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

Bakalářská práce

Benchmarking na platformě Android

Plzeň, 2012 Jan Pavlík

Prohlášení

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

V Plzni dne 10. května 2012

Jan Pavlík

Děkuji vedoucímu práce Ing. Ladislavu Pešičkovi za odborné vedení a cenné rady při vypracování bakalářské práce.

Abstract

This Bachelor thesis focuses on benchmarking of mobile devices with operating system Android. The first part deals with different types of benchmarks and their utilization. The process of benchmarking in information technology is described in this part too. The theoretical part includes an overview of the most frequently used benchmarks of mobile devices.

The main goal of this thesis is to propose a possible way of testing and consequently to create an application which will test a filesystem and computing power of mobile devices. The application will send the acquired results to a server so that they are processed. This will enable the users to compare the performance of their device with others.

Obsah

1 Úvod

Platforma Android je v dnešní době velmi rozšířená a běží na velkém množství rozdílných zařízení, která se liší svým výkonem. Vzniká tak přirozeně potřeba je mezi sebou porovnávat. Vhodným nástrojem pro porovnání výkonu různých zařízení je benchmarking.

Cílem první části práce je poskytnout čtenáři teoretický základ o benchmarkingu v oblasti informačních technologií a zároveň také představit vhodné postupy pro benchmarking souborového systému a výpočetního výkonu mobilních zařízení.

Cílem praktické části práce je navrhnout a vytvořit aplikaci na testování mobilních zařízení s možností centrálního zpracování naměřených výsledků. Vlastní testování se zaměřuje především na souborový systém, neboť vytvořená aplikace má být v budoucnu využita na testování souborového systému KIVFS, který je vyvíjen na katedře informatiky a výpočetní techniky ZČU.

2 Benchmarking

Benchmarking je důležitým pojmem v oblasti počítačových věd. Využívá se například při návrhu, vývoji a realizaci počítačových systémů nebo také při výběru nejlepší možnosti z dostupných alternativ. Hlavním cílem zpravidla bývá dosáhnout maximálního výkonu za minimální cenu. K řešení tohoto typického zadání využíváme právě benchmarkingu.

Benchmarking je proces, při němž se podle zvolených parametrů vyhodnocuje a porovnává výkon různých objektů, které jsou založeny na odlišných programových strukturách či hardwarových architekturách. Na základě výsledků získaných benchmarkingem se může následně jeho vykonavatel rozhodnout, které řešení je pro něj nejvhodnější [Fei11]. Zároveň je také třeba dodat, že není správné spoléhat jen na výsledky benchmarkingu, protože ty v některých případech nemusí korespondovat s výsledky v reálném použití (viz 2.4).

2.1 Benchmark

Termín benchmark je definován jako standard, pomocí kterého se něco měří a vyhodnocuje. Speciálně v oblasti počítačů je to běh programu nebo sady programů za účelem vyhodnocení relativního výkonu měřeného objektu (procesoru, grafické karty, souborového systému). Stejný benchmark je tedy aplikován na různé počítačové systémy, které se mají porovnávat. Aby mělo porovnání smysl, měl by daný benchmark splňovat několik základních požadavků [Wei02].

1. **Přenositelnost (Portability):** Benchmark musí být spustitelný na různých počítačových systémech.
2. **Férovost (Fairness):** Stává se, že některé benchmarky obsahují části, které vyhovují jednomu danému systému. Je logické, že takový systém bude mít lepší výsledky než ostatní. Spravedlivý benchmark se snaží tyto náklonnosti k jednotlivým systémům minimalizovat.
3. **Reprezentativnost (Representativeness):** Benchmark by měl co nejvěrněji simulovat chování nějakého systému v reálném využití. Nemá

význam porovnávat výkon pomocí benchmarku založeného na nepoužívaných programech nebo neobvyklých algoritmech.

4. **Měřitelnost (Measurability):** Je zjevné, že nejlepším měřítkem výkonu je samotná uživatelská aplikace. Upravovat danou aplikaci tak, aby bylo možné testovat výkon na velkém počtu systémů, by bylo příliš nákladné. Právě proto je důležité, aby byl benchmark také dostatečně univerzální a dokázal tak měřit výkon na velkém množství různých systémů.

5. **Jednoduchost interpretace (Easy to explain):** Uživatel by měl rozumět výsledkům benchmarku a jejich významu, aby byl schopen efektivně a správně hodnotit výkon měřených systémů.

2.1.1 Definice benchmarků

Definice benchmarků dělíme do třech hlavních tříd. Na úvod je třeba poznamenat, že benchmark není definován pouze zdrojovými či binárními kódy, ale zároveň také vstupními daty a dokumentem, který popisuje pravidla pro provádění benchmarku a požadavky na měřící prostředí daného systému [Wei02].

Nejrozšířenější formou benchmarků jsou benchmarky definované **zdrojovým kódem**. Pro spuštění těchto benchmarků je zapotřebí překladače pro cílový systém. Patří sem například SPEC[1], EEMBC[2] benchmarky.

Další kategorií jsou benchmarky definované přímo **binárními soubory**. Z této definice vyplývá, že jsou zmíněné benchmarky spustitelné pouze na omezeném množství systémů. Benchmarky tohoto typu ale také nacházejí svoje uplatnění, protože existují velmi rozšířené platformy (např. Intel/Windows PC), na nichž je lze ve vysokém počtu provádět.

Třetí způsob definice benchmarku je definice pomocí specifikace. To znamená, že benchmark je definován pouze **specifikačním dokumentem** a tato definice neobsahuje žádný zdrojový nebo binární kód. Jedním z nejznámějších vydavatelů těchto standardů je nezisková firma TPC[3].

[1]The Standard Performance Evaluation Corporation
[2]The Embedded Microprocessor Benchmark Consortium
[3]Transaction Processing Performance Council

2.1.2 Typy benchmarků

Benchmarky můžeme podle jejich způsobu provádění, získávání výsledků a objektů, které jsou měřeny zařadit do dvou kategorií. Jsou to mikrobenchmarky a makrobenchmarky [Sma97][Tra08].

Jak už název napovídá, cílem **mikrobenchmarků** je měřit výkon nějaké menší specifické části daného systému. Informace získané mikrobenchmarkem nám následně mohou v kombinaci s použitím makrobenchmarku pomoci při určování, porozumění a vyhodnocování konkrétních rozdílů mezi systémy. Mikrobenchmarky se také používají jako diagnostický nástroj, protože jsou navrženy přímo na jednotlivé části systému a dokážou je velmi dobře měřit.

Opakem mikrobenchmarků jsou **makrobenchmarky**, jejichž úkolem je vyhodnocení celkového výkonu měřeného systému při různých mírách zátěže. Právě podle typu zátěže můžeme tyto benchmarky ještě dále dělit. Existují makrobenchmarky, které napodobují zátěž v **reálném prostředí** a naopak tzv. **syntetické** (uměle vytvořené) makrobenchmarky. Oba tyto typy benchmarků mají svoje klady a zápory, které spolu úzce souvisejí. Benchmarky založené na aplikacích z reálného světa nám dávají užitečné a relevantní výsledky, ale je velice obtížné tyto výsledky přesně interpretovat. Naproti tomu syntetické benchmarky se snadno vyhodnocují, ale nevrací tolik reprezentativní výsledky jako předchozí benchmarky, a to právě proto, že jednotlivé testy jsou vytvořeny uměle.

2.2 Proces benchmarkingu

Samotný proces benchmarkingu probíhá v několika specifických krocích. Nejprve je třeba určit, co bude měřeno. Dalším krokem je návrh benchmarku s ohledem na měřené veličiny. Následuje spuštění benchmarku a po něm vyhodnocování výsledků. Pokud je benchmark používán pro vědecké a výzkumné účely, je vhodné, aby jeho autor svůj benchmark nakonec také dostatečně popsal a usnadnil tak ostatním porozumět účelu a výsledkům jeho práce, případně na jeho práci navázat a rozšířit ji [Sma97].

2.2.1 Co a jak měřit

Při stanovení měřené veličiny se rozhodujeme na základě odlišností měřených systémů. Pokud měříme systémy, které se od sebe liší jen v některých aspektech (např. různé verze stejného souborového systému) a jinak jsou si velmi podobné, používá se **mikrobenchmark**, protože se dokáže nejlépe zaměřit na **zvolenou veličinu**, která určuje rozdíly mezi těmito systémy. Poté se ještě spouští **makrobenchmark**, aby bylo možné zjistit, jak chování námi zvolené veličiny ovlivňuje **celý systém** pod reálnou zátěží (viz 2.1.2).

V opačném případě, když se měřené systémy odlišují razantněji (různé operační systémy, hardwarové architektury), se používá makrobenchmark, který určí hlavní rozdíly mezi systémy. Pro zjištění nebo upřesnění toho, jak silný vliv mají jednotlivé odlišnosti systémů na celkový výkon, lze ještě následně nasadit příslušné mikrobenchmarky.

2.2.2 Návrh benchmarku

V případě, že jsou stanoveny veličiny, které se budou měřit, přichází na řadu rozhodování jaký benchmark zvolit. Pokud v dané situaci nevyhovuje žádný z dostupných standardních benchmarků, nezbývá než navrhnout a vytvořit benchmark vlastní.

Při návrhu benchmarku se požadavky liší podle toho, jestli se jedná o mikrobenchmark nebo makrobenchmark. U mikrobenchmarků je klíčové, aby výsledky **nebyly příliš ovlivňovány** různými doprovodnými operacemi benchmarku. Výsledky takového měření mohou být nepřesné nebo dokonce klamavé.

U makrobenchmarků převládá snaha o testování pomocí **zátěže z reálného světa**, ovšem takových případů, kdy je to možné, není mnoho. Většinou se testovací sada přizpůsobí tak, aby přesně vyhovovala danému případu, z čehož ale vyplývá, že aplikace takto vytvořeného benchmarku na jiné případy je velmi obtížná. Další možností je použití uměle vytvořené testovací sady, která by vyhovovala většímu množství případů, ale zároveň je nutné kontrolovat, aby takový benchmark byl reprezentativní nebo aby dokonce nebyl v rozporu s chováním reálné aplikace. Vhodným způsobem, jak tyto problémy eliminovat, je navrhnout **syntetický benchmark** tak, aby se v něm využilo znalostí získaných z reálného prostředí. Příkladem takového benchmarku je

LADDIS benchmark (součást testovací sady SPEC SFS[4]), který se zaměřuje na výkon NFS[5] serverů.

Společným problémem při návrhu všech typů benchmarků je zajistit, aby byly jejich výsledky co nejméně zkreslovány nežádoucími systémovými vlivy. Ty lze např. v případě testování souborového systému eliminovat tak, že se po několika iteracích testování odpojí měřený diskový oddíl, následně se opět připojí a pokračuje se v testování. Touto metodou se eliminuje vliv vyrovnávací paměti na výkon samotného souborového systému. Další možností je restart celého systému.

2.2.3 Průběh testování

Pokud opakovaně spouštíme benchmark na stejném systému beze změn, očekáváme, že průběh a výsledky testování budou shodné nebo velmi podobné. V momentě, kdy tomu tak není, je třeba určit původ těchto výkyvů. Rozdíly při testování mohou být způsobeny např. neočekávanými systémovými událostmi nebo také chybným návrhem benchmarku. V opačném případě, když se změní nastavení systému, očekáváme, že se provedené **změny při testování projeví** a uživatel na základě výsledků benchmarku bude schopen tyto změny rozpoznat a vyhodnotit jejich vliv na chování měřeného systému. Je důležité si uvědomit, že během testování by se mělo **měnit pouze nastavení měřeného systému** a ne příslušného benchmarku. Výsledky běhu upraveného benchmarku by neměly být srovnávány s výsledky běhu původního benchmarku, i když se provedené změny mohou na první pohled jevit jako nepodstatné.

V některých případech testování (např. testování systému jako celku) se snažíme nastavit prostředí benchmarku tak, aby co nejvíce odpovídalo reálnému prostředí, a to **včetně procesů běžících na pozadí**, které jsou v jiných situacích nežádoucí. Snažíme se tedy společně s benchmarkem spouštět různý počet procesů v různých fázích životního cyklu (vytvořený, běžící, blokovaný, čekající, ukončený, atd.) přesně jako v reálném systému, přičemž ale musíme zajistit, aby vliv těchto procesů nebyl příliš velký a neohrozil tak relevantnost výsledků.

[4]System File Server
[5]Network File System

2.2.4 Vyhodnocování výsledků

Ještě před vyhodnocováním benchmarku je vhodné na základě znalostí měřeného systému a jeho chování **odhadnout, jak budou výsledky vypadat** a poté je porovnat s naměřenými hodnotami. Na místa, kde vzniknou menší či větší odchylky od předpokládaných hodnot, se potom snažíme zaměřit.

Obecně se vychází z předpokladu, že každý výsledek benchmarku, ať už vypadá jakkoliv, má svoje **logické vysvětlení**. Původ neočekávaných výsledků může být jak na straně benchmarku, tak na straně měřeného systému a někdy je k jejich vysvětlení zapotřebí detailních znalostí dané problematiky (např. pokud testujeme souborový systém, předpokládá se, že známe jeho vlastnosti a dostatečně rozumíme tomu, jak funguje).

Pokud dokážeme nějakým způsobem vysvětlit, čím jsou neočekávané hodnoty zapříčiněné, měli bychom ještě **ověřit**, zda je toto odůvodnění správné. Například v případě domněnky, že za tyto výsledky může výkon procesoru, použijeme navíc vhodný nástroj, který bude tento výkon při běhu benchmarku sledovat. Tímto způsobem lze **monitorovat** i ostatní **systémové části a zdroje**, ale je třeba zajistit, aby tyto nástroje příliš neovlivňovaly výsledky benchmarku. Je třeba dodat, že není správné věnovat se pouze těm částem výsledků, které nebyly očekávány. Proces schvalování výsledků by měl být prováděn i když jsou ověřovány předpokládané výsledky.

Při vyhodnocování výsledků vlastního benchmarku, může jeho autor narazit také na různé chyby a nedostatky v návrhu nebo implementaci. Díky tomu může být **benchmark následně upraven** a tato nová verze tak bude poskytovat věrohodnější výsledky. Tyto problémy odpadají při používání standardizovaných benchmarků, kdy se uživatel může soustředit pouze na změny nastavení měřeného systému.

2.2.5 Popis benchmarku

Závěrečnou fází procesu benchmarkingu je popis vytvořeného benchmarku, který je vhodný zejména pokud se daný benchmark používá pro vědecké účely. Je důležité, aby autor benchmarku poskytl ostatním dostatek informací o jeho práci. Tyto informace mohou být totiž velmi cenné pro ostatní odborníky, kteří je mohou využít ve svých pracích.

Samotný popis by měl obsahovat informace o tom proč byl benchmark vytvořen, jaký je jeho účel, měřené veličiny, způsob měření, atd. Pokud jsou přiloženy naměřené výsledky, je důležité popsat také parametry testovaného systému.

2.3 Časté chyby při vyhodnocování

Při vyhodnocování výkonu různých systémů bychom se měli držet doporučované metodologie (viz 2.2.4). I přes snahu dodržet tato pravidla mohou vzniknout při vyhodnocování chyby. Uvedeme si ty nejčastější [Jai91].

- **Absence cílů:** Před vyhodnocováním výkonu by mělo být přesně určeno, čeho chceme dosáhnout a proč toto hodnocení vlastně provádíme.
- **Nesystematický přístup:** Kromě stanovení cílů je důležité přesně určit také parametry[6], proměnné[7], metriky[8] a testovací zátěže systému.
- **Nevhodně zvolené metriky:** Typickým příkladem špatně zvolené metriky je porovnání dvou procesorů s instrukčními sadami CISC a RISC[9] metrikou MIPS[10], protože instrukční sady obou procesorů jsou velmi odlišné.
- **Nevhodně zvolená testovací zátěž:** Použitá testovací zátěž by měla být reprezentativní (viz 2.1).
- **Zanedbávání důležitých parametrů a proměnných:** Při určování těchto charakteristik systému by zároveň měla být stanovena jejich důležitost, aby při hodnocení nedošlo k přehlédnutí těch nejdůležitějších.
- **Žádná nebo chybná analýza:** Nestačí pouze shromáždit velké množství naměřených dat, ale je důležité získná data vhodně analyzovat, jinak nemají provedená měření žádný význam.
- **Nevhodné vyhodnocení ojedinělých hodnot:** Ve výsledcích se mohou objevit hodnoty příliš velké nebo malé oproti ostatním. Je nutné

[6] atributy, které ovlivňují výkon systému
[7] parametry, které se během testování mění
[8] kritérium pro vyčíslení výkonu systému
[9] komplexní a redukovaná instrukční sada
[10] milióny operací za sekundu

správně určit jejich původ a podle toho je buď ignorovat, nebo zahrnout do analýzy.

2.4 Důvěryhodnost benchmarků

Jak je popsáno výše, benchmarking je proces, pomocí kterého se snažíme zjistit výkon testovaného programu nebo zařízení. Může nám také pomoci při hledání slabých a silných stránek nějakého systému. V komerční sféře se ale bohužel benchmarky stále více využívají jako **nástroj marketingu** a není divu, že odborná veřejnost se o nich vyjadřuje např. takto: "There are lies, damn lies, and then benchmarks."(Existují lži, naprosté lži a potom benchmarky) [Lay09].

Firmy, nabízející své produkty s pomocí benchmarků, poskytují zákazníkům mnohdy **zkreslené a nepravdivé informace** o výkonu daného produktu v porovnání s ostatními. Metodika a výsledky benchmarků zpravidla nebývají nijak firmami popsány a vysvětleny a to hlavně proto, že samotný benchmark je navržen tak, aby vyhovoval nabízenému produktu. Naopak se také stává, že firmy optimalizují svůj produkt tak, aby měl dobré výsledky v některém z všeobecně uznávaných benchmarků.

2.5 Dostupné benchmarky mobilních zařízení

Vzhledem k tomu, že počet tzv. smartphonů[11] a tabletů stále roste, můžeme na internetu nalézt odpovídající množství různých benchmarků. V tomto přehledu se zaměříme na benchmarky pro tři nejvíce používané operační systémy, kterými jsou Android, iOS a Windows Phone.

2.5.1 Android

Antutu benchmark

Tento benchmark se řadí mezi nástroje, které testují všechny části mobilního zařízení (procesor, paměť, grafika, úložiště). Dále nabízí možnost srovnání s ostatními uživateli na světě, pokud se rozhodnete své výsledky nahrát k centrálnímu zpracování. Zde je seznam testů, které lze provádět [Kil12][Ant12]:

- rychlost operační paměti
- výkon procesoru (celá čísla nebo čísla s plovoucí desetinnou čárkou)
- výkon grafiky (2D nebo 3D)
- rychlost operací na externím úložišti (čtení, zápis)
- rychlost operací s databází

Quadrant Standard Edition

Quadrant je dalším benchmarkem, který se nezaměřuje pouze na jeden aspekt výkonu. Verze Standard Edition je neplacená, ale neobejde se bez reklam. Oproti Antutu benchmarku nabízí přímo v mobilní aplikaci grafy se srovnáním s ostatními zařízeními. Tento benchmark nabízí velké množství testů, z toho jeden kompletní, který dává uživateli detailní informace o celkovém výkonu zařízení [Kan11][Aur10].

- výkon procesoru (aritmetické operace, parsování XML, dekódování multimédií)

[11] telefon s pokročilými funkcemi (multimédia, internet, kancelář)

- rychlost operační paměti
- rychlost vstupně-výstupních operací (souborový systém, databáze)
- výkon grafiky (2D nebo 3D)

Smartbench 2012

Poslední komplexní benchmark, jenž si uvedeme, je Smartbench, který se stal poměrně oblíbeným díky tomu, že jako jeden z prvních podporoval vícejádrové procesory. Odlišností od ostatních benchmarků je to, že se uživatel na konci testování dozví, jak jeho zařízení obstálo při testu běžných operací (Productivity), ale také v testu herního výkonu (Games) [Kil12].

AndroBench

Tento benchmark se soustředí přímo na výkon interního nebo externího úložiště. Další funkcí tohoto benchmarku je testování výkonu operací s databází. Jednotlivé výsledky jsou zobrazovány jednoduše jako číselné hodnoty (grafy nezobrazuje). Dále je zde možnost porovnání s ostatními zařízeními. Aplikace také udržuje historii s dříve vykonanými testy. Před testováním může uživatel nastavit cílové úložiště (interní, externí), velikost souboru, velikost bufferu a počet databázových transakcí [And12]. Tímto benchmarkem lze testovat:

- rychlost čtení (sekvenční nebo náhodné)
- rychlost zápisu (sekvenční nebo náhodné)
- databázové operace (insert, update, delete)

Linpack for Android

Nakonec si uvedeme nástroj, který měří výkon systému pomocí výpočtů s čísly s plovoucí desetinnou čárkou. Během testu se měří čas, který je potřebný na vyřešení soustavy $N*N$ lineárních rovnic metodou Gaussovy eliminace. Z asymptotické složitosti zvoleného algoritmu vyplývá, že na vyřešení této úlohy je zapotřebí přibližně N^3 operací s plovoucí desetinnou čárkou. Výsledný výkon je udáván v MFLOPS [Gre10].

Je třeba poznamenat, že existuje velké množství jiných benchmarků, které provádí podobné testy (např. místo řešení soustav rovnic se počítá desetinný rozvoj čísla π atd.). V tomto přehledu jsou uvedeny ty nejpoužívanější.

2.5.2 iOS

Operační systém iOS najdeme v mobilních zařízeních od firmy Apple. Těchto zařízení není mnoho (vždy několik verzí telefonu iPhone, multimediálního přehrávače iPod, tabletu iPad) a vzhledem k tomu, že pochází od jednoho výrobce, není např. v porovnání s operačním systémem Android tolik dostupných benchmarků. Logicky vzato ani nevzniká potřeba je vyvíjet, protože výkonnostní rozdíly jednotlivých zařízení jsou přímo dány jejich specifikacemi.

Dají se však najít případy, kdy měření výkonu smysl má (např. rozdíl výkonu před a po aktualizaci systému) a proto si některé nástroje k tomu vhodné uvedeme [Wol11].

Geekbench

Tento benchmark od společnosti Primate Labs měří výkon procesoru a operační paměti. Je to velmi jednoduchý nástroj, který provádí předdefinované testy bez možnosti detailnějšího nastavení. Výhodou je podpora vícejádrových zařízení a také možnost spustit tuto aplikaci na mnoha operačních systémech (Mac OS X, Windows, Linux, Android a iOS) a porovnat tak výkon velkého množství zařízení [Pri12].

Gauge Mathematical Tool

Jak je vidět z názvu, testování pomocí tohoto benchmarku probíhá na základě matematických operací. Testované zařízení se zatěžuje např. výpočty desetinného rozvoje čísla pí až do 100 miliónů míst, generováním prvočísel nebo zobrazováním Mandelbrotova fraktálu. Pomocí tohoto nástroje lze testovat [Hun10]:

- výkon procesoru
- výkon operační paměti
- rychlost čipové sady a sběrnic
- výkon grafiky
- rychlost čtení a zápisu daného úložiště

2.5.3 Windows Phone

WP Bench

Windows Phone Bench je nástroj, pomocí kterého mohou testovat výkon uživatelé vlastnící zařízení se systémem Windows Phone 7. V první verzi tohoto benchmarku ještě nebyly dostupné grafy, ale s uvolněním druhé verze přišly jak grafy, tak také možnost porovnání vlastních výsledků s ostatními formou zaslání dat na server. WP Bench se řadí mezí benchmarky, které vyhodnocují všechny aspekty výkonu zařízení, a to [Var11]:

- výkon procesoru (jedno vlákno, více vláken)
- rychlost operační paměti (čtení, zápis)
- rychlost úložiště (čtení, zápis)
- výkon grafiky
- zobrazení barev na displeji
- výdrž baterie

3 Popis testů

Při návrhu popisu testů byl důraz kladen především na to, aby byl popis co **nejuniverzálnější** (tzn. aby bylo možné definovat testy pro různé platformy). V současné podobě návrhu je možné definovat testy pro interní a externí úložiště mobilního zařízení a také testy výpočetního výkonu.

Jedním z cílů této práce bylo připravit testovací aplikaci tak, aby byla v budoucnu použitelná i pro testování souborového systému KIVFS[1]. To bude prováděno pomocí KIVFS klienta pro Android, přičemž návrh popisu testu zůstane **nezměněn**. Drobných změn dozná pouze aplikace, která bude popisem definované operace fakticky provádět.

Jednotlivé testy, ať už jde o testy souborového systému nebo výpočetního výkonu, jsou popsány pomocí značkovacího jazyka XML[2]. Struktura XML souboru byla navržena tak, aby sestavování jednotlivých testů bylo jednoduché a univerzální.

3.1 Test souborového systému

Při testování souborového systému je třeba nadefinovat několik důležitých parametrů. V první části popisu se definují základní parametry testu jako např. počet opakování, úložiště nebo typ prováděné operace. Následuje popis adresářové struktury a popis jednotlivých souborů.

3.1.1 Základní definice

Nejprve je třeba definovat, o jaký typ testu se jedná. V kořenovém elementu `<test>` je proto atribut `category`, který je v případě testu souborového systému nastaven na hodnotu `filesystem`. Dále se v tomto elementu nastavuje název a počet opakování testu (viz Kód 3.1).

[1] souborový systém vyvíjený na katedře informatiky a výpočetní techniky ZČU
[2] Extensible Markup Language

Kód 3.1: Element test

```
<test category="filesystem" name="big file 2"
 repeat="20">
```

Jako další přichází na řadu určení úložiště, které se vyplní v elementu `<storage>`. Pokud chceme testovat na interní paměti zařízení musí být nastavena hodnota `memory` a pokud se testuje na paměťové kartě tak `SD card`. V následujícím elementu `<operation>` je stanoveno, jaká operace bude při testu prováděna. Může to být čtení (`read`), zápis (`write`) nebo mazání (`delete`).

Posledním bodem úvodní definice je nastavení velikosti bloku dat pro zápis souborů. Tento parametr se nastavuje v elementu `<blocksize>` a určuje velikost částí, po kterých se bude soubor zapisovat. Tento parametr výrazně ovlivňuje dobu zápisu celého souboru. Při stanovení velikosti bloku je nejprve nutné jako atribut elementu `<blocksize>` zvolit jednotku (`unit`) (viz Kód 3.2).

Kód 3.2: Základní definice

```
<test category="filesystem" name="big file 2"
 repeat="20">
    <storage>SD card</storage>
    <operation>write</operation>
    <blocksize unit="MB">1</blocksize>
```

3.1.2 Definice adresářové struktury

Po definici základních informací o testu přichází na řadu popis adresářové struktury. To se provádí uvnitř elementu `<directories>`, který má jeden atribut `description`. Existují dva způsoby (`exact` a `non-exact`), kterými lze složky definovat. První možností je popsat složku přesně jejím názvem a zanořením do ostatních složek (viz Kód 3.3).

Kód 3.3: Přesný popis adresářové struktury

```
<directories description="exact">
    <directory>
        <name>dir2/</name>
    </directory>
    <directory>
```

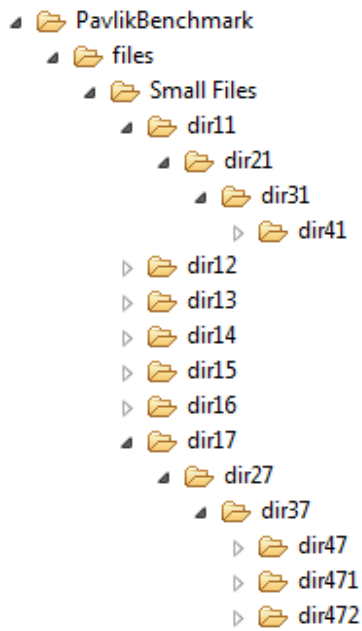
```
        <name>dir2/dir21/</name>
    </directory>
</directories>
```

Druhým způsobem jak popsat adresářovou strukturu je určení hloubky zanoření a počet složek, které mají být vytvořeny. Hloubka se určuje v elementu `<depth>` a počet složek v elementu `<count>` (viz Kód 3.4).

Kód 3.4: Popis adresářové struktury pomocí hloubky a počtu složek

```
<directories description="non-exact">
    <depth>4</depth>
    <count>30</count>
</directories>
```

Složky se vytvářejí rovnoměrně v každé úrovni zanoření, což znamená, že v každé úrovni bude vytvořen stejný počet složek. Pokud stanovená hloubka není dělitelem počtu souborů, bude v poslední úrovni vytvořeno více složek tak, aby byl dodržen přesný počet složek. V tomto příkladu zadání (Kód 3.4) to znamená, že budou vytvořeny tři úrovně s počtem složek 7 a poslední úroveň bude mít 9 složek (viz Obrázek 3.1).



Obrázek 3.1: Rozložení složek

3.1.3 Definice souborů

Poslední částí XML souboru, jenž popisuje daný test, je definice souborů, které se mají vytvořit. Jednotlivé soubory se definují uvnitř elementu `<files>`, který má dva atributy. Prvním atributem (`description`) se určí jakým způsobem budou soubory popsány. Může to být stejně jako v případě složek přesný (`exact`) popis nebo určení pomocí počtu souborů (`non-exact`).

Druhý atribut (`type`) nám říká, jaká data se do souboru zapisují. Zapisovat lze buď samé nuly (`zeros`), jedničky (`ones`) nebo náhodný vzor (`random`) (viz Kód 3.5).

Kód 3.5: Definice souborů

```
<files description="exact" type="random">
```

V případě přesného popisu se soubory definují v elementu `<file>`. U každého souboru je třeba určit jeho název (element `<name>`), velikost (`<size>`) a složku (`<directory>`), ve které má být vytvořen. Tato složka musí být uvedena v definici adresářové struktury (viz 3.1.2).

Velikost souboru lze určit dvěma způsoby. Buď uvedením přesné velikosti (atribut `description` je nastaven na `exact`), nebo stanovením intervalu, ve kterém se může velikost souboru pohybovat (`range`). Element `<size>` má stejně jako element `<blocksize>` atribut `unit`, kterým se určuje jednotka velikosti souboru (viz 3.1.1). Příklad správně definovaného souboru je na ukázce Kód 3.6.

Kód 3.6: Definice souboru přesnou velikostí

```
<files description="exact" type="random">
    <file>
        <name>file1</name>
        <size unit="MB" description="exact">150</size>
        <directory>dir1/</directory>
    </file>
    ...
</files>
```

Pokud se velikost souboru zadává intervalem, je nutné určit jeho meze. To se provádí v elementech `<min>` a `<max>`. Jednotka těchto hodnot se definuje stejně jako u přesného zadání velikosti atributem `unit` v elementu `<size>`

(viz Kód 3.7).

Kód 3.7: Definice velikosti souboru intervalem

```
<files description="exact" type="random">
    <file>
        <name>file1</name>
        <size unit="MB" description="range">
            <min>10</min>
            <max>12</max>
        </size>
        <directory>dir1/</directory>
    </file>
    ...
</files>
```

Pokud soubory definujeme jejich počtem, stačí výše popsaným způsobem určit jejich velikost a v elementu `<count>` uvést počet souborů (viz Kód 3.8).

Kód 3.8: Definice souborů jejich počtem

```
<files description="non-exact" type="random">
    <size unit="kB" description="range">
        <min>1</min>
        <max>2</max>
    </size>
    <count>500</count>
</files>
```

Při tomto způsobu popisování souborů se neudává složka, ve které se má soubor vytvořit. Soubory se vkládají do složek, které jsou nejhlouběji v definované adresářové struktuře. Soubory se do složek distribuují rovnoměrně. Pokud se má např. vytvořit 530 souborů v 50ti složkách, bude v každé složce minimálně 10 souborů s tím, že 30 složek bude mít 11 souborů.

3.2 Test výpočetního výkonu

Oproti popisům testů, které jsou zaměřeny na souborový systém, jsou popisy výkonových testů velmi jednoduché a jejich velikost závisí pouze na počtu parametrů. Díky tomu, že tyto testy nemají mnoho společných vlastností,

nemohou mít jejich popisy zcela jednotnou podobu. Pro ilustraci je na ukázce Kód 3.9 jednoduchá šablona tohoto typu testů.

Kód 3.9: Test výpočetního výkonu

```
<test category="performance" name="test name"
 repeat="30">
    <task>task_name</task>
    <param1>value</param1>
    <param2>value</param2>
    ...
</test>
```

Kořenový element `<test>` je stejný jako u testů souborového systému s tím rozdílem, že atribut `category` má hodnotu nastavenou na `performance`. Následuje název úkolu (`task`), který se bude provádět a poté seznam parametrů.

Na ukázce Kód 3.10 je vidět příklad jednoduchého testu, který měří čas potřebný na výpočet prvočísel do stanoveného maxima.

Kód 3.10: Výpočet prvočísel

```
<test category="performance" name="Prime Numbers"
 repeat="30">
    <task>PrimeNumber</task>
    <maximum>100000</maximum>
</test>
```

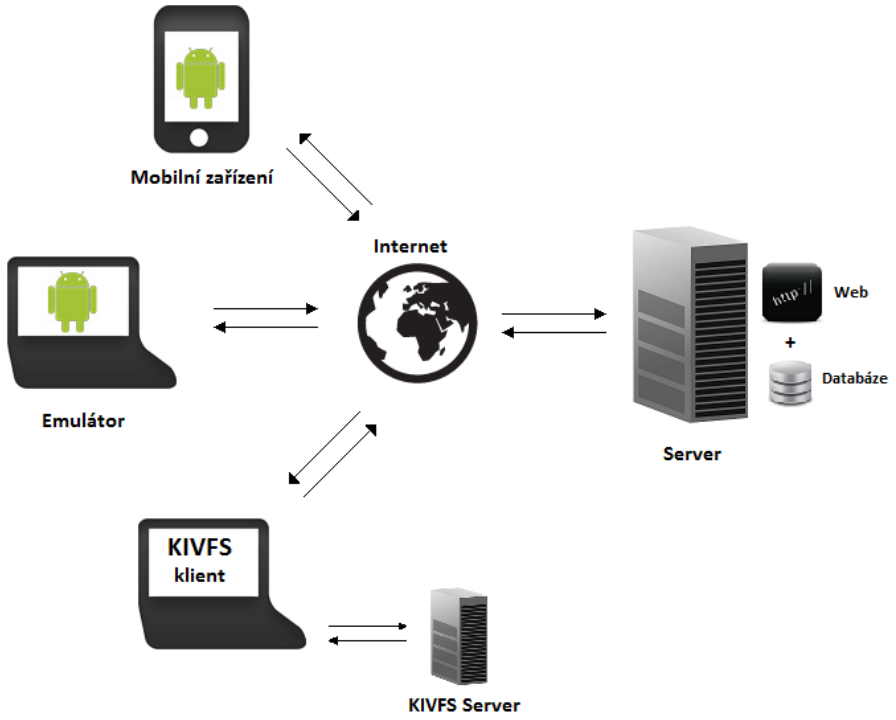
Na rozdíl od popisů testů souborového systému nejsou tyto popisy univerzálně zpracovatelné. Pokud je navržen nový test, je nutné rozšířit program, který zpracovává XML popisy testů tak, aby byl schopen tento nový test načíst a provést.

4 Návrh systému

V této kapitole bude nejprve popsán návrh systému jako celku a následně se podíváme na klientskou a serverovou část.

4.1 Schéma systému

Celý systém je navržen na základě síťové architektury typu **klient-server**. Klient a server spolu komunikují prostřednictvím jednoduchých zpráv, které obě komunikující strany znají (společný protokol). V praxi to vypadá tak, že klient pošle na server nějaký požadavek, ten ho zpracuje (pokud to dokáže a zároveň pokud má klient dostatečná oprávnění pro daný požadavek) a odešle klientovy požadovaná data. Pro ilustraci je na obr. 4.1 schéma celého systému.



Obrázek 4.1: Schéma navrženého systému

Výhodou této je architektury je to, že klient nemusí znát strukturu serveru (pouze požádá o data a ty následně dostane, přičemž ho nezajímá, jakým způsobem je server vyprodukoval). V mém systému server neslouží pouze jako **poskytovatel dat**, ale také jako **úložiště pro data (databáze)**, která jednotliví klienti zasílají zpět (výsledky testů a informace o zařízení).

Ve spodní části obr. 4.1 si můžeme všimnout již zmiňovaného klienta KIVFS a KIVFS serveru (viz 3). Systém je v současné době připraven na integraci KIVFS klienta, která však může být provedena až bude dostupná jeho produkční verze.

Smyslem toho, že byl tento KIVFS klient a server přidán do schématu, je poukázat na různé možnosti využití vyvíjené aplikace. Konkrétní klient totiž nemusí využívat pouze zdroje vlastního zařízení (jako je tomu u mobilního zařízení a emulátoru), ale obecně může využívat i další **síť'ové zdroje** (KIVFS klient posílá po síti příkazy KIVFS serveru, který je provádí a testovací aplikace je zpracovává stejným způsobem, jako by to byly příkazy prováděné přímo na samotném klientovi).

4.2 Komunikace mezi klientem a serverem

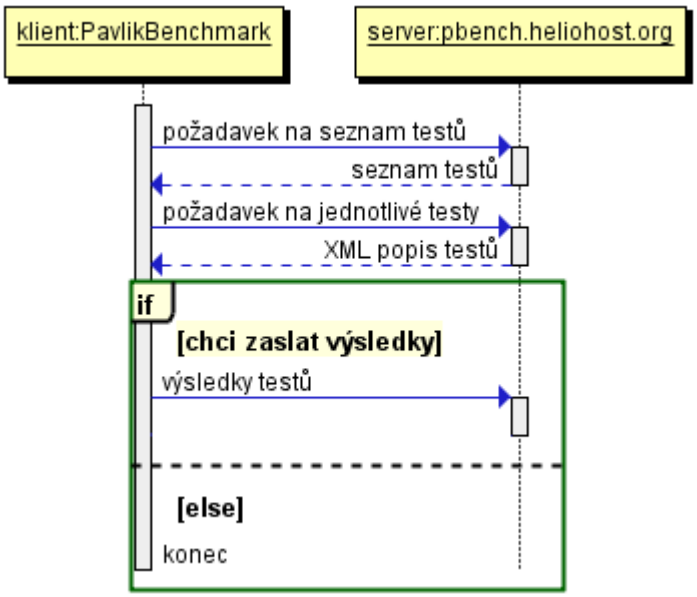
Klient a server spolu komunikují celkem ve třech krocích, přičemž poslední krok je podmíněný a závisí na volbě klienta. Průběh komunikace je zachycen na sekvenčním diagramu (obr. 4.2).

V první fázi komunikace pošle klient na server požadavek na **seznam testů**. Server na základě přijatého požadavku zašle klientovi soubor se zdrojovým kódem stránky, na které se popisy testů nacházejí. Klient poté tyto informace zpracuje a zobrazí uživateli seznam testů, které je server momentálně schopen zaslat.

Dále si uživatel ze seznamu dostupných testů vybere ty z nich, které chce provést a potvrdí svůj výběr. Klientská aplikace tedy zašle na server požadavek na jednotlivé **popisy testů**, které si uživatel vybral. Server pošle zpět klientovi XML soubory s popisy jednotlivých testů.

Po zpracování těchto XML souborů následuje samotné testování. Po jeho skončení je uživatel dotázán, zda chce **naměřené hodnoty** zaslat k centrálnímu zpracování na server. Pokud uživatel zvolí ano, tak se klientská aplikace

připojí k databázi na serveru a jednoduchým SQL[1] dotazem do ní vloží data o zařízení, provedeném testu a nakonec jednotlivé hodnoty.



Obrázek 4.2: Průběh komunikace

Pokud se uživatel rozhodne, že nechce zasílat výsledky testování a informace o zařízení na server, komunikace je ukončena. Nezávisle na této volbě (poslat nebo neposlat) aplikace uživateli zobrazí výsledky ve formě grafu a číselných hodnot.

4.3 Klient

Klientská aplikace byla vytvářena v programovacím jazyce Java a to tak, aby byla spustitelná na operačním systému Android od verze 2.3.3. (API[2] Level 10) Testování proběhlo i na současné verzi 4.0.4 (API Level 15), která byla uvolněna v březnu 2012. Aplikaci můžeme rozdělit na datovou, zobrazovací a vykonávací část. V následujícím textu si tyto části podrobněji rozebereme.

[1]Structured Query Language - dotazovací jazyk pro práci s relačními databázemi
[2]Application Programming Interface - rozhraní pro programování aplikací

4.3.1 Datová část

Dříve než aplikace může začít testovat, musí nashromáždit potřebná data o konkrétních testech. V podkapitole 4.2 je popsáno, jak klient data od serveru získá a poté následuje zpracování přijatých dat.

Získání seznamu testů

Nejprve se zpracovává výpis adresáře, který se nachází na serveru a obsahuje popisy jednotlivých testů. Názvy jednotlivých testů se získávají parsováním zdrojového kódu jednoduché webové stránky (jazyk HTML[3]). Řetězce, které se aplikace snaží získat jsou uloženy v tabulce (HTML tagy `<table>`, `<td>`, `<tr>`). Struktura zdrojového souboru je naznačena v ukázce Kód 4.1.

Kód 4.1: Struktura souboru s názvy testů

```
<td><a href="test_1.xml">test_1.xml</a></td>
<td><a href="test_2.xml">test_2.xml</a></td>
<td><a href="test_3.xml">test_3.xml</a></td>
```

V cyklu do konce souboru se načítají jednotlivé řádky a z nich se získávají názvy testů. Ty jsou v první fázi ještě s podtržítkem mezi jednotlivými slovy (např. `FS_Big_Files_EW.xml`), ale ještě před předáním dat k zobrazení se všechna podtržítka nahradí mezerami.

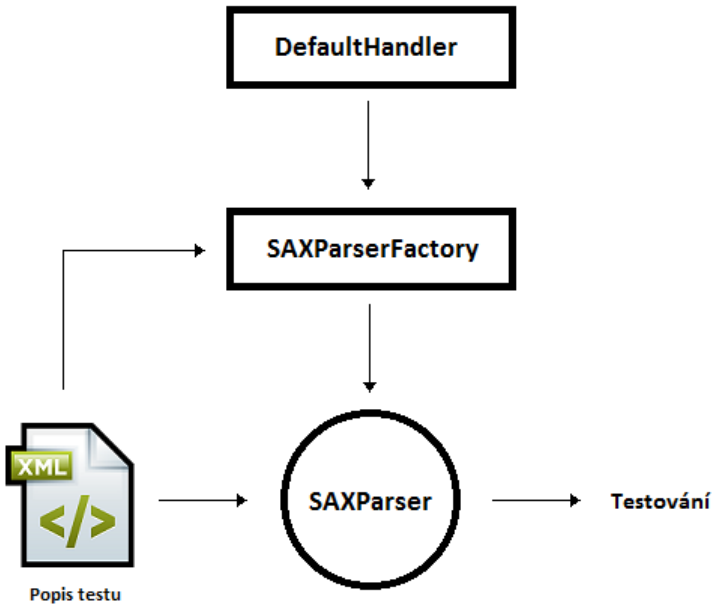
Zpracování popisů testů

Po získání seznamu testů si uživatel vybere, které testy chce provést. Popisy vybraných testů se následně stáhnou do zařízení k dalšímu zpracování. Jednotlivé testy jsou popsány XML souborem, jehož struktura je probrána v kapitole 3.

V okamžiku, kdy jsou staženy všechny zvolené soubory, přichází na řadu jejich parsování. To je prováděno pomocí rozhraní SAX[4]. Toto rozhraní přistupuje k souboru v režimu sekvenčního čtení (sequential read only access) a pro zpracování celého souboru je zapotřebí pouze jeden průchod. Výhodou je, že soubor, který je zpracováván, nemusí být načten v paměti celý, takže je parsování velmi rychlé a efektivní [Lai03]. Naopak nevýhodou může být, že toto rozhraní nedokáže měnit zpracovávaný soubor (na rozdíl od rozhraní

[3]HyperText Markup Language - jazyk pro tvorbu webových stránek
[4]Simple API for XML parsing

DOM[5]), ale v mé práci jsem využíval pouze čtení. Na obr. 4.3 je naznačeno, jak parsování pomocí rozhraní SAX funguje.



Obrázek 4.3: Parsování pomocí rozhraní SAX

Nejprve je třeba vytvořit **obsluhu událostí**, které chceme zachytit (tzn. budeme definovat, co se má provést, když se parser dostane na konkrétní element nebo atribut elementu v XML souboru). To se provádí např. vytvořením potomka třídy `DefaultHandler`. V této nově vytvořené třídě je třeba přepsat metody `startElement, endElement` a `characters`. Na ukázce Kód 4.2 je znázorněna obsluha události poté, co se parser dostal na element `<name>` uvnitř elementu `<directory>`.

Kód 4.2: Definice Handleru

```
if(directory && name) {
    ((FSTestCase)testCase).addDirectory(value);
    name = false;
}
```

Když je `Handler` definován, stačí vytvořit novou instanci třídy `SAXParserFactory`, pomocí které se vytvoří nová instance třídy `SAXParser`. Tato instance bude nastavena podle parametrů dané `SAXParserFactory`. Nakonec

[5]Document Object Model

se zavolá metoda `parse`, jejímž prvním parametrem je soubor (`File`), který má být parsován, a `DefaultHandler` pro obsluhu událostí (Kód 4.3).

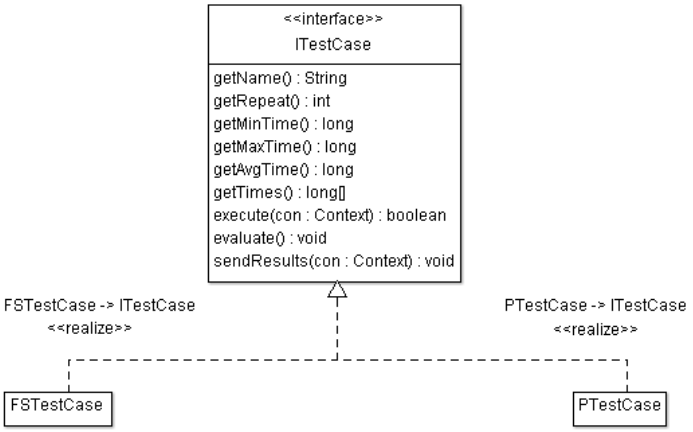
Kód 4.3: Parsování XML souboru

```
factory = SAXParserFactory.newInstance();
sParser = factory.newSAXParser();
sParser.parse(new File(name), handler);
```

Datová reprezentace testů

Všechny atributy jednotlivých testů, které se získají z XML popisu testů, se ukládají do připravených datových struktur. Informace o testu souborového systému se ukládají do instance třídy `FSTestCase` a informace o výkonovém testu do instance třídy `PTestCase`. Obě tyto třídy navíc implementují rozhraní `ITestCase`, pro zjednodušení práce s jednotlivými testy.

Využívá se zde totiž tzv. **polymorfismu (mnohotvarosti)**. V rozhraní `ITestCase` se nadefinují hlavičky metod, které musí všechny třídy, které toto rozhraní implementují přepsat (u každé třídy vypadají tyto metody jinak). Potom, např. při spouštění sady testů (pole objektů typu `ITestCase`), voláme metodu `execute` jednoduše nad objektem `ITestCase` a provede se kód, který je definován v jednotlivých třídách, jež implementují toto rozhraní (tzn. pokud se za objektem `ITestCase` skrývá test výkonu, provede se metoda `execute` definovaná v třídě `PTestCase`). Pro ilustraci je na obr. 4.4 diagram těchto tříd.



Obrázek 4.4: Diagram tříd

4.3.2 Zobrazovací část

Jednotlivé obrazovky, které se uživateli zobrazují, se definují pomocí tzv. **aktivit** (`android.app.Activity`). Každá obrazovka v dané aplikaci je tedy popsána v nějaké třídě, která je potomkem třídy `Activity`.

Definice layoutu

Existují dva způsoby, jak lze definovat rozložení jednotlivých komponent na obrazovce. První možností je definovat jednotlivé komponenty **pomocí XML souboru**. Druhým způsobem je definovat prvky přímo **ve zdrojovém kódu aktivity**, což je užitečné např. pokud je potřebujeme přidávat dynamicky (za běhu programu, v reakci na nějakou událost).

Nejprve si ukážeme definici layoutu pomocí XML souboru. Jednotlivé komponenty se přidávají do souboru jako elementy, jejichž názvy udávají o jaký prvek se jedná (`LinearLayout, TextView, Button`). Vlastnosti těchto prvků se upravují za pomoci atributů daného elementu (viz Kód 4.4).

Kód 4.4: Definice layoutu XML souborem

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <TextView
        android:id="@+id/title2"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:gravity="center_horizontal"
        android:text="" />
    ...
</LinearLayout>
```

Když je layout nadefinován, stačí už jen v dané aktivitě nastavit XML soubor jako zdrojový, což se provádí metodou `setContentView`. Pro přístup k jednotlivým komponentám slouží metoda `findViewById`, do které se jako parametr vkládá `id`, které bylo uvedeno jako atribut elementu v XML souboru (viz Kód 4.4). Následně lze dané komponentě přiřadit libovolnou funkcionalitu (např. definovat co se má provést po stisku tlačítka).

Layout lze také definovat přímo ve zdrojovém kódu aktivity. Tuto variantu jsem využil v aktivitě `ChooseTestActivity`, kde bylo nutné přidávat zaškrtávací tlačítka (`CheckBox`) v závislosti na seznamu testů, který je získán ze serveru.

Nejprve je nutné definovat nějaký typ layoutu (v mém případě `LinerLayout`) a pokud očekáváme, že zobrazovaný obsah přesáhne velikost obrazovky, je dobré celý layout ještě vložit do `ScrollView`, pomocí kterého se můžeme libovolně posouvat (rolovat) po celém obsahu obrazovky (viz Kód 4.5).

Následně se ještě pomocí jednoduchých konstruktorů (jediným parametrem je kontext aplikace) vytvoří ostatní prvky, které mají být zobrazeny. Pro zjišťování nebo nastavování vlastností těchto prvků slouží getry a setry, které lze dohledat v dokumentaci jednotlivých tříd (viz Kód 4.5).

Kód 4.5: Vytvoření a nastavení komponent

```
final ScrollView sView = new ScrollView(this);
final LinearLayout layout = new LinearLayout(this);
final TextView title = new TextView(this);
final Button download = new Button(this);

download.setText(R.string.download);
download.setOnClickListener(new DownloadBtListener());
title.setGravity(Gravity.CENTER_HORIZONTAL);
```

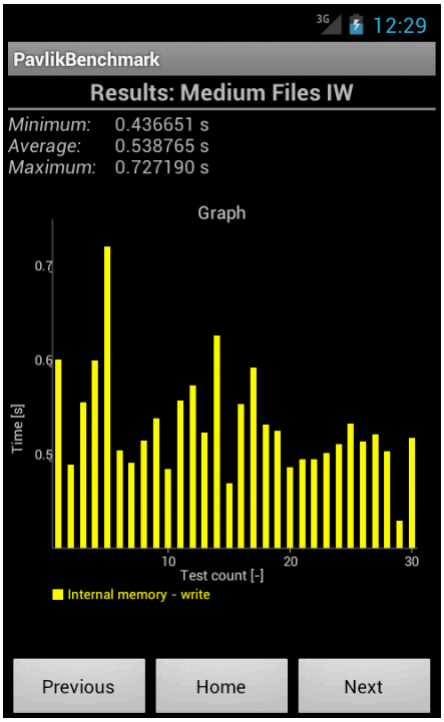
Na závěr, když jsou všechny komponenty nadefinované a nastavené, stačí je pouze metodou `addView` vložit do layoutu a ten podobně jako u definice pomocí XML souboru nastavit jako zdrojový metodou `setContentView` (viz Kód 4.6).

Kód 4.6: Vkládání komponent

```
for(int i = 0;i < testList.size();i++) {
    chBoxes[i] = new CheckBox(this);
    chBoxes[i].setText(testList.get(i).replace('_',' '));
    layout.addView(chBoxes[i]);
}
layout.addView(download);
sView.addView(layout);
setContentView(sView);
```

Zobrazení výsledků

Zobrazení naměřených hodnot je realizováno ve třídě `EvaluateActivity`. V horní části obrazovky jsou vypsány tři hodnoty (minimum, maximum, průměr). Pod nimi se nachází graf, ve kterém jsou vyneseny všechny hodnoty provedeného testu. Pro procházení jednotlivých testů a návrat na počáteční obrazovku jsou ve spodní části tři tlačítka (viz obr. 4.5).



Obrázek 4.5: Zobrazení výsledků

Graf je vytvářen pomocí knihovny `AChartEngine` ve verzi 0.7.0. Nejprve je třeba v metodě `buildXYRenderer` nastavit vlastnosti grafu (velikost textu, barva textu, barva os a sloupců, odsazení, atd.). Po nastavení vzhledu grafu, se předají naměřená data (do datové sady `XYMultipleSeriesDataset` se vkládají řady hodnot `XYSeries`). Následně se metodou `setChartSettings` nastaví všechny popisky, minimální a maximální hodnoty.

Hotový graf, reprezentovaný jako instance třídy `GraphicalView` (potomek třídy `android.view.View`), se vytvoří voláním metody `getBarChartView` ze třídy `ChartFactory`, která má jako parametry kontext aplikace, datovou sadu, nastavený renderer a typ sloupcového grafu.

4.3.3 Testování

Když jsou nashromážděna všechna potřebná data a uživatel potvrdí svůj výběr testů, přejde se k samotnému testování. Metody, které se využívají v testech souborového systému, se nacházejí ve třídě `IOOperations` a metody pro test výpočetního výkonu ve třídě `PTestCase`.

Operace souborového systému

V testech souborového systému se měří časy operací čtení, zápisu a mazání souboru. Ve třídě `IOOperations` jsou metody, které tyto operace provádí pro jeden soubor a v případě čtení a zápisu jsou zde metody, které přečtou (resp. smažou) celý obsah složky včetně podsložek.

Pro vytvoření a **zápis** do souboru je zde metoda `createFile`, která má následující parametry:

- `String name` - název souboru (cesta v souborovém systému)
- `long size` - velikost souboru v bytech
- `int blockSize` - velikost zapisovaného bloku dat v bytech
- `int storage` - typ úložiště (interní, externí)
- `int mode` - typ zapisovaných dat (jedničky, nuly, střídání)
- `Context con` - kontext aplikace

Nejprve se na základě velikosti zapisovaného bloku a typu zapisovaných dat naplní `buffer`. Pokud hodnoty velikosti souboru a velikosti zapisovaného bloku nejsou dělitelné, je navíc třeba pro přesné dodržení velikosti souboru vytvořit pole bytů (`rest`), které má velikost danou zbytkem po celočíselném dělení velikosti souboru velikostí zapisovaného bloku. Toto pole se zapíše na konec souboru.

Pokud se zapisuje na externí úložiště použijeme instanci třídy `RandomAccessFile` v režimu `READ_WRITE_MODE` (čtení a zápis). V cyklu se až do naplnění velikosti souboru zapisuje předem připravený `buffer`, přičemž se pomocí metody `System.nanoTime()` měří čas celé operace (viz Kód 4.7).

Kód 4.7: Zápis do souboru

```
RandomAccessFile raf = new RandomAccessFile(file,
 READ_WRITE_MODE);

start = System.nanoTime();
for(int i = 0;i < (size / blockSize);i++) {
    raf.write(buffer);
}
if(rest != null) raf.write(rest);
time = System.nanoTime() - start;
raf.close();
```

V případě zápisu na interní úložiště je postup stejný s tím rozdílem, že se pro zápis využívá namísto `RandomAccessFile` instance třídy `DataOutputStream`.

Pro čtení souboru se používá metoda `readFile` s parametry:

- `String name` - název souboru (cesta v souborovém systému)
- `int storage` - typ úložiště (interní, externí)
- `int blockSize` - velikost čteného bloku dat v bytech
- `Context con` - kontext aplikace

Podobně jako u zápisu se také u **čtení** nejprve vytvoří `buffer` o velikosti bloku dat (`blockSize`), který ovšem v tomto případě bude sloužit pro uložení čtených dat. Data se čtou v cyklu až do dosažení konce souboru. Čas se měří stejným způsobem jako při zápisu (`System.nanoTime()`).

Nejjednodušší z těchto tří operací je operace **mazání** souboru. Ta se provádí v metodě `deleteFile`, která má parametry:

- `String name` - název souboru (cesta v souborovém systému)
- `int storage` - typ úložiště (interní, externí)
- `Context con` - kontext aplikace

K souboru se přistupuje pomocí instance třídy `File` se zadanou cestou k souboru. Soubor se pak smaže jednoduše voláním metody `delete()`.

Na operaci mazání si ukážeme, jak se prochází celý obsah složky včetně podložek (metoda `deleteFiles`). Postup je stejný jako u čtení kompletního obsahu složky, který se provádí metodou `readFiles`. Obě tyto metody jsou rekurzivní (volají samy sebe).

Jediným parametrem metody je soubor (`File`), který reprezentuje složku identifikovanou svojí cestou v souborovém systému. Metodou `listFiles()` se získají všechny složky a soubory uvnitř současného adresáře. Ty se v cyklu procházejí s tím, že pokud se narazí na složku, tak se rekurzivním voláním metody `deleteFiles` tato složka prohledá a pokud se narazí na soubor, tak se smaže (viz Kód 4.8).

Kód 4.8: Mazání obsahu složky

```
if(f.isDirectory()) {
    for(File child : f.listFiles()) {
        deleteFiles(child);
    }
}
start = System.nanoTime();
f.delete();
time = System.nanoTime() - start;
```

4.3.4 Identifikace zařízení

Existuje několik možností, jak lze identifikovat mobilní zařízení se systémem Android. Protože ale někteří výrobci nevyplňují všechny údaje (např. sériové číslo), bylo nutné zvolit **kombinaci několika způsobů identifikace**. Výsledný klíč jednoznačně identifikuje jakékoliv zařízení s operačním systémem Android [Mot11].

První možností je identifikovat zařízení pomocí **čísla IMEI**[6], což je číslo, které je unikátní pro každý telefon. Pokud by ovšem byla aplikace spuštěna na tabletu, toto číslo by nebylo dostupné.

Dále můžeme vytvořit pseudo-unikátní klíč, který lze sestavit pro všechna

[6]International Mobile Equipment Identity

mobilní zařízení se systémem Android. Tento klíč se skládá z **údajů ze třídy** `android.os.Build`. Délky jednotlivých hodnot z této třídy jsou nejprve děleny deseti. Zbytky po tomto dělení se sečtou a vznikne další část unikátního identifikátoru.

Identifikátor systému Android (Android ID), který se nachází ve třídě `android.provider.Settings.Secure`, je další alternativou identifikace zařízení, ale může se stát, že získaná hodnota `null`.

Další možností je rozlišovat zařízení pomocí **MAC**[7] **adresy** wi-fi adaptéru. Je ovšem jasné, že pokud zařízení wi-fi adaptér nemá, nebude dostupná ani jeho MAC adresa. Situace je naprosto stejná u MAC adresy zařízení se standardem Bluetooth.

Výsledný unikátní klíč, pomocí kterého se v databázi budou zařízení rozlišovat, **sestavíme z výše uvedených identifikátorů** (IMEI, pseudo ID, Android ID, Wi-fi MAC, Bluetooth MAC). Pokud některá z částí klíče není dostupná, je nahrazena implicitní hodnotou. Jednotlivé řetězce se spojí dohromady a následně se toto spojení použije pro výpočet MD5 hashe[8].

4.3.5 Manifest

Manifest je XML soubor, který operačnímu systému poskytuje informace o tom, jaké komponenty jsou dostupné, jaká systémová práva aplikace požaduje, atd.

Nejprve se v manifestu definuje název balíčku (package) se zdrojovými kódy a verze aplikace. Dále se v manifestu mé aplikace stanovuje minimální verze platformy Android, na které lze aplikaci spustit (viz Kód 4.9).

Kód 4.9: Definice minimální verze platformy Android

```
<uses-sdk android:minSdkVersion="10"/>
```

Některé operace, které aplikace provádí, vyžadují zvláštní systémová práva (permissions). V mé aplikace bylo nutné získat celkem 5 oprávnění. Jelikož aplikace komunikuje se serverem přes internet, je nutné povolit připojení k internetu. Další oprávnění je potřebné pro zápis na externí úložiště a kvůli

[7] Media Access Control

[8] Message-Digest - algorimtus, který z libovolného vstupu vytvoří výstup stejné délky

generování unikátního identifikátoru zařízení je třeba navíc povolení přístupu k informacím o stavu telefonu, wi-fi a bluetooth. Na ukázce kódu 4.10 je zmíněno povolení přístupu k internetu.

Kód 4.10: Nastavení oprávnění

```
<uses-permission
    android:name="android.permission.INTERNET" />
```

Jediný element manifestu, který musí být povinně definován, je element `<application>`. Uvnitř tohoto elementu se popisují jednotlivé komponenty aplikace (activity, broadcast reciever, service, content provider). Vytvořená aplikace se skládá z osmi aktivit (viz Kód 4.11).

Kód 4.11: Definice aktivit

```
<application
 android:icon="@drawable/ic_launcher"
 android:label="@string/app_name" >
 <activity
  android:label="@string/app_name"
  android:name=".BenchmarkActivity"
  android:launchMode="singleTask">
  <intent-filter >
   <action android:name="android.intent.action.MAIN" />
   <category
    android:name="android.intent.category.LAUNCHER" />
  </intent-filter>
 </activity>
 <activity
  android:name=".ChooseTestActivity"></activity>
 <activity
  android:name="org.achartengine.GraphicalActivity">
 </activity>
 <activity android:name=".EvaluateActivity"></activity>
 <activity android:name=".SummaryActivity"></activity>
 <activity android:name=".SettingsActivity"></activity>
 <activity android:name=".ErrorActivity"></activity>
 <activity android:name=".IdActivity"></activity>
</application>
```

4.3.6 Použité knihovny

Při vytváření aplikace jsem využil dvě knihovny. Pomocí první knihovny se vykreslují grafy a druhá knihovna se používá na spojení s databází.

AChartEngine 0.7.0

Knihovna AChartEngine pro vytváření grafů je vyvíjena společností 4ViewSoft. Pro volně šiřitelné a nekomerční aplikace je dostupná zdarma (licence Apache 2.0).

Pomocí této knihovny lze vykreslit velké množství různých grafů (např. liniový, plošný, bodový, časový, sloupcový, výsečový či prstencový). U všech těchto grafů lze nastavit velké množství parametrů tak, aby výsledné grafy byly co nejpřehlednější.

MySQL Connector/J 5.1.18

Knihovna MySQL Connector od společnosti Oracle slouží pro připojení klientských aplikací k MySQL databázi pomocí ovladače JDBC[9]. Je volně dostupná pod licencí GPL[10].

Nejprve je nutné připojit se k databázi. To se provádí tak, že se vytvoří nová instance JDBC ovladače a potom se voláním metody `DriverManager.getConnection` získá připojení. Dále už stačí jen sestavit SQL příkaz (`Statement`) a provést ho. Pokud jde o zjišťovací dotaz (`SELECT`), vrátí metoda `ResultSet`, který obsahuje získaná data. Naopak pokud se jedná např. o vkládací dotaz, vrátí metoda počet úspěšně vložených řádků.

4.4 Server

Serverovou stranu systému můžeme rozdělit na část databázovou, která běží na databázovém systému MySQL, a na část webovou, která byla vytvořena ve skriptovacím programovacím jazyce PHP[11]. Na serveru jsou také uloženy XML popisy testů, které si klienti stahují, aby mohli otestovat svoje zařízení.

[9] Java DataBase Connectivity
[10] General Public License
[11] Hypertext Preprocessor

4.4.1 Databáze

Navržená relační databáze je spuštěna na databázovém systému MySQL ve verzi 5.1.56 a je spravována pomocí nástroje phpMyAdmin ve verzi 3.4.7.1. Databáze obsahuje celkem tři tabulky, které mají jako formát úložiště (storage engine) zvolen InnoDB, a to kvůli tomu, že je nutné spojovat tabulky (vytvářet relace mezi tabulkami) pomocí cizích klíčů, což např. u často používaného typu úložiště MyISAM není podporováno.

Tabulka Device

V tabulce `Device` jsou v šesti sloupcích uloženy informace o zařízení, na němž byly provedeny testy. Zařízení je jednoznačně identifikováno klíčem, vytvořeným jak z údajů přednastavených od výrobce, tak i z údajů, které jsou na výrobci nezávislé (viz 4.3.4).

- `serial` - varchar(50), primární klíč, unikátní identifikátor zařízení
- `manufacturer` - varchar(50), výrobce zařízení
- `model` - varchar(50), název modelu zařízení
- `ip` - varchar(50), IP adresa zařízení
- `user_name` - varchar(50), jméno vlastníka zařízení
- `os_version` - varchar(10), verze operačního systému Android

Tabulka Test

Tabulka `Test` má čtyři sloupce a obsahuje informace o provedených testech. Testy jsou identifikovány pomocí sloupce `id` (primární klíč), který je automaticky inkrementován (funkce `AUTO_INCREMENT`) v momentě, kdy klientská aplikace vloží nový test.

- `id` - int(10) unsigned, primární klíč, identifikátor testu
- `dev_serial` - varchar(50), index, ID zařízení
- `time` - bigint(20), čas provedení testu
- `name` - varchar(30), název testu

Tabulka Result

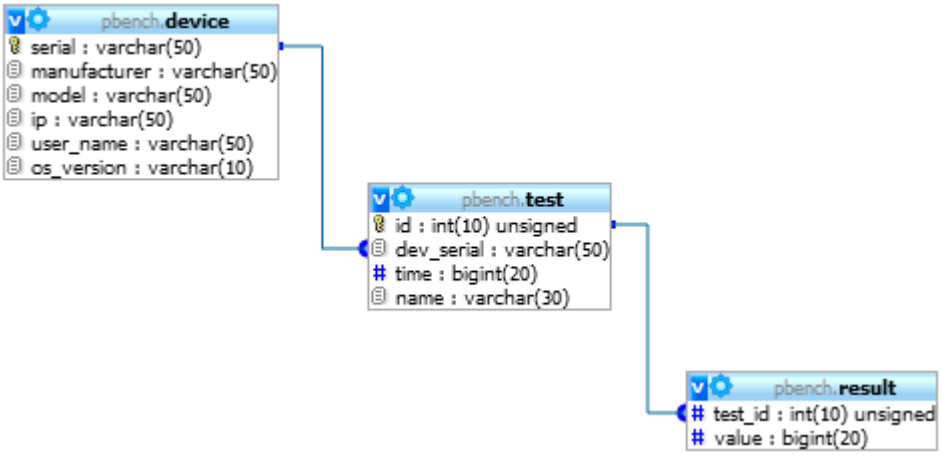
V tabulce `Result`, která má dva sloupce, jsou uloženy naměřené hodnoty jednotlivých testů.

- `test_id` - int(10) unsigned, index, ID testu
- `value` - bigint(20), naměřená hodnota

Relace

Tabulky jsou propojeny dvěma relacemi typu 1:N. Slovně lze tyto vazby vyjádřit tak, že jedno zařízení může mít několik (N) testů, ale jeden test náleží vždy jen jednomu zařízení. Analogicky pro druhou vazbu můžeme říct, že jeden test může mít několik hodnot (podle počtu opakování testu), ale jedna hodnota náleží vždy jen jednomu testu.

Na ERA modelu[12] (viz obr. 4.6), který byl vygenerován pomocí nástroje phpMyAdmin, jsou vidět vazby a struktura tabulek v grafické podobě.



Obrázek 4.6: ERA model databáze

[12]Entity-relationship model

4.4.2 Webová část

Webová část serveru slouží k tomu, aby si mohli uživatelé, kteří na svých zařízeních provedli testy a odeslali je na server, prohlédnout všechny své výsledky přehledně na jednom místě. Dále také mohou např. zjistit, jak ve stejných testech dopadla jiná zařízení.

Celá webová aplikace se skládá ze tří PHP skriptů, které pro své správné fungování potřebují data z databáze. V prvním skriptu si nejprve uživatel vybere svoje zařízení a poté se mu zobrazí všechny zaslané testy. Druhý skript zobrazuje detail konkrétního testu a poslední skript zajišťuje vykreslení grafu.

Výběr zařízení

Na úvodní stránce aplikace je nutné zvolit zařízení, jehož testy mají být zobrazeny. To se provádí pomocí jednoduchého formuláře (`<form>`), který obsahuje tři výběrová pole (`<select>`) a tlačítko pro potvrzení výběru.

Jednotlivé volby výběrových polí (`<option>`) jsou načítány z databáze a mění se podle výběru uživatele. Výběr zařízení probíhá ve třech krocích. Nejprve uživatel zvolí výrobce zařízení (manufacturer). Po tomto výběru je stránka pomocí jednoduchého příkazu javascriptu `window.open` aktualizována s parametrem `dev` nastaveným na hodnotu zvoleného výrobce. Realizace této funkcionality je ukázána na ukázce kódu 4.12.

Kód 4.12: Aktualizace formuláře

```
echo "Manufacturer:
<select name='man'
        onchange=\"window.open('results.php?dev=
            '+this.options[this.selectedIndex].value,
            '_self', false)\">";
```

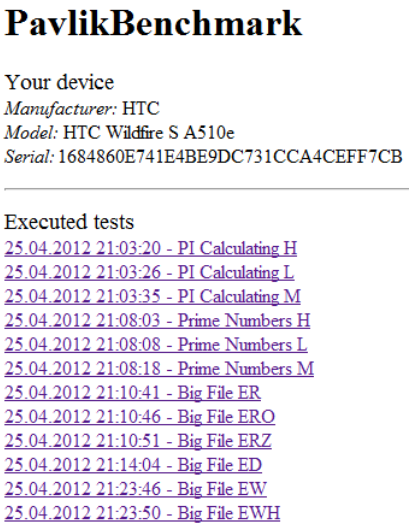
Po aktualizaci stránky se provede výběrový dotaz (`SELECT`), který z databáze získá všechny modely od výrobce, jenž byl předán jako parametr `dev`. Jelikož chceme, aby byly modely ve formuláři vždy jen jednou (v případě, že v databázi je více stejných modelů různých uživatelů), je nutné přidat do dotazu klíčové slovo `DISTINCT` (vrátí pouze záznamy s jedinečnými hodnotami). Data získaná tímto dotazem se následně předají do výběrového pole `model`. Dotaz je pro ilustraci uveden na ukázce kódu 4.13.

Kód 4.13: Dotaz na seznamu modelů

```
$query="SELECT DISTINCT model
                FROM pbench.Device
                WHERE manufacturer='".$_GET['dev']."'";
```

Následuje výběr modelu. Výběrové pole samozřejmě nabídne uživateli pouze modely od výrobce zvoleného v předchozím kroku. Po zvolení modelu se formulář stejným způsobem jako při výběru výrobce aktualizuje a nabídne seznam identifikačních čísel zvoleného modelu.

Jakmile uživatel zvolí identifikační číslo zařízení a potvrdí výběr tlačítkem `Submit`, zobrazí se seznam všech provedených a zároveň zaslaných testů. Testy jsou seřazeny podle času provedení (viz obr. 4.7).



Obrázek 4.7: Seznam testů

Detail testu

Pokud uživatel klikne na jeden z odkazů ze seznamu testů, zobrazí se jeho detail. V horní části stránky je zobrazen minimální, maximální a průměrný čas získaný ze všech naměřených hodnot konkrétního testu.

Jednotlivé testy jsou identifikovány časem svého provedení, takže pro získání všech hodnot stačí provést výběrový dotaz nad tabulkou `Result` s podmínkou, aby byl sloupec `time` roven času provedení testu, který si uživatel

vybral. Poté se vybrané hodnoty projdou v cyklu, ve kterém se zjišťuje minimum, maximum a také součet všech hodnot, který je potřebný pro výpočet průměrné hodnoty (viz Kód 4.14).

Kód 4.14: Získání hodnot testu

```
while ($row = mysql_fetch_array($result, MYSQL_NUM)) {
    $i++;
    $count = $count + $row[0];
    if($i == 1) {
        $min = $row[0];
    }
    if($row[0] < $min) {
        $min = $row[0];
    }
    if($row[0] > $max) {
        $max = $row[0];
    }
}
$avg = $count / $i;
```

Pod zobrazovanými hodnotami se nachází graf, ve kterém jsou vyneseny všechny hodnoty testu a také průměrná hodnota ze všech shodných testů (tzn. že pokud je v databázi uloženo např. pět testů `Medium Files EWT` bude v grafu zobrazena průměrná hodnota ze všech těchto pěti testů).

Vykreslení grafu

Skript na vykreslení grafu byl vytvořen s využitím knihovny GD[13] s funkcemi pro práci s obrázky (viz 4.4.3). Stejně jako v klientské aplikaci se i v této webové aplikaci vytváří sloupcový graf. Pro jeho vytvoření bylo zapotřebí vykreslovat obdélníky (funkce `imagefilledrectangle`), úsečky (funkce `imageline`) a text (funkce `imagestring`). Nakonec se funkcí `imagepng` vytvoří výstup ve formátu PNG[14] obrázku.

Při vykreslování sloupcového grafu se nejvíce využívá funkce na vytvoření vyplněného obdélníku a proto si používání knihovny GD ukážeme právě na této funkci. Nejprve je nutné vytvořit prázdný obrázek. To se provádí funkcí `imagecreate`, která má dva parametry, a to šířku a výšku obrázku.

[13] Graphics Draw
[14] Portable Network Graphics

Tato funkce vrací identifikátor, s jehož pomocí budeme k obrázku přistupovat při kreslení grafu.

Jakmile je nový obrázek vytvořen, můžeme do něj začít kreslit. Na ukázce kódu 4.15 je cyklus, ve kterém se vytvářejí sloupce podle naměřených dat (`$values[]`).

Kód 4.15: Vykreslování sloupců

```
for($i=1;$i <= $total_bars; $i++) {
  $x1 = $margins + ($i - 1) * ($gap+$bar_width) +
       ($bar_width / 2);
  $y1 = $margins + $graph_height -
       floatval($values[$i] * $ratio);

  $x2 = $x1 + $bar_width;
  $y2 = $img_height - $margins;

  imagefilledrectangle($img,$x1,$y1,$x2,$y2,$bar_color);
}
```

Nejdříve je třeba vypočítat počáteční (`$x1, $y1`) a koncové (`$x2, $y2`) souřadnice obdélníku. Souřadnice (0,0) se nachází v levém horním rohu obrázku. Do funkce `imagefilledrectangle` se jako parametr postupně vkládá identifikátor obrázku, počáteční a koncové souřadnice a nakonec barva výplně obdélníku, která se definuje funkcí `imagecolorallocate`.

4.4.3 Použité knihovny

GD Graphics Library

Funkce volně šiřitelné knihovny GD od vývojáře Thomase Boutella byly využity při vytváření grafu v detailu testu. Knihovna byla vytvořena v programovacím jazyce C a používá se často při generování grafů, obrázků nebo např. miniatur dynamicky za běhu programu (on the fly). Mimo jazyka PHP je dostupná třeba pro Perl, Python a další [Bou12].

5 Testování

Funkčnost celého systému byla ověřena při testování dvou mobilních telefonů a také při testování na emulátoru, který byl spuštěn na dvou počítačích. V první části kapitoly budou popsány testy, které byly na zařízeních prováděny. Dále se podíváme na specifikace testovaných zařízení a nakonec budou ukázány některé naměřené hodnoty.

5.1 Testovací sada

Zařízení byla testována sadou, ve které je celkem 66 testů (60 testů souborového systému, 6 výkonnostních testů). Testy souborového systému jsou podle objemu dat rozděleny do tří kategorií. V jedné kategorii se nacházejí testy, které jsou odvozeny od základního vzoru změnou parametrů (úložiště, typ dat, velikost bufferu, atd.). Výkonnostní testy se liší pouze velikostí zátěže.

Některé testy souborového systému nemohly být provedeny, protože na interní paměti zapůjčených zařízení nebyl dostatek volného místa. Volné místo by se dalo získat odinstalováním programů, ale kvůli riziku ztráty důležitých dat nebylo s dříve nainstalovanými aplikacemi manipulováno.

5.1.1 Testy souborového systému

Testování souborového systému probíhalo ve třech fázích. Nejprve se testoval výkon při práci s velkým souborem, poté s několika středně velkými soubory a nakonec větším množstvím menších souborů. Navíc byl ještě pro porovnání vytvořen test, který místo jednoho velkého souboru vytváří dva soubory poloviční velikosti.

V tabulce 5.1 jsou vidět hlavní rozdíly mezi třemi definovanými kategoriemi testů souborového systému. Od těchto základních vzorů jsou odvozeny další testy.

	Big File	**Medium Files**	**Small Files**
Počet souborů	1	5	100
Velikost souborů	50 MB	1500 kB	10 kB
Velikost bloku dat	1 MB	64 kB	1 kB
Počet složek	1	1	30
Hloubka složek	1	1	4
Počet opakování	10	30	10

Tabulka 5.1: Porování testů

Navržená struktura popisu testů (viz 3) umožňuje snadno definovat velké množství různých testů. V následujícím výčtu si ukážeme, které parametry byly obměňovány při vytváření testovací sady.

- **E** - externí úložiště
- **I** - interní úložiště
- **R** - čtení
- **W** - zápis
- **D** - mazání
- **O** - typ dat, jedničky
- **Z** - typ dat, nuly
- **T** - dvojitá velikost bloku dat
- **H** - poloviční velikost bloku dat

Změny parametrů testu se projevují i v jeho názvu (písmena na konci názvu), aby byl uživatel při výběru testů schopen změny rozlišit.

5.1.2 Testy výpočetního výkonu

Výpočetní výkon zařízení je testován pomocí velkého množství základních aritmetických operací (sčítání, odčítání, násobení a dělení). Na rozdíl od testů souborového systému, není u těchto testů tolik parametrů, které lze měnit.

V prvním testu se počítá desetinný rozvoj čísla π až do zadané hranice počtu cifer (základní verze 5000 cifer). V druhém testu se v cyklu od dvou do stanového maxima (střední zátěž 100000) ověřuje, jestli aktuální číslo splňuje podmínky prvočíselnosti.

U obou těchto testů jsou přednastaveny tři stupně zátěže. Zátěž se mění změnou parametrů testu (např. zvýšením počtu cifer pro výpočet desetinného rozvoje π). Stejně jako u testů souborového systému jsou tyto změny popsány písmeny.

- **H** - vysoká zátěž
- **M** - střední zátěž
- **L** - nízká zátěž

5.2 Testovaná zařízení

Testování bylo prováděno na dvou mobilních telefonech srovnatelného výkonu a na dvou počítačích, jejichž výkon se lišil více. V tabulce 5.2 jsou nejdůležitější parametry testovaných mobilních telefonů.

	HTC Wildfire S	**Samsung Galaxy Mini**
Procesor	600 MHz ARM 11	600 MHz ARM 11
Int. paměť - RAM	512 MB	384MB
Int. paměť - data	418 MB	160 MB
Ext. paměť	2 GB (až 32 GB)	2 GB (až 32 GB)
GPU	Adreno 200	Adreno 200
Verze OS Android	2.3.5	2.3.4

Tabulka 5.2: Specifikace testovaných telefonů

Vývoj klientské aplikace probíhal výhradně na emulátoru, jenž byl pro porovnání s fyzickým zařízením využit také pro testování. Na obou počítačích byla vytvořena dvě virtuální zařízení s různými verzemi systému Android (2.3.3 a 4.0.3). Parametry virtuálních zařízení byly nastaveny na obou počítačích stejně.

Rozdíly ve výkonu počítačů, na kterých byla zařízení emulována, jsou naznačeny v tabulce 5.3.

	ASUS X59SL	**Lenovo Y55OP**
Procesor	Intel C2D @ 2 GHz	Intel Core i3 @ 2.27 GHz
Operační paměť	2 GB	4 GB
Grafická karta	ATI Radeon HD3470	NVIDIA GF 240M GT
Pevný disk	250 GB (5400 RPM)	500 GB (5400 RPM)
Operační systém	Windows 7 32b.	Windows 7 64b.

Tabulka 5.3: Specifikace počítačů na testování

5.3 Naměřené hodnoty

Během testování bylo na šesti zařízeních (4 virtuální, 2 fyzická) provedeno celkem 337 testů, které obsahují 5930 naměřených hodnot. V této sekci se podíváme na některé zajímavé nebo neočekávané výsledky.

5.3.1 Obecné poznatky

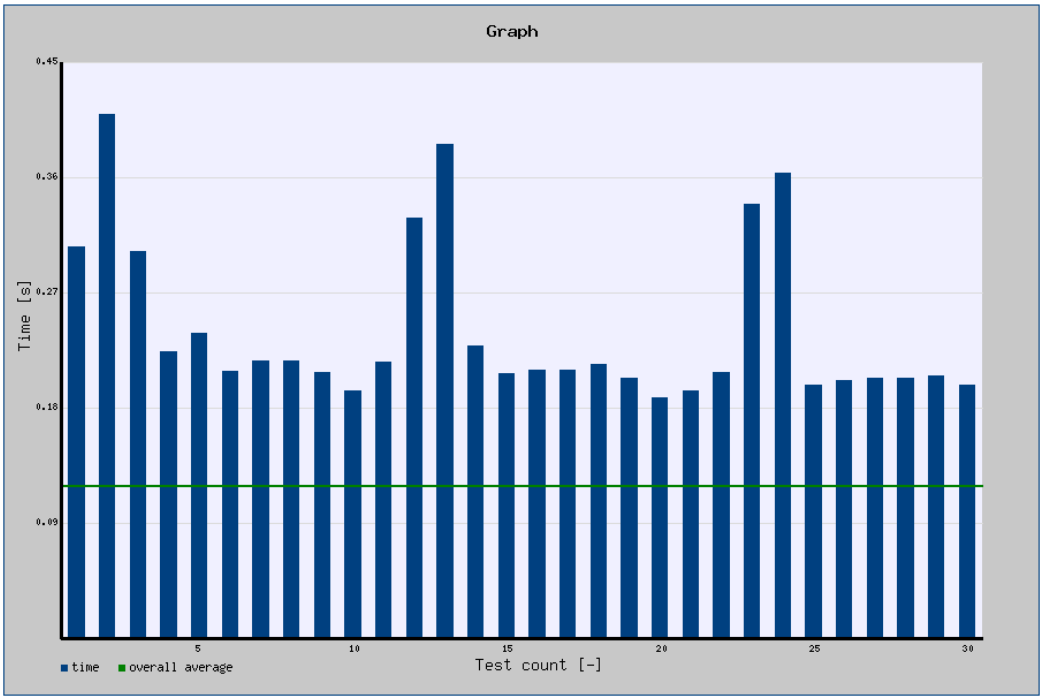
Při vyhodnocování provedených testů se objevilo několik výsledků, které se pravidelně opakovaly. Některé z nich si nyní uvedeme.

V drtivé většině testů na emulátoru mělo lepší výsledky zařízení se starší verzí systému (Android 2.3.3) oproti novější verzi (Android 4.0.3). To může být způsobeno např. vyššími nároky grafického uživatelského rozhraní na výkon zařízení (obě virtuální zařízení měla stejné parametry). Testovaná fyzická zařízení měla téměř stejnou verzi operačního systému (2.3.4 a 2.3.5), takže nemohl být zkoumán jeho vliv na výsledky testů.

Další zajímavé hodnoty přinesl test, při kterém se počítá desetinný rozvoj čísla π. Jak je vidět na obr. 5.1, jednotlivé hodnoty jsou až na tři výkyvy poměrně ustálené. K vysvětlení těchto výkyvů posloužil nástroj Logcat, pomocí kterého lze nahlédnout do systémového logu.

Při samotném testování se mimo jiné zvyšoval nápor na operační paměť a právě z výpisu systémového logu pak bylo vidět, že byl v průběhu testu něko-

likrát spuštěn automatický správce paměti (garbage collector), který ovlivnil výsledky testování.

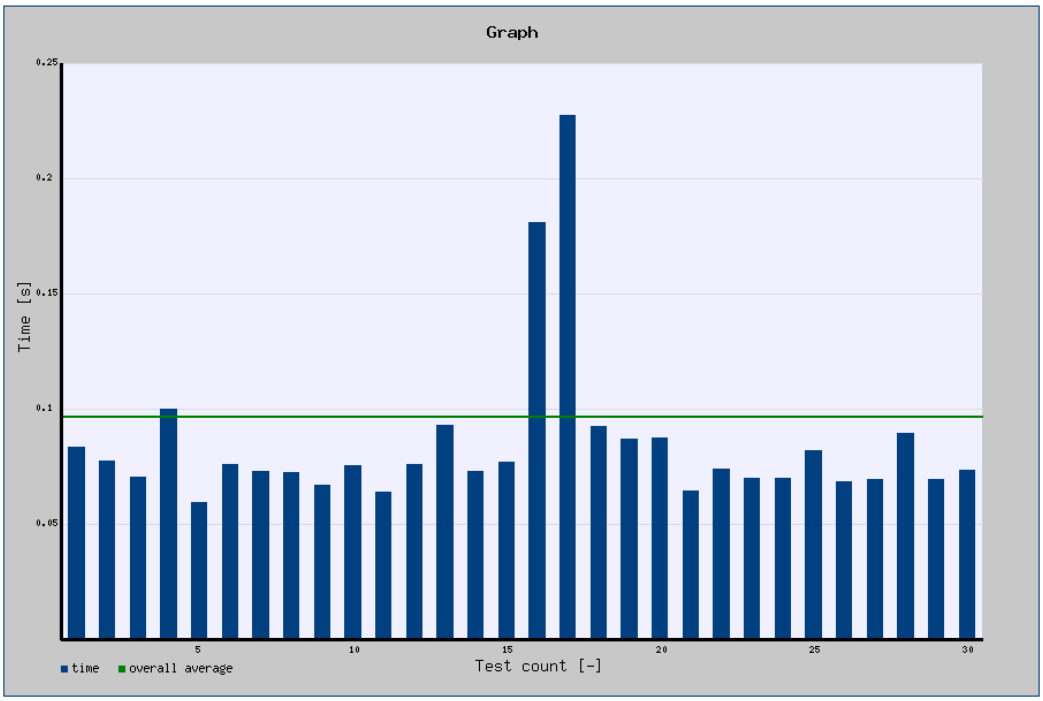


Obrázek 5.1: Výkyvy v testu výpočetního výkonu

Podobné hodnoty byly opakovaně změřeny při provádění stejného testu na virtuálních i fyzických zařízeních.

5.3.2 Anomální hodnoty

V průběhu testování se občas vyskytly neočekávané a nepravidelné hodnoty, které příliš nekorespondovaly s ostatními hodnotami v testu. Často (ovšem ne při každém spuštění) byla hned první hodnota testu vyšší než ostatní hodnoty. Na obr. 5.2 je vidět příliš velká hodnota uprostřed testu čtení.



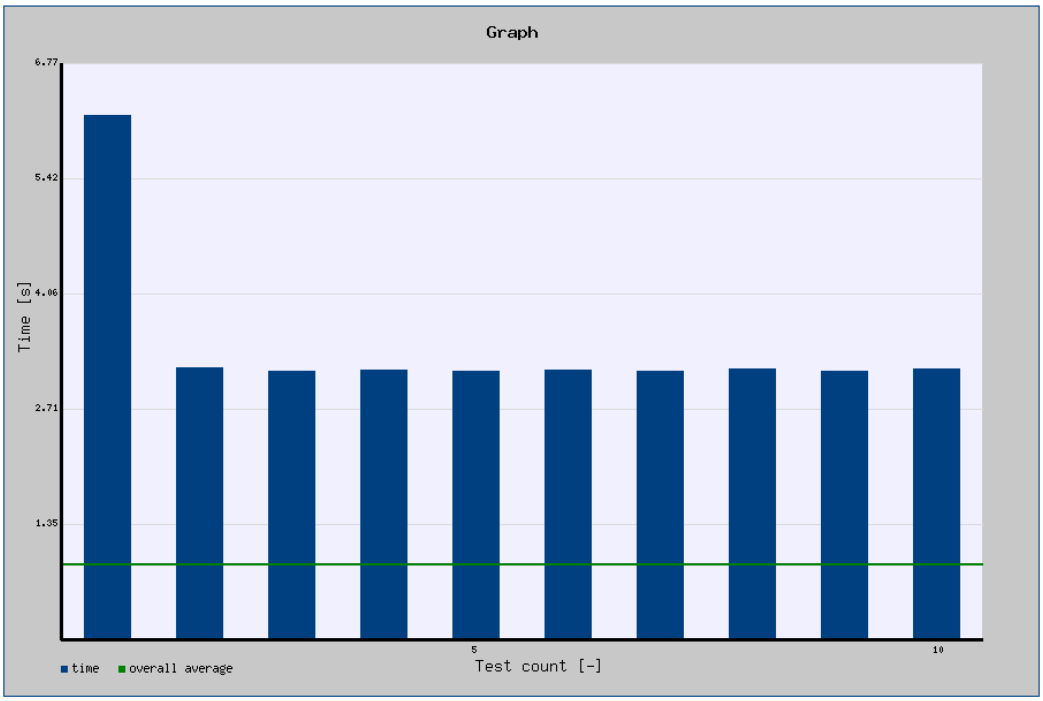
Obrázek 5.2: Neočekávané hodnoty v testu čtení

V některých případech dosahovaly anomální hodnoty až čtyřnásobku průměrné hodnoty celého testu, ale většinou se pohybovaly okolo dvojnásobku průměru. Zcela výjimečně se v testu objevila také viditelně nízká hodnota, ale rozdíl od ostatních hodnot nebyl tak velký jako v případě nadstandardně vysokých hodnot.

5.3.3 Fyzické zařízení vs. emulátor

Porovnání výsledků reálných zařízení s virtuálními přineslo také některé zajímavé poznatky. Na úvod je třeba zmínit, že obě fyzická zařízení generovala občas rozdílné výsledky, přestože jsou jejich specifikace velmi podobné (viz Tabulka 5.2), ale i tak nabídla dobré možnosti srovnání s emulátorem.

Nejprve se podíváme na testy souborového systému. Při operacích s velkým souborem dosahovala virtuální zařízení lepších výsledků. Telefon od společnosti HTC měl výsledky srovnatelné s emulátorem, ovšem Samsung Galaxy Mini výrazně zaostával. V testu zápisu to bylo přibližně o 60 % a v testu čtení dokonce o více než 500 % oproti telefonu HTC. Na obr. 5.3 jsou vidět hodnoty Samsungu v testu čtení velkého souboru. Zelená čára značí průměrnou hodnotu všech ostatních zařízení dohromady.



Obrázek 5.3: Čtení velkého souboru

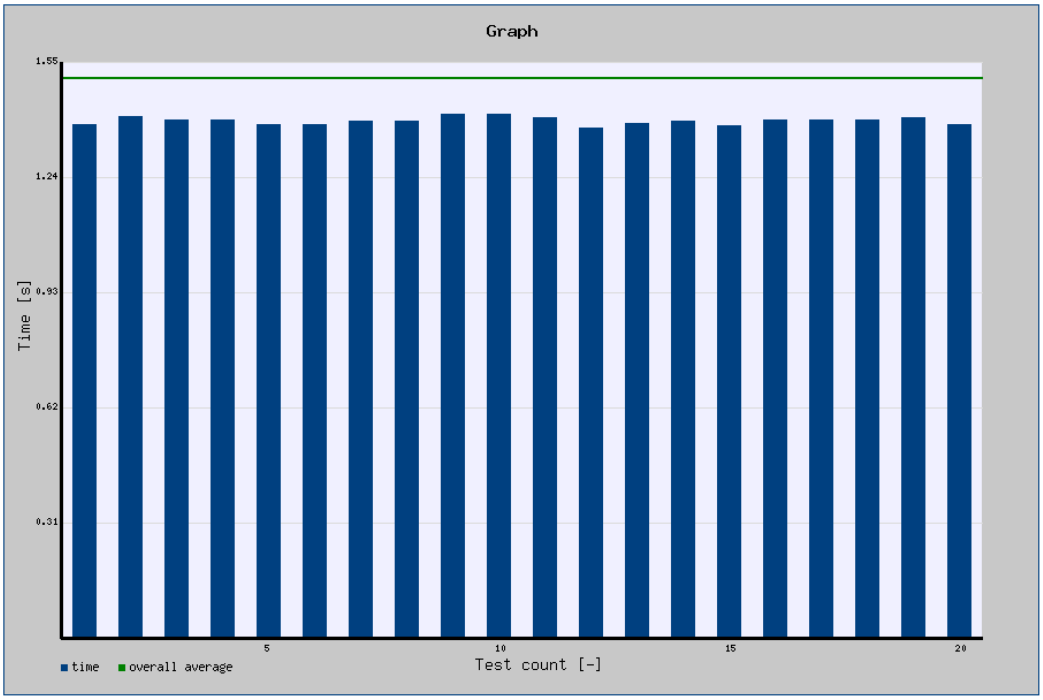
Situace se naprosto změnila v testech se středními a malými soubory. Nejlepších výsledků celkově dosahoval telefon Samsung Galaxy Mini. Telefon HTC Wildfire S měl velmi špatné výsledky především při zápisu a nejvíce markantní byly rozdíly v zápisu na interní paměť zařízení (viz Ta-

bulka 5.4). Virtuální zařízení[1] generovala poměrně stabilní výsledky, které však byly v některých případech o poznání horší než výsledky Samsungu.

	HTC	**Samsung**	**AVD 1**	**AVD 2**	**AVD 3**	**AVD 4**
Medium	3,16 s	0,07 s	0,55 s	0,55 s	0,72 s	0,56 s
Small	1,06 s	0,12 s	0,23 s	0,29 s	0,18 s	0,29 s

Tabulka 5.4: Zápis na interní paměť

V testech výpočetního výkonu byly výsledky lepší u fyzických zařízení, a to hlavně proto, že virtuální zařízení měla nastavenou menší operační paměť. Můžeme tedy říci, že emulátor v tomto ohledu simuluje výkon zařízení poměrně věrně. Na obr. 5.4 je příklad testu, ve kterém se počítají prvočísla. Je vidět, že hodnoty jsou velmi podobné (bez výkyvů).



Obrázek 5.4: Test výpočetní výkonu (prvočísla)

[1]AVD - Android Virtual Device

6 Závěr

Tato bakalářská práce byla zaměřena na benchmarking mobilních zařízení s operačním systémem Android.

Nejprve jsem se věnoval benchmarkingu obecně. Čtenář byl stručně seznámen s různými typy benchmarků, jednotlivými fázemi procesu benchmarkingu a nakonec byl uveden přehled nejpoužívanějších benchmarků mobilních zařízení.

V dalších kapitolách jsem se zaměřil na vytvořenou aplikaci. Nejdříve byla popsána struktura testů, které byly prováděny. Poté jsem se věnoval popisu systému jako celku a zároveň také popisu samotné klientské a serverové části aplikace. V předposlední kapitole bylo na několika příkladech ukázáno, jak probíhalo testování na fyzických i virtuálních zařízeních a jaké přineslo výsledky.

Domnívám se, že vytvořená aplikace splňuje požadavky, které byly stanoveny. Je velmi jednoduchá na používání a zároveň v ní lze provádět velké množství testů. To je umožněno díky tomu, že struktura popisu testů je dostatečně univerzální, což bylo jedním z hlavních bodů zadání.

Na druhou stranu si myslím, že existuje potenciál pro další rozvoj aplikace. Například na straně serveru se do budoucna nabízí přidání dalších funkcí pro porovnávání výsledků provedených testů.

Seznam obrázků

Seznam tabulek

Přehled použitých zkratek a značení

API	Application Programming Interface. Rozhraní pro vývoj aplikací.
Bluetooth	Standard pro bezdrátovou komunikaci.
CISC	Complex Instruction Set Computer. Označení procesorů s komplexní instrukční sadou.
DOM	Document Object Model. API pro přístup nebo změnu XML dokumentu.
EEMBC	The Embedded Microprocessor Benchmark Consortium. Nezisková organizace, která vyvíjí benchmarky pro vestavěné systémy.
ERA model	Entity-relationship model. Znázornění dat pomocí datového modelování.
GPL	GNU General Public License. Licence pro svobodný software.
HTML	HyperText Markup Language. Jazyk pro tvorbu webových stránek.
IMEI	International Mobile Equipment Identity. Unikátní identifikátor mobilního telefonu.

JDBC	Java Database Connectivity. API pro přístup k databázi pomocí jazyka Java.
KIVFS	Souborový systém, vyvíjený na katedře informatiky a výpočetní techniky ZČU.
Logcat	Nástroj pro přístup k systémovému logu Androidu.
MAC	Media Access Control. Unikátní identifikátor síťového zařízení.
MD5	Message-Digest Algorithm. Hašovací algoritmus, který vytváří výstup stejné délky.
MFLOPS	Million Floating-point Operations per Second. Počet operací s plovoucí desetinnou řádkou za sekundu.
MIPS	Million Instruction Per Second. Jednotka, která udává počet instrukcí za sekundu.
NFS	Network File System. Protokol pro přístup k souborům přes počítačovou síť.
PHP	Hypertext Preprocessor. Skriptovací programovací jazyk.
RISC	Reduced Instruction Set Computer. Označení procesorů s redukovanou instrukční sadou.
SAX	Simple API for XML. Nástroj pro přístup ke XML dokumentům.
SFS	System File Server.
SPEC	Standard Performance Evaluation Corporation. Organizace, která vyvíjí benchmarky pro počítače.

SQL	Structured Query Language. Dotazovací jazyk pro práci s relačními databázemi.
TPC	Transaction Processing Performance Council. Organizace, která definuje benchmarky.
Wi-fi	Wireless LAN. Standard pro bezdrátovou komunikaci.
XML	Extensible Markup Language. Značkovací jazyk určený především pro výměnu dat mezi aplikacemi.

Literatura

[And12] ANDROBENCH. *Androbench* [online]. 2012. [cit. 23.4.2012]. Dostupné z: `http://www.androbench.org/wiki/AndroBench`.

[Ant12] ANTUTULABS. *AnTuTu Benchmark* [online]. 2012. [cit. 23.4.2012]. Dostupné z: `http://www.antutulabs.com/AnTuTu-Benchmark`.

[Aur10] AURORASOFTWORKS. *Quadrant Standard Edition* [online]. 2010. [cit. 23.4.2012]. Dostupné z: `http://www.aurorasoftworks.com`.

[Bou12] BOUTELL, T. *GD Graphics Library* [online]. 2012. [cit. 3.5.2012]. Dostupné z: `http://www.boutell.com/gd/`.

[Fei11] FEITELSON, D.G. *Workload Modeling for Computer Systems Performance Evaluation.* The Hebrew University of Jerusalem, 2011. Dostupné z: `http://www.cs.huji.ac.il/~feit/wlmod/`.

[Gre10] GREENECOMPUTING. *Linpack for Android* [online]. 2010. [cit. 23.4.2012]. Dostupné z: `http://www.greenecomputing.com/apps/linpack/`.

[Hun10] HUNTINGTON, T. *Gauge Mathematical Tool* [online]. 2010. [cit. 23.4.2012]. Dostupné z: `http://thomashuntington.com/iOS/Gauge_Mathematical_Tool.html`.

[Jai91] JAIN, R. *Art of Computer Systems Performance Analysis Techniques For Experimental Design Measurements Simulation And Modeling.* Wiley Computer Publishing, 1991. ISBN 0471503363.

[Kan11] KANJILAL, C. *Benchmark your Android Phone with Quadrant* [online]. 2011. [cit. 23.4.2012]. Dostupné z: `http://www.lostintechnology.com/mobile/benchmark-your-android-phone-with-quadrant/`.

[Kil12] KILIÁN, K. *Programy pro měření výkonu („benchmarky“)* [online]. 2012. [cit. 23.4.2012]. Dostupné z: http://www.svetandroida.cz/svetandroida-doporucuje-programy-pro-mereni-vykonu-benchmarky-201204.

[Lai03] LAI, R. *J2EE Platform Web Services* [online]. Prentice Hall, 2003. [cit. 26.4.2012]. ISBN 0131014021 Dostupné z: http://flylib.com/books/en/1.586.1.26/1/

[Lay09] LAYTON, J. B. *Lies, Damn Lies and File System Benchmarks* [online]. 2009. [cit. 10.4.2012]. Dostupné z: http://www.linux-mag.com/id/7464/.

[Mot11] MOTISAN, R. *Android Unique Device ID* [online]. February 2011. [cit. 20.4.2012]. Dostupné z: http://www.pocketmagic.net/?p=1662.

[Pri12] PRIMATELABS. *GeekBench* [online]. 2012. [cit. 23.4.2012]. Dostupné z: http://www.primatelabs.ca/geekbench/.

[Sma97] SMALL, C. et al. *Does Systems Research Measure Up?* Harvard University, 1997. [cit. 8.4.2012]. Dostupné z: http://www.eecs.harvard.edu/cs261/background/small.pdf.

[Tra08] TRAEGER, A. et al. *A nine year study of file system and storage benchmarking* [online]. ACM Transactions on Storage (TOS), v.4 n.2 May 2008. [cit. 8.4.2012]. Dostupné z: http://www.fsl.cs.sunysb.edu/docs/fsbench/fsbench.pdf.

[Var11] VARGA, R. *WP Bench* [online]. 2011. [cit. 23.4.2012]. Dostupné z: http://www.windowsphone.com/cs-CZ/apps/e447e949-01c0-43f1-8b65-76d752d7d305.

[Wei02] WEICKER, R. P. *Performance Evaluation of Complex Systems: Techniques and Tools : Performance 2002 Tutorial Lectures,* chapter Benchmarking. Lecture Notes in Computer Science. Springer, 2002. Dostupné z: http://www.google.cz/books?id=VEyvL-vscCgC. ISBN 9783540442523.

[Wol11] WOLFE, B. M. *First iOS 5 Benchmarks Suggest Most Will Be Happy With New OS* [online]. 2011. [cit. 23.4.2012]. Dostupné z: http://appadvice.com/appnn/2011/10/first-ios-5-benchmarks-suggest-most-will-be-happy-with-new-os.

A Přílohy

A.1 Instalační příručka

A.1.1 Klientská aplikace

Instalační balíček klientské části aplikace je dostupný na internetové adrese `http://pbench.heliohost.org/app/BP.apk`. Po stažení a spuštění balíčku se uživateli zobrazí dialog se seznamem všech povolení, která aplikace vyžaduje. Pokud uživatel souhlasí se všemi požadavky, stačí stisknout tlačítko `Install` a aplikace se nainstaluje na interní paměť zařízení.

Pro odinstalaci se doporučuje použít správce aplikací v nastavení daného zařízení.

A.1.2 Serverová aplikace

Pro správné fungování serverové části aplikace by měl server splňovat následující požadavky.

- PHP 5
- MySQL verze 5.1 a vyšší
- nainstalovaná knihovna GD (viz 4.4.3)

A.2 Uživatelská příručka

A.2.1 Klientská aplikace

První obrazovka aplikace obsahuje tři tlačítka. Po stisku tlačítka `Show Available Tests` se zobrazí seznam dostupných testů. Pokud ovšem nebude nalezeno připojení k internetu, objeví se krátká chybová hláška. Další dvě tlačítka slouží pro zobrazení identifikačního čísla zařízení a ukončení aplikace.

V případě, že se podařilo získat seznam testů, stačí pomocí zaškrtávacích tlačítek vybrat požadované testy. Stisknutím tlačítka `Download Selected Test` se vybrané testy stáhnou do zařízení, zpracují se a uživateli se zobrazí jejich hlavní parametry.

Pokud uživatel bude chtít změnit výběr, použije tlačítko `Back`. V opačném případě stisknutím tlačítka `Run Tests` spustí testování. Testování se dá kdykoliv zrušit pomocí tlačítka `Cancel`. Jakmile je testování ukončeno, uživatel je dotázán, zda chce naměřené výsledky odeslat na server.

Na poslední obrazovce aplikace uživatel může procházet výsledky všech provedených testů. Mezi jednotlivými testy se přechází pomocí tlačítek `Previous` a `Next`. Tlačítko `Home` slouží pro návrat na úvodní obrazovku aplikace.

Kdykoliv při běhu aplikace se dá stisknutím tlačítka `Menu` na daném zařízení zobrazit nastavení, ve kterém si uživatel může zadat jméno, jenž se bude odesílat spolu s informacemi o zařízení na server.

A.2.2 Serverová aplikace

Výsledky všech zaslaných testů je možné zobrazit pomocí webové aplikace, která se nachází na adrese `http://pbench.heliohost.org/results.php`. Tato stránka obsahuje jednoduchý formulář, pomocí kterého si uživatel vybere svoje zařízení nebo jiné dostupné zařízení v případě, že bude chtít porovnat své výsledky s ostatními.

Uživatel postupně ze seznamu možností vybere výrobce, model a identifikační číslo zařízení. Toto číslo je generováno klientskou částí aplikace, ze které je jednoduše dostupné (viz A.2.1).

Po odeslání formuláře stisknutím tlačítka `Submit` se zobrazí seznam provedených testů seřazený podle času jejich provedení. Pro zobrazení detailu testu stačí pouze kliknout na příslušný odkaz.

A.3 Test zápisu velkého souboru

```
<test category="filesystem" name="Big␣File␣EWZ" repeat="10">
    <storage>SD card</storage>
    <operation>write</operation>
    <blocksize unit="MB">1</blocksize>
    <directories description="exact">
        <directory>
            <name>dir1</name>
        </directory>
    </directories>
    <files description="exact" type="zeros">
        <file>
            <name>file1</name>
            <size unit="MB" description="exact">50</size>
            <directory>dir1/</directory>
        </file>
    </files>
</test>
```

A.4 Test čtení malých souborů

```
<test category="filesystem" name="Small␣Files␣IR" repeat="10">
    <storage>memory</storage>
    <operation>read</operation>
    <blocksize unit="kB">1</blocksize>
    <directories description="non-exact">
        <depth>4</depth>
        <count>30</count>
    </directories>
    <files description="non-exact" type="random">
        <size unit="kB" description="exact">10</size>
        <count>100</count>
    </files>
</test>
```