Національний технічний університет України
«Київський політехнічний інститут»

Лабораторна робота №3

з дисципліни: «*Паралельне програмування-2*»
Тема: «C#. Семафори. Події. Мютекси. Критичні секції.Volatile змінні.»

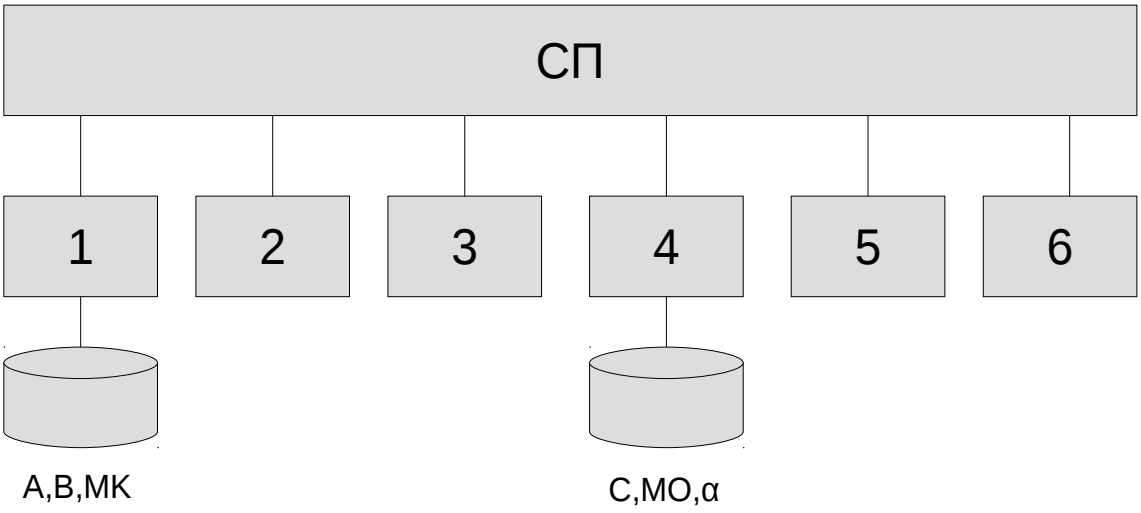
Виконав: *студент групи ІП-32*
Ковальчук Олександр Миронович

Київ-2016р.

Технічне завдання: $A = sort\left(\alpha \cdot B + C \cdot \left(MO \cdot MK\right)\right)$

Мова програмування: C#,

Засоби взаємодії: семафори, події, мютекси, критичні секції, volatile змінні.



Виконання роботи:

Етап 1. Побудова паралельного алгоритму.

1. $A_H = sort\left(\alpha \cdot B_H + C \cdot \left(MO \cdot MK_H\right)\right)$
2. $A_{2H} = sort'\left(A_H, A_H\right)$
3. $A_{3H} = sort'\left(A_{2H}, A_H\right)$
4. $A = sort'\left(A_{3H}, A_{3H}\right)$

Спільний ресурс: C, MO, α

Етап 2. Розробка алгоритмів роботи кожного процесу.

Задача T1

№	Дія	КД/ТС
1	Введення B, MK	
2	Сигнал задачам T2 — T6 про введення B, MK	S2-1, S3-1, S4-1, S5-1, S6-1
3	Очікувати введення у задачі T4	W4-1
4	Копіювати C1 = C, MO1 = MO	КД
5	Обчислення $A_H = sort\left(\alpha \cdot B_H + C1 \cdot \left(MO1 \cdot MK_H\right)\right)$	
6	Очікувати завершення обчислень у задачі T2	W2-1
7	Сортування злиттям $A_{2H} = sort'\left(A_H, A_H\right)$	
8	Очікувати завершення обчислень у задачі T3	W3-1
9	Сортування злиттям $A_{3H} = sort'\left(A_{2H}, A_H\right)$	
10	Очікувати завершення сортування у задачі T4	W4-2
11	Сортування злиттям $A = sort'\left(A_{3H}, A_{3H}\right)$	
12	Вивід A	

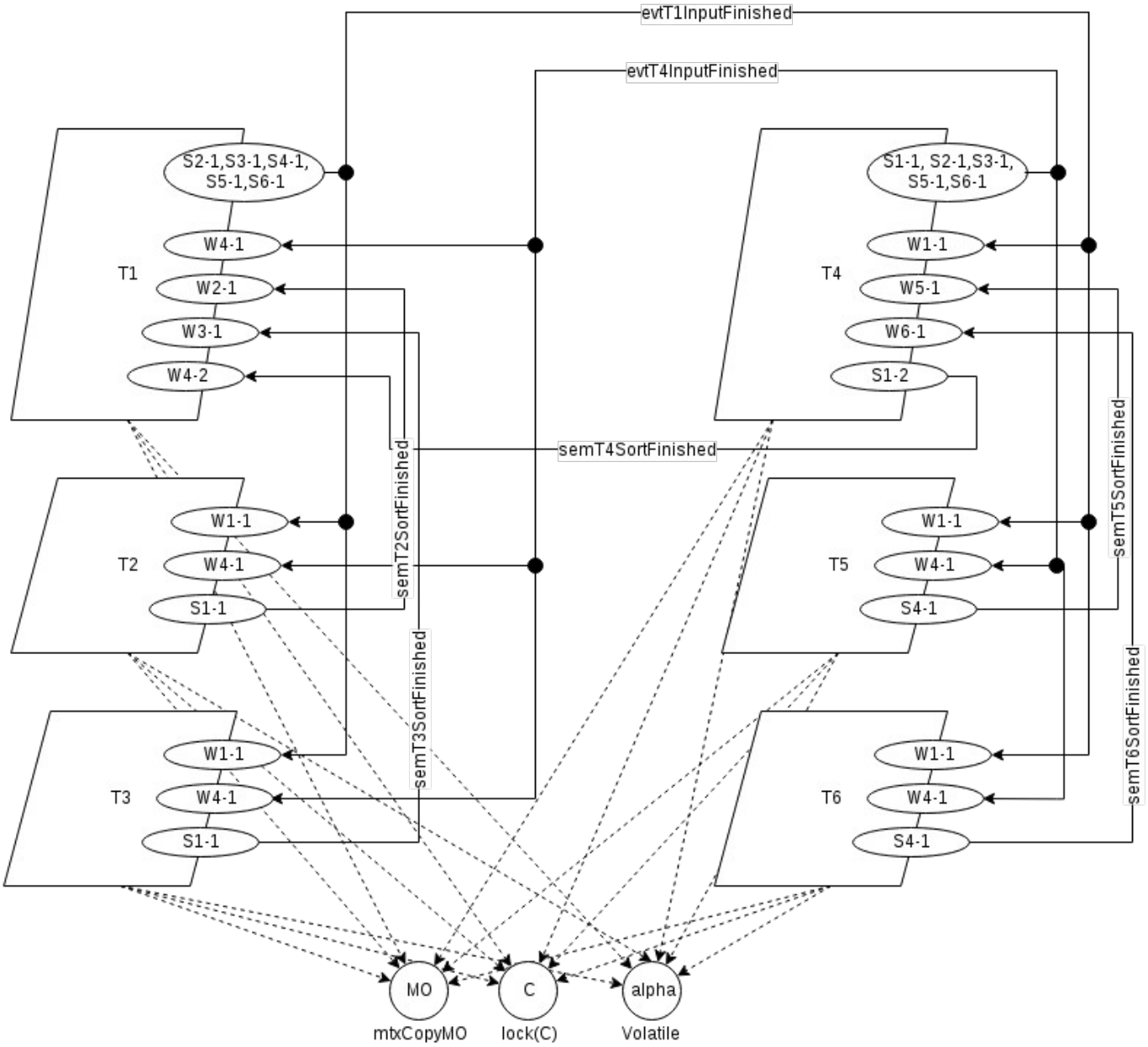
Задача T4

№	Дія	КД/ТС
1	Введення C, MO, α	
2	Сигнал задачам T1 — T3 та T5 — T6 про введення C, MO, α	S1-1, S2-1, S3-1, S5-1, S6-1
3	Очікувати введення у задачі T1	W1-1
4	Копіювати C4 = C, MO4 = MO	КД
5	Обчислення $A_H = sort\left(\alpha \cdot B_H + C4 \cdot \left(MO4 \cdot MK_H\right)\right)$	
6	Очікувати завершення обчислень у задачі T5	W5-1
7	Сортування злиттям $A_{2H} = sort'\left(A_H, A_H\right)$	
8	Очікувати завершення обчислень у задачі T6	W6-1
9	Сортування злиттям $A_{3H} = sort'\left(A_{2H}, A_H\right)$	
10	Сигнал задачі T1 про закінчення сортування	S1-2

Задачі Ti, $i \in \{2,3,5,6\}$

№	Дія	КД/ТС
1	Очікувати введення у задачах T1, T4	W1-1, W4-1
2	Копіювати Ci = C, MOi = MO	КД
3	Обчислення $A_H = sort\left(\alpha \cdot B_H + Ci \cdot \left(MOi \cdot MK_H\right)\right)$	
4	Сигнал задачі $\begin{cases} T1, якщо\ i<4 \\ T4, якщо\ i>4 \end{cases}$ про закінчення обчислень	$\begin{cases} S1-1, якщо\ i<4 \\ S4-1, якщо\ i>4 \end{cases}$

Етап 3. Розробка структурної схеми взаємодії задач.



Етап 4. Лістинг програми

```
/**
 * Parallel Programming 2
 * C#. Mutex, Semaphore, CriticalSection, Event, volatile variables
 * A = sort(alpha*B + C*(MO * MK))
 *
 * @author Oleksandr Kovalchuk
 * @group IP-32
 */
using System;
using System.Threading;

namespace _03_csharp {

    class Lab3 {

        public static readonly int N = 6;
        public static readonly int P = 6;
        public static readonly int H = N / P;

        public static EventWaitHandle evtT1InputFinished;
        public static EventWaitHandle evtT4InputFinished;

        public static Mutex mtxCopyMO;

        public static Semaphore semT2SortFinished;
        public static Semaphore semT3SortFinished;
        public static Semaphore semT4SortFinished;
        public static Semaphore semT5SortFinished;
        public static Semaphore semT6SortFinished;

        static void Main(string[] args) {
            Console.WriteLine("Lab 3 started");
            // Prepare synchronization objects
            evtT1InputFinished = new EventWaitHandle(initialState: false, mode: EventResetMode.ManualReset);
            evtT4InputFinished = new EventWaitHandle(initialState: false, mode:EventResetMode.ManualReset);

            mtxCopyMO = new Mutex(initiallyOwned: false);

            semT2SortFinished = new Semaphore(initialCount: 0, maximumCount: 1);
            semT3SortFinished = new Semaphore(initialCount: 0, maximumCount: 1);
            semT4SortFinished = new Semaphore(initialCount: 0, maximumCount: 1);
            semT5SortFinished = new Semaphore(initialCount: 0, maximumCount: 1);
            semT6SortFinished = new Semaphore(initialCount: 0, maximumCount: 1);

            // Create threads
            Thread[] threads = new Thread[P];
            threads[0] = new Thread(TaskDefinition.thread1);
            threads[1] = new Thread(TaskDefinition.thread2);
            threads[2] = new Thread(TaskDefinition.thread3);
            threads[3] = new Thread(TaskDefinition.thread4);
            threads[4] = new Thread(TaskDefinition.thread5);
            threads[5] = new Thread(TaskDefinition.thread6);

            // Run threads
            foreach (Thread t in threads) { t.Start(); }

            // Wait threads to finish
            foreach (Thread t in threads) { t.Join(); }

            Console.WriteLine("Lab 3 finished");
            Console.ReadKey();
        }
    }
}

namespace _03_csharp {
    /**
     * Define storage class with all
     * required data
     */
    class Storage {
        public static int[][] MO;
        public static int[][] MK;
        public static int[] A = new int[Lab3.N];
        public static int[] B;
        public static int[] C;
        public static volatile int alpha;
    }
}


using System;
using System.Text;


namespace _03_csharp {
    class TaskDefinition {
        public static void thread1() {
            Console.WriteLine("thread 1 started");

            // Input B, MK
            Storage.B = Util.generateVector(Lab3.N);
```

```
    Storage.MK = Util.generateMatrix(Lab3.N);

    // Signal other threads about B, MK input finished
    Lab3.evtT1InputFinished.Set();

    // Wait for T4 to finish input
    Lab3.evtT4InputFinished.WaitOne();

    // Copy shared data
    int[] C1;
    int[][] MO1;

    // The following is the syntaxic sugar for
    // try {
    //     Monitor.Enter(Storage.C);
    //     C1 = Util.copyVector(Storage.C);
    // } finally { Monitor.Exit(Storage.C); }
    lock (Storage.C) { C1 = Util.copyVector(Storage.C); }

    Lab3.mtxCopyMO.WaitOne();
    MO1 = Util.copyMatrix(Storage.MO);
    Lab3.mtxCopyMO.ReleaseMutex();

    // Compute
    for (int i = 0; i < Lab3.H; ++i) {
        Storage.A[i] = 0;
        for (int j = 0; j < Lab3.N; ++j) {
            for (int k = 0; k < Lab3.N; ++k) {
                Storage.A[i] += C1[j] * MO1[j][k] * Storage.MK[k][i];
            }
        }
        Storage.A[i] += Storage.alpha * Storage.B[i];
    }

    Array.Sort(Storage.A, index: 0, length: Lab3.H);

    // Wait T2 to finish sorting
    Lab3.semT2SortFinished.WaitOne();
    Util.merge_(Storage.A, 0, Lab3.H, 2 * Lab3.H);

    // Wait T3 to finish sorting
    Lab3.semT3SortFinished.WaitOne();
    Util.merge_(Storage.A, 0, 2 * Lab3.H, 3 * Lab3.H);

    // Wait T4 to finish sorting
    Lab3.semT4SortFinished.WaitOne();
    Util.merge_(Storage.A, 0, 3 * Lab3.H, Lab3.N);

    StringBuilder sb = new StringBuilder("Result: [");
    foreach (int i in Storage.A) { sb.Append(" " + i + " "); }
    sb.Append(']');

    if (Lab3.N < 16) { Console.WriteLine(sb.ToString()); }
    Console.WriteLine("thread 1 finished");
}

public static void thread2() {
    Console.WriteLine("thread 2 started");

    // Wait for T1 and T4 to finish input
    Lab3.evtT1InputFinished.WaitOne();
    Lab3.evtT4InputFinished.WaitOne();

    // Copy shared data
    int[] C2;
    int[][] MO2;

    // The following is the syntaxic sugar for
    // try {
    //     Monitor.Enter(Storage.C);
    //     C2 = Util.copyVector(Storage.C);
    // } finally { Monitor.Exit(Storage.C); }
    lock (Storage.C) { C2 = Util.copyVector(Storage.C); }

    Lab3.mtxCopyMO.WaitOne();
    MO2 = Util.copyMatrix(Storage.MO);
    Lab3.mtxCopyMO.ReleaseMutex();

    // Compute
    for (int i = Lab3.H; i < 2 * Lab3.H; ++i) {
        Storage.A[i] = 0;
        for (int j = 0; j < Lab3.N; ++j) {
            for (int k = 0; k < Lab3.N; ++k) {
                Storage.A[i] += C2[j] * MO2[j][k] * Storage.MK[k][i];
            }
        }
        Storage.A[i] += Storage.alpha * Storage.B[i];
    }

    Array.Sort(Storage.A, index: Lab3.H, length: Lab3.H);
    Lab3.semT2SortFinished.Release();

    Console.WriteLine("thread 2 finished");
}

public static void thread3() {
```

```
    Console.WriteLine("thread 3 started");

    // Wait for T1 and T4 to finish input
    Lab3.evtT1InputFinished.WaitOne();
    Lab3.evtT4InputFinished.WaitOne();

    // Copy shared data
    int[] C3;
    int[][] MO3;

    // The following is the syntaxic sugar for
    // try {
    //     Monitor.Enter(Storage.C);
    //     C3 = Util.copyVector(Storage.C);
    // } finally { Monitor.Exit(Storage.C); }
    lock (Storage.C) { C3 = Util.copyVector(Storage.C); }

    Lab3.mtxCopyMO.WaitOne();
    MO3 = Util.copyMatrix(Storage.MO);
    Lab3.mtxCopyMO.ReleaseMutex();

    // Compute
    for (int i = 2 * Lab3.H; i < 3 * Lab3.H; ++i) {
        Storage.A[i] = 0;
        for (int j = 0; j < Lab3.N; ++j) {
            for (int k = 0; k < Lab3.N; ++k) {
                Storage.A[i] += C3[j] * MO3[j][k] * Storage.MK[k][i];
            }
        }
        Storage.A[i] += Storage.alpha * Storage.B[i];
    }

    Array.Sort(Storage.A, index: 2 * Lab3.H, length: Lab3.H);
    Lab3.semT3SortFinished.Release();

    Console.WriteLine("thread 3 finished");
}

public static void thread4() {
    Console.WriteLine("thread 4 started");

    // Input C, MO, alpha
    Storage.C = Util.generateVector(Lab3.N);
    Storage.MO = Util.generateMatrix(Lab3.N);
    Storage.alpha = 1;

    // Signal other threads about B, MK input finished
    Lab3.evtT4InputFinished.Set();

    // Wait for T4 to finish input
    Lab3.evtT1InputFinished.WaitOne();

    // Copy shared data
    int[] C4;
    int[][] MO4;

    // The following is the syntaxic sugar for
    // try {
    //     Monitor.Enter(Storage.C);
    //     C4 = Util.copyVector(Storage.C);
    // } finally { Monitor.Exit(Storage.C); }
    lock (Storage.C) { C4 = Util.copyVector(Storage.C); }

    Lab3.mtxCopyMO.WaitOne();
    MO4 = Util.copyMatrix(Storage.MO);
    Lab3.mtxCopyMO.ReleaseMutex();

    // Compute
    for (int i = 3 * Lab3.H; i < 4 * Lab3.H; ++i) {
        Storage.A[i] = 0;
        for (int j = 0; j < Lab3.N; ++j) {
            for (int k = 0; k < Lab3.N; ++k) {
                Storage.A[i] += C4[j] * MO4[j][k] * Storage.MK[k][i];
            }
        }
        Storage.A[i] += Storage.alpha * Storage.B[i];
    }

    Array.Sort(Storage.A, index: 3 * Lab3.H, length: Lab3.H);

    // Wait T5 to finish sorting
    Lab3.semT5SortFinished.WaitOne();
    Util.merge_(Storage.A, 3 * Lab3.H, 4 * Lab3.H, 5 * Lab3.H);

    // Wait T6 to finish sorting
    Lab3.semT6SortFinished.WaitOne();
    Util.merge_(Storage.A, 3 * Lab3.H, 5 * Lab3.H, 6 * Lab3.H);

    Lab3.semT4SortFinished.Release();

    Console.WriteLine("thread 4 finished");
}

public static void thread5() {
    Console.WriteLine("thread 5 started");
```

```
            // Wait for T1 and T4 to finish input
            Lab3.evtT1InputFinished.WaitOne();
            Lab3.evtT4InputFinished.WaitOne();

            // Copy shared data
            int[] C5;
            int[][] MO5;

            // The following is the syntaxic sugar for
            // try {
            //     Monitor.Enter(Storage.C);
            //     C5 = Util.copyVector(Storage.C);
            // } finally { Monitor.Exit(Storage.C); }
            lock (Storage.C) { C5 = Util.copyVector(Storage.C); }

            Lab3.mtxCopyMO.WaitOne();
            MO5 = Util.copyMatrix(Storage.MO);
            Lab3.mtxCopyMO.ReleaseMutex();

            // Compute
            for (int i = 4 * Lab3.H; i < 5 * Lab3.H; ++i) {
                Storage.A[i] = 0;
                for (int j = 0; j < Lab3.N; ++j) {
                    for (int k = 0; k < Lab3.N; ++k) {
                        Storage.A[i] += C5[j] * MO5[j][k] * Storage.MK[k][i];
                    }
                }
                Storage.A[i] += Storage.alpha * Storage.B[i];
            }

            Array.Sort(Storage.A, index: 4 * Lab3.H, length: Lab3.H);
            Lab3.semT5SortFinished.Release();

            Console.WriteLine("thread 5 finished");
        }

        public static void thread6() {
            Console.WriteLine("thread 6 started");

            // Wait for T1 and T4 to finish input
            Lab3.evtT1InputFinished.WaitOne();
            Lab3.evtT4InputFinished.WaitOne();

            // Copy shared data
            int[] C6;
            int[][] MO6;

            // The following is the syntaxic sugar for
            // try {
            //     Monitor.Enter(Storage.C);
            //     C6 = Util.copyVector(Storage.C);
            // } finally { Monitor.Exit(Storage.C); }
            lock (Storage.C) { C6 = Util.copyVector(Storage.C); }

            Lab3.mtxCopyMO.WaitOne();
            MO6 = Util.copyMatrix(Storage.MO);
            Lab3.mtxCopyMO.ReleaseMutex();

            // Compute
            for (int i = 5 * Lab3.H; i < 6 * Lab3.H; ++i) {
                Storage.A[i] = 0;
                for (int j = 0; j < Lab3.N; ++j) {
                    for (int k = 0; k < Lab3.N; ++k) {
                        Storage.A[i] += C6[j] * MO6[j][k] * Storage.MK[k][i];
                    }
                }
                Storage.A[i] += Storage.alpha * Storage.B[i];
            }

            Array.Sort(Storage.A, index: 5 * Lab3.H, length: Lab3.H);
            Lab3.semT6SortFinished.Release();

            Console.WriteLine("thread 6 finished");
        }
    }
}


using System;

namespace _03_csharp {
    class Util {
        public static int[] generateVector(int size, int filler = 1) {
            int[] result = new int[size];
            for (int i = 0; i < size; ++i) {
                result[i] = filler;
            }
            return result;
        }

        public static int[][] generateMatrix(int size, int filler = 1) {
            int[][] result = new int[size][];
            for (int r = 0; r < size; ++r) {
                result[r] = generateVector(size, filler);
            }
            return result;
```

```
        }

        // _ means function has a side-effect
        public static void merge_(int[] arr, int start, int middle, int end){
            int[] first = new int[middle - start];
            int[] second = new int[end - middle];
            int fptr = 0, sptr = 0, rptr = start;


            Array.Copy(arr, start, first, 0, middle - start);
            Array.Copy(arr, middle, second, 0, end - middle);

            while (rptr < end) {
                if (fptr == first.Length) {
                    Array.Copy(second, sptr, arr, rptr, end - rptr);
                    break;
                }

                if (sptr == second.Length) {
                    Array.Copy(first, fptr, arr, rptr, end - rptr);
                    break;
                }

                arr[rptr++] = first[fptr] <= second[sptr] ? first[fptr++] : second[sptr++];
            }
        }

        public static int[] copyVector(int[] original) {
            int[] copy = new int[original.Length];
            Array.Copy(original, copy, original.Length);
            return copy;
        }

        public static int[][] copyMatrix(int[][] original) {
            int[][] copy = new int[original.Length][];
            for (int r = 0; r < original.Length; ++r) {
                copy[r] = copyVector(original[r]);
            }
            return copy;
        }
    }
}
```