**Topics**

Multithreading

Synchronization

Deadlocks

**Description**

Create a GUI program that provides for merging two files: FileA and FileB. Access to each of the two files be synchronized by a different object monitor. For example, access to FileA will be synchronized using objectA (an instance of Object) monitor and accesss to FileB will be synchronized by objectB (an instance of Object) monitor. The merging will be done in two different ways by two different threads. One thread will merge FileA and FileB into a file FileAB. The second thread will merge FileB and FileA into file FileBA. The first thread will obtain access to FileA using objectA monitor, sleep for 100ms (artificially simulating time slice), and then obtain access to FileB using objectB monitor. Then it will merge the two files into FileAB containing first line from FileA, second line from FileB etc. The second thread will obtain access to FileB using objectB monitor, sleep for 100ms (artificially simulating time slice), and then obtain access to FileA using objectA monitor. Then it will merge the two files into FileBA containing first line from FileB, second line from FileA etc.

The  program will run in two modes:

Merge with a deadlock. (Merge Deadlock)

Merge without a deadlock. (Merge Deadlock Free)

**Pseudo-code Merge Deadlock Mode**

In Merge Deadlock mode, there will be separate monitors, objectA and objectB for getting access to FileA and FileB respectively. (There will be no monitor for gaining access to both the files together).  Pseudo-code for this mode is given below:

Pseudo-code First thread:

(Obtain objectA monitor; obtain objectB monitor; merge the two files)

synchronized (objectA) {

        Thread.sleep(100);

        synchronized (objectB) {

                Thread.sleep(100);

                merge FileA and FileB into FileAB

        }

}

Pseudo-code Second Thread:

(Obtain objectB monitor; obtain objectA monitor; merge the two files)

synchronized (objectB) {

        Thread.sleep(100);

        synchronized (objectA) {

                Thread.sleep(100);

                merge FileB and FileA into FileAB

        }

}

Deadlock

The above pseudo-code will cause a deadlock. The first thread will run and get ObjectA monitor, then sleep for 100 ms (i.e. simulated time slice). In the meantime, the second thread will run and get ObjectB monitor, then sleep for 100 ms (i.e. simulated time slice). Then the first thread will end its sleep and run and will attempt to get ObjectB monitor but will not be able to get it as it is already taken by the second thread. It will be put in the wait queue of ObjectB. Then the second thread will end its sleep and run and will attempt to get ObjectA monitor but will not be able to get it as it is already taken by the first thread. It will be put in the wait queue of ObjectA. The two threads will deadlock waiting for the other to finish. Both will remain in wait state indefinitely.

**Pseudo-code Merge Deadlock Free Mode**

In Merge Deadlock Free mode, there will be separate monitors, objectA and objectB for getting access to FileA and FileB respectively as in the previous mode. However, there will also be an additional monitor, objectAB, for gaining access to both the files together. This objectAB monitor will be used by both the threads for gaining access to the two files together regardless of whether a thread wants to gains access to FileA first and then to FileB or vice versa.

Pseudo-code for this mode is given below:

Pseudo-code First thread:

(Obtain objectAB monitor; obtain objectA monitor; obtain objectB monitor; merge the two files)

synchronized (objectAB) {

        Thread.sleep (100);

synchronized (objectA) {

        Thread.sleep(100);

        synchronized (objectB) {

                Thread.sleep(100);

                merge FileA and FileB into FileAB

        }

}

}

Pseudo-code Second Thread:

(Obtain objectAB monitor; obtain objectB monitor; obtain objectA monitor; merge the two files)

synchronized (objectAB) {

synchronized (objectB) {

        Thread.sleep(100);

        synchronized (objectA) {

                Thread.sleep(100);

                merge FileB and FileA into FileAB

        }

}

}

Deadlock Free:

Each thread attempts to get objectAB monitor. The one, who gets it first, gains accesses to both the files and performs merging. In the meantime, the other thread runs and tries to get objectAB monitor but it�s taken by the first thread. So it waits in objectAB queue for objectAB monitor to become available. Only after the first thread is finished merging the two files, that the objectAB monitor becomes available. The second thread gains access to objectAB monitor and the two files and performs merging. Thus the two threads take turn in obtaining objectAB to perform merging. This avoids the deadlock.

**Deadlock** **Implementation**

**Class MergeRunnable**

Create the Class MergeRunnable that will implement the Runnable interface and provide for the following:

file1 - a String instance variable to keep the name of the first file

file2 � a String instance variable to keep the name of the second file

filem - a  String instance variable to keep the name the merged file

obj1 - an Object reference variable to synchronize file1

obj2 - an Object reference variable to synchronize file2

It will provide a five parameter constructor to initialize the instance variables.

It will have a method run which will make use of the instance variables and implement the following algorithm.

synchronized (obj1) {

        Thread.sleep(100);

        synchronized (obj2) {

                Thread.sleep(100);

                merge File1 and File2 into File12

        }

}

**Deadlock Free Implementation**

**Class MergeRunnable2**

Create the Class MergeRunnable2 that will implement the Runnable interface and provide for the following:

file1 - a String instance variable to keep the name of the first file

file2 � a String instance variable to keep the name of the second file

filem - a  String instance variable to keep the name the merged file

obj1 - an Object reference variable to synchronize file1 individually

obj2 - an Object reference variable to synchronize file2 individually

obj12 - an Object reference variable to synchronize file1 and file2 jointly

It will provide a six parameter constructor to initialize the instance variables.

It will have a method run which will make use of the instance variables and implement the following algorithm.

synchronized (obj12) {

        Thread.sleep(100);

synchronized (obj1) {

        Thread.sleep(100);

        synchronized (obj2) {

                Thread.sleep(100);

                merge File1 and File2 into File12

        }

}

}

**GUI Implementation**

The GUI will contain three panels:

The North panel will contain the following:

A labeled text field for the FileA name.

A labeled text field for the FileB name.

The Center panel will contain scrollable text area for displaying debug messages.

The South panel will contain the following:

A button labeled �Merge (Deadlock)� for running the two threads that will cause deadlock.

A button labeled �Merger (Deadlock Free)� for running the two threads: each will merge the two files without causing a deadlock.

**Class JFrameExt**

Create a class JFrameExt that will contain the following:

A constructor JFrameExt that will create the Graphical User Interface (GUI).

A handler that will be invoked when the user clicks the �Merge (Deadlock)� button:

A handler that will be invoked when the user clicks the �Merge (Deadlock Free)� button.

**Handler Merge Deadlock:**

It will create the following:

fileA � a String variable that will contain the name of the file obtained from first text field

fileB � a String variable that will contain the name of the file obtained from second text field.

fileAB � a String variable that will contain the name of the merged file as �fileAB.txt� (hard-coded).

fileBA -  a String variable that will contain the name of the merged file as �fileBA.txt� (hard-coded).

objA � An object of class Object

objB � An object of class Object

It will create an instance of MergeRunnable and pass its constructor fileA, fileB, fileAB, objA, objB respectively.

It will create a second instance of MergeRunnable and pass its constructor fileB, fileA, fileBA, objB, objA respectively.

It will create two threads: the first one will run the first instance of MergeRunnable above and the second one will run the second instance of MergeRunnable above

**Handler Merge Deadlock Free:**

It will do the following:

fileA � a String variable that will contain the name of the file obtained from first text field

fileB � a String variable that will contain the name of the file obtained from second text field.

fileAB � a String variable that will contain the name of the merged file as �fileAB.txt� (hard-coded).

fileBA -  a String variable that will contain the name of the merged file as �fileBA.txt� (hard-coded).

objA � An object of class Object

objB � An object of class Object

objAB � An object of class Object

It will create an instance of MergeRunnable2 and pass its constructor fileA, fileB, fileAB, objA, objB, objAB respectively.

It will create a second instance of MergeRunnable2 and pass its constructor fileB, fileA, fileBA, objB, objA, objAB respectively.

It will create two threads: the first one will run the first instance of MergeRunnable2 above and the second one will run the second instance of MergeRunnable2 above

**Testing**

**Test Preparation**

Prepare two text only files, fileA.txt and fileB.txt, using an editor. Save these files in the project folder.

The fileA.txt will contain:

Hi1

Hi2

Hi3

The fileB.txt will contain

Bye1

Bye2

Bye3

**Test Run 1**

Enter in text fields: FileA name as fileA.txt and FileB name as fileB.txt.

Click the button Merge (Deadlock).

Check

fileAB.txt and fileBA.txt will not be created because both threads are deadlocked.

**Test Run 2**

Enter in text fields: Enter FileA name as fileA.txt and FileB name as fileB.txt.

Click the button Merge (Deadlock Free).

Check

fileAB.txt and fileBA.txt will be created and their contents will be result of merging the two files.

**Turn In**

source code.

two screen shots: one for each run.

**GUI**

