Introduction to Concurrency

SWEN-220
Therac-25 Medical Accelerator

• **1985-1987** - A radiation therapy device malfunctions and delivers lethal radiation doses at several medical facilities. Based upon a previous design, the Therac-25 was an "improved" therapy system that could deliver two different kinds of radiation: either a low-power electron beam (beta particles) or X-rays. The Therac-25's X-rays were generated by smashing high-power electrons into a metal target positioned between the electron gun and the patient. A second "improvement" was the replacement of the older Therac-20's electromechanical safety interlocks with software control, a decision made because software was perceived to be more reliable.

• What engineers didn't know was that both the 20 and the 25 were built upon an operating system that had been kludged together by a programmer with no formal training. Because of a subtle bug called a "race condition," a quick-fingered typist could accidentally configure the Therac-25 so the electron beam would fire in high-power mode but with the metal X-ray target out of position. At least five patients die; others are seriously injured.

(Source: “History's Worst Software Bugs”, Wired)

Video: [https://www.youtube.com/watch?v=izGSOsAGIVQ](https://www.youtube.com/watch?v=izGSOsAGIVQ)
Lufthansa Airbus A320-200

• **1993** – A Lufthansa Airbus was landing in heavy rain at Warsaw airport in Poland. The plane was not getting any traction on the wet runway, but the pilots knew they could use the reverse thrusters on the main engines to stop the plane. As it happened, the thrust reversers failed to deploy in time and the plane overshot the end of the runway – two people died in the accident.

• Thrust reversers should never be deployed when a plane is in flight. Built in protection looks for these three conditions:
  1. The landing gear must be down.
  2. The wheels must be turning.
  3. The weight of the plane must be carried on the wheels.

• In this case the landing gear was down, but the wheels were hydroplaning, and an unexpected tailwind provided enough lift on the wings that the *control software did not decide* until nine seconds after touchdown that the plane had landed.
Fundamental Problems of Concurrency

• What’s the common thread between these two software engineering disasters?
  – All the individual components of each system were no doubt designed in a very sensible manner.
  – However, they were not designed to cope with the unexpected combination of events that occurred.
  – These types of problems have to do uniquely with the interaction of multiple, concurrently executing components.

• These type of systems are extraordinarily hard to test
  – To test a piece of code, it should be possible to administer a series of reproducible test and evaluate the results.
  – Execution in a concurrent system is beyond the control of testers in a standard testing environment.
  – As a result, some of the most difficult to diagnose bugs can slip through and hide as residual defects in production code.
  – Defects become failures given the “right” sequence of execution.
What Is "Concurrency?"

Two events are said to be concurrent if they occur within the same time interval. Two or more tasks executing over the same time interval are said to execute concurrently. In concurrent programming, there are two basic units of execution: processes and threads.

What Is a Process?

A process has a self-contained execution environment. A process generally has a complete, private set of basic run-time resources; in particular, each process has its own memory space.

What Is a Thread?

Threads exist within a process — every process has at least one. Threads share the process's resources, including memory and open files. This makes for efficient, but potentially problematic, communication.
Thread: a single sequential flow of control within a program

https://docs.oracle.com/javase/tutorial/essential/concurrency/index.html
Subclassing the **Thread** class:

```java
public class SimpleThread extends Thread {
    public SimpleThread(String str) {
        super(str);
    }
    public void run() {
        for (int i = 0; i < 10; i++) {
            System.out.println(i + " " + getName());
            try {
                sleep((long)(Math.random() * 1000));
            } catch (InterruptedException e) {}
        }
        System.out.println("DONE! " + getName());
    }
}
```

```java
public class TwoThreadsTest {
    public static void main (String[] args) {
        SimpleThread jThread = new SimpleThread("Jamaica");
        SimpleThread fThread = new SimpleThread("Fiji");
        jThread.start();
        fThread.start();
    }
}
```

Note that each time we run TwoThreadsTest the output results may differ based on the scheduling of the Jamaica and Fiji threads.
The Promises of Concurrency

• Original (OS centric processes)
  – Better resource utilization.
  – Fairness among multiple users with multiple computations.
The Promises of Concurrency

• Original (OS centric processes)
  – Better resource utilization.
  – Fairness among multiple users with multiple computations.

• Current (process centric threads)
  – Exploiting multiple processors
  – Modeling: Divide & conquer on loosely related tasks.
  – Simplify handling asynchronicity (e.g., mouse events)
  – Throughput (even on single CPU systems)
  – Responsiveness
The Perils of Concurrency

• Safety: Nothing bad happens
  – Incorrect behavior in context of concurrency
  – Race conditions
  – Memory barrier (caching)
  – Overly optimistic compiler optimizations
The Perils of Concurrency

• Safety: Nothing bad happens
  – Incorrect behavior in context of concurrency
  – Race conditions
  – Memory barrier (caching)
  – Overly optimistic compiler optimizations

• Liveness: Good things eventually happen
  – One or more threads cannot make progress
  – Deadlock
The Perils of Concurrency

• Safety: Nothing bad ever happens
  – Incorrect behavior in context of concurrency
  – Race conditions
  – Memory barrier (caching)
  – Overly optimistic compiler optimizations

• Liveness: Good things eventually happen for someone
  – One or more threads cannot make progress
  – Deadlock

• Fairness: Good things eventually happen for everyone
  – Starvation
  – Livelock
The Perils of Concurrency

• Safety: Nothing bad happens
  – Incorrect behavior in context of concurrency
  – Race conditions
  – Memory barrier (caching)
  – Overly optimistic compiler optimizations

• Liveness: Good things eventually happen for someone
  – One or more threads cannot make progress
  – Deadlock

• Fairness: Good things eventually happen for everyone
  – Starvation
  – Livelock

• Performance
  – TANSTAAFL (There Ain’t No Such Thing As A Free Lunch)
  – Context switching overhead
  – Disabled compiler optimizations
The Perils of Concurrency

- **Safety**: Nothing bad happens
  - Incorrect behavior in context of concurrency
  - Race conditions
  - Memory barrier (caching)
  - Overly optimistic compiler optimizations

- **Liveness**: Good things eventually happen
  - One or more threads cannot make progress
  - Deadlock

- **Fairness**: Let's share, boys and girls
  - Starvation
  - Livelock

- **Performance**
  - TANSTAAFL
  - Context switching overhead
  - Disabled compiler optimizations

- **Testing, hair-pulling, and Heisenbugs** (see: Heisenberg)