Návrh časově kritických systémů IV: realizace prostředky RTOS

V článku je na jednoduchém příkladu ukázán vztah mezi formální specifikací časově kritického systému (RTS, *systém RT*), množinou úloh RT a realizací systému RT s použitím služeb operačního systému reálného času (RTOS). Aby ilustrace byla jednoduchá, stručná a výstižná, vychází se z formální specifikace systému RT vyjádřené prostředky časovaných automatů (TA) a jeho realizace s použitím prostředků RTOS µC/OS-II.

Ve čtvrtém, posledním ze série článků na téma návrhu časově kritických systémů (systémy reálného času, *Real-Time Systems*, *systémy RT*) je ukázán vztah mezi formální specifikací systému RT, množinou úloh RT (blíže viz [5], [6], [7]) a realizací systému RT s použitím prostředků RTOS (*Real-Time Operating System). Vztah je* ilustrován na jednoduchém demonstračním systému RT tvořeném *tlačítkem* určeným ke generování podnětů, *řídicí částí* sloužící k vyhodnocování podnětů a generování reakcí a *svítidlem*, intenzita jehož svitu bude ovládána tlačítkem.

Specifikace demonstračního systému

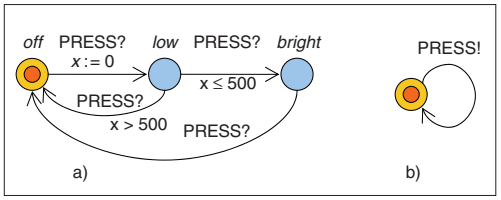
Chování uvažovaného systému lze popsat v přirozeném jazyce následovně (specifikace systému pochází z [1]):

- svítidlo se může nacházet v jednom ze tří stavů:
  - *off* (zhasnuto),
  - *low* (svítí slabě),
  - *bright* (svítí silně);
- počátečním stavem svítidla je stav *off*;
- je-li stisk tlačítka detekován v době, kdy se svítidlo nachází ve stavu *off*, svítidlo začne svítit slabě, tj. přejde do stavu *low*;
- bude-li nejpozději za 500 ms od přechodu do stavu *low* stisknuto tlačítko, svítidlo začne svítit silně, tj. přejde do stavu *bright*;
- je-li stisk tlačítka detekován v době, kdy se svítidlo nachází v jednom ze stavů *low*, *bright*, svítidlo zhasne, tj. přejde do výchozího stavu *off*.

Toto chování lze popsat použitím např. prostředků TA (*Timed Automaton*), jak je ukázáno na *obr. 1*, přičemž chování (stisky) tlačítka je modelováno automatem podle *obr. 1b*, zatímco chování části pro řízení intenzity svitu svítidla automatem vyobrazeným na *obr. 1a*.

Automat podle *obr. 1b* má jediný stav, který je současně stavem počátečním. Jediná hrana tohoto automatu je vždy proveditelná, tzn. že automat je schopen v libovolném čase vyslat po kanálu *PRESS* signál informující o stisku tlačítka. Toto chování je reálné, protože i reálný uživatel může tlačítko stisknout v libovolném čase.

Automat z *obr. 1a* má tři stavy s názvy odpovídajícími slovní specifikaci systému. V každém ze stavů se čeká na stisk tlačítka – v případě daného modelu na příjem signálu na kanálu *PRESS*. Přechod z počátečního stavu (*off*) do stavu *low* je tedy možný až poté, co je detekován stisk tlačítka. Současně s provedením tohoto přechodu jsou nulovány hodiny ($x = 0$), aby bylo možné následně vyhodnocovat dobu strávenou ve stavu *low*. Je-li



Obr. 1. Specifikace vyjádřená prostředky TA (Timed Automaton): a) automat řídící intenzitu svitu, b) automat modelující chování (stisky tlačítka)

v tomto stavu stisknuto tlačítko a přitom neuplynulo více než 500 ms, automat přejde do stavu *bright*. Stisk tlačítka za více než 500 ms od vstupu do stavu *low* vede k přechodu automatu do stavu *off*. Ve stavu *bright* automat čeká na stisk tlačítka; po stisku tlačítka přechází do stavu *off*.

Množina úloh RT

Nyní následuje krátká úvaha související s modelováním chování systému na úrovni množin úloh RT: Jelikož úkolem systému je zpracovat pouze jediný podnět, množina úloh RT bude jednoprvková (tj. $|\Gamma| = 1$) a bude obsahovat pouze úlohu τ_{press} zajišťující reakci na stisk tlačítka. Jelikož tlačítko může být obecně stisknuto v libovolném čase, tato úloha jistě nebude periodická, ale aperiodická či sporadická (s ohledem na konstrukci tlačítka lze totiž stanovit nejkratší možnou dobu mezi dvěma stisky tlačítka prstem lidské ruky, a tedy i nejkratší dobu mezi příchody podnětů vedoucích k vyvolání úlohy τ_{press}). Pro plánování této (jednoprvkové) množiny úloh tedy nelze použít mechanismus přiřazující priority podle hodnot parametru T (např. mechanismus RM), ale je nutné použít jiné mechanismy, např. DM či EDF [7]. Avšak vzhledem k tomu, že množina je tvořena pouze jedinou úlohou, nejjednodušším řešením je přiřadit jí nejvýznamnější prioritu a použít preemptivní či nepreemptivní plánovač.

Bez ohledu na to, zda po realizaci systému bude modelu úlohy τ_{press} odpovídat jedna či několik úloh běžících nad konkrétním RTOS, je důležité si uvědomit, že model úlohy τ_{press} musí zahrnovat zejména režie spojené s detekcí stavu tlačítka (tj. nejdelší možnou prodlevu mezi stiskem tlačítka a zaznamenáním informace o stisku systémem), se šířením informace o stisku (tj. nejdelší možnou prodlevu mezi vysláním signálu informujícího o stisku a příjmem tohoto signálu) a s vyhodnocením podnětu a generováním příslušné odezvy (v daném případě změny intenzity svitu svítidla v závislosti na aktuálním stavu systému).

Poznamenejme, že v případě uvažovaného triviálního systému by pravděpodobně bylo možné obejít se i bez RTOS a vystačit s pouhým využitím přerušovacího podsystému cílové platformy a realizací řídicí části systému v podobě konečného stavového automatu. Z demonstračních důvodů je však následující část článku zaměřena na způsob realizace již popsaného systému RT právě prostřednictvím RTOS. Pro realizaci daného systému byl zvolen µC/OS-II a jeho mechanismus zasílání zpráv. Systém µC/OS-II byl zvolen zejména pro jednoduchost a přehlednost jeho jádra, služeb a pro dostupnost podrobné dokumentace [2] [4]. V současné době je µC/OS-II k dispozici pro většinu existujících platforem a díky jeho dostupnosti ve formě zdrojových kódů může být snadno přenesen na libovolnou další platformu.

Realizace prostředky RTOS µC/OS-II

Základní definice a deklarace potřebné k realizaci jsou uvedeny na *obr. 2* a deklarace proměnných pro příjem zprávy v systému µC/OS-II na *obr. 3* (na řádku 00 je deklaro-

```
01 #define TASK_STK_SIZE          200
02 #define LAMP_STK_SIZE          200
03 #define MAX_MSG_LEN             10
04
05 OS_STK StartupTaskStk[STARTUP_STK_SIZE];
06 OS_STK LampTaskStk[LAMP_STK_SIZE];
```

Obr. 2. Základní definice a deklarace

vána proměnná *msg_press* typu ukazatel na schránku zpráv, na řádku 01 vyrovnávací paměť *buf_msg_press*, určená k uložení obsahu zprávy). Z rozhraní µC/OS-II byly využity zejména tyto služby:

```
01 OS_EVENT *msg_press;
02 INT8U buf_msg_press[MAX_MSG_LEN];
```

Obr. 3. Deklarace proměnných pro využití schránky zpráv indikujících stisk tlačítka

- *OSMboxCreate*, sloužící k vytvoření schránky zpráv; ukazatel na ni je uložen do proměnné *msg_press* (*obr. 4*, řádek 06);

```
01 void  main (void)
02 {
03   CPU_INT08U  err;
04
05   OSInit();                  /* Inicializace struktur jádra uC/OS-II */
06   msg_press = OSMboxCreate((void *)0);
07
08   OSTaskCreate(StartupTask,
09                (void *)0,
10               (OS_STK *)&StartupTaskStk[STARTUP_STK_SIZE - 1],
11               0);
12
13   OSStart(); /* Spuštění plánovače uC/OS-II */
14 }
```

Obr. 4. Tělo funkce main

```
01 static  void  StartupTask (void *p_arg)
02 {
03   (void)p_arg;
04
05   OSTaskCreate(LampTask,
     /* vytvoření úlohy pro ovládání svítidla */
06               (void *)0,
07               (OS_STK *)&LampTaskStk[LAMP_STK_SIZE - 1],
08                                  1);
09
10   for (;;)
11   {
12     if((PTAD&0x04)==0x00)
13     {
14       OSTimeDlyHMSM(0,0,0,10);
15       if((PTAD&0x04)==0x00)
16       {
17         OSMboxPost(msg_press, (void *)&buf_msg_press[0]);
18         OSTimeDly(1);
19         while ((PTAD&0x04)==0x00);
20       }
21     }
22   }
23 }
```

Obr. 5. Tělo úlohy StartupTask

- *OSMboxPost* pro uložení zprávy informující o stisku tlačítka do schránky zpráv určené ukazatelem *msg_press* (*obr. 5*, řádek 17);
- *OSMboxPend* pro čekání na příchod zprávy informující o stisku tlačítka do schránky zpráv určené ukazatelem *msg_press* (*obr. 6*, řádky 10, 15, 20, 25);
- *OSTaskCreate*, určená k vytvoření úloh:
  - *StartupTask*, jejímž úkolem je zavést do systému úlohu *LampTask* (*obr. 5*, řádky 05–08) a poté v dotazovací smyčce testovat příznak stisku tlačítka (*obr. 5*, řádky 10–22),
  - *LampTask*, obsahující realizaci automatu z *obr. 1a.*

V dalším textu je blíže popsáno chování úloh *StartupTask* a *LampTask*.

Úloha *StartupTask*

Po zavedení úlohy *LampTask* do systému (*obr. 5*, řádky 05 až 08) běží úloha *StartupTask* v nekonečné smyčce, v níž na začátku každé iterace testuje (*obr. 5*, řádek 12), zda došlo ke stisku tlačítka (pozn.: tlačítko je připojeno na bit 4 portu A mikrořadiče a jeho stisk je indikován načtením hodnoty *log 0* na vývodu příslušejícím tomuto bitu). Jakmile je na čtvrtém bitu portu A detekována *log 0*, tělo úlohy pokračuje na řádku 14, tj. úloha sebe sama uspí na dobu 10 ms, což postačuje k tomu, aby odezněly zákmity související se stiskem tlačítka. Je-li po uplynutí této doby tlačítko stále stisknuto (*obr. 5*, řádek 15), stisk tlačítka je vyhodnocen jako platný a úloha uloží zprávu do schránky zpráv (*obr. 5*, řádek 17), na dobu jedné jednotky logického času předá k dispozici procesor úloze *LampTask* k obsloužení stisku (*obr. 5*, řádek 18) a poté čeká na uvolnění tlačítka (*obr. 5*, řádek 19). Uvedené řešení bylo zvoleno zejména s ohledem na jeho ilustrativnost, neboť z hlediska praxe by mohlo být výhodnější řešit detekci stisku tlačítka s následným uložením zprávy do schránky zpráv přes přerušovací podsystém cílové platformy, čímž by odpadla nutnost aktivního testování příznaku stisku tlačítka v nekonečné smyčce, a tím i plýtvání výpočetním časem.

Úloha *LampTask*

Úloha *LampTask* běží také v nekonečné smyčce (*obr. 6*, řádky 06 až 29). Začátek těla smyčky představuje stav *off* systému, tj. svítidlo je zhasnuto (*obr. 6*, řádek 09), a poté úloha čeká na příchod zprávy (*obr. 6*, řádek 10). Během čekání je úloha blokována, tj. je vyřazena z množiny připravených úloh. Zpět mezi připravené úlohy je zařazena po příchodu očekávané zprávy do schránky zpráv. Po přijetí zprávy (*obr. 6*, řádky 11 a další) přechází úloha do stavu odpovídajícího stavu *low* systému, tj. svítidlo začne svítit slabě (*obr. 6*, řádek 14). Poté úloha čeká po dobu 500 ms na příchod zprávy (*obr. 6*, řádek 15). Nepřijde-li zpráva do uplynutí této doby, úloha pokračuje vykonáváním úseku kódu určeného k obsluze vypršení časového limitu pro příchod zprávy (*obr. 6*, řádky 24 až 26), tj. čekáním na stisk tlačítka pro přechod do stavu *off* automatu (*obr. 6*, řádek 25). Přijde-li zpráva před uplynutím této doby (tj. tlačítko je stisknuto do 500 ms), úloha pokračuje prováděním kódu odpovídajícího chování automatu ve stavu *bright*, tj. zajistí silný svit svítidla a čeká na přijetí zprávy pro přechod do stavu *off* (*obr. 6*, řádky 18 až 21).

Konkrétní realizace funkcí *lamp_off()*, *lamp_low()*, a *lamp_bright()* zde není uvedena; je ponechána k zamyšlení čtenáři/čtenářce tohoto článku. Obecně se však očekává, že funkce volané v rámci těl úloh RT budou realizovány tak, že doby jejich běhů budou pokud možno konstantní a co nejkratší. Jinými slovy, očekává se, že úloha nebude konzumovat čas procesoru déle než po dobu nezbytně nutnou k vykonání požadované funkce. Jelikož obvyklou cílovou platformou pro realizaci systémů RT jsou mikrořadiče, je pro realizaci funkcí *lamp_off()*, *lamp_low()*, a *lamp_bright()* výhodné využít periferie, které jsou v mikrořadičích běžně dostupné – např. časovače. Ty jsou schopny, paralelně s činností procesoru, řídit intenzitu svitu svítidla např. s použitím pulzní šířkové modulace. Důležité je, že řízením intenzity svitu svítidla se pak procesor, kromě potřebného nastavení časovače formou přiřazovacího příkazu zajišťujícího změnu střídy signálu, nemusí aktivně zabývat.

```
01 static void LampTask (void * p_arg)
02 {
03   INT8U err;
04   void *msg;
05
06   for(;;)
07   {
08     /* --- OFF --- */
09     lamp_off();
10     msg = OSMboxPend(msg_press, 0, &err);
11     if (msg != (void *)0)
12     {
13       /* --- LOW --- */
14       lamp_low();
15       msg = OSMboxPend(msg_press, 500, &err);
16       if (msg != (void *)0)
17       {
18         /* --- BRIGHT --- */
19         lamp_bright();
20         msg = OSMboxPend(msg_press, 0, &err);
21         if (msg != (void *)0);
22       } else
23       {
24         /* --- OFF --- */
25         msg = OSMboxPend(msg_press, 0, &err);
26         if (msg != (void *)0);
27       }
28     }
29   }
30 }
```

Obr. 6. Tělo úlohy LampTask

Závěrem

V článku byl na jednoduchém příkladu nastíněn vztah mezi třemi významnými etapami vývojového cyklu systému RT – etapou specifikace a verifikace, etapou modelování a simulace a etapou realizace systému RT. Cílem bylo jednak upozornit na typická úskalí, kterými je třeba se při přechodech mezi jednotlivými etapami zabývat, a jednak shrnout a o další souvislosti doplnit informace odděleně představené v předchozích článcích seriálu na téma navrhování časově kritických systémů. Následující odstavce celý seriál stručně shrnují.

První článek seriálu je věnován úvodu do problematiky návrhu časově kritických systémů, klasifikaci mezí odezev úloh a systémů RT a také základním pojmům a principům souvisejícím se specifikací a verifikací systémů RT. Při tvorbě článku byla upřednostněna ilustrativnost před teoreticky vyčerpávajícím obsahem, což platí o všech článcích seriálu.

Ve druhém článku jsou představeny základní pojmy související s modelem úloh RT a plánováním množin úloh RT. Je zdůrazněna klíčová role mechanismu přiřazování priorit, realizovaného v rámci plánovače, na chod systému RT.

Třetí článek je věnován základním mechanismům přiřazování priorit úlohám RT. Jsou představeny dva mechanismy statického (RM, DM) a dva mechanismy dynamického (EDF, LLF) přiřazování priorit, shrnuty jejich hlavní vlastnosti a naznačeny hlavní přednosti a nedostatky spojené s jejich použitím. Je ilustrováno, že mechanismy garantující plánovatelnost větší množiny úloh RT vedou k výpočetně náročnější konstrukci plánovačů. Jelikož volba vhodného mechanismu přiřazování priorit patří k nejdůležitějším úkolům návrháře systému RT, je nutné ji pečlivě zvážit. Z tohoto pohledu je nezbytné, aby návrháři časově kritických systémů měli dostatečný přehled v oblasti mechanismů přiřazování priorit – není možné spokojit se pouze se znalostí mechanismů představených v tomto seriálu, ale je nutné studovat další mechanismy použitelné např. pro plánování závislých úloh, pro plánování při přetížení systému, pro plánování v síťovém či víceprocesorovém prostředí apod.

Závěrečný článek seriálu byl shrnující, s cílem zdůraznit vztah mezi tématy představenými v předchozích statích a naznačit základní problémy spojené s realizací systému RT. V seriálu bylo záměrně odkazováno pouze na produkty volně dostupné široké veřejnosti k nekomerčnímu využití (Uppaal, TimesTool, Cheddar, µC/OS-II) s tím záměrem, aby po přečtení jednotlivých článků seriálu měl každý možnost si vhodné produkty zdarma nainstalovat a vyzkoušet si s jejich použitím jednotlivé etapy návrhu systémů RT.

Poděkování

Článek vznikl za podpory výzkumného záměru MSM0021630528 *Výzkum informačních technologií z hlediska bezpečnosti* (agentura CEZ MŠMT) a projektu specifického výzkumu FIT-S-10-1 (VUT v Brně).

Literatura:

[1] BEHRMANN, G. – DAVID, A. – LARSEN, K. G.: *A Tutorial on Uppaal*. In: Formal Methods for the Design of Real-Time Systems, 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems. Springer-Verlag, 2004, pp. 200–236, LNCS 3185.

[2] LABROSSE, J. J.: *MicroC/OS-II – The Real-Time Kernel*. CMP Books, 2002, 648 s., ISBN 978-1-57820-103-7.

[3] SROVNAL, V.: *Operační systémy pro řízení v reálném čase*. VŠB TU, Ostrava, 2003, 218 s., ISBN 80-248-0503-0.

[4] *Micrium* [online]. 2010 [cit. 2010-03-26]. Dokument dostupný z <http://www.micrium.com>.

[5] STRNADEL, J.: *Návrh časově kritických systémů I: specifikace a verifikace*. Automa, 2010, roč. 16, č. 10, s. 42–44, ISSN 1210-9592.

[6] STRNADEL, J.: *Návrh časově kritických systémů II: úlohy reálného času*. Automa, 2010, roč. 16, č. 12, s. 18–19, ISSN 1210-9592.

[7] STRNADEL, J.: *Návrh časově kritických systémů III: priorita úloh*. Automa, 2011, roč. 17, č. 2, s. 50–52, ISSN 1210-9592.

Ing. Josef Strnadel, Ph.D.,
Fakulta informačních technologií,
Vysoké učení technické v Brně
(strnadel@fit.vutbr.cz)

Ing. Josef Strnadel, Ph.D., je odborným asistentem na Fakultě informačních technologií Vysokého učení technického v Brně. Své vědeckovýzkumné i pedagogické aktivity dlouhodobě směruje do oblastí vestavných systémů, systémů pracujících v reálném čase a testování číslicových systémů.

V Norimberku se konal veletrh Embedded World 2011

V Norimberku (SRN) se 1. až 3. března 2011 konal veletrh s konferencí Embeded World 2011. Letošní účast vystavovatelů prolomila hranici 800 a ve srovnání s loňským rokem vzrostla o 10 %. Třináct procent vystavovatelů bylo letos na veletrhu poprvé. Vzrostl také podíl vystavovatelů ze zahraničí, a to na letošních 43 %. Veletrh, spojený s odbornou konferencí, je tak nejvýznamnější evropskou událostí v oboru vestavných systémů v Evropě.

Klíčová témata letošního veletrhu byla dvě. První téma bylo optimalizace energetické spotřeby vestavných systémů, nutná nejen z hlediska ohledu k životnímu prostředí (zde nejde o zařízení s příkonem v kilowattech, tudíž úspora není tak významná jako např. u elektrických pohonů), ale zejména pro lepší využitelnost a větší komfort u mobilních zařízení. Druhé téma bylo komunikace, a to ze dvou hledisek: zaprvé komunikace mezi jednotlivými zařízeními (M2M; společný stánek dvacítky vystavovatelů byl v hale 12) a zadruhé komunikace jako základ pro tzv. *cloud computing* v oblasti vestavných systémů.

Obr. 1. Návštěvníci veletrhu Embedded World 2011

Součástí veletrhu byla také specializovaná výstava displejů a komponent pro zobrazování.

Embedded Awards

První den veletrhu byly uděleny ceny Embedded Awards. V kategorii softwaru ji získala společnost Hitex za bezpečnostní balíček pro TriCore. Balíček umožňuje snadno a bez zásahů lidské obsluhy, která by mohla být zdrojem náhodných nebo i úmyslných chyb, plně certifikovat bezpečnostně kritické úlohy využívající mikroprocesory Infineon. V kategorii hardwaru získala cenu firma Freescale za MPC5646CMCU, mikroprocesor určený pro hardwarové šifrování dat v oblasti automobilové techniky. Speciální cenu získal Fraunhoferův ústav pro integrované obvody za miniaturní kameru s rozlišením HD a s integrovaným převodníkem DVB-T. Kamera je příkladem vysoce integrovaného vestavného systému.