UNIVERZITA PARDUBICE
Fakulta elektrotechniky a informatiky

Návrh správce vláken v programování

Michal Picpauer

Bakalářská práce
2013

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2012/2013**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Michal Picpauer**
Osobní číslo:	**I10172**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Návrh správce vláken v programování.**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování:

Student navrhne objektového správce vláken (threads), který bude spravovat základní operace vlákna:
- spuštění vláken
- ukončení vláken
- monitorování vláken
- a další.

V praktické části student použije správce vláken na správu 10 vláken, 50 vláken. Některé vlákna budou běžet neustále (nekonečná smyčka), jiná se ukončí po provedení nějaké simulované úlohy. Praktická část bude poskytovat monitorování stavu vláken.

Rozsah grafických prací:

Rozsah pracovní zprávy:

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

Seznam odborné literatury:

MSDN
http://qt-project.org/doc/qt-4.8/
Mistrovství v C++, Computer press, 2007-10-08, 3. vydaní, Stephen Prata
C 2005 , Computer press, 2007-01-04, 1. vydání, Jurgen Bayer
C 2010, Computer press, 2012-01-02, 1. vydání, M.Virius

Vedoucí bakalářské práce: **Ing. Jaroslav Štroch**
Katedra informačních technologií

Datum zadání bakalářské práce: **21. prosince 2012**
Termín odevzdání bakalářské práce: **10. května 2013**

L.S.

prof. Ing. Simeon Karamazov, Dr.
děkan

Ing. Lukáš Čegan, Ph.D.
vedoucí katedry

V Pardubicích dne 29. března 2013

Prohlášení autora

Prohlašuji, že jsem tuto práci 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 16. 5. 2013 Michal Picpauer

Poděkování

Tímto bych rád poděkoval svému vedoucímu práce panu Ing. Jaroslavu Štrochovi za jeho odbornou pomoc a cenné rady, které mi pomohly při zpracování bakalářské práce. Dále bych chtěl poděkovat své rodině za podporu během absolvování studia.

Anotace

Náplní práce je návrh objektového správce vláken, který bude spravovat základní operace s vlákny, jako je spuštění, ukončení a monitorování vláken. Vytvořená aplikace je napsána v jazyce C++ s použitím Qt frameworku, který umožňuje snadnou přenositelnost na různé platformy.

Klíčová slova

C++, Qt, vlákna, správce vláken, QThread

Title

Design thread manager in programming

Annotation

Content of bachelor work is to design object thread manager, which will manage the basic operations with threads, like starting threads, stopping threads and monitor threads. Created application is written in C++ with using the Qt framework, which allows easy portability to different platforms.

Keywords

C++, Qt, threads, thread manager, QThread

Obsah

Seznam zkratek

CPU	Central Processing Unit
I/O	Input/Output
PC	Personal computer
OS	Operating systém
ULT	User-Level Threads
KLT	Kernel-Level Threads
GUI	Graphical user interface

Seznam obrázků

Úvod

Náplní práce je návrh objektového správce vláken, který bude spravovat základní operace s vlákny, jako je spuštění, ukončení a monitorování vláken. Vytvořená aplikace je napsána v jazyce C++ s použitím Qt frameworku, který umožňuje snadnou přenositelnost na různé platformy.

V teoretické části si nejprve přiblížíme několik termínů, které souvisejí s vlákny v programování. Je to vhodné pro pochopení, proč vůbec byla vlákna do programování zavedena a co to vlákna jsou. Dále pak bude podrobně vysvětleno, k čemu jsou vlákna vhodná. Další kapitolou pak bude seznámení s tvorbou vláken v různých programovacích jazycích. V praktické části bude popsáno, jak aplikace funguje.

V první kapitole se seznámíme s pojmy multitasking, multithreading a multiprogramming. Vysvětlíme si rozdíly mezi těmito pojmy a přiblížíme si vývoj těchto technik.

V druhé kapitole si podrobně popíšeme, co to jsou vlákna a jak fungují, kdy je vhodné vlákna použít, jejich nevýhody a možné problémy.

Ve třetí kapitole si ukážeme, jak je možné tvořit vlákna v jazyce Java, C# a v jazyce C++ s Qt frameworkem.

V poslední kapitole si představíme vytvořenou aplikace. Ukážeme si, jak aplikace funguje spolu s popisem návrhu aplikace.

1 Multiprogramming, multitasking, multithreading

1.1 Historie

„*V dobách, kdy počítače byly velké, pomalé a drahé, bylo žádoucí, aby každý z nich mohl sloužit potřebám více uživatelů. S vývojem výpočetní techniky se však vyvíjel i způsob, jakým se toho dosahovalo.“*[11]

Nejstarším používaným mechanizmem bylo tzv. dávkové zpracování[1], kdy si každý uživatel připravil předem svou úlohu ve formě dávky, a ta se pak ve vhodnou dobu spustila. Počítač se přitom věnoval právě a pouze této jedné úloze, dokud ji nedokončil. Tento mechanizmus však nevyužíval vždy všechny části počítači na plno, proto nebyl moc efektivní. Snahou odstranit nerovnoměrné vytížení jednotlivých částí počítače bylo zavedení co možná největší míry paralelismu[2]. Aby například jedna úloha v rámci své vstupně-výstupní operace pracovala s diskem, jiná mohla něco číst z magnetické pásky, a další úloha mohla mezitím provádět výpočty na procesoru.

Pokud ale procesor zůstal stále jen jeden, musely se na něm všechny úlohy vhodně střídat. Po určitý časový interval byl procesor vyhrazen jedné úloze a po uplynutí této doby se procesor začal věnovat jiné úloze, která byla také připravena ke zpracování na procesoru, ale musela čekat. Této technice se říkalo přidělování časových kvant (time slicing). Tato technika však nezajišťovala skutečný paralelní běh více úloh, ale pouze tzv. pseudoparalelismus[3]. Díky rychlému přepínání měl uživatel iluzi, že počítač se jejich úloze věnuje soustavně.

Další možností, jak dosáhnout pseudoparalelismu, bylo dobrovolné střídání úloh, kdy jedna úloha sama předala procesor jiné úloze například v okamžiku, kdy se rozhodla zahájit určitou vstupně-výstupní operaci, která mohla probíhat bez účasti procesoru.

Se zmenšováním, zrychlováním a hlavně zlevněním počítačů pomalu přestávali sloužit více uživatelům současně, ale mohly být naopak vyhrazeny jen jednomu jedinému uživateli. To však neznamenalo, že všechny doposud používané techniky a mechanismy, umožňující provozovat více úloh současně, jsou zbytečné. I jednomu jedinému uživateli se může velmi hodit, když si na svém počítači může spustit více úloh současně. Pro příklad si uvedeme situaci, kdy uživatel na svém počítači píše dokument pomocí editoru, a potřebuje si zjistit nějaké informace pomocí webového prohlížeče. Pracuje-li v prostředí, které neumožňuje současný běh více úloh, musí rozepsaný dokument uložit, práci s editorem ukončit, spustit webový prohlížeč, najít všechny potřebné informace, prohlížeč musí opět ukončit a poté znovu otevřít editor a pokračovat v psaní rozepsaného dokumentu. Pokud ale uživatel pracuje v prostředí umožňující současný běh více úloh, může mít spuštěný editor a webový prohlížeč současně a podle potřeby se mezi nimi přepínat.

[1] Z angl. batch processing. Vykonávání série programů (tzv. dávek) na počítači bez účasti uživatele.
[2] Souběžného provádění více různých činností.
[3] Paralelismus je pouze předstíraný.

Tento příklad je právě ukázkou souběžného prováděním více úloh, liší se jen technika střídání jednotlivých úloh na jediném procesoru. Zde dochází k přepínání mezi úlohami až na zásah uživatele (obvykle po mnohem větších časových intervalech než v případě techniky přidělování časových kvant), této variantě se obvykle říká context-switching (přepínání kontextu). Je to však pouze další forma realizace téhož mechanismu, současného běhu více úloh na jednom počítači neboli tzv. multitaskingu. V tomto konkrétním případě jde tedy o multitasking s přepínáním kontextu. Další variantou je pak tzv. kooperativní multitasking (cooperative multitasking), kdy si jednotlivé úlohy předávají procesor dobrovolně, kdy to samy uznají za vhodné.

Shrňme si nyní ještě jednou to, co jsme si dosud řekli. Multitasking neboli současný běh více úloh je mechanismus. Může být realizován různými způsoby (přepínáním kontextů, přidělováním časových kvant, jako kooperativní multitasking) a může sloužit různým účelům. Jednou možností je sdílení jednoho počítače více uživateli (jde o tzv. sdílení času). Další možností je provozování více úloh jedním uživatelem na jednom počítači současně, případně obě možnosti mohou být provozovány současně.

Multitasking je zajišťován operačním systémem za podmínky, že hardware příslušného počítače (jedná se především o procesor) je k tomu vhodně uzpůsoben. Příkladem operačního systému, který s multitaskingem nepočítal, byl oblíbený a zároveň i zatracovaný operační systém MS DOS pro osobní počítače (multitasking umožňovala až jeho nadstavba MS Windows, ale jen v tzv. enhanced režimu na procesorech 386 a vyšších). Opačným příkladem je operační systém Unix, který od začátku počítal se současným během více úloh a obsahuje vše, co je k tomu potřebné. Unix je navíc víceuživatelským operačním systémem, a umožňuje tedy i práci v režimu sdílení času, kdy je v rámci multitaskingu provozováno více úloh patřících různým uživatelům. [11]

1.2 Multiprogramming

První počítač používající techniku multiprogramování byl systém britský Leo III ve vlastnictví J. Lyons a Co. Několik různých programů v dávce bylo načteno do paměti počítače, a první z nich se spustil. Když první program dosáhl instrukce „čekej na periferii[4]“, byl kontext tohoto programu uložen odděleně a druhý program v paměti dostal možnost ke spuštění. Tento proces pokračuje, dokud nejsou všechny programy ukončeny.

Použití multiprogramování bylo posíleno vznikem technologie virtuální paměti a virtuálního stroje, který umožnil jednotlivým programům využívání paměti a zdrojů operačního systému tak, jako kdyby ostatní souběžně běžící programy neexistovaly nebo byly neviditelné.

Multiprogramování ale nedává žádnou záruku, že bude program spuštěn včas. Může dojít k takové situaci, kdy první spuštěný program poběží i několik hodin bez nutnosti přístupu k počítačové periferii. To se však dříve nejevilo jako problém, protože nebyli žádní

[4] Periferie je obvykle zařízení rozšiřující možnosti použití počítače (*počítačová periferie)*. Počítačová periferie slouží ke vstupu a výstupu dat z počítače.

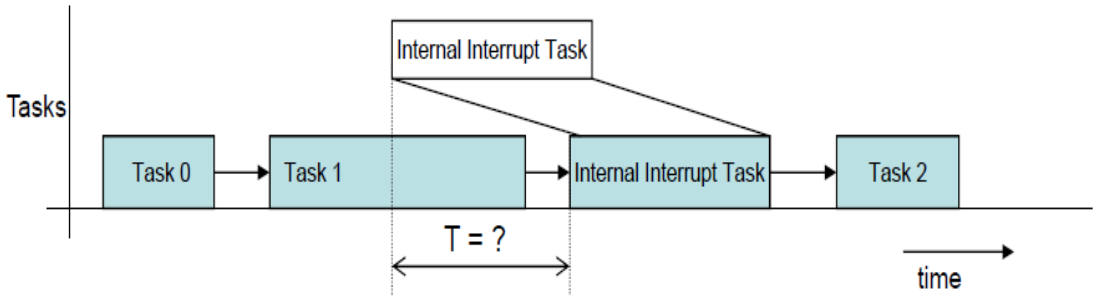
uživatelé čekající na terminálu. Uživatelé odevzdali balíček děrných štítků operátorovi a vrátili se o několik hodin později pro výsledky tisku. Multiprogramování výrazně snížilo čekací doby, kdy bylo zpracováváno více dávek najednou. [15]

1.3 Kooperativní (nepreemptivní) multitasking

Jeho princip vychází přímo z mechanizmu přepínání programů. Tam mohlo dojít k přepnutí kontextu pouze na žádost uživatele. To vedlo k tomu že, značné množství procesorového času bylo nevyužito. To si tvůrci kooperativního multitaskingu uvědomovali a byli vedeni snahou tento nevyužitý čas přidělit jinému procesu.

Kooperativní multitasking je druh multitaskingu počítače, ve kterém operační systém nikdy neiniciuje přepnutí kontextu z běžícího procesu na jiný proces. Nazývá se kooperativní, protože všechny programy musí spolupracovat, aby pracoval správně. Jakmile úloha vstoupí do CPU, setrvává tam tak dlouho, dokud neskončí nebo se dobrovolně CPU nevzdá. Nepreemptivní multitasking má minimální nebo žádnou možnost úlohu, zastavit, přerušit, odstranit a dát tak příležitost jiné úloze.

V případě, že aktivní proces například čeká na stisk klávesy, uvolní sám procesor a může být spuštěn proces jiný. Poté co tento proces zavolá přerušovací službu, vrátí se procesor ke zpracovávání původního procesu. Proces čekající, v tomto případě na stisk klávesnice, se nazývá proces na popředí, procesy na pozadí jsou pak ty, které využívají procesorový čas, který by jinak byl promarněn. Takovýto multitasking byl použit např. u Windows 3.x, Windows 95 pro 16.bit aplikace, MAC OS.



Obrázek 1 – Příklad nepreemptivního multitaskingu [8]

Na obrázku 1 je znázorněna ukázka kooperativního multitaskingu. *Task*[5] *0* skončil a plánovač spustil *Task 1*. Uprostřed vykonávání *Task 1* přichází požadavek na vykonání *Internal Interrupt* (např. uživatel stiskl tlačítko). Obsluha tohoto přerušení však nemůže být vykonána, dokud procesor zpracovává *Task 1*. Čas mezi okamžikem, kdy přišel požadavek na obsluhu přerušení a kdy k tomu skutečně došlo nelze předvídat a chování systému je časově nedeterministické[6].

[5] Task – anglicky úloha.
[6] Deterministický - vlastnost procesu, jehož každý stav je určen předcházejícím.

Kooperativní multitasking má všechny výhody, které poskytuje systém s neomezeným přepínáním programů. Navíc lépe využívá procesor. Vyznačuje se ale některými závažnými nevýhodami.

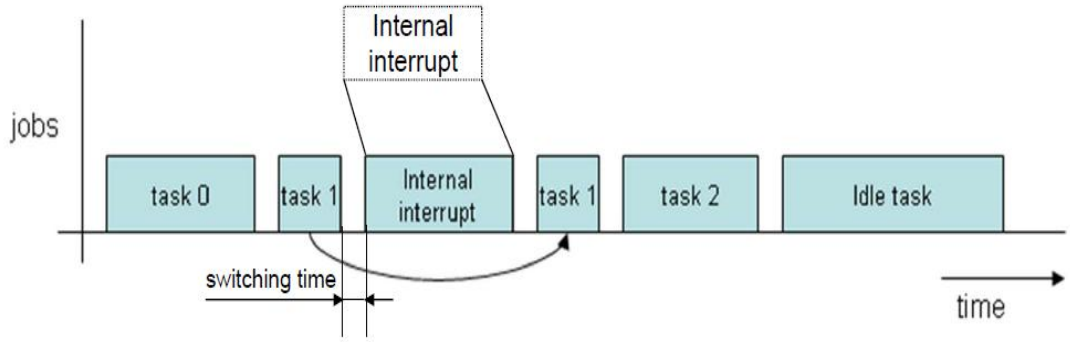
- Zpomalení procesu na popředí se odvíjí od toho, jak je proces běžící na pozadí naprogramován.
- Nelze použít pro realizaci paralelních úloh.
- Může dojít k situaci, že přerušovací služba procesu na pozadí nebude nikdy zavolána (nekonečná smyčka). Dojde pak k zablokování nejen tohoto procesu, ale celého systému. S tím je samozřejmě spojena velmi reálná možnost ztráty dat ostatních spuštěných aplikací. Kooperativní multitasking tedy není a nemůže být z tohoto hlediska bezpečný.
- Programování aplikací pro kooperativní multitasking je omezeno množstvím konvencí. Veškerá časově náročnější činnost programu musí být rozdělena na kratší úseky oddělené voláním přerušení. To vedlo k tomu, že řada firem označila své programy "foreground only" čímž je tento systém degradován pouze na systém s neomezeným přepínáním. [8], [10], [15]

1.4 Preemptivní multitasking

Preempce je akt dočasného přerušení prováděného úkolu pomocí počítačového systému, se záměrem odložení provádění úkolu na pozdější dobu. Taková technika je také známa jako přepínání kontextu.

Jednoduše řečeno preemptivní multitasking vyžaduje použití přerušovacího mechanismu, který pozastaví aktuálně prováděný proces a vyvolá plánovač úloh, který určí, jaký proces by se měl vykonat jako další. Proto všechny procesy dostanou vždy určité množství procesorového času. V preemptivním multitaskingu, může i jádro operačního systému vydat podnět k přepnutí kontextu a dodržet tak prioritní omezení pro plánování. Operační systém tedy umožňuje přerušení kteréhokoli procesu na kterémkoli místě, aniž by proces sám do toho musel nějakým způsobem zasahovat.

Zpracovávání procesů se velmi rychle střídá (řádově desítky až stovky milisekund) a tak vzniká dojem paralelního zpracování. Velikost intervalu, po jehož uplynutí dochází ke střídání procesů, se nazývá časové kvantum (time-slice). Po vypršení časového kvanta tedy dojde k zastavení jednoho procesu a spuštění dalšího. To také umožňuje systému rychle se vypořádat s důležitými vnějšími událostmi, jako jsou příchozí data, která by mohla vyžadovat okamžitou pozornost jiného procesu.



Obrázek 2 – Příklad preemptivního multitaskingu [8]

V jakémkoliv konkrétním čase, lze procesy rozdělit do dvou kategorií. Ty, které čekají na vstup či výstup (tzv. "I/O bound[7]"), a ty, které plně využívají procesoru ("CPU bound[8]"). Jak již bylo popisováno v kapitole 1.1, v dřívějších systémech procesy při čekání na požadovaný vstup (jako například disk, klávesnici nebo síťový vstup) musely čekat. Během této doby, proces nevykonával užitečnou práci, ale stále udržoval úplnou kontrolu nad procesorem. S příchodem přerušení a preemptivního multitaskingu, mohou být tyto „I/O bound" procesy blokovány do doby, než budou potřebná data k dispozici a jiné procesy tak mohou mezitím využívat CPU. Jakmile jsou požadovaná data k dispozici, generuje se opět přerušení a blokované procesy mají zaručen včasný návrat k realizaci úkolu.

Dnes, téměř všechny operační systémy podporují preemptivní multitasking, včetně aktuálních verzí systému Windows, Mac OS, GNU / Linux, iOS a Android. Některé z dřívějších operačních systémů pro běžné uživatele, které podporovaly preemptivní multitasking, byly Sinclair QDOS (1984) a Amiga OS (1985). Oba tyto systémy běžely na mikroprocesorech Motorola 68000 bez správy paměti.

I když byly techniky multitaskingu původně vyvinuty, aby umožnili více uživatelům sdílet jeden stroj, brzy se ukázalo, že multitasking je užitečný, bez ohledu na počet uživatelů jednoho PC. Mnoho operačních systémů, od mainframů po PC používaných jedním uživatelem a řídicích systémů nepotřebující zásah uživatele, prokázaly užitečnost podpory multitaskingu z různých důvodů. Multitasking umožňuje jedinému uživateli spouštět více aplikací současně, nebo běh procesů "na pozadí" a přitom je zachována kontrola nad počítačem. [8], [10], [15]

[7] Bound angl. vázán. „I/O bound" – např. proces vyžadující data z disku je odkázán na jeho rychlost a přístupovou dobu, tzn. je tak rychle vykonán jako je rychlý disk počítače.
[8] To samé jako „I/O bound", ale nyní je proces závislý na rychlosti procesoru.

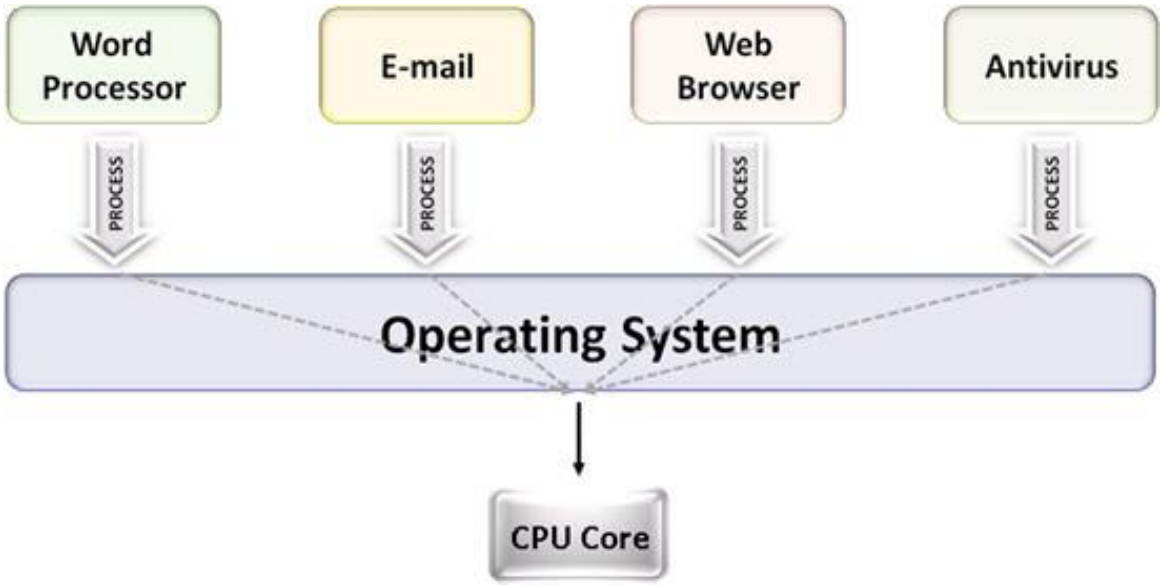
1.5 Rozdíl mezi multithreadingem a multitaskingem

Multitasking je tedy metoda, pomocí které několik úloh, také známé jako procesy, sdílí stejné výpočetní zdroje, jako je CPU. Díky multitaskingu OS, jako je například Windows XP, můžeme současně spustit více aplikací. Multitasking označuje schopnost operačního systému rychle přepínat mezi jednotlivými úlohami a vyvolat tak dojem, že různé aplikace vykonávají více činností současně.

Jak rychlosti CPU v průběhu doby stabilně rostou, nejen že aplikace běží rychleji, ale i operační systémy mohou přepínat mezi aplikacemi rychleji. To poskytuje lepší celkový výkon. Více úloh se může vykonávat na počítači najednou a jednotlivé aplikace mohou běžet rychleji.

1.5.1 Jednojádrový systém

V případě, že počítač je vybaven CPU pouze s jedním jádrem, může v daném okamžiku běžet pouze jedna úloha. Multitasking tento problém řeší tím, že naplánuje, která úloha může běžet v daném okamžiku a kdy se další čekající úloha dostane na řadu.

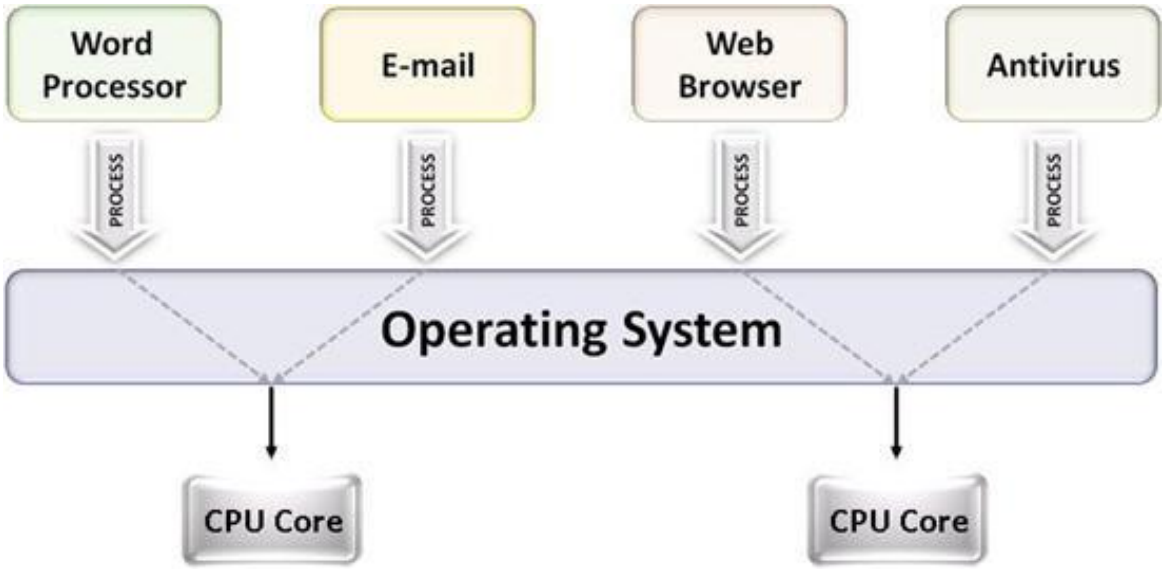


Obrázek 3 - Multitasking na jednojádrovém systému [3]

1.5.2 Vícejádrový systém

Při zpracování úloh na vícejádrovém systému, může multitasking OS skutečně vykonávat více úkolů současně. Více výpočetních jader pracuje nezávisle na různých úkolech.

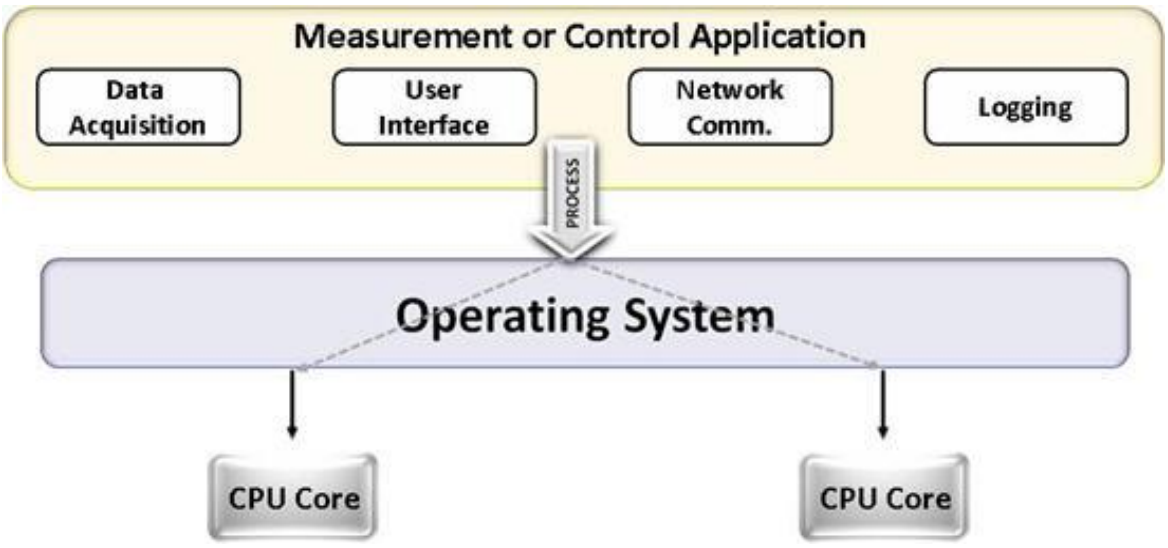
Například na dvoujádrovém systému máme čtyři aplikace, textový editor, emailového klienta, webový prohlížeč a antivirový software. Tyto čtyři aplikace se nyní dělí o dvě výpočetní jádra. Můžeme dělat více věcí najednou stejně tak jako na jednojádrovém systému, ale zde dochází k celkovému zlepšení výkonu aplikace.



Obrázek 4 - Multitasking na dvoujádrovém systému [3]

1.5.3 Multithreading

Multithreading rozšiřuje myšlenku multitaskingu do samotných aplikací. Tím je myšleno, že můžeme rozdělit určité operace v rámci jedné aplikace do jednotlivých vláken. Multithreading je tedy schopnost programu, sám sebe větvit. Program se dělí na tzv. vlákna (threads), která se vyvolávají dojem současného běhu. To má výhodu při provádění nějaké náročnější a dlouhotrvající operace (např. ukládání dlouhého souboru), kdy se aplikace „nezasekne“. Programátor zpravidla vytvoří jedno hlavní vlákno, které vytvoří všechna grafická okna a jedno nebo více vláken, která počítají.



Obrázek 5 - Multithreading na dvoujádrovém systému [3]

Zatímco u multitaskingu jsou jednotlivé procesy zcela oddělené (vlastní paměť, atd.), při multithreadingu všechna vlákna (jedné aplikace) sdílejí stejné systémové zdroje, jako jsou paměť, soubory, globální proměnné. Každé vlákno má svůj zásobník, to znamená, že automatické proměnné jsou pro každé vlákno zvlášť.

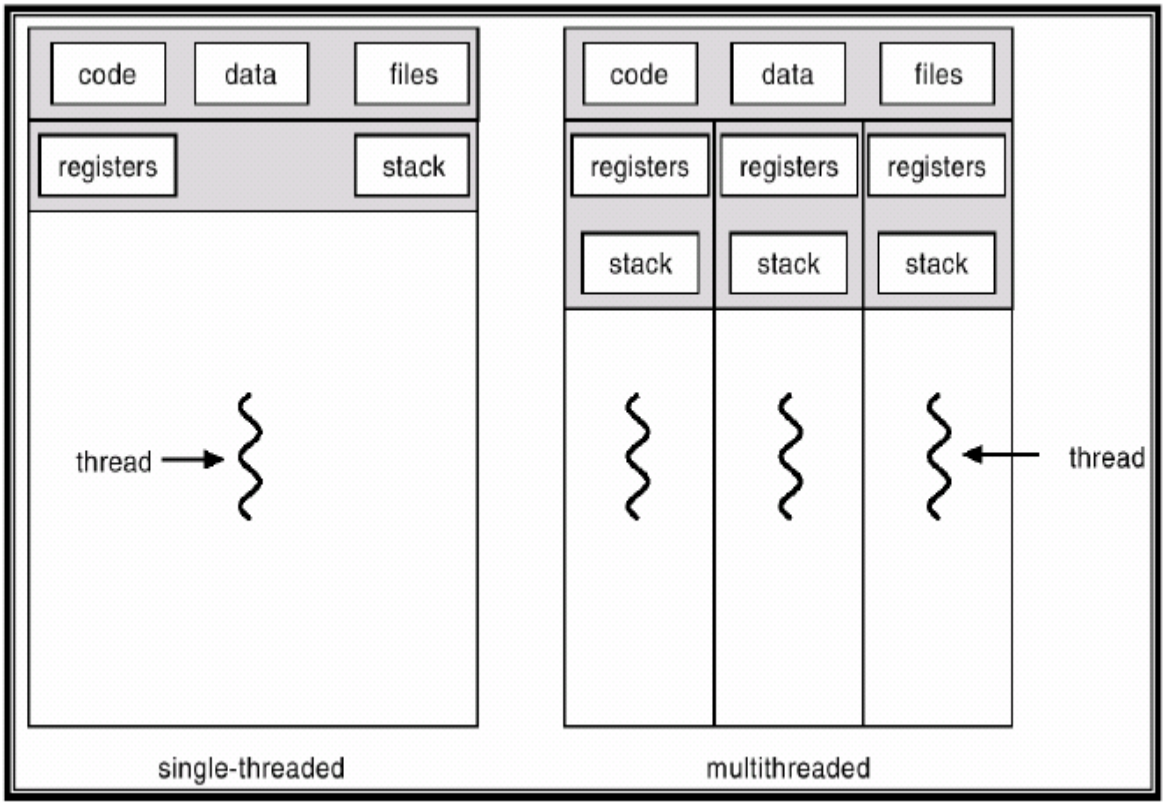
Ve Windows 3.x je každá běžící aplikace jedna úloha. Procesor byl přidělován těmto úlohám pomocí kooperativního multitaskingu a spoléhá na aplikace, že umožní předání procesorového času jiné úloze. Ve Win32 se už nemluví o úlohách, ale o procesech a vláknech. Proces se může skládat z jednoho či více vláken. Proces a vlákna jsou přerušovány preempcí. Ve Win32 proces vzniká jako jedno vlákno, které může vytvořit vlákna další běžící na pozadí. Vláknům je procesorový čas předáván zcela nezávisle, jako samostatným programům. [3], [13], [15]

2 Vlákna

2.1 Co to jsou vlákna

V informatice je vlákno (anglicky thread) nejmenší sekvence naprogramovaných instrukcí. Pomocí vláken se snižuje režie operačního systému při změně kontextu, který je nutný pro zajištění multitaskingu nebo se využívají při masivních paralelních výpočtech. Vlákno je „odlehčený“ proces, někdy také označován jako „podproces“. Implementace vláken a procesů se v různých operačních systémech liší, ale ve většině případů je vlákno obsažené v procesu. Zatímco běžné procesy jsou navzájem striktně odděleny, vlákna sdílí nejen společný paměťový prostor, ale i další struktury. Zejména vlákna jednoho procesu sdílí kódu procesu a jeho kontext (hodnoty, na které proměnné procesu odkazují v daném okamžiku).

Každá spuštěná aplikace má alespoň jeden proces a každý proces má alespoň jedno vlákno, ve kterém počítá. Dříve platilo, že proces měl jen jedno vlákno. Operační systém, který vlákna nepodporuje, má technicky jedno vlákno na každý proces, zatímco při podpoře vláken je možné v rámci jediného procesu vytvořit mnoho vláken. Vlákna usnadňují díky sdílené paměti vzájemnou komunikaci, což však přináší další komplikace v podobě synchronizace. [1], [5], [9]



Obrázek 6 - Příklad procesu s jedním vláknem a vícevláknového procesu [9]

2.2 Charakteristika

Rozdíly mezi procesy a vlákny:

- Procesy jsou obvykle nezávislé, zatímco vlákna existují jako podmnožiny procesu.
- Procesy uchovávají mnohem více informací než vlákna, ale více vláken v rámci procesu sdílejí stav procesu a stejně tak paměť a další zdroje.
- Procesy mají oddělené adresní prostory, zatímco vlákna sdílejí adresový prostor.
- Procesy komunikovat pouze prostřednictvím mechanismů nabízených systémem pro komunikaci mezi procesy.

Praktický rozdílem mezi vlákny a procesy je kromě sdílené paměti (která některé věci usnadňuje a jiné naopak ztěžuje) režie při přepínání:

- Přepnutí mezi vlákny bývá výrazně rychlejší, neboť vlákna sdílejí stejnou paměť a uživatelská práva svého mateřského procesu a není je třeba při přepínání měnit.
- V některých případech není třeba při přepínání vláken volat jádro OS.
- Rychlejší může být i vytváření a rušení vlákna.
- Vlákno také spotřebuje méně paměti, což je důležité pro aplikace, které používají stovky nebo více vláken. [5], [9], [14]

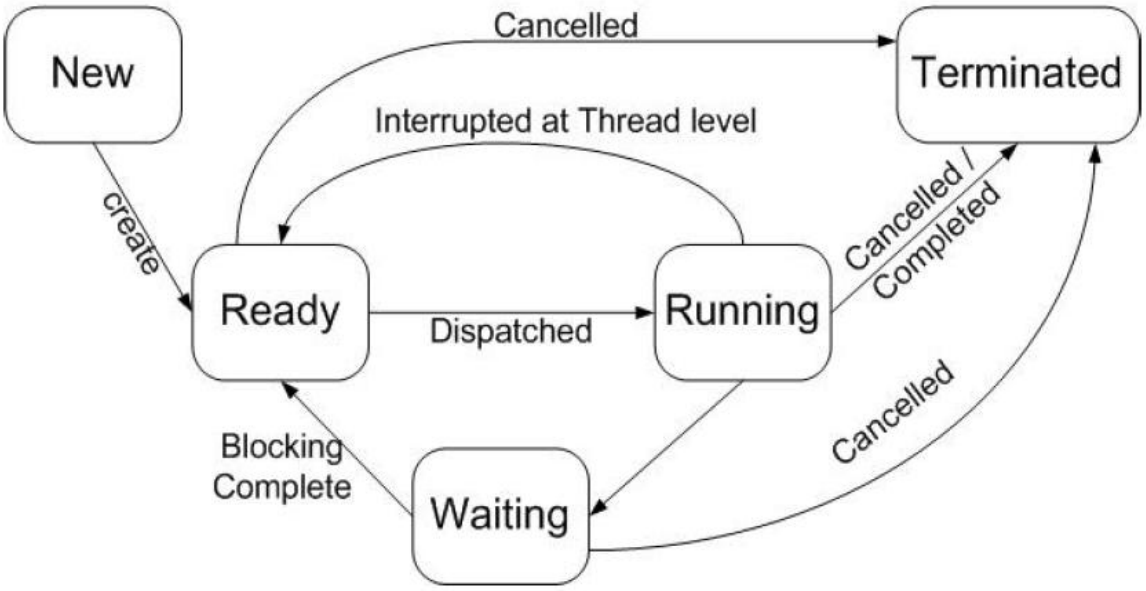
2.3 Podpora vláken v operačním systému

Z hlediska spravování vláken OS můžeme vlákna rozdělit na:

- vlákna na uživatelské úrovni (ULT),
- vlákna na úrovni jádra (KLT),
- kombinace KLT a ULT.

2.3.1 Vlákna na uživatelské úrovni (ULT)

Správu na úrovni uživatele provádí vláknová knihovna (thread library), jádro operačního systému o jejich existenci neví a přepojování mezi vlákny nepožaduje provádění funkcí jádra. To znamená, že aplikace má své specifické přepínání vláken a jeho plánování a OS vidí pouze jeden proces. Programátor má možnost zvolit si nejvhodnější plánovací strategii a algoritmus sám. Stavový diagram pro vlákna vypadá podobně jako pro procesy (proces má stavy new, running, waiting, ready, terminated):



Obrázek 7 - Stavový diagram vlákna na uživatelské úrovni [9]

Uživatelská vlákna jsou obvykle rychlá a mají nízké náklady na řízení (vláknová knihovna je v adresovém prostoru procesu a je tak volání funkcí této knihovny rychlé). Tato metoda má své výhody i nevýhody. Např. když vlákno čeká na I/O, celý proces je blokován, protože operační systém vidí proces, jak čeká na I/O. Pokud má tedy aplikace velké množství vláken, které by mohly blokovat často celý proces, pak vlákna na uživatelské úrovni nemusí být tou nejlepší volbou.

Jak již bylo uvedeno dříve v kapitole, pro provozování vláken na uživatelské úrovni je třeba speciální knihovna, která musí umožňovat:

- rušení a vytváření vláken,
- předávání dat a zpráv mezi vlákny,
- plánování běhu vláken,
- uchovávání a obnova kontextu vláken.

Mezi **výhody** vláken na uživatelské úrovni můžeme zařadit:

- nezávislost na podpoře vláken v OS,
- přepínání mezi vlákny je nezávislé na OS (může být rychlejší),
- výrazně rychlejší tvorba a přepínání vláken,
- plná kontrola procesu nad správou běhu vláken.

Mezi **nevýhody** potom tedy patří:

- volání služby (např. I/O) jedním vláknem zablokuje všechna vlákna procesu,

- nutnost dodatečného programování (řízení vláken programátorem),
- pokud jádro přiděluje procesor pouze procesům, nemohou dvě vlákna téhož procesu běžet současně, i když systém obsahuje více procesorů. [5], [9], [14]

2.3.2 Vlákna na úrovni jádra OS (KLT)

Vlákna jádra jsou implementovány na úrovni operačního systému, a proto má vyšší režijní náklady a je pomalejší, kvůli systémovým voláním. OS má za úkol vytváření, rušení a plánování všech vláken jádra a nemusí tak docházet k blokování celého procesu (např. pokud jedno vlákno volá službu, přejde pouze do stavu čekání (waiting) a nezablokuje tak celý proces).

Výhody vláken na úrovni jádra jsou:

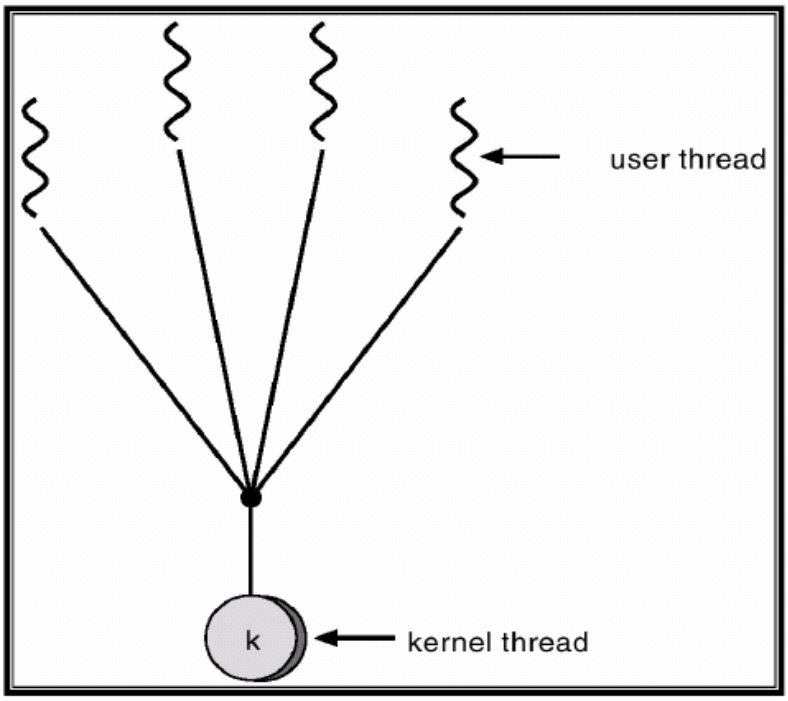
- Volání systému neblokuje ostatní vlákna téhož procesu.
- Jeden proces může využít více procesorů.
- Tvorba, rušení a přepínání mezi vlákny je levnější než mezi procesy.
- Programy jádra mohou mít také vícevláknový charakter.

Nevýhody:

- Správa je nákladnější než u čistě uživatelských vláken. [5], [9], [14]

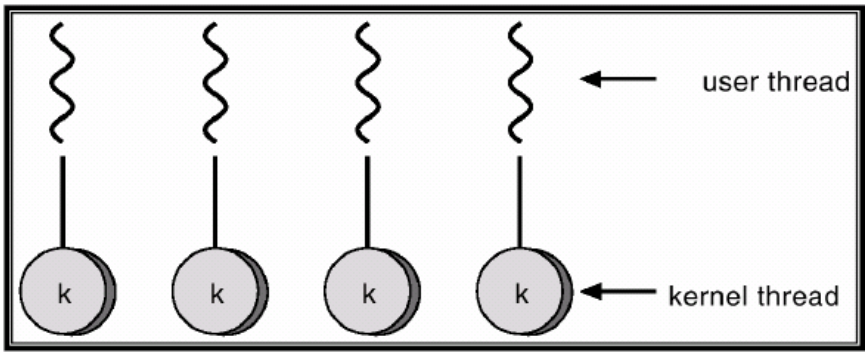
2.3.3 Kombinace ULT a KLT

Vlákna jak na uživatelské úrovni i na úrovni jádra mají své výhody i nevýhody, takže pomocí kombinace těchto metod dostaneme to nejlepší z obou metod. Většina implementací vláken je kombinací těchto způsobů. Přiřazování uživatelských vláken k systémovým se děje automaticky (bez programátora) anebo programátor může nastavit počet vláken na úrovni jádra. Společným přístupem je sdružit několik uživatelských vláken na několik vláken jádra.



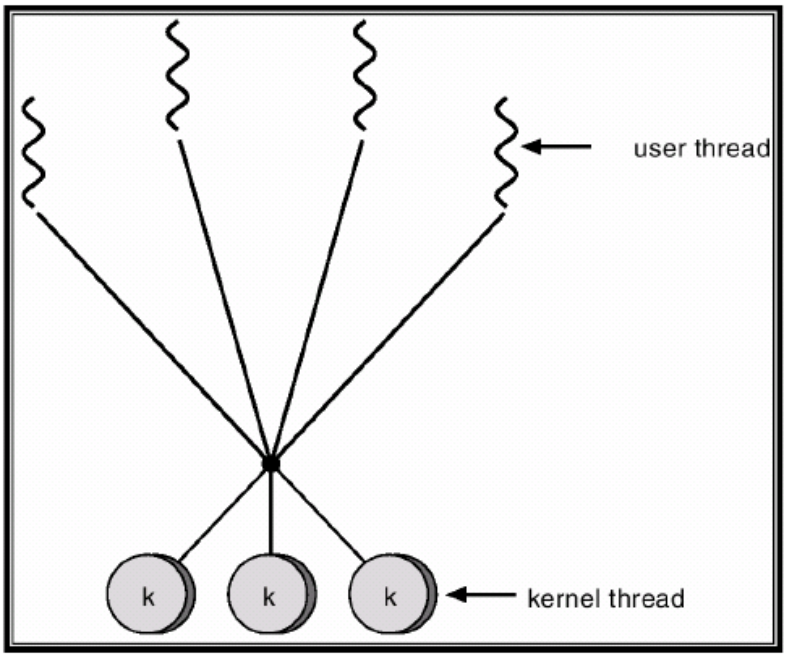
Obrázek 8 - Kombinace m:1 (many-to-one) [9]

Kombinace více uživatelských vláken k jednomu vláknu jádra (Obrázek 8) je to samé jako multithreading na uživatelské úrovni. Mnoho vláken na uživatelské úrovni je mapováno do jednoho vlákna jádra. Všechno řízení vláken se odehrává v uživatelském prostoru, takže je to efektivní, ale ve skutečnosti trpí problémem, kdy jedno vlákno blokuje celý proces.



Obrázek 9 - Kombinace 1:1 (one-to-one) [9]

Kombinace jednoho uživatelského vlákna k jednomu vláknu jádra (Obrázek 9) je to samé, jako když jsou všechna vlákna na úrovni jádra. Každé uživatelské vlákno je přiřazeno právě k jednomu vláknu jádra. Pokud u tohoto modelu je jedno vlákno blokované, může běžet jiné. Zároveň s tím však souvisí vyšší náklady na režii.



Obrázek 10 - Kombinace m:n (many-to-many) [9]

Kombinace více uživatelských vláken k více vláknům jádra (Obrázek 10) je běžnou metodou pro implementaci vláken. Uživatelská knihovna řídí vše, např. která vlákna jsou na úrovni jádra systému, která nejsou a také řídí přepínání, atd. Knihovna se snaží řídit vlákna tak, aby nebyla všechna vlákna jádra zablokována současně, tím, že přiřadí čekající uživatelské vlákno jinému vláknu jádra. Tato metoda má tedy za úkol zajistit, aby vždy alespoň jedno uživatelské vlákno bylo spustitelné kdykoliv. Navíc je poměrně složitá a režijní náklady jsou v operačním systému i u aplikací. [5], [9], [14]

2.4 Kdy se vlákna používají

- **Obsluha periferií**
  - U některých zařízení je třeba periodicky testovat stav hardware.
  - Vláknu pak nemusí zbývat mnoho času na obsluhu uživatelského rozhraní.
  - Jedno vlákno je použito pro komunikaci s uživatelem a druhé obsluhuje hardware
- **Síťová komunikace**
  - Jedno vlákno akceptuje příchozí komunikace.
  - Jedno vlákno odesílá data.
  - Jedno vlákno zpracovává data.
- **Vyvolání dojmu rychlé odezvy programu**
  - Práce s velkým objemem dat uložených v databázi.
  - Hlavní vlákno pouze obsluhuje uživatelské rozhraní, další pracuje s databází.
- **Urychlení výpočtu**
  - Lze-li spustit na víceprocesorovém stroji kooperující vlákna na několika procesorech.
- **Vhodné pro architekturu aplikace**
  - Např. pro simulace, kdy jedno vlákno počítá vlastní simulaci.

    - Další vlákno periodicky vzorkuje stav simulace a zobrazuje ho.
    - Primární vlákno obsluhuje uživatelské rozhraní.
- **Efektivita**
    - Některé aplikace jsou ze své podstaty nevhodná pro jednovláknovou architekturu.
    - Použití vláken může vést k výraznému zpřehlednění programového kódu.
    - Každý OS má maximální strop na počet vláken, kdy je plánování procesu stále ještě efektivní.

Vlákna je výhodné použít, pokud aplikace splňuje některé z následujících kritérií:

- Je složena z nezávislých úloh.
- Může být blokována po dlouhou dobu.
- Obsahuje výpočetně náročnou část.
- Musí reagovat na asynchronní[9] události.
- Obsahuje úlohy s nižší nebo vyšší prioritou než zbytek aplikace [5], [14]

2.5 Nevýhody vláken

- Vlákna by se měla používat jen tam, kde je to opravdu nutné, protože tvorba vláken na určitých platformách je poněkud zdlouhavá operace.
- K vyřešení tohoto problému se u vícevláknových aplikací používá mechanismus sdružování vláken.
- Každé vytvořené vlákno vytváří v paměti vlastní zásobník, do kterých jsou ukládány mezivýsledky, stavy proměnných, adresy apod.
- I když je teoreticky možné vytvářet obrovský počet vláken, maximální počet vláken je omezen platformou.
- Zvýšená složitost kódu.
- Daleko složitější sledovat tok programu.
- Sdílení prostředků, za které je většinou odpovědný programátor.
- K řízení sdílení prostředků se používá synchronizace, zaručuje, že stav dat nebude změněn z více vláken zároveň. [5], [14]

[9] Asynchronní = nesoudobý, nesoučasný, nesynchronizovaný.

2.6 Problémy vícevláknových aplikací

Hlavní problémy vícevláknových aplikací souvisí se synchronizací. Mezi tyto problémy patří:

- Uváznutí (deadlock) je situace, kdy úspěšné dokončení první akce je podmíněno předchozím dokončením druhé akce, přičemž druhá akce může být dokončena až po dokončení první akce.
- Souběh (race conditions) je přístup více vláken ke sdíleným proměnným a alespoň jedno vlákno nevyužívá synchronizačních mechanismů. Vlákno čte hodnotu, zatímco jiné vlákno zapisuje. Zápis a čtení nejsou atomické[10] a data mohou být neplatná.
- Vyhladovění (starvation) je stav, kdy jsou vláknu neustále odepírány prostředky. Bez těchto prostředků program nikdy nedokončí svůj úkol. [5], [14]

[10] Atomické = nedělitelné.

3 Programování vláken v různých jazycích

3.1 Vlákna v Javě

Java je objektově orientovaný programovací jazyk, který vyvinula firma Sun Microsystems. Je jedním z nejpoužívanějších programovacích jazyků na světě.

Chceme-li vytvořit vlákno, můžeme využít následující tři způsoby:

- Vytvoření instance třídy *Thread* nebo jejího potomka.
- Implementování rozhraní *Runnable*.
- Použitím anonymních vnitřních tříd.

3.1.1 Třída *Thread*

Každé vlákno v Javě je instancí třídy *Thread* (z balíku `java.lang`) nebo jejího potomka. Tato třída definuje základní metody, jako je spuštění, zastavení a ukončení vlákna. Jednoduché vlákno se naprogramuje tak, že se vytvoří potomek třídy *Thread*, který definuje metodu:

```
public void run()
```

Tato metoda obsahuje kód, který bude prováděn paralelně. Spuštění vlákna se provede zavoláním metody `start()` (metoda `run()` se přímo nevolá).

Příklad vytvoření potomka třídy *Thread*:

```
// Třída definující vlákno
class MojeVlakno extends Thread {
   MojeVlakno (String nazev) {
      super(nazev);
   }

   public void run() {
      for (int i=0; i<5; i++) {
         System.out.println("Vlakno: " + getName());
         yield();
      }
   }
}


// Hlavní program
public class Vlakna {
   public static void main(String[] args) {
      MojeVlakno v1 = new MojeVlakno ("v1");
      MojeVlakno v2 = new MojeVlakno ("v2");
      v1.start();
      v2.start();
   }
}
```

Nejdůležitější metody definované ve třídě `Thread` jsou:

- `public static Thread currentThread()` - vrací referenci na právě běžící vlákno,
- `public String getName()` - vrací jméno vlákna,
- `public int getPriority()` - vrací prioritu vlákna,
- `public void join()` - čeká na ukončení vlákna,
- `public void resume()` - probudí vlákno „odstavené“ metodou `suspend()`,
- `public void run()` - jádro vlákna,
- `public void start()` - zahájí běh vlákna, tj. provádění metody `run()`,
- `public static void sleep(long ms)` - uspí (dočasně zastaví) vlákno na zadaný počet milisekund. Pak jej probudí a vlákno opět pokračuje v běhu,
- `public setPriority(int priorita)` - nastaví prioritu vlákna,
- `public void suspend()` – „odstaví“ vlákno,
- `public static void yield()` - umožní běh jiného vlákna. [4], [6]

3.1.2 Rozhraní *Runnable*

Vytvoření potomka třídy *Thread* je jednoduché, ovšem pokud budeme chtít třídu, která je už potomkem jiné třídy (např. *JFrame* v GUI aplikacích) a kterou chceme zároveň spouštět jako vlákno, musíme právě využít tohoto rozhraní. V tomto případě budeme tedy implementovat rozhraní *`Runnable`* (vyřešení problému vícenásobné dědičnosti v Javě).

Implementující třída:

- musí implementovat metodu run(),
- není zatím vlákno. Pro spuštění vlákna potřebuje třídu *Thread* k vytvoření instance vlákna.

Příklad použití rozhraní *Runnable*:

```
public class Animace extends java.applet.Applet
implements Runnable {
   Thread animator = null;
   int xpos = 0;

   public void init() {
      animator = new Thread(this);           // (1)
      animator.start();
   }

   public void run() {                       // (2)
      while(animator != null) {              // (3)
         repaint();                          // (4)
         try {
            Thread.sleep(80);                // (5)
         } catch(InterruptedException e) {}
      }
   }

   public void paint(java.awt.Graphics g) {  // (6)
      g.drawRect(xpos, 0, 20, 20);
      xpos = (xpos+1) % 100;
   }
}
```

Applet má „na pozadí“ přehrávat animaci. Vytvoří proto vlákno (1) konstruktorem `Thread(Runnable r)`, kterému předá odkaz na sebe (instanci implementující rozhraní `Runnable`) (2). Metoda `run()` vykreslí fázi animace voláním metody `repaint()` (4), uspí se na 80 ms (5), a cyklus se opakuje, čímž vzniká animace, vykreslování zajišťuje metoda `paint()` (6) volaná prohlížečem na žádost metody `repaint()` (4). [6]

3.2 Vlákna v C#

C# je vysokoúrovňový objektově orientovaný programovací jazyk vyvinutý firmou Microsoft zároveň s platformou .NET Framework. Microsoft založil C# na jazycích C++ a Java (a je tedy nepřímým potomkem jazyka C, ze kterého čerpá syntaxi).

3.2.1 Třída *Thread*

Vlákna jsou vytvářena pomocí konstruktoru třídy *Thread* předávajícího delegáta *ThreadStart*. Ten označuje metodu, kde by měla začít práce vlákna. Takhle vypadá deklarace delegáta *ThreadStart*:

```
public delegate void ThreadStart();
```

Poté následuje zavolání metody `Start()` na instanci vlákna, tato akce uvede vlákno do provozu. Funguje až do chvíle, kdy zpracuje všechny příkazy, které jsme mu zadali. Když vše dokončí, Garbage Collector ho odklidí a uvolní paměť.

Následující program vytvoří dvě paralelní vlákna, která vždy vypíšou své jméno a pak čekají zadaný počet milisekund.

```
using System;
using System.Threading;

public class MojeVlakno
{
        string jmeno;
        int interval;

        public MojeVlakno(string jmeno, int interval)
        {
                this.jmeno = jmeno;
                this.interval = interval;
        }

        public void Run()
        {
                while( true ) {
                        Console.WriteLine("Vlakno {0}", jmeno);
                        Thread.Sleep(interval);
                }
        }

        public static void Main()
        {
                MojeVlakno v1 = new MojeVlakno("v1", 500);
                MojeVlakno v2 = new MojeVlakno("v2", 1000);
```

```
                Thread t1 = new Thread(new ThreadStart(v1.Run));
                Thread t2 = new Thread(new ThreadStart(v2.Run));
                t1.Start();
                t2.Start();
                t1.Join();
                t2.Join();
        }
}
```

Nejdůležitější metody definované ve třídě *Thread* jsou:

- `public void Abort()` - volání této metody obvykle ukončí vlákno,
- `public static AppDomain getDomain()` - vrací aktuální doménu, ve které je spuštěno aktuální vlákno,
- `public void Join()` - čeká na ukončení vlákna,
- `public void Start()` - zahájí běh vlákna, tj. provádění metody `run()`,
- `public static void Sleep(int ms)` - uspí (dočasně zastaví) vlákno na zadaný počet milisekund. Pak jej probudí a vlákno opět pokračuje v běhu,
- `public static bool Yield()` - umožní běh jiného vlákna. [2], [7]

3.3 Vlákna v Qt frameworku

Qt je framework programovacího jazyka C++, ale podporuje spoustu jiných jazyků jako např. Python, Java, Perl, C#. Qt je multiplatformní framework pro tvorbu aplikací s grafickým uživatelským rozhraním, ale také konzolových aplikací např. pro servery.

3.3.1 Třída *QThread*

Třída *QThread* poskytuje platformově nezávislý způsob pro spravování vláken.

Jednou z možností jak vytvořit vlákno je vytvoření instance třídy *QThread* a třídu která dědí z *QObject* obsahující kód, který chceme spustit v jiném vlákně, pomocí metody `moveToThread(QThread*)` přiřadíme dané instanci *QThread*.

```
class Worker : public QObject
 {
     Q_OBJECT

 public slots:
     void doWork() {
         ...
     }
 };

 void MyObject::putWorkerInAThread()
 {
     Worker *worker = new Worker;
     QThread *workerThread = new QThread(this);

     connect(workerThread, SIGNAL(started()), worker, SLOT(doWork()));
     connect(workerThread, SIGNAL(finished()), worker,
             SLOT(deleteLater()));
     worker->moveToThread(workerThread);
     workerThread->start(); // Spustí vlákno
 }
```

Další možností tvorby vláken je vytvořit třídu, která dědí z třídy *QThread* a reimplementovat její chráněnou metodu `run()`, která se provede po spuštění vlákna metodou `start()`.

```
class MyThread : public QThread
 {
 Q_OBJECT

  ...

 protected:
     void run();

  ...
 };
```

Nejdůležitější metody definované ve třídě `Thread` jsou:

- `public void exit(int returnCode = 0)` - volání této metody obvykle ukončí vlákno,
- `public bool isFinished()` - vrací *true* pokud vlákno dokončilo činnost,
- `public bool isRunning()` - vrací *true* pokud vlákno běží,
- `public Priority priority()` - vrací prioritu běžícího vlákna,
- `public void setPriority(Priority priority)` – nastavuje prioritu běžícímu vláknu,
- `public static QThread* currentThread()` – vrací ukazatel na QThread,
- `public static void yieldCurrentThread()` – pozastaví aktuální vlákno a umožní běh jinému vláknu,
- `public bool wait(unsigned long time = ULONG_MAX)` – zablokuje vlákno na zadanou dobu v milisekundách.

Sloty:

- `public void quit()` – ekvivalentní k volání `QThread::exit(0)`,
- `public void start(Priority priority = InheritPriority)` - zahájí běh vlákna, tj. provádění metody `run()`,
- `public void terminate()` – terminuje provádění vlákna.

Signály:

- `void finished()` – signál je emitován, když vlákno skončí,
- `void started()` - signál je emitován, když vlákno zahájí vykonávání,
- `void terminated()` - signál je emitován, když je vlákno terminováno. [12]

3.4 Srovnání

Tvorba vláken v Javě, C# a Qt frameworku je velice podobná. Pokaždé máme k dispozici třídu, která zprostředkovává metody pro práci s vlákny. V jazyce Java se velice často používá rozhranní *Runnable,* kde stačí pouze implementovat metodu `run()`. V C# lze zase pohodlně předat třídu instanci třídy *Thread,* kterou chceme spustit v jiném vlákně, pomocí delegáta *ThreadStart*. V Qt frameworku je práce s vlákny velice podobná jako v jazyce Java, navíc nabízí možnost využívat signály a sloty. Qt framework má navíc velice dobře zpracovanou dokumentaci.

4 Aplikace

Zadáním bakalářské práce bylo navrhnout objektového správce vláken (threads), který bude spravovat základní operace vláken:

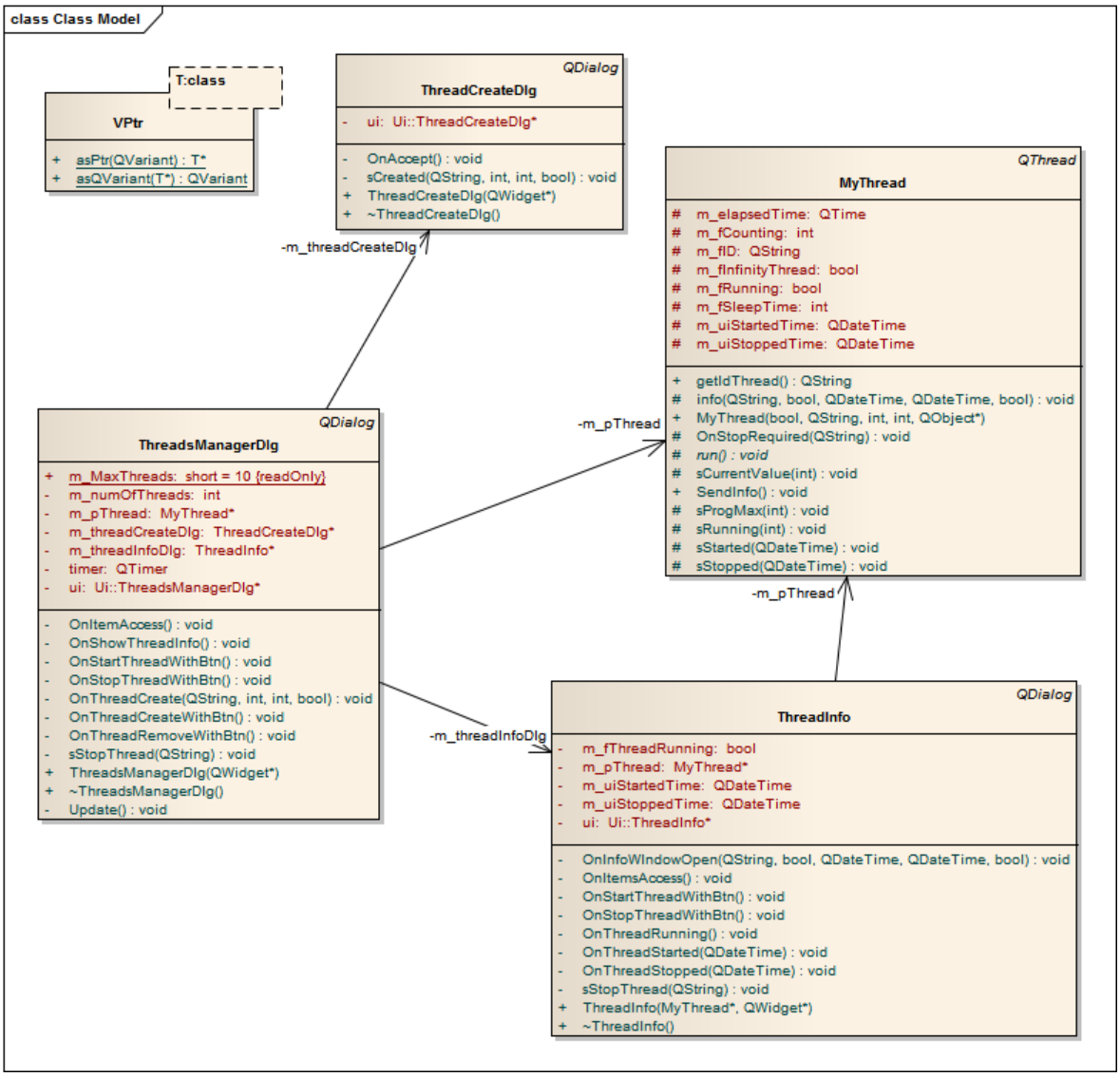
- spuštění vláken,
- ukončení vláken,
- monitorování vláken.

Pro otestování funkčnosti aplikace se mělo použít 10 až 50 vláken pro správu. Některá vlákna měla běžet neustále (v nekonečné smyčce), jiná se měla ukončit po provedení úlohy. Aplikace měla poskytovat monitorování stavu všech vláken.

Aplikace je napsána v jazyce C++ s použitím Qt frameworku. Jako vývojové prostředí je použit Qt Creator. Aplikace Threads Manager má grafické uživatelské rozhraní navržené v Qt Designeru.

Qt je použito z důvodu snadné přenositelnosti na jiné platformy, dalším důvodem je pohodlné programování vláken s použitím třídy *QThread*.

4.1 UML diagram



Obrázek 11 - UML diagram tříd

UML diagram na obrázku 11 zobrazuje všechny třídy aplikace. Hlavním oknem aplikace je *ThreadsManagerDlg*. Třída *ThreadInfo* zobrazuje informace o vybraném vlákně, třída *ThreadCreateDlg* slouží k zobrazení dialogového okna pro vytvoření vlákna a všechna vlákna jsou instancí třídy *MyThread*. Šablona třídy *VPtr* slouží k převodu ukazatele vlákna na datový typ *QVariant.*

4.2 Popis tříd

4.2.1 ThreadsManagerDlg

Tato třída je hlavním oknem aplikace. Hlavní činností je ukládání ukazatelů na vlákna do komponenty *QListWidgetItem* a zároveň i odebírání ukazatelů z této komponenty. Následuje ukázka ukládání ukazatele do komponenty:

```
m_pThread = new MyThread(sInfiniteThread, sUniqueId, sThreadCounting,
sSleepTime, this);
QVariant v = VPtr<MyThread>::asQVariant(m_pThread);
QListWidgetItem *item = new QListWidgetItem(QString("ID vlakna:
                            %1").arg(sUniqueId), ui->listWidget);
item->setData(Qt::UserRole, v);
```

Na ukázce je vidět vytvoření instance třídy *MyThread*, přiřazení ukazatele m_pThread datovému typu *QVariant* pomocí šablony třídy *VPtr* a následně vložení do *QListWidgetu* pomocí konstruktoru třídy *QListWidgetItem*. Odebírání poté vypadá následovně:

```
QListWidgetItem *item = ui->listWidget->currentItem();
m_pThread = VPtr<MyThread>::asPtr(item->data(Qt::UserRole));
delete m_pThread;
int iPosition = ui->listWidget->currentRow();
ui->listWidget->takeItem(iPosition);
```

Po vytvoření a vložení vlákna do *QListWidgetu* se v této komponentě zobrazí ID uloženého vlákna. Po označení vlákna klikem myši, je možné vlákno spustit, zastavit, odebrat nebo si zobrazit informace o daném vlákně. Pokud dané vlákno běží, v *QListWidgetu* se podbarvuje žlutou barvou. Pro kontrolu, která vlákna běží je použit časovač z třídy *QTimer*:

```
QObject::connect(&timer, SIGNAL(timeout()), this, SLOT(Update()));
timer.start(300);
```

Po uplynutí doby 300 milisekund se cyklem projdou všechna vytvořená vlákna. Následně se podbarvují žlutou barvou, pokud běží a inkrementuje se počítadlo běžících vláken. Po skončení se nastaví text komponenty *QLabel* s počtem aktuálně běžících vláken:

```
ui->labelNumRunning->setText(QString::number(nRunning));
```

Pokud chceme spustit vybrané vlákno, stačí kliknout na příslušnou komponentu *QPushButton* a vlákno se spustí:

```
QListWidgetItem *item = ui->listWidget->currentItem();
m_pThread = VPtr<MyThread>::asPtr(item->data(Qt::UserRole));
if (m_pThread != 0) {
    QObject::connect(this, SIGNAL(sStopThread(QString)),m_pThread,
                     SLOT(OnStopRequired(QString)));
    m_pThread->start();
}
```

K třídě *ThreadsManagerDlg* je také připojen formulářový soubor threadsmanagerdlg.ui ve kterém je uloženo uživatelské prostředí hlavního okna, rozmístění komponent, jejich nastavení.

4.2.2 ThreadCreateDlg

Tato třída slouží pouze k získání informací o vytvářeném vlákně. Jedná se o modální okno s několika komponentami, jako jsou *QLineEdit*, *QCheckBox*, *QGroupBox*, *QLabel* a *QDialogButtonBox*. Pro zachycení signálu z *QDialogButtonBox* do slotu je použit tento connect:

```
connect(ui->buttonBox, SIGNAL(accepted()), this, SLOT(OnAccept()));
```

Pro zisk dat z komponenty *QLineEdit* je použita metoda `text()` a pro převod textových hodnot na datový typ int je použita metoda `toInt()`. Ukázka části kódu:

```
QString szID = ui->lineEditIDThread->text();
int szTCount = ui->lineEditThreadCounting->text().toInt();
int szSleep = ui->lineEditSleepTime->text().toInt();
```

Pro odeslání dat zpět do hlavního okna je emitován signál.

```
emit sCreated(szID, szTCount, szSleep, szInfinite);
```

4.2.3 ThreadInfo

ThreadInfo je další třídou pro dialogové okno. Toto okno má na starosti zobrazování informací o vybraném vlákně. Pomocí konstruktoru třídy se předává ukazatel na vlákno. Tato třída zachytává signály přímo z třídy *MyThread* a má tak přístup ke všem informacím o vlákně. Pro ukázku několik statických metod connect:

```
QObject::connect(this, SIGNAL(sStopThread(QString)),
                 m_pThread, SLOT(OnStopRequired(QString)));

QObject::connect(m_pThread, SIGNAL(sStarted(QDateTime)),
                 this, SLOT(OnThreadStarted(QDateTime)));
```

Tato třída má možnost dané vlákno zastavit nebo opět spustit. Aby nebylo možné znovu spustit již spuštěné vlákno, je třeba správně povolovat komponenty *QPushButton*.

```
ui->m_PbtnStart->setEnabled(!m_fThreadRunning);
ui->m_PbtnStop->setEnabled(m_fThreadRunning);
```

Pro grafické zobrazení stavu probíhajícího vlákna je použita komponenta *QProgressBar*.

```
QObject::connect(m_pThread, SIGNAL(sProgMax(int)),
          ui->m_PrgBar, SLOT(setMaximum(int)));
```

O vláknech se také zaznamenává datum a čas spuštění a ukončení vlákna. K tomu je použita třída *QDateTime*, která poskytuje funkce pro datum a čas. Pro výpis data a času do komponenty *QLabel* je potřeba nejdříve zformátovat datum pomocí metody toString:

```
QString szOut=QString(m_uiStartedTime.toString("hh:mm:ss dd.MM.yyyy"));
ui->m_LblStartedDate->setText(szOut);
```

4.2.4 MyThread

Tato třída je potomkem třídy *QThread* a tak bylo potřeba implementovat virtuální metodu `virtual void run();`. Emituje signály pro spuštění a zastavení vlákna, jestli vlákno stále běží a emituje signály s informacemi o daném vlákně jako je např. aktuální hodnota čítače v metodě `run()`.

Přetížená metoda run():

```
void MyThread::run()
{
// odstartovano
m_uiStartedTime = QDateTime::currentDateTime();
emit sStarted(m_uiStartedTime);

// funkcnost vlakna (pocitani)
int niCounter=0;
const int iMax=m_fCounting;
const int iSleepTime = m_fSleepTime;
m_elapsedTime.start();
// while promenna
m_fRunning=true;
while(m_fRunning)
{
    // ukonceni vlakna nebo znovu pocitani
    if(niCounter>iMax)
    {
        if(m_fInfinityThread==false)
        {
            m_fRunning=false;
            continue;
        }
        niCounter=0;
    }
    emit sCurrentValue(niCounter);
    emit sRunning(m_elapsedTime.elapsed()/1000);
    msleep(iSleepTime);
    niCounter++;
}
// konec behu vlakna
m_uiStoppedTime = QDateTime::currentDateTime();
emit sStopped(m_uiStoppedTime);
}
```

4.2.5 VPtr

Třída *VPTr* slouží k přetypování ukazatele určitého datového typu na ukazatel typu void* a následně vkládá jeho kopii do datového typu *QVariant*. Tento datový typ je potřeba pro uložení dat do třídy *QListWidgetItem*.

```
static QVariant asQVariant(T* ptr)
{
    return qVariantFromValue((void *) ptr);
}
```

Stejně tak umí získat data zpět z datového typu *QVariant* a přetypovat zpět na ukazatel datového typu jaký potřebujeme.

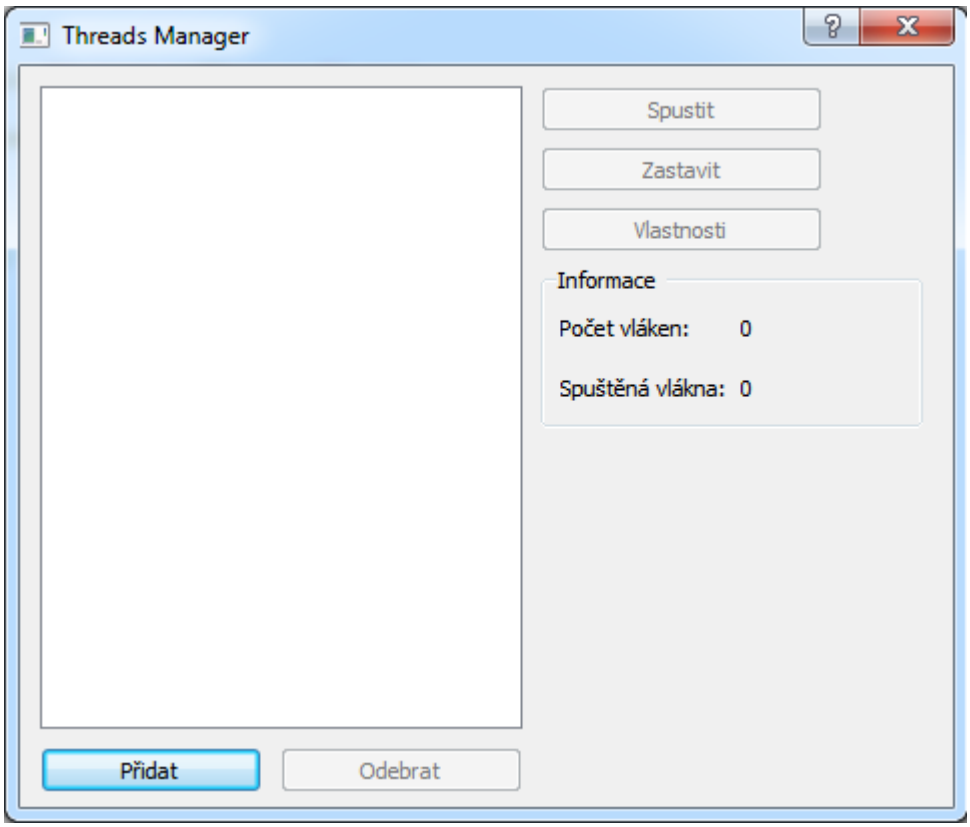
```
static T* asPtr(QVariant v)
{
    return (T *) v.value<void *>();
}
```

4.3 Funkce aplikace

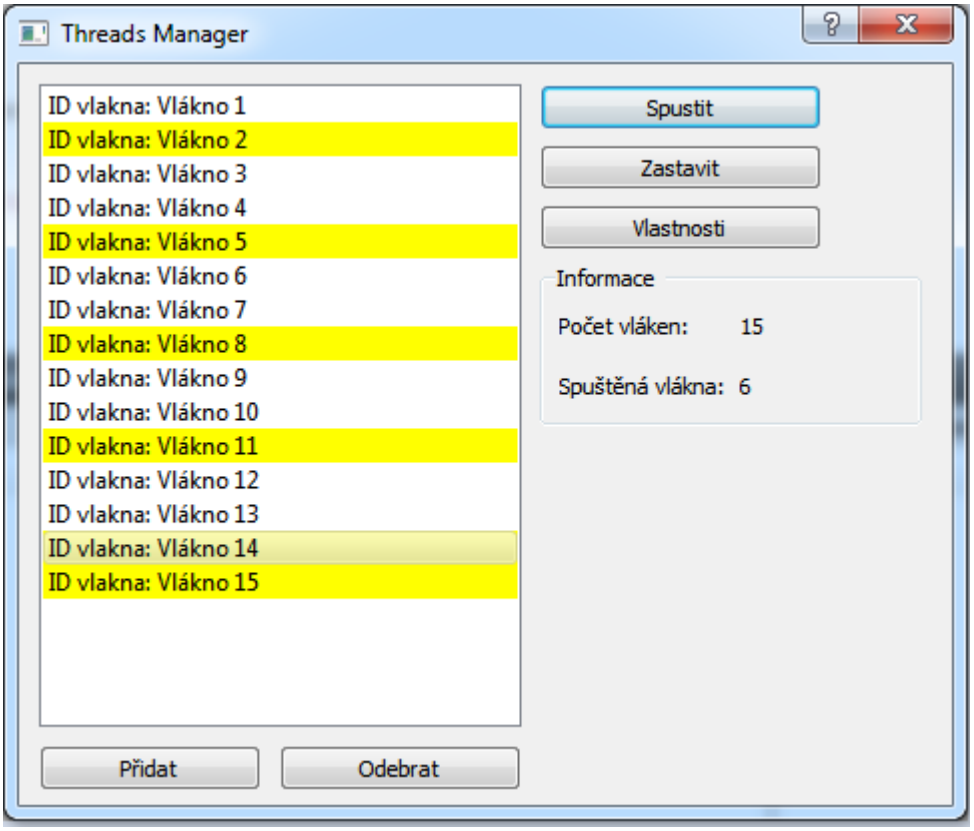
Aplikace je navržena tak, aby byla přehledná a jednoduchá k ovládání. Obsahuje tři dialogová okna. Hlavní dialogové okno, dialogové okno pro vytvoření vlákna a dialogové okno s informacemi o vlákně. Všechna dialogová okna dále popíši.

4.3.1 Okno Threads Manager

Toto okno je hlavní dialogové okno, obsahuje komponentu, která zobrazuje všechna vytvořená vlákna. Pod ní jsou dvě tlačítka přidat a odebrat pro přidávání a odebírání vláken. Na pravé straně aplikace se nacházejí tlačítka pro práci s vybraným vláknem. Umožňují vlákno spustit, zastavit nebo zobrazit podrobné informace o vlákně. Pod tlačítky se nachází informační box o počtu aktuálně vytvořených vláken a počtu spuštěných vláken.



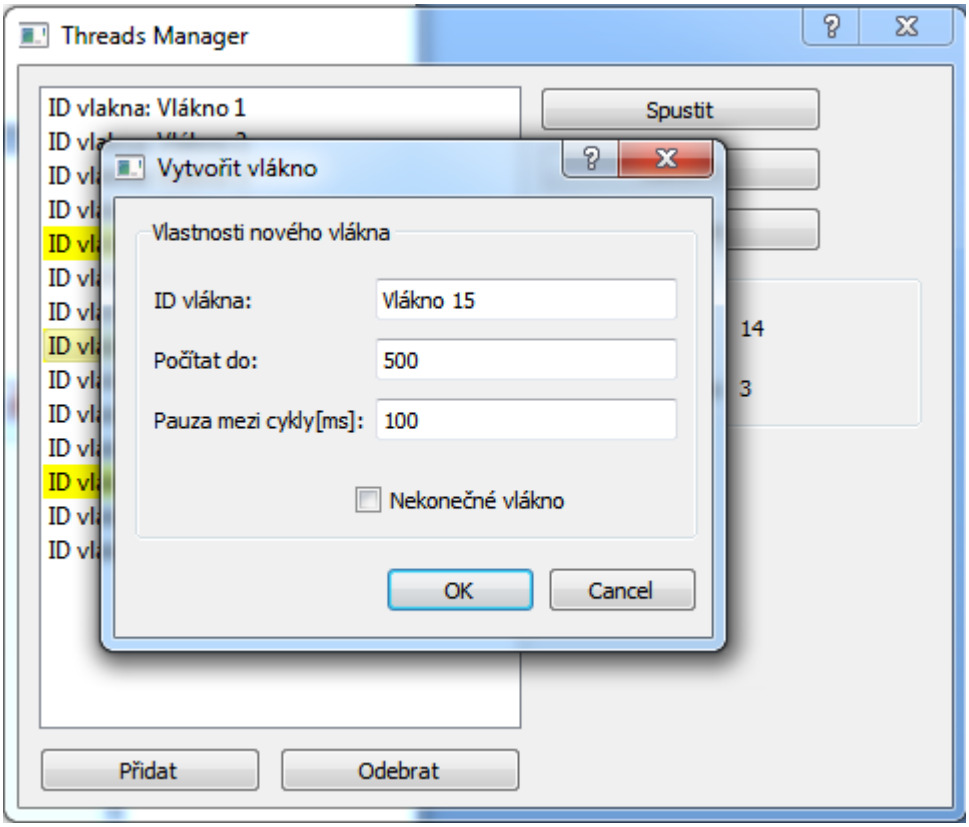
Obrázek 12 - Hlavní okno ihned po spuštění aplikace



Obrázek 13 - Hlavní okno s vytvořenými a spuštěnými vlákny

4.3.2 Okno vytvoření vlákna

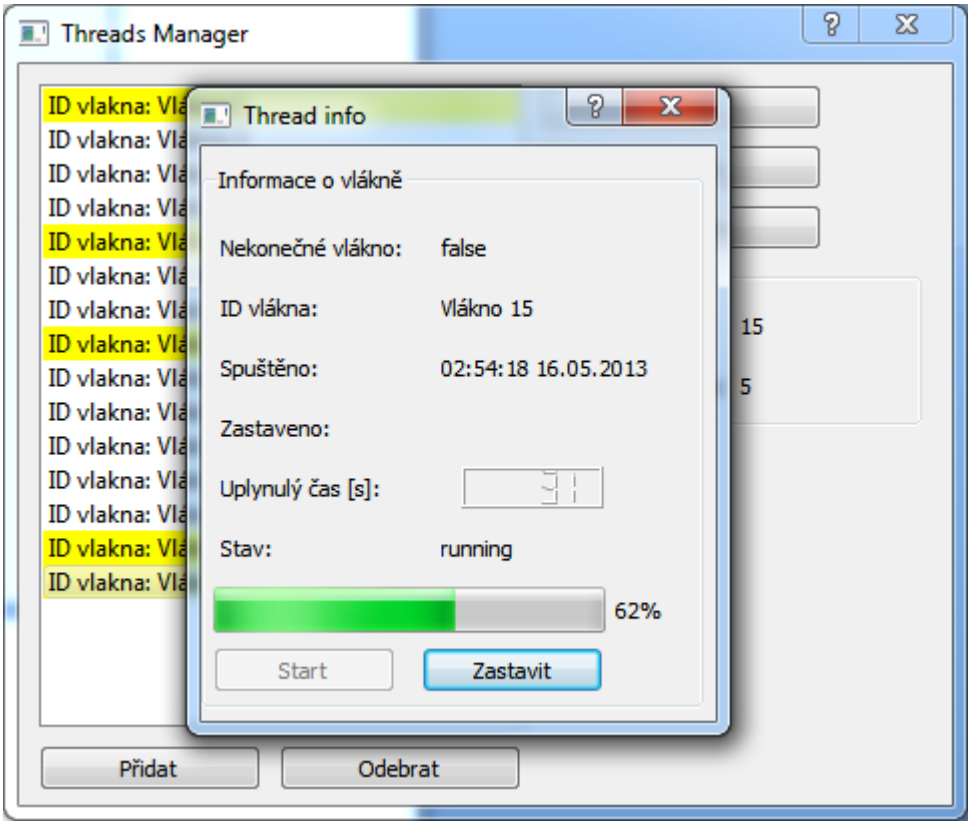
Toto okno je modální a slouží k zadání parametrů vlákna, které chceme vytvořit. První možností je ID vlákna. Další je číslo, do kterého má algoritmus počítat a do posledního pole se vyplňuje čas, po který vlákno „odpočívá“. Poslední možností je určení, zdali bude vlákno nekonečné. Zaškrtnutím políčka a stiskem tlačítka OK vytvoříme nekonečné vlákno, které poběží tak dlouho, dokud ho sami neukončíme.



Obrázek 14 - Okno pro vytvoření vlákna

4.3.3 Okno thread info

Toto okno zobrazuje veškeré informace o vybraném vlákně. Tlačítky Start a Zastavit může vlákno spouštět nebo zastavit. Jako první je informace, jestli je vlákno nekonečné, dále je zobrazeno ID vlákna, kdy bylo spuštěno a zastaveno. Další informací je uplynulý čas od spuštění vlákna v sekundách. Předposlední informací je stav vlákna (running – běží, stopped - zastaveno). Progress bar zobrazuje stav počítání.



Obrázek 15 - Okno s informacemi o vlákně

Závěr

Cílem této práce byl návrh objektového správce vláken, který bude spravovat základní operace s vlákny, jako je spuštění, ukončení a monitorování. Vytvořená aplikace je tedy napsána v jazyce C++ s použitím Qt frameworku. Před samotnou implementací aplikace bylo zapotřebí udělat návrh a analýzu řešení a zvolit nejlepší možnou cestu. Aplikace byla vyvíjena, tak aby bylo možné ji spustit na více platformách, proto byl právě použit Qt framework. Dalším důvodem je velice přehledná dokumentace a potřeba rozšířit své znalosti.

Při tvorbě aplikace byly použity znalosti získané při studiu na vysoké škole, především v oblasti objektově orientovaného programování a programování v jazyce C++. V případě problémů a nejasností byla většinou použita literatura dostupná online a konzultace s vedoucím bakalářské práce. Největší problém se naskytl během implementace metody pro vkládání ukazatele na vlákno do komponenty *QListWidget* přes *QListWidgetItem*, který uchovává data v zastaralém datovém typu *QVariant*. Za pomocí literatury dostupné online jsem daný problém vyřešil.

S výsledkem své práce jsem spokojen a myslím si, že vytvoření této aplikace rozšířilo moje znalosti a schopnosti v oblasti programování. Zadané cíle práce byly splněny.

Literatura

[1] **Beneš, Miroslav.** Procesy a vlákna. *VŠB | Katedra informatiky FEI VŠB-TUO.* [Online] 2003. [Citace: 1. Květen 2013.] http://www.cs.vsb.cz/benes/vyuka/pte/texty/vlakna/ch01s01.html.

[2] **Beneš, Miroslav.** VŠB | Katedra informatiky FEI VŠB-TUO. *Spuštění vlákna.* [Online] 2003. [Citace: 1. Květen 2013.] http://www.cs.vsb.cz/benes/vyuka/pte/texty/vlakna/ch03s01.html.

[3] **National Instruments.** National Instruments. *Differences Between Multithreading and Multitasking for Programmers.* [Online] 25. Duben 2013. [Citace: 1. Květen 2013.] http://www.ni.com/white-paper/6424/en.

[4] **Java™ Platform, Standard Edition 6.** [Online] [Citace: 1. Květen 2013.] http://docs.oracle.com/javase/6/docs/api/.

[5] **Kopetschke, Ing. Igor.** Ústav nových technologií a aplikované informatiky. *DPG – Vlákna.* [Online] 4. Březen 2008. [Citace: 1. Květen 2013.] http://www.nti.tul.cz/cz/images/3/31/DPG-3.pdf.

[6] **Kotala, Z. a Toman, P.** Java: Vlákna (threads). *Vlákna (threads).* [Online] 4. Únor 2001. [Citace: 1. Květen 2013.] http://v1.dione.zcu.cz/java/sbornik/16.html.

[7] **Kottnauer, Jakub.** Albahari.com. *Vlákna v C#.* [Online] 22. Prosinec 2008. [Citace: 1. Květen 2013.] http://www.albahari.com/threading/threading_czech.pdf.

[8] **Kučera, Pavel.** Personal page of Pavel Kucera. *Operační Systémy Rálného Času.* [Online] 20. Listopad 2012. [Citace: 1. Květen 2013.] http://taceo.eu/lectures/bppa/data/bppa_rtos_1.pdf.

[9] **Northwood, Chris.** Computer science notes. *Operating Systems.* [Online] 2010. [Citace: 1. Květen 2013.] http://www.pling.org.uk/cs/ops.html.

[10] **Pěch, Roman.** linux.poweroff.cz, Informace ze světa Linuxu. *Multitasking & Multithreading.* [Online] 2. Únor 2002. [Citace: 1. Květen 2013.] http://linux.poweroff.cz/index.php?clanek=14.

[11] **Peterka, Jiří.** Jiří Peterka: archiv článků a přednášek. *Multitasking.* [Online] 2011. [Citace: 1. Květen 2013.] http://www.earchiv.cz/a93/a310c120.php3.

[12] **Qt Reference Documntetion.** [Online] [Citace: 1. Květen 2013.] http://doc.qt.nokia.com/.

[13] **Wikipedia: Multitasking.** [Online] [Citace: 1. Květen 2013.] http://cs.wikipedia.org/wiki/Multitasking.

[14] **Wikipedia: Vlákno (program).** [Online] [Citace: 1. Květen 2013.] http://cs.wikipedia.org/wiki/Vl%C3%A1kno_%28program%29.

[15] **Wikipedia: Computer multitasking.** [Online] [Citace: 1. Květen 2013.] http://en.wikipedia.org/wiki/Computer_multitasking.

[16] **PRATA**, **Stephen**. *Mistrovství v C*. 3. aktualiz. vyd. Překlad Boris Sokol. Brno: Computer Press, 2007, 1119 s. ISBN 978-80-251-1749-1.

Příloha A – Přiložené CD

CD obsahuje zdrojové kódy aplikace, tento text ve formátu pdf, a zkompilovanou aplikaci ve formátu exe.