RTX přepínací strategie (Scheduling Options)

- **Pre-emptive scheduling**

Každý task má **různou prioritu** a běží až ho přeruší task s vyšší prioritou nebo zablokuje volání funkce OS.

- **Round-Robin scheduling**

Každý task má **stejnou prioritu** a běží fixní periodu nebo ho zablokuje volaní funkce OS.

- **Co-operative multi-tasking**

Každý task má **stejnou prioritu** a Round-Robin je zakázán. Task zablokuje volaní OS nebo funkce **os_tsk_pass()**.

Cooperative Multitasking

Jestliže zakážete strategii Round – Robin, musíte vytvořit tasky které **kooperují**. Zvláště musíte volat funkce OS čekání **os_dly_wait()** nebo **os_tsk_pass()** funkci v každém tasku. Tyto funkce signalizují jádru RTX přepnutí kontextu .

Příklad pro kooperující strategií.

Dva tasky jsou nekonečné smyčky. RTX startuje task 1 , jehož funkce se jmenuje **job1**. Tato funkce startuje další task zvaný **job2**.Když **job1** inkrementuje **jeden krát** , RTX přepne kontext na **job2**. když **job2** inkrementuje počítadlo 2 **jeden krát**, RTX přepne kontext na **job1**. Tyto činnosti se opakují neustále.

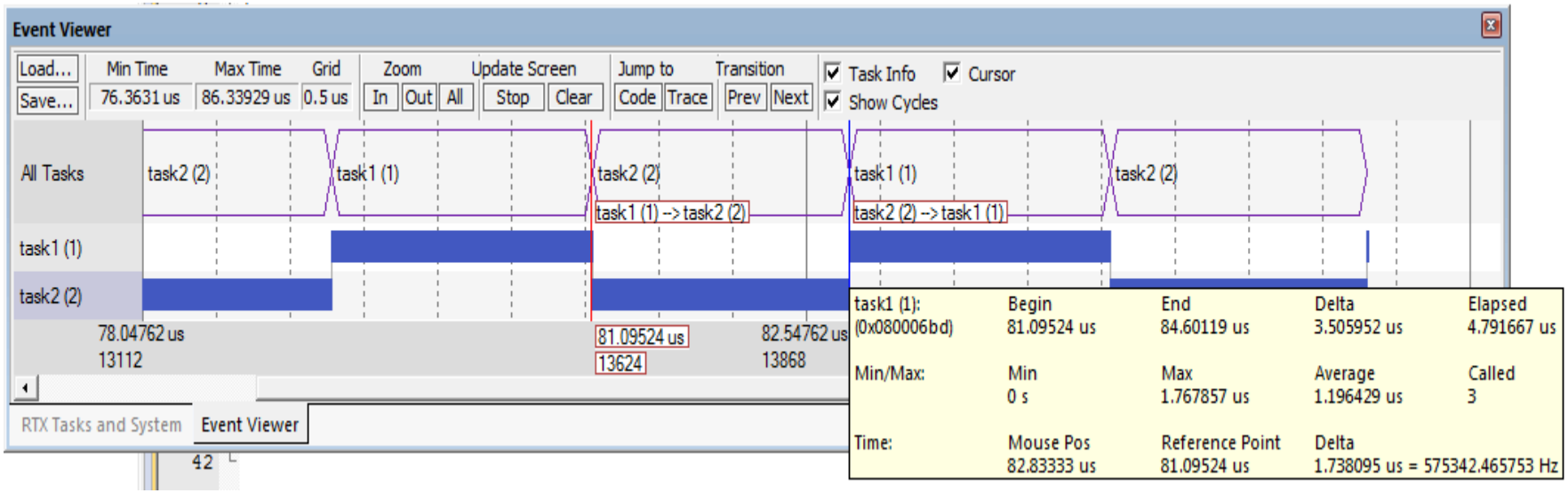
```
int counter1;
int counter2;

__task void task1 (void);
__task void task2 (void);

__task void task1 (void) {
  os_tsk_create (task2, 0);  /* vytvoří  task 2 a oznamuje  stav ready */
  for (;;) {
    counter1++;
    os_tsk_pass ();         /* přepne kontext do 'task2' */
  }
}
__task void task2 (void) {
  for (;;) {
    counter2++;
    os_tsk_pass ();         /* přepne kontext do 'task1' */
  }
}

void main (void) {
  os_sys_init(task1);       /* Inicializuje  RTX Kernel a startuje  task 1 */
  for (;;);
}
```

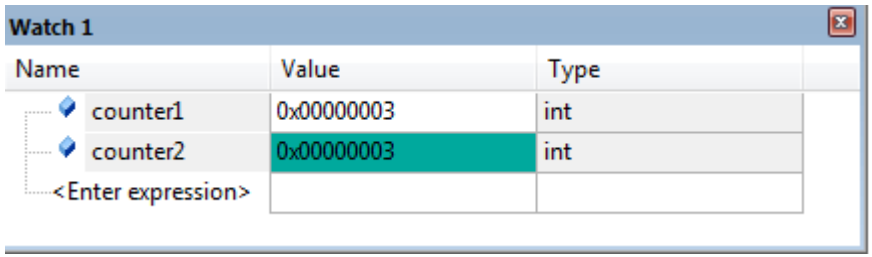
Poznámky :

Rozdíl mezi funkci **OS wait()** a **os_tsk_pass()** je ten, že **wait()** umožní čekat na událost, zatímco **os_tsk_pass()** přepne task ihned.

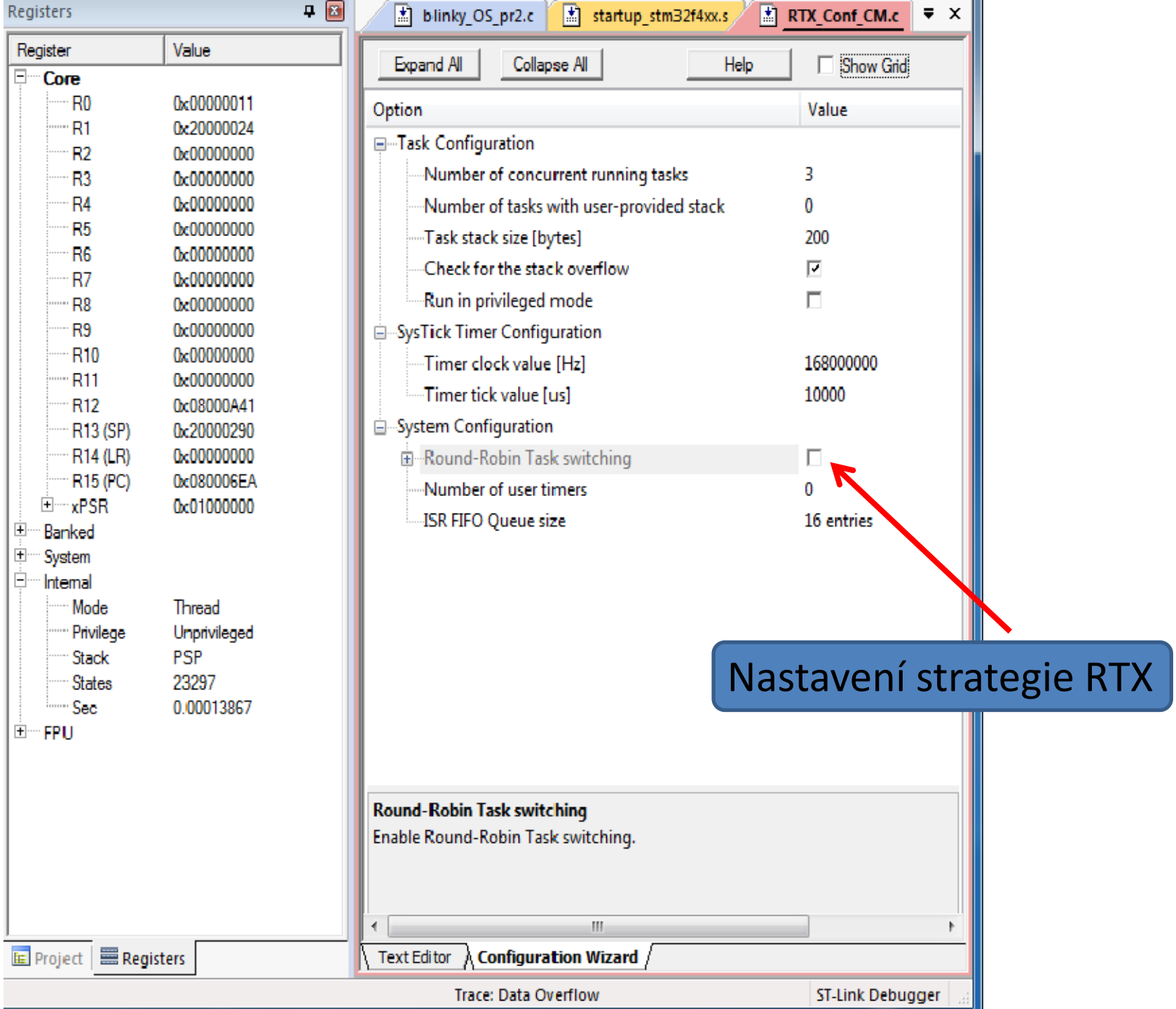
Jestliže následující **task** ve frontě připravených procesů má **nižší prioritu** než prováděný task, potom volání funkce **os_tsk_pass()** **nepřepne** kontext

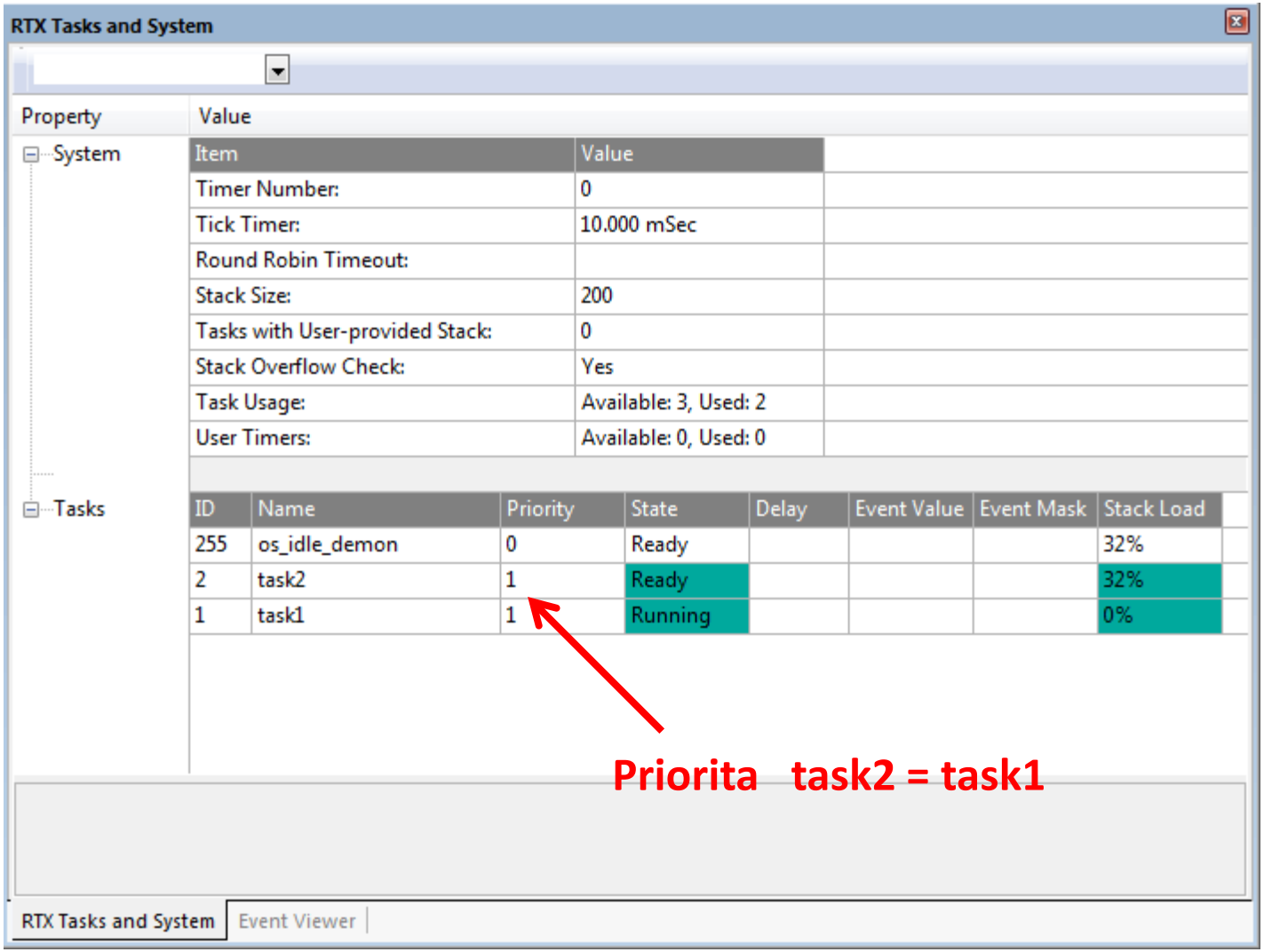
Sledování události na časové ose

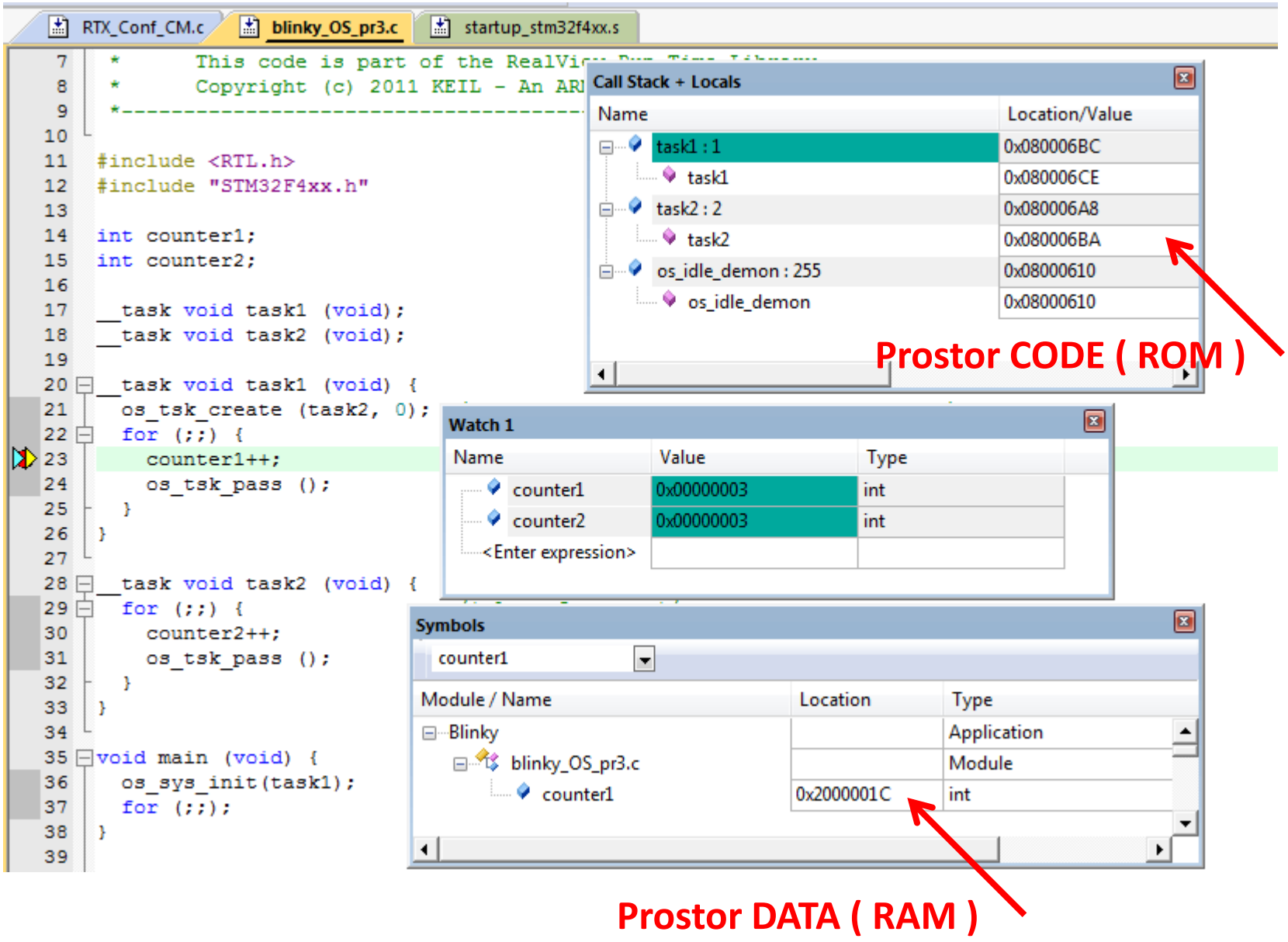




Trasování proměnných v obou procesech


Registers
Register
Value
Core
R0 0x00000011
R1 0x20000024
R2 0x00000000
R3 0x00000000
R4 0x00000000
R5 0x00000000
R6 0x00000000
R7 0x00000000
R8 0x00000000
R9 0x00000000
R10 0x00000000
R11 0x00000000
R12 0x08000A41
R13 (SP) 0x20000290
R14 (LR) 0x00000000
R15 (PC) 0x080006EA
xPSR 0x01000000
Banked
System
Internal
Mode Thread
Privilege Unprivileged
Stack PSP
States 23297
Sec 0.00013867
FPU
Project
Registers
blinky_OS_pr2.c
startup_stm32f4xx.s
RTX_Conf_CM.c
Expand All
Collapse All
Help
Show Grid
Option
Value
Task Configuration
Number of concurrent running tasks 3
Number of tasks with user-provided stack 0
Task stack size [bytes] 200
Check for the stack overflow
Run in privileged mode
SysTick Timer Configuration
Timer clock value [Hz] 168000000
Timer tick value [us] 10000
System Configuration
Round-Robin Task switching
Number of user timers 0
ISR FIFO Queue size 16 entries
Round-Robin Task switching
Enable Round-Robin Task switching.
Text Editor
Configuration Wizard
Trace: Data Overflow
ST-Link Debugger
Nastavení strategie RTX


RTX Tasks and System
Property
Value
System
Item
Value
Timer Number:
0
Tick Timer:
10.000 mSec
Round Robin Timeout:
Stack Size:
200
Tasks with User-provided Stack:
0
Stack Overflow Check:
Yes
Task Usage:
Available: 3, Used: 2
User Timers:
Available: 0, Used: 0
Tasks
ID
Name
Priority
State
Delay
Event Value
Event Mask
Stack Load
255
os_idle_demon
0
Ready
32%
2
task2
1
Ready
32%
1
task1
1
Running
0%
Priorita task2 = task1
RTX Tasks and System
Event Viewer


RTX_Conf_CM.c
blinky_OS_pr3.c
startup_stm32f4xx.s
Call Stack + Locals
Name
Location/Value
task1 : 1
0x080006BC
task1
0x080006CE
task2 : 2
0x080006A8
task2
0x080006BA
os_idle_demon : 255
0x08000610
os_idle_demon
0x08000610
Prostor CODE (ROM)
Watch 1
Name
Value
Type
counter1
0x00000003
int
counter2
0x00000003
int
<Enter expression>
Symbols
counter1
Module / Name
Location
Type
Blinky
Application
blinky_OS_pr3.c
Module
counter1
0x2000001C
int
Prostor DATA (RAM)