

Readers-Writers Problem: Writer needs exclusive access to resource while multiple Readers can access the resource concurrently.

Fairness Problem: If readers continue to access the shared resource process starvation can occur because writers can always be blocked by incoming readers.

Given Code:

```
// Number of readers.
int readcount = 0;
// Mutual exclusion to readcount
Semaphore mutex = 1;
// Exclusive writer or reader
Semaphore w_or_r = 1;

writer {
    wait(w_or_r);
    write();
    signal(w_or_r);
}

reader {
    wait(mutex);
    readcount += 1;
    if(readcount == 1) {
        wait(w_or_r);
    }
    signal(mutex);
    read();
    wait(mutex);
    readcount -= 1;
    if(readcount == 0) {
        signal(w_or_r);
    }
    signal(mutex);
}
```

Code Modification: Additional readers should be blocked once a writer requests access to the shared resource. An additional semaphore is needed to indicate that readers should wait before accessing the resource.

```
// Number of readers.
int readcount = 0;
// Mutual exclusion to readcount
Semaphore mutex = 1;
// Exclusive writer or reader
```

```

Semaphore w_or_r = 1;
// Block new readers after write request
Semaphore w_fair = 1;

writer {
    // A writer waiting to get access
    wait(w_fair);
    wait(w_or_r);
    write();
    signal(w_or_r);
    // Release the lock
    signal(w_fair);
}

reader {
    // Is there are Pending Writer?
    wait(w_fair);

    // We are not a writer so we can signal (this allows more readers to join while no writer)
    // If we move this to the end, readers (and writers) will have to wait each other to finish
    // which is not optimal
    signal(w_fair);
    wait(mutex);
    readcount += 1;
    if(readcount == 1) {
        wait(w_or_r);
    }
    signal(mutex);
    read();
    wait(mutex);
    readcount -= 1;
    if(readcount == 0) {
        signal(w_or_r);
    }
    signal(mutex);
}
}

```