2.1 Lecture Summary

2.1 Critical Sections

Lecture Summary: In this lecture, we learned how *critical sections* and the *isolated* construct can help concurrent threads manage their accesses to shared resources, at a higher level than just using locks. When programming with threads, it is well known that the following situation is defined to be a *data race* error — when two accesses on the same shared location can potentially execute in parallel, with least one access being a write. However, there are many cases in practice when two tasks may legitimately need to perform concurrent accesses to shared locations, as in the bank transfer example.

With critical sections, two blocks of code that are marked as isolated, say `A` and `B`, are guaranteed to be executed in mutual exclusion with `A` executing before `B` or vice versa. With the use of isolated constructs, it is impossible for the bank transfer example to end up in an inconsistent state because all the reads and writes for one isolated section must complete before the start of another isolated construct. Thus, the parallel program will see the effect of one isolated section completely before another isolated section can start.

Optional Reading:

1. Wikipedia article on Critical Sections.

2. Wikipedia article on Atomicity.

✓ Complete   Go to next item