Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

Лабораторная работа №2 по курсу

«Операционные системы»

Группа: М8О-209Б-24

Студент: Филь Н.А.

Преподаватель: Миронов Е.С.

Оценка: _____

Дата: 10.11.25

Москва, 2025

Постановка задачи

Вариант 19.

Реализация многопоточной программы для кластеризации методом k-средний с ограничением максимального количества потоков.

Общий метод и алгоритм решения

В лабораторной работе использованы системные вызовы POSIX:

- int pthread_create(...) – создание нового потока;
- int pthread_join(...) – ожидание завершения потока;
- int sem_wait(...), sem_post(...) – ограничение числа активных потоков;
- int pthread_mutex_lock(...), pthread_mutex_unlock(...) – синхронизация общих данных;
- execve(), clone(), futex() – низкоуровневые системные вызовы, фиксируемые в strace при запуске программы.

Алгоритм:

В лабораторной работе реализована многопоточная обработка данных с использованием POSIX-потоков (pthread). Для ограничения числа активных потоков используется семафор. Синхронизация флага изменения кластеров осуществляется через мьютекс. Каждый поток обрабатывает часть массива точек, вычисляя ближайший кластер. После завершения всех потоков пересчитываются центры кластеров, и процесс повторяется до достижения сходимости.

Код программы

lab2.cpp

```
#include <iostream>
#include <vector>
#include <cmath>
#include <pthread.h>
#include <semaphore.h>
#include <chrono>
#include <random>
#include <mutex>

using namespace std;

struct Point {
    double x, y;
    int cluster;
};

struct Cluster {
    double x, y;
};

vector<Point> points;
vector<Cluster> clusters;
int k;
int maxThreads;
bool changed = true;

sem_t thread_sem;
pthread_mutex_t mutex_sum;

double distance(const Point &a, const Cluster &b) {
    return sqrt(pow(a.x - b.x, 2) + pow(a.y - b.y, 2));
```

```
}

void* assign_points(void* arg) {
    long id = (long)arg;
    size_t total_points = points.size();
    size_t points_per_thread = total_points / maxThreads;

    size_t start = id * points_per_thread;
    size_t end = (id == maxThreads - 1) ? total_points : start + points_per_thread;

    bool local_changed = false;

    for (size_t i = start; i < end; ++i) {
        double min_dist = 1e9;
        int cluster_index = 0;

        for (int j = 0; j < k; ++j) {
            double dist = distance(points[i], clusters[j]);
            if (dist < min_dist) {
                min_dist = dist;
                cluster_index = j;
            }
        }

        if (points[i].cluster != cluster_index) {
            points[i].cluster = cluster_index;
            local_changed = true;
        }
    }

    if (local_changed) {
        pthread_mutex_lock(&mutex_sum);
        changed = true;
        pthread_mutex_unlock(&mutex_sum);
    }

    sem_post(&thread_sem);
    return nullptr;
}

void update_clusters() {
    vector<double> sum_x(k, 0.0), sum_y(k, 0.0);
    vector<int> count(k, 0);

    for (auto &p : points) {
        sum_x[p.cluster] += p.x;
        sum_y[p.cluster] += p.y;
        count[p.cluster]++;
    }

    for (int i = 0; i < k; ++i) {
        if (count[i] > 0) {
            clusters[i].x = sum_x[i] / count[i];
            clusters[i].y = sum_y[i] / count[i];
        }
    }
}

int main(int argc, char* argv[]) {
    if (argc < 2) {
        cout << "Использование: ./lab4 <макс_потоков>\n";
        return 1;
    }

    maxThreads = stoi(argv[1]);
    cout << "Введите число кластеров: ";
    cin >> k;

    int n;
    cout << "Введите количество точек: ";
    cin >> n;

    random_device rd;
    mt19937 gen(rd());
```

```
    uniform_real_distribution<> dis(0, 100);

    points.resize(n);
    for (auto &p : points) {
        p.x = dis(gen);
        p.y = dis(gen);
        p.cluster = rand() % k;
    }

    clusters.resize(k);
    for (auto &c : clusters) {
        c.x = dis(gen);
        c.y = dis(gen);
    }

    sem_init(&thread_sem, 0, maxThreads);
    pthread_mutex_init(&mutex_sum, nullptr);

    auto start_time = chrono::high_resolution_clock::now();

    while (changed) {
        changed = false;

        vector<pthread_t> threads(maxThreads);

        for (int i = 0; i < maxThreads; ++i) {
            sem_wait(&thread_sem);
            pthread_create(&threads[i], nullptr, assign_points, (void*)(long)i);
        }

        for (int i = 0; i < maxThreads; ++i) {
            pthread_join(threads[i], nullptr);
        }

        update_clusters();
    }

    auto end_time = chrono::high_resolution_clock::now();
    auto duration = chrono::duration_cast<chrono::milliseconds>(end_time - start_time).count();

    cout << "\nАлгоритм завершён.\nВремя выполнения: " << duration << " мс\n";

    sem_destroy(&thread_sem);
    pthread_mutex_destroy(&mutex_sum);
    return 0;
}
```

Протокол работы программы

Тестирование:

```
n1k@n1k-vim:~/Documents$ g++ -pthread -std=c++17 lab2.cpp -o lab2
n1k@n1k-vim:~/Documents$ ./lab2 4
Введите число кластеров: 3
Введите количество точек: 1000

Алгоритм завершён.
Время выполнения: 64 мс
```

Strace:

Фрагмент вывода strace (ключевые места):

1. Запуск программы:

```
3527  execve("./lab2", ["./lab2", "4"], 0xfffff832c9f0 /* 75 vars */) = 0
```

Системный вызов execve() используется ядром Linux для запуска исполняемого файла с аргументами.

2. Создание потоков

```
3527  clone(child_stack=..., flags=CLONE_VM|CLONE_FS|CLONE_FILES|
CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|
CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, parent_tid=[3552], ...) = 3552
3527  clone(child_stack=..., flags=CLONE_VM|CLONE_THREAD|CLONE_SIGHAND|...) = 3553
3527  clone(child_stack=..., flags=CLONE_VM|CLONE_THREAD|CLONE_SIGHAND|...) = 3554
```

Системный вызов clone() — это низкоуровневая реализация создания потоков в Linux.
Флаг CLONE_THREAD указывает, что создаются **POSIX-потоки**, а не отдельные процессы.
Каждый вызов pthread_create() в коде соответствует системному вызову clone() в логе strace.

3. Синхронизация потоков

```
3527  futex(0xffff9e816290, FUTEX_WAIT, 3571, NULL)
3527  futex(0xffff9f017290, FUTEX_WAIT, 3570, NULL)
```

Системный вызов futex() используется библиотекой pthread для реализации механизмов синхронизации, таких как:

- ожидание завершения потоков (pthread_join);
- блокировка мьютексов (pthread_mutex_lock);
- работа семафоров (sem_wait, sem_post).

Появление вызова futex подтверждает, что синхронизация потоков выполняется корректно и с использованием POSIX-примитивов.

4. Работа с памятью и вводом/выводом

```
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0)
write(1, "Алгоритм завершён.\n", 20) = 20
```

mmap() — выделяет память под стеки потоков и другие структуры данных программы.
write() — выполняет системный вывод информации в стандартный поток вывода.

5. Завершение потоков

```
3552  exit(0)
3553  +++ exited with 0 +++
```

Каждый поток завершается системным вызовом exit(0), что свидетельствует о корректном завершении без ошибок.

Анализ

Число потоков	Время исполнения (мс)	Ускорение	Эффективность
1	16874	1,00	1,000
2	8300	2,03	1,016
3	4075	4,14	1,380
4	3624	4,66	1,164
5	2683	6,29	1,258
6	2381	7,09	1,181

Для измерений использовано разное число потоков, 20 кластеров и 200 тыс. точек

Расчеты:

- **2 потока:** Ускорение = 16874/8300 = 2,03 | Эффективность = 2,03/2 = 1,016
- **3 потока:** Ускорение = 16874/4075 = 4,14 | Эффективность = 4,14/3 = 1,380
- **4 потока:** Ускорение = 16874/3624 = 4,66 | Эффективность = 4,66/4 = 1,164
- **5 потоков:** Ускорение = 16874/2683 = 6,29 | Эффективность = 6,29/5 = 1,258
- **6 потоков:** Ускорение = 16874/2381 = 7,09 | Эффективность = 7,09/6 = 1,181

Основные наблюдения:

1. Высокое ускорение параллельного алгоритма
    - **На 6 потоках достигнуто ускорение** 7,09x **при времени выполнения 2381 мс против 16874 мс на одном потоке**
    - **Это свидетельствует об эффективной параллелизации алгоритма k-средних**
2. Сверхлинейное ускорение
    - **Эффективность превышает 1,0 на всех конфигурациях (максимум 1,38 на 3 потоках)**
    - **Такое явление обычно связано с:**
        - Кэш-эффектами **- данные распределяются между кэшами разных ядер**
        - Улучшением доступа к памяти **в многопоточной среде**
        - Оптимизациями процессора **при параллельном выполнении**
3. Неравномерный рост производительности
    - **Наибольший прирост между 2 и 3 потоками (удвоение ускорения)**
    - **После 4 потоков рост замедляется, но продолжается до 6 потоков**

Вывод

В ходе выполнения лабораторной работы была реализована программа кластеризации методом k-средних с использованием POSIX потоков. Создание и синхронизация потоков подтверждены анализом вывода утилиты strace. Были изучены принципы работы системных вызовов clone(), futex() и механизмов семафоров и мьютексов. Программа показала корректную работу и позволяет исследовать ускорение при увеличении количества потоков.