Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №2 по курсу**

**«Операционные системы»**

**Управление потоками в ОС**

Студент: Салихов Тимур Русланович

Группа: М8О–212Б–22

Вариант: 4

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2023.

**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 4: Отсортировать массив целых чисел при помощи TimSort.

**Общие сведения о программе**

Программа компилируется из файла sort.cpp. В программе используются следующие системные вызовы:

1. **pthread\_create** - создает новый поток
2. **pthread\_join** - присоединяется к потоку
3. **pthread\_exit** - завершает работу потока с указанным кодом

Программа реализует сортировку TimSort с шагом 4 элемента. Массив разделяется на подмассивы длины 4, которые сортируются при помощи insertion sort. Далее применяется алгоритм merge с шагом, увеличивающимся в 2 раза на каждом проходе. Сортировка является параллельной, то есть insertion\_sort и merge применяются одновременно на разных потоках. Поскольку работа происходит с разными частями массива, мьютекс не требуется.

**Общий метод и алгоритм решения**.

Для реализации задачи необходимо:

1. Написать insertion\_sort
2. Написать merge
3. Объединить их в один TimSort
4. Читать ключ, задающий количество потоков
5. Модифицировать TimSort для работы с многопоточностью
6. Произвести анализ зависимости времени работы от количества потоков.

**Основные файлы программы**

**sort.cpp:**

#include <pthread.h>

#include <chrono>

#include <iostream>

typedef struct args\_ins {

int\* array;

int start;

int end;

} args\_ins\_t;

typedef struct args\_merge {

int\* array;

int start\_l;

int start\_r;

int end;

} args\_merge\_t;

void\* insertion\_sort(void\* input) {

int\* array = ((args\_ins\_t\*)input)->array;

int start = ((args\_ins\_t\*)input)->start;

int end = ((args\_ins\_t\*)input)->end;

int current, temp, pos;

for (size\_t i = start + 1; i < end; ++i) {

current = array[i];

pos = i - 1;

while (pos >= start && array[pos] > array[pos + 1]) {

temp = array[pos];

array[pos] = current;

array[pos + 1] = temp;

--pos;

}

}

pthread\_exit(0);

}

void\* merge(void\* input) {

int\* array = ((args\_merge\_t\*)input)->array;

int start\_l = ((args\_merge\_t\*)input)->start\_l;

int start\_r = ((args\_merge\_t\*)input)->start\_r;

int end = ((args\_merge\_t\*)input)->end;

int left = start\_l;

int right = start\_r;

int temp[end - start\_l];

size\_t i = 0;

while (left != start\_r && right != end) {

if (array[left] <= array[right]) {

temp[i++] = array[left++];

} else {

temp[i++] = array[right++];

}

}

if (right == end) {

while (i != end - start\_l) {

temp[i++] = array[left++];

}

} else {

while (i != end - start\_l) {

temp[i++] = array[right++];

}

}

for (size\_t i = start\_l; i < end; ++i) {

array[i] = temp[i - start\_l];

}

pthread\_exit(0);

}

void TimSort(int\* array, size\_t size, int threads) {

size\_t run = 4;

pthread\_t tid[threads];

for (size\_t i = 0; i < size; i += run) {

int created = 0;

while (created != threads && i < size) {

args\_ins\_t\* data = (args\_ins\_t\*)malloc(sizeof(args\_ins\_t));

data->array = array;

data->start = i;

data->end = std::min(i + run, size);

pthread\_create(&tid[created], NULL, insertion\_sort, data);

++created;

i += run;

}

i -= run;

for (size\_t i = 0; i < threads; ++i) {

pthread\_join(tid[i], NULL);

}

}

for (size\_t mergeSize = run; mergeSize < size; mergeSize \*= 2) {

for (size\_t start\_l = 0; start\_l < size; start\_l += 2 \* mergeSize) {

int created = 0;

while (created != threads && start\_l < size) {

int start\_r = start\_l + mergeSize;

int end = std::min(start\_l + 2 \* mergeSize, size);

if (end > start\_r) {

args\_merge\_t\* data = (args\_merge\_t\*)malloc(sizeof(args\_merge\_t));

data->array = array;

data->start\_l = start\_l;

data->start\_r = start\_r;

data->end = end;

pthread\_create(&tid[created], NULL, merge, data);

}

created++;

start\_l += 2 \* mergeSize;

}

start\_l -= 2 \* mergeSize;

for (size\_t i = 0; i < threads; ++i) {

pthread\_join(tid[i], NULL);

}

}

}

}

void printArr(int\* a, size\_t size) {

std::cout << "[ ";

for (size\_t i = 0; i != size - 1; ++i) {

std::cout << a[i] << ", ";

}

std::cout << a[size - 1] << " ]\n";

}

int main(int argc, char\* argv[]) {

if (argc != 2) {

std::cerr << "Something is wrong with the arguments\n";

return 1;

}

int threads = atoi(argv[1]);

int size;

std::cout << "What size array do you what to sort?: ";

std::cin >> size;

int arr[size];

for (size\_t i = 0; i != size; ++i) {

arr[i] = rand() % 100;

}

printArr(arr, size);

auto start = std::chrono::high\_resolution\_clock::now();

TimSort(arr, size, threads);

auto end = std::chrono::high\_resolution\_clock::now();

auto duration = std::chrono::duration\_cast<std::chrono::microseconds>(end - start);

printArr(arr, size);

std::cout << "The sort was completed in " << duration.count() << " microseconds" << std::endl;

}

**Пример работы**

timursalihov@MacBook-Pro-Timur-2 ~/O/L/build (measurements)>

/Users/timursalihov/OperatingSystems/LW2/build/sort 1

What size array do you want to sort?: 100

[ 7, 49, 73, 58, 30, 72, 44, 78, 23, 9, 40, 65, 92, 42, 87, 3, 27, 29, 40, 12, 3, 69, 9, 57, 60, 33, 99, 78, 16, 35, 97, 26, 12, 67, 10, 33, 79, 49, 79, 21, 67, 72, 93, 36, 85, 45, 28, 91, 94, 57, 1, 53, 8, 44, 68, 90, 24, 96, 30, 3, 22, 66, 49, 24, 1, 53, 77, 8, 28, 33, 98, 81, 35, 13, 65, 14, 63, 36, 25, 69, 15, 94, 29, 1, 17, 95, 5, 4, 51, 98, 88, 23, 5, 82, 52, 66, 16, 37, 38, 44 ]

[ 1, 1, 1, 3, 3, 3, 4, 5, 5, 7, 8, 8, 9, 9, 10, 12, 12, 13, 14, 15, 16, 16, 17, 21, 22, 23, 23, 24, 24, 25, 26, 27, 28, 28, 29, 29, 30, 30, 33, 33, 33, 35, 35, 36, 36, 37, 38, 40, 40, 42, 44, 44, 44, 45, 49, 49, 49, 51, 52, 53, 53, 57, 57, 58, 60, 63, 65, 65, 66, 66, 67, 67, 68, 69, 69, 72, 72, 73, 77, 78, 78, 79, 79, 81, 82, 85, 87, 88, 90, 91, 92, 93, 94, 94, 95, 96, 97, 98, 98, 99 ]

The sort was completed in 2162 microseconds

timursalihov@MacBook-Pro-Timur-2 ~/O/L/build (measurements)>

/Users/timursalihov/OperatingSystems/LW2/build/sort 2

What size array do you want to sort?: 100

[ 7, 49, 73, 58, 30, 72, 44, 78, 23, 9, 40, 65, 92, 42, 87, 3, 27, 29, 40, 12, 3, 69, 9, 57, 60, 33, 99, 78, 16, 35, 97, 26, 12, 67, 10, 33, 79, 49, 79, 21, 67, 72, 93, 36, 85, 45, 28, 91, 94, 57, 1, 53, 8, 44, 68, 90, 24, 96, 30, 3, 22, 66, 49, 24, 1, 53, 77, 8, 28, 33, 98, 81, 35, 13, 65, 14, 63, 36, 25, 69, 15, 94, 29, 1, 17, 95, 5, 4, 51, 98, 88, 23, 5, 82, 52, 66, 16, 37, 38, 44 ]

[ 1, 1, 1, 3, 3, 3, 4, 5, 5, 7, 8, 8, 9, 9, 10, 12, 12, 13, 14, 15, 16, 16, 17, 21, 22, 23, 23, 24, 24, 25, 26, 27, 28, 28, 29, 29, 30, 30, 33, 33, 33, 35, 35, 36, 36, 37, 38, 40, 40, 42, 44, 44, 44, 45, 49, 49, 49, 51, 52, 53, 53, 57, 57, 58, 60, 63, 65, 65, 66, 66, 67, 67, 68, 69, 69, 72, 72, 73, 77, 78, 78, 79, 79, 81, 82, 85, 87, 88, 90, 91, 92, 93, 94, 94, 95, 96, 97, 98, 98, 99 ]

The sort was completed in 1876 microseconds

timursalihov@MacBook-Pro-Timur-2 ~/O/L/build (measurements)>

/Users/timursalihov/OperatingSystems/LW2/build/sort 4

What size array do you want to sort?: 100

[ 7, 49, 73, 58, 30, 72, 44, 78, 23, 9, 40, 65, 92, 42, 87, 3, 27, 29, 40, 12, 3, 69, 9, 57, 60, 33, 99, 78, 16, 35, 97, 26, 12, 67, 10, 33, 79, 49, 79, 21, 67, 72, 93, 36, 85, 45, 28, 91, 94, 57, 1, 53, 8, 44, 68, 90, 24, 96, 30, 3, 22, 66, 49, 24, 1, 53, 77, 8, 28, 33, 98, 81, 35, 13, 65, 14, 63, 36, 25, 69, 15, 94, 29, 1, 17, 95, 5, 4, 51, 98, 88, 23, 5, 82, 52, 66, 16, 37, 38, 44 ]

[ 1, 1, 1, 3, 3, 3, 4, 5, 5, 7, 8, 8, 9, 9, 10, 12, 12, 13, 14, 15, 16, 16, 17, 21, 22, 23, 23, 24, 24, 25, 26, 27, 28, 28, 29, 29, 30, 30, 33, 33, 33, 35, 35, 36, 36, 37, 38, 40, 40, 42, 44, 44, 44, 45, 49, 49, 49, 51, 52, 53, 53, 57, 57, 58, 60, 63, 65, 65, 66, 66, 67, 67, 68, 69, 69, 72, 72, 73, 77, 78, 78, 79, 79, 81, 82, 85, 87, 88, 90, 91, 92, 93, 94, 94, 95, 96, 97, 98, 98, 99 ]

The sort was completed in 1201 microseconds

**Таблица зависимости времени работы от количества потоков:**

|  |  |
| --- | --- |
| **Количество потоков** | **Время работы, микросекунды** |
| **1** | **13 774** |
| **2** | **10 850** |
| **3** | **11 640** |
| **4** | **10 499** |
| **5** | **11 940** |
| **6** | **9 671** |
| **7** | **15 058** |

**Вывод**

В процессе работы я изучил работу с многопоточностью в ОС. Одной из моих первоначальных ошибок было то, что я забыл добавить pthread\_join из-за чего потоки не успевали завершиться и в результате массив портился. Также нетривиальной задачей оказалось переписывание сортировки с изменяемым числом потоков, так как пришлось менять внутренний цикл для распределения задач по доступному количеству потоков. В процессе замеров времени выполнения я выяснил, что основной прирост в производительности происходит при использовании 2 потоков. Дальнейшее увеличение числа потоков не дает серьезного прироста в производительности, а после 7 потоков, производительность начинает падать.