Concurrency Problem Analysis

Multi-threading is the ability for software to make use of threads to achieve multiple tasks concurrently. It centers on the use of threads - a lightweight piece of running code.

In order to perform multi-threading, take into consideration that design is king. Multi-threading is more harder to debug and assess than single-threaded programming as **emergent** properties. Bugs and problems are not obvious from threads separately, but they do appear on how those threads interact.

Some **problems** include:

- Race condition - lost updates when two threads interact with a resource.
- Deadlocks - threads wait for each other and freeze as a whole.
- Livelocks - threads keep reacting to each other but never actually progress.
- Starvation - one thread keeps hogging resource from the rest
- Heisenbugs - time-sensitive bugs from thread interleaving, disappearing once inspected.

Check Concurrency Bugs to have a detailed explanation.

Rather than dealing with these without thought, create a good outline to tackle them.

Design Analysis

> "Plan for what it is difficult while it is easy, do what is great while it is small"
> - Sun Tzu (probably)

First, cover **high level requirements** first as that will determine what is necessary to program to solve the problem.

Then, consider **low level implementation** afterwards which determine how to code your solution.

Finally, consider **solution scaling** as your programmed solution will now use more computer resources than ever than a usual singular-thread version to solve the problem.

Analysis Guides:

- High Level Analysis - for high level requirements
- Low Level Analysis - for low level implementation
- Scalability Analysis - for solution scaling