Problem X

We want to implement a version of the Readers and Writers problem such that Semaphores are used completely (all places needed) and efficiently (release semaphores as quickly as possible). It is acceptable to have multiple Readers to the database, but if one process is writing, then no other processes may have access to the database, not even Readers. In our solution, the first Reader to get access controls access to the database, while the last Reader releases access to the database, the variable "Rcount" is used to control this. You can use as many semaphores as you would like... but fix the following code which has no Semaphores, be sure both to declare your semaphores and to initialize them.

Notes:
There is starvation in this solution, but that is not your problem.
You have Create, Initialize, Up, and Down functions for semaphores, so don’t worry about creating these.
int Rcount;
sem Db = 1;
sem Mutex = 1;

void reader(void)
{
    while (TRUE)
    {
        Down(Mutex);     // protect Rcount
        Rcount = Rcount + 1;  // one more reader
        if (Rcount == 1)
        {
            Down(Db)       // only the first Reader needs to get DB
            //--get access to database -- 1st read captures database access
        }
        Up(Mutex);      // Done updating & Checking Mutex
        Read_Data_Base;
        Down(Mutex);    // protect Rcount
        Rcount = Rcount - 1;  // I'm done, one less reader
        if (Rcount == 0)
        {
            Up(Db)       // last reader release DB
            //--release access to database
        }       // last reader releases database
        Up(Mutex);       // Done updating & Checking Mutex
        use-the-data;
    }
}

void writer(void)
{
    while (TRUE)
    {
        think-up-data;
        Down(Db)       // capture database
        write-data-to-database();
        Up(Db)         // release database
    }
}
Issues

- Need 2 Semaphores; one for Rcount, and one for Database. If use one semaphore, then writer() will also be locking processes out of Rcount.
- Must protect all accesses to Rcount, thus the need for protecting the comparisons