A variable which requires unique access is being used in more than one closure at the same time.

Erroneous code example:

```
fn set(x: &mut isize) {
    *x += 4;
}

fn dragoooon(x: &mut isize) {
    let mut c1 = || set(x);
    let mut c2 = || set(x); // error!

    c2();
    c1();
}
```

To solve this issue, multiple solutions are available. First, is it required for this variable to be used in more than one closure at a time? If it is the case, use reference counted types such as `Rc` (or `Arc` if it runs concurrently):

```
use std::rc::Rc;
use std::cell::RefCell;

fn set(x: &mut isize) {
    *x += 4;
}

fn dragoooon(x: &mut isize) {
    let x = Rc::new(RefCell::new(x));
    let y = Rc::clone(&x);
    let mut c1 = || { let mut x2 = x.borrow_mut(); set(&mut x2); };
    let mut c2 = || { let mut x2 = y.borrow_mut(); set(&mut x2); }; // ok!

    c2();
    c1();
}
```

If not, just run closures one at a time:

```
fn set(x: &mut isize) {
    *x += 4;
}

fn dragoooon(x: &mut isize) {
    { // This block isn't necessary since non-lexical lifetimes, it's just to
      // make it more clear.
        let mut c1 = || set(&mut *x);
        c1();
```

```
    } // `c1` has been dropped here so we're free to use `x` again!
    let mut c2 = || set(&mut *x);
    c2();
}
```