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: Тарасов Егор Дмитриевич
Группа: М8О-209Б-23
Вариант: 5
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2024

Содержание

Репозиторий

https://github.com/EgorTarasov1/mai-os-labs

Постановка задачи

Цель работы
Приобретение практических навыков в:
· Управление потоками в ОС
· Обеспечение синхронизации между потоками

Задание

Составить программу на языке C++, 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 5: Отсортировать массив при помощи чётно-нечётной сортировки Бетчера

Общие сведения о программе

Программа написана на языке C++ на операционной системе Windows 11.

Для сборки и компиляции программы использовались CMake и g++ соответственно. Для запуска программы, находясь в папке build, требуется прописать ./bitonic_sort <N> (где N – максимальное допустимое количество одновременно используемых потоков) в терминал.

Общий метод и алгоритм решения

Максимальное количество потоков для сортировки подаётся пользователем на вход программы.

Основной алгоритм сортировки реализован в функции batcherSort. Это итеративный процесс чётно-нечётной сортировки Бетчера. На каждой

итерации выполняются две фазы сортировки: четная и нечетная. Для реализации многопоточности массив разделяется на примерно равные части, каждая из которых обрабатывается отдельным потоком.

Для каждой фазы сортировки (четной и нечетной) создаются потоки, которые выполняют сравнения и обмены элементов в своей части массива.

Каждый поток выполняет функцию sortThread, которая сравнивает пары элементов в назначенной ему части массива и меняет их местами, если они находятся в неправильном порядке. Четная фаза обрабатывает пары с четными индексами, нечетная - с нечетными.

После завершения работы всех потоков в каждой фазе программа проверяет, отсортирован ли массив полностью. Если нет, процесс повторяется.

Исходный код

main.cpp

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <random>
#include <windows.h>

const int MAX_THREADS = 64;

struct ThreadData {
    std::vector<int>* arr;
    int start;
    int end;
    bool even;
};

DWORD WINAPI sortThread(LPVOID param) {
    ThreadData* data = static_cast<ThreadData*>(param);
    std::vector<int>& arr = *(data->arr);
    int start = data->start;
    int end = data->end;
    bool even = data->even;

    for (int i = start; i < end; i += 2) {
        if (even && i + 1 < arr.size()) {
            if (arr[i] > arr[i + 1]) {
```

```
                std::swap(arr[i], arr[i + 1]);
            }
        } else if (!even && i + 1 < arr.size()) {
            if (arr[i] > arr[i + 1]) {
                std::swap(arr[i], arr[i + 1]);
            }
        }
    }
    return 0;
}

void batcherSort(std::vector<int>& arr, int numThreads) {
    int n = arr.size();
    bool sorted = false;

    std::vector<HANDLE> threads(numThreads);
    std::vector<ThreadData> threadData(numThreads);

    while (!sorted) {
        sorted = true;

        for (int phase = 0; phase < 2; ++phase) {
            int elementsPerThread = (n + numThreads - 1) / numThreads;

            for (int i = 0; i < numThreads; ++i) {
                threadData[i].arr = &arr;
                threadData[i].start = i * elementsPerThread + phase;
                threadData[i].end = std::min((i + 1) * elementsPerThread, n);
                threadData[i].even = (phase == 0);

                threads[i] = CreateThread(NULL, 0, sortThread, &threadData[i], 0,
NULL);
            }

            WaitForMultipleObjects(numThreads, threads.data(), TRUE, INFINITE);

            for (int i = 0; i < numThreads; ++i) {
                CloseHandle(threads[i]);
            }
        }

        for (int i = 0; i < n - 1; ++i) {
            if (arr[i] > arr[i + 1]) {
                sorted = false;
                break;
            }
        }
    }
}
```

```
void printArray(const std::vector<int>& arr) {
    for (int num : arr) {
        std::cout << num << " ";
    }
    std::cout << std::endl;
}

int main(int argc, char* argv[]) {
    if (argc != 2) {
        std::cerr << "Usage: " << argv[0] << " <num_threads>" << std::endl;
        return 1;
    }

    int numThreads = std::atoi(argv[1]);
    if (numThreads <= 0 || numThreads > MAX_THREADS) {
        std::cerr << "Number of threads should be between 1 and " << MAX_THREADS
<< std::endl;
        return 1;
    }

    const int arraySize = 20;
    std::vector<int> arr(arraySize);

    std::random_device rd;
    std::mt19937 gen(rd());
    std::uniform_int_distribution<> dis(1, 100);

    for (int& num : arr) {
        num = dis(gen);
    }

    std::cout << "Original array:" << std::endl;
    printArray(arr);

    batcherSort(arr, numThreads);

    std::cout << "Sorted array:" << std::endl;
    printArray(arr);

    return 0;
}
```

Демонстрация работы программы

PS C:\Users\Xiaomi\Desktop\ll2\build> ./bitonic_sort 4

Original array:

19 75 44 1 25 62 50 47 41 98 17 20 55 44 31 97 44 37 74 61

Sorted array:

1 17 19 20 25 31 37 41 44 44 44 47 50 55 61 62 74 75 97 98

Выводы

Язык Си позволяет пользователю взаимодействовать с потоками операционной системы. Для этого при работе на системе Windows требуется подключить библиотеку <windows.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 C++ можно совершать системные запросы на создание потока, ожидания завершения потока, а также использовать различные примитивы синхронизации.