Portál pre odborné publikovanie ISSN 1338-0087

Systémy reálneho času a Java Real-time System

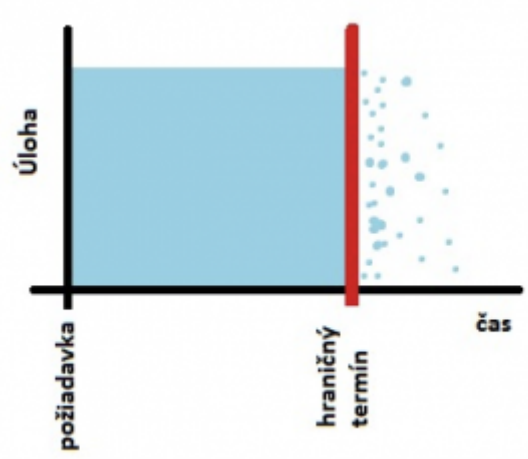
Bielko Samuel · Informačné technológie

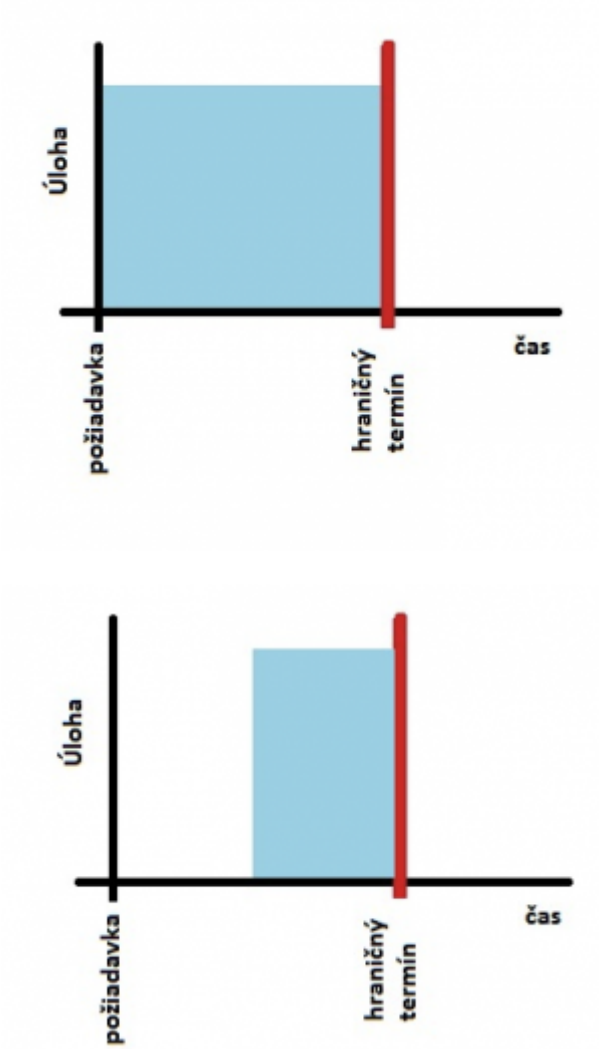
29.04.2011

Keď sa povie real-time systém, predstava o tom čo vlastne real-time je, nie je vždy jednotná. Niektorí si predstavia niečo veľmi výkonné a rýchle, iní zasa nejakú neustálu činnosť. To do definície real-time spadá, avšak ani zďaleka to nie je úplná definícia. Nakoľko, keď si zoberieme napríklad budovu, aká je jej časová konštanta? Je to milisekunda či sekunda? Skôr by som tvrdil, že hodiny až deň, a predsa stále existuje v reálnom čase.

Preto by sa real-time systém dal definovať nasledovne: ,,Hlavnou úlohou pri real-time systéme je, aby systém vykonal svoje úlohy ešte pred definovaným hraničným termínom. Pričom nezáleží, či je tento hraničný termín meraný v milisekundách alebo dňoch, pokiaľ je daná úloha ukončená pred hraničným termínom, systém je považovaný za real-time.”[1]

Existuje niekoľko druhov real-time systémov: soft, hard a isochronal real-time. Pri soft real-time sa príležitostne môže prekročiť hraničný termín, bez toho aby vznikla nejaká vážnejšia chyba. Pri hard real-time sa po prekročení hraničného termínu vykonávanej úlohy systém dostáva do abnormálneho stavu. Pri isochronal real-time musí systém odpovedať na požiadavku, resp. vykonať úlohu ešte pred hraničným termínom, ale ani nie príliš skoro. Má teda presne vyhradený časový rámec.

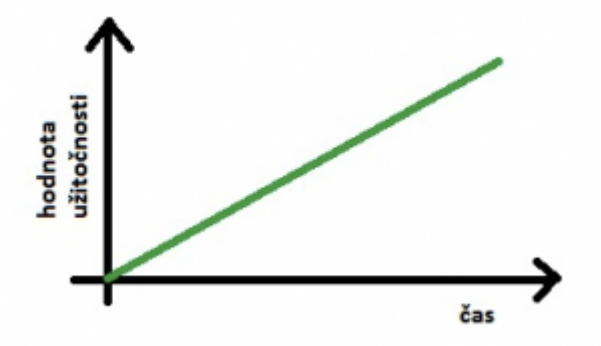


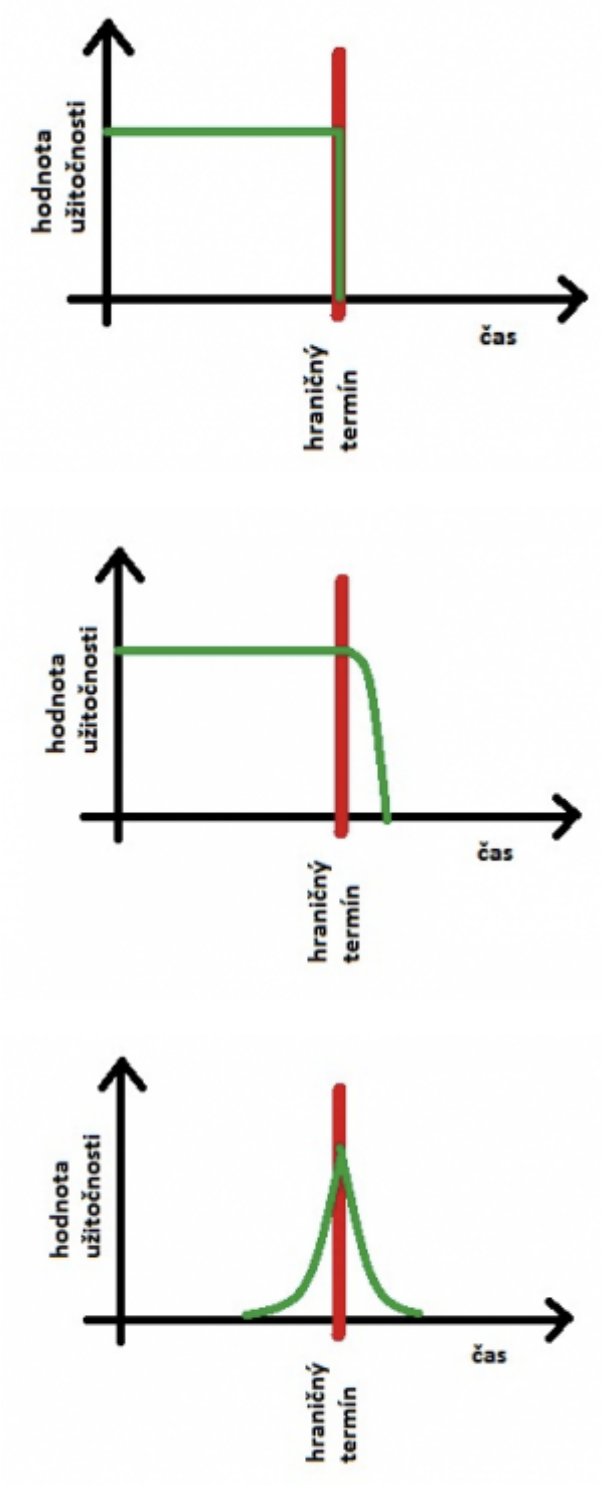


Obr.1: grafické znázornenie real-time systémov a) soft b) hard c) isochronal

Druh real-time systému sa dá charakterizovať aj hodnotou užitočnosti ukončenia úloh:

1. Pri nonreal-time systéme je vnímaná hodnota ukončenia úloh priamo úmerná celkovému počtu za čas.
2. V soft real-time systéme hodnota ukončenia úloh po hraničnom termíne postupne slabne.
3. Hodnota ukončenia úloh v hard real-time systéme je nulová v momente ako uplynie hraničný termín.
4. Hodnota ukončenia úloh v isochronal real-time je nulová, keď sa ukončia skoro alebo neskoro.





Obr.2: hodnota ukončenia úloh pre a) nonreal-time b) hard real-time c) soft real-time d) isochronal real-time

Dôležité vlastnosti real-time systémov

Real-time systém musí byť predvídateľný a deterministický. Predvídateľnosť znamená, že musíme byť schopní matematicky určiť množstvo práce, ktorú je nutné vykonať a či je to vôbec možné pred hraničným termínom. Determinizmus znamená, že systém sa bude správať tak, ako bolo plánované, a to po celý čas a bez problémov. S týmito dvoma kvalitami sa spájajú pojmy latencia a jitter. Latencia je čas medzi konkrétnou udalosťou a odpoveďou systému na ňu, jitter je vlastne kolísanie latencie pre podobné spracovávanie udalostí.

Rýchlosť je často potrebná ale real-time nie je len o nej. Dôležité je, aby real-time systém spĺňal individuálne požiadavky jednotlivých úloh, a teda neprekračoval hraničné termíny. Ak však požadujeme nízku reakčnú dobu, treba systém podporiť kvalitným hardvérom. Taktiež nie je nutnou podmienkou, aby mal real-time systém vysoký prechodový výkon, nakoľko vtedy veľmi kolíše latencia vykonávania jednotlivých úloh, teda sa tu úplne stráca kvalita predvídateľnosti a determinizmu. Preto spravidla real-time systémy mávajú nižší prechodový výkon.

Aby sa úlohy vykonali, musia mať vyhradený čas na procesore. Plánovanie vykonávania úloh majú na starosti plánovače (Scheduler), ktoré sa vyrovnávajú s tromi hlavnými skupinami obmädzení:

zdroje:

Jedná sa najmä o hardvérové zdroje ako je CPU, pamäť, komunikačná linka a pod, ale aj zdieľané objekty. Pri vykonávaní úloh často vzniká potreba zablokovania zdrojov. V real-time systéme treba však zabezpečiť, aby kritické úlohy pokračovali bez obmedzení. To sa dá dosiahnuť prideľovaním priorít alebo vymedzením procesorových skupín čisto len pre kritické vlákna a úlohy.

priorita:

Najkritickejšie úlohy musia mať spravidla najvyššiu prioritu. Na opis prideľovania priority sa dá využiť takzvaná analógia s diaľnicou. Jazdné pruhy pre autá predstavujú úlohy a jazdné pruhy predstavujú zdroje (CPU, pamäť,..). Ak je na diaľnici veľa áut, premávka je pomalá a nepredvídateľná. Ak sa však pridá na diaľnicu prioritný pruh, autá s vysokou prioritou(sanitky, vládne kolóny, policajti...) sa môžu pohybovať predvídateľne a bez obmedzení.

Teda úlohy ktoré sú v prioritnom pruhu majú zabezpečený dostatok zdrojov na svoje vykonanie, pri preťažení systému sú ovplyvnené iba nízkoprioritné úlohy. Do prioritného pruhu vstupujú úlohy len v určitých konkrétnych bodoch. Kvôli prioritnému pruhu systém spravidla stratí prechodový výkon, ale získa na predvídateľnosti a determinizme.

načasovanie:

Plánovače musia zohľadňovať množstvo časových parametrov, aby sa zabezpečilo bezproblémové vykonanie úlohy ešte pred uplynutím hraničného termínu. Patria sem najmä: perióda, hraničný termín, obmeny v časoch vypustenia, čas začiatku a vykonávania, latencia a ďalšie.

Čo prináša Java Real-Time System

Java Real-Time System(RTS) posúva Javu do novej úrovne, nakoľko táto implementácia sa už dokáže vysporiadať s časovo náročnými a deterministickými úlohami. V nasledujúcich riadkoch zľahka načrtnem novinky, ktoré táto implementácia prináša.

Java RTSJ (Real-Time Specification for Java) prináša 7 hlavných oblastí vylepšenia pre vývoj real-time aplikácií v Jave, pričom syntax sa vôbec nezmenila. A sú to: plánovanie

vlákien, manažment pamäte, zdieľanie zdrojov, obsluha asynchrónnych udalostí, asynchrónny prenos riadenia, asynchrónne ukončovanie vlákien a prístup k fyzickej pamäti. Všetky tieto vylepšenia sú dostupné v rámci balíka javax.realtime. Ďalšou veľkou zmenou je použitie Real-Time Garbage Collectora (GC). Tento pracuje konkurenčne a inkrementálne, nikdy neprerušuje kritické vlákna a odstraňuje nedostatky svojich predchodcov ako časovo a pracovne založené garbage collectory.

Real-time vlákna:

Java RTS obsahuje interface Schedulable, ktorý rozširuje Runnable, pričom všetky plánovateľné objekty sú jeho inštanciou. V Jave RTS sú štyri triedy implementujúce Schedulable, a to: RealtimeThread (RTT), NoHeapRealtimeThread (NHRT), AsyncEventHandler (AEH), BoundAsyncEventHandler (BAEH). Pomocou podtried singletonu PriorityScheduler sa dajú objektom tried Schedulable priradiť rôzne rozsahy priorít a možnosti realizovateľnosti. Jedná sa o nasledovné podtriedy: PriorityParameters, ImportanceParameters a ReleaseParameters. Trieda ReleaseParameters je abstraktná, preto sa musia využívať jej podtriedy PeriodicParameters, AperiodicParameters a SporadicParameters.

RealtimeThread trieda rozširuje java.lang.Thread a implementuje Schedulable interface. Rozdielom medzi java.lang.Thread a RTT je, že aj tá najnižšia priorita RTT je vyššia ako u GC a ostatných vlákien. Inak sa správa podobne. Pri RTT môžeme pomocou SchedulingParameters , ReleaseParameters a MemoryParameters objektu špecifikovať periódu, prideľovanie pamäte a ďalšie.

NoHeapRealtimeThread je typom RTT, ktoré nemá prístup k heap pamäti. Objekty tohto typu musia byť vytvorené a spustené iba v regiónoch pamäte scoped a immortal.

Pamäť:

RTSJ definuje 4 pamäťové regióny: heap, scoped memory, immortal memory, a physical memory. Tieto sú reprezentované Java triedami: HeapMemory, ScopedMemory, ImmortalMemory, ImmortalPhysicalMemory, ktoré rozširujú abstraktnú základnú triedu MemoryArea. Heap reprezentuje časť voľnej pamäte pre dynamickú alokáciu a automatickú regeneráciu Java objektov a je predstavovaná jedinečným Java objektom HeapMemory. V RTSJ, heap zostáva ten istý, ako je definovaný v štandardnej Jave, ako aj väčšina pravidiel.

Scoped memory je prvý nový koncept manažmentu pamäte, ktorý RTSJ prináša. Je to región pamäte reprezentovaný objektom z triedy ScopedMemory alebo jednej z podtried. Môžeme špecifikovať parametre ako počiatočnú veľkosť, maximálnu veľkosť, po ktorú môže narásť a voliteľne aj Runnable objekt, ktorý sa realizuje v rámci scoped. Scoped memory nie je predmetom zbiehania GC. Namiesto toho využíva formu referenčného počítania aby zrušila celé obsahy scope naraz.

Immortal memory je špeciálny globálny región, ktorý je vytvorený keď štartuje real-time Java Virtual Machine (VM) a je prístupný cez globálny unikátny objekt ImmortalMemory. V RTSJ je iba jeden takýto región a objekty vytvorené v ňom žijú rovnako dlho ako Java VM. Objekty v immortal memory nikdy nie sú objektom meškaní spôsobených GC, nakoľko tu nikdy nezbieha. Prednastavené sídlia v immortal memory

všetky statické dáta a realizujú sa tu statické inicializátory. Je to dôležité vedieť, nakoľko immortal memory je limitovaný GC-om neovplyvnený zdroj, ktorý sa dokáže rýchlo minúť keď deklarujeme premenné a triedy ako static a často používame statické inicializačné bloky v rámci kódu.

Phisical memory sa používa najmä vtedy, keď potrebujú real-time aplikácie komunikovať s nízkym oneskorením cez špecializovaný hardvér s vonkajším svetom. V rámci MemoryArea existuje mnoho metód, ktoré nám umožnia určiť veľkosť, zistiť množstvo zostávajúceho priestoru a iné.

Synchronizácia:

Synchronizácia prístupu k zdrojom pri aplikácii s viacnásobnými vláknami je pri real-time požiadavkách kritická, nakoľko blokovanie zdrojov môže ovplyvniť prioritu a spôsobiť rôzne oneskorenia. Preto RTSJ určuje takzvané čakacie rady pre synchronizáciu, ktoré využívajú tieto dve pravidlá:

- blokované vlákna pripravené bežať dostanú prístup k synchronizovaným zdrojom v prioritnom poradí
- aby sa vyhlo oneskoreniu, tak pre prístup ku všetkým zdieľaným zdrojom musí byť použité riadenie inverznej priority

Riadenie inverznej priority znemená, že keď kvôli zámku na objekte môže vlákno s nižšou prioritou zbehnúť prednostne pred vláknom s vyššou prioritou.

,,Používanie NoHeapRealtimeThread je výhodné z hľadiska determinizmu a minimálneho oneskorenia, avšak obmedzením môže byť, že tieto nemajú prístup k heap pamäti, napríklad ak si potrebujú vymeniť dáta s klasickým vláknom. Na uľahčenie tejto výmeny medzi NHRT a zvyškom Java aplikácie, RTSJ definuje wait-fre--queue triedy, ktoré podporujú neblokovaný chránený prístup k zdieľaným zdrojom.”[1] Taktiež je zabezpečené, aby kód v NoHeapRealtimeThread neblokoval aktivitu GC, keď sa synchronizuje s objektom zdieľaným štandardným vláknom. Tieto čakacie rady sú zahrnuté v triedach: WaitFreeReadQueue , and WaitFreeWriteQueue.

WaitFreeWriteQueue je pre použitie RTT vláknom, ktoré nesmie nikdy blokovať keď pridáva objekty do radu. Do radu sa vkladá metódou write, pričom táto metóda je časovo spútaná a nespôsobuje časovú nestálosť. Ak máme viac ako jeden Schedulable objekt na písanie do radu, musíme vlastnoručne synchronizovať prístup k radu.

Producenti nikdy neblokujú, ale konzumenti môžu. Keď konzument nestíha s rýchlosťou pridávania správ do radu, je tam potenciál, že narastie do nekonečna, avšak počítač samozrejme nemá nekonečný úložný priestor. Aby sa to zvládlo, vždy musíme udať maximálnu kapacitu radu. Potom ak sa producent pokúsi write do radu, ktorý dosiahol maximálnu kapacitu, vráti sa jednoducho false a objekt sa nevloží. Ak však je nejaký obzvlášť dôležitý objekt a už je rad plný, môžeme použiť metódu force.

Táto dokáže prepísať najnovšie vložený objekt novým v prípade, že je rad plný. Metóda read zavolaná konzumentom vráti null, ak je rad prázdny alebo správu, ak nejaká čaká, pričom nespôsobuje časovú nestálosť a je časovo presne určiteľná. Na druhej strane, producent dostane blok pri write, ak je rad už plný. Viacnásobné čítacie vlákna musia

byť synchronizované našim vlastným kódom.

Real-time Clock API:

Java RTS implementuje RTSJ Clock API, ktorý definuje real-time hodiny a prostriedky časovača. Tieto triedy predstavujú tú najlepšiu presnosť a precíznosť vzhľadom na podpory hardvéru. Taktiež umožňujú vytváranie časovačov, či už periodických alebo deterministických. Ich implementácia Javou RTS uspokojuje aj real-time požiadavku na poskytnutie prístupu k časovačom s vysokým rozlíšením.

V Jave RTS sú real-time časové objekty inštanciami tried AbsoluteTime alebo RelativeTime a sú spojené so singleton real-time hodinmi. Obe HighResolutionTime triedy dovoľujú vykonávať časovú aritmetiku, ako sčítavanie a odčítavanie rôznych časových objektov. Trieda AbsoluteTime reprezentuje špecifický časový bod, ktorý je udaný a meraný kombináciou milisekúnd a nanosekúnd. Trieda RelativeTime reprezentuje časový interval. Často sa používa najmä pri špecifikácii periódy RT vlákien.

Asynchrónne udalosti:

Java RTS má rozsiahlu podporu pre periodické a plánovateľné RT úlohy, ale veľa udalostí v reálnom svete je asynchrónnych. Na zvládanie rôznych systémových a programátorom definovaných real-time udalostí, ktoré si aplikácia vyžaduje je v RTSJ definovaný asynchronous event handler(AEH). Java RTS umožňuje zviazať vykonávanie kódu k udalosti v rámci Java VM ale aj mimo Java VM. RTSJ špecifikuje dve triedy pre AEH:

- AsyncEvent- objekt tohto typu reprezentuje vlastne udalosť samotnú. Objekt však neobsahuje dáta vzhľadom k udalosti, tie musia byť doručené inými prostriedkami
- AsyncEventHandler- RTSJ Schedulable objekt, ktorý je vykonaný, keď je daná udalosť spustená. Tento objekt má pridružené ReleaseParameters, SchedulingParameters a MemoryParameters objekty.

Výhodou AEH je hlavne to, že nemusíme pre každú udalosť vytvárať vlákno a zaoberať sa zdrojmi pre toto vlákno. Výnimkou je trieda BoundAsyncEventHandler, ktorou môžeme obsluhu udalosti priradiť RT vláknu.

Java RTS nám taktiež umožňuje rýchlo a efektívne preniesť vykonávanie kódu z jednej časti do druhej implementovaním asynchronous transfer of control. S touto možnosťou, vykonávanie kódu v rámci jedného Schedulable objektu môže byť zmenené z jednej metódy do inej cez akcie iného Schedulable objektu.

Real-time systémy modelujúce odozvy na udalosti reálneho sveta často vyžadujú zrušenie vykonávania vlákna ako odpoveď. To sa v Jave SE dalo dosiahnuť metódou destroy, ktorá však mohla viesť k nečakanému stavu, preto je to v RTSJ riešené cez asynchrónne ukončovanie vlákien.

Real-time Garbage Collector (RTGC):

RTGC v Jave RTS VM je založený na téze Rogera Henrikssona. Tento collector pracuje

konkurenčne a inkrementálne, nikdy neprerušuje kritické real-time vlákna a nemá nedostatky ako časovo a pracovne založené kolektory. Vďaka tomu pracuje dobre na viacjadrových procesoroch, iné ako real-time vlákna budú ním oveľa viac ovplyvnené (soft real-time správanie), kritické real-time vlákna sa neprerušujú(hard real-time správanie). RTGC je hlavne samoladiaci – modifikuje sa za behu pomocou analýzy potrieb pamäte, ale sú isté parametre, ktoré môžeme naladiť manuálne. Podľa potreby pracuje v niekoľkých módoch, pričom podmienky prestupov medzi týmito módmi môžeme upravovať cez command line.

V normálnom móde RT vlákna bežia na vyššej priorite ako RTGC, pričom tento konštantne meria najhoršie možné prípady alokácie, aby bol zaručený determinizmus a beží dostatočne dlho na to, aby zabezpečil dostatok voľnej pamäte pre aplikáciu. Ak tento mód nedokáže zabezpečiť dostatok voľnej pamäte, prechádza do boosted módu. Keď rapídne klesá množstvo voľnej pamäte, dokonca až pod tretiu hranicu nazvanú "critical reserved bytes threshold" RTGC vstúpi do deterministického módu. Má boosted prioritu, viac vykonávacích vlákien a všetky nekritické vlákna sú blokované od alokácie pamäte.

OS a Java RTS

Existuje množstvo real-time operačných systémov pre rôzne platformy, napr. Windows CE, RTLinux, LunxOS, QNX, VxWorks atď. Javu RTS je zatiaľ možné používať v troch operačných systémoch (nasledujúce verzie alebo novšie):

- Solaris 10 (Update 6, Update 7)
- SUSE Linux Enterprise Real Time 10 Service Pack 2 (SP2) update 6 (2.6.22.19-0.22-rt kernel)
- Red Hat Enterprise MRG 1.1 Errata (2.6.24.7-126.el5rt kernel)

Solaris

Solaris má podporu real-time už od verzie 8. Medzi hlavné real-time kvality nami testovanej verzie Solaris 10 patria:

- zoskupovanie niekoľkých procesov do skupiny za účelom vykonávania nejakej aplikácie, pričom táto skupina môže byť chránená od prerušení
- Vlákno v Solarise môže patriť do jednej zo 6 skupín: Interactive , Timeshare, Fixed Priority, Fair Share, System a Realtime. RT vlákna tu bežia s najvyššou prioritou. Predbiehajú ich iba prerušenia- interupts.
- vlákna RT triedy sa vybavujú v separovaných radoch od ostatných
- abstrakcia ,,turnstile" na zvládnutie inverznej priority, pričom je zabezpečené, aby sa inverzia priority neobjavovala pri RTT
- tiky časovača sú vysokofrekvenčné, až na nanosekundovej úrovni
- solaris dovoľuje zdieľať dáta medzi procesmi pomocou: POSIX radov správ, POSIX semaforov, zdieľanej pamäte, rúr a ďalších iných možností

Solaris sme vyskúšali na viacerých PC, pričom sme pozorovali, že je vyberavý z hľadiska hardvéru. Mali sme napríklad problémy s grafickou a sieťovou kartou. Pre Java RTS je to však podľa môjho názoru lepší operačný systém, nakoľko má jemnejšie vzorkovanie a nemalú úlohu krá aj fakt, že oba produkty (Solaris a Java RTS)

pochádzajú od toho istého výrobcu. Preto v kombinácii so SPRAC hardvérom tvoria neprekonateľnú trojku. Solaris navyše oproti linuxu poskytuje RT monitorovací nástroj Dynamic Tracing (DTrace), na ktorom je založený aj grafický vizualizátor vlákien. Pomocou tohto nástroja môžeme získať dokonalý prehľad o behu našich kódov a analyzovať jednotlivé vlákna a úlohy.

Linux

Hlavné real-time kvality linuxových distribúcií:

- rovnako ako u Solarisu je tu implementovaný protokol o dedičnosti priority, aby sa vyhlo inverzii priority
- diskrétnosť v mikrosekundách
- všetky vlákna v rámci jadra ako aj aplikácie sú predchádzateľné
- prerušenia môžu prebiehať iba na istom sete CPU, ponechávajúc ostatné procesory nedotknuté
- priraďovanie setu procesorov vláknam
- nová trieda real-time vlákien s jemnejším vzorkovaním priority
- prerušenia sú ovládané vláknami v rámci jadra a môžu byť plánované a predchádzané ako ostatné vlákna v systéme.

My sme skúšali Java RTS na SUSE Linux Enterprise Real-Time 10 a 11. Najskôr bolo potrebné nainštalovať SUSE Linux Enterprise Server a potom sa doinštalovala real-time nadstavba. Nevýhodou Linuxu bolo, že nie je zadarmo tak ako Sloaris, a má hrubšie vzorkovanie. Na druhej strane neboli však žiadne problémy s inštaláciou a so systémom sa pracovalo akosi lepšie.

Pre používanie Java RTS je doporučené minimum dvojjadrové CPU a 512 MB RAM. My sme ju otestovali aj na jednojadrovom procesore, avšak vznikala priveľká latencia a nebolo možné využiť priraďovanie vlákien jednotlivým procesorom, nakoľko všetko zbiehalo na jednom. Čo do vývojových prostredí, je možné pracovať klasicky v NetBeans alebo Eclipse, do ktorých si len pridáme novú platformu a pri vytváraní projektu si nastavíme, s akou verziou Java chceme ďalej pracovať.

Test Java RTS

Na test oneskorení RTT sme si vytvorili program s dvomi vláknami, pričom jedno bolo java.lang.Thread a druhé RealtimeThread. Obe sa opakovali 1000-krát s periódami postupne : 5,4,3,2 ms. V každom vlákne, v každej perióde sa posúvali tri 2D objekty a rátalo sa naprázdno do 100 000. RealtimeThread bolo priradené na jadro 1 procesora Intel Core 2 Duo, a inak celá VM zbiehala na jadre 0. Jadro jedna bolo nastavené tak, aby v ňom žiadne iné ako RealtimeThread vlákna nemohli zbiehať. Meraním sme získali hodnoty, ktoré sú uvedené v tabuľke Tab. 1.

Tab. 1.: Hodnoty oneskorení pri meraní presnosti RTT vlákien

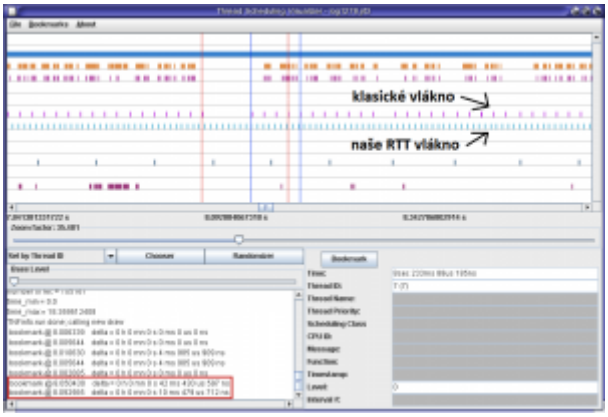
Meranie pri perióde [ms]	Počet prekročení periódy pri 1000 opakovaniach	Priemerná doba prekročenia preiódy[µs]	Maximálne prekročenie [µs]

5 ms	491	7,713	63,213 (419,005)
4 ms	486	6,812	51,143 (173,279)
3 ms	487	4,575	54,145 (210,467)
2 ms	500	2,292	32,456 (142,784)

Hodnoty, ktoré sú uvedené v stĺpci ,,Maximálne prekročenie" v zátvorke sa vzťahujú k prvým periódam, kde sa vlákno inicializovalo, a preto prekročilo periódu o niečo viac ako v ostatných prípadoch. Po celý zvyšok činnosti sa vlákno však správalo lepšie. Pri meraní sme overili garanciu, ktorú vo svojej prednáške ([5]) uvádza G. Bollella. Podľa nej nemá byť oneskorenie RTT väčšie ako 200 µs.

Vlákno java.lang.Thread sa správalo veľmi nepresne a ani zmena periódy veľmi nevplývala na jeho výslednú rýchlosť. Z log súborov vizualizátora vlákien bolo nemožné vyhodnotiť doby jednotlivých periód, nakoľko vlákno zbiehalo miestami nepravidelne a viackrát počas periódy malo zámok na procesore na nepravidelné časy.

Preto sme pre ilustráciu zmerali jeho periódu v grafickom prostredí (Obr. 3.), pričom prednastavená perióda bola 5 ms + čas potrebný na vykonanie úlohy v run metóde (menej ako 1 ms). Z obrázka môžeme odčítať, že pri zmeraní dvoch náhodných periód sme dostali hodnoty 42 ms 430 µs 597 ns a pri druhej perióde 10 ms 479 µs 712 ns. Z tohto jasne vidieť, prečo je Java SE v real-time aplikáciách nepoužiteľná.



Obr. 3.: Zmeranie dvoch náhodných periód java.lang.Thread vlákna

Literatúra

1. Bollella G. , Bruno E. J., Real-Time Java Programming with Java RTS, 2009, USA Crawfordsville: R.R. Donelley 2009. 409 s. ISBN 978-0-13-714298-9
2. Oracle Sun Developer Network: Thread Scheduling Visualizer 2.0, [online], Aktualizované 8.4.2010, Dostupné z http://java.sun.com/javase/technologies/realtime/reference/TSV/JavaRTS-TSV.html
3. ORACLE Sun Developer Network: Java Real-Time System, [online] ,2010, Dostupné z http://java.sun.com/javase/technologies/realtime/index.jsp
4. Princton University: Solaris Process Scheduling, [online], 2007, Dostupné z: http://www.princeton.edu/~unix/Solaris/troubleshoot/schedule.html
5. Bollella, G. 2007. Sun Java Real/Time System 2.0: A deep Dive + Some Random Stuff. [online]. 2007. Dostupné z http://java.sun.com/javase/technologies/realtime/pdf/bollella_deepdiverts.pdf