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: Е. А. Айрапетова

Группа: М8О–206Б–20

Вариант: 2

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2020.

**Постановка задачи**

## Цель работы

Приобретение практических навыков в:

* Управление потоками в ОС
* Обеспечение синхронизации между потоками

## 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 2: Отсортировать массив целых чисел при помощи параллельного алгоритма быстрой сортировки.

**Общие сведения о программе**

Программа компилируется из файла lab3.c. Также используется заголовочные файлы: stdio.h, stdlib.h, string.h, pthread.h. Благодаря библиотеке pthread.h, мы имеем доступ к следующим фукциям:

* pthread\_join() – функция, используемая для синхронизации потоков. Она ожидает завершения потока, который поступает к ней в качестве аргумента, а затем уже возвращает значение.
* pthread\_mutex\_destroy() – уничтожение объекта мьютекса, на который он ссылается.
* pthread\_create() – создание нового мьютекса.

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

* Изучить принципы работы pthread.
* Написать функцию быстрой сортировки.
* Написать функцию запуска многопоточной обработки.
* Написать обработку ошибок.
* Написать тесты.

**Основные файлы программы**

**lab3.c**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <pthread.h>

pthread\_mutex\_t mutex = PTHREAD\_MUTEX\_INITIALIZER;

int max\_threads;

int check() {

int result;

if (max\_threads <= 0){

return 0;

}

pthread\_mutex\_lock(&mutex);

if (max\_threads > 0){

result = 1;

}

else{

result = 0;

}

--max\_threads;

pthread\_mutex\_unlock(&mutex);

return result;

}

typedef struct vector{

int size;

int\* data;

}vector;

typedef struct args {

int\* array;

int left;

int right;

}args;

void\* quick\_sort(void \*args) {

int\* array = ((struct args\*)args)->array;

int left = ((struct args\*)args)->left;

int right = ((struct args\*)args)->right;

int i = left, j = right, tmp;

int t1 = 0, t2 = 0;

int pivot = array[(left + right) / 2];

while(i <= j) {

for (; array[i] < pivot; ++i);

for (; array[j] > pivot; --j);

if (i <= j) {

tmp = array[i];

array[i++] = array[j];

array[j--] = tmp;

}

}

pthread\_t thread1 = 0;

pthread\_t thread2 = 0;

if (left < j) {

if(!check(args)) {

struct args\* args1 = malloc(sizeof(args));

args1->array = array;

args1->left = left;

args1->right = --j;

quick\_sort(args1);

}

else {

++t1;

struct args\* args1 = malloc(sizeof(args));

args1->array = array;

args1->left = left;

args1->right = --j;

int status = pthread\_create(&thread1, NULL, quick\_sort, args1);

if (status != 0){

printf("CAN'T CREATE THREAD");

return NULL;

}

}

}

if (i < right) {

if(!check(args)) {

struct args\* args1 = malloc(sizeof(args));

args1->array = array;

args1->left = ++i;

args1->right = right;

quick\_sort(args1);

}

else {

++t2;

struct args\* args1 = malloc(sizeof(args));

args1->array = array;

args1->left = ++i;

args1->right = right;

int status = pthread\_create(&thread2, NULL, quick\_sort, args1);

if (status != 0) {

printf("CAN'T CREATE THREAD");

return NULL;

}

}

}

if (t1 != 0) {

int status1 = pthread\_join(thread1, 0);

if (status1 != 0) {

printf("JOIN ERROR");

return NULL;

}

}

if (t2 != 0) {

int status2 = pthread\_join(thread2, 0);

if (status2 != 0) {

printf("JOIN ERROR");

return NULL;

}

}

free(args);

return NULL;

}

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

if (argc != 2 || atoi(argv[1]) < 1) {

printf("WRONG INPUT\n");

return 1;

}

max\_threads = atoi(argv[1]) - 1;

vector\* vector = malloc(sizeof(vector));

int s;

if(!scanf("%d", &s)) {

printf("WRONG INPUT\n");

return 1;

}

vector->data = (int\*)malloc(sizeof(int)\* s);

int x;

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

if(!scanf("%d", &x)) {

printf("WRONG INPUT\n");

return 1;

}

vector->data[i] = x;

}

printf("Entered vector:\n");

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

printf("%d\n", vector->data[i]);

}

vector->size = s;

args\* args1 = malloc(sizeof(args));

args1->array = vector->data;

args1->left = 0;

args1->right = vector->size - 1;

quick\_sort(args1);

printf("Result:\n");

for (int i = 0; i < vector->size; ++i) {

printf("%d\n", vector->data[i]);

}

free(vector);

pthread\_mutex\_destroy(&mutex);

return 0;

}

**Пример работы**

jane@Evgenia:/mnt/c/Files/ОСи$ ./a.out 8 < test

Entered vector:

4

6

15

0

3

2

7

1

11

17

17

17

17

Result:

0

1

6

3

4

2

7

15

11

17

17

17

17

**Вывод**

В процессе выполнения данной работы, я научилась работать с потоками, а также отметила для себя некоторые преимущества. Например, создание потока занимает меньше времени, чем создание процесса. Также, из-за того, что потоки, в отличие от процессов используют одну область памяти, можно ускорить работу программы, отводя каждому потоку определённое действие.