Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Лабораторная работа №2 по курсу
«Операционные системы»

Студент: Калинин Иван Николаевич
Группа: М8О-209Б-23
Вариант: 1
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2024
Содержание

1. Репозиторий
2. Постановка задачи
3. Общие сведения о программе
4. Общий метод и алгоритм решения
5. Исходный код
6. Демонстрация работы программы
7. Выводы

Репозиторий

https://github.com/wistfullly/OS_MAI-labs/tree/main/2_lab_os

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 обработке использовать стандартные средства создания потоков операционной системы (Windows/Unix). Ограничение потоков должно быть задано ключом запуска программы.

Необходимо уметь продемонстрировать количество потоков, используемых программой, с помощью стандартных средств операционной системы.

Привести исследование зависимости ускорения и эффективности алгоритма от входящих данных и количества потоков. Объяснить получившиеся результаты.

Вариант 1: Отсортировать массив целых чисел при помощи битонической сортировки

Общие сведения о программе

Программа написана на языке Си в UNIX-подобной операционной системе (Ubuntu). Для компиляции программы требуется указать ключ -fsanitize=thread. Для запуска программы в качестве 1 аргумента командной строки необходимо указать количество элементов в массиве, в качестве 2 аргумента - количество потоков.

Общий метод и алгоритм решения

Дается массив из n элементов и m потоков. Цель программы — отсортировать массив, используя метод битонной сортировки в многопоточном режиме. Каждый поток обрабатывает часть массива в соответствии с текущим этапом сортировки. Массив разбивается на фрагменты, которые обрабатываются потоками. Количество одновременно работающих потоков имеет значение m, заданое ранее. На каждом этапе сортировки потока обрабатывается фиксированная подгруппа элементов, выполняющая либо восходящую, либо нисходящую сортировку в зависимости от текущих этапов битонического слияния. Для корректной

работы всех потоков на каждом этапе сортировки используются барьеры синхронизации. Это позволяет завершить свою операцию перед переходом к следующему этапу алгоритма. После выполнения всех этапов битонной сортировки главный поток собирает результат и выводит отсортированный массив.

Исходный код

main.c

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


typedef struct {
    int *array;
    int start;
    int length;
    int dir; //1-по возрастанию, 0-по убыванию
    int max_threads;
    pthread_mutex_t *mutex;
    int *current_threads;
} ThData;


void swap(int *x1, int *x2) {
    int x3 = *x1;
    *x1 = *x2;
    *x2 = x3;
}
```

```
void bitonic_merge(int *array, int start, int length, int dir) {
    if (length <= 1) return;
    int half = length / 2;

    for (int i = start; i < start + half; i++) {
        if ((dir == 1 && array[i] > array[i + half]) ||
            (dir == 0 && array[i] < array[i + half])) {
            swap(&array[i], &array[i + half]);
        }
    }

    bitonic_merge(array, start, half, dir);
    bitonic_merge(array, start + half, half, dir);
}


void *bitonic_sort(void *arr) {
    //sleep(1);
    ThData *data = (ThData*)arr;

    if (data->length <= 1) return NULL;

    int half = data->length / 2;
    pthread_t threads[2] = {0};
    ThData data_[2];

    for (int i = 0; i < 2; i++) {
        data_[i] = (ThData) {
```

```
            .array = data->array,
            .start = data->start + i * half,
            .length = half,
            .dir = (i == 0) ? 1 : 0,
            .max_threads = data->max_threads,
            .mutex = data->mutex,
            .current_threads = data->current_threads
        };

        pthread_mutex_lock(data->mutex);
        if (*data->current_threads < data->max_threads) {
            (*data->current_threads)++;
            pthread_mutex_unlock(data->mutex);

            pthread_create(&threads[i], NULL, bitonic_sort, &data_[i]);
        } else {
            pthread_mutex_unlock(data->mutex);
            bitonic_sort(&data_[i]);
        }
    }
    for (int i = 0; i < 2; i++) {
        if (threads[i]) {
            pthread_join(threads[i], NULL);

            pthread_mutex_lock(data->mutex);
            (*data->current_threads)--;
            pthread_mutex_unlock(data->mutex);
        }
    }
```

```
    bitonic_merge(data->array, data->start, data->length, data->dir);
    return NULL;
}

int main(int argc, char *argv[]) {
    if (argc != 3) {
        printf("Use: 'name' 'size' 'max_threads'\n");
        return 1;
    }
    int size_ = atoi(argv[1]);
    int max_threads = atoi(argv[2]);

    if (size_ <= 0 || max_threads <= 0) {
        printf("Error - size and max_threads must be positive\n");
        return 1;
    }

    int *array = malloc(size_ * sizeof(int));
    for (int i = 0; i < size_; i++) {
        array[i] = rand() % 100;
    }
    printf("Random array:\n");
    for (int i = 0; i < size_; i++) {
        printf("%d ", array[i]);
    }
    printf("\n");

    pthread_mutex_t mutex;
    pthread_mutex_init(&mutex, NULL);
    int current_threads = 0;
```

```
    ThData data = {
        .array = array,
        .start = 0,
        .length = size_,
        .dir = 1,
        .max_threads = max_threads,
        .mutex = &mutex,
        .current_threads = &current_threads
    };

    bitonic_sort(&data);

    printf("Sorted array:\n");
    for (int i = 0; i < size_; i++) {
        printf("%d ", array[i]);
    }
    printf("\n");

    free(array);
    return 0;
}
```

Демонстрация работы программы

```
/usr/bin/make -f /home/dmitrii/second_kurs/2_lab_os/Makefile –C /home/dmitrii/sec-
ond_kurs/2_lab_os all
make: Entering directory '/home/dmitrii/second_kurs/2_lab_os'
mkdir build clang -o build/main src/main.c -fsanitize=thread
./build/main 32 8
Random array: 83 86 77 15 93 35 86 92 49 21 62 27 90 59 63 26 40 26 72 36 11 68 67 29 82 30
62 23 67 35 29 2
Sorted array: 2 11 15 21 23 26 26 27 29 29 30 35 35 36 40 49 59 62 62 63 67 67 68 72 77 82 83
86 86 90 92 93
rm -r ./build
make: Leaving directory '/home/dmitrii/second_kurs/2_lab_os'
```

Выводы

Язык Си позволяет пользователю взаимодействовать с потоками операционной системы. Для этого на Unix-подобных системах требуется подключить библиотеку pthread.h.

Создание потоков происходит быстрее, чем создание процессов, а все потоки используют одну и ту же область данных. Поэтому многопоточность – один из способов ускорить обработку каких-либо данных: выполнение однотипных, не зависящих друг от друга задач, можно поручить отдельным потокам, которые будут работать параллельно.

Средствами языка Си можно совершать системные запросы на создание потока, ожидания завершения потока, а также использовать различные примитивы синхронизации.