Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №2 по курсу**

**«Операционные системы»**

**УПРАВЛЕНИЕ ПОТОКАМИ В ОС**

Студент: Юрков Евгений Юрьевич

Группа: М8О–212Б–22

Вариант: 6

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2023.

**Постановка задачи**

## Цель работы

Целью является приобретение практических навыков в:

* Управление потоками в ОС
* Обеспечение синхронизации между потоками

## Задание

Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При обработке использовать стандартные средства создания потоков операционной системы Unix. Ограничение максимального количества потоков, работающих в один момент времени, должно быть задано ключом запуска вашей программы.

Так же необходимо уметь продемонстрировать количество потоков, используемое вашей программой с помощью стандартных средств операционной системы.

В отчете привести исследование зависимости ускорения и эффективности алгоритма от входных данных и количества потоков. Получившиеся результаты необходимо объяснить.

**Вариант 6:** Произвести перемножение 2-ух матриц, содержащих комплексные числа

**Общие сведения о программе**

Основной файл программы - main.cpp, complex\_double.cpp и complex\_double.hpp – файлы для работы с комплексными числами. Также используется заголовочные файлы: iostream, vector, string, pthread.h . В программе используются следующие системные вызовы:

1. **pthread\_create** – создание потока.
2. **pthread\_join** – ожидание потока и его завершение.

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы pthread\_create, pthread\_join, pthread\_detach, pthread\_mutex.
2. Написать класс complex\_double для работы с комплексными числами.
3. Написать файл main.cpp, в котором будет осуществляться перемножение матриц в многопоточном режиме.

**Основные файлы программы**

**main.cpp:**

#include <iostream>

#include <vector>

#include <pthread.h>

#include <string>

#include "complex\_double.hpp"

std::vector<std::vector<complex\_double>> matrix1, matrix2, result;

void\* multiply(void \*arg) {

int i = \*((int\*)arg);

int j = \*((int\*)arg + 1);

int step = \*((int\*)arg + 2);

for (int k = 0; k < step; ++k) {

result[i][j] = 0;

for (int x = 0; x < matrix1[0].size(); ++x) {

result[i][j] += matrix1[i][x] \* matrix2[x][j];

}

i = (i \* matrix1.size() + j + 1) / matrix1.size();

j = (j + 1) % matrix1.size();

}

return nullptr;

}

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

if (argc == 1) {

std::cerr << "Usage: ./main2 [number of threads]" << std::endl;

return -1;

}

int number\_of\_threads = atoi(argv[1]);

if (number\_of\_threads < 1) {

std::cerr << "Too few threads." << std::endl;

return -1;

}

int n1, m1, n2, m2;

std::cin >> n1 >> m1;

matrix1.resize(n1, std::vector<complex\_double>(m1));

for (int i = 0; i < n1; ++i) {

for (int j = 0; j < m1; ++j) {

std::cin >> matrix1[i][j];

}

}

std::cin >> n2 >> m2;

matrix2.resize(n2, std::vector<complex\_double>(m2));

for (int i = 0; i < n2; ++i) {

for (int j = 0; j < m2; ++j) {

std::cin >> matrix2[i][j];

}

}

if (m1 != n2) {

std::cerr << "Math error: can't to multiply matrixes\n";

return -1;

}

result.resize(n1, std::vector<complex\_double>(m2, complex\_double()));

number\_of\_threads = std::min(number\_of\_threads, n1 \* m2);

std::vector<pthread\_t> threads(number\_of\_threads);

std::vector<int[3]> args(number\_of\_threads);

int nm = n1 \* m2;

int position = 0;

for (int \_ = 0; \_ < threads.size(); ++\_) {

int step = nm / number\_of\_threads--;

nm -= step;

int i = position / m2;

int j = position % m2;

args[\_][0] = i; args[\_][1] = j; args[\_][2] = step;

pthread\_create(&threads[\_], NULL, multiply, args[\_]);

position += step;

}

nm = n1 \* m2, number\_of\_threads = threads.size();

position = 0;

for (int \_ = 0; \_ < threads.size(); ++\_) {

pthread\_join(threads[\_], nullptr/\* &values \*/);

}

std::cout << '\n';

for (int i = 0; i < result.size(); ++i) {

for (int j = 0; j < result[i].size(); ++j) {

std::cout << result[i][j] << ' ';

}

std::cout << '\n';

}

}

**complex\_double.hpp:**

#pragma once

#include <iostream>

struct complex\_double {

double x;

double i;

complex\_double();

complex\_double(double&);

~complex\_double() = default;

friend std::istream& operator>>(std::istream&, complex\_double&);

friend std::ostream& operator<<(std::ostream&, complex\_double&);

complex\_double operator=(const double&);

complex\_double operator=(const complex\_double&);

// complex\_double& operator=(complex\_double&&);

complex\_double operator+(const complex\_double&);

complex\_double operator-(const complex\_double&);

complex\_double operator\*(const complex\_double&);

complex\_double operator+=(const complex\_double&);

};

**main.c**

#include "complex\_double.hpp"

complex\_double::complex\_double() {

x = 0;

i = 0;

}

complex\_double::complex\_double(double& val) {

x = val;

i = 0;

}

std::istream& operator>>(std::istream& in, complex\_double& val) {

std::string s;

in >> s;

int n = 0;

bool sign = true;

for (char c : s) {

if (c <= '9' && c >= '0') {

n = n \* 10 + (c - '0');

}

if (c == 'i') {

if (n == 0) {

if (sign)

val.i = 1;

else

val.i = -1;

}

else {

if (sign)

val.i = n;

else

val.i = -n;

n = 0;

}

}

if (c == '+') {

if (sign)

val.x = n;

else

val.x = -n;

n = 0;

sign = true;

}

if (c == '-') {

if (sign)

val.x = n;

else

val.x = -n;

n = 0;

sign = false;

}

}

if (n > 0){

if (sign)

val.x = n;

else

val.x = -n;

}

return in;

}

std::ostream& operator<<(std::ostream& out, complex\_double& val) {

out << val.x;

if (val.i == 0) {

return out;

}

else if (val.i >= 0) {

out << '+';

}

else {

out << '-';

}

out << std::abs(val.i) << 'i';

return out;

}

complex\_double complex\_double::operator=(const double& val) {

x = val;

i = 0;

return \*this;

}

complex\_double complex\_double::operator=(const complex\_double& other) {

x = other.x;

i = other.i;

return \*this;

}

complex\_double complex\_double::operator+(const complex\_double& other) {

complex\_double \*res = new complex\_double();

res->x = x + other.x;

res->i = i + other.i;

return \*res;

}

complex\_double complex\_double::operator-(const complex\_double& other) {

complex\_double res = complex\_double();

res.x = x - other.x;

res.i = i - other.i;

return res;

}

complex\_double complex\_double::operator\*(const complex\_double& other) {

complex\_double res = complex\_double();

res.x = x \* other.x - i \* other.i;

res.i = i \* other.x + x \* other.i;

return res;

}

complex\_double complex\_double::operator+=(const complex\_double& other) {

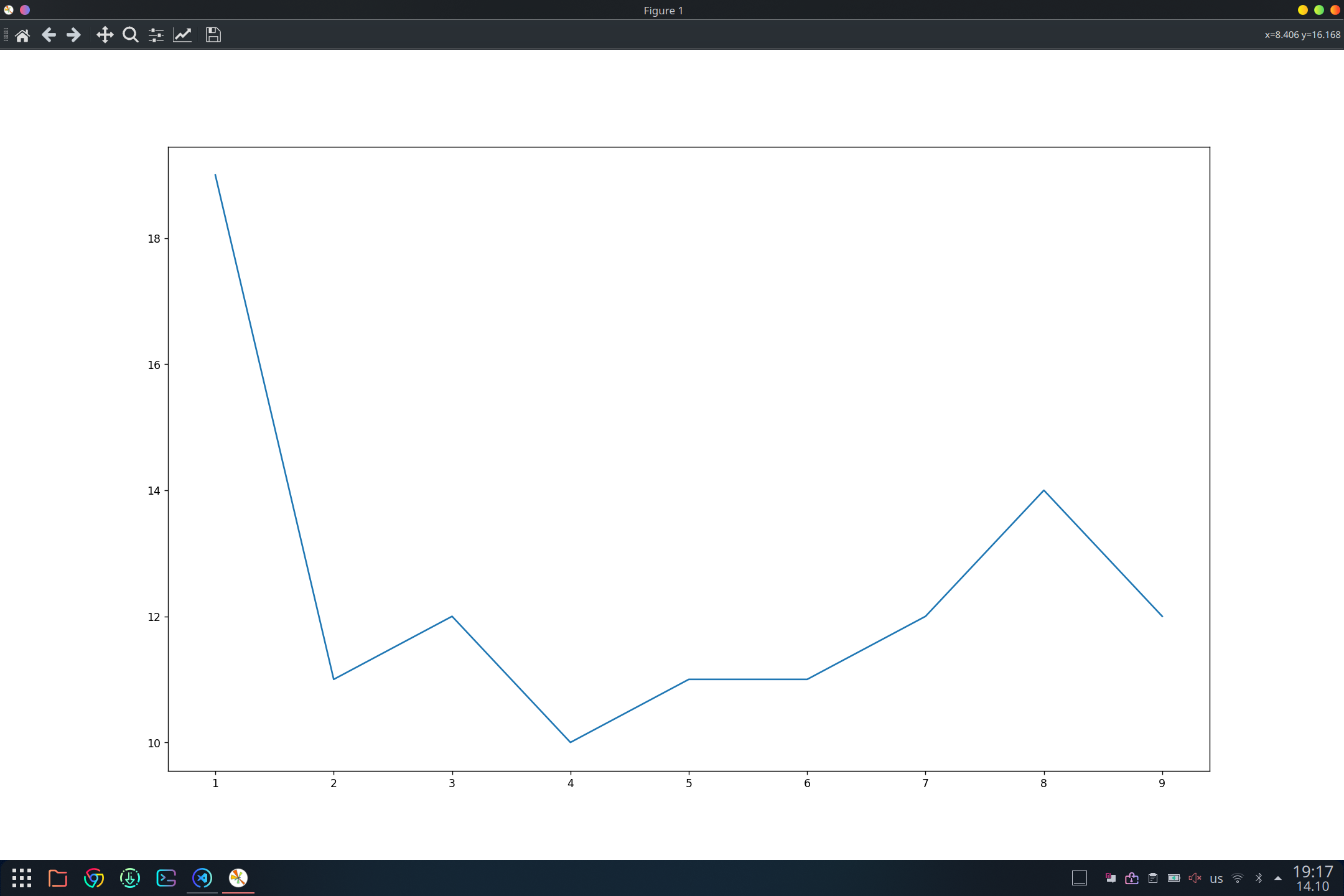
x += other.x;

i += other.i;

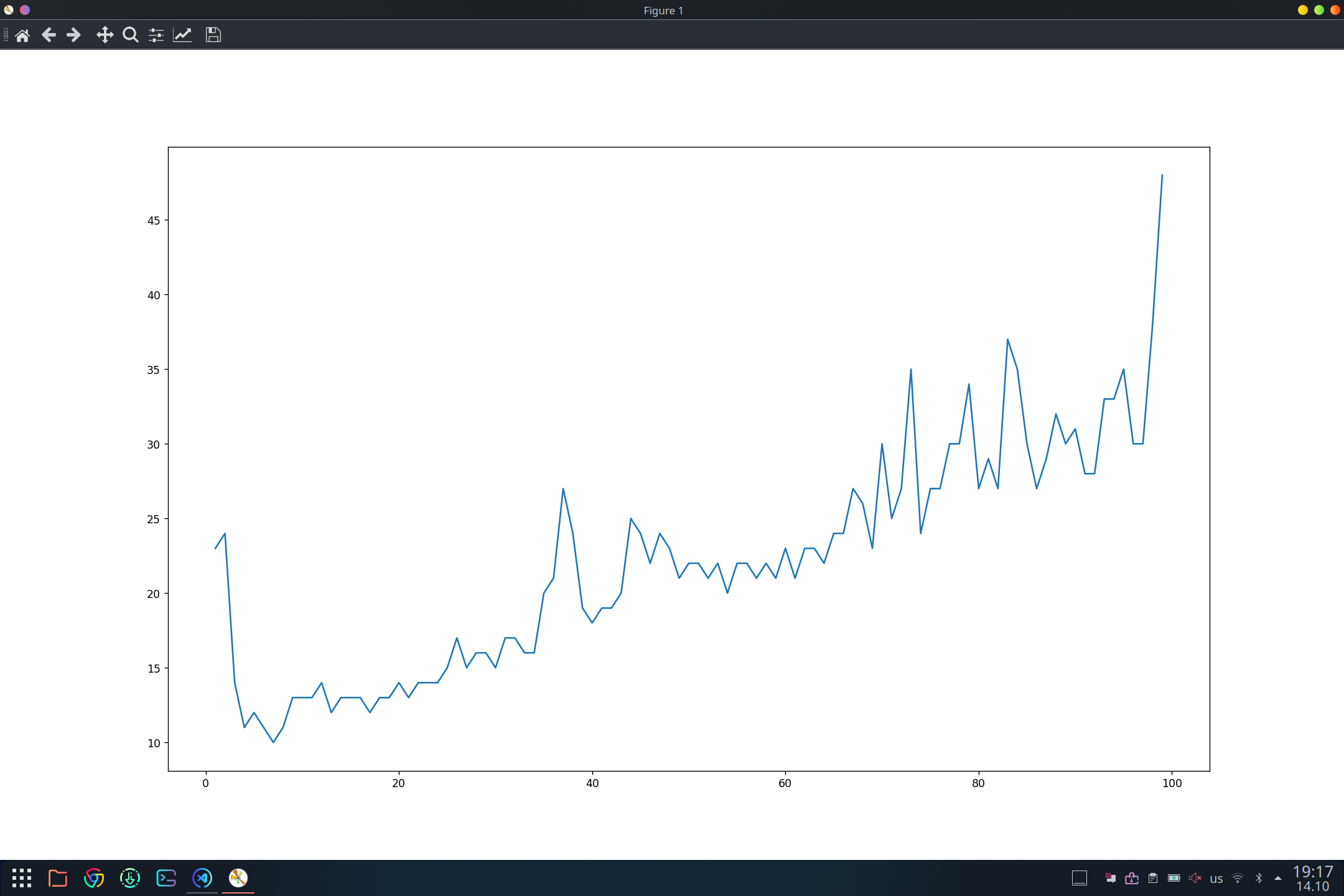
return \*this;

}

**Пример работы**



Многопоточные программы зачастую работают быстрее обычных.



Однако использование чрезмерно большого количества потоков может лишь замедлить работу программы.

**Тесты:**

[kruyneg@matebook14 OS]$ cat ./lab2/tests/test1.txt

3 2

1 1

2 2

3 3

2 3

1 2 3

1 2 3

[kruyneg@matebook14 OS]$ cat ./lab2/tests/test2.txt

3 3

1 1 1

2 2 2

3 3 3

3 3

1 2 3

1 2 3

1 2 3

[kruyneg@matebook14 OS]$ cat ./lab2/tests/test4.txt

3 3

1+i 2+i 3+i

4+i 5+i 6+i

7+i 8+i 9+i

3 3

1+i 1+2i 1+3i

2+4i 2+5i 2+6i

3+7i 3+8i 3+9i

[kruyneg@matebook14 OS]$ cat ./lab2/tests/test5.txt

3 3

1-i 2-i 3-i

4+i 5+i 6+i

7+i 8+i 9+i

3 3

1-i 1+2i 1+3i

2-4i 2+5i 2+6i

3-7i 3+8i 3+9i

**Результат**:  
[kruyneg@matebook14 OS]$ ./build/main2 5 < ./lab2/tests/test1.txt

2 4 6

4 8 12

6 12 18

[kruyneg@matebook14 OS]$ ./build/main2 5 < ./lab2/tests/test2.txt

3 6 9

6 12 18

9 18 27

[kruyneg@matebook14 OS]$ ./build/main2 5 < ./lab2/tests/test4.txt

2+36i -1+42i -4+48i

20+72i 17+87i 14+102i

38+108i 35+132i 32+156i

[kruyneg@matebook14 OS]$ ./build/main2 5 < ./lab2/tests/test5.txt

2-36i 29+30i 32+36i

44-60i 17+87i 14+102i

62-96i 35+132i 32+156i

**Вывод**

Проделав работу, я успешно реализовал разделение вычислений на несколько потоков, что позволило эффективно использовать ресурсы и ускорить процесс перемножения матриц содержащих комплексные числа. Также я реализовал использование конкретного числа потоков, заданного пользователем. Я получил практические навыки работы с многопоточностью в ОС UNIX. Основными проблемами, с которыми я столкнулся при работе c потоками, были осуществление доступа потоков к общим данным и разделение данных передаваемых в разные потоки.

Многопоточность позволяет улучшить производительность программы в задачах, которые могут быть разделены на отдельные задачи. Однако использование чрезмерно большого количества потоков может лишь замедлить работу программы. На основе построенных графиков можно сделать вывод, что использование до 10 потоков является оптимальным.