МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 "Компьютерные науки и прикладная математика"
Кафедра 806 "Вычислительная математика и программирование"

Лабораторная работа №2

По курсу «Операционные системы»

Студент: Степанов Н.Е.
Группа: М8О-208Б-23
Вариант: 4
Преподаватель: Миронов Е. С.

Дата: _____

Оценка: _____

Подпись: _____

Москва, 2024

Содержание

Репозиторий

https://github.com/n0w3e/os_labs/tree/lab2

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 максимального количества потоков, работающих в один момент
времени, 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 входных
данных и количества потоков. Получившиеся результаты необходимо объяснить.

Вариант задания:

4. Отсортировать массив целых чисел при помощи TimSort

Общие сведения о программе

Программа реализует алгоритм сортировки Timsort, который сочетает сортировку вставками и слиянием. Она поддерживает как однопоточную, так и многопоточную версии сортировки. В многопоточной версии массив разбивается на части, каждая из которых сортируется в отдельном потоке, после чего части объединяются. Программа также включает функцию для вывода массива и демонстрацию работы с пользовательским вводом.

Общий метод и алгоритм решения

Реализовать алгоритм сортировки Timsort, который сочетает сортировку вставками и слиянием. Предусмотреть как однопоточную, так и многопоточную версии алгоритма. Предоставить возможность пользователю вводить размер массива и количество потоков для сортировки.

Базовая сортировка Timsort:

Разделить массив на подмассивы фиксированного размера (RUN = 32). Отсортировать каждый подмассив с помощью сортировки вставками. Последовательно слить подмассивы, увеличивая размер слияния в два раза на каждом шаге.

Многопоточная сортировка:

Разделить массив на части, количество которых равно числу потоков. Каждый поток сортирует свою часть массива с использованием базовой сортировки Timsort. После завершения работы всех потоков объединить отсортированные части массива в один с помощью функции слияния.

Слияние отсортированных частей:

Создать временный массив для хранения результата. Использовать индексы для отслеживания текущей позиции в каждой части. Выбирать наименьший элемент из всех частей и помещать его во временный массив. Копировать результат обратно в исходный массив.

Исходный код

timsort.h:

```
#ifndef TIMSORT_H

#define TIMSORT_H


#include <cstddef>


void TimSort(int* array, size_t size);

void MultithreadedTimsort(int* array, size_t size, int num_threads);

void MergeSortedChunks(int* array, size_t size, size_t chunk_size, int num_chunks);

void TimsortWrapper(int* array, size_t size, int num_threads);


#endif
```

timsort.cpp:

```
#include "../include/timsort.h"

#include <pthread.h>
```

```
#include <vector>
#include <cstring>
#include <algorithm>
#include <climits>

constexpr size_t RUN = 32;

struct ThreadArgs {
    int* array;
    size_t start;
    size_t end;
};

void InsertionSort(int* array, size_t left, size_t right) {
    for (size_t i = left + 1; i <= right; ++i) {
        int key = array[i];
        size_t j = i;
        while (j > left && array[j - 1] > key) {
            array[j] = array[j - 1];
            --j;
        }
        array[j] = key;
    }
}



void Merge(int* array, size_t left, size_t mid, size_t right) {
    size_t len1 = mid - left + 1;
    size_t len2 = right - mid;
```

```
    std::vector<int> left_part(len1);
    std::vector<int> right_part(len2);

    std::memcpy(left_part.data(), &array[left], len1 * sizeof(int));
    std::memcpy(right_part.data(), &array[mid + 1], len2 * sizeof(int));

    size_t i = 0, j = 0, k = left;

    while (i < len1 && j < len2) {
        if (left_part[i] <= right_part[j]) {
            array[k++] = left_part[i++];
        } else {
            array[k++] = right_part[j++];
        }
    }

    while (i < len1) {
        array[k++] = left_part[i++];
    }

    while (j < len2) {
        array[k++] = right_part[j++];
    }
}

void TimSort(int* array, size_t size) {
    for (size_t i = 0; i < size; i += RUN) {
        size_t right = std::min(i + RUN - 1, size - 1);
```

```
        InsertionSort(array, i, right);
    }

    for (size_t run_size = RUN; run_size < size; run_size *= 2) {
        for (size_t left = 0; left < size; left += 2 * run_size) {
            size_t mid = left + run_size - 1;
            size_t right = std::min(left + 2 * run_size - 1, size - 1);

            if (mid < right) {
                Merge(array, left, mid, right);
            }
        }
    }
}

void* TimsortThread(void* args) {
    ThreadArgs* threadArgs = static_cast<ThreadArgs*>(args);
    size_t segment_size = threadArgs->end - threadArgs->start;

    TimSort(threadArgs->array + threadArgs->start, segment_size);

    pthread_exit(nullptr);
    return nullptr;
}

void MultithreadedTimsort(int* array, size_t size, int num_threads) {
    size_t chunk_size = size / num_threads;
    pthread_t threads[num_threads];
    ThreadArgs threadArgs[num_threads];
```

```
    for (int i = 0; i < num_threads; ++i) {
        threadArgs[i] = {array, i * chunk_size, (i == num_threads - 1) ? size : (i + 1) * chunk_size};
        pthread_create(&threads[i], nullptr, TimsortThread, &threadArgs[i]);
    }

    for (int i = 0; i < num_threads; ++i) {
        pthread_join(threads[i], nullptr);
    }

    MergeSortedChunks(array, size, chunk_size, num_threads);
}

void MergeSortedChunks(int* array, size_t size, size_t chunk_size, int num_chunks) {
    std::vector<int> temp(size);
    std::vector<size_t> indices(num_chunks, 0);

    for (size_t i = 0; i < size; ++i) {
        int min_index = -1;
        int min_value = INT_MAX;

        for (int chunk = 0; chunk < num_chunks; ++chunk) {
            size_t chunk_start = chunk * chunk_size;
            size_t chunk_end = std::min(chunk_start + chunk_size, size);

            if (indices[chunk] < chunk_end - chunk_start) {
                int value = array[chunk_start + indices[chunk]];
                if (value < min_value) {
                    min_value = value;
```

```
                min_index = chunk;
            }
        }
    }

    temp[i] = min_value;
    ++indices[min_index];
  }

  std::copy(temp.begin(), temp.end(), array);
}


void TimsortWrapper(int* array, size_t size, int /*num_threads*/) {
  TimSort(array, size);
}
```

main.cpp:

```
#include "timsort.h"
#include <iostream>
#include <vector>
#include <cstdlib>
#include <ctime>

void PrintArray(const int* array, size_t size) {
  for (size_t i = 0; i < size; ++i) {
    std::cout << array[i] << " ";
  }
  std::cout << std::endl;
```

```
}

int main() {
    size_t size;
    int num_threads;

    std::cout << "Enter the size of the array: ";
    std::cin >> size;
    std::cout << "Enter the number of threads: ";
    std::cin >> num_threads;

    std::vector<int> array(size);
    std::srand(static_cast<unsigned>(std::time(nullptr)));

    for (size_t i = 0; i < size; ++i) {
        array[i] = std::rand() % 100;
    }

    std::cout << "Original array: ";
    PrintArray(array.data(), size);

    MultithreadedTimsort(array.data(), size, num_threads);

    std::cout << "Sorted array: ";
    PrintArray(array.data(), size);

    return 0;
}
```

Демонстрация работы программы

n0wee@DESKTOP-8QSPN1P:~/Coding/os_labs/build/lab2$ ./timsort

n0wee@DESKTOP-8QSPN1P:~/Coding/os_labs/build/lab2$ ./lab2

Enter the size of the array: 10

Enter the number of threads: 2

Original array: 57 13 58 58 2 68 78 64 26 36

Sorted array: 2 13 26 36 57 58 58 64 68 78

Вывод

В ходе выполнения лабораторной работы я изучил и реализовал алгоритм сортировки Timsort, который сочетает сортировку вставками и слиянием. Этот алгоритм оказался эффективным как для небольших, так и для больших массивов данных. Я научился разбивать задачу на подзадачи и обрабатывать их параллельно с использованием многопоточности. Реализация многопоточной версии сортировки позволила мне глубже понять, как работают потоки и как их можно использовать для ускорения выполнения задач. В целом, работа над данной лабораторной помогла мне улучшить навыки программирования, а также понять принципы построения эффективных алгоритмов сортировки.