МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ
КАФЕДРА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

ЛАБОРАТОРНА РОБОТА №6

з дисципліни «Паралельні та розподілені обчислення
Паралельне програмування-2»

Тема: «Java. Монітори»

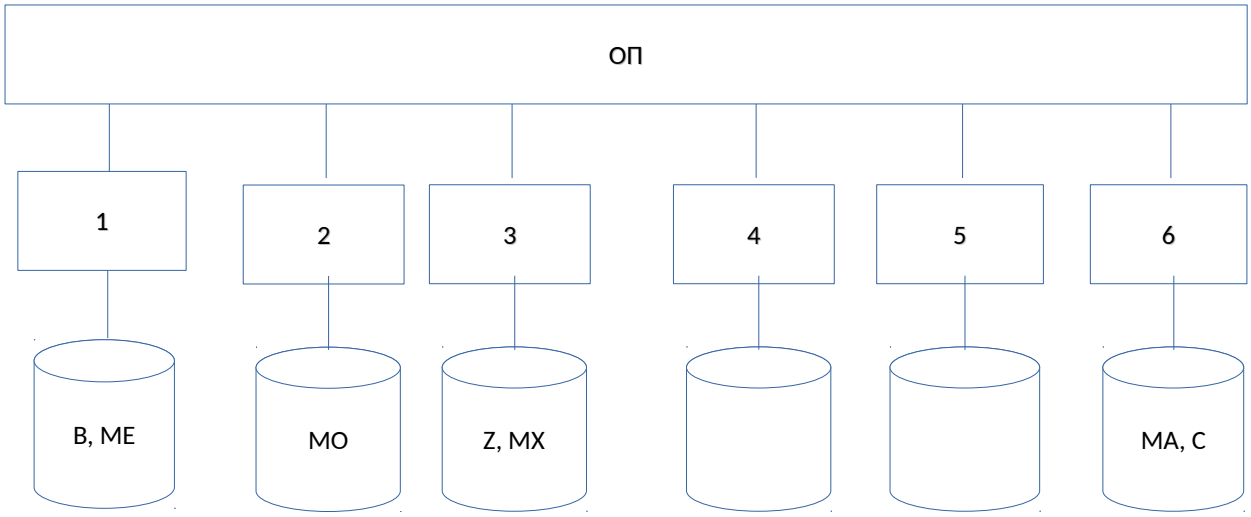
Виконав:
студент 3-го курсу
групи ІП-42
з номер заліковки 4206
Дзюба Влад

Київ – 2016

Завдання

Мета роботи: розробка програми для ПКС зі СП
Мова програмування: Java
Засоби організації взаємодії процесів: монітори мови Java, синхронізовані блоки, Fork-Join.

Варіант



$$MA = min(Z) \cdot MO + (B \cdot C)(MX \cdot ME)$$

Математичний паралельний алгоритм:

1. $z_i = min(Z_h), i \in [0..P-1]$
2. $z = min(z, z_i), i \in [0..P-1]$
3. $d_i = B_h \cdot C_h, i \in [0..P-1]$
4. $d = d + d_i, i \in [0..P-1]$
5. $MA_h = z \cdot MO_h + d \cdot (MX \cdot ME_h)$

CP: z, d, MX.

Алгоритм для кожного процесу:

TPi — означення ThreadPool в алгоритмі.

T_1:		T_2:	
1. Ввести B.		1. Ввести MO.	
2. Сигнал про завершення вводу B.	S_1	2. Сигнал про завершення вводу MO.	S_1
3. Ввести ME.		3. Чекати сигнал про завершення обрахунку z та d з T_1 .	$W_{1,1}$
4. Сигнал про завершення вводу ME.	S_2	4. Чекати сигнал про завершення вводу ME з T_1 .	$W_{1,2}$
5. Чекати сигнал про завершення вводу Z з T_3 .	$W_{3,1}$	5. Чекати сигнал про завершення вводу MX з T_3 .	$W_{3,3}$
6. Обрахувати $z=min(Z_h)$	TP_1	6. Копіювати $z_2=z$	$КУ\,1$
7. Чекати сигнал про завершення вводу C з T_6 .	$W_{6,2}$	7. Копіювати $d_2=d$	$КУ\,2$
8. Обрахувати $d=B\cdot C$	TP_2	8. Копіювати $MX_2=MX$	$КУ\,3$
9. Сигнал про завершення обрахунку z та d до всіх потоків.	S_3	9. Обрахувати $MA_h=z_2\cdot MO_h+d_2\cdot(MX_2\cdot ME_h)$.	
10. Чекати сигнал про завершення вводу MO з T_2 .	$W_{2,3}$	10. Сигнал про завершення обрахунку MA_h .	S_2
11. Чекати сигнал про завершення вводу MX з T_3 .	$W_{3,4}$		
12. Копіювати $z_1=z$	$КУ\,1$		
13. Копіювати $d_1=d$	$КУ\,2$		
14. Копіювати $MX_1=MX$	$КУ\,3$		
15. Обрахувати $MA_h=z_1\cdot MO_h+d_1\cdot(MX_1\cdot ME_h)$.			
16. Сигнал про завершення обрахунку MA_h до T_6 .	$S_{(6,4)}$		

<table>
<tr>
<td>T_3:

1. Ввести Z.
2. Сигнал про завершення вводу Z.
3. Ввести MX.
4. Сигнал про завершення вводу MX.
5. Чекати сигнал про завершення обрахунку z та d з усіх потоків.
6. Чекати сигнал про завершення вводу MO з T_2 .
7. Чекати сигнал про завершення вводу ME з T_1 .
8. Копіювати $z_3 = z$
9. Копіювати $d_3 = d$
10. Копіювати $MX_3 = MX$
11. Обрахувати $MA_h = z_3 \cdot MO_h + d_3 \cdot (MX_3 \cdot ME_h)$.
12. Сигнал про завершення обрахунку MA_h .</td>
<td>S_1

S_2
$W_{1,1}$
$W_{2,2}$
$W_{1,3}$

$КУ\ 1$
$КУ\ 2$
$КУ\ 3$

S_3</td>
</tr>
<tr>
<td>$T_i, i \in \{4,5,7,8\}$:

1. Чекати сигнал про завершення обрахунку z та d з T_1 .
2. Чекати сигнал про завершення вводу ME з T_1 .
3. Чекати сигнал про завершення вводу MO з T_2 .
4. Чекати сигнал про завершення вводу MX з T_3 .
5. Копіювати $z_i = z$
6. Копіювати $d_i = d$
7. Копіювати $MX_i = MX$
8. Обрахувати $MA_h = z_i \cdot MO_h + d_i \cdot (MX_i \cdot ME_h)$.
9. Сигнал про завершення обрахунку MA_h .</td>
<td>$W_{1,1}$

$W_{1,2}$

$W_{2,3}$

$W_{3,4}$

$КУ\ 1$
$КУ\ 2$
$КУ\ 3$

S_1</td>
<td>T_6:

1. Ввести C.
2. Сигнал про завершення вводу C.
3. Чекати сигнал про завершення обрахунку z та d з T_1 .
4. Чекати сигнал про завершення вводу ME з T_1 .
5. Чекати сигнал про завершення вводу MO з T_2 .
6. Чекати сигнал про завершення вводу MX з T_3 .
7. Копіювати $z_6 = z$
8. Копіювати $d_6 = d$
9. Копіювати $MX_6 = MX$
10. Обрахувати $MA_h = z_6 \cdot MO_h + d_6 \cdot (MX_6 \cdot ME_h)$.
11. Чекати сигнал про завершення обрахунку MA_h .
12. Вивести MA</td>
<td>S_1

$W_{1,1}$

$W_{1,2}$

$W_{2,3}$

$W_{3,4}$
S_2

W_3

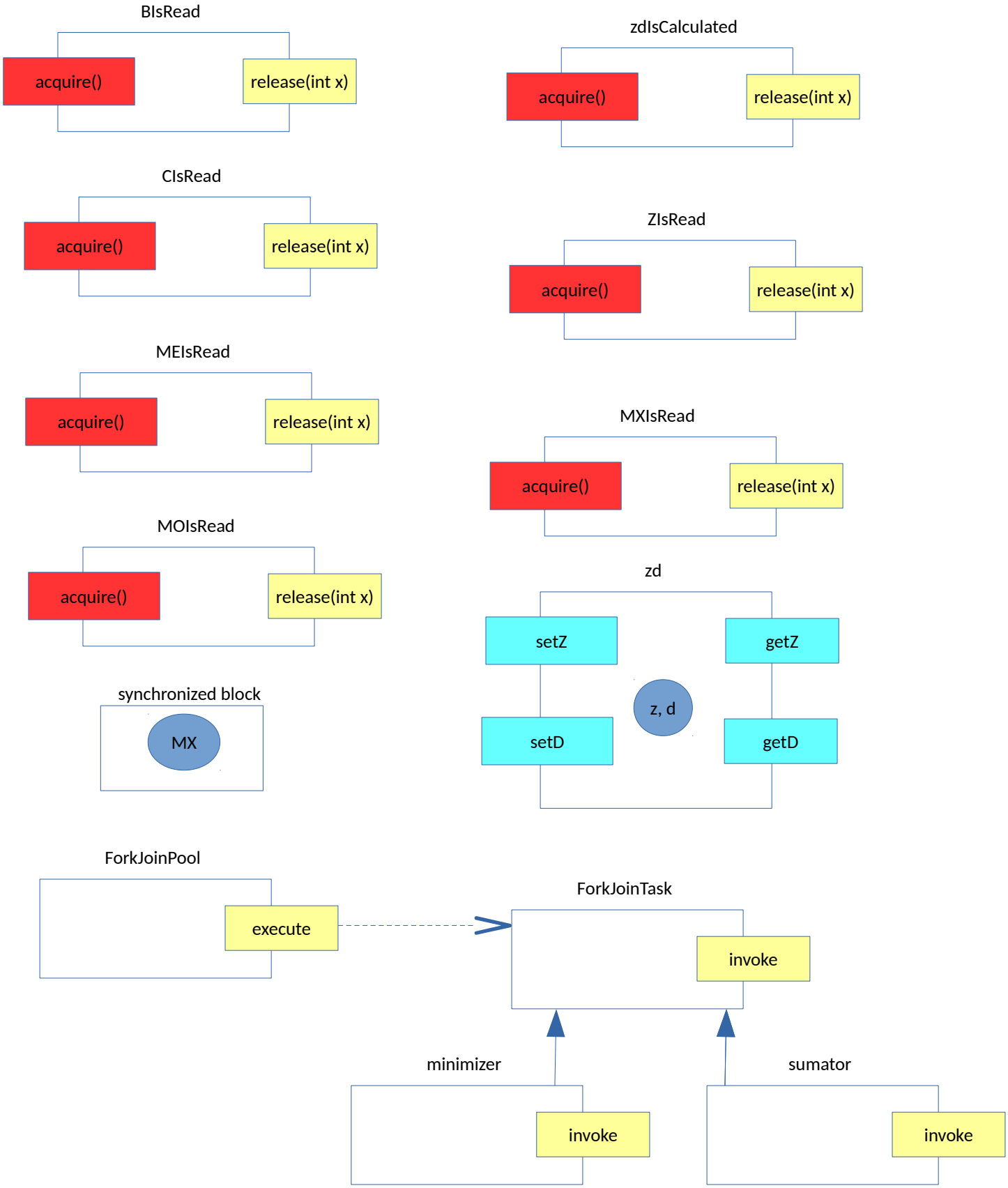
$W_{1,4}$

$W_{2,5}$

$W_{3,6}$
$КУ\ 2$
$КУ\ 3$
$КУ\ 4$

W_7</td>
</tr>
</table>

Структурна схема взаємодії процесів



Лістінг програми:

Main.java

```
/*
 * Dziuba Vlad, IP-42
 * */
import java.util.concurrent.Semaphore;
import java.util.concurrent.RecursiveTask;
import java.util.concurrent.ForkJoinPool;
import java.util.concurrent.ForkJoinTask;
import java.util.function.IntBinaryOperator;

public class Main {
  private static final int N = 4000;
  private static final int P = 8;
  private static final int H = N / P;


  private static DataZD zd = new DataZD();
  private static int[] Z;
  private static Semaphore ZIsRead = new Semaphore(0);
  private static int[][] MO;
  private static Semaphore MOIsRead = new Semaphore(0);
  private static int[] B;
  private static Semaphore BIsRead = new Semaphore(0);
  private static int[] C;
  private static Semaphore CIsRead = new Semaphore(0);
  private static int[][] MX;
  private static Semaphore MXIsRead = new Semaphore(0);
  private static int[][] ME;
  private static Semaphore MEIsRead = new Semaphore(0);

  private static int[][] MA;

  private static Semaphore zdIsCalculated = new Semaphore(0);
  private static Semaphore MAIsCalculated = new Semaphore(0);

  private static Runnable[] calculations = new Runnable[P];

  private static final IntBinaryOperator minZ = (begin, end) -> {
    int min = Integer.MAX_VALUE;
    for (int i = begin; i < end; i++) {
      min = Math.min(min, Z[i]);
    }
    return min;
  };

  private static final IntBinaryOperator sumBC = (begin, end) -> {
    int sum = 0;
    for (int i = begin; i < end; i++) {
      sum += B[i] * C[i];
    }
    return sum;
  };
```

```
  private static final ForkJoinTask<Integer> minimizer =
    new ArraySplitTask(Math::min, minZ, 0, N).task;
  private static final ForkJoinTask<Integer> summator =
    new ArraySplitTask((x, y) -> (x + y), sumBC, 0, N).task;

  static {
    MA = new int[N][];
    for (int i = 0 ; i < N; i++) {
      MA[i] = new int[N];
    }
  }

  public static class DataZD {
    private int z;
    private int d;

    public synchronized void setZ(int z) {
      this.z = z;
    }

    public synchronized void setD(int d) {
      this.d = d;
    }

    public synchronized int getZ() {
      return z;
    }

    public synchronized int getD() {
      return d;
    }
  }

  public static void main(String... args) {
    for (int i = 0; i < P; i++) {
      calculations[i] = new Calculation(i);
    }
       calculations[0] = new ReadCalculation(calculations[0], new
Read1());
       calculations[1] = new ReadCalculation(calculations[1], new
Read2());
       calculations[2] = new ReadCalculation(calculations[2], new
Read3());
       calculations[5] = new ReadWriteCalculation(calculations[5],
new Read6(), new Write6());

    for (int i = 0; i < P-1; i++) {
      new Thread(calculations[i]).start();
    }
    Thread lastThread = new Thread(calculations[P-1]);
    lastThread.start();
```

```
  ForkJoinPool.commonPool().execute(ForkJoinTask.adapt(() -> {
    ForkJoinTask<?> t1 = ForkJoinTask.adapt(() -> {
      try {
        ZIsRead.acquire();
      } catch (Exception e) {
        e.printStackTrace();
      }
      zd.setZ(minimizer.invoke());
    });
    ForkJoinTask<?> t2 = ForkJoinTask.adapt(() -> {
      try {
        BIsRead.acquire();
        CIsRead.acquire();
      } catch (Exception e) {
        e.printStackTrace();
      }
      zd.setD(summator.invoke());
    });
    t1.fork();
    t2.fork();
    t1.join();
    t2.join();
    zdIsCalculated.release(P);
  }));

  try {
    lastThread.join();
  } catch (Exception e) {
    e.printStackTrace();
  }
}

static class Calculation implements Runnable {
  private int index;

  public Calculation(int index) {
    this.index = index;
  }

  @Override
  public void run() {
    try {
      zdIsCalculated.acquire();
    } catch (Exception e) {
      e.printStackTrace();
    }
    int zi, di;
    int[][] MXi;
    zi = zd.getZ();
    di = zd.getD();
    try {
      MXIsRead.acquire();
    } catch (Exception e) {
      e.printStackTrace();
```

```
    }
    synchronized (MX) {
      MXi = MX;
    }
    try {
      MOIsRead.acquire();
      MEIsRead.acquire();
    } catch (Exception e) {
      e.printStackTrace();
    }
    for (int i = index * H; i < (index + 1) * H; i++) {
      for (int j = 0; j < N; j++) {
        MA[j][i] = zi * MO[j][i];
        for (int k = 0; k < N; k++) {
          MA[j][i] += di * MXi[j][k] * ME[k][i];
        }
      }
    }
    MAIsCalculated.release();
  }
}

static int[] initVector() {
  int[] res = new int[N];
  for (int i = 0; i < N; i++) {
    res[i] = 1;
  }
  return res;
}

static int[][] initMatrix() {
  int[][] res = new int[N][];
  for (int i = 0; i < N; i++) {
    res[i] = initVector();
  }
  return res;
}

static class ArraySplitTask {
  private IntBinaryOperator op;
  private IntBinaryOperator opChunk;
  public Recurser task;

  public ArraySplitTask(IntBinaryOperator op,
      IntBinaryOperator opChunk, int begin, int end) {
    this.op = op;
    this.opChunk = opChunk;
    this.task = new Recurser(begin, end);
  }

  @SuppressWarnings("serial")
  public class Recurser extends RecursiveTask<Integer> {
    private int begin;
    private int end;
```

```
    public Recurser(int begin, int end) {
      this.begin = begin;
      this.end = end;
    }

    protected Integer compute() {
      if (end - begin <= H) {
        return opChunk.applyAsInt(begin, end);
      }
      int mid = (begin + end) / 2;
      Recurser t1 = new Recurser(begin, mid);
      t1.fork();
      Recurser t2 = new Recurser(mid, end);
      return op.applyAsInt(t2.compute(), t1.join());
    }
  }
}

static class Read1 implements Runnable {
  @Override
  public void run() {
    B = initVector();
    BIsRead.release();
    ME = initMatrix();
    MEIsRead.release();
  }
}

static class Read2 implements Runnable {
  @Override
  public void run() {
    MO = initMatrix();
    MOIsRead.release();
  }
}

static class Read3 implements Runnable {
  @Override
  public void run() {
    Z = initVector();
    ZIsRead.release();
    MX = initMatrix();
    MXIsRead.release();
  }
}

static class Read6 implements Runnable {
  @Override
  public void run() {
    C = initVector();
    CIsRead.release();
  }
}
```

```
  static class Write6 implements Runnable {
    @Override
    public void run() {
      try {
        MAIsCalculated.acquire(P);
      } catch (Exception e) {
        e.printStackTrace();
      }
      if (N < 20) {
        for (int i = 0; i < N; i++) {
          for (int j = 0; j < N; j++) {
            System.out.print(MA[i][j]);
            System.out.print(" ");
          }
          System.out.println();
        }
      }
    }
  }

  static class ReadCalculation implements Runnable {
    private Runnable reader;
    private Runnable calculation;

     public ReadCalculation(Runnable calculation, Runnable reader)
{
      this.reader = reader;
      this.calculation = calculation;
    }

    @Override
    public void run() {
      reader.run();
      calculation.run();
    }
  }

  static class ReadWriteCalculation implements Runnable {
    private Runnable writer;
    private ReadCalculation readCalculation;

       public  ReadWriteCalculation(Runnable  calculation,  Runnable
reader, Runnable writer) {
          this.readCalculation  =  new  ReadCalculation(calculation,
reader);
      this.writer = writer;
    }

    @Override
    public void run() {
      readCalculation.run();
      writer.run();
    }
```

```
    }
}
```