Operating system 2 Project

Cover sheet & Documentation

Project Title : Readers and Writers Problem

Group#.....

Discussion time :- 10:40 ..... Instructor : Eng Abdelrahman

ID	Name(Arabic)	Bounce	Minus	Total Grade	Comment
201900179	أمير حنا ثابت فهيم				
201900199	أياد ايمن محمد مصيلحي				
201900885	مينا وديع صدقي عوض				
202000171	آيات ايمن حافظ الشال				
202000510	عبدالرحمن حمدي عبدالعاطي عبدالحميد				
202000542	عبدالرحمن ياسر سمير محمد				
202000548	عبدالله شعبان عبدالرازق سيد				
202000776	محمد عادل محمد حسني محمد				

Project Documentation

Solution Pseudocode :

Class ReaderWriterVariables:

Declare Static readerCount variable of type integer

Initilaize the readerCount variable to 0

Declare Static sharedData variable of type String

Initilaize sharedData variable to "ReaderWriterDefaultValue"

Declare Object mutexSemaphore, readSemaphore and writeSemaphore fro Semaphore Class

main:

Declare Object from Read Class

Declare Object from Write Class

Create 4 Threads

2 Threads for Read

2 Threads for Write

then start the threads

Create Class Read implements Runnable:

override

run:

acquire readSemaphore

acquire mutexSemaphore

increments readerCount by 1

if readerCount = 1 :

acquire writeSemaphore

release mutexSemaphore

display thread name

sleep for secnds

acquire mutexSemaphore

Decrements readerCount by 1

if readerCount = 0:

release writeSemaphore

release mutexSemaphore

release readSemaphore

Create Write Class implements Runnable:

override

run Method:

acquire readSemaphore

acquire writeSemaphore

display thread name is Writing and finished

Sleep for seconds

release writeSemaphore

release readSemaphore

The Deadlock :

Examples :

Frist I would like to know what it is Deadlock :

If two threads are waiting for each other to complete their execution then such a type of infinite waiting situation is called deadlock.

Let us understand it through an example:-

Assume there are two threads. The first thread is waiting for the completion of the second thread, once the second thread execution is completed then the first thread will continue its execution. Similarly, the second thread is waiting for the completion of the first thread, once the first thread execution is completed then the second thread will continue its execution. Both threads are waiting for each other an infinite number of times.

Let us understand the deadlock through a real-life example:-

You and your friend want to go to the movie on Saturday evening, you both bought tickets a few days ago. There are two roads to go to the movie hall either passing through your house or passing through your friend's house, the overall distance is approximately the same.

On Saturday evening, you are waiting for your friend at your home so that when he/she comes then both will go together. Similarly, your friend is waiting for you at his/her own home so that when you come then both will go together. Due to some reason neither you want to call/message your friend nor your friend wants. Both of you also don't want to go alone, just waiting for each other. This situation is nothing but a deadlock situation where two threads are waiting for each other forever.

Deadlock Example in Java

What is the keyword that causes deadlock? Synchronized keyword. Synchronization is the only reason for a deadlock situation, hence while using the synchronized keyword we must be careful because the wrong implementation could lead to the deadlock situation.
In Java, If we don't use the synchronized keyword then we never encounter the deadlock situation.

Program to demonstrate deadlock in Java using the synchronized method,

```
public class MyThread extends Thread {

    public static A a = new A();
    public static B b = new B();

    public static void main(String[] args) {
        System.out.println("Main thread Started.");
        MyThread t1 = new MyThread();
        t1.start();
        a.m1(b);
        System.out.println("Main thread Ended.");
    }

    public void run() {
        System.out.println("Thread-0 Started.");
        b.m2(a);
        System.out.println("Thread-0 Ended.");
    }
}

class A {
    public synchronized void m1(B b) {
        try {
            Thread.sleep(2000); // 2 sec
        } catch (InterruptedException ie) {}
        System.out.println("Calling display()");
        b.display();
    }
    public synchronized void show() {
        System.out.println("show() method");
    }
}

class B {
    public synchronized void m2(A a) {
        try {
            Thread.sleep(2000); // 2 sec
        } catch (InterruptedException ie) {}
        System.out.println("Calling show()");
        a.show();
    }
    public synchronized void display() {
        System.out.println("display() method");
    }
}
```

Output:-

```
Main thread Started.
Thread-0 Started.
Calling display()
Calling show()
<program-stuck>
```

Solution of Deadlock in our project

How did we solve deadlock in our program …….?

We use two semaphore

Before we read from data or write in it

we call Aquire() function

then release

The Starvation :

Examples :

Frist I would like to know what it is Starvation :

Starvation describes a situation where a thread is unable to gain regular access to shared resources and is unable to make progress.
This happens when shared resources are made unavailable for long periods by "greedy" threads.

For example :

suppose an object provides a synchronized method that often takes a long time to return. If one thread invokes this method frequently, other threads that also need frequent synchronized access to the same object will often be blocked.

Example :-

When two threads are having different priority then the thread having higher priority will get the first chance to execute. Low priority thread has to wait until completing all high priority threads. It may have to wait for a long time period for its execution but waiting will end at a certain point. This situation is an example of starvation.

Let us understand starvation in Java through an example:-

Assume there are 10,000 threads running. Among them, one thread is having priority=1 (lowest priority), and the remaining all threads are having a priority greater than 1 (priority > 1). In this case, the thread having priority=1 has to wait till the completion of all remaining threads having high priority.

Another Example of Starvation in Java through the synchronized block,

```
class MyThread extends Thread {
    public void run() {
        String threadName = Thread.currentThread().getName();
        System.out.println(threadName + " Started");
        synchronized(MyThread.class) { // lock
            // doing some useful work
            try {
                Thread.sleep(2000); // 2 sec
            } catch (InterruptedException ie){}
        }
        System.out.println(threadName + " End");
    }
}

public class Test {
    public static void main(String[] args) {
        System.out.println("Start of Main thread");
        MyThread mt[] = new MyThread[10];
         for (int i=0; i<mt.length; i++) {
            mt[i] = new MyThread(); // create thread
            mt[i].start();
         }
         System.out.println("End of Main thread");
    }
}
```

Output is :-

```
Start of Main thread
End of Main thread
Thread-9 Started
Thread-8 Started
Thread-0 Started
Thread-6 Started
Thread-7 Started
Thread-4 Started
Thread-2 Started
Thread-5 Started
Thread-3 Started
Thread-1 Started
Thread-9 End
Thread-1 End
Thread-7 End
Thread-3 End
Thread-4 End
Thread-5 End
Thread-2 End
Thread-6 End
Thread-0 End
Thread-8 End
```

In this example the main thread created 10 child threads. To execute some portion of the run() method (synchronized block) each child thread needs the lock of the current class. At a time only one thread can get the lock of one object. And to complete execution, each thread required more than 2 seconds time.

Among these 10 threads, there will be some threads executing at last. They were waiting for a long period of time because the thread was unable to gain regular access to shared resources (lock of current class) and was unable to make progress. In the above output, thread-8 waited for a long period of time

How did we solve the starvation…….?

We used two semaphore

And we use it by Queue (frist in frist out)

Explanation for our real world application and how did we apply the problem….!?

We create a warehouse(store) app which control the adding , removing or reading the items in the store

You can check which item is there in it by Read() function

When you can add more goods in store or remove other by write() function