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 в ОС**

Студент: Федоров А.Д.

Группа: М8О–206Б–20

Вариант: 13

Преподаватель: Соколов Андрей Алексеевич

Оценка: 3

Дата: 21.12.2021

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2021

**Постановка задачи**

## Задание

Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При обработки использовать стандартные средства создания потоков операционной системы (Windows/Unix). Ограничение потоков 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 входящих данных и количества потоков. Получившиеся результаты необходимо объяснить.

Вариант 16:

Задаётся радиус окружности. Необходимо с помощью метода Монте-Карло рассчитать её площадь.

**Общие сведения о программе**

Программа компилируется из файла lab3.c. Также используется заголовочные файлы: stdio.h, unistd.h, stdlib.h, ctype.h, stdbool.h, pthread.h, stdint.h, inttypes.h. В программе используются следующие системные вызовы:

1. **pthread\_create ––** (является оберткой над системным вызовом clone) создает новый поток в вызывающем процессе. В качестве аргументов принимает указатель на структуру-идентификатор потока pthread\_t, атрибуты потока, функцию, которая будет запускаться в потоке, список аргументов для функции в виде указателя на void. В случае успеха возвращает 0, иначе возвращает номер ошибки.
2. **pthread\_join ––** используется для ожидания завершения потока. Данная функция блокирует вызывающий поток, пока указанный поток не завершится. В качестве аргументов принимает структуру pthread\_t потока и указатель на переменную, в которую будет записан результат, возвращаемый потоком. В случае успеха возвращает 0, иначе возвращает номер ошибки.
3. **pthread\_exit ––** завершает вызываемый поток. В качестве аргумента принимает значение, которое вернется при завершении потока. Функция всегда завершается успехом.
4. **read ––** предназначена для чтения какого-то числа байт из файла, принимает в качестве аргументов файловый дескриптор, буфер, в который будут записаны данные и число байт. В случае успеха вернет число прочитанных байт, иначе -1.
5. **write ––** предназначена для записи какого-то числа байт в файл, принимает в качестве аргументов файловый дескриптор, буфер, из которого будут считаны данные для записи и число байт. В случае успеха вернет число записанных байт, иначе -1.

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы с потоками в C.
2. Написать вспомогательные функции для работы с матрицами.
3. С помощью принципа разбиения задачи на маленькие подзадачи, реализовать функцию, которая будет проводить частичную обработку данных, для ее запуска в многопоточном режиме.
4. Реализовать функцию, которая будет создавать потоки, запускать потоковую функцию, управлять потоками.
5. Реализовать обработку системных ошибок согласно заданию

**Основные файлы программы**

**Lab3.c:**

#include <pthread.h>

#include <time.h>

#include <stdlib.h>

#include <math.h>

#include <stdio.h>

#include <limits.h>

double calc\_square(double r){

return r \* r \* 3.14;

}

void\* calculate\_dots(void\* args){

double dots\_per\_thread = \*((double\*) args);

double r = \*((double\*) args + 1);

srand(time(NULL));

rand();

int\* dots = malloc(sizeof(int));

\*dots = 0;

double rMax = r;

for (int i = 0; i < dots\_per\_thread; ++i) {

double x = (double)rand() / RAND\_MAX \* rMax;

double y = (double)rand() / RAND\_MAX \* rMax;

printf("%lf %lf\n", x , y);

if((x \* x + y \* y) <= r \* r){

++(\*dots);

}

}

return (void\*)dots;

}

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

if(argc > 2){

printf("Usage: %s + <number\_of\_threads>\n", argv[0]);

return -1;

}

int n = 1;

if(argc == 2){

n = atoi(argv[1]);

printf("%d\n", n);

if(n == 0){

printf("Usage: %s + <number\_of\_threads>\n", argv[0]);

return -1;

}

}

const int dots\_cnt = 1000000;

double dots\_per\_thread = dots\_cnt / n;

printf("%lf\n", dots\_per\_thread);

double r;

scanf("%lf", &r);

double args[2] = {dots\_per\_thread, r};

pthread\_t threads[n];

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

pthread\_create(&threads[i], NULL, calculate\_dots, &args);

}

int counted\_dots = 0;

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

int\* cur;

pthread\_join(threads[i], (void\*\*)&cur);

counted\_dots += \*cur;

printf("!%d\n", \*cur);

free(cur);

}

printf("Calculated square: %lf\n", r \* r \* counted\_dots / (dots\_per\_thread \* n) \* 4.0);

printf("Estimated square: %lf\n", calc\_square(r));

return 0;

}

**Пример работы**

mediocrity@LAPTOP-EEF7UMGS:~$ ./LR3 6

6

166666.000000

3

!131089

!130944

Calculated square: 28.303277

Estimated square: 28.260000

**Вывод**

В ходе выполнения данной лабораторной работы я познакомился с тем, как производится работа с потоками в языке Си. Основная идея применения потоков состоит в разбиение большой задачи на множество более мелких задач, которые выполняются параллельно. Но данный подход требует особенной внимательности программиста к совместно используемым данным и тщательного продумывания алгоритма.