Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №3 по курсу**

**«Операционные системы»**

**Управление потоками и синхронизация**

Группа: М80-207Б-20

Студент: Пономарев Н.В.

Преподаватель: Миронов Е.С.

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

Дата: 03.10.21

Москва, 2021

# Постановка задачи

Составить и отладить программу на языке С++, обрабатывающую данные в многопоточном режиме. Использовать стандартные средства создания потоков операционной системы Unix. Предусмотреть возможность ограничить максимальное количество потоков, используемых в программе.

**Вариант задания:** 11.

Наложить K раз фильтры эрозии и наращивания на матрицу, состоящую из вещественных чисел. На выходе получается 2 результирующие матрицы

# Общие сведения о программе

Программа состоит из файла **main.cpp**, в котором реализован многопоточный алгоритм наложения матриц наращивания и эрозии на матрицу вещественных чисел.

В программе используются заголовочные файлы iostream, thread, vector.

Используются следующие системные вызовы

1. **thread() –** создает новый поток выполнения в программе.
2. **thread.join()** – для блокировки потока.

# Общий метод и алгоритм решения

* Получить из входного потока количество К наложений матриц, матрицу эрозии и наращивания, максимальное возможное количество потоков и матрицу, к которой мы будем применять фильтры.
* Выполняем построчный обход матрицы, создавая, учитывая ограничение потоков, новый поток для вычисления наложения матрицы для очередной строки.
* После создания потоков мы закрываем их.
* После подсчета очередного наложения фильтров ри необходимости приводим получившуюся матрицу к изначальному виду, обрамляя полученный результат нулевыми элементами.
* После завершения работы потоков заданное количество раз, выводим результат.

**Код программы**

**main.cpp:**

#include <iostream>

#include <fstream>

#include <thread>

#include <vector>

using namespace std;

void prep(int str\_num, int n, int m, int s, double \*matrix, double \*sviortka, double\* results){

double mmatrix[n][m];

double msviortka[s][s];

double mresults[n-2];

int q = 0;

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

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

mmatrix[i][j] = matrix[q];

q++;

}

}

q = 0;

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

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

msviortka[i][j] = sviortka[q];

q++;

}

}

// теперь есть матрица и матрица свёртки

for(int i = 0; i < m - s + 1; i++){

double res = 0;

int j\_core = 0, k\_core = 0;

for(int j = str\_num; j < str\_num + s; j++){

for(int k = i; k < i + s; k++){

res += mmatrix[j][k] \* msviortka[j\_core][k\_core];

k\_core++;

}

k\_core = 0;

j\_core++;

}

mresults[i] = res;

}

for (int i = 0; i < m - 2; i++){

results[str\_num \* (m - s + 1) + i] = mresults[i];

}

}

int main(){

int n\_errosion;

int K, N;

cout << "Please input number of multiplications K: " << endl;

cin >> K;

cout << "Please input number of threads if there are no restrictions enter -1: " << endl;

cin >> N;

if(!N || (N < 0 && N != -1)){

cout << "Error, please enter correct number of threads";

return 0;

}

cout << "Insert size of errosion matrix: " << endl;

cin >> n\_errosion;

double \*errosion = new double[n\_errosion \* n\_errosion];

cout << "Insert errosion matrix elements: " << endl;

for(int i = 0; i < n\_errosion \* n\_errosion; i++){

cin >> errosion[i];

}

//blinking matrix

cout << "Insert size of blinking matrix: ";

int n\_blinking;

cin >> n\_blinking;

double \*blinking = new double[n\_blinking \* n\_blinking];

cout << "Insert blinking matrix elements: " << endl;

for (int i = 0; i < n\_blinking \* n\_blinking; i++){

cin >> blinking[i];

}

//matrix of image

cout << "Insert amount of strings and columns of image matrix: " << endl;

int n, m;

cin >> n >> m;

double \*matrix = new double[n \* m];

double \*matrix2 = new double[n \* m];

if(n < n\_errosion || n < n\_blinking || m < n\_errosion || m < n\_blinking){

cout << "Error, incorrect matrix sizes entered";

return 0;

}

cout << "Insert image matrix elements: " << endl;

for(int i = 0; i < n \* m; i++){

cin >> matrix[i];

matrix2[i] = matrix[i];

}

thread th\_e[n - n\_errosion + 1];

thread th\_b[n - n\_blinking + 1];

double \*e\_results = new double[(n - 2) \* (m - 2)];

for(int i = 0; i < (n - 2) \* (m - 2); i++){

e\_results[i] = 0;

}

double \*b\_results = new double[(n - 2) \* (m - 2)];

for(int i = 0; i < (n - 2) \* (m - 2); i++){

b\_results[i] = 0;

}

while(K > 0){

if(N == -1 || N >= n - n\_errosion + 1){

for(int i = 0; i < n - n\_errosion + 1; i++){

th\_e[i] = thread(prep, i, n, m, n\_errosion, matrix, errosion, e\_results);

}

for(int i = 0; i < n - n\_errosion + 1; i++){

th\_e[i].join();

}

} else {

for(int i = 0; i < n - n\_errosion + 1; i++){

if(i >= N && i != 0){

th\_e[i - N].join();

}

th\_e[i] = thread(prep, i, n, m, n\_errosion, matrix, errosion, e\_results);

}

for(int i = n - n\_errosion + 1 - N; i < n - n\_errosion + 1; i++){

th\_e[i].join();

}

}

int q = 0;

for (int j = 0; j < n \* m; j++){

if ((j >= 0 && j < m) || (j % m == 0) || ((j + 1) % m == 0) || (j < n \* m && j > n \* m - m)){

matrix[j] = 0;

} else {

matrix[j] = e\_results[q];

q++;

}

}

if(N == -1 || N >= n - n\_blinking + 1){

for(int i = 0; i < n - n\_blinking + 1; i++){

th\_b[i] = thread(prep, i, n, m, n\_blinking, matrix2, blinking, b\_results);

}

for(int i = 0; i < n - n\_blinking + 1; i++){

th\_b[i].join();

}

} else {

for(int i = 0; i < n - n\_blinking + 1; i++){

if(i >= N && i != 0){

th\_b[i - N].join();

}

th\_b[i] = thread(prep, i, n, m, n\_blinking, matrix2, blinking, b\_results);

}

for(int i = n - n\_blinking + 1 - N; i < n - n\_blinking + 1; i++){

th\_b[i].join();

}

}

int p = 0;

for (int j = 0; j < n \* m; j++){

if ((j >= 0 && j < m) || (j % m == 0) || ((j + 1) % m == 0) || (j < n \* m && j > n \* m - m)){

matrix2[j] = 0;

} else {

matrix2[j] = b\_results[p];

p++;

}

}

K--;

}

cout << "Erosion result: " << endl;

for (int i = 0; i < (n \* m); i++){

cout << matrix[i] << " ";

if ((i + 1) % (m) == 0){

cout << endl;

}

}

cout << "Blinking result: " << endl;

for (int i = 0; i < (n \* m); i++){

cout << matrix2[i] << " ";

if ((i + 1) % (m) == 0){

cout << endl;

}

}

delete errosion;

delete blinking;

delete matrix;

delete matrix2;

delete e\_results;

delete b\_results;

}

# Демонстрация работы программы

**Пример 1:**

Please input number of multiplications K:

5

Please input number of threads if there are no restrictions enter -1:

2

Insert size of errosion matrix:

3

Insert errosion matrix elements:

0 1 0

1 1 1

0 1 0

Insert size of blinking matrix: 3

Insert blinking matrix elements:

1 0 1

0 0 0

1 0 1

Insert amount of strings and columns of image matrix:

3 6

Insert image matrix elements:

1 2 3 4 5 6

7 8 9 10 11 12

13 14 15 16 17 18

Erosion result:

0 0 0 0 0 0

0 2370 4385 5040 3460 0

0 0 0 0 0 0

Blinking result:

0 0 0 0 0 0

0 0 0 0 0 0

0 0 0 0 0 0

**Пример 2:**

Please input number of multiplications K:

3

Please input number of threads if there are no restrictions enter -1:

-1

Insert size of errosion matrix:

3

Insert errosion matrix elements:

1 1 1

1 1 1

1 1 1

Insert size of blinking matrix: 3

Insert blinking matrix elements:

0 1 0

1 1 0

0 1 1

Insert amount of strings and columns of image matrix:

6 3

Insert image matrix elements:

1 2 3

4 5 6

7 8 9

10 11 12

13 14 15

16 17 18

Erosion result:

0 0 0

0 333 0

0 630 0

0 738 0

0 522 0

0 0 0

Blinking result:

0 0 0

0 200 0

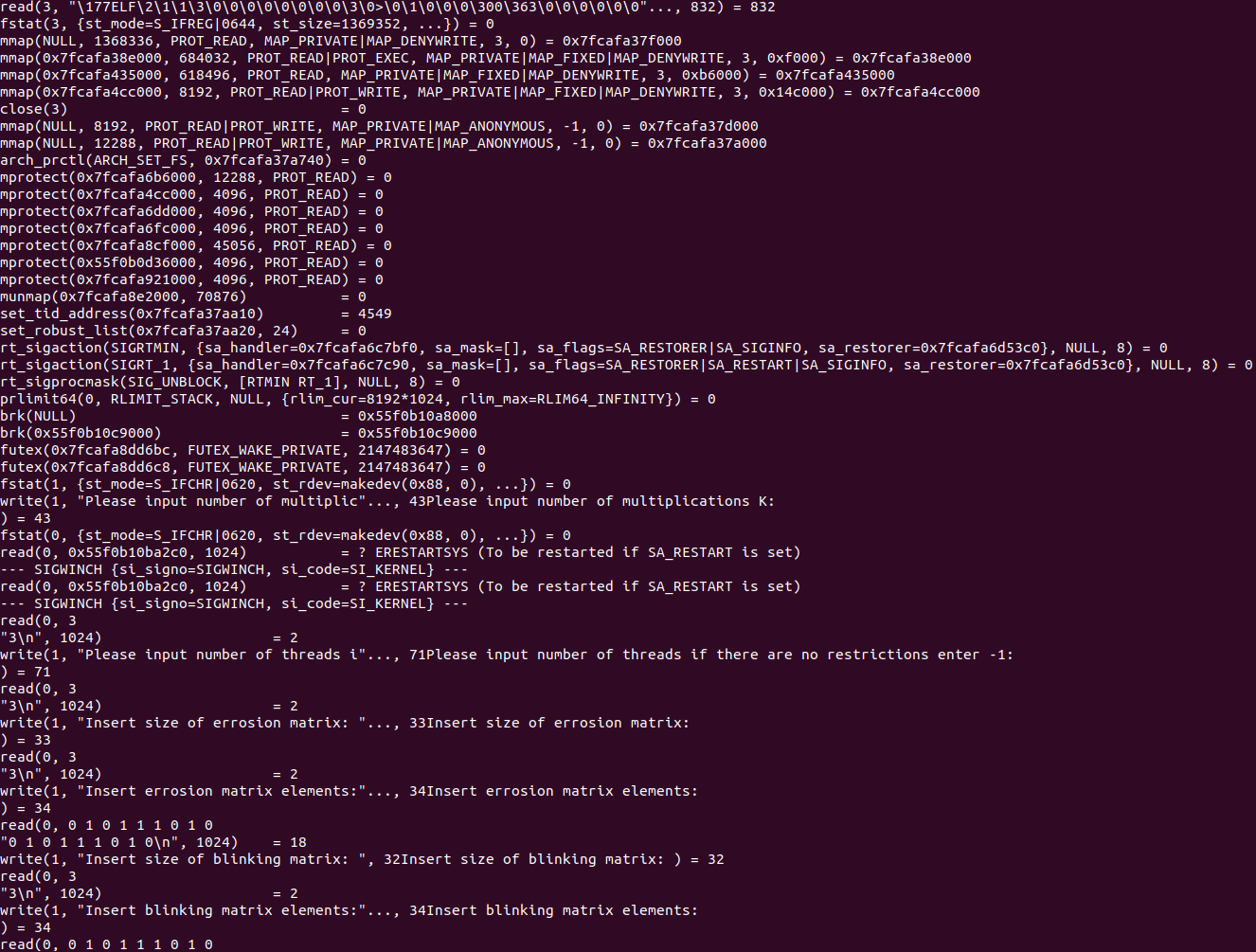
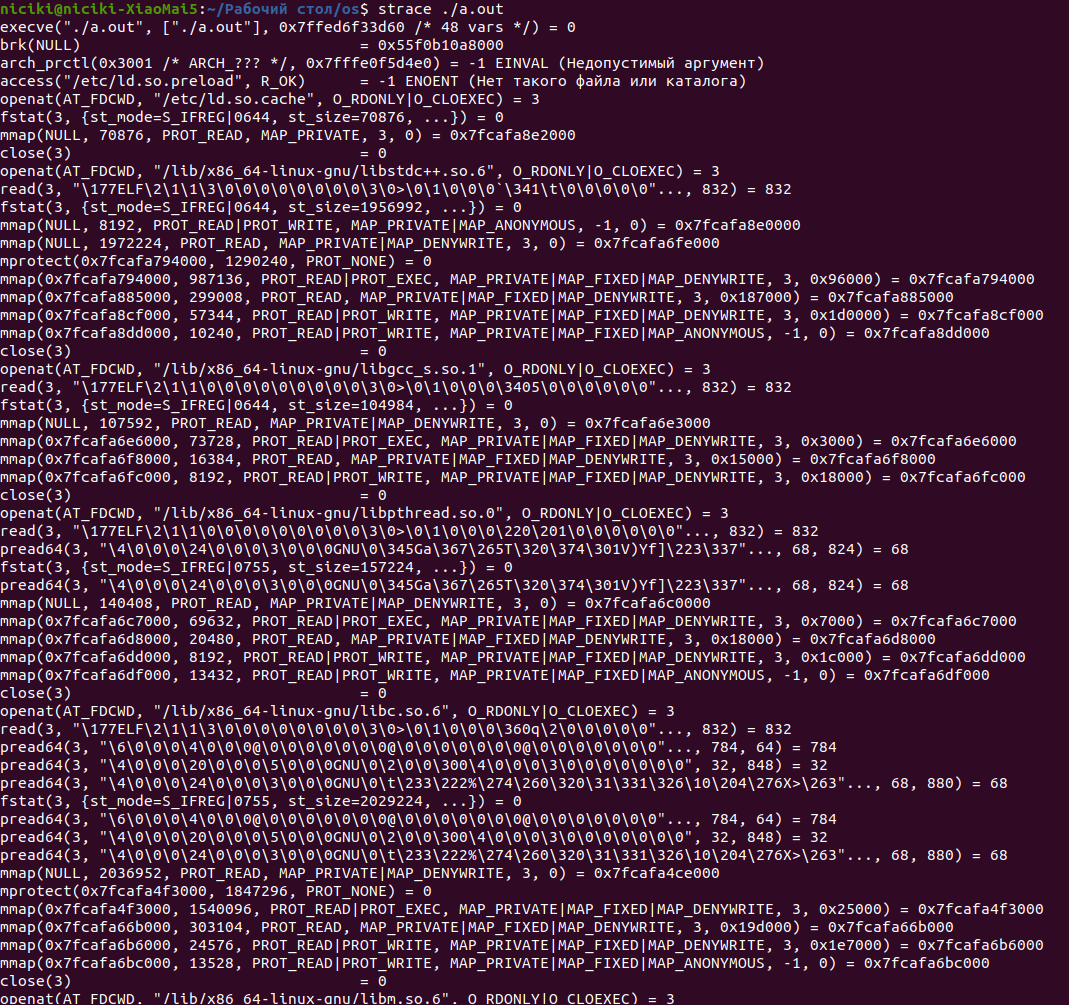
0 374 0

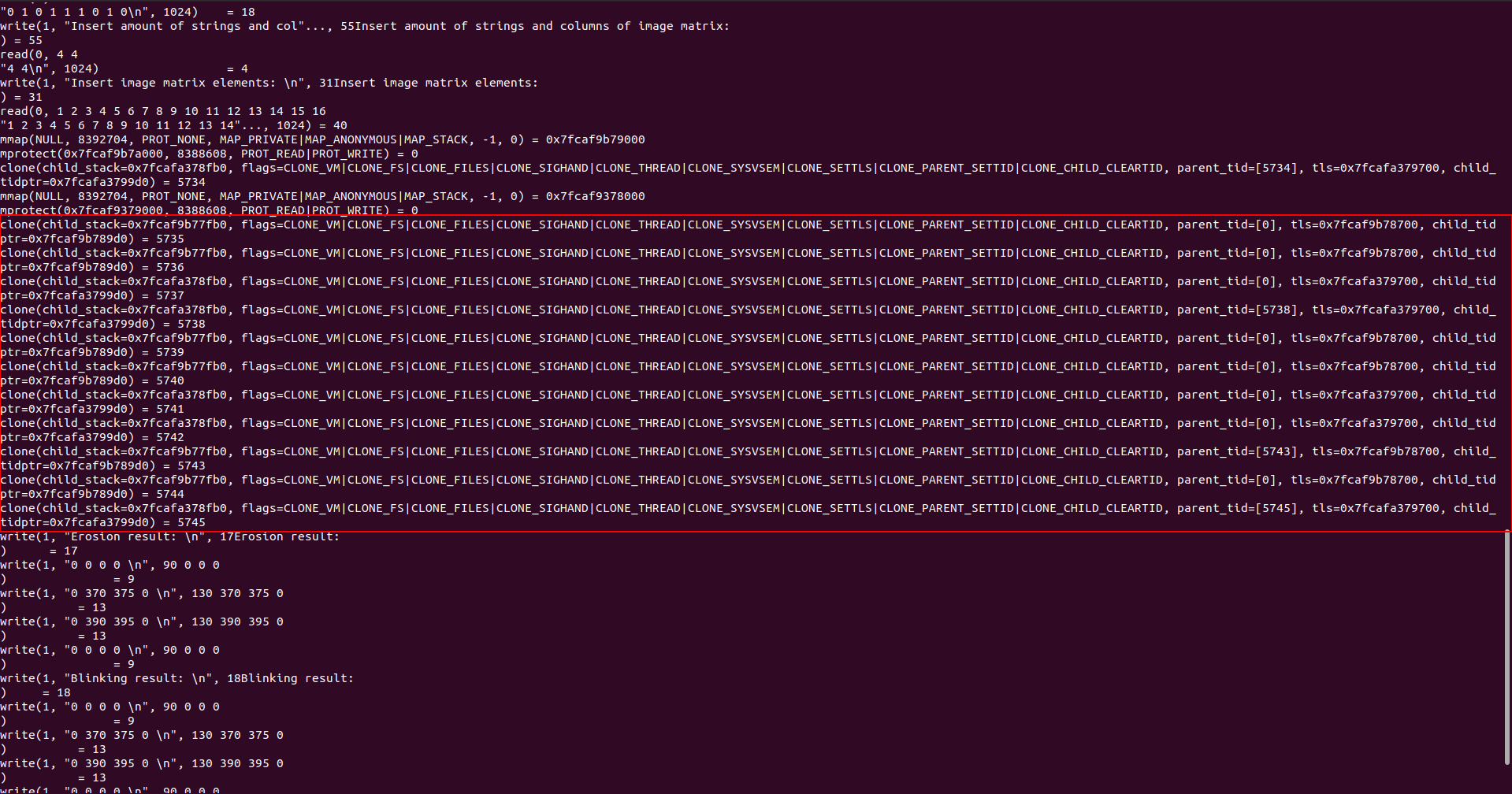
0 434 0

0 305 0

0 0 0

# **Использование утилиты strace**





Листинг выполнения программы с использованием утилиты strace приведен в виде изображений. Работа с потоками отмечена красным прямоугольником на последнем слайде.

# **Сравнение эффективности работы:**

Были проведены тесты работы программы с одним/несколькими потоками.

* Tp – время выполнения на p различных потоках/вычислительных ядрах
* Т1 - время выполнения без многопоточности
* Sp – ускорение. Sp = T1/Tp, Sp < p
* Xp – эффективность/загруженность. Xp = Sp/p, Xp<1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Номер теста и описание | Тp | T1 | Sp | Эффективность |
| Тест 1.  Матрица изображения 100 на 10 производится 1000 наложений фильтров эрозии и наращивания, каждый из которых размерностями 3 на 3. 10 Потоков. | Результат: 6.834 сек | Результат: 12.649 сек | Результат: 1.85 | Результат: 0.185 |
| Матрица изображения 10 на 100 производится 1000 наложений фильтров эрозии и наращивания, каждый из которых размерностями 3 на 3. 10 потоков. | Результат: 6.90 сек | Результат: 10.12 сек | Результат:  1.46 | Результат: 0.146 |
| Матрица изображения 10 на 10 производится 10000 наложений фильтров эрозии и наращивания, каждый из которых размерностями 4 на 4. 10 потоков. | Результат: 4.318 сек | Результат: 5.415 сек | Результат:1.25 | Результат:0.125 |
| Матрица изображения 5 на 5 производится 100 наложений фильтров эрозии и наращивания, каждый из которых размерностями 3 на 3. 2 потока. | Результат: 0.033 сек | Результат: 0.0542 | Результат:  1.64 | Результат:  0.82 |

Таким образом, мы видим, что использование многопоточности действительно ускоряет вычисление заданной задачи(средний результат ускорения составляет около 1.5, а эффективность падает при увеличении, входных размеров и количества наложений уменьшается), при небольших входных данных разница не столь существенна, но, по мере увеличения размеров входных изображений это разница увеличивается.

# Вывод

Выполняя данную лабораторную работу, я познакомился с методами работы с многопоточностью, с тем, как можно оценивать эффективность и ускорение программы относительно однопоточной. Потоки помогают, распараллелить(но чаще всего имеет место псевдопараллельность) несколько разных вычислений, но не всегда количество влияет на качество: большое количество потоков играет роль на скорости вычислений в этих самых потоках. Также стоит отметить, что при создании поток имеет доступ ко всему адресному пространству процесса, в котором он создан. Разные потоки могут производить чтение/запись в одинаковые блоки памяти, что, с одной стороны улучшает производительность параллельных вычислений, а с другой стороны, может приводить к ошибкам. Например, из за невнимательности программиста несколько потоков могут работать с одним и тем же блоком памяти и, если не предусмотрено использование мьютексов или обращение к разным ячейкам памяти не запрещено, то может произойти непредвиденные действия например, один поток может считать данные, затем в них после чтения сразу запишет данные какой либо другой поток, затем первый снова запишет данные, но на основании тех, что он считал. Следовательно, данные, записанные вторым потоком, потеряются. Для того, чтобы справиться с данной проблемой, были придуманы различные примитивы синхронизации, ограничивающие доступ потокам к тем или иным участкам памяти.