Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Komparativní analýza multitaskingových operačních systémů pro embedded aplikace

Plzeň, 2014 Bc. Jiří Beneš

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: **2013/2014**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Jiří BENEŠ**
Osobní číslo:	**A12N0026P**
Studijní program:	**N3902 Inženýrská informatika**
Studijní obor:	**Číslicové systémy**
Název tématu:	**Komparativní analýza multitaskingových operačních systémů pro embedded aplikace**
Zadávající katedra:	**Katedra informatiky a výpočetní techniky**

Zásady pro vypracování:

Cílem práce je vyhotovit komparativní studii multitaskingových OS a RTOS v embedded systému na základě praktické verifikace jejich funkcionality. Praktická část bude realizována pro platformu Raspberry Pi.

1. Prostudujte dostupné multitaskingové OS a RTOS, modifikované pro Raspberry Pi.
2. Prostudujte dostupný software a metody, jimiž je možné simulovat benchmark vybraných multitaskingových OS a RTOS.
3. Vytvořte testovací scénáře pro komparaci vybraných multitaskingových OS a RTOS.
4. Seznamte se s hardwarem, na kterém budete provádět praktickou verifikaci.
5. Na základě provedených teoretických studií proveďte praktickou komparaci operačních systémů a analyzujte získané výsledky.

Rozsah grafických prací:	**dle potřeby**
Rozsah pracovní zprávy:	**doporuč. 50 s. původního textu**
Forma zpracování diplomové práce:	**tištěná**

Seznam odborné literatury:

dodá vedoucí diplomové práce

Vedoucí diplomové práce:	**Dr. Ing. Karel Dudáček** Katedra informatiky a výpočetní techniky
Datum zadání diplomové práce:	**30. srpna 2013**
Termín odevzdání diplomové práce:	**15. května 2014**

Doc. Ing. František Vávra, CSc.
děkan

L.S.

Západočeská univerzita v Plzni
Fakulta aplikovaných věd

Prof. Ing. Jiří Šafařík, CSc.
vedoucí katedry

V Plzni dne 12. září 2013

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 15. května 2014

.....

Bc. Jiří Beneš

Poděkování

Rád bych na tomto místě poděkoval panu Dr. Ing. Karlu Dudáčkovi za odborné vedení a ochotu, kterou mi v průběhu zpracovávání diplomové práce věnoval.

Abstract

This master thesis deals with a comparative analysis of multitasking OS and RTOS specified for microcomputer *Raspberry Pi*. Except for implementation of conventional OS there is a methodology of an OS kernel modification for a deterministic run, specifically a transformation to fully preemptive RTOS, described in the thesis. The key area of the analysis, however, is a benchmark of real-time characteristics according to testing scripts defined in advance. On the basis of the benchmark, there are measured latencies during radical stress described at the close of the thesis. Abysmal time differences among OS and transformed RTOS are obvious. By the means of the verification of real-time characteristics implemented on OS and RTOS, it was proven that the *Raspberry Pi* microcomputer may be used not only as an embedded hardware alternative to a PC or HTPC units, but also for an application in all sorts of fields of industrial automatization.

Abstrakt

Diplomová práce má za úkol komparativně analyzovat multitaskingové OS a RTOS specifikované pro mikropočítač *Raspberry Pi*. Kromě implementace konvenčních OS je v práci popisována i metodika modifikace jádra OS pro profit deterministického chodu. Konkrétně se jedná o transformaci na plně preemptivní RTOS. Kardinální oblastí analýzy je pak benchmark realtimových vlastností dle předem definovaných testovacích scénářů. Na základě benchmarku jsou v závěru práce uvedeny naměřené latence při radikální zátěži, z nichž jsou jasně vidět propastné časové diference mezi OS a transformovanými RTOS. Verifikací realtimových vlastností naimplementovaných OS a RTOS bylo dokázáno, že mikropočítač *Raspberry Pi* lze použít nejen jako embedded hardware alternativní k PC, či HTPC jednotkám, ale i pro aplikaci v nejrůznějších oblastech průmyslové automatizace.

Obsah

1 Úvod

Diplomová práce by měla čtenáři poskytnout ucelený pohled na problematiku realtimových vlastností u dostupných OS a RTOS specifikovaných pro embedded hardware v podobě mikropočítače *Raspberry Pi*. Konkrétně je pozornost soustředěna na detailní popis metodiky implementace vybraných OS a RTOS. Dále jsou v práci uvedeny nástroje určené pro simulaci benchmarku a prostřednictvím nich i sestaveny testovací scénáře. Cíl práce spočívá v realizaci praktické komparace naimplementovaných OS a RTOS výhradně zaměřené na otestování realtimových vlastností. Na základě testování je pak hlavním účelem analýza získaných výsledků.

V preambuli teoretické části je tato práce soustředěna na definici kooperativního a preemptivního multitaskingu. Následuje základní charakteristika RTOS včetně jeho podtypů, kterými jsou hard RTOS, soft RTOS a firm RTOS. Jádrem teoretické části jsou pak charakteristiky konvenčních OS a RTOS specifikovaných pro mikropočítač *Raspberry Pi*. Přičemž v rámci kapitoly pojednávající o RTOS jsou hlavní oblastí zájmu řešení, jenž by modifikovala standardní OS tak, aby získal realtimové vlastnosti (předmětem zájmu je operační systém *Linux*[1]). V závěru teoretické části je dále popsán software vhodný pro simulaci benchmarku a sestavené testovací scénáře určené pro praktickou realizaci. Jako poslední je interpretována platforma *Raspberry Pi*, což kromě základní charakteristiky zahrnuje i popis hierarchie bootování a architektury ARM.

Realizační část práce v prvé řadě popisuje implementaci konvenčních OS a RTOS na mikropočítač *Raspberry Pi*. Veškeré kroky implementace jsou přitom doplněny o použité konzolové příkazy, aby čtenář získal lepší orientaci v textu (konzolové příkazy jsou v celé práci kvůli vyšší přehlednosti uváděny bez promptů). V kapitole pojednávající o implementaci RTOS jsou jednotlivé pasáže soustředěny na aplikaci a posléze i verifikaci frameworku *Xenomai* a patche *PREEMPT RT*. Na základě výše uvedených implementací je pak v další kapitole uvedena praktická komparace vybraných OS a RTOS dle definovaných testovacích scénářů. Přičemž výhradní oblastí této komparace je test realtimových vlastností. Jedná se tedy o měření latencí systému v klidovém režimu a v režimu radikální zátěže. Získaná data jsou v závěru práce dále analyzována.

[1] *Označení Linux, které se běžně používá, je ve skutečnosti pouze název jádra operačního systému. Pro celkovou koncepci operačního systému je třeba jádro a další specifický software. V tomto případě se jedná o software ve formě balíčku GNU (GNU Core Utilities, GNU Compiler Collection, apod.). Korektní název operačního systému by tedy byl GNU/Linux. Jedná se ovšem o kontroverzní označení, proto bude nadále uváděno pouze označení Linux, jakožto název operačního systému [1].*

2 Teoretická část

Teoretická část by měla čtenáři poskytnout komplexní představu o problematice dostupných OS a RTOS specifikovaných pro mikropočítač *Raspberry Pi* a pravidlech jejich komparativního testování s primárním zaměřením na realtimové vlastnosti. Jedná se o jakousi bázi pro část analytickou, resp. realizační, kde je pozornost soustředěna na implementaci těchto OS a RTOS.

Konkrétně tato část popisuje základní princip multitaskingu u operačních systémů a dále pak charakterizuje RTOS. Jádrem teoretické části je popis multitaskingových OS a RTOS specifikovaných pro mikropočítač *Raspberry Pi*. Dalšími podstatnými pasážemi jsou kapitoly pojednávající o softwaru a metodách pro simulaci benchmarku a o testovacích scénářích. Na závěr je uvedena interpretace platformy *Raspberry Pi*.

2.1 Princip multitaskingu u operačních systémů

Multitasking, jako takový, je schopnost operačního systému simulovat současný běh více úloh. Jedná se v podstatě o pseudoparalelismus, kdy multitaskingový operační systém za určitých podmínek velmi rychle přepíná jednotlivé běžící úlohy (což vytváří pouze dojem současného běhu). Každá úloha má pak na svůj běh předem vyměřený čas, resp. časové kvantum [2][3].

2.1.1 Kooperativní multitasking

Kooperativní multitasking je historicky mladší, než v další podkapitole zmiňovaný, preemptivní multitasking. Jedná se o druh multitaskingu, ve kterém všechny běžící úlohy mezi sebou aktivně spolupracují. Přičemž každá tato úloha musí mnohokrát za sekundu prostřednictvím systémového volání předat řízení zpět operačnímu systému. Na základě profitu řízení může operační systém spustit další úlohu. Operační systém musí ovšem opět čekat na okamžik, kdy se běžící úloha sama vzdá řízení [3].

Z výše uvedeného textu jasně plyne zásadní nevýhoda kooperativního multitaskingu. Uvažujme situaci, kdy právě běžící úloha nepředá operačnímu systému zpět řízení (předpokladem budiž chyba v běžící úloze, která vede k nekonečné regresi). Operační systém tedy nebude schopen spustit další úlohu, čímž může dojít k zamrznutí celého systému.

2.1.2 Preemptivní multitasking

U preemptivního multitaskingu spočívá zásadní rozdíl v privilegiu operačního systému přerušit každou běžící úlohu. Operační systém má tedy plnou kontrolu nad řízením. Prostřednictvím časovače v určitých intervalech dochází k přerušení dané běžící úlohy a je provedena kontrola stavu (např. kontrola priorit). Operační systém pak spustí jinou úlohu nebo nechá tu právě přerušenou úlohu dále běžet. Může ovšem také dojít k situaci, kdy se běžící úloha sama vzdá svého časového kvanta, resp. řízení a takzvaně se uspí [2].

Po analýze textu výše je zřejmé, že, vzhledem k plné kontrole operačního systému nad řízením, by v případě chyby v běžící úloze (opět předpokladem budiž chyba vedoucí k nekonečné regresi) operační systém prostřednictvím časovače přidělil výpočetní čas jiné úloze. Při srovnání kooperativního a preemptivního multitaskingu lze vyvodit jednu podstatnou vlastnost. Preemptivní multitasking je, co se bezpečnosti týká, na vyšší úrovni.

Nevýhodou preemptivního multitaskingu je však větší hardwarová náročnost.

2.2 Operační systémy reálného času

Operační systémy reálného času (RTOS) jsou specifické OS, které jsou schopny zabezpečit spouštění úloh tak, aby dávaly korektní výsledky v požadovaném čase [4].

2.2.1 Základní charakteristika

RTOS je charakterizován několika základními vlastnostmi. Jako první lze uvést predikovatelnost. Tím se myslí, že pro zadané parametry úloh je možné určit chování RTOS (maximální latence systému při přerušení apod.). Další důležitou vlastností je spolehlivost. Je zřejmé, že stabilní RTOS je základ pro vytvoření spolehlivého softwaru. Opomenout nelze také výkonost, která je určena například dobou potřebnou k přepnutí jednotlivých úloh, apod. Je nutné hledět i na kompaktnost. Řada zařízení, na kterých běží RTOS, má omezené zdroje, tudíž se očekává například určitá kapacitní nenáročnost na paměť. Jako poslední zde uvedenou vlastností bude škálovatelnost. Přičemž škálovatelný RTOS má modulární koncepci, lze tedy sestavit optimální konfiguraci pro danou úlohu [4].

RTOS je, dle tolerance k době určené pro danou operaci, rozdělován na tři podtypy. Všechny tyto podtypy jsou uvedeny v následujících podkapitolách.

2.2.2 Hard RTOS

Výpočet musí být proveden v požadovaném čase za všech okolností. Je striktně vyžadována nulová tolerance k opožděným výsledkům. Tyto systémy lze využít v různých řídících aplikacích [4].

2.2.3 Soft RTOS

Pokud není výpočet proveden v požadovaném čase, je to nepříjemné, ale bez fatálních následků. Není striktně vyžadována nulová tolerance k opožděným výsledkům. Jako příklad využití těchto systému lze uvést DVD přehrávače [4].

2.2.4 Firm RTOS

Není-li výpočet proveden v požadovaném čase, nejsou výsledky použitelné. Neplynou z toho však žádné fatální následky. Příkladem těchto systémů může být systém provádějící předpověď počasí [4].

2.3 Multitaskingové OS specifikované pro *Raspberry Pi*

Multitaskingových operačních systémů (dále jen OS), které mají specifikaci přímo pro mikropočítač *Raspberry Pi,* je v současné době několik. V rámci této cílové platformy existují konvenční OS, které jsou k dispozici na oficiálních webových stránkách britské nadace *Raspberry Pi Foundation*. Jedná se o operační systém *RISC OS* a pak jednotlivé distribuce OS *Linux*. Konkrétně to jsou distribuce *Raspbian, Pidora, Arch Linux ARM, RaspBMC* a *OpenELEC*. Kromě těchto lze na mikropočítač *Raspberry Pi* implementovat i operační systémy jako *Firefox OS* či *Android* (resp. *CyanogenMod*), které jsou jako konvenční brány spíše v oblasti chytrých telefonů a tabletů. Existují i nestandardní operační systémy, které jsou vyvíjené spíše za účelem výzkumu. Jako příklad takového operačního systému, který by byl zároveň specifikován pro mikropočítač *Raspberry Pi,* lze uvést *Plan 9 from Bell Labs*.

K dispozici jsou samozřejmě i jiné operační systémy, ale ty, které jsou uvedeny výše, dostatečně zasahují do široké škály aplikačních oblastí.

2.3.1 *Raspbian*

Raspbian je distribuce OS *Linux*, přičemž se v podstatě jedná o modifikovanou distribuci *Debian* OS *Linux*, která je optimalizována přímo pro mikropočítač *Raspberry Pi*.

Z technického hlediska je vznik distribuce *Raspbian* zapříčiněn modifikací oficiálního portu ArmHardFloatPort[2] distribuce *Debian*. Port byl neoficiální cestou modifikován, aby bylo možné původní distribuci *Debian* přenést na mikropočítač *Raspberry Pi*. Výsledkem tohoto neoficiálního portu, který využívá hard-float ABI, je právě distribuce *Raspbian* (existuje i starší port pro mikropočítač *Raspberry Pi*, který využívá soft-float ABI). Vzhledem k hard-float ABI lze pak očekávat výrazně vyšší výkon u aplikací, které generují nezanedbatelnou zátěž prostřednictvím aritmetických operací s plovoucí řádovou čárkou (a nejen u nich) [5].

Distribuce *Raspbian* si klade za cíl, být primární volbou distribuce, resp. operačního systému pro většinu uživatelů vlastnící mikropočítač *Raspberry Pi* (což je z velké části splněno). Další cíl distribuce *Raspbian* je zůstat co „nejblíže" distribuci *Debian*. Tento cíl má logické opodstatnění. Distribuce *Debian* má po celém světě miliony uživatelů, z čehož vyplývá i široká škála různých dokumentací či návodů. Současný stav, kdy většina informací vztahujících se na distribuci *Debian* není problém aplikovat i na stejnou verzi distribuce *Raspbian*, je ideální (proto by měl být tento stav úzkého provázání obou distribucí zachován) [5].

[2] *ArmHardFloatPort, resp. armhf je jeden z dostupných portů distribuce Debian pro architekturu ARM. Konkrétně je tento port cílen pro procesory architektonické verze ARMv7 a vyšší. Neoficiální verze portu ovšem podporuje i procesory architektonické verze ARMv6. Kardinální podstata neoficiálního portu spočívá v možnosti přenést distribuci Raspbian na mikropočítač Raspberry Pi.*

Raspbian, jakožto distribuce OS *Linux*, používá preemptivní multitasking.

Na závěr této podkapitoly je v tabulce č. 2.3.1 uveden stručný přehled elementárních informací o distribuci *Raspbian* OS *Linux* (aktuální k 18. 2. 2014).

Tab. č. 2.3.1 – Tabulka elementárních informací distribuce *Raspbian* OS *Linux*

Aktuální stabilní verze	January 2014 Wheezy
Datum vydání	7. 1. 2014
Webové stránky projektu	http://raspbian.org/
Kapacita obrazu disku (IMG souboru)	2 892 800 kB
Implicitní uživatelské rozhraní	CLI

2.3.2 *Pidora*

Pidora, respektive *Raspberry Pi Fedora Remix*, je distribuce OS *Linux* optimalizována pro běh na mikropočítači *Raspberry Pi*. Distribuce *Pidora* je založena na projektu *Fedora ARM Secondary Architecture Project*, který je součástí samotného projektu *Fedora* [5].

Distribuce *Pidora* vznikla separací původní modifikované distribuce *Fedora*, která se snažila o portaci na mikropočítač *Raspberry Pi*. Hlavním důvodem této separace byla odlišná struktura jádra operačního systému, což vedlo k nedostatečné podpoře [5].

Aktuální verze distribuce *Pidora* zpřístupňuje všechny softwarové balíčky z repozitářů organizace *Seneca's Centre for Development of Open Technology* (označována také jako CDOT). Tato organizace softwarové balíčky zároveň vytváří i spravuje. Důležitou poznámkou je též, že veškeré softwarové balíčky jsou kompatibilní s procesory architektonické verze ARMv6 [6].

V prologu této podkapitoly byl, kromě distribuce *Pidora*, uveden i název *Raspberry Pi Fedora Remix* (přesněji pouze *Fedora Remix*). Je vhodné vysvětlit, proč je tento název možné používat. Jedná se totiž o sekundární označení distribuce *Pidora*, které stručně identifikuje, z čeho je daná distribuce odvozena (cílem je upozornit koncové uživatele, že dostanou k dispozici modifikovaný software distribuce *Fedora*) [6].

Pidora, jakožto distribuce OS *Linux*, používá preemptivní multitasking.

V tabulce č. 2.3.2 je uveden stručný přehled elementárních informací o distribuci *Pidora* OS *Linux* (aktuální k 18. 2. 2014).

Tab. č. 2.3.2 – Tabulka elementárních informací distribuce *Pidora* OS *Linux*

Aktuální stabilní verze	18
Datum vydání	9. 8. 2013
Webové stránky projektu	http://pidora.ca/
Kapacita obrazu disku (IMG souboru)	1 750 269 kB
Implicitní uživatelské rozhraní	GUI

2.3.3 *Arch Linux ARM*

Arch Linux ARM je distribuce OS *Linux*, určená pro procesory s architekturou ARM. Distribuce obsahuje podporu soft-float u procesorů s architektonickou verzí ARMv5TE a podporu hard-float u procesorů s architektonickou verzí ARMv6 a ARMv7 [7].

Distribuce *Arch Linux ARM* pokračuje ve filozofii standardní distribuce *Arch Linux*. Snaží se tedy být nenáročnou a zároveň uživatelsky přívětivou, přičemž uživateli garantuje i úplnou kontrolu nad systémem (tzn. i odpovědnost). Výhodný může být vývojový cyklus distribuce, díky kterému vycházejí aktualizace skoro každý den (tzv. rolling updates). Aktualizace tedy mají formu menších balíčků, které systému postupně přidávají různá vylepšení [7].

Arch Linux ARM, jakožto distribuce OS *Linux*, používá preemptivní multitasking.

V tabulce č. 2.3.2 je uveden stručný přehled elementárních informací o distribuci *Arch Linux ARM* OS *Linux* (aktuální k 18. 2. 2014).

Tab. č. 2.3.3 – Tabulka elementárních informací distribuce *Arch Linux ARM* OS *Linux*

Aktuální stabilní verze	January 2014
Datum vydání	6. 1. 2014
Webové stránky projektu	http://archlinuxarm.org/
Kapacita obrazu disku (IMG souboru)	1 914 880 kB
Implicitní uživatelské rozhraní	CLI

2.3.4 *RISC OS Pi*

RISC OS Pi je první oficiální vydání operačního systému *RISC OS* optimalizovaného pro běh na mikropočítači *Raspberry Pi*. Technicky je tedy *RISC OS Pi* pouze distribuce operačního systému *RISC OS*. Přičemž *RISC OS*, jako takový, je britský operační systém, který byl navržen speciálně pro architekturu ARM. Důležitá je i skutečnost, že tento operační systém byl vyvíjen stejným týmem lidí, kteří stáli u zrodu samotné architektury ARM. Vhodné je také zmínit, že operační systém *RISC OS* není odvozen a ani nemá žádnou souvislost s operačními systémy *Linux* či *Microsoft Windows* [8].

Distribuce *RISC OS Pi* je, co se struktury týče, velmi jednoduchá, kompaktní a přitom efektivní. Výhodou distribuce je i hardwarová nenáročnost. Pro běžného uživatele pak může být zajímavostí programovací jazyk BBC Basic, který je implicitně součástí distribuce (což souvisí s historií operačního systému *RISC OS*, který byl původně vyvíjen společností *Acorn Computers*) [8].

Existují ovšem i určitá omezení, která distribuce *RISC OS Pi* má. Patří mezi ně podpora pouze jednoho uživatelského přístupu, či kooperativní multitasking. Pro některé aplikační oblasti je ale tato distribuce díky svým vlastnostem ideální [9].

V tabulce č. 2.3.4 je uveden stručný přehled elementárních informací o distribuci *RISC OS Pi* (aktuální k 18. 2. 2014).

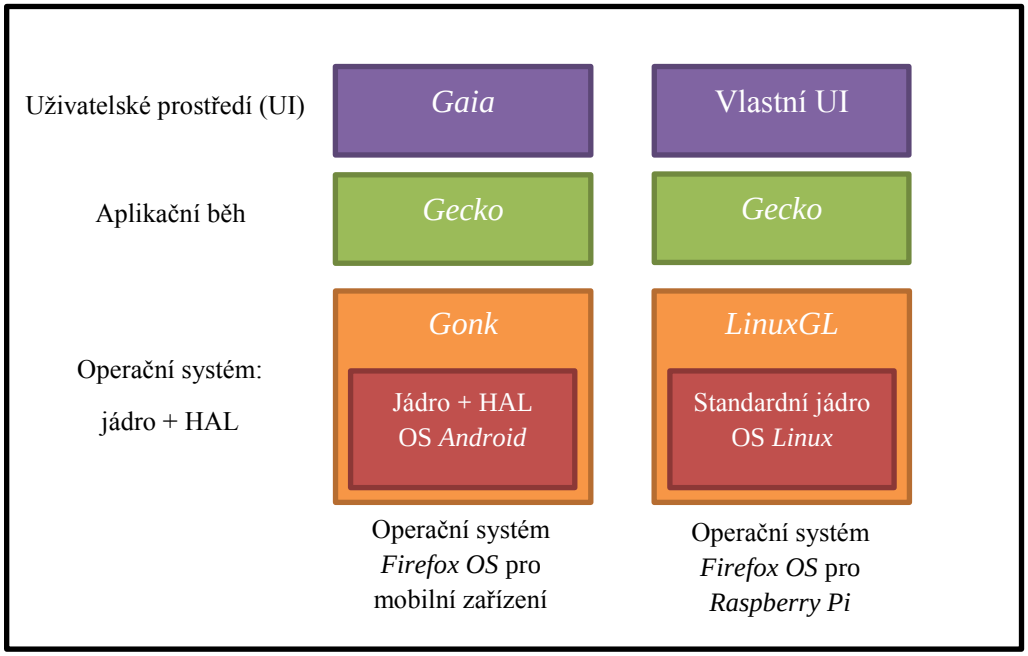
Tab. č. 2.3.4 – Tabulka elementárních informací distribuce *RISC OS Pi*

Aktuální stabilní verze	5.21
Datum vydání	10. 7. 2013
Webové stránky projektu	https://www.riscosopen.org/
Kapacita obrazu disku (IMG souboru)	1 921 024 kB
Implicitní uživatelské rozhraní	GUI

2.3.5 *Firefox OS*

Firefox OS, původním názvem *Boot to Gecko* (*B2G*), je operační systém vycházející z OS *Linux*. Primárně je tento systém cílen do oblasti chytrých telefonů a tabletů. Výhodou je, že operační systém *Firefox OS* používá otevřené standardy, tedy žádný proprietární[3] software či technologie [10].

V současné době je k dispozici modifikovaný operační systém *Firefox OS*, který je optimalizovaný pro běh na mikropočítači *Raspberry Pi* (jedná se o neoficiální projekt, jejž vytvořil Oleg Romashin). Tento modifikovaný OS má však oproti standardnímu operačnímu systému *Firefox OS* odlišnou strukturu, což lze vidět na obr. č. 2.3.1. V následujícím textu mu tedy bude, pro lepší pochopení, věnována větší pozornost.



Obr. č. 2.3.1 – Struktura standardního a modifikovaného operačního systému *Firefox OS*

[3] *Proprietární software je software, který má zcela uzavřený zdrojový kód (closed source). Typicky je takový software upraven licencí, která definuje eventuální oblast použití.*

Obr. č. 2.3.1 znázorňuje na levé straně modelu standardní operační systém *Firefox OS*. Základ tvoří nejspodnější vrstva zvaná *Gonk*. Ta je v podstatě kombinací jádra a HAL z operačního systému *Android*. Další vrstva *Gecko* slouží k renderování webových stránek a grafického rozhraní. Vrchní vrstva *Gaia* je pak jediná, která je viditelná pro koncové uživatele (uživatelské rozhraní). Tedy v podstatě vše, co se objeví po spuštění operačního systému *Firefox OS* na obrazovce, je vykresleno prostřednictvím vrstvy *Gaia* (zámek obrazovky, domovská stránka a další aplikace). Pro kompletní vytvoření této vrstvy byly použity technologie HTML, CSS a JavaScript [11].

Aby bylo možné operační systém *Firefox OS* použít i pro mikropočítač *Raspberry Pi*, muselo dojít k některým změnám původního modelu (viz obr. č. 2.3.1). První změna se týká hned nejspodnější vrstvy, kde bylo použito standardní, resp. vanilla[4] jádro OS *Linux* (totéž platí i pro HAL). Tato modifikovaná vrstva pak dostala označení *LinuxGL* (kombinace operačního systému *Linux* a standardu *OpenGL*). Jak již název napovídá, vykreslování grafiky se provádí prostřednictvím *OpenGL* bez *X Windows System (X11)*. Další vrstva *Gecko* pak zastává stejnou funkci, jako v případě levé strany modelu na obr. č. 2.3.1 (viz předchozí odstavec). Vrchní vrstva modelu je intuitivní. Lze použít buď nativní vrstvu *Gaia* nebo je možné vytvořit vlastní uživatelské prostředí [11].

V tabulce č. 2.3.5 je uveden stručný přehled elementárních informací o modifikovaném operačním systému *Firefox OS* (aktuální k 18. 2. 2014).

Tab. č. 2.3.5 – Tabulka elementárních informací modifikovaného operačního systému *Firefox OS*

Aktuální stabilní verze	b2g-17.0a1
Datum vydání	28. 8. 2012
Webové stránky projektu	http://romaxa.info/b2g/
Kapacita kompletního projektu	85 561 kB
Implicitní uživatelské rozhraní	GUI

2.3.6 *CyanogenMod*

Nejvhodnější je označit *CyanogenMod* jako alternativní distribuci operačního systému *Android*. Přičemž tato distribuce modifikuje chování standardního OS *Android* a přidává mu řadu zajímavých vlastností (např. podpora FLAC souborů, OpenVPN klient, inkognito mód, výkonný DSP ekvalizér apod.). Alternativních distribucí je k dispozici samozřejmě více, ale *CyanogenMod* je v současné době nejrozšířenější [12].

Jelikož neexistuje žádný oficiální port, který by umožňoval běh distribuce *CyanogenMod* na mikropočítači *Raspberry Pi*, je dobré si představit projekt *RazDroid*. Jedná se o neoficiální komunitní projekt, který se zabývá optimalizací distribuce *CyanogenMod* pro možnost aplikace v mikropočítači *Raspberry Pi*. Konkrétně je tato snaha cílena

[4] *Označení vanilla (vanilla kernel) definuje elementární „čisté“ jádro OS Linux, které není modifikované prostřednictvím různých patchů. Kód tohoto jádra je vhodný pro širokou škálu aplikačních oblastí, lze ho tedy brát jako univerzální základ.*

na distribuci *CyanogenMod* verze 7.2 (založená na OS *Android* verze 2.3 s označením *Gingerbread*) a distribuci *CyanogenMod* verze 9 (založená na OS *Android* verze 4.0 s označením *Ice Cream Sandwich*).

Momentální stav projektu *RazDroid* je takový, že jako „použitelná“ se bere pouze distribuce *CyanogenMod* verze 7.2. Přičemž je k dispozici buď s doplňkem ve formě Ethernet menu nebo bez něho. Tato modifikovaná distribuce je však velmi pomalá a nestabilní (obsahuje přetrvávající problémy s hardwarovou akcelerací).

V tabulce č. 2.3.6 je uveden stručný přehled elementárních informací o distribuci *CyanogenMod* OS *Android* (aktuální k 18. 2. 2014).

Tab. č. 2.3.6 – Tabulka elementárních informací modifikované distribuce *CyanogenMod* OS *Android*

Aktuální „použitelná“ verze	7.2 (Android 2.3)
Datum vydání	16. 2. 2012
Webové stránky projektu	https://github.com/Razdroid
Kapacita obrazu disku (IMG souboru)	1 992 704 kB
Implicitní uživatelské rozhraní	GUI

2.3.7 *Plan 9 from Bell Labs*

Plan 9 from Bell Labs (dále jen *Plan 9*) je distribuovaný operační systém, vyvíjený jakožto nástupce operačního systému *Unix*, určený primárně pro výzkum. Operační systém *Plan 9* by ve své podstatě měl být schopen po připojení k lokální síti velmi jednoduše spolupracovat s ostatními připojenými počítači se stejným operačním systémem a zároveň fungovat i zcela samostatně. Tato kooperace je dokonce tak pokročilá, že se všechny propojené systémy dokáží navzájem dělit nejen o místo na svých pevných discích, ale také o výkon procesoru, operační paměť a další systémové zdroje [13].

Pro přístup k souborům a výpočetním zdrojům vzdáleného počítače byla navržena zcela nová architektura a nový síťový protokol. Spojovacím článkem se nakonec stal protokol 9P. Tento protokol se pak stal nativním pro většinu odvozenin operačního systému *Plan 9* (např. OS *Plan B*) [13].

Port, který uzpůsobuje běh operačního systému *Plan 9* i na mikropočítači *Raspberry Pi*, vytvořil Richard Miller (Miller 9Pi). V tabulce č. 2.3.7 je uveden stručný přehled elementárních informací o tomto operačním systému (aktuální k 18. 2. 2014) [14].

Tab. č. 2.3.7 – Tabulka elementárních informací operačního systému *Plan 9*

Aktuální stabilní verze	4. edice (Miller 9Pi)
Datum vydání	18. 11. 2013
Webové stránky projektu	http://plan9.bell-labs.com/
Kapacita obrazu disku (IMG souboru)	471 040 kB
Implicitní uživatelské rozhraní	GUI

2.3.8 *RaspBMC*

RaspBMC je distribuce OS *Linux*, která implicitně obsahuje XBMC systém a je cílena pro platformu *Raspberry Pi*. Pro lepší pochopení účelovosti této distribuce je dobré uvést, že XBMC systém představuje jedno z prostředí multimediálních center. Při aplikaci v mikropočítači *Raspberry Pi* se pak může jednat o vhodnou alternativu k HTPC[5] [14][15].

Vhodné je ještě poznamenat, že distribuci *RaspBMC* vytvořil Sam Nazarko, který je zároveň i jejím správcem. Z hlediska architektonické koncepce je distribuce *RaspBMC* založená na již dříve popisované distribuci *Raspbian* (podkapitola 2.3.1). Tento fakt umožňuje přepínání mezi XBMC systémem a standardním CLI režimem na bázi OS *Linux*. Není tedy problém distribuci *RaspBMC* prostřednictvím balíčků doplnit o požadované vlastnosti [16].

V tabulce č. 2.3.8 je uveden stručný přehled elementárních informací o distribuci *RaspBMC* OS *Linux* (aktuální k 18. 2. 2014).

Tab. č. 2.3.8 – Tabulka elementárních informací distribuce *RaspBMC* OS *Linux*

Aktuální stabilní verze	December 2013
Datum vydání	23. 12. 2013
Webové stránky projektu	http://www.raspbmc.com/
Kapacita obrazu disku (IMG souboru)	1 331 200 kB
Implicitní uživatelské rozhraní	GUI

2.3.9 *OpenELEC*

OpenELEC je distribuce OS *Linux* určená pro embedded hardware (včetně mikropočítače *Raspberry Pi*) a stejně, jako výše uvedená distribuce *RaspBMC*, obsahuje XBMC systém. Distrubuce *OpenELEC* sice neumožňuje přepnutí do CLI režimu, avšak snaží se to kompenzovat o něco vyšší rychlostí XBMC systému (např. doba jeho inicializace) [17].

V tabulce č. 2.3.9 je uveden stručný přehled elementárních informací o distribuci *OpenELEC* OS *Linux* (aktuální k 18. 2. 2014).

Tab. č. 2.3.9 – Tabulka elementárních informací distribuce *OpenELEC* OS *Linux*

Aktuální stabilní verze	3.2.0
Datum vydání	14. 9. 2013
Webové stránky projektu	http://openelec.tv/
Kapacita kompletního projektu	105 107 kB
Implicitní uživatelské rozhraní	GUI

[5] *Označení HTPC nesou speciální počítačové jednotky určené pro multimediální aplikace. Tyto jednotky dokáží simulovat většinu zařízení, používaných v rámci domácího kina a navíc jsou schopny přidat klíčové benefity personálního počítače.*

2.4 Multitaskingové RTOS specifikované pro *Raspberry Pi*

V následujících podkapitolách jsou uvedeny důvody pro selekci vhodných RTOS, přičemž tyto vybrané RTOS jsou pak dále podrobně rozebírány. Konkrétně se jedná o framework *Xenomai* a patch *PREEMPT RT* (prostřednictvím nichž lze „vyrobit“ RTOS).

2.4.1 Kritéria pro selekci vhodného RTOS

Jako kritérium pro selekci vhodného RTOS bylo stanoveno, že se musí jednat o stabilní operační systém reálného času, který je specifikován pro platformu *Raspberry Pi*. Ideálně by se mělo jednat o hard RTOS variantu. Dalším kritériem je kompatibilita s prostředím operačního systému *Linux* (kvůli relevantnější komparaci s konvenčními OS, jež jsou vesměs postavené na OS *Linux*).

2.4.2 OS *Linux* v embedded aplikacích

Vzhledem k výše stanoveným kritériím je třeba zaměřit se na OS *Linux* v embedded aplikacích. Konkrétně pak na podporu zpracování v reálném čase. Standardní vanilla jádro OS *Linux* je sice stabilní a dobře odladěné, ale bez speciálních úprav není příliš vhodné pro časově kritické aplikace. Avšak existuje možnost modifikace tohoto standardního jádra. Dle oblasti aplikace se tyto modifikovaná jádra dělí do skupin uvedených níže [1].

- distribuční jádra
  - vznikají za účelem zařazení do distribucí OS *Linux*
- jádra se zvýšenou stabilitou
  - určeno pro účely, kde má stabilita přednost před výkonem
- experimentální jádra
  - používají se pro otestování nových technologií
- jádra pro embedded účely
  - obsahují modifikace pro specifická určení, nejčastěji se jedná o úpravy pro časově kritické operace

Hlavní oblastí zájmu v této podkapitole jsou pak výše uvedené jádra pro embedded účely. Po prostudování dostupných možností bylo zjištěno, že pro platformu *Raspberry Pi* by mohl být použit patch *PREEMPT RT*. Ten modifikuje standardní jádro OS *Linux* tím, že mu implementuje realtimové vlastnosti. Ovšem toto řešení pouze konverguje k charakteristice hard RTOS.

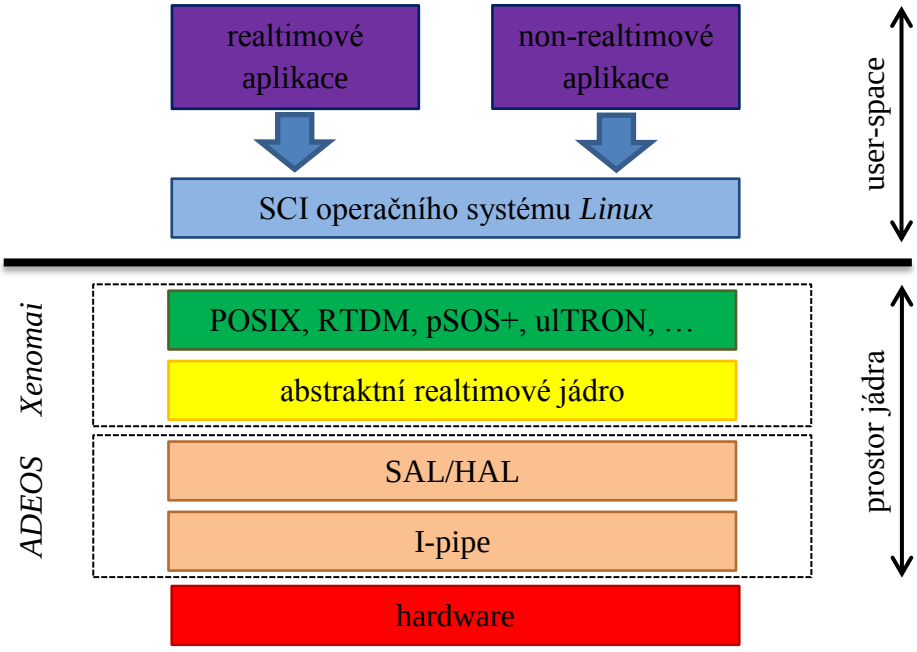
Existuje ale i řešení, které zcela splňuje charakteristiku hard RTOS. Jedná se o framework *Xenomai*. Přičemž tento framework poskytuje hard realtimovou podporu pro user-space aplikace (pouze pro software vytvořený v API frameworku *Xenomai*). Více informací je uvedeno v následující podkapitole.

2.4.3 Framework *Xenomai*

Xenomai je realtimový framework spolupracující s jádrem OS *Linux*, jenž je primárně určen pro vývojové účely. Cílem frameworku je poskytnout kvalitní interface-agnostic a hard realtimovou podporu pro user-space aplikace. Samozřejmostí je bezproblémová integrace do prostředí OS *Linux* [18].

Důležité je pochopit vztah mezi frameworkem *Xenomai* a OS *Linux*. Podstatou problému je, že framework *Xenomai* poskytuje realtimové vlastnosti pouze pro software, který využívá API frameworku *Xenomai*. Přičemž toto API je mnohem „menší“ než API operačního systému *Linux*. Je to logické, a to už jen proto, že API frameworku *Xenomai* neposkytuje tolik služeb. V případě, že je požadavek na služby, které neposkytuje API frameworku *Xenomai*, dá se k tomu použít API OS *Linux*. To ovšem může být problém, protože OS *Linux* má pod frameworkem *Xenomai* nižší prioritu než nativní *Xenomai* aplikace a tudíž i horší realtimové vlastnosti. Pokud by se výše uvedené hodně zobecnilo, lze tvrdit, že OS *Linux* běží jako úloha s nejnižší prioritou ve frameworku *Xenomai*.

Co se týká celkové koncepce, tak framework *Xenomai* je založen na realtimovém abstraktním jádru, nad kterým jsou efektivně emulovány API. Tím je zjednodušen proces přenesení aplikací běžících v konvenčních RTOS jako *VxWorks*, *pSOS+*, *VRTX* či *ulTRON* do prostředí OS *Linux*. Základní snahou frameworku *Xenomai* je tedy přenos aplikací vyžadující realtimové záruky z prostoru jádra do user-space (ostatně viz obr. č. 2.4.1). Celý framework *Xenomai* je realizován na vrstvě ADEOS kvůli prioritnímu zpracování hardwarového přerušení prostřednictvím I-pipe (Interrupt Pipe – jednoduchá „roura přerušení“) a způsobu spolupráce mezi realtimovým rozšířením a jádrem OS *Linux* [18].



Obr. č. 2.4.1 – Funkcionální schéma frameworku *Xenomai*

Výhoda frameworku *Xenomai* spočívá v jeho konstrukci, díky které má výborné realtimové vlastnosti. Latence systému u aplikací vytvořených v API frameworku *Xenomai* se pohybují v řádu desítek mikrosekund. Je ovšem nutné upozornit, že tyto aplikace nejsou normálními procesy operačního systému *Linux* v user-space. Jedná se totiž o speciální realtimové procesy. Z toho zároveň plyne i nevýhoda frameworku *Xenomai*.

2.4.4 Patch *PREEMPT RT*

Patch *PREEMPT RT* umožňuje přidat jádru operačního systému *Linux* podporu plně preemptivního módu a další klíčové realtimové vlastnosti, díky čemuž se hodně přibližuje charakteristice hard RTOS (v podstatě je snahou minimalizovat všechny možné zdroje latencí). Proces aplikace tohoto patche spočívá v modifikaci standardního vanilla jádra OS *Linux*, resp. v jeho rozšíření o realtimové vlastnosti. Patch *PREEMPT RT* je vytvářen malou skupinou vývojářů jádra OS *Linux*, v jejichž čele stojí Ingo Molnar [19].

V prvé řadě je dobré zaměřit se právě na ony klíčové vlastnosti, které transformují jádro OS *Linux* na realtimové. Je tedy vhodné věnovat se zdrojům nežádoucích latencí, které mohou způsobovat například sekce se spinlocky. To je v případě patche *PREEMPT RT* řešeno nahrazením spinlocků spícími zámky, označovaných také jako rt_mutexy (ne všech). Tyto spící zámky využívají mechanismus priority inheritance, resp. dědičnosti priorit (tzn. za předpokladu, že existuje prioritní proces, který čeká na uvolnění zdroje drženého nižším procesem, tak priorita nižšího procesu je dočasně povýšena na úroveň procesu, který je ve stavu čekajícím – v podstatě se jedná o řešení problému s neohraničenou prioritní inverzí) [19].

Dále je vhodné se zmínit, že kritické sekce, ošetřené strukturami spinlock_t a rwlock_t, jsou prostřednictvím patche *PREEMPT RT* transformovány na preemptivní. Vytváření nepreemptivních kritických sekcí v jádře OS *Linux* je přitom stále možné prostřednictvím struktury raw_spinlock_t [19].

Patch *PREEMPT RT* dále transformuje obsluhu IRQ na preemptivní, tzn. tak, aby šla spouštět ve vláknech jádra OS *Linux*. Důležitou věcí, týkající se realtimových vlastností, jsou také časovače, přesněji časovače s vysokým rozlišením (jelikož realtimové aplikace vyžadují „jemné časování“). Tyto časovače patch *PREEMPT RT* také obsahuje [19].

Implicitně jsou v rámci patche *PREEMPT RT* k dispozici i další vylepšení, která ovlivňují realtimové vlastnosti systému. Jako příklad lze uvést validátor zámků, který dokáže detekovat těžko vyvolatelná uváznutí (deadlock), aniž by k nim muselo opravdu dojít. Dále například sledovač latencí, či detektor SMI pro regulaci nečekaných prodlev systému [19].

Výhoda patche *PREEMPT RT* je v tom, že pouze „poupraví“ jádro OS *Linux*. Není tedy problém, abychom měli realtimové procesy, jakožto normální aplikace operačního systému *Linux* v user-space (na rozdíl od frameworku *Xenomai*). Nevýhodou jsou ale vyšší latence systému, které se pohybují v řádu stovek mikrosekund.

2.5 Software a metody pro simulaci benchmarku

Kardinální podstata této diplomové práce, mimo jiné, spočívá v praktické komparaci operačních systémů. Výhradní oblastí komparace jsou pak realtimové vlastnosti. Po prostudování dostupného softwaru a metod, jimiž by bylo možné simulovat benchmark vybraných OS a RTOS, se pak jako ideální jevily nástroje uvedené v následujících podkapitolách. Důvodem této volby je jejich architektonická struktura a přední postavení, jakožto nástrojů nejvhodnějších k testování realtimových vlastností v prostředí OS *Linux*.

2.5.1 *Cyclictest*

Nástroj *Cyclictest* slouží k měření anomálií časovače jádra operačního systému a to s vysokou přesností. Jedná se o nejpoužívanější metodiku k testování realtimových vlastností operačního systému. Funkcionalita nástroje *Cyclictest* spočívá v měření doby mezi vygenerováním přerušení a okamžikem, kdy se o přerušení dozví obslužná user-space aplikace (tzn., měří se doba, která je skutečně ovlivněna operačním systémem). Pokud má aplikace, obsluhující přerušení, nejvyšší prioritu v systému, tak bude výsledná hodnota odpovídat nejhorší latenci (reakční době) způsobené operačním systémem. Autorem nástroje *Cyclictest* je Thomas Gleixner [20].

2.5.2 *Hackbench*

Hackbench je nástroj primárně určený k měření výkonu plánovače jádra operačního systému. Sekundárně slouží jako nezanedbatelný generátor zátěže. Vzhledem ke své popularitě se stal v této oblasti předním benchmarkem. Princip práce nástroje *Hackbench* spočívá v tom, že vytváří skupinu procesů a prostřednictvím soketů nebo „rour“ (pipe) mezi nimi rychle předává zprávy. Výstupem nástroje je pak doba potřebná k odeslání a přijmutí dané skupiny procesů. Uvedený nástroj vytvořil Rusty Russell ve spolupráci s týmem, jehož členy byli Yanmin Zhang, Ingo Molnar a David Sommerseth [21][22].

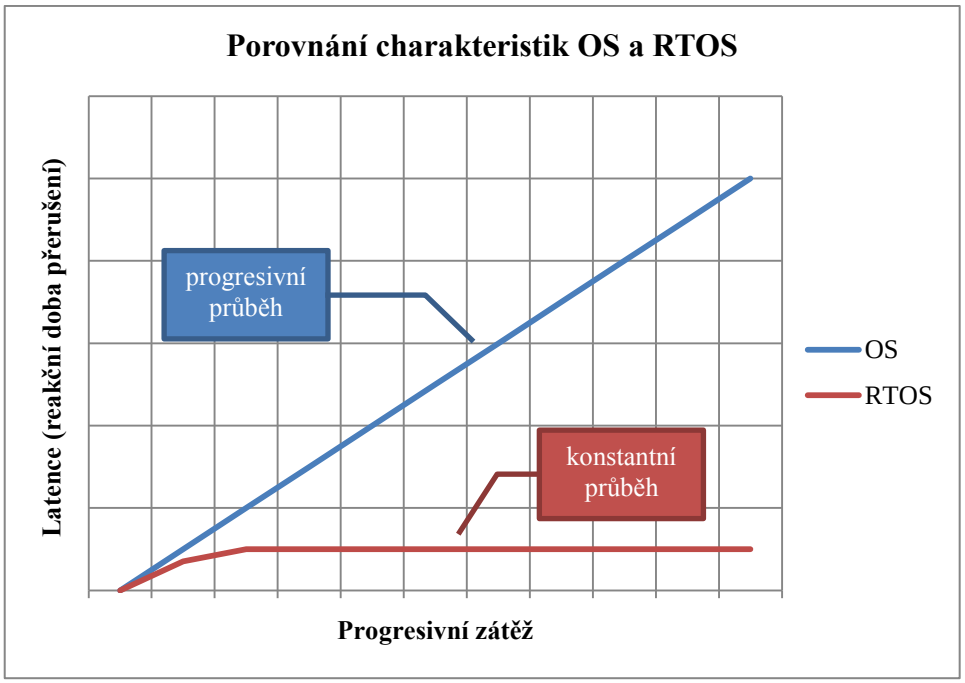
2.5.3 *Cache Calibrator*

Nástroj *Cache Calibrator* je primárně určený k analýze paměťového systému. Sekundárně se jedná o kvalitní generátor silné zátěže. Nástroj během svého běhu řeší úkoly, které jsou velmi náročné na výpočetní výkon. Jako příklad lze uvést analyzování každé úrovně vyrovnávací paměti procesoru či analyzování všech záznamů stránkovacích tabulek TLB. Výstupem pak jsou jednotlivé hodnoty získané výše uvedenou analýzou. V závislosti na analyzované oblasti jsou to hodnoty jako kapacita paměti, počet přístupů, naměřená latence apod. Nástroj *Cache Calibrator* vytvořil Stefan Manegold [23].

2.6 Testovací scénáře

Před samotnou volbou vhodných testovacích scénářů je nutné vymezit operační systémy, jež budou testovány. Výhradní pozornost testování je orientována na realtimové vlastnosti, je tedy zřejmé, že v testu budou zahrnuty všechny implementované operační systémy, které tyto vlastnosti obsahují. Tzn. OS *Linux* modifikovaný frameworkem *Xenomai* a OS *Linux* modifikovaný patchem *PREEMPT RT*. Dále je třeba vybrat všechny operační systémy, jež mají z architektonického hlediska obdobnou strukturu (kvůli relevantnější metodice testování) a zároveň je lze považovat za stabilní. Tyto kritéria splňují *Raspbian*, *Pidora* a *Arch Linux ARM*, jakožto distribuce operačního systému *Linux* (právě prostředí OS *Linux* je zde klíčové).

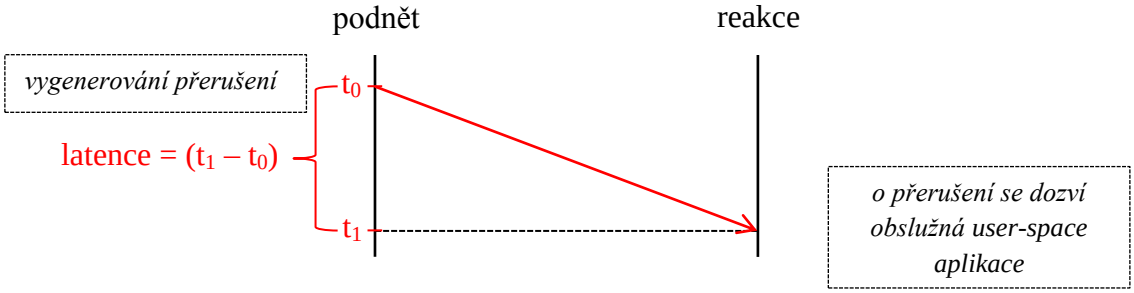
Nyní je důležité definovat, jak budou nástroje uvedené v předchozí kapitole využity. Je evidentní, že OS disponující realtimovými vlastnostmi, by měl mít i při silné zátěži konstantní latence (tzn. ani silná zátěž by neměla ovlivnit „chování“ systému). Jedná se o situaci zjednodušeně ilustrovanou na obr. č. 2.6.1 (zásadní je konstantní průběh u RTOS).



Obr. č. 2.6.1 – Porovnání charakteristik OS a RTOS

Proto je dobré provést měření těchto latencí v klidovém režimu a v režimu silné zátěže. Pro účely simulace zátěže se ideálně hodí nástroje *Hackbench* a *Cache Calibrator*, kdy každý nástroj zatěžuje systém rozdílným způsobem (viz podkapitola 2.5.2 a 2.5.3). Lze tedy očekávat i rozdílné výsledky dle daných testovaných OS či RTOS. K monitorování latencí je pak nutné zvolit nástroj, který je vytvořen v API frameworku *Xenomai* a zároveň stabilně funguje i v prostředí OS *Linux*. Tyto kritéria zcela splňuje

nástroj *Cyclictest* (viz podkapitola 2.5.1). Pro připomenutí je dobré zopakovat, že nástroj *Cyclictest* konkrétně měří dobu mezi vygenerováním přerušení a okamžikem, kdy se o tomto přerušení dozví user-space aplikace. Zjednodušeně řečeno, jedná se o čekací dobu reakce na podnět, resp. reakční dobu přerušení definovanou jakožto latence. Tato situace je znázorněna ve formě jednoduchého diagramu na obr. č. 2.6.2.



Obr. č. 2.6.2 – Časový úsek měřený nástrojem *Cyclictest*

Na základě předchozího uvedeného textu lze sestavit celkem tři testovací scénáře (viz tabulka č. 2.6.1).

Tab. č. 2.6.1 – Tabulka charakterizující testovací scénáře

Testovací scénář	Simulace zátěže	Měření latencí	Časový úsek
1.	žádná	*Cyclictest*	15 min
2.	*Hackbench*	*Cyclictest*	15 min
3.	*Cache Calibrator*	*Cyclictest*	15 min

První varianta testovacího scénáře spočívá v měření latence systému v klidovém režimu, prostřednictvím nástroje *Cyclictest*. Nástroj *Cyclictest* je použit jakožto monitorovací prostředek pro měření latence i v druhém a třetím testovacím scénáři. Druhý scénář pak předpokládá paralelní simulaci zátěže s kontinuálně spuštěným monitorovacím nástrojem *Cyclictest*. Jako zátěž je v tomto případě použit nástroj *Hackbench*. U posledního testovacího scénáře je pak situace obdobná jako u druhého, ale s tím rozdílem, že zátěž je zde simulována nástrojem *Cache Calibrator*.

Doba provádění každého scénáře byla, na základě prostudování obdobných testů souvisejících s frameworkem *Xenomai* a patchem *PREEMPT RT*, stanovena na 15 minut. Tedy po celý tento časový úsek je třeba kontinuálně měřit latenci daného systému.

2.7 Interpretace platformy *Raspberry Pi*

Jako platforma pro realizaci této práce byl zvolen mikropočítač *Raspberry Pi*, který vyvíjí britská nadace *Raspberry Pi Foundation*. Konkrétně se jedná o model B revize 2.0. V době zpracovávání diplomové práce se jednalo o nejvýkonnější variantu, která byla distribuována široké veřejnosti. Text v následující podkapitole bude soustředěn právě na tuto variantu.

2.7.1 Základní charakteristika

Základem mikropočítače je SoC s označením BCM2835 od firmy *Broadcomm*, který nese oficiální specifikaci „*High Definition 1080p Embedded Multimedia Applications Processor*“. Charakteristika mikropočítače *Raspberry Pi* je uvedena níže v tabulce č. 2.7.1. Konkrétní model B určený pro praktickou realizaci této práce je vidět na obr. č. 2.7.1 [24].

Tab. č. 2.7.1 – Tabulka charakteristických parametrů mikropočítače *Raspberry Pi*

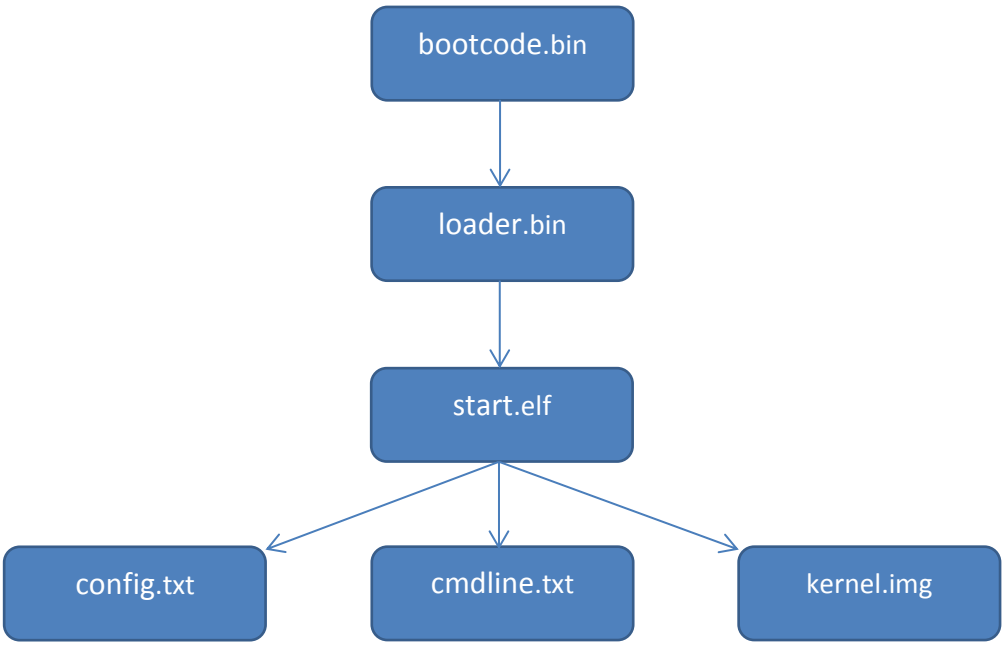
CPU	*ARM11 (jádro ARM1176JZF-S), 700 MHz (OC až 1 GHz)*
GPU	*VideoCore IV, OpenGL ES 2.0, 1080p30, MPEG-4*
Paměť (SDRAM)	*512 MB RAM*
USB 2.0 porty	*2*
Video výstupy	*Kompozitní RCA (PAL a NTSC), HDMI, DSI*
Audio výstupy	*3,5 mm jack*
Úložiště dat	*SD/MMC/SDIO paměťový slot*
Síťová karta	*Ethernetový adaptér 10/100 (RJ45)*
Připojení periférií	*8x GPIO, UART, I^2C sběrnice, SPI sběrnice*
Spotřeba	*700 mA (3,5W)*
Napájení	*5 V přes MicroUSB nebo GPIO*
Rozměry	*85,6 × 53,98 mm*
Hmotnost	*45 g*

Obr. č. 2.7.1 – Mikropočítač *Raspberry Pi* určený pro praktickou realizaci (mod. B, rev. 2.0)

2.7.2 Hierarchie bootování

Tato podkapitola zde má své opodstatnění, přičemž důvod je následující. Mikropočítač *Raspberry Pi* nemá žádnou interní paměť, kam by mohl uložit operační systém či další data. Pokud chceme do mikropočítače zavést operační systém, lze to pouze přes SDHC paměťovou kartou. Tato karta pak slouží jako lokální úložiště. Pro korektní nabootování operačního systému je tedy třeba znát princip, jak mikropočítač v tomto ohledu funguje. Níže je uvedena posloupnost celého procesu bootování (viz také obr. č. 2.6.2).

1.) Do mikropočítače přivedeme zdroj elektrické energie, čímž ho uvedeme do aktivního stavu.
2.) Procesor ARM11 je prozatím vypnut, avšak grafický procesor VideoCore IV je uveden do aktivního stavu. SDRAM je prozatím striktně blokována.
3.) Grafický procesor provede první fázi z procedury bootloaderu (je uložen v ROM na SoC). Ta spočívá v tom, že bootloader přečte SDHC kartu a v druhé fázi načte soubor **bootcode.bin** do vyrovnávací paměti, přesněji do vrstvy označované jako L2. Následně provede spuštění.
4.) Prostřednictvím souboru **bootcode.bin** dojde k odblokování SDRAM.
5.) Provede se třetí fáze. Bootloader opět přečte SDHC kartu a do vyrovnávací paměti načte soubor **loader.bin**, u něhož posléze provede i spuštění.
6.) Prostřednictvím souboru **loader.bin** dojde k přečtení firmwaru GPU (souboru **start.elf**).
7.) A prostřednictvím souboru **start.elf** dojde nakonec k přečtení souborů **config.txt**, **cmdline.txt** a **kernel.img** (které jsou pro vlastní implementaci OS či RTOS velmi důležité, což lze vidět v realizační části této práce).



Obr. č. 2.6.2 – Schéma posloupnosti práce se soubory při procesu bootování

2.7.3 Architektura ARM

Architektura ARM je vyvíjena korporací *ARM Limited*, která sídlí v Británii. Je důležité zmínit se, že tato architektura používá redukovanou instrukční sadu RISC. Ta sice nabízí menší počet instrukcí než komplexní instrukční sada CISC, nicméně tyto instrukce pokrývají všechny běžné potřeby. Instrukce mají stejnou délku, a tudíž stejně dlouhou dobu vykonávání. Níže jsou uvedeny obecné výhody a nevýhody architektury ARM [25].

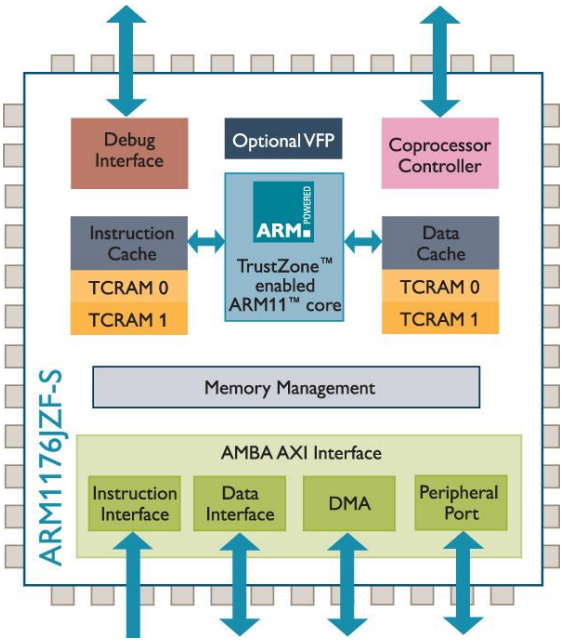
Obecné výhody:

- Nízké vyzařované teplo => pasivní chlazení
- Velmi nízká spotřeba

Obecné nevýhody:

- Chybí výpočetní výkon pro náročné úlohy
- Kompatibilita aplikací => nutná modifikace

Pokud by náš zájem byl soustředěn konkrétně na platformu *Raspberry Pi*, byla by specifikace následující. Mikropočítač *Raspberry Pi* obsahuje procesor s jádrem ARM1176JZF-S, který spadá do rodiny procesorů ARM11 (viz obr. č. 2.7.3). Ta pohání řadu chytrých telefonů a využívá se i pro různé embedded aplikace. Procesor také disponuje extrémně nízkou spotřebou, přičemž frekvenční rozsah je od 350 Mhz do 1 Ghz+. Dále je třeba zmínit dostupnost 32-bitové SIMD pro zpracování médií. Důležitá je též technologie *TrustZone*, kterou procesor také obsahuje. Jedná se ve zkratce o ochranný prostředek určený pro komplexní bezpečnou funkci celého systému. Na obr. č. 2.7.3 lze vidět i jednotku VFP, což je alternativní technologie typu „floating point“ k technologii FPA (starší technologie). Technologie VFP je vzhledem ke své koncepci určená pro různé aplikační oblasti (konvoluční filtry, rychlá Fourierova transformace atd.) [26].



Obr. č. 2.7.3 – Blokové schéma procesoru ARM1176 (převzato z www.arm.com)

3 Realizační část

Tato kapitola pojednává o analytické části diplomové práce a je rozdělena do tří podkapitol. Jsou zde tedy popisovány implementace OS a RTOS specifikovaných pro mikropočítač *Raspberry Pi*. V závěru jsou pak uvedeny výsledky z komparativního testování těchto vybraných naimplementovaných OS a RTOS.

V první podkapitole je oblastí zájmu implementace konvenčních OS, která je i detailně dokumentována. Samotná implementace zahrnuje proces předpřipravení datové struktury SDHC karty, základní nastavení konfigurace a případně i instalaci vhodného GUI. Mimo to jsou u daných OS uvedeny i jejich základní informace a snímky obrazovky.

Druhá podkapitola je orientována na implementaci RTOS, což zahrnuje aplikaci konkrétního řešení a následně i verifikaci funkcionality. Celý princip spočívá ve specifické modifikaci standardního OS, která mu přidá potřebné realtimové vlastnosti jako jsou spolehlivost či determinismus. Tímto postupem v podstatě dojde k transformaci na RTOS. V práci je pak detailně, prostřednictvím konzolových příkazů, uveden celý výše uvedený proces.

Poslední podkapitola této části je orientována na testování vybraných OS a RTOS. Výhradní oblastí testování jsou přitom realtimové vlastnosti. Jedná se tedy o měření latencí systému v klidovém režimu a v režimu radikální zátěže. Cílem tohoto testu je odhalit, zda se u transformovaných RTOS neprojeví nežádoucí anomálie a jak si oproti nim stojí standardní OS.

3.1 Implementace multitaskingových OS na *Raspberry Pi*

V následujících podkapitolách je popsán postup implementace jednotlivých multitaskingových operačních systémů, které jsou specifikovány pro mikropočítač *Raspberry Pi*. Pro samotnou implementaci byly vybrány všechny konvenční doporučené operační systémy a dále pak systémy, které jsou svou konstrukcí zajímavé (v rámci mikropočítače *Raspberry Pi*).

3.1.1 *Raspbian*

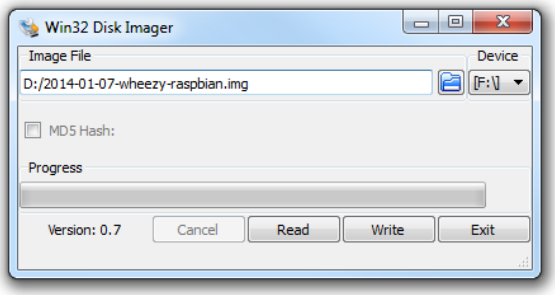
Raspbian je jedna z dostupných distribucí OS *Linux*. Pro její implementaci je nejprve třeba stáhnout potřebný obraz disku, což je soubor ve formátu IMG. K tomu je nejvhodnější použít úložiště níže (aktuální k 18. 2. 2014).

http://downloads.raspberrypi.org/raspbian_latest

Pro praktickou realizaci byl použit obraz disku, který obsahuje distribuci *Raspbian* ve verzi January 2014 s označením *Wheezy*. Stažený soubor se tudíž analogicky dle data vydání jmenuje 2014-01-07-wheezy-raspbian. Dále je nutné tento obraz přenést na SDHC kartu. K tomu lze použít jak OS *Microsoft Windows*, tak OS *Linux*. V následujícím textu je nejprve uveden postup pro OS *Microsoft Windows*.

Aby bylo možné obraz disku přenést na SDHC kartu, je k tomu potřeba nějaký program. Konkrétně program *Win32 Disk Imager* (případně jiný, poskytující obdobnou funkcionalitu). V rámci praktické realizace byl tento program použit ve verzi 0.7 (vzhledem ke kompatibilitě s OS *Microsoft Windows 7* a korektní funkcionalitě při operaci formátování a následném zápisu dat).

Po spuštění programu stačí do kolonky „*Image File*“ napsat cestu k IMG souboru s projektem *Machinoid*, u selektoru „*Device*“ zvolit správnou jednotku SDHC karty a pak jen kliknout na kolonku „*Read*“ a posléze „*Write*“ (viz obr. č. 3.1.1).



Obr. č. 3.1.1 – Program *Win32 Disk Imager* připravený k zápisu distribuce *Raspbian*

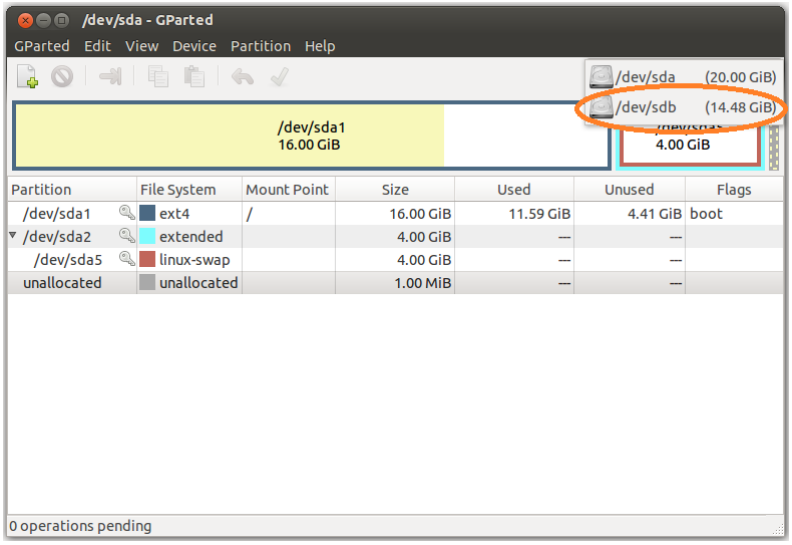
V případě použití OS *Linux* bude, v závislosti na zvolené distribuci a množství nainstalovaných balíčků, možná potřeba doinstalovat program *GParted*.

```
sudo apt-get install gparted
```

Následně se provede spuštění programu.

```
sudo gparted
```

Mělo by se zobrazit okno podobné, jako je na obr. č. 3.1.2. V pravém horním selektoru je třeba zjistit správnou cestu k SDHC kartě. Pokud jde například o SDHC kartu s kapacitou 16 GB, bude se zřejmě jednat o cestu, která je ohraničená oranžovou barvou na tomtéž obrázku, tzn. /dev/sdb.



Obr. č. 3.1.2 – Program *GParted* použitý pro zjištění cesty k SDHC kartě

Další krok spočívá v přenesení dat z IMG souboru na SDHC kartu. To lze v OS *Linux* lehce provést prostřednictvím příkazu níže.

```
sudo dd bs=4M if=/home/user/Downloads/2014-01-07-wheezy-raspbian.img
of=/dev/sdb
```

Přičemž parametr bs=4M udává, že velikost bloku se nastaví na 4 MB (lze zadat například i hodnotu 1 MB, ale doba zápisu dat bude delší). Parametr if=<path> udává cestu k IMG souboru s distribucí *Raspbian*. Poslední parametr of=<path> pro změnu udává cestu k SDHC kartě. Po provedení příkazu je nutné zkontrolovat, zda se opravdu přenesla potřebná kapacita dat. V případě úspěšné kontroly je SDHC karta připravena k použití.

Pro spuštění distribuce *Raspbian* postačí vložit tuto SDHC do mikropočítače *Raspberry Pi* a uvést ho do aktivního stavu (přivedením zdroje elektrické energie). Po spuštění se

nejdříve provede kompletní kontrola systému a v případě, že je tato kontrola validní, zobrazí se konfigurační menu „*Raspberry Pi Software Configuration Tool (raspi-config)*“. Jedná se o nástroj, který umožňuje pozměnit základní chování systému. Po nastavení vhodné konfigurace lze výše popisované menu ukončit prostřednictvím volby „*Finish*“. Dojde k uložení nastavených hodnot a následuje zobrazení konzole. Lze již tedy zadávat příkazy. Pro informativní výpis o systému a dané platformě postačí napsat příkaz níže.

```
uname -a
```

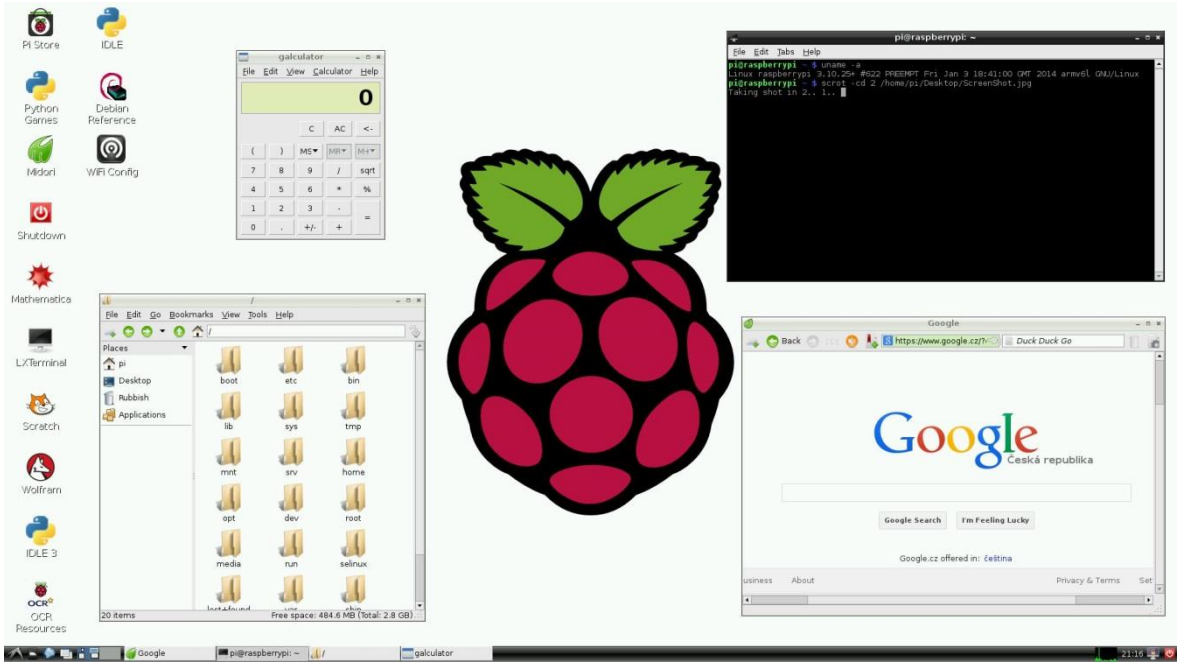
Budou vypsány následující informace.

```
Linux raspberrypi 3.10.25+ #622 PREEMPT Fri Jan 3 18:41:00 GMT 2014 armv6l GNU/Linux
```

Grafické prostředí se pak spustí následujícím příkazem.

```
startx
```

Distribuce *Raspbian* používá grafické prostředí LXDE, tedy jedná se spíše o jednoduché, ale nenáročné prostředí (ideální pro mikropočítač *Raspberry Pi*). Snímek obrazovky tohoto prostředí, v rámci distribuce *Raspbian,* lze vidět na obr. č. 3.1.3.



Obr. č. 3.1.3 – Snímek obrazovky naimplementované distribuce *Raspbian* OS *Linux*

Důležité upozornění se týká dalšího spuštění systému. Distribuce *Raspbian* již bude pro přihlášení požadovat uživatelské jméno a heslo. Proto je dobré zde uvést, že implicitní uživatelské jméno je pi a heslo raspberry.

3.1.2 *Pidora*

Pidora je jedna z dostupných distribucí OS *Linux*. Pro její implementaci je nejprve třeba stáhnout potřebný obraz disku, což je soubor ve formátu IMG. K tomu je nejvhodnější použít úložiště níže (aktuální k 18. 2. 2014).

http://downloads.raspberrypi.org/pidora_latest

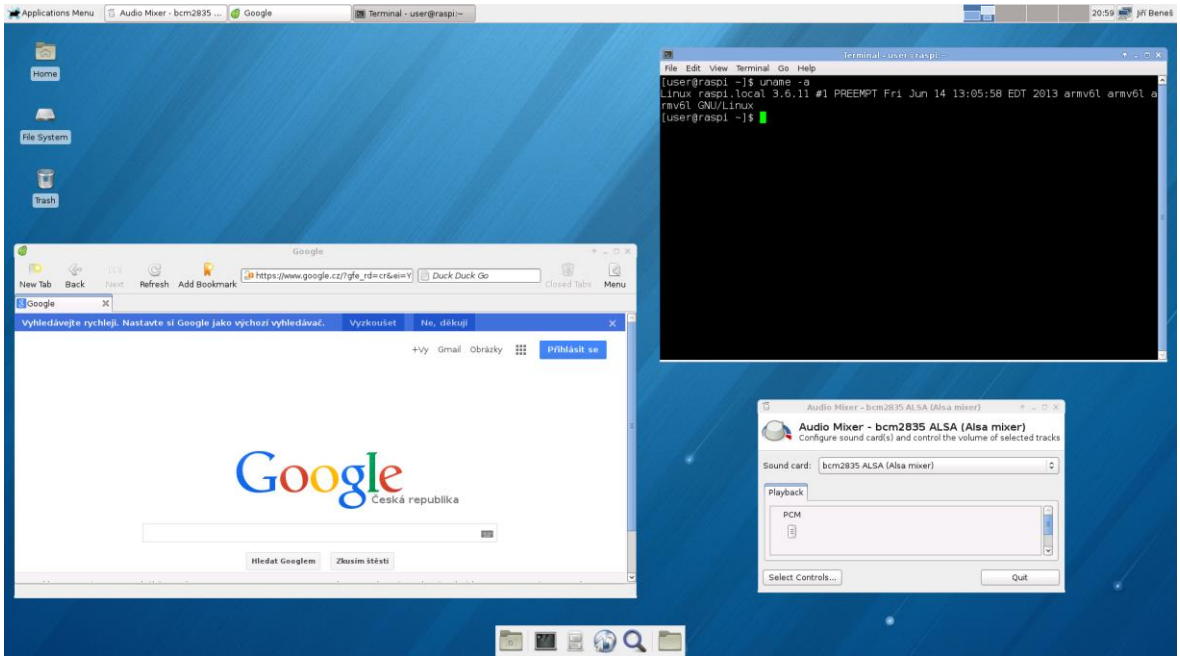
Pro praktickou realizaci byl použit obraz disku, který obsahuje distribuci *Pidora* ve verzi 18 R2C. Stažený soubor má pak název pidora-18-r2c. Dále je nutné tento obraz přenést na SDHC kartu. K tomu lze použít jak OS *Microsoft Windows*, tak OS *Linux*. Postup přenesení obrazu na SDHC kartu je analogický jako u distribuce *Raspbian* (viz podkapitola 3.1.1). Proto zde již nebude znovu popisován (pro následující text se tedy předpokládá SDHC karta s korektně přenesenou distribucí *Pidora*).

Po uvedení mikropočítače *Raspberry Pi* do aktivního stavu se nejprve provede kompletní kontrola systému a posléze se spustí samotná distribuce *Pidora*. Při prvním spuštění se zobrazí uvítací průvodce, v němž je možné postupně nastavit základní konfiguraci. Poté, co bude průvodce ukončen, dojde k automatickému spuštění grafického prostředí XFCE.

Prostřednictvím Terminálu lze poté pořídit informativní výpis o systému a dané platformě.

Linux raspi.local 3.6.11 #1 PREEMPT Fri Jun 14 13:05:58 EDT 2013 armv6l GNU/Linux

Snímek obrazovky spuštěného grafického prostředí XFCE v rámci distribuce *Pidora* OS *Linux*, je možné vidět na obr. č. 3.1.4.



Obr. č. 3.1.4 – Snímek obrazovky naimplementované distribuce *Pidora* OS *Linux*

3.1.3 *Arch Linux ARM*

Arch Linux ARM je, jak již název napovídá, jedna z dostupných distribucí OS *Linux*. Pro její implementaci je třeba stáhnout potřebný obraz disku, což je soubor ve formátu IMG. K tomu je nejvhodnější použít úložiště níže (aktuální k 18. 2. 2014).

```
http://downloads.raspberrypi.org/arch_latest
```

Pro praktickou realizaci byl použit obraz disku, který obsahuje distribuci *Arch Linux ARM* ve verzi January 2014. Stažený soubor má pak název ArchLinuxARM-2014.01-rpi. Dále je nutné tento obraz přenést na SDHC kartu. K tomu lze použít jak OS *Microsoft Windows*, tak OS *Linux*. Postup přenesení obrazu na SDHC kartu je analogický jako u distribuce *Raspbian* (viz podkapitola 3.1.1). Proto zde již nebude znovu popisován (pro následující text se tedy předpokládá SDHC karta s korektně přenesenou distribucí *Arch Linux ARM*).

Po uvedení mikropočítače *Raspberry Pi* do aktivního stavu se nejprve provede kompletní kontrola systému a posléze se spustí samotná distribuce *Arch Linux ARM* (v CLI režimu). Bude vyžadováno uživatelské jméno a heslo. Implicitně je nastaveno uživatelské jméno jako root a heslo taktéž root. Jedná se o „holou“ distribuci, tudíž bude třeba provést počáteční konfiguraci manuálně. V prvním kroku je vhodné nastavit lokalizaci, viz příkaz níže.

```
nano /etc/locale.gen
```

Spustí se textový editor, ve kterém je třeba odkomentovat řádky cs_CZ.UTF-8 UTF-8 a cs_CZ ISO-8859-2. Pro vygenerování lokalizačních souborů pak postačí napsat příkaz níže.

```
locale-gen
```

Dále je možné implementovat grafické prostředí (pro distribuci *Arch Linux* je doporučováno grafické prostředí LXDE). K tomu je nutné nainstalovat důležité komponenty projektu X.org[6]. Opět se použije příkaz uvedený níže.

```
pacman -S xorg-server xorg-xinit xorg-server-utils xterm
```

V dalším kroku je dobré nainstalovat systém pro renderování 3D grafiky, knihovnu Mesa.

```
pacman -S mesa
```

Následně pak nainstalovat ovladače grafické karty XF Video, monitorovací nástroj pro soubory a adresáře Gamin a nástroj D-bus, umožňující vzájemnou komunikaci aplikací.

```
pacman -S xf86-video-fbdev gamin dbus
```

[6] *Projekt X.org poskytuje open-source implementaci softwaru X Windows System (X11). Tedy softwaru, prostřednictvím kterého lze vytvořit grafické prostředí daného standardu.*

Vhodné je též nainstalovat univerzální kryptografickou knihovnu Libgcrypt.

```
pacman -S libgcrypt
```

Pokud všechny přechozí příkazy proběhly korektně, je nyní možné nainstalovat samotné prostředí LXDE. Použije se příkaz níže.

```
pacman -S lxde
```

Aby bylo možné grafické prostředí spustit standardním příkazem startx, je ještě třeba vytvořit soubor .xinitrc a vložit do něj řádek exec startlxde. Tedy použít příkaz níže.

```
echo 'exec startlxde' >> ~/.xinitrc
```

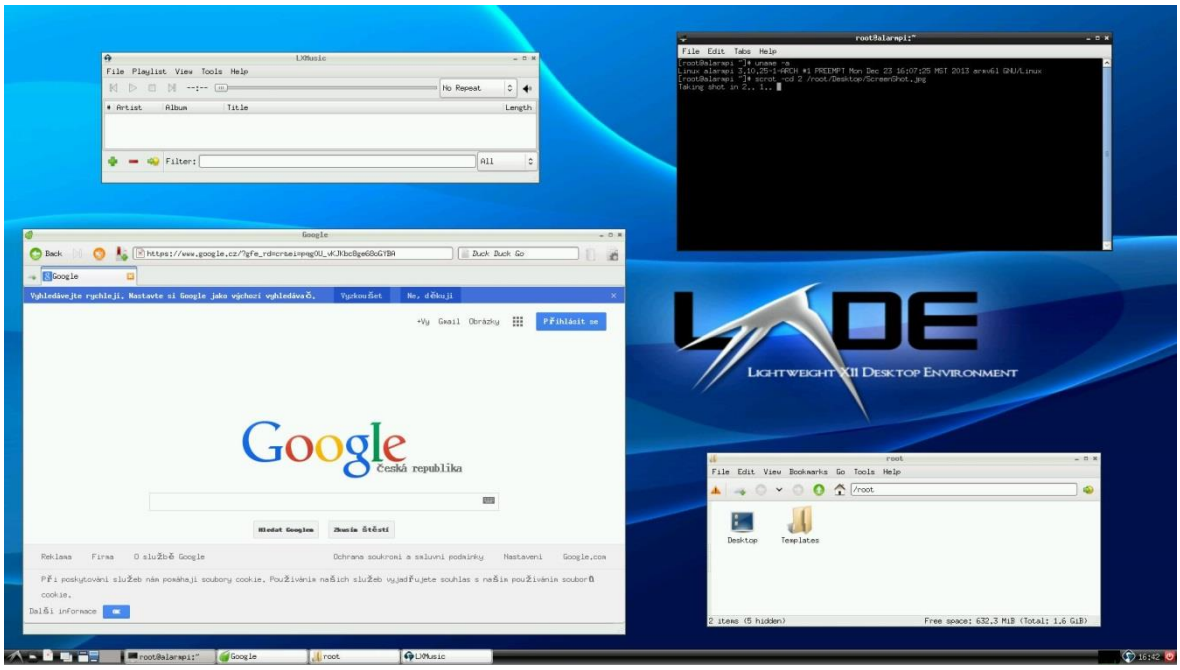
Grafické prostředí se pak spustí příkazem, který již byl výše zmiňován.

```
startx
```

Pro detailnější specifikaci implementované distribuce je níže uveden informativní výpis o systému a dané platformě, který byl pořízen prostřednictvím Terminálu.

```
Linux alarmpi 3.10.25-1-ARCH #1 PREEMPT Mon Dec 23 16:07:25 MST 2013 armv6l
GNU/Linux
```

Spuštěné grafické prostředí LXDE v rámci distribuce *Arch Linux ARM* OS *Linux*, lze v případě zájmu vidět na obr. č. 3.1.5.



Obr. č. 3.1.5 – Snímek obrazovky naimplementované distribuce *Arch Linux ARM* OS *Linux*

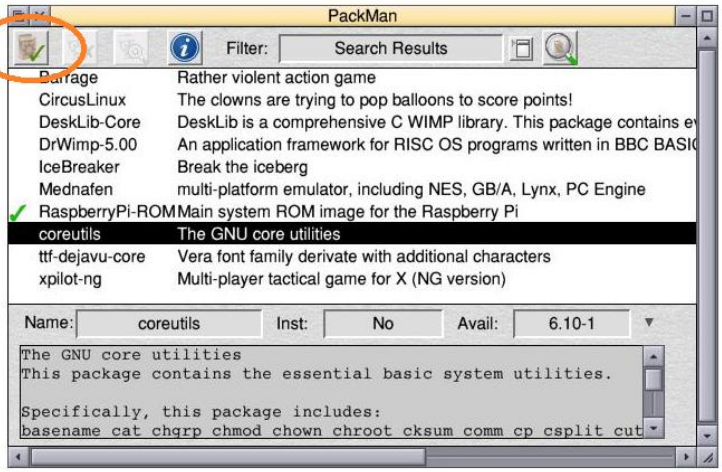
3.1.4 *RISC OS Pi*

RISC OS Pi je distribuce minimalistického a velmi rychlého operačního systému RISC OS, jejíž součástí je i grafické prostředí podporující „okénkový“ režim. Pro implementaci této distribuce je třeba stáhnout potřebný obraz disku, což je soubor ve formátu IMG. K tomu je nejvhodnější použít úložiště níže (aktuální k 18. 2. 2014).

http://downloads.raspberrypi.org/riscos_latest

Pro praktickou realizaci byl použit obraz disku, který obsahuje distribuci *RISC OS Pi* ve verzi 5.21 RC11. Stažený soubor má pak, dle data vydání, název riscos-2013-07-10-RC11. Dále je nutné tento obraz přenést na SDHC kartu. K tomu lze použít jak OS *Microsoft Windows*, tak OS *Linux*. Postup přenesení obrazu na SDHC kartu je analogický jako u distribuce *Raspbian* OS *Linux* (viz podkapitola 3.1.1). Proto zde již nebude znovu popisován (pro následující text se tedy předpokládá SDHC karta s korektně přenesenou distribucí *RISC OS Pi*).

Po uvedení mikropočítače *Raspberry Pi* do aktivního stavu se nejprve provede kompletní kontrola systému a posléze se načte samotná distribuce *RISC OS Pi*. Tím je v podstatě celá implementace hotova. Pro lepší unifikaci s OS *Linux* (alespoň z hlediska komparace informativního výpisu o systému a dané platformě s distribucemi OS *Linux*, uvedenými v předchozích kapitolách) je však dobré provést instalaci balíku coreutils[7] (GNU Core Utilities). K tomu poslouží nativní balíčkovací manažer, program *PackMan* (viz obr. č. 3.1.6). Implicitně by se měla ikona programu nacházet přímo na pracovní ploše (název !PackMan). V případě její absence lze program alternativně spustit přímo z umístění SDFS::RISCOSpi.$.Apps. Po spuštění programu postačí do vyhledávače napsat klíčové slovo „coreutils“ a posléze kliknout na tlačítko, které je na obr. č. 3.1.6 ohraničeno oranžovou barvou.



Obr. č. 3.1.6 – program *PackMan* (balíčkovací manažer)

[7] *Balík coreutils obsahuje základní nástroje jako cat, ls, pwd, uname apod. pro UNIX-like operační systémy.*

Tím se automaticky provede stažení potřebného balíčku a následná instalace (o čemž průběžně informuje externí okno programu, viz obr. č. 3.1.7). Po dokončení těchto operací je doporučeno provést restart systému.

Obr. č. 3.1.7 – Externí okno programu *PackMan*

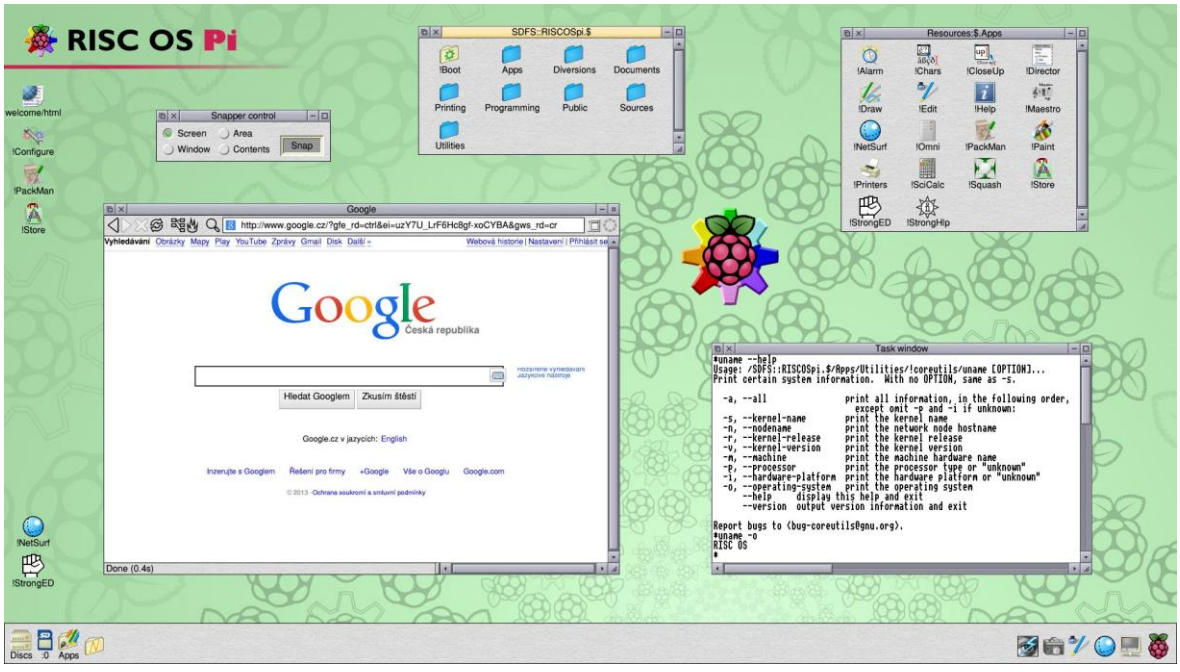
V distribuci *RISC OS Pi* je k dispozici speciální okno pro zadávání úkolů. Jedná se o jakousi obdobu Terminálu, který je k dispozici u OS *Linux*. Okno lze pak vyvolat kombinací kláves Ctrl + F12. Přičemž pro informativní výpis o systému a dané platformě postačí napsat příkaz níže (tedy stejný jako u OS *Linux*).

```
uname -a
```

Budou vypsány následující informace.

```
RISC OS RISCOSpi 5.21 1.0 arm RISC OS
```

Ukázka grafického prostředí distribuce *RISC OS Pi* ve formě snímku obrazovky je k dispozici na obr. č. 3.1.8.



Obr. č. 3.1.8 – Snímek obrazovky naimplementované distribuce *RISC OS Pi*

3.1.5 *Firefox OS*

Firefox OS (původním názvem *Boot to Gecko*) je operační systém, založený na OS *Linux*. Existuje i modifikovaná verze tohoto operačního systému, která je určena speciálně pro mikropočítač *Raspberry Pi* (viz podkapitola 2.3.5). Popis implementace v následujícím odstavci je tedy směřován právě k modifikované verzi operačního systému *Firefox OS*.

Pro implementaci operačního systému *Firefox OS* je zapotřebí mít k dispozici SDHC kartu, na níž je naimplementovaná distribuce *Raspbian* OS *Linux* (viz podkapitola 3.1.1). Další postup předpokládá aktivní stav mikropočítače *Raspberry Pi* s přihlášenou distribucí *Raspbian* (není nutné mít spuštěné grafické prostředí).

V prvé řadě je dobré vytvořit pracovní adresář, do kterého budou staženy soubory potřebné k běhu operačního systému *Firefox OS* (adresářová cesta např. v /home/pi/).

```
mkdir Firefox_OS
```

Následně se příkazem níže provede přesunutí do vytvořeného pracovního adresáře.

```
cd Firefox_OS
```

Do pracovního adresáře je pak nutné stáhnout soubory operačního systému *Firefox OS*.

```
wget http://romaxa.info/b2g/b2g-17.0a1.linuxgl-gnueabi-armhf_v6.tar.gz
```

Jedná se o zkomprimovaný archiv, tudíž je třeba provést rozbalení tohoto archivu a posléze se přesunout do adresáře s názvem b2g.

```
tar -xzvf b2g-17.0a1.linuxgl-gnueabi-armhf_v6.tar.gz
cd b2g/
```

Dále je ještě vhodné provést kontrolu závislosti na sdílených knihovnách. Prostřednictvím níže uvedeného příkazu je tedy možné zjistit, jaké dynamické knihovny potřebuje dynamicky linkovaný program b2g (což je hlavní spouštěcí program).

```
ldd b2g
```

Měly by být vypsány obdobné knihovny, jako jsou ty, uvedené níže.

```
/usr/lib/arm-linux-gnueabihf/libcofi_rpi.so (0xb6f65000)
libpthread.so.0 => /lib/arm-linux-gnueabihf/libpthread.so.0 (0xb6f3b000)
libdl.so.2 => /lib/arm-linux-gnueabihf/libdl.so.2 (0xb6f30000)
libstdc++.so.6 => /usr/lib/arm-linux-gnueabihf/libstdc++.so.6 (0xb6e63000)
libm.so.6 => /lib/arm-linux-gnueabihf/libm.so.6 (0xb6df2000)
libgcc_s.so.1 => /lib/arm-linux-gnueabihf/libgcc_s.so.1 (0xb6dca000)
libc.so.6 => /lib/arm-linux-gnueabihf/libc.so.6 (0xb6c9b000)
/lib/ld-linux-armhf.so.3 (0xb6f73000)
```

V dalším kroku se provede stažení dat uživatelského profilu (pro ukládání konfigurace).

```
wget http://romaxa.info/b2g/profile.tar.gz
```

Jedná se opět o zkomprimovaný archiv, příkazem níže se tedy provede rozbalení archivu.

```
tar -xzvf profile.tar.gz
```

Pokud všechny výše uvedené příkazy proběhly korektně, je operační systém *Firefox OS* připraven ke svému prvnímu spuštění. To lze provést následujícím příkazem.

```
./b2g --profile profile --screen=<value_1>x<value_2>
```

Parametr --profile profile nastaví, jaký uživatelský profil bude zaveden při spuštění operačního systému. Druhý výše uvedený parametr --screen=<value_1>x<value_2> pak nastaví maximální pracovní rozlišení obrazu (resp. v jak velkém okně bude vykreslováno grafické prostředí operačního systému *Firefox OS*), přičemž <value_1> udává hodnotu horizontálního rozlišení a <value_2> udává hodnotu vertikálního rozlišení (obě tyto hodnoty je třeba zadávat v pixelech). Ukončení činnosti operačního systému *Firefox OS* se provede kombinací kláves Ctrl + C (což způsobí návrat do prostředí distribuce *Raspbian* OS *Linux*).

Snímek obrazovky spuštěného operačního systému *Firefox OS* na mikropočítači *Raspberry Pi* lze vidět na obr. č. 3.1.9.

Obr. č. 3.1.9 – Snímek obrazovky naimplementovaného operačního systému *Firefox OS*

3.1.6 *CyanogenMod*

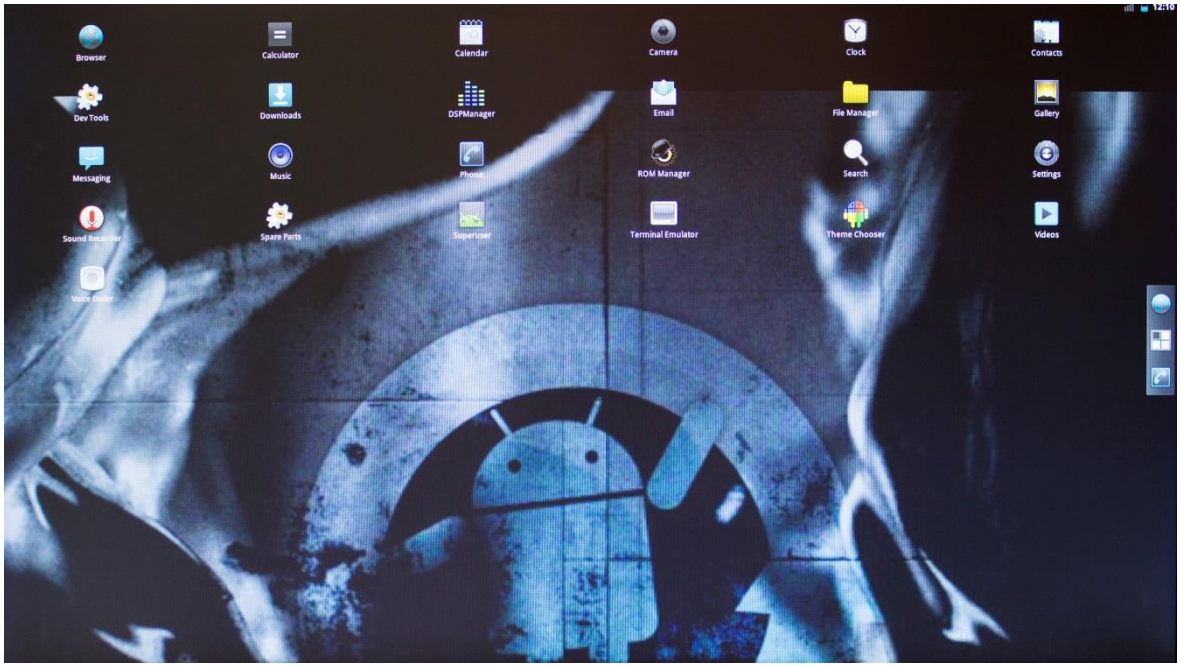
CyanogenMod je alternativní distribuce operačního systému *Android* (viz podkapitola 2.3.6). Pro implementaci této distribuce je třeba stáhnout potřebný obraz disku, což je soubor ve formátu IMG. K tomu lze použít například úložiště níže (aktuální k 18. 2. 2014).

http://rosefire.us/%7Erazdroid/aaa801/Gingerbread+EthernetManager.7z

Pro praktickou realizaci byl použit obraz disku, který obsahuje distribuci *CyanogenMod* ve verzi 7.2, navíc s doplňkem ve formě Ethernet menu (tato distribuce je založená na OS *Android* verze 2.3 s označením *Gingerbread*). Přičemž stažený soubor má pak název Gingerbread+EthernetManager. Dále je nutné tento obraz přenést na SDHC kartu. K tomu lze použít jak OS *Microsoft Windows*, tak OS *Linux*. Postup přenesení obrazu na SDHC kartu je analogický jako u distribuce *Raspbian* OS *Linux* (viz podkapitola 3.1.1). Proto zde již nebude znovu popisován (pro následující text se tedy předpokládá SDHC karta s korektně přenesenou distribucí *CyanogenMod*).

Po uvedení mikropočítače *Raspberry Pi* do aktivního stavu se nejprve provede kompletní kontrola systému a posléze se spustí samotná distribuce *CyanogenMod* (při této akci lze na obrazovce pozorovat průběh programu init, který inicializuje jednotlivé prvky potřebné pro běh distribuce *CyanogenMod*).

Snímek obrazovky spuštěné distribuce *CyanogenMod* OS *Android* je možné vidět na obr. č. 3.1.10.



Obr. č. 3.1.10 – Snímek obrazovky naimplementované distribuce *CyanogenMod* OS *Android*

3.1.7 *Plan 9 from Bell Labs*

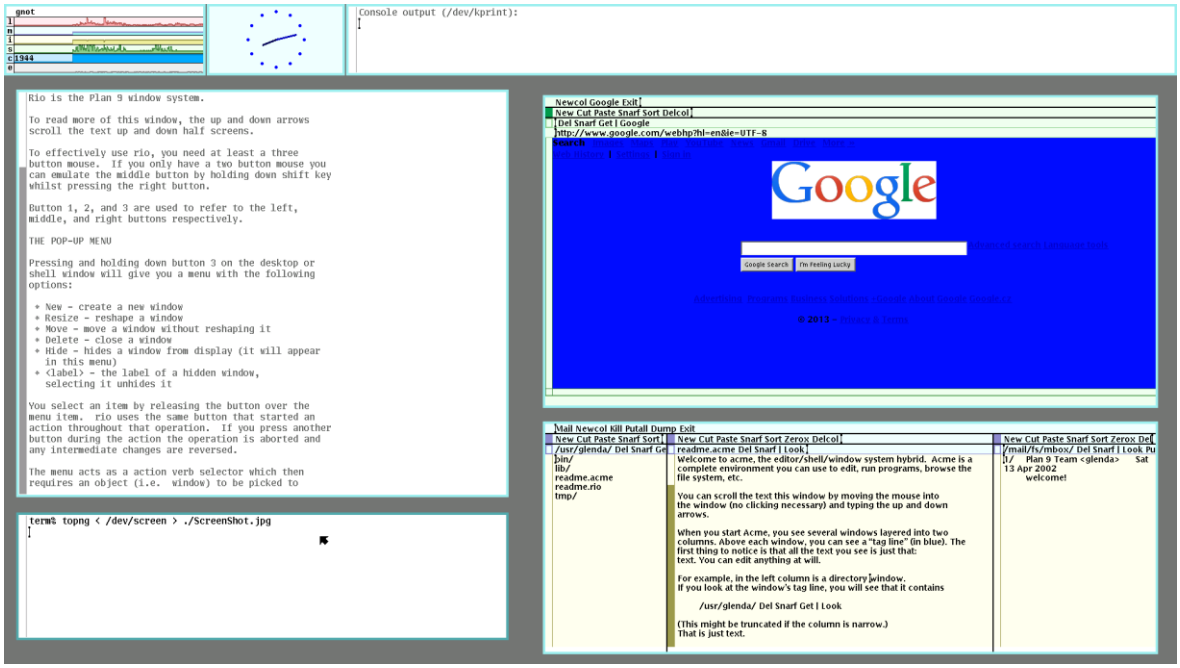
Plan 9 from Bell Labs je distribuovaný operační systém, vyvíjený jakožto nástupce operačního systému *Unix*. Pro implementaci tohoto operačního systému je třeba stáhnout potřebný obraz disku, což je soubor ve formátu IMG. K tomu je nejvhodnější použít oficiální úložiště, viz níže (aktuální k 18. 2. 2014).

http://plan9.bell-labs.com/sources/contrib/miller/9pi.img.gz

Pro praktickou realizaci byl použit obraz disku, který obsahuje operační systém *Plan 9 from Bell Labs* ve verzi 4 (Miller 9Pi). Přičemž stažený soubor má pak název pi. Dále je nutné tento obraz přenést na SDHC kartu. K tomu lze použít jak OS *Microsoft Windows*, tak OS *Linux*. Postup přenesení obrazu na SDHC kartu je analogický jako u distribuce *Raspbian* OS *Linux* (viz podkapitola 3.1.1). Proto zde již nebude znovu popisován (pro následující text se tedy předpokládá SDHC karta s korektně přeneseným operačním systémem *Plan 9 from Bell Labs*).

Po uvedení mikropočítače *Raspberry Pi* do aktivního stavu se nejprve provede kompletní kontrola systému a posléze se spustí samotný operační systém *Plan 9 from Bell Labs* (celý tento proces je velmi rychlý). Automaticky se spustí i nativní grafické prostředí, které používá „okénkový“ systém *Rio*.

Snímek obrazovky spuštěného operačního systému *Plan 9 from Bell Labs* je možné vidět na obr. č. 3.1.11.



Obr. č. 3.1.11 – Snímek obrazovky naimplementovaného OS *Plan 9 from Bell Labs*

3.1.8 *RaspBMC*

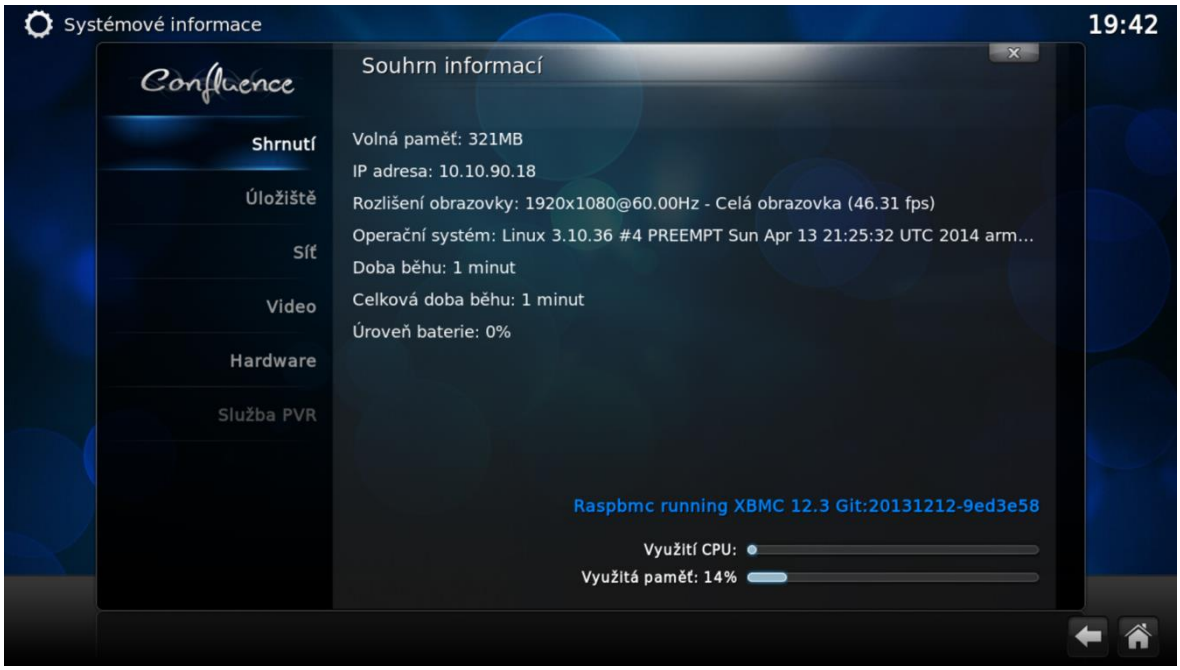
RaspBMC je distribuce OS *Linux*, která obsahuje XBMC systém (multimediální centrum). Pro její implementaci je nejprve třeba stáhnout potřebný obraz disku, což je soubor ve formátu IMG. K tomu je nejvhodnější použít úložiště níže (aktuální k 18. 2. 2014).

http://downloads.raspberrypi.org/raspbmc_latest

Pro praktickou realizaci byl použit obraz disku, který obsahuje distribuci *RaspBMC* ve verzi December 2013 (obsahující systém XBMC ve verzi 12.3 s označením *FRODO*). Stažený soubor má pak název `Raspbmc`. Dále je nutné tento obraz přenést na SDHC kartu. K tomu lze použít jak OS *Microsoft Windows*, tak OS *Linux*. Postup přenesení obrazu na SDHC kartu je analogický jako u distribuce *Raspbian* (viz podkapitola 3.1.1). Proto zde již nebude znovu popisován (pro následující text se tedy předpokládá SDHC karta s korektně přenesenou distribucí *RaspBMC*).

Po uvedení mikropočítače *Raspberry Pi* do aktivního stavu se automaticky spustí instalační program *Raspbmc Installer*, který provede základní konfiguraci operačního systému (založení nového uživatelského profilu apod.). Následně se prostřednictvím aktualizačního programu *Raspbmc Updater* provede stažení a nainstalování nejnovějšího programového vybavení (za předpokladu, že má mikropočítač *Raspberry Pi* přístup k Internetu). V poslední řadě je třeba zvolit vyhovující jazyk, čímž lze implementaci distribuce *RaspBMC* OS *Linux* na mikropočítač *Raspberry Pi* považovat za dokončenou.

Snímek obrazovky spuštěného XBMC systému s implicitním motivem *Confluence*, v rámci distribuce *RaspBMC* OS *Linux*, je možné vidět na obr. č. 3.1.12.



Obr. č. 3.1.12 – Snímek obrazovky naimplementované distribuce *RaspBMC* OS *Linux*

3.1.9 *OpenELEC*

OpenELEC je další možná distribuce OS *Linux*, která obsahuje XBMC systém (multimediální centrum) a podporuje i běh na mikropočítači *Raspberry Pi*. Pro její implementaci je nejprve třeba stáhnout potřebná zdrojová data. K tomu je nejvhodnější použít úložiště níže (aktuální k 18. 2. 2014).

http://downloads.raspberrypi.org/openelec_latest

Pro praktickou realizaci byla použita distribuce *OpenELEC* ve verzi 3.2.0, používající systém XBMC ve verzi 12.2 s označením *FRODO*. Stažený soubor ve formě komprimovaného archivu má pak název OpenELEC-RPi.arm-3.2.0. Pro vytvoření potřebné struktury na SDHC kartě je dále nutné provést níže uvedené instrukce (vzhledem k SH skriptu se kterým je třeba operovat, jsou tyto instrukce popisovány v prostředí OS *Linux*).

V prvé řadě se provede extrakce komprimovaného archivu. Následuje spuštění SH skriptu.

```
tar -xvf OpenELEC-RPi.arm-3.2.0.tar
sudo ./create_sdcard /dev/sd<X>
```

Přičemž za <X> se doplní správný znak pro udání cesty k SDHC kartě (problematika ohledně detekce korektní cesty k SDHC kartě je zmiňována již v podkapitole 3.1.1).

Po uvedení mikropočítače *Raspberry Pi* do aktivního stavu se po vstupní kontrole spustí průvodce, prostřednictvím kterého lze nastavit základní konfiguraci XBMC systému v rámci distribuce *OpenELEC* (implicitní motiv je zde opět *Confluence*, viz obr. č. 3.1.13).

Obr. č. 3.1.13 – Snímek obrazovky naimplementované distribuce *OpenELEC* OS *Linux*

3.2 Implementace multitaskingových RTOS na *Raspberry Pi*

Postup implementace multitaskingových operačních systémů reálného času na *Raspberry Pi* je v následujících podkapitolách členěn dle použitého softwaru určeného pro modifikaci. Nedílnou součástí implementace daného softwaru je i verifikace, zda implementovaný software vykazuje korektní chování, resp. funkcionalitu. Proto je kromě popisu implementace v podkapitolách, uvedena i verifikace funkcionality daného softwaru.

3.2.1 Aplikace frameworku *Xenomai*

Po vstupní analýze dostupných řešení, ohledně implementace frameworku *Xenomai* na mikropočítač *Raspberry Pi*, byl zvolen projekt *Machinoid*, jakožto optimální varianta pro následnou aplikaci. Hlavním faktorem pro tuto volbu byla koncepce projektu, která poskytuje předpřipravený software v podobě obrazu disku ve formátu IMG.

Projekt *Machinoid*, celým názvem „*Machinoid Hard Real-Time Distribution Optimized for Machines*“ je hard realtimová distribuce, která je primárně cílená pro aplikování v oblastech robotiky, CNC systémů a 3D tisku. Hlavní předností projektu jsou tyto aspekty:

- Stabilita systému
- Hard realtimová podpora
- Podpora „headless“ provozu[8]

Další předností, související s konkrétním zařízením určeným pro praktickou realizaci, je kompatibilita s distribucí *Raspbian* a podpora embedded hardwaru včetně LCD, I^2C, SPI apod. [27].

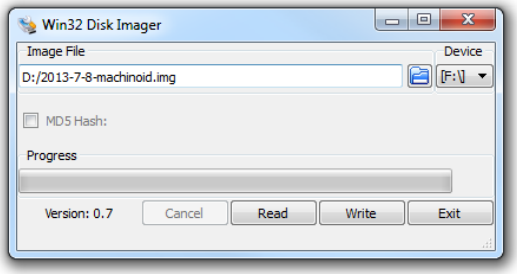
Aby bylo možné provést implementaci projektu *Machinoid*, je třeba stáhnout obraz disku ve formátu IMG, který je k dispozici na webových stránkách autora (aktuální k 18. 2. 2014).

http://machinoid.com

Pro praktickou realizaci byl použit obraz disku verze 2013-7-8. Tento IMG soubor obsahuje základní OS *Linux* s jádrem verze 3.5.7, připravený pro aplikaci frameworku *Xenomai*. Další postup implementace spočívá v přípravě vhodného formátu pro SDHC paměťovou kartu a následném překopírování potřebných dat, obsažených v IMG souboru. Pokud se pro výše uvedený postup použije OS *Microsoft Windows*, bude nejprve třeba stáhnout program *Win32 Disk Imager* (nebo jiný, poskytující obdobnou funkcionalitu). V rámci této práce byl zvolen program *Win32 Disk Imager* ve verzi 0.7 (vzhledem ke kompatibilitě s OS *Microsoft Windows 7* a korektní funkcionalitě při operaci

[8] *„Headless“ provozem se rozumí běh aplikací na systému, k němuž nejsou připojené periferie jako monitor, klávesnice či myš. Tento typ provozu se používá u embedded systémů.*

formátování a následném zápisu dat). Po spuštění programu stačí do kolonky „*Image File*“ napsat cestu k IMG souboru s projektem *Machinoid*, u selektoru „*Device*“ zvolit správnou jednotku SDHC karty a pak kliknout na kolonku „*Read*“ a posléze „*Write*“ (obr. č. 3.2.1).



Obr. č. 3.2.1 – Program *Win32 Disk Imager* připravený k zápisu projektu *Machinoid*

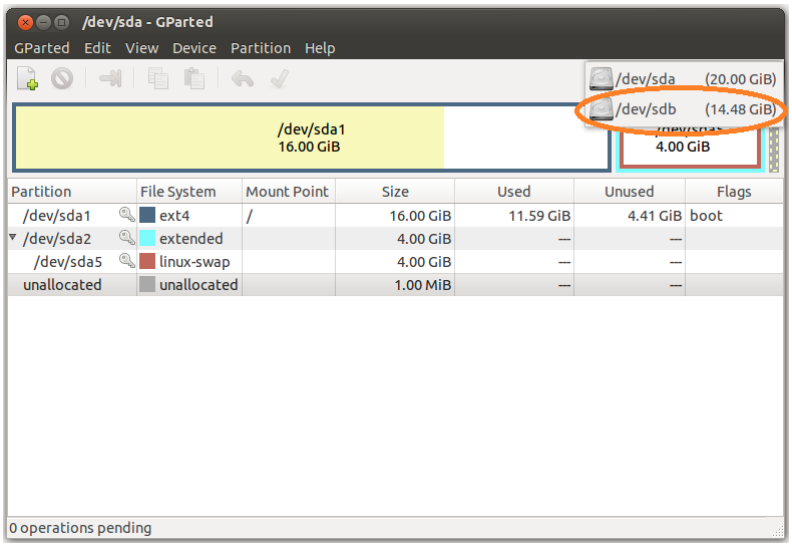
V případě použití OS *Linux* bude, v závislosti na zvolené distribuci a množství nainstalovaných balíčků, třeba doinstalovat program *GParted*.

```
sudo apt-get install gparted
```

Následně se provede spuštění programu.

```
sudo gparted
```

Měl by se nám naskytnout pohled podobný, jako je na obr. č. 3.2.2. V pravém horním selektoru je třeba zjistit správnou cestu k SDHC kartě. Pokud se například jedná o SDHC kartu s kapacitou 16 GB, bude se zřejmě jednat o cestu, která je ohraničená oranžovou barvou na tomtéž obrázku, tzn. /dev/sdb.



Obr. č. 3.2.2 – Program *GParted* použitý pro zjištění cesty k SDHC kartě

Další krok bude spočívat v přenesení dat z IMG souboru na SDHC kartu. To lze v OS *Linux* lehce provést prostřednictvím příkazu níže.

```
sudo dd bs=4M if=/home/user/Downloads/2013-7-8-machinoid.img of=/dev/sdb
```

Přičemž parametr bs=4M udává, že velikost bloku se nastaví na 4 MB (lze zadat například i hodnotu 1 MB, ale doba zápisu dat bude delší). Parametr if=<path> udává cestu k IMG souboru s projektem *Machinoid*. Poslední parametr of=<path> pro změnu udává cestu k SDHC kartě. Po provedení příkazu by se měl zobrazit výpis podobný jako je níže.

```
225+0 records in
225+0 records out
943718400 bytes (944 MB) copied, 214.109 s, 4.4 MB/s
```

Důležitá hodnota, která by měla být konstantní, je počet přenesených dat, u projektu *Machinoid* tedy 944 MB (pokud by tato hodnota byla odlišná, indikuje to chybu přenosu). Pokud hodnota souhlasí, tak je SDHC karta korektně připravena.

Dále je třeba SDHC kartu vložit do mikropočítače *Raspberry Pi* a po připojení potřebných periférií mikropočítač uvést do aktivního stavu. To by mělo způsobit načtení operačního systému *Linux*, respektive distribuce *Raspbian*. Pro samotnou implementaci frameworku *Xenomai* je třeba stáhnout instalační soubor *Xenomai* verze 2.6.2.1 (ADEOS-Ipipe patch). Do konzole, která se spustí po načtení operačního systému, tedy postačí napsat příkaz níže.

```
wget http://download.gna.org/xenomai/stable/xenomai-2.6.2.1.tar.bz2
```

Jedná se o zkomprimovaný archiv, proto je třeba nejdříve tento archiv rozbalit.

```
tar -jxvf xenomai-2.6.2.1.tar.bz2
```

V další fázi se, prostřednictvím příkazů níže, přesuneme do nově vytvořeného adresáře xenomai-2.6.2.1 a provedeme spuštění skriptu configure. Ten otestuje systém, nastaví korektně konfiguraci a posléze vytvoří soubory Makefile.

```
cd xenomai-2.6.2.1
./configure
```

Následuje kompilace zdrojových souborů a následné přesunutí spustitelných souborů do systémových adresářů (viz příkazy níže).

```
make
make install
```

Tímto postupem se docílí korektní modifikace distribuce *Raspbian*, frameworkem *Xenomai* na platformě *Raspberry Pi*.

3.2.2 Verifikace funkcionality frameworku *Xenomai*

Verifikace funkcionality implementovaného frameworku *Xenomai* lze rozdělit do dvou částí. Nejdříve se provede testování jádra, což bude první část verifikace. V druhé části bude pozornost soustředěna na testování user-space podpory.

V prvé řadě je proveden informativní výpis základních informací o operačním systému a dané platformě. Použije se příkaz níže.

```
uname -a
```

Přičemž budou vypsány následující informace.

```
Linux machinoid-rpi 3.5.7+ #3 PREEMPT Tue Jun 4 13:02:39 UTC 2014 armv6l GNU/Linux
```

Následuje kontrola řídících zpráv systému. Konkrétně zpráv týkajících se frameworku *Xenomai*. Příkaz tedy bude vypadat následovně.

```
dmesg | grep Xenomai
```

Po odfiltrování jsou k dispozici tyto výsledky.

```
[    1.216064] I-pipe: head domain Xenomai registered.
[    1.221004] Xenomai: hal/arm started.
[    1.224933] Xenomai: scheduling class idle registered.
[    1.230097] Xenomai: scheduling class rt registered.
[    1.240412] Xenomai: real-time nucleus v2.6.2.1 (Day At The Beach) loaded.
[    1.247403] Xenomai: debug mode enabled.
[    1.251783] Xenomai: starting native API services.
[    1.256661] Xenomai: starting POSIX services.
[    1.261186] Xenomai: starting RTDM services.
```

Žádná ze zpráv neindikuje chybový stav, přičemž vypsané informace jsou korektní. Z výše uvedených výsledků lze tedy konstatovat, že první část verifikace je úspěšně splněna (tedy část týkající se testování jádra).

Druhá část verifikace se týká testování latence systému prostřednictvím nativního softwaru frameworku *Xenomai* (resp. testování maximální latence, protože ta je důležitá vzhledem k vlastnostem RTOS). Funkcionalita tohoto testu spočívá v periodickém vypisování zprávy, která zobrazuje aktuální minimální, maximální a průměrnou latenci (pojmem latence v tomto případě rozumíme dobu mezi vygenerováním přerušení a okamžikem, kdy se o přerušení dozví user-space aplikace). Přičemž zpráva se implicitně vypisuje každou sekundu. Zkušební test lze spustit příkazem níže.

```
/usr/xenomai/bin/xeno latency -p 100 -T 10
```

Parametrem -p se nastavuje vzorkovací perioda (v mikrosekundách). Dalším parametrem, tedy -T se nastavuje doba, jak dlouho má tento test běžet (v sekundách).

Pro zvolené nastavení se tedy provede test, který vypadá následovně.

```
== Sampling period: 100 us
== Test mode: periodic user-mode task
== All results in microseconds
warming up...
RTT|  00:00:01  (periodic user-mode task, 100 us period, priority 99)
RTH|----lat min|----lat avg|----lat max|-overrun|---msw|---lat best|--lat worst
RTD|     2.000|     5.000|    35.000|        0|       0|     2.000|    35.000
RTD|     2.000|     8.000|    22.000|        0|       0|     2.000|    35.000
RTD|     1.000|     8.000|    23.000|        0|       0|     1.000|    35.000
RTD|     2.000|     8.000|    22.000|        0|       0|     1.000|    35.000
RTD|     2.000|     8.000|    27.000|        0|       0|     1.000|    35.000
RTD|     2.000|     8.000|    22.000|        0|       0|     1.000|    35.000
RTD|     2.000|     8.000|    21.000|        0|       0|     1.000|    35.000
RTD|     2.000|     8.000|    22.000|        0|       0|     1.000|    35.000
RTD|     2.000|     8.000|    36.000|        0|       0|     1.000|    36.000
-----|----------|----------|----------|---------|--------|--------------------------
RTS|     1.000|     7.000|    36.000|        0|       0|   00:00:10/00:00:10
```

Z testu lze zjistit, že maximální latence v době měření byla 36 mikrosekund (tento údaj je nejdůležitější), minimální latence byla 1 mikrosekunda a průměrná latence byla 7 mikrosekund. Přičemž test běžel deset sekund. Dále je důležité zkontrolovat, zda jsou vypsané hodnoty korektní. Tedy zda se nezobrazily chybové zprávy či neočekávané hodnoty. Jedná se ovšem pouze o demonstrační příklad. Tento test je třeba nechat spuštěný alespoň 10 minut, abychom dostali nějaké relevantnější výsledky. Níže je uveden test, který se stejnou vzorkovací periodou na mikropočítači *Raspberry Pi* běžel čtvrt hodiny.

```
== Sampling period: 100 us
== Test mode: periodic user-mode task
== All results in microseconds
warming up...
RTT|  00:00:01  (periodic user-mode task, 100 us period, priority 99)
RTH|----lat min|----lat avg|----lat max|-overrun|---msw|---lat best|--lat worst
RTD|     2.000|     4.000|    22.000|        0|       0|     2.000|    22.000
RTD|     2.000|     5.000|    23.000|        0|       0|     2.000|    23.000
RTD|     2.000|     5.000|    21.000|        0|       0|     2.000|    23.000
RTD|     2.000|     5.000|    23.000|        0|       0|     2.000|    23.000
[…]
RTD|     2.000|     5.000|    21.000|        0|       0|     1.000|    41.000
RTD|     2.000|     5.000|    23.000|        0|       0|     1.000|    41.000
RTD|     2.000|     5.000|    35.000|        0|       0|     1.000|    41.000
-----|----------|----------|----------|---------|--------|--------------------------
RTS|     1.000|     5.000|    41.000|        0|       0|   00:15:00/00:15:00
```

U čtvrthodinového testu je vidět mírné zvýšení maximální latence na 41 mikrosekund. To už lze brát jako relevantnější hodnotu. Stále to ale není „zaručená“ hodnota. Test sice běžel čtvrt hodiny, ale v klidovém režimu (tzn. systém nebyl během testu nijak zatěžován).

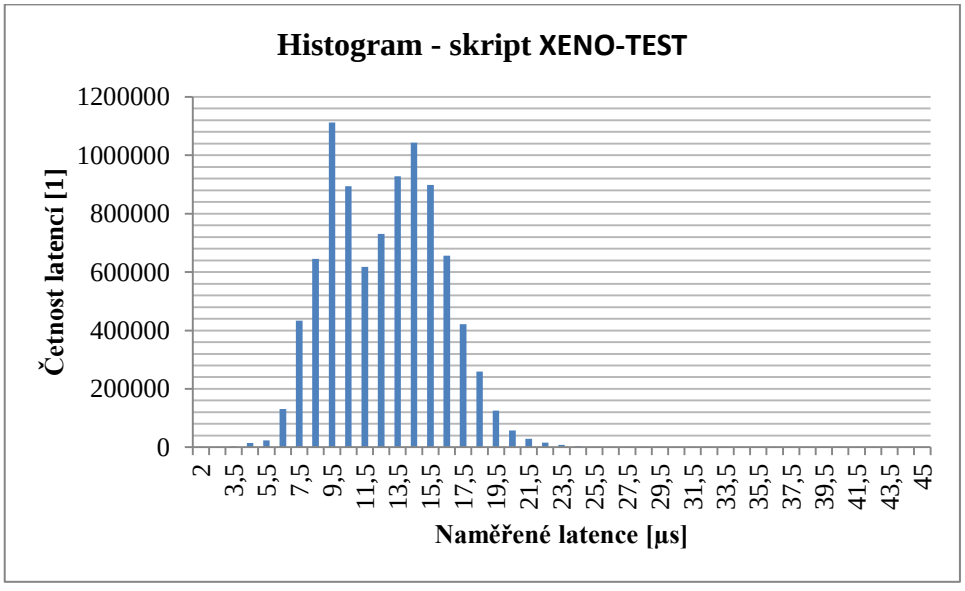
Poslední verifikace implementovaného frameworku *Xenomai* tedy spočívá v provedení komplexního testu, který bude spuštěn paralelně se zátěží. Jako zátěž bude použit nástroj DOHELL. Ten zatěžuje systém prostřednictvím běžně dostupných příkazů (v rámci OS *Linux*), které budou paralelně spuštěné k testu latence xeno latency (viz předchozí strana). Spouštěcí příkaz bude mít následující podobu.

```
/usr/xenomai/bin/./xeno-test -l "dohell 900" -p 100 -g histo
```

Příkaz výše spustí skript XENO-TEST, který sekvenčně provede řadu testů a v konečné fázi spustí paralelně test latence se zátěží. Přičemž zátěž, tedy nástroj DOHELL je nastaven jako dohell 900. To znamená, že doba běhu nástroje je 900 sekund (jedná se o implicitní hodnotu). Hodnota vzorkovací periody u testu latence je pak opět 100 mikrosekund. Níže jsou vypsány výsledné hodnoty z konečného testu latence při zátěži (kompletní výpis všech provedených testů skriptem XENO-TEST je k dispozici v příloze A).

```
RTH|----lat min|----lat avg|----lat max|-overrun|---msw|---lat best|--lat worst
---|----------|----------|----------|--------|------|-------------------------
RTS|      2.000|     12.000|     44.000|       0|     0|    00:15:05/00:15:05
```

Z výpisu je vidět, že maximální latence je při zátěži 44 mikrosekund (tzn. oproti testování bez zátěže je to zanedbatelná změna). Lze tedy konstatovat, že systém má realtimové vlastnosti a implementace frameworku *Xenomai* modifikací distribuce *Raspbian* na platformě *Raspberry Pi* byla úspěšná. Pro zajímavost je na obr. č. 3.2.3 uveden histogram naměřených latencí prostřednictvím výše popisovaného skriptu XENO-TEST.



Obr. č. 3.2.3 – Histogram z výstupních hodnot nástroje XENO-TEST

3.2.3 Aplikace patche *PREEMPT RT*

Princip implementace patche *PREEMPT RT* spočívá v modifikaci jádra OS *Linux*. Na standardní jádro OS *Linux* se aplikuje výše zmíněný patch, čímž dojde k modifikaci jeho realtimových vlastností. Tímto modifikovaným jádrem se pak nahradí implicitní jádro používané distribucí *Raspbian* (OS *Linux*). Získáme tak OS *Linux*, resp. jeho distribuci *Raspbian*, která je použitelná jakožto RTOS. Níže je uveden postup, jak výše uvedené implementace docílit.

Pro snazší pochopení problému je postup prezentován autenticky přesně dle průběhu praktické realizace tak, jak byla prováděna na mikropočítači *Raspberry Pi*. Důležité je také zmínit, že následující postup je popisován v rámci prostředí operačního systému *Linux* (při praktické realizaci byla konkrétně použita distribuce *Ubuntu 13.10 Saucy Salamander*). V prvé řadě je tedy vhodné vytvořit adresář pro zdrojové soubory *PREEMPT RT* patche a adresář pro potřebné moduly. Předpokladem je aktuální umístění v /home/user/Desktop/.

```
mkdir preempt_rt_project
cd preempt_rt_project
mkdir rpi_modules
```

Dále je třeba vytvořit kopii existujícího repozitáře Git (tzn. naklonovat), přesněji vytvořit kopii projektů *linux* a *tools*. Přičemž projekt *linux* obsahuje veškeré potřebné zdrojové soubory pro kompilaci jádra. Projekt *tools* pak obsahuje toolchain[9], z něhož se využije vhodný kompilátor.

```
git clone https://github.com/raspberrypi/linux.git
git clone https://github.com/raspberrypi/tools.git
```

Následuje vytvoření konfiguračního souboru prostřednictvím překopírování obsahu ze souboru bcmrpi_cutdown_defconfig do prázdného souboru .config. Konfigurační soubor .config pak obsahuje důležité údaje, které jsou potřebné pro kompilaci jádra. Zde existuje i alternativní řešení, které spočívá v překopírování obsahu souboru config.gz, přímo z funkční distribuce *Rasbian*, běžící na mikropočítači *Raspberry Pi* (přičemž soubor má umístění /proc/config.gz).

```
cd linux
cp arch/arm/configs/bcmrpi_cutdown_defconfig .config
```

V dalším kroku se provede stažení samotného patche *PREEMPT RT* a to ve verzi 3.10.32-rt30. Verze patche je závislá na verzi jádra, na které bude tento patch aplikován (tzn. dle verze jádra obsaženého v naklonovaném projektu *linux* – viz výše). V době provádění praktické realizace byla v rámci projektu *linux* k dispozici nejvyšší verze jádra 3.10.32 (proto byla zvolena i výše uvedená verze patche *PREEMPT RT*).

[9] *Toolchain je kolekce nástrojů určených k programování (typicky se jedná o kompilátory a linkery).*

Patch lze stáhnout prostřednictvím příkazu níže (URL adresa se může časem změnit, resp. umístění daného patche v adresářové struktuře).

```
wget https://www.kernel.org/pub/linux/kernel/projects/rt/3.10/older/patch-3.10.32-rt30.patch.gz
```

Následuje rozbalení archivu a aplikování patche *PREEMPT RT* na jádro OS *Linux*.

```
gunzip patch-3.10.32-rt30.patch.gz
patch -p1 < patch-3.10.32-rt30.patch
```

Je však možné, že některé soubory byly při aplikování patche odmítnuty. Pokud k této situaci dojde, vytvoří se tzv. REJ soubory. Proto je dobré provést kontrolu těchto souborů, k čemuž slouží příkaz níže.

```
find ./ -name *.rej -print
```

V případě praktické realizace např. došlo k odmítnutí souboru Makefile. Výpis má tedy následující formu.

```
./drivers/misc/Makefile.rej
```

Nutností je oprava těchto souborů, která spočívá v ruční modifikaci původních souborů pomocí výstupu vygenerovaného prostřednictvím nástroje *Diff*[10] (tedy čistý PATCH soubor, který byl aplikován na jádro). V podstatě se jedná o přidání/odebrání určitých řádků kódu v daném souboru.

Dále je vhodné přidat si adresářové cesty do proměnných ARCH, CROSS_COMPILE a INSTALL_MOD_PATH. Všechny tyto proměnné budou v dalším postupu použity. Proměnná ARCH pro nastavení architektury, pro kterou se bude jádro kompilovat (tedy v případě mikropočítače *Raspberry Pi* to bude architektura ARM). Proměnná CROSS_COMPILE pro zvolení správného křížového[11] kompilátoru a proměnná INSTALL_MOD_PATH pro nastavení adresářové cesty, kam se nainstalují potřebné moduly. Příkazy lze vidět níže.

```
export ARCH=arm
export CROSS_COMPILE=/home/user/Desktop/preempt_rt_project/tools/arm-bcm2708/gcc-linaro-arm-linux-gnueabihf-raspbian/bin/arm-linux-gnueabihf-
export INSTALL_MOD_PATH=/home/user/Desktop/preempt_rt_project/rpi_modules
```

V dalším kroku bude třeba spustit grafické prostředí pro konfiguraci jádra. Jedná se o menu, prostřednictvím kterého lze modifikovat soubor .config (který byl předchozími příkazy již vytvořen). Lze samozřejmě tento soubor modifikovat i přes textový editor, ale

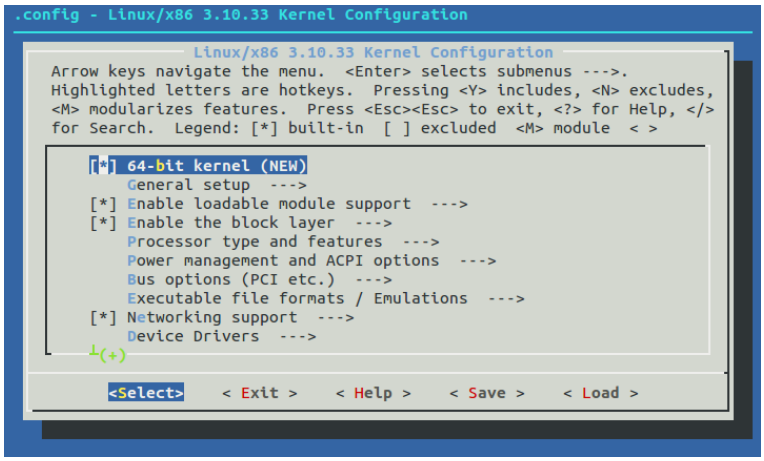
[10] *Nástroj Diff slouží ke zjištění rozdílů mezi textovými soubory. Princip spočívá ve vypisování řádků, kterými se dané soubory liší. Výstupem nástroje Diff pak je soubor ve formátu PATCH (*.patch).*

[11] *Křížový kompilátor umí generovat spustitelný kód pro jinou platformu, než na které probíhá samotná kompilace.*

byla by to zbytečně složitá a nepřehledná volba. Grafické prostředí pro konfiguraci jádra tedy spustíme následujícím příkazem (k provedení příkazu je zapotřebí mít nainstalovanou knihovnu ncurses-devel[12]).

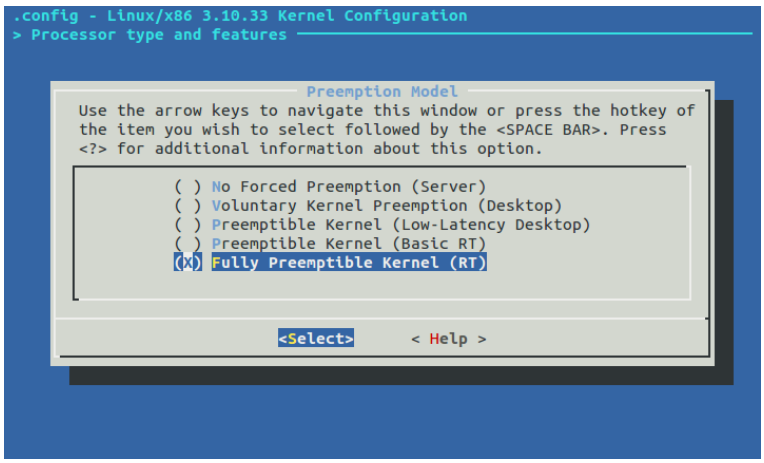
```
make menuconfig
```

Příkazem se vyvolá konfigurační menu, které je vidět na obr. č. 3.2.4. Nyní je třeba se přesunout do menu s názvem „*Processor type and features*“ a dále pak do submenu s názvem „*Preemption Model*“.



Obr. č. 3.2.4 – Hlavní stránka konfiguračního menu

Dojde k zobrazení submenu (viz obr. č. 3.2.5), které je určené pro výběr konkrétního preemptivního modelu. Aplikováním patche *PREEMPT RT* v tomto submenu přibyl nový model, který nese označení „*Fully Preemptible Kernel (RT)*“. Tento model je třeba zvolit.



Obr. č. 3.2.5 – Výběr preemptivního modelu (konfigurační menu)

[12] *Knihovna ncurses-devel slouží pro optimalizované ovládání textových oken. Pro instalaci této knihovny postačí do konzole napsat příkaz „sudo apt-get install libncurses5-dev“ (OS Linux, distribuce Ubuntu).*

Prostřednictvím konfiguračního menu lze nastavit i širokou škálu jiných realtimových vlastností (např. časovače s vysokým rozlišením apod.). Nastavení plně preemptivního modelu je ale vlastností klíčovou. Po uložení zvolené konfigurace se změny zapíší do souboru .config a jádro je tím připravené ke kompilaci. Kompilace jádra se tedy může níže uvedeným příkazem spustit (parametr -j2 říká, že se úloha rozdělí na dva podprocesy).

```
make -j2 Image
```

Kompilace jádra OS *Linux* je časově náročná, přičemž i dost vytěžuje systém (dle použité platformy může kompilace trvat i několik hodin). Po skončení kompilace dojde k vytvoření souboru Image, který se bude nacházet v arch/arm/boot/.

Dále se provede kompilace modulů, což jsou soubory, které jsou součástí jádra (jsou načteny za běhu dle aktuální potřeby). Čím více je těchto modulů, tím déle bude kompilace trvat. Opět se použije příkaz uvedený níže.

```
make -j2 modules
```

Jako poslední je třeba provést instalaci zkompilovaných modulů. Tzn. zkopírovat všechny tyto moduly do adresáře definovaným v proměnné INSTALL_MOD_PATH. K tomu poslouží následující příkaz.

```
make -j2 modules_install
```

Nyní tedy stačí přetransformovat soubor Image, který vznikl kompilací jádra, na soubor, který bude ve formátu IMG (k provedení příkazu je zapotřebí mít nainstalovanou knihovnu python[13]).

```
cd ..
cd tools/mkimage/
python imagetool-uncompressed.py
/home/user/Desktop/preempt_rt_project/linux/arch/arm/boot/Image
```

Po korektním provedení příkazu se v aktuálním adresáři vytvoří soubor kernel.img. Tím je vše připraveno pro vlastní modifikaci distribuce *Raspbian*. V prvním kroku bude třeba vložit SDHC kartu s naimplementovanou distribucí *Raspbian* (viz podkapitola 2.3.1) do zařízení, na kterém byly prováděny všechny předchozí příkazy popisované v této podkapitole (tzn. na kterém je k dispozici soubor kernel.img a adresář rpi_modules). Bude následovat odstranění původního jádra distribuce *Raspbian* a nahrazení jádrem, které bylo předchozími příkazy modifikováno patchem *PREEMPT RT* (viz příkazy níže).

```
rm /media/user/boot/kernel.img
cp kernel.img /media/user/boot/
```

[13] *Jedná se o standardní knihovnu dodávanou k programovacímu jazyku Python. Pro instalaci této knihovny postačí do kozole napsat příkaz „sudo apt-get install python“ (OS Linux, distribuce Ubuntu).*

Pro korektní funkcionalitu jádra je dále třeba zkopírovat i moduly, které jsou uložené v adresáři rpi_modules.

```
sudo cp -rp ../rpi_modules/lib/modules/3.10.32-rt30+/ /media/<SDHC>/lib/modules
```

SDHC karta bude v případě distribuce *Raspbian* rozdělena na dva oddíly. Přičemž názvy oddílů by měly být podobné jako níže.

- boot
- fc254b57-8fff-4f96-9609-ea202d871acf

Název druhého z uvedených oddílů je třeba použít místo označení <SDHC> ve výše uvedeném příkazu.

Jako poslední věc je doporučováno deaktivovat mód nízké latence pro ovladač SDHCI. Lze tak obejít bug způsobující problémy s připojováním kořenového oddílu SDHC karty. To se provede přidáním řádky, obsahující deaktivující informaci, do souboru cmdline.txt.

```
echo 'sdhci_bcm2708.enable_llm=0 ' >> /media/user/boot/cmdline.txt
```

Nyní je SDHC karta připravena pro vložení do mikropočítače *Raspberry Pi*. Po uvedení mikropočítače do provozu se spustí distribuce *Raspbian*, která poběží v plně preemptivním módu. Bude tedy „obohacena“ o realtimové vlastnosti.

3.2.4 Verifikace funkcionality patche *PREEMPT RT*

Verifikace funkcionality implementovaného patche *PREEMPT RT* bude rozčleněna do více částí. Nejdříve se provede informativní výpis základních informací o operačním systému a dané platformě. Použije se příkaz níže.

```
uname -a
```

Přičemž budou vypsány následující informace.

```
Linux raspberrypi 3.10.33-rt30+ #1 PREEMPT RT Mon Mar 10 13:19:19 PDT 2014 armv6l
GNU/Linux
```

Aplikováním patche by mělo automaticky u verze jádra přibýt označení RT (jak lze vidět výše).

Pro další verifikaci se použije specifický program, který zjistí, zda se opravdu jedná o realtimové jádro. Program funguje na principu detekce validních hodnot v souboru realtime, který je umístěn v /sys/kernel/ (zdrojový kód programu je uveden v příloze B). V případě, že program opravdu zjistí přítomnost realtimového jádra, vypíše následující.

```
this is a PREEMPT RT kernel
```

Jedna z vlastností patche *PREEMPT RT* je možnost spouštět obsluhy IRQ ve vláknech. Pokud byl tento patch řádně aplikován, musí tuto vlastnost získat i modifikované jádro. To lze snadno ověřit příkazem níže.

```
ps -eo pid,pri,rtprio,cmd | grep irq
```

Příkaz vypíše stavy všech aktivních procesů, které byly odfiltrovány řetězcem irq, viz níže.

```
  3    41     1     [ksoftirqd/0]
 14    90     50    [irq/65-ARM Mail]
 32    90     50    [irq/16-bcm2708_]
 36    90     50    [irq/66-VCHIQ do]
 38    90     50    [irq/32-dwc_otg]
 39    90     50    [irq/32-dwc_otg_]
 40    90     50    [irq/32-dwc_otg_]
 42    90     50    [irq/77-bcm2708_]
 43    90     50    [irq/84-mmc0]
2180   90     50    [irq/83-uart-pl0]
2371   19     -     grep irq
```

První sloupec zprava obsahuje identifikační čísla procesů, druhý obsahuje priority procesů a třetí obsahuje realtimové priority procesů. V posledním sloupci jsou pak uvedena jména procesů. Procesy, které nesou jména ve tvaru [irq/<číslo>-<popis>] jsou konsekvencí právě implementace patche *PREEMPT RT*. Každý tento proces pak obsahuje jedno nebo více vláken. Pokud by nás zajímala statistika vláken pro konkrétní aktivní proces (např. pro proces [irq/65-ARM Mail]), tak postačí použít příkaz níže.

```
top -H -p 14
```

Dostaneme kompletní statistiku pro proces s identifikačním číslem 14, tedy pro proces [irq/65-ARM Mail]. Ze statistiky lze vyčíst, že proces obsahuje jedno vlákno, které je momentálně ve spícím stavu.

```
top - 22:51:10 up 16 min,  2 users,  load average: 0.37, 0.34, 0.23
Threads:  1 total,  0 running,  1 sleeping,  0 stopped,  0 zombie
%Cpu(s): 12.2 us, 25.3 sy,  0.0 ni, 61.2 id,  0.0 wa,  0.0 hi,  1.2 si,  0.0 st
KiB Mem:      450068 total,  38248  used,   311820 free,   13120 buffers
KiB Swap:     102396 total,      0  used,   102396 free,   59176 cached

  PID USER      PR  NI  VIRT  RES  SHR S  %CPU %MEM    TIME+  COMMAND
   14 root      -51  0     0    0    0 S   0.0  0.0   0:00.00 irq/65-ARM Mail
```

Výše uvedeným postupem se tedy dokázalo, že inkriminovaná vlákna (resp. v tomto konkrétním příkladu vlákno) jsou k dispozici. V dalším kroku verifikace se lze přesunout k samotnému testování, resp. měření latence systému (pojmem latence v tomto případě rozumíme dobu mezi vygenerováním přerušení a okamžikem, kdy se o přerušení dozví

user-space aplikace). K samotnému testování se použije nástroj *Cyclictest*, který byl pro tyto účely navržen (viz podkapitola 2.5.1). Co se týká formy testování, tak nejprve je proveden čtvrthodinový test na systému, který není nijak zatěžován. Použije se příkaz níže.

```
cyclictest -p 99 -t5 -n
```

Příkaz s uvedenými parametry spustí pět vláken (-t5), přičemž každé toto vlákno bude mít realtimovou prioritu nejvýše 99 (-p 99). Parametr -n pak říká, že se pro testování použije clock_nanosleep[14] namísto intervalových POSIX časovačů.

Vsuvkou zde budiž ještě krátká informace o tom, proč je třeba, aby výše uvedená vlákna měla realtimovou prioritu 99 (obdobné nastavení priority bylo použito i při praktické realizaci testovacích scénářů, proto je dobré následujícímu textu věnovat větší pozornost). Priorita 99 je zvolena záměrně, protože je nutností, aby priorita vláken nástroje *Cyclictest* byla v okamžiku testování na daném systému nejvyšší. Tím lze dosáhnout relevantnějších výsledků, konkrétně přesnějších naměřených latencí. Například jaderná vlákna realizující obsluhu IRQ mají v případě aplikovaného patche *PREEMPT RT* realtimovou prioritu 50. Je tedy potřeba, aby vlákna nástroje *Cyclictest* měla tuto prioritu znatelně vyšší.

```
# /dev/cpu_dma_latency set to 0us
policy: fifo: loadavg: 2.23 2.37 2.16 2/178 2583

T: 0 ( 2575) P:99 I:1000 C: 877939 Min:    18 Act:   37 Avg:   47 Max:    131
T: 1 ( 2576) P:99 I:1500 C: 585293 Min:    19 Act:   44 Avg:   46 Max:    129
T: 2 ( 2577) P:99 I:2000 C: 438970 Min:    21 Act:   35 Avg:   39 Max:     74
T: 3 ( 2578) P:99 I:2500 C: 351176 Min:    21 Act:   55 Avg:   47 Max:    128
T: 4 ( 2579) P:99 I:3000 C: 292647 Min:    21 Act:   35 Avg:   39 Max:     86
```

Po čtvrt hodině testování lze výše vidět výsledky. Jednotlivé řádky, začínající písmenem T, označují vlákna č. 0 až č. 4. Důležité jsou pak údaje jako minimální, aktuální, průměrná a maximální latence. Nejvyšší hodnota latence byla naměřena u vlákna č. 0, tedy 131 mikrosekund. V tomto testu se tedy jedná o nejhorší možný případ.

Další čtvrthodinový test spočívá v použití stejného příkazu jako výše (i stejnými parametry), ale při zátěži. Pro potřeby simulace zátěže byl použit nástroj *Cache Calibrator* (viz podkapitola 2.5.3), který umí produkovat silné zatížení vyrovnávací paměti, což má zásadní vliv na latenci systému (*Cache Calibrator* je doporučovaným nástrojem pro simulaci zátěže při verifikaci patche *PREEMPT RT*). Nástroj byl po celou dobu testu cyklicky spouštěn s následujícími parametry.

```
cache_calibrator 700 100M output
```

Parametry, u příkazu výše, nastaví frekvenci CPU na 700 MHz a velikost použité paměti na 100 MB.

[14] *clock_nanosleep umožňuje provést uspání vláken s vysokou přesností, resp. s přesností na nanosekundy.*

Po čtvrthodinovém paralelním běhu nástroje *Cyclictest* s nástrojem *Cache Calibrator* jsou k dispozici výsledky níže.

```
# /dev/cpu_dma_latency set to 0us
policy: fifo: loadavg: 3.02 2.96 1.98 4/179 2537

T: 0 ( 2520) P:99 I:1000 C: 877367 Min:    17 Act:   42 Avg:   45 Max:    119
T: 1 ( 2521) P:99 I:1500 C: 584911 Min:    18 Act:   65 Avg:   51 Max:    122
T: 2 ( 2522) P:99 I:2000 C: 438683 Min:    18 Act:   43 Avg:   41 Max:    102
T: 3 ( 2523) P:99 I:2500 C: 350946 Min:    19 Act:   41 Avg:   42 Max:    132
T: 4 ( 2524) P:99 I:3000 C: 292456 Min:    19 Act:   33 Avg:   43 Max:    104
```

Z výsledků lze vidět, že nejvyšší hodnota latence byla naměřena u vlákna č. 3 a sice 132 mikrosekund. Je tedy zřejmé, že v tomto testu se jedná o nejhorší možný případ.

Pokud bychom provedli srovnání nejvyšší hodnoty latence u testu bez zatížení s testem se zatížením, zjistíme, že rozdíl činí pouhá 1 mikrosekunda. Je tedy vidět, že ani při silné zátěži nedojde k výraznému vlivu na latenci systému. Ze všech uskutečněných testů lze tedy konstatovat, že distribuce *Raspbian* operačního systému *Linux* získala aplikací patche *PREEMPT RT* realtimové vlastnosti.

3.3 Testování vybraných multitaskingových OS a RTOS

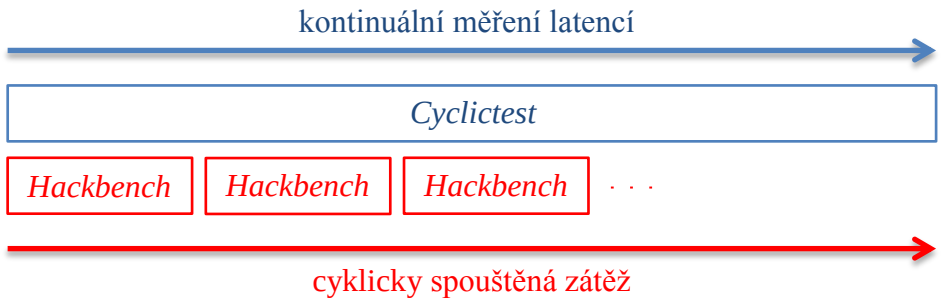
Samotné testování vybraných multitaskingových OS a RTOS se řídí předem definovanými testovacími scénáři (viz kapitola 2.6). Jako zástupci RTOS byly zvoleny distribuce *Raspbian* OS *Linux*, modifikovaná frameworkem *Xenomai* a distribuce *Raspbian* OS *Linux*, modifikovaná patchem *PREEMPT RT*. Zástupci OS pak tvoří *Raspbian*, *Pidora* a *Arch Linux ARM*, jakožto distribuce operačního systému *Linux*.

Co se týká realizace testovacích scénářů po praktické stránce, tak veškeré testy byly prováděny na „čerstvě“ naimplementovaných operačních systémech (případně se provedla instalace základních balíčků pro správnou funkci systému a testovacích nástrojů). Aby se testovací scénáře mohly alespoň z části vykonávat automaticky, byl pro tyto účely vytvořen BASH skript Stress_generator (zdrojový kód je k dispozici v příloze C).

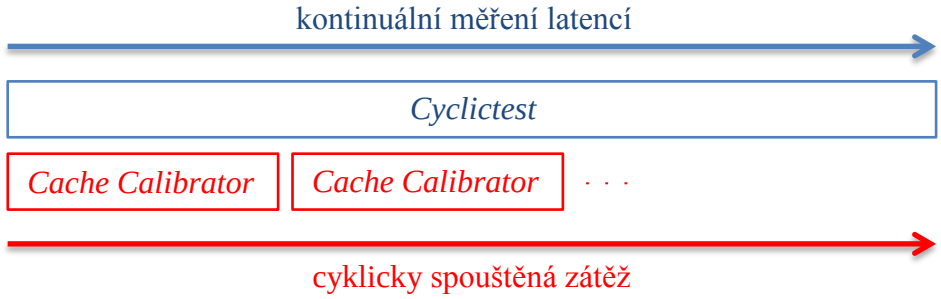
Skript realizuje zátěžovou část testování. V první variantě tedy cyklicky spouští nástroj *Hackbench*, dokud neuplyne časový úsek 15 minut. Ve variantě druhé pro změnu spouští nástroj *Cache Calibrator* a to opět cyklicky a po stejně dlouhý časový úsek. Po celou dobu běhu skriptu Stress_generator je paralelně spuštěno kontinuální měření latencí, což obstarává nástroj *Cyclictest* (spouštěcí parametry nástroje jsou uvedeny viz níže). Tato situace je názorně ilustrována na obr. č. 2.6.1 a 2.6.2.

```
cyclictest -p 99 -t1 -n
```

Pro minimalizaci anomálií při měření bylo stanoveno pět opakování pro každý scénář.



Obr. č. 2.6.1 – Schéma první varianty testovacího scénáře



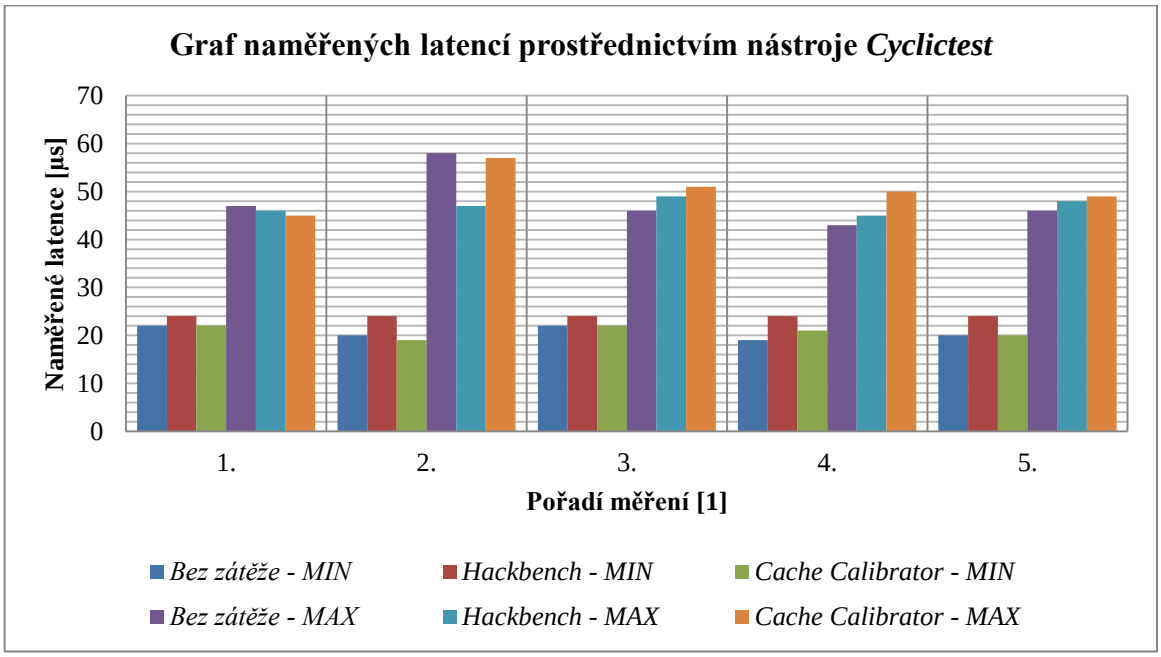
Obr. č. 2.6.2 – Schéma druhé varianty testovacího scénáře

3.3.1 *Raspbian* - modifikace prostřednictvím frameworku *Xenomai*

Výsledné hodnoty z měření latencí nástrojem *Cyclictest*, v prostředí distribuce *Raspbian* OS *Linux* modifikované prostřednictvím frameworku *Xenomai*, jsou uvedeny v tabulce č. 3.3.1. Ve formě grafu jsou pak tyto hodnoty k dispozici na obr. č. 3.3.1.

Tab. č. 3.3.1 – Tabulka naměřených latencí (*Raspbian* - modifikace prostřednictvím frameworku *Xenomai*)

Použitá zátěž	Pořadí měření	Minimální latence	Maximální latence
	1.	22 µs	47 µs
	2.	20 µs	**58 µs**
	3.	22 µs	46 µs
	4.	19 µs	43 µs
	5.	20 µs	46 µs
	ø	17 µs	48 µs
Nástroj *Hackbench*	1.	24 µs	46 µs
	2.	24 µs	47 µs
	3.	24 µs	**49 µs**
	4.	24 µs	45 µs
	5.	24 µs	48 µs
	ø	24 µs	47 µs
Nástroj *Cache Calibrator*	1.	22 µs	45 µs
	2.	19 µs	**57 µs**
	3.	22 µs	51 µs
	4.	21 µs	50 µs
	5.	20 µs	49 µs
	ø	21 µs	50 µs



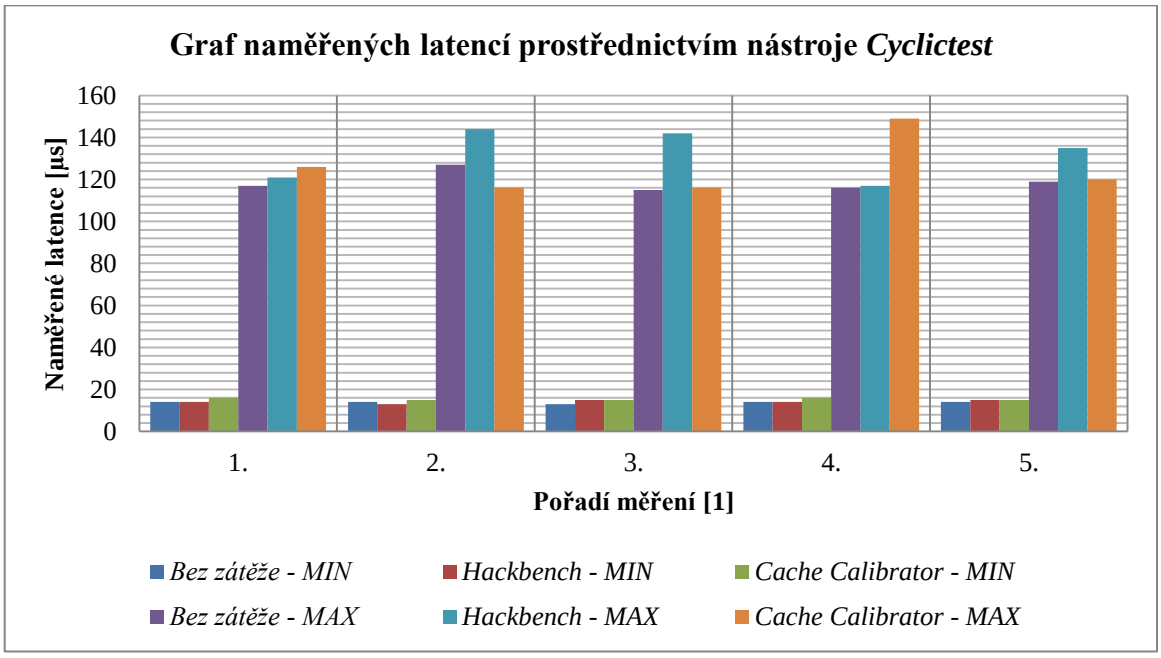
Obr. č. 3.3.1 – Graf naměřených latencí (*Raspbian* - modifikace prostřednictvím frameworku *Xenomai*)

3.3.2 *Raspbian* - modifikace prostřednictvím patche *PREEMPT RT*

Výsledné hodnoty z měření latencí nástrojem *Cyclictest*, v prostředí distribuce *Raspbian* OS *Linux* modifikované prostřednictvím patche *PREEMPT RT*, jsou uvedeny v tabulce č. 3.3.2. Ve formě grafu jsou pak tyto hodnoty k dispozici na obr. č. 3.3.2.

Tab. č. 3.3.2 – Tabulka naměřených latencí (*Raspbian* - modifikace prostřednictvím patche *PREEMPT RT*)

Použitá zátěž	Pořadí měření	Minimální latence	Maximální latence
	1.	14 µs	117 µs
	2.	14 µs	**127 µs**
	3.	13 µs	115 µs
	4.	14 µs	116 µs
	5.	14 µs	119 µs
	ø	14 µs	119 µs
Nástroj *Hackbench*	1.	14 µs	121 µs
	2.	13 µs	**144 µs**
	3.	15 µs	142 µs
	4.	14 µs	117 µs
	5.	15 µs	135 µs
	ø	14 µs	132 µs
Nástroj *Cache Calibrator*	1.	16 µs	126 µs
	2.	15 µs	116 µs
	3.	15 µs	116 µs
	4.	16 µs	**149 µs**
	5.	15 µs	120 µs
	ø	15 µs	125 µs



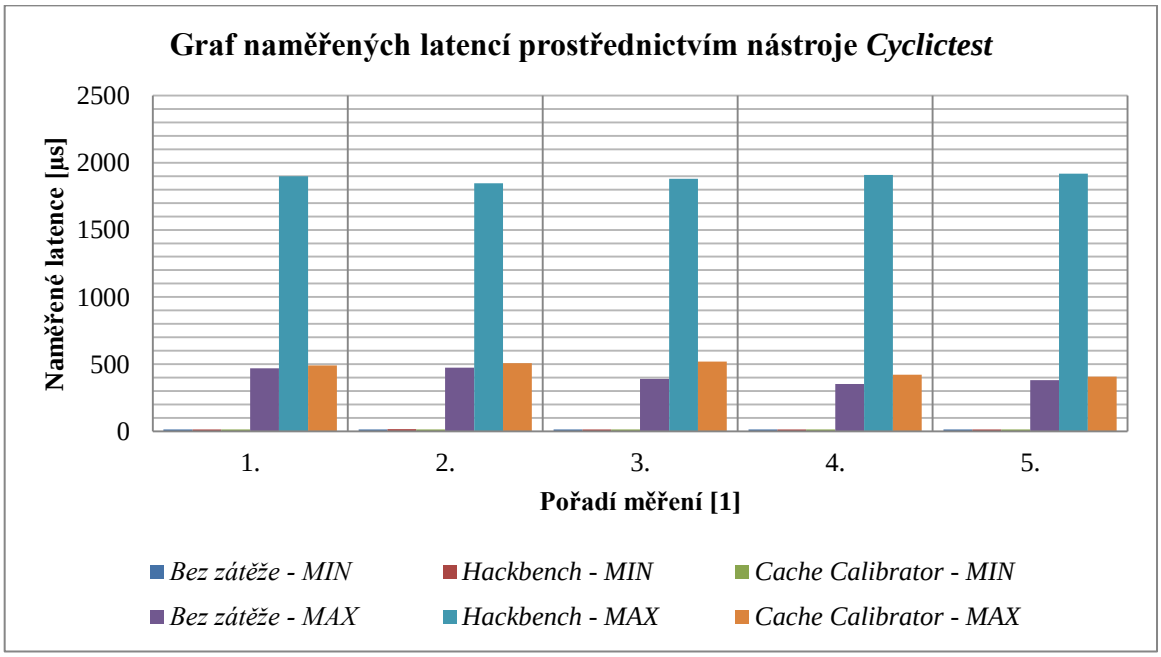
Obr. č. 3.3.2 – Graf naměřených latencí (*Raspbian* - modifikace prostřednictvím patche *PREEMPT RT*)

3.3.3 *Raspbian*

Výsledné hodnoty z měření latencí nástrojem *Cyclictest*, v prostředí distribuce *Raspbian* OS *Linux*, jsou uvedeny v tabulce č. 3.3.3. Ve formě grafu jsou pak tyto hodnoty k dispozici na obr. č. 3.3.3.

Tab. č. 3.3.3 – Tabulka naměřených latencí (*Raspbian*)

Použitá zátěž	Pořadí měření	Minimální latence	Maximální latence
	1.	15 µs	470 µs
	2.	15 µs	**474 µs**
	3.	16 µs	392 µs
	4.	16 µs	353 µs
	5.	16 µs	382 µs
	ø	16 µs	414 µs
Nástroj *Hackbench*	1.	16 µs	1898 µs
	2.	17 µs	1846 µs
	3.	16 µs	1880 µs
	4.	16 µs	1909 µs
	5.	16 µs	**1917 µs**
	ø	16 µs	1890 µs
Nástroj *Cache Calibrator*	1.	16 µs	492 µs
	2.	16 µs	507 µs
	3.	16 µs	**520 µs**
	4.	16 µs	422 µs
	5.	16 µs	407 µs
	ø	16 µs	470 µs



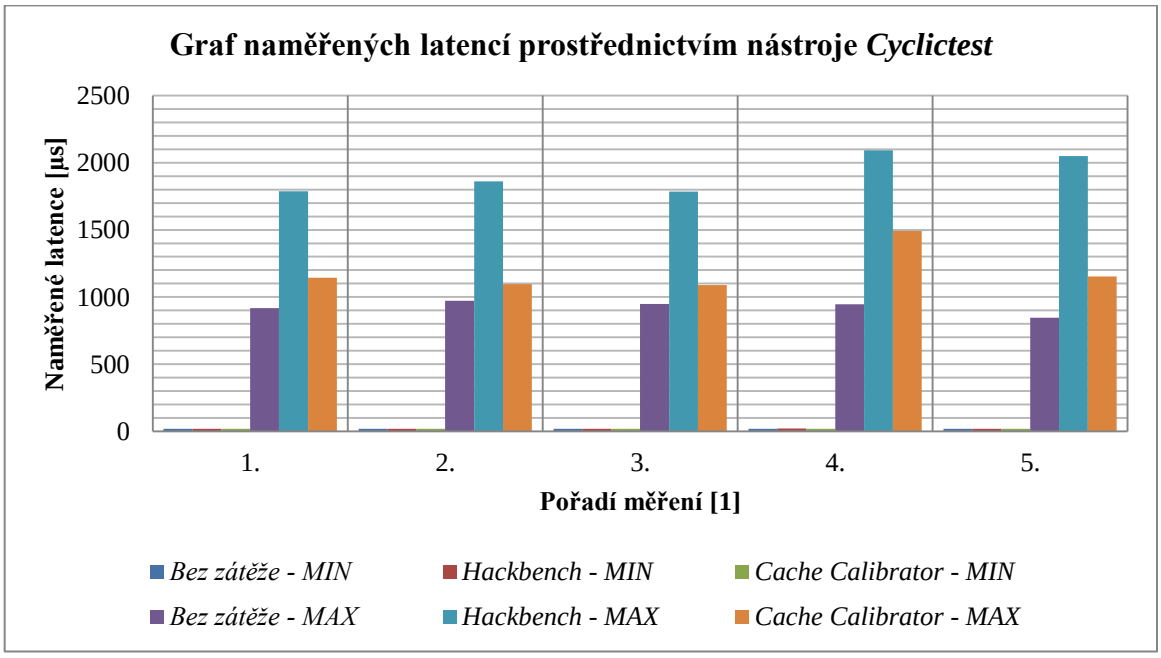
Obr. č. 3.3.3 – Graf naměřených latencí (*Raspbian*)

3.3.4 *Pidora*

Výsledné hodnoty z měření latencí nástrojem *Cyclictest*, v prostředí distribuce *Pidora* OS *Linux*, jsou uvedeny v tabulce č. 3.3.4. Ve formě grafu jsou pak tyto hodnoty k dispozici na obr. č. 3.3.4.

Tab. č. 3.3.4 – Tabulka naměřených latencí (*Pidora*)

Použitá zátěž	Pořadí měření	Minimální latence	Maximální latence
	1.	19 µs	917 µs
	2.	19 µs	**972 µs**
	3.	19 µs	949 µs
	4.	19 µs	946 µs
	5.	19 µs	846 µs
	ø	19 µs	926 µs
Nástroj *Hackbench*	1.	20 µs	1786 µs
	2.	20 µs	1862 µs
	3.	19 µs	1785 µs
	4.	21 µs	**2091 µs**
	5.	19 µs	2048 µs
	ø	20 µs	1914 µs
Nástroj *Cache Calibrator*	1.	19 µs	1145 µs
	2.	19 µs	1097 µs
	3.	20 µs	1090 µs
	4.	19 µs	**1491 µs**
	5.	19 µs	1153 µs
	ø	19 µs	1195 µs



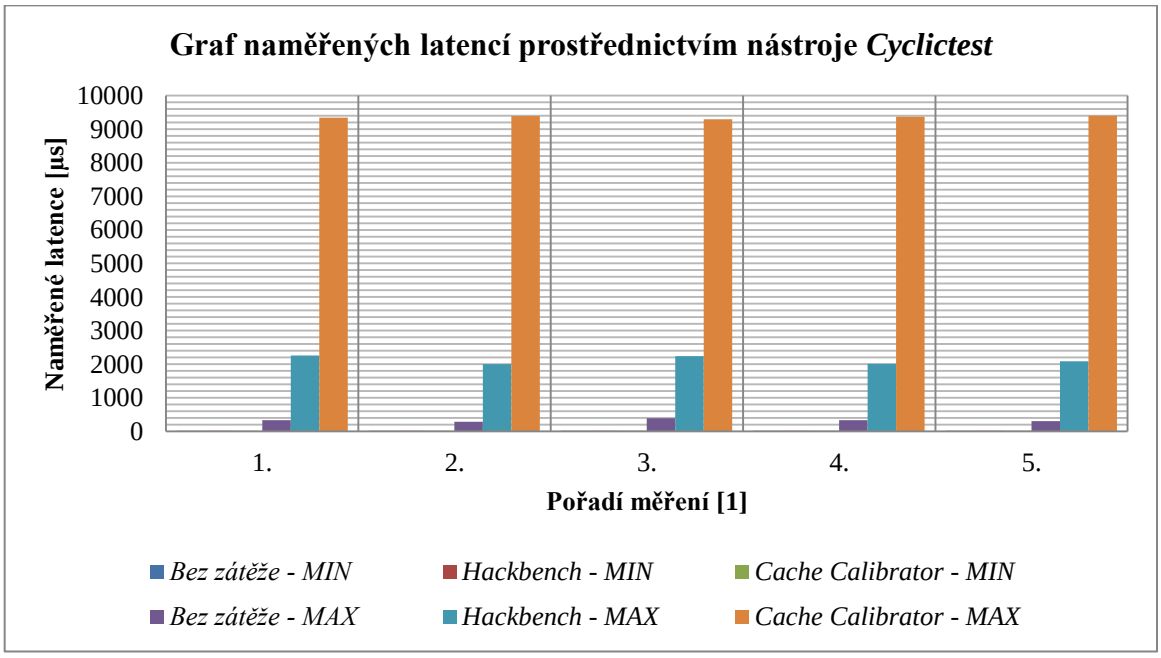
Obr. č. 3.3.4 – Graf naměřených latencí (*Pidora*)

3.3.5 *Arch Linux ARM*

Výsledné hodnoty z měření latencí nástrojem *Cyclictest*, v prostředí distribuce *Arch Linux ARM* OS *Linux*, jsou uvedeny v tabulce č. 3.3.5. Ve formě grafu jsou pak tyto hodnoty k dispozici na obr. č. 3.3.5.

Tab. č. 3.3.5 – Tabulka naměřených latencí (*Arch Linux ARM*)

Použitá zátěž	Pořadí měření	Minimální latence	Maximální latence
	1.	19 µs	340 µs
	2.	19 µs	290 µs
	3.	19 µs	**393 µs**
	4.	19 µs	338 µs
	5.	19 µs	310 µs
	ø	19 µs	334 µs
Nástroj *Hackbench*	1.	19 µs	**2261 µs**
	2.	19 µs	2001 µs
	3.	20 µs	2238 µs
	4.	20 µs	2010 µs
	5.	19 µs	2093 µs
	ø	19 µs	2121 µs
Nástroj *Cache Calibrator*	1.	19 µs	9336 µs
	2.	19 µs	9384 µs
	3.	18 µs	9289 µs
	4.	18 µs	9366 µs
	5.	19 µs	**9398 µs**
	ø	19 µs	9355 µs



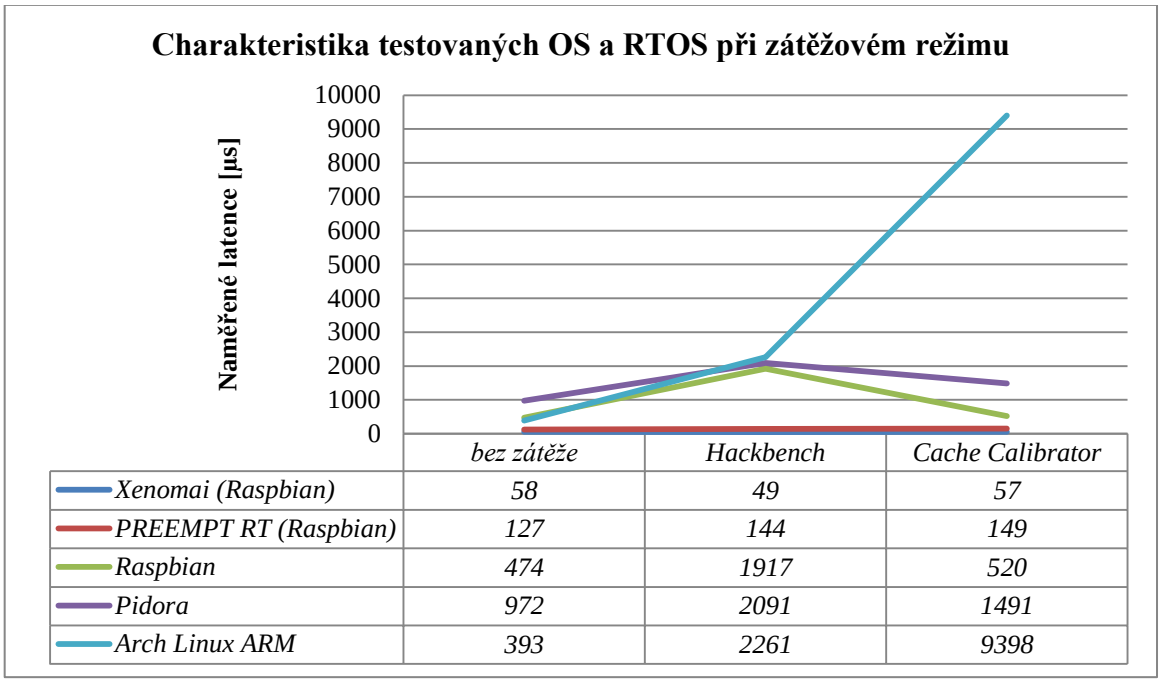
Obr. č. 3.3.5 – Graf naměřených latencí (*Arch Linux ARM*)

4 Dosažené výsledky

Tato kapitola je členěna do dvou podkapitol, ve kterých je na základě provedených testů popsána celková analýza získaných výsledků a následně je i celý postup testování kriticky zhodnocen. Kritické zhodnocení se vesměs zaobírá porovnáním ideálního stavu s reálným.

4.1 Celková analýza

V rámci praktické realizace byly komparativně testovány tři standardní OS a dva transformované RTOS. U každého tohoto OS či RTOS byly naměřeny různé latence. Nejdůležitější jsou pak latence maximální, které se mohou dost razantně odchýlit od latencí průměrných. Právě na tyto odchylky, resp. anomálie, je třeba se soustředit, protože u RTOS je tento jev nežádoucí. Na obr. č. 4.1.1. je uveden graf, který zobrazuje maximální latence testovaných OS a RTOS v klidovém režimu a v režimu radikální zátěže. Nutno ještě podotknout, že se jedná o nejvyšší hodnotu maximální latence ze všech pěti uskutečněných měření, v grafu jsou tedy zaneseny vždy nejhorší možné případy.



	bez zátěže	*Hackbench*	*Cache Calibrator*
Xenomai (Raspbian)	*58*	*49*	*57*
PREEMPT RT (Raspbian)	*127*	*144*	*149*
Raspbian	*474*	*1917*	*520*
Pidora	*972*	*2091*	*1491*
Arch Linux ARM	*393*	*2261*	*9398*

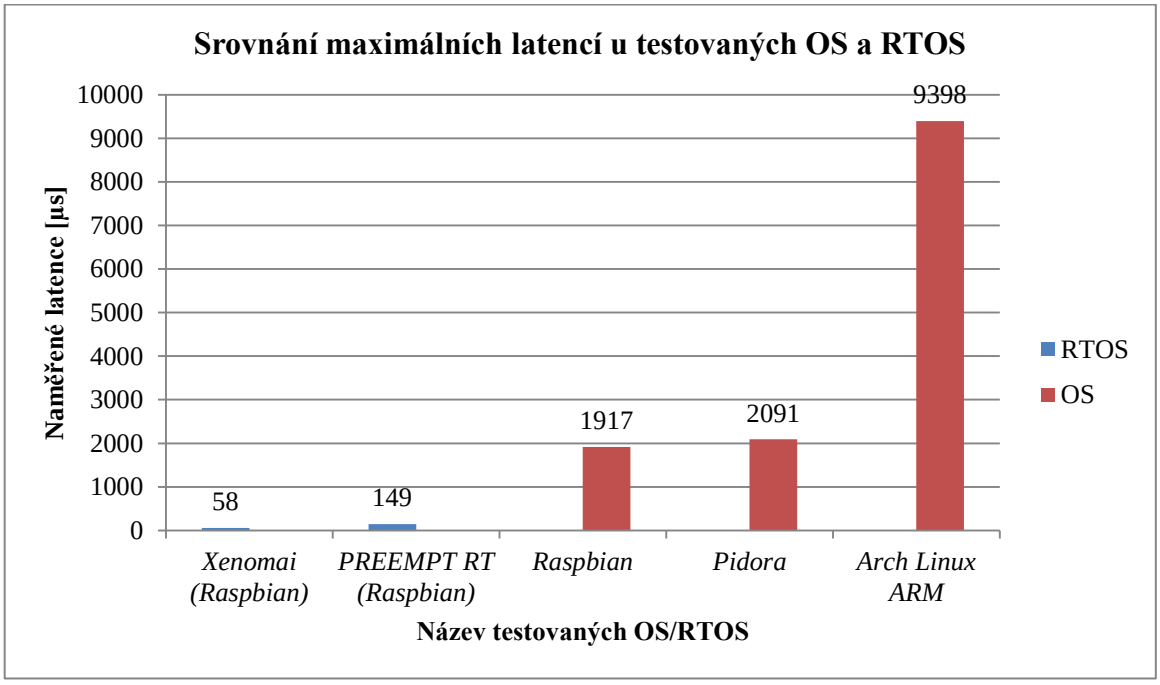
Obr. č. 4.1.1 – Charakteristika testovaných OS a RTOS při zátěžovém režimu

Z grafu lze vidět, že při simulaci zátěže nástrojem *Hackbench* došlo u všech standardních OS k výraznému navýšení maximální latence, jejíž hodnota se pohybuje kolem 2 milisekund. Zajímavé jsou však výsledky při použití nástroje *Cache Calibrator*. U distribucí *Raspbian* a *Pidora* došlo oproti nástroji *Hackbench* k poklesu maximálních

latencí, avšak u distribuce *Arch Linux ARM* došlo k enormnímu navýšení maximální latence až na 9 milisekund (což je i nejvyšší naměřená hodnota při celém procesu testování). V případě transformovaných RTOS, tedy distribuce *Raspbian* modifikované frameworkem *Xenomai* či patchem *PREEMPT RT*, byla však situace zcela odlišná. U modifikace patchem *PREEMPT RT* došlo při simulaci zátěže nástrojem *Cache Calibrator* k mírnému vzestupu maximální latence na hodnotu 149 mikrosekund (při použití nástroje *Hackbench* byla naměřena hodnota maximální latence 144 mikrosekund). Nejlépe si pak vedla modifikace frameworkem *Xenomai*, kde byla maximální latence při zatížení dokonce nižší jak při klidovém režimu (nejvyšší naměřená hodnota z maximálních latencí činila 58 mikrosekund). Režim radikální zátěže v této oblasti tedy vůbec neovlivnil chování systému.

Pokud by se výše uvedené shrnulo, tak lze konstatovat, že transformované RTOS mají i při radikální zátěži konstantní průběh. Tedy generovaná zátěž nemá vliv na odezvu systému. Jako optimální se v tomto ohledu osvědčila distribuce *Raspbian* modifikovaná frameworkem *Xenomai*, kde použitá zátěž na odezvu systému neměla absolutně žádný vliv. U distribuce *Raspbian*, modifikované patchem *PREEMPT RT*, došlo při zátěži k mírnému chvění, které způsobilo navýšení latence o 22 mikrosekund oproti klidovému stavu (vůči testovaným standardním OS lze však toto navýšení zanedbat a považovat chování celého RTOS i při zátěži za konstantní).

Pro zajímavost je níže uveden obr. č. 4.1.2, na němž je ve formě sloupcového grafu vyobrazené srovnání maximálních latencí u testovaných OS a transformovaných RTOS. Jedná se o zjednodušenou informaci o tom, jaké nejvyšší latence lze u daných OS či RTOS očekávat. U každého OS či RTOS je uvedena nejvyšší možná latence, která byla při testování naměřena (tzn. při realizaci všech částí testovacích scénářů).



Obr. č. 4.1.2 – Srovnání maximálních latencí u testovaných OS a RTOS

Nejnižší latence lze, dle předpokladu, očekávat u transformovaných RTOS, tedy u distribuce *Raspbian* modifikované frameworkem *Xenomai* či patchem *PREEMPT RT*. Nejvyšší latence pak samozřejmě byly naměřeny u standardních OS, u nichž si nejhůře vedla distribuce *Arch Linux ARM*.

4.2 Kritické zhodnocení

Praktická komparace vybraných OS a RTOS byla realizována na základě definovaných testovacích scénářů. Tyto scénáře však byly přizpůsobeny možnostem reálné aplikace. Byly tedy dodrženy mezní kritéria, avšak nejedná se o zcela ideální stav. Konkrétně je toto upozornění soustředěno na dobu vykonávání jednotlivých testovacích scénářů. RTOS musí poskytovat jasné záruky. Příkladem budiž situace, kdy je garantováno, že se u daného RTOS nebude vyskytovat vyšší latence jak 100 mikrosekund (tedy, že při vygenerování přerušení se o tomto přerušení dozví obslužná user-space aplikace nejpozději do 100 mikrosekund). Uvažujme, že tento RTOS bude prostřednictvím simulovaného zatížení testován celý rok a po celou tuto dobu budou i monitorovány jeho maximální latence. Pokud by se během této doby vyskytla byť jen jediná anomálie, mohlo by to přinést fatální následky (např. jednorázové vychýlení maximální latence na hodnotu 150 mikrosekund). Čím delší je tedy doba testování, tím lze očekávat relevantnější výsledky. V závislosti na aplikační oblasti daného RTOS je tudíž třeba zvolit dostačující zátěž a dobu testování, neboť to jsou hlavní dva faktory, ovlivňující realtimové charakteristiky.

V případě praktické realizace byly použity nástroje *Hackbench* a *Cache Calibrator*, jakožto simulovaná zátěž. Přičemž nástroj *Hackbench* je doporučovanou zátěží k otestování aplikovaného frameworku *Xenomai* a nástroj *Cache Calibrator* je pro změnu doporučovanou zátěží k otestování aplikovaného patche *PREEMPT RT*. Doporučovanou zátěží jsou tyto nástroje proto, že byly na základě testů provedených širokou veřejností označeny, jakožto nejhorší možný případ pro odezvu systému. Pro verifikaci tohoto tvrzení byly v rámci praktické realizace odzkoušeny i jiné simulace zátěže, ale žádná nedokázala systém zatížit „tak silně“ (jednalo se např. o řadící algoritmy s různou časovou složitostí). Nástroje *Hackbench* a *Cache Calibrator* lze tedy považovat pro tuto oblast za optimální.

Doba testování byla stanovena opět na základě studia obdobných testovacích scénářů provedených širokou veřejností. Minimální doba, potřebná k tomu, aby se vlastnosti RTOS plně projevily, je 15 minut. Tento časový úsek byl tedy definován jako optimální. V ideálním případě by však bylo třeba každou část testovacího scénáře vykonávat mnohem delší dobu. Například *OSADL*[15] provozuje QA serverovou farmu, na níž se testují realtimové vlastnosti patche *PREEMPT RT*. V rámci existujícího, předem definovaného testovacího scénáře, tak byly kontinuálně po dobu půl roku měřeny latence systému s aplikovaným patchem *PREEMPT RT*. Je zřejmé, že vzhledem k této době se jistě podařilo získat relevantnější výsledky.

[15] *OSADL je sdružení, jehož primárním cílem je podporovat a koordinovat vývoj open-source softwaru určeného pro aplikaci v různých oblastech průmyslové automatizace.*

5 Závěr

Na základě provedených teoretických studií byla realizována praktická komparace OS a RTOS, která se řídila sestavenými testovacími scénáři. Výsledky získané komparací, tedy naměřené latence OS a RTOS, byly následně analyzovány. Bylo zjištěno, že nejlepší odezvu má z testovaných vzorků distribuce *Raspbian* OS *Linux* modifikovaná frameworkem *Xenomai*. Odezva systému byla v tomto případě i při silné zátěži konstantní. V těsném závěsu je pak distribuce *Raspbian* modifikovaná patchem *PREEMPT RT*. Ta má odezvu o něco vyšší a lze pozorovat i mírné kolísání odezvy při silné zátěži. U distribucí *Raspbian, Pidora* a *Arch Linux ARM* jsou pak znatelné vysoké odezvy jak v klidovém režimu, tak i při zátěži. V režimu zátěže se však jedná o silné vychýlení, zátěž má na chování těchto distribucí přímý vliv. Nejhorší reakci na zatížení má pak distribuce *Arch Linux ARM*, kde je odezva enormně vysoká.

Pokud by se provedlo porovnání původních představ s dosaženými výsledky, lze tvrdit, že výsledky jsou očekávané. Chování RTOS, jenž představují distribuce *Raspbian* modifikovaná frameworkem *Xenomai* a distribuce *Raspbian* modifikovaná patchem *PREEMPT RT,* odpovídalo obecným zákonitostem determinismu. Oba tyto testované RTOS mají omezené odezvy resp. latence, což je důležitější faktor než její konkrétní hodnota. Je podstatné uvědomit si, že úkolem RTOS není být co „nejrychlejší“, ale být tak „rychlý“, jak je pro danou aplikační oblast třeba. Naopak z chování standardních OS, tedy distribucí *Raspbian, Pidora* a *Arch Linux ARM* je zřejmé, že těmto zákonitostem neodpovídají.

Rozšíření diplomové práce by mohlo spočívat ve vytvoření BASH skriptů, které by umožnily automatické zpracování a byly by tak alternativou k manuálním modifikacím OS pro profit realtimových vlastností. Dále by bylo zajímavé do implementační části zahrnout i další operační systémy, jako distribuce *Gentoo* OS *Linux* či emulátor *RetroPie*.

Cíl diplomové práce, tedy vyhotovení komparativní studie multitaskingových OS a RTOS v embedded systému na základě praktické verifikace jejich funkcionality, byl splněn. Verifikací realtimových vlastností naimplementovaných OS a RTOS bylo dokázáno, že mikropočítač *Raspberry Pi* lze použít nejen jako embedded hardware alternativní k PC či HTPC jednotkám, ale i pro aplikaci v nejrůznějších oblastech průmyslové automatizace.

Přehled zkratek

3D	*Three-Dimensional Space*
ABI	*Application Binary Interface*
ADEOS	*Adaptive Domain Environment for Operating Systems*
AMBA	*Advanced Microcontroller Bus Architecture*
API	*Application Programming Interface*
AXI	*Advanced eXtensible Interface*
BASH	*Bourne Again Shell*
CISC	*Complex Instruction Set Computer*
CLI	*Command Line Interface*
CNC	*Computer Numerical Control*
CPU	*Central Processing Unit*
CSS	*Cascading Style Sheets*
DMA	*Direct Memory Access*
DMESG	*Display Message or Driver Message*
DSI	*Display Serial Interface*
FPA	*Floating Point Accelerator*
GNU	*GNU's Not Unix!*
GPU	*Graphics Processing Unit*
GPIO	*General-Purpose Input/Output*
GUI	*Graphical User Interface*
HAL	*Hardware Abstraction Layer*
HDMI	*High-Definition Multimedia Interface*
HTML	*HyperText Markup Language*
HTPC	*Home Theater Personal Computer*
HTTP	*HyperText Transfer Protocol*
I^2C	*Inter-Integrated Circuit*
IRQ	*Interrupt ReQuest*
L2	*Level 2 (cache)*
LCD	*Liquid-Crystal Display*
LXDE	*Lightweight X11 Desktop Environment*

MMC	*Multi-Media Card*
MPEG	*Moving Picture Experts Group*
NTSC	*National Television System(s) Committee*
OC	*OverClock*
OpenELEC	*Open Embedded Linux Entertainment Center*
OpenGL	*Open Graphics Library*
OS	*Operating System*
OSADL	*Open Source Automation Development Lab*
PAL	*Phase Alternating Line*
POSIX	*Portable Operating System Interface*
pSOS	*Portable Software On Silicon*
RAM	*Random-Access Memory*
RISC	*Reduced Instruction Set Computing*
RTDM	*Real-Time Driver Model*
RTOS	*Real-Time Operating System*
SAL	*System Abstraction Layer*
SCI	*SysCall Interface*
SDIO	*Secure Digital Input Output*
SDHC	*Secure Digital High Capacity*
SDHCI	*Secure Digital Host Controller Interface*
SDRAM	*Synchronous Dynamic Random Access Memory*
SIMD	*Single Instruction Multiple Data*
SMI	*System Management Interrupt*
SoC	*System on a Chip*
SPI	*Serial Peripheral Interface*
TLB	*Translation Lookaside Buffer*
UART	*Universal Asynchronous Receiver/Transmitter*
URL	*Uniform Resource Locator*
USB	*Universal Serial Bus*
VFP	*Vector Floating Point*
VRTX	*Versatile Real-Time Executive*
XFCE	*XForms Common Environment*

Literatura

[1] JELÍNEK, Lukáš. *Jádro systému Linux: kompletní průvodce programátora*. Vyd. 1. Brno: Computer Press, 2008, 686 s. ISBN 978-80-251-2084-2.

[2] DIXIT, J.B. *Computer Fundamentals and Programming in C*. 1st ed. New Delhi: Laxmi Publications, 2006. ISBN 81-700-8882-8.

[3] ČADA, Ondřej. *Operační systémy*. Praha, ČR: Grada, a. s., 1994. ISBN 80-85623-44-7.

[4] DUDÁČEK, Karel. *Operační systémy reálného času* [online]. Plzeň, ČR, 2010 [cit. 2014-04-07]. Dostupné z: https://courseware.zcu.cz/wps/PA_Courseware/DownloadDokumentu?id=58491. Přednášky z předmětu Navrhování mikropočítačových systémů. ZČU, FAV.

[5] Raspbian Documentation. *Raspbian* [online]. 2013 [cit. 2014-04-08]. Dostupné z: http://www.raspbian.org/RaspbianDocumentation

[6] Pidora FAQ. *Open Source@Seneca* [online]. 2013 [cit. 2014-04-09]. Dostupné z: http://zenit.senecac.on.ca/wiki/index.php/Pidora_FAQ

[7] Arch Linux ARM. *Arch Linux ARM* [online]. © 2009-2014 [cit. 2014-04-09]. Dostupné z: http://archlinuxarm.org/

[8] RISC OS Pi. *RISC OS Open* [online]. © RISC OS Open Limited 2011 [cit. 2014-04-09]. Dostupné z: https://www.riscosopen.org/content/sales/risc-os-pi

[9] Raspberry Pi podporuje minimalistický operační systém RISC OS. *Živě.cz* [online]. 2012 [cit. 2014-04-10]. Dostupné z: http://www.zive.cz/bleskovky/raspberry-pi-podporuje-minimalisticky-operacni-system-risc-os/sc-4-a-166394/default.aspx

[10] Firefox OS. *Mozilla Developer Network* [online]. © 2005-2014 [cit. 2014-04-10]. Dostupné z: https://developer.mozilla.org/en-US/Firefox_OS

[11] Firefox OS for Raspberry Pi. *Philipp's Weblog* [online]. 2013 [cit. 2014-04-10]. Dostupné z: https://www.philipp-wagner.com/ffos-for-rpi/manual/index.html

[12] Co je CyanogenMod?. *CyanogenMod.cz* [online]. © 2012 [cit. 2014-04-11]. Dostupné z: http://cyanogenmod.cz/cyanogenmod/

[13] Světem OS skrz na skrz: Plan 9. *Root.cz - informace nejen ze světa Linuxu* [online]. 2001 [cit. 2014-04-11]. Dostupné z: http://www.root.cz/clanky/plan-9/

[14] Other hardware. *Plan 9 from Bell Labs* [online]. 2013 [cit. 2014-04-19]. Dostupné z: http://www.plan9.bell-labs.com/wiki/plan9/Other_hardware/

[15] About Raspbmc. *Raspbmc* [online]. 2012 [cit. 2014-04-19]. Dostupné z: http://www.raspbmc.com/about/

[16] XBMC: Build Your Own HTPC. *CyberNet News* [online]. 2012 [cit. 2014-04-19]. Dostupné z: http://cybernetnews.com/build-xbmc-htpc/

[17] What is OpenELEC?. *OpenELEC Mediacenter - Home* [online]. © 2009-2013 [cit. 2014-04-19]. Dostupné z: http://openelec.tv/component/content/article?id=1

[18] Xenomai: Real-Time Framework for Linux. *Xenomai* [online]. 2006 [cit. 2014-03-17]. Dostupné z: http://www.xenomai.org/

[19] PREEMPT RT patch - Frequently Asked Questions. *Real-Time Linux Wiki* [online]. 2012 [cit. 2014-04-30]. Dostupné z: https://rt.wiki.kernel.org/index.php/Frequently_Asked_Questions

[20] Cyclictest - High resolution test program. *Ubuntu Manpage* [online]. 2007 [cit. 2014-04-20]. Dostupné z: http://manpages.ubuntu.com/manpages/oneiric/man8/cyclictest.8.html

[21] Hackbench - scheduler benchmark/stress test. *Ubuntu Manpage* [online]. 2010 [cit. 2014-04-20]. Dostupné z: http://manpages.ubuntu.com/manpages/precise/man8/hackbench.8.html

[22] Jaderné noviny – 30. 6. 2010. *AbcLinuxu.cz - Linux na stříbrném podnose* [online]. 2010 [cit. 2014-04-20]. Dostupné z: http://www.abclinuxu.cz/clanky/jaderne-noviny-30.-6.-2010

[23] The Calibrator (v0.9e), a Cache-Memory and TLB Calibration Tool. *CWI Amsterdam* [online]. 2004 [cit. 2014-04-20]. Dostupné z: http://homepages.cwi.nl/~manegold/Calibrator/

[24] Technické parametry. *Raspberry Pi: miniaturní levný počítač* [online]. 2013 [cit. 2014-03-17]. Dostupné z: http://raspberrypi.cz/ops4/

[25] TICHÝ, Miroslav. *Představení a vývoj architektury ARM* [online]. Ostrava, ČR, 2009 [cit. 2014-03-18]. Dostupné z: http://wh.cs.vsb.cz/mil051/images/4/43/PAP_Architektura_procesoru_ARM_prezentace_%28Miroslav_Tich%C3%BD%29.pdf. Prezentace. VŠB-TUO.

[26] ARM1176 Processor. *ARM The Architecture for the Digital World* [online]. © ARM Ltd. Copyright 2014 [cit. 2014-03-18]. Dostupné z: http://www.arm.com/products/processors/classic/arm11/arm1176.php

[27] Machinoid Hard Real-Time Distribution Optimized for Machines. *Machinoid* [online]. 2013 [cit. 2014-03-17]. Dostupné z: www.machinoid.com

Seznam příloh

A. Verifikace frameworku *Xenomai* prostřednictvím skriptu XENO-TEST
B. Zdrojový kód pro verifikaci patche *PREEMPT RT*
C. Zdrojový kód BASH skriptu Stress_generator pro generování zátěže
D. Adresářová struktura přiloženého DVD

Příloha A

Činnost skriptu XENO-TEST při verifikaci frameworku *Xenomai*

Spouštěcí příkaz:

```
/usr/xenomai/bin/./xeno-test -l "dohell 900" -p 100 -g histo
```

Průběh testování:

```
Started child 26084: /bin/bash /usr/xenomai/bin/xeno-test-run-wrapper
/usr/xenomai/bin/./xeno-test -p 100 -g histo
++ echo 0
++ /usr/xenomai/bin/arith
mul: 0x79364d93, shft: 26
integ: 30, frac: 0x4d9364d9364d9364

signed positive operation: 0x03ffffffffffffff * 1000000000 / 33000000
inline calibration: 0x0000000000000000: 750.000 ns, rejected 9996/10000
inlined llimd: 0x79364d9364d9362f: 1326.200 ns, rejected 40/10000
inlined llmulshft: 0x79364d92ffffffe1: -250.000 ns, rejected 9998/10000
inlined nodiv_llimd: 0x79364d9364d9362f: -250.000 ns, rejected 9998/10000
out of line calibration: 0x0000000000000000: 500.000 ns, rejected 9998/10000
out of line llimd: 0x79364d9364d9362f: 1605.000 ns, rejected 40/10000
out of line llmulshft: 0x79364d92ffffffe1: 166.600 ns, rejected 9997/10000
out of line nodiv_llimd: 0x79364d9364d9362f: 166.600 ns, rejected 9997/10000

signed negative operation: 0xfc00000000000001 * 1000000000 / 33000000
inline calibration: 0x0000000000000000: 500.000 ns, rejected 9998/10000
inlined llimd: 0x86c9b26c9b26c9d1: 1612.400 ns, rejected 41/10000
inlined llmulshft: 0x86c9b26d0000001e: 166.600 ns, rejected 9997/10000
inlined nodiv_llimd: 0x86c9b26c9b26c9d1: 0.000 ns, rejected 9998/10000
out of line calibration: 0x0000000000000000: 500.000 ns, rejected 9998/10000
out of line llimd: 0x86c9b26c9b26c9d1: 1619.200 ns, rejected 42/10000
out of line llmulshft: 0x86c9b26d0000001e: 0.000 ns, rejected 9998/10000
out of line nodiv_llimd: 0x86c9b26c9b26c9d1: 0.000 ns, rejected 9998/10000

unsigned operation: 0x03ffffffffffffff * 1000000000 / 33000000
inline calibration: 0x0000000000000000: 0.000 ns, rejected 9999/10000
inlined nodiv_ullimd: 0x79364d9364d9362f: 500.000 ns, rejected 9998/10000
out of line calibration: 0x0000000000000000: 500.000 ns, rejected 9998/10000
out of line nodiv_ullimd: 0x79364d9364d9362f: 0.000 ns, rejected 9998/10000
++ /usr/xenomai/bin/clocktest -C 42 -T 30
== Tested clock: 42 (CLOCK_HOST_REALTIME)
CPU     ToD offset [us] ToD drift [us/s]     warps max delta [us]
--- -------------------- ---------------- ---------- --------------
  0               2.0          0.000         0          0.0
++ /usr/xenomai/bin/switchtest -T 30
== Testing FPU check routines...
d0: 1 != 2
d1: 1 != 2
d2: 1 != 2
```

```
d3: 1 != 2
d4: 1 != 2
d5: 1 != 2
d6: 1 != 2
d7: 1 != 2
d8: 1 != 2
d9: 1 != 2
d10: 1 != 2
d11: 1 != 2
d12: 1 != 2
d13: 1 != 2
d14: 1 != 2
d15: 1 != 2
== FPU check routines: OK.
== Threads: sleeper_ufps0-0 rtk0-1 rtk0-2 rtk_fp0-3 rtk_fp0-4 rtk_fp_ufpp0-5
rtk_fp_ufpp0-6 rtup0-7 rtup0-8 rtup_ufpp0-9 rtup_ufpp0-10 rtus0-11 rtus0-12
rtus_ufps0-13 rtus_ufps0-14 rtuo0-15 rtuo0-16 rtuo_ufpp0-17 rtuo_ufpp0-18
rtuo_ufps0-19 rtuo_ufps0-20 rtuo_ufpp_ufps0-21 rtuo_ufpp_ufps0-22
RTT|  00:00:01
RTH|---------cpu|ctx switches|-------total
RTD|          0|        14490|       14490
RTD|          0|        14626|       29116
RTD|          0|        14492|       43608
RTD|          0|        14626|       58234
RTD|          0|        14701|       72935
RTD|          0|        14693|       87628
RTD|          0|        14628|      102256
RTD|          0|        14697|      116953
RTD|          0|        14285|      131238
RTD|          0|        14488|      145726
RTD|          0|        14494|      160220
RTD|          0|        14555|      174775
RTD|          0|        14490|      189265
RTD|          0|        14628|      203893
RTD|          0|        14421|      218314
RTD|          0|        14559|      232873
RTD|          0|        14563|      247436
RTD|          0|        14557|      261993
RTD|          0|        14557|      276550
RTD|          0|        14494|      291044
RTD|          0|        14490|      305534
RTT|  00:00:22
RTH|---------cpu|ctx switches|-------total
RTD|          0|        14490|      320024
RTD|          0|        14490|      334514
RTD|          0|        14555|      349069
RTD|          0|        14492|      363561
```

```
RTD|      0|     14628|    378189
RTD|      0|     14354|    392543
RTD|      0|     14559|    407102
RTD|      0|     14557|    421659
RTD|      0|     12905|    434564
++ /usr/xenomai/bin/cond-torture-native
simple_condwait
relative_condwait
absolute_condwait
sig_norestart_condwait
sig_restart_condwait
sig_norestart_condwait_mutex
sig_restart_condwait_mutex
sig_norestart_double
sig_restart_double
cond_destroy_whilewait
Test OK
++ /usr/xenomai/bin/cond-torture-posix
simple_condwait
relative_condwait
absolute_condwait
sig_norestart_condwait
sig_restart_condwait
sig_norestart_condwait_mutex
sig_restart_condwait_mutex
sig_norestart_double
sig_restart_double
cond_destroy_whilewait
Test OK
++ /usr/xenomai/bin/mutex-torture-native
simple_wait
recursive_wait
timed_mutex
mode_switch
pi_wait
lock_stealing
NOTE: lock_stealing mutex_trylock: not supported
deny_stealing
simple_condwait
recursive_condwait
auto_switchback
Test OK
++ /usr/xenomai/bin/mutex-torture-posix
simple_wait
recursive_wait
errorcheck_wait
timed_mutex
```

```
mode_switch
pi_wait
lock_stealing
NOTE: lock_stealing mutex_trylock: not supported
deny_stealing
simple_condwait
recursive_condwait
auto_switchback
Test OK
++ start_load
++ echo start_load
++ check_alive /usr/xenomai/bin/latency -p 100 -g histo
++ echo check_alive /usr/xenomai/bin/latency -p 100 -g histo
++ wait_load
Started child 26151: dohell 900
++ read rc
Started child 26155: /usr/xenomai/bin/latency -p 100 -g histo
== Sampling period: 100 us
== Test mode: periodic user-mode task
== All results in microseconds
warming up...
RTT|  00:00:01  (periodic user-mode task, 100 us period, priority 99)
RTH|----lat min|----lat avg|----lat max|-overrun|---msw|---lat best|--lat worst
RTD|      4.000|    11.000|      25.000|         0|       0|      4.000|     25.000
RTD|      3.000|    11.000|      37.000|         0|       0|      3.000|     37.000
RTD|      4.000|    11.000|      29.000|         0|       0|      3.000|     37.000
RTD|      3.000|    11.000|      24.000|         0|       0|      3.000|     37.000
RTD|      3.000|    11.000|      25.000|         0|       0|      3.000|     37.000
RTD|      4.000|    11.000|      24.000|         0|       0|      3.000|     37.000
RTD|      3.000|    11.000|      24.000|         0|       0|      3.000|     37.000
[…]
RTD|      4.000|    12.000|      25.000|         0|       0|      2.000|     44.000
RTD|      3.000|    12.000|      25.000|         0|       0|      2.000|     44.000
RTD|      3.000|    12.000|      24.000|         0|       0|      2.000|     44.000
RTD|      3.000|    12.000|      29.000|         0|       0|      2.000|     44.000
RTT|  00:15:04  (periodic user-mode task, 100 us period, priority 99)
RTH|----lat min|----lat avg|----lat max|-overrun|---msw|---lat best|--lat worst
RTD|      3.000|    12.000|     25.000|         0|       0|      2.000|     44.000
RTD|      3.000|    12.000|     30.000|         0|       0|      2.000|     44.000
Load script terminated, terminating checked scripts
HSH|--param|--samples-|--average--|---stddev--
HSS|       min|         905|       3.189|     0.522
HSS|       avg|   9051616|      12.256|     3.339
HSS|       max|         905|      27.524|     4.258
-----|-------------|--------------|------------|------------|---------|------------------------
RTS|      2.000|     12.000|    44.000|          0|       0|   00:15:05/00:15:05
pipe_in: /tmp/xeno-test-in-31124
```

Příloha B

Zdrojový kód pro verifikaci patche *PREEMPT RT*

Níže uvedený zdrojový kód v programovacím jazyce C po následném zkompilování slouží pro verifikaci, zda byl na jádro OS *Linux* skutečně aplikován patch *PREEMPT RT* (zdrojový kód byl převzat z dokumentace „*RT PREEMPT HOWTO*“, jakožto součást podkapitoly „*Runtime detection of an RT-PREEMPT Kernel*“).

```
#include <string.h>
#include <stdio.h>
#include <sys/utsname.h>

int main(int argc, char **argv)
{
        struct utsname u;
        int crit1, crit2 = 0;
        FILE *fd;

        uname(&u);
        crit1 = strcasestr (u.version, "PREEMPT RT");

        if ((fd = fopen("/sys/kernel/realtime","r")) != NULL)
        {
                int flag;
                crit2 = ((fscanf(fd, "%d", &flag) == 1) && (flag == 1));
                fclose(fd);
        }
        fprintf(stderr, "this is a %s kernel\n",
      (crit1 && crit2)  ? "PREEMPT RT" : "vanilla");
}
```

Po zkompilování a následném spuštění byl na distribuci *Raspbian*, modifikované patchem *PREEMPT RT* obdržen následující výpis.

```
this is a PREEMPT RT kernel
```

Příloha C

Zdrojový kód BASH skriptu Stress_generator pro generování zátěže

```
#!/bin/bash

stress_time=900
echo "Ma byt provedena kompilace zdrojovych kodu nastroju?"
echo "1) Ano"
echo "2) Ne"
echo -n "Volba: "
read -n 1 compile_number
echo
case $compile_number in
1)      # Kompilace zdrojoveho kodu nastroje Hackbench
        gcc -g -Wall -O2 -o hackbench hackbench.c -lpthread ;
        sleep 5 ;
        if [  !  -f  hackbench ] ; then
                echo "Chyba pri kompilaci zdrojoveho kodu nastroje Hackbench"
        fi ;
        # Kompilace zdrojoveho kodu nastroje Cache Calibrator
        patch -p1 < round_error.patch ;
        gcc calibrator.c -o cache_calibrator -lm ;
        sleep 5 ;
        if [  !  -f  cache_calibrator ] ; then
                echo "Chyba pri kompilaci zdrojoveho kodu nastroje Cache Calibrator"
        fi ;;
2)      echo "Vyberte zadany nastroj pro simulaci zateze"
        echo "1) Hackbench"
        echo "2) Cache Calibrator"
        echo -n "Volba: "
        read -n 1 tool_number
        echo
        case $tool_number in
        1)      echo "Spousteni nastroje Hackbench" ;
                T="$(date +%s)"
                while [ "$(($(date +%s)-T))" -le $stress_time ] ; do
                        ./hackbench --pipe 50 ;
                done ;;
        2)      echo "Spousteni nastroje Cache Calibrator" ;
                T="$(date +%s)"
                while [ "$(($(date +%s)-T))" -le $stress_time ] ; do
                        ./cache_calibrator 700 100M output;
                done ;;
        *)      echo "Neplatny znak - skript ukoncen" ;
                exit 1 ;;
        esac ;;
*)      echo "Neplatny znak - skript ukoncen" ;
        exit 1 ;;
esac
exit
```

Příloha D

Adresářová struktura přiloženého DVD

DVD

- **Operační systémy reálného času**
  - *Raspbian-PREEMPT_RT.img.gz*
  - *Raspbian-Xenomai.img.gz*
- **Software pro klonování**
  - Win32 Disk Imager
    - *Changelog.txt*
    - *GPL-2*
    - *LGPL-2.1*
    - *libgcc_s_dw2-1.dll*
    - *libstdc++-6.dll*
    - *mingwm10.dll*
    - *QtCore4.dll*
    - *QtGui4.dll*
    - *README.txt*
    - *Win32DiskImager.exe*
- **Software pro testování**
  - *calibrator.c*
  - *hackbench.8*
  - *hackbench.c*
  - *round_error.patch*
  - *Stress_generator.sh*
- **Text diplomové práce**
  - *A12N0026P_DP.docx*
  - *A12N0026P_DP.pdf*
- **Zadání diplomové práce**
  - *Zadání DP - strana 1.jpg*
  - *Zadání DP - strana 2.jpg*