Systémy reálného času (3)

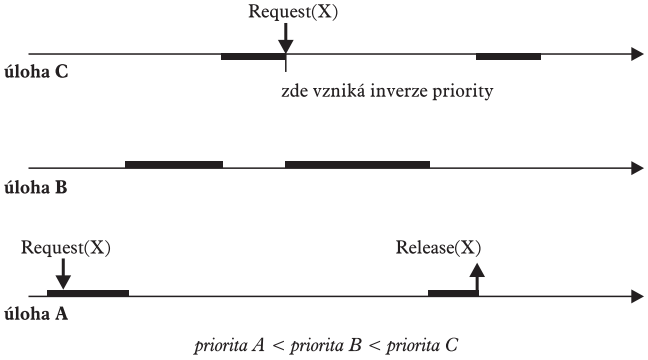
Pevné a dynamické priority

Fixně-prioritní plánovač rozvrhuje úlohy přesně podle předem programátorem stanovených priorit úlohám. Na druhé straně plánovač používající dynamické priority umožňuje měnit priority jednotlivých úloh během chodu systému. Na rozdíl od fixně prioritních plánovačů netrpí problémem tzv. inverze priority.

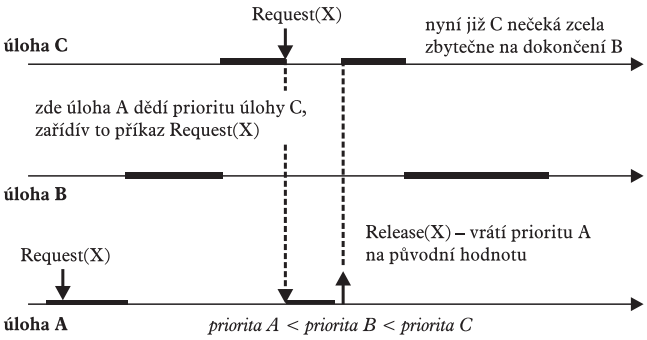
Inverze priority

K inverzi priority dojde např. v případě, kdy chod úlohy s vyšší prioritou je blokován úlohou s nižší prioritou, která má v držení zdroj, který potřebuje ke svému chodu vysokoprioritní úloha. Zdroj je výpočetní prostředek, který je sdílen mezi procesy a který může být v daném okamžiku v držení pouze jediného procesu.

Jednoduchá situace tohoto typu je znázorněna na obr. 4. Nízkoprioritní úloha A potřebuje ke své činnosti zdroj, který má v držení po příkazu Request(X). Přichází úloha B s vyšší prioritou a drží



Obr.4 Inverze priority



Obr.5 Řešení inverze priority děděním priority

procesor až do okamžiku než přijde úloha C s ještě větší prioritou. Ta však zakrátko požaduje zdroj X, který má však v držení úloha A; ta však nemůže pokračovat, protože je zde úloha B s prioritou vyšší než má A a tak C musí čekat nejdříve na dokončení B a pak na dokončení A, než může pokračovat, přestože má ze všech úloh nejvyšší prioritu.

Inverzi priority lze předejít velmi pečlivým programováním RT úloh, avšak lze ji také snadno přehlédnout. Nástroje, které tuto situaci pomohou řešit jsou speciální protokoly; tzv. priority inheritance protocol a priority ceiling emulation protocol. V prvním případě jde o tzv. dědění priority, které spočívá v tom, že zdroj, na nějž čeká vysokoprioritní úloha přiřadí úloze vlastnící zdroj prioritu vysokoprioritní úlohy až do okamžiku, než nízkoprioritní úloha zdroj uvolní.

Při použití druhého protokolu je přiřazena nízkoprioritní úloze dočasně nejvyšší možná priorita nějaké úlohy, která by mohla tento zdroj vyžadovat.

Problémy s prioritním rozvrhováním

Atribut priorita vyjadřuje, jak důležité jsou jednotlivé procesy, resp. úlohy, ale nevypovídá vůbec nic o skutečném časování úloh. Priorita může být teoreticky užitečná zejména v případě, jestliže se systém snaží minimalizovat škody, které vzniknou tím, že se nesplní požadované časové termíny (deadline), protože se snaží splnit termíny nejdůležitějších, resp. nejkritičtějších úloh. Zdá se mnohem rozumnější sdělit systému, kdy má být každá úloha dokončena a jaké k tomu potřebuje zdroje. To je to, co je podstatné u RT systému. Cílem je splnit v čase všechny termíny, ne dělat nejdůležitější věci nejdříve. Proto v případě RT systémů se ukazuje jako užitečnější rozvrhování na základě kritických termínů (deadline scheduling).

Rozvrhování na základě kritických termínů

Rozvrhování procesů na základě kritických termínů vychází z následujících předpokladů

- Rozvrhování programu (úlohy) je primárně řízeno kritickými termíny
- Programátor může přesně určit časové nároky úlohy na její zpracování
- Plánovač může vyhovět všem kritickým termínům

Rozvrhování na základě kritických termínů je jistou formou techniky rozvrhování pomocí dynamických priorit, přičemž priorita je určována de facto plánovačem. Jedním z typických rozvrhovacích algoritmů tohoto druhu je algoritmus EDF (Earliest Deadline First), neboli „nejbližší termín první".

Algoritmus EDF

Pro tento algoritmus platí, že pokud existuje nějaký rozvrh, v němž budou splněny všechny kritické termíny, tak rozvrh, který vybere k provedení úlohu, jejíž kritický termín nastane nejdříve, bude úspěšný. Slabinou EDF algoritmu je to, že nemá způsob jak zjistit, že takto realizovaný rozvrh bude úspěšný anebo způsob jak minimalizovat škody, jestliže některý z kritických termínů nebude splněn. Existují proto modifikace tohoto algoritmu jako například robustní EDF, které se umí vyrovnat se situacemi vznikajícími při přetížení systému, pokud existuje nějaká apriorní informace umožňující tato přetížení vzít do úvahy.

Jinou modifikací algoritmu EDF je varianta, která využívá k rozhodování „nejmenší zbytkový čas", tzv. slack time (nebo laxity time), což je rozdíl mezi časem kdy skončí výpočet a časem kritického termínu. Pro úspěšnost takto vytvářeného rozvrhu platí stejné tvrzení, jako pro čistý EDF algoritmus, na rozdíl od něho však v případě selhání bude toto pravděpodobně méně katastrofické než v případě EDF (protože se vybírají k provedení úlohy, které mají nejmenší časové rezervy).

Periodické rozvrhování

Jestliže velká část zátěže systému může být popsána výroky typu „udělej x každých y mikrosekund", může být užitečný periodický rozvrhovací algoritmus. Pokud bychom se omezili výhradně na periodické úlohy, velmi se problém rozvrhování zjednoduší. Zátěž může být charakterizována seznamem úloh, jejich periodami a dobami výpočtu. Ve srovnání s tím u aperiodických úloh startovaných na základě nějaké události je charakteristickým parametrem doba výpočtu a maximální doba odezvy, která definuje kritický termín reakce. Nejznámějším algoritmem rozvrhování periodických úloh je algoritmus RMS (rate monotonic scheduling), neboli frekvenčně monotónní algoritmus rozvrhování. Jeho princip je založen na fixním přiřazení priorit periodickým úlohám tak, že úlohy s kratší periodou mají větší prioritu. Existuje přesný matematický důkaz, který ukazuje, za jakých podmínek vede toto přiřazení k vytváření tzv. přípustného rozvrhu, v němž všechny periodické úlohy splní své kritické termíny.

Rozvrhování aperiodických úloh

Mnoho podnětů z reálných procesů má aperiodický, sporadický charakter. Nevznikají z předpověditelnou frekvencí. V nejlepším případě mohou vznikat v souladu s normálním statistickým rozložením. V nejhorším případě pak jsou zcela náhodné.

Není-li časové rozložení náhodně vznikajících událostí nějak omezeno není možné, aby na ně systém zaručoval nějakou deterministickou odezvu. Lze tedy použít nějaký rozvrhovací algoritmus podobný EDF, který bude ovšem pracovat pouze do okamžiku, než bude systém takovými aperiodickými událostmi přetížen. V systému obsahujícím jak periodické tak aperiodické procesy, lze tento problém řešit pomocí periodického rozvrhovacího algoritmu takto: Definujeme si specifický periodický proces nazvaný například „nerozvrhovatelné události" a dovolme, aby přerušoval (ve smyslu preempce) všechny rozvrhované procesy. Stanovme maximální dobu, po kterou tento proces poběží a tento čas odečtěme od času spotřebovávaného pro periodické úlohy. Pak tento proces spouštějme jako periodický a přiřaďme mu nějakou periodu. Tento proces, zvaný server, pak nechme ošetřovat v jeho cyklech aperiodické události.

Pro aperiodické události s různými požadavky na dobu odezvy ovšem budeme potřebovat více takových serverů. Uvedený způsob řešení, kdy obsluha aperiodických událostí se svěří do péče periodickému serveru, má výhodu v tom, že lze jistým způsobem používat techniky pro rozvrhování periodických úloh, je však zřejmé, že úspěšnost a efektivnost tohoto postupu při naprosté nepředvídatelnosti aperiodických událostí závisí na vyladění periodických serverů pomocí parametrů doba odezvy a doba výpočtu, které jsou odvislé od typu ošetření aperiodické události a na jejich sladění se zbytkem periodických úloh v systému.

Zpracování přetížení

Nemůže-li plánovač vyhovět všem kritickým termínům, říkáme, že je přetížen. I za těchto okolností by se měl plánovač chovat definovaně. Není to tak jednoduché. EDF algoritmus se slepě snaží vyhovět nejbližšímu kritickému termínu i když je to nemožné a následně způsobí dominový efekt selhání více procesů v nejbližší budoucnosti. Plánovač využívající tzv. slack-time (zbytkový čas) neztrácí čas s beznadějnými případy, ale soustředí se na úlohy s největšími požadavky na procesor a využívá tak času, který by mohl uspokojit více úloh s méně kritickými termíny.

Rozvrhovací algoritmy navržené pro hladkou degradaci systému za podmínek přetížení se snaží přidat nějakou míru důležitosti ke každému kritickému termínu. Nejchytřejší algoritmy umí řídit rozvrhování při přetížení tak, aby se minimalizovala škoda. Naneštěstí tím rozvrhovací algoritmus obvykle ztratí příliš mnoho času a může počítat rozvrh větší dobu, než po jakou ho posléze provádí.

Nelze však navrhnout optimální algoritmus pro zotavení z přetížení v systému s aperiodickými vstupy. Taková šance by existovala jedině tehdy, kdyby byl algortimus „jasnozřivý", neboli kdyby věděl, jaké události se stanou v nejbližším čase.

Časování je často pouze pravděpodobné

Čas výpočtu na moderních procesorech je mimořádně proměnlivý. Dokonce, i když je řízeno stránkování, čas provedení krátké rutiny může být občas řádově o dva řády horší než její typický čas. (*Jistá oblast výzkumu v oblasti systémů reálného času se zabývá problematikou návrhu HW pro použití v RT aplikacích, který by vykazoval obecně nejenom deterministické ale i pro nějaký aplikačně specifický účel optimalizované chování. Viz například* [1].) Pro jednoduché rozvrhovací algoritmy to není nic dobrého. Ty budou slepě sledovat svoji strategii a přitom propásnou důležité kritické časy ve prospěch nedůležitých anebo nechají vzniknout řetězci propásnu-

tých kritických termínů, protože se nedokázaly rozhodnout pro ztrátu jediného (prvního) z nich.

Reakce na nedeterministický hardware je obtížná pro každý rozvrhovací algortimus, avšak prioritní řízení je tolerantní vůči proměnlivým dobám chodu jednotlivých úloh. Ve své podstatě totiž prioritní rozvrhovací algoritmy odsuzují k výpadku úlohy s nejnižší prioritou předpokládajíce, že slovo priorita znamená důležitost. Na druhé straně rozvrhování podle kritických termínů se soustřeďuje na čas CPU a v důsledku toho je poměrně citlivé k časovým změnám.

Faktor využití procesoru

Mějme množinu Γ n periodických úloh, pak faktor využití procesoru je část času procesoru spotřebovaného k provedení množiny úloh [LL]. Protože poměr c_i/T_i je část času procesoru spotřebovaného na provedení úlohy τ_i, je faktor využití n úloh dán jako

$$U = \sum_{i=1}^{n} \frac{c_i}{T_i}$$

Faktor využití procesoru stanovuje výpočetní náklady CPU náležející množině periodických úloh. Ačkoliv lze využití CPU zlepšit zvýšením výpočetní doby úlohy nebo snížením její periody, existuje maximální hodnota U, pod níž je Γ plánovatelná a nad níž je Γ neplánovatelná. Tento limit závisí na konkrétní množině úloh (jejich periodách) a na algoritmu použitém k plánování úloh.

Jestliže faktor využití procesoru množiny úloh je větší než jedna, množina úloh nemůže být plánována žádným algoritmem. Nechť tedy T je nejmenší společná perioda všech period, nebo $T = T_1 \,.\, T_2 \,\ldots\, T_n$. Jestliže $U > 1$, potom také $U \,.\, T > T$, a platí

$$\sum_{i=1}^{n} \frac{T}{T_i} \cdot c_i > T$$

Faktor (T/T_i) udává kolikrát je i prováděna v intervalu T, kdežto $(T/T_i) \,.\, c_i$ je celkový výpočetní čas vyžadovaný τ_i v intervalu T. Suma na levé straně rovnice tedy představuje celkový požadavek výpočetního času vyžadovaného všemi úlohami v intervalu T. Převyšuje-li celkový požadavek dostupný čas procesoru, množina úloh není plánovatelná.

Frekvenčně monotónní algoritmus

Používáme-li fixně prioritní rozvrhovací preemptivní algoritmus (RMS) a zátěž je dána periodickými nezávislými úlohami, můžeme aplikovat frekvenčně monotónní analýzu abychom zjistili, zda náš RMS algoritmus bude na dané množině úloh úspěšný. Princip RMS lze vyjádřit pozoruhodně jednoduchým tvrzením, že „může-li být periodická zátěž rozvrhována fixně prioritním preemptivním algoritmem, může být rozvrhována na základě „monotónního" přiřazení priorit úlohám podle jejich frekvence spouštění. Čím větší je frekvence spouštění úlohy, tím větší bude její priorita".

Liu a Layland [LL] ukázali, že RMS je optimální v tom smyslu, že žádný jiný fixně-prioritní algoritmus nemůže úspěšně rozvrhovat množinu úloh, jestliže tato množina není rozvrhovatelná pomocí RMS. Liu a Layland dále odvodili nejmenší horní hranici faktoru využitelnosti procesoru pro obecnou množinu n periodických úloh.

n	U_{lub}	n	U_{lub}
1	1,000	6	0,735
2	0,828	7	0,729
3	0,780	8	0,724
4	0,757	9	0,721
5	0,743	10	0,719

Obr.6 U_{lub} jako funkce n

Pro libovolnou množinu periodických úloh je nejnižší horní hranice faktoru využití U_{lub} (lub =least upper bound) procesoru v RMS algoritmu $U_{lub} = n(2^{1/n} - 1)$. Tato hranice se snižuje s n, hodnoty pro některá n ukazuje obr. 6.

Pro neomezeně rostoucí hodnoty n konverguje U_{lub} k hodnotě $l_n\, 2 \cong 0{,}69$. To je poměrně pozoruhodný výsledek ukazující, že i při nekonečném počtu procesů, jejichž faktor využití procesoru bude menší než 0.69, stačí použit fixně prioritní RMS algoritmus a poměrně jednoduchý rozvrhovací mechanismus přiřazující v daném okamžiku procesor procesu s nejvyšší prioritou, může bez velkého „přemýšlení" realizovat přípustný rozvrh. Hodnota 0,69 také říká, že v tomto krajním případě musí existovat zhruba jednatřicetiprocentní rezerva ve výkonu procesoru. Svým způsobem tato teorie potvrzuje to, co by každý zkušený inženýr intuitivně vyžadoval, pokud by měl řešit problém rozvrhování periodických procesů. Jednak by přikládal větší důležitost a dával prioritu častěji se opakujícím procesům a dále by určitě vyžadoval, aby zařízení které procesy „realizuje" mělo dostatečnou výkonovou rezervu.

Teorie týkající se rozvrhování procesů je poměrně rozsáhlá a určitě má svou vypovídací sílu. Existuje totiž řada plánovacích algoritmů, lišících se často jen specifikací různých podmínek tvořících předpoklady, za nichž musí být platnost, resp. účinnost těchto algoritmů ověřena. Na druhé straně si je třeba uvědomit, že teoretické výsledky jsou odvozovány právě jen za určitých předpokladů, které v krajních případech nemusejí tak beze zbytku platit. Takovým předpokladem je například zanedbání času na přepnutí kontextu, což pro malá n může být přijatelné, ale pro $n \to \infty$ tomu tak určitě nebude.

Naštěstí v praxi s nekonečně mnoha procesy nepočítáme a tendence dokumentovaná v obr. 6 bude určitě platná; s rostoucím n je třeba počítat s větší rezervou ve výkonu. Sám vztah pro faktor využití procesoru nám dává poměrně zřetelný návod co dělat, pokud nás vypočtená hodnota U neuspokojí : buď zkrátit dobu výpočtu (optimalizací nebo volbou výkonnějšího HW) anebo zvětšit periodu aktivace některých procesů (pokud to dovolují fyzikální resp. technické předpoklady o řízeném systému).

Závěr

V článku byly představeny základní pojmy z problematiky systémů reálného času se záměrem poukázat na některé jejich záludnosti i způsoby řešení vybraných problémů, zejména pokud jde o rozvrhování úloh. Teorie týkající se této oblasti je poměrně rozsáhlá [4]. Při praktickém řešení je však třeba, kromě rozvrhování, vážit i další aspekty ovlivňující chod úloh RT systému. S ohledem na rozsah tohoto článku jsme se však soustředili pouze jen na jeden z nich. Zdá se, že v souvislosti s rozšiřujícím se nasazováním vestavných systémů i s ohledem na postupný návrat jazyka JAVA do této oblasti [6] bude problematika týkající se výzkumu i vývoje RT aplikací stále velmi zajímavým předmětem odborného zájmu a zdrojem intelektuálního potěšení. Neustále rostoucí kvalita i výkonnost mikroprocesorové techniky, pokroky v technologiích programování i stále větší počet realizací řídicích aplikacích jako distribuovaných, však možná soustředí teoretický výzkum v oblasti RT systémů do prostoru především náročných a bezpečnostně kritických aplikací a do oblasti distribuovaných systémů.

Článek byl zpracován v rámci řešení projektu GAČR 102/05/0467.

doc. RNDr. Jindřich Černohorský, CSc.
prof. Ing. Vilém Srovnal, CSc.

Katedra měřicí a řídicí techniky, FEI
VŠB Technická univerzita Ostrava
17. listopadu 15/2172, 708 33 Ostrava-Poruba, ČR
e-mail: jindrich.cernohorsky@vsb.cz
vilem.srovnal.@vsb.cz