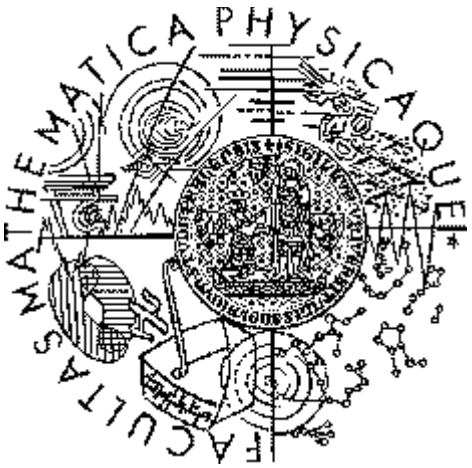
Univerzita Karlova v Praze

Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Jan Dolejš

Úprava některých vlastností vláken v jádře Windows

Středisko informatické sítě a laboratoří

Vedoucí bakalářské práce: RNDr. Vojtěch Jákl

Studijní program: Informatika, Programování

2008

Děkuji panu RNDr. Vojtěchu Jáklovi za odborné vedení mé práce, za rady a za čas, který mi během vypracovávání této práce věnoval.

Prohlašuji, že jsem svou bakalářskou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne Jan Dolejš

Název práce: Úprava některých vlastností vláken v jádře Windows
Autor: Jan Dolejš
Katedra: Středisko informatické sítě a laboratoří
Vedoucí bakalářské práce: RNDr. Vojtěch Jákl
E-mail vedoucího: vjj@mff.cuni.cz

Abstrakt:
Předmětem práce byla úprava těch částí systému Windows, které se týkají dynamických změn priority vláken a strategie pro přidělování času procesoru. Jsou implementovány nové experimentální strategie pro přidělování času procesoru a porovnány v praxi pomocí několika měření. Dále byly vyhledány, analyzovány, zdokumentovány a případně upraveny části zdrojových kódů NTOS, které se zabývají změnou priority vláken. Nakonec byly všechny provedené změny zahrnuty do aplikace komunikující s upraveným jádrem tak, aby bylo možné měnit jeho vlastnosti dynamicky a nebylo třeba pro otestování každé změny překládat nové jádro.

Klíčová slova: Windows, vlákna, priorita, časové kvantum

Title: Thread properties modification in WRK
Author: Jan Dolejš
Department: Network and Labs Management Center
Supervisor: RNDr. Vojtěch Jákl
Supervisor's e-mail address: vjj@mff.cuni.cz

Abstract:
The topic of my thesis was modification of those MS Windows parts which are related to the dynamical threads priority changes and the strategy of rationing quantum. There are implemented new experimental strategies which set different size of quantum for each thread and then they are compared in several measurements. Next issue was to find out, analyze, made a documentation and eventually to modify NTOS source code parts which deal with threads priority changes. Finally, all changes which were made were included to the changed core communicating application in the way where it is able to change the core features dynamically and where it is no need to compile new core for each change testing.

Keywords: Windows, threads, priority, quantum

1 ÚVOD

Historie operačního systému Windows rodiny NT[1] započala v polovině 70. let, kdy skupina okolo Davida Cutlera vyvíjela systém VMS pro procesor VAX Digitalu. V roce 1989 přechází Cutler spolu s dalšími lidmi z Digitalu do Microsoftu, kde začínají pracovat na projektu OS NT OS/2 zpětně kompatibilní s OS/2. Po úspěchu Windows 3.0 (prodalo se více jak 10 miliónů kopií) se dodatečně rozhodlo o zajištění zpětné kompatibility s Windows a celý projekt byl přejmenován na Windows NT. V roce 1993 je uveden na trh první OS z rodiny NT pod označením 3.1.

Starší vývojová větev OS Windows, která byla nadstavbou MS-DOSu a jejímž zástupcem jsou např. Windows 95, koexistovala spolu s OS založenými na jádře NT až do roku 2000, kdy byl vydán poslední zástupce této větve – Windows Me. Od té doby jsou všechny vydané OS společnosti Microsoft založené na jádře NT.

Řada kritiků NTOS[2] argumentovala uzavřeností systému, o jehož fungování byly pouze nepřímé reference (např. kniha (1)). Tento stav trval mnoho let, ale v současné době již existují oficiální cesty k získání zdrojových kódů NTOS. Jednou z těchto cest, díky níž mohla vzniknout tato práce, je Windows Academic Program, zpřístupňující některé části zdrojových kódů NTOS pro akademické a nekomerční použití. V této práci jsou probrány reálně implementované části NTOS, které se zabývají strategií přidělování času procesoru vláknům a změnou jejich priority.

1.1 Cíl práce

Cíle práce sice byly stanoveny na jejím počátku, ale bylo nutné určit jejich proveditelnost. Nejprve bylo nutné určit, zda jsou zdrojové kódy skutečně použitelné, tj. zda je lze přeložit a používat jako normální Windows. Dalším krokem bylo určit, do jaké míry lze provádět úpravy tohoto kódu. Zdrojové kódy jsou totiž distribuovány s jediným ne příliš povedeným příkladem úprav jádra. Po prostudování dostupných materiálů a po několika prvních jednoduchých experimentech se cíle stanovené v počátku ukázaly jako reálné:

[1] Původně vychází z názvu hardwarového emulátoru N-Ten na němž Microsoft vyvíjel první verzi Windows NT. Pro marketingové účely písmena expandovala do názvu New Technology (čerpáno z webu (10)).
[2] Operační systémy z rodiny NT.

- Vyhledat, analyzovat, zdokumentovat a případně se pokusit o úpravy těch částí zdrojových kódů NTOS, které se zabývají změnou priority vláken podle knihy (1). Pokud to bude možné, upravit zdrojové kódy tak, aby se příslušné změny priority daly dynamicky vypínat/zapínat.
- Implementovat další, experimentální, strategie pro přidělování času procesoru vláknům. Práce implementuje a popisuje tři nové.
- Navrhnout a implementovat aplikaci, která dovolí prostřednictvím grafického uživatelského rozhraní ovládat veškeré změny implementace.
- Otestovat některé tyto změny pomocí experimentů tak, aby bylo ukázáno, že se podařilo identifikovat správné části kódu. Některé z experimentů přímo implementovat do výše zmíněné aplikace, k jiným použít externí nástroje.

1.2 Struktura práce

Práce je rozdělena do pěti hlavních kapitol, kde ve třetí a čtvrté kapitola dochází k naplnění stanovených cílů.

Druhá kapitola slouží k seznámení s Windows Academic Programem, díky němuž mohla vzniknout tato práce. Shrnuje informace o jeho komponentách a dále se věnuje pouze části Windows Research Kernelu, která poskytuje část zdrojových kódů NTOS. Také je zde uveden postup, jakým přeložit a provozovat vlastní jádro NTOS spolu s jeho připojením na debugger.

V třetí kapitole je popsán program pro komunikaci s upraveným jádrem sloužící k jeho dynamickým změnám. V této části je obsažena jak uživatelská dokumentace s popisem možností tohoto programu, tak i dokumentace programátorská, ve které jsou zdůvodněny důležitá rozhodnutí činěná během vývoje spolu s návodem pro rozšíření jeho funkčnosti.

Čtvrtá kapitola je rozdělena na tři části. V první části jsou popsány ty pojmy z teorie vláken v NTOS, které budou využívány v částech následujících. Druhá část popisuje rozdíly mezi různými experimentálními strategiemi pro přidělování času procesoru vláknům spolu s ukázkami implementovaného kódu. Poslední (třetí) část se zabývá jak popisem situací které vedou ke zvýšení priority vlákna, tak i analýzou a dokumentací těch částí ve zdrojovém kódu WRK.

Poslední kapitolou je porovnání cílů a dosažených výsledků. V úplném závěru jsou zhodnoceny možnosti této práce do budoucna.

1.3 Poznámka k citacím

Citace a bibliografie jsou podle normy „ISO 690 - číselná reference“ implementované v Microsoft Word 2007. Odkazy na citovanou literaturu jsou uzavřeny v kulatých závorkách (například „podle knihy (1)).

2 PREREKVIZITY

2.1 Windows Academic Program

Windows Academic Program je projekt pro univerzitní a nekomerční použití. Je rozdělen do tří částí.

- **CRK** – Curriculum Resource Kit. Obsahuje materiály založené na knize (1) a nástroje sloužící k pochopení a interaktivnímu testování fungování NTOS. Lze získat z oficiálních stránek (2).
- **WRK** – Windows Research Kernel. Obsahuje množství zdrojových kódů jádra NTOS kompatibilních s Windows Server 2003/XP pro x86 a AMD64. Zahrnuje správu objektů, procesy, vlákna, virtuální paměť, vstupně/výstupní systém a další. Části, které nejsou přístupné v podobě zdrojových kódů jako např. Plug-and-Play jsou dodány ve formě binárních souborů, které se linkují společně se zdrojovými kódy. Výsledkem je plně funkční NTOS, který může být použit v 32bitových Windows Server 2003 SP1, nebo 64bitových Windows XP.
- **ProjectOZ** – Pro tuto část existuje pouze kostra návrhu, proto ji zde nebudu dále uvádět.

Více podrobnějších informací lze získat na oficiálních stránkách (3). Tato práce je založena na používání Windows Research Kernel (dále jen WRK).

2.2 Instalace a nastavení Windows Research Kernel

Oficiální návod na nastavení a instalaci přiložený na cd není vždy jednoznačný. Z toho důvodu je v této části uveden konkrétní postup pro x86 a systém Microsoft Windows Server 2003, který samozřejmě z oficiálního návodu vychází.

Uvedený postup zahrnuje navíc doporučené nastavení virtuálního stroje. Je rozdělen do tří navazujících částí, kde pouze část týkající se překladu a zavedení nového jádra do virtuálního stroje je ve tvaru posloupnosti kroků.

Pro provoz virtuálního systému je použit program VirtualPC 2007, který je volně ke stažení ze stránek (4). Jako debugger, který se připojuje na přeložený NTOS, je použit doporučený windbg (který je také volně ke stažení z oficiální stránek (5)).

2.2.1 Doporučení pro nastavení virtuálního stroje

Pro provoz virtuálního systému Windows Server 2003 je potřeba nejméně 2GB volného místa na disku společně s nejlépe 512MB a více vyhrazených z RAM.

Pro přenos souborů mezi virtuálním a nevirtuálním systémem je výhodné použít sdílení adresářů, které je dostupné v konfiguraci aplikace VirtualPC 2007[3] po nainstalování „VM Additions“. Pro komunikaci s debuggerem se jako port využívá COM1 nastavený v konfiguraci „Named pipe“ na \\.\pipe\debug.

2.2.2 Překlad a zavedení nového jádra NTOS

Pro opětovný překlad stačí opakovat body 3–5. První a druhý bod se provádí pouze při prvním překladu.

1) Zkopírujte WRK na pevný disk (např. C:\WRK).
2) Zjistěte správnou verzi HAL (Hardware Abstraction Layer) knihovny.
   a) Zobrazte si vlastnosti souboru C:\Windows\System32\hal.dll ve virtuálním systému.
   b) Položku „Internal Name“ v záložce „Version“ použijte jako klíč k prvnímu sloupci následující tabulky (Tabulka 1).
   c) Verzi knihovny, kterou je potřeba nakopírovat, naleznete ve druhém sloupci stejné tabulky na stejném řádku.
   d) Zkopírujte příslušnou knihovnu ze složky C:\WRK\WS03SP1HALS\x86\… do složky C:\Windows\System32\ virtuálního systému.

Tabulka 1: Volba vhodné HAL knihovny

Interní jméno souboru hal.dll	***HAL knihovna, kterou budete kopírovat***	***Komentář***
halacpi.dll	*halacpim.dll*	*used by VirtualPC*
halaacpi.dll	*halmacpi.dll*	*ACPI APIC-based PC*
halapic.dll	*halmps.dll*	*MPS*

[3] VirtualPC „Console -> Settings“

3) Spusťte příkazový řádek a postupně proveďte následující příkazy[4]:
   a) path C:\WRK\tools\x86;%path%
   b) cd C:\WRK\base\ntos
   c) nmake -nologo x86=
4) Nakopírujte nové jádro do virtuálního systému. Je třeba zkopírovat soubor C:\WRK\base\ntos\BUILD\EXE\x86\wrkx86.exe do adresáře C:\Windows\system32\ ve virtuálním systému. Lze využít například možnosti sdílení složek virtuálního a hostitelského systému.
5) Do souboru boot.ini virtuálního systému přidejte nový řádek – viz poslední řádek v následující ukázce:

```
[boot loader]
timeout=30
default=multi(0)disk(0)rdisk(0)partition(1)\WINDOWS
[operating systems]
multi(0)disk(0)rdisk(0)partition(1)\WINDOWS="Windows Server 2003, Standard"
multi(0)disk(0)rdisk(0)partition(1)\WINDOWS="wrk" /kernel=wrkx86.exe
/hal=halacpim.dll[5] /debug /debugport=com1[6]
```

2.2.3 Postup připojení debuggeru k jádru testovaného systému

Použití debuggeru není v této práci nutné, ale pro úplnost je zde uveden postup na jeho nastavení a zprovoznění.

Před použitím debuggeru je nutné nastavit cestu k debug symbolům. Tato cesta je složena z cesty k symbolům systému a cesty k symbolům HAL knihovny. Nastavuje se ve spuštěném windbg pomocí menu „File->Symbol File Path“:

```
C:\WRK\base\ntos\BUILD\EXE;C:\WRK\WS03SP1HALS\x86\halacpim
```

Samozřejmě jsou nutné také všechny kroky, které se týkají ladění a byly zmíněny výše.

Pro samotné ladění je třeba spustit windbg s těmito parametry:

```
-k com:pipe,port=\\.\pipe\debug,resets=0,reconnect
```

[4] Je možné přeložit i pomocí připraveného projektu programu Microsoft Visual Studio 2003, který naleznete na stránkách (11) u článku „Using Visual Studio with the Windows Research Kernel”.
[5] Na tomto místě se nachází název odpovídající nakopírované Hardware Abstraction Layer knihovny.
[6] Pouze v případě, že budete chtít tento systém ladit debuggerem.

Po úspěšném propojení je třeba debugger nejprve zastavit (příkazem „Break“) a poté znovu rozběhnout (příkazem g nebo ekvivalentním). Debugger je připraven k ladění upraveného systému.

3 NÁVRH A IMPLEMENTACE APLIKACE PRO KOMUNIKACI S JÁDREM

3.1 Požadavky na aplikaci

Otestování každé jednotlivé změny ve WRK by vyžadovalo samostatný překlad jádra. To by, vzhledem k požadovaným cílům, znamenalo desítky překladů, kopírování a startování do jednotlivých upravených systémů. To není praktické, a proto vznikla následující aplikace pro komunikaci s jádrem, jejímž cílem je dynamicky měnit upravené vlastnosti nového jádra. Navíc obsahuje část umožňující provádět experimenty s různými strategiemi pro přidělování času procesoru vláknům.

3.2 Uživatelská dokumentace

Program WRKExperimentalEnviroment je aplikace pro upravený operační systém Windows Server 2003 pro x86 a kompatibilní. Byl vytvořen ve vývojovém prostředí Microsoft Visual Studio 2008; tedy vyžaduje nainstalovaný .NET Framework 2.0 nebo vyšší. Ten je možné stáhnout zdarma z oficiálních stránek (6) (Naleznete jej také na přiloženém DVD).

WRKExperimentalEnviroment není třeba nainstalovat, stačí jej pouze nakopírovat do cílového počítače. Spouští se souborem WRKExperimentalEnv.exe, který se nachází v podadresáři output domovské složky programu. Jako ukázka funkcí jsou v podadresáři appimg uloženy obrázky ze všech možných obrazovek programu.

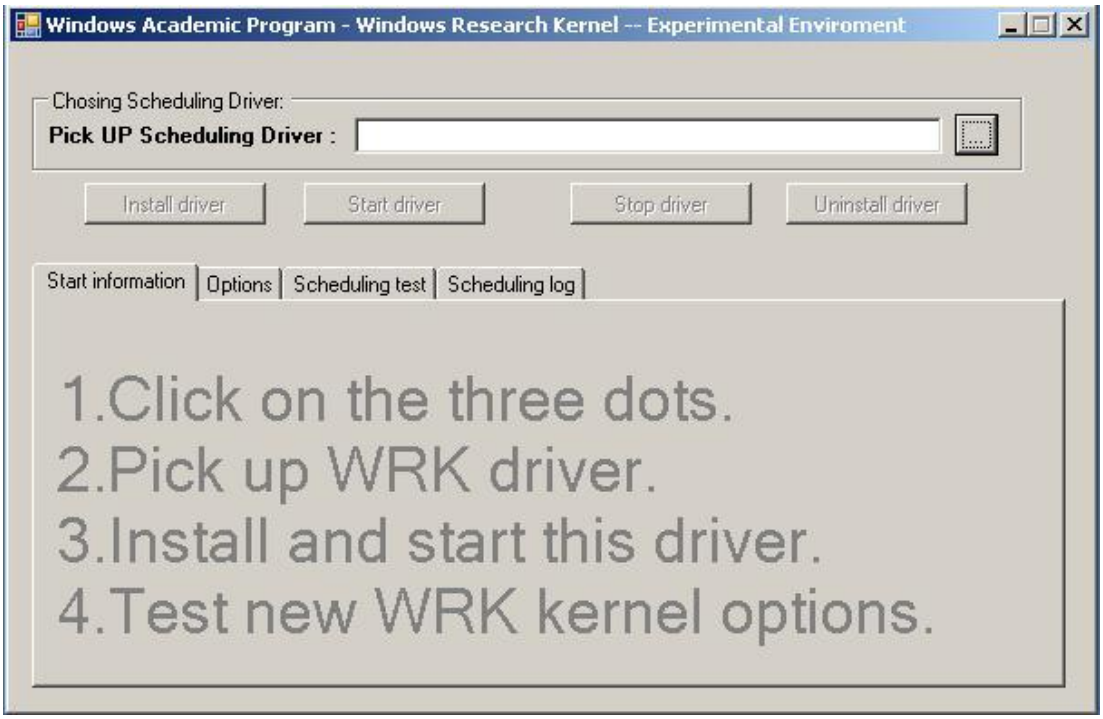
3.2.1 Spuštění a ukončení aplikace

Situace po spuštění aplikace zachycuje Obrázek 1. Aktivní je pouze tlačítko pro výběr souboru ovladače. V části záložek program obsahuje:

- **Start information** – Úvodní obrazovka s popisem akcí, které jsou nutné k odemknutí ostatních záložek.
- **Options** – Záložka pro ovládání části jádra týkající se dynamických změn priority vláken. Navíc obsahuje políčka pro aktivaci různých ladících informací z jádra pro pohodlnější sledování provedených změn.

- **Scheduling test** – Záložka pro testování různých strategií přidělování času procesoru vláknům. Vždy je vybrána jedna plánovací strategie, která se aplikuje na „vybrané“[7] procesy. Dále obsahuje část pro startování „vybraných“ procesů společně s možností vypnutí zvyšování jejich priority standardní cestou.
- **Scheduling log** – Poslední záložka slouží pro výpis výstupu z experimentů záložky „Scheduling test“. Každý experiment má přiřazeno pořadové číslo v rámci aplikace, které je také obsaženo v logu.

Obrázek 1: Program WRKExperimentalEnv po spuštění



Pro odemknutí všech záložek je třeba vybrat, nainstalovat (pomocí tlačítka „Install driver“) a spustit (pomocí tlačítka „Start driver“) ovladač. Standardní ovladač pro dodané upravené jádro WRK se jmenuje WRKDriver.sys a nachází se v podadresáři output/i386/ domovské složky aplikace. Je možné použít i ovladač vlastní, ale doporučuji použít dodaný ovladač.

[7] „Vybrané“ procesy jsou procesy, které obsahují „vybraná“ vlákna. „Vybrané“ vlákno je vlákno, které má ve své struktuře v jádře položku „BucketEnabled“ nastavenu na hodnotu „TRUE“.

Úspěch či neúspěch[8] při instalaci i startování ovladače je signalizován po stisknutí příslušných tlačítek. V případě úspěchu všech akcí jsou odemčeny zbývající záložky.

Po ukončení práce s programem je doporučeno zastavit (pomocí tlačítka „Stop driver“) a odinstalovat (pomocí tlačítka „Uninstall driver“) ovladač. Během standardního ukončování programu jsou tyto akce prováděny automaticky (v případě jejich potřeby). Pokud by ovladač nebyl odinstalován, nastal by problém při příštím pokusu o jeho nainstalování (tento problém popisuje poznámka pod čarou 8).

3.2.2 Záložka Options

Tato záložka je rozdělena do dvou částí (Jak ji zachycuje Obrázek 2). Horní část obsahuje přepínače pro aktivaci/deaktivaci výpisu přidaných ladících informací z jádra. Okolnosti, při kterých dochází k výpisu ladících informací, shrnuje následující seznam:

- **Enable Common BdgPrints** – Při vstupu do nově přidané funkce v jádře NTOS. Tyto funkce mění strukturu, která uchovává informace o aktuálním nastavení jádra.
- **Enable Boost DbgPrints** – Při zvýšení priority vlákna. Jsou vypsány informace o důvodu zvýšení priority a jméno funkce jádra, ve které toto zvýšení nastává. Zahrnuje situace, které je možno zapínat/vypínat.
- **Enable Special DbgPrints** – Aktivní pouze s aktivním „Enable Common DbgPrints“. Přidává informaci o parametrech funkcí.
- **Enable Quantum DbgPrints** – Vypisuje novou hodnotu časového kvanta přiděleného „vybraným“ vláknům, těsně před přidělením procesoru. Tyto vlákna jsou vytvářeny pomocí experimentu na záložce „Scheduling test“.

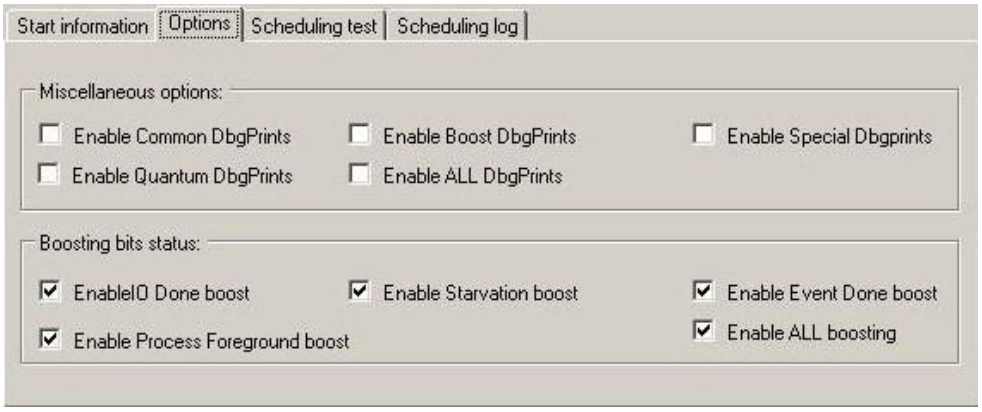
Dolní část obsahuje tlačítka pro aktivaci/deaktivaci částí jádra, které zvyšují prioritu vláknům:

- **Enable IO Done boost** – Zvýšení priority po dokončení I/O operace. Dochází k němu po provedení funkce IoCompleteRequest.

[8] V případě, že ovladač nepůjde nainstalovat, protože se již v tomto systému nachází, Vám aplikace nabídne jeho odinstalování. Po jeho odinstalování ukončete aplikaci a restartujte systém. Poté opětovně spusťte aplikaci. K této situaci může dojít po nestandardním ukončení aplikace.

- **Enable Starvation boost** – Zvýšení priority v důsledku „vyhladovění“[9] vlákna.
- **Enable Event Done boost** – Zvýšení priority po probuzení vlákna čekajícího na semafor, nebo událost. Dochází k němu po provedení funkcí SetEvent, PulseEvent a ReleaseSemaphore.
- **Enable process foregroung boost** – Zvýšení priority po probuzení „foregroundového“ vlákna, které náleží procesu na popředí.

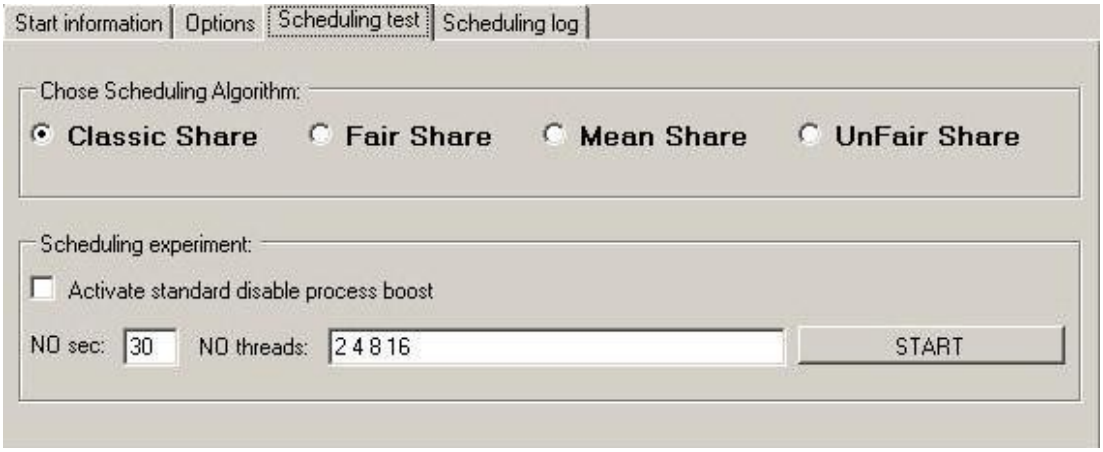
Obrázek 2: Obsah záložky Options



3.2.3 Záložka Scheduling test

Záložka pro výběr různých strategií přidělování času procesoru vláknům a jejich porovnávání[10] pomocí připraveného experimentu. Její obsah zachycuje Obrázek 3.

Obrázek 3: Obsah záložky Scheduling test



[9] „Hladová“ vlákna v NTOS jsou taková vlákna, která jsou ve stavu „ready“ po dobu delší než čtyři vteřiny.
[10] Jeho cílem není vyvozovat jakékoliv závěry, ale ukázat, že různé strategie opravdu dávají různé výsledky (důsledkem změny v jádře).

V horní části se nachází implementované experimentální strategie pro přidělování času procesoru vláknům (kde strategie „Classic Share“ je nezměněná strategie pro přidělování času procesoru). Tyto strategie jsou použity pouze na „vybraná“ vlákna (viz 4.2.1).

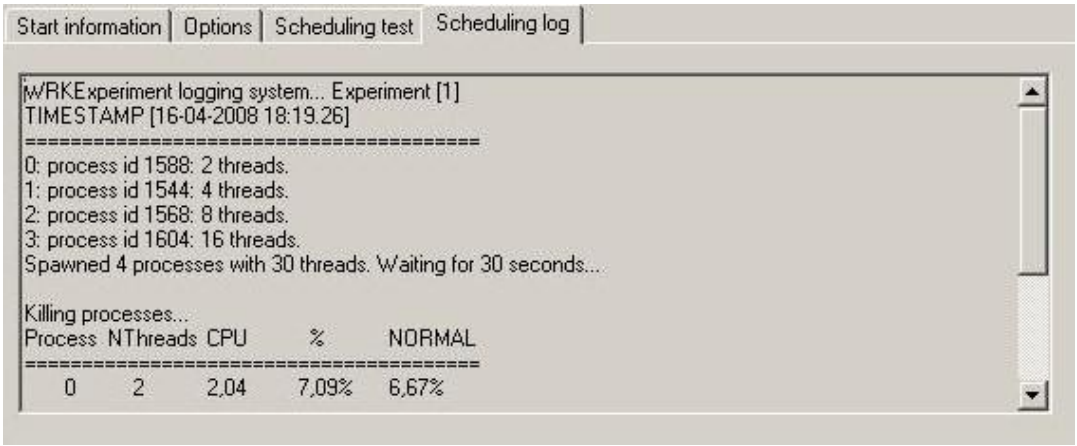
Část „Scheduling experiment“ slouží k vytvoření několika „vybraných“ procesů. Tyto procesy jsou aktivní po dobu, která je zadána do políčka s popiskem „NO sec:“. V dalším editačním poli jsou počty „vybraných“ vláken v jednotlivých procesech[11] oddělené pomocí mezery. Například po stisknutí tlačítka „START“, jak jej zachycuje Obrázek 3, budou vytvořeny celkem čtyři procesy s dvěma, čtyřmi, osmi a šestnácti vlákny na dobu třiceti vteřin. Na jejich „vybraná“ vlákna se aplikuje strategie „Classic Share“.

Maximální doba trvání jednoho experimentu je nastavena na 999 vteřin. Maximální počty vytvářených „vybraných“ vláken jsou 512 (v jednom procesu) a 2 000 (celkem). Zaškrtávací tlačítko „Activate standard disable process boost“ slouží k nastavení standardního příznaku pro zákaz zvýšení priority (aplikuje se na vytvářené „vybrané“ procesy).

3.2.4 Záložka Scheduling log

Po spuštění experimentu stisknutím tlačítka „START“ se začne zapisovat do logu. Tento log je jediná komponenta „Scheduling log“ záložky. Možný obsah logu zachycuje Obrázek 4.

Obrázek 4: Obsah záložky Scheduling log



[11] Všechna vlákna v těchto procesech budou „vybraná“.

Během experimentu je možné program dále ovládat. Není jej ale vhodné vypínat, protože by nedošlo k ukončení nastartovaných procesů (a tím pádem by vytížení procesoru zůstalo na hranici 100 %). Po ukončení experimentu je opět zpřístupněna oblast pro jeho spuštění.

Informace zahrnuté v logu popisuje Tabulka 2.

Tabulka 2: Význam jednotlivých sloupců v logu

Název sloupce	***Význam sloupce***
Process	*Pořadové číslo procesu*
NThreads	*Počet vláken procesu*
CPU	*Čas přidělený všem vláknům procesu*
%	*Procentuální poměr přiděleného času procesu vzhledem k celkovému času*
NORMAL	*Očekávaný procentuální poměr v případě použití strategie „Classic Share“*

3.3 Programátorská dokumentace

Účelem programátorské dokumentace je zasvěcení programátora do logiky programu, ne opisování tříd. Z tohoto důvodu zde nejsou uvedeny žádné tabulky, ani diagramy (tříd, funkcí,…), protože jejich získání není obtížné (například pomocí IDE). Místo toho jsou prezentována a zdůvodněna rozhodnutí (použité třídy, datové struktury), činěná během vývoje programu. V popisech jednotlivých modulů se nachází i návod k rozšíření jejich funkčnosti.

3.3.1 Možnosti komunikace s jádrem

Existuje několik možností, jak může aplikace komunikovat s jádrem operačního systému. V této části jsou hlavní z nich představeny a porovnány z hlediska dvou kritérií. Prvním je standardnost jejich používání, druhé je obtížnost implementace.

- **Interrupt 2e** – Pomocí vloženého assemblerovského kódu lze volat funkce, které se nacházejí v tabulce služeb. Windows Server 2003 Enterprise Edition obsahuje necelých tři sta takovýchto funkcí. Do registru AX se uloží číslo volané funkce a v BX se nachází seznam parametrů k této funkci. Implementace tohoto skoku do jádra není složitá, ale v současné době se již nepoužívá.
- **Sysenter** – Do registru asociovaného k této instrukci se při startování Windows uloží adresa funkce, která zpracovává její volání. V registru EAX je

opět uloženo číslo služby a v registru EDX seznam parametrů. Jedná se o novější a rychlejší variantu přechodu do jádra než Interrupt 2e. Tento způsob je momentálně využíván ve Windows API.

- **Syscall** – Obdoba sysenter pro 32bitové procesory AMD.
- **DeviceIoControl** – funkce Win32 API – jedná se o standardní způsob komunikace aplikací s drivery. Pomocí driveru lze zpřístupnit veškeré struktury a volat veškeré funkce implementované v jádře. Tento způsob je často a dlouho používaný, avšak o něco náročnější na implementaci, protože je zapotřebí navrhnout a naprogramovat vlastní ovladač.
- **Použití nedokumentované API funkce** – Některé funkce jádra jsou pro potřeby aplikací Microsoftu zpřístupněny i prostřednictvím nedokumentovaných API funkcí se stejným jménem a parametry. Přechod mezi dvojníky z „ringu 3“ na „ring 0“ je potom realizován pomocí některé z první tří uvedených metod. Pro jejich použití je třeba příslušná statická knihovna. Tento postup volání funkcí v jádře je použit i v zatím jediném oficiálním ukázkovém příkladu změny jádra. Bohužel o tom, že přechod do jádra je ukryt v nedokumentované funkci, není nikde v dostupné dokumentaci WRK žádná informace. Tento způsob je limitován omezeným množstvím takto zpřístupněných funkcí (ne všechny oblasti jsou pokryty).

Použití nedokumentovaných API funkcí je pro účely tohoto programu nevhodné, protože by musely být upraveny[12] tak, aby reflektovaly změny ve WRK.

Možnosti použití instrukce Syscall (a i Interrupt 2e) jsou o něco větší. Do tabulky služeb je možné přidat i nově vytvořenou funkci[13]. Toto řešení není vhodné, protože jakákoliv záplata operačního systému může tuto tabulku služeb změnit (což by mohlo způsobit nefunkčnost tohoto řešení).

Jediné správné (doporučované) řešení jak pomocí aplikace volat funkce jádra, je použití API funkce DeviceIoControl spolu s vlastním ovladačem. Ovladač může používat všechny funkce, které jádro exportuje. Jakým způsobem přidat (i nově

[12] Podobný princip, který byl použit u jediného oficiálního příkladu „Fair Share“. V tomto příkladu byla upravena nedokumentovaná API funkce NtSetInformationThread.
[13] Návod na rozšíření tabulky služeb je uveden na stránkách (11) v článcích „Howto: Implementation of new system service calls I a II“.

vytvořenou) funkci ve WRK mezi exportované popisuje návod na stránkách (6) ve článku „Howto: Export Kernel Symbols“.

3.3.2 Grafické uživatelské rozhraní

Modul s grafickým uživatelským rozhraním je rozdělen na celkem čtyři části:

- **MainForm.cs** – Zastřešuje práci celého programu. Obsahuje grafický návrh i práci s grafickým uživatelským rozhraním společně s reakcemi na relevantní události. Tento modul používá ostatní moduly, kterým pouze připravuje parametry.
- **Program.cs –** Automaticky vygenerovaný obsah pro inicializaci a spuštění grafické aplikace.
- **WRKUnmanaged.cs** – Obsahuje třídu s funkcemi pro komunikaci s jádrem. Samotné funkce jsou implementovány v knihovně communication.dll, pro jejich importování se používá P/Invoke (7).
- **experiment.cs –** Implementuje třídu pro vytváření „vybraných“ procesů a experiment pro přidělování časového kvanta vláknům (viz kapitola 4.2).

Zvláštní pozornost bude věnována modulu „experiment.cs“.

Účelem třídy Experiment implementované ve zdrojovém souboru experiment.cs je spouštět externí proces, jehož úkolem je vytvoření tolika „vybraných“ vláken, kolik je požadováno (parametrem, který mu je předán na příkazové řádce). V této třídě také dochází k nastavení požadovaných parametrů[14].

Prvním rozhodnutí týkající se experimentu bylo jeho zařazení do nového vlákna. Experiment může být aktivní v každý okamžik pouze jeden (aby nedocházelo k jejich vzájemnému ovlivňování), ale díky přesunutí experimentu do nového vlákna je možné v jeho průběhu dynamicky zapínat/vypínat např. ladící výpisky. Implementace nového vlákna používá standardní třídu Thread ze jmenného prostoru System.Threading.

[14] Některé parametry jsou předány po stisknutí tlačítka „START“ v záložce „Scheduling test“ (počet jeho „vybraných“ vláken), jiné jsou nastaveny automaticky pro všechny stejně (prioritní třída tohoto procesu, zobrazení, a další).

Dalším důležitým rozhodnutím bylo, jakým způsobem vytvářet proces (popsaný výše) a jak nastavovat jeho vlastnosti (prioritní třída a další). Platforma .NET nabízí dvě možnosti pro práci s procesy:

- Pomocí P/Invoke importovat příslušné API funkce (např. CreateProcess).
- Použít standardní třídu System.Diagnostic.Process.

Ve třídě Process jmenného prostoru System.Diagnostic lze nastavit všechny požadované vlastnosti[15] procesu. Proto bylo toto řešení použito.

Posledním problémem je nutnost komunikace třídy Experiment s třídou obsahující grafické uživatelské rozhraní. Komunikace je nutná jak kvůli průběžnému zápisu do logu, tak i kvůli nutnosti upozornit na ukončení experimentu (aby mohl být opět spuštěn další).

I tento problém se dá řešit minimálně dvěma způsoby:

- Nastavení příslušných (log, skupina políček pro nastavení experimentu) komponent jako statické, nebo předání odkazu na ně do třídy Experiment.
- Použití uživatelsky definovaných událostí (8).

První způsob je velice nepraktický a nešikovný. Naproti tomu využití standardního nástroje v C# (uživatelsky definované události) je jak elegantním, tak i funkčním řešením. Princip uživatelsky definovaných událostí je stejný, jako u standardních událostí (např. „onClick“) – ve třídě s grafickým návrhem si zaregistrujeme rutinu reagující na danou událost (tedy třída Experiment nabízí tuto událost) a ve třídě Experiment tuto událost vyvoláváme. Před vyvoláním události je třeba kontrolovat, existuje-li nějaká rutina, která na ni reaguje.

3.3.3 Knihovna pro komunikaci s jádrem communication.dll

Tato knihovna je pouze prostředník mezi grafickou aplikací a ovladačem. Je implementována z důvodu oddělení grafického návrhu od komunikace s jádrem. Exportuje funkce pro komunikaci s ovladačem společně s funkcemi pro jeho nainstalování/odinstalovaní a nastartování/zastavení.

[15] Třída jeho priority, předávání příkazové řádky, možnosti jeho ukončení, a další.

Jak už bylo řečeno výše, funkce pro komunikaci s ovladačem slouží pouze jako prostředník mezi aplikací a ovladačem. Navíc kontrolují předané parametry a hlásí chybové stavy pomocí dialogového okna.

Těla těchto funkcí se liší pouze minimálně, hlavně v identifikaci služby ovladače (IOCTL_…). Práci těchto funkcí zachycuje následující výřez ze zdrojového kódu:

```
hDevice = CreateFile (completeDeviceName, …);

if (hDevice != ((HANDLE)-1))
{
        DeviceIoControl (hDevice, IOCTL_…, …);
        CloseHandle (hDevice);
}
```

Přechod do jádra

Prologem a epilogem je získání (respektive zneplatnění) identifikátoru na ovladač (kterému se ve Windows říká „handle“). Komunikaci s ovladačem zajišťuje API funkce DeviceIoControl, která má za parametry jak odkaz na ovladač společně s identifikací požadované služby, tak i vstupní/výstupní zásobník pro parametry předávané do/z ovladače.

Pro rozšíření funkčnosti je třeba nejprve naimplementovat požadovanou službu do ovladače (a případně i do WRK) a poté pomocí výše popsaného postupu spouštět. Protože došlo k fyzickému oddělení souborů ovladače a této knihovny, je třeba udržovat dva stejné seznamy s kódy těchto služeb (IOCTL_…), které se nalézají v hlavičkovém souboru jak knihovny, tak ovladače.

3.3.4 Ovladač jádra WRKDriver.sys

Ovladač jádra je navržen pro x86. Pro jeho implementaci byl použit návod, prezentovaný na přednášce (9). Je rozdělen do čtyř modulů:

- **Makefile** – Pouze pro čtení.
- **Sources** – Obsahuje soubory, z kterých se výsledný ovladač překládá. Protože jsou v ovladači použity funkce z jádra, linkuje se statickou knihovnou, která je exportuje (tato knihovna se nachází ve stejné složce, jako přeložené jádro WRK).
- **WRKDriver.h** – Obsahuje hlavičky volaných funkcí z jádra společně s kódy služeb poskytovaných tímto ovladačem.
- **WRKDriver.c** – Implementace poskytovaných služeb.

Zvláštní pozornost bude věnována modulu WRKDriver.c.

Ve funkci DriverEntry se ovladač inicializuje. Důležitou částí je uložení odkazu na funkci, zpracovávající požadavky na službu (zavoláním funkce IoDeviceControl):

```
DriverObject->MajorFunction[IRP_MJ_DEVICE_CONTROL] = DrvDispatch;
```

Struktura funkce DrvDispatch realizující nabízené služby vypadá následovně:

```
NTSTATUS DrvDispatch( IN PDEVICE_OBJECT DeviceObject, IN PIRP Irp )
{
       PIO_STACK_LOCATION  irpStack;
       PVOID               ioBuffer;
       ULONG               inputBufferLength;
       ULONG               outputBufferLength;
       ULONG               ioControlCode;

       irpStack = IoGetCurrentIrpStackLocation (Irp);

       ioBuffer           = Irp->AssociatedIrp.SystemBuffer;
       inputBufferLength  = irpStack->
              Parameters.DeviceIoControl.InputBufferLength;
       outputBufferLength = irpStack->
              Parameters.DeviceIoControl.OutputBufferLength;

       switch (irpStack->MajorFunction)
       {
       case IRP_MJ_CREATE:
       case IRP_MJ_CLOSE:

       case IRP_MJ_DEVICE_CONTROL:

              ioControlCode = irpStack->
                              Parameters.DeviceIoControl.IoControlCode;

              switch (ioControlCode)
              {
              case IOCTL_WRK_IODONEB:

              …

              default:

                     Irp->IoStatus.Status = STATUS_INVALID_PARAMETER;
                     break;
              }

              break; //IRP_MJ_DEVICE_CONTROL
              }
       }

       IoCompleteRequest (Irp, IO_NO_INCREMENT );
       return Irp->IoStatus.Status;
}
```

Prolog

Požadovaná akce

Požadavek na službu voláním funkce „DeviceIoControl“

Obsluha

Identifikace požadované služby

Obsluha služby s kódem IOCTL_WRK_IODONEB

Epilog

V části “Prolog“ jsou předány vstupní parametry spolu s důvodem vyvolání. V případě, že je důvodem požadavek služby, se extrahuje její kód[16] (proměnná

[16] Tento kód je 32bitové číslo. Zahrnuje informace o typu ovladače, identifikaci požadované služby, způsobu předávání parametrů a přístupových právech. Pro snadnější vytváření tohoto kódu definuje hlavičkový soubor wdm.h makro CTL_CODE.

ioControlCode) podle kterého se volá příslušná obsluha. Případné vstupní parametry jsou uloženy v proměnné ioBuffer do které se ukládají i parametry výstupní. Nakonec je proveden „Epilog“, ve kterém je volajícímu upozorněn o vykonání požadované služby.

Pro přidání nové služby se nejprve musí vytvořit její kód (unikátní) a poté přidat jeho obsluhu jako nový případ do části „Obsluha“ výše popsané funkce. Jak už bylo napsáno v 3.3.3, je nutné tento kód vložit jak do hlavičkového souboru ovladače, tak i knihovny.

Ovladač je možné přeložit pomocí prostředí „Windows Vista and Windows Server 2008 x86 Free Build Environment“ Windows Driver Kitu (dříve Driver Development Kit), který se nachází na přiloženém DVD. V tomto prostředí přejděte do podadresáře driver domovské složky aplikace (stejnými příkazy jako v případě příkazového řádku Windows) a zavolejte příkaz „build“. Pokud překlad proběhne bez chyb, vytvoří se v požadované složce (danou nastavením v souboru sources) soubor ovladače se symboly pro debugger.

4 VLÁKNA VE WINDOWS

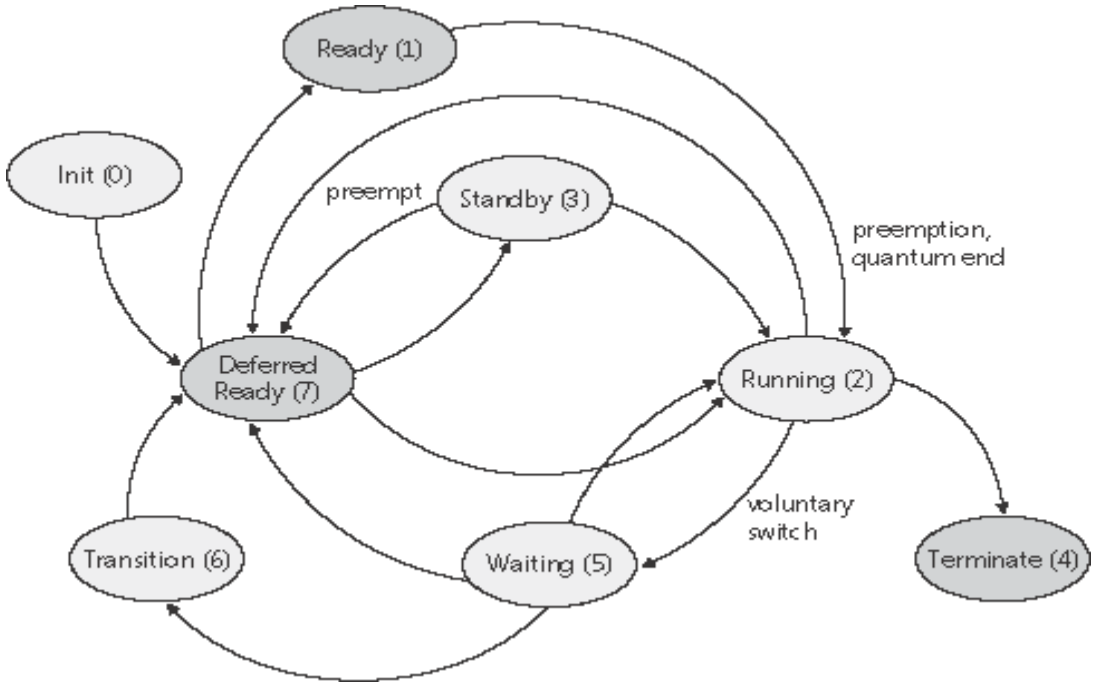
4.1 Výběr ze základní teorie

Cílem této kapitoly je nastínit teoretický základ fungování vláken v NTOS pro ty části, které se přímo týkají formulovaných cílů práce (1.1). Mezi ně patří informace o možných stavech vlákna, prioritách a časovém kvantu. Tyto informace jsou čerpány převážně z knihy (1), a proto jsou v názvu podkapitol i čísla stránek, ze kterých bylo čerpáno.

4.1.1 Stavy vlákna (1 pp. 334-338)

Možné stavy vlákna ve Windows Server 2003 zachycuje Obrázek 5 (1 p. 338).

Obrázek 5: Stavy vlákna na Windows Server 2003



- **Init** – Stav, ve kterém se nachází nově vytvářené vlákno během inicializace vnitřních struktur.
- **Ready –** Vlákno v tomto stavu je připraveno získat procesor.
- **Running** – Po přepnutí kontextu[17] (mezi vláknem, kterému byl právě přidělen procesor a vláknem, které ho mělo jako poslední) je stav vlákna

[17] Uložení stavu registrů a ukazatelů posledního vlákna a následné načtení stavu registrů a ukazatelů nového vlákna.

změněn na „running“. Vlákno v tomto stavu má k dispozici procesor (po dobu přiděleného časového kvanta) a provádí na něm své instrukce. Během této doby se ho může vzdát a vstoupit do stavu „waiting“, nebo mu může být odebrán (ve prospěch „ready“ vlákna s vyšší prioritou).

- **Standby** – V tomto stavu se nachází vlákno, které získá konkrétní procesor po jeho uvolnění (jedná se o formu rezervace). Každý procesor může mít „rezervováno“ pouze jedno vlákno. Tuto „rezervaci“ může nahradit jiné vlákno vyšší priority.
- **Terminate** – Po provedení všech instrukcí dojde k dobrovolnému ukončení vlákna a přepnutí do tohoto stavu.
- **Waiting** – Vlákno vstupuje do stavu „waiting“ z několika důvodů – dobrovolné čekání na objekt k synchronizaci, přepnutí operačním systémem v důsledku stránkování… Po jeho probuzení, je přepnuto zpátky do stavu „running“, nebo vloženo do příslušné fronty „ready (podle priority).
- **Transition** – Vlákna v tomto stavu jsou sice připravena získat procesor, ale mají vystránkovaný „kernel stack“. Po jeho vrácení do paměti je vlákno přepnuto do stavu „ready“.
- **Deferred ready** – Vlákna v tomto stavu byla vybrána pro získání konkrétního procesoru, ale ještě nebyla naplánována.

4.1.2 Priority v NTOS (1 pp. 327-334)

Priorita vlákna je jediným kritériem, podle kterého se vybírá vlákno, které získá procesor. V NTOS je implementováno celkem dvaatřicet prioritních úrovní (0–31), které jsou rozděleny do tří oblastí:

- 0: Priorita 0 je pro speciální systémové vlákno.
- 1–15: tzv. dynamické priority. Na vlákna těchto priorit se aplikují algoritmy pro zvýšení priority (které jsou popsány v 4.3).
- 16–31: tzv. realtime priority. Priorita těchto vláken se nemění. Určeno hlavně pro systémová vlákna.

Na úrovně priority existují dva pohledy. Jeden pohled je systémový s prioritami 0–31. Druhý pohled je skrz Windows API, které procesy (i vlákna) sdružuje do

prioritních tříd (Real-time, High, Above normal, Normal, Below normal a Idle pro procesy). Prioritní třída procesu určuje možný rozsah priority jeho vláken. Rozsahy priorit prioritních tříd procesů se vzájemně překrývají[18] (s výjimkou třídy Real-time, která je jako jediná v oblasti „realtime“ priorit).

Konkrétní priorita vlákna procesu je spočtena na základě prioritní třídy procesu a posuvu v rámci třídy daného prioritní třídou tohoto vlákna[19]. Takto spočtená priorita se nazývá „bázová“. Vlákna jednoho procesu mohou mít různé hodnoty „bázové“ priority (v závislosti na jejich prioritní třídě). Prioritní třída procesu (vlákna) se nastavuje při jeho vytvoření. Tato třída jde také změnit například pomocí API funkce SetPriorityClass (SetThreadPriority). Změna prioritní třídy procesu (vlákna) vyvolá přepočet „bázové“ priority všech dotčených[20] vláken.

Jak již bylo řečeno výše, priorita vláken se může[21] zvyšovat v důsledku několika událostí (které jsou popsány v 4.3). Proto je třeba uchovávat pro každé vlákno dvě hodnoty priority. Jedna z nich je „bázová“ a druhá „aktuální“. Hodnota „aktuální“ priority je vždy větší, nebo rovna prioritě bázové a maximálně rovna patnácti (horní hranice „dynamického“ oblasti priorit). Při výběru vlákna, které získá procesor jako další, se vybere vlákno s nejvyšší hodnotou „aktuální“ priority.

4.1.3 Časové kvantum vlákna (1 pp. 340-343)

Doba, po kterou má vlákno ve stavu „running“ k dispozici procesor, se nazývá časové kvantum.

Ve Windows 2000 Professional a Windows XP má vlákno k dispozici procesor standardně po dobu dvou hodinových tiků, naproti tomu Windows Server 2003 standardně přiděluje šestkrát delší časové kvantum (celkem dvanáct hodinových tiků). Standardní počet přidělených tiků se dá změnit v grafickém nastavení systému.

Delší časové kvantum omezuje přepínání kontextu. Díky tomu má serverová aplikace probuzená na žádost klienta větší šanci na dokončení požadované operace a opětovný přechod do stavu „waiting“ před vypršením jejího časového kvanta.

[18] Jejich konkrétní rozsah hodnot zachycuje obrázek 6-13 v knize (1 p. 329).
[19] Tyto třídy jsou (Time-critical, Highest, Above normal, Normal, Below normal, Lowest a Idle).
[20] V případě, že se mění prioritní třída vlákna, se přepočítává pouze jeho „bázová“ priorita.
[21] Je-li hodnota „bázové“ priority vlákna v oblasti „dynamických“ priorit.

Interval mezi jednotlivými hodinovými tiky závisí na použité hardwarové platformě. Frekvence přerušení stanovuje HAL a ne jádro systému. Například pro většinu jednoprocesorových x86 je tento interval 10ms a u víceprocesorových 15ms.

Každý proces má uloženu výchozí hodnotu časového kvanta ve své vnitřní struktuře a tato hodnota se nastaví i jeho vláknům (během jejich vytváření). Vnitřně je tato hodnota trojnásobkem počtu tiků. To znamená, že ve Windows 2000 Professional a Windows XP se procesům standardně nastaví délka časového kvanta na šest (2 * 3) a ve Windows Server 2003 standardně na šestatřicet (12 * 3).

Důvodem (uvedeným v knize (1 p. 341) kde se nachází i podrobné vysvětlení) pro nastavení velikosti časového kvanta na trojnásobek počtu hodinových tiků, je získání možnosti snížení kvanta po probuzení vlákna. Snižování kvanta tedy nastává ve dvou případech:

- Snížení o tři každým tikem.
- Snížení o jedna po probuzení ze stavu „waiting“.

Při poklesu hodnoty časového kvanta na (nebo pod) hodnotu nula je volána funkce, jejímž úkolem je vybrat vlákno, kterému bude přidělen uvolněný procesor.

4.1.4 Výběr vlákna pro přidělení času procesoru (1 pp. 345-347)

V NTOS je implementován preemptivní, prioritně řízený plánovací systém. To znamená, že vlákno s nejvyšší „aktuální“ prioritou, které je ve stavu „ready“ získá[22] procesor (maximálně po dobu jeho časového kvanta, poté následuje další „soutěž“ o daný procesor).

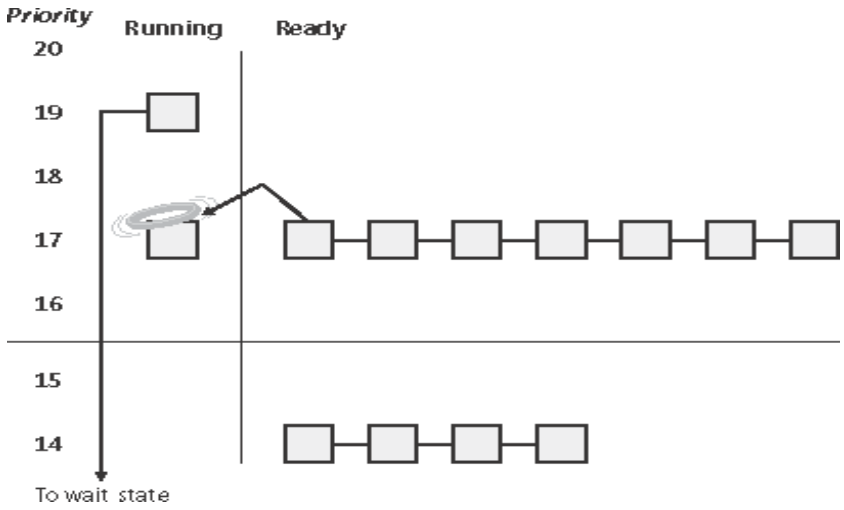
Pro lepší představu fungování plánovače NTOS jsou v další části uvedeny možné „plánovací scénáře“. Použité ilustrační obrázky byly převzaty z knihy (1 pp. 6:345, 7:346, 8:347):

- **Dobrovolné přepnutí** – Vlákno se dobrovolně vzdá procesoru, pokud začne čekat na nějaký objekt (jako třeba událost, semafor, proces, vlákno, atd.)

[22] Nemusí platit ve víceprocesorovém NTOS, protože v něm má každé vlákno uložen seznam všech procesorů, které může získat (respektive o které samo stojí). Vlákno, které nemůže daný procesor získat se „soutěže“ o něj neúčastní. Tento seznam se dá změnit API funkcí SetThreadAffinityMask, nebo například Správcem úloh.

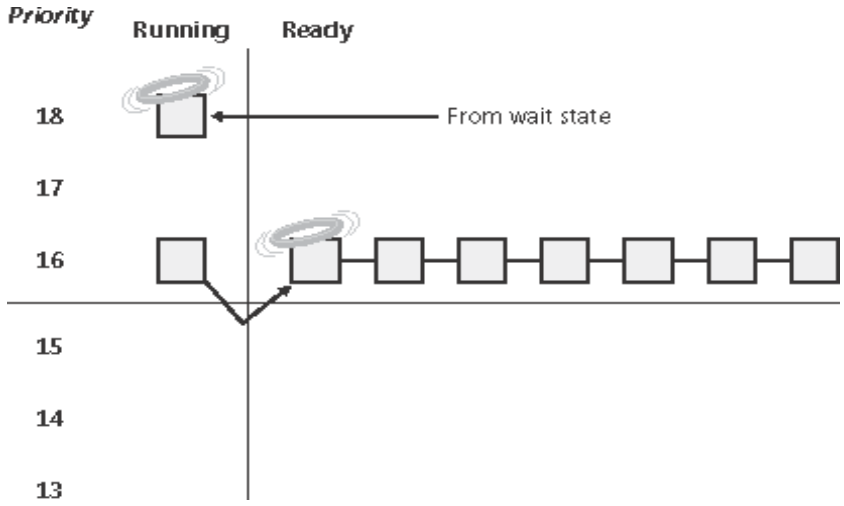
zavoláním jedné z Windows API funkcí (WaitForSingleObject nebo WaitForMultipleObjects). Tuto situaci ilustruje Obrázek 6[23].

Obrázek 6: Dobrovolné přepnutí



- **Nucená výměna** – Situace, při které je vlákno s nižší prioritou, donuceno vzdát se procesoru ve prospěch vlákna s vyšší prioritou. Vláknu, které je donuceno vzdát se procesoru, je stav změněn na „ready“ a vrátí se do čela fronty příslušné priority. Tato situace může nastat v případě, že dojde k probuzení jiného vlákna s vyšší prioritou (jak zachycuje Obrázek 7), nebo ke změně priority libovolného vlákna.

Obrázek 7: Nucená výměna

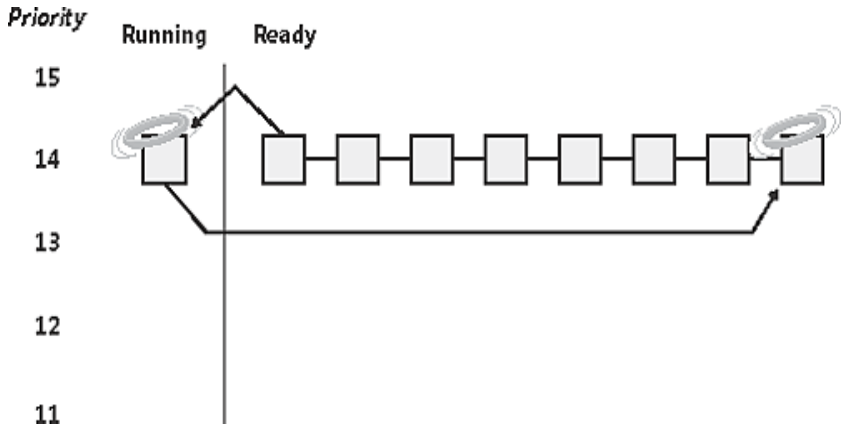


- **Vypršení časového kvanta –** V případě, že vlákno vyčerpá přidělené časové kvantum, musí systém určit, má-li být jeho priorita snížena a které vlákno

[23] Vláknu, které se vzdá procesoru, se priorita nesnižuje (může to tak podle obrázku vypadat). Pouze se přesunuje do fronty vláken čekající na daný objekt. Hodnota zbývajícího časového kvanta se nemění (změní se až po jeho probuzení).

získá jako další čas procesoru. Pokud se priorita snižuje, hledá se vlákno s nejvyšší prioritou. V opačném případě je aktuální vlákno zařazeno na konec fronty „ready“ dané priority a procesor se přiřadí prvnímu vláknu z této fronty[24]. Tuto situaci zachycuje Obrázek 8.

Obrázek 8: Vypršení časového kvanta



- **Ukončení běhu vlákna** – Vlákno je přepnuto ze stavu „running“ do stavu „terminated“.

4.2 Strategie přidělování času procesoru vláknům

V této části jsou popsány nově implementované experimenty se strategiemi pro přidělování času procesoru vláknům. Primárním účelem nebylo vytvářet nové strategie nějakým „rozumným“ způsobem pro lepší chování systému, ale ověřit, že je možné změnami ve WRK měnit jádro systému.

V první části jsou nejprve popsány principy, podle kterých je standardně přidělováno časové kvantum vláknům. Následně jsou představeny nově implementované strategie a rozebráno jejich očekávané chování. Druhá část vysvětluje provedené změny jádra. V třetí části je pomocí sady několika měření ukázáno, že různé strategie dávají opravdu různé výsledky (kvůli změnám jádra).

[24] V případě, že byla fronta „ready“ dané priority prázdná, vybere se právě vložené vlákno.

4.2.1 Popis strategie přidělování času procesoru v NTOS

Jak bylo uvedeno v kapitole 4.1.3, doba, po kterou má vlákno k dispozici procesor se nazývá časové kvantum. Strategie implementovaná v NTOS (v této práci pojmenovaná „Classic Share“) standardně přiděluje automaticky všem vláknům stejně dlouhá časová kvanta. Z pohledu procesu je celkové množství přiděleného času přímo úměrné počtu jeho vláken (i když samotné procesy žádné časové kvantum nedostávají). Tedy čím více má proces vláken, tím více času získá.

V následujícím výčtu jsou popsány nově implementované experimentální strategie pro přidělování různě dlouhých časových kvant vláknům:

- **Fair Share** – Strategie, která byla inspirována výše zmíněnou „nespravedlností“ mezi procesy. Její filosofií je vyrovnat celkový čas přidělený procesům bez ohledu na počet jejich vláken. V principu tedy každému vláknu přiděluje časová kvanta, jejichž délka je nepřímo úměrná počtu vláken v procesu. Čím méně má proces vláken (vzhledem k průměru), tím delší časová kvanta jeho vlákna získávají a naopak. Tato strategie je implementována v rámci jediného oficiálního příkladu použití WRK.
- **Mean Share** – Strategie, která je kompromisem mezi standardní strategií a strategií „Fair Share“. Délka časového kvanta je tedy delší pro vlákna procesu s méně vlákny (vzhledem k průměru) a naopak, ale rozdíly mezi vlákny jsou menší, než je tomu u „Fair Share“.
- **UnFair Share** – Strategie, která je filosofickým opakem „Fair Share“. Cílem tedy je, zvýhodňovat procesy s více vlákny v ještě větší míře než je tomu u „Classic Share“. Čím méně má proces vláken (vzhledem k průměru), tím kratší časová kvanta jeho vlákna získávají a naopak.

4.2.2 Reference do zdrojového kódu WRK

Následující úprava zdrojových kódů vychází z jediného oficiálního příkladu změny jádra pomocí WRK. Nedokumentované a nepopsané zdrojové kódy oficiální úpravy WRK „Fair Share“ laskavě poskytl pan Dave Probert.

Protože zásah do přidělování času procesoru by vedl k horší stabilitě[25] systému, je použito řešení, kdy se změny aplikují pouze na vlákna „vybraných“ procesů. „Vybraná“ vlákna jsou vlákna určená pro testování změn chování systému. Aby bylo možné vlákno označit jako „vybrané“, přidaly se následující položky do struktury vlákna:

```
wrk\base\ntos\inc\ps.h:

typedef struct _ETHREAD {
KTHREAD Tcb;
…
union {
        ULONG SameThreadPassiveFlags;

        struct {

          …
          ULONG BucketsEnabled  : 1;
          ULONG ScheduleBucket  : 5;
      #define MAXIMUM_SCHEDULE_BUCKETS 32
        };
     };
…
} ETHREAD, *PETHREAD;
```

Položka indikující, je-li vlákno „vybrané“ (hodnotou TRUE)

Položka určující pořadové číslo „vybraného“ procesu

Kde „vybraný“ proces je proces, který obsahuje alespoň jedno „vybrané“ vlákno. Položka čísla „vybraného“ procesu slouží k identifikaci „vybraných“ vláken tohoto procesu („vybraná“ vlákna z jednoho procesu mají uloženu stejnou hodnotu v proměnné „ScheduleBucket“).

Pro nastavení vlákna jako „vybraného“ byla upravena implementace nedokumentované API funkce NtSetInformationThread společně se strukturou _THREADINFOCLASS, do které byl přidán „důvod“ volání této funkce:

```
wrk\base\ntos\inc\ps.h:

typedef enum _THREADINFOCLASS {
    …
    ThreadSetScheduleBucket,
    MaxThreadInfoClass
    } THREADINFOCLASS;


wrk\base\ntos\ps\psquery.c:

NTSTATUS NtSetInformationThread(
    __in HANDLE ThreadHandle,
    __in THREADINFOCLASS ThreadInformationClass,
    __in_bcount(ThreadInformationLength) PVOID ThreadInformation,
    __in ULONG ThreadInformationLength)
```

Přidaný „důvod“ pro volání funkce „NtSetInformationThread“

[25] Systémová vlákna by „strádala“ kratším časovým kvantem a delšími prodlevami (hlavně ta, jichž hodnota „bázové“ priority se nachází v „dynamické“ oblasti.

```
{
  …
  switch (ThreadInformationClass) {
  …
  case ThreadSetScheduleBucket:
    …
    ScheduleBucket = *(PULONG)ThreadInformation;

    Thread = PsGetCurrentThread();

    Thread->BucketsEnabled = TRUE;
    Thread->ScheduleBucket = ScheduleBucket;

    return STATUS_SUCCESS;

  default:
    return STATUS_INVALID_INFO_CLASS;
  }
}
```

Podle „důvodu“ volání této funkce se upravují příslušné hodnoty ve struktuře dodaného vlákna („ThreadHandle“)

Označení vlákna jako „vybraného“

Podle „důvodu“ volání funkce NtSetInformationThread jsou upraveny některé parametry předaného vlákna. Pokud byl tento důvod ThreadSetScheduleBucket, označí se vlákno jako „vybrané“. Pořadové číslo „vybraného“ procesu je předáno v parametru ThreadInformation.

Pro výpočet upravené délky časového kvanta je nutné znát celkový počet „vybraných“ procesů a vláken společně s počty „vybraných“ vláken v jednotlivých „vybraných“ procesech. Tyto jsou uchovány v datové struktuře:

```
wrk\base\ntos\ke\ki.h:

struct {
    ULONG ReadyGroups;

    ULONG ReadyThreads;

    ULONG Buckets[MAXIMUM_SCHEDULE_BUCKETS];

} BucketScheduling;
```

Celkový počet „vybraných vláken

Celkový počet „vybraných“ procesů

Počty „vybraných“ vláken ve „vybraných“ procesech

Tato struktura se dynamicky mění; uchovává pouze „vybraná“ vlákna, která jsou ve stavu „ready“. Změně stavu vlákna na „ready“ předchází volání následujících funkcí:

```
wrk\base\ntos\ke\ki.h: KxQueueReadyThread
wrk\base\ntos\ke\thredsup.c: KiDeferredReadyThread

BucketScheduling.ReadyThreads++;
if (++BucketScheduling.Buckets[eThread->ScheduleBucket] == 0)
  BucketScheduling.ReadyGroups++;
```

Úprava struktury „BucketScheduling“ v případě, že se stav „vybraného“ vlákna bude měnit na „ready“

Kde funkce KxQueueReadyThread je volána například po nucené výměně (kvůli vláknu vyšší priority), nebo vypršení časového kvanta vlákna. Funkce KiDeferredReadyThread slouží k přepnutí vlákna ze stavu „Deferred ready“ do stavu

„standby“ (v případě, že existuje volný procesor, na kterém může dané vlákno běžet, nebo je ve stavu „standby“ tohoto procesoru vlákno s nižší prioritou), nebo „ready“ (jinak). V této funkci také dochází k faktickému zvýšení priority po situacích 4.3.2, 4.3.3, 4.3.4 a 4.3.5.

Naopak změna stavu vlákna z „ready“ na jiný stav předchází volání následujících funkcí:

```
wrk\base\ntos\ke\balmgr.c: KiScanReadyQueues
wrk\base\ntos\ke\ki.h: KiSelectReadyThread
wrk\base\ntos\ke\thredsup.c: KiFindReadyThread

BucketScheduling.ReadyThreads--;
if (--BucketScheduling.Buckets[eThread->ScheduleBucket] == 0)
  BucketScheduling.ReadyGroups--;
```

Úprava struktury „BucketScheduling“ v případě, že se stav „vybraného“ vlákna bude měnit z „ready“

Funkce KiScanReadyQueues slouží k vyhledávání „hladových“ vláken (viz 4.3.6). V případě, že nějaké nalezne, nastaví jeho hodnotu „aktuální“ priority na patnáct (nalezené vlákno je nejprve odstraněno ze současné prioritní fronty „ready“ a poté je opětovně přidáno do nové prioritní fronty „ready“). Funkce KiSelectReadyThread i KiFindReadyThread slouží k vyhledání vlákna, kterému bude přidělen procesor po dobu jeho časového kvanta (jeho stav tedy bude změněn na „running“). Rozdíl mezi nimi je ten, že KiSelectReadyThread je omezena pouze dolní hranicí „aktuální“ priority vlákna, ale KiFindReadyThread omezuje navíc afinita vlákna (seznam procesorů, o které „má zájem“ – viz poznámka pod čarou 22).

Poslední upravenou funkcí je již výše zmíněná KiSelectReadyThread. Tato funkce slouží k vybrání vlákna, kterému bude přidělen procesor a nově i k úpravě délky časového kvanta „vybraným“ vláknům podle zvolené strategie:

```
wrk\base\ntos\ke\ki.h:

PKTHREAD KiSelectReadyThread (IN KPRIORITY LowPriority, IN PKPRCB Prcb)
{
  // … výběr vlákna dané priority
  eThread = CONTAINING_RECORD(Thread, ETHREAD, Tcb);
  if (eThread->BucketsEnabled) {

    ULONG bucket = eThread->ScheduleBucket;

    ULONG newQuantum;
    ULONG normalQuantum = Thread->QuantumReset
    ULONG fairQuantum  = …; // normalQuantum*M/(N*n[bucket]);

    switch(UpgradeScheduling.QuantumChangingAlgorithm)
    {
      case SHARING_CLASSIC:newQuantum = Thread->Quantum;
                           break;
      case SHARING_FAIR:   newQuantum = fairQuantum;
                           break;
      case SHARING_MEAN:   newQuantum = (normalQuantum+fairQuantum) / 2;
                           break;
```

Délka časového kvanta se upravuje pouze „vybraným vláknům.

Výpočet kvanta pro „Fair Share“

Úprava délky časového kvanta v závislosti na vybrané strategii

```
      case SHARING_UNFAIR: if (fairQuantum > 2*normalQuantum)
                                newQuantum = THREAD_QUANTUM;
                           else newQuantum = 2*normalQuantum - fairQuantum;
                           break;
      default:             newQuantum = normalQuantum;

    }

    // Kvantum muze byt pouze v urcitem intervalu
    Thread->Quantum = (SCHAR)newQuantum;
    // Vlakno bude menit stav z ready. Odstraneni ze struktury
  }
  return Thread;
}
```

Omezení délky nového časového kvanta a úprava vlákna

Po vybrání vlákna z fronty dané priority se kontroluje, je-li „vybrané“. Pokud ano, vypočítá se jeho nová délka časového kvanta (tato délka je omezena intervalem <6,127>).

4.2.3 Experimenty s různými strategiemi pro přidělování času procesoru

V této části jsou zpracovány výsledky měření jednotlivých experimentálních strategií pro přidělování časového kvanta vláknům na upraveném jádře (úpravy byly popsány výše). Výstup každé sady měření je zpracován do jedné z tabulek (Tabulka 3, Tabulka 4 a Tabulka 5). Každá z těchto tabulek má celkem pět sloupců. První sloupec určuje počet „vybraných“ vláken „vybraného“ procesu, kterým se měřil celkový přidělený čas. Celkový přidělený čas (ve vteřinách) je obsažen ve sloupcích 2–5 pro jednotlivé experimentální strategie. Aby se eliminovaly extrémní situace, byl každý experiment opakován třikrát a do každé buňky tabulky se uložila průměrná hodnota z těchto měření. Celkem tedy proběhlo šestatřicet (tři tabulky, čtyři strategie, tři opakování) měření každé po dobu sta vteřin.

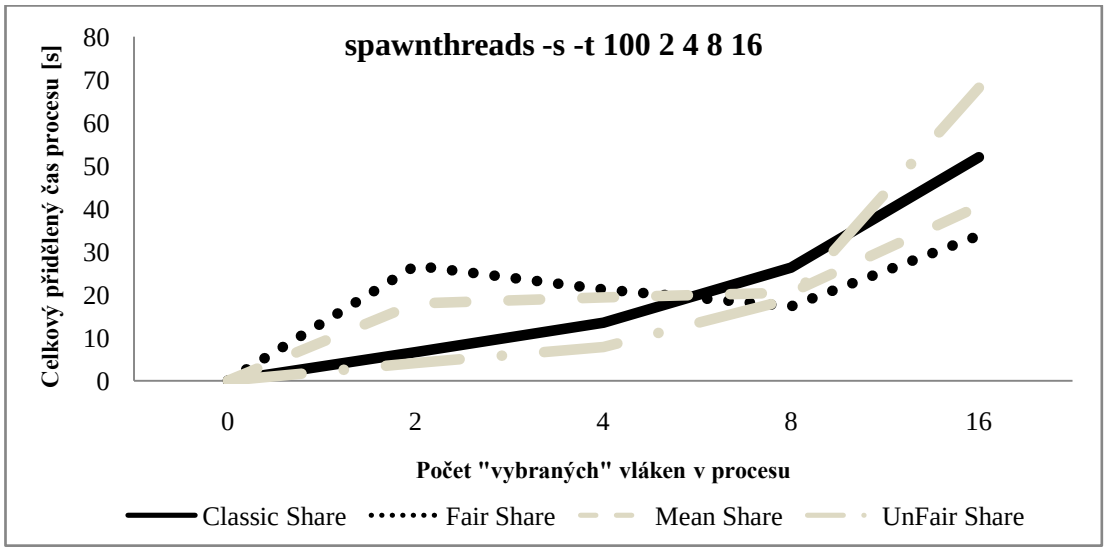
Pro lepší ilustrativnost byl ke každé tabulce vytvořen i graf. Názvy těchto grafů vyjadřují příkaz, kterým by se experimenty spouštěly pomocí nástroje spawnthreads.

Cílem těchto měření není vyvozovat žádné závěry, nebo se snažit interpretovat naměřené hodnoty, ale ukázat, že různé strategie dávají opravdu různé výsledky (kvůli změnám v jádře).

Tabulka 3: 1. sada měření na modifikovaném jádře

Počet „vybraných“ vláken v procesu	*Classic Share*	*Fair Share*	*Mean Share*	*UnFair Share*
2	*6,66*	*26,76*	*17,92*	*4,07*
4	*13,47*	*21,2*	*19,36*	*7,82*
8	*26,29*	*17,25*	*20,48*	*19*
16	*52*	*33,73*	*40,56*	*68,16*

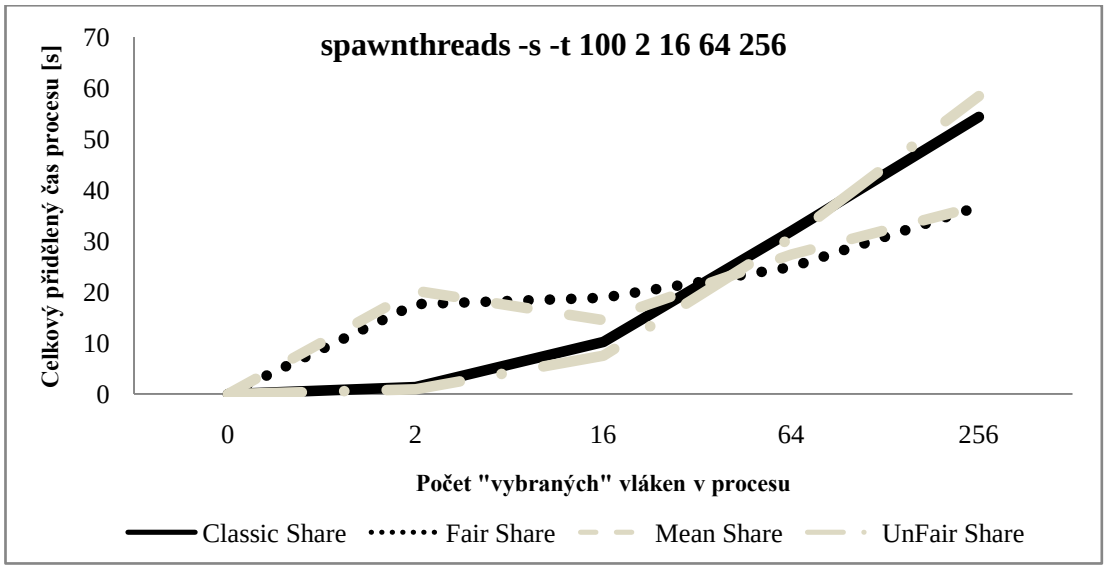
Graf 1: 1.sada měření



Tabulka 4: 2. sada měření na modifikovaném jádře

Počet „vybraných“ vláken v procesu	*Classic Share*	*Fair Share*	*Mean Share*	*UnFair Share*
2	*1,39*	*17,67*	*20,24*	*0,97*
16	*10,25*	*18,92*	*14,47*	*7,5*
64	*31,92*	*24,96*	*27,33*	*30,71*
256	*54,36*	*36,8*	*36,9*	*58,43*

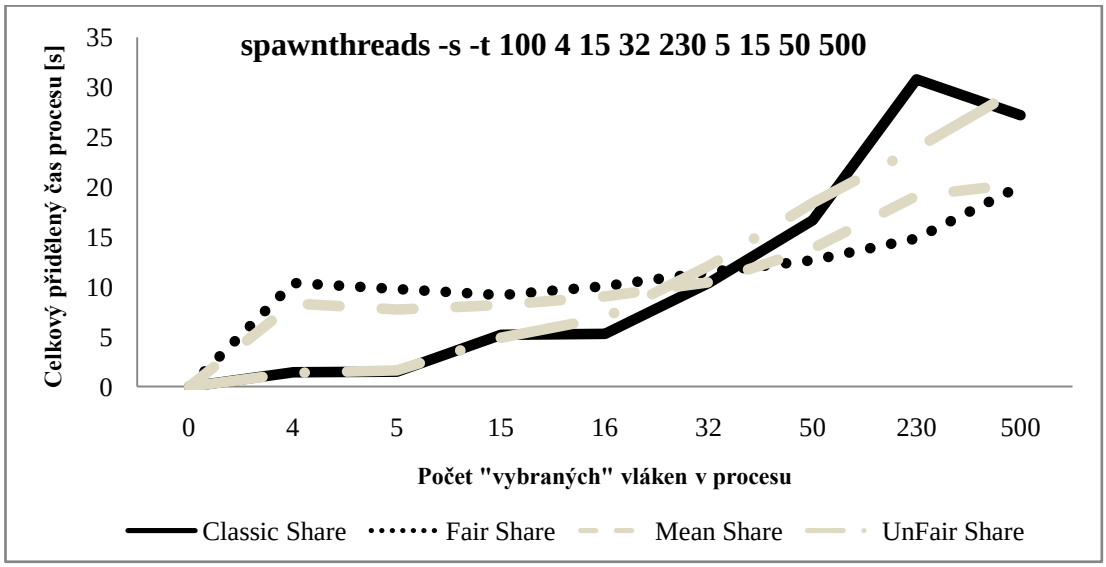
Graf 2: 2.sada měření



Tabulka 5: 3. sada měření na modifikovaném jádře

Počet „vybraných“ vláken v procesu	*Classic Share*	*Fair Share*	*Mean Share*	*UnFair Share*
4	*1,43*	*10,39*	*8,34*	*1,38*
5	*1,51*	*9,77*	*7,73*	*1,65*
15	*5,18*	*9,16*	*8,19*	*4,91*
16	*5,28*	*10,07*	*9,04*	*6,93*
32	*10,38*	*11,49*	*10,45*	*12,09*
50	*16,67*	*12,66*	*13,84*	*18,41*
230	*30,77*	*14,84*	*19,12*	*23,9*
500	*27,19*	*20,1*	*20,33*	*29,92*

Graf 3: 3.sada měření



Uvedené sady experimentů dokázaly, že změnou zdrojových kódů WRK se skutečně změnilo chování systému (při použití upravené verze jádra).

4.3 Zvyšování priority vláken

Pro hledání funkcí, ve kterých dochází k navýšení „aktuální“ priority byla použita kniha (1). Nejprve byly vytipovány okruhy funkcí. Podle jejich kódu a obsaženého komentáře byly poté vybrány ty funkce, ve kterých dochází k navýšení „aktuální“ priority vlákna. Protože ne všechny funkce byly nalezeny hned, přišlo na řadu pátrání v okruhu vytipovaných zdrojových souborů (například u „Vlákna na popředí“). Po nalezení všech dostupných funkcí navyšující „aktuální“ prioritu vlákna byl jejich kód změněn tak, aby bylo možné jednotlivé navýšení vypínat/zapínat. Aby nezůstala žádná funkce opomenuta, byly prohledány komentáře všech základních funkcí pro práci s vlákny. Na závěr byly pomocí experimentů otestovány ty změny, pro které

bylo možné tyto experimenty realizovat (tyto experimenty jsou uvedeny dále v práci).

Protože je priorita zvyšována několika způsoby, je začátek této kapitoly věnován jejich představení spolu s krátkými ukázkami kódu. V dalších částech jsou popsány konkrétní důvody pro zvyšování „aktuální“ priority vlákna.

4.3.1 Způsoby zvyšování „aktuální“ priority vlákna

Různé možnosti zvýšení „aktuální“ priority vlákna zachycuje Tabulka 6.

Tabulka 6: Zvyšování "aktuální" priority vlákna

Krátké jméno události	***Důvod zvýšení priority***	***Prioritní přírůstek***	***Snižování priority***
„Vyhladovění“ vlákna	*Vlákno je ve stavu „ready“ déle než 4 vteřiny*	*15 – „aktuální“ priorita*	*Skokové*
Vlákno na popředí	*Probuzení vlákna procesu na popředí*	*PsPrioritySeparation*	*Skokové*
Speciální událost	*Nastala událost nastavená funkcí NtSetEventBoostPriority*	*„aktuální“ priorita*	*Skokové*
Semafor nebo událost	*Nastala událost, nebo byl uvolněn semafor*	*1*	*Postupné*
Dokončení I/O operace	*Probuzení vlákna po dokončení I/O operace*	*0+ (určuje ovladač)*	*Postupné*
GUI	*Probuzení vlákna kvůli práci s oknem*	*2*	*Postupné*

Horní mezí nové aktuální priority je hodnota patnáct[26].

Jedním z rozdílů mezi navyšováním „aktuální“ priority je způsob jejího následného snižování. Může být buďto postupný (po jedničce), nebo skokový. Další část této kapitoly je věnována praktické implementaci způsobu snižování priority ve WRK.

Struktura vlákna ve WRK obsahuje tři položky týkající se jeho priority:

```
wrk\base\ntos\inc\ke.h:

typedef struct _KTHREAD {
        …
        SCHAR Priority;
        …
        SCHAR BasePriority;
        SCHAR PriorityDecrement;
        …
} KTHREAD, *PKTHREAD, *PRKTHREAD;
```

„aktuální“ priorita

„bázová“ priorita

krok snižování

Pro určení „rychlosti“ snižování priority je použita proměnná PriorityDecrement. Při postupném snižování je hodnota této proměnné rovna nule. V případě snižování skokového je hodnota této proměnné nastavena na požadovanou velikost skoku.

[26] Horní hranice „dynamické“ oblasti priorit (viz 4.1.2).

Funkce, která je volána po vypršení časového kvanta vlákna KiQuantumEnd (v souboru wrk\base\ntos\ke\dpcsup.c) počítá i novou hodnotu „aktuální“ priority vlákna. Tato hodnota je dána výstupem z funkce KiComputeNewPriority(Thread,1) která je implementována následovně:

```
wrk\base\ntos\ke\ki.h:

FORCEINLINE SCHAR KiComputeNewPriority (
    IN PKTHREAD Thread,
    IN SCHAR Adjustment)
{

    SCHAR Priority;

    Priority = Thread->Priority;

    if (Priority < LOW_REALTIME_PRIORITY) {
        Priority = Priority - Thread->PriorityDecrement - Adjustment;
        if (Priority < Thread->BasePriority) {
            Priority = Thread->BasePriority;
        }

        Thread->PriorityDecrement = 0;
    }

    return Priority;
}
```

Priorita je snižována jen vláknům z „dynamickém“ oblasti priorit

„Aktuální“ priorita vlákna nesmí být menší než priorita „bázová“

Je tedy implementováno, že po každém vypršení časového kvanta se hodnota „aktuální“ priority vlákna sníží o 1 + PriorityDecrement. Samozřejmě i zde platí omezení, že hodnota „aktuální“ priority vlákna nesmí být menší, než hodnota priority „bázové“.

4.3.2 Dokončení I/O operace (1 pp. 349-350)

Správce procesů v NTOS zvyšuje prioritu vláknům po dokončení některých I/O operací. Tím se zvýší šance těchto vláken na brzké (nebo okamžité) získání procesoru. Doporučené hodnoty zvýšení priority se nacházejí v hlavičkovém souboru WRK\base\ntos\inc\exboosts.h (některé doporučené hodnoty uvádí Tabulka 7), ale o skutečnou změnu určuje ovladač příslušného zařízení. Tento ovladač specifikuje zvolenou hodnotu jako parametr funkce IoCompleteRequest, kterou zavolá po dokončení požadované I/O operace.

Tabulka 7: Doporučené navýšení priority některých I/O operací

Zařízení:	***Doporučené hodnoty zvýšení priority:***
Disk, CD-ROM, paralelní port, video	*1*
Síť, pojmenovaná roura, sériový port	*2*
Klávesnice, myš	*6*
Zvuk	*8*

Po dokončení této operace se hodnota „aktuální“ priority příslušného vlákna nastaví podle výrazu:

```
Maximum(„bázová“ + „navýšení“; “aktuální“)
```

Kde „navýšení“ je hodnota která byla předána jako parametr funkce IoCompleteRequest. V případě, že se priorita navyšuje je následné snižování priority postupné (viz 0).

Stále platí to, co bylo uvedeno v kapitole 4.1.2 (ke zvýšení „aktuální“ priority dochází pouze u vláken, jejichž hodnota „bázové“ priority se nachází v „dynamické“ oblasti). Navíc platí omezení o maximální hodnotě „aktuální“ priority. Například vláknu s prioritou čtrnáct se „aktuální“ priorita zvýší maximálně na patnáct. Vláknu s prioritou patnáct se „aktuální“ priorita již nenavyšuje.

Funkce IoCompleteRequest je ve WRK implementována voláním funkce IofCompleteRequest, ve které se pomocí ukazatele na funkci pIofCompleteRequest volá funkce IopfCompleteRequest. Konečně v této funkci jsou provedeny požadované operace. Pro účely dynamického vypínání/zapínání zvýšení priority v důsledku dokončení I/O operace byla rozhodující podmínka přidána do původní funkce (IoCompleteRequest).

Při vypínání/zapínání zvýšení priority v důsledku dokončení I/O operace nebyla zjištěna žádná měřitelná změna a ani nedošlo k navýšení testovacích proměnných. Je to nejpravděpodobněji tím, že skutečná funkce volaná po dokončení I/O operace není součástí WRK ve formě zdrojového kódu, ale pouze jako binární soubor.

4.3.3 Semafor nebo událost (včetně speciální události) (1 p. 350)

Po probuzení vlákna čekajícího na událost (zavoláním API funkce SetEvent nebo PulseEvent) nebo semafor (zavoláním API funkce ReleaseSemaphore) je hodnota jeho „aktuální“ priority nastavena podle výrazu:

```
Maximum(„aktuální“; „bázová“ + 1[27])
```

Uvedeným důvodem tohoto navýšení priority je zlepšení situace čekajících vláken oproti vláknům, které stále vyžadují procesor.

[27] Ve výše zmíněném hlavičkovém souboru exboost.h jsou pro tato navýšení vyhrazeny speciální konstanty „EVENT_INCREMENT“ a „SEMAPHORE_INCREMENT“.

Tabulka 6 uvedla, že po tomto navýšení priority následuje postupné snižování, ale protože se priorita zvyšuje pouze o jedničku, není mezi těmito strategiemi (postupné a skokové) žádný rozdíl. Ve WRK je toto navýšení implementováno ve stejném místě zdrojového kódu jako „Dokončení I/O operace“ a „GUI“, a proto je následné klesání zařazeno mezi postupné.

Funkce uvedené v knize (1) jsou předehrou pro zvýšení priority. Jejich implementaci čtenář najde ve zdrojových souborech:

- wrk\base\ntos\ke\eventobj.c pro KeSetEvent a KePulseEvent.
- wrk\base\ntos\ke\semphobj.c pro KeReleaseSemaphore).

Speciální událost se vyvolá funkcemi NtSetEventBoostPriority (využívají například kritické sekce v ntdll.dll) a KeSetEventBoostPriority. Po probuzení vlákna čekajícího na událost jednou z uvedených funkcí se jeho „aktuální“ priorita zdvojnásobí (pokud není větší než třináct – tuto podmínku kontroluje funkce KiDeferredReadyThread zdrojového souboru wrk\base\ntos\ke\thredsup.c). Následuje implementace funkce pro vyvolání speciální události ve WRK:

```
wrk\base\ntos\ke\eventobj.c:

VOID KeSetEventBoostPriority (
    __inout PRKEVENT Event,
    __in_opt PRKTHREAD *Thread)
{
    //inicializace

    if (IsListEmpty(&Event->Header.WaitListHead) != FALSE) {
        …
    } else {

        WaitBlock = CONTAINING_RECORD(…);

        if (WaitBlock->WaitType == WaitAll) {
            …
        } else {

            WaitThread = WaitBlock->Thread;
            if (ARGUMENT_PRESENT(Thread)) {
                *Thread = WaitThread;
            }

            …

            WaitThread->AdjustIncrement = CurrentThread->Priority;
            WaitThread->AdjustReason = (UCHAR)AdjustBoost;

            KiReadyThread(WaitThread);
        }
    }
    // finalizace
}
```

Do této větve vždy projde „fast mutex“ a „resource“

Zvýšení priority o hodnotu „aktuální“ priority čekajícího vlákna

Snížení bude skokové (nastaveno následně ve funkci KiDeferredReadyThread

Navýšení priority pomocí této funkce využívá systém. Proto v této práci není implementováno vypínání/zapínání tohoto navýšení.

4.3.4 Vlákno na popředí (1 p. 351)

Po probuzení vlákna na popředí aplikace na popředí dochází k novému nastavení „aktuální“ priority vlákna podle následujícího výrazu:

```
Maximum(„bázová“ + „navýšení“ + PsPrioritySeparation; „aktuální“)
```

Kde „navýšení“ je hodnota jiného navýšení priority v důsledku probuzení vlákna („GUI“, „Semafor nebo událost“ a „Dokončení I/O). V případě, že se priorita opravdu navýšila, se do proměnné PriorityDecrement nastaví hodnota daná výrazem:

```
Minimum(Maximum(15 – „bázová“ – „navýšení“; 0); PsPrioritySeparation)
```

Uvedeným důvodem pro toto navýšení je zlepšit odezvu interaktivních aplikací přidáním priority vláknům na popředí (zvýhodnění oproti vláknům na pozadí).

Kniha (1) popisuje implementaci tohoto navýšení s několika rozdíly oproti WRK:

- (1): Popsané navýšení priority nelze standardně vypnout.
- (1): Zvýšení priority má na starost funkce KiUnwaitThread.

Naproti ve WRK zvýšení implementuje funkce KiDeferredReadyThread, která se stará o zvýšení priority probuzeného vlákna. Navíc je toto zvýšení možné vypnout standardním způsobem (pomocí API funkce SetThreadPriorityBoost s příslušnými parametry).

Spolu s funkcí KiDeferredReadyThread je zde pro úplnost uvedena i kostra funkce KiUnwaitThread zajišťující probuzení vlákna a přípravu navýšení jeho priority:

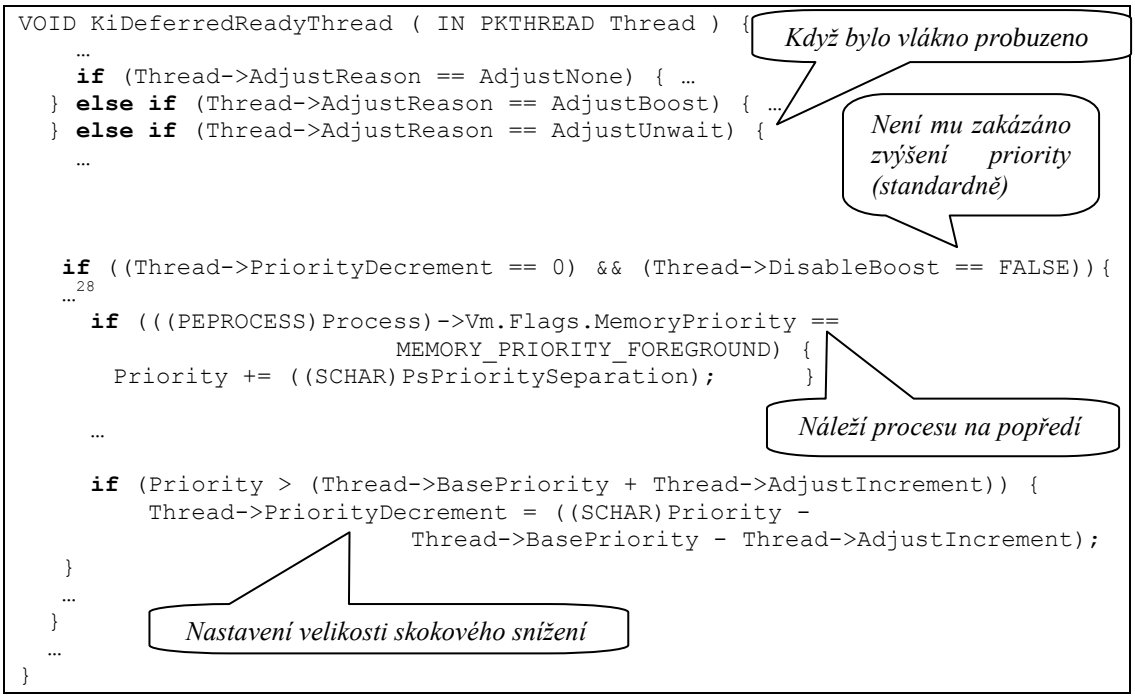
```
wrk\base\ntos\ke\waitsup.c:

VOID FASTCALL KiUnwaitThread (
    IN PRKTHREAD Thread,
    IN LONG_PTR WaitStatus,
    IN KPRIORITY Increment)
{
    …
    Thread->AdjustIncrement = (SCHAR)Increment;
    Thread->AdjustReason = (UCHAR)AdjustUnwait;
    …
}

e:\wrk\base\ntos\ke\thredsup.c:
```

Nastavení navýšení priority. Samotné navýšení je implementováno v „KiDeferredReadyThread“

Důvod navýšení je probuzení vlákna



Všechna zvýšení priority vlákna po probuzení jsou implementována funkcí KiDeferredReadyThread, ale v této funkci je již není možné od sebe odlišit (s výjimkou zvýšení priority po speciální události).

Následující praktická ukázka zvyšování priority probuzených vláken na popředí, vychází z experimentu uvedeného v knize (1 pp. 351-353). Tato ukázka používá dvě externí aplikace (CPU Stress a Performance Monitor, které jsou součástí „Windows 2000 Professional Resource Kit“ a přiloženého DVD). Její realizaci popisují následující kroky:

1) V „Control Panel“ (Ovládací panely) vybrání položky „System“ (Systém), poté kliknutí na záložku „Advanced“ a následně na tlačítko „Setting“ v části „Performace“. Poté přepnutí do záložky „Advanced“. V této záložce vybrání volby „Programs“ v části „Processor scheduling“. Tím se hodnota proměnné PsPrioritySeparation nastaví na dva.
2) Spuštění nástroje CPU Stress.
3) Spuštění nástroje Performance Monitor.
4) Vyvolání dialogu „Add to chart“ stisknutím tlačítka „Add counter“ (nebo pomocí klávesové zkratky Ctrl+I) v panelu nástrojů Performance Monitoru.

[28] Na tomto místě se nachází příkaz Priority = Thread->BasePriority + Thread->AdjustIncrement, který zvýší prioritu, vzhledem k bázové o hodnotu, nastavenou ve funkci KiUnwaitThread.

5) V tomto dialogu vybrání „Thread“ v oblasti „Object“ a následné vybrání „CPUSTRESS ==> 1“ v oblasti „Instance“.

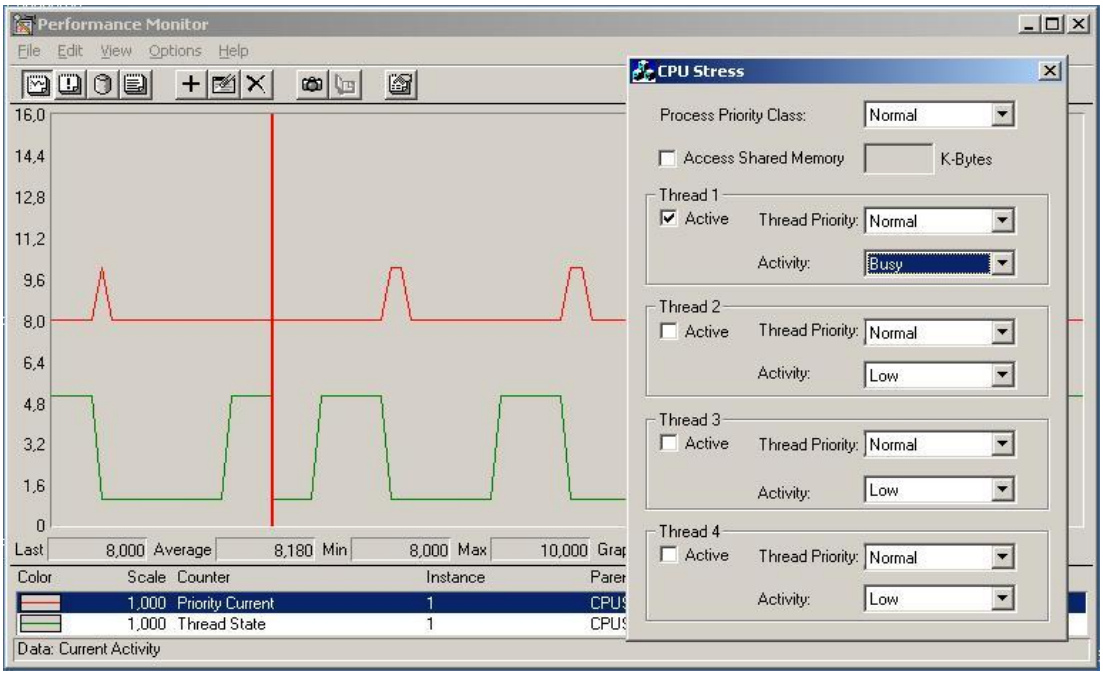
6) Poté v části „Counter“ vybrání „Priority Current“ a stisknutí tlačítka „Add“. Dále ve stejné části vybrání možnosti „Thread State“ a další stisknutí „Add“. Nakonec stisknutí tlačítka „Done“.

7) Kliknutí na tlačítko „Options“ (nebo pomocí klávesové zkratky Ctrl+O) v panelu nástrojů Performance Monitoru. Změna hodnot „Vertical Maximum“ na šestnáct a „Interval“ na 0,01. Nakonec stisknutí „OK“.

8) Přenesení programu CPU Stress do popředí.

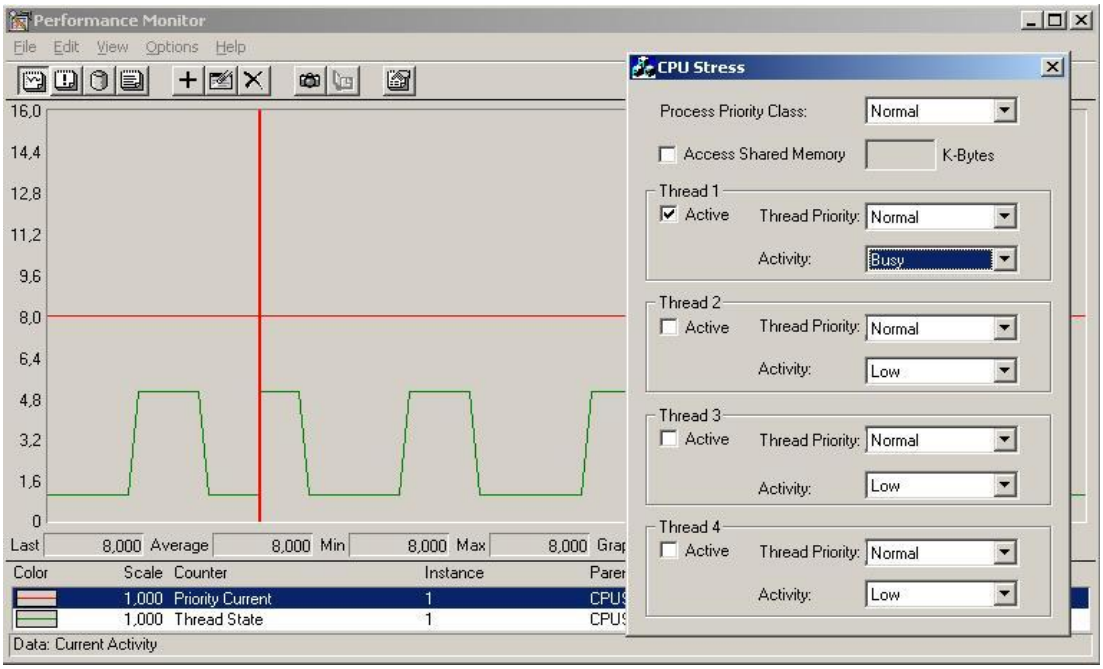
V případě, že je aktivní zvýšení priority po probuzení vlákna procesu na popředí uvidíte, jak se „aktuální“ priorita periodicky zvyšuje o dva a následně klesne zpět (viz Obrázek 9). Je-li toto zvýšení priority vypnuto (Programem popsaným v kapitole 3 zrušením zaškrtnutí tlačítka „Enable Process Foreground boost“ v záložce „options“), se priorita nezmění (viz Obrázek 10).

Obrázek 9: Process foreground boost enabled



Změnou položky „Activity“ u aktivního vlákna programu CPU Stress se změní i interval „probouzení“ tohoto vlákna. Nastavením na „Maximum“ se naopak vlákno nikdy neuspí a tedy nemůže dojít k jeho probuzení (a zvýšení priority).

Obrázek 10: Process foreground boost disabled



Tento experiment ukázal, že výše popsaná část zdrojového kódu WRK skutečně slouží k navýšení priority „Vlákno na popředí“. Také ukázal, že pomocí navržené aplikace WRKExperimentalEnv lze toto navýšení vypínat/zapínat.

4.3.5 GUI (1 p. 353)

Po probuzení vlákna kvůli práci (například po přijetí zprávy) s jeho grafickým oknem je hodnota jeho „aktuální“ priorita nastavena na hodnotu danou výrazem:

```
Maximum(„aktuální“; “bázová“ + 2)
```

Toto navýšení je zakomponováno v systému pro práci s okny (Win32k.sys). Důvodem uvedeným v knize (1) je další úroveň zlepšení odezvy grafických aplikací.

Protože je toto navýšení zabudování uvnitř systému pro práci s okny není možné jej vypínat/zapínat úpravou zdrojového kódu WRK. Existuje způsob, kterým lze toto omezení obejít. Jedná se však o nespolehlivý způsob vycházející z toho, že k veškerému navýšení priority po probuzení vlákna dochází v jedné části funkce KiDeferredReadyThread (popsané výše). V případě testu na požadované zvýšení priority vlákna o dva uvnitř této funkce by znamenalo nejen odfiltrování tohoto navýšení, ale i odfiltrování navýšení priority po dokončení I/O operace (v případě, že by se priorita měla zvyšovat o dva). Z tohoto důvodu není tato metoda použita.

4.3.6 „Vyhladovění“ vlákna (1 pp. 354-355)

Představme si následující situaci se třemi vlákny:

- **Vlákno 1 –** Je ve stavu „ready“, hodnota „bázové“ i „aktuální“ priority je rovna čtyřem.
- **Vlákno 2 –** Je ve stavu „running“, hodnota jeho „bázové“ i „aktuální“ priority je sedm.
- **Vlákno 3 –** Je ve stavu „waiting“ (čeká na objekt, na který vlastní zámek Vlákno 1), hodnota jeho “bázové“ i „aktuální“ priority je jedenáct.

Po vypršení časového kvanta druhého vlána, získá toto vlákno další časové kvantum (jeho „aktuální“ priorita bude nejvyšší ze všech vláken ve stavu „ready“). Tím zabrání prvnímu vláknu, aby uvolnilo objekt, na který čeká vlákno třetí. Tedy obě zbývající vlákna jsou zablokována. Tato situace by narušila fungování principu systému priorit, ve kterém vlákno s nejvyšší „aktuální“ prioritou získá procesor.

Pro předcházení zablokování vláken je v NTOS implementováno speciální vlákno tzv. „balance set manager“, jehož sekundární[29] funkcí je vyhledávat tzv. „hladová“ vlákna (Vlákno je „hladové“ v případě, že je ve stavu „ready“ déle než čtyři vteřiny.) Každou jednu vteřinu je vlákno „balance set manageru“ probouzeno a „hladovým“ vláknům, která nalezne, zvýší dočasně hodnotu „aktuální“ priority na patnáct a prodlouží jejich kvantum na dvojnásobek (ve Windows 2000 a Windows XP) nebo čtyřnásobek (ve Windows Server 2003) jejich standardního časového kvanta. Po vypršení zvýšeného časového kvanta je hodnota „aktuální“ priority vrácena zpět na hodnotu priority „bázové“.

„Balance set manager“ ve skutečnosti neprohledává všechny fronty „ready“ během každého svého běhu, ale pouze šestnáct vláken z těchto front. Poté si uloží pozici vlákna, u kterého skončil a začne na ní v příštím běhu. Dalším limitem je zvýšení priority maximálně deseti vláknům. Tyto opatření jsou zavedena proto, aby se procesor neblokoval na dlouhou dobu, protože „bázová“ priorita vlákna „balance set manageru“ je šestnáct (již v oblasti „realtime“).

V implementaci „balance set manageru“ ve WRK jsou z hlediska navýšení priority vláken důležité dvě funkce. První z nich je samotná funkce vlákna (tedy funkce, která je předána jako parametr obdobě API funkce CreateThread). Druhá funkce

[29] Primárně slouží ke správě paměti.

slouží k vyhledávání, navyšování priority a prodloužování časového kvanta „hladovým“ vláknům. Implementace první z nich je následující:

wrk\base\ntos\ke\balmgr.c:

```
VOID KeBalanceSetManager ( IN PVOID Context)
{

    LARGE_INTEGER DueTime;
    KTIMER PeriodTimer;
    KDPC ScanDpc;
    PVOID WaitObjects[MaximumObject];

    KeSetPriorityThread(KeGetCurrentThread(),LOW_REALTIME_PRIORITY);

    KeInitializeTimerEx(&PeriodTimer, SynchronizationTimer);
    KeInitializeDpc(&ScanDpc, &KiScanReadyQueues, &KiReadyScanLast);


    DueTime.QuadPart = - PERIODIC_INTERVAL;
    KeSetTimerEx(&PeriodTimer, DueTime, PERIODIC_INTERVAL/10000, &ScanDpc);

    WaitObjects[TimerExpiration] = (PVOID)&PeriodTimer;

    do {

        Status = KeWaitForMultipleObjects(…);

        switch (Status) {

        case TimerExpiration:
             …
        case WorkingSetManagerEvent:
             …
        default:
            KdPrint(("BALMGR: Illegal wait status, %lx =\n", Status));
            break;
        }

    } while (TRUE);
    return;
}
```

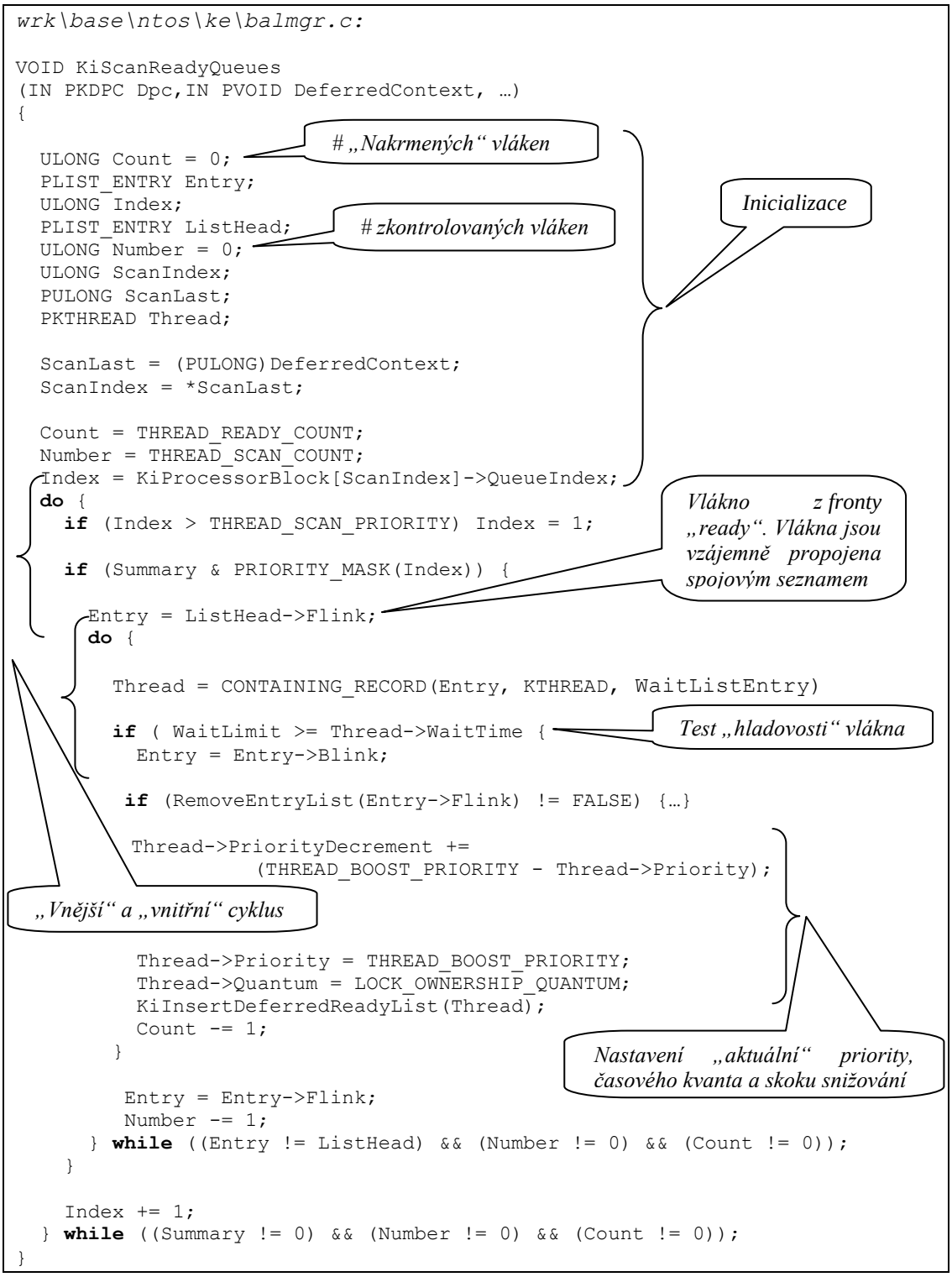
Inicializace „Balance set manageru“

Nastavení funkce pro skenování front „ready“

Po uplynutí 1 vteřiny se volá funkce pro skenování front „ready“

V inicializační části se nastaví priorita tohoto vlákna na šestnáct společně s nastavením objektů popisující práci tohoto vlákna. Pro potřebu zvýšení priority vlákna se nastaví časovač společně s funkcí (KiScanReadyQueues), která bude periodicky volána po vypršení jedné vteřiny. Objekt časovače je vložen do pole objektů (na které se následně čeká). Poté v nekonečné smyčce čeká na objekt a podle typu objektu zpracuje příslušné funkce.

Funkce pro skenování front „ready“:



V inicializační části se nejprve zjistí procesor, u kterého budou „hladová“ vlákna vyhledávána a číslo poslední fronty při minulém prohledávání.

Vnějším cyklus probíhá přes celkový počet vláken ve frontách (Summary), počet možných zkontrolovaných vláken (Number) a počet možných upravených vláken (Count). V tomto cyklu funkce získá ukazatel na čelo fronty vláken, ve které se aktuálně nachází.

Vnitřní cyklus postupuje přes vlákna z této fronty, počet možných zkontrolovaných vláken a počet možných upravených vláken. V případě, že je nějaké vlákno dané fronty „hladové“, vymaže se z ní a po úpravě jeho „aktuální“ priority a délky časového kvanta se vloží do seznamu vláken ve stavu „Deferred ready“

Po ukončení obou cyklů je uložen index aktuální fronty a iterováno číslo procesoru, u nějž se budou vyhledávat „hladová“ vlákna při příštím běhu.

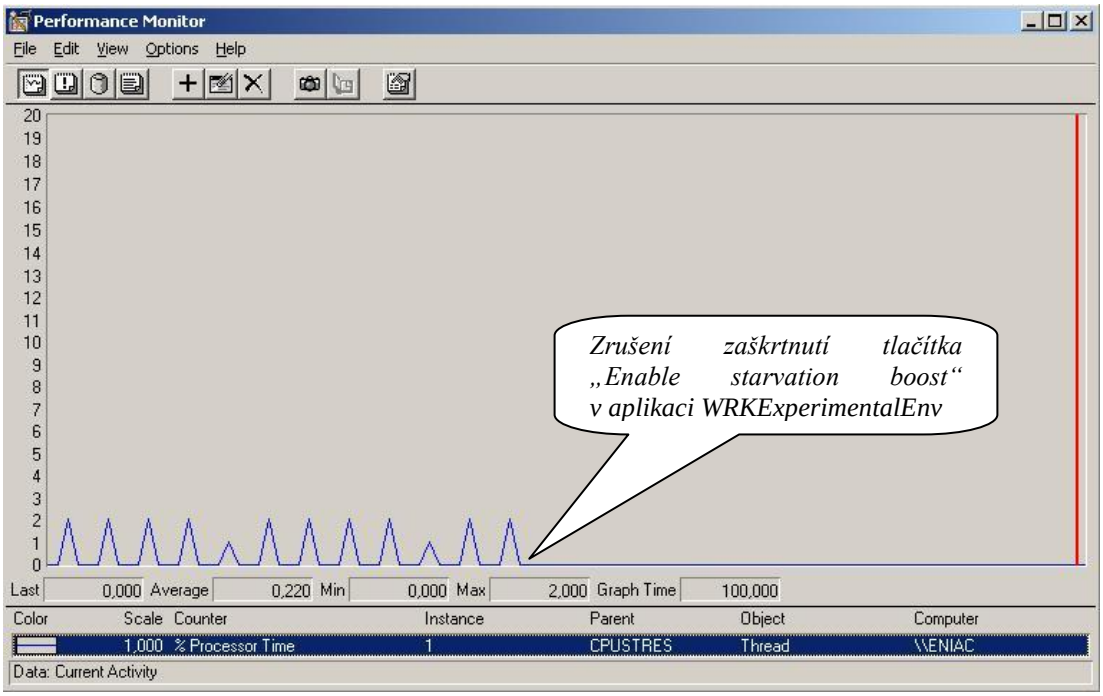
Následující praktická ukázka zvyšování priority „hladových“ vláken vychází z experimentu uvedeného v knize (1 pp. 355-356). Tato ukázka používá dvě externí aplikace (CPU Stress a Performance Monitor, které jsou součástí „Windows 2000 Professional Resource Kit“ a přiloženého DVD). Její realizaci popisují následující kroky:

1) Spuštění CPU Stress. Změna úrovně aktivity aktivního vlákna (standardně „Thread 1“) z „Low“ na „Maximum“. Změna priority tohoto vlákna z „Normal“ na „Below Normal“.
2) Spuštění nástroje Performance Monitor.
3) Vyvolání dialogu „Add to chart“ stisknutím tlačítka „Add counter“ (nebo pomocí klávesové zkratky Ctrl+I) v panelu nástrojů Performance Monitoru.
4) V tomto dialogu vybrání „Thread“ v oblasti „Object“ a následné vybrání „CPUSTRESS ==> 1“ v oblasti „Instance“.
5) V části „Counter“ vybrání „% Processor Time“ a stisknutí tlačítka „Add“. Poté stisknutí tlačítka „Done“.
6) Zvýšení priority Performace Monitoru na „realtime“ pomocí Správce úloh.
7) Spuštění další kopie Cpustress.exe. V této kopii nastavení aktivity aktivního vlákna z „Low“ na „Maximum“.
8) Přepnutí zpět do Performace Monitoru.

V případě, že je aktivní zvýšení priority v důsledku „vyhladovění“ vlákna, je vidět aktivita procesoru cca každé čtyři vteřiny. Je to dáno dočasným zvýšením priority na patnáct právě v důsledku „vyhladovění“ vlákna. (viz levá část Obrázek 11). Je-li toto zvýšení priority vypnuto (Programem popsaným v kapitole 3 zrušením zaškrtnutí

tlačítka „Enable Starvation boost“ v záložce „options“), nedojde k žádnému zvýšení priority (viz pravá část Obrázek 11).

Obrázek 11: Experiment s vyhladověním



Tento experiment ukázal, že výše popsaná část zdrojového kódu WRK skutečně navyšuje prioritu „vyhladovělým“ vláknům. Také ukázal, že pomocí navržené aplikace WRKExperimentalEnv lze toto navýšení vypínat/zapínat.

5 ZÁVĚR

Všechny vytyčené cíle byly v této práci splněny. Nejprve jsem se věnoval možnostem WRK spolu otestováním a dokumentací jediného oficiálního příkladu „Fair Share“. Dalším krokem bylo přidání dalších experimentálních strategií pro přidělování časového kvanta vláknům a jejich vzájemné porovnání. Na uvedených experimentech se prokázalo, že různé strategie přidělují čas procesoru různě. Posledním krokem v rámci úpravy zdrojového kódu WRK bylo nalezení, analyzování, zdokumentování a upravení těch částí zdrojových kódu, které souvisí se zvyšováním „aktuální“ priority vlákna. Pomocí provedených úprav je možné různé způsoby navýšení priority vypínat/zapínat.

Aby bylo možné všechny provedené změny jednoduše testovat, byl vytvořen grafický program, který komunikuje s vlastním ovladačem jádra pomocí API funkce DeviceIoControl a umožňuje dynamicky měnit chování upraveného jádra. K tomuto programu byla vytvořena uživatelská i programátorská dokumentace, která je součástí této práce.

Možností použití WRK z hlediska experimentování s jádrem NTOS je nepřeberné množství. Možným rozšíření této práce může být například:

- Úprava konstant, které určují velikost navýšení priority.
- Přidání dalšího způsobu navýšení priority.
- Úprava rozsahu „dynamických“ a „realtime“ oblastí priorit a měření následné změny v chování jádra.
- Změna způsobu výběru vlákna, které získá procesor

Implementace zmíněných bodů ve WRK by mohla být dalším krokem v prozkoumávání chování vláken v NTOS.

6 BIBLIOGRAFIE

1. **Solomon, David A. and Russinovich, Mark E.** *Windows Internals 4th Edition.* s.l. : Microsoft Press, 2005.

2. **Solomon, David A., Russinovich, Mark E. a Polze, Andreas.** CRK in MSDNAA Curriculum Repository. [Online] Únor 2006. http://www.msdnaa.net/curriculum/pfv.aspx?ID=6191.

3. Microsoft Academic Program. [Online] Microsoft. http://www.microsoft.com/resources/sharedsource/windowsacademic/default.mspx.

4. *Microsoft VirtualPC 2007.* [Online] Microsoft. http://www.microsoft.com/windows/products/winfamily/virtualpc/default.mspx.

5. Debugging tools for Windows – Overview. [Online] Microsoft. http://www.microsoft.com/whdc/DevTools/Debugging/default.mspx.

6. *.NET Framework Version 2.0 Redistributable Package (x86).* [Online] Microsoft. [Citace: 1. 5 2008.] http://www.microsoft.com/downloads/details.aspx?FamilyID=0856EACB-4362-4B0D-8EDD-AAB15C5E04F5&displaylang=en.

7. .NET Column: Calling Win32 DLLs in C# with P/Invoke. [Online] Microsoft. [Citace: 7. 4 2007.] http://msdn.microsoft.com/en-us/magazine/cc164123.aspx.

8. *Events (C# Programming Guide).* [Online] Microsoft. [Citace: 7. 4 2008.] http://msdn.microsoft.com/cs-cz/library/awbftdfh(en-us).aspx.

9. **Jákl, Vojtěch J.** Programování pro Windows. [Online] [Citace: 20. 2 2008.] http://edgar.ms.mff.cuni.cz/winprog/default.aspx.

10. Windows NT – Wikipedia, the free encyclopedia. *Wikipedia.* [Online] [Cited: 4 2, 2008.] http://en.wikipedia.org/wiki/Windows_NT.

11. Windows Research Kernel @ HPI - news, howtos and labdescriptions. [Online] Hasso Plattner Institut. http://www.dcl.hpi.uni-potsdam.de/research/WRK/.

7 OBSAH PŘILOŽENÉHO DVD

Na přiloženém DVD naleznete:

- Obrázky použité v této práci a práci samotnou ve formátu pdf v podadresáři Bakalářská práce.
- Přeložený program WRKExperimentalEnv, který je popsaný v kapitole 3 spolu s dalšími nástroji pro experimenty s upraveným jádrem WRK (jako například Performance Monitor) v podadresáři Měřící nástroje.
- .NET Framework verze 1.1 a 2.0, instalační balíček VirtualPC 2007, windbg, WDK a „doximentaci“ WRK v podadresáři Prerekvizity.
- Zdrojové kódy upraveného WRK v podadresáři WRK.
- Zdrojové kódy programu WRKExperimentalEnv v podadresáři WRK_Experiment_Enviroment.
- Nově přeložené upravené jádro WRK v podadresáři WRKEXE.
- Obraz disku s nainstalovaným Windows Server 2003 SP1 (x86) spolu s upraveným jádrem WRK a nástroji pro experimenty (WRKExperimentalEnv, Performance Monitor a CPU Stress). Tento obraz byl používán pro testování změn WRK během celého projektu. Experimenty v této práci byly prováděny pod tímto systémem.