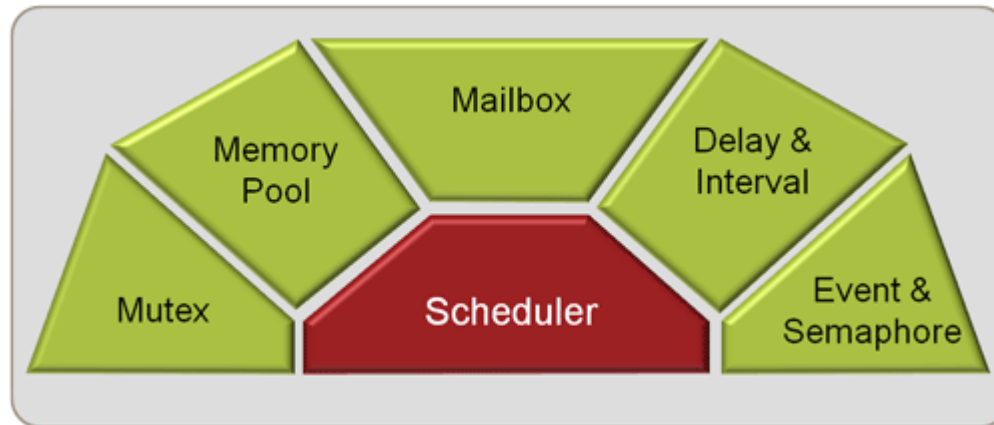


RTOS koncept část 2



Popis RTX Keil

- Startuje i končí procesy
- Obsluhuje meziprocesorovou komunikaci synchronizaci
- Spravuje přístup ke společným oblastem paměti i periferiím

meziprocesorová komunikace :

Event flags – proces má až 16 vlajek pro sledování události. Task čeká na všechny vlajky (AND-connection) nebo stačí jedna ze skupiny (OR-connection).

Semaphores – když více procesů čeká na přístup např. k společné paměti. Binární semafor obsahuje **TOKEN**. Kernel předá token prvnímu procesu, ostatní nemohou vejít do Kritické sekce. Když proces skončí vrací token semaforu a může ho vzít jiný proces.

Mutex – alternativní prostředek pro synchronizaci přístupu např. k společné paměti.

Mailboxes – jeden proces pošle zprávu druhému. Vhodné pro implementaci protokolů horních vrstev jak TCP-IP, UDP a ISDN. Zprávu prezentuje **ukazatel** (pointer) do rámce paměti, který se obsluhuje dynamicky (alokuje i uvolňuje). Kernel probudí task když dostane zprávu

Technická Data :

Description	ARM7™/ARM9™	Cortex™-M
Defined Tasks	Unlimited	Unlimited
Active Tasks	250 max	250 max
Mailboxes	Unlimited	Unlimited
Semaphores	Unlimited	Unlimited
Mutexes	Unlimited	Unlimited
Signals / Events	16 per task	16 per task
User Timers	Unlimited	Unlimited
Code Space	<4.2 Kbytes	<4.0 Kbytes
RAM Space for Kernel	300 bytes + 80 bytes User Stack	300 bytes + 128 bytes Main Stack
RAM Space for a Task	TaskStackSize + 52 bytes	TaskStackSize + 52 bytes
RAM Space for a Mailbox	MaxMessages * 4 + 16 bytes	MaxMessages * 4 + 16 bytes
RAM Space for a Semaphore	8 bytes	8 bytes
RAM Space for a Mutex	12 bytes	12 bytes
RAM Space for a User Timer	8 bytes	8 bytes
Hardware Requirements	One on-chip timer	SysTick timer
User task priorities	1 - 254	1 - 254
Task switch time	<5.3 µsec @ 60 MHz	<2.6 µsec @ 72 MHz
Interrupt lockout time	<2.7 µsec @ 60 MHz	Not disabled by RTX

Podrobný popis a konfigurace **RTX KEIL**

Timer Tick Interrupt – generuje periodicky přerušení v privilegovaném modu. Hlavní časovač pro scheduler. Konfigurace v souboru [RTX Config.c](#) .

System Task Manager – se aktivuje po každém **Timer Tick Interrupt**, má max. prioritu , přepíná kontext. Čas CPU je dělen na části (např. 10 ms) a procesy jsou simultanně prováděny. Funkce **os_tsk_pass** nebo **wait** způsobí přepnutí kontextu. Nastavení času přepnutí v souboru [RTX Config.c](#) .

Task Management

Idle Task – jestliže žádný task neběží, ten **musí** pracovat. Periferie mohou pracovat.

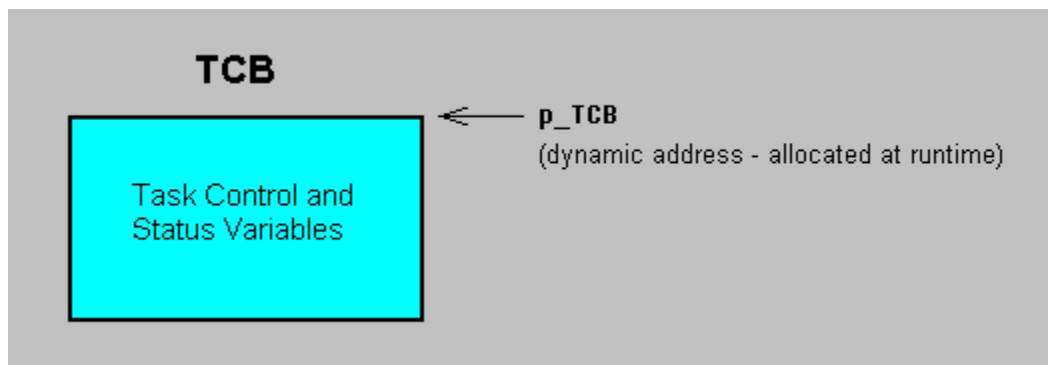
```
for (;;);
```

Task Management

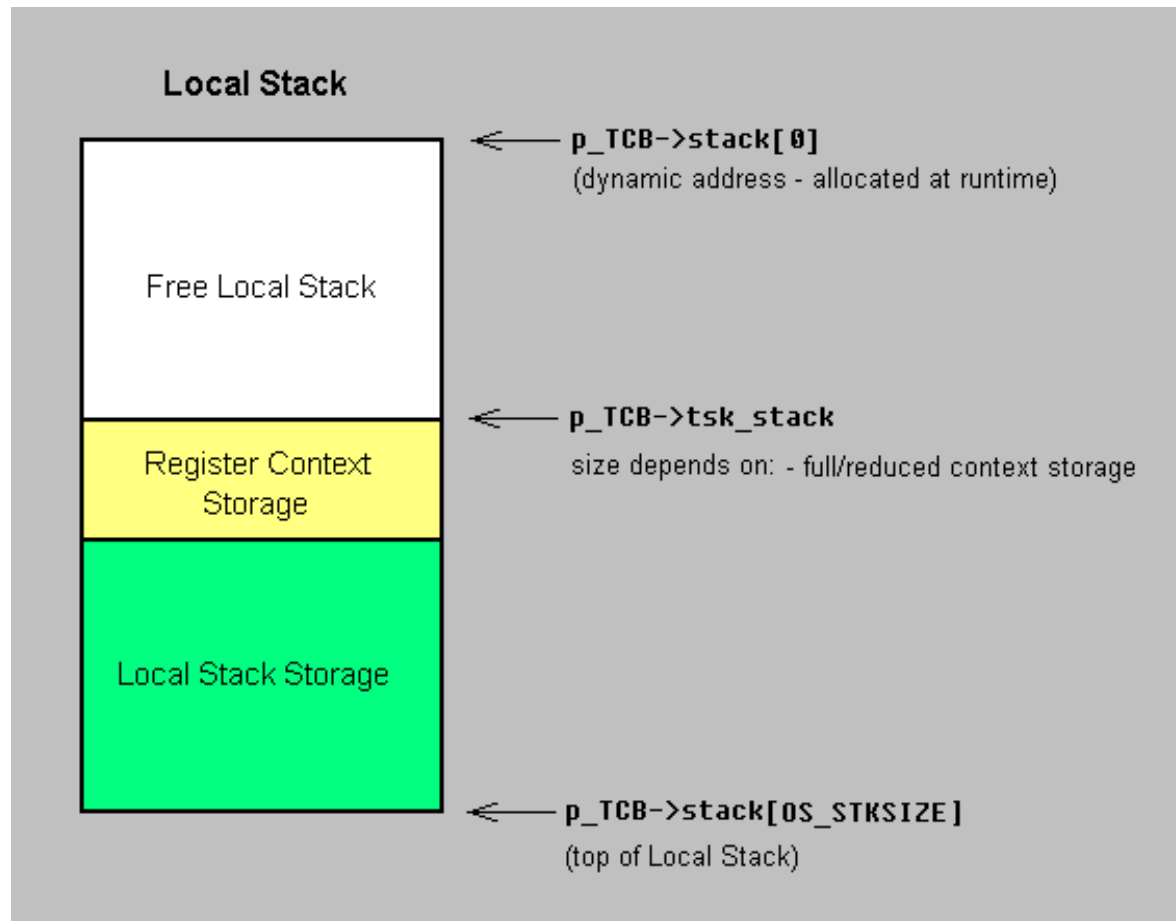
State	Description
RUNNING	The task that is currently running is in the RUNNING state. Only one task at a time can be in this state. The <code>os_tsk_self()</code> returns the Task ID (TID) of the currently executing task.
READY	Tasks which are ready to run are in the READY state. Once the running task has completed processing, RTX selects the next ready task with the highest priority and starts it.
WAIT_DLY	Tasks which are waiting for a delay to expire are in the WAIT_DLY State. Once the delay has expired, the task is switched to the READY state. The <code>os_dly_wait()</code> function is used to place a task in the WAIT_DLY state.
WAIT_ITV	Tasks which are waiting for an interval to expire are in the WAIT_ITV State. Once the interval delay has expired, the task is switched back to the READY State. The <code>os_itv_wait()</code> function is used to place a task in the WAIT_IVL State.
WAIT_OR	Tasks which are waiting for at least one event flag are in the WAIT_OR State. When the event occurs, the task is switched to the READY state. The <code>os_evt_wait_or()</code> function is used to place a task in the WAIT_OR state.
WAIT_AND	Tasks which are waiting for all the set events to occur are in the WAIT_AND state. When all event flags are set, the task is switched to the READY state. The <code>os_evt_wait_and()</code> function is used to place a task in the WAIT_AND state.
WAIT_SEM	Tasks which are waiting for a semaphore are in the WAIT_SEM state. When the token is obtained from the semaphore, the task is switched to the READY state. The <code>os_sem_wait()</code> function is used to place a task in the WAIT_SEM state.
WAIT_MUT	Tasks which are waiting for a free mutex are in the WAIT_MUT state. When a mutex is released, the task acquire the mutex and switch to the READY state. The <code>os_mut_wait()</code> function is used to place a task in the WAIT_MUT state.

WAIT_MBX	<p>Tasks which are waiting for a mailbox message are in the WAIT_MBX state. Once the message has arrived, the task is switched to the READY state. The <u>os_mbx_wait()</u> function is used to place a task in the WAIT_MBX state.</p> <p>Tasks waiting to send a message when the mailbox is full are also put into the WAIT_MBX state. When the message is read out from the mailbox, the task is switched to the READY state. In this case the <u>os_mbx_send()</u> function is used to place a task in the WAIT_MBX state.</p>
INACTIVE	<p>Tasks which have not been started or tasks which have been deleted are in the INACTIVE state. The <u>os_tsk_delete()</u> function places a task that has been started (with <u>os_tsk_create()</u>) into the INACTIVE state.</p>

Systemové zdroje



Každý proces popisuje **Task Control Block** (TCB). Jsou alokovány dynamicky v paměti RAM. Blok alokuji funkce RTX [os_tsk_create](#) nebo [os_tsk_create_user](#). Velikost TCB definuje [RTX_Config.c](#). Každý task má vymezen v paměti RAM **local Stack**.



Přepínací strategie (Scheduling Options)

- Pre-emptive scheduling

Každý task má **různou prioritu** and will run until it is pre-empted or has reached a blocking OS call.

- Round-Robin scheduling

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

- Co-operative multi-tasking

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