МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ имени М.В.ЛОМОНОСОВА

**Практикум по курсу**

**"Распределенные системы"**

**Разработка параллельной версии программы для решения определенного интеграла методом Симпсона MPI**

**ОТЧЕТ**

**о выполненном задании**

Выполнил студент ПМ401 группы

Хаметов Марк Владиминович

edu-cmc-skpod22-313-7

Севастополь, 2024 г.

# Постановка задачи

16 процессов, находящихся в узлах транспьютерной матрицы размером 4\*4, одновременно выдали запрос на вход в критическую секцию.  
Реализовать программу, использующую централизованный алгоритм на транспьютерной матрице для прохождения всеми процессами критических секций.

Критическая секция:  
<проверка наличия файла “critical.txt”>;  
if (<файл “critical.txt” существует>) {  
<сообщение об ошибке>;  
<завершение работы программы>;  
} else {  
<создание файла “critical.txt”>;  
Sleep (<случайное время>);  
<уничтожение файла “critical.txt”>;  
}

Для межпроцессорных взаимодействий использовать средства MPI.

Получить временную оценку работы алгоритма. Оценить сколько времени потребуется для прохождения всеми критических секций, если координатор расположен в узле (0,0)? Время старта равно 100, время передачи байта равно 1 (Ts=100,Tb=1). Процессорные операции, включая чтение из памяти и запись в память, считаются бесконечно быстрыми.

# Описание

**Централизованный алгоритм**

Решение о доступе к критическим секциям принимается центральным управляющим элементом(координатор) (у нас с индексом 0)

В транспьютерной матрице, где процессоры расположены в виде двумерной сетки, центральный узел может быть физически расположенным в определенной точке этой сетки. Можно рассчитать номер процесса по номеру столбца и номеру строки, зная размер матрицы. Так как в условии (0,0), то это индекс координатора 0.

Алгоритм

1. Инициализация всех процессов сразу (координатора тоже).
2. Координатор слушает. Остальные пытаются запросить доступ в критическую секцию.
3. Координатор отправляет разрешение. Процесс i входит в крит. секцию
4. Крит секция для процесса i
5. Процесс i отправляет координатору сообщение о завершении. Координатор слушает.

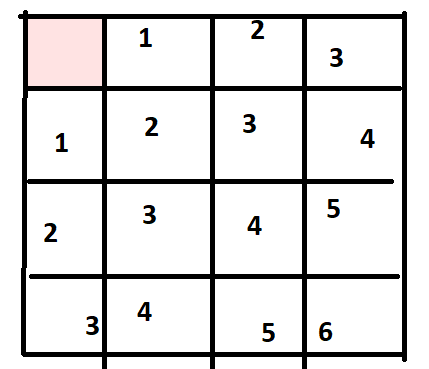
**Временная оценка алгоритма**

T = 100 + 3 \* (1+1+2+2+2+3+3+3+3+4+4+4+5+5+6) = 244

Начало из условия

До и от координатора до момента вхождения в крит секцию. До координатора после критсекции. (Сигналы всегда последовательны поэтому получаем тройное прохождение дистанции)

Кол-во шагов одного сигнала



# Код

#include <mpi.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <time.h>

#define **N** 4 // Размер матрицы

#define **M** 4

void **criticalSection**(int world\_rank) {

    if (world\_rank == 0) {  // =========================== координатор

        FILE \*file;

        const char \*filename = "critical.txt";

        file = **fopen**(filename, "r");

        if (file) {

            // Файл существует

**fclose**(file);

**printf**("Ошибка: файл %s уже существует.\n", filename);

**exit**(**EXIT\_FAILURE**);

        } else {

            // Файл не существует

            file = **fopen**(filename, "w");

            if (!file) {

**printf**("Ошибка при создании файла %s\n", filename);

**exit**(**EXIT\_FAILURE**);

            }

**fclose**(file);

**srand**(**time**(**NULL**));

            int sleepTime = **rand**() % 4;

**printf**("Спим на %d секунд\n", sleepTime);

**sleep**(sleepTime);

            if (**remove**(filename) != 0) {

**printf**("Ошибка при удалении файла %s\n", filename);

**exit**(**EXIT\_FAILURE**);

            }

        }

    } else { // ======================================================= не координатор

        // сигнал процесс координатору

**MPI\_Send**(**NULL**, 0, MPI\_INT, 0, 0, MPI\_COMM\_WORLD);

        // ждем обратный сигнал

**MPI\_Recv**(**NULL**, 0, MPI\_INT, 0, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

        int row = world\_rank / **N**;

        int column = world\_rank % **M**;

**printf**("[%d,%d] Критическая секция  %d успешно выполнена.\n",row,column, world\_rank);

    }

}

int **main**(int argc, char\*\* argv) {

**MPI\_Init**(**NULL**, **NULL**);

    int world\_rank;

**MPI\_Comm\_rank**(MPI\_COMM\_WORLD, &world\_rank);

    int world\_size;

**MPI\_Comm\_size**(MPI\_COMM\_WORLD, &world\_size);

    if (world\_size != 16) {

**printf**("Запущено %d процессов. Ожидалось 16 процессов\n", world\_size);

**printf**("Попробуйте mpirun -np 16 --oversubscribe ./a.out");

**MPI\_Finalize**();

            return **EXIT\_FAILURE**;

            }

    if (world\_rank == 0) { // =========================== координатор

        for (int i = 1; i < world\_size; i++) {

**MPI\_Recv**(**NULL**, 0, MPI\_INT, i, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE); //ждем сигнал от других процессов

**MPI\_Send**(**NULL**, 0, MPI\_INT, i, 0, MPI\_COMM\_WORLD); // координатор отправляет сигнал другому процессу

        }

    } else { // ======================================================= не координатор

**criticalSection**(world\_rank);

    }

**MPI\_Finalize**();

}