CS 350 - Operating Systems **Winter 2018**

Lecture 4: January 15, 2018

Lecturer: Lesley Istead *Notes By: Harsh Mistry*

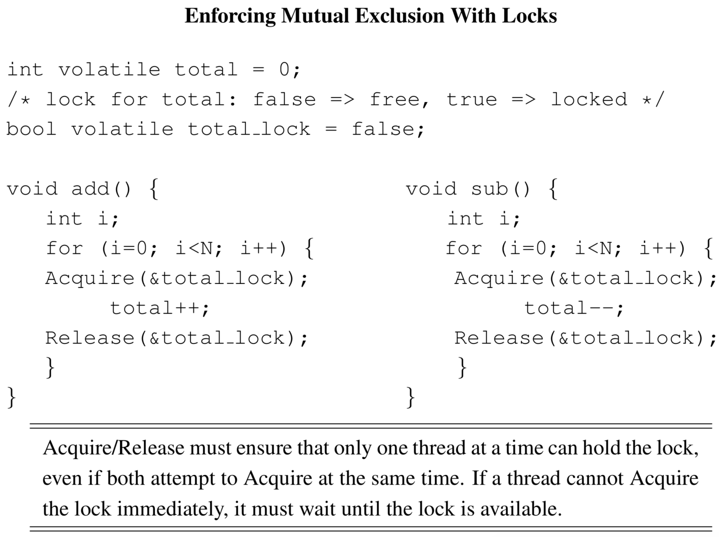
4.1 Synchronization

4.1.1 Thread Synchronization

- All threads in a concurrent program share access to the programs global variables and the heap.
- The part of a concurrent program in which a shared object is accessed is called a critical section.
- To avoid conflicts known as race conditions we can enforce mutual exclusion and use variables
- Using `volatile` keyword forces the compiler to not optimize the code and forcefully load and store teh value of the variable upon every use

4.1.2 Mutual Exclusion

Mutual exclusion is the process of defining a specific area of code that must execute without interruption. This can be achieved through locks.



Enforcing Mutual Exclusion With Locks

```
int volatile total = 0;
/* lock for total: false => free, true => locked */
bool volatile total_lock = false;

void add() {                    void sub() {
   int i;                          int i;
   for (i=0; i<N; i++) {           for (i=0; i<N; i++) {
   Acquire(&total_lock);            Acquire(&total_lock);
        total++;                          total--;
   Release(&total_lock);            Release(&total_lock);
   }                                 }
}                               }
```

Acquire/Release must ensure that only one thread at a time can hold the lock, even if both attempt to Acquire at the same time. If a thread cannot Acquire the lock immediately, it must wait until the lock is available.

Taken from lecture notes

Both Acquire and Release must be atomic functions, therefore, we can not rely on a software solution for efficient locking/releasing. As a result, we must use the CPUs hardware instruction to enforce mutual exclusion