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-215Б-23

Студент: Агафонов А.С.

Преподаватель: Миронов Е.С.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: 31.10.24

Москва, 2024

**Постановка задачи**

**Вариант 5.**

Отсортировать массив целых чисел при помощи четно-нечетной сортировки Бетчера.

**Общий метод и алгоритм решения**

Использованные системные вызовы:

* ifstream infile(input\_filename); - открытие файла для чтения входных данных.
* ofstream outfile(output\_filename); - открытие файла для записи выходных данных.
* pthread\_t - создание потока с использованием POSIX-потоков для параллельной сортировки.
* pthread\_create(&thread, nullptr, threadFunc, args); - создание нового потока и запуск функции сортировки threadFunc.
* pthread\_join(thread, nullptr); - ожидание завершения выполнения созданного потока.
* pthread\_mutex\_init(&thread\_count\_mutex, nullptr); - инициализация мьютекса для синхронизации доступа к глобальной переменной current\_threads.
* pthread\_mutex\_lock(&thread\_count\_mutex); - блокировка мьютекса для безопасного доступа к переменной current\_threads.
* pthread\_mutex\_unlock(&thread\_count\_mutex); - разблокировка мьютекса после завершения доступа к переменной current\_threads.
* pthread\_mutex\_destroy(&thread\_count\_mutex); - уничтожение мьютекса после завершения работы программы.

В данном коде реализован метод параллельной сортировки массива с использованием **Batcher Odd-Even Merge Sort** (сортировка Батчера методом "нечетно-четное слияние"). Эта сортировка особенно эффективна для параллельных вычислений, поскольку процесс можно разделить на независимые этапы, которые могут выполняться параллельно. Для решения задачи используется механизм потоков (**threads**), что позволяет ускорить сортировку за счёт одновременной обработки различных частей массива.

#### Общий метод решения:

1. **Разделение массива на подмассивы**: Исходный массив делится на две части (левую и правую) рекурсивно до тех пор, пока размер подмассива не станет равен одному элементу. Это соответствует типичному подходу "разделяй и властвуй", где каждую подзадачу можно решать независимо.
2. **Параллельная обработка**:
   * Для повышения эффективности применяется механизм потоков. Потоки создаются для выполнения сортировки левой и правой половины массива.
   * Ограничение на количество потоков задаётся параметром MAX\_THREADS, чтобы избежать избыточного использования ресурсов и не допустить перегрузки системы.
   * Мьютекс (**mutex**) используется для контроля количества активных потоков. Это позволяет избежать создания потоков, если текущее число потоков достигает MAX\_THREADS.
3. **Слияние отсортированных подмассивов**: После сортировки левой и правой части выполняется **слияние методом нечетно-четной сортировки**. Это ключевая особенность алгоритма Батчера, который выполняет попарное сравнение и обмен значений в массивах на заданных расстояниях (например, на половине размера), а затем рекурсивно сливает элементы с уменьшенными интервалами.

**Код программы**

#include <iostream>

#include <vector>

#include <pthread.h>

#include <cstdlib>

#include <fstream>

#include <getopt.h>

#include <algorithm>

#include <cmath>

using namespace std;

// Структура для передачи аргументов в потоки

struct ThreadArgs {

    vector<int>\* data;

    int left;

    int right;

    int level;

};

// Глобальные переменные

int MAX\_THREADS;

int current\_threads = 0;

pthread\_mutex\_t thread\_count\_mutex;

// Прототипы функций

void batcherOddEvenMergeSort(vector<int>& data, int left, int right, int level);

void batcherOddEvenMerge(vector<int>& data, int left, int right, int r);

void compareExchange(vector<int>& data, int i, int j);

void\* threadFunc(void\* args);

// Опции командной строки

string input\_filename;

string output\_filename;

// Функция для чтения данных из файла или стандартного ввода

bool readData(vector<int>& data, istream& in) {

    int n;

    if (!(in >> n)) {

        cerr << "Не удалось прочитать количество элементов." << endl;

        return false;

    }

    data.resize(n);

    for (int i = 0; i < n; ++i) {

        if (!(in >> data[i])) {

            cerr << "Не удалось прочитать элемент " << i << "." << endl;

            return false;

        }

    }

    return true;

}

// Функция для записи данных в файл или стандартный вывод

void writeData(const vector<int>& data, ostream& out) {

    out << data.size() << endl;

    for (size\_t i = 0; i < data.size(); ++i) {

        out << data[i] << (i + 1 < data.size() ? ' ' : '\n');

    }

}

// Главная функция

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

    // Значения по умолчанию

    MAX\_THREADS = 1;

    // Разбор опций командной строки

    int option;

    while ((option = getopt(argc, argv, "t:i:o:")) != -1) {

        switch (option) {

            case 't':

                MAX\_THREADS = atoi(optarg);

                break;

            case 'i':

                input\_filename = optarg;

                break;

            case 'o':

                output\_filename = optarg;

                break;

            default:

                cerr << "Использование: " << argv[0] << " [-t max\_threads] [-i input\_file] [-o output\_file]" << endl;

                return 1;

        }

    }

    // Инициализация мьютекса

    pthread\_mutex\_init(&thread\_count\_mutex, nullptr);

    // Чтение входных данных

    vector<int> data;

    if (!input\_filename.empty()) {

        ifstream infile(input\_filename);

        if (!infile) {

            cerr << "Не удалось открыть входной файл: " << input\_filename << endl;

            return 1;

        }

        if (!readData(data, infile)) {

            return 1;

        }

        infile.close();

    } else {

        cout << "Введите количество элементов, а затем сами элементы:" << endl;

        if (!readData(data, cin)) {

            return 1;

        }

    }

    // Запуск сортировки

    batcherOddEvenMergeSort(data, 0, data.size() - 1, 0);

    // Запись отсортированных данных

    if (!output\_filename.empty()) {

        ofstream outfile(output\_filename);

        if (!outfile) {

            cerr << "Не удалось открыть выходной файл: " << output\_filename << endl;

            return 1;

        }

        writeData(data, outfile);

        outfile.close();

    } else {

        cout << "Отсортированный массив:" << endl;

        writeData(data, cout);

    }

    // Уничтожение мьютекса

    pthread\_mutex\_destroy(&thread\_count\_mutex);

    return 0;

}

// Функция сортировки Батчера "нечетно-четное слияние"

void batcherOddEvenMergeSort(vector<int>& data, int left, int right, int level) {

    int size = right - left + 1;

    if (size <= 1) return;

    int mid = (left + right) / 2;

    bool spawn\_thread = false;

    pthread\_mutex\_lock(&thread\_count\_mutex);

    if (current\_threads < MAX\_THREADS) {

        spawn\_thread = true;

        current\_threads++;

    }

    pthread\_mutex\_unlock(&thread\_count\_mutex);

    if (spawn\_thread) {

        // Создание потока для левой половины

        pthread\_t thread;

        ThreadArgs\* args = new ThreadArgs{&data, left, mid, level + 1};

        pthread\_create(&thread, nullptr, threadFunc, args);

        // Сортировка правой половины в текущем потоке

        batcherOddEvenMergeSort(data, mid + 1, right, level + 1);

        // Ожидание завершения сортировки левой половины

        pthread\_join(thread, nullptr);

        pthread\_mutex\_lock(&thread\_count\_mutex);

        current\_threads--;

        pthread\_mutex\_unlock(&thread\_count\_mutex);

        delete args;

    } else {

        // Сортировка обеих половин в текущем потоке

        batcherOddEvenMergeSort(data, left, mid, level + 1);

        batcherOddEvenMergeSort(data, mid + 1, right, level + 1);

    }

    // Слияние двух половин

    batcherOddEvenMerge(data, left, right, right - left + 1);

}

// Функция "нечетно-четное слияние" Батчера

void batcherOddEvenMerge(vector<int>& data, int left, int right, int size) {

    if (size <= 1) return;

    int mid = size / 2;

    // Сравнение и обмен элементов с расстоянием mid

    for (int i = left; i + mid <= right; ++i) {

        compareExchange(data, i, i + mid);

    }

    // Рекурсивное слияние двух половин

    batcherOddEvenMerge(data, left, left + mid - 1, mid);

    batcherOddEvenMerge(data, left + mid, right, mid);

}

// Функция сравнения и обмена

void compareExchange(vector<int>& data, int i, int j) {

    if (j >= data.size()) return; // Избегаем выхода за пределы массива

    if (data[i] > data[j]) {

        swap(data[i], data[j]);

    }

}

// Функция потока

void\* threadFunc(void\* args) {

    ThreadArgs\* arg = static\_cast<ThreadArgs\*>(args);

    batcherOddEvenMergeSort(\*arg->data, arg->left, arg->right, arg->level);

    return nullptr;

}

**Протокол работы программы**

**Тестирование:**

time ./sort -t 3 -i test.txt -o output4.txt

real 0m14.773s

user 0m32.154s

sys 0m0.322s

**Демонстрация количества потоков:**

root@bdc60f1003f5:/workspaces/os\_base# ps -o nlwp 5508

NLWP

4

**Strace:**

execve("./sort", ["./sort", "-t", "2", "-i", "test.txt", "-o", "output4.txt"], 0x7ffe92931d98 /\* 30 vars \*/) = 0

brk(NULL) = 0x1445000

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7fe07f0fa000

access("/etc/ld.so.preload", R\_OK) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "/etc/ld.so.cache", O\_RDONLY|O\_CLOEXEC) = 3

newfstatat(3, "", {st\_mode=S\_IFREG|0644, st\_size=25258, ...}, AT\_EMPTY\_PATH) = 0

mmap(NULL, 25258, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7fe07f0f3000

close(3) = 0

openat(AT\_FDCWD, "/usr/local/lib64/libstdc++.so.6", O\_RDONLY|O\_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0\0"..., 832) = 832

newfstatat(3, "", {st\_mode=S\_IFREG|0755, st\_size=2530008, ...}, AT\_EMPTY\_PATH) = 0

mmap(NULL, 2543808, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7fe07ee85000

mmap(0x7fe07ef2a000, 1216512, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0xa5000) = 0x7fe07ef2a000

mmap(0x7fe07f053000, 581632, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1ce000) = 0x7fe07f053000

mmap(0x7fe07f0e1000, 57344, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x25c000) = 0x7fe07f0e1000

mmap(0x7fe07f0ef000, 12480, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7fe07f0ef000

close(3) = 0

openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libm.so.6", O\_RDONLY|O\_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0\0"..., 832) = 832

newfstatat(3, "", {st\_mode=S\_IFREG|0644, st\_size=907784, ...}, AT\_EMPTY\_PATH) = 0

mmap(NULL, 909560, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7fe07eda6000

mmap(0x7fe07edb6000, 471040, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x10000) = 0x7fe07edb6000

mmap(0x7fe07ee29000, 368640, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x83000) = 0x7fe07ee29000

mmap(0x7fe07ee83000, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0xdc000) = 0x7fe07ee83000

close(3) = 0

openat(AT\_FDCWD, "/usr/local/lib64/libgcc\_s.so.1", O\_RDONLY|O\_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0\0"..., 832) = 832

newfstatat(3, "", {st\_mode=S\_IFREG|0644, st\_size=906528, ...}, AT\_EMPTY\_PATH) = 0

mmap(NULL, 181160, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7fe07ed79000

mmap(0x7fe07ed7d000, 143360, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x4000) = 0x7fe07ed7d000

mmap(0x7fe07eda0000, 16384, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x27000) = 0x7fe07eda0000

mmap(0x7fe07eda4000, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x2b000) = 0x7fe07eda4000

close(3) = 0

openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libc.so.6", O\_RDONLY|O\_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\20t\2\0\0\0\0\0"..., 832) = 832

pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784

newfstatat(3, "", {st\_mode=S\_IFREG|0755, st\_size=1922136, ...}, AT\_EMPTY\_PATH) = 0

pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784

mmap(NULL, 1970000, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7fe07eb98000

mmap(0x7fe07ebbe000, 1396736, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x26000) = 0x7fe07ebbe000

mmap(0x7fe07ed13000, 339968, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x17b000) = 0x7fe07ed13000

mmap(0x7fe07ed66000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1ce000) = 0x7fe07ed66000

mmap(0x7fe07ed6c000, 53072, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7fe07ed6c000

close(3) = 0

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7fe07eb96000

arch\_prctl(ARCH\_SET\_FS, 0x7fe07eb97480) = 0

set\_tid\_address(0x7fe07eb97750) = 6212

set\_robust\_list(0x7fe07eb97760, 24) = 0

rseq(0x7fe07eb97da0, 0x20, 0, 0x53053053) = 0

mprotect(0x7fe07ed66000, 16384, PROT\_READ) = 0

mprotect(0x7fe07eda4000, 4096, PROT\_READ) = 0

mprotect(0x7fe07ee83000, 4096, PROT\_READ) = 0

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7fe07eb94000

mprotect(0x7fe07f0e1000, 45056, PROT\_READ) = 0

mprotect(0x407000, 4096, PROT\_READ) = 0

mprotect(0x7fe07f12c000, 8192, PROT\_READ) = 0

prlimit64(0, RLIMIT\_STACK, NULL, {rlim\_cur=8192\*1024, rlim\_max=RLIM64\_INFINITY}) = 0

munmap(0x7fe07f0f3000, 25258) = 0

futex(0x7fe07f0ef73c, FUTEX\_WAKE\_PRIVATE, 2147483647) = 0

getrandom("\x00\x18\xb3\xc6\x50\xfc\x00\xfe", 8, GRND\_NONBLOCK) = 8

brk(NULL) = 0x1445000

brk(0x1466000) = 0x1466000

openat(AT\_FDCWD, "test.txt", O\_RDONLY) = 3

read(3, "4\n5310515 1933384 7154451 698388"..., 8191) = 35

close(3) = 0

rt\_sigaction(SIGRT\_1, {sa\_handler=0x7fe07ec1e6a0, sa\_mask=[], sa\_flags=SA\_RESTORER|SA\_ONSTACK|SA\_RESTART|SA\_SIGINFO, sa\_restorer=0x7fe07ebd4050}, NULL, 8) = 0

rt\_sigprocmask(SIG\_UNBLOCK, [RTMIN RT\_1], NULL, 8) = 0

mmap(NULL, 8392704, PROT\_NONE, MAP\_PRIVATE|MAP\_ANONYMOUS|MAP\_STACK, -1, 0) = 0x7fe07e393000

mprotect(0x7fe07e394000, 8388608, PROT\_READ|PROT\_WRITE) = 0

rt\_sigprocmask(SIG\_BLOCK, ~[], [], 8) = 0

clone3({flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, child\_tid=0x7fe07eb93990, parent\_tid=0x7fe07eb93990, exit\_signal=0, stack=0x7fe07e393000, stack\_size=0x7fff80, tls=0x7fe07eb936c0}strace: Process 6213 attached

=> {parent\_tid=[6213]}, 88) = 6213

[pid 6213] rseq(0x7fe07eb93fe0, 0x20, 0, 0x53053053 <unfinished ...>

[pid 6212] rt\_sigprocmask(SIG\_SETMASK, [], <unfinished ...>

[pid 6213] <... rseq resumed>) = 0

[pid 6212] <... rt\_sigprocmask resumed>NULL, 8) = 0

[pid 6213] set\_robust\_list(0x7fe07eb939a0, 24 <unfinished ...>

[pid 6212] mmap(NULL, 8392704, PROT\_NONE, MAP\_PRIVATE|MAP\_ANONYMOUS|MAP\_STACK, -1, 0 <unfinished ...>

[pid 6213] <... set\_robust\_list resumed>) = 0

[pid 6212] <... mmap resumed>) = 0x7fe07db92000

[pid 6213] rt\_sigprocmask(SIG\_SETMASK, [], <unfinished ...>

[pid 6212] mprotect(0x7fe07db93000, 8388608, PROT\_READ|PROT\_WRITE <unfinished ...>

[pid 6213] <... rt\_sigprocmask resumed>NULL, 8) = 0

[pid 6212] <... mprotect resumed>) = 0

[pid 6213] rt\_sigprocmask(SIG\_BLOCK, ~[RT\_1], <unfinished ...>

[pid 6212] rt\_sigprocmask(SIG\_BLOCK, ~[], <unfinished ...>

[pid 6213] <... rt\_sigprocmask resumed>NULL, 8) = 0

[pid 6212] <... rt\_sigprocmask resumed>[], 8) = 0

[pid 6213] madvise(0x7fe07e393000, 8368128, MADV\_DONTNEED <unfinished ...>

[pid 6212] clone3({flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, child\_tid=0x7fe07e392990, parent\_tid=0x7fe07e392990, exit\_signal=0, stack=0x7fe07db92000, stack\_size=0x7fff80, tls=0x7fe07e3926c0} <unfinished ...>

[pid 6213] <... madvise resumed>) = 0

[pid 6213] exit(0strace: Process 6214 attached

) = ?

[pid 6212] <... clone3 resumed> => {parent\_tid=[6214]}, 88) = 6214

[pid 6214] rseq(0x7fe07e392fe0, 0x20, 0, 0x53053053 <unfinished ...>

[pid 6212] rt\_sigprocmask(SIG\_SETMASK, [], <unfinished ...>

[pid 6213] +++ exited with 0 +++

[pid 6214] <... rseq resumed>) = 0

[pid 6212] <... rt\_sigprocmask resumed>NULL, 8) = 0

[pid 6214] set\_robust\_list(0x7fe07e3929a0, 24 <unfinished ...>

[pid 6212] futex(0x7fe07e392990, FUTEX\_WAIT\_BITSET|FUTEX\_CLOCK\_REALTIME, 6214, NULL, FUTEX\_BITSET\_MATCH\_ANY <unfinished ...>

[pid 6214] <... set\_robust\_list resumed>) = 0

[pid 6214] rt\_sigprocmask(SIG\_SETMASK, [], NULL, 8) = 0

[pid 6214] rt\_sigprocmask(SIG\_BLOCK, ~[RT\_1], NULL, 8) = 0

[pid 6214] madvise(0x7fe07db92000, 8368128, MADV\_DONTNEED) = 0

[pid 6214] exit(0) = ?

[pid 6212] <... futex resumed>) = 0

[pid 6214] +++ exited with 0 +++

futex(0x7fe07f0ef748, FUTEX\_WAKE\_PRIVATE, 2147483647) = 0

openat(AT\_FDCWD, "output4.txt", O\_WRONLY|O\_CREAT|O\_TRUNC, 0666) = 3

write(3, "4\n", 2) = 2

write(3, "1933384 5310515 6983887 7154451\n", 32) = 32

close(3) = 0

exit\_group(0) = ?

+++ exited with 0 +++

Массив 4194304.

|  |  |  |  |
| --- | --- | --- | --- |
| Число потоков | Время исполнения (мс) | Ускорение | Эффективность |
| 1 | 4525 | 1 | 1 |
| 2 | 3904 | 1.159 | 0.58 |
| 3 | 3232 | 1.4 | 0.467 |
| 4 | 3224 | 1.403 | 0.0351 |
| 5 | 3087 | 1.466 | 0.292 |
| 6 | 3060 | 1.479 | 0.245 |
| 7 | 2955 | 1.531 | 0.219 |
| 8 | 3031 | 1.493 | 0.187 |

**Ускорение** показывает во сколько раз применение параллельного алгоритма уменьшает время решения задачи по сравнению с последовательным алгоритмом. Ускорение определяется величиной SN=T1/TN, где Т1 - время выполнения на одном потоке, TN - время выполнения на N потоках.

**Эффективность** - величина EN = SN/N, где SN - ускорение, N - количество используемых потоков.

**Вывод**

На основе тестирования программы с разным количеством потоков и объемом данных можно сделать следующие выводы:

1. Многопоточность значительно ускоряет выполнение программы при грамотном распределении нагрузки между потоками.
2. Для максимального ускорения следует выбирать оптимальное количество потоков, которое соответствует вычислительным возможностям компьютера (например, числу ядер процессора) и объему задачи.
3. Избыточное количество потоков может снижать эффективность работы из-за накладных расходов на управление потоками и синхронизацию.

Таким образом, многопоточность является эффективным инструментом для повышения производительности, если её правильно применять в зависимости от аппаратных характеристик и сложности задачи.