„Kuchařka“ pro vytvoření FreeRTOS projektu pro Nucleo STM32F103RB – Keil 5.21a

---

1. Nový projekt do nového adresáře
    a. Funkční základní blikání LED s „for“ čekáním, ideálně RGB na MBED Shieldu
2. Nakopírovat do adresáře projektů „**knihovnu FreeRTOS**“, stačí pouze Source z adresáře FreeRTOS ze staženého ZIP.
    a. tj. strom adresářů je

```
FreeRTOS_vXXXX
        Source
                Portable
                Include
Lib
        Deska.c
        ...
Projekt_Rtos
        Projekt_Rtos.uvproj
        Main.c
Projekt_2 (další projekty)
        Projekt_2.uvproj
        Main.c
```

3. Nastavení projektu:
    a. C/C++ - přidat „Include Paths“ na FreeRTOS, takže výsledek bude

```
.
..\Lib
..\FreeRTOS_9.0.0\Source\include
..\FreeRTOS_9.0.0\Source\portable\RVDS\ARM_CM3
```

4. V „Target“ přidat nový „Source Group“ se jménem např. FreeRTOS a přidat existující soubory
    a. z adresáře FreeRTOS\Source (pokus se některá funkcionalita nebude v projektu využívat, nemusí se přidávat/překládat, zde připojíme všechny):

```
tasks.c
queue.c
timers.c
list.c
event_groups.c
croutine.c
```

    b. nutno zvolit vhodnou strategii správy hromady, vybrat jeden – pro naši HW konfiguraci se hodí

```
FreeRTOS_9.0.0\Source\portable\MemMang\heap_2.c
```

    c. z adresáře FreeRTOS\Source\portable\RVDS\ARM_CM3

```
port.c
```

5. Konfigurační soubor FreeRTOSConfig musí být v pracovním adresáři a určuje konfiguraci FreeRTOSu. Je možné si základ „půjčit“ z demo-příkladů v ZIP archívu – nejbližší našemu HW je zřejmě **FreeRTOS_9.0.0\FreeRTOS\Demo\CORTEX_STM32F103_Keil\FreeRTOSConfig.h**, nebo vytvořit prázdný soubor a doplnit příslušné volby
    a. Pokud některá volba (#define) není nastavena, knihovny FreeRTOS si defaultní hodnotu doplní „samy“

   b. Seznam voleb viz. Dokumentace on-line
6. Přidat do **main1.c** potřebný **#include "FreeRTOS.h"**
7. Překlad by nyní měl proběhnout bez chyb, brány s RGB LED jsou stále funkční

8. Pro RTOS aplikaci je nutné připojit pomocí #include hlavičkové soubory RTOS funkcí:

```
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "semphr.h"
```

9. Nastavit prioritu pro tasky – nejlépe vyjít ze stávajících/defaultních priorit:

```
#define mainLED_TASK_PRIORITY     ( tskIDLE_PRIORITY  +  1 )
```

10. Připravit funkci obsluhy jednotlivých tasků – zde bude jedna společná pro LED a podle předaného parametru při vytváření tasku bude prováděna akce na příslušné LED a zároveň podle toho určen čas další „neaktivity“ tasku:

```
void LEDFlashTask(void *params)
{
  char c = ((char *)params)[0];
  portTickType lastWakeTime = xTaskGetTickCount();

  while(1)
  {
    switch(c)
    {
      case 'R': TOGGLE_LED_R; break;
      case 'G': TOGGLE_LED_G; break;
      case 'B': TOGGLE_LED_B; break;
    }

//    vTaskDelay(10 * c);
    vTaskDelayUntil(&lastWakeTime, 10 * c);
  }
}
```

11. Ve funkci main je třeba vytvořit jednotlivé tasky a spustit scheduler, který již potom celý proces řídí sám – do další části kódu se to již nedostane (jedině pokud by všechny tasky skončily):

```
xTaskCreate(LEDFlashTask, "LED_R", configMINIMAL_STACK_SIZE, "R", mainLED_TASK_PRIORITY, NULL);
xTaskCreate(LEDFlashTask, "LED_G", configMINIMAL_STACK_SIZE, "G", mainLED_TASK_PRIORITY, NULL);
xTaskCreate(LEDFlashTask, "LED_B", configMINIMAL_STACK_SIZE, "B", mainLED_TASK_PRIORITY, NULL);
vTaskStartScheduler();
```

12. Hotová aplikace se dá přeložit, ale „nic nedělá“. Při nahlédnutí do FAQ sekce webu FreeRTOS.org (http://www.freertos.org/FAQHelp.html) se lze dočíst v části „Special note to ARM Cortex-M users“, že do config souboru je nutno doplnit následující řádky:

```
#define vPortSVCHandler SVC_Handler
#define xPortPendSVHandler PendSV_Handler
#define xPortSysTickHandler SysTick_Handler
```

13. Základní funkčnost by nyní měla být v pořádku.

14. Zjištění provozních informací = využití jednotlivých procesů - využití funkce vTaskGetRunTimeStats, která naplní textový buffer výpisem. Ideálně ve formě vlastního procesu (spouštěného pomocí xTaskCreate):

```
void DEBUGTask(void *params)
{
  char debugBuffer[256];
  portTickType lastWakeTime = xTaskGetTickCount();

  vTaskDelayUntil(&lastWakeTime, 5000);  // oddal první spusteni o 5s

  while(1)
  {
    //  http://www.freertos.org/a00021.html#vTaskGetRunTimeStats
    vTaskGetRunTimeStats(debugBuffer);

    //TODO: odeslat přes UART nebo zobrazit na LCD

    vTaskDelayUntil(&lastWakeTime, 1000);
  }
}
```

15. Dále je nutno spustit „nějaký“ čítač, který bude připočítávat globální proměnnou, ve které si OS drží dobu trvání určitého procesu:
    a. Použít např. TIM3
    b. Nastavit periodu 50us:

```
TIM3->PSC = SystemCoreClock/1000000 - 1; // Prescale to 1 us timer
TIM3->ARR = 49;                          // Autoreload (N-1) * 1us
```

    c. Povolit přerušení a mít jednoduchou obsluhu:

```
void TIM3_IRQHandler(void)
{
  TIM3->SR &= ~TIM_SR_UIF;     // Clear update event interrupt flag
  ulRunTimeStatsClock++;
}
```

16. Upravit v FreeRTOSConfig.h potřebné parametry:
    a. Změnit v řádku (cca 90) na:

```
#define configUSE_TRACE_FACILITY      1
```

    b. Přidat:

```
// vseobecne nastaveni
#define configGENERATE_RUN_TIME_STATS 1
#define configUSE_STATS_FORMATTING_FUNCTIONS 1

// HW zavisle nastaveni
extern unsigned long ulRunTimeStatsClock;
#define portCONFIGURE_TIMER_FOR_RUN_TIME_STATS() ulRunTimeStatsClock = 0
#define portGET_RUN_TIME_COUNTER_VALUE() ulRunTimeStatsClock
```