Verifying the FreeRTOS Real-Time OS

Deepak D'Souza

Department of Computer Science and Automation
Indian Institute of Science, Bangalore.

Joint work with Sumesh Divakaran, Anirudh Kushwah, Prahlad Sampath, Jim Woodcock, and others.

Project funded by UKIERI (2009-12)

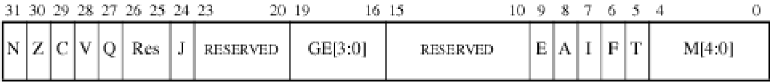
10 January 2013

Outline

How interrupts are handled on an ARM processor

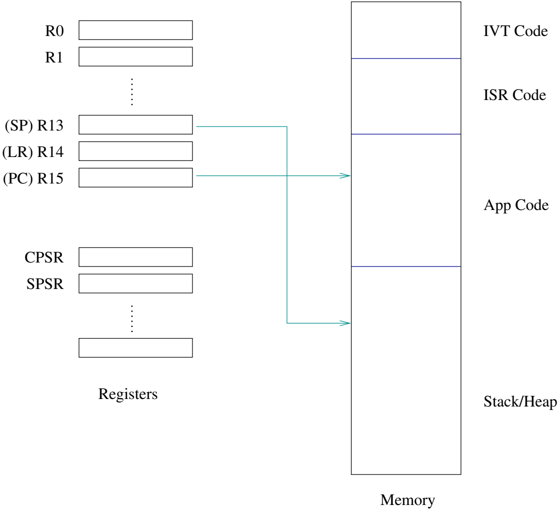
- Various kinds of interrupts may be generated (hardware eg. timer, software, instruction exceptions).
- Corresponding mode bits are set in CPSR[4:0] (Eg. 10011 for SWI).

Exception	Resulting Mode	IVT address	Priority
Reset	Supervisor	0x00000000	1
Undefined Inst.	Undef	0x00000004	6
Software Interrupt	Supervisor	0x00000008	6
Abort prefetch	Abort	0x0000000C	2
Abort data	Abort	0x00000010	2
IRQ	IRQ	0x00000018	4
FIQ	FIQ	0x0000001C	3



What the processor does on an interrupt

- Saves current PC in LR', CPSR in CPSR'.
- Changes to "super" mode.
- Disables lower priority interrupts.
- Branches to appropriate IVT entry.



What RTOS provides the programmer

Ways to:

- Create and manage multiple tasks.
- Schedule tasks based on priority-based pre-emption.
- Let tasks communicate (via message queues, semaphores, mutexes).
- Let tasks delay and timeout on blocking operations.

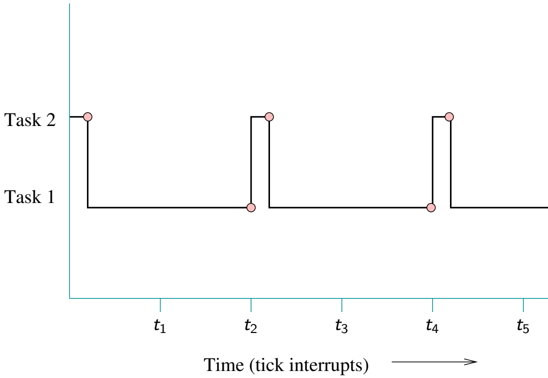
Example application that uses RTOS

Sample RTOS application

```
int main(void){
    xTaskCreate(foo, "Task 1", 1000, NULL, 1, NULL);
    xTaskCreate(bar, "Task 2", 1000, NULL, 2, NULL);
    vTaskStartScheduler();
}

void foo(void* params){
    for(;;);
}

void bar(void* params){
    for(;;){
        vTaskDelay(2);
    }
}
```

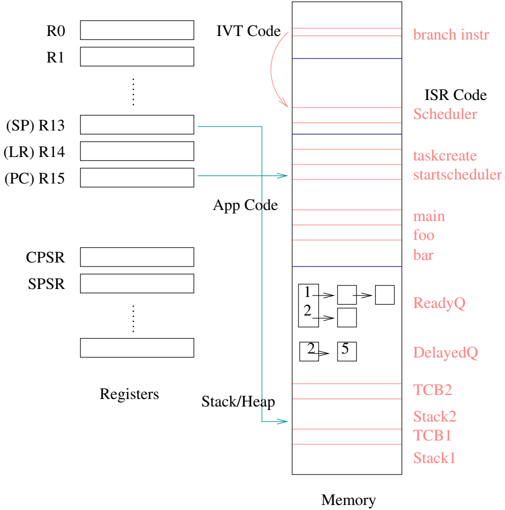
Task execution in example application

```
int main(void){
    xTaskCreate(foo, "Task 1", ...);
    xTaskCreate(bar, "Task 2", ...);
    vTaskStartScheduler();
}

void foo(void* params){
    for(;;);
}

void bar(void* params){
    for(;;){
        vTaskDelay(2);
    }
}
```



Example application: execution sequence

main
- vtaskCreate(1)
- vtaskCreate(2)
- vtaskStartScheduler()
  - create Idle task
- task2
  - vtaskdelay()
    - yield()
- task1
  - timer interrupt
  - timer interrupt
- task2



About RTOS implementation

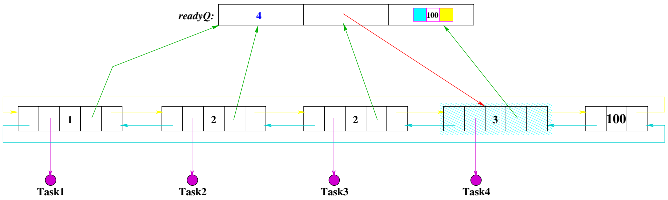
- Written mostly in C.
- Assembly language for processor-specific code.
- Portable:
  - Processor independent code is in 3 C files.
  - Processor dependent code (called a "port" in RTOS) is organised by Compiler-Processor pairs.
  - (19 compilers, 27 processors supported).
- Small footprint ($\approx$3,000 lines), engineered for efficiency.
- Well-written, and well-documented through comments.

Key data structures: Task Control Block

pxTopOfStack	points to the top element in stack
xMPUSettings	MPU setting – part of port layer
xGenericListItem	to place the Task in READY and BLOCKED lists
xEventListItem	to place the Task in event lists
uxPriority	priority of the task
pxStack	points to the start of the stack
pcTaskName	descriptive name for task – for debugg
pxEndOfStack	points to the end of stack – for checking overflows
uxCriticalNesting	for critical section nesting
uxTCBNumber	for tracing the scheduler – the task count
uxBasePriority	for priority inheritance – last assigned priority
pxTaskTag	task hook function
ulRunTimeCounter	MPU time used by the task

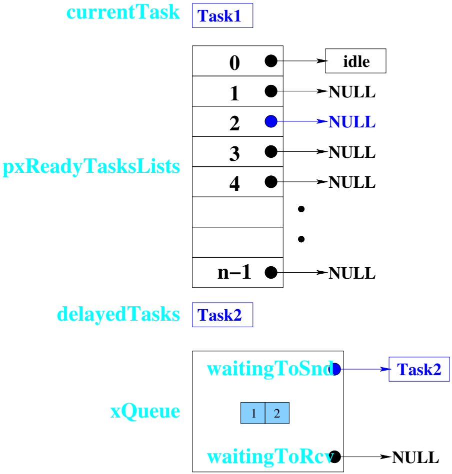
Task Control Block (TCB)

Key data structures: xList



- Operations it provides: initialize, insert at end, insert ordered by itemvalue, remove a node, set itemvalue of a node, etc. (some 13 operations).
- xList is used to implement
  - FIFO queues (ReadyQ),
  - priority queues (Delayed list, Event lists).

RTOS data structures using xList



Extracts from code: macro to add task to ready queue

```
/* Place the task represented by pxTCB into the appropriate
ready queue for the task. It is inserted at the end of the
list. */

#define prvAddTaskToReadyQueue(pxTCB){
    if(pxTCB->uxPriority > uxTopReadyPriority){
         uxTopReadyPriority = pxTCB->uxPriority;
    }
    vListInsertEnd((xList*) &(pxReadyTasksLists[pxTCB->uxPriorit
              &(pxTCB->xGenericListItem));
}
```

Extracts from code: vTaskDelay()

```
void vTaskDelay(portTickType xTicksToDelay){
    portTickType xTimeToWake;
    signed portBASE_TYPE xAlreadyYielded = pdFALSE;

    if( xTicksToDelay > (portTickType) 0){
        vTaskSuspendAll();
    /* Calculate the time to wake - this may overflow but this
       is not a problem. */
    xTimeToWake = xTickCount + xTicksToDelay;
    /* We must remove ourselves from the ready list before addin
       ourselves to the blocked list as the same list item is us
       for both lists. */
    vListRemove((xListItem *) &(pxCurrentTCB->xGenericListItem))
    /* The list item will be inserted in wake time order. */
    listSET_LIST_ITEM_VALUE(&(pxCurrentTCB->xGenericListItem),
                            xTimeToWake);
    ....
    portYIELD_WITHIN_API();
```

Extracts from code: vPortYieldProcessor()

```
/* ISR to handle manual context switches like taskYIELD()). */
void vPortYieldProcessor(void) __attribute__((interrupt("SWI"),

void vPortYieldProcessor(void){
  /* Within an IRQ ISR the link register has an offset from
     the true return address... */
  __asm volatile ("ADD LR, LR, #4");

  /* Perform the context switch.  First save the context of
     the current task. */
  portSAVE_CONTEXT();

  /* Find the highest priority task that is ready to run. */
  __asm volatile ("BL vTaskSwitchContext");

  /* Restore the context of the new task. */
  portRESTORE_CONTEXT();
}
```

Extracts from code: portSAVECONTEXT()

```
#define portSAVE_CONTEXT() {
    extern volatile void * volatile pxCurrentTCB;
    /* Push R0 as we are going to use the register. */
        /* Set R0 to point to the task stack pointer. */
        "STMDB  SP,{SP}^"
        "SUB    SP, SP, #4"
        "LDMIA  SP!,{R0}"
        /* Push the return address onto the stack. */
        "STMDB  R0!, {LR}"
        /* Now we have saved LR we can use it instead of R0. */
        "MOV    LR, R0"
        /* Pop R0 so we can save it onto the system mode stack.
        "STMDB  LR,{R0-LR}"
        /* Store the new top of stack for the task. */
        "LDR    R0, =pxCurrentTCB"
        "LDR    R0, [R0]"
        "STR    LR, [R0]"
    )
```

Extracts from code: Critical section / Disabling interrupts

```
void vTaskEnterCritical(void){   /* in task.c */
    ...
    portDISABLE_INTERRUPTS();
}

#define portDISABLE_INTERRUPTS()
    __asm volatile (
          "STMDB  SP!, {R0}"   /* Push R0.*/
          "MRS    R0, CPSR "   /* Get CPSR.*/
          "ORR    R0, R0, #0xC0"   /* Disable IRQ, FIQ.*/
          "MSR    CPSR, R0"   /* Write back modified value. */
          "LDMIA  SP!, {R0}"  /* Pop R0.*/
      )
```

Main functionality of RTOS

- Implement its stated scheduling policy (fixed priority pre-emptive scheduling).
- Trap SWI interrupts
  - Find highest priority ready task to run.
  - Save context of yielding task.
  - Restore context of new task.
- Trap timer IRQ interrupt
  - Update tickcount,
  - Check delayed tasks, and move to ready if required,
  - Switch context if required.
- Provide API's for:
  - Task creation, deletion, set priority, etc.
  - Inter-task communication through queues, semaphores, and mutexes.

Separate requirements: Scheduling-related and port-specific

- Scheduling-related
  - Implement its stated scheduling policy (fixed priority pre-emptive scheduling).
  - Provide API's for:
    - Task creation, deletion, set priority, etc.
    - Inter-task communication through queues, semaphores, and mutexes.
  - Handle timer event correctly
    - Update tickcount,
    - Check delayed tasks, and move to ready if required,
- Port-specific
  - Trap SWI and timer interrupts correctly.
  - Perform context-switching (save and restore) correctly.
  - Provide correct "enterCritical" and "exitCritical" implementation.

Example Z model: Resource allocater

ResourceManager

$free : \mathbb{F}\,\mathbb{N}$

Allocate

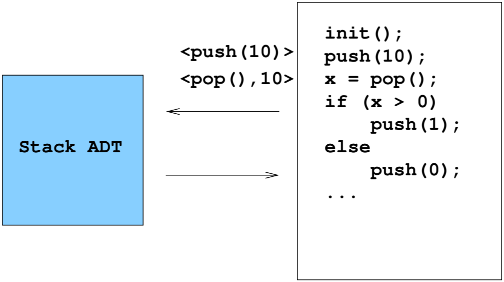
$\Delta ResourceManager$

$r! : \mathbb{N}$

$r! \in free \ \wedge \ free' = free \setminus \{r!\}$

Notion of refinement

Program using a `Stack` ADT:



- Consider `Stack'` which satisfies property:

  Every sequence of operations on `Stack'` can be matched by a sequence on `Stack`.

  Then `Stack'` is said to refine `Stack`.
- If a client program is happy with an ADT, it will also be happy with a refinement of it.

Refinement example: Resource allocater

ResourceManager

$free : \mathbb{F}\, \mathbb{N}$

Allocate

$\Delta ResourceManager$

$r! : \mathbb{N}$

$r! \in free \;\wedge\; free' = free \setminus \{r!\}$

Refinement example: Resource allocater

ResourceManager

$free : \mathbb{F}\,\mathbb{N}$

Allocate

$\Delta ResourceManager$

$r! : \mathbb{N}$

$r! \in free \ \wedge \ free' = free \setminus \{r!\}$

A refinement of `allocate`:

$Allocate_1$

$\Delta ResourceManager$

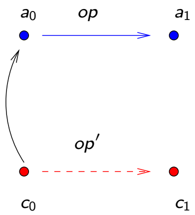
$r! : \mathbb{N}$

$r! = \min free \ \wedge \ free' = free \setminus \{r!\}$

Sufficient condition for refinement

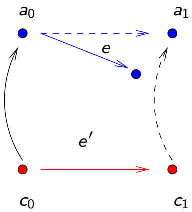
Init strengthening: every concrete init state is related to an abstract init state.

Guard strengthening:



If op is enabled in an abstract state then the concrete op is also enabled in any related concrete state.

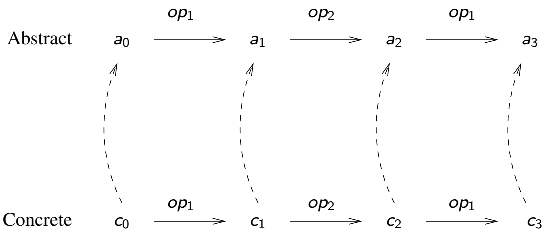
Simulation:



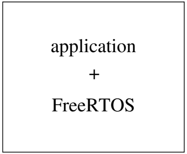
If a_o and c_0 are related, if op is enabled in a_0, and if a concrete op op' takes us from c_0 to c_1, then there exists a_1 related to c_1, such that op takes us from a_0 to a_1.

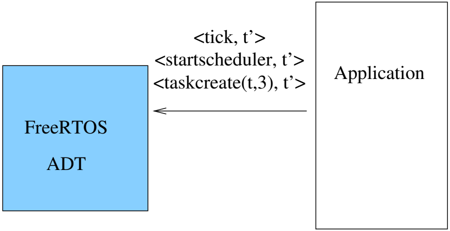
Refinement conditions imply matching sequence property

The concrete is simulated-by the abstract.



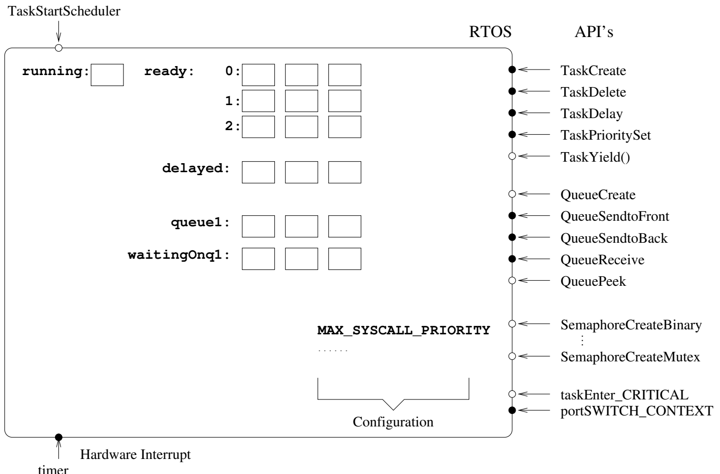
Viewing FreeRTOS as an ADT





FreeRTOS as an ADT

- The OS essentially services API calls from the running task.
- View as a state machine with operations corresponding to API calls.



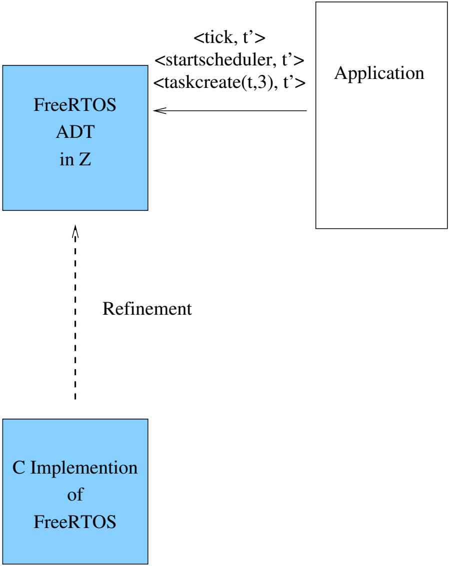
RTOS as ADT in Z

CreateTask

$$\Delta Task$$
$$taskIn?: TASK$$
$$prio?: \mathbb{N}$$

$$taskIn? \notin tasks$$
$$tasks' = tasks \cup \{taskIn?\}$$
$$ready' = ready \frown \langle taskIn? \rangle$$
$$priority' = priority \oplus \{(taskIn? \mapsto prio?)\}$$

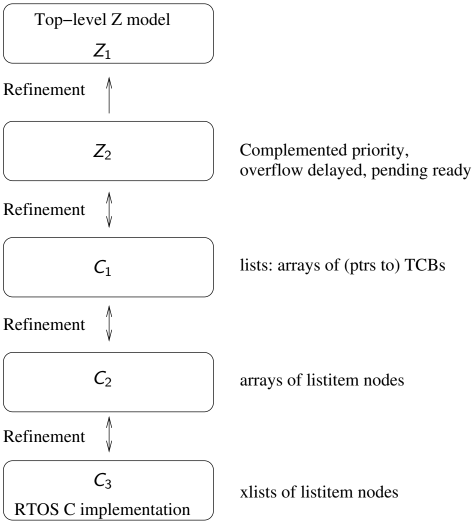
Specification of FreeRTOS



Verification Strategy

For $Z_2 \leftrightarrow C_1$:

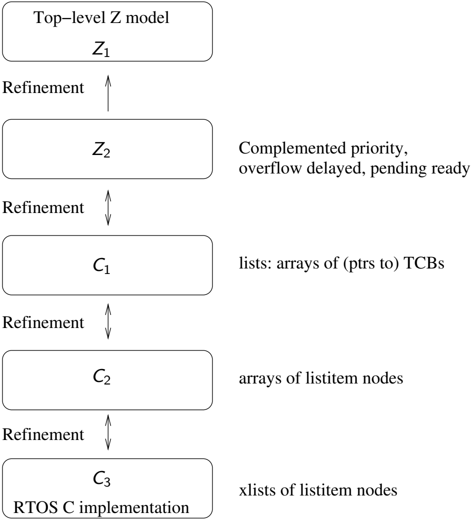
- Import guards, invariants, and BAP from Z_2.
- Check validity of model C_1 in VCC.



Verification Strategy

For $C_1 \leftrightarrow C_2$, $C_2 \leftrightarrow C_3$:

- Phrase and check validity of models C_2 and C_3 in VCC.
- Check refinement conditions in VCC.



Verification in VCC: xList data structure

- Verified that xList is a refinement of a simple array (ghost map in VCC) based implementation.
- original code 500 lines, VCC annotations 600 lines.

```
_(ghost xListItem * xArray[portTickType])
/* map is used as an ARRAY. */
        /* ADT that is maintained as a shadow copy. */

_(ghost portTickType xArrayIndex[xListItem * ])
/* map that stores the index of element in the array. */
....

void vListInsertEnd( xList *pxList, xListItem *pxNewListItem ){
xListItem *pxIndex;
pxIndex = pxList->pxIndex;
_(assert pxIndex->pxNext \in pxList->\owns)
_(assert pxIndex->pxPrevious \in pxList->\owns)
_(unwrap pxNewListItem)
pxNewListItem->pxNext = pxIndex->pxNext;
pxNewListItem->pxPrevious = pxList->pxIndex;
_(wrap pxNewListItem)
```

Verification in VCC: taskdelay API

```
void vTaskDelay( portTickType xTicksToDelay )
{
        portTickType xTimeToWake;
        signed portBASE_TYPE xAlreadyYielded = pdFALSE;
        if( xTicksToDelay > ( portTickType ) 0 )
        {
                _(unwrap (&SCR))
                _(assert \forall int i ; ( (i >= 0) && (i < configMAX_PRIORITIE
\wrapped(&pxReadyTasksLists[i])) )

                uxSchedulerSuspended++;
                        xList *list = (pxReadyTasksLists + uxTopReadyPriority);
                        _(unchecked)xTimeToWake = xTickCount + xTicksToDelay;
                        _(unwrapping pxCurrentTCB){
                                _(assert (&( pxCurrentTCB->xGenericListItem ))
                                vListRemove( (&( pxCurrentTCB->xGenericListItem
uxTopReadyPriority))  );

                                _(assert \wrapped(pxReadyTasksLists + uxTopRead
                                _(assert (pxReadyTasksLists +uxTopReadyPriority
(pxReadyTasksLists +uxTopReadyPriority)->tSize)

                                _(unwrapping (&( pxCurrentTCB->xGenericListItem
```

Problem with taskPrioritySet() function in RTOS

Problem found by Sumesh Divakaran while trying to understand code in detail.

- According to RTOS User Guide: When an unblocking event occurs,

  > *The task that is unblocked will always be the highest priority task that is waiting for the event. If the blocked tasks have equal priority, then the task that has been waiting for the longest period will be unblocked.*

- However, if taskPrioritySet is called on a blocked task, its new priority is not considered while selecting the task to be unblocked.

Example to illustrate setpriority problem

RTOS application

```
int main(void){
        xTaskCreate(vTask1,"Task1",configMINIMAL_STACK_SIZE,NULL,1,&xTask1Handle);
        xTaskCreate(vTask2,"Task2",configMINIMAL_STACK_SIZE,NULL,2,&xTask2Handle);
        xTaskCreate(vTask3,"Task3",configMINIMAL_STACK_SIZE,NULL,3,&xTask2Handle);
        vTaskStartScheduler();
}

void vTask1(void *pvParameters){
        long lData = 10;
        xQueueSendToBack(xQueue,&lData,0);
        for(;;);
}

void vTask2(void *pvParameters){
        long lData = 20;
        xQueueSendToBack(xQueue,&lData,0);
        for(;;);
}

void vTask3(void *pvParameters){
        long lData = 30;
        xQueue = xQueueCreate(1,sizeof (long));
        xQueueSendToBack(xQueue,&lData,0);
        vTaskDelay(2);
        vTaskPrioritySet(xTask1Handle,4);
        xQueueReceive(xQueue,&lData,0);
        vTaskPrioritySet(NULL,0);
        for(;;);
}
```

Sequence of events produced by application

Event sequence produced by test application

```
main, xTaskCreate(Task1,1)
main, xTaskCreate(Task2,2)
main, xTaskCreate(Task3,3)
Task3, xQueueCreate(xQueue,1)
Task3, xQueueSendToBack(xQueue,30)
Task3, vTaskDelay(2);

Task2, xQueueSendToBack(xQueue,20)_b;
Task1, xQueueSendToBack(xQueue,10)_b;

Task3, vTaskPrioritySet(Task1,4);
Task3, xQueueReceive(xQueue,30);
Task3, vTaskPrioritySet(Task3,0);

Task2, xQueueSendToBack(xQueue,20)_e;
```

Other bugs found

- Problem with priority inheritance mechanism.
- Problem with `vTaskSuspend` and `vTaskResume` API's.
- Problem with scheduling newly created tasks.

Benefits of verification technique

- Finding bugs:
  - Problem with `vTaskSuspend` and `vTaskResume` API's.
  - Problem with scheduling newly created tasks.
  - Problem with `vTaskPrioritySet` API in RTOS.
- Gives us conditions for correct API usage:
  - `vTaskDelay` should not be called on sole ready task (FreeRTOS crashes!).

Other things to check: Absence of data-races

- Tasks are essentially threads that can be interleaved in execution.
- API code called by tasks should not lead to a data-race when interrupted and interleaved with an API call from another task.
- For example:
  - Task1 calls `QueueSend` API
  - Pointers start getting adjusted to insert new message in the Queue
  - Tick interrupt occurs
  - Task2 runs and makes a call to `QueueReceive`.
- Need to ensure that API's use critical sections when they have to.
- Could use a data-race detection tool on suitably modified RTOS code.

Port-specific aspects of correctness

- Trap SWI and timer interrupts correctly.
- Perform context-switching (save and restore) correctly.
  - `SAVE_CONTEXT` saves the "necessary" information about the swapped-out task on its stack.
  - `RESTORE_CONTEXT` correctly restores this information from the task's stack.
- Provide correct "enterCritical" and "exitCritical" implementation.
  - Interrupts should be correctly disabled and re-enabled.

The End

Thank you.