# Concurrency Analysis for Asynchronous APIs

Anirudh Santhiar and Aditya Kanade

Department of Computer Science and Automation, IISc

## Asynchronous Programming Model

- Waiting in line for your Idly

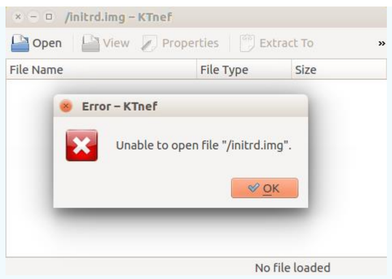
vs.

- Registering your order
- Doing other things
- Having the restaurant call you

However, asynchronous programs can suffer from bugs such as race conditions and deadlocks

Images courtesy guardian.com and en.wikipedia.org

## Race Example




**Bug:** Close the window when an error dialog is shown.

- The FileOpen event's handler spins a programmatic event loop during the time the error dialog is shown
- There is a race between FileHandler and QuitHandler that runs in the programmatic event loop

Goal: Reason about non-determinism introduced by programmatic event loops to detect such races.

## Deadlocks in Asynchronous Programs

Mixing synchronous and asynchronous waiting in C#'s asynchronous programming model can lead to deadlocks.

```
public static async Task<String> GetContentsAsync(Uri uri)
{
  using (var client = new HttpClient())
  {
    // asynchronous wait
    var contents = await client.GetStringAsync(uri);
    return Process(contents);
  }
}


public void Button1_Click(...)
{
  var t = GetContentsAsync(...);
  resultBox.Text = t.Result; // synchronous wait
}
```

- **t.Result** is a blocking call that prevents *GetContentsAsync* from completing
- In turn, the only way to unblock **t.Result** is for *GetContentsAsync* to complete

## Our Work

We analyze the concurrency behaviours of

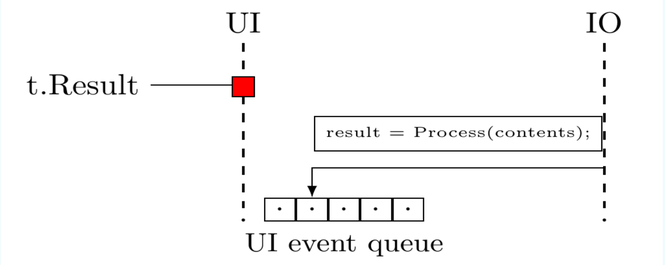
1. Event driven asynchronous libraries with programmatic event loops to detect races (joint work with S. Kaleeswaran)
2. C# asynchronous programs to find deadlocks

Software using this concurrency model includes OS APIs, GUI frameworks, web browsers and libraries for cloud computing

## Dynamic Race Detection

- Find bugs using instrumented non-buggy executions
- Design trace language to record interesting operations
- Design *happens-before rules* to detect possible *reorderings* of these
- Determine if there is a re-ordering of event handlers so that conflicting operations such as uses and frees can be reordered to induce bugs
- Notify programmer about such re-orderings along with debug information

## Deadlocks in Asynchronous Programs




**The deadlock is observed even though there is no explicit thread creation or locking**

- Design a static analysis to detect such deadlocks
- Static analysis captures C# semantics for scheduling and async/await
- Preliminary results are encouraging – found previously unknown deadlocks in 7 open source applications

## Races involving programmatic event loops

Event Loop:

```
while (! exit ) {
  e = nextEvent ( );
  process e;
}
```

- An Event Loop is the basic scheduling mechanism for programs that respond to asynchronous events
- We consider frameworks where event loops can also be spun programmatically by event handlers
- Prone to interference between handler spinning event loop and handler running inside the loop

## Race Detection: Technical Highlights and Results

- Powerful framework to **handle races beyond the state-of-the art**
- Account for all general scheduling scenarios e.g., **recursive and cascaded programmatic event loops**
- Novel sparse representation of happens-before relation enabling faster race detection

Efficient computation of the happens-before relation: 5X speedup over baseline

Our tool, SparseRacer found 13 new and harmful use-after-free race conditions in 9 popular open-source applications including Okular, Kate and KOrganizer

## Related Publications and Information

- Anirudh Santhiar, Shalini Kaleeswaran and Aditya Kanade. **Race Detection in the presence of Programmatic Event Loops**. Accepted, ISSTA '16
- Web page: http://www.iisc-seal.net/