Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Porovnání grafických knihoven na vestavěném zařízení

Jan Šíma

Bakalářská práce
2009

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Katedra informačních technologií
Akademický rok: **2008/2009**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Jan ŠÍMA**

Studijní program: **B2646 Informační technologie**

Studijní obor: **Informační technologie**

Název tématu: **Porovnání grafických knihoven na vestavěném zařízení**

Zásady pro vypracování:

Teoretická část 1. Vytvořte přehled dostupných grafických knihoven pro GUI na platformě X Window. 2. Shrňte vlastnosti a možnost použití na vestavěném systému. 3. Popište kompilaci programů pro jinou platformu (cross-compiling), zaměřte se na kompilaci pro procesor ARM na platformě IA-32. Praktická část 1. Vyberte vhodné kandidáty pro použití na platformě ARM: Procesor AT91SAM9263, 200 MHz. Barevné displeje rozlišení QVGA nebo VGA. OS Linux 2.6.22, X?Server Xfbdev. (Nyní je používána knihovna GTK+ 2.12.6, která je implementována jako klient-server, což se ukázalo jako špatná volba: není možné přistupovat přímo na objekty knihovny, není možné měnit objekty za běhu, pokud pro to není implementována funkce, potřeba implementovat stejnou funkci 2 ? na straně serveru i klienta). Do výběru je možné zahrnout i komerční produkty. Knihovna by měla umět pracovat pod X?Serverem ? přímý přístup na frame-buffer není požadován. Přínosná je existence návrháře GUI. 2. Proveďte testy na vybraných knihovnách a knihovny porovnejte: velikost knihovny, snadnost použití (existence návrháře), náročnost na HW (paměť, zatížení procesoru, rychlost překreslování celých obrazovek za vteřinu), dostupná podpora. Většina testů by měla jít provádět na PC, rychlost překreslování obrazovky bude třeba zkoušet na zařízení. 3. Pro každou knihovnu by měla být napsána jednoduchá aplikace, která bude obsahovat minimálně dvě okna/tabulky, mezi kterými půjde přepínat stiskem kláves (pro subjektivní vyhodnocení rychlosti).

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

*** Kolektiv autorů: LINUX - dokumentační projekt. Computer Press 1998 (3. aktualizované vydání), ISBN 80-7226-761-2. * Stones, Richard, Matthew, Neil: Linux - Začínáme programovat. Praha: Computer Press, 2000. ISBN 80-7226-307-2. * Stones, Richard, Matthew, Neil: Linux - Programujeme profesionálně. Praha: Computer Press, 2001. ISBN 80-7226-532-6. * Reimer, Klaus: ARM cross-compiling howto [online]. 2005-08-28 [cit. 2009-01-08]. URL: http://www.ailis.de/ k/archives/19-ARM-cross-compiling-howto.html**

Vedoucí bakalářské práce: **Mgr. Tomáš Hudec**
Katedra informačních technologií

Datum zadání bakalářské práce: **15. ledna 2009**

Termín odevzdání bakalářské práce: **15. května 2009**

Univerzita Pardubice
Česká republika

L.S.

doc. Ing. Simeon Karamazov, Dr.
děkan

Ing. Lukáš Čegan
vedoucí katedry

V Pardubicích dne 31. března 2009

Prohlášení autora

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 29. 4. 2009

Jan Šíma

Poděkování

Na této stránce, která je učená pro poděkování, bych chtěl poděkovat hlavně Mgr. Tomáši Hudcovi, který mi velmi pomohl při zpracování bakalářské práce. Dále bych chtěl poděkovat Jarmile Šímové při pomoci s opravou českého pravopisu. Dále Ing. Janu Kroulíkovi, „embedded Linux developerovi“, ze společnosti MIKRO-ELEKTRONIKA spol. s r. o.. Všem zmíněným děkuji za pomoc při mé práci.

Anotace a klíčová slova

Anotace

Práce je věnována paměťové a časové náročnosti grafických knihoven pro X Window se zaměřením na multiplatformní knihovny. Věnuje se knihovnám Qt, GTK+, wxWidgets, FLTK, Fast Toolkit. Zabývá se jejich porovnáním mezi sebou a následnému testování jednotlivých knihoven.

Klíčová slova

GTK+, Qt, FLTK, wxWidgets, Fast Toolkit, grafické knihovny, paměť, čas, Unix, RAM

TITLE

Comparison of graphic libraries for embedded device

ANNOTATION

This work is dedicated to measure the memory and time performance of graphics libraries for X Window with a focus on multi-platform library. It focuses on the Qt, GTK+, wxWidgets, FLTK, Fast Toolkit libraries. It deals with the comparison among them and the subsequent testing of individual libraries.

KEYWORDS

GTK+, Qt, FLTK, wxWidgets, Fast Toolkit, graphic libraries, memory, time, Unix, RAM

Obsah

Úvod

Tato práce přibližuje řadu grafických knihoven a jejich historii. Specifikuje jejich vlastnosti, klady a zápory. Práce byla zadána soukromým subjektem s požadavkem otestovat grafické knihovny. Práce obsahuje i specifikace ne tak běžného charakteru.

Dále jsou popsány aplikace, programy a prostředí. Jsou zde přiblíženy jejich nastavení a také důvody použití právě zmiňovaných aplikací. Jsou zde testovány jednotlivé knihovny z důvodů zjištění počtu svých prvků (tlačítka, popisky) ve fyzické paměti a testování rychlosti zobrazení jednotlivých prvků.

Hlavním kritériem testování je použití na platformě ARM s procesorem AT91SAM9263 200MHz, rozlišení displeje QVGA nebo VGA, OS Linux, X Server Xfbdev. Nyní se na této platformě používá GTK+ 2.12.6. . . Všechny testy v této práci byly ostestovány na platformě i386 a následně testováno na platformě ARM. V závěru je srovnání všech prováděných testů a jejich vlastní vyhodnocení.

1 Přehled grafických knihoven pro GUI na platformě X Window

Grafických knihoven pro X Window je nespočet, od velmi propracovaných až po knihovny, které jsou teprve v počátcích vývoje. V práci jsou obsaženy knihovny, které jsou už nějakým způsobem zažité a mají svoji historii a kvality. Mezi vybrané knihovny patří GTK+, Qt, Fox Toolkit, FLTK(Fast Light Toolkit) a wxWindows. U každé knihovny je naznčena její historie a vlastnosti.

1.1 GTK+

Historie – GTK+ byl původně vytvořen v roce 1997 členy eXperimental Computing Facility (XCF) Spencerem Kimballem, Peterem Mattisem a Joshem MacDonaldem. První stabilní verze vyšla v dubnu v roce 1998. Další velký zlom byl v březnu 2002, kdy vyšla verze 2.0. Tato verze s sebou přinesla zdokonalení vykreslování textů pomocí Pango. A další z novinek je dokončení přechodu na Unicode, tedy UTF-8. Nevýhodou přechodu je nekompatibilita s verzí 1.0.

Podporované platformy GTK+ jsou X Window – tato platforma je i zároveň hlavní prioritou tohoto projektu –, dalšími platformami, na které se GTK+ zaměřuje, jsou Microsoft Windows (Windows 2000 a vyšší), DirectFB a Quartz (Max OS X v.10.4 a vyšší).

Programovací jazyk v této knihovně je použit jazyk „C“, ale je portován do mnoha dalších jazyků – viz *Tabulka 1*. **Licence** je už od počátku vedena pod GNU LGPL 2.1 (GNU Lesser General Public License). **Rad Application Development –** GTK+ nemá svoje prostředí či oficiální implementaci do známých IDE, takže se musí použít prostředí třetích stran – Glade, Anjuta, Eclipse, ale i další, pokud bychom chtěli použít jiný portovaný jazyk.

Dokumentace je dobře zpracována, má vlastní styl. A komunita okolo GTK+ je dost velká. Ale spíše anglická fóra a české návody se velmi těžko hledají. Pro mne velmi těžkopádná dokumentace, co jsem potřeboval, jsem buď vůbec nenašel a nebo to bylo velmi náročné. Pro mne dosti neintuitivní.

Tyto dokumentace jsou ve většině případů velmi těžkopadné a dosti neintuitivní.

Přednosti ohledně lokalizace jsou na velmi dobré úrovni. GTK obsahuje hodně vstupních metod pro využití mezinárodních jazyků. Podporuje i psaní zprava doleva pro arabské jazyky. A řetězce Unicode jsou samozřejmostí. Gtkmm podporuje namespace a je velmi čistě napsané.

Nedostatkem je import na platformu win32 – podle komunity okolo GTK jsou s ním stále problémy, i když se je snaží programátoři napravit.

Tabulka 1: GTK+ podporované jazyky (7)

Klíč	
G	Oficiální podpora GNOME
A	Podporováno
C	Částečná podpora
N	Nepodporováno

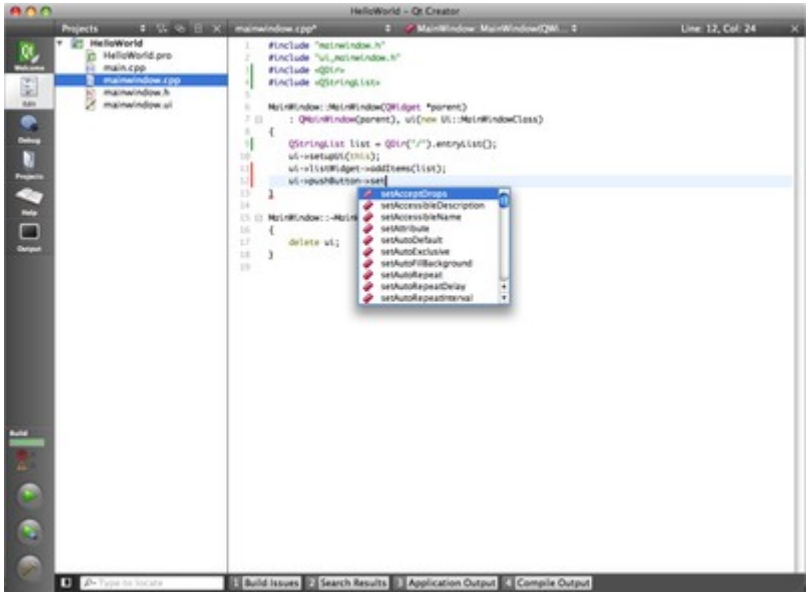
Jazyk	**Projekt**	**2.6**	**2.8**	**2.10**	**2.12**	
C++	gtkmm	A	A	A	A	G
C#	Gtk#	A	A	A	A	G
Java	java-gnome	A	A	A	A	G
Python	PyGTK	A	A	A	A	G
Perl	gtk2-perl	A	A	A	A	G
R	RGtk2	A	A	A	A	
Lua	lua-gtk	A	A	A	A	
Guile	guile-gnome	A	A	A	C	
Ruby	Ruby-GNOME2	A	A	A	C	
PHP	PHP-GTK	A	A	C	C	
Ada	GtkAda	A	A	C	N	
OCaml	LablGTK	A	A	C	N	
Haskell	Gtk2Hs	C	C	N	N	
S-Lang	SLgtk	C	C	C	N	
D	gtkd	N	N	N	A	

1.2 Qt

Historie – Haavard Nord a Eirik Chambe-Eng zahájily vývoj „Qt“ v roce 1991, tři roky předtím, než byla založena firma Quasar Technologies, poté změnila název na Troll Tech, a nyní vystupuje pod jménem Trolltech. Tato knihovna byla nazvána Qt, protože písmeno „Q“ se líbilo Haavard-ovy v editoru Emacs a písmeno „t“ bylo inspirováno z X toolkit („Xt“).V roce 1998 začaly obavy, že KDE usiluje o to, být jedním z předních desktopových prostředí pro Linux. Protože KDE bylo založeno na Qt, tak mnoho lidí z hnutí svobodného software se obávalo, že zásadní díl jejich hlavního operačního systému je proprietární. Toto vyústilo hlavně k vytvoření Gnome, který byl založen na GTK+.

Podporované platformy vývojových aplikací a uživatelské prostředí je možné nasadit na Windows, Mac, Linux/X11, embedded Linux, Windows CE a S60 (brzy ve vývoji) bez přepisování kódu. **Programovací jazyk** je pro tuto knihovnu C++ se speciálním pre-compilerem. Pro tento jazyk je i plná podpora. Další zajímavá vlastnost je portování na jazyk JAVA přes Qt Jambi. Tento jazyk nepodporuje v této míře žádná ze zmiňovaných knihoven. Jazyk Java je dosti populární. Další podporované jazyky jsou C#/.NET Languages (Mono), Python, Ada, Pascal, Perl, PHP, Ruby.

Qt **licenční** model se skládá ze tří licencí, pro přehlednost je znázorňuje *tabulka 2*. **Rapid Application Development –** Qt má svoje vlastní IDE „Qt Creator“ (*obrázek 1*). Toto prostředí je multiplatformní jako knihovna samotná. IDE je dosti



Obrázek 1: Qt Creator (2)

mladé, ale obsahuje všechny náležitosti novodobého prostředí, např. našeptávání, návrhář uživatelského prostředí v podobě integrovaného „Qt Designer“. Je to velmi příjemné prostředí a hlavní výhodou je, že podporuje všechny funkcionality Qt. Častým použitím je integrování do prostředí Eclipse či MS Visual Studio, z důvodů oblíbeného používání těchto IDE.

Jak již bylo zmíněno o podpoře integrace přímo do těchto prostředí, musíme také velmi pochválit i tyto integrace, jsou velmi příjemné a jiné zmiňované knihovny toto nemají.

Dokumentace v Qt je na perfektní úrovni, ze zmiňovaných je nejvíce přehledná a velmi podobná s dokumentací C++. Komunita je také početná, ale na českém internetu malinko zaostává za světem.

Přednosti této knihovny jsou jak už zmíněné vlastní IDE, tak podporovaná přímá integrace do jiných prostředí. Velmi příjemné je dosti předpřipravených widgetů. Podpora Unicode je už dnes u nejpoužívanějších knihoven samozřejmostí. A určitě nesmíme zapomenout, na velmi přehlednou dokumentaci.

Nedostatkem je špatná integrace češtiny v „Qstring“. Tato vlastnost je velmi nepříjemná, dá se ovšem odstranit, ale není to úplně jednoduché. Qt nevyužívá jmenný prostor, ale používá prefix „Q“. Tedy všechny třídy Qt začínají písmenem „Q<název třídy>“.

Průhlednost – od verze 4.1 byla zlepšena podpora alfa-kanálu, ale i použití masky pro průhlednost. Od této verze je možnost využití „backing store“ (všechny operace s okny se dějí v paměti a okna se pak kopírují na obrazovku) na všech platformách. Tato vlastnost využívá „persistent off-screen buffer“ (vykreslování obrazovky neprobíhá přímo na obrazovku, ale mimo ní do paměti pro to určené a následně se zobrazí až výsledný obraz) pro každé okno. Všechny rastrové operace jsou vykreslovány na „backing store“, který velmi zrychluje vykreslování na všech platformách.

Tabulka 2: Qt licence (2)

	Komerční verze	**LGPL verze**	**GPL verze**
Cena licence	Licenční poplatek	zdarma	zdarma
Musí se poskytovat zdrojový kód	Ne, může být uzavřený	Ano	Ano
Možnost vytvořit proprietární aplikaci	Ano	Ano podle licence LGPL	Ne
Předpoklad aktualizací	Ano	Ano, volně dostupné	Ano, volně dostupné
Podpora	Ano	Ne ale dá se dokoupit zvlášť	Ne ale dá se dokoupit zvlášť
Poplatek za Runtimes	Ano	Ne	Ne

1.3 wxWidget

Historie – wxWidgets dříve pojmenovaný wxWindows byl odstarován v roce 1992 Julian-em Smart-em na Universitě Edinburgh. 20. února 2004 vývojáři wxWindows oznámili, že tento projekt bude přejmenován na wxWidgets. Protože Microsoft žádal Juliana Smarta, aby respektoval Microsoft. Z důvodu že Windows je obchodní známkou Microsoftu ve Spojených státech.

Podporované platformy jsou X Window, Win32, MacOS, OS/2, Embedded Devices, PalmOS. Programovací jazyk: hlavním jazykem je C++, ale je portována i do jazyků BASIC, Eiffel, Java, Javascript, Lua, Perl, Python. **Licence** wxWidgets je možné provozovat pod licencí wxWindows library license, GNU GPL, GNU L-GPL. **Rapid Application Development** – nemá vlastní IDE ani integraci do nějakého prostředí.

Dokumentace je velmi komplexní.

Přednosti – používá nativně widgety. Podporuje zdroj obrázků XML. Zabudovaný konfigurační manager. Má zabudovanou podporu tisku (generuje Postscript na X). Integrovaný OpenGL, HTML.

Nedostatky – tato knihovna nepodporuje namespace a používá prefix „wx“. Hlavním nedostatkem je, že nepodporuje nativně témata, tedy uživatel nemá kontrolu nad tím, jak jeho okna budou vypadat.

1.4 FLTK

Historie – Fast Light Tool Kit je grafická knihovna pro C++ s nástroji pro X (UNIX), OpenGL a Microsoft Windows NT 4.0, 95 a 98. Byla původně založena panem Billem Spitzakem a v současné době ji udržuje malá skupina programátorů po celém světě s centrálním úložištěm v USA.

Podporované platformy jsou UNIX/Linux (X11), Microsoft Windows, MacOS X. **Licence** této knihovny je vedena pod GNU L-GPL. **Rapid Application Development** – nemá vlastní IDE ani žádnou integraci do žádného prostředí.

Dokumentace – tvořena referencemi a tutoriály. Je velmi kvalitní, přehledná a srovnatelná s dokumentací Qt.

Přednosti FLTK – je velmi kladen důraz na optimalizaci a velmi malé obsazení paměti. Programování je jednoduché, tedy standardní C++. Přímá podpora OpenGL, GLUT. Hlavní smyčka non-blokátorů.

1.5 Fox Tool Kit

Historie – vývoj Fox Toolkitu začal v létě roku 1997. Po testování na několika různých systémech od OSF Motif, NeXTstep, MS Windows a systému Intergraph 5 autor navrhl představy, jak by měl ideální GUI toolkit vypadat. Po několika falešných začátcích a experimentech s různými nápady pod různými operačními systémy se zrodil Fox Toolkit.

Podporované platformy – hlavní podporované platformy jsou Unix/X11 a MS Windows, jako vedlejší je možné použít na SGI IRIX, Sun OS/Solaris, HP-UX, IBM AIX, DEC ALPHA/COMPAQ Tru64. **Licence** této knihovny je vedena pod GNU LGPL. **Rapid Application Development** – nemá vlastní IDE ani integraci do nějakého prostředí.

Dokumentace je dosti obsáhlá, ale strukturovaná, spíše jako návody „HOW-TO“.

Přednosti, podporuje OpenGL. Má zabudovaný systém pro regulární výrazy a konfigurační manager. A od verze 1.6 využívá namespace.

2 Kompilace programů pro jinou platformu (cross-compiling)

Co je vlastně cross-compiling? Pod tímto anglickým spojením dvou slov, kde v přímém překladu cross znamená kříž a compiling kompilace. Tedy křížová kompilace. Cross-kompilace je překlad programů či jakéhokoliv kódu do binárního stavu spustitelného systémem pro platformu, která je jiná než ta, na které kompilujeme.

V práci se zaměříme na kompilaci pro platformu ARM, kterou používá například i Palm. Teď by někdo mohl namítat, na co je toto potřeba, vždyť si to každý může zkompilovat na platformě, na které to budeme používat. Právě, že tomu tak vždy není, představme si systém – v případě této práce čtečku karet v autobuse. Tento systém za prvé nemá skoro žádný výstupní interface, a hlavně nemá pro kompilaci dostupné potřebné prostředky jako je například místo pro rozbalení zdrojových souborů či paměť RAM. Pokud bychom si představili systém, který má prostředky na kompilaci, tak cross-kompilace může mít upotřebení i zde. V představované sestavě nebudeme mít dost rychlý výpočetní výkon a kompilace by byla zdlouhavá. Můžeme si kompilaci provést právě na sestavě o mnoho rychlejší a ušetřit tak čas.

Teď, když víme proč kompilovat, si musíme položit otázku jak tedy cross-kompilovat? Před samotnou kompilací bychom si měli zodpovědět pár velmi důležitých otázek:

Jakou máme sestavu?

Platforma ARM s procesorem AT91SAM9263 (200 MHz), s grafickým výstupem QVGA či VGA. Operační sytém Linux 2.6.22. Používán je tenký framebufferový X Server Xfbdev.

Na jakou platformu se bude kompilovat?

Kompilovat na procesor ARM (AT91SAM9263).

Teď tedy víme, pro jakou platformu musíme náš program přeložit. Je třeba si připravit knihovny a programy pro cross-kompilaci. Použijeme systém Kubuntu a sestavu disponující mobilním procesorem AMD Turion™ 64 1MB L2 cache, tedy platforma x86. Pro názornost využijeme programy, které jsou dostupné z příkazové řádky.

Pro ukázku zde uveďme cross-kompilaci v prostředí Netbeans 6.7beta, toto prostředí je používáno pro vytvoření testovacích aplikací.

Je třeba získat tool chain potřebný pro cross-kompilaci na platformu *ARM9*, která se nachází na adrese *http://www.codesourcery.com/gnu_toolchains/arm*. Poté spustit Netbeans. Po spuštění Netbeans přejít do záložky *Tools->Options->C/C++*, kde je třeba v levé části založit novou kolekci a *Base Directory* nastavit na cestu k rozbalenému tool chain pro *ARM9*. Jméno lze volit libovolné. Poté je třeba v nastavení projektu pod *C/C++ Compiler->Additional options* nastavit naši architekturu pomocí přepínače *-march=armv5te.* Po této úpravě už lze kompilovat pro zvolenou architekturu.

3 Vhodní kandidáti pro testování

Vhodné kandidáty máme už v podstatě vybrané v přehledu grafických knihoven. V přehledu grafických knihoven by měly být všechny knihovny, ale zde je vybráno pět knihoven, které jsou více či méně používané po celém světě. Ostatní neuvedené knihovny jsou buď tak malé a v počátcích, že se nedají využít, a spíše s nimi řešíme problémy. Nebo tu jsou knihovny, které jsou zaměřeny pro jeden druh práce. Proto máme vybráno pět knihoven už na začátku práce. Tyto knihovny jsou v dnešní době předně používané.

Při výběru vhodných kandidátů bylo překvapení, jak málo je dostupných knihoven. Které se dají využít pro běžnou práci. V podstatě na trhu jsou dvě největší knihovny GTK+ a Qt. Tyto knihovny jsou nejvyvinutější a je to dáno i jejich dlouhou historií. Proto pokud nehledáme přesné zaměření na určitý problém, jsou dostupní tyto dva giganti.

4 Využité programy a technika pro testování

4.1 Příprava potřebných knihoven a programů

Jak už bylo znázorněno výše, testování provádíme na distribuci Ubuntu (Ubuntu 9.04). Tato distribuce je odnoží Debianu, tedy používá balíčky formátu „.deb“. Teď si připravíme systém pro testování. Nainstalujme si tedy Xfbdev pro testování, je to tenký X Window server. Za další budeme potřebovat pro testování potřebné soubory jednotlivých knihoven tedy Qt, wxWidgets, FLTK, Fox Toolkit, Gtk+ plus jejich developer balíčky „-dev“. Pro naprogramování jednotlivých testovacích aplikací je využito prostředí Netbean verze 6.7beta. Hlavním testovacím programem pro testování je valgrind, taktéž si ho nainstalujme.

Aby se zde nemusely vypisovat jednotlivé instalace, tak v příkazové řádce spusťme tento instalační příkaz:

```
sudo apt-get install libqt4-core libqt4-dev libgtkmm-2.4-1c2a libgtkmm-2.4-dev
libfltk1.1 libfltk1.1-dev libfox-1.6-0 libfox-1.6-dev libwxbase2.8-0 libwxbase2.8-
dev valgrind
```

4.2 Testováni využité paměti

U této problematiky jsem strávil velmi času. Nejdříve jsem chtěl využívat přímo C++ funkce a zjistit velikost zabrané paměti, ale následně jsem zjistil že to dosti dobře nejde, při dynamické alokaci začíná být člověk v koncích a vlastně by napsání takového testu zabralo více než samotný test, takže jsem od této varianty upustil. Možná by to ani nakonec nešlo, ale to už jsem dále nezjišťoval.

Poté jsem zjistil, že existují programy pro ladění ale žádný mi nevyhovoval nedával informace co jsem potřeboval. Z těchto programů myslím například oprofiler, gprof. Nakonec jsem narazil na program valgrind, který obsahuje velmi ladících a profilovacích programů.

Paměť se tedy testuje pomocí ladícího a profilovacího programu valgrind. Tento nástroj automaticky detekuje mnoho chyb správy paměti. Takže můžeme zjišťovat, kde program využívá špatně paměť, odladit ho a docílit tak větší stability. Může se také provést podrobná diagnostika pomocí profilovacích programů a tím zrychlit program.

Valgrind má více programů pro testování, jak už bylo zmíněno a zde se pokusíme přiblížit ty hlavní:

- **Memcheck** – Tento program se zaměřuje, jak už z názvu vyplývá, na testování paměti pro programy napsané v C a C++. Memcheck informuje o jakékoliv chybě, která v paměti nastává, například přetečení, paměť není uvolněna a je nedostupná. Okamžitě vypisuje chybovou správu či informaci s číslem, které značí číslo řádku v našem kódu programu.
- **Cachegrind** – Detailní profilovací program cache, provádí simulaci L1, L2, D1 cache pamětí procesoru.
- **Callgrind** – Rozšíření Cachegrind. Poskytuje veškeré informace, které Cachegrind, plus další informace o callgraphs. A samostatně je pro vizualizaci nástroj KcacheGrind.
- **Massif** – Tento program, je profilační aplikace haldy. Provádí detailní profil haldy, přičemž v pravidelných momentech z programu haldy vyrábí graf zobrazující využití haldy v čase, včetně informací o tom, které části programu jsou odpovědné za přidělení paměti. Pod Massif běží programy asi 20x pomaleji než normálně.
- **Helgrind** – Je vláknový debugger, který zjistí údaje o vícevláknových programech.

4.1 Testování času

Test času se dělá přímo v programu a vypisuje se do konzole. Test času se dá vypínat či zapínat pomocí parametrů. K měření času je použita C++ třída clock_t, která měří čas procesoru strávený programem na procesoru. Nezapočítává do měření čas, kdy je program například odstaven a čeká na přidělení procesoru.

4.2 Parametrizace programů

Jak už je naznačeno výše, každý program dané knihovny má následující parametry:

- **-p** – počet opakování testované věci,
- **-dt** – zapne měření času,
- **-b** – testování tlačítek,
- **-l** – testování popisků,
- **-h** – výpis všech parametrů a nápovědy k programu.

4.1 Server X (Xfbdev)

Server X je grafický server umožňující zobrazení grafiky a spouštění aplikací s grafickým uživatelským rozhraním (GUI – Graphics User Interface). Protože grafické prostředí běží jako server, umožňuje zobrazení i na vzdálených linuxových nebo unixových stanicích. Jako grafické zařízení používá FrameBuffer (/dev/fb0).

Tento server je využíván ze dvou důvodů, první z nich je, že zadáním práce bylo použít tento server pro testování, ale jsou v něm i jiné výhody – tento server totiž díky své nenáročnosti využívá velmi málo systémových prostředků. Proto test bude minimálně zkreslený.

Spuštění serveru X provedeme příkazem:

```
Xfbdev -ac -br -screen 640x480 -mouse mouse
```

Jelikož na serveru X nebude spuštěno žádné grafické prostředí, musíme posílat do serveru přímo daný testovací program. To se provede pomocí linuxového exportu obrazovky, kde základní nastavení je nultý server:

```
export DISPLAY=:0
```

Dále na této obrazovce spustíme Xfbdev server:

```
Xfbdev -ac -br -screen 640x480 -mouse mouse &
```

Jelikož byla exportována globální proměnná DISPLAY, můžeme nyní spustit jakoukoliv grafickou aplikaci v této konzoli a výstup se nám zobrazí na serveru X. Pokud chceme aplikaci či cokoliv v Linuxu poslat na jinou obrazovku, použijeme hodnotu proměnné DISPLAY ve tvaru <program> :<číslo obrazovky>.

4.2 Konfigurace framebuffer

Pokud chceme Xfbdev používat, musíme si nastavit parametry jádra kernel *vesafb:mtrr,ywrap vga=ask*. Tyto parametry nám zapnou framebuffer a po zavedení jádra se nás zeptá na rozlišení obrazovky, my si nastavíme *640×480×16* a provedeme v tomto nastavení veškeré testy.

Abychom pokaždé nemuseli psát zaváděcí parametry, vložme si do /boot/grub/menu.ls tedy do konfiguračního souboru GRUB tyto řádky:

```
title		Ubuntu 9.04, kernel 2.6.28-11-generic /w Xfbdev & Framebuffer
uuid		5230f288-2094-46c5-a325-39737cf5ee37
kernel		/boot/vmlinuz-2.6.28-11-generic root=UUID=5230f288-
2094-46c5-a325-39737cf5ee37 rhgb quiet vesafb:mtrr,ywrap vga=ask
initrd		/boot/initrd.img-2.6.28-11-generic
quiet
```

4.3 Použitý správce oken pro Xfbdev

Testovací aplikace nejsou napsány tak, aby se korektně po vytvoření a zobrazení všech elementů zavřely. Byl využit jeden z mnoha správců oken pro Xfbdev. Hlavní prioritou při výběru je nenáročnost. Jako první byl vybrán FluxBox, ale tento správce oken se již tak vyvinul a pro testování je zbytečně složitý. Proto byla zvolena velmi minimalistická sada programů pro správu plochy Matchbox, která je nenáročná na výkon počítače. V práci využijeme jen jednu část z této sady a to správce oken matchbox-windows-manager. Pro testování je velmi důležité, aby tento správce byl nenáročný a to je splněno na jedničku.

Instalace:

```
sudo apt-get matchbox-windows-manager
```

A pokud je spuštěn na počítači Xfbdev, spustíme správce oken příkazem:

```
matchbox-windows-manager
```

4.4 Příprava skriptu pro testování

Pro testování máme připravený skript, abychom nemuseli jednotlivé kroky pořád opakovat ručně. Tento skript je napsán pro shell. Uvedu zde pro představu část okomentovaného kódu, který bude srozumitelnější než popisování v textu.

```
#!/bin/sh
# Nastavení globálních proměnných
pathLib="lib" #cesta do adresáře s testovacími aplikacemi
pathSave="save" # cesta do adresáře kam chceme ukládat výsledky
rm -R $pathSave # smaže adresář do které ho chci ukládat
mkdir $pathSave # a znovu ho vytvořím
programs=`ls $pathLib` # načte do pole programy z daného adresáře

for program in $programs
do
echo "Spustim program: $program"
mkdir $pathSave/$program

# Test programu bez parametru
jmeno="$program-massif"
# Test paměti pomocí programu vagrind
/usr/bin/valgrind -v --tool=massif --time-unit=B --max-snapshots=1000 --detailed-
freq=1 –peak-inaccuracy=0.5 --massif-out-file=$pathSave/$program/$jmeno.out
$pathLib/$program
# Převedení výstupu valgrind do dsv souboru
./ms_print --csv $pathSave/$program/$jmeno.out > $pathSave/$program/
$jmeno.csv
# Test času stráveného na procesoru
./$pathLib/$program -dt > $pathSave/$program/$jmeno-testtime.txt

#Test programu s parametry
jmeno="$program-massif-p10-l"
prepinace="-p 10 -l"
/usr/bin/valgrind -v --tool=massif --time-unit=B --max-snapshots=1000 --detailed-
freq=1 –peak-inaccuracy=0.5 --massif-out-file=$pathSave/$program/$jmeno.out
$pathLib/$program $prepinace
./ms_print --csv $pathSave/$program/$jmeno.out > $pathSave/$program/
$jmeno.csv
./$pathLib/$program $prepinace -dt > $pathSave/$program/$jmeno-testtime.txt

… pokračování najdete na přiloženém CD
```

Dále se kód opakuje pro všechny parametry například pro dvacet tlačítek atd. Pokud se podíváte do kódu výše, tak zde najdete řádek začínající „/usr/bin/valgrind“, což je program pro testování paměti. Parametrem „-tool“ nastavíme jeho podprogram massif, který slouží pro testování využité paměti.

Dále nastavuji parametrem „-time-unit“ časovou jednotku byte, kdy při každé změně v bytech se nám bude generovat výstup. Další parametr je „-max-snapshot“, kterým nastavíme maximum zaznamenaných testů, pokud valgrid přesáhne toto číslo, vezme se každé druhý záznam, tedy padesát procent. Pokud tedy máme 1000 záznamů vezme se 500 a pokračuje se v testování. Tímto se dá dosáhnout přesnosti i šetření místa.

Dále parametr „-detailed-freq“ je nastavení, po kolika testech se bude dělat detailní výstup, nastavíme zde „jedna“ pro každý test, abychom mohli zkoumat, co se přesně v testovací aplikaci provádí. A poslední parametr „-peak-inaccuracy“ nám udává procentuální část od posledního záznamu. Ve skriptu je nastaveno: pokud nedosáhne větší než 0,5%, tak se nezaznamenává. Čím je tato hodnota menší, tím se valgrind exponenciálně zpomaluje.

Dalším programem, který se využívá je pro generování srozumitelných údajů pro člověka. K tomu se využívá skript, který se dodává přímo s programem valgrind, ale velkou část tohoto skriptu máme upravenou pro naše potřeby, a to aby generoval soubory CSV, které následně načítáme do tabulkového procesoru. Poté už jen danou aplikaci se zadanými parametry otestujeme na časovou náročnost. Tuto proceduru opakujeme pro všechny testy a programy v cyklu.

5 Ukázka kódu programu

Zde představíme napsaný kód pro knihovnu Qt. Zdůrazníme zde část kódu, která je důležitá pro testování. Uvedu zde okomentovaný kód, který je přehlednější než cokoliv jiné.

5.1 Hlavní část programu zavedení knihovny Qt

```
int main(int argc, char *argv[]) {
	// zavedení knihovny Qt
	QApplication a(argc, argv);
	QtTest w;
	w.show();
	// dekódování argumentů programu, které musí vědět před ostatními
	int optind = 1;
	bool ok;
	while ((optind < argc)) {
		QString sw = argv[optind];
		// počet opakování daného testování
		if (sw == "-p") {
			optind++;
			QString val = argv[optind];
			_pocetOpakovani = val.toInt(&ok, 10);
		// zapnutí/vypnutí testování času
		} else if (sw == "-dt") {
			_debugTime = true;
	}
		optind++;
	}
	optind = 1;
	// dekódování ostatních argumentů programu
	while ((optind <= argc)) {
		QString sw = argv[optind];
		// zapnutí testu tlačítek
		if (sw == "-b") {
			w.testButton(_pocetOpakovani, _debugTime);
		}
		// zapnutí testu popisků
		if (sw == "-l") { w.testLabel(_pocetOpakovani, _debugTime); }
		optind++;
	}
	return a.exec();
}
```

5.2 Ukázka jedné testovací metody

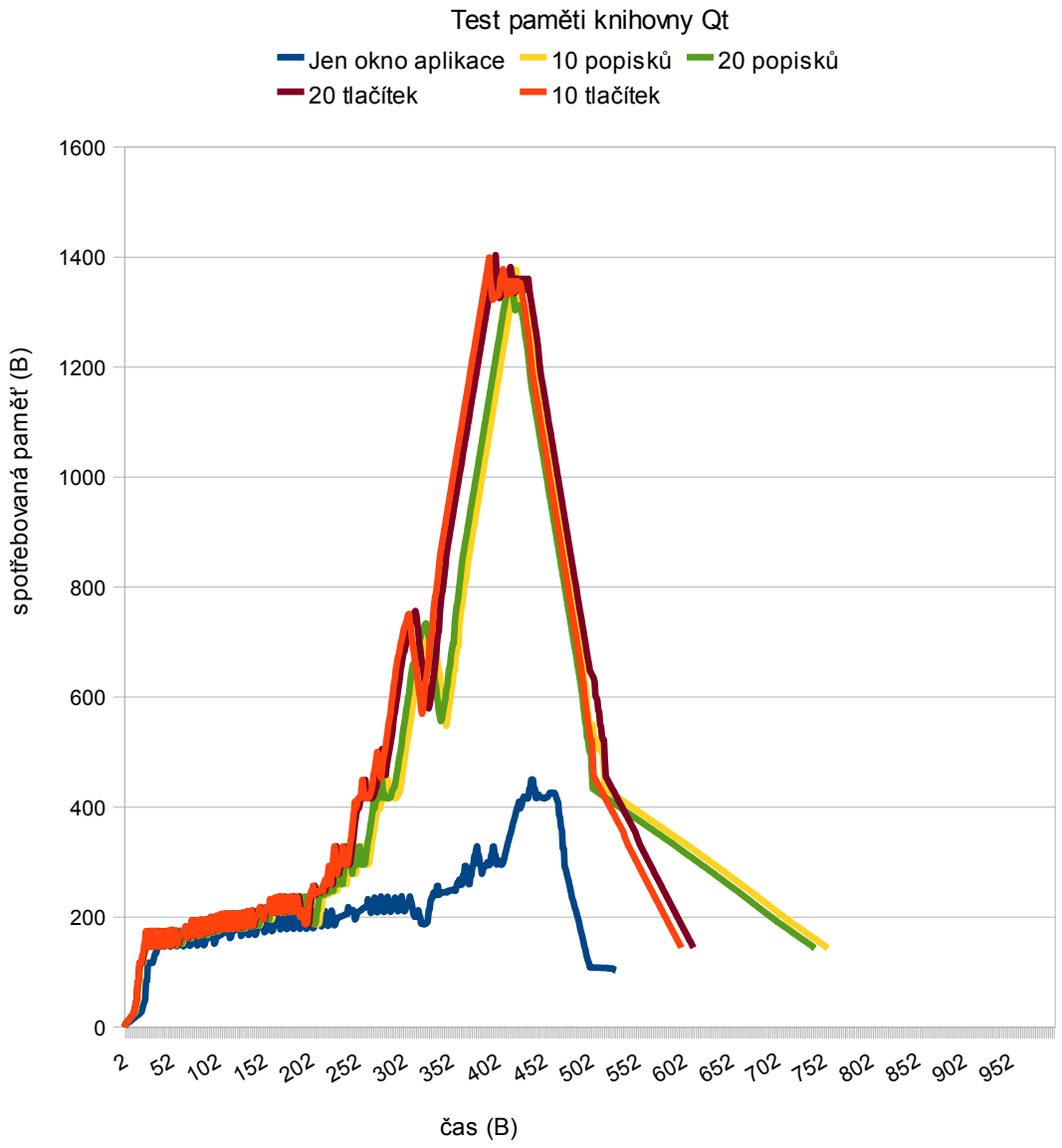
Tato metoda slouží k testování tlačítek Qt. Ostatní metody i pro další knihovny jsou chronologicky stejné. Liší se jen v sytaxi pro použitou knihovnu.

```
// Test tlačítek
void QtTest::testButton(int pocetOpakovani, bool debugTime) {
	// Vytvoření pole ukazatelů na tlačítka
	QPushButton *PB[pocetOpakovani];
	// Jestliže je zaplé testování času zapni ho
	if (debugTime)
		start = clock();
	// Vytvoření a zobrazení samotných tlačítek
	for (int i = 0; i < pocetOpakovani; i++) {
		PB[i] = new QPushButton("Test Button", this);
		PB[i]->resize(100, 30);
		PB[i]->move(30, 30 + i * 30);
		PB[i]->show();
	}
	// Změření času a výpis hodnoty na obrazovku konsole
	if (debugTime) {
		end = clock();
		printf("%s show %f seconds\n", __FUNCTION__, (double) (end -
start)
				/ CLOCKS_PER_SEC);
	}
	// Testování času změny textu na tlačítkách
	if (debugTime)
		start = clock();
	// Změna textu tlačítka
	for (int i = 0; i < pocetOpakovani; i++) {
		PB[i]->setText("Zmena textu");
	}
	// Změření času a výpis hodnoty na obrazovku konsole
	if (debugTime) {
		end = clock();
		printf("%s setText %f seconds\n", __FUNCTION__, (double) (end -
start)
				/ CLOCKS_PER_SEC);
	}
}
```

6 Testování jednotlivých knihoven

6.1 Test paměti knihovny Qt

Ot je velmi využívaná grafická knihovna, která už má neduhy takto velké knihovny. Jelikož takováto knihovna je velmi využívaná, musí být všestranně napsaná. Tudíž musí zahrnovat plno kódů, které pro konkrétní situaci nepotřebují, ale tento problém mají všechny takto velké knihovny a musíme s tímto faktorem počítat. Tuto vlastnost má i dále zmiňovaná knihovna GTK+.



Obrázek 2: Test paměti Qt

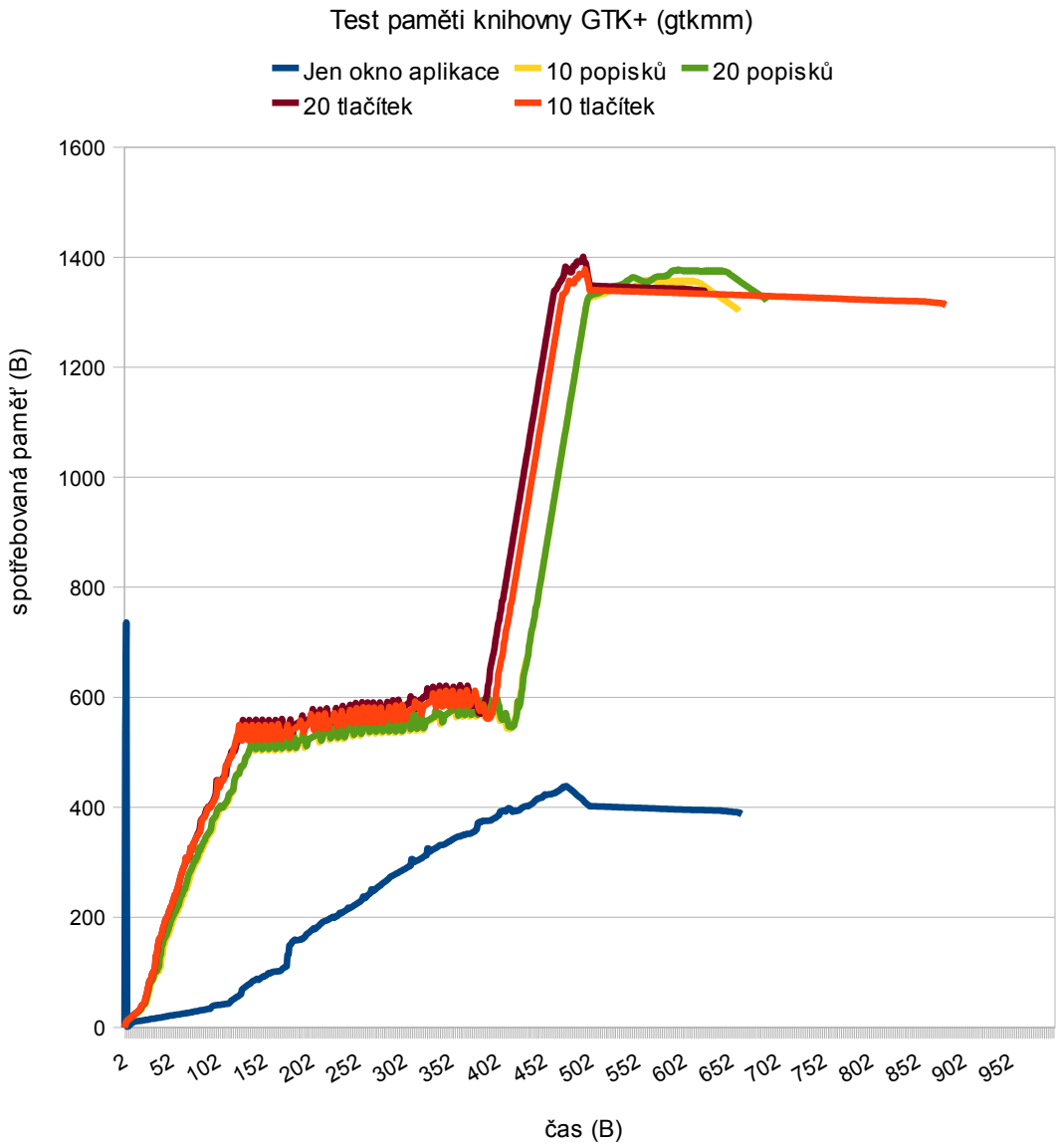
Obrázek 2 zobrazuje využití paměti této knihovny, srovnání s ostatními knihovnami se budeme zabývat dále, zde uvedeme jen odlišnosti práce s pamětí, které můžeme vyčíst z tohoto grafu. Hlavní odlišností proti ostatním knihovnám je fakt, že jediná knihovna Qt při ukončování aplikace nejdříve uvolní všechny prvky (widgety) z paměti a po té teprve zavře aplikaci. Pokud ukončujeme aplikaci, mohlo by se toto zdát zbytečné, vždyť se o toto postará systém, ale nemusí tomu tak být a mohlo by nám v paměti zůstat alokovaná paměť. Je to i důkaz že Qt je čistě napsaná knihovna a nenechává nic na nadřazených procesech.

Z grafu se dá rovněž vyčíst, že při vytváření popisků či tlačítek zabere stejně místa v paměti v rámci chybové odchylky. A pokud se podíváme na časovou osu zjistíme, že při vytváření tlačítek knihovna Qt udělá v paměti méně operací, než při vytváření popisků. Také se dá z grafu vyčíst, že si v paměti vytvoří jedno tlačítko/popisek a dále tuto komponentu využívá a při vytváření dalších tlačítek/popisků se na stejné věci jen odkazuje.

6.2 Test paměti knihovny GTK+

Nedostatkem GTK+ oproti Qt i když je to stejně využívaná knihovna se při ukončování vůbec nestará o uvolnění prvků z paměti. Není to zas tak velký nedostatek, ale myslím si, že by takto velká knihovna by se měla starat o uvolnění paměti před svým ukončením.

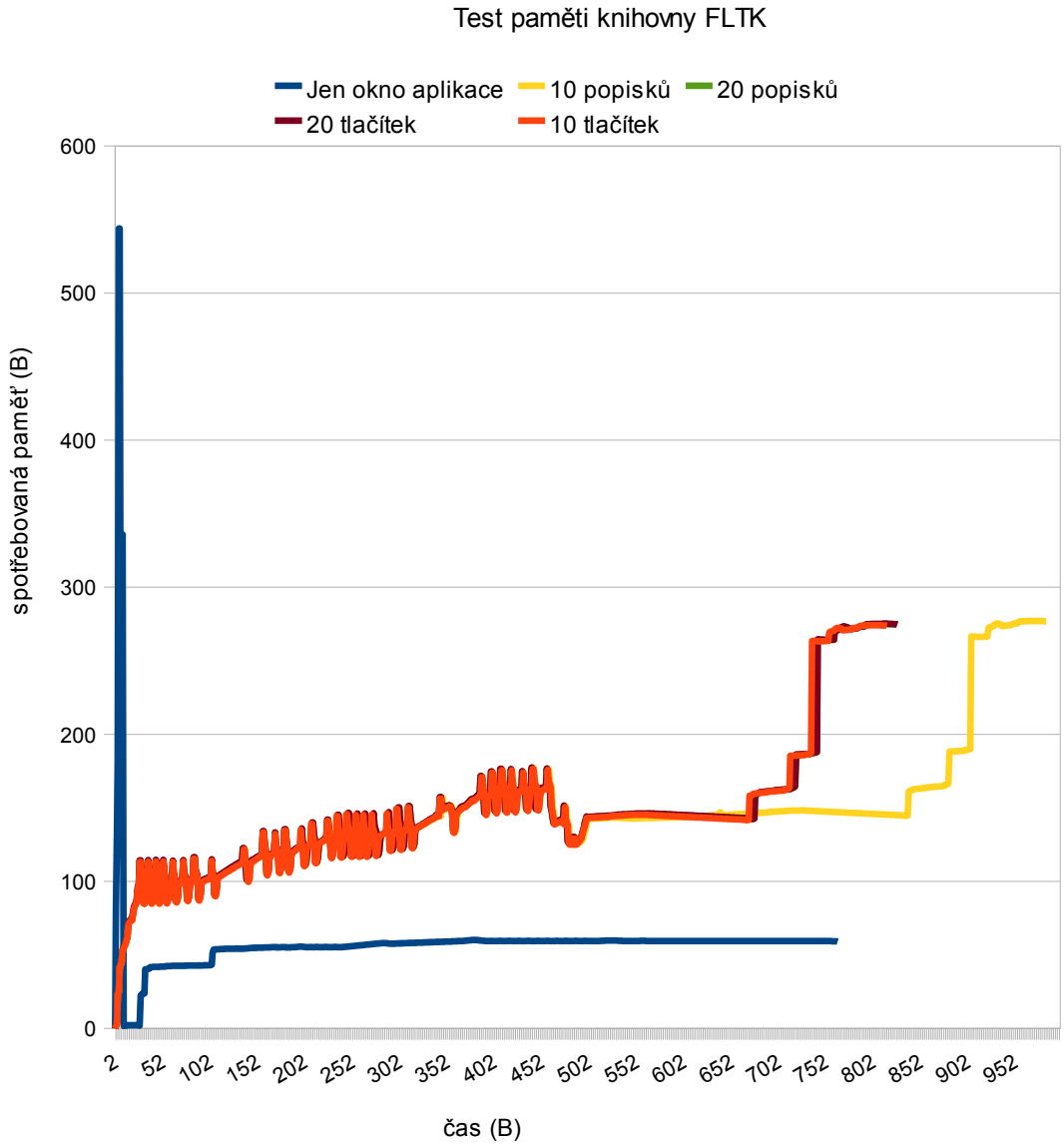
Z grafu *Obrázku 3* je viditelné, že nároky na paměť ve srovnání s Qt jsou stejné v rámci chybové odchylky. Z grafu je vidět, že i u knihovny GTK+ udělá více operací v paměti při vytváření popisků.



Obrázek 3: Test paměti knihovny GTK+

6.3 Test knihovny FLTK

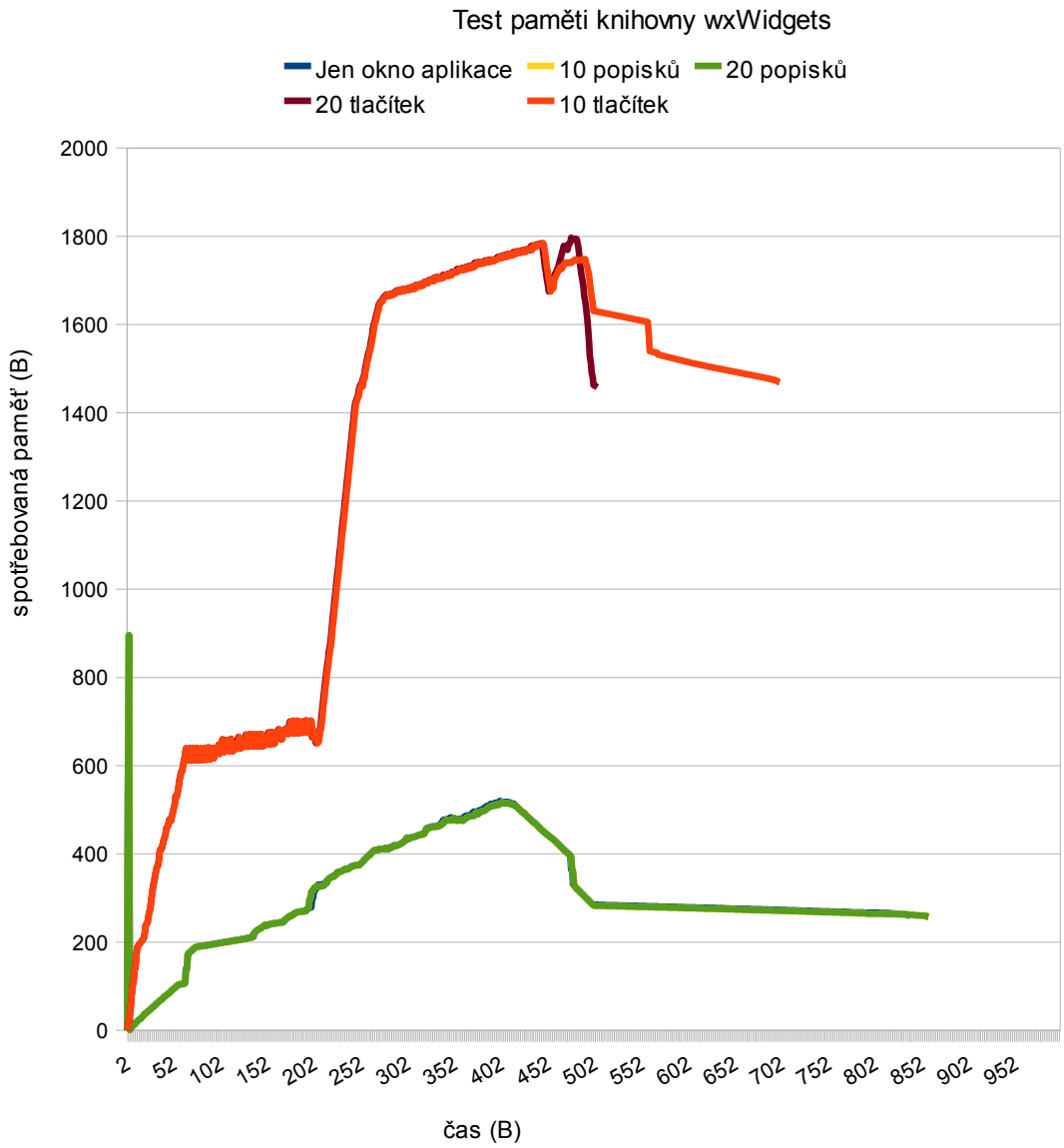
V dokumentaci k této knihovně je kladen největší důraz na čistotu kódu a práce s pamětí. Pokud se podíváme na *Obrázek 4* vidíme, že to nebyl jen trik, jak upoutat pozornost na tuto knihovnu. Z grafu je zřejmé, že pro zobrazení spotřebuje velmi málo paměti a při porovnání s ostatními knihovnami má tato knihovna nejmenší nároky.



Obrázek 4: Test paměti knihovny FLTK

7 Test knihovny wxWidgets

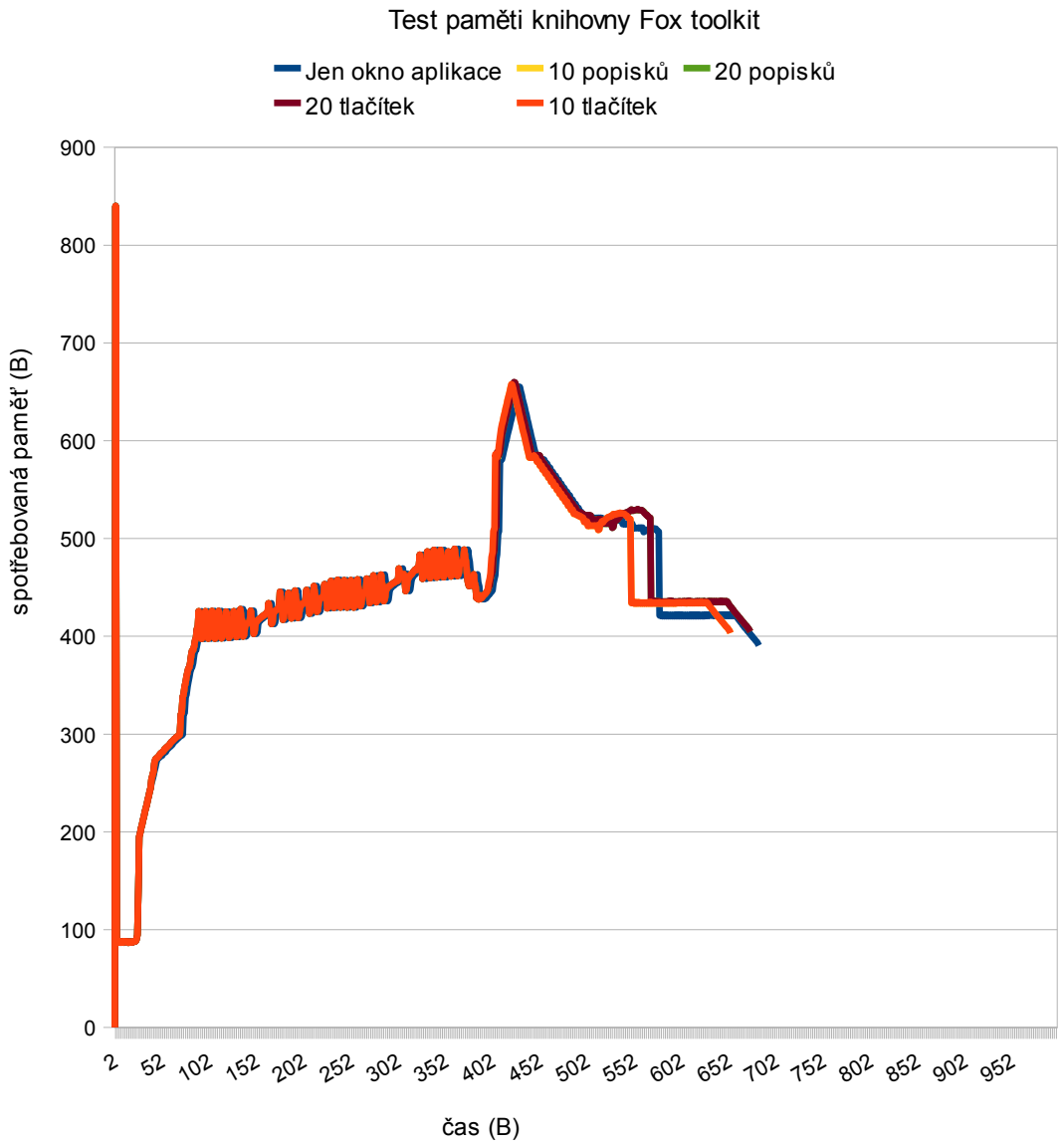
Při zjišťování informací o této knihovně bylo zjištěno, že je tato knihovna dosti využívaná. Ale pokud si porovnáme *obrázek 5* s ostatními knihovnami kromě knihovny FLTK zjistíme, že tato knihovna i při takto lehkých operacích jako je vykreslení tlačítek, spotřebuje nejvíce paměti. I když na druhou stranu nejméně paměti při vytvoření popisků.



Obrázek 5: Test paměti knihovny wxWidgets

8 Test knihovny Fox Toolkit

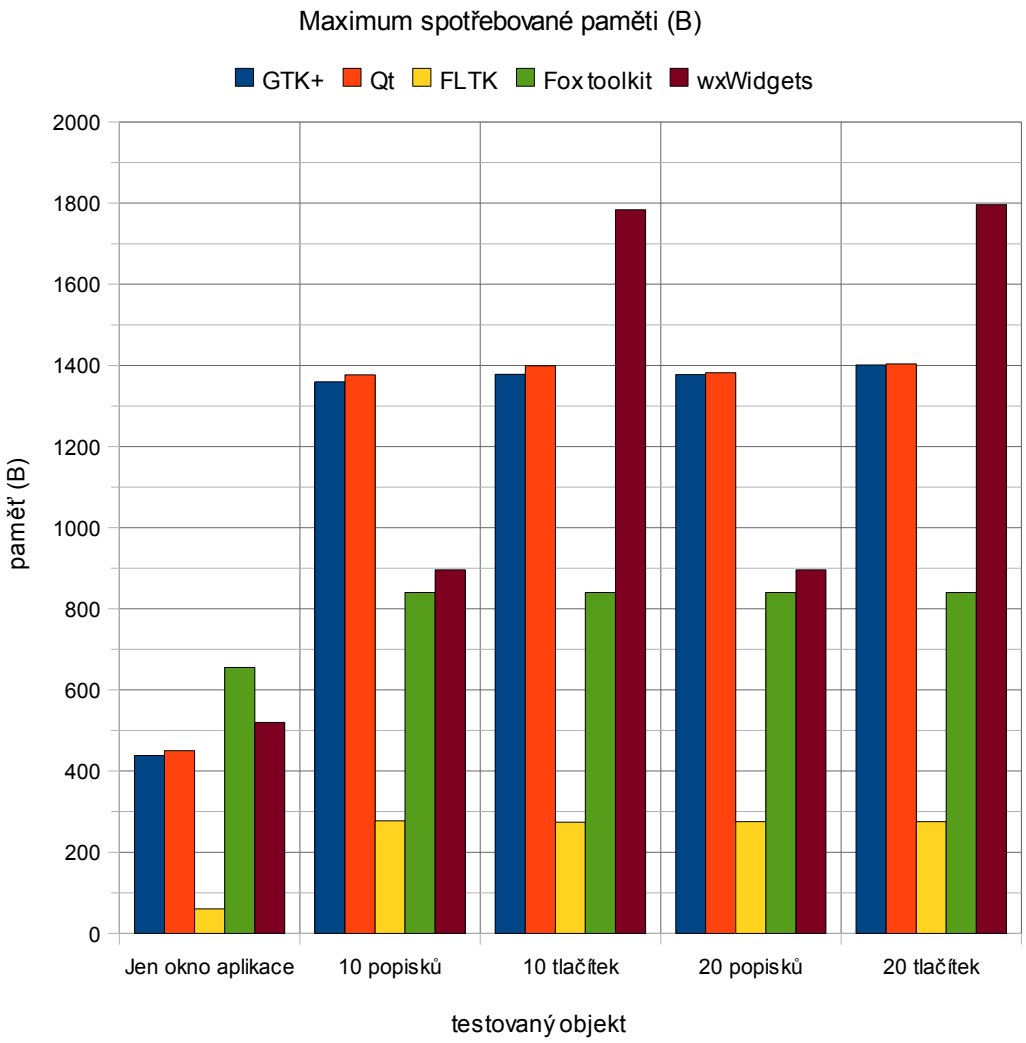
Po zhodnocení této knihovny z *obrázku 6* je zřejmé, že tato knihovna oproti ostatním knihovnám používá jiný algoritmus pro vytváření svých komponent. Z grafu je zřejmé, že ať se vytvoří jen okno aplikace či dvacet tlačítek, testovací aplikace nám v paměti zabrala stejně místa v rámci chybové odchylky.



Obrázek 6: Test knihovny Fox toolkit

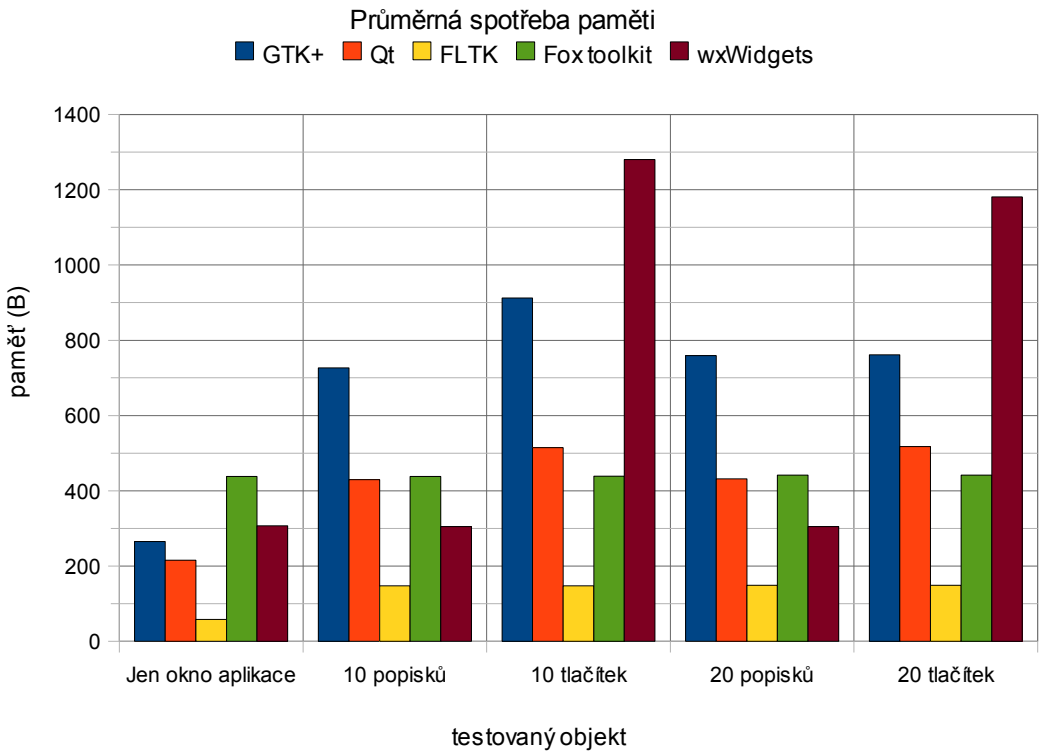
9 Celkové porovnání paměťové náročnosti všech knihoven

Předem musíme upozornit o snahu o co nejmenší zktreslení a u vést faktrory, které ovlivnily měření. Hlavním faktorem je, že byl využit správce oken Matchbox-Windows-Manager, ten zvedne využitou paměť i když velmi málo, protože tento okenní manager je velmi nenáročný. Ale využívá se ho u všech testovaných knihoven, takže navýšení o tuto hodnotu je u všech stejné.



Obrázek 7: Maximum spotřebované paměti

Pokud se podíváme na podrobné grafy testovaných knihoven, vidíme, že na začátku vzniká u všech knihoven kromě Qt chyba. Tuto chybu si neumím vysvětlit, ale domnívám se, že by mohla být zapříčiněna kódem pro zavádění knihovny. Proto jsem tuto chybu vypustil pro graf maximálních hodnot na *obrázku 7*, protože tato hodnota by výsledky ovlivnila v tomto grafu. A beru tuto hodnotu jako chybu měření. Pokud vás zajímají průměrné hodnoty jsou zobrazeny na *obrázek 8*.



Obrázek 8: Průměrná spotřeba paměti

10 Test časové náročnosti knihoven

V testu časové náročnosti zkušební test byl proveden v prosředí KDE. Výsledné hodnoty byly větší i v řádu sekund. Z toho vyplívá fakt, že záleží i na prostředí, v kterém časovou náročnost měříme. Pokud hledáme odpověď na otázku *Proč?*, tak je to hlavně kvůli paměťové náročnosti grafických komponent správcem oken daného prostředí. Finální test byl proveden v Xfbdev, jak znázorňuje *tabulka 3*. Zde je patrné, že wxWidgets si vedlo nejhůře, za ním bychom mohli uvést GTK+, na kterém byly naměřeny nenulové hodnoty. Ostatní knihovny jsou si rovny, tedy na těchto knihovnách nebyla naměřena žádná hodnota. Jelikož test byl proveden na procesoru 1,6 Ghz, používal Xfbdev a správce oken Matchbox, které mají velmi malou náročnost na hardware a testovací aplikace jsou jednoduché, tak nebyly naměřeny velké hodnoty. Pro testování časové náročnosti bychom potřebovali velmi slabý procesor a nebo vytvořit složité aplikace.

Pokud zhodnotíme časovou náročnost knihoven, tak jednoznačně si nejhůře vede wxWidgets, u kterého se naměřily nejvyšší hodnoty. Ostatní knihovny jsou v rámci chybové odchylky stejné.

První sloupec v *tabulce 3* je nulový, protože nebylo docíleno v programech toho, abychom vykreslení okna změřili a test nebyl ovlivněn jinými faktory. Pokud bychom vytvořili druhé okno a to měřili, byla by hodnota ovlivněna hlavním oknem, z kterého by využila již zpracované části. Kdybychom měřili hodnotu od vytvoření po uzavření hlavního okna, byla by tato hodnota ovlivněna časem, kdy je okno zobrazeno.

Tabulka 3: Test časové náročnosti na procesoru

Test časové náročnosti na procesoru (sekundy)					
Knihovna	**Jen okno aplikace**	**10 popisků**	**10 tlačítek**	**20 popisků**	**20 tlačítek**
GTK+	0	0,01	0,02	0,02	0,03
Qt	0	0,01	0,01	0	0
FLTK	0	0	0	0	0
Fox toolkit	0	0	0	0	0
wxWidgets	0	0,5	0,05	0,5	0,5

11 Zhodnocení kvality měření

Na začátku jsem si myslel, že kvalita měření bude stoprocentní a že vytvořím program, nebo využiji program, pro testovaní a naměřené hodnoty jen vyhodnotím. Ale po testování vidím, že tomu tak není. Hlavním problémem při takto nenáročných testech je dnešní výkon počítačů. Hlavně při měření času by tento krok bylo vhodnější měřit na velmi pomalých výpočetních strojích (blížících se rychlostí cílové platformě). Tento krok jsem ze své práce mohl v podstatě vypustit, protože není technika, kterou bych čas kvalitně změřil. Ale zjistil jsem, že u tohoto měření je nutné si dávat pozor na okolní aspekty jako jsou správce oken prostředí a tak dále. Tyto prvky hodnotu velmi ovlivňují a zkreslují. Ale dle mého názoru a testovacích zkušeností zjišťuji, že tento bod je velmi zkreslený a je potřeba testy dělat přesně na tom stroji, na kterém daná aplikace bude provozována.

Při měření využité paměti byl využit správce oken Xfbdev. Tento správce oken je nenáročný a bude využíván i na konečném produktu. Test by mohl být proveden i bez správce oken, nebo na propracovanějších správcích jako jsou KDE a GTK+. Testy by se ve všech případech lišily, protože by se do testu nezapočítávaly respektive započítavaly grafické prvky reprezentované správcem oken.

Závěr

Tuto práci jsem si vybral jako alternativu jiné práce, která byla už zabrána. Když jsem si tuto práci vybral, myslel jsem si, že to bude naprogramování pár aplikací a zjištění daných hodnot. Ale když jsem se pustil do vypracování, zjistil jsem první problém – a to, že vůbec neznám syntaxe jednotlivých knihoven – do této doby jsem se žádnou knihovnou nepracoval v širším využití. Po třech týdnech studování dokumentací jsem si myslel, že mám vyhráno, ale to jsem nevěděl, co mě čeká za největší problém. Největším problémem se pro mne stalo, jak zjistit využitou paměť aplikací. Při zjišťování jsem obešel a poptal se u plno zkušených programátorů. Ale nikdo mi nebyl schopen doporučit spolehlivou cestu. Po dalším studování jsem narazil na program valgrind, který jsem využil v této práci.

Pokud tedy mám zhodnotit mnou vybrané a testované knihovny, tak jednoznačně pro mne zvítězila knihovna Qt. Udělala na mne dojem velmi dobře zpracovanou dokumentací, která je podobná C++. Také na mne následně po testování udělala dojem čistotou kódu – myslím tím například, že jako jediná knihovna si po sobě při uzavírání aplikace uvolňuje paměť. Dále tato knihovna velmi velkou podporu – jak už vlastní návrhářem GUI, tak podporou integrace do Visual Studia a Eclipse. Tuto knihovnu bych si vybral pro naprogramování svých aplikací.

Další knihovna, která na mně udělala dojem a kterou bych si vybral pro programování minimalistických aplikací na málo výkonné zařízení, je knihovna FLTK, která na mne udělala dojem hlavně svou nenáročností na paměť.

Na druhé straně knihovnu, kterou bych si určitě nevybral pro své aplikace je knihovna wxWidgets, i když podle článků na internetu a diskuzí ji programátoři chválí. Já jsem tu zjistil, že tato knihovna zabírá velmi paměťového prostoru pro své operace. A jako u jediné jsem naměřil vyšší hodnoty času stráveného na procesoru. Dokumentace a podpora u této knihovny není také to, co bych očekával.

Závěrem bych chtěl napsat, že tato práce mě zabrala velmi času spíše studováním než testováním a v tištěné či elektronické podobě není o této problematice dostupný takřka žádný materiál. Ale byla pro mne přínosem v daném směru. Zjistil jsem, že na trhu pro unixové systémy pro mne opravdu nemá cenu hledat alternativu knihovny QT.

Bibliografie

(1) The GTK+ Team. *GTK+ Dokumentace* [online]. c2007–2008 [cit. 2009-05-10]. Dostupný z WWW: <http://www.gtk.org/documentation.html>.

(2) Nokia. *Qt 4.5.1 dokumentace* [online]. c2009 [cit. 2009-05-10]. Dostupný z WWW: <http://doc.trolltech.com/4.5/index.html>.

(3) *wxWidgets dokumentace* [online]. c2000–2009 [cit. 2009-05-10]. Dostupný z WWW: <http://www.wxwidgets.org/>.

(4) SPITZAK, Bill, et al. *Fast Light Toolkit* [online]. Bill Spitzak and others, c1998–2008 [cit. 2009-05-10]. Dostupný z WWW: <http://www.fltk.org/documentation.php>.

(5) ZIJP, Jeroen. *Fox Toolkit dokumentace* [online]. c1997–2009 [cit. 2009-05-10]. Dostupný z WWW: <http://www.fox-toolkit.org/>.

(6) Valgrind Developers. *Valgrind dokumentace* [online]. c2000–2009 [cit. 2009-05-10]. Dostupný z WWW: <http://valgrind.org/docs/manual/manual.html>.

(7) The GTK+ Team. *GTK+ Language Bindings* [online]. c2007–2008 [cit. 2009-10-05]. Dostupný z WWW: <http://www.gtk.org/language-bindings.html>.

(8) Sun Microsystems. *Netbeans* [online]. [2000–2009] [cit. 2009-05-10]. Dostupný z WWW: <http://www.netbeans.org/>.

Příloha A

	GTK+	Qt	wxWidgets	FLTK	FoxToolkit
Domovská stránka	www.gtk.org	www.qtsoftware.com	www.wxWidgets.org	www.fltk.org	www.fox-toolkit.org
Licence	GNU LGPL 2.1	Komerční veze, GNU LGPL, GNU GPL	wxWindows library license, GNU GPL, GNU L-GPL	GNU L-GPL 2	GNU L-GPL
Podpora vzhledů	ANO	ANO	ANO ale ne nativně	ANO	NE
Rozšiřitelnost	OOP(gtkmm dědění)	Plug-in interface pro vytvoření vlastních widgetů	Standardní C++ dědění	Standardní C++ dědění	Standardní C++ dědění
Antialiasing	ANO	ANO	ANO	ANO	ANO
Průhlednost	ANO	ANO	ANO	ANO	ANO
Základní jazyk	C(gtkmm C++)	C++	C++	C++	C++
RAD	NE	ANO	NE	NE	NE

Tabulka 4: Vzájemné porovnání jednotlivých knihoven

Podpora písma					
	GTK+	**Qt**	**wxWidgets**	**FLTK**	**FoxToolkit**
Bitmapové	ANO	ANO			
Vektorové	ANO	ANO			
Freetype	ANO	ANO	ANO	ANO	ANO

Tabulka 5: Vzájemné porovnání jednotlivých knihoven