Operating Systems

Assignment 2

180010002

1. For this assignment, we had to familiarize ourselves with pthreads API by writing codes for reader-writer locks. First we had to write the structure that capture the reader-writer lock in ***rwlock.h***, then we had to complete the functions mentioned in the files ***rwlock-reader-pref.cpp*** and ***rwlock-writer-pref.cpp***, the following images show the implemented functions:-

```
struct read_write_lock
{
    int readerCount;
    int writersWaiting;
    bool writerActive;
    pthread_cond_t condVariable;
    pthread_mutex_t lock;
};
```

Figure 1 – structure of reader-writer lock in rwlock.h

```
void InitalizeReadWriteLock(struct read_write_lock * rw)
{
    //  Write the code for initializing your read-write lock.
    rw->readerCount = 0;
    rw->writersWaiting = 0;
    rw->writerActive = false;
    rw->condVariable = PTHREAD_COND_INITIALIZER;
    rw->lock = PTHREAD_MUTEX_INITIALIZER;
}
```

Figure 2 – Suitable initializations of the locks and variables in both cpp files

```
void WriterLock(struct read_write_lock * rw)
{
    //  Write the code for aquiring read-write lock by the writer.
    pthread_mutex_lock(&rw->lock);
    rw->writersWaiting += 1;
    while (rw->readerCount > 0 || rw->writerActive)
        pthread_cond_wait(&rw->condVariable, &rw->lock);

    rw->writersWaiting -= 1;
    rw->writerActive = true;
    pthread_mutex_unlock(&rw->lock);
}

void WriterUnlock(struct read_write_lock * rw)
{
    //  Write the code for releasing read-write lock by the writer.
    pthread_mutex_lock(&rw->lock);
    rw->writerActive = false;
    pthread_cond_broadcast(&rw->condVariable);
    pthread_mutex_unlock(&rw->lock);
}
```

Figure 3 – writer lock and unlock functions in both cpp files

```
void ReaderLock(struct read_write_lock * rw)
{
    //  Write the code for aquiring read-write lock by the reader.
    pthread_mutex_lock(&rw->lock);
    while (rw->writerActive)
        pthread_cond_wait(&rw->condVariable, &rw->lock);

    rw->readerCount += 1;
    pthread_mutex_unlock(&rw->lock);
}

void ReaderUnlock(struct read_write_lock * rw)
{
    //  Write the code for releasing read-write lock by the reader.
    pthread_mutex_lock(&rw->lock);
    rw->readerCount -= 1;
    if (rw->readerCount == 0)
        pthread_cond_broadcast(&rw->condVariable);

    pthread_mutex_unlock(&rw->lock);
}
```

Figure 4 – reader lock and unlock functions in rwlock-reader-pref.cpp

```
void ReaderLock(struct read_write_lock * rw)
{
    //  Write the code for aquiring read-write lock by the reader.
    pthread_mutex_lock(&rw->lock);
    while (rw->writersWaiting > 0 || rw->writerActive)
        pthread_cond_wait(&rw->condVariable, &rw->lock);

    rw->readerCount += 1;
    pthread_mutex_unlock(&rw->lock);
}

void ReaderUnlock(struct read_write_lock * rw)
{
    //  Write the code for releasing read-write lock by the reader.
    pthread_mutex_lock(&rw->lock);
    rw->readerCount -= 1;
    if (rw->readerCount == 0)
        pthread_cond_broadcast(&rw->condVariable);

    pthread_mutex_unlock(&rw->lock);
}
```

Figure 5 – reader lock and unlock functions in rwlock-writer-pref.cpp

We can clearly see from the above images that the writer lock and unlock functions remain unchanged only the **ReaderLock** function is changed. For reader preference, we just removed the writerWaiting condition from the while condition, that was checking if any writer is waiting then we don't allow any reader to enter. So, we just removed that condition for reader preference.

Reference for the pseudocode: Wikipedia