“The race is not always to the swift, nor the battle to the strong, but that's the way to bet.”

– Hugh E. Keough
Anti-Patterns for Race Conditions:
- Unprotected access to shared variables
- Shared variables not declared volatile
- Not accounting for interrupts and task switching in timing analysis
- Ignoring non-reproducible faults

Race condition: multiple threads compete
- Computation outcome depends upon timing
  - Usually it is infrequent and hard to debug
- Concurrent access to shared variable
  - Need to lock shared resources
- Not accounting for multi-tasking
  - Task switch or interrupt causes delays
  - “Starvation” and priority inversion

Race Conditions

Software-Controlled Radiation Therapy Mishaps

Problems included:
- Operators “too fast” on keyboard (8 second window)
- Bypassed safety checks when counter rolled over to 0
Concurrent Management Bugs

- CPU switches among its tasks (multi-tasking)
  - What if switching happens at the wrong time?

- Concurrency bugs due to shared resources
  - Example: shared global variable, two tasks
    - Task 1 reads shared variable and computes new value
    - Task 2 preempts task 1, updates shared variable
    - Task 1 resumes, over-writing task 2’s update
  - Results of concurrency bug depend upon ordering
    - Usually bug won’t manifest (example: 9)
    - Sometimes bug will result in wrong value (example: 6, 8)

© 2017 Philip Koopman
**Mutex For Concurrency Management**

- **Easy solution for concurrency bug:**
  - Disable interrupts when touching shared variable
    - Inhibits task switches
    - But, need to keep it very brief to avoid timing problems

- **To hold resources longer, use a mutex**
  - "**Mutual Exclusion**" flag; True=busy / False=available
  - To access shared resource:
    - Get the mutex (wait for it to be false, then set to true)
    - Access shared resource
    - Other tasks will wait while mutex is locked (resource busy)
    - When done, set mutex to false to release resource
  - Mutexes are themselves a special type of shared variable
    - And therefore subject to race conditions!
    - Getting them right is tricky; let the RTOS do this for you
Minimize time interrupts are disabled
- Disabled task switching delays task switching
- **Blocking Time:** high priority tasks can miss deadlines

Mutexes indirectly cause blocking time
- **Priority Inversion:** low priority task blocks high priority task
  - Locked mutex prevents high priority task from making progress
  - Only affects tasks that actually use mutex, not all tasks
  - **BUT... there is a critical problem (next slide)**
Unbounded Priority Inversion

Priority inversion can be unbounded for three tasks:

- Medium priority task blocks high task without ever touching mutex:

```
Priority Inversion

TASK
PRIORITY:
High

Med

Low

TIME

Normal Execution  M  Mutex Locked (Critical Section)  F  Fails To Get Mutex

F

M

M

M

F
```
Solution to unbounded priority inversion: **priority inheritance**

- Task priority elevated when locking mutex; restored when frees mutex
- This is complicated! Let the RTOS handle it

**Priority Inheritance**

![Diagram showing bounded priority inversion]

- **Low: Hoisted**
  - Task with low priority
- **High**
  - Task with high priority
- **Med**
  - Task with medium priority
- **Low**
  - Task with low priority

- **Normal Execution**
- **M Mutex Locked (Critical Section)**
- **F Fails To Get Mutex**

**Tasks:**

- **Task 1:** Low priority
- **Task 2:** Medium priority
- **Task 3:** High priority

**Time:**

- **DELAYED**
- **Priority Increased While Using Mutex**
Mars Pathfinder Incident

July 4, 1997 – Pathfinder lands on Mars
- First US Mars landing since Vikings in 1976; first rover

But, a few days later...
- Multiple system resets occur via VxWorks RTOS
  - Watchdog timer saves the day! Sets system to safe state
  - Reproduced on ground; patch uploaded to fix it
- Scenario pretty much identical to High/Medium/Low priority picture
  - Developers didn’t have Priority Inheritance turned on!
  - Why? “The data bus task executes very frequently and is time-critical -- we shouldn't spend the extra time in it to perform priority inheritance” [Jones07]

https://goo.gl/W5wHrU
Best Practices Avoiding Race Conditions

■ Always consider task interactions
  ● What if task switches at a bad time?
  ● What if tasks read data at different times?
  ● What if half-formed data structure is read?
  ● What if multiple writers compete for data?
  ● Use RTOS services to help

■ Pitfalls:
  ● Failing to use interrupt masking or mutexes
    – Failing to deal with unbounded priority inversion
    – Failing to declared shared variables volatile
  ● Assuming that non-reproducible problems aren’t bugs
  ● Trying to write your own bullet-proof concurrency services

18-348 Lecture explaining mutex operation at: https://goo.gl/wH9Q44