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,

Examples of Data Used
- The detected races to create a certified race
- Model check a finite subset of reduced models
- Verification
  - Model accesses to shared data structures
  - Model control flow
  - 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

Critical Section
- Scheduler
  - Suspend
  - Resume
- Suspended

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:
```
void thread1()
{
    x = x + 1;
}

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

Proposed Concurrency Model

Experimental Evaluation
- Model checking M2, on a 128 GB RAM, 2 x 8 core Intel Xeon Haswell 3.6 GHz system
  - Raced States explored: ~3 hours
  - 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 = ~2039 such reduced models

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

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
  - Performed suitable abstractions
  - For scalability, model check a small number of reduced models
- Concrete instantiation of our approach
  - Modelling concurrency behaviors of FreeRTOS kernel APIs and ISRs
  - Model checked 23 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

Table:

<table>
<thead>
<tr>
<th>Task 1</th>
<th>Task 2</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>userQueue</td>
<td>vQueueDelete</td>
<td>aQueueSend</td>
</tr>
<tr>
<td>userQueue</td>
<td>vQueueDelete</td>
<td>aQueueSendFull</td>
</tr>
<tr>
<td>userQueue</td>
<td>aQueueSend</td>
<td>aQueueSendFromISR</td>
</tr>
<tr>
<td>userQueue</td>
<td>aQueueReceiveFromISR</td>
<td>aQueueSendFromISR</td>
</tr>
<tr>
<td>userQueue</td>
<td>aQueueReceiveFromISR</td>
<td>vQueueMessagesWaiting</td>
</tr>
<tr>
<td>userQueue</td>
<td>aQueueReceiveFromISR</td>
<td>vQueueReceiveFromISR</td>
</tr>
<tr>
<td>userQueue</td>
<td>aQueueReceiveFromISR</td>
<td>vQueueDeleteFromISR</td>
</tr>
<tr>
<td>userQueue</td>
<td>aQueueReceiveFromISR</td>
<td>vQueueDeleteFullFromISR</td>
</tr>
</tbody>
</table>

Some Identified Data-races

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.