READERS AND WRITERS PROBLEM

pseudocode

THE STRUCTURE OF A WRITER'S PROCESS:

```
while (true) {
wait(rw_mutex);
.../* writing is performed */ ...
signal(rw_mutex);
```

THE STRUCTURE OF A READER'S PROCESS:

```
do {
wait(mutex);
read_count++;
if(read_count== 1)
    wait(rw_mutex);
   signal(mutex);
 /* reading is performed
           */
wait(mutex);
read count--;
if(read_count== 0)
      signal(rw_mutex);
  signal(mutex);
} while (true);
```

starvation

First variation (writer will starve) :
no reader kept waiting unless the writer has permission to use a shared object
Second variation (Reader will starve) :
once a writer is ready, it performs the write ASAP. In other words, if a writer is waiting to access the object, no new readers may start reading.

Both may have starvation leading to even more variations

⇨ Problem is solved on some systems by kernel providing reader-writer locks

Read lock and Write lock which allows a thread to lock the Read Write Lock either for reading or writing.

Read lock: If there is no thread that has requested the write lock and the lock for writing, then multiple threads can lock the lock for reading. It means multiple threads can read the data at the very moment, as long as there’s no thread to write the data or to update the data.

Write Lock: If no threads are writing or reading, only one thread at a moment can lock the lock for writing. Other threads have to wait until the lock gets released. It means, only one thread can write the data at the very moment, and other threads have to wait.

Methods: There are two methods that Read Write lock provides:
Lock readLock()
Lock writeLock()

```java
Lock readLock = rwLock.readLock();

readLock.lock();
try {
   // statements
}
finally {
   readLock.unlock();
}
```

```java
Lock writeLock = rwLock.writeLock();

writeLock.lock();
try {
    statements to write the data
}
finally {
    writeLock.unlock();
}
```

Deadlock

two threads holding reader locks both attempt to upgrade to writer locks, a deadlock is created

The deadlock can be avoided by allowing only one thread to acquire the lock in "readmode with intent to upgrade to write"

Real-world

Banking system: read account balances versus update

Airline reservation system: read available seats in plane and make reservations for the clients