

synchronization simulator

Student Name: Haya Tahboub.

Student No: 211072.

Overview:

After the problem and take requirements (make synchronization simulator that two or more processes changes on shared variable(s) by adding or subtracting values from these variables from n processes or threads and show the result with and without synchronization).

- The programming language used: Java.
- Basic Classes: Consumer, Producer, Data, main classes.
- The technique used: Threads.
- Types of Threads: Consumer who Adds, Producer who consumes.
- Note: The code was written jointly between my colleague lujain abu rajab and me but the results are separate.

Classes:

Data class:

Contain shared variable between threads or presses in this case we have X, as shared variable that acts as buffer and the initial value is 1000.

```
package os;

/**
 *
 * @author user
 */
public class Data {
    public static int x=1000;
}
```

Consumer class:

In this class, we create threads work as consumer that consume the value of the data and we can change the work by change the implementation of the run method.

```
    /**
    package os;

    /**
    *
    * @author user
    */
    public class consumer extends Thread {
        public void run() {
            Data.x --;
        }
    }
```

Producer class:

In this class as the consumer class, we create threads but work as producer that produce the value of the data and we can change the work by change the implementation of the run method.

```
    /**
    package os;

    /**
    *
    * @author user
    */
    public class produser extends Thread{
        public void run() {
            Data.x ++;
        }
    }
```

Before of Synchronization:

In this main class, we can run threads without synchronization and each thread in core by create random threads sub of them are consumer and others are producer and each thread can produce or consume random items and they can do this concurrently but in this case, there are difference between real result and what they consumed and produced.

However, in my simulation the threads work sequentially because they run in one core not multicore so there is no difference between real result and what the thread consumed and produced.

```
public static void main(String[] args) {
    // TODO code application logic here

    Random random = new Random();
    int ConCounter = 0;
    int ProCounter = 0;
    int randomNumThPro = random.nextInt(10)+1;
    int randomNumThCun = random.nextInt(15)+1;

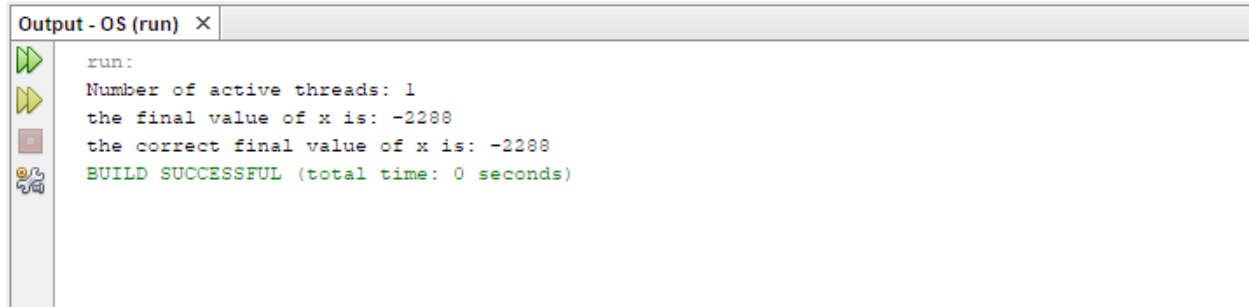
    Random random1 = new Random();
    int randomNumPro = random1.nextInt(900)+100;
    int randomNumCun = random1.nextInt(901) + 100;

    for(int i=0;i<randomNumThCun;i++){
        consumer MyConsumer = new consumer();
        for(int j=1;j<randomNumCun;j++){
            MyConsumer.run();
            ConCounter++;
        }
    }

    for(int i=0;i<randomNumThPro;i++){
        produser MyProducer = new produser();
        for(int j=1;j<randomNumPro;j++){
            MyProducer.run();
            ProCounter++;
        }
    }

    int activeThreadCount = Thread.activeCount();
    System.out.println("Number of active threads: " + activeThreadCount);
    System.out.println("the final value of x is: "+Data.x);
    System.out.println("the correct final value of x is: "+(1000+ProCounter-ConCounter));
}
```

Result:

A screenshot of an IDE's output window titled "Output - OS (run) X". The window contains the following text: "run:", "Number of active threads: 1", "the final value of x is: -2288", "the correct final value of x is: -2288", and "BUILD SUCCESSFUL (total time: 0 seconds)". On the left side of the window, there are four icons: a green play button, a yellow play button, a red stop button, and a blue bug icon.

```
run:
Number of active threads: 1
the final value of x is: -2288
the correct final value of x is: -2288
BUILD SUCCESSFUL (total time: 0 seconds)
```

Number of active threads means that all threads execute in the same core.

After of Synchronization:

In this main class, we can run threads with synchronization and each thread in core by create random threads sub of them are consumer and others are producer and each thread can produce or consume random items and they can do this concurrently. but in this case, we have a synchronization method call semaphore lock that controls the entry and exit of each Thread separately.

Using semaphore.acquire ()

semaphore.release()

And we have used full/empty method that solve bounded buffer problem assuming the shared variable as buffer then producer can produce 1800 item maximum and consumer can consume 0 item maximum and in java we can implement this method using semaphore.

However, because in my simulation threads works sequential so if producer start then when they reach 1800 should wait consumer for new empty place but consumer will never execute so producer enter infinite loop and the same scenario if consumer started but consumer should wait producer for full place.

```

public static void main(String[] args) {
    // TODO code application logic here

    int ConCounter = 0;
    int ProCounter = 0;
    Random random = new Random();
    int randomNumThPro = random.nextInt(10)+1;
    int randomNumThCun = random.nextInt(15)+1;

    Random random1 = new Random();
    int randomNumPro = random1.nextInt(900)+100;
    int randomNumCun = random1.nextInt(900) + 100;

    Semaphore semaphore = new Semaphore(15);

    int BUFFER_SIZE = 800;
    Semaphore full = new Semaphore(1000);
    Semaphore empty = new Semaphore(BUFFER_SIZE);
    int[] buffer = new int[BUFFER_SIZE];

```

```

for(int i=0;i<randomNumThPro;i++){
    producer MyProducer = new producer();
    for(int j=1;j<randomNumPro;j++){
        try {
            if(ProCounter==800){
                System.out.println("the value of buffer is reach to 1799 and make infinte loop");
            }

            empty.acquire();
            semaphore.acquire();
        } catch (InterruptedException ex) {
            Logger.getLogger(OSWithSync.class.getName()).log(Level.SEVERE, null, ex);
        }
        MyProducer.run();
        ProCounter++;
        int item = i;
        buffer[i % BUFFER_SIZE] = item;
        semaphore.release();
        full.release();
    }
}

```

```

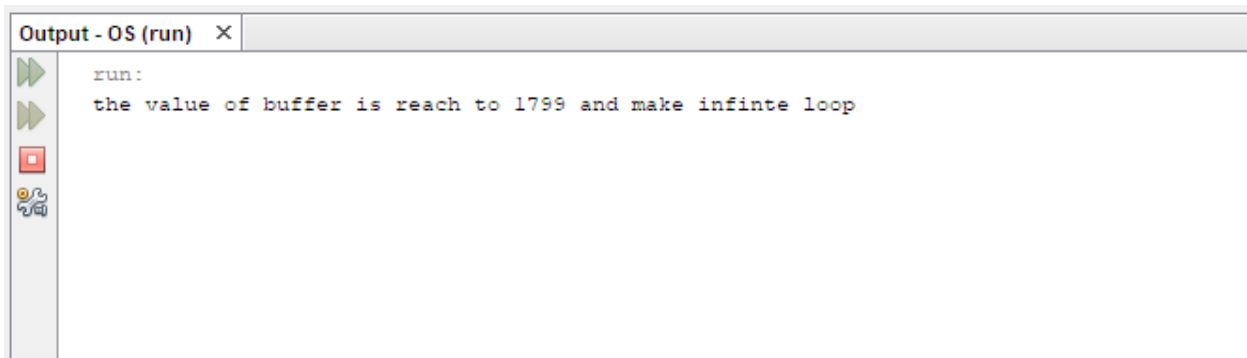
for(int i = 0; i < randomNumThCun ; i++){
    consumer MyConsumer = new consumer();
    for(int j=1;j<randomNumCun;j++){
        try {
            if(ConCounter==1000){
                System.out.println("the value of buffer is reach to 0 and make infinte loop");
            }
            full.acquire();
            semaphore.acquire();
        } catch (InterruptedException ex) {
            Logger.getLogger(OSWithSync.class.getName()).log(Level.SEVERE, null, ex);
        }
        MyConsumer.run();
        int item = buffer[i % BUFFER_SIZE];
        ConCounter++;
        semaphore.release();
        empty.release();
    }
}

System.out.println("the final value of x with sync is: "+Data.x);
}
}

```

Result:

In case, we start with producer:



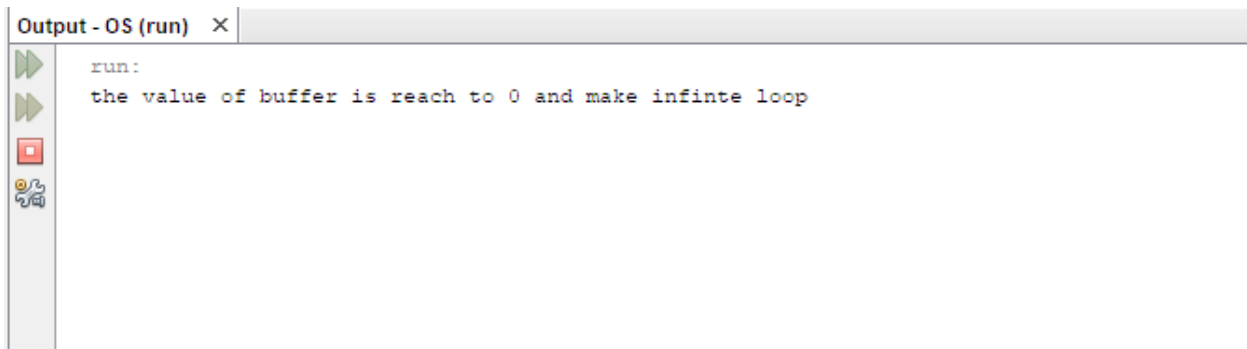
Output - OS (run) X

```

run:
the value of buffer is reach to 1799 and make infinte loop

```

In case, we start with consumer:



Output - OS (run) X

```

run:
the value of buffer is reach to 0 and make infinte loop

```

