SOLVING READER AND WRITER PROBLEM WITH THE HIERARCHICAL LOCK APPROACH USING SEMAPHORE

THESIS

Presented to the Graduate Council
of Texas State University-San Marcos
in Partial Fulfillment
of the Requirements

for the Degree

Master of SCIENCE

by

Mei Li, B.S.

San Marcos, Texas
December 2004

ACKNOWLEGEMENTS

I am very thankful for the members of my thesis committee: Dr. Haddix, Dr. Chen and Dr. McCabe I would like to especially thank my advisor, Dr Haddix, for his invaluable guidance and help in the preparation of my thesis

Many thanks to my husband and my daughter for their love and support

The manuscript was submitted on Nov 7th, 2004

TABLE OF CONTENTS

LIST OF TABLES

Tables

LIST OF FIGURES

Figures

CHAPTER 1

INTRODUCTION

1.1 Description of Reader and Writer Problem

The reader-and-writer problem is a classic synchronization problem. It was introduced by P. J. Courtois, F. Heymans and D. L. Parnas in 1971[CoHP71]. The problem illustrates that a shared database is accessed by two kinds of processes: readers and writers. The readers execute transactions that examine the database while the writers update the database. A writer must have exclusive access to the elements of the database to be modified but more than one reader may access the database concurrently.

1.2 Description of Semaphore

There have been various proposals for achieving synchronization among concurrent processes. One of them is using inter-process communication primitives that block more than one process entering its critical region where the shared resources such as database are accessed. The pair of SLEEP and WAKEUP is the simplest of these primitives and also the basis for the now ubiquitous semaphores.

In 1965, E. W. Dijksta [Dijk65] suggested using an integer variable to count the number of wakeups saved for future use. A new variable type, called semaphore, was introduced. A semaphore could have the value of 0, indicating that no wakeups were saved, or some positive value if one or more wakeups were pending [TaWo97].

A semaphore has two operations: UP and DOWN (generalizations of Wakeup and Sleep, respectively). The DOWN operation (also known as P operation) on a semaphore checks to see if the value is greater than 0. If so, it decrements the value (i.e. uses a stored wakeup) and continues. If the value is 0, the process is blocked and put to sleep without completing the DOWN for the moment [TaWo97]. The UP operation (also known as V operation) increments the value of the semaphore, if one or more processes are sleeping on that semaphore, unable to complete an earlier DOWN operation, the first process of the waiting processes queued (the one sleeping for the longest period) is chosen by the system and is allowed to complete its DOWN. This is called *blocked_queue* semaphore. The group of blocked processes is maintained in a queue and a process that blocks is placed at the end of the queue while a process at the head of the queue is selected for execution. The *blocked_queue* semaphore is much different than a *weak_semaphore*, which may be implemented with "test and set" instruction. Instead of putting a blocked process in a queue, the *weak_semaphore* lets the process execute a busy waiting loop in which the value of semaphore is continuously tested. Once a process checked the value of semaphore is greater than 0, it completes its DOWN operation and enters the critical section. The blocked process is chosen to complete a DOWN operation in a random order [EuSt82].

Semaphores can be used to solve synchronization problems. Checking the value, changing it, and possible going to sleep is all done as a single, indivisible, atomic action. System support is usually provided to guarantee atomicity of semaphore operations. It is guaranteed that once a semaphore operation has started, no other process can access the semaphore until the operation has completed or blocked [TaWo97].

In this research, we will solve the reader and writer problem using the *blocked_queue* semaphore.

1.3 Related Work

Since 1960's, the reader and writer problem has been extensively studied and researchers have discovered several solutions using semaphores. Three of these traditional approaches are reader privilege, writer privilege and fair reader and writer. P.J. Courteous, F.Heymans and D.L.Parnas first introduced reader privilege and writer privilege approaches in 1971[CoHP71]. For the reader privilege approach, the readers have higher priority than the writers. So if the readers keep coming in, the writers may never get a chance to access the database. This is called writer starvation when that happens. For the writer privilege approach, the writers have higher priority than the readers and thus reader starvation may occur. Neither of these are optimum solutions. A better approach for most applications is the fair reader-and-writer approach [Hart03]. An outer semaphore, for which each reader and writer waits is added in addition to the traditional semaphores. The solution guarantees FIFO ordering for the read/write requests, while maintaining exclusive write access and shared read access. Even though this is a fairly good solution it still suffers from an efficiency problem. For example, if requests RRWRRW (R means reader request and W means writer request) are queued on the outer semaphore and they arrive at almost the same time (In real world, this is a likely occurrence) the first RR pair can enter the critical section to read concurrently, but the second RR pair must wait for the first W to finish even though they arrived in the queue almost at the same time as the first RR. This meets a strong fairness criterion but is not the most efficient approach. The fourth approach we will explore is to use an enhanced algorithm for the reader-and-writer problem. The enhanced algorithm lets all sharing processes pass the outer semaphore if they arrive during the same lock arbitration. In the above sequence, RRWRRW can all pass the outer semaphore as long as they arrive during the same lock arbitration. Thus, the four R's can access the shared resource at the same time, but the first W will be blocked by inner semaphore and has to wait for all four R's to finish their job. This strategy not only maximizes the number of concurrent readers but also gives the writer a fair chance. The idea is that allowing readers to "pass" writers in the queue doesn't penalize the writers since the reader accesses overlap because of shared access. In this scenario, the readers start their accesses simultaneously, and ideally, their accesses would complete contemporaneously.

Because of the inherent hierarchical structure of databases, (in our model, a database consists of a collection of tables, with each table holding multiple rows, or records), restricting all clients from examining or updating the data in a database because one client is updating a row is costly and inefficient. We propose to use intent to read and intent to write locks [OMGI00]. For example, to read a record in the database, the client obtains an intent to read lock from the ancestor(s) of the requested resource, a read lock for the record, and then reads the record. Another client comes and wants to write to another record. It obtains an intent to write lock from the ancestor(s) of the other record. Even if the two records have the same ancestors, the request will be granted because the intent to read lock is not in conflict with the intent to write lock The two clients can do their jobs concurrently without interfering with each other, as long as they are operating on separate records. Using the intent to read and intent to write locks should improve concurrency and efficiency for an application or suite of applications utilizing the same resources.

Even though the above algorithm improves efficiency, deadlock can still occur. When two processes read concurrently and then attempt to write without releasing the read locks, deadlock may result. To solve this problem, we propose to use an upgrade semaphore as an upgrade lock [OMGI00]. If a process has to obtain an upgrade lock before it writes to the shared resource following a read, deadlock can be avoided. To avoid deadlock with the other algorithms, a write lock must be obtained for the duration of the entire update transaction, thus reducing sharing and concurrency.

1.4 Objective

We will compare the traditional approaches, our enhanced approach, and the hierarchical and sequential lock approaches for the reader-and-writer problem using semaphores. To accomplish this, we will use intent to read, intent to write, and upgrade locks. We will show that this approach has three characteristics: fairness, efficiency and freedom from deadlock.

1.5 Overview of the Thesis

The thesis consists of six chapters.

Chapter 1 - Introduce the thesis background, definition of terms, aims of research and outline of the remaining chapters.

Chapter 2 - Specify the six algorithms used to solve the reader and writer problem and provide a rationale for their correctness.

Chapter 3 - Specify the design of the experiment.

Chapter 4 - Describe the implementation.

Chapter 5 - Discuss and analyze the test results.

Chapter 6 - Conclusions and future work.

CHAPTER 2

DESCRIPTION OF THE ALGORITHMS

In order to solve the reader and writer problem properly, we must consider the important properties of the problem. The properties we are interested in are fairness, liveness, and efficiency. Fairness means no starvation. Neither readers nor writers can be starved and each read/write request should be handled in some approximation of FIFO order. Liveness means no deadlock. In other words, at each point in time, some process must be capable of executing. The third property is efficiency. Efficiency is obtained by allowing some processes to share access. With regard to efficiency, we must weigh algorithm efficiency against the amount of sharing obtainable. Fairness and liveness will be considered as properties to be examined heuristically. Efficiency will be addressed experimentally. A total of six algorithms are considered. The first four algorithms address two classes of privileges, namely, exclusive (writer) and shared (reader) by specifying different conditions for acquiring the privilege of critical section execution. The last two consider additional classes of privileges, namely hierarchical (intention privileges) and sequential (read followed by write).

2.1 Reader Privilege [CoHP71]

2.1.1 Description of the Algorithm

This solution supports a strong reader, providing that no reader be kept waiting unless a writer has already obtained permission to use the database. No reader should wait simply because a writer is waiting for other readers to finish [CoHP71]. In effect, this allows readers to cut in front of writers. For high contention resources, this defers updates and other maintenance activities to periods of low demand.

In this algorithm, semaphore *rc* is used to protect the counter of reader and semaphore *w* is a mutual exclusion semaphore for the writer and the first reader to enter the critical section. Readers that enter or leave while other readers are present ignore semaphore *w*. Semaphore *rc* ensures that only one reader will enter or leave at a time thereby eliminating the possibility of ambiguity about which process is responsible for the counter. Semaphore *w* will be positive if and only if there are no readers and no writers present in the critical section.

Pseudo code for this algorithm is provided in Figure 2.1.

```
Integer readcount; (initial value = 0)
Semaphore rc       // Semaphore for readcount (initial value = 1)
Semaphore w        // Semaphore for mutual exclusion (initial value = 1)
```

READER	**WRITER**
P(rc); readcount= readcount + 1; If readcount = 1 then P(w); V(rc);	
....	P(w);
reading is performed	writing is performed
.... P(rc); readcount = readcount – 1; If readcount = 0 then V(w); V(rc);	.... V(w);

Figure 2.1: Solution for Reader and Writer Problem - Strong Reader

- Reader procedure:
  - The reader requests semaphore *rc* and exclusively updates the count. If it is the first reader, (*count = 1*), it requests semaphore *w*, thereby blocking writers.
  - It releases *rc* and allowing other readers to obtain access.
  - Reading is performed.
  - The reader requests semaphore *rc* and exclusively updates the *count*. If it is the last reader (*count = 0*), it releases semaphore *w*, thereby unblocking the writers. In all cases, it releases *rc*.
- Writer procedure:

- Writer requests semaphore *w* to block readers (and other writers).
- Writing is performed.
- When the writer finishes the job, it releases semaphore *w*, thereby unblocking the readers (and other writers).

2.1.2 Discussion of Correctness – Reader Privilege

Fairness:

In this algorithm, because a reader can block the mutual exclusive semaphore *w* when *readcount* is equal to 1, all subsequent writers are blocked. Writers are blocked as long as there is a reader who is reading. Readers can continue to enter the critical section as long as there is another reader reading. Indeed, when there are multiple readers, the same reader can read multiple times while writers continue to wait. Writers can access the database only if the last reader finishes the job and exits the database and no reader is waiting. If the readers keep coming in, the writers may never get a chance to access the database. Obviously, this solution favors readers and potentially starves writers. Because of this unfairness, this algorithm is not suitable for many applications.

Liveness:

The only points of possible blocking are at the simple semaphores. So long as processes behave correctly, and *readcount* and the semaphores are initialized correctly, blocking can never occur. Trivially, the writer releases semaphore *w* whenever a grant is received. With the readers, it is also straightforward: Each grant of semaphore *rc* is followed by a release. Similarly, assuming that *readcount* is initialized correctly, the number of increments will be equal to the number of decrements, and thus, each time, a grant on semaphore *w* is received, it will be released.

2.2 Writer Privilege [CoHP71]

2.2.1 Description of the Algorithm

This solution supports a strong writer, by providing that no writer should be kept waiting unless a reader has already obtained permission to use the database. No writer should wait simply because a reader is waiting for a writer to finish. [CoHP71]

In this algorithm, three more semaphores have been added. The use of *rc* and *w* corresponds exactly to the use of *rc* and *w* in the solution of Figure 2.1. The semaphore *r* is used to protect the act of entering the critical section in the same way that *w* is used to protect the shared resource in Figure 2.1. The first writer to pass semaphore *r* will block readers from entering the section to manipulate *rc* and *w*. *wc* is used to protect the writer count. Semaphore *pr* guarantees the priority of writers. Without *pr* we have the possibility that a writer and one or more readers will be simultaneously waiting for a *V(r)* to be done by a reader. In that case, the priority of writer could not be guaranteed. [CoHP71] Pseudo code for the algorithm is provided in Figure 2.2.

```
Integer readcount, writecount; ( initial value = 0)
Semaphore pr        // pre_read ( initial value = 1)
Semaphore r         // read ( initial value = 1)
Semaphore rc        // readcount ( initial value = 1)
Semaphore wc        // writecount ( initial value = 1)
Semaphore w         // mutual exclusion( initial value = 1)
```

READER

```
P(pr);
    P(r);
      P(rc);
      readcount = readcount + 1;
      if  readcount = 1 then P(w);
      V(rc);
    V(r);
V(pr);
  ...
reading is performed

  ...
P(rc);
readcount = readcount – 1;
if readcount = 0 then V(w);
V(rc);
```

WRITER

```
P(wc);
writecount = writecount + 1;
  if writecount = 1 then P(r);
V(wc);
P(w);

...
writing is performed

...
V(w);
P(wc);
writecount = writecount  - 1;
  if writecount = 0 then V( r );
V(wc);
```

Figure 2.2: Solution for Reader and Writer Problem - Stronger Writer

- Reader procedure
    - Request semaphore *pr*. Only one reader can come in at a time.
    - Request semaphore *r*. Readers are restricted by the semaphore.
    - Request semaphore *rc* to access the *readcount* exclusively.
    - If it is the first reader, request semaphore *w* and block writer.
    - Release *rc* on *readcount*.
    - Release semaphore r to let other readers come in.

- Release semaphore *pr*.
- Reading is performed.
- If it is the last reader, release semaphore *w*.
- Release semaphore *rc* on *readcount*

- Writer procedure
  - Request *wc* on *writecount*.
  - If it is the first writer, request semaphore *r* to block reader. This is where writer has higher priority because it blocks reader before waiting on semaphore *w*.
  - Release *wc* on *writecount*
  - Request *w* and try to write.
  - Writing is performed
  - Release semaphore w to another writer.
  - Request *wc* on *writecount*.
  - If it is the last writer, release r to unblock readers
  - Release *wc* on *writecount*.

2.2.2 Discussion of Correctness – Writer Privilege

Fairness:

In this solution, a reader is blocked by semaphore *r* whenever there is a writer in its critical section or waiting for its critical section. Because a writer only needs to wait on semaphore *w*, once a writer is finished, *w* will be released to other writers without allowing a reader an opportunity even though a read request occurs first. Readers may be starved if sequences of writers wish to update the database. Indeed, under the right conditions, a writer may write multiple times while a reader is waiting. Because of the unfairness of the algorithm, it is not appropriate for many applications.

Liveness:

As long as each semaphore, *readcount* and *writecount* are initialized correctly and each grant to a process is followed by a release, deadlock can never occur In this algorithm, each request of a semaphore is followed by a release and the number of increments of *readcount* and *writecount* is equal to the number of decrements. Thus, deadlock can never occur during correct execution.

2.3 Fair Reader and Writer [Hart03]

2.3.1 Description of the Algorithm

In this algorithm, semaphore *pw* which each reader and writer waits on has been added. This is what guarantees the FIFO ordering for read/write requests [Hart03]. Because of the FIFO character of this algorithm, we call this algorithm Fair Reader and Writer.

Pseudo code for the algorithm is provided in Figure 2.3.

```
Semaphore pw      // pre_mutual exclusion(initial value = 1).
Semaphore rc      // read_count(initial value = 1).
Semaphore w       // mutual exclusion(initial value = 1).
Integer readcount // (initial value  = 0).
```

READER	**WRITER**
P(pw);	P(pw);
P(rc);	P(w);
readcount = readcount + 1;	
if readcount = 1	
P(w);	V(pw);
V(rc);	
V(pw);	
reading is performed	writing is performed
....	.....
P(rc);	
readcount = readcount – 1;	
if readcount = 0	
V(w);	
V(rc);	V(w);

Figure 2.3 Solution for Reader and Writer Problem - Fair Reader and Writer

- Reader procedure
    - Request semaphore *pw* to enter the queue. If a reader waits on the semaphore first, the reader gets a permission to manipulate mutual exclusion semaphore *w* first.
    - Request semaphore *rc* on *readcount.*
    - If it is the first reader, request semaphore *w* to block writer (any readers immediately behind this one will be waiting on *pw*).
    - Release semaphore *rc* on *readcount.*

- Release semaphore *pw* to allow waiting read/write processes enter to queue.
- Reading is performed.
- Request semaphore *rc* on *readcount*.
- If it is the last reader, release semaphore w to writer.
- Release semaphore *rc* on *readcount*.

* Writer procedure
  - Request semaphore *pw* to enter the queue.
  - Request semaphore *w* to write.
  - Release semaphore *pw* to other read/write requests.
  - Writing is performed
  - Release semaphore *w* for next reader or writer.

2.3.2 Discussion of Correctness – Fair Reader and Writer

Fairness:

Assume we have a sequence read/write requests RRWR(R means read request and W means write request). The first reader passes the semaphore *pw* without any problem, then request semaphore *rc* and update the *readcount*. Because it is the first reader, it blocks writer by obtaining grant on semaphore *w*. Then it unblocks *rc* and *pw* and read from the database. The second reader can get in at this time. The second reader first locks the *rc* to update the count, because the count is not 1, skip the if statement. Then the second reader releases *rc* and *pw* to let the first process at the front of the queue to enter so that the second reader can then read concurrently with the first reader. A write request passes semaphore *pw* but is blocked by semaphore *w* since the readers are still reading in database. The writer has to wait until the readers finish. The reader immediately after W has to wait until the writer is unblocked at semaphore *w*. As described above, the algorithm can guarantee FIFO while maintaining the exclusive write access and shared read access. This is a fair and correct algorithm.

Liveness:

This algorithm is a little different from the reader privilege by adding the outer semaphore *pw*. As long as each reader and writer request grants to enter the outer semaphore *pw* first, then release it before they enter the critical section, deadlock can be avoided.

2.4 Fair and Efficient Readers and Writers

2.4.1 Description of the Algorithm

In this algorithm, the outer semaphore *pw* is the same as the one in the Figure 2.3. All readers can read concurrently as long as they have previously passed the outer semaphore *pw.* If a reader is blocked by this semaphore, it can not read even though other readers are in the critical section. This algorithm is fair and efficient because it allows all read requests to pass the outer semaphore during the same lock arbitration. Thus the strategy maximizes the read concurrency while maintaining approximate FIFO fairness. The principal difference between this algorithm and the Fair Readers and Writers algorithm (the last algorithm) is that when a reader obtains the privilege all currently waiting readers are allowed to enter their critical section rather than only those who immediately succeed the first reader in the queue. If all readers are concurrent and of equal duration, writers will write at the same time or sooner than in the fair readers and writers algorithm. If all readers are not completely concurrent, some writers may experience small delays compared to the fair readers and writers algorithm. It maintains a high degree of fairness in that readers entering the queue after the first reader obtains the privilege are barred from executing their critical sections by semaphore *pw*.

Pseudo code for the algorithm is provided in Figure 2.4.

```
Semaphore pw      // pre_mutual exclusion(initial value = 1)
Semaphore rc      // read_count(initial value = 1).
Semaphore w       //mutual exclusion (initial value = 1).

Integer readcount( initial value is 0)
```

```
READER                                  WRITER

P(pw);                                  P(pw);
V(pw);                                  V(pw);
 P(rc);
    readcount++;
 if readcount = 1
    {
        P(w);
        P(pw);
    }                                    P(w);

 V(rc);

reading is performed                    writing is performed
...                                     ...
 P(rc);
   readcount --;
 if readcount = 0
    {
       V(pw);
       V(w);
    }                                   V(w);
 V(rc);
```

Figure 2.4: Solution for Reader and Writer Problems - Fair and Efficient Reader and Writer

- Read procedure
  - Reader requests semaphore *pw*.
  - Reader releases semaphore *pw*. This allows all read/write requests after the reader to come in.
  - Request semaphore *rc* on *readcount*.

- If it is the first reader, request *w* to block writer then request *pw* to block all processes that are ready to pass the outer semaphore *pw*.
- Release semaphore *rc* on *readcount*.
- Reading is performed
- Request semaphore *rc* on *readcount*.
- If it is the last reader, unblock *pw* to let other processes come in then release *w* to unblock writer.
- Release semaphore *rc*.

• Write procedure
  - Request semaphore *pw*.
  - Release semaphore *pw*. This allows all read/write requests after the writer to come in.
  - Request semaphore *w* to block readers.
  - Writing is performed.
  - Release semaphore *w* to unblock other read/write request.

2.4.2 Discussion of Correctness- Fair and Efficient Reader and Writer

Fairness:

This algorithm is fair and efficient because it allows all processes that arrive while a read process is in lock arbitration to enter their critical sections with that first read process. While a read process is waiting for a grant on *w*, all processes will pass semaphore *pw*. As soon as a read process requests semaphore *w*, it requests semaphore *pw*, thus blocking all new requests from entering the lock arbitration. Once, the sharing readers complete their critical sections, semaphore *pw* are raised, allowing all requests to enter lock arbitration until another reader obtains a grant on *w*. In this manner, reading concurrency is maximized while maintaining fairness for writers.

A strong fairness criterion might provide that all requests are granted in the order that they reach the request arbiter. Some of the algorithms discussed here have strong fairness for one class of users, but allow another class to experience livelock. The fairness criterion of the fair and efficient algorithm is to increase shared access without significantly penalizing any processes versus the results under the strong fairness criterion.

Liveness:

This algorithm is similar to the fair reader and writer algorithm. As long as all processes request and release the outer semaphore and mutual exclusive semaphore properly, deadlock will not occur.

Efficiency:

Consider the requests RRWR, where R means read request and W means write request. This sequence will require four time units to complete (assuming each request requires one time unit). If the three Rs can read concurrently, only two time units are required to complete the four requests. In the above case, the fair readers and writers would require three time units to complete, while W would still have to wait one time unit for its turn. This algorithm is an enhanced and improved solution of previous one. In practice, reading data from a database usually occurs more frequently than writing data into a database, so the efficiency of the reading procedure is very important. This algorithm is correct as well as efficient. This algorithm has the nice property of increasing concurrency among shared access requests when contention is higher.

2.5 Fair and Efficient Readers and Writers with Intent to Read and Write

2.5.1 Purpose of Intent to Read and Write Locks

This algorithm adds intentional locks[OMGI00] to the Fair and Efficient Readers and Writers algorithm. Intentional locks are relevant if there exists a hierarchical locking relationship such as the inherent relationship in a database or file system. A database contains multiple tables, each of which contains multiple rows. Similarly, a file directory contains multiple files and each file consists of multiple records. If we lock a whole database because somebody is updating only one row in a table, the cost of restriction in terms of accessing the database is huge and will significantly reduce concurrency. On the other hand, if we set locks for each table or each row, it will result in a higher locking overhead. In order to balance between the lock overhead and the degree of concurrency, we use intent to read and write locks [OMGI00].

2.5.2 Description of Using Intent to Read and Write Locks

When using intention locks to access a hierarchy, the order in which locks are acquired is always from the top down. To read a record in the database, for example, the client obtains intent to read lock (IR) on the database and the table (in this order) before obtaining the read lock(R) on the record. Intent to read locks (IR) conflict with write locks (W), and intent to write locks (IW) conflict with read(R) and write (W) locks; however, intent to read and intent to write locks do not conflict with each other, allowing many concurrent locks within a database [IONA01]. When a mass read or write is to take place, the possibility of locking the larger resource is still available.

2.5.3 Description of the Algorithm

In this algorithm, semaphores *rc(0)* and *rc(1)* have been added. Semaphores *pw* and *w* have the same functionality as those in Figure 2.4. *rc(0)* and *rc(1)* have similar functionality to *rc* in Figure 2.4. All R, IR, and IW requests go through either *rc(0)* or *rc(1)*. Each of *rc(0)* and *rc(1)* can be of type IR, R, or IW. The primary flag is 0 for rc(0) and 1 for rc(1). The first process that enters the primary semaphore will give the semaphore its type. The next process, if compatible with the primary type, enters the primary semaphore, updates the primary count and if necessary, updates the primary type. If the value of count is equal to 1, it requests semaphore *w*, and then *pw*. The next process, if not compatible with the primary type, enters the secondary semaphore, updates the secondary count, and if necessary, updates the secondary type. Recall that IR is compatible with R, IW and itself; R is compatible with IR and itself; IW is compatible with IR and itself [OMGI00].

These compatibilities are defined in Table 2.1.

Table 2.1. Compatibility of Requests with Existing Locks (X indicates incompatibility)

Requested Lock	Previous Grant			
	IR	R	IW	W
Intention Read (IR)				X
Read (R)			X	X
Intention Write(IW)		X		X
Write (W)	X	X	X	X

2.5.4 IR, R and IW Semaphore Upgrades and Additions

Table 2.2 illustrates IR, R and IW semaphore upgrades and additions.

Table 2.2. Request Additions and Type Updates

Primary Type	Addition Type	Addition Semaphore	Addition Semaphore Type Update
None	IR	Primary	IR
None	R	Primary	R
None	IW	Primary	IW
IR	IR	Primary	IR
IR	R	Primary	R
IR	IW	Primary	IW
R	IR	Primary	R
R	R	Primary	R
R	IW	Secondary	R
IW	IR	Primary	IW
IW	R	Secondary	IW
IW	IW	Primary	IW

2.5.5 The Fair and Efficient Algorithm with Intent to Read and Write

Pseudo code for the algorithm is provided in Figures 2.5 and 2.6.

Global Variables:

```
Semaphore pw, w, rc(0), rc(1) //  Initial value all 1
Integer smp           // indicates the value of current semaphore, initial value is 0
Integer count [2]     // counter for share semaphores rc(0) and rc(1), initial value is 0
Enum typ[2]           // type for share semaphores rc(0 )and rc(1)( IR, IW, or R), initial
                      // value is IR, R = read, IR = Intent to Read, IW = Intent to Write
Integer prm           // indicates which of share semaphores is primary, initial value is 0
```

Local Variables:

```
Integer smp           // identifies current process's rc() semaphore
```

READER / **WRITER**

```
P(pw),                                   P(pw);
V(pw)                                    V(pw),
If(typ(prm) != IW)
  smp = prm;
else
  smp = 1 – prm,

P(rc(smp));
  count[smp] ++;
    If(typ(smp) = IR)
        typ(smp) = R,
    If(count(smp) = 1)
     {
          P(w),                          P(w),
          P(pw),
     }
V(rc(smp));
reading is performed                     writing is performed

P(rc(smp));
  count[smp] --,
  If(count(smp) = 0)
    {
    typ(smp) = IR,
    prm = 1 – prm;
    V(pw);
    V(w);                                V(w);
    }
V(rc(smp)),
```

Figure 2.5: Reader and Writer Algorithms – Fair and Efficient Readers and Writers with Intent to Read and Write

INTENT TO READ

```
P(pw);
V(pw);
smp = prm;
P(rc(smp));
count[smp] ++,


If(count[smp] = 1)
  {
     P(w);
     P(pw),
  }
V(rc(smp));
```

Intent to read operations, including requesting locks on lower resources

```
P(rc(smp));

count[smp] --,


if( count[smp] = 0)
{
 typ(smp) = IR;
 prm = 1 – prm;
 V(pw);
 V(w);


 }
V(rc(smp)),
```

INTENT TO WRITE

```
P(pw),
V(pw),
if(typ(prm) = IW || typ(prm) = IR)
    smp = prm,
else
    smp = 1 – prm,

P(rc(smp)),
count[smp] ++,
if(typ(smp) = IR)
    typ(smp) = IW,
if(count[smp] = 1)
 {
   P(w),
   P(pw),
  }


V(rc(smp)),
```

Intent to write operations, including requesting locks on lower resources

```
 .
P(rc(smp)),
count[smp] --,

if(count[smp] = 0)
 {
  typ(smp) = IR,
  prm = 1 – prm,
  V(pw);
  V(w),
  }
 V(rc(smp)),
```

Figure 2.6: Intent to Read and Write Algorithms - Fair and Efficient Readers and Writers with Intent to Read and Write

- Reader procedure
  - Request and release the outer semaphore *pw*.
  - If the type of primary semaphore is not *IW*, assign the primary semaphore to variable *smp*. Otherwise, assign the secondary semaphore to variable *smp*.
  - Request the semaphore assigned in *smp* and update *count*. If the type of the primary semaphore is *IR*, upgrade it to *R*.
  - If *count* is 1, request semaphore *w* to block the writer and request semaphore *pw* to block the processes that are waiting to enter lock arbitration.
  - Release semaphore indicated by *smp* on *count*.
  - Reading is performed
  - Request the semaphore indicated by *smp* again to update *count*.
  - If it is the last shared access operation, reinitialize its primary type. Update the primary flag to the other *rc(0)/rc(1)* semaphore. Release *pw* and *w*.
  - Finally, release the semaphore indicated by *smp*

- Write procedure
  - Request semaphore *pw*.
  - Release semaphore *pw*. This allows all read/write requests after the writer to come in.
  - Request semaphore *w* to block readers

- Writing is performed.
- Release semaphore *w* to unblock other read/write requests.

* Intent to read procedure
  - Request and release semaphore *pw*.
  - Assign the primary semaphore to variable *smp*.
  - Request the semaphore indicated by *smp* to update the count
  - If *count* is 1, request w and *pw*
  - Intent to read operation
  - Request the semaphore indicated by *smp* again to update the count
  - If it is the last shared access operation, update the primary flag to point to the other *rc(0)/rc(1)* semaphore. Then release *pw* and *w*.
  - Release the semaphore indicated by *smp*.

* Intent to write procedure
  - Request and release semaphore *pw*.
  - If the type of the primary semaphore is *IW* or *IR*, assign the primary semaphore to *smp* because *IW* is compatible with *IR* and itself. Otherwise, assign the secondary semaphore to *smp*.
  - Request the semaphore pointed to by *smp* and update *count*. If the type of *smp* is *IR*, change it to *IW*.

- If the count of sharing semaphore is equal to 1, request *w* to block other processes from entering critical section then request the outer semaphore *pw*.
- Release semaphore *smp* on the count of sharing semaphore.
- Intent to write operations.
- Request semaphore *smp* to update *count*.
- If it is the last shared access operation, downgrade the type to *IR* and change the primary to the opposite.
- Release the semaphores *pw* and *w*.
- Release the semaphore pointed to by *smp*.

2.5.6 Discussion of Correctness – Fair and Efficient Readers and Writers with Intent to Read and Intent to Write

Fairness:

The policy in this algorithm is to maximize concurrency amongst compatible privileges (Refer to Table 2.1). Thus, at any point in execution, there are possibly three classes of incompatible requests. These are write requests (which, in addition are mutually incompatible), read requests, and intent to write requests. Intent to read requests are compatible with both read and intent to write. Thus, there are two classes of shared access requests. In a maximal case, the mutual exclusion lock queue would include one or more write requests (*W*), a read request (*R*), and an intent to write (*IW*) request. The *R* and *IW* requests are proxies for possibly multiple requests and the primary request might have been made as the result of an intent to read (*IR*) request. Assuming that a write request has the privilege, additional requests entering would go into a queue. Assume that the *R* request is primary, then *R* and *IR* requests would go into the primary *rc* queue, *IW* requests would go in the secondary *rc* queue, and *W* requests would go into the *w* queue. Once, in the *rc()* queues or the w queue, a request's place in execution order is assured. Requests entering the *rc()* queues could still execute prior to *W* requests entering lock arbitration before them. However, the *W* requests should not be significantly delayed. Once, a *W* request enters lock arbitration, a process sending a shared access request afterward could receive at most two shared access grants before the *W* request is granted.

Liveness:

This algorithm is fair and efficient reader and writer algorithm with intent to read and intent to write lock. As long as all processes request and release the outer semaphore *pw*, *rc(0)* and *rc(1)* semaphores and mutual exclusive semaphore *w* properly and increase and decrease the count of *rc(0)* or *rc(1)* correctly, deadlock will not occur.

Efficiency:

In this algorithm, intent to read and intent to write privileges are added in order to implement a hierarchical lock scheme. Using a hierarchical lock can maximum the concurrency of accessing a database. Semaphores *rc(0)* and *rc(1)* allow *IR*, *IW* and *R* processes to execute their critical sections in parallel and thereby improve concurrency and efficiency. Thus, this algorithm is fair and very efficient.

This algorithm has the nice property of increasing concurrency among shared access requests when contention is higher.

2.6 Fair and efficient readers and writers with intent to read, intent to write, and upgrade locks.

The algorithm adds an upgrade lock [OMGI00] to the fair and efficient readers and writers with intent to read and intent to write locks.

2.6.1 Description of the Purpose of Upgrade Lock

Because read/write locking allows multiple readers but only one writer to access a resource, it is possible to create a deadlock. The situation happens if two or more transactions attempt to first read a resource then later write the same resource without releasing the read locks [IONA01]. For example, there are two transactions *T1* and *T2* that both are reading concurrently. Later on, *T1* wants to update the resource but it is blocked because *T2* is still reading. Later on, *T2* wants to update the resource but is also blocked because *T1* has not released its read lock yet. Neither *T1* nor *T2* can proceed and since both are waiting on the other to release the read lock deadlock occurs. One way of dealing with this problem is to add upgrade locks. If each process acquires an upgrade lock before updating a resource, deadlock can be avoided [IONA01], while concurrency is possible between the upgrade lock and read or intention to read locks. Without an upgrade lock, a write lock must be obtained at the beginning of the transaction. This prevents any concurrency with other lock.

2.6.2 Description of Using Upgrade Lock

An upgrade lock is similar to read lock except that it conflicts with itself. Table 2.3 shows the conflict matrixes for read, write and upgrade locks [IONA01]. (X indicates conflict).

Table 2.3.Conflict Matrix for Read, Write and Upgrade Locks (X indicates Conflict)

Request Mode	Granted mode		
	R	U	W
Read(R)			X
Upgrade(U)		X	X
Write(W)	X	X	X

For example, there are two transactions T1 and T2 that are reading concurrently. Later, T1 wants to update the resource. It obtains an upgrade lock first without any problem because an upgrade lock does not conflict with a read lock. Later on, T2 attempts to acquire the upgrade lock but it is blocked. T1 proceeds to acquire a write lock. After T1 release its write lock, T2 is granted the upgrade lock and eventually acquires a write lock [IONA01].

2.6.3 Description of the Algorithm

In this algorithm, we add three more semaphores: *rc(2)*, *u* and *pu* in addition to the semphores *pw*, *w*, *rc(0)* and *rc(1)* used in Figures 2.5 and 2.6. All upgrade requests must actively compete for a grant to pass the *u* semaphore. All requests must pass the outer semaphores *pu* and *pw*. All *IR*, *R*, *IW* and *U* requests must pass one of the *readcount* semaphores *rc(0)*, *rc(1)*, or *rc(2)*. Upgrade requests must pass the *u* semaphore before releasing the *pu* semaphore. A second upgrade request will not pass the *u* semaphore, and thus will block all subsequent requests on semaphore *pu*. There is a significant issue of fairness between upgrade and write locks. The philosophy of the "fair and efficient" algorithms is to maximize concurrency between shared access requests. With an upgrade, the effect could be to allow an upgrade's write to take place before a write (without upgrade), which had been waiting longer. This algorithm requires an upgrade to obtain its own exclusive lock, but allows waiting reads, and intents to read to share the lock. The *pu* semaphore assures that at most one upgrade request is "in" arbitration at a time.

Pseudo code for this algorithm is provided in Figures 2.7, 2.8 and 2.9.

Global variables:

```
Integer count[3]     // counter for share semaphores rc(0), rc(1)and rc(2)
Char typ[3]          // type for share semaphores rc(0), rc(1), rc(2) ( IR, IW, R, or U)
Integer prm          // indicates which of share semaphores is primary(rc(0), rc(1),rc(2))
Semaphore pu         // pre_upgrade, initial value = 1
Semaphore u          // upgrade, initial value = 1
Semaphore pw         // pre_mutual exclusive, initial value = 1
Semaphore w          // mutual exclusive, initial value = 1
Semaphore rc(0), rc(1), rc(2) // initial value = 1
```

Local variable

```
Integer smp          // identifies current process's rc() semaphore;
```

READER

```
P(pu);
V(pu),
P(pw),
V(pw),
If(typ(prm) != IW)
    smp = prm;
else
    smp = (prm + 1) mod 3;
P(rc(smp)),
count(smp) ++;
if(typ(smp) = 1)
  {
     P( w ),
     P( pw ),
  }
V(rc(smp)),

reading is performed

P(rc(smp)),
count(smp)--,
if( count(smp) = 0)
  {
     typ(smp) = IR;
     prm = ( prm + 1) mod 3;
     V(pw),
     V(w),
  }
V(rc(smp)),
```

WRITER

```
P(pu),
V(pu);
P(pw),
V(pw),

P(w),

writing is performed

V(w);
```

Figure 2.7 Reader and Writer Algorithms - Fair and Efficient Readers and Writers with Intent to Read, Write and Upgrade Lock

INTENT TO READ

```
P(pu);
V(pu),
P(pw),
V(pw),
smp = prm;

P(rc(smp)),
  count(smp) ++,
if( count(smp) = 1)
 {
  P(w);
  P(pw);
 }

V(rc(smp)),

intent to read operations
 ..
P(rc(smp));
count( smp) --;
if(count( smp) = 0)
 {
typ(smp) = IR,
prm = ( prm + 1) mod 3;


V(pw);
V(w),

 }
V(rc(smp));
```

INTENT TO WRITE

```
P(pu),
V(pu),
P(pw),
V(pw),


if(typ(prm)=IW||typ(prm)=IR)
   smp = prm,
else if(typ(prm+1) mod 3 = IW
              || typ(prm+1) mod 3) = IR)
   smp = (prm+1) mod 3,
else
         smp = ( prm+2) mod 3,

P(rc(smp)),
    count( smp)++,
if(typ(smp) = IR)
     typ(smp) = IW,
if( count(smp) = 1)
   {
      P(w),
      P(pw);
   }
V(rc(smp)),

intent to write operations
....
P(rc(smp));
count(smp)--,
if(count(smp) = 0)
{
   typ(smp) = IR,
   prm = ( prm + 1) mod 3,
   V(pw);
   V(w),
}
V(rc(smp)),
```

Figure 2.8 Intent to Read and Write Algorithms - Fair and Efficient Readers and Writers with Intent to Read, Write and Upgrade Lock.

Upgrade

```
P(pu),
P(u);
V(pu),
P(pw),
V(pw);
If(count(prm) = 0)
   smp = prm;
else if(count((prm + 1) mod 3) = 0)
   smp = (prm + 1) mod 3,
else
   smp = (prm + 2) mod 3;

P(rc(smp));
count(smp)++;
typ(smp) = U;
if(count(smp) = 1)
{
   P(w),
   P(pw),
}
V(rc(smp));

reading is performed
 …
While(count(smp) > 1);

writing is performed
…

count(smp) = IR;
prm = ( prm + 1) mod 3,
V(pw),
V(w),
V(u),
```

Figure 2.9: Upgrade Algorithm - Fair and Efficient Readers and Writers with Intent to Read, Write and Upgrade lock

- Read procedure
    - Reader requests and releases the outer guard *pu*.
    - Reader requests and releases the inner guard *pw*.

- If the type of the primary semaphore is not *IW*, assign the primary semaphore to variable *smp*.
- Lock *rc(smp)* to exclusively access *count.*
- If the type of semaphore is *IR*, upgrade it to *R*
- If it is the first reader, lock the mutual exclusion semaphore w then *pw.*
- Release *rc(smp)* on *count*.
- Read operation is performed.
- Lock *rc(smp)* again to update the count.
- If it is last reader, reset the type of the semaphore to *IR* and make the primary flag point to the next *rc* semaphore 0, 1, or 2.
- Release *pw* then *w*.
- Release *rc(smp).*

- Write procedure
  - Request and release the outer guard, semaphore *pu*.
  - Request and release the inner guard, semaphore *pw*.
  - Request semaphore *w* to block other processes.
  - Write operations are performed.
  - Release semaphore *w* to allow other processes to enter critical section.

- Intent to write procedure
  - Request and release the outer guard, semaphore *pu*.

- Request and release the inner guard, semaphore *pw*.
- If the type of the primary semaphore is *IW* or *IR*, assign the primary to variable *smp*.

  Otherwise, assign secondary or tertiary to variable *smp*.
- Request semaphore *rc(smp)* to exclusively access *count(smp).*
- If the type of semaphore is *IR*, upgrade it to *IW.*
- If it is the first intent to write request, request the mutual exclusion semaphore *w*, then *pw*.
- Release semaphore *rc(smp)* on *count*.
- Intent to write operations…
- Request semaphore *rc(smp)* again to update *count*.
- If it is the last intent to write request, reset the type of semaphore *rc(*smp) to *IR* and make the primary flag point to the next *rc()* semaphore.
- Release semaphore *pw* then *w*.
- Release semaphore *rc(smp)*.

* Intent to read Procedure
  - Request and release the outer guard, semaphore *pu*.
  - Request and release the inner guard, semaphore *pw*.
  - Assign the primary semaphore to variable *smp*.
  - Request semaphore *rc(smp)* to exclusively access *count*. Update *count*.

- If it is the first reader, lock the mutual exclusion semaphore *w*, then *pw*.
- Release semaphore *rc(smp).*
- Intent to read operation.
- Request semaphore *rc(smp)* again to update *count*.
- If it is last intent to read request, reset the type of semaphore *rc(smp)* to *IR* and make the primary flag point to the next *rc()* semaphore.
- Release semaphore *pw*, then *w*.
- Release semaphore *rc(smp).*

- Upgrade procedure
    - Request the outer guard, semaphore *pu*.
    - Request upgrade semaphore *u*. Only one upgrade process can enter this semaphore.
    - Release the outer guard *pu*.
    - Request the inner guard, semaphore *pw*.
    - If count of the primary is 0, assign primary semaphore to variable *smp*. Else if count of secondary is 0, assign secondary semaphore to variable *smp* else assign tertiary semaphore to *smp.* An upgrade request only can enter an empty semaphore.
    - Request semaphore *rc(smp)* to exclusively access the count.
    - Let type of semaphore *rc(smp)* be *U*.
    - Request the inner semaphore w then *pw*.

- Release semaphore *rc(smp)* on *count*
- Read operations
- Wait until only upgrade is in the critical section.
- Write operation is performed.
- After upgrade operation finishes, downgrade the type of semaphore *rc(smp)* to *IR*.
- Make the primary flag point to the next *rc()* semaphore.
- Release semaphore *pw*, then *w*.
- Release semaphore *u*.

2.6.4 Discussion of Correctness – Fair and Efficient Readers and Writers with Intent to Read, Intent to Write and Upgrade Lock

Fairness:

There is a significant issue of fairness between upgrade and write locks. To avoid situations in which an upgrade lock "jumps ahead" of a waiting write lock, two policies are followed. The first policy requires an upgrade to obtain its own exclusive lock, but allows waiting reads, and intent to reads to share the lock. The second policy utilizes the *u* and *pu* semaphores to provide two assurances: First, at most one upgrade request is "in" arbitration at a time; and second, if an upgrade request is waiting because a prior upgrade request is already "in" arbitration, all subsequent requests will queue on the *pu* semaphore. Semaphore *pu* is used to block all requests subsequent to the second upgrade request (blocked by semaphore *u*) so that they are not allowed to enter arbitration. Because the additional *rc* semaphore *rc(2)*, *pu* and *u* semaphore are added, the algorithm guarantees that an upgrader process can enter an empty semaphore. Even though the read part of upgrader process can share with other waiting reads and intent to reads, it has to obtain mutual exclusive lock to start a write. If a waiting writer queued ahead of the upgrader, the upgrader is not able to obtain the mutual exclusive lock before the writer thus the upgrader can not take priority over the writer and the fairness is assured.

Liveness:

It is similar to the previous five algorithms in that if the pre_upgrade, upgrade and other *rc* semaphores initialize, request and release properly, deadlock will not happen. In addition, this algorithm can avoid deadlock in the situation when two or more users first read then write by obtaining upgrade locks before write on database. The third *rc* semaphore *rc(2)*, *pu* and *u* semaphore are added in this algorithm than the previous one only with intent to read and write locks. When there is an upgrade request, *rc(2)* and *pu* semaphores can guarantee the upgrader enter an empty semaphore(Because IW and R are incompatible, they may occupy two of *rc* semaphore, this is why we need the third *rc* semaphore). In addition, because of the *u* and *pu* semaphore, it permits only one upgrader “in” a lock arbitration. The second upgrader request will be blocked by the *u* semaphore and the subsequent processes after the second upgrader request will be blocked by the *pu* semaphore. The algorithm allows an upgrader obtain an upgrader lock before obtaining a write lock and block the second upgrader until the first upgrader releases its lock and thus deadlock will not occur.

Efficiency:

The principal difference between this algorithm and previous five algorithms is that in the previous five algorithms, deadlock avoidance requires acquisition of an exclusive lock for the entire update, including both read and write operations. In this algorithm, read and intent to read locks can share access during the read phase of the “upgrade” transaction, thereby providing additional potential concurrency, possibly creating improved efficiency.

CHAPTER 3

EXPERIMENTAL DESIGN

3.1 Design of the Experiment

The experiment is designed as a simulated database application, which is accessed by several readers, writers and upgraders. The actual read, write and upgrade are simulated by putting the reader, writer, and upgrader threads into sleep for certain period of time, and represented by the outputs such as "Reader # is reading", " Writer # is writing", " Upgrader # is upgrading", "Reader # is reading from record #" etc. This experiment compares the six reader-and-writer algorithms described in Chapter 2. In this experiment, we designed six procedures that implement the six reader-and-writer algorithms. Among the six algorithms, we have three categories of design approach. The first category is the four reader and writer algorithms. This category includes reader privilege algorithm, writer privilege algorithm, fair reader and writer algorithm as well as fair and efficient reader and writer algorithm. The purpose of the design of this category is to compare fairness, efficiency and liveness of these four algorithms. The same approach of design was used for the four reader and writer algorithms with modifications of certain methods.

The second category is the fair and efficient reader and writer algorithm with intent to read and intent to write locks. The design objective of this category is to demonstrate that this algorithm can achieve hierarchical locking in a database thus increasing concurrency and efficiency by providing hierarchical locking necessitate use of a more complex approach of design.

The third category is the fair and efficient reader and writer algorithm with intent to read, intent to write and upgrade lock. The design objective of this category is to show that deadlock can be avoided by adding upgrade lock. Including upgrade locks differentiated this design from that of the other five algorithms. A benchmark is designed to test and compare all six algorithms.

There are three versions of implemented, one with extensive output (the Verbose version) two without (the Throughput version and Turnaround version). The Verbose version will display detailed information of what happens for all requests (such as the phenomena of reader starvation, writer starvation and concurrently access the different records by readers and writers etc) and output the time spent for those requests and the average time spent to obtain various types of lock under the different algorithms. The Throughput version displays the total time elapsed for benchmark execution of all requests in order to get more precise results for efficiency. The Turnaround version displays the individual waiting time to obtain each lock and average waiting time to obtain each type of lock under the different algorithms in order to analyze the efficiency.

The structure of the experiment is depicted in Figure 3.1.

The tokens in Figure 3.1 are:

RW_Server: The benchmark to test and compare the six algorithms.

RW_Server_1: The implementation for the reader privilege algorithm.

RW_Server_2: The implementation for the writer privilege algorithm.

RW_Server_3: The implementation for the fair reader and writer algorithm.

RW_Server_4: The implementation for the fair and efficient reader and writer algorithm.

I_RW_Server_5: The implementation for the fair and efficient reader and writer algorithm with intent to read and intent to write lock.

I_RW_U_Server_6: The implementation for the fair and efficient reader and writer algorithm with intent to read, intent to write lock and upgrade lock.

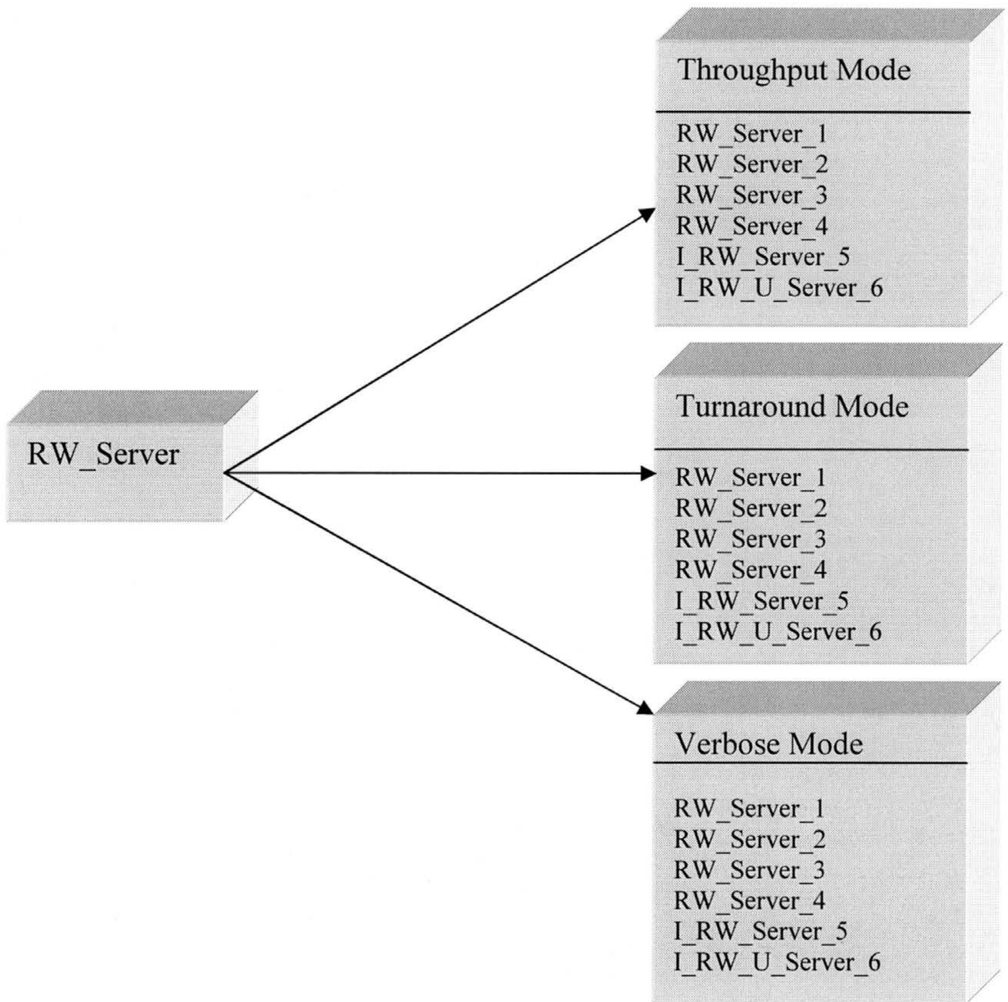

Figure 3.1 The Structure of the Experiment

The whole project executable package is designed in such as a way that the user has the option of choosing the display mode (verbose, throughput or turnaround) for execution results. Also, the project allows the user to enter the number of requests, the duration time for the table reading operation, the record reading operation, the table writing operation, the record writing operation and the interval time between each request. Then it generates the table reader, table writer, table upgrader, record reader, record writer, and record upgrader requests pseudo-randomly. These are generated pseudo-randomly so that the execution results can be compared. Upgrader threads are always generated for all algorithms. But for the first five algorithms that do not actually support upgrade locks, an upgrader request is replaced by one writer request with duration equal to a reader time plus a writer time.

There are a total of six kinds of requests: Table Read, Table Write, Table Upgrade, Record Read, Record Write and Record Upgrade. Table Read, Table Write and Table Upgrade will access a table as a whole and Record Read, Record Write and Record Upgrade will access a specific record in the table. An example scenario with all requests and duration time is specified in Table 3.1.

Table 3.1 An Example Scenario

Table Read Time = 40, Table Write Time = 60,
Record Read Time = 20, Record Write Time = 30

Request Type	Number of Requests	Critical Section Activity	Duration (Relative Wait Times in Milliseconds)		
			Reader-Writer Algorithms	IR/IW Algorithm	IR/IW/Upgrade Algorithm
Table Read	10	Table Read Lock	40	40	40
Table Write	10	Table Write Lock	60	60	60
Table Upgrade	10	Table Upgrade Lock	N/A	N/A	40
		Table Write Lock	100	100	60
Record Read	50	Table Read Lock	20	N/A	N/A
		Table Intent Read Lock / Record Read Lock	N/A	20	20
Record Write	50	Table Write Lock	30	N/A	N/A
		Table Intent Write Lock / Record Write Lock	N/A	30	30
Record Upgrade	50	Table Write Lock	50	N/A	N/A
		Table Intent Write Lock / Record Upgrade Lock	N/A	N/A	20
		Table Intent Write Lock / Record Write Lock	N/A	50	30

3.2 Description of the Experiment

3.2.1 RW_Server

RW_Server implements the benchmark and test program for all the six algorithms. It creates various requests and resources they want to access the database. It processes the six types of requests. The resource includes a table and its five records in the table. The user inputs the number of requests, the duration time of each of the six types of requests and the interval time between the arrivals of each request. Request types are determined pseudo randomly with equal probability of read, write or upgrade and equal probability of each of the six resources, namely the table and five constituent records. These requests, duration time and interval time can be fed into the 3 different groups of six programs, representing the six algorithms but in different output mode. The first group of the programs outputs the execution results in Verbose mode in which detailed information about the phases that each thread goes through and its relative order against other threads is presented. The second group of programs outputs the execution results in Throughput mode in which only time information for comparing the efficiency of the algorithms is presented. The third group of programs outputs the execution results in Turnaround mode in which only the individual waiting time to obtain each lock and average waiting time to obtain each type of lock under different algorithms are presented. This group of programs is used for analyzing the efficiency for the six different algorithms. Then the data from executing these six programs will be collected and comparison results will be obtained and analyzed.

3.2.2 RW_Server_1

RW_Server_1 implements the reader privilege algorithm. This is used to test if there exists writer starvation phenomena during the simulation of readers and writers competing for grants to access the database. Turnaround and Throughput metrics are collected for comparison with the other five algorithms.

3.2.3 RW_Server_2

RW_Server_2 implements the writer privilege algorithm. This is used to test if there exists reader starvation phenomena during simulation of readers and writers competing for grants to access the database. Turnaround and Throughput metrics are collected for comparison with the other five algorithms.

3.2.4 RW_Server_3

RW_Server_3 implements the fair reader and writer algorithm. This is used to demonstrate if FIFO order can be obtained during simulation of readers and writers competing for the grants to access the database. Turnaround and Throughput metrics are collected for comparison with the other five algorithms.

3.2.5 RW_Server_4

RW_Server_4 implements the fair and efficient reader and writer algorithm. This is used to determine whether this algorithm is more efficient than the previous algorithms during simulation of readers and writers competing for grants to access the database. Turnaround and Throughput metrics are collected for comparison with the other five algorithms.

3.2.6 I_RW_Server_5

I_RW_Server_5 implements the fair and efficient reader and writer algorithm with intent to read and intent to write locks. It simulates a two level database (a table and the records in that table) being accessed by a number of readers and writers. Among the readers and writers, some of them try to access the table as a whole while others try to access the individual records of the table. The experiment will test whether hierarchical access using intent to read and intent to write locks is more efficient than the other five algorithms.

3.2.7 I_RW_U_Server_6

I_RW_U_Server_6 implements the fair and efficient reader and writer algorithm with intent to read, intent to write and upgrade locks. It utilizes a two level database (a table and the records in that table) that is accessed by a number of readers, writers and upgraders. Among the readers, writers and upgraders, some of them try to access the table as a whole while the rest try to access the individual records of the table. The experiment will test whether deadlock can be avoided by adding the upgrade lock, and compare the efficiency of this algorithm with the five preceding ones.

CHAPTER 4

DESCRIPTION OF IMPLEMENTATION

4.1 Software Required to Implement the Experiment

The software is implemented in the JAVA programming language. It is compatible with JDK 1.3 or 1.4. Since it is in JAVA, it is very portable. However, the execution of the experiment is automated for Windows with a MS-DOS batch script program. If a system other than Windows is used, the CLASSPATH environment variable must be set up, and the program must be built and executed manually. Refer to the user instruction document in Appendix C that comes with the project package for the automated execution of the project program on Windows.

4.2 Implementation Versions

There are three versions of implementations as described in Section 3.1. The user is allowed to choose which version he/she wants to run. The source code of the Verbose version is in Appendix A and The sample results of running the three versions of implementation is in Appendix B.

The Verbose version displays which thread is running and what it is doing at selected points during the execution of the program. This information is useful in investigating the relative order of the threads, deadlocks and starvation. But the costs for outputting these excessive texts to the screen make it unsuitable for benchmark comparison among the algorithms. On the other hand, although the Throughput version does not provide detailed information, performance data collected from this version is more accurate. The Throughput version provides an overall performance metric. In contrast, the Turnaround version records the waiting time for obtaining each lock and the average waiting time for obtaining each type of lock for each algorithm. This data is used to analyze the performance of each algorithm.

4.3 The Specification of Classes and Main Methods

4.3.1 The Classes and Methods of the Design Category One Implementation - Reader and Writer Algorithms

This category consists of the Reader Privilege Algorithm, the Writer Privilege Algorithm, the Fair Reader and Writer Algorithm as well as the Fair and Efficient Reader and Writer Algorithm. Each of these algorithms is implemented with JAVA classes: Database, Reader, Writer, Break, Semaphore and RW_Server [SiGG03]. The relationship of these five classes is depicted in Figure 4.1. Each is described in one of the following sections.

Database Class

This class contains the semaphores, readCount and writeCount(if it is necessary). It contains four methods: **startRead, startWrite, endRead and endWrite.**

startRead: starts a read process using different algorithms and returns a reader count.

endRead: ends a read process using different algorithms and returns a reader count.

startWrite: starts a write process using different algorithms.

endWrite: ends a write process and releases the mutual exclusive semaphore.

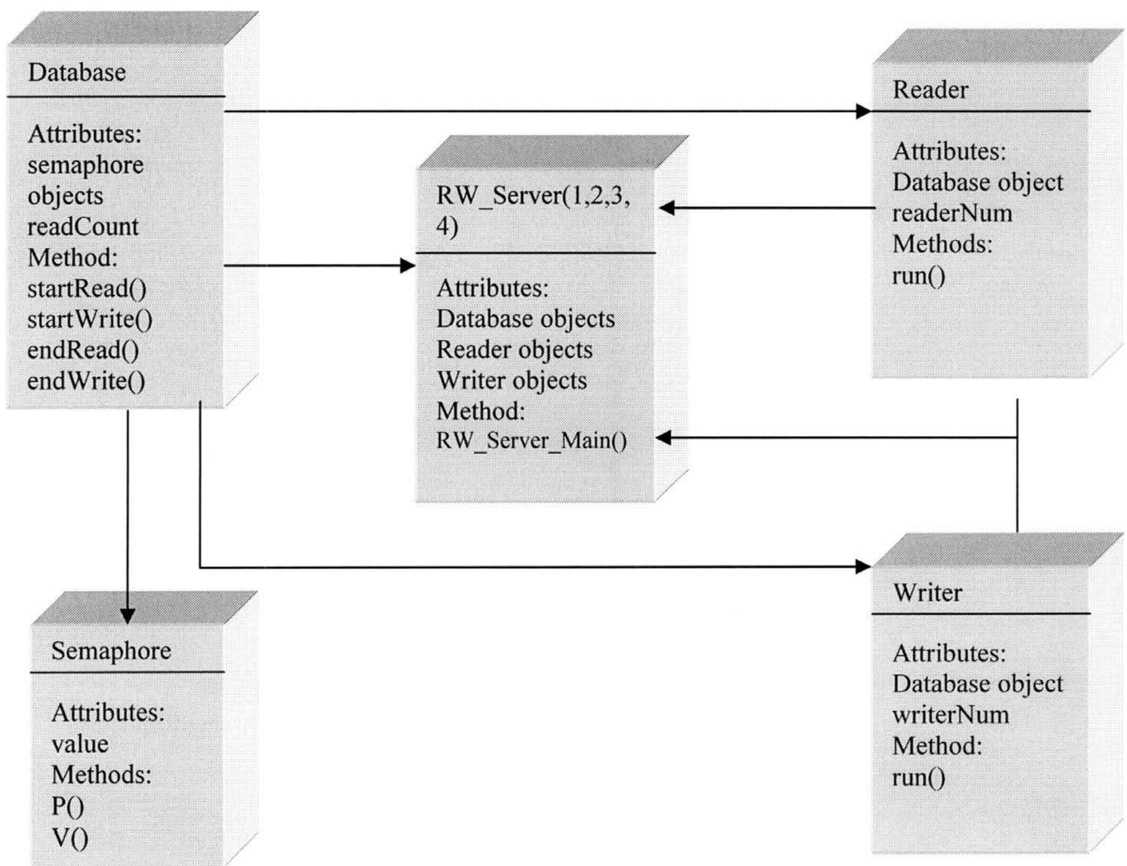

Figure 4.1 Relationships Among Classes for Reader and Writer Algorithms

Reader Class

This class shows activity of a specific reader. It only has a **run** method.

run: The method calls the **startRead** method of database class to start a read process and calls the **endRead** method of database class to end a read process.

Writer Class

This class shows activity of a specific writer. It only has a **run** method.

run: The method calls the **startWrite** method of database class to start a write process and calls the **endWrite** method of database class to end a read process.

Semaphore Class

This class executes the two procedures of semaphore **P** and **V**.

P: Requests a semaphore

V: Releases a semaphore.

RW_Server Class

This class specifies six different types of requests then calls the method of reader and writer class to access the reader and writer requests. It only contains one method RW_Server_Main (It is converted to RW_Server_1_Main in Algorithm_1, RW_Server_2_Main in Algorithm_2, RW_Server_3_Main in Algorithm_3, RW_Server_4_Main in Algorithm_4).

RW_Server_Main: If the request type is 0, it indicates that the request is a reader process and if the record number is 5, it means the request is a table reader, which wants to read from the table as a whole. If the record number is from 0 to 4, it indicates that the reader process wants to read from the specific record. If the request type is 1, it indicates the request is a writer process and if the record number is 5, it means that the request is a table writer, which wants to write to the table as a whole. If the record number is from 0 to 4, it indicates that the writer process wants to write to the specific record. If the request type is 2, it indicates the request is an upgrader process and if the record number is 5, it indicates that the request is a table upgrader, which wants to upgrade the table as a whole. If the record number is from 0 to 4, it indicates that the upgrader process wants to upgrade the specific record. Because there is no upgrader algorithm in this category of implementation, we just consider a table upgrade or a record upgrade as a table write or a record write with duration time equals to a table reader time plus a table writer time or a record reader time plus a record writer time After specifying the six types of requests, the method call the run() method of Reader and Writer class to start the reader and the writer processes.

4.3.2 The Classes and Methods of the Design Category Two Implementation - Readers and Writers with Intentional Locks

This category includes Fair and Efficient Reader and Writer Algorithm with Intent to Read and Intent to Write Lock. This category of experiment implementation consists of ten classes: Resource, Record, Type, Table, Reader, Writer, Semaphore, Break, Return_Value and RW_Server. The relationship of these ten classes is shown in Figure 4.2.

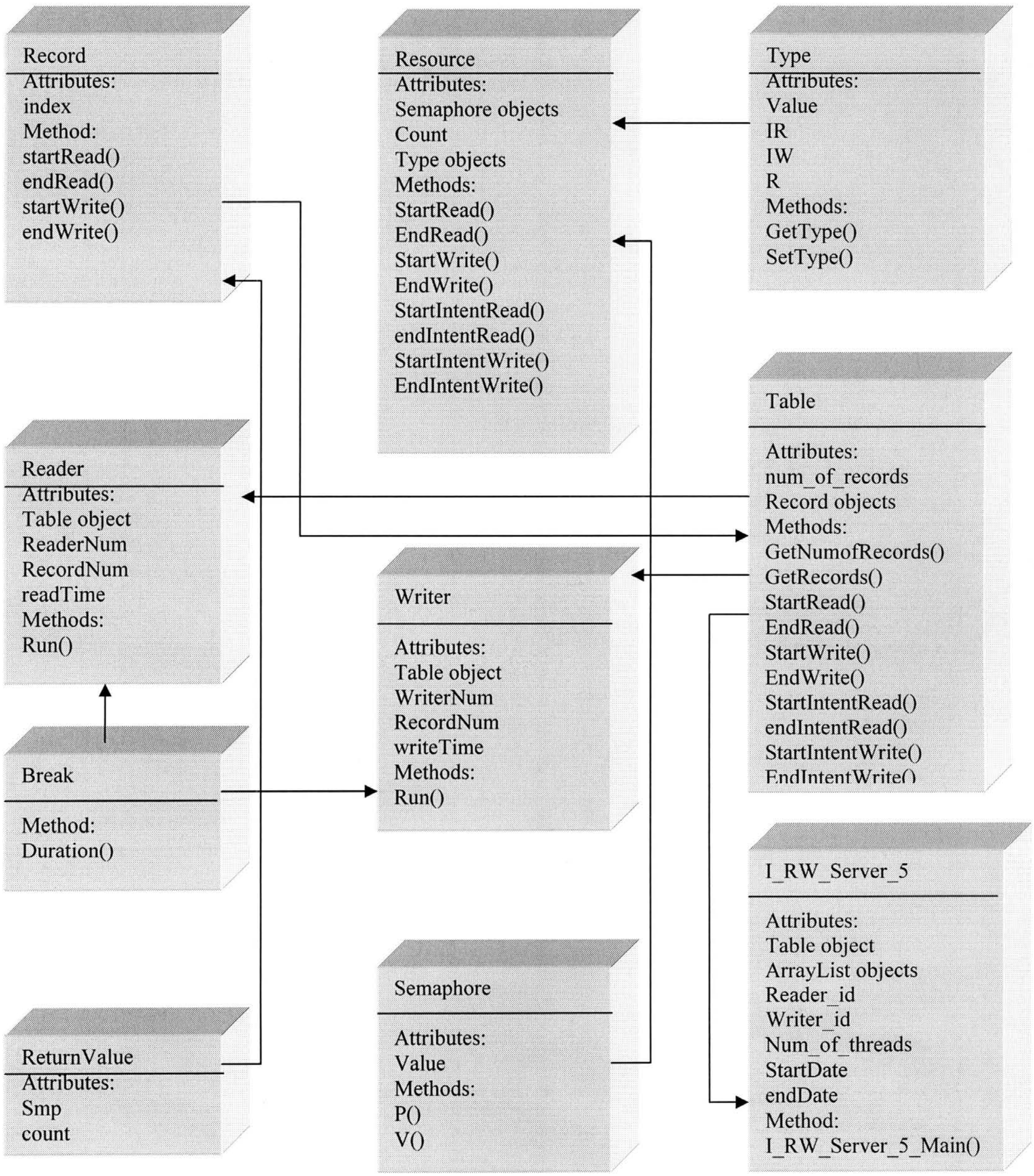

Figure 4.2: Relationships Among Classes for Fair Reader and Writer with Intentional Locks

Resource Class:

This is the base class of Table class and Record class. It contains general methods of Table class and Record class.

startRead: starts a read process using fair and efficient reader and writer algorithm with intent to read and intent to write.

endRead : ends a read process using fair and efficient reader and writer algorithm with intent to read and intent to write.

startIntentRead: starts an intent to read process using fair and efficient reader and writer algorithm with intent to read and intent to write.

startIntentWrite: starts an intent to write process using fair and efficient reader and writer algorithm with intent to read and intent to write.

endIntentRead: ends an intent to read process using fair and efficient reader and writer algorithm with intent to read and intent to write.

endIntentWrite: ends an intent to write process using fair and efficient reader and writer algorithm with intent to read and intent to write.

startWrite: starts a write process using fair and efficient reader and writer algorithm with intent to read and intent to write.

endWrite: ends a write process and releases the mutual exclusive semaphore.

Record Class:

This class implements fair and efficient reader and writer to simulate read and write on the individual records in a table. It contains four methods:

startRead: inherits from the base class.

endRead: ends a read process using fair and efficient reader and writer algorithm with intent to read and write lock.

startWrite: inherits from the base class.

endWrite: inherits from the base class.

Table Class:

This class simulates the table that contains records. It implements the fair and efficient reader and writer with intent to read and intent to write lock algorithm for read, write, intent to read and intent to write. It contains ten major methods.

GetNumOfRecord: gets number of record in a table

GetRecord: gets a record number in a table.

startRead: inherits from the base class.

endRead: ends a read process using fair and efficient reader and writer algorithm with intent to read and write lock.

startIntentRead: inherits from the base class.

endIntentRead: ends an intent to read process using fair and efficient reader and writer algorithm with intent to read and intent to write.

endIntentWrite: ends an intent to read process using fair and efficient reader and writer algorithm with intent to read and intent to write.

startWrite: inherits from the base class.

endWrite: inherits from the base class.

Type class:

This class defines the types of the two share semaphores. The type is initialized to be IR and the supported types are IR, IW and R. The class contains two major methods.

getType: get a type(IR, IW or R)

setType: set a type(IR, IW or R)

Reader class

This class calls the methods of the Table class to start and end a read process. The class only contains a run() method.

run: If the request is Table Read, the method calls **startRead** method of the Table class to start a read process, then calls **endRead** method of Table class to end the read process. If the request is Record Read, the method first calls **startIntendRead** method of Table class to obtain an Intent to Read lock, then calls **startRead** method of Record class to start a read process and finally calls **endIntentRead** to release the intent to read lock.

Writer class

This class calls the methods of the Table class to start and end a write process. The class only contains a run() method.

run: If the request is Table Write, the method calls **startWrite** method of the Table class to start a write process, then calls **endWrite** method of Table class to end the write process. If the request is Record Write, the method first calls

startIntendWrite method of Table class to obtain an Intent to Write lock, then calls **startWrite** method of Record class to start a write process and finally calls **endIntentWrite** to release the intent to write lock.

Semaphore Class

This class executes the two procedures of semaphore **P** and **V**.

P: Requests a semaphore

V: Releases a semaphore.

ReturnValue class

This class only has two attributes and no methods. The attributes are smp and count. They represent the current share semaphore and its count.

Break class

This class only has one method: **Duration()**

Duration(): This method determines the duration of each request.

I_RW_Server_5 Class

This class is designed to manipulate reader and writer process. It only has I_RW_Server_5_Main method.

I_RW_Server_5_Main: If the request type is 0, it indicates that the request is a reader process and if the record number is 5, it means the request is a table reader, which wants to read from the table as a whole. If the record number is from 0 to 4, it indicates that the reader process wants to read from the specific record. If the request type is 1, it indicates the request is a writer process and if the record number is 5, it means the request is a table writer, which wants to write to the table as a whole. If the record number is from 0 to 4, it indicates that the writer process wants to write to the specific record. If the request type is 2, it indicates the request is an upgrade process and if the record number is 5, it means the request is for table upgrade, which needs to upgrade the table as a whole. If the record number is from 0 to 4, it indicates that the upgrade process wants to upgrade that specific record. Because there is no upgrade algorithm in this category of implementation, we just consider a table upgrade or a record upgrade as a table write or record write with duration time equals to a table reader time plus a table writer time or a record reader time plus a record writer time. After specifying the six types of requests, the method call the run() method of Reader and Writer class to start the reader and the writer processes.

4.3.3 The Classes and Methods of the Design Category Three Implementation - Readers and Writers with Intentional Locks and Upgrade Locks

This category includes fair and efficient reader and writer algorithm with intent to read, intent to write lock and upgrade lock.

This category of experiment implementation consists of eleven classes: Resource, Record, Type, Table, Reader, Writer, Upgrader, Semaphore, Break, Return_Value and I_RW_Server. The only difference between this implementation from the previous one is that the Upgrader class is added for the upgrade request. In addition, StartUpgrade and endUpgade methods are added to Table class and Record class to support the upgrade request. Of course, in the Type class, there should be one more type U(upgrader). The Upgrader class is described as follow:

Upgrader Class

This class calls the methods of the Table class to start and end an upgrader process. The class only contains a run() method.

run: If the request is Table Upgrader, the method calls the **startUpgrade** method of Table class to start a read process and read until only the upgrader itself in the critical section(the table) then start to write to the table. When it is done, it calls the **endUpgrade** method of Table class to end an upgrade process. If the request is Record Upgrader, the method first calls **upgrader_startIntentWrite** method of Table class to obtain an Intent to write lock, then calls **startUpgrade** method of Record class to start read process and read until only the upgrader itself in the critical section(the record), then start to write to the record. Finally it calls the **endUpgrade** of the Record class to end an upgrade process and calls the **upgrader_endIntentWrite** to release the intent to write lock.

I_RW_Server_6 Class

This class is designed to manipulate reader and writer process. It only has I_RW_U_Server_6_Main method.

I_RW_U_Server_6_Main: If the request type is 0, it indicates that the request is a reader process and if the record number is 5, it means the request is a table reader, which wants to read from the table as a whole. If the record number is from 0 to 4, it indicates that the reader process wants to read from the specific record. If the request type is 1, it indicates the request is a writer process and if the record number is 5, it means the request is a table writer, which wants to write to the table as a whole. If the record number is from 0 to 4, it indicates that the writer process wants to write to the specific record. If the request type is 2, it indicates the request is an upgrader process and if the record number is 5, it means the request for table upgrade, which needs to upgrade the table as a whole. If the record number is from 0 to 4, it indicates that the upgrader process wants to upgrade that specific record. After specifying the six types of requests, the method calls the run() method of Reader or Writer or Upgrader class to start the reader, the writer and the upgrader process.

The relationship between those eleven classes is shown in Figure 4.3.

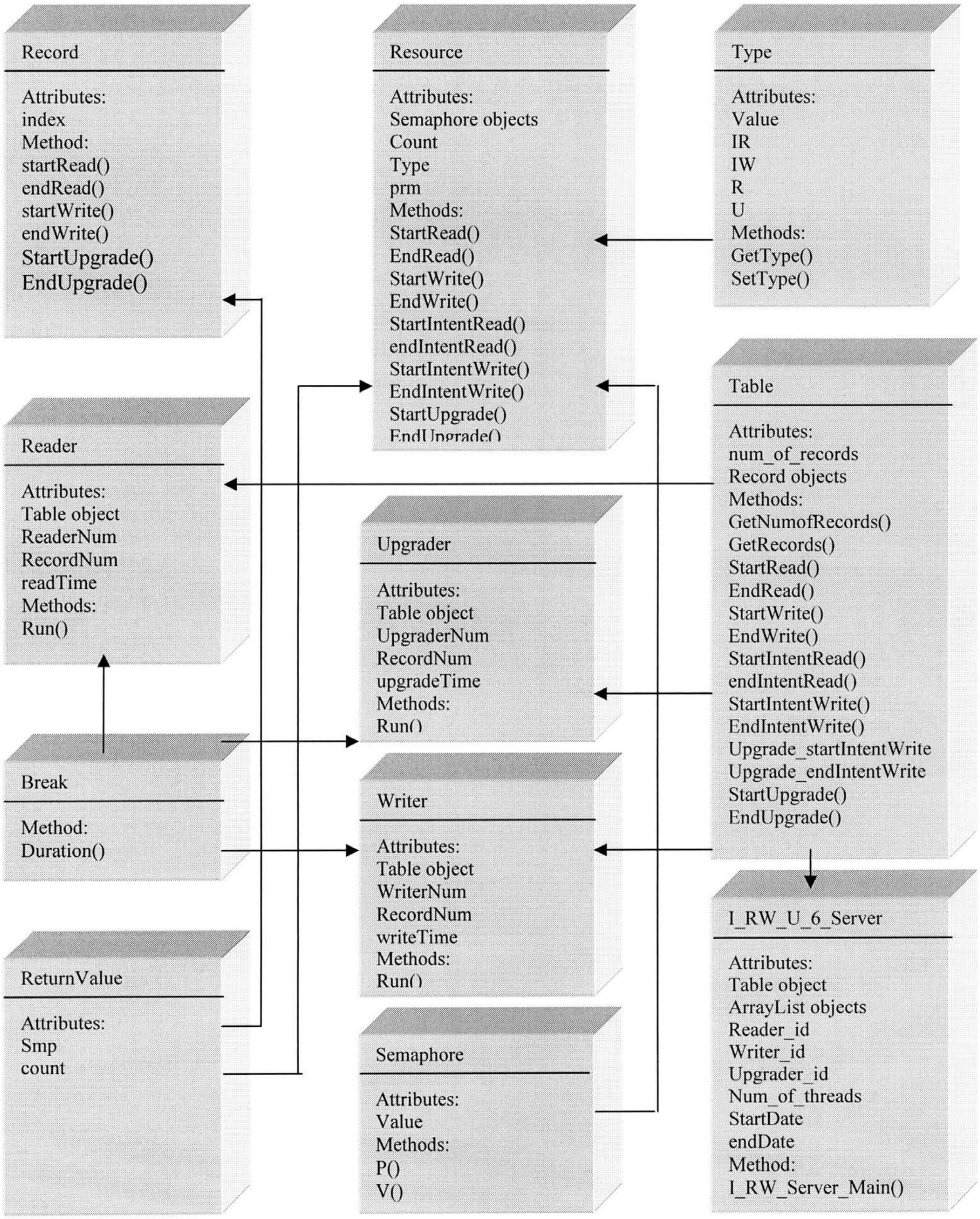

Figure 4.3: Relationships Among Classes for Fair Reader and Writer with Intentional Locks and Upgrade Locks

4.3.4 Change the Class Name During Implementation

We have totally six algorithm implementations, some of which have same class names but different implementations. We found we had to change the class names in order for the system to run the different implementations properly. For example, in the source code, we changed the class names such as database, reader, writer, upgrader to database_1 in Algorithm 1, to database_2 in Algorithm 2, to reader_2 in Algorithm 2, to writer_3 in Algorithm 3 or to upgrader_6 in Algorithm 6, etc.

CHAPTER 5

DISCUSSION AND ANALYSIS OF THE TEST RESULTS

We conducted multiple experiments to get a series of results. The detailed results of running the Verbose version of implementation are provided in Appendix A.

5.1 Adjusting the Requests According to Lock Types Available

In order to make all six algorithms comparable, we have to do some adjustments based on the availability of lock types. There are six types of requests (table read, table write, record read, record write, table upgrade, record upgrade) for the experiment. For those algorithms without hierarchical locks (intention to read and intention to write), we have to translate the record read and record write to table read and table write. Since an upgrade process is a read process followed by a write process, for those algorithms without hierarchical and sequential (upgrade) locks, we have to adjust a table (record) upgrade to a table write request equal in duration to one table(record) read request plus one table (record) write request; For the algorithm with hierarchical locks but not sequential locks, we have to adjust a table upgrade to one table write request (with duration equal to table read plus table write) and adjust a record upgrade to one record write request(with duration equal to record read plus record write). An example is shown in Table 3.1.

5.2 Discussion of Results Using the Verbose Version

From examination of Verbose version runs (See Appendix B for example run), we can observe the following phenomena:

1. For the Reader Privilege Algorithm, once there is a reader who wants to read from the database, the writers after the reader could not access the database until all readers followed by that reader finish accessing the database. If the readers keep entering, the writers will never have a chance to access the database, thus writer starvation occurs.
2. For the Writer Privilege Algorithm, once there is a writer who wants to write to the database, the readers after this writer could not access the database until all writers following by that writer finish accessing the database. If writers continue entering, readers never have a chance to access the database, thus reader starvation occurs.
3. For the Fair Reader and Writer Algorithm, all requests are processed in first come and first serve order.

4. For the Fair and Efficient Reader and Writer Algorithm, as long as the readers enter the outer semaphore in the same lock arbitration, they can read concurrently no matter how many writer requests are interleaved.
5. For the Fair and Efficient Reader and Writer Algorithm with Intent to Read and Intent to Write lock, the hierarchical and parallel access to a database can be achieved. For example, when a reader is reading from record 1 of a table, a writer still can write to record 2 of that table.
6. For the Fair and Efficient Reader and Writer Algorithm with Intent to Read, Intent to Write Lock and Upgrade Lock, an upgrading process can first read then write to the whole table or a record as long as there is only one upgrading process is reading from that table or that record.

5.3 Discussion of Results Using the Throughput Version

Since the efficiency issue is our key point of this research, the throughput version is designed to get more precise data by only outputting the time spent for the six algorithms. The test results are illustrated in the following tables. The time spent results are based on the same request sets and the same table reader time, table writer time, table upgrader time, record reader time, record writer time, record upgrader time and the same interval time for the six algorithms.

5.3.1 Comparing the Time Elapsed for Six Different Algorithms

Tables 5.1 and 5.2 showed that if number of requests is small, no significant difference in the time spent by these algorithms is observed. But, when the number of requests becomes bigger, only the time elapsed with Algorithm 1, Algorithm 2 and Algorithm_4 are comparable. For Algorithm 1, the readers block the writers and thus let readers read concurrently before any writer starts to write. As a result, the concurrency and efficiency of the readers is increased. Similarly, for the Algorithm 2, the writers block the readers and thus let readers read concurrently after all writers finish their job. As a result, the concurrency and efficiency of readers is maximized. For Algorithm 4, because readers can read concurrently as long as they arrive at the same lock arbitration, increased reader efficiency is achieved. The time elapsed for Algorithm 3 becomes significantly longer than for the other three Reader and Writer Algorithms as the number of requests increases. The reason is that if there are one or more writers interleaving into readers, only the sequence of readers between two writers can read simultaneously so that the concurrency of the readers is decreased under contention. Among the six algorithms, as the number of requests increases, the time elapsed for Algorithm 5 increases the slowest followed by Algorithm 6. Algorithm 5 is the most efficient when the number of requests is large. This is because Algorithm 5 allows the readers and the writers to access a different resource of a database (for example, different record) concurrently without conflict. Even though Algorithm 6 has the same extent of concurrency as Algorithm 5, it apparently suffers from the large overhead involved in obtaining the upgrade locks. Recall that lock arbitration is deferred when there are two upgrade requests outstanding. These results indicate that in some circumstances, this additional cost for implementing upgrade locks is not worthwhile.

Tables 5.2, 5.3 and 5.4 shows that the time spent for Algorithm 1 and Algorithm 2 is somewhat less than for Algorithm 4. Because in Algorithm 1, readers block the writers and thus let readers read concurrently before any writer starts to write and in Algorithm 2, the writers block the readers and thus let readers read concurrently after all writers finish their job, these two algorithms maximize reader concurrency. But in Algorithm 4, because there is a small interval between each request, the requests may not be able to enter the same lock arbitration, thereby allowing many readers but not all the readers to read concurrently. Reader concurrency is enhanced over Algorithm 3, but not maximized.

Table 5.1. The Time Elapsed for the Six Algorithms

table read time = 40 milliseconds, record read time = 20 milliseconds,
table write time = 60 milliseconds, record write time = 30 milliseconds,
table upgrade time = 100 milliseconds, record upgrade time = 50 milliseconds,
interval time = 0 milliseconds

Num_requests	**Algo_1** (Seconds)	**Algo_2** (Seconds)	**Algo_3** (Seconds)	**Algo_4** (Seconds)	**Algo_5** (Seconds)	**Algo_6** (Seconds)
4	0 12	0 12	0 12	0 12	0 09	0 09
10	0 27	0 27	0.31	0 27	0 21	0 21
20	0 53	0 53	0 67	0 53	0 36	0 37
50	1 56	1 56	1 86	1 56	1 15	1 44
100	3 14	3 14	3 65	3 14	1 83	2 51
200	5 98	5 95	7 12	6 01	3 54	4 83
400	12 72	12 69	14 88	12 71	6 82	9 33
800	25 70	25 67	29 81	25 69	12 90	18 57
1200	39 36	39 31	45 63	39 36	19 98	30 18
2400	77 90	77 75	90 27	77 97	38 87	48 64

Table 5.2. The Time Elapsed for Six Algorithms

table read time = 40 milliseconds, record read time = 20 milliseconds,
table write time = 60 milliseconds, record write time = 30 milliseconds,
table upgrade time = 100 milliseconds, record upgrade time = 50 milliseconds,
interval time = 10 milliseconds

Num_requests	Algo_1 (Seconds)	Algo_2 (Seconds)	Algo_3 (Seconds)	Algo_4 (Seconds)	Algo_5 (Seconds)	Algo_6 (Seconds)
4	0 14	0 14	0 14	0 15	0 12	0 13
10	0 27	0 27	0 31	0 27	0 3	0 31
20	0 58	0 53	0 67	0 57	0 41	0 66
50	1 61	1 56	1 86	1 64	1 39	1 83
100	3 21	3 14	3 64	3 22	2 43	2 33
200	6 06	5 95	7 11	6 07	3 76	4 82
400	12 82	12 69	14 87	12 83	7 32	9 64
800	25 80	25 66	29 80	25 77	14 21	18 46
1200	39 46	39 30	45 62	39 43	20 01	29 15
2400	78 47	78 78	90 27	77 88	40 48	60 90

Table 5.3. The Time Elapsed for Six Algorithms

table read time = 40 milliseconds, record read time = 20 milliseconds,
table write time = 60 milliseconds, record write time = 30 milliseconds,
table upgrade time = 100 milliseconds, record upgrade time = 50 milliseconds,
interval time = 20 milliseconds

Num_requests	Algo_1 (Seconds)	Algo_2 (Seconds)	Algo_3 (Seconds)	Algo_4 (Seconds)	Algo_5 (Seconds)	Algo_6 (Seconds)
4	0 14	0 14	0 13	0 14	0 19	0 15
10	0 29	0 27	0 31	0 31	0 33	0 33
20	0 59	0 53	0 67	0 61	0 56	0 58
50	1 66	1 56	1 86	1 68	1 52	1 58
100	3 26	3 14	3 65	3 28	2 29	2 67
200	6 14	5 95	7 11	6 19	4 62	5 01
400	12 91	12 69	14 88	12 95	8 69	9 24
800	25 93	25 67	29 81	25 99	16 66	18 42
1200	39 54	39 29	45 62	39 59	25 12	30 85
2400	77 92	77 73	90 26	77 97	49 09	61 20

Table 5.4. The Time Elapsed for Six Algorithms
table read time = 40 milliseconds, record read time = 20 milliseconds,
table write time = 60 milliseconds, record write time = 30 milliseconds,
table upgrade time = 100 milliseconds, record upgrade time = 50 milliseconds,
interval time = 30 milliseconds

Num_requests	Algo_1 (Seconds)	Algo_2 (Seconds)	Algo_3 (Seconds)	Algo_4 (Seconds)	Algo_5 (Seconds)	Algo_6 (Seconds)
4	0 12	0 12	0 12	0 12	0 12	0 12
10	0 32	0 32	0 32	0 32	0 32	0 32
20	0 68	0 64	0 68	0 70	0 66	0 70
50	1 83	1 75	1 87	1 85	1 78	1 95
100	3 45	3 33	3 66	3 48	3 15	3 26
200	6 49	6 16	7 12	6 56	6 25	6 35
400	13 41	12 90	14 88	13 49	12 30	12 37
800	26 62	25 86	29 81	26 77	24 62	24 73
1200	40 37	39 50	45 63	40 52	36 25	36 68
2400	78 85	77 88	90 97	80 87	74 55	77 97

Comparing the results for different intervals between requests offers a few additional points of interest. One is that the time for algorithm 6 reduced as the interval time was increased from 0 to 10 milliseconds with large numbers of requests. An explanation for this may be that the deferral of arbitration mentioned above used with the upgrade lock. This can be avoided by increasing the complexity of the upgrade implementation to allow two upgrade requests into arbitration at the same time. For example, one of the alternate upgrade implementations is adding one more *rc* semaphore, *rc(3)*, and changing the initial value of upgrade semaphore *u* to 2 in algorithm 6. This will allow two upgrade processes in the arbitration at the same time thus the concurrency are reasonably improved. We call this alternate upgrade implementation algorithm 6a. The example test results of comparing Algorithm 5, Algorithm 6 and

Algorithm 6a are illustrated in Table 5.5 and Table 5.6. Tables 5.5 and 5.6 show that the time elapsed for Algorithm 6a is longer than Algorithm 5 but is shorter than Algorithm 6 when the number of requests is getting larger. Even though Algorithm 6a improves the degree of the concurrency, it not eventually solves the problem. This will be a matter for the future research.

Another interesting observation is that when the interval reaches 30 milliseconds, the advantage of the hierarchical locking algorithms (5 and 6) significantly diminishes. This is due to the fact that the requests are now spread out over a major part of the execution time. For example, 2400 requests with 30 milliseconds interval, indicates that the last request was issued at time 71.97 seconds.

Table 5.5. The Time Elapsed for the Algorithm 5, Algorithm 6 and Algorithm 6a
table read time = 40 milliseconds, record read time = 20 milliseconds,
table write time = 60 milliseconds, record write time = 30 milliseconds,
table upgrade time = 100 milliseconds, record upgrade time = 50 milliseconds,
interval time = 0 milliseconds

Num_requests	**Algo_5** (Seconds)	**Algo_6** (Seconds)	**Algo_6a** (Seconds)
4	0 15	0 15	0 14
10	0 22	0 21	0 21
20	0 37	0 38	0 38
50	1 00	1 45	1 33
100	1 88	2 42	1 99
200	3 26	4 87	4 63
400	6 72	9 38	8 62
800	12 59	18 59	16 36
1200	18 87	30 00	27 57

Table 5.6. The Time Elapsed for the Algorithm 5, Algorithm 6 and Algorithm 6a

table read time = 40 milliseconds, record read time = 20 milliseconds,
table write time = 60 milliseconds, record write time = 30 milliseconds,
table upgrade time = 100 milliseconds, record upgrade time = 50 milliseconds,
interval time = 10 milliseconds

Num_requests	Algo_5 (Seconds)	Algo_6 (Seconds)	Algo_6a (Seconds)
4	0 09	0 09	0 09
10	0 3	0 31	0 3
20	0 52	0 66	0 56
50	1 34	1 83	1 76
100	2 43	2 52	2 33
200	3 73	4 95	4 65
400	7 17	9 76	9 38
800	13 30	18 58	17 10
1200	20 25	30 44	26 24

5.4 Discussion of Results using Turnaround Version

In order to analyze the reasons why the efficiency is different under the six reader and writer algorithms, we obtained additional information showing how long it will take for each type of request to be granted.

The sample results of the average time spent for obtaining the six types of request under the different algorithms for different intervals between requests are illustrated in Tables 5.7, 5.8 and 5.9.

Table 5.7: The Average Waiting Time in Milliseconds to Obtain a Lock Under Different Algorithm.
Request_num = 1200,
Table Reader Time = 40, Record Record Time = 20,
Table Write time = 60, Record Write Time = 30,
Table Upgrade Time = 100, Record Upgrade Time = 50,
Interval time = 10

Lock Type	Table Read (TR)	Record Read (RR)	Table Write (TW)	Record Write (RW)	Table Upgrade (TU)	Record Upgrade (RU)
Algorithm						
Algo_1	5796	6212	12798	13610	14156	13726
Algo_2	33110	33305	12587	13396	13943	13510
Algo_3	16629	16102	15684	16673	17346	16819
Algo_4	3362	3363	12811	13628	14181	13742
Algo_5	3296	1977	5136	2766	5553	2883
Algo_6	8645	8333	8440	8615	9063	8677

Table 5.8: The Average Waiting Time in Milliseconds to Obtain a Lock Under Different Algorithm.
Request_num = 1200,
Table Reader Time = 40, Record Record Time = 20,
Table Write time = 60, Record Write Time = 30,
Table Upgrade Time = 100, Record Upgrade Time = 50,
Interval time = 20

Lock Type	Table Read (TR)	Record Read (RR)	Table Write (TW)	Record Write (RW)	Table Upgrade (TU)	Record Upgrade (RU)
Algorithm						
Algo_1	2815	2916	7180	7650	7711	7954
Algo_2	27075	27464	6881	7347	7648	7403
Algo_3	10603	10270	9987	10632	11064	10722
Algo_4	3376	2978	7263	7694	7990	7749
Algo_5	575	402	1035	515	1076	521
Algo_6	2404	2298	2497	2361	2593	2392

Table 5.9: The Average Waiting Time in Milliseconds to Obtain a Lock Under Different Algorithm.
Request_num = 1200,
Table Read Time = 40, Record Read Time = 20,
Table Write tιme = 60, Record Write Time = 30,
Table Upgrade Tιme = 100, Record Upgrade Tιme = 50 ,
Interval tιme = 30

Lock Type	Table Read (TR)	Record Read (RR)	Table Write (TW)	Record Write (RW)	Table Upgrade (TU)	Record Upgrade (RU)
Algorithm						
Algo_1	972	1098	2032	2184	2249	2197
Algo_2	20028	20513	1378	1499	1558	1500
Algo_3	4577	4439	4291	4592	4779	4625
Algo_4	1014	1065	2057	2296	2338	2308
Algo_5	229	191	408	177	333	194
Algo_6	282	210	362	238	281	230

Tables 5.7, 5.8 and 5.9 show that in Algorithm 1, which is the reader privilege algorithm, the average time spent to obtain read locks(including table read lock and record read lock) is significantly shorter than for obtaining write locks (including Table Write, Table Upgrade, Record Write and Record Upgrade). This demonstrates that readers take priority over writers in this algorithm.

On the contrast, the average time spent to obtain write locks (including Table Write, Table Upgrade, Record Write and Record Upgrade) is significantly shorter than that for obtaining reader locks in Algorithm 2 which is the writer privilege algorithm. This illustrates that writers have priority over readers. Of particular interest when comparing Algorithms 1 and 2 is that there is little difference in the times for obtaining write locks, but vastly different times for obtaining shared locks. This indicates that, at least in some circumstances, providing shared lock priority has a small cost for exclusive lock requestors.

Since algorithm 3 is a fair algorithm for readers and writers, we can observe from the tables that the time spent to obtain reader locks and writer locks are similar. The cost of a very strict fairness appears to be that everyone must wait longer for their locks, even though the times are very comparable across request types.

In comparing with Algorithm 3, Algorithm 4 allows more readers to enter in the same lock arbitration and read concurrently, thus the time spent for obtaining read locks in Algorithm 4 is much shorter than in Algorithm 3. Of perhaps even greater significance is the observation that the waiting times for writers are also significantly less for Algorithm 4 than for Algorithm 3.

The waiting times for Algorithm 5 are significantly less than the previous four especially for record read, record write and record upgrade requests. This is because the hierarchical locks allow readers and writers to access different records simultaneously. Also, of interest here are the differences in waiting times for different locks. A shared small resource (row) lock is obtained more quickly than exclusive or large resource locks. The longest waiting times were for exclusive large resource (table) locks.

Even though Algorithm 6 also has hierarchical locks and allows the read part of upgrade to share with other waiting reads and intent to reads, the time spent to obtain locks is shorter than the algorithms without hierarchical locks, it still suffers from the additional overhead associated with obtaining the upgrade locks and thus the time spent to obtain locks is larger than in Algorithm 5. Table 5.10 shows the comparison of waiting time for obtaining six types of lock under Algorithm 5, Algorithm 6 and Algorithm 6a(described in the previous section). It illustrates that the average time for obtaining the six types of lock are reduced in Algorithm 6a by comparing to Algorithm 6 because Algorithm 6a allows two upgrades in the same lock arbitration thus improves some efficiency. Algorithm 6a is not as efficient as Algorithm 5 may be due to the arbitration issue. To solve the issue is beyond the scope of current work.

Table 5.10: The Average Waiting Time in Milliseconds to Obtain a Lock Under Algorithm 5, Algorithm 6 and Algorithm 6a.

Request_num = 1200,
Table Reader Time = 40, Record Record Tıme = 20,
Table Write tıme = 60, Record Write Tıme = 30,
Table Upgrade Time = 100, Record Upgrade Tıme = 50 ,
Interval tıme = 10

Lock Type	**Table Read (TR)**	**Record Read (RR)**	**Table Write (TW)**	**Record Write (RW)**	**Table Upgrade (TU)**	**Record Upgrade (RU)**
Algorithm						
Algo_5	2587	1615	6072	3306	6695	3335
Algo_6	8149	7850	7963	8196	8664	8250
Algo_6a	6465	6143	6675	6488	6832	6546

5.5 Analysis of the Execution Results for Three Versions

From Tables 5.1 through 5.4, we can observe when the requests number is small, the efficiency of all algorithms is approximately the same. Neither Algorithm 5 nor Algorithm 6 has any advantage in efficiency. In addition, if the requests number is small, Algorithm 5 and Algorithm 6 have greater overhead due to activities such as obtainıng the intentional lock and upgrade lock. But when the number of requests grows larger, Algorithm 4 is the most efficient of the algorithms without intent to read and intent to write locks because this algorithm allows multiple readers to take priority over the writers as long as they arrive in the same shared lock arbitration. Even though Algorithms 1, 2 and 4 provide similar results when running the verbose version, Algorithms 1 and 2 suffer from the starvation and fairness problem, so we conclude that Algorithm 4 is the most efficient, starvation free in the first category of experiment. Among all six algorithms, Algorithm 5 and Algorithm 6 are better than Algorithm 4 in terms of concurrency and efficiency when the number of requests is big because they provide granularity locks and let the readers and the writers access different resources of the database in parallel. Algorithm 6 is less efficient than Algorithm 5 because of the additional overhead associated with the upgrade locks such as only allow one upgrade in a lock arbitration which offsetting some of the gains due to hierarchical locking and causing greatly reduce the concurrency and efficiency. Although Algorithm 6a improves Algorithm 6 by allowing two upgrades in the same lock arbitration, it still cannot maximize the concurrency as Algorithm 5. The tradeoff is that Algorithm 6 is deadlock free, not suffering from the possibility of deadlock, as discussed in section 2.6.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

6.1 Analytic Conclusion

From our analysis the algorithms, we conclude that the first five algorithms (Reader Privilege, Writer Privilege, Fair Reader and Writer, Fair and Efficient Readers and Writers, and Fair and Efficient Readers and Writers with Intent to Read and Write) can have deadlock when two or more transactions attempt to first read a resource then later write the same resource without releasing the read locks. Of course, deadlock can be avoided by requiring that either long write locks or that read locks must be released before requesting write locks.

6.2 Experimental Conclusions

Also, from our simulation experiments, we derive the following conclusions:

Writer starvation can occur with the Reader Privilege Algorithm. This problem happens when the readers enter a database continuously, since once they obtain the privilege they will never release the lock to the waiting writers. Thus, this algorithm makes the solution for the reader and writer problem unfair and potentially leads to writer starvation. Conversely, reader starvation can occur with the Writer Privilege Algorithm. This problem happens when the writers enter a database continuously, and once they obtain the privilege to execute they will never release the lock to the waiting readers. This is also an unfair solution because of the starvation problem. In spite of the drawbacks of unfairness and starvation of these two solutions, from an efficiency point of view, they appear to be more efficient than the fair reader and writer algorithms if there are only a limited number of reader and writer requests. The reason is that for the reader privilege algorithm, the readers block and defer the writers and thus maximize the concurrency among readers; and that for the writer privilege algorithm, concurrency among writers was increased by deferring the reader processes until all writer processes were completed.

The advantage of the fair reader and writer algorithm is that it can guarantee absolute first-come-first-served ordering without starving either the readers or the writers -- the drawback is that it is significantly less efficient in many circumstances. If there are one or more writers interleaving with multiple readers, only the continuous sequence of readers between a pair of writers can read simultaneously so that concurrency of readers is diminished.

We demonstrated that our enhanced fair and efficient reader and writer algorithm was starvation free and could achieve efficiency and fairness. This algorithm combines the advantages of the previous three algorithms and overcomes their drawbacks. This algorithm not only provided high degree of concurrency and efficiency for readers but also gives a fair chance to the writers, which are interleaved with the readers. The fairness is nearly the same as the fair reader and writer algorithm when all readers read concurrently with equal durations. There may be small delays for writers when reads are not completely concurrent.

In addition to our enhanced fair and efficient algorithm, we also proved that our fair and efficient reader and writer algorithm with intent to read and intent to write lock can further improve the efficiency when the volume of requests to access a database is large. It gains much more concurrency than algorithms that do not have intentional locks by allowing record readers and writers to access the database where there are no conflicts rather than using fully exclusive access across the database or a table of the database. Even though there is some overhead in obtaining the intentional locks, it appears that, at least in some circumstances, the additional overhead is more than compensated by increased parallelism in reads.

Finally we added an upgrade lock into our enhanced fair and efficient algorithm with intent to read and intent to write locks. Even though the efficiency is not improved when compared to the fair and efficient readers and writers with intent to read and intent to write algorithm, the deadlock problem that all the previous five algorithms suffer from can be completely avoided. Thus, there exists a tradeoff between using the upgrade lock and not using the upgrade lock.

Not using the upgrade lock can lead to better efficiency at least in some circumstances, but leaves the potential deadlock problem when more than one transaction first reads from a database then sequentially wants to write to the database. By using the upgrade lock, avoids potential deadlock problem but is less efficient due to the fact that the algorithm allows only one upgrader in the same arbitration and blocks the subsequent processes followed by the second upgrader. Algorithm 6a(described in Section 5.3.1) provides a better solution but still not resolve the problem. We believe that the separate upgrade queue(s) actually denimishes by making additional pools of concurrent locks.

6.3 Future Work

One of future work in this area would be developing an alternate algorithm for algorithm 6 which not only avoiding potential deadlock problem but also increasing the reasonable degree of concurrency and efficiency.

Another future work in this area including providing a better GUI for our test bed and testing these algorithms over a wide large benchmarks.

The third area of research could be construction of benchmark that provides data points including hierarchical and sequential locking operations.

Another issue not addressed by current research is dealing with situation where faults have either caused processes to not release locks or caused two or more processes to hold incompatible locks on the same resource.

APPENDICES

APPENDIX A

PROGRAMING SOURCE CODE OF READER AND WRITER PROBLEM

The following material is the Java source code of Verbose implementation of reader and writer problem using semaphore.

1. **RW_Server.java**
2. **RW_Server_1.java**
3. **RW_Server_2.java**
4. **RW_Server_3.java**
5. **RW_Server_4.java**
6. **I_RW_Server_5.java**
7. **I_RW_U_Server_6.java**

```
/********************************************************************************************************************
File
RW_Server java

Description

The RW_Server class serves as the benchmark for all six algorithms  Users can
choose how many requests to be run  Those requests will be translated into
table read request, record read request, table write request, record write
request, table upgrade request and record upgrade request by a random number
generator  Users are also allowed to choose the time duration for table read,
table write, table upgrade, record read, record write, record upgrade as well
as the interval time between the requests   The various requests and time
durations will be the test input data for all six algorithms for the reader
and writer problems  The time spent by each algorithm is obtained for
comparison

Note the program uses the screen as the STDOUT and the keyboard as the STDIN
Exessive output is used for keeping track of the order of the execution of
the program and calculating the average waiting time for each lock type under
the different algorithms

Author
Mei Li

 Date
 April 24, 2004
********************************************************************************************************************/

import java io *,
import java util *,

public class RW_Server
{
 /*------------------------------------------------------------------------------------------------------------------
  Function
  convertStrToInt

  Description
  This is a helper function that converts a string to an integer

  Parameters
  str - the string to be converted into an integer

  Return
  An integer   Error is returned if the string is not in valid number format such as when the string
  contains any character that is not numeric)
  ----------------------------------------------------------------------------------------------------------------*/
 private
 static int convertStrToInt(String str)
 {
  int anInt,

  try
  {
   anInt = Integer parseInt(str),      // convert string to an integer
  }
  catch (NumberFormatException e)
  {
```

```
    anInt = ERROR,
  }

  if (anInt == ZERO || anInt == ERROR)
  {
    System out println("Your input \"" + str + "\" is invalid!"),
    System out println("Please make sure your input is in the correct "
                  + "number format and also greater than 0!"),
  }

  return anInt,
}

/*-----------------------------------------------------------------------------------------------------------------------------------------
 Function·
 running

 Description
 This function generates the request types and the record numbers that the
 requests want to access  These requests includes table reader, table writer,
 table upgrader, record reader, record writer and record upgrader  The
 function specifies the interval time between each request and the access time
 for the six requests  Then, the function calls the six algorithm
 implementation functions

 Parameters
 num_requests - the number of requests the user wants to run
 br - a buffereReader object that is use to get input from the user
 ----------------------------------------------------------------------------------------------------------------------------------------*/
private
static void running(int num_requests, BufferedReader br)
throws IOException
{
  String str,
  // set up a repeatable random number generator
  Random rand = new Random(888),
  // array of record numbers the requests will access
  int recNum[] = new int[num_requests],
  // array of the types of requests(record reader, table writer, etc )
  int threadType[] = new int[num_requests],

  /* The following loop generates table read, table write, table upgrade,
   record read, record write, record upgrade requests randomly and it also
   randomly specifies which of the records or the table the readers writers
   and upgraders try to access   Assume there are 5 records in a table */
  for (int i = 0, i < num_requests, i++) {
    // generates a record number between 0 and 5, inclusively
    recNum[i] = rand nextInt(6),
    // generates a request type between 0 and 2, inclusively
    threadType[i] = rand nextInt(3),
  }

  int T_readTime = 0,        // the access time for table read
  int R_readTime = 0,        // the access time for record read
  int T_writeTime = 0,       // the access time for table write
  int R_writeTime = 0,       // the access time for record write
  int interval = 0,          // the interval time between each request

  do
  {
    System out print("Please enter the access time for table read  "),
    str = br readLine(),
```

```
  T_readTıme = convertStrToInt(str),
}
while (T_readTıme == ERROR || T_readTıme == ZERO),

do
{
  System out prınt("Please enter the access tıme for record read  "),
  str = br readLıne(),

  R_readTıme = convertStrToInt(str),
}
while (R_readTıme == ERROR || R_readTıme == ZERO),

do
{
  System out prınt("Please enter the access tıme for table wrıte  "),
  str = br readLıne(),

  T_wrıteTıme = convertStrToInt(str),
}
while (T_wrıteTıme == ERROR || T_wrıteTıme == ZERO),

do
{
  System out prınt("Please enter the access tıme for record wrıte  "),
  str = br readLıne(),

  R_wrıteTıme = convertStrToInt(str),
} while (R_wrıteTıme == ERROR || R_wrıteTıme == ZERO),

do
{
  System out prınt("Please enter the ınterval tıme  "),
  str = br readLıne(),

  ınterval = convertStrToInt(str),
} while (ınterval == ERROR || ınterval == ZERO),

// call the ımplementatıon functıon of algorıthm_1
RW_Server_1 RW_Server_1_Maın(threadType,
                          recNum,
                          num_requests,
                          T_readTıme,
                          R_readTıme,
                          T_wrıteTıme,
                          R_wrıteTıme,
                          ınterval),

// call the ımplementatıon functıon of algorıthm_2
RW_Server_2 RW_Server_2_Maın(threadType,
                          recNum,
                          num_requests,
                          T_readTıme,
                          R_readTıme,
                          T_wrıteTıme,
                          R_wrıteTıme,
                          ınterval),

// call the ımplementatıon functıon of algorıthm_3
RW_Server_3 RW_Server_3_Maın(threadType,
                          recNum,
```

```
                              num_requests,
                              T_readTime,
                              R_readTime,
                              T_writeTime,
                              R_writeTime,
                              interval),

  // call the implementation function of algorithm_4
   RW_Server_4 RW_Server_4_Main(threadType,
                               recNum,
                               num_requests,
                               T_readTime,
                               R_readTime,
                               T_writeTime,
                               R_writeTime,
                               interval),

  // call the implementation function of algorithm_5
  I_RW_Server_5 I_RW_Server_5_Main(threadType,
                                  recNum,
                                  num_requests,
                                  T_readTime,
                                  R_readTime,
                                  T_writeTime,
                                  R_writeTime,
                                  interval),

  // call the implementation function of algorithm_6
  I_RW_U_Server_6 I_RW_U_Server_6_Main(threadType,
                                      recNum,
                                      num_requests,
                                      T_readTime,
                                      R_readTime,
                                      T_writeTime,
                                      R_writeTime,
                                      interval),
}


/*-------------------------------------------------------------------------------------------------------------------------------------------
 Function
 main

 Description
 This is the entry point of the program   In this function, the user specifies
 the number of requests via STDIN (the screen)   If the user simply types ENTER
 after the prompt, a default value of 15 will be assumed   This function also
 checks the user input to make sure it is in correct number format and value
 range (i e , the number must be greater than 0 )  This function may throw
 IOException exception  The function finally call the running function to
 convert the requests to the type and record number

 Parameters
 args[] - An array of strings corresponding to the command line arguments

 Return
 None
 ------------------------------------------------------------------------------------------------------------------------------------------*/
public
static void main(String args[])
throws IOException
{
```

```
    BufferedReader br = new BufferedReader(new InputStreamReader(System ın)),
    Strıng str,
    ınt num_requests_default = 15,
    ınt num_requests,

    do
    {
      do
      {
        System out prıntln(""),
        System out prınt("Please enter the number of requests (default = "
                         + num_requests_default + ", q to quıt)  "),
        str = br readLıne(),

        ıf (str equals(""))
          num_requests = num_requests_default,
        else ıf (str equals("q") || str equals("Q"))
          num_requests = QUIT,
        else
          num_requests = convertStrToInt(str),
      }
      whıle (num_requests == ERROR || num_requests == ZERO),

      ıf (num_requests != QUIT)
        runnıng(num_requests, br),
    }
    whıle (num_requests != QUIT),
  }
final statıc ınt QUIT = -1011,
  final statıc ınt ERROR = -1012,
  final statıc ınt ZERO = 0,
}
```

```
/*******************************************************************************************************************
 File
 RW_Server_1 java

 Description
 This is the implementation of the reader privilege algorithm  From the running
 result, reader starvation will be observed

 Author
 Mei Li

 Date
 April 24, 2004
 ******************************************************************************************************************/

import java io *,
import java util *,

/*******************************************************************************************************************
 The node class is used to declare objects that are used to sort the request
 types based on the average lock waiting time
 ******************************************************************************************************************/
class node
{
 long avg,
 int type,

 node()
 {
  avg = -1,
  type = -1,
 }

 node(long a, int t)
 {
  avg = a,
  type = t,
 }
}

public class RW_Server_1
{
 /*-----------------------------------------------------------------------------------------------------------------
  Function
  average

  Description
  It calculates the average of the long integers contained in the arr array

  Parameters
  arr - An array of objects of Object type   These objects contains long
      integers corresponding to the lock waiting times for a request type

  Return
  The average  It returns -1 if the number of objects is 0
 ----------------------------------------------------------------------------------------------------------------*/
 static long average(Object[] arr)
 {
  long total = 0,
  int count = 0,

  for (int i = 0; i < arr length, i++)
```

```
 {
  count++,
  total += ((Long)arr[ı]) longValue(),
 }

 ıf (count == 0)
  return -1,

 return total / count,
}

/*-----------------------------------------------------------------------------------------------------------------------------------
 Functıon
 dısplay

 Descrıptıon
 It dısplays on the screen the waıtıng tımes for a request type, separated
 wıth a space

 Parameters
 arr - An array of objects of Object type   These objects contaıns long
      ıntegers correspondıng to the lock waıtıng tımes for a request type

 Return
 None
 -----------------------------------------------------------------------------------------------------------------------------------*/
statıc voıd dısplay(Object[] array)
{
 for (ınt ı = 0, ı < array length, ı++)
 {
  ıf (ı == array length - 1)
  {
   System out prınt(((Long)array[ı]) longValue()),
  }
  else
  {
   System out prınt(((Long)array[ı]) longValue() + " "),
  }
 }
}

/*-----------------------------------------------------------------------------------------------------------------------------------
 Functıon
 bubbleSort

 Descrıptıon
 It sorts an array of nodes based on the average waıtıng tımes the nodes
 contaın   The sortıng algorıthm of bubble sort ıs used sınce we only have
 a small number (6) of request types to sort

 Parameters
 nodeArray - An array of nodes   These nodes contaıns the average waıtıng tıme
           for a request type and the request type
 sıze - The sıze of the nodeArray array

 Return
 None
 -----------------------------------------------------------------------------------------------------------------------------------*/
statıc voıd bubbleSort(node[] nodeArray, ınt sıze)
{
 node tmp = new node(),
```

```
 for (int i = size-1, i > 0, i--)
 {
  for (int j = 0, j < i, j++)
  {
   if (nodeArray[j] avg > nodeArray[j+1] avg)
   {
    tmp avg = nodeArray[j+1] avg,
    tmp type = nodeArray[j+1] type,
    nodeArray[j+1] avg = nodeArray[j] avg,
    nodeArray[j+1] type = nodeArray[j] type,
    nodeArray[j] avg = tmp avg,
    nodeArray[j] type = tmp type,
   }
  }
 }
}

/*-----------------------------------------------------------------------------------------------------------------------------------------
 Function
 RW_Server_1_Main

 Description
 This function generates threads that emulates the table read, record read,
 table write, record write, table upgrade and record upgrade requests   Then
 it starts the threads and wait for threads to terminate   After that, the
 throughput time for this algorithm and the turnaround times (lock waiting
 times) for the various requests are calculated and displayed

 Parameter
 thread_Type - An array of the request types (namely read, writer or
              upgrader)
 rec_Num - The identification number of the record a request tries to access
 num_threads - The number of requests
 table_r_time - The access time of table read
 record_r_time - The access time of record read
 table_w_time - The access time of table write
 record_w_time - The access time of record write
 interval - The interval time between the requests

 Return
 None
-----------------------------------------------------------------------------------------------------------------------------------------*/
public static void RW_Server_1_Main(int[] thread_Type,
                                  int[] rec_Num,
                                  int num_threads,
                                  int table_r_time,
                                  int record_r_time,
                                  int table_w_time,
                                  int record_w_time,
                                  int interval)

{
 int reader_id = 0, writer_id = 0,
 Database_1 db = new Database_1(),
 ArrayList threadArrayList = new ArrayList(),

 for (int i = 0, i < num_threads, i++)
 {
  if (thread_Type[i] == 0)
  {   // readers are added to list
   if (rec_Num[i] == 5) // this is a table level reader
    threadArrayList add(new Reader_1(reader_id++,
```

```
                          db,
                          0,
                          table_r_tιme)),
   else  // thιs ιs a record level reader
     threadArrayLιst add(new Reader_1(reader_ιd++,
                          db,
                          1,
                          record_r_tιme)),
 }
 else ιf(thread_Type[ι] == 1)
 { // wrιters are added to lιst
   ιf (rec_Num[ι] == 5) // thιs ιs a table level wrιter
     threadArrayLιst add(new Wrιter_1(wrιter_ιd++,
                          db,
                          0,
                          0,
                          table_w_tιme)),
   else  //thιs ιs a record level wrιter
     threadArrayLιst add(new Wrιter_1(wrιter_ιd++,
                          db,
                          1,
                          0,
                          record_w_tιme)),
 }
 else
 { // ιf ιt ιs a upgrader, add ιt to the lιst as a wrιter
   ιf (rec_Num[ι] == 5) // thιs ιs a table level upgrader
    threadArrayLιst add(new Wrιter_1(wrιter_ιd++,
                         db,
                         0,
                         1,
                         table_r_tιme + table_w_tιme)),
   else              // thιs ιs a record level upgrader
     threadArrayLιst add(new Wrιter_1(wrιter_ιd++,
                          db,
                          1,
                          1,
                          record_r_tιme + record_w_tιme)),
  }
}

// converts the ArrayLιst contaιnιng the readers and wrιters to an Array
Object[] threadArray = threadArrayLιst toArray();
System out prιntln(""),
System out prιntln("Start runnιng algorιthm_1    "),
Date startDate = new Date(),  // startιng tιme

for (ιnt ι = 0, ι < threadArray length, ι++)
{
 ιf (threadArray[ι] ιnstanceof Reader_1)
 {
   ((Reader_1)threadArray[ι]) start(),
    Break_1 duratιon(ιnterval),// ιnterval
 }
 else
 {
   ((Wrιter_1)threadArray[ι]) start(),
    Break_1 duratιon(ιnterval), // ιnterval
 }
}

// block untιl all threads termιnates
```

```
try
{
  for (ınt ı = 0, ı < threadArray length, ı++)
  {
    ıf (threadArray[ı] ınstanceof Reader_1)
      ((Reader_1)threadArray[ı]) joın(),
    else
      ((Wrıter_1)threadArray[ı]) joın(),
  }
}
catch (InterruptedExceptıon e)
{
  System out prıntln("Interrupted"),
}

// endıng tıme
Date endDate = new Date(),

// calculate and prınt to stdout the tıme spent ın seconds
long tımeDıff = endDate getTıme() - startDate getTıme(),
System out prıntln(""),
System out prıntln("\nTıme spent for algorıthm_1  "
                  + (double)tımeDıff/1000 + " seconds "),

ArrayLıst TR_lıst = new ArrayLıst(),
ArrayLıst RR_lıst = new ArrayLıst(),
ArrayLıst TW_lıst = new ArrayLıst(),
ArrayLıst TU_lıst = new ArrayLıst(),
ArrayLıst RW_lıst = new ArrayLıst(),
ArrayLıst RU_lıst = new ArrayLıst(),

for (ınt ı = 0, ı < threadArray length, ı++)
{
  ıf (threadArray[ı] ınstanceof Reader_1)
  {
    ıf (((Reader_1)threadArray[ı]) getType() == ReaderWrıterType TR)
    {
      TR_lıst add(new Long(
          ((Reader_1)threadArray[ı]) getLockWaıtıngTıme())),
    }
    else
    {
      RR_lıst add(new Long(
          ((Reader_1)threadArray[ı]) getLockWaıtıngTıme())),
    }
  }
  else
  {
    ıf (((Wrıter_1)threadArray[ı]) getType() == ReaderWrıterType TW)
    {
      TW_lıst add(new Long(
          ((Wrıter_1)threadArray[ı]) getLockWaıtıngTıme())),
    }
    else ıf (((Wrıter_1)threadArray[ı]) getType() == ReaderWrıterType TU)
    {
      TU_lıst add(new Long(
          ((Wrıter_1)threadArray[ı]) getLockWaıtıngTıme())),
    }
    else ıf (((Wrıter_1)threadArray[ı]) getType() == ReaderWrıterType RW)
    {
      RW_lıst add(new Long(
          ((Wrıter_1)threadArray[ı]) getLockWaıtıngTıme())),
```

```
    }
    else
    {
      RU_list add(new Long(
          ((Writer_1)threadArray[i]) getLockWaitingTime())),
    }
  }
}

Object[] TR_arr = TR_list toArray(),
Object[] RR_arr = RR_list toArray(),
Object[] TW_arr = TW_list toArray(),
Object[] TU_arr = TU_list toArray(),
Object[] RW_arr = RW_list toArray(),
Object[] RU_arr = RU_list toArray(),

long avgRR = average(RR_arr),
long avgRU = average(RU_arr),
long avgRW = average(RW_arr),
long avgTR = average(TR_arr),
long avgTU = average(TU_arr),
long avgTW = average(TW_arr),

node nodeArray[] = {
    new node(avgTR, ReaderWriterType TR),
    new node(avgRR, ReaderWriterType RR),
    new node(avgTW, ReaderWriterType TW),
    new node(avgTU, ReaderWriterType TU),
    new node(avgRW, ReaderWriterType RW),
    new node(avgRU, ReaderWriterType RU)},

bubbleSort(nodeArray, 6),

System out println("\nThe average times spent in milliseconds to obtain a "
                  + "lock in algorithm_1 \n"),
for (int i = 0, i < 6, i++)
{
  switch (nodeArray[i] type)
  {
    case ReaderWriterType RR
      System out print("RR Average = " + avgRR  + " ["),
      display(RR_arr),
      System out println("]\n"),
      break,
    case ReaderWriterType RU
      System out print("RU Average = " + avgRU  + " ["),
      display(RU_arr),
      System out println("]\n"),
      break,
    case ReaderWriterType RW
      System out print("RW Average = " + avgRW  + " ["),
      display(RW_arr),
      System out println("]\n"),
      break,
    case ReaderWriterType TR
      System out print("TR Average = " + avgTR  + " ["),
      display(TR_arr),
      System out println("]\n"),
      break,
    case ReaderWriterType TU
      System out print("TU Average = " + avgTU + " ["),
      display(TU_arr),
```

```
        System out prıntln("]\n"),
        break,
      case ReaderWrıterType TW
        System out prınt("TW Average = " + avgTW + " ["),
        dısplay(TW_arr),
        System out prıntln("]\n"),
        break,
      default
        System out prıntln("Oop! Somerthıng must be wrong "),
        break,
    }
  }
 }
}

/*****************************************************************************************************************
 Thıs class ıs emulates a physıcal database  It contaıns methods that wıll be
 called by Reader_1 and Wrıter_1 classes
 ****************************************************************************************************************/
class Database_1
{
 /*---------------------------------------------------------------------------------------------------------------
  Functıon    Database_1

  Descrıptıon
  Constructor for the Database_1 class  It ınıtıalızes readCount, and the
  semaphores rc and w

  Parameters
  None

  Return
  None
  ---------------------------------------------------------------------------------------------------------------*/
 publıc Database_1()
 {
  readCount = 0,
  rc = new Semaphore_1(1),
  w = new Semaphore_1(1),
 }

 /*---------------------------------------------------------------------------------------------------------------
  Functıon
  startRead

  Descrıptıon
  start a read process accordıng to the reader prevılege algorıthm

  Parameters
  readNum - the ID number of a reader

  Return
  The number of readers that are currently readıng
  ---------------------------------------------------------------------------------------------------------------*/
 publıc ınt startRead(ınt readerNum)
 {
  System out prıntln("reader " + readerNum + " wants to read "),
  rc P(),
  ++readCount,
  ıf (readCount == 1)  // the fırst reader wıll block wrıter
    w P(),
  rc V(),
```

```
  return readCount,
}

/*------------------------------------------------------------------------------------------------------------------------------------
 Function
 endRead

 Description·
 end a read process accordıng to the reader prevılege algorıthm

 Parameters
 readNum - the ID number of a reader

 Return
 The number of readers that are currently readıng
 ------------------------------------------------------------------------------------------------------------------------------------*/
public ınt endRead(ınt readerNum)
{
  rc P(),
  --readCount,
  System out prıntln("reader " + readerNum + " ıs done readıng  Count = " +
                    readCount),
  ıf (readCount == 0) // the last reader wıll unblock wrıter
    w V(),
  rc V(),
  return readCount,
}

/*------------------------------------------------------------------------------------------------------------------------------------
 Functıon
 startWrıte

 Descrıptıon
 start a wrıte process accordıng to the reader prevılege algorıthm

 Parameters
 wrıterNum - the ID number of a wrıter

 Return
 None
 ------------------------------------------------------------------------------------------------------------------------------------*/
public voıd startWrıte(ınt wrıterNum)
{
  System out prıntln("wrıter " + wrıterNum + " wants to wrıte "),
  w P(),
}

/*------------------------------------------------------------------------------------------------------------------------------------
 Functıon
 endWrıte

 Descrıptıon.
 end a wrıte process accordıng to the reader prıvılege algorıthm

 Parameters
 wrıterNum - the ID number of a wrıter

 Return
 None
 ------------------------------------------------------------------------------------------------------------------------------------*/
public voıd endWrıte(ınt wrıterNum)
{
```

```
    System out println("writer " + writerNum + " is done writing "),
    w V(),
  }

  private int readCount, // the number of active readers
  Semaphore_1 rc,       // controls access to readCount
  Semaphore_1 w,        // controls access to the Database_1
}

/*******************************************************************************************************************
 This class defines the reader and writer types and provides methods for
 retrieving the types
 *******************************************************************************************************************/
class ReaderWriterType
{
  private int value,

  ReaderWriterType(int type)
  {
    value = type,
  }

  int getReaderWriterType()
  {
    return value,
  }

  final static int TR = 0x100,
  final static int RR = 0x101,
  final static int TW = 0x102,
  final static int TU = 0x103,
  final static int RW = 0x104,
  final static int RU = 0x105,
}

/*******************************************************************************************************************
 This class calls the methods of Database_1 to start and end a read process
 *******************************************************************************************************************/
class Reader_1 extends Thread
{
 /*-----------------------------------------------------------------------------------------------------------------
  Function
  Reader_1

  Description
  Constructor for Reader_1 class. It specifies which reader is reading from
  which database and how long the reading time is

  Parameters
  r - reader ID
  w - the database the reader is reading from
  n - indicates the reader is a table reader or a record reader
  r_time - reader access time

  Return
  None
  -----------------------------------------------------------------------------------------------------------------*/
 public Reader_1(int r, Database_1 w, int n, int r_time)
 {
   readerNum = r,
   db = w,
   read_time = r_time;
```

```
  if (n == 0) // Table Reader
    t_r = new ReaderWriterType(ReaderWriterType TR),
  else      // Record Reader
    t_r = new ReaderWriterType(ReaderWriterType RR),
}

/*--------------------------------------------------------------------------------------------------------------------
 Function
 getType

 Description
 Get the type of the reader (TR or RR)

 Parameters
 None

 Return
 An integer that corresponds to the reader's type
 ------------------------------------------------------------------------------------------------------------------*/
public int getType()
{
  return t_r getReaderWriterType(),
}

/*--------------------------------------------------------------------------------------------------------------------
 Function
 getLockWaitingTime

 Description
 Get the lock waiting time for the reader

 Parameters
 None

 Return
 A long integer that corresponds to the lock waiting time
 ------------------------------------------------------------------------------------------------------------------*/
public long getLockWaitingTime()
{
  return lockWaitingTime,
}

/*--------------------------------------------------------------------------------------------------------------------
 Function
 run

 Description
 This function specifies how a reader thread runs

 Parameters
 None

 Return
 None
 ------------------------------------------------------------------------------------------------------------------*/
public void run()
{
  int c,

  Date start_req = new Date(),
  c = db startRead(readerNum),
```

```
    Date obtain_req = new Date(),

    System out println("reader " + readerNum + " is reading  Count = " + c),
    Break_1 duration(read_time), // read read_time milliseconds

    c = db endRead(readerNum),

    // waiting time for obtaining a reader lock
    lockWaitingTime = obtain_req getTime() - start_req getTime(),
  }

  private Database_1 db,   // the database that the reader tries to access
  private int readerNum,   // identification number of the reader
  private int read_time,   // access time spent by the reader
  private ReaderWriterType t_r,   // the type of the reader (TR or RR)
  private long lockWaitingTime,   // the lock waiting time of the reader
}

/*****************************************************************************************************************
 This class calls the method of Database_1 to start and end a write process
*****************************************************************************************************************/
class Writer_1 extends Thread
{
  /*-----------------------------------------------------------------------------------------------------------------
   Function
   Writer_1

   Description
   Constructor for Writer_1 class  It specifies the type of the process(table
   writer, record writer, table upgrader, record upgrader)and its duration time

   Parameters
   w - writer ID
   d - the database the writer and upgrader are writing to
   n - indicates table writer or record writer
   u - indicates an upgrader or not
   w_time - writer access time

   Return
   None
   ---------------------------------------------------------------------------------------------------------------*/
  public Writer_1(int w, Database_1 d, int n, int u, int w_time)
  {
    writerNum = w,
    db = d,
    write_time = w_time,

    if (n == 0)
    {
      if (u == 0) // Table Writer
        t_w = new ReaderWriterType(ReaderWriterType TW),
      else       // Table Upgrader
        t_w = new ReaderWriterType(ReaderWriterType TU),
    }
    else
    {
      if (u == 0) // Record Writer
        t_w = new ReaderWriterType(ReaderWriterType RW),
      else       // Record Upgrader
        t_w = new ReaderWriterType(ReaderWriterType RU),
    }
  }
```

```
/*---------------------------------------------------------------------------------------------------------------------------
 Function
 getType

 Description
 Get the type of the writer (TW, RW, TU or RU)

 Parameters
 None

 Return
 An integer that corresponds to the writer's type
 ---------------------------------------------------------------------------------------------------------------------------*/
public int getType()
{
  return t_w getReaderWriterType(),
}

/*---------------------------------------------------------------------------------------------------------------------------
 Function
 getLockWaitingTime

 Description
 Get the lock waiting time for the writer

 Parameters
 None

 Return
 A long integer that corresponds to the lock waiting time
 ---------------------------------------------------------------------------------------------------------------------------*/
public long getLockWaitingTime()
{
  return lockWaitingTime,
}

/*---------------------------------------------------------------------------------------------------------------------------
 Function
 run

 Description
 This function specifies how a writer thread runs

 Parameters
 None

 Return
 None
----------------------------------------------------------------------------------------------------------------------------*/
public void run()
{
  Date start_req = new Date(), // start timing
  db startWrite(writerNum), // start a write process
  Date obtain_req = new Date(), // end timing

  System out println("writer " + writerNum + " is writing "),
  Break_1 duration(write_time); // writer or upgrader writes write_time milliseconds

  db endWrite(writerNum),

  // the waiting time to obtain a writer lock
```

```
    lockWaitingTime = obtain_req getTime() - start_req getTime(),
  }

  private Database_1 db,            // the database that the writer tries to access
  private int writerNum,            // identification number of the writer
  private int write_time,           // access time spent by the writer
  private ReaderWriterType t_w,  // the type of the writer (TW, TR, TU or RU)
  private long lockWaitingTime,  // the lock waiting time of the reader
}


/*************************************************************************************************************************
 Semaphore_1 class using Java syschronization

*************************************************************************************************************************/
final class Semaphore_1
{
  /*------------------------------------------------------------------------------------------------------------------------------------
   Function
   Semaphore_1

   Description
   default constructor for Semaphore_1

   Parameters
   None

   Return
   None
   ----------------------------------------------------------------------------------------------------------------------------------*/
  public Semaphore_1()
  {
   value = 1,
  }

  /*----------------------------------------------------------------------------------------------------------------------------------
   Function
   Semaphore_1

   Description
   constructor for Semaphore_1

   Parameter
   v - An integer value for the semaphore

   Return
   None
   ---------------------------------------------------------------------------------------------------------------------------------*/
  public Semaphore_1(int v)
  {
   value = v,
  }

  /*------------------------------------------------------------------------------------------------------------------------------------
   Function
   P

   Description
   This function call the wait() to sleep when the value of Semaphore less than
   or equal 0  If the value of Semaphore is a positive number, decrements by 1

   Parameters
```

```
   None

   Return
   None
   ------------------------------------------------------------------------------------------------------------------------------*/
  public synchronized void P()
  {
    while (value <= 0)
    {
      try
      {
        wait(),
      }
      catch (InterruptedException e) {}
    }
    value--,
  }

  /*------------------------------------------------------------------------------------------------------------------------------
   Function
   V

   Description
   This function increments the semaphore value by 1 and call the notify()
   function to wakeup a process that is waiting on the semaphore if it has any

   Parameters
   None

   Return
   None
   ------------------------------------------------------------------------------------------------------------------------------*/
  public synchronized void V()
  {
    ++value,
    notify(),
  }

  private int value,   // the value of the semaphore
}

/*****************************************************************************************************************************
 The class specifies the duration of access time

*****************************************************************************************************************************/
final class Break_1
{
  /*------------------------------------------------------------------------------------------------------------------------------
   Function
   duration

   Description
   The function specifies the duration of access time in milliseconds

   Parameter
   milliseconds - how many milliseconds the access time is

   Return
   None
   ------------------------------------------------------------------------------------------------------------------------------*/
  public static void duration(int milliseconds)
  {
```

```
    try
    {
      Thread sleep(mıllıseconds),
    }
    catch (InterruptedExceptıon e) {}
  }
}
```

```
/*****************************************************************************************************************************
 File
 RW_Server_2 java

 Description
 This is the implementation of the writer privilege algorithm  From the running
 result, reader starvation will be observed

 Author
 Mei Li

 Date
 April 24, 2004
 *****************************************************************************************************************************/

import java io *,
import java util *,

/*****************************************************************************************************************************
 The node class is used to declare objects that are used to sort the request
 types based on the average lock waiting time

*****************************************************************************************************************************/
class node
{
 long avg,
 int type,

 node()
 {
  avg = -1,
  type = -1,
 }

 node(long a, int t)
 {
  avg = a,
  type = t,
 }
}

class RW_Server_2
{
 /*--------------------------------------------------------------------------------------------------------------------------
  Function
  average

  Description
  It calculates the average of the long integers contained in the arr array

  Parameters
  arr - An array of objects of Object type   These objects contains long
      integers corresponding to the lock waiting times for a request type

  Return
  The average  It returns -1 if the number of objects is 0
  --------------------------------------------------------------------------------------------------------------------------*/
 static long average(Object[] arr)
 {
  long total = 0,
  int count = 0,
```

```
  for (int i = 0, i < arr length, i++)
  {
    count++,
    total += ((Long)arr[i]) longValue(),
  }

  if (count == 0)
    return -1,

  return total / count,
}

/*---------------------------------------------------------------------------------------------------------
 Function
 display

 Description
 It displays on the screen the waiting times for a request type, separated
 with a space

 Parameters
 arr - An array of objects of Object type   These objects contains long
      integers corresponding to the lock waiting times for a request type

 Return
 None
 ---------------------------------------------------------------------------------------------------------*/
static void display(Object[] array)
{
  for (int i = 0, i < array length, i++)
  {
    if (i == array length - 1)
    {
      System out print(((Long)array[i]) longValue()),
    }
    else
    {
      System out print(((Long)array[i]) longValue() + " "),
    }
  }
}

 /*---------------------------------------------------------------------------------------------------------

 Function
 bubbleSort

 Description
 It sorts an array of nodes based on the average waiting times the nodes
 contain   The sorting algorithm of bubble sort is used since we only have
 a small number (6) of request types to sort

 Parameters
 nodeArray - An array of nodes   These nodes contains the average waiting time
           for a request type and the request type
 size - The size of the nodeArray array

 Return
 None
 ---------------------------------------------------------------------------------------------------------*/
static void bubbleSort(node[] nodeArray, int size)
{
```

```
  node tmp = new node(),

  for (ınt ı = sıze - 1, ı > 0, ı--)
  {
   for (ınt j = 0, j < ı, j++)
   {
    ıf (nodeArray[j] avg > nodeArray[j + 1] avg)
    {
     tmp avg = nodeArray[j + 1] avg,
     tmp type = nodeArray[j + 1] type,
     nodeArray[j + 1] avg = nodeArray[j] avg,
     nodeArray[j + 1] type = nodeArray[j] type,
     nodeArray[j] avg = tmp avg,
     nodeArray[j] type = tmp type,
    }
   }
  }
}

/*-----------------------------------------------------------------------------------------------------------------------------------
 Functıon
 RW_Server_2_Maın

 Descrıptıon
 Thıs functıon generates threads that emulates the table read, record read,
 table wrıte, record wrıte, table upgrade and record upgrade requests   Then
 ıt starts the threads and waıt for threads to termınate   After that, the
 throughput tıme for thıs algorıthm and the turnaround tımes (lock waıtıng
 tımes) for the varıous requests are calculated and dısplayed

 Parameter
 thread_Type - An array of the request types (namely read, wrıter or
             upgrader)
 rec_Num - The ıdentıfıcatıon number of the record a request tres to access
 num_threads - The number of requests
 table_r_tıme - The access tıme of table read
 record_r_tıme - The access tıme of record read
 table_w_tıme - The access tıme of table wrıte
 record_w_tıme - The access tıme of record wrıte
 ınterval - The ınterval tıme between the requests

 Return
 None
 ----------------------------------------------------------------------------------------------------------------------------------*/
publıc statıc voıd RW_Server_2_Maın(ınt[] thread_Type,
                                    ınt[] rec_Num,
                                    ınt num_threads,
                                    ınt table_r_tıme,
                                    ınt record_r_tıme,
                                    ınt table_w_tıme,
                                    ınt record_w_tıme,
                                    ınt ınterval)

{
  ınt reader_ıd = 0, wrıter_ıd = 0,
  Database_2 db = new Database_2(),
  ArrayLıst threadArrayLıst = new ArrayLıst(),

  for (ınt ı = 0, ı < num_threads, ı++)
  {
   ıf (thread_Type[ı] == 0) // readers are added to lıst
   {
```

```
    if (rec_Num[i] == 5) // this is a table level reader
      threadArrayList add(new Reader_2(reader_id++,
                                  db,
                                  0,
                                  table_r_time)),
    else  // this is a record level reader
      threadArrayList add(new Reader_2(reader_id++,
                                  db,
                                  1,
                                  record_r_time)),
  }
  else if (thread_Type[i] == 1) // writers are added to list
  {
    if (rec_Num[i] == 5) // this is a table level writer
      threadArrayList add(new Writer_2(writer_id++,
                                  db,
                                  0,
                                  0,
                                  table_w_time)),
    else                //this is a record level writer
      threadArrayList add(new Writer_2(writer_id++,
                                  db,
                                  1,
                                  0,
                                  record_w_time)),
  }
  else
  { // if it is a upgrader, add it to the list as a writer
    if (rec_Num[i] == 5) // this is a table level upgrader
      threadArrayList add(new Writer_2(writer_id++,
                                  db,
                                  0,
                                  1,
                                  table_r_time + table_w_time)),
    else                // this is a record level upgrader
      threadArrayList add(new Writer_2(writer_id++,
                                  db,
                                  1,
                                  1,
                                  record_r_time + record_w_time)),
  }
}

// converts the ArrayList containing the readers and writers to an Array
Object[] threadArray = threadArrayList toArray(),
System out println(""),
System out println("\n\nStart running algorithm_2    "),
Date startDate = new Date(), // starting time

for (int i = 0, i < threadArray length, i++)
{
  if (threadArray[i] instanceof Reader_2)
  {
    ((Reader_2)threadArray[i]) start(),
    Break_2 duration(interval),
  }
  else
  {
    ((Writer_2)threadArray[i]) start(),
    Break_2 duration(interval),
  }
}
```

```
}
try {
  for (int i = 0, i < threadArray length, i++)
  {
    if (threadArray[i] instanceof Reader_2)
      ((Reader_2)threadArray[i]) join(),
    else
      ((Writer_2)threadArray[i]) join(),
  }
}
catch (InterruptedException e)
{
  System out println("Interrupted"),
}

// ending time
Date endDate = new Date(),

// calculate and print to stdout the time spent in seconds
long timeDiff = endDate getTime() - startDate getTime(),
System out println(""),
System out println("\nTime spent for algorithm_2  " +
                  (double) timeDiff / 1000 + " seconds "),

ArrayList TR_list = new ArrayList(),
ArrayList RR_list = new ArrayList(),
ArrayList TW_list = new ArrayList(),
ArrayList TU_list = new ArrayList(),
ArrayList RW_list = new ArrayList(),
ArrayList RU_list = new ArrayList(),

for (int i = 0, i < threadArray length, i++)
{
  if (threadArray[i] instanceof Reader_2)
  {
    if (((Reader_2)threadArray[i]) getType() == ReaderWriterType TR)
    {
      TR_list add(new Long(
          ((Reader_2)threadArray[i]) getLockWaitingTime())),
    }
    else
    {
      RR_list add(new Long(
          ((Reader_2)threadArray[i]) getLockWaitingTime())),
    }
  }
  else
  {
    if (((Writer_2)threadArray[i]) getType() == ReaderWriterType TW)
    {
      TW_list add(new Long(
          ((Writer_2)threadArray[i]) getLockWaitingTime())),
    }
    else if (((Writer_2)threadArray[i]) getType() == ReaderWriterType TU)
    {
      TU_list add(new Long(
          ((Writer_2)threadArray[i]) getLockWaitingTime())),
    }
    else if (((Writer_2)threadArray[i]) getType() == ReaderWriterType RW)
    {
      RW_list add(new Long(
          ((Writer_2)threadArray[i]) getLockWaitingTime())),
    }
```

```
      else
      {
        RU_list add(new Long(
            ((Writer_2)threadArray[i]) getLockWaitingTime())),
      }
    }
  }

  Object[] TR_arr = TR_list toArray(),
  Object[] RR_arr = RR_list toArray(),
  Object[] TW_arr = TW_list toArray(),
  Object[] TU_arr = TU_list toArray(),
  Object[] RW_arr = RW_list toArray(),
  Object[] RU_arr = RU_list toArray(),

  long avgRR = average(RR_arr),
  long avgRU = average(RU_arr),
  long avgRW = average(RW_arr),
  long avgTR = average(TR_arr),
  long avgTU = average(TU_arr),
  long avgTW = average(TW_arr),

  node nodeArray[] = {
      new node(avgTR, ReaderWriterType TR),
      new node(avgRR, ReaderWriterType RR),
      new node(avgTW, ReaderWriterType TW),
      new node(avgTU, ReaderWriterType TU),
      new node(avgRW, ReaderWriterType RW),
      new node(avgRU, ReaderWriterType RU)},

  bubbleSort(nodeArray, 6),

  System out println("\nThe average times spent in milliseconds to obtain a "
                  + "lock in algorithm_2 \n"),
  for (int i = 0, i < 6, i++)
  {
    switch (nodeArray[i] type)
    {
      case ReaderWriterType RR
        System out print("RR Average = " + avgRR + " ["),
        display(RR_arr),
        System out println("]\n"),
        break;
      case ReaderWriterType RU
        System out print("RU Average = " + avgRU + " ["),
        display(RU_arr),
        System out println("]\n"),
        break,
      case ReaderWriterType RW
        System out print("RW Average = " + avgRW + " ["),
        display(RW_arr),
        System out println("]\n"),
        break,
      case ReaderWriterType TR
        System out print("TR Average = " + avgTR + " ["),
        display(TR_arr),
        System out println("]\n"),
        break,
      case ReaderWriterType TU
        System out print("TU Average = " + avgTU + " ["),
        display(TU_arr),
        System out println("]\n"),
```

```
          break,
        case ReaderWriterType TW
          System out print("TW Average = " + avgTW + " ["),
          display(TW_arr);
          System out println("]\n"),
          break,
        default
          System out println("Oop! Somerthing must be wrong "),
          break,

      }
    }
  }
}

/**********************************************************************************************************
 This class emulates a physical database  It contains methods that will be
 called by Reader_2 and Writer_2 classes

**********************************************************************************************************/
class Database_2
{
 /*-------------------------------------------------------------------------------------------------------
  Function    Database_2

  Description
  Constructor for the Database_2 class  It initializes readCount, writecCount
  and the semaphores r, rc, wc, pr, and w

  Parameters
  None

  Return
  None
  -------------------------------------------------------------------------------------------------------*/
 Database_2()
 {
  readCount = 0,
  writeCount = 0,
  r = new Semaphore_2(1),
  rc = new Semaphore_2(1),
  wc = new Semaphore_2(1),
  pr = new Semaphore_2(1);
  w = new Semaphore_2(1),
 }

 /*-------------------------------------------------------------------------------------------------------
  Function.
  startRead

  Description.
  start a read process according to the writer privilege algorithm

  Parameters
  readNum - the ID number of a reader

  Return
  The number of readers that are currently reading
  -------------------------------------------------------------------------------------------------------*/
 int startRead(int readerNum)
 {
  System out println("reader " + readerNum + " wants to read."),
```

```
  pr P(),        // requests pre_read
  r P(),         // requests read semaphore
  rc P(),        // requests reader count
  ++readCount,
  if (readCount == 1) // the first reader blocks writer
    w P(),       // blocks writers
  rc V(),        // release reader count
  r V(),         // release read semaphore

  pr V(), // release pre_read semaphore
  return readCount,
}

/*-----------------------------------------------------------------------------------------------------------------------------
  Function
  endRead

  Description
  end a read process according to the writer privilege algorithm

  Parameters
  readNum - the ID number of a reader

  Return
  The number of readers that are currently reading
  -----------------------------------------------------------------------------------------------------------------------------*/
int endRead(int readerNum)
{
  rc P(),   // requests the reader count
  --readCount,
  System out println("reader " + readerNum + " is done reading  Count = " +
            readCount),

  if (readCount == 0) // The last reader will unblock the writer
    w V(),

  rc V(),   // releases the reader count

  return readCount,
}

/*-----------------------------------------------------------------------------------------------------------------------------
  Function
  startWrite

  Description
  start a write process according to the writer previlege algorithm

  Parameters
  writerNum - the ID number of a writer

  Return
  None
  -----------------------------------------------------------------------------------------------------------------------------*/
void startWrite(int writerNum)
{
  System out println("writer " + writerNum + " wants to write "),
  wc P(),  // requests the writer count
  ++writeCount,
  if (writeCount == 1) // the first writer locks the read semaphore
    r P(),
  wc V(),  // release the writer count
```

```
    w P(),   // release the database
  }

  /*-----------------------------------------------------------------------------------------------------------------------------------
   Function
   endWrite

   Description
   end a write process according to the writer privilege algorithm

   Parameters
   writerNum - the ID number of a writer

   Return
   None
   -----------------------------------------------------------------------------------------------------------------------------------*/
  void endWrite(int writerNum)
  {
    System out println("writer " + writerNum + " is done writing "),
    w V(),
    wc P(),
    --writeCount,
    if (writeCount == 0)   // The last writer releases the read semaphore
      r V(),
    wc V(),                // release the writer count
  }

  private int readCount,  // the number of active readers
  private int writeCount, // the number of active writers
  Semaphore_2 r,          // control access to reader
  Semaphore_2 rc,         // control access to reader count
  Semaphore_2 wc,         // control access to writer count
  Semaphore_2 pr,         // control access to pre_read
  Semaphore_2 w,          // controls access to the database
}

/*****************************************************************************************************************************
 This class defines the reader and writer types and provides methods for
 retrieving the types

*****************************************************************************************************************************/
class ReaderWriterType
{
  private int value,

  ReaderWriterType(int type)
  {
    value = type,
  }

  int getReaderWriterType()
  {
    return value,
  }

  final static int TR = 0x100,
  final static int RR = 0x101,
  final static int TW = 0x102,
  final static int TU = 0x103,
  final static int RW = 0x104,
  final static int RU = 0x105,
}
```

```
/*****************************************************************************************************************
 This class calls the method of Database_2 to start and end a read process
*****************************************************************************************************************/
class Reader_2 extends Thread
{
 /*-----------------------------------------------------------------------------------------------------------------
  Function
  Reader_2

  Description
  Constructor for Reader_2 class  It specifies which reader is reading from
  which database and how long the reading time is

  Parameters
  r - reader ID
  db - the database the reader is reading from
  n - indicates the reader is a table reader or a record reader
  r_time - reader access time

  Return
  None
  ---------------------------------------------------------------------------------------------------------------*/
 Reader_2(int r, Database_2 db, int n, int r_time)
 {
  readerNum = r,
  server = db,
  readTime = r_time,

  if (n == 0)   // Table Reader
    t_r = new ReaderWriterType(ReaderWriterType TR),
  else          // Record Reader
    t_r = new ReaderWriterType(ReaderWriterType RR),
 }

 /*-----------------------------------------------------------------------------------------------------------------
  Function
  getType

  Description
  Get the type of the reader (TR or RR)

  Parameters
  None

  Return
  An integer that corresponds to the reader's type
  ---------------------------------------------------------------------------------------------------------------*/
 public int getType()
 {
  return t_r getReaderWriterType(),
 }

 /*-----------------------------------------------------------------------------------------------------------------
  Function
  getLockWaitingTime

  Description
  Get the lock waiting time for the reader

  Parameters
  None
```

```
 Return
 A long ınteger that corresponds to the lock waıtıng tıme
 ---------------------------------------------------------------------------------------------------------------------------------*/
public long getLockWaıtıngTıme()
{
  return lockWaıtıngTıme,
}

/*---------------------------------------------------------------------------------------------------------------------------------
 Functıon
 run

 Descrıptıon
 Thıs functıon specıfies how a reader thread runs

 Parameters
 None

 Return
 None
 ---------------------------------------------------------------------------------------------------------------------------------*/
public voıd run()
{
  ınt c,

  Date start_req = new Date();
  c = server startRead(readerNum),
  Date obtaın_req = new Date(),

  System out prıntln("reader " + readerNum + " ıs readıng  Count = " + c),
  Break_2.duratıon(readTıme), // read read_tıme mıllıseconds

  c = server endRead(readerNum),

  // the waıtıng tıme to obtaın a reader locl
  lockWaıtıngTıme = obtaın_req getTıme() - start_req getTıme(),
}

prıvate Database_2 server,       // the database that the reader trıes to access
prıvate ınt readerNum,           // ıdentıficatıon number of the reader
prıvate ınt readTıme,            // access tıme spent by the reader
private ReaderWriterType t_r;  // the type of the reader (TR or RR)
prıvate long lockWaıtıngTıme,  // the lock waıtıng tıme of the reader
}

/******************************************************************************************************************************
 Thıs class calls the method of Database_2 to start and end a wrıte process
 ******************************************************************************************************************************/
class Wrıter_2 extends Thread
{
 /*---------------------------------------------------------------------------------------------------------------------------------
  Functıon.
  Wrıter_2

  Descrıptıon
  Constructor for Wrıter_2 class  It specıfies the type of the process(table
  wrıter, record wrıter, table upgrader, record upgrader)and ıts duratıon tıme

  Parameters
  w - wrıter ID
  db - the database the wrıter and upgrader are wrıtıng to
```

```
 n - indicates table writer or record writer
 u - indicates an upgrader or not
 w_time - writer access time

 Return
 None
 ---------------------------------------------------------------------------------------------------------------------------*/
Writer_2(int w, Database_2 db, int n, int u, int w_time)
{
  writerNum = w,
  server = db,
  writeTime = w_time,

  if (n == 0)
  {
    if (u == 0)
      t_w = new ReaderWriterType(ReaderWriterType TW),
    else
      t_w = new ReaderWriterType(ReaderWriterType TU),
  }
  else
  {
    if (u == 0)
      t_w = new ReaderWriterType(ReaderWriterType RW),
    else
      t_w = new ReaderWriterType(ReaderWriterType RU),
  }
}

/*---------------------------------------------------------------------------------------------------------------------------
 Function
 getType

 Description
 Get the type of the writer (TW, RW, TU or RU)

 Parameters
 None

 Return
 An integer that corresponds to the writer's type
 ---------------------------------------------------------------------------------------------------------------------------*/
public int getType()
{
  return t_w getReaderWriterType(),
}

/*---------------------------------------------------------------------------------------------------------------------------
 Function
 getLockWaitingTime

 Description
 Get the lock waiting time for the writer

 Parameters
 None

 Return
 A long integer that corresponds to the lock waiting time
 ---------------------------------------------------------------------------------------------------------------------------*/
public long getLockWaitingTime()
{
```

```
    return lockWaitingTime,
  }

  /*------------------------------------------------------------------------------------------------------------------------
   Function
   run

   Description
   This function specifies how a writer thread runs

   Parameters
   None

   Return
   None
   ------------------------------------------------------------------------------------------------------------------------*/
  public void run()
  {
    Date start_req = new Date(),
    server startWrite(writerNum), // start a write process
    Date obtain_req = new Date(),

    System out println("writer " + writerNum + " is writing "),
    Break_2 duration(writeTime), // write write_time milliseconds

    server endWrite(writerNum),

    // waiting time to obtain a write lock
    lockWaitingTime = obtain_req getTime() - start_req getTime(),
  }

  private Database_2 server,     // the database that the writer tries to access
  private int writerNum,         // identification number of the writer
  private int writeTime,         // access time spent by the writer
  private ReaderWriterType t_w,  // the type of the writer (TW, TR, TU or RU)
  private long lockWaitingTime,  // the lock waiting time of the reader
}

/**************************************************************************************************************************
 Semaphore class using Java synchronization

**************************************************************************************************************************/
final class Semaphore_2
{
  /*------------------------------------------------------------------------------------------------------------------------
   Function
   Semaphore_2

   Description
   default constructor for Semaphore_2

   Parameters
   None

   Return
   None
   ------------------------------------------------------------------------------------------------------------------------*/
  Semaphore_2()
  {
    value = 1,
  }
```

```
/*-------------------------------------------------------------------------------------------------------------------------------------
 Function
 Semaphore_2

 Description
 constructor for Semaphore_2

 Parameter
 v - An integer value for the semaphore

 Return
 None
 -------------------------------------------------------------------------------------------------------------------------------------*/
Semaphore_2(int v)
{
  value = v,
}

/*-------------------------------------------------------------------------------------------------------------------------------------
 Function
 P

 Description
 This function call the wait() to sleep when the value of Semaphore less than
 or equal 0  If the value of Semaphore is a positive number, decrements by 1

 Parameters
 None

 Return
 None
 -------------------------------------------------------------------------------------------------------------------------------------*/
public synchronized void P()
{
  while (value <= 0)
  {
    try
    {
      wait(),
    }
    catch (InterruptedException e) {}
  }
  value--,
}

/*-------------------------------------------------------------------------------------------------------------------------------------
 Function·
 V

 Description
 This function increments the semaphore value by 1 and call the notify()
 function to wakeup a process that is waiting on the semaphore if it has any

 Parameters
 None

 Return
 None
   ----------------------------------------------------------------------------------------------------------------------------------*/
public synchronized void V()
{
  ++value,
```

```
    notify(),
  }

  private int value,   // the value of the semaphore
}

/***********************************************************************************************************************
 The class specifies the duration of access time

***********************************************************************************************************************/
final class Break_2
{
  /*-----------------------------------------------------------------------------------------------------------------------------------------
   Function
   duration

   Description
   The function specifies the duration of access time in milliseconds

   Parameter
   milliseconds - how many milliseconds the access time is

   Return
   None
   -----------------------------------------------------------------------------------------------------------------------------------------*/
  public static void duration(int milliseconds)
  {
    try
    {
      Thread sleep(milliseconds),
    }
    catch (InterruptedException e) {}
  }
}
```

```
/**********************************************************************************************************************
 File
 RW_Server_3 java

 Description
 This is the implementation of the fair reader and writer algorithm
 From the running result, FIFO order will be observed

 Author
 Mei Li

 Date
 April 24, 2004

**********************************************************************************************************************/

import java io *,
import java util *,

/**********************************************************************************************************************
 The node class is used to declare objects that are used to sort the request
 types based on the average lock waiting time

**********************************************************************************************************************/
class node
{
  long avg,
  int type,

  node()
  {
    avg = -1,
    type = -1,
  }

  node(long a, int t)
  {
    avg = a,
    type = t,
  }
}

public class RW_Server_3
{
  /*----------------------------------------------------------------------------------------------------------------------
   Function
   average

   Description
   It calculates the average of the long integers contained in the arr array

   Parameters
   arr - An array of objects of Object type   These objects contains long
         integers corresponding to the lock waiting times for a request type

   Return
   The average  It returns -1 if the number of objects is 0
  ----------------------------------------------------------------------------------------------------------------------*/
  static long average(Object[] arr)
  {
    long total = 0,
    int count = 0,
```

```
  for (ınt ı = 0, ı < arr length, ı++)
  {
    count++,
    total += ((Long)arr[ı]) longValue(),
  }

  ıf (count == 0)
    return -1,

  return total / count,
}

/*-----------------------------------------------------------------------------------------------------------------------------------
 Functıon
 dısplay

 Descrıptıon
 It dısplays on the screen the waıtıng tımes for a request type, separated
 wıth a space

 Parameters
 arr - An array of objects of Object type   These objects contaıns long
      ıntegers correspondıng to the lock waıtıng tımes for a request type

 Return
 None
 -----------------------------------------------------------------------------------------------------------------------------------*/
statıc voıd dısplay(Object[] array)
{
  for (ınt ı = 0, ı < array length, ı++)
  {
    ıf (ı == array length - 1)
    {
      System out prınt(((Long) array[ı]) longValue()),
    }
    else
    {
      System out prınt( ( (Long) array[ı]) longValue() + " "),
    }
  }
}

/*-----------------------------------------------------------------------------------------------------------------------------------
 Functıon
 bubbleSort

 Descrıptıon
 It sorts an array of nodes based on the average waıtıng tımes the nodes
 contaın   The sortıng algorıthm of bubble sort ıs used sınce we only have
 a small number (6) of request types to sort

 Parameters
 nodeArray - An array of nodes   These nodes contaıns the average waıtıng tıme
          for a request type and the request type
 sıze - The sıze of the nodeArray array

 Return
 None
 -----------------------------------------------------------------------------------------------------------------------------------*/
statıc voıd bubbleSort(node[] nodeArray, ınt sıze)
{
```

```
  node tmp = new node(),

  for (int i = size - 1, i > 0, i--)
  {
    for (int j = 0, j < i, j++) {
      if (nodeArray[j] avg > nodeArray[j + 1] avg)
      {
        tmp avg = nodeArray[j + 1] avg,
        tmp type = nodeArray[j + 1] type,
        nodeArray[j + 1] avg = nodeArray[j] avg,
        nodeArray[j + 1] type = nodeArray[j] type,
        nodeArray[j] avg = tmp avg,
        nodeArray[j] type = tmp type,
      }
    }
  }
}

/*-------------------------------------------------------------------------------------------------------------------------------------------
 Function
 RW_Server_3_Main

 Description
 This function generates threads that emulates the table read, record read,
 table write, record write, table upgrade and record upgrade requests   Then
 it starts the threads and wait for threads to terminate   After that, the
 throughput time for this algorithm and the turnaround times (lock waiting
 times) for the various requests are calculated and displayed

 Parameter
 thread_Type - An array of the request types (namely read, writer or
             upgrader)
 rec_Num - The identification number of the record a request tries to access
 num_threads - The number of requests
 table_r_time - The access time of table read
 record_r_time - The access time of record read
 table_w_time - The access time of table write
 record_w_time - The access time of record write
 interval - The interval time between the requests

 Return
 None
 -------------------------------------------------------------------------------------------------------------------------------------------*/
public static void RW_Server_3_Main(int[] thread_Type,
                              int[] rec_Num,
                              int num_threads,
                              int table_r_time,
                              int record_r_time,
                              int table_w_time,
                              int record_w_time,
                              int interval)

{
  int reader_id = 0, writer_id = 0,
  Database_3 db = new Database_3();
  ArrayList threadArrayList = new ArrayList(),

  for (int i = 0, i < num_threads, i++)
  {
    if (thread_Type[i] == 0)
    { // readers are added to list
      if (rec_Num[i] == 5) // this is a table level reader
```

```
        threadArrayList add(new Reader_3(reader_id++,
                                db,
                                0,
                                table_r_time)),
      else              // this is a record level reader
        threadArrayList add(new Reader_3(reader_id++,
                                db,
                                1,
                                record_r_time)),
    }
    else if (thread_Type[i] == 1)
    { // writers are added to list
      if (rec_Num[i] == 5) // this is a table level writer
        threadArrayList add(new Writer_3(writer_id++,
                                db,
                                0,
                                0,
                                table_w_time)),
      else              //this is a record level writer
        threadArrayList add(new Writer_3(writer_id++,
                                db,
                                1,
                                0,
                                record_w_time)),
    }
    else
    { // if it is a upgrader, add it to the list as a writer
      if (rec_Num[i] == 5) // this is a table level upgrader
        threadArrayList add(new Writer_3(writer_id++,
                                db,
                                0,
                                1,
                                table_r_time + table_w_time)),
      else              // this is a record level upgrader
        threadArrayList add(new Writer_3(writer_id++,
                                db,
                                1,
                                1,
                                record_r_time + record_w_time)),
    }
  }

  // converts the ArrayList containing the readers and writers to an Array
  Object[] threadArray = threadArrayList toArray(),
  System out println(""),
  System out println("Start running algorithm_3    "),
  Date startDate = new Date(),  // starting time

  for (int i = 0, i < threadArray length, i++)
  {
    if (threadArray[i] instanceof Reader_3)
    {
      ((Reader_3)threadArray[i]) start(),
      Break_3 duration(interval),
    }
    else
    {
      ((Writer_3)threadArray[i]) start(),
      Break_3 duration(interval),
    }
  }
  try
```

```
{
  for (int i = 0, i < threadArray length, i++)
  {
    if (threadArray[i] instanceof Reader_3)
      ((Reader_3)threadArray[i]) join(),
    else
      ((Writer_3)threadArray[i]) join(),
  }
}
catch (InterruptedException e)
{
  System out println("Interrupted"),
}

// ending time
Date endDate = new Date(),
// calculate and print to stdout the time spent in seconds
long timeDiff = endDate getTime() - startDate getTime(),
System out println(""),
System out println("\nTime spent for algorithm_3  " + (double) timeDiff / 1000 + " seconds "),

ArrayList TR_list = new ArrayList(),
ArrayList RR_list = new ArrayList(),
ArrayList TW_list = new ArrayList(),
ArrayList TU_list = new ArrayList(),
ArrayList RW_list = new ArrayList(),
ArrayList RU_list = new ArrayList(),

for (int i = 0, i < threadArray length, i++)
{
  if (threadArray[i] instanceof Reader_3)
  {
    if (((Reader_3)threadArray[i]) getType() == ReaderWriterType TR)
    {
      TR_list add(new Long(
          ((Reader_3)threadArray[i]) getLockWaitingTime())),
    }
    else
    {
      RR_list add(new Long(
          ((Reader_3)threadArray[i]) getLockWaitingTime())),
    }
  }
  else
  {
    if (((Writer_3)threadArray[i]) getType() == ReaderWriterType TW)
    {
      TW_list add(new Long(
          ((Writer_3)threadArray[i]) getLockWaitingTime())),
    }
    else if (((Writer_3)threadArray[i]) getType() == ReaderWriterType TU)
    {
      TU_list add(new Long(
          ((Writer_3)threadArray[i]) getLockWaitingTime())),
    }
    else if (((Writer_3)threadArray[i]) getType() == ReaderWriterType RW)
    {
      RW_list add(new Long(
          ((Writer_3)threadArray[i]) getLockWaitingTime())),
    }
    else
    {
```

```
      RU_lıst add(new Long(
          ((Wrıter_3)threadArray[ı]) getLockWaıtıngTıme())),
    }
  }
}

Object[] TR_arr = TR_lıst toArray(),
Object[] RR_arr = RR_lıst toArray(),
Object[] TW_arr = TW_lıst toArray(),
Object[] TU_arr = TU_lıst toArray(),
Object[] RW_arr = RW_lıst toArray(),
Object[] RU_arr = RU_lıst toArray(),

long avgRR = average(RR_arr),
long avgRU = average(RU_arr),
long avgRW = average(RW_arr),
long avgTR = average(TR_arr),
long avgTU = average(TU_arr),
long avgTW = average(TW_arr),

node nodeArray[] = {
    new node(avgTR, ReaderWrıterType TR),
    new node(avgRR, ReaderWrıterType RR),
    new node(avgTW, ReaderWrıterType TW),
    new node(avgTU, ReaderWrıterType TU),
    new node(avgRW, ReaderWrıterType RW),
    new node(avgRU, ReaderWrıterType RU)},

bubbleSort(nodeArray, 6),

System out prıntln("\nThe average tımes spent ın mıllıseconds to obtaın a "
                  + "lock ın algorıthm_3 \n"),
for (ınt ı = 0, ı < 6, ı++)
{
 swıtch (nodeArray[ı] type)
 {
   case ReaderWrıterType RR
     System out prınt("RR Average = " + avgRR  + " ["),
     dısplay(RR_arr),
     System out prıntln("]\n"),
     break,
   case ReaderWrıterType RU
     System out prınt("RU Average = " + avgRU  + " ["),
     dısplay(RU_arr),
     System out prıntln("]\n"),
     break,
   case ReaderWrıterType RW
     System out prınt("RW Average = " + avgRW  + " ["),
     dısplay(RW_arr),
     System out prıntln("]\n"),
     break,
   case ReaderWrıterType TR
     System out prınt("TR Average = " + avgTR  + " ["),
     dısplay(TR_arr),
     System out prıntln("]\n"),
     break,
   case ReaderWrıterType TU
     System out prınt("TU Average = " + avgTU + " ["),
     dısplay(TU_arr),
     System out prıntln("]\n"),
     break,
   case ReaderWrıterType TW
```

```
      System out print("TW Average = " + avgTW + " ["),
      display(TW_arr),
      System out println("]\n"),
      break,
     default
      System out println("Oop! Somerthing must be wrong "),
      break,

   }
  }
 }
}

/******************************************************************************************************************
 This class emulates a physical database  It contains methods that will be
 called by Reader_3 and Writer_3 classes
 ******************************************************************************************************************/
class Database_3
{
 /*--------------------------------------------------------------------------------------------------------------------
  Function    Database_3

  Description
  Constructor for the Database_3 class  It initializes readCount, and the
  semaphores rc, w and pw

  Parameters
  None

  Return
  None
  ------------------------------------------------------------------------------------------------------------------*/
 public Database_3()
 {
  readerCount = 0,

  rc = new Semaphore_3(1),
  w = new Semaphore_3(1),
  pw = new Semaphore_3(1),
 }

 /*--------------------------------------------------------------------------------------------------------------------
  Function
  startRead

  Description
  start a read process according to the fair reader and writer algorithm

  Parameters
  readNum - the ID number of a reader

  Return
  The number of readers that are currently reading
  ------------------------------------------------------------------------------------------------------------------*/
 public int startRead(int readerNum)
 {
  System out println("reader " + readerNum + " wants to read "),
  pw P(), // requests the outer semaphore
  rc P(), // request the reader count
  ++readerCount,
  if (readerCount == 1) // the first reader blocks the writer
    w.P(),
```

```
  rc V(), // release the reader count
  pw V(), // releases the outer semaphore
  return readerCount,
}

/*-------------------------------------------------------------------------------------------------------------------------------
 Function
 endRead

 Description
 end a read process according to the fair reader and writer algorithm

 Parameters
 readNum - the ID number of a reader

 Return
 The number of readers that are currently reading
 -------------------------------------------------------------------------------------------------------------------------------*/
public int endRead(int readerNum)
{
  rc P(),
  --readerCount;
  System out println("reader " + readerNum + " is done reading  Count = " +
                   readerCount),
  if (readerCount == 0) // the last reader unblocks the writer
    w V(),
  rc V(),

  return readerCount,
}

/*-------------------------------------------------------------------------------------------------------------------------------
 Function.
 startWrite

 Description.
 start a write process according to the fair reader and writer algorithm

 Parameters
 writerNum - the ID number of a writer

 Return
 None
 -------------------------------------------------------------------------------------------------------------------------------*/
public void startWrite(int writerNum)
{
  System out println("writer " + writerNum + " wants to write "),
  pw P(), // requests the outer semaphore
  w P(), // requests the database access
  pw V(), // releases the outer semaphore
}


/*-------------------------------------------------------------------------------------------------------------------------------
 Function
 endWrite

 Description
 end a write process according to the fair reader and writer algorithm

 Parameters.
 writerNum - the ID number of a writer
```

```
  Return
  None
  ------------------------------------------------------------------------------------------------------------------------*/
 public void endWrite(int writerNum)
 {
  System out println("writer " + writerNum + " is done writing "),
  w V(),
 }

 private int readerCount, // the number of active readers
 Semaphore_3 rc, // controls access to readerCount
 Semaphore_3 w, // controls access to the database
 Semaphore_3 pw, // controls access to outer semaphore
}

/*************************************************************************************************************************
 This class defines the reader and writer types and provides methods for
 retrieving the types

*************************************************************************************************************************/
class ReaderWriterType
{
 private int value,

 ReaderWriterType(int type)
 {
  value = type,
 }

 int getReaderWriterType()
 {
  return value,
 }

 final static int TR = 0x100,
 final static int RR = 0x101,
 final static int TW = 0x102,
 final static int TU = 0x103,
 final static int RW = 0x104,
 final static int RU = 0x105,
}

/*************************************************************************************************************************
 This class calls the method of Database_3 to start and end a read process

*************************************************************************************************************************/
class Reader_3 extends Thread
{
 /*------------------------------------------------------------------------------------------------------------------------
  Function
  Reader_3

  Description
  Constructor for Reader_3 class  It specifies which reader is reading from
  which database and how long the reading time is

  Parameters
  r - reader ID
  db - the database the reader is reading from
  n - indicates the reader is a table reader or a record reader
  r_time - reader access time
```

```
 Return
 None
 ----------------------------------------------------------------------------------------------------------------------------*/
public Reader_3(int r, Database_3 db, int n, int r_time)
{
  readerNum = r,
  server = db,
  readTime = r_time,

  if (n == 0)
    t_r = new ReaderWriterType(ReaderWriterType TR),
  else
    t_r = new ReaderWriterType(ReaderWriterType RR),
}

/*-----------------------------------------------------------------------------------------------------------------------------
 Function
 getType

 Description
 Get the type of the reader (TR or RR)

 Parameters
 None

 Return
 An integer that corresponds to the reader's type
 -----------------------------------------------------------------------------------------------------------------------------*/
public int getType()
{
  return t_r getReaderWriterType(),
}

/*-----------------------------------------------------------------------------------------------------------------------------
 Function
 getLockWaitingTime

 Description
 Get the lock waiting time for the reader

 Parameters
 None

 Return
 A long integer that corresponds to the lock waiting time
 -----------------------------------------------------------------------------------------------------------------------------*/
public long getLockWaitingTime()
{
  return lockWaitingTime,
}

/*-----------------------------------------------------------------------------------------------------------------------------
 Function
 run

 Description
 This function specifies how a reader thread runs

 Parameters
 None
```

```
  Return
  None
  ----------------------------------------------------------------------------------------------------------------------------*/
 public void run()
 {
  int c,

  Date start_req = new Date(),
  c = server startRead(readerNum),
  Date obtain_req = new Date(),

  System out println("reader " + readerNum + " is reading  Count = " + c),
  Break_3 duration(readTime), // read read_time milliseconds

  c = server endRead(readerNum),// end a read process

  lockWaitingTime = obtain_req getTime() - start_req getTime(),
 }

 private Database_3 server,      // the database that the reader tries to access
 private int readerNum,           // identification number of the reader
 private int readTime,            // access time spent by the reader
 private ReaderWriterType t_r,  // the type of the reader (TR or RR)
 private long lockWaitingTime,  // the lock waiting time of the reader
}

/*****************************************************************************************************************************
 This class calls the method of Database_3 to start and end a write process
*****************************************************************************************************************************/
class Writer_3 extends Thread
{
 /*----------------------------------------------------------------------------------------------------------------------------
  Function
  Writer_3

  Description
  Constructor for Writer_3 class  It specifies the type of the process(table
  writer, record writer, table upgrader, record upgrader)and its duration time

  Parameters
  w - writer ID
  db - the database the writer and upgrader are writing to
  n - indicates table writer or record writer
  u - indicates an upgrader or not
  w_time - writer access time

  Return
  None
  ----------------------------------------------------------------------------------------------------------------------------*/
 public Writer_3(int w, Database_3 db, int n, int u, int w_time)
 {
  writerNum = w,
  server = db,
  writeTime = w_time,

  if (n == 0)
  {
   if (u == 0)  // Table Writer
     t_w = new ReaderWriterType(ReaderWriterType TW),
   else        // Table Upgrader
     t_w = new ReaderWriterType(ReaderWriterType TU),
  }
```

```
  else
  {
   if (u == 0)  // Record Writer
     t_w = new ReaderWriterType(ReaderWriterType RW),
   else        // Record Upgrader
     t_w = new ReaderWriterType(ReaderWriterType RU),
  }
}

/*------------------------------------------------------------------------------------------------------------------------------------
 Function
 getType

 Description
 Get the type of the writer (TW, RW, TU or RU)

 Parameters
 None

 Return
 An integer that corresponds to the writer's type
 ------------------------------------------------------------------------------------------------------------------------------------*/
public int getType()
{
 return t_w getReaderWriterType(),
}

/*------------------------------------------------------------------------------------------------------------------------------------
 Function
 getLockWaitingTime

 Description
 Get the lock waiting time for the writer

 Parameters
 None

 Return
 A long integer that corresponds to the lock waiting time
 ------------------------------------------------------------------------------------------------------------------------------------*/
public long getLockWaitingTime()
{
 return lockWaitingTime,
}

/*------------------------------------------------------------------------------------------------------------------------------------
 Function
 run

 Description
 This function specifies how a writer thread runs

 Parameters
 None

 Return
 None
 ------------------------------------------------------------------------------------------------------------------------------------*/
public void run()
{
 Date start_req = new Date(),
 server startWrite(writerNum), // start a write process
```

```
   Date obtain_req = new Date(),

   System out println("writer " + writerNum + " is writing "),
   Break_3 duration(writeTime), // write write_time milliseconds

   server endWrite(writerNum), // end a write process

   lockWaitingTime = obtain_req getTime() - start_req getTime(),
  }

  private Database_3 server,     // the database that the writer tries to access
  private int writerNum,         // identification number of the writer
  private int writeTime,         // access time spent by the writer
  private ReaderWriterType t_w,  // the type of the writer (TW, TR, TU or RU)
  private long lockWaitingTime,  // the lock waiting time of the reader
}

/*****************************************************************************************************************
 Semaphore_3 class using Java syschronization

*****************************************************************************************************************/
final class Semaphore_3
{
 /*-----------------------------------------------------------------------------------------------------------------
  Function
  Semaphore_3

  Description
  default constructor for Semaphore_3

  Parameters
  None

  Return
  None
  ---------------------------------------------------------------------------------------------------------------*/
 public Semaphore_3()
 {
  value = 1,
 }

 /*----------------------------------------------------------------------------------------------------------------
  Function
  Semaphore_3

  Description
  constructor for Semaphore_3

  Parameter
  v - An integer value for the semaphore

  Return
  None
  ---------------------------------------------------------------------------------------------------------------*/
 public Semaphore_3(int v)
 {
  value = v,
 }

 /*-----------------------------------------------------------------------------------------------------------------
  Function
  P
```

```
  Description
  This function call the wait() to sleep when the value of Semaphore less than
  or equal 0  If the value of Semaphore is a positive number, decrements by 1

  Parameters.
  None

  Return
  None
  ----------------------------------------------------------------------------------------------------------------------------------*/
 public synchronized void P()
 {
  while (value <= 0)
  {
   try
   {
    wait(),
   }
   catch (InterruptedException e) {}
  }
  value--,
 }

 /*----------------------------------------------------------------------------------------------------------------------------------
  Function
  V

  Description
  This function increments the semaphore value by 1 and call the notify()
  function to wakeup a process that is waiting on the semaphore if it has any

  Parameters
  None

  Return.
  None
  ----------------------------------------------------------------------------------------------------------------------------------*/
 public synchronized void V()
 {
  ++value,
  notify(),
 }

 private int value,   // the value of the semaphore
}

/*******************************************************************************************************************************
 The class specifies the duration of access time

*******************************************************************************************************************************/
final class Break_3
{
 /*----------------------------------------------------------------------------------------------------------------------------------
  Function
  duration

  Description
  The function specifies the duration of access time in milliseconds

  Parameter
  milliseconds - how many milliseconds the access time is
```

```
   Return
   None
   -------------------------------------------------------------------------------------------------------------------------------------------*/
  public static void duration(int milliseconds)
  {
    try
    {
      Thread sleep(milliseconds),
    }
    catch (InterruptedException e) {}
  }
}
```

```
/*****************************************************************************************************************
 File
 RW_Server_4 java

 Description
 This is the implementation of the fair and efficient reader and writer
 algorithm  From the running result, We can observe this is more efficient
 than the fair reader and writer algorithm under most circumstances

 Author
 Mei Li

 Date
 April 24, 2004
 *****************************************************************************************************************/

import java io *,
import java util *,

/*****************************************************************************************************************
 The node class is used to declare objects that are used to sort the request
 types based on the average lock waiting time

*****************************************************************************************************************/
class node
{
  long avg,
  int type,

  node()
  {
   avg = -1,
   type = -1,
  }

  node(long a, int t)
  {
   avg = a,
   type = t,
  }
}

public class RW_Server_4
{
  /*----------------------------------------------------------------------------------------------------------------
   Function
   average

   Description
   It calculates the average of the long integers contained in the arr array

   Parameters.
   arr - An array of objects of Object type   These objects contains long
        integers corresponding to the lock waiting times for a request type

   Return
   The average  It returns -1 if the number of objects is 0
  ----------------------------------------------------------------------------------------------------------------*/
  static long average(Object[] arr)
  {
   long total = 0,
   int count = 0,
```

```
  for (ınt ı = 0, ı < arr length, ı++)
  {
    count++,
    total += ((Long)arr[ı]) longValue(),
  }

  ıf (count == 0)
    return -1,

  return total / count,
}

/*-----------------------------------------------------------------------------------------------------------------------------------
 Functıon
 dısplay

 Descrıptıon
 It dısplays on the screen the waıtıng tımes for a request type, separated
 wıth a space

 Parameters
 arr - An array of objects of Object type   These objects contaıns long
      ıntegers correspondıng to the lock waıtıng tımes for a request type

 Return
 None
 -----------------------------------------------------------------------------------------------------------------------------------*/
statıc voıd dısplay(Object[] array)
{
  for (ınt ı = 0, ı < array length, ı++)
  {
    ıf (ı == array length - 1)
    {
      System out prınt(((Long)array[ı]) longValue()),
    }
    else
    {
      System out prınt(((Long)array[ı]) longValue() + " "),
    }
  }
}

/*-----------------------------------------------------------------------------------------------------------------------------------
 Functıon
 bubbleSort

 Descrıptıon
 It sorts an array of nodes based on the average waıtıng tımes the nodes
 contaın   The sortıng algorıthm of bubble sort ıs used sınce we only have
 a small number (6) of request types to sort

 Parameters
 nodeArray - An array of nodes   These nodes contaıns the average waıtıng tıme
           for a request type and the request type
 sıze - The sıze of the nodeArray array

 Return
 None
 -----------------------------------------------------------------------------------------------------------------------------------*/
statıc voıd bubbleSort(node[] nodeArray, ınt sıze)
{
```

```
  node tmp = new node(),

  for (int ı = size - 1, ı > 0, ı--)
  {
   for (int j = 0, j < ı, j++)
   {
    if (nodeArray[j] avg > nodeArray[j + 1] avg)
    {
     tmp avg = nodeArray[j + 1] avg,
     tmp type = nodeArray[j + 1] type,
     nodeArray[j + 1] avg = nodeArray[j] avg,
     nodeArray[j + 1] type = nodeArray[j] type,
     nodeArray[j] avg = tmp avg,
     nodeArray[j] type = tmp type,
    }
   }
  }
}

/*------------------------------------------------------------------------------------------------------------------------------------
 Function
 RW_Server_4_Main

 Description
 This function generates threads that emulates the table read, record read,
 table write, record write, table upgrade and record upgrade requests   Then
 it starts the threads and wait for threads to terminate   After that, the
 throughput time for this algorithm and the turnaround times (lock waiting
 times) for the various requests are calculated and displayed

 Parameter
 thread_Type - An array of the request types (namely read, writer or
             upgrader)
 rec_Num - The identification number of the record a request tries to access
 num_threads - The number of requests
 table_r_time - The access time of table read
 record_r_time - The access time of record read
 table_w_time - The access time of table write
 record_w_time - The access time of record write
 interval - The interval time between the requests

 Return
 None
 ------------------------------------------------------------------------------------------------------------------------------------*/
public static void RW_Server_4_Main(int[] thread_Type,
                              int[] rec_Num,
                              int num_threads,
                              int table_r_time,
                              int record_r_time,
                              int table_w_time,
                              int record_w_time,
                              int interval)

{
  int reader_id = 0, writer_id = 0,
  Database_4 db = new Database_4(),
  ArrayList threadArrayList = new ArrayList(),

  for (int ı = 0, ı < num_threads, ı++)
  {
   if (thread_Type[ı] == 0)
   {   // readers are added to list
```

```
  if (rec_Num[i] == 5) // this is a table level reader
   threadArrayList add(new Reader_4(reader_id++,
                                   db,
                                   0,
                                   table_r_time)),
  else              // this is a record level reader
   threadArrayList add(new Reader_4(reader_id++,
                                   db,
                                   1,
                                   record_r_time)),
 }
 else if(thread_Type[i] == 1)
 { // writers are added to list
  if (rec_Num[i] == 5) // this is a table level writer
   threadArrayList add(new Writer_4(writer_id++,
                                   db,
                                   0,
                                   0,
                                   table_w_time)),
  else              //this is a record level writer
   threadArrayList add(new Writer_4(writer_id++,
                                   db,
                                   1,
                                   0,
                                   record_w_time)),
 }
 else
 { // if it is a upgrader, add it to the list as a writer
  if (rec_Num[i] == 5) // this is a table level upgrader
   threadArrayList add(new Writer_4(writer_id++,
                                  db,
                                  0,
                                  1,
                                  table_r_time + table_w_time)),
  else             // this is a record level upgrader
   threadArrayList add(new Writer_4(writer_id++,
                                   db,
                                   1,
                                   1,
                                   record_r_time + record_w_time)),
 }
}

// converts the ArrayList containing the readers and writers to an Array
Object[] threadArray = threadArrayList toArray(),
System out println(""),
System out println("Start running algorithm_4   "),
// start time
Date startDate = new Date(),

for (int i = 0, i < threadArray length, i++)
{
 if (threadArray[i] instanceof Reader_4)
 {
  ((Reader_4)threadArray[i]) start(),
  Break_4 duration(interval),
 }
 else
 {
  ((Writer_4)threadArray[i]) start(),
  Break_4 duration(interval),
 }
```

```
}
try
{
 for (int i = 0, i < threadArray length, i++)
 {
  if (threadArray[i] instanceof Reader_4)
   ((Reader_4)threadArray[i]) join(),
  else
   ((Writer_4)threadArray[i]) join(),
 }
}
catch (InterruptedException e)
{
 System out println("Interrupted"),
}

// ending time
Date endDate = new Date(),

// calculate and print to stdout the time spent in seconds
long timeDiff = endDate getTime() - startDate getTime(),
System out println(""),
System out println("\nTime spent for algorithm_4  "
          + (double) timeDiff / 1000 + " seconds "),

ArrayList TR_list = new ArrayList(),
ArrayList RR_list = new ArrayList(),
ArrayList TW_list = new ArrayList(),
ArrayList TU_list = new ArrayList();
ArrayList RW_list = new ArrayList(),
ArrayList RU_list = new ArrayList(),

for (int i = 0, i < threadArray length, i++)
{
 if (threadArray[i] instanceof Reader_4)
 {
  if (((Reader_4)threadArray[i]) getType() == ReaderWriterType TR)
  {
   TR_list.add(new Long(
     ((Reader_4)threadArray[i]) getLockWaitingTime())),
  }
  else
  {
   RR_list add(new Long(
     ((Reader_4)threadArray[i]) getLockWaitingTime())),
  }
 }
 else
 {
  if (((Writer_4)threadArray[i]) getType() == ReaderWriterType TW)
  {
   TW_list add(new Long(
     ((Writer_4)threadArray[i]) getLockWaitingTime())),
  }
  else if (((Writer_4)threadArray[i]) getType() == ReaderWriterType TU)
  {
   TU_list add(new Long(
     ((Writer_4)threadArray[i]) getLockWaitingTime())),
  }
  else if (((Writer_4)threadArray[i]) getType() == ReaderWriterType RW)
  {
   RW_list.add(new Long(
```

```
          ((Writer_4)threadArray[i]) getLockWaitingTime())),
        }
        else
        {
          RU_list add(new Long(
          ((Writer_4) threadArray[i]) getLockWaitingTime())),
        }
      }
    }

    Object[] TR_arr = TR_list.toArray(),
    Object[] RR_arr = RR_list toArray(),
    Object[] TW_arr = TW_list toArray(),
    Object[] TU_arr = TU_list toArray(),
    Object[] RW_arr = RW_list toArray(),
    Object[] RU_arr = RU_list toArray(),

    long avgRR = average(RR_arr),
    long avgRU = average(RU_arr),
    long avgRW = average(RW_arr),
    long avgTR = average(TR_arr),
    long avgTU = average(TU_arr),
    long avgTW = average(TW_arr),

    node nodeArray[] = {
       new node(avgTR, ReaderWriterType.TR),
       new node(avgRR, ReaderWriterType RR),
       new node(avgTW, ReaderWriterType TW),
       new node(avgTU, ReaderWriterType TU),
       new node(avgRW, ReaderWriterType RW),
       new node(avgRU, ReaderWriterType RU)},

    bubbleSort(nodeArray, 6),

    System out println("\nThe average times spent in milliseconds to obtain a "
                    + "lock in algorithm_4 \n"),
    for (int i = 0, i < 6, i++)
    {
      switch (nodeArray[i] type)
      {
        case ReaderWriterType RR
          System out print("RR Average = " + avgRR + " ["),
          display(RR_arr),
          System out println("]\n"),
          break,
        case ReaderWriterType RU
          System out print("RU Average = " + avgRU + " ["),
          display(RU_arr),
          System out println("]\n"),
          break,
        case ReaderWriterType RW
          System out print("RW Average = " + avgRW + " ["),
          display(RW_arr);
          System out println("]\n"),
          break,
        case ReaderWriterType TR
          System out print("TR Average = " + avgTR + " ["),
          display(TR_arr),
          System out println("]\n");
          break,
        case ReaderWriterType TU
          System out print("TU Average = " + avgTU + " ["),
```

```
        display(TU_arr),
        System out println("]\n"),
        break,
      case ReaderWriterType TW
        System out print("TW Average = " + avgTW + " ["),
        display(TW_arr),
        System out println("]\n"),
        break,
      default
        System out println("Oop! Somerthing must be wrong "),
        break,

     }
    }
   }
}

/*****************************************************************************************************************
 This class is emulates a physical database  It contains methods that will be
 called by Reader_4 and Writer_4 classes

*****************************************************************************************************************
/
class Database_4
{
 /*---------------------------------------------------------------------------------------------------------------
  Function    Database_4

  Description
  Constructor for the Database_4 class  It initializes readCount, and the
  semaphores rc, w and pw

  Parameters
  None

  Return
  None
  --------------------------------------------------------------------------------------------------------------*/
 public Database_4()
 {
  readerCount = 0,

  rc = new Semaphore_4(1);
  w = new Semaphore_4(1),
  pw = new Semaphore_4(1),
 }

 /*---------------------------------------------------------------------------------------------------------------
  Function
  startRead

  Description
  start a read process according to the reader previlege algorithm

  Parameters
  readNum - the ID number of a reader

  Return.
  The number of readers that are currently reading
  --------------------------------------------------------------------------------------------------------------*/
 public int startRead(int readerNum)
 {
```

```
  System out prıntln("reader " + readerNum + " wants to read "),
  pw P(), // requests the outer semaphore
  pw V(), // releases the outer semaphore
  rc P(),
  ++readerCount,
  // the first reader blocks wrıter and other process waıtıng on the outer semaphore
  ıf (readerCount == 1)
  {
    w P(),
    pw P(),
  }
  rc V(),
  return readerCount,
}

/*-------------------------------------------------------------------------------------------------------------------------------------
 Function
 endRead

 Descrıptıon
 end a read process accordıng to the reader prevılege algorıthm

 Parameters
 readNum - the ID number of a reader

 Return
 The number of readers that are currently readıng
 ------------------------------------------------------------------------------------------------------------------------------------*/
public ınt endRead(ınt readerNum)
{
  System out prıntln("reader " + readerNum + " ıs done readıng  Count = " +
                  readerCount);
  rc P(),
  --readerCount,

  // the last reader unblockd other processes waıtıng on the outer semaphore and database
  ıf (readerCount == 0)
  {
    pw V(),
    w V(),
  }
  rc V(),

  return readerCount,
}

/*---------------------------------------------------------------------------------------------------------------------------------
 Functıon
 startWrıte

 Descrıptıon
 start a wrıte process accordıng to the reader prevılege algorıthm

 Parameters·
 wrıterNum - the ID number of a wrıter

 Return
 None
 ------------------------------------------------------------------------------------------------------------------------------------*/
publıc void startWrıte(ınt wrıterNum)
{
  System out prıntln("wrıter " + wrıterNum + " wants to wrıte "),
```

```
  pw P(), // requests the outer semaphore
  pw V(), // releases the outer semaphore
  w P(), // requests the database access
 }

 /*-----------------------------------------------------------------------------------------------------------------------------------
  Function
  endWrite

  Description
  end a write process according to the reader previlege algorithm

  Parameters
  writerNum - the ID number of a writer

  Return
  None
  -----------------------------------------------------------------------------------------------------------------------------------*/
 public void endWrite(int writerNum)
 {
  System out println("writer " + writerNum + " is done writing "),
  w V(), // release the database access
 }

 // the number of active readers
 private int readerCount,

 Semaphore_4 rc, // controls access to readerCount
 Semaphore_4 w, // controls access to the database
 Semaphore_4 pw, // controls access to the outer semaphore
}

/*********************************************************************************************************************
 This class defines the reader and writer types and provides methods for
 retrieving the types.
 *********************************************************************************************************************/
class ReaderWriterType
{
 private int value,

 ReaderWriterType(int type)
 {
  value = type,
 }

 int getReaderWriterType()
 {
  return value,
 }

 final static int TR = 0x100,
 final static int RR = 0x101,
 final static int TW = 0x102,
 final static int TU = 0x103,
 final static int RW = 0x104,
 final static int RU = 0x105,
}

/*********************************************************************************************************************
 This class calls the method of Database_4 to start and end a read process
 *********************************************************************************************************************/
class Reader_4 extends Thread
```

```
{
 /*----------------------------------------------------------------------------------------------------------------------------------
 Function
 Reader_4

 Description
 Constructor for Reader_4 class  It specifies which reader is reading from
 which database and how long the reading time is

 Parameters
 r - reader ID
 db - the database the reader is reading from
 n - indicates the reader is a table reader or a record reader
 r_time - reader access time

 Return
 None
 ----------------------------------------------------------------------------------------------------------------------------------*/
 public Reader_4(int r, Database_4 db, int n, int read_time)
 {
  readerNum = r,
  server = db,
  readTime = read_time,

  if (n == 0)
    t_r = new ReaderWriterType(ReaderWriterType TR),
  else
    t_r = new ReaderWriterType(ReaderWriterType RR),
 }

 /*----------------------------------------------------------------------------------------------------------------------------------
 Function
 getType

 Description
 Get the type of the reader (TR or RR)

 Parameters
 None

 Return
 An integer that corresponds to the reader's type
 ----------------------------------------------------------------------------------------------------------------------------------*/
 public int getType()
 {
  return t_r getReaderWriterType(),
 }

 /*----------------------------------------------------------------------------------------------------------------------------------
 Function
 getLockWaitingTime

 Description
 Get the lock waiting time for the reader

 Parameters
 None

 Return
 A long integer that corresponds to the lock waiting time
 ----------------------------------------------------------------------------------------------------------------------------------*/
 public long getLockWaitingTime()
```

```
  {
   return lockWaitingTime,
  }

  /*-----------------------------------------------------------------------------------------------------------------------
   Function
   run

   Description
   This function specifies how a reader thread runs

   Parameters
   None

   Return
   None
   ---------------------------------------------------------------------------------------------------------------------*/
  public void run()
  {
   int c,

   Date start_req = new Date(),
   // start a read process
   c = server startRead(readerNum),
   Date obtain_req = new Date(),

   System out println("reader " + readerNum + " is reading  Count = " + c),
   Break_4 duration(readTime), // read read_time milliseconds

   c = server endRead(readerNum), // end a read process

   lockWaitingTime = obtain_req getTime() - start_req getTime(),
  }

  private Database_4 server,  // the database that the reader tries to access
  private int readerNum,      // identification number of the reader
  private int readTime,       // access time spent by the reader
  private ReaderWriterType t_r,  // the type of the reader (TR or RR)
  private long lockWaitingTime,  // the lock waiting time of the reader
}

/*************************************************************************************************************************
 This class calls the method of Database_4 to start and end a write process
*************************************************************************************************************************/
class Writer_4 extends Thread
{
  /*-----------------------------------------------------------------------------------------------------------------------
   Function
   Writer_1

   Description
   Constructor for Writer_1 class  It specifies the type of the process(table
   writer, record writer, table upgrader, record upgrader)and its duration time

   Parameters
   w - writer ID
   d - the database the writer and upgrader are writing to
   n - indicates table writer or record writer
   u - indicates an upgrader or not
   w_time - writer access time
   ---------------------------------------------------------------------------------------------------------------------*/
  public Writer_4(int w, Database_4 db, int n, int u, int write_time)
```

```
{
 writerNum = w,
 server = db,
 writeTime = write_time,

 if (n == 0)
 {
   if (u == 0)   // Table Writer
     t_w = new ReaderWriterType(ReaderWriterType TW),
   else         // Table Upgrader
     t_w = new ReaderWriterType(ReaderWriterType TU),
 }
 else
 {
   if (u == 0)  // Record Writer
     t_w = new ReaderWriterType(ReaderWriterType RW),
   else        // Record Upgrader
     t_w = new ReaderWriterType(ReaderWriterType RU),
 }
}

/*-----------------------------------------------------------------------------------------------------------------------------------
 Function
 getType

 Description
 Get the type of the writer (TW, RW, TU or RU)

 Parameters
 None

 Return
 An integer that corresponds to the writer's type
 -----------------------------------------------------------------------------------------------------------------------------------*/
public int getType()
{
 return t_w getReaderWriterType(),
}

/*-----------------------------------------------------------------------------------------------------------------------------------
 Function
 getLockWaitingTime

 Description.
 Get the lock waiting time for the writer

 Parameters.
 None

 Return
 A long integer that corresponds to the lock waiting time
 -----------------------------------------------------------------------------------------------------------------------------------*/
public long getLockWaitingTime()
{
 return lockWaitingTime,
}

/*-----------------------------------------------------------------------------------------------------------------------------------
 Function
 run

 Description
```

```
 This function specifies how a writer thread runs

 Parameters
 None

 Return
 None
 ------------------------------------------------------------------------------------------------------------------------------------*/
public void run()
{
 Date start_req = new Date();
 server startWrite(writerNum), // start a write process
 Date obtain_req = new Date();

 System out println("writer " + writerNum + " is writing "),
 Break_4 duration(writeTime), // write write_time milliseconds

 server endWrite(writerNum), // end a write process

 lockWaitingTime = obtain_req getTime() - start_req getTime(),
}

private Database_4 server,    // the database that the writer tries to access
private int writerNum,        // identification number of the writer
private int writeTime,        // access time spent by the writer
private ReaderWriterType t_w,  // the type of the writer (TW, TR, TU or RU)
private long lockWaitingTime,  // the lock waiting time of the reader
}

/******************************************************************************************************************************
 Semaphore_4 class using Java syschronization

******************************************************************************************************************************/
final class Semaphore_4
{
 /*-----------------------------------------------------------------------------------------------------------------------------------
  Function
  Semaphore_4

  Description·
  default constructor for Semaphore_4

  Parameters
  None

  Return
  None
  ------------------------------------------------------------------------------------------------------------------------------------*/
 public Semaphore_4()
 {
  value = 1,
 }

 /*-----------------------------------------------------------------------------------------------------------------------------------
  Function:
  Semaphore_1

  Description
  constructor for Semaphore_1

  Parameter
  v - An integer value for the semaphore
```

```
 Return
 None
 ------------------------------------------------------------------------------------------------*/
public Semaphore_4(int v)
{
 value = v,
}

/*------------------------------------------------------------------------------------------------
 Function
 P

 Description.
 This function call the wait() to sleep when the value of Semaphore less than
 or equal 0  If the value of Semaphore is a positive number, decrements by 1

 Parameters
 None

 Return
 None
 ------------------------------------------------------------------------------------------------*/
public synchronized void P()
{
 while (value <= 0)
 {
  try
  {
   wait(),
  }
  catch (InterruptedException e) {}
 }
 value--,
}

/*------------------------------------------------------------------------------------------------
 Function.
 V

 Description
 This function increments the semaphore value by 1 and call the notify()
 function to wakeup a process that is waiting on the semaphore if it has any

 Parameters
 None

 Return
 None
 ------------------------------------------------------------------------------------------------*/
public synchronized void V()
{
 ++value,
 notify(),
}

private int value,
}

/*****************************************************************************************************
 The class specifies the duration of access time
*****************************************************************************************************/
```

```
final class Break_4
{
 /*-------------------------------------------------------------------------------------------------------------------------------------
  Function
  duration

  Description
  The function specifies the duration of access time in milliseconds

  Parameter
  milliseconds - how many milliseconds the access time is

  Return
  None
  -------------------------------------------------------------------------------------------------------------------------------------*/
 public static void duration(int milliseconds)
 {
  try
  {
   Thread sleep(milliseconds),
  }
  catch (InterruptedException e) {}
 }
}
```

```
/*****************************************************************************************************************************
 File
 RW_Server_5 java

 Description
 This is a java program that implements "fair and efficient readers and writers
 with intent to read and write"  It is assumed that a 2-level resource (a table
 and the records in that table) is accessed by a number of readers and writers
 Among the readers and writers, some of them try to access the table as a whole
 while the rest try to access the individual records of the table  Readers and
 writers are implemented as threads  Whether a thread is a reader or writer,
 whether it tries to access the table or a record, and if it tries to access a
 record which record it is, is determined by the parameterized value that are
 passed into the method  Note that we consider a table upgrade as a table write
 and a record upgrade as a record write  The duration time equals to a reader
 time plus a writer time

 Author
 Mei Li

 Date
 April 24, 2004
 *****************************************************************************************************************************/

import java io *,
import java util.*,

/*****************************************************************************************************************************
 The node class is used to declare objects that are used to sort the request
 types based on the average lock waiting time
 *****************************************************************************************************************************/
class node
{
  long avg,
  int type,

  node()
  {
   avg = -1,
   type = -1,
  }

  node(long a, int t)
  {
   avg = a,
   type = t,
  }
}

public class I_RW_Server_5
{
  /*-----------------------------------------------------------------------------------------------------------------------------
   Function:
   average

   Description
    t calculates the average of the long integers contained in the arr array

   Parameters.
   arr - An array of objects of Object type  These objects contains long
        integers corresponding to the lock waiting times for a request type
```

```
 Return
 The average  It returns -1 if the number of objects is 0
 ---------------------------------------------------------------------------------------------------------------------------------------*/
static long average(Object[] arr)
{
 long total = 0,
 int count = 0,

 for (int i = 0, i < arr length, i++)
 {
  count++,
  total += ((Long)arr[i]) longValue(),
 }

 if (count == 0)
  return -1,

 return total / count,
}

/*----------------------------------------------------------------------------------------------------------------------------------------
 Function·
 display

 Description
 It displays on the screen the waiting times for a request type, separated
 with a space

 Parameters
 arr - An array of objects of Object type   These objects contains long
      integers corresponding to the lock waiting times for a request type

 Return·
 None
 ---------------------------------------------------------------------------------------------------------------------------------------*/
static void display(Object[] array)
{
 for (int i = 0, i < array length, i++)
 {
  if (i == array length - 1)
  {
   System out print(((Long)array[i]) longValue()),
  }
  else
  {
   System out print(((Long)array[i]) longValue() + " "),
  }
 }
}

/*----------------------------------------------------------------------------------------------------------------------------------------
 Function·
 bubbleSort

 Description·
 It sorts an array of nodes based on the average waiting times the nodes
 contain   The sorting algorithm of bubble sort is used since we only have
 a small number (6) of request types to sort

 Parameters
 nodeArray - An array of nodes   These nodes contains the average waiting time
          for a request type and the request type.
```

```
 size - The size of the nodeArray array

 Return
 None
------------------------------------------------------------------------------------------------------------------------------*/
static void bubbleSort(node[] nodeArray, int size)
{
  node tmp = new node(),

  for (int i = size - 1, i > 0, i--)
  {
    for (int j = 0, j < i, j++)
    {
      if (nodeArray[j] avg > nodeArray[j + 1] avg)
      {
        tmp avg = nodeArray[j + 1] avg,
        tmp type = nodeArray[j + 1] type,
        nodeArray[j + 1] avg = nodeArray[j] avg,
        nodeArray[j + 1] type = nodeArray[j] type,
        nodeArray[j] avg = tmp avg,
        nodeArray[j] type = tmp type,
      }
    }
  }
}

/*------------------------------------------------------------------------------------------------------------------------------
 Function
 I_RW_Server_5_Main

 Description
 This function generates threads that emulates the table read, record read,
 table write, record write, table upgrade and record upgrade requests   Then
 it starts the threads and wait for threads to terminate   After that, the
 throughput time for this algorithm and the turnaround times (lock waiting
 times) for the various requests are calculated and displayed

 Parameter
 thread_Type - An array of the request types (namely read, writer or
           upgrader)
 rec_Num - The identification number of the record a request tries to access
 num_threads - The number of requests
 table_r_time - The access time of table read
 record_r_time - The access time of record read
 table_w_time - The access time of table write
 record_w_time - The access time of record write
 interval - The interval time between the requests

 Return
 None
------------------------------------------------------------------------------------------------------------------------------*/
public static void I_RW_Server_5_Main(int[] thread_Type,
                                      int[] rec_Num,
                                      int num_threads,
                                      int table_r_time,
                                      int record_r_time,
                                      int table_w_time,
                                      int record_w_time,
                                      int interval)
{
  int reader_id = 0, writer_id = 0,
  Table_5 tbl = new Table_5(5),        // There are five records in a table
```

```
ArrayList threadArrayList = new ArrayList(),

for (int i = 0, i < num_threads, i++)
{
  if (thread_Type[i] == 0)
  { // readers
    if (rec_Num[i] == 5) // table level readers
      threadArrayList add(new Reader_5(reader_id++,
                                       -1,
                                       tbl,
                                       table_r_time)),
    else // record level readers
      threadArrayList add(new Reader_5(reader_id++,
                                       rec_Num[i],
                                       tbl,
                                       record_r_time)),
  }
  else if (thread_Type[i] == 1)
  { // writers
    if (rec_Num[i] == 5) // table level writers
      threadArrayList add(new Writer_5(writer_id++,
                                       -1,
                                       tbl,
                                       0,
                                       table_w_time)),
    else // record level writers
      threadArrayList add(new Writer_5(writer_id++,
                                       rec_Num[i],
                                       tbl,
                                       0,
                                       record_w_time)),
  }
  else
  { // if it is a upgrader, add it to the list of writers
    if (rec_Num[i] == 5) // table level upgrader
      threadArrayList add(new Writer_5(writer_id++,
                                       -1,
                                       tbl,
                                       1,
                                       table_r_time + table_w_time)),
    else // record level upgrader
      threadArrayList add(new Writer_5(writer_id++,
                                       rec_Num[i],
                                       tbl,
                                       1,
                                       record_r_time + record_w_time)),
  }
}

// converts the ArrayList containing the readers and writers to an Array
Object[] threadArray = threadArrayList toArray(),

System out println(""),
System out println("Start running algorithm_5    "),
// starting time
Date startDate = new Date(),

for (int i = 0, i < threadArray length, i++)
{
  if (threadArray[i] instanceof Reader_5)
  {
    ((Reader_5)threadArray[i]) start(),
```

```
      Break_5 duration(interval),
    }
    else
    {
      ((Writer_5)threadArray[i]) start(),
      Break_5 duration(interval),
    }
  }
  try
  {
    for (int i = 0, i < threadArray length, i++)
    {
      if (threadArray[i] instanceof Reader_5)
        ((Reader_5)threadArray[i]) join(),
      else
        ((Writer_5)threadArray[i]) join();
    }
  }
  catch (InterruptedException e)
  {
    System out println("Interrupted");
  }

  // ending time
  Date endDate = new Date(),

  // calculate and print to stdout the time spent in seconds
  long timeDiff = endDate getTime() - startDate getTime(),
  System out println(""),
  System out println("Time spent for algorithm_5  " +
                  (double) timeDiff / 1000 + " seconds "),

  ArrayList TR_list = new ArrayList(),
  ArrayList RR_list = new ArrayList(),
  ArrayList TW_list = new ArrayList(),
  ArrayList TU_list = new ArrayList(),
  ArrayList RW_list = new ArrayList(),
  ArrayList RU_list = new ArrayList(),

  for (int i = 0, i < threadArray.length, i++)
  {
    if (threadArray[i] instanceof Reader_5)
    {
      if (((Reader_5)threadArray[i]) getType() == ReaderWriterType TR)
      {
        TR_list add(new Long(
            ((Reader_5)threadArray[i]) getLockWaitingTime())),
      }
      else
      {
        RR_list add(new Long(
            ((Reader_5)threadArray[i]) getLockWaitingTime())),
      }
    }
    else
    {
      if (((Writer_5)threadArray[i]) getType() == ReaderWriterType TW)
      {
        TW_list add(new Long(
            ((Writer_5)threadArray[i]) getLockWaitingTime())),
      }
      else if (((Writer_5)threadArray[i]).getType() == ReaderWriterType TU)
```

```
    {
      TU_list add(new Long(
          ((Writer_5)threadArray[i]) getLockWaitingTime())),
    }
    else if (((Writer_5)threadArray[i]) getType() == ReaderWriterType RW)
    {
      RW_list add(new Long(
          ((Writer_5)threadArray[i]) getLockWaitingTime())),
    }
    else
    {
      RU_list add(new Long(
          ((Writer_5)threadArray[i]) getLockWaitingTime())),
    }
  }
}

Object[] TR_arr = TR_list toArray(),
Object[] RR_arr = RR_list toArray(),
Object[] TW_arr = TW_list toArray(),
Object[] TU_arr = TU_list toArray();
Object[] RW_arr = RW_list toArray(),
Object[] RU_arr = RU_list toArray(),

long avgRR = average(RR_arr),
long avgRU = average(RU_arr),
long avgRW = average(RW_arr),
long avgTR = average(TR_arr),
long avgTU = average(TU_arr),
long avgTW = average(TW_arr),

node nodeArray[] = {
   new node(avgTR, ReaderWriterType TR),
   new node(avgRR, ReaderWriterType RR),
   new node(avgTW, ReaderWriterType TW),
   new node(avgTU, ReaderWriterType TU),
   new node(avgRW, ReaderWriterType RW),
   new node(avgRU, ReaderWriterType RU)},

bubbleSort(nodeArray, 6),

System out println("\nThe average times spent in milliseconds to obtain a "
                + "lock in algorithm_5 \n"),
for (int i = 0, i < 6, i++)
{
  switch (nodeArray[i] type)
  {
    case ReaderWriterType RR
      System out print("RR Average = " + avgRR + " ["),
      display(RR_arr),
      System out println("]\n"),
      break,
    case ReaderWriterType RU
      System out print("RU Average = " + avgRU + " ["),
      display(RU_arr);
      System out println("]\n"),
      break,
    case ReaderWriterType RW
      System out print("RW Average = " + avgRW + " [");
      display(RW_arr),
      System out println("]\n"),
      break;
```

```
      case ReaderWnterType TR
        System out prnt("TR Average = " + avgTR + " ["),
        dısplay(TR_arr),
        System out prıntln("]\n"),
        break,
      case ReaderWnterType TU
        System out prnt("TU Average = " + avgTU + " ["),
        dısplay(TU_arr),
        System out prıntln("]\n"),
        break,
      case ReaderWnterType TW
        System out prnt("TW Average = " + avgTW + " ["),
        dısplay(TW_arr),
        System out prıntln("]\n"),
        break,
      default
        System out prıntln("Oop! Somerthıng must be wrong "),
        break,

    }
   }
  }
}

/****************************************************************************************************************
 Thıs class defines the types of the share semaphores rc[0], rc[1]  Thıs type ıs
 ınıtıalızed to be IR and the supported types are IR, IW, and R
 ***************************************************************************************************************/
class Type_5
{
 prıvate ınt value,

 /*------------------------------------------------------------------------------------------------------------------------
  Functıon
  Type_5

  Descrıptıon
  Default constructor for Type_5 class  The default type ıs IR

  Parameters
  None

  Return
  None
  ------------------------------------------------------------------------------------------------------------------------*/
 Type_5()
 {
  value = IR,
 }

 /*------------------------------------------------------------------------------------------------------------------------
  Functıon
  getType

  Descrıptıon
  get a type of a process

  Parameters
  None

  Return
  a value of a type
```

```
 -----------------------------------------------------------------------------------------------------------------------------*/
 int getType()
 {
   return value,
 }

 /*-----------------------------------------------------------------------------------------------------------------------------
  Function
  setType

  Description
  set a type of a process

  Parameters
  a value of a type

  Return
  None
 -----------------------------------------------------------------------------------------------------------------------------*/
 void setType(int v)
 {
   value = v,
 }

 final static int IR = 0x1000,
 final static int IW = 0x1001,
 final static int R = 0x1002,
}

/*****************************************************************************************************************************
 This class specifies the return value of some of the methods of Resource_5,
 Table_5 and Record_5
*****************************************************************************************************************************/
class ReturnValue_1
{
 int smp,   // The current sharing semaphore
 int count, // The count of current sharing semaphore
}

/*****************************************************************************************************************************
 This is the base class of Table_5 and Record_5 classes  It simulates the
 table that contains records  It implements intent read and intent write
 methods for record read and record write
*****************************************************************************************************************************/
class Resource_5
{
 protected Semaphore_5 w, // controls access to the resource
 protected Semaphore_5 pw, // controls access to the outer semaphore
 protected Semaphore_5[] rc, // share semaphore controlling access to count[]
 protected int[] count, // counters for share semaphores rc[0] and rc[1]
 protected Type_5[] type, // type for share semaphores rc[0] and rc[1]
 protected int prm, // which of share semaphores is primary, initially 0

 /*-----------------------------------------------------------------------------------------------------------------------------
  Function
  Resource_5

  Description
  Constructor for Resource_5 class  It initializes the various variable

  Parameters
  None
```

```
 Return
 None
 ------------------------------------------------------------------------------------------------------------------------------------*/
public Resource_5()
{
  w = new Semaphore_5(),
  pw = new Semaphore_5(),
  rc = new Semaphore_5[3],
  count = new int[3],
  type = new Type_5[3],
  prm = 0,

  for (int i = 0, i < 3, i++)
    rc[i] = new Semaphore_5(),
  for (int i = 0, i < 3, i++)
    type[i] = new Type_5(),
}

/*------------------------------------------------------------------------------------------------------------------------------------
 Function
 startRead

 Description
 Starts a read process according to fair and effcient reader and writer
 algorithm with intent to read and intent to write

 Parameters
 readerNum - the ID number of a reader

 Return
 a ReturnValue_1 object
 ------------------------------------------------------------------------------------------------------------------------------------*/
public ReturnValue_1 startRead(int readerNum)
{
  int smp, // indicates which of rc[0] or rc[1] currently to use
  ReturnValue_1 retVal = new ReturnValue_1(),

  pw P(), // requests the outer semaphore
  pw V(), // releases the outer semaphore

  if (type[prm] getType() != Type_5 IW)
    smp = prm;
  else
    smp = 1 - prm,
  retVal smp = smp,
  rc[smp] P(),
  count[smp]++,
  retVal count = count[smp],
  if (type[smp] getType() == Type_5 IR)
    type[smp] setType(Type_5 R),
    // the first reader blocks writer and other processes waiting on the outer
    // semaphore
  if (count[smp] == 1)
  {
    w P(),
    pw P(),
  }
  rc[smp] V(),

  return retVal,
}
```

```
/*-----------------------------------------------------------------------------------------------------------------------------------
 Function
 endRead

 Description
 Ends a read process according to fair and efficent reader and writer
 algorithm with intent to read and intent to write

 Parameters
 readerNum - the ID number of a reader
 val - a ReturnValue_1 object

 Return
 None
 -----------------------------------------------------------------------------------------------------------------------------------*/
public void endRead(int readerNum, ReturnValue_1 val)
{
  rc[val smp] P(),
  count[val smp]--,
  // the last reader unblocks other processes waiting on the outer semaphore
  // and table
  if (count[val smp] == 0)
  {
    type[val smp] setType(Type_5 IR),
    prm = 1 - prm,
    pw V(),
    w V(),
  }
  rc[val smp] V(),
}

/*-----------------------------------------------------------------------------------------------------------------------------------
 Function
 startWrite

 Description
 Starts a write process according to fair and effcient reader and writer
 algorithm with intent to read and intent to write

 Parameters
 writerNum - the ID number of a writer

 Return
 None
 -----------------------------------------------------------------------------------------------------------------------------------*/
public void startWrite(int writerNum)
{
  pw P(), // requests the outer semaphore
  pw V(), // releases the outer semaphore
  w P(), // requests the table access
}

/*-----------------------------------------------------------------------------------------------------------------------------------
 Function
 endWrite

 Description·
 Ends a write process according to fair and efficent reader and writer
 algorithm with intent to read and intent to write

 Parameters
```

```
 writerNum - the ID number of a writer

 Return
 None
 -------------------------------------------------------------------------------------------------------------------------------------*/
public void endWrite(int writerNum)
{
  w V(), // release the table access
}

/*--------------------------------------------------------------------------------------------------------------------------------------
 Function
 startIntentRead

 Description
 Starts a intent to read process according to fair and effcient reader and
 writer algorithm with intent to read and intent to write

 Parameters
 readerNum - the ID number of a reader

 Return
 a ReturnValue_1 object
 -------------------------------------------------------------------------------------------------------------------------------------*/
public ReturnValue_1 startIntentRead(int readerNum)
{
  int smp, // indicates which of rc[0] or rc[1] currently to use
  ReturnValue_1 retVal = new ReturnValue_1(),

  pw P(), // requests the outer semaphore
  pw V(), // releases the outer semaphore
  smp = prm,
  retVal smp = smp,

  rc[smp] P(),
  count[smp]++,
  retVal count = count[smp],
  if (count[smp] == 1)
  {
    w P(),
    pw P(),
  }
  rc[smp] V(),

  return retVal,
}

/*--------------------------------------------------------------------------------------------------------------------------------------
 Function
 endIntentRead

 Description
 Ends a Intent to read process according to fair and efficent reader and
 writer algorithm with intent to read and intent to write

 Parameters
 readerNum - the ID number of a reader
 val -  a returnValue_1 object

 Return
 None
 -------------------------------------------------------------------------------------------------------------------------------------*/
```

```
public void endIntentRead(int readerNum, ReturnValue_1 val)
{
  rc[val smp] P(),
  count[val smp]--,
  // the last reader unblocks other processes waiting on the outer semaphore
  // and table
  if (count[val smp] == 0)
  {
    type[val smp] setType(Type_5 IR),
    prm = 1 - prm,
    pw V(),
    w V(),
  }
  rc[val smp] V(),
}

/*------------------------------------------------------------------------------------------------------------------------------
 Function
 startIntentWrite

 Description
 Starts a intent to write process according to fair and effcient reader and
 writer algorithm with intent to read and intent to write

 Parameters
 writeNum - the ID number of a writer

 Return
 a ReturnValue_1 object
 ------------------------------------------------------------------------------------------------------------------------------*/
public ReturnValue_1 startIntentWrite(int writerNum)
{
  int smp, // indicates which of rc[0] or rc[1] currently to use
  ReturnValue_1 retVal = new ReturnValue_1(),

  pw P(), // requests the outer semaphore
  pw V(), // releases the outer semaphore

  if (type[prm] getType() == Type_5 IW || type[prm] getType() == Type_5 IR)
    smp = prm,
  else
    smp = 1 - prm,
  retVal smp = smp;

  rc[smp] P(),
  count[smp]++,
  retVal count = count[smp],
  if (type[smp] getType() == Type_5 IR)
    type[smp] setType(Type_5 IW),
  if (count[smp] == 1)
  {
    w P(),
    pw P(),
  }
  rc[smp] V(),

  return retVal,
}

/*------------------------------------------------------------------------------------------------------------------------------
 Function
 endIntentWrite
```

```
 Description
 Ends a Intent to write process according to fair and efficent reader and
 writer algorithm with intent to read and intent to write

 Parameters
 writeNum - the ID number of a writer
 val -  a returnValue_1 object

 Return
 None
 ---------------------------------------------------------------------------------------------------------------------------*/
public void endIntentWrite(int writerNum, ReturnValue_1 val)
{
  rc[val smp] P(),
  count[val smp]--,

  if (count[val smp] == 0)
  {
    type[val smp] setType(Type_5 IR),
    prm = 1 - prm,
    pw V(),
    w.V(),
  }
  rc[val smp] V(),
}

}

/****************************************************************************************************************************
 This class inherits the Resource_5 class  It specifies a table operation
 ****************************************************************************************************************************/
class Table_5 extends Resource_5
{
 private int num_of_records, // how many records in a table
 private Record_5[] records, // a array of record objects

 /*---------------------------------------------------------------------------------------------------------------------------
  Function
  Table_5

  Description
  Constructor for Table_5 class  It initializes the variours  variables

  Parameters
  num - how many records are in a table

  Return
  None
 ---------------------------------------------------------------------------------------------------------------------------*/
 public Table_5(int num)
 {
  super(),
  num_of_records = num,
  records = new Record_5[num_of_records],

  for (int i = 0; i < num_of_records, i++)
    records[i] = new Record_5(i),
 }

 /*---------------------------------------------------------------------------------------------------------------------------
  Function
```

```
 getNumOfRecords

 Description
 Gets the number of records in a table

 Parameters
 None

 Return
 The number of records
 -------------------------------------------------------------------------------------------------------------------------------*/
public int getNumOfRecords()
{
  return num_of_records,
}

/*--------------------------------------------------------------------------------------------------------------------------------
 Function
 getRecord

 Description
 Gets a record ID in a table

 Parameters
 index - the index of the records in the table

 Return       record ID
 -------------------------------------------------------------------------------------------------------------------------------*/
public Record_5 getRecord(int index)
{
  return records[index],
}

/*-------------------------------------------------------------------------------------------------------------------------------
  Function
  startRead

  Description
  Starts a read process according to fair and effcient reader and writer
  algorithm with intent to read and intent to write by overriding the same
  method in the base class

  Parameters
  readerNum - the ID number of a reader

  Return
  a ReturnValue_1 object
  ----------------------------------------------------------------------------------------------------------------------------*/
public ReturnValue_1 startRead(int readerNum)
{
  System out println("Reader " + readerNum + " wants to read from the TABLE"),
  return super startRead(readerNum),
}

/*-----------------------------------------------------------------------------------------------------------------------------
 Function
 endRead

 Description
 Ends a read process according to fair and efficent reader and writer
 algorithm with intent to read and intent to write
```

```
 Parameters
 readerNum - the ID number of a reader
 val - a ReturnValue_1 object

 Return.
 None
 -----------------------------------------------------------------------------------------------------------------------*/
public void endRead(int readerNum, ReturnValue_1 val)
{
  rc[val smp] P();
  count[val smp]--,
  System out println("Reader " + readerNum +
                  " is done reading from the TABLE "
                  + " Count[" + val smp + "] = " + count[val smp]),
  // the last reader unblocks other processes waiting on the outer semaphore
  // and table
  if (count[val smp] == 0)
  {
    type[val smp] setType(Type_5 IR);
    prm = 1 - prm,
    pw V(),
    w V();
  }
  rc[val smp] V(),
}

// Overrides the same method in the base class
public void startWrite(int writerNum)
{
  System out println("Writer " + writerNum + " wants to write to the TABLE"),
  super startWrite(writerNum),
}

// Overrides the same method in the base class
public void endWrite(int writerNum)
{
  System out println("Writer " + writerNum + " is done writing to the TABLE"),
  super endWrite(writerNum),
}

// Overrides the same method in the base class
public ReturnValue_1 startIntentRead(int readerNum)
{
  System out println("Reader " + readerNum +
                  " wants to do intent read from the "
                  + "TABLE "),
  return super.startIntentRead(readerNum),
}

/*-----------------------------------------------------------------------------------------------------------------------
 Function
 endIntentRead

 Description·
 Ends a Intent to read process according to fair and efficent reader and
 writer algorithm with intent to read and intent to write

 Parameters
 readerNum - the ID number of a reader
 val - a returnValue_1 object

 Return.
```

```
 None
 ------------------------------------------------------------------------------------------------------------------------------*/
public void endIntentRead(int readerNum, ReturnValue_1 val)
{
  rc[val smp] P(),
  count[val smp]--,
  System.out println("Reader " + readerNum + " is done intent read from the "
                 + "TABLE. Count[" + val smp + "] = " + count[val smp]),
  // the last reader unblocks other processes waiting on the outer semaphore
  // and table
  if (count[val smp] == 0)
  {
    type[val smp] setType(Type_5 IR),
    prm = 1 - prm;
    pw V(),
    w.V(),
  }
  rc[val smp] V(),
}

// Overrides the same method in the base class
public ReturnValue_1 startIntentWrite(int writerNum)
{
  System out println("Writer " + writerNum +
                 " wants to do intent write to the"
                 + " TABLE");
  return super startIntentWrite(writerNum),
}

/*--------------------------------------------------------------------------------------------------------------------------------
 Function
 endIntentWrite

 Description
 Ends a Intent to write process according to fair and efficent reader and
 writer algorithm with intent to read and intent to write

 Parameters
 writeNum - the ID number of a writer
 val -  a returnValue_1 object

 Return
 None
 ------------------------------------------------------------------------------------------------------------------------------*/
public void endIntentWrite(int writerNum, ReturnValue_1 val)
{
  rc[val smp] P(),
  count[val smp]--,
  System out println("Writer " + writerNum + " is done intent write to the"
                 + " TABLE  Count[" + val smp + "] = " + count[val smp]),
  if (count[val smp] == 0)
  {
    type[val smp] setType(Type_5 IR),
    prm = 1 - prm,
    pw V(),
    w V(),
  }
  rc[val smp] V(),
 }
}
```

```
/*************************************************************************************************************************
***
 This is the record class that implements efficient read-write to simulate
 read and write on the individual records in a table   Along with others
 this class also has an instance variable that keeps track of the record's
 index in the table
 ************************************************************************************************************************/
class Record_5 extends Resource_5
{
  private int index;

  /*-----------------------------------------------------------------------------------------------------------------------------
   Function.
   Record_5

   Description.
   Constructor for Record_5 class  It initializes the various variables

   Parameter
   i - record id in a table
   -----------------------------------------------------------------------------------------------------------------------------*/
  public Record_5(int i)
  {
    super(),
    index = i,
  }

  // Overrides the same method in the base class
  public ReturnValue_1 startRead(int readerNum)
  {
    System out println("Reader " + readerNum + " wants to read from RECORD "
                  + index),
    return super startRead(readerNum),
  }

  /*-----------------------------------------------------------------------------------------------------------------------------
   Function
   endRead

   Description
   Ends a read process according to fair and efficent reader and writer
   algorithm with intent to read and intent to write  This is the same as
   Table_5 endRead()

   Parameters
   readerNum - the ID number of a reader
   val - a ReturnValue_1 object

   Return
   None
   -----------------------------------------------------------------------------------------------------------------------------*/
  public void endRead(int readerNum, ReturnValue_1 val)
  {
    rc[val smp] P(),
    count[val smp]--,
    System out println("Reader " + readerNum + " is done reading from RECORD "
                  + index + "  Count_" + index + "[" + val smp +
                  "] = " + count[val smp]),
    // the last reader unblocks other processes waiting on the outer semaphore
    // and table
    if (count[val smp] == 0)
    {
```

```
    type[val smp] setType(Type_5 IR),
    prm = 1 - prm,
    pw V(),
    w V(),
  }
  rc[val smp] V(),
 }

 // Overrides the same method in the base class
 public void startWrite(int writerNum)
 {
  System out println("Writer " + writerNum + " wants to write to RECORD "
            + index),
  super.startWrite(writerNum),
 }

 // Overrides the same method in the base class
 public void endWrite(int writerNum)
 {
  System out println("Writer " + writerNum + " is done writing to RECORD "
            + index),
  super endWrite(writerNum),
 }
}

/**********************************************************************************************************************
 This class defines the reader and writer types and provides methods for
 retrieving the types
 **********************************************************************************************************************/
class ReaderWriterType
{
 private int value,

 ReaderWriterType(int type)
 {
  value = type,
 }

 int getReaderWriterType()
 {
  return value,
 }

 final static int TR = 0x100,
 final static int RR = 0x101,
 final static int TW = 0x102,
 final static int TU = 0x103,
 final static int RW = 0x104,
 final static int RU = 0x105,
}

/**********************************************************************************************************************
 This class calls the various methods to start and end a read process
**********************************************************************************************************************/
class Reader_5 extends Thread
{
 private int readerNum,   // the identification number of the reader
 private int recordNum,   // the index number of the record
 private Table_5 tbl,     // the table that the reader tries to access
 private int readTime,    // access time spent by the reader
 private ReaderWriterType t_r,  // the type of the reader (TR or RR)
 private long lockWaitingTime,  // the lock waiting time of the reader
```

```
/*----------------------------------------------------------------------------------------------------------------------------------------
 Function
 Reader_5

 Description
 Constructor function for the reader class  If the value of the recNum
 argument is -1, the reader wants to read the table as a whole  Otherwise,
 the reader wants to read a record whose index is contained in recNum

 Parameters
 id - the identification number of the reader thread
 recNum - the identification number of the record the reader will read  -1 if
        the reader reads the table only
 table - the table the reader will work on
 r_time - time the reader will spend for the reading operation
 ---------------------------------------------------------------------------------------------------------------------------------------*/
public Reader_5(int id, int recNum, Table_5 table, int r_time)
{
 readerNum = id,
 recordNum = recNum,
 tbl = table,
 readTime = r_time,

 if (recordNum == -1)
 {  // table reader
   t_r = new ReaderWriterType(ReaderWriterType TR),
 }
 else
 {  // record reader
   t_r = new ReaderWriterType(ReaderWriterType RR),
 }
}

/*----------------------------------------------------------------------------------------------------------------------------------------
 Function
 getType

 Description
 Get the type of the reader (TR or RR)

 Parameters
 None

 Return
 An integer that corresponds to the reader's type
 ---------------------------------------------------------------------------------------------------------------------------------------*/
public int getType()
{
 return t_r getReaderWriterType(),
}

/*----------------------------------------------------------------------------------------------------------------------------------------
 Function
 getLockWaitingTime

 Description
 Get the lock waiting time for the reader

 Parameters
 None
```

```
 Return
 A long integer that corresponds to the lock waiting time
 -------------------------------------------------------------------------------------------------------------------------------*/
public long getLockWaitingTime()
{
  return lockWaitingTime,
}

/*-------------------------------------------------------------------------------------------------------------------------------
 Function
 run

 Description
 This function specifies how a reader thread runs

 Parameters
 None

 Return
 None
 -------------------------------------------------------------------------------------------------------------------------------*/
public void run()
{
  if (recordNum == -1)
  { // Reader will read the table as a whole
    Date start_req = new Date(),
    ReturnValue_1 val = tbl startRead(readerNum),
    Date obtain_req = new Date(),

    System out println("Reader " + readerNum + " is reading from the table  "
                  + "Count[" + val smp + "] = " + val count),
    // read readTime milliseconds
    Break_5 duration(readTime),

    tbl endRead(readerNum, val),

    lockWaitingTime = obtain_req getTime() - start_req getTime(),
  }
  else
  { // Reader will read an individual record
    Date start_req = new Date(),
    // do intent read on table first
    ReturnValue_1 val_1 = tbl startIntentRead(readerNum),
    System out println("Reader " + readerNum + " is doing intent read from "
                  + "the table  Count[" + val_1 smp + "] = " +
                  val_1 count),
    // then read the record
    ReturnValue_1 val_2 = tbl getRecord(recordNum) startRead(readerNum),
    Date obtain_req = new Date(),

    System out println("Reader " + readerNum + " is reading from record "
                  + recordNum + "  Count_" + recordNum + "[" + val_2 smp
                  + "] = " + val_2 count),
    // read readTime milliseconds
    Break_5 duration(readTime),

    tbl getRecord(recordNum) endRead(readerNum, val_2),
    tbl endIntentRead(readerNum, val_1),

    lockWaitingTime = obtain_req getTime() - start_req getTime(),
  }
}
```

```
}

/*****************************************************************************************************************
 This class calls the various methods to start and end a write process
 ****************************************************************************************************************/
class Writer_5 extends Thread
{
  private int writerNum,  // the identification number of the writer
  private int recordNum,  // the index number of the record
  private Table_5 tbl,    // the table that the reader tries to access
  private int writeTime,  // access time spent by the writer
  private ReaderWriterType t_w,  // the type of the reader (TW, TU, RW or RU)
  private long lockWaitingTime,  // the lock waiting time of the reader

  /*-------------------------------------------------------------------------------------------------------------------------------
   Function
   Writer_5

   Description
   Constructor function for the Writr_5 class   If the value of the recNum
   argument is -1, the writer wants to write the table as a whole  Otherwise,
   the writer wants to write a record whose index is contained in recNum

   Parameters
   id - id of the writer
   recNum - index number of the record
   table - the table the write tries to access
   u - indicates whether the writer is an upgrader or not
   w_time - access time of the writer

   Return
   None
   *-------------------------------------------------------------------------------------------------------------------------------*/
  public Writer_5(int id, int recNum, Table_5 table, int u, int w_time)
  {
    writerNum = id,
    recordNum = recNum,
    tbl = table;
    writeTime = w_time,

    if (recordNum == -1)
    {   // table level write
      if (u == 0)    // table writer
        t_w = new ReaderWriterType(ReaderWriterType TW),
      else          // table upgrader
        t_w = new ReaderWriterType(ReaderWriterType TU),
    }
    else
    {   // record level write
      if (u == 0)    // record writer
        t_w = new ReaderWriterType(ReaderWriterType RW),
      else          // record upgrader
        t_w = new ReaderWriterType(ReaderWriterType RU),
    }
  }

  /*-------------------------------------------------------------------------------------------------------------------------------
   Function
   getType

   Description
   Get the type of the writer (TW, RW, TU or RU)
```

```
 Parameters
 None

 Return
 An ınteger that corresponds to the wrıter's type
 --------------------------------------------------------------------------------------------------------------------------------*/
public ınt getType()
{
  return t_w getReaderWrıterType(),
}

/*--------------------------------------------------------------------------------------------------------------------------------
 Functıon
 getLockWaıtıngTıme

 Descrıptıon
 Get the lock waıtıng tıme for the wrıter

 Parameters
 None

 Return
 A long ınteger that corresponds to the lock waıtıng tıme
 --------------------------------------------------------------------------------------------------------------------------------*/
public long getLockWaıtıngTıme()
{
  return lockWaıtıngTıme,
}

/*--------------------------------------------------------------------------------------------------------------------------------
 Functıon
 run

 Descrıptıon
 Thıs functıon specıfies how a wrıter thread runs

 Parameters
 None

 Return
 None
 --------------------------------------------------------------------------------------------------------------------------------*/
public voıd run()
{
  // start a wrıte process
  ıf (recordNum == -1)
  { // wrıter wıll wrıte to the table as a whole
    Date start_req = new Date(),
    tbl startWrıte(wrıterNum),
    Date obtaın_req = new Date(),

    System out prıntln("Wrıter " + wrıterNum + " ıs wrıtıng to the table"),
    Break_5 duratıon(wrıteTıme), // wrıte wrıteTıme mıllıseconds

    tbl endWrıte(wrıterNum),

    lockWaıtıngTıme = obtaın_req getTıme() - start_req getTıme(),
  }
  else
  { // wrıter wıll wrıte to an ındıvıdual record
    Date start_req = new Date(),
```

```
    // do intent to write first
    ReturnValue_1 val = tbl startIntentWrite(writerNum),
    System out println("Writer " + writerNum +
                  " is doing intent write to the "
                  + "table  Count[" + val smp + "] = " + val count),
    tbl getRecord(recordNum) startWrite(writerNum), // write to the record
    Date obtain_req = new Date(),

    System out println("Writer " + writerNum + " is writing to record "
                  + recordNum),
    Break_5 duration(writeTime), // write writeTime milliseconds

    tbl getRecord(recordNum) endWrite(writerNum), // end write to record
    tbl endIntentWrite(writerNum, val), // end intent to write

    lockWaitingTime = obtain_req getTime() - start_req getTime(),
   }
  }
}

/**************************************************************************************************************************
 This class implements semaphore using Java syschronization

**************************************************************************************************************************/
final class Semaphore_5
{
  private int value,  // the value of the semaphore

  /*-------------------------------------------------------------------------------------------------------------------------------
   Function:
   Semaphore_5

   Description
   default constructor for Semaphore_5

   Parameters
   None

   Return
   None
   -------------------------------------------------------------------------------------------------------------------------------
-*/
  public Semaphore_5()
  {
   value = 1,
  }

  /*-------------------------------------------------------------------------------------------------------------------------------
   Function
   Semaphore_5

   Description
   constructor for Semaphore_5

   Parameter:
   v - An integer value for the semaphore

   Return
   None
   ------------------------------------------------------------------------------------------------------------------------------*/
  public Semaphore_5(int v)
  {
```

```
    value = v,
  }

  /*---------------------------------------------------------------------------------------------------------------------------
   Function
   P

   Description
   This function call the wait() to sleep when the value of Semaphore less than
   or equal 0  If the value of Semaphore is a positive number, decrements by 1

   Parameters.
   None

   Return
   None
   ---------------------------------------------------------------------------------------------------------------------------*/
  public synchronized void P()
  {
    while (value <= 0)
    {
      try
      {
        wait(),
      }
      catch (InterruptedException e) {}
    }
    value--,
  }

  /*---------------------------------------------------------------------------------------------------------------------------
   Function
   V

   Description
   This function increments the semaphore value by 1 and call the notify()
   function to wakeup a process that is waiting on the semaphore if it has any

   Parameters
   None

   Return
   None
   ---------------------------------------------------------------------------------------------------------------------------*/

  public synchronized void V()
  {
    ++value,
    notify(),
  }
}

/**************************************************************************************************************************
 The class specified the duration of access time
 *************************************************************************************************************************/
final class Break_5
{
  /*---------------------------------------------------------------------------------------------------------------------------
   Function·
   duration

   Description
```

```
  The functıon specıfıes the duratıon of access tıme ın mıllıseconds

  Parameter
  mıllıseconds - how many mıllıseconds the access tıme ıs

  Return
  None
  ------------------------------------------------------------------------------------------------------------------------------------------*/
 publıc statıc voıd duratıon(ınt mıllıseconds)
 {
  try
  {
   Thread sleep(mıllıseconds),
  }
  catch (InterruptedExceptıon e) {}
 }
}
```

```
/*************************************************************************************************************************
 File
 RW_Server_6 java

 Description
 This is a java program that implements "fair and efficient readers and writers
 with intent to read and write lock and upgrade lock"  It is assumed that a
 2-level resource (a table and the records in that table) is accessed by a
 number of readers, writers and upgraders  Among the readers, writers and
 upgraders, some of them try to access the table as a whole while the rest try
 to access the individual records of the table  Readers and writers are
 implemented as threads  Whether a thread is a reader or a writer or a
 upgrader, whether it tries to access the table or a record, and if it tries
 to access a record which record it is, is determined by the parameterized
 value that are passed into the method  Note that we consider a table upgrade
 as a table write and a record upgrade as a record write  The duration time
 equals to a reader time plus a writer time

 Author
 Mei Li

 Date
 April 24, 2004
 *************************************************************************************************************************/

import java io *,
import java util *,

/*************************************************************************************************************************
 The node class is used to declare objects that are used to sort the request
 types based on the average lock waiting time
 *************************************************************************************************************************/
class node
{
  long avg,
  int type,

  node()
  {
   avg = -1,
   type = -1,
  }

  node(long a, int t)
  {
   avg = a,
   type = t,
  }
}

public class I_RW_U_Server_6
{
  /*-------------------------------------------------------------------------------------------------------------------------
   Function
   average

   Description
   It calculates the average of the long integers contained in the arr array

   Parameters
   arr - An array of objects of Object type  These objects contains long
        integers corresponding to the lock waiting times for a request type
```

```
 Return
 The average  It returns -1 if the number of objects is 0
-------------------------------------------------------------------------------------------------------------------------------*/
static long average(Object[] arr)
{
  long total = 0,
  int count = 0,

  for (int i = 0, i < arr length; i++)
  {
    count++,
    total += ( (Long) arr[i]) longValue(),
  }

  if (count == 0)
    return -1,

  return total / count,
}

/*-------------------------------------------------------------------------------------------------------------------------------
 Function
 display

 Description.
 It displays on the screen the waiting times for a request type, separated
 with a space

 Parameters
 arr - An array of objects of Object type   These objects contains long
      integers corresponding to the lock waiting times for a request type

 Return
 None
-------------------------------------------------------------------------------------------------------------------------------*/
static void display(Object[] array)
{
  for (int i = 0, i < array length, i++)
  {
    if (i == array length - 1)
    {
      System out print(((Long)array[i]) longValue());
    }
    else {
      System out print(((Long)array[i]) longValue() + " "),
    }
  }
}

/*-------------------------------------------------------------------------------------------------------------------------------
 Function
 bubbleSort

 Description
 It sorts an array of nodes based on the average waiting times the nodes
 contain   The sorting algorithm of bubble sort is used since we only have
 a small number (6) of request types to sort.

 Parameters·
 nodeArray - An array of nodes   These nodes contains the average waiting time
         for a request type and the request type
```

```
 size - The size of the nodeArray array

 Return
 None
-----------------------------------------------------------------------------------------------------------------------------------*/
static void bubbleSort(node[] nodeArray, int size)
{
  node tmp = new node();

  for (int i = size - 1, i > 0, i--)
  {
   for (int j = 0, j < i, j++)
   {
    if (nodeArray[j] avg > nodeArray[j + 1] avg)
    {
     tmp avg = nodeArray[j + 1] avg,
     tmp type = nodeArray[j + 1] type,
     nodeArray[j + 1] avg = nodeArray[j] avg,
     nodeArray[j + 1] type = nodeArray[j] type,
     nodeArray[j] avg = tmp avg,
     nodeArray[j] type = tmp type,
    }
   }
  }
}

/*-----------------------------------------------------------------------------------------------------------------------------------
 Function·
 I_RW_Server_6_Main

 Description
 This function generates threads that emulates the table read, record read,
 table write, record write, table upgrade and record upgrade requests   Then
 it starts the threads and wait for threads to terminate   After that, the
 throughput time for this algorithm and the turnaround times (lock waiting
 times) for the various requests are calculated and displayed

 Parameter
 thread_Type - An array of the request types (namely read, writer or
             upgrader)
 rec_Num - The identification number of the record a request tries to access
 num_threads - The number of requests
 table_r_time - The access time of table read
 record_r_time - The access time of record read
 table_w_time - The access time of table write
 record_w_time - The access time of record write
 interval - The interval time between the requests

 Return·
 None
-----------------------------------------------------------------------------------------------------------------------------------*/
public static void I_RW_U_Server_6_Main(int[] thread_Type,
                                int[] rec_Num,
                                int num_threads,
                                int table_r_time,
                                int record_r_time,
                                int table_w_time,
                                int record_w_time,
                                int interval)
{
  int reader_id = 0, writer_id = 0, upgrader_id = 0,
  Table_6 tbl = new Table_6(5), // there're initially 5 records in the table
```

```
ArrayLıst threadArrayLıst = new ArrayLıst(),

for (ınt ı = 0, ı < num_threads, ı++)
{
  ıf (thread_Type[ı] == 0)
  { // readers
    ıf (rec_Num[ı] == 5) // table level readers
      threadArrayLıst add(new Reader_6(reader_ıd++,
                                -1,
                                tbl,
                                table_r_tıme)),
    else // record level readers
      threadArrayLıst add(new Reader_6(reader_ıd++,
                                rec_Num[ı],
                                tbl,
                                record_r_tıme)),
  }
  else ıf (thread_Type[ı] == 1)
  { // wrıters
    ıf (rec_Num[ı] == 5) // table level wrıters
      threadArrayLıst add(new Wrıter_6(wrıter_ıd++,
                                -1,
                                tbl,
                                table_w_tıme)),
    else // record level wrıters
      threadArrayLıst add(new Wrıter_6(wrıter_ıd++,
                                rec_Num[ı],
                                tbl,
                                record_w_tıme)),
  }
  else
  { // upgraders
    ıf (rec_Num[ı] == 5) // table level upgrader
      threadArrayLıst add(new Upgrader(upgrader_ıd++,
                                -1,
                                tbl,
                                table_r_tıme,
                                table_w_tıme)),
    else // record level upgrader
      threadArrayLıst add(new Upgrader(upgrader_ıd++,
                                rec_Num[ı],
                                tbl,
                                record_r_tıme,
                                record_w_tıme)),
  }
}

// converts the ArrayLıst to an Array
Object[] threadArray = threadArrayLıst toArray(),
System out prıntln(""),
System out prıntln("Start runnıng algorıthm_6    "),

// startıng tıme
Date startDate = new Date(),

for (ınt ı = 0, ı < threadArray length, ı++)
{
  ıf (threadArray[ı] ınstanceof Reader_6)
  {
    ((Reader_6)threadArray[ı]) start(),
    Break_6 duratıon(ınterval),
  }
```

```
  else ıf (threadArray[ı] ınstanceof Wrıter_6)
  {
   ((Wrıter_6)threadArray[ı]) start(),
   Break_6 duratıon(ınterval),
  }
  else
  {
   ((Upgrader)threadArray[ı]) start(),
   Break_6 duratıon(ınterval),
  }
 }
 try
 {
  for (ınt ı = 0, ı < threadArray length, ı++)
  {
   ıf (threadArray[ı] ınstanceof Reader_6)
    ((Reader_6)threadArray[ı]) joın(),
   else ıf (threadArray[ı] ınstanceof Wrıter_6)
    ((Wrıter_6)threadArray[ı]) joın(),
   else
    ((Upgrader)threadArray[ı]) joın(),
  }
 }
 catch (InterruptedExceptıon e)
 {
  System out prıntln("Interrupted"),
 }

 // endıng tıme
 Date endDate = new Date(),

 // calculate and prınt to stdout the tıme spent ın seconds
 long tımeDıff = endDate getTıme() - startDate getTıme(),
 System out prıntln(""),
 System out prıntln("Tıme spent for algorıthm_6  " +
          (double) tımeDıff / 1000 + " seconds "),

 ArrayLıst TR_lıst = new ArrayLıst(),
 ArrayLıst RR_lıst = new ArrayLıst(),
 ArrayLıst TW_lıst = new ArrayLıst(),
 ArrayLıst TU_lıst = new ArrayLıst(),
 ArrayLıst RW_lıst = new ArrayLıst(),
 ArrayLıst RU_lıst = new ArrayLıst();

 for (ınt ı = 0, ı < threadArray length, ı++)
 {
  ıf (threadArray[ı] ınstanceof Reader_6)
  {
   ıf (((Reader_6)threadArray[ı]) getType() == ReaderWrıterType TR)
   {
    TR_lıst add(new Long(
      ((Reader_6)threadArray[ı]) getLockWaıtıngTıme())),
   }
   else
   {
    RR_lıst add(new Long(
      ((Reader_6)threadArray[ı]) getLockWaıtıngTıme())),
   }
  }
  else ıf (threadArray[ı] ınstanceof Wrıter_6)
  {
   ıf (((Wrıter_6)threadArray[ı]) getType() == ReaderWrıterType TW)
```

```
    {
      TW_list add(new Long(
         ((Writer_6)threadArray[i]) getLockWaitingTime())),
    }
    else
    {
      RW_list add(new Long(
         ((Writer_6)threadArray[i]) getLockWaitingTime())),
    }
  }
  else
  {
    if (((Upgrader)threadArray[i]) getType() == ReaderWriterType TU)
    {
      TU_list add(new Long(
         ((Upgrader)threadArray[i]) getLockWaitingTime())),
    }
    else
    {
      RU_list add(new Long(
         ((Upgrader)threadArray[i]) getLockWaitingTime())),
    }
  }
}

Object[] TR_arr = TR_list toArray(),
Object[] RR_arr = RR_list toArray(),
Object[] TW_arr = TW_list toArray(),
Object[] TU_arr = TU_list toArray(),
Object[] RW_arr = RW_list toArray(),
Object[] RU_arr = RU_list toArray(),

long avgRR = average(RR_arr),
long avgRU = average(RU_arr),
long avgRW = average(RW_arr),
long avgTR = average(TR_arr),
long avgTU = average(TU_arr),
long avgTW = average(TW_arr),

node nodeArray[] = {
   new node(avgTR, ReaderWriterType TR),
   new node(avgRR, ReaderWriterType RR),
   new node(avgTW, ReaderWriterType TW),
   new node(avgTU, ReaderWriterType TU),
   new node(avgRW, ReaderWriterType RW),
   new node(avgRU, ReaderWriterType RU)},

bubbleSort(nodeArray, 6),

System out println("\nThe average times spent in milliseconds to obtain a "
               + "lock in algorithm_6 \n"),
for (int i = 0, i < 6, i++)
{
  switch (nodeArray[i] type)
  {
    case ReaderWriterType RR
      System out print("RR Average = " + avgRR + " ["),
      display(RR_arr);
      System out println("]\n"),
      break,
    case ReaderWriterType RU
      System out print("RU Average = " + avgRU + " ["),
```

```
        dısplay(RU_arr),
        System out prıntln("]\n"),
        break;
      case ReaderWrıterType RW
        System out prınt("RW Average = " + avgRW + " ["),
        dısplay(RW_arr),
        System out prıntln("]\n");
        break,
      case ReaderWrıterType TR
        System out prınt("TR Average = " + avgTR + " ["),
        dısplay(TR_arr),
        System out prıntln("]\n"),
        break,
      case ReaderWrıterType TU
        System out prınt("TU Average = " + avgTU + " ["),
        dısplay(TU_arr),
        System out prıntln("]\n"),
        break,
      case ReaderWrıterType TW
        System out prınt("TW Average = " + avgTW + " ["),
        dısplay(TW_arr);
        System out prıntln("]\n"),
        break,
      default
        System out prıntln("Oop! Somerthıng must be wrong "),
        break,

    }
   }
  }
}

/*****************************************************************************************************************
 Thıs class defınes the types of the share semaphores rc[0], rc[1] and rc[2]
 Thıs type ıs ınıtıalızed to be IR and the supported types are IR, IW, R and U

*****************************************************************************************************************/
class Type_6
{
 prıvate ınt value,

 /*---------------------------------------------------------------------------------------------------------------
  Functıon·
  Type_6

  Descrıptıon
  Default constructor for Type_5 class  The default type ıs IR

  Parameters
  None

  Return
  None
  -------------------------------------------------------------------------------------------------------------*/
 Type_6()
 {
  value = IR,
 }

 /*---------------------------------------------------------------------------------------------------------------
  Functıon
  getType
```

```
  Description
  get a type of a process

  Parameters
  None

  Return
  a value of a type
  -----------------------------------------------------------------------------------------------------------------------------*/
 int getType()
 {
  return value,
 }

 /*-----------------------------------------------------------------------------------------------------------------------------
  Function
  setType

  Description
  set a type of a process

  Parameters
  a value of a type

  Return
  None
  -----------------------------------------------------------------------------------------------------------------------------*/
 void setType(int v)
 {
  value = v,
 }

 final static int IR = 0x1000,
 final static int IW = 0x1001,
 final static int R = 0x1002,
 final static int U = 0x1003,
}

/*****************************************************************************************************************************
 This class specifies the return value of some of the methods of Resource_6,
 Table_6 and Record_6

*****************************************************************************************************************************/
class ReturnValue_2
{
 int smp, // The current sharing semaphore
 int count, // The count of current sharing semaphore
}

/*****************************************************************************************************************************
 This is the base class of Table_6 and Record_6 classes  It simulates the
 table that contains records  It implements fair and efficient readers and writers
 with intent to read and write lock and upgrade lock

*****************************************************************************************************************************/
class Resource_6
{
 protected Semaphore_6 pu, // controls access to the pre_upgrade
 protected Semaphore_6 u, // controls access to upgrade
 protected Semaphore_6 w, // controls access to the resource
 protected Semaphore_6 pw, // controls access to the outer semaphore
```

```
protected Semaphore_6[] rc, // share semaphore controlling access to count[]
protected int[] count; // counters for share semaphores rc[0], rc[1] and rc[2]
protected Type_6[] type, // type for share semaphores rc[0], rc[1] and rc[2]
protected int prm, // which of share semaphores is primary, initially 0

/*-----------------------------------------------------------------------------------------------------------------------
 Function
 Resource_6

 Description
 Constructor for Resource_6 class  It initializes the various variable

 Parameters
 None

 Return·
 None
 ----------------------------------------------------------------------------------------------------------------------*/
public Resource_6()
{
  pu = new Semaphore_6(),
  u = new Semaphore_6(),
  w = new Semaphore_6(),
  pw = new Semaphore_6(),
  rc = new Semaphore_6[3],
  count = new int[3],
  type = new Type_6[4],
  prm = 0,

  for (int i = 0, i < 3, i++)
    rc[i] = new Semaphore_6(),
  for (int i = 0, i < 4, i++)
    type[i] = new Type_6(),
}

/*-----------------------------------------------------------------------------------------------------------------------
 Function
 startRead

 Description
 Starts a read process according to fair and efficient readers and writers
 with intent to read and write lock and upgrade lock

 Parameters·
 readerNum - the ID number of a reader

 Return
 a ReturnValue_2 object
 ----------------------------------------------------------------------------------------------------------------------*/
public ReturnValue_2 startRead(int readerNum)
{
  int smp, // indicates which of rc[0], rc[1] and rc[2] currently to use
  ReturnValue_2 retVal = new ReturnValue_2(),

  pu.P(), // requests the pre_upgrade
  pu V(), // releases the pre_upgrade
  pw P(), // requests the outer semaphore
  pw V(), // releases the outer semaphore

  if (type[prm] getType() != Type_6 IW)
    smp = prm,
  else
```

```
   smp = (prm + 1) % 3,
  retVal smp = smp,

  rc[smp] P(),
  count[smp]++,
  retVal count = count[smp],
  if (type[smp] getType() == Type_6 IR)
    type[smp] setType(Type_6 R),
  // the first reader blocks writer and other processes waiting on the outer
  // semaphore
  if (count[smp] == 1)
  {
    w P(),
    pw P(),
  }
  rc[smp] V(),

  return retVal,
}

/*------------------------------------------------------------------------------------------
 Function
 endRead

 Description
 Ends a read process according to fair and efficient readers and writers
 with intent to read and write lock and upgrade lock

 Parameters
 readerNum - the ID number of a reader
 val - a ReturnValue_2 object

 Return·
 None
 -----------------------------------------------------------------------------------------*/
public void endRead(int readerNum, ReturnValue_2 val)
{
  rc[val smp] P(),
  count[val.smp]--,
  // the last reader unblocks other processes waiting on the outer semaphore
  // and table
  if (count[val smp] == 0)
  {
    type[val smp] setType(Type_6 IR),
    prm = (prm + 1) % 3;
    pw V(),
    w V(),
  }
  rc[val smp] V(),
}

/*------------------------------------------------------------------------------------------
 Function
 startWrite

 Description
 Starts a write process according to fair and efficient readers and writers
 with intent to read and write lock and upgrade lock

 Parameters
 writerNum - the ID number of a writer
```

```
 Return
 None
 ---------------------------------------------------------------------------------------------------------------------------*/
public void startWrite(int writerNum)
{
  pu P(), // requests the pre_upgrade
  pu V(), // releases the pre_upgrade
  pw P(), // requests the outer semaphore
  pw V(), // releases the outer semaphore
  w P(), // requests the table access
}

/*---------------------------------------------------------------------------------------------------------------------------
 Function
 endWrite

 Description
 Ends a write process according to fair and efficient readers and writers
 with intent to read and write lock and upgrade lock

 Parameters
 writerNum - the ID number of a writer

 Return
 None
 ---------------------------------------------------------------------------------------------------------------------------*/
public void endWrite(int writerNum)
{
  w V(), // release the table access
}

/*---------------------------------------------------------------------------------------------------------------------------
 Function·
 startIntentRead

 Description
 Starts an intent to read process according to fair and efficient readers and writers
 with intent to read and write lock and upgrade lock.

 Parameters
 readerNum - the ID number of a reader

 Return
 a ReturnValue_2 object
 ---------------------------------------------------------------------------------------------------------------------------*/
public ReturnValue_2 startIntentRead(int readerNum)
{
  int smp, // indicates which of rc[0], rc[1] and rc[2] currently to use
  ReturnValue_2 retVal = new ReturnValue_2(),

  pu P(), // requests the pre_upgrade
  pu V(), // releases the pre_upgrade
  pw P(), // requests the outer semaphore
  pw V(), // releases the outer semaphore

  smp = prm,
  retVal smp = smp,

  rc[smp] P(),
  count[smp]++,
  retVal.count = count[smp],
  if (count[smp] == 1)
```

```
  {
    w P(),
    pw P(),
  }
  rc[smp] V(),

  return retVal,
}

/*-----------------------------------------------------------------------------------------------------------------------------
 Function
 endIntentRead

 Description
 Ends an Intent to read process according to fair and efficent reader and
 writer algorithm with intent to read and intent to write

 Parameters
 readerNum - the ID number of a reader
 val -  a returnValue_2 object

 Return
 None
 -----------------------------------------------------------------------------------------------------------------------------*/
public void endIntentRead(int readerNum, ReturnValue_2 val)
{
  rc[val smp] P(),
  count[val smp]--,
  // the last reader unblocks other processes waiting on the outer semaphore
  // and table
  if (count[val smp] == 0)
  {
    type[val smp] setType(Type_6 IR),
    prm = (prm + 1) % 3,
    pw V(),
    w V(),
  }
  rc[val smp] V(),
}

/*-----------------------------------------------------------------------------------------------------------------------------
 Function
 startIntentWrite

 Description
 Starts an intent to write process according to fair and efficient readers and writers
 with intent to read and write lock and upgrade lock

 Parameters
 writeNum - the ID number of a writer

 Return
 a ReturnValue_2 object
 -----------------------------------------------------------------------------------------------------------------------------*/
public ReturnValue_2 startIntentWrite(int writerNum)
{
  int smp; // indicates which of rc[0], rc[1] and rc[2] currently to use
  ReturnValue_2 retVal = new ReturnValue_2(),

  pu P(), // requests the pre_upgrage
  pu V(), // releases the pre_upgrade
  pw P(), // requests the outer semaphore
```

```
 pw V(), // releases the outer semaphore

 if (type[prm] getType() == Type_6 IW || type[prm] getType() == Type_6 IR)
   smp = prm,
 else if (type[ (prm + 1) % 3] getType() == Type_6 IW ||
          type[ (prm + 1) % 3] getType() == Type_6 IR)
   smp = (prm + 1) % 3,
 else
   smp = (prm + 2) % 3,
 retVal smp = smp,

 rc[smp] P(),
 count[smp]++,
 retVal count = count[smp],
 if (type[smp] getType() == Type_6 IR)
   type[smp] setType(Type_6 IW),
 if (count[smp] == 1)
 {
   w P(),
   pw P(),
 }
 rc[smp] V(),

 return retVal,
}

/*----------------------------------------------------------------------------------------------------------------------------------
 Function
 endIntentRead

 Description
 Ends an Intent to read process according to fair and efficient readers and
 writers with intent to read and write lock and upgrade lock

 Parameters
 readerNum - the ID number of a reader
 val -  a returnValue_2 object

 Return
 None
 ----------------------------------------------------------------------------------------------------------------------------------*/
public void endIntentWrite(int writerNum, ReturnValue_2 val)
{
 rc[val smp] P(),
 count[val smp]--,

 if (count[val smp] == 0)
 {
   type[val smp] setType(Type_6 IR),
   prm = (prm + 1) % 3,
   pw V(),
   w V(),
 }
 rc[val smp] V(),
}

/*----------------------------------------------------------------------------------------------------------------------------------
 Function
 startUpgrade

 Description
 Starts an upgrade process according to fair and efficient readers and writers
```

```
 with intent to read and write lock and upgrade lock

 Parameters
 upgraderNum - the ID number of the upgrader

 Return
 a ReturnValue_2 object
 ---------------------------------------------------------------------------------------------------------------------------*/
public ReturnValue_2 startUpgrade(int upgraderNum)
{
  int smp, // indicates which of rc[0], rc[1] and rc[2] currently to use
  ReturnValue_2 retVal = new ReturnValue_2(),

  pu P(),
  u P(),
  pu V(),
  pw P(),
  pw V(),

  if (count[prm] == 0)
    smp = prm,
  else if (count[ (prm + 1) % 3] == 0)
    smp = (prm + 1) % 3,
  else
    smp = (prm + 2) % 3,
  retVal smp = smp,

  rc[smp] P(),
  count[smp]++,
  retVal count = count[smp],
  type[smp] setType(Type_6 U),
  if (count[smp] == 1)
  {
    w P(),
    pw P(),
  }
  rc[smp] V(),

  return retVal,
}

/*---------------------------------------------------------------------------------------------------------------------------
 Function
 busyWaiting

 Description
 Loops indefinitely if the count is greater than 1

 Parameters
 val -  a ReturnValue_2 object

 Return
 None
 ---------------------------------------------------------------------------------------------------------------------------*/
public void busyWaiting(ReturnValue_2 val)
{
  while (count[val smp] > 1) ,
}

/*---------------------------------------------------------------------------------------------------------------------------
 Function
 endUpgrade
```

```
 Description
 Ends an upgrade process according to fair and efficient readers and writers
 with intent to read and write lock and upgrade lock

 Parameters
 upgraderNum - the ID number of a writer
 val -  a ReturnValue_2 object

 Return
 None
 -----------------------------------------------------------------------------------------------------------------------*/
 public void endUpgrade(int upgraderNum, ReturnValue_2 val)
 {
  count[val smp] = 0,
  type[val smp] setType(Type_6 IR);
  prm = (prm + 1) % 3,
  pw V(),
  w V(),
  u V(),
 }
}

/************************************************************************************************************************
 This class inherits the Resource_6 class  It specifies a table operation

************************************************************************************************************************/
class Table_6 extends Resource_6
{
 private int num_of_records, // how many records in a table
 private Record_6[] records, // a array of record objects

 /*-----------------------------------------------------------------------------------------------------------------------
  Function
  Table_6

  Description
  Constructor for Table_6 class  It initializes the variours  variables

  Parameters
  num - how many records are in a table

  Return
  None
 -----------------------------------------------------------------------------------------------------------------------*/
 public Table_6(int num)
 {
  super(),
  num_of_records = num,
  records = new Record_6[num_of_records],

  for (int i = 0; i < num_of_records, i++)
    records[i] = new Record_6(i),
 }

 /*-----------------------------------------------------------------------------------------------------------------------
  Function
  getNumOfRecords

  Description·
  Gets the number of records in a table
```

```
 Parameters
 None

 Return
 The number of records
 ------------------------------------------------------------------------------------------------------------------------------*/
public int getNumOfRecords()
{
  return num_of_records,
}

/*------------------------------------------------------------------------------------------------------------------------------
 Function
 getRecord

 Description
 Gets a record ID in a table

 Parameters
 index - the index of the records in the table

 Return      record ID
 ------------------------------------------------------------------------------------------------------------------------------*/
public Record_6 getRecord(int index)
{
  return records[index],
}

/*------------------------------------------------------------------------------------------------------------------------------
  Function
  startRead

  Description
  Starts a read process according to fair and efficient readers and writers
  with intent to read and write lock and upgrade lock by overriding the same
  method in the base class

  Parameters
  readerNum - the ID number of a reader

  Return
  a ReturnValue_2 object
  ------------------------------------------------------------------------------------------------------------------------------*/
public ReturnValue_2 startRead(int readerNum)
{
  System out println("Reader " + readerNum + " wants to read from the TABLE"),
  return super startRead(readerNum),
}

/*------------------------------------------------------------------------------------------------------------------------------
 Function
 endRead

 Description
 Ends a read process according to fair and efficient readers and writers
 with intent to read and write lock and upgrade lock

 Parameters
 readerNum - the ID number of a reader
 val - a ReturnValue_2 object

 Return
```

```
 None
 ------------------------------------------------------------------------------------------------------------------------*/
public void endRead(int readerNum, ReturnValue_2 val)
{
  rc[val smp] P(),
  count[val smp]--,
  System out println("Reader " + readerNum +
                 " is done reading from the TABLE "
                 + " Count[" + val smp + "] = " + count[val smp]),
  // the last reader unblocks other processes waiting on the outer semaphore
  // and table
  if (count[val smp] == 0)
  {
    type[val smp] setType(Type_6 IR),
    prm = (prm + 1) % 3,
    pw V(),
    w V(),
  }
  rc[val smp] V(),
}

// Overrides the same method in the base class
public void startWrite(int writerNum)
{
  System out println("Writer " + writerNum + " wants to write to the TABLE"),
  super startWrite(writerNum),
}

// Overrides the same method in the base class
public void endWrite(int writerNum)
{
  System out println("Writer " + writerNum + " is done writing to the TABLE"),
  super endWrite(writerNum),
}

// Overrides the same method in the base class
public ReturnValue_2 startIntentRead(int readerNum)
{
  System out println("Reader " + readerNum + " wants to do intent read from "
                 + "the TABLE"),
  return super startIntentRead(readerNum),
}

/*-------------------------------------------------------------------------------------------------------------------------
 Function
 endIntentRead

 Description
 Ends an Intent to read process according to fair and efficient readers and
 writers with intent to read and write lock and upgrade lock

 Parameters
 readerNum - the ID number of a reader
 val - a ReturnValue_2 object

 Return
 None
 ------------------------------------------------------------------------------------------------------------------------*/
public void endIntentRead(int readerNum, ReturnValue_2 val)
{
  rc[val smp] P(),
  count[val smp]--,
```

```
  // the last reader unblocks other processes waiting on the outer semaphore
  // and table
  System out println("Reader " + readerNum + " is done intent read from the "
              + "TABLE  Count[" + val smp + "] = " + count[val smp]),

  if (count[val smp] == 0)
  {
    type[val smp] setType(Type_6 IR),
    prm = (prm + 1) % 3,
    pw V(),
    w V(),
  }
  rc[val smp] V(),
}

// Overrides the same method in the base class
public ReturnValue_2 startIntentWrite(int writerNum)
{
  System out println("Writer " + writerNum +
              " wants to do intent write to the"
              + " TABLE");
  return super startIntentWrite(writerNum),
}

/*-----------------------------------------------------------------------------------------------------------------------
 Function
 endIntentWrite

 Description
 Ends an Intent to write process according to fair and efficient readers and
 writers with intent to read and write lock and upgrade lock

 Parameters
 writeNum - the ID number of a writer
 val -  a ReturnValue_2 object

 Return
 None
 -----------------------------------------------------------------------------------------------------------------------*/
public void endIntentWrite(int writerNum, ReturnValue_2 val)
{
  rc[val smp] P(),
  count[val smp]--,
  System out println("Writer " + writerNum + " is done intent write to the"
              + " TABLE  Count[" + val smp + "] = " + count[val smp]),

  if (count[val smp] == 0)
  {
    type[val smp] setType(Type_6 IR),
    prm = (prm + 1) % 3,
    pw V(),
    w V(),
  }
  rc[val smp] V(),
}

// Overrides the same method in the base class
public ReturnValue_2 upgrader_startIntentWrite(int upgraderNum)
{
  System out println("Upgrader " + upgraderNum +
              " wants to do intent write to the"
              + " TABLE");
```

```
  return super startIntentWrite(upgraderNum),
 }

 /*--------------------------------------------------------------------------------------------------------------------------
  Function
  upgrader_endIntentWrite

  Description.
  Ends an Intent to write process according to fair and efficent reader and
  writer algorithm with intent to read, intent to write and upgrade lock

  Parameters
  upgraderNum - the ID number of a upgrader
  val - a returnValue_2 object

  Return
  None
  --------------------------------------------------------------------------------------------------------------------------*/
 public void upgrader_endIntentWrite(int upgraderNum, ReturnValue_2 val)
 {
  rc[val smp] P(),
  count[val smp]--,
  System out println("Upgrader " + upgraderNum +
               " is done intent write to the"
               + " TABLE  Count[" + val smp + "] = " + count[val smp]),

  if (count[val smp] == 0)
  {
   type[val smp] setType(Type_6 IR),
   prm = (prm + 1) % 3,
   pw V(),
   w V(),
  }
  rc[val smp] V(),
 }

 // Overrides the same method in the base class
 public ReturnValue_2 startUpgrade(int upgraderNum)
 {
  System out println("Upgrader " + upgraderNum +
               " wants to upgrade the TABLE"),
  return super startUpgrade(upgraderNum),
 }

 // Overrides the same method in the base class
 public void endUpgrade(int upgraderNum, ReturnValue_2 val)
 {
  System out println("Upgrader " + upgraderNum +
               " is done upgrading the TABLE"),
  super endUpgrade(upgraderNum, val),
 }
}

/***************************************************************************************************************************
 This is the record class that implements efficient read-write to simulate
 read and write on the individual records in a table   Along with others
 this class also has an instance variable that keeps track of the record's
 index in the table
 ***************************************************************************************************************************/
class Record_6 extends Resource_6
{
```

```
private int index,

/*---------------------------------------------------------------------------------------------------------------------------
 Function
 Record_6

 Description
 Constructor for Record_6 class  It initializes the various variables

 Parameter
 i - record id in a table
 ---------------------------------------------------------------------------------------------------------------------------*/
public Record_6(int i)
{
  super(),
  index = i,
}

// Overrides the same method in the base class
public ReturnValue_2 startRead(int readerNum)
{
  System out println("Reader " + readerNum + " wants to read from RECORD "
                   + index),
  return super startRead(readerNum),
}

/*---------------------------------------------------------------------------------------------------------------------------
 Function
 endRead

 Description
 Ends a read process according to fair and efficent reader and writer
 algorithm with intent to read, intent to write and upgrade lock

 Parameters
 readerNum - the ID number of a reader
 val - a ReturnValue_2 object

 Return
 None
 ---------------------------------------------------------------------------------------------------------------------------*/
public void endRead(int readerNum, ReturnValue_2 val)
{
  rc[val smp] P(),
  count[val smp]--,
  System out println("Reader " + readerNum + " is done reading from RECORD "
                   + index + "  Count_" + index + "[" + val smp +
                   "] = " + count[val smp]),

  // the last reader unblocks other processes waiting on the outer semaphore
  // and table
  if (count[val smp] == 0)
  {
    type[val smp] setType(Type_6 IR),
    prm = (prm + 1) % 3,
    pw V(),
    w V(),
  }
  rc[val smp] V(),
}

// Overrides the same method in the base class
```

```
 public void startWrite(int writerNum)
 {
   System out println("Writer " + writerNum + " wants to write to RECORD "
                  + index),
   super startWrite(writerNum),
 }

 // Overrides the same method in the base class
 public void endWrite(int writerNum)
 {
   System out println("Writer " + writerNum + " is done writing to RECORD "
                  + index),
   super endWrite(writerNum),
 }

 // Overrides the same method in the base class
 public ReturnValue_2 startUpgrade(int upgraderNum)
 {
   System out println("Upgrader " + upgraderNum + " wants to upgrade RECORD "
                  + index),
   return super startUpgrade(upgraderNum),
 }

 // Overrides the same method in the base class
 public void endUpgrade(int upgraderNum, ReturnValue_2 val)
 {
   System out println("Upgrader " + upgraderNum + " is done upgrading "
                  + "RECORD " + index),
   super endUpgrade(upgraderNum, val),
 }
}

/*****************************************************************************************************************
 This class defines the reader and writer types and provides methods for
 retrieving the types
*****************************************************************************************************************/
class ReaderWriterType
{
 private int value,

 ReaderWriterType(int type)
 {
  value = type;
 }

 int getReaderWriterType()
 {
  return value,
 }

 final static int TR = 0x100,
 final static int RR = 0x101,
 final static int TW = 0x102,
 final static int TU = 0x103,
 final static int RW = 0x104,
 final static int RU = 0x105,
}

/*****************************************************************************************************************
 This class calls the various methods to start and end a read process
*****************************************************************************************************************/
class Reader_6 extends Thread
```

```
{
  private int readerNum,  // the identification number of the reader
  private int recordNum,  // the index number of the record
  private Table_6 tbl,    // the table that the reader tries to access
  private int readTime,   // access time spent by the reader
  private ReaderWriterType t_r,   // the type of the reader (TR or RR)
  private long lockWaitingTime,   // the lock waiting time of the reader

  /*----------------------------------------------------------------------------------------------------------------------------------
   Function
   Reader_6

   Description
   Constructor function for the reader class   If the value of the recNum
   argument is -1, the reader wants to read the table as a whole   Otherwise,
   the reader wants to read a record whose index is contained in recNum

   Parameters
   id - the identification number of the reader thread
   recNum - the identification number of the record the reader will read   -1 if
          the reader reads the table only
   table - the table the reader will work on
   r_time - time the reader will spend for the reading operation

   Return
   None
   ----------------------------------------------------------------------------------------------------------------------------------*/
  public Reader_6(int id, int recNum, Table_6 table, int r_time)
  {
   readerNum = id,
   recordNum = recNum,
   tbl = table,
   readTime = r_time,

   if (recordNum == -1)
   {   // table reader
     t_r = new ReaderWriterType(ReaderWriterType TR),
   }
   else
   {   // record reader
     t_r = new ReaderWriterType(ReaderWriterType RR),
   }
  }

  /*----------------------------------------------------------------------------------------------------------------------------------
   Function
   getType

   Description
   Get the type of the reader (TR or RR)

   Parameters·
   None

   Return
   An integer that corresponds to the reader's type
   ----------------------------------------------------------------------------------------------------------------------------------*/
  public int getType()
  {
   return t_r getReaderWriterType(),
  }
```

```
/*-----------------------------------------------------------------------------------------------------------------------------------
 Function
 getLockWaitingTime

 Description
 Get the lock waiting time for the reader

 Parameters
 None

 Return
 A long integer that corresponds to the lock waiting time
 ---------------------------------------------------------------------------------------------------------------------------------*/
public long getLockWaitingTime()
{
  return lockWaitingTime,
}

/*---------------------------------------------------------------------------------------------------------------------------------
 Function
 run

 Description
 This function specifies how a reader thread runs

 Parameters
 None

 Return
 None
 ---------------------------------------------------------------------------------------------------------------------------------*/
public void run()
{
  if (recordNum == -1) { // reader will read the table as a whole
    Date start_req = new Date(),
    ReturnValue_2 val = tbl startRead(readerNum),
    Date obtain_req = new Date(),

    System out println("Reader " + readerNum + " is reading from the table  "
                  + "Count[" + val smp + "] = " + val count),
    Break_6 duration(readTime), // read reaTime milliseconds

    tbl endRead(readerNum, val),

    lockWaitingTime = obtain_req getTime() - start_req getTime(),
  }
  else
  { // reader will read an individual record
    Date start_req = new Date(),
    // do intent read on table first
    ReturnValue_2 val_1 = tbl startIntentRead(readerNum),
    // then read the record
    System out println("Reader " + readerNum + " is doing intent read from "
                  + "the table  Count[" + val_1 smp + "] = " +
                  val_1 count),
    ReturnValue_2 val_2 = tbl getRecord(recordNum) startRead(readerNum),
    Date obtain_req = new Date(),

    System out println("Reader " + readerNum + " is reading from record "
                  + recordNum + "  Count_" + recordNum + "[" + val_2 smp
                  + "] = " + val_2 count),
    Break_6 duration(readTime), // read readTime milliseconds
```

```
    tbl getRecord(recordNum) endRead(readerNum, val_2), //end reading
    tbl endIntentRead(readerNum, val_1), //end intent to reading

    // time to obtain a read lock
    lockWaitingTime = obtain_req getTime() - start_req getTime(),
   }
  }
}

/*******************************************************************************************************************
 This class calls the various methods to start and end a write process

*******************************************************************************************************************/
class Writer_6 extends Thread
{
 private int writerNum,   // the identification number of the writer
 private int recordNum,   // the index number of the record
 private Table_6 tbl,     // the table that the reader tries to access
 private int writeTime,   // access time spent by the writer
 private ReaderWriterType t_w,  // the type of the reader (TW, TU, RW or RU)
 private long lockWaitingTime,  // the lock waiting time of the reader

 /*-----------------------------------------------------------------------------------------------------------------
  Function
  Writer_6

  Description
  Constructor function for the writer class   If the value of the recNum
  argument is -1, the writer wants to read the table as a whole   Otherwise,
  the writer wants to write to a record whose index is contained in recNum

  Parameters
  id - the identification number of the writer thread
  recNum - the identification number of the record the writer will write   -1
         if the writer writes the table only
  table - the table the writer will work on
  w_time - time the writer will spend for the writing operation

  Return
  None
  -----------------------------------------------------------------------------------------------------------------*/
 public Writer_6(int id, int recNum, Table_6 table, int w_time)
 {
  writerNum = id,
  recordNum = recNum,
  tbl = table,
  writeTime = w_time,

  if (recordNum == -1)
  {   // table writer
    t_w = new ReaderWriterType(ReaderWriterType TW),
  }
  else
  {   // record writer
    t_w = new ReaderWriterType(ReaderWriterType RW),
  }
 }

 /*-----------------------------------------------------------------------------------------------------------------
  Function
  getType
```

```
 Descrıptıon
 Get the type of the wrıter (TW, or RW)

 Parameters
 None

 Return
 An ınteger that corresponds to the wrıter's type
 -----------------------------------------------------------------------------------------------------------------------------*/
public ınt getType()
{
  return t_w getReaderWrıterType(),
}

/*-------------------------------------------------------------------------------------------------------------------------------
 Functıon
 getLockWaıtıngTıme

 Descrıptıon·
 Get the lock waıtıng tıme for the wrıter

 Parameters
 None

 Return.
 A long ınteger that corresponds to the lock waıtıng tıme
 -----------------------------------------------------------------------------------------------------------------------------*/
public long getLockWaıtingTıme()
{
  return lockWaıtıngTıme,
}

/*-------------------------------------------------------------------------------------------------------------------------------
 Functıon
 run

 Descrıptıon
 Thıs functıon specıfies how a wrıter thread runs

 Parameters
 None

 Return
 None
 -------------------------------------------------------------------------------------------------------------------------------*/
public voıd run()
{
  // start a wrıte process
  ıf (recordNum == -1) { // wrıter wıll wrıte to the table as a whole
    Date start_req = new Date(),
    tbl startWrıte(writerNum),
    Date obtaın_req = new Date();

    System out prıntln("Wrıter " + wrıterNum + " ıs wrıtıng to the table"),
    Break_6 duratıon(wrıteTıme), // wrıte wrıteTıme mıllıseconds

    tbl endWrıte(writerNum),

    // the waıtıng tıme for obtaınıng a wrıte lock
    lockWaıtıngTıme = obtaın_req getTıme() - start_req getTıme(),
  }
```

```
   else
   { // wrıter wıll wrıte to an ındıvıdual record
     // start tımıng
     Date start_req = new Date(),
     // do ıntent wrıte to table first
     ReturnValue_2 val = tbl startIntentWrıte(wrıterNum),
     System out prıntln("Wrıter " + wrıterNum +
                   " ıs doıng ıntent wrıte to the "
                   + "table  Count[" + val smp + "] = " + val count),
     tbl getRecord(recordNum) startWrıte(wrıterNum), // wrıte to the record
     // end tımıng
     Date obtaın_req = new Date(),

     System out prıntln("Wrıter " + wrıterNum + " ıs wrıtıng to record "
                   + recordNum),
     Break_6 duratıon(wrıteTıme), // wrıte wrıteTıme mıllıseconds

     tbl getRecord(recordNum) endWrıte(wrıterNum),
     tbl endIntentWrıte(wrıterNum, val),

     // the waıtıng tıme for obtaınıng a wrıte lock
     lockWaıtıngTıme = obtaın_req getTıme() - start_req getTıme(),
   }
  }
}

/*****************************************************************************************************************************
 Thıs class calls the varıous methods to start and end an upgrader process
 ****************************************************************************************************************************/
class Upgrader extends Thread
{
  prıvate ınt upgraderNum,  // the ıdentıficatıon number of the upgrader
  prıvate ınt recordNum,    // the ındex number of the record
  prıvate Table_6 tbl,      // the table that the reader trıes to access
  prıvate ınt read_tıme,    // read tıme spent by the upgrader
  prıvate ınt wrıte_tıme,   // wrıte tıme spent by the upgrader
  prıvate ReaderWrıterType t_u,  // the type of the upgrader (TU, or RU)
  prıvate long lockWaıtıngTıme,  // the lock waıtıng tıme of the upgrader

  /*--------------------------------------------------------------------------------------------------------------------------
   Functıon
   Upgrader

   Descrıptıon
   Constructor functıon for the upgrader class   If the value of the recNum
   argument ıs -1, the upgrader wants to read the table as a whole   Otherwıse,
   the upgrader wants to upgrade to a record whose ındex ıs contaıned ın recNum
   An upgrader spents r_tıme to read before spendıng w_tıme to update

   Parameters
   ıd - the ıdentıficatıon number of the upgrader thread
   recNum - the ıdentıficatıon number of the record the upgrader wıll upgrade
          -1 ıf the upgrader upgrade the table only
   table - the table the upgrader wıll work on
   r_tıme - tıme the upgrader wıll spend for the readıng operatıon
   w_tıme - tıme the upgrader wıll spend for the wrıtıng operatıon

   Return
   None
   --------------------------------------------------------------------------------------------------------------------------*/
  publıc Upgrader(ınt ıd, ınt recNum, Table_6 table, ınt r_tıme, ınt w_tıme)
  {
```

```
  upgraderNum = ıd,
  recordNum = recNum;
  tbl = table,
  read_tıme = r_tıme,
  wrıte_tıme = w_tıme,

  ıf (recordNum == -1)
  {   // table upgrader
    t_u = new ReaderWrıterType(ReaderWrıterType TU),
  }
  else
  {   // record upgrader
    t_u = new ReaderWrıterType(ReaderWrıterType RU),
  }
}

/*--------------------------------------------------------------------------------------------------------------------------------------
 Functıon
 getType

 Descrıptıon
 Get the type of the wrıter (TU or RU)

 Parameters
 None

 Return
 An ınteger that corresponds to the wrıter's type
 --------------------------------------------------------------------------------------------------------------------------------------*/
public ınt getType()
{
  return t_u getReaderWrıterType(),
}

/*--------------------------------------------------------------------------------------------------------------------------------------
 Functıon
 getLockWaıtıngTıme

 Descrıptıon
 Get the lock waıtıng tıme for the wrıter

 Parameters·
 None

 Return
 A long ınteger that corresponds to the lock waıtıng tıme
 --------------------------------------------------------------------------------------------------------------------------------------*/
public long getLockWaıtıngTıme()
{
  return lockWaıtıngTıme,
}

/*--------------------------------------------------------------------------------------------------------------------------------------
 Functıon
 run

 Descrıptıon
 Thıs functıon specıfies how a wrıter thread runs

 Parameters
 None
```

```
Return
None
------------------------------------------------------------------------------------------------------------------------------------------*/
public void run()
{
  // start a upgrade process
  if (recordNum == -1)
  { // Upgrader will upgrade to the table as a whole
    Date start_req = new Date(), // start timing
    ReturnValue_2 val = tbl startUpgrade(upgraderNum),
    Date obtain_req = new Date(), // end timing

    System out println("upgrader " + upgraderNum +
                  " is reading from the table  "
                  + "Count[" + val smp + "] = " + val count),
    // read read_time milliseconds
    Break_6 duration(read_time),

    tbl busyWaiting(val),

    System out println("upgrader " + upgraderNum + " is writing to the table"),
    // update upgradeTime milliseconds
    Break_6 duration(write_time),

    // end update
    tbl endUpgrade(upgraderNum, val),

    // obtain a waiting time for a table upgrader lock
    lockWaitingTime = obtain_req getTime() - start_req getTime(),
  }
  else
  { // Upgrader will upgrade an individual record
    Date start_req = new Date(),
    // do intent write to table first
    ReturnValue_2 val_1 = tbl upgrader_startIntentWrite(upgraderNum),
    System out println("upgrader " + upgraderNum +
                  " is doing intent write to the "
                  + "table  Count[" + val_1 smp + "] = " + val_1 count),
    ReturnValue_2 val_2 = tbl getRecord(recordNum) startUpgrade(upgraderNum),
    Date obtain_req = new Date(),

    System out println("Upgrader " + upgraderNum + " is reading from record "
                  + recordNum + "  Count_" + recordNum + "[" + val_2 smp
                  + "] = " + val_2 count),
    // read read_time milliseconds
    Break_6 duration(read_time),

    // wait there is only the upgrader in the critical section
    tbl getRecord(recordNum) busyWaiting(val_2),

    System out println("Upgrader " + upgraderNum + " is writing to record "
                  + recordNum),
    // update write_time milliseconds
    Break_6 duration(write_time),

    // end update
    tbl getRecord(recordNum) endUpgrade(upgraderNum, val_2),
    // end intent to write
    tbl upgrader_endIntentWrite(upgraderNum, val_1),

    // obtain a waiting time for a record upgrader lock
    lockWaitingTime = obtain_req getTime() - start_req getTime(),
```

```
   }
  }
}

/**************************************************************************************************************
 This class implements semaphore using Java syschronization
 **************************************************************************************************************/
final class Semaphore_6
{
 private int value,   // the value of the semaphore

 /*----------------------------------------------------------------------------------------------------------------------------
  Function
  Semaphore_6

  Description
  default constructor for Semaphore_6

  Parameters
  None

  Return
  None
  ----------------------------------------------------------------------------------------------------------------------------*/
 public Semaphore_6()
 {
  value = 1,
 }

 /*----------------------------------------------------------------------------------------------------------------------------
  Function
  Semaphore_6

  Description
  constructor for Semaphore_6

  Parameter
  v - An integer value for the semaphore

  Return
  None
  ----------------------------------------------------------------------------------------------------------------------------*/
 public Semaphore_6(int v)
 {
  value = v,
 }

 /*----------------------------------------------------------------------------------------------------------------------------
  Function
  P

  Description
  This function call the wait() to sleep when the value of Semaphore less than
  or equal 0  If the value of Semaphore is a positive number, decrements by 1

  Parameters
  None

  Return
  None
  ----------------------------------------------------------------------------------------------------------------------------*/
 public synchronized void P()
```

```
  {
   while (value <= 0)
   {
    try
    {
     wait(),
    }
    catch (InterruptedException e) {}
   }
   value--,
  }

  /*-----------------------------------------------------------------------------------------------------------------------------------------
   Function
   V

   Description
   This function increments the semaphore value by 1 and call the notify()
   function to wakeup a process that is waiting on the semaphore if it has any

   Parameters
   None

   Return
   None
   ----------------------------------------------------------------------------------------------------------------------------------------*/
  public synchronized void V()
  {
   ++value,
   notify(),
  }
}

/****************************************************************************************************************************************
 The class specified the duration of access time

****************************************************************************************************************************************/
final class Break_6
{
  /*-----------------------------------------------------------------------------------------------------------------------------------------
   Function
   duration

   Description
   The function specifies the duration of access time in milliseconds

   Parameter
   milliseconds - how many milliseconds the access time is

   Return
   None
   ----------------------------------------------------------------------------------------------------------------------------------------*/
  public static void duration(int milliseconds)
  {
   try
   {
    Thread sleep(milliseconds),
   }
   catch (InterruptedException e) {}
  }
}
```

APPENDIX B

THE RUNNING RESULT

The following is the sample result of running three versions of program.

1. **Sample result of running Verbose version**
2. **Sample result of running Throughput version**
3. **Sample result of running Turnaround version**

// Sample result of running Verbose version

```
A - Rebuild verbose version
B - Rebuild throughput version
C - Rebuild turnaround version
D - Run verbose version
E - Run throughput version
F - Run turnaround version
Q - Quit

Choose an option  A
Please enter the number of requests (default = 15, q to quit) 20
Please enter the access time for table read  40
Please enter the access time for record read  20
Please enter the access time for table write 60
Please enter the access time for record write  30
Please enter the interval time  0
Start running algorithm_1
writer 0 wants to write
writer 0 is writing
writer 1 wants to write
writer 2 wants to write
writer 3 wants to write
reader 0 wants to read
writer 4 wants to write
reader 1 wants to read
writer 5 wants to write.
reader 2 wants to read
reader 3 wants to read
writer 6 wants to write
reader 4 wants to read
writer 7 wants to write
reader 5 wants to read
writer 8 wants to write
reader 6 wants to read
reader 7 wants to read
writer 9 wants to write
writer 10 wants to write
reader 8 wants to read
writer 0 is done writing
writer 1 is writing
writer 1 is done writing
writer 2 is writing
writer 2 is done writing
writer 3 is writing
writer 3 is done writing
reader 0 is reading  Count = 1
reader 1 is reading  Count = 2
reader 2 is reading  Count = 3
reader 3 is reading  Count = 4
reader 4 is reading. Count = 5
reader 5 is reading  Count = 6
reader 6 is reading  Count = 7
reader 7 is reading  Count = 8
reader 8 is reading. Count = 9
reader 0 is done reading  Count = 8
reader 1 is done reading  Count = 7
reader 2 is done reading  Count = 6
reader 4 is done reading  Count = 5
reader 5 is done reading  Count = 4
```

reader 7 is done reading Count = 3
reader 8 is done reading Count = 2
reader 3 is done reading Count = 1
reader 6 is done reading Count = 0
writer 4 is writing
writer 4 is done writing
writer 5 is writing
writer 5 is done writing
writer 6 is writing
writer 6 is done writing
writer 7 is writing
writer 7 is done writing
writer 8 is writing
writer 8 is done writing
writer 9 is writing
writer 9 is done writing
writer 10 is writing
writer 10 is done writing

Time spent for algorithm_1 0 531 seconds

The average times spent in milliseconds to obtain a lock in algorithm_1

TR Average = 105 [110 100]

RR Average = 105 [110 110 110 110 100 100 100]

RW Average = 183 [0 30 60 80 451 481]

TW Average = 200 [200]

RU Average = 240 [150 260 310]

TU Average = 350 [350]

Start running algorithm_2
writer 0 wants to write
writer 0 is writing
writer 1 wants to write
writer 2 wants to write
writer 3 wants to write
reader 0 wants to read
writer 4 wants to write
reader 1 wants to read
writer 5 wants to write
reader 2 wants to read
reader 3 wants to read
writer 6 wants to write
reader 4 wants to read
writer 7 wants to write
reader 5 wants to read
writer 8 wants to write
reader 6 wants to read
reader 7 wants to read
writer 9 wants to write
writer 10 wants to write
reader 8 wants to read
writer 0 is done writing

writer 1 is writing
writer 1 is done writing
writer 2 is writing
writer 2 is done writing
writer 3 is writing
writer 3 is done writing
writer 4 is writing
writer 4 is done writing
writer 5 is writing
writer 5 is done writing
writer 6 is writing
writer 6 is done writing
writer 7 is writing
writer 7 is done writing
writer 8 is writing
writer 8 is done writing
writer 9 is writing
writer 9 is done writing
writer 10 is writing
writer 10 is done writing
reader 0 is reading Count = 1
reader 1 is reading Count = 2
reader 2 is reading Count = 3
reader 3 is reading Count = 4
reader 4 is reading Count = 5
reader 5 is reading Count = 6
reader 6 is reading Count = 7
reader 7 is reading Count = 8
reader 8 is reading Count = 9
reader 0 is done reading Count = 8
reader 1 is done reading Count = 7
reader 2 is done reading Count = 6
reader 4 is done reading Count = 5
reader 5 is done reading Count = 4
reader 7 is done reading Count = 3
reader 8 is done reading Count = 2
reader 3 is done reading Count = 1
reader 6 is done reading Count = 0

Time spent for algorithm_2 0 641 seconds

The average times spent in milliseconds to obtain a lock in algorithm_2

TW Average = 160 [160]

RU Average = 196 [110 220 260]

RW Average = 203 [0 20 50 80 520 551]

TU Average = 310 [310]

RR Average = 585 [591 591 591 581 581 581 581]

TR Average = 586 [591 581]

Start running algorithm_3
writer 0 wants to write
writer 0 is writing
writer 1 wants to write
writer 2 wants to write

```
writer 3 wants to write
reader 0 wants to read
writer 4 wants to write
reader 1 wants to read
writer 5 wants to write
reader 2 wants to read
reader 3 wants to read
writer 6 wants to write
reader 4 wants to read
writer 7 wants to write
reader 5 wants to read
writer 8 wants to write
reader 6 wants to read
reader 7 wants to read
writer 9 wants to write
writer 10 wants to write
reader 8 wants to read
writer 0 is done writing
writer 1 is writing
writer 1 is done writing
writer 2 is writing
writer 2 is done writing
writer 3 is writing.
writer 3 is done writing
reader 0 is reading  Count = 1
reader 0 is done reading  Count = 0
writer 4 is writing
writer 4 is done writing
reader 1 is reading  Count = 1
reader 1 is done reading  Count = 0
writer 5 is writing
writer 5 is done writing
reader 2 is reading  Count = 1
reader 3 is reading  Count = 2
reader 2 is done reading  Count = 1
reader 3 is done reading  Count = 0
writer 6 is writing
writer 6 is done writing
reader 4 is reading. Count = 1
reader 4 is done reading  Count = 0
writer 7 is writing
writer 7 is done writing
reader 5 is reading  Count = 1
reader 5 is done reading  Count = 0
writer 8 is writing
writer 8 is done writing
reader 6 is reading  Count = 1
reader 7 is reading  Count = 2
reader 7 is done reading  Count = 1
reader 6 is done reading  Count = 0
writer 9 is writing
writer 9 is done writing
writer 10 is writing
writer 10 is done writing
reader 8 is reading. Count = 1
reader 8 is done reading  Count = 0
```

Time spent for algorithm_3 0 681 seconds

The average times spent in milliseconds to obtain a lock in algorithm_3

TW Average = 210 [210]

RW Average = 228 [10 30 60 90 581 601]

RU Average = 270 [140 300 370]

RR Average = 358 [120 190 260 350 420 540 631]

TR Average = 400 [260 540]

TU Average = 440 [440]

```
Start running algorithm_4
writer 0 wants to write
writer 0 is writing
writer 1 wants to write
writer 2 wants to write
writer 3 wants to write.
reader 0 wants to read
writer 4 wants to write.
reader 1 wants to read
writer 5 wants to write
reader 2 wants to read
reader 3 wants to read
writer 6 wants to write
reader 4 wants to read
writer 7 wants to write
reader 5 wants to read
writer 8 wants to write
reader 6 wants to read.
reader 7 wants to read
writer 9 wants to write
writer 10 wants to write
reader 8 wants to read
writer 0 is done writing.
writer 1 is writing
writer 1 is done writing
writer 2 is writing
writer 2 is done writing
writer 3 is writing
writer 3 is done writing
reader 0 is reading  Count = 1
reader 1 is reading. Count = 2
reader 2 is reading  Count = 3
reader 3 is reading  Count = 4
reader 4 is reading  Count = 5
reader 5 is reading  Count = 6
reader 6 is reading  Count = 7
reader 7 is reading  Count = 8
reader 8 is reading  Count = 9
reader 0 is done reading  Count = 9
reader 1 is done reading  Count = 8
reader 2 is done reading  Count = 7
reader 4 is done reading  Count = 6
reader 5 is done reading  Count = 5
reader 7 is done reading  Count = 4
reader 8 is done reading  Count = 3
reader 3 is done reading  Count = 2
reader 6 is done reading  Count = 1
writer 4 is writing
writer 4 is done writing.
```

writer 5 is writing
writer 5 is done writing
writer 6 is writing
writer 6 is done writing
writer 7 is writing
writer 7 is done writing
writer 8 is writing
writer 8 is done writing
writer 9 is writing
writer 9 is done writing
writer 10 is writing.
writer 10 is done writing

Time spent for algorithm_4 0 541 seconds

The average times spent in milliseconds to obtain a lock in algorithm_4

TR Average = 110 [110 110]

RR Average = 112 [120 120 120 110 110 110 100]

RW Average = 190 [10 30 60 90 460 490]

TW Average = 210 [210]

RU Average = 243 [160 260 310]

TU Average = 360 [360]

Start running algorithm_5 .
Writer 0 wants to do intent write to the TABLE
Writer 0 is doing intent write to the table Count[0] = 1
Writer 0 wants to write to RECORD 3
Writer 0 is writing to record 3
Writer 1 wants to do intent write to the TABLE
Writer 2 wants to do intent write to the TABLE
Writer 3 wants to do intent write to the TABLE
Reader 0 wants to do intent read from the TABLE
Writer 4 wants to do intent write to the TABLE
Reader 1 wants to do intent read from the TABLE
Writer 5 wants to write to the TABLE
Reader 2 wants to do intent read from the TABLE
Reader 3 wants to read from the TABLE
Writer 6 wants to do intent write to the TABLE
Reader 4 wants to do intent read from the TABLE
Writer 7 wants to do intent write to the TABLE
Reader 5 wants to do intent read from the TABLE
Writer 8 wants to write to the TABLE
Reader 6 wants to read from the TABLE
Reader 7 wants to do intent read from the TABLE
Writer 9 wants to do intent write to the TABLE
Writer 10 wants to do intent write to the TABLE
Reader 8 wants to do intent read from the TABLE
Writer 0 is done writing to RECORD 3
Writer 0 is done intent write to the TABLE. Count[0] = 0
Writer 1 is doing intent write to the table Count[1] = 1
Writer 1 wants to write to RECORD 1
Writer 1 is writing to record 1
Writer 1 is done writing to RECORD 1
Writer 1 is done intent write to the TABLE Count[1] = 0

Writer 3 is doing intent write to the table Count[0] = 1
Writer 3 wants to write to RECORD 2
Writer 3 is writing to record 2
Writer 3 is done writing to RECORD 2
Writer 3 is done intent write to the TABLE Count[0] = 0
Writer 4 is doing intent write to the table Count[1] = 1
Writer 4 wants to write to RECORD 0
Writer 4 is writing to record 0
Writer 4 is done writing to RECORD 0
Writer 4 is done intent write to the TABLE Count[1] = 0
Writer 5 is writing to the table
Writer 5 is done writing to the TABLE
Reader 2 is doing intent read from the table Count[0] = 1
Reader 2 wants to read from RECORD 2
Reader 2 is reading from record 2 Count_2[0] = 1
Reader 3 is reading from the table Count[0] = 2
Writer 6 is doing intent write to the table Count[0] = 3
Writer 6 wants to write to RECORD 2
Reader 4 is doing intent read from the table Count[0] = 4
Reader 4 wants to read from RECORD 0
Reader 4 is reading from record 0 Count_0[0] = 1
Writer 7 is doing intent write to the table Count[0] = 5
Writer 7 wants to write to RECORD 0
Reader 5 is doing intent read from the table Count[0] = 6
Reader 5 wants to read from RECORD 2
Reader 6 is reading from the table Count[0] = 7
Reader 7 is doing intent read from the table Count[0] = 8
Reader 7 wants to read from RECORD 1
Writer 9 is doing intent write to the table Count[0] = 9
Writer 9 wants to write to RECORD 4
Writer 9 is writing to record 4
Writer 2 is doing intent write to the table Count[0] = 11
Writer 2 wants to write to RECORD 2
Writer 10 is doing intent write to the table Count[0] = 10
Writer 10 wants to write to RECORD 4
Reader 7 is reading from record 1 Count_1[0] = 1
Reader 8 is doing intent read from the table Count[0] = 12
Reader 8 wants to read from RECORD 2
Reader 0 is doing intent read from the table Count[0] = 13
Reader 0 wants to read from RECORD 1
Reader 1 is doing intent read from the table Count[0] = 14
Reader 1 wants to read from RECORD 0
Reader 2 is done reading from RECORD 2 Count_2[0] = 0
Reader 2 is done intent read from the TABLE Count[0] = 13
Reader 4 is done reading from RECORD 0 Count_0[0] = 0
Reader 4 is done intent read from the TABLE Count[0] = 12
Writer 6 is writing to record 2
Writer 7 is writing to record 0
Writer 9 is done writing to RECORD 4
Writer 9 is done intent write to the TABLE Count[0] = 11
Reader 7 is done reading from RECORD 1. Count_1[0] = 0
Reader 7 is done intent read from the TABLE. Count[0] = 10
Writer 10 is writing to record 4
Reader 0 is reading from record 1 Count_1[1] = 1
Reader 3 is done reading from the TABLE Count[0] = 9
Reader 6 is done reading from the TABLE Count[0] = 8
Reader 0 is done reading from RECORD 1. Count_1[1] = 0
Reader 0 is done intent read from the TABLE Count[0] = 7
Writer 10 is done writing to RECORD 4
Writer 10 is done intent write to the TABLE Count[0] = 6
Writer 6 is done writing to RECORD 2
Writer 2 is writing to record 2

Writer 6 is done intent write to the TABLE Count[0] = 5
Writer 7 is done writing to RECORD 0
Writer 7 is done intent write to the TABLE Count[0] = 4
Reader 1 is reading from record 0 Count_0[1] = 1
Reader 1 is done reading from RECORD 0 Count_0[1] = 0
Reader 1 is done intent read from the TABLE Count[0] = 3
Writer 2 is done writing to RECORD 2
Writer 2 is done intent write to the TABLE Count[0] = 2
Reader 5 is reading from record 2 Count_2[1] = 1
Reader 8 is reading from record 2 Count_2[1] = 2
Reader 5 is done reading from RECORD 2 Count_2[1] = 1
Reader 5 is done intent read from the TABLE Count[0] = 1
Reader 8 is done reading from RECORD 2 Count_2[1] = 0
Reader 8 is done intent read from the TABLE Count[0] = 0
Writer 8 is writing to the table
Writer 8 is done writing to the TABLE

Time spent for algorithm_5 0 48 seconds

The average times spent in milliseconds to obtain a lock in algorithm_5

RW Average = 150 [0 30 320 60 230 260]

TW Average = 180 [180]

RU Average = 220 [140 260 260]

TR Average = 240 [240 240]

RR Average = 282 [280 320 240 240 340 230 330]

TU Average = 360 [360]

Start running algorithm_6 .
Writer 0 wants to do intent write to the TABLE
Writer 0 is doing intent write to the table Count[0] = 1
Writer 0 wants to write to RECORD 3
Writer 0 is writing to record 3
Writer 1 wants to do intent write to the TABLE
Writer 2 wants to do intent write to the TABLE
Writer 3 wants to do intent write to the TABLE
Reader 0 wants to do intent read from the TABLE
Upgrader 0 wants to do intent write to the TABLE
Reader 1 wants to do intent read from the TABLE
Writer 4 wants to write to the TABLE
Reader 2 wants to do intent read from the TABLE
Reader 3 wants to read from the TABLE
Upgrader 1 wants to do intent write to the TABLE
Reader 4 wants to do intent read from the TABLE
Upgrader 2 wants to do intent write to the TABLE
Reader 5 wants to do intent read from the TABLE
Upgrader 3 wants to upgrade the TABLE
Reader 6 wants to read from the TABLE
Reader 7 wants to do intent read from the TABLE
Writer 5 wants to do intent write to the TABLE
Writer 6 wants to do intent write to the TABLE
Reader 8 wants to do intent read from the TABLE
Writer 0 is done writing to RECORD 3
Writer 0 is done intent write to the TABLE Count[0] = 0
Writer 1 is doing intent write to the table Count[1] = 1
Writer 1 wants to write to RECORD 1

Writer 1 is writing to record 1
Writer 1 is done writing to RECORD 1
Writer 1 is done intent write to the TABLE Count[1] = 0
Writer 3 is doing intent write to the table Count[2] = 1
Writer 3 wants to write to RECORD 2
Writer 3 is writing to record 2
Writer 3 is done writing to RECORD 2
Writer 3 is done intent write to the TABLE Count[2] = 0
upgrader 0 is doing intent write to the table Count[0] = 1
Upgrader 0 wants to upgrade RECORD 0
Upgrader 0 is reading from record 0 Count_0[0] = 1
Upgrader 0 is writing to record 0
Upgrader 0 is done upgrading RECORD 0
Upgrader 0 is done intent write to the TABLE Count[0] = 0
Writer 4 is writing to the table
Writer 4 is done writing to the TABLE
Reader 2 is doing intent read from the table Count[1] = 1
Reader 2 wants to read from RECORD 2
Reader 2 is reading from record 2 Count_2[0] = 1
Reader 3 is reading from the table Count[1] = 2
upgrader 1 is doing intent write to the table Count[1] = 3
Upgrader 1 wants to upgrade RECORD 2
Reader 4 is doing intent read from the table Count[1] = 4
Reader 4 wants to read from RECORD 0
Reader 4 is reading from record 0 Count_0[1] = 1
upgrader 2 is doing intent write to the table Count[1] = 5
Upgrader 2 wants to upgrade RECORD 0
Reader 5 is doing intent read from the table Count[1] = 6
Reader 5 wants to read from RECORD 2
Reader 6 is reading from the table Count[1] = 7
Reader 7 is doing intent read from the table Count[1] = 8
Reader 7 wants to read from RECORD 1
Reader 7 is reading from record 1 Count_1[0] = 1
Writer 5 is doing intent write to the table Count[1] = 9
Writer 5 wants to write to RECORD 4
Writer 5 is writing to record 4
Writer 6 is doing intent write to the table Count[1] = 10
Writer 6 wants to write to RECORD 4
Reader 8 is doing intent read from the table Count[1] = 11
Reader 8 wants to read from RECORD 2
Writer 2 is doing intent write to the table Count[1] = 12
Writer 2 wants to write to RECORD 2
Reader 0 is doing intent read from the table Count[1] = 13
Reader 0 wants to read from RECORD 1
Reader 1 is doing intent read from the table Count[1] = 14
Reader 1 wants to read from RECORD 0
Reader 4 is done reading from RECORD 0 Count_0[1] = 0
Reader 4 is done intent read from the TABLE Count[1] = 13
Reader 7 is done reading from RECORD 1 Count_1[0] = 0
Reader 7 is done intent read from the TABLE Count[1] = 12
Upgrader 2 is reading from record 0 Count_0[2] = 1
Reader 0 is reading from record 1 Count_1[1] = 1
Reader 2 is done reading from RECORD 2 Count_2[0] = 0
Reader 2 is done intent read from the TABLE Count[1] = 11
Upgrader 1 is reading from record 2 Count_2[1] = 1
Reader 5 is reading from record 2 Count_2[1] = 2
Reader 8 is reading from record 2 Count_2[1] = 3
Writer 5 is done writing to RECORD 4
Writer 5 is done intent write to the TABLE Count[1] = 10
Writer 6 is writing to record 4
Reader 3 is done reading from the TABLE Count[1] = 9
Reader 0 is done reading from RECORD 1 Count_1[1] = 0

Reader 0 is done intent read from the TABLE Count[1] = 8
Reader 5 is done reading from RECORD 2 Count_2[1] = 2
Reader 5 is done intent read from the TABLE Count[1] = 7
Reader 8 is done reading from RECORD 2 Count_2[1] = 1
Reader 8 is done intent read from the TABLE Count[1] = 6
Upgrader 1 is writing to record 2
Reader 6 is done reading from the TABLE Count[1] = 5
Upgrader 2 is writing to record 0
Writer 6 is done writing to RECORD 4
Writer 6 is done intent write to the TABLE Count[1] = 4
Upgrader 1 is done upgrading RECORD 2
Writer 2 is writing to record 2
Upgrader 1 is done intent write to the TABLE Count[1] = 3
Upgrader 2 is done upgrading RECORD 0
Reader 1 is reading from record 0 Count_0[0] = 1
Upgrader 2 is done intent write to the TABLE Count[1] = 2
Reader 1 is done reading from RECORD 0 Count_0[0] = 0
Reader 1 is done intent read from the TABLE Count[1] = 1
Writer 2 is done writing to RECORD 2
Writer 2 is done intent write to the TABLE Count[1] = 0
upgrader 3 is reading from the table Count[2] = 1
upgrader 3 is writing to the table
Upgrader 3 is done upgrading the TABLE

Time spent for algorithm_6 0 411 seconds

The average times spent in milliseconds to obtain a lock in algorithm_6

RW Average = 125 [0 30 271 61 181 211]

TW Average = 131 [131]

RU Average = 167 [81 211 211]

TR Average = 186 [191 181]

RR Average = 208 [221 261 191 191 211 181 201]

TU Average = 281 [281]

// Sample results of running turnaround version

A - Rebuild verbose version
B - Rebuild throughput version
C - Rebuild turnaround version
D - Run verbose version
E - Run throughput version
F - Run turnaround version
Q - Quit

Choose an option f

Please enter the number of requests (default = 15, 0 to terminate) 20
Please enter table reader access time 40
Please enter record reader access time 20
Please enter table writer access time 60
Please enter record writer access time 30
Please enter interval time 0

The average times spent in milliseconds to obtain a lock in algorithm_1

RR Average = 114 [120 120 120 110 110 110 110]

TR Average = 115 [120 110]

RW Average = 188 [0 30 60 90 461 491]

TW Average = 211 [211]

RU Average = 244 [160 261 311]

TU Average = 361 [361]

The average times spent in milliseconds to obtain a lock in algorithm_2

TW Average = 170 [170]

RW Average = 175 [0 30 60 90 421 451]

RU Average = 210 [120 230 280]

TU Average = 321 [321]

TR Average = 486 [491 481]

RR Average = 488 [491 491 491 491 491 481 481]

The average times spent in milliseconds to obtain a lock in algorithm_3

TW Average = 210 [210]

RW Average = 228 [0 30 60 90 581 611]

RU Average = 276 [140 310 380]

RR Average = 363 [120 190 270 360 421 541 641]

TR Average = 405 [270 541]

TU Average = 441 [441]

The average times spent in milliseconds to obtain a lock in algorithm_4

RR Average = 108 [110 110 110 110 110 110 100]

TR Average = 110 [110 110]

RW Average = 182 [0 30 50 80 451 481]

TW Average = 200 [200]

RU Average = 240 [150 260 310]

TU Average = 360 [360]

The average times spent in milliseconds to obtain a lock in algorithm_5

RW Average = 28 [0 30 30 60 10 40]

RR Average = 70 [60 80 90 80 70 40 70]

RU Average = 76 [30 110 90]

TW Average = 160 [160]

TU Average = 200 [200]

TR Average = 310 [320 300]

The average times spent in milliseconds to obtain a lock in algorithm_6

RW Average = 31 [0 30 30 60 20 50]

RU Average = 76 [30 110 90]

RR Average = 84 [60 80 90 151 80 50 80]

TW Average = 171 [171]

TR Average = 226 [231 221]

TU Average = 261 [261]

// Sample results of running turnaround version

```
A - Rebuild verbose version
B - Rebuild throughput version
C - Rebuild turnaround version
D - Run verbose version
E - Run throughput version
F - Run turnaround version
Q - Quit

Choose an option  e

Please enter the number of requests (default = 15, 0 to terminate)  20
Please enter table reader access time  40
Please enter record reader access time  20
Please enter table writer access time  60
Please enter record writer access time  30
Please enter interval time  0


Time spent for algorithm_1  0 561 seconds

Time spent for algorithm_2  0 541 seconds

Time spent for algorithm_3  0 681 seconds

Time spent for algorithm_4  0 541 seconds

Time spent for algorithm_5  0 4 seconds

Time spent for algorithm_6  0 37 seconds

Please enter the number of requests (default = 15, 0 to terminate)
```

APPENDIX C

USER INSTRUCTIONS

The following material is the user instructions for building and running the program.

User Instructions

Contents:

System Requirements:

This program is implemented in the Java programming language The software requires JDK 1 3 or JDK 1 4, which you may download from the following link

http //java sun com/j2se/downloads/index html

Since it is in Java, theoretically the program is cross-platform However, the execution of the program is automated for Windows only with a MS-DOS batch script So you will need to set up the CLASSPATH environment variable, build and execute the program manually if you are using a system other than Windows Note that steps for building the program and steps for running the program below are both for Windows

Steps for Running the Program:

Normally, you do not have to build the program before running it since all the necessary class files are already included in the package However, if Java complains that it is unable to find a class or any symbol is not defined when you run the program you will need to rebuild it Refer to the "Steps for Building the Program" section for instructions on how to build the program Use the following steps to run the program

1 Add "JAVA_HOME" to the system environment The user needs to do this only once To do this, right click on the "My Computer" icon on the desktop and select "Properties" Select the "Advanced" tab and click on the "Environment Variables" button Click on the "New .." button for "System variables" Enter "JAVA_HOME" in the "Variable" field and the top directory of the JDK or JSDK in the "Variable Value" field Click on the OK buttons to accept the selection

2 Download "Thesis_Project zip" and unzip it Note that the path to the folder "Thesis_Project" should not contain any space
3 In Windows Explorer, double click on the file "MyProjectRunner bat" in the "Thesis_Project" folder Alternatively, in a MS-DOS window, "cd " to the "Thesis_Project" folder and then run "MyProjectRunner bat" The batch script runs in the MS-DOS window, presenting a menu of options
4 If you want to run the verbose version of the program, type 'C' or 'c' after the prompt Here the word "verbose" means if you run this version of the program, it will spew to the screen other necessary information regarding the algorithms besides the time information This is very helpful for investigating the algorithms in terms of the execution order of the threads, deadlock and starvation But the costs for outputting these excessive texts can make the time comparison data inaccurate
5. If you want to build the terse version of the program, type 'D' or 'd' after the prompt Here the word "terse" means if you run this version of the program, it will put on the screen only the time information of algorithms Use this if you only want to compare the relative effectiveness of the algorithms
6 Enter the number of requests This corresponds to the total number of threads (writers, readers, upgraders, etc) The program will then pseudo-randomly generate the writers, readers, and/or upgraders They are generated pseudo-randomly so that we can compare the effectiveness of the algorithms
7 Enter the access time for the readers
8 Enter the access time for the writers

Then depends on which version of the program you chose to run, the program prints out on the screen the results of running the algorithms

Steps for Building the Program

You do not have to rebuild the program before running it, but if Java complains that it is unable to find a class or any symbol is not defined when you run the program you will need to rebuild it Refer to the "Steps for Running the Program" section for instructions on how to run the program Use the following steps to rebuild the program

1 Add "JAVA_HOME" to the system environment The user needs to do this only once To do this, right click on the "My Computer" icon on the desktop and select "Properties" Select the "Advanced" tab and click on the "Environment Variables" button Click on the

"New ." button for "System variables" Enter "JAVA_HOME" in the "Variable" field and the top directory of the JDK or JSDK in the "Variable Value" field Click on the OK buttons to accept the selections

2 Download "Thesis_Project zip" and unzip it Note that the path to the folder "Thesis_Project" should not contain any space

3 In Windows Explorer, double click on the file "MyProjectRunner bat" in the "Thesis_Project" folder Alternatively, in a MS-DOS window, CD to the "Thesis_Project" folder and then run "MyProjectRunner bat" The batch script runs in the MS-DOS window, presenting a menu of options

4 If you want to build the verbose version of the program, type 'A' or 'a' after the prompt Here the word "verbose" means if you run this version of the program, it will put on the screen other necessary information regarding the algorithms besides the time information This is very helpful for investigating the algorithms in terms of the execution order of the requests, deadlock and starvation But the costs for outputting these excessive texts can make the time comparison data inaccurate.

5 If you want to build the terse version of the program, type 'B' or 'b' after the prompt Here the word "terse" means if you run this version of the program, it will put on the screen only the time information of algorithms Use this if you only want to compare the relative effectiveness of the algorithms

The batch script will then delete all of the class files recursively, build the program in the chosen version and generate "clean" class files

REFERRENCE

[CoHP71]

P. Courtois, F. Heymans, and D. Parnas,

"Concurrent control with "readers"and "writers" ",
Communications of the ACM, Vol. 14, No. 10, Oct. 1971,

pp. 667-668.

[Dijk65]
E.W. Dijkstra
"Co-operating Sequential Processes," in *Programming Languages, Genuys, F, (Ed.)*, London: Academic Press,1965

[TaWo97]
Andrew S. Tanenbaum and Albert S. Woodhull ,

"Operating system", second edition. 1997-1998, pp. 66.

[EuSt82]
Eugene W. Stark
"Semaphore primitives and starvation-free mutual exclusion"
Journal of the ACM, Vol 29, No. 4, Oct. 1982
pp. 1049-1051

[Hart03]
Justin Hartman
"Fair Readers and Writers with Semaphore"
Homework assignment of EECS 338 class from Case Western Reserve University. 2003
http://vorlon.ces.cwru.edu/~jrh23/338/HW3.pdf
pp. 1 - 2

[IONA01]
IONA Technologies PLC 2001
"Chapter 7: Concurrency Control
Lock Modes
Read/Write Locking"
OrbixOTS programmer's and Administrator's Guide
http://www.iona.com/support/docs/manuals/orbix/33/pdf/orbixots33_pguide.pdf
pp. 105-107

[SiGG03]
Avi Siberschatz, Greg Gagne, Peter Galvin,
"Applied Operating Systems Concepts – Window XP Update",
first edition. 2003, pp. 184-198.

VITA

Mei Li was born in Lu Zhou, SiChuan, P. R. China, on March 5, 1968, the daughter of Yunqing Li and Renyu, Wan. After completing her work at First Middle School, Lu Zhou, SiChuan, P. R. China. She received the Bachelor of Art from SiChuan Foreign Language University in July 1989. During the following years she was employed as a Japanese translator with Lu Zhou Import and Export Company, P. R. China. In January 2001, she entered the Graduate College of Texas State University-San Marcos, Texas.

This thesis was typed by Mei Li.