Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №3 по курсу

«Операционные системы»

ПОТОКИ

Студент: Белоносов Кирилл Алексеевич
Группа: М8О–208Б–21
Вариант:2
Преподаватель: Соколов Андрей Алексеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2022.

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 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 состоит из одного файла — main.c. Максимальное количество потоков указывается как аргумент программы. На вход программа получает размер входного вектора n и n чисел.

1. **pthread_create()** - создать новый поток
2. **pthread_mutex_init()** - создать новый мьютекс
3. **pthread_join()** - дождаться завершения потоками
4. **pthread_mutex_destroy()** - уничтожить мьютекс
5. **pthread_mutex_lock()** - заблокироваться на мьютексе
6. **pthread_mutex_unlock()** - освободить мьютекс

Общий метод и алгоритм решения.

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы pthread_create, pthread_mutex_lock и т.д.
2. Изучить алгоритм quicksort
3. Реализовать многопоточный алгоритм quicksort
4. Провести тесты

Основные файлы программы

main.c:

```
#include "unistd.h"
#include "pthread.h"
#include "stdlib.h"
#include "stdio.h"
#include <time.h>
#include <errno.h>
typedef struct {
    int * vec;
    int l;
    int r;
} QSin;
int THREAD_COUNT = 0;
int MAXTHREAD = 4;
pthread_mutex_t mutex;
int partition(int * vec, int l, int r) {
    int v = vec[(l + r) / 2];
    int i = l;
    int j = r;
    while(i <= j) {
        while(vec[i] < v)
            i++;
        while(vec[j] > v)
            j--;
        if(i >= j)
```

```
            break;
        int w = vec[i];
        vec[i++] = vec[j];
        vec[j--] = w;
    }
    return j;
}
void * quicksort(void * arg) {
    int err;
    QSin * data = (QSin*) arg;
    pthread_t id = pthread_self();
    if(data->l < data->r) {
        int q = partition(data->vec, data->l, data->r);
        QSin * data1 = (QSin *)malloc(sizeof(QSin));
        data1->vec = data->vec;
        data1->l = data->l;
        data1->r = q;
        pthread_t id1 = 0;
        if(THREAD_COUNT < MAXTHREAD) {
            if(pthread_create(&id1, NULL, &quicksort, data1) == 0) {
                pthread_mutex_lock(&mutex);
                ++THREAD_COUNT;
                pthread_mutex_unlock(&mutex);
            } else {
                quicksort(data1);
            }
        } else {
            quicksort(data1);
        }
        QSin * data2 = (QSin *)malloc(sizeof(QSin));
        data2->vec = data->vec;
        data2->l = q + 1;
        data2->r = data->r;
```

```
        pthread_t id2 = 0;
        if(THREAD_COUNT < MAXTHREAD) {
            if(pthread_create(&id2, NULL, &quicksort, data2) == 0) {
                pthread_mutex_lock(&mutex);
                ++THREAD_COUNT;
                pthread_mutex_unlock(&mutex);
            } else {
                quicksort(data2);
            }
        } else {
            quicksort(data2);
        }
        if(id1 != 0) {
            if(pthread_join(id1, NULL) == 0) {
                pthread_mutex_lock(&mutex);
                --THREAD_COUNT;
                pthread_mutex_unlock(&mutex);
            }
        }
        if(id2 != 0) {
            if(pthread_join(id2, NULL) == 0) {
                pthread_mutex_lock(&mutex);
                --THREAD_COUNT;
                pthread_mutex_unlock(&mutex);
            }
        }
    }
}

int main (int argc, char * argv[]) {
    if(argc < 2) {
        fprintf(stderr, "input count of threads\n");
        exit(-1);
```

```
    } else {
        char * c = argv[1];
        MAXTHREAD = atoi(c);
    }
    int n;
    int err;
    scanf("%d", &n);
    int * vec = calloc(n, sizeof(int));
    for (int i = 0; i < n; ++i) {
        scanf("%d", &vec[i]);
    }
    QSin * in = (QSin *)malloc(sizeof(QSin));
    in->vec = vec;
    in->l = 0;
    in->r = n - 1;
    pthread_mutex_init(&mutex, NULL);
    pthread_t id;
    err = pthread_create(&id, NULL, &quicksort, in);
    if(err != 0) {
        perror("thread not create");
    }
    err = pthread_join(id, NULL);
     if(err != 0) {
        perror("error join to thread");
    }
    pthread_mutex_destroy(&mutex);
    for (int i = 0; i < n; ++i)
    {
        printf("%d ", vec[i]);
    }
    printf("\n");
    return 0;
}
```

Пример работы

kirill@kirill:~/localProjects/OSlabs/lab3/test$ ./run.sh

[50%] Built target main

[100%] Built target qs

multithreading algorithm

----------test 1----------

0 0 1 4 5 6 6 7 7 7 7 8 8 11 12 14 14 14 15 15 16 17 18 18 19 22 22 25 27 30 31 33 33 37 38 39 40 42 42 42 42 43 43 47 47 48 48 49 51 51 51 54 54 55 59 60 61 61 63 63 64 69 69 69 69 70 70 70 71 72 73 73 73 75 76 76 78 78 79 79 80 83 84 85 86 89 92 92 92 92 93 93 94 95 95 97 98 98 98 100

Time: 0.000599

----------test 2----------

1 2 3 4 5 6 7 8 9 10

Time: 0.000526

----------test 3----------

1 2 3 4 5 6 7 8 9 9 10 11 12 13 14 15 16 17 18 19

Time: 0.000570

base algorithm

----------test 1----------

0 0 1 4 5 6 6 7 7 7 7 8 8 11 12 14 14 14 15 15 16 17 18 18 19 22 22 25 27 30 31 33 33 37 38 39 40 42 42 42 42 43 43 47 47 48 48 49 51 51 51 54 54 55 59 60 61 61 63 63 64 69 69 69 69 70 70 70 71 72 73 73 73 75 76 76 78 78 79 79 80 83 84 85 86 89 92 92 92 92 93 93 94 95 95 97 98 98 98 100

Time: 0.000016

----------test 2----------

1 2 3 4 5 6 7 8 9 10

Time: 0.000003

----------test 3----------

1 2 3 4 5 6 7 8 9 9 10 11 12 13 14 15 16 17 18 19

Time: 0.000004

Вывод

В результате данной лабораторной работы, я научился работать я потоками и мьютексами, реализовал многопоточный алгоритм быстрой сортировки. Сравнивая результаты времени сортировки двух алгоритмов мы получили, что многопоточный алгоритм работает медленнее, что скорее всего связано с большим количеством системных вызовов, связанных с работой потоков. Полученные мною знания пригодятся при дальнейшей работе.