
**KINGDOM OF SAUDI ARABIA**

**Ministry of Education**

**Taibah University**

**College of Computer Science & Engineering**


---

**CS424 – Introduction to Parallel Computing**

**Semester II 2019/2020**

---

# ASSIGNMENT


**Submit By:**

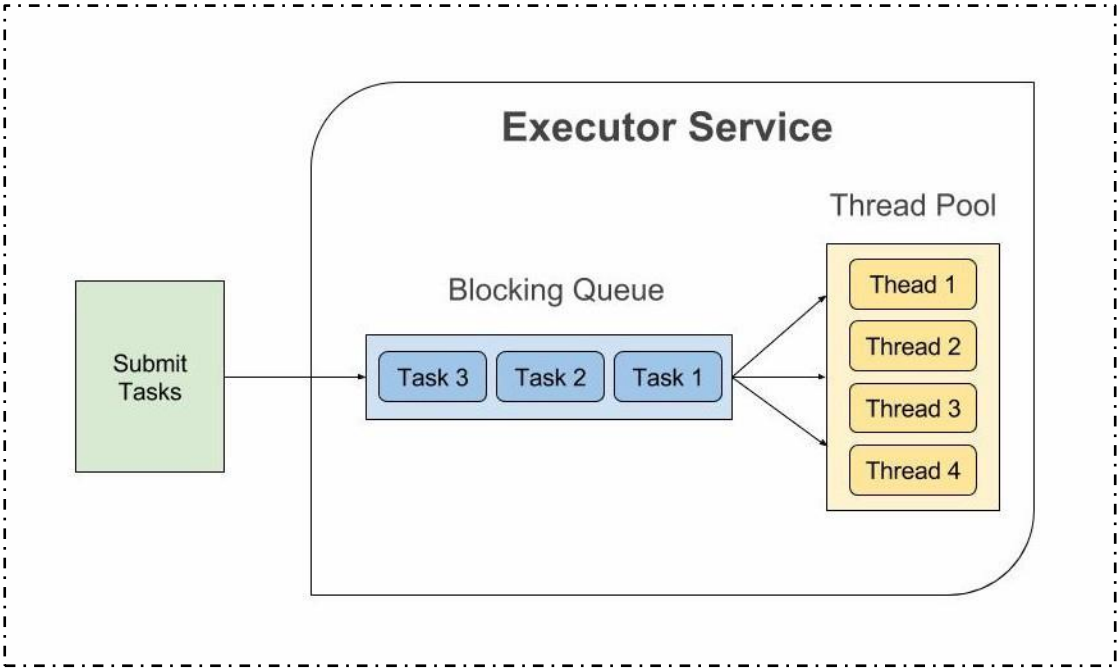
**Basem Awwad Al-Moziny**
**No. 3706683**

A. WRITE A PROGRAM THAT LAUNCHES 1,000 THREADS. EACH THREAD ADDS 1 TO A VARIABLE SUM THAT INITIALLY IS 0. DEFINE A VARIABLE TO HOLD SUM.

1. DESCRIPTION OF THE PROBLEM AND THE SEQUENTIAL SOLUTION

```
2
3
4   int sum = 0;
5       for( int i = sum; i < 1000; i++ )
6       {
7           sum+=i;
8          System.out.print(" " + i);
9
10
```

2. PARALLEL ALGORITHM DESIGN



3. PARALLEL CODE

i. RUN THE PROGRAM WITHOUT SYNCHRONIZATION

```
package javaapplication36;

import java.util.concurrent.*;

public class JavaApplication36 {

private static Addation add = new Addation();

  public static void main(String[] args) {
    ExecutorService executor = Executors.newCachedThreadPool();

    // Create and launch 1000 threads
    for (int i = 0; i < 1000; i++) {
      executor.execute(new AddAPennyTask());
    }

    executor.shutdown();

    // Wait until all tasks are finished
    while (!executor.isTerminated()) {
    }

    System.out.println("SUM = " + add.getTotal());
  }

  // A thread for adding a penny to the account
  private static class AddAPennyTask implements Runnable {
    public void run() {
      add.opadd(1);
    }
  }

  // An inner class for account
  private static class Addation {
    private int SUM = 0;

    public int getTotal() {
      return SUM;
    }

    public void opadd(int Value) {
      int newValue = SUM + Value;

      // This delay is deliberately added to magnify the
      // data-corruption problem and make it easy to see.
      try {
        Thread.sleep(5);
      }
      catch (InterruptedException ex) {
      }

      SUM = newValue;
    }
  }
}
```

```
run:
SUM = 11
BUILD SUCCESSFUL (total time: 0 seconds)
```

ii. RUN THE PROGRAM WITH SYNCHRONIZATION IN THREE DIFFERENT TECHNIQUES

1st TECHNIQUE:

```
public class JavaApplication36 {


private static Addation add = new Addation();

  public static void main(String[] args) {
    ExecutorService executor = Executors.newCachedThreadPool();

    // Create and launch 1000 threads
    for (int i = 0; i < 1000; i++) {
      executor.execute(new AddAPennyTask());
    }

    executor.shutdown();

    // Wait until all tasks are finished
    while (!executor.isTerminated()) {
    }

    System.out.println("SUM = " + add.getTotal());
  }

  // A thread for adding a penny to the account
  private static class AddAPennyTask implements Runnable {
    public void run() {
      add.opadd(1);
    }
  }

  // An inner class for account
  private static class Addation {
    private int SUM = 0;

    public int getTotal() {
      return SUM;
    }

    public synchronized void opadd(int Value) {
      int newValue = SUM + Value;

      // This delay is deliberately added to magnify the
      // data-corruption problem and make it easy to see.
      try {
        Thread.sleep(5);
      }
      catch (InterruptedException ex) {
      }

      SUM = newValue;
    }
  }
}
```

```
run:
SUM = 1000
BUILD SUCCESSFUL (total time: 6 seconds)
```

2nd TECHNIQUE:

```java
package javaapplication36;

import java.util.concurrent.*;


public class JavaApplication36 {

private static Addation add = new Addation();

  public static void main(String[] args) {
    ExecutorService executor = Executors.newCachedThreadPool();

    for (int i = 0; i < 1000; i++) {
      executor.execute(new AddAPennyTask());
    }

    executor.shutdown();

    while (!executor.isTerminated()) {
    }

    System.out.println("SUM = " + add.getTotal());
  }

  private static class AddAPennyTask implements Runnable {
    public void run() {
      add.opadd(1);
    }
  }

  private static class Addation {
    private int SUM = 0;

    public int getTotal() {
      return SUM;
    }

    public void opadd(int Value) {
      synchronized (add){
        int newValue = SUM + Value;
.
      try {
        Thread.sleep(5);
      }
      catch (InterruptedException ex) {
      }

      SUM = newValue;
      }
    }
  }
}
```

```
run:
SUM = 1000
BUILD SUCCESSFUL (total time: 6 seconds)
```

3rd TECHNIQUE:

```java
package javaapplication36;

import java.util.concurrent.*;
import java.util.concurrent.locks.*;

public class JavaApplication36 {

private static Addation add = new Addation();

  public static void main(String[] args) {
    ExecutorService executor = Executors.newCachedThreadPool();

    for (int i = 0; i < 1000; i++) {
      executor.execute(new AddAPennyTask());
    }

    executor.shutdown();

    // Wait until all tasks are finished
    while (!executor.isTerminated()) {
    }

    System.out.println("SUM = " + add.getTotal());
  }

  public static class AddAPennyTask implements Runnable {
    public void run() {
      add.opadd(1);
    }
  }
  public static class Addation {
    private static Lock lock = new ReentrantLock(); // Create a lock
    private int SUM = 0;
    public int getTotal() {
      return SUM;
    }

    public void opadd(int Value) {
      lock.lock(); // Acquire the lock

      try {
        int newValue = SUM + Value;

        Thread.sleep(5);

        SUM = newValue;
      }
      catch (InterruptedException ex) {
      }
      finally {
        lock.unlock(); // Release the lock
      }
    }
  }
```

```
run:
SUM = 1000
BUILD SUCCESSFUL (total time: 6 seconds)
```

B. WRITE A PROGRAM TO COMPUTE A FACTORIAL OF A GIVEN NUMBER IN PARALLEL

```
2
3  #ifdef _OPENMP
4
5  #include <omp.h>
6
7    #else
8
9  #define omp_get_thread_num() 0
10 #endif
11
12 #include <stdio.h>
13
14 int main(void){
15     int i,n=100;
16     int fac=1;
17 #pragma omp parallel for shared(n) private(i) reduction(*:fac)
18     for(i=1;i<=n;i++){
19         fac*=i;
20     }
21     printf("Thread%d - fac(%d)=%d\n",omp_get_thread_num(),n,fac);
22     return 0;
23 }
```