Московский Авиационный Интститут
(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

Студент: Артемьев Дмитрий Иванович
Группа: М8О-206Б-18
Вариант: 2
Преподаватель: Соколов Андрей Алексеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2019

Условие

2. Отсортировать массив строк при помощи параллельного алгоритма быстрой сортировки.

Описание программы

Код программы находится в файле main.c.

Ход выполнения программы

1. Запуск программы с параметрами количества потоков depth (2^{depth}), длины сортируемого массива
2. Считывание элементов массива
3. Запуск параллельной быстрой сортировки
4. на определённом уровне рекурсии если максимальная глубина рекурсии, на которой можно создавать новые потоки не достигнута, создаются новые потоки, выполняющие сортировку в данном им куске массива. Иначе потоки не создаются и оставшуюся часть работы выполняют существующие потоки.
5. Вывод результата сортировки.
6. Завершение работы программы.

Недочёты

Нельзя задавать произвольное количество потоков, создающихся в ходе выполнения.

Выводы

Я научился работать с потоками в Linux, обеспечивать синхронизацию между ними, решать возникающие состязательные ситуации с помощью mutex, которые, однако, не используются мной в самой лабораторной.

Исходный код

main.cpp

```
#include <iomanip>
#include <iostream>
#include <string>
#include <cassert>
#include <pthread.h>
#include <algorithm>
#include <vector>

int partition(std::vector<std::string>& values, int left, int right) {
    std::string pivotValue = values[left];
    int leftWall = left;

    for (int i = left+1; i <= right; i++) {
        if (values[i] <= pivotValue) {
            std::swap(values[i], values[leftWall]);
            leftWall += 1;
        }
    }
    std::swap(values[leftWall], pivotValue);

    return leftWall;
}

void print_vector(std::vector<std::string>& values) {
    for (int i = 0 ; i < values.size(); i++) {
        std::cout << values[i] << "\n";
    }
}

void quicksort(std::vector<std::string>& values, int left, int right) {
     if (right > left) {
        int pivot = partition(values, left, right);
        quicksort(values, left, pivot-1);
        quicksort(values, pivot+1, right);
     }
}

struct qsort_starter {
    std::vector<std::string>& values;
```

```
    int left;
    int right;
    int depth;
};

void parallel_quicksort(std::vector<std::string>& values, int left, int right, int depth

void* quicksort_thread(void *init) {
    qsort_starter *start = (qsort_starter*)init;
    parallel_quicksort(start->values, start->left, start->right, start->depth);
    return NULL;
}

void parallel_quicksort(std::vector<std::string>& values, int left, int right, int depth
    if (right > left) {
        int pivot = partition(values, left, right);

        if (depth-- > 0){
            qsort_starter arg = {values, left, pivot-1, depth};
            pthread_t thread;
            int ret = pthread_create(&thread, NULL, quicksort_thread, &arg);
            assert((ret == 0) && "Thread creation failed");

            parallel_quicksort(values, pivot+1, right, depth);

            pthread_join(thread, NULL);
        }
        else {
            quicksort(values, left, pivot-1);
            quicksort(values, pivot+1, right);
        }
    }
}


int main(int argc, char **argv) {
    std::ios::sync_with_stdio(false);
    if (argc < 3) {
        std::cout << "Usage: ./main <depth> <size> < filename\n";
        return 0;
    }
```

```
    int depth = 0;
    if (argc > 1) {
        depth = strtol(argv[1], NULL, 10);
    }

    int size = 100;
    if (argc > 2) {
        size = strtol(argv[2], NULL, 10);
    }

    std::vector<std::string> values(size);

    for (int i = 0; i < size; ++i) {
        std::string str;
        std::cin >> values[i];
    }

    //print_vector(values);

    parallel_quicksort(values, 0, size-1, depth);

    std::cout << "\n";
    print_vector(values);

    return 0;
}
```