Task Execution
(slides are based on Prof. Dr. Jian-Jia Chen and http://www.freertos.org)

Anas Toma

LS 12, TU Dortmund

December 14, 2017
Task Creation (Recall)

- All the related information of a task is stored in a task control block (TCB) so that the operating systems can use it for operating on a task.

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top of Stack</td>
<td>pointer to the last item placed on the stack for the task</td>
</tr>
<tr>
<td>Task State</td>
<td>list item used to place the TCB in ready and blocked queues</td>
</tr>
<tr>
<td>Event List</td>
<td>list item used to place the TCB in the event lists</td>
</tr>
<tr>
<td>Priority</td>
<td>task priority (0=lowest)</td>
</tr>
<tr>
<td>Stack Start</td>
<td>pointer to the start of the process stack</td>
</tr>
<tr>
<td>Others</td>
<td>other information</td>
</tr>
</tbody>
</table>
Task Execution Sequence

Kernel

Task 1

Task 2

Kernel runs in tick interrupt to select next task

Tick interrupt occurs

Newly selected task runs when the tick interrupt completes

t1 t2 t3

[http://www.freertos.org]
Task Execution - Prior to the RTOS tick interrupt

![Diagram showing task execution context and registers](http://www.freertos.org)
Task Execution - RTOS tick interrupt occurs

- PC is placed on the TaskA stack by interrupt.

**General Purpose Registers**
- R0(A)
- R1(A)
- ...
- R30(A)
- R31(A)

**Status**
- SREG(A)

**Program Counter**
- PC(A)

**Stack Pointer**
- SPH
- SPL

**TaskA Code**
- LDI R0, 0
- LDI R1, 1
- ADD R0, R1

**TaskA Stack**
- PC(A)
- 0xff
- 0xee

[http://www.freertos.org](http://www.freertos.org)
Task Execution - RTOS tick interrupt is executed
Incrementing the Tick Count

- `vTaskIncrementTick()` executes after the TaskA context has been saved.
- TaskB has higher priority than TaskA and is ready to run.
- `vTaskSwitchContext()` selects TaskB as the task to be given processing time when the ISR completes.
Task Execution - TaskB stack pointer is retrieved

Stack pointer now points to the top of the TaskB context

TaskB Stack

- R31(B)
- R30(B)
- ...
- R1(B)
- SREG(B)
- R0(B)
- PC(B)
- 0x12
- 0x34

TaskB context saved when TaskB was suspended

TaskB application stack

The kernel stores a copy of the stack pointer for each task

Copy of TaskA Stack Pointer
- SPH
- SPL

Copy of TaskB Stack Pointer
- SPH
- SPL

[http://www.freertos.org]
Task Execution - Restore the context of TaskB

TaskB context has been restored

- **General Purpose Registers**
  - R0(B)
  - R1(B)
  - R30(B)
  - R31(B)

- **Status**
  - SREG(B)

- **Program Counter**
  - PC

- **Stack Pointer**
  - SPH
  - SPL

- **TaskB Code**
  - CLR R15
  - MOVW R18, R14
  - CALL 0xC4

- **TaskB Stack**
  - PC(B)
    - 0x12
    - 0x34

[http://www.freertos.org]
Task Execution - The RTOS tick exits

TaskB will now execute on return from interrupt

General Purpose Registers
- R0(B)
- R1(B)
- ...
- R30(B)
- R31(B)

Status
- SREG(B)

Program Counter
- PC

Stack Pointer
- SPH
- SPL

TaskB Code
- CLR R15
- MOVW R18, R14
- CALL 0xC4

TaskB Stack
- 0x12
- 0x34

[http://www.freertos.org]
Task Control Block and Task Execution

- Central Processing Unit (CPU)
  - Program Counter (PC), Stack Pointer (SP), Registers (Reg)
- Task Control Block (TCB)
  - Stack pointer, Waiting time (Tme), priority (Prio)
typedef struct tskTaskControlBlock{
  volatile portSTACK_TYPE *pxTopOfStack; /*< Points to the location of the last item placed on the tasks stack. */
  ... 
  xListItem xGenericListItem; /*< List item used to place the TCB in ready and blocked queues. */
  xListItem xEventListItem; /*< List item used to place the TCB in event lists. */
  unsigned portBASE_TYPE uxPriority; /*< The priority of the task where 0 is the lowest priority. */
  portSTACK_TYPE *pxStack; /*< Points to the start of the stack. */
  signed char pcTaskName[configMAX_TASK_NAME_LEN]; /*< Descriptive name given to the task when created. Facilitates debugging only. */
... 
} tskTCB;
xTaskGenericCreate

```c
signed portBASE_TYPE xTaskGenericCreate( ... ){
    tskTCB * pxNewTCB;
    configASSERT( pxTaskCode );
    configASSERT( ((uxPriority & (~portPRIVILEGE_BIT)) < configMAX_PRIORITIES) );
    /* Allocate the memory required by the TCB and stack for the new task, checking that the allocation was successful. */
    pxNewTCB = prvAllocateTCBAndStack( usStackDepth, puxStackBuffer );
    if( pxNewTCB != NULL ) {
        /* Setup the newly allocated TCB with the initial state of the task. */
        prvInitialiseTCBVariables( pxNewTCB, pcName, uxPriority, xRegions, usStackDepth );
    }
    ...
xTaskGenericCreate (cont.)

1 /* We are going to manipulate the task queues to add this task to a ready list, so must make sure no interrupts occur. */
2 taskENTER_CRITICAL(); {
3 uxCURRENTNUMBEROFTASKS++;
4 if ( pxCURRENTTCB == NULL ) {
5 /* There are no other tasks, or all the other tasks are in the suspended state — make this the current task. */
6 pxCURRENTTCB = pxNEWTCB;
7 .... } 
8 else{
9 /* If the scheduler is not already running, make this task the current task if it is the highest priority task to be created so far. */
10 if ( xSCHEDULERRUNNING == pdFALSE ){
11 if ( pxCURRENTTCB->uxPriority <= uxPriority ){
12 pxCURRENTTCB = pxNEWTCB;
13 }
14 }
15 ....
xTaskGenericCreate (cont.)

1 prvAddTaskToReadyQueue( pxNewTCB );
2 xReturn = pdPASS;
3 ...
4 taskEXIT_CRITICAL();
5 }
6 ...
7 if( xReturn == pdPASS ){
8 if( xSchedulerRunning != pdFALSE ){
9  /* If the created task is of a higher priority than the
     current task then it should run now. */
10 if( pxCurrentTCB->uxPriority < uxPriority ){
11  portYIELD_WITHIN_API();
12 }
13 }
14 }
15 return xReturn;
16 }

Lets read the source code for a while.