Obsah

Kapitola 5

Časovače

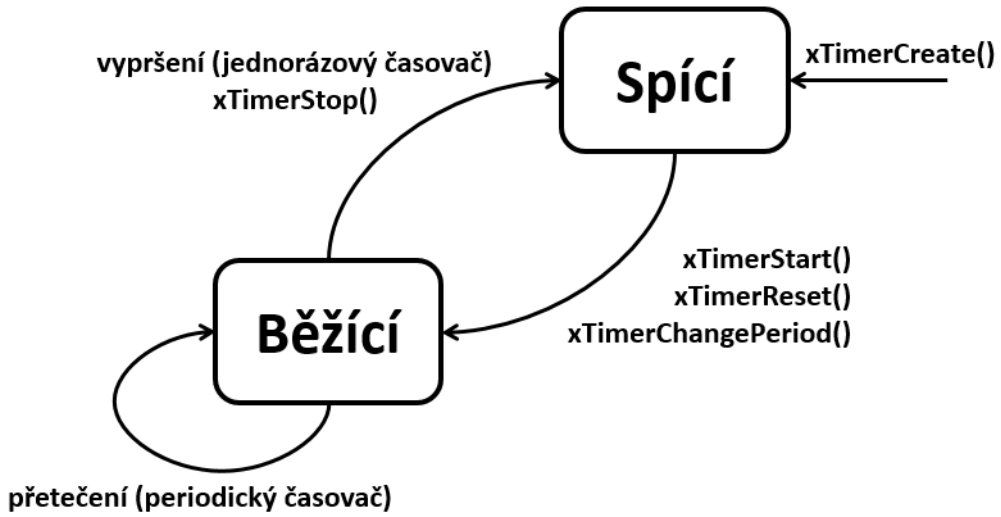
Kapitola o časovačích je v této knize zařazena brzy, jelikož díky jejich funkci a nástrahám čtenář získá hlubší vhled do problematiky časových úseků a plánovacího algoritmu, které se v budoucích kapitolách již tolik věnovat nebudeme.

5.1 Funkce

Časovač umožňuje provedení funkce po uplynutí přesně daného času (jednorázový časovač), nebo opakované provedení funkce s přesně danou periodou (periodický časovač). Časovač je implementován softwarově. Není tedy nijak závislý na hardwarových časovačích.

Časovač pracuje na mechanismu volání ***callback*** funkce v přesně daný okamžik. Oproti funkcím vláken tato funkce nesmí obsahovat nekonečnou smyčku a nesmí se přepnout do blokovaného stavu. (Funkce může obsahovat knihovní funkce operačního systému, které ***callback*** funkci mohou zablokovat, ale parametr času zablokování musí mít nulovou hodnu.) Funkce by měla být co možná nejkratší.

Časovač se může nacházet v jednom ze dvou stavů – spící, běžící. Přechod mezi stavy může nastat při volání různých knihovních funkcí. Přechody mezi stavy je možné vidět v následujícím obrázku:



Všimněme si, že nejenom funkce určená k zapnutí časovače ***xTimerStart()*** zapne časovač.

Ze stavového diagramu je vidět rozdíl mezi časovačem jednorázovým a periodickým. Jednorázový se po vypršení času (první přetečení) přepne z běžícího do spícího stavu. Periodický časovač se přepne do spícího stavu pouze použitím příkazu, jinak běží opakovaně.

Při přetečení či vypršení časovače dojde k volání ***callback*** funkce, kterou může programátor libovolně upravovat.

Veškerá manipulace s časovači – zapínání, vypínaní, resetování, nastavení, ***callback*** funkce – se děje ve speciálně určeném vlákně, které je vytvořeno a spuštěno automaticky. Toto stínové vlákno (***Daemon task***) přijímá příkazy spojené s časovači (zapnutí, vypnutí . . .). Příkazy se ukládají do fronty příkazů (funkce fronty bude popsána v budoucí kapitole). Stínové vlákno provede všechny příkazy shromážděné ve frontě až v momentě, kdy se aktivuje. Pokud má tedy toto vlákno příliš malou prioritu, nebo nenastane chvíle, kdy jsou všechna uživatelská vlákna v blokovaném stavu, může se stát, že se příkaz nikdy neprovede a ***callback*** funkce se nikdy nespustí. Stínové vlákno je v blokovaném stavu, dokud některé z uživatelských vláken nevloží příkaz do fronty příkazů, nebo nenastane událost spojená s časovačem. Tento mechanismus a jeho nástrahy jsou dobře vidět v praktických příkladech.

Než budeme moci použít časovače v programu, musíme je povolit definováním konstanty ***configUSE_TIMERS*** s hodnotou 1 v konfiguračním souboru ***FreeRTOSConfig.h***.

5.1.1 Příklad 9 – periodický a jednorázový časovač

Tento příklad ukáže, jakým způsobem spustit, zastavit či resetovat časovače. Dále uvidíme, jak nevyzpytatelně se časovače mohou chovat, když je s nimi špatně zacházeno.

Program bude každou sekundu vypisovat znak "1", každé dvě sekundy znak "2"a jednorázově 4 sekundy po startu programu vypíše znak "3". Odesláním jakéhokoliv znaku z PC do kontroleru se časovače resetují a znovu spustí. To znamená, že pokud mikrokontroler bude přijímat znaky z PC častěji než jednou za sekundu, neodešle žádná data. Výjimku bude tvořit přijatý znak "s", který všechny časovače zastaví.

V našem programu si nejprve vytvoříme tři ukazatele na časovače, které budeme v jednom z vláken využívat na práci s nimi.

Dále vytvoříme ***callback*** funkce pro jednorázový a periodický časovač. Periodické časovače budou sdílet jednu ***callback*** funkci.

```
TimerHandle_t periodickyCasovac1 = NULL;
TimerHandle_t periodickyCasovac2 = NULL;
TimerHandle_t jednorazovyCasovac = NULL;

static void JednorazovyCallback(TimerHandle_t Casovac)
{
        HAL_UART_Transmit(&huart2, "3", 1, 10);
}

static void PeriodickyCallback(TimerHandle_t Casovac)
{
        uint8_t id = pvTimerGetTimerID(Casovac);

        if(id == 1)
        {
                HAL_UART_Transmit(&huart2, "1", 1, 10);
        }
        else
        {
                HAL_UART_Transmit(&huart2, "2", 1, 10);
        }
}
```

V callback funkci periodického časovače odesíláme rozdílný znak na základě toho, jakým časovačem byla funkce volána. Rozlišení bychom mohli provést buď porovnáním adresy ukazatele, nebo jako v našem případě pomocí identifikačního čísla časovače. K získání identifikátoru využijeme funkci

void* pvTimerGetTimerID(adresaCasovace), kde

- ***adresaCasovace*** je adresa na časovače, jehož identifikátor chceme získat.

Funkce vrací ukazatel na void, který můžeme použít na jakýkoliv číselný datový typ.

Poté implementujeme uživatelské vlákno, které bude s časovači pracovat. Vlákno při svém vytvoření všechny časovače spustí a nastaví komunikaci ***UART*** na příjem jednoho znaku. Vlákno v nekonečné smyčce zastaví časovače v momentě, kdy kontroler přijme znak "s". Při příjmu jakéhokoliv jiného znaku časovače resetuje (resetování časovače zároveň opět spustí).

```
static void VlaknoCasovace( void *pvParameters )
{
        uint8_t data = 0U;

        xTimerStart(periodickyCasovac1, pdMS_TO_TICKS(1));
        xTimerStart(periodickyCasovac2, pdMS_TO_TICKS(1));
        xTimerStart(jednorazovyCasovac, pdMS_TO_TICKS(1));

        HAL_UART_Receive_IT(&huart2, &data, 1);

        while(1)
        {
                // prisel znak na zastaveni casovacu
                if(data == 's')
                {
                        xTimerStop(periodickyCasovac1, pdMS_TO_TICKS(1));
                        xTimerStop(periodickyCasovac2, pdMS_TO_TICKS(1));
                        xTimerStop(jednorazovyCasovac, pdMS_TO_TICKS(1));

                        HAL_UART_Receive_IT(&huart2, &data, 1);
                        data = 0U;
                }
                // prisla jakakoliv jina data
                else if(data != 0U)
                {
                        // resetuje periodicke casovace
                        xTimerReset(periodickyCasovac1, pdMS_TO_TICKS(1));
                        xTimerReset(periodickyCasovac2, pdMS_TO_TICKS(1));

                        // resetuje jednorazovy casovac
                        xTimerReset(periodickyCasovac2, pdMS_TO_TICKS(1));

                        HAL_UART_Receive_IT(&huart2, &data, 1);
                        data = 0U;
                }
        }
}
```

Vlákno používá funkce na spuštění, zastavení a resetování časovače. Všechny mají stejné vstupní parametry, proto popíšeme pouze funkci na spuštění:

BaseType_t xTimerStart(adresaCasovace, pocetCasovychUseku), kde

- ***adresaCasovace*** je adresa časovače, který chceme spustit, a
- ***pocetCasovychUseku*** je maximální počet časových úseků, po které bude funkci volající vlákno zablokováno, než se zařadí příkaz ke spuštění časovače do fronty příkazů pro stínové vlákno. Poté bude vlákno odblokováno.

Návratová hodnota má hodnoty ***pdPASS*** nebo ***pdFAIL*** podle toho, zda se v daném maximálním počtu časových úseků povedlo zařadit příkaz do fronty příkazů časovačů.

Nyní v hlavním programu vytvoříme tři časovače a jedno vlákno.

```
/* USER CODE BEGIN RTOS_THREADS */
  periodickyCasovac1 = xTimerCreate("Periodicky1", pdMS_TO_TICKS(1000), pdTRUE, 1,
                                    PeriodickyCallback);

  periodickyCasovac2 = xTimerCreate("Periodicky2", pdMS_TO_TICKS(2000), pdTRUE, 2,
                                    PeriodickyCallback);

  jednorazovyCasovac = xTimerCreate("Jednorazovy", pdMS_TO_TICKS(4000), pdFALSE, 1,
                                    JednorazovyCallback);

  xTaskCreate(VlaknoCasovace, "VlaknoCasovace", 128, NULL, 1, NULL);

  /* USER CODE END RTOS_THREADS */

  /* Start scheduler */
  osKernelStart();
```

K vytvoření časovače využíváme funkci

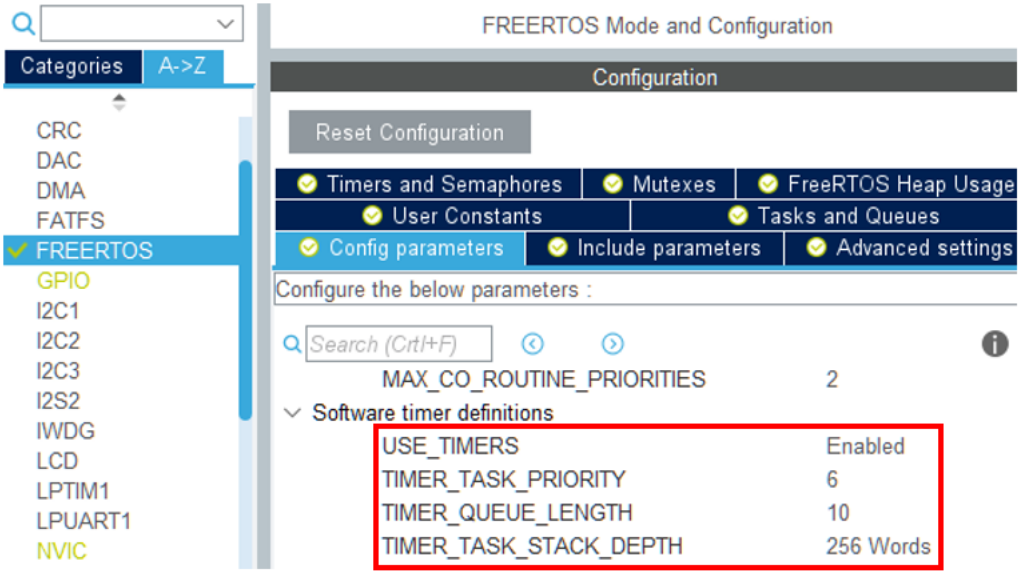
TimerHandle_t xTimerCreate(uzivatelkeJmeno, perioda, autoReload, identifikator, adresaCallbackFunkce), kde

- ***uzivatelskeJmeno*** je ukazatel na řetězec s uživatelským jménem,
- ***perioda*** udává počet časových úseků, než dojde k přetečení časovače,
- ***autoReload*** určuje, zda bude časovač periodický (vložením hodnoty ***pdTRUE***) či jednorázový (vložením hodnoty ***pdFALSE***),
- ***identifikator*** je identifikátor časovače a
- ***adresaCallbackFunkce*** je adresa callback funkce, která se zavolá při přetečení časovače.

Funkce vrací adresu na vytvořený časovač. Pokud je návratová hodnota ***NULL***, není pro nový časovač místo v paměti.

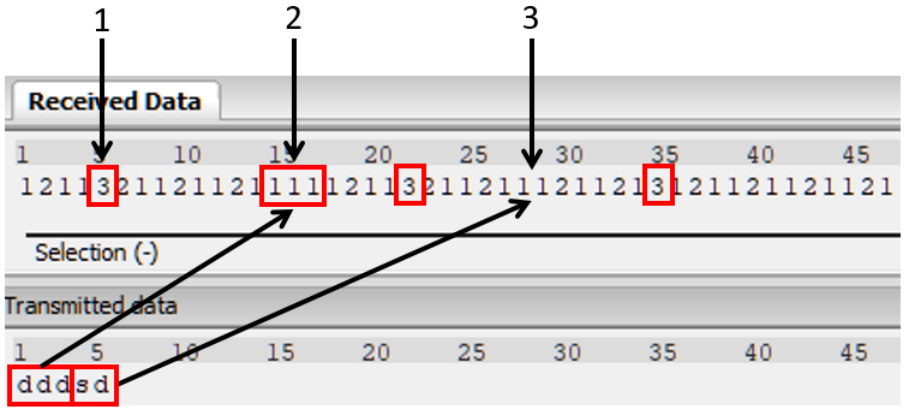
Periodické časovače našeho programu mají periodu 1000 a 2000 ms a mají rozdílné identifikátory. Jednorázový časovač má periodu 4000 ms. Časovačům jsme posledním argumentem určili, jakou ***callback*** funkci mají používat.

Předtím, než program přeložíme, musíme v konfigurátoru povolit používání časovačů, což provedeme následující změnou:



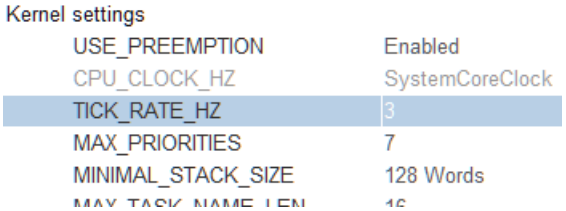
Je doporučené, aby vlákna časovačů měla nejvyšší možnou prioritu (***configTIMER_TASK_PRIORITY***). Později si ukážeme důvod, proč je toto nastavení žádoucí. Parametr ***configTIMER_QUEUE_LENGTH*** určuje velikost fronty příkazů.

Textový výstup může vypadat následovně:



Sekundu po spuštění programu se odešle první znak "1", po druhé sekundě dva znaky "21", jelikož přetekly dva časovače zároveň. Po třetí sekundě opět pouze "1". V bodě **(1)**, který odpovídá čtvrté sekundě běhu programu, přeteče jednorázový časovač. Vypisuje se tedy text "321". To se po dalších 4 sekundách již neopakuje. Po čtvrté sekundě v bodě **(2)** odešleme z PC postupně se sekundovým odstupem třikrát libovolný znak (v tomto případě třikrát "d"). To způsobí třikrát reset všech časovačů. Jelikož je odstup znaků přibližně jedna sekunda, pouze časovač s kratší periodou stíhá přetéct a vypisovat znaky "1". Čtyři sekundy po posledním resetu jednorázový časovač opět pošle znak "3". V bodě **(3)** PC odešle znak "s". Od tohoto momentu časovače nefungují a příjem znaků se zastaví. Po nějaké chvíli kontroler přijme libovolný znak, což všechny časovače resetuje a opět spustí. Tento další reset způsobil tři znaky "1"v řadě. Po 4 sekundách se opět ozve jednorázový časovač. V této ukázce však již naposledy.

Změňme nyní délku časového úseku. Místo jedné milisekundy (1 KHz) bude trvat 333 ms (3 Hz). (Podobnou změnu jsme dělali v příkladu 1.)



Dále přidejme do ***callback*** funkce přepínání LED. Konkrétněji do části, která se provede při přetečení periodického časovače 1.

```
static void PeriodickyCallback(TimerHandle_t Casovac)
{
        uint8_t id = pvTimerGetTimerID(Casovac);

        if(id == 1)
        {
                HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);

                HAL_UART_Transmit(&huart2, "1", 1, 10);
        }
        else
        {
                HAL_UART_Transmit(&huart2, "2", 1, 10);
        }
}
```

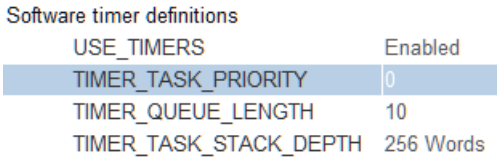
Poslední úpravou je změna periody periodického časovače z 1000 na 1500 ms.

```
  periodickyCasovac1 = xTimerCreate("Periodicky1", pdMS_TO_TICKS(1500), pdTRUE, 1,
                                    PeriodickyCallback);
```

Po spuštění získáme následující výstupní signál:

Očekávali jsme přepínání úrovně signálu každých 1500 ms, ale jak na obrázku vidíme, i časovače jsou zatížené rozlišovací schopností. Ta je závislá na délce časového úseku. Jelikož 1500 a 333 ms nejsou soudělné hodnoty, pomocí volaného makra ***pdMS_TO_TICKS()*** se čas převede na 4 časové úseky, které tvoří přibližně 1333 ms. Toto je potřeba při programování zohlednit.

Nyní pojďme upravit prioritu stínového vlákna. Změníme ji na úroveň 0, tedy nižší než má uživatelské vlákno.

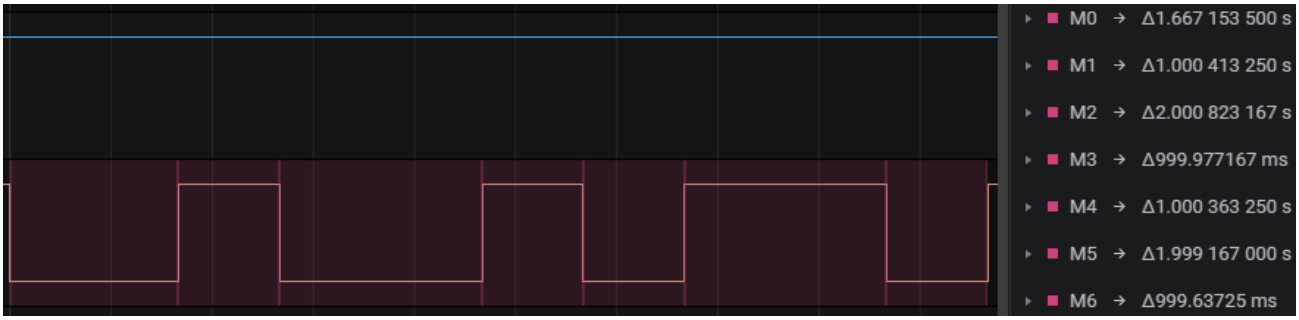


Když program s takovýmto nastavením spustíme, stínové vlákno se nikdy nedostane ke slovu a nespustí se žádná ***callback*** funkce časovačů. Lékem je zablokování uživatelského vlákna v jeho nekonečné smyčce alespoň do konce časového úseku. Má to ale háček. Vlákno s vyšší prioritou může svým během zabrat více časových úseků, než se zablokuje. Upravme tedy uživatelské vlákno tak, že na konci iterace smyčky dva úseky pracuje (zpožďovací smyčka) a poté se na jeden úsek zablokuje:

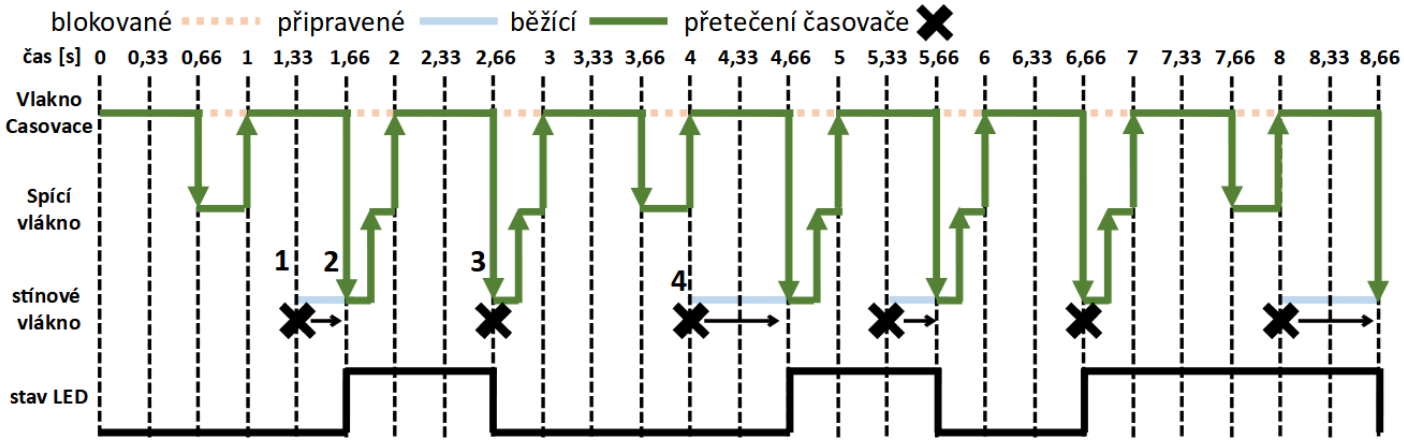
```
static void VlaknoCasovace( void *pvParameters )
{
        uint8_t data = 0U;

        xTimerStart(periodickyCasovac1, pdMS_TO_TICKS(1));
        xTimerStart(periodickyCasovac2, pdMS_TO_TICKS(1));
        xTimerStart(jednorazovyCasovac, pdMS_TO_TICKS(1));

        HAL_UART_Receive_IT(&huart2, &data, 1);

        while(1)
        {
                // prisel znak na zastaveni casovacu
                if(data == 's')
                {
                        xTimerStop(periodickyCasovac1, pdMS_TO_TICKS(1));
                        xTimerStop(periodickyCasovac2, pdMS_TO_TICKS(1));
                        xTimerStop(jednorazovyCasovac, pdMS_TO_TICKS(1));

                        HAL_UART_Receive_IT(&huart2, &data, 1);
                        data = 0U;
                }
                // prisla jakakoliv jina data
                else if(data != 0U)
                {
                        // resetuje periodicke casovace
                        xTimerReset(periodickyCasovac1, pdMS_TO_TICKS(1));
                        xTimerReset(periodickyCasovac2, pdMS_TO_TICKS(1));

                        // resetuje jednorazovy casovac
                        xTimerReset(periodickyCasovac2, pdMS_TO_TICKS(1));

                        HAL_UART_Receive_IT(&huart2, &data, 1);
                        data = 0U;
                }

                // zpozdeni 2 casove useky
                HAL_Delay(1);

                // zablokuje vlakno do konce useku
                vTaskDelay(1);
```

```
        }
}
```

Výstupní signál vypadá takto: (časovým úsekům zleva doprava odpovídají časové hodnoty shora dolů)



Výstupní signál je velmi neperiodický a chová se podivně. Jeho chování vysvětlí následující graf průběhů:



Program běží dle očekávání až do bodu **(1)**, kdy dojde k přetečení časovače (přetéká vždy po 1333 ms). Ve vláknu přechodu úseků se odešle příkaz do fronty k obsloužení časovače ve stínovém vláknu (označeno symbolem **X**). Bohužel je ale nyní aktivní vlákno s vyšší prioritou, a proto se stínové vlákno dostane ke slovu o jeden časový úsek později v bodě **(2)**. Teprve v ten moment se zavolá ***callback*** funkce a přepne se stav LED. V bodě **(3)** shodou okolností dojde k přetečení časovače a zablokování vlákna *VlasknoCasovace* ve stejný okamžik. Obsluha časovače se tedy provede okamžitě. V bodě **(4)** dojde k přetečení a konci blokace uživatelského vlákna ve stejný moment, plánovací algoritmus tedy aktivuje uživatelské vlákno. Stínové vlákno s obsluhou časovače nyní musí čekat na běh další dva časové úseky. Průběh ve stejném duchu pokračuje dál.

Mějme na vědomí, že zpoždění se netýká pouze volání ***callback*** funkcí, které jako jediné v tomto příkladu používáme. Stejný problém nastane i u veškerých knihovních funkcí, které pracují s časovači (zapnutí, vypnutí, reset...).

Udělejme poslední změnu. Uživatelské vlákno nyní bude v aktivním stavu delší dobu, než je perioda časovače. Například 8 časových cyklů. Jak bude vypadat průběh signálu?