**Part 1  Pseudo-code the new features**

**READER/WRITER CLASS:**

Extend class to Program and implement Serializable.

Declare a new database, reader Thread and writer Thread array.

Declare reader, writer iterations, record as integers.

**FOR:**

Instantiate new database

**FOR** iterations:

**For** readerArray:

Instantiate reader as new Thread.

Call start method for reader array.

Increment a count variable.

**IF** rand less than database array length

Set rand to zero

**IF** count is greater than record

Set rand as a next random to the database length.

**FOR** writer:

Set *value* as a random number

Instantiate writerArray as a new Thread

Call start for writerArray

Increment count variable

**IF** rand less than database array length

Set *rand* to zero

**IF** *count* is greater than *record*

Set rand to next random number of Database length.

Call Schedule method()

**IF** interrupted is true:

Call terminate method()

Return success.

**Terminate Method:**

**FOR** reader:

Call interrupt() on readerArray

Call join() on readerArray

Print message that the thread was interrupted.

**FOR** writer:

Call interrupt() on writerArray

Call join() on writerArray

Print message that the tread was interrupted.

Print message that the Reader/Writer class was interrupted.

**Client socket:**

Declare a string *line*

Instantiate New Socket with ip address 127.0.0.1 and port 6013

Instantiate new Streams and PrintWriter

Instantiate BufferedReader.

Call readLine() method

Close all the streams

Return the string.

**System Call Class:**

Declare ServerSocket, String array and Socket types.

Close(){

Call the close method for the socket

}

SystemCall(){

Instantiate new ServerSocket

Instantiate new String array set to the fileSize.

}

Run(){

**WHILE** true:

Call the accept() method for socket

Instantiate new BufferedReader, PrintWriter

Call readLine() method.

Call split mehthod to split a space character.

EX. We want to break 4 123.23

**SWITCH**:

**CASE:** “size”

Print database length.

**CASE:** “write”

Print database record 2 to record 1.

**CASE:** “read”

Print database string at record 1.

**DEFAULT:**

Print error message on client side.

close all streams.

}

**Part 2  Answer the following questions**

1. **Is deadlock possible in your application? How?**

Yes, it is possible because we have to fix it in lab 6. It is possible, because the client and server will be trying to talk back and forth with each other at the same time.

1. **How did you make sure that all of the application threads were terminated?**

We tested if the thread was interrupted first. If it was, we called our terminate method, which would cycle through and close the remaining applications.

1. **What would happen if all of the application threads were not terminated?**

If the read and write threads are not terminated, then they will keep running in the processes until its fully done, but it won’t close on its own, causing deadlock.

1. **What socket classes and methods did you use?**

We used Socket, OutputStream and PrintWriter for classes, and we used getOuputStream() and getInputStream() and the close() methods.

**5.  How did your reader and writer threads get access to the Program method to initiate system calls?**

We made the system method static.