Московский Авиационный Институт
(Национальный исследовательский Университет)

Факультет: «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»

Лабораторная работа №3
по курсу «Операционные системы»

Студент:	Марков А.Н.
Группа:	М80-208Б-18
Преподаватель:	Миронов Е.С.
Оценка:	
Дата:	

Москва
2019

Содержание

1. Постановка задачи.

Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При обработке использовать стандартные средства создания потоков операционной системы. При создании необходимо предусмотреть ключи, которые позволяли бы задать максимальное количество потоков, используемое программой. При возможности необходимо использовать максимальное количество возможных потоков. Ограничение потоков может быть задано или ключом запуска программы, или алгоритмом.

Вариант задания №2. Отсортировать массив строк при помощи параллельного алгоритма быстрой сортировки.

2. Общие сведения о программе.

Исходный код хранится в файле multi_threading_qsort.c. В данном файле используются заголовочные файлы stdio.h, stdlib.h, pthread.h, string.h. В программе используются следующие системные вызовы для работы с потоками из заголовочного файла pthread.h:

1. pthread_create — для создания нового потока.
2. pthread_join — для ожидания завершения потока, используется для синхронизации потоков.

Количество потоков, которые используются при сортировке, передается ключом запуска программы. Программа считывает данные с stdin. Сначала считывается количество элементов в массиве, который необходимо отсортировать. Затем циклом считываются все строки.

3. Общий метод и алгоритм решения.

Исходный набор данных расположен на первом потоке, с него начинается работа алгоритма:

1. Поток выбирает опорный элемент, сортирует остальные элементы относительно него (большие — в правую часть, меньшие или равные — в левую).
2. Если есть возможность создать поток, то меньшую часть основной поток отдает меньшую часть новому потоку. Иначе сам сортирует ее.
3. После этого основной поток продолжает работать с большей частью одновременно (или нет), в то время как другой поток работает с меньшей по принципу начиная с первого пункта.

4. Основные файлы программы.

Файл multi_threading_qsort.c (параллельная быстрая сортировка):

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <string.h>

int MAX;

int THREAD_MAX;

struct Args {
    char **A;
    int p;
    int r;
};
typedef struct Args Args;

void swap(char **o1, char **o2) {
    char *tmp = *o1;
    *o1 = *o2;
    *o2 = tmp;
}

int Partion(char **A, int p, int r) {
    char *x = A[r];
    int i = p - 1;
    for (int j = p; j <= r - 1; j++) {
        if (strcmp(A[j], x) <= 0) {
            i++;
            swap(&A[i], &A[j]);
        }
    }
    swap(&A[i + 1], &A[r]);

    return i + 1;
}

void *Quick_sort(void *arg) {
    Args tmp_args = *((Args *) arg);

    if (tmp_args.p < tmp_args.r) {
        int q = Partion(tmp_args.A, tmp_args.p, tmp_args.r);
        Args new_left_args;
        new_left_args.A = tmp_args.A;
```

```
            new_left_args.p = tmp_args.p;
            new_left_args.r = q - 1;
            if (THREAD_MAX != 0) {
                pthread_t left_thread;
                THREAD_MAX--;
                pthread_create(&left_thread, NULL, &Quick_sort, &new_left_args);
                Args new_right_args;
                new_right_args.A = tmp_args.A;
                new_right_args.p = q + 1;
                new_right_args.r = tmp_args.r;
                Quick_sort(&new_right_args);
                pthread_join(left_thread, NULL);
            }
            else {
                Quick_sort(&new_left_args);
                Args new_right_args;
                new_right_args.A = tmp_args.A;
                new_right_args.p = q + 1;
                new_right_args.r = tmp_args.r;
                Quick_sort(&new_right_args);

            }
        }

        return NULL;
}

int main(int argc, char **argv) {
        if (argc < 2) {
            printf("Input number of threads: ./multi_threading_qsort <threads>\n");
            return 0;
        }
        THREAD_MAX = atoi(argv[1]);
        printf("Input number of elements in the array: ");
        scanf("%d", &MAX);
        char **A = (char **) malloc(sizeof(char *) * MAX);
        for (int i = 0; i < MAX; i++) {
          A[i] = (char*)malloc(sizeof(char)*30);
          scanf("%s", A[i]);
       }
        printf("_____\n");
        Args begin_args;
        begin_args.A = A;
        begin_args.p = 0;
        begin_args.r = MAX - 1;
```

```
    pthread_t begin_thread;
    pthread_create(&begin_thread, NULL, &Quick_sort, &begin_args);

    pthread_join(begin_thread, NULL);
    for (int i = 0; i < MAX; i++) {
        printf("%s\n", A[i]);
    }
    printf("\n");

    return 0;
}
```

Файл standart_qsort.c (однопоточная быстрая сортировка):

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int MAX;

void swap(char **o1, char **o2) {
    char *tmp = *o1;
    *o1 = *o2;
    *o2 = tmp;
}

int Partion(char **A, int p, int r) {
    char *x = A[r];
    int i = p - 1;
    for (int j = p; j <= r - 1; j++) {
        if (strcmp(A[j], x) <= 0) {
            i++;
            swap(&A[i], &A[j]);
        }
    }
    swap(&A[i + 1], &A[r]);

    return i + 1;
}

void Quicksort(char **A, int p, int r) {
    if (p < r) {
        int q = Partion(A, p, r);
        Quicksort(A, p, q - 1);
        Quicksort(A, q + 1, r);
```

```
    }
}

int main() {
    printf("Input number of elements in the array: ");
    scanf("%d", &MAX);
    char **A = (char **) malloc(sizeof(char *) * MAX);
    for (int i = 0; i < MAX; i++) {
      A[i] = (char*)malloc(sizeof(char)*30);
      scanf("%s", A[i]);
   }
    printf("_____\n");

    Quicksort(A, 0, MAX - 1);
    for (int i = 0; i < MAX; i++) {
        printf("%s\n", A[i]);
    }
    printf("\n");

    return 0;
}
```

Файл make_test.cpp (создание тестов):

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <ctime>

int main() {
    srand(time(NULL));
    int MAX;
    std::cin >> MAX;

    std::ofstream out("test.txt");
    out << MAX << "\n";
    for (int i = 0; i < MAX; i++) {
        int rand_cnt = 1 + rand() % 29;
        for (int j = 0; j < rand_cnt; j++) {
            out << (char) ('a' + (rand() % 26));
        }
        out << "\n";
    }

    return 0;
```

}

5. Демонстрация работы программы.

```
oem@Alex-PC:~/Documents/OS/OS/lab03/src$ ./multi_threading_qsort 2
Input number of elements in the array: 10
cge
cc
c
gge
aaaa
aaa
bb
bg
yu
uy
_____________________________________________
aaa
aaaa
bb
bg
c
cc
cge
gge
uy
yu
```

6. Тесты.

Размер массива	Параллельный алгоритм (2 потока)	Стандартный алгоритм	Ускорение
1000000	0m0,980s	0m1,077s	1,10
2000000	0m2,857s	0m3,285s	1,15
4000000	0m8,455s	0m11,285s	1,33
8000000	0m29,921s	0m39,481s	1,32

7. Вывод.

Потоки удобно применять для многозадачности и для ускорения некоторых алгоритмов. В отличие от процессов потоки имеют возможность совместно использовать одно адресное пространство. Также потоки быстрее выполняют операции создания и уничтожения, так как с потоком не связаны никакие ресурсы. В данной лабораторной работе была продемонстрирована реализация параллельного алгоритма быстрой сортировки. И если смотреть на результаты тестов, то время работы действительно различается.