České vysoké učení technické v Praze
Fakulta elektrotechnická

Bakalářská práce

Návrh programového rozhraní pro řízení mikropolohovací platformy z PC v reálném čase

Praha, 2007 **Autor: František Hrdina**

Prohlášení

Prohlašuji, že jsem svou bakalářskou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

V Praze dne _____ _____

podpis

Poděkování

Děkuji vedoucímu bakalářské práce Ing. Ondřeji Holubovi za pomoc i přínosnou kritiku. Děkuji rodičům za podporu v době studia.

Abstrakt

Cílem této bakalářské práce je vytvořit programové rozhraní pro řízení mikropolohovací platformy z PC v reálném čase. Nejprve se práce zabývá využitím API operačního systému Windows XP pro úlohy v reálném čase. V další části navrhuje možnosti rozšíření jádra operačního systému a poskytuje seznam hardwarových prostředků k tomuto vhodných. Uvádí dostupná komerční rozšíření. Poté je popsána implementace zvoleného řešení.

Abstract

The aim of this bachelor thesis is to create software interface for real time control of a micropositioning stage. At first utilization of Windows XP operating system's API for real time tasks is dealt. In the next part means of extension of operating system's kernel are proposed and list of suitable hardware resources for the extension is provided. Available commercial extensions are shown. Finally, the implementation of chosen solution is described.

Obsah

Seznam obrázků **vii**

Seznam tabulek **ix**

1 Úvod **1**

2 Windows XP **3**

2.1 Architektura 3

2.1.1 Plánovač a správa priorit 4

2.2 User mode 4

2.2.1 Charakteristika 4

2.2.2 Časovače 5

2.2.3 Měření času 6

2.2.4 Vývojové prostředky 7

2.3 Kernel mode 8

2.3.1 Charakteristika 8

2.3.2 Časovače 8

2.3.3 Měření času 9

2.3.4 Vývojové prostředky 9

3 Rozšíření Windows XP **11**

3.1 Zdroje latencí 11

3.2 Přerušení 12

3.2.1 APIC a PIC 12

3.2.2 IRQ a INTI 16

3.2.3 Vyřizování požadavků přerušení 17

3.3 ACPI 20

3.3.1 SMM 20
3.4 Hardwarové časovače 21
3.4.1 LAPIC Timer 23
3.4.2 Performance Monitoring MSR 25
3.5 Hardwarové prostředky pro měření času 26
3.5.1 TSC 26
3.5.2 PM timer 27
3.6 Komerční rozšíření 28
3.6.1 Ardence RTX 28

4 Řízení mikropolohovací platformy z PC v reálném čase 31
4.1 Měření napětí 31
4.2 Řízení piezomotorku 32
4.3 Časové požadavky 32
4.3.1 Multimedia Timers 33
4.3.2 Aktivní čekání 34
4.4 Zhodnocení 36

5 Závěr 39

A Zkratky I

B Příkazy Windbg III

C Obsah přiloženého CD V

Seznam obrázků

3.1 SCI — vyvolané přetečením PM Timeru. 21
3.2 RTX — časovač s periodou 1 ms. 29
3.3 RTX — časovač s periodou 1 ms, CLI. 30

4.1 MM Timer — časovač s periodou 1 ms. 34
4.2 Busy waiting v Matlabu — časovač s periodou 1 ms. 35
4.3 Busy waiting — časovač s periodou 1 ms. 36

Seznam tabulek

3.1 PIC 14
3.2 APIC 15
3.3 ACPI MADT 17
3.4 Mapování zdrojů přerušení na vstupy PIC a IO APIC 18
3.6 ACPI FADT 28

Kapitola 1

Úvod

Použití standardních PC pro úlohy typu sběr dat nebo řízení je problematické. Tyto úlohy jsou časově náročné, běžné operační systémy (General Purpose OS) však byly navržený pro maximální průměrný výkon a ne pro dodržení časových požadavků [1].[1]

Řešením je použít plánování označované jako Cyclic Executive Pattern, hojně využívané v avionických systémech pro svoji determiničnost a jednoduchost[2][2]. Spočívá ve spuštění úlohy (task), která nemůže být přerušena, dokud sama neskončí. Tato úloha může být spouštěna událostí (např. periodicky časovačem), pak se jedná o Triggered Cyclic Executive Pattern[3]. Pokud nalezneme spolehlivý časovač a zaručíme atomicitu běhu úlohy, získáme deterministické plánování při zachování všech výhod běžného operačního systému.

Tato práce zkoumá možnosti implementace periodických i aperiodických časovačů (Timers), měření času (Clocks) a možnosti nepřerušeného běhu úlohy v prostředí Windows XP a obecně na platformě PC. V první části se zabývá využitím API operačního systému a adresuje nedostatky, které znemožňují dosáhnout vysokých frekvenci a dostatečné přesností. V části druhé pak pro tyto problémy navrhuje řešení využívající hardware dostupný v moderních PC.

Výsledkem práce je program pro řízení mikropolohovaci platformy. Vzhledem k dodaným softwarovým prostředkům pro ovládání USB zařízení byla jediná možná volba

operační systém Windows. Proto tato práce nestuduje vlastnosti RTLinuxu.

[1]Použití speciálních real-time operačních systému (RTOS) může být nevhodné nebo přímo nemožné z různých důvodů: nekompatibilita s existujícími programy/ovladači, nové prostředí/API pro programátora, pro jednoduchou úlohu je RTOS zbytečně složitý, různé počítače pro vývoj a spouštění programu, atd.

[2]„Implementation of this pattern is so easy it is hard to get it wrong, at least in any gross way.“ Pokud by se však v kódu úlohy vyskytlo zacyklení, dojde k uváznutí celého systému (protože je úloha nepřerušitelná).

[3]Postup přináší velké problémy, pokud je úloha příliš dlouhá (trvá déle, než je perioda spouštěcí události) a musí být rozdělena na více částí — to ale v případě sběru dat nebo řízení nenastane, v opačném případě je vhodnější použít jiný způsob plánování, implementovaný ve specializovaném RTOS.

Kapitola 2

Windows XP

Windows XP je operační systém z řady Windows NT, která byla na trh prvně uvedena již v roce 1993 a dnes (2007) je *de facto* standard, nejrozšířenější OS pro PC. Je napsán v C, malá část v C++ a hardwarově závislé části v assembleru [1]. Zdrojový kód je uzavřený[1] nebo sdílený (Shared source), např. v rámci Microsoft Windows Academic Program lze získat zdrojové kódy kernelu — Windows Research Kernel [20].

Tato kapitola se nejprve zabývá architekturou Windows XP. Alespoň základní představa o fungovaní Windows je důležitá pro porozumění dalším částem práce. Po úvodu do architektury následuje popis časovačů, které Windows nabízí. Jsou zmíněny vývojové prostředky a doporučená literatura. Text je doplněn ukázkami v jazyce C.

2.1 Architektura

Systém byl původně navržen jako microkernel, z důvodů vyššího výkonu je ale tzv. hybridkernel (ponechává si strukturu microkernelu, služby operačního systému ale běží z výkonnostních důvodů v kernel módu — privilegovaný režim) [1] . Na procesorech x86 využívá možnosti spouštění kódu s různými privilegii — tzv. Ring 0-3. Kernel mode je Ring 0 — má nejvyšší privilegia[3], běžný uživatelský kód pak běží v user módu — Ring 3 — nejnižší privilegia, Ring 1 a 2 není použít.

Windows NT byla od počátku navržena tak, aby fungovala na různých procesorových

[1]Zkoumání nedokumentované implementace lze provádět metodami tzv. reverzního inženýrství (Reverse Engineering). Zejména se jedná o dynamickou analýzu kernel debuggerem WinDbg [2][26] a statickou analýzu pomocí nástroje IDA [25] (disassembler).

[3]Existuje mód procesoru, který má ještě vyšší prioritu než Ring 0 - tedy jakýsi Ring -1, nazývá se System Management mode (SMM).

architekturách. O adaptaci na konkrétní hardware se stará HAL (Hardware Abstraction Layer). V kernel módu běží kromě HALu (micro)kernel (plánovač, multiprocesorová synchronizace, obsluha některých přerušení — s KINTERRUPT — ostatní obsluhuje HAL, ...), exekutiva (služby OS: správa paměti, správa procesů, ...), ovladače zařízení (sem patří i filesystem) a grafický systém. Exekutiva poskytuje user módu takzvané nativní API, to nevyužívají uživatelské programy přímo, ale prostřednictvím tzv. subsystémů. Subsystém je např. Win32 (část tohoto subsystému běží ale v kernel módu), POSIX jako volitelný doplněk Services for UNIX, OS/2, 16bit DOS. Win32 musí běžet vždy, poskytuje totiž navíc služby ostatním subsytemům.

2.1.1 Plánovač a správa priorit

Ze všech komponent Windows XP jsou pro účely této kapitoly nejdůležitější plánovač a způsob správy priorit. Plánovač Windows NT je prioritou řízený, preemptivní a plánuje thready, nikoliv procesy. Každý thread má přidělené kvantum času, po které může běžet, poté je přeplánován (pokud není před uplynutím kvanta přeplánován kvůli jinému threadu s vyšší prioritou) na jiný thread se stejnou (round-robin) nebo vyšší prioritou. Když takový thread neexistuje, původní thread běží i po uplynutí kvanta. Priority se dělí na user mode (celkem 32, aktuální priorita threadu = základní priorita procesu + relativní priorita) a kernel mode (tzv. IRQL 0-31, user mode thready běží v nejnižším IRQL 0, APC[4] — IRQL 1 a DPC[5] — IRQL 2 implementováno jako softwarové přerušení).

2.2 User mode

2.2.1 Charakteristika

User mode neboli uživatelský režim je přirozeným prostředím pro běžné aplikace, které jsou zde spuštěny ve formě procesu a jeho threadu. Běžný uživatelsky kód má však jistá omezení — nelze například přistupovat do paměti jádra (na 32-bitových Windows bez /3GB přepínače je to paměť vyšší než 8000 0000H), přistupovat k paměťové mapovaným

[4]Asynchronní volání procedur — vzdálené připomíná unixové signály. Rozlišujeme user mode (IRQL 0), kernel mode (IRQL 1) a speciální APC

[5]Odložené volání procedury — použití např. při obsluze přerušení: místo aby ISR prováděla všechen kód, vykoná pouze nejnutnější část a do fronty přidá objekt DPC, kód z DPC bude vykonán jakmile poklesne IRQL na úroveň 2 a na dané DPC přijde řada.

zařízením (většinou jsou mapována do oblasti jádra), využívat IO porty a provádět privilegované instrukce. Nemůže tedy přímo pracovat s hardware. Zvláště nemožnost pracovat s IO porty nesli programátoři zvyklí na DOS nelibě a proto vznikaly různé knihovny, které pomocí driveru[6] umožnily přístup k IO [30]. V systému XP SP2 existuje rozhraní pro debugger `ZwSystemDebugControl`, které výše uvedené operace umožňuje (kromě některých privilegovaných instrukcí), bohužel v dalších verzích (Server 2003 SP1) byla jeho funkčnost z bezpečtnostních důvodů omezena. Standardní a správný způsob je privilegovaný kód umístit do driveru a s ním komunikovat pomocí `DeviceIoControl` nebo `Read/WriteFile`. Open source driver umožňující přístup k hardware lze nalézt na [31].

Dalším nedostatkem uživatelských aplikaci je nízká priorita v rámci IRQL. I když threadu přiřadíme nejvyšší možnou uživatelskou prioritu 31:

```
SetPriorityClass(GetCurrentProcess(), REALTIME_PRIORITY_CLASS);
SetThreadPriority(GetCurrentThread(), THREAD_PRIORITY_TIME_CRITICAL);
```

— nebude ho moci přerušit žádný jiný uživatelský ani systémový thread, stále ale bude mít pouze IRQL 0. Tzn. může být přerušen APC, DPC (viz 2.1.1) i jakýmkoliv hardwarovým přerušením. V žádném případě tedy v user módu nemůžeme splnit požadavek na nepřerušitelný běh úlohy.

Přes výše popsané nevýhody by měl být user mode vždy volbou číslo jedna. V uživatelském režimu má nejlepší časovač rozlišení 1 ms, což je pro mnoho úloh dostačující. Vývoj je velmi snadný — chyba aplikace nezpůsobí pád celého systému, k dispozici je rozsáhlá dokumentace a propracované vývojové prostředky. Lze využívat všech služeb operačního systému.

2.2.2 Časovače

Windows XP nabízí v user módu API pro časovače umožňující volat zadanou rutinu v předem určený čas nebo můžeme časovač emulovat tak, že thread na zadanou dobu uspíme. Použití funkcí nebude popsáno, protože jsou dokumentovány v SDK (viz 2.2.4).

SetTimer Nastavuje časovač, který buď volá určenou rutinu, nebo oknu posílá zprávu WM_TIMER.

[6] Použijeme-li nedokumentovanou funkci NtSetInformationProcess, můžeme povolit přístup na IO porty i bez driveru. Testováno na Windows XP SP2.

SetWaitableTimer Nastavuje časovač vytvořený pomocí `CreateWaitableTimer`. Na takto vytvořený časovač lze nahlížet jako na synchronizační objekt a čekat na něj pomoci `WaitForSingleObject`. Implementováno pomoci APC.

Sleep Uspí thread na zadaný počet ms.

SleepEx Stejné jako `Sleep`, thread je ale během čekání ve stavu „alertable“, tzn. může přijmout user mode[7] APC vyslané např. pomoci `QueueUserAPC`.

Rozlišení výše uvedených funkcí je odvozeno od tiků systémového časovače a je okolo 10 ms (`GetSystemTimeAdjustment`). Tyto funkce nejsou spolehlivé pro čekání v řádech ms. Nejlepší user mode časovače nalezneme v knihovně WINMM.DLL: Multimedia Timers [18]. Využívají totiž nedokumentovanou funkci `NtSetTimerResolution`, která změní granularitu systémového časovače až na 1 ms. Příklad kódu, který nastaví periodický časovač:

```
TIMECAPS tc;
MMRESULT timerID;

// dotaz na rozlišení (v ms)
timeGetDevCaps(&tc, sizeof(TIMECAPS));
// nastavit nejlepší možné rozlišení
timeBeginPeriod(tc.wPeriodMin);
// spustit časovač, bude volat fci TimerCallback, T=1ms
timerID = timeSetEvent(1, wTimerRes, TimerCallback, 0, TIME_PERIODIC);
...
// ukončit časovač
timeKillEvent(timerID);
```

Multimedia Timer není naprosto přesný, jitter může být až v řádech milisekund, podrobnou studii latencí tohoto časovače může čtenář najít v [4] a pokus o jeho vylepšení v [35]. Pro běžné soft-realtime úlohy je však dostačující.

2.2.3 Měření času

K dispozici je následující API:

[7] Kernel mode APC může thread přijmout vždy.

timeGetTime Rozlišení až 1 ms. Multimedia Timer.

GetSystemTime Vrací datum `SYSTEMTIME`-dd/mm/y... včetně údaje o ms. Rozlišení okolo 10 ms.

GetTickCount Rozlišení vrací `GetSystemTimeAdjustment`, zhruba 10 ms.

NtQuerySystemTime Rozlišení zhruba 10ms.

QueryPerformanceCounter/QueryPerformanceFrequency Stovky nanosekund.

Nejpřesnější je duo `QueryPerformanceCounter/QueryPerformanceFrequency`. Nejprve se pomoci `QueryPerformanceFrequency` zeptáme na frekvenci čítače (která se po startu Windows již nemění) a poté pomoci `QueryPerformanceCounter` zjistíme počet tiků. Tyto dvě funkce mohou být implementovány pomoci instrukce RDTSC, na víceprocesorovém počítači pak je nutno měřit tiky jen na jednom procesoru (`SetThreadAffinityMask`) a frekvence čítače je rovná frekvenci CPU. Nebo jsou implementovány pomoci PM Timer a frekvence je pak rovná 3579545Hz. Příklad použití:

```
LARGE_INTEGER time1, time2, tps;
QueryPerformanceFrequency(&tps);
printf ("frequency: %I64Ld ticks/sec\n", tps);
QueryPerformanceCounter(&time1);
...
QueryPerformanceCounter(&time2);
printf ("%.9f seconds %d ticks\n",
        ((float)(time2.QuadPart-time1.QuadPart)/(float)tps.QuadPart),
        time2.QuadPart - time1.QuadPart);
```

2.2.4 Vývojové prostředky

Základním vybavením pro vývoj v user módu je Platform SDK, dostupné z webu Microsoftu zdarma. Obsahuje knihovny *.lib, hlavičkové soubory a především kvalitní dokumentaci. Hlavním kompilátorem pro Windows je cl.exe, lze ho získat zdarma např. s distribuci .NET Framework SDK nebo s Visual Express C++.

2.3 Kernel mode

2.3.1 Charakteristika

Korektní způsob, jak spouštět kód v Ringu 0, je vytvořit tzv. driver neboli ovladač. Nemusí přitom ovládat žádné fyzické zařízení. Driver je načten buď dynamicky, podobně jako DLL v user módu, nebo automaticky při bootu systému. Driver má stejná privilegia jako operační systém. Pokud dojde k výjimce v uživatelské aplikaci, nic se nestane, maximálně je aplikace ukončena. Pokud dojde k neošetřené výjimce v kernel módu, většinou je zavolána funkce `KeBugCheck`, která uloží zprávu o chybě včetně současného stavu systému (do adresáře %WINDOWS%\Minidump), vypíše na obrazovku vzkaz pro uživatele (nechvalně známá modrá obrazovka — BSOD) a ukončí systém.

V kernel módu se používá zcela odlišné API, nelze například použít ani všechny standardní funkce jazyka C. Některé funkce API lze volat jen při IRQL nižším než určitá mez. Adresový prostor je pro všechny drivery stejný (v user módu má každý proces svůj vlastní, izolovaný, tvořený stránkovanou paměti). Při práci s pamětí rozlišujeme PagedPool a NonPagedPool. NonPagedPool je přítomný v paměti vždy, data z něj nikdy nemohou být odložena na disk. Pokud kód běží s prioritou IRQL >= 2, neměl by nikdy přistupovat ke stránkované paměti. Může dojít k výjimce, kterou kvůli příliš vysokému IRQL nebude možno obsloužit. Na to je třeba dbát zvláště při psaní kódu DPC nebo volaného z ISR hardwarových časovačů[8] a používat pouze paměť z NonPagedPool.

2.3.2 Časovače

KeSetTimer Nastaví časovač vytvořený `KeInitializeTimer`. Lze použít jako synchronizační objekt i pro spuštění DPC v daný čas.

KeDelayExecutionThread Ekvivalent user mode funkce `Sleep`. Lze volat pouze pokud je IRQL = 0 (PASSIVE_LEVEL).

KeStallExecutionProcessor Podobně jako `Sleep`, implementováno ale jako aktivní čekání, proto použití doporučené jen pro doby menší než 50 μs.

[8] Aby měl časovač vždy přednost před ostatními přerušeními, běží s nejvyšším možným IRQL

2.3.3 Měření času

KeQuerySystemTime Inkrementuje každých 10 ms. Vrací počet 100 ns intervalu od 1. ledna 1601 UTC (obdoba unixového timestamp — počet sekund od 1. ledna 1970).

KeQueryTickCount Vrací počet tiků systémového časovače od bootu systému. Časový přírůstek na jeden tik lze zjistit pomoci `KeQueryTimeIncrement`.

KeQueryInterruptTime Podle dokumentace `KeQueryTickCount` jde o variantu s lepším časovým rozlišením.

KeQueryPerformanceCounter Tato funkce je volaná `QueryPerformanceCounter`, vrací zároveň frekvenci — rozlišení. Opět nejpřesnější dostupná funkce.

Funkce pro časovače i měření času jsou opět odvozeny od periody tiků systémového časovače. Periodu tiků nelze bohužel pomocí API změnit.

2.3.4 Vývojové prostředky

To, co je pro user mode Platform SDK, je pro kernel mode DDK — Driver Development Kit, v době Windows Vista nahrazován WDK. Ke stažení zdarma [28]. Obsahuje vše potřebné pro vývoj: hlavičkové soubory, knihovny, dokumentaci, kompilátory a další. Bohužel, mnoho funkcí kernelu je nedokumentovaných.

Nepostradatelným nástrojem je v úvodu kapitoly zmíněný Windbg [26]. Lze s ním ladit lokální i vzdálený kernel (např. OS spuštěný ve VmWare) a analyzovat minidumpy. Je vhodný také pro ladění běžných aplikací.

Monografie zabývající se vývojem kernel mode driverů je např. [3].

Kapitola 3

Rozšíření Windows XP

Pojmem rozšíření (Extensions) rozumíme programové prostředky, které nějakým způsobem rozšiřují jádro Windows a propůjčují mu nové vlastnosti.

Na úvod je nutné říci, že dostupná rozšíření pro Windows XP jsou velmi drahá a proto pro běžné aplikace takřka nedostupná.

V této kapitole bude popsán hardware, pomoci něhož lze klíčovou část rozšíření — časovač — implementovat. Časovače jsou zde považovány za programovatelné zdroje přerušení. Nejprve bude popsána implementace mechanismu přerušení, poté bude v sekci ACPI poukázáno na problémy znemožňující nepřerušitelný běh úlohy a poté zmíněny hardwarové časovače a hardwarové prostředky pro měření času.

Text je doplněn ukázkami kódu pro Windbg [26]. Přehled použitých příkazů nalezne čtenář v příloze B.

3.1 Zdroje latencí

Rozlišení (resolution, granularity) časovače závisí na hardware, jímž je časovač realizován. Skutečná přesnost (accuracy) závisí mimo jiné na způsobu vyřízení požadavku přerušení od časovače k procesoru. Ke zpoždění může dojít, pokud je právě obsluhováno přerušení s vyšší prioritou, přerušení jsou zakázána (maskována) nebo je procesor v SM módu (způsobeno řízením spotřeby — ACPI) a použitím ISR, která ukládá kontext (standardní způsob s `KINTERRUPT`). Dalšími ovlivňujícími faktory jsou např. koherence cache, probíhající DMA přenos, atp.

3.2 Přerušení

Architektura vyřizování požadavku přerušení je na počítačích PC poznamenána historickým vývojem. Vedle sebe tu v rámci kompatibility existují dva řadiče přerušní. Obecně lze říci, že od zařízení, které požaduje přerušení, do procesoru, který požadavek přerušení obslouží, vede několik cest. Který řadič je rychlejší, může záležet na konkrétním hardware.

3.2.1 APIC a PIC

Procesory rodiny P6 a vyšší obsahují APIC [11], který se skládá ze dvou části: Local APIC (integrován v CPU) a IO APIC (na základní desce: samostatně [10], integrován v SouthBridge [9, 9.5] nebo jinde, jeden a více). Local APIC má dvě základní funkce. První úloha je zpracování vnitřních (LINTI0-1, Timer, Performance Mononitor, Thermal Sensor) a vnějších, po sběrnici doručených, (IO APIC, MSI, ...) požadavků přerušení. Druhá úloha je přijímání/vysílání IPI (ve víceprocesorových konfiguracích).

Z důvodů zpětně kompatibility[1] (PC-AT) se na základní desce často nachází také PIC (master a slave) a APIC může být vyřazen z provozu, nebo mohou fungovat oba současné (standard ACPI zakazuje). LAPIC má vstupní piny LINTI0 a LINTI1. Výstup PIC je připojen na LINTI0. Pokud není APIC aktivní, systém je nakonfigurován v módu kompatibility: LINTI0 je nakonfigurován jako ExtINT. LINTI1 má stejnou funkci jako při zapnutém APIC — tedy příjem NMI.

Požadavků na přerušení zpracovávaných LAPICem (Delivery Mode) je několik typů. Typ požadavku můžeme nastavit v IO APIC (externí zdroj), LAPIC díky mechanismu generování meziprocesorových přerušení dokáže vyslat libovolný typ požadavku přerušení také sám sobě[2], typ můžeme nastavit i u vnitřních zdrojů.

Fixed: Standardní typ přerušení.

> Zdrojem jsou zařízení připojená na IO APIC, která přerušení využívají k signalizaci nějaké události (např. stisk klávesy, dokončení DMA přenosu). Je volána obslužná rutina v IDT určená vektorem požadavku. Vektor je číslo od 10H do 0FEH. Konec přerušení je signalizován zapsáním nuly do registru EOI v LAPIC.

Lowest Priority: Meziprocesorové přerušení.

[1] Pěkný přehled vývoje může čtenář nalézt v [16] a [17].

[2] Tímto způsobem implementuji Windows softwarová přerušení pro APC a DPC ve funkci `hal!HalRequestSoftwareInterrupt`.

Přerušení stejné jako Fixed, s tím rozdílem, že je doručeno procesoru který právě pracuje s nejnižší prioritou. Implementačně specifické.

SMI: System Management Interrupt.

Zdrojem je vodič chipsetu napojený na pin procesoru SMI#, viz např. [13, 4.2 Alphabetical Signals Reference]. Toto přerušení nelze maskovat instrukcí CLI. Jedná se o speciální typ přerušení, volána rutina na adrese SMM_BASE+8000H. Viz 3.3.1.

NMI: Non Maskable Interrupt.

Zdrojem[3] bývá vodič chipsetu napojený na pin LINTI1, příčinou pak například chyba hardware (chyba parity DRAM — pokud modul paměti takovou kontrolu umožňuje, apod.). Vždy je volána rutina na vektoru 2. NMI nelze maskovat pomocí instrukce CLI (CLI nemaskuje přerušení s vektorem menším než 32, tedy rezervované vektory[11, 5.8.1 Masking Maskable Hardware Interrupts]).

INIT: Inicializace.

Speciální typ přerušení, provede inicializaci procesoru. Pro tento signál má procesor i pin INIT#.

ExtINT: Přerušení v režimu kompatibility.

Zdrojem tohoto přerušení je PIC, procesor na něj odpoví speciálním cyklem INTA, který je chipsetem (NB ho přeloží na PCI Interrupt Acknowledge Cycle a pošle po PCI do SB) přeložen na signál do PIC, oznamující obdržení požadavku přerušení. PIC (integrovaný v SB) si tento signál přeloží na dva pulzy INTA# a pošle je do jádra kompatibilního s 8259. Poté je procesoru doručen vektor přerušení a spuštěna příslušná obslužná rutina v IDT. Konec přerušení je signalizován příkazem EOI do PIC. Pokud je PIC v AEOI módu, pak není třeba konec přerušení signalizovat, protože je ukončeno automaticky po obdržení druhého pulzu INTA#. Celý proces je popsán v datasheetu SB [9, 5.8.1 Interrupt Handling].

Z výše uvedeného vyplývá, že nejvhodnějším typem přerušení je NMI, které nemůže být zakázano. Pokud používáme samostatně starý řadič PIC, pak NMI nelze použít, všechna přerušení jsou typu Fixed. V případě konfigurace s IO APIC může NMI generovat libovolné zařízení nebo PIC připojený na LAPIC.

[3] Lze zakázat (maskovat) zapsáním bitů 7 na port 70H: `ob 70 80`.

Windows XP SP2 v režimu ACPI (standardní konfigurace, viz [21]) využívají APIC, zatímco všechna přerušení z PIC jsou maskovaná. PIC tedy můžeme bezpečně použít pro získání NMI (po konfiguraci LAPICu).

Tabulka 3.1: PIC

```
lkd> !pic
------- IRQ Number ------- 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
Physically in service:  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
Physically masked:      Y  Y  Y  Y  Y  Y  Y  Y  Y  Y  Y  Y  Y  Y  Y  Y
Physically requested:   Y  .  .  .  .  Y  .  .  .  .  Y  Y  .  .  .  .
```

Tabulka 3.2: V tabulce jsou patrné vektory i typy přerušní, jak byly jednotlivým zdrojům přiřazeny od Windows. Např. INTI02 — PIT — není použit.

```
Lokální zdroje přerušení
lkd> !apic
Apic @ fffe0000  ID:0 (40010)  LogDesc:01000000  DestFmt:ffffffff  TPR 41
TimeCnt: ffffffffclk  SpurVec:1f  FaultVec:e3  error:2
Ipi Cmd: 00040041  Vec:41  FixedDel    Dest=Self      edg  high
Timer..: 000300fd  Vec:FD  FixedDel    Dest=Self      edg  high       masked
Linti0.: 0001001f  Vec:1F  FixedDel    Dest=Self      edg  high       masked
Linti1.: 000004ff  Vec:FF  NMI         Dest=Self      edg  high
TMR: 73, 83, B1

Externí zdroje přerušení
lkd> !ioapic
IoApic @ FEC00000  ID:2 (11)  Arb:0
Inti00.: 000100ff  Vec:FF  FixedDel  PhysDest:00      edg  high       masked
Inti01.: 00000993  Vec:93  LowestDl  Lg:01000000      edg  high
Inti02.: 000100ff  Vec:FF  FixedDel  PhysDest:00      edg  high       masked
Inti03.: 000100ff  Vec:FF  FixedDel  PhysDest:00      edg  high       masked
Inti04.: 00000992  Vec:92  LowestDl  Lg:01000000      edg  high
Inti05.: 000100ff  Vec:FF  FixedDel  PhysDest:00      edg  high       masked
Inti06.: 000100ff  Vec:FF  FixedDel  PhysDest:00      edg  high       masked
Inti07.: 000100ff  Vec:FF  FixedDel  PhysDest:00      edg  high       masked
Inti08.: 000008d1  Vec:D1  FixedDel  Lg:01000000      edg  high
Inti09.: 0000a9b1  Vec:B1  LowestDl  Lg:01000000      lvl  low
Inti0A.: 000100ff  Vec:FF  FixedDel  PhysDest:00      edg  high       masked
Inti0B.: 000100ff  Vec:FF  FixedDel  PhysDest:00      edg  high       masked
Inti0C.: 000009a3  Vec:A3  LowestDl  Lg:01000000      edg  high
Inti0D.: 000100ff  Vec:FF  FixedDel  PhysDest:00      edg  high       masked
Inti0E.: 00000962  Vec:62  LowestDl  Lg:01000000      edg  high
Inti0F.: 00000982  Vec:82  LowestDl  Lg:01000000      edg  high
Inti10.: 0000a973  Vec:73  LowestDl  Lg:01000000      lvl  low
Inti11.: 000100ff  Vec:FF  FixedDel  PhysDest:00      edg  high       masked
Inti12.: 0000a983  Vec:83  LowestDl  Lg:01000000      lvl  low
Inti13.: 0000a9b4  Vec:B4  LowestDl  Lg:01000000      lvl  low
Inti14.: 0000a963  Vec:63  LowestDl  Lg:01000000      lvl  low
Inti15.: 0000a9a4  Vec:A4  LowestDl  Lg:01000000      lvl  low
Inti16.: 000100ff  Vec:FF  FixedDel  PhysDest:00      edg  high       masked
Inti17.: 0000a994  Vec:94  LowestDl  Lg:01000000      lvl  low
```

3.2.2 IRQ a INTI

Zařízení (např. PIT) je fyzicky připojeno na vstupní pin IO APIC a PIC. Číslo vstupního pinu u PIC se uvádí jako IRQ0:15 a je určeno [6], u IO APIC se uvádí jako INTI00:23 (počet vstupů IO APIC závisí na konkrétní implementaci). Přiřazení IRQ/INTI zařízením se mírně liší — na INTI00 je připojen signál INTR (požadavek přerušení) vedoucí z master PIC, zatímco na IRQ0 je připojen PIT. V IO APIC je PIT připojen na INTI02, na IRQ2 je připojen INTR ze slave PIC. Výstup IO APIC je přes APIC sběrnici (P4 a novější přes systémovou sběrnici) připojen k LAPIC, odtud jsou požadavky vedeny přímo do jádra procesoru. Výstup PIC je připojen kromě INTI00 také na LAPIC, konkrétně LINTI0.

Způsob zapojení IO APIC pro konkrétní implementaci lze zjistit z ACPI tabulky MADT [5, s. 111], kde zapojení lišící se od standardu je uvedeno jako Interrupt Source Override, kde Source označuje číslo IRQ a Global Systam Interrupt označuje číslo INTI. Z následujícího příkladu vyplývá, že IRQ0 (PIT) je mapováno na INTI2. Vidíme také, že NMI vede do LINTI1. Výstup PIC tedy může vést kromě INTNI0 také do LINTI0.

Tabulka 3.3: ACPI MADT

```
lkd> !mapic
MAPIC - HEADER - ffffffff8b0e0c0
  Signature:               APIC
...
MAPIC - BODY - ffffffff8b0e0e4
  Local APIC Address:      0xfee00000
  Flags:                   0x000001
    PC-AT dual 8259 compatible setup
  Processor Local Apic
    ACPI Processor ID:     0x01
    APIC ID:               0x00
    Flags:                 0x00000001
      Processor is Enabled
  IO Apic
    IO APIC ID:            0x02
    IO APIC ADDRESS:       0xfec00000
    System Vector Base:    0x00000000
  Interrupt Source Override
    Bus:                   0x00
    Source:                0x00
    Global Interrupt:      0x00000002 ...
  Non Maskable Interrupt Source - local to processor
    Flags:                 0x0005
    Processor:             0x01
    LINTIN:                0x01 ...
```

3.2.3 Vyřizování požadavků přerušení

- ▷ PIT →▸ PIC →▸ LAPIC (LINTI0)
- ▷ PIT →▸ PIC →▸ IO APIC (INTI00) →▸ LAPIC
- ▷ PIT →▸ IO APIC (INTI2) →▸ LAPIC

Poté co koncové zařízení (např. PIT) vyšle žádost o přerušení, dorazí tento signál do IO APIC a zároveň/nebo do PIC. Pro naše účely je zajímavé sledovat vyřizovaní přerušení z hlediska priority.

Tabulka 3.4: Mapování zdrojů přerušení na vstupy PIC a IO APIC

	IRQ	INTI
	PIC master:	IO APIC:
0	PIT	PCI master INTR[b]
1	Keyboard	
2	PIC slave INTR	PIT[b]
3	COM 2	
4	COM 1	
5	LPT 2[a]	
6	Floppy	
7	LPT 1	
	PIC slave:	
8	RTC	
9	SCI[b]/-	
10	SCI[b]/COM 4[a]	
11	SCI[b]/COM 3[a]	
12	Mouse PS/2	
13	Coprocessor	
14	Primary IDE	
15	Secondary IDE	
16:19		PCI
20:21		SCI[b]/Motherboard
22		SCI[b]/General purpose
23		SCI[b]

[b]Jedna z možnosti, viz [9], konkrétní implementaci popisuje ACPI, např. pro PCI určeno ve struktuře FADT [5].

[a]Specifikace [6] definuje jako nepovinné.

IO APIC: Zde je provedeno přesměrování — IO APIC obsahuje tabulku (Redirection Table) dvaceti čtyř 64-bitových záznamů — pro každý INTI jeden. Každý záznam obsahuje tyto informace: typ signálu (hrana nebo úroveň + aktivní v low/high), vektor (32-255) a priorita/typ přerušení, cílový procesor, způsob výběru procesoru (statický/dynamický). IO APIC provádí výběr přerušení kruhově, tzn. ne podle priority - toto řízení je ponecháno na LAPIC, do kterého putuje požadavek přerušení buď po APIC sběrnici (přímo) nebo po systémové sběrnicí (přes North Bridge).

LAPIC: Standardní přerušení, označovaná jako Fixed, nesou informaci o vektoru přerušní. Třída priority = vektor/16, rozdělí vektory na třídy 0-15, vyšší číslo třídy značí vyšší prioritu. Třídy 0 a 1 jsou rezervovaně. Nejnižší 4 bity značí prioritu ve třídě. Opět vyšší číslo znamená vyšší prioritu. Jednoduše: čím vyšší číslo vektoru, tím vyšší priorita. Přerušení lze softwarově maskovat („priority treshold“) pomoci TaskPriorityRegister TPR (FEE0 0080h) a současnou prioritu zpracovávaného přerušení lze číst z ProcessorPriorityRegister (FEE0 00A0h).[4] Přerušení typu NMI/SMI/INIT/ExtInt jsou vyřizována přednostně (TPR je ignorován), EOI se nesmí signalizovat. SMI má přednost před NMI. U všech typu přerušení je informace o vektoru, u SMI však nemá smysl a u NMI je vektor vždy 2.

PIC: Interpretuje priority takto (od nejdůležitější k nejméně důležité): IRQ0, 1, 8-15, 3-7. Pořadí je způsobeno tím, že na IRQ2 je připojen slave PIC, proto jeho IRQ8-15 má větší prioritu než IRQ3-7 na master PIC. Požadavek na přerušení putuje z PIC do IOAPIC (a zároveň po APIC/systémové sběrnici do procesoru (LAPIC) na vstup LINT0).

Z výše uvedeného vyplývá: IO APIC se prioritou nezabývá, PIC má priority pevně dané. Pro LAPIC znamená vyšší vektor požadavku vyšší prioritu, NMI má však přednost.

[4] Pokud je obsluhováno přerušení, tzn. je nastaven příslušný bit ve 256-bitovém reistru ISR (In Service Register) a mezitím přijde přerušení s nižší prioritou, je uloženo do 256-bitového IRR (Interrupt Request Register). Ve frontě tak může čekat 512 přerušení, od každého vektoru dvě. U procesoru P6 a Pentium mohou být ve frontě pouze přerušení různé třídy priority [11, 8.8.4 Interrupt Acceptance for Fixed Interrupts]. Naopak, pokud přijde přerušení s vyšší prioritou, přeruší vykonávaní obslužné rutiny přerušení s nižší prioritou.

3.3 ACPI

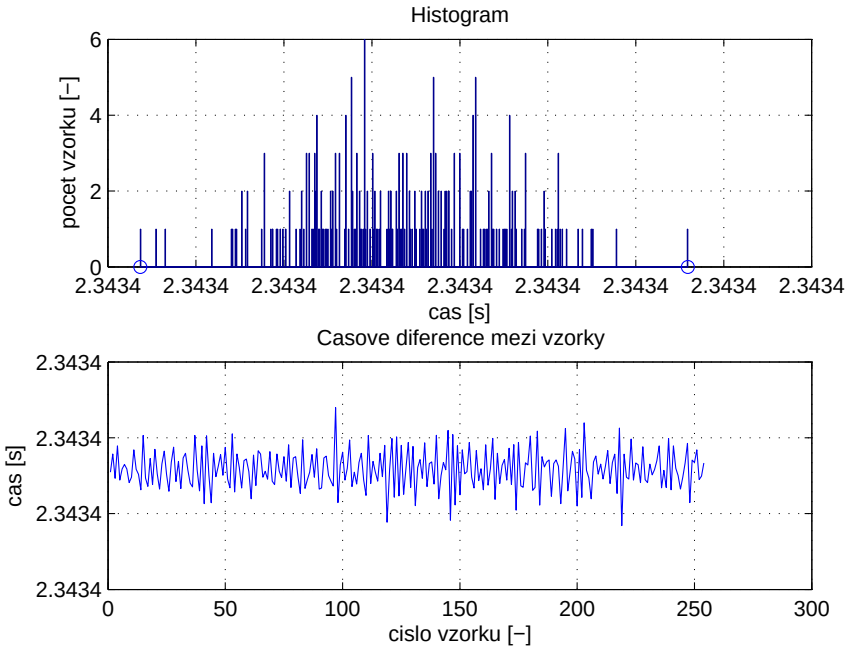
Advanced Configuration and Power Interface je standard vyvinutý mimo jinými Microsoftem a Intelem. Poskytuje informace o konfiguraci hardware a umožňuje jednotný přístup k vlastnostem zařízení, jejichž rozhraní není definováno standardem (např. časování IDE řadiče). Díky ACPI máme k dispozici nový časovač — PM Timer.

Při snaze o implementaci přesných časovačů však může ACPI představovat překážku. Problémem je řízení spotřeby a tím změna frekvencí nebo „processor throttling“ (vkládání idle cyklu). Procesor může vykazovat různé chování v různých stavech definovaných ACPI.

3.3.1 SMM

Na druhou stranu, ACPI řeší jeden ze zásadních problémů. V předchozím standardu (APM) byly všechny události řízení spotřeby spjaté se System Management Modem. Jedná se o speciální režim procesoru, kdy pracuje v 16-bitovém flat real módu[5]. Do SMM může procesor přejít pouze vyvoláním SMI (System Management Interruptu) a vrátit se může jen instrukcí RSM. V SMM běží kód dodávaný výrobcem BIOSu, všechny žádosti o přerušení jsou ignorovány, dokonce i NMI. ACPI definuje nové přerušení typu Fixed, SCI, o obsluhu se tak stará operační systém (konkrétně ve Windows driver acpi.sys) v běžném módu procesoru, viz obrázek 3.1. Operační systémy, které ACPI nepodporují (např. DOS) nelze tedy pro real-time úlohy použít. Zpoždění způsobené SMI může být v řádech ms.

[5]Někdy nazýváno unreal mode.



Obrázek 3.1: SCI vyvolané přetečením PM Timeru jak bylo zachyceno ve Windows XP. V případě OS bez podpory ACPI je místo SCI vyvoláno SMI a je spouštěn kód BIOSu v SMM.

Bohužel ne všechny zdroje přerušení SMI jsou přemapována na SCI. SMM lze mimo jiné použít pro virtualizaci chybějících zařízení. Příkladem je USB klávesnice. Operační systém komunikuje s běžnou kompatibilní klávesnicí přes IO porty, USB klávesnici tedy nevidí. Jakmile se OS pokusí číst z IO portu pro klávesnici, služba chipsetu zvaná IO Port Trap [9, 7.1.47 IOTRn — I/O Trap Register (0–3)] vyvolá přerušení. Kód v SMM potom změní data IO instrukce tak, jako by byla přítomna běžná klávesnice, zatímco skutečná komunikace s klávesnicí probíhá po USB. Dalším příkladem může být boot disk připojený přes USB. Přehled zdrojů způsobujících SMI lze najít v datasheetu pro SB, např. [9, 5.13.4 SMI#/SCI Generation].

Na většině chipsetu lze SMI zakázat, naneštěstí ne vždy je dostupný datasheet.

3.4 Hardwarové časovače

Počítač standardu IA-PC-AT nabízí několik hardwarových časovačů, které po uplynutí zadané doby vyvolají přerušení (interrupt). Podle specifikace [6] musí být vždy dostupné

tyto časovače: RTC a PIT. Novější procesory navíc nabízí Local APIC timer a lze využít i čítače procesoru, které byly navrženy pro profilování výkonu a generují přerušení při přetečení (PerfMon). Současná PC (2007) mají zabudován HPET, časovač který má odstranit nedostatky všech jeho předchůdců [19].

Při výběru časovače můžeme mít různá kritéria. Např. pokud chceme aperiodický časovač, požadujeme rychlé rozhraní pro jeho programování. To nabízí LAPIC Timer, PerfMon nebo HPET, které mají paměťově mapované registry. RTC a PIT se programují přes IO porty, což je pomalejší.

Dalším kritériem je rozlišení časovače. Zde opět dobře vychází LAPIC Timer, který má frekvenci odvozenou od sběrnice FSB. Ještě lepší rozlišení pak nabízí funkce procesoru pro profilování výkonu, odvozené od frekvence CPU.

Důležitá je také stabilita časovače, tzn. frekvence inkrementace je stále stejná. Tento požadavek splňuje HPET, PIT a RTC. I LAPIC Timer lze považovat za stabilní. Zcela nepolehlivý je pak PerfMon.

Možnost generování NMI. Tuto vlastnost mají všechny zmíněné časovače, kromě LAPIC Timer.

Podstatný požadavek je, aby časovač nebyl využíván operačním systémem Windows XP. To platí pro LAPIC Timer, HPET a funkce procesoru pro profilování výkonu. RTC a PIT mohou být využívány v závislosti na konfiguraci Windows XP.

Z výše uvedených časovačů je výrobci komerčních real time rozšíření používán LAPIC Timer. Přestože nedokáže generovat NMI, tzn. nedokáže vyvolat přerušení, když jsou maskována instrukcí CLI. Výhody převažují: rozlišení, stabilita, rychlé programovaní a navíc je dostupný na většině počítačů (na rozdíl od HPET). Místo NMI se používá přerušení typu Fixed s nejvyšší prioritou.

Nyní bude LAPIC Timer popsán podrobněji, protože je z dostupných časovačů nejvýhodnější. PerfMon bude zmíněn, protože jeho použití jako časovače není obvyklé a může být v určitých případech přínosné.

3.4.1 LAPIC Timer

rozlišení	desítky až jednotky ns, frekvence FSB
vstup	hodnota čítače, násobič čítače (dělič frekvence)
mód	periodický, one-shot
rozhraní	registry procesoru
NMI	ne
stabilní	ne
konflikt s XP	možný (Kernrate)

LAPIC Timer je přímo součástí procesoru, jeho programování je tedy velmi rychlé, provádí se zapisováním a čtením paměťově mapovaných registrů. Způsob programování je podrobně popsán v manuálech firmy Intel a AMD, např. v [11, 8.5.1 Local Vector Table], [11, 8.5.4 APIC Timer]. Spočívá v několika krocích:

1. Detekce, zda je LAPIC zapnut/podporován, případně aktivace LAPIC.
2. Instalace ISR do IDT.
3. Nastavení módu, vektoru přerušení a maskovaní přerušení pomoci LVT Timer Register (FEE0 0320H).
4. Nastavení dělice frekvence pomoci Divide Configuration Register (FEE0 03E0H).
5. Nastavení počáteční hodnoty čítače do Initial Count Register (FEE0 0380H) — tím je časovač spuštěn.

Z výše uvedeného postupu je problematická pouze případná aktivace LAPIC. Detekci lze provést dvěma způsoby: provedeme instrukci CPUID s EAX=1, výsledek vrácen v registru EDX, bit 9 (pokud má hodnotu 1, je LAPIC aktivován). Druhý způsob je kontrola bitu 11 v MSR IA32_APIC_BASE (1BH).

```
lkd> rdmsr 1b
msr[1b] = 00000000`fee00900
```

Pokud je tento bit 0, je LAPIC vypnut. Na novějších procesorech s doručováním přerušení po FSB lze nastavením bitů 11 opět LAPIC povolit. Pokud jsou však přerušení doručovaná po 3-drátové APIC sběrnicí, nelze LAPIC povolit, dokud není počítač restartován.

MSR IA32_APIC_BASE definuje bit 9 (BSP — zda je procesor bootstrap — spouští ostatní procesory ve víceprocesorové konfiguraci) a bázi registru (standardně FEE0 0000H), může být změněna, Windows ji však nemění.

Pokud se pokusíme LAPIC povolit, je třeba nakonfigurovat přerušení tak, aby HAL mohl dál bezchybně pracovat. Nebo změnit HAL na takový, který APIC využívá. To lze provést jako aktualizaci ovladače: Win+R, devmgmt.msc, Počítač/%název_počítače% /Vlastnosti/Aktualizovat ovladač... a v následujících krocích zvolíme Jedno/Víceprocesorový počítač s rozhraním ACPI. %název_počítače% odpovídá současnému HALu. Původní název souboru (např. halaacpi.dll) zjistitelný z hal.dll (na tento název je při instalaci přejmenován) popisuje typ HALu [22]. Další způsob je změnit typ HALu v souboru boot.ini přepínačem /HAL=název_souboru [23].

Windows XP SP2 standardně LAPIC Timer nevyužívají, pouze pro profilování kernelu programem Kernrate [27]. Profilování funguje takto: Kernrate nastaví časovač do periodického módu, rutina obsluhy přerušení (hal!HalpProfileInterrupt) má vektor 0FDH (tedy nejvyšší prioritu). V okamžiku přerušení uloží obslužná rutina hodnotu EIP (instruction pointer) a Kernrate při skončení profilování podle hodnot EIP určí který modul jádra zrovna běžel. Tím odhadne spotřebu času procesoru jednotlivými moduly.

LAPIC Timer je využíván mnoha programy třetích stran, zejména diagnostické nástroje (např. [29]) a real-time rozšíření.

Frekvence FSB (zdroj pro LAPIC) může být změněna např. pro účely řízení spotřeby či přetaktování. Hodiny pro FSB bývají přístupné prostřednictvím SMBus. Standardně by však frekvence neměla být měněna.

Použitím LAPIC Timeru jako časovače pro sběr dat se zabývá [36]. Práce uvádí, že Windows na jednoprocesorových systémech APIC vypínají a poté již nemůže být znovu aktivován, až do restartu. První tvrzení je nepravdivé, APIC je zapnut pokud OS usoudí že je počítač kompatibilní s ACPI, záleží tedy například na verzi BIOSu a ne na počtu procesorů. Druhé tvrzení je pravdivé jen pro APIC s doručováním požadavků přerušení po APIC sběrnicí. Další sporné tvrzení v uvedené práci je doporučení volit vektor přerušení metodou pokus-omyl.

3.4.2 Performance Monitoring MSR

rozlišení	až stovky ps, frekvence CPU
vstup	hodnota čítače
mód	one-shot
rozhraní	registry procesoru
NMI	ano
stabilní	ne
konflikt s XP	ne

Současné procesory umožňují sledování různých událostí z důvodů analýzy výkonu systému. Událost je např. vykonání instrukce (retired instruction), vyprázdnění instrukční pipeline, počet cyklu s maskovanými přerušeními, počet naplnění cache, atd. Které události lze monitorovat, záleží na výrobci i na modelu procesoru.

Programátorský model sledování události je tvořen dvěma MSR. První registr (PerfCtr) je čítač události, např. u AMD má šířku 48bit. Druhým registrem (PerfEvtSel) se volí typ události a další vlastnosti. Lze například nastavit aby bylo generováno přerušení při přetečení čítače. Aktuální stav čítače lze kromě PerfCtr číst také pomoci instrukce RDPMC (Read Performance-Monitoring Counters)[6] Počet parů PerfCtr a PerfEvtSel je dán implementací, např. u AMD jsou čtyři, u Intelu dva. Procesory Intel — např. Xeon, P4 — nabízí také nový programátorský model (registry ESCR Event selection control, CCCR counter configuration control a čítače) s možností sledovat 18 událostí.

K realizaci časovače je u procesorů AMD vhodné použít typ události 76H — CPU Clocks not Halted [15, 10.2.1 Performance Monitor Events]. Procesory Intel tuto událost znají jako 03CH — UnHalted Core Cycles (jiné postupy viz [12, 18.13.9.2 Non-Sleep Clockticks]). Procesor je ve stavu „not-halted“ pokud není zastaven instrukcí HLT[7]. Instrukce HLT je používaná k úspoře energie v době když procesor není využíván. Jednoduchou prevencí je spustit thread s nízkou prioritou, jak je dokonce doporučeno v [15, 10.2 Performance Event-Select Registers]. Naneštěstí, procesor neikrementuje čítač také když čeká na dokončení IO operace. Z tohoto důvodů je použití Performance Monitoring MSR k implementaci časovače značně nespolehlivé.

1. Detekce typu procesoru.

[6] Pro Ring1-2 nutno povolit nastavením bitů 8 (PCE) v CR4.

[7] Další příčinou přepnutí do stavu „halted“ je řízení spotřeby (Power Management): STPCLK, jiný ACPI stav než C0, atd. [12, 18.11.2 Pre-define Architectural Performance Events], [5].

2. Instalace ISR do tabulky přerušení.
3. Nastavení módu, vektoru přerušení a maskovaní přerušení pomoci LVT Performance Counter Register v LAPIC (FEE0 0340H[8]).
4. Nastvení PerfCtr0 (C001 0004H — AMD, 0C1H — Intel) na hodnotu $-x$, kde x je požadovaný počet tiku než dojde k přetečení a tím k vyvolání přerušení.
5. Nastavení PerfEvtSel0 (C001 0000H — AMD, 186H — Intel): událost CPU Clocks not Halted, Operating-Systém Mode, Enable APIC Interrupt, Enable Counter.
6. Poté, co je vyvoláno přerušení, je nutné znovu nastavit hodnotu PerfCtr0.

   Tento časovač využijeme zejména na jednoprocesorovém počítači, když požadujeme NMI a nejvyšší možnou frekvenci přerušení (perioda v řádech mikrosekund).

3.5 Hardwarové prostředky pro měření času

K měření času lze použít TSC, PM Timer, HPET nebo čítač LAPIC Timeru. K praktickému použití je vhodný buď PM Timer nebo TSC (HPET je jen v novějších počítačích a LAPIC může být použit jako zdroj přerušení). PM Timer i TSC jsou běžně dostupné. PM Timer tiká se stálou frekvencí, ale přistup k němu je pomalý a přeteče po několika sekundách. Naproti tomu čtení TSC je rychlé, ale může dojít ke změně frekvence inkrementace.

3.5.1 TSC

rozlišení	až stovky ps, frekvence CPU
šířka	64 bit
rozhraní	registry procesoru
stabilní	ne

Time Stamp Counter je při zapnutí procesoru nastaven na nulu a poté při každém tiku hodin procesoru svoji hodnotu inkrementuje. Hodnotu TSC čteme pomocí instrukce RDTSC (vrací hodnotu v EDX:EAX) nebo z MSR TSC[9] (10H). Instrukce RDTSC je standardně dostupná i pro Ring 3 (user Mode), lze ji však pro Ring 3 zakázat registrem CR4, konkrétně nastavením bitu 2 (TSD — Time Stamp Disable)

[8] U procesoru Intel není dán tento registr architekturou, v některé implementaci se může lišit.
[9] Nazýván také IA32_TIME_STAMP_COUNTER

TSC je výhodný pro měření délky trvání instrukcí, tzv. CPI (Cycles Per Instruction). Například vytvoříme smyčku a změříme počet strojových cyklů na jeden průchod smyčky (lze vytvořit smyčku s dvěma cykly na jeden průchod). Tak můžeme vytvořit Busy Waiting Timer[10] — aktivní čekání, které je vhodné pro časy kratší než 1μs. Pro časy delší využijeme časovače generující přerušení.

Před měření délky trvání instrukcí je vhodné provést serializaci instrukcí — vyprázdnit instrukční pipeline [8]. To lze provést například zavoláním CPUID [11, 7.4 SERIALIZING INSTRUCTIONS]. Jinak by se mohlo stát, že instrukce která je v kódu až za RDTSC, bude díky paralelnímu zpracování vykonána dříve.

Při použití TSC nezáleží na stavu procesoru — narozdíl od Performance Monitoring MSR procesor inkrementuje čítač i pokud byl zastaven instrukcí HLT nebo signálem #STPCLK [12, 18.9 TIME-STAMP COUNTER]. Frekvence inkrementace TSC je stejná jako frekvence CPU, proto je důležité dbát, aby tato byla konstantní a nebyla měněna, např. z důvodu řízení spotřeby[11] (Intel Speed Step, atp.).

Na víceprocesorových systémech je nutné zajistit běh kódu pouze na jednom procesoru — například nastavením afinity threadu (`KeSetSystemAffinityThread`, `KeSetAffinityThread`, `SetThreadAffinityMask`).

3.5.2 PM timer

rozlišení	stovky ns, 3579545Hz
šířka	24 nebo 32 bit
rozhraní	IO
stabilní	ano

Power Management Timer je definován v [5]. Je to čítač s frekvencí 3579545Hz. Port, na kterém můžeme číst aktuální hodnotu čítače, není pevně dán. Jedná se o ACPI zařízení, adresu portu přečteme z FADT (konkrétně offset 76 — PM_TMR_BLK), stejně jako šířku čítače (většinou bývá 24 bitů) - je určena v Flags (offset 112) bitem TMR_VAL_EXT (bit 8, 0 — 24 bitů, 1 — 32 bitů). Následující příklad ukazuje adresu portu 0808H, 24bit.

[10] Podobně je implementována funkce `nt!KeStallExecutionProcessor`. Ta volá `hal!HalpPmTimerStallExecProc` která se ve smyčce pomoci `hal!QueryTimer` dotazuje PM Timeru jestli už čas čekání vypršel.

[11] Znemožnění přechodu mezi P-stavy [5, 8.1 Processor Power States] ve Windows: Ovládací panely / Možnosti napájení / zvolit Vždy zapnuto.

Tabulka 3.6: ACPI FADT

```
lkd> !fadt
FADT --- 806fe4c0HEADER - ffffffff806fe4c0
  Signatuře:               FACP
  ...
  PM Timer Block:          0x00000808
  PM Timer Length:         0x004
  ...
```

3.6 Komerční rozšíření

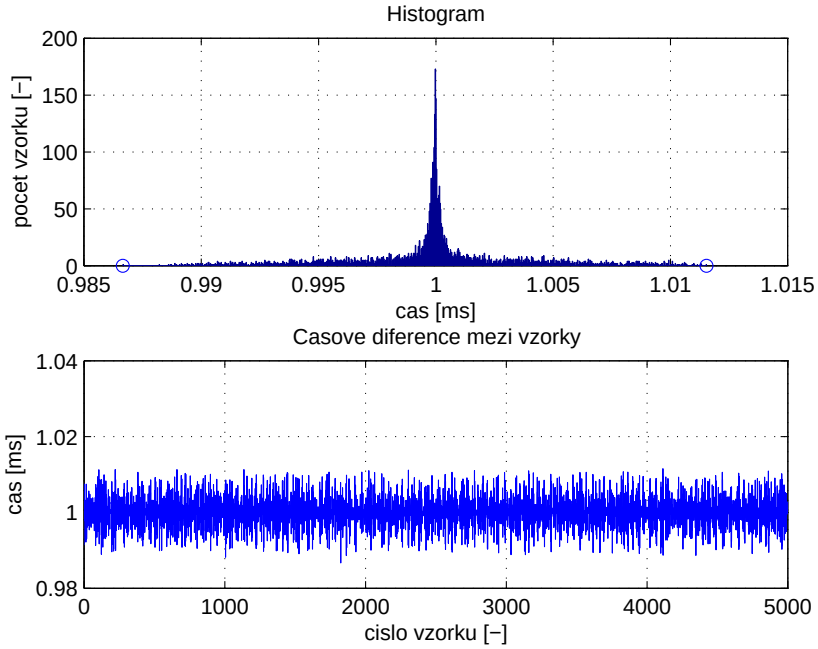
Na trhu lze najít rozšíření, která se snaží propůjčit Windows real-timové vlastnosti. Často však za velmi vysokou cenu. Některá real-timová rozšíření jsou součástí Matlabu, například Real-Time Windows Target (založený na LAPIC Timeru), demonstrační příklady se spouští příkazem `rtwtdemo`. Naproti tomu Real Time Toolbox [33] je nutné dokoupit za cenu v řádech desítek tisíc korun. Pravděpodobně nejpoužívanější real-time rozšíření pro Windows je RTX od firmy Ardence [32], v ceně několika tisíc dolarů.

3.6.1 Ardence RTX

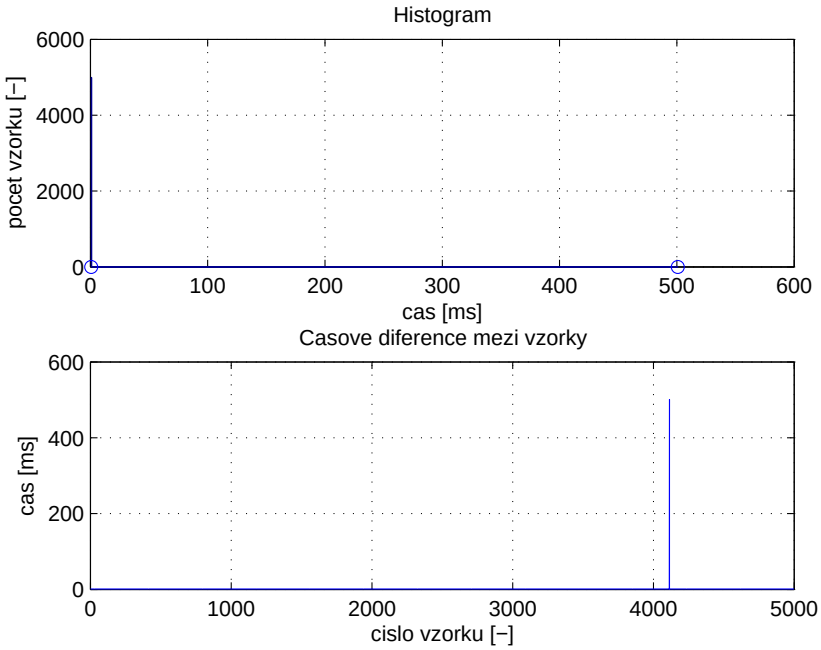
Toto rozšíření vytváří z Windows plnohodnotný RTOS s Win32 API. Hlavní zdroj přerušení tiká s nejmenší periodou 100 μs (standardně 500 μs) a určuje tak rozlišení API časovačů. Firma Ardence tvrdí, že RTX je hard real-time. Tedy takový systém, kde ke zpoždění (nesplnění časových požadavků) nesmí nikdy dojít, protože by to mělo katastrofální následky. To je v kontrastu s implementací tohoto rozšíření — nejen že nijak neošetřuje SMI, ale navíc jako zdroj tiků používá LAPIC Timer. Celé rozšíření je proto citlivé na maskovaní požadavků přerušení pomoci instrukce CLI[12]. Tato instrukce by se neměla používat (náhradou je spinlock implementovaný v ACPI HALu pomoci TPR), její výskyt například ve špatně napsaném driveru nelze vyloučit. Jako řešení těchto latencí

[12]Podobně zranitelný je i Real-Time Windows Target a pravděpodobně i Real Time Toolbox, ten však nebyl testován. Naproti tomu, dříve nabízená rozšíření od německé firmy Kuka byla dodávána s kartou, která generovala pravidelná nemaskovatelná přerušení a tento problém eliminovala.

firma doporučuje testování pomoci nabízeného nástroje Platform Evaluator a případnou změnu hardware a tím i ovladačů. Názorně je vidět negativní vliv maskovaní požadavků přerušení po dobu 0.5 s na následujících obrázcích:



Obrázek 3.2: RTX — časovač s periodou 1 ms.



Obrázek 3.3: RTX — časovač s periodou 1 ms, CLI.

Kapitola 4

Řízení mikropolohovací platformy z PC v reálném čase

Mikropolohovací platforma pod mikroskop představuje z pohledu programátora vstupně-výstupní zařízení. Vstupem/výstupem do počítače je napětí na kapacitním senzoru polohy. Výstupem je ovládání rychlosti akčních členů — tří piezoelektrických motorků, které pohybují deskou, na níž je umístěn zmíněný kapacitní senzor.

Z pohledu teorie řízení pak platforma po zjednodušení představuje fyzikální systém, který má dva vstupy (nastavení rychlosti motorků v osách x a y) a dva výstupy (poloha na osách x a y). V tomto systému je ještě podsystém, který představuje kapacitní senzor. Měření napětí (a tedy polohy) totiž neprobíhá přímo, ale pomocí zpětné vazby. Cílem je realizovat zpětnou vazbu pro hlavní systém, která umožní přesné řízení polohy. Od vedoucího bakalářské práce jsem dostal zadané tyto časové požadavky: perioda vzorkování pro měření napětí 10 ms (lépe však 1 ms) a perioda řízení celého systému 100 ms.

Mým úkolem bylo navrhnout a implementovat rozhraní, do kterého bude snadné zahrnout algoritmus zpětnovazebního řízení.

4.1 Měření napětí

Měření napětí na kapacitním senzoru je implementováno pomocí zpětné vazby. Senzor je fyzicky připojen k PCI měřicí kartě MF624 od firmy Humusoft. Tato karta obsahuje 14-bitové převodníky, osm digitálně-analogových a osm analogové-digitálních. Dále nabízí digitální vstupy (8x) a výstupy (8x), vstupy inkrementálních čidel (4x) a čítače/časovače (4x). K měření využijeme DA a AD převodníky. Čas převodu se pohybuje v řádu μs

(nejdéle trvá ustálení analogových výstupů, 30 μs). Karta tedy nebude představovat pro splnění časových požadavků žádný problém.

Kartu lze ovládat třemi způsoby: z prostředí Matlab (Real Time Toolbox, Real-Time Windows Target, xPC Target), prostřednictvím operačního systému pomocí dodaných ovladačů nebo čtením a zapisováním přímo na registry karty. Karta je tedy multiplatformní, nezávislá na OS. Zvolil jsem druhou možnost. Humusoft dodává dynamickou knihovnu (DLL) s hlavičkovými soubory v jazyce C, takže implementovat komunikaci s kartou je i díky kvalitní dokumentaci a příkladům velmi snadné. V podstatě stačí zavolat funkci `HudaqOpenDevice` pro získání tzv. handlu použitého v `HudaqAIReadMultiple`, `HudaqAIWriteMultiple` a provádět jimi čtení resp. zápis více kanálu najednou. Volání těchto funkci trvá řádově jednotky μs.

Při práci jsem se setkal s problémem — nefungovaly první čtyři vstupní analogové kanály. Naštěstí jich má karta celkem osm.

4.2 Řízení piezomotorku

Piezomotorky jsou ovládány pomocí zařízení, které se připojuje přes USB. To generuje n pulzů o zvolené periodě. Zařízení se ovládá opět pomoci dodané DLL knihovny, hlavičkové soubory jsou pro Pascal/Delphi. Dokumentace je v tomto případě stručnější a je v němčině. Jediným podporovaným OS jsou Windows.

Na ovládání stačí dvě funkce (pokud nebereme v potaz kontrolu chyb): `ioSetTimer` a `ioSetState`. Pomoci `ioSetTimer` vybereme směr (dopředu, dozadu, neaktivní) pro osy x a y, periodu a počet pulzů (0 pro neomezený počet). Funkce `ioSetState` zapíná vysoké napětí, LED diodu, případně změní časovou základnu pro periodu signálu. Doba volání těchto funkcí je opět v řádech μs.

4.3 Časové požadavky

Napětí na kapacitním senzoru je nutné vzorkovat s periodou menší než 10 ms. Jak bylo výše uvedeno, samotné volání funkcí ovládajících připojená zařízení je rychlé a nepředstavuje problém. Zbývá tedy najít časovač, který má periodu pod 10 ms. To žádná funkce standardního API Windows nedokáže. Na druhou stranu, požadavky nejsou tak přísně, aby bylo nutné použít hardwarové časovače popsané v předchozí kapitole. Použil jsem tedy knihovnu Multimedia Timers a dvě varianty aktivního čekání.

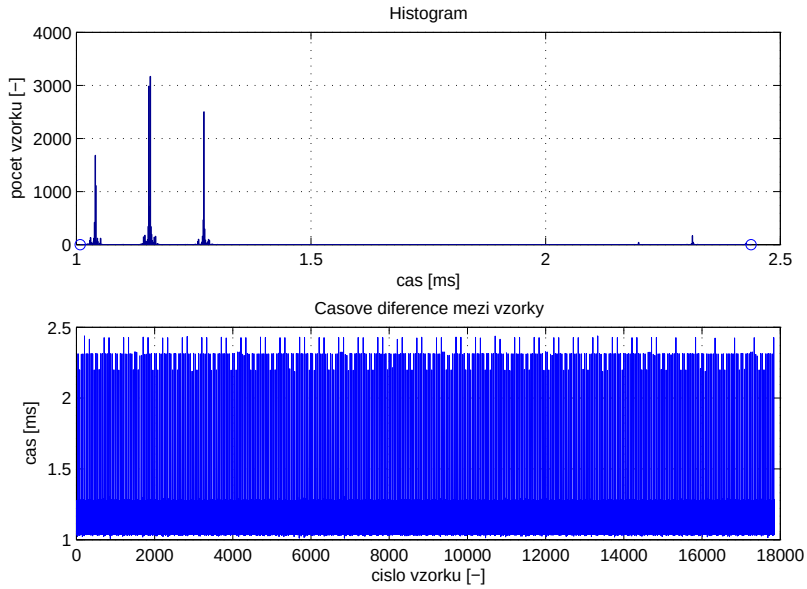
4.3.1 Multimedia Timers

Knihovna MM Timers poskytuje časovač s periodou až 1 ms. Funkce, která je volaná časovačem každou milisekundu pak vypadá takto:

```
void CALLBACK TimerCallback(
        UINT wTimerID,
        UINT msg,
        DWORD dwUser,
        DWORD dw1, DWORD dw2)  {

        // Měření napětí - každou milisekundu.
        HudaqControl();
        if(CallsNr%XYTABLE_INTERVAL_MS==0)
                // Ovládání motorku - každých 100 ms.
                XYTableControl();
        CallsNr++;
}
```

Časovač Multimedia Timer není příliš spolehlivý a při nastavené periodě 1 ms trvala nejdelší prodleva méně než 2.5 ms. Tím se požadované zadání, perioda vzorkování nejvýše 10 ms, podařilo bez problémů splnit. To je vidět na následujícím grafu 4.3.1. Čtenář si může také povšimnout, že časovač dostupný v rozšíření RTX je mnohem přesnější než Multimedia Timer poskytovaný Windows. Naneštěstí, tyto výkyvy jsou problematické při implementaci řídícího algoritmu.



Obrázek 4.1: MM Timer — časovač s periodou 1 ms.

4.3.2 Aktivní čekání

Aktivní čekání, v anglické literturře známé jako busy waiting, je metoda použitelná na víceprocesorových systémech nebo na systémech s vícejádrovými procesory. Jeden procesor se pak stará o záležitosti operačního systému a druhý procesor čeká ve věčné smyčce a kontroluje čas. Ve správný okamžik pak zavolá danou rutinu. Kód v Matlabu může vypadat například takto:

```
while i<N,
    t(i+1) = toc;
    while (t(i+1)-t(i))<Ts,
        t(i+1) = toc;
    end
    u(i+1) = AIRead(merk, 1);
    i = i+1;
end
```

`Toc` měří uplynulý čas a `AIRead` je rutina, kterou chceme v daný čas volat. Pro přístup k PCI kartě lze použít funkce dodané Humusoftem, pro ovládání USB zařízení pak použijeme funkce Matlabu zvané Generic DLL Interface. Stačí vytvořit příslušný

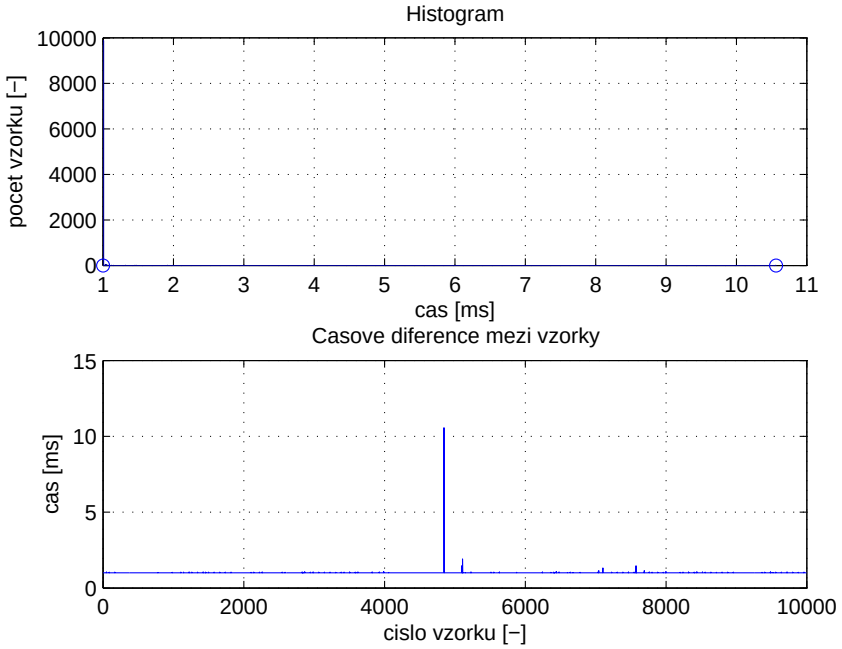
hlavičkový soubor (XYTableUsb.h):

```
int __stdcall ioPending();
int __stdcall ioLastError();
...
```

Poté je nutné načíst DLL a již lze volat její funkce, viz následující ukázka:

```
loadlibrary('XYTableUsb', 'XYTableUsb.h')
libfunctions('XYTableUsb') % zobrazí funkce
libfunctionsview('XYTableUsb') % zobrazí funkce včetně parametrů
calllib('XYTableUsb', 'ioPending') % zavolá funkci
unloadlibrary XYTableUsb
```

Protože však kód Matlabu standardně neběží s nejvyšší prioritou, může být přerušen jinými thready. To je patrné z grafu 4.2. Na jednoprocesorovém systému je tento přístup navíc naprosto nepoužitelný, protože Matlab bude sdílet procesorový čas s ostatními thready.



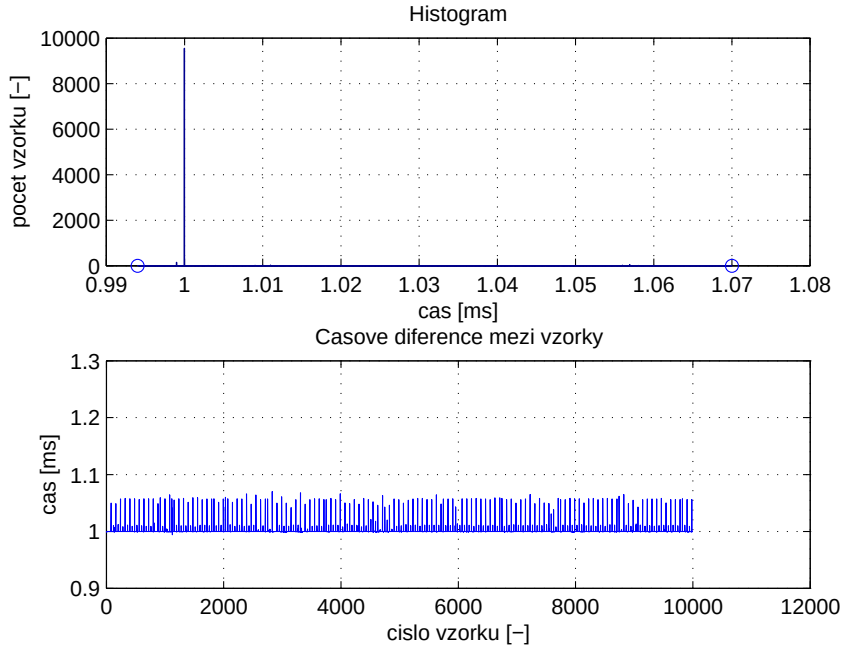
Obrázek 4.2: Busy waiting v Matlabu — časovač s periodou 1 ms.

Metodu busy waiting popsanou výše lze vylepšit následujícím způsobem. Na základě poznatků z kapitoly 2 nastavíme nejvyšší možnou prioritu threadu. Nastavíme také aby

thread běžel pouze na prvním procesoru:

```
SetPriorityClass(GetCurrentProcess(), REALTIME_PRIORITY_CLASS);
SetThreadPriority(GetCurrentThread(), THREAD_PRIORITY_TIME_CRITICAL);
SetThreadAffinityMask(GetCurrentThread(), 1);
```

Nyní již busy waiting thread nemůže být přerušen jiným threadem, pouze požadavkem přerušení. Výsledek je vidět v grafu 4.3. Tento postup lze použít i na jednoprocesorovém systému, počítač však přestane reagovat, dokud busy waiting thread sám neskončí. To nemusí být na závadu, pokud máme pro úlohu vyhrazen zvláštní počítač.



Obrázek 4.3: Busy waiting — časovač s periodou 1 ms.

4.4 Zhodnocení

Čas volání funkcí ovládání PCI karty ani USB zařízení nepředstavuje v požadovaném řádu milisekund žádný problém. Volba časovače: pokud máme k dispozici multiprocesorový počítač nebo vícejaderný procesor (náš případ), jeví se jako nejvhodnější použít busy waiting. Perioda se nezpozdila o více než 70 μs. V případě jednoprocesorového počítače použijeme tuto metodu také, pokud nepotřebujeme aby na počítači běžel jiný proces. V opačném případě lze použít MM Timer, zde však musíme očekávat rozptyl v řádech

milisekund.

Kapitola 5

Závěr

Cílem práce bylo implementovat řízení mikropolohovací platformy z PC v reálném čase, které bude možné snadno rozšířit o zpětnou vazbu. Tento úkol byl úspěšně splněn. Program byl napsán pro operační systém Windows, protože dodané programové vybavení volbu jiného operačního systému neumožňovalo. Podařilo se dosáhnout periody vzorkování 1 ms pomocí aktivního čekání.

Práce také zkoumá možnost realizace plánování označovaného jako Cyclic Executive Pattern na platformě PC, se zaměřením na operační systém Windows XP. Základem tohoto přístupu jsou přesné časovače a nepřerušitelný běh úlohy. Nyní budou výsledky shrnuty a uvedena obecná doporučení.

Na víceprocesorových systémech lze použít aktivní čekání. Jeho přesnost závisí na prostředcích použitých k měření času a na eliminaci nežádoucích přerušení. Velkou výhodou je možnost implementovat tento přístup v user modu a využít tak komfort služeb operačního systému.

Pokud požadujeme periodu časovačů v řádech **sekund** až **stovek milisekund**, je vhodné použít standardní API Windows. Nepřerušitelnost běhu úlohy nelze zaručit, lze se jí však přiblížit nastavením real time priority — thread pak nemůže být přeplánován.

Pro periodu v řádech **desítek** až **jednotek milisekund** se jeví nejvhodnější použít časovače Multimedia Timer, který je standardní součástí Windows. Lze dosáhnout až 1 ms a nejhorší případ zpravidla nepřekročí 3 ms. Časovač je stabilní i při zatíženém systému. Kupodivu, ani v kernelu nenalezneme lepší časovač, ekvivalentní API dokonce není z kernel módu přístupné (není exportované).

Periodu v řádech **jednotek** až **desetin milisekund** nabízí real time rozšíření RTX firmy Ardence. Časovač je stabilnější (jitter byl okolo 20 μs) než Multimedia Timer.

Nevýhodou je vysoká cena činící několik tisíc dolarů.

Přerušení s periodou v řádech blížících se k **desetinám mikrosekundy** lze získat s některým z hardwarových časovačů. Periody kratší než několik μs nemají smysl, vzhledem k době vyřizování požadavků přerušení. Tyto časovače je vhodné využít aperiodicky, kdy se plně projeví jejich přesnost. V takto krátkých časových okamžicích musíme brát v úvahu řadu faktorů a na použití služeb operačního systému můžeme zapomenout. Nepřerušitelnosti úlohy se lze přiblížit maskováním přerušení a ošetřením SMI. Z hardwarových časovačů lze pro většinu soft real time aplikací doporučit LAPIC Timer (je i základem pro RTX a dalších rozšíření).

Měření času v řádech **nanosekund** lze provádět pomoci instrukce RDTSC. Musíme však zvážit, zda to pro naší aplikaci má vůbec nějaký smysl. Například latence při přístupu na sběrnici PCI mohou být řádově mikrosekundy, nemluvě o vlivu negativních faktorů, jako např. probíhající DMA přenos. Tato přesnost by mohla být užitečná pro zařízení připojená do slotu paměti SDRAM nebo pro měření doby trvání jednotlivých instrukcí.

Literatura

[1] RUSSINOVICH, Mark E., SOLOMON, David A. *Microsoft® Windows® Internals, Fourth Edition: Microsoft Windows Server™ 2003, Windows XP, and Windows 2000*. [s.l.] : Microsoft Press, 2004. 976 s. ISBN 0735619174.

[2] DOUGLASS, Bruce Powel. Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems. [s.l.] : Addison Wesley, 2002. 528 s. ISBN 0-201-69956-7.

[3] ONEY, Walter. Programming the Microsoft Windows Driver Model. 2nd compl. edition. Redmond, Washington : Microsoft Press, 2003. 880 s. ISBN 0-7356-1803-8.

[4] JONES, Michael, REGEHR, John. *The Problems You're Having May Not Be the Problems You Think You're Having: Results from a Latency Study of Windows NT* [online]. 1999 [cit. 2007-08-02]. Dostupný z WWW: <http://research.microsoft.com/~mbj/papers/tr-98-29.html>.

[5] *Advanced Configuration and Power Interface Specification* [online]. Hewlett-Packard Corporation, Intel Corporation, Microsoft Corporation, Phoenix Technologies Ltd., Toshiba Corporation. Revision 3.0b. October 10, 2006 [cit. 2007-08-10]. PDF. 611 s. Dostupný z WWW: <http://www.acpi.info/DOWNLOADS/ACPIspec30b.pdf>.

[6] *PC 99 System Design Guide : A Technical Reference for Designing PCs and Peripherals for the Microsoft® Windows® Family of Operating Systems* [online]. Intel Corporation, Microsoft Corporation. Version 0.3. February 3, 1998 [cit. 2007-08-10]. PDF. 521 s. Dostupný z WWW: <http://www-pc.uni-regensburg.de/hardware/TECHDOK/PC_99_1.pdf>.

[7] *IA-PC HPET (High Precision Event Timers) Specification* [online]. Intel Corporation. Rev. 1.0a. October 2004 [cit. 2007-08-10]. PDF. 33 s. Dostupný z WWW: <http://www.intel.com/technology/architecture/hpetspec.htm>.

[8] *Using the RDTSC Instruction for Performance Monitoring* [online]. Intel Corporation. 1998 [cit. 2007-08-10]. PDF. 12 s. Dostupný z WWW: <http://www.ccsl.carleton.ca/~jamuir/rdtscpm1.pdf>.

[9] *Intel® I/O Controller Hub 8 (ICH8) Family : Datasheet* [online]. Intel Corporation. May 2007 [cit. 2007-08-10]. PDF. 890 s. Dostupný z WWW: <http://www.intel.com/design/chipsets/datashts/313056.htm>.

[10] *82093AA I/O ADVANCED PROGRAMMABLE INTERRUPT CONTROLLER (IOAPIC)* [online]. Intel Corporation. May 1996 [cit. 2007-08-10]. PDF. 20 s. Dostupný z WWW: <http://www.intel.com/design/chipsets/datashts/29056601.pdf>.

[11] *IA-32 Intel® Architecture Software Developer's Manual : Volume 3A: System Programming Guide, Part 1* [online]. Intel Corporation. June 2006 [cit. 2007-08-10]. PDF. 640 s. Dostupný z WWW: <http://www.intel.com/design/processor/manuals/253668.pdf>.

[12] *IA-32 Intel® Architecture Software Developer's Manual : Volume 3B: System Programming Guide, Part 2* [online]. Intel Corporation. June 2006 [cit. 2007-08-10]. PDF. 530 s. Dostupný z WWW: <http://www.intel.com/design/processor/manuals/253669.pdf>.

[13] *Intel® Core™2 Extreme Quad-Core Processor QX6000$^{\Delta}$ Sequence and Intel® Core™2 Quad Processor Q6000$^{\Delta}$ Sequence : Datasheet* [online]. Intel Corporation. August 2007 [cit. 2007-08-10]. PDF. 98 s. Dostupný z WWW: <http://download.intel.com/design/processor/datashts/31559205.pdf>.

[14] *AMD64 Technology : AMD64 Architecture Programmer's Manual Volume 2: System Programming* [online]. Advanced Micro Devices, Inc. Revision: 3.12. September 2006 [cit. 2007-08-10]. PDF. 488 s. Dostupný z WWW: <www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/24593.pdf>.

[15] *BIOS and Kernel Developer's Guide for AMD Athlon™ 64 and AMD Opteron™ Processors* [online]. Advanced Micro Devices, Inc. Revision: 3.30. February 2006 [cit. 2007-08-10]. PDF. 402 s. Dostupný z WWW:

<www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/26094.PDF>.

[16] RYŠÁNEK, František. *PCI Express - mýty a fakta* [online]. [cit. 2007-08-10]. Dostupný z WWW: <http://www.fccps.cz/download/adv/frr/pci-e/pci-e.htm>.

[17] RYŠÁNEK, František. *Routing perušení a kolize prostedku na platform x86 aneb sedm generací PC AT* [online]. [cit. 2007-08-10]. Dostupný z WWW: <http://www.fccps.cz/download/adv/frr/x86.pdf>.

[18] Microsoft Corporation. *Inside Windows NT High Resolution Timers* [online]. November 1, 2006 [cit. 2007-08-10]. Dostupný z WWW: <www.microsoft.com/technet/sysinternals/information/highresolutiontimers.mspx>.

[19] Microsoft Corporation. *Guidelines For Providing Multimedia Timer Support* [online]. September 20, 2002 [cit. 2007-08-10]. Dostupný z WWW: <http://www.microsoft.com/whdc/system/CEC/mm-timer.mspx>.

[20] Microsoft Corporation. *Windows Research Kernel* [online]. October 12, 2006 [cit. 2007-08-10]. Dostupný z WWW: <http://www.microsoft.com/resources/sharedsource/licensing/researchkernel.mspx>.

[21] Microsoft Corporation. *Jak systém Windows určuje kompatibilitu s rozhraním ACPI* [online]. Revize 3.0. 7. ledna 2004 [cit. 2007-08-10]. Dostupný z WWW: <http://support.microsoft.com/kb/216573/cs>.

[22] Microsoft Corporation. *Jak přesunout instalaci systému Windows na jiný hardware* [online]. Revize 14.2. 21. listopadu 2006 [cit. 2007-08-10]. Dostupný z WWW: <http://support.microsoft.com/kb/249694/cs>.

[23] Microsoft Corporation. *Možnosti přepínačů v souboru Boot.ini v systémech Windows XP a Windows Server 2003* [online]. Revize 5.0. 31. května 2006 [cit. 2007-08-10]. Dostupný z WWW: <http://support.microsoft.com/kb/833721/cs>.

[24] Bios Central. *CMOS Memory Map* [online]. [cit. 2007-08-10]. Dostupný z WWW: <http://bioscentral.com/misc/cmosmap.htm>.

[25] DataRescue. *IDA Pro Disassembler* [počítačový program]. Ver. 5.0. [Liege, Belgium], 2007 [cit. 2007-08-10]. Dostupný z WWW: <http://www.datarescue.com/idabase/index.htm>.

[26] Microsoft Corporation. *Debugging Tools for Windows* [počítačový program]. Ver. 6.6.7.5. July 18, 2006 [cit. 2007-08-10]. Dostupný z WWW: <http://www.microsoft.com/whdc/devtools/debugging/installx86.mspx>.

[27] Microsoft Corporation. *KrView - the Kernrate Viewer* [počítačový program]. Ver. 5.2.3790.1101. January 14, 2004 [cit. 2007-08-10]. Dostupný z WWW: <http://www.microsoft.com/whdc/system/sysperf/krview.mspx>.

[28] Microsoft Corporation. *DDK — Windows Driver Development Kit* [počítačový program]. Ver. Windows Server 2003 SP1 DDK. [cit. 2007-08-10]. Dostupný z WWW: <http://www.microsoft.com/whdc/devtools/ddk/default.mspx>.

[29] Lavalys Consulting Group, Inc. *EVEREST Ultimate Edition* [počítačový program]. Ver. 4.0. 2007-04-05 [cit. 2007-08-10]. Dostupný z WWW: <http://www.lavalys.com/products/overview.php?pid=3&ps=UE&lang=en>.

[30] PEACOCK, Craig. *PortTalk* [počítačový program, open source]. Ver. 2.2. 6th April 2007 [cit. 2007-08-10]. Dostupný z WWW: <http://www.beyondlogic.org/porttalk/porttalk.htm>.

[31] OpenLibSys.org. *WinRing0* [počítačový program, open source]. Ver. Alpha11. 2007/08/04 [cit. 2007-08-10]. Dostupný z WWW: <http://openlibsys.org/>.

[32] Ardence, Inc. *RTX® — Real-time Extension for Control of Windows®*. [počítačový program]. Ver. 7.0.0.0. [cit. 2007-08-10]. Dostupný z WWW: <http://www.ardence.com/embedded/products.aspx?ID=70>.

[33] HUMUSOFT s.r.o. *Real Time Toolbox* [počítačový program]. Ver. 4.0. [cit. 2007-08-10]. Dostupný z WWW: <http://www.humusoft.cz/rt/index.htm>.

[34] The MathWorks, Inc. *Real-Time Windows Target* [počítačový program]. Ver. 2.7. [cit. 2007-08-10]. Dostupný z WWW: <http://www.mathworks.com/products/rtwt/>.

[35] GROBLER, Janno, KOURIE, Derrick. Design of a High Resolution Soft Real-Time Timer under a Win32 Operating. In *SAICSIT*. [s.l.] : [s.n.], 2005. s. 226-235.

[36] CINKELJ, Justin, MIHELJ, Matjaz, MUNIH Marko. Soft Real-Time Acquisition in Windows XP. In *WISES*. Hamburg, Germany : Hamburg University of Technology, May 20, 2005. s. 110-116. ISBN 3-902463-03-1.

[37] HARNESK, Andreas, TENSER, David. *Real-Time Performance of Windows XP Embedded*. [s.l.], April 30, 2006. 74 s. Mälardalen University, Department of Computer Science and Electronics, Västeras, Sweden. Vedoucí diplomové práce Frank Lüders. Dostupný z WWW: <http://www.idt.mdh.se/utbildning/exjobb/files/TR0470.pdf>.

Dodatek A

Zkratky

AD	Analog to Digital
APC	Asynchronous Procedure Call (Windows NT)
API	application programming interface
APIC	Advanced Programmable Interrupt Controller, tvoren LAPIC a IOAPIC
BIOS	Basic I/O System
DA	Digital to Analog
DDK	Driver Development Kit
DPC	Deferred Procedure Call (Windows NT)
EOI	End of Interrupt (LAPIC, PIC)
GPOS	General Purpose Operating System
HPET	High Performance Timer
IDT	Interrupt Descriptor Table
INTI	také INTIN, Interrupt input (IOAPIC)
INTR	Interrupt Request
IOAPIC	I/O Advanced Programmable Interrupt Controller (např. 82093AA)
IPI	Inter-Processor-Interrupt
IRQ	Interrupt Request or the interrupt input lines
IRQL	Interrupt Request Level (Windows NT)
ISR	Interrupt Service Routine
LAPIC	Local Advanced Programmable Interrupt Controller
LVT	Local Vector Table (LAPIC)
MADT	Multiple APIC Description Table
MSI	Message Signaled Interrupt (PCI)

MSR	Machine Specific Register
MTRR	Memory Type Range Register
NB	North Bridge, Host-to-PCI
NMI	Non Maskable Interrupt
OS	Operating System
PCI	Peripheral Component Interconnect
PIC	Programmable Interrupt Controller (napr. 8259A)
PIT	Programmable Interval Timer
PM	Power Management
RE	Reverse Engineering
RTC	Real Time Clock
RTOS	Real-time operating system
SB	South Bridge
SDK	Software Development Kit
SDRAM	Synchronous Dynamic Random Access Memory
SM	System Management
SMI	System Management Interrupt
SMM	System Management Mode
UTC	Coordinated Universal Time
WMI	Windows Management Instrumentation

Dodatek B

Příkazy Windbg

!apic ACPI informace o Local Advanced Programmable Interrupt Controller.

!fadt ACPI informace o konfiguraci systému.

!i{b,w,d} cislo_portu Přečte byte/slovo/dvojité slovo ze zvoleného portu.

!ioapic Informace o IO Advanced Programmable Interrupt Controller.

!mapic Zobrazí ACPI tabulku s informacemi o řadičích přerušení.

!pic Informace o Programmable Interrupt Controller.

!rdmsr cislo_registru Přečte zadaný Machine Specific Register.

Dodatek C

Obsah přiloženého CD

bp.pdf Tato bakalářská práce.

Zdrojové kódy Zdrojové kódy řízení mikropolohovací platformy, včetně potřebných knihoven.