Національний технічний університет України
«Київський політехнічний інститут»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Лабораторна робота №2

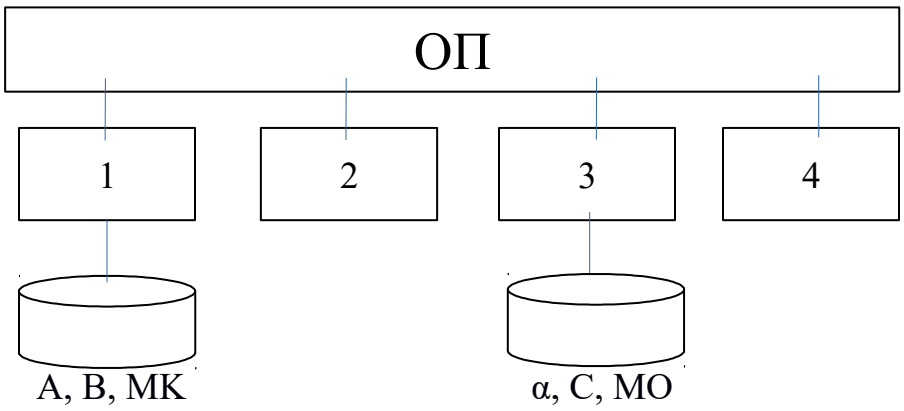
З предмету: «Паралельне програмування - 2»

Виконнав: студент групи ІП-32
Ковальчук Олександр Миронович

Київ 2016р.

Технічне завдання

1. Структура паралельної комп’ютерної системи з спільною пам’яттю:



2. Задача:
   $A = sort(\alpha \cdot B + C \cdot (MO \cdot MK))$
3. Мова програмування:
   C++, Win32
4. Засіб взаємодії задач:
   Семафори, мютекси, події, критичні секції.

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. $A_{2H} = merge\left(A_H, A_H\right)$
3. $A = merge\left(A_{2H}, A_{2H}\right)$

Спільні ресурси: a, C, MO

Етап 2. Розробка алгоритму процесів (задач)

	T1	**ТС / КД**
1.	Введення B, MK	
2.	Сигнал задачам T2, T3, T4 про введення MK, B	S2-1, S3-1, S4-1
3.	Очікувати введення в задачі T3	W3-1
4.	Копіювати MO1 = MO, α1= α, C1=C,	Кд
5.	Обчислення $A_H = sort\left(\alpha 1 \cdot B_H + C1 \cdot \left(MO1 \cdot MK_H\right)\right)$	
6.	Чекати на завершення сортування A_H в T2	W2-1
7.	Злиття $A_{2H} = merge\left(A_H, A_H\right)$	
8.	Чекати на завершення сортування A_{2H} в T3	W3-2
9.	Злиття $A = merge\left(A_{2H}, A_{2H}\right)$	
10.	Виведення результату A	

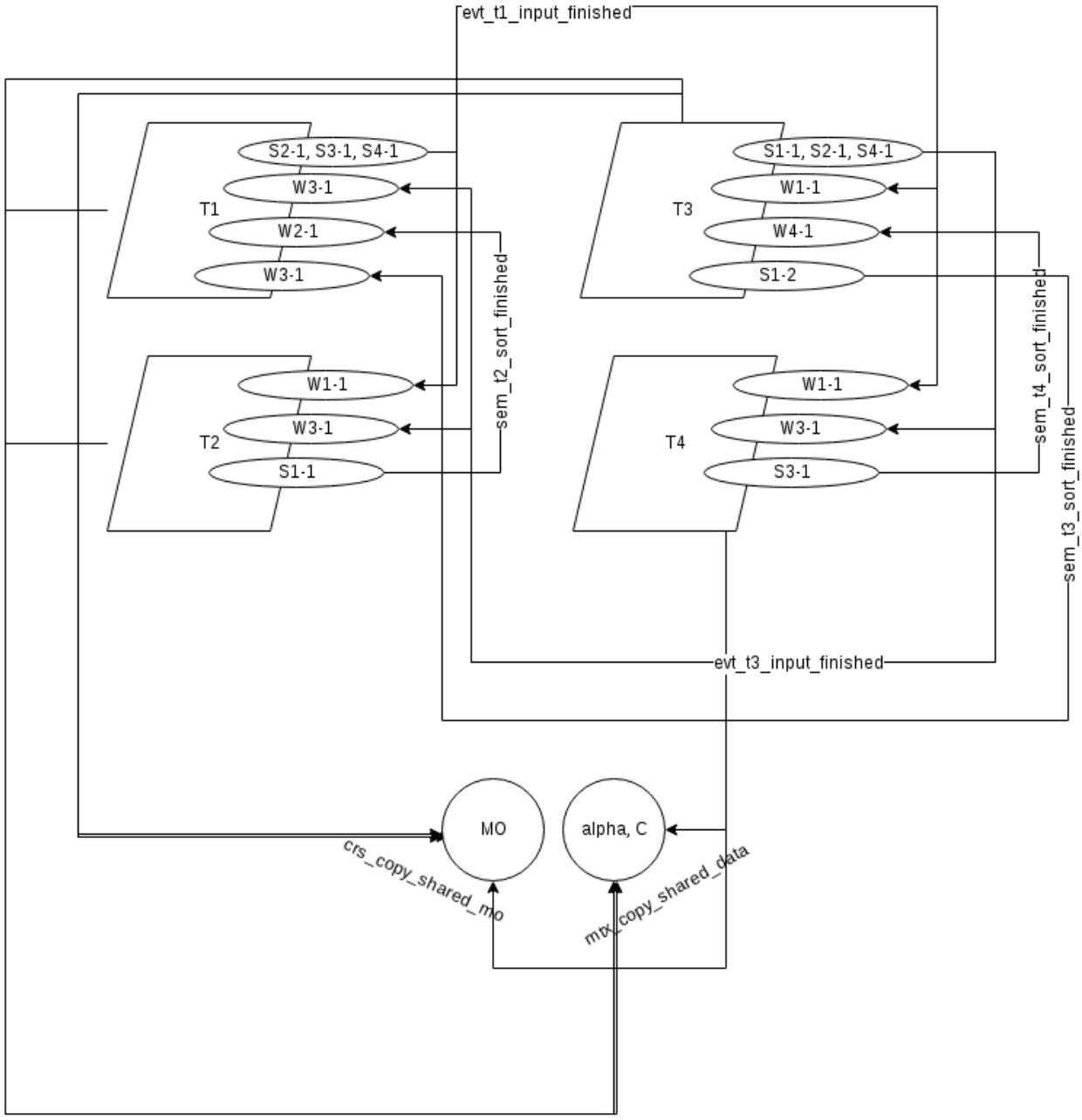
	T2	**ТС / КД**
1.	Чекати на введення даних у задачах T1, T3	W1-1, W3-1
2.	Копіювати MO2 = MO, α2= α, C2=C	КД
3.	Обчислення $A_H = sort\left(\alpha 2 \cdot B_H + C2 \cdot \left(MO2 \cdot MK_H\right)\right)$	
4.	Сигнал T1 про завершення обчислення A_H	S1-1

	T3	**ТС / КД**
1	Введення α, C, MO	

2	Сигнал T1, T2, T4 про введення α, C, MO	S1-1, S2-1, S4-1
3.	Чекати на введення даних у задачі T1	W1-1
4	Копіювати MO3 = MO, α3= α, C3=C	КД
5.	Обчислення $A_H = sort\left(\alpha 3 \cdot B_H + C3 \cdot \left(MO3 \cdot MK_H\right)\right)$	
6.	Чекати на завершення сортування A_H в T4	W4-1
7.	Злиття $A_{2H} = merge\left(A_H, A_H\right)$	
8.	Сигнал T1 про завершення обчислення A_{2H}	S1-2

	T4	**ТС / КД**
1.	Чекати на введення даних у задачах T1, T3	W1-1, W3-1
2.	Копіювати MO4 = MO, α4= α, C4=C	КД
3.	Обчислення $A_H = sort\left(\alpha 4 \cdot B_H + C4 \cdot \left(MO4 \cdot MK_H\right)\right)$	
4.	Сигнал T3 про завершення обчислення A_H	S3-1

Етап 3. Розробка структурної взаємодії задач



Лістинг

```
/*
 * Parallel Proramming 2
 * Lab 2
 * Win32. MTX, SEMA, CRS, EVT
 * A = sort(alpha * B + C * (MO * MK))
 *
 * @author Oleksandr Kovalchuk
 * @group IP-32
 */
#include <iostream>
#include <Windows.h>
#include <sstream>
#include <algorithm>

typedef int* vector;
typedef int** matrix;

void task1();
void task2();
void task3();
void task4();
vector inVect();
matrix inMatr();
vector copyVect(int*);
matrix copyMatr(int**);

const int N = 4;
const int P = 4;
const int H = N / P;

vector A = new int[N], B = new int[N], C = new int[N];
matrix MO = new vector[N], MK = new vector[N];
int alpha;

HANDLE mtx_copy_shared_data;
HANDLE evt_t1_input_finished;
HANDLE evt_t3_input_finished;
HANDLE sem_t2_sort_finished;
HANDLE sem_t3_sort_finished;
HANDLE sem_t4_sort_finished;
CRITICAL_SECTION crs_copy_shared_mo;


int main(int argc, char *argv[]) {
	std::cout << "Lab 02 started" << std::endl;

	// Prepare sync objs
	mtx_copy_shared_data = CreateMutex(NULL, FALSE, NULL);
	evt_t1_input_finished = CreateEvent(NULL, TRUE, FALSE, TEXT("EVT T1 FINISHED INPUT"));
	evt_t3_input_finished = CreateEvent(NULL, TRUE, FALSE, TEXT("EVT T2 FINISHED INPUT"));
	sem_t2_sort_finished = CreateSemaphore(NULL, 0, 1, TEXT("SEM T2 FINISHED_SORTING"));
	sem_t3_sort_finished = CreateSemaphore(NULL, 0, 1, TEXT("SEM T3 FINISHED_SORTING"));
	sem_t4_sort_finished = CreateSemaphore(NULL, 0, 1, TEXT("SEM T4 FINISHED_SORTING"));
	InitializeCriticalSection(&crs_copy_shared_mo);

	// Prepare threads
	HANDLE threads[4];
	threads[0] = CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)task1, NULL, CREATE_SUSPENDED, NULL);
	threads[1] = CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)task2, NULL, CREATE_SUSPENDED, NULL);
	threads[2] = CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)task3, NULL, CREATE_SUSPENDED, NULL);
	threads[3] = CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)task4, NULL, CREATE_SUSPENDED, NULL);

	for (int i = 0; i < 4; ++i) {
		ResumeThread(threads[i]);
	}

	WaitForMultipleObjects(4, threads, true, INFINITE);

	for (int i = 0; i < 4; ++i) {
		CloseHandle(threads[i]);
	}
	CloseHandle(mtx_copy_shared_data);
	CloseHandle(sem_t2_sort_finished);
	CloseHandle(sem_t3_sort_finished);
	CloseHandle(sem_t4_sort_finished);
	CloseHandle(evt_t1_input_finished);
	CloseHandle(evt_t3_input_finished);
	DeleteCriticalSection(&crs_copy_shared_mo);

	std::cout << "Lab 02 finished" << std::endl;
	std::cin.get();
	return 0;
}

void task1() {
	std::cout << "T1 started" << std::endl;

	// Input B, MK
	B = inVect();
	MK = inMatr();
	// Create resulting vector as well:
```

```
        // Signalize about input finished
        SetEvent(evt_t1_input_finished);

        // Wait for t3 finish input
        WaitForSingleObject(evt_t3_input_finished, INFINITE);

        // Copy shared data
        WaitForSingleObject(mtx_copy_shared_data, INFINITE);
        int alpha_1 = alpha;
        vector C_1 = copyVect(C);
        ReleaseMutex(mtx_copy_shared_data);

        EnterCriticalSection(&crs_copy_shared_mo);
        matrix MO_1 = copyMatr(MO);
        LeaveCriticalSection(&crs_copy_shared_mo);

        // Compute A_1_unsorted = alpha_1 * B + C * (MO * MK)
        for (int i = 0; i < H; ++i) {
                A[i] = 0;
                for (int j = 0; j < N; ++j) {
                        for (int k = 0; k < N; ++k) {
                                A[i] += C_1[j] * MO_1[j][k] * MK[k][i];
                        }
                }
                A[i] += alpha_1 * B[i];
        }

        // Sort partial result
        std::sort(A, A + H);

        // Wait for thread 2 to finish sorting and merge t1 t2
        WaitForSingleObject(sem_t2_sort_finished, INFINITE);
        std::inplace_merge(A, A + H, A + 2 * H);

        // Wait for thread 3 to finish sorting and merge t1 t3
        WaitForSingleObject(sem_t3_sort_finished, INFINITE);
        std::inplace_merge(A, A + 2 * H, A + N);

        // Output result
        if (N < 10) {
                std::stringstream ss;
                ss << "Result is: [ ";
                for (int i = 0; i < N; ++i) {
                        ss << A[i] << " ";
                }
                ss << "];" << std::endl;
                std::cout << ss.str() << std::endl;
        }

        std::cout << "T1 finished" << std::endl;
}


void task2() {
        std::cout << "T2 started" << std::endl;

        // Wait for T1, T3 to finish input
        WaitForSingleObject(evt_t1_input_finished, INFINITE);
        WaitForSingleObject(evt_t3_input_finished, INFINITE);

        // Copy shared data
        WaitForSingleObject(mtx_copy_shared_data, INFINITE);
        int alpha_2 = alpha;
        vector C_2 = copyVect(C);
        ReleaseMutex(mtx_copy_shared_data);

        EnterCriticalSection(&crs_copy_shared_mo);
        matrix MO_2 = copyMatr(MO);
        LeaveCriticalSection(&crs_copy_shared_mo);

        // Compute A_1_unsorted = alpha_1 * B + C * (MO * MK)
        for (int i = H; i < 2 * H; ++i) {
                (A)[i] = 0;
                for (int j = 0; j < N; ++j) {
                        for (int k = 0; k < N; ++k) {
                                A[i] += C_2[j] * MO_2[j][k] * MK[k][i];
                        }
                }
                A[i] += alpha_2 * B[i];
        }


        // Sort partial result
        std::sort(A + H, A + 2 * H);

        // Signalize T1 about sorting finished
        ReleaseSemaphore(sem_t2_sort_finished, 1, NULL);

        std::cout << "T2 finished" << std::endl;
}


void task3() {
```

```
        std::cout << "T3 started" << std::endl;

        // Input alpha, MO
        alpha = 1;
        MO = inMatr();
        C = inVect();

        // Signalize about input finished
        SetEvent(evt_t3_input_finished);
        WaitForSingleObject(evt_t1_input_finished, INFINITE);

        // Copy shared data
        WaitForSingleObject(mtx_copy_shared_data, INFINITE);
        int alpha_3 = alpha;
        vector C_3 = copyVect(C);
        ReleaseMutex(mtx_copy_shared_data);

        EnterCriticalSection(&crs_copy_shared_mo);
        matrix MO_3 = copyMatr(MO);
        LeaveCriticalSection(&crs_copy_shared_mo);

        // Compute A_3_unsorted = alpha_1 * B + C * (MO * MK)
        for (int i = 2 * H; i < 3 * H; ++i) {
                A[i] = 0;
                for (int j = 0; j < N; ++j) {
                        for (int k = 0; k < N; ++k) {
                                A[i] += C_3[j] * MO_3[j][k] * MK[k][i];
                        }
                }
                A[i] += alpha_3 * B[i];
        }

        // Sort partial result
        std::sort(A + 2 * H, A + 3 * H);

        // Wait for thread 4 to finish sorting and merge t3, t4
        WaitForSingleObject(sem_t4_sort_finished, INFINITE);
        std::inplace_merge(A + 2 * H, A + 3 * H, A + N);
        ReleaseSemaphore(sem_t3_sort_finished, 1, NULL);

        std::cout << "T3 finished" << std::endl;
}


void task4() {
        std::cout << "T4 started" << std::endl;

        // Wait for T1,T3 to finish input
        WaitForSingleObject(evt_t1_input_finished, INFINITE);
        WaitForSingleObject(evt_t3_input_finished, INFINITE);

        // Copy shared data
        WaitForSingleObject(mtx_copy_shared_data, INFINITE);
        int alpha_4 = alpha;
        vector C_4 = copyVect(C);
        ReleaseMutex(mtx_copy_shared_data);

        EnterCriticalSection(&crs_copy_shared_mo);
        matrix MO_4 = copyMatr(MO);
        LeaveCriticalSection(&crs_copy_shared_mo);

        // Compute A_4_unsorted = alpha_4 * B + C * (MO_4 * MK)
        for (int i = 2 * H; i < N; ++i) {
                A[i] = 0;
                for (int j = 0; j < N; ++j) {
                        for (int k = 0; k < N; ++k) {
                                A[i] += C_4[j] * MO_4[j][k] * MK[k][i];
                        }
                }
                A[i] += alpha_4 * B[i];
        }

        // Sort partial result
        std::sort(A + 3 * H, A + N);
        ReleaseSemaphore(sem_t4_sort_finished, 1, NULL);

        // Signalize T3 about sorting finished
        std::cout << "T4 finished" << std::endl;
}

vector inVect() {
        int* outVect = new int[N];
        for (int i = 0; i<N; i++) {
                outVect[i] = 1;
        }
        return outVect;
}

matrix inMatr() {
        int** outMatr = new int*[N];
        for (int i = 0; i < N; i++)
        {
                outMatr[i] = new int[N];
                for (int j = 0; j<N; j++) {
```

```
                        outMatr[i][j] = 1;
                }
        }
        return outMatr;
}

vector copyVect(int* vectToCopy) {
        int* outVector = new int[N];
        for (int i = 0; i < N; i++)
        {
                outVector[i] = vectToCopy[i];
        }
        return outVector;
}

matrix copyMatr(int** matrToCopy) {
        int** outMatrix = new int*[N];
        for (int i = 0; i<N; i++) {
                outMatrix[i] = new int[N];
                for (int j = 0; j < N; j++) {
                        outMatrix[i][j] = matrToCopy[i][j];
                }
        }
        return outMatrix;
}
```