Verifying Data-race Freedom of Kernel APIs in a Real Time Operating System

Suvam Mukherjee
Advisor : Prof. Deepak D' Souza
Department of Computer Science and Automation, Indian Institute of Science

Problem Definition

❑ Verify data-race freedom of a library of kernel APIs.

❑ Case Study: FreeRTOS, a popular real-time embedded operating system
- Find data-races
- Create data-race free version of FreeRTOS

Preliminaries

- Examples of **Kernel API** Operations:
  - Task Creation,
  - Queue Creation,
  - Inter-Task Communication, etc.

Application Programmer Interface (API)	Interrupt Service Routines (ISR)
xTaskCreate	xTaskGetTickCountFromISR
VTaskDelay	xTaskResumeFromISR
vTaskDelete	xQueueIsEmptyFromISR
xQueueCreate	xQueueIsFullFromISR
xTaskGetTickCount	uxQueueMessagesWaitingFromISR
uxTaskGetNumberOfTasks	xQueueReceiveFromISR
uxTaskPriorityGet	xQueueSendFromISR
vTaskPrioritySet	Tick
...	

- **Data-races**
  - Non atomic execution of critical sections
  - Can cause system failures
  - Difficult to reproduce and debug, as it depends on specific interleavings

Example 1:

```
global int x = 0; // shared variable

void main()
{
        start(thread1);
        start(thread2);

void thread1()                          void thread2()
{                                       {
        x = x + 1;                              x = x + 1;
}                                       }

        join(thread1);
        join(thread2);
        assert(x==2);
}
```

Assertion may be Violated!

Example 2:

```
void vQueueDelete( xQueueHandle pxQueue )
{
        traceQUEUE_DELETE( pxQueue );
        vQueueUnregisterQueue( pxQueue );
        vPortFree( pxQueue->pxHead );
        vPortFree( pxQueue );
}
```

API

```
unsigned portBASE_TYPE uxQueueMessagesWaitingFromISR(
                          const xQueueHandle pxQueue)
{
        unsigned portBASE_TYPE uxReturn;
        uxReturn = pxQueue->uxMessagesWaiting;
        return uxReturn;
}
```

Bad Queue state may be read!

Interrupt

- **Verification**

❑ Guarantees for any application with an arbitrary number of tasks (unlike bug-finding)

❑ Helps to create a version of the RTOS certified against data races

Proposed Solution

1. Model control flow
2. Model accesses to shared data structures
3. Perform suitable abstractions
4. Model check a finite subset of reduced models
   - Enhances scalability
   - Preserves soundness guarantees

A Case Study: FreeRTOS

❑ One of the most popular real time operating systems

❑ Over 100,000 downloads in 2014 alone

❑ Uses a preemptive flag-based and priority-based scheduling policy

❑ Rich set of APIs performing a wide variety of operations
- Creating tasks,
- Creating queues,
- Communication between tasks, and many more

❑ Presence of interrupts
- Specific set of functions which interrupt handlers can invoke

API

Critical Section

Scheduler Suspended

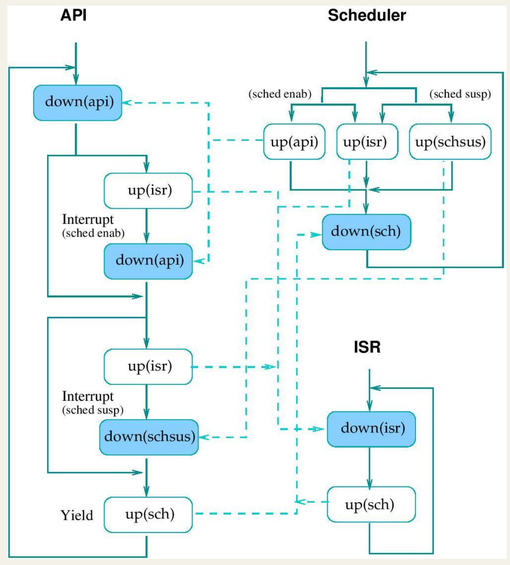
```
int QueueSend(QHandle pxQ, void *ItemToQueue) {
  // Repeat till successful send
  DISABLE_INTERRUPTS();
  if(!QueueFull(pxQ)) { // queue is not full
    // Copy data to queue
    CopyDataToQueue(pxQ, ItemToQueue);
    if(!empty(pxQ->WaitingToReceive)) {
      // Move task at head of WaitingToReceive list to
      // ReadyTasksList
    }
    ENABLE_INTERRUPTS();
    return PASS;
  }
  ENABLE_INTERRUPTS();

  // Reach here when queue is full
  ++SchedulerSuspended; // Suspend scheduler
  LockQueue(pxQ); // Lock queue
  if(QueueFull(pxQ)) { // check if queue is still full
    // move current task from ReadyTasksList to the
    // WaitingToSend list of pxQ
  }
  UnlockQueue(pxQ);
  // Resume scheduler, yield if higher priority task woken
  --SchedulerSuspended; // Resume scheduler
  if () { // higher priority task woken
    YIELD();
  }
}
```

ISR

```
void TaskIncrementTick() {
  if(SchedulerSuspended == 0) {
    ++TickCount;
    if(TickCount == 0) {
      // swap delayed lists
      Temp = DelayedTaskList;
      DelayedTaskList = OverflowDelayedTaskList;
      OverflowDelayedTaskList = Temp;
      ...
    }
    // Move tasks whose time-to-awake is now
    // from DelayedTaskList to ReadyTasksList.
    CheckDelayedTasks();
  }
  else {
    ++MissedTicks;
  }
}
```

Proposed Concurrency Model



Experimental Evaluation

❑ Model checking M2, on a 128 GB RAM, 2 X (8 core Intel Xeon Haswell 2.6 GHz) system

Running time: > 3 hours

RAM used: > 39 GB

States explored: 4.43 X 10^8

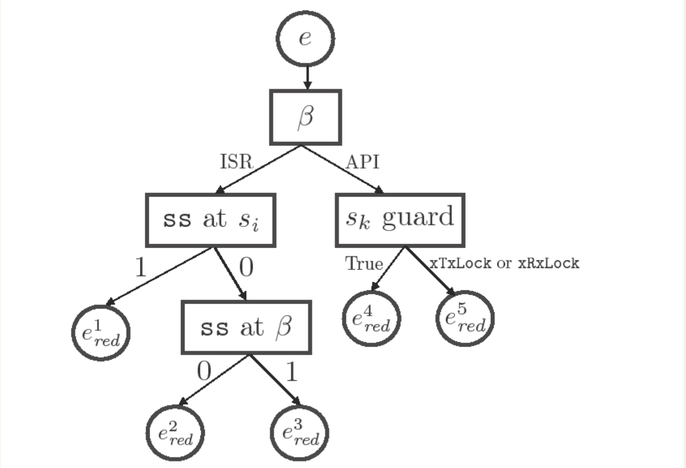
Hard!

❑ Reduced models
- **Process 1: API**
- **Process 2: API**
- **Process 3: ISR**
- **Process 4: Tick Interrupt**
- **Process 5: Scheduler**

❑ Model check 17 X 17 X 7 = **2023** such reduced models
(API API ISR)

Problem	RAM	Running Time
M2	> 39 GB	> 3 hours
M_red	~ 3GB	~1.85 hours

Proof-Sketch



Any execution **e** with a data-race, of the model with **n** tasks, can be reduced to an execution of *some* reduced model. Moreover, the latter execution preserves the data-race.

Some Identified Data-races

Racing Data Structure	Task 1	Task 2	Type
userQueue	vQueueDelete	xQueueSend	H
userQueue	vQueueDelete	xQueueIsQueueFullFromISR	H
userQueue	xQueueSend	xQueueReceiveFromISR	H
userQueue	vQueueDelete	uxQueueMessagesWaitingFromISR	H
userQueue	vQueueDelete	xQueueReceive	H
userQueue	vQueueDelete	xQueueSendFromISR	H
userQueue	xQueueReceive	xQueueSendFromISR	H
userQueue	vQueueDelete	uxQueueMessagesWaiting	H
userQueue	xQueueSend	xQueueSendFromISR	H
userQueue	vQueueDelete	xQueueReceiveFromISR	H
userQueue	xQueueReceive	xQueueReceiveFromISR	H
userQueue	vQueueDelete	xQueueIsQueueEmptyFromISR	H
userQueue	vQueueDelete	vQueueDelete	H
pxCurrentTCB	xTaskCreate	Tick	B
pxCurrentTCB	xTaskCreate	xTaskCreate	B
pxCurrentTCB	vTaskResume	Tick	B
pxCurrentTCB	vTaskResume	xTaskCreate	B
uxPriority	xTaskCreate	vTaskPrioritySet	B
uxCurrentNumberOfTasks	xTaskCreate	uxTaskGetNumberOfTasks	B

H : Harmful
B : Benign

Conclusion

❑ Proposed an approach to model and exhaustively check a library of Kernel APIs in an RTOS for data races

❑ The proposed steps:
- Model control flow and access to shared data structures
- Perform suitable abstractions
- For scalability, model check a small number of reduced models

❑ Concrete instantiation of our approach
- Modelled concurrency behaviors of FreeRTOS Kernel APIs and ISRs
- Model checked 2023 reduced models in under 2 hours
- Detected 30 data races and classified them as harmful or benign
- Used the detected races to create a certified race-free version of FreeRTOS

Future Directions

❑ Carry out further instantiations, for example, OSEK, `java.util.concurrent` etc.

❑ Identify general patterns which allow reductions to model checking a finite set of "smaller" models