '#-#'.

Main

Main se od předchozí verze liší pouze v tom, že po restartu jen reportuje výsledky a skončí.

TEST params

Tato sekce není přímo funkcí, poze je to sekce oddělená komentářem a je umístěna nad sekcí Main a tak je provedena před začátkem hlavního těla.

```
#########################################################################
# TEST params
#########################################################################
export brew_base=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#-# Cesta k brew, kde jsou umístěny zdrojové kódy přeložených jader
export http_base=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
#-# Cesta k aktuálním kCOV patchům
export UNAME=$(uname -i)
#-# Aktuální architektura

# MYKERNEL  2.6.18-8.1.1.el5
# $(uname -r)  2.6.18-8.1.8.el5
# Specific kernel release, or just running kernel
if [ -z $MYKERNEL ]; then
    DeBug ,,no MYKERNEL specified, running KernelVersion $(uname -r)‘‘
    WgetPatch ,,$(uname -r)‘‘
else
    DeBug ,,MYKERNEL $MYKERNEL, running KernelVersion $MYKERNEL‘‘
    WgetPatch ,,$MYKERNEL‘‘
fi
```

```
#-# Zvolení verze kernelu, pro který bude coverage prováděna.
#-# Testuji, jestli není proměnná již definována. Definice může být
#-# při zadání parametru ve workflow.
#-# Pokud není zadána verze, použije aktuální
#-# Zároveň je pak stáhnut patch pro zvolené jádro
#-# Funkce WgetPatch je vysvetlena níže

# KERNELV=2.6.18-gcov
export KERNELV=$(echo $KERNSRC | cut -d- -f2),,-gcov‘‘
#-# Proměnná závislá na vybraném kernelu

# GCOV_ALL y
# Do I want to profile entire kernel?
if [ -z $GCOV_ALL ]; then
    export GCOV_ALL=’y’
else
    if [ $GCOV_ALL != ’n’ ]; then
        export GCOV_ALL=’y’
    fi
fi
#-# Udává, jestli se bude profilovat celé jádro, nebo jen části
#-# Defaultně se profiluje celé
```

WgetPatch

Tato funkce detekuje patch pro vybrané jádro a stáhne jej. Nejprve se pokusí najít verzi přímo pro zvolené jádro, pokud patch není dostupný, vybere patch určený pro větev jádra (např. pokud není patch pro jádro `2.6.18-53.el5`, zkusí `2.6.18`. Pokud není dostupná ani tato verze, skončí s chybou.

```
function WgetPatch ()
{
    # KERNSRC=kernel-2.6.18-53.el5
    export KERNSRC=kernel-$1
    DeBug ,,Trying wget --spider $http_base/gcov-$KERNSRC.patch‘‘
    wget -O $OUTPUTDIR/gcov-$KERNSRC.patch $http_base/\
gcov-$KERNSRC.patch
    if [ $? -ne 0 ]; then
        NecessarlyWork ,,wget -O $OUTPUTDIR/gcov-$KERNSRC.patch \
 $http_base/gcov-$(echo $KERNSRC | cut -d- -f2).patch‘‘ || return $?
```

```
    fi
    return 0
}
```

3.3.3 Popis kódu - kCOV collect

Nahraje jaderný modul gcov-proc a spustí sběr pokrytí kódu. Následně reportuje výsledky. Celý test je opět opatřen volitelnými parametry, které se nastavují v sekci `TEST params`.

Nezměněné sekce zde nejsou podrobně rozebrány, celý kód je na přiloženém CD.

Kód je doplněn o komentáře vždy pod řádkou s komentovaným kódem. Komentář začíná `'#-#'`.

Main

Main je oproti předchozí verzi kompletně přepsán. Opět pouze řídí chod a na závěr zapíše výsledky.

```
################################################################################
# Main
################################################################################
modprobe gcov-proc
#-# Naloaduje modul gcov-proc. Není to nutné a při klasickém použití
#-# kCOV selže, neboť gcov-proc je zakompilován přímo do jádra. Je
#-# zde pro případ, že někdo nastaví své jádro a využije jen tuto
#-# (collect) část testu.
if [ $(uname -r | grep gcov) -a -d /proc/gcov ]; then
    cd $OUTPUTDIR
    DeBug ,,cd $OUTPUTDIR‘‘

    RunPart ,,CaptureData‘‘
    #-# Spustí sběr dat

    if [ $GENSUM -ne 0 ]; then
        DeBug ,,$(/mnt/tests$TEST/gensum kernel.info)‘‘
    fi
    #-# Pokud je zapnutá volba GENSUM, spustí program GenSum a do logu
```

```
    #-# přidá informace o celkovém pokrytí kódu.

    if [ $GENINFO -ne 0 ]; then
        SubmitLog kernel.info
    fi
    #-# Pokud je zaplá volba GENINFO, připojí soubor kernel.info
    SubmitLog $DEBUGLOG
    RprtRslt $TEST PASS 0
    exit 0
else
    DeBug ,,This is not gcov kernel‘‘
    DeBug ,,You have to run /kernel/kCOV/install before this test‘‘
    SubmitLog $DEBUGLOG
    RprtRslt $TEST FAIL 0
    exit 1
fi
#-# Pokud je aktuální kernel profilován a je vytvořena struktura
#-# /proc/gcov, zjistí pokrytí kódu. V opačném případě zareportuje
#-# že se nejedná o správné jádro a ukončí úlohu s chybou.
```

TEST params

Tato sekce není přímo funkcí, pouze je to sekce oddělená komentářem a je umístěna nad sekcí Main, a tak je provedena před začátkem hlavního těla.

```
################################################################################
# TEST params
################################################################################
if [ -z $GENINFO]; then
    GENINFO=1
else
    if [ $GENINFO -ne 0 ]; then
        GENINFO=1
    fi
fi
#-# Kdyz není definována proměnná GENINFO, zapne ji. Pokud je
#-# definována, normalizuje její hodnotu

if [ -z $GENSUM ]; then
    GENSUM=1
```

```
else
    if [ $GENSUM -ne 0 ]; then
        GENSUM=1
    fi
fi
#-# Když není definována proměnná GENSUM, zapne ji. Pokud je
#-# definována, normalizuje její hodnotu
```

3.4 kCOV workflow

Konkrétní komunikace v rámci RHTS workflow podléhá firemnímu tajemství.

Workflow slouží k přímé komunikaci se RHTS schedulerem. Formou scriptu umožňujě vytvořit job a naplánovat jej. Tento způsob umožňuje zadat rozšířené parametry testu, předpřipravit určitý sled či naplánovat větší počet testů.

Bez použití workflow je stále možné spustit kCOV test za použití webového rozhraní RHTS. Tímto způsobem prozatím není možno zadat rozšiřující parametry.

4 VÝSLEDKY

4.1 Použití

Nová verze kCOV rozšířila možnosti použití. Na počátku se uvažovalo pouze o využití k certifikaci jader operačního systému RHEL. Díky volitelným parametrům, flexibilnějšímu stylu vkládání vlastních testů či dokonce i možnosti manuálního zásahu při běhu profilovaného jádra se využití podstatně rozšířilo. V dalších částech se podrobněji věnuji standardnímu použití třech základních typů použití.

4.2 Výstup

Podle požadavků existují dvě základní formy výstupu pokrytí kódu. Původní grafická verze generovaná programem lcov, nebo prostá řádka s trojicí souhrnných informací o celkovém pokrytí.

4.2.1 Grafický výstup - lcov

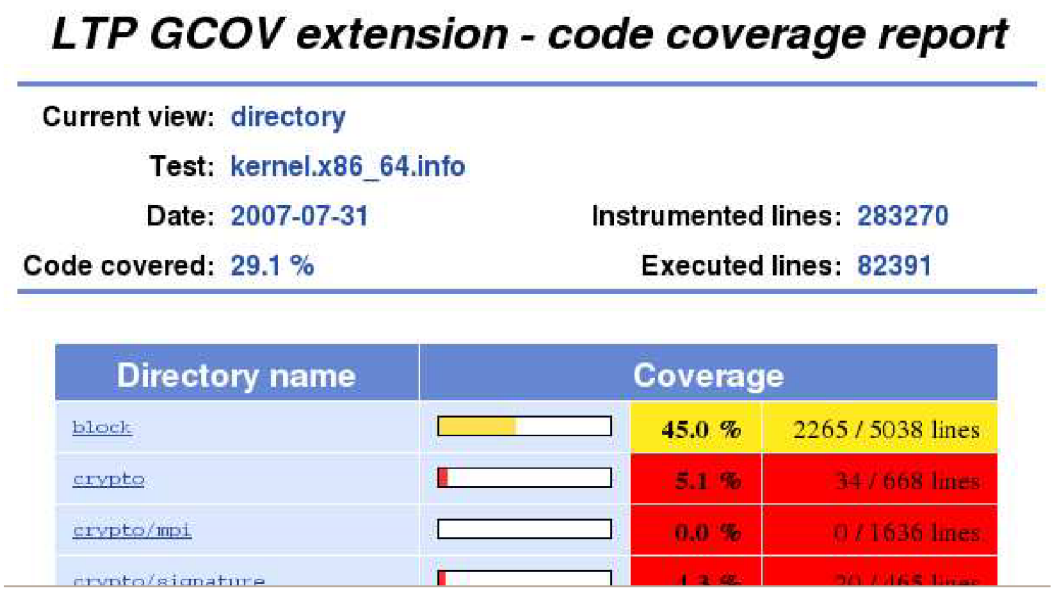
Přístup ke grafické verzi výstupu se zapne pomocí volby `GENINFO` při zadávání jobu. Tím se zahrne soubor `kernel.info`, obsahující informace o pokrytí kódu jádra, do výstupů na výsledkový server. Tento soubor ovšem nezobrazuje data v lidsky přívětivém formátu, a proto je nutné tento soubor dále zpracovat. Pro toto zpracování je nutné mít stroj architektury `x86` (`i386`, nebo `x86_64`), balíček lcov s plnými závislostmi a zdrojové kódy jádra ve verzi, ve které byly na testovaném stroji. Program lcov následně vygeneruje hypertextově provázané stránky se statistikami. Stránky obsahují, jak je možno vidět z obrázků, souhrnný adresářový náhled (4.1), i podrobný souborový výstup (4.2).

Adresářová struktura

V horní části obrazovky je hlavička, která obsahuje základní informace o umístění - v jaké adresářové struktuře se zrovna nacházíme, název testu, datum, počty celkových a vykonaných řádek kódu a procentzální pokrytí kódu. Hlavička i informace o pokrytí se vztahují pouze na všechny podadresáře. Pokud chceme celkové pokrytí projektu, je nutno se podívat na kořenový adresář.

V hlavní části jsou položky s adresáři a soubory obsaženými v aktuálním umístení a ke každému z nich souhrnné informace o pokrytí a počtech řádků.

Pro větší přehlednost jsou jednotlivé řádky obarveny dle 3 hranic, tyto hranice jsou nastavitelné a odlišují kód málo, průměrně či dobře pokrytý.



Obrázek 4.1: lcov - adresářová struktura

Souborová struktura

Horní část obrazovky je opět vyhrazena pro hlavičku, ta se vztahuje pouze na aktuální soubor.

Hlavní část obrazovky (středový rámec) obsahuje zdrojový kód obohacený, podobně jako při prostém použití gcovu, o informace o pokrytí kódu. První sloupec udává řádku v kódu, druhý počet spuštění aktuální řádky a poslední velká kolonka obsahuje přímo text kódu.

Oproti adresářové struktuře zde může být zaveden třetí rámec, který obsahuje obrázek - náhled - kódu. Tato volba je užitečná, neboť nám poskytne jakýsi náhled na zdrojový kód a jsme schopni i u rozsáhlého souboru rychle najít nepokrytá místa.

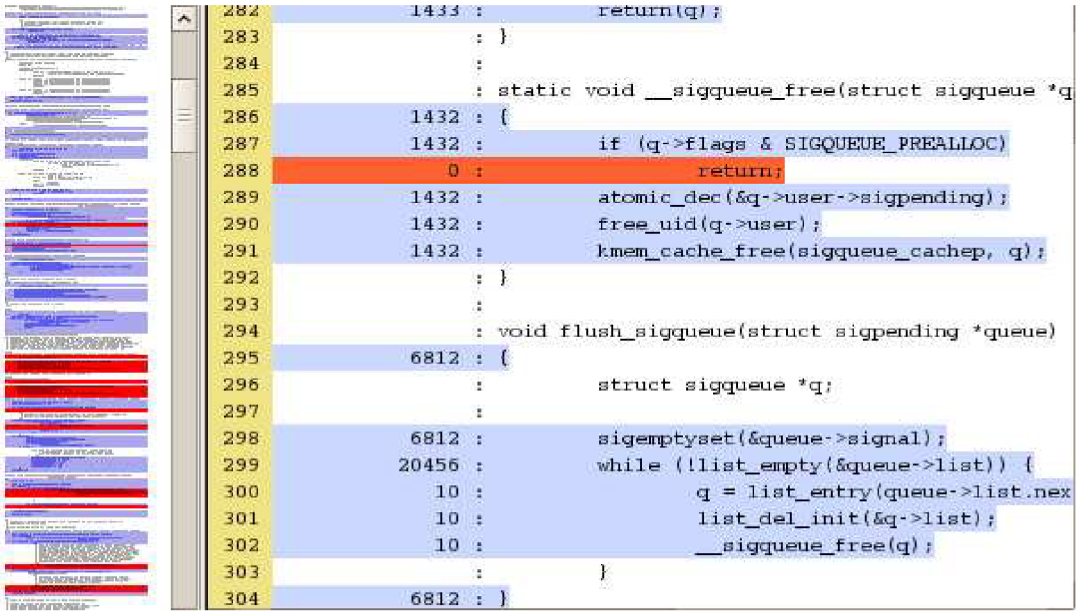
Opět zde platí členění do tří skupin pokrytí oddělených barvami.

4.2.2 Textový výstup - GenSum

Textový výstup pomocí miniprogramu GenSum nám umožní přímo v průběhu testu dostat na výstup souhrnné informace o pokrytí kódu. Výstup pak bude zahrnut do logu testu a řádek vypadá takto:

```
20071109-111304 : 293666 / 73085 = 24.887117
#-# $DATUM-$ČAS : $ŘÁDEK / $VYKONANÝCH = $PROCENTUÁLNÍ_POKRYTÍ
```

Hlavní výhodou je to, že GenSum je multiplatformní, a tak jej lze spustit na cílovém stroji. Z toho vyplývá i to, že na rozdíl od lcovu nepotřebuje žádný zásah. Druhou výhodou je samotná rychlost vykonávání, GenSum je 30x rychlejší.



Obrázek 4.2: lcov - souborová struktura

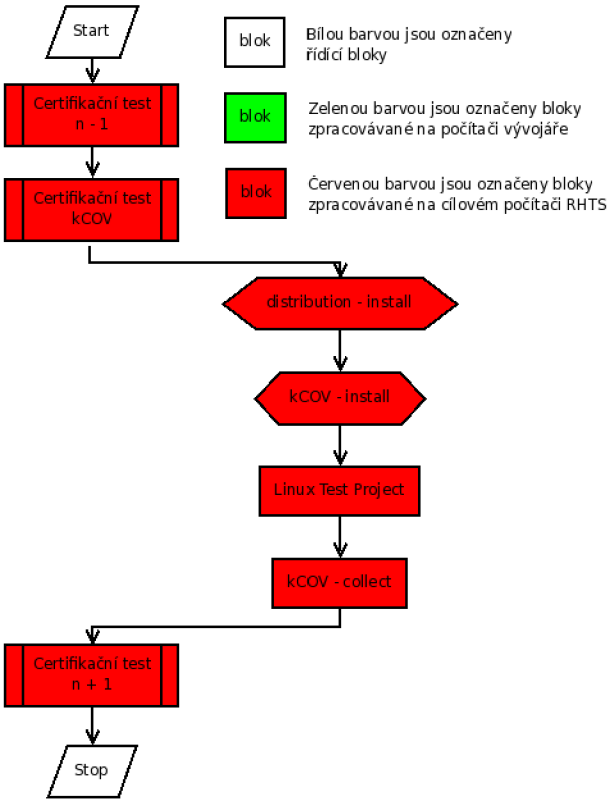
4.3 Standardní použití

V následujících sekcích je vysvětleno použití z pohledu tří základních stran.

4.3.1 Certifikace

Toto použití (detail viz obrázek 4.3) bylo primárním, a proto je totožné s funkcí kCOV v1. Je určeno pro předem definované, pro každou verzi stejné automatické testování před vydáním distribuce. Jeho cílem není odhalit chyby, ale celkově otestovat již stabilní jádro. Procentuální pokrytí kódu pak ukazuje, jak široce zaměřená certifikace byla. Zde se výhradně používá řádková forma pokrytí kódu (Statement coverage). K testování se používá výhradně sada testů Linux Test Project. Jako forma výstupu postačuje GenSum a pokrytí se běžně pohybuje kolem 25 - 30%. Mohlo by se zdát, že testování je velmi slabé, nicméně jedná se o procentuální pokrytí celého jádra včetně ošetření chybových stavů, všech driverů a architektur. Právě ošetření chybových podmínek je globálně nerealizovatelné[1] a v rámci předchozího testování by bylo pouze zbytečným mrháním procesorového času.

[1]Není realizovatelné otestovat všechny chybové stavy na všech podporovaných ovladačích; například testovat vyndání hot-swap disku za chodu pro všechny drivery, či jiné funkčnosti



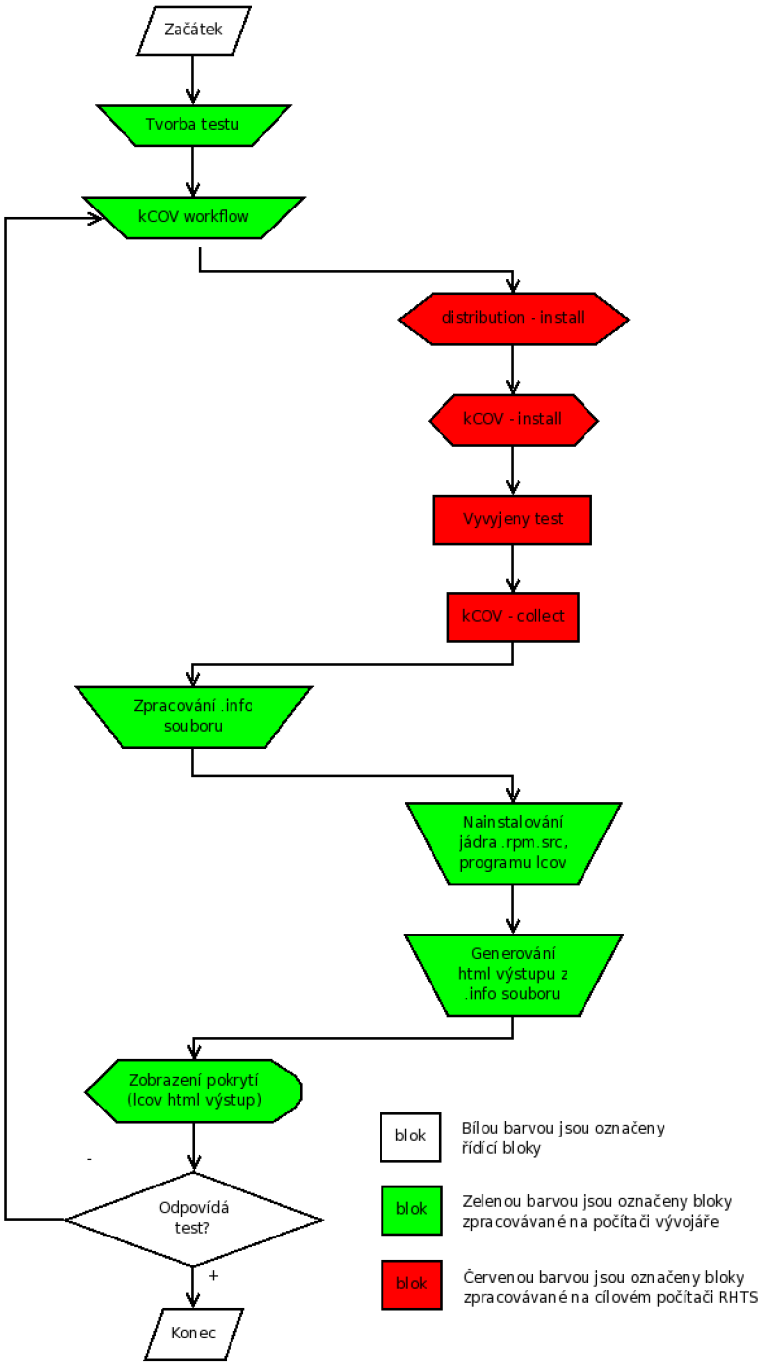
Obrázek 4.3: Proces certifikace

4.3.2 Tester

Využití z pohledu testera se od certifikace značně liší. Jejich práce je většinou ověřit, že určitá funkčnost dříve nebyla a po aplikace patche je v pořádku bez vedlejších účinků. Tyto chyby se týkají malých částí jádra. Často není jednoduché poznat, zda test skutečně dělá činnost, na kterou je navrhován. Proto je šikovné při vytváření testu použít modifikované jádro, kde podle pokrytí lze poznat průběh testu. Testeři většinou plně využijí výstup pomocí programu LCOV. Průběh je naznačen v obrázku 4.4

4.3.3 Vývojář

Činost developera spočívá v tom, že se stará o konkrétní ovladač či komponentu jádra. Když se v té komponeně vyskytne chyba, je zákazníkem vyžadovaná nová funkce, nebo upstream udělá změny, vývojář musí zareagovat a tuto novou funkci zajistit. Hlavní rozdíl mezi testerem a vývojářem je v tom, že tester má k dispozici existující patch. Oproti tomu vývojář tuto funkci teprve zavádí. Práce vývojáře je tak mnohem více manuální, nepotřebuje profilovat celé jádro a potřebuje velice de-



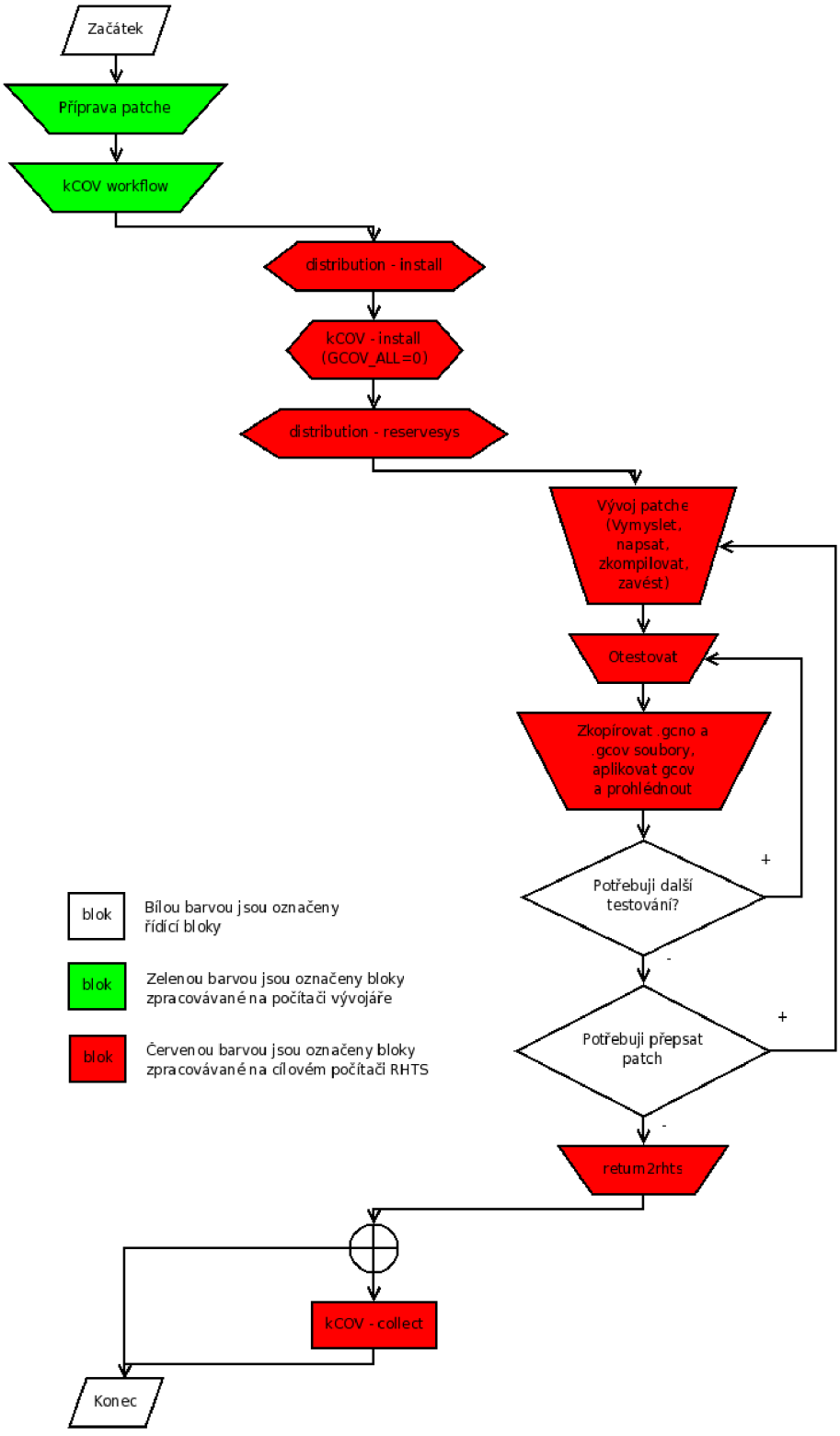
Obrázek 4.4: Proces testera

tailní informace přímo při jednotlivých fázích zkoušení modulu[2]. K poskytnutí těchto informací slouží přímá interakce s cílovým počítačem. Vývojář kopíruje v určitých okamžicích soubor `.gcno` a `.gcda` z adresářové struktury `/proc/gcov/$CESTA_K_MODULU`. Následně je zpracovávají přímo `userspace` aplikací `gcov`. Tímto způsobem je schopen získat všechny druhy pokrytí kódu v rozličných fázích funkce modulu. Často se tato funkce využívá při vývoji ovladačů na hardware, neboť dokáží nahradit častou metodu pomocí `printk`[3].

Postup vývojáře je zobrazen na obrázku 4.5

[2]Většina funcí lze implementovat jako modul, který lze za chodu přidat či odstranit a následně přepracovat a bez restartu opět zavést; pokud vyvýjená komponenta nelze vytvořit jako modul, je nutno pokaždé překompilovat jádro a restartovat počítač.

[3]metoda `printk` spočívá v tom, že do zajimavých sekcí vloží vývojář funkci výtisku na monitor (`printk`) a následně pak sleduje log na přítomnost těchto výpisů.



Obrázek 4.5: Proces vývojáře

5 ZÁVĚR

5.1 Zhodnocení cílů

V průběhu realizace došlo oproti zadání k značnému rozšíření projektu. Šlo hlavně o zájem vývojářů i testerů po prezentaci tohoto projektu na Developer konferenci 2007. Velké změny také umožnilo ukončení vývoje nové verze RHTS v3. Šlo hlavně o změnu funkce plánovače testů, řízení testování a v neposlední řadě i úpravu některých generických testů.

Původní verze kCOV v1 byla určena výhradně pro certifikaci a s manuálním zásahem i k optimalizaci. Implementována je dnes již výhradně na RHEL5.

Nová verze kCOV v2 je kompletně přepsanou verzí, jež uspokojuje požadavky certifikace, výstupní kontroly i vývojářů. V certifikaci bylo nutností upravit workflow tak, aby nebyl spouštěn pouze test kCOV, ale aby spustil kCOV_install, libovolnou sadu testů a následně kCOV_collect. Tato úvodní změna velice zjednodušila a zminimalizovala nutnost manuálního zásahu při vydávání nových verzí.

Celý skript je významově větven tak, že je rychle pochopitelný, přehledný a lehce rozšiřitelný. Svou modulárností udává směr, jakým se budou psát všechny budoucí testy.

REFERENCE

[1] Jonathan Corbet, Alessandro Rubini a Greg Kroah-Hartman *Linux Device Drivers v3,* [online]. Dostupná z URL: <http://lwn.net/Kernel/LDD3>.

[2] Robert Love *Linux Kernel Development - Second Edition,* Novell Press, 2005. 401 s. ISBN 0-672-32720-1

[3] Manoj Iyer a kolektiv *Dokumentace projektu Linux Test Project,* [online]. Dostupná z URL: <http://ltp.sourceforge.net/coverage>.

[4] Developeři GNU/CC *Dokumentace projektu gcc/gcov,* [online]. Dostupná z URL: <http://gcc.gnu.org/onlinedocs/gcc-3.0/gcc.html>.

[5] Developeři projektu Linux Test Project *Dokumentace projektu gcov-kernel,* [online]. Dostupná z URL: <http://ltp.sourceforge.net/coverage/gcov-kernel.readme.php>.

[6] Developeři projektu Linux Test Project *Dokumentace projektu lcov,* [online]. Dostupná z URL: <http://ltp.sourceforge.net/coverage/lcov.readme.php>.

[7] *Dokumentace RHTS,* interní dokument Red Hat, Inc.

[8] Kolektiv autorů 1. vydání Olomouc *Pravidla českého pravopisu,* FIN PUBLISHING, 1998. 575 s. ISBN 80-86002-40-3

SEZNAM POJMŮ A ZKRATEK

RHTS Red Hat Test System – systém umožňující automatické plánování a řízení testů

RHEL Red Hat Enterprise Linux - verze linuxu vyvíjená a podporovaná firmou Red Hat

GNU GNU's Not Unix - GNU Není Unix - projekt zaměřený na svobodný softwre, inspirovaný operačními systémy unixového typu

Linux Linux - jádro operačního systému GNU/Linux; často se používá i jako označení celého operačního systému

GNU/CC The GNU Compiler Collection - zk. GCC - je sada kompilátorů vytvořených v rámci projektu GNU

GCOV Program na zpracování profilačních informací poskytnutých programem GNU/CC

LCOV Sada perlových scriptů umožňující přehledné zobrazení pokrytí kódu rozlehlých projektů

GenSum Program napsaný v jazyce C počítající celkové pokrytí kódu z .info souboru

gcov-kernel Projekt zabývající se zpřístupněním profilačních informací kernelu do userspacu

Test Scheduler (plánovač testů) - komponenta RHTS řídící samotný proces testování a chod počítačových labolatoří

(Individual) Test (jednotlivý test) - komponenta RHTS, je napsána v jazyce srozumitelném Test Scheduleru a umožňuje tak spuštění obsaženého testu na stroji řízeném Schedulerem

Job (práce) - přesně definovaná úloha pro Scheduler, může obsahovat více počítačů, architektur i testů

Workflow (pracovní tok) - skript přesně formující RHTS job

Patch (záplata) - v počítačové oblasti vyjadřuje opravu, nebo také změnu kódu od předcházející verze; aplikací patche dostaneme novou verzi

SEZNAM PŘÍLOH

A POUŽITÍ A MOŽNOSTI KCOV-WORKFLOW

Vytvořený WORKFLOW je doporučeným způsobem zadávání testů. Umožňuje zadat všechny potřebné parametry a naplánovat celý proces přímo z příkazové řádky počítače.

Parametry jsou popsány v nápovědě (parametr `-h`):

```
usage: kCOV-workflow.py [-a i386] [-m STROJ] [-d RHEL5-Server-GOLD]
 -f RedHatEnterpriseLinux5 -u user@redhat.com [-M 2.6.18-8.1.1.el5]
[-A n] [-i 0] [-s 0] [-t /kernel/distribution/ltp/20070731 ]
[-T 3600] [-S SCHEDULER] [-e development] [-X] [-C]

  -h, --help            zobrazí nápovědu a skončí
  -a ARCH(s)            Přidá architekturu k testování.
                        NOTE: žádná znamená všechny.
  -d DISTRO             Použije vybranou distribuci
  -f FAMILY             Přidá rodinu distribucí do požadavků
  -M MYKERNEL, --mykernel=MYKERNEL
                        INSTALL: Nastaví požadovanou verzi jádra.
                        (defaultně použije verzi běžícího jádra)
  -A GCOV_ALL, --gcov_all=GCOV_ALL
                        INSTALL: Profilovat celé jádro? {*y/n}
  -i GENINFO, --geninfo=GENINFO
                        COLLECT: Vrátit soubor .INFO? {*1,0}
                        (Zapnout výstup LCOV)
  -s GENSUM, --gensum=GENSUM
                        COLLECT: Zapnout výstup gensum? {*1,0}
  -u SUBMITTER          Email zadavatele
  -S HOSTNAME           Adresa scheduleru
  -e TEST_REPO          Zahrnout repozitář
  -X                    Zapnout rozšířený výstup
  -C                    Pouze vyzkoušet, neposílat na scheduler
  -m MACHINE            Jméno stroje, na kterém má být úloha naplánována
  -t TESTS, --tests=TESTS
                        Přidej test; Umístění je mezi /kernel/kCOV/install
                        a /kernel/kCOV/collect
  -T TIME, --time=TIME  Rezervuj systém pro manuální použití. Proběhne
                        jako poslední test před /kernel/kCOV/collect.
```

Pro použití `kCOV-workflow` je nutné mít nainstalovány RHTS-HELPERY.

B POSTUP POUŽITÍ PRO TESTERA

Pro jendoduchost a rychlé zaučení testerů je zde uveden většinou používaný postup.

Předpokládá se, že tester vytvořil test, u kterého potřebuje zjistit, zda-li testuje tu část, pro kterou je napsán, případně z jiného důvodu vidět podrobné pokrytí kódu.

Pro správnou funkci je nutné mít nainstalovány následující vývojářské nástroje: `gcc`, `unifdef`, `rpmbuild`, `lcov` a `rhts-devel`[1].

- `#-#` - znamená komentář
- `#` - znamená řádek na příkazové řádce
- `$NAZEV` - proměnná s názvem NÁZEV, obsah je popsán buď v komentáři, je očekávatelný z názvu, nebo je z bezpečnostních důvodů skryt

Vlastní postup

```
#-# Na svém lokálním počítači
# TMP=$ADRESÁŘ_KDE_BUDE_PRACOVAT
# kCOV-workflow.py -a ia64 -d RHEL5-Server-GOLD -u ldoktor@redhat.com \
-t $NÁZEV_TESTU -S $NAZEV_SCHEDULERU
#-# Naplánuje úlohu na stroj itanium, distribuci RHEL5-Server-GOLD...

#-# Po dokončení úlohy tester zkontroluje výstup, jestli pokrytí
#-# v logu (GenSum) odpovídá. Když je vše v pořádku, zkopíruje soubor
#-# kernel.info do $TMP adresáře

#-# Stáhne .src.rpm balíček jádra ve verzi odpovídající verzi použité
#-# při testování
# rpm -ivh $KERNEL.src.rpm
# rpmbuild -bp --target ia64 /usr/src/redhat/SPECS/kernel-2.6.spec
#-# Vytvoří zdrojové kódy jádra totožné s těmi na testovaném stroji

# genhtml -f -o $TMP/html/ $TMP/kernel.info
#-# Vytvoří html stránky s pokrytím kódu

#-# Otevře stránky a zkontroluje výsledek.
```

[1] Postup instalace viz.
`https://wiki.108.redhat.com/wiki/index.php/Testing/Rhts/Docs/TestWriting`

C POSTUP POUŽITÍ PRO VÝVOJÁŘE

Pro jendoduchost a rychlé zaučení vývojářů je zde uveden většinou používaný postup.

Předpokládá se, že vývojář vytvořil patch a potřebuje zkontrolovat, jestli se chová tak, jak předpokládá.

Pro správnou funkci je nutné mít nainstalován `rhts-devel`[1].

- `#-#` - znamená komentář
- `#` - znamená řádek na příkazové řádce
- `$NAZEV` - proměnná s názvem NÁZEV, obsah je popsán buď v komentáři, je očekávatelný z názvu, nebo je z bezpečnostních důvodů skryt

Vlastní postup

```
#-# Na svém lokálním počítači
# kCOV-workflow.py -a s390x -d RHEL5-Server-GOLD -u ldoktor@redhat.com \
-t $NÁZEV_TESTU -S $NAZEV_SCHEDULERU -M $JMENO_JADRA -T 28800
#-# Naplánuje úlohu na stroj system Z, distribuci RHEL5-Server-GOLD...
#-# -M může použít pro zadání jádra, které vybuildil přes BREW, nebo
#-# jen požadovanou verzi, do které následně přidá svůj vyvýjený
#-# modul.
#-# -T 28800 rezervuje systém na 8 hodin.

#-# Po nainstalování počítače a modifikaci jádra pro sběr profilačních
#-# dat příjde vývojáři email potvrzující rezervaci počítače.
#-# Vývojář se přihlásí na cílový stroj
# ssh $TESTOVANÝ_POČÍTAČ

#-# Na testovaném počítači
# TMP=$ADRESÁŘ_KDE_BUDE_PRACOVAT

#-# Vývojář udělá svou práci.
#-# (zkompiluje modul, vloží jej, odstraní,...)

#-# Kdykoliv potřebuje, může pomocí
#-# # echo 0 > /proc/gcov/vmlinux
```

[1] Postup instalace viz.
`https://wiki.108.redhat.com/wiki/index.php/Testing/Rhts/Docs/TestWriting`

```
#-# vynulovat čítače pokrytí

#-# Pokud zkopíruje strukturu, nebo jen zajímavou část, z /proc/gcov
#-# do $TMP adresáře, může pomocí aplikace gcov získat podrobné
#-# pokrytí kódu ve všech kombinacích, co program gcov nabízí.
#-# Je možné tyto soubory zkopírovat v určitých okamžicích
#-# a porovnánvat, stav bude odpovídat stavu zkopírovaných souborů,
#-# ne stavu v době spuštění gcov.

# cp -R /proc/gcov $TMP/cas_1
# sleep 10
# cp -R /proc/gcov $TMP/cas_2
# gcov $TMP/cas_[12]/$CESTA_K_MODULU
#-# Získá pokrytí kódu ve 2 okamžicích 10s od sebe vzdálených

# COUNT=0
# while [ $COUNT != 5 ]; do\
>   echo 0 > /proc/gcov/vmlinux
>   cp -R /proc/gcov $TMP/$COUNT
>   COUNT=$(expr $COUNT + 1)
>   sleep 10
> done
# gcov $TMP/[01234]/$CESTA_K_MODULU
#-# Postupně získává pokrytí kódu v různých okamžicích, pokaždé
#-# vynuluje čítače. Toto je jen pár naznačených způsobů,
#-# většinou je nutné přidávat určité eventy (zápis na disk,
#-# vyřazení disku, čtení,...).

#-# Po skončení testování vývojář nemusí řešit žádný úklid,
#-# pouze pokud chce zachovat nějaké výstupy, uloží si je
#-# k sobě. (scp, ftp, nfs, ...)

#-# Nakonec vrátí systém do RHTS
# return2rhts
```

D POSTUP POUŽITÍ PRO CERTIFIKACI

Postup při certifikaci.

Předpokládá se, že se má vydat nová verze systému, nebo jádra.

Jsou dva základní způsoby. Vybrat jednotlivé testy pomocí webového rozhraní RHTS, zaškrtnout všechny architektury, jako druhý test zvolit `/kernel/kCOV/install` a jako poslední `/kernel/kCOV/collect`. Následně stačí vyčkat, než přijde email s výsledkem.

Druhý způsob je poprvé složitější, pro opakované použití velmi jednoduchý. Pro jeho použití musí být nainstalován `rhts-devel`[1].

- `#-#` - znamená komentář
- `#` - znamená řádek na příkazové řádce
- `$NAZEV` - proměnná s názvem NÁZEV, obsah je popsán buď v komentáři, je očekávatelný z názvu, nebo je z bezpečnostních důvodů skryt

Vlastní postup

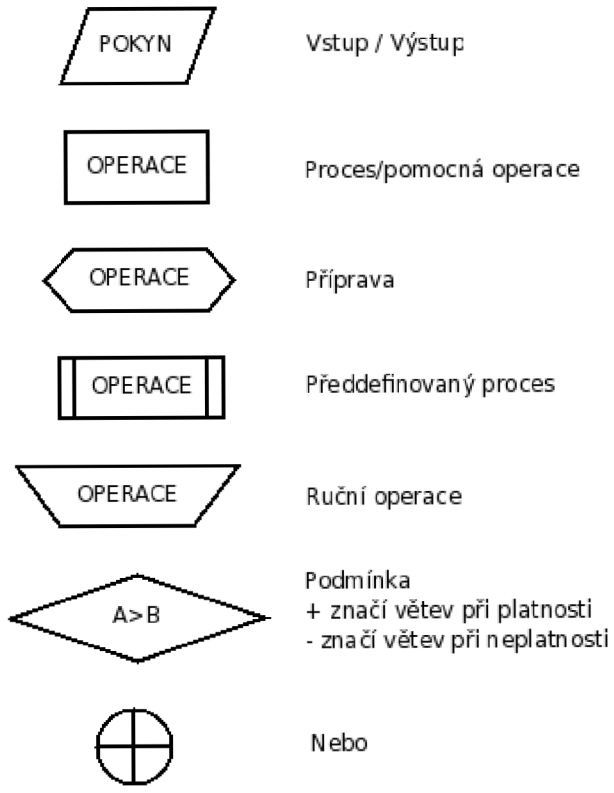
```
#-# Na svém lokálním počítači
# kCOV-workflow.py -d $VERZE_NOVÉHO_SYSTÉMU -u ldoktor@redhat.com \
-t $PRVNI_CERTIFIKAČNÍ_TEST -t $DRUHY_CERTIFIKAČNÍ_TEST \
-t $POSLEDNÍ_CERTIFIKAČNÍ_TEST -S $NAZEV_SCHEDULERU -i 0
#-# Naplánuje úlohu na všechny architektury,
#-# distribuci vybere $VERZE_NOVÉHO_SYSTÉMU.
#-# parametrů -t může být libovolně mnoho, vykonají se postupně
#-# mezi testy /kernel/kCOV/install a /kernel/kCOV/collect.
#-# -i 0 zaručí, že se nebude reportovat .info soubor.

#-# Následně opět člověk čeká na email oznamující konec testování
```

Tento druhý způsob je používaný, protože jej lze zahrnout do plně automatizovaného testování. Testy se vypíší pouze poprvé, příště se změní pouze proměnná `$VERZE_NOVÉHO_SYSTÉMU`.

[1] Postup instalace viz.
`https://wiki.108.redhat.com/wiki/index.php/Testing/Rhts/Docs/TestWriting`

E VYSVĚTLIVKY K VÝVOJOVÝM DIAGRAMŮM



Obrázek E.1: Vysvětlivky k diagramům

F OBSAH CD

- `/ SRC /` - obsahuje zdrojové kódy aplikací
  - `/ kCOV_v1 /` - zdrojové kódy kCOV verze 1 včetně Makefile
  - `/ kCOV_v2 /` - zdrojové kódy kCOV verze 2
    - `/ kCOV_install /` - zdrojové kódy kCOV před restartem
    - `/ kCOV_collect /` - zdrojové kódy kCOV po restartu
  - `/ workflow /` - složka pro kCOV_workflow, v této verzi z licenčních důvodů obsahuje pouze textový výstup nápovědy
  - `/ GenSum /` - složka obsahuje zdrojové kódy k aplikaci GenSum
  - `/ gcov-kernel /` - obsahuje modifikované patche gcov-kernel
- `/ RPM /` - obsahuje neobvyklé potřebné RPM balíčky
  - `/ tmp-kernel-kCOV-collect-1.1-0.noarch.rpm` - kCOV_collect test
  - `/ tmp-kernel-kCOV-install-1.1-0.noarch.rpm` - kCOV_install test
  - `/ rhts-tools-repo-2.6-21.noarch.rpm` - YUM repozitář s nutnými nástroji
  - `/ lcov-1.4-2.fc6.noarch.rpm` - LCOV (Není defaultně obsažen v RHEL)
- `/ DOC /` - obsahuje text této bakalářské práce
- `/ OUT /` - obsahuje výstupy
  - `/ html /` - html výstupy LCOVu pro jednotlivé architektury
  - `/ log /` - logy z RHTS (výstup GenSum)
  - `/ proc /` - kopie souborové struktury `/proc/gcov` vytvořené na profilovaném jádře