

Міністерство освіти і науки України

Національний технічний університет України

«Київський політехнічний інститут ім. Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

**Лабораторна робота №2**

“WinAPI. Семафоры, События, Мютексы, Критические секции”

з дисципліни “Програмування для Паралельних Компьютерних Систем”

Виконав:

студент 3 курсу групи ІО-52

Бояршин Ігор

Номер заліковки: 5207

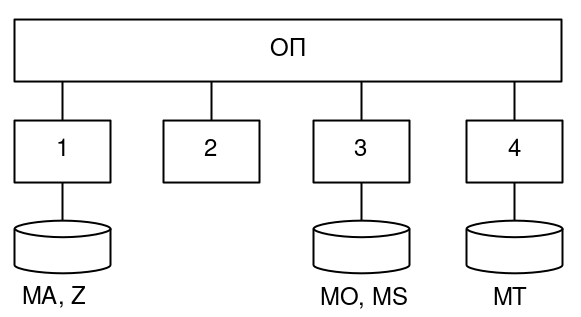
Перевірив:

Корочкін О.В.

Київ 2018 р.

**Техническое задание**

Структура паралельной компъютерной системы с общей памятью:



Выражение для подсчёта:

MA = max(Z) \* (MO\*MT) + min(Z) \* MS

Библиотека параллельного программирования:

WinAPI

Средства взаимодействия задач:

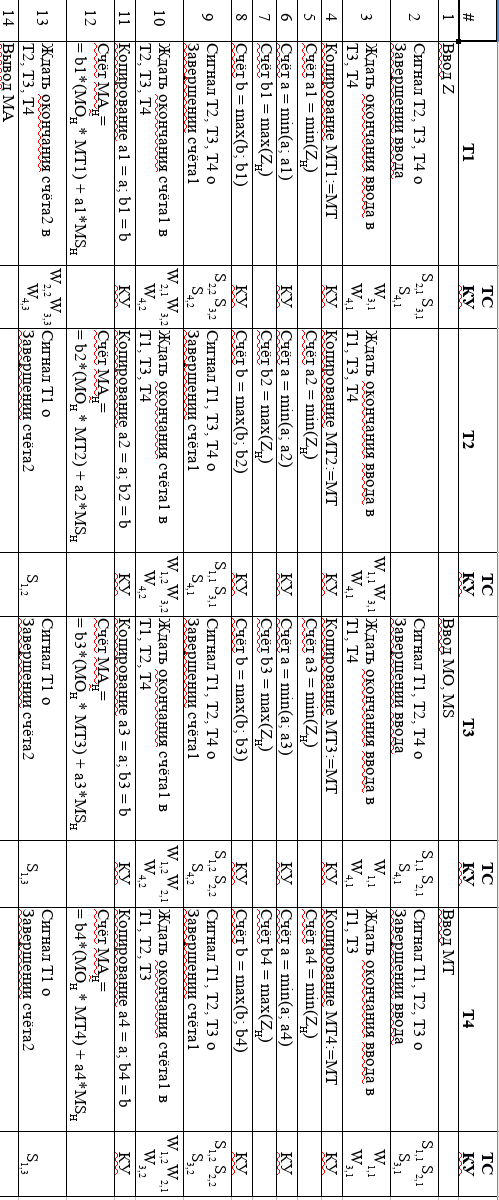
Семафоры, События, Мютексы, Критические секции

**Выполнение работы**

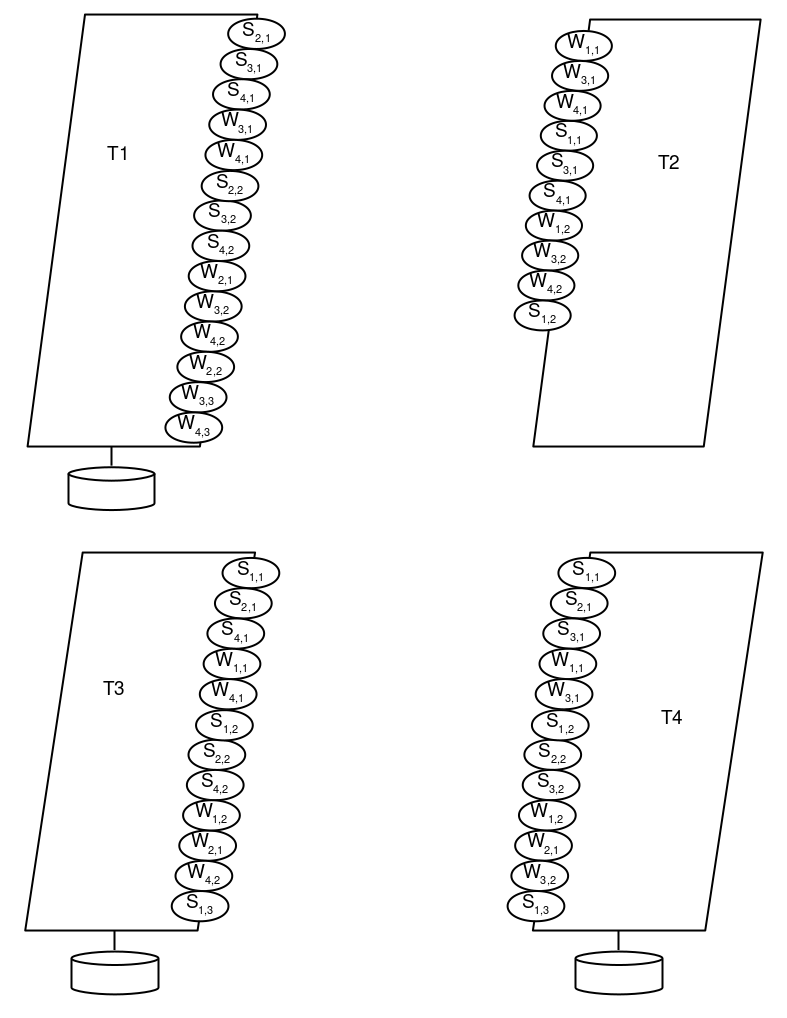
**Этап 1:** Разработка параллельного математического алгоритма

1. ai = min(ZH)
2. a = min(a; ai)
   * ОР: a
3. bi = max(ZH)
4. b = max(b; bi)
   * ОР: b
5. MAH = b\* (MOH \* MT) + a\* MSH
   * ОР: MT, a, b

**Этап 2:** Разработка алгоритмов потоков



**Этап 3:** Разработка схемы взаимодействия потоков



**Этап 4:** Разработка программы

//-----------------------------------------------------------------------------

// Lab2: WinAPI. Semaphores, Events, Mutexes, Critical Sections

// Task: MA = max(Z) \* (MO \* MT) + min(Z) \* MS

// Author: Igor Boyarshin

// Date: 12.03.2018

//-----------------------------------------------------------------------------

#include "windows.h"

#include <iostream>

// Constants

const unsigned int N = 8;

const unsigned int P = 4;

const unsigned int H = N / P;

const unsigned int STACK\_SIZE = 100000000;

// Types

struct Vector {

private:

unsigned int elements[N];

public:

Vector() : elements() {

std::fill(elements, elements + N, 0);

}

unsigned int& operator[](unsigned int row) {

return elements[row];

}

const unsigned int& operator[](unsigned int row) const {

return elements[row];

}

};

struct Matrix {

private:

struct Vector elements[N];

public:

//Matrix() {

//for(unsigned int i = 0; i < N; i++) elements[i] = Vector;

//}

struct Vector& operator[](unsigned int row) {

return elements[row];

}

const struct Vector& operator[](unsigned int row) const {

return elements[row];

}

Matrix copy() const {

Matrix m;

for (unsigned int i = 0; i < N; i++) {

const struct Vector& v = elements[i];

for (unsigned int j = 0; j < N; j++) {

m[i][j] = v[j];

}

}

return m;

}

};

// Functions

void fillVector(struct Vector& vector, unsigned int value);

void fillMatrix(Matrix& matrix, unsigned int value);

void outputMatrix(const Matrix& matrix);

unsigned int findMin(const struct Vector& Vector, unsigned int low, unsigned int high);

unsigned int findMax(const struct Vector& Vector, unsigned int low, unsigned int high);

//unsigned int min(unsigned int a, unsigned int b) { return a < b ? a : b; }

//unsigned int max(unsigned int a, unsigned int b) { return a > b ? a : b; }

// Data

unsigned int a, b;

Vector Z;

Matrix MO, MT, MS, MA;

// Semaphores

HANDLE Semaphore\_Calculations1EndIn1;

HANDLE Semaphore\_Calculations1EndIn2;

HANDLE Semaphore\_Calculations1EndIn3;

HANDLE Semaphore\_Calculations1EndIn4;

// Mutexes

HANDLE Mutex\_SetA;

HANDLE Mutex\_SetB;

// Critical Sections

CRITICAL\_SECTION CriticalSection\_CopyMT;

CRITICAL\_SECTION CriticalSection\_CopyAb;

// Events

HANDLE Event\_InputFinishIn1;

HANDLE Event\_InputFinishIn3;

HANDLE Event\_InputFinishIn4;

HANDLE Event\_Calculation2EndIn2;;

HANDLE Event\_Calculation2EndIn3;;

HANDLE Event\_Calculation2EndIn4;;

//-----------------------------------------------------------------------------

// Thread 1

void T1() {

const unsigned int threadIndex = 1;

const unsigned int low = (threadIndex - 1) \* H;

const unsigned int high = threadIndex \* H;

std::cout << "Thread " << threadIndex << " started..." << std::endl;

// 1. Input Z

fillVector(Z, 1);

// 2. Signal T2, T3, T4 about input finish

SetEvent(Event\_InputFinishIn1);

// 3. Wait for input finish in T3, T4

WaitForSingleObject(Event\_InputFinishIn3, INFINITE);

WaitForSingleObject(Event\_InputFinishIn4, INFINITE);

// 4. Copy MT1 = MT

EnterCriticalSection(&CriticalSection\_CopyMT);

const Matrix MT1 = MT.copy();

LeaveCriticalSection(&CriticalSection\_CopyMT);

// 5.

unsigned int a1 = findMin(Z, low, high);

// 6.

WaitForSingleObject(Mutex\_SetA, INFINITE);

a = min(a, a1);

ReleaseMutex(Mutex\_SetA);

// 7.

unsigned int b1 = findMax(Z, low, high);

// 8.

WaitForSingleObject(Mutex\_SetB, INFINITE);

b = max(b, b1);

ReleaseMutex(Mutex\_SetB);

// 9. Signal T2, T3, T4 about Calculation1 finish

ReleaseSemaphore(Semaphore\_Calculations1EndIn1, 3, NULL);

// 10. Wait for Calculation1 finish in T2, T3, T4

WaitForSingleObject(Semaphore\_Calculations1EndIn2, INFINITE);

WaitForSingleObject(Semaphore\_Calculations1EndIn3, INFINITE);

WaitForSingleObject(Semaphore\_Calculations1EndIn4, INFINITE);

// 11. Copy a1 = a; b1 = b;

EnterCriticalSection(&CriticalSection\_CopyAb);

a1 = a;

b1 = b;

LeaveCriticalSection(&CriticalSection\_CopyAb);

// 12. Calculation2

for (unsigned int i = low; i < high; i++) {

for (unsigned int j = 0; j < N; j++) {

unsigned int product = 0;

for (unsigned int k = 0; k < N; k++) {

product += MO[i][k] \* MT1[k][j];

}

MA[i][j] = b1 \* product + a1 \* MS[i][j];

}

}

// 13. Wait for Calculation2 finish in T2, T3, T4

WaitForSingleObject(Event\_Calculation2EndIn2, INFINITE);

WaitForSingleObject(Event\_Calculation2EndIn3, INFINITE);

WaitForSingleObject(Event\_Calculation2EndIn4, INFINITE);

// 14. Output MA

if (N <= 8) {

outputMatrix(MA);

}

std::cout << "Thread " << threadIndex << " finished." << std::endl;

};

//-----------------------------------------------------------------------------

// Thread 2

void T2() {

const unsigned int threadIndex = 2;

const unsigned int low = (threadIndex - 1) \* H;

const unsigned int high = threadIndex \* H;

std::cout << "Thread " << threadIndex << " started..." << std::endl;

// 3. Wait for input finish in T1, T3, T4

WaitForSingleObject(Event\_InputFinishIn1, INFINITE);

WaitForSingleObject(Event\_InputFinishIn3, INFINITE);

WaitForSingleObject(Event\_InputFinishIn4, INFINITE);

// 4. Copy MT2 = MT

EnterCriticalSection(&CriticalSection\_CopyMT);

const Matrix MT2 = MT.copy();

LeaveCriticalSection(&CriticalSection\_CopyMT);

// 5.

unsigned int a2 = findMin(Z, low, high);

// 6.

WaitForSingleObject(Mutex\_SetA, INFINITE);

a = min(a, a2);

ReleaseMutex(Mutex\_SetA);

// 7.

unsigned int b2 = findMax(Z, low, high);

// 8.

WaitForSingleObject(Mutex\_SetB, INFINITE);

b = max(b, b2);

ReleaseMutex(Mutex\_SetB);

// 9. Signal T1, T3, T4 about Calculation1 finish

ReleaseSemaphore(Semaphore\_Calculations1EndIn2, 3, NULL);

// 10. Wait for Calculation1 finish in T1, T3, T4

WaitForSingleObject(Semaphore\_Calculations1EndIn1, INFINITE);

WaitForSingleObject(Semaphore\_Calculations1EndIn3, INFINITE);

WaitForSingleObject(Semaphore\_Calculations1EndIn4, INFINITE);

// 11. Copy a2 = a; b2 = b;

EnterCriticalSection(&CriticalSection\_CopyAb);

a2 = a;

b2 = b;

LeaveCriticalSection(&CriticalSection\_CopyAb);

// 12. Calculation2

for (unsigned int i = low; i < high; i++) {

for (unsigned int j = 0; j < N; j++) {

unsigned int product = 0;

for (unsigned int k = 0; k < N; k++) {

product += MO[i][k] \* MT2[k][j];

}

MA[i][j] = b2 \* product + a2 \* MS[i][j];

}

}

// 13.Signal T1 about Calculations2 finish

SetEvent(Event\_Calculation2EndIn2);

std::cout << "Thread " << threadIndex << " finished." << std::endl;

};

//-----------------------------------------------------------------------------

// Thread 3

void T3() {

const unsigned int threadIndex = 3;

const unsigned int low = (threadIndex - 1) \* H;

const unsigned int high = threadIndex \* H;

std::cout << "Thread " << threadIndex << " started..." << std::endl;

// 1. Input MO, MS

fillMatrix(MO, 1);

fillMatrix(MS, 1);

// 2. Signal T1, T2, T4 about input finish

SetEvent(Event\_InputFinishIn3);

// 3. Wait for input finish in T1, T4

WaitForSingleObject(Event\_InputFinishIn1, INFINITE);

WaitForSingleObject(Event\_InputFinishIn4, INFINITE);

// 4. Copy MT3 = MT

EnterCriticalSection(&CriticalSection\_CopyMT);

const Matrix MT3 = MT.copy();

LeaveCriticalSection(&CriticalSection\_CopyMT);

// 5.

unsigned int a3 = findMin(Z, low, high);

// 6.

WaitForSingleObject(Mutex\_SetA, INFINITE);

a = min(a, a3);

ReleaseMutex(Mutex\_SetA);

// 7.

unsigned int b3 = findMax(Z, low, high);

// 8.

WaitForSingleObject(Mutex\_SetB, INFINITE);

b = max(b, b3);

ReleaseMutex(Mutex\_SetB);

// 9. Signal T1, T2, T4 about Calculation1 finish

ReleaseSemaphore(Semaphore\_Calculations1EndIn3, 3, NULL);

// 10. Wait for Calculation1 finish in T1, T2, T4

WaitForSingleObject(Semaphore\_Calculations1EndIn1, INFINITE);

WaitForSingleObject(Semaphore\_Calculations1EndIn2, INFINITE);

WaitForSingleObject(Semaphore\_Calculations1EndIn4, INFINITE);

// 11. Copy a3 = a; b3 = b;

EnterCriticalSection(&CriticalSection\_CopyAb);

a3 = a;

b3 = b;

LeaveCriticalSection(&CriticalSection\_CopyAb);

// 12. Calculation2

for (unsigned int i = low; i < high; i++) {

for (unsigned int j = 0; j < N; j++) {

unsigned int product = 0;

for (unsigned int k = 0; k < N; k++) {

product += MO[i][k] \* MT3[k][j];

}

MA[i][j] = b3 \* product + a3 \* MS[i][j];

}

}

// 13.Signal T1 about Calculations2 finish

SetEvent(Event\_Calculation2EndIn3);

std::cout << "Thread " << threadIndex << " finished." << std::endl;

};

//-----------------------------------------------------------------------------

// Thread 4

void T4() {

const unsigned int threadIndex = 4;

const unsigned int low = (threadIndex - 1) \* H;

const unsigned int high = threadIndex \* H;

std::cout << "Thread " << threadIndex << " started..." << std::endl;

// 1. Input MT

fillMatrix(MT, 1);

// 2. Signal T1, T2, T3 about input finish

SetEvent(Event\_InputFinishIn4);

// 3. Wait for input finish in T1, T3

WaitForSingleObject(Event\_InputFinishIn1, INFINITE);

WaitForSingleObject(Event\_InputFinishIn3, INFINITE);

// 4. Copy MT4 = MT

EnterCriticalSection(&CriticalSection\_CopyMT);

const Matrix MT4 = MT.copy();

LeaveCriticalSection(&CriticalSection\_CopyMT);

// 5.

unsigned int a4 = findMin(Z, low, high);

// 6.

WaitForSingleObject(Mutex\_SetA, INFINITE);

a = min(a, a4);

ReleaseMutex(Mutex\_SetA);

// 7.

unsigned int b4 = findMax(Z, low, high);

// 8.

WaitForSingleObject(Mutex\_SetB, INFINITE);

b = max(b, b4);

ReleaseMutex(Mutex\_SetB);

// 9. Signal T1, T2, T3 about Calculation1 finish

ReleaseSemaphore(Semaphore\_Calculations1EndIn4, 3, NULL);

// 10. Wait for Calculation1 finish in T1, T2, T3

WaitForSingleObject(Semaphore\_Calculations1EndIn1, INFINITE);

WaitForSingleObject(Semaphore\_Calculations1EndIn2, INFINITE);

WaitForSingleObject(Semaphore\_Calculations1EndIn3, INFINITE);

// 11. Copy a4 = a; b4 = b;

EnterCriticalSection(&CriticalSection\_CopyAb);

a4 = a;

b4 = b;

LeaveCriticalSection(&CriticalSection\_CopyAb);

// 12. Calculation2

for (unsigned int i = low; i < high; i++) {

for (unsigned int j = 0; j < N; j++) {

unsigned int product = 0;

for (unsigned int k = 0; k < N; k++) {

product += MO[i][k] \* MT4[k][j];

}

MA[i][j] = b4 \* product + a4 \* MS[i][j];

}

}

// 13.Signal T1 about Calculations2 finish

SetEvent(Event\_Calculation2EndIn4);

std::cout << "Thread " << threadIndex << " finished." << std::endl;

};

//-----------------------------------------------------------------------------

int main() {

std::cout << "Main started..." << std::endl;

// Preparations

fillMatrix(MA, 0); // TODO: can remove

a = 4294967295;

b = 0;

// Semaphores

Semaphore\_Calculations1EndIn1 = CreateSemaphore(

NULL, // default security attributes

0, // initial count

3, // maximum count

NULL // unnamed semaphore

);

Semaphore\_Calculations1EndIn2 = CreateSemaphore(NULL, 0, 3, NULL);

Semaphore\_Calculations1EndIn3 = CreateSemaphore(NULL, 0, 3, NULL);

Semaphore\_Calculations1EndIn4 = CreateSemaphore(NULL, 0, 3, NULL);

// Mutexes

Mutex\_SetA = CreateMutex(

NULL, // default security attributes

FALSE, // initially not owned

NULL // unnamed

);

Mutex\_SetB = CreateMutex(NULL, FALSE, NULL);

// Critical Sections

InitializeCriticalSection(&CriticalSection\_CopyMT);

InitializeCriticalSection(&CriticalSection\_CopyAb);

// Events

// Manual reset

Event\_InputFinishIn1 = CreateEvent(

NULL, // default security attibutes

TRUE, // manual reset

FALSE, // initial - not signaled

NULL // unnamed event

);

Event\_InputFinishIn3 = CreateEvent(NULL, TRUE, FALSE, NULL);

Event\_InputFinishIn4 = CreateEvent(NULL, TRUE, FALSE, NULL);

// Automatic reset

Event\_Calculation2EndIn2 = CreateEvent(NULL, FALSE, FALSE, NULL);

Event\_Calculation2EndIn3 = CreateEvent(NULL, FALSE, FALSE, NULL);

Event\_Calculation2EndIn4 = CreateEvent(NULL, FALSE, FALSE, NULL);

// Threads

DWORD Tid1, Tid2, Tid3, Tid4;

HANDLE Thread1 = CreateThread(

NULL, // Default secutiry attributes

STACK\_SIZE, // stack size

(LPTHREAD\_START\_ROUTINE) T1, // thread function name

NULL, // argument to thread function

0, // creating flags(0 => run immediately after creation)

&Tid1 // thread identifier

);

HANDLE Thread2 = CreateThread(NULL, STACK\_SIZE, (LPTHREAD\_START\_ROUTINE) T2, NULL, 0, &Tid2);

HANDLE Thread3 = CreateThread(NULL, STACK\_SIZE, (LPTHREAD\_START\_ROUTINE) T3, NULL, 0, &Tid3);

HANDLE Thread4 = CreateThread(NULL, STACK\_SIZE, (LPTHREAD\_START\_ROUTINE) T4, NULL, 0, &Tid4);

std::cout << "Main finished." << std::endl;

std::cin.get();

}

//-----------------------------------------------------------------------------

// Function definitions

void fillVector(struct Vector& Vector, unsigned int value) {

for (unsigned int i = 0; i < N; i++)

Vector[i] = value;

}

void fillMatrix(Matrix& matrix, unsigned int value) {

for (unsigned int i = 0; i < N; i++)

for (unsigned int j = 0; j < N; j++)

matrix[i][j] = value;

}

void outputMatrix(const Matrix& matrix) {

for (unsigned int i = 0; i < N; i++) {

const Vector& v = matrix[i];

for (unsigned int j = 0; j < N; j++) {

std::cout << v[i] << " ";

}

std::cout << std::endl;

}

}

unsigned int findMin(const Vector& Vector, unsigned int low, unsigned int high) {

unsigned int m = Vector[low];

for (unsigned int i = low; i < high; i++) {

const unsigned int current = Vector[i];

if (current < m) {

m = current;

}

}

return m;

}

unsigned int findMax(const Vector& Vector, unsigned int low, unsigned int high) {

unsigned int m = Vector[low];

for (unsigned int i = low; i < high; i++) {

const unsigned int current = Vector[i];

if (current > m) {

m = current;

}

}

return m;

}