7.8 Read-Write locks

- Support one-writer & many readers
- lockShared() // Reader
- unlockShared() // Reader
- lockExclusive() // Writer
- unlockExclusive() // Writer
- upgrade() // from shared to exclusive
- downgrade() //from exclusive to shared

41

Design consideration

- Avoid needless wake up
- Reader release
  - only the last to wake up a single writer
- Writer release:
  - prefer writer
    - Reader starvation
  - wake up all the readers
    - Writer starvation
    - lockShared() block if there is any writer waiting //new readers

42

Implementation

```
struct rwlock{
int nActive;// number of readers, -1 means a writer is active
int nPendingReads;
int nPendingWrites;
spinlock_t sl;
condition canRead;
condition canWrite;
};
```

43

un/lockShared

```
void lockShared(struct rwlock *r) {
spin_lock(&r->sl);
r->nPendingReads++;
if (r->nPendingWrites>0)
        wait(&r->canRead, &r->sl);
while(r->nActive <0)
        wait(&r->canRead, &r->sl);
r->nActive++;
r->nPendingReads--;
spin_unlock(&r->sl);
}
```

```
void unlockShared(struct rwlock *r) {
spin_lock(&r->sl);
r->nActive --;
if (r-> nActive ==0)     {
    spin_unlock (&r->sl);
    do_signal(&r->canWrite);
} else
    spin_unlock(&r->sl);
}
```

44

un/lockExclusive

```
void lockExclusive(struct rwlock *r) {
spin_lock(&r->sl);
r->nPendingWrites++;
while(r->nActive)
        wait(&r->canWrite, &r->sl);
 r->nPendingWrites--;
r->nActive =-1;
spin_unlock(&r->sl);
}
```

```
void unlockExclusive(struct rwlock *r) {
boolean_t wakeReaders;
spin_lock(&r->sl);
r->nActive =0;
wakeReaders = (r->PendingReads!=0);
spin_unlock(&r->sl);
if (wakeReaders)        {
   do_broadcast(&r->canRead);
else
   do_signal(&r->canWrite);
}
```

45

Up/downgrade()

```
void downgrade(struct rwlock *r) {
boolean_t wakeReaders;
spin_lock(&r->sl);
r->nActive =1;
wakeReaders =
   (r->PendingReads!=0)
spin_unlock(&r->sl);
if (wakeReaders)        {
   do_broadcast(&r->canRead);
}
```

```
void upgrade(struct rwlock *r) {
spin_lock(&r->sl);
if (r->nActive ==1) r->nActive =-1;
else{
r->nPendingWrites++;
r->nActive--;
while (r->nActive)
        wait(&r->canWrite, &r->sl);
r->nPendingWrites--;
r->nActive =-1;
}
spin_unlock(&r->sl);
}
```

46

Using R/W locks

```
rwlock  l;
T1() {
lockShared(&l);
reading;
upgrade(&l)
writing;
downgrade(&l)
unlockShared(&l);
}
```

```
T2() {
lockExclusive(&l);
writing;
downgrade(&l);
reading;
upgrade(&l);
unlockExclusive(&l);
}
```

47